Quantum machine learning

I. Machine learning implemented on quantum devices, e.g.

• Quantum neural networks
• Classical ML trained by optimization using quantum devices (such as D-wave’s annealer)

II. Machine learning for solving quantum problems

• Predicting the properties of complex quantum systems by ML
Goal of Quantum Machine Learning

QML seeks to exploit quantum devices in order to design:

- Algorithms that could be trained faster
- Algorithms that yield more accurate results
- Algorithms that process big data faster

Here, `faster’ and `more accurate’ is by comparison with classical ML algorithms
Example 1: Quantum annealing
Quantum annealing

Say, we have a Hamiltonian

\[ \hat{H} = \sum_i b_i \sigma_i^z + \sum_{i,j} w_{i,j} \sigma_i^z \sigma_j^z \]

with given \( b_i \) and \( w_{i,j} \)

Question: what is the ground state of the system described by this Hamiltonian?
D-wave’s quantum annealer

\[ \hat{H} = A(t) \sum_i \sigma_i^x - B(t) \left[ \sum_i b_i \sigma_i^z + \sum_{i,j} w_{i,j} \sigma_i^z \sigma_j^z \right] \]

\[ A(t = 0) \gg B(t = 0) \approx 0 \quad A(t) \approx 0 \ll B(t) \]

\( b_i \) and \( w_{i,j} \) are tunable parameters

M. H. Amin, 2015, ‘Searching for quantum speed-up in quasistatic quantum annealers’
Example 2: Quantum Boltzmann machines
A type of a neural network

Boltzmann machine

FIG. 1: A graphical representation of two types of Boltzmann machines. (a) A 4–layer deep restricted Boltzmann machine (dRBM) where each black circle represents a hidden or visible unit and each edge represents a non–zero weight for the corresponding interaction. The output layer is often treated as a visible layer to provide a classification of the data input in the visible units at the bottom of the graph. (b) An example of a 5–unit full Boltzmann machine (BM). The hidden and visible units no longer occupy distinct layers due to connections between units of the same type.

\[ \mathcal{O}_{\text{ML}} \]

where \( N_{\text{train}} \) is the size of the training set, \( x_{\text{train}} \) is the set of training vectors, and \( \lambda \) is an \( L_2 \)-regularization term to combat overfitting. The derivative of \( \mathcal{O}_{\text{ML}} \) with respect to the weights is

\[
\frac{\partial \mathcal{O}_{\text{ML}}}{\partial w_{i,j}} = h_v^i h_j^i \]

where the brackets denote the expectation values over the data and model for the BM. The remaining derivatives take a similar form \[ 5 \].

Computing these gradients directly from (1) and (4) is exponentially hard in \( n_v \) and \( n_h \); thus, classical approaches resort to approximations such as contrastive divergence \[ 3, 5 \text{–} 8 \]. Unfortunately, contrastive divergence does not provide the gradient of any true objective function \[ 9 \], it is known to lead to suboptimal solutions \[ 10 \text{–} 12 \], it is not guaranteed to converge in the presence of certain regularization functions \[ 9 \], and it cannot be used directly to train a full Boltzmann machine. We show that quantum computation provides a much better framework for deep learning and illustrate this by providing efficient alternatives to these methods that are elementary to analyze, accelerate the learning process and lead to better models for the training data.

**GEQS Algorithm**

We propose two quantum algorithms: Gradient Estimation via Quantum Sampling (GEQS) and Gradient Estimation via Quantum Amplitude Estimation (GEQAE). These algorithms prepare a coherent analog of the Gibbs state for Boltzmann machines and then draw samples from the resultant state to compute the expectation values in (4).

Formal descriptions of the algorithms are given in the appendix. Existing algorithms for preparing these states \[ 13 \text{–} 17 \] tend not to be efficient for machine learning applications or do not offer clear evidence of a quantum speedup. The inefficiency of \[ 15, 17 \] is a consequence of the uniform initial state having small overlap with the Gibbs state. The complexities of these prior algorithms, along with our own, are given in Table I.

Our algorithms address this problem by using a non-uniform prior distribution for the probabilities of each configuration, which is motivated by the fact that we know a priori from the weights and the biases that certain configurations will be less likely than others. We obtain this distribution by using a mean–field (MF) approximation to the configuration probabilities. This approximation is classically efficient and typically provides a good approximation to the Gibbs states observed in practical machine learning problems \[ 7, 18, 19 \]. Our algorithms exploit this prior knowledge to refine the Gibbs state from copies of the MF state. This allows the Gibbs distribution to be prepared efficiently and exactly if the two states are sufficiently close.

The MF approximation, \( Q(v,h) \), is defined to be the product distribution that minimizes the Kullback–Leibler divergence \( KL(Q||P) \). The fact that it is a product distribution means that it can be efficiently computed and also

N. Wiebe, A. Kapoor, K. M. Svore, 2015, `Quantum Deep Learning'
Boltzmann machine

\[ \mathbf{v} \Rightarrow [v_1, v_2, ..., v_V] \quad \text{visible units} \]
\[ \mathbf{h} \Rightarrow [h_1, h_2, ..., h_H] \quad \text{hidden units} \]
\[ v_i = \pm 1 \quad h_i = \pm 1 \]
\[ z = (\mathbf{v}, \mathbf{h}) \]

Energy:
\[ E(\mathbf{v}, \mathbf{h}) = - \sum_i b_i z_i - \sum_{i,j} w_{i,j} z_i z_j \]

Probability to observe state \( \mathbf{v} \) of visible units:
\[ P_\mathbf{v} = \frac{-\sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}}{Z} \]

Partition function:
\[ Z = \sum_{\mathbf{z}} e^{-E(\mathbf{v}, \mathbf{h})} \]
How to train a Boltzmann machine

Let’s say you have a certain number of state vectors, (each with dimension equal to the number of visible units)

The goal is find a Boltzmann distribution, in which the training vectors have high probability

Thus, we want to maximize the following function:

\[
\mathcal{L} \propto \sum_{v \in \text{training set}} \log P_v \\
\]

\[
P_v = \frac{\sum_h e^{-E(v,h)}}{Z} 
\]
Let’s say you have a certain number of state vectors, (each with dimension equal to the number of visible units)

The goal is to find a Boltzmann distribution, in which the training vectors have high probability

Thus, we want to maximize the following function:

\[ \mathcal{L} \propto \sum_{\mathbf{v} \in \text{training set}} \log P_{\mathbf{v}} \]

\[ \frac{\partial \mathcal{L}}{\partial w_{i,j}} \propto \langle v_i h_j \rangle_{\text{training}} - \langle v_i h_j \rangle_{\text{model}} \]
How to train a Boltzmann machine

Another way to formulate the problem.

Let’s say you have a certain distribution of state vectors

Let’s denote it by $P_v^+$

The goal is to find a Boltzmann machine, where $P_v = P_v^+$

Thus, we want to maximize the following function:

$$\mathcal{L} \propto \sum_v P_v^+ \log P_v$$

$$P_v = \frac{\sum_h e^{-E(v,h)}}{Z}$$
Let’s say you have several classes of data. How to discriminate between them?

- Include a `label’ of the class as a visible unit
- Train the BM
- Try the test vector with all the different class labels
- The one with the lowest free energy is the most likely class
The goal is to compute:

\[ \frac{\partial L}{\partial w_{i,j}} \propto \langle v_i h_j \rangle_{\text{training}} - \langle v_i h_j \rangle_{\text{model}} \]

For restricted Boltzmann Machines (with bipartite graph structure),
the first term is easy to compute using:

\[ P(h_j = 1|v) = \text{sigm} \left( b_j + w_{i,j} v_i \right) \]

However, the second term is hard to compute!
Training a general Boltzmann machine is difficult!

From WikiPedia:

the time the machine must be run in order to collect equilibrium statistics grows exponentially with the machine's size, and with the magnitude of the connection strengths.

Quantum computing can help!

N. Wiebe, A. Kapoor, K. M. Svore, 2015, `Quantum Deep Learning'
Quantum sampling for training a Boltzmann machine

Energy: \[ E(\mathbf{v}, \mathbf{h}) = - \sum_i b_i z_i - \sum_{i,j} w_{i,j} z_i z_j \]

Quantum system: \[ \hat{H} = - \sum_i b_i \sigma_i^z - \sum_{i,j} w_{i,j} \sigma_i^z \sigma_j^z \]

D-wave’s annealer:

\[ \hat{H} = A(t) \sum_i \sigma_i^x - B(t) \left[ \sum_i b_i \sigma_i^z + \sum_{i,j} w_{i,j} \sigma_i^z \sigma_j^z \right] \]

\[ A(t = 0) \gg B(t = 0) \approx 0 \quad A(t) \approx 0 \ll B(t) \]

\[ b_i \text{ and } w_{i,j} \text{ are tunable parameters} \]
Quantum sampling for training a Boltzmann machine

The goal is to compute:

\[
\frac{\partial L}{\partial w_{i,j}} \propto \langle v_i h_j \rangle_{\text{training}} - \langle v_i h_j \rangle_{\text{model}}
\]

For restricted Boltzmann Machines (with bipartite graph structure), the first term is easy to compute using:

\[
P(h_j = 1|v) = \text{sigm} \left( b_j + w_{i,j}v_i \right)
\]

However, the second term is hard to compute!

Run quantum annealing $N$ times and compute

\[
\langle v_i h_j \rangle_{\text{model}} = \frac{1}{N} \sum_n v_i^n h_j^n
\]
Combine machine learning with quantum dynamics calculations to solve problems generally considered unfeasible.
Complexity of many-body quantum systems

Disorder

Interactions

Environment

Disorder
Complexity of many-body quantum systems

- Disorder
- Interactions
- Environment
Complexity of many-body quantum systems

Interactions

Environment

Disorder

Non-Markovian environments

Classical chaos

weak localization

Anderson localization

Many-body localization
Amenable to theoretical study

- Many-body localization
- Weak localization
- Anderson localization
- Non-Markovian environments
- Classical chaos
Can Phase I be used to predict the properties of Phases II and III?
Machine Learning for extrapolation of quantum properties

Gaussian Process = Gaussian Random Function
Gaussian Process Regression

Dynamical observables vs. Input variable

-4 -2 0 2
-2 0 2
Gaussian Process Regression

- Constrained random functions

![Graph showing Gaussian Process Regression](image-url)
Step 1. Calculate the observable at a few points of this multi-dimensional space.
Step 2. Estimate correlations between the calculated points

Assume some mathematical form for the correlations, e.g.,

$$R(x, x') = \prod_{i=1}^{q} \left( 1 + \frac{\sqrt{5}|x_i - x'_i|}{\omega_i} + \frac{5(x_i - x'_i)^2}{3\omega_i^2} \right) \exp \left( -\frac{\sqrt{5}|x_i - x'_i|}{\omega_i} \right)$$

and find the maximum likelihood estimates of the unknown coefficients.
We do this by maximizing the log-likelihood function:

$$\log \mathcal{L}(\omega|Y^n) = -\frac{1}{2} \left[ n \log \hat{\sigma}^2 + \log(\det(A)) + n \right]$$

with

$$\hat{\sigma}^2(\omega) = \frac{1}{n} (Y^n - \beta)^\top A^{-1} (Y^n - \beta)$$

$$\hat{\beta}(\omega) = (I^\top A^{-1} I)^{-1} I^\top A^{-1} Y^n$$
Gaussian Process Regression – how does it work?

Step 2. Estimate correlations

\[
R(x, x') = \prod_{i=1}^{q} \left( 1 + \frac{\sqrt{5}|x_i - x_i'|}{\omega_i} + \frac{5(x_i - x_i')^2}{3\omega_i^2} \right) \exp \left( -\frac{\sqrt{5}|x_i - x_i'|}{\omega_i} \right).
\]

- Any simple function works for interpolation
- Extrapolation is harder!
- Necessary to find kernels capable of extrapolating physically!
How to find the best kernel?

\[
k_{\text{LIN}}(x_i, x_j) = x_i^T x_j
\]

\[
k_{\text{RBF}}(x_i, x_j) = \exp \left( -\frac{1}{2} r^2(x_i, x_j) \right)
\]

\[
k_{\text{MAT}}(x_i, x_j) = \left( 1 + \sqrt{5} r^2(x_i, x_j) + \frac{5}{3} r^2(x_i, x_j) \right)
\]
\[
\times \exp \left( -\sqrt{5} r^2(x_i, x_j) \right)
\]

\[
k_{\text{RQ}}(x_i, x_j) = \left( 1 + \frac{|x_i - x_j|^2}{2\alpha \ell^2} \right)^{-\alpha}
\]
Gaussian Process Regression – kernel optimization

$$\text{BIC} = -\frac{1}{2} \left[ n \log \hat{\sigma}^2 + \log(\det(\mathbf{A})) + n \right] - \frac{1}{2} \mathcal{M} \log n$$

Extrapolation across phase transitions

Goal: extrapolate quantum properties across phase transitions

How? Can’t use properties that diverge at the transitions

Idea: find the quantity that varies smoothly across the transition, “learn it”, extrapolate it, and derive the discontinuous properties from it

Extrapolation across phase transitions

Heisenberg model

\[ \hat{H} = -\frac{J}{2} \sum_{i,j} \vec{S}_i \cdot \vec{S}_j \]
Heisenberg model

\[ \hat{H} = -\frac{J}{2} \sum_{i,j} \vec{S}_i \cdot \vec{S}_j \]

Free energy density at temperature T:

\[ f(T, m) \approx \frac{1}{2} \left( 1 - \frac{T_c}{T} \right) m^2 + \frac{1}{12} \left( \frac{T_c}{T} \right)^3 m^4 \]
Extrapolation across phase transitions

Extrapolation across phase transitions

Extrapolation across phase transitions

Extrapolation across phase transitions

Extrapolation across phase transitions

Extrapolation across phase transitions

Extrapolation across phase transitions

Extrapolation across phase transitions

Kernel complexity matters!

It is difficult to build models with kernels that are very complex

Can kernels be optimized using a quantum computing device?