QUANTUM NEURAL NETWORKS

By

PHILIP ANDREW RICKEY

A dissertation submitted in partial fulfillment of
the requirements for the degree of

BACHELORS OF SCIENCE

WASHINGTON STATE UNIVERSITY
Department of Physics and Astronomy

MAY 2025

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of PHILIP ANDREW RICKEY find it satisfactory and recommend that it be accepted.

Nicholas Cerruti, Advisor

,

,

ACKNOWLEDGMENT

I would like to thank Dr. Nicholas Cerruti for being my guide through this literature review.

QUANTUM NEURAL NETWORKS

Abstract

by Philip Andrew Rickey, BS
Washington State University
May 2025

: Nicholas Cerruti

Quantum neural networks (QNNs) are theorized to transcend the computational and energy limitations of classical deep learning techniques by exploiting superposition and entanglement. While classical neural networks power modern search engines and recommendation systems, scaling them demands ever-greater resources, pushing classical hardware toward its limits. In this literature review, the basics of classical neural networks were looked into. Thereafter, the core principles of quantum computation were examined and compared to their respective classical counterparts. The implementations of quantum neural networks were then examined, and the future of such networks and the quantum advantage were looked into.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter One

Introduction

Artificial Neural Networks (ANNs) are computational models that mimic the way a human brain functions and processes information. ANNs are made up of interconnected layers of nodes, known as neurons, that process data from one layer to the next through weighted connections. The architecture of an ANN begins with the input layer, which is just the raw data in the form of a vector. Each sequential layer is known as a hidden layer, with each layer extracting specific features from the input layer and sending the findings to the next layer. The final layer, known as the output layer, is the end result of the network.

Such computational models have been used in a myriad of ways, such as recommendation algorithms like YouTube's (Covington, Adams, and Sargin, 2016), search engines (Mitra and Craswell, 2018), and now in new emerging technologies such as self-driving cars (Elallid et al., 2022). However, the computing power, time, energy, and resources needed to power these neural networks are also increasing as the tasks being looked at to utilize neural networks are getting more complex. With a neural network being made up of layers of nodes—the nodes being the decision-makers of a network—the more decisions that need to be made, the more intricate the layers of nodes need to be. As tasks become more complex, new technologies need to be utilized in order to increase efficiency and expand the possibilities of what can be done with a network.

Neural networks were conceived back in the early 1940s by neuroscientist Warren

McCulloch and logician Walter Pitts. As the name suggests, they took inspiration from the human brain and used mathematical functions, or neural "cells," to compute logical and/or arithmetic functions, just like the brain uses neurons for logical processing (McCulloch and Pitts, 1943). A few years later, in 1949, Donald Hebb detailed how synaptic connections can be strengthened through repeated use through excitation (Hebb, 1949).

With these ideas in place, in the late 1950s, the groundwork for neural networks was being set in place. Frank Rosenblatt, a Cornell psychologist, developed the famous Perceptron (Rosenblatt, 1958). The perceptron proposed is a simple linear activation function that "fires" a 1 if a certain condition was met. It was used in a simple two-layer network: the input layer, output layer, and adjustable weights that connect the input layer with the output layer. The input layer is the information (in vector form) that is being put into the network, the weights are how important each piece of the vector is, and the output layer is the final product by the network. The weights would be adjusted to change the output to a result closer to what was wanted, essentially "training" the network. Rosenblatt demonstrated this on an IBM 704 computer, with the network successfully being able to distinguish whether a black square appeared on the left or right side of a card. After 50–100 trials, it was consistent in determining the side (Anonymous, 1958).

Decades later, during the 1980s, one of the most important breakthroughs in neural network research emerged: the development and practical realization of the backpropagation algorithm. While researchers understood that multilayer networks could in theory approximate complex functions, they lacked an efficient method for training deep networks. During training, small errors between the network's predictions and the true outputs accumulate across layers; without a systematic method to assign credit or blame to each weight, these errors would propagate chaotically and make learning unstable.

The conceptual foundation for propagating error signals backward through a multilayer network was first articulated by Paul Werbos in 1974 (Werbos, 1974), who recognized that the chain rule of calculus could be used to efficiently compute gradients layer by layer.

This insight provided a mathematical framework for adjusting internal weights based on how much each parameter contributed to the final error.

In 1986, Rumelhart, Hinton, and Williams transformed this theoretical idea into a practical algorithm (Rumelhart, Hinton, and Williams, 1986). Their version of backpropagation systematically applies the chain rule to compute the gradient of the loss function with respect to every weight in the network. These gradients are then used in gradient descent, a calculus-based optimization method that updates each weight in the direction that most reduces the overall error.

By combining multilayer perceptrons, nonlinear activation functions, and the backpropagation algorithm, Rumelhart and colleagues provided the first scalable method for training deep networks, establishing the foundation for modern deep learning.

Following these developments, neural networks began to improve over time in their complexity, computing power, and use cases. However, as modern-day tasks become more complex, and the accuracy of neural networks becomes more imperative, computing power of such networks has to keep up. The productivity and accuracy of networks can only be as good as the data they are trained on, and usually the more training data, the better. With more training data, more computing power is needed to process the data. This is where the classical limit of neural networks occurs (Thompson et al., 2020).

With computing power beginning to limit further progress, the search for solutions has begun. As such, a promising solution is quantum computing.

In the early 1980s, Benioff constructed the first relation between the paradigms of quantum mechanics and the mechanisms of a Turing machine. The Turing machine is an idealized model of a computer. Defined as an infinite tape divided into cells (memory), each cell can hold a symbol, often a 1, 0, or blank. A head can read or write one cell at a time and only move right or left along the tape. This simple construction can essentially compute any function given enough time and computing power, given that function follows a finite, mechanical procedure. Benioff argues in his paper that you could encode the tape, head, and control state of a Turing machine into quantum states. He then described how, using a Hamiltonian, you can make the system itself evolve sequentially, just like an algorithm. By encoding each machine configuration as a basis vector in a Hilbert space, then building a Hamiltonian that links successive configurations, it is essentially a quantum algorithm. In summary, the data 0s and 1s are stored as basis states, and the operations are steps implemented using a Hamiltonian (Benioff, 1980).

David Deutsch expanded on this idea in 1985, tying quantum theory with the Church–Turing principle, laying the groundwork for the first universal quantum computer. The Church–Turing principle stated that a Turing machine can compute any function that is effectively computable. Unlike Benioff, who described explicit Hamiltonians to relate Schrödinger evolution steps to how a Turing machine computes functions, Deutsch instead defined a *gate-based* quantum computer that can simulate processes and computations, with each gate being an operation/step in an algorithm. And to this day, modern quantum computers and quantum computation theories use these *quantum gates* in order to compute functions and generate outputs. Deutsch also implemented the crucial unitary transformation U. A unitary transformation U mathematically has the property:

$$U^\dagger U = I$$

This guarantees that every step can be undone by $U^\dagger$ and the total probability stays as 1. Deutsch explains how computation is a physical process. And because in quantum physics, if a closed system is assumed, evolution is unitary, then for the computer the step-by-step processes should be unitary too. This helps to bridge quantum ideas with classical computer implementations (Deutsch, 1985).

In the early 1990s, two quantum algorithms were produced that were able to surpass classical algorithms: *Shor's algorithms*, which were used for factoring and discrete logs (Shor, 1994), as well as *Grover's search algorithm* that was used for unstructured searching (Grover, 1997).

In the mid-1990s, various methods were researched to create realized quantum computers. Among them were *trapped ions* (Monroe et al., 1995) and *superconducting qubits* (Cirac and Zoller, 1995). A *qubit* is to quantum computing as a bit is to classical computing. It stores binary information (one or zero/spin up or spin down), but unlike a classical bit it can exist in a superposition of one and zero due to the principles of quantum mechanics. This allows what is known as *quantum advantage*, meaning its being able to be one and zero simultaneously can enable a more efficient means of performing complex calculations over its classical counterparts thanks to this quantum-mechanical specific property.

With the advent of quantum computing and a newfound modern interest due to advancements in the sector, quantum computing is becoming a more realized technology for commercial use/research. The purpose of this literature review is to explore how, with this technology, the boundaries of the classical limit can be addressed and this quantum advantage can be utilized in order to improve neural networks and machine learning.

The groups that can benefit from this study are companies that employ neural networks in their products, such as YouTube and Google with their recommendation systems. Companies that rely on image processing neural networks will also benefit. Companies that use neural networks to sift through massive amounts of data for financial or medical purposes will also benefit. All of these companies are reaching the classical limit due to a lack of computational power, which is what the quantum advantage can solve.

This literature review will go over classical neural networks, their applications, and the classical limit. Thereafter, quantum neural networks will be investigated, and the future of quantum neural networks as replacements for classical ANN's will be discussed.

Chapter Two

Review Of Literature

As discussed in the introduction, the basic building block of all neural networks is the concept of the perceptron. It takes several binary inputs and produces a single binary output.

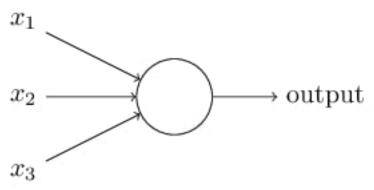

Figure 2.1 Basic perceptron detailing how 3 inputs produce a binary output. Below is the mathematical representation of the perceptron (Nielsen, 2019)

$$\text{output} = \begin{cases} 0, & \text{if } \sum_j w_j x_j \leq \text{threshold}, \\ 1, & \text{if } \sum_j w_j x_j > \text{threshold}. \end{cases} \tag{2.1}$$

These can then be put together in the classic neural network structure shown below:

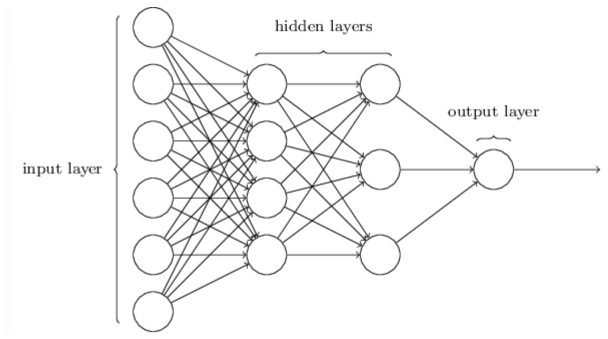

Figure 2.2 Diagram of a classical neural network. The arrows represent the inputs to other nodes in the graph (Nielsen, 2019)

For a neural network, how information is read and processed is done with the process known as *feedforwarding*. It is described as using the output from the previous layer as the input for the next. Below is the process:

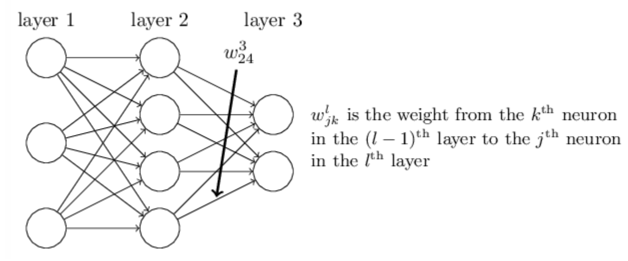
layer 1
layer 2
layer 3
w^3_{24}
w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

Figure 2.3 Diagram detailing weights and biases and the feedforward process (Nielsen, 2019)

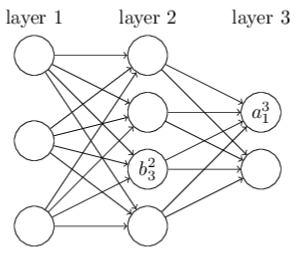

Figure 2.4 Biases and activations (Nielsen, 2019)

The information can use all of the information above to compute the output of each layer like so:

$$a^{\ell}_j = \sigma\left(\sum_k w^{\ell}_{jk}\, a^{\ell-1}_k + b^{\ell}_j\right), \tag{2.2}$$

After feedfowarding, the margin of error is calculated via a cost function:

$$C = \frac{1}{2n} \sum_x \| \, y(x) - a^L(x) \, \|^2. \tag{2.3}$$

where n is the number of training examples, x is an input, *y(x)* is the desired output, and $a^L x$ is the actual result the network produced. Since the actual output and desired output are vectors, the differences in each component is how off the output is to a desired one.

Computing the gradient allows us to see how changing the weights and biases in the network actually changes the cost function. It also gives us insight into error propagation throughout the network and how some weights in the network actually mess up the output. As such, if the error was computed, then backpropagation is used to refine the weights and biases to get closer to the desired results from the output layer. There are four equations that are used for backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

Figure 2.5 Backpropagation Equations (Nielsen, 2019)

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{2.4}$$

The term δ^L represents the error in the output layer. It is computed as the Hadamard (element-wise) product between the gradient of the cost function with respect to the activations, $\nabla_a C$, and the derivative of the activation function evaluated at the weighted input z^L.

$$\delta^l = \left((w^{l+1})^T \delta^{l+1}\right) \odot \sigma'(z^l) \tag{2.5}$$

This recursive relation defines the error for any hidden layer l. The error δ^l is obtained by propagating the next layer's error δ^{l+1} backward through the transposed weight matrix $(w^{l+1})^T$, followed by element-wise multiplication with the derivative of the activation function $\sigma'(z^l)$. This captures how each neuron influences the network's total error.

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.6}$$

The gradient of the cost function with respect to the bias b_j^l of neuron j in layer l is equal to the error term δ_j^l for that neuron. This follows directly from the chain rule and reflects the direct contribution of each bias to the total error.

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}, \delta_j^l \tag{2.7}$$

The gradient with respect to a weight w_{jk}^l connecting neuron k in layer $l-1$ to neuron j in layer l is given by the product of the activation from the previous layer, a_k^{l-1}, and the error term δ_j^l. This term quantifies how much changing the weight w_{jk}^l affects the overall cost.

Together, Equations 2.4–2.7 form the mathematical foundation of the backpropagation algorithm. They enable efficient computation of gradients across all parameters in the network, allowing optimization methods such as stochastic gradient descent (SGD) to minimize the cost function.

Above are all the components for creating a classical neural network. The information gets fed forward, then gets processed using weights and biases. The cost function is then used to see how off the network is, then the gradient of the cost function is used to backpropagate errors to tweak weights and biases in order to improve results.

As information gets more complex, more computing power is necessary. Networks these days can have hundreds of thousands of hidden layers to compute, as well as a plethora of information to be trained on. But as computing power has been slowing down, so has the progress of classical neural networks. As such, the classical limit of neural networks is starting to be realized. This is when the quantum advantage of quantum computers can come into play.

The earliest whisper of utilizing quantum computation for machine learning was in the 2013 paper *A variational eigenvalue solver on a quantum processor* (Peruzzo et al., 2014). This paper helped lay the groundwork for the techniques that would be employed in future quantum neural networks. The variational eigenvalue solver in question is a hybrid algorithm that prepares a quantum state with a specified theta (trainable parameter, like a weight) on a device, then measures the energy of a target Hamiltonian, then uses a classical optimizer (a classical computer optimization algorithm) to minimize the energy

$$E(\theta) = \langle \psi(\theta) \mid H \mid \psi(\theta) \rangle \tag{2.8}$$

where $E(\theta)$ is the energy (cost) of the quantum system, $\psi(\theta)$ is the quantum state prepared by a parameterized circuit (with trainable parameters θ), and H is the Hamiltonian of the system whose ground state is of interest. This expectation value defines the loss function used for optimization. Essentially, comparing this to classical neural networks, the trainable parameter theta is the gate rotation angle, and the layers are the repeated blocks of single-qubit rotations and gates. The learning comes from estimating the loss of energy, computing gradients (known as a parameter shift, i.e., how it evolved over time), then updating the parameters using a classical optimization function.

The seminal 2017 paper *Quantum Machine Learning* (Biamonte et al., 2017). While not providing specific algorithms or equations, it laid out the landscape for the possibilities of quantum machine learning. They described it as utilizing large Hilbert spaces, but also noted hardware and software hurdles. Shortly after, in 2018, the book *Supervised Learning with Quantum Computers* by Schuld and Petruccione was published (Schuld and Petruccione, 2018). This book was integral in introducing supervised learning in classical machine learning models as well as explaining the classical algorithms and the classical limit, which helped to motivate exploring using quantum computing. It also detailed qubits, quantum circuits, data encoding, and a plethora of other information crucial to the understanding and development of quantum machine learning models.

Another notable paper is the 2018 *Quantum Circuit Learning* paper (Mitarai et al., 2018). This turned what the 2013 paper was constructing into a more realized, general blueprint for modern machine learning frameworks. It argued that unlike the 2013 paper, which focused on how to change a Hamiltonian by minimizing the energy, it explained how classical datasets could be turned into quantum states. By taking a classical input, say x, it is possible to encode it into a quantum state using a feature map, a data-to-quantum-state-encoder

$$|x\rangle = \frac{1}{\sqrt{\sum_i |x_i|^2}} \sum_i x_i |i\rangle \tag{2.9}$$

where $|x\rangle$ is the quantum state encoding the classical input vector $x = (x_0, x_1, \ldots)$ into the amplitudes of a quantum system, and $|i\rangle$ represents the computational basis states. This form of data encoding is known as amplitude encoding, and it allows exponentially compact representation of classical data. After that, the quantum-computing version of the forward pass and backpropagation are implemented through gates. Then the classical optimization is added. A typical supervised quantum model computes the prediction as an expectation value or probability derived from a measurement. One general form is: The gradient obtained through the parameter-shift rule is given by

$$g(\theta) = \frac{f\left(\theta + \frac{\pi}{2}\right) - f\left(\theta - \frac{\pi}{2}\right)}{2}. \tag{2.10}$$

Here, $g(\theta)$ denotes the value of the gradient computed using two evaluations of the quantum circuit at shifted parameter values $\theta \pm \frac{\pi}{2}$. This equation enables gradient-based optimization on quantum hardware without numerical approximations. In summary, the exact derivatives for a quantum circuit's loss can be computed on hardware by re-running the circuit with smaller, fixed parameter shifts. The gradients could then be passed to a classical optimizer, thus achieving a true backpropagation-like training loop for the variational quantum modes introduced in 2013.

The power of quantum neural networks (Abbas et al., 2021) helped in comparing quantum neural networks to their classical counterparts. Weights and biases for a classical neural network would be turned into gates for the weights without the need for biases. The layers would just be quantum gates, manipulating the qubit through rotations just like how each layer manipulates the bits in a classical neural network. Activation functions would be feature maps and quantum measurements. For backpropagation, the backbone of how a network learns, the quantum equivalent would just be computing the gradient by using slightly shifted circuits and comparing outcomes, then using a regular optimization algorithm that updates circuit parameters using the gradients estimated. The following equation details this process: A typical supervised quantum model computes the prediction as an expectation value or probability derived from a measurement.

$$f_\theta(x) = \mathrm{Tr}\left(M\, U(\theta)\, |x\rangle\langle x|\, U^\dagger(\theta)\right) \tag{2.11}$$

The function $f_\theta(x)$ represents the expected measurement-outcome of a quantum circuit parameterized by the set of tunable angles θ. $U(\theta)$ denotes a unitary operator that depends on the variational parameters θ, $|x\rangle$ is the input quantum state encoded from classical data x, and M is a measurement operator—typically a Hermitian observable such as a Pauli operator or tensor product thereof. The trace operator $\mathrm{Tr}(\cdot)$ computes the expected value of M over the density matrix $\rho = U(\theta)|x\rangle\langle x|U^\dagger(\theta)$, which describes the evolved quantum state after applying the parameterized circuit.

Intuitively, this expression can be understood as the quantum expectation value

$$f_\theta(x) = \langle x|\, U^\dagger(\theta)\, M\, U(\theta)\, |x\rangle,$$

which gives the mean value of the measurement outcome associated with M for the circuit defined by $U(\theta)$ and input state $|x\rangle$. This formulation forms the foundation of variational quantum algorithms (VQAs) and quantum neural networks (QNNs), where the goal is to adjust θ to minimize a cost function based on $f_\theta(x)$—analogous to optimizing weights in a classical neural network.

With the current understandings of quantum neural networks, the process of applying these theories can start. The classical limits of neural networks are now being challenged by the quantum advantages that quantum computers provide. with this well-understood theory, quantum neural networks can move into the realm of reality instead of just frameworks for potential implementations.

In 2019, a quantum neural network was used to encode small patches of an image into separate qubits. Those qubits were then run through a quantum circuit, then finally measured to produce classical outputs. The outputs were then passed through a classical neural network for image classification. When this technique was tested on the MNIST handwritten digit dataset, a dataset used to test the accuracy of neural networks, it achieved a slightly higher accuracy than, as well as a faster training speed than its classical counterpart (Henderson et al., 2020).

In 2025, a method called *Nav-Q* was introduced to address the self-driving car problem of collision-free navigation. In layman's terms, how to train a car to drive without crashing and avoiding obstacles and the like. *Nav-Q* is a quantum-supported learning algorithm that uses a hybrid quantum-classical neural network for the training, but then uses classical means for the testing and deployment of the system. When the driving simulator known as *CARLA* was used to compare the classical baseline and *Nav-Q*, the latter achieved better training behavior such as lower variance between runs. It was noted that it wasn't a massive improvement, but the technology and training rival classical means, meaning it is a realized technology that can also be improved and used (Sinha, Macaluso, and Klusch, 2025).

Chapter Three

Findings

While there are advantages of using quantum neural networks over ANN's, the true problem lies in realizing a widespread use of QNN's. ANN's have been researched, developed, implemented, and improved upon for decades, leading to them being widespread and easily accessible to create. As such, they can be quickly developed and created, unlike their quantum counterparts. A QNN needs to be ran on qubit superconducting chips. Unlike ANN's that can be ran on mass-produced GPU's, the chips needed for QNN development are far more rare.

On just a cost-per-qubit basis, estimates run anywhere between $10K and $50K (PatentPC, 2025). Combining the cost of the cooling system, and qubit chips, and advanced error correction, the total cost can be estimated to be upwards of $10 million (SpinQanta, 2025). This causes a barrier of entry for other companies to enter the sector, as well as a cost-creep for bigger companies as improving quantum computers will cost a substantial amount of capital.

The environments and hardware needed to create such quantum chips also make it a challenge to mass-produce them. Creating an ideal quantum chip in and of itself is a challenge since fabricating large superconducting chips is limited by factors such as how many qubits are defective and the variability in the qubit parameters (Wu et al., 2024). After creating a suitible chip, it needs to be kept at a cool 10-20 millikelvin (Krinner et al., 2019). Such environments and careful precision in creating a chip cause limitations in mass producing them, which is the first step in realizing quantum neural networks for general use over the already mass-produced ANN.

Chapter Four

Summary and Conclusion

This literature review explored how classical neural networks came to be, and how they evolved into their modern form. Beginning with the perceptron, adding in layers and techniques such as backpropagation and cost functions, the full picture of classical neural networks was formed. With applications in recommendation systems and image recognition, neural networks began to be implemented everywhere. However, with these fields becoming more complex, the demands on processing speed, memory, and power efficiency grew. These growing struggles gave birth to the classical limit, a modern problem with explorations into possible solutions. One of those proposed solutions being quantum computing, the focal point of this paper.

Quantum computing is noted a possible solution to the classical limit. By utilizing and leveraging qubits which can exist in multiple states simultaneously, it is discovered that quantum neural networks can represent and process data far more efficiently than their classical counterparts. Henderson discovered that a quantum neural network could outclass a classical neural network on something that was seen as perfected and figured out by classical neural networks. Sinha also proved that quantum neural networks can achieve greater stability and improved optimization for self-driving cars, a field that is emerging and rapidly advancing in the modern age.

However, current limitations and costs prevent quantum neural networks from being mass-produced. The resources, time, effort, and costs make quantum chips a challenge to create as opposed to ANN's.

With all the advantages of quantum neural networks over ANN's, the limitations keep them from being more realized. Research should be conducted into optimizing and improving the qubit chip making process, as well as reducing the cost-per-qubit production. Research into more innovative and easier-to-manufacture cooling methods should be done as well in order to make having and maintaining a qubit chip more appealing. Research into these avenues shall make the mass production of quantum chips more of a possibility in the future, meaning more chips can be used to power quantum neural networks for use in more applications.

REFERENCES

Abbas, Amira et al. (2021). “The power of quantum neural networks”. In: *PubMed* 1.6, pp. 403–409. DOI: 10.1038/s43588-021-00084-1. arXiv: 2011.00027.

Anonymous (July 1958). “Electronic ’Brain’ Teaches Itself”. In: *The New York Times*. Describes the IBM 704 demo: after training on ~100 trials, it distinguishes whether a black square is on the left or right; reports 97/100 correct. URL: https://www.cs.ucf.edu/~lboloni/Teaching/CAP5636_Fall2025/homeworks/Reading%201%20-%20Perceptron-NYTimes-1958-07-13.pdf.

Benioff, Paul (May 1980). “The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines”. In: *Journal of Statistical Physics* 22.5, pp. 563–591. DOI: 10.1007/BF01011339. URL: https://link.springer.com/article/10.1007/BF01011339.

Biamonte, Jacob et al. (Sept. 2017). “Quantum machine learning”. In: *Nature* 549.7671, pp. 195–202. DOI: 10.1038/nature23474. URL: https://www.nature.com/articles/nature23474.

Cirac, J. I. and Peter Zoller (May 1995). “Quantum Computations with Cold Trapped Ions”. In: *Physical Review Letters* 74.20, pp. 4091–4094. DOI: 10.1103/PhysRevLett.74.4091. URL: https://link.aps.org/doi/10.1103/PhysRevLett.74.4091.

Covington, Paul, Jay Adams, and Emre Sargin (2016). “Deep Neural Networks for YouTube Recommendations”. In: *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys ’16)*. ACM, pp. 191–198. DOI: 10.1145/2959100.2959190.

Deutsch, David (1985). “Quantum Theory, the Church–Turing Principle and the Universal Quantum Computer”. In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818, pp. 97–117. DOI: 10.1098/rspa.1985.0070. URL: https://royalsocietypublishing.org/doi/10.1098/rspa.1985.0070.

Elallid, B. B. et al. (2022). “A Comprehensive Survey on the Application of Deep and Reinforcement Learning Approaches in Autonomous Driving”. In: *Journal of King Saud University – Computer and Information Sciences* 34.9, pp. 7366–7390. DOI: 10.1016/j.jksuci.2022.03.013.

Grover, Lov K. (1997). "Quantum Mechanics Helps in Searching for a Needle in a Haystack". In: *Physical Review Letters* 79.2, pp. 325–328. DOI: 10.1103/PhysRevLett.79.325. URL: https://link.aps.org/doi/10.1103/PhysRevLett.79.325.

Hebb, Donald O. (1949). *The Organization of Behavior: A Neuropsychological Theory*. See p. 62 for the learning rule. New York: Wiley, p. 378.

Henderson, Maxwell et al. (2020). "Quanvolutional Neural Networks: Powering Image Recognition with Quantum Circuits". In: *arXiv preprint arXiv:1904.04767*. URL: https://arxiv.org/abs/1904.04767.

Krinner, Sebastian et al. (2019). "Engineering cryogenic setups for 100-qubit scale superconducting circuit systems". In: *EPJ Quantum Technology* 6.2. DOI: 10.1140/epjqt/s40507-019-0072-0.

McCulloch, Warren S. and Walter Pitts (1943). "A Logical Calculus of the Ideas Immanent in Nervous Activity". In: *The Bulletin of Mathematical Biophysics* 5, pp. 115–133. DOI: 10.1007/BF02478259.

Mitarai, K. et al. (2018). "Quantum circuit learning". In: *Physical Review A* 98.3, p. 032309. DOI: 10.1103/PhysRevA.98.032309. URL: https://doi.org/10.1103/PhysRevA.98.032309.

Mitra, Bhaskar and Nick Craswell (2018). "An Introduction to Neural Information Retrieval". In: *Foundations and Trends in Information Retrieval* 13.1, pp. 1–126. DOI: 10.1561/1500000061.

Monroe, C. et al. (Dec. 1995). "Demonstration of a Fundamental Quantum Logic Gate". In: *Physical Review Letters* 75.25, pp. 4714–4717. DOI: 10.1103/PhysRevLett.75.4714. URL: https://link.aps.org/doi/10.1103/PhysRevLett.75.4714.

Nielsen, Michael (2019). *Neural Networks and Deep Learning*. URL: http://neuralnetworksanddeeplearning.com/.

PatentPC (Nov. 2025). "The Cost of Quantum Computing: How Expensive Is It to Run a Quantum System?" In: *PatentPC Blog*. Accessed on [insert date you accessed it]. URL: https://patentpc.com/blog/the-cost-of-quantum-computing-how-expensive-is-it-to-run-a-quantum-system-stats-inside.

Peruzzo, Alberto et al. (2014). "A variational eigenvalue solver on a photonic quantum processor". In: *Nature Communications* 5, p. 4213. DOI: 10.1038/ncomms5213. URL: https://doi.org/10.1038/ncomms5213.

Rosenblatt, Frank (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain". In: *Psychological Review* 65.6, pp. 386–408. DOI: 10.1037/h0042519. URL: https://pubmed.ncbi.nlm.nih.gov/13602029/.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). “Learning Representations by Back-Propagating Errors”. In: *Nature* 323.6088, pp. 533–536. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0.

Schuld, Maria and Francesco Petruccione (2018). *Supervised Learning with Quantum Computers*. Quantum Science and Technology. Cham: Springer, p. 287. ISBN: 978-3-319-96423-2. DOI: 10.1007/978-3-319-96424-9. URL: https://link.springer.com/book/10.1007/978-3-319-96424-9.

Shor, Peter W. (1994). “Algorithms for Quantum Computation: Discrete Logarithms and Factoring”. In: *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, pp. 124–134. DOI: 10.1109/SFCS.1994.365700. URL: https://users.cs.duke.edu/~reif/courses/randlectures/Quantum.papers/shor.factoring.pdf.

Sinha, Akash, Antonio Macaluso, and Matthias Klusch (2025). “Nav-Q: Quantum Deep Reinforcement Learning for Collision-Free Navigation of Self-Driving Cars”. In: *Quantum Machine Intelligence* 7. DOI: 10.1007/s42484-024-00226-4. URL: https://link.springer.com/article/10.1007/s42484-024-00226-4.

SpinQanta (Jan. 2025). “Superconducting Quantum Computer Prices: 2025 Update”. In: *SpinQanta Blog*. Accessed on [insert date you accessed it]. URL: https://www.spinquanta.com/news-detail/superconducting-quantum-computer-price-what-you-need-to-know20250116100111.

Thompson, Neil C. et al. (Sept. 2020). *The Computational Limits of Deep Learning*. Tech. rep. 2020, Vol. 4. MIT Initiative on the Digital Economy. URL: https://ide.mit.edu/wp-content/uploads/2020/09/RBN.Thompson.pdf.

Werbos, Paul John (Aug. 1974). “Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences”. Introduces reverse-mode (backpropagation) for training multi-layer networks. PhD thesis. Cambridge, MA: Harvard University, Committee on Applied Mathematics. URL: https://gwern.net/doc/ai/nn/1974-werbos.pdf.

Wu, Xuntao et al. (2024). “Modular Quantum Processor with an All-to-All Reconfigurable Router”. In: *Physical Review X* 14.4, p. 041030. DOI: 10.1103/PhysRevX.14.041030.