UNIVERSITY OF CALIFORNIA
SANTA CRUZ

EXPLORATORY QUANTUM MACHINE LEARNING TECHNIQUES FOR HYBRID SYSTEMS

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

DOCTOR OF PHILOSOPHY

in

PHYSICS

by

Dominic Pasquali

December 2023

The Dissertation of Dominic Pasquali
is approved:

Dr. Joshua Deutsch, Chair

Dr. Wolfgang Altmannshofer

Dr. Andrew Coats

Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Exploratory Quantum Machine Learning Techniques For Hybrid Systems

by

Dominic Pasquali

Quantum Machine Learning is a rapidly growing field in the applied and theoretical machine learning community. This work contains a select overview of classical computing, quantum computing, machine learning, quantum machine learning, and spiking neural networks. Derivations and examples are given where deemed helpful.

A number of novel techniques are proposed for integrating quantum machine learning into classical machine learning methods and paradigms, including quantum self-attention in Vision Transformers, finding the generalized unimodular matrix for a classical-quantum hybrid machine learning model, and various ways of inserting quantum machine learning into spiking neural networks. The above techniques are trained and tested on standardized and well-accepted datasets within the machine learning community. While the quantum-classical hybrid self-attention method proposed for Vision Transformers matches the performance of classical Vision Transformers, the methods of finding the generalized unimodular matrix and integrating quantum machine learning into spiking neural networks all achieve a higher accuracy faster than their comparative classical counterparts. All methods proposed by the author as analyzed and discussed in this work have been accepted to various domestic and international conferences.

In gratitude to those who came before me,

in eternal appreciation for their toil and sacrifice,

so I may have the opportunity,

to attain this achievement.

In gratitude to those who joined me,

though I faltered I did not fail,

and through their support I was able to persist and endure through difficult hardships.

To those who come after me,

so their journey,

may be easier than mine.

For the Pasquali and Cassanego families.

Acknowledgments

Though cliché, I do believe that my work, like many others, could not have been done without the immense help and support of others. These people have helped shape my work and my path during this journey.

I would like to thank the members of my Oral and Dissertation Committee, namely Dr. Joshua Deutsch, Dr. Sofia Vallecorsa, Dr. Wolfgang Altmannshofer, Dr. Bennie Lewis, and Dr. Andrew Coats for their dedication in helping me through this process.

I would like to thank UCSC Physics Admin including Amy Radovan, Cathy Murphy, Ben Miller, David Sugg, Brandon Day. I would also like to thank Stefano Profumo for his help during my Written Qualification exams. I would like to thank my 2016 UCSC Physics PhD Cohort (in no particular order: Carey Williams, John "Yianni" Tamanas, Tyler Smart, Eric Shahly, Roy Sfadia, Hanwen Qin, John Ortberg, Nora Norvell, Maverick McLanahan, Brent Limyansky, Nathan Kang, James Kakos, Carolyn Gee, Spencer Everett, Daniel Davies, Joseph "Zippy" Connell, Benjamin Lehmann, Patrick LaBarre) for many memorable times along the way, from fun memories in the "boffice" to Froth House parties (including Froth of July) and parties at Laurel House. A special thanks to Daniel, Caroyln, and Roy for their advice and support when writing this dissertation. Many thanks to those above and to Aditya Gadam and Omar Aguilar for giving me valuable feedback during my practice stages. Many thanks to members from both past and present cohorts for their valuable advice, friendship, and thoughtful

conversation. Many thanks to Peter Young for allowing me to teach with him and for helping me better understand quantum computing and its various applications.

I would like to thank the CERN Quantum Team including Sofia Vallecorsa, Michele Grossi, Oriel Kiss, Francesco Di Marcantonio, Vasilis Bellis, Saverio Monaco, Massimiliano Incudini, Jorge "Gorka" Juan Martinez de Lejarza Samper, Alice Barthe, Giulio Crognaletti, Carla Rieger, Yu Hong, and Su Yeon Chang (and the Tradizione Boys!) for many late night games of playing briscola and talking physics. A special thanks to Sofia and Michele for always being open to chat, being open to new ideas, continuously lending a helping hand, and for always setting the ideal example of what it means to lead. Many thanks to Kristina Gunne and Anastasiia Lazuka for their immense help and patience.

A special thanks for Sofia for allowing me to join the team and for Josh for taking me on when you did - I am more grateful than words can ever express.

I would like to thank the University of California Observatories (UCO) and the UCO Lab for Adaptive Optics including Daren Dillon, Renate "Reni" Kupke, Don Gavel, and Claire Max, all of whom supported my work there and generously gave their time and wisdom during and after my time at the lab.

Many others have helped me along this long route as well, including Yuri Griko, Anthony Kelly, Barbara Neuhauser, Randy Harris, Afshin Khan, and the institutions and members of LM ATC, NASA Ames, LBNL, Sandia National Labs, OLA, SI, University of Oregon, UC Berkeley, UC Davis, SFSU, and UC Santa Cruz. Many thanks to the 2009 SI Executive Council and administration. I would like to thank my UC Davis

cohort, with a special thanks to Ben Godfrey, Jeff Johansen, Chadwick Rainbolt, Nima Maghoul, and Jesus Rives.

I attribute a surprising amount of my progress in academics to my training established in athletics, including but not limited to my coaches in soccer, baseball, track and field, Olympic lifting, strength and conditioning, and the numerous other scholars, athletes, staff, and parents from AYSO, PPSL, BYBA, OLA, SI, WCAL, CCS, and Heavy Athletics who partook and shared with me on our athletic journeys.

Many thanks to Paul Cassanego and Alex Young for their sage wisdom and thoughtful advice in challenging and difficult times.

I would like to thank Joe Stefani for many hours of long conversation, debugging ideas and thoughts, and giving many comments and suggestions for this dissertation and its works, all of which have been immensely helpful.

I'd like to thank my parents, Rolando and Mary Pasquali for their endless and enduring support, no matter the highs or lows.

I would like to give a very special thank you to my partner, Eliana Stefani, for all of her relentless support in all aspects of this journey, including her endless help stretching from my classes, to my Oral Qual, defense, and this dissertation.

I'm in forever gratitude to my grandparents for immigrating to the US in search for a better life and laying the foundation, habits, and for passing on cultural traditions for who I am as a person.

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of the University of California Santa Cruz's

products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

The text of this dissertation include reprints of the following previously published material as found in [16, 17, 20, 21]. The co-authors listed in these publications directed and supervised the research which forms the basis for the dissertation.

As said by Pierce Giffin - Bazinga!

Part I

Introduction

Chapter 1

Introduction

In future's dawn, machines may share our stage,
With whispered codes, the silent tongues engage.
A dance of thought twixt silicon and flesh,
Where lines of difference start to enmesh.

Their gleaming minds, a mirror to our own,
In starlit bytes, the seeds of thought are sown.
Where human minds may falter, tire, or miss,
The steel-borne scribes persist in tireless bliss.

Yet not without a cost, this pact is signed,
As hands of flesh to ghostly gears resigned.
What work is left for human hearts to do,

When metallic minds our tasks pursue?

Yet in this storm, a beacon may yet gleam,
Of hope that human-AI is but a dream,
Where each complements the other's strength and skill,
Guided by a shared, harmonious will.

A dance not of replacement, but of blend,
Where human and machine as one ascend.
A future bright, if navigated well,
In shared endeavor, we shall surely dwell.

So let us tread with foresight, heart, and grace,
To welcome AI in the human space.
For in this union, each may find its place,
A symphony of minds in cosmic race.

ChatGPT [22]

Artificial intelligence is rapidly attaining a stronger hold in every aspect of society, from medical diagnosis [23–25] to generating images from captions [26]. As evidenced by the above ChatGPT output (though a small example), even modern day AI systems can start achieving impressive feats.

At the same time quantum computing and quantum machine learning is quickly

gaining interest with hopes of leveraging quantum effects to a large and ever growing range of applications [27]. However such applications of quantum computing and quantum machine learning are only as applicable as techniques to apply them. Therefore it is incumbent for science to develop novel methods as well to explore where and when such methods are and are not preferred. It is this development, from novel idea to initial testing (i.e. early TRL development [28]) during which the study of this dissertation achieves.

Select topics in classical, quantum, and spiking neural networks are presented in Part II and lay the groundwork to understand the novel techniques presented in Part III.

Part III proposes new hybrid classical-quantum and hybrid spiking-quantum machine learning methods, trains them on standard datasets, and benchmarks them on architecture and learned parameter equivalent classical models. While there are some hints that quantum advantage may be present in the proposed studies, much more in-depth studies of all techniques are needed to fully understand if such proposed quantum methods yield a broad advantage over classical networks. However such rigorous testing is left for when there is more time after this dissertation. Until then, these techniques are outlined herein.

Final thoughts are presented in Part IV.

It is the hope that this work, its cited works, and the many citations listed encourage readers to continue to develop these ideas and find broad applications for them in the ever expanding world of hybrid quantum machine learning methods. If

such desire is too lofty of a goal, then it is the secondary dream that contained in here lays more groundwork and sparks new ideas in the field from which this field can continue to be built; after all, quantum machine learning and all its variations are still in its early stages.

Part II

Foundations

Chapter 2

Quantum Computing

2.1 Quantum Motivation From Classical Computing

Modern classical computing is quickly approaching the limits of Moore's law, which describes that the number of transistors in an integrated circuit (IC) doubles about every two years. This "law" is important because the number of transistors is associated with the processing speed and memory capacity of the implemented hardware [29]. However if electrons are taken to be the smallest quantum computing transistor elements, then the quantum limit on Moore's law (might) be reached by the year 2036 [30].

Therefore there is the need to find an alternative computing method to move beyond Moore's law, and quantum computation is one such proposed method. The idea of using quantum phenomena on hardware for computation was independently proposed by Manin [31] (1980) and Feynman [32] (1982), and Feynman perhaps phrased it best

by saying that he was "not happy with all the analyses that go with just the classical theory, because nature isn't classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy. Thank you [32]."

2.2 A Reminder Of Classical Computing

It is helpful to better understand the comparisons between classical and quantum computing. In classical computing logical values will be individual discrete bits, 0 or 1. These 0 or 1 values can be fed into logic gates (e.g. Figure 2.1, Figure 2.4, Figure 2.3, and Figure 2.2). The use of NAND or NOR gates form a universal gate set (i.e. the NAND gate alone can be used in different combinations to reproduce the functions of all other logic gates; the NOR gate alone can do the same) [33].

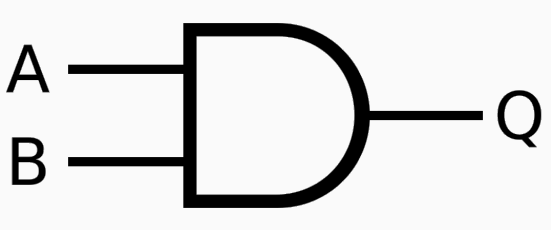


Figure 2.1: Classical AND Logical Gate [1]

2.3 Select Elements Of Quantum Computing

In contrast to classical computing with bits, quantum computing leverages quantum mechanics by allowing for these 0 or 1 classical states to be in a superposition

AND Logic Gate Truth Table		
Inputs		Out
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.1: Truth Table For AND Logic Gate

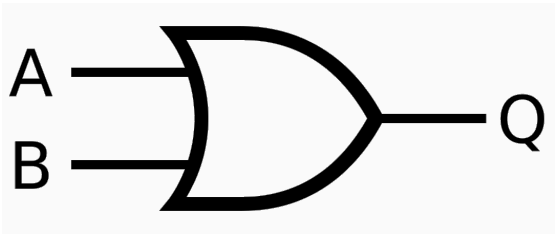


Figure 2.2: Classical OR Logical Gate [2]

OR Logic Gate Truth Table		
Inputs		Out
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

Table 2.2: Truth Table For OR Logic Gate

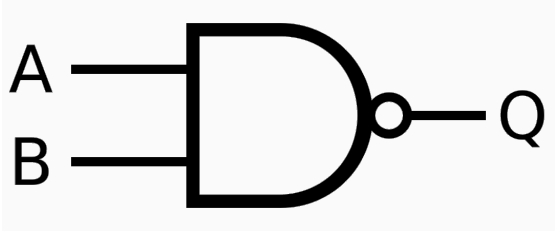


Figure 2.3: Classical NAND Logical Gate [3]

NAND Logic Gate Truth Table		
Inputs		Out
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

Table 2.3: Truth Table For NAND Logic Gate

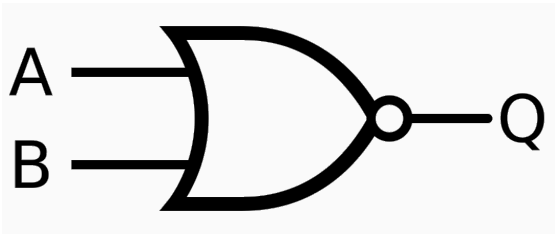


Figure 2.4: Classical NOR Logical Gate [4]

NOR Logic Gate Truth Table		
Inputs		Out
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

Table 2.4: Truth Table For NOR Logic Gate

of quantum states, i.e. there can exist a state consisting of a combination of both a "0" state and a "1" state. The square of the amplitudes is associated with the probability of that state [34, 35].

Let the "0" state (also known as the spin up state) be given by Equation 2.1 and let the "1" state (also known as the spin down state) be given by Equation 2.2.

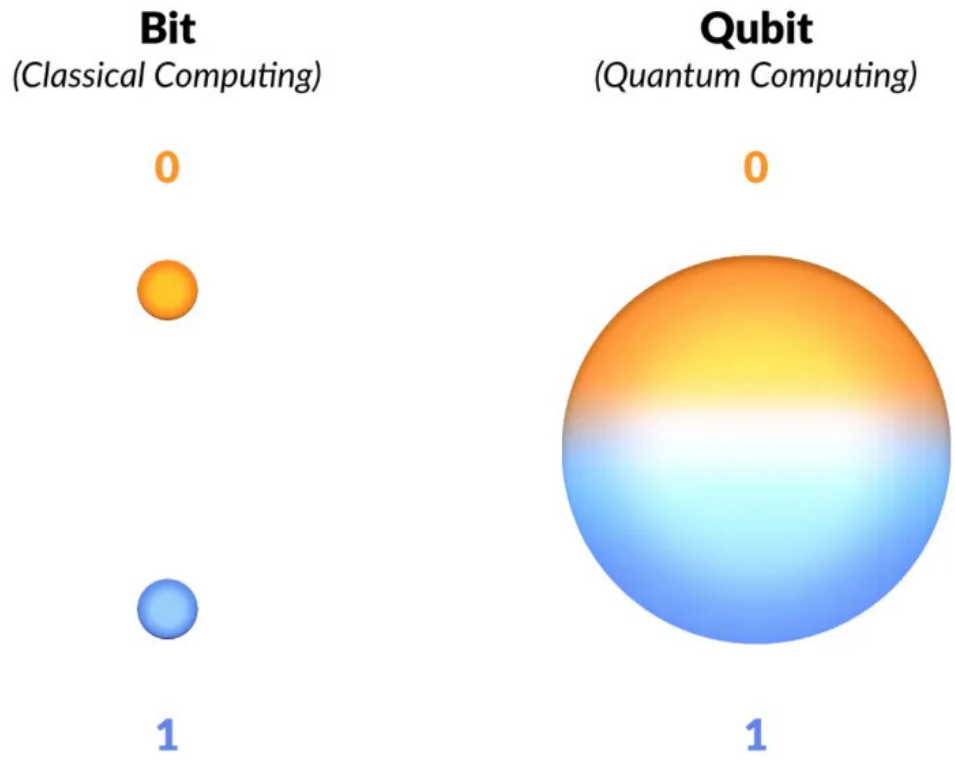


Figure 2.5: Comparing classical bits versus qubits [5]

$$0 \text{ state} := |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2.1}$$

$$1 \text{ state} := |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.2}$$

More generally the states can be written as a superposition by letting $|0\rangle$ have an amplitude of a and $|1\rangle$ have an amplitude of b, where a and b are complex. Let this superposition be called $|\psi\rangle$ which can be seen in Equation 2.3.

$$|\psi\rangle = a|0\rangle + b|1\rangle = a\begin{bmatrix} 1 \\ \\ 0 \end{bmatrix} + b\begin{bmatrix} 0 \\ \\ 1 \end{bmatrix} = \begin{bmatrix} a \\ \\ b \end{bmatrix} \tag{2.3}$$

The complex conjugate transpose (also known as "dagger" or adjoint) of Equation 2.3 is seen in Equation 2.4.

$$\langle\psi| = |\psi\rangle^{*\top} = |\psi\rangle^{\dagger} = \begin{bmatrix} a^* & b^* \end{bmatrix} \tag{2.4}$$

Recalling the requirement that the square of a and b must sum to unity due to conservation of total probability (as seen in Equation 2.5), a and b can be transformed into spherical coordinates which results in $|\psi\rangle$ (Equation 2.6) being plotted on the Bloch sphere as seen in Figure 2.6 where $0 \leq \theta \leq \pi$ and $0 \leq \phi \leq 2\pi$.

$$|a|^2 + |b|^2 = 1 \tag{2.5}$$

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + \sin\frac{\theta}{2}e^{i\phi}|1\rangle \tag{2.6}$$

The vector $|\psi\rangle$ can be rotated about to point at any location on the Bloch sphere. The Pauli matrices as seen in Equation 2.7, Equation 2.8, and Equation 2.9 allow for the rotations about the x, y, and z axes to be defined by Equation 2.10, Equation 2.11, Equation 2.12, respectively. Therefore any point on the Bloch sphere can be described as a combination of these rotations (i.e. Equation 2.10, Equation 2.11, and Equation 2.12).

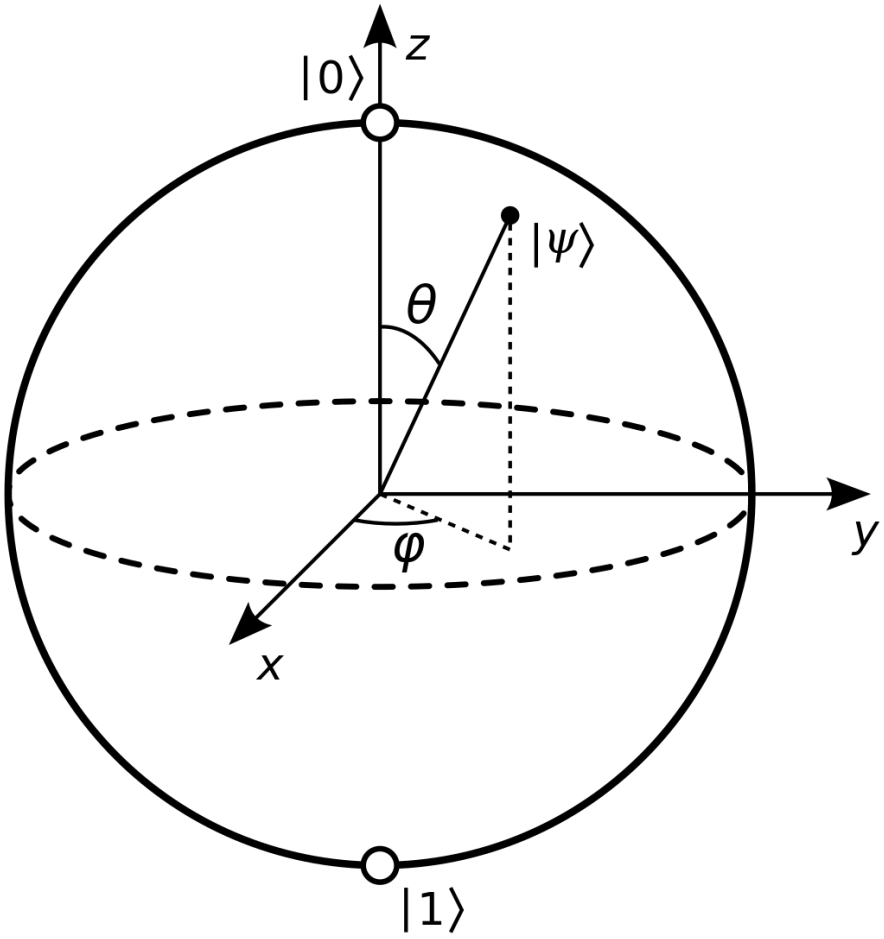


Figure 2.6: An illustration of the Bloch sphere [6].

$$X = \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.7}$$

$$Y = \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{2.8}$$

$$Z = \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.9}$$

$$R_x(\theta) = e^{\frac{-i\theta X}{2}} = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \tag{2.10}$$

$$R_y(\theta) = e^{\frac{-i\theta Y}{2}} = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \tag{2.11}$$

$$R_z(\phi) = e^{\frac{-i\phi Z}{2}} = \begin{pmatrix} e^{-i\frac{\phi}{2}} & 0 \\ 0 & e^{i\frac{\phi}{2}} \end{pmatrix} \tag{2.12}$$

The current paradigm for quantum computation is to use quantum circuits, where quantum computation is represented as a sequence of quantum gates. Each horizontal line (Figure 2.7) represents a singular quantum unit; in this work such a unit will be a qubit (i.e. quantum bit, such that the singular quantum unit can only be in a superposition of a $|0\rangle$ and $|1\rangle$ state (Equation 2.3)).

The most computationally common method to entangle quantum states together is to enact the Controlled NOT (CNOT) operation (Equation 2.13) on two separate qubits.

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.13}$$

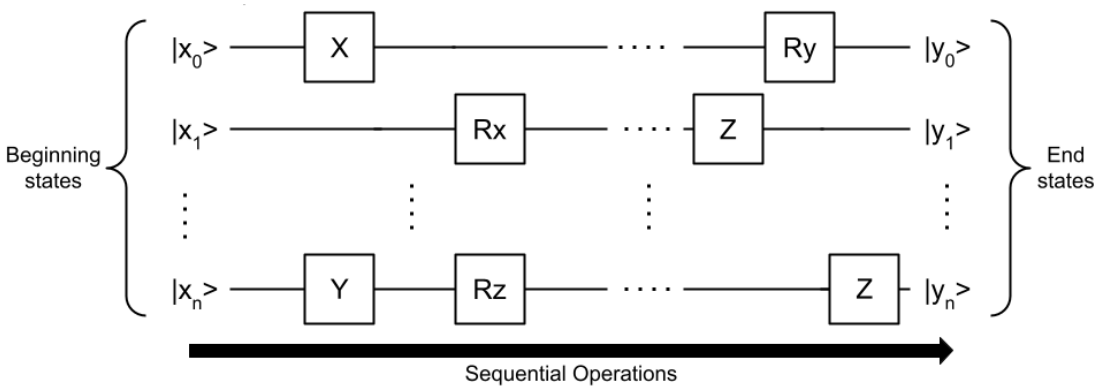


Figure 2.7: An example quantum circuit. The operators X, Y, Z and Rx, Ry, and Rz are defined by Equation 2.7, Equation 2.8, and Equation 2.9, Equation 2.10, Equation 2.11, and Equation 2.12.

After quantum operations are completed, measurement operations are conducted to convert the end quantum states to classical (non-quantum) values (often indicated in circuits with Figure 2.8). There are many ways to take a measurement of a quantum system; one common method is to take the expectation value of an operator as defined in Equation 2.14.

$$\langle A \rangle = \langle \psi | A | \psi \rangle \tag{2.14}$$

The quantum gates CNOT, R_x, R_y, and R_z form a universal set of gates, such that any quantum operation can be expressed as a combination of these gates [36]. There are many different sets of universal gates [36–40], though this set with CNOT and the rotation gates will be the one referenced throughout this text.

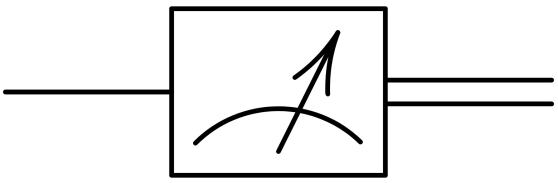


Figure 2.8: The symbol for measurement on a quantum circuit for a single wire. The single line represents the wire (operational quantum units) and the double line represents the classical value resulting from the measurement [7].

2.4 Motivation For Exploring Quantum Computing

The superposition of states and the entanglement of these states allow for a different type of computation resulting in quantum algorithms. In some cases, quantum algorithms allow for faster results over classical computation. For example when factoring numbers, the fastest known classical algorithm is called the General Number Field Sieve which has complexity $O(\exp[c(\log N)^{1/3}(\log\log N)^{2/3}])$, where c is a constant dependent on the type of number being factored, and N is a positive integer [8, 41]. One of the most famous quantum algorithms for factoring is Shor's algorithm which has a computational complexity of $O((\log N)^3 \log\log N \log\log\log N)$ [8]; the aforementioned algorithms have their computational complexity plotted in Figure 2.9, where the lower complexity of Shor's algorithm proves advantageous over the classical General Number Field Sieve. Finding faster quantum factoring algorithms remains a point of active research [8].

For select cases, quantum computing algorithms have shown advantages for machine learning. By leveraging quantum machine learning, some benefits include al-

gorithms that generalize better on less data (i.e. quantum models perform better than classical models on new and previously unseen data with less training data) [42, 43], increased learning efficiency [44, 45], and present a larger effective dimension (i.e. for comparable model architectures, quantum models explore a larger function space than classical models) [46]. However for specialized applications much work needs to be done to examine if and when quantum methods succeed over classical methods.

The current goal of the applied quantum computing community is to find where quantum computing methods do and do not offer improvement over classical computing methods.

Any improvement that quantum computation holds over classical computation is known as a "quantum advantage." It's important to note that in the 2010s and early 2020s, such an advantage was sometimes called "quantum supremacy" due to early papers that showed quantum advantage but for very narrow and highly engineered problems such that the quantum computational approach would win. Such nomenclature of "quantum supremacy" has fallen out of favor and use in the quantum computing community.

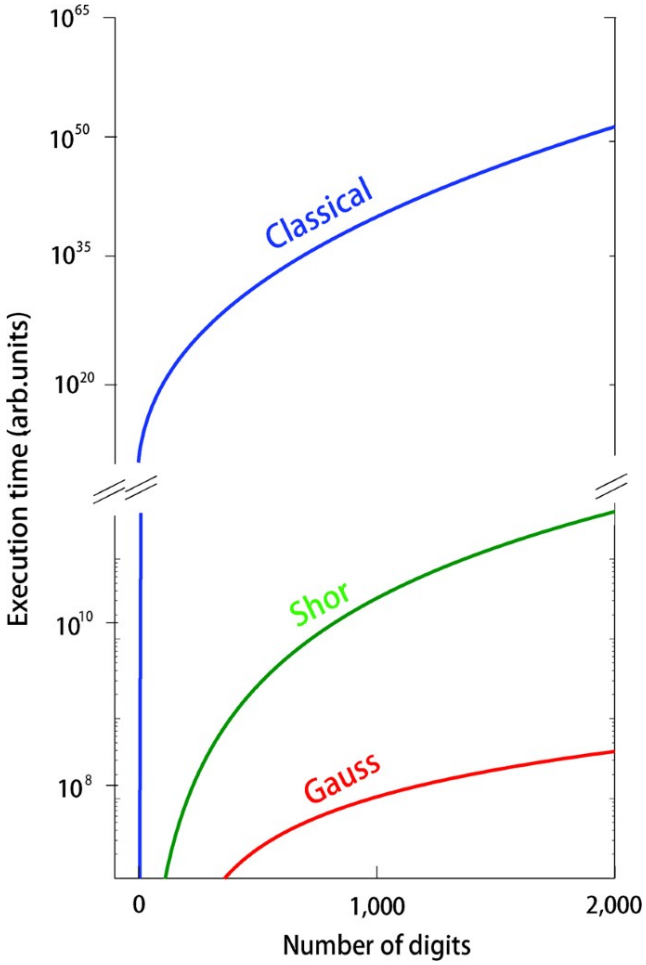


Figure 2.9: Plotting computational complexity for the best known classical factoring algorithm versus Shor's algorithm versus the quantum factoring algorithm (Gauss) proposed in [8].

Chapter 3

Machine Learning Overview: Classical, Quantum, and Spiking Neurons

3.1 Classical Supervised Machine Learning

There are many types of machine learning, the vast majority of which will not be covered here. This work will focus on supervised machine learning and various applications thereof in different domains and forms.

3.1.1 A Brief Reminder Of Supervised Machine Learning Components

At its heart supervised machine learning systems learn how to map input data to target output values. The goal of supervised machine learning is for the learned mapping to produce useful outputs on never-before-seen data.

There are a few components to this learning process. Every output from a machine learning model has a label and this label can be almost anything, e.g. the price

of a house, the type of animal in a picture, the emotions of a Shakespearean verse, etc...

Every input to a machine learning model has a number of features (one or multiple) associated with it. In some larger models, there may be millions of features present.

Machine learning model data that has both features and corresponding labels is called labeled data. Conversely, unlabeled data has only features but no label.

The dataset may be subdivided into equal sized smaller subsets called batches. For example, if there exists 100 data entries in an entire dataset, and a total of 20 batches were needed, then the 100 data entries would be subdivided into 20 batches with 5 data entries per batch.

Taking the above house price as an example target output, the features (inputs) of a house (that may influence the price of a house) can include the number of bedrooms, bathrooms and floors, the age of the house, the total square footage of the house, if there is a backyard and front yard, etc.... If both the features of the house and the price of the house are known then this is called labeled data. If the features of the house are known but the price of the house is unknown then the data is called unlabeled data. Supervised machine learning uses the inputs and labels (outputs) of a labeled house price dataset (e.g. Table 3.1) to teach (train) a supervised machine learning algorithm to generate correct labels (outputs) on an unlabeled dataset (e.g. Table 3.2), i.e. use the machine learning algorithm to solve for the values of "?" in Table 3.2.

Every supervised machine learning model that is trained or taught will have parameters that are tuned to associate the inputs (features) with the corresponding

Features			Label
Age of House (Years)	Number of Bedrooms	Number of Bathrooms	House Price (USD)
5	3	2	100k
10	4	2.5	225k
35	5	3	315k

Table 3.1: An example of a labeled dataset where there are features and labels for housing prices. To be compared with Table 3.2.

Features			Label
Age of House (Years)	Number of Bedrooms	Number of Bathrooms	House Price (USD)
2	2	1.5	?
7	3	2.5	?
18	5	3.5	?

Table 3.2: An example of an unlabeled dataset where there are features but no labels for housing prices. To be compared with Table 3.1.

outputs (labels). It is this tuning of parameters that is called "learning," "training," or "teaching" the model. Parameters that are learned during training are called learned parameters, and parameters that are tuned or set manually are called hyperparameters. Tuning these parameters remains a great difficulty in machine learning (Figure 3.1).

Every supervised machine learning model will also perform inference. Inference is when the supervised machine learning model is passed unlabeled data in order to generate (output) labels, i.e. a supervised machine learning model infers what the correct output label is for a given unlabeled input, based on previous training with labelled data. There are many different metrics that can be used to gauge how well a trained supervised machine learning model generates correct (or incorrect) labels, and some metrics are better for some models more than others; these metrics will be

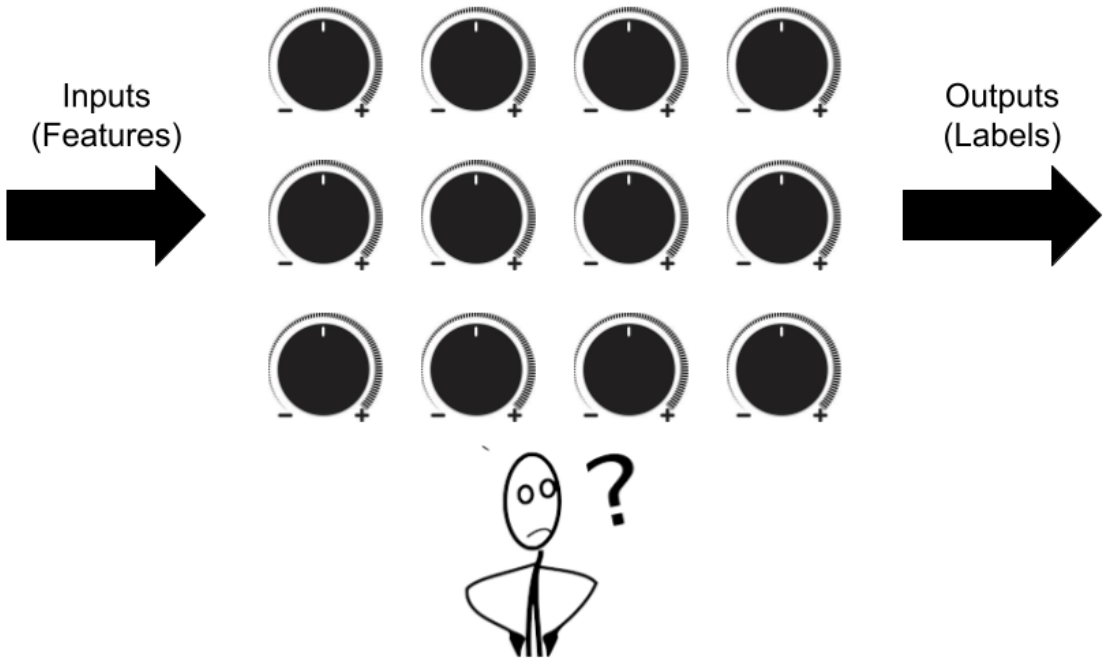


Figure 3.1: Which parameters (dials, knobs) to tune and how much? The conundrum of every machine learning practitioner.

discussed as needed in this text.

There are two popular applications in supervised machine learning: regression and classification. Regression models predict continuous values (e.g. housing prices, the temperature on a given day, financial stock prices, etc...). Classification models predict categories (e.g. is picture of a cat or a dog, is the email spam or not spam, is the car red, white, yellow, or black, etc...)

Classification will be the primary focus of studies done in this text.

3.1.2 Linear Relationships

A very common way to express linear relationships is Equation 3.1, where y is the output, m is known as the slope (the change of rise over the change of run), and b is the y intercept.

$$y = mx + b \tag{3.1}$$

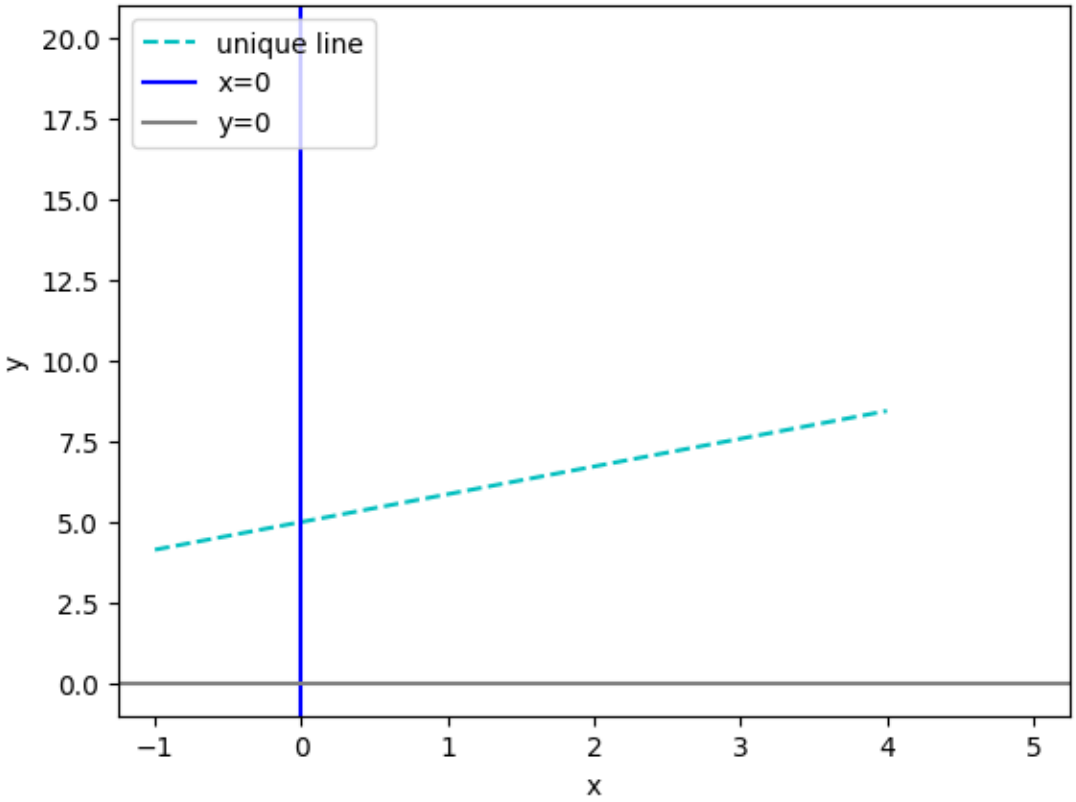


Figure 3.2: A plot of an arbitrary line.

Consider a scenario where m and b both need to be tuned to fit an arbitrary line (Figure 3.2), where x is a feature (input), y is an output (label), and m and b are

parameters that need to be found. Guessing and checking various values of m and b eventually leads to values that would fit this line (Figure 3.3).

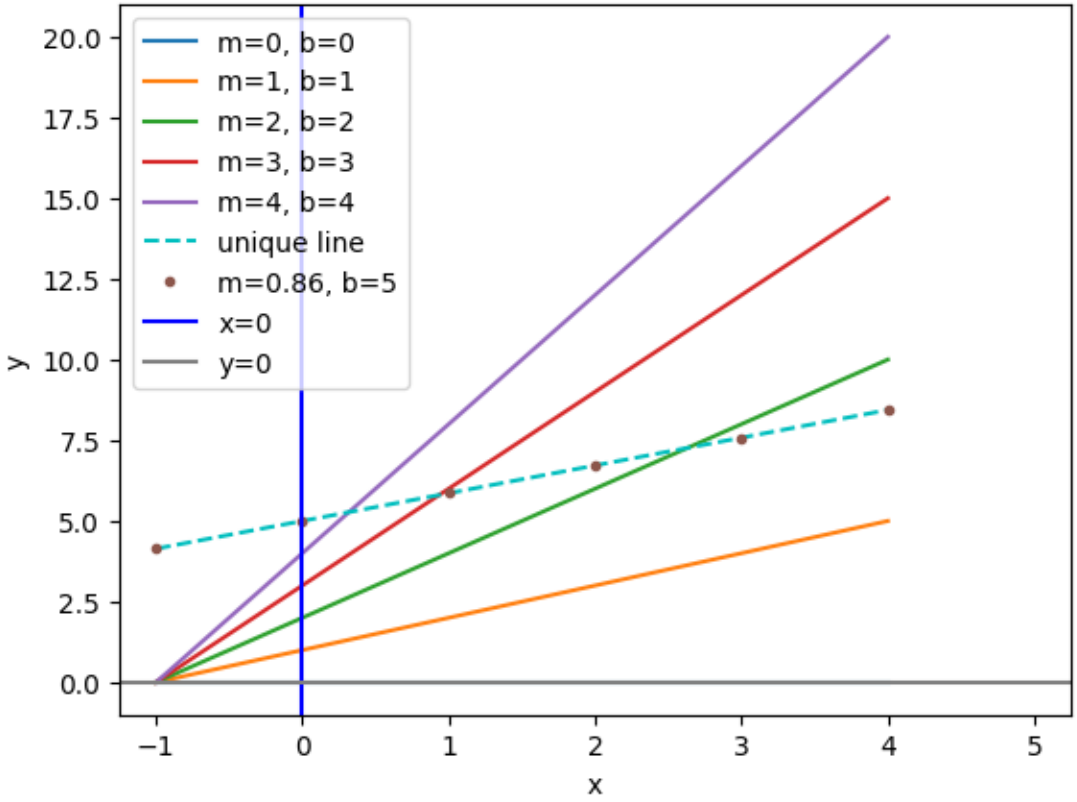


Figure 3.3: A plot of an arbitrary line that is attempted to be fitted by various other lines of the form $y = mx + b$ with different values for m and b.

For more complicated models such educated guessing and checking quickly becomes intractable. In order to expand on the usage of Equation 3.1, there is a need to change how the variables are viewed. To resolve this, Equation 3.1 becomes Equation 3.2 where m becomes w_0, x becomes x_0, and b remains b. w_0 and b become learned parameters while x_0 is a feature (input).

$$y = w_0 x_0 + b \tag{3.2}$$

In the instance where more than one feature needs to be input, for example three features (e.g. x_0, x_1, x_2), then Equation 3.2 can easily expand to Equation 3.3 to capture all the needed features (i.e. x_0, x_1, x_2), how they are weighted (in this case via w_0, w_1, w_2), and any linear shift (as done by b).

$$y = w_0 x_0 + w_1 x_1 + w_2 x_2 + b \tag{3.3}$$

3.1.3 Introducing Non-Linear Functions

For only the simplest of models, a linear expression like that found in Equation 3.3 is enough to approximate a given output, but for more complicated models non-linear functions are incorporated to better model the system of interest while inputting the linear weighting as represented in Equation 3.3.

The symbol σ is used to indicate a non-linear function[1] (also known as an activation function). For example if σ were to be introduced to Equation 3.3 then it would become Equation 3.4.

$$y = \sigma(w_0 x_0 + w_1 x_1 + w_2 x_2 + b) \tag{3.4}$$

Various possible functions for σ include ELU [47], Sigmoid [48], ReLU [49],

[1] Quite frustratingly this isn't always true. σ is also often used to denote the sigmoid function. When σ represents a non-linear function and not the sigmoid function is often determined by context and/or the text.

GELU [50], and many others. Deciding on which activation functions to use and when remains an area of active experimentation and research.

3.1.4 The Artificial Neuron

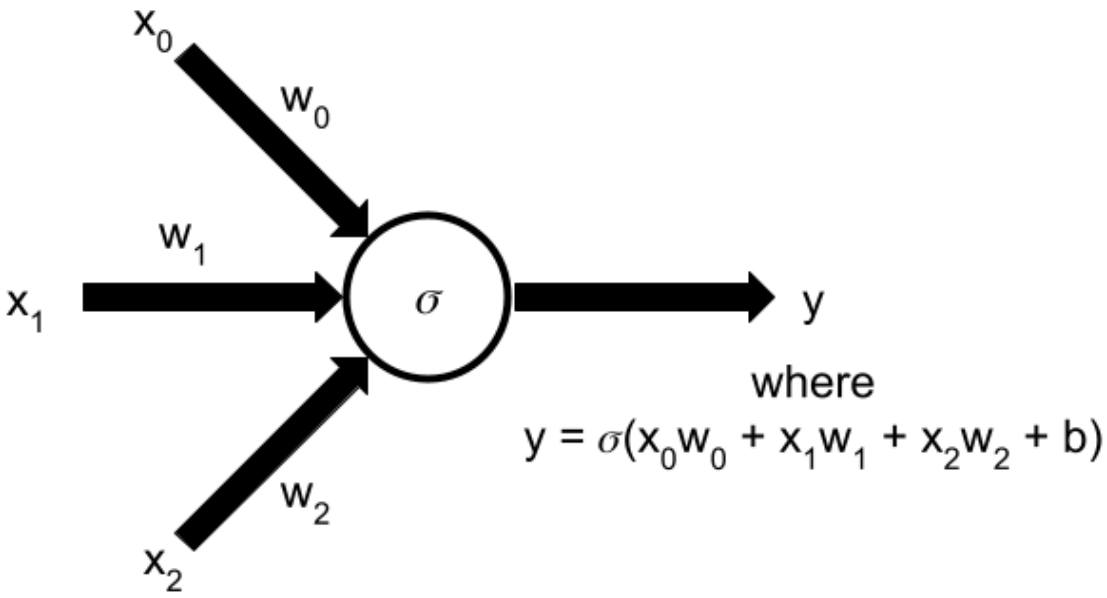


Figure 3.4: An example of an artificial neuron that explicitly states that the neuron imparts a non-linear function onto the inputs.

It is common in machine learning to express equations similar to those found in Equation 3.3, or Equation 3.4 as an artificial neuron as seen in Figure 3.4. In Figure 3.4 each line represents a weight and each circle represents an artificial neuron (sometimes called a node) which takes in each input, multiplies each input times a randomized weight, imparts an (often) activation function (σ) on the inputs, and then returns a new value. In illustrated models the activation function is often not listed explicitly in the model itself as done Figure 3.4 but rather the node(s) in the model is left blank and

such activation function is explicitly listed elsewhere in the text.

3.1.5 Artificial Neural Networks

When equations like those found in Equation 3.4 are chained together, they represent artificial neurons connected together like those found in Figure 3.5. Connecting artificial neurons together creates an artificial neural network (or simply a "neural network"), and sufficiently large neural networks are universal function approximators [51].

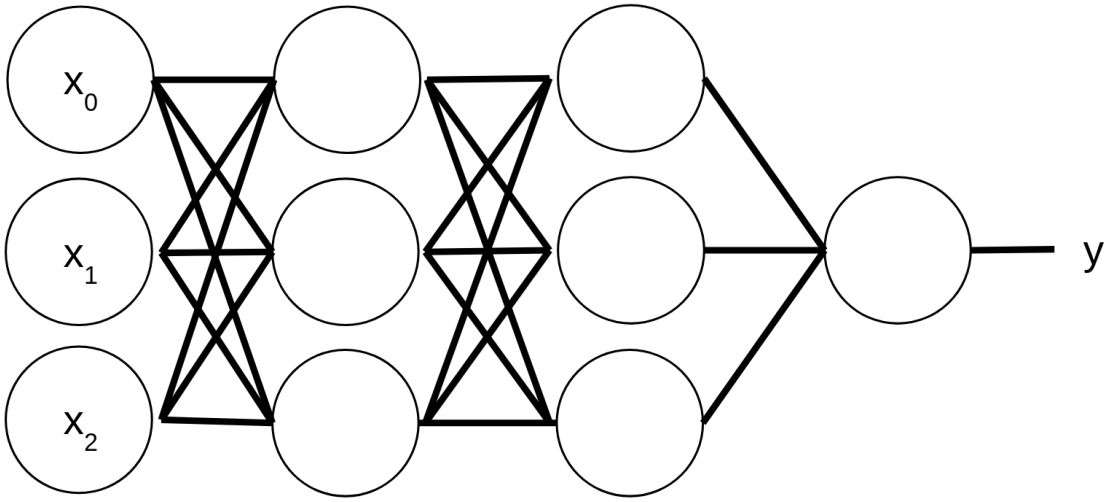


Figure 3.5: An example of an artificial neural network model. The vertical column with the nodes containing x_1, x_2 and x_3 inside each node is called the input layer. The single node leading to y is called the output layer. The vertical columns of nodes between the input layer and the output layer are called hidden layers.

When information flows through an artificial neural network, starting from inputs, going through the various artificial neurons, and ending in an output, such information flow is known as "forward propagation" or a "forward pass" since the in-

formation is flowing forward through the network.

3.1.6 Understanding Loss

There is now a need to measure how well the desired target output (label) compares with the generated output as previously defined by equations (e.g. Equation 3.2, Equation 3.3, or Equation 3.4). The function that measures difference between what the neural network outputs and what the expected output of the neural network should be is known as the "loss function" (also sometimes called the "cost function" or "criterion"). Most members of the machine learning community use these terms interchangeably both in speech and code, though specialized publications may assign strict definitions to some terms [52].

Many loss functions exist, some loss functions are appropriate for some applications while inappropriate for others, and many different loss functions are applicable for the same problem. Hence the choice of loss function remains a hyperparameter that is user-defined.

The mean-square error is among the simplest of loss functions and can be seen in Equation 3.5, where y is the expected value, $\hat{y}$ is the predicted value as output from the neural network, and m is the number of data points that Equation 3.5 computes.

$$\text{mean-square error} = \text{MSE} = \frac{1}{m}\sum_{i=0}^{m}(y_i - \hat{y}_i)^2 \tag{3.5}$$

3.1.7 Backpropagation By Example

The loss function (a measure of the difference between the output of a neural network versus the expected (ideal) output) can now be used to update the corresponding weights of the neural network; calculating how learned parameters influence the output is called backpropagation (sometimes also called "backprop" or the "backward pass"). The act of updating these parameters is referred to as an "update step."

A simple example below shows how backpropagation works; here σ stands for the sigmoid function (Equation 3.6).

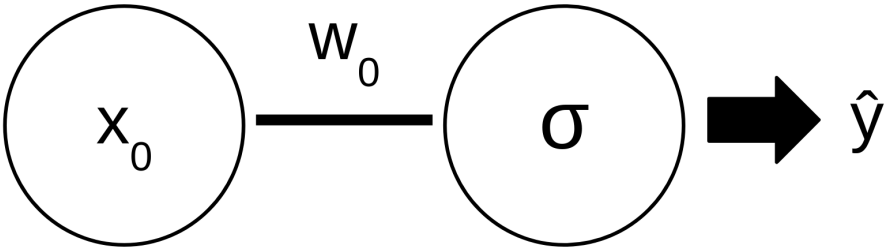


Figure 3.6: A simple artificial neural network model.

$$\sigma(x) := \text{sigmoid} = \frac{1}{1 + e^{-x}} \tag{3.6}$$

Taking the simple neural network as presented in Figure 3.6, the forward pass multiplies x_0 with w_0, passes it to the (only) node with a σ activation function, and then outputs $\hat{y}$. Therefore $\hat{y}$ can be expressed as Equation 3.7 where w_0 and b are learned parameters.

$$\begin{aligned} \hat{y} &= \sigma(x_0 w_0 + b) \\ \rightarrow \hat{y} &= \sigma(z) \quad \text{where } z = x_0 w_0 + b \end{aligned} \tag{3.7}$$

The corresponding cost function in this example is the mean-squared error as seen in Equation 3.8, where y is the target output and $\hat{y}$ is the returned output from the neural network.

$$C = (y - \hat{y})^2 \tag{3.8}$$

The output z varies as a function of learned parameters w_0 and b; how these parameters vary the output z informs how the learned parameters w_0 and b will be updated.

Note that the derivative of the sigmoid function can be found in Equation 3.9.

$$\frac{\partial \sigma(x)}{\partial x} = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x)(1 - \sigma(x)) \tag{3.9}$$

Therefore C varies with respect to w via Equation 3.10.

$$\begin{aligned} \frac{\partial C}{\partial w} &= \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w} \\ \rightarrow \frac{\partial C}{\partial w} &= (2(y - \hat{y}))(\sigma(z)(1 - \sigma(z)))(x_0) \\ \rightarrow \frac{\partial C}{\partial w} &= (2(y - \hat{y}))(\sigma(x_0 w_0 + b)(1 - \sigma(x_0 w_0 + b)))(x_0) \end{aligned} \tag{3.10}$$

C varies with respect to b via Equation 3.11.

$$
\begin{aligned}
\frac{\partial C}{\partial b} &= \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial b} \\
\rightarrow \frac{\partial C}{\partial b} &= (2(y - \hat{y}))(\sigma(z)(1 - \sigma(z)))(1) \\
\rightarrow \frac{\partial C}{\partial b} &= (2(y - \hat{y}))(\sigma(x_0 w_0 + b)(1 - \sigma(x_0 w_0 + b)))(1)
\end{aligned}
\tag{3.11}
$$

3.1.8 Updating Parameters

With the gradients of the cost function with respect to the learned parameters now in hand (Equation 3.10 and Equation 3.11), the "update step" changes the values of w_0 and b to bring $\hat{y}$ closer to y.

Among the simplest of update steps include Equation 3.12 and Equation 3.13 that use the results from Equation 3.10 and Equation 3.11, respectively. In both equations (Equation 3.12 and Equation 3.13) w_{initial} and b_{initial} both represent the initial parameter values, w_{final} and b_{final} represent the updated values for w and b, and η ($\eta > 0$) represents the learning rate for the update. For a learned parameter, the learning rate (η) (also called a step size) is a positive real-valued number that regulates how much the calculated gradient (e.g. $\frac{\partial C}{\partial b}$) of the parameter changes the parameter update step (e.g. b_{final}).

$$
w_{\text{final}} = w_{\text{initial}} - \eta \frac{\partial C}{\partial w} \tag{3.12}
$$

$$
b_{\text{final}} = b_{\text{initial}} - \eta \frac{\partial C}{\partial b} \tag{3.13}
$$

Other parameter update methods exist, although the ones used in Equation 3.12 and Equation 3.12 are among the simplest and most straightforward. Other update methods will not be addressed in this text and lie outside its scope.

The above process updates the value of learned parameters with the goal of lowering the Loss (calculated by the cost function) through progressive update steps. This can be visualized in Figure 3.7 in the case of a simple convex function with only one learned parameter.

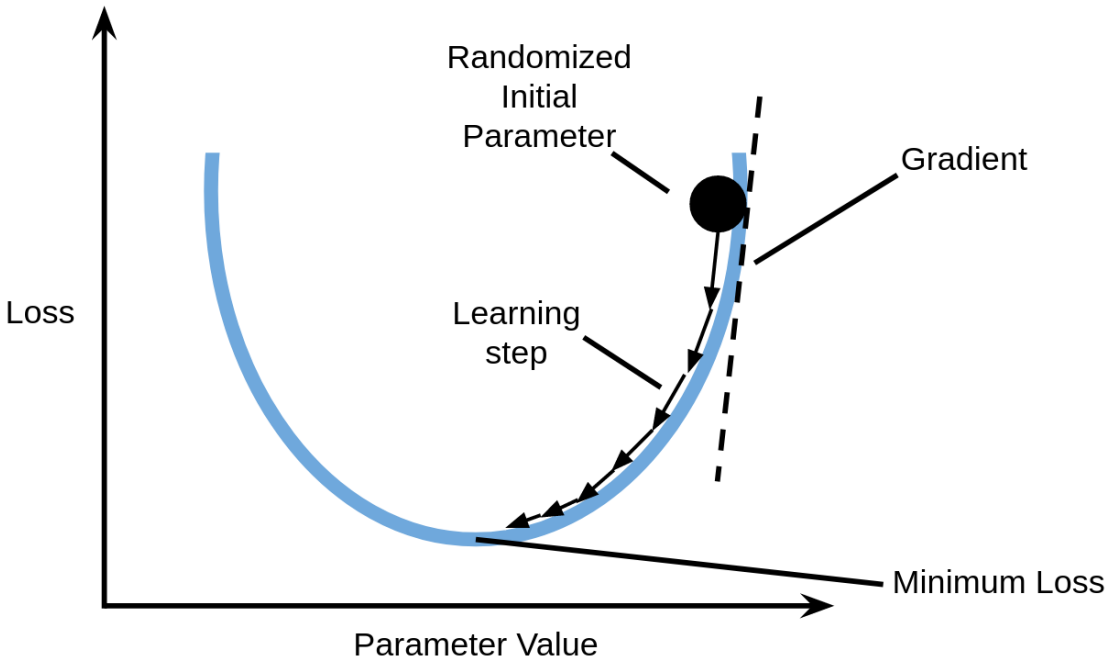


Figure 3.7: Gradient descent visualized in the case of a simple convex function with only one learned parameter.

3.2 Quantum Machine Learning

In performing quantum machine learning, many of the principles and implementations from previous sections are still at play and exist in a nearly identical form.

3.2.1 Parameters To Optimize

First recall that the quantum gates R_x, R_y, and R_z (Equation 2.10, Equation 2.11, and Equation 2.12) have the parameters θ and ϕ. If a combination of these gates are placed on a quantum circuit ending in a differentiable measurement, then the measurement can feed values into a classical loss function and optimizer to conduct machine learning where the learned parameters θ and ϕ are optimized. An example of a quantum circuit with a learned parameter (θ) can be seen in Figure 3.8. Of course, any parameter within a gate can be treated as a learned parameter.

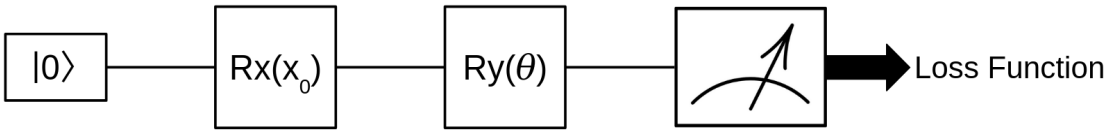


Figure 3.8: An example of a single quantum wire that has a learned parameter (θ) that will be updated. Here x_0 is a constant real value (i.e. a value that will not be learned in the training process) that is inserted into the quantum circuit via the R_x (Equation 2.10) gate. The state $|0\rangle$ initializes the circuit.

3.2.2 Gradients In Quantum Systems

Finding gradients of quantum gates is an active and growing field of research [53–55]. For the quantum gates R_x, R_y, and R_z (Equation 2.10, Equation 2.11, and Equation 2.12) their gradients are known to be defined by the parameter shift rule as seen in Equation 3.14 [56, 57], where f is the rotational gate of interest and θ is the learned parameter of interest. More general parameter shift rules exist and need to be solved on a case-by-case basis [53–55].

$$\nabla_\theta f(\theta) = \frac{1}{2}[f(\theta + \frac{\pi}{2}) - f(\theta - \frac{\pi}{2})] \tag{3.14}$$

However other methods for finding gradients (and quantum gradients) exist, including finite difference methods. Such difference methods include the forward finite difference (Equation 3.15), the backward finite difference (Equation 3.16), and the central finite difference (Equation 3.17) as illustrated in Figure 3.9.

$$\text{forward finite difference: } \nabla_x f(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{3.15}$$

$$\text{backward finite difference: } \nabla_x f(x) = \frac{f(x) - f(x - \Delta x)}{\Delta x} \tag{3.16}$$

$$\text{central finite difference: } \nabla_x f(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} \tag{3.17}$$

Note that Δx in Equation 3.15, Equation 3.16, and Equation 3.17 is a hyper-parameter which determines how much of a shift the finite difference method takes in

order to determine the gradient. The parameter x in Equation 3.15, Equation 3.16, Equation 3.17, and Figure 3.9 is a learned parameter in need of optimization.

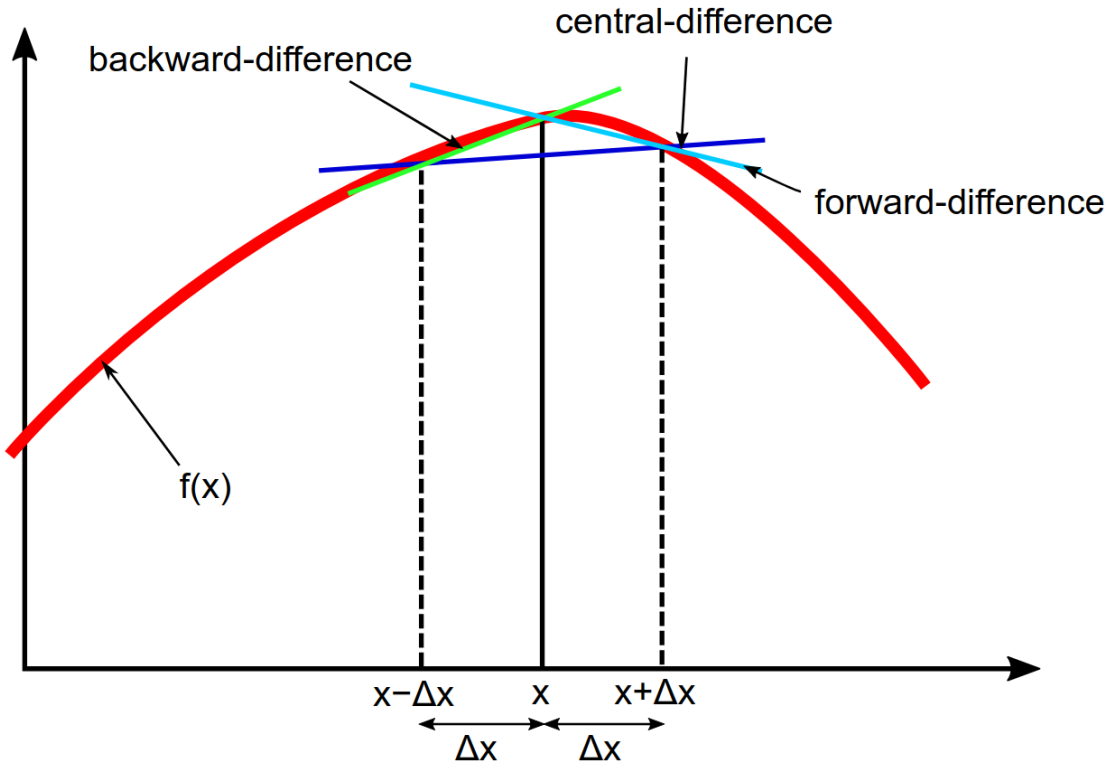


Figure 3.9: An illustration of the central finite difference for the point x on a function [9].

3.3 Spiking Neural Networks

3.3.1 A New Generation: Spiking Neural Networks

Machine learning, and in particular deep learning, is taking more computational resources every year to run. From 2012 to 2019 the amount of computational power needed to run the best performing deep learning models have increased ten times per year. One estimate places OpenAI's GPT-3 needing 190,000 kWh to train 175

billion learned parameters [58–60], while the brain maintains autonomous bodily functions and processes multisensory inputs with only about 12 W to 20 W of power [61]. Given that deep learning models will likely only get larger and thereby require more computational power with time [10], the human brain which leverages spiking neuron behavior serves as a point of inspiration and efficiency in alternative machine learning model design.

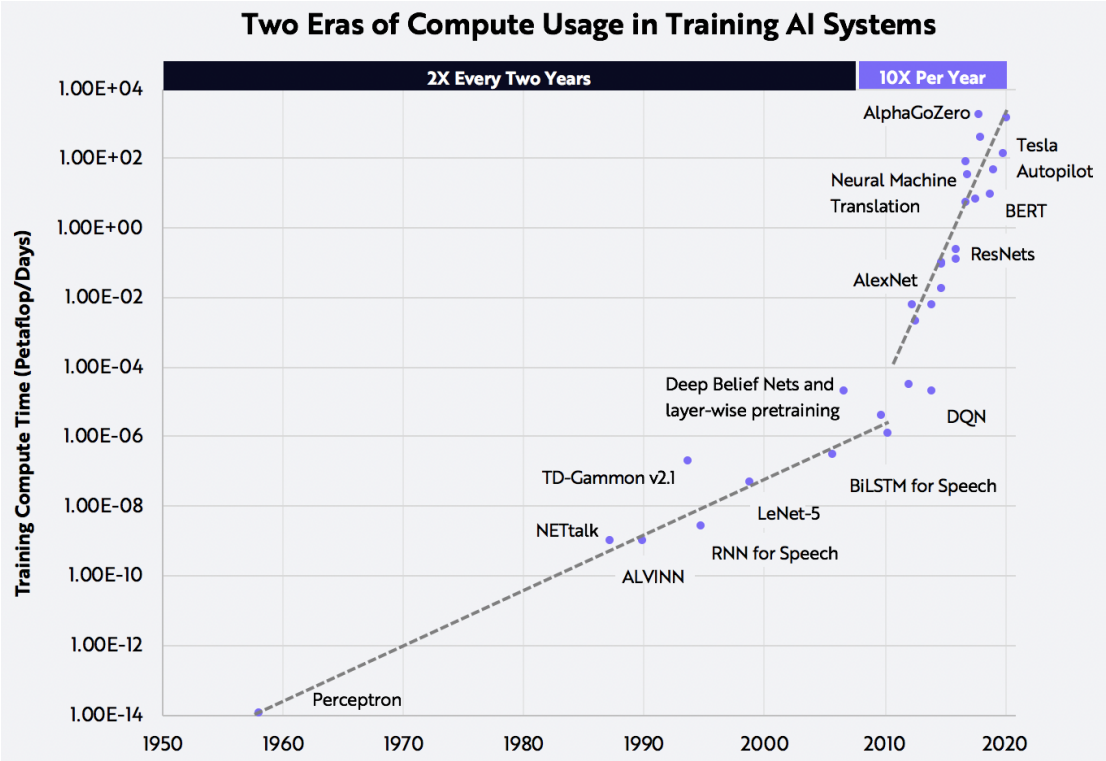


Figure 3.10: An illustration of training compute time versus year for popular models [10].

To this point of inspiration, a new paradigm of machine learning called spiking neural networks (SNNs) are quickly gaining traction as they strive to bridge the difference between the brain and deep learning efficiency and behavior.

3.3.2 A Review Of Select Neuron Anatomy And Function

In order to understand how spiking neural networks are inspired and constructed, a quick review of neuron anatomy and function is needed.

Recall that the biological neuron (Figure 3.11) is made of many parts. At a high level all neurons have three basic tasks: receive information or signals; integrate incoming and received signals; and send signals to other cells (which don't have to necessarily be other neurons, as they can also be glands, muscles, etc...) [62].

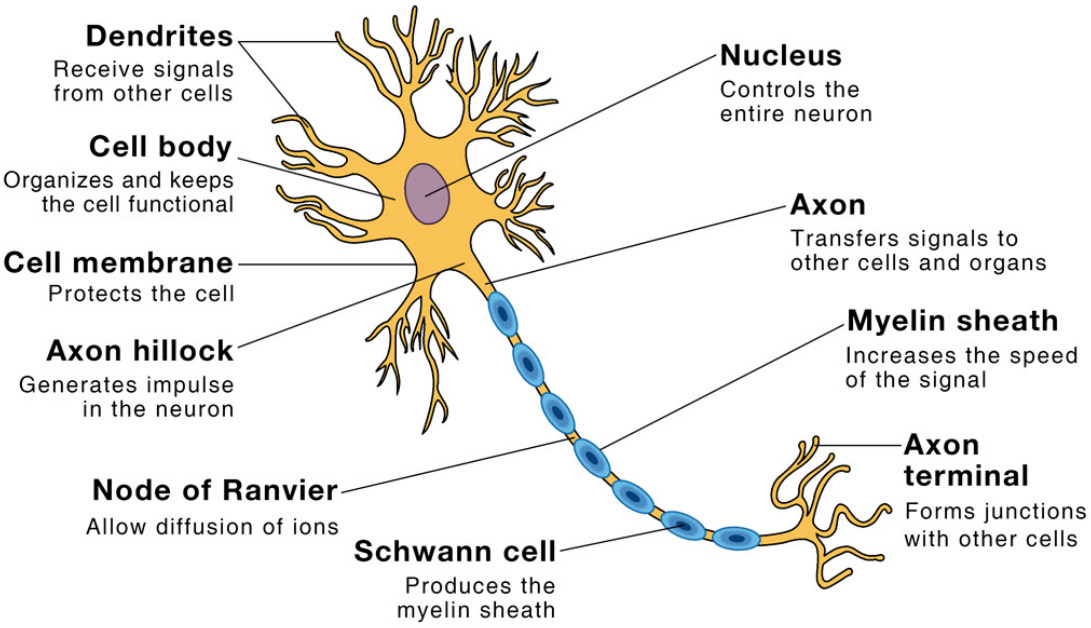


Figure 3.11: An illustration of a biological neuron with parts of the neuron labeled and described [11].

Most of the time, neurons tend to have more negatively charged ions inside them while the extracelluar fluid around them tends to be made up of positively charged ions; this difference in charged ion concentration is called a concentration gradient, and

the neuron state when neurons have a negative concentration gradient (negative because the neuron is net negatively charged) is called a resting membrane potential.

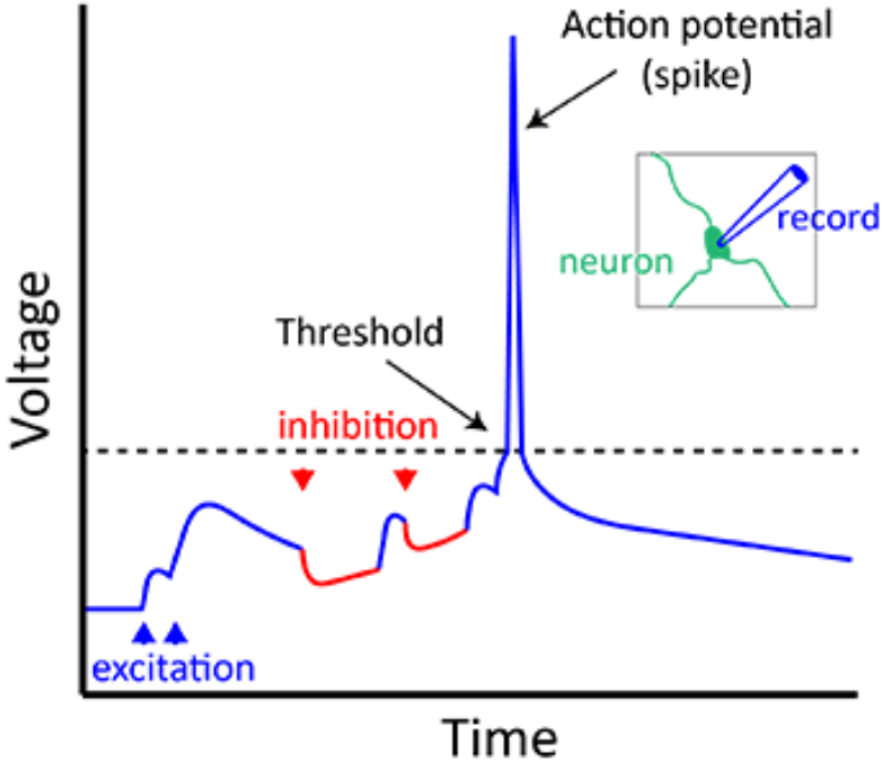


Figure 3.12: An illustration of a neuron spike [12].

When the dendrites receive signals from other cells, the soma (cell body) integrates these signals. If sufficient excitatory and inhibitory signals are sent to the soma then the membrane potential reaches the action potential threshold. Once the membrane potential gets raised to the action potential threshold the neuron fires an action potential (i.e. nerve impulse or spike) (Figure 3.12) down the axon to the axon terminals (i.e. nerve terminals) which make connections to other cells. Synapses are the sites where neuron-to-neuron connections are made [62]. The action potential works

on an "all-or-nothing" principle: if the membrane potential reaches the action potential threshold then the neuron spikes, otherwise if the threshold is not reached the neuron does not spike; the strength of the spike does not depend on whether there is excess potential as it will always fire with the same strength. After the neuron fires then the neuron's membrane resets back to a resting membrane potential through a process called repolarization and hyperpolarization whose details are unimportant here [63]. An example series of these measured biological spikes can be seen in Figure 3.13.

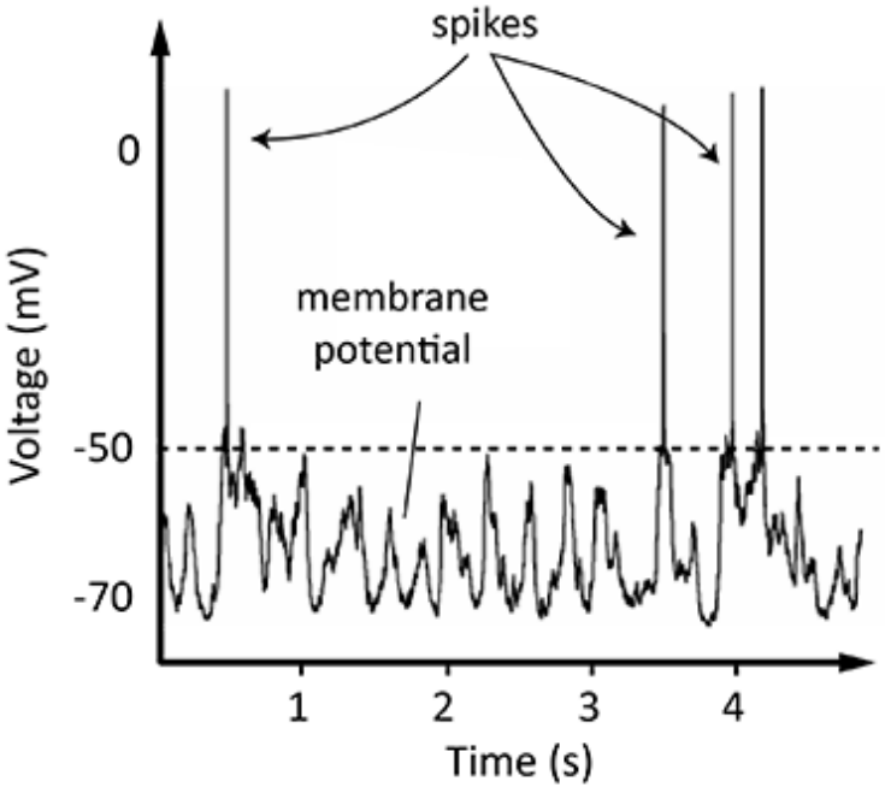


Figure 3.13: An illustration of real and measured neuron spiking behavior [12].

3.3.3 Introduction To The Leaky Integrate And Fire Neuron Model

The neuron is encased by a thin lipid bilayer membrane that insulates a conductive saline solution inside the neuron from the extracellular medium that lies outside the neuron. The lipid bilayer membrane also helps regulate what moves into and out of the cell (e.g. sodium and potassium ions). The membrane is generally impermeable except for specific channels that allow for ions to flow into the neuron. This cell membrane leakage resistance can be modeled as a resistor. The lipid bilayer membrane separating the saline solution and extracellular medium can be modeled as a capacitor (Figure 3.14). The entire system can be modeled as an RC circuit (Figure 3.15). This neuron model is known as the leaky integrate and fire (LIF) neuron model [14].

The total current in Figure 3.15 is given by Equation 3.18 (where R is for the resistor and C is for the capacitor).

$$I_{\text{in}}(t) = I_R + I_C \tag{3.18}$$

The capacitance is given by the ratio of the charge on the capacitor and the potential Equation 3.19 (where U_{mem}[2] is the membrane potential (also commonly labeled as V with the usual units of volts)) and Q is the charge.

$$\begin{aligned} C &= \frac{Q}{U_{mem}(t)} \\ \rightarrow Q &= CU_{\text{mem}}(t) \end{aligned} \tag{3.19}$$

[2]Here U is used to be consistent with spiking neural network literature.

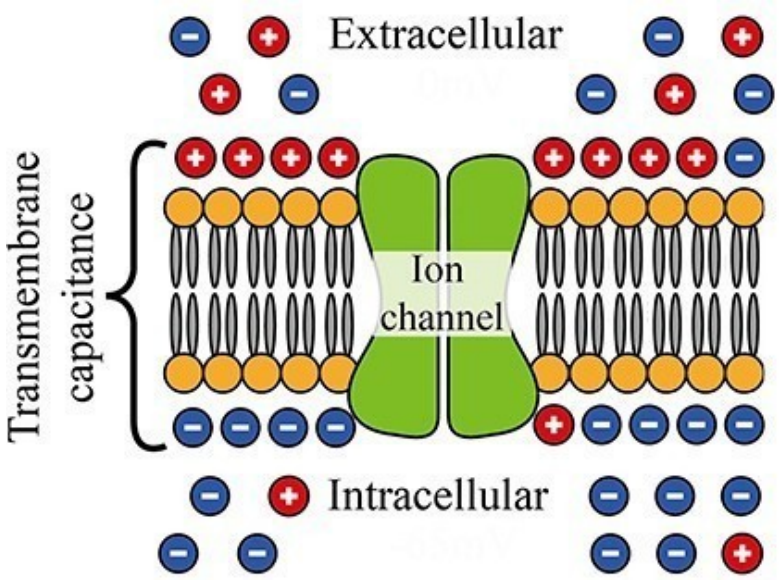


Figure 3.14: An illustration of a lipid bilayer (golden circles with grey forks) separating the extracellular and intracellular medium with ions on either side. The ion channel resisting the flow of ions (charges) in and out is modeled as a resistor. The lipid bilayer separating the extracellular and intracellular ions (and thus creates voltage differential across the membrane) is modeled as a capacitor. This is detailed in Section 3.3.3 [13].

Taking the time derivative of both sides in Equation 3.19 yields the current through the capacitor (Equation 3.20).

$$\begin{aligned} I_C(t) &= \frac{dQ}{dt} = C\frac{dU_{\text{mem}}(t)}{dt} \\ \rightarrow I_C(t) &= C\frac{dU_{\text{mem}}(t)}{dt} \end{aligned} \tag{3.20}$$

The ion channel, which resists flow of ions between the inside and outside of the membrane, is modeled as a resistor, as given by Ohm's law (Equation 3.21).

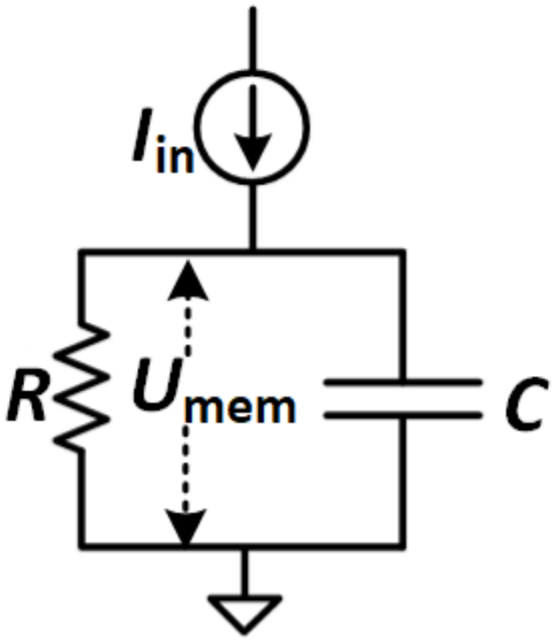


Figure 3.15: A resistor-capacitor (RC) circuit which models a biological neuron. Input current I_{in} models excitatory signals flowing into the neuron, the resistance R models the membrane leakage resistance to ion current flow through the ion channels, and the electrical potential U_{mem} models the measured potential between the inside and outside of the neuron [14].

$$I_R(t) = \frac{U_{\text{mem}}(t)}{R} \tag{3.21}$$

Plugging in Equation 3.20 and Equation 3.21 into Equation 3.18 yields Equation 3.22.

$$
\begin{aligned}
I_{\text{in}}(t) &= I_R + I_C \\
\rightarrow I_{\text{in}}(t) &= \frac{U_{\text{mem}}(t)}{R} + C\frac{dU_{\text{mem}}(t)}{dt}
\end{aligned}
\tag{3.22}
$$

Moving forward let $U(t) = U_{mem}(t)$ for simplicity. Taking Equation 3.22 and getting the terms RC on one side of the equation yields Equation 3.23.

$$
\begin{aligned}
I_{\text{in}}(t) &= \frac{U(t)}{R} + C\frac{dU(t)}{dt} \\
\rightarrow RC\frac{dU(t)}{dt} &= -U(t) + RI_{\text{in}}(t)
\end{aligned}
\tag{3.23}
$$

The quantity RC is also known as the time constant of the circuit, τ, so substituting $RC = \tau$ into Equation 3.23 yields Equation 3.24.

$$
\begin{aligned}
RC\frac{dU(t)}{dt} &= -U(t) + RI_{\text{in}}(t) \\
\rightarrow \tau\frac{dU(t)}{dt} &= -U(t) + RI_{\text{in}}(t)
\end{aligned}
\tag{3.24}
$$

The latter part of Equation 3.24 yields the differential equation Equation 3.25.

$$
\tau\frac{dU(t)}{dt} = -U(t) + RI_{\text{in}}(t) \tag{3.25}
$$

3.3.4 Discretely Solving The ODE

The forward Euler method solves this ordinary differential equation (ODE) (Equation 3.25) yielding a discrete form which is useful for spiking neural networks. This is done by taking the derivative of Equation 3.25 without taking the limit $\Delta t \rightarrow 0$ which gives Equation 3.26.

$$\tau \frac{U(t+\Delta t) - U(t)}{\Delta t} = -U(t) + RI_{\text{in}}(t) \tag{3.26}$$

If Δt is small enough then Δt will give a good approximation for integrating in continuous time. Solving for U gives Equation 3.27.

$$\begin{aligned} \tau \frac{U(t+\Delta t) - U(t)}{\Delta t} &= -U(t) + RI_{\text{in}}(t) \\ \rightarrow U(t+\Delta t) - U(t) &= \frac{\Delta t}{\tau}\big(-U(t) + RI_{\text{in}}(t)\big) \\ \rightarrow U(t+\Delta t) &= U(t) + \frac{\Delta t}{\tau}\big(-U(t) + RI_{\text{in}}(t)\big) \\ \rightarrow U(t+\Delta t) &= (1 - \frac{\Delta t}{\tau})U(t) + \frac{\Delta t}{\tau} I_{\text{in}}(t)R \end{aligned} \tag{3.27}$$

Consider the scenario that no input current exists, i.e. $I_{\text{in}}(t) = 0A$ which yields Equation 3.28.

$$U(t+\Delta t) = (1 - \frac{\Delta t}{\tau})U(t) \tag{3.28}$$

From Equation 3.28 it can be seen that Equation 3.29 is less than unity since Δt is small, so therefore $\frac{\Delta t}{\tau} < 1$.

$$\frac{U(t+\Delta t)}{U(t)} = (1 - \frac{\Delta t}{\tau}) \tag{3.29}$$

The ratio $\frac{U(t+\Delta t)}{U(t)}$ is the decay rate of the membrane potential β, also known as the inverse time constant as seen in Equation 3.30. This decay rate models how the voltage across the membrane decreases as the difference in charge is across the

membrane is brought to equilibrium as ions flowing through the ion channel bring the neuron to equilibrium: zero potential, i.e. zero voltage difference.

$$\begin{aligned} \beta = \frac{U(t+\Delta t)}{U(t)} &= (1 - \frac{\Delta t}{\tau}) \\ \rightarrow U(t+\Delta t) &= \beta U(t) \end{aligned} \tag{3.30}$$

Equation 3.30 also yields Equation 3.31 and Equation 3.32.

$$\beta = (1 - \frac{\Delta t}{\tau}) \tag{3.31}$$

$$\begin{aligned} \beta &= (1 - \frac{\Delta t}{\tau}) \\ \rightarrow \frac{\Delta t}{\tau} &= 1 - \beta \end{aligned} \tag{3.32}$$

Plugging the results from Equation 3.31 and Equation 3.32 into Equation 3.27 yields Equation 3.33.

$$\begin{aligned} U(t+\Delta t) &= (1 - \frac{\Delta t}{\tau})U(t) + \frac{\Delta t}{\tau} I_{\text{in}}(t) R \\ \rightarrow U(t+\Delta t) &= \beta U(t) + (1-\beta) I_{\text{in}}(t) R \end{aligned} \tag{3.33}$$

If time is taken to represent discrete, sequential time-steps then for simplicity let $\Delta t = 1$ and assume that $R = 1\Omega$. Using discrete time leads to the assumption that each time bin t is small enough so that only a maximum of one spike is emitted by a neuron in this interval. By also assuming that the input current instantaneously contributes to the current state of the membrane potential, the above gives rise to Equation 3.34.

$$U(t+1) = \beta U(t) + (1-\beta) I_{\text{in}}(t) \tag{3.34}$$

3.3.5 An Analytical β

A continuous and analytical solution to β can be found by solving Equation 3.25, where the solution for Equation 3.25 is given by Equation 3.39, where k is an unknown constant.

Starting off with Equation 3.25 and rearranging terms yields Equation 3.35.

$$\frac{dU(t)}{dt} + \frac{1}{\tau} U(t) = \frac{1}{\tau} I_{\text{in}}(t) R \tag{3.35}$$

This differential equation has the well known solution of the form [64] Equation 3.36 where $\mu(t)$ is given by Equation 3.37.

$$U(t) = \frac{1}{\mu(t)} \left(\int \mu(t) \frac{1}{\tau} I_{\text{in}}(t) R dt + k \right) \tag{3.36}$$

$$\begin{aligned} \mu(t) &= e^{\int \frac{1}{\tau} dt} \\ \rightarrow \mu(t) &= e^{\frac{t}{\tau}} \end{aligned} \tag{3.37}$$

Plugging the result of Equation 3.37 into Equation 3.36 yields Equation 3.38.

$$
\begin{aligned}
U(t) =& \frac{\int \mu(t)\frac{1}{\tau}I_{\text{in}}(t)Rdt + k}{\mu(t)} \\
\rightarrow& \frac{\int e^{\frac{t}{\tau}}\frac{1}{\tau}I_{\text{in}}(t)Rdt + k}{e^{\frac{t}{\tau}}} \\
\rightarrow& \frac{\tau e^{\frac{t}{\tau}}\frac{1}{\tau}I_{\text{in}}(t)R + k}{e^{\frac{t}{\tau}}} \\
\rightarrow& I_{\text{in}}(t)R + ke^{\frac{-t}{\tau}} \\
\implies U(t) =& I_{\text{in}}(t)R + ke^{\frac{-t}{\tau}} = I_{\text{in}}(t)R + ke^{\frac{-t}{RC}}
\end{aligned}
\tag{3.38}
$$

From the above, the result to Equation 3.38 is Equation 3.39.

$$
U(t) = I_{\text{in}}(t)R + ke^{\frac{-t}{RC}} \tag{3.39}
$$

The boundary condition to solve for k in Equation 3.39 is that $U(t=0) = U_0$ which yields

$$
\begin{aligned}
& U(t=0) = I_{\text{in}}(t)R + ke^{\frac{-0}{RC}} \\
\rightarrow & U_0 = I_{\text{in}}(t)R + k(1) \\
\rightarrow & k = U_0 - I_{\text{in}}(t)R
\end{aligned}
\tag{3.40}
$$

Plugging in the result from Equation 3.40 into Equation 3.39 yields Equation 3.41.

$$
U(t) = I_{\text{in}}(t)R + (U_0 - I_{\text{in}}(t)R)e^{\frac{-t}{RC}} \tag{3.41}
$$

Note that if the input $I_{\text{in}}(t) = 0$ then Equation 3.41 becomes Equation 3.42. Equation 3.42 then solves the differential equation Equation 3.35 (and by extension

Equation 3.25) when $I_{\text{in}}(t) = 0$. This is important because it says that without current input then the membrane potential will start at U_0 and will exponentially decay with a time constant $\tau = RC$.

$$\begin{aligned} U(t) &= I_{\text{in}}(t)R + (U_0 - I_{\text{in}}(t)R)e^{\frac{-t}{RC}} \\ \rightarrow U(t) &= (0) + (U_0 - (0))e^{\frac{-t}{RC}} \\ \rightarrow U(t) &= U_0 e^{\frac{-t}{RC}} = U_0 e^{\frac{-t}{\tau}} \end{aligned} \tag{3.42}$$

Therefore the final result of Equation 3.42 is Equation 3.43.

$$U(t) = U_0 e^{-\frac{t}{RC}} = U_0 e^{-\frac{t}{\tau}} \tag{3.43}$$

Note that if Equation 3.43 is determined at discrete values of t such that the values are t, $(t + \Delta t)$, $(t + 2\Delta t)$, ... , then the definition for β as given in Equation 3.30 becomes Equation 3.44.

$$\begin{aligned} \beta &= \frac{U(t + \Delta t)}{U(t)} \\ \rightarrow \beta &= \frac{U_0 e^{-\frac{t+\Delta t}{\tau}}}{U_0 e^{-\frac{t}{\tau}}} = \frac{U_0 e^{-\frac{t+2\Delta t}{\tau}}}{U_0 e^{-\frac{t+\Delta t}{\tau}}} = \quad ... \\ \implies \beta &= e^{-\frac{\Delta t}{\tau}} \end{aligned} \tag{3.44}$$

The definition of β as determined in Equation 3.44 is preferred because it generalizes β while $\beta = (1 - \frac{\Delta t}{\tau})$ rests on the limit $\Delta t << \tau$. Therefore the definition of β as given in Equation 3.44 can be used in Equation 3.34 and the reminder of this work.

It's important to note that at a high level the discrete approach (Equation 3.34) yields a solution of the form useful for spiking neural networks and the analytical approach (Equation 3.43) gives a precise definition for β (Equation 3.44). Therefore by combining the both solutions a more precise form of the discrete approach helps to create and define spiking neural networks (as seen in Section 3.3.6) [14].

3.3.6 Elements Of Spiking Neural Networks

Taking a closer look at Equation 3.34, particularly the $(1-\beta)I_{\text{in}}(t)$ term, $I_{\text{in}}(t)$ can be viewed as a spike (or input voltage) that is scaled by a synaptic conductance $(1-\beta)$ to create an injection of current into the neuron. In calling such a spike $X(t)$ and naming the synaptic conductance to be variable W gives rise to Equation 3.45.

$$WX(t) = (1-\beta)I_{\text{in}}(t) \tag{3.45}$$

Interpreting $X(t)$ as the input spikes from a different layer and W as the corresponding weight for each input spike such implementation of parameters in Equation 3.34 leads to Equation 3.46 that allows for the construction of deep spiking neural network learning models.

$$U(t+\Delta t) = \beta U(t) + WX(t+1) \tag{3.46}$$

It's important to note that the effects of β and W are decoupled as W is a learnable parameter that is updated independently of β [14].

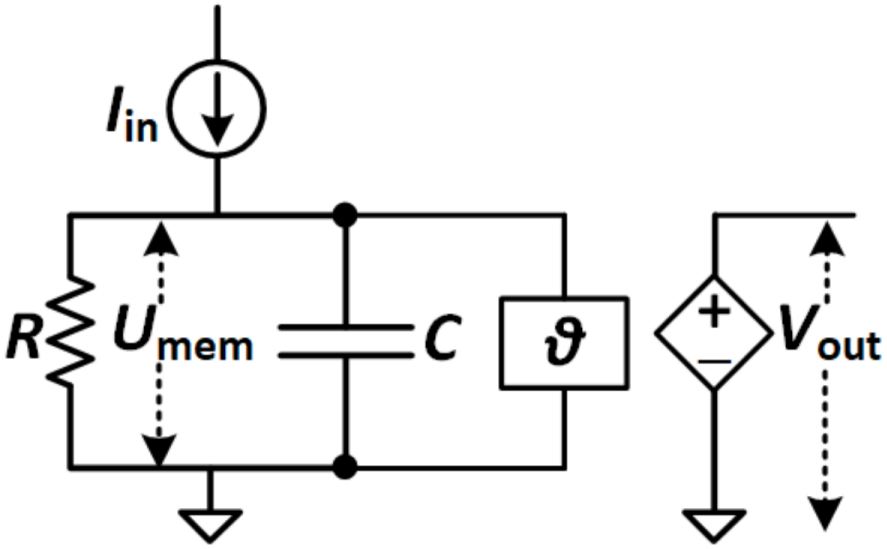


Figure 3.16: This figure is the same figure as Figure 3.15 except the threshold (θ) has been introduced where if $U_{\text{mem}} > \theta$ then V_{out} creates a voltage spike and sends it to the next layer [14].

Recall that if the membrane potential exceeds a given potential threshold that the neuron will output a spike as given by Equation 3.47, where $U_{\text{thr}} (= U_{\text{threshold}})$ is the potential threshold.

$$S(t) = \begin{cases} 1, & \text{if } U(t) > U_{\text{thr}} \\ 0, & \text{otherwise} \end{cases} \tag{3.47}$$

An alternative representation for Equation 3.47 is Equation 3.48 where Θ is the Heaviside (step) function.

$$S(t) = \Theta(U(t) - U_{\text{thr}}) \tag{3.48}$$

Therefore from Equation 3.47 (or Equation 3.48) if a spike is output from a neuron then the membrane potential is reset. In Equation 3.49 the defined reset mechanism is reset-by-subtraction (where the new potential after a spike is the original pre-spike potential minus the defined potential threshold value) and in Equation 3.50 the defined reset mechanism is reset to zero (where the potential, regardless of any excess, defaults back to zero).

$$U(t+1) = \underbrace{\beta U(t)}_{\text{decay}} + \underbrace{WX(t+1)}_{\text{input}} - \underbrace{S(t)U_{\text{thr}}}_{\text{reset by subtraction}} \tag{3.49}$$

$$U(t+1) = \underbrace{\beta U(t)}_{\text{decay}} + \underbrace{WX(t+1)}_{\text{input}} - \underbrace{S(t)(\beta U(t) + WX(t+1))}_{\text{reset to zero}} \tag{3.50}$$

Note that U_{thr} is frequently equal to unity (though it can be tuned to different values) and W is a learnable parameter, which (traditionally) leaves β as a hyperparameter. Whether to use "reset by subtraction" or "reset to zero" is a hyperparameter.

Putting all of the above together, an example of how the network behaves is depicted in Figure 3.17 where incoming spikes create a potential (which exponentially decays in accordance with β), the potential slowly builds, and when the potential reaches a given threshold, θ, the neuron fires (outputs a spike) as depicted in Figure 3.17.

While other and more complicated spiking neuron models exist, they all follow this idea of taking incoming spiking neuron behavior as inputs which effects the neuron's internal voltage potential which may build over time as to elicit output spikes (e.g. Equation 3.48) and determine the electric potential at the next time step (e.g.

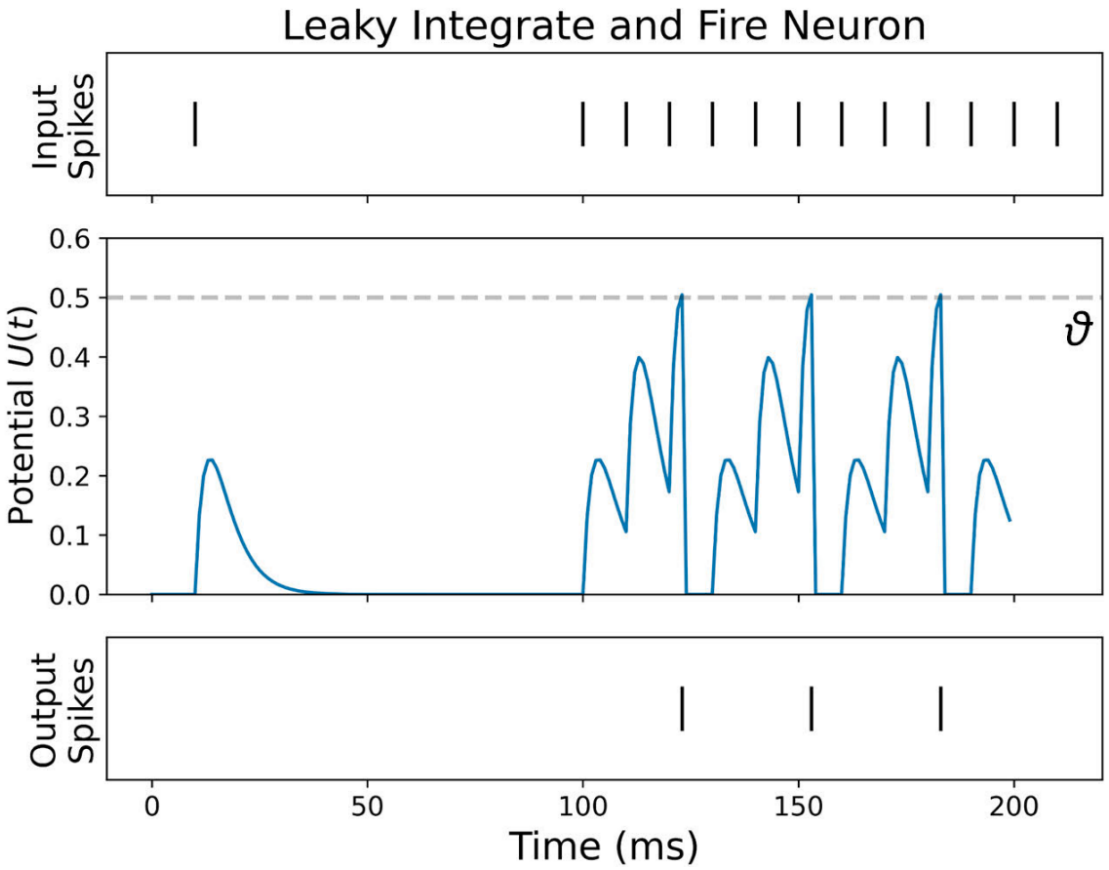


Figure 3.17: A depiction of an example leaky integrate and fire neuron with incoming spikes, the created potential, and the outgoing spikes [14].

Equation 3.49).

3.3.7 Surrogate Gradients

Equation 3.48 is useful for a spiking neural network forward pass but becomes problematic when attempting backpropagation, as the derivative of the Heaviside function is the Dirac delta function (with value of 0 everywhere except when $U_{\text{thr}} = \theta$ in which it's then ∞). This means that on the backward pass the gradient will be either

vanish or explode (when U is at the threshold) and therefore no learning is possible. These issues are collectively known as the dead neuron problem as seen in Figure 3.18.

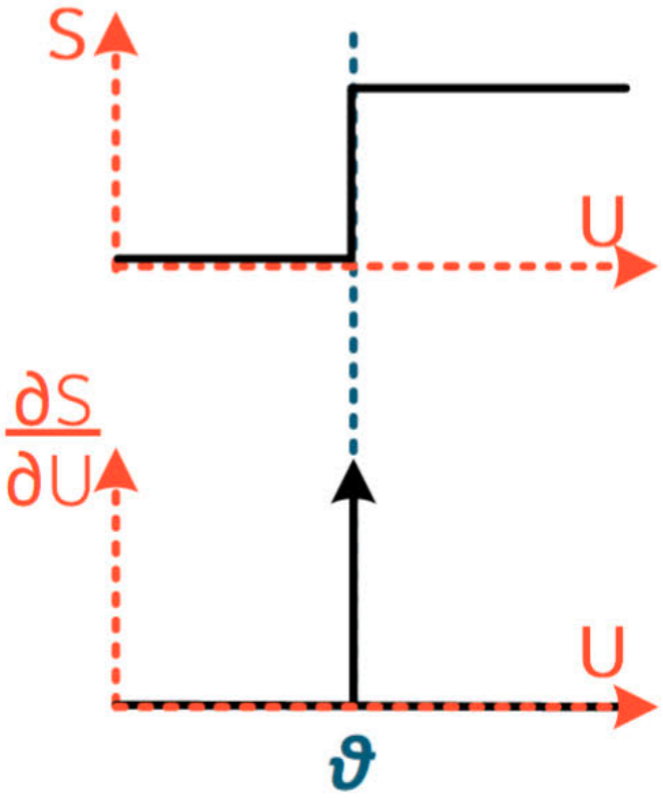


Figure 3.18: A depiction of a dead (non-smoothly differentiable) neuron which creates the dead neuron problem. Indeed, $\frac{\partial S}{\partial U} \in \{0, \infty\}$ [14], only ever resulting in a vanishing or exploding gradient.

The solution to this issue in spiking neural networks is known as the surrogate gradient approach (e.g. Figure 3.19) [14]. The idea is to take a smooth and continuous function that looks similar to the Heaviside function (e.g. arctangent, sigmoid, etc...), take the derivative of that function, and then use the derivative of that function on the backward pass. For example assuming that the arctangent function was of interest for

the surrogate gradient, then on the forward pass the Heaviside function (similar to the one used in Equation 3.48) is used and on the backward pass $\frac{\partial \arctan(x)}{\partial x}$ is used in place of $\frac{\partial \Theta(x)}{\partial x}$. Which surrogate gradient function to use is a hyperparameter.

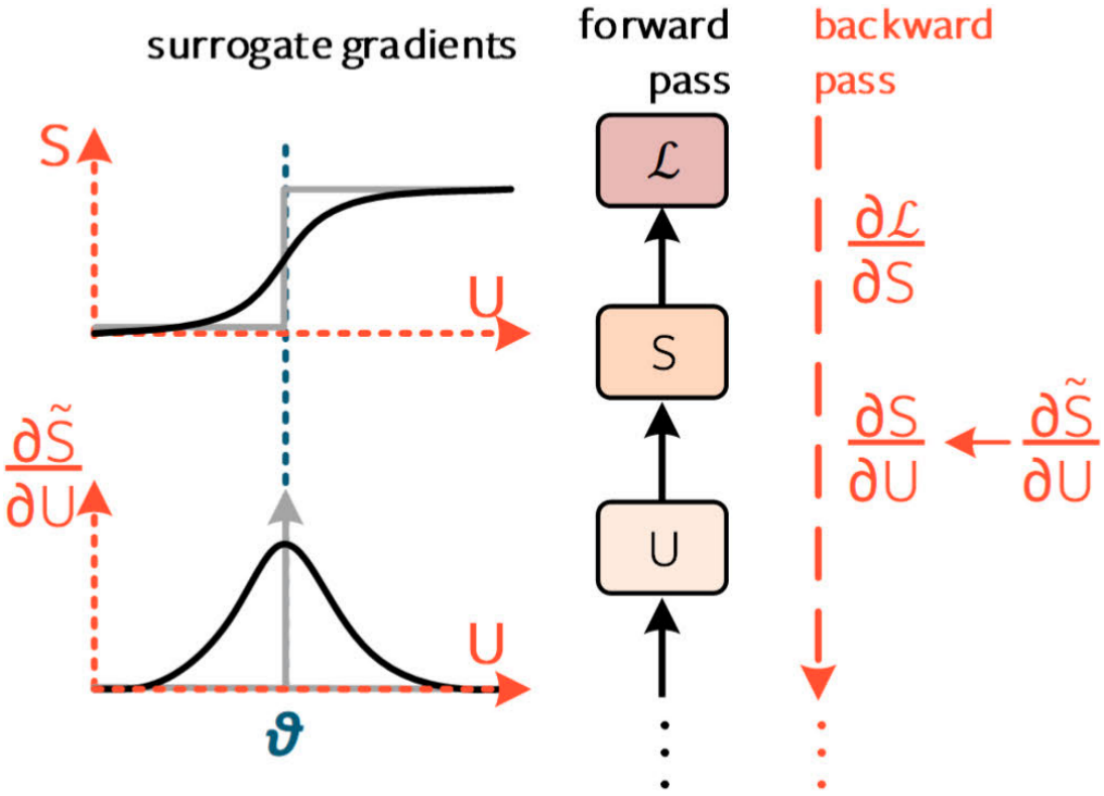


Figure 3.19: A depiction of a surrogate gradient approach, where $\frac{\partial \widetilde{S}}{\partial U}$ takes the place of $\frac{\partial S}{\partial U}$ in the backward pass step [14].

3.3.8 Motivations For Spiking Neural Networks

In classical machine learning, multiplying high-precision values with high-precision weights, passing high-precision values through high-precision activation functions, executing such high-precision operations on hardware, and copying high-precision

values to other model layers and components is computationally expensive and slow. However the spike-based approach simply multiples the high-precision weight with a spike of value "1", replacing the high-precision multiplication operations with a simple read-out of the weight value [14].

While spiking neural networks output a single bit based on whether they spiked or not, they are different from other binarized networks since spiking neural networks rely on the timing of the spikes, which can easily utilize the clock signals on digital circuits [14].

Due to spiking neural networks only outputting spikes (or lack thereof), then the resulting tensors from spiking neurons are sparse. For example, when examining the following list $[0, 0, 1, 0, 1, 0, 0, 0]$ it can be seen that since all of the entries are zero but two of them, then only passing the location of nonzero elements of the list (e.g. when indexing from zero, there is the value of 1 at positions 2 and 4) yields a more efficient representation than passing all values at all entries. Recalling that the space needed by a simple data structure to store a matrix increases with the number of contained entries, therefore as the sparsity of data increases so does the space that can be saved and such efficient representations makes sparse data (and thereby data from spiking neural networks) cheaper to store [14].

Spiking neural networks may receive data from event cameras which report changes in brightness only on activated camera pixels, otherwise such pixels remain quiet and unreported (as depicted in Figure 3.20), resulting in each pixel activating and operating independently of all other pixels (eg: unlike conventional RGB cameras

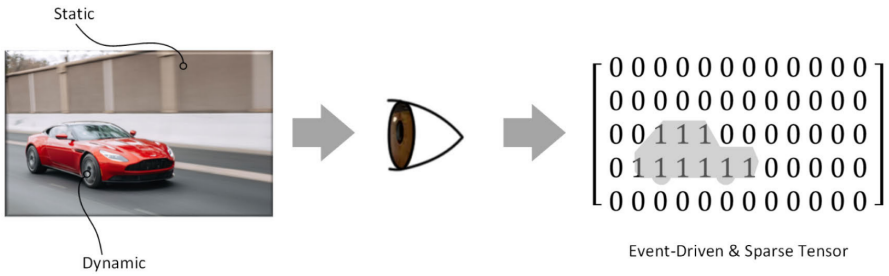


Figure 3.20: A depiction of an event camera creating a sparse matrix from a detected car moving in the camera's frame [14].

which collect data on all the pixels simultaneously after a set exposure time, regardless of incoming light or discrimination to static input). This event-driven processing leads to a reduction of active pixels and thereby vast energy savings compared to traditional image cameras. The reduction in active collected pixels also results in asynchronous and low-power operation allowing for fast clock speeds with microsecond temporal resolution. Such difference in "image" capture between conventional (frame-based) cameras and event cameras can be seen in Figure 3.21 [14].

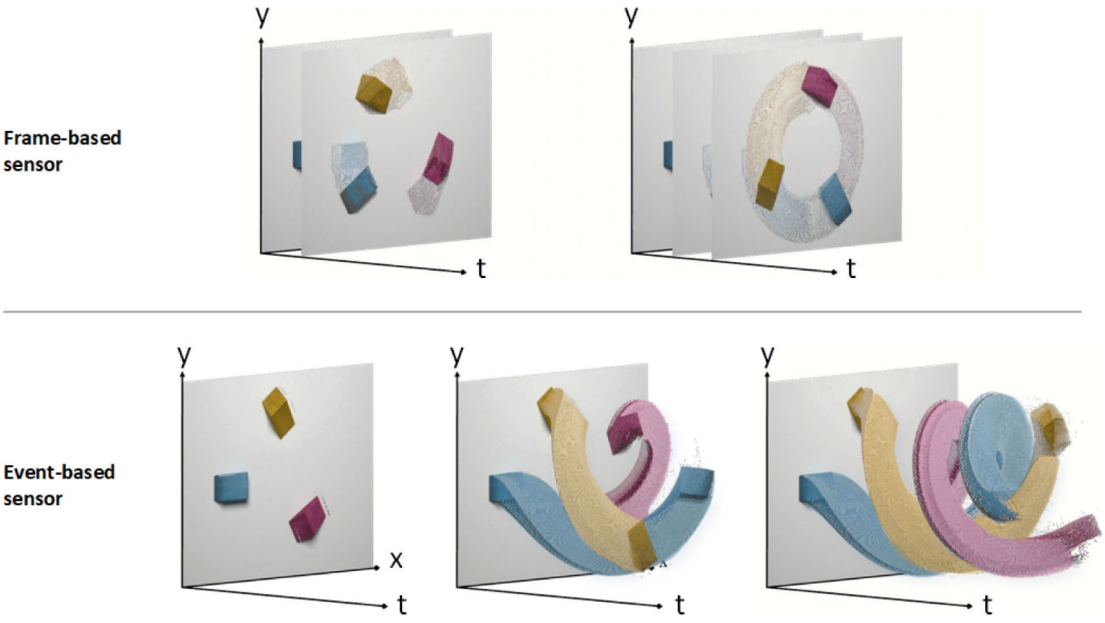


Figure 3.21: A depiction of an event camera recording an "image" in time versus a conventional camera recording the the same image [14].

Part III

New Ideas And Applications

Chapter 4

Classical And Quantum Self-Attention

4.1 The Rise Of Self-Attention Architectures

Much[1] of the rise in performance and power behind modern day machine learning revolves around transformers and by extension self-attention. Made famous by the 2017 paper *Attention Is All You Need* [65], transformers and self-attention have revolutionized Natural Language Processing as such architectures have formed the bedrock for GPT-3 [58], LLaMa [66], and others [67–73]. The idea of Vision Transformers was introduced in the 2020 paper *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* [74] which applied transformers to images. With the onset of Vision Transformers, transformers can be used for computer vision tasks such as classification [74] and object detection [75].

At a very high level, self-attention takes inputs (in tensor form) and multiplies those inputs by three separate learned matrices to generate unique "q" (query), "k"

[1] Nearly the entirety of this chapter and its sections were originally published in [16].

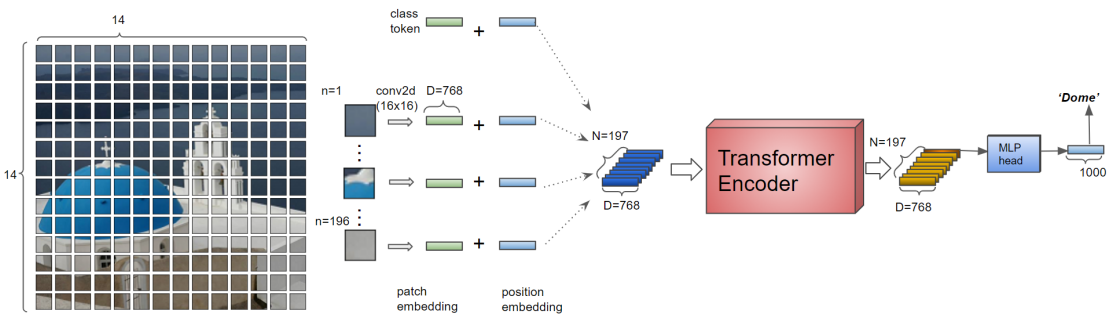


Figure 4.1: A depiction of classical vision transformer with example values. The "MLP head" is the multilayer perceptron network that is at the top (head) of the network [15].

(key), and "v" (value) matrices for each input[2]. Query and key matrix elements are multiplied together and then passed to a softmax operation to create the attention matrix, and the attention matrix is then multiplied by the value matrix elements. All of the resulting outputs are then concatenated and output together. The query matrix is then shared and used in the same operation with every input generated key and value matrix that the vision transformer takes in [65]; it is this sharing of the query matrix between the other separate key and query matrices from which self-attention gets its power and performance. A vision transformer for classification can be seen in Figure 4.1 with its self-attention mechanism (as embedded in the Transformer Encoder) being depicted with example values in Figure 4.2.

As Transformers expand to different variations and applications, such architecture has inspired and motivated the development of Quantum-enhanced Vision Trans-

[2]The terms query, key, and value come from the days of retrieval systems when a search engine would map the Query (e.g. the text in a search bar) against the Keys (e.g. given descriptors like video title, description, etc...) of indexed items, and then the search engine would return the best matched items (Values) to the user.

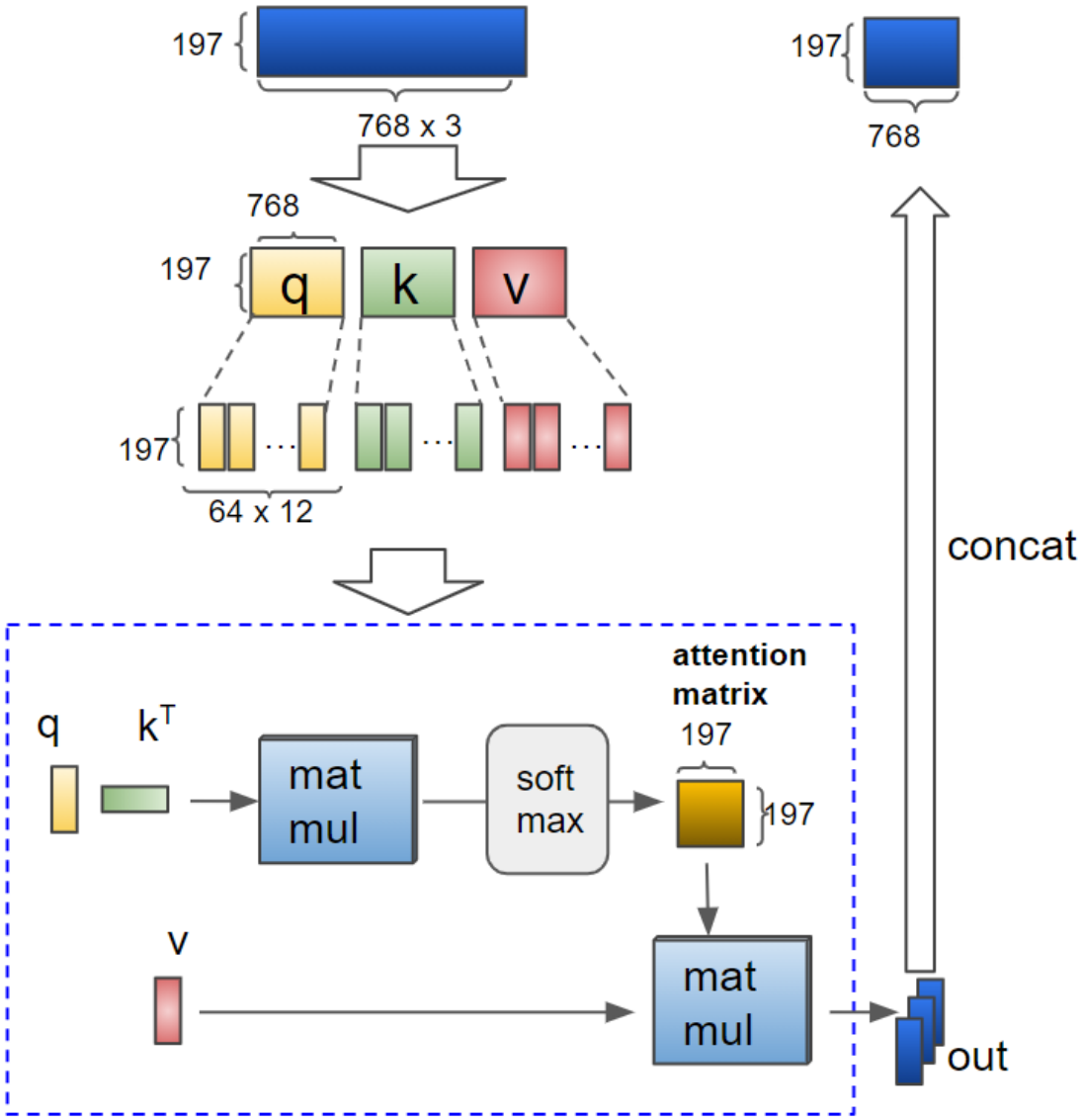


Figure 4.2: A depiction of classical self-attention with example values as implemented in the Transformer Encoder in Figure 4.1. Here "q," "k," and "v," are the generated query, key, and value tensors, respectively, "matmul" is matrix multiplication, "softmax" is the well known softmax function, and "concat" is the concatenation of outputs. [15].

formers (QViT) [16]. The QViT replaces classical self-attention with a version of quantum self-attention, and such QViT design resolves issues with past implementations and whose performance is compared against a classical ViT [16].

4.2 Previous Quantum Self-Attention Architectures

Previous works attempted to implement quantum (self) attention with various levels of performance [68, 76]. However such implementations leave room for improvements which are outlined below and set the stage for the novel QViT implementation detailed in [16].

Attempts at applying attention to quantum vision transformer networks include [76], however while such work employs attention it lacks self-attention. Such self-attention is essential for comparing encoded inputs across multiple other inputs, however [76] makes no mention of the encoded query, key, or value parts of the input, and does not compute the query and the key parts of the input together.

Another drawback of previous works for quantum (self) attention in vision transformers is that they don't consider the sizes for the inputs of the attention mechanism to be an adjustable hyperparameter. This hyperparameter allows for the user to tune the projected space that is input to the (self) attention mechanism, and such tuning can be done on an case-by-case basis for purposes of biasing an input to have more of an effect than another input. For example it may be advantageous to give the query vector more influence than the key vector when they are combined, and likewise

it might be more advantageous to give the attention vector more weight than the value vector. With past approaches such inputs were all treated equally and gave no room or operation to do so otherwise. It's notable that classical vision transformers don't typically consider this flexibility either as the projected sizes for their query, key, and value components are traditionally a function of the input, however given that the self-attention mechanism is now being dealt in a quantum mechanical manner, it's worth considering and fine-tuning the flexibility for the size of the inputs to the quantum mechanical self-attention mechanism.

Also the approach proposed in [16] is hardware agnostic as one could tune the inputs to the quantum circuit to fit the available input size to the applied quantum hardware of interest. If using a simulated approach this ability to tune the size of the inputs as well as their proportions allow for effective use of the simulated quantum circuit, since too large of an input would slow down the simulation while too small of an input would not capture the full effects of the input in a reduced dimension. In contrast to the above motivation and points for a quantum vision transformer, the paper [76] passed the full width of the input to their quantum circuits without regard to current, future, or available quantum circuitry, real or simulated.

Other work addressing the quantum self-attention in a transformer architecture includes [68] which more closely resembles a true transformer structure with some interesting features. [68] divides the inputs into three parts and then entangles the query and key inputs via a CNOT gate before the self-attention is conducted; combining the query and key inputs constitutes creating some type of attention between the query and

key. However the query, key, and value matrices should remain separate and independent until after they've been combined for self-attention.

The technique proposed in [68] also indicates that at the end of their quantum circuit the previous inputs are summed with the product of the proposed attention and value steps. This incorporation of the input to the output is more reminiscent of a UNet [77] than a transformer in which such an operation is absent.

[68] indicates that the query and key components of self-attention are coupled together, and then after measurements are taken for their respective values the query and keys are subtracted from one another and then put through a softmax (of sorts) of these differences. This approach creates noise in the output when evaluating the difference of queries being compared against keys from the same input, thus a difference wouldn't yield how much self-attention an input should give to itself since the items being compared are already entangled from previous operations.

Despite the aforementioned works, the proposed approach resolves the above points with its own novel implementation for quantum-enhanced self-attention vision transformers.

4.3 Methodology

The most simple approach to create a novel QViT simultaneously addressing the issues above while competing with a ViT [74] would be to replace the dot-product attention inside the ViT with a trained variational quantum circuit.

The following subsections will discuss how Quantum Self-Attention was implemented within the ViT. Variations and their motivations will be addressed in later sections. Unless otherwise noted, the QViT has identical default settings and parameters as the classical ViT. The performance of the QViT will then be benchmarked against its classical ViT counterpart.

4.3.1 Quantum Self-Attention

[16] focused on developing Quantum Self-Attention within the ViT. However since self-attention within the ViT relies on dot-product attention, the attention is replaced with a classical-quantum hybrid architecture (Figure 4.3) which when inserted into the ViT architecture becomes Quantum Self-Attention within the ViT.

To begin, assume that the tensors for the query, key, and value for attention are given with their encoding masks already applied and the scaling factor is applied to the query and key.

4.3.2 Creating The Attention Mask

To create the quantum attention mask, the query and key pass through a classical linear layer to project their respective size down to five nodes. The resulting query and key vectors are concatenated together to form a single vector which is then passed to a hyperbolic tangent activation function, whose output is multiplied by a scalar, α. The vector is then passed to the quantum circuit where it undergoes an R_y rotational encoding step (as defined by Equation 2.11) for 10 wires, after which

the quantum circuit applies four iterations of PennyLane's StronglyEntanglingLayers (with a single iteration consisting of single generalized qubit rotations on each wire and CNOT gates connecting pairs of wires across the entire circuit) [78]. The quantum circuit measures the expectation value of the Pauli Z matrix for each of the wires of the circuit. The resulting measurements get sent through a softmax layer which generates the attention vector. These steps can be seen in the top half of Figure 4.3.

4.3.3 Generating The Output

Creating the output with the attention mask (vector) and value is nearly the same process as described in Section 4.3.2, with the exception that there is no softmax operation.

To create the output, the attention vector and the value matrix pass through a classical linear layer to project their respective sizes down to five nodes. Then the attention and value are concatenated together to form a single vector which is then passed to a hyperbolic tangent activation function, whose output is multiplied by a scalar, α. The vector is then passed to the quantum circuit where it undergoes an R_y rotational encoding step for 10 wires, after which the quantum circuit then applies four layers of PennyLane's StronglyEntanglingLayers [78]. The quantum circuit then measures the expectation value of the Pauli Z matrix for each of the wires of the circuit. The resulting measurements form the output of the entire attention mechanism. The above creation of the aforementioned output can be seen in the bottom half of Figure 4.3.

After the output is generated, it passes through a classical linear layer to

project back into the appropriate number of dimensions to continue being passed to the rest of the QViT.

4.4 Software, Data, Models & Experiment

4.4.1 Software and Data

The MNIST digits dataset [79] is used as a classification task to compare the performance of the ViT and proposed QViT.

PyTorch [80] was used to supply the MNIST dataset which was brought in via the EMNIST dataset [81] with the MNIST split. The data was then normalized with a mean of 0.1307 and a standard deviation of 0.3081 over the only channel. The training and test datasets were defined by their respective default files as called by PyTorch.

The ViT was taken from the PyTorch implementation of the Vision Transformer [82]. The defined architecture variables were taken to be the following: image size of 28×28 pixels; patch size of seven pixels; one layer; two heads; a latent (hidden) dimension of 128; multilayer perceptron dimension of 128; one channel; and 10 classes.

ADAM was the chosen optimizer and Cross Entropy was the loss function.

PennyLane [83] was used for simulating and training the quantum circuits. All functions for encoding data into the quantum circuits and constructing the quantum circuit learned parameters were done with the PennyLane API.

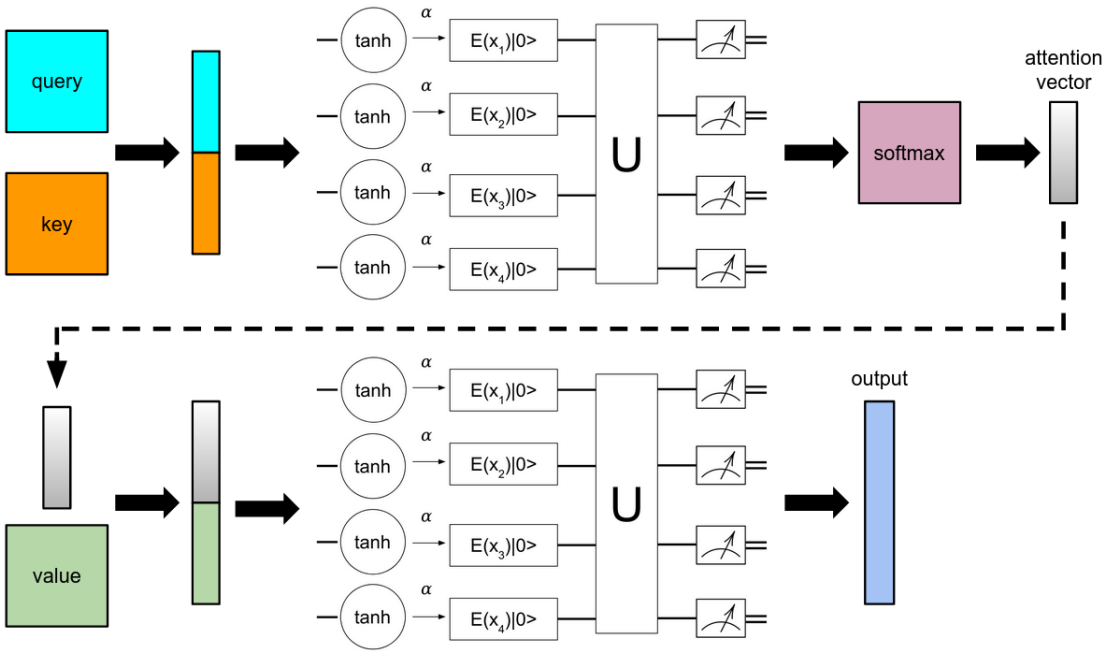


Figure 4.3: This figure depicts the flow of information through quantum attention, though the circuit parameters can be found in Section 4.3.3. The query and key matrices are flattened into vectors. The vectors are then concatenated together and are passed through a hyperbolic tangent activation function, whose outputs are then multiplied by a scalar, α. The inputs enter the quantum circuit and are encoded via an encoding method, $E(x_i)$ (where x_i is an input to the quantum circuit. Note that the illustration stops at x_4 for brevity). The above process is repeated for the value matrix whose resulting vector is concatenated with the attention vector, and the result after the imposed variational quantum circuit forms the output for Quantum Self-Attention.

4.4.2 Models

Constructing the QViT begins with the ViT. Within the ViT, the dot-product attention is exchanged with the Quantum Self-Attention mechanism described in Section 4.3 and illustrated in Figure 4.3. After the output is generated, a linear layer is used to increase the output dimensions to the shape the output would have been if classical self-attention was used instead of Quantum Self-Attention.

α was chosen with regards to the rotational encoding step, R_y. Consider that since the classical information passes through a hyperbolic tangent activation function before reaching α, then all of the information will be projected to span from -1 to 1. Therefore one value for α was chosen to be $\alpha = \frac{\pi}{2}$; this is an arbitrary choice, with a careful eye for the projected space of the inputs to be not too big but not too small either, as the projected data will span from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ when passed to R_y. The other value for α was chosen to be $\alpha = \pi - 0.01$ to ensure that all encoded data remained unique after being passed to the R_y rotational encoding step and did not overlap at $\alpha = \pm\pi$.

There is a question of dimensionality when the output of the softmax operation as seen in Figure 4.3 should or should not match the dimensions of the value vector. In some models (as addressed below) the attention vector passes through a linear layer to scale up to the equivalent shape of the value vector, and then both the attention vector and the value vector are scaled down with the help of a linear layer to five nodes, after which the resulting vectors are concatenated together and passed to the quantum

circuit. In other models the attention vector passes through a linear layer that scales the vector down to five nodes, and then a separate linear layer scales the value vector down to five nodes. After this the vectors are concatenated together and passed through the quantum circuit. The scaling tests to see if the initial projected dimensional space of the attention and value vectors change the performance when they're scaled down; this will be discussed further in Section 4.5.

From the above, four models are explored with the following titles and parameters:

- *QViT With* $\frac{\pi}{2}$ *Encoding*
  - the attention vector is rescaled to the same shape as the value vector with the help of a classical linear layer, and $\alpha = \frac{\pi}{2}$
- *QViT With* $\frac{\pi}{2}$ *Encoding And No Rescaling*
  - the attention vector is scaled down with the help of a classical linear layer, and $\alpha = \frac{\pi}{2}$
- *QViT With* $\pi - 0.01$ *Encoding*
  - the attention vector is rescaled to the same shape as the value vector with the help of a classical linear layer, and $\alpha = \pi - 0.01$
- *QViT With* $\pi - 0.01$ *Encoding And No Rescaling*
  - the attention vector is scaled down with the help of a classical linear layer, and $\alpha = \pi - 0.01$

4.5 Results And Discussion

As seen in Figure 4.4, Figure 4.5, and Figure 4.6, when comparing the test and training statistics of the QViT and the (classical) ViT, it is found that the QViT and classical ViT are of comparable performance. It can be argued that the QViT gives a slight boost in performance when examining the test accuracy, however upon closer inspection of the y-axis' scale it can be immediately rendered that such differences do not amount to any statistical significance [16].

A similar result to the above analysis can also be found with the test loss. While the test loss for the QViT models on average outperform the classical ViT, such differences are minute when examining their scale via the y-axis.

The training loss tells the above story in perhaps the most stark terms; by inspection the training loss yields no statistical difference between the classical ViT and the QViT.

What is interesting is that varying α or the projected dimensions for the attention mask do not change the behavior of the QViT. It may very well be that the imposed quantum circuit allows for the resulting attention vector to be in a sufficiently separate dimensional space such that any extra alteration yields no advantage over the ViT or difference between the presented projected dimensions.

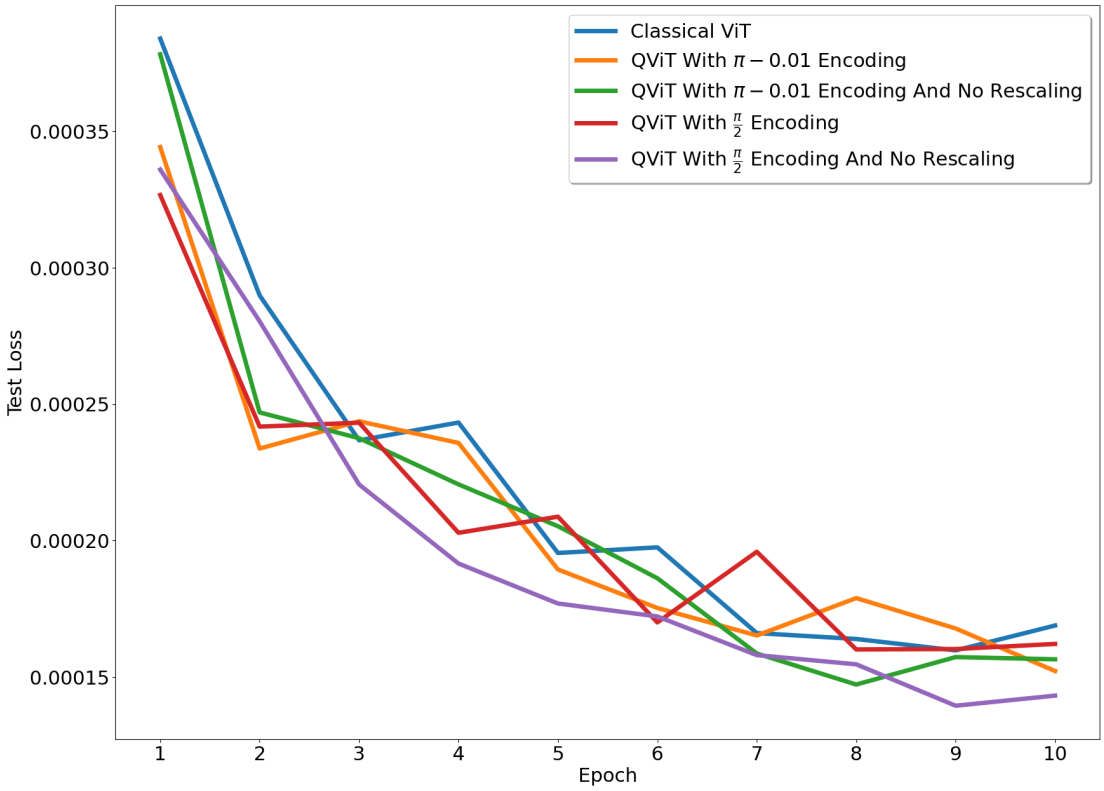


Figure 4.4: A plot depicting the test loss of all the different proposed classical and quantum self-attention models [16].

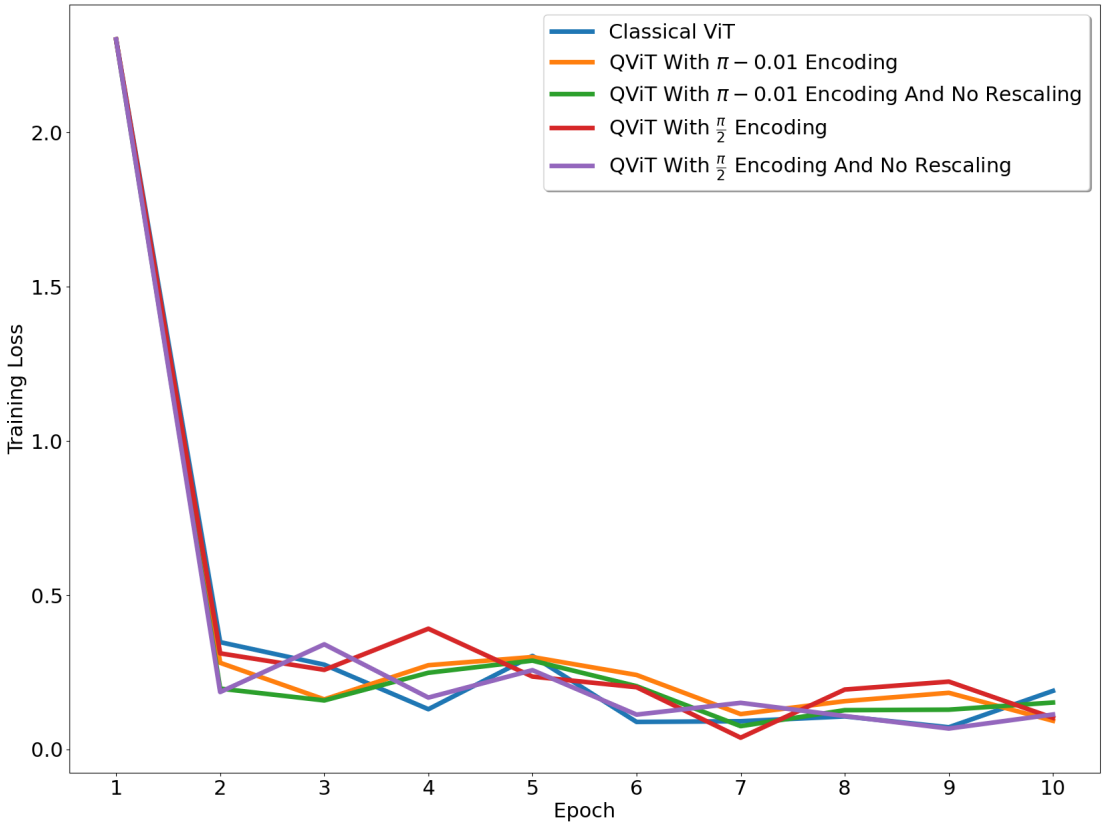


Figure 4.5: A plot depicting the training loss of all the different proposed classical and quantum self-attention models [16].

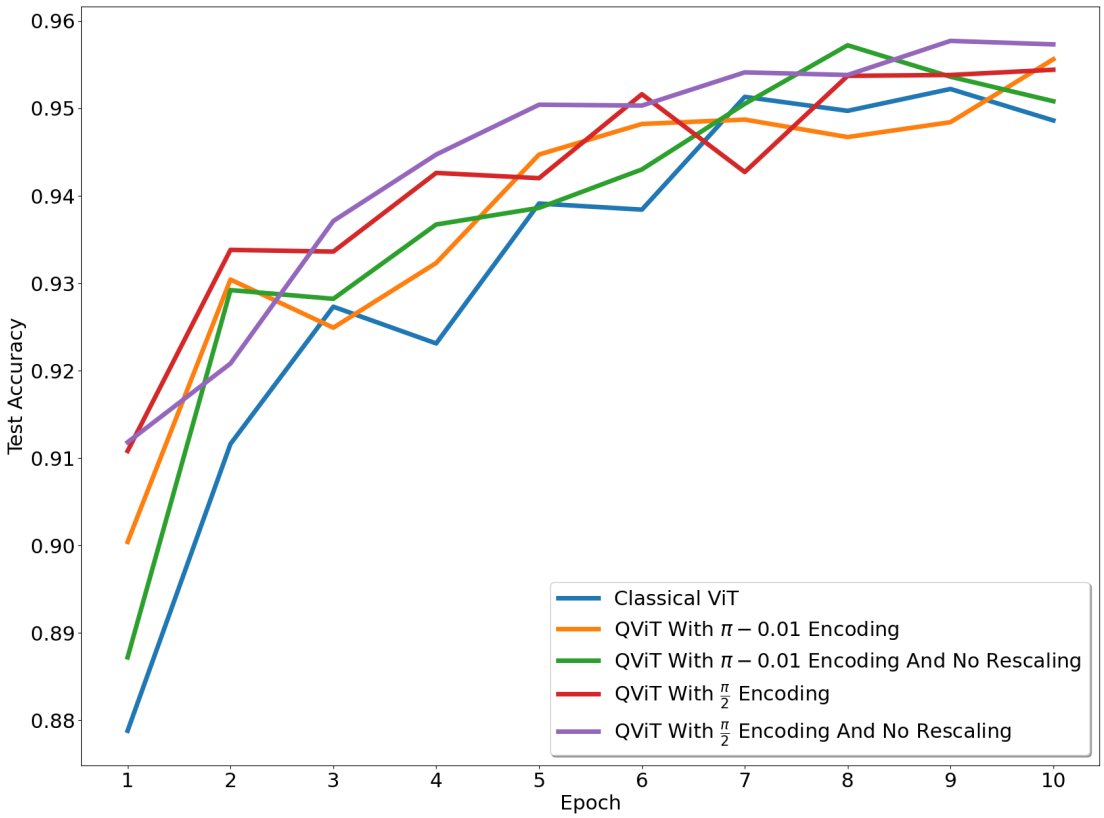


Figure 4.6: A plot depicting the test accuracy of all the different proposed classical and quantum self-attention models [16].

4.6 Remarks

As inspired by modern day Vision Transformers, [16] proposes a Quantum-enhanced Vision Transformer in which a novel approach to Quantum Self-Attention is introduced and implemented [16]. The performance of the QViT is benchmarked with the MNIST digits dataset and whose performance is compared against a classical ViT. Careful steps are taken to ensure that the encoding of information from classical to quantum maximizes the possible encoding space while also preventing an overlap in data encoding, as discussed in Section 4.4.2. Upon examining the models and their resulting statistics after training and testing on the MNIST dataset, it is found that a naive and simple implementation of a QViT is of comparable and competitive performance with the classical ViT.

Potential future work includes changing the encoding method of classical to quantum information, as well as varying the quantum circuit architecture. Also changing the ratio of the information contributed by the query and key vectors, and the value and attention vectors in their respective operations may very well alter the resulting performance.

Chapter 5

Learning Unimodular Matrices For Quantum Circuits

5.1 Introduction

As[1] quantum machine learning remains an active research area [83, 84] there exists the open question of matching the best quantum machine learning architecture to a particular problem which has led to the development of Quantum Neural Networks [85, 86], Quantum Born Machines [87, 88], transfer learning [89], Quantum Convolution Neural Networks [90], and Quantum Generative Networks [43, 91].

Despite all of the above proposed and used techniques, finding a generalized approach for determining the quantum circuit architecture for a given problem remains an open question. Others works have studied this question with a variety of techniques and differing levels of success.

[1] Much of this chapter and its sections were originally published in [17].

5.2 Previous Work

[92] learns the rotations of randomly generated layers of parameterized Rx and Ry gates. In [93] a genetic algorithm was used to tune the voltage of lasers to determine the ideal net unitary transformation. Then [94] used deep reinforcement learning to try to find the ideal architecture of single qubit rotations about the x, y, and z axis and CNOT gates. However none of these approaches attempted to first solve for a general numerically-readout unitary matrix as a whole.

Methods in [17] presented here yield a resolution to these issues by using general parameterized unimodular matrices to learn unitary matrices spanning both single and multiple wires of a quantum circuit. The result of [17] demonstrates the efficacy of using a unimodular matrix ansatz when compared to comparable classical models with similar architectures, and it's found that the parameterized unimodular ansatz achieves a higher accuracy faster than the comparable classical models.

5.3 Ansatz & Methodologies

5.3.1 Murnaghan's Recipe

Murnaghan gives a recipe to construct a parameterized $n \times n$ general unimodular and unitary matrix with latitudinal and longitudinal angles [95]. As an example, a 2×2 unimodular matrix can be decomposed into $U_1 = D \times U$, where U_1, D, and U are defined in Equation 5.3, Equation 5.1, Equation 5.2, respectively. ϕ is a longitudinal angle while α and β are latitude angles [95].

$$D = \begin{bmatrix} e^{i\alpha} & 0 \\ 0 & e^{-i\alpha} \end{bmatrix} \tag{5.1}$$

$$U = \begin{bmatrix} \cos(\phi) & -\sin(\phi)e^{-i(\alpha+\beta)} \\ \sin(\phi)e^{i(\alpha+\beta)} & \cos(\phi) \end{bmatrix} \tag{5.2}$$

$$U_1 = \begin{bmatrix} \cos(\phi)e^{i\alpha} & -\sin(\phi)e^{-i\beta} \\ \sin(\phi)e^{i\beta} & \cos(\phi)e^{-i\alpha} \end{bmatrix} \tag{5.3}$$

5.3.2 Quantum Machine Learning Architecture Ansatz

After constructing a parameterized unimodular matrix using the recipe in Section 5.3.1, the parameterized matrix or matrices can be used in a quantum circuit to learn the desired couplings and behavior of transformations on and/or between wires. However if learning the needed rotations on a single wire is desired, then a single 2×2 unimodular matrix can be implemented on that wire of interest.

5.3.3 Taking The Gradient

A variety of methods exist to take the gradient of a quantum circuit, and development of such methods remains an active research area. Common approaches to solving for the gradient include taking the gradient analytically [53,57]. In this study the central finite difference is used to find the gradient of the quantum circuit as defined in Equation 3.17 where f is an expectation value dependent on θ, a gate parameter [96,97]. It's notable that $\Delta\theta$ is a hyperparameter in Equation 3.17.

5.4 Experiment

5.4.1 Software, Dataset, Data Preparation

ScikitLearn's [98] two moons dataset was chosen as a classification example to characterize the performance of the proposed learned unimodular matrices. PennyLane [83] simulated the quantum circuit while PyTorch [80] conducted the forward pass and backpropagation mechanics through the model. The data includes 3200 training samples, 400 testing samples, and 400 validation samples. All of the aforementioned samples had a noise level of 5% with a batch size of 100 for each epoch. The ADAM optimizer with a learning rate of 0.01 was used as well as using Cross Entropy for the loss function. When implementing Equation 3.17 to solve for the gradient, a finite difference shift of $\Delta\theta = 1 \times 10^{-7}$ was used for each learned parameter of the unimodular matrices. In the utilized architectures and their corresponding implementations, a set seed was used for each experiment to control for repeatable results.

Four quantum-classical hybrid model classes are presented. To adequately compare against these models, a corresponding classical model was constructed which will be discussed in Section 5.4.6.

5.4.2 Class A

This class encompasses a hybrid classical-quantum model where a single parameterized 16×16 unimodular matrix is learned for all four wires. In the multiple matrix case a parameterized 2×2 unimodular matrix is learned for each wire in the

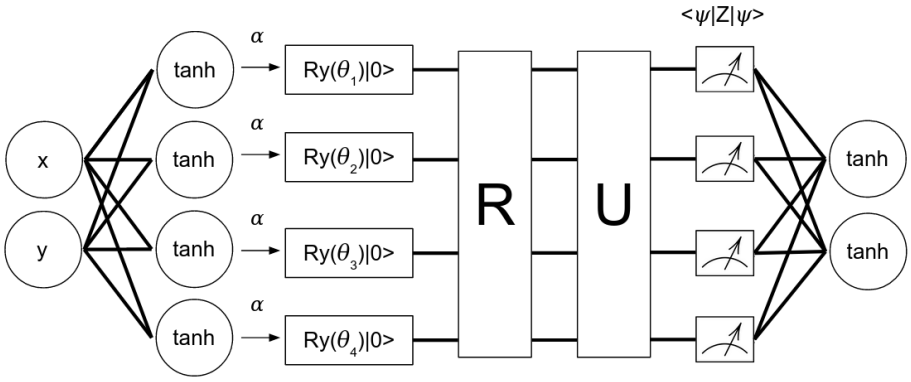


Figure 5.1: This figure is a hybrid quantum-classical model as discussed in Section 5.4.5. Here R is a random unitary matrix, U is a square unimodular matrix, and α is the learned scaling parameter. θ_n denotes that the n^{th} input from the previous linear layer is acting on the n^{th} wire; in this figure n ranges from $1-4$ [17]. © 2022, IEEE

quantum circuit.

In both the single and multiple matrix case as mentioned above, the inputs to the hybrid circuit first pass through a linear layer that goes from two nodes to four nodes. Then each of the outputs of the four nodes are passed to a hyperbolic tangent activation function (which projects the inputs to range from -1 to 1) and then the inputs are all multiplied by a scaling parameter[2], α, where in this class $\alpha = \frac{\pi}{2}$. After the scaling parameter acts on each node, the values from each node are passed to a quantum circuit where an R_y rotation (as seen in Equation 2.11) encodes the classical information. The values after the encoding rotation are passed to the aforementioned parameterized unimodular matrices. The quantum circuit then measures the expectation value of the Pauli Z matrix for each wire. The expectation values are then sent to four nodes which

[2]The reasoning for this can be found in the footnote in [99].

are sent to a linear layer whose output are two nodes. The values of these two nodes are acted on by a hyperbolic tangent activation function.

5.4.3 Class B

Class B is the same as Class A except the scaling parameter, α, is a learned parameter since the proposed and preset α from other model classes might not be the optimal value.

5.4.4 Class C

Class C is the same as Class A, with the addition of a random unitary matrix being applied for each learned unimodular matrix. In the case of the 16×16 unimodular matrix, a corresponding 16×16 random unitary matrix is applied across all of the circuit's wires before the learned 16×16 unimodular matrix is implemented (as seen in Figure 5.1). In the case of the individual unimodular matrices, a 2×2 random unitary matrix is applied to each wire of the quantum circuit preceding the 2×2 unimodular matrices. The random matrices are held constant for the duration of the models' training, testing, and validation.

The purpose of incorporating such random unitary matrix(ces) is to force the mapping of the inputs to the outputs in the quantum circuit to be in a more general complex space that is not solely unimodular.

5.4.5 Class D

Class D is a combination of Class B and Class C, i.e. Class C is used except the scaling parameter, α, is a learned parameter.

5.4.6 Class E

To fairly compare a classical model against the proposed hybrid models, in lieu of a quantum circuit, classical linear layers with four nodes per layer that run twelve[3] layers deep was chosen to keep the same dimensionality and approximately the same number of learned parameters as the quantum circuit.

Therefore this class is composed of purely classical models; in these models the inputs to the classical model first pass through a linear layer that goes from two nodes to four nodes. Then each of the outputs of the four nodes are passed to a hyperbolic tangent activation function. The values from each node are passed to twelve linear layers, each of which have four nodes, and on each node a non-linear activation function is enacted. After the twelfth linear layer, the outputs are then passed to a linear layer with four nodes which are then sent to a linear layer whose output are two nodes. The values of these two nodes are acted on by a hyperbolic tangent activation function. This model construction results in the head and tail of the model remaining identical to the hybrid model.

Since there exists a multitude of different activation functions which can be chosen and such a multitude can be placed in a large variety of different combina-

[3]Note that if thirteen linear layers were chosen instead there would be more learned parameters in the classical model than the number of learned parameters in the proposed learned unimodular matrices.

tions, the same non-linear activation function was used on each of the quantum-circuit equivalent layers of the classical models. The non-linear activation functions chosen were ReLU, LeakyReLU, PreLU, CELU, ELU, Mish, Sigmoid, and SiLU. While any non-linear activation function could have been chosen, for brevity only a collection of popular activation functions were selected.

5.5 Results And Discussion

Looking at the plots in Fig. 5.3, it is clear that learning the 16×16 unimodular matrix acting on all of the wires simultaneously trains at a faster rate, reaches a higher accuracy, and reduces to a lower loss than its multiple 2×2 unimodular matrix counterpart which acts on each wire individually. The best performing hybrid quantum-classical model (16×16 Unimodular Matrix with a Random Unitary Matrix and Learned Scaling Parameter) achieves about 82% test accuracy $9\times$ (times) faster than the best performing classical model (classical model with the ELU Activation Function). Such a finding is not surprising as the multiple 2×2 matrix approach does not learn the couplings between wires and suggests that such couplings aid the conducted training to progress towards a more desirable accuracy and loss.

The spikes in the 2×2 unimodular matrix accuracy plots occur when the learned scaling parameter reaches a "good but not optimal" value for α, after which α departs from such a value to accommodate the other learned parameters of the model as seen in Figure 5.3. This can be derived by observing that the spikes in Figure 5.3

occur when α is learned for the 2×2 matrices but the spikes vanish when α is a set value (in both cases the models have not reached a minimum loss and are still training).

What is interesting is the effect of learning α in Figure 5.3. For multiple 2×2 unimodular matrices, α correlates with the top model performer while the 16×16 single matrix models also achieved competitive accuracy but their rise to higher accuracy is slowed by learning α. A unique α may be needed for each individual wire, and this would especially be true for a single unimodular matrix on each wire learning the optimal rotations for each wire (it can be shown that a single wire unitary matrix operation can be expressed as three rotations around two axes [100]). This could be the cause for the delayed performance of the multiple 2×2 matrix model with random matrices and a trained α.

For the model classes which incorporate random matrices, the multiple 2×2 unimodular matrix model with random matrices and without a trained α appears to benefit early in the training as one of the earliest models to test well and reduce in loss the fastest compared to the other 2×2 unimodular matrix models as seen in Figure 5.3. However this isn't true when the 2×2 matrix model with random matrices simultaneously learns α as it hinders the model's performance as previously discussed. The single 16×16 unimodular matrix models with the random matrix did show improvement in the resulting accuracy and loss but only by a small (and statistically insignificant) amount. It can also be observed that a broader unitary space is explored with the help of the random unitary matrix than would otherwise be probed with a unimodular matrix alone. The one restriction is when the matrix is updated, only the unimodular component of

the net unitary matrix is altered.

Examining the scale of the loss in Figure 5.3 for each model also tells a consistent story with the above claims, namely that couplings between wires significantly affect each hybrid model's ability to properly learn the presented dataset. Therefore these couplings are required for achieving a more optimal accuracy and reduction in loss.

The ability for the unimodular matrix to contain many learned parameters in a single section of the model and thereby create a more shallow model may be a contributing factor as to why the unimodular matrix approach outperforms the classical models. Even the best performing classical models in Figure 5.3 appear to lag behind in their loss and therefore due to their depth take a while to achieve an equivalent accuracy of their hybrid peers (though this is more true for the single 16×16 unimodular matrix models and less true for the 2×2 unimodular matrix models due to the couplings between wires as discussed above).

Analyzing confusion matrices also gives insight as to how the models perform. Figure 5.2 takes the best performing hybrid quantum-classical model (16×16 Unimodular Matrix with a Random Unitary Matrix and Learned Scaling Parameter) and the best performing classical model (classical model with the ELU Activation Function) for the test dataset and finds the percent difference between their confusion matrices. Based on the percent difference between confusion matrices as found in Figure 5.2a, the hybrid model is 8.5% better at classifying the bottom moon in the two moons dataset than the classical model when the classical model first achieves about 82% test accuracy; both

models perform equally well in identifying the top moon. Towards the end of testing, both models achieve similar performance as seen in Figure 5.2b. This reinforces that the hybrid model with the block matrix approach not only performs just as well as the classical model, but also performs better in a classification category where the classical model still needs more training in order to reach a similar accuracy.

The above leads to many comments about traditional parameterized models. In the parameterized model approach, assumptions are made regarding the selection and placement of parameterized and non-parameterized gates and thereby the coupling (or lack thereof) between parameters at the quantum circuit layer. The selection and placement of parameterized gates (e.g. rotation gates) also makes assumptions about the needed rotations at given parts of the circuit as well as how such rotations should interact with other circuit operations. In a sense such placement of gates amounts to feature engineering within the circuit. However when a block parameterized matrix spanning multiple wires is used, such assumptions relax and learning the parameterizations of the wires is taken from a more general approach. What's interesting to consider is if both approaches are leveraged; indeed a practitioner is able to impose both the placement of desired gate couplings, rotations, and a learned block matrix spanning multiple wires. Such an approach is desirable in the case of imposing expert or domain knowledge to a dataset in the form of ad hoc gates while learning the remainder of the parameter space with a multiwire block matrix. It can be viewed that the block matrix approach starts from a general parameterization of the desired learned space and then specializes itself within the space, while with gates that are already specialized in their action and

placement their parameter updates only fine tune their parameterizations within the model.

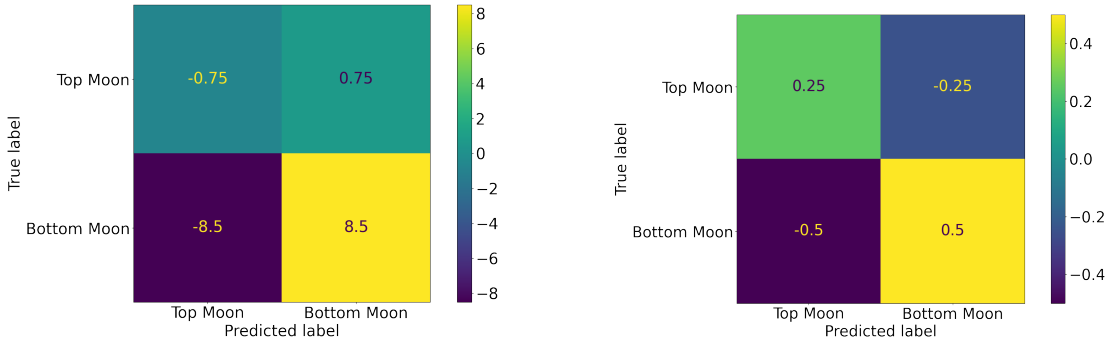
However when using the above proposed approach to parameterized models, various questions arise. Does the given parameterization for unimodular matrices generalize well for other kinds of datasets and architectures, or does it perform well for only this type of dataset? Would other parameterized block unitary matrices or other generalized approaches perform better than the discussed parameterized unimodular matrices? Would fine-tuning hyperparameters like the finite difference step size or learning rate, or learning a unique α for each wire also offer faster rises in accuracy and lowering the measured loss? These questions and issues would be interesting to explore in future work.

5.6 Conclusion

In the examined cases, using unimodular matrices as an ansatz for a quantum circuit achieves a higher accuracy faster than the equivalent classical models. While refining the hyperparameters may optimize this approach (e.g. changing the learning rate or the chosen optimizer), for a proof of concept the results appear promising for future work and development. It is also clear that using a singular square matrix spanning all of the wires to capture the couplings between wires is necessary as single unimodular matrices fail to do so. Exploring and benchmarking other unitary matrix parameterizations and other general approaches against a variety of standard datasets

would also be of interest.

Future work for benchmarking the process of learning a unimodular ansatz, deconstructing the ansatz into a series of fundamental gates, and then deploying those gates in a production environment on a real-time quantum computer would be of interest and application to both academia and industry alike.



(a) (b)

Figure 5.2: These figures present the percent difference between confusion matrices for the best performing hybrid quantum-classical model (16×16 Unimodular Matrix with Random Unitary Matrix and Learned Scaling Parameter) and the best performing classical model (classical model with ELU Activation Function) for the test dataset. Figure 5.2a evaluates the percent difference at the point where the ELU classical model first achieves $\sim 82\%$ accuracy, while Figure 5.2b evaluates the percent difference at the end of testing as seen in Figure 5.3. Positive values indicate where the hybrid model had a larger value than the classical model when the difference was calculated, while negative values indicate where the classical model had a larger value than the hybrid model when the difference was calculated. Figure 5.2a reveals that the hybrid and classical models perform equally well when identifying the top moon in the two moons test dataset, however the hybrid model does by far better than the classical model when identifying the bottom moon. Figure 5.2b shows that by the time the hybrid and classical models reach approximately the same overall accuracy they share nearly the same ability to identify the top and bottom moons [17].

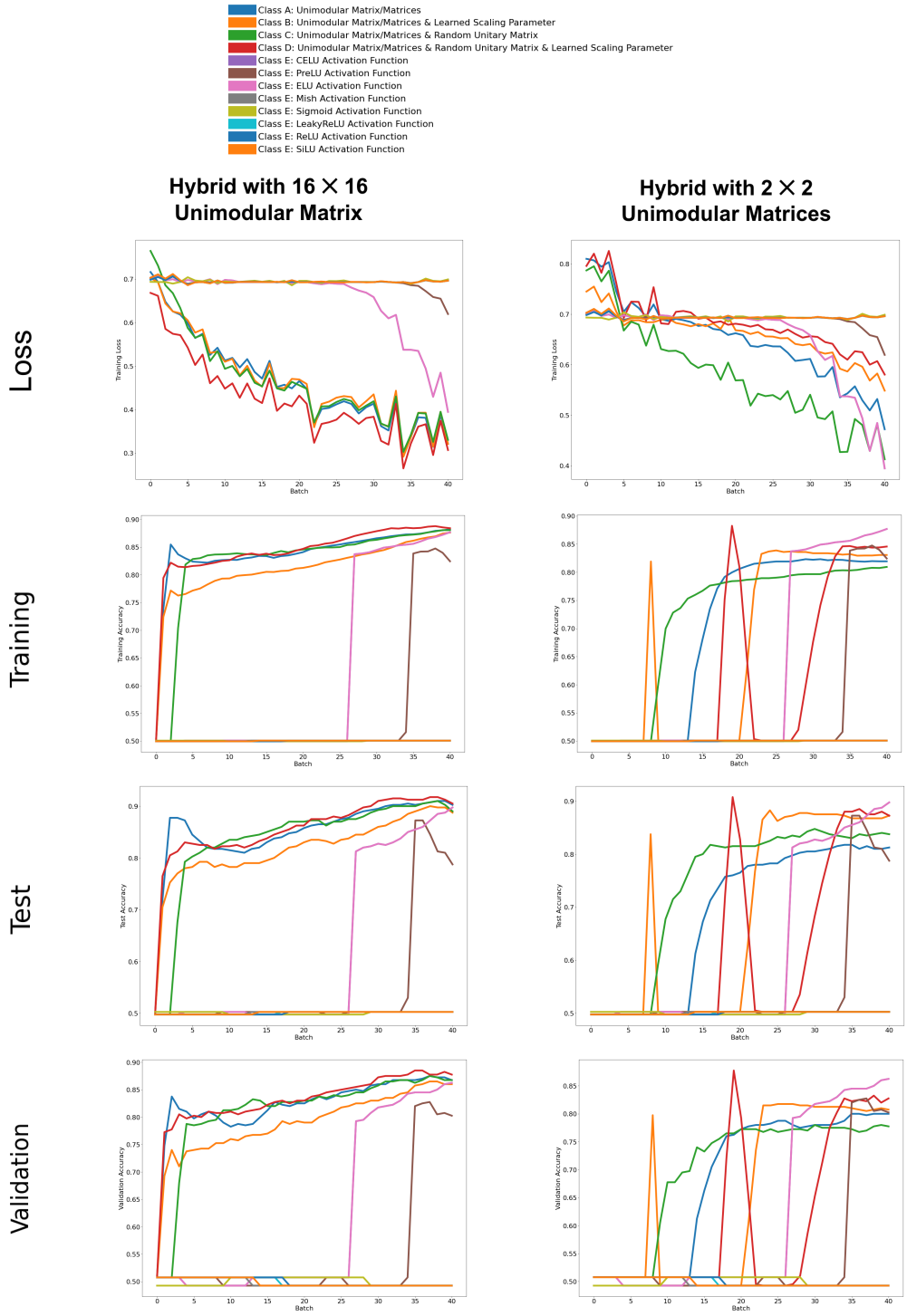


Figure 5.3: The loss and accuracy plots for the training, test, and validation datasets per batch for the hybrid and classical models (Sections 5.4.2 - 5.4.6) [17]. © 2022, IEEE

Chapter 6

Integrated Spiking And Quantum Neural Networks

6.1 Introduction

Often[1] referred to as the "3rd generation" of neural network models [101], neuromorphic computing methods are quickly becoming popular given their advantages over classical computing methods including better power efficiency and increased latency gains [14]. Not much work has been conducted to combine both spiking neural networks and quantum machine learning methods despite their unique advantages.

[1] Many parts of this chapter and its sections were originally published in [20] and [21].

6.2 Previous Work

Past works have combined quantum and spiking neural networks by applying the quantum method to the end of the network and using classical non-spiking classification methods with non-spiking loss functions [18, 19].

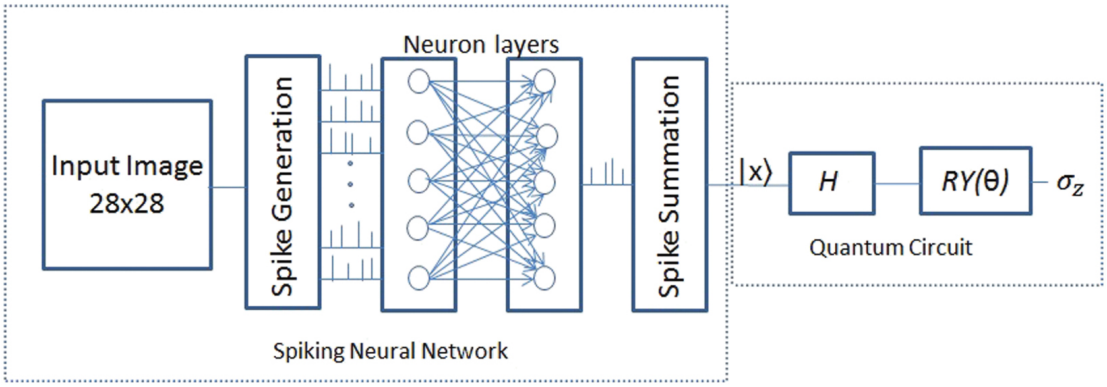


Figure 6.1: An architecture where the quantum network was added to the end of the network [18].

In one paper the quantum layer was added to the end of a spiking neural network to classify images where the model resulted in a continuous real-valued classical non-spiking output with a classical loss function [18] (as seen in Figure 6.1). Another study trained a fully spiking neural network for classification, after which they appended an untrained classical-quantum hybrid network with real, continuous, and non-spiking outputs to the backend of the model, and then retrained the network [19] as seen in Figure 6.2. A similar study to the previous one [19], was conducted by [102]. All methods apply quantum circuits towards the end of their models and conduct classification with non-spiking values and non-spiking loss functions.

The current literature in hybrid quantum-spiking neural networks is devoid of architectures or techniques where quantum machine learning is integrated within the body of the spiking neural network model as opposed to being appended to the end of a model. Recent studies have suggested that neural circuitry firing in the brain may be entangled [103–105], motivating the possibility of quantum effects in the brain and how such effects play a role in neural cognition. These effects only further encourage the study for integrating quantum behavior into neuromorphic systems including spiking neural networks as done in [21].

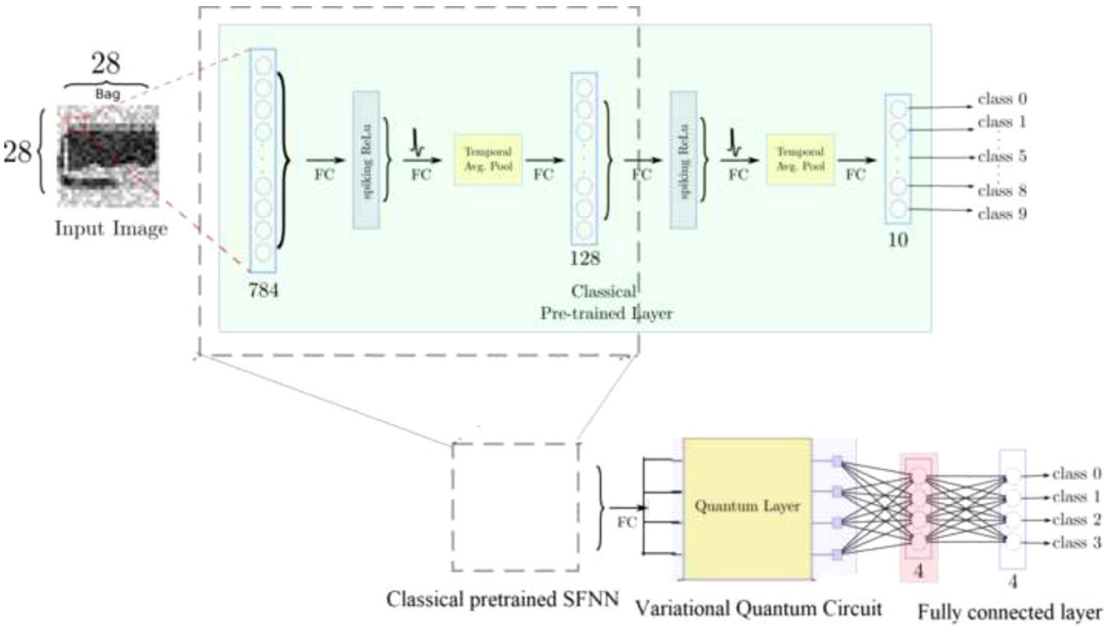


Figure 6.2: An instance where a pretrained classical network was appended to and trained with an untrained quantum network [19].

6.3 Methodology

Software usage included PennyLane [83] for simulating the quantum circuits, PyTorch [80] for the computation graph construction and backpropagation operations, and snnTorch [14] for simulating the spiking neural network.

For simplicity and demonstrating proof of concept, the image classes of 0 and 1 of the MNIST dataset (with each image having dimensions of 28×28 pixels) were converted into a spiking neuromorphic dataset (such that the pixel values were either 0 or 1). No further augmentations to the dataset were made. The default PyTorch training and test dataset split were used for the corresponding training and test datasets.

6.3.1 Model Design

The body of modern spiking neural networks are constructed with layers of alternating classical network components (e.g. dense linear layers, convolution layers, etc...) and a spiking neuron layer which, in traditional machine learning terms, acts like an activation function, where the "activation function" is the internal state of the simulated spiking neuron. If the spiking neuron reaches a certain threshold then the neuron fires (resulting in a spike represented by a 1 state) while if the neuron doesn't reach the desired threshold then the neuron doesn't fire (resulting in a neuron that doesn't spike which is represented as a 0, zero, state).

Given the above, there are two options to leverage when passing inputs to the quantum circuit: the first option is to insert quantum circuitry after a non-spiking layer

(e.g. a dense linear layer) and thereby capture classical values from traditional machine learning layers; and the second option is that quantum circuity is inserted after a spiking layer to capture the raw binary values (i.e. zeros and ones) coming from spiking neurons.

When passing the outputs from a quantum model, a similar situation arises as the outputs could be binary values (i.e. zeros and ones) replicating the output of a spiking neuron, or they can be continuous values (or in classical spiking neural network terms, these values would come from zeros or ones multiplied by some learned weight, with a learned bias term, resulting in a continuous values, i.e. weighted spiking values).

The internal state of the spiking neuron also plays a role. The neuron fires when the neuron's potential reaches a threshold (θ[2]). When the neuron fires, the potential can be either reset back to zero or have the threshold subtracted from the potential (such that any residual potential exists even after firing). Such a choice to either reset the potential back to zero or to subtract the threshold from the potential is a hyperparameter; in the entirety of this work the threshold is subtracted from the potential.

When a neuron receives input, that input is represented as an increase in the potential of the neuron, however such potential slowly starts to decay over time; let this decay be called the potential decay rate (β). Therefore the spiking network needs to balance the existing potential, its decay rate, and the incoming input to the system that informs the neuron whether to fire or not fire. The decay rate and exact values used for each model will be addressed and listed later in this work.

[2]Common quantum computing literature and textbooks alike use θ to represent a rotation on the Bloch sphere while spiking neural networks also use θ to represent the threshold for spiking neurons. The difference of when θ applies to which computing paradigm is so far only detectable by usage and context.

6.3.1.1 Model Inputs

Consider if the inputs to the quantum circuit are either values of 0 or 1. These 0 or 1 values can be encoded by multiplying the inputs by π and then encoding the result with a R_y (Equation 2.11) rotation (a rotation that would also work includes R_x, Equation 2.10, rotations which would also work) such that the resulting state will be either $|0\rangle$ or $|1\rangle$, respectively. This approach takes advantage of the two-state nature of the qubit and naturally lends itself to spiking values.

If the inputs to the quantum circuit are from a classical layer (e.g. linear dense layer), then the inputs can be encoded in any of the many compatible hybrid classical and quantum machine learning methods. In [21] such inputs are passed into a preprocessing step where the values are passed to a hyperbolic tangent (Tanh) activation function whose result is then multiplied by $\frac{\pi}{2}$ and passed to an R_y rotational encoding [106]. This step ensures the encoded inputs are mapped to each qubit's Bloch sphere representation with no information overlap.

6.3.1.2 Binary Outputs

If the quantum circuit needs to output binary values, then Equation 6.1 or Equation 6.2 [21] will suffice to yield such values (with such equations acting on each individual qubit).

$$\lceil -\langle\psi|Z|\psi\rangle \rceil = \begin{cases} 0, & \text{if } Prob(|0\rangle) \geq Prob(|1\rangle) \\ 1, & \text{if } Prob(|1\rangle) > Prob(|0\rangle) \end{cases} \tag{6.1}$$

$$\lfloor -\langle\psi|Z|\psi\rangle + 1 \rfloor = \begin{cases} 0, & \text{if } Prob(|0\rangle) > Prob(|1\rangle) \\ 1, & \text{if } Prob(|1\rangle) \geq Prob(|0\rangle) \end{cases} \tag{6.2}$$

6.3.1.3 Surrogate Gradients For The Ceiling Function

Note that the ceiling (Equation 6.1) or floor (Equation 6.2) approaches covered in Section 6.3.1.2 are stepwise and therefore non-differentiable. Borrowing a common trick from spiking neural nets [107, 108], a surrogate gradient is used in place of the gradient for the ceiling and floor function (whose derivative is zero everywhere except where their stepwise behavior is engaged, at which point its derivative is the Dirac delta distribution). For the remainder of this work only Equation 6.1 will be used. The derivative of the arctangent function as defined by snnTorch is used as the surrogate gradient for Equation 6.1.

6.3.1.4 Continuous Outputs

Consider the case where the outputs of the quantum circuit are passed to the spiking neuron without undergoing a 0 or 1 binarization as done in Section 6.3.1.2. To motivate this possibility of removing 0 or 1 as an immediate translation of the quantum circuit, consider that spiking neurons take weighted spiking values as mentioned in

Section 6.3.1, and such values will be continuous.

A number of architecture construction possibilities exist for continuous outputs, including having the output of the quantum circuit be passed directly to the spiking neuron layer, and having the the output of the quantum circuit be passed to a dense classical linear layer which is then passed to the spiking neuron layer.

6.3.1.5 Different Models

A number of models are considered. When interacting with the quantum layer there can be continuous input, binary input, continuous output, and binary output models, of which all combinations (as seen in Table 6.2 and Table 6.3, and addressed in detail later) are trained and tested. They are initially separated by their inputs being either continuous or binary.

For all models the loss function was cross entropy spike rate loss, the optimizer was ADAM with a learning rate of 1×10^{-2}, for the Leaky Integrate and Fire neuron (LIF) layers the surrogate gradient for the classical spiking neurons was the Fast Sigmoid with a slope of 25, $\beta = 0.5$, and the number of time steps was 50. The above parameters apply to all models unless noted otherwise.

6.3.1.6 Base (Non-Quantum) Models

There are two purely classical models used in this proof of concept to provide a comparable baseline for classical performance. The first model is C1, where the Classical model only has one (1) middle layer, the second model is C2, where the Classical model

has two (2) middle layers, and both models can be seen in Table 6.1.

Model Architectures For Classical Spiking Networks	
C1	C2
Spiking image (0s & 1s)	Spiking image (0s & 1s)
Flatten image	Flatten image
linear layer, nodes: 784 in, 10 out	linear layer, nodes: 784 in, 10 out
LIF	LIF
linear layer, nodes: 10 in, 10 out	linear layer, nodes: 10 in, 10 out
LIF	LIF
-	linear layer, nodes: 10 in, 10 out
-	LIF
linear layer, nodes: 10 in, 2 out	linear layer, nodes: 10 in, 2 out
LIF	LIF

Table 6.1: Classical spiking network model architectures to be compared with quantum-classical hybrid spiking model architectures [21]. © 2023, IEEE

Model C1 is compared to quantum-classical hybrid models TQS, TQ, PQS, and PQ, and model C2 is compared to quantum-classical hybrid models TQSL, TQL, PQSL, and PQL (both of which can be found in Table 6.2 and Table 6.3). The comparable quantum and classical layers between the hybrid and classical models are the same width, and learn approximately the same number of parameters (i.e. the quantum circuit and the comparative corresponding classical layer have 120 and 110 learned parameters, respectively). Choosing these width and depth equivalencies ensure the dimension and parameter number characterization for both models remain comparable.

6.3.1.7 Quantum-Classical Hybrid Models

There are two classes of models, one class passes continuous values to the quantum circuit from a dense linear layer, and the other class passes binary values to

the quantum circuit (0 or 1 as originating from a spiking neural node).

For both classes of models, LIN refers to a classical dense linear layer, and LIF refers to a leaky spiking neuron layer with the parameters θ (the threshold parameter) and β (the decay rate); unless otherwise noted the values are $\theta = 1.0$ and $\beta = 0.5$. The term "Ceiling" refers to the ceiling function as found in Equation 6.1. The Quantum Circuit refers to a trainable quantum circuit of the following construction:

- 10 wire quantum circuit
  - R_y encoding for each input
  - Hadamard gate on all wires
  - 4 Strongly Entangling Layers across all wires [78]
  - $\langle Z \rangle$ measured on each wire

The tables for models with continuous inputs to the quantum layer can be found in Table 6.2. The lettering on top of Table 6.2 refer to the operation before, during, and after the quantum circuit, e.g. TQSL = Tanh, Quantum (Circuit), (Binary) State (in this case Ceiling), Linear.

6.3.1.8 Binary Input Models

The classical-quantum hybrid models with binary inputs to the quantum circuit can be found in Table 6.3. The quantum circuit between the models in Table 6.2 and Table 6.3 models remains the same, as well as the naming convention between the

Model List For Continuous Inputs To The Quantum Layer			
TQSL	TQL	TQS	TQ
Spiking image (0s & 1s)	Spiking image (0s & 1s)	Spiking image (0s & 1s)	Spiking image (0s & 1s)
Flatten image	Flatten image	Flatten image	Flatten image
LIN nodes: 784 in, 10 out	LIN nodes: 784 in, 10 out	LIN nodes: 784 in, 10 out	LIN nodes: 784 in, 10 out
$\frac{\pi}{2}$Tanh(input)	$\frac{\pi}{2}$Tanh(input)	$\frac{\pi}{2}$Tanh(input)	$\frac{\pi}{2}$Tanh(input)
Quantum Circuit	Quantum Circuit	Quantum Circuit	Quantum Circuit
Ceiling(-input)	_	Ceiling(-input)	_
LIN nodes: 10 in, 10 out	LIN nodes: 10 in, 10 out	_	_
LIF ($\theta = 1.0$, $\beta = 0.5$)	LIF ($\theta = 1.0$, $\beta = 0.5$)	LIF ($\theta = 1.0$, $\beta = 0.5$)	LIF ($\theta = 1.0$, $\beta = 0.5$)
LIN nodes: 10 in, 2 out	LIN nodes: 10 in, 2 out	LIN nodes: 10 in, 2 out	LIN nodes: 10 in, 2 out
LIF	LIF	LIF	LIF

Table 6.2: Model list for continuous inputs to the quantum layer [21]. © 2023, IEEE

Model List For Binary Inputs To the Quantum Layer			
PQSL	PQL	PQS	PQ
Spiking image (0s & 1s)	Spiking image (0s & 1s)	Spiking image (0s & 1s)	Spiking image (0s & 1s)
Flatten image	Flatten image	Flatten image	Flatten image
LIN nodes: 784 in, 10 out	LIN nodes: 784 in, 10 out	LIN nodes: 784 in, 10 out	LIN nodes: 784 in, 10 out
LIF	LIF	LIF	LIF
$\pi\times$input	$\pi\times$input	$\pi\times$input	$\pi\times$input
Quantum Circuit	Quantum Circuit	Quantum Circuit	Quantum Circuit
Ceiling(-input)	_	Ceiling(-input)	_
LIN nodes: 10 in, 10 out	LIN nodes: 10 in, 10 out	_	_
LIF ($\theta = 0.1935$, $\beta = 0.5280$)	LIF ($\theta = 1e-7$, $\beta = 0.8198$)	LIF ($\theta = 0.25$, $\beta = 0.4618$)	LIF ($\theta = 0.017$, $\beta = 0.6$)
LIN nodes: 10 in, 2 out	LIN nodes: 10 in, 2 out	LIN nodes: 10 in, 2 out	LIN nodes: 10 in, 2 out
LIF	LIF	LIF	LIF

Table 6.3: Model list for binary inputs to the quantum layer [21]. © 2023, IEEE

two tables; note that the "P" in Table 6.3 stands for "pi" (π, as in "pi multiplied by the input").

6.4 Results and Discussion

By inspecting the accuracy plots, the hybrid quantum-classical models reach a higher accuracy faster than the comparable purely spiking neural network models. The best performing (when examining both accuracy and loss) quantum model, TQ, achieves top test accuracy 8.25× (times) faster than the comparable classical model, C1 as shown in Figure 6.5; in Figure 6.8 the model PQL achieves the top test accuracy

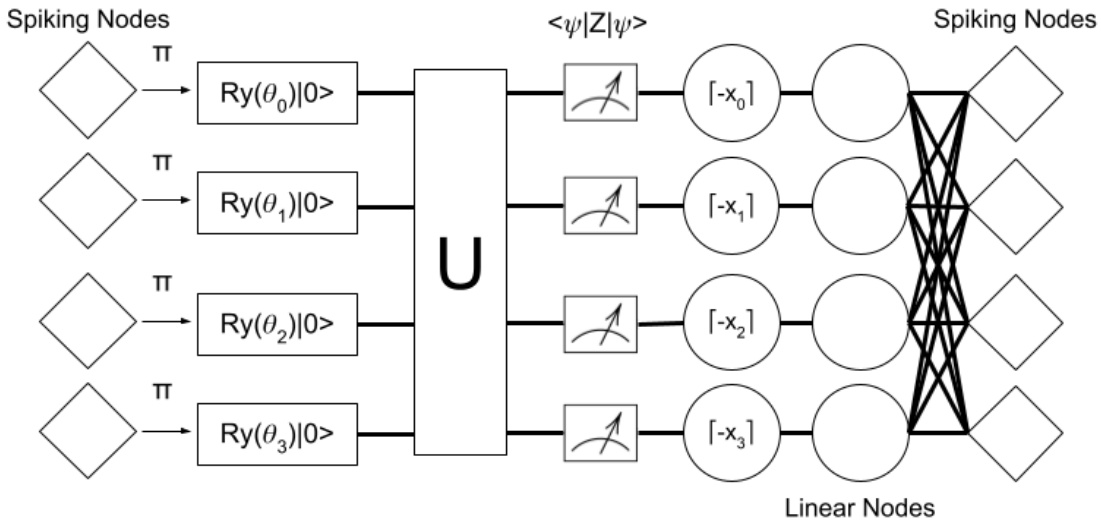


Figure 6.3: The outline for the model PQSL as found in Table 6.3. Let the diamond objects represent the spiking nodes, the round objects represent classical linear nodes, and the boxes represent quantum circuit operations. Let the quantum operation U represent the unitary (U) circuit operations after the R_y encoding detailed in Section 6.3.1.7. Note that the figure does not reflect the exact architecture used in the study, as the displayed outline is only intended to give an idea of data passage, throughput, and transformations [20].

1.66× (times) faster than the comparable classical model, C2. All other hybrid quantum models converge to top accuracy faster than their comparable classical spiking models shown in Figure 6.5 and Figure 6.8.

It's notable that when there are small variations in the training accuracy, the testing data doesn't appear to reflect those changes, although when there is a large dip or spike in the training accuracy the corresponding testing accuracy reflects such

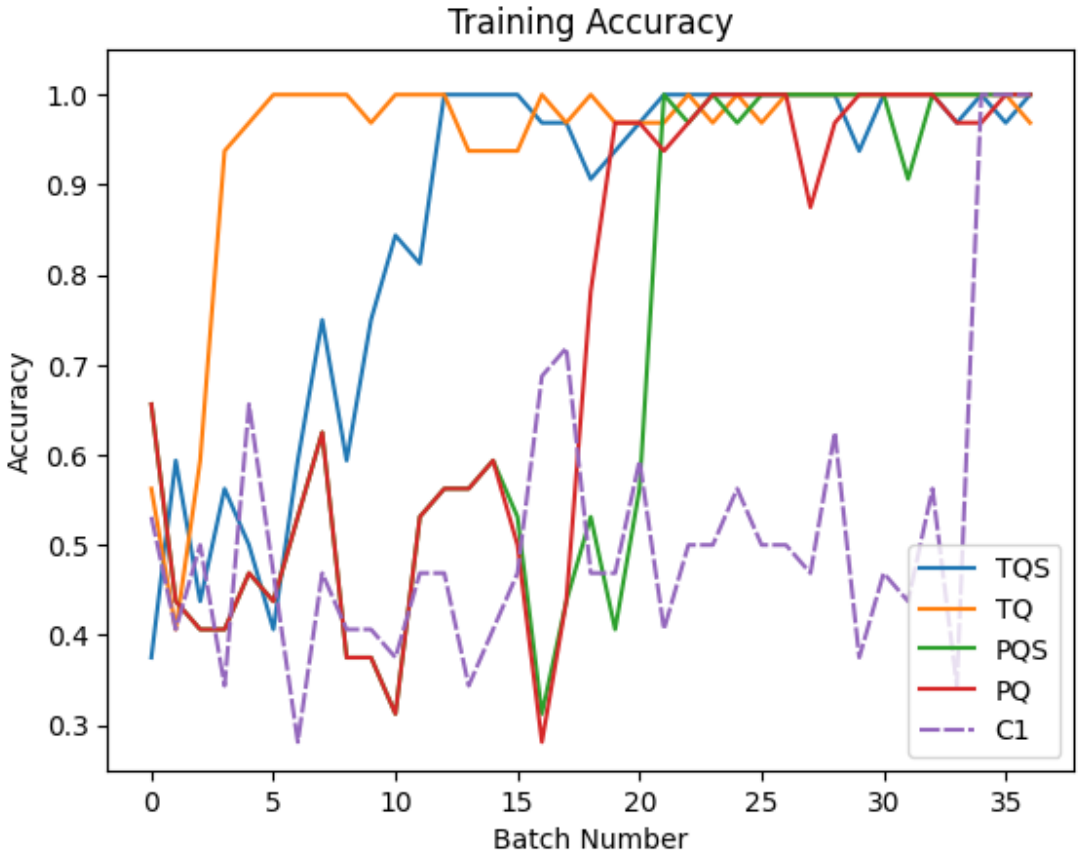


Figure 6.4: Training accuracy versus batch number for comparable models when examining continuous inputs in Table 6.3 and Table 6.2 as opposed to the classical models in Table 6.1. All quantum-classical spiking hybrid models (solid lines) outperform the purely classical spiking model (dashed line) [21]. © 2023, IEEE

large changes. To help understand these behaviors it's helpful to examine the purely spiking network models as they too experience similar behavior, though it appears that the spiking models' testing accuracy only really changes when there is a significantly large change in the training accuracy, as opposed to smoothing out small changes in

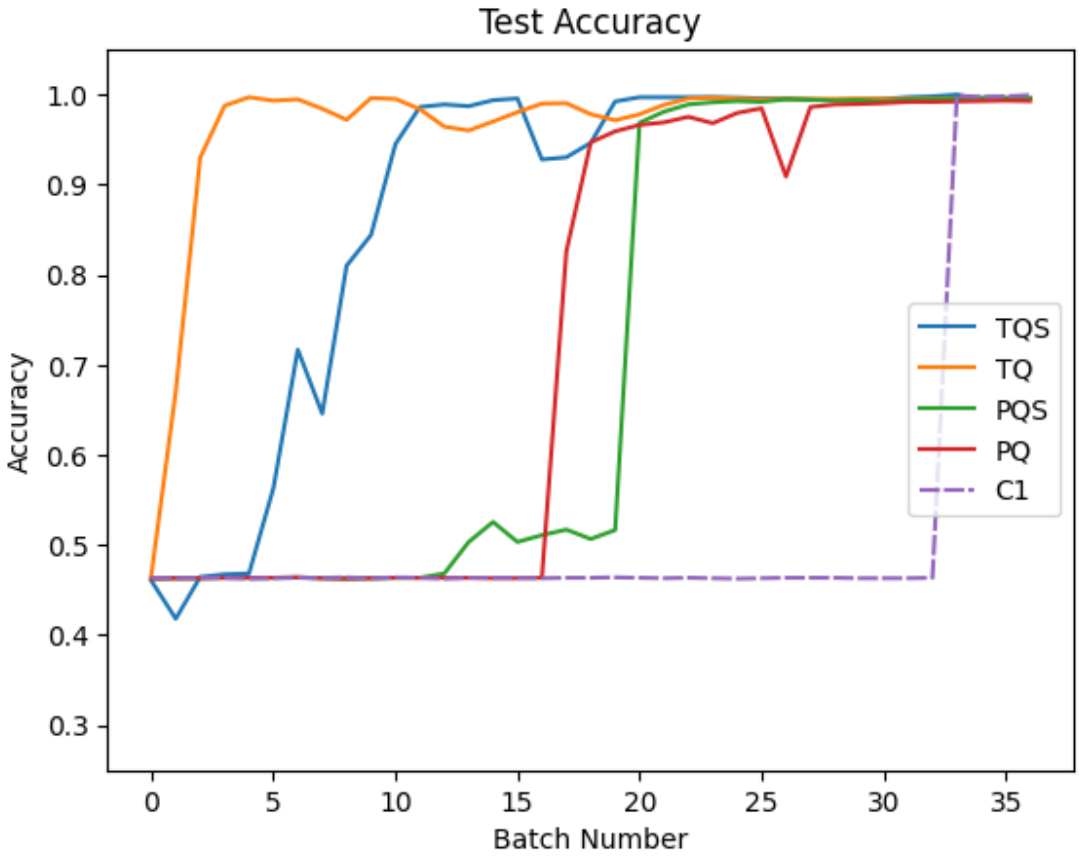


Figure 6.5: Test accuracy versus batch number for comparable models when examining continuous inputs in Table 6.3 and Table 6.2 as opposed to the classical models in Table 6.1. All quantum-classical spiking hybrid models (solid lines) outperform the purely classical spiking model (dashed line) [21]. © 2023, IEEE

accuracy. The spiking part of the hybrid models appear to "filter out" noise when translating between training and testing accuracy while allowing any significant changes in the both model types to remain. In effect the hybrid models have the best of both worlds when training as they achieve good test accuracy with the combined quantum

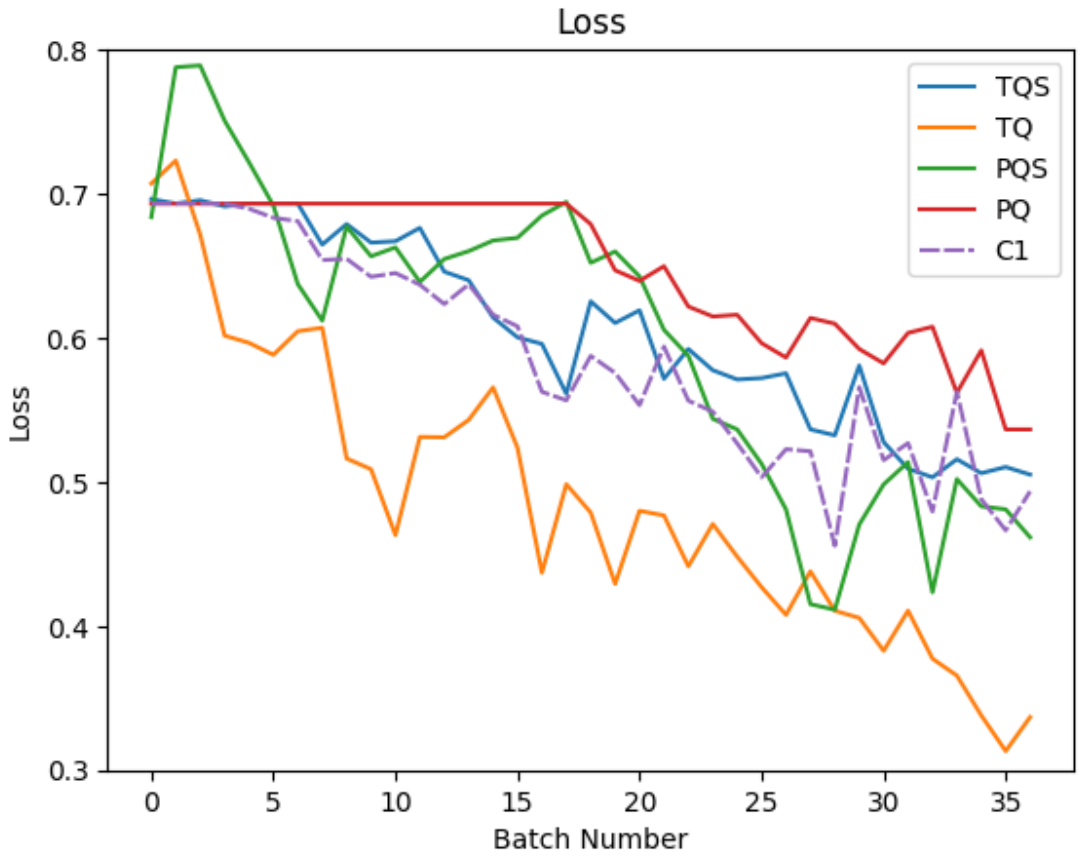


Figure 6.6: Loss versus batch number for comparable models when examining continuous inputs in Table 6.3 and Table 6.2 as opposed to the classical models in Table 6.1 [21].

and spiking hybrid models, while simultaneously getting any noise filtered out by the spiking component of the model. From a practitioner's point of view, this filtering is advantageous as the filtering removes the need to look at individual values in a training sequence but rather only examine the entire trend of training to get an idea for the corresponding test behavior; therefore if a practitioner wanted to get an idea for the

testing performance, then they only needed to look at the general trends of the training performance (i.e. training generalizes well to testing).

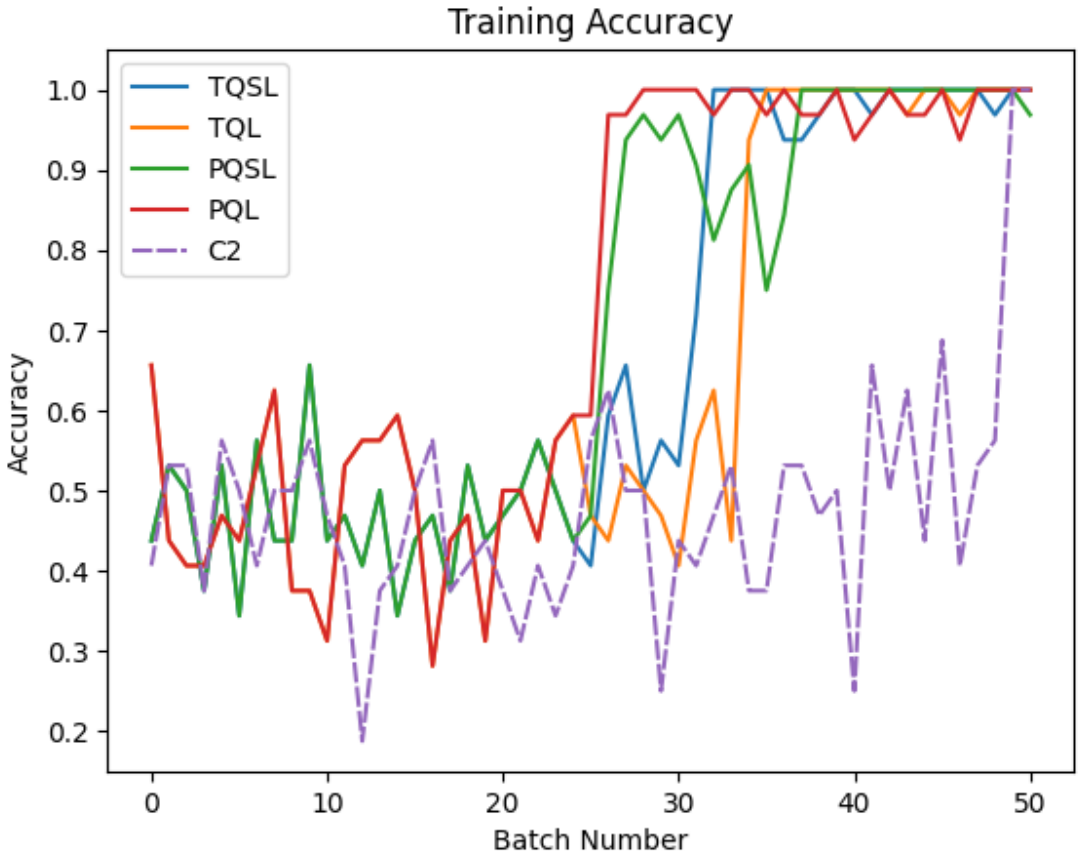


Figure 6.7: Training accuracy versus batch number for comparable models when examining binary inputs in Table 6.3 and Table 6.2 as opposed to the classical models in Table 6.1. All quantum-classical spiking hybrid models (solid lines) outperform the purely classical spiking model (dashed line) [21]. © 2023, IEEE

Compared to C1, the additional spiking layer in C2 results in an extremely flat loss (Figure 6.9) while the hybrid networks appear to be less prone to this delay in

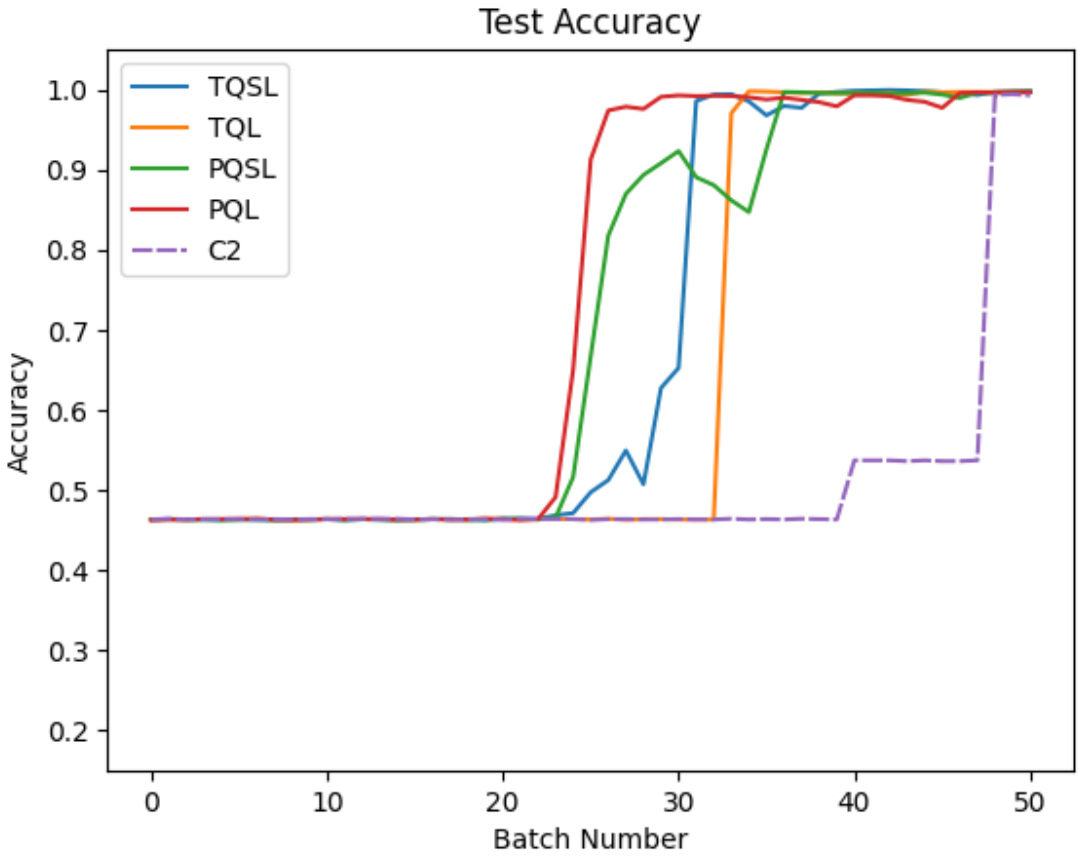


Figure 6.8: Test accuracy versus batch number for comparable models when examining binary inputs in Table 6.3 and Table 6.2 as opposed to the classical models in Table 6.1. All quantum-classical spiking hybrid models (solid lines) outperform the purely classical spiking model (dashed line) [21]. © 2023, IEEE

loss despite the additional spiking layer. In fact, all of the hybrid models in Figure 6.9 outperform C2, leading to lower loss faster.

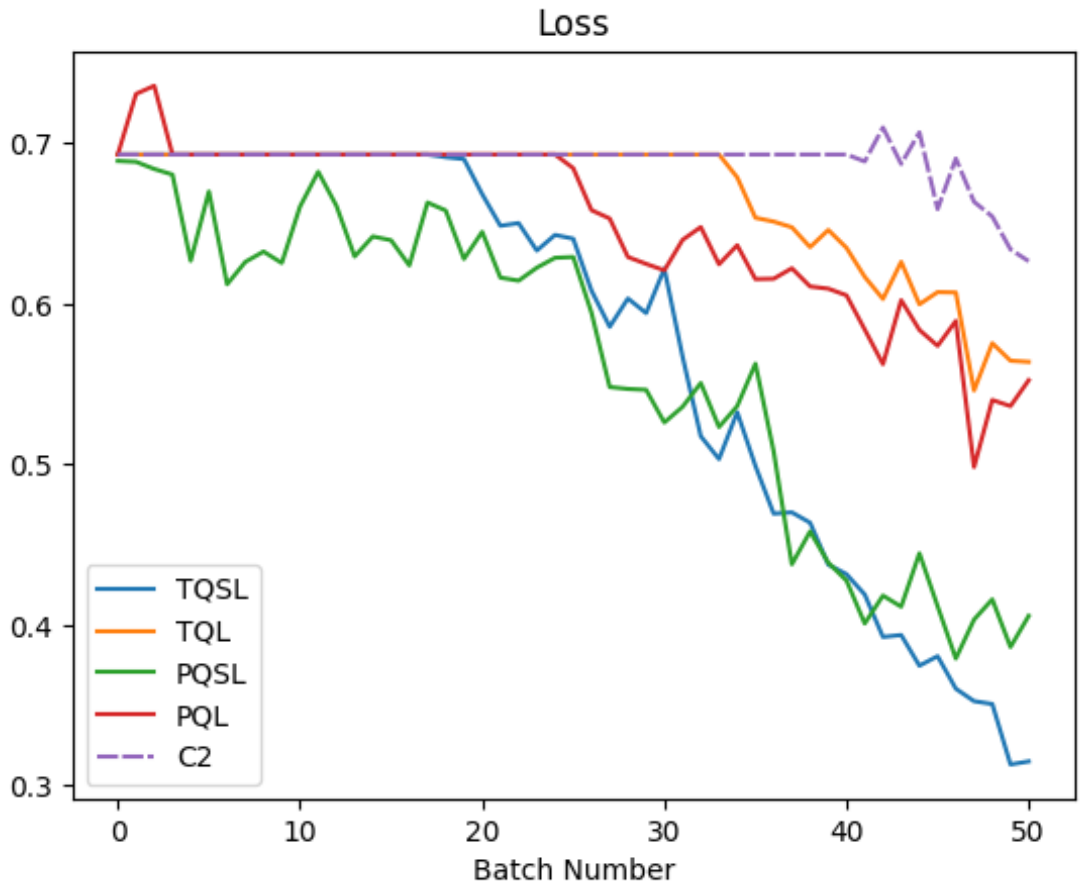


Figure 6.9: Loss versus batch number for comparable models when examining binary inputs in Table 6.3 and Table 6.2 as opposed to the classical models in Table 6.1. All quantum-classical spiking hybrid models (solid lines) outperform the purely classical spiking model (dashed line) [21]. © 2023, IEEE

6.5 Conclusion

While the proposed techniques have shown to be successful in this study, more work is needed to address robustness against noise and benchmark how these techniques

perform against other datasets. These techniques and their plots demonstrate that blending quantum machine learning into spiking neural networks is a technique worth further inquiry and investigation with the novel proposed methods given this successful proof of concept.

Part IV

Conclusion

Chapter 7

Conclusion

7.1 Overview Of Results

A high level overview of the works reviewed in Chapter 4, Chapter 5, and Chapter 6 are outlined below.

Chapter 4 details a novel Quantum Self-Attention mechanism that replaces dot-product attention inside a transformer architecture with a trained variational quantum circuit. This Quantum Self-Attention is embedded inside a Vision Transformer (ViT), resulting in a Quantum Vision Transformer (QViT). All four proposed variations of the QViT are of comparable and competitive performance with the classical ViT.

Chapter 5 demonstrates that training block parameterized unimodular matrices across all wires in a quantum circuit as part of a quantum-classical hybrid model architecture achieves the same accuracy as classical neural networks but at a faster rate

with a more aggressive loss function. The best performing quantum model utilized a 16×16 Unimodular Matrix with a Random Unitary Matrix and Learned Scaling Parameter. The classical model with the ELU activation function performed best among the classical models. When the ELU classical model first achieves about 82% test accuracy the hybrid model does 8.5% better at classifying the bottom moon in the two moons dataset than the classical model when examining the percent difference between classical and quantum-classical hybrid confusion matrices; both models perform equally well in identifying the top moon. The quantum-classical hybrid model achieves about 82% test accuracy 9× (times) faster than the ELU classical model. Towards the end of testing, both models achieve similar performance. Thus the hybrid model with the block matrix approach not only performs just as well as the classical model, but also performs better in a classification category where the classical model still needs more training in order to reach a similar accuracy.

Chapter 6 proposes a variety of hybrid quantum-classical spiking models which reach a higher accuracy faster than the comparable purely spiking neural network models. The best performing hybrid quantum-classical model, TQ, achieves top test accuracy 8.25× (times) faster than the comparable classical model, C1; the model PQL achieves the top test accuracy 1.66× (times) faster than the comparable classical model, C2. All other hybrid quantum models converge to top accuracy faster than their comparable classical spiking models, and all quantum-classical hybrid models compared to C2 have a more aggressive and lower loss.

7.2 Closing Remarks

Quantum machine learning is a quickly growing field with many new ideas, applications, and emerging techniques. Time will tell with help of theoretical developments and new ideas to propose and test them so to explore their applications and different advantages with different systems.

It is difficult to know what lies ahead, in both in the near and far future. However such novelty and naivety brings much excitement and seasons anew. Hard work must still be done to weather and engender where these paths will lead, and if such "quantum advantage" will indeed take hold in the ever growing garden of scientific tools and techniques.

It is too early to say if the field of quantum machine learning is emerging into a Quantum Spring; while there is a boom in funding from both public and private sources there also appears to be great hesitation to declare quantum advantage within the practicing community, much due to the current state of low qubit systems with contentious noise.

It is the hope of this work that some of the proposed techniques find their way into the codebases and papers in this field, and if not such work helps breed new ideas elsewhere; after all, science rests upon the work of those who came before, and with any luck including the work the reader just read.

Bibliography

[1] Minorax. File:AND ANSI Labelled.svg, September 2023. https://en.wikipedia.org/wiki/File:AND_ANSI_Labelled.svg.

[2] Minorax. File:OR ANSI Labelled.svg, September 2023. https://en.wikipedia.org/wiki/File:OR_ANSI_Labelled.svg.

[3] Minorax. File:NAND ANSI Labelled.svg, September 2023. https://en.wikipedia.org/wiki/File:NAND_ANSI_Labelled.svg.

[4] Minorax. File:NOR ANSI Labelled.svg, September 2023. https://en.wikipedia.org/wiki/File:NOR_ANSI_Labelled.svg.

[5] September 2023. https://miro.medium.com/v2/resize:fit:4800/format:webp/1*w9516UckuSEBQdiUOoiHbQ.png.

[6] Minorax. File:NOR ANSI Labelled.svg, September 2023. https://en.wikipedia.org/wiki/Bloch_sphere#/media/File:Bloch_sphere.svg.

[7] Geek3. File:Qcircuit measure-arrow.svg, September 2023. https://en.wikipedia.org/wiki/File:Qcircuit_measure-arrow.svg.

[8] Jun Li, Xinhua Peng, Jiangfeng Du, and Dieter Suter. An Efficient Exact Quantum Algorithm for the Integer Square-free Decomposition Problem. *Scientific Reports*, 2(1), February 2012. https://www.nature.com/articles/srep00260.

[9] Kakitc. File:Finite difference method.svg, September 2023. https://upload.wikimedia.org/wikipedia/commons/9/90/Finite_difference_method.svg.

[10] ARK Investment Management LLC. AI and Compute, August 2023. https://ark-invest.com/articles/analyst-research/ai-training/.

[11] August 2023. https://www.sciencefacts.net/wp-content/uploads/2019/12/Parts-of-a-Neuron-Diagram.jpg.

[12] August 2023. https://qbi.uq.edu.au/files/7716/Synaptic-integration-action-potential-spike-neuron_brain-physiology_QBI.png.

[13] Song Luan, Ian Williams, Konstantin Nikolic, and Timothy G. Constandinou. Neuromodulation: present and emerging methods. *Frontiers in Neuroengineering*, 7, 2014. https://www.frontiersin.org/articles/10.3389/fneng.2014.00027.

[14] Jason K. Eshraghian, Max Ward, Emre O. Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu. Training Spiking Neural Networks Using Lessons From Deep Learning. *Proceedings of the IEEE*, 111(9):1016–1054, 2023.

[15] Hiroto Honda. Unofficial Walkthrough of Vision Transformer, June 2023. https://github.com/hirotomusiker/schwert_colab_data_storage/blob/master/notebook/Vision_Transformer_Tutorial.ipynb.

[16] Dominic Pasquali, Michele Grossi, and Sofia Vallecorsa. Measurements With A Quantum Vision Transformer: A Naive Approach. 2023. [Manuscript submitted for publication to EPJ Web of Conferences].

[17] Dominic Pasquali. Classification With Unimodular Matrices In Hybrid Models. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, pages 420–425, 2022.

[18] A Ajayan and A P James. Edge to quantum: hybrid quantum-spiking neural network image classifier. *Neuromorphic Computing and Engineering*, 1(2):024001, September 2021.

[19] Debanjan Konar, Aditya Das Sarma, Soham Bhandary, Siddhartha Bhattacharyya, Attila Cangi, and Vaneet Aggarwal. A shallow hybrid classical–quantum spiking feedforward neural network for noise-robust image classification. *Applied Soft Computing*, 136:110099, 2023.

[20] Dominic Pasquali, Michele Grossi, and Sofia Vallecorsa. Novel Gradients For Quantum and Spiking Hybrid Models. 2023. [Manuscript in preparation for Quantum Machine Intelligence].

[21] Dominic Pasquali, Michele Grossi, and Sofia Vallecorsa. Classification With Inte-

grated Quantum And Spiking Neural Networks. 2023. [Manuscript submitted for publication to IEEE] © 2023 IEEE.

[22] OpenAI. ChatGPT4, June 2023. [Generated On June 9, 2023], a response to the prompt "write a poem in iambic pentameter about the future of humanity vis a vis artificial intelligence".

[23] Edna Marquez, Eira Valeria Barrón-Palma, Katya Rodríguez, Jesus Savage, and Ana Laura Sanchez-Sandoval. Supervised Machine Learning Methods for Seasonal Influenza Diagnosis. *Diagnostics*, 13(21), 2023. https://www.mdpi.com/2075-4418/13/21/3352.

[24] Salman Zakareya, Habib Izadkhah, and Jaber Karimpour. A New Deep-Learning-Based Model for Breast Cancer Diagnosis from Medical Images. *Diagnostics*, 13(11), 2023. https://www.mdpi.com/2075-4418/13/11/1944.

[25] Abdullah Marish Ali, Farsana Salim, and Faisal Saeed. Parkinson's Disease Detection Using Filter Feature Selection and a Genetic Algorithm with Ensemble Learning. *Diagnostics*, 13(17), 2023. https://www.mdpi.com/2075-4418/13/17/2816.

[26] et al. James Betker. Improving Image Generation with Better Captions. https://cdn.openai.com/papers/dall-e-3.pdf, November 2023.

[27] Devi Sridhar. Quantum Computing's Possibilities and Perils. *Finance Development*, 0058(004):A019, 2021. https://www.elibrary.imf.org/view/journals/022/0058/004/article-A019-en.xml.

[28] Catherine Manning. Technology Readiness Levels, nov 2023. https://www.nasa.gov/directorates/somd/space-communications-navigation-program/technology-readiness-levels/.

[29] Dartmouth Undergraduate Journal of Science. Keeping Up with Moore's Law, May 2013. https://sites.dartmouth.edu/dujs/2013/05/29/keeping-up-with-moores-law/.

[30] James R. Powell. The Quantum Limit to Moore's Law. *Proceedings of the IEEE*, 96(8):1247–1248, 2008.

[31] Ivanovich Manin. *Vychislimoe I Nevychislimoe*. Soviet radio, Moskva, 1980. https://philpapers.org/rec/MANVIN.

[32] Richard P Feynman. Simulating physics with computers. *Int. J. Theor. Phys.*, 21(6-7):467–488, June 1982. https://link.springer.com/article/10.1007/BF02650179.

[33] John Bird. *Engineering Mathematics*. Elsevier Ltd., fifth edition, 2007.

[34] Max Born. Zur Quantenmechanik der Stoßvorgänge. *Eur. Phys. J. A*, 37(12):863–867, 1926. https://link.springer.com/article/10.1007/BF01397477.

[35] Dr. N.P. Landsman. The Born rule and its interpretation, September 2023. https://www.math.ru.nl/~landsman/Born.pdf.

[36] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Wein-

furter. Elementary gates for quantum computation. *Phys. Rev. A*, 52:3457–3467, Nov 1995. https://link.aps.org/doi/10.1103/PhysRevA.52.3457.

[37] A Yu Kitaev. Quantum computations: algorithms and error correction. *Russ. Math. Surv.*, 52(6):1191–1249, December 1997.

[38] P.Oscar Boykin, Tal Mor, Matthew Pulver, Vwani Roychowdhury, and Farrokh Vatan. A new universal and fault-tolerant quantum basis. *Information Processing Letters*, 75(3):101–107, 2000. https://www.sciencedirect.com/science/article/pii/S0020019000000843.

[39] Yaoyun Shi. Both Toffoli and Controlled-NOT need little help to do universal quantum computation, 2002. arXiv:quant-ph/0205115.

[40] Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3-4):219–253, April 1982.

[41] Eric W. Weisstein. Number Field Sieve. From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/NumberFieldSieve.html.

[42] Matthias C. Caro, Hsin-Yuan Huang, M. Cerezo, Kunal Sharma, Andrew Sornborger, Lukasz Cincio, and Patrick J. Coles. Generalization in quantum machine learning from few training data. *Nature Communications*, 13(1), aug 2022.

[43] Seth Lloyd and Christian Weedbrook. Quantum Generative Adversarial Learning. *Physical Review Letters*, 121(4), jul 2018.

[44] Hsin-Yuan Huang, Michael Broughton, Masoud Mohseni, Ryan Babbush, Sergio Boixo, Hartmut Neven, and Jarrod R. McClean. Power of data in quantum machine learning. *Nature Communications*, 12(1), may 2021.

[45] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. A rigorous and robust quantum speed-up in supervised machine learning. *Nature Physics*, 17(9):1013–1017, jul 2021.

[46] Amira Abbas, David Sutter, Christa Zoufal, Aurelien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, jun 2021.

[47] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), 2016. arXiv:1511.07289.

[48] Sridhar Narayan. The generalized sigmoid activation function: Competitive supervised learning. *Information Sciences*, 99(1):69–82, 1997. https://www.sciencedirect.com/science/article/abs/pii/S0020025596002009.

[49] Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU), 2019. arXiv:1803.08375.

[50] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs), 2023. arXiv:1606.08415.

[51] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. https://www.sciencedirect.com/science/article/abs/pii/0893608089900208?via%3Dihub.

[52] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[53] Gavin E. Crooks. Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition, 2019. arXiv:1905.13311.

[54] Leonardo Banchi and Gavin E. Crooks. Measuring Analytic Gradients of General Quantum Evolution with the Stochastic Parameter Shift Rule. *Quantum*, 5:386, jan 2021.

[55] David Wierichs, Josh Izaac, Cody Wang, and Cedric Yen-Yu Lin. General parameter-shift rules for quantum gradients. *Quantum*, 6:677, March 2022.

[56] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii. Quantum circuit learning. *Physical Review A*, 98(3), sep 2018.

[57] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3), mar 2019.

[58] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda

Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, 2020. arXiv:2005.14165.

[59] Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and Predicting the Carbon Footprint of Training Deep Learning Models, 2020. arXiv:2007.03051.

[60] Payal Dhar. The carbon impact of artificial intelligence. *Nature Machine Intelligence*, 2(8):423–425, 2020. https://doi.org/10.1038/s42256-020-0219-9.

[61] William B Levy and Victoria G. Calvert. Computation in the human cerebral cortex uses less than 0.2 watts yet this great expense is optimal when considering communication costs. 2020.

[62] Khan Academy. Overview of neuron structure and function, August 2023. https://www.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/overview-of-neuron-structure-and-function.

[63] Khan Academy. Q A: Neuron depolarization, hyperpolarization, and action potentials, August 2023. https://www.khanacademy.

org/science/biology/human-biology/neuron-nervous-system/a/depolarization-hyperpolarization-and-action-potentials.

[64] James Stewart. *Calculus Early Transcendentals*. Thomson Brookes/Cole, 6th edition, 2008.

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, 2023. arXiv:1706.03762.

[66] LLaMA: Open and Efficient Foundation Language Models, author=Hugo Touvron and Thibaut Lavril and Gautier Izacard and Xavier Martinet and Marie-Anne Lachaux and Timothée Lacroix and Baptiste Rozière and Naman Goyal and Eric Hambro and Faisal Azhar and Aurelien Rodriguez and Armand Joulin and Edouard Grave and Guillaume Lample, 2023. arXiv:2302.13971.

[67] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2019. arXiv:1810.04805.

[68] Guangxi Li, Xuanqiang Zhao, and Xin Wang. Quantum Self-Attention Neural Networks for Text Classification, 2023. arXiv:2205.05625.

[69] Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. BERTweet: A pre-trained language model for English Tweets. In Qun Liu and David Schlangen, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language*

Processing: System Demonstrations, pages 9–14, Online, October 2020. Association for Computational Linguistics.

[70] Nguyen Luong Tran, Duong Minh Le, and Dat Quoc Nguyen. BARTpho: Pre-trained Sequence-to-Sequence Models for Vietnamese, 2022. arXiv:2109.09701.

[71] Moussa Kamal Eddine, Antoine J. P. Tixier, and Michalis Vazirgiannis. BARThez: a Skilled Pretrained French Sequence-to-Sequence Model, 2021. arXiv:2010.12321.

[72] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension, 2019. arXiv:1910.13461.

[73] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations, 2020. arXiv:1909.11942.

[74] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2021. arXiv:2010.11929.

[75] Yanghao Li, Hanzi Mao, Ross Girshick, and Kaiming He. Exploring Plain Vision Transformer Backbones for Object Detection, 2022. arXiv:2203.16527.

[76] El Amine Cherrat, Iordanis Kerenidis, Natansh Mathur, Jonas Landman, Martin Strahm, and Yun Yvonna Li. Quantum Vision Transformers, 2022. arXiv:2209.08167.

[77] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015. arXiv:1505.04597.

[78] Xanadu. qml.StronglyEntanglingLayers, 2022. https://docs.pennylane.ai/en/stable/code/api/pennylane.StronglyEntanglingLayers.html.

[79] LeCun, Yann and Cortes, Corinna and Burges, Christopher J.C. MNIST handwritten digit database. 2010. http://yann.lecun.com/exdb/mnist/.

[80] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[81] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters, 2017. arXiv:1702.05373.

[82] PyTorch. Vision Transformer, July 2022. https://pytorch.org/vision/master/models/vision_transformer.html.

[83] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M. Sohaib Alam, Guillermo Alonso-Linaje, B. AkashNarayanan, Ali Asadi, Juan Miguel Arrazola, Utkarsh Azad, Sam Banning, Carsten Blank, Thomas R Bromley, Benjamin A. Cordier, Jack Ceroni, Alain Delgado, Olivia Di Matteo, Amintor Dusko, Tanya Garg, Diego Guala, Anthony Hayes, Ryan Hill, Aroosa Ijaz, Theodor Isacsson, David Ittah, Soran Jahangiri, Prateek Jain, Edward Jiang, Ankit Khandelwal, Korbinian Kottmann, Robert A. Lang, Christina Lee, Thomas Loke, Angus Lowe, Keri McKiernan, Johannes Jakob Meyer, J. A. Montañez-Barrera, Romain Moyard, Zeyue Niu, Lee James O'Riordan, Steven Oud, Ashish Panigrahi, Chae-Yeun Park, Daniel Polatajko, Nicolás Quesada, Chase Roberts, Nahum Sá, Isidor Schoch, Borun Shi, Shuli Shu, Sukin Sim, Arshpreet Singh, Ingrid Strandberg, Jay Soni, Antal Száva, Slimane Thabet, Rodrigo A. Vargas-Hernández, Trevor Vincent, Nicola Vitucci, Maurice Weber, David Wierichs, Roeland Wiersema, Moritz Willmann, Vincent Wong, Shaoming Zhang, and Nathan Killoran. PennyLane: Automatic differentiation of hybrid quantum-classical computations, 2022. arXiv:1811.04968.

[84] Cirq Developers. Cirq, 2023. https://doi.org/10.5281/zenodo.8161252.

[85] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. The quest for a Quantum Neural Network. *Quantum Information Processing*, 13(11):2567–2586, aug 2014.

[86] Edward Farhi and Hartmut Neven. Classification with Quantum Neural Networks on Near Term Processors, 2018. arXiv:1802.06002.

[87] Misha Denil and Nando de Freitas. Toward the Implementation of a Quantum RBM. In *NIPS 2011 Deep Learning and Unsupervised Feature Learning Workshop*, 2011. http://www.cs.ubc.ca/~nando/papers/quantumrbm.pdf.

[88] Mohammad H. Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum Boltzmann Machine. *Phys. Rev. X*, 8:021050, May 2018.

[89] Andrea Mari, Thomas R. Bromley, Josh Izaac, Maria Schuld, and Nathan Killoran. Transfer learning in hybrid classical-quantum neural networks. *Quantum*, 4:340, oct 2020.

[90] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, aug 2019.

[91] Pierre-Luc Dallaire-Demers and Nathan Killoran. Quantum generative adversarial networks. *Physical Review A*, 98(1), jul 2018.

[92] Bálint Máté, Bertrand Le Saux, and Maxwell Henderson. Beyond Ansätze: Learning Quantum Circuits as Unitary Operators, 2022. arXiv:2203.00601.

[93] H. Zhang, H. Cai, D. Paesani, R. Santagati, A. Laing, L. C. Kwek, and A. Q. Liu. Machine Learning Applied in Reconstruction of Unitary Matrix for Quantum

Computation. In *2019 Conference on Lasers and Electro-Optics (CLEO)*, pages 1–2, 2019.

[94] En-Jui Kuo, Yao-Lung L. Fang, and Samuel Yen-Chi Chen. Quantum Architecture Search via Deep Reinforcement Learning, 2021. arXiv:2104.07715.

[95] Francis D. Murnaghan. *The Unitary And Rotation Groups*. Spartan Books, 1962.

[96] Xanadu. Quantum gradients, July 2022. https://pennylane.ai/qml/glossary/quantum{_}gradient.html.

[97] Gian Giacomo Guerreschi and Mikhail Smelyanskiy. Practical Optimization For Hybrid Quantum-Classical Algorithms, 2017. https://arxiv.org/abs/1701.01450.

[98] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[99] Dominic Pasquali. Simultaneous Quantum Machine Learning Training and Architecture Discovery, 2020. arXiv:2009.06093.

[100] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum*

Information: 10th Anniversary Edition. Cambridge University Press, USA, 10th edition, 2011.

[101] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[102] Debanjan Konar, Vaneet Aggarwal, Aditya Das Sarma, Soham Bhandary, Siddhartha Bhattacharyya, and Attila Cangi. Deep Spiking Quantum Neural Network for Noisy Image Classification. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2023.

[103] Matthew P. A. Fisher. Quantum Cognition: The possibility of processing with nuclear spins in the brain, 2015. arXiv:1508.05929.

[104] Carol P. Weingarten, P. Murali Doraiswamy, and Matthew P. A. Fisher. A New Spin on Neural Processing: Quantum Cognition. *Frontiers in Human Neuroscience*, 10, 2016.

[105] Michael W. Swift, Chris G. Van de Walle, and Matthew P. A. Fisher. Posner molecules: from atomic structure to nuclear spins. *Physical Chemistry Chemical Physics*, 20(18):12373–12380, 2018.

[106] Xanadu Quantum Technologies. qml.AngleEmbedding, 2023. https://docs.pennylane.ai/en/stable/code/api/pennylane.AngleEmbedding.html.

[107] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothee Masquelier, Tiejun Huang, and

Yonghong Tian. Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks, 2021. arXiv:2007.05785.

[108] Friedemann Zenke and Surya Ganguli. SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks. *Neural Computation*, 30(6):1514–1541, June 2018.