Imperial College London
Department of Physics

Jesse A Garman
October 6, 2011

Submitted in partial fulfilment of the requirements for the
degree of Master of Science of Imperial College London

---

# A Heuristic Review of Quantum Neural Networks

---

This paper contains a brief review of research into quantum neural networks (QNNs). A summary of foundational material in quantum computation and neural networks is given. The motivations behind QNNs are discussed and a number of QNN models and their properties are examined.

Supervisor:
Dr. Terry G. Rudolph

Assessor:
Prof. Henrik J. Jensen

# Contents

# 1 Introduction

In 1982, in a keynote speech to a conference of physicists, Richard Feynman proposed that in order to simulate quantum mechanical systems effectively, computers which used quantum phenomena to perform calculations might be needed [1]. In the years that followed, through the work of David Deutsch, Peter Shor, and many others, the field of *quantum computation* emerged, with indications that such computers were not only physically constructible, but may in some applications be exponentially faster than their classical counterparts, and today the field is one of the fastest growing in physics.

In 1982, John Hopfield published his work on an associative memory network [2], in what was a culmination of decades of research worldwide into *neural networks*: layers of interconnected processing units whose behaviour was modelled on the biological neurons of the human brain, and which excel over conventional computers in many-input tasks such as pattern recognition. Such parallel processors have been used in a variety of applications, ranging from modelling stock prices to basic artificial intelligence, and through periods of varying expectations are entering more practical domains.

In the last fifteen years there has been some study into models which merge these spheres, in the form of quantum neural networks (QNNs): neural networks which use a quantum formalism in their operation. The hope behind these models is that such a network may gain some of the advantages of a quantum computer, as naïvely one might hope that a neural network could make fuller use of quantum parallelism, or allow a physical implementation which is more automated and easier to construct than a quantum computer. As evocative as quantum computing and neural networks both are in history, with promises both great and, in the latter case, greatly forestalled, it is perhaps not surprising that even professional researchers have taken QNNs as a certainly useful field and have attempted to argue as such without themselves being experts in both. As much research has been motivated by implicit endorsement of unproven hypotheses,

or by earlier results which suggest an almost fantastic use of quantum behaviour, the author believes that a more critical review of some models would be useful.

The aim of this dissertation is to offer a brief but representative review of a range of proposed models of quantum neural network, without covering proposed applications. After this introductory chapter, Sections 2 and 3 cover foundational material as relevant to the models discussed, briefly covering the most important results of quantum information theory and neural network theory respectively. Section 4 briefly introduces QNNs with some further discussion on the motivations in studying it, and on what attributes such networks may possess. Section 5 then discusses four QNN models in depth, covering a range of physical interpretations. Some final remarks and areas of further review are given in Section 6.

The author would like to make clear that this review is by no means complete, in the sense that not all proposed models of QNNs have been covered, and of those not to their full depth. Ideally a review would explore the proposed applications of QNNs, some of which have been studied by authors of the models in Section 5 and would perhaps justify their statements on the possible use of QNNs. Some more complex forms of neural network behaviour described in Section 3 are explored in these papers; for example, Behrman et al. and Ventura and Martinez have written on different methods of associative memory recall in QNNs, of which only the latter is covered as it encompasses a major part of the model's formulation, whereas Behrman et al. develop their model more generally. However, in the scope and duration of this dissertation it is not feasible to cover a wide range of such applications, and it is not appropriate to select a few at the exclusion of others as this would fall short of a useful minimum range and overrepresent some models. This is left for any extension to this review, which it is the author's intention to complete at length after submitting this more constrained version.

## 2 Quantum Computation

The field of quantum information studies how quantum mechanical effects can be used to process information, usually in a way which fundamentally differs from classical information [3]. The most basic principles in the operation of quantum computers as required in this review are covered here. This section is partially based on sections in the comprehensive title *Quantum Computation and Quantum Information* by Nielsen and Chuang [3], including figures on quantum gates, and assumes a good understanding of quantum mechanics and linear algebra.

### 2.1 Qubits

In classical information, all information is mediated through binary objects called *bits*, which can hold the value 0 or 1. By relating these to high and low voltages in a wire, these can be used in *logic gates* formed from transistors to perform useful calculations, forming the basis for modern computers.

In quantum information, the fundamental object used is the *quantum bit*, or *qubit*, which can take the state $|0\rangle$, $|1\rangle$, or a *superposition* of the two, so that generally

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1}$$

where $|\psi\rangle$ is the qubit state and $\alpha$ and $\beta$ are the complex probability amplitudes of the states $|0\rangle$ and $|1\rangle$ respectively, such that $|\alpha|^2 + |\beta|^2 = 1$. If we were to measure the qubit at any time, we would obtain either the state $|0\rangle$ or $|1\rangle$, similarly to a bit, as we have *collapsed* the superposition just as in quantum mechanics. However, if we do not measure the qubit, we find that some of the unusual behaviour of quantum objects will enter into any computer using qubits, so that quantum logic gates form very different species from their classical counterparts.

In years since the invention of quantum mechanics, it has become clear that the

intrinsically weird logical connection that can exist between two quantum particles, their *entanglement*, is itself perhaps the chief resource of quantum information theory, to which there is no counterpart in classical information theory. The 'simplest' form of entanglement, most commonly discussed in quantum mechanics, appears in the two-qubit Bell states $|\phi^{\pm}\rangle = \frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle)$ and $|\psi^{\pm}\rangle = \frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle)$, whose qubits are always correlated (or anticorrelated) so that, if the Bell state is known, then by observing one qubit the other will necessarily and instantaneously collapse to the same (or opposite) state. These states are considered to be maximally entangled, in that any entangled state $\alpha|00\rangle + \beta|11\rangle$ can be produced from them while the reverse is not generally true, and any new set of maximally entangled states can be constructed by applying some unitary operation to each Bell state.

It is possible for two qubits to be entangled such that a form of *quantum teleportation* can occur: through the communication of a small number of classical bits, the arbitrary probability amplitudes of a qubit can be exchanged with that of a control qubit without physical interaction, as forced by the logical requirements of the entanglement. As a resource, this is an area of furious study with particular applications in quantum error correction, and which emphasises the importance of entanglement as a resource in quantum information, as it is known that one *e*-bit or entanglement-bit with two classical bits is at least sufficient to encode one qubit.

## 2.2 Quantum gates

As classical computers perform calculations using classical logic gates, so do quantum computers use quantum logic gates in their operation. As with classical computers, it is possible to draw a graphical representation of the action of *quantum circuits*, even if the physical action is further removed than the wires and transistors one may find familiar. This section briefly discusses quantum gates for single and multiple qubits and gives some examples.

As a qubit is described by a superposition of two energy states, we can effect any transformation on it by means of a unitary operator, like those familiar in quantum mechanics. These are reversible actions, with, for example, the application of any Pauli spin matrix twice returning a qubit to its original form. This can bee seen by considering the action of the gates in Figure 2.1: here the $X$ and $Z$ gates act like the Pauli-$X$ and Pauli-$Z$ matrices respectively, and $H$ is the Hadamard gate, defined in Figure 2.2a, which by inspection restore the original qubit when applied twice. In such quantum circuits, the lines are 'wires', representing perhaps a photon with polarization level $|0\rangle$ and $|1\rangle$ travelling in space, or a stationary quantum object, such as a two-energy level particle in an ion trap, travelling in time.

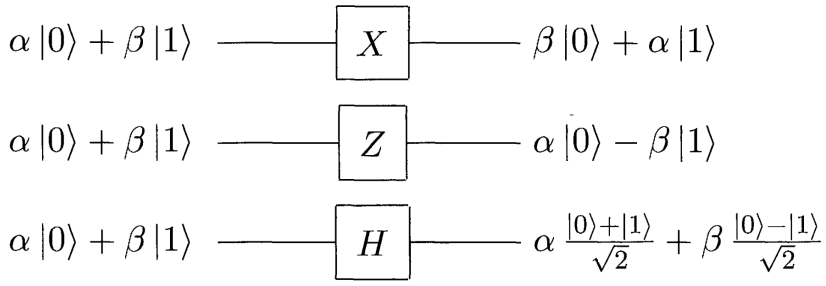
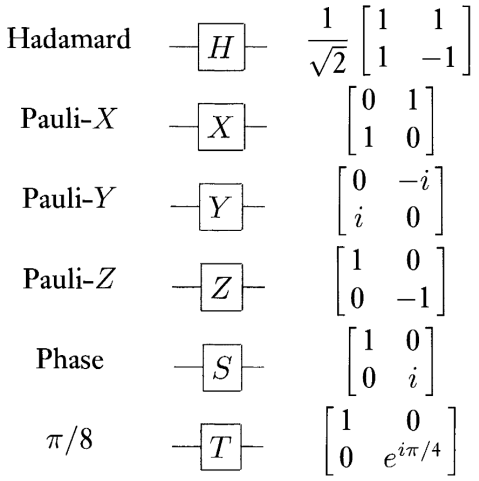
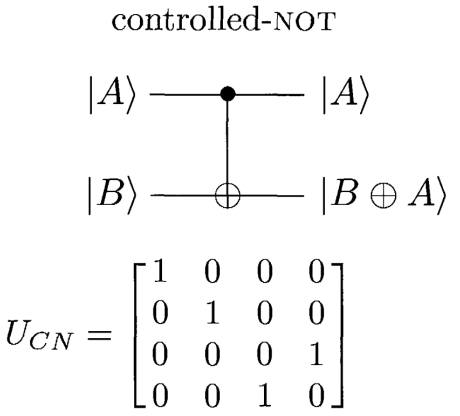



**Figure 2.1:** Action of X, Z, and Hadamard gates on one qubit and their circuit representations

Unlike many classical logic gates such as AND and XOR, any quantum logic gate will be reversible. It is possible for quantum computers to perform the action of classical computer, however, by means of universal, reversible gates such as the Toffoli gate, shown in Figure 2.3a. Classically, this is a three-bit gate which acts as a NOT gate on the third bit if both other bits are 1; it is in some sense a controlled-CNOT gate, the CNOT gate itself being a controlled-NOT gate, whose action is described in Figure 2.3a and is one of the principal quantum gates. The Toffoli gate is universal, in that all other gates can be constructed from it, as taking $c = 1$, the output value for $c$ is that of a NAND gate on $a$ and $b$, and taking $c = 0$ results in the output value of $c$ equalling $a$, i.e., the FANOUT operation, and these operations together form a universal set of gates,



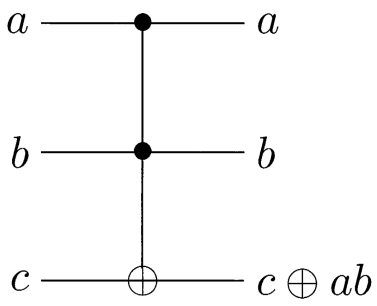

**(a)** Names, symbols, and unitary matrices for common one-qubit gates



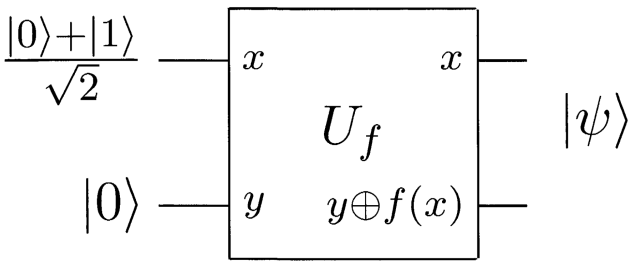

**(b)** Circuit representation and matrix form of CNOT gate

**Figure 2.2:** Circuit representations for Toffoli gate and a gate utilizing quantum parallelism

and through repeated use of Toffoli gates it is possible to simulate gates acting on more than three qubits. Since the action of a Toffoli gate can be effected by an $8 \times 8$ unitary matrix, it is possible to use a quantum computer to at least reproduce classical logic gates.




**(a)** Toffoli gate




**(b)** Gate evaluating both possible outputs simultaneously

**Figure 2.3:** Circuit representations for CNOT gate and common one-qubit gates

However, as these are quantum gates, we can include elements of quantum formalism in them to see what other sorts of behaviour are allowed. An obvious example of this is the use of qubits in superpositions to process more than one possible input at once. This

sort of *quantum parallelism* will result in a quantum gate producing a superposition of all possible output states, given a superposition of inputs received, in a single operation: Figure 2.3b illustrates an example of this. However, just as with any measurement of a superposition of states which will collapse to one constituent state with some probability, measurement of the output state will only yield one possible output. The difficulty in somehow making use of this inaccessible information is a chief difficulty in constructing useful quantum algorithms, which are discussed in the next section as we see a quantum circuit used in practice.

## 2.3 Quantum algorithms

Ultimately, the useful applications of quantum computers today are limited by the ability of researchers to invent novel ways of utilising quantum information. Those *quantum algorithms* which have been developed represent methods by which a quantum computer can be used to perform some calculation with an efficiency significantly greater than any known classical method. For example, Shor's algorithm, developed by Peter Shor in 1997, outlines a process by which the prime factors of a number can be found in polynomial time, whereas the most efficient known classical algorithm for this problem takes exponential time. This is accomplished using, amongst other methods, the *quantum fourier transform*, a quantum algorithm which can perform a Fourier transform on each entry of arbitrarily many qubits. Although relatively few quantum algorithms are known, their numbers have been steadily increasing over the past 15 years, and their potential applications are substantial enough that significant effort is spent on studying them and attempting to produce quantum computers of sufficient power to employ them.

One quantum algorithm which is discussed here is known as the *quantum search algorithm*, or *Grover's algorithm* for the researcher who discovered it in 1996 Grover96. If we have, for example, a container with $N$ compartments of which all but one are empty, to find the filled compartment we would normally have to check containers one by one

until the correct one was found, taking $O(\frac{N}{2})$ time. However, by using a many-qubit register, it is possible using the quantum search algorithm to find the filled compartment in $O(\sqrt{N})$ time. The algorithm is shown diagrammatically in Figure 2.4. Here the $n$-slash notation through a line indicates $n$ quantum wires being used in parallel, the $H^{\otimes n}$ gate represents a Hadamard gate applied to each of the $n$ incoming wires, $U_\omega$ is some known subroutine which can compare input entries according to some criterion, and the generic meter symbol with two emerging lines represents measurement with classical communication relaying the output.

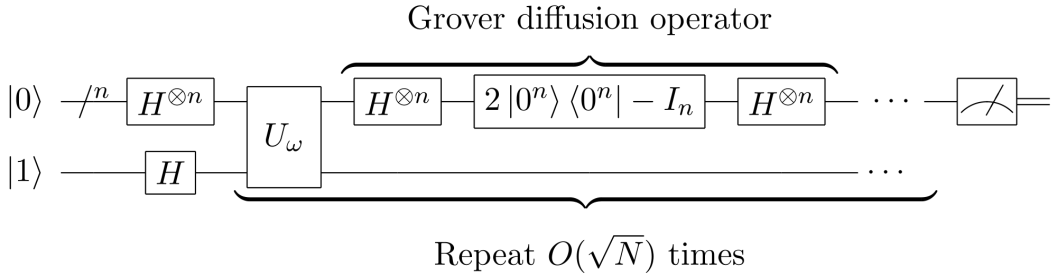



**Figure 2.4:** Circuit representation of quantum search algorithm

Qualitatively, the action of the quantum search algorithm in each iteration is to increase the probability amplitude of the target state whilst driving down that of all others. For a set of $N$ equally likely states, one of which is the target, this algorithm first applies the $U_\omega$ gate, whose action is to change the sign on the target state's probability amplitude (so that its probability remains the same) whilst leaving all others unchanged. The action of Grover's diffusion operator in Figure 2.4 is then to find the average amplitude of all states, and to 'reflect' each about that average so that states which are slightly above the average by some amount are now slightly below it by that amount. Since with a negative target state amplitude the average is now slightly lower than before, all non-target states will be reduced slightly, whereas the target state will now increase and flip sign again to roughly four times its original value. By repeating this process a number of times, the probability of observing a non-target state on measurement can be made

arbitarily low, and measuring the target state a near certainty. As higher-dimensional gates are difficult to construct, those involved in the algorithm can be constructed from a larger number of three-qubit Toffoli, ultimately requiring $O(\sqrt{N})$ operations.

## 2.4 Construction

As the fundamental component of a quantum computer, the qubit, can be formed by any of a wide range of quantum objects, there is a similarly wide range of ways of physically implementing a quantum computer. However, constructing such system remains difficult as the constraints in accuracy and stability for productive quantum computing are very high. Quantum computers require qubits to be maintained in particular states for long periods, avoiding decoherence due to noise and interaction with the environment, and for the precise manipulation of individual qubits to effect quantum gates. Depending on the physical realization used, a quantum computer may be robust or very sensitive to these, or have other properties which affect their usefulness. Here three popular implementations are outlined, as discussed by Nielsen and Chuang [3] and Preskill [4].

The *ion trap* uses single ions contained in an electromagnetic trap as qubits, with ground and excited states as $|0\rangle$ and $|1\rangle$ respectively. Quantum gates are effected via laser manipulation and measurement by photodetectors. These ion states tend to be quite stable and easy to read through laser absorption. However, interactions between qubits occur via phonons, and ions are so well isolated that these are short-lived, and the device is slow as to prepare the states requires lasers bursts on the order of microseconds. However, as of 2011, ion trap quantum computers have been constructed with up to 14 ions, and the method shows promise in scalability.

*Optical cavity QED* uses a number of atoms trapped in a high finesse optical cavity, which couple to the electric field such that only a few optical modes can exist. This coupling, through laser manipulation, allows control over interactions between atoms. Qubits are contained either on the states of the atoms or on the polarization of pho-

tons, which are stored on and interact via the atoms. However, in present models the transformations produced are not strictly unitary, with multiple-qubit gates suffering significantly as a result. However, this is likely to improve as models used to describe such behaviour are further developed, and this form of quantum computing is particularly useful in modelling many forms of quantum behaviour.

Finally, *nuclear magnetic resonance* uses very stable nuclear spin states, relative to some magnetic field, as qubits, with another pulsed magnetic field used to induce oscillations and, by suitable use, transformations on a single spin state. Interactions occur either through dipole-dipole coupling of spins states, or even chemical bonds of molecules containing those nuclei. However, as these nuclei are very high energy, so that the difference between the 'ground' and 'excited' states is very small compared to the total energy, it is necessary to use many nuclei in a thermal ensemble to perform calculations. NMR quantum computers appear to suffer from issues of scalability, without more than perhaps ten qubits realizable, although an NMR quantum computer containing seven qubits has successfully implemented a form of Shor's algorithm.

# 3 Neural Networks

A neural network is a set of interconnected processing units, or *nodes*, whose behaviour is based on the biological neuron. These strengths of these interconnections, or *weights*, can change according to some *training* rule. Neural networks thus do not execute programs as such, but are *trained* according to a set of input patterns [6].

One of the most useful applications of neural networks is in pattern recognition. Through the use of appropriate training sets, a neural network can *generalize* so that, presented with some new input pattern, if that pattern is sufficiently similar to the common information of those training sets, the network can identify it as such. For example, given a set of images of the same object but with different degree of noise, a network properly trained on these will be able to correctly classify a new image with a different distribution of noise as being one of that original object.

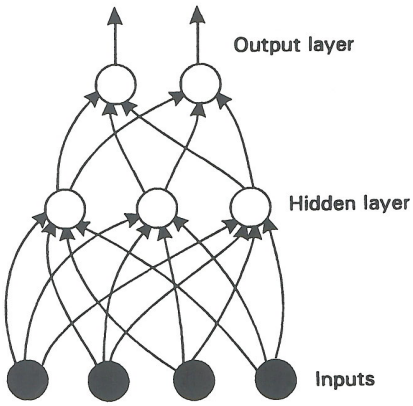



**Figure 3.1:** A simple neural network

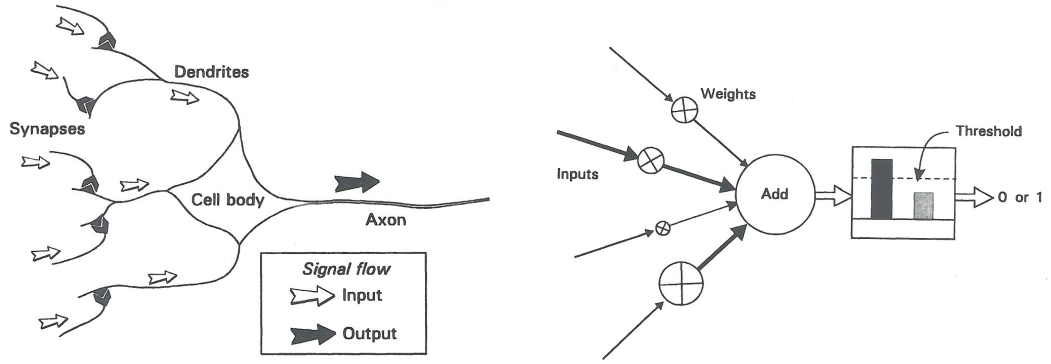
The general structure and all figures in this section are derived from Chapters 1-8 of Gurney's 1997 book, *An Introduction to Neural Networks*, with an emphasis on summarising material for a reader with no background in neural network research [6].

## 3.1 The TLU

The most basic element or neuron of a neural network is an object known as a *threshold logic unit*, or TLU. The basic properties of a TLU are described in this section.

### 3.1.1 Neurons

Biological neurons are grossly described as cells which receive electrical inputs from a large number of *synapses*, where the potential difference at each is determined by the nature of the involved neurotransmitter chemicals, as outlined in Figure 3.2a. The potentials produced effect a total potential at the cell body, or *soma*, which, if it passes above a certain threshold, will cause the cell to fire an *action potential* down the *axon*, which then branches to connect at synapses to many other neurons. Through electrical activity and changes in neurotransmitter chemicals, the potentials at each synapse can change over time. It is this behaviour which an artificial neuron attempts to reproduce, using weights to represent changes in neurotransmitter chemicals, shown in Figure 3.2b.




**(a)** Biological neuron **(b)** Artifical neuron

**Figure 3.2:** Stylized biological and artificial neurons, indicating signal flow

### 3.1.2 Activation and threshold

The total potential at the soma is the *activation*, denoted $a$, which is simply taken to be the sum of the products of different weights, $w_i$, with their associated inputs, $x_i$:

$$a = \sum_{i=1}^{n} w_i x_i, \tag{2}$$

where $n$ is the number of inputs connected to the node. If this is greater than or equal to the node's threshold, denoted $\theta$, the soma will fire a signal down th axon.

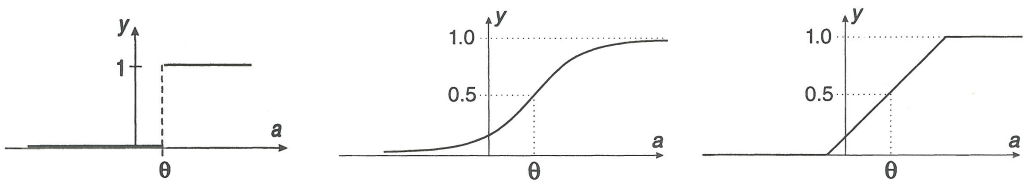
### 3.1.3 Frequency and time dependence

In the above case, the output $y$ is described by a step function, as in Figure 3.3a. In real neurons, after firing an action potential the neuron must recover over a quiescent period when no firing may occur, so that under an activation surpassing its threshold the neuron will produce a series of potential spikes of some frequency $f$. If the 'information' in the neural network is actually conveyed by this frequency, given as a fraction of some maximum frequency $f_{max}$, we can accommodate this by expressing the output as a sigmoid function, given by

$$y = \frac{1}{1 + e^{-(a-\theta)/\rho}}, \tag{3}$$

where $\rho$ is some quantity determining the steepness of the slope, as shown in Figure 3.3b. At values near to the threshold value, this is well simulated by a piecewise treatment of linear segments, as in Figure 3.3c.

As in any physical system, for a neuron whose synapses are at varying distances from the soma, due to the time required for signals to propagate we would not expect the synaptic potentials of two different synapses to sum at the soma, but for their effects to linger rather than exist instantaneously. Thus the activation is more accurately described as like the charge of a capacitor, with an activation potential being driven up (or down)




**(a)** Step function **(b)** Sigmoid function **(c)** Piecewise function

**Figure 3.3:** Threshold diagrams for different activation schemes. Here $y$ is given as a fraction of the maximum frequency, with $\theta$ taking the midopint value in each case.

under some received synaptic potential and decaying under no input. Thus the time-dependent activation $a$ is redefined by the decay equation

$$\frac{da}{dt} = -\alpha a + \beta s, \tag{4}$$

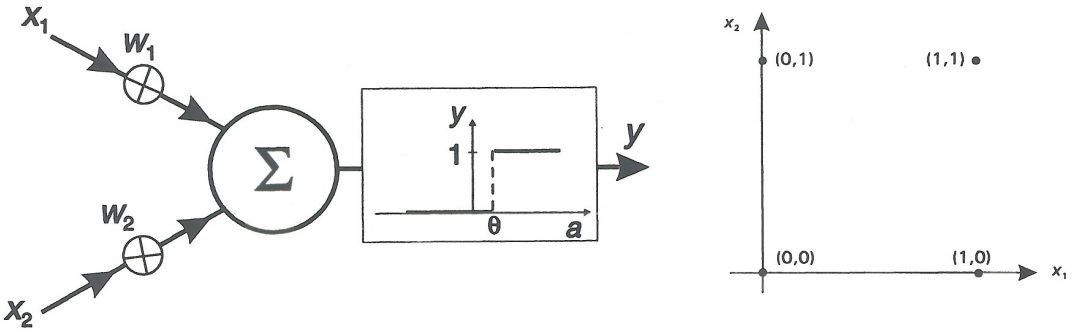
where $\alpha$ and $\beta$ are real, positive constants and $s$ is defined exactly as in Equation 2, so that the correct interpretation of $a$ is as the potential satisfying the above equation.

### 3.1.4 Pattern space and decision lines

Although the above physical considerations are necessary to simulate biological neurons accurately, much of the useful behaviour of neural networks is captured by the simpler step function approach; in particular, it is sufficient to introduce pattern space as a device for finding logical boundaries.

In the case where we have many inputs with different weights, it is useful to have a geometric boundary above which a given pattern outputs either a 1 or a 0. Consider the case of a simple two-input TLU, illustrated in 3.4a, with unit weights and inputs. The resulting pattern space is drawn from the four possible input combinations, shown in Figure 3.4b, which are to be considered as vectors. If we assume, say, a threshold of 1.5 for the TLU, only one input pattern will output a 1, namely (1, 1). Note that in this formulation, if we consider the inputs and weights to be vector components, then the

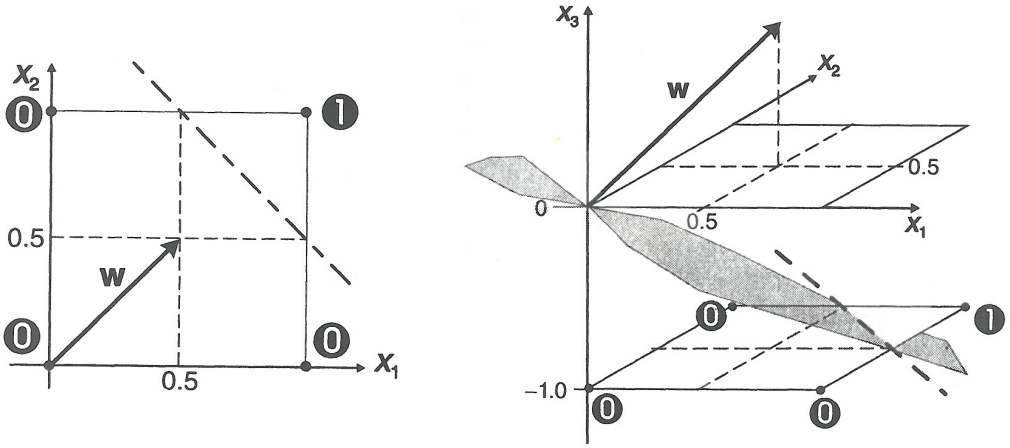
activation may be more succinctly defined as $a = \mathbf{w} \cdot \mathbf{x}$, so that $\mathbf{w} \cdot \mathbf{x} - \theta \geq 0$ outputs 1.




**(a)** Two-input TLU **(b)** 2-d pattern space

**Figure 3.4:** Diagrammatic representation of two-input TLU and its pattern space

If we now relax the condition of unit weights and inputs to allow any values, we will have an unlimited number of combinations leading to a 1 output. For two inputs, the threshold now defines a *decision line*, which is a straight line marking the logical boundary between 0 and 1 outputs, as shown in the dashed diagonal line of Figure 3.5a. Here the vector $w$ illustrates a possible activation–for instance, with unit inputs and both weights equalling 0.5–and how the decision line clearly distinguishes its logical outcome.




**(a)** Figure 3.4b with $\theta = 1.5$ decision line **(b)** 3-d pattern space with $\theta = 1.5$ decision plane

**Figure 3.5:** Examples of decision lines in two and three dimensions

The above formalism is easily extended to more inputs, corresponding to higher-dimensional pattern space. For example, the pattern space of a three-input TLU, extended simply from the two-input TLU case given, is shown in Figure 3.5b, where the $x_3 = -1$ is the 2-d pattern space of the two-input case and the shaded region is the new *decision plane*. The pattern space formalism can thus be applied to any TLU with $n$ inputs, generating an $n$-dimensional pattern space and decision hyperplane. Through the use of multiples layers of nodes or TLUs, it is possible to create a more complex, non-linear *decision surface*, which is discussed further in the next two sections.

## 3.2 Training

Given an input pattern, it is possible for the connection weights and thresholds in a neural network to adapt according to a training rule so that it becomes more useful in some application. This network adaptation is called *learning* or *training* and is briefly described in this section.

### 3.2.1 The perceptron

We begin training a TLU by the use of training sets whose outcomes we wish to yield a certain value but which the network may presently give incorrectly. With some $n$-dimensional vector $\mathbf{w}$ representing the weights of $n$ inputs to the TLU, we represent a training set by some vector of different weights, $\mathbf{v}$. If the network outputs 0 with the training set when a 1 is expected, we then evolve the TLU's weight vector to something closer to that of the training set through a vector transformation:

$$\mathbf{w}' = \mathbf{w} + \alpha\mathbf{v}, \tag{5}$$

where $0 < \alpha < 1$ is some learning rate and is discussed further in with training rules in Section 3.2.3. Similarly, if the network outputs 1 under the training set when 0 is

expected, $\mathbf{w}$ must shift away from $\mathbf{v}$, corresponding to a sign change in the equation. When the TLU successfully outputs 0 or 1 as expected under the set, no change occurs. This is most generally expressed using the difference between the target output $t$ and the output $y$, so that

$$\mathbf{w}' = \mathbf{w} + \alpha\,(t - y)\,\mathbf{v}. \tag{6}$$

This treatment forms a basic algorithm known as the *perceptron learning algorithm*, which trains the TLU with a number of training sets by repeating the process for each set until the output of each matches it target. There is a general theorem which states that for any two linearly separable classes of vectors, repeated application of this algorithm will produce a weight vector defining a TLU whose decision hyperplane separates those classes, which is expanded on in the next subsection, although this may take many repetitions or the algorithm, or *epochs*, to achieve.

It should be noted that although a TLU satisfying the above is strictly a special case of the more general perceptron, they are often used interchangeably. To avoid confusion in later reference to perceptrons in QNN papers, the perceptron is shown in Figure 3.6, in which the weight inputs are given by association units derived from a number of input elements. If these units are trivial (one-to-one with no change in input value) then the perceptron is equivalent to the TLU.

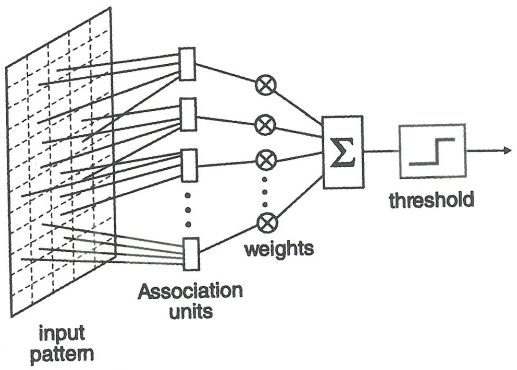


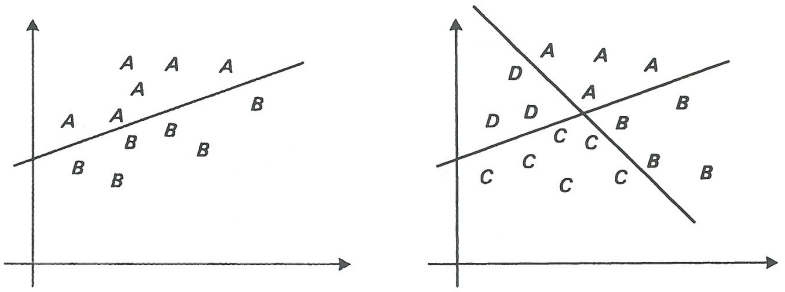

**Figure 3.6:** The perceptron

The use of training sets is one of the strengths of neural networks as they are useful in building pattern recognition. For example, a neural network containing 26 nodes could be trained to identify a Latin alphabet character represented by some area of pixels in a particular computer font, corresponding to, for example, the input pattern grid in Figure

3.6. By using training sets containing small amounts of random error, we can train the network to identify the correct pattern even in the presence of noise.

### 3.2.2 Multiple nodes

As described above, for linearly separable classes of inputs in pattern space, i.e., any two classes separable by some (decision) hyperplane, a single node is sufficient to distinguish between them, which is represented graphically in Figure 3.7a. In the case of a greater number of classes which remain linearly separable in some higher-dimensional pattern space, one extra node is required for each class, each operating and trained in parallel to the other nods in a single-layer neural network.




**(a)** Two linearly separable classes **(b)** Four linearly separable classes

**Figure 3.7:** Diagrams illustrating classification and linear separability of two-input sets in pattern space

Suppose we have four classes which are not completely separable, as in Figure 3.7b. In this case, four TLUs are no longer sufficient as for each there is no single TLU which will output 1 for a given class and 0 for all others; however, the combined classes AB and DC, or AD and CB, remain separable and can be distinguished by a pair of TLUs, so that given either another pair of TLUs can distinguish between the two constituent classes. Thus the four classes are fully separable using two layers of nodes. Note that this approach requires prior knowledge of the classes, as the pairs AC and BD are not linearly separable, so we cannot combine them arbitrarily. The next subsection outlines

a more general method of training TLUs to distinguish classes.

### 3.2.3 Learning rules

As a step toward truly autonomous training algorithms, we can remove the need for a priori knowledge of the system by using a supervised approach, where we monitor each epoch of the algorithm. One approach is to treat the difference between the weight vector at some stage and the target vector as an error to be gradually reduced. The minimum sum of errors over a training set should thus direct us toward a good general solution for the problem. This method is known as the *delta rule* as it uses the rate of change of the difference between weight vector components to find the minimum error.

Given a single TLU with $n$ inputs and weight vector $\mathbf{w}$, we can define the error $E = E(w_1, \ldots, w_{n+1})$ as the average error over all input patterns, so that

$$E = \frac{1}{N} \sum_{p=1}^{N} e^p, \tag{7}$$

where $p$ is one of the $N$ training patterns used and $e^p$ is the error attributed to it. This is proportional to the square of the difference between the pattern's activation and the target output, that is

$$e^p = \frac{1}{2} \left(t^p - a^p\right)^2, \tag{8}$$

where the activation $a$ is used rather than the output $y$ as it is as continuous function and the bracket is squared to maintain positivity of the error component, and the coefficient is added to accommodate a convention in limits.

Returning to finding the change in the weight vector, by defining it as the gradient descent of the continuous function $E(w)$ we have in one component

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i} = \alpha \left(t^p - a^p\right) x_i^p, \tag{9}$$

where we have used the expression $a_i = w_i x_i$ to write $\partial e^p / \partial w_i = -(t^p - a^p) x_i^p$ and substituted this as the expression for $\partial E / \partial w_i$ generally. This will thus be an estimate of the weight vector change, which will not succeed in every epoch, but one accurate enough to inform us how to find the minimum error efficiently. This quantity is determined and applied to the weight vector for each training set until the rate of change of the error is below some threshold, which is typically chosen through experience with the algorithm, as with the choice of the training rate $\alpha$. This outlines the delta rule, as applied to a single TLU.

Briefly, the formalism here can be extended to a single layer of many TLUs by adding a summation over that number to Equation 8, since they are independent. It is also possible to restore the output in the above formalism, which we had replaced by the continuous activation, by introducing a suitable smooth function depending on the activation. In this case the derivative of the sigmoid, $\sigma(a)$, which takes the form of a Gaussian function, is appropriate as we expect small changes in the activation to effect very small changes in the error, whereas rapid changes will correspond to the sharpest slope on the sigmoidal output function we have used. Thus for a single TLU, one of $j$ nodes in a single layer, we may rewrite Equation 9 as

$$\Delta w_{ji} = \alpha \sigma'(a_j) \left( t_j^p - y_j^p \right) x_{ji}^p. \tag{10}$$

If the learning rate is sufficiently low, the network will over a number of epochs smoothly approach the minimum error, or with limited overshooting like the Newton-Rhapson algorithm; if the rate is too high it may actually take much longer to find the minimum, and for $\alpha \geq 1$ will never obtain a solution or simply diverge, hence our earlier requirement that $0 < \alpha < 1$. The ideal learning rate typically depends on the application and is a matter of experience in using the network.

## 3.3 Multilayer Nets

Up to now the discussion of neural networks has concentrated on single nodes or layers, whereas most useful neural networks will have multiple layers as they can construct more complex decision surfaces. The added difficulty here is that the intermediate *hidden* layers themselves must be trained without knowing their target outputs for some training set. In this section a more general algorithm for training multilayer nets is found and training multilayer nets is further discussed.

### 3.3.1 Backpropagation algorithm

In principle, the nodes in a hidden layer may be trained using the delta rule as for the output nodes, but since we do not have target vectors for the hidden nodes the weight changes must be derived from those of the layers to which they output. This process is known as the *backpropagation algorithm* as the weight changes "propagate" in reverse through the network, from the output layer to hidden layers closer to the nodes receiving the input pattern. The weight changes for the hidden nodes are thus derived from the delta rule expressions: expressing the hidden layer equivalent to $t_j^p - y_j^p$ as $\delta^k$, into which the sigmoid term is absorbed, the learning rule for the $k$th node in a hidden layer becomes

$$\Delta w_{ki} = \alpha \delta^k x_{ki}^p. \tag{11}$$

These hidden nodes each affect a number of nodes in the layer below them, with the connection between them having some weight $w_{jk}$ for hidden node $k$ and output node $j$. We would then expect this weight change $\delta^k$ to be a sum of products between connected $\delta^k$'s and these weights, with each $\delta^k$ in each expression taking a similar form down to the output layer, where we instead use the appropriate expression as found for the delta rule. Thus we write

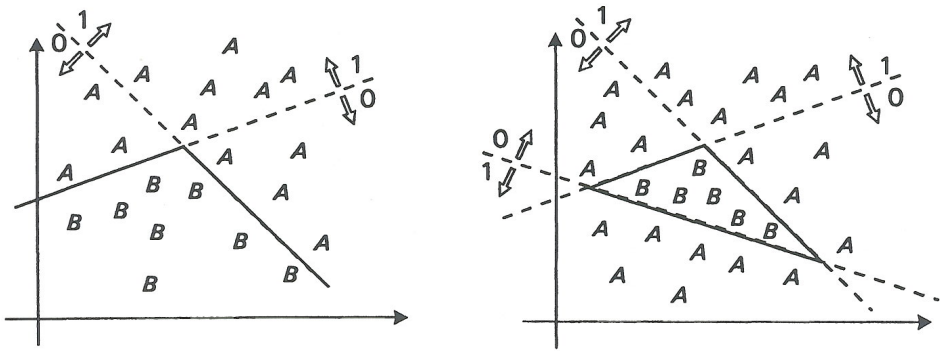
$$\delta^k = \sum_{j \in I_k} \delta^j w_{jk}, \tag{12}$$

where $I_k$ is the set of nodes receiving input from the hidden node $k$. The weight change for any single connection in this scheme is thus given by Equation 11 where

$$\delta^k = \begin{cases} \sigma'(a_k)\left(t_k^p - y_k^p\right) & \text{for output nodes} \\ \sigma'(a_k)\sum\limits_{j \in I_k} \delta^j w_{jk} & \text{for hidden nodes.} \end{cases}$$

### 3.3.2 Pattern space in multilayer nets

In Section 3.2.2 we introduced a two-layer network that could distinguish four nonlinearly separable classes of input vectors by first sorting them into two linearly separable classes. Using backpropagation, such a network can be trained to separate a nonlinear region. Figure 3.8a represents a case as before but where discrimination by one decision line cannot separate the class B. Such a decision surface again requires two hidden nodes, although only one output node is required as we have only two possible outcomes.




**(a)** Two nonlinearly separable classes **(b)** Nonlinearly separable finite region

**Figure 3.8:** Diagrams illustrating classification and separability for two and three hidden nodes

In general, one $n$-input hidden node can be trained to demark a hyperplane in $n$-dimensional pattern space, i.e., to form one decision hyperplane. Figure 3.8b illustrates a case where three hidden nodes can discriminate a class in some finite region. Adding more hidden nodes allows more complex decision surfaces to be formed, including non-convex and disconnected regions, although for our purposes the examples above should

show the ability of more complex neural networks to apply to more difficult problems.

### 3.3.3 Feature extractors

It has been noted previously that neural networks are useful at solving problems involving pattern recognition, and an example of this is given here. Suppose we have a set of training patterns with input components $x_i$ and output $y$, as in Figure 3.9 where a set of four 11-input training patterns are given with their outputs. It would be useful to determine which of the components $x_i$ are correlated or anticorrelated with the output and to what degree, as these may offer a fast way to "sort" an incoming pattern into an expected output, and a neural network can be trained to find such a measure. This can be done using a correlation measure

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | $x_{10}$ | $x_{11}$ | $y$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

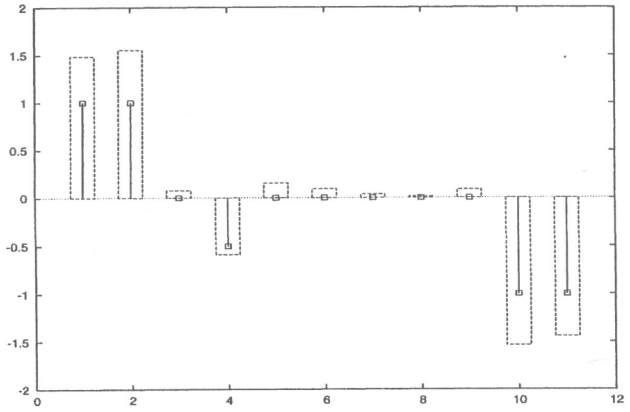



**Figure 3.9:** Set of input vectors and corresponding features generated by a trained TLU

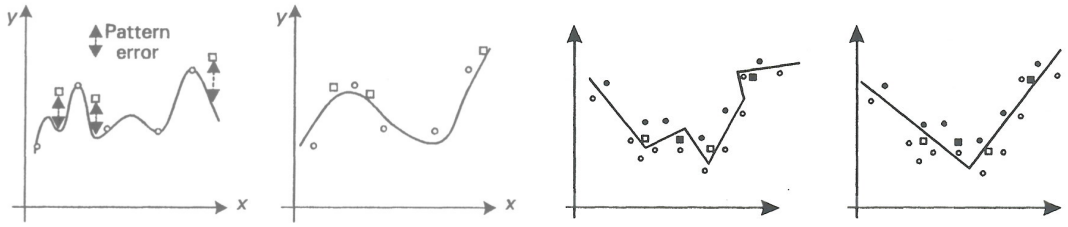
$$c_i = \frac{1}{4} \sum_p \bar{x}_i^p \bar{y}^p, \tag{13}$$

where $\bar{x}_i^p$ and $\bar{y}^p$ are the input components and output for a pattern $p$ defined as previously, but taking values of -1 or 1, rather than 0 or 1. This measure will attribute a positive value to components which are closely correlated to the output, a negative value to those anticorrelated, and for balanced output values (i.e. an equal number of 1's and 0's) will give a measure of zero for constant components. This is illustrated in Figure 3.9 where the solid lines indicate the calculated correlation measures and the dashed bars

show the values given by a single trained node, yielding values roughly proportional to the correlation measures.

### 3.3.4 Generalization and overtraining

In mathematics, it is a standard problem to find the polynomial which passes through some given points, but the *exact* solution will rarely offer the best fit for points following some pattern. In Figure 3.10a we see two graphs of training patterns, given by filled and empty circles, one where a curve has been fit exactly to each and another where a smoother curve has been used which does not pass exactly through each. By introducing some new patterns, given by filled and empty squares, which are near to other points on the graph, we see that the exact solution does a considerably poorer job at finding the pattern common to them. By adhering too closely to the original points, the generality of the solution was lost.




**(a)** Function approximation **(b)** Pattern space

**Figure 3.10:** Overtraining and generalization for different input schemes. Here circles represent the different training patterns used and squares represent test patterns.

A similar case applies to neural networks trained by a set of input patterns. If we repeat the training algorithm too many times, we may produce a decision surface which correctly separates every point of the filled and empty classes, and classifies each set correctly. However, by introducing some *test patterns* containing similar elements to the original pattern, we find that the *overtrained* case can easily misclassify them. By constraining the number of times we train the network or by limiting the number of nodes used, as in Figure 3.10b, we may be able to produce a decision surface which,

although it may misclassify some of the training patterns, nonetheless offers a more general solution which is more robust to noise and error; we say that such a net has *generalized* well. Choosing the ideal number of hidden nodes for an application and judging how long to train the network are nontrivial problems with a number of possible treatments we cannot afford to discuss here.

## 3.4 Hopfield Nets

The final part in this section introduces one of the most fundamental and useful devices in modern neural networks: Hopfield nets, a form of neural network capable of *associative memory recall*. Their basic operation and some related physical considerations are covered here.

### 3.4.1 Associative memory

Associative memory refers to the association of certain patterns to some stored data or idea. Consider the variations on the pixellated character 'T' in Figure 3.11, with the character itself and cases with random noise in the lower half or 20% noise throughout. In each case a literate human can likely guess the correct character: the human has a notion as to what pattern corresponds to 'T' versus another character and can reach this conclusion despite some noise. An example of a neural network which can perform a similar task would be one which for some input pattern we wish it to 'remember' will output the same pattern, so that if it is sufficiently general it can take the input pattern with some noise in its entries and output the original pattern.

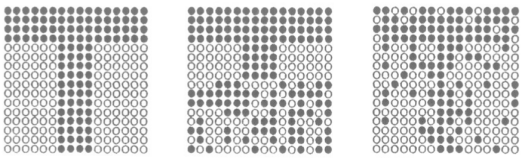


**Figure 3.11:** Examples of noise on the binary pixellated character 'T'

This sort of behaviour has a simple physical analogy which ties neural network behaviour somewhat closer to behaviour in the human brain, where electrical signals in biological neurons frequently follow paths of least potential. If we consider a particle on a potential with a number of maxima and minima, as in Figure 3.12, then it is natural to suppose that a particle somewhere on the graph will decay to some minimum, corresponding to an input pattern with some noise being classified by a neural network to the 'stored' pattern. The local minima in this case represent spurious states which can emerge through training a network too precisely, i.e., without some of the approximations used in the backpropogation algorithm, which would introduce a random element to the particle's movement as it decays sufficient to 'jump' out of local minima and find the true global minimum. The energy interpretation used here forms a useful description of memory and is the way in which we quantify the action of the Hopfield net.

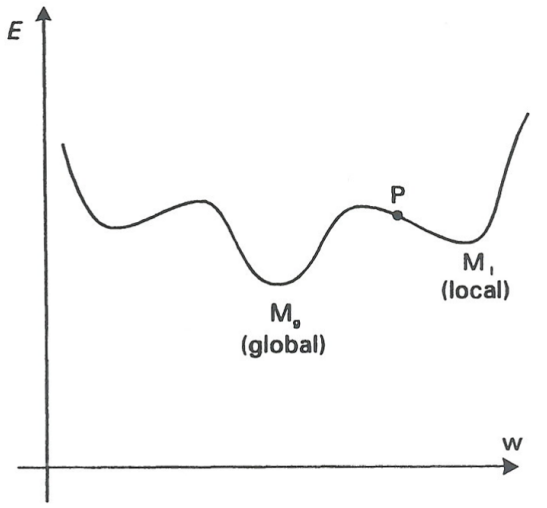



**Figure 3.12:** Potential diagram of a particle with global minimum $M_g$ and local minimum $M_l$

### 3.4.2 Hopfield net dynamics

In brief, a Hopfield net will contain nodes which act on each other, with symmetric weights between them, such that when one TLU fires, the network may transit to another state, i.e., would output that state if now checked. Once the network is in such a state that a TLU fires and the state does not change, or the network cycles through a set of states, it has 'recalled' its memory; these are the 'stored' states of the network. Figure 3.13 shows a simple three-TLU Hopfield net with some given weights between the nodes.

This sort of network is different from those encountered so far as there are no obvious layers; we might say that for any firing node the nodes to which it outputs are the next layer, but since the nodes can be reused this is not a very good description. Such networks like the Hopfield net are called *recurrent* for this reason, whereas the networks described so far which can use the backpropogation algorithm to train are *feedforward*, whereas a Hopfield net once formed is not trained as such, as only once its nodes are trained via feedforward methods can the Hopfield net take form, after a particular memory will depend on their stability.

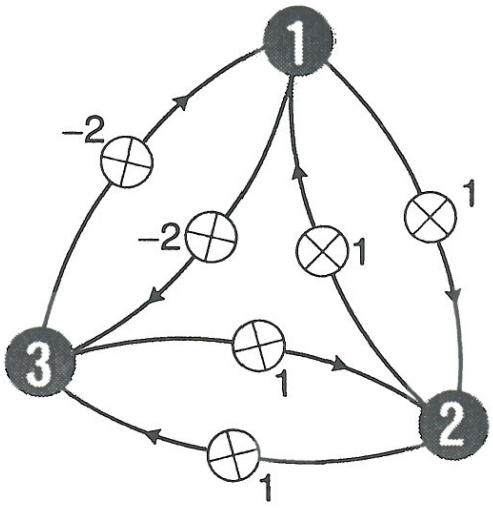



**Figure 3.13:** A three-TLU Hopfield net

| Initial state | | | | Next state (number labels) | | |
|---|---|---|---|---|---|---|
| Components | | | Number | After node 1 | After node | After node 3 |
| $x_1$ | $x_2$ | $x_3$ | label | has fired | 2 has fired | has fired |
| 0 | 0 | 0 | 0 | 4 | 2 | 1 |
| 0 | 0 | 1 | 1 | 1 | 3 | 1 |
| 0 | 1 | 0 | 2 | 6 | 2 | 3 |
| 0 | 1 | 1 | 3 | 3 | 3 | 3 |
| 1 | 0 | 0 | 4 | 4 | 6 | 4 |
| 1 | 0 | 1 | 5 | 1 | 7 | 4 |
| 1 | 1 | 0 | 6 | 6 | 6 | 6 |
| 1 | 1 | 1 | 7 | 3 | 7 | 6 |

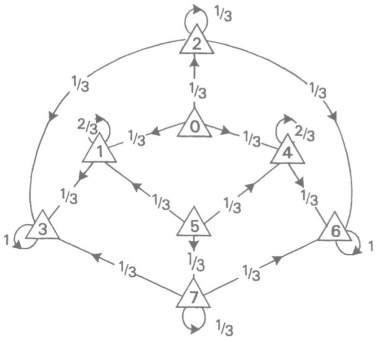
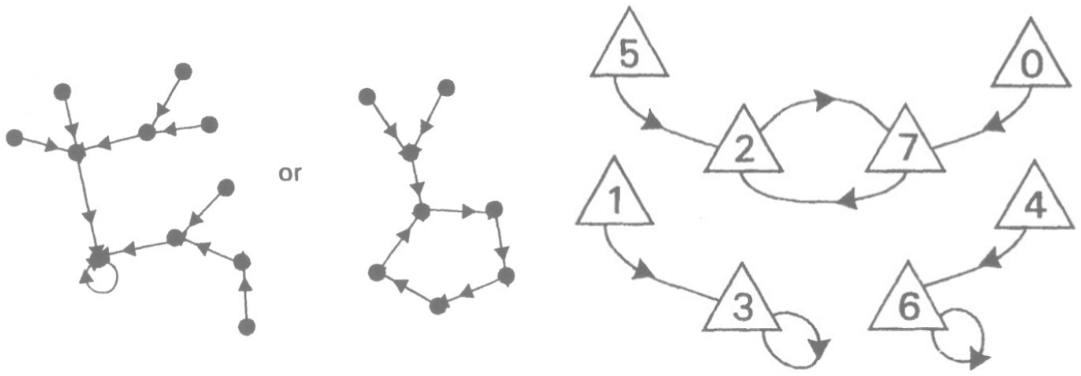



**Figure 3.14:** State transition table and diagram for the three-TLU Hopfield net with binary inputs. Here numbered triangles represent that number state, arrows indicate state transitions, and their affixed numbers represent the probability of transition given a node has fired.

For a Hopfield net with three TLUs and binary inputs ('1' or '0' only), there will be eight possible input vectors, given in the table of Figure 3.14. Suppose the network is in state 2 so that $x_2 = 1$ and the others are zero. If node 1 fires, we are evaluating the activation on it, which is a simple product sum of weights with the connected node inputs, rounded to the nearest binary value. Thus $x_2 = a = w_{12}x_2 + w_{13}x_3 = 1 \cdot 1 + (-2) \cdot 0 = 1$,

and in this case since the node was zero the others are not affected. The network is now in state 6, with probability 1/3 if firing is random. By a similar process other states can change in a manner represented in Figure 3.14, and the complete set of input states and transitions is known as the *state space* of the network. As the table and diagram illustrate, some states return to themselves under transition, so that any further firing will have no effect, and in this sense the network has recalled one of the 'memories' encoded into it by the connection weights. Such states are known as *basins of attraction* as states tend to evolve toward them, and may be singular or cyclic, as represented in Figure 3.15a, although typically cyclic states emerge as a result of *synchronous* node firing which, if all nodes update simultaneously, evolves the system deterministically. The state space structure for the three-TLU net, shown in Figure 3.15b, identifies two single-state cycles, states 3 and 6, and one multiple-state cycle, states 2 and 7 together.

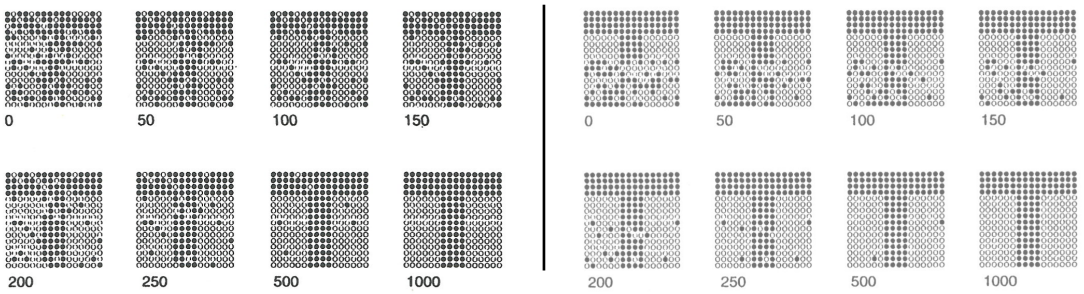



**(a)** Examples of state space behaviour **(b)** Stace space structure for three-TLU net

**Figure 3.15:** State space representations for general single-state and multiple-state cycles, and for the three-TLU Hopfield net defined above

For a network trained to handle an input pattern representing the character 'T', as in Figure 3.11, as the constituent nodes fire the state held by the nodes will gradually converge toward the stored memory. Figure 3.16a illustrates this for both cases of noise previously given, with 50% noise in the lower half and the upper half unaffected or

*clamped* or with 20% noise throughout.




**(a)** Evolution of 'T' with 20% noise **(b)** Evolution of 'T' with upper half clamped

**Figure 3.16:** Evolution of states in Figure 3.11 after number of nodes updated

It is possible to quantify these notions of stable memory states using the energy description adopted earlier, as these should correspond to lowest energy states. Suppose the nodes and connection weights of the network correspond to springs connected to some masses. Here the minimum energy of the system is approached when the masses on springs in tension approach each other, and when the masses on springs in compression move apart. If we relate these to positive and negative weights respectively, as illustrated in Figure 3.17, then we can introduce some internode energy, $e$, corresponding to the elastic potential energy of the springs. For two nodes $i$ and $j$, we define this internode energy as

$$e_{ij} = -w_{ij}x_i x_j, \tag{14}$$

such that for binary inputs the only non-zero value of $e_{ij}$ occurs when both $x_i$ and $x_j$ are unity, and $e_{ij} = -w_{ij}$. Thus for a positive weight, the state (1, 1) gives a negative internode energy and must be the lowest energy state, whereas for a negative weight the state gives a positive internode energy and the other states are lower energy.

For an arbitrary number of nodes, the total energy of the system is taken as the sum

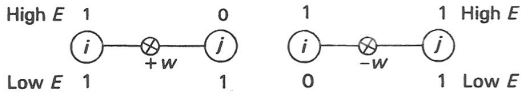



**Figure 3.17:** Weights and energies in paired nodes of Hopfield net

of all possible internode energies, i.e.,

$$E = \sum_{\text{pairs}} e_{ij} = -\frac{1}{2} \sum_{i,j} w_{ij} x_i x_j, \tag{15}$$

where the factor $1/2$ is added due to overcounting of pairs. If we expand this to separate the term for one node $k$, we have

$$E = -\frac{1}{2} \sum_{\substack{i \neq k \\ j \neq k}} w_{ij} x_i x_j - \sum_{i} w_{ki} x_k x_i, \tag{16}$$

where we have used $w_{ik} = w_{ki}$ to reduce the second term, which can now be expressed in terms of the activation of node $k$ as $x_k a^k$. Taking a change in energy corresponding to a change in the input of node $k$, we can write

$$\Delta E = -\Delta x_k a^k, \tag{17}$$

where the summation excluding $k$ is constant and does not appear. For $a_k \geq 0$ the output either increases to 1 or stays at 1, so that $\Delta x_k \geq 0$ and $\Delta E \leq 0$. Similarly, for $a_k \leq 0$ the output either decreases to 0 or stays at 0, so that $\Delta x_k \leq 0$ and $\Delta E \leq 0$. Thus for any firing node $\Delta E \leq 0$ and the energy may decrease until eventually some minimum energy state is reached where $\Delta E = 0$.

# 4 Quantum Neural Networks

In Section 2 we introduced the quantum computer as a theoretical device for processing information which, despite its technical difficulties, can utilise quantum mechanical effects to perform certain calculations with what is thought to be greater efficiency than any classical computer. In Section 3 we introduced neural networks as an older, classical model of computation which uses many processing units connected in parallel to perform some tasks presumably more efficiently than linear computers.

Presently, if there is no obvious overlap between the spheres where these very different models of computing are most efficient, then no attempt is made to find one as the final limits and applications of both forms of computing are not known even to experts in their fields. What remains an open question is whether some form of computing can be constructed utilising both a quantum formalism and a neural network topology to gain some of the advantages of both. One may imagine that in the best case if these could be combined the resulting QNN would possess advantages not yet matched by any other computational model; however, more realistically, we can expect many, if not all, of these advantages to be compromised by some fundamental disparities between both forms of computing, to the point where if a QNN can be constructed, it may offer no meaningful benefit.

It is a common misconception that superposition in quantum computers allows for massively parallel processing, which is to misunderstand the nature of quantum parallelism as something directly exploitable. Neural networks, however, are parallel processors by nature, and a hypothetical QNN would be a parallel computer even if only as a consequence of this. Whether the parallelism of neural networks can be employed fruitfully by quantum information theory is of course an outstanding question, which some of the models below attempt to address by different methods and to varying degrees. One may think that if quantum computers and neural networks are both in some

sense *probabilistic* there will exist a way to translate between them, so that the disparity may not be as profound as, say, the classical and quantum modes of computation, but most models below sacrifice some degree of quantum behaviour in forcing it into the framework of neural networks. In all respects, this review aims provide the most critical presentation possible whilst accurately representing each model.

It should be noted that, as most QNN models so far proposed are somewhat tentative, none so far make a serious attempt to reproduce major results or developments in quantum information. To a large degree this is understandable as there is a diminished motivation; for example, quantum error-correction codes are in part obviated by the inherent noise-tolerance of generalized neural networks, and QNNs are inappropriate for attempting to execute many quantum algorithms or for simulating quantum systems far removed in behaviour from neural networks. Several models of 'quantum' neural network, including some of those described below, do not use entanglement or many other quantum phenomena at all, and perhaps could be argued are closer to a 'many-universe' neural network, using only quantum language to inform their model. Nonetheless, many authors claim that, if not providing a mechanism for describing some completely new system or with an exponential speedup, QNNs may simply provide a way of performing the action of classical neural networks more efficiently.

The precise benefits or drawbacks of any union of quantum computing and neural networks will depend entirely upon the model employed and its physical realization. However, as motivation for why certain models are have been studied and as a summary of the claims made by their authors, we here review some of the claimed or suggested advantages of a quantum neural network, and the difficulties involved with obtaining them.

## 4.1 Motivation

### 4.1.1 Memory

In a neural network, memories are formed through an iterative, feedforward training scheme to produce a net which will later identify a similar pattern as that memory, as described in Section 3.4, where the number of states which can be stored is typically proportional to the number of neurons. Ventura and Martinez [22] claim that, through a combination of Grover's algorithm and a quantum algorithm of their design [21], an algorithm for quantum associative recall can be formed such that a QNN employing it will have a storage capacity which increases exponentially with the number of neurons.

### 4.1.2 Learning

In typical applications, neural networks require anywhere from dozens to many thousands of epochs to train. Menneer and Narayanan [15] claim that in simulations of a model with limited degrees of quantum behaviour, learning times in some cases were reduced by up to orders of magnitude, and in others were somewhat slower than comparable classical networks, with performance generally superior to classical networks.

### 4.1.3 Physical constraints

One of the chief difficulties in neural networks is that, due to the large number of connections required between neurons, a physical architecture resembling neurons quickly becomes overcrowded with wires as the number of neurons increases, and technology cannot yet replicate the arrangement of neurons as in, for example, the animal brain. Behrman et al. [8, 10] describe using phonon interactions to obviate the need for wires, based on a spatial array of quantum dots. In such a case, it is claimed that the *wiring problem* of complex neural networks is solved, so that arbitrarily complex nets can be constructed.

Behrman et al. also suggest [9] that the advantages of quantum time scales and dimensions would translate into neurons capable of processing on the order of $10^{10}$ bits per second and a scale of $10^{11}$ neurons per cubic millimetre, and that the energy cost and dissipation of such a system would be less than that of similar integrated systems, although this is done without discussion of the physical difficulties involved in manipulating quantum-scale objects in such conditions.

### 4.1.4 Biological models

Historically, one of the chief motivations for neural networks has been as a framework to describe the operation of the human brain. Many neural network researchers have declared interest in QNNs as they suggest some range of quantum mechanical effects may be required for this. That to accurately model brain behaviour requires quantum mechanics is known as the *quantum mind* hypothesis and has been studied by a number of researchers, such Penrose and Hameroff, and is discussed in some reviews of QNNs [7, 12]. Presently there is evidence to suggest that some proteins, including those in conditions similar to the brain, can exist in extended states of coherence which play a role in, for example, photosynthesis, but the hypothesis is otherwise not presently well substantiated. There exists division in the scientific community over the quantum mind hypothesis, which as it rarely enters into QNN models beyond as a motivation it is outside the scope of this review to discuss.

# 5 QNN Models

The following models of QNN are a broad selection those which have been published to date, including approaches which use little quantum formalism in framing neural networks and those which make extensive use of concepts in quantum information. Table 5.1 provides a summary of the physical frameworks each model uses, derived from [7].

| Model | Neuron | Connections | Transformation | Network | Dynamics |
|---|---|---|---|---|---|
| Behrman | quantum dot molecule | phonon interactions | nonlinear | spatio-temporal | Feynman path integral |
| Chrisley | classical | classical | nonlinear | multilayer | non-superpositional |
| Menneer | classical | classical or quantum | nonlinear | single-item networks in many universes | classical |
| Ventura | qubit | entanglement | - | single-item modules in many universes | unitary and non-unitary transforma-tions |

**Table 5.1:** Summary of physical analogies used in each QNN model

## 5.1 Behrman et al.

In 1996, Behrman, Neimel, Steck, and Skinner [8] proposed a QNN model which uses Feynman path integrals to provide a quantum form of the activation function in a neural network. This model was later developed [9] to use a spatial array of quantum dot molecules to improve its physical implementation. In 2000, Behrman et al. [10] discussed simulations of this model and claim to have demonstrated an improvement in complexity and power over classical neural networks by training a quantum Hopfield net. In 2002,

Behrman et al. [11] discussed further simulations which they claim demonstrated that a quantum neural network could be trained to calculate the entanglement of a quantum state. The chief details of the temporal and spatial models are covered here, leaving further discussion of application to a quantum Hopfield net or a calculation of entanglement for an extended version of this review.

### 5.1.1 Temporal model

The model initially supposes a quantum dot molecule with five dots arranged as in Figure 5.1 [8], embedded in a substrate such as GaAs, which are sufficiently close that electron tunnelling is possible between neighbouring particles. Introducing two electrons can create a doubly-degenerate ground state with electrons at opposite corners, or a dipole when both electrons occupy the same dot, and these states correspond to $S = +1$ and $S = -1$ spin states respectively. Interchanging them thus corresponds to the action of a Pauli-$z$ operator, $\sigma_z$: the intention is that such a molecule can effect the action of a quantum gate. Connections are mediated by phonon interactions in the substrate, and an external field is used to perform training; the difficulty in manipulating this field to such a precision on the timescales used in calculations is one of the chief problems with this model, which is presently assumed to be surmountable [9].

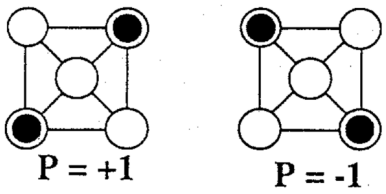



**Figure 5.1:** A quantum dot molecule as employed by the Behrman model

The model uses a mathematical isomorphism between the activation equation for a single neuron and the time evolution of a single quantum state, as both depend on the system's previous state or layer [10]. Suppose we summarise the action of an entire neural network by the operator equation

$$\varphi_{\text{output}} = \mathcal{F}_W \varphi_{\text{input}}, \tag{18}$$

where $\varphi$ represent input and output vectors and $\mathcal{F}_{_W}$ is a weight operator acting on the input vector. Similarly, the time evolution of a quantum state is expressed as

$$|\Psi(\varphi_f, T)\rangle = G_{_V}(\varphi_f, T; \varphi_0, 0)|\Psi(\varphi_0, 0)\rangle, \tag{19}$$

where $|\Psi(\varphi_t, t)\rangle$ is the state at some time $t$ and $G_{_V}$ is the Green's function for the evolution from some initial state ($t = 0$) to some final state ($t = T$). This is related to the Feynman path integral, so that over these initial and final limits,

$$|\Psi(\varphi_f, T)\rangle = \int D[x(t)] \exp\left[\frac{i}{\bar{h}} \int d\tau \left(\frac{1}{2} m\dot{\varphi}^2 - V(\varphi, t)\right)\right] |\Psi(\varphi_0, 0)\rangle \tag{20}$$

$$= \lim_{p\to\infty} \int \mathrm{d}\varphi_1 \cdots \mathrm{d}\varphi_p \exp\left[\frac{iT}{\bar{h}p} \sum_{j=0}^{p} \left(\frac{m}{2}\left(\frac{\varphi_{j+1} - \varphi_j}{T/p}\right)^2 - V(\varphi_j)\right)\right] |\Psi(x_0, 0)\rangle, \tag{21}$$

where the second line is the discretization of the first for $p$ intermediate states, so that for finite $p$ this equation will correspond to the states of $p$ quantum neurons, each separated by a period $j(T/p)$. The temporal model thus evolves a single quantum molecule or layer of molecules to obtain the next (virtual) layer, via phonon interactions and an external field, until the final layer is produced; the network is thus not a feedforward net as such, as there exists at any time only one layer, and all neurons are connected. Each possible state contributes ultimately to the output state by an amount depending on the potential energy $V(\varphi)$, which will depend on the phonons and the external field, so that through these $V(\varphi)$ can be used to train the network by adjusting the weights. Accounting for each of these factors and making them explicit, we may write Equation

21 as

$$|\Psi(T)\rangle = \lim_{p\to\infty} \int \mathrm{d}\varphi_1 \cdots \mathrm{d}\varphi_p \int \mathrm{d}\sigma_{z1} \cdots \mathrm{d}\sigma_{zp} \exp\left[\frac{iT}{\bar{h}p} \sum_{t=0}^{p} \left( K\sigma_x(t) + \Delta(t)\sigma_z(t) + \frac{m}{2}\dot{x}^2(t) + \frac{m\omega^2}{2}x^2(t) + \mu x(t)\sigma_z(t) \right)\right] |\Psi(0)\rangle, \tag{22}$$

where integration over $\sigma_z$ and $x$ relate to the states of the system and a single phonon respectively, $\sigma_z(t)$ is the instantaneous state of the dipole, $\Delta(t)$ is the external field, $\mu$ is the state-phonon coupling parameter, and $K$ is a kinetic energy-dependent tunnelling parameter [10]. Behrman et al. claim that the effect of some noise can be accounted for in training by damping one of the oscillators, provided damping is small enough that the system does not decohere on the time scale of the calculations [9]. By integrating out the degrees of freedom introduced by $x$ for each phonon mode (introduced by a general extension of the equation), for finite $p$ we can thus write

$$\begin{aligned}\langle \pm|\Psi(T/p)\rangle = {} & (\cos(KT/p)\sin(KT/p))^{p/2} \sum_{S_1=\pm1} \sum_{S_2=\pm1} \cdots \\ & \times \sum_{S_{N-1}=\pm1} \left( \tan^{-1/2} \sum_{j=0}^{p-1} S_j S_{j+1} \; i^{-1/2 \sum_{j=0}^{p-1} |S_j - S_{j+1}|} \exp\left[ -\frac{i}{\bar{h}} \sum_{j=0}^{p} \Delta_j S_j \right] \right. \\ & \left. \times \exp\left[ \sum_{j\neq f=0}^{p} S_j E_{jj'} S_{j'} \right] \right), \end{aligned} \tag{23}$$

where $S_j$ is the instantaneous spin value assumed by the system at a time slice $j$ and $E$ is an influence phase containing the effect of any phonons. This is a probability amplitude as it has taken the inner product of our final state for some known input state $|\pm\rangle$, corresponding to the $|+\rangle$ or $|-\rangle$ states with $S = \pm 1$ of the quantum dot molecule: the final output is thus the amplitude squared, subject to the net's threshold. An example of the system's behaviour is shown in Figure 5.2, where the spin after each period for

finite $p$ is visible.

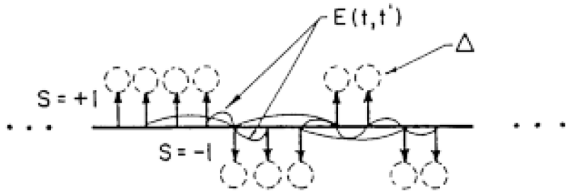
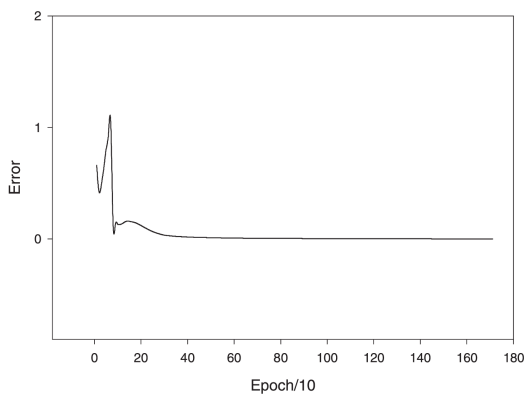
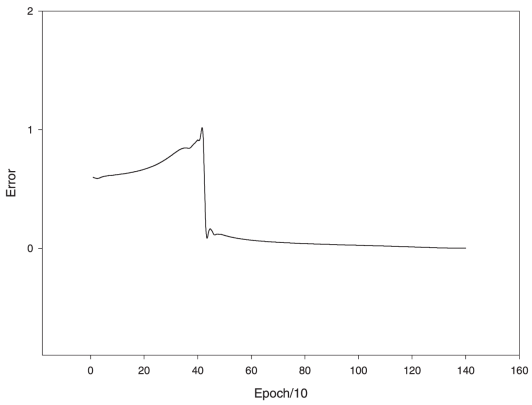



**Figure 5.2:** A single quantum dot molecule evolving over time under an external field $\Delta$, with non-local interactions $E(t, t')$

Comparing this output to the desired output, Behrman et al. [8, 10] performed simulations using gradient descent for two logic gates to test the system, with errors as shown in Figure 5.3. By choosing values for the above parameters in each case, they claim that after a number of training epochs they were able to simulate both an AND gate (Figure 5.3a) and an OR gate (Figure 5.3b) with low errors after over 1000 epochs.




**(a)** Error vs. epoch for the AND gate using five neurons and two phonon frequencies, with a final error of 0.1% after 1800 epochs




**(b)** Error vs. epoch for the OR gate using four neurons and one phonon frequency, with a final error of 0.01% after 1400 epochs

**Figure 5.3:** Error vs. epoch for two logic gates on a simulated quantum neural network [10]

### 5.1.2 Spatial model

The chief physical limitations of the temporal model are the time scale of manipulations of the external field, and possibly the overproduction of phonons through defects. In

a later paper [9], Behrman et al. briefly outlined an extension of this model to use a spatial array of quantum dot molecules which now also interact Coulombically without an external field, and where instead of a single evolving layer of neurons the molecules are arranged in an array with one layer influencing the next, rather more like the intuitive notion on a feedforward net.

The authors outline a network of six molecules, arranged as in a $3 \times 2$ grid with two nodes at each layer, from the input via hidden nodes to the output, with the output given as the average spin on the first row's output. This probability will depend on the Boltzmann factor, containing the Hamiltonian of the final quantum dot, which as in Equation 22 of the temporal model will depend on the kinetic energy and the phonon-state interactions, but now also the Coulombic interactions of the spatial array. By calculating the final output according to its equilibrium value, and using a Monte Carlo technique to find the average values for some training sets and parameters, the authors claim that through sampling the minimal error can be estimated in order to facilitate training, although this is not described explicitly, and that by this method the spatial net can be used to perform any logic gate as in the temporal model.

## 5.2 Chrisley

In 1997, as part of a broader and more speculative article, Chrisley [12] proposed a simple QNN model which uses the two-slit experimental setup to effect a non-superpositional quantum neural network. In sacrificing much of the quantum formalism available in favour of more familiar technology on a macroscopic scale, he attempted to show that some correspondence between neural networks and quantum systems exists, with the idea of inspiring other models of QNN.

Chrisley distinguishes between superpositional and non-superpositional quantum computers as those which do and do not exploit superposition in algorithms they perform, regardless of whether superpositioned states emerge in their operation. In this case

a non-superpositional quantum computer is unrestricted by the difficulty of exploiting quantum parallelism in unique ways, as in Shor's or Grover's algorithm, only in that it does not use them. In this case the apparent theoretical overlap between quantum mechanics and neural networks is further diminished; however, Chrisley's article attempts to show that some useful correspondence exists.

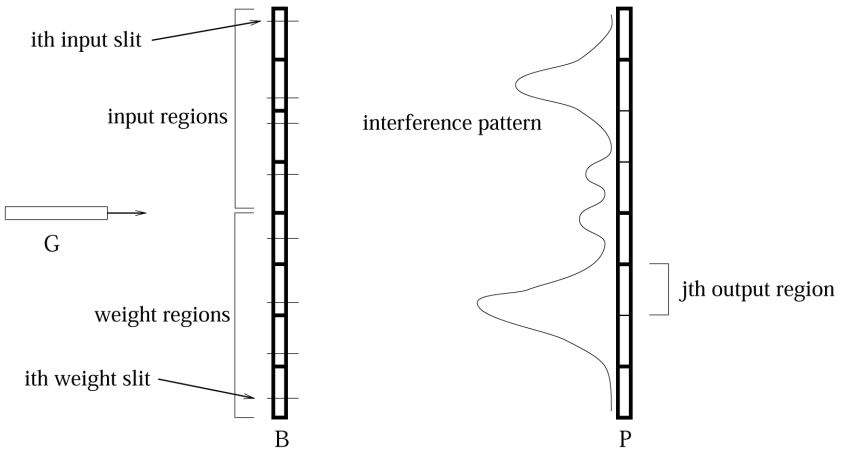



**Figure 5.4:** A feedforward quantum neural network utilising a many-slit experimental setup [12]

The model uses the apparatus of a typical two-slit experiment to create measurable regions corresponding to inputs, outputs, and weights, as illustrated in Figure 5.4, in a manner Chrisley describes as physically impractical but sufficient for his purposes. Here a particle beam fires at a barrier containing several slits, of which some are designated *inputs slits* and other *weight slits*, producing an interference pattern on a photosensitive plate which is separated into a number of *output regions*.

It is assumed some way exists of converting input data into a particular input slit configuration, and from some interference pattern to similar output data, by associating some $n$-dimensional input vector with $n$ input slits and some $m$ output regions to some $m$-dimensional output vector. For example, the $i$th coordinate of an input vector may be related to the relative position of the $i$th slit within the $i$th section of the barrier,

as in Figure 5.4, or for binary input vectors a "1" coordinate value may simply be associated with a slit being present in that region. For a binary output vector, the $m$th coordinate value may depend on a sigmoidal threshold of the average intensity within the $m$th region. Using this, and referring to the formalism in Section 3.2.3, the error of the network is defined as

$$E = \sum_i (d_i - a_i)^2 \tag{24}$$

for some desired and actual output vectors $\vec{d}$ and $\vec{a}$. For some interference pattern function $S(\vec{x}, \vec{w})$ for input vector $\vec{x}$ and weight vector $\vec{w}$, we expect $\vec{a} = O(S(\vec{x}, \vec{w}))$ where O is a mapping from the interference pattern to the output, with a similar mapping $I$ from the input vector to the input slit configuration.

From this, the network is trained first by collecting input/output pair samples $\{\vec{x}, \vec{d} = f(\vec{x})\}$ to use in training and randomizing the weight slits. For each sample, the slits are configured according to $I(\vec{x})$, with the plate cleared if necessary, and the particle beam is used until a clear interference pattern forms, generating an output according to $\vec{a} = O(S(I(\vec{x}), \vec{w}))$. From this, for some weight slit $w_j$, the change in slit weight is estimated by finding the partial derivative in the error for that slit, as in Equation 9, where $\frac{\partial E}{\partial w_j} = (\vec{d} - O(S(I(\vec{x}), \vec{w})))^2$. This is then used to find the weight change required in $w_j$ to minimise the error, as in Section 3.2.3. Following this process a number of times should find weights which minimise the error. Chrisley points to this as indicating a successful reproduction of the behaviour of a neural network.

It should be noted that, whereas this model was outlined by Chrisley to illustrate some correspondence between a quantum formalism and neural networks, rather than to produce a practical physical setup, the other models discussed in this review, some of which predate Chrisley's, have taken markedly different approaches. This model fits more in with the speculative vein of his article and is not presented with simulations under certain training parameters, but rather as an easily visualizable setup to argue

that neural network behaviour may be reproducible with some form of quantum behaviour included. Physically, the model requires some considerable off-site computation to calculate errors for backpropogation, as well as control over slit positions and possibly clearing to replacing the photosensitive plate. In the article, Chrisley continues to briefly discuss these issues and suggests a modification: by replacing both barrier and plate with photosensitive plates containing slits, with sensitive areas facing inward, and adding a second particle beam behind the plate and firing inward, then two detectable interference patterns will be produced, so that by some method the pattern on one would open slits which affected the pattern on the other, ultimately reaching some equilibrium.

## 5.3 Menneer-Narayanan

In 1995, Menneer and Narayanan [13] outlined a QNN model which uses a 'superposition' of many single-layer networks to emulate the training of a single neural network using a set of patterns, creating what they term a *quantum-inspired* neural network (QUINN), and performed brief simulations which suggested QUINNs are marginally more efficient than classical neural networks in two examined cases; this work formed part of Menneer's 1998 Ph.D. thesis [14]. In 2000, Menneer and Narayanan [15, 16] further discussed implementations of this model and identify network structures which are most effective in simulations for certain tasks.

### 5.3.1 Network architecture

The model proposed by Menneer and Narayanan [13] draws a correspondence between a quantum formalism and neural network behaviour by regarding each pattern in a neural network's training set as a possible particle state in the *many universes* interpretation of quantum mechanics. In this respect, the training of a single neural network using a set of patterns is replicated using a 'superposition' of many single-layer networks, each trained on only one pattern, which are used to generate a superpositional weight vector,

or *quantum-inspired wavefunction* (QUIWF), for the QUINN according to some weight metric depending on the training patterns used.

Menneer and Narayanan initially consider a set of three networks, each consisting of four input nodes connected to one output node, labelled a1-a4, a2-d2, and a3-d3 for networks 1, 2, and 3 respectively, each trained according to a training pattern as in Figure 5.5. Physically, the node connections themselves are represented by an object or ensemble of objects, with each weight corresponding to a different state. For some secondary particle, associated with a 1 or 0, which interacts with one of the connection objects, it enters a superposition which depends on each network. For example, if a '1' interacts with connection 1 (column 2 in Figure 5.5), it will evolve in the QUIWF to a superposition of a1 and a2, as these are the weights trained to distinguish an input of '1'. Extending this, for some input pattern, say 1111, each link will yield a superposition of weights: in this case, we obtain a1 and a2, b1, c1 and c2, and d1 and d3 for links 1, 2, 3, and 4 respectively. The only network which yields the correct output is 1, so a 'collapse' is said to occur as the other networks, corresponding to states in other universes, have been excluded as those in which the calculation occurred. For input patterns not matched exactly by one of the training patterns used, the network chosen is that which satisfies the most inputs.

| | Weight values | | | | Training input | | | |
|---|---|---|---|---|---|---|---|---|
| Net | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |
| 1 | a1 | b1 | c1 | d1 | 1 | 1 | 1 | 1 |
| 2 | a2 | b2 | c2 | d2 | 1 | 0 | 1 | 0 |
| 3 | a3 | b3 | c3 | d2 | 0 | 0 | 0 | 1 |

**Figure 5.5:** Example weight values for three single-layer networks in Menneer-Narayanan model

In a later paper, Menneer and Narayanan [15] describe a physical implementation of this model using a many-slit experimental setup which is in some ways similar to that proposed by Chrisley [12], discussed in Section 5.2. Figure 5.6 illustrates a one-layer quantum neural network which, for some input photon pattern, uses slits as the input neurons and regions on a detector screen exposed to the interference pattern as output neurons, with the superimposed waves forming the connections, as in Chrisley's

model. However, here the slit positions are not altered to effect weight changes in the network, but rather a phase-shifting layer is introduced between the slits and the detector, which through manipulation can be used to implement backpropogation, by methods described by Grover [17, 18]. By introducing more slit barriers and phase-shifting layers, an arbitrary number of hidden layers and nodes can be added.

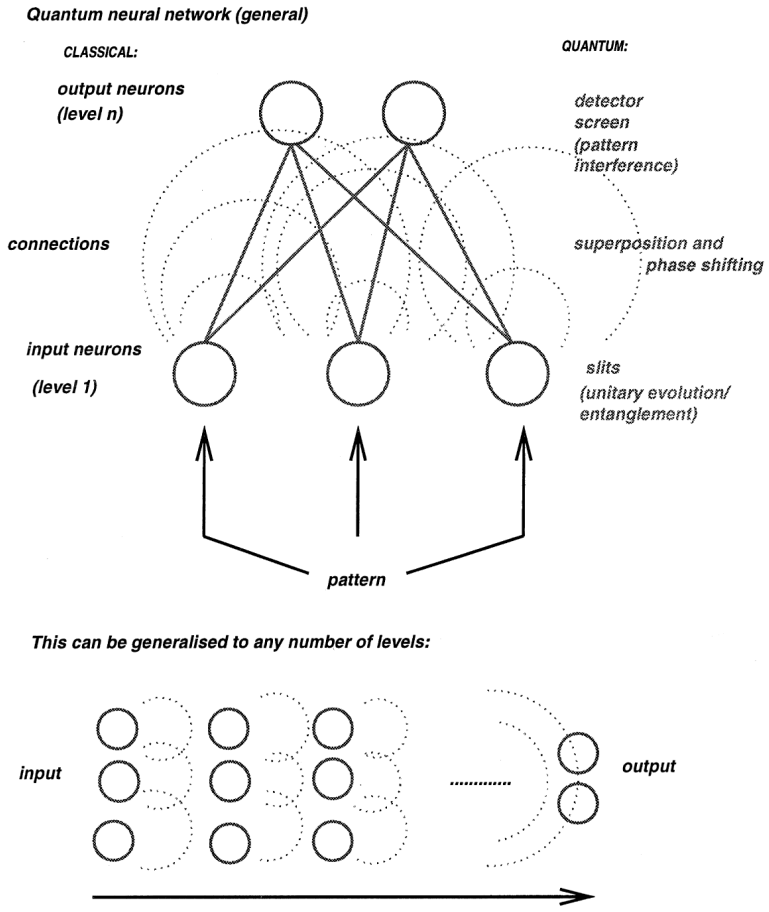



**Figure 5.6:** A quantum neural network utilising a many-slit experimental setup with phase-shifting layers [15]

The model allows for QNNs with varying degrees of quantum behaviour, as illustrated in Figure 5.7: here the unit weights for the input-hidden connections and the hidden-output connections may be quantum, i.e. a superposition of inputs as in the earlier description of the model, or classical as in typical neural networks, and similarly the activations at the hidden layer ot the output layer may be quantum, so that some superposition of outputs is produced, or classical. For any particular network structure there are then four potentially quantum components, for a total of 16 physical variations on each network. These are classified by the nature of their components, so that for a two-layer QNN the binary output $[a_1a_2a_3a_4]$, where $a_i$ is either $q$ (quantum) or $c$ (classical) for the input-hidden weights, the hidden-output weights, the hidden activation, and the output activation respectively. Thus the network $[cccc]$ represents a purely classical two-layer network, $[qcqc]$ a network whose hidden layer receives quantum inputs and with quantum activation, and $[qqqq]$ represents a purely quantum network. Figure 5.7 lists all possible variations of a network with two

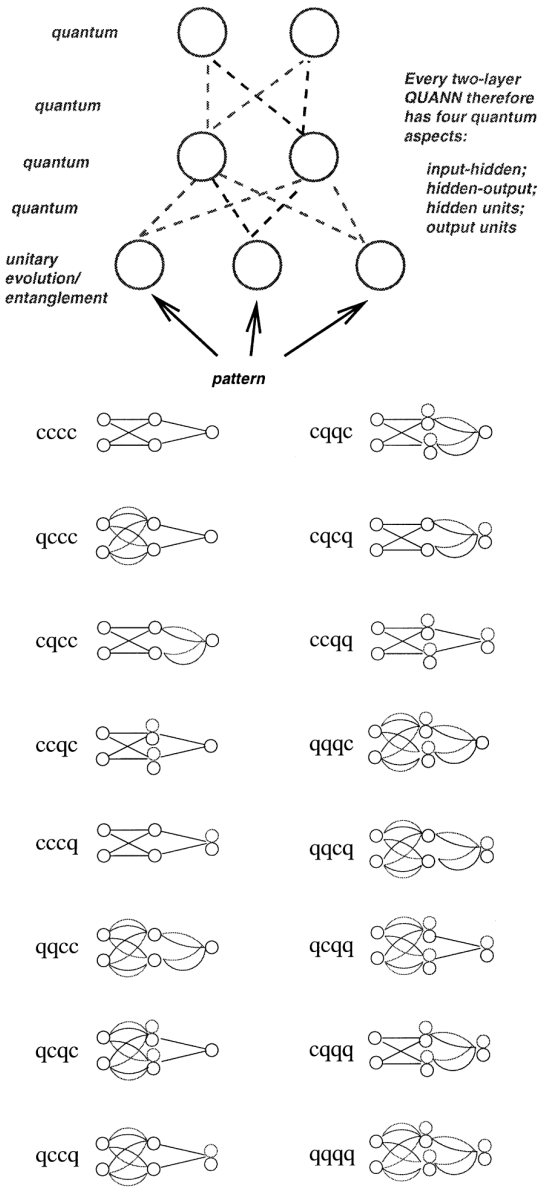
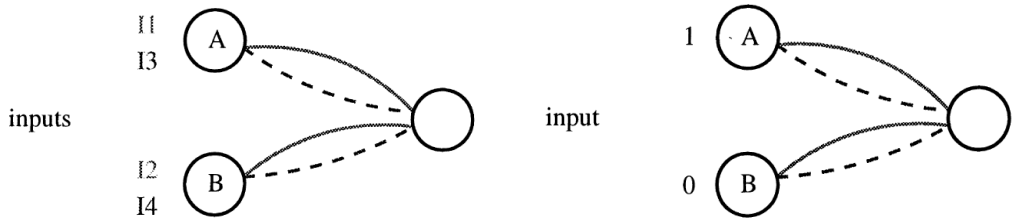



**Figure 5.7:** Illustration of quantum and classical architectures for a two-layer network. Here double-line connections represent quantum weights, double-circle nodes represent those with quantum activations, and dashed lines represent either [15].

input nodes, two hidden nodes, and one output node, where all networks are considered QNNs aside from the purely classical network [*cccc*].

For training, a set of $N$ homogeneous components is first set up, each associated with one of a set of $N$ training patterns, so that each component is trained on only these. These may be trained either as a superposition of sections (depending on their quantum nature), or as a set of networks in a superposition, either of which are evaluated by the same method. The QNN is constructed so that each link forms a superposition of the associated weight values in each component. Menneer and Narayanan claim that these weights are 'entangled' in the sense that, when given a test pattern which collapses some weight in a link superposition to that of a particular component, it forces the network to collapse to the set of weights of only that component; i.e., the QNN collapses to that component.




**(a)** Training a bit under two training patterns: I1I2 and I3I4

**(b)** Testing a bit under two training bits: 1 and 0

**Figure 5.8:** Method for training and testing QNNs in classical simulation.

Figure 5.8 illustrates the input during training or testing for a single bit in this network. If the network is trained on two patterns, say 11 (I1I2) and 10 (I3I4) as in Figure 5.8a, then comparison to a test pattern 01 will match only on the connection to node B for pattern 11, as in Figure 5.8b: this connection is strengthened by rotating the phase to increase its amplitude, whereas connections without matching bits are left the same. Collapse under testing occurs first with each component being assigned a coefficient $c$ with weight coefficients $e(c, j, i)$, for some connected output unit $j$ and input

unit $i$. The weight coefficients are defined as $e(c, j, i) = 1 - |N(i) - T(c, i)|$, where $N(i)$ is the unique pattern activation for unit $i$ and $T(c, i)$ is the training pattern activation for unit $i$ in component $c$, so that the highest weight occurs when the pattern and training activations are equal. The collapse thus occurs to the component with the greatest sum of weight coefficients under that test input, i.e., $\sum_{i,j} e(c, j, i) = \max$, the maximum sum for a component. Menneer [14] outlines other collapse schemes, but the process outlined above is used for the simulations performed on this model.

### 5.3.2 Simulations

In their first paper [13], Menneer and Narayanan used a simulated QUINN, with a comparable classical neural network trained on multiple patterns as control, to classify a number of inputs based on several properties into two categories, both using the delta rule for training with a number of different training parameters. On excluding results for nonlinearly separable input patterns which the QUINN distinguished but which the control network did not, the QUINN was found to distinguish the input patterns with similar accuracy to the control network with approximately half as many weight changes required as the total across all constituent one-layer networks. Menneer and Narayanan argue that as generalization was approximately equal in both cases, QUINNs may train faster than classical networks for this setup, and provide a solution using single-layer networks for nonlinearly separable input patterns.

In their later paper [15], Menneer and Narayanan argue that QNNs of the form they describe would have lower training times without loss of generalization because the different components forming the QNN are themselves trained independently and so will not interfere, which they also state would reduce the risk of catastrophic forgetting. They perform simulations, following the above setup, to test whether such a QNN would possess any advantages over a classical neural network, using the 16 network structures given in Figure 5.7 in such a setup that the network [*cccc*] was the most effective classical

network at the chosen task. In this case, to reduce computation time, they assumed that after each collapse the component with the highest probability was obtained, with further considerations given by Menneer in her thesis [14]. They found that performance was sensitive to the training parameters used, but that for appropriate values in certain tasks some QNN models were more efficient than the classical control network.

In two microfeatural tasks, one a modified form of the test used in their earlier paper [13] and the other a scheme for classifying edible or inedible mushrooms, the most efficient QNN was found to be [*ccqc*], i.e. a classical network with quantum hidden-layer activations. This produced very similar generalization percentages to the control network in both cases, with training times somewhat faster and slightly slower for the two tasks respectively.

In a formal language domain task, using seven formal languages to test the efficiency of the networks, the most efficient QNN was found to be [*cccq*], i.e. a classical network with quantum output-layer activations. In this case for each language used the QNN gave comparable generalization to the control network, with training times significantly longer than the control network in two languages, between one and two thirds as long in two others, and with training 20, 30, and 500 times faster in the remaining three, although in some languages these are noted to be non-significant variations.

Menneer and Narayanan claim that these simulations demonstrate advantages of QNNs over classical networks in cases where a low error tolerance is allowed, showing greater improvements over the classical network in these conditions, as in some cases where QNNs learned more slowly, in extended versions of the problems their performance surpassed that of the classical networks, which was seen to occur when the training sets were very large. They argue that if all components in a superposition can be trained in parallel, rather than in series as the simulations assume, the training time would improve and at any rate not exceed that of the classical network. Thus they argue that QNNs are at least as powerful as classical neural networks, and since they do not form decision

planes in learning patterns, they can learn training sets which classical networks cannot. They note, however, that they have not examined the performance of QNNs against self-organizing classical neural networks, and indicate this as an area of further research.

## 5.4 Ventura

In 1997, Ventura and Martinez [19] outlined a quantum analog of the perceptron where weight vectors are replaced by basis states of a wavefunction. In 1998, they described a model for *quantum associative memory* [19] using qubits as quantum neurons and argued that such a process would show a capacity for pattern storage which is exponential in the number of qubits. In 1999, using a quantum algorithm of their invention for initializing states [21], the authors developed their quantum associative memory model [22] by using the quantum search algorithm for pattern recall. This work is discussed further in articles by Ventura and others [7, 23].

### 5.4.1 Quantum perceptron

As in Section 3.2.1, the perceptron is a neuron defined at some time according to its weight vector $\mathbf{w}$ which will output 1 if the activation $a$ exceeds the threshold $\theta$ and 0 otherwise (or -1 through rescaling, as below). Ventura and Martinez [19] describe a quantum analog where $\mathbf{w}$ is replaced by a wavefunction $\Psi(\mathbf{w}, t)$ whose basis states in Hilbert space now constitute a number of unique weight vectors, so that the neuron is now defined by a superposition of all possible weight vectors with some probability amplitudes.

Ventura and Martinez give an example of this as applied to the one-input NOT gate, whose output we expect to be the input with a sign change. We would expect a neuron which simulates this behaviour to output 0 for any activation above zero, and output 1 otherwise, i.e.,

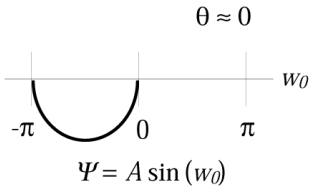



**Figure 5.9:** Solution for NOT

$\theta = 0$. By bounding the single weight element as $-\pi < w_0 < \pi$, this is satisfied by the standard infinite potential well solution in quantum mechanics,

$$\Psi(w_0) = A \sin\left(\frac{n\pi w_0}{a}\right), \tag{25}$$

where A is a normalization constant, $n$ is a positive integer, and $a$ is the box width. Figure 5.9 illustrates the solution for this, which shows that the only weights with non-zero probability amplitudes generated by the function will be negative, as is the action of a NOT gate, which for the one-dimensional wavefunction above with only one basis vector will always yield the correct weight. The authors extend this to a two-input XOR gate, whose solution we expect should have two weight vectors and will take the form

$$\Psi(w_0, w_1) = A \sin\left(\frac{n_{w_0}\pi w_0}{a}\right) \sin\left(\frac{n_{w_1}\pi w_1}{a}\right). \tag{26}$$

Figure 5.10 illustrates this, drawn at 45° for viewing, where the line $w_0 = -w_1$ has zero probability of producing a solution, resulting in ambiguity for input patterns like (-1, -1) and (1, 1) which have specific outputs and whose classification would require additional operators in the quantum neuron. Ventura and Martinez argue that because this has produced a (partial) treatment for a linearly inseparable problem like the XOR gate, which cannot be obtained on a perceptron, this treatment represents an improvement in some sense over a classical setup. They claim that though solutions as above are only cases of the square well solutions, that the use of other

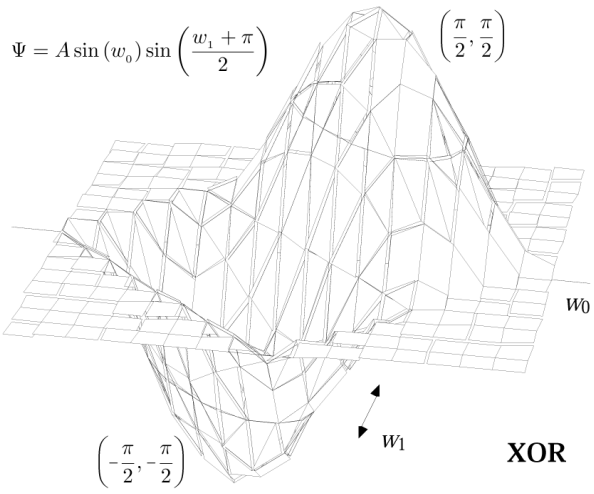



**Figure 5.10:** Contour plot of $\Psi$ for XOR

quantum systems such as harmonic oscillators may yield other ranges of solutions.

### 5.4.2 Quantum associative memory

Ventura and Martinez expanded the above model to form a basis for a quantum associative memory [20]. In this modification, the authors use a spin-1 qubit, representing a superposition of states $|0\rangle$, $|1\rangle$, and $|\star\rangle$, as a quantum neuron. This is motivated by the use of a ternary $n$-dimensional feature vector $\vec{z}$, with components 0, 1, or $\star$ where the last represents a 'neutral' value which does not affect feature extraction, with a binary $n$-dimensional input vector $\vec{x}$. These vectors 'match' only if for some $j$-th component of $\vec{x}$, the $j$-th component of $\vec{z}$ is the same or $\star$, where the number of non-$\star$ components gives the order $r$ of the feature vector, so that, for example, $(1\star\star)$ is a first-order feature vector for the input vector (101). The Hamming distance between some input vector and feactur vector defines the matching functions $m$ and $w$, such that

$$m(x,z) = \begin{cases} 1 & \text{if } z = \star \\ \rho & \text{if } x = z \\ 0 & \text{otherwise,} \end{cases} \qquad w(\vec{x},\vec{z}) = \prod_j m(x_j, z_j). \tag{27}$$

where $\rho$ is a form of pattern recall parameter which can be modified in simulations. The vectors $\vec{x}$ and $\vec{z}$ are associated with binary $s$-dimensional putput vectors $\vec{o}$ and $\vec{p}$ respectively. Ventura and Martinez thus define the state $|\psi\rangle$ of a quantum neuron as

$$|\psi\rangle = \sum_{\vec{z}\vec{p}} c_{\vec{z}\vec{p}} |\vec{z}\vec{p}\rangle \tag{28}$$

for a set of $n$ spin-1 systems and $s$ spin-1/2 systems, where $c_{\vec{z}\vec{p}}$ is the probability ampitude of state $|\vec{z}\vec{p}\rangle$. Thus the state $|\psi\rangle$ represents a superposition of $3^n \times 2^s$ possible input/output pattern combinations.

The action of learning in this model is given by two operators, $\hat{A}$ and $\hat{B}$, acting on

the state. The matrix elements of $\hat{A}$ are defined as

$$a_{\vec{z}_1\vec{p}_1,\vec{z}_2\vec{p}_2} = \begin{cases} 0 & \text{if } \vec{z}_1 \text{ does not match } \vec{x}_i \\ w(\vec{x}_i, \vec{z}_1) & \text{if } \vec{z}_1 = \vec{z}_2 \text{ and } \vec{o}_i = \vec{p}_1 = \vec{p}_2 \end{cases}, \tag{29}$$

so that the action of $\hat{A} = \sum_{i=1}^{q} \hat{A}_{\vec{x}_i\vec{o}_i}$ is to increase the probability amplitudes of eigenstates in $|\psi\rangle$ corresponding to features, given a set of $q$ training patterns. Similarly, the elements of $\hat{B}$ are defined as

$$b_{\vec{z}_1\vec{p}_1,\vec{z}_2\vec{p}_2} = \begin{cases} 0 & \text{if } \vec{z}_1 \text{ does not match } \vec{x}_i \\ -w(\vec{x}_i, \vec{z}_1) & \text{if } \vec{z}_1 = \vec{z}_2 \text{ and } \vec{o}_i \neq \vec{p}_i \\ w(\vec{x}_i, \vec{z}_1) & \text{if } \vec{z}_1 = \vec{z}_2 \text{ and } \vec{o}_i = \vec{p}_i \end{cases}. \tag{30}$$

so that the action of $\hat{B} = \sum_{i=1}^{q} \hat{B}_{\vec{x}_i\vec{o}_i}$ is to create interference between the amplitudes, increasing it for pairings whose input-output ($\vec{o}_i$) and feature-output ($\vec{p}_i$) vector entries match and decreasing it for those which do not. Training thus occurs first by preparing the neuron $|\psi\rangle$ so that the probability amplitudes $c$ give a uniform distribution, then by calculating and applying the joint operator $\hat{A}\hat{B}$ to $|\psi\rangle$. Given another input vector $\vec{y}$, which defines a third operator

$$y_{\vec{z}_1\vec{p}_1,\vec{z}_2\vec{p}_2} = \begin{cases} 0 & \text{if } \vec{z}_1 \text{ does not match } \vec{y}_i \\ w(\vec{y}_i, \vec{z}_1) & \text{if } \vec{z}_1 = \vec{z}_2 \end{cases}, \tag{31}$$

similarly to $\hat{B}$, training proceeds by applying the operator $\hat{Y}$ to the state. The state $|\psi\rangle$ is then observed and the values obtained in the spin-1/2 system are taken as the new vector components of $\vec{y}$. The process is then repeated, sans initial preparation of the state, as required to obtain generalization.

Ventura and Martinez performed simulations of this model in three standard tests (Lenses, Hayes, and LED) [20], for a range of values of $\rho$ using different numbers of

neurons, using standard datasets in an appropriate binary form over ten trials. In these cases, in each test the pattern completion accuracy and generalization accuracy was determined for both the $\hat{A}$ operator and the joint $\hat{A}\hat{B}$ operator. The results of these tests are given in Figure 5.11.

| | $\rho = 2$ | $\rho = 4$ | $\rho = 8$ | $\rho = 16$ | $\hat{A}$ |
|---|---|---|---|---|---|
| Lenses | 0.77 | 0.98 | 1.00 | 1.00 | 1.00 |
| Hayes | - | - | 0.90 | 0.91 | 0.91 |
| LED | - | - | 0.63 | 0.66 | 0.73 |

**(a)** Pattern completion accuracy

| | $\hat{A}$ | $\hat{T}$ | Other |
|---|---|---|---|
| Lenses | 0.43 | 0.68 | 0.73 |
| Hayes | 0.45 | 0.48 | 0.66 |
| LED | 0.59 | 0.56 | 0.68 |

**(b)** Generalization accuracy

**Figure 5.11:** Accuracy of Ventura-Martinez quantum associative memory model under standard tests and datasets

In pattern completion accuracy, whereas in a classical neural network the number $n$ of patterns which can be stored are typically of $O(n)$, Ventura and Martinez state they have used 5 neurons to represent 23 patterns in the Lenses test, 8 neurons to represent 83 patterns in the Hayes test, and 7 neurons to represent 200 patterns with noise in the LED test, with accuracies as given in Figure 5.11a, which they claim is indicative of an exponential storage capacity as a consequence of the exponential number of eigenstates in some number of qubits, allowing a density of accurate pattern recall which is not possible classically. In each case for pattern recall accuracy, performance was equal or slightly better using only the $\hat{A}$ operator, which in each case was near the theoretical maximum, so that the effect of the operator $\hat{B}$ is not beneficial. In the cases of generalization accuracy, the authors expect that the operator $\hat{B}$ should improve generalization by utilizing interference of features as outlined above, in so far as they do not overcompensate or overtrain on bad features. Their results, shown in Figure 5.11b, indicate that performance with $\hat{B}$, through the operator $\hat{T} = \hat{A}\hat{B}$, was better than with $\hat{A}$ alone in two out of three tests; however, in no cases did the model perform better than existing classical learning algorithms, with results represented by the column 'Other'.

The authors suggest that some new operator may perform more accurately than the joint operator $\hat{T}$ they use.

This model of quantum associative memory was subsequently expanded by Ventura and Martinez, first by proposing a quantum algorithm for initializing the amplitude distribution of a quantum system according to some training set [21], an action summarised by some operator $\hat{P}$. The authors then combine this algorithm with the quantum search algorithm [22], outlined in Section 2.3, to perform pattern recall. Patterns are first encoded onto some easily prepared ground state $|\bar{0}\rangle$ to yield $|\psi\rangle = \hat{P}|\bar{0}\rangle$, whose eigenstates represent the stored patterns on the neuron. If some $n-1$ bits of a binary $n$-dimensional pattern are known and we wish to recall it, this is done by repeated application of the quantum search algorithm to search all possible eigenstates. Ventura and Martinez thus argue that, for $2n+1$ neurons, this process can store $2^n$ patterns in $O(n2^n)$ time and recall a pattern in $O(\sqrt{2^n})$ time, which they claim is optimal.

# 6 Conclusions

The intention of this review was to present a limited number of quantum neural network models as impartially as possible, with sufficient background material to be accessible to anyone with a good understanding of quantum mechanics. Four models have been examined and their properties discussed, with results of simulations by the authors given with their interpretations. In three models simulations were performed to test their effectiveness at standard neural network problems, and in each case the authors report their models of QNNs performing better in some cases than comparable classical neural networks.

It was the original intention of this dissertation, which the scope of necessary work prevented, to produce a much more comprehensive review of QNNs and research into them. In particular, several models exist which could not be studied here, including

those described in work by Kak [26], Peruš [24], Zak and Williams [25], and Gupta and Zia [27], and in further discussions by Ventura et al. EzhovVentura00b, Ventura00b, Ezhov [30], and many others. In those models discussed, the review has aimed chiefly to summarize the authors' working and to report their claims for the model's properties, without commenting on further physical considerations which may cause difficulty for the model. A deeper study of neural networks, including the operation of *self-organizing* neural networks which through unsupervised learning can discover clusters of data independently, is a possible extension in foundational material, as one of the possible applications proposed by authors of QNNs not covered here.

None of the models discussed make extensive use of results in quantum information theory, the most obvious being entanglement. Ventura and Martinez claim [7] to use entanglement to maintain connections in their model, but this is not described or mentioned in their most extensive paper on the subject [23]. It is possible that as many of the authors cited are primarily experts in neural networks, they may have misused results in quantum theory in constructing their models, particularly on the wavefunction collapse, the no-cloning theorem, and the degree to which quantum parallelism may be exploited; the author has chosen to err on the side of caution by not attempting to find such errors, leaving that for an extension of this review after further study.

As a prognosis for the field of QNNs, this review is perhaps not promising, but strictly incomplete and inconclusive. It is hoped that this review could form a useful base for further study into quantum neural networks by familiarizing the reader with existing models, which they may build upon or use as inspiration for another model. It should be noted that most papers cited above date from 1996 to 2001, after which most authors appeared to discontinue study, or at any rate ceased to publish on the topic. Whether this is because of unpublished negative results, a loss of personal motivation, issues with funding, or any other reason is not known, but it opens a conspicuous area for possible study given an additional ten years of rapid research into quantum information theory.

## 6.1 Acknowledgements

The author would like to thank his supervisor Dr. Terry Rudolph and assessor Prof. Henris Jensen for useful conversations informing the direction of this review, Prof. Kellogg Stelle for allowing an extension of the deadline when unforseen events delayed it, and the gentleperson who enterprisingly obtained my laptop and the partial dissertation on it for leaving my favourite mouse, which allowed me complete this thesis in time.

# References

[1] R.P. Feynman, *Simulating Physics with Computers*, International Journal of Theoretical Physics **21** (6-7), pp. 467-488, 1982

[2] J.J. Hopfield, *Neural networks and physical systems with emergent collective computational abilities*, Proceedings of the National Academy of Sciences of the United States of America **79** (8), pp. 2554-2558, 1982

[3] M.A. Nielsen and I.L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2000

[4] J. Preskill, *Ph219/CS219 Lecture notes*, `http://www.theory.caltech.edu/~preskill/ph219/ph219_2011`, last visited on Sept 28, 2011

[5] L.K. Grover, *A fast quantum mechanical algorithm for database search*, Proceedings of the 28th Annual ACM Symposium on Theory of Computing, ACM, New York, pp. 212-219, 1996

[6] K. Gurney, *An Introduction to Neural Networks*, Taylor & Francis, London, 1997

[7] A. Ezhov and D. Ventura, *Quantum Neural Networks*, Future Directions for Intelligent Systems and Information Science, 2000

[8] E.C. Behrman, J. Niemel, J.E. Steck, and S.R. Skinner, *A Quantum Dot Neural Network*, Proceedings of the Fourth Workshop on Physics and Computation (PhysComp 94), p. 22, New England Complex Systems Institute, 1996

[9] E.C. Behrman, J.E. Steck, and S.R. Skinner, *A Spatial Quantum Neural Computer*, Neural Networks, pp. 874-877, 1999

[10] E.C. Behrman, L.R. Nash, J.E. Steck, V.E. Chandrashekar, S.R. Skinner, *Simulations of Quantum Neural Networks*, Information Sciences **128** (3-4), pp. 257-269, 2000

[11] E.C. Behrman, Z. Wang, J.E. Steck, V.E. Chandrashekar, C.K. Belur, S.R. Skinner, *A Quantum Neural Network Computes Entanglement*, submitted to Phys. Rev. Lett, arXiv:quant-ph/0202131v1, 2002

[12] R.L. Chrisley, *Learning in Non-superpositional Quantum Neurocomputers*, P. Pylkkänen and P. Pylkkö (Eds.), Brain, Mind and Physics, IOS Press, 1997

[13] T. Menneer and A. Narayanan, *Quantum-inspired neural networks*, Department of Computer Science, University of Exeter 1995

[14] T. Menneer, *Quantum artificial neural networks*, Ph.D. thesis, Faculty of Science, University of Exeter 1998

[15] T. Menneer and A. Narayanan, *Quantum artificial neural network architectures and components*, Information Sciences **128** (3-4), pp. 231-255 2000

[16] T. Menneer and A. Narayanan, *Quantum artificial neural networks vs. Classical artificial neural networks: Experiments in Simulation*, Proceedings of the IEEE Fourth International Conference on Computational Intelligence and Neuroscience, pp. 757-759, 2000

[17] L.K. Grover, *Quantum computers can search arbitrarily large databases by a single query*, Phys. Rev. Lett. **79**, pp. 4709-4712, 1997

[18] L.K. Grover, *Quantum mechanics helps in searching for a needle in a haystack*, Phys. Rev. Lett. **79**, pp. 325-328, 1997

[19] D. Ventura and T. Martinez, *An Artificial Neuron with Quantum Mechanical Properties*, Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, pp. 482-485, 1997

[20] D. Ventura and T. Martinez, *Quantum Associative Memory with Exponential Memory*, Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms, pp. 509-513, 1998

[21] D. Ventura and T. Martinez, *Initializing the Amplitude Distribution of a Quantum State*, Found. Phys. Lett **12** (6), pp. 547-559, 1999

[22] D. Ventura and T. Martinez, *A Quantum Associative Memory Based on Grover's Algorithm*, Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, pp. 22-27, 1999

[23] D. Ventura and T. Martinez, *Quantum associative memory*, Information Sciences **124** (1-4), pp. 273-296 2000

[24] M. Peruš, *Neuro-Quantum Parallelism in Brain-Mind and Computers*, Informatica **20**, pp. 173-183 1996

[25] M. Zak and C.P. Williams, *Quantum Neural Nets* , International Journal of Theoretical Physics **37** (2), pp. 651-684 1997

[26] S. Kak, *The Initialization Problem in Quantum Computing*, Found. Phys. **29**, pp. 267-279 1999

[27] S. Gupta and R.K.P. Zia, *Quantum Neural Networks*, Journal of Computer and System Sciences **63**, pp. 355-383 2001

[28] A.A. Ezhov, A. Nifanova, and D. Ventura, *Distributed Queries for Quantum Associative Memory*, Information Sciences **128** (3-4), pp.271–293 2000

[29] J. Howell, J. Yeazell, and D. Ventura, *Optically Simulating a Quantum Associative Memory*, Phys. Rev. A **62**, 2000

[30] A.A. Ezhov, *Pattern Recognition with Quantum Neural Networks*, Lecture Notes in Computer Science **2013**, pp. 60-71 2001