Lecture Note on Deep Learning and Quantum Many-Body Computation

Jin-Guo Liu, Shuo-Hui Li, and Lei Wang*

Institute of Physics, Chinese Academy of Sciences
Beijing 100190, China

November 23, 2018

Abstract

This note introduces deep learning from a computational quantum physicist's perspective. The focus is on deep learning's impacts to quantum many-body computation, and vice versa. The latest version of the note is at http://wangleiphy.github.io/. Please send comments, suggestions and corrections to the email address in below.

* wanglei@iphy.ac.cn

CONTENTS

INTRODUCTION

Deep Learning (DL) $\subset$ Machine Learning (ML) $\subset$ Artificial Intelligence (AI). Interestingly, DL is younger than ML; ML is younger than AI. This might be an indication that it is easier to make progress on a concrete subset of problems, even if you have a magnificent goal. See [1, 2] for an introduction and historical review of AI. ML is about finding out regularities in data and making use of them for fun and profit. Human is good at identifying patterns in data. The whole history of science can be attributed as searching for patten in Nature and summarizing them into physical/chemistry/biological laws. Those laws explain observed data, and more importantly, predict future observations. ML tries to automate such procedure with algorithms run on computers. Lastly, DL is a bunch of rebranded ML approaches involving neural networks, which were popular in 1980's under the name *connectionism*. An even more enlightening name to emphasize the modern core technologies is *differentiable programing*.

We are at the third wave of AI. There were two upsurges in 50's, and in 80's. In between, it is the so called AI winter. Part of the reasons for those winters were that the researchers made overly optimistic promises in their papers, proposals and talks, which failed to deliver later.

What is *really* different this time ? We have seen broad industrial successes of the AI technologies. Advances in the deep learning quickly get deployed from research papers into products. Training much larger models are possible now because computers run much faster and it is much easier to collect training data. Also, thanks to arXiv, Github, Twitter, Youtube etc, information spread at a much faster speed. So it is faster to iterate on other people's success and trigger interdisciplinary progresses. In sum, the keys behind this recent renaissance are

1. The emphasize of big data,

2. Modern computing hardware and software frameworks,

3. Representation learning.

Why are we learning these as physicists ? It is a game changing technique. It has changed computer vision, natural language processing, and robotics. Eventually, like the steam engine, electricity or

information technologies, it will change industry, business, our daily life, and scientific discovery.

This lecture note tries to bring to you the core ideas and techniques in deep learning from a physicist's perspective. We will explain what are the typical problems in deep learning and how does one solve them. We shall aim at a principled and unified approach to these topics, and emphasize their connections to quantum many-body computation.

Please be aware that

1. It is not magic. In fact, any sufficiently analyzed magic is indistinguishable from science. "No one really understands quantum mechanics", but this does not prevent us making all kinds of precise predictions about molecules and solids. Similar is true about AI, with a practical and scientific attitude you will understand subtle things like "artist style of a painting", at least compute and make use of it [3].

2. Physics background helps. With the mind set of a computational quantum physicist it is relatively easy to grasp deep learning. Since what you are facing are merely calculus, linear algebra, probability theory, and information theory. Moreover, you will find that many concepts and approaches in deep learning have extremely close relation to statistical and quantum physics.

3. Get your hands dirty. Try to understand things by deriving equations and programing them into computers. Good intuitions build on these laborious efforts.

That's it. Have fun!

2

DISCRIMINATIVE LEARNING

One particular class of tasks is to make predictions of properties of unseen data based on existing observations. You can understand this as function fitting $y = f(x)$, either for interpolation or for extrapolation. Alternatively, in a probabilistic language, the goal is to learn the conditional probability $p(y|x)$. We call x data, and y label. When the labels are discrete values, the task is called *classification*. While for continuous labels, this is called *regression*. You can find numerous approaches for these tasks at scikit-learn. However, the so called "no free lunch" theorem [4] states that no algorithm is better than any other if we average the performance over all possible data distribution. Well..., why should we prefer one approach over another one ? It is because we either explicitly or implicitly have certain prior knowledge about the typical problems we care about. The "deep learning" approach we focus on here is successful for natural images and languages. One reason is that the designed deep neural nets fit nicely into the symmetry, compositional nature, and correlation of the physical world. Another reason is a technical one, these deep learning algorithms run very efficiently on modern specialized hardwares such GPUs.

In general, there are four key components of the machine learning applications, *data*, *model*, *objective function* and *optimization*. With combination of each component, we will have vast different machine learning algorithms for different tasks.

2.1 DATA REPRESENTATION

For discriminative learning we have a dataset $\mathcal{D} = \{(x, y)\}$, which is a set of tuples of data and labels. Taking a probabilisitic view, one can think a dataset contains *samples* of certain probability distribution. This view is quite natural to those of us who generate training data using Monte Carlo simulation. For many deep learning applications, one also assume the data are independent and identically distributed (i.i.d.) samples from an unknown data generation probability given by the nature. It is important to be aware of the difference between having direct access to i.i.d. data or analytical expression of an *unnormalized* probability distribution (i.e. Boltzmann weight). This can lead to very different design choices of the learning algorithms.
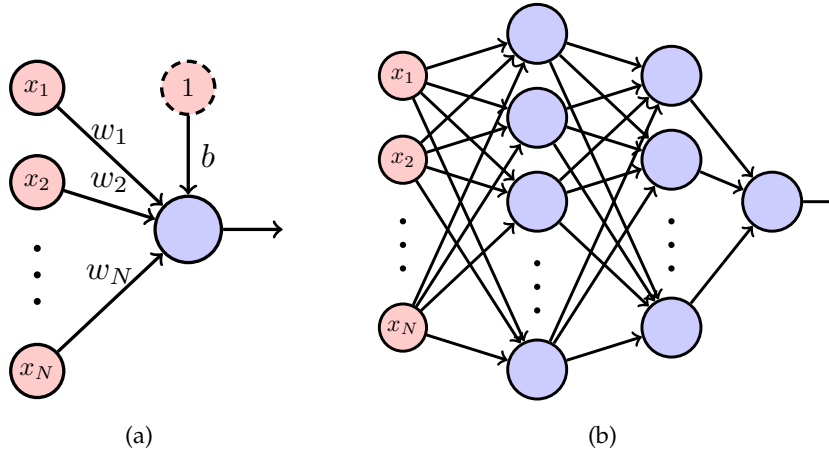
Data generation probability

Figure 1: (a) An artificial neuron and (b) an artificial neuron network.

Besides i.i.d, another fundamental requirement is that the data contains the key variation. For example, in the Atari game example by DeepMind, a single screencast of the video can not tell you the velocity of the spaceship [5]. While in the phase transition applications, it is actually fundamentally difficult to tell the phases apart from one Monte Carlo snapshot instead of an ensemble, especially the system is at criticality.

Even though the raw data is information complete in principe, we are free to manually prepare some additional features to help the machine to learn. In computer vision, people used to manually prepare features. Even AlphaGo (but not AlphaGo Zero) took human designed features from the board. Similarly, feature design is a key to the material and chemistry applications of the machine learning. For most of the material machine learning project, finding the right "descriptor" for learning is an important research topic. For many-body problems, general feature design based on symmetries of the physical problem is acceptable. Sometimes it is indeed helpful to build in the physical symmetries such as spin inversion or translational invariances in the input data or the network structure. However, it would be even more exciting if the neural network can automatically discover the *emerged* symmetry or relevant variables, since the defining feature of deep learning is to learn the right representation and avoid manual feature design.

To let computers learn, we should first present the dataset in an understandable format to them. In the simplest form, we can think a dataset as a two dimensional array of the size $(Batch, Feature)$, where each row is a sample, and each column is a feature dimension. Take an image dataset as an example, each image is reshaped into a row vector. While for quantum-many body problems, each row can be a snapshot sampled according to wavefunction, a quantum Monte

5

Carlo configuration etc. The label y can be understood as a single column vector of the length for regression. While for category labels, a standard way is to represent them in the one-hot encoding.

2.2 MODEL: ARTIFICIAL NEURAL NETWORKS

Connectionists believe that intelligence emerges from *connections* of simple building blocks. The biological inspired building block is called artificial neuron, shown in Fig. 1(a). The neuron multiplies weights to the input vector and add a bias, and then passes the results through an activation function. You can think the artificial neuron as a switch, which activates or not depending on the weighted sum of the inputs. An artificial neural network consists of many of artificial neurons connected into a network, see Fig. 1(b). This particular form of neural network is the also called feedforward neural network since all the connections has a direction. The data flows from left to the right, and gives the output from the finial layer. We denote the input data as $x^0 = x$. It flows through the network and gives rise to $x^{\ell=1,2,\cdots}$, eventually the finial output. The output of a neural network does not need to be a scalar. Having multiple number of output when dealing with categorical labels.

A neural network expresses complex multi-variables functions using nested transformations. It is a universal function approximator in the sense that even with a single layer of the hidden neurons, it can approximate any continuous function to arbitrary accuracy by increasing the number of hidden neurons [6, 7]. However, this nonconstructive mathematical theorem does not tell us how to construct appropriate neural network architecture for particular problem at hand. It does not tell us how to determine the parameter of a neural network even with a given structure, either. Based on engineering practices, people find out that it is more rewarding to increase the depth of the neural network, hence the name "deep learning". Surprisingly, in a trained neural network neurons in the shallow layer care more about low level features such as edge information, while the neurons in the deep layers care more about global features such as shape. This reminds physicists renormalization group flow [8, 9].

Classical texts [10, 11] contain many toy examples which are still enlightening today. One can get familiar with artificial neural networks by analytically working out some toy problems.

Exercise 1 (Parity Problem). *Construct a neural network to detect whether the input bit string contains even or odd number of 1's. This is the famous XOR problem if the input length is two.*

It is better to zoom out from individual neurons and take a modular perspective of the neural network. Typically the data flow in a feedforward neural network alternates between linear affine transformation and element-wise nonlinear layers. One can see that these are

in general two non-commuting operations, and both are necessary ingredient to model nontrivial functions. Composition of two linear transformation is still a linear transformation. While element-wise nonlinear transformation can not extracts correlation between input variables. Thus, the general pattern of alternating between linear and nonlinear units also applies to more complicated neutral networks.

Table 1 summarized typical nonlinear activation functions used in neural networks. Since they have different range of the output, they are used for different purposes. For example, Linear, ReLU and soft-plus for regression problems, sigmoid and softmax for classification problems. We will see that output type ties closely to the cost functions in the probabilistic interpretation.
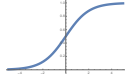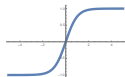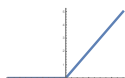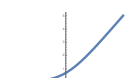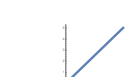
The basic linear transformation of a vanilla neural network performs an affine transformation to the input data

$$x_\nu^{\ell+1} = \sum_\mu x_\mu^\ell W_{\mu,\nu} + b_\nu, \tag{1}$$

where $W_{\mu,\nu}$ and b_ν are the parameters. Since this layer mixes components, it is sometimes called the dense layer. Afterwards, one pass the output to an element-wise nonlinear transformation, which is denoted as the activation function of the neurons.

To be more explicit about the spatial structure of the input data, image data is represented as four dimensional tensors. For example $(Batch, Channel, Height, Width)$ in some of the modern deep learn-

Table 1: Popular activation functions. Except Softmax, these functions apply element-wise to the variables.

Name	Function	Output range	Graph
Sigmoid	$\sigma(z) = (1 + e^{-z})^{-1}$	$(0, 1)$	
Tanh	$\tanh(z) = 2\sigma(2z) - 1$	$(-1, 1)$	
ReLU	$\max(z, 0)$	$[0, \infty)$	
Softplus	$\ln(1 + e^z)$	$(0, \infty)$	
ELU	$\begin{cases} \alpha(e^z - 1), & \text{if } z < 0 \\ z, & \text{otherwise} \end{cases}$	$(-\alpha, \infty)$	
Softmax	$e^{z_i} / \sum_i e^{z_i}$	$(0, 1)$	

ing frameworks, where "channel" denotes RGB channels of the input. For each sample, the convolutional operation performs the following operation (omitting the batch index)

$$x_{v,i,j}^{\ell+1} = \sum_{\mu} \sum_{m,n} x_{\mu,i+m,j+n}^{\ell} W_{\mu,v,m,n} + b_v, \tag{2}$$

where the parameters are $W_{\mu,v,m,n}$ and b_v. The convolutional kernel is a four dimensional tensor, which performs the matrix-vector multiplication in the channel space μ, v and computes the cross correlation in the spatial dimension i, j. The number of learnable parameters of each convolutional kernel does not scale with the spatial size of the input. If one requires the summed indices do not exceed the size of the input, the output of Eq. (2) will have different spatial shape with the input, which is denoted as "valid" padding. Alternatively, one can also pad zeros around the original input, such that the spatial size is the same as the input, which is denoted as the "same" padding. In fact, for many of the physical applications, one tempts to have a "periodic" padding. Moreover, one can generalize Eq. (2) so the filter have different stride and dilation factors.

Exercise 2 (Padding and Kernel Size in ConvNet). *(a) Convince yourself that with 3×3 convolutional kernel and padding 1 the spatial size of output remains the same. (b) Think about how to implement periodic padding. (c) Some times people use 1×1 convolutional kernel, please explain why it makes sense.*

After the convolutional layer, one typically perform downsampling, such as taking the maximum or average over a spatial region. This operation is denoted as pooling. Pooling is also a linear transformation. The idea of convolution + pooling is to identify translational invariant features in the input, then response to these features. Standard neural network architectures consists many layers of convolutional layer, pooling layers to extract invariant features, and finally have a few fully connected layers to perform the classification. With latest ideas in neural network architecture design such as ResNet [12] and highway, one can successfully train neural networks more than hundreds layers.

Typical network layout

Overall, it is important to put the prior knowledge into the neural network structure. The hierarchical structure of a deep neural network works since it fits the compositional nature of the world. Therefore, lower layers extract fine features while deeper layers cares more about the coarse grained features of the data. Moreover, convolutional neural network respects the translational invariance of most image data, in which the local receptive field with shared weight scan through the images and search for appearance of a common feature.

There are three levels of understanding of a neural network. First, one can view it as a function approximator $y = f(x)$. There is nothing wrong about such understanding, however it will severely limits

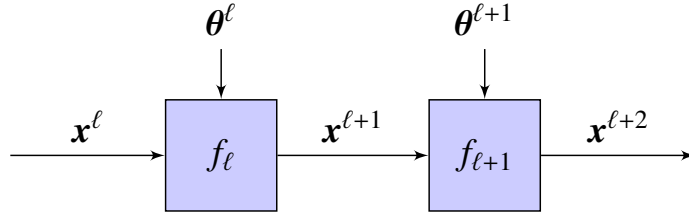Figure 2: Layers of a feedforward network. Each layer is a function $x^{\ell+1} = f_\ell(\theta^\ell, x^\ell)$.

one's imagination when it comes to applications. Next, one views it from a probabilistic perspective. For example, the neural net expresses the conditional probability of the output given the input. Or, it transforms the probability distribution of the input data and output data. We will see many of the generative models are doing exactly this. Finally, one can view the neural network as information processing devices. Drawing analogies to the tensor networks and even quantum circuits can be as fruitful as making connections to neuroscience. For example, it could be quite instructive to use information theoretical measures to guide the structure design and learning of the neural net, like we did for tensor networks.

2.3 COST FUNCTION

Probabilistic interpretation of the neural network provides a unified view for designing the cost functions. Imaging the output of the neural network parametrizes a conditional probability distribution of the predicted label based on the data. The goal is to minimize the negative log-likelihood averaged over the training dataset

$$\mathcal{L} = -\frac{1}{|\mathcal{D}|} \sum_{(x,y) \in \mathcal{D}} \ln\left[p_\theta(y|x)\right]. \tag{3}$$

For example, when dealing with real valued labels we assume the neural network outputs the mean of a Gaussian distribution $p_\theta(y|x) = \mathcal{N}(y; \hat{y}(x, \theta), \mathbf{1})$. The cost function will then be the familiar mean-squared error (MSE). While for binary classification problem we can assume the neural network outputs mean of a Bernoulli distribution $p_\theta(y|x) = [\hat{y}(x, \theta)]^y[1 - \hat{y}(x, \theta)]^{(1-y)}$. To make sure the output of the neural network is between 0 and 1 one can use the sigmoid output. More generally, for categorical output one can use the softmax layer such that the finial output will be normalized probability.

The sum and product rule of probabilities

$$p(A) = \sum_B p(A, B), \qquad (4)$$

$$p(A, B) = p(B|A)p(A), \qquad (5)$$

where $p(A, B)$ is the joint probability, $p(B|A)$ is the conditional probability of B given A. The Bayes rule reads

$$p(B|A) = \frac{p(A|B)p(B)}{p(A)}, \qquad (6)$$

also known as "Posterior $= \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$".

A standard way to prevent overfitting is to add a regularization term to the cost function,

$$\mathcal{L} \leftarrow \mathcal{L} + \lambda \Omega(\boldsymbol{\theta}). \qquad (7)$$

For example, the L^2 norm of all weights $\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||^2$. The presence of such regularization term prevents the weight from growing to large values. In the practical gradient descend update, the L^2 norm regularization leads to decay of the weight, so such regularization is also called weight decay. Another popular form of regularization is the L^1 norm, which favors sparse solutions.

A principled way to introduce the regularization terms is the Maximum a posteriori (MAP) estimation of the Bayesian statistics. We view the parameter $\boldsymbol{\theta}$ also as the stochastic variable to be estimated. Thus

$$\arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{x}, y) = \arg\max_{\boldsymbol{\theta}} \left[\ln p(y|\boldsymbol{x}, \boldsymbol{\theta}) + \ln p(\boldsymbol{x}, \boldsymbol{\theta}) \right]. \qquad (8)$$

One sees that the regularization terms can be understood as the prior of the parameters.

Another form of the regularization is to include randomness in the training and average over the randomness at testing time. For example, the drop out regularization randomly masks out output of some neurons in the training. This ensures that the neural network can not count on certain particular neuron output to make the prediction. In the testing time, all the neurons are used but their outputs are reweighed with a factor related to the drop out probability. This is

Dropout, Data argumentation, Transfer learning

similar to taking a vote from an ensemble. A related regularization approach is data argumentation. You can make small modifications to the training set (shift, rotation, jittering) to artificially enlarge the training set. The thing is that label should be unchanged for irrelevant perturbations, so the neural network is forced to learn about the more robust mappings from the pixels to the label. This is particularly useful when the dataset is small. Finally, generalization via transfer learning. People train a neural network on a much larger dataset and take the resulting network and fine tune the last few layers for special tasks.

2.4 OPTIMIZATION

Finally, given the data, the neural network model and a suitable cost function, we'd like to learn the model from data by performing optimization over the cost function. There are many optimization tricks you can use, random guessing, simulate annealing, evolution strategies, or whatever you can think of. A particular powerful algorithms is using the gradient information of the loss with respect to the network parameters.

2.4.1 *Back Propagation*

A key algorithm component of the deep neural network is the back propagation algorithm, which computes the gradient of the neural network output w.r.t. its parameters in an efficient way. This is the core algorithm run under the hood of many successful industrial applications. The idea is simply to apply the chain rule iteratively. A modular and graphical representation called computation graph is useful for dealing with increasingly complex modern neural network structures. You can think the computation graph as "Feynman diagrams" for deep learning. Another analogy, graphical notations are used extensively for visualizing contractions and quantum entanglement in tensor networks. When using neural nets to study physics, one should ask similar questions: what are the meaning of all these connections ?

As shown in Fig. 2, we would like to compute the gradient of the loss function with respect to the neural network parameters efficiently

BackProp is neither symbolic differentiation nor numerical finite difference differentiation

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}^\ell} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^{\ell+1}} \frac{\partial \boldsymbol{x}^{\ell+1}}{\partial \boldsymbol{\theta}^\ell} = \delta^{\ell+1} \frac{\partial \boldsymbol{x}^{\ell+1}}{\partial \boldsymbol{\theta}^\ell} \tag{9}$$

$$\delta^\ell = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^\ell} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^{\ell+1}} \frac{\partial \boldsymbol{x}^{\ell+1}}{\partial \boldsymbol{x}^\ell} = \delta^{\ell+1} \frac{\partial \boldsymbol{x}^{\ell+1}}{\partial \boldsymbol{x}^\ell} \tag{10}$$

Equation (10) is the key of the back propagation algorithm, in which it connects the gradient of the loss with respect to the output of each

Back Propagation is nothing but "reverse mode automatic differentiation" applied to neural network. See [13] for survey of automatic differentiation techniques applied to machine learning.

layer. In practice, we compute the l.h.s. using information on the r.h.s., hence the name "back" propagation. Note that $\frac{\partial x^{\ell+1}}{\partial x^{\ell}}$ is the Jacobian matrix of each layer of the size $(output, input)$. One sees that back propagation involves iterative matrix-vector multiplications (which become matrix-matrix multiplications if you consider batch dimension). One should already be caution with the possible numerical issue such as vanishing or exploding gradient. The Jacobian of element-wise layer is a diagonal matrix.

The steps of the back propagation algorithm is summarized in Algorithm 1. In two passes one evaluate the gradient with respect all parameters up to the machine precision. This scaling of computational cost is linear with respect to the number of parameters of the neural network. And the memory cost is linear with respect to the depth of the network since in the forward pass one caches intermediate results for efficient calculation of the Jacobian-Vector product in the backward pass.

One can trade between computational and memory cost by saving intermediate results less frequently. This is called "checkpoint". And for reversible network one can reduce the memory cost more aggressively to be independent of the network depth [14, 15].

Algorithm 1 Computing gradient of the loss function with respect to the neural network parameters using back propagation.

Require: Loss function for the input data x
Require: Neural network with parameters θ^{ℓ}
Ensure: $\frac{\partial \mathcal{L}}{\partial \theta^{\ell}}$ for $\ell = 0, \dots, L-1$
$\quad x^{\ell=0} = x$
$\quad$**for** $\ell = 0, \cdots, L-1$ **do** $\qquad\qquad\qquad$ ▷ Feedforward pass
$\quad\quad x^{\ell+1} = f_{\ell}(x^{\ell}, \theta^{\ell})$
$\quad$**end for**
$\quad$Compute $\delta^L = \frac{\partial \mathcal{L}}{\partial x^L}$ for the last layer
$\quad$**for** $\ell = L-1, \cdots, 0$ **do** $\qquad\qquad\qquad$ ▷ Backward pass
$\quad\quad$Evaluate $\frac{\partial \mathcal{L}}{\partial \theta^{\ell}} = \delta^{\ell+1} \frac{\partial x^{\ell+1}}{\partial \theta^{\ell}}$ $\qquad\qquad$ ▷ Eq. (9)
$\quad\quad$Evaluate $\delta^{\ell} = \delta^{\ell+1} \frac{\partial x^{\ell+1}}{\partial x^{\ell}}$ $\qquad\qquad\qquad$ ▷ Eq. (10)
$\quad$**end for**

Exercise 3 (Gradient of input data). *In Algorithm 1 where do you get the gradient of the loss with respect to the input data ? It is a useful quantity for adversarial training [16], deep dream [17] and style transfer [3].*

> **Example**

A dense layer consists of a affined transformation followed by an elementweise nonlinearity. We can view them as two sequential layers

$$
\begin{aligned}
x_\nu^{\ell+1} &= x_\mu^\ell W_{\mu\nu} + b_\nu, &(11)\\
x_\nu^{\ell+2} &= \sigma(x_\nu^{\ell+1}). &(12)
\end{aligned}
$$

We can back propagate the gradient information using Jacobian of each layer

$$
\begin{aligned}
\delta_\mu^\ell &= \delta_\nu^{\ell+1} W_{\mu\nu}, &(13)\\
\delta_\nu^{\ell+1} &= \delta_\nu^{\ell+2} \sigma'(x_\nu^{\ell+1}), &(14)
\end{aligned}
$$

where Eq. (14) involves elementwise multiplication of vectors, also known as the Hadamard product. And the gradient with respect to the parameters are

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W_{\mu\nu}} &= \delta_\nu^{\ell+1} x_\mu^\ell, &(15)\\[2mm]
\frac{\partial \mathcal{L}}{\partial b_\nu} &= \delta_\nu^{\ell+1}, &(16)
\end{aligned}
$$

where Eq. (15) involves outer product of vectors.

Each unit of the back propagation can be understood in a modular way. When transverse through the network graph, each module only takes care of the Jacobian-Vector product locally, and later we can back propagate the gradient information. Note one can control the fine grained resolution when define each module. Each block can be an elementary math function, a layer, or even a neural network itself. Developing a modular thinking greatly helps when one builds up more complicated projects.

BackProp is modular

2.4.2 *Gradient Descend*

After evaluation of the gradient, one can perform the gradient decent update of the parameters

$$
\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}, \tag{17}
$$

where $\eta > 0$ is the so called learning rate. Notice that in the Eq. (3) there is a summation over the training set, which can be as large as million for large dataset. Faithfully going through the whole dataset for an evaluation of the gradient can be time consuming. One can

stochastically draw a "mini-batch" of samples from the dataset to estimate the gradient. Sampling the gradient introduced noises in the optimization, which is not necessarily a bad idea since fluctuation may bring us out of local minimum. In the extreme case where one estimates the gradient use only one sample, it is called stochastic gradient descend (SGD). However, this way of computing is very unfriendly to modern computing hardware (caching and vectorization), so typically one still uses a bit larger batch size (say 64) to speed up the calculation. A typical flowchart of training is summarized in Algorithm 2. One random shuffles the training dataset, and updates the parameters on many mini-batches. It is called one epoch after one has gone through the whole dataset on average. Typically training of a deep neural network can take thousands of epochs.

Algorithm 2 Train a neural network using SGD.

Require: Training dataset $\mathcal{T} = \{(x, y)\}$
Require: A loss function
Ensure: Neural network parameters θ
  while stop criterion not met **do**
    for $b = 0, 1, \ldots, |\mathcal{T}|/|\mathcal{D}| - 1$ **do**
      Sample a minibatch of training data $\mathcal{D}$
      Evaluate gradient of the loss using Backprop   ▷ Eq. (3)
      Update θ ▷ Eq. (17)
    end for
  end while

Over years, machine learning practitioners have developed more sophisticated optimization schemes than SGD. Some concepts are worth knowing besides using them as black box optimizers. Momentum means that we keep mix some of the gradient of last step. This will keep the parameter moving when the gradient is small. While if the cost function surface has a narrow valley, this will damp the oscillation perpendicular to the steepest direction [18]

Beyond SGD

$$
\begin{aligned}
v &= \alpha v - \frac{\partial \mathcal{L}}{\partial \theta}, & (18) \\
\theta &= \theta + v. & (19)
\end{aligned}
$$

Adaptive learning rate means that one uses different learning rate for various parameters at different learning stages, including RMPprop, Adagrad, Adam etc. These optimizers are all first order method so they scale to billions of network parameters. Moreover, it is even possible to use the information of second order gradient. For example, L-BFGS algorithm is suitable if you can afford to use the whole batch to evaluate the gradient since the method is more sensitive to the sampling noise.

A typical difficulty with training deep neural nets is the gradient

Unsupervised pretraining

vanishing/exploding problem. Finding a good initial values of the network parameters can somehow alleviate such problem. Historically, people used unsupervised feature learning to support supervised learning by providing initial weights. Now, with progresses in activations functions (ReLU) and network architecture (BatchNorm [19], ResNet [12]), pure gradient based optimization is unimaginably successful and unsupervised pretraining became unnecessary (or, unfashionable) in training deep neural network.

Having said so much about optimization, it is important to emphasize a crucial point in neural network training: it is not at all an optimization problem. In fact our goal is NOT to obtain the global minimum of the loss function Eq. (3). Since the loss function is just a surrogate function evaluated on the finite training data. Achieving the global minimum of the surrogate does not mean that the model will generalize well. Since our ultimate goal is to make the predictions on unseen data, we can divide the dataset into training, validation and test sets. We optimize the neural net using the gradient information computed on the training dataset and monitor the loss on the validation data. Typically, at certain point the loss function computed on the validation dataset will increase, although it is still decreasing on the training data. We should stop the training at this point to prevent overfitting, which is called early stoping. Moreover, one can tune the so called hyperparameters (e.g. batch size, learning rate etc) based on the performance on the validation set. Finally, one can report the actual performance on the test set, which is used only once at the very end.

Learning is NOT optimization

One key problem in training the neural network is what should one do if the performance is not good. If the neural network overfits, one can try to increase the regularization, for example, increase data size or perform data argumentation, increase the regularization strength or using drop out. Or one can decrease the model complexity. While if the model underfits, beside making sure that the training data is indeed representative and following i.i.d, one should increase the model complexity. However, this does not always guarantee better performance as the optimization might be the problem. In this case, one can tune the hyperparameters in the optimization, or use better optimization methods and better initialization. Overall, when the model has the right inductive bias one can alleviate the burden of the optimization. So, overall, designing better network architecture is more important than parameter turning.

2.5 UNDERSTANDING, VISUALIZATION AND APPLICATIONS BEYOND CLASSIFICATION

There are strong motivation of visualizing and understanding what is going on in deep neural networks. For that, one can visualize

Visualize weights, features, activations

the weight of the filter, thus to know what are the features they are looking for. Or, one can view the output from the layer before the classifier as features, and feed all images into the neural network and search for nearest neighbors or perform dimensionality reduction on these features for understanding. Or, one can do gradient ascent to generate a synthetic image that maximally activates a neuron by performing Backprop. Therefore one knows what are the features that the particular neuron is looking for in the original image [20].

Neural arts

The neural networks can actually acquire generative ability even though they are trained for pattern recognition. A well trained deep neural network is a great resources for creative tasks. For example, one can have some fun with neural arts. Deep dream amplifies existing features in the image [17]. Neural style transfer = Texture synthesis (style) + Feature reconstruction (content) [3].

Adversarial samples

Finally, we have mentioned that smaller perturbation should not change the label when doing the data argumentation. However, this is not always true. One can create "adversarial samples" by deliberately making small updates to the image to confuse the neural net [16]. Although some of these adversarial samples looks identical to human, the neural nets can make wrong predictions. Existence of adversarial samples have at least two implications: 1. researches on AI safety; 2. there are crucial differences in the neural nets and human brain. Some physicists interpret adversarial attacks as finding the softest mode in a Stat-Mech system.

GENERATIVE MODELING

3.1 UNSUPERVISED PROBABILISTIC MODELING

The previous chapter on discriminative learning deals with predictions, i.e. finding out the function mapping $y = f(x)$ or modeling the conditional probability $p(y|x)$. These techniques are hugely successful in industrial applications. However, as can be seen from the discussions, they are far from being intelligent. Feynman has a famous quote "what I can not create, I do not understand". Indeed, having the ability to create new instances is a good indication of deeper understanding and higher level of intelligence. Generative modeling is the forefront of deep learning research [21].

Generative modeling aims to learn about the joint probability distribution of the data and label $p(x, y)$. With a generative model at hand, one can support the discriminative task through the Bayes formula $p(y|x) = p(x, y)/p(x)$, where $p(x) = \sum_y p(x, y)$. Moreover, one can generate new samples conditioned on its label $p(x|y) = p(x, y)/p(y)$. The generative models are also useful to support semi-supervised learning and reinforcement learning.

To wrap up, the goal of generative modeling is to *represent*, *learn* and *sample* from such high dimensional probability distributions. In the following, we will focus on learning $p(x)$. So the dataset would be a set of unlabelled data $\mathcal{D} = \{x\}$. Learning joint probability distribution is similar.

First, let us review a key concept in the information theory, the Kullback-Leibler (KL) divergence

$$\mathbb{KL}(\pi||p) = \sum_x \pi(x) \ln \left[\frac{\pi(x)}{p(x)} \right], \tag{20}$$

which measures the distance between two probability distributions. We have $\mathbb{KL} \geq 0$ due to the Jensen inequality. The equality is achieved only when the two distributions are identity. The KL-divergence is not symmetric with respect to its arguments. $\mathbb{KL}(\pi||p)$ places high probability in p anywhere the data probability π is high, while $\mathbb{KL}(p||\pi)$ places low probability where the data probability π is low [22].

The Jensen inequality [23] states that for convex $\smile$ functions f

$$\langle f(x) \rangle \geq f(\langle x \rangle). \tag{21}$$

Examples of convex $\smile$ functions $f(x) = x^2, e^x, e^{-x}, -\ln x, x \ln x$.

Introducing Shannon entropy $\mathbb{H}(\pi) = -\sum_x \pi(x) \ln \pi(x)$ and cross-entropy $\mathbb{H}(\pi, p) = -\sum_x \pi(x) \ln p(x)$, one sees that $\mathbb{KL}(\pi||p) = \mathbb{H}(\pi, p) - \mathbb{H}(\pi)$. Minimization of the KL-divergence is then equivalent to minimization of the cross-entropy since only it depends on the to-be-optimized parameters. In typical DL applications one only has i.i.d. samples from the target probability distribution $\pi(x)$, so one replaces it with the empirical estimation $\pi(x) = \frac{1}{|\mathcal{D}|} \sum_{x' \in \mathcal{D}} \delta(x - x')$. The cross entropy then turns out to be the negative log-likelihood (NLL) we met in the last chapter

$$\mathcal{L} = -\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \ln[p_\theta(x)]. \tag{22}$$

Minimizing the NLL is a prominent (but not the only) way to train generative models, also known as Maximum Likelihood Estimation (MLE). The Eq. (22) appears to be a minor change compared to the discriminative task Eq. (3). However it causes huge challenges to change the conditional probability to probability function in the cost function. How to represent and learn such high dimensional probability distributions with the intractable normalization factor ? How could we marginalize and sample from such high dimensional probability distributions ? We will see that physicists have a lot to say about these problems since they love high dimensional probability, Monte Carlo methods and mean-field approaches. In fact, generative modeling has close relation to many problems in statistical and quantum physics, such as inverse statistical problems, modeling a quantum state and quantum state tomography.

Idea 1: use a neural net to represent $p(x)$, but how to normalize ? how to sample ? Idea 2: use a neural net to transform simple prior z to complex data x, but what is the likelihood ? How to actually learn ?

Exercise 4 (Positivity of NLL). *Show that the NLL is positive for probability distributions of discrete variables. What about probability densities of continuous variables ?*

3.2 GENERATIVE MODEL ZOO

This section we review several representative generative models. The key idea is to impose certain structural prior in the probability distribution. Each model has its own strengths and weakness. Exploring new approaches or combining the existing ones is an active research field in deep learning, with some ideas coming from physics.

3.2.1 *Boltzmann Machines*

As a prominent statistical physics inspired approach, the Boltzmann Machines (BM) model the probability as a Boltzmann distribution

$$p(\boldsymbol{x}) = \frac{e^{-E(\boldsymbol{x})}}{\mathcal{Z}}, \tag{23}$$

where $E(\boldsymbol{x})$ is an energy function and $\mathcal{Z}$, the partition function, is a normalization factor. The task of probabilistic modeling is then translated into designing and learning of the energy function to model observed data. For binary data, this is identical to the so called inverse Ising problem. Exploiting the maximum log-likelihood estimation, the gradient of Eq. (22) is

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left\langle \frac{\partial E(\boldsymbol{x})}{\partial \boldsymbol{\theta}} \right\rangle_{\boldsymbol{x} \sim \mathcal{D}} - \left\langle \frac{\partial E(\boldsymbol{x})}{\partial \boldsymbol{\theta}} \right\rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})}. \tag{24}$$

The two terms are called positive and negative phase respectively. Intuitively, the positive phase tries to push down the energy of the observed data, therefore increases the model probability on the observed data. While the negative phase tries to push up the energy on samples drawn from the model, therefore to make the model probability more evenly distributed.

Example

Consider a concrete example of the energy model $E = -\frac{1}{2}W_{ij}x_ix_j$, the gradient Eq. (24) can be simply evaluated. And the gradient descent update Eq. (17)

$$W_{ij} = W_{ij} + \eta \left(\langle x_ix_j \rangle_{\boldsymbol{x} \sim \mathcal{D}} - \langle x_ix_j \rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})} \right). \tag{25}$$

The physical meaning of such update is quite appealing: one compares the correlation on the dataset and on the model, then strengthen or weaken the coupling accordingly.

The positive phase are quite straightforward to estimate by simply sampling batches from the dataset. While the negative phase typically involves the Markov chain Monte Carlo (MCMC) sampling. It can be very expensive to thermalize the Markov chain at each gradient evaluation step. The contrastive divergence (CD) algorithm [24] initialize the chain with a sample draw from the dataset and run the Markov chain only k steps. The reasoning is that if the BM has learned the probability well, then the model probability $p(\boldsymbol{x})$ resembles the one of the dataset anyway. Furthermore, the persistent CD [25] algorithm use the sample from last step to initialize the Monte Carlo chain. The
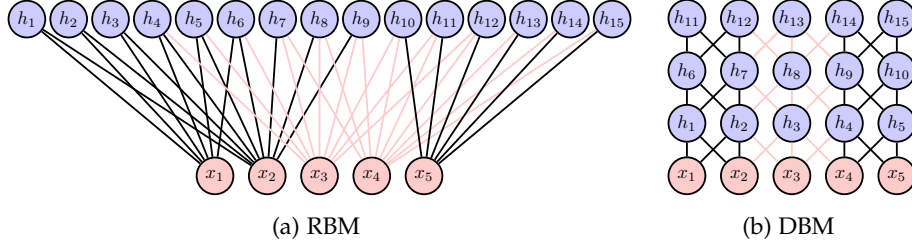
(a) RBM   (b) DBM

Figure 3: RBM and DBM with the same number of neurons and connections. Information theoretical consideration shows that the DBM can potentially capture patterns that are impossible for the RBM [26].

logic being that in the gradient descent update to the model is small anyway, so accumulation of the Monte Carlo samples helps mixing. In practice, one run a batch of the Monte Carlo chain in parallel to estimate the expected value of the negative phase.

Exercise 5 (Mind the Gradient). *Define* $\Delta = \langle E(x) \rangle_{x \sim \mathcal{D}} - \langle E(x) \rangle_{x \sim p(x)}$. *How is its gradient with respect to θ related to Eq. (24) ?*

To increase the representational power of the model, one can introduce hidden variables in the energy function and marginalize them to obtain the model probability distribution

$$p(x) = \frac{1}{\mathcal{Z}} \sum_h e^{-E(x,h)}. \tag{26}$$

This is equivalent to say that $E(x) = -\ln \sum_h e^{-E(x,h)}$ in Eq. (23), which can be quite complex even for simple joint energy function $E(x, h)$. Differentiating the equation, we have

$$\frac{\partial E(x)}{\partial \theta} = \frac{\sum_h e^{-E(x,h)} \frac{\partial E(x,h)}{\partial \theta}}{\sum_h e^{-E(x,h)}} = \sum_h p(h|x) \frac{\partial E(x,h)}{\partial \theta}, \tag{27}$$

Therefore, in the presence of the hidden variables the gradient in Eq. (24) becomes

$$\frac{\partial \mathcal{L}}{\partial \theta} = \left\langle \frac{\partial E(x,h)}{\partial \theta} \right\rangle_{x \sim \mathcal{D}, h \sim p(h|x)} - \left\langle \frac{\partial E(x,h)}{\partial \theta} \right\rangle_{(x,h) \sim p(x,h)}, \tag{28}$$

which remains simple and elegant. However, the downside of introducing the hidden variables is that one needs even to perform expensive MCMC for the positive phase. An alternative approach is to use the mean-field approximation to evaluate these expectations approximately.

The restricted Boltzmann Machine (RBM) aims to have a balanced expressibility and learnability. The energy function reads

$$E(x, h) = -\sum_i a_i x_i - \sum_j b_j h_j - \sum_{i,j} x_i W_{ij} h_j. \tag{29}$$

Since the RBM is defined on a bipartite graph shown in Fig. 3(a), its conditional probability distribution factorizes $p(\boldsymbol{h}|\boldsymbol{x}) = \prod_j p(h_j|\boldsymbol{x})$ and $p(\boldsymbol{x}|\boldsymbol{h}) = \prod_i p(x_i|\boldsymbol{h})$, where

$$p(h_j = 1|\boldsymbol{x}) = \sigma\left(\sum_i x_i W_{ij} + b_j\right), \tag{30}$$

$$p(x_i = 1|\boldsymbol{h}) = \sigma\left(\sum_j W_{ij} h_j + a_i\right). \tag{31}$$

This means that given the visible units we can directly sample the hidden units in parallel, vice versa. Sampling back and force between the visible and hidden units is called block Gibbs sampling. Such sampling approach appears to be efficient, but it is not. The visible and hidden features tend to lock to each other for many steps in the sampling. In the end, the block Gibbs sampling is still a form of MCMC which in general suffers from long autocorrelation time and transition between modes.

Despite of appealing theory and historic importance, BM is now out of fashion in industrial applications due to limitations in its learning and sampling efficiency.

Info

For an RBM, one can actually trace out the hidden units in the Eq. (23) analytically and obtain

$$E(\boldsymbol{x}) = -\sum_i a_i x_i - \sum_j \ln(1 + e^{\sum_i x_i W_{ij} + b_j}). \tag{32}$$

This can be viewed as a Boltzmann Machine with fully visible units whose energy function has a softplus interaction. Using Eq. (24) and Eq. (32) one can directly obtain

$$-\frac{\partial \mathcal{L}}{\partial a_i} = \langle x_i \rangle_{\boldsymbol{x} \sim \mathcal{D}} - \langle x_i \rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})}, \tag{33}$$

$$-\frac{\partial \mathcal{L}}{\partial b_j} = \langle p(h_j = 1|\boldsymbol{x}) \rangle_{\boldsymbol{x} \sim \mathcal{D}} - \langle p(h_j = 1|\boldsymbol{x}) \rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})}, \tag{34}$$

$$-\frac{\partial \mathcal{L}}{\partial W_{ij}} = \langle x_i p(h_j = 1|\boldsymbol{x}) \rangle_{\boldsymbol{x} \sim \mathcal{D}} - \langle x_i p(h_j = 1|\boldsymbol{x}) \rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})}. \tag{35}$$

On see that the gradient information is related to the difference between correlations computed on the dataset and the model.

Exercise 6 (Improved Estimators). *To reconcile Eq. (28) and Eqs. (33-35), please convince yourself that $\langle x_i p(h_j = 1|\boldsymbol{x}) \rangle_{\boldsymbol{x} \sim \mathcal{D}} = \langle x_i h_j \rangle_{\boldsymbol{x} \sim \mathcal{D}, \boldsymbol{h} \sim p(\boldsymbol{h}|\boldsymbol{x})}$ and $\langle x_i p(h_j = 1|\boldsymbol{x}) \rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})} = \langle x_i h_j \rangle_{(\boldsymbol{x}, \boldsymbol{h}) \sim p(\boldsymbol{x}, \boldsymbol{h})}$. The formal are improved estimators with reduced variances. In statistics this is known as the Rao-Blackwellization trick. Whenever you can perform marginalization analytically in a Monte Carlo calculation, please do it.*

Although in principle the RBM can represent any probability distribution given sufficiently large number of hidden neurons, the requirement can be exponential. To further increase the representational efficiency, one introduces the deep Boltzmann Machine (DBM) which has more than one layers of hidden neurons, see Fig. 3(b). Under information theoretical considerations, one can indeed show there are certain data which is impossible to represent using an RBM, but can possibly be represented by the DBM with the same number of hidden neurons and connections [26]. However, the downside of DBMs is that they are even harder to train and sample due the interactions among the hidden units [27].

3.2.2 *Autoregressive Models*

Arguably the simplest probabilistic model is the autoregressive models. They belong to the *fully visible Bayes network*. Basically, they breaks the full probability function into products of conditional probabilities, e.g.,

$$p(\boldsymbol{x}) = \prod_i p(x_i|\boldsymbol{x}_{<i}). \tag{36}$$

One can parameterize and learn the conditional probabilities using neural networks. In practice, one can model all these conditional probabilities using a single neural network, either a recurrent neural network with variable length, or using a feedforward neural network with masks. Note that these neural networks do not directly output the sample x_i, but the *parameters* of the conditional probability. For example, for continuous variables we can demand $p(x_i|\boldsymbol{x}_{<i}) = \mathcal{N}(x_i; \mu_i, \sigma_i^2)$, where the mean and variance are functions of $\boldsymbol{x}_{<i}$. The log-likelihood of a given data is easily computed as

$$\ln p(\boldsymbol{x}) = -\frac{1}{2} \sum_i \left(\left(\frac{x_i - \mu_i}{\sigma_i} \right)^2 + \ln(2\pi\sigma_i) \right). \tag{37}$$

To sample from the autoregressive model, we can sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{1})$ and iterate the update rule

$$x_i = \sigma_i(\boldsymbol{x}_{<i})\epsilon_i + \mu_i(\boldsymbol{x}_{<i}). \tag{38}$$

Info

A slightly awkward but very enlightening way to compute the log-likelihood of the autoregressive model is to treat Eq. (38) as an invertible mapping between $\boldsymbol{x}$ and $\boldsymbol{\epsilon}$, and invoke the probability transformation

$$\ln p(\boldsymbol{x}) = \ln \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{1}) - \ln \left| \det \left(\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\epsilon}} \right) \right|. \tag{39}$$

Notice that Jacobian matrix is triangular, whose determinant can be easily computed to be Eq. (37). Generalizing this idea to more complex bijective transformations bring us to a general class of generative models called *Normalizing Flow* [28–36]. In particular, a stack of autoregressive transformations is called autoregressive flow (AF).

Despite their simplicity, autoregressive networks have achieved state of the art performances in computer vision (PixelCNN and Pixel-RNN [33]) and speech synthesis (WaveNet [34]). The downside of autoregressive models is that one has to impose an order of the conditional dependence which may not correspond to the global hierarchical structure of the data. Moreover, sequential sampling of the autoregressive model such as Eq. (38) is considered to be slow since they can not take advantage of modern hardware. Nevertheless, the generative process Eq. (38) is *direct* sampling, which is much more efficient compared to the Gibbs sampling of Boltzmann Machines.

---- Info ----

The inverse autoregressive flow (IAF) [32] changes the transformation Eq. (38) to be

$$x_i = \sigma_i(\boldsymbol{\epsilon}_{<i})\epsilon_i + \mu_i(\boldsymbol{\epsilon}_{<i}), \qquad (40)$$

so that one can generate the data in parallel. The log-likelihood of the generated data also follows Eq. (37). However, the downside of the IAF is that it can not efficiently compute the likelihood of an arbitrary given data which is not generated by itself. Thus, IAF is not suitable for density estimation. IAF was originally introduced to improve the encoder of the VAE [32]. Recently, DeepMind use an IAF (parallel WaveNet) [37] to learn the probability density of an autoregressive flow (WaveNet) [34], thus to improve the speech synthesis speed to meet the needs in real-world applications [38]. To train the parallel WaveNet, they minimize the *Probability Density Distillation* loss $\mathbb{KL}(p_{\text{IAF}}||p_{\text{AF}})$ [37] since it is easy to draw sample from IAF, and easy to compute likelihood of AF on given data.

3.2.3 *Normalizing Flow*

Normalizing flow is a family of bijective and differentiable (i.e., diffeomorphism) neural networks which maps between two continuous variables z and x of the same dimension. The idea is that the

physical variables can have more complex realistic probability density compared to the latent variables [28–36]

$$\ln p(x) = \ln p(z) - \ln \left| \det \left(\frac{\partial x}{\partial z} \right) \right|. \tag{41}$$

Since diffeomorphism forms a group, the transformation is compositional $x = g(z) = \cdots \circ g_2 \circ g_1(z)$, where each step is a diffeomorphism. And the log-Jacobian determinant in Eq. (41) is computed as $\ln \left| \det \left(\frac{\partial x}{\partial z} \right) \right| = \sum_i \ln \left| \det \left(\frac{\partial g_{i+1}}{\partial g_i} \right) \right|$. To compute the log-likelihood of a given data, one first infer $z = g^{-1}(x)$ and keep track of the log-Jacobian determinant in each step.

The abstraction of a diffeomorphism neural network is called a bijector [39, 40]. Each bijector should provide interface to compute forward, inverse and log-Jacobian determinant in an efficient way. The bijectors can be assembled in a modular fashion to perform complex probability transformation. Because of their flexibility, they can act as drop in components of other generative models.

Bijectors are modular

─── **Example** ───

As an example of Eq. (41), consider the famous Box-Muller transformation which maps a pair of uniform random variables z to Gaussian random variables x

$$\begin{cases} x_1 = \sqrt{-2 \ln z_1} \cos(2\pi z_2), \\ x_2 = \sqrt{-2 \ln z_1} \sin(2\pi z_2). \end{cases} \tag{42}$$

Since $\left| \det \left(\frac{\partial x}{\partial z} \right) \right| = \left| \det \begin{pmatrix} \frac{-\cos(2\pi z_2)}{z_1 \sqrt{-2\pi \ln z_1}} & -2\pi \sqrt{-2 \ln z_1} \sin(2\pi z_2) \\ \frac{-\sin(2\pi z_2)}{z_1 \sqrt{-2\pi \ln z_1}} & 2\pi \sqrt{-2 \ln z_1} \cos(2\pi z_2) \end{pmatrix} \right| = \frac{2\pi}{z_1}$, we confirm that $p(x) = p(z) / \left| \det \left(\frac{\partial x}{\partial z} \right) \right| = \frac{1}{2\pi} \exp \left(-\frac{1}{2}(x_1^2 + x_2^2) \right)$. Normalization flows are generalizations of this trick to higher dimensional spaces while still keeping the Jacobian determinants easy to compute.

We take the real-valued non-volume preserving transformation (Real NVP) [31] as an example of the normalizing flow. For each layer of the Real NVP network, we divide multi-dimensional variables x^ℓ into two subgroups $x^\ell = x^\ell_< \cup x^\ell_>$ and transform one subgroup conditioned on the other group at each step

$$\begin{cases} x^\ell_< = x^\ell_< \\ x^\ell_> = x^\ell_> \odot e^{s_\ell(x^\ell_<)} + t_\ell(x^\ell_<) \end{cases} \tag{43}$$

where $s_\ell(\cdot)$ and $t_\ell(\cdot)$ are two arbitrary functions (with correct input/output dimension) which we parametrize using neural networks.

It is clear that this transformation is easy to invert by reversing the scaling and translation operations. Moreover, the Jacobian determinant of the transformation is also easy to compute since the matrix is triangular. By applying a chain of these elementary transformations to various bipartitions one can transform in between a simple prior density and a complex target density. The Real NVP network can be trained with standard maximum likelihood estimation on data. After training, one can generate new samples directly by sampling latent variables according to the prior probability density and passing them through the network. Moreover, one can perform inference by passing the data backward through the network and obtain the latent variables. The log-probability of the data is efficiently computed as

$$\ln p(\boldsymbol{x}) = \ln p(\boldsymbol{z}) - \sum_{\ell,i} (s_\ell)_i, \tag{44}$$

where the summation over index i is for each component of the output of the s function.

3.2.4 Variational Autoencoders

Variational autoencoder (VAE) is an elegant framework for performing variational inference [41], which also has deep connection variational mean field approaches in statistical physics. In fact, the predecessor of VAE is called Helmholtz machines [42]. The general idea of an autoencoder is to let the input data go through a network with bottleneck and restore itself. After training, the first half of the network is an encoder which transform the data $\boldsymbol{x}$ into the latent space $\boldsymbol{z}$. And the second half of the network is a decoder which transform latent variables into the data manifold. The bottleneck means that we typically require that the latent space has lower dimension or simpler probability distribution than the original data.

One of the creators of VAE, Max Welling, did his PhD on gravity theory under the supervision of 't Hooft in late 90s.

Suppose the latent variables $p(\boldsymbol{z})$ follow a simple prior distribution, such as an independent Gaussian. The decoder is parameterized by a neural network which gives the conditional probability $p(\boldsymbol{x}|\boldsymbol{z})$. Thus, the joint probability distribution of the visible and latent variables is also known $p(\boldsymbol{x},\boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$. However, the encoder probability given by the posterior $p(\boldsymbol{z}|\boldsymbol{x}) = p(\boldsymbol{x},\boldsymbol{z})/p(\boldsymbol{x})$ is much more difficult to evaluate since normalization factor $p(\boldsymbol{x})$ is intractable. One needs to marginalize the latent variables $\boldsymbol{z}$ in the joint probability distribution $p(\boldsymbol{x}) = \int p(\boldsymbol{x},\boldsymbol{z})\mathrm{d}\boldsymbol{z}$.

Intractable posterior

The intractable integration over the latent variables also prevent us minimizing the NLL on the dataset. To deal with such problem, we employ variational mean-field approach in statistical physics.

Consider in the statistical physics where $\pi(z) = e^{-E(z)}/\mathcal{Z}$ and $\mathcal{Z} = \sum_z e^{-E(z)}$. In Stat-Mech we try to minimize the free energy $-\ln \mathcal{Z}$, which is unfortunately intractable in general. To proceed, we define a variational free energy

$$\mathcal{L} = \sum_z q(z) \ln \left[\frac{q(z)}{e^{-E(z)}} \right] = \langle E(z) + \ln q(z) \rangle_{z \sim q(z)} \qquad (45)$$

for a normalized variational probability distribution $q(z)$. The two terms have the physical meaning of "energy" and "entropy" respectively. Crucially, since

$$\mathcal{L} + \ln \mathcal{Z} = \mathbb{KL}(q||\pi) \geq 0, \qquad (46)$$

thus Eq. (45) is a variational upper bound of the physical free energy, $-\ln \mathcal{Z}$. The approximation becomes exact when the variational distribution approaches to the target probability. Equation (46) is known as Gibbs-Bogoliubov-Feynman inequality in physics.

In analogy to variational free energy calculation in statistical physics, we have variational Bayes methods. For each data we introduce

$$\mathcal{L}(x) = \langle -\ln p(x, z) + \ln q(z|x) \rangle_{z \sim q(z|x)}, \qquad (47)$$

which is a variational upper bound of $-\ln p(x)$ since $\mathcal{L}(x) + \ln p(x) = \mathbb{KL}(q(z|x)||p(z|x)) \geq 0$. We see that $q(z|x)$ provides a variational approximation of the posterior $p(z|x)$. By minimizing $\mathcal{L}$ one effectively pushes the two distributions together. And the variational free energy becomes exact only when $q(z|x)$ matches to $p(z|x)$. In fact, $-\mathcal{L}$ is called evidence lower bound (ELBO) in variational inference.

We can obtain an alternative form of the variational free energy

$$\mathcal{L}_{\theta,\phi}(x) = - \langle \ln p_\theta(x|z) \rangle_{z \sim q_\phi(z|x)} + \mathbb{KL}(q_\phi(z|x)||p(z)). \qquad (48)$$

This breakup is also the foundation of the Expectation-Maximization algorithm, where one iterates alternatively between optimizing the variational posterior (E) and the parameters (M) to learn models with latent variables [43].

The first term of Eq. (48) is the reconstruction negative log-likelihood, while the second term is the KL divergence between the approximate posterior distribution and the latent prior. We also be explicit about the network parameters θ, ϕ of the encoder and decoder.

The decoder neural network $p_\theta(x|z)$ accepts the latent vector z and outputs the parametrization of the conditional probability. It can be

$$\ln p_\theta(x|z) = \sum_i x_i \ln \hat{x}_i + (1 - x_i) \ln(1 - \hat{x}_i), \qquad (49)$$

$$\hat{x} = \texttt{DecoderNeuralNet}_\theta(z), \qquad (50)$$

for binary data. And

$$\ln p_\theta(x|z) = \ln \mathcal{N}(x; \mu, \sigma^2 \mathbf{1}), \qquad (51)$$

$$(\mu, \sigma) = \texttt{DecoderNeuralNet}_\theta(z), \qquad (52)$$

for continuous data. Gradient of Eq. (48) with respect to $\boldsymbol{\theta}$ only depends on the first term.

Similarly, the encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is also parametrized as a neural network. To optimize $\boldsymbol{\phi}$ we need to compute the gradient with respect to the sampling process, which we invoke the *reparametrization trick*. To generate sample $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ we first sample from an independent random source, say $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{1})$ and pass it through an invertible and differentiable transformation $\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon})$. The probability distribution of the encoder is related to the one of the random source by

$$\ln q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \ln \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{1}) - \ln \left| \det \left(\frac{\partial g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon})}{\partial \boldsymbol{\epsilon}} \right) \right|. \qquad (53)$$

Suppose that the log-determinant is easy to compute so we can sample the latent vector $\boldsymbol{z}$ given the visible variable $\boldsymbol{x}$ and an independent random source $\boldsymbol{\epsilon}$. Now that the gradient can easily pass through the sampling process

$$\nabla_{\boldsymbol{\phi}} \left\langle f(\boldsymbol{x}, \boldsymbol{z}) \right\rangle_{\boldsymbol{z} \sim q_{\phi}(\boldsymbol{z}|\boldsymbol{x})} = \left\langle \nabla_{\boldsymbol{\phi}} f(\boldsymbol{x}, g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon})) \right\rangle_{\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{1})}. \qquad (54)$$

Info

As an alternate, the REINFORCE [44] (score function) estimator of the gradient reads

$$\nabla_{\boldsymbol{\phi}} \left\langle f(\boldsymbol{x}, \boldsymbol{z}) \right\rangle_{\boldsymbol{z} \sim q_{\phi}(\boldsymbol{z}|\boldsymbol{x})} = \left\langle f(\boldsymbol{x}, \boldsymbol{z}) \nabla_{\boldsymbol{\phi}} \ln q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) \right\rangle_{\boldsymbol{z} \sim q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}. \qquad (55)$$

Compared to the reparametrization Eq. (54) REINFORCE usually has larger variance because it only uses the scalar function $\ln q_{\phi}(\boldsymbol{z}|\boldsymbol{x})$ instead of the vector information of the gradient $\nabla_{\boldsymbol{\phi}} f(\boldsymbol{x}, \boldsymbol{z})$. An advantage of REINFORCE is that it can also work with discrete latent variables. See Ref. [45] for the research frontier for low variance unbiased gradient estimation for discrete latent variables.

Suppose each component of the latent vector follows independent Gaussian whose mean and variance are determined by the data $\boldsymbol{x}$, we have

$$\begin{aligned} \ln q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) &= \ln \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \sigma^2 \mathbf{1}), \qquad &(56) \\ (\boldsymbol{\mu}, \sigma) &= \texttt{EncoderNeuralNet}_{\boldsymbol{\phi}}(\boldsymbol{x}). \qquad &(57) \end{aligned}$$

And the way to sample the latent variable is

$$\begin{aligned} \boldsymbol{\epsilon} &\sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{1}), \qquad &(58) \\ \boldsymbol{z} &= \boldsymbol{\mu} + \sigma \odot \boldsymbol{\epsilon}. \qquad &(59) \end{aligned}$$

The KL term in Eq. (48) can be evaluated analytically [41] in this case.

After training of the VAE, we obtain an encoder $q(z|x)$ and a decoder $p(x|z)$. The encoder performs dimensionality reduction from the physical space into the latent space. Very often, different dimensions in the latent space acquire semantic meaning. By perform arithmetic operations in the latent space one can interpolate between physical data. Optimization of chemical properties can also be done in the low dimensional continuous latent space. The decoder is a generative model, which maps latent variable into the physical variable with rich distribution.

Info

The marginal NLL of the VAE can be estimated using importance sampling

$$-\ln p(x) = -\ln \left\langle \frac{p(x, z)}{q(z|x)} \right\rangle_{z \sim q(z|x)}. \tag{60}$$

By using the Jensen's inequality (21) one can also see that the variational free energy Eq. (47) is an upper bound of Eq. (60).

3.2.5 *Tensor Networks*

A new addition to the family of generative models is the tensor network state. In a quantum inspired approach one models the probability as the wavefunction square

$$p(x) = \frac{|\Psi(x)|^2}{\mathcal{Z}}, \tag{61}$$

where $\mathcal{Z}$ is the normalization factor. This representation, named as Born Machine [46], transforms many approaches of representing quantum state into machine learning. Consider binary data, we can represent wavefunction using the matrix product state (MPS) [47]

$$\Psi(x) = \text{Tr}\left(\prod_i A^i[x_i]\right). \tag{62}$$

The size of each matrix is called the bond dimension of the MPS representation. They control the expressibility of the MPS parameterization. The MPS can be learned using maximum likelihood estimation as before. Although other loss functions such as fidelity of quantum states can also be considered [48, 49].

An advantage of using MPS for generative modeling is that one can adopt algorithms developed for quantum many-body states such as the DMRG for parameter learning. For example, one can perform "two-site" optimization by merging two adjacent matrices together and optimizing its tensor elements. After the optimization the rank of the two site tensor may grow, one can thus dynamically adjust the

Adaptive learning

28

bond dimension of the MPS representation during learning. As a consequence, the expressibility of the model grows as it observes the data, which is different from conventional generative models with fixed network with fixed number of parameters.

Another advantage of MPS as a generative model is that the gradient of the NLL (22) can be computed efficiently

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = -2 \left\langle \frac{\partial \ln \Psi(\boldsymbol{x})}{\partial \boldsymbol{\theta}} \right\rangle_{x \sim \mathcal{D}} + 2 \left\langle \frac{\partial \ln \Psi(\boldsymbol{x})}{\partial \boldsymbol{\theta}} \right\rangle_{x \sim p(\boldsymbol{x})}. \tag{63}$$

Note that the negative phase (second term) can also be written as Z'/Z, where $Z' = 2 \sum_x \Psi'(\boldsymbol{x}) \Psi(\boldsymbol{x})$ and the prime means derivatives with respect to the network parameter $\boldsymbol{\theta}$. Crucially, for MPS both Z' and Z can be evaluated efficiently via tensor contractions. So the gradient can be computed efficiently without resorting to the contrastive divergence, in contrast to the Boltzmann Machines (24). The NLL is also tractable so that MPS model knows the normalized density of each sample.

Finally, tractable normalization factor of MPS allows one to perform *direct* sampling instead of using MCMC used in the Boltzmann Machines. While compared to the autoregressive models, one can perform data restoration by removing any part of the data. This is because tensor networks expresses an undirected (instead of directed) probability dependence fo the data.

These aforementioned advantages apply as well to other unitary tensor networks such as the tree tensor network and MERA. It is yet to been seen whether one can unlock the potential of tensor networks for real world AI applications. Using Eq. (61) and associated quantum-inspired approaches (or even a quantum device) provide a great chance to model complex probabilities. While on a more conceptual level, one wish to have have more quantitative and interpretable approaches inspired by quantum physics research. For example, Born Machine may give us more principled structure designing and learning strategies for modeling complex dataset, and provide a novel theoretical understandings of the expressibility of generative models the quantum information perspective.

3.2.6 *Generative Adversarial Networks*

Different from the generative models introduced till now, the Generative Adversarial Networks (GAN) belong to the *implicit* generative models. That is to say that although one can generate samples using GAN, one does not have direct access to its likelihood. So obviously training of GAN is also not based on maximum likelihood estimation.

A generator network maps random variables $\boldsymbol{z}$ to physical data $\boldsymbol{x}$. A discriminator network D is a binary classifier which tries tell

whether the sample is from the dataset $\mathcal{D}$ (1) or synthesized (0). On the expanded dataset $\{(x, 1), (G(z), 0)\}$, the cross-entropy cost reads

$$\mathcal{L} = - \langle \ln D(x) \rangle_{x \sim \mathcal{D}} - \langle \ln\left(1 - D(G(z))\right) \rangle_{z \sim p(z)} . \tag{64}$$

Such cost function defines a minimax game $\max_G \min_D \mathcal{L}$ between the generator and the discriminator, where the generator tries to forge data to confuse the discriminator.

Since the loss function does not involve the probability of the generated samples, one can use an arbitrary neural network as the generator. Giving up likelihood increases the flexibility of the generator network at the cost that it is harder to train and evaluate. Assess the performance of GAN in practice often boils down to beauty contest. Lacking an explicit likelihood function also limits its applications to physics problems where quantitative results are important.

Recall the Born Machine mentioned in Sec. 3.2.5, suppose one implements a Born Machine using a quantum circuit, the resulting model would be an implicit model [50]. Since one usually does not have direct access to the quantum state of an actual quantum state, adversarial training against a classical neural network can be a way to learn the quantum circuit as a probabilisitic generative model [51].

Table 2: A summary of generative models and their salient features. Question marks mean generalizations are possible, but nontrivial.

Name	Training Cost	Data Space	Latent Space	Architecture	Sampling	Likelihood	Expressibility	Difficulty (Learn/Sample)
RBM	Log-likelihood	Arbitrary	Arbitrary	Bipartite	MCMC	Intractable partition function	★	☠☠ / ☠☠☠
DBM	ELBO	Arbitrary	Arbitrary	Bipartite	MCMC	Intractable partition function & posterior	★★★	☠☠ / ☠☠☠
Autoregressive Model	Log-likelihood	Arbitrary	None	Ordering	Sequential	Tractable	★★	☠ / ☠☠☠
Normalizing Flow	Log-likelihood	Continuous	Continuous, Same dimension as data	Bijector	Parallel	Tractable	★★	☠ /
VAE	ELBO	Arbitrary	Continuous	Arbitrary?	Parallel	Intractable posterior	★★★	☠ /
MPS/TTN	Log-likelihood	Arbitrary?	None or tree tensor	No loop	Sequential	Tractable	★★★	☠☠ / ☠☠
GAN	Adversarial	Continuous	Arbitrary?	Arbitrary	Parallel	Implicit	★★★★	☠☠☠ / ☠☠
Quantum Circuit	Adversarial	Discrete	Discrete	Arbitrary	Parallel	Implicit	★★★★★	☠☠☠ / ☠

In the discussions of generative models we have touched upon a field called probabilistic graphical models [52]. They represent independence relation using graphical notations. The graphical models with undirected edges are called Markov random field, which can be understood as statistical physics models (Sec. 3.2.1). Typically, it is hard to sample from a Markov random field unless it has a tree structure. While the graphical models with directed edges are called Bayes network, which describe conditional probability distribution (Sec. 3.2.2). The conditional probabilities allows ancestral sampling which start from the root node and follow the conditional probabilities.

As we have seen, feedforward neural networks can be used as key components for generative modeling. They transform the probability distribution of the input data to certain target probability distribution. Please be aware that there are subtle differences in the interpretations of these neural nets' outputs. They can either be parametrization of the conditional probability $p(x|z)$ (Secs. 3.2.2, 3.2.4) or be the samples x themselves (Secs. 3.2.3, 3.2.6). Table 2 summarized and compared the main features of various generative models discussed in this note.

In fact, various models introduce in this section is also related. Seeking their relation or trying to unify them provides one a deeper understanding on generative modeling. First of all, the Boltzmann Machines, and in general probabilistic graphical models, are likely to be closely related to the tensor networks. In particular cases, the exact mappings between RBM and tensor networks has been worked out [26]. In general, it is still rewarding to explore the connections of representation and learning algorithms between the two classes of models. Second, the autoregressive models are closely related to the normalizing flows viewed as a transformation of probability densities. While in [28] it was even argued on the connections to the variational autoencoder. Finally, combining models to take advantage of both worlds is also a rewarding direction [30, 32, 53].

APPLICATIONS TO QUANTUM MANY-BODY PHYSICS AND MORE

We now discuss a few applications of deep learning to quantum many-body physics. Since it is rather difficult to have a complete survey of this fast growing field, we select a few representative examples that we have meaningful things to say. Note that the selection is highly biased by the interests of the authors. We sincerely apologize for not mentioning your favorite papers because of our ignorance. We will also try to comment on the outlooks and challenges saw by authors (again, biased opinions).

The interested readers can check the ⟨Physics|Machine Learning⟩ blog for recent news, events and papers. The lecture materials of the KITS Workshop on Machine Learning and Many-Body Physics provides a sampled snapshot of the field up to summer of 2017.

4.1 MATERIAL AND CHEMISTRY DISCOVERIES

It is natural to combine machine learning techniques with materials genome project and high throughput screening of materials and molecules. In its most straightforward application, regression from microscopic composition to the macroscopic properties can be used to bypass laborious ab-initio calculation and experimental search. Finding appropriate descriptors for materials and molecules sometimes become a key. And such representation learning is exact what deep learning techniques are designed for. One of the crucial consideration in constructing these representations is to respect the translational, rotational and permutational symmetry of the physical system. See Refs. [54] and references therein for possible solutions.

A recent example in chemistry design is to use the VAE to map string representation of molecules to a continuous latent space and then perform differential optimization for desired molecular properties [55]. Like many deep learning applications in natural language and images, the model learned meaningful low dimensional representation in the latent space. Arithmetics operations have physical (or rather chemical) meanings. There were also attempts of using GANs for generating molecules.

IPAM@UCLA had a three months program on this topic in 2016, see the White Paper for a summary of progresses and open problems by participants of the program.

4.2 DENSITY FUNCTIONAL THEORY

Searching for density functionals using machine learning approaches is an active research frontier. Density functional theory (DFT) is in principle exact, at least for ground state energy and density distribution. However no one knows the universal exchange-correlation functional. Machine Learning modeling of exact density functional has been demonstrated in one dimension [56] with exact results coming from the density-matrix-renormalization-group calculation in continuous space. For more general realistic cases, besides how to model the density functionals, another problem is how to get accurate training data. If that problem get solved, then how about time-dependent DFT, where the functional is over space and time ?

Taking one step back, even in the regime of local density approximation, searching for a good kinetic energy functional can already be extremely useful since it can support orbital free DFT calculations [57, 58]. Bypassing Kohn-Sham orbitals (which are auxiliary objects anyway) can greatly accelerate the search of stable material structures

4.3 "PHASE" RECOGNITION

Ever since the seminal work by Carrasquilla and Melko [59], there are by now a large number of papers on classifying phases using neural networks. Among them, one of the authors (L.W.) advocated unsupervised learning approaches for discovering phase transitions [60]. To the authors' understanding, a grand goal would be to identify and even discover elusive phases and phase transitions (e.g. topological ones) which are otherwise difficult to capture. However, typically the machine learning models tend to pick up short-range features such as the energy, which is unfortunately non-universal. Thus, one of the great challenges is to *discover* nonlocal signatures such as the topological order (instead of manual feature engineering or fitting the topological invariance directly). Reading classical texts in pattern recognition [10, 11] may bring inspirations from the founding fathers of the field.

4.4 VARIATIONAL ANSATZ

Reference [61] obtained excellent variational energy for non-frustrated quantum spin systems by adopting the Restricted Boltzmann Machines in Sec. 3.2.1 as a variational ansatz. The ansatz can be viewed

as an alternative of Jastrows. But it is more flexible in the sense that it encodes multi-body correlations in an efficient way.

Later studies [26, 62, 63] connect the RBM variational ansatz to tensor network states. References [64, 65] analyzed their expressibility from quantum entanglement and computational complexity points of view respectively. Out of these works, one sees that the neural network states can be advantageous for describing highly entangled quantum states, and models with long range interactions. Another particular interesting application is on the chiral topological states, in which the standard PEPS ansatz suffer from fundamental difficulties [63, 66].

Another interesting direction is to interpret that RBM, in particular, the one used in [61] as shallow convolutional neural networks. Along this line, it is natural to go systematically to deeper neural networks and employ deep learning frameworks for automatic differentiation in the VMC calculation [67]. Reference [68] carried out the VMC calculation for small molecules in the first quantization formalism, in which the antisymmetric property of the wavefunction was taken with special care. It appears to us that further development calls for innovations in the the optimization scheme which is beyond the wavefunction ansatz, e.g. direct generative sampling, and low variance gradient estimator.

4.5 RENORMALIZATION GROUP

Renormalization Group (RG) is a fundamental concept in theoretical physics. In essence, RG keeps relevant information while reducing the dimensionality of data. The connection of RG and deep learning is quite intriguing since on one hand side it brings deep learning machineries into solving physical problems with RG, and on the other hand site, it may provide theoretical understanding to deep learning.

References [8] proposed a generative Bayesian network with a MERA inspired structure. Reference [9] connects the Boltzmann Machines with decimation transformation in real-space RG. Reference [69] connects principal component analysis with momentum shell RG. Reference [70] proposed to use mutual information as a criteria for restoring the RG behavior in the training of Boltzmann Machines. Lastly, Reference [71] proposed a variational RG framework by stacking the bijectors (Sec. 3.2.3) into a MERA-liked structure. The approach provides a way to identify collective variables and and their effective interaction. The collective variables in the latent space has reduced mutual information. They can be regarded as nonlinear and adaptive generalizations of wavelets. Training of the NueralRG network employs the probability density distillation (Sec. 3.2.2) on the bare energy function, in which the training loss provides a variational upper bound of the physical free energy. The NeuralRG approach imple-

ments an information preserving RG procedure, which is useful for exploring holographic mapping [72].

4.6 MONTE CARLO UPDATE PROPOSALS

Markov chain Monte Carlo (MCMC) finds wide applications in physics and machine learning. Since the major drawback of MCMC compared to other approximate methods is its efficiency, there is a strong motivation to accelerate MCMC simulations within both physics and machine learning community. Loosely speaking, there can be at least three ideas of accelerating Monte Carlo sampling using machine learning techniques.

First, Reference [73] trained surrogate functions to speed up hybrid Monte Carlo simulation [74] for Bayesian statistics. The surrogate function approximates the potential energy surface of the target problem and provides an easy way to compute derivatives. Recently, there were papers reporting similar ideas for physics problems. Here, the surrogate model can be physical models such as the Ising model [75] or molecular gases [76], or general probabilistic graphical models such as the restricted Boltzmann machine [77]. For Monte Carlo simulations involving fermion determinants [76, 78] the approach is more profitable since the updates of the original model is much heavier than the surrogate model. However, the actual benefit depends on the particular problem and the surrogate model. A drawback of these surrogate function approaches is that they require training data to start with, which is known as the "cold start" problem in analog to the recommender systems [76]. Using the adaptive approach of [79] one may somewhat alleviate this "chicken-egg" problem.

Second, there were more recent attempts in machine learning community trying to directly optimize the proposal probability via reinforcement learning [80–82]. These papers directly parameterize the proposal probability as neural networks and optimize objective functions related to the efficiency, e.g., the difference of samples or the squared jump. To ensure detailed balance condition, it is crucial to keep track of the proposal probability of an update and its reverse move. Both A-NICE-MC [80] and L2HMC [81] adopted normalizing flows (Sec. 3.2.3). The later paper is particularly interesting because it systematically generalizes the celebrated hybrid Monte Carlo [74] to a learnable framework. Reference [83] used reinforcement learning to devise updates for frustrated spin models. In fact, besides the efficiency boost one can aim at algorithmic innovations in the Monte Carlo updates. Devising novel update strategies which can reduce the auto correlation between samples was considered to be the art MCMC methods. An attempt along this line is Ref. [84], which connected the Swendsen-Wang cluster and the Boltzmann Machines and explored a few new cluster updates.

Lastly, the NeuralRG technique [71] provides another approach to learn Monte Carlo proposal without data. Since the mapping to the latent space reduces the complexity of the distribution, one can perform highly efficient (hybrid) Monte Carlo updates in the latent space and obtain unbiased physical results. This can be regarded as an extension of the Fourier space or wavelets basis Monte Carlo methods, except that now the representation is learned in an adaptive fashion.

By the way, to obtain unbiased physical results one typically ensures detailed balance condition using Metropolis-Hastings acceptance rule. Thus, one should employ generative models with *explicit* and *tractable* densities for update proposals. This rules out GAN and VAE in the game, at least for the moment.

4.7 TENSOR NETWORKS

Reference [85] is the first in the physics community to apply tensor network approach to pattern recognition. The paper and blog post mentioned some parallel attempts in the computer science community. In a second paper [49], Miles Stoudenmire boosted the performance by first performing unsupervised feature extraction using a tree tensor network. Reference [47] uses MPS for generative modeling of the classical binary data. Since generative tasks are harder, the authors believe that it is in these class of problems one may unlock bigger potential of tensor networks.

There are at least two motivations of using the tensor networks for machine learning. First, tensor network and algorithms provide principled approaches for discriminative and generative tasks with possibly stronger representational power. In fact, the mathematical structure of tensor network states and quantum mechanics appear naturally when one tries to extend the probabilistic graphical models while still attempt to ensure the positivity of the probability density [86, 87]. Second, tensor networks are doorways to one form of quantum machine learning because the tensor networks are formally equivalent to quantum circuits. Tensor network states provide means of architecture design and parameter initialization for quantum circuits [88]. By now, tensor networks have certainly caught attentions of some of the machine learning practitioners [89, 90]. We are still awaiting for an event similar to AlexNet, where the tensor network machine learning approach wins over the traditional approaches by a large margin. With accumulations of the results and techniques, this is likely to happen in the coming years, at least in one specific application domain.

Quantum machine learning is a general term with many applications. See reviews [91, 92] for its past.

Getting closer to the topics covered in this lecture note, Boltzmann Machines were used to decode quantum error [93] and for quantum state tomography [94]. Building on the probabilistic interpretation of quantum mechanics, one can envision a quantum generative model [95]. It expresses the probability distribution of a dataset as the probability associated with the wavefunction. The theory behind it is that even measured on a fixed bases, the quantum circuit can express probability distribution that are intractable to classical computers. In particular, there was an experiment [96] on generative machine learning. To the authors, the biggest concern is how to train the quantum circuits analogously to the neural networks using back-propagation. The tricks on market (quantum classical hybrid approach with gradient free optimizer) are not scalable.

Finally, there are reinforcement learning for quantum control [97] and designing new quantum optics experiment [98].

4.9 MISCELLANEOUS

There is an attempt to train regression model as the impurity solver of dynamical mean-field theory [99] and for analytical continuation of imaginary data [100]. Like many of the regressions applications, the concern is where to obtain the labelled training data in the first place. The later application is representative in applying machine learning to inverse problems by exploiting the fact that it is relatively easy to collect training data by solving the forward problem.

Lastly, concerning the fermion sign problem in Monte Carlo simulation. There was an attempt of classify the nodal surface in diffusion Monte Carlo (DMC) [101]. This appears to be an interesting attempt since understanding the nodal surface in DMC would imply one can complete solve the ground state. Besides the concern on lacking labelled training data, the nodal surface might be fractal in the most challenging case. Finally, there are attempts of alleviating the sign problem via deforming the integration path in the complex domain [102, 103].

HANDS ON SESSION

Differential programing and GPU acceleration are two key ingredients for the success of contemporary deep learning application. In this chapter, we will get familiar with these tools for studying quantum many-body systems step by step, with examples.

Codes in this tutorial have been uploaded to the following git repository https://github.com/GiggleLiu/marburg.git

5.1 COMPUTATION GRAPH AND BACK PROPAGATION

We have introduced the basic idea of training neural network using back propagation in Sec.2.4.1. In order to make back propagation easy to implement, especially on GPUs, people often use computation graphs to represent the loss functions. A computation graph consists of nodes and edges.

- An edge represents a function argument (or data dependency).

- A node is a function. If there is no incomming edges on this node, it represents special function that feeds variables to this graph.

- A node knows how to compute its output and the its derivative w.r.t each input argument.

The third clause ensures that back propagation is applicable on this graph. Figure 4 is a computation graph for the single layer perceptron. It is easy to infer that this graph represents an operation $y = \sigma(Wx + b)$.

Please check out Viznet, a package for drawing neural networks, tensor networks, and quantum circuits.

Exercise 7 (**BackProp and Computation Graph in Numpy**). *Now we implement back propagation in computation graph style using pure numpy. Read and run notebook notebooks/computation_graph.ipynb, or an online version https://goo.gl/6d2sei. Complete the task at the end of the notebook.*
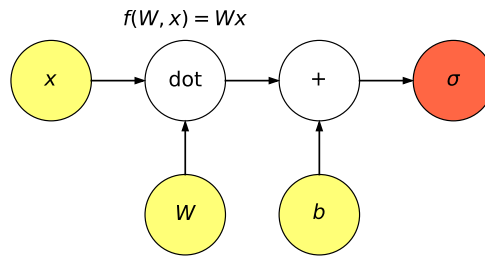
Figure 4: Computation graph for a single layer perceptron model, yellow nodes are inputs, the orange node is output, white nodes are functions. Colors used here are for illustration purpose, not a requirement of computation graphs.

Device	# Cores	Clock Speed	Memory	Price
CPU(Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.4 GHz	Shared with system	$339
CPU(Intel Core i7-6950X)	10 (20 threads with hyperthreading)	3.5 GHz	Shared with system	$1723
GPU (NVIDIA Titan Xp)	3840	1.6 GHz	12 GB GDDR5X	$1200
GPU (NVIDIA GTX 1070)	1920	1.68 GHz	8 GB GDDR5	$399

Table 3: A table of comparison between CPU and GPU. This table is from course cs231n of Stanford University, April 27 2017.

CPU vs GPU

Table 3 shows a GPU can have more than 1k CUDA cores. Each CUDA core is relatively weak comparing with a CPU core. Its clock speed is much lower, and is only able to process two floating point operations per cycle.

By counting floating point operations per second (FLOPS), GPU will give you > 100 times speed up with respect to a CPU on the same price. In practice, this value is barely achieved due to the overhead of data transfer between system memory and GPU memory.

Problems which have a high arithmetic intensity and regular memory access patterns are typically easier to implement on GPUs.

model	dataset	epochs	batch	Knet	Theano	Torch	Caffe	TFlow
LinReg	Housing	10K	506	2.84	1.88	2.66	2.35	5.92
Softmax	MNIST	10	100	2.35	1.40	2.88	2.45	5.57
MLP	MNIST	10	100	3.68	2.31	4.03	3.69	6.94
LeNet	MNIST	1	100	3.59	3.03	1.69	3.54	8.77
CharLM	Hiawatha	1	128	2.25	2.42	2.23	1.43	2.86

Table 4: Performance benchmark for selected libraries using different datasets and neural network models. Table is from [104].

5.2 DEEP LEARNING LIBRARIES

In fact, you don't need to realize all computation nodes from scrach. People have built several high performance libraries that implement computation graph. A comparison of their single GPU performance are shown in Table 4. In the list of neural network libraries shown above, Theano and Tensorflow (TFlow) are Python libraries. Torch is in Lua but its cousine PyTorch is the python version with a similar performance. All these libs are based on CUDA programming, especially cuDNN. cuDNN is an officially optimized library for deep learning with respect to hardware. Due to the usage of this common backend, there is no reason that one lib is significantly better than another providing used properly.

PyTorch is our primary coding choice for the following tutorials. PyTorch is an open source machine learning library primarily developed by Facebook's artificial-intelligence research group. Unlike tensorflow that uses static graph, pytorch uses dynamic graph. For the static graphs, you should first draw the computation graph and then feed data to this graph and run it on chosen device (define-and-run). Using dynamic graphs, the graph structure is defined on-the-fly via the actual forward computation. This is a far more natural style of programming (define-by-run).

Pytorch's numpy like coding style makes it easy to use, debug and make extensions. However, before fully embracing pytorch, you should know its weaknesses.

- Pytorch does not have complex number support, the github issue on complex valued network is in low priority. This kind of concern do not exist in tensorflow.

- Pytorch's initial release is in Oct. 2016. So it has comparatively smaller community, less tool chains (e.g. its visualization is not as good as tensorflow's tensorboard) and also less stable.
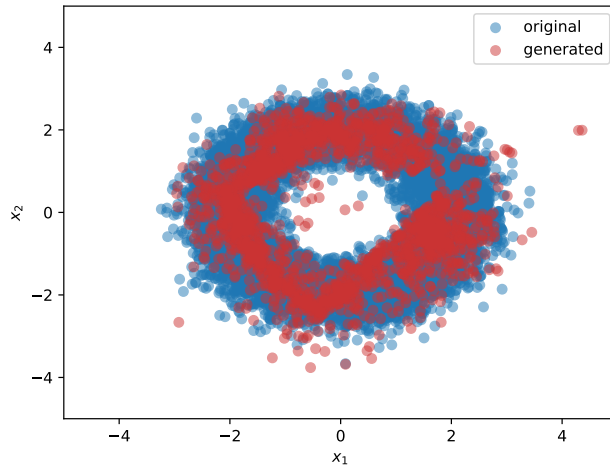
Figure 5: Samples from a 2d ring distribution (blue dots) and samples generated by NICE network (red dots).

The goal of this section is to implement a Normalizing Flows network introduced in Sec. 3.2.3 using pytorch for density estimation. In this task, we are provided with samples from an unknown distribution, and we need to estimate the probability density of each sample by transforming it to a sample from simple distributions like Gaussian distribution . Conversely, we can also generate samples directly using the trained network.

The samples given is sampled from a 2d ring distribution(the blue dots in Fig. 5). Recall that Normalizing Flow can be built with layers of bijectors. Here we are going to implement a Non-linear Independent Components Estimation (NICE) network, where each layer performs the following transformation [28]

$$
\begin{cases}
\boldsymbol{x}^{\ell}_{<} = \boldsymbol{x}^{\ell}_{<} \\
\boldsymbol{x}^{\ell}_{>} = \boldsymbol{x}^{\ell}_{>} + t_{\ell}(\boldsymbol{x}^{\ell}_{<})
\end{cases}
\tag{65}
$$

Using the trained NICE network we can obtain new samples following the 2d Ring distribution, see the red dots in Fig 5.

Exercise 8 (**Learning and Sampling using Normalizing Flow networks**). *Read and run notebook notebooks/nice.ipynb, or an online version https://goo.gl/8Caymh. Complete the task at the end of the notebook. In this task, with only little change, you can turn the NICE network into Real NVP network, recall that in Real NVP network, in each layer the transform is Eq. (43).*
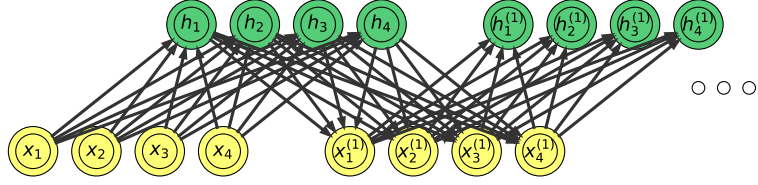
Figure 6: Gibbs sampling of an RBM.

5.4 RESTRICTED BOLTZMANN MACHINE FOR IMAGE RESTORATION

Restricted Boltzmann Machine can learn about data probability distribution and generate new samples accordingly. Eq. (24) explains how to obtain the gradient of the NLL loss of an RBM. For the negative phase we use Gibbs sampling to obtain samples from the the joint distribution $p(x, h)$. The Gibbs sampling of the RBM samples back and forth between the visible and hidden units using the conditional probabilities $p(x|h)$ and $p(h|x)$. The samples will converge to $x^\infty \sim p(x)$

In practice, we make a truncation to the sampling process, and use x^k to approximate x^∞, which is called k-step contrastive divergence (CD-k). Note that fixed points of parameters for different k are not the same one in general [105].

Exercise 9 (**Image restoration using RBM**). *We provide an example to train an RBM to generate and restore MNIST images. Read and run notebook notebooks/rbm_generation.ipynb, or an online version https://goo.gl/d7kPzy. Complete the task at the end of the notebook.*

5.5 NEURAL NETWORK AS A QUANTUM WAVE FUNCTION ANSATZ

Variational Monte Carlo (VMC) [106] tries variational optimize a state ansatz $\psi_\theta(x) = \langle x|\psi_\theta\rangle$ to minimize the energy of a target hamiltonian H

$$E_\theta = \frac{\langle \psi_\theta|H|\psi_\theta\rangle}{\langle \psi_\theta|\psi_\theta\rangle},$$

where θ represents tunable parameters of this ansatz, and x is a string of ± 1's which represent a spin configuration, with 1 spin up and -1 spin down.

Obviously, E_θ is the loss function we want to minimize. The gradient of E with respect to θ is given by

$$\frac{\partial E}{\partial \theta} = \langle E_{\text{loc}}\Delta_{\text{loc}}^*\rangle - \langle E_{\text{loc}}\rangle\langle \Delta_{\text{loc}}^*\rangle, \tag{66}$$
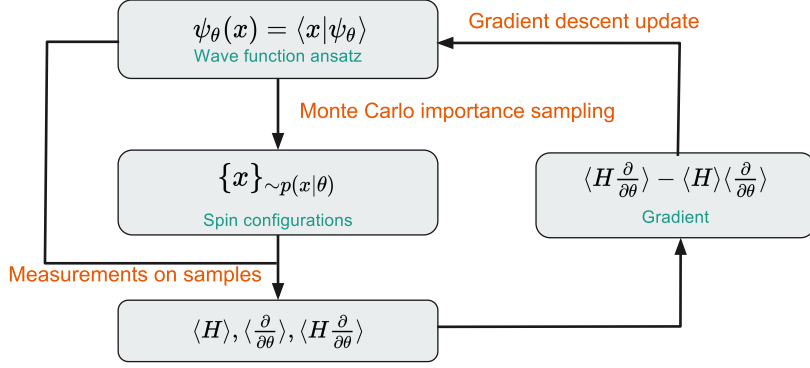
Figure 7: The flowchart of VMC calculation.

where $\langle \cdot \rangle$ is the ensemble average over Monte Carlo importance sampling over probability distribution $x \sim |\langle x|\psi_\theta \rangle|^2$, $E_{\text{loc}} = \frac{\langle x|H|\psi_\theta \rangle}{\langle x|\psi_\theta \rangle}$, $\Delta_{\text{loc}} \equiv \frac{\partial \ln \psi_\theta(x)}{\partial \theta}$.

The training process is shown in Fig. 7. Deep learning are incorporated into VMC framework by writing wave function ansatz in computation graph form. In the sampling phase, VMC queries $p_\theta(x) \propto |\psi_\theta(x)|^2$ from computation graph, with only forward pass executed. In the evaluation phase, we perform local measurements to obtain E_{loc} and Δ_{loc}. The evaluation of E_{loc} again requires many forward calls. On the other side, Δ_{loc} requires back propagation with respect to $\ln \psi_\theta(x)$.

After obtaining the gradient in Eq. (66), we update parameters according to gradient descent $\theta = \theta - \eta \frac{\partial E}{\partial \theta}$, with η the learning rate.

Exercise 10 (Deep Learning for VMC). *Read and run notebook notebooks/rbm_generative.ipynb, or an online version https://goo.gl/vPFtdU. Complete the task at the end of the notebook. This is an example of using RBM wave function ansatz for VMC. You can insert your own ansatz by replacing RBM with other computation graph. Also, we prepared a frustrated J1-J2 Hamiltonian with a non-trivial sign structure for you. Please challenge yourself.*

6

CHALLENGES AHEAD

Now that you have learned about theory and practice about deep learning, and its applications to quantum many-body problems, we'd like to pose a few challenges to you. Overall, we believe it is important to solve *new problems* besides reproducing known knowledge.

1. Can it discover a new phase of matter ?

2. Would it discover new algorithms for us ?

3. Would it be possible for us to make progress on fermion sign problem ?

4. Non-stochastic, or better ways for optimizing, renormalizing, and evolving Neural Networks.

5. Information pattern aware structure learning of neural networks.

7

RESOURCES

The book [107] is a good introductory reading, which gives you an overview of the "five schools" of the machine learning. Classical texts on neural networks are [10] and [11]. Modern textbooks are [43, 108] and [22]. The book by David MacKay [23] is a great source for inspirations. More broadly on AI, there is [1]. Mehta et al has a hundred pages long review (plus Python notebooks) oriented to physicists [109].

Besides books, there are great online resources to learn about deep learning. The online book by Michael Nielsen is a very accessible introduction to neural networks (written by a former quantum physicist). Andrew Ng's lectures at Coursera and Stanford's CS231n are standard references. We also find EPFL EE-559 "Deep Learning" taught by François Fleuret and U. Toronto CSC 321 "Intro to Neural Networks and Machine Learning" taught by Roger Grosse quite helpful. The in-progress book Machine Learning for Artists contains many fun demos and visualizations. Distill is an online journal for machine learning research, which feature media interactive media. Arxiv-Sanity is a good place to keep track of most recent progresses in deep learning.

For more of physics orientated lectures on machine learning, please see Simon Trebst's lectures at Cornell, I and II; Giuseppe Carleo's series lectures at ICTP; Juan Carrasquilla and Roger Melko's minisourse at São Paulo. And there is a lecture series at the perimeter institute. Teaching materials and videos by Florian Marquardt of the course "Machine Learning for Physicists" can be found here.

ACKNOWLEDGEMENTS

BIBLIOGRAPHY

[1] Stuart Jonathan Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 2003.

[2] Nils J. Nilsson. *The Quest for Artificial Intelligence*. Cambridge University Press, 2009. URL http://ebooks.cambridge.org/ref/id/CBO9780511819346.

[3] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A Neural Algorithm of Artistic Style. 2015. URL https://arxiv.org/abs/1508.06576.

[4] David H Wolpert. The Lack of A Priori Distinctions Between Learning Algorithms. *Neural Comput.*, 8:1341, 1996.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529, 2015. doi: 10.1038/nature14236. URL http://dx.doi.org/10.1038/nature14236.

[6] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Math. Control. Signals, Syst.*, 2:303, 1989. doi: 10.1007/BF02836480.

[7] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991. doi: 10.1016/0893-6080(91)90009-T.

[8] Cédric Bény. Deep learning and the renormalization group. *arXiv*, 2013. URL http://arxiv.org/abs/1301.3124.

[9] Pankaj Mehta and David J. Schwab. An exact mapping between the Variational Renormalization Group and Deep Learning. *arXiv*, 2014. URL http://arxiv.org/abs/1410.3831.

[10] Marvin Minsky and Papert Seymour A. *Perceptrons, Expanded Edition*. The MIT Press, 1988.

[11] David E Rumelhart and James L McClelland. *Parallel Distributed Processing*. A Bradford Book, 1986.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv*, 2015. URL https://arxiv.org/abs/1512.03385.

[13] A. Gunes Baydin, B. A. Pearlmutter, A. Andreyevich Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *ArXiv e-prints*, February 2015.

[14] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse. The Reversible Residual Network: Backpropagation Without Storing Activations. *ArXiv e-prints*, July 2017.

[15] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural Ordinary Differential Equations. *ArXiv e-prints*, June 2018.

[16] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv*, 2013. URL http://arxiv.org/abs/1312.6199.

[17] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going Deeper into Neural Networks, 2015. URL https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

[18] Gabriel Goh. Why momentum really works. *Distill*, 2017. doi: 10.23915/distill.00006. URL http://distill.pub/2017/momentum.

[19] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv*, 2015. doi: 10.1007/s13398-014-0173-7.2. URL http://arxiv.org/abs/1502.03167.

[20] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

[21] Andrej Karpathy, Pieter Abbeel, Greg Brockman, Peter Chen, Vicki Cheung, Rocky Duan, Ian GoodFellow, Durk Kingma, Jonathan Ho, Rein Houthooft, Tim Salimans, John Schulman, Ilya Sutskever, and Wojciech Zaremba. OpenAI Post on Generative Models, 2016. URL https://blog.openai.com/generative-models/.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org/.

[23] David J C MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[24] Geoffrey E Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Comput.*, 14:1771, 2002.

[25] Tijmen Tieleman. Training Restricted Boltzmann Machines using Approximations to the Likelihood Gradient. In *Proceedings of teh 25th international conference on Machine Learning*, page 1064, 2008.

[26] Jing Chen, Song Cheng, Haidong Xie, Lei Wang, and Tao Xiang. On the Equivalence of Restricted Boltzmann Machines and Tensor Network States. *Phys. Rev. B*, 97:085104, 2018. doi: 10.1103/PhysRevB.97.085104. URL http://arxiv.org/abs/1701.04831.

[27] Ruslan Salakhutdinov. Learning Deep Generative Models. *Annu. Rev. Stat. Its Appl.*, 2:361–385, 2015. doi: 10.1146/annurev-statistics-010814-020120. URL http://www.annualreviews.org/doi/10.1146/annurev-statistics-010814-020120.

[28] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Nonlinear Independent Components Estimation. *arXiv*, 2014. doi: 1410.8516. URL http://arxiv.org/abs/1410.8516.

[29] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked Autoencoder for Distribution Estimation. *arXiv*, 2015. URL http://arxiv.org/abs/1502.03509.

[30] Danilo Jimenez Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. *arXiv*, 2015. URL http://arxiv.org/abs/1505.05770.

[31] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *arXiv*, 2016. doi: 1605.08803. URL http://arxiv.org/abs/1605.08803.

[32] Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving Variational Inference with Inverse Autoregressive Flow. *arXiv*, 2016. URL http://arxiv.org/abs/1606.04934.

[33] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel Recurrent Neural Networks. In *Int. Conf. Mach. Learn. I(CML)*, volume 48, pages 1747—-1756, 2016. URL https://arxiv.org/pdf/1601.06759.pdf{%}0Ahttp://arxiv.org/abs/1601.06759.

[34] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. *arXiv*, 2016. URL http://arxiv.org/abs/1609.03499.

[35] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked Autoregressive Flow for Density Estimation. *arXiv*, 2017. URL http://arxiv.org/abs/1705.07057.

[36] D. P. Kingma and P. Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. *ArXiv e-prints*, July 2018.

[37] Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C. Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel WaveNet: Fast High-Fidelity Speech Synthesis. *arXiv*, 2017. URL http://arxiv.org/abs/1711.10433.

[38] Aaron van den Oord, Yazhe Li, and Igor Babuschkin. High-fidelity speech synthesis with WaveNet, 2017. URL https://deepmind.com/blog/high-fidelity-speech-synthesis-wavenet/.

[39] Pyro Developers. Pyro, 2017. URL http://pyro.ai/.

[40] Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A. Saurous. TensorFlow Distributions. *arXiv*, 2017. URL http://arxiv.org/abs/1711.10604.

[41] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. 2013. URL http://arxiv.org/abs/1312.6114.

[42] Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The Helmholtz Machine. 904:889–904, 1995.

[43] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[44] Ronald J. Wiliams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.*, 8:229–256, 1992. doi: 10.1023/A:1022672621406.

[45] George Tucker, Andriy Mnih, Chris J. Maddison, Dieterich Lawson, and Jascha Sohl-Dickstein. REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models. *arXiv*, 2017. URL http://arxiv.org/abs/1703.07370.

[46] Song Cheng, Jing Chen, and Lei Wang. Information Perspective to Probabilistic Modeling: Boltzmann Machines versus Born Machines. *arXiv*, 2017. URL http://arxiv.org/abs/1712.04144.

[47] Zhao-Yu Han, Jun Wang, Heng Fan, Lei Wang, and Pan Zhang. Unsupervised Generative Modeling Using Matrix Product States. 2017. URL http://arxiv.org/abs/1709.01662.

[48] Ding Liu, Shi-Ju Ran, Peter Wittek, Cheng Peng, Raul Blázquez García, Gang Su, and Maciej Lewenstein. Machine Learning by Two-Dimensional Hierarchical Tensor Networks: A Quantum Information Theoretic Perspective on Deep Architectures. *arXiv*, 2017. URL http://arxiv.org/abs/1710.04833.

[49] E. M. Stoudenmire. Learning Relevant Features of Data with Multi-scale Tensor Networks. 2017. URL http://arxiv.org/abs/1801.00315.

[50] J.-G. Liu and L. Wang. Differentiable Learning of Quantum Circuit Born Machine. *ArXiv e-prints*, April 2018.

[51] H. Situ, Z. He, L. Li, and S. Zheng. Adversarial training of quantum Born machine. *ArXiv e-prints*, July 2018.

[52] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models*. MIT Press, 2009.

[53] Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational Lossy Autoencoder. *arXiv*, 2016. URL http://arxiv.org/abs/1611.02731.

[54] L. Zhang, J. Han, H. Wang, W. A. Saidi, R. Car, and W. E. End-to-end Symmetry Preserving Inter-atomic Potential Energy Model for Finite and Extended Systems. *ArXiv e-prints*, May 2018.

[55] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *arXiv*, 2016. doi: 10.1021/acscentsci.7b00572. URL http://arxiv.org/abs/1610.02415.

[56] Li Li, Thomas E Baker, Steven R White, and Kieron Burke. Pure density functional for strong correlations and the thermodynamic limit from machine learning Li. *arXiv*, 2016. URL http://arxiv.org/abs/1609.03705.

[57] John C Snyder, Matthias Rupp, Katja Hansen, Klaus-Robert Mu, and Kieron Burke. Finding Density Functionals with Machine Learning. *Ph*, 108:253002, 2012. doi: 10.1103/PhysRevLett.108. 253002.

[58] Felix Brockherde, Leslie Vogt, Li Li, Mark E Tuckerman, and Kieron Burke. By-passing the Kohn-Sham equations with machine learning. *arXiv*, 2017. URL http://arxiv.org/abs/1609. 02815.

[59] Juan Carrasquilla and Roger G. Melko. Machine learning phases of matter. *Nat. Phys.*, 13:431–434, 2017. doi: 10.1038/ nphys4035.

[60] Lei Wang. Discovering phase transitions with unsupervised learning. *Phys. Rev. B*, 94:195105, 2016. doi: 10.1103/PhysRevB. 94.195105.

[61] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355:602, 2017.

[62] Stephen R. Clark. Unifying Neural-network Quantum States and Correlator Product States via Tensor Networks. 2017. URL http://arxiv.org/abs/1710.03545.

[63] Ivan Glasser, Nicola Pancotti, Moritz August, Ivan D. Rodriguez, and J. Ignacio Cirac. Neural Networks Quantum States, String-Bond States and chiral topological states. *Phys. Rev. X*, 8:11006, 2017. doi: 10.1103/PhysRevX.8.011006. URL http://arxiv.org/abs/1710.04045.

[64] Dong Ling Deng, Xiaopeng Li, and S. Das Sarma. Quantum entanglement in neural network states. *Phys. Rev. X*, 7:1–17, 2017. doi: 10.1103/PhysRevX.7.021021.

[65] Xun Gao and Lu-Ming Duan. Efficient Representation of Quantum Many-body States with Deep Neural Networks. *Nat. Commun.*, pages 1–5, 2017. doi: 10.1038/s41467-017-00705-2. URL http://arxiv.org/abs/1701.05039{%}0Ahttp: //dx.doi.org/10.1038/s41467-017-00705-2.

[66] Raphael Kaubruegger, Lorenzo Pastori, and Jan Carl Budich. Chiral Topological Phases from Artificial Neural Networks. *arXiv*, 2017. URL http://arxiv.org/abs/1710.04713.

[67] Zi Cai and Jinguo Liu. Approximating quantum many-body wave-functions using artificial neural networks. 035116:1–8, 2017. doi: 10.1103/PhysRevB.97.035116. URL http://arxiv. org/abs/1704.05148.

[68] J. Han, L. Zhang, H. Wang, and W. E. Solving Many-Electron Schro2dinger Equation Using Deep Neural Networks. *ArXiv e-prints*.

[69] Serena Bradde and William Bialek. PCA Meets RG. *J. Stat. Phys.*, 167:462–475, 2017. doi: 10.1007/s10955-017-1770-6.

[70] Maciej Koch-Janusz and Zohar Ringel. Mutual Information, Neural Networks and the Renormalization Group. *arXiv*, 2017. URL http://arxiv.org/abs/1704.06279.

[71] Shuo-Hui Li and Lei Wang. Neural Network Renormalization Group. *arXiv*, 2018. URL http://arxiv.org/abs/1802.02840.

[72] Yi-Zhuang You, Zhao Yang, and Xiao-Liang Qi. Machine learning spatial geometry from entanglement features. *Phys. Rev. B*, 97:045153, Jan 2018. doi: 10.1103/PhysRevB.97.045153. URL https://link.aps.org/doi/10.1103/PhysRevB.97.045153.

[73] C. E. Rasmussen. Gaussian Processes to Speed up Hybrid Monte Carlo for Expensive Bayesian Integrals. *Bayesian Stat. 7*, pages 651–659, 2003. URL http://www.is.tuebingen.mpg.de/fileadmin/user{_}upload/files/publications/pdf2080.pdf.

[74] Simon Duane, A D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Phys. Lett. B*, 195:216–222, 1987. doi: https://doi.org/10.1016/0370-2693(87)91197-X. URL http://www.sciencedirect.com/science/article/pii/037026938791197X.

[75] Junwei Liu, Yang Qi, Zi Yang Meng, and Liang Fu. Self-learning Monte Carlo method. *Phys. Rev. B*, 95:1–5, 2017. doi: 10.1103/PhysRevB.95.041101.

[76] Li Huang, Yi Feng Yang, and Lei Wang. Recommender engine for continuous-time quantum Monte Carlo methods. *Phys. Rev. E*, 95:031301(R), 2017. doi: 10.1103/PhysRevE.95.031301.

[77] Li Huang and Lei Wang. Accelerated Monte Carlo simulations with restricted Boltzmann machines. *Phys. Rev. B*, 95:035105, 2017. doi: 10.1103/PhysRevB.95.035105.

[78] Junwei Liu, Huitao Shen, Yang Qi, Zi Yang Meng, and Liang Fu. Self-learning Monte Carlo method and cumulative update in fermion systems. *Phys. Rev. B*, 95:241104, 2017. doi: 10.1103/PhysRevB.95.241104.

[79] Faming Liang, Chuanhai Liu, and Raymond J Carroll. *Advanced Markov Chain Monte Carlo Methods: Learning from Past Samples*. Wiley, 2011.

[80] Jiaming Song, Shengjia Zhao, and Stefano Ermon. A-NICE-MC: Adversarial Training for MCMC. *arXiv*, 2017. URL http://arxiv.org/abs/1706.07561.

[81] Daniel Levy, Matthew D. Hoffman, and Jascha Sohl-Dickstein. Generalizing Hamiltonian Monte Carlo with Neural Networks. *arXiv*, 2017. URL http://arxiv.org/abs/1711.09268.

[82] Marco F. Cusumano-Towner and Vikash K. Mansinghka. Using probabilistic programs as proposals. *arXiv*, 2018. URL http://arxiv.org/abs/1801.03612.

[83] Troels Arnfred Bojesen. Policy Guided Monte Carlo: Reinforcement Learning Markov Chain Dynamics. *arXiv*, 2018. URL http://arxiv.org/abs/1808.09095.

[84] Lei Wang. Can Boltzmann Machines Discover Cluster Updates ? *arXiv*, 2017. doi: 10.1103/PhysRevE.96.051301. URL http://arxiv.org/abs/1702.08586.

[85] E. Miles Stoudenmire and David J. Schwab. Supervised Learning with Quantum-Inspired Tensor Networks. *arXiv*, 2016. URL http://arxiv.org/abs/1605.05775.

[86] Raphael Bailly. Quadratic weighted automata:spectral algorithm and likelihood maximization. In Chun-Nan Hsu and Wee Sun Lee, editors, *Proceedings of the Asian Conference on Machine Learning*, volume 20 of *Proceedings of Machine Learning Research*, pages 147–163, South Garden Hotels and Resorts, Taoyuan, Taiwain, 14–15 Nov 2011. PMLR. URL http://proceedings.mlr.press/v20/bailly11.html.

[87] Ming-Jie Zhao and Herbert Jaeger. Norm-observable operator models. *Neural Computation*, 22(7):1927–1959, 2010. doi: 10.1162/neco.2010.03-09-983. URL https://doi.org/10.1162/neco.2010.03-09-983. PMID: 20141473.

[88] W. Huggins, P. Patel, K. B. Whaley, and E. Miles Stoudenmire. Towards Quantum Machine Learning with Tensor Networks. *ArXiv e-prints*, March 2018.

[89] Yoav Levine, David Yakira, Nadav Cohen, and Amnon Shashua. Deep Learning and Quantum Entanglement: Fundamental Connections with Implications to Network Design. 2017. URL http://arxiv.org/abs/1704.01552.

[90] A. Cichocki, A-H. Phan, Q. Zhao, N. Lee, I. V. Oseledets, M. Sugiyama, and D. Mandic. Tensor Networks for Dimensionality Reduction and Large-Scale Optimizations. Part 2 Applications and Future Perspectives. *arXiv*, 2017. doi: 10.1561/2200000067. URL http://arxiv.org/abs/1708.09165.

[91] Scott Aaronson. Read the fine print. *Nat. Phys.*, 11:291–293, 2015. doi: 10.1038/nphys3272. URL http://dx.doi.org/10.1038/nphys3272.

[92] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549:195–202, 2017. doi: 10.1038/nature23474. URL http://dx.doi.org/10.1038/nature23474.

[93] Giacomo Torlai and Roger G. Melko. Neural Decoder for Topological Codes. *Phys. Rev. Lett.*, 119:030501, 2017. doi: 10.1103/PhysRevLett.119.030501.

[94] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. Many-body quantum state tomography with neural networks. *arXiv*, 2017. URL http://arxiv.org/abs/1703.05334.

[95] Xun Gao, Zhengyu Zhang, and Luming Duan. An efficient quantum algorithm for generative machine learning. *arXiv*, 2017. URL http://arxiv.org/abs/1711.02038.

[96] Marcello Benedetti, Delfina Garcia-Pintos, Yunseong Nam, and Alejandro Perdomo-Ortiz. A generative modeling approach for benchmarking and training shallow quantum circuits. *arXiv*, 2018. URL http://arxiv.org/abs/1801.07686.

[97] Marin Bukov, Alexandre G R Day, Dries Sels, Phillip Weinberg, Anatoli Polkovnikov, and Pankaj Mehta. Reinforcement Learning in Different Phases of Quantum Control. *arXiv*, 2017. URL http://arxiv.org/abs/1705.00565.

[98] Alexey A Melnikov, Hendrik Poulsen, Mario Krenn, Vedran Dunjko, and Markus Tiersch. Active learning machine learns to create new quantum experiments. pages 1–6, 2017. doi: 10.1073/pnas.1714936115.

[99] Louis-François Arsenault, O. Anatole von Lilienfeld, and Andrew J Millis. Machine learning for many-body physics: efficient solution of dynamical mean-field theory. *arXiv*, 2015. URL http://arxiv.org/abs/1506.08858.

[100] Louis-François Arsenault, Richard Neuberg, Lauren A Hannah, and Andrew J Millis. Projected regression method for solving Fredholm integral equations arising in the analytic continuation problem of quantum physics. *Inverse Probl.*, 33:115007, 2017. doi: 10.1088/1361-6420/aa8d93. URL http://stacks.iop.org/0266-5611/33/i=11/a=115007?key=crossref.a8698118ad7bdcd7dc901b78e3029959.

[101] Erin Ledell, Prabhat Dmitry, Yu Zubarev, Brian Austin, and William A Lester. Classification of nodal pockets in many-electron wave functions via machine learning. *J Math Chem*, 2012. doi: 10.1007/s10910-012-0019-5.

[102] Yuto Mori, Kouji Kashiwa, and Akira Ohnishi. Application of neural network to sign problem via path optimization method. 2017. URL http://arxiv.org/abs/1709.03208.

[103] Andrei Alexandru, Paulo Bedaque, Henry Lamm, and Scott Lawrence. Finite-Density monte carlo calculations on Sign-Optimized manifolds. 2018. URL http://arxiv.org/abs/1804.00697.

[104] Deniz Yuret. Knet : beginning deep learning with 100 lines of Julia. (Nips), 2016.

[105] Miguel A Carreira-Perpinan and Geoffrey E Hinton. On contrastive divergence learning. In *Aistats*, volume 10, pages 33–40, 2005. URL https://pdfs.semanticscholar.org/39eb/fbb53b041b97332cd351886749c0395037fb.pdf#page=42.

[106] Sandro Sorella. Variational monte carlo and markov chains for computational physics. In *Strongly Correlated Systems*, pages 207–236. Springer, 2013.

[107] Pedro Domingos. *The master algorithm*. Basic Books, 2015.

[108] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[109] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. URL http://arxiv.org/abs/1803.08823.