Quantum Machine Learning

Dong-Sheng Wang

April 21, 2021

Definition. *Quantum machine learning is a class of quantum algorithm that uses a quantum training algorithm to train a quantum machine, which is another quantum algorithm.*

Keywords

Data science; automation; machine learning; neuron networks; quantum gradient descent; hybrid classical-quantum algorithms.

Abstract

This is an overview of the emerging topic of quantum machine learning (QML). We first explain machine learning and some primary types of machine networks. We then introduce QML, which has not been fully understood yet, and then compare with the relatively easier task of hybrid classical-quantum ML, which only uses quantum computing as the subroutine machine.

1 Minimal version of QML

1.1 Opening

Machine learning (ML) is a type of classical algorithm, which deals with 'big data', in particular. The key feature of ML is that it has a *training*

stage which is an optimization procedure using sample data to determine the machine, i.e., the parameters in a neuron network. Compared with building a neuron network, the training stage is more nontrivial. Just like people learning a new thing, it is better to learn or grasp the main features, i.e., the essence, then new problems can be solved after learning.

The neuron network has many parameters such as weights and biases, while the learning algorithm also has many so-called 'hyper-parameters', which needs to be chosen externally. That is to say, ML is a cascade algorithm that the learning algorithm takes data as input and the (parameters of the) machine as output, and the machine itself is also an algorithm. It is exactly the idea of cascade, or *automation*, that makes ML powerful. Sometimes algorithms used to find hyper-parameters are called 'meta-algorithms'. It is not hard to see the training of ML is a kind of meta-algorithm. You might imagine a multi-level cascade of algorithms, but that went too far.

ML has wide applications in many fields of science, but mainly for artificial intelligence and big data science. In physics, it has been applied to statistical physics, particle physics, and also starts for quantum physics. It shall be treated as a component of computational physics; however, it also reveals some physical principles (not very clear yet), and the QML is expected to boost quantum computing and also quantum physics.

1.2 Basics

Here we analyze the basics of ML. In computer science, algorithms are used to solve problems. For different problems, the algorithms being used may appear very different. Is there a sort of universal algorithm that can be used or tuned to solve different problems? Well, ML is roughly of this kind. A ML algorithm contains two parts: a training algorithm (teacher) $\mathcal{A}_1$ and a machine (student) $\mathcal{A}_2$, which is also an algorithm. The central idea is that the machine $\mathcal{A}_2$ is represented by a sort of universal structure, e.g., neuron networks, but with tunable parameters, denoted by a vector $\vec{\theta}$. Given a problem, the trainer $\mathcal{A}_1$ is used to find the optimal parameters $\vec{\theta}$ by optimizing a cost function given a large set of data.

Let's see how to make a neuron network. A neuron is just a function $f(x)$, and the variable x is not limited to be binary. As the name hinted, it is designed to mimic the behavior of biological neuron: it is usually parameterized by a weight and a bias. But, of course, the formula of neuron is not unique. Now a neuron network (NN) is just a cascade layered sequence of

neurons, with input (output) neurons arranged in an input (output) layer, and the functional neurons called 'hidden layers'. If you can only see the input and output layers, the rest layers are really hidden.

The power of NN comes from the fact that it is *universal*, meaning that it can be used to simulate any Boolean circuit efficiently. The idea to prove the universality is also simple. There exists the so-called universal gate set for Boolean circuits, these gates include NOT, OR, NAND, etc. The NAND gate itself is universal, and it is easy to find a neuron to simulate it. Therefore, NN is a universal computational model.

Let's call the number of hidden layers as the depth of the network. It is known that depth-one NN is also universal; namely, a higher-depth NN can be efficiently simulated by a depth-one NN. We will not review the proof here, but note that it is quite simple. ML with high-depth NN is the so-called 'deep learning'. Higher depth provides another dimension to exhibit the power of NN, and data structures can be attributed along the depth dimension. For instance, in the convolutional NN (CNN), the later layers can be viewed as a kind of coarse-graining of former layers, mimic the idea of renormalization transformation. Given a big data set, a CNN can learn features of the data on different 'levels', just like a catalog.

We have not explained how the training works. The parameterized NN is not treated as a black box; instead, it is used explicitly in the training process and used many times. Let's see an example: to classify hand-written letters, which is the most canonical examples of ML. Given a set of samples, we use a part of it as training samples, and the rest as test samples after the training. The training samples are altogether denoted as an input vector $\vec{x}$. Now, there are two basic ways to proceed: one is called supervised training, and the other unsupervised. For the supervised training, each sample is also assigned a correct label, altogether as $\vec{y}$. Now, the goal of training is to optimize parameters $\vec{\theta}$ of NN, which is a function $f(\vec{x}, \vec{\theta})$, so that a cost function $g(\vec{x}, \vec{y}, \vec{\theta})$ is minimal. The space of all possible f is called the hypothesis space. The cost function g in general is very complicated and mostly do not have an explicit formula, and it is not guaranteed to be a global minima. On the other hand, the unsupervised training can automatically learn features of letters, and then classify them by optimizing a cost function that will be different from the supervised case.

Given the samples, the training will take a long time. It is in general not known how the optimization cost scales with the size of the sample set. The

primary optimization scheme is the gradient descent (GD) via the so-called *back-propagation* method. It can compute the derivative $\partial f/\partial\theta_i$ for all θ_i in the same layer at the same time. With derivatives, the values of $\vec{\theta}$ can be updated via GD method. The derivatives are computed from the last layer to the former ones, so it is a back-propagation of information. This can be proved based on matrix analysis. As we will see later, the quantum analog of back-propagation is not established yet.

Finally, we mention that there are also other models such as support-vector machine, Boltzmann machine etc which are then easy to understand. The Boltzmann machine is usually employed in unsupervised learning to generate samples according to a certain probability distribution. It is easy to quantize it as a quantum spin system, with most of them as latent (hidden) spins. A quantum advantage is expected (but not proved) since quantum thermalization and tunneling may offer speedup, which is one of the goal of quantum annealing.

1.3 Main: Hybrid classical-quantum ML

Usually when people talk about QML, what they mean probably is the hybrid classical-quantum ML, denoted by HML here. This uses a classical algorithm $\mathcal{A}_1$ to train a quantum algorithm $\mathcal{Q}_2$. The HML framework agrees with usual quantum algorithms, which uses quantum computers to solve classical problems. In HML, the quantum part $\mathcal{Q}_2$ replaces the classical NN, and is expected to show a speedup.

Let's see how this works. Similar with ML, we have a set of classical data at hand. In order to use $\mathcal{Q}_2$, we have to upload classical data to it. So we need to design $\mathcal{Q}_2$ first. What is the quantum analog of the classical NN? Well, this is an unsolved problem yet. However, there are two approaches: one uses quantum circuits, and the other uses tensor network states. A tensor network state can also be prepared by a quantum circuit by replacing tensors with quantum gates. So roughly speaking, a quantum NN is just a parametric quantum circuit. The parameters in it are usually qubit rotation angles.

Now we can discuss the upload step. This is to encode classical data x as a quantum state $|\phi(x)\rangle$. This encoding is called the *feature map*. It turns out the map is not unique. An input $x = (x_i)$ is a vector of independent numbers. As we know, a qubit pure state has two real parameters, $|\varphi\rangle = \cos\alpha|0\rangle + \sin\alpha e^{i\theta}|1\rangle$, so we can encode two x_i in it. This is called the 'qubit

encoding'. Another way is to encode x_i directly as the amplitude of a multi-qubit entangled state, which is common in many quantum algorithms. This is called the 'amplitude encoding'. We see that the qubit encoding is maximally local: a x_i is carried locally by a qubit, while the amplitude encoding is maximally nonlocal: a x_i is carried nonlocally by all qubits. There are some tradeoffs between the two methods. Here we only mention one of them. If the size of x is N, then we can see that N qubits are needed for qubit encoding, while only $\log N$ needed for amplitude encoding, which is a huge saving of memory! However, the encoded multi-qubit entangled state might be expensive to prepare on the first hand!

Following the feature map circuit, $U(x)$, we use a quantum circuit $W(\vec{\theta})$ as the quantum NN, followed by some measurement, the outcome of which will be used in the classical training algorithm $\mathcal{Q}_1$. Given the classical data set, the quantum circuit will be run for many times, with parameters $\vec{\theta}$ adjusted accordingly. How does $W(\vec{\theta})$ look like? It turns out there are some circuit topology to follow. One method is to use sine-cosine decomposition to represent any $U \in SU(d)$ as a sequence of controlled rotations, called Givens rotations, wherein the locations of entangling gates are fixed, while qubit rotation angles can be changed. Another method is to use the quantum circuit for preparing matrix-product states, which has a cascade structure, and the free parameters in it are also qubit rotation angles.

2 Advanced topics: QML based on GD

Now let's see how to 'quantize' the whole process of ML. This is the truly interesting part. We have to use a quantum trainer $\mathcal{Q}_1$ and a quantum student $\mathcal{Q}_2$, and they need to be combined together in some way.

As $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are both quantum operations, namely, unitary circuits, $\mathcal{Q}_2$ cannot be the output from $\mathcal{Q}_1$ directly. Can they be part of a more fundamental circuit, $\mathcal{Q}$? A recent method is of this kind. Recall that $\mathcal{Q}_2$ contains parameters that need to be optimized. The idea is to treat these parameters, denoted each as ϕ here, as the eigenvalues of an operator $\hat{\phi}$. This is a 2nd quantization! Obviously, one has to use harmonic oscillators or even quantum fields to achieve this! Leaving this aside, we can now have a hyper-system with operator $\hat{\phi}$ and its conjugate, $\hat{\pi}$, and design a Hamiltonian that will control the evolution of ϕ, hence the circuit $\mathcal{Q}_2$. If $\mathcal{H}_2$ denotes the Hilbert space for $\mathcal{Q}_2$, then we need a new Hilbert space $\mathcal{H}_\phi$ for the quantized

hyper-parameters. The dimension of $\mathcal{H}_\phi$ is infinite. The coupling between the two spaces is due to the gate

$$U = \int d\phi \, |\phi\rangle\langle\phi| \otimes U(\phi), \tag{1}$$

which appears succinct while may be very hard to realize. It is likely a coupling between a discrete-variable and continuous-variable quantum systems. This will parallelize the action of $U(\phi)$ for all values of ϕ. Now the learning algorithm is defined by a Hamiltonian

$$H = \mu\hat{\pi}^2 + \eta V(\hat{\phi}), \tag{2}$$

and a learning step is a short-time evolution $e^{itH} \approx e^{it\mu\hat{\pi}^2} e^{it\eta V(\hat{\phi})}$. the choice of parameters μ, η may need a meta-algorithm, or just some meta-knowledge. The potential term $e^{it\eta V(\hat{\phi})}$ will shift $\hat{\pi}$ by $\nabla V(\hat{\phi})$, and the kinetic term $e^{it\mu\hat{\pi}^2}$ will shift $\hat{\phi}$ by $\hat{\pi}$. This omits higher order terms in η. The operator $V(\hat{\phi})$ is nothing but the cost function, which is now an operator serving as the potential. So a learning step will change the parameters according to the gradient of the cost function, just like the classical case.

How does the cost-function operator work? It is from the product $U^\dagger e^{i\eta L} U$, which is the conjugate of the coupling gate U on an evolution, which acts on the space $\mathcal{H}_2$. Now the magic occurs: if we choose η, which is called the *learning rate*, very small, then this evolution will not generate entanglement between the spaces $\mathcal{H}_2$ and $\mathcal{H}_\phi$; instead, it is approximated by a pure evolution on the space $\mathcal{H}_\phi$, with an effective Hamiltonian, denoted as $V(\hat{\phi})$. This is the origin of the cost-function operator. It is said that the effect of $e^{i\eta L}$ on $\mathcal{H}_2$ is 'kicked back' to the parameter space $\mathcal{H}_\phi$. This process is a bit similar with the so-called *weak measurement* protocol: if you use a device to measure a quantum system weakly, then there will not be entanglement between the device and the system; instead, the device may get a kick by the system, which will read out the measurement outcome.

The above sketched method works in principle. However, it is not clear how good it is. Actually, it appears very cumbersome since the learning has to be done very slowly, very close to an adiabatic process. The entanglement issue mentioned at the beginning of this section is avoided simply because the learning is almost adiabatic. It is not clear if other methods can do better, or if there are some fundamental reasons for quantum learning to be adiabatic.

3 Most relevant theories: optimization

Here we analyze some optimization methods since the training stage of ML heavily relies on optimization. Optimization is a big research area in computer science. In quantum computing, the study of optimization problems is not that popular. Instead, popular ones are group problems, linear algebra, quantum simulation etc.

A seminal approach for optimization problem is the adiabatic quantum computing (AQC). The idea of AQC is to encode the solution of a problem into a target state $|\psi\rangle$, which is the unique ground state of a Hamiltonian $H(t)$ at a special value of t. The model $H(t)$ usually takes the form

$$H(t) = (1-t)H_0 + tH_1 \tag{3}$$

for $t \in [0,1]$ for simplicity. Note here t does not need to be time itself; instead, it could be any variable that depends on time. An algorithm uses the adiabatic evolution of $H(t)$ which starts from the ground state of H_0 and ends up with the ground state of H_1. The model $H(t)$ is assumed to be gapped so that the state will not jump to other excited states during the evolution. A seminal example of AQC is quantum annealing, which we will not introduce it here.

As adiabatic process is not easy to realize, we can try to mimic it. Recently a quantum approximate optimization algorithm (QAOA) is proposed which can simulate adiabatic process. The QAOA is a hybrid algorithm in which a classical optimization is used to optimize the parameters of a quantum circuit, which is the alternating Hamiltonian evolution $e^{it_1H_1}$ and $e^{it_2H_2}$. The term H_2 is the cost-function operator, which is measured at each run of the circuit, and the outcome is feedforward to the classical optimizer to adjust the parameters t_1 and t_2 (and also others). The goal is to minimize the value of H_2, which is the cost. We see that the role of the quantum circuit is to produce the value of the cost, and this may have a speedup compared with classical circuits. We could apply QAOA to the training stage of ML, and this is expected to demonstrate the ability of small quantum computers to boost ML.

4 Frontiers

Here we discuss several hot topics in the field of QML. First, the classical ML still has lots of space to improve. For instance, one problem is that the learning slows down as the training goes on. This is somehow analog with people: our interest on new knowledge also decays as you know more. However, this could be improved by adjusting the learning process. How to improve the learning process is a big topic, and the meta-algorithms is one of those schemes.

Another direction is to find and prove speedup for HML. This will likely dominate the field of QML, although it is not fully quantum. A fully QML will need thousands of qubits, which could not be realized in a decade. Currently, the concept of quantum NN is more or less established. Future work of HML applied to real problems will need massive numerical simulations. This will benefit some fields in physics with big data such as particle physics.

The next frontier is of course to understand the fully QML in theory. The method we mentioned above uses a somehow 2nd quantized system to train a 1st quantized system. This should not be the only way. A different approach can encode a quantum process into quantum states, and treat the states as the output of the training algorithm. Whether this works or not needs further study.

5 History, people, and story

The idea of neural networks was created very early in the 1950s, while the real progress was made around 2000. With supercomputers, AlphaGo trained by deep learning beat world's top Go players. And the three key figures, Y. Bengio, G. Hinton, and Y. LeCun won the 2018 Turing Award. This greatly boosts the field of artificial intelligence.

Quantum computing also blooms around 2000, while until now there still has not been a quantum computer that can solve real problems much faster. Maybe the main reason is that quantum computing is a totally revolutionary theory, and people do not know what are the problems that suits it. In this regard, QML is one candidate.

There was a hope recently that HML may have a speedup for recommendation problem, but this was shoot down by a teenager E. Tang, who finds a fast classical algorithm comparable to quantum ones around 2018. She got

to work on this as a project assigned by her supervisor S. Aaronson. The background is that HML based on quantum linear-system solver is expected to have an exponential speedup, but this is not proven. This is very hard to prove. Instead, Tang found an efficient quantum-inspired classical algorithm, which is fairly simple. So we learn that it is so important to work on the right problem at the right time with the right people.

A bit more about the reason. Quantum linear-system solver may have exponential speedup for processing data, but once it has to load or read data, the speedup is lost. This appears as a limitation of quantum algorithms, while also raises the question: will the fully quantum ML be better and also suitable to solve real-world classical problems?

References

- M. Nielsen, Neural networks and deep learning, 2015.
- I. Goodfellow, Y. Bengio, A. Courville, Deep learning, 2016.

Concept map