Quantum Associative Memory

Troy Astorino
MIT Department of Physics, 77 Massachusetts Ave., Cambridge, MA 02139-4307
(Dated: May 4, 2012)

This paper investigates quantum associative memory (QuAM), a result from the young field of quantum neural networks. As background material we give an overview of two areas of research that give rise to quantum neural networks: quantum computation and artificial neural networks. We compare our main result, a QuAM system, with a classical analog of Hopfield networks, and find the QuAM improves on memory requirements for pattern training exponentially.

I. Introduction

Quantum information science is a field that emerged in the 20^{th} century as a natural extension of classical information science to the realm of quantum phenomena. Much of the interest in quantum information has been stimulated by the potential applications of quantum computing: quantum algorithms that are exponentially faster than their classical counterparts, and exotic-sounding effects such as quantum teleportation. The most famous and celebrated result of quantum computation is arguably Shor's discovery of an algorithm that allows polynomial-time factorization of large integers [14]. A practical implementation of this algorithm would drastically alter current cryptographic techniques that depend on classical algorithms requiring exponential time to factorize integers. Quantum information, however, is a valuable area of study beyond the potential practical uses of quantum computation. A formal study of quantum information and the ways in which it is manipulated provides a deeper understanding of the fundamental physics of quantum systems. By giving us another way to look at physical phenomena, quantum computation gives us a way to more fully understand how the world works.

Part of the initial motivation for the development of artificial neural networks (ANNs) was to gain a greater understanding of how the human brain works. ANNs were first introduced by MuCulloch and Pitts in 1943 [12], and although this original incarnation was a very rough abstraction of true brain function, it was found to be very useful in practical learning problems, especially those involving pattern recognition. Artificial neural networks give a relatively straightforward learning procedure that allows large, nonlinear mappings from inputs to outputs to arise. This same basic model, with some nuanced variations, is still used to great effect today.

In modeling the brain, ANNs are in effect a large system of neuron representations and the connections between them. Because each neuron sends information to every other neuron it is connected to, and all the neurons are sending information continuously, this is a massively parallel system. Conventional computers are based on sequential execution, and so attempts to simulate this parallelism become very computationally expensive with increased ANN size. Quantum computers, however, could take advantage of what is termed quantum parallelism. Quantum parallelism refers to unitary operators acting on all states in a superposition simultaneously; if each state represents information, this is effectively simultaneous manipulation of multiple pieces of information. This gives hope an implementation of a neural network on a quantum system, a quantum neural network, could take advantage of quantum parallelism to achieve exponential performance improvements.

Quantum associative memory (QuAM) is an example of this kind of performance improvements. Hopfield networks, first introduced in 1982[8], are a widely-used classical associative memory (Section III B). A QuAM, a quantum analog of a Hopfield network, requires exponentially less memory to store a set of patterns, a small indication of the potential for quantum neural networks.

Before we can understand how QuAMs function, we must develop the necessary background. We start with quantum computation, beginning with the fundamental unit of quantum information, the qubit, and build to the universal quantum computer. We conclude our introduction to quantum computation with a famous quantum algorithm, Grover's search algorithm, as we will require it in a slightly modified form for the QuAM. We then introduce ANNs, using the Hopfield network as an example that gives rise to an associative memory. With this background in place, we finally turn to QuAM, first giving algorithms for encoding and recalling patterns, and then discussing its speedup over the Hopfield network.

II. Quantum Computation

We will use this section both to introduce the basic components of quantum information and to introduce the notation that will be used throughout the rest of the paper.

A. Qubits

The bit, which can take a value of 0 or 1, is the elementary unit of classical information. The quantum analog of the bit is the qubit. The qubit is the elementary unit of quantum information: the information of a two-state system. The two basis states of this system are labeled $|0\rangle$ and $|1\rangle$. This could, for example, be the spin of an electron, wbere $|0\rangle \equiv |\downarrow\rangle$ and $|1\rangle \equiv |\uparrow\rangle$. This is fundamentally different than the classical bit because the qubit can exist in a superposition: $c_1 |0\rangle + c_2 |1\rangle$. The state of the qubit exists in a 2-dimensional Hilbert space. As such, we can as usual represent the state by a column vector:

$$c_1 |0\rangle + c_2 |1\rangle = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

Note: for notational convenience, in the quantum computation literature a superposition of states will not always be normalized.

When we have a system of multiple qubits, we take the tensor product of the qubits. This is notated in a contracted form.

$$|0\rangle \otimes |1\rangle \equiv |0\rangle |1\rangle \equiv |01\rangle$$

With a multiple qubit system, vectors and matrices are represented in what is called the computational basis. For a two-qubit system, this corresponds to:

$$|00\rangle \equiv \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, |01\rangle \equiv \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, |10\rangle \equiv \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, |11\rangle \equiv \begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

This of course can be extrapolated up to n qubits; the ordering of the states in the computational basis is

$$\{|0...00\rangle, |0...01\rangle, |0...10\rangle, |0...11\rangle, ..., |1...11\rangle\}$$

This basis is used because each state in the basis can represent a binary number, and the ordering of the basis states represents the ordering of binary numbers. In fact, the notation will sometimes be compacted even further by converting the binary representation to base 10, i.e. $|1001\rangle \equiv |9\rangle$. To distinguish between a single qubit in the $|0\rangle$ or $|1\rangle$ state, and a set of qubits in the $|0...00\rangle$ or $|1...111\rangle$ state, we will use $|\bar{0}\rangle$ for $|0...000\rangle$ and $|\bar{1}\rangle$ for $|1...11\rangle$.

Reflecting this binary representation, a system of n qubits gives us a 2^n dimensional Hilbert space of possible states. To arbitrary precision, we can represent the state of any finite quantum system as simply a vector in this Hilbert space by a finite number of qubits. [15] Because of this, we can use qubits as a universal store of quantum information.

B. Quantum Gates

A quantum system evolves through action by unitary operators. In the context of quantum computation, these unitary operations are called quantum logic gates, analogous to the gates of a classical computer. There can be single-qubit and multi-qubit gates, although single-qubit and two-qubit gates are most common.

As you may recall from Section I, unitary operators acting on a superposition of states is sometimes referred to as quantum parallelism in the context of quantum computation.

$$U(c_1 |\psi\rangle + c_2 |\phi\rangle) = c_1 U |\psi\rangle + c_2 U |\phi\rangle$$

This ability for a quantum gate to simultaneously act on multiple states, which represent pieces of information, cannot be simulated on a classical computer without exponential time increases.

The Pauli matrices are commonly used single qubit quantum gates. We notate them as

$$X \equiv \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}, Y \equiv \begin{bmatrix}0 & -i\\i & 0\end{bmatrix}, Z \equiv \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$$

X is the quantum equivalent of the NOT gate, effectively flipping the qubit. Another commonly used single-qubit quantum gate is the Hammard, or Walsh, gate:

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} \tag{1}$$

This gate can create an equally weighted superposition of the two basis states from either one of the basis states.

A very important gate is the controlled not gate, or $CNOT$ gate. It executes a NOT on the second qubit controlled by the first qubit, i.e., it flips the state of the second qubit if the first qubit is in the state $|1\rangle$.

$$CNOT \equiv \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix} \tag{2}$$

The combination of a single-qubit rotation gate and $CNOT$ gate is a universal quantum gate: all possible unitary transformations of n qubits can be accomplished through combinations a single-qubit rotation operator and a $CNOT$ operator[5].

C. Universal Quantum Computer

A universal quantum computer must be able to simulate the execution of any finite quantum system without exponential time costs. We discussed earlier how a set of qubits could represent any quantum state, and how specific application of a universal quantum gate can execute any possible unitary transformation. Steane [15] summarized the rough requirements for a quantum computer:

1. Each qubit can be prepared in some known state, which we will call $|0\rangle$
2. Each qubit can be measured in the basis $\{|0\rangle, |1\rangle\}$
3. A universal quantum gate can be applied to any subset of the qubits
4. The qubits do not evolve other than by desired transformations of the type specified above

In practice, these requirements are very difficult to satisfy, particularly the requirements prohibiting unwanted evolution of the qubits. Difficulty with controlling quantum dechoerence has limited the size of quantum computers; researches have only been able to create quantum computers that have on the order of 10 qubits. Many different systems have been pursued for physical implementations of a quantum computer; quantum computers using ion traps or nuclear magnetic resonance found some early success. Moving forward with this paper, we will assume the algorithms we are developing will operate on a function quantum computer.

D. Grover's Search Algorithm

As we will require a modified version of the generalized Grover search algorithm in order to implement quantum associative memory, we will introduce Grover's algorithm as an illustrative example of a quantum algorithm.

Grover's algorithm performs a search over N items to find a single item that uniquely satisfies some condition. A common example used to illustrate this is database search. For this, we have N addresses, each corresponding to some information held in memory. Given an address, we want to find it in the database so we can retrieve the information corresponding to it. We assume we are dealing with an unstructured list; there isn't any ordering of the addresses that can speed up search.

The best classical algorithm for this problem is a sequential search through the database: going through each address and checking if it is the one we are looking for. This will on average take $N/2$ operations, and so has a time-complexity $O(N)$, i.e., it requires order of N operations. Grover's algorithm, however, achieves a quadratic increase in speed by taking advantage of properties of quantum computation, and has a time-complexity of $O(\sqrt{N})$. For a large database this is a very significant increase: if our database holds one million items, classically we would need to perform order of one million operations, but using Grover's algorithm we only require order of one thousand operations

For the implementation of Grover's algorithm, each address will be represented by a n qubit state, so we will have $N = 2^n$ addresses. We will label states using the computation basis defined in Section II A.

To initialize our system, start in the state $|\bar{0}\rangle$. Apply the Hammard gate (Equation 1) to each of the qubits. This puts the system into a superposition of all the states in the computational basis:

$$|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N} |i\rangle \tag{3}$$

Now we define some operators that will be required for the execution of Grover's algorithm. Let $|j\rangle$ be the item we are looking for. Define U_j to be an operator that rotates the phase of $|j\rangle$ by π, and leaves the rest of the states unchanged:

$$U_j |i\rangle = \begin{cases} |i\rangle, & i \neq j \\ -|j\rangle, & i = j \end{cases} \tag{4}$$

D inverts the amplitude of each component of the state vector about the average amplitude. The components of the matrix representation of D are

$$D_{ij} = \begin{cases} \frac{2}{N} - 1, & i = j \\ \frac{2}{N}, & i \neq j \end{cases} \tag{5}$$

To see why this is an 'amplitude inversion about average' operator, take a state $|c\rangle = (c_1, c_2, ..., c_N)$. Let $|c'\rangle = D|c\rangle$

$$\begin{aligned} c_i{}' &= \left(\frac{2}{N} - 1\right) c_i + \frac{2}{N} \sum_{j \neq i} c_j \\ &= 2\bar{c} - c_i, \qquad \bar{c} = \frac{1}{N} \sum_{j=0}^{N} c_j \\ &= \bar{c} - (c_i - \bar{c}) \end{aligned}$$

Acting on $|\psi_0\rangle$ with U_j effectively marks the state we are looking for by changing the phase of its coefficient, and then acting on $|\psi_0\rangle$ with D increases the value of $|c_j|^2$ while lowering the value of all the other $|c_i|^2$. Because we want of find state $|j\rangle$, we want $\frac{|c_j|}{|c_i|}$ to be as large as possible.We define the new state $|\psi_1\rangle$, so

$$|\psi_1\rangle = DU_j |\psi_0\rangle \tag{6}$$

Looking at this another way, DU_j effectively rotates $|\psi_0\rangle$ in the direction of $|j\rangle$[15]. In order to maximize $|c_j|$, we want to evolve the system to a state that is closest to $|j\rangle$. We will call this system $|\psi_k\rangle$, where $|\psi_k\rangle$ is the state after applying k rotations:

$$|\psi_k\rangle = (DU_j)^k |\psi_0\rangle \tag{7}$$

Applying too many rotations will cause the state vector to begin to move away from $|j\rangle$, and so will cause $|c_j|^2$ to start decreasing. Grover [6] found the optimal k to be

$$k \approx \frac{\pi}{4}\sqrt{N} \tag{8}$$

After evolving to this state, the system can be measured. $|c_j|^2 \sim 1 - \frac{1}{N}$, so we will find state $|j\rangle$ with error probability $O(\frac{1}{N})$. Interestingly, this shows that the probability of a successful measurement increases with the size of the system. If a greater probability of success is desired, the process can be repeated m times and so give an $O(1 - \frac{1}{N^m})$ success probability.

III. Neural Networks

A. Artificial Neural Network Construction

A neural network is a system of neurons and the axons, dendrites, and synapses that connect them. Neurons communicate through changing electrical potentials: when a neuron exhibits a spike in electrical potential it is said to fire, and those spikes travel through axons to stimulate other neurons, possibly leading to other neurons firing. This system of neuronal firing and synaptic transmission is the control system for almost all living organisms, and is where intelligence arises for human beings.

Artificial neural networks try to simulate the information processing features of these biological neural networks. They are composed of nodes, which represent artificial neurons, and weighted, directed edges, which represent connections between neurons. The simplest artificial neural network, called the single-layer perceptron, is shown in Figure 1. It consists of a single neuron, with any number of inputs, and a single output. This generalizes to representing an artificial neuron in almost any artificial neural network. The perceptron performs a computation on the vector of inputs $\mathbf{x}$, parametrized by the weight vector $\mathbf{w}$, in order to produce an output

$$y = f(\phi(\mathbf{w}, \mathbf{x}))$$

Conceptually, the inputs represent firing rates of other neurons, the weights represent synaptic strength, and the output represents the firing rate of the neuron.

Typically $\phi(\mathbf{w}, \mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$, and $f(\varphi)$ will map its input into an acceptable range of output for the neuron, such as $\mathrm{sgn}(\varphi)$, where

$$\mathrm{sgn}(\varphi) \equiv \begin{cases} 1, & \varphi \geq 0 \\ -1 & \varphi < 0 \end{cases}$$

Complex neural networks can be created by having systems of many neurons, and setting the weights on the connections between them. A typical artificial neural network used for

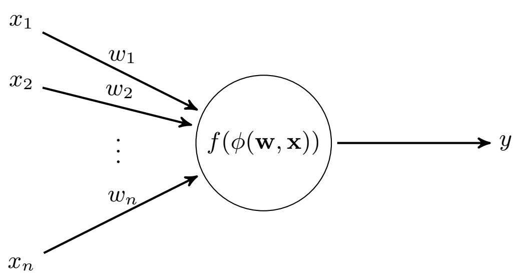

FIG. 1: Single-layer perceptron

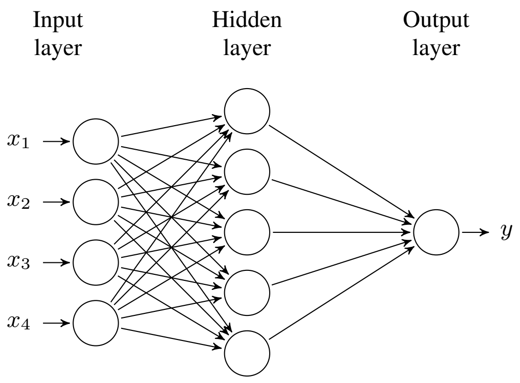

FIG. 2: Feedforward neural network

learning, a feedforward neural network, is shown in Figure 2. It is called feedforward because the output of a given neuron only moves forward through the network.

The weights on connections are usually set through some learning algorithm. A training set of data, $S = \{\mathbf{x}_i, y_i\}$, which has known outputs y_i for inputs $\mathbf{x}_i$ to the system, is used to update the weights. With some set of weights, an input $\mathbf{x}_i$ is fed through the neural network and the true output, y_i, is compared to the generated output: if they are the same, the weights aren't changed, but if they are different, the weights are slightly adjusted according to the magnitude and direction of the difference in the outputs. For more information on neural network learning algorithms see Cheng and Titterington[3].

B. Hopfield networks

A Hopfield network is a system of N fully connected neurons. Each neuron has a connection to every over neuron, as can be seen in Figure 3. The firing rate of all the neurons x_i are updated in virtual timesteps,

$$\mathbf{x}_{t+1} = \text{sgn}(W\mathbf{x}_t) \tag{9}$$

where W is the weight matrix of the connection weights between neurons. Connections between neurons are symmetrical, $w_{ij} = w_{ji}$, and neurons do not self-excite, $w_{ii} = 0$.

A Hopfield network is an example of associative memory. After training the network on a set of patterns, the network will be able to input, which could be a corrupted version of a training patters, and associate it with one of the training patterns. For instance, if we trained the 5 neuron Hopfield network in Figure 3 on the patterns $\{(1,0,0,1,0),(1,1,0,0,1)\}$,

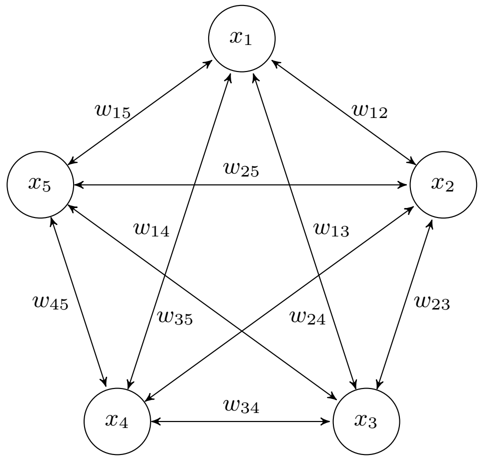

FIG. 3: 5 neuron Hopfield network

and we gave the network an initial input of $(1,1,0,1,1)$, after a few iterations the network would converge to the $(1,1,0,0,1)$.

The network is trained by choosing the weight matrix W. A simple rule called the generalized Hebb rule is used to store patterns in a Hopfield network. If we would like to store p patterns, the weight matrix is built in the following fashion:

$$W = \frac{1}{p}\sum_{k=1}^{p} \mathbf{x}\mathbf{x}^T$$

A Hopfield network can be applied in categorization problems. After training the network on a set of categories, represented by vectors, we can present the network with a noisy input and it will identify the input with one of the categories. Even more intriguing about associative memories, however, is that they operate in a manner thought to be similar to human memory: a stimulus of some pattern becomes associated with some previously remembered pattern.

Unfortunately, Hopfield networks, and other classical associative memory networks, have large memory requirements that limit their utility. Storing length n patterns requires n neurons, and in order for a Hopfield network to have a good pattern association success rate, only $\sim 0.15n$ patterns can be stored[9].

IV. Quantum Associative Memory

The application of quantum computing to artificial neural networks holds great promise. Because ANNs are typically used as prediction systems, they are by nature probabilistic, which aligns with the intrinsic probabilistic nature of quantum computation. Additionally, classical simulations of neural networks are limited by an inability to efficiently implement the parallelism present in real neural networks. Quantum parallelism could to overcome this issue. Simulations of neural networks using quantum computation are quantum neural networks.

There have been many proposals for implementing quantum neural networks, some for implementation in physical systems [1], and some for implementation on a universal quantum computer [7] [11] [4]. We will examine one of the latter types, a QuAM developed by Dan Ventura and Tony Martinez in 2000[16]. Although some proposed QNNs very explicitly draw analogies between the components of the quantum system and the simulated neural network, the QuAM has inexact analogies. We will make analogies to the Hopfield network, because we will ultimately be comparing the performance of the QuAM against the Hopfield network.

The QuAM recalls incomplete patterns, differing slightly from the Hopfield network that recalls slightly corrupted patterns, although both are certainly associative memory systems. For the QuAM, each qubit represents a neuron: in the Hopfield network, a pattern is read from in the firing rates of the n neurons, and in the QuAM, the pattern is read from the state of the qubits. The superposition of the states encodes the connections between different states: the weight matrix of the Hopfield network controls evolution the input pattern to the final state, and the coefficients of the superposition of the states gives the probability of evolving to a certain pattern.

We divide our algorithm for the QuAM into two components: pattern encoding and pattern recall. The first component is the training phase in which we create a superposition of all the patterns in the training set. The second component is the actual operation of the associative memory: given an incomplete pattern, the system will return the closest pattern from the training set.

A. Pattern Encoding

We will first take a high-level overview of how the pattern storage algorithm works, and then we'll examine the operators used.

1. *Overview*

We store p patterns of length n: $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_p\}$. This requires $2n+1$ qubits. We divide these qubits into 3 registers, and will label the basis states by $|r, g, c\rangle$. r has n of the qubits, and is where patterns are stored. g has $n-1$ of the qubits, and is a garbage register used to identify a particular state. We will not examine the use for the garbage register as it adds little the understanding of the development of the algorithm. c has 2 qubits, and holds information about the current state of the algorithm.

The algorithm starts with the first pattern $\mathbf{x}_1$. It selectively acts with the X operator on qubits in the r register until $|r\rangle$ is the same state as $|\mathbf{x}_1\rangle$. A superposition is then created by actions on the control qubits with a modified Hammard gate. The first component of the superposition has coefficient $\frac{1}{\sqrt{p}}$, and is marked to be unchanged by the rest of the algorithm by setting c_2, the second control qubit, to $|1\rangle$. $\frac{1}{\sqrt{p}}$ is the value we want for all the coefficients of the final superposition of the patterns. Though both of these pieces of the superposition have the same state in the r register, the state with $|c_2\rangle = |1\rangle$ will not be changed by the rest of the pattern encoding algorithm, and the state with $|c_2\rangle = |0\rangle$ will have its r register used to encode the rest of the $p-1$ patterns. The second pattern, $\mathbf{x}_2$, is encoded taken next. Now, only the qubits in the r register of the state with $|c_2\rangle = |0\rangle$ are selectively acted on with the X operator until $|r\rangle$ is as $|\mathbf{x}_2\rangle$. A superposition of the component with $|c_2\rangle = |0\rangle$ is created by again acting with the modified Hammard gates on the control bits, and again a piece with coefficient $\frac{1}{\sqrt{p}}$ is marked for saving by setting $|c_2\rangle$ to $|1\rangle$. This process of setting $|r\rangle$ of the state with $|c_2\rangle = |0\rangle$ to be equal to one of the x_i patterns, and saving part of that to be in the final superposition of states, is continued until all p patters have been encoded. At the end of the algorithm, the control qubits and the garbage qubits are all in the same state, and so are unnecessary in the notation of the superposition. We are left with

$$|r\rangle = \frac{1}{\sqrt{p}} \sum_{i=0}^{p} \mathbf{x}_i \tag{10}$$

2. *Detailed operation*

As we present the operators required to execute this algorithm, we will carry through an example aid understanding of how the operators function. Say we are trying to encode 3 two-qubit patterns: $\{|01\rangle, |10\rangle, |11\rangle\}$. Our initial quantum state is

$$|\psi_0\rangle = |00, 0, 00\rangle \tag{11}$$

The pattern setting algorithm applies a series of operators on the system for each pattern x_i. We will index operators with an i subscript as a reminder that each will be applied one time for each pattern encoded.

First introduce the F_i operator, the operator that sets the r register of states with $|c_2\rangle = |0\rangle$ to be equal to $\mathbf{x}_i$. We will require a variant of the $CNOT$ operator (Equation 2) which we will call $CNOT'$. $CNOT'$ acts with an X operator on the second qubit of a two-qubit system if the first qubit is in the state $|0\rangle$.

$$CNOT' \equiv \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

The algorithm proceeds in order, first encoding $\mathbf{x}_1$, then $\mathbf{x}_2$, and so on. Because of this, F_i is dependent on $|r\rangle$ the component of the superposition with $|c_2\rangle = |0\rangle$. Before F_i acts on it, this $|r\rangle$ will be in the $|\mathbf{x}_{i-1}\rangle$ state. We define $\mathbf{x}_0 = |\bar{0}\rangle$ for the $i = 1$ case.

F_i works as follows:

1. For each qubit $x_{i,j}$ in $\mathbf{x}_i$, if $x_{i,j} \neq x_{i-1,j}$, apply the operator $CNOT'^{c_2, r_j}$. The superscripts on the $CNOT'$ operator indicate which qubits the operator will act on.
2. Apply $CNOT'^{c_1, c_2}$

Acting on our sample system (Equation 11) with F_1, where $|x_1\rangle = |01\rangle$, gives:

$$|\psi_{0,F}\rangle = F_1 |\psi_0\rangle = |01, 0, 10\rangle \tag{13}$$

We next create a weighting superposition of states by acting on control qubits. Define

$$H_m = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \sqrt{\frac{m-1}{m}} & \frac{-1}{\sqrt{m}} \\ 0 & 0 & \frac{1}{\sqrt{m}} & \sqrt{\frac{m-1}{m}} \end{bmatrix} \tag{14}$$

where $m = p - i + 1$. In particular, we apply H^{c_1,c_2}_{p-1+1}. H_m can be thought of as similar to a controlled Hammard gate (Equation 1). If $|c_1\rangle = |1\rangle$ qubit is in the state $|1\rangle$, then H_m will create a superposition of the two control qubits, with one having probability amplitude $\frac{1}{m}$.

In our example, we are iterating on the first pattern $|\mathbf{x}_1\rangle = |01\rangle$, so $i = 1$. We apply H_3 to our state in Equation 13

$$|\psi_{0,F,H}\rangle = H_3 |\psi_{0,F}\rangle = \sqrt{\frac{1}{3}} |01,0,11\rangle + \sqrt{\frac{2}{3}} |01,0,10\rangle \tag{15}$$

We see that we have two components differing only in $|c_2\rangle$; the component with $|c_2\rangle = |1\rangle$ will not have its r register modified by the rest of the pattern encoding algorithm.

Our final operator is S_i, which saves the state with $|c_2\rangle = |1\rangle$ in the superposition generated by H^{p-i+1} as one that should remain in the final superposition, and arranges the control qubits so that future F operators do not effect that state. The details of the S_i operator are not very illuminating or instructive, so if the reader desires they can be found in the Ventura et. al paper [16]. S_i behaves as follows:

$$S_i |r, g, c\rangle = \begin{cases} |r, 0..0, 00\rangle, & |c_2\rangle = 1 \\ |r, 0..0, 00\rangle, & |c_2\rangle = 0 \end{cases}$$

Let us examine how S_i acts on our example, continued from Equation 15:

$$|\psi_1\rangle = S_1 |\psi_{0,F,H}\rangle = \sqrt{1/3}\, |01,0,01\rangle + \sqrt{2/3}\, |01,0,00\rangle \tag{16}$$

Because the first component in the superposition has $|c_2\rangle = |1\rangle$, it will not be altered when the other patterns are encoded.

We can now finish example by adding the second and third patterns to the superposition. First apply the operations needed to include $|\mathbf{x}_2\rangle = |10\rangle$:

$$\begin{aligned} |\psi_2\rangle &= S_2 H_2 F_2 |\psi_2\rangle && (17)\\ &= S_2 H_2 (\sqrt{1/3}\, |01,0,01\rangle + \sqrt{2/3}\, |10,0,10\rangle) && (18)\\ &= S_2 (\sqrt{1/3}\, |01,0,01\rangle + \sqrt{1/2}\sqrt{2/3}\, |10,0,11\rangle && (19)\\ &\quad + \sqrt{1/2}\sqrt{2/3}\, |10,0,10\rangle) && (20)\\ &= \sqrt{\frac{1}{3}} |01,0,01\rangle + \sqrt{\frac{1}{3}} |10,0,01\rangle + \sqrt{\frac{1}{3}} |10,0,00\rangle && (21) \end{aligned}$$

As we can see, we have added a component to the superposition where $|r\rangle = |10\rangle$, and have marked is to be saved for the rest of the algorithm by setting $|c\rangle = |01\rangle$.

We finally encode $|\mathbf{x}_3\rangle = |11\rangle$:

$$\begin{aligned} |\psi_3\rangle &= S_3 H_1 F_3 |\psi_2\rangle && (22)\\ &= \sqrt{\frac{1}{3}} |01,0,01\rangle + \sqrt{\frac{1}{3}} |10,0,01\rangle + \sqrt{\frac{1}{3}} |11,0,01\rangle && (23) \end{aligned}$$

The c and g registers of all components of the superposition are now the same, and are thus not necessary to notate the superposition. So we have as our final state

$$|\psi_3\rangle = \frac{1}{\sqrt{3}} |01\rangle + \sqrt{\frac{1}{3}} |10\rangle + \sqrt{\frac{1}{3}} |11\rangle$$

Thus, a series of patterns can be encoded as a quantum state. Now that we have covered training for our associative memory system, we must turn our attention to pattern recall.

B. Pattern Recall

With the patterns trained as we have done above, we will soon see that pattern association is very similar to the list search of Grover's algorithm. We first must make some slight modifications.

1. Modifications of Grover's Search Algorithm

Grover's algorithm assumes that search will occur over a uniform superposition of states, i.e., all of the states have equal probability amplitudes. In our system, however, we have initialized certain states corresponding to patterns in the training set to have coefficient $\frac{1}{\sqrt{p}}$, and to have all the other state have coefficient 0, and so we have some initial distribution over the states that we do not want to lose. A generalization of Grover's search algorithm to handle arbitrary initial amplitude distributions was first suggested by Biron et al[2], and was refined by Ventura et. al [16].

We introduce a new operator, $U_{\mathcal{P}}$, which is similar to the U operator of Equation 4. Instead of rotating a single state $|j\rangle$ by π, it rotates all p states in the set $\mathcal{P} = \{|\mathbf{x}_1\rangle, |\mathbf{x}_2\rangle, ..., |\mathbf{x}_p\rangle\}$ by π. Now say we have an incomplete pattern that will serve as the input to our our associative memory, $|011?\rangle$, i.e., the 4^{th} qubit is unknown. Let ρ be all the patterns in our training set, and τ be $\{|0110\rangle, |0111\rangle$. Our algorithm for pattern recall is:

$$|\psi_k\rangle = (DU_\tau)^k DU_\rho DU_\tau |\psi\rangle$$

with

$$k \approx \frac{\pi}{4}\sqrt{N} - 2$$

U_τ is directly analogous to U_j of Grover's search algorithm, only differing in that we must search for a set of states instead of a single one. U_ρ is necessary because of the D 'inversion about average' operator. Without the application of U_ρ, the states not in the training set would develop coefficients of comparable magnitude to the states in the training set that are not being searched for. This increase of coefficients would decrease the accuracy of the algorithm by lowering the probability that the desired state would be measured at the end of execution. Applying U_ρ instead of the second U_τ ensures that the coefficients of the states in the training set stay larger than the other coefficients, and so maintain the high accuracy of the unmodified Grover's algorithm.

V. Discussion

This system provides an exponential decrease in the memory required to store m patterns. With patterns of length n, it takes $O(mn)$ operations for pattern encoding, and $O(\sqrt{N})$ for pattern recall, where $N = 2^n$. The large gain made possible by the QuAM is in the amount of memory required to store patterns. Because the patterns exist as a superposition of states, we can theoretically have 2^n stable states. This is an exponential increase over the $\sim 0.15n$ patterns that can be stably stored in a Hopfield network. We would not actually want to train the fully possible 2^n patterns on our QuAM, as this would mean all patterns are equally likely and reduce to being a problem of list search. Even if we trained with 2^{n-2} of the possible patterns, which could be a useful number, we would exponentially more patterns than would be possible on a Hopfield network of the same size. This a significant gain over the classical alternatives.

A great advance for QuAM research would be validation through implementation on a physical quantum computer. This, of course, requires the continued progress in the general field of quantum computation. In the general field of QNNs, there is much work needed in unification of the many different approaches and representations of QNNs. A rigorous foundation foundations for the ideas of QNNs could help to provide for this need. Quantum neural networks are promising structures offer the hope for more practical algorithms for quantum computation, and faster learning systems than exist today.

Acknowledgments

I would like to thank Gus Downs all the time and help he gave me in developing and iterating on this paper, and Maksym Serbyn for his comments that helped me bring the components this paper together.

[1] E.C. Behrman, J. Nimel, J.E. Steck, and S.R. Skinner. A quantum dot neural network. *Complex Systems*, 19, 1996.

[2] David Biron, Ofer Biham, Eli Biham, Markus Grassl, and Daniel Lidar. Generalized grover search algorithm for arbitrary initial amplitude distribution. In Colin Williams, editor, *Quantum Computing and Quantum Communications*, volume 1509 of *Lecture Notes in Computer Science*, pages 140–147. Springer Berlin / Heidelberg, 1999.

[3] Bing Cheng and D. M. Titterington. Neural networks: A review from a statistical perspective. *Statistical Science*, 9(1):pp. 2–30, 1994.

[4] Adenilton J. da Silva, Wilson R. de Oliveira, and Teresa B. Ludermir. Classical and superposed learning for quantum weightless neural networks. *Neurocomputing*, 75(1):52 – 60, 2012.

[5] D. Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[6] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM.

[7] Sanjay Gupta and R.K.P. Zia. Quantum neural networks. *Journal of Computer and System Sciences*, 63(3):355 – 383, 2001.

[8] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[9] A.K. Jain, Jianchang Mao, and K.M. Mohiuddin. Artificial neural networks: a tutorial. *Computer*, 29(3):31 –44, mar 1996.

[10] Subhash C. Kak. Quantum neural computing. In Peter W. Hawkes, editor, *Advances in Imaging and Electron Physics*, volume 94, pages 259 – 313. Elsevier, 1995.

[11] Noriaki Kouda, Nobuyuki Matsui, Haruhiko Nishimura, and Ferdinand Peper. Qubit neural network and its learning efficiency. *Neural Computing & Applications*, 14:114–121, 2005.

[12] Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5:115–133, 1943.

[13] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 1 edition, 2000.

[14] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134, nov 1994.

[15] Andrew Steane. Quantum computing. *Reports on Progress in Physics*, 61(2):117, 1998.

[16] Dan Ventura and Tony Martinez. Quantum associative memory. *Information Sciences*, 124(14):273 – 296, 2000.