Deep learning by the lake

Lei Wang[*]

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

May 2, 2019

Abstract

This is an idiosyncratic introduction to deep learning
from a computational quantum physicists' perspective.
The focus is on deep learning's impacts to quantum many-
body computation, and vice versa.

This note is used for the Song Shan Lake Spring
School on "Deep Learning and Quantum Program-
ming". Associated tutorial codes can be found
at https://github.com/QuantumBFS/SSSS. Please send
comments, suggestions and corrections to the email ad-
dress in below.

[*] wanglei@iphy.ac.cn

CONTENTS

<div align="right"># 0</div>

RESOURCES

Before diving in, you may also consider other excellent resources which may fit better to your prior.

0.1 BOOKS

The book [1] is a good introductory read which guides you a tour of "five schools" of the machine learning. Classical texts on neural networks are [2] and [3]. Modern textbooks are [4, 5] and [6]. The book by David J. MacKay [7] is a great source for inspirations. More broadly on artificial intelligence (AI), there is [8].

0.2 ONLINE RESOURCES

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 421 "Neural Networks and Deep Learning" taught by Roger Grosse quite helpful. 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.

0.3 LECTURE NOTES AND REVIEW ARTICLES

- "Machine Learning for Physicists" by Florian Marquardt

- "A high-bias, low-variance introduction to Machine Learning for physicists" by Pankaj Mehta et al

- "Machine learning and the physical sciences" by Giuseppe Carleo et al

DEEP LEARNING: THE THEORETICAL MINIMUM

In general, there are four key components of the machine learning applications, *data*, *model*, *objective function* and *optimization*. We will see vast different machine learning algorithms for different tasks by combining various components.

Take the pattern recognition task as an example. The goal 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*.

There are numerous approaches to achieve this task in scikit-learn. However, the so called "no free lunch" theorem [9] 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 images, languages and many natural data. 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.

1.1 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 *un-*
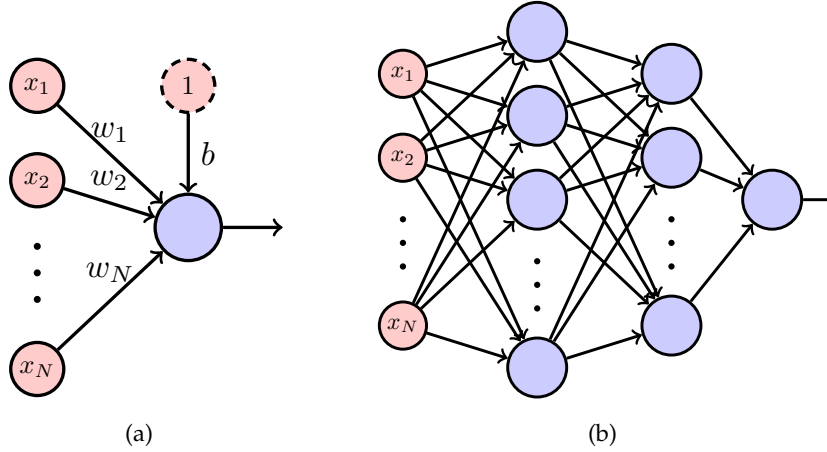
Data generation probability

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

normalized probability distribution (i.e. Boltzmann weight). This can lead to very different design choices of the learning algorithms.

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 [10]. 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.

Information complete

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.

Feature engineering

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

Data format

vector. While for quantum-many body problems, each row can be a snapshot sampled according to wavefunction, a quantum Monte 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.

1.2 MODEL: ARTIFICIAL NEURAL NETWORKS

An artificial neural network expresses high dimensional functions by composing simple building blocks. Many texts starts the introduction from biological inspired artificial neuron. However, to our purpose, it is better to take a geometric perspective. Typically, the data flow in a feedforward neural network alternates between linear affine transformation and element-wise nonlinear activation layers. These are in general two non-commuting operations. And both operations are necessary to model nontrivial functions. This general pattern of alternating between linear and nonlinear units also applies to more complicated neutral networks.

Composition of two linear transformation is still a linear transformation. Element-wise nonlinear transformation does not mix input variables.

The neural network 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 [11, 12]. 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 [13, 14].

Classical texts [2, 3] 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.*

Table 1 summarized typical nonlinear activation functions used in neural networks. They transform the signal in an elementwise fashion. For the activations used for the outputs, one choices the activation function with the suitable output range. For example, Linear, ReLU and softplus 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

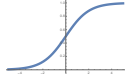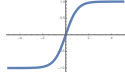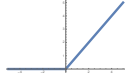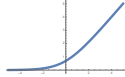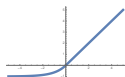$$x_{\nu}^{\ell+1} = \sum_{\mu} x_{\mu}^{\ell} W_{\mu,\nu} + b_{\nu}, \tag{1}$$

where $W_{\mu,\nu}$ and b_{ν} are the parameters. Since this layer mixes components with a dense matrix, it is also called the dense layer.

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

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

where the parameters are $W_{\mu,\nu,m,n}$ and b_{ν}. The convolutional kernel is a four dimensional tensor, which performs the matrix-vector multiplication in the channel space μ, ν 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,

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	$e^{z_i} / \sum_i e^{z_i}$	$(0,1)$	

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 [15] 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 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.

1.3 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.

___ **Info** _____

The sum and product rule of probabilities

$$p(A) = \sum_B p(A, B), \tag{4}$$

$$p(A, B) = p(B|A)p(A), \tag{5}$$

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

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

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,

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

For example, the L^2 norm of all weights $\Omega(\theta) = ||w||^2$. 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.
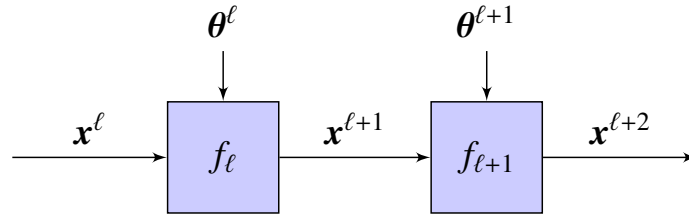
Figure 2: Layers of a feedforward network. Each layer is a function $x^{\ell+1} = f_\ell(\boldsymbol{\theta}^\ell, \boldsymbol{x}^\ell)$.

> **Info**
>
> 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 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.

Dropout, Data argumentation, Transfer learning

1.4 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.

1.4.1 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

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

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

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 layer. In practice, we compute the l.h.s. using information on the r.h.s., hence the name "back" propagation. Note that $\frac{\partial \boldsymbol{x}^{\ell+1}}{\partial \boldsymbol{x}^\ell}$ 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.

BackProp is neither symbolic differentiation nor numerical finite difference differentiation

Back Propagation is nothing but "reverse mode automatic differentiation" applied to neural network. See [16] for survey of automatic differentiation techniques applied to machine learning.

Vector-Jacobian multiplication is a recurring pattern in automatic differentiation. Think twice before actually allocating space for the Jacobian matrix. Do you really need itself ?

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 [17, 18].

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 $\ell = 0, \ldots, L-1$
  $x^{\ell=0} = x$
  for $\ell = 0, \cdots, L-1$ **do**   ▷ Feedforward pass
    $x^{\ell+1} = f_\ell(x^\ell, \theta^\ell)$
  end for
  Compute $\bar{x}^L = \frac{\partial \mathcal{L}}{\partial x^L}$ for the last layer
  for $\ell = L-1, \cdots, 0$ **do**   ▷ Backward pass
    Evaluate $\frac{\partial \mathcal{L}}{\partial \theta^\ell} = \bar{x}^{\ell+1} \frac{\partial x^{\ell+1}}{\partial \theta^\ell}$ ▷ Eq. (9)
    Evaluate $\bar{x}^\ell = \bar{x}^{\ell+1} \frac{\partial x^{\ell+1}}{\partial x^\ell}$ ▷ Eq. (10)
  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 [19], deep dream [20] and style transfer [21].*

Example

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

$$x_\nu^{\ell+1} \quad = \quad x_\mu^\ell W_{\mu\nu} + b_\nu, \tag{11}$$

$$x_\nu^{\ell+2} \quad = \quad \sigma(x_\nu^{\ell+1}). \tag{12}$$

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

$$\bar{x}_\mu^\ell \quad = \quad \bar{x}_\nu^{\ell+1} W_{\mu\nu}, \tag{13}$$

$$\bar{x}_\nu^{\ell+1} \quad = \quad \bar{x}_\nu^{\ell+2} \sigma'(x_\nu^{\ell+1}), \tag{14}$$

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

$$\frac{\partial \mathcal{L}}{\partial W_{\mu\nu}} \quad = \quad \bar{x}_\nu^{\ell+1} x_\mu^\ell, \tag{15}$$

$$\frac{\partial \mathcal{L}}{\partial b_\nu} \quad = \quad \bar{x}_\nu^{\ell+1}, \tag{16}$$

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

1.4.2 *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

Beyond SGD

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 [22]

$$v = \alpha v - \frac{\partial \mathcal{L}}{\partial \theta}, \tag{18}$$

$$\theta = \theta + v. \tag{19}$$

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.

Unsupervised pretraining

A typical difficulty with training deep neural nets is the gradient 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 [23], ResNet [15]), pure gradient based optimization is unimaginably successful and unsupervised pretraining became unnecessary (or, unfashionable) in training deep neural network.

Learning is NOT optimization

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.

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

DIFFERENTIABLE PROGRAMMING

Differentiable programming is a fresh programming paradigm which composes parameterized algorithmic components and trains them using automatic differentiation (AD). The concept emerges from deep learning, but is not limited to training neural networks. Here we dive a bit deeper into this technique and see how it means for scientific computation.

2.1 AUTOMATIC DIFFERENTIATION

Automatic differentiation mechanically computes derivatives of functions expressed in terms of computer programs [24]. Unlike numerical differentiation or symbolic differentiation, AD computes the value of the derivatives to the machine precision without generating a symbolic expression of the gradient function. Moreover, the performance of AD has a general theoretical guarantee, which does not exceed the algorithmic complexity of the original program [25, 26]. AD is the computational engine of modern deep learning applications [27, 28]. Moreover, AD also finds applications in quantum optimal control [29] and quantum chemistry calculations [30].

Central to the AD is the concept of the computation graph. A computation graph is a directed acyclic graph (DAG) composed by elementary computation steps. The nodes of the graph represent data, which can be scalars, vectors, matrices, or tensors. The graph connectivity indicates the dependence of the data flow in the computation process. The simplest computation graph is a chain shown in Fig 3(a). Starting from, say vector valued, input parameters θ one computes a series of intermediate results until reaching the final output $\mathcal{L}$, which we assume to be a scalar. The so called forward evaluation simply traverses the chain graph in sequential order $\theta \to T^1 \to \cdots \to T^n \to \mathcal{L}$.

To compute the gradient of the objective function with respect to input parameters, one can exploit the chain rule

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial T^n} \frac{\partial T^n}{\partial T^{n-1}} \cdots \frac{\partial T^2}{\partial T^1} \frac{\partial T^1}{\partial \theta}. \tag{20}$$

Since we consider the case where the input dimension is larger than the output dimension, it is more efficient to evaluate the gradient in (20) by multiplying terms from left to the right using a series of
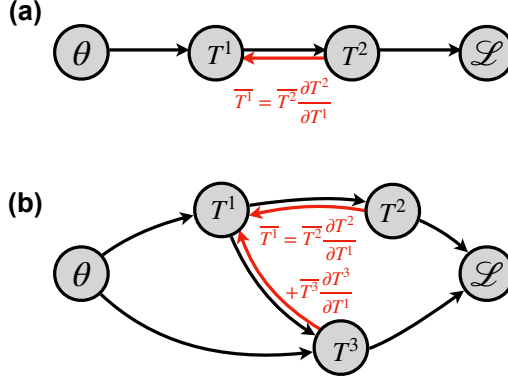
Figure 3: Reverse mode automatic differentiation on computation graphs. Black arrows indicate the forward function evaluation from inputs to outputs. Red arrows indicate backward steps for adjoint backpropagation. (a) A chain graph. (b) A more general computation graph. In the backward pass, the adjoint of a given node is computed according to Eq. (21).

vector-Jacobian products. In terms of the computation graph shown in Fig. 3(a), one traverses the DAG backward and propagates the gradient signal from the output back to the input. Computing the derivative this way is called *reverse mode* AD. This approach, commonly referred to as the backpropagation algorithm [27], is arguably the most successful method for training deep neural networks.

It is instructive to introduce the *adjoint variable* $\overline{T} = \partial \mathcal{L}/\partial T$ to denote the gradient of the final output $\mathcal{L}$ with respect to the variable T. One sees that the reverse mode AD propagates the adjoint from $\overline{T^n} = \overline{\mathcal{L}} \frac{\partial \mathcal{L}}{\partial T^n}$ with $\overline{\mathcal{L}} = 1$ all the way back to $\overline{T^i} = \overline{T^{i+1}} \frac{\partial T^{i+1}}{\partial T^i}$ with $i = n-1, \ldots, 1$, and finally computes $\overline{\theta} = \overline{T^1} \frac{\partial T^1}{\partial \theta}$. In each step, one propagates the adjoint backward via a local vector-Jacobian product.

The adjoint backpropagation picture generalizes well to more complex computation graphs. In general, the DAG contains both fan-out and fan-in, which correspond to reusing data from previous steps in a downstream computation. Therefore, a data node, such as T^1 in Fig. 3(b), can affect the final outcome via several different paths. And in the backward pass one needs to accumulate all contributions from the child nodes to obtain the adjoint of the node

$$\overline{T^i} = \sum_{j:\,\text{child of } i} \overline{T^j} \frac{\partial T^j}{\partial T^i}. \tag{21}$$

The reverse mode AD algorithm can be understood as a message passing process on the DAG. After a topological sort of the computation graph defined by the forward pass, one visits the graph backward from the output node with adjoint $\overline{\mathcal{L}} = 1$. Each node collects information from its child nodes to compute its own adjoint, then

passes this information to its parents. Thus, one can compute the gradient with respect to all parameters in one forward and one backward pass. Typically one caches necessary information in the forward pass for efficient evaluation of the vector-Jabobian product in the backward pass.

The building blocks of a differentiable program are called *primitives*. The primitives can be elementary operations such as addition, multiplication, and math functions [31]. Each primitive has an associated backward function in addition to the ordinary forward function. The backward function backpropagates the adjoints according to the vector-Jacobian product Eq. (21). Note that one does not need to explicitly instantiate or store the full Jacobian matrices. Moreover, one can group many elementary computation steps together as a primitive. For example, the linear algebra operations such as matrix multiplication can be regarded as a primitive. In this way, the forward pass of these customized primitive can be operated as a black box. Designing the differentiable program in such a modular way allows one to control the level of granularity of AD. There are several advantages of crafting customized primitives for domain specific problems. First, this can reduce the computation and memory cost. Second, in some cases, it is numerically more stable by grouping several steps together. Third, one can wrap function calls to external libraries into primitives, without the need to track each individual computation step.

Modern machine learning frameworks such as `TensorFlow` [32] and `PyTorch` [33] support AD either by explicitly constructing the computation graphs or tracking the program execution order at run time. Besides math functions, one can also differentiate through control flows, loops and function calls. Building on these infrastructures, *differentiable programming* is emerging as a new programming paradigm that emphasizes assembling differentiable components and learning the whole program via end-to-end optimization [28]. Letting machines deal with mechanical AD has greatly reduced laborious human efforts and reallocated human creativity to design more sophisticated deep learning models.

Finally, we note that it is also possible to evaluate Eq. (20) from right to left, which corresponds to the *forward mode* AD. The operation of the forward mode AD is akin to the perturbation theory. One can compute the objective function and the gradient in a single forward pass without storing any intermediate results. However, the forward mode AD is not favorable for computation graphs whose input dimension is much larger than the output dimension [28]. Therefore, the majority of deep learning work employs the reverse mode AD.

To compute gradients of a program using reverse mode AD, one needs to trace the composition of the primitive functions and propagate the adjoint information backward on the computation graph. Thankfully, modern differentiable programming frameworks [32–35] have taken care of tracing and backpropagation for their basics data structure, differentiable tensors, automatically. Usually, one only needs to focus on the forward pass without worrying about how the gradient is computed. In more advanced user cases, on needs to identifying suitable primitives and define customized vector-Jacobian products.

There are, however, several other aspects deserves one's attention for stable, efficient, and scalable backpropagation. It turns out having a deep understanding of the following topics allows one to push the limit of the current deep learning frameworks.

2.2.1 *Memory efficient reverse mode AD with checkpointing function*

A straightforward implementation of the reverse mode AD for long programs will consume large memory. This is because one needs to store all intermediate results in the forward pass for evaluating the vector-Jacobian products in the backward pass. The number of stored variables is related to the level of granularity in the AD implementation. In any case, the memory consumption of reverse mode AD will be proportional to the depth of the computation graph. This is particularly worrying for tensor networks with large bond dimensions and a large number of renormalization iterations.

The solution to the memory issue of reverse mode AD is a well known technique called checkpointing [31, 39]. The idea is to trade the computational time with memory usage. Taking the chain computation graph in Eq. (20) as an example, one stores the tensor every a few steps in the forward process. And in the backward pass, one re-computes intermediate tensors whenever needed by running a small segment of the computation graph forwardly. In this way, one can greatly reduce the memory usage with no more than twice of the computational effort.

In a deeper understanding, checkpointing amounts to define customized primitives which encapsulates a large part of the computation graph. These primitives have their own special backward rules which locally runs the forward pass again and then backpropagates the adjoint. Therefore, in the forward pass one does not need to cache internal states of these checkpointing primitives.

The checkpointing is a general strategy that is applied to the computation graph of any topological structure. For example, when applied to tensor network algorithms, it is natural to regard the renor-

malization steps as checkpointing primitives. In this way, one avoids storing some large intermediate tensors in the forward pass.

2.2.2 *Backward through fixed point iteration*

Fixed point iteration is a recurring pattern in tensor network algorithms. For example, one iterates the function $T^{i+1} = f(T^i, \theta)$ until reaching a converged tensor T^* and uses it for downstream calculations. To compute the gradient with respect to the parameter θ, one can certainly unroll the iteration to a deep computation graph and directly apply the reverse mode AD. However, this approach has the drawback of consuming large memory if it takes long iterations to find the fixed point.

One can solve this problem by using the implicit function theorem on the fixed point equation [40]. Taking the derivative on both sides of $T^* = f(T^*, \theta)$, we have

$$\bar{\theta} = \overline{T^*} \frac{\partial T^*}{\partial \theta} = \overline{T^*} \left[I - \frac{\partial f(T^*, \theta)}{\partial T^*} \right]^{-1} \frac{\partial f(T^*, \theta)}{\partial \theta}$$

$$= \sum_{n=0}^{\infty} \overline{T^*} \left[\frac{\partial f(T^*, \theta)}{\partial T^*} \right]^n \frac{\partial f(T^*, \theta)}{\partial \theta}. \tag{22}$$

The second line expands the matrix inversion in the square bracket as a geometric series. Therefore, to backpropagate through a fixed point iteration the basic operation is just the vector-Jacobian products involving the single step iteration function. And one performs iteration in the backward function to accumulate the gradient $\bar{\theta}$ until convergence. The geometric series show the same convergence rate to the fixed point as the forward iteration [40]. Since the backward iteration is now independent of the forward step, one avoids caching a long chain of intermediate results in the forward iteration.

2.2.3 *Backprop for ODE solver*

Ordinary differential equations (ODE) are ubiquitous in science and engineering. Moreover, several modern deep neural network can be regarded as discretization of the ODE integrations. Backpropagation through ODE solver can be hugely useful for deep learning as well as more conventional study.

Consider an ODE

$$\frac{d\boldsymbol{x}}{dt} = f(\boldsymbol{x}, \boldsymbol{\theta}, t). \tag{23}$$

To solve it we integrate the equation from time 0 to T. Assuming then one has certain downstream scalar loss $\mathcal{L}$ depending on the final state $\boldsymbol{x}(T)$. To compute its gradient with respect to the parameter $\boldsymbol{\theta}$, one can certainly directly apply the ordinary reverse mode AD to the

existing ODE solver. However, this will invoke large memory Consumption. Using the checkpointing approach discussed in Sec. 2.2.1 may alleviated this issue. Although, there is a more elegant solution to this by deriving the ODE satisfied by the adjoint themselves [41].

Define the adjoint $\overline{x}(t) = \frac{\partial \mathcal{L}}{\partial x(t)}$. Since $\overline{x}(t) = \overline{x}(t+\epsilon)\frac{\partial x(t+\epsilon)}{\partial x(t)} = \overline{x}(t+\epsilon)\left[1 + \epsilon\frac{\partial f(x,\theta,t)}{\partial x(t)}\right]$, the adjoint satisfies another set of ODE

$$\frac{d\overline{x}(t)}{dt} = -\overline{x}(t)\frac{\partial f(x,\theta,t)}{\partial x}. \tag{24}$$

Thus, one can obtain the adjoint by setting $\overline{x}(T) = \frac{\partial \mathcal{L}}{\partial x(T)}$ and integrating backwards in time to 0. In this way, one does not need to store anything on the forward pass.

Afterwards, one has

$$\frac{\partial \mathcal{L}}{\partial \theta} = \int_0^T dt\, \overline{x}(t)\frac{\partial f(x,\theta,t)}{\partial \theta}. \tag{25}$$

2.2.4 *Backprop for linear algebra operations*

The key components of neural network and tensor network algorithms are the matrix and tensor algebras. And there are established results on backward through these operations [36–38]. First of all, it is straightforward to wrap all `BLAS` routines as primitives with customized backward functions. Next, it is also less trivial to derive backward rules for many `LAPACK` routines such as the eigensolver, SVD, and QR factorization [36]. By treating these linear algebra operations as primitives, one can rely on efficient implementations of matrix libraries in a differentiable program.

There are a few practical obstacles to stable and scalable implementation of differentiable tensor network programs. First, the backward for the eigensolver and SVD may face numerical instability with degeneracy in the eigenvalues or singular values. Second, the reverse mode AD may incur large memory consumption, which prevents one from reaching the same bond dimension of an ordinary tensor network program. We present solutions to these problems in below. We present several key results on matrix derivatives involving linear algebra operations that are relevant to tensor network algorithms. Recall the modular nature of reverse mode AD, one just needs to specify the local backward function to integrate these components into a differentiable program. We will comment on their connections to physics literature and pay special attention to stable numerical implementations [42]. For more information, one can refer to [36–38].

Symmetric eigensolver

The forward pass reads $A = UDU^T$, where the diagonal matrix D are the eigenvalues and each column of the orthogonal matrix U

is a corresponding eigenvector. This is a fan out in the computation graph from A to U and D.

In the backward pass, given the adjoint $\overline{U}$ and $\overline{D}$, we have [36, 38]

$$\overline{A} = U \left[\overline{D} + F \odot (\overline{U}^T U - U^T \overline{U})/2 \right] U^T, \tag{26}$$

where $F_{ij} = (D_j - D_i)^{-1}$ if $i \neq j$ and zero otherwise. The symbol $\odot$ denotes an element-wise Hardmard product. One can readily check that the gradient is also a symmetric matrix.

Equation (26) can be regarded as "reverse" perturbation theory. When the downstream calculation does not depend on the eigenstate, i.e. $\overline{U} = 0$, the backward equation is related to the celebrated Hellmann-Feynman theorem [43] which connects the perturbation to the Hamiltonian and its eigenvalues. Ref. [44] applied this special case of Eq. (26) for inverse Hamiltonian design based on energy spectra.

The appearance of the eigenvalue difference in the denominator of F is a reminder of the first order nondegenerate perturbation theory. Reference [30] concerned about the stability of the backward pass through the eigensolver, thus turned to less efficient forward mode AD for variational Hartree-Fock calculations of quantum chemistry problems. We found that by using a Lorentzian broadening with $1/x \to x/(x^2 + \varepsilon)$ with $\varepsilon = 10^{-12}$, one can stabilize the calculation at the cost of introducing a small error in the gradient, see also [38].

Lastly, very often in scientific computation one only access a few eigenstates with iterative solvers (such as Lanczos or Arnoldi). Even in this case, it is possible to carry out efficient reverse mode AD even in this case. The details are a bit tricky, we refer to the MIT 18.336 course note by.Steven G. Johnson for details.

Singular value decomposition

A ubiquitous operation in tensor network algorithms is the matrix SVD, which is used for canonicalization and factorization of tensor networks [45–47]. The forward pass reads $A = UDV^T$., where A is of the size (m, n), and U, V^T has the size (m, k) and (k, n) respectively, and $k = \min(m, n)$. In the reverse mode AD, given the adjoints $\overline{U}, \overline{D}$ and $\overline{V}$, one can obtain [37]

$$\begin{aligned} \overline{A} \quad &= \tfrac{1}{2} U \left[F_+ \odot \left(U^T \overline{U} - \overline{U}^T U \right) + F_- \odot \left(V^T \overline{V} - \overline{V}^T V \right) \right] V^T \\ &\quad U \overline{D} V^T + (I - UU^T) \overline{U} D^{-1} V^T + U D^{-1} \overline{V}^T (I - VV^T), \end{aligned} \tag{27}$$

where $[F_\pm]_{ij} = \frac{1}{D_j - D_i} \pm \frac{1}{D_j + D_i}$ for $i \neq j$ and zero otherwise. To prevent the numerical issue in case of degenerate singular values, we use the same Lorentzian broadening as Sec. 2.2.4 for the first term, which works well in our experience. In practice, for variational tensor network calculation starting from random tensors, the chance of

having exact degenerate eigenvalues is small. And even if this happens, applying the rounding is a reasonable solution. While for the cases of degeneracy due to intrinsic reasons [48–51], one will still obtain the correct gradient as long as the end-to-end gradient is well defined. Lastly, inverting the vanishing singular values in Eq. (27) is not a concern since the corresponding space is usually truncated.

QR factorization

QR factorization is often used for canonicalization of tensor networks [45–47]. In the forward pass, one factorizes $A = QR$, where $Q^T Q = I$ and R is an upper triangular matrix [1]. Depending on the dimensions (m, n) of the matrix A there are two cases for the backward function.

For input shape of A matrix $m \geq n$, R is a $n \times n$ matrix. The backward pass reads [38]

$$\overline{A} = \left[\overline{Q} + Q\, \mathtt{copyltu}(M)\right] R^{-T}, \tag{28}$$

where $M = R\overline{R}^T - \overline{Q}^T Q$ and the `copyltu` function generates a symmetric matrix by copying the lower triangle of the input matrix to its upper triangle, $[\mathtt{copyltu}(M)]_{ij} = M_{\max(i,j),\min(i,j)}$. The multiplication to R^{-T} can be dealt with by solving a linear system with a triangular coefficient matrix.

For the case of $m < n$, Q is of the size (m, m), and R is a $m \times n$ matrix. We denote $A = (X, Y)$ and $R = (U, V)$, where X and U are full rank square matrices of size (m, m). This decomposition can be separated into two steps, first $X = QU$ uniquely determines Q and U, then we calculate $V = Q^T Y$. Applying the chain rule, the backward rule gives

$$\overline{A} = \left(\left[(\overline{Q} + \overline{V}Y^T) + Q\, \mathtt{copyltu}(M)\right] U^{-T}, Q\overline{V}\right), \tag{29}$$

where $M = X\overline{X}^T - (\overline{Q} + \overline{V}Y^T)^T Q$.

2.3   SOFTWARE SUPPORT

Software support for differentiable programming is fast developing with strong demand from deep learning application. Table 2 summarizes some key features of current deep learning frameworks.

---

[1] One can also require the diagonal of the R matrix to be positive to fix the redundant gauge degrees of freedom. This can be easily implemented on top of the existing QR libraries with backward support.

	Linalg AD	Complex	GPU
PyTorch	✓	✗	✓
TensorFlow	✓	✓	✓
Autograd	✓	✓	✗
Jax	✎	✓	✓
Flux.jl	✗	✗	✓
Zygote.jl	✗	✓	✓

Table 2: Software support for differentiable programming. The ✎ symbol indicates under development.

GENERATIVE MODELING

What I can not create, I do not understand

from Richard Feynman's
blackboard

Deep learning is more than discriminative tasks such as classification or regression (pattern recognition). Generative modeling, by its name, it is about generating new instances from the learned probability distribution. Generative models are fun, useful but also challenging. Thus, they are at the forefront of deep learning research [52].

3.1 PROBABILISTIC GENERATIVE MODELING

The goal of generative modeling is to *represent*, *learn* and *sample* from high-dimensional probability distributions. Given data x and label y, generative models capture the joint probability distribution $p(x,y)$. A well trained generative model can support discriminative tasks 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.

For simplicity, let us focus on *density estimation* first, where the goal is to model the joint probability $p(x)$ of a given an unlabelled dataset $\mathcal{D} = \{x\}$. Information theory consideration defines an objective function for this task. The Kullback-Leibler (KL) divergence reads

$$\mathbb{KL}(\pi||p) = \sum_x \pi(x) \ln\left[\frac{\pi(x)}{p(x)}\right], \tag{30}$$

which measures the dissimilarity between two probability distributions. We have $\mathbb{KL} \geq 0$ due to the Jensen inequality. The equality is achieved only when the two distributions match exactly. The KL-divergence is not symmetric with respect to its arguments. So it is not a proper distance measure. $\mathbb{KL}(\pi||p)$ 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 [6].

> **Info**
>
> The Jensen inequality [7] states that for convex $\smile$ functions f
>
> $$\langle f(x) \rangle \geq f(\langle x \rangle). \tag{31}$$
>
> 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

$$\mathcal{L} = -\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \ln[p(x)]. \tag{32}$$

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 the network?

Minimizing the NLL is a prominent (but not the only) way to train generative models, also known as Maximum Likelihood Estimation (MLE). The Eq. (32) appears to be a minor change compared to the discriminative task. 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.

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

"Solving" means: 1. compute expected value of physical observables, 2. sampling configurations at various temperatures, 3. compute marginal probabilities given arbitrary subgroup of variables

Density estimation on data is not the only thing physicists wanted to do. For example, in statistical physics one wants to solve the problem given bare energy function. Generative models can also help us on that with variational calculation by minimizing the reverse KL-divergence. Consider a statistical physics problem where $\pi(x) = e^{-E(x)}/\mathcal{Z}$ and $\mathcal{Z} = \sum_x e^{-E(x)}$. We try to minimize the free

energy $-\ln \mathcal{Z}$, which is unfortunately intractable in general. To proceed, we define a variational free energy

$$\mathcal{L} = \sum_{x} q(x) \ln \left[\frac{q(x)}{e^{-E(x)}} \right] = \langle E(x) + \ln q(x) \rangle_{x \sim q(x)} \qquad (33)$$

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

$$\mathcal{L} + \ln \mathcal{Z} = \mathbb{KL}(q||\pi) \geq 0, \qquad (34)$$

thus Eq. (33) 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 distribution. Equation (34) is known as Gibbs-Bogoliubov-Feynman inequality in physics.

The *density estimation* and *variational calculation* examples are by no means the only two applications of the generative models. Since they capture the whole distribution, by defining and adjusting the suitable cost function you can use generative model for creative applications, or enhance performance of downstream tasks. One thing we already see is that the requirement on the generative models are task dependent. For density estimation, we need to efficiently compute the likelihood (or at least its gradient given data). While for variational calculation, we need to be able to efficiently sample from the model as well.

3.2 GENERATIVE MODEL ZOO

This section we review several representative generative models. The key idea is to impose certain structural prior in the probabilistic model. 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 many 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

$$p(x) = \frac{e^{-E(x)}}{\mathcal{Z}}, \qquad (35)$$

where $E(x)$ 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. Density estimation of binary data is related to the so

called inverse Ising problem. Exploiting the maximum log-likelihood estimation, the gradient of Eq. (32) is

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left\langle \frac{\partial E(\boldsymbol{x})}{\partial \boldsymbol{\theta}} \right\rangle_{\boldsymbol{x} \sim \mathcal{D}} - \left\langle \frac{\partial E(\boldsymbol{x})}{\partial \boldsymbol{\theta}} \right\rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})}. \tag{36}$$

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 $E = -\frac{1}{2} W_{ij} x_i x_j$, the gradient Eq. (36) can be simply evaluated. And the gradient descent update
>
> $$W_{ij} = W_{ij} + \eta \left(\langle x_i x_j \rangle_{\boldsymbol{x} \sim \mathcal{D}} - \langle x_i x_j \rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})} \right). \tag{37}$$
>
> 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 [53] initialize the chain with a sample drawn 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 [54] algorithm use the sample from last step to initialize the Monte Carlo chain. The logic being that the gradient descent update of the model is small anyway, so accumulation of the Monte Carlo samples helps mixing. In practice, one run a batch of the Monte Carlo chains in parallel to estimate the expected value of the negative phase.

Exercise 5 (**Mind the Gradient**). *Define* $\Delta = \langle E(\boldsymbol{x}) \rangle_{\boldsymbol{x} \sim \mathcal{D}} - \langle E(\boldsymbol{x}) \rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})}$. *How is its gradient with respect to* $\boldsymbol{\theta}$ *related to Eq. (36) ?*

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

$$p(\boldsymbol{x}) = \frac{1}{\mathcal{Z}} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{x}, \boldsymbol{h})}. \tag{38}$$
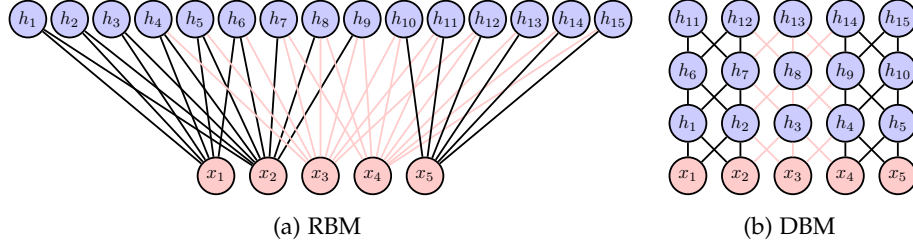
(a) RBM   (b) DBM

Figure 4: 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 [55].

This is equivalent to say that $E(x) = -\ln \sum_h e^{-E(x,h)}$ in Eq. (35), which can be quite complex even for simple joint energy function $E(x,h)$. Differentiating the equation, we have

$$\frac{\partial E(x)}{\partial \theta} = \frac{\sum_h e^{-E(x,h)} \frac{\partial E(x,h)}{\partial \theta}}{\sum_h e^{-E(x,h)}} = \sum_h p(h|x) \frac{\partial E(x,h)}{\partial \theta}, \qquad (39)$$

Therefore, in the presence of the hidden variables the gradient in Eq. (36) becomes

$$\frac{\partial \mathcal{L}}{\partial \theta} = \left\langle \frac{\partial E(x,h)}{\partial \theta} \right\rangle_{x \sim \mathcal{D}, h \sim p(h|x)} - \left\langle \frac{\partial E(x,h)}{\partial \theta} \right\rangle_{(x,h) \sim p(x,h)}, \qquad (40)$$

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) strives to have a balanced expressibility and learnability. The energy function reads

$$E(x,h) = -\sum_i a_i x_i - \sum_j b_j h_j - \sum_{i,j} x_i W_{ij} h_j. \qquad (41)$$

Since the RBM is defined on a bipartite graph shown in Fig. 4(a), its conditional probability distribution factorizes $p(h|x) = \prod_j p(h_j|x)$ and $p(x|h) = \prod_i p(x_i|h)$, where

$$p(h_j = 1|x) = \sigma \left(\sum_i x_i W_{ij} + b_j \right), \qquad (42)$$

$$p(x_i = 1|h) = \sigma \left(\sum_j W_{ij} h_j + a_i \right). \qquad (43)$$

This means that given the visible units we can directly sample the hidden units in parallel, vice versa. Sampling back and forth between

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.

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.

Info

For an RBM, one can actually trace out the hidden units in the Eq. (35) analytically and obtain

$$E(\boldsymbol{x}) = -\sum_i a_i x_i - \sum_j \ln(1 + e^{\sum_i x_i W_{ij} + b_j}). \qquad (44)$$

This can be viewed as a Boltzmann Machine with fully visible units whose energy function has a softplus interaction. Using Eq. (36) and Eq. (44) one can directly obtain

$$-\frac{\partial \mathcal{L}}{\partial a_i} = \langle x_i \rangle_{\boldsymbol{x} \sim \mathcal{D}} - \langle x_i \rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})}, \qquad (45)$$

$$-\frac{\partial \mathcal{L}}{\partial b_j} = \big\langle p(h_j = 1|\boldsymbol{x}) \big\rangle_{\boldsymbol{x} \sim \mathcal{D}} - \big\langle p(h_j = 1|\boldsymbol{x}) \big\rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})}, \qquad (46)$$

$$-\frac{\partial \mathcal{L}}{\partial W_{ij}} = \big\langle x_i p(h_j = 1|\boldsymbol{x}) \big\rangle_{\boldsymbol{x} \sim \mathcal{D}} - \big\langle x_i p(h_j = 1|\boldsymbol{x}) \big\rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})}. \qquad (47)$$

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. (40) and Eqs. (45-47), please convince yourself that* $\big\langle x_i p(h_j = 1|\boldsymbol{x}) \big\rangle_{\boldsymbol{x} \sim \mathcal{D}} = \big\langle x_i h_j \big\rangle_{\boldsymbol{x} \sim \mathcal{D}, \boldsymbol{h} \sim p(\boldsymbol{h}|\boldsymbol{x})}$ *and* $\big\langle x_i p(h_j = 1|\boldsymbol{x}) \big\rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})} = \big\langle x_i h_j \big\rangle_{(\boldsymbol{x},\boldsymbol{h}) \sim p(\boldsymbol{x},\boldsymbol{h})}$. *The former ones are improved estimators with reduced variances. In statistics this is known as the Rao-Blackwellization trick. Remember that: 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. 4(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 [55]. However, the downside of DBMs is that they are even harder to train and sample due the interactions among the hidden units [56].

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.,

$$p(\boldsymbol{x}) = \prod_i p(x_i|\boldsymbol{x}_{<i}). \tag{48}$$

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 datum is easily computed as

$$\ln p(\boldsymbol{x}) = -\frac{1}{2} \sum_i \left(\left(\frac{x_i - \mu_i}{\sigma_i} \right)^2 + \ln(2\pi\sigma_i) \right). \tag{49}$$

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

$$x_i = \sigma_i(\boldsymbol{x}_{<i})\epsilon_i + \mu_i(\boldsymbol{x}_{<i}). \tag{50}$$

Info

A slightly awkward but very enlightening way to compute the log-likelihood of the autoregressive model is to treat Eq. (50) as an invertible mapping between $\boldsymbol{x}$ and $\boldsymbol{\epsilon}$, and invoke the probability transformation

$$\ln p(\boldsymbol{x}) = \ln \mathcal{N}(\boldsymbol{\epsilon}; \boldsymbol{0}, \boldsymbol{1}) - \ln \left| \det \left(\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\epsilon}} \right) \right|. \tag{51}$$

Notice that Jacobian matrix is triangular, whose determinant can be easily computed to be Eq. (49). Generalizing this idea to more complex bijective transformations bring us to a general class of generative models called *Normalizing Flow* [57–65]. 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 [62]) and speech synthesis (WaveNet [63]). 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. (50) is considered to be slow since they can not take advantage of modern hardware. Nevertheless, the generative process Eq. (50) is *direct* sampling, which is can be more efficient compared to the Gibbs sampling of Boltzmann Machines.

Info

The inverse autoregressive flow (IAF) [61] changes the transformation Eq. (50) to be

$$x_i = \sigma_i(\boldsymbol{\epsilon}_{<i})\epsilon_i + \mu_i(\boldsymbol{\epsilon}_{<i}), \tag{52}$$

so that one can generate the data in parallel. The log-likelihood of the generated data also follows Eq. (49). 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 [61]. Recently, DeepMind use an IAF (Parallel WaveNet) [66] to learn the probability density of an autoregressive flow (WaveNet) [63], thus to improve the speech synthesis speed to meet the needs in real-world applications [67]. To train the parallel WaveNet, they minimize the *Probability Density Distillation* loss $\mathbb{KL}(p_{\text{IAF}}||p_{\text{AF}})$ [66] 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 [57–65]

$$\ln p(x) = \ln q(z) - \ln\left|\det\left(\frac{\partial x}{\partial z}\right)\right|. \tag{53}$$

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. (53) 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 bi-

Bijectors are modular

jector [68, 69]. 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.

___ Example ___

As an example of Eq. (53), consider the famous Box-Muller transformation which maps a pair of uniform random variables z to Gaussian random variables x

$$\begin{cases} x_1 = \sqrt{-2\ln z_1}\cos(2\pi z_2), \\ x_2 = \sqrt{-2\ln z_1}\sin(2\pi z_2). \end{cases} \tag{54}$$

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 $q(x) = p(z)/\left|\det\left(\frac{\partial x}{\partial z}\right)\right| = \frac{1}{2\pi}\exp\left(-\frac{1}{2}(x_1^2 + x_2^2)\right)$. 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) [60] 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

$$\begin{cases} x_<^\ell = x_<^\ell \\ x_>^\ell = x_>^\ell \odot e^{s_\ell(x_<^\ell)} + t_\ell(x_<^\ell) \end{cases} \tag{55}$$

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

$$\ln q(x) = \ln p(z) - \sum_{\ell,i}(s_\ell)_i, \tag{56}$$

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

From the above discussion, one sees that the crucial point of design flow-based generative model is to balance the expressibility and the computational effort of the Jacobian calculation. Typically, this is implemented via imposing internal structure of the network [57–65]. Recently, References shows that is is possible to take the "continuous-time" limit of the flow, which relax the structure constrain [41, 70]. Interestingly, the resulting generative model can be viewed as integrating an ordinary differential system for the data and its likelihood.

Reference [71] further revealed the optimal transport perspective of continuous-time normalizing flow and discussed means to impose the physical symmetries in the generating process. Consider latent variables z and physical variables x both living in $\mathbb{R}^N$. Given a diffeomorphism between them, $x = x(z)$, the probability densities in the latent and physical spaces are related by $p(z) = q(x) \left| \det\left(\frac{\partial x}{\partial z} \right) \right|$. The Brenier theorem [72] implies that instead of dealing with a multivariable generative map, one can consider a scalar valued generating function $x = \nabla u(z)$, where the convex Brenier potential u satisfies the Monge-Ampère equation [73]

Monge-Ampère Flow

$$\frac{p(z)}{q(\nabla u(z))} = \det\left(\frac{\partial^2 u}{\partial z_i \partial z_j} \right). \tag{57}$$

Given the densities p and q, solving the Monge-Ampère equation for u turns out to be challenging due to the nonlinearity in the determinant. Moreover, for typical machine learning and statistical physics problems, one faces an additional challenge that one does not even have direct access to both probability densities p and q. Instead, one only has independent and identically distributed samples from one of them, or one only knows one of the distributions up to a normalization constant. Therefore, solving the Brenier potential in these contexts is a control problem instead of a boundary value problem. An additional computational challenge is that even for a given Brenier potential, the right-hand side of (57) involves the determinant of the Hessian matrix, which scales unfavorably as $\mathcal{O}(N^3)$ with the dimensionality of the problem.

To address these problems, we consider the Monge-Ampère equation in its *linearized form*, where the transformation is infinitesimal [74]. We write the convex Brenier potential as $u(z) = ||z||^2/2 + \epsilon\varphi(z)$, thus $x - z = \epsilon\nabla\varphi(z)$, and correspondingly $\ln q(x) - \ln p(z) = -\text{Tr}\ln\left(I + \epsilon\frac{\partial^2\varphi}{\partial z_i\partial z_j} \right) = -\epsilon\nabla^2\varphi(z) + \mathcal{O}(\epsilon^2)$. In the last equation we expand the logarithmic

function and write the trace of a Hessian matrix as the Laplacian operator. Finally, taking the continuous-time limit $\epsilon \to 0$, we obtain

$$\frac{d\boldsymbol{x}}{dt} = \nabla \varphi(\boldsymbol{x}), \tag{58}$$

$$\frac{d \ln p(\boldsymbol{x}, t)}{dt} = -\nabla^2 \varphi(\boldsymbol{x}), \tag{59}$$

such that $\boldsymbol{x}(0) = \boldsymbol{z}$, $p(\boldsymbol{x}, 0) = p(\boldsymbol{z})$, and $p(\boldsymbol{x}, T) = q(\boldsymbol{x})$, where t denotes continuous-time and T is a fixed time horizon. For simplicity here, we still keep the notion of $\boldsymbol{x}$, which now depends on time. The evolution of $\boldsymbol{x}$ from time $t = 0$ to T then defines our generative map.

The two ordinary differential equations (ODEs) compose a dynamical system, which describes the flow of continuous random variables and the associated probability densities under iterative change-of-variable transformation. One can interpret equations (58) and (59) as fluid mechanics equations in the Lagrangian formalism. Equation (58) describes the trajectory of fluid parcels under the velocity field given by the gradient of the potential function $\varphi(\boldsymbol{x})$. While the time derivative in (59) is understood as the "material derivative" [75], which describes the change of the local fluid density $p(\boldsymbol{x}, t)$ experienced by someone travels with the fluid.

The fluid mechanics interpretation is even more apparent if we write out the material derivative in (59) as $d/dt = \partial/\partial t + d\boldsymbol{x}/dt \cdot \nabla$, and use (58) to obtain

$$\frac{\partial p(\boldsymbol{x}, t)}{\partial t} + \nabla \cdot [p(\boldsymbol{x}, t)\boldsymbol{v}] = 0, \tag{60}$$

which is the continuity equation of a *compressible fluid* with density $p(\boldsymbol{x}, t)$ and velocity field $\boldsymbol{v} = \nabla \varphi(\boldsymbol{x})$. Obeying the continuity equation ensures that the flow conserves the total probability mass. Moreover, the velocity field is curl free $\nabla \times \boldsymbol{v} \equiv 0$ and the fluid follows a form of *gradient flow* [76]. The irrotational flow matches one's heuristic expectation that the flow-based generative model transports probability masses.

It should be stressed that although we use the optimal transport theory to motivate model architecture design, i.e., the gradient flow for generative modeling, we do not have to employ the optimal transport objective functions for the control problem. The difference is that in generative modeling one typically fixes only one end of the probability density and aims at learning a suitable transformation to reach the other end. While for optimal transport one has both ends fixed and aims at minimizing the transportation cost. References [77–79] adapted the Wasserstein distances in the optimal transport theory as an objective function for generative modeling.

3.2.4 *Variational Autoencoders*

Variational autoencoder (VAE) is an elegant framework for per-

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

forming variational inference [80], which also has deep connection variational mean field approaches in statistical physics. In fact, the predecessor of VAE is called Helmholtz machines [81]. 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 x into the latent space 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.

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

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 approach originated from statistical physics. The variational Bayes methods is a an application of the variational free energy calculation in statistical physics Eq. (34) for inference problem. For each data we introduce

$$\mathcal{L}(x) = \langle -\ln p(x, z) + \ln q(z|x) \rangle_{z \sim q(z|x)}, \tag{61}$$

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

$$\mathcal{L}_{\theta, \phi}(x) = -\langle \ln p_\theta(x|z) \rangle_{z \sim q_\phi(z|x)} + \mathbb{KL}(q_\phi(z|x)||p(z)). \tag{62}$$

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 [5].

The first term of Eq. (62) 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

$$\ln p_\theta(x|z) = \sum_i x_i \ln \hat{x}_i + (1 - x_i) \ln(1 - \hat{x}_i), \tag{63}$$

$$\hat{x} = \texttt{DecoderNeuralNet}_\theta(z), \tag{64}$$

for binary data. And

$$\ln p_{\theta}(x|z) \;=\; \ln \mathcal{N}(x; \mu, \sigma^2 \mathbf{1}), \tag{65}$$
$$(\mu, \sigma) \;=\; \texttt{DecoderNeuralNet}_{\theta}(z), \tag{66}$$

for continuous data. Gradient of Eq. (62) with respect to θ only depends on the first term.

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

$$\ln q_{\phi}(z|x) = \ln \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{1}) - \ln \left| \det \left(\frac{\partial g_{\phi}(x, \epsilon)}{\partial \epsilon} \right) \right|. \tag{67}$$

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

$$\nabla_{\phi} \langle f(x, z) \rangle_{z \sim q_{\phi}(z|x)} = \langle \nabla_{\phi} f(x, g_{\phi}(x, \epsilon)) \rangle_{\epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{1})}. \tag{68}$$

___ **Info** _____

As an alternate, the REINFORCE [82] (score function) estimator of the gradient reads

$$\nabla_{\phi} \langle f(x, z) \rangle_{z \sim q_{\phi}(z|x)} = \langle f(x, z) \nabla_{\phi} \ln q_{\phi}(z|x) \rangle_{z \sim q_{\phi}(z|x)}. \tag{69}$$

Compared to the reparametrization Eq. (68) REINFORCE usually has larger variance because it only uses the scalar function $\ln q_{\phi}(z|x)$ instead of the vector information of the gradient $\nabla_{\phi} f(x, z)$. An advantage of REINFORCE is that it can also work with discrete latent variables. See Ref. [83] 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 x, we have

$$\ln q_{\phi}(z|x) \;=\; \ln \mathcal{N}(z; \mu, \sigma^2 \mathbf{1}), \tag{70}$$
$$(\mu, \sigma) \;=\; \texttt{EncoderNeuralNet}_{\phi}(x). \tag{71}$$

And the way to sample the latent variable is

$$\epsilon \;\sim\; \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{1}), \tag{72}$$
$$z \;=\; \mu + \sigma \odot \epsilon. \tag{73}$$

The KL term in Eq. (62) can be evaluated analytically [80] 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
>
> $$-\ln p(x) = -\ln \left\langle \frac{p(x, z)}{q(z|x)} \right\rangle_{z \sim q(z|x)}. \tag{74}$$
>
> By using the Jensen's inequality (31) one can also see that the variational free energy Eq. (61) is an upper bound of Eq. (74).

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

$$p(x) = \frac{|\Psi(x)|^2}{\mathcal{Z}}, \tag{75}$$

where $\mathcal{Z}$ is the normalization factor. This representation, named as Born Machine [84], transforms many approaches of representing quantum state into machine learning. Consider binary data, we can represent wavefunction using the matrix product state (MPS) [85]

$$\Psi(x) = \text{Tr}\left(\prod_i A^i[x_i]\right). \tag{76}$$

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 [86, 87].

Adaptive learning

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 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 (32) can be computed efficiently

Efficient gradient

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = -2 \left\langle \frac{\partial \ln \Psi(\boldsymbol{x})}{\partial \boldsymbol{\theta}} \right\rangle_{\boldsymbol{x} \sim \mathcal{D}} + 2 \left\langle \frac{\partial \ln \Psi(\boldsymbol{x})}{\partial \boldsymbol{\theta}} \right\rangle_{\boldsymbol{x} \sim p(\boldsymbol{x})}. \qquad (77)$$

Note that the negative phase (second term) can also be written as $\mathcal{Z}'/\mathcal{Z}$, where $\mathcal{Z}' = 2\sum_{\boldsymbol{x}} \Psi'(\boldsymbol{x})\Psi(\boldsymbol{x})$ and the prime means derivatives with respect to the network parameter $\boldsymbol{\theta}$. Crucially, for MPS both $\mathcal{Z}'$ and $\mathcal{Z}$ 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 (36). 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.

Direct sampling

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. (75) 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.

In a more general context, there are at least two reasons 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 appear naturally when one tries to extend the probabilistic graphical models while still attempt to ensure the positivity of the probability density [88, 89]. Second, tensor networks are doorways to quantum machine learning because many of the tensor networks are formally equivalent to quantum circuits. Tensor network states provide means of architecture design and parameter initialization for quantum circuits [90]. By

now, tensor networks have certainly caught attentions of some of the machine learning practitioners [91, 92]. However, 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.

A final note, besides various tensor network *architectures*, there are large number of *algorithms* for optimizing, evolving, factorizing, and compressing tensor networks. Beside simulating quantum many-body physics, they have far reaching impacts in applications ranging from classical graphical models all the way to quantum computing. Please refer to this Website for more information.

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 [93].

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 z to physical data 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

$$\mathcal{L} = -\left\langle \ln D(x) \right\rangle_{x \sim \mathcal{D}} - \left\langle \ln\left(1 - D(G(z))\right)\right\rangle_{z \sim p(z)}. \tag{78}$$

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.

3.2.7 *Generative Moment Matching Networks*

The generative moment matching networks (GMMN) [94, 95] is another class of implicit generative model which employs the kernel two-sample test as the cost function [96].

The idea is to simply compare the distance in the kernel feature space on the samples drawn from the target and the model distribu-

tions. We refer the following loss function as the squared maximum mean discrepancy (MMD) [97, 98]

$$
\begin{aligned}
\mathcal{L} &= \left\| \sum_x p(x)\phi(x) - \sum_x \pi(x)\phi(x) \right\|^2 \\
&= \langle K(x,y) \rangle_{x\sim p, y\sim p} - 2\langle K(x,y) \rangle_{x\sim p, y\sim \pi} + \langle K(x,y) \rangle_{x\sim \pi, y\sim \pi}.
\end{aligned}
\tag{79}
$$

The summation in the first line runs over the whole Hilbert space. The expectation values in the second line are for the corresponding probability distributions. The function ϕ maps x to a high-dimensional reproducing kernel Hilbert space [99]. However, as common in the kernel methods, by defining a kernel function $K(x,y) = \phi(x)^T\phi(y)$ one can avoid working in the high-dimensional feature space. We employ a mixture of Gaussians kernel $K(x,y) = \frac{1}{c} \sum_{i=1}^{c} \exp\left(-\frac{1}{2\sigma_i}|x-y|^2\right)$ to reveal differences between the two distributions under various scales. Here, σ_i is the bandwidth parameter which controls the width of the Gaussian kernel. The MMD loss with Gaussian kernels asymptotically approaches zero if and only if the output distribution matches the target distribution exactly [97, 98].

Table 3: 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	★★★★	☠☠☠ / ☠

3.3 SUMMARY

In the discussions of generative models we have touched upon a field called probabilistic graphical models [100]. They represent dependence between random variables 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 model unless it has a tree structure. (Or, in special cases such as planar Ising model in the absence of magnetic field.) 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 starting 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 probability distribution of input data to certain target probability distribution. Please note 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 3 summarized and compared the main features of various generative models discussed in this note.

No surprisingly, various generative models are related in various ways. Revealing their connections and seeking for a unified framework calls for a deeper understanding of generative modeling. First of all, the Boltzmann Machines, and in general all 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 [55]. It is still rewarding to explore more connections of representation and learning algorithms between these two classes of models. Second, the autoregressive models for continuous variables are closely related to normalizing flows. While Ref. [57] also discussed connection of normalizing flows to the variational autoencoders. Finally, combining models to take advantage of both worlds is also a rewarding research direction [59, 61, 101, 102].

APPLICATIONS TO QUANTUM MANY-BODY PHYSICS AND MORE

We now discuss a few applications of deep learning to quantum many-body physics. It is impossible 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 future prospects and challenges saw by authors (again, biased views).

The interested readers can check the ⟨Physics|Machine Learning⟩ blog for recent news, events and papers.

4.1 RENORMALIZATION GROUP

Renormalization Group (RG) is a fundamental concept in theoretical physics. In essence, RG keeps relevant information while reducing the dimensionality of data. This strongly resembles the quest for representation learning in generative modeling. The connection of RG and deep learning is both intriguing and rewarding. On one hand side it may provide theoretical understanding to deep learning. And on the other hand, it brings deep learning machineries into solving physical problems with RG.

References [13] proposed a generative Bayesian network with a MERA inspired structure. Reference [14] connects the Boltzmann Machines with decimation transformation in real-space RG. Reference [125] connects principal component analysis with momentum shell RG. Reference [126] proposed to use mutual information as a criteria for restoring the RG behavior in the training of Boltzmann Machines. Lastly, Reference [127] 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 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 NeuralRG 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 implements an information preserving RG procedure, which is useful for exploring holographic duality [128]. And Ref. [129] carried such investigation for the critical XY model.

4.2 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. [103] 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 [104]. Like many deep learning applications to 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. See [105] for a recent review.

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.

4.3 "PHASE" RECOGNITION

4.3.1

Ever since the seminal work by Carrasquilla and Melko [109], there are by now a large number of papers on classifying phases using neural networks. I also tried unsupervised learning approaches for discovering phase transitions [110]. 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 [2, 3] may bring inspi-

rations from the founding fathers of the field. Some recent progress includes [111].

Moreover, the field is fast moving towards handling directly experimental (STM, NMR etc) instead of simulation data.

4.4 COMPUTATION TECHNIQUES

4.4.1 *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 [106] 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 [107, 108]. Bypassing Kohn-Sham orbitals (which are auxiliary objects anyway) can greatly accelerate the search of stable material structures

4.4.2 *Variational Ansatz*

Reference [112] 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 a generalization of Jastrow trial wavefunctions. The RBM is more flexible in the sense that it encodes multi-body correlations in an efficient way [113]. Moreover, employing complex parametrization extends the ability of RBM, so it may also capture quantum states of frustrated systems and unitary dynamics of quantum circuits [112, 114].

References [55, 115, 116] connect the RBM variational ansatz to tensor network states. References [117, 118] 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. An interesting application is on the chiral topological states, in which the standard PEPS ansatz suffers from fundamental difficulties [116, 119].

Another interesting direction is based on the fact that the RBM, in particular, the one used in [112] is equivalent to a 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 [120]. Reference [121] performed variational calculation of excited states using backprop optimization of deep neural networks. Reference [122] 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 the author that further development calls for innovations in the optimization scheme, which goes beyond the wavefunction ansatz, e.g. direct generative sampling, and low variance gradient estimator. As a one step towards this goal, the authors employ variational autoregressive networks to study Ising and spin glasses problems [123]. This calculation follows the variational free energy framework of Eq. (34), which generalizes the celebrated Weiss and Bethe mean field theories. This was then extended to the case of quantum states [124].

4.4.3 *Monte Carlo Update Strategy*

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 are at least three ideas to accelerate Monte Carlo sampling using machine learning techniques.

First, Reference [130] trained surrogate functions to speed up hybrid Monte Carlo simulation [131] 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 [132] or molecular gases [133], or general probabilistic graphical models such as the restricted Boltzmann machine [134]. For Monte Carlo simulations involving fermion determinants [133, 135] 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 [133]. Using the adaptive approach of [136] 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 [137–141]. These papers directly parameterize the proposal probability as neural networks and optimize objective functions related to the efficiency, e.g., the difference of proposals such as the squared jump distances. To ensure unbiased physical results, it is crucial to keep track of the proposal probability of an update and its reverse move for the detailed balance condition. Both A-NICE-MC [137] and L2HMC [138] adopted normalizing flows (Sec. 3.2.3). The later paper is particularly interesting because it systematically generalizes the celebrated hybrid Monte Carlo [131] to a learnable framework. Reference [140] 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. [142], which connected the Swendsen-Wang cluster and the Boltzmann Machines and explored a few new cluster updates.

Lastly, the NeuralRG technique [127] 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.

4.4.4 *Tensor Networks*

Reference [143] applied MPS to pattern recognition. The approach can be interpreted in the framework of kernel learning. The paper and blog post mentioned some parallel attempts in the computer science community. In a second paper [87], Miles Stoudenmire boosted the performance by first performing unsupervised feature extraction using a tree tensor network. Reference [85] 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.

In the reverse direction, applying differentiable programing techniques to tensor network state optimization, the authors has achieved

the state of the art variational energy for infinite size Heisenberg model [144].

4.5 QUANTUM COMPUTATION AND INFORMATION

4.5.1 *Quantum Error Correction*

Quantum error correction is a key step to reach large scale quantum computing. Often this relies on a classical decoder which infer the quantum error from the observed syndrome. For a large class of codes, this inference problem is equivalent to estimating free energy of a classical stat-mech system [145].

Ref. [146] proposed a neural decoder by modeling the syndrome error distribution using an RBM. And there are other attempts using supervised classification [147], reinforcement learning [148–150]. Moreover, Ref. [151] generalized classical belief propagation decoder to a neural network form.

4.5.2 *Quantum Machine Learning*

Quantum machine learning is a vague which covers many applications. Conventionally it refers to apply the HHL algorithm as subroutine of machine learning tasks, see [152, 153] for these aspects. Lately, quantum circuits were also used for classification or evaluating kernels.

Building on the probabilistic interpretation of quantum mechanics, one can envision a quantum generative model [154]. 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. There were already several experiments [155] on generative machine learning. A quantum circuit implementation of the "Born Machine" would be an implicit generative model since one usually does not have direct access to the quantum state of an actual quantum state. The moment matching (Sec. 3.2.7) or adversarial training (Sec. 3.2.6) against a classical neural network can be a way to learn the quantum circuit Born machine [156, 157]. References [158] explores the practical aspects of scaling up the generative learning on actual quantum devices and gradient based training in this setting.

4.5.3 *Quantum State Tomography*

Due to its close similarity to density estimation on classical dataset, neural network was used for reconstruct quantum state from projective measurement [159, 160]. The scheme was recently demonstrated

in actual Rydberg atom experiments [161]. Some recent progress also employed modern generative models [160].

4.5.4 *Quantum Control*

Reinforcement learning was used for quantum control [162, 163] and designing new quantum optics experiment [164]. In cases where the model is known, or when there is an accurate simulator, it is anticipated the differentiable programming approach outperforms reinforcement learning at control tasks.

4.6 MISCELLANEOUS

There is an attempt to train regression model as the impurity solver of dynamical mean-field theory [165] and for analytical continuation of imaginary data [166]. 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) [167]. 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 [168, 169].

ACKNOWLEDGEMENTS

BIBLIOGRAPHY

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

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

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

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

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

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

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

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

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

[10] 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.

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

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

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

[14] 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.

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

[16] 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.

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

[18] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural Ordinary Differential Equations. *ArXiv e-prints*, June 2018. URL https://arxiv.org/abs/1806.07366.

[19] 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.

[20] 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.

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

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

[23] 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.

[24] Michael Bartholomew-Biggs, Steven Brown, Bruce Christianson, and Laurence Dixon. Automatic differentiation of algorithms. *J. Comput. Appl. Math.*, 124(1-2):171–190, 2000. ISSN 03770427. doi: 10.1016/S0377-0427(00)00422-2.

[25] Walter Baur and Volker Strassen. The Complexity of Partial Derivatives. *Theor. Comput. Sci.*, 22:317, 1983. doi: https://doi.org/10.1016/0304-3975(83)90110-X.

[26] Andreas Griewank. On Automatic Differentiation. In *Math. Program. Recent Dev. Appl.*, pages 83–108. Kluwer Academic Publishers, 1989. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.2317.

[27] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533, 1986. URL https://www.nature.com/articles/323533a0.

[28] Atilim Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *J. Mach. Learn.*, 18(14):1–43, feb 2015. ISSN 03091708. doi: 10.1016/j.advwatres.2018.01.009. URL http://jmlr.org/papers/v18/16-107.htmlhttp://arxiv.org/abs/1502.05767.

[29] Nelson Leung, Mohamed Abdelhafez, Jens Koch, and David Schuster. Speedup for quantum optimal control from automatic differentiation based on graphics processing units. *Phys. Rev. A*, 95:042318, 2017. ISSN 24699934. doi: 10.1103/PhysRevA.95.042318.

[30] Teresa Tamayo-Mendoza, Christoph Kreisbeck, Roland Lindh, and Alán Aspuru-Guzik. Automatic Differentiation in Quantum Chemistry with Applications to Fully Variational Hartree-Fock. *ACS Cent. Sci.*, 4(5):559–566, 2018. ISSN 23747951. doi: 10.1021/acscentsci.7b00586.

[31] Andreas Griewank and Andrea Walther. *Evaluating Derivatives*. Society for Industrial and Applied Mathematics, second edi edition, 2008. ISBN 978-0-89871-659-7. doi: 10.1137/1.9780898717761. URL http://epubs.siam.org/doi/book/10.1137/1.9780898717761.

[32] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv*, 2016. ISSN 0270-6474. URL http://arxiv.org/abs/1603.04467.

[33] Adam Paszke, Gregory Chanan, Zeming Lin, Sam Gross, Edward Yang, Luca Antiga, and Zachary Devito. Automatic differentiation in PyTorch. *31st Conf. Neural Inf. Process. Syst.*, (Nips), 2017.

[34] Dougal Maclaurin. *Modeling, Inference and Optimization with Composable Differentiable Procedures*. PhD thesis, Harvard University, 2016. URL https://dash.harvard.edu/handle/1/33493599.

[35] Mike Innes. Flux: Elegant machine learning with julia. *Journal of Open Source Software*, 2018. doi: 10.21105/joss.00602.

[36] Mike Giles. An extended collection of matrix derivative results for forward and reverse mode algorithmic differentiation. Technical report, 2008. URL https://people.maths.ox.ac.uk/gilesm/files/NA-08-01.pdf.

[37] James Townsend. Differentiating the Singular Value Decomposition. Technical report, 2016. URL https://j-towns.github.io/papers/svd-derivative.pdf.

[38] Matthias Seeger, Asmus Hetzel, Zhenwen Dai, Eric Meissner, and Neil D. Lawrence. Auto-Differentiating Linear Algebra. *arXiv*, 2017. URL http://arxiv.org/abs/1710.08717.

[39] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv*, 2016.

[40] Bruce Christianson. Reverse accumulation and attractive fixed points. *Optim. Methods Softw.*, 3(4):311–326, 1994. ISSN 10294937. doi: 10.1080/10556789408805572.

[41] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. 2018. URL http://arxiv.org/abs/1806.07366.

[42] See https://github.com/wangleiphy/tensorgrad for code implementation in PyTorch.

[43] Richard P. Feynman. Forces on Molecules. *Phys. Rev.*, 56(4): 340–343, 1939. ISSN 0031899X. doi: 10.1103/PhysRev.56.340.

[44] Hiroyuki Fujita, Yuya O Nakagawa, Sho Sugiura, and Masaki Oshikawa. Construction of Hamiltonians by supervised learning of energy and entanglement spectra. *Phys. Rev. B*, 97:075114, 2018. ISSN 24699969. doi: 10.1103/PhysRevB.97.075114.

[45] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Ann. Phys. (N. Y).*, 326(1): 96–192, 2011. ISSN 00034916. doi: 10.1016/j.aop.2010.09.012.

[46] Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Ann. Phys. (N. Y).*, 349:117–158, 2014. ISSN 1096035X. doi: 10.1016/j.aop.2014.06.013. URL http://dx.doi.org/10.1016/j.aop.2014.06.013.

[47] Jutho Haegeman and Frank Verstraete. Diagonalizing transfer matrices and matrix product operators: a medley of exact and computational methods. *Annu. Rev. Condens. Matter Phys.*, 8:355, 2017. ISSN 1947-5454. doi: 10.1146/annurev-conmatphys-031016-025507. URL https://www.annualreviews.org/doi/10.1146/annurev-conmatphys-031016-025507.

[48] Zheng-Cheng Gu and Xiao-Gang Wen. Tensor-entanglement-filtering renormalization approach and symmetry-protected topological order. *Phys. Rev. B*, 80(15):155131, 2009. ISSN 10980121. doi: 10.1103/PhysRevB.80.155131.

[49] Frank Pollmann, Ari M. Turner, Erez Berg, and Masaki Oshikawa. Entanglement spectrum of a topological phase in one dimension. *Phys. Rev. B*, 81(6):064439, 2010. ISSN 10980121. doi: 10.1103/PhysRevB.81.064439.

[50] Z. Y. Xie, J. Chen, J. F. Yu, X. Kong, B. Normand, and T. Xiang. Tensor Renormalization of Quantum Many-Body Systems Using Projected Entangled Simplex States. *Phys. Rev. X*, 4:011025, 2014. doi: 10.1103/physrevx.4.011025.

[51] Efi Efrati, Zhe Wang, Amy Kolan, and Leo P. Kadanoff. Real-space renormalization in statistical mechanics. *Rev. Mod. Phys.*, 86(2):647–667, 2014. ISSN 15390756. doi: 10.1103/RevModPhys.86.647.

[52] 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/.

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

[54] 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.

[55] 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.

[56] 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.

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

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

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

[60] 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.

[61] 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.

[62] 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.

[63] 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.

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

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

[66] 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.

[67] 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/.

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

[69] 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.

[70] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. *ArXiv e-prints*, October 2018. URL https://arxiv.org/abs/1810.01367.

[71] L. Zhang, W. E, and L. Wang. Monge-Amp\'ere Flow for Generative Modeling. *ArXiv e-prints*, September 2018. URL https://arxiv.org/abs/1809.10188.

[72] Yann Brenier. Polar factorization and monotone rearrangement of vector-valued functions. *Communications on pure and applied mathematics*, 44(4):375–417, 1991.

[73] Luis A Caffarelli, NSF-CBMS Conference on the Monge Ampère Equation: Applications to Geometry, Optimization, and Mario Milman. *Monge Ampere equation: applications to geometry and optimization*. American Mathematical Society, 1998.

[74] Cédric Villani. *Topics in optimal transportation*. Number 58. American Mathematical Soc., 2003.

[75] Kip S Thorne and Roger D Blandford. *Modern Classical Physics: Optics, Fluids, Plasmas, Elasticity, Relativity, and Statistical Physics*. Princeton University Press, 2017.

[76] Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media, 2008.

[77] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/arjovsky17a.html.

[78] Olivier Bousquet, Sylvain Gelly, Ilya Tolstikhin, Carl-Johann Simon-Gabriel, and Bernhard Schoelkopf. From optimal transport to generative modeling: the VEGAN cookbook. 2017. URL http://arxiv.org/abs/1705.07642.

[79] Aude Genevay, Gabriel Peyré, and Marco Cuturi. GAN and VAE from an Optimal Transport Point of View. 2017. URL http://arxiv.org/abs/1706.01807.

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

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

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

[83] 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.

[84] 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.

[85] 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.

[86] 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.

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

[88] 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.

[89] 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.

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

[91] 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.

[92] 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.

[93] Shakir Mohamed and Balaji Lakshminarayanan. Learning in Implicit Generative Models. URL http://arxiv.org/abs/1610.03483.

[94] Yujia Li, Kevin Swersky, and Richard Zemel. Generative moment matching networks. . URL http://arxiv.org/abs/1502.02761.

[95] Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. URL https://arxiv.org/abs/1505.03906.

[96] N.H. Anderson, P. Hall, and D.M. Titterington. Two-sample test statistics for measuring discrepancies between two multivariate probability density functions using kernel-based density estimates. *Journal of Multivariate Analysis*, 50(1):41 − 54, 1994. ISSN 0047-259X. URL http://www.sciencedirect.com/science/article/pii/S0047259X84710335.

[97] Arthur Gretton, Karsten M Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola. A ker-

nel method for the two-sample-problem. In *Advances in Neural Information Processing Systems*, pages 513–520, 2007. URL http://papers.nips.cc/paper/3110-a-kernel-method-for-the-two-sample-problem.pdf.

[98] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012. URL http://www.jmlr.org/papers/v13/gretton12a.html.

[99] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008. URL http://www.jstor.org/stable/25464664.

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

[101] 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.

[102] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. MMD GAN: Towards Deeper Understanding of Moment Matching Network. (Theorem 1), . URL http://arxiv.org/abs/1705.08584.

[103] 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. URL http://arxiv.org/abs/1805.09003.

[104] 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.

[105] Benjamin Sanchez-Lengeling and Alán Aspuru-Guzik. Inverse molecular design using machine learning: Generative models for matter engineering. *Science*, 361(6400):360–365, 2018.

[106] 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.

[107] 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.

[108] 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.

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

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

[111] Grigory Bednik. Probing topological properties of 3D lattice dimer model with neural networks. (1):1–7.

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

[113] G. Carleo, Y. Nomura, and M. Imada. Constructing exact representations of quantum many-body systems with deep neural networks. *ArXiv e-prints*, February 2018. URL https://arxiv.org/abs/1802.09558.

[114] B. Jónsson, B. Bauer, and G. Carleo. Neural-network states for the classical simulation of quantum computing. *ArXiv e-prints*, August 2018. URL https://arxiv.org/abs/1808.05232.

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

[116] 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.

[117] 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.

[118] 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.

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

[120] 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.

[121] Kenny Choo, Giuseppe Carleo, Nicolas Regnault, and Titus Neupert. Symmetries and many-body excitations with neural-network quantum states. *Phys. Rev. Lett.*, 121:167204, Oct 2018. doi: 10.1103/PhysRevLett.121.167204. URL https://link.aps.org/doi/10.1103/PhysRevLett.121.167204.

[122] J. Han, L. Zhang, H. Wang, and W. E. Solving Many-Electron Schro2dinger Equation Using Deep Neural Networks. *ArXiv e-prints*. URL http://arxiv.org/abs/1807.07014.

[123] D. Wu, L. Wang, and P. Zhang. Solving Statistical Mechanics using Variational Autoregressive Networks. *ArXiv e-prints*, September 2018. URL https://arxiv.org/abs/1809.10606.

[124] Or Sharir, Yoav Levine, Noam Wies, Giuseppe Carleo, and Amnon Shashua. Deep autoregressive models for the efficient variational simulation of many-body quantum systems. 2019. URL http://arxiv.org/abs/1902.04057.

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

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

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

[128] 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.

[129] Hong-Ye Hu, Shuo-Hui Li, Lei Wang, and Yi-Zhuang You. Machine Learning Holographic Mapping by Neural Network Renormalization Group. 1:1–11, 2019. URL http://arxiv.org/abs/1903.00804.

[130] 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.

[131] 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.

[132] 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.

[133] 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.

[134] 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.

[135] 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.

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

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

[138] 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.

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

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

[141] Kai-Wen Zhao, Wen-Han Kao, Kai-Hsin Wu, and Ying-Jer Kao. Generation of ice states through deep reinforcement learning. pages 1–9, 2019. URL http://arxiv.org/abs/1903.04698.

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

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

[144] Hai-jun Liao, Jin-guo Liu, Lei Wang, and Tao Xiang. Differentiable Programming Tensor Networks. pages 1–10.

[145] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *J. Math. Phys.*, 43(9):4452–4505, 2002. ISSN 00222488. doi: 10.1063/1.1499754.

[146] Giacomo Torlai and Roger G. Melko. Neural Decoder for Topological Codes. *Phys. Rev. Lett.*, 119(3):1–5, 2017. ISSN 10797114. doi: 10.1103/PhysRevLett.119.030501.

[147] Chaitanya Chinni, Abhishek Kulkarni, and Dheeraj M. Pai. Neural Decoder for Topological Codes using Pseudo-Inverse of Parity Check Matrix. pages 1–12, 2019. URL http://arxiv.org/abs/1901.07535.

[148] Ryan Sweke, Markus S Kesselring, Evert P L Van Nieuwenburg, and Jens Eisert. Reinforcement Learning Decoders for Fault-Tolerant Quantum Computation. pages 1–15, 2018.

[149] Hendrik Poulsen Nautrup, Nicolas Delfosse, Vedran Dunjko, Hans J Briegel, and Nicolai Friis. Optimizing Quantum Error Correction Codes with Reinforcement Learning. pages 1–16, 2018.

[150] Philip Andreasson, Joel Johansson, Simon Liljestrand, and Mats Granath. Quantum error correction for the toric code using deep reinforcement learning. pages 28–33.

[151] Ye-hua Liu and David Poulin. Neural Belief-Propagation Decoders for Quantum Error-Correcting Codes. pages 1–6.

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

[153] 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.

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

[155] 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.

[156] H. Situ, Z. He, L. Li, and S. Zheng. Adversarial training of quantum Born machine. *ArXiv e-prints*, July 2018. URL http://arxiv.org/abs/1807.01235.

[157] J. Zeng, Y. Wu, J.-G. Liu, L. Wang, and J. Hu. Learning and Inference on Generative Adversarial Quantum Circuits. *ArXiv e-prints*, August 2018. URL https://arxiv.org/abs/1808.03425.

[158] J.-G. Liu and L. Wang. Differentiable Learning of Quantum Circuit Born Machine. *ArXiv e-prints*, April 2018. URL https://arxiv.org/abs/1804.04168.

[159] 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.

[160] J. Carrasquilla, G. Torlai, R. G. Melko, and L. Aolita. Reconstructing quantum states with generative models. *ArXiv e-prints*, October 2018. URL http://arxiv.org/abs/1810.10584.

[161] G. Torlai, B. Timar, EPL Van Nieuwenburg, H. Levine, A. Keesling, A. Omran, H. Bernien, M. Greiner, V. Vuletic, M. D. Lukin, R. Melko, and M. Endres. Hybridizing Neural Networks with a Quantum Simulator for State Reconstruction. *To Be Publ.*, pages 1–15, 2019.

[162] 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.

[163] Murphy Yuezhen Niu, Sergio Boixo, Vadim Smelyanskiy, and Hartmut Neven. Universal Quantum Control through Deep Reinforcement Learning. pages 1–21, 2018.

[164] 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.

[165] 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.

[166] 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.

[167] 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.

[168] 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.

[169] 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.