On deep learning in physics

by

Kyle Mills

A thesis submitted to the

School of Graduate and Postdoctoral Studies in partial

fulfillment of the requirements for the degree of

Doctor of Philosophy in Modelling and Computational Sciences.

Faculty of Science

The University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

April 2021

THESIS EXAMINATION INFORMATION

Submitted by: **Kyle Mills**

Doctor of Philosophy in Modelling and Computational Sciences.

Thesis title: On deep learning in physics

An oral defense of this thesis took place on March 31, 2021 in front of the following examining committee:

Examining committee:

Chair of Examining Committee	FRANCO GASPARI
Research Supervisor	ISAAC TAMBLYN
Graduate Program Representative	LENNAERT VAN VEEN
Examining Committee Member	HENDRICK DE HAAN
Examining Committee Member	FAISAL QURESHI
University Examiner	GREG LEWIS
External Examiner	GIUSEPPE CARLEO

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Abstract

Machine learning, and most notably deep neural networks, have seen unprecedented success in recent years due to their ability to learn complex nonlinear mappings by ingesting large amounts of data through the process of training. This learning-by-example approach has slowly made its way into the physical sciences in recent years. In this dissertation I present a collection of contributions at the intersection of the fields of physics and deep learning. These contributions constitute some of the earlier introductions of deep learning to the physical sciences, and comprises a range of machine learning techniques, such as feed forward neural networks, generative models, and reinforcement learning. A focus will be placed on the lessons and techniques learned along the way that would influence future research projects.

Keywords: machine learning; reinforcement learning; materials science; deep neural networks; deep learning

Author's Declaration

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I authorize the University of Ontario Institute of Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ontario Institute of Technology to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

Kyle Mills

Statement of Contributions

Some work in this thesis was completed as part of coauthored works. Only primary-author works are included verbatim as part of the main body of this dissertation. Secondary-author works are significantly abridged to reflect my contribution to the work while still emphasizing the contribution of the work to this dissertation. Secondary-author works that are not of direct significance to this dissertation exist, but have been omitted entirely.

Section 3.1 is from K. Mills and I. Tamblyn, "Deep neural networks for direct, featureless learning through observation: The case of two-dimensional spin models," *Physical Review E*, vol. 97, no. 3, 2018. Isaac Tamblyn was involved in the ideation, authoring, and peer review process. I carried out all experiments and analysis.

Section 3.2 is from K. Mills, M. Spanner, and I. Tamblyn, "Deep learning and the Schrödinger equation," *Physical Review A*, vol. 96, no. 4, p. 042113, 2017. Michael Spanner provided example code for the numerical solution of the Schrödinger equation and was involved, along with Isaac Tamblyn, in the ideation of this work. Isaac Tamblyn wrote the first draft of the introduction, which was subsequently modified during revisions. All authors participated in revising the manuscript and participated in the peer review process.

STATEMENT OF CONTRIBUTIONS

Section 3.3 borrows text from K. Ryczko, K. Mills, I. Luchak *et al.*, "Convolutional neural networks for atomistic systems," *Computational Materials Science*, vol. 149, pp. 134–142, 2018. I contributed to this work, but did not serve as the primary author. As such, I have included only my direct contributions that are relevant to this dissertation.

Section 3.4 is from K. Mills, K. Ryczko, I. Luchak *et al.*, "Extensive deep neural networks for transferring small scale learning to large scale systems," *Chemical Science*, vol. 10, no. 15, pp. 4129–4140, 2019. Kevin Ryczko generated and assisted in generating the hexagonal sheet and porous graphene data sets, respectively. Additionally Kevin Ryczko ran the Monte Carlo simulation used to generate Figure 13, and developed an independent EDNN implementation to assist in the validation and debugging of the method. Iryna Luchak contributed extensively to the design of the methodology, and orchestrated experiments during the development of the method. The EDNN technique was mostly Isaac Tamblyn's idea. Kevin Ryczko, Iryna Luchak, Chris Beeler, and Isaac Tamblyn all participated in revisions of the manuscript and experimental discussions during the project. Adam Domurad wrote a very preliminary code implementation.

Section 3.5 is from preprint K. Mills and I. Tamblyn, "Weakly-supervised multi-class object localization using only object counts as labels," 2021. Isaac Tamblyn revised the manuscript and supervised the project. All work was

carried out by Kyle Mills.

Section 4.1 is from preprint K. Mills and I. Tamblyn, "Phase space sampling and operator confidence with generative adversarial networks," *arXiv*, 2017. Isaac Tamblyn supervised the project, contributed text to the introduction and was involved in discussions throughout.

Section 4.2 and Section 4.3 are from C. Casert, K. Mills, T. Vieijra *et al.*, "Optical lattice experiments at unobserved conditions and scales through generative adversarial deep learning," *arXiv*, pp. 1–11, 2020 and K. Mills, C. Casert, and I. Tamblyn, "Adversarial Generation of Mesoscale Surfaces from Small-Scale Chemical Motifs," *Journal of Physical Chemistry C*, vol. 124, no. 42, pp. 23 158–23 163, 2020, respectively. Corneel Casert developed the code framework and carried out preliminary tests (that would be used for Ref. [7] (Section A)), which Kyle Mills then adapted to work with porous graphene sheets (Section 4.3). Kyle Mills created the encoding, and modified the framework to work with hexagonal convolutions. Isaac Tamblyn was involved in discussion and the revision of the manuscripts.

Section 5.1 is from K. Mills, P. Ronagh, and I. Tamblyn, "Finding the ground state of spin Hamiltonians with reinforcement learning," *Nature Machine Intelligence*, vol. 2, no. 9, pp. 509–517, 2020. All authors contributed to the ideation and design of the research. Pooya Ronagh and Isaac Tamblyn jointly supervised this work and contributed to revisions of the manuscript.

Acknowledgments

This work encompasses research that was carried out over a number of years, at a number of institutions, with influence from so many individuals that it would be impossible to fairly list every one. With that being said, I would like to acknowledge groups of people that have been influential throughout this endeavor.

To my colleagues from the University of Ontario Institute of Technology: I appreciate our late-night assignment collaborations, the mid-day coffee breaks, and overall amusing office conversation. I couldn't have asked for a better group of individuals–friends–with whom to share the formative years of graduate school.

To the exceptional researchers and students of the computational chemistry group at the National Research Council: you are role models for how science should be conducted and your passion for your work is an inspiration. To my colleagues in CLEAN, including those who have come and gone: thank you for the discussions and distractions, and for making every day a rewarding experience.

To my coworkers at 1QBit: thank you for making me feel welcomed in Vancouver and showing me the best parts of your beautiful city. I am appreciative of the unique research-centred industry experience that you provide, and I am looking forward to working with you all in the future.

To my friends whose acquaintance was not through academia: thank you for the often much-needed distractions and fun times throughout. Thank you

to Andrea Pagotto for encouraging and motivating me in the final months.

Thank you to John Powell for the composition of my most-listened album, the “How to Train Your Dragon” soundtrack. While I was training neural networks and not dragons, the inspiration was transferable.

Thank you to my parents, Mark and Cecilia, who have supported me throughout. I would like to acknowledge the stories my mother told me as a child of the adventures of Gizmo, a boy quite like myself whose curiosity of the inner workings of his family’s appliances often left him with a nonfunctional pile of pieces, but who was always able to figure out how to reconstruct just in time. Combined with my father’s hands-on practical problem solving abilities, I’m sure this formed my combination of curiosity and practical approach that is so valuable in Science. A special thank you to my mother for proof reading many of the included manuscripts.

Finally, it is with deep gratitude that I’d like to thank my supervisor, Isaac Tamblyn for introducing me to the idea of grad school and for the events that transpired on Friday, July 17, 2015, known within the research group as “Crazy Tuesday” that lead to his decision to pursue research in the uncharted field of deep learning. I am exceedingly grateful for the more-rare-than-once-in-a-lifetime opportunity to work at the forefront of research in this emerging field. Isaac’s guidance, encouragement, advice, enthusiasm, and intensity made this a very rewarding and fun experience. It is also necessary to acknowledge the time and effort Isaac spent, and continues to spend, on the acquisition of funding, computational resources, and hardware that enabled this research, without which none of this would have been possible.

Contents

Chapter 1

Introduction

1.1 History

Within the past two decades, the fields of artificial intelligence (A.I.), computer vision, and natural language processing have advanced at unprecedented rates. Artificial intelligence is now commonplace; voice assistants such as Apple's Siri, Amazon Echo, and Google Assistant all use artificial intelligence for voice recognition and audio synthesis. Predictive mobile phone keyboards [10] predict the next word to be typed, and messaging applications suggest entire message responses. Driver-assisted vehicles capable of almost full autonomy no longer face technical, but rather regulatory hurdles. These are merely the consumer-facing technologies; behind-the-scenes A.I. applications in fraud detection, supply chain management, media recommendations, and advertising seem limitless.

Recently, deep neural networks (DNNs) have gained momentum as the architecture of choice for much of machine learning. Enthusiastic adoption of

DNNs by large tech companies such as Amazon and Google, and the use of graphical processing units (GPUs) as accelerators [11, 12] has lead to a competitive environment, encouraging hardware manufacturers such as NVIDIA to focus their efforts on technologies to accelerate the computations [13].

In 2013, deep neural networks were combined with the fields of optimal control and reinforcement learning when DeepMind designed an A.I. algorithm that learned and mastered seven classic Atari 2600 video games [14]. Importantly, this was accomplished through experience with the game environment, with no knowledge of the rules provided. Through exploration and experimentation, modern A.I. can learn complex control schemes.

In 2016, deep neural networks gained mainstream attention when AlphaGo, using a combination of deep neural networks and Monte Carlo tree-search, beat some of the best human Go players [15], a full decade before many in the field of artificial intelligence had anticipated [16]. Unlike DeepBlue's [17] chess victory two decades prior, due to the size of the game, optimal moves in the game of Go cannot be computed through brute force search alone [18]. In order to win, AlphaGo was trained by watching thousands of professional human matches. A year later, AlphaGoZero [19] was trained without any human input, learning its human-level intuition and strategy solely by playing against itself.

In the fields of physics and physical chemistry, the adoption has been more delayed, seemingly due to the black-box nature of A.I. algorithms. When a

fundamental aspect of these fields is exploring the question of *why* Nature acts in the way it does, to some, A.I. methods indeed produce unsatisfactory results due to their obfuscation of reasoning and lack of a physical explanation, even in spite of impressively accurate predictions. As such, scientific machine learning has seen a push for explainability and interpretability.

Nonetheless, many deep learning approaches have made their way into the physical sciences, and researchers are focusing on interpretable machine learning-based approaches to problems in the physical sciences. This dissertation is a collection of one researcher's contributions to the field. These contributions constitute some of the earlier introductions of deep learning to the physical sciences, and an important focus will be placed on the lessons and techniques learned along the way that would influence future research projects.

Techniques used in modern machine learning are inspired by principles brought forward by twentieth-century psychologists in their attempts to understand the neurological processes that occur in the brain. In 1949, Donald O. Hebb published his book *The Organization of Behaviour* [20], which biologically introduces one of the inspiring principles for modern neural networks. Commonly summarized as "cells that fire together, wire together", it summarizes the way in which neural networks (in the brain) learn; neurons that are repeatedly activated together have their connections (weights) strengthened during the learning process. This principle became inspiration for Rosenblatt

in 1958 when he presented his Perceptron [21], a linear classifier packaged with an *algorithm to learn from data* (prior to this in 1943, McCulloch and Pitts [22] proposed a very similar "artificial neuron", but it had to be hand-tuned to the problem and could not "learn" from data). Perceptrons were able to perform linear classification, however such an architecture was unable to represent even a simple nonlinear problem, such as representing the output of an XOR (exclusive or) gate. By combining single-unit perceptrons, grouping them into layers, and producing what is referred to as a multilayer perceptron (MLP) [23], more complicated functions could be represented, however it was still difficult to "train" these neurons for all but the most simple tasks, with learning algorithms suffering from exploding and vanishing gradients.

A breakthrough came when a method, now called backpropagation (Section 2.1.3), was proposed as a way to compute gradients of weights and propagate the error backward through layers in a network of perceptrons. Attribution of the development of backpropagation is a contentious issue in the field of deep learning. Several independent researchers developed the technique, but it is mostly accepted that Linnainmaa [24, 25] was the first to do so in his Master's thesis. He made no application to neural networks specifically, but presented the method (complete with FORTRAN code) required to apply the chain rule of calculus explicitly and efficiently to "arbitrary, discrete, possibly sparsely-connected, neural network-like networks" [26].

In 1981, the first application of backpropagation to neural networks was presented by Werbos [27], and gained mainstream attention in 1986 with an experimental study [28] demonstrating that the hidden layers of neural networks learn internal representations (features) when trained through backpropagation. This is perhaps the birth of deep, "featureless" machine learning, although multi-layer neural networks, even when using backpropagation, remained difficult to train. This was partially due to the limited computational resources available in the 1980s and 90s; backpropagation of errors through multiple layers is a computationally expensive task, especially for computers of the late twentieth century. Furthermore, it was identified [29] that the vanishing gradient problem (Section 2.1.4) was a major hindrance to the training of deep, multi-layer neural networks.

Around the turn of the century, improvements on gradient descent algorithms (Section 2.1.5), and clever "tricks of the trade", such as unsupervised pretraining, had made marginal gains. Impressive performance of convolutional neural networks (LeNet, [30]) foreshadowed a bright future for neural networks in image recognition, however large scale models and wide acceptance would take a decade more. Throughout the early 2000s, researchers were making progress with neural networks, but models were small, deep networks were unstable in training, and the ingestion of large amounts of data was still infeasible [31].

In 2009, Andrew Ng and colleagues proposed the use of Graphical Processing Units (GPUs) to accelerate the computations required for deep learning [31]. This was effectively the turning point for deep learning research; larger networks immediately became tractable, and large amounts of data could be used to efficiently train deep neural networks. Ng's GPU-based training algorithm claimed between one and two orders of magnitude speed up over CPU-based algorithms, and this would be improved in the near future with deep learning-specific hardware and libraries being released by hardware manufacturers. Soon, these larger networks trained on GPUs shattered records for handwriting recognition [32], and image classification [33]. The release of open source and well-documented software tools implementing automatic differentiation (automatic computation of gradients to perform backpropagation; Section 2.1.3) such as TensorFlow [34] and PyTorch [35] lowered the barrier to entry, enabling casual researchers to experiment with deep learning without significant investment into implementation. These flexible software packages, coupled with the acceleration capability of consumer-grade GPU hardware is what spurred the advent of the deep learning [36] revolution.

1.2 Deep learning in physics and materials science

Soon, the technologies developed for computer vision began to percolate into fields within the physical sciences. DNNs were shown to be able to differentiate between phases [37–39] and predict critical parameters [40] of condensed matter systems. They were capable of assisting in the analysis of particle accelerator experiments [41] and were being used in astronomy to classify galaxies [42, 43].

In materials science prior to machine learning, atomic interactions were handled through the approximation of the underlying interaction mechanisms (e.g. mean field, tight binding, Born-Oppenheimer molecular dynamics, etc.). Some of the earliest examples of what would technically be considered machine learning was the employment of "force-fields"–phenomenological fits to a limited number of either experimental observations or theoretical results [44–50]. Traditionally, these fits have been limited to a small number of parameters, but the recent adoption of machine learning techniques has lead to more complicated and accurate models. High-level *ab initio* calculations have been used to train artificial neural networks to fit high-dimensional interaction models [51–56], and to predict material properties [57, 58]. These feature-based

approaches [59–61] are quite powerful, but the hand-selected nature of the features is arguably a significant shortcoming. After all, featureless, or "representation learning" [62] has renewed interest in the fields of handwriting recognition and image classification, where the performance of the traditional hand-selected feature approach has stagnated [63].

Researchers have attempted to explain the success of deep learning based on principles borrowed from physics [64], and physicists realized that the principles that are important for modelling their systems-of-interest were not necessarily a concern of the computer scientists developing the deep learning architectures. For example, physical models should be exactly invariant to rotations, something convolutional deep neural networks can only traditionally approximate [43,65] and the antisymmetry of electrons in quantum mechanics remained a challenge until very recently [66].

1.3 Unique concerns of scientific ML

Commercial applications of machine learning tend to favour performance over all other aspects of the models. Scientists, on the other hand, often care deeply about what a model has learned and would like to glean insights from the model instead of merely making accurate predictions. This opens a divide between scientific machine learning and industrial machine learning. There are

several research directions that are attempting to address specifically the concerns of scientific machine learning. In 2019, the United States Department of Energy released a thorough report [67] outlining these research directions.

In scientific research, here are physical rules that scientists know–so called domain knowledge such as conservation of energy, relativity, and symmetries–that must be respected in any valid solution. It is unclear how to inform a neural network of this knowledge. This is the research direction of informed machine learning [68]. At this point, options for incorporating domain knowledge are limited; domain knowledge can be incorporated using hard or soft constraints during training (e.g. clipping, regularization in the loss function, etc.). Furthermore, the model itself can be devised in a way that incorporates knowledge about the system (e.g. convolutional neural networks for visual input).

Another research direction is that of interpretable and explainable machine learning [69]. When using a model, a scientist might like to know which features are most relevant to making a prediction. We will discuss in Section 2.1.7.2 how L1-regularization can be used for global interpretability, determining which features are not relevant, but another method is to look at which features in the input contribute most to the output for a given prediction (local interpretability). This is known as sensitivity analysis [70, 71]. Activation maximization [71, 72] can be used to get some limited insight into what each

filter in the neural network is learning although with abstract representations such as the ones found in scientific machine learning (e.g. electrostatic potentials, electron densities, spin configurations), these maps can be equally difficult to interpret. Since the physical sciences are often concerned with discovery, interpretable machine learning approaches that can be probed may aid in uncovering new intuition.

A measure of predictive confidence is important in scientific machine learning (arguably, also in commercial machine learning). ML is famously good at making predictions near the domain on which it was trained, but fails miserably, and more problematically fails silently, when asked to extrapolate past its training domain. Achieving a reliable measure of uncertainty in predictions is an open research question. One method is to use Bayesian Neural Networks, which predict distributions instead of making point estimates [73–77]. The learned variance of predictions can be used as a measure of uncertainty. Along a similar vein, Monte Carlo approaches can be used to gain a measure of variance. When making point predictions, additional inference passes can be made, using slight perturbations on the input to gain a measure of variance by exploring the local support neighbourhood [78]. In an alternative Monte Carlo method, dropout [79] (Section 2.1.7.2) can be used during training and inference [80]. During inference, multiple predictions are made using multiple dropout configurations (e.g. random nodes are disabled). The idea here is that

if there is little variation in these outputs, multiple sets of nodes have had the opportunity to co-adapt. This would likely only happen if the neural network has seen similar examples during training.

While deep learning has achieved great success, both in and out of the physical sciences, there are still many questions to ask and research problems to tackle before deep learning will revolutionize the physical sciences.

Chapter 2

Machine Learning Methods

2.1 Neural network basics

Supervised machine learning, in its most basic form, is the act of fitting the parameters of some parameterized function such that, given an input, the function reproduces a desired output.

The supervision comes in the form of labelled training data, that is, a set of data points $\mathbf{x}$, and corresponding labels y_{true}, that are (or at least we hope[1] are) related through a function f^*. That is,

$$f^*(\mathbf{x}) = y_{\text{true}}.$$

Through the process that we will call training, we wish to determine an approximation to the function f^* that reproduces the mapping for the values of $\mathbf{x}$ in our training set. We hope that our approximation f performs well on

[1] Some apparent relationships might not have any functional mapping due to an abundance of noise, or purely the absence of a relationship entirely. For example, one could construct a labelled data set mapping people's favourite colours to their ability to correctly predict a coin flip. There is no function that could be constructed to accurately predict such a mapping.

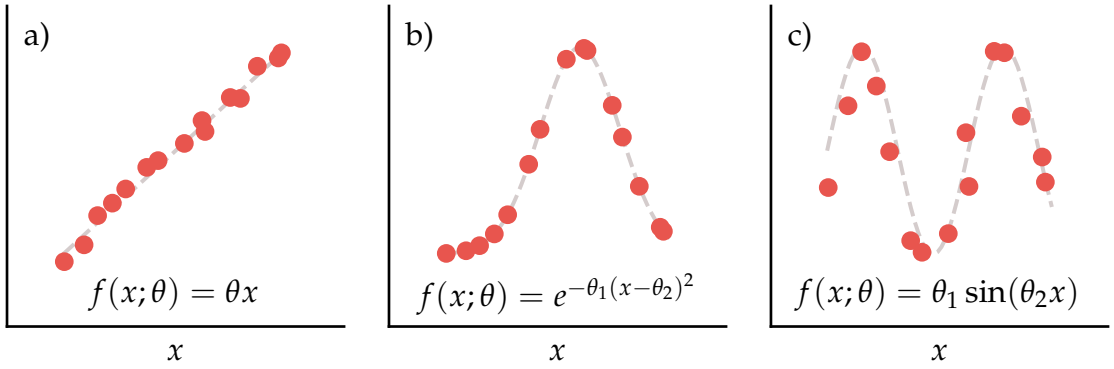


Figure 2.1: Three example parameterized functions. The red points represent some training data, and the dashed lines represent a good choice of fitting parameters θ to fit $f(x;\theta)$ to the training data. The parameters are the quantities that are learned during the training process.

the training data, but also that it captures the true relationship between $\mathbf{x}$ and y_{true} so that it generalizes to new data in the future. Figure 2.1 shows three examples that would be considered machine learning in the most basic sense. The task is to tweak the parameters θ_n (often called weights in machine learning literature, although there is a slight distinction to be made later), to best approximate the true function. These examples are incredibly simple, and near-optimal values of θ are trivially easy to compute using linear regression since we know the functional form of f^*. The more interesting applications of machine learning come when f^* is unknown, expensive to compute, analytically impossible to represent, or we do not know which variables in $\mathbf{x}$ influence f^*.

Let us consider an example where each data point $\mathbf{x}$ is an array of features

pertaining to homes that have recently been sold in a city: size, number of bedrooms and bathrooms, the presence of a pool, etc. We wish to devise a function to predict the sale price of new homes. In this case, we cannot write down a mathematical expression, in fact, f^* does not even exist in an analytical form due to the subjectivity of home sales. The best we can do in many cases is to approximate f^* accurately for most cases. We could devise a simple predictor of house price comprising a weighted sum, such as

$$f = \mathbf{W}^\top \mathbf{x},$$

where $\mathbf{W}$ is a column vector of weights for each feature present in $\mathbf{x}$. This can be drawn as a graph with nodes representing features and edges representing the weights, as shown in Figure 2.2a. This is likely to give a poor estimate because of the relationships between features (e.g. a pool in a hot climate is likely valued higher than a pool in a cold climate, and the ratio of bathrooms to bedrooms is important in addition to the absolute quantities).

One method to capture these relationships is to devise handcrafted features based on the raw features and known relationships. This works if the relationships are known, but there may be correlations present that we are unaware of. We can therefore come up with a more general predictive function by inserting an additional layer of complexity: an intermediate "hidden" calculation $\mathbf{z}$ com-

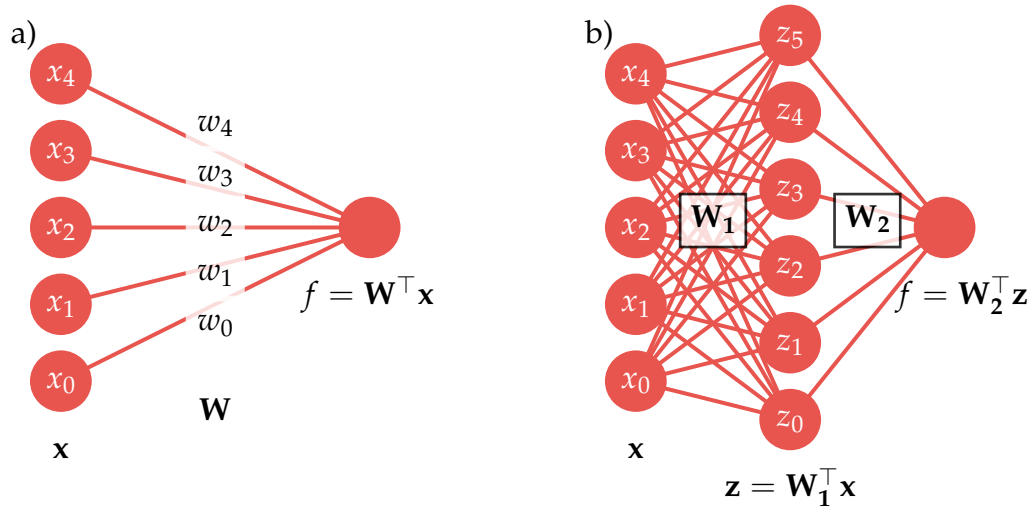


Figure 2.2: Two simple linear function approximators represented as graphs. a) A single-layer approximator is shown, which is effectively a weighted sum of the five input features. b) A two-layer approximator: by introducing a hidden layer with output $\mathbf{z}$, we allow the five features to mix, effectively resulting in higher-level features being constructed out of the original low-level features.

prised of correlations of features. Essentially, we create a number of weighted sums, that we then combine with one final weighted sum. Figure 2.2b shows a graphical representation of what this structure looks like. We call this intermediate computation a hidden layer, and it is calculated the same way as the output in the single-layer graph:

$$\mathbf{z} = \mathbf{W}_1^\top \mathbf{x}.$$

We now have two sets of weights, $\mathbf{W}_1$, now a matrix, denoting the weights of the first layer, and a matrix of weights $\mathbf{W}_2$ denoting the weights of the second

(output) layer. The output of the entire function is therefore

$$f = \mathbf{W}_2^\top \mathbf{z} = \mathbf{W}_2^\top (\mathbf{W}_1^\top \mathbf{x}),$$

and a problem becomes apparent. Arranging the computation in such a way has accomplished nothing; $\mathbf{W}_2^\top \mathbf{W}_1^\top$ is itself just a matrix, and therefore this two-layer setup is identical to the single-layer setup, just with different weights. In other words, a linear combination of linear combinations is itself just a different linear combination, and therefore we will never be able to represent a nonlinear function with such an approach. Furthermore, our model has a fixed intercept; that is, an input of zero will yield an output of zero and we can only learn the slope by modifying the weights.

A simple solution to the first limitation is to introduce a nonlinearity in the hidden layer, that is, pass the output of the hidden layer $\mathbf{z}$ through a nonlinear function σ before entering the output layer. The second limitation is easily addressed by introducing a zeroth-order term to each layer. We call these intercept parameters biases and they are learned in a similar style as the weights. Then the output of our predictor function is, in entirety

$$f(\mathbf{x}; \theta) = \mathbf{W}_2^\top \sigma(\mathbf{W}_1^\top \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2. \tag{2.1}$$

The nonlinearity function σ is more commonly referred to as an **activation**

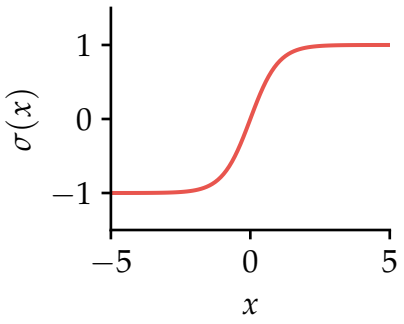


Figure 2.3: The hyperbolic tangent function, $\tanh(x)$ commonly used as a nonlinearity (activation function) in neural networks.

function due to how it "activates" the nonlinear nature of a node in the network. For now, we will consider the activation function to be a hyperbolic tangent function (Figure 2.3). There are several other choices for activation functions that we will discuss in Section 2.1.4.

This basic structure, repeated computational layers of *"weights* $\times$ *input* + *bias, fed into nonlinearity"* is what we call a neural network. When all output nodes of one layer are connected to all input nodes in the next layer, as in the example presented in Figure 2.2b and discussed here, we call this a fully-connected network, or sometimes, a dense network.

At this point, I will now formally make the distinction between weights and parameters: θ is a set of all learnable parameters. So far, we have only introduced weights and biases but additional learnable quantities will be considered later. Thus θ is used as notation in the general case where the specifics of the parameter's function is not important.

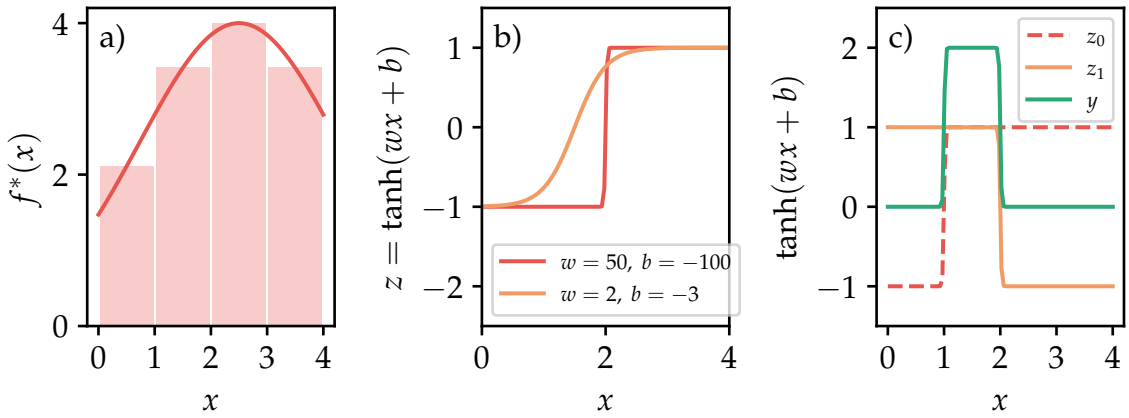


Figure 2.4: We wish to design a neural network to approximate the solid line function in a). We will do this by approximating this function as discrete boxes as shown. b) The output of a neural network node z is dependent upon its weight and bias. In this case, $z = \tanh(wx + b)$. Weights are used to "sharpen" the function and biases are used to shift the function left and right. c) A rectangular step function can be constructed by adding the two outputs, e.g. $y = z0 + z1 = \tanh(50x - 50) + \tanh(-50x + 100)$.

2.1.1 Universal approximation

The universal approximation theorem is commonly cited as being the motivation for using neural networks, that is, there exists a neural network comprising a single hidden layer can be made to approximate any desired function to arbitrary accuracy and arbitrary domain. The caveat of this is the number of neurons in this layer must increase as the desired accuracy and domain increases. This makes it a not-so-useful justification in practice, but it is still instructive to demonstrate how such universality can be accomplished.

Let us manually design a neural network that approximates the function in

Figure 2.4a. We will approximate this function discretely using four rectangles. It can be seen in Figure 2.4b that we can make the activated output of a neuron ($z = \tanh(wx + b)$, i.e. the output after being passed through the activation function), sharp by making the magnitude of the weight w large. The function can be reflected horizontally by choosing $w < 0$ and can be shifted left and right by modifying the bias b. Therefore, by summing two neurons with weights $\mathbf{W} = [50, -50]^\top$ and biases $\mathbf{b} = [-50, 100]^\top$ we can construct a function that is of magnitude 2 on the interval $[1, 2]$ and effectively zero elsewhere as seen in Figure 2.4c.

We can repeat this process to produce the other three blocks comprising our approximation. We require eight nodes, with two nodes controlling each discrete block in the approximation. Figure 2.5 shows the structure of a neural network that could be used to approximate this function. The outputs of the nodes in the first layer (z_0 through z_7) are bounded between -1 and 1 because of the hyperbolic tangent activation function, and therefore we require another layer to obtain the output magnitudes required to approximate f^*. This final (output) layer does not get passed through an activation function and is thus effectively scaled by the weights. The full functional form of our approximation is precisely that of Equation 2.1.

Figure 2.5b shows both f and f^*. While this is a very simple example there are some observations that we can make that will extend to more complicated

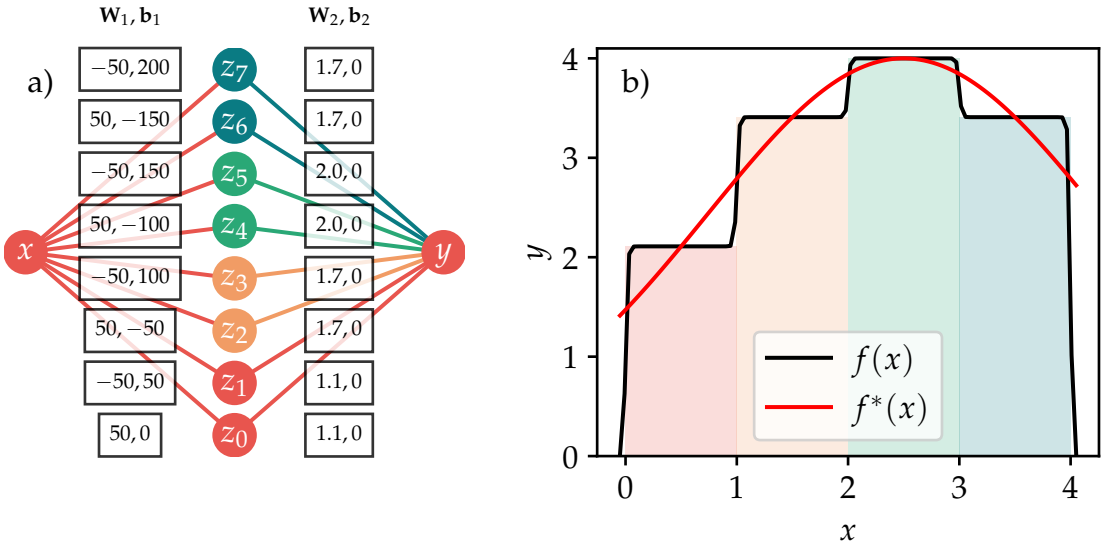


Figure 2.5: a) A graph representation of the neural network required to discretely approximate the function shown as a red curve in b). The weights and biases (w, b) for each layer are displayed. b) The output of the neural network is shown. The shaded regions denote the nodes that are being used to control the respective intervals.

examples. First, we are able to approximate the function to arbitrary accuracy by adding more rectangles, and consequently, more nodes. Secondly, we notice that while we can construct a good approximation over the domain we are concerned with, as soon as we stray from this domain, our approximation is useless. This is a demonstration of a ubiquitous problem in machine learning: neural networks are excellent for interpolation between data points, but fail miserably at extrapolation (when evaluated outside of the domain on which they were designed, i.e. trained). Of course, this can be solved in this toy example by adding another rectangle, and in practice would be solved by extending the domain of training data.

In the previous example, we had a two layer neural network with a single hidden layer. The single hidden layer did effectively all of the work in approximating our function of interest. However, the eight nodes are somewhat redundant, in fact, four of the nodes are just reflected versions of other nodes. Let us try to remove some of this redundancy. First, we will construct three sharp sigmoid functions, as we did previously, that transition from -1 to 1 at the boundaries of our approximation rectangles, that is, $x = 1$, $x = 2$, and $x = 3$. These basic "features" ($\mathbf{z_1}(x) = \{z_{1,0}, z_{1,1}, z_{1,2}\}$) are constructed with parameters

$$\theta_1 = \{\mathbf{W}_1, \mathbf{b}_1\} = 50\left\{\begin{bmatrix}1 & 1 & 1\end{bmatrix}, \begin{bmatrix}-1 & -2 & -3\end{bmatrix}^\top\right\}$$

and are shown in Figure 2.6c. We then use a second hidden layer to mix these features in different quantities and place them in two separate outputs. The parameters associated with this layer are

$$\theta_2 = \{\mathbf{W}_2, \mathbf{b}_2\} = \left\{\begin{bmatrix}0.065 & 0\\ 0.03 & 0\\ 0 & 0.03\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix}\right\}.$$

The next and final layer is the output output layer and is linear (i.e. no activa-

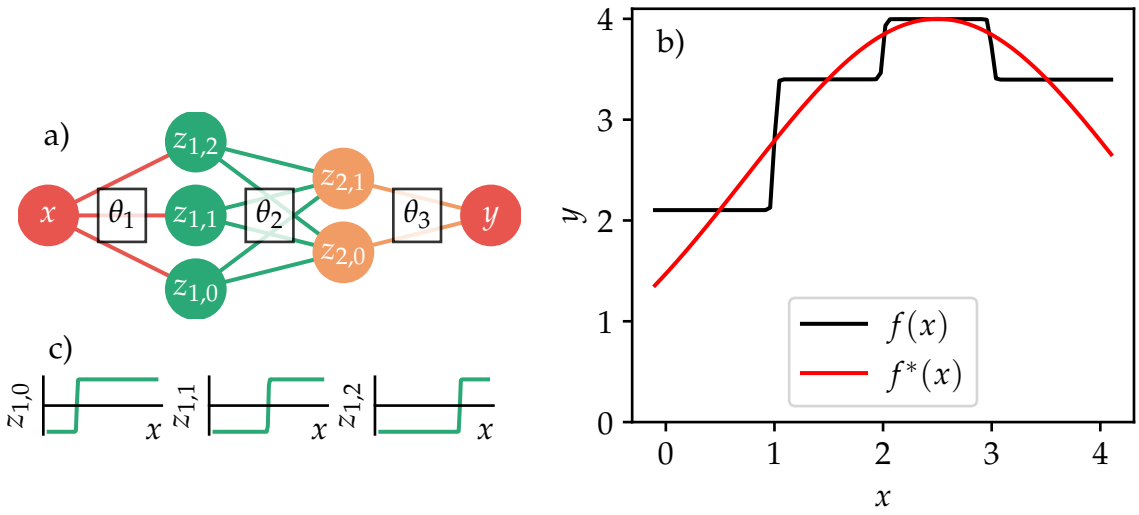


Figure 2.6: a) A graph representation of the neural network required to discretely approximate the function shown as a red curve in b). The weights and biases (w, b) for each layer are displayed. b) The output of the neural network is shown. The shaded regions denote the nodes that are being used to control the respective intervals.

tion function). It combines the previous hidden layer using parameters

$$\theta_3 = \{\mathbf{W}_3, \mathbf{b}_3\} = \left\{ \begin{bmatrix} 10 \\ -10 \end{bmatrix}, [2.75] \right\}$$

Note that one weight is negative and one weight is positive, allowing the network to subtract one mixture of features from the other. When all of this is combined, it produces the function

$$f(x) = y = \mathbf{W}_3^\top(\tanh(\mathbf{W}_2^\top(\tanh(\mathbf{W}_1^\top \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)) + \mathbf{b}_3,$$

which we plot in Figure 2.6b along with $f^*(x)$. As one can see, this neural network produces the same output as the single hidden layer network of Figure 2.5, however it uses fewer hidden neurons: five instead of eight. We can achieve this by exploiting the hierarchical structure; the first layer is used to construct features, the second layer mixes these features together, and the final layer mixes these features even further (in our case, subtracting one set of mixed features from the other) to produce the output. This is what we call a deep neural network. As we have seen, deep neural networks prove more efficient than single-layer neural networks, but still possess the ability to be universal approximators. A downside of deep neural networks is that the operations that the nodes are executing are far more abstract. In the "shallow" network, the function of each individual node was clear, but this is much less the case in this deep network. While it might seem obvious how the deep network is using its parameters to construct the output function, keep in mind that we specifically chose the parameters in this example by hand to emphasize clarity. In fact, there are an infinite number of parameter sets that achieve the desired output and we could have just as easily presented one that is more abstract.

The universality of neural networks extends to higher dimensions, both in input and output, which is at first an encouraging prospect. In reality though, as the number of neurons, layers, and connections increase, the difficulty in

finding a set of parameters that yields sufficient accuracy increases. The entire field of artificial intelligence and machine learning is essentially coming up with better ways to find these parameters. This can be either through clever exploitation of data structure (convolutional neural networks), network architectures (residual neural networks), or through live data acquisition and optimization methods (reinforcement learning).

2.1.2 Gradient descent

All of machine learning boils down to finding the right set of parameters that best approximates the data we have. This optimization problem is typically tackled with some form of gradient descent. To formalize this approach, let us define the concept of a cost function (also called a loss function), essentially a functional that measures how well our approximation matches the data in our data set. A common loss functional is the mean-squared error,

$$J(\theta) = \frac{1}{N} \sum_{i=0}^{N} \left(f^*(\mathbf{x}_i) - f(\mathbf{x}_i; \theta) \right)^2 , \tag{2.2}$$

where the summation is taken over all N data points we have available to us.

Note that we write the cost functional $J(\theta)$ as having only a dependence on the parameters. Of course, it also depends on the data $\mathbf{x}$, however we treat the training set as constant (we have no control over the data during the training

process[2]) and the goal of the optimization process is to minimize this quantity by varying θ. In most machine learning applications, this cost function is extremely high-dimensional as there are thousands, if not millions of individual parameters held within θ and therefore we cannot simply minimize Equation 2.2. This is where gradient descent is useful. We initialize the parameters randomly (more on this in Section 2.1.8), and then using the data in our data set, we update the parameters repeatedly, in small steps. The amount by which we should update each parameter can be calculated using the derivative of the loss function with respect to the parameters, and we update the parameters at each step through the update rule

$$\theta_{\text{new}} = \theta - \alpha \nabla_\theta J(\theta), \tag{2.3}$$

where α is a scaling constant that controls the magnitude of the updates. We call this the learning rate.

When the parameters are far from optimal, the quadratic nature of Equation 2.2 means that the updates will be large, bringing us rapidly toward the minimum. As the parameters near optimality, the magnitude of the updates will decrease and we will slowly approach ideal conditions. Since the gradient is a linear operator, and J is an average over all training examples, the

[2]Of course, in machine learning, the data is important and its integrity is of utmost importance. I merely mean that θ is modified in the minimization of this metric, meaning the data is immutable during training

gradient in Equation 2.3 is equivalently the average gradient over all training examples. If all training examples are used to compute the average gradient before the weights are updated, we call the algorithm **gradient descent**. We can also forgo the average and update the weights based on single training examples. This is called **stochastic gradient descent** (SGD). SGD has the benefit of making more rapid weight updates, homing in on the optimal solution quickly, but the updates are very noisy [81]. In practice, a technique called **batch gradient descent** is most often used; the training data is split into several subsets (batches) and the gradients are averaged over each subset, with weights being updated between batches. When using batch gradient descent, one needs to choose the batch size to average over. For efficiency, it is usually suggested that one use the largest batch size that hardware (e.g. GPU memory) permits, however there is still substantial discussion about whether large or small batch sizes are optimal for convergence [82–85].

2.1.3 Computing gradients: backpropagation

In order to do any form of gradient descent we must compute the gradient of the loss function with respect to the parameters. The loss function for a single training example is

$$J(\theta) = \left(f^* - f(\theta)\right)^2 . \tag{2.4}$$

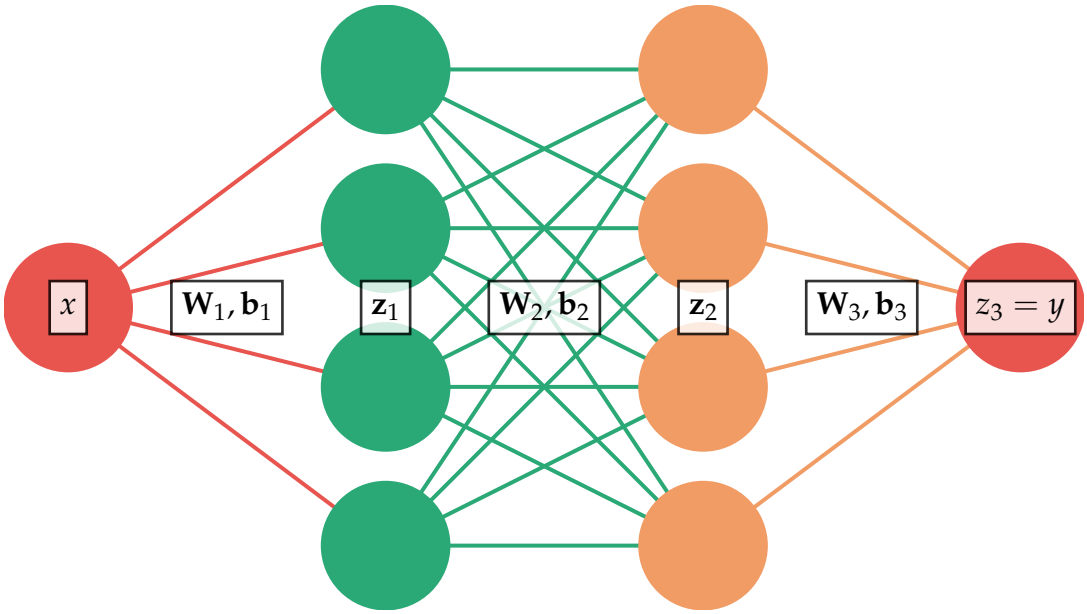


Figure 2.7: A simple three-layer (two hidden layer) neural network.

Since we will be taking derivatives with respect to the parameters, we can ignore the dependence on the input data $\mathbf{x}$; what we are computing will apply for any $\mathbf{x}$. We will therefore treat the neural network output as a function dependent on the parameters, i.e. $y = f(x;\theta) = f(\theta) = f(w_1, w_2, ..., b_1, b_2, ...)$

Let us demonstrate by example how to compute the update for a single parameter in the neural network presented in Figure 2.7. To compute the update for w_1 (i.e. the first entry of the weight matrix $\mathbf{W}_1$), we need to compute a specific entry of $\nabla_\theta J(\theta)$, namely $\partial J/\partial w_1$. Referring to Figure 2.7, let us write explicitly the function this network is computing:

$$y(w_1, ...) = z_3 = z_3(z_2(z_1(w_1))) \tag{2.5}$$

We can differentiate Equation 2.4 to obtain

$$\nabla_{w_1} J = \frac{\partial J}{\partial w_1} = -2(f^* - y)\frac{\partial y}{\partial w_1}, \tag{2.6}$$

and then use the chain rule on Equation 2.5 to compute the remaining partial derivative. Noting that $y = z_3$,

$$\frac{\partial y}{\partial w_1} = \frac{\partial z_3}{\partial w_1} = \frac{\partial z_3}{\partial \mathbf{z}_2}\frac{\partial \mathbf{z}_2}{\partial \mathbf{z}_1}\frac{\partial \mathbf{z}_1}{\partial w_1}, \tag{2.7}$$

and since the output of each individual layer is given by

$$z_3 = \mathbf{W}_3^\top \mathbf{z}_2 + \mathbf{b}_3, \quad \mathbf{z}_2 = \sigma(\mathbf{W}_2^\top \mathbf{z}_1 + \mathbf{b}_2), \quad \text{and} \quad \mathbf{z}_1 = \sigma(\mathbf{W}_1^\top x + \mathbf{b}_1) \equiv \sigma(\mathbf{z}_0), \tag{2.8}$$

then

$$\frac{\partial z_3}{\partial w_1} = \mathbf{W}_3^\top \sigma'(\mathbf{z}_1)\mathbf{W}_2^\top \sigma'(\mathbf{z}_0)x. \tag{2.9}$$

Here, $\sigma'(\cdot)$ is the analytic derivative of the activation function. Taking inspiration from Equation 2.3, we would therefore update this weight with the operation

$$w_{1,\text{new}} = w_{1,\text{old}} + 2\alpha(f^* - z_3)\mathbf{W}_3^\top \sigma'(\mathbf{z}_1)\mathbf{W}_2^\top \sigma'(\mathbf{z}_0)x. \tag{2.10}$$

At this point, let us note several things. First, this is the update for *a single* parameter in the network. Networks, in practice, have millions of parameters, and therefore computing the gradients is the most costly part of training a neural network. Luckily, most of the computations are reusable. For example, if we wish to compute the parameter update for b_1, the computation remains identical except that the final partial derivative in Equation 2.7 should be differentiated with respect to b_1, e.g. $\partial \mathbf{z}_1 / \partial b_1$. Therefore, when designing software that performs backpropagation, it is beneficial to collect the intermediate results and store them as they can be used over and over again.

It is also important to note that the activation function we choose influences the weight updates significantly in several ways. It has so much importance that we should have a slight digression into a discussion of activation functions.

2.1.4 Activation functions

Any nonlinear function can be used as an activation function, however there is a set of nonlinear functions that are commonly used. I plot several of the most common activation functions with their derivatives in Figure 2.8, alongside a qualitative measure of evaluation time for both $\sigma(z)$ and $\sigma'(z)$.

The linear activation is included here for completeness, and is not an activation function as it does not provide a nonlinearity. It is commonly used when doing regression tasks where the neural network output assumes continuous

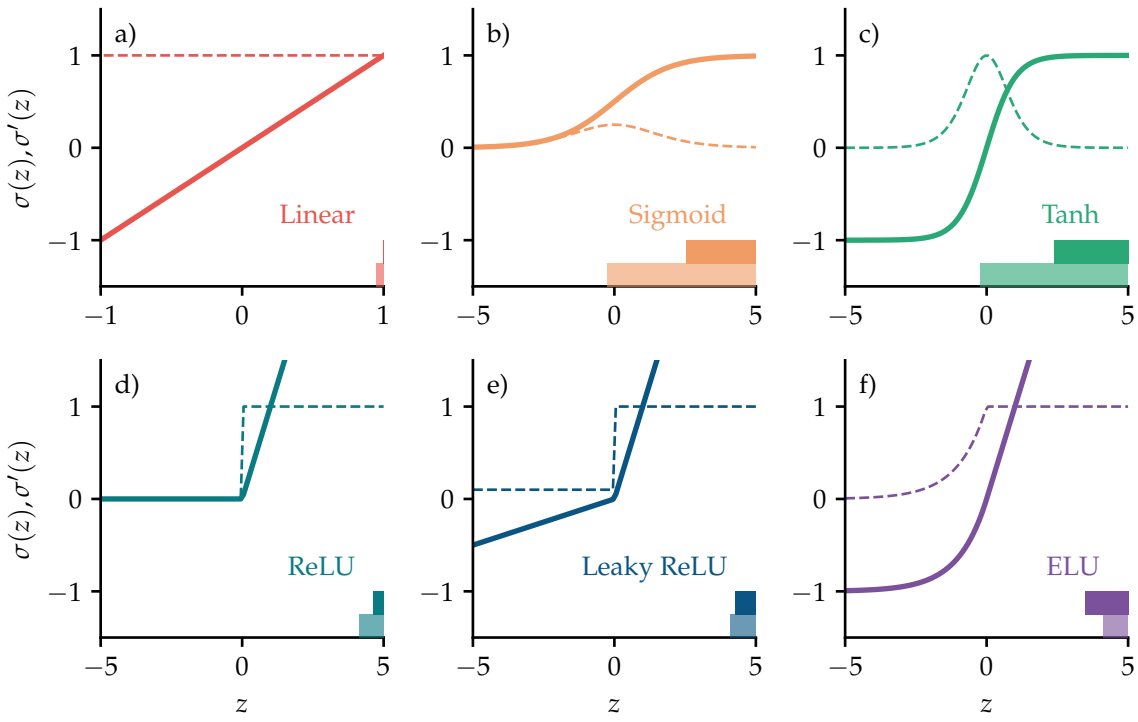


Figure 2.8: Six commonly-used activation functions and their derivatives. A qualitative measure of execution time for $\sigma(z)$, and its derivative $\sigma'(z)$ is shown under each graph as a dark bar and a light bar, respectively.

values.

The sigmoid and hyperbolic tangent activation functions are commonly used when the neural network output should be bounded, such as in predicting probabilities between 0 and 1 (sigmoid) or when performing binary classification. Sigmoid and tanh activations have several drawbacks. They are somewhat expensive to compute, as are their derivatives, which is an important consideration given the large number of evaluations required for training and then ultimately for using the model to make predictions (inference). Furthermore,

the derivatives have vanishing magnitude as the input magnitude increases. Should the input to an activation stray far away from 0, it is very difficult for gradient descent to bring it back as the gradients are so small. Furthermore, it is important to carefully initialize parameters in the neural network so that the gradients start out large, and learning does not immediately stagnate. This is commonly referred to as the "vanishing gradient problem".

Rectified Linear Units (ReLU) solve the execution time dilemma; they are extremely fast to compute. ReLU suffers from the vanishing gradient problem; for all negative inputs the gradient is zero, and thus gradient descent steps that push an input into this region are irrecoverable. With ReLU, nodes tend to become "dead" as a zero magnitude gradient means that the parameters can never be updated through gradient descent.

Leaky ReLU and ELU strive to solve this problem. The gradient of the negative side of the function is set to a user-defined value (or defined based on a user-defined parameter in the case of ELU).

In most current, practical, applications, Leaky ReLU is the activation function of choice for hidden layers because it never produces dead neurons, and its execution time is low.

2.1.5 Optimizers

In all modern deep learning frameworks, such as Tensorflow [34] and Pytorch [35], the job of performing backpropagation and the efficient recording of gradients is performed under-the-hood automatically. This is referred to as "automatic differentiation". Backpropagation only refers to the computation of the gradients; what is actually done with those gradients in order to update the parameters is performed by an **optimizer**. Gradient descent (and its stochastic and mini-batched descendants, all typically referred to as "SGD") are the simplest optimizers available, but improvements have been made to optimize objective functions more efficiently.

The concept of momentum can be added to gradient descent to improve its performance in "ravines"—regions of the parameter space where the gradient in one dimension is much smaller than the gradient in another dimension. In such a case, updates can oscillate from side to side, where gradients are large, while little progress is made in the other dimension. Momentum helps by cancelling out these higher-frequency oscillations. In terms of the update rule, Equation 2.3 becomes a two-step computation. Let us use g_t as shorthand for the gradients, that is $g_t \equiv \nabla_\theta J(\theta_t)$. Then the weights are updated according

to

$$u_t = \gamma u_{t-1} + \alpha g_t \tag{2.11}$$

$$\theta_{t+1} = \theta_t - u_t \tag{2.12}$$

in which the weight update u_t at step t, is recorded and used in the next step, discounted by a factor $\gamma < 1$, dampening oscillatory behaviour.

Since not all features appear in a data set with equal frequency, neurons that evolve to contribute to more rare features can take a very long time to be refined, as they infrequently are provided with any feedback (i.e. gradients with respect to these parameters are small). The optimizer `Adagrad` [86] attempts to tackle this by keeping track of a dynamic learning rate for every parameter. It does so by accumulating the sum of squared gradients independently for each parameter, and then normalizing the updates by this factor:

$$G_\theta \equiv \sum_{t'=0}^{t} (g_{t'})^2 \tag{2.13}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\epsilon + G_\theta}} g_t. \tag{2.14}$$

As such, gradients that are frequent or strong produce weaker updates over the course of the training procedure. A small factor ϵ is used to avoid division by zero. A benefit of `Adagrad` is that the standard problem of choosing a good

learning rate α becomes far less important. The problem with this approach is that G_θ is a monotonically increasing function and thus all updates become smaller over the course of training, eventually stagnating the learning process.

`RMSProp` (included in a lecture by Geoffrey Hinton [87], but never published) fixes this problem by replacing the summation with an exponentially decaying average of the squared gradients, with updates according to

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{2.15}$$

$$u_t = \frac{\alpha}{\sqrt{\epsilon + E[g^2]_t}} g_t \tag{2.16}$$

$$\theta_{t+1} = \theta_t - u_t. \tag{2.17}$$

At the same time `RMSProp` was proposed by Hinton, `Adadelta` was independently developed by Matthew Zeiler at Google [88]. It is essentially identical to `RMSProp`, however the Zeiler identifies that dividing by the squared gradients introduces erroneous units into the update rule (units of squared gradient in the denominator). Therefore, he multiplies by another exponential average, this time of the past squared weight updates, to nondimensionalize

the adaptive scaling. Altogether this results in the following update rules:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{2.18}$$

$$E[u^2]_{t-1} = \gamma E[u^2]_{t-2} + (1-\gamma) u_{t-1}^2 \tag{2.19}$$

$$u_t = \frac{E[u^2]_{t-1}}{\sqrt{\epsilon + E[g^2]_t}} g_t \tag{2.20}$$

$$\theta_{t+1} = \theta_t - u_t. \tag{2.21}$$

Notice that `Adadelta` avoids the need to set a learning rate in the first place, as the updates are scaled purely adaptively.

Adaptive Momentum Estimation, `Adam` [89], incorporates the idea of momentum, as well as scaling by the squared gradients. Decaying averages of the first two moments are computed according to

$$m_t = \frac{1}{1-\beta_1^t} \left(\beta_1 m_{t-1} + (1-\beta_1) g_t \right) \tag{2.22}$$

$$v_t = \frac{1}{1-\beta_2^t} \left(\beta_2 v_{t-1} + (1-\beta_2) g_t^2 \right). \tag{2.23}$$

$$\tag{2.24}$$

Typically, m_0 and v_0 are initialized to zero, introducing a bias toward zero[3]. As such, the authors multiply by the fraction in front to compensate for this

[3]It took me longer than I'd like to admit to understand what this terminology meant. When you force the first term in what is essentially an average to be zero, you bias the average "toward zero" and the running average will underestimate the true average. The effect of such bias decays exponentially with the number of terms in the average, and thus the β^t terms

bias. β_1 and β_2 are typically left unchanged at default values of 0.9 and 0.999, respectively. The update is then performed according to

$$u_t = \frac{\alpha m_t}{\sqrt{v_t} + \epsilon} \tag{2.25}$$

$$\theta_{t+1} = \theta_t - u_t. \tag{2.26}$$

There are several other, lesser-used optimizers, but as of writing, `Adam` is generally preferred as the best all-around optimizer for general tasks, since it combines the features of stochastic gradient descent with momentum, and the adaptive per-parameter learning rates introduced with `Adagrad`, and refined in `Adadelta` and `RMSProp`. Sebastian Ruder provides an excellent comparison of the most common optimizers is his report in ref. [90].

When training a neural network, it is common to plot the objective function (loss) as learning progresses. In Figure 2.9, we see several loss curves. Typically, SGD performs the most poorly and is rarely used in practice. We can see that `Adagrad` suffers from stagnation, learning quickly at first, but being overtaken by other methods. It's important to note that these are just example curves for a single application; optimizers behave differently for various problems depending on a huge number of factors. It is also clear in hindsight to see which optimizer performed the best; clearly `Adam` is the best choice *in this specific case* if the computational budget permits 1 million iterations. One

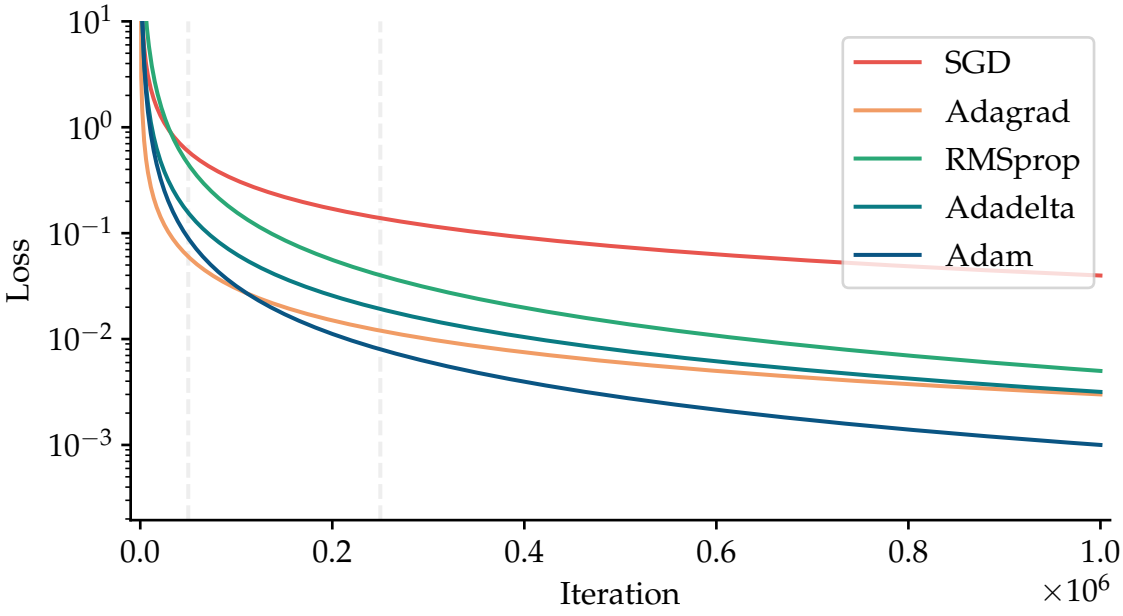


Figure 2.9: Plotting the loss as training proceeds is a common way to monitor the training process. Here we see curves demonstrating what one could expect from several optimizers.

million iterations is arbitrary; we could have stopped training at 50 000 iterations, at which point the picture would appear very different. There are many considerations when selecting an optimizer, that make no single optimizer the best: perhaps the computational budget is low and will not afford the ability to investigate the effect of different learning rates. In this case, `Adadelta` might be enticing as there is no need to set a learning rate. Perhaps we have a huge amount of computational resources available to us, but we have a tight deadline in real time. In this case, it is probably best to just choose a good standard, such as `Adam` and run many parallel trials testing out many different learning rate choices.

It is useful to note that the term “gradient descent” is used as a collective term to refer to all of these flavours of gradient-based optimization strategies.

2.1.6 Hyperparameters

By now, we have talked a lot about parameters, which are the variables that are being directly modified by the learning algorithm. There are, however, many other parameters that greatly affect the training process and the performance of a neural network. These parameters are not involved in the automatic differentiation and are modified by hand by the researcher. As such, they are called **hyperparameters**, and a good portion of the research time and resources in developing a model are allocated to the determination of good values. Hyperparameters comprise such quantities as the learning rate and decay parameters in optimizers, parameters associated with regularization methods (discussed in Section 2.1.7.2), but other, more conceptual quantities can also be considered hyperparameters. The batch size, number of hidden layers, width of hidden layers, the total number of training examples, and even the choice of optimizer itself must be determined by the researcher when developing a model.

A common approach is to perform a grid search [91], defining a range for each parameter and iterating over selections from this range to find optimal values. For a very small number of hyperparameters, this can work, but with a larger number, this becomes infeasible. In such cases, random search [92]

can actually perform better. More intelligent optimization strategies using Bayesian optimization have also been used [93]. No matter the method, it is a particularly difficult task because training for a short amount of time, as demonstrated in Figure 2.9, does not necessarily lead to results that are indicative of the long term behaviour. In most cases, unless extremely high performance is necessary, or the problem at hand is unstable or very difficult, it is sufficient to choose "reasonable" values without the grand expense of a comprehensive hyperparameter tuning experiment.

2.1.7 Overfitting

A perennial problem in machine learning is the trade off between bias and variance. In this context, bias refers to error in predictions due to the inability of the chosen model to capture the relationships in the data. Using linear regression on a nonlinear problem is an extreme example; no amount of training or additional training data will reduce the underfitting of a high-bias model. Bias is clearly not good and you can reduce bias by increasing the complexity of the model (adding layers, widening layers, etc.), however at some point, the model will begin to overfit the training data, leading to what is known as high variance. Balancing bias and variance is an important part of designing machine learning models.

The goal of machine learning is to model the trends present in some training

data, and then be able to apply the model to new data. As such, we say we want the model to be "**generalizeable**" to new data. Overfitting (high variance) is a major concern with neural networks because of the typically large number of parameters (usually numbering in the millions), and the relative sparsity of training data. Overfitting occurs when the model is tuned to perform so well on the training set that it fails to generalize to new data. Figure 2.10a demonstrates this phenomenon. The red dots represent the training data, and the orange curve represents a polynomial that has been overfit to the training data. We can see here that, between training examples, the polynomial fails to capture the true distribution of data (green crosses) and would therefore fail to correctly predict the y coordinate of a green cross.

2.1.7.1 Detecting overfitting

In order to detect overfitting, the training data is split into two separate data sets at random, which we call the "training" data set, and the "validation" data set[4]. One might typically choose to use 90% of the data for training and reserve 10% for validation. The way this helps detect overfitting is quite simple: we train the model only on the training set, and monitor the loss as training proceeds. Additionally, every so often, we use our trained model to make

[4]Sometimes the data is split into three data sets, with the third set being labelled the "testing" data set. For our purposes, we will combine the concepts of validation and testing sets into one set that we will formally call the validation set. In some included publications, we use the term "testing set" to refer to what others might call the "validation set"

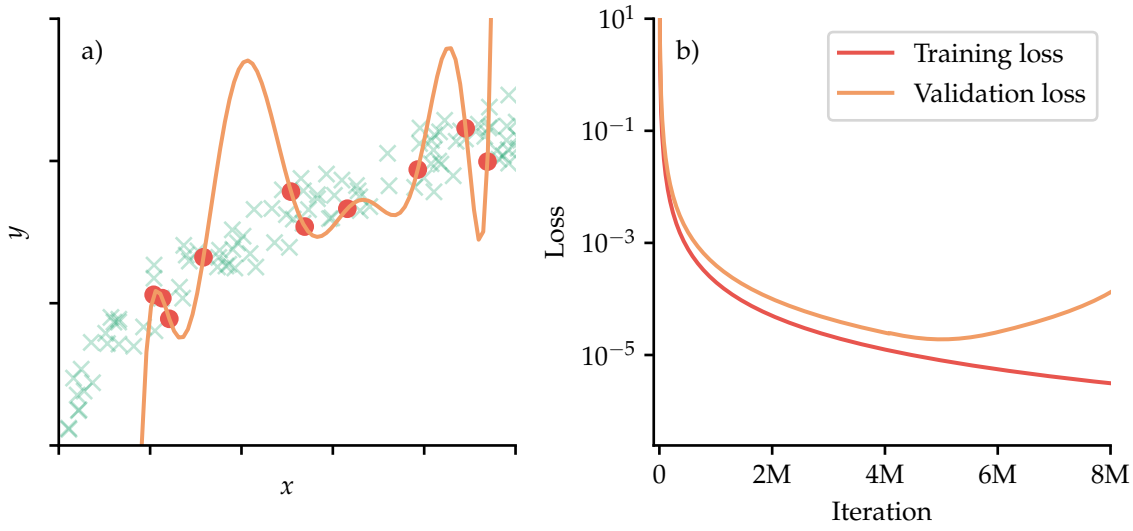


Figure 2.10: a) The orange function is fit to the red points and models them perfectly, however if we look at more points from the same generating distribution (green crosses), the orange fit fails to represent these completely. In this case, we say that the model is overfit to the red training data and fails to generalize. b) When monitoring loss during training, one should set aside some training examples to act as a "validation set". When the loss on the validation set begins to increase, while the loss on the training set continues to fall, the model is overfitting to the training set.

predictions on the data in the validation set and record its loss as well. The validation loss should always be greater in magnitude than the training loss (although there are rare but valid technical reasons[5] why this could not be the case), and both losses should decrease on average over the training process. At some point, we expect overfitting to occur, and when this begins to happen, the performance of the model on the validation set will suffer, and the validation loss will begin to rise. Figure 2.10b demonstrates this process. As loss can

[5]If training loss is calculated after each iteration (every batch) and validation loss is calculated after every pass through the training set (every epoch), then the validation loss benefits from more weight updates, and thus could appear lower than the training loss

fluctuate during training due to the stochasticity in batch gradient descent, it can take several iterations to be able to identify with certainty that the validation loss is increasing, and therefore it is important to frequently save (i.e. "checkpoint") the model parameters so it is possible to roll back to near where the validation loss achieved its minimum.

2.1.7.2 Regularization

We can detect overfitting, but it would be better to prevent it, or at least delay it so that the validation loss achieves a lower value before overfitting. The easiest way to prevent overfitting is to use more training data. Including more training data means that a model of the same capacity (i.e. number of parameters) will not be as capable of fitting to specific examples. Of course, in some situations, data is limited or difficult to acquire and it is not possible to acquire more. Taking the opposite approach, it is sometimes sufficient to reduce the model capacity; decreasing the width and depth of a network reduces its ability to "memorize" training examples. This, of course, can also hinder the ability of the network to learn complex features, and doing so, like many other aspects of machine learning, is a balancing act.

To reduce the burden of finding the precise ratio of training data to network capacity that prevents overfitting, researchers have come up with methods, collectively called regularization, that make models less susceptible to overfitting,

but also retain the network capacity in the event that the true problem requires it. Encouraging the magnitudes of parameters to be small assists in the prevention of overfitting, as it makes it more unlikely that a single weight could be tuned to detect a single training example, thus encouraging the weights to "work together" to detect features. Such encouragement mathematically takes form as a penalty on the loss function. Typically, either an L1-norm is used:

$$J(\theta) = J_0 + \lambda \sum_{\theta' \in \theta} |\theta'|, \tag{2.27}$$

or an L2-norm:

$$J(\theta) = J_0 + \lambda \sum_{\theta' \in \theta} (\theta')^2, \tag{2.28}$$

where in both cases J_0 is the unregularized loss function, e.g. Equation 2.2. In this section I will refer to L1- and L2-regularization as "L1" and "L2" for simplicity. L1 has the beneficial consequence of pushing weights that have little impact on the output to zero, allowing the compression of a network (zero weights need not be stored or computed as weights contribute through a summation). In the case where L1 is used on the input layer, it can serve as an indicator that a feature is not important, and can thus be omitted from the data set completely. Figure 2.11a demonstrates the differences between the two regularization methods for a two-parameter model. A hypothetical loss function is plotted as contour lines with a black point at the global minimum.

Two isolines of constant unit regularization penalty are plotted: $1 = |\theta_1| + |\theta_2|$ representing L1 penalty and $1 = \theta_1^2 + \theta_2^2$ representing the L2 penalty. Supposing the global minimum is outside the constraint region, the lowest possible loss that can be achieved with L1 (at constant regularization penalty) occurs when $\theta_2 = 0$. In this case, θ_2 is the least descriptive weight, meaning varying it has less effect on the cost than varying θ_1, and thus L1 would kill it off.

L2 on the other hand, results in a lower overall cost but preserves the magnitude of both weights. Recall that the regularization terms are added to the cost function and are thus minimized via the optimizer. Thus, the regularization term encourages the contraction toward zero of the regularization boundaries shown here, and both L1 and L2 encourage the magnitude of weights to be reduced.

A different regularization approach that has gained popularity is called dropout [79, 94]. Dropout works by randomly removing different neurons from the neural network (i.e. setting node activations to zero), at the beginning of every forward pass. A diagram of what this could look like is shown in Figure 2.11a. The probability of a given neuron being removed is controlled by a hyperparameter p; the authors suggest that $p = 0.5$ is generally suitable for hidden layers, and a value closer to $p \approx 0$ is best for input layers [79]. When the input to a latter node is removed due to dropout in the previous layer, the remaining inputs must be scaled accordingly. Dropout should be disabled during

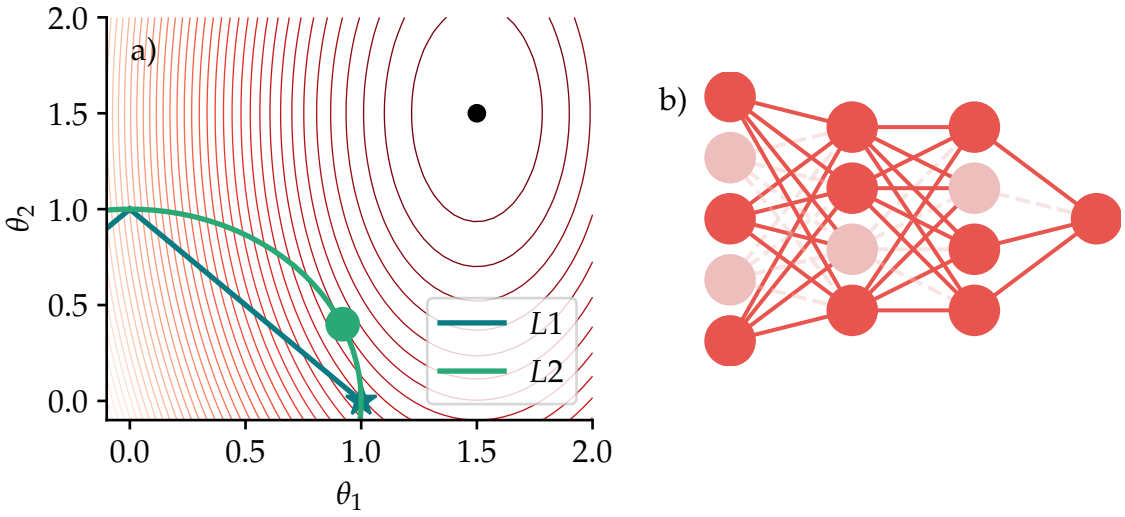


Figure 2.11: a) The effective difference between L1 and L2 regularization is demonstrated on a two-parameter model. The contours represent the loss function, with the black point representing the global minimum. Two isolines of unit regularization penalty are plotted. Under L1-regularization, the lowest loss value is obtained through zeroing θ_2, while with L2-regularization, both parameters together obtain the lowest loss value. b) We plot a schematic of the effect of using dropout. Every time the network is evaluated, weights are set to zero at random. This reduces the dependence upon individual weights and forces weights to "work together".

validation.[6] Dropout prevents overfitting by preventing codependence of neurons in a neural network; it forces neurons to act independently as they cannot trust the output of other neurons being present, and are therefore hindered in their fitting ability. Dropout is essentially a stochastic way to contemporaneously train an ensemble of neural networks that all are tasked with predicting the same output. When it comes time for inference, dropout is disabled, and outputs are effectively the averaged samples of many neural networks.

[6]except in specific cases, where dropout is being used to gain a measure of confidence in prediction [95–97].

Dropout leads to the inclusion of more neurons, and thus dropout is an effective, but not necessarily an efficient way to perform regularization.

2.1.8 Initialization

For the optimization process to be efficient, one needs to choose good initial values for all trainable parameters. Different parameters must be chosen for each neuron so as to break symmetry (identical weights belonging to two different neurons with identical activation functions will always result in the same gradients via backpropagation and thus the same optimizer updates and thus the two neurons will remain identical forever.

One way to break symmetry is to generate the parameters randomly. Almost always, weights are drawn from a normal distribution or a uniform distribution, and biases are set to some constant value (another hyperparameter) [98]. However, if the initial parameters are too large, activation functions will saturate and learning will be slow due to vanishing gradients. Large weights will cause large gradients which can lead to instability in the gradient descent process. If parameters are initialized too small, then it will take a very long time for symmetry to be broken and for neurons to specialize to different features.

An initialization strategy proposed by Xavier Glorot and Yoshua Bengio [99]

is to sample weights from a uniform distribution

$$\mathbf{W} \in \mathbb{R}^{m \times n} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right), \tag{2.29}$$

where m and n represent the number of input and output connections of a given layer, respectively. This scheme is interchangeably referred to as Xavier initialization and Glorot Initialization, and is usually a sufficient balance.

2.1.9 Classification

So far, we have only used mean-squared error as a loss function, which is appropriate for **regression tasks** (i.e. tasks where we are predicting continuous-valued functions). Neural networks are commonly used for **classification** tasks as well, that is, tasks where the desired output is interpreted as a categorical assignment of the input data. It is common to "one-hot encode" such an output, where the labels in the data set are encoded as a vector of zeros, with a single unit-valued entry corresponding to the correct category. The neural network is then tasked with outputting a similar vector, and through training, we hope that the neural network predicts low values for vector elements that do not correspond to the true category, and a high value for the element that does. An example classification of an image of a cat is shown in Figure 2.12. The neural network is outputting a high magnitude for the cat element, meaning it

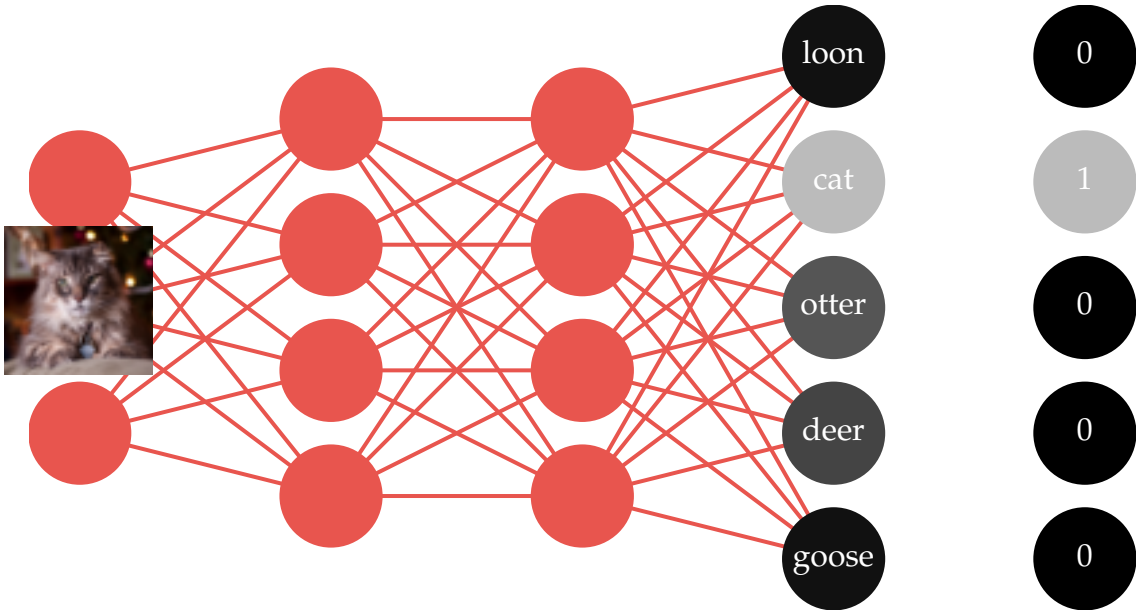


Figure 2.12: When features pertaining to a cat are fed into an animal-classifying neural network, the output node with greatest activation corresponds to the "choice" the neural network is making. Such networks are trained using one-hot encodings as ground truth labels, such as the vector of zeros and one shown on the right.

is classifying correctly. The moose and deer labels are slightly more activated than those of goose and loon; this can be attributed to the fact that a cat visually shares more similar features with the ungulates, and much less with the avians.

In classification tasks, it is common to feed the unactivated output of the final fully connected layer ($\mathbf{z}$) through a softmax layer, which computes the

softmax function over all k classes:

$$\mathbf{f} = \left\{ f(\mathbf{z})_j = \frac{e^{z_j}}{\sum_k e^{z_k}}, \quad \text{for } j = 1, ..., k \right\}. \tag{2.30}$$

This produces a normalized output which can be interpreted as the probability of the input belonging to class j. The output of the softmax function is then used to compute the cross entropy,

$$J(\mathbf{f}) = -\frac{1}{k} \sum_j \left(y_j \ln f_j + (1 - y_j) \ln(1 - f_j) \right) \quad \text{for } j = 1, .., k, \tag{2.31}$$

where $\mathbf{f}$ is the output of the softmax function, and y_j are the corresponding correct values (labels). This is the quantity that is optimized during training (the loss).

2.2 Convolutional Neural Networks

2.2.1 Motivation

All of the neural networks presented so far are of the fully-connected variety. These networks excel when applied to unstructured data. Typically they are limited to the number of input features due to computational complexity. If, however, the input is structured (such as in the way the pixels in an image

are), utilizing an MLP is wasteful for two main reasons, and often infeasible anyway.

Firstly, images are inherently translationally invariant. A cat is a cat whether it appears in the top right, or the bottom left of an image. An MLP has no way to capture translational invariance, and must develop features to detect both top-right cats and bottom-left cats, and cats at any other region of the input space.

Secondly, nearby pixels are highly correlated and work together to form larger scale structures. For example, the nose of a cat comprises many pixels that are all within the same region, or **receptive field**, and effectively detecting this has little to do with the distant pixels on the other side of the image. The receptive field in an image is limited in extent.

Furthermore, images are commonly many hundreds of pixels in a single dimension, resulting in thousands, or even millions of input pixels. A fully-connected network is infeasible, as an input layer of such a size would require millions, if not billions of weights.

Convolutional neural networks (CNNs), proposed by Yann LeCun in 1990 [100] solve all of these shortcomings. In a convolutional neural network, the learnable parameters are grouped together in what are called **filters** (what a mathematician might call a **kernel**). During the forward pass, D individual kernels are convolved with the input data, creating a new **feature map**. Deep

convolutional networks chain together many convolutional operations in successive layers[7]. Usually some type of size-reduction method is used to reduce the spatial extent of the feature maps. This could be in the form of pooling or strided convolutions. All of the previously established techniques still apply, such as activation functions on the layer outputs, dropout can be used to prevent overfitting, et cetera.

CNNs didn't hit their stride until 2012 when Krizhevsky, et al. used a CNN [101] to beat the ImageNet challenge, significantly surpassing other methods of the time. Since this, CNNs have become ubiquitous in the field of deep learning and much development has been spent improving their performance. The response of hardware manufacturers in the development of specialized hardware (NVIDIA's Tensor Cores) that excels at low-precision (most images only use 8 bits per channel) convolutional operations has further accelerated the development and widespread use of CNNs.

2.2.2 Details

For the purposes of further discussing the CNN, let us assume a two-dimensional input of size $C \times L \times L$ (i.e. a C-channel image of size $L \times L$ pixels; for a photograph, $C = 3$ for red, green, and blue). The principles are the same in higher and lower dimensions. Figure 2.13 shows a possible first layer of a convolu-

[7] Almost all convolutional neural networks are deep

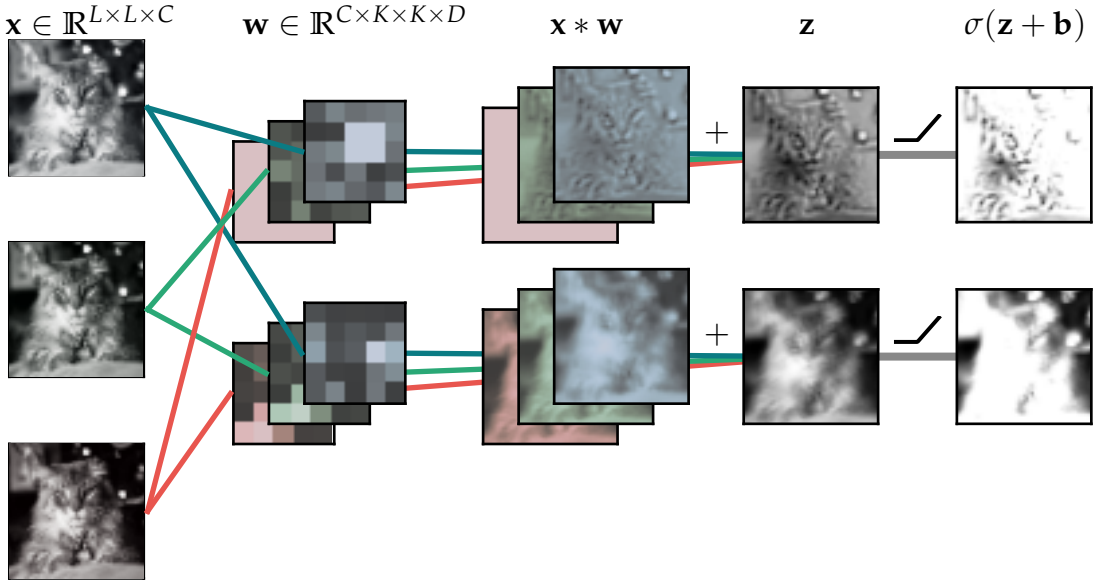


Figure 2.13: The first convolutional layer of a CNN could look like this. An input image comprised of $C = 3$ channels is convolved with $D = 2$ kernels of size $K \times K$. Since there are three input channels, each of the two kernels is of size $3 \times K \times K$. In this example $K = 5$. The outputs of the convolutional operations are summed element-wise for each filter separately, mixing together the channels. A bias b is added, and an activation function is applied. These final images, or "feature maps" would serve as the input to a subsequent convolution layer, and the process repeats for many layers in the case of a deep convolutional neural network.

tional neural network. The three RGB input channels are convolved with a kernel of weights of shape $\mathbf{w} \in \mathbb{R}^{C \times K \times K \times D}$. D controls the width of the layer and can be thought of as another hyperparameter. D directly relates to the number of feature maps produced by the layer, and thus affects the model capacity greatly. The size of the kernel is defined by K and effectively sets the length scale on which the layer can detect features. When performing the convolution, there are usually a number of choices to make. Firstly, the type of padding used

is important both for consistency with the data as well as managing the size of the feature maps flowing through the network. There are several common types of padding:

- no padding - if no padding is used, the output feature map will be smaller in size than the input, as the edge pixels must be skipped in order to fit the kernel inside the image. The reduction in size is dependent on the kernel size. Usually this is called “valid” padding.
- constant padding - the perimeter of the input can be padded with a constant value to preserve the output size. The constant value is most often zero, and thus this is commonly referred to as “zero padding”. Since the output is the same shape as the input, it belongs to a class of padding methods commonly called “same” padding.
- reflected padding - another “same” padding method, this method reflects the pixels at the edges to produce an output of the same size as the input.
- periodic (circular) padding - useful in situations where the input data is periodic (i.e. defined on a circle (1d) or torus (2d)), which is often the case in computational materials science applications.

In any case when padding is used, the amount of padding is denoted by P, for example, if one pixel is added to all four edges of a two-dimensional input, the padding amount would be $P = 2$ (one pixel added on each side).

In addition to padding, convolutional layers often employ strided convolutions in order to reduce the size of the feature maps. Reducing the size of the feature maps is beneficial as subsequent layers will require far fewer calculations as there will be fewer pixels in the image. Additionally, decreasing the size of the feature maps allows subsequent layers to learn a hierarchy of features with much smaller convolutional kernels than would otherwise be required. Instead of sliding the kernel one pixel at a time, a strided convolution slides the kernel by S pixels, thus reducing the size of the feature map by a factor of $1/S$.

For a given convolutional layer, the spatial size of the output L' can be calculated based on the size of the input L, the kernel size K, the padding amount P, and the stride length S, through

$$L' = \frac{L - K + P}{S} + 1. \tag{2.32}$$

In a deep CNN, a number of convolutional layers are chained together, giving the network its depth. The subsequent layers work qualitatively in much the same way as in an MLP, learning to extract a hierarchy of visual features. After the desired depth of convolutional network is reached, the final feature maps are most often[8] flattened to a single vector and fed through a fully-connected network, which performs a final weighted reduction to the de-

[8]except in the case of novel architectures such as fully-convolutional networks [102]

sired output shape, perhaps a one-hot encoding for a classification task, or a vector of numbers for a multi-dimensional regression task.

When one is designing a CNN architecture, the choices of K, P, and S place an upper bound on the depth of the network (i.e. the number of layers that are possible), as each layer has the potential to reduce the size, and there is of course a lower bound on the useful size of an image.

Backpropagation of gradients through convolutional layers works in the same way as multilayer perceptron networks. CNNs still employ the same *"weights $\times$ input + bias, fed into nonlinearity"* paradigm as MLPs, however the gradient of each parameter will be averaged over not only training examples, but also the several input features (pixels) that get multiplied by that particular weight in the forward pass. This results in a significant amount of bookkeeping, but is swept under-the-hood in most deep learning frameworks that offer automatic differentiation (e.g. TensorFlow [34] and PyTorch [35]).

2.3 Generative models

So far, we have only discussed supervised, predictive models, that is, models that map an input to a deterministic output. Machine learning can do far more though, and generative models are promising approaches for a variety of applications. Instead of fitting a function, generative models can be formulated as a

maximum-likelihood problem. The objective of a maximum-likelihood problem is to find a parameterized distribution that maximizes the probability that the observed data (the training set, p_{data}) comes from our model distribution. That is, we wish to find the parameter set θ^* such that

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{x \sim p_{\text{data}}} \log p_{\text{model}}(\mathbf{x}|\theta). \tag{2.33}$$

The way in which we represent and train p_{model} is what separates different techniques in generative machine learning. Some techniques, such as deep belief networks [103], PixelCNN [104], variational autoencoders [105], and Boltzmann machines [106–108] attempt to learn a functional form of this distribution directly. These are known as **explicit density** generative models because they explicitly model the probability density function. Other generative methods learn instead an internal representation that can be sampled to obtain examples that come from p_{model} (which we desire to match the underlying distribution p_{data}), but one cannot query for this probability directly. These are the class of **implicit density** generative models. Implicit generative models are based on the core premise that in order to synthesize example data, an understanding of the relevant features must be somehow present in the model, and the training procedure attempts to learn these relevant features, usually in the form of optimizing a set of coefficients in a function that transforms examples drawn from a latent distribution p_z into examples from p_{model} that are

indistinguishable from examples drawn from p_{data}.

2.3.1 Generative adversarial networks

Most important for the contents of this dissertation is the generative adversarial network (GAN). The GAN [109] became a relatively widespread phenomenon after Goodfellow's 2014 introduction of the method. The GAN is the combination of two "players" (separate neural networks), working against each other as adversaries. One player acts as a generator, taking a vector of random noise z as input and transforming these to examples that ideally come from the p_{data} distribution. The other player works as a discriminator and learns to tell the difference between examples coming from the generator and true, ground truth examples. These players are trained simultaneously with the generator trying to trick the discriminator, and the discriminator learning how to better tell apart the generator's propositions from the true training examples. A successfully trained GAN converges to a state where the generator is so good at producing examples that the discriminator cannot tell the generated examples from the ground truth examples. The key to the success of this method is that the generator is provided with hints about its failure in the form of the gradients, i.e. both the generator and discriminator perform backpropagation using the gradients from the discriminator network. This allows the generator to learn not only whether or not it succeeded in tricking the discriminator, but

effectively gives it access to the reasoning behind its success or failure.

Formally, let us assume we have a collection of data $x \in \mathbb{R}^S$ (a training set) of uniform shape S. This data set p_{data} is a glimpse into an underlying distribution p_x. Let us define a prior distribution $p_z(z)$, and a neural network $G_{\theta'}(z)$ that is parameterized by parameters θ'. The desire is to have $G_{\theta'}$ transform samples drawn from p_z into samples that appear to come from p_x; $G_{\theta'}$ represents the model distribution p_{model}. We define a second and different neural network $D_\theta(x)$ parameterized by parameters θ that outputs a single scalar value between 0 and 1. $D_\theta(x)$ directly represents the probability that x comes from p_{x} rather than p_{model}. Through training, we wish to maximize the probability that D_θ assigns the training examples and samples from $G_{\theta'}$ correctly. Since $G_{\theta'}$ generates examples from noise, and D_θ discriminates between generated and data set examples, we call the two networks the **generator** and **discriminator**, respectively.

Thus the objective for both networks will be captured by the minimax game

$$\min_G \max_D \left[\mathop{\mathbb{E}}_{x \sim p_{\text{data}}} \log D_\theta(x) + \mathop{\mathbb{E}}_{z \sim p_z} \log(1 - D_\theta(G_{\theta'}(z))) \right]. \tag{2.34}$$

The generator and discriminator can also be provided with labels y for the true examples. This is called conditioning and acts to help stabilize the generative adversarial network [110], improving the notoriously sensitive training

process [111]. The modification of the objective function is minimal:

$$\min_G \max_D \left[\mathop{\mathbb{E}}_{x \sim p_{\text{data}}} \log D_\theta(x|y) + \mathop{\mathbb{E}}_{z \sim p_z} \log(1 - D_\theta(G_{\theta'}(z|y))) \right]. \tag{2.35}$$

In a conditional GAN, the conditioning happens through the addition of an extra input to both the generator and the discriminator. The discriminator is provided with "true" labels when it sees an example from p_{data}, and thus learns an internal interpretation of this value. Since the discriminator can use this additional piece of information to base its classification on, the generator must learn to produce examples that realistically correspond to the conditioning data that it receives as input.

Once the GAN is trained, the generator is capable of converting random input into realistic looking data, and can be limitlessly sampled to augment data sets [112–115], perform image super-resolution [116], image-to-image translation [117], and cross-domain tasks, such as pairing shoes with matching handbags [118], and even generating an image, given a text caption [119]. They have additionally been applied to solutions of differential equations involving transport phenomena [120].

2.3.2 Wasserstein GAN

As discussed, generative adversarial networks are implicit density models, meaning that they provide a way to draw samples from probability distribution p_{model}, which we hope through training, will eventually produce results that are indistinguishable from those samples x drawn from the data distribution p_x. We need a way to compare these two distributions so that we can quantify how much they differ and thus construct a loss function to minimize this discrepancy. One standard method to compare distributions is the Kullback-Leibler (KL) divergence.

The KL divergence between two distributions $p(x)$ and $q(x)$ is defined as

$$D_{\text{KL}}(p, q) = \int_{\text{x}} p(x) \log\left(\frac{p(x)}{q(x)}\right) \, \mathrm{d}x. \tag{2.36}$$

Note that since a GAN is an implicit density model, we must operate on samples drawn from the distributions and thus will write these integrals interchangeably as expectation values, since

$$\int_{x} p(x) f(x) \, \mathrm{d}x = \mathbb{E}_{x \sim p(x)}[f(x)].$$

KL divergence is a poor metric to use, however, since it is not symmetric. If $p(x)$ and $q(x)$ are disjoint (as almost certainly p_{model} and p_x are) in some regions

either $q(x)$ will be zero, causing KL divergence to be infinite, or $p(x)$ will be zero causing $q(x)$ to be irrelevant.

Traditional GANs employ the Jensen-Shannon divergence [109]:

$$D_{\mathrm{JS}}(p, q) = \frac{1}{2} D_{\mathrm{KL}}\left(p, \frac{p+q}{2}\right) + \frac{1}{2} D_{\mathrm{KL}}\left(q, \frac{p+q}{2}\right), \tag{2.37}$$

effectively a symmetrized form of the KL divergence. JS divergence is defined (but constant) in disjoint regions where either $p(x)$ or $q(x)$ are zero. When the two distributions overlap, JS divergence goes to zero. The problem with this metric is that GANs produce distributions with very low support (e.g. the multidimensional distributions have very sparse spikes), and in such a case JS divergence provides only a measure of whether the distributions match, but no feedback as to how to improve overlap if they do not match. Take Figure 2.14a and its caption as an example. This discontinuity manifests as unstable training and mode collapse during training.

Thus, researchers proposed [121] using the Wasserstein distance metric in the loss function. This metric, also known as the "earth mover distance" provides a measure of both how far apart (in x) and different in magnitude the two distributions are. Wasserstein distance between two distributions $p(x)$ and $q(x)$ is defined as

$$W(p, q) = \inf_{\gamma \sim \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma} \left\| x - y \right\|. \tag{2.38}$$

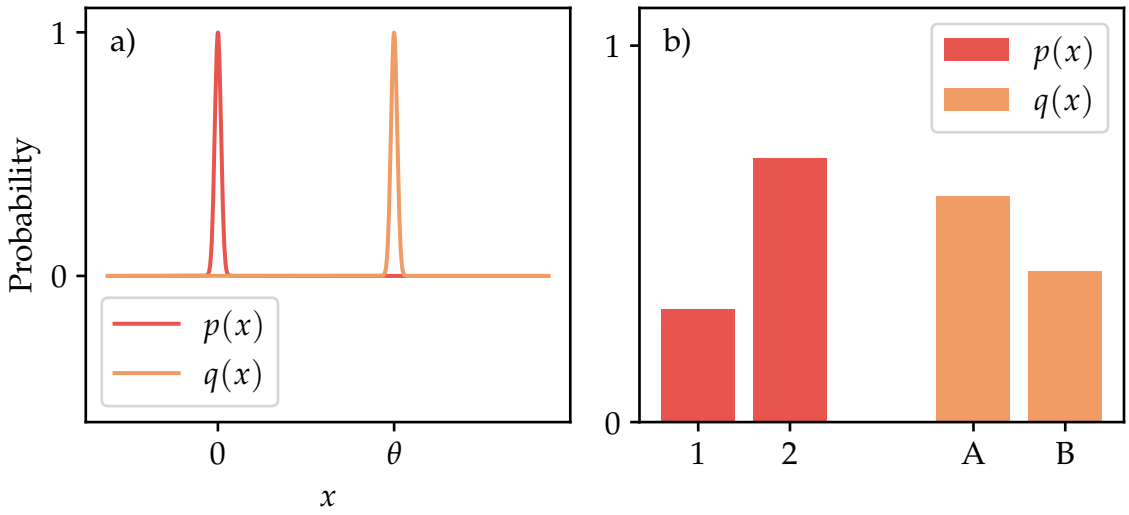


Figure 2.14: a) When the low-support (i.e. negligibly wide) distributions $p(x)$ and $q(x)$ do not overlap, JS divergence is a constant value ($\log(2)$). When the two distributions overlap, JS divergence is zero. For low-support distributions (which are the case with GANs), JS divergence provides a very discontinuous measure of overlap. b) There are an infinite number of ways to move mass from bars A and B to match bars 1 and 2: some of A could be moved to 1, the remaining to 2, and B could be moved to 2; alternatively B could be moved to 1 and the remainder moved to 2 and then A could be used to top up 2. Computing the most efficient way to move this mass (especially in a continuous domain) means the infimum in the Wasserstein distance is intractable.

Thus we look at distributions p and q and come up with a way to move "mass" from function q onto function p so that they match while doing the least possible amount of work (infimum).

Using the same previous example in Figure 2.14a, the Wasserstein distance evaluates to

$$W(p, q) = 1(\|\theta\|) = \theta$$

and thus this improved metric is a function of the distance between the two

distributions as well as their height.

However, this infimum is intractable. Figure 2.14b gives an example as to why; essentially there is an infinite number of possible ways to transform any nontrivial probability mass function into another.

Luckily there exists the Kantorovich-Rubenstein theorem that states

$$\inf_{\gamma \sim \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma} \left\| x - y \right\| = \sup_{\|C\| \leq 1} \left[\mathbb{E}_{x\sim p} C(x) - \mathbb{E}_{y\sim q} C(y) \right]. \tag{2.39}$$

This permits us to deal with the two distributions separately, maximizing the difference between two identical functions evaluated on samples from p and q independently.

Now we have two concerns: firstly, we do not know the function C, but we can modify our discriminator to approximate this function. Now, instead of being called a discriminator, this neural network is referred to as a critic (it is no longer discriminating between real and generated examples). Indeed, now it is just a function approximator so that we can compute the Wasserstein distance. The second concern is that the KR theorem only holds if $\|C\| \leq 1$, that is, if C is 1–Lipschitz continuous. C is 1–Lipschitz continuous if

$$|C(x_1) - C(x_2)| \leq |x_1 - x_2| \quad \forall x_1, x_2 \in \mathbb{R}.$$

The WGAN creators propose that Lipschitz continuity can *kind of*[9] be achieved by clipping the weights of D after each update.

Thus, the objective function for our WGAN will be a minimax game between the critic neural network C and the generator neural network G:

$$\min_{G} \max_{\|C\| \leq 1} \left[\mathop{\mathbb{E}}_{x \sim p_{\text{data}}} [C_\theta(x)] - \mathop{\mathbb{E}}_{z \sim p_z} [C_\theta(G_{\theta'}(z))] \right] \tag{2.40}$$

where θ are parameters of the critic neural network and θ' are parameters of the generator neural network. After each weight update of the discriminator, weight clipping enforces

$$\theta = \min(\max(\theta, -\epsilon), \epsilon),$$

where ϵ is a clipping threshold to enforce Lipschitz continuity.

Since a function is 1-Lipschitz continuous if it has gradients with at most unit norm, another way to encourage Lipschitz continuity is through regularization which penalizes the critic based on the magnitude of its derivative during training. This is discussed more in Supplementary material of Section A.

[9] They themselves state "weight clipping is a clearly terrible way to enforce a Lipschitz constraint." [121]

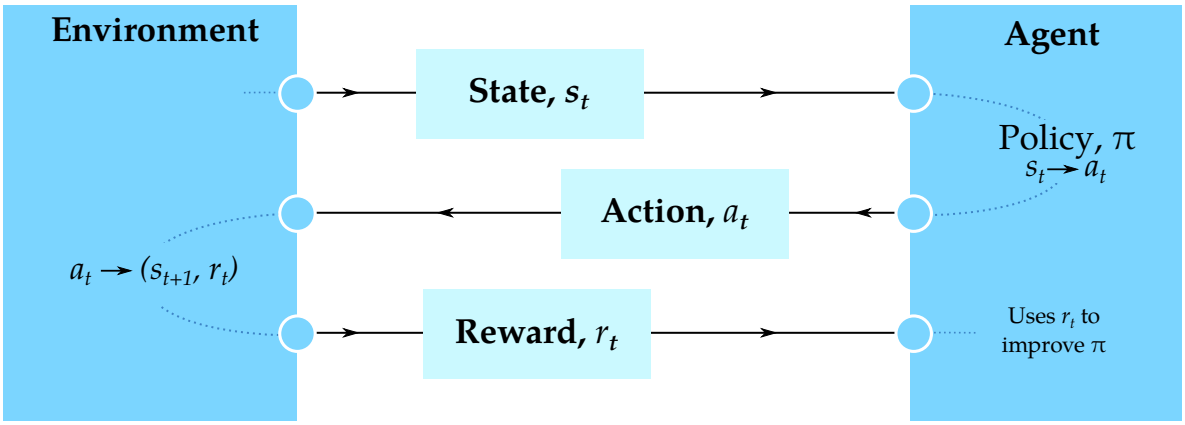


Figure 2.15: The general setup of RL algorithms. An environment emits a state observation s_t. The agent processes this observation with its policy π, producing a choice of action, a_t. The environment evolves by taking this action, ending up in state s_{t+1}. Additionally, some associated reward is provided to the agent. The agent can then use this to improve its policy.

2.4 Reinforcement Learning

For most of its history, the field of reinforcement learning (RL) has evolved separately from machine learning and neural networks. In reinforcement learning, an entity, which we will call an agent, residing in state s_t at time t, learns to take an action a_t that maximizes a cumulative reward signal R by dynamically interacting with an environment [122]. Through the training process, the agent arrives at a policy π that depends on some observation (or "state") of the system. Figure 2.15 shows the general flow of reinforcement learning algorithms.

In the case of simple tasks, π could simply be the action of choosing the best option from a lookup table of all possible states, with corresponding ideal ac-

tions associated with each state. Learning how to fill in this lookup table from experience interacting with the environment is the process of reinforcement learning (specifically Q learning [105]).

In recent years, deep neural networks have taken over as the *de facto* function approximator for the policy function, and thus the field of deep reinforcement learning was born. Deep RL has seen unprecedented success, achieving superhuman performance in a variety of video games [14, 123–125], board games [15, 19, 126], and other puzzles [127, 128].

Many different reinforcement learning algorithms exist, and the first distinction to be made between classes of algorithms is whether an algorithm is **model-based** or **model-free**. In model-based RL, the agent either learns [129, 130] or is provided a model [19] of state transitions. As such, the agent has access to information about which state s_{t+1} it will arrive in if it takes action a_t from state s_t. This is commonly used for long term planning [19].

The opposite of this, model-free RL, employs no model about state transition probabilities and attempts to directly optimize for reward. In this case, there are several approaches as to what quantity to optimize. **Policy gradient** methods optimize the policy π_θ directly using gradients of an objective function $J(\pi_\theta)$. They are known as **on-policy** methods because they only use samples generated from the current policy (e.g. the current set of weights) to perform weight updates. Asynchronous Advantage Actor Critic [131], Proximal Policy

Optimization [132], and Trust Region Policy Optimization [133], are examples of policy gradient methods.

An alternative approach is to focus on learning an approximation $Q_\theta(s, a)$ for the action-value function. As the ideal action-value function is usually denoted as Q^*, this approach is known as **Q-learning**. When a deep neural network is used as the approximator Q_θ, it is referred to as a Deep Q Network [14]. In Q-learning, weight updates to the Q function are performed **off-policy**, making the method very sample-efficient since a buffer of past samples can be maintained and reused.

Policy gradient methods are typically more reliable and stable in training, given that they directly optimize the policy function. Q-learning is less stable, but far more sample efficient given its off-policy nature. A drawback of Q-learning is that it is only defined for discrete action spaces, and it is incapable of (without modification) handling continuous action spaces. It turns out that combining both of these approaches leads to very good performance, and some state-of-the-art algorithms do just this, such as Deep Deterministic Policy Gradient [134] (effectively Q-learning for continuous action spaces) and Soft Actor-Critic [135–137].

2.4.1 Proximal Policy Optimization

I will introduce, in detail, one algorithm, proximal policy optimization (PPO) [132] since it is used in a work in this dissertation.

First we define our policy $\pi(a_t \mid s_t)$ as the likelihood that the agent will take action a_t while in state s_t; through training, the desire is that the best choice of action will become the most probable. To choose an action, this distribution can be sampled. With deep RL, it is common to use a neural network (parameterized by weights θ) to represent the policy, and thus we denote the policy with a subscript θ. By assuming that $\pi_\theta(a_t \mid s_t)$ is a normal distribution and interpreting the output nodes of the neural network as the mean, μ, and variance, σ^2, we can then input the state into the neural network, obtain the moments of the normal distribution, and sample this distribution to obtain an action.

We define a function $Q_{\pi_\theta}(s_t, a_t)$ as the expected future discounted reward if the agent takes action a_t at time t and then follows policy π_θ for the remainder of the episode. We additionally define a value function $V_{\pi_\theta}(s_t)$ as the expected future discounted reward starting from state s_t and following the current policy π_θ until the end of the episode. We introduce the concept of *advantage*, $\hat{A}_t(s_t, a_t)$, as the difference between these two quantities. Q_{π_θ} and V_{π_θ} are not known and must be approximated. We assume the features necessary to represent π are generally similar to the features necessary to estimate the value

function, and thus we can use the same neural network to predict the value function by merely having it output a third quantity.

To train the network, we need to define a cost function. Since A_t can be thought of as "how much better than expected it actually was to take action a_t", and $\pi_\theta(a_t|s_t)$ as "how likely it is to take action a_t", it makes sense to couple these together; good actions should be more probable, and bad actions should be less probable. We construct the typical policy gradient cost function by coupling the advantage of a state–action pair with the probability of the action being taken,

$$L^{\mathrm{PG}}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \right],$$

which we want to maximize by modifying the weights θ through the training process. It is, however, more efficient to maximize the improvement ratio r_t of the current policy over a policy from a previous iteration $\pi_{\theta_{\mathrm{old}}}$ [133, 138]:

$$L^{\mathrm{TRPO}}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)} \hat{A}_t \right] \equiv \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right].$$

Note, however, that maximizing this quantity can be trivially achieved by making the new policy drastically different from the old policy, which is not the desired behaviour. The PPO algorithm [132] deals with this by considering only the minimum between the improvement and a clipped version of the improve-

ment

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right].$$

To train the value function estimator, a squared error is used, that is,

$$L^{\text{VF}}(\theta) = \hat{\mathbb{E}}_t[(V_{\pi_\theta}(s_t) - V_t^{\text{targ}})^2],$$

and to encourage exploration, an entropic regularization functional S is used. This all amounts to a three-term cost function

$$L^{\text{PPO}}(\theta) = \hat{\mathbb{E}}_t \left[L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t) \right],$$

where c_1 and c_2 are hyperparameters.

At the core of reinforcement learning is the concept of reward engineering, that is, developing a reward scheme to inject a notion of success into the system. When discussing the reward scheme of a problem there is a distinction to be made between tasks that provide frequent feedback through the reward, and tasks where reward is delayed. A problem can be one of sparse reward, that is, not every action corresponds directly to a reward. Commonly in problems like these, reward is provided only at the termination of an episode, or infrequently throughout. Since the only feedback that an agent gets as to its performance is the reward, sparse reward schemes are thus considerably more difficult to

learn. This is known as the credit assignment problem (CAP).

Sometimes reinforcement learning practitioners create auxiliary reward schemes to provide frequent feedback to assist in the CAP for tasks with severely-delayed reward, however, this can sometimes have surprising results with agents learning to exploit these hand-tailored reward schemes, informally called "reward hacking".

Chapter 3

Supervised learning

3.1 The importance of training data

Any machine learning researcher will attest to the importance of training data. So-called “big-data” approaches like deep neural networks often ingest huge amounts of data, fitting complicated models in an attempt to generalize to new data. “Real-world” data sets are limited in the amounts and variety of data that can be collected, perhaps by government regulation, or the mere fact that there is not an easy way to acquire millions of images of fish, giraffes, or capybaras. In computational physics, abundance of data is less of an issue; it is usually, at least in theory, possible to *generate* an arbitrarily large amount of data using traditional mathematical tools. Of course, this could possibly mean incurring great computational expense, but the limitations are not “physical”.

As such, it is easy to generate a large amount of data, and train a model that, at first glance, performs extremely well. In “Deep neural networks for direct, featureless learning through observation", we learn that the careful sam-

pling of data is important to train a robust model, and that a high accuracy is not necessarily a good indication of acceptable generalization.

In addition to this, we show that a neural network can indeed be used to fit an energy function to sufficient accuracy to reproduce physical phenomena in simulation and capture the dynamics of a physical system: the ferromagnetic Ising model. This model is ubiquitous in physics because of its simplicity. It is well-studied, the mapping from configuration to label (energy) is easy and quick to compute, and interesting phenomena occurs (phase transition) during a physical simulation. The fact that the mapping is easy to compute means machine learning is entirely unnecessary for this problem; it is definitely easier (and less expensive computationally) to just compute the energy using the closed-form solution. This is, however, precisely what makes this model a good candidate for the first foray into machine learning.

Section B.1 discusses the sampling methods used in this work in more detail.

Deep neural networks for direct, featureless learning through observation: The case of two-dimensional spin models

Kyle Mills*
Department of Physics, University of Ontario Institute of Technology, Oshawa, Ontario, Canada

Isaac Tamblyn†
Department of Physics, University of Ontario Institute of Technology, Oshawa, Ontario, Canada
and Department of Physics, University of Ottawa & National Research Council of Canada, Ottawa, Ontario, Canada

(Received 23 June 2017; revised manuscript received 31 January 2018; published 16 March 2018)

We demonstrate the capability of a convolutional deep neural network in predicting the nearest-neighbor energy of the 4×4 Ising model. Using its success at this task, we motivate the study of the larger 8×8 Ising model, showing that the deep neural network can learn the nearest-neighbor Ising Hamiltonian after only seeing a vanishingly small fraction of configuration space. Additionally, we show that the neural network has learned both the energy and magnetization operators with sufficient accuracy to replicate the low-temperature Ising phase transition. We then demonstrate the ability of the neural network to learn other spin models, teaching the convolutional deep neural network to accurately predict the long-range interaction of a screened Coulomb Hamiltonian, a sinusoidally attenuated screened Coulomb Hamiltonian, and a modified Potts model Hamiltonian. In the case of the long-range interaction, we demonstrate the ability of the neural network to recover the phase transition with equivalent accuracy to the numerically exact method. Furthermore, in the case of the long-range interaction, the benefits of the neural network become apparent; it is able to make predictions with a high degree of accuracy, and do so 1600 times faster than a CUDA-optimized exact calculation. Additionally, we demonstrate how the neural network succeeds at these tasks by looking at the weights learned in a simplified demonstration.

DOI: 10.1103/PhysRevE.97.032119

I. INTRODUCTION

The collective behavior of interacting particles, whether electrons, atoms, or magnetic moments, is the core of condensed matter physics. The difficulties associated with modeling such systems arise due to the enormous number of free parameters defining a near-infinite configuration space for systems of many particles. In these situations, where exact treatment is impossible, machine-learning methods have been used to build better approximations and gain useful insight. This includes the areas of dynamical mean-field theory, strongly correlated materials, phase classification, and materials exploration and design [1–8].

As an introductory many-particle system, one is commonly presented with the square two-dimensional Ising model, a ubiquitous example of a ferromagnetic system of particles. The model consists of an $L \times L$ grid of discrete interacting "particles" that either possess a spin-up ($\sigma = 1$) or spin-down ($\sigma = -1$) moment. The internal energy associated with a given configuration of spins is given by the Hamiltonian $\hat{H} = -J \sum \sigma_i \sigma_j$, where the sum is computed over all nearest-neighbor pairs ($\langle i, j \rangle$), and J is the interaction strength. For $J > 0$, the system behaves ferromagnetically; there is an energetic cost of having opposing neighboring spins, and neighboring aligned spins are energetically favorable.

The canonical Ising model defined on an infinite domain (i.e., periodic boundary conditions) is an example of a simple system that exhibits a well-understood continuous phase transition at a critical temperature $T_c \approx 2.269$. At temperatures below the critical temperature, the system exhibits highly ordered behavior, with most of the spins in the system aligned. Above the critical temperature, the system exhibits disorder, with, on average, roughly equivalent numbers of spin-up and spin-down particles. The "disorder" in the system can be represented by an order parameter known as the "magnetization" M, which is merely the average of all L^2 individual spins.

Configurations of the Ising model can be thought to belong to one of two phases. Artificial neural networks have been shown to differentiate between the phases [9,10], effectively discovering phase transitions. This is, however, merely a binary classification problem based on the magnetization order parameter. The membership of a configuration to either the high- or low-energy class does not depend upon any interaction between particles within the configuration. Furthermore, convolutional neural networks have been trained to estimate critical parameters of Ising systems [11]. Machine-learning methods have been demonstrated previously in many-body physics applications [6] and other two-dimensional topological models are discussed frequently [12,13] in quantum field theory research. However, the use of deep convolutional neural networks remains infrequent, despite their recently presented parallels to renormalization group [14] and their frequent successes in difficult machine-learning and computer-vision tasks, some occurring a decade ahead of expectations [15,16].

*kyle.mills@uoit.net
†isaac.tamblyn@nrc.ca

We demonstrate that a convolutional deep neural network, trained on a sufficient number of examples, can take the place of conventional operators for a variety of spin models. Traditional machine-learning techniques depend upon the selection of a set of descriptors (features) [17]. Convolutional deep neural networks have the ability to establish a set of relevant features without human intervention, by exploiting the spatial structure of input data (e.g., images, arrays). Without human bias, they detect and optimize a set of features, and ultimately map this feature set to a quantity of interest. For this reason, we choose to call deep convolutional neural networks "featureless learning." We take a more in-depth look into this process in the section titled "A closer look at the convolutional kernels" later in this work. Furthermore, we demonstrate that a neural network trained in this way can be practically applied in a simulation to accurately compute the temperature dependence of the heat capacity. In doing so, we observe its peak at the critical temperature, a well-understood [18] indication of the Ising phase transition. Additionally, we investigate the effectiveness of the deep learning approach on three additional spin model Hamiltonians.

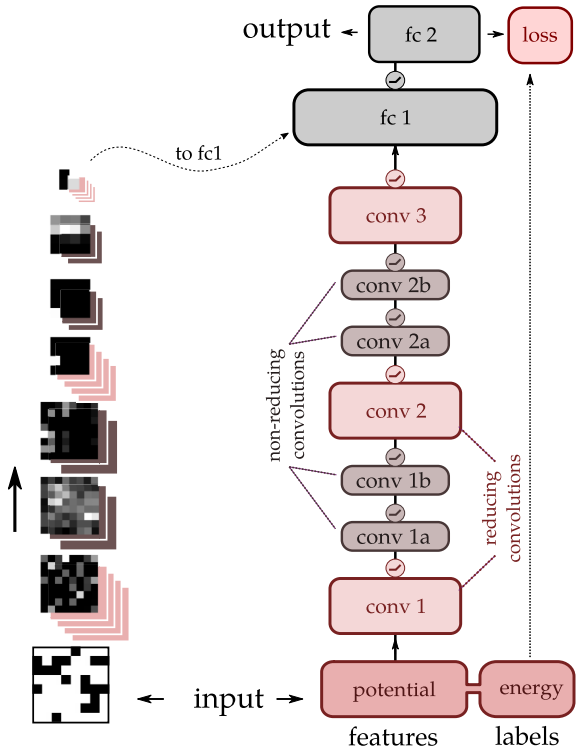


FIG. 1. The deep neural network architecture used for predicting spin model operators. The network consists of 7 convolutional layers with two fully connected layers at the top. On the left, the output of a given filter path is shown for an example spin configuration. The final 2×2 output is fed into a wide fully connected layer. ReLU (rectified linear unit) activation is used on all convolutional layers.

II. METHODS

A. Deep learning

We used a very simple deep neural network architecture, shown in Fig. 1. In previous work, we demonstrated that the same neural network structure, differing only in the number of repeat units, was capable of predicting quantities associated with the one-electron Schrödinger equation [19]. In this network, the convolutional layers work by successively mapping an image into a feature space where interpolation is possible using a linear boundary. The final decision layers impose this boundary and produce an output value. Other methods can be trained in a featureless learning fashion, such as kernel ridge regression and random forests. We compare our approach to these methods in "DNN versus other methods" below.

Our neural network consists of 7 subsequent convolutional layers. We use two different types of convolutional layers, which we call "reducing" and "nonreducing."

The 3 reducing layers operate with filter (kernel) sizes of 3×3 pixels. Each reducing layer operates with 32 filters and a stride of 2×2, effectively reducing the data resolution by a factor of two at each step. In between each pair of these reducing convolutional layers, we have inserted two convolutional layers (for a total of 4), which operate with 16 filters of size 4×4. These filters have unit stride and therefore preserve the resolution of the image. The purpose of these layers is to add additional trainable parameters to the network, and we have previously [19] found that 2 was a good balance between speed and accuracy. All convolutional layers have ReLU (rectified linear unit) activation.

The final convolutional layer is fed into a fully connected layer of width 1024, also with ReLU activation. This layer feeds into a final fully connected layer of size 15. This output can be interpreted as a vector of the probability of membership to each energy class. For the larger 8×8 configurations, there are 63 possible energy values, and therefore the final fully connected layer is modified to have a width of 63. This output is used to compute the softmax cross-entropy loss.

To train the models, we minimized this loss function using the AdaDelta [20] optimization scheme, with a global learning rate of 0.001. We monitored the loss function as training proceeded and terminated training after the loss function appeared to converge sufficiently.

Training of the models was carried out on a custom-built computer housing multiple graphical processing units. We used a custom TensorFlow [21] implementation to make use of the GPUs in parallel. We placed a complete copy of the neural network on each GPU, so that each can compute a forward and back-propagation iteration on one full batch of images. Our effective batch size was 800 images per iteration. After each iteration, the GPUs share their independently computed gradients and the optimizer proceeds to adjust the parameters in the direction that minimizes the loss function.

The series of convolutional layers in this network is designed to operate on images of size 16×16. For this reason, when training the 4×4 Ising model, we perform lossless upscaling by repeating each row and column 4 times to achieve a compatible input size. With the 8×8 configurations, we upscale by a factor of 2 to obtain the same input size. This does not notably affect the performance of the models, and it permits the use of the same network architecture for both sizes. In practice, one could use a layout similar to that suggested in

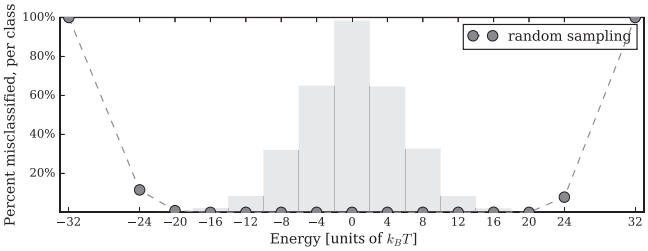


FIG. 2. The overall accuracy for the model trained on randomly sampled data is 99.88%; however, the neural network misclassifies the majority of high- and low-energy configurations. With larger Ising models, this effect would be more significant as the central degeneracy is greater.

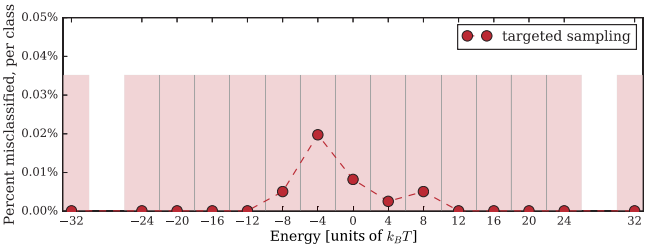


FIG. 3. When trained on an even distribution of energies (targeted sampling dataset) the deep neural network was able to classify all but a handful of configurations. The misclassified configurations appear central to the energy distribution as there is more variation in this region.

Ref. [22] to accommodate arbitrarily sized Ising model grids, including grids differing in size to those in the training set.

III. DATASETS

Two-dimensional spin-model configurations (with the exception of the Potts model) can be represented as binary-valued arrays, with each element having a value of either $\sigma = -1$ (spin-down) or $\sigma = 1$ (spin-up). As such, a simple method to generate an arbitrarily large amount of training data is to randomly draw the state of each spin, with uniform probability of it being -1 and 1. This method, while trivially easy to implement, results in an energy distribution centered sharply around zero (histogram of Fig. 2), since the central energy levels of the Ising model are highly degenerate. There is very little probability of generating a high-energy ("checkerboard") or low-energy ("solid") configuration. This will be problematic to the application of the deep learning model, as the neural network will not have been exposed to features present in the high- and low-energy configurations during training. We initially trained the neural network naively on approximately 12 500 randomly generated training examples. The training dataset contains only 10 343 of the 65 536 possible configurations (16%).

We evaluated this model on the complete set of 65 536 4×4 configurations and it achieved an accuracy of 99.88% (99.88% of the configurations were classified correctly). While this appears to be excellent performance, closer inspection reveals that many configurations with energies below $-16J$, and above $16J$, are misclassified, as shown in Fig. 2. This problem would greatly affect a Monte Carlo simulation replicating the low-temperature phase transition, as this phenomenon is dependent upon the correct evaluation of low-energy configurations.

The motivation for a more intelligent sampling method is clear; a more even distribution of energies is necessary if one wishes to accurately predict the low- and high-energy configurations. We implemented a modified form of umbrella sampling, which we have named "targeted sampling" (TS) in an attempt to achieve a more even distribution of energies. This sampling method resembles the Metropolis-Hastings algorithm in structure, but instead of seeking low-energy states, we "target" specific energies, accepting configurations that move us toward the target energy, and rejecting ones (with a Gaussian probability) that lead us away. In this way, we can collect examples across the energy spectrum in an intelligent way, achieving a very even distribution of energies, as seen in the histogram of Fig. 3.

IV. RESULTS

A. The 4 × 4 Ising model

We begin our investigation with the 4×4 Ising model, as the configuration space is of a manageable size that we can easily compute *all possible configurations* (65 536 total unique configurations). Because of this, we can explicitly evaluate how well a model performs by evaluating the model on the entirety of configuration space. The energies of these configurations are discrete, taking on 15 possible values (for the 4×4 model) ranging from $-32J$ to $32J$. The discrete energy values allow us to treat energy prediction as a machine-learning "classification problem." In the areas of handwriting recognition and image classification, deep convolutional neural networks with such an output structure have excelled time and time again [23–26]. The value of J can be any constant, as the input and output of the neural network can be scaled appropriately. In this work we use $J = 1$.

We generated three independent datasets, each consisting of 27 000 training examples (1800 examples per class), gathered using the targeted sampling approach. In Fig. 4, we show the classification accuracy of the deep neural network with respect

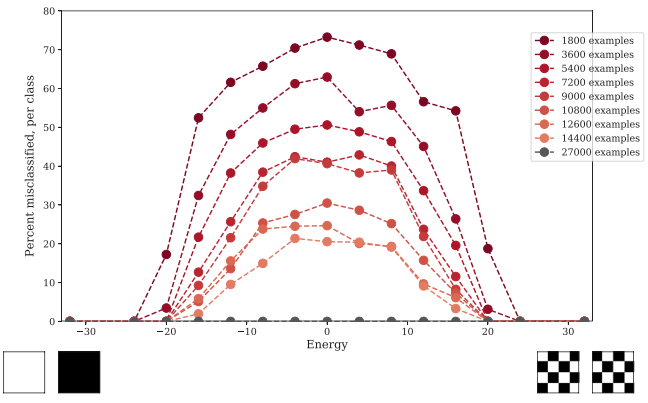


FIG. 4. We investigate how the classification accuracy of the neural network depends on the number of training examples. Since 27 000 training examples resulted in almost perfect accuracy, we chose it as the number, giving rise to 1800 configurations per class.

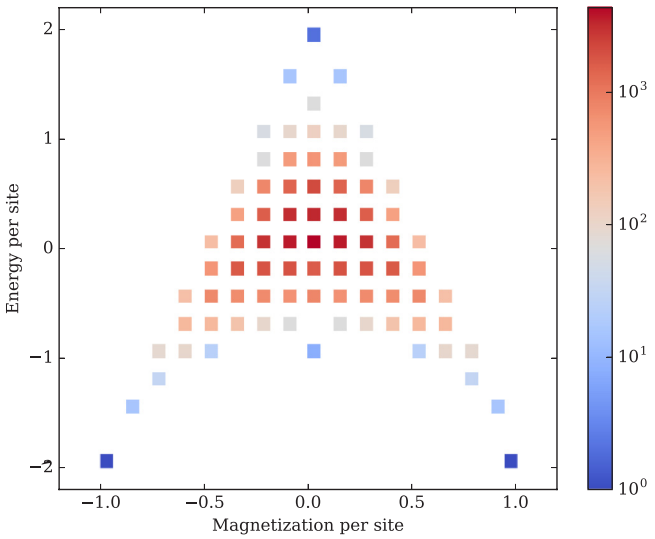


FIG. 5. Configuration space of the 4×4 Ising model. The colors represent the density of states.

to the number of example configurations provided during training. Exceptional accuracy is achieved at 27 000 training examples (1800 per class). Each dataset contains less than 20% of configuration space (some energy classes are over-sampled to fill the 1800-example quota). We trained our neural network architecture on each of these three datasets. The neural network was able to classify all but a handful of Ising configurations, on average. On one dataset, it achieved an accuracy of 100%. In all cases of misclassification, 100% of misclassified examples only have an error of ± 1 energy level, indicating the neural network is just barely failing to classify such examples. All misclassified configurations had energies near zero. In this region there is considerable variation due to the degeneracy of the Ising model (apparent in Fig. 5), and therefore predictions based on a uniform number of training examples per class are slightly more challenging. At the extreme energies (± 32), individual configurations are repeated many times to fill the quota of training examples. It is worth noting again that this neural network had access to less than 20% of configuration space, so it is clearly correctly inferring information about examples it has not yet seen.

1. The 8×8 *Ising model*

Although the 4×4 model is instructive, larger Ising models such as the 8×8 model are interesting since the enormity of configuration space ($2^{8^2} \approx 10^{19}$) precludes training on even a modest fraction of possible configurations, so the neural network truly needs to "learn" how to predict energies from seeing only a minuscule fraction of configuration space.

We performed a convergence study to determine the optimal number training examples. At 100 000, the neural network is able to classify 8×8 Ising configurations into their 63 discrete energy levels with 99.922% accuracy as shown in Fig. 6. Note that we can no longer report an accuracy computed over the entirety of configuration space, so we must report the accuracy of the model on a separate testing set of data.

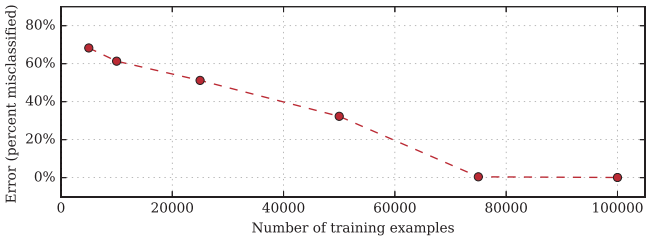


FIG. 6. More training examples unsurprisingly leads to better performance. At 100 000, the neural network is able to classify 8×8 Ising configurations into their 63 discrete energy levels with 99.922% accuracy. This represents a vanishingly small subset of configuration space, the entirety of which consists of 2^{64} configurations.

The testing dataset consists of 50 000 examples separated from the training dataset prior to training. No examples in the testing set appear in the training set. This is ensured by separating examples into testing and training based on their SHA3 hash (see Appendix). Importantly, as with the 4×4 model, in the few cases where the model did fail, it did not fail by very much: 100% of the time, the predicted class is either correct or only one energy class away from correct. The neural network does exceptionally well at predicting energies when only exposed to a small subset of configuration space during training.

2. Regression

In practice, a deep neural network capable of classifying configurations into well-defined bins is less appealing than one which could predict continuous variables. "Real-life" systems rarely exhibit observables and characteristics that are quantized at the scales relevant to the macroscopic problem. As such, this is a good opportunity to investigate a form of deep neural network output structure known as "regression." In a regression network, instead of the final fully connected layer having a width equal to the number of classes, we use a fully connected layer of width 1: a single output value. In this case, a softmax cross-entropy loss layer is no longer appropriate. The simplest form of loss function for a single-output regression network is the mean-squared error between network predictions and the true value of the energy.

We modify the deep neural network in this way to perform regression. Changing nothing about the training process other than the loss function, we see that the model performs quite well, with a median absolute error of $1.782J$. With the Ising model, the allowed energy classes are separated by 4 energy units, so an error of $1.782J$ is consistent with the capability of the network to accurately classify examples into these bins of width 4. Additionally, we trained the deep neural network to learn the magnetization; it performs exceptionally well with a median absolute error of 4×10^{-3} per spin and 2×10^{-3} per spin for the 4×4 and 8×8 models, respectively. This is not particularly surprising as the magnetization is a very simple, noninteracting operator. This effectively amounts to using a convolutional neural network to compute the sum of an array; we present it merely as a demonstration of a neural network's ability to learn multiple mappings.

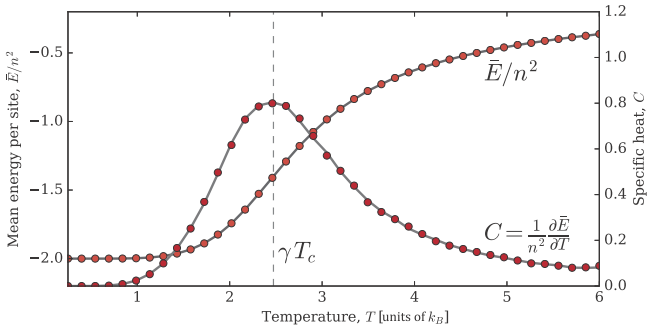


FIG. 7. The average energy per site at various temperatures, as well as the heat capacity, C for the 4×4 Ising model. The solid lines and dots indicate the energy evaluation methods used: the exact Hamiltonian, and DNN, respectively. These results are averaged over 400 independent simulations. The standard deviation is negligibly small.

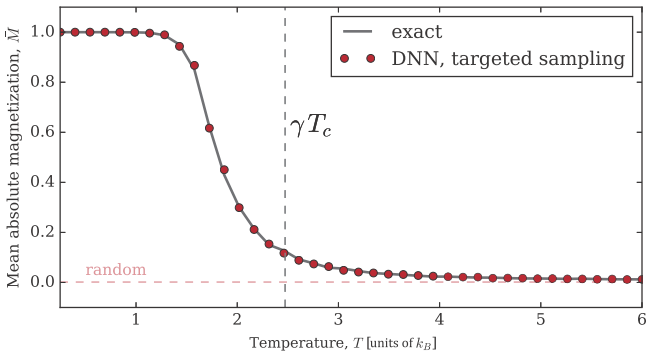


FIG. 8. The magnetization per site at various temperatures for the 4×4 Ising model. The solid line denotes the simulation using the exact magnetization and energy operators, and the dots represent the deep-learned energy and magnetization operators. These results are averaged over 400 independent simulations. The standard deviation is negligibly small. The horizontal dashed line indicates the mean absolute magnetization of a purely random distribution of spins (the entropy-dominating high-temperature limit), which is close to, but not equal to zero.

3. *Replicating the Ising Model phase transition*

The Ising model defined on an infinite domain exhibits a phase transition at a critical temperature $T_c \approx 2.269$. For a finite domain under periodic boundary conditions, however, a correction factor γ is necessary to compensate for the correlations between periodic lattice images. This behavior is discussed in detail in Ferdinand and Fisher's 1969 work (Ref. [18]), and in this analysis we will denote the "theoretical" critical temperature as γT_c.

Using a Metropolis-Hastings algorithm, one can sample the configuration space accessible to the Ising model at a given temperature. Using a Boltzmann rejection probability, the mean energy per site, $\bar{E}$, can be computed for a given temperature. Repeating for various temperatures allows one to plot $\bar{E}$ against T and observe the phase transition. The Metropolis-Hastings algorithm, and thus the demonstration of this phase transition, depends on the accurate evaluation of Ising configuration energies. As with any mathematical model, the ultimate test is its ability to make predictions of sufficient quality that one can recover a physically realistic phenomena.

We generated the phase diagram for the 4×4 Ising model, evaluating the energy using the exact Hamiltonian. Then, we replaced the magnetization and energy operators with the trained deep neural networks. The phase diagrams match exactly and are presented in Figs. 7 and 8.

We repeated this exercise with both the 8×8 classification and regression models, and observe the phase transition. As Fig. 9 shows, in the case of classification, the deep neural network is able to learn the energy and magnetization operators with sufficient precision to replicate the phase transition. In the case of regression, the phase transition is still observed, however, at a slightly lower temperature. This is not completely surprising, as the classification method effectively snaps any slightly incorrect predictions to the nearest correct value.

4. *Long-range interactions*

The extent of the traditional Ising Hamiltonian is very short-range, including only nearest-neighbor interactions. Physical systems frequently depend on long-range interactions. Herein, we demonstrate that the same deep neural network is able to learn the energies associated with two long-range interactions. First, we demonstrate the screened Coulomb Hamiltonian: the traditional pairwise Coulomb interaction attenuated by an exponential term [27,28]. We computed this energy for 120 000 8×8 Ising configurations using an explicit sum method and periodic boundary conditions, ensuring the infinite summation was converged sufficiently for the effective cutoff we used of 64 units, i.e., 8 times the size of the unit lattice. The summation is very computationally expensive, as it must be computed for every pair of spins between the unit lattice and all periodic images until the effective cutoff radius is reached and the sum converges. Since the algorithm is amenable to parallelization, we implemented it in CUDA for performance

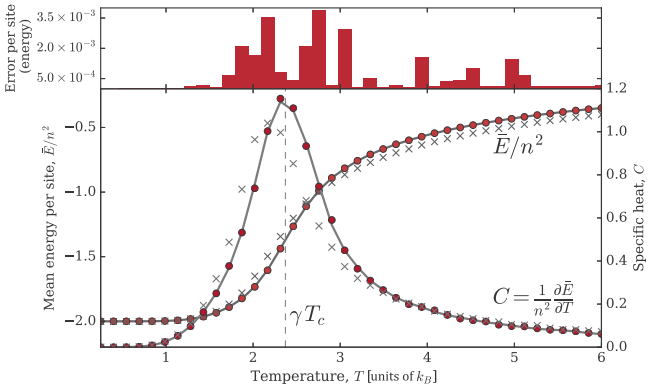


FIG. 9. The average energy per site at various temperatures, as well as the heat capacity for the 8×8 Ising model. The solid line denotes the simulation using the analytic Hamiltonian, the dots represent the deep-learned Hamiltonian with classification, and the crosses represent the deep-learned Hamiltonian with regression. The absolute difference between the per-site-energies obtained from the analytic Hamiltonian and deep-learned classification model are plotted above. These results are averaged over 400 independent simulations. The standard deviation is negligibly small.

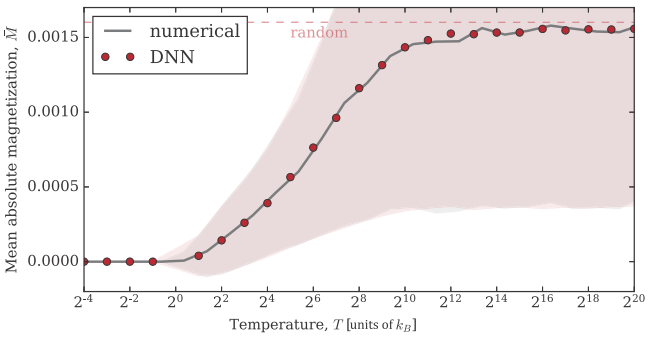


FIG. 10. The average magnetization per site at various temperatures for the (antiferromagnetic) long range screened Coulomb Hamiltonian. The solid line denotes the simulation using the numerical energy operator, and the dots represent the simulation run with the deep-learned energy operator. The dashed line labeled "random" denotes the mean absolute spin of a purely random distribution of spins, the entropy-dominating theoretical limit for infinite temperature. The filled area represents one standard deviation of the mean.

[29]. We trained our deep neural network to perform regression on a set of 100 000 examples, and tested the network on a nonintersecting set of 20 000 examples. Our neural network is able to learn this long-range Hamiltonian with considerable accuracy, performing with a median absolute error of $0.640J$ energy units. The performance of the model is shown in Fig. 11(a).

Furthermore, we repeated the Metropolis-Hastings simulation and discovered a phase transition as the temperature increases. We then tuned our neural network to this "thermally sampled" data and repeated the simulation. The results are plotted in Fig. 10. Since this Hamiltonian represents an antiferromagnetic interaction, the most energetically stable configuration is the "checkerboard" configuration, in contrast to the ferromagnetic Ising model. At (very) high temperature, the mean absolute magnetization approaches a value consistent with a purely randomly drawn set of spins. Again our model performs well, matching the numerical simulation well.

Second, to demonstrate the applicability of such a deep neural network architecture to arbitrary long range interactions, we modified the screened Coulomb Hamiltonian to have a sinusoidal dependence in r, e.g.,

$$\hat{H} = J \sum_{\{i,j\}}^{r_{\text{cut}}} \sigma_i \sigma_j \frac{e^{-k r_{ij}}}{r_{ij}} \sin(r_{ij}), \tag{1}$$

where the summation, like in the screened Coulomb Hamiltonian, is carried out over all combinations of spins between the configuration and all neighbouring periodic images out to a radius of r_{cut}. This is, intentionally, a completely *arbitrary* modification to the Hamiltonian made to demonstrate the wide generalizability of the deep neural network approach. Following an identical training procedure as the screened Coulomb Hamiltonian, the deep neural network was able to make predictions with an accuracy of $0.253J$ energy units. The performance is plotted in Fig. 11(b).

While the accurate learning of the long-range interactions is in itself impressive, additionally the deep neural network drastically outperforms the explicit calculations in terms of speed. The deep neural network can make predictions at a rate 1600 times faster than the CUDA-enabled "exact" calculation (performing at comparable median error), when running on a single NVIDIA Tesla K40.

5. A modified Potts model

Our approach is not limited to discrete spin values. The planar Potts model is a generalized form of the Ising model wherein the set of possible spin states is expanded to include more than just binary spin-up and spin-down states [30]. The Hamiltonian is given by

$$\hat{H} = J \sum_{\{i,j\}} \cos(\sigma_i - \sigma_j). \tag{2}$$

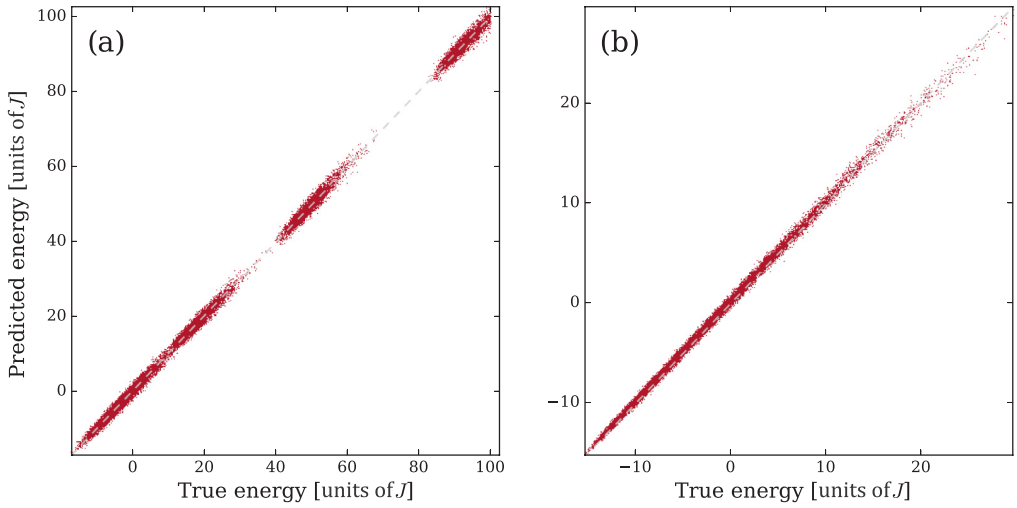


FIG. 11. The deep neural network is able to learn arbitrary long-range interactions with high accuracy. Here we plot the DNN-predicted (a) screened Coulomb energy, and (b) sinusoidal screened Coulomb energy against the explicitly calculated energy for the 20 000 examples in the test set. The training and test set are randomly sampled.

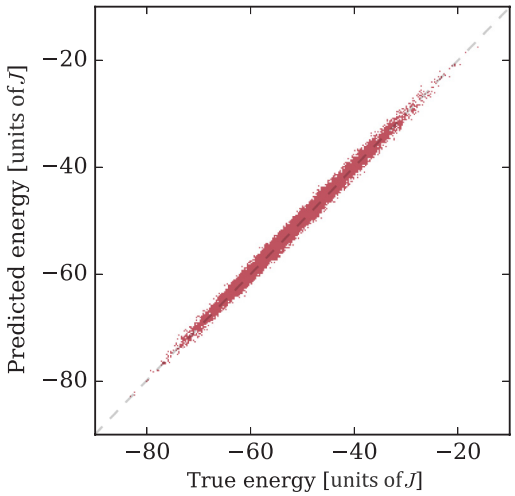


FIG. 12. The deep neural network is able to learn the nonbinary-state Potts Hamiltonian with a MAE of $0.542J$ on a randomly sampled dataset.

In his analysis, Potts [31] used discrete spin values. We train a regression model to compute the energy of Eq. (2) using a continuum of spins randomly generated on the interval $[0,\pi)$. Our neural network is able to learn the mapping after observing 500 000 examples with a MAE of $0.542J$. In Fig. 12 we plot predicted energies against the true energies.

6. Disordered Hamiltonians and off-lattice models

Disordered Hamiltonians are designed to model systems where particles do not form a regular lattice. Because of the irregular interatomic spacing, the bonds within these structures experience differing amounts of compression and expansion. "Spin" Hamiltonians attempt to map the disorder in atomic positions (off-lattice) to a regular (on-lattice) model with disordered interactions instead. Treating spin-glass materials with convolutional neural networks should be possible, but encoding the disorder into the Hamiltonian (i.e., creating an on-lattice model) is not, as it introduces spatially dependent operators, a feature inherently (and intentionally) ignored by convolutional neural networks. Rather, one could use an off-lattice model such as those used in Refs. [22,32].

7. DNN versus other methods

A question one may ask is whether deep neural networks are the right tool for the job. Certainly, there are other machine-learning methods that do not involve such an expensive training process as deep neural networks demand. In addition to a deep convolutional neural network, we tried two other commonly used machine-learning algorithms, kernel ridge regression (KRR) and random forests (RF), on various dataset sizes. For the 8×8 regression model, KRR performed at best poorly with a median absolute error of $60.3J$. RF performed much better than KRR with a MAE of $5.8J$ (still far inferior to the deep neural network). Additional machine-learning methods have previously been demonstrated on the Ising model [33], and all present errors significantly greater than we observe with our deep neural network.

8. A closer look at the convolutional kernels

One might question how a convolutional neural network succeeds at learning the energies of Ising model configurations. Convolutional neural networks optimize a set of weights, which when applied to the input in a specific way, result in an output representation of the original image that can then be interpreted by a final "traditional" neural network (i.e., the "decision layer"). As such, the learned weights, or more appropriately named the kernels (convolution kernels are made up of $k \times k$ individual weights) act as "feature detectors."

We illustrate this through demonstration. Consider a very simple convolutional deep neural network: a single convolutional layer with 3×3 kernels, operating on a 9×9 Ising model with stride 3. With the stride equal to the kernel size, each kernel applies to a unique region of the input space, with a given kernel only acting on each input pixel once. We will use 512 kernels for this convolutional layer ($3 \times 3 \times 512$ individual weights). The convolutional layer output is passed through a fully connected layer of size 1024 and then reduced to a single output: the energy prediction.

We train the neural network on 200 000 randomly generated examples (admittedly poor practice normally, but fine for this demonstration). This simple network performs significantly more poorly than the network used throughout this work, but it serves as a good example. After the network has converged, we can look at the optimized convolutional kernels. All 512 kernels are presented in Fig. 13. It is difficult to tell exactly what these detect from looking at the raw weights. The magnitude of the weights are very close to zero, with some being negative (red) and some positive (blue). We can get a better idea of what the weights have adapted to detect by finding an input image that maximizes the output of the respective channel [34]. Using random noise as input, we can compute the gradient of the activated output with respect to the input image and optimize the image to maximize the output (gradient ascent). This will show us the Ising configuration that maximizes the activation of the filter and thus give us an idea of what the filter has learned to detect. Demonstrating this on example filters produces the input images shown in Fig. 14.

In this simple model, the filters learn to activate the resulting output when they see the block that they have adapted to detect. This example only works so cleanly on this very simple neural network. In our production code, the neural network is more complex than this model, having far fewer parameters (16 or 64 per layer instead of 512), so the filters must learn to pick up on only the most relevant features and combinations of possible features. Additional subsequent convolutional layers provide a mechanism for the neural network to mix these features together and provide a hierarchy of feature detection, with early layers detecting more small-scale structures, and later layers picking up combinations of these small-scale features. The final fully connected layer then learns to take this information and map it to the energy or magnetization. Straying from this simple example, the interpretation of the weights becomes more abstract, but the core idea remains the same: individual kernels detect features, which when combined with all of the other kernels provide a mechanism for the neural network to map the input data to a space where interpolation is possible through the final fully connected layers.

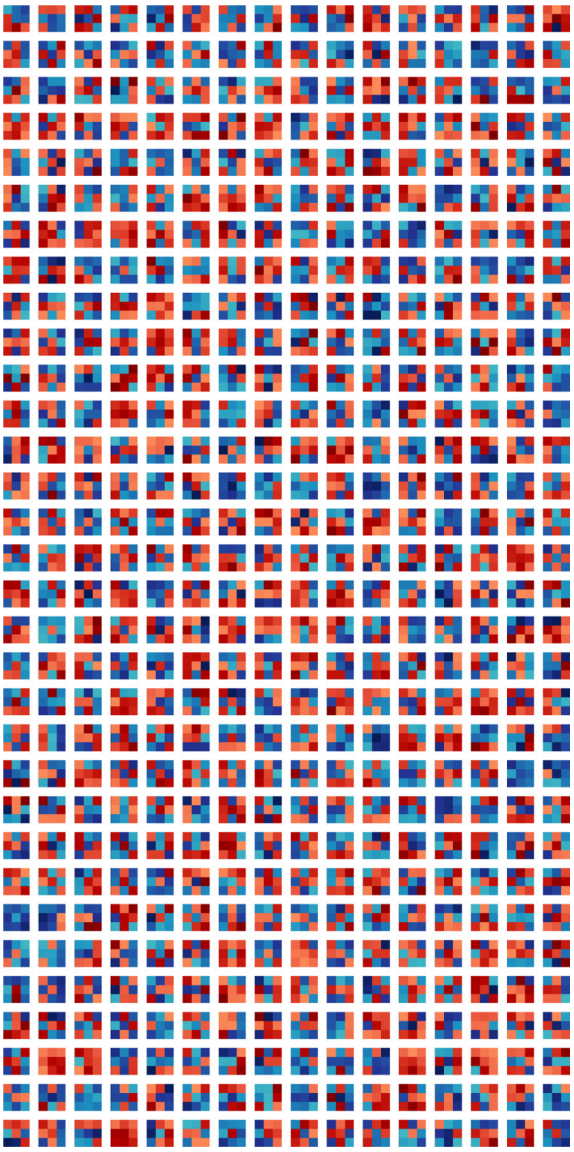


FIG. 13. The 512 optimized filters of our demonstration convolutional neural network. Red weights indicate negative values and blue indicate positive values. The intensity of the color represents the magnitude of the weight.

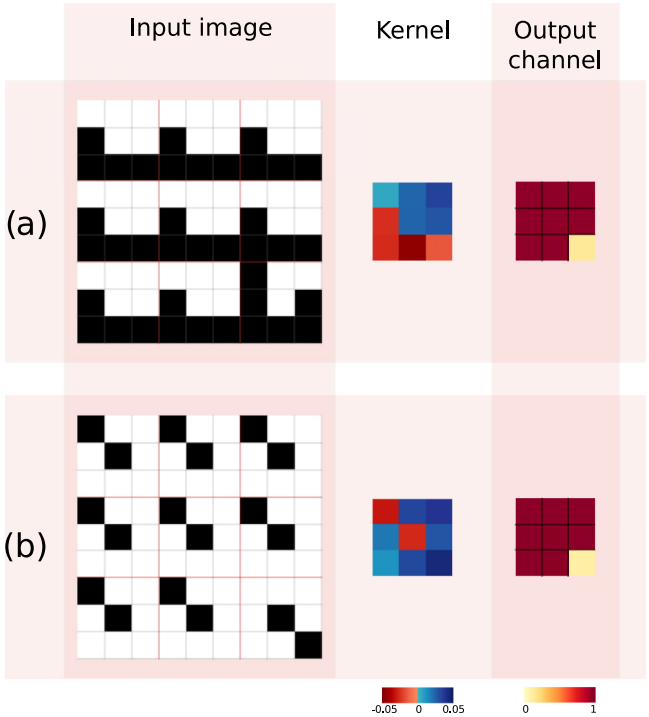


FIG. 14. The input images are optimized to maximize the channel output of the first (a) and last (b) filters of Fig. 13. In both cases, we have manually set the lower right 3×3 block to demonstrate how the output is affected when the filter meets a block it has not adapted to detect.

9. Conclusion

We have trained a deep neural network to accurately classify spin-model configurations based on their energies. Earlier work on the Ising model has focused on the identification of phases or latent parameters through either supervised or unsupervised learning [9,10]. Following this work, we focus on learning the operators directly. Our deep neural network learns to classify configurations based on an interacting Hamiltonian, and it can use this information to make predictions about configurations it has never seen. We demonstrate the ability of a neural network to learn the interacting Hamiltonian operator and the noninteracting magnetization operator on both the 4×4 and the 8×8 Ising model. The performance of the larger 8×8 model demonstrates the ability of the model to generalize its intuition to never-before-seen examples. We demonstrate the ability of the deep neural network in making "physical" predictions by replicating the phase transitions using the trained energy and magnetization operators. To replicate this phase diagram, the deep neural network must use the intuition developed from observing a limited number of configurations, to evaluate configurations it has never before seen. A physical simulation such as this is the ultimate test of a mathematical model. Indeed, it succeeds and is capable of reproducing the phase diagram precisely. We demonstrate the ability of a neural network to accurately predict the screened Coulomb interaction (a long-range interaction) and its phase transition, and we observe a speed up of three orders of magnitude over the CUDA-accelerated explicit summation. We demonstrate the ability of a deep neural network to predict the continuous-valued sinusoidally screened Coulomb Hamiltonian as well as a nonbinary modified Potts Hamiltonian. The rapid development of featureless deep learning implementations and their ongoing successes in the technology sector motivate their consideration for physical and scientific problems.

ACKNOWLEDGMENTS

The authors acknowledge funding from NSERC and SOSCIP. Compute resources were provided by SOSCIP and an NVIDIA Faculty Hardware Grant.

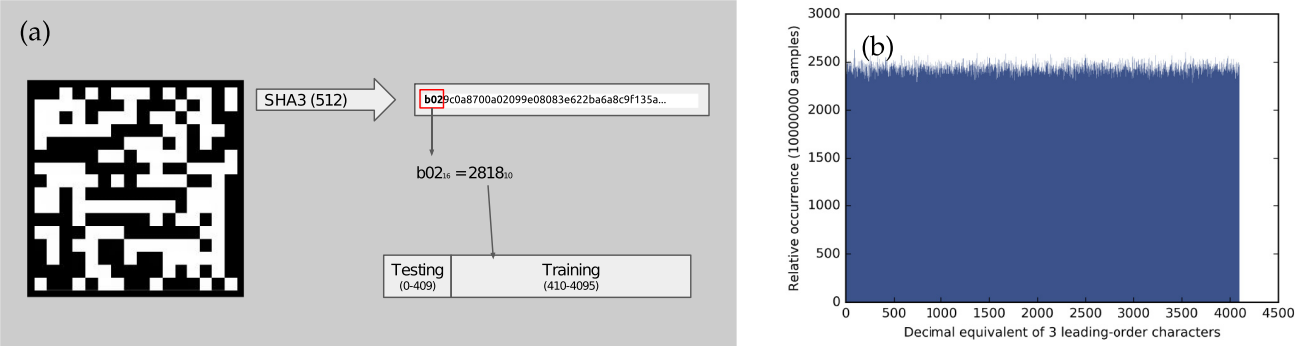


FIG. 15. (a) To randomly divide Ising configurations into training and testing datasets we compute the SHA3 hash of the configuration and use the 12 most significant bits to assign the configuration to either testing or training. (b) This process depends on these 12 bits being uniformly distributed for many configurations.

APPENDIX: TRAINING-TESTING DIVISION

When training machine-learning models, it is typical to divide all available data into two sets: training and testing. In this way, one trains the model on the training data and then evaluates the performance of the model on the testing dataset. It is important that the two sets are nonintersecting (i.e., no test examples appear in the training dataset), so that a fair evaluation of the generalizeability of the model is obtained.

In traditional machine-learning applications, such as image classification, etc., this nonintersecting splitting is quite easy. Since no two images are alike, randomly assigning images to the training or testing sets is appropriate. With the Ising model, and both sampling methods discussed above, there is the potential for duplication of training examples. This is especially the case with targeted sampling, as duplicated training examples are necessary to achieve an even distribution of examples across the energy range. Thus, to separate examples into training and testing datasets, so that no example in the test set appears in the training set, we need a property of the configuration that ultimately can be used to produce a binary value (e.g., 0 = "test," 1 = "train") in arbitrary proportions, say 10% testing, 90% training. We can easily obtain a unique identifier by converting the configuration to a binary value, but the binary value is correlated to the energy, thus the split would not be random. We could randomize a static one-to-one mapping to solve this issue, but storing such a mapping, even for the 6×6 Ising model, would take 275 GB of memory. Our solution to determine whether an example should be assigned to the testing or training set is to compute the SHA3 hash of the configuration and obtain the 512-bit hexadecimal digest. Then the 3 most significant hexadecimal characters (12 most significant bits) determine whether the example gets assigned to the test set or the training set. This allows splitting into arbitrary proportions at a resolution of 1/4096. Figure 15(a) shows a schematic of the process. This procedure depends on 12 most significant bits of the SHA3 hash being uniformly distributed, and indeed they are as shown in Fig. 15(b).

[1] O. S. Ovchinnikov, S. Jesse, P. Bintacchit, S. Trolier-Mckinstry, and S. V. Kalinin, Phys. Rev. Lett. **103**, 157203 (2009).
[2] A. G. Kusne, T. Gao, A. Mehta, L. Ke, M. C. Nguyen, K. Ho, V. Antropov, C.-Z. Wang, M. J. Kramer, C. Long, and I. Takeuchi, Sci. Rep. **4**, 6367 (2015).
[3] S. Jesse, M. Chi, A. Belianinov, C. Beekman, S. V. Kalinin, A. Y. Borisevich, and A. R. Lupini, Sci. Rep. **6**, 26348 (2016).
[4] P. V. Balachandran, D. Xue, J. Theiler, J. Hogden, and T. Lookman, Sci. Rep. **6**, 19660 (2016).
[5] G. Carleo and M. Troyer, Science **355**, 602 (2017).
[6] L. F. Arsenault, A. Lopez-Bezanilla, O. A. Von Lilienfeld, and A. J. Millis, Phys. Rev. B **90**, 155136 (2014).
[7] K. Ch'ng, J. Carrasquilla, R. G. Melko, and E. Khatami, Phys. Rev. X **7**, 031038 (2017).
[8] E. P. L. van Nieuwenburg, Y.-H. Liu, and S. D. Huber, Nat. Phys. **13**, 435 (2017).
[9] J. Carrasquilla and R. G. Melko, Nat. Phys. **13**, 431 (2017).
[10] L. Wang, Phys. Rev. B **94**, 195105 (2016).
[11] A. Tanaka and A. Tomiya, J. Phys. Soc. Jpn. **86**, 063001 (2017).
[12] A. Kitaev and J. Preskill, Phys. Rev. Lett. **96**, 110404 (2006).
[13] M. Levin and X.-G. Wen, Phys. Rev. Lett. **96**, 110405 (2006).
[14] S. Dieleman, K. W. Willett, and J. Dambre, Mon. Not. R. Astron. Soc. **450**, 1441 (2015); arXiv:1410.3831
[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, Nature **518**, 529 (2015).
[16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, Nature **529**, 484 (2016).
[17] L. M. Ghiringhelli, J. Vybiral, S. V. Levchenko, C. Draxl, and M. Scheffler, Phys. Rev. Lett. **114**, 105503 (2015).
[18] A. E. Ferdinand and M. E. Fisher, Phys. Rev. **185**, 832 (1969).
[19] K. Mills, M. Spanner, and I. Tamblyn, Phys. Rev. A **96**, 042113 (2017).
[20] M. D. Zeiler (2012), arXiv:1212.5701.

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, None **1**, 19 (2016); arXiv:1603.04467.

[22] I. Luchak, K. Mills, K. Ryczko, A. Domurad, and I. Tamblyn (2017), arXiv:1708.06686.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Proc. IEEE **86**, 2278 (1998); arXiv:1102.0183

[24] P. Simard, D. Steinkraus, and J. Platt, in *Proceedings of the 7th International Conference on Document Analysis and Recognition*, Vol. 1 (IEEE Comput. Soc., Washington, DC, 2003), pp. 958–963.

[25] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, Tech. Rep. No. IDSIA-01-11, Vol. 22 (Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Manno, Switzerland, 2011), pp. 1237–1242.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (IEEE, 2015), pp. 1–9.

[27] P. Debye and E. Hückel, Phys. Z. **24**, 185 (1923).

[28] H. Yukawa, in *Proceedings of the Physico-Mathematical Society of Japan, 3rd Series*, (Cambridge University Press, Cambridge, 1935), Vol. 17 Chap. I, pp. 48–57.

[29] S. K. Lam, A. Pitrou, and S. Seibert, *Proceedings of the 2nd Workshop on the LLVM Compiler Infrastructure in HPC* (ACM New York, NY, 2015), Article No. 7.

[30] F. Y. Wu, Rev. Mod. Phys. **54**, 235 (1982).

[31] R. B. Potts, The Mathematical Investigation of Some Cooperative Phenomena, Ph.D. thesis, Oxford University (1951).

[32] K. Ryczko, K. Mills, I. Luchak, C. Homenick, and I. Tamblyn (2017), arXiv:1706.09496.

[33] N. Portman and I. Tamblyn, J. Comput. Phys. 43 (2017) arXiv:1611.05891 .

[34] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, Technical Report, Univeristé de Montréal, 2009.

3.2 A well-studied physical system

Another foray into deep learning was largely a proof-of-concept attempt to demonstrate to ourselves, and the broader research community, that deep learning could be used to predict physical quantities, and model well-defined functions relevant to physics. Up to this point, deep learning had been primarily demonstrated on computer vision tasks, where an analytic mapping between input and output cannot be written down. In this sense, the "features" are unknown, and this is why "featureless" deep learning approaches, such as deep convolutional neural networks, enjoy unprecedented, even apparently magical success.

In the physical sciences, the story is slightly different. Oftentimes, there exists a model of a system that can be solved exactly (or approximated to sufficient accuracy to be useful). In such a system, the advantage of deep learning is not immediately evident, as one could employ the traditional method. When, however, the systems become more complicated, the traditional approaches become infeasible, either because there is no known solution, or the process of obtaining the solution is computationally intractable.

"Deep learning and the Schrödinger equation" tackles the first step of the problem: can a deep neural network learn a mapping that can, in theory, be computed exactly? The one-electron Schrödinger equation is not a mystery

by any means, but if deep learning is to enter the field of physics, proving its ability on a well-known, solvable system is paramount to its success and eventual acceptance. Without further ado, I present “Deep learning and the Schrödinger equation”.

PHYSICAL REVIEW A **96**, 042113 (2017)

Deep learning and the Schrödinger equation

Kyle Mills*
Department of Physics, University of Ontario Institute of Technology, Oshawa, Ontario, Canada L1H 7K4

Michael Spanner
National Research Council of Canada, Ottawa, Ontario, Canada K1A 0R6

Isaac Tamblyn†
Department of Physics, University of Ontario Institute of Technology, Oshawa, Ontario, Canada L1H 7K4
and National Research Council of Canada, Ottawa, Ontario, Canada K1A 0R6

(Received 6 February 2017; revised manuscript received 8 June 2017; published 18 October 2017; corrected 25 May 2018)

We have trained a deep (convolutional) neural network to predict the ground-state energy of an electron in four classes of confining two-dimensional electrostatic potentials. On randomly generated potentials, for which there is no analytic form for either the potential or the ground-state energy, the model was able to predict the ground-state energy to within chemical accuracy, with a median absolute error of 1.49 mHa. We also investigated the performance of the model in predicting other quantities such as the kinetic energy and the first excited-state energy.

DOI: 10.1103/PhysRevA.96.042113

I. INTRODUCTION

Solving the electronic structure problem for molecules, materials, and interfaces is of fundamental importance to a large number of disciplines including physics, chemistry, and materials science. Since the early development of quantum mechanics, it has been noted, by Dirac among others, that "...approximate, practical methods of applying quantum mechanics should be developed, which can lead to an explanation of the main features of complex atomic systems without too much computation" [1]. Historically, this has meant invoking approximate forms of the underlying interactions (mean field, tight binding, etc.) or relying on phenomenological fits to a limited number of either experimental observations or theoretical results (e.g., force fields) [2–8]. The development of feature-based models is not new in the scientific literature. Indeed, prior even to the acceptance of the atomic hypothesis, van der Waals argued for an equation of state based on two physical features [9]. Machine learning (i.e., fitting parameters within a model) has been used in physics and chemistry since the dawn of the computer age. The term machine learning is new; the approach is not.

More recently, high-level *ab initio* calculations have been used to train artificial neural networks to fit high-dimensional interaction models [10–15] and to make informed predictions about material properties [16,17]. These approaches have proven to be quite powerful, yielding models trained for specific atomic species or based upon hand-selected geometric features [18–20]. Hand-selected features are arguably a significant limitation of such approaches, with the outcomes dependent upon the choice of input representation and the inclusion of all relevant features. This limitation is well known in the fields of handwriting recognition and image classification, where the performance of the traditional hand-selected feature approach has stagnated [21].

Such feature-based approaches are also being used in materials discovery [22–24] to assist materials scientists in efficiently targeting promising material candidates. Unsupervised learning techniques have been used to identify phases in many-body atomic configurations [25]. In previous work, an artificial neural network was shown to interpolate the mapping of position to wave function for a specific electrostatic potential [26–28], but the fit was not transferable, a limitation also present in other applications of artificial neural networks to partial differential equations [29,30]. By transferable, we mean that a model trained on a particular form of partial differential equation will accurately and reliably predict results for examples of the same form (in our case, different confining potentials).

Machine learning can also be used to accelerate or bypass some of the heavy machinery of the *ab initio* method itself. In [31], the authors replaced the kinetic energy functional within density-functional theory with a machine-learned one, and in [32,33], the authors "learned" the mappings from potential to electron density and from charge density to kinetic energy, respectively.

Here we use a fundamentally different approach inspired by the successful application of deep convolutional neural networks to problems in computer vision [34–37] and computational games [38,39]. Rather than seeking an appropriate input representation to capture the relevant physical attributes of a system, we train a highly flexible model on an enormous collection of ground-truth examples. In doing so, the deep neural network learns both the features (in weight space) and the mapping required to produce the desired output. This approach does not depend on the appropriate selection of input representations and features; we provide the same data to both the deep neural network and the numerical method. As such, we call this featureless learning. Such an approach may offer a more scalable and parallelizable approach to large-scale electronic structure problems than existing methods can offer.

*kyle.mills@uoit.net
†isaac.tamblyn@nrc.ca

In this paper we demonstrate the success of a featureless machine learning approach, a convolutional deep neural network, at learning the mapping between a confining electrostatic potential and quantities such as the ground-state energy, kinetic energy, and first excited state of a bound electron. The excellent performance of our model suggests deep learning as an important direction for treating multielectron systems in materials.

It is known that a sufficiently large artificial neural network can approximate any continuous mapping [40,41], but the cost of optimizing such a network can be prohibitive. Convolutional neural networks make computation feasible by exploiting the spatial structure of input data [42], similar to how the neurons in the visual cortex function [43]. When multiple convolutional layers are included, the network is called a deep convolutional neural network, forming a hierarchy of feature detection [44]. This makes them particularly well suited to data rooted in physical origin [45,46], since many physical systems also display a structural hierarchy. Applications of such a network structure in the field of electronic structure, however, are few (although recent work focused on training against a geometric matrix representation looks particularly promising [47]).

II. METHODS

A. Training set: Choice of potentials

Developing a deep learning model involves both the design of the network architecture and the acquisition of training data. The latter is the most important aspect of a machine learning model, as it defines the transferability of the resulting model. We investigated four classes of potentials: simple harmonic oscillators (SHOs), infinite wells (IWs) (i.e., particle in a box), double-well inverted Gaussians (DIG), and random potentials. Each potential can be thought of as a grayscale image: a grid of floating-point numbers.

B. Numerical solver

We implemented a standard finite-difference [48] method to solve the eigenvalue problem

$$\hat{H}\psi \equiv (\hat{T} + \hat{V})\psi = \varepsilon\psi \tag{1}$$

for each potential V we created. The potentials were generated with a dynamic range and length scale suitable to produce ground-state energies within a physically relevant range. With the random potentials, special care was taken to ensure that some training examples produced nontrivial wave functions (Fig. 1). Atomic units are used, such that $\hbar = m_e = 1$. The potentials are represented on a square domain from -20 to 20 a.u., discretized on a 256×256 grid. As the simple-harmonic-oscillator potentials have an analytic solution, we used this as reference with which to validate the accuracy of the solver. The median absolute error between the analytic and the calculated energies for all simple-harmonic-oscillator potentials was 0.12 mHa. We discuss the generation of all potentials further in the Appendixes.

The simple harmonic oscillator presents the simplest case for a convolutional neural network as there is an analytic solution dependent on two simple parameters (k_x and k_y) that uniquely define the ground-state energy of a single electron [$\varepsilon_0 = \frac{\hbar}{2}(\sqrt{k_x} + \sqrt{k_y})$]. Furthermore, these parameters represent a very physical and visible quantity: the curvature of the potential in the two primary axes. Although these parameters are not provided to the neural network explicitly, the fact that a simple mapping exists means that the convolutional neural network need only learn it to accurately predict energies.

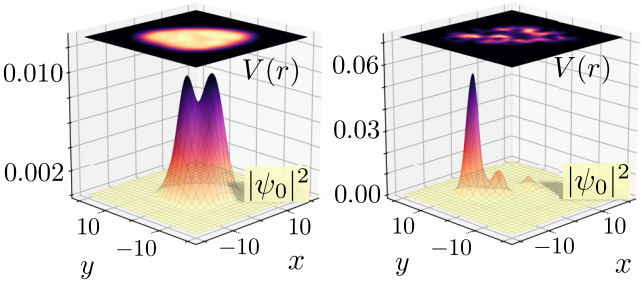


FIG. 1. Wave functions (probability density) $|\psi_0|^2$ and the corresponding potentials $V(r)$ for two random potentials.

A similar situation exists for the infinite well. Like the simple harmonic oscillator, the ground-state energy depends only on the width of the well in the two dimensions [$\varepsilon_0 = \frac{1}{2}\pi^2\hbar^2(L_x^{-2} + L_y^{-2})$]. It would be no surprise if even a modest network architecture is able to accurately "discover" this mapping. An untrained human, given a ruler, sufficient examples, and an abundance of time, would likely succeed in determining this mapping.

The double-well inverted Gaussian data set is more complex in two respects. First, the potential, generated by summing a pair of two-dimensional (2D) Gaussians, depends on significantly more parameters; the depth, width, and aspect ratio of each Gaussian; in addition, the relative positions of the wells will impact the ground-state energy. Furthermore, there is no known analytical solution for a single electron in a potential well of this nature. There is, however, still a concise function that describes the underlying potential, and while this is not directly accessible to the convolutional neural network, one must wonder if the existence of such simplifies the task of the convolutional neural network. Gaussian confining potentials appear in works relating to quantum dots [49,50].

The random data set presents the ultimate challenge. Each random potential is generated by a multistep process with randomness introduced at numerous steps along the way. There is no closed-form equation to represent the potentials and certainly not the eigenenergies. A convolutional neural network tasked with learning the solution to the Schrödinger equation through these examples would have to base its predictions on many individual features, truly learning the mapping of potential to energy. One might question our omission of the Coulomb potential as an additional canonical example. The singular nature of the Coulomb potential is difficult to represent within a finite dynamic range and, more importantly, the electronic structure methods that we would ultimately seek to reproduce already have frameworks in place to deal with these singularities (e.g., pseudopotentials).

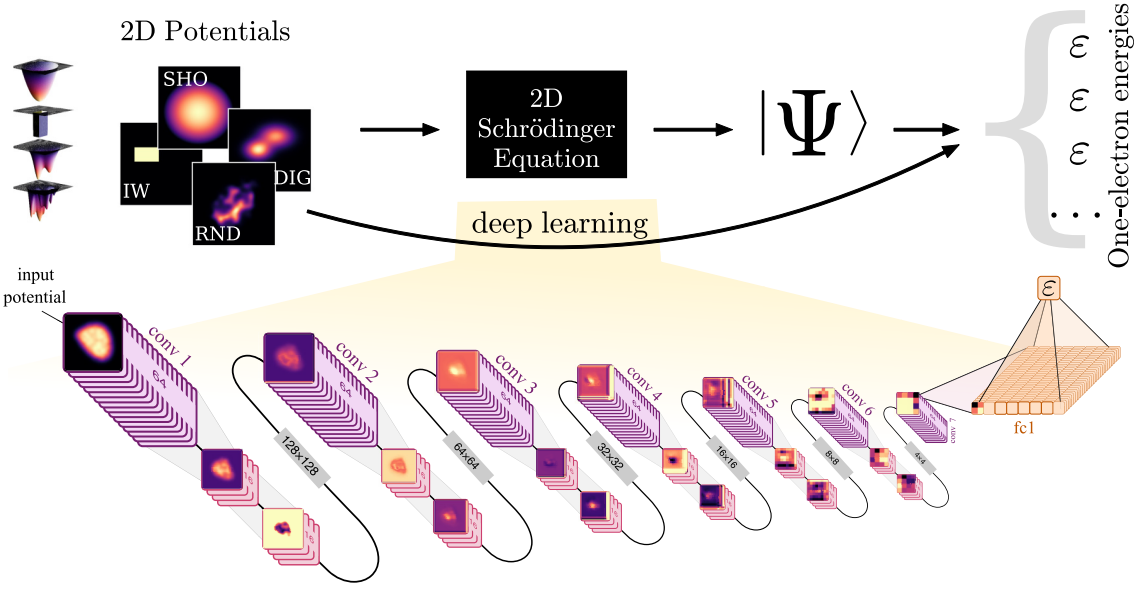


FIG. 2. In this work, we use the machinery of deep learning to learn the mapping between potential and energy, bypassing the need to numerically solve the Schrödinger equation and the need for computing wave functions. The architecture we used (shown here) consisted primarily of convolutional layers capable of extracting relevant features of the input potentials. Two fully connected layers at the end serve as a decision layer, mapping the automatically extracted features to the desired output quantity. No manual feature selection is necessary; this is a featureless-learning approach.

C. Deep neural network

We chose to use a simple yet deep neural network architecture (shown in Fig. 2) composed of a number of repeated units of convolutional layers, with sizes chosen for a balance of speed and accuracy (inset of Fig. 3). We use two different types of convolutional layers, which we call reducing and nonreducing.

The seven reducing layers operate with filter (kernel) sizes of 3×3 pixels. Each reducing layer operates with 64 filters and a stride of 2×2, effectively reducing the image resolution by a factor of 2 at each step. In between each pair of these reducing convolutional layers, we have inserted two convolutional layers (for a total of 12) that operate with 16 filters of size 4×4. These filters have unit stride and therefore preserve the resolution of the image. The purpose of these layers is to add additional trainable parameters to the network. All convolutional layers have rectified linear unit (ReLU) activation.

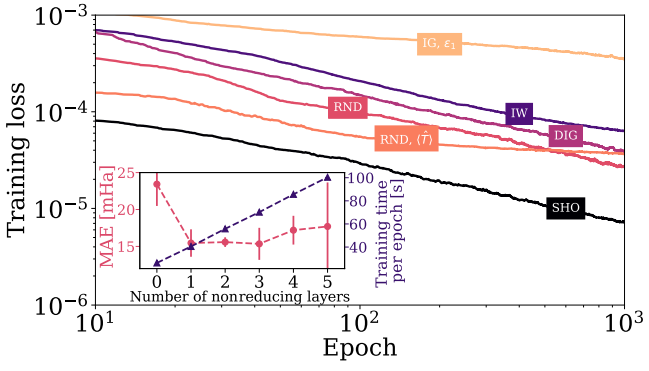


FIG. 3. Training loss curve for each model we trained. Since the training loss is based upon the training data sets, it does not necessarily indicate how well the model generalizes to new examples. The convergence seen here indicates that 1000 epochs is an adequate stopping point; further training would produce further reduction in loss, however 1000 epochs provides sufficient evidence that the method performs well on the most interesting (i.e., random) potentials. In the inset, we see that two nonreducing convolution layers is a consistent balance of training time and low error.

The final convolutional layer is fed into a fully connected layer of width 1024, also with ReLU activation. This layer feeds into a final fully connected layer with a single output. This output is the output value of the deep neural network (DNN). It is used to compute the mean-square error between the true label and the predicted label, also known as the loss.

We used the AdaDelta [51] optimization scheme with a global learning rate of 0.001 to minimize this loss function (Fig. 3), monitoring its value as training proceeded. We found that after 1000 epochs (1000 times through all the training examples), the loss no longer decreased significantly.

We built a custom TensorFlow [52] implementation in order to make use of four graphical processing units (GPUs) in parallel. We placed a complete copy of the neural network on each of the four GPUs, so that each could compute a forward- and backpropagation iteration on one full batch of images. Thus our effective batch size was 1000 images per iteration (250 per GPU). After each iteration, the GPUs share their independently computed gradients with the optimizer and the optimizer moves the parameters in the direction that minimizes the loss function. Unless otherwise specified, all training data sets consisted of 200 000 training examples and training was run for 1000 epochs. All reported errors are based on evaluating the trained model on validation data sets consisting of 50 000 potentials not accessible to the network during the training process.

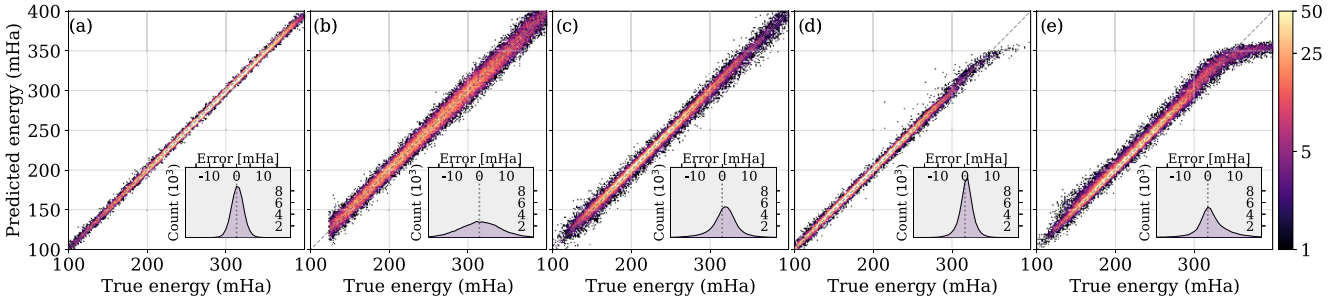


FIG. 4. Histograms of the true vs predicted energies for each example in the test set indicate the performance of the various models: (a) simple harmonic oscillator, (b) infinite well, (c) DIG potential, (d) random potential, and (e) DIG potential on random model. The insets show the distribution of error away from the diagonal line representing perfect predictions. A 1-mHa^2 square bin is used for the main histograms and a 1-mHa bin size for the inset histogram. During training, the neural network was not exposed to the examples on which theses plots are based. The higher error at high energies in (d) is due to fewer training examples being present in the data set at these energies. The histogram shown in (d) is for the further-trained model, described in the text.

III. RESULTS

Figures 4(a)–4(d) displays the results for the simple-harmonic-oscillator, infinite well, double-well inverted Gaussian, and random potentials. The simple harmonic oscillator, being one of the simplest potentials, performed extremely well. The trained model was able to predict the ground-state energies with a median absolute error (MAE) of 1.51 mHa.

The infinite well potentials performed moderately well with a MAE of 5.04 mHa. This is notably poorer than the simple-harmonic-oscillator potentials, despite their similarity in being analytically dependent upon two simple parameters. This is likely due to the sharp discontinuity associated with the infinite well potentials, combined with the sparsity of information present in the binary-valued potentials.

The model trained on the double-well inverted Gaussian potentials performed moderately well with a MAE of 2.70 mHa and the random potentials performed quite well with a MAE of 2.13 mHa. We noticed, however, that the loss was not completely converged at 1000 epochs, so we provided an additional 200 000 training examples to the network and allowed it to train for an additional 1000 epochs. With this added training, the model performed exceptionally well, with a MAE of 1.49 mHa, below the threshold of chemical accuracy (1 kcal/mol, 1.6 mHa). In Fig. 4(d), it is evident that the model performs more poorly at high energies, a result of the relative absence of high-energy training examples in the data set. Given the great diversity in this latter set of potentials, it is impressive that the convolutional neural network was able to learn how to predict the energy with such a high degree of accuracy.

Now that we have a trained model that performs well on the random test set, we investigated its transferability to another class of potentials. The model trained on the random data set is able to predict the ground-state energy of the double-well inverted Gaussian potentials with a MAE of 2.94 mHa. We can see in Fig. 4(e) that the model fails at high energies, an expected result given that the model was not exposed to many examples in this energy regime during training on the overall lower-energy random data set. This moderately good performance is not entirely surprising; the production of the random potentials includes an element of Gaussian blurring, so the neural network would have been exposed to features similar to what it would see in the double-well inverted Gaussian data set. However, this moderate performance is a testament to the transferability of convolutional neural network models. Furthermore, we trained a model on an equal mixture of all four classes of potentials. It performs moderately with a MAE of 5.90 mHa. This error could be reduced through further tuning of the network architecture, allowing it to better capture the higher variation in the data set.

The total energy is just one of the many quantities associated with these one-electron systems. To demonstrate the applicability of deep neural networks to other quantities, we trained a model on the first excited-state energy ε_1 of the double-well inverted Gaussian potentials. The model achieved a MAE of 10.93 mHa. We now have two models capable of predicting the ground-state and first-excited-state energies separately, demonstrating that a neural network can learn quantities other than the ground-state energy.

The ground-state and first excited state are both eigenvalues of the Hamiltonian. Therefore, we investigated the training of a model on the expectation value of the kinetic energy $\langle\hat{T}\rangle = \langle\psi_0|\hat{T}|\psi_0\rangle$ under the ground-state wave function ψ_0 that we computed numerically for the random potentials. Since $\hat{H}$ and $\hat{T}$ do not commute, the prediction of $\langle\hat{T}\rangle$ can no longer be summarized as an eigenvalue problem. The trained model predicts the kinetic energy value with a MAE of 2.98 mHa. While the spread of testing examples in Fig. 5(a) suggests that the model performs more poorly, the absolute error is still small.

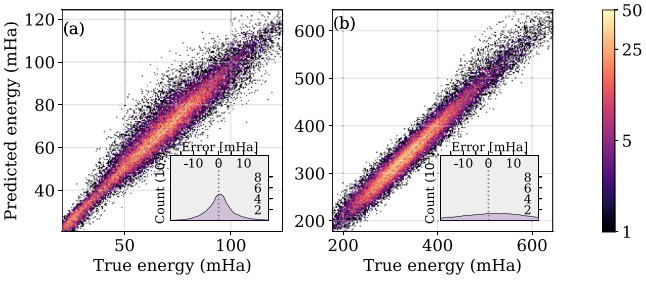


FIG. 5. Histograms of the true vs predicted energies for the model trained on the (a) kinetic energy $\langle\hat{T}\rangle$ of the random potential and (b) excited-state energy ε_1 of the double-well inverted Gaussian.

IV. CONCLUSION

We note that many other machine learning algorithms exist and have traditionally seen great success, such as kernel ridge regression [18,20,32,53–55] and random forests [18,56]. Like these algorithms, convolutional deep neural networks have the ability to learn relevant features and form a nonlinear input-to-output mapping without prior formulation of an input representation [47,57]. In our tests, these methods performed more poorly and scaled such that a large number of training examples is infeasible. We have included a comparison of these alternative machine learning methods in the Appendixes, justifying our decision of using a deep convolutional neural network. One notable limitation of our approach is that the efficient training and evaluation of the deep neural network requires uniformity in the input size. Future work should focus on an approach that would allow transferability to variable input sizes.

Additionally, an electrostatic potential defined on a finite grid can be rotated in integer multiples of 90°, without a change to the electrostatic energies. Convolutional deep neural networks do not natively capture such rotational invariance. Clearly, this is a problem in any application of deep neural networks (e.g., image classification) and various techniques are used to compensate for the desired invariance. The common approach is to train the network on an augmented data set consisting of both the original training set and rotated copies of the training data [58]. In this way, the network learns a rotationally invariant set of features.

In demonstration of this technique, we tuned our model trained on the random potentials by training it further on an augmented data set of rotated random potentials. We then tested our model on the original testing data set as well as a rotated copy of the test set. The median absolute error in both cases was less than 1.6 mHa. The median absolute difference in predicted energy between the rotated and unaltered test sets was however larger, at 1.7 mHa. This approach to training the deep neural network is not absolutely rotationally invariant, however the numerical error experienced due to a rotation was on the same order as the error of the method itself. Recent proposals to modify the network architecture itself to make it rotationally invariant are promising, as the additional training cost incurred with using an augmented data set could be avoided [59,60].

In summary, convolutional deep neural networks are promising candidates for application to electronic structure calculations as they are designed for data that have a spatial encoding of information. As the number of electrons in a system increases, the computational complexity grows polynomially. Accurate electronic structure methods (e.g., coupled cluster) exhibit a scaling with respect to the number of particles of N^7 and even the popular Kohn-Sham formalism of density-functional theory scales as N^3 [61,62]. The evaluation of a convolutional neural network exhibits no such scaling, and while the training process for more complicated systems would be more expensive, this is a one-time cost.

In this work we have taken a simple problem (one electron in a confining potential) and demonstrated that a convolutional neural network can automatically extract features and learn the mapping between $V(r)$ and the ground-state energy ε_0 as well as the kinetic energy $\langle\hat{T}\rangle$, and the first-excited-state energy ε_1. Although our focus here has been on a particular type of problem, namely, an electron in a confining 2D well, the concepts here are directly applicable to many problems in physics and engineering. Ultimately, we have demonstrated the ability of a deep neural network to learn, through example alone, how to rapidly approximate the solution to a set of partial differential equations. A generalizable, transferable deep learning approach to solving partial differential equations would impact all fields of theoretical physics and mathematics.

The supporting data for this article are available from the digital repository of the National Reserach Council of Canada [63].

ACKNOWLEDGMENTS

The authors would like to acknowledge fruitful discussions with P. Bunker, P. Darancet, D. Klug, and D. Prendergast. K.M. and I.T. acknowledge funding from NSERC and SOSCIP. Compute resources were provided by SOSCIP, Compute Canada, National Research Council of Canada, and an NVIDIA Faculty Hardware Grant.

APPENDIX A: COMPARISON OF MACHINE LEARNING METHODS

One might question the use of a convolutional deep neural network over other more traditional machine learning approaches. After all, kernel ridge regression (KRR), random forests (RF), and artificial neural networks (ANN) have proven to be quite useful (see the main text for references to appropriate work). Here we compare the use of our convolutional deep neural network approach to kernel ridge regression and random forests, the latter two implemented through Scikit-learn [64].

1. Kernel ridge regression

We trained a kernel ridge regression model on a training set of simple-harmonic-oscillator images, recording the wall time

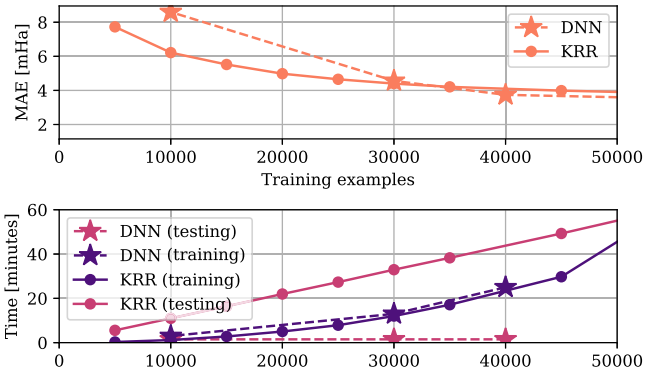


FIG. 6. Kernel ridge regression on simple-harmonic-oscillator potentials. When few training examples are provided, kernel ridge regression performs better; however, with a larger number of training examples, both methods perform comparably, with DNN slightly better. The training time for kernel ridge regression scales quadratically. The evaluation time for a fixed number of testing examples scales linearly with respect to the number of training examples in the case of kernel ridge regression. In the case of the deep neural network, the training set size does not affect the testing set evaluation.

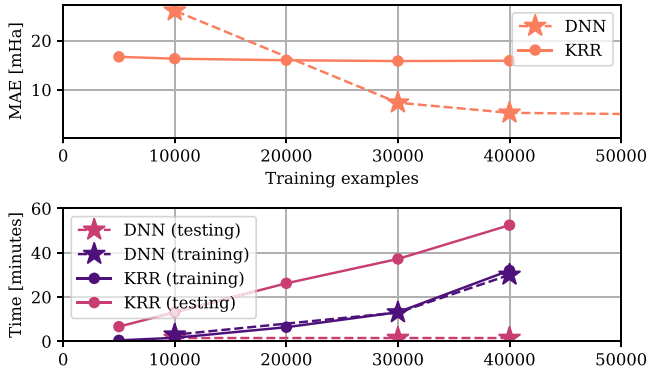


FIG. 7. Kernel ridge regression on random potentials. When few training examples are present, kernel ridge regression performs better (at constant training time). This is likely due to the fact that the DNN is only given 10 s to run. At larger training set sizes, the deep neural network performs much better; kernel ridge regression barely improves as training set size increases, however training wall time increases dramatically.

(real-world time) taken to train the model. Then we evaluated the trained model on a test set (the same test set was used throughout). We recorded both the evaluation wall time and the MAE observed from the trained model. We then trained our deep neural network on the same training data set, allowing it the same training wall time as the KRR model. We then evaluated the deep neural network on the same testing set of data, again recording the MAE and the evaluation wall time. This process was repeated for various training set sizes and on training data from both the simple-harmonic-oscillator and random data sets. The results are presented in Figs. 6 and 7.

2. Random forests

We carried out an identical process, training a random forests regressor. The results are presented in Figs. 8 and 9.

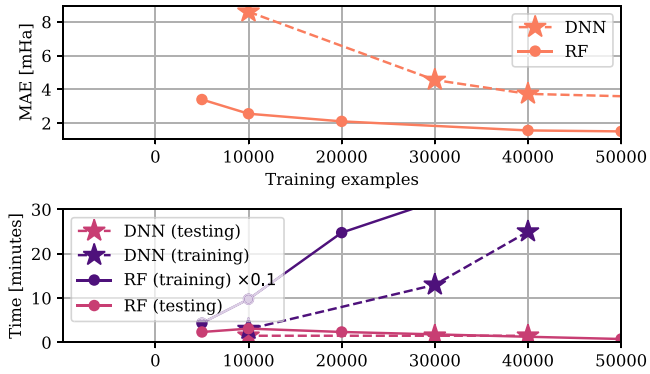


FIG. 8. Random forests on simple harmonic oscillator. Random forests perform better than deep neural networks for all training set sizes on the relatively trivial simple-harmonic-oscillator data set. Random forests takes a very long time to train. Note that the training times plotted above have been scaled by a factor of 0.1 for plotting and thus the true times are ten times greater than shown.

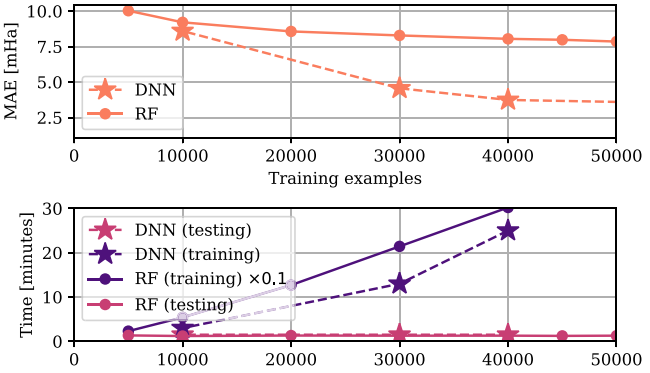


FIG. 9. Random forests on random potentials. On the more complicated random potentials, random forests perform significantly worse than the deep neural network. This combined with the extremely high training time suggests that the deep neural network is much better equipped to handle these more varied potentials.

3. Discussion

While the timing comparison is not quantitatively fair (the random forest algorithm is not parallelized and uses only one CPU core, the kernel ridge regression algorithm is parallelized and ran across all available cores, and the deep neural network is highly parallelized via GPU optimization and runs across thousands of cores), this investigation gives useful insight into the time-to-solution advantages of deep neural networks. The error rates, however are quantitatively comparable, as the KRR and random forest (RF) algorithms were permitted to run until convergence. The DNN was able to perform better in most cases given the same amount of wall time.

We see that for all but the simplest cases, our deep neural network is vastly superior to both kernel ridge regression and random forests. For very simple potentials, it is understandable that the machinery of the deep neural network was unnecessary and that the traditional methods perform well. For more complicated potentials with more variation in the input data, the deep neural network was able to provide significantly better accuracy in the same amount of time.

APPENDIX B: DATA-SET GENERATION

The potentials are defined on a grid from $x, y = -20$ to 20 a.u. on a 256×256 grid.

1. Simple harmonic oscillator

The SHO potentials are generated with the scalar function

$$V(x,y) = \tfrac{1}{2}[k_x(x-c_x)^2 + k_y(y-c_y)^2], \qquad \text{(B1)}$$

TABLE I. Random number generation criteria for the simple-harmonic-oscillator dataset.

Parameter	Description	Lower bound	Upper bound
k_x	spring constant	0.0	0.16
k_y	spring constant	0.0	0.16
c_x	center position	−8.0	8.0
c_y	center position	−8.0	8.0

TABLE II. Random number generation criteria for the double-well-inverted-Gaussian dataset.

Parameter	Description	Lower bound	Upper bound
A_1	well 1 depth	2.0	4.0
A_2	well 2 depth	2.0	4.0
c_{x_1}	well 1 center x	−8.0	8.0
c_{y_1}	well 1 center y	−8.0	8.0
c_{x_2}	well 2 center x	−8.0	8.0
c_{y_2}	well 2 center y	−8.0	8.0
k_{x_1}	well 1 width	1.6	8.0
k_{y_1}	well 1 length	1.6	8.0
k_{x_2}	well 2 width	1.6	8.0
k_{y_2}	well 2 length	1.6	8.0

where k_x, k_y, c_x, and c_y are randomly generated according to Table I. The potentials are truncated at 20.0 Ha (i.e., if $V > 20$, $V = 20$).

2. Infinite well

The IW potentials are generated with the scalar function

$$V(x,y) = \begin{cases} 0 & \frac{1}{2}(2c_x - L_x) < x \leqslant \frac{1}{2}(2c_x + L_x), \\ & \frac{1}{2}(2c_y - L_y) < y \leqslant \frac{1}{2}(2c_y + L_y) \\ 20 & \text{otherwise}, \end{cases} \quad \text{(B2)}$$

where 20.0 is used as numerical infinity, an appropriate choice given the scale of energies used. Because of the nature of the IW energy, randomly generating L_x and L_y independently leads to a distribution of energies highly biased toward low-energy values (it is more likely to randomly produce a large well than a small). Since we want a distribution that is as even as possible over the range of energies, we need to take a slightly different approach. We randomly generate the energy E uniformly on the interval [0,0.4] Ha. We then generate L_x randomly on the interval [4.0,15.0], defining the width of the well. We then solve for the value of L_y that will produce an energy of E, given L_x, e.g.,

$$L_y = 1 \Big/ \sqrt{\frac{2E}{\pi^2} - \frac{1}{L_x^2}}. \quad \text{(B3)}$$

Not all combinations of L_x and E lead to valid solutions for L_y, so we keep trying until one does. We then swap the values of L_x and L_y with a 50% probability to prevent one dimension of the well always being larger. This process leads to a relatively even distribution of energies.

TABLE III. Random number generation criteria for the random potential dataset. In column 2, SD denotes standard deviation.

Parameter	Description	Lower bound	Upper bound
σ_1	SD blur 1	6	10
k	blob points	2	7
R	blob size	80	180
σ_2	SD blur 2	10	16

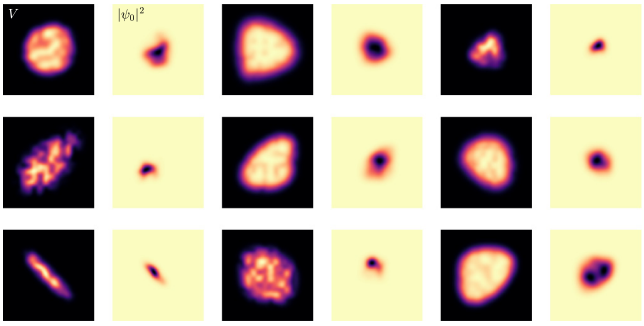


FIG. 10. Some example random potentials V and the norm of their associated ground-state wave functions $|\psi_0|^2$.

3. Double-well inverted Gaussians

The DIG potentials are generated with the scalar function

$$V(x,y) = -A_1 \exp\left[-\left(\frac{x - c_{x_1}}{k_{x_1}}\right)^2 - \left(\frac{y - c_{y_1}}{k_{y_1}}\right)^2\right] - A_2 \exp\left[-\left(\frac{x - c_{x_2}}{k_{x_2}}\right)^2 - \left(\frac{y - c_{y_2}}{k_{y_2}}\right)^2\right], \quad \text{(B4)}$$

where the parameters are randomly sampled from a uniform distribution within the ranges given in Table II. These ranges were determined through trial and error to achieve energies in the range of 0–400 mHa.

4. Random potentials

The random potentials are generated through a lengthy process motivated by three requirements: The potentials must (a) be random (i.e., extremely improbable that two identical potentials ever be generated), (b) be smooth, and (c) go to a maximum of 20.0 at the boundary. First, we generate a 16 × 16

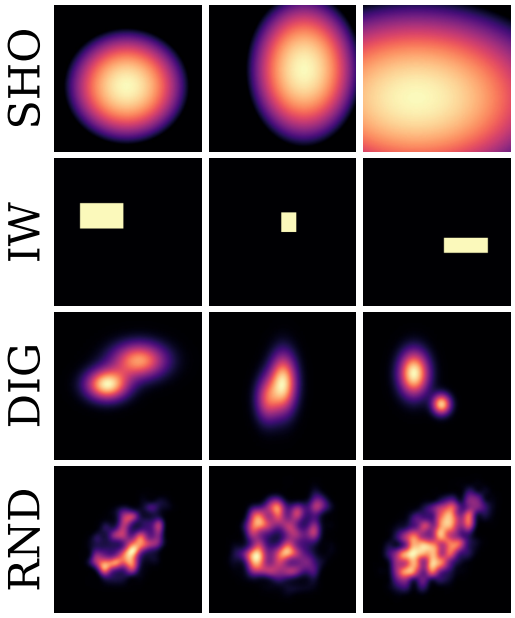


FIG. 11. Examples of the four classes of potentials.

binary grid of 1's and 0's and upscale it to 256×256. We then generate a second 16×16 binary grid and upscale it to 128×128. We center the smaller grid within the larger grid and then subtract them elementwise. We then apply a Gaussian blur with standard deviation σ_1 to the resulting image, where σ_1 is generated uniformly within the range given in Table III. The potential is now random and smooth, but does not achieve a maximum at the boundary.

To achieve this, we generate a mask that smoothly goes to 0 at the boundary and 1 in the interior. We wish the mask to be random, e.g., a randomly generated blob. To generate the blob, we generate k^2 random coordinate pairs on a 200×200 grid, where k is an integer between 2 and 7, inclusive. We then throw away all points that lie inside the convex hull of these points and smoothly interpolate the remaining points with cubic splines. We then form a binary mask by filling the inside of this closed blob with 1's and the outside with 0's. Resizing the blob to a resolution of $R \times R$ and applying a Gaussian blur with standard deviation σ_2, we arrive at the final mask. Here R and σ_2 are generated uniformly within the ranges given in Table III.

Elementwise multiplication of the mask with the randomly blurred image gives a random potential that approaches zero at the boundary. We randomize the "sharpness" of the potential by then exponentiating by $d = 0.1$, 0.5, 1.0, or 2.0, chosen at random with equal probabilities (i.e., $V := V^d$). We then subtract the result from its maximum to invert the well.

This process, while lengthy, produces very random potentials, of which no two are alike. The energy range of 0–400 mHa is appropriate for producing wave functions that span a moderate portion of the domain, as shown in Fig. 10. Examples of all classes of potentials can be seen in Fig. 11.

[1] P. A. M. Dirac, Proc. R. Soc. London Ser. A **123**, 714 (1929).
[2] M. J. Cherukara, B. Narayanan, A. Kinaci, K. Sasikumar, S. K. Gray, M. K. Chan, and S. K. R. S. Sankaranarayanan, J. Phys. Chem. Lett. **7**, 3752 (2016).
[3] M. Riera, A. W. Götz, and F. Paesani, Phys. Chem. Chem. Phys. **18**, 30334 (2016).
[4] A. Jaramillo-Botero, S. Naserifar, and W. A. Goddard, J. Chem. Theory Comput. **10**, 1426 (2014).
[5] B. W. H. van Beest, G. J. Kramer, and R. A. van Santen, Phys. Rev. Lett. **64**, 1955 (1990).
[6] J. W. Ponder and D. A. Case, Adv. Protein Chem. **66**, 27 (2003).
[7] V. Hornak, R. Abel, A. Okur, B. Strockbine, A. Roitberg, and C. Simmerling, Proteins **65**, 712 (2006).
[8] D. J. Cole, M. C. Payne, G. Csányi, S. Mark Spearing, and L. Colombi Ciacchi, J. Chem. Phys. **127**, 204704 (2007).
[9] J. D. van der Waals, De continuiteit van den gasen Vloeistoftoestand, Ph.D. thesis, University of Leiden, 1873.
[10] H. Z. Li, L. Li, Z. Y. Zhong, Y. Han, L. Hu, and Y. H. Lu, Math. Prob. Eng. **2013**, 860357 (2013).
[11] J. Behler and M. Parrinello, Phys. Rev. Lett. **98**, 146401 (2007).
[12] T. Morawietz and J. Behler, J. Phys. Chem. A **117**, 7356 (2013).
[13] J. Behler, R. Martonák, D. Donadio, and M. Parrinello, Phys. Status Solidi B **245**, 2618 (2008).
[14] P. E. Dolgirev, I. A. Kruglov, and A. R. Oganov, AIP Adv. **6**, 085318 (2016).
[15] N. Artrith and A. Urban, Comput. Mater. Sci. **114**, 135 (2016).
[16] W. Tian, F. Meng, L. Liu, Y. Li, and F. Wang, Sci. Rep. **7**, 40827 (2017).
[17] M. Rupp, A. Tkatchenko, K. R. Müller, and O. A. von Lilienfeld, Phys. Rev. Lett. **108**, 058301 (2012).
[18] F. A. Faber, L. Hutchison, B. Huang, J. Gilmer, S. S. Schoenholz, G. E. Dahl, O. Vinyals, S. Kearnes, P. F. Riley, and O. A. von Lilienfeld, J. Chem. Theory Comput. (2017), doi: 10.1021/acs.jctc.7b00577.
[19] G. Montavon, M. Rupp, V. Gobre, A. Vazquez-Mayagoitia, K. Hansen, A. Tkatchenko, K. R. Müller, and O. Anatole von Lilienfeld, New J. Phys. **15**, 095003 (2013).
[20] A. Lopez-Bezanilla and O. A. von Lilienfeld, Phys. Rev. B **89**, 235411 (2014).
[21] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, arXiv:1408.5093.
[22] S. Curtarolo, D. Morgan, K. Persson, J. Rodgers, and G. Ceder, Phys. Rev. Lett. **91**, 135503 (2003).
[23] G. Hautier, C. C. Fischer, A. Jain, T. Mueller, and G. Ceder, Chem. Mater. **22**, 3762 (2010).
[24] Y. Saad, D. Gao, T. Ngo, S. Bobbitt, J. R. Chelikowsky, and W. Andreoni, Phys. Rev. B **85**, 104104 (2012).
[25] L. Wang, Phys. Rev. B **94**, 195105 (2016).
[26] C. Monterola and C. Saloma, Opt. Commun. **222**, 331 (2003).
[27] Y. Shirvany, M. Hayati, and R. Moradian, Commun. Nonlinear Sci. Numer. Simul. **13**, 2132 (2008).
[28] C. Caetano, J. L. Reis, J. Amorim, M. R. Lemes, and A. D. Pino, Int. J. Quantum Chem. **111**, 2732 (2011).
[29] B. P. van Milligen, V. Tribaldos, and J. A. Jiménez, Phys. Rev. Lett. **75**, 3594 (1995).
[30] G. Carleo and M. Troyer, Science **355**, 602 (2017).
[31] J. C. Snyder, M. Rupp, K. Hansen, K.-R. Müller, and K. Burke, Phys. Rev. Lett. **108**, 253002 (2012).
[32] F. Brockherde, L. Vogt, L. Li, M. E. Tuckerman, K. Burke, and K.-R. Müller, arXiv:1609.02815.
[33] K. Yao and J. Parkhill, J. Chem. Theory Comput. **12**, 1139 (2016).
[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Proc. IEEE **86**, 2278 (1998).
[35] P. Simard, D. Steinkraus, and J. Platt, in *Proceedings of the Seventh International Conference on Document Analysis and Recognition* (IEEE, Piscataway, 2003), Vol. 1, pp. 958–963.
[36] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence* (AAAI, Palo Alto, 2011), Vol. 22, pp. 1237–1242.
[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (IEEE, Piscataway, 2015), pp. 1–9.
[38] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, Nature (London) **529**, 484 (2016).

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, Nature (London) **518**, 529 (2015).
[40] K. I. Funahashi, Neural Networks **2**, 183 (1989).
[41] J. L. Castro, C. J. Mantas, and J. M. Benítez, Neural Networks **13**, 561 (2000).
[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, in *Proceedings of the 25th International Conference on Neural Information Processing Systems* (Curran, Red Hook, 2012), pp. 1097–1105.
[43] D. H. Hubel and T. N. Wiesel, J. Physiol. **195**, 215 (1968).
[44] Y. Bengio, Found. Trends Mach. Learn. **2**, 1 (2009).
[45] P. Mehta and D. J. Schwab, arXiv:1410.3831.
[46] H. W. Lin, M. Tegmark, and D. Rolnick, J. Stat. Phys. **168**, 1223 (2017).
[47] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, Nat. Commun. **8**, 13890 (2017).
[48] P. Frolkovič, *Acta Applicandae Mathematicae*, 3rd ed. (Cambridge University Press, New York, 1990), Vol. 19, pp. 297–299.
[49] A. Gharaati and R. Khordad, Superlatt. Microstruct. **48**, 276 (2010).
[50] S. S. Gomez and R. H. Romero, Cent. Eur. J. Phys. **7**, 12 (2009).
[51] M. D. Zeiler, arXiv:1212.5701.
[52] M. Abadi *et al.*, arXiv:1603.04467 (2015).
[53] L. F. Arsenault, A. Lopez-Bezanilla, O. A. von Lilienfeld, and A. J. Millis, Phys. Rev. B **90**, 155136 (2014).
[54] L. Li, J. C. Snyder, I. M. Pelaschier, J. Huang, U.-N. Niranjan, P. Duncan, M. Rupp, K.-R. Müller, and K. Burke, Int. J. Quantum Chem. **116**, 819 (2016).
[55] T. Suzuki, R. Tamura, and T. Miyazaki, Int. J. Quantum Chem. **117**, 33 (2017).
[56] L. Ward, A. Agrawal, A. Choudhary, and C. Wolverton, Npj Comp. Mat. **2**, 16028 (2016).
[57] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, J. Comput.-Aided Mol. Des. **30**, 595 (2016).
[58] S. Dieleman, K. W. Willett, and J. Dambre, Mon. Not. R. Astron. Soc. **450**, 1441 (2015).
[59] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow, arXiv:1612.04642.
[60] S. Dieleman, J. De Fauw, and K. Kavukcuoglu, arXiv:1602.02660.
[61] W. Kohn, Int. J. Quantum Chem. **56**, 229 (1995).
[62] S. A. Kucharski and R. J. Bartlett, J. Chem. Phys. **97**, 4282 (1992).
[63] https://doi.org/10.4224/PhysRevA.96.042113.data.
[64] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, J. Mach. Learn. Res. **12**, 2825 (2011).

3.3 Atoms as pictures

In the previous work, we trained a neural network to predict properties of "on-lattice" models, that is, a model where the individual features (or spins) are confined to a precise, regular lattice. In materials science applications, atoms are indeed confined to a lattice; aluminum, copper, and gold atoms are arranged in face-centered-cubic lattice structure. In fact, it is the crystal structure that determines much of the macroscopic properties of a material. For example, graphite, graphene, diamond, and even charcoal are all composed of solely carbon atoms, with the various structural, electrical, and thermal properties owing to the arrangement of atoms.

Thus on-lattice models have the limitation that they can only represent pristine crystals in a single lattice structure. Furthermore, interesting properties arising from defects in the crystal, such as vacancies or grain boundaries are consequently impossible to represent.

"Convolutional neural networks for atomistic systems" introduces a representation of atomic systems, essentially representing a configuration of atoms as a grayscale image, with Gaussian wells centred on each atom. This representation would go on to be used in several other works as a way to represent two-dimensional atomic structures as input for neural networks.

Disclaimer: *The following section is borrowed heavily from "Convolutional*

Neural Networks for Atomistic Systems" (Ryczko, 2017) [3]. As I contributed to this work, but did not serve as the primary author, I have included only my direct contributions that are relevant to the remainder of this dissertation.

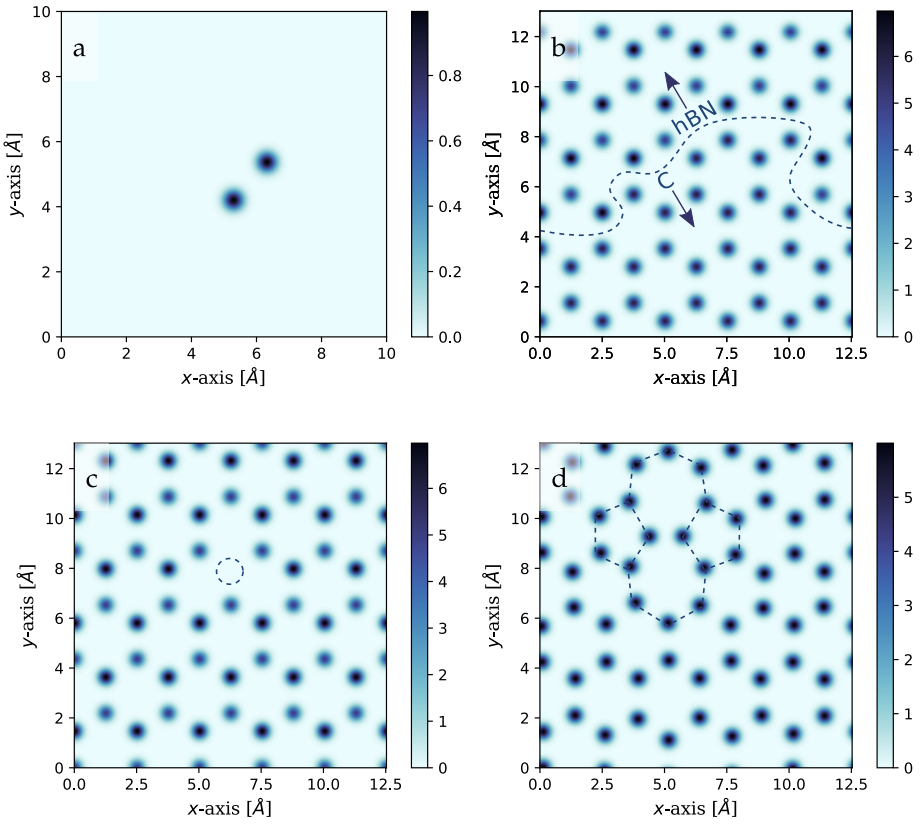


Figure 3.1: Four example atomic configurations represented as intensity maps (images). Figure reproduced from [3] with permission. a) two identical atoms, b) a composite hexagonal boron nitride-graphene sheet. The dashed line denotes the boundary between hexagonal boron nitride (hBN) and graphene (C). c) A hBN sheet with a point defect (vacancy) highlighted by a dashed region. d) A graphene sheet with a Stone-Wales defect highlighted by dashed lines.

Since we ultimately wish to use a deep convolutional neural network for modelling atomic systems, which exploits spatial structure in the input data,

we decided to represent our atomic configurations as approximations of the nuclear potential evaluated on a real-space mesh. While a Coulomb potential is the initial obvious choice, we used an atom-centered Gaussian representation to avoid the diverging Coulomb singularity. We evaluate our function on a real-space grid, with the value at point (x, y, z) given by:

$$V(x, y, z) = \sum_{i=1}^{N} Z_i \exp\left(-\frac{[(x-x_i)^2+(y-y_i)^2+(z-z_i)^2]}{2\gamma^2}\right) \tag{3.1}$$

where x_i, y_i, z_i are the coordinates of atom i with atomic number Z_i. The summation is taken over all N atoms in the system. We chose $\gamma = 0.2$ Å as the width of the Gaussian wells, consistent with Brockherde *et al.* [139]. For the two dimensional images, the z coordinate is not included (i.e. $z = z_i = 0$).

This results in single-channel images (i.e. intensity maps) representing the atomic configurations. Some examples are shown in Figure 3.1. Since this representation is defined in continuous real-space, many interesting defects, such as vacancies (Figure 3.1c) and Stone-Wales defects (Figure 3.1d) can be represented. Additional defect types such as substitutions, or even adsorption (by advancing to a three-dimensional representation), can be captured by this representation.

3.4 Going bigger

Convolutional neural networks are commonly described as translationally-invariant network structures. This is true in a technical sense: convolution *operations* are indeed translationally invariant, and the benefit of using convolutional operations is effective weight-sharing for feature detectors over the spatial dimensions of the input data. In practice, however, it is common to include one or more fully-connected layers, which serve as a final "decision layer". Including such layers anywhere in the network makes the network as a whole no longer translationally invariant. It imposes a restriction as to the size of the input data that can then be processed by the network; a network with fully-connected layers can then only be evaluated on inputs of the same size as the data on which it was trained. In computer vision, this is rarely an issue. If the goal of the network is image classification, the image can be cropped and scaled arbitrarily, and the network will still be able to identify the subject of the photo.

With physical systems, however, scale plays an important role, and furthermore, the concept of extensivity is one which a neural network should ideally be able to capture. Extensivity is a characteristic of physical properties. A physical property is extensive if, when the system is subdivided, the property is subdivided as well. The number of particles in a system is an extensive

property. When a sample is divided evenly into two subsystems, the number of particles in each subsystem is half of what it was initially. Temperature and pressure are examples of the opposite: intensive properties. These properties do no change as a result of macroscopic subdivision.

In "Extensive deep neural networks for transferring small scale learning to large scale systems", we propose an organizational structure for a neural network that performs domain decomposition in an intelligent way that is useful for modelling extensive properties. By breaking the input to the neural network up into overlapping (focus) regions and non-overlapping (context) regions, we present a way to evaluate a neural network on an arbitrarily large input size, enabling the computation of properties of extremely large systems, larger than those that could be handled with traditional methods. We demonstrate this by evaluating the total energy of a porous graphene sheet comprised of over thirty-million atoms, to the accuracy of state-of-the-art computational chemistry methods (density functional theory).

Extensive deep neural networks is an important milestone in the handling of different length scales with neural networks in physics, and extending trained models to large scale systems.

Author contributions: Kevin Ryczko generated and assisted in generating the hexagonal sheet and porous graphene data sets, respectively. Additionally Kevin Ryczko ran the Monte Carlo simulation used to generate Figure 13,

and developed an independent EDNN implementation to assist in the validation and debugging of the method. Iryna Luchak contributed extensively to the design of the methodology, and orchestrated experiments during the development of the method. The EDNN technique was mostly Isaac Tamblyn's idea. Kevin Ryczko, Iryna Luchak, Chris Beeler, and Isaac Tamblyn all participated in revisions of the manuscript and experimental discussions during the project. Adam Domurad wrote a very preliminary code implementation.

Chemical Science

EDGE ARTICLE

View Article Online
View Journal | View Issue

Cite this: *Chem. Sci.*, 2019, **10**, 4129

All publication charges for this article have been paid for by the Royal Society of Chemistry

Extensive deep neural networks for transferring small scale learning to large scale systems

Kyle Mills, *[a] Kevin Ryczko,[b] Iryna Luchak,[c] Adam Domurad,[d] Chris Beeler[a] and Isaac Tamblyn[abe]

We present a physically-motivated topology of a deep neural network that can efficiently infer extensive parameters (such as energy, entropy, or number of particles) of arbitrarily large systems, doing so with $\mathcal{O}(N)$ scaling. We use a form of domain decomposition for training and inference, where each sub-domain (tile) is comprised of a non-overlapping focus region surrounded by an overlapping context region. The size of these regions is motivated by the physical interaction length scales of the problem. We demonstrate the application of EDNNs to three physical systems: the Ising model and two hexagonal/graphene-like datasets. In the latter, an EDNN was able to make total energy predictions of a 60 atoms system, with comparable accuracy to density functional theory (DFT), in 57 milliseconds. Additionally EDNNs are well suited for massively parallel evaluation, as no communication is necessary during neural network evaluation. We demonstrate that EDNNs can be used to make an energy prediction of a two-dimensional 35.2 million atom system, over 1.0 μm^2 of material, at an accuracy comparable to DFT, in under 25 minutes. Such a system exists on a length scale visible with optical microscopy and larger than some living organisms.

Received 14th October 2018
Accepted 28th February 2019

DOI: 10.1039/c8sc04578j

rsc.li/chemical-science

1 Introduction

Within the past decade, the fields of artificial intelligence, computer vision, and natural language processing have advanced at unprecedented rates. Computerized identification and classification of images, video, audio, and written text have all improved to the extent they are now part of everyday technologies. With the recent advances in hardware acceleration,[1,2] deep neural networks have been at the forefront of these developments due to their ability to perform "featureless-learning", automatically learning both the features and the mapping between raw data and quantities of interest.[3–5]

Machine learning methods are rapidly being adopted by chemists, physicists, and materials scientists, and have performed well at making predictions in the fields of dynamical mean-field theory, many-body physics,[6,7] strongly correlated materials,[8–10] phase transitions and classification,[11–15] and materials exploration and design.[16–23] Machine learning models have been shown to be of sufficient accuracy to provide fast and accurate chemical insights.[20,24–28]

Convolutional deep neural networks have been used to predict the kinetic energy of hydrocarbons and were successful in reproducing the Kohn–Sham potential energy surfaces,[29] and have been used to classify reciprocal-space diffraction patterns for crystal lattices.[30] Deep neural networks have proven their classification power in astronomical applications[31] and particle physics applications[32–34] but have yet to be widely adopted throughout the physics community for accurate numerical predictions. There are a growing number of methods being proposed to capture relevant chemistry within representations of atomic environments[35] and a consensus is forming calling for the need to incorporate physics into network design and utilize the physics of the underlying problem to motivate the use of specific network structures and techniques.[36–38] The work of Brockherde *et al.*[39] focuses on low dimensional systems and small molecules. In addition their architecture is specialized to either work with the external potential or electron density using kernel ridge regression. These models are highly specialized and do not do any sort of energy partitioning. Their respective runtime is based on the computational complexity of kernel ridge regression for training. Depending on the dimensionality of the input, we have found that kernel ridge regression requires large amounts of RAM, whereas the required memory is less for deep neural networks (DNNs).

DNNs have the ability to replace both classical[40] and quantum mechanical operators.[41,42] In comparison to other machine learning methods, convolutional deep neural networks prevailed as the most accurate and best-scaling

[a]University of Ontario Institute of Technology, Oshawa, Ontario, Canada. E-mail: kyle.mills@uoit.net; isaac.tamblyn@nrc.ca
[b]University of Ottawa, Ottawa, Ontario, Canada
[c]University of British Columbia, Vancouver, British Columbia, Canada
[d]University of Waterloo, Waterloo, Ontario, Canada
[e]National Research Council Canada, Ottawa, Ontario, Canada

method for all but the most simple cases.[41,43] Deep neural networks were able to learn the mapping from spin configuration to energy for multiple cases of Ising-like classical spin models. For the case of a confined quantum particle, a deep neural network successfully learned the energy of the ground state, first excited state, and kinetic energy.[41] A similar approach was able to map the structure of two dimensional hexagonal crystal lattice energies computed within the density functional theory framework.[42] All of this was accomplished *via* the aforementioned "featureless" deep learning; the network was presented with raw spatial data, without any preliminary attempt at manual feature selection.

Traditionally with deep neural networks, the training process employed is not transferable to systems of arbitrary length scales. In practice, this means that for square, $L \times L$ lattice systems, a model trained on 4×4 configurations can not be used on an 8×8 cell (or *vice versa*) without retraining at least part of the network. With smaller or larger inputs impossible, this size limitation is clearly a major shortcoming of deep neural network techniques. Beyond the practical limitations, from a fundamental standpoint it is unsatisfying to use a model that has no concept of extensivity.

A physical property is extensive if it can be divided among subsystems. A common example is the number of particles in a system. When a sample is divided evenly into two subsystems, the number of particles in each subsystem is halved. This is in contrast to intensive quantities such as temperature that are unchanged by subdivision or addition of subsystems.

Maintaining extensivity has not been a focus of machine learning researchers, as in traditional vision- and audio-based applications of deep learning, extensivity is not a common requirement. Most classification problems (*e.g.* identification of an animal or shape in an image) are invariant to the physical dimensions of an image (*e.g.* number of pixels that comprise a cat). Indeed, absolute scale is not normally recorded in a photograph, and therefore scale invariant models are necessary and commonplace. Furthermore, photographs, handwriting, and audio recordings too large to be processed by the deep neural network can be resized, cropped, and segmented without destroying the features necessary to make a prediction;[44] there is no absolute spatial scale upon which the label of interest depends.

In a physical measurement or simulation however, the physical scale of a pixel matters critically. Consider the case of an X-ray diffraction experiment where the interference pattern recorded at the detector depends strongly on the wavelength of scattered light and the physical length scale over which the signal is collected. It is not possible to reconstruct the signal properly unless such parameters are known and are consistent. Extensivity is critical in describing chemical systems; configuration interaction with single and double excitations (CISD) is infamous for its lack of extensivity.

In this work, we propose a general method that preserves the extensivity of physical quantities, and also accommodates arbitrary input size. We propose a new deep neural network structure, which once trained, can operate on (effectively) arbitrary-sized inputs and length scales while maintaining the physical requirement of extensivity. Unlike atom-centred approaches, we avoid the problem of energy assignment or projection onto specific atoms by forcing the neural network to automatically learn by viewing the entire structure at once.

We call our approach Extensive Deep Neural Networks (EDNNs), employing domain decomposition to solve the problem of operator evaluation across length scales. Although domain decomposition techniques have a long history in computer simulation and modelling, here we have taken a new approach and allow the model itself to identify and self-optimize the overlap of tiles at domain boundaries.

Previous work[45] has identified the necessity of extensivity.[46,47] Our method is sufficiently general to allow for applications to any system in which extensivity holds, such as the spin and atomic systems we demonstrate in this work and even the charge density (*e.g.* a scalar field representing an extensive quantity). Furthermore, our method results in a model that can be evaluated on arbitrarily large system sizes, which we demonstrate.

2 Extensive deep neural networks

The overall objective of an EDNN is to learn the mapping between an input structure and one or more extensive properties, ε (*e.g.* total energy, entropy, magnetization, particle number, charge, *etc.*). To date, our input structures have consisted of continuous, regular, real-space grids, analogous to grayscale images.

Now, one might ask: "If the property we wish to predict is extensive, can we not just split the input into blocks and add up the individual answers?" Fig. 1 provides an example. If the goal of the neural network is to count the number of dots in the box (left), then division into non-overlapping subsystems will indeed yield the correct answer. If, however, the task is to count the number of multicolored pairs (right), the process is not so straightforward, and subdivision without accounting for the boundary yields erroneous answers.

The spatial extent over which features in the configuration influence the value of an operator is known as locality. In general, operators (such as the number operator, Hamiltonian,

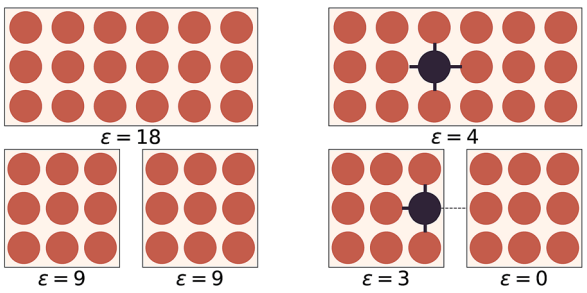


Fig. 1 On the left, the counting operator is local and the sum of the operator applied to individual subsystems results in the same answer as the operator applied to the complete system. On the right, the semi-local nearest-neighbour operator (*i.e.* the count of the number of black-red neighbours) cannot be applied to subsystems separately and then summed.

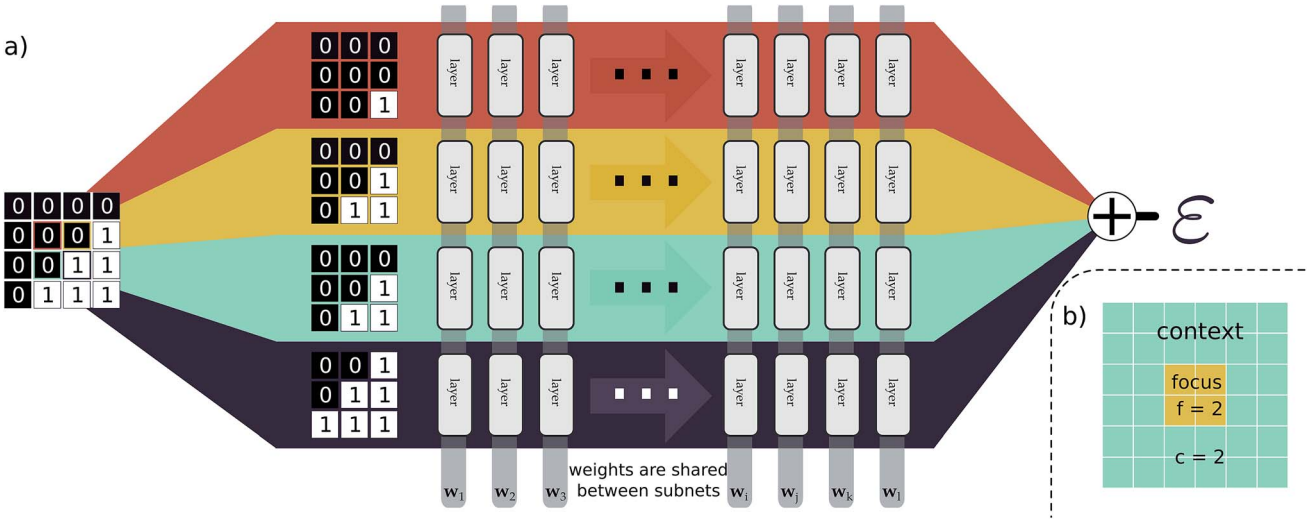


Fig. 2 An input example is decomposed into four tiles, with each tile consisting of a focus and context region. (a) As a pedagogical example, we expand 4 adjacent tiles comprising a generic binary grid. For this case both the focus and the context are unit width, resulting in 3 × 3 tiles. The tiles are simultaneously passed through the same neural network (*i.e.* the same weights). The individual outputs are summed, producing an estimate of ε, an extensive quantity. When training, the cost function is assessed after this summation, forcing the weight updates to consider all input tiles simultaneously. In (b), we show an example tile with a focus of $f = 2$, and a context of $c = 2$. The optimal selection of f and c depend on the physical length scale of the target (learned) function.

magnetization, *etc.*) may be described as local, semi-local, or fully non-local. In the density functional theory framework,[48,49] exchange–correlation functionals are often identified in these categories. The local density approximation is considered local, generalized gradient approximations like PW91 [50] and PBE[51] considered semi-local and some hybrid functionals (*e.g.* B3LYP[52,53]) and exact exchange considered non-local.

We define l to be the length scale of an operator's locality. For example, in the counting example above, the number operator is fully local ($l = 0$), as computing ε requires only local knowledge. The nearest-neighbour example is non-local with $l = 1$, meaning knowledge of the surrounding region is necessary to make a prediction. The gradient operator using a second-order finite difference method is an example of a semi-local operator with $l = 2$.

For many systems, such as the Coulomb ($1/r$) interaction, there is no hard cut off, but typically one expects the importance of a feature to diminish as the distance from the feature increases. For example, in a material, the screening environment (*i.e.* the importance of many-body effects) has a strong influence over how quickly this attenuation occurs. In metals it occurs quickly, but in large band-gap insulators the falloff is much more gradual. Even though quantum mechanics involves fully non-local operators, it has been noted that matter is, in practice, near-sighted.[54]

This idea of operator locality is the primary motivation for the subdivision technique used in EDNNs: an $L \times L$ training example is divided into $N = L^2/f^2$ non-overlapping regions of size $f \times f$. We call these regions focus. Then to each focus region, we provide overlapping context of width c. Each of these N tiles (Fig. 2b) of size $(f + 2c) \times (f + 2c)$ is then fed into an identical neural network (Fig. 2a), and the N individual outputs are summed to impose the extensivity of the operator. The loss is computed with respect to this final, summed value, and

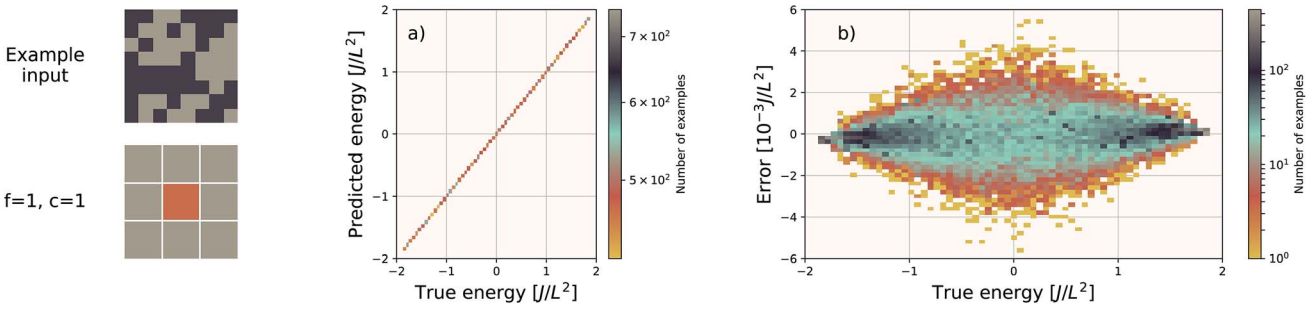


Fig. 3 Performance of an EDNN tasked with learning the energy E operator for the 8 × 8 Ising model. Since E is semi-local ($l = 1$), $f = c = 1$ is an optimal configuration. Additional information in the form of a larger context region does not help the network predict values, and in fact makes the training more difficult, as the network must learn to ignore a significant amount of information. (a) Predicted *vs.* true energies (per spin) for optimal model. (b) Error (predicted − true energy) *vs.* true energy for optimal EDNN model.

backpropagation is used to update the weights. In a normal domain-decomposition technique, some method to compensate for the inherent double-counting of the overlapping context regions would be necessary, however with EDNNs, we leave the task of rectifying this double-counting to the deep neural network itself; it must somehow learn to partially ignore the overlapping context regions.

2.1 EDNN input, topology, and training

Our explanation of EDNNs was very general, with no information about the neural network we used, the loss function employed, or the representation of the input data used. This is intentional, as the crux of the EDNN technique is the way in which the input data is subdivided, and the individual contributions summed prior to backpropagation. In practice, creating an EDNN requires only slight (albeit fundamental) additions to the neural network topologies commonly used. In fact, there is no strict requirement that any neural network be used within the EDNN framework at all; any supervised machine learning model can be used. Furthermore, the technique can be applied to any representation of a spatially multidimensional field, provided spatial correlations are captured and consistent across the input representation, and the quantity of interest is extensive. This means that quantities such as entropy, magnetic moment, and electron density (in addition to energy as demonstrated in this work) are ideal candidates for the EDNN technique.

As for the "brain" of the model, we chose to demonstrate the technique with a neural network; the neural network within the EDNN can be of arbitrary complexity. We have made use of both deep convolutional neural networks as well as fully-connected artificial neural networks, depending on the complexity of the underlying problem. Since the input to the neural network is much smaller than the overall example, network architectures that would normally be prohibitively expensive become tractable with EDNNs (*e.g.* within the EDNN framework it would be possible to use a fully-connected artificial neural network to process a many-megapixel input).

In our case, we have focused on neural networks. Details of training hyperparameters are provided in Table 1. The fully-connected artificial neural network used in training the Ising model is comprised of a fully-connected layer of size $32(f+2c)^2$ (note that $(f+2c)$ is the size of a single tile), followed by two fully-connected layers with 64 outputs, which finally feed into a fully-connected layer of size 1. The convolutional deep neural network used to train the DFT models is constructed from 13 convolutional layers and two fully-connected layers. The first 2 layers are reducing and operate with filter sizes of 3×3 pixels. Each of these reducing layers operates with 64 filters and a stride of 2×2. The next 6 layers are non-reducing, meaning they have unit stride and preserve the resolution of the image. Each of these non-reducing layers operates with 16 filters of size 4×4. The ninth convolutional layer operates with 64 filters of size of 3×3 and a stride of 2×2. The last four convolutional layers are non-reducing, and operate with 32 filters of size 3×3. The final convolutional layer is fed into a fully-connected layer of size 1024. This layer feeds into a final fully-connected layer with a single output. The contribution from each tile is summed, and the loss is computed as the mean-squared error between this value and the true value.

We note that parallel branching within neural networks is a common technique, usually taking one of two forms. The first technique, which is used by *e.g.* GoogLeNet (a.k.a Inception),[55] uses repeating modules of parallel convolutional layers. Ref. 44 uses a similar approach, with multiple preprocessing techniques feeding a variety of data representations through many branches of a neural network architecture. Each branch of these neural networks has its own set of weights, and learns different features. Ultimately the output from each branch is concatenated to produce an ensemble of learned features that is subsequently fed into a decision layer.

The second approach employs a single set of weights, shared across the parallel branches such as in ref. 56. This technique facilitates efficient parallelization of the neural network training as each branch can be evaluated on separate hardware with little communication between devices. When training in parallel, the gradients from each separate branch are averaged, and the weights of each branch are updated synchronously. This effectively leads to a multiplicative speed-up in training and inference.

EDNNs are fundamentally different from these approaches, however, since in contrast to the former, the contribution from each branch is summed and not concatenated, and in contrast to the latter, the gradients used to update the weights are computed after the extensivity-imposing summation.

2.2 Focus and context

EDNNs introduce two new hyperparameters to the design of the neural network: the focus and the context sizes. The following considerations can ease in the choice of an appropriate focus and context:

- $c \approx l$ in order to provide sufficient context to the network, but should not be too much greater so as to introduce

Table 1 Parameters used to train the various example models. Mean Squared Error (MSE) is used as the loss function in all examples

Model	Total training examples	Batch size	Network	Optimizer	Learning rate	Loss	Epochs
Ising model	100 000	2000	DNN	Adam[57]	0.0001	MSE	500
Hexagonal sheets	18 515	100	CNN	Adam	0.00001	MSE	500
Porous graphene	501 473	64	CNN	Adam	1×10^{-6}	MSE	500

redundant computations into the network. Context too large is not only inefficient for evaluation, but also makes the weight-optimization process more challenging as the network must learn to ignore a larger fraction of the input signal.

• Choosing f is a balancing act between parallelizability and overall computational cost. Minimizing f results in more tiles, which can be computed independently and thus parallelized efficiently. On the other hand, small focus leads to a greater overall computational demand, as for every focus region, there are overlapping context pixels that are being computed multiple times.

• A quantitative comparison of different focus and context pairs is difficult, as varying these parameters consequently changes the architecture of the neural network (*e.g.* number of weights, or required number of layers to reduce the image to a predetermined size through strided convolutions), modifying the fitting capabilities of the network.

We note that as the locality length scale of an operator grows, the optimal EDNN approaches a single tile being processed by a normal deep neural network; the context region is simply periodic padding. This is because for fully non-local operators, it is physically impossible to divide the problem up in the style of EDNNs since the full volume is needed to make an inference. Such systems are rare in practice, thankfully. EDNNs therefore represent a framework that can naturally handle the full continuum of possible screening environments. EDNNs can describe all phases of the electron gas. We note that when dealing with operators that have large values of l, it is often useful to recast the problem in reciprocal space, and such an approach could be useful too with EDNNs.

3 Results

As illustrative examples of the method, we have trained an EDNN on three systems: (1) the ferromagnetic Ising model and (2) quantum mechanical total energy calculations (within the density functional theory framework) for (a) hexagonal systems (*e.g.* boron nitride, graphite, and heterostructures of the two), and (b) porous graphene sheets.

3.1 Example: the Ising model

The Ising model is a two-state spin (σ) model with $\sigma = \pm 1$ with a total energy Hamiltonian

$$\hat{H} = -J\sum_{\langle i,j\rangle}\sigma_i\sigma_j \qquad (1)$$

where J is the interaction strength, and the summation is computed over nearest-neighbour pairs ($\langle i,j\rangle$). For $J = 1$ the system is ferromagnetic; it is favourable for neighbouring spins to align. Application of EDNN to the Ising model is particularly instructive because the locality length scale of the Hamiltonian operator is known explicitly; as previously discussed, it is an $l = 1$ operator. The nearest-neighbour interaction means that $c > 1$ provides no additional information to the EDNN. Including this data makes the task more difficult, as the network must learn to completely ignore these features. As the size of the context region grows beyond the locality length scale of the operator, the learning process is less efficient. Our optimal EDNN trained on the Ising model, using raw, binary spin values ($\sigma \sim \{1,-1\}$) as input, achieves a MAE (median absolute error) of $0.028J/L^2$ on the testing set, sufficiently accurate[40] to reproduce the finite temperature phase transition for which the model is so well-known. In comparison, the Ising models subdivided using unit focus with $c = 2$ and $c = 3$ produce an error of $1.090J/L^2$ and $14.816J/L^2$, respectively. All three EDNNs were trained for the same number of iterations and since the layer size of the artificial neural network is dependent upon the input size, larger context means significantly more parameters in the neural network. With $f = 1$, $c = 3$ the neural network contains over 7 times the parameters as the $f = 1$, $c = 1$ neural network and therefore is much more difficult to optimize. This is another motivation for considering carefully an appropriate choice for f and c.

3.2 Example: density functional theory

3.2.1 Hexagonal sheets. For comparison with previous work, we reuse a previously reported dataset[42] of 2d crystalline structures. This dataset consists of 26 449 structures of crystalline and defect (missing atom) hexagonal surfaces. The technique of EDNNs does not depend on the atomic representation used as input to the neural network, provided the spatial structure is properly represented; we use the previously established atomic representation using an image generated from the function

$$V(x,y) = \sum_{i=1}^{N} Z_i \exp\left(-\frac{\left[(x-x_i)^2 + (y-y_i)^2\right]}{2\gamma^2}\right). \qquad (2)$$

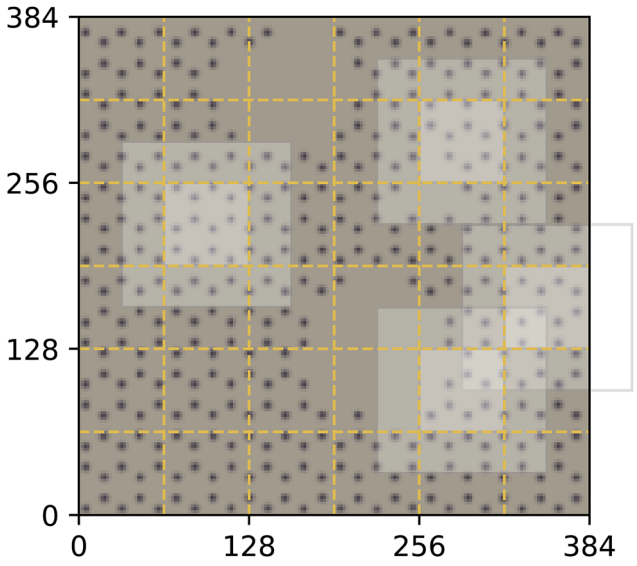


Fig. 4 Decomposition of input image for the quantum mechanical density functional theory calculation using $f = 64$ and $c = 32$. Four tiles consisting of a focus region and context region are highlighted. Overlap in the context region is by design and the EDNN must learn to ignore this overlap in the final reduction of the extensive quantity.

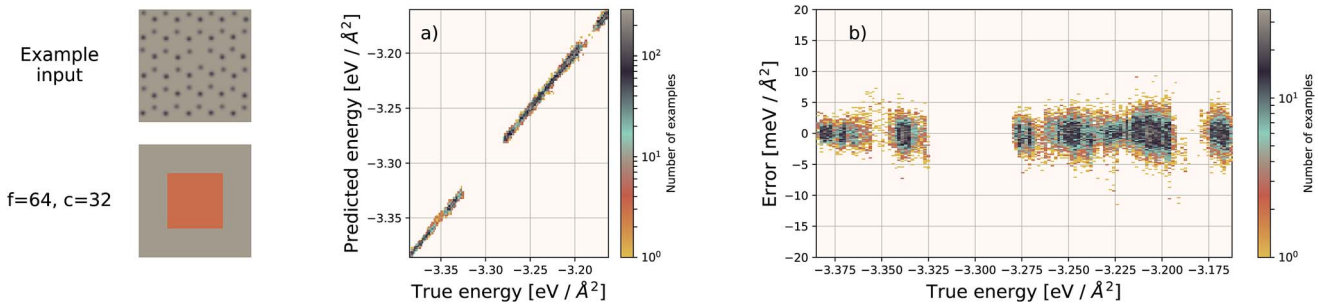


Fig. 5 Left: an example graphene sheet. (a) Performance of our EDNN model on a testing set (predicted *vs.* true). (b) Error ((predicted − true) *vs.* true) for the EDNN model trained on the total energy as calculated through the density functional theory framework.

Here x_i, y_i are the coordinates of the i^{th} atom with atomic number Z_i. $\gamma = 0.2$ Å was used as the width of the atomic potential wells, consistent with Brockherde *et al.*[39] (Fig. 4). This function is evaluated on a 256 × 256 grid representing a 12.5 Å × 13 Å unit cell containing 60 atoms arranged in a hexagonal lattice. The dataset contains both graphene and hexagonal boron nitride, with and without defects.

For these quantum mechanical systems (computed within the generalized gradient approximation of the density functional theory[48,49] framework), it is not possible to determine a specific value of l. Within the hierarchy of approximations, the method we used to compute the total energy (PBE[51]) is considered to be a semi-local approximation, since the exchange–correlation potential includes only gradients of the charge density. Other terms within the total energy are fully non-local (although they are subject to screening by the electron gas). Nonetheless, total energy is an extensive quantity and again the EDNN performs favourably compared to previously reported (non-EDNN) results.

Using $f = 64$ and $c = 32$, we achieve a MAE of 1.122 meV Å^{-2} on our test set after 500 training epochs, notably better than the MAE of 2.529 meV Å^{-2} on a traditional (non-EDNN) network (Fig. 5).[42] For this choice of (f, c) our input representation is divided into 16 tiles, enabling an inference speed-up factor of 16 due to the ability to calculate the contribution from each tile in parallel. This is on top of the inherent speed-up of evaluating the EDNN compared to DFT. A conservative estimate for this latter speed-up is on the order of 1 million in terms of CPU-hours.

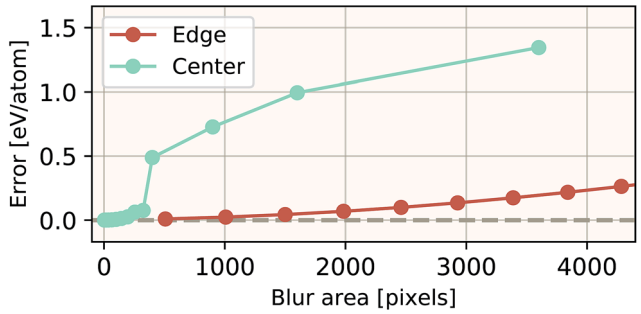


Fig. 6 Resilience of an EDNN to the addition of obfuscating Gaussian blur. When a constant amount of information (constant area) within the tiles is blurred, examples that had context area blurred result in more accurate inference than examples that had focus blurred. This is evidence that the EDNN is learning to ignore the context regions.

The ability of an EDNN to learn to ignore, or compensate for redundant context can be explored by measuring the performance of the model when information is partially obfuscated through the application of a Gaussian blur to select regions of the input. In Fig. 6, we plot the performance of the network when blur is applied within the context region (edge) during inference. As expected, when the blur is applied at the periphery of the context region, the network reports very similar values for ε as when there is no blur present. As the blurring encroaches on more context, the predictions become poorer; the neural network is evidently learning to ignore the context region. This is to be expected, as the data will appear again in the focus region of another tile. This is how double-counting is avoided. When a small area in the center of the focus region is blurred, the neural network is able to make accurate predictions, (likely due to the limited amount of information being lost), but as the extent of the blur increases within the focus region, the accuracy of inference suffers greatly.

3.2.2 Porous graphene sheets. As a more challenging example of the applicability of this technique, we developed a dataset of porous graphene sheets. We generated 3137 starting geometries by randomly removing varying numbers of regions of various sizes from pristine graphene sheets of size 35 Å × 35 Å. We separated the starting configurations into one set of 349 starting configurations, reserved for testing the ability of the trained EDNN to generalize to data that it has never seen (validation), and the remaining 2788 configurations were used for generating a training set.

We ran molecular dynamics at a temperature of 1000 K using forces obtained through the density functional theory framework (using VASP[58–61] and the PBE[51] functional), collecting configurations from the molecular dynamics trajectories as training. In all, we collected 501 473 training configurations and 60 744 testing configurations.

We use the same Gaussian-based input representation, and the same deep convolutional neural network architecture as the "hexagonal sheets" investigation. The larger supercell size of the porous graphene sheets requires discretization on a larger grid; we chose a 384 × 384 grid. This

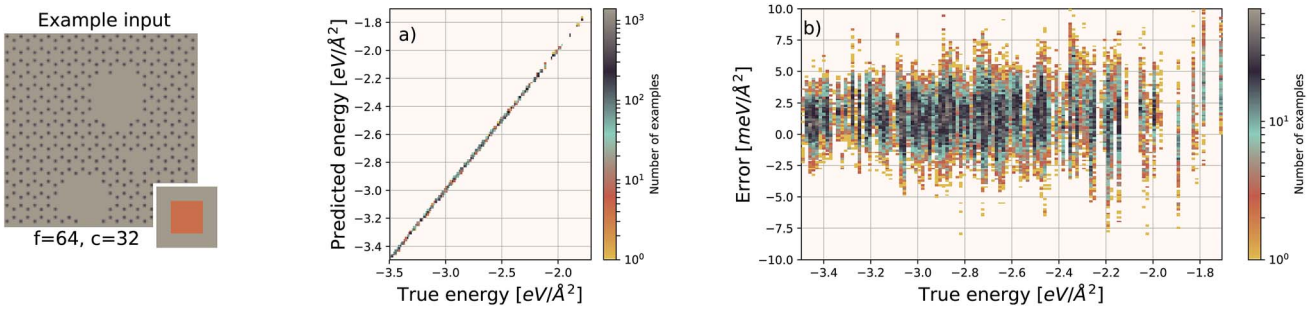


Fig. 7 Left: an example porous graphene sheet. (a) The true (DFT) *vs.* predicted (EDNN) total energy in eV Å^{-2}. The tight clustering along the diagonal indicates the EDNN performs well at predicting the total energy. (b) The error (DFT energy – EDNN energy), in meV Å^{-2}, is very close to zero.

results in more tiles required, but the training procedure remains identical in all other aspects. The results are shown in Fig. 7. The EDNN performs well with a median absolute error of 1.685 meV Å^{-2}.

3.3 Transferability to arbitrary input size

EDNNs have the capability of making predictions on input configurations of arbitrary size so long as the physical length scale (*i.e.* the real space extent of a pixel in the input representation), is preserved, and the input size remains an integer multiple of the focus region.

To demonstrate this, we used the neural network trained on 8×8 Ising configurations to make energy predictions of 128×128 Ising configurations sampled near the critical temperature. Without any further training, the median absolute error on these much larger configurations was $2.055J/L^2$. This is substantially larger than the 8×8 error, but this is to be expected. Since there is some error associated with the prediction of a given tile, it indeed makes sense that this error will scale with the extensivity of the system. In other words, the error relative to the absolute energy is still small. The predicted values and the prediction error is plotted against the true values in Fig. 8.

Additionally we test the DFT model trained on a 12.5 Å × 13 Å, 256 × 256 grid ($N = 60$ atoms) on several larger domains up to 62.6 Å × 65.1 Å (1024 × 1024 grid, $N = 1500$ atoms). During inference, the EDNN performs similarly well predicting energies of configurations larger than those in the training set, as seen in Fig. 9, and does so many orders of magnitude faster than conventional numerical methods. This is a powerful feature of EDNN, as one can generate a testing set of many training examples for significantly less computational expense, and then apply it to larger systems without the $\mathcal{O}(N^3)$ (or worse) scaling inherent in Kohn–Sham density functional theory. The evaluation of the EDNN scales as $\mathcal{O}(N)$. The fact that an EDNN can take a $\mathcal{O}(N^3)$ problem and map it to $\mathcal{O}(N)$ might seem suspicious at first. Recall though that HK DFT does scale linearly and that the polynomial scaling of Kohn–Sham DFT is due to the diagonalization of the Kohn–Sham Hamiltonian. We avoid such an evaluation during inference, and therefore we achieve scaling consistent with orbital free DFT.

As a proof of concept and to further demonstrate the exceptional scaling of the EDNN approach, we generated a porous graphene sheet comprised of 35.2 million atoms, with a supercell size of 1.0 μm^2. EDNN inference is trivially parallel, so using a custom distributed TensorFlow implementation, we were able to compute the total energy of the sheet using 448 cores across 16 nodes in 24.7 minutes. A "ground truth" DFT calculation at this scale is intractable, but based on the results on smaller-scale tests (Fig. 8 and 9) we can confidently conclude that the relative error is comparable to that of a DFT method. These results are shown in Fig. 10 and 11.

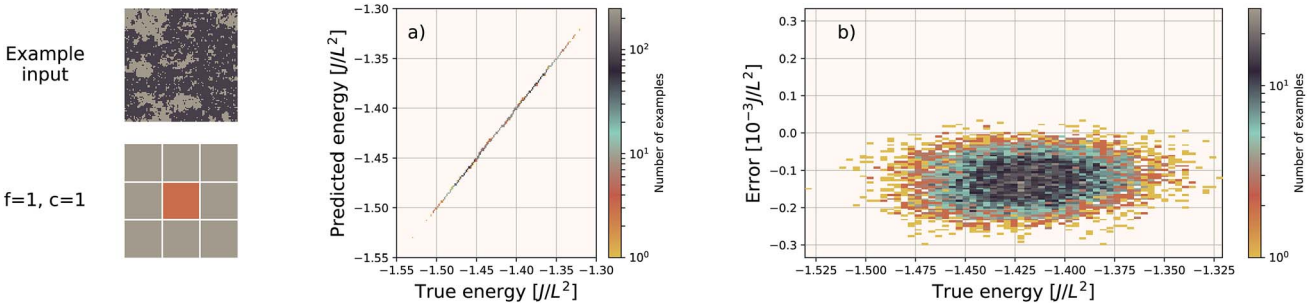


Fig. 8 An EDNN trained only on 8×8 Ising training examples is capable of making accurate predictions of the 128×128 Ising model near criticality. While the absolute error is higher at $2.055J/L^2$, the relative error is very small. While it appears the EDNN consistently overpredicts the energy, this is not an effect of large scale inference, but rather that the input configurations are from an energy window where the original EDNN also slightly overpredicted the energy. This is evident when compared to the appropriate region of Fig. 3b.

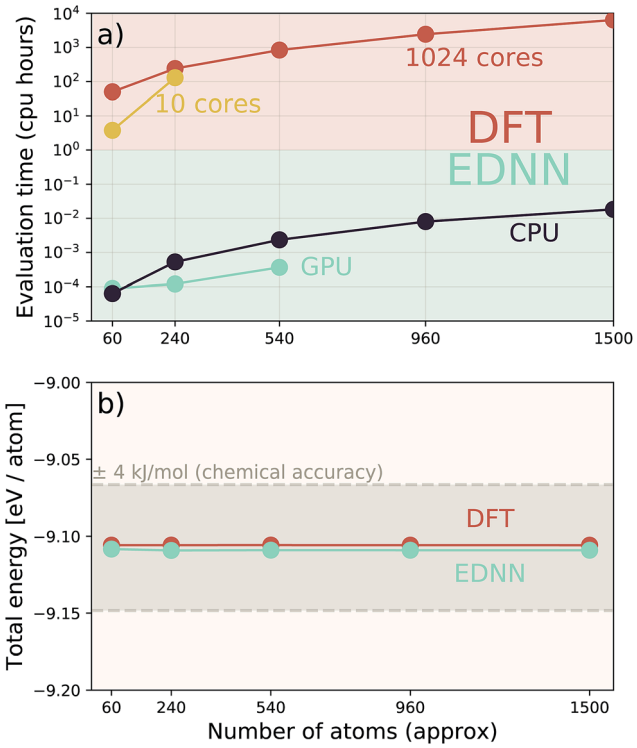


Fig. 9 A single EDNN was trained on a 12.5 Å × 13 Å unit cell. This trained model was used to make accurate predictions on larger unit cells not present in the training set. (a) The inference time for large systems was about 1 million times smaller than the equivalent density functional theory approach, with CPU evaluation performing better than GPU evaluation on large systems. (b) The resulting energy predictions are consistent within chemical accuracy of 1 kcal mol^{-1}. The scale of the error can be expected to scale linearly with the size of the system (*i.e.* $\mathcal{O}(N)$).

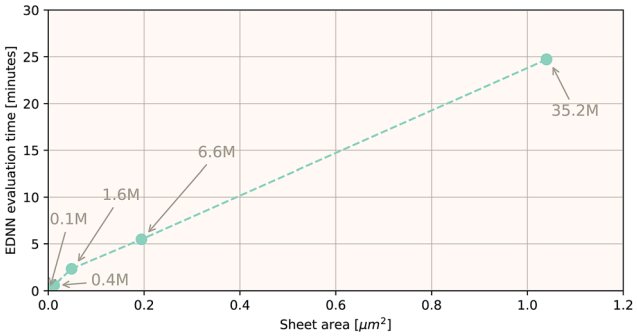


Fig. 10 Using the model trained on many small porous graphene sheets, we used a multi-node, distributed TensorFlow implementation to make predictions on large sheets. The model evaluation time scales linearly with the cell area (and thus, under the assumption of homogeneous density, the number of atoms). The annotations refer to the number of atoms in the configuration. The EDNN allows for total energy calculations at DFT accuracy of more than 1.0 μm^2 of material in 24.7 minutes (Fig. 12). Importantly, the model was trained on a dataset of configurations consisting of only around 500 atoms, and therefore collection of training data does not require accurate simulation of large configurations. All EDNN evaluations were carried out on a 20-node cluster with 28 cores per node.

3.4 EDNN in use

We have demonstrated that EDNNs can be used to accurately learn the mapping from atomic coordinates to energy. How though, do EDNNs perform when used as the energy evaluation function in an actual simulation? To investigate this we performed a Metropolis Monte Carlo (MC) simulation of pristine graphene, the same 12.5 Å × 13 Å unit cell used in the dataset from Section 3.2.1 above. During the MC calculation we use the EDNN to evaluate the energy of the atomic configuration. The accurate evaluation of energy is important within the MC framework, as the evolution of the atom positions depends exponentially on the accurate evaluation of energy. In Fig. 13, we plot the radial distribution function, $g(r)$ for the atoms in the EDNN (MC) simulation alongside $g(r)$ for a molecular dynamics (MD) simulation of pristine graphene using DFT (VASP). For the MD, we used the same VASP input parameters as the MD used to generate the EDNN training/testing dataset. Both simulations occur at a temperature of 1000 K. The two radial distribution functions are in close agreement, with the peaks in exact agreement. While there is a bit of discrepancy between the two functions, it is evident that the EDNN technique can be used to perform physically relevant simulations at a fraction of the cost of the quantum mechanical alternative; during the MC simulation the EDNN takes 0.39 seconds per energy-evaluation, while the DFT (MD) takes 9.9 seconds. The bulk of the EDNN calculation time is spent within our inefficient image representation construction, *i.e.* the evaluation of eqn (2). Since this is not the focus of this work, we have not yet optimized this evaluation, so considerably higher performance is possible in practice.

3.5 EDNN advantages

We note that EDNNs are particularly well suited for massive parallelization, particularly during inference, since neural network evaluation can be distributed across hardware; there is no need for communication between them until the final summation. Beyond scalability, EDNNs have the feature that they can operate on inputs of arbitrary shape and size (to within integer multiples of f). This is particularly useful for treating large-scale mesoscopic structures *in silico*.

Since the neural network of the EDNN only operates on a single tile at a time, EDNNs permit the use of more computationally intensive network architectures (*e.g.* fully-connected networks), that would normally be infeasible. Additionally EDNNs are well suited for Monte–Carlo sampling, as local updates would only require the re-evaluation of nearby tiles, not the entire configuration.

Under the EDNN framework, there is no requirement to assign energy to a particular region of space or atomic species, unlike atom-centred methods. Rather, the network is simply told that the extensive property applies to the entire system. This is important because it is extremely flexible; it allows for a seamless method that can learn quantum molecular mechanics, implicit solvation energies, entropy, *etc.* Furthermore EDNNs can operate on any spatial field, such as the electron density.

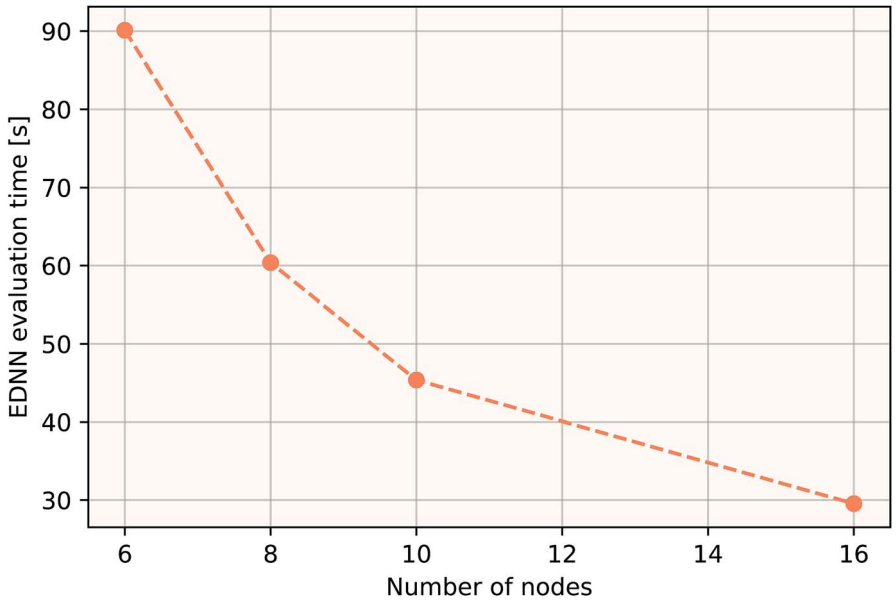


Fig. 11 Using the model trained on many small porous graphene sheets, we used a multi-node implementation of TensorFlow to perform inference on larger systems. At over 400 000 atoms, we achieve better-than-linear scaling, even with only typical gigabit ethernet interconnect. In theory, since the evaluation of an EDNN is perfectly subdivisible into separate parts, with the only communication cost incurred during the final summation, scaling to large system sizes should be parallel. In practice, overhead is incurred in the distribution of input data, but we achieve impressive scaling nonetheless.

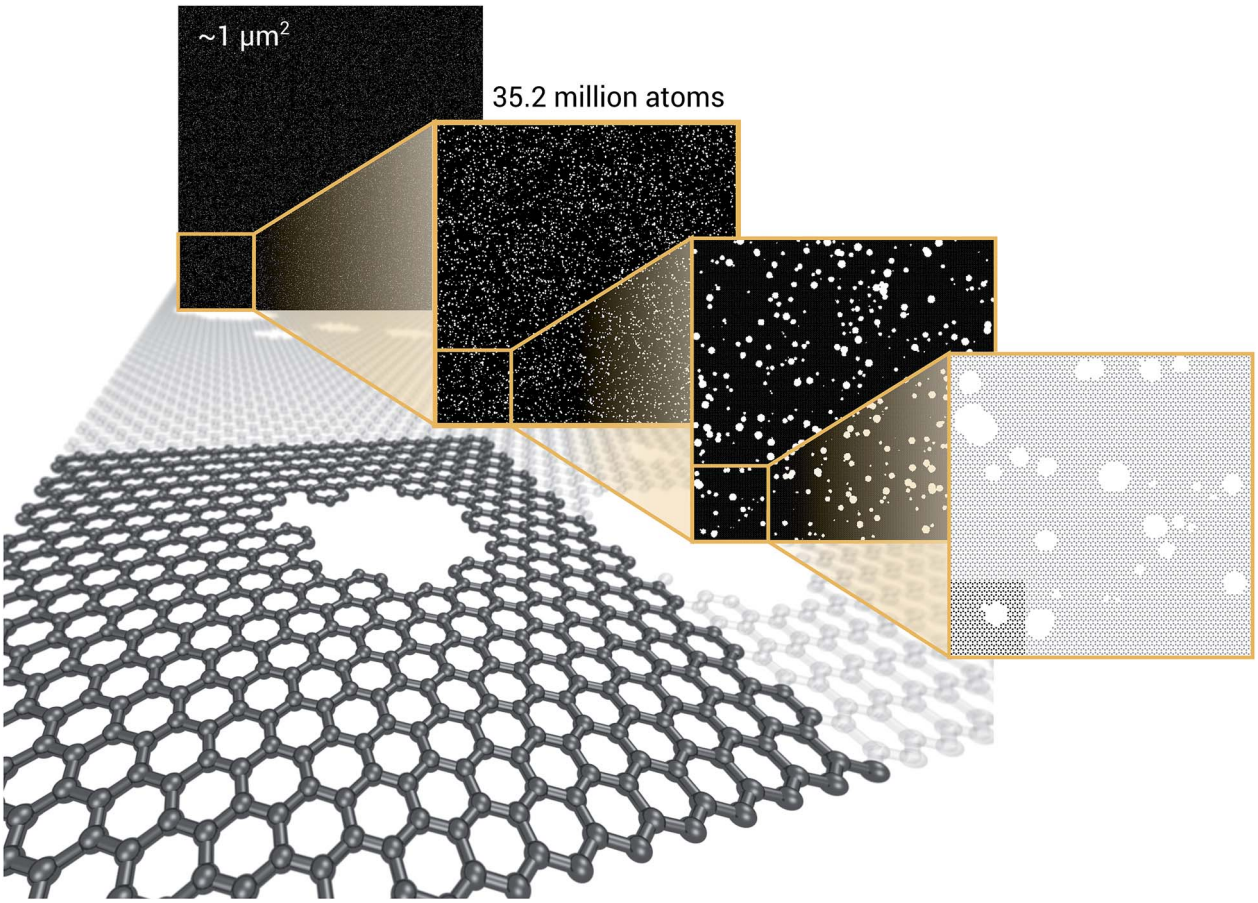


Fig. 12 We demonstrate that EDNNs can be used to make an energy prediction of a two-dimensional 35.2 million atom system, over 1.0 μm^2 of material, at an accuracy comparable to density functional theory, in under 25 minutes. Additionally, the evaluation of the neural network scales linearly with the number of atoms (assuming relatively homogeneous density), so this evaluation-time estimate can be driven lower with wider hardware configurations. Such a system exists on a length scale visible with optical microscopy and larger than some living organisms.

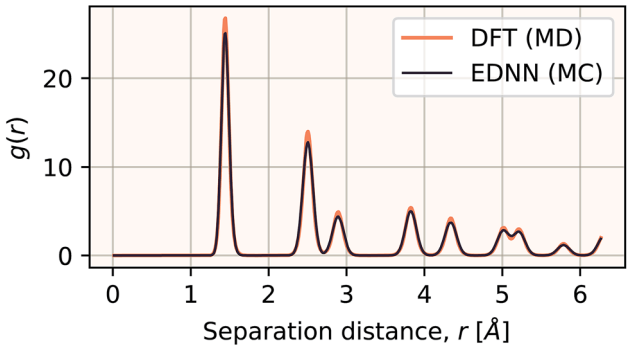


Fig. 13 The radial distribution functions $g(r)$ plotted for two calculations. The black line is $g(r)$ for a Metropolis Monte Carlo simulation using the EDNN as the energy evaluation function. The orange line is $g(r)$ for a molecular dynamics calculation using density functional theory as the energy evaluation criteria. Since the two methods differ algorithmically, a direct comparison is difficult, but we can see that both methods yield exactly the same peak positions, indicating the EDNN is capable of making predictions at an accuracy suitable for performing physical simulations.

EDNN can be thought of as a generalization of a complex, many-body force-field. Features are learned automatically by the EDNN, and domain decomposition is handled intrinsically. Traditional force-fields assume that extensive properties such as the total energy can be expressed as a many-body sum over interacting particles, and in some cases, an implicit solvation environment. While many different models exist, almost all share the common feature of expanding the total energy in terms of neighbour interactions, typically within a fixed cutoff radius. Bond lengths, angles, and partial charges are used as features, and the coefficients of the relative terms are trained against a fixed set of examples. Even methods designed for metallic environments (*e.g.* the embedded atom method) make use of structural "features" (*e.g.* the density of neighbours) which are then fed into a feature based decision algorithm. When used for atomistic modelling, EDNN accomplish the same task as a force-field without the requirement of hand selecting features. They are also extremely straightforward to implement in parallel, and do not require complex calculations of angles, dihedrals, feature vectors, or neighbour lists for efficient parallelization.

3.6 EDNNs capturing physics

At the most fundamental level, many-body interactions within a material are subject to screening. Screening occurs due to elementary excitations that occur within the system (*e.g.* electronic, rotational, *etc.*). Depending on the characteristic of the electron gas, screening effects can result in a rapid interaction decay length (*e.g.* as in a metal), or they may attenuate much more slowly. Screening and the various length scales at which its effects are observed are emergent phenomena.

EDNNs are built on the idea that interactions are screened at some length scale, but that *a priori*, the user does not know what it is. The training data itself encodes this length scale, and the network takes advantage of it. The generality of the concept and the implementation are why EDNNs are so useful; the physical property that permits the decomposition of the problem is actually revealed by the data itself.

On a similar note, (*i.e.* that the data should reveal the physics, rather than incorporating it *a priori*) relates to the question of invariance; there are currently two schools of thought about how symmetries should be built into models. Within the chemical literature, there is currently a strong bias toward enforcing symmetries within models first, and then developing methods that are constrained by those symmetries. This "symmetry first" view is generally not what has been the approach in the computer vision/artificial intelligence community. We are of the belief that symmetries (*e.g.* rotations) can be learned and should not necessarily be enforced. This is both from a pragmatic point of view (too many constraints on a network during the learning process can restrict it to local minima), and from somewhat of a philosophical point of view. In deep learning, features can and should be learned from the dataset itself. The way to teach a neural network a physical law is to provide it data from which it can learn.

4 Conclusion

We have demonstrated a new form of deep neural network, motivated by physics, that can operate on arbitrary sized input and physical length scales while maintaining the extensivity of properties inferred by the network. Networks of this form are particularly well suited to large-scale parallel inference, as the individual components of the input data can be computed independently of one another. We demonstrate the ability of EDNNs to learn extensive operators, such as the nearest-neighbour Ising Hamiltonian and the density functional theory total energy operator. The process of optimizing the focus and context hyperparameters provides physical insight into the interaction length scales of the physical problem. We demonstrate the efficiency of the EDNN approach in inferring properties of systems much larger than those on which it was trained. Finally, we demonstrate the ability of an EDNN to infer the total energy of a porous graphene sheet comprised of 35.2 million atoms, to DFT accuracy, in under 25 minutes. Although we have chosen to demonstrate three specific examples in the field of physics, the techniques and arguments that we present are quite general, and naturally apply to many problems in physics and image processing.

Conflicts of interest

There are no conflicts to declare.

Data and code

The porous graphene data set, as well as links to EDNN tutorials and code can be found at http://doi.org/10.4224/c8sc04578j.data.

Appendix

Let us formalize the EDNN technique.

The (two dimensional, single channel) EDNN takes as input a batch of images $\mathbf{x} \in \mathbb{R}^{N_B \times L \times L \times 1}$, where N_B is the batch size. EDNNs are not limited to 2-dimensional single-channel data and can be used for three (or more) dimensions and multiple channels.

We construct the input tensor $\mathbf{X}$ by taking a periodically-padded copy of $\mathbf{x}$ and performing a strided slice, starting from the origin with stride equal to the focus. We extract a patch of size $(f + 2c) \times (f + 2c)$, *e.g.*

$$\mathbf{X}_{b,i,j,d} = \mathbf{x}_{b,if-c:if+f+c,jf-c:jf+f+c,d},\ i,j \in [1 \ldots L/f]$$

meaning

$$\mathbf{X} \in \mathbb{R}^{N_B \times (L/f) \times (L/f) \times d \times (f+2c) \times (f+2c)}$$

In words, $\mathbf{X}$ is comprised of L^2/f^2 tiles of size $(f + 2c)^2$ d-channel pixels for each of the N_B images in a batch.

Each tile is passed individually through the approximation function (*e.g.* a neural network), which is parametrized by θ:

$$\mathcal{C}_{b,i,j} = f\left(X_{b,i,j}; \theta\right).$$

We call the $\mathcal{C}$ tensor the "tile contributions", that is, how much of the final answer is contained in each tile. The tile contributions are reduced through a summation, preserving only the batch dimension:

$$\hat{\varepsilon}_b = \sum_i^{L/f} \sum_j^{L/f} \mathcal{C}_{b,i,j}.$$

This vector, $\hat{\varepsilon}_b$ is the predicted extensive quantity from the EDNN, one entry for each example in the input batch.

The batch loss is computed as the mean-squared error between the predicted and the true value (*i.e.* the "label"):

$$\mathcal{L} = \frac{1}{N_B} \sum_{i=0}^{N_B} \left(\hat{\varepsilon}_i - \varepsilon_i\right)^2$$

This loss function is then minimized as in a normal neural network, using some form of gradient descent to tweak the parameters θ so that the prediction matches the true answer as closely as possible.

Acknowledgements

The authors acknowledge funding from NSERC and SOSCIP. Compute resources were provided by Compute Canada, SOSCIP, National Research Council of Canada, and an NVIDIA Faculty Hardware Grant.

Notes and references

1 S. Chetlur and C. Woolley, arXiv, 2014, **1–9**.
2 G. Lacey, G. W. Taylor and S. Areibi, arXiv, 2016.
3 Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama and T. Darrell, arXiv, 2014, 675–678.
4 D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel and D. Hassabis, *Nature*, 2016, **529**, 484–489.
5 V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, *Nature*, 2015, **518**, 529–533.
6 G. Carleo and M. Troyer, *Science*, 2017, **355**, 602–606.
7 X. Liang, W.-Y. Liu, P.-Z. Lin, G.-C. Guo, Y.-S. Zhang and L. He, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 2018, **98**, 104426.
8 O. S. Ovchinnikov, S. Jesse, P. Bintacchit, S. Trolier-Mckinstry and S. V. Kalinin, *Phys. Rev. Lett.*, 2009, **103**, 2–5.
9 L. F. Arsenault, A. Lopez-Bezanilla, O. A. von Lilienfeld and A. J. Millis, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 2014, **90**, 155136.
10 K. Ch'ng, J. Carrasquilla, R. G. Melko and E. Khatami, *Phys. Rev. X*, 2017, **7**, 031038.
11 E. P. L. van Nieuwenburg, Y.-H. Liu and S. D. Huber, *Nat. Phys.*, 2017, **13**, 435–439.
12 J. Carrasquilla and R. G. Melko, *Nat. Phys.*, 2017, **13**, 431–434.
13 R. B. Jadrich, B. A. Lindquist and T. M. Truskett, *J. Chem. Phys.*, 2018, **149**, 194109.
14 X. L. Zhao and L. B. Fu, arXiv, 2018.
15 D. Kim and D.-h. Kim, *Phys. Rev. E*, 2018, **98**, 022138.
16 T. Bereau, R. A. DiStasio, A. Tkatchenko and O. A. von Lilienfeld, *J. Chem. Phys.*, 2018, **148**, 241706.
17 A. G. Kusne, T. Gao, A. Mehta, L. Ke, M. C. Nguyen, K. Ho, V. Antropov, C.-Z. Wang, M. J. Kramer, C. Long and I. Takeuchi, *Sci. Rep.*, 2015, **4**, 6367.
18 S. Jesse, M. Chi, A. Belianinov, C. Beekman, S. V. Kalinin, A. Y. Borisevich and A. R. Lupini, *Sci. Rep.*, 2016, **6**, 26348.
19 P. V. Balachandran, D. Xue, J. Theiler, J. Hogden and T. Lookman, *Sci. Rep.*, 2016, **6**, 19660.
20 N. Artrith and A. Urban, *Comput. Mater. Sci.*, 2016, **114**, 135–150.
21 J. N. Wei, D. Duvenaud and A. Aspuru-Guzik, *ACS Cent. Sci.*, 2016, **2**, 725–732.
22 R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams and A. Aspuru-Guzik, *ACS Cent. Sci.*, 2018, **4**, 268–276.
23 L. Messina, A. Quaglino, A. Goryaeva, M.-c. Marinica, C. Domain, N. Castin, G. Bonny and R. Krause, arXiv, 2018.
24 K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko and K.-R. Müller, *J. Chem. Phys.*, 2018, **148**, 241722.
25 J. Behler and M. Parrinello, *Phys. Rev. Lett.*, 2007, **98**, 146401.

26 M. Hellström and J. Behler, *Phys. Chem. Chem. Phys.*, 2017, **19**, 82–96.
27 J. S. Smith, O. Isayev and A. E. Roitberg, *Chem. Sci.*, 2017, **8**, 3192–3203.
28 V. L. Deringer and G. Csányi, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 2017, **95**, 094203.
29 K. Yao and J. Parkhill, *J. Chem. Theory Comput.*, 2016, **12**, 1139–1147.
30 A. Ziletti, D. Kumar, M. Scheffler and L. M. Ghiringhelli, *Nat. Commun.*, 2018, **9**, 1–10.
31 E. J. Kim and R. J. Brunner, *Mon. Not. R. Astron. Soc.*, 2016, **000**, 1–13.
32 A. Aurisano, A. Radovic, D. Rocco, A. Himmel, M. Messier, E. Niner, G. Pawloski, F. Psihas, A. Sousa and P. Vahle, *J. Instrum.*, 2016, **11**, P09001.
33 R. Acciarri, C. Adams, R. An, J. Asaadi, M. Auger, L. Bagby, B. Baller, G. Barr, M. Bass, F. Bay, M. Bishai, A. Blake, T. Bolton, L. Bugel, L. Camilleri, D. Caratelli, B. Carls, R. C. Fernandez, F. Cavanna, H. Chen, E. Church, D. Cianci, G. Collin, J. Conrad, M. Convery, J. Crespo-Anadón, M. Del Tutto, D. Devitt, S. Dytman, B. Eberly, A. Ereditato, L. E. Sanchez, J. Esquivel, B. Fleming, W. Foreman, A. Furmanski, G. Garvey, V. Genty, D. Goeldi, S. Gollapinni, N. Graf, E. Gramellini, H. Greenlee, R. Grosso, R. Guenette, A. Hackenburg, P. Hamilton, O. Hen, J. Hewes, C. Hill, J. Ho, G. Horton-Smith, C. James, J. J. de Vries, C.-M. Jen, L. Jiang, R. Johnson, B. Jones, J. Joshi, H. Jostlein, D. Kaleko, G. Karagiorgi, W. Ketchum, B. Kirby, M. Kirby, T. Kobilarcik, I. Kreslo, A. Laube, Y. Li, A. Lister, B. Littlejohn, S. Lockwitz, D. Lorca, W. Louis, M. Luethi, B. Lundberg, X. Luo, A. Marchionni, C. Mariani, J. Marshall, D. M. Caicedo, V. Meddage, T. Miceli, G. Mills, J. Moon, M. Mooney, C. Moore, J. Mousseau, R. Murrells, D. Naples, P. Nienaber, J. Nowak, O. Palamara, V. Paolone, V. Papavassiliou, S. Pate, Z. Pavlovic, D. Porzio, G. Pulliam, X. Qian, J. Raaf, A. Rafique, L. Rochester, C. R. von Rohr, B. Russell, D. Schmitz, A. Schukraft, W. Seligman, M. Shaevitz, J. Sinclair, E. Snider, M. Soderberg, S. Söldner-Rembold, S. Soleti, P. Spentzouris, J. Spitz, J. St. John, T. Strauss, A. Szelc, N. Tagg, K. Terao, M. Thomson, M. Toups, Y.-T. Tsai, S. Tufanli, T. Usher, R. Van de Water, B. Viren, M. Weber, J. Weston, D. Wickremasinghe, S. Wolbers, T. Wongjirad, K. Woodruff, T. Yang, G. Zeller, J. Zennamo and C. Zhang, *J. Instrum.*, 2017, **12**, P03011.
34 W. Bhimji, S. A. Farrell, T. Kurth, M. Paganini, Prabhat and E. Racah, *J. Phys.: Conf. Ser.*, 2018, **1085**, 042034.
35 A. P. Bartók, R. Kondor and G. Csányi, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 2013, **87**, 184115.
36 Y. Levine, O. Sharir, N. Cohen and A. Shashua, *Phys. Rev. Lett.*, 2019, **122**, 065301.
37 T. Xie and J. C. Grossman, *Phys. Rev. Lett.*, 2018, **120**, 145301.
38 K. Yao, J. E. Herr, D. W. Toth, R. Mckintyre and J. Parkhill, *Chem. Sci.*, 2018, **9**, 2261–2269.
39 F. Brockherde, L. Vogt, L. Li, M. E. Tuckerman, K. Burke and K.-R. Müller, *Nat. Commun.*, 2017, **8**, 872.
40 K. Mills and I. Tamblyn, *Phys. Rev. E*, 2018, **97**, 032119.
41 K. Mills, M. Spanner and I. Tamblyn, *Phys. Rev. A*, 2017, **96**, 042113.
42 K. Ryczko, K. Mills, I. Luchak, C. Homenick and I. Tamblyn, *Comput. Mater. Sci.*, 2018, **149**, 1–18.
43 N. Portman and I. Tamblyn, *J. Comput. Phys.*, 2017, **43**.
44 D. Ciresan, U. Meier and J. J. Schmidhuber, Computer Vision and Pattern Recognition (CVPR) *IEEE Conference on 2012*, 2012, pp. 3642–3649.
45 A. K. Rappe, C. J. Casewit and K. S. Colwell, *J. Am. Chem. Soc.*, 1992, **2**, 10024–10035.
46 K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller and A. Tkatchenko, *Nat. Commun.*, 2017, **8**, 13890.
47 W. Pronobis, K. T. Schütt, A. Tkatchenko and K.-R. Müller, *Eur. Phys. J. B*, 2018, **91**, 178.
48 P. Hohenberg and W. Kohn, *Phys. Rev.*, 1964, **136**, B864–B871.
49 W. Kohn and L. J. Sham, *Phys. Rev.*, 1965, **140**, A1133–A1138.
50 J. Perdew, J. Chevary, S. Vosko, K. Jackson, M. Pederson, D. Singh and C. Fiolhais, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 1992, **46**, 6671–6687.
51 J. P. Perdew, M. Ernzerhof and K. Burke, *J. Chem. Phys.*, 1996, **105**, 9982–9985.
52 C. Lee, W. Yang and R. G. Parr, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 1988, **37**, 785–789.
53 A. D. Becke, *J. Chem. Phys.*, 1993, **98**, 5648–5652.
54 E. Prodan, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 2006, **73**, 085108.
55 C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.
56 K. Alex, I. Sutskever and G. E. Hinton, *Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.
57 D. P. Kingma and J. Ba, arXiv, 2014, **1–15**.
58 G. Kresse and J. Hafner, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 1993, **47**, 558–561.
59 G. Kresse and J. Hafner, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 1994, **49**, 14251–14269.
60 G. Kresse and J. Furthmüller, *Comput. Mater. Sci.*, 1996, **6**, 15–50.
61 G. Kresse, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 1996, **54**, 11169–11186.

3.5 Beyond physics

While they are inspired by physics, the principles of extensive deep neural networks are not limited to applications in physics. We have already discussed how the number of particles is a basic extensive property, and thus counting is a natural application for extensive deep neural networks.

Counting tasks in computer vision are challenging for a number of reasons. If we assume we are counting objects in an image, then we are likely to have some degree of perspective to deal with; objects nearer to the camera will appear larger than objects far from the camera. Furthermore, obfuscation is likely, with objects being in front of other objects fully or partially obscuring the ability of achieving an accurate count. Finally, merely obtaining the correct count label with which to train a model can be challenging; this is often done manually, with humans hand-annotating images to count objects.

With extensive deep neural networks, we can train counting tasks in a weakly-supervised way; the only label information required is the total count of objects. Furthermore, we can easily count multiple classes of objects within the same image by using a vector of counts as the label on which to train.

A very important "bonus" of using an EDNN for this task is the ability to then localize objects within the original image. Prior to the final summation reduction (recall Figure 2 in Section 3.4 for an EDNN schematic) we have access

to the individual count contributions of each tile that comprises the full image. By looking at these count contributions during inference, we can construct a density map of objects at a resolution of the nonoverlapping focus regions. This can be used to localize objects within the image.

In "Weakly-supervised multi-class object localization using only object counts as labels" we present this technique along with several new data sets that can be used for such tasks. We present several modified MNIST data sets as well as two data sets of rubber ducks and bouncy balls floating in various scenes. We demonstrate that an EDNN is able to learn to count and localize both ducks and balls, when only provided a label such as "[5 ducks, 2 balls]". Furthermore, since this approach uses an EDNN, we demonstrate how an EDNN trained on small images is able to learn transferable knowledge to large images, and even transfer knowledge about counting ducks to a different scene entirely.

This is not the first application of a neural network for weakly-supervised counting of objects. Seguì, Pujol, and Vitrià [140] employ a convolutional neural network, but the use of fully-connected layers across the entire visual field limits transferability to other sizes, and localization must be obtained through auxiliary clustering techniques. To our knowledge, this is the first weakly-supervised counting and localization approach that is transferable to systems of a different size, and inherently localizes objects without further clustering techniques.

Author contributions: Isaac Tamblyn revised the manuscript and supervised the project. This work is available as a preprint [5].

Weakly-supervised multi-class object localization using only object counts as labels

Kyle Mills
Faculty of Science
University of Ontario Institute of Technology
Oshawa, Ontario, Canada
Vector Institute for Artificial Intelligence,
Toronto, Ontario, Canada
kyle.mills@uoit.net

Isaac Tamblyn
National Research Council Canada
Ottawa, Ontario, Canada
Vector Institute for Artificial Intelligence,
Toronto, Ontario, Canada
isaac.tamblyn@nrc.ca

Abstract

We demonstrate the use of an extensive deep neural network to localize instances of objects in images. The EDNN is naturally able to accurately perform multi-class counting using only ground truth count values as labels. Without providing any conceptual information, object annotations, or pixel segmentation information, the neural network is able to formulate its own conceptual representation of the items in the image. Using images labelled with only the counts of the objects present, the structure of the extensive deep neural network can be exploited to perform localization of the objects within the visual field. We demonstrate that a trained EDNN can be used to count objects in images much larger than those on which it was trained. In order to demonstrate our technique, we introduce seven new datasets: five progressively harder MNIST digit-counting data sets, and two data sets of 3d-rendered rubber ducks in various situations. On most of these datasets, the EDNN achieves greater than 99% test set accuracy in counting objects.

1 Introduction

The goal of automated object localization research is to take a two-dimensional projection of a scene (e.g. a photograph) and construct a spatial density map, indicating where in the image the objects of interest appear. The integral of this density map can be evaluated to arrive at a count of the number of objects of various types present in the full three-dimensional area [1]. There are numerous applications where accurate counting of objects from a visual camera signal is beneficial, such as population monitoring in the Seregeti[2], counting humans in crowded locations [3, 4, 5, 6, 7, 8], or counting bacteria on microscope slides [9]. In order to train supervised computer vision and machine learning methods for this task, labels are required. In the most strongly-supervised techniques, each pixel is assigned to a class and the task is called segmentation, with the neural network tasked at predicting labels at pixel resolution. Coming in slightly weaker are datasets labelled with bounding boxes around objects and the problem is cast as one of detection. More weakly-supervised techniques that use point annotations have been employed to count cells [10, 11, 12, 13, 14, 9, 15, 16], cars [17, 18, 19], and penguins [20, 16], among other things. Point annotations are more appropriate when object occlusion is present and bounding boxes or pixel-specific labels would overlap significantly [13]. The main difficulty with these approaches lies in obtaining the detailed annotation labels (e.g. bounding boxes or positions). These annotations are traditionally provided by humans, such as the task of clicking on penguins in the arctic [20]. Crowdsourcing this tedious work has become common, but with non-experts and anonymous users providing the labels, a training set could easily become fouled with erroneous labels, making training an accurate model difficult [21]. It is substantially easier to arrive at a count of the number of objects present in the field of view than it is to assign all pixels to class or draw bounding boxes around objects.

1.1 Related work

Patch-based counting and localization approaches are common, with the standard approach being to cast the problem as a density-map regression problem. For example, Refs. [10, 19, 18, 9] blur point annotations to construct a density map. Then they train a regression model to, acting on a single patch at a time, reproduce this density map. Once the model is trained, a density map can be constructed from new images, providing localization information, and the integral of the density map can provide total count information.

Our application aims to make the task even simpler; knowing only the number of objects present in an image and using this as a label, we train an end-to-end deep learning model capable of achieving both accurate multi-class counting and multi-class localization of objects. One of the benefits of using only the raw object counts is that count information is more easily obtained than precise location annotations, and furthermore, the count signal need not originate from the visual signal itself. Measurements from other devices, e.g. weight sensors, optical tripwires, etc. can be used as labels to train a model operating on the visual signal. Unlike other similar approaches [22], this approach relies only on a single scalar label, and ground truth segmentation data is not required for obtaining object localization or counting. Once trained, the model can be applied to arbitrarily large images. Because the neural network only acts on a small subset of the input image, the actual convolutional neural network model can be relatively small, and the systematic way in dividing input images into evenly-sized patches is very easy to implement.

2 Methods

Extensive deep neural networks (EDNN) were proposed by Mills, et. al [23] as a way to capture the extensive nature of physical properties in physics and chemistry applications. An extensive property is one that scales linearly with system size (such as number of objects, or total energy in the case of chemistry applications), with the obverse being an intensive quantity (such as temperature or density). The EDNN technique itself is relatively simple: break the input image $x \in \mathbb{R}^{W \times H \times d}$ (where d is the number of channels, e.g. 1 for grayscale and 3 for RGB) into non-overlapping patches of size $f \times f \times d$, called focus regions. These focus regions are then padded with a border of width c, adding "context" around the non-overlapping patches. Each $(f + 2c) \times (f + 2c) \times d$ "tile" is then fed through the same neural network, outputting a single number for each tile. These results are summed and the final summed value is used as the "prediction" against which to compute the loss and perform back-propagation. The effect of this technique is that the neural network only needs to learn to predict a fractional contribution of the final object count. The neural network learns automatically how to treat the overlapping context regions so as not to double count contributions. Furthermore, since each tile is comprised of non-overlapping focus regions, one can consider the neural network output for a single tile the fractional count of the number of objects present and a density map can be constructed over the original image at a resolution determined by the focus size f. In the case of small focus, this can enable the precise localization of objects in the image.

We have designed multiple data sets through which we demonstrate the EDNN-counting and localization. The MNIST variants are constructed by extracting 4800 examples of each of the ten numerals from the original MNIST dataset. For each of the ten digits, 480 members are reserved for the testing sets and 4320 are used to construct the training sets (i.e. no unique MNIST example is present in both the training and testing sets) The examples are constructed by choosing a random number N_i between 0 and $L_{\max}$ for each numeral i present in the data set. This will serve as the label. N_i digits are then chosen randomly from either the testing or training subset and are composited randomly on an empty 256×256 pixel image through element-wise summation. In the case where occlusion is permitted, pixel values are clipped at 255. The images are saved as greyscale 8-bit PNG images, with labels in a separate JSON file.

The RD-2 dataset is generated using a script inspired by the CLEVR [25] dataset, using Blender to perform the 3d rendering. For each of the label-scene pairs (e.g. "5 ducks, 2 balls, mountain scene"), we generated 1024 images by randomly placing, scaling, and rotating the appropriate number of ducks and balls. 896 images of each scene-label pair are used for training and 128 for testing. The RD-1 dataset is a subset of the RD-2 dataset (the examples with zero balls). The images are generated at a resolution of 256×256 pixels and stored in RGB (3 channel) 8-bit PNG images. Labels are stored in a separate JSON file.

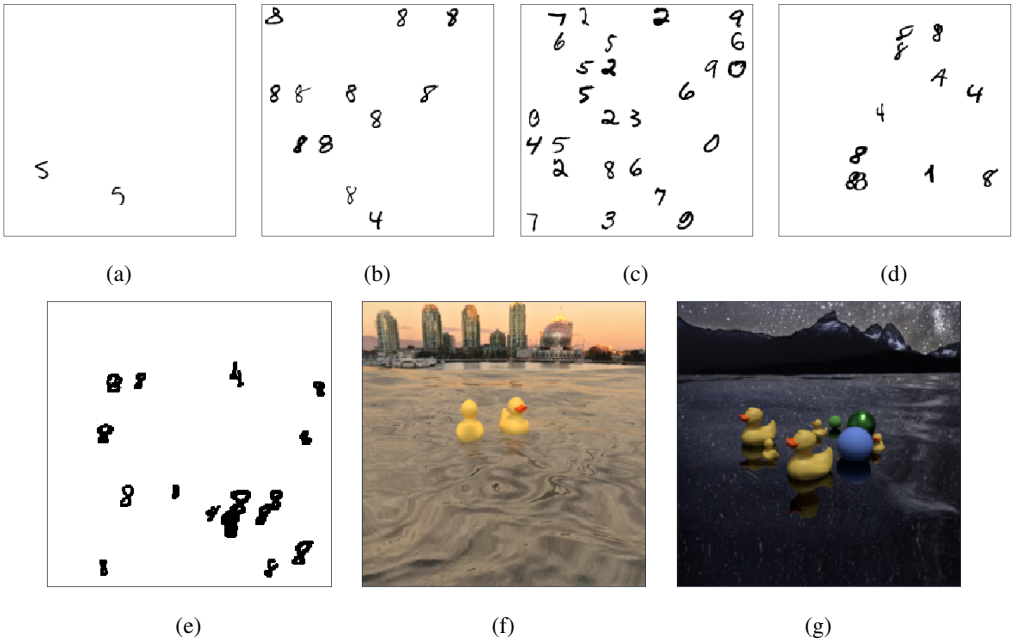


Figure 1: An example image from each of the 7 datasets presented in this paper. The datasets are described in Table 1 and additional examples are included in the Supplementary Information.

Name	Description	$L_{\max}$
(a) MNIST-1	Single category collage of hand-drawn examples of the digit 5 from the original MNIST data set [24]. No digits overlap and all are the same size.	25
(b) MNIST-2	Two category collage of hand-drawn examples of the digit 4 and 8 from the original MNIST data set. No digits overlap and all are the same size.	12
(c) MNIST-10	Ten category collage of hand drawn digits from the original MNIST data set. No digits overlap and all are the same size	6
(d) MNIST-2-occ	Two category collage of hand drawn examples of the digit 4 and 8 from the original MNIST data set. Digits are permitted to overlap (**occ**lude) other digits. All digits are the same size.	15
(e) MNIST-2-occ-vs	Two category collage of hand drawn examples of the digit 4 and 8 from the original MNIST data set. Digits are permitted to overlap (**occ**lude) other digits, and the digits are scaled randomly before placement (using a standard scaling function employing bicubic interpolation).	15
(f) RD-1	Single category 3d renderings of zero to five rubber ducks in different scenes. The ducks are scaled and placed randomly and partial occlusion is permitted (full occlusion is prevented). Each image is 256×256 pixels with three (RGB) channels.	5
(g) RD-2	Two category 3d renderings of zero to five rubber ducks, and zero to five floating spheres ("balls"). Some high-number count combinations are omitted (e.g. 5 ducks and 5 balls) due to the difficulty of packing the objects while preventing occlusion. The spheres are randomly coloured and scaled and randomly assigned either a matte or metallic finish. Each image is 256×256 pixels with three (RGB) channels.	5

Table 1: Summary of the designed datasets. $L_{\max}$ denotes the largest label value (i.e. the maximum count of each object in a given image).

An important consideration in the design of an EDNN was how to choose an appropriate tile size (e.g. focus and context region). Mills et. al. discusses this process based on the length scale of the physics of the underlying problem. In the case of counting variable-sized objects present in an image, the process for choosing a focus and context is less clear. From intuition, we can suggest that the total tile size ($f + 2c$) should be large enough to capture an identifiable feature of the largest objects we wish to count. For example, to count camera-facing waterfowl, one does not need to actually be able to identify a bird; an accurate count could be obtained by only learning to identify beaks. We found that using a focus of $f = 8$ and a context of $c = 8$ worked well for the MNIST counting purposes. For the rubber duck counting, we used a slightly larger context, $c = 12$, as the largest ducks covered more pixels than the MNIST digits. We will discuss the benefits of different focus and context values further in the discussion.

With our choice of focus and context, the neural network must be designed to act on tiles of size $(f+2c)\times(f+2c)$, 24×24 for MNIST and 32×32 for the ducks. The input images are zero-padded with c pixels on all sides, and the tiles are constructed using standard TensorFlow [26] image functions operating on these zero-padded copies of the input data.

We used a standard convolutional neural network with $N = \mathrm{floor}(\log 2(f+2c) - 1)$ layers operating with $K = 64$ square kernels of size $k = 4$. Each layer operated with stride $S = 2$. The output of the final convolutional network is flattened and passed into a dense layer with 1024 outputs, which is then fed into a final dense layer. The final layer has l outputs, where l is the dimensionality of the labels (i.e. the number of classes being counted).

In practice, the computational graph is constructed to take a batch of N_{batch} $L \times L$ images, and deconstruct them into $N_{\mathrm{tiles}} = L^2/f^2$ tiles. The input data is concatenated along the first axis; this results in a tensor of shape $[N_{\mathrm{batch}} \times N_{\mathrm{tiles}}, f + 2c, f + 2c]$. The convolutional neural network operates on this tensor, performing the same operations on all tiles of all images in the batch, reducing it to shape $[N_{\mathrm{batch}} \times N_{\mathrm{tiles}}, l]$. This tensor is then reshaped to $[N_{\mathrm{batch}}, N_{\mathrm{tiles}}, l]$; in practice, this can be thought of "how much stuff" should be attributed to the $f \times f$ focus region. We will refer later to this tensor, so we will label it $\mathcal{C}$. Then for the core of the EDNN technique: a summation reduction over the second axis. This results in a $[N_{\mathrm{batch}}, l]$ tensor denoting the prediction of the l extensive quantities for each image in the batch. The loss is computed as the mean squared error between this vector and the vector of labels, and the Adam optimizer [27] is used to minimize this loss.

Standard neural network training procedures were employed using TensorFlow [26] and the Adam optimizer [27] with a learning rate of 10^{-4}. We trained the EDNN until the loss dropped below 10^{-3}, between 100 and 500 epochs, depending on the difficulty of the dataset.

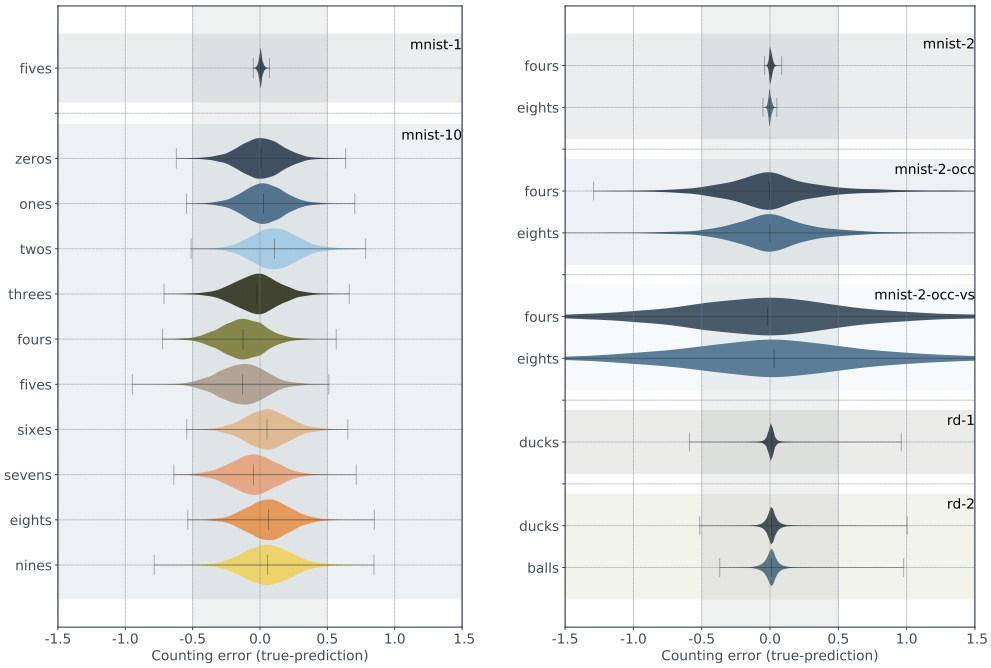
3 Results

3.1 MNIST variants

The simplest case is the MNIST-1 dataset. The EDNN is able to count the number of fives with $100.00\ \%$ accuracy. Localization of the digits within the input image can be achieved by assigning the contents of the tile contribution tensor $\mathcal{C}$ to the focus regions over which its contributions were obtained. Doing so results in the ability to detect not only the number of objects in the visual field, but additionally pinpoint their location with considerable accuracy.

Next is the MNIST-2 dataset, testing the ability of the EDNN to perform mutli-class counting and localization. The EDNN is tasked with counting 4s and 8s, and does so with $100.00\ \%$ accuracy for both categories. Similarly, localization can be achieved. Interestingly, the EDNN learns to assign a positive contribution to the class of interest, while assigning a negative contribution to the other "negative" class. This is because of the EDNN's limited receptive field; since it only sees a small region of the input image, in the tiles surrounding a digit, the EDNN cannot see enough information to identify which class the digit belongs to, and thus assigns a small, positive contribution. Then when the EDNN sees a tile more central to the negative-class digit, it must output a negative value to compensate for its misclassification of the exterior regions. The result is still a very precise localization of each class of objects in the receptive field.

The MNIST-10 dataset includes examples of all ten handwritten digits, from 0 through 9. The EDNN performs exceptionally well; it performed worst at counting fives, but still performed with $98.08\ \%$ accuracy. The performance of the EDNN on MNIST-10 is shown in Fig. 3.



(a) (b)

Figure 2: Error distributions for all classes presented in this paper. The minimum, median, and maximum error for each class is displayed as vertical lines on the distributions. The interval of correct counting (-0.5 to 0.5) is shaded. A rounding of the final count to integer values would result in any examples within this interval being counted correctly. We can see easily from these plots that the MNIST digit counting with occlusion is the most difficult task.

A more difficult challenge is when the digits are permitted to overlap (occlude) other digits. Nonetheless, the EDNN is able to count the digits quite accurately with an accuracy of $84.82\ \%$ and $88.50\ \%$ for counting fours and eights, respectively.

When variable-sized digits are included in the dataset, and digits are allowed to occlude each other, the task is considerably more difficult with only $56.68\ \%$ and $54.72\ \%$ accuracy for counting fours and eights, respectively. This is not surprising; looking at the example image in Fig. 3, it is difficult to differentiate many of the digits even by eye.

The error distributions for all datasets are shown in Fig. 2.

3.2 Rubber ducks

Next we move on to the rubber ducks. The EDNN is able to count variable-sized rubber ducks that are permitted to partially occlude each other, and does so with high accuracy ($99.39\ \%$). The dataset includes five different scenes and a variety of camera angles and lighting conditions affording the EDNN the opportunity to learn to identify objects and not merely base its prediction on the presence of a particular colour in the image.

Next we trained an EDNN from scratch on the RD-2 dataset, including multi-colored balls in addition to the rubber ducks. The EDNN performs exceptionally well, counting the two distinct object classes, "ducks" and "balls", with $99.58\ \%$ and $99.38\ \%$ accuracy, respectively on the test images. An example is shown in Fig. 3.

One might question whether the trained model generalizes to rubber ducks which find themselves in situations not present in the training set. To test this, we construct a 3d-scene of multiple ($l_0 = 16$) ducks in a bathroom scene [28]. The ducks are various sizes, in various orientations and surrounded by other 3d objects. The overall image is rendered at a much higher resolution (1920×1080) than

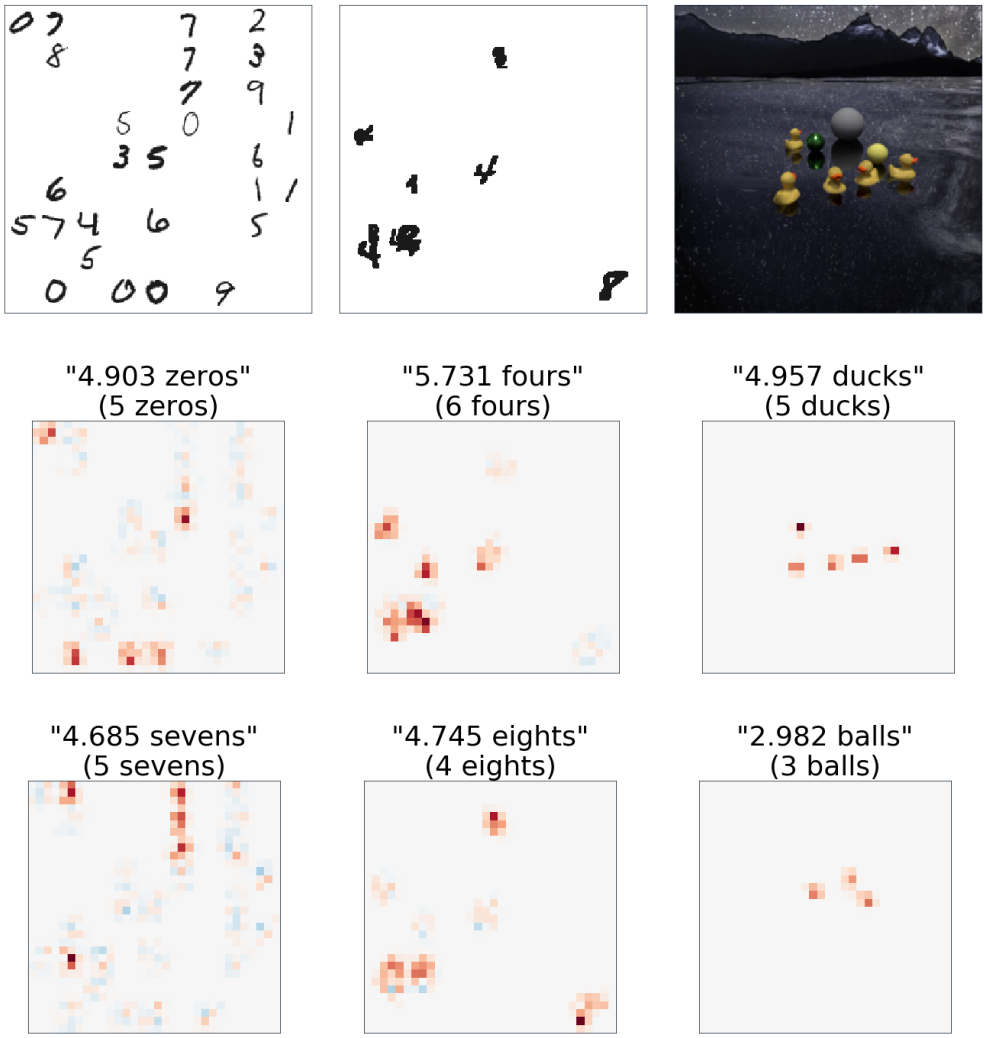


Figure 3: Output of the EDNN for counting and localization of a testing example from three datasets: MNIST-10, MNIST-2-occ-vs, and RD-2. The top row represents the reference image x. The second and third rows show the tile contributions $\mathcal{C}$ for two of the multi-class labels, e.g. l_0, l_N. Red denotes a large positive contribution, whereas blue denotes a negative contribution. Above, the predicted object count is displayed (the integral (sum) of $\mathcal{C}$) alongside the true count in parentheses.

the images present in the training set, however the EDNN technique can easily handle this increase in size; there are simply more tiles to evaluate prior to the final summation. In fact, EDNN can be evaluated on arbitrary input sizes so long as the image resolution is a multiple of the focus region (zero padding can be employed if this is not the case, effectively making this constraint moot). Most of the ducks in the larger image are of comparable scale to those in the training set. Fig. 4a shows the scene. The EDNN trained on the duck-only RD-1 dataset does poorly on evaluating the number of ducks in the bathroom. By looking at $\mathcal{C}$ we can form a hypothesis as to why. Summing the interior of a region of $\mathcal{C}$ will tell us how many ducks the EDNN has decided are present within the boundary of the region. When we do this for some regions-of-interest, we can see that many ducks are over-counted, while others are under-counted. It is not possible to tell exactly what the problem is, however the EDNN is clearly considering the yellow balls to be somewhat duck-like. Our best guess for this failure is that the EDNN is placing too much reliance on the boundary between "yellow" and "non-yellow" as a good indicator of a duck, and is thus miscounting both large ducks and yellow balls. If this is indeed the case, the model trained on RD-2 should perform better as the training examples included yellow balls that the EDNN would have needed to learn are not ducks.

Fig. 4c shows an identical analysis for the model trained on the RD-2 dataset evaluated on the bathroom scene. The model misses three ducks, but this time the reason is clear: the erroneous counts are ducks that differ from the ducks in the training set. One duck is significantly smaller than any ducks present in the training set, and the other two have fallen over, an orientation absent from the training set.

A remedy for such an issue should be clear; train the EDNN additionally on images of sideways rubber ducks. This can be accomplished most simply by applying a random rotation by an integer multiple of $\pi/2$ to the input image pipeline during training, augmenting the data set. After doing this, the EDNN does indeed count the sideways rubber ducks and additionally identifies the tiny duck, although at a reduced count since it is smaller. Variations in object sizes can also be handled through data augmentation, scaling down the input images and zero-padding the boundary.

4 Conclusion

We demonstrate the use of an extensive deep neural network for providing multi-class object localization in images, using weakly-supervised learning. By training an EDNN to count objects in images, we arrive at a model which can both accurately count instances of objects in images as well as spatially localize them using only object counts as labels. We demonstrate that multiple classes of objects can be counted simultaneously (multi-class counting). The EDNN is, through training, able to develop an internal representation of the objects suitable for counting without any bounding boxes, point annotations, or ground truth segmentation data. The spatial structure of the EDNN can be exploited to additionally provide a density map of the objects in the image, providing accurate localization within the field of view (multi-class localization). Once trained, we demonstrate the EDNN can be used on images significantly larger and different than those present in the training dataset. The EDNN technique is a simple and useful technique for computer vision applications.

References

[1] Tobias Stahl, Silvia L Pintea, and Jan C. van Gemert. Divide and Count: Generic Object Counting by Image Divisions. *IEEE Transactions on Image Processing*, 28(2):1035–1044, feb 2019.

[2] Alexandra Swanson, Margaret Kosmala, Chris Lintott, Robert Simpson, Arfon Smith, and Craig Packer. Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna. *Scientific Data*, 2:150026, jun 2015.

[3] Kang Han, Wanggen Wan, Haiyan Yao, and Li Hou. Image Crowd Counting Using Convolutional Neural Network and Markov Random Field. pages 1–6, jun 2017.

[4] Yaocong Hu, Huan Chang, Fudong Nian, Yan Wang, and Teng Li. Dense crowd counting from still images with convolutional neural networks. *Journal of Visual Communication and Image Representation*, 38:530–539, 2016.

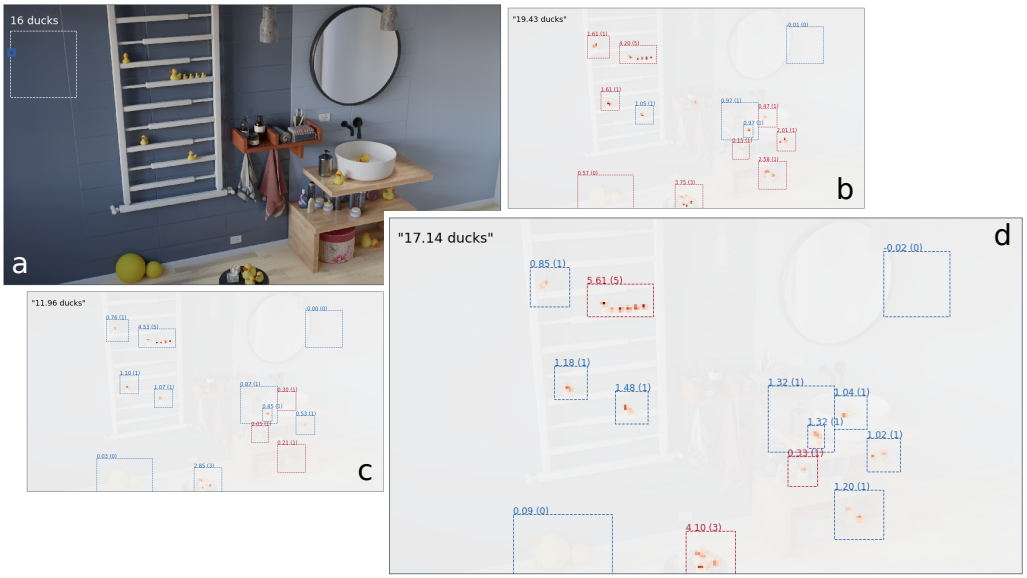


Figure 4: We used the models trained on the RD-1 and RD-2 datasets to count the number of ducks in a completely new scene ("bathroom"), at a resolution of 1920×1080. We show the scene in (a), which contains 16 rubber ducks of various sizes. The dashed-line inset in (a) denotes the size of the training dataset images, 256×256 pixels, and a tile is displayed, showing both the focus (red) and context (blue) regions to scale. In (b) and (c) we overlay the contents of the $\mathcal{C}$ tensor, with some regions highlighted with dashed rectangles. The sums of the interior regions bounded by the rectangles are printed above, as well as the true number of ducks contained within each region. Regions containing a correct count (post-rounding) are shown in blue, whereas regions with an incorrect count are shown in red. In (b), the RD-1 model performs quite poorly, misidentifying the yellow balls as parts of a duck, and miscounts many regions. In (c), the RD-2 model performs well, counting most groups of ducks quite accurately. It correctly ignores the yellow balls, as during training it has seen yellow balls. Three ducks are miscounted; however, these ducks are either in an orientation or of a size not present in the training set, and thus misidentification is expected. In (d), we fix this by further training the model on an augmented pipeline with rotations of the original images included. This results in better overall results, including the fallen ducks and identifying the tiny duck.

[5] Lingke Zeng, Xiangmin Xu, Bolun Cai, Suo Qiu, and Tong Zhang. Multi-scale convolutional neural networks for crowd counting. *Proceedings - International Conference on Image Processing, ICIP*, 2017-Septe:465–469, 2018.

[6] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-Image Crowd Counting via Multi-Column Convolutional Neural Network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 589–597. IEEE, jun 2016.

[7] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. Cross-scene crowd counting via deep convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 833–841. IEEE, jun 2015.

[8] Antoni B. Chan, Zhang-Sheng John Liang, and Nuno Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7. IEEE, jun 2008.

[9] Joseph Paul Cohen, Genevieve Boucher, Craig A. Glastonbury, Henry Z. Lo, and Yoshua Bengio. Count-ception: Counting by Fully Convolutional Redundant Counting. mar 2017.

[10] Luca Fiaschi ; Ullrich Koethe ; Rahul Nair ; Fred A. Hamprecht. Learning to Count with Regression Forest and Structured Labels. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. [IEEE], 2012.

[11] Antoni B. Chan, Zhang-Sheng John Liang, and Nuno Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7. IEEE, jun 2008.

[12] Elad Walach and Lior Wolf. Learning to Count with CNN Boosting. pages 660–676. Springer, Cham, 2016.

[13] Weidi Xie, J. Alison Noble, and Andrew Zisserman. Microscopy cell counting and detection with fully convolutional regression networks. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 6(3):283–292, may 2018.

[14] Yao Xue, Nilanjan Ray, Judith Hugh, and Gilbert Bigras. Cell Counting by Regression Using Convolutional Neural Network. pages 274–290. Springer, Cham, 2016.

[15] Alexander Gomez Villa, Augusto Salazar, and Igor Stefanini. Counting Cells in Time-Lapse Microscopy using Deep Neural Networks. jan 2018.

[16] Mark Marsden, Kevin Mcguinness, Suzanne Little, Ciara E Keogh, and Noel E O'connor. People, Penguins and Petri Dishes: Adapting Object Counting Models To New Visual Domains And Object Types Without Forgetting. Technical report.

[17] Jiyong Chung and Keemin Sohn. Image-Based Learning to Measure Traffic Density Using a Deep Convolutional Neural Network. *IEEE Transactions on Intelligent Transportation Systems*, 19(5):1670–1675, may 2018.

[18] Shiv Surya and Venkatesh Babu R. TraCount: a deep convolutional neural network for highly overlapping vehicle counting. In *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing - ICVGIP '16*, pages 1–6, New York, New York, USA, 2016. ACM Press.

[19] Daniel Oñoro-Rubio and Roberto J López-Sastre. Towards Perspective-Free Object Counting with Deep Learning. pages 615–629. 2016.

[20] Carlos Arteta, Victor Lempitsky, and Andrew Zisserman. Counting in the Wild. pages 483–498. Springer, Cham, 2016.

[21] Vishwanath A. Sindagi and Vishal M. Patel. A survey of recent advances in CNN-based single image crowd counting and density estimation. *Pattern Recognition Letters*, 107:3–16, 2018.

[22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. Technical report.

[23] Kyle Mills, Kevin Ryczko, Iryna Luchak, Adam Domurad, Chris Beeler, and Isaac Tamblyn. Extensive deep neural networks for transferring small scale learning to large scale systems. *Chemical Science*, 10(15):4129–4140, aug 2019.

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[25] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. dec 2016.

[26] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *None*, 1(212):19, mar 2016.

[27] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. pages 1–15, dec 2014.

[28] Davide Tirindelli. Blue bathroom, 2018.

Chapter 4

Adversarial learning

Adversarial learning is a technique where multiple (usually two) neural networks are trained in parallel. It gets its name from the fact that the literature usually presents the two networks as being in a competitive relationship, working against each other to "fool" the other network. Adversarial networks are usually presented as a form of generative model, as typically one of the networks is formulated in such a way that it learns to produce realistic-looking example data.

4.1 GAN and the Ising model

In our first investigation of adversarial learning to physics, we show that a Generative Adversarial Network can be used to learn to sample different distributions, using the Ising model as a proof-of-concept.

Furthermore, we demonstrate that an interesting consequence of the adversarial learning method (that was often overlooked at the time), is that the discriminator can be forced to predict a label (e.g. energy), in a very similar

way to traditional supervised learning. Once trained, the discriminator can be used as both a predictor of the label (as in any traditional regression task), but it has also learned a valuable representation of "truth", that can be exploited to perform anomaly detection.

We demonstrate these features and more in "Phase space sampling and operator confidence with generative adversarial networks" [6].

Author contributions: Isaac Tamblyn supervised the project, contributed text to the introduction and was involved in discussions throughout.

Phase space sampling and operator confidence with generative adversarial networks

Kyle Mills*
Department of Physics, University of Ontario Institute of Technology

Isaac Tamblyn†
Department of Physics, University of Ontario Institute of Technology, University of Ottawa & National Research Council of Canada

(Dated: October 24, 2017)

We demonstrate that a generative adversarial network can be trained to produce Ising model configurations in distinct regions of phase space. In training a generative adversarial network, the discriminator neural network becomes very good a discerning examples from the training set and examples from the testing set. We demonstrate that this ability can be used as an "anomaly detector", producing estimations of operator values along with a confidence in the prediction.

INTRODUCTION

As machine learned potentials come to be used as replacements for conventional force-fields (such as CHARMM, AMBER, etc.) in large-scale and long-time atomistic simulations, it will be important to ensure that these new "model-free" methods provide a level of confidence alongside each of their property predictions.

In the field of self-driving vehicles, it has long been recognized that it is more important to develop algorithms which are constantly aware of their accuracy than models which naively return a prediction irrespective of input. Should a vehicle enter an unfamiliar situation (i.e. one that does not exist within the training set), it is important that it identify the drop in predictive confidence so that the human driver can intervene, or the car can safely come to a stop. Naively continuing to make predictions when an algorithm is making large errors is rarely a good plan, and in some cases would be catastrophic.

A similar problem can occur in a numerical simulation based on a configuration-to-energy or configuration-to-force model. When a user is aware that a model is having difficulty making predictions (perhaps because the system has evolved into a fundamentally new region of configuration space, or is violating a conservation law), it is possible to take action. Such actions could include reducing the integration time step or collecting more training data and producing a new model which includes configurations from the new regime.

Without a confidence metric, there is a risk that a model will become unreliable and begin to return unphysical predictions on new data. For the case of molecular dynamics or a Monte Carlo simulation, these erroneous predictions can result in the system venturing further away from the reference set, exacerbating the problem. The possibility of unphysical predictions outside of the training regime is not a new one in the field of numerical simulation based on fitting. The issue of "transferability" is often discussed in the context of force-fields and pseudo-potential construction.

Model-free machine-learned potentials, however, may return predictions which have errors substantially larger in magnitude than those of a conventional force-field. In a traditional force-field expansion, the internal energy and forces acting on system are generally expressed as a sum over bonded and non-bonded terms. The form of these terms are typically set by simple cases which can either be solved analytically, or where the limits are well known. The CHARMM force-field, for example, contains terms relating to bond angles, dihedral angles, and torsions [1].

The fact that force-fields are usually parameterized in terms of simple physical features such as distance, angle, etc, also make it easier to detect when the simulation is approaching an unfamiliar region of configuration space (bond lengths shrink beyond a threshold, for example).

Any form of numerical fitting procedure will be most reliable when predictions are made within the space of training data (i.e. interpolation). Extrapolation, making predictions for properties which are outside of the manifold where data was collected, will invariably result in higher errors. Force-fields which are based on a simple physical expansion tend to be well behaved when extrapolating, as they implicitly contain information about limiting cases through the choice of expansion. The typical Lennard-Jones (6-12 potential) form used for non-bonded interactions, for example, naturally goes to zero at large distances, and diverges when particles become too close.

While such simple functions have the feature that they require very few training examples to make generally acceptable predictions, the cost of this favourable extrapolation behaviour is that these models typically are much less sophisticated than the underlying physics they are designed to describe, and have an upper bound to their accuracy even within the interpolation region.

Model-free machine learned potentials require substantially more data, and therefore the collection of training data becomes important. The usual concerns with regard to sampling come into play.

In most cases, Nature is able to sample a thermal distribution of states efficiently, although there are notable

exceptions such as the glass transition. *In silico*, sampling the distribution of possible configurations is a very difficult task due to the enormous number of free parameters (e.g. position, spin, charge, etc.) defining a near-infinite configuration space for systems of even a modest number of particles. This "curse of dimensionality", for all but the most trivial systems, precludes directly sampling configuration space at non-zero, finite temperature [2]. Traditionally, Markov Chain Monte Carlo sampling methods have been devised to obtain random samples from an underlying distribution, but these algorithms, such as Metropolis-Hastings [3, 4], depend on the ability to efficiently evaluate both the energy and property of a microstate (E_i) and (O_i), which can in many cases be a very costly computation. Furthermore, this calculation must be carried out repeatedly, many more times than the desired number of final configurations.

Generative adversarial networks

Here we propose the use of a generative adversarial network (GAN) [5] to carry out both the sampling of configuration space and as a means of providing a confidence estimate for predicted properties. Generative models are common approaches to unsupervised machine learning [6]. Generative adversarial networks are typically applied to problems in image processing, such as to image super-resolution [7], image-to-image translation [8], and cross-domain pairings (e.g. pairing shoes with matching handbags) [9]. They have recently been applied to solutions of differential equations involving transport phenomena [10].

Generative models are based on the core premise that in order to synthesize example data, an understanding of the relevant features must be somehow present in the model, and the training procedure attempts to learn these relevant features, usually in the form of optimizing a set of coefficients in a latent variable space. A generative adversarial network is a relatively new unsupervised machine learning technique; it is the combination of two "players", working against each other as adversaries. One player acts as a generator, taking random noise as input and producing examples that fall within a probability distribution. The other player works as a discriminator and learns to tell the difference between examples coming from the generator and true, ground truth examples. These players are trained simultaneously with the generator trying to trick the discriminator, and the discriminator learning how to better tell apart the generator's propositions from the true training examples. A successfully trained generative adversarial network converges to a state where the generator is so good at producing examples that the discriminator cannot tell the generated examples from the ground truth examples. The key to the success of this method is that the generator is provided with hints about its failure in the form of the gradients, i.e. both the generator and discriminator perform backpropagation using the gradients from the discriminator network. This allows the generator to learn not only whether or not it succeeded in tricking the discrimnator, but effectively gives it access to the reasoning behind its success or failure.

The discriminator can also be provided with relevant labels for the true examples. This acts to help condition the generative adversarial network [11], stabilizing the notoriously sensitive training process [12], but also enabling the trained discriminator to make observable predictions about new data. Some refer to a network trained with such provisions as a Conditional Generative Adversarial Network (cGAN) [10]. In the case where the discriminator is asked to also reproduce the labels (i.e. the cost function includes an error associated with the labels), the discriminator's output can be interpreted as a likelihood that its label prediction is correct, since the discriminator gets exceptionally good at identifying real examples from the distribution. In this way, the discriminator of the trained generative adversarial network can be used as an anomaly detector, providing label predictions alongside a probability that the prediction is correct.

In theory, the generator and discriminator could be any learning algorithm capable of backpropagation. In this work, we use deep convolutional neural networks as their success on the Ising model has been demonstrated in supervised machine learning classification and prediction [13, 14].

The Ising Model

The square two-dimensional Ising model is a well-studied example of a ferromagnetic system of particles [15]. The model consists of an $L \times L$ grid of discrete interacting "particles" which either possess a spin up ($\sigma = 1$) or spin down ($\sigma = -1$) moment. The internal energy associated with a given configuration of spins is given by the Hamiltonian $\hat{H} = -J \sum \sigma_i \sigma_j$ where the sum is computed over all nearest-neighbour pairs ($\langle i, j \rangle$), and J is the interaction strength. For $J = 1$, the system behaves ferromagnetically; there is an energetic cost of having opposing neighbouring spins, and neighbouring aligned spins are energetically favourable. A measure of "disorder" in the system can be represented by an order parameter known as the "magnetization" M, which is merely the average of all L^2 individual spins. Because both the internal energy and magnetization depend on the discrete spins within the system, both quantities exhibit a discrete distribution. The configuration space of the 8×8 Ising model is of size 2^{8^2}, and thus sampling from all possible configurations is impossible. Given a configuration, while computing properties of interest (e.g. energy) is

generally possible through some theoretical framework, the reverse is not true; it is not possible to obtain a configuration that satisfies a given property. In order to generate examples of a specific energy, one must employ some form of Monte Carlo approach, or devise another clever sampling algorithm [16]. Because of its simplicity and ubiquity, the Ising model has made many recent appearances in machine-learning investigations [2, 13, 17–19]

METHODS

Training datasets

We used a targeted sampling procedure [13] resembling the Metropolis-Hastings algorithm to produce four datasets used for training:

- A low-energy dataset, generated by using a target energy of $-1.3L^2$,
- a high-energy dataset, generated by using a target energy of $+1.3L^2$,
- a bimodal dataset consisting of a equal combination of the previous two datasets, and
- a dataset consisting of an equal number of configurations from each of the 63 possible energy values.

The energy distributions of the first three datasets are displayed as the dashed lines in Fig. 3.

Network

The generator takes an array of random noise z as input and produces examples attempting to mimic the true distribution. Our generator function takes a $2 \times 2 \times 128$ random array sampled from a zero-centered normal distribution with standard deviation of 0.7. It passes the random data through 7 transposed convolution layers with varying kernel, stride, and filter counts arriving at the appropriately shaped 8×8 output, $G(z)$. After every transposed convolutional layer (with the exception of the last two), we include a dropout layer with a 0.7 retention probability, to hopefully prevent the generator from memorizing the training set. Since individual spins of the Ising model can either be 1 or -1, all transposed convolution layers have hyperbolic tangent (tanh) activation, which produces output at an appropriate scale.

The discriminator takes an 8×8 array as input. This can either be $G(z)$ from the generator, or an example x from the training set. Through a series of 9 convolutional layers, and two fully connected layers it reduces the example to a single output D. In the case where we provide labels to the discriminator, the output is of size $1 + N$,

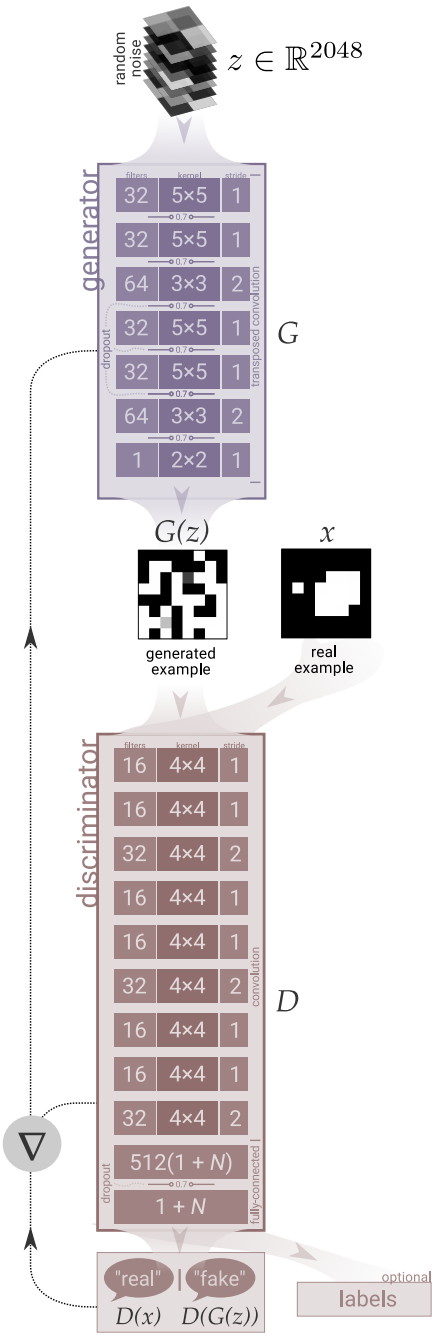


FIG. 1. The generative adversarial network architecture we used. The generator takes random noise, and through a series of transposed convolutions, produces an example. The discriminator takes true examples and "fake" examples from the generator and predicts the probability that each is a real example. The gradients from the discriminator are used by both adversaries to improve their weights.

where N is the number of conditioning labels provided to the discriminator. All layers except for the final layer use rectified linear unit (ReLU) activation. We use a dropout layer with a 0.7 retention probability between the final two fully-connected layers.

We implemented the model in TensorFlow [20]. If trained naively, we discovered that the discriminator learned much more quickly than then generator. When this happened, the discriminator did not provide sufficient feedback (gradients) to the generator and therefore the generator was unable to improve. To remedy this problem, we used a learning rate five times greater for the generator, essentially handicapping the discriminator, and leading to better convergence of both opponents.

The objective of the generator is to minimize the loss function

$$L^{(\mathrm{G})} = -\log(D(G(z)), \tag{1}$$

and the objective of the discriminator is to minimize the loss function

$$L^{(\mathrm{D})}_{\mathrm{GAN}} = \log(1 - D(G(z))) + \log(D(x)), \tag{2}$$

while simultaneously minimizing the mean-squared error of the labels y:

$$L^{(\mathrm{D})}_{\mathrm{LABEL}} = \frac{1}{N_{\mathrm{E}}} \sum_{i=0}^{N_{\mathrm{E}}} \sum_{j=0}^{N_{\mathrm{L}}} \alpha_j \left(y_j^{(\mathrm{predicted})} - y_j^{(\mathrm{true})} \right)^2 \tag{3}$$

where N_{E} is the number of training examples, N_{L} is the number of conditioning labels (in our case $N_{\mathrm{L}} = 2$: energy and magnetization), and α_j is an optional label scaling parameter to account for the relative importance and/or scale when using multiple conditioning labels (for example, the range of Ising energies is always twice that of the Ising magnetizations, so correcly predicting energies would be considered "more important" by the discriminator unless scaled appropriately). The discriminator loss is simply a weighted sum of these two individual losses:

$$L_D = \beta L^{(\mathrm{D})}_{\mathrm{GAN}} + \gamma L^{(\mathrm{D})}_{\mathrm{LABEL}}. \tag{4}$$

We found we obtained the best performance when we initially set $\beta = \gamma = 0.5$. After training for 1000 epochs, we reduced γ to 0.02 and left β unchanged. This permitted the discriminator, which had at this point learned to accurately predict the labels, to "focus" on its ability to differentiate between real and fake outputs, improving anomaly detection.

Contrary to supervised learning, monitoring the loss functions is not an informative method to verify convergence of the model. Both players are simultaneously trying to reduce their own loss functions; a decrease in one leads to an increase in the other. Therefore, during training, we periodically plot the energy and magnetization distributions produced by the generator to verify they are approaching the training distributions.

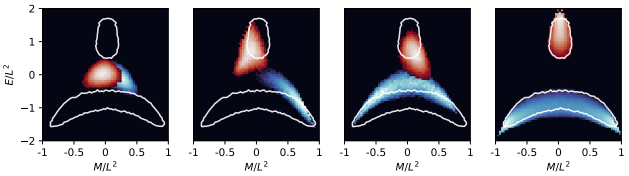


FIG. 2. An example progression of the generator through phase space during the training process for the high energy and low energy target distributions. The generated distributions are represented by the heatmap, and the white boundary represents the target (i.e. training) distribution.

Training proceeds by passing a batch of 512 images to the discriminator. 256 images are from the training set, and 256 are the output from the generator. Initially, the generator has not learned how to produce realistic looking examples, and outputs nothing more than random noise. We use the Adam optimization method [21] with a learning rate of 0.00002 to update the discriminator's weights. Then it is the generator's turn; we feed a batch of random arrays to the generator. The generator uses its learned filters to process the random information into realistic-looking Ising configurations. Of course, the generator performs very poorly during the first few iterations, but the generator's Adam optimizer receives feedback from the discriminator and is able to improve on its kernels to produce better examples. This process is repeated thousands of times, until the energy and magnetization distributions (Figs. 2 and 3) are deemed sufficiently similar. We used a total of $N_{\mathrm{E}} = 50,000$ training images for each generative adversarial network we trained.

RESULTS

We trained three generative adversarial networks: one on the low-energy distribution, one on the high-energy distribution, and one on the bimodal mixture of high- and low-energy distributions. Shortly after training begins, the generator produces examples composed of random spins, and therefore the distributions of energy and magnetization are mostly centered at the zero-point (Fig. 2). As the generator and discriminator learn from each other, the generator begins to better match the training distributions. In the final frame of Fig. 2, the high-energy distribution (red) matches almost exactly with the high-energy training distribution (white outline), and the low-energy distribution (blue) matches closely with its corresponding training distribution. The third row in Fig. 3 shows the performance of the bimodal dataset. The trained generative adversarial network shown here

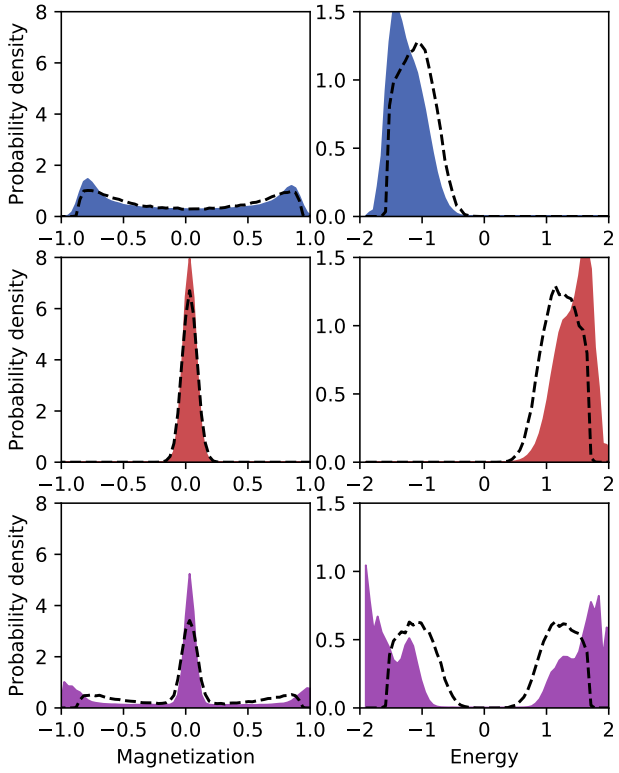


FIG. 3. The energy and magnetization distributions for the training data (dashed lines) and generator output (solid fill). We do not specifically request that the generative adversarial network reproduce these distributions; it must do so automatically. While not matching the exact shape, the generative adversarial network is able to produce examples from the same energy and magnetization ranges, permitting efficient sampling.

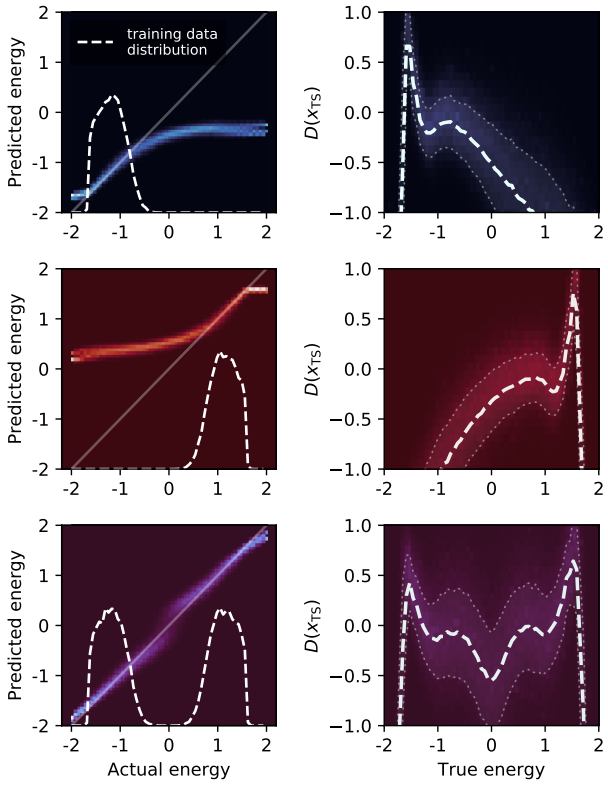


FIG. 4. In addition to its task of opposing the generator, the discriminator can learn to make predictions about operator values if provided with the corresponding labels. Left column: histograms of the predicted vs. true energy for examples from the training set. Right column: when trained using a label, the discriminator's output can be interpreted as a measure of confidence in its prediction. When provided with a uniform distribution of energies across the entire energy range, the discriminator has a much higher average confidence in the region it has not yet seen.

was one of the few training runs that, with moderate success, captured both modes of the distribution. In many training runs, the generator collapsed to either the high-energy or low-energy mode. This "mode-collapse" in generative adversarial networks is a common, and understood phenomena with ongoing research investigating possible solutions [22–25].

In training the generative adversarial network, the discriminator by design becomes very good at identifying configurations which fall outside of the desired distribution, with its output falling roughly on a scale between -1 (the example is suspected to be "fake", i.e. not from the training distribution) and 1 (the example is deemed "real", i.e. likely to be from the training distribution). Provided the discriminator is also able to produce an estimate of the value of an operator (e.g. magnetization and/or energy labels were provided to the discriminator during training), then the output of the discriminator can be interpreted as a confidence of its operator prediction. Thus the generative adversarial network process produces an anomaly detector, providing both a continuous, real-valued operator evaluation (regression) as well as an indication that the predicted value is correct.

After training the discriminator, we presented it with examples drawn from an even distribution across the energy range. In such an application, one would expect that the discriminator makes predictions both *accurately and confidently* near the region on which it was trained. Arguably more importantly is the ability of the discriminator to output a low confidence when it is unsure of the answer, such as in the regions where training data was not provided. Fig. 4 shows that this is indeed the case; the discriminator outputs the highest confidence in regions where it was provided training examples. In regions devoid of training examples, the discriminator indicates its incompetence in making predictions by outputting a low confidence, precisely what one would desire of a practical prediction engine.

CONCLUSION

We have trained a generative adversarial network to be able to efficiently sample phase space, producing examples from a target distribution without the necessity of *a priori* knowledge of the distribution. A generative adversarial network uses to separate convolutional neural networks, the generator and the discriminator. The generator is tasked with producing examples so realistic that the discriminator cannot tell them apart, and the two networks are trained in tandem. Ultimately, the generator becomes so good at producing realistic examples that the discriminator cannot differentiate between the training examples and the output of the generator. If one provides labelled data to the discriminator, we show that the discriminator can be trained to make accurate predictions as well as indicate its confidence in the predictions, essentially anomaly detection through supervised learning.

The generator of a trained generative adversarial network can be used to produce examples which are larger than those on which it was trained [26]. This has been used to produce textures which have arbitrarily large spatial extent, but are based on a spatially-finite training set. Successful application of this technique to physical systems would be incredibly valuable. As the spatial extent of a physical system increases, the individual features comprising the system do not increase in size, but rather in number (extensivity). The simulation cost for such a system, however, increases dramatically. Therefore, a generative adversarial network trained on a small system which is capable of producing examples from a spatially larger distribution would be incredibly useful, and we propose this as a future application of generative adversarial networks.

Generative adversarial networks are notoriously difficult to train, and the training process can be quite unstable, however ongoing research aimed at stabilizing the training process and making generative adversarial networks more robust will lead to generative adversarial networks as a promising tool for use in the physical sciences.

ACKNOWLEDGEMENTS

The authors acknowledge funding from NSERC and SOSCIP. Compute resources were provided by the National Research Council of Canada, SOSCIP, and Compute Canada.

* kyle.mills@uoit.net
† isaac.tamblyn@nrc.ca

[1] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, Journal of Computational Chemistry **4**, 187 (1983).
[2] J. Carrasquilla and R. G. Melko, Nature Physics **13**, 431 (2017), arXiv:1605.01735.
[3] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, The Journal of Chemical Physics **21**, 1087 (1953).
[4] B. Y. W. K. Hastings, , 97 (1970).
[5] I. J. Goodfellow, J. Pouget-abadie, M. Mirza, B. Xu, and D. Warde-farley, , 1arXiv:/arxiv.org/abs/1406.2661v1 [https:].
[6] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, (2016).
[7] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, (2016), 10.1109/CVPR.2017.19, arXiv:1609.04802.
[8] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, (2017), arXiv:1703.10593.
[9] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, (2017), arXiv:1703.05192.
[10] A. B. Farimani, J. Gomes, and V. S. Pande, **94305** (2017), arXiv:1709.02432.
[11] M. Mirza and S. Osindero, , 1 (2014), arXiv:1411.1784.
[12] T. Salimans, I. Goodfellow, V. Cheung, A. Radford, and X. Chen, , 1 (2016).
[13] K. Mills and I. Tamblyn, (2017), arXiv:1706.09779.
[14] A. Morningstar and R. G. Melko, (2017), arXiv:1708.04622.
[15] L. Onsager, Physical Review **65**, 117 (1944).
[16] N. Portman and I. Tamblyn, Journal of Computational Physics , 43 (2017), arXiv:1611.05891.
[17] I. Luchak, K. Mills, K. Ryczko, A. Domurad, and I. Tamblyn, arXiv (2017), arXiv:arXiv:1708.06686.
[18] L. Wang, Physical Review B **94**, 195105 (2016), arXiv:1606.00318.
[19] A. Tanaka and A. Tomiya, , 1 (2016), arXiv:1609.09087.
[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, None **1**, 19 (2016), arXiv:1603.04467.
[21] D. P. Kingma and J. Ba, , 1 (2014), arXiv:1412.6980.
[22] A. Srivastava, L. Valkov, C. Russell, M. Gutmann, and C. Sutton, (2017), arXiv:1705.07761.
[23] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, , 1 (2016), arXiv:1611.02163.
[24] I. Tolstikhin, S. Gelly, O. Bousquet, C.-J. Simon-Gabriel, and B. Schölkopf, , 1 (2017), arXiv:1701.02386.
[25] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, , 1 (2016), arXiv:1606.03498.
[26] U. Bergmann, N. Jetchev, and R. Vollgraf, (2017), arXiv:1705.06566.

4.2 Introducing RUGAN: an improvement on the GAN

In our previous work with generative adversarial networks, we identified that GANs could be used to perform sampling, generating configurations that come from the same distribution as those in the training set. We also mentioned, however, that GANs are notoriously difficult to train. A common pitfall with GANs at the time was mode collapse, where the generator begins producing examples from only one mode of a multimodal distribution. Additionally, a general difficulty in training GANs lead to instability in attaining an equilibrium between the generator and the discriminator. The generator or discriminator often begins to outperform the other, and its opponent is unable to restore the the equilibrium, leading to complete failure.

These problems suggested that, while GAN technology was promising, awaiting the maturation of the field would be the best way forward with GAN applications to physics.

Two years later, we began to revisit the application of GANs to physics with the introduction of the Regressive Upscaling Generative Adversarial Network (RUGAN) [7]. RUGAN employed some of the new developments in GAN techniques, mainly changing the loss function to use the Wasserstein distance met-

ric [121], stabilizing training significantly. The motivation for the move to the Wasserstein distance is presented in Section 2.3.2.

In addition to this modification, we made sure to use only convolutional layers in the generator, removing the fully-connected output layer entirely. Since convolutional layers are translationally invariant, this simple modification of using solely this type of layer permits the evaluation of the generator on a latent space of arbitrary spatial scale. As such, after the generator is trained (i.e. during inference), feeding in a larger latent noise block produces an output that is correspondingly larger in size, and still contains the unique motifs that were learned from the small scale training set.

One further modification that is important for many physical systems is the use of periodic (i.e. cyclical, circular) padding for all convolutional layers. This means that the generator output consequently is seamlessly tiled with periodic boundary conditions.

An important feature of RUGAN is that it can be conditioned on a label, causing the generator to output examples that both resemble those from a training set, and are additionally of a desired label value. This both stabilizes training and results in a generator that can be queried for an output *of a specific desired conditioning value*. This makes RUGAN a powerful sampling tool.

These improvements to the original GAN work were made in the context of

physical experiments in “Optical lattice experiments at unobserved conditions and scales through generative adversarial deep learning” [7]. This manuscript is included as Section A, as its methodology is important for this dissertation.

Author contributions: Corneel Casert developed the code framework and carried out preliminary tests (that would be used for Ref. [7] (Section A), which Kyle Mills then adapted to work with porous graphene sheets, devised the encoding, and modified the framework to work with hexagonal convolutions. Isaac Tamblyn was involved in discussion and the revision of the manuscript.

4.3 RUGAN and hexagonal sheets

After the RUGAN methodology was established, we further applied RUGAN to generate mesoscale porous graphene surfaces in “Adversarial Generation of Mesoscale Surfaces from Small-Scale Chemical MotifsâĂİ. We train RUGAN on porous graphene sheets, the same sheets that were previously presented in “Extensive Deep Neural Networks”. After training RUGAN, and conditioning on the total energy of the training set microstates, RUGAN is able to generate additional, unseen microstates (configurations of porous graphene sheets) that are both at a requested energy value and periodically wrap seamlessly for use in common computational chemistry techniques. Additionally, and more importantly, RUGAN can transfer what it learned and generate very large porous

graphene sheets, much larger than those on which it was trained. RUGAN is a promising approach that captures the insights present in a small scale data set and can extend these insights to produce large scale structures that would otherwise be infeasible to produce using traditional computational techniques.

THE JOURNAL OF PHYSICAL CHEMISTRY C

pubs.acs.org/JPCC Article

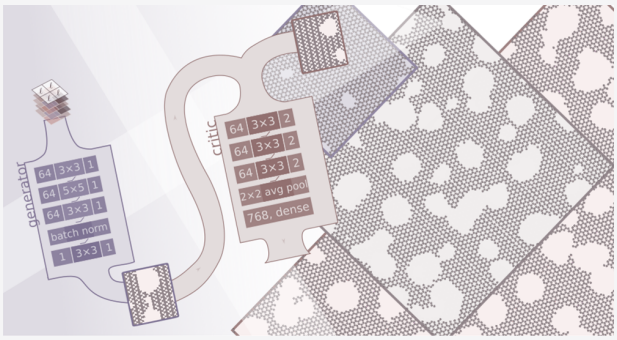
Adversarial Generation of Mesoscale Surfaces from Small-Scale Chemical Motifs

Published as part of The Journal of Physical Chemistry virtual special issue "Machine Learning in Physical Chemistry".

Kyle Mills,* Corneel Casert,* and Isaac Tamblyn*

Cite This: https://dx.doi.org/10.1021/acs.jpcc.0c06673 Read Online

ABSTRACT: We demonstrate the use of a regressive upscaling generative adversarial network (RUGAN) as an effective way to sample state space for hexagonal porous graphene sheets. The RUGAN can, after being trained on a set of small-scale examples, generate new, energetically relevant microstates (atomic configurations). The RUGAN can generate configurations across a continuum of total energy values and produce configurations at requested energy values. The microstates produced respect periodic boundary conditions, and importantly, the fully convolutional nature of the generator allows the generation of arbitrarily large microstates, after being trained on only a small-scale data set.



INTRODUCTION

In materials science and materials physics, first-principles theoretical investigation of large- and meso-scale phenomena is often intractable in part due to the difficulty of sampling configurations from a near-infinite set of microstates. Obtaining valid, large-scale, low-energy microstates that are likely to occur in a macroscopic ensemble can be a time-consuming sampling task.[1] In most cases, Nature is able to sample such distributions efficiently, although there are notable exceptions such as the glass transition. *In silico*, sampling the distribution of possible configurations is a very difficult task due to the enormous number of free parameters (e.g., position, spin, charge, etc.) defining a near-infinite number of microstates for systems of even a modest number of particles. This "curse of dimensionality", for all but the most trivial systems, precludes directly sampling configuration space at nonzero, finite temperature.[2] Traditionally, Markov chain Monte Carlo sampling methods have been devised to obtain random samples from an underlying distribution, but these algorithms, such as Metropolis–Hastings,[3,4] depend on the ability to efficiently evaluate both the energy and property of a microstate (or at least the difference in these properties between two states) which can, in many cases, be a very costly computation. Furthermore, this calculation must be carried out repeatedly, many more times than the desired number of final microstates.

The use of machine learning in materials science has become very prominent in recent years with numerous demonstrations of predictive machine learning algorithms being used for accelerated materials discovery.[5] Even using these accelerated models, which frequently perform orders of magnitude faster than traditional computational methods, brute-force sampling is challenging, if not impossible, due to the size of the configuration space.[6] Thus, generative machine learning models have entered the fields of physics and materials science, learning to produce configurations after being trained on data sets of numerous example configurations, namely, utilizing generative adversarial networks (GANs),[7−12] Boltzmann machines,[1,13] variational autoencoders,[14,15] and PixelCNN.[16,17]

In this work, we use our previously reported technique,[8] regressive upscaling generative adversarial network (RUGAN) (Figure 1a), that can generate unique microstates from the distribution of possible microstates after observing only a very small subset. By conditioning the GAN on an associated quantity, such as the total energy of the microstate, we can "request" that the generated configuration be of a specific energy. Most importantly, our RUGAN can transfer the

Received: July 21, 2020
Revised: September 18, 2020
Published: September 24, 2020

https://dx.doi.org/10.1021/acs.jpcc.0c06673
J. Phys. Chem. C XXXX, XXX, XXX−XXX

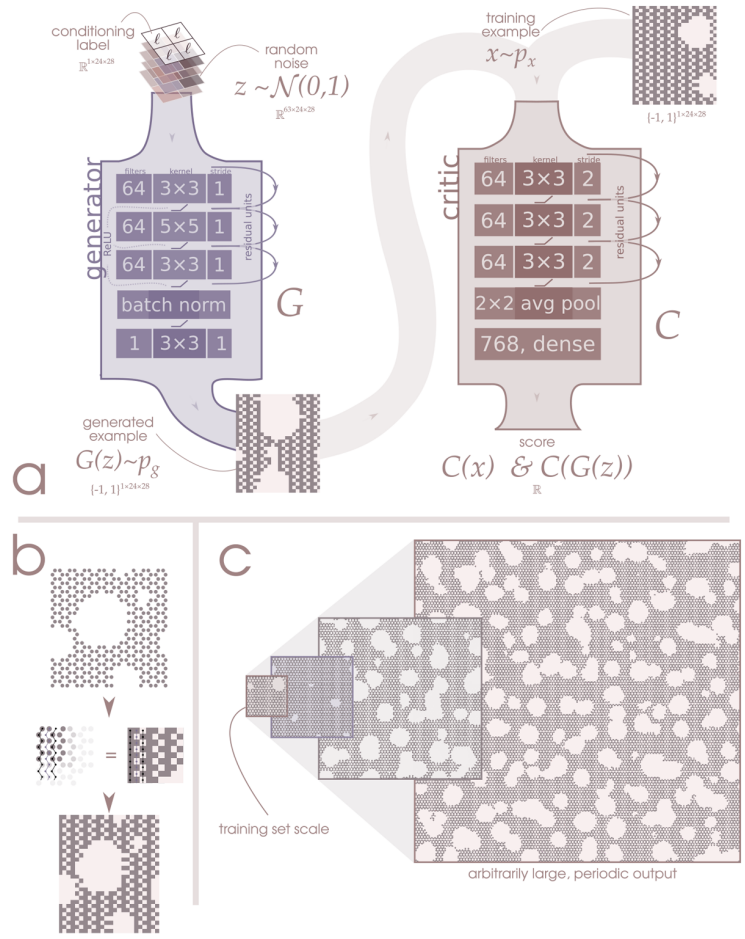


Figure 1. (a) Schematic representation of the regressive upscaling generative adversarial network (RUGAN) used in this work. The generator G takes a latent vector as input (concatenated with a conditioning label channel) and, using translationally invariant convolutional layers, produces an output encoding of a microstate. The critic C takes the proposed microstates from the generator in addition to microstates from a training set and learns to assign a score, differentiating whether the input came from the generator distribution p_g or the training set distribution p_x. (b) The encoding used to represent the hexagonal lattice on a 2d rectangular grid. (c) Through the adversarial training procedure, the generator of the RUGAN is able to learn relevant features from small-scale training examples and extend that knowledge to large-scale microstate generation. Since the generator uses only translationally invariant convolutional layers, increasing the size of the input latent vector consequently increases the spatial scale of the output microstate. Importantly, large-scale generated microstates respect periodic boundary conditions so they can be easily used with standard electronic structure approaches common in materials simulation.

knowledge learned by observing small-scale microstates to generate arbitrarily large-scale states beyond those used in training (Figure 1c); it is not limited to small-scale generation.[9] This technique enables one to access large-scale microstates while only running expensive sampling methods on a small number of small systems.

METHODS

We demonstrate the technique on a data set of porous graphene sheets, previously presented in ref 18. The study of such systems could be useful in predicting large-scale material properties, such as how the strength of a material depends on hole size or hole density (for example), but acquiring a sufficient number of relevant, large-scale microstates so as to compute statistics is prohibitively expensive. The sheets are approximately 35 Å × 35 Å with a random number of randomly sized holes introduced. To represent these structures in a way amenable to a convolutional neural network, we "zig-zag" across the hexagonal lattice, recording the presence of an atom as a 1 and the absence as a −1 at each site (Figure 1b). This results in a 24 × 28 rectangular lattice, representing the hexagonal lattice on which the graphetic sheets are defined. Others have implemented rectangular encodings of hexagonal sheets by encoding the possible pairs of atoms as a single variable.[19] This has several drawbacks, however. Firstly, the number of possible values required scales polynomially with the number of species present in the lattice and exponentially with the number of atoms in the encoding; in the case of Dong et al., they limited their investigation to only two of the nine possible atom pairs when dealing with three species. Furthermore, a rectangular encoding of a hexagonal sheet is not translationally invariant with respect to macroscopic features. Under a rectangular encoding, the same multi-atom feature encodes differently depending on if it is shifted left or right by a single atom. Since our encoding maps a single atom to a single pixel, the number of possible values at each site scales linearly with the number of species. Furthermore, since we perform convolutions using a library designed for

https://dx.doi.org/10.1021/acs.jpcc.0c06673
J. Phys. Chem. C XXXX, XXX, XXX−XXX

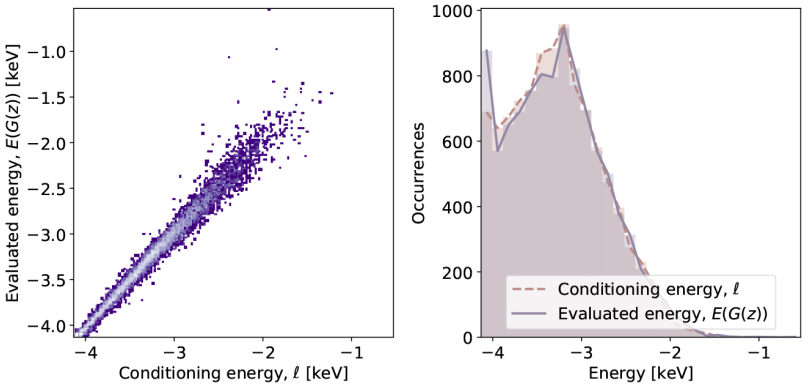


Figure 2. On the left we plot a heatmap of the energy of the generated microstates from the RUGAN (computed using the extensive deep neural network of ref 18) against the energy requested by means of the conditioning label. The color indicates the frequency of each point, with brighter colors occurring more often. On the right, we plot a histogram of the two distributions. The diagonal trend on the left and the closely matching distributions on the right confirm that the generator has indeed learned to produce configurations that match the requested energy values at the same length scale as the training data.

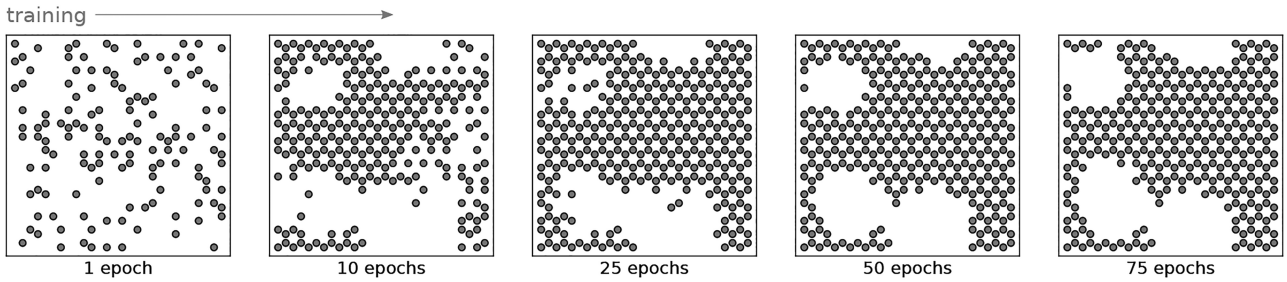


Figure 3. We plot the evolution of the generator output for several epochs during early training. Identical latent arrays are fed into the generator at each epoch shown. After a single epoch, the output is very noisy but rapidly begins to resemble a porous sheet. Through the training process, the generator refines its intuition.

hexagonally sampled data, the encoding is translationally invariant. Thus, in theory, the complexity of the neural network need not increase if more species are introduced, so long as it is of sufficient complexity to represent the relevant inter-species interactions. Future work is required to adapt this technique to more complicated systems, for example, where atoms are not confined to a lattice or if functionalization of atoms is introduced. In such situations, continuous representations[15,20,21] might prove useful.

Our new generative adversarial network[22] is based on a conditional Wasserstein GAN.[23,24] The generator $G(z)$ in our work maps latent samples to new configurations, a technique inspired by texture synthesis.[25] It is comprised of three residual convolutional layers,[26] one batch normalization layer and a final 2d convolutional layer. All convolution operations are implemented with periodic padding and are implemented in hexagonal coordinates using HexagDLy.[27]

The generator takes as input a block of noise sampled from a Gaussian distribution $z_R \in \mathcal{N}(0, 1)^{63\times24\times28}$ concatenated with a label channel $L = \{l\}^{1\times24\times28}$ where l is the conditioning value, for a full latent input of $z \in \mathbb{R}^{64\times24\times28}$. Through its several layers, the generator transforms this input into an output of $G(z) \in \{-1, 1\}^{1\times24\times28}$. Sigmoid activation is used to bound the output values, and a mask is applied to force a valid hexagonal structure (some sites must always be empty). The critic C receives encoded configurations x from the training set distribution p_x, as well as the examples $G(z)$ originating from the generator's output distribution p_g. Through a series of layers shown in Figure 1a, it outputs a single scalar value. Through training, the critic is optimized to calculate the Wasserstein distance between p_x and p_g, essentially differentiating between "true" ($x \sim p_x$) and "artificial" ($G(z) \sim p_z$) examples. Through the training process, the generator also learns how to improve its capability, with p_g improving toward matching p_x.

We train the RUGAN for 1000 epochs on a data set of 32 768 porous graphene sheets and their corresponding density functional theory energy, computed using the extensive deep neural network trained and validated in ref 18. We use the Adam optimizer[28] with a learning rate of $\alpha = 10^{-4}$ and hyperparameters $\beta_1 = 0$, $\beta_2 = 0.9$, and $\epsilon = 10^{-8}$ to minimize the WGAN loss function[29,30] with regularizing parameters $\lambda_1 = 10$ and $\lambda_2 = 2$ for the gradient penalty and consistency terms, respectively. To stabilize the prediction of the Wasserstein distance, we perform 10 weight updates (10 batches) of the critic for every update of the generator. After 1000 epochs, the generator has learned to approximate the data distribution, and we can use $G(z)$ to generate examples that appear to come from p_x. Furthermore, since G was conditioned on the energies of the microstates, we can request microstates of a specific energy.

Upscaling. In the generator, we intentionally use only translationally invariant layers (e.g., convolutional layers) as well as periodic padding. Doing so enables the generation of larger-scale microstates that adhere to the periodic boundary

https://dx.doi.org/10.1021/acs.jpcc.0c06673
J. Phys. Chem. C XXXX, XXX, XXX−XXX

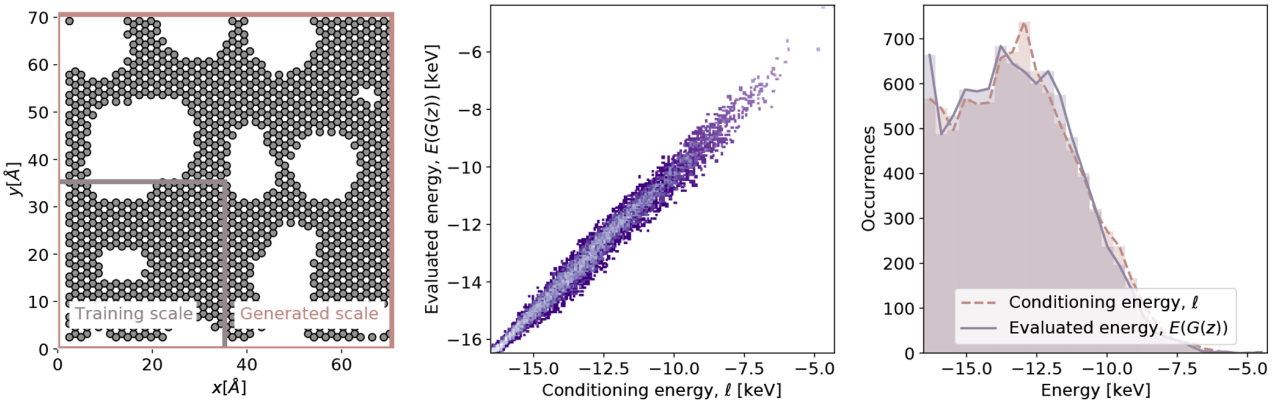


Figure 4. On the left, we show an example large-scale configuration, superimposed with an indicator of the size of the training set. In the center we plot a heatmap of the energy of the larger microstates produced by the RUGAN generator (computed using the extensive deep neural network of ref 18) against the energy requested by means of the conditioning label. The color indicates the density of each point, with brighter colors occurring more often. On the right, we plot a histogram of the two distributions. The diagonal trend in the center and the closely matching distributions on the right confirm that the generator has indeed learned to produce configurations that match the requested energy values.

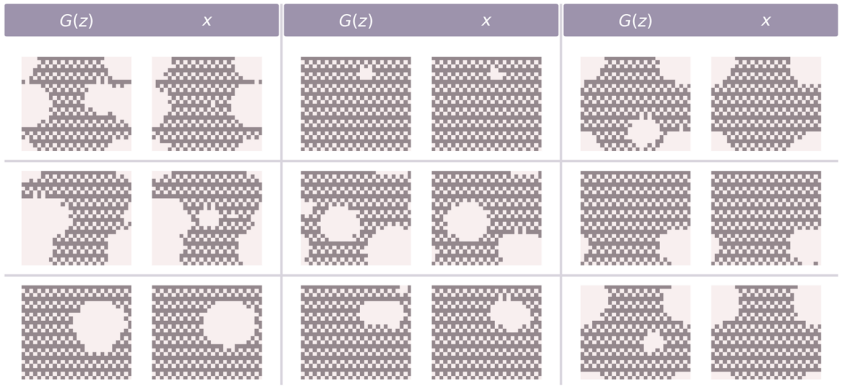


Figure 5. We plot several example encodings from the generator (labeled $G(z)$), as well as the most similar training example (labeled x). The variation between each $G(z)$ and its corresponding x highlights that RUGAN captures the underlying distribution of the training data, rather than just memorizing the training data.

conditions merely by changing the size of the input random noise. For example, if we feed a $z \in \mathbb{R}^{64\times48\times56}$ noise block into the generator, $G(z)$ will produce a microstate $G(z) \in \{-1, 1\}^{1\times48\times56}$, representing a 70 Å × 70 Å sheet, four times as large as the states on which it was trained.

RESULTS

We train the RUGAN on the data set of porous graphene sheets from ref 18, conditioning the generator on the provided total energy (computed under the density functional theory (DFT) framework), normalized by the surface area. Once trained, the job of the critic is complete, and we focus our attention on what the generator has achieved. We feed 10 000 randomly generated latent blocks z_R and conditioning values l randomly sampled from the training data into the trained generator and receive 10 000 encoded microstates. To verify that the generated examples do indeed represent the energy requested through the conditioning, we used the extensive deep neural network, E, trained independently of this work[18] to evaluate the energy of each generated microstate. Thus, we can compare the distribution of l to the distribution of $E(G(z))$ to investigate the hypothesis that the generator has successfully learned the training distribution p_x. We plot the two distributions in Figure 2, and it appears that they closely match. We plot several example encodings from the generator, as well as those in the training set that are most similar in Figure 5, highlighting that RUGAN captures the underlying distribution of these encodings, rather than just memorizing the training data. Additionally, we show the evolution of the generator during the training process in Figure 3.

Next, we repeat the process, feeding in larger random blocks to the generator, which in turn produces spatially larger output configurations. We again use the extensive deep neural network to evaluate the "DFT energy" of the generated configurations and compare it to the energies requested through conditioning. Similar to before, the distributions are plotted in Figure 4 and appear to match. We can conclude that the generator is successful at producing output at scales larger than the scale on which it was trained.

Since the configurations vary in the number of atoms, the total energy is dominated by the number of atoms present. As such, while it appears the generator is capable of reproducing the qualitative features of the training set, to successfully match the conditioning energy, it needs only to produce the correct number of atoms. Therefore, we additionally train a separate RUGAN, conditioning on the energy per atom. The results are presented in Figure 6. When conditioned on energy per atom,

https://dx.doi.org/10.1021/acs.jpcc.0c06673
J. Phys. Chem. C XXXX, XXX, XXX−XXX

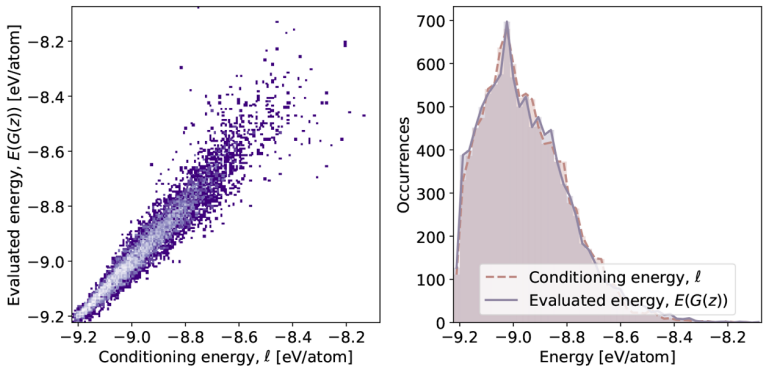


Figure 6. RUGAN conditioned on energy per atom performs more poorly than one conditioned on total energy. Using energy per atom as a conditioning label makes the task of the generator more difficult; it must produce a configuration with the correct number of atoms and with relevant features that would be present at that energy value. Additional training data and additional training time are likely to improve the performance.

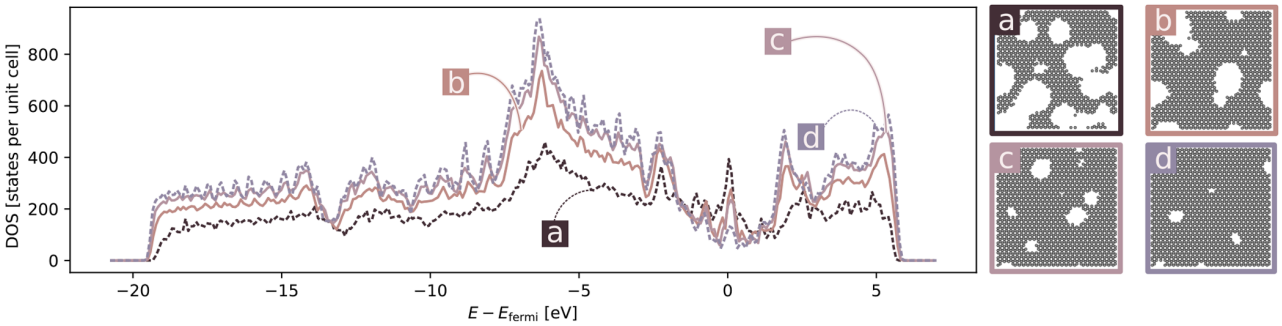


Figure 7. Our periodic configurations are fully compatible with standard materials simulation protocols. Here we plot the density of states for four of the large-scale distributions that were created using the generator trained on small-scale structures, obtained using VASP.[31−34]

the generator performs more poorly than when conditioned on total energy; this is due to the fact that it must produce configurations with both the correct number of atoms and structures representative of the training set.

We can use the generated microstates as starting configurations for further density functional theory calculations. For example, in Figure 7 we plot the density of states for four different configurations generated through the RUGAN approach.

A trained RUGAN enables the fast and accurate generation of energetically relevant microstates after being provided a small number of training examples, enabling rapid sampling of configuration space. Furthermore, the RUGAN, with its translationally invariant and periodic design, empowers one to sample the configuration space of large-scale structures, a task that is traditionally infeasible, providing a basis for the investigation of large-scale structures.

CONCLUSION

In this work, we show that a RUGAN can be trained to generate unique microstates from the distribution of possible microstates after observing only a very small subset of the total state space. It learns adversarially, with one component of the network (the generator) learning to generate examples that appear to come from the training set and another component (the critic) used to provide criticism to the generator and improve its capability. By conditioning the GAN on an associated quantity, in our case, the total energy of the microstate, we show that the trained generator can generate configurations at requested energies.

Most importantly, through careful choice of the network architecture, our RUGAN can transfer the knowledge learned by observing small-scale microstates to generate large-scale states beyond what was presented during training. This is very powerful, as the calculations required to obtain the conditioning energy are typically expensive (or altogether computationally impossible) and could not be carried out on large-scale systems. RUGAN is designed to capture, reproduce, and most importantly extend the insights present in a data set so that one can carry out the expensive computation on a small system, and the RUGAN can then transfer these insights to the large scale.

AUTHOR INFORMATION

Corresponding Authors

Kyle Mills − *University of Ontario Institute of Technology, Oshawa, Canada; Vector Institute for Artificial Intelligence, Toronto, Canada;* orcid.org/0000-0003-1768-6873; Email: kyle.mills@ontariotechu.net

Isaac Tamblyn − *National Research Council of Canada, Ottawa, Canada; Vector Institute for Artificial Intelligence, Toronto, Canada;* orcid.org/0000-0002-8146-6667; Email: isaac.tamblyn@nrc.ca

Corneel Casert − *Ghent University, Ghent, Belgium;* Email: corneel.casert@ughent.be

Complete contact information is available at:

https://dx.doi.org/10.1021/acs.jpcc.0c06673
J. Phys. Chem. C XXXX, XXX, XXX−XXX

https://pubs.acs.org/10.1021/acs.jpcc.0c06673

Notes

The authors declare no competing financial interest.
The code and dataset is available for download at http://clean.energyscience.ca/codes.

■ ACKNOWLEDGMENTS

The authors would like to thank Jannes Nys and Tom Vieijra for useful discussions. I.T. and K.M. acknowledge funding from NSERC. The authors would like to thank the Vector Institute for the provision of computational resources. I.T acknowledges computational support from NERSC. This work was included in an earlier form in the 2019 Neurips Machine Learning and the Physical Sciences workshop.

■ REFERENCES

(1) Noe, F.; Olsson, S.; Kohler, J.; Wu, H. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science* **2019**, *365*, 1147.
(2) Carrasquilla, J.; Melko, R. G. Machine learning phases of matter. *Nat. Phys.* **2017**, *13*, 431−434.
(3) Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H.; Teller, E. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.* **1953**, *21*, 1087−1092.
(4) Hastings, B. Y. W. K. *Biometrika* **1970**, *57*, 97−109.
(5) Schmidt, J.; Marques, M. R.; Botti, S.; Marques, M. A. Recent advances and applications of machine learning in solid-state materials science. *npj Computational Materials* **2019**, *5*, 1−36.
(6) Chen, C.; Gu, G. X. Generative Deep Neural Networks for Inverse Materials Design Using Backpropagation and Active Learning. *Advanced Science* **2020**, *7*, 1902607.
(7) Mills, K.; Tamblyn, I. Phase space sampling and operator confidence with generative adversarial networks. *arXiv preprint: 1710.08053*, **2017**.
(8) Casert, C.; Mills, K.; Vieijra, T.; Ryckebusch, J.; Tamblyn, I. Optical lattice experiments at unobserved conditions and scales through generative adversarial deep learning. *arXiv preprint: 2002.07055*, **2019**.
(9) Dong, Y.; Li, D.; Zhang, C.; Wu, C.; Wang, H.; Xin, M.; Cheng, J.; Lin, J. Inverse Structural Design of Graphene/Boron Nitride Hybrids by Regressional GAN. *arXiv preprint: 1908.07959*, **2019**.
(10) Kim, B.; Lee, S.; Kim, J. Inverse design of porous materials using artificial neural networks. *Science Advances* **2020**, *6*, eaax9324.
(11) Mao, Y.; He, Q.; Zhao, X. Designing complex architectured materials with generative adversarial networks. *Science Advances* **2020**, *6*, eaaz4169.
(12) Méndez-Lucio, O.; Baillif, B.; Clevert, D. A.; Rouquié, D.; Wichard, J. De novo generation of hit-like molecules from gene expression signatures using artificial intelligence. *Nat. Commun.* **2020**, *11*, 1−10.
(13) Carrasquilla, J.; Torlai, G.; Melko, R. G.; Aolita, L. Reconstructing quantum states with generative models. *Nature Machine Intelligence* **2019**, *1*, 155−161.
(14) Stein, H. S.; Guevarra, D.; Newhouse, P. F.; Soedarmadji, E.; Gregoire, J. M. Machine learning of optical properties of materials-predicting spectra from images and images from spectra. *Chemical Science* **2019**, *10*, 47−55.
(15) Noh, J.; Kim, J.; Stein, H. S.; Sanchez-Lengeling, B.; Gregoire, J. M.; Aspuru-Guzik, A.; Jung, Y. Inverse Design of Solid-State Materials via a Continuous Representation. *Matter* **2019**, *1*, 1370−1384.
(16) Kilgour, M.; Gastellu, N.; Hui, D.; Bengio, Y.; Simine, L. Towards Generation of Multi-Scale Amorphous Molecular Structures using Deep Learning: a study in 2D. *J. Phys. Chem. Lett.* **2020**, 8532.
(17) van den Oord, A.; Kalchbrenner, N.; Vinyals, O.; Espeholt, L.; Graves, A.; Kavukcuoglu, K. Conditional Image Generation with PixelCNN Decoders. *Adv. Neural Information Processing Systems* **2016**, 4797−4805.
(18) Mills, K.; Ryczko, K.; Luchak, I.; Domurad, A.; Beeler, C.; Tamblyn, I. Extensive deep neural networks for transferring small scale learning to large scale systems. *Chemical Science* **2019**, *10*, 4129−4140.
(19) Dong, Y.; Wu, C.; Zhang, C.; Liu, Y.; Cheng, J.; Lin, J. Bandgap prediction by deep learning in configurationally hybridized graphene and boron nitride. *npj Computational Materials* **2019**, *5*, 26.
(20) Brockherde, F.; Vogt, L.; Li, L.; Tuckerman, M. E.; Burke, K.; Müller, K. R. Bypassing the Kohn-Sham equations with machine learning. *Nat. Commun.* **2017**, *8*, 1−10.
(21) Ryczko, K.; Mills, K.; Luchak, I.; Homenick, C.; Tamblyn, I. Convolutional neural networks for atomistic systems. *Comput. Mater. Sci.* **2018**, *149*, 134−142.
(22) Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Networks. *arXiv preprint: 1406.2661*, **2014**, 1−9.
(23) Mirza, M.; Osindero, S. Conditional Generative Adversarial Nets. *arXiv:1411.1784* **2014**, 1−7.
(24) Arjovsky, M.; Chintala, S.; Bottou, L. Wasserstein gan. *arXiv preprint: 1701.07875*, **2017**.
(25) Jetchev, N.; Bergmann, U.; Vollgraf, R. Texture synthesis with spatial generative adversarial networks. *arXiv preprint: 1611.08207*, **2016**.
(26) Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A. Improved training of wasserstein GANs. *Advances in Neural Information Processing Systems* **2017**, 5768−5778.
(27) Steppa, C.; Holch, T. L. HexagDLy—Processing hexagonally sampled data with CNNs in PyTorch. *SoftwareX* **2019**, *9*, 193−198.
(28) Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980* **2014**, 1−15.
(29) Wei, X.; Gong, B.; Liu, Z.; Lu, W.; Wang, L. Improving the Improved Training of Wasserstein GANs: A Consistency Term and Its Dual Effect. *arXiv preprin: 1803.01541t*, **2018**.
(30) Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A. Improved Training of Wasserstein GANs. *arXiv preprint: 1704.00028*, **2017**.
(31) Kresse, G.; Hafner, J. Ab initio molecular dynamics for liquid metals. *Phys. Rev. B: Condens. Matter Mater. Phys.* **1993**, *47*, 558−561.
(32) Kresse, G.; Hafner, J. Ab initio molecular-dynamics simulation of the liquid-metal−amorphous-semiconductor transition in germanium. *Phys. Rev. B: Condens. Matter Mater. Phys.* **1994**, *49*, 14251−14269.
(33) Kresse, G.; Furthmüller, J. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. *Comput. Mater. Sci.* **1996**, *6*, 15−50.
(34) Kresse, G. Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set. *Phys. Rev. B: Condens. Matter Mater. Phys.* **1996**, *54*, 11169−11186.

Chapter 5

Reinforcement learning

5.1 Controlled online optimization learning

Reinforcement learning (RL) is a branch of optimal control that has gained considerable attention in recent years after the concepts of artificial intelligence, mainly deep neural networks, were incorporated into the reinforcement learning theory.

RL is interesting in that it tackles problems fundamentally different than those of normal supervised (or unsupervised) learning. RL is used to learn processes, that is, learn trajectories through interaction with an environment. It seeks to control the outcome of the process and steer the trajectory toward a desired outcome. RL is useful in situations where the success of a given outcome is known (i.e. a given outcome can be "scored"), the steps required to reach that outcome are unknown.

In "Finding the ground state of spin Hamiltonians with reinforcement learn-

ing", we present an approach, which we call "Controlled online optimization learning" (COOL), that uses reinforcement learning to control the temperature of simulated annealing experiments for optimizing spin glass systems. With simulated annealing, the ideal temperature schedule is one that cools the system slowly enough so that it does not get trapped in local minima, but quickly enough so as not to incur undue computational expense. This is an ideal problem for RL, as comparative success can easily be determined, but the ideal schedule for a given Hamiltonian is not known and changes dynamically based on the stochastic evolution of the system.

Author contributions: All authors contributed to the ideation and design of the research. Pooya Ronagh and Isaac Tamblyn jointly supervised this work and contributed to revisions of the manuscript.

nature machine intelligence

ARTICLES

https://doi.org/10.1038/s42256-020-0226-x

Check for updates

Finding the ground state of spin Hamiltonians with reinforcement learning

Kyle Mills[1,2,3]✉, Pooya Ronagh[1,4,5]✉ and Isaac Tamblyn[2,3,6]✉

Reinforcement learning (RL) has become a proven method for optimizing a procedure for which success has been defined, but the specific actions needed to achieve it have not. Using a method we call 'controlled online optimization learning' (COOL), we apply the so-called 'black box' method of RL to simulated annealing (SA), demonstrating that an RL agent based on proximal policy optimization can, through experience alone, arrive at a temperature schedule that surpasses the performance of standard heuristic temperature schedules for two classes of Hamiltonians. When the system is initialized at a cool temperature, the RL agent learns to heat the system to 'melt' it and then slowly cool it in an effort to anneal to the ground state; if the system is initialized at a high temperature, the algorithm immediately cools the system. We investigate the performance of our RL-driven SA agent in generalizing to all Hamiltonians of a specific class. When trained on random Hamiltonians of nearest-neighbour spin glasses, the RL agent is able to control the SA process for other Hamiltonians, reaching the ground state with a higher probability than a simple linear annealing schedule. Furthermore, the scaling performance (with respect to system size) of the RL approach is far more favourable, achieving a performance improvement of almost two orders of magnitude on $L = 14^2$ systems. We demonstrate the robustness of the RL approach when the system operates in a 'destructive observation' mode, an allusion to a quantum system where measurements destroy the state of the system. The success of the RL agent could have far-reaching impacts, from classical optimization, to quantum annealing and to the simulation of physical systems.

The process of annealing is used in metallurgy and materials science to equilibrate the positions of atoms to obtain perfect low-energy crystals. Heat provides the energy necessary to break atomic bonds, and high-stress interfaces are eliminated by the migration of defects. By slowly cooling the metal to room temperature, the metal atoms become energetically locked in a lattice structure more favourable than the original structure. Metallurgists can tune the temperature schedule to arrive at final products that have desired characteristics, such as ductility and hardness. Annealing is a biased stochastic search for the ground state.

An analogous in silico technique, simulated annealing (SA)[1], can be used to find the ground state of spin glass models, an NP-hard problem (NP, non-deterministic polynomial time)[2]. A spin glass is a graphical model consisting of binary spins σ_i. The connections between spins are defined by the coupling constants J_{ij}, and a linear term with coefficients h_i can apply a bias to individual spins. The Hamiltonian

$$\mathcal{H} = -\sum_{i<j} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i, \quad \sigma_i = \pm 1$$

defines the energy of the microstates[3]. The choices of the quadratic coupling coefficients J_{ij} and the linear bias coefficients h_i effect the interesting dynamics of the model: J_{ij} can be randomly distributed according to a Gaussian distribution[3], encompass all i, j combinations for a fully connected Hamiltonian, or be limited to short-range (for example, nearest-neighbour, $\langle i, j\rangle$) interactions, to name a few. For example, when the positive, unit-magnitude coupling is limited to nearest-neighbour pairs, the ubiquitous ferromagnetic Ising model[4] is recovered. Examples of the Hamiltonians we investigate in this work are presented in Fig. 1 and discussed in further detail in the 'Hamiltonians' section.

Finding the ground state of such systems (that is, 'solving') is interesting from the perspective of thermodynamics, as one can observe phenomena such as phase transitions[5,6], but it is also practically useful as discrete optimization problems can be mapped to spin glass models (for example, the travelling salesperson problem or the knapsack problem)[7]. The Metropolis–Hastings algorithm[8,9] can be used to simulate the spin glass at arbitrary temperature, T; thus, it is used ubiquitously for SA. By beginning the simulation at a high temperature, one can slowly cool the system over time, providing sufficient thermal energy to escape local minima, and arrive at the ground state 'solution' to the problem. The challenge is to find a temperature schedule that minimizes computational effort while still arriving at a satisfactory solution; if the temperature is reduced too rapidly, the system will become trapped in a local minimum, and reducing the temperature too slowly results in an unnecessary computational expense. Kirkpatrick and colleagues[1,10] proposed starting at a temperature that results in an 80% acceptance ratio (that is, 80% of Metropolis spin flips are accepted) and reducing the temperature geometrically. They also recommended monitoring the objective function and reducing the cooling rate if the objective value (for example, the energy) drops too quickly. More sophisticated adaptive temperature schedules have been investigated[11]; however, simple linear and reciprocal temperature schedules are commonly used in practice[12,13]. We will refer to SA using a linear schedule as 'classic SA' throughout this work. Nevertheless, in his 1987 paper, Bounds[14] said that 'choosing an annealing schedule for practical purposes is still something of a black art'.

[1]1QB Information Technologies (1QBit), Vancouver, British Columbia, Canada. [2]University of Ontario Institute of Technology, Oshawa, Ontario, Canada. [3]Vector Institute for Artificial Intelligence, Toronto, Ontario, Canada. [4]Institute for Quantum Computing (IQC), Waterloo, Ontario, Canada. [5]Department of Physics and Astronomy, University of Waterloo, Waterloo, Ontario, Canada. [6]National Research Council Canada, Ottawa, Ontario, Canada. ✉e-mail: kyle.mills@1qbit.com; pooya.ronagh@1qbit.com; isaac.tamblyn@nrc.ca

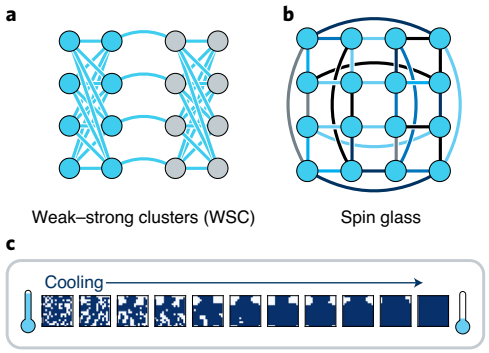


Fig. 1 | Two classes of Hamiltonian problems are depicted. a, The weak–strong clusters (WSC) model comprises two bipartite clusters. The left cluster is biased upward ($h_i > 0$) and the right cluster is biased downward ($h_i < 0$). All couplings are equal and of unit magnitude. The two clusters are coupled via the eight central nodes. This model exhibits a deep local minimum very close in energy to the model's global minimum. When initialized in the local minimum, the RL agent is able to learn schemes to escape the local minimum and arrive at the global minimum, without any explicit knowledge of the Hamiltonian. **b**, An example spin glass model. The nodes are coupled to nearest neighbours with random Gaussian-distributed coupling coefficients. The nodes are unbiased ($h_i = 0$), and the couplings are changed at each instantiation of the model. The reinforcement learning (RL) algorithm is able to learn a dynamic temperature schedule by observing the system throughout the annealing process, without explicit knowledge of the form of the Hamiltonian, and the learned policy can be applied to all instances of randomly generated couplings. We demonstrate this on variably sized spin glasses and investigate the scaling with respect to a classic linear SA schedule. **c**, Snapshots of a sample progression of a configuration undergoing SA under the ferromagnetic Ising model Hamiltonian and a constant cooling schedule. The terminal state, all spins-up, is the ground state, and this anneal would be considered successful.

When framed in the advent of quantum computation and quantum control, establishing robust and dynamic scheduling of control parameters becomes even more relevant. For example, the same optimization problems that can be cast as classical spin glasses are also amenable to quantum annealing (QA)[15–19], exploiting, in lieu of thermal fluctuations, the phenomenon of quantum tunnelling[20–22] to escape local minima. QA was proposed by Finnila et al.[23] and Kadowaki and Nishimori[24] and, in recent years, physical realizations of devices capable of performing QA (quantum annealers) have been developed[25–28] and are being commercialized rapidly. As these technologies progress and become more commercially viable, practical applications[19,29] will continue to be identified, and resource scarcity will spur the already extant discussion of the efficient use of annealing hardware[30,31].

Nonetheless, there are still instances where the classical (SA) outperforms the quantum (QA)[32], and improving the former should not be undervalued. In silico and hardware annealing solutions such as Fujitsu's FPGA-based digital annealer[33], NTT's laser-pumped coherent Ising machine (CIM)[34–36] and the quantum circuit model algorithm known as QAOA[37,38] all demand the scheduling of control parameters, whether it is the temperature in the case of the digital annealer or the power of the laser pump in the case of the CIM. Heuristic methods based on trial-and-error experiments are commonly used to schedule these control parameters, and an automatic approach could expedite development and improve the stability of such techniques.

In this work, we demonstrate the use of a reinforcement learning (RL) method to learn the 'black art' of SA temperature scheduling, and show that an RL agent is able to learn dynamic control parameter schedules for various problem Hamiltonians. The schedules that the RL agent produces are dynamic and reactive, adjusting to the current observations of the system to reach the ground state quickly and consistently without a priori knowledge of a given Hamiltonian. We believe that RL will be important for quantum information processing, especially for hardware- and software-based control.

The environment and architecture

Reinforcement learning. Reinforcement learning is a branch of dynamic programming whereby an agent, residing in state s_t at time t, learns to take an action a_t that maximizes a cumulative reward signal R by dynamically interacting with an environment[39]. Through the training process, the agent arrives at a policy π that depends on some observation (or 'state') of the system, s. In recent years, neural networks have taken over as the de facto function approximator for the policy. Deep reinforcement learning has seen unprecedented success, achieving superhuman performance in a variety of video games[40–43], board games[44–46], and other puzzles[47,48]. Although many RL algorithms exist, we have chosen to use proximal policy optimization (PPO)[49], implemented within Stable Baselines[50] for its competitive performance on problems with continuous action spaces.

The environment. We developed an OpenAI gym[51] environment, which serves as the interface to the 'game' of simulated annealing. Let us now define some terminology and parameters important to SA. For a given Hamiltonian, defining the interactions of L spins, we create N_{reps} randomly initialized replicas (unless otherwise specified). The initial spins of each replica are drawn from a Bernoulli distribution with probability of a spin-up being randomly drawn from a uniform distribution. These independent replicas are annealed in parallel. The replicas follow an identical temperature schedule with their uncoupled nature providing a mechanism for the statistics of the system to be represented through an ensemble of measurements. In the context of the Metropolis–Hastings framework, we define one 'sweep' to be L proposed random spin flips (per replica), and one 'step' to be N_{sweeps} sweeps. After every step, the environment returns an observation of the current state s_t of the system, an $N_{\text{reps}} \times L$ array consisting of the binary spin values present. This observation can be used to make an informed decision of the action a_t that should be taken. The action, a single scalar value, corresponds to the total inverse temperature change $\Delta\beta$ (where $\beta = 1/T$) that should be carried out over the subsequent step. The choice of action is provided to the environment, and the process repeats until N_{steps} steps have been taken, comprising one full anneal, or 'episode' in the language of RL. If the chosen action would result in the temperature becoming negative, no change is made to the temperature and the system continues to evolve under the previous temperature.

In our investigations, we choose $N_{\text{steps}} = 40$ and $N_{\text{sweeps}} = 100$, resulting in 4,000 sweeps per episode. These values define the maximum size of system we can compare to classic SA. This number of sweeps is sufficient for a linear schedule to attain measurable success on all but the largest system size we investigate.

Observations. For the classical version of the problem, an observation consists of the explicit spins of an ensemble of replicas. In the case of an unknown Hamiltonian, the ensemble measurement is important as the instantaneous state of a single replica does not provide sufficient information about the current temperature of the system. Providing the agent with multiple replicas allows it to compute statistics and have the possibility of inferring the temperature. For example, if there is considerable variation among replicas, then the system is probably hot, whereas if most replicas look the same, the system is probably cool.

When discussing a quantum system, where the spins represent qubits, direct mid-anneal measurement of the system is not possible

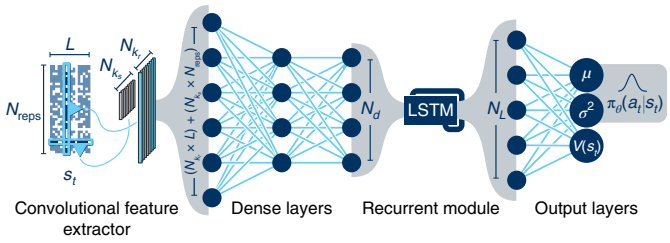


Fig. 2 | A neural network is used to learn the control parameters for several SA experiments. By observing a lattice of spins, the neural network can learn to control the temperature of the system in a dynamic fashion, annealing the system to the ground state. The spins at time t form the state s_t fed into the network. Two concurrent convolutional layers extract features from the state. These features are combined with a dense layer and fed into a recurrent module (an LSTM module) capable of capturing temporal characteristics. The LSTM module output is reduced to two parameters used to form the policy distribution $\pi_\theta(a_t|s_t)$ as well as to approximate the value function $V(s_t)$ used for the generalized advantage estimate.

as measurement causes a collapse of the wavefunction. To address this, we discuss experiments conducted in a 'destructive observation' environment, where measurement of the spins is treated as a 'one-time' opportunity for inclusion in RL training data. The subsequent observation is then based on a different set of replicas that have evolved through the same schedule, but from different initializations.

When running the classic SA baselines, to keep comparison fair, each episode consists of N_{reps} replicas as in the RL case. If even one replica reaches the ground state, the episode is considered a success.

Reinforcement learning algorithm. Through the framework of RL, we wish to produce a policy function $\pi_\theta(a_t|s_t)$ that takes the observed binary spin state $s_t \in \{-1, 1\}^{N_{\text{reps}} \times L}$ and produces an action a_t corresponding to the optimal change in the inverse temperature.

We now briefly introduce PPO[49]. First we define our policy $\pi_\theta(a_t|s_t)$ as the likelihood that the agent will take action a_t while in state s_t; through training, the desire is that the best choice of action will become the most probable. To choose an action, this distribution can be sampled. We will use a neural network that is parameterized by weights θ to represent the policy by assuming that $\pi_\theta(a_t|s_t)$ is a normal distribution and interpreting the output nodes of the neural network as the mean, μ, and variance, σ^2.

We define a function $Q_{\pi_\theta}(s_t, a_t)$ as the expected future discounted reward if the agent takes action a_t at time t and then follows policy π_θ for the remainder of the episode. We additionally define a value function $V_{\pi_\theta}(s_t)$ as the expected future discounted reward starting from state s_t and following the current policy π_θ until the end of the episode. We introduce the concept of *advantage*, $\hat{A}_t(s_t, a_t)$, as the difference between these two quantities. Q_{π_θ} and V_{π_θ} are not known and must be approximated. We assume the features necessary to represent π are generally similar to the features necessary to estimate the value function, and thus we can use the same neural network to predict the value function by merely having it output a third quantity.

$\hat{A}_t$ is effectively an estimate of how much better the agent did in choosing action a_t, compared to what was expected. We construct the typical policy gradient cost function by coupling the advantage of a state–action pair with the probability of the action being taken:

$$L^{\text{PG}}(\theta) = \hat{\mathbb{E}}_t\left[\log \pi_\theta(a_t|s_t)\hat{A}_t\right]$$

which we want to maximize by modifying the weights θ through the training process. It is, however, more efficient to maximize the improvement ratio r_t of the current policy over a policy from a previous iteration $\pi_{\theta_{\text{old}}}$ (refs. [52,53]):

$$L^{\text{TRPO}}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] \equiv \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

Note, however, that maximizing this quantity can be trivially achieved by making the new policy drastically different from the old policy, which is not the desired behaviour. The PPO algorithm[49] deals with this by clipping the improvement and taking the minimum

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

To train the value function estimator, a squared error is used:

$$L^{\text{VF}}(\theta) = \hat{\mathbb{E}}_t[(V_{\pi_\theta}(s_t) - V_t^{\text{targ}})^2]$$

and to encourage exploration, an entropic regularization functional S is used. This all amounts to a three-term cost function

$$L^{\text{PPO}}(\theta) = \hat{\mathbb{E}}_t\left[L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t)\right]$$

where c_1 and c_2 are hyperparameters.

Policy network architecture. The neural network is composed of two parts: a convolutional feature extractor and a recurrent network to capture the temporal characteristics of the problem (Fig. 2). The feature extractor comprises two parallel two-dimensional (2D) convolutional layers. The first convolutional layer has N_{k_r} kernels of size $1 \times L$, and aggregates along the replicas dimension, enabling the collection of spin-wise statistics across the replicas. The second convolutional layer has N_{k_s} kernels of size $N_{\text{reps}} \times 1$ and slides along the spin dimension, enabling the aggregation of replica-wise statistics across the spins. The outputs of these layers are flattened, concatenated and fed into a dense layer of size N_{d} hidden nodes. This operates as a latent space encoding for input to a recurrent neural network (a long short-term memory, or LSTM, module [54]), used to capture the sequential nature of our application. The latent output of the LSTM module is of size N_{L}. For simplicity, we set $N_{k_r} = N_{k_s} = N_{\text{d}} = N_{\text{L}} = 64$. All activation functions are hyperbolic tangent (tanh) activations. Because a_t can assume a continuum of real values, this task is referred to as having a continuous action space, and thus standard practice is for the network to output two values corresponding to the first and second moments of a normal distribution, which can be sampled to produce predictions.

Reward. At the core of RL is the concept of reward engineering, that is, developing a reward scheme to inject a notion of success into the system. As we only care about reaching the ground state by the end of a given episode, we use a sparse reward scheme, with a reward of zero for every time step before the terminal step, and a reward equal to the negative of the minimum energy as the reward for the terminal step:

$$R_t = \begin{cases} 0, & t < N_{\text{steps}} \\ -\min_k \mathcal{H}(\phi_k(s_t)), & t = N_{\text{steps}} \end{cases} \quad (1)$$

where $k \in [1, N_{\text{reps}}]$ and

$$\phi_k(s_t) \in \{-1, 1\}^{1 \times L}$$

is an indexing function that returns the binary spin values for the kth replica of state s_t. This reward function is agnostic to system size; as the system size increases, the correlation time will also

increase, and additional sweeps may be required between actions, but the reward function remains applicable. Furthermore, with this reward scheme, we encourage the agent to arrive at the lowest possible energy by the time the episode terminates, without regard to what it does in the interim. In searching for the ground state, the end justifies the means.

Hyperparameters. When optimizing the neural network, we use a PPO discount factor of $\gamma = 0.99$, eight episodes between weight updates, a value function coefficient of $c_1 = 0.5$, an entropy coefficient of $c_2 = 0.001$, a clip range of $\epsilon = 0.05$, a learning rate of $\alpha = 1 \times 10^{-6}$ and a single minibatch per update. Each agent is trained over the course of 25,000 episodes (anneals), with $N_{\text{steps}} = 40$ steps per episode and $N_{\text{sweeps}} = 100$ sweeps separating each observation. We used $N_{\text{reps}} = 64$ replicas for each observation.

Evaluation

Whereas the RL policy can be made deterministic, meaning a given state always produces the same action, the underlying Metropolis algorithm is stochastic; thus, we must statistically define the metric for success. We borrow this evaluation scheme from ref. [55]. Each RL episode will either result in 'success' or 'failure'. Let us define the 'time to solution' as

$$T_s = \tau n_{99} \tag{2}$$

that is, the number of episodes that must be run to be 99% sure the ground state has been observed at least one time (n_{99}), multiplied by the time τ taken for one episode. As τ depends specifically on the hardware used and the efficiency of software implementations, we will focus on n_{99} alone as the metric we desire to minimize.

Let us also define X_i as the binary outcome of the ith episode, with $X_i = 1(0)$ if at least one (none) of the N_{reps} replicas are observed to be in the ground state at episode termination. The quantity $Y \equiv \sum_{i=1}^{n} X_i$ is the number of successful episodes after a total of n episodes, and $p \equiv P(X_i = 1)$ denotes the probability that an anneal i will be successful. Thus, the probability of exactly k out of n episodes succeeding is given by the probability mass function of the binomial distribution

$$P(Y = k|n, p) = \binom{n}{k} p^k (1-p)^{n-k} \tag{3}$$

To compute the time to solution, our quantity of interest is the number of episodes n_{99} where $P = 0.99$:

$$P(Y \geq 1|n_{99}, p) = 0.99$$

From this and equation (3), it can be shown that

$$n_{99} = \frac{\log(1 - 0.99)}{\log}(1 - p)$$

In the work of Aramon et al.[55], p is estimated using Bayesian inference due to their large system sizes sometimes resulting in zero successes, precluding the direct calculation of p. In our case, to evaluate a policy, we perform 100 runs for each of 100 instances and compute p directly from the ratio of successful to total episodes, that is, $p = \bar{X}$.

Hamiltonians

We present an analysis of two classes of Hamiltonians. The first, which we call the weak–strong clusters model (WSC; Fig. 1a), is an $L = 16$ bipartite graph with two fully-connected clusters, inspired by the 'Chimera' structure used in D-Wave Systems' quantum annealing hardware[56]. In our case, one cluster is negatively biased with $h_i = -0.44$ and the other positively biased with $h_i = 1.0$. All couplings are ferromagnetic and have unit magnitude. This results in an energy landscape with a deep local minimum where both clusters are aligned to their respective biases, but a slightly lower global minimum when the two clusters are aligned together, sacrificing the benefit of bias-alignment for the satisfaction of the intercluster couplings. For all WSC runs, the spins of the lattice are initialized in the local minimum.

The second class of Hamiltonians are nearest-neighbour square spin glasses (Fig. 1b). Couplings are periodic (that is, the model is defined on a torus) and drawn from a normal distribution with standard deviation 1.0. All biases are zero. Hamiltonian instances are generated as needed during training. To evaluate our method and compare against classic SA, we must have a testing set of instances for which we know the true ground state. For each lattice size investigated ($\sqrt{L} = [4, 6, 8, 10, 12, 14, 16]$) we generate $N_{\text{test}} = 100$ unique instances and obtain the true ground-state energy for each instance using the branch-and-cut method[57] through the Spin Glass Server[58].

Results

WSC model. We demonstrate the use of RL on the WSC model shown in Fig. 1a. RL is able to learn a simple temperature schedule that anneals the WSC model to the ground state in 100% of episodes, regardless of the temperature in which the system is initialized. In Fig. 3b, we compare the RL policy to classic SA schedules with several constant cooling rates.

When the system is initially hot (small β), both RL and classic SA are capable of reaching the ground state with 100% success as there exists sufficient thermal energy to escape the local minimum. In Fig. 3c, we plot an example schedule. The RL policy (red) increases the temperature slightly at first, but then begins to cool the system almost immediately. An abrupt decrease in the Metropolis acceptance rate is observed (Fig. 3e). The blue dashed line in Fig. 3c represents the average schedule of the RL policy over 100 independent anneals. The standard deviation is shaded. It is apparent that the schedule is quite consistent between runs at a given starting temperature, with some slight variation in the rate of cooling.

When the system is initially cold (large β), there exists insufficient thermal energy to overcome the energy barrier between the local and global minima, and SA fails with a constant cooling rate. The RL policy, however, is able to identify, through observation, that the temperature is too low and can rapidly decrease β initially, heating the system to provide sufficient thermal energy to avoid the local minimum. In Fig. 3f, we see an increase in the Metropolis acceptance ratio, followed by a decrease, qualitatively consistent with the human-devised heuristic schedules that have been traditionally suggested[1,10,11]. In Fig. 3d, we plot an example schedule. Initially, the RL algorithm increases the temperature to provide thermal energy to escape the minimum, then begins the process of cooling. Similar to Fig. 3c, the broadness of the variance of the policies is greatest in the cooling phase, with some instances being cooled more rapidly than others. The RL agent does not have access to the current temperature directly, and bases its policy solely on the spins. The orthogonal unit-width convolutions provide a mechanism for statistics over spins and replicas, and the LSTM module provides a mechanism to capture the time-dependent dynamics of the system.

Spin glass model. We now investigate the performance of the RL algorithm in learning a general policy for an entire class of Hamiltonians, investigating whether the RL algorithm can learn to generalize its learning to accommodate a theoretically infinite set of Hamiltonians of a specific class. Furthermore, we investigate how RL performs with various lattice sizes, and compare the trained RL model to a linear (with respect to β) classic SA schedule such as the ones used in refs. [12,13]. The results of this investigation are shown in Fig. 4.

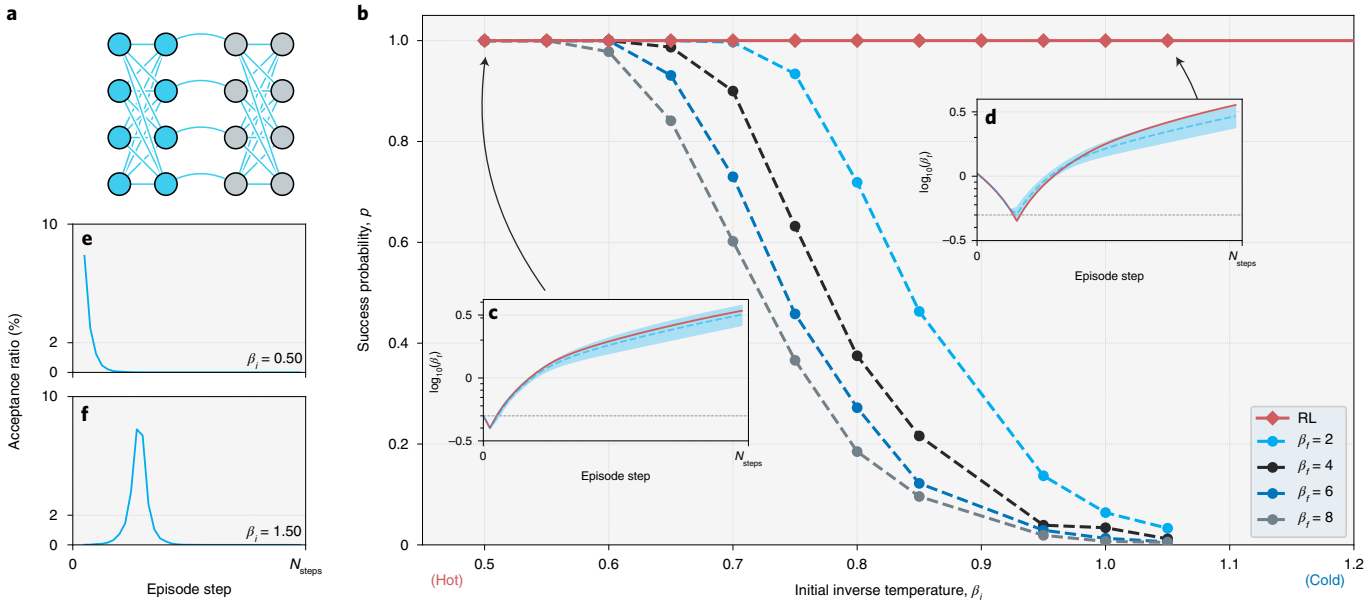


Fig. 3 | An RL policy learns to anneal the WSC model. a, The WSC model. **b**, Plot of the performance of various classic SA schedules, cooling linearly from β_i to β_f, as well as the performance of the RL policy for a variety of starting temperatures. When the initial inverse temperature is sufficiently small, both the RL and classic SA algorithms achieve 100% success (that is, every episode reaches the ground state). When the system is initialized with a large β_i, there is insufficient thermal energy for classic SA to overcome the energy barrier and reach the ground state, and consequently a very low success probability. A single RL policy achieves almost perfect success across all initial temperatures. **c**,**d**, Plots of the RL inverse temperature schedule (in red) for episodes initialized with low (**c**) and high (**d**) inverse temperatures. The average RL policy is shown in blue for the specific starting temperature. The RL algorithm can identify a cold initialization from observation and increases the temperature before then decreasing it (as shown in **d**). Shaded areas show the standard deviation. **e**,**f**, Plots of the Metropolis acceptance ratio for two episodes, initialized at two extreme temperatures: low β_i (**e**) and high β_i (**f**). In this work, we use $N_{steps} = 40$ episode steps.

In all cases, the RL schedule obtains a better (lower) n_{99} value, meaning far fewer episodes are required for us to be confident that the ground state has been observed. Furthermore, the n_{99} value exhibits much better scaling with respect to the system size (that is, the number of optimization variables). In Fig. 4e–k, we plot some of the schedules that the RL algorithm produces. In many cases, we see initial heating, followed by cooling; although, in the case of the larger models (Fig. 4i–k) we see much more complex, but still successful, behaviour. In all cases, the variance of the policies with respect to time (shown as the shaded regions in Fig. 4e–k) indicates that the agent is using information from the provided state to make decisions, and not just basing its decisions on the elapsed time using the internal state of the LSTM module. If schedules were based purely on some internal representation of time, there would be no variance between episodes.

Comparing easy and difficult instances. The learned strategy of the RL agent is relatively simple in concept: increase the temperature to a sufficiently high value and then use the remaining time to cool the system as seen in the average policies in Fig. 4e–k. In this section, we demonstrate the degree to which the performance improvement can be attributed to the ability of the RL agent to base its decisions on the various dynamics in the system.

We divide the instances in the 10 × 10 test set into two subsets, which we label 'easy' and 'difficult' based on the success of the classic SA baseline. This results in 14 difficult instances in which classic SA succeeds in only 3% of anneals, and 86 easy instances in which classic SA succeeds in more than 3% of anneals.

We compare three temperature scheduling methods on 100 episodes of each instance in both subsets: (1) classic (linear) SA; (2) the RL agent; (3) an RL agent (not yet discussed) that does not include a recurrent LSTM module. As shown in Fig. 5a, linearly scheduled classic SA solves the easy instances in 19% of anneals, whereas the RL agent manages to solve the same instances with a 53% success probability. With the difficult instances, the difference is more extreme; classic SA manages only 1% success, whereas RL performs substantially better with 29% success.

A variant of the agent without an LSTM module performs more poorly, but still better than classic SA. This agent is simply provided with a floating point representation of the episode step concatenated to the state vector, but without a recurrent network, it has no mechanism to capture the time dependence (history) of the problem. It therefore can only use the current observation in making decisions, and evidently does so more poorly than the agent with access to an LSTM module. For our formulation of the environment, an LSTM module is theoretically important to achieve a well-defined Markov decision process.

In Fig. 5b we plot the average action taken and in Fig. 5c we plot the average inverse temperature of the system at each step in the test episodes driven by the RL agent, averaged over the easy and difficult instances separately. There is no notable difference in the average schedules of the two subsets. This fact, combined with the considerable magnitude of the standard deviation (plotted as a shaded region for difficult instances and vertical bars for easy ones) suggests that the RL agent is adaptive to the specific instantiation of each Hamiltonian. Some of these dynamics can be seen in the successful schedules randomly selected for plotting in Fig. 5f.

We then take the average schedules plotted in Fig. 5b,c and use them as if they were RL-designed general heuristic schedules, removing the necessity to conduct observations during the evaluation procedure. Both the difficult and easy average schedules perform very poorly on both the difficult and easy subsets, succeeding in fewer than 10% of episodes. This is strong evidence of the specificity of the RL agent's actions to the particular dynamics of each

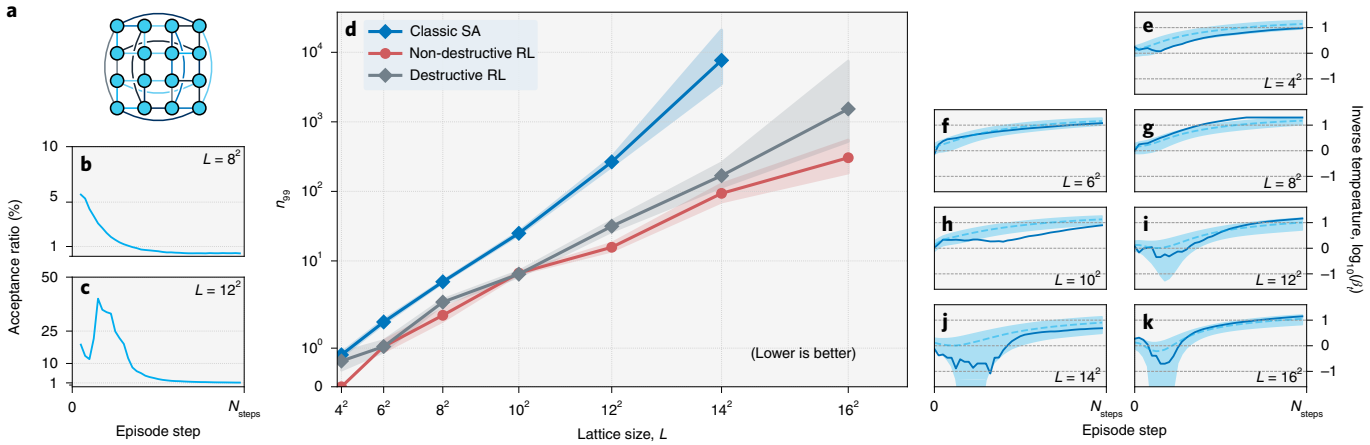


Fig. 4 | An RL policy learns to anneal spin glass models. a, An example ($L = 4^2$) lattice. **b**,**c**, Plots of the acceptance ratios over time for the $L = 8^2$ (**b**) and $L = 12^2$ (**c**) lattices. **d**, Comparison of the scaling of the RL policy with respect to system size and comparison to classic SA. We plot the n_{99} value (the number of anneals required to be 99% certain of observing the ground state; in the case of 100% success, n_{99} is undefined and plotted as zero) as a function of system size for both the RL and the best linear simulated annealing schedule we observed. The 95% confidence interval is shown as a shaded region. For all system sizes investigated, the learned RL policy is able to reach the ground state in significantly fewer runs. The destructive observation results are also plotted; these also outperform the linear schedules. We note that the destructive observation requires far more Monte Carlo steps per episode to simulate the destructive measurements; this plot should not be interpreted as a comparison of run time with regard to the destructive observation result. **e**–**k**, Example inverse temperature schedules (solid lines) and average inverse temperature schedules (for all testing episodes) (dashed lines) for lattice sizes of $L = 4^2$ (**e**), 6^2 (**f**), 8^2 (**g**), 10^2 (**h**), 12^2 (**i**), 14^2 (**j**) and 16^2 (**k**). The shaded regions denote the standard deviation. In this work, we use $N_{steps} = 40$ episode steps.

episode and refutes the hypothesis that a single, average policy, even if trained by RL, is a good case for generic instances.

We repeated the previous analysis with subsets based on the performance of the RL agent, arriving at identical conclusions (Fig. 5e).

Destructive observation. A key element of the nature of quantum systems is the collapse of the wavefunction when a measurement of the quantum state is made. When dealing with quantum systems, one must make control decisions based on quantum states that have evolved through an identical policy but have never before been measured. We model this restriction on quantum measurements by allowing any replica observed in the anneal to be consumed as training data for the RL algorithm only once. We simulate this behaviour by keeping track of the policy decisions (the changes in inverse temperature) in an action buffer as we play through each episode. When a set of N_{reps} replicas are measured, they are consumed and the system is reset to a new set of initial conditions, as if it was a new episode. The actions held in the buffer are replayed on the new replicas.

In this situation, the agent cannot base its decision on any replica-specific temporal correlations between given measurements; this should not be a problem early in each episode, as the correlation timescale of a hot system is very short, and the system, even under non-destructive observation, would have evolved sufficiently in the time window between steps to be uncorrelated. However, as the system cools, the correlation timescale increases exponentially, and destructive observation prevents the agent from relying on temporal correlations of any given replica.

We evaluate an agent trained in this 'quantum-inspired' way and plot its performance alongside the non-destructive (that is, classical) case in Fig. 4d. In the case of destructive observation, the agent performs marginally less well than the non-destructive case, but still performs better than SA in all cases. As it is a more complicated task to make observations when the system is temporally uncorrelated, it is understandable that the performance would be inferior to the non-destructive case. Nonetheless, RL is capable of outperforming SA in both the destructive and non-destructive cases.

The relative performance in terms of computational demand between destructive observation and SA alludes to an important future direction in the field of RL, especially when applied to physical systems where observation is destructive, costly and altogether difficult. With destructive observations, N_{steps} systems must be initialized and then evolved together under the same policy. Each copy is consumed one by one, as observations are required for decision making, thus incurring an unavoidable $N_{steps}^2/2$ penalty in the destructive case. In this sense, it is difficult to consider RL to be superior; prescheduled SA simply does not require observation. However, if the choice to observe were to be incorporated into the action set of the RL algorithm, the agent would choose when observation would be necessary.

For example, in the systems presented in this work, the correlation time of the observations is initially small; the temperatures are high and frequent observations are required to guide the system through phase space. As the system cools, however, the correlation time grows exponentially, and the observations become much more similar to each previous observation; in this case, it would be beneficial to forgo some expensive observations, as the system would not be evolving substantially. With such a scheme, RL stands a better chance at achieving greater performance.

Policy analysis. To glean some understanding into what the RL agent is learning, we train an additional model on a well-understood Hamiltonian, the ferromagnetic Ising model of size 16×16. In this case, the temperatures are initialized randomly (as in the WSC model). This model is the extreme case of a spin glass, with all $J_{ij} = 1$. In Fig. 6a, we display the density of states $g(M,E)$ of the Ising model, plotted in phase space, with axes of magnetization per spin (M/L) and energy per spin (E/L). The density of states is greatest in the high-entropy $M = E = 0$ region and lowest in the low-entropy 'corners'. We show the spin configurations at the three corners ('chequerboard', all spin-up and all spin-down) for clarity. The density of states is obtained numerically using Wang–Landau sampling[59]. Magnetization and energy combinations outside of the dashed 'triangle' are impossible.

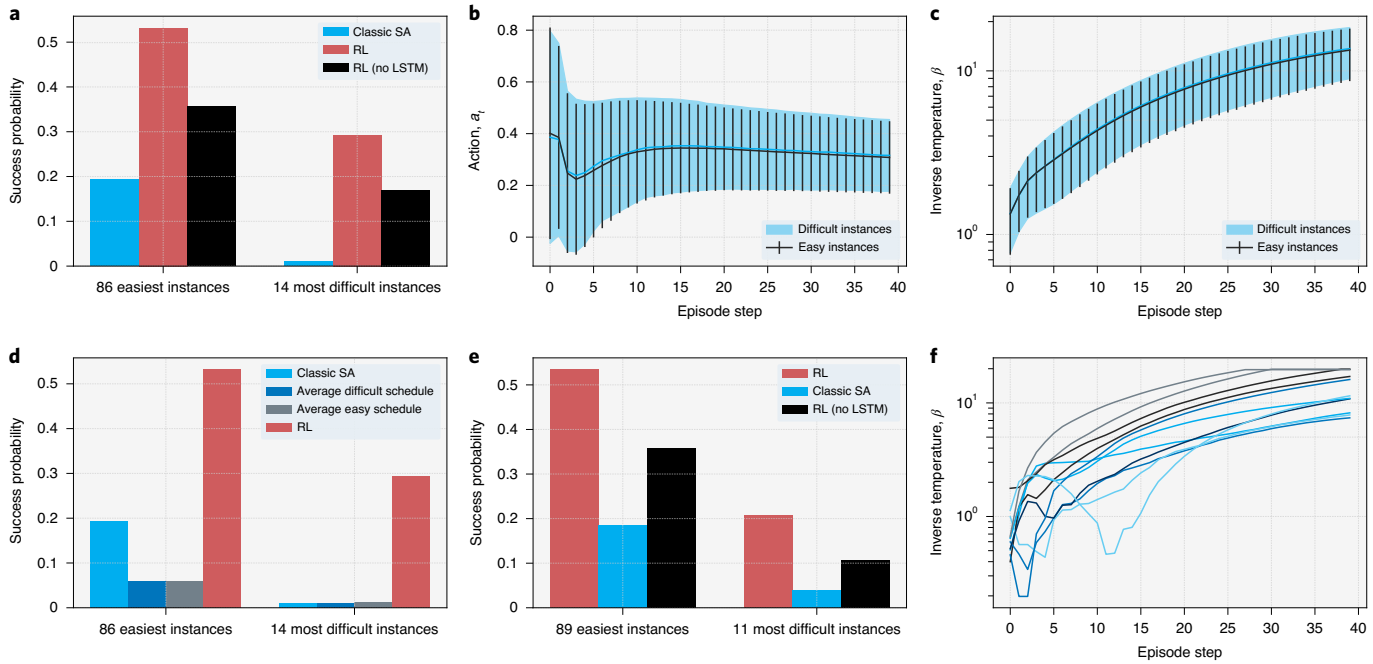


Fig. 5 | We separate the 10 × 10 spin glass instances in the test set into two subsets (easy and difficult), depending on the success of classic SA in finding their ground states. a, Plot of the performances of three different temperature scheduling approaches on the easy and difficult subsets. RL exhibits superior performance over classic SA in both subsets; however, it demonstrates dramatic superiority in the case of the difficult instances. RL without an LSTM module still performs better than classic SA; it can still dynamically modify the schedule and is not constrained to a constant temperature change at each step, so is more akin to a traditional heuristic temperature scheduling approach. **b**,**c**, Plots of the average RL actions (**b**) and schedule (**c**), respectively, for both the difficult and easy instance subsets. The standard deviations of the policies are plotted with error bars (easy instances) and shaded regions (difficult instances). The average difficult policy is very similar to the average easy policy, both having a large standard deviation, suggesting a high degree of specificity of the policy to each individual episode (see **f**). **d**, The performance when we apply the average actions presented in **b** as a static policy. The average policies perform even more poorly than classic SA. This is further evidence that the RL agent's ability to observe the system is crucial to its high performance. One might object to the method used to split the instances into the difficult and easy subsets; we have explicitly chosen to split the subsets at a boundary that makes classic SA perform poorly on the difficult instances. **e**, We thus consider a difficult (easy) instance as one that the RL agent performs poorly (well) on, and the story remains unchanged. **f**, Plots of several successful schedules; each schedule is quite different from the others, but each results in a successful episode.

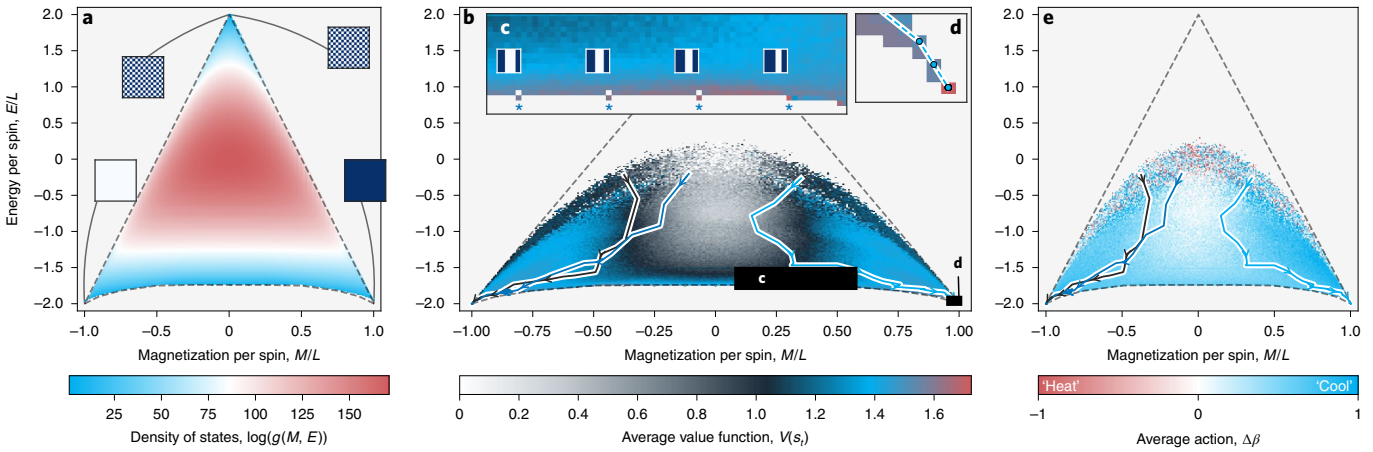


Fig. 6 | We train an agent on a special case of the spin glass Hamiltonians: the 16 × 16 ferromagnetic Ising model where all couplings $J_{ij} = 1$. a, We plot the density of states log (g(M, E)) for the 16 × 16 Ising model in the phase space of energy and magnetization, sampled numerically using the Wang–Landau algorithm[59], and indicate the location of four of the novel high- and low-energy spin configurations: two 'chequerboard' configurations, all spin-up and all spin-down. **b**, For the trained model, we plot the average of the learned value function $V(s_t)$ for each possible energy–magnetization pair. Additionally, we plot the trajectories of the first replica for three episodes of annealing to demonstrate the path through phase space the algorithm learns to take. **c**,**d**, Enlarged views of two high-value regions of interest. **e**, Plot of the average action taken at each point in phase space, as well as the same trajectories plotted in **b**.

In Fig. 6b, we plot a histogram of the average value function $V(s_t)$ on the phase plane, as well as three trajectories. Note that because each observation s_t is composed of N_{reps} replicas, we count each observation as N_{reps} separate points on the phase plot when computing the histogram, each with an identical contribution of $V(s_t)$ to the average. As expected, the learned value function trends higher toward the two global energy minima. The lowest values are present in the initialization region (the high-energy band along the top). We expand two regions of interest in Fig. 6c,d. In Fig. 6d, we can see that the global minimum is assigned the highest value; this is justifiable in that if the agent reaches this point, it is likely to remain here and reap a high reward so long as the agent keeps the temperature low for the remainder of the episode.

In Fig. 6c, we identify four noteworthy energy–magnetization combinations, using asterisks. These four energy–magnetization combinations have identical energies, with increasing magnetization, and correspond to banded spin structures of decreasing width (four example spin configurations are shown). The agent learns to assign a higher value to the higher-magnetization structures, even though the energy, which is the true measure of 'success', is identical. This is because the higher-magnetization bands are closer to the rightmost global minimum in action space; that is, the agent can traverse from the small-band configuration to the ground state in fewer spin flips than if traversing from the wide-band configurations.

In Fig. 6e, we plot a histogram of the average action taken at each point in phase space. The upper high-energy band exhibits more randomness in the actions chosen, as this is the region in which the system lands upon initialization. When initialized, the temperature is at a randomly drawn value, and sometimes the agent must first heat the system to escape a local minimum before then cooling, and thus the first action is, on average, of very low magnitude. As the agent progresses toward the minimum, the agent becomes more aggressive in cooling the system, thereby thermally trapping itself in lower energy states.

Scaling and time to solution. Figure 4d indicates that both non-destructive and destructive RL perform substantially better, not only in absolute terms, but also in terms of scaling. It is important to note that we have specifically chosen a neural network architecture (convolutional) that scales linearly with system size, and have trained each model for the same number of episodes, each consisting of the same number of sweeps. The computation time for each sweep scales linearly with the system size, and thus the training time of our RL models scales linearly with system size. Using RL does indeed impose an additional inference cost, as the observation must be processed by the neural network; on the $L = 10^2$ system, inference takes one-third the amount of time as does each episode step. However, this cost has not been optimized and could be lowered substantially through optimization of the neural network inference or even by offloading the policy network onto specialized hardware designed for inference.

Conclusion

In this work, we show that reinforcement learning is a viable method for learning dynamic control schemes for the task of SA. We show that, on a simple spin model, the RL agent is capable of devising a temperature control scheme that can consistently escape a local minimum and then anneal to the ground state. It arrives at a policy that generalizes to a range of initialization temperatures; in all cases, it learns to cool the system. However, if the initial temperature is too low, the RL agent learns to first increase the temperature to provide sufficient thermal energy to escape the local minimum. It achieves this without being provided explicit knowledge of the temperature.

We then demonstrate that the RL agent is capable of learning a policy that can generalize to an entire class of Hamiltonians and that the problem need not be restricted to a single set of couplings. By training multiple RL agents on increasing numbers of variables (increasing lattice sizes), we investigate the scaling of the RL algorithm and find that it outperforms a classic SA schedule both in absolute terms and in terms of its scaling.

Our technique is not limited to the system sizes we present in this work; larger system sizes are also within its reach. At some point, as the size of the system increases, correlation times in the underlying Metropolis–Hastings simulation become larger than the intervals between observations, and the number of sweeps must be increased. Additionally, we have specifically chosen a neural network architecture that scales linearly with system size (convolutional neural networks) as opposed to traditional multilayer perceptron networks that scale exponentially. In fact, the entire procedure scales at most polynomially with system size.

We analyse the value function that the agent learns and see that it attributes an intuitive representation of value to specific regions of phase space.

We discuss the nature of RL in the physical sciences, specifically in situations where observing systems is destructive ('destructive observation') or costly (for example, performing quantum computations where observations collapse the wavefunction or conducting chemical analysis techniques that destroy a sample material). We demonstrate that our implementation of RL is capable of performing well in a destructive observation situation, albeit inefficiently. We propose that the future of physical RL (that is, RL in the physical sciences) will be one of 'controlled observation', where the algorithm can choose when an observation is necessary, minimizing the inherent costs incurred when observations are expensive, slow or difficult.

Data availability

The test datasets necessary to reproduce these findings are available at https://doi.org/10.5281/zenodo.3897413.

Code availability

The code necessary to reproduce these findings is available at https://doi.org/10.5281/zenodo.3897413.

Received: 4 March 2020; Accepted: 5 August 2020;
Published online: 7 September 2020

References

1. Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P. Optimization by simulated annealing. *Science* **220**, 671–680 (1983).
2. Barahona, F. On the computational complexity of Ising spin glass models. *J. Phys. A* **15**, 3241–3253 (1982).
3. Sherrington, D. & Kirkpatrick, S. Solvable model of a spin-glass. *Phys. Rev. Lett.* **35**, 1792–1796 (1975).
4. Ising, E. Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift Phys.* **31**, 253–258 (1925).
5. Onsager, L. Crystal statistics. I. A two-dimensional model with an order–disorder transition. *Phys. Rev.* **65**, 117–149 (1944).
6. Ferdinand, A. E. & Fisher, M. E. Bounded and inhomogeneous Ising models. I. Specific-heat anomaly of a finite lattice. *Phys. Rev.* **185**, 832–846 (1969).
7. Lucas, A. Ising formulations of many NP problems. *Front. Phys.* **2**, 1–14 (2014).
8. Hastings, B. Y. W. K. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* **57**, 97–109 (1970).
9. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. & Teller, E. Equation of state calculations by fast computing machines. *J. Chem. Phys.* **21**, 1087–1092 (1953).
10. Kirkpatrick, S. Optimization by simulated annealing: quantitative studies. *J. Stat. Phys.* **34**, 975–986 (1984).
11. van Laarhoven, P. J. M. & Aarts, E. H. L. Simulated Annealing: Theory and Applications (Springer, 1987).
12. Stander, J. & Silverman, B. W. Temperature schedules for simulated annealing. *Stat. Comput.* **4**, 21–32 (1994).
13. Heim, B., Rønnow, T. F., Isakov, S. V. & Troyer, M. Quantum versus classical annealing of Ising spin glasses. *Science* **348**, 215–217 (2015).

14. Bounds, D. G. New optimization methods from physics and biology. *Nature* **329**, 215–219 (1987).
15. Farhi, E. et al. A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. *Science* **292**, 472–475 (2001).
16. Hen, I. & Young, A. P. Solving the graph-isomorphism problem with a quantum annealer. *Phys. Rev. A* **86**, 042310 (2012).
17. Boixo, S. et al. Evidence for quantum annealing with more than one hundred qubits. *Nat. Phys.* **10**, 218–224 (2014).
18. Bian, Z. et al. Discrete optimization using quantum annealing on sparse Ising models. *Front. Phys.* **2**, 1–10 (2014).
19. Venturelli, D., Marchand, D. J. J. & Rojo, G. Quantum annealing implementation of job-shop scheduling. Preprint at https://arxiv.org/pdf/1506.08479.pdf (2015).
20. Ray, P., Chakrabarti, B. K. & Chakrabarti, A. Sherrington–Kirkpatrick model in a transverse field: absence of replica symmetry breaking due to quantum fluctuations. *Phys. Rev. B* **39**, 11828–11832 (1989).
21. Martoňák, R., Santoro, G. E. & Tosatti, E. Quantum annealing by the path-integral Monte Carlo method: the two-dimensional random Ising model. *Phys. Rev. B* **66**, 094203 (2002).
22. Santoro, G. E. Theory of quantum annealing of an Ising spin glass. *Science* **295**, 2427–2430 (2002).
23. Finnila, A., Gomez, M., Sebenik, C., Stenson, C. & Doll, J. Quantum annealing: a new method for minimizing multidimensional functions. *Chem. Phys. Lett.* **219**, 343–348 (1994).
24. Kadowaki, T. & Nishimori, H. Quantum annealing in the transverse Ising model. *Phys. Rev. E* **58**, 5355–5363 (1998).
25. Harris, R. et al. Experimental demonstration of a robust and scalable flux qubit. *Phys. Rev. B* **81**, 134510 (2010).
26. Harris, R. et al. Experimental investigation of an eight-qubit unit cell in a superconducting optimization processor. *Phys. Rev. B.* **82**, 024511 (2010).
27. Johnson, M. W. et al. Quantum annealing with manufactured spins. *Nature* **473**, 194–198 (2011).
28. McGeoch, C. C. & Wang, C. Experimental evaluation of an adiabiatic quantum system for combinatorial optimization. In *Proceedings of the ACM International Conference on Computing Frontiers*, *CF '13*, Vol. 23, 1–11 (ACM, 2013).
29. Ikeda, K., Nakamura, Y. & Humble, T. S. Application of quantum annealing to nurse scheduling problem. *Sci. Rep.* **9**, 12837 (2019).
30. Dickson, N. G. et al. Thermally assisted quantum annealing of a 16-qubit problem. *Nat. Commun.* **4**, 1903 (2013).
31. Okada, S., Ohzeki, M. & Tanaka, K. Efficient quantum and simulated annealing of Potts models using a half-hot constraint. *J. Phys. Soc. Jpn* **89**, 094801 (2020).
32. Battaglia, D. A., Santoro, G. E. & Tosatti, E. Optimization by quantum annealing: lessons from hard satisfiability problems. *Phys. Rev. E* **71**, 066707 (2005).
33. Tsukamoto, S., Takatsu, M., Matsubara, S. & Tamura, H. An accelerator architecture for combinatorial optimization problems. *Fujitsu Sci. Technical J.* **53**, 8–13 (2017).
34. Inagaki, T. et al. A coherent Ising machine for 2,000-node optimization problems. *Science* **354**, 603–606 (2016).
35. Leleu, T., Yamamoto, Y., McMahon, P. L. & Aihara, K. Destabilization of local minima in analog spin systems by correction of amplitude heterogeneity. *Phys. Rev. Lett.* **122**, 040607 (2019).
36. Tiunov, E. S., Ulanov, A. E. & Lvovsky, A. I. Annealing by simulating the coherent Ising machine. *Opt. Express* **27**, 10288–10295 (2019).
37. Farhi, E., Goldstone, J. & Gutmann, S. A quantum approximate optimization algorithm. Preprint at https://arxiv.org/pdf/1411.4028.pdf (2014).
38. Farhi, E., Goldstone, J., Gutmann, S. & Zhou, L. The quantum approximate optimization algorithm and the Sherrington–Kirkpatrick model at infinite size. Preprint at https://arxiv.org/pdf/1910.08187.pdf (2019).
39. Sutton, R. & Barto, A. *Reinforcement Learning: An Introduction* 2nd edn (MIT Press, 2018).
40. Berner, C. et al. Dota 2 with large scale deep reinforcement learning. Preprint at https://arxiv.org/pdf/1912.06680.pdf (2019).
41. Zhang, Z. et al. Hierarchical reinforcement learning for multi-agent MOBA Game. Preprint at https://arxiv.org/pdf/1901.08004.pdf (2019).
42. Vinyals, O. et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **575**, 350–354 (2019).
43. Mnih, V. et al. Playing Atari with deep reinforcement learning. Preprint at https://arxiv.org/pdf/1312.5602.pdf (2013).
44. Silver, D. et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).
45. Silver, D. et al. Mastering the game of Go without human knowledge. *Nature* **550**, 354–359 (2017).
46. Silver, D. et al. A general reinforcement learning algorithm that masters chess, shogi and Go through self-play. *Science* **362**, 1140–1144 (2018).
47. Agostinelli, F., McAleer, S., Shmakov, A. & Baldi, P. Solving the Rubik's cube with deep reinforcement learning and search. *Nat. Mach. Intell.* **1**, 356–363 (2019).
48. Akkaya, I. et al. Solving Rubik's cube with a robot hand. Preprint at https://arxiv.org/pdf/1910.07113.pdf (2019).
49. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal policy optimization algorithms. Preprint at https://arxiv.org/pdf/1707.06347.pdf (2017).
50. Hill, A. et al. Stable Baselines (2018); https://github.com/hill-a/stable-baselines
51. Brockman, G. et al. OpenAI Gym. Preprint at https://arxiv.org/pdf/1606.01540.pdf (2016).
52. Schulman, J., Levine, S., Abbeel, P., Jordan, M. & Moritz, P. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning* 1889–1897 (ICML, 2015).
53. Kakade, S. & Langford, J. Approximately optimal approximate reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning* 267–274 (ICML, 2002).
54. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Comput.* **9**, 1735–1780 (1997).
55. Aramon, M. et al. Physics-inspired optimization for quadratic unconstrained problems using a Digital Annealer. *Front. Phys.* 7 (2019); https://doi.org/10.3389/fphy.2019.00048
56. Bunyk, P. I. et al. Architectural considerations in the design of a superconducting quantum annealing processor. *IEEE Trans. Appl. Superconductivity* **24**, 1–10 (2014).
57. Liers, F., Jünger, M., Reinelt, G. & Rinaldi, G. in *New Optimization Algorithms in Physics* Vol. 4, 47–69 (Wiley, 2005).
58. Jünger, M. Spin glass server; https://informatik.uni-koeln.de/spinglass/
59. Wang, F. & Landau, D. P. Efficient, multiple-range random walk algorithm to calculate the density of states. *Phys.Rev. Lett.* **86**, 2050–2053 (2001).

Acknowledgements

I.T. acknowledges support from NSERC. K.M. acknowledges support from Mitacs. We thank B. Krayenhoff for valuable discussions in the early stages of the project and thank M. Bucyk for reviewing and editing the manuscript.

Author contributions

All authors contributed to the ideation and design of the research. K.M. developed and ran the computational experiments and wrote the initial draft of the the manuscript. P.R. and I.T. jointly supervised this work and revised the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to K.M., P.R. or I.T.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Chapter 6

Conclusion

In the physical sciences, artificial intelligence techniques such as deep learning, convolutional neural networks, and reinforcement learning have begun to receive adoption in recent years. Particularly after the great success of convolutional neural networks on image classification challenges [101, 141], the possibilities of these techniques became apparent, and adoption into the physical sciences began. There are important concerns in the physical sciences that are not present in computer vision, and it is the job of the scientist wishing to use machine learning techniques to adapt them to the physical systems on which they are being applied.

This dissertation presented a collection of such adaptations of existing techniques to various problems in the physical sciences. Most of these adaptations occurred in the early phase of adoption of artificial intelligence (specifically deep learning and its derivatives) into the physical sciences.

We showed that a deep neural network could be used to solve the one-electron Schrödinger equation [142], proving the viability of deep, featureless learning for physical systems. We demonstrated, using the Ising model (a very

well-understood physical system) how sampling is of great importance in training a neural network, and verified that neural networks could learn physical operators to sufficient accuracy to reproduce physical simulations [1]. We developed a method through which we could represent atomic systems of arbitrary atom placement in a way amenable to deep neural networks [143]. These models were all limited in one major way: the information that the neural networks learned could only be applied to structures of identical size, and transferring this knowledge to larger systems was not possible. We identified the need for neural networks that are extensive, that is, neural networks that can scale to arbitrarily large systems without suffering performance loss or poor scaling. By designing a new neural network architecture, we were able to develop a technique that could make predictions on arbitrarily large systems after observing only a small subset of observations [144]. These extensive deep neural networks have seen use in materials discovery for exploring large chemical spaces [145], and in object counting tasks (Section 3.5).

Generative machine learning is an area of artificial intelligence that holds great promise in the physical sciences, and we investigated the use of a Generative Adversarial Network on the Ising model [6], showing that a GAN can be trained to both generate new, unseen examples of specific energy distributions, as well as make predictions *with a confidence score*, as to their properties. Generative Adversarial Network techniques improved over time and we later

applied new techniques to both atomic [8] and experimental [7] systems. This new-and-improved GAN can perform property-targeted generation of training examples of arbitrary size.

6.0.1 Outlook

As technologies progressed and continue to progress, many of the techniques used in this dissertation have not remained state-of-the-art. Those that have will be superseded by newer technologies that will surpass what is currently available. It is not too soon, however, to begin using these technologies in practical applications and not allow these proof-of-concept approaches to remain as only concepts.

An important focus in the physical sciences is the idea of transparency and interpretability, and new approaches are attempting to address this. With this will come a deeper understanding of the limitations of machine learning techniques, but also perhaps an approach for discovery as transparent techniques can be more easily probed. As new techniques are developed, the projects in this dissertation can serve as baseline comparisons. We used common, easy-to-understand physical examples because it is important to start small when applying a new technique. Basing future investigations on these simple examples (namely, the Ising model and its variants), will allow quick iteration and make it easy to identify oversights quickly in the process.

Furthermore, the physics community should learn from the computer science community in its propensity to share open source data sets. A considerable amount of effort must be taken in constructing a data set that is balanced, useful, and reusable. As such, providing this data accomplishes two objectives: it assists in making any new methodology transparent and reproducible, and ultimately accelerates the production and validation of future approaches. When data sets can be reused, less time can be spent on the construction of data, more time is available for the development of methods.

Along a similar vein is the idea that deep learning can enable physics and materials science to adopt the "big data" approaches of industry, sharing models and centralizing learned knowledge. In materials science, for example, computations are commonly initiated from scratch, ignoring the world of other researchers who have perhaps previously run very similar computations. Deep models have the ability to learn from a tremendous amount of data, and generalize to different data.

Ideas from physics will be used to motivate advances in the machine learning community. The motivation for extensive deep neural networks was the physical concepts of length scale and extensivity, but these ideas are useful for normal computer vision counting tasks outside of physics. We are likely to see more learning algorithms that draw inspiration from the physical sciences in the future.

When we demonstrated the use of reinforcement learning on a physically-motivated process: simulated annealing [9], we proposed that reinforcement learning will likely play a major role in the future of research within the physical sciences. As such, an important area of future research will be to make reinforcement learning algorithms that can perform "controlled observation" of their surroundings. Observations of physical systems and experiments are often expensive, or outright impossible without destroying the system, and algorithms that can choose conservatively when an observation is necessary will be important for the adoption of these techniques. We showed that, at least for our system, algorithms could cope with this destructive observation. Furthermore, especially for experimental physical sciences, the sample efficiency of reinforcement learning techniques must be improved, as performing millions of physical experiments is not likely possible.

Machine learning, and deep learning techniques are constantly evolving, and new techniques are being proposed regularly. It is unclear how the field will look in the future, but one thing is clear: the introduction of deep learning to the physical sciences is still in its infancy, and there is an exciting era of machine learning-driven physics ahead.

Appendix A

RUGAN manuscript

The RUGAN manuscript is included in entirety below. Important for this dissertation is the Supplementary Information, which develops the motivation for RUGAN and demonstrates the method on the Ising model.

Optical lattice experiments at unobserved conditions and scales through generative adversarial deep learning

Corneel Casert,[1,*] Kyle Mills,[2,3] Tom Vieijra,[1] Jan Ryckebusch,[1] and Isaac Tamblyn[2,3,4,†]

[1] *Department of Physics and Astronomy, Ghent University, 9000 Ghent, Belgium*
[2] *Vector Institute for Artificial Intelligence, Toronto, Ontario, Canada*
[3] *Ontario Tech University, Oshawa, Ontario, Canada*
[4] *National Research Council Canada, Ottawa, Ontario, Canada*

Machine learning provides a novel avenue for the study of experimental realizations of many-body systems, and has recently been proven successful in analyzing properties of experimental data of ultracold quantum gases. We here show that deep learning succeeds in the more challenging task of modelling such an experimental data distribution. Our generative model (RUGAN) is able to produce snapshots of a doped two-dimensional Fermi-Hubbard model that are indistinguishable from previously reported experimental realizations. Importantly, it is capable of accurately generating snapshots at conditions for which it did not observe any experimental data, such as at higher doping values. On top of that, our generative model extracts relevant patterns from small-scale examples and can use these to construct new configurations at a larger size that serve as a precursor to observations at scales that are currently experimentally inaccessible. The snapshots created by our model—which come at effectively no cost—are extremely useful as they can be employed to quantitatively test new theoretical developments under conditions that have not been explored experimentally, parameterize phenomenological models, or train other, more data-intensive, machine learning methods. We provide predictions for experimental observables at unobserved conditions and benchmark these against modern theoretical frameworks. The deep learning method we develop here is broadly applicable and can be used for the efficient large-scale simulation of equilibrium and nonequilibrium physical systems.

Ultracold atoms provide a controlled environment for the study of emergent phenomena in many-body physical systems—including high-temperature superconductivity, many-body localization, or topological quantum phases—as well as fields such as cosmology and quantum chemistry [1–4]. Hence, finding a unifying theoretical or numerical model able to create configurations with statistics that conform to all experimental observations is of crucial importance. Indeed, this allows one to use the model to make predictions for conditions that currently can not be experimentally realized, in addition to obtaining a better understanding of the system. Herein we propose using generative deep learning to create such a model, that learns to produce configurations indistinguishable from those experimentally obtained and can also make predictions for configurations at larger scale or at unobserved control parameter values. The capability of *discriminative* machine learning to analyze physical systems has by now been well established, both for data obtained through numerical simulations [5–11], and from experimental observations through electronic quantum matter visualization [12], quantum gas microscopy [13], or momentum-space density images [14]. In these machine learning applications, a neural network is trained to predict properties $\mathbf{y}$ of configurations $\mathbf{x}$, *i.e.* learn the conditional probability $p(\mathbf{y}|\mathbf{x})$. Examples include the characterization of phases of matter, or efficiently calculating properties of individual microstates. The use of *generative* machine learning is relatively unexplored within experimental science; it is a much more complex problem as it requires the modeling and sampling of a probability distribution $p(\mathbf{x}, \mathbf{y})$ not known *a priori*. Yet, generative learning provides a particularly attractive approach as it relies on automatic pattern recognition—and hence does not focus on the reproduction of a specific physical quantity, which can introduce bias, or require prior knowledge about the system. Recently, Boltzmann generators [15] were trained on the energy functional of many-body systems to directly generate low-energy equilibrium configurations and can overcome rare event sampling problems in simulations. A particular class of generative models, namely restricted Boltzmann machines, has seen use as efficient variational ansätze for quantum many-body wave functions [16–21]. In return, physics-inspired algorithms (tensor networks) are now being explored for discriminative and generative tasks in machine learning [22–24].

Here, we show that generative deep learning can be used to represent and sample the distribution of snapshots of an experimental realization of the Fermi-Hubbard model with ultracold atoms in an optical lattice. The format of this article is as follows: we first discuss our application area; next, we outline our generative model 'RUGAN', and finally we apply it to previously reported experimental data.

* corneel.casert@ugent.be
† isaac.tamblyn@nrc.ca

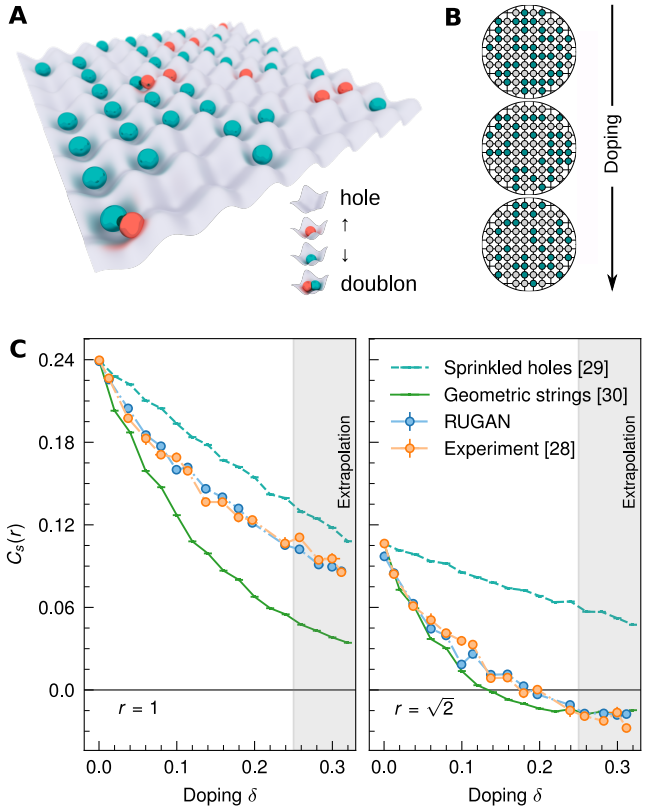


FIG. 1. The Fermi-Hubbard model with ultracold atoms in an optical lattice. **A**, Experimental realization of the Fermi-Hubbard model. The two spin types are trapped in a two-dimensional square optical lattice. The quantum state can be imaged with quantum gas microscopy. **B**, Our generative model is trained on site-resolved snapshots of quantum states, obtained at a fixed temperature and for a range of hole doping values δ. Here, only one spin species is studied, while the other spin species, holes, and doublons, are all detected as empty sites. **C**, Sign-corrected spin correlations $C_s(r)$ of Eq. (2) as a function of the hole doping δ for nearest neighbors (left, $r = 1$) and next-nearest neighbors (right, $r = \sqrt{2}$). The correlations obtained with our generative model 'RUGAN' (1000 snapshots generated for each doping) are consistent with experiment. Note that our generative model has not been optimized on data corresponding to the largest doping values $\delta \gtrsim 0.24$ (shaded area), but is still able to produce configurations with correlations that match well with experimental observations. The geometric string theory consistently underestimates the nearest-neighbour correlations $C_s(1)$ while for both distances considered, the spin correlations obtained with sprinkled holes are overestimated.

The Fermi-Hubbard model is of particular interest as it is suggested to hold the key to understanding high-temperature superconductivity [3, 25, 26]. The Fermi-Hubbard model is described by the Hamiltonian

$$\widehat{\mathcal{H}} = -t \sum_{\langle \mathbf{i},\mathbf{j}\rangle,\sigma} \left(\hat{c}^\dagger_{\mathbf{i},\sigma} \hat{c}_{\mathbf{j},\sigma} + \text{h.c.} \right) + U \sum_{\mathbf{i}} \hat{c}^\dagger_{\mathbf{i},\uparrow} \hat{c}_{\mathbf{i},\uparrow} \hat{c}^\dagger_{\mathbf{i},\downarrow} \hat{c}_{\mathbf{i},\downarrow}, \quad (1)$$

where $\hat{c}^\dagger_{\mathbf{i},\sigma}(\hat{c}_{\mathbf{i},\sigma})$ is the creation (annihilation) operator of a spin $\sigma \in \{\uparrow,\downarrow\}$ on site $\mathbf{i}$. The first term corresponds to tunneling between neighboring lattice sites $\mathbf{i}$ and $\mathbf{j}$. The second term accounts for the on-site interaction between fermions with opposite spin. Here we consider the strongly correlated regime, where $U/t \gg 1$. The Fermi-Hubbard model can be experimentally realized with ultracold atoms trapped in an optical lattice (Fig. 1A) [1–3, 27]. Quantum gas microscopy provides us with site-resolved snapshots of these quantum states, imaging either the total atom distribution or that of a single spin species. Holes and doubly-occupied sites (doublons) are detected as empty sites (Fig. 1B). Recently, the use of discriminative machine learning for classifying snapshots of ultracold atomic gases has been investigated [13, 14] and we use the same data set [28] as in Ref. [13] to train our generative model. At half filling, the Fermi-Hubbard model is theoretically relatively well understood and maps to the Heisenberg model with superexchange coupling $J = 4t^2/U$. In this case, long-range antiferromagnetic (AFM) correlations are present throughout finite-size systems for temperatures $T \ll J$. The Fermi-Hubbard model is not as well understood when straying from the half-filling regime through the addition of holes. The motion of these holes displaces strings of spins and hence hides the AFM order observed at half filling; this has recently been experimentally observed [29]. These observations can be partially explained in the framework of geometric string theory, in which strings of displaced spins are added to a background of experimental snapshots produced at half filling [30] (see Supplementary Material). The length of these strings is dependent on the ratio t/J and the strength of the AFM correlations present in the states obtained at half filling. Note that considering the motion of the holes is crucial to describe experimentally observed hidden order, as this is not correctly accounted for by randomly adding holes to a state obtained at half filling ("sprinkled holes").

The objective of this work is to develop and train a generative model capable of representing and sampling the distribution of the experimental snapshots at requested doping values. Such a model allows for efficient augmentation of the experimental data set, and can offer predictions for properties of microstates at dopings for which no experimental data is available, or at larger spatial scales, and hence can be used for quantitative testing of theoretical frameworks. Our model is validated by comparing observables obtained experimentally and with the deep learning algorithm. We first consider the AFM correlations [29, 31–33] in snapshots created by our model—discussed in detail below—by evaluating the sign-corrected spin correlation for sites at relative displacement $\mathbf{r}$ with $r = \|\mathbf{r}\|_2$ and $\|\cdot\|_p$ denoting the p-norm:

$$C_s(r) = 4(-1)^{\|\mathbf{r}\|_1} \left(\left\langle \hat{S}^z_{\mathbf{i}} \hat{S}^z_{\mathbf{i}+\mathbf{r}} \right\rangle - \left\langle \hat{S}^z_{\mathbf{i}} \right\rangle \left\langle \hat{S}^z_{\mathbf{i}+\mathbf{r}} \right\rangle \right). \quad (2)$$

More details on the calculation of these correlations are given in the Supplementary Material. The correlations

between nearest neighbours ($r = 1$) and next-nearest neighbours ($r = \sqrt{2}$) measured on snapshots created by our generative model are shown in Fig. 1C along with the theoretical predictions by both the geometric string theory [30] and sprinkled holes [29]. Note that neither of these theoretical models create samples with the correct correlations at large hole-doping values δ for both $r = 1$ and $r = \sqrt{2}$. Our generative model is able to accurately capture the correlations across all hole-doping values—even at doping values for which it is not optimized.

For this work, we developed a new generative approach called a "regressive upscaling generative adversarial network" (RUGAN). After being shown a data set of small-scale configurations—called the training set—of a physical system, the RUGAN allows for the direct generation of new microstates at any given scale and with desired properties. The RUGAN is able to generalize in two ways: *1)* it can create microstates with properties for which no training data is available, and *2)* it is able to create samples at a much larger scale (or 'upscale') than the training examples. The former is relevant for systems where obtaining configurations is numerically or experimentally feasible only for a limited set of system properties. As only small-scale samples are required for training, the latter generalization enables efficient sampling of configurations at scales inaccessible to traditional methods, either due to excessive computational cost or experimental restrictions on the imageable system size.

The RUGAN is based on generative adversarial networks (GANs) [34–36], which are the combination of two neural networks competing against each other as adversaries. One network, G, acts as a generator, taking samples $\mathbf{z}$ randomly drawn from a latent space as input and transforming these to create new samples with distribution $\mathbb{P}_g \sim G(\mathbf{z})$. The other network works as a critic, learning to discern between samples coming from the generator and the example data set with distribution $\mathbb{P}_r$. These networks depend on many free parameters that are trained simultaneously, with the generator trying to trick the critic, and the critic learning how to better tell apart the generator's propositions from the training examples by measuring the distance between $\mathbb{P}_r$ and $\mathbb{P}_g$. A successfully trained GAN converges to a state where the generator is so good at producing samples that the critic cannot tell the generated samples from the reference training set. The generator and critic of a GAN can be conditioned on additional information such as known properties of individual samples [35]. This allows one to control the region of configuration space from which the generator produces new configurations. GANs are typically used in image processing such as super-resolution [37] or cross-domain pairings (*e.g.*, pairing shoes with matching handbags) [38], but have also been applied to the Ising model [39, 40], scalar field theories [41, 42], and inverse molecular design [43].

To allow the RUGAN to create samples of arbitrary size, our generator is designed such that it consists solely of translationally equivariant operations that can be applied to latent inputs of any size. Motivated by the locality of the interactions in the models studied here, the RUGAN consists of deep residual convolutional networks. Convolutional neural networks, by construction, have a limited receptive field defined by the size of the convolutional kernels and the depth of the network, and can not account for arbitrary-range interactions. Hence, the network depth required to accurately model a physical data set gives us a proxy for the typical correlation lengths present in the individual configurations. However, long-range interactions could also be efficiently included by making use of attention layers [44]. Once optimized on the training data, such a generator can then efficiently create configurations containing a much larger number of sites. The validity of the upscaling procedure is dependent on the same physical length scales being present in both the small-size training examples and the sizes to which we scale. This also means that the failure of creating configurations at a larger scale (*i.e.*, resulting in different statistics than computationally or experimentally obtained) will point us towards the appearance of physics at a new, larger length scale — a typical example is the divergence of the correlation length at a critical point which cannot easily be captured by RUGAN. Another advantage of the fully-convolutional design of the RUGAN is that it provides an optimized starting point for the training of larger systems.

Along with technical details on the design and training of the RUGAN, we give a simple illustration and results on classical spin models in the Supplementary Material.

We now return to the use of generative deep learning to tackle the challenging task of modelling an experimentally obtained sample distribution. To this aim, we train a RUGAN on experimental snapshots of a doped two-dimensional Fermi-Hubbard model on a square lattice, and condition it on the doping ratio δ. The output of the RUGAN is hence a series of synthetic snapshots at prescribed doping values (Fig. 2A). We then apply the same analysis procedure to these as to the experimentally obtained snapshots [29]. In Fig. 1C, we demonstrate that a RUGAN can in this way produce synthetic snapshots with high AFM spin correlations $C_s(r)$ at small hole doping values δ, and that it correctly models the decay of $C_s(r)$ in the snapshots upon increasing δ. We now show that its synthetic configurations also capture the more intricate hidden order present in the experimental snapshots, quantified by the number and average length of strings of spins displaced by hole motion as a function of δ. More information on the determination of these observables can be found in the Supplementary Material. These

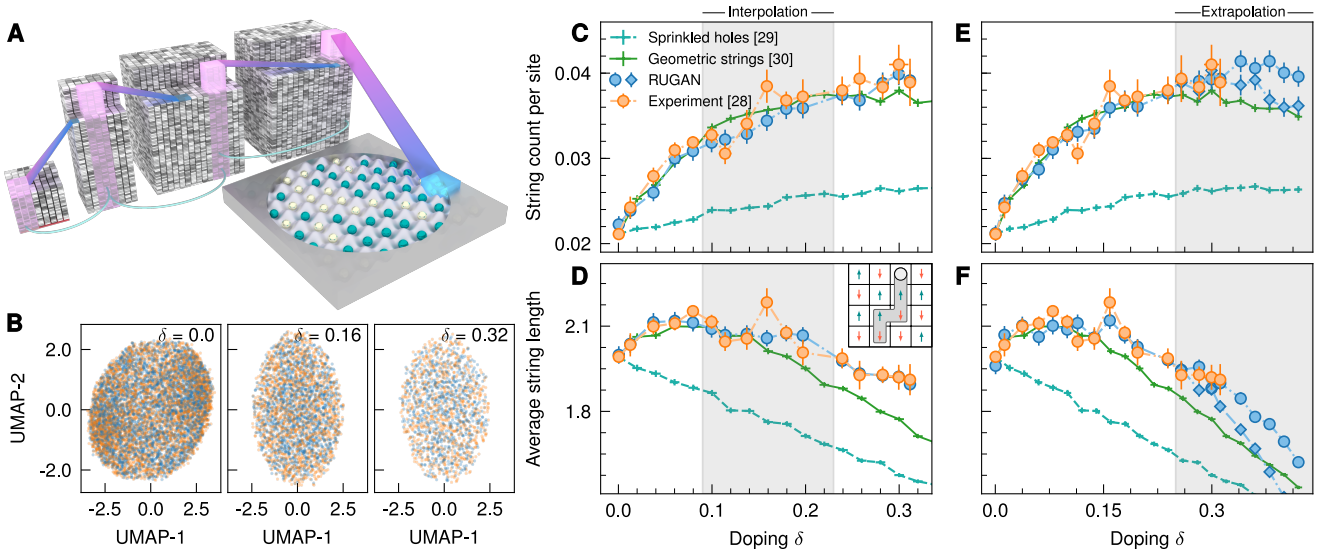


FIG. 2. String patterns in the doped Fermi-Hubbard model. **A**, Through a series of residual convolutional layers, the generative neural network transforms a latent sample to a new snapshot of a single spin species at a prescribed value of the doping δ. **B**, The distribution of snapshots created by the RUGAN (blue) cannot be distinguished from the experimental snapshots (orange) with other unsupervised machine learning methods. Here, we show a dimensionality reduction with the UMAP algorithm [45] of an equal number of synthetic and experimental snapshots. **C**, The number of strings exceeding a length of 2 per system site and **D**, the average length (sites) of the string patterns detected by the algorithm described in [29], as a function of the doping δ. The string statistics obtained with the RUGAN (with which 1000 snapshots were created for each doping value) are consistent with experimental observations. The intermediate doping values $0.09 \lesssim \delta \lesssim 0.23$ in the shaded area are not included during training of the generative model, and the RUGAN interpolates between its knowledge at low and high δ for its creation of snapshots at these dopings. Note that both theoretical frameworks match perfectly with the experimental data at half filling by construction. For high values of the doping δ, the theoretical models underestimate the average string length. (Inset) Illustration of the string pattern formation due to a hole moving through an AFM ordered state, which leaves behind a trail of displaced spins. **E** and **F**, Same as in **C** and **D**, but now the RUGAN is not provided with data corresponding to large doping values $\delta \gtrsim 0.24$ during training and is required to extrapolate to new values of δ. For the extrapolation regime, we show the statistics obtained by two independently trained RUGANs.

statistics are shown in Fig. 2 for the synthetic samples, along with the experimentally determined values, and the predictions made by the theoretical frameworks developed in Refs. [29, 30]. Remarkably, the RUGAN is able to accurately generate snapshots that exhibit the correct string statistics even at values of δ on which it is not optimized. In Fig. 2C,D we show that when the RUGAN is trained on a subset of the experimental snapshots restricted to the extrema of experimentally available doping values, it still succeeds in generating configurations at intermediate δ with string statistics matching closely with those observed in experiment. Exploiting this feature dramatically reduces the already small number of experimental observations required to train the RUGAN. Excluding the largest values of δ from the training set allows us to assess the RUGAN's ability to extrapolate its learned knowledge of configurations with smaller δ. The observation in Fig. 2E,F that the string statistics obtained with this extrapolation procedure again match closely with experimental results, showcases the RUGAN's capability to predict complex correlation patterns, and indicates that the synthetic configurations can serve as a benchmark for quantitative comparison with theoretical developments at conditions where no experimental data is available. We stress that, though still performing better than theoretical predictions for doping values not included in training, the reliability of extrapolated predictions does of course eventually decrease as predictions are made at doping values farther and farther away from those used during training. In Fig. 2E,F, we show an example of the variation in extrapolated values that occurs far from training conditions.

A current limitation in experiments with quantum gas microscopy on ultracold atoms is the limited number of sites that can be imaged, so that experimental snapshots of the optical lattices currently typically consist of less than 100 sites. The upscaling ability of the RUGAN allows us to obtain a useful precursor of what could be observed at large scale while experimental realizations containing more lattice sites are still unavailable. Given the success in performing this upscaling for classical spin models (see Supplementary Material), and the excellent agreement with experimental data for small-scale samples, these configurations can serve as a benchmark for

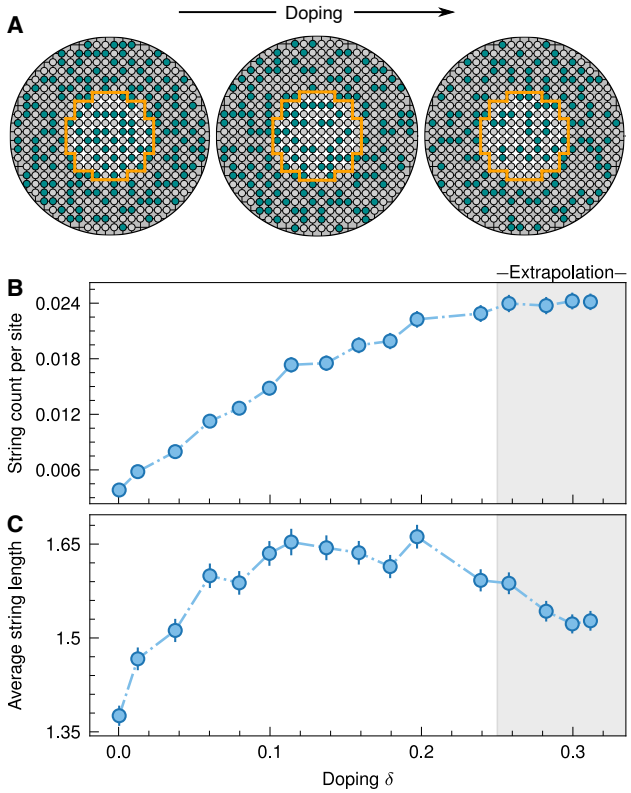


FIG. 3. Size extrapolation: generating large-scale snapshots of the Fermi-Hubbard model in an optical lattice. **A**, Example snapshots of a doped Fermi-Hubbard model created at a large scale, here consisting of approximately four times as many sites as the training examples (orange line) in Fig. 1B. **B**, The number of strings exceeding a length of 2 per system site and **C**, the average length (sites) of the strings, for large-scale snapshots created with the RUGAN. For each value of δ, we use the RUGAN of Fig. 2E,F to create 1000 large-scale snapshots.

future experimental observations of larger optical lattices. In Fig. 3A, we show such configurations obtained by our RUGAN consisting of approximately four times as many sites as the experimental examples. As RUGAN enables us to synthesize a distribution of large-scale snapshots, we can use these to predict the behavior of physical observables at scales that are experimentally inaccessible. In Fig. 3B,C we show the string count per site and average string length as a function of doping δ, measured on snapshots created by the same RUGAN as in Fig. 2E,F.

We have demonstrated the success of a RUGAN in modeling the distribution of experimental snapshots of a doped Fermi-Hubbard model. To this aim, deep neural networks are trained on quantum gas microscopy images of ultracold atoms in an optical lattice. RUGAN efficiently generates synthetic samples at requested doping values, indistinguishably distributed from the training data, and these are analyzed in the same way as one would treat experimental observations. Whereas current theoretical frameworks of this model often focus on the description of a number of specified observables, such as spin-spin correlators or hidden order, the power of generative learning lies in its unbiased learning procedure. Hence, especially at large doping values, the synthetic snapshots created by RUGAN provide a better match with experimental observations than current theoretical predictions. On top of that, the RUGAN provides the ability to sample snapshots at experimentally unobserved doping values or at larger spatial scales, and thus opens the door for quantitative testing of new phenomenological, analytical, and numerical models on synthetic data under conditions where no experimental data is available. The observation that a RUGAN is able to make highly accurate predictions across the whole doping regime provides evidence for the existence of a unifying theoretical model. Establishing such bounds is extremely useful across many physical domains including nucleation, phase transitions, and non-equilibrium growth.

SUPPLEMENTARY MATERIAL

A. RUGAN

1. Generative adversarial networks

A generative adversarial network [34] consists of a generator, which maps latent samples to new configurations, and a critic, which measures the distance between the distributions of the real samples $\mathbb{P}_r$ and those of the generated samples $\mathbb{P}_g$. The distance metric used by the critic in our work is the Wasserstein-1 or Earth-Mover distance (WGAN), which allows for stable training [36]. The Wasserstein-1 distance W between $\mathbb{P}_r$ and $\mathbb{P}_g$ is given by

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma} \left[\|\mathbf{x} - \mathbf{y}\|_2\right], \quad (3)$$

with $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ the set of joint distributions whose marginals are $\mathbb{P}_r$ and $\mathbb{P}_g$. As the infimum in Eq. (3) is intractable, the Kantorovich-Rubinstein duality is used to write

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_r} \left[f(\mathbf{x})\right] - \mathbb{E}_{\tilde{\mathbf{x}}\sim\mathbb{P}_g} \left[f(\tilde{\mathbf{x}})\right], \quad (4)$$

with the supremum taken over all 1-Lipschitz continuous functions. The training objective can now be formulated as a minimax game between two neural networks, the critic C and generator G:

$$\min_G \max_{\|C\|_L \leq 1} \mathcal{L}(C, G), \quad (5)$$

where

$$\mathcal{L}(C, G) = \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_r} \left[C(\mathbf{x})\right] - \mathbb{E}_{\tilde{\mathbf{x}}\sim\mathbb{P}_g} \left[C(\tilde{\mathbf{x}})\right]. \quad (6)$$

During training, the critic is optimized to maximize $\mathcal{L}$, and thus finds an estimate for $W(\mathbb{P}_r, \mathbb{P}_g)$. The generator then learns to minimize this distance, so that its sampled distribution $\mathbb{P}_g$ is similar to $\mathbb{P}_r$ (Fig. 4A).

The critic in the WGAN construction needs to be 1-Lipschitz continuous over the whole domain to find a correct estimate for the Wasserstein distance. While enforcing this constraint everywhere is impracticable, a good approximation can be obtained by adding two regularizing terms to the loss function of Eq. (6):

$$\begin{aligned}
\mathcal{L}' = \quad & \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_r} \left[C(\mathbf{x})\right] - \mathbb{E}_{\tilde{\mathbf{x}}\sim\mathbb{P}_g} \left[C(\tilde{\mathbf{x}})\right] && \text{(7a)} \\
& + \lambda_1 \, \mathbb{E}_{\hat{\mathbf{x}}\sim\mathbb{P}_{\hat{\mathbf{x}}}} \left[(\|\nabla_{\hat{\mathbf{x}}} C(\hat{\mathbf{x}})\|_2 - 1)^2\right] && \text{(7b)} \\
& + \lambda_2 \, \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_r} \left[\|C(\mathbf{x} + \boldsymbol{\delta}_1) - C(\mathbf{x} + \boldsymbol{\delta}_2)\|_2\right], && \text{(7c)}
\end{aligned}$$

and as new objective

$$\min_G \max_C \mathcal{L}'(C, G). \quad (8)$$

A differentiable function is 1-Lipschitz continuous if it has gradients with at most unit norm over the whole domain. Hence, the first of these regularizing terms (Eq. (7b)) penalizes the critic such that the norm of the gradient equals one for samples $\hat{\mathbf{x}}$ sampled from $\mathbb{P}_{\hat{\mathbf{x}}}$ [46]. As enforcing this over the whole support domain is intractable, the distribution $\mathbb{P}_{\hat{\mathbf{x}}}$ is sampled uniformly on straight lines between data points in the training distribution $\mathbb{P}_r$ and generated distribution $\mathbb{P}_g$.
The limitation on how $\mathbb{P}_{\hat{\mathbf{x}}}$ is sampled leaves much of the domain unconstrained. In particular, Lipschitz continuity over the manifold that supports the training distribution $\mathbb{P}_r$ is not properly enforced until the distribution $\mathbb{P}_g$ lies close to $\mathbb{P}_r$. To alleviate the lack of Lipschitz-constraint on the training manifold, a third term (Eq. (7c)) is added to the loss function that explicitly enforces Lipschitz continuity close to it [47]. Lipschitz continuity requires that for two points $\mathbf{x}'$ and $\mathbf{x}''$ close to one another, the distance $\|C(\mathbf{x}') - C(\mathbf{x}'')\|_2$ is bounded by a constant. Enforcing this criterion is accomplished by perturbing every training data point twice, with small random perturbations $(\boldsymbol{\delta}_1, \boldsymbol{\delta}_2)$, and minimizing the distance between the critic output of these configurations. In practice, the perturbation of samples is achieved by adding dropout [48], which disables nodes in a layer with a specified probability, to several layers of the critic and feeding it the same data point twice.

To condition the generator on system properties, we provide it with both a random sample from the latent space and labels describing the desired properties (*e.g.*, the energy, magnetization, or doping of configurations) as input [35]. Meanwhile, the critic is shown this same label for the generated configurations, while receiving the exact label for real samples. The critic uses this additional label during its estimation of the Wasserstein distance, prompting the generator to adapt by creating configurations that have features accurately described by their label. Note that since the critic and generator find efficient internal representations of these quantities through training, they are never explicitly evaluated during training. This implies that when we want to condition the generation on expensive operators, the vast amount of computational effort lies in generation of the small-scale samples. Creation of new, large configurations only requires a single pass through a convolutional neural network, and hence comes at a much smaller cost. Additionally, making generative models interpretable, *i.e.* understanding how it models the interactions between the degrees of freedom [49–51], would lead to new insight for further theoretical developments.

2. Neural network architectures

Both the critic and generator are implemented as deep residual convolutional neural networks, where residual functions with respect to the layer inputs are learned (Fig. 4C), which allows for more stable training [52]. In

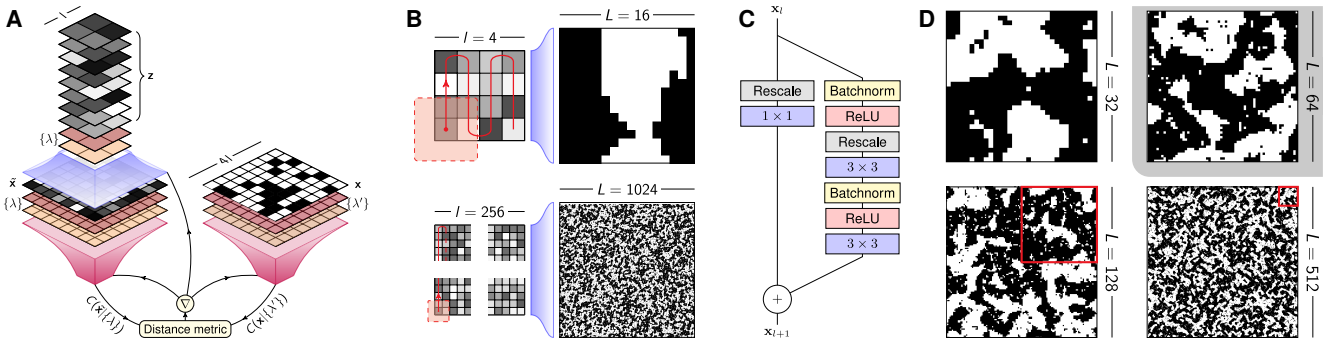


FIG. 4. Generating Ising model configurations with a RUGAN. **A**, The setup of the RUGAN method. The generator G (blue) transforms latent samples of size l, and optionally labels $\{\lambda\}$, to proposed configurations of size $4l$. The critic C (pink) is used to measure the distance between the distributions of generated and real configurations. This information is used to update both neural networks. **B**, The generator consists of translationally equivariant convolutional operations and therefore can be applied to inputs of arbitrary size, allowing for the creation of configurations at different spatial scales. **C**, The l-th hidden layer of the residual convolutional networks transforms an input $\mathbf{x}_l$ into $\mathbf{x}_{l+1}$, which can have a different spatial scale than the input. When reducing the spatial scale, the rescaling operation is an average-pooling layer with kernel size 2 and stride 2. When increasing the spatial scale, the rescaling consists of nearest-neighbor interpolation with a scale factor of 2. The convolutional operations with their corresponding kernel sizes are shown in blue. Note that we do not use batch normalization in the critic [46]. **D**, Ising configurations created by a RUGAN, trained on $L = 64$ configurations, at different spatial scales. The training size is shown in red on the larger configurations.

order to be equivariant under translational operations and achieve the upscaling described in the main text, the generator consists only of convolutional layers, with no dense (i.e. fully-connected) layers that are commonly included in neural networks (Fig. 4C). We add a hyperbolic tangent activation function to the last layer of the generator in order to obtain a valid output representation. As we want to generate configurations with an approximately circular shape for the optical lattice data set, we manually apply a mask that sets the borders to zero after creating a square configuration with the generative network.

For the hyperparameters in our WGAN implementation, we use $\lambda_1 = 10$ and $\lambda_2 = 2$ in Eq. (7) [46, 47]. The dropout rate is set to 25% for two layers in the critic to evaluate Eq. (7c). The weights of the neural networks are optimized with the ADAM optimizer [53], where we set the learning rate $\alpha = 10^{-4}$ and the exponential decay rates for the first and second moment estimates to $\beta_1 = 0$ and $\beta_2 = 0.9$. To gain a more reliable estimate of the Wasserstein distance before updating the generator's weights, on the experimental data we train the critic on 20 batches for every generator training iteration. For the classical spin models we use a 10:1 ratio of critic updates to generator updates. Each model is trained for 2000 epochs.

Details on the network architectures (*e.g.*, number of layers and channels) can be found in our open-source implementation at `http://clean.energyscience.ca/codes`.

3. *Model selection*

Once training is complete, we use each model epoch to generate a number of configurations for every conditioning label. As the Wasserstein distance quickly stabilizes (after a couple of epochs) to a constant value during training, we resort to a different selection criterion to decide on which model is ultimately deployed. Here, we evaluate the squared deviation between several observables (as shown in the main text) measured with the

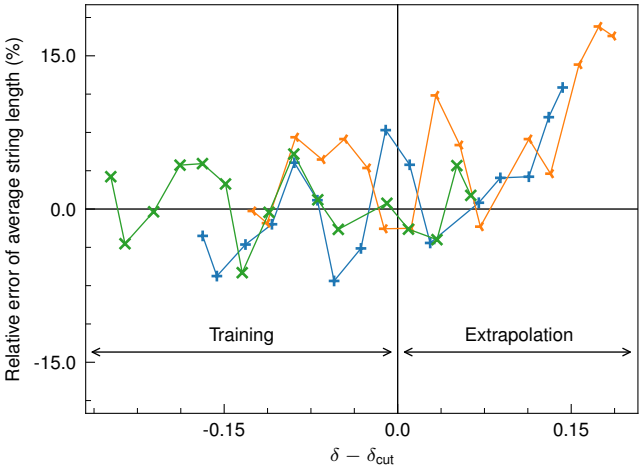


FIG. 5. Relative error on the average string length for RUGANs trained on different subsets of the available data. For the green line, the doping values on which we train are cut off to only include the 12 lowest values (same as in Fig. 2F); this is 9 and 7 for the blue and orange line respectively.

experimental and RUGAN snapshots (weighted by the experimental error) and select the model that minimizes this deviation. Naturally, when data corresponding to certain conditioning labels is not shown during training (*i.e.*, for the demonstration of interpolation and extrapolation), data generated at these labels is also not used for model selection, and we select the model that performs best on the training regime. The results obtained when interpolating between the lowest and highest doping values (Fig. 2C,D) consistently match well with experimental values across the model epochs. As expected, in the case of extrapolation, the results when generating data at doping values much higher than those trained on are less robust, as detailed in Fig. 2E,F. We also demonstrate this in Fig. 5, where we provide the RUGAN with even smaller subsets of the available doping values than in Fig. 2E,F. Though again better than theoretical predictions, the string statistics measured on the synthetic configurations start to deviate from the experimental observations for the highest doping values, even for the model epochs that perform best on the training set.

4. *An example: the Ising model*

We now give a detailed illustration of our framework on the prototypical classical Ising model on a two-dimensional square lattice of length L with Hamiltonian $\mathcal{H} = -J\sum_{\langle \mathbf{i},\mathbf{j}\rangle} s_\mathbf{i} s_\mathbf{j}$, where the sum runs over nearest-neighbor spins and we set $J = 1$. For N binary spins $s \in \{-1, +1\}$, the configuration space of the Ising model has a dimension of 2^N; sampling configurations with desired properties directly from this space is intractable for all but trivial system sizes. Here we show that this task can be accomplished with a RUGAN by training it on a data set of small-scale Ising configurations, and conditioning it on the energy and magnetization density $m = \sum_i s_i/N$ of each training example. The conditioning allows for the efficient creation of microstates with desired properties from the high-dimensional configuration space as well as the creation of microstates with conditioning labels for which no training examples are available. We implemented a modified form of umbrella sampling called "targeted sampling" [11] so that we can obtain a training set with a uniform energy distribution. This sampling method resembles the Metropolis-Hastings algorithm in structure, but instead of seeking low-energy states, we target specific energies, accepting configurations that move us toward the target energy, and rejecting ones (with a Gaussian probability) that lead us away. In doing this, we can collect examples across the energy spectrum in an efficient way. In Fig. 6A, we train a RUGAN on a data set of Ising microstates restricted to high and low energy values and magnetizations near zero, and use it to sample the entire space of possible energy and magnetization combinations. The generator only makes large errors on the conditioning label combinations with a relatively small density of

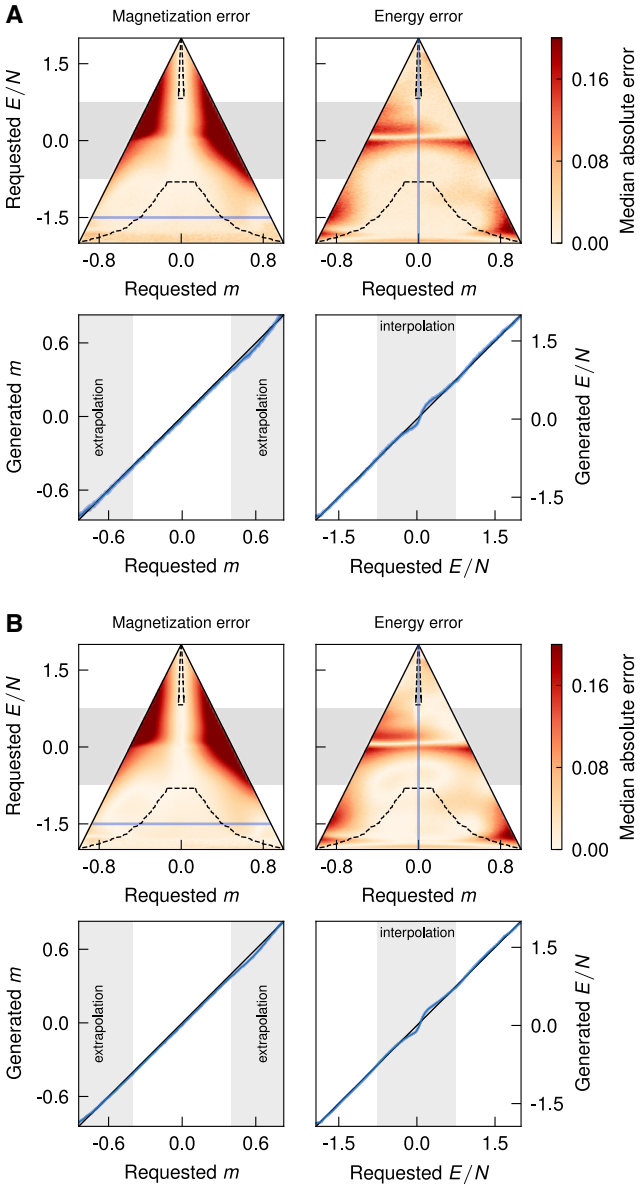


FIG. 6. Generating Ising model configurations with a RUGAN. **A**, During training, we condition the RUGAN on the energy and magnetization of each configuration, and only show it examples with labels in the regions enclosed by dashed lines. After training, it is requested to create configurations with all possible labels. The median absolute error made here is shown in the top row. In the bottom row, we show the average error and its standard deviation at fixed energy (left) and fixed magnetization (right), indicated by the blue lines in the top row. The results are obtained by sampling 100 configurations at possible combinations of the energy and magnetization per site with energy spacing $\Delta(E/L^2) = 1/64$ and magnetization spacing $\Delta(m) = 1/1024$. Results shown here are for the training system size $L = 64$. **B**, Same as A, but now with configurations created at a larger scale $L = 256$, containing 16 times more spins than the training examples.

states. Note that spin-flip symmetry is not enforced, which could potentially decrease the errors shown here. The accuracy in this conditioning is retained when using the upscaling property to create configurations at much larger scales (Fig. 6B). This implies that we can greatly accelerate sampling of uncorrelated large scale synthetic configurations, as costly simulations are only needed for a small subset of the configuration space and only of small-scale microstates.

B. Hidden order in the two-dimensional Fermi-Hubbard model

To benchmark the string patterns in the experimental data and our RUGAN, we compare them to the frameworks of sprinkled holes and geometric string theory [30] and apply an analysis procedure identical to what has been previously used to describe experimental observations [29]. For simplicity, we here recapitulate these but point towards Refs. [29, 30] for more detailed information.

1. Sprinkled holes and geometric string theory

Sprinkled holes is a model for the doped Fermi-Hubbard model in the limit of non-interacting holes. To obtain snapshots at different dopings, we start from experimentally obtained snapshots at half filling and add holes on random positions until the doping matches the requested one.

Geometric string theory is a theoretical model where holes do not interact with each other but do interact with the surrounding spins. First, a single hole is placed at a random position on the lattice. The dynamics of the hole can be described by introducing an effective Hamiltonian and an effective Hilbert space for a single string [29, 30]. The Hilbert space consists of string patterns, which can be viewed as paths without loops on a Cayley tree with coordination number $z = 4$. Using the frozen spin approximation and $U \gg t$, the strings can be modeled by the effective Schrödinger equation

$$t\sum_{s}^{z-1}\psi_{l+1,s} + t\psi_{l-1} + V_l\psi_l = E\psi_l, \tag{9}$$

where ψ_l is a shorthand notation for a path on the Cayley tree of length l, and $\psi_{l+1,s}$ denotes the string obtained by continuing the string ψ_l along one of the $z-1$ directions on the Cayley tree. The parameter t is the coupling constant for tunneling between string lengths (equal to t in the Fermi-Hubbard Hamiltonian) and V_l is an effective potential. The effective potential $V_l = (\mathrm{d}E/\mathrm{d}l)l + g\delta_{l,0}$ consists of a linear tension with magnitude $\mathrm{d}E/\mathrm{d}l$ and an attractive term with magnitude g. Solving this string model for finite temperature yields a string length distribution $p(l)$. The snapshots are then created by starting from experimental snapshots at half filling and adding strings on random positions in the lattice—with a length according to the string length distribution $p(l)$—until the desired doping is reached. More details can be found in Refs. [29, 30].

2. String detection algorithm

In the main text the number of strings and their average length as a function of doping are compared between the RUGAN, experiment and the theories explained above. The detection algorithm of these strings is applied to snapshots where one of the two spin species and doublons are removed, and is performed in multiple steps [29]. The geometric strings describe the deviation between the doped Fermi-Hubbard snapshots and a checkerboard state. Hence, the first step involves selecting a window (here with a diameter of 7 sites) for each configuration with the highest staggered magnetization. Using a window with a diameter smaller than the configuration itself negates some of the finite temperature effects. For a given doping, 60% of the resulting windows with the highest staggered magnetization are kept for further analysis. In the next step, each of these windows is compared to a checkerboard state. The strings are then identified as deviations from this checkerboard, and the string-pattern length distribution $p^\delta(l)$ is measured.

As for the string count shown in Fig. 2, only those patterns of length greater than two are included as to negate the contribution from quantum fluctuations such as doublon-hole pairs [29]. The average string lengths $\bar{l}(\delta)$ are calculated from the string length histograms as $\bar{l}(\delta) = \sum_l l p^\delta(l) / \sum_l p^\delta(l)$.

3. Spin-spin correlators

One way to assert the validity of the snapshots created by the RUGAN described in the main text is to verify whether the sign-corrected two-point spin correlator

$$C_s(r) = 4(-1)^{\|\mathbf{r}\|_1}\left(\left\langle \hat{S}^z_{\mathbf{i}}\hat{S}^z_{\mathbf{i}+\mathbf{r}}\right\rangle - \left\langle \hat{S}^z_{\mathbf{i}}\right\rangle\left\langle \hat{S}^z_{\mathbf{i}+\mathbf{r}}\right\rangle\right) \tag{10}$$

matches well with experimental results. Here, $\hat{S}^z_{\mathbf{i}} = \frac{1}{2}(\hat{n}^\uparrow_{\mathbf{i}} - \hat{n}^\downarrow_{\mathbf{i}})$ with $\hat{n}^\sigma_{\mathbf{i}}$ the number operator for spin σ on site $\mathbf{i}$. The spin correlator can be calculated from the experimental snapshots as [31]

$$\begin{aligned} C_s(r) = (-1)^{\|\mathbf{r}\|_1}\Bigg[2\sum_{\sigma\in\{\uparrow,\downarrow\}}\left(\langle pp\rangle_{R\sigma} - \langle p\rangle^2_{R\sigma}\right) \\ - \left(\langle pp\rangle_{NR} - \langle p\rangle^2_{NR}\right)\Bigg], \end{aligned} \tag{11}$$

where the spatial indices are dropped for simplicity. Here, p denotes a singly occupied site, the expectation value $\langle\cdot\rangle_{NR}$ is taken over images where neither spin species was removed, and $\langle\cdot\rangle_{R\sigma}$ over images where the spin state σ was removed. To calculate the expectation values $\langle\cdot\rangle_{NR}$, we train a second RUGAN on a data set of snapshots containing both spin species, and also condition it on the doping. Here, the doping conditioning can be explicitly checked, as the doping $\delta \approx 1.22(0.905 - n_s)$ where n_s is the density of singly-occupied sites [29].

4. Data set

The experimental data of the Fermi-Hubbard model is obtained from Ref. [28]. These data are obtained at temperature $T = (0.65 \pm 0.04)J$, and the ratio $U/t = 8.1(2)$. Here, a mixture of the two lowest hyperfine states of ^{6}Li is trapped in a two-dimensional optical lattice. Site-resolved measurements of the occupation in the optical lattice are obtained with high-resolution quantum gas microscopy [33]. The experimental snapshots of the atomic distributions consist of a circular region of 80 sites. In total, 8822 images of the atomic distributions with both spin components present and 17233 with one spin component removed are available. We augment the data set by also including these samples rotated by multiples of 90°.

C. Acknowledgements

We thank Christie S. Chiu, Phil de Luna, Dennis Klug, Roger Melko, Jannes Nys and Michael Schuurman for helpful discussions. Work at NRC was carried out under the auspices of the MCF and AI4D Program. The computational resources and services used in this work were partly provided by the VSC and the Flemish Government – department EWI. T.V. is supported as an 'FWO-aspirant' under contract number FWO18/ASP/279. K.M. & I.T. acknowledge support from NSERC. The Titan V used for this research was donated by the NVIDIA Corporation.

[1] M. Lewenstein, A. Sanpera, V. Ahufinger, B. Damski, A. Sen(De), and U. Sen, Advances in Physics **56**, 243 (2007).
[2] I. Bloch, J. Dalibard, and W. Zwerger, Reviews of Modern Physics **80**, 885 (2008).
[3] C. Gross and I. Bloch, Science **357**, 995 (2017).
[4] I. Georgescu, S. Ashhab, and F. Nori, Reviews of Modern Physics **86**, 153 (2014).
[5] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová, Reviews of Modern Physics **91**, 045002 (2019).
[6] E. P. L. van Nieuwenburg, Y.-H. Liu, and S. D. Huber, Nature Physics **13**, 435 (2017).
[7] J. Carrasquilla and R. G. Melko, Nature Physics **13**, 431 (2017).
[8] Y.-H. Liu and E. P. van Nieuwenburg, Physical Review Letters **120**, 176401 (2018).
[9] K. Ch'ng, J. Carrasquilla, R. G. Melko, and E. Khatami, Physical Review X **7**, 031038 (2017).
[10] K. Mills, M. Spanner, and I. Tamblyn, Physical Review A **96**, 042113 (2017).
[11] K. Mills and I. Tamblyn, Physical Review E **97** (2018), 10.1103/PhysRevE.97.032119.
[12] Y. Zhang, A. Mesaros, K. Fujita, S. D. Edkins, M. H. Hamidian, K. Ch'ng, H. Eisaki, S. Uchida, J. C. S. Davis, E. Khatami, and E.-A. Kim, Nature **570**, 484 (2019).
[13] A. Bohrdt, C. S. Chiu, G. Ji, M. Xu, D. Greif, M. Greiner, E. Demler, F. Grusdt, and M. Knap, Nature Physics **15**, 921 (2019).
[14] B. S. Rem, N. Käming, M. Tarnowski, L. Asteria, N. Fläschner, C. Becker, K. Sengstock, and C. Weitenberg, Nature Physics **15**, 917 (2019).
[15] F. Noé, S. Olsson, J. Köhler, and H. Wu, Science **365**, eaaw1147 (2019).
[16] G. Carleo and M. Troyer, Science **355**, 602 (2017).
[17] G. Torlai, G. Mazzola, J. Carrasquilla, M. Troyer, R. Melko, and G. Carleo, Nature Physics **14**, 447 (2018).
[18] G. Torlai and R. Melko, Annual Review of Condensed Matter Physics (2020), 10.1146/annurev-conmatphys-031119-050651.
[19] K. Choo, G. Carleo, N. Regnault, and T. Neupert, Physical Review Letters **121**, 167204 (2018).
[20] T. Vieijra, C. Casert, J. Nys, W. De Neve, J. Haegeman, J. Ryckebusch, and F. Verstraete, Physical Review Letters **In press**, arXiv: 1905.06034.
[21] R. G. Melko, G. Carleo, J. Carrasquilla, and J. I. Cirac, Nature Physics **15**, 887 (2019).
[22] Z.-Y. Han, J. Wang, H. Fan, L. Wang, and P. Zhang, Physical Review X **8**, 031012 (2018).
[23] E. Stoudenmire and D. J. Schwab, in *Advances in Neural Information Processing Systems 29* (2016) pp. 4799–4807.
[24] E. M. Stoudenmire, Quantum Science and Technology **3**, 034003 (2018).
[25] P. A. Lee, N. Nagaosa, and X.-G. Wen, Rev. Mod. Phys. **78**, 69 (2006).
[26] B. Keimer, S. A. Kivelson, M. R. Norman, S. Uchida, and J. Zaanen, Nature **518**, 179 (2015).
[27] T. Esslinger, Annual Review of Condensed Matter Physics **1**, 129 (2010).
[28] C. Chiu, G. Ji, A. Bohrdt, M. Xu, M. Knap, E. Demler, F. Grusdt, M. Greiner, and D. Greif, (2019), 10.7910/DVN/1CSVBV.
[29] C. S. Chiu, G. Ji, A. Bohrdt, M. Xu, M. Knap, E. Demler, F. Grusdt, M. Greiner, and D. Greif, Science **365**, 251 (2019).
[30] F. Grusdt, M. Kánasz-Nagy, A. Bohrdt, C. Chiu, G. Ji, M. Greiner, D. Greif, and E. Demler, Physical Review X **8**, 011046 (2018).

[31] M. F. Parsons, A. Mazurenko, C. S. Chiu, G. Ji, D. Greif, and M. Greiner, Science **353**, 1253 (2016).
[32] T. A. Hilker, G. Salomon, F. Grusdt, A. Omran, M. Boll, E. Demler, I. Bloch, and C. Gross, Science **357**, 484 (2017).
[33] A. Mazurenko, C. S. Chiu, G. Ji, M. F. Parsons, M. Kanász-Nagy, R. Schmidt, F. Grusdt, E. Demler, D. Greif, and M. Greiner, Nature **545**, 462 (2017).
[34] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, arXiv:1406.2661 (2014).
[35] M. Mirza and S. Osindero, arXiv:1411.1784 [cs, stat] (2014), arXiv: 1411.1784.
[36] M. Arjovsky, S. Chintala, and L. Bottou, arXiv:1701.07875 (2017).
[37] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, arXiv:1609.04802 (2016).
[38] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, arXiv:1703.05192 (2017).
[39] Z. Liu, S. P. Rodrigues, and W. Cai, arXiv:1710.04987 (2017).
[40] K. Mills and I. Tamblyn, arXiv:1710.08053 (2017).
[41] J. M. Urban and J. M. Pawlowski, arXiv:1811.03533 (2018).
[42] K. Zhou, G. Endrődi, L.-G. Pang, and H. Stöcker, Physical Review D **100**, 011501 (2019).
[43] B. Sanchez-Lengeling and A. Aspuru-Guzik, Science **361**, 360 (2018).
[44] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, arXiv:1805.08318 (2019).
[45] L. McInnes, J. Healy, N. Saul, and L. Großberger, Journal of Open Source Software **3**, 861 (2018).
[46] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, arXiv:1704.00028 (2017).
[47] X. Wei, B. Gong, Z. Liu, W. Lu, and L. Wang, arXiv:1803.01541 (2018).
[48] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, arXiv:1207.0580 (2012).
[49] P. Ponte and R. G. Melko, Physical Review B **96**, 205146 (2017).
[50] S. J. Wetzel and M. Scherzer, Physical Review B **96**, 184410 (2017).
[51] C. Casert, T. Vieijra, J. Nys, and J. Ryckebusch, Physical Review E **99**, 023304 (2019).
[52] K. He, X. Zhang, S. Ren, and J. Sun, arXiv:1512.03385 (2015).
[53] D. P. Kingma and J. Ba, arXiv:1412.6980 (2014).

Appendix B

Sampling methods

B.1 Metropolis–Hastings

The purpose of this appendix is to elaborate on the process of targeted energy sampling, specifically applied to the Ising model.

The typical Metropolis–Hastings algorithm, when applied to the Boltzmann distribution, proceeds as follows:

1. Generate a starting configuration, A
2. Propose a new configuration, B by a slight, random modification of A. In the case of the Ising model, we flip a single spin at random.
3. If the energy of the second configuration B has a lower energy than A, accept the transition and the new state becomes B, otherwise reject it with probability

$$p(\text{reject}) = 1 - e^{-(E_B - E_A)/kT}, \tag{B.1}$$

where T is the effective temperature of the system and k is the Boltzmann

constant.

4. If B was accepted, set $A := B$, otherwise leave A unchanged.

5. Record configuration A (if statistics are desired)

6. Return to step (2).

When completed correctly (e.g. sufficiently uncorrelated samples are recorded), this results in a thermal distribution: the Boltzmann distribution.

This process inherently targets low energy (the most stable) configurations as random fluctuations that result in a lower energy are always accepted. We would like a method, however, that can acquire configurations from the entire range of energies. To achieve this, we modify the Boltzmann distribution in the Metropolis–Hastings method above to seek a specific "target" energy E_{target}; instead of using the Boltzmann probability, use a Gaussian rejection probability centered at a single target energy. The process is as follows:

A targeted sampling run

1. Choose a target energy E_{target}

2. Generate a starting configuration, A

3. Propose a new configuration, B by flipping a single spin at random.

4. If $|E_B - E_{\text{target}}| < |E_A - E_{\text{target}}|$, accept B, otherwise reject the move with probability

$$p(\text{reject}) = 1 - e^{-(E_B - E_A)^2/kT}, \tag{B.2}$$

   where the "thermal energy" kT defines a Gaussian-shaped envelope around the target energy that will be accessible to the algorithm.

5. If B was accepted, set $A := B$, otherwise leave A unchanged.

6. Record configuration A.

7. Return to step (3), repeating the process N_{steps} times.

8. Repeat the entire process from (1) for N_{seed} seeds.

When recording configurations to use in the training data set, it is necessary to not record configurations at every iteration to avoid the examples being correlated. The correlation interval t_{corr} depends on the system, and the degree to which a single modification affects the position in configuration space.

To generate an Ising data set using TS, I implement a multi-pass approach, sweeping multiple times across the range of energies, collecting examples and placing them in N_{bin} evenly-spaced bins over the range of energies, from E_{min} to E_{max}. The algorithm terminates once all bins contain N_{cap} configurations. In the case of the Ising model, determination of E_{max} and E_{min} is simple, as we know that the checkerboard and uniform-spin configurations represent the

extrema. Without this prior knowledge, the range could still be determined by performing two TS runs with the target energy set to extreme values, say -10^{10} and 10^{10}, allowing the algorithm to "discover" the lowest and highest possible energies, after sufficient equilibration time. The entire process goes as follows:

1. Determine the range of possible energies, and create an array of size [N_{bin}, N_{cap}, ...], to hold the accepted configurations. Here the ellipsis (...) represents an arbitrarily shaped configuration (e.g. [16,16] for the Ising model of that size).

Phase I: Discovery

2. Generate a random starting configuration. Set $E_{\text{target}} = E_{\text{min}}$ and run a short TS run. As equilibration proceeds, an example is recorded every t_{corr} iterations.

3. We then set $E_{\text{target}} = E_{\text{max}}$ and run a short TS run to discover high-energy configurations. The equilibration process gives the algorithm the opportunity to explore configuration space as it moves toward the target energy, and for this reason, we do not use a large equilibration period prior to recording examples.

4. We then set $E_{\text{target}} = 0.5(E_{\text{min}} + E_{\text{max}})$ and perform another TS run.

   During the discovery phase, each TS run is repeated four times, each

with different seed configurations. A high value of kT ($kT = 0.1$) is used as discovery is the goal and a wide target envelope is desired.

Phase II: Energy sweep

5. Divide the energy range into 10 evenly spaced target energies.
6. Run a TS run with each of these energies as the target energy.
7. Divide the energy range into 100 evenly spaced target energies.
8. Run a TS run with each of these energies as the target energy.

   During the energy sweep phase, kT is lowered to 0.01 as we want to focus more closely on the target energies.

Phase III: Bin filling

The following steps (9 through 12) are repeated until all bins contain either zero or N_{cap} examples. N_{iter}, the number of Metropolis-Hastings iterations per target energy is set to $2t_{\text{corr}}$.

9. N_{iter} is set to 2
10. A list of all partially-filled bins is acquired.
11. For each partially-filled bin, a TS run is performed with the bin center as the target energy.

12. N_{iter} is doubled. This is because if by this point the bin is not full, it is likely because the TS run is terminating before equilibration to the target energy is reached. Increasing N_{iter} is equivalent to giving the algorithm more opportunity to equilibrate, reaching the region in configuration space where relevant configurations live.

    After all bins are either completely full, or completely empty, we save all of the accepted configurations to a data file. This process gives us a perfectly even distribution of examples over the entire range. The algorithm itself is inherently sequential, but it can be run in multiple processes in parallel, using different starting seeds and acquiring training examples with linear scaling with respect to number of processes.

Appendix C

Reproduction rights

C.0.1 Convolutional Neural Networks for Atomistic systems

"As the author of this Elsevier article, you retain the right to include it in a thesis or dissertation, provided it is not published commercially. Permission is not required, but please ensure that you reference the journal as the original source. "

C.0.2 Extensive Deep Neural Networks

American Physical Society
Reuse and Permissions License

20-Jan-2021

This license agreement between the American Physical Society ("APS") and Kyle Mills ("You") consists of your license details and the terms and conditions provided by the American Physical Society and SciPris.

Licensed Content Information

License Number:	**RNP/21/JAN/035451**
License date:	20-Jan-2021
DOI:	10.1103/PhysRevA.96.042113
Title:	Deep learning and the Schrodinger equation
Author:	Kyle Mills, Michael Spanner, and Isaac Tamblyn
Publication:	Physical Review A
Publisher:	American Physical Society
Cost:	USD $ 0.00

Request Details

Does your reuse require significant modifications: No

Specify intended distribution locations:	Worldwide
Reuse Category:	Reuse in a thesis/dissertation
Requestor Type:	Author of requested content
Items for Reuse:	Whole Article
Format for Reuse:	Electronic

Information about New Publication:

University/Publisher:	Ontario Tech University
Title of dissertation/thesis:	Deep learning and physics
Author(s):	Kyle Mills
Expected completion date:	Apr. 2021

License Requestor Information

Name:	Kyle Mills
Affiliation:	Individual
Email Id:	kyle.mills@ontariotechu.net
Country:	Canada

American Physical Society
Reuse and Permissions License

TERMS AND CONDITIONS

The American Physical Society (APS) is pleased to grant the Requestor of this license a non-exclusive, non-transferable permission, limited to Electronic format, provided all criteria outlined below are followed.

1. You must also obtain permission from at least one of the lead authors for each separate work, if you haven't done so already. The author's name and affiliation can be found on the first page of the published Article.
2. For electronic format permissions, Requestor agrees to provide a hyperlink from the reprinted APS material using the source material's DOI on the web page where the work appears. The hyperlink should use the standard DOI resolution URL, http://dx.doi.org/{DOI}. The hyperlink may be embedded in the copyright credit line.
3. For print format permissions, Requestor agrees to print the required copyright credit line on the first page where the material appears: "Reprinted (abstract/excerpt/figure) with permission from [(FULL REFERENCE CITATION) as follows: Author's Names, APS Journal Title, Volume Number, Page Number and Year of Publication.] Copyright (YEAR) by the American Physical Society."
4. Permission granted in this license is for a one-time use and does not include permission for any future editions, updates, databases, formats or other matters. Permission must be sought for any additional use.
5. Use of the material does not and must not imply any endorsement by APS.
6. APS does not imply, purport or intend to grant permission to reuse materials to which it does not hold copyright. It is the requestor's sole responsibility to ensure the licensed material is original to APS and does not contain the copyright of another entity, and that the copyright notice of the figure, photograph, cover or table does not indicate it was reprinted by APS with permission from another source.
7. The permission granted herein is personal to the Requestor for the use specified and is not transferable or assignable without express written permission of APS. This license may not be amended except in writing by APS.
8. You may not alter, edit or modify the material in any manner.
9. You may translate the materials only when translation rights have been granted.
10. APS is not responsible for any errors or omissions due to translation.
11. You may not use the material for promotional, sales, advertising or marketing purposes.
12. The foregoing license shall not take effect unless and until APS or its agent, Aptara, receives payment in full in accordance with Aptara Billing and Payment Terms and Conditions, which are incorporated herein by reference.
13. Should the terms of this license be violated at any time, APS or Aptara may revoke the license with no refund to you and seek relief to the fullest extent of the laws of the USA. Official written notice will be made using the contact information provided with the permission request. Failure to receive such notice will not nullify revocation of the permission.
14. APS reserves all rights not specifically granted herein.
15. This document, including the Aptara Billing and Payment Terms and Conditions, shall be the entire agreement between the parties relating to the subject matter hereof.

American Physical Society
Reuse and Permissions License

20-Jan-2021

This license agreement between the American Physical Society ("APS") and Kyle Mills ("You") consists of your license details and the terms and conditions provided by the American Physical Society and SciPris.

Licensed Content Information

License Number:	**RNP/21/JAN/035452**
License date:	20-Jan-2021
DOI:	10.1103/PhysRevE.97.032119
Title:	Deep neural networks for direct, featureless learning through observation: The case of two-dimensional spin models
Author:	Kyle Mills and Isaac Tamblyn
Publication:	Physical Review E
Publisher:	American Physical Society
Cost:	USD $ 0.00

Request Details

Does your reuse require significant modifications: No

Specify intended distribution locations:	Worldwide
Reuse Category:	Reuse in a thesis/dissertation
Requestor Type:	Author of requested content
Items for Reuse:	Whole Article
Format for Reuse:	Electronic

Information about New Publication:

University/Publisher:	Ontario Tech University
Title of dissertation/thesis:	Deep learning and physics
Author(s):	Kyle Mills
Expected completion date:	Apr. 2021

License Requestor Information

Name:	Kyle Mills
Affiliation:	Individual
Email Id:	kyle.mills@ontariotechu.net
Country:	Canada

American Physical Society
Reuse and Permissions License

TERMS AND CONDITIONS

The American Physical Society (APS) is pleased to grant the Requestor of this license a non-exclusive, non-transferable permission, limited to Electronic format, provided all criteria outlined below are followed.

1. You must also obtain permission from at least one of the lead authors for each separate work, if you haven't done so already. The author's name and affiliation can be found on the first page of the published Article.
2. For electronic format permissions, Requestor agrees to provide a hyperlink from the reprinted APS material using the source material's DOI on the web page where the work appears. The hyperlink should use the standard DOI resolution URL, http://dx.doi.org/{DOI}. The hyperlink may be embedded in the copyright credit line.
3. For print format permissions, Requestor agrees to print the required copyright credit line on the first page where the material appears: "Reprinted (abstract/excerpt/figure) with permission from [(FULL REFERENCE CITATION) as follows: Author's Names, APS Journal Title, Volume Number, Page Number and Year of Publication.] Copyright (YEAR) by the American Physical Society."
4. Permission granted in this license is for a one-time use and does not include permission for any future editions, updates, databases, formats or other matters. Permission must be sought for any additional use.
5. Use of the material does not and must not imply any endorsement by APS.
6. APS does not imply, purport or intend to grant permission to reuse materials to which it does not hold copyright. It is the requestor's sole responsibility to ensure the licensed material is original to APS and does not contain the copyright of another entity, and that the copyright notice of the figure, photograph, cover or table does not indicate it was reprinted by APS with permission from another source.
7. The permission granted herein is personal to the Requestor for the use specified and is not transferable or assignable without express written permission of APS. This license may not be amended except in writing by APS.
8. You may not alter, edit or modify the material in any manner.
9. You may translate the materials only when translation rights have been granted.
10. APS is not responsible for any errors or omissions due to translation.
11. You may not use the material for promotional, sales, advertising or marketing purposes.
12. The foregoing license shall not take effect unless and until APS or its agent, Aptara, receives payment in full in accordance with Aptara Billing and Payment Terms and Conditions, which are incorporated herein by reference.
13. Should the terms of this license be violated at any time, APS or Aptara may revoke the license with no refund to you and seek relief to the fullest extent of the laws of the USA. Official written notice will be made using the contact information provided with the permission request. Failure to receive such notice will not nullify revocation of the permission.
14. APS reserves all rights not specifically granted herein.
15. This document, including the Aptara Billing and Payment Terms and Conditions, shall be the entire agreement between the parties relating to the subject matter hereof.

Adversarial Generation of Mesoscale Surfaces from Small-Scale Chemical Motifs

Author: Kyle Mills, Corneel Casert, Isaac Tamblyn
Publication: The Journal of Physical Chemistry C
Publisher: American Chemical Society
Date: Oct 1, 2020

PERMISSION/LICENSE IS GRANTED FOR YOUR ORDER AT NO CHARGE

This type of permission/license, instead of the standard Terms & Conditions, is sent to you because no fee is being charged for your order. Please note the following:

- Permission is granted for your request in both print and electronic formats, and translations.
- If figures and/or tables were requested, they may be adapted or used in part.
- Please print this page for your records and send a copy of it to your publisher/graduate school.
- Appropriate credit for the requested material should be given as follows: "Reprinted (adapted) with permission from (COMPLETE REFERENCE CITATION). Copyright (YEAR) American Chemical Society." Insert appropriate information in place of the capitalized words.
- One-time permission is granted only for the use specified in your request. No additional uses are granted (such as derivative works or other editions). For any other uses, please submit a new request.

BACK CLOSE WINDOW

Finding the ground state of spin Hamiltonians with reinforcement learning

Author: Kyle Mills et al

Publication: Nature Machine Intelligence

Publisher: Springer Nature

Date: Sep 7, 2020

Author Request

If you are the author of this content (or his/her designated agent) please read the following. If you are not the author of this content, please click the Back button and select no to the question "Are you the Author of this Springer Nature content?".
Ownership of copyright in original research articles remains with the Author, and provided that, when reproducing the contribution or extracts from it or from the Supplementary Information, the Author acknowledges first and reference publication in the Journal, the Author retains the following non-exclusive rights:
To reproduce the contribution in whole or in part in any printed volume (book or thesis) of which they are the author(s).
The author and any academic institution, where they work, at the time may reproduce the contribution for the purpose of course teaching.
To reuse figures or tables created by the Author and contained in the Contribution in oral presentations and other works created by them.
To post a copy of the contribution as accepted for publication after peer review (in locked Word processing file, of a PDF version thereof) on the Author's own web site, or the Author's institutional repository, or the Author's funding body's archive, six months after publication of the printed or online edition of the Journal, provided that they also link to the contribution on the publisher's website.
Authors wishing to use the published version of their article for promotional use or on a web site must request in the normal way.

If you require further assistance please read Springer Nature's online author reuse guidelines.

For full paper portion: Authors of original research papers published by Springer Nature are encouraged to submit the author's version of the accepted, peer-reviewed manuscript to their relevant funding body's archive, for release six months after publication. In addition, authors are encouraged to archive their version of the manuscript in their institution's repositories (as well as their personal Web sites), also six months after original publication.

v1.0

BACK CLOSE WINDOW

Bibliography

[1] K. Mills and I. Tamblyn, "Deep neural networks for direct, featureless learning through observation: The case of two-dimensional spin models," *Physical Review E*, vol. 97, no. 3, 2018.

[2] K. Mills, M. Spanner, and I. Tamblyn, "Deep learning and the Schrödinger equation," *Physical Review A*, vol. 96, no. 4, p. 042113, 2017.

[3] K. Ryczko, K. Mills, I. Luchak *et al.*, "Convolutional neural networks for atomistic systems," *Computational Materials Science*, vol. 149, pp. 134–142, 2018.

[4] K. Mills, K. Ryczko, I. Luchak *et al.*, "Extensive deep neural networks for transferring small scale learning to large scale systems," *Chemical Science*, vol. 10, no. 15, pp. 4129–4140, 2019.

[5] K. Mills and I. Tamblyn, "Weakly-supervised multi-class object localization using only object counts as labels," 2021.

[6] K. Mills and I. Tamblyn, "Phase space sampling and operator confidence with generative adversarial networks," *arXiv*, 2017.

[7] C. Casert, K. Mills, T. Vieijra *et al.*, "Optical lattice experiments at unob-

served conditions and scales through generative adversarial deep learning," *arXiv*, pp. 1–11, 2020.

[8] K. Mills, C. Casert, and I. Tamblyn, "Adversarial Generation of Mesoscale Surfaces from Small-Scale Chemical Motifs," *Journal of Physical Chemistry C*, vol. 124, no. 42, pp. 23 158–23 163, 2020.

[9] K. Mills, P. Ronagh, and I. Tamblyn, "Finding the ground state of spin Hamiltonians with reinforcement learning," *Nature Machine Intelligence*, vol. 2, no. 9, pp. 509–517, 2020.

[10] SwiftKey, "Introducing the world's first neural network keyboard," 2015.

[11] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *ACM International Conference Proceeding Series*, vol. 382. New York, New York, USA: ACM Press, 2009, pp. 873–880.

[12] P. Y. Izotov, N. L. Kazanskiy, D. L. Golovashkin *et al.*, "CUDA-enabled implementation of a neural network algorithm for handwritten digit recognition," *Optical Memory and Neural Networks (Information Optics)*, vol. 20, no. 2, pp. 98–106, 2011.

[13] S. Chetlur, C. Woolley, P. Vandermersch *et al.*, "cuDNN: Efficient Primitives for Deep Learning," *arXiv*, pp. 1–9, 2014.

[14] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[15] D. Silver, A. Huang, C. J. Maddison *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[16] D. Hassabis and G. Deepmind, "AlphaGo : Mastering the ancient game of Go with," 2016.

[17] M. Campbell, A. J. Hoane, and F. H. Hsu, "Deep Blue," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[18] J. Burmeister and J. Wiles, "The challenge of go as a domain for AI research: A comparison between go and chess," *ANZIIS 1995 - Proceedings of the 3rd Australian and New Zealand Conference on Intelligent Information Systems*, pp. 181–186, 1995.

[19] D. Silver, J. Schrittwieser, K. Simonyan *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[20] D. Hebb, *The Organization of Behavior*. Wiley, 2005.

[21] F. Rosenblatt, "The perceptron: A probabilistic model for information

storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[22] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[23] J. Nievergelt, *R69-13 Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969, vol. C-18, no. 6.

[24] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *Bit*, vol. 16, no. 2, pp. 146–160, 1976.

[25] S. Linnainmaa, "Alogritmin kumulatiivinen pyoristysvirhe yksittaisten pyoristysvirheiden Taylor-kehitelmana," 1970.

[26] J. Schmidhuber, "Deep Learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[27] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *System Modeling and Optimization*, 2005, pp. 762–770.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[29] D. A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (ELUs),” Diploma Thesis, TU Munich, 2016.

[30] Y. LeCun, L. Bottou, Y. Bengio *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[31] R. Raina, A. Madhavan, and A. Y. Ng, “Large-scale deep unsupervised learning using graphics processors,” Tech. Rep., 2009.

[32] D. C. CireÅ§an, U. Meier, L. M. Gambardella *et al.*, “Deep, big, simple neural nets for handwritten digit recognition,” *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” Tech. Rep. 6, 2017.

[34] M. Abadi, P. Barham, J. Chen *et al.*, “TensorFlow: A system for large-scale machine learning,” *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, pp. 265–283, 2016.

[35] A. Paszke, S. Gross, F. Massa *et al.*, “PyTorch: An imperative style, high-performance deep learning library,” *arXiv*, 2019.

[36] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[37] J. Carrasquilla and R. G. Melko, "Machine learning phases of matter," *Nature Physics*, vol. 13, no. 5, pp. 431–434, 2017.

[38] L. Wang, "Discovering phase transitions with unsupervised learning," *Physical Review B*, vol. 94, no. 19, p. 195105, 2016.

[39] K. Ch'Ng, J. Carrasquilla, R. G. Melko *et al.*, "Machine learning phases of strongly correlated fermions," *Physical Review X*, vol. 7, no. 3, p. 031038, 2017.

[40] A. Tanaka and A. Tomiya, "Detection of phase transition via convolutional neural networks," *Journal of the Physical Society of Japan*, vol. 86, no. 6, pp. 1–7, 2017.

[41] A. Aurisano, A. Radovic, D. Rocco *et al.*, "A convolutional neural network neutrino event classifier," *Journal of Instrumentation*, vol. 11, no. 9, pp. P09 001–P09 001, 2016.

[42] S. Dieleman, K. W. Willett, and J. Dambre, "Rotation-invariant convolutional neural networks for galaxy morphology prediction," *Monthly Notices of the Royal Astronomical Society*, vol. 450, no. 2, pp. 1441–1459, 2015.

[43] S. Dieleman, K. W. Willett, and J. Dambre, "Rotation-invariant convolutional neural networks for galaxy morphology prediction," *Monthly Notices of the Royal Astronomical Society*, vol. 450, no. 2, pp. 1441–1459, 2015.

[44] M. J. Cherukara, B. Narayanan, A. Kinaci *et al.*, "Ab Initio-Based Bond Order Potential to Investigate Low Thermal Conductivity of Stanene Nanostructures," *Journal of Physical Chemistry Letters*, vol. 7, no. 19, pp. 3752–3759, 2016.

[45] M. Riera, A. W. Götz, and F. Paesani, "The i-TTM model for ab initio-based ion-water interaction potentials. II. Alkali metal ion-water potential energy functions," *Physical Chemistry Chemical Physics*, vol. 18, no. 44, pp. 30 334–30 343, 2016.

[46] A. Jaramillo-Botero, S. Naserifar, and W. A. Goddard, "General multiobjective force field optimization framework, with application to reactive force fields for silicon carbide," *Journal of Chemical Theory and Computation*, vol. 10, no. 4, pp. 1426–1439, 2014.

[47] B. W. Van Beest, G. J. Kramer, and R. A. Van Santen, "Force fields for silicas and aluminophosphates based on ab initio calculations," *Physical Review Letters*, vol. 64, no. 16, pp. 1955–1958, 1990.

[48] J. W. Ponder and D. A. Case, "Force fields for protein simulations," *Advances in Protein Chemistry*, vol. 66, pp. 27–85, 2003.

[49] V. Hornak, R. Abel, A. Okur *et al.*, "Comparison of multiple amber force fields and development of improved protein backbone parameters," *Proteins: Structure, Function and Genetics*, vol. 65, no. 3, pp. 712–725, 2006.

[50] D. J. Cole, M. C. Payne, G. Csányi *et al.*, "Development of a classical force field for the oxidized Si surface: Application to hydrophilic wafer bonding," *Journal of Chemical Physics*, vol. 127, no. 20, p. 204704, 2007.

[51] H. Z. Li, L. Li, Z. Y. Zhong *et al.*, "An accurate and efficient method to predict Y-NO bond homolysis bond dissociation energies," *Mathematical Problems in Engineering*, vol. 2013, pp. 1–10, 2013.

[52] J. Behler and M. Parrinello, "Generalized neural-network representation of high-dimensional potential-energy surfaces," *Physical Review Letters*, vol. 98, no. 14, pp. 1–4, 2007.

[53] T. Morawietz and J. Behler, "A density-functional theory-based neural network potential for water clusters including van der waals corrections," *Journal of Physical Chemistry A*, vol. 117, no. 32, pp. 7356–7366, 2013.

[54] J. Behler, R. Martonák, D. Donadio *et al.*, "Pressure-induced phase transitions in silicon studied by neural network-based metadynamics simu-

lations," *Physica Status Solidi (B) Basic Research*, vol. 245, no. 12, pp. 2618–2629, 2008.

[55] P. E. Dolgirev, I. A. Kruglov, and A. R. Oganov, "Machine learning scheme for fast extraction of chemically interpretable interatomic potentials," *AIP Advances*, vol. 6, no. 8, p. 085318, 2016.

[56] N. Artrith and A. Urban, "An implementation of artificial neural-network potentials for atomistic materials simulations: Performance for TiO2," *Computational Materials Science*, vol. 114, pp. 135–150, 2016.

[57] W. Tian, F. Meng, L. Liu *et al.*, "Lifetime prediction for organic coating under alternating hydrostatic pressure by artificial neural network," *Scientific Reports*, vol. 7, no. January, p. 40827, 2017.

[58] M. Rupp, A. Tkatchenko, K. R. Müller *et al.*, "Fast and accurate modeling of molecular atomization energies with machine learning," *Physical Review Letters*, vol. 108, no. 5, p. 058301, 2012.

[59] F. A. Faber, L. Hutchison, B. Huang *et al.*, "Prediction Errors of Molecular Machine Learning Models Lower than Hybrid DFT Error," *Journal of Chemical Theory and Computation*, vol. 13, no. 11, pp. 5255–5264, 2017.

[60] G. Montavon, M. Rupp, V. Gobre *et al.*, "Machine learning of molecu-

lar electronic properties in chemical compound space," *New Journal of Physics*, vol. 15, no. 9, p. 095003, 2013.

[61] A. Lopez-Bezanilla and O. A. Von Lilienfeld, "Modeling electronic quantum transport with machine learning," *Physical Review B - Condensed Matter and Materials Physics*, vol. 89, no. 23, p. 235411, 2014.

[62] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[63] Y. Jia, E. Shelhamer, J. Donahue *et al.*, "Caffe: Convolutional architecture for fast feature embedding," *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*, pp. 675–678, 2014.

[64] H. W. Lin, M. Tegmark, and D. Rolnick, "Why Does Deep and Cheap Learning Work So Well?" *Journal of Statistical Physics*, vol. 168, no. 6, pp. 1223–1247, 2017.

[65] S. Dieleman, J. De Fauw, and K. Kavukcuoglu, "Exploiting cyclic symmetry in convolutional neural networks," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, pp. 2799–2808, 2016.

[66] D. Pfau, J. S. Spencer, A. G. Alexander *et al.*, "Ab-initio solution of

the many-electron schrödinger equation with deep neural networks," p. 033429, 2019.

[67] N. Baker, S. Lee, F. Alexander *et al.*, "Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence," U.S. Department of Energy, Office of Science, Tech. Rep., 2019.

[68] L. Von Rueden, S. Mayer, K. Beckh *et al.*, "Informed machine learning - a taxonomy and survey of integrating knowledge into learning systems," *arXiv*, 2019.

[69] W. Samek, T. Wiegand, and K. R. Müller, "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models," *arXiv*, 2017.

[70] D. Baehrens, T. Schroeter, S. Harmeling *et al.*, "How to explain individual classification decisions," *Journal of Machine Learning Research*, vol. 11, pp. 1803–1831, 2010.

[71] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *2nd International Conference on Learning Representations, ICLR 2014 - Workshop Track Proceedings*, 2014.

[72] D. Erhan, Y. Bengio, A. Courville *et al.*, "Visualizing higher-layer features of a deep network," in *Bernoulli*, no. 1341, Montreal, 2009, pp. 1–13.

[73] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014.

[74] J. M. Hernández-Lobato and R. P. Adams, "Probabilistic backpropagation for scalable learning of Bayesian neural networks," *32nd International Conference on Machine Learning, ICML 2015*, vol. 3, pp. 1861–1869, 2015.

[75] C. Blundell, J. Cornebise, K. Kavukcuoglu *et al.*, "Weight uncertainty in neural networks," *32nd International Conference on Machine Learning, ICML 2015*, vol. 2, pp. 1613–1622, 2015.

[76] A. Kendall and Y. Gal, "What uncertainties do we need in Bayesian deep learning for computer vision?" Tech. Rep., 2017.

[77] M. Fortunato, C. Blundell, and O. Vinyals, "Bayesian recurrent neural networks," *arXiv*, 2017.

[78] L. Zhu and N. Laptev, "Deep and Confident Prediction for Time Series at Uber," *IEEE International Conference on Data Mining Workshops, ICDMW*, vol. 2017-Novem, pp. 103–110, 2017.

[79] N. Srivastava, G. Hinton, A. Krizhevsky *et al.*, "Dropout: A simple way to prevent neural networks from overfitting," Tech. Rep., 2014.

[80] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," *33rd International Conference on Machine Learning, ICML 2016*, vol. 3, pp. 1651–1660, 2016.

[81] Y. A. LeCun, L. Bottou, G. B. Orr *et al.*, "Efficient backprop," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer, Berlin, Heidelberg, 2012, vol. 7700 LECTU, pp. 9–48.

[82] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: Closing the generalization gap in large batch training of neural networks," Tech. Rep., 2017.

[83] J. Dean, G. S. Corrado, R. Monga *et al.*, "Large scale distributed deep networks," Tech. Rep., 2012.

[84] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *arXiv*, 2018.

[85] N. S. Keskar, J. Nocedal, P. T. P. Tang *et al.*, "On large-batch training for deep learning: Generalization gap and sharp minima," *5th International*

Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings, 2017.

[86] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," Tech. Rep., 2010.

[87] G. E. Hinton, "Neural Networks for Machine Learning Lecture 6a Overview of mini-batch gradient descent," Tech. Rep., 2012.

[88] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," Tech. Rep., 2012.

[89] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[90] S. Ruder, "An overview of gradient descent optimization algorithms," 2016.

[91] H. Larochelle, D. Erhan, A. Courville *et al.*, "An empirical evaluation of deep architectures on problems with many factors of variation," Tech. Rep., 2007.

[92] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," Tech. Rep., 2012.

[93] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," Tech. Rep., 2012.

[94] G. E. Hinton, N. Srivastava, A. Krizhevsky *et al.*, "Improving neural networks by preventing co-adaptation of feature detectors," Tech. Rep., 2012.

[95] L. Zhu and N. Laptev, "Deep and Confident Prediction for Time Series at Uber," *IEEE International Conference on Data Mining Workshops, ICDMW*, vol. 2017-Novem, pp. 103–110, 2017.

[96] J. Hron, A. G. De G Matthews, and Z. Ghahramani, "Variational Bayesian dropout: Pitfalls and fixes," Tech. Rep., 2018.

[97] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," Tech. Rep., 2016.

[98] A. Courville, I. Goodfellow, and Y. Bengio, *Deep learning*. MIT Press, 2016.

[99] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," Tech. Rep., 2010.

[100] Y. LeCun, B. Boser, J. Denker *et al.*, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Pro-*

cessing Systems, D. Touretzky, Ed., vol. 2. Morgan-Kaufmann, 1990, pp. 396–404.

[101] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Tech. Rep. 6, 2017.

[102] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," Tech. Rep. 4, 2017.

[103] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," Tech. Rep. 7, 2006.

[104] A. Van Den Oord, N. Kalchbrenner, O. Vinyals *et al.*, "Conditional image generation with PixelCNN decoders," *Advances in Neural Information Processing Systems*, pp. 4797–4805, 2016.

[105] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," Tech. Rep., 2014.

[106] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted Boltzmann machines for collaborative filtering," Tech. Rep., 2007.

[107] R. Salakhutdinov and G. Hinton, "Deep Boltzmann machines," Tech. Rep., 2009.

[108] M. Á. Carreira-Perpiñán and G. E. Hinton, "On contrastive divergence learning," Tech. Rep., 2005.

[109] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza *et al.*, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 3, no. January, pp. 2672–2680, 2014.

[110] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," pp. 1–7, 2014.

[111] T. Hinz, M. Fisher, O. Wang *et al.*, "Improved techniques for training single-image GANs," *arXiv*, no. Nips, pp. 1–9, 2020.

[112] V. Sandfort, K. Yan, P. J. Pickhardt *et al.*, "Data augmentation using generative adversarial networks (CycleGAN) to improve generalizability in CT segmentation tasks," *Scientific Reports*, vol. 9, no. 1, p. 16884, 2019.

[113] S. W. Huang, C. T. Lin, S. P. Chen *et al.*, "AugGAN: Cross domain adaptation with GAN-based data augmentation," Tech. Rep., 2018.

[114] M. Frid-Adar, E. Klang, M. Amitai *et al.*, "Synthetic data augmentation using GAN for improved liver lesion classification," in *Proceedings - International Symposium on Biomedical Imaging*, vol. 2018-April. IEEE Computer Society, may 2018, pp. 289–293.

[115] C. Bowles, L. Chen, R. Guerrero *et al.*, "GAN augmentation: Augmenting training data using generative adversarial networks," *arXiv*, 2018.

[116] C. Ledig, L. Theis, F. Huszár *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 105–114, 2017.

[117] J. Y. Zhu, T. Park, P. Isola *et al.*, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 2242–2251, 2017.

[118] T. Kim, M. Cha, H. Kim *et al.*, "Learning to discover cross-domain relations with generative adversarial networks," *34th International Conference on Machine Learning, ICML 2017*, vol. 4, pp. 2941–2949, 2017.

[119] H. Zhang, T. Xu, H. Li *et al.*, "StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 8, pp. 1947–1962, 2019.

[120] A. Barati Farimani, J. Gomes, and V. S. Pande, "Deep Learning the Physics of Transport Phenomena," *arXiv*, vol. 94305, 2017.

[121] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GaN," *arXiv*, 2017.

[122] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, sep 1998, vol. 9, no. 5.

[123] C. Berner, G. Brockman, B. Chan *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv*, 2019.

[124] Z. Zhang, H. Li, L. Zhang *et al.*, "Hierarchical reinforcement learning for multi-agent MOBA game," *arXiv*, 2019.

[125] O. Vinyals, I. Babuschkin, W. M. Czarnecki *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[126] D. Silver, T. Hubert, J. Schrittwieser *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[127] F. Agostinelli, S. McAleer, A. Shmakov *et al.*, "Solving the Rubik's cube with deep reinforcement learning and search," *Nature Machine Intelligence*, vol. 1, no. 8, pp. 356–363, 2019.

[128] I. Akkaya, M. Andrychowicz, M. Chociej *et al.*, "Solving Rubik's cube with a robot hand," *arXiv*, pp. 1–51, 2019.

[129] D. Hafner, T. Lillicrap, I. Fischer *et al.*, "Learning latent dynamics for planning from pixels," Tech. Rep., 2019.

[130] S. Bansal, R. Calandra, K. Chua *et al.*, "MBMF: Model-based priors for model-free reinforcement learning," Tech. Rep., 2017.

[131] V. Mnih, A. P. Badia, L. Mirza *et al.*, "Asynchronous methods for deep reinforcement learning," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, pp. 2850–2869, 2016.

[132] J. Schulman, F. Wolski, P. Dhariwal *et al.*, "Proximal policy optimization algorithms," *arXiv*, pp. 1–12, 2017.

[133] J. Schulman, S. Levine, P. Moritz *et al.*, "Trust region policy optimization," in *32nd International Conference on Machine Learning, ICML 2015*, vol. 3, 2015, pp. 1889–1897.

[134] T. P. Lillicrap, J. J. Hunt, A. Pritzel *et al.*, "Continuous control with deep reinforcement learning," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[135] T. Haarnoja, S. Ha, A. Zhou *et al.*, "Learning to walk via deep reinforcement learning," *arXiv*, 2018.

[136] T. Haarnoja, A. Zhou, K. Hartikainen *et al.*, "Soft Actor-Critic Algorithms and Applications," *arXiv*, 2018.

[137] T. Haarnoja, A. Zhou, P. Abbeel *et al.*, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *35th*

International Conference on Machine Learning, ICML 2018, vol. 5, pp. 2976–2989, 2018.

[138] S. Kakade and J. Langford, "Approximately Optimal Approximate Reinforcement Learning," in *Proceedings of the 19th International Conference on Machine Learning*, vol. 2, 2002, pp. 267–274.

[139] F. Brockherde, L. Vogt, L. Li *et al.*, "Bypassing the Kohn-Sham equations with machine learning," *Nature Communications*, vol. 8, no. 1, pp. 1–8, 2017.

[140] S. Segui, O. Pujol, and J. Vitria, "Learning to count with deep object features," Tech. Rep., 2015.

[141] C. Szegedy, W. Liu, Y. Jia *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June. IEEE, jun 2015, pp. 1–9.

[142] K. Mills, M. Spanner, and I. Tamblyn, "Deep learning and the Schrödinger equation," *Physical Review A*, vol. 96, no. 4, p. 042113, 2017.

[143] K. Ryczko, K. Mills, I. Luchak *et al.*, "Convolutional neural networks for atomistic systems," *Computational Materials Science*, vol. 149, pp. 134–142, 2018.

[144] K. Mills, K. Ryczko, I. Luchak *et al.*, "Extensive deep neural networks

for transferring small scale learning to large scale systems," *Chemical Science*, vol. 10, no. 15, pp. 4129–4140, 2019.

[145] H. Choubisa, M. Askerka, K. Ryczko *et al.*, "Crystal Site Feature Embedding Enables Exploration of Large Chemical Spaces," *Matter*, vol. 3, no. 2, pp. 433–448, 2020.