Training a Quantum Neural Network

Bob Ricks
Department of Computer Science
Brigham Young University
Provo, UT 84602
cyberbob@cs.byu.edu

Dan Ventura
Department of Computer Science
Brigham Young University
Provo, UT 84602
ventura@cs.byu.edu

Abstract

Quantum learning holds great promise for the field of machine intelligence. The most studied quantum learning algorithm is the quantum neural network. Many such models have been proposed, yet none has become a standard. In addition, these models usually leave out many details, often excluding how they intend to train their networks. This paper discusses one approach to the problem and what advantages it would have over classical networks.

1 Introduction

One way to leverage quantum computation in the field of machine intelligence is the idea of a quantum neural network. Many prototypes for quantum neural networks have been proposed. Some of them are very similar to their classical counterparts [1], while others use quantum operators that have no classical equivalents, such as phase shifts [2] [3]. Quantum neural networks have been proposed with a wide variety of different network structures. The most common are lattices [4] and dots [3], as well as the standard feed-forward designs [5].

Almost universally, quantum models either do not mention how the networks are trained or simply state that they use a standard gradient descent algorithm. This assumes that training a quantum neural network will be straightforward and analogous to classical methods. While it would probably not be too hard to determine "the direction that reduces the error for each training pair" [6], as Behrman et. al. do, it is certainly not trivial. Even worse, they propose "summing the amplitudes for all possible 2^{2n} states" [6] in order to do so. Altaisky does mention possible training rules and the feasibility of such training in the quantum world [2] for one of his networks, but that is about the extent of the research devoted to training these networks.

Some early results from simulations of quantum neural networks (QNN) are very optimistic. It has been shown that QNN should have roughly the same computational power as ANN [5]. Other results have shown that QNN may work best with some classical components as well as quantum components [1]. Overall the results have been positive, but there are still a lot of unknowns.

2 Quantum Computation

Several necessary ideas that form the basis for the study of quantum computation are briefly reviewed here. For a good treatment of the subject, see [7].

2.1 Linear Superposition

Linear superposition is closely related to the familiar mathematical principle of linear combination of vectors. Quantum systems are described by a wave function ψ that exists in a Hilbert space. The Hilbert space has a set of states, $|\phi_i\rangle$, that form a basis, and the system is described by a quantum state $|\psi\rangle = \sum_i c_i |\phi_i\rangle$. $|\psi\rangle$ is said to be coherent or to be in a linear superposition of the basis states $|\phi_i\rangle$, and in general the coefficients c_i are complex. A postulate of quantum mechanics is that if a coherent system interacts in any way with its environment (by being measured, for example), the superposition is destroyed. This loss of coherence is governed by the wave function ψ. The coefficients c_i are called probability amplitudes, and $|c_i|^2$ gives the probability of $|\psi\rangle$ being measured in the state $|\phi_i\rangle$. Note that the wave function ψ describes a real physical system that must collapse to exactly one basis state. Therefore, the probabilities governed by the amplitudes c_i must sum to unity. A two-state quantum system is used as the basic unit of quantum computation. Such a system is referred to as a quantum bit or qubit and naming the two states $|0\rangle$ and $|1\rangle$, it is easy to see why this is so.

2.2 Operators

Operators on a Hilbert space describe how one wave function is changed into another and they may be represented as matrices acting on vectors (the notation $|\cdot\rangle$ indicates a column vector and the $\langle\cdot|$ a [complex conjugate] row vector). Using operators, an eigenvalue equation can be written $A|\phi_i\rangle = a_i|\phi_i\rangle$, where a_i is the eigenvalue. The solutions $|\phi_i\rangle$ to such an equation are called eigenstates and can be used to construct the basis of a Hilbert space as discussed in Section 2.1. In the quantum formalism, all properties are represented as operators whose eigenstates are the basis for the Hilbert space associated with that property and whose eigenvalues are the quantum allowed values for that property. It is important to note that operators in quantum mechanics must be linear operators and further that they must be unitary.

2.3 Interference

Interference is a familiar wave phenomenon. Wave peaks that are in phase interfere constructively while those that are out of phase interfere destructively. This is a phenomenon common to all kinds of wave mechanics from water waves to optics. The well known double slit experiment demonstrates empirically that at the quantum level interference also applies to the probability waves of quantum mechanics. The wave function interferes with itself through the action of an operator – the different parts of the wave function interfere constructively or destructively according to their relative phases just like any other kind of wave.

2.4 Entanglement

Entanglement is the potential for quantum systems to exhibit correlations that cannot be accounted for classically. From a computational standpoint, entanglement seems intuitive enough – it is simply the fact that correlations can exist between different qubits – for example if one qubit is in the $|1\rangle$ state, another will be in the $|1\rangle$ state. However, from a physical standpoint, entanglement is little understood. The questions of what exactly it is and how

it works are still not resolved. What makes it so powerful (and so little understood) is the fact that since quantum states exist as superpositions, these correlations exist in superposition as well. When coherence is lost, the proper correlation is somehow communicated between the qubits, and it is this "communication" that is the crux of entanglement. Mathematically, entanglement may be described using the density matrix formalism. The density matrix ρ_ψ of a quantum state $|\psi\rangle$ is defined as $\rho_\psi = |\psi\rangle\langle\psi|$ For example, the quantum state $|\xi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle$ appears in vector form as $|\xi\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\\0\\0\end{pmatrix}$ and it may also be represented as the density matrix $\rho_\xi = |\xi\rangle\langle\xi| = \frac{1}{2}\begin{pmatrix}1&1&0&0\\1&1&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}$ while the state $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ is represented as $\rho_\psi = |\psi\rangle\langle\psi| = \frac{1}{2}\begin{pmatrix}1&0&0&1\\0&0&0&0\\0&0&0&0\\1&0&0&1\end{pmatrix}$ where the matrices and vectors are indexed by the state labels 00,..., 11. Notice that ρ_ξ can be factorized as $\rho_\xi = \frac{1}{2}\left(\begin{pmatrix}1&0\\0&0\end{pmatrix}\otimes\begin{pmatrix}1&1\\1&1\end{pmatrix}\right)$ where $\otimes$ is the normal tensor product. On the other hand, ρ_ψ can not be factorized. States that can not be factorized are said to be entangled, while those that can be factorized are not. There are different degrees of entanglement and much work has been done on better understanding and quantifying it [8, 9]. -Finally, it should be mentioned that while interference is a quantum property that has a classical cousin, entanglement is a completely quantum phenomenon for which there is no classical analog. It has proven to be a powerful computational resource in some cases and a major hindrance in others.

To summarize, quantum computation can be defined as representing the problem to be solved in the language of quantum states and then producing operators that drive the system (via interference and entanglement) to a final state such that when the system is observed there is a high probability of finding a solution.

2.5 An Example – Quantum Search

One of the best known quantum algorithms searches an unordered database quadratically faster than any classical method [10] [11]. The algorithm begins with a superposition of all N data items and depends upon an oracle that can recognize the target of the search. Classically, searching such a database requires $O(N)$ oracle calls; however, on a quantum computer, the task requires only $O(\sqrt{N})$ oracle calls. Each oracle call consists of a quantum operator that inverts the phase of the search target. An "inversion about average" operator then shifts amplitude towards the target state. After $\pi/4 * \sqrt{N}$ repetitions of this process, the system is measured and with high probability, the desired datum is the result.

3 A Simple Quantum Neural Network

We propose a simple QNN with features that make it easy for us to model, yet powerful enough to leverage quantum physics. The first feature we would like is to have a QNN that is able to use known quantum algorithms and gates whenever possible. Second, we would like a QNN that will not require fancy measurements (see [3]) that are currently impractical. Another useful, and related, feature would be the ability to measure the weight of each neuron in the QNN. Finally, we would like to be able to simulate a moderately

sized QNN with a reasonable amount of accuracy in a reasonable amount of time. Another feature would be the ability to transfer learning from the QNN to classical systems.

Our QNN operates much like a classical ANN composed of several layers of perceptrons – an input layer, one or more hidden layers and an output layer. Each layer is fully connected to the previous layer. Each hidden layer computes a weighted sum of the outputs of the previous layer. If this is sum above a threshold, the node goes high, otherwise it stays low. The output layer does the same thing as the hidden layer(s), except that it also checks its accuracy against the target output of the network. The network as a whole computes a function by checking which output bit is high. There are no checks to make sure exactly one output is high.

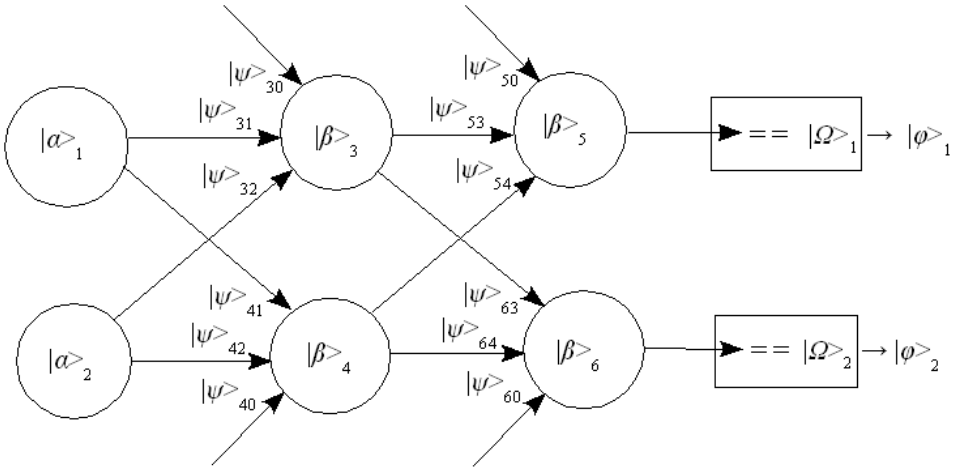

Figure 1: Simple QNN to compute XOR function

The QNN in Figure 1 is an example of such a network, with sufficient complexity to compute the XOR function. Each input in this example is a bit, although more bits could be used for each input register. Each input node i is represented by a register, $|\alpha\rangle_i$. The two hidden nodes compute a weighted sum of the inputs, $|\psi\rangle_{i1}$ and $|\psi\rangle_{i2}$, and compare the sum to a threshold weight, $|\psi\rangle_{i0}$. If the weighted sum is greater than the threshold the node goes high. The $|\beta\rangle_k$ represent internal calculations that take place at each node. The output layer works similarly, taking a weighted sum of the hidden nodes and checking against a threshold. The QNN then checks each computed output and compares it to the target output, $|\Omega\rangle_j$ sending $|\varphi\rangle_j$ high when they are equivalent. The performance of the network is denoted by $|\rho\rangle$, which is the number of computed outputs equivalent to their corresponding target output.

The overall network works as follows on a training set. In our example, the network has two input parameters, so all n training examples will have two input registers. These are represented as $|\alpha\rangle_{11}$ to $|\alpha\rangle_{n2}$. The target answers are kept in registers $|\Omega\rangle_{11}$ to $|\Omega\rangle_{n2}$. Each hidden or output node has a weight vector, represented by $|\psi\rangle_i$, each vector containing weights for each of its inputs. After classifying a training example, the registers $|\varphi\rangle_1$ and $|\varphi\rangle_2$ reflect the networks ability to classify that the training example. As a simple measure of performance, we increment $|\rho\rangle$ by the sum of all $|\varphi\rangle_i$. When all training examples have been classified, $|\rho\rangle$ will be the sum of the output nodes that have the correct answer throughout the training set and will range between zero and the number of training examples times the number of output nodes.

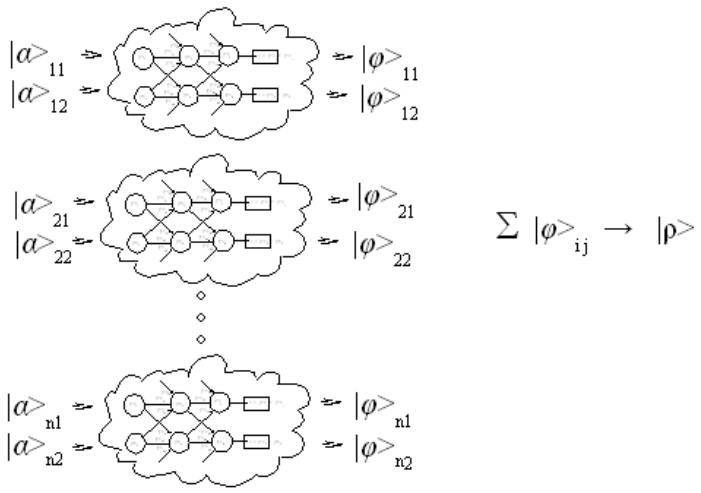

Figure 2: QNN Training

4 Using Quantum Search to Learn Network Weights

One possibility for training this kind of a network is to search through the possible weight vectors for one which is consistent with the training data. We would like to find a solution which classifies all training examples correctly; in other words we would like $|\rho\rangle = n * m$ where n is the number of training examples and m is the number of output nodes. Since we generally do not know how many weight vectors will do this, we use a generalization of the original search algorithm [12], intended for problems where the number of solutions t is unknown. The basic idea is that we will put $|\psi\rangle$ into a superposition of all possible weight vectors and search for one which classifies all training examples correctly.

We start out with $|\psi\rangle$ as a superposition of all possible weight vectors. All other registers ($|\beta\rangle$, $|\varphi\rangle$, $|\rho\rangle$), besides the inputs and target outputs are initialized to the state $|0\rangle$. We then classify each training example, updating the performance register, $|\rho\rangle$. By using a superposition we classify the training examples with respect to every possible weight vector simultaneously. Each weight vector is now entangled with $|\rho\rangle$ in such a way that $|\rho\rangle$ corresponds with how well every weight vector classifies all the training data. In this case, the oracle for the quantum search is $|\rho\rangle = n * m$, which corresponds to searching for a weight vector which correctly classifies the entire set.

There are at least two things about this algorithm that are still undesirable. First, not all training data will have any solution networks that correctly classify all training instances. This means that nothing will be marked by the search oracle, so every weight vector will have an equal chance of being measured. It is also possible that even when a solution does exist, it is not desirable because it overfits the training data. Second, the amount of time needed to find a vector which correctly classifies the training set is $O(\sqrt{2^b/t})$, which has exponential complexity with respect to the number of bits in the weight vector.

One way to deal with the first problem is to search until we find a solution which covers an acceptable percentage, p, of the training data. In other words, the search oracle is modified to be $|\rho\rangle \geq n * m * p$.

5 Piecewise Weight Learning

The quantum search algorithm gives us a good polynomial speed-up to the exponential task of finding a solution to the QNN. Unfortunately, in an arbitrarily large QNN, it will still require exponential time in the size of the composite weight vector. Therefore, we propose a randomized training algorithm which searches each node's weight vector independently.

[Insert section about why we need to use a Piecewise Algorithm]

We start off, once again with training examples in $|\alpha\rangle$, the corresponding answers in $|\Omega\rangle$, and zeros in all the other registers. We randomly select one of the nodes and put its weight vector, $|\psi\rangle_i$ into superposition. All other weight vectors start with random initial weights or can be initialized to zero. We then search for a weight vector for this node that causes the entire network to classify a certain percentage, p, of the training examples correctly. This is repeated, iteratively decreasing p, until a new weight vector is found. Then we train another node using the new weight vector for the last node.

Searching each node's weight vector separately is, in effect, a random search through the weight space where we select weight vectors which give a good level of performance for each node. We expect that all nodes will take on weight vectors that tend to increase performance with some amount of randomness to help keep us out of local minima. This search can be terminated when an acceptable level of performance has been reached.

There are a few improvements to the basic design which help speed convergence. First, we want to insure that hidden nodes find weight vectors that compute something useful, whether or not any of the output nodes are set up to take advantage of that. We accomplish this by adding a small performance penalty to weight vectors which cause the hidden node to output a constant value. This helps select weight vectors which contain useful information for the output nodes. Another improvement is that we only count the performance of the one node when we are working with an output node. Since each output node's performance is independent of all other output node's performance we don't need to worry about how well the other nodes are classifying the data.

We can also add a few extra nodes to search a larger proportion of the space at once. It is especially useful to add a few extra output nodes which have the same target output as one of the original targets. Since there are often a number of weight vectors which will achieve the same level of performance, the different output nodes will often have different weight vectors. The hidden nodes will then be able to improve against both of these output nodes and should have a better chance of finding a solution.

6 Results

The XOR problem is a good problem for simple neural networks because it requires at least two layers, yet it is simple enough to see what is going on. Each of the hidden and output nodes are thresholded nodes with three weights, one for each of its two inputs and one for the threshold. Since we need negative as well as positive numbers and zero, we use 2 qubits for each weight.

The randomized search algorithm also did well on the XOR problem. After an average of 58 epochs, the algorithm was able to cover the training data. Since this is a randomized algorithm both in the number of iterations of the search algorithm before measuring and the order in which the nodes update their weight vectors, the standard deviation for this method was much higher, but still reasonable at 64.836.

We also tried the randomized search algorithm for a problem which was much larger than the XOR problem, the iris data set. In this data set there are four input parameters which

are classified as three different species of irises. To do this we created a QNN with four input nodes, three hidden nodes and three output nodes. Since the inputs are real numbers, we used non-thresholded hidden nodes and then thresholding the outputs. We assume that floating point could be done in a quantum system, although the inputs could be converted to fixed point or integers numbers as well. The weights were all 2 qubit integers as in the XOR problem.

After an average of 74 epochs, the randomized search algorithm was able to achieve 95% accuracy on the training set. Considering that we are using 2 qubit integer weights for a floating-point problem that is pretty good. Backpropagation classified 98% of the training set correctly after around 300 epochs.

7 Conclusions and Future Work

We propose a method of training quantum neural networks which entangles a minimum number of qubits to classify the entire training set. The algorithm is able to search for solutions that cover an arbitrary percentage of the training set. This could be very useful for problems which require a very accurate solution. The drawback is that it is an exponential algorithm, even with the significant quantum decrease in complexity.

In order to avoid some of the exponential increases in complexity with problem size, we have also proposed a randomized version. This algorithm is exponential in the number of qubits of each node's weight vector instead of the composite weight vector of the entire network. This enables the complexity of the algorithm to increase with the number of connections to a node and the precision of each individual weight. This dramatically decreases complexity for problems with large numbers of nodes, while increasing the number of epochs needed to train the network. For problems with a large number of nodes and/or small training sets this could be a great improvement. Preliminary results for both algorithms have been very positive.

There may be quantum methods which could be used to improve current gradient descent and other learning algorithms. It may also be possible to combine some of these with a quantum search. An example would be to use gradient descent to try and refine a composite weight vector found by quantum search. Conversely, a quantum search could start with the weight vector of a gradient descent search. This would allow the search to start with an accurate weight vector and search locally for weight vectors which improve overall performance. Finally the two methods could be used simultaneously to try and take advantage of the benefits of each technique.

Other types of QNNs may be able to use a quantum search as well since the algorithm only requires a weight space which can be searched in superposition. In addition, more traditional gradient descent techniques might benefit from a quantum speed-up themselves.

Future work could also be done improving the existing algorithms proposed in this paper. It may be possible to use amplitudes only for the performance levels instead of each possible weight vector. This would speed up inversion about the average operations and measurement and would probably help reduce round-off errors in calculations. It would also help to have quantum floating point, or at least fixed-point arithmetic to better classify data sets with real numbered inputs.

References

[1] Ajit Narayanan and Tammy Menneer. Quantum artificial neural network architectures and components. In *Information Sciences*, volume 124 nos. 1-4, pages 231–255, 2000.

[2] M. V. Altaisky. Quantum neural network. Technical report, 2001. http://xxx.lanl.gov/quant-ph/0107012.

[3] E. C. Behrman, J. Niemel, J. E. Steck, and S. R. Skinner. A quantum dot neural network. In *Proceedings of the 4th Workshop on Physics of Computation*, pages 22–24. Boston, 1996.

[4] Yukari Fujita and Tetsuo Matsui. Quantum gauged neural network: U(1) gauge theory. Technical report, 2002. http://xxx.lanl.gov/cond-mat/0207023.

[5] Sanjay Gupta and R. K. P. Zia. Quantum neural networks. Technical report, 2002. http://xxx.lanl.gov/quant-ph/0201144.

[6] E. C. Behrman, V. Chandrasheka, Z. Wank, C. K. Belur, J. E. Steck, and S. R. Skinner. A quantum neural network computes entanglement. Technical report, 2002. http://xxx.lanl.gov/quant-ph/0202131.

[7] Michael A. Nielsen and Isaac L. Chuang. Quantum computation and quantum information. Cambridge University Press, 2000.

[8] V. Vedral, M. B. Plenio, M. A. Rippin, and P. L. Knight. Quantifying entanglement. In *Physical Review Letters*, volume 78(12), pages 2275–2279, 1997.

[9] R. Jozsa. Entanglement and quantum computation. In S. Hugget, L. Mason, K.P. Tod, T. Tsou, and N.M.J. Woodhouse, editors, *The Geometric Universe*, pages 369–379. Oxford University Press, 1998.

[10] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th ACM STOC*, pages 212–219, 1996.

[11] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. In *Physical Review Letters*, volume 78, pages 325–328, 1997.

[12] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. In *Proceedings of the Fourth Workshop on Physics and Computation*, pages 36–43, 1996.