Implementing An Artificial Quantum Perceptron

Ashutosh Hathidara (ashuhath@iu.edu)
Lalit Pandey (lpandey@iu.edu)

Abstract

A Perceptron is a fundamental building block of a neural network. The flexibility and scalability of perceptron make it ubiquitous in building intelligent systems. Studies have shown the efficacy of a single neuron in making intelligent decisions. Here, we examined and compared two perceptrons with distinct mechanisms, and developed a quantum version of one of those perceptrons. As a part of this modeling, we implemented the quantum circuit for an artificial perception, generated a dataset, and simulated the training. Through these experiments, we show that there is an exponential growth advantage and test different qubit versions. Our findings show that this quantum model of an individual perceptron can be used as a pattern classifier. For the second type of model, we provide an understanding to design and simulate a spike-dependent quantum perceptron. Our code is available on GitHub[1].

1 Introduction

A perceptron is an artificial unit of an intelligent system capable of making decisions. This artificial unit is inspired by the biological neurons found in the human brain. The human brain has a network of billions of neurons connected to each other. This connectivity leads to the formation of a deep network. Thus, a perceptron is used as a fundamental building block in deep learning systems. In classical computing, these perceptrons have two states, 0 and 1. When the input of the perceptron is sufficient enough to generate an output over the threshold limit, the perceptron is said to be in 'ON' or 1 state. On the other hand, if the output of the perceptron is less than its threshold value, then it is in 'OFF' or 0 state [9].

Decades of research in the field of classical deep learning have given rise to state-of-the-art systems [7] [16] that mimic human-level intelligence. Drawing from recent research that suggests the role of quantum entanglement in consciousness, there has been growing interest in exploring the potential of quantum computing to advance artificial intelligence. However, despite this progress, there remains a gap when implementing quantum algorithms in AI. In this study, we aim to bridge this gap by implementing a quantum model of a perceptron. Here, we review the available literature [12] and implement the quantum circuit using Qiskit quantum simulator [1] to simulate the training of a single perceptron.

- **Problem statement:** We review two papers [12] and [6] which illustrate the design of two distinct types of quantum perceptrons. The fundamental idea behind training a perceptron is to pass the input and a randomly initialized weight vector and to predict the probability of the outcome (in the case of the sigmoid perceptron). For this, first, we implement the quantum circuit in the simulation tool to build a perceptron. Then we generate a dataset that has value-label pairs. Finally, we pass the input and weight vectors to train different qubit versions of a perceptron.

[1]https://github.com/ashutosh1919/quantum-perceptron

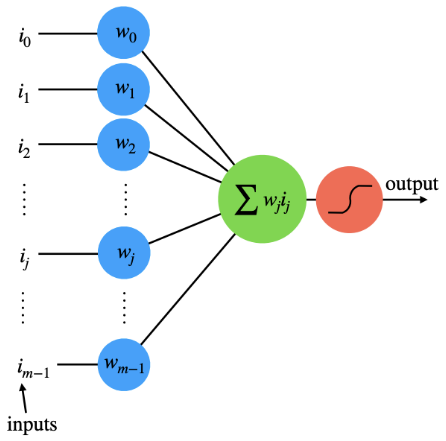


Figure 1: A classical perceptron used in deep learning systems. The perceptron takes multiple input values $\{i_0, i_1, \ldots, i_{n-1}\}$. Internally, it initializes random weight values $\{w_0, w_1, \ldots, w_{n-1}\}$ corresponding to each of the input values. The perceptron computes the dot product of the input and weight vector i.e. $\vec{i} \cdot \vec{w} = \sum_{j=0}^{n-1} i_j w_j$. This dot product result is passed through a non-linear sigmoid [10] function which computes the probability. This probability can be used to compute the loss using the supervised label. The computed loss can then be used to train the perceptron by backpropagating gradients [13] and updating the weights.

- **Motivation:** Almost every advanced deep learning system has artificial neurons as the fundamental building block. Inspired by the success in the classical machine learning field, we intend to implement a quantum version of a perceptron that mimics the properties of a classical perceptron but has the benefits of a quantum system and obeys the rules of quantum mechanics.
- **Our contribution:** We survey paper [12] that introduces a novel architecture and quantum algorithm to design a quantum version of a perceptron. We examine the algorithm and simulate it to test the efficacy of the quantum algorithm. For the implementation, we use QisKit quantum simulation tool and construct quantum gates as specified in the algorithm. We then develop an end-to-end pipeline to generate datasets, initialize weights, train the perceptron, and simulate the probability behavior as discussed in [12]. Following the training process, we conduct a comprehensive analysis to confirm the trained perceptron's ability to accurately classify patterns. Ultimately, we clarify the mathematical consequences of using a perceptron that relies on spikes [6] versus one that does not [12], and differentiate between the two approaches.
- **Existing work:** The concept of a perceptron was first introduced in [4], which presented the classical mathematical framework for utilizing a perceptron as a supervised data classifier. Numerous successful examples have demonstrated the effective application of this mathematical principle in real-world scenarios. In 2013, Lloyed et. al. [8] introduced a theoretical notion of quantum perceptron for supervised and unsupervised learning. Such perceptrons require generalized values and use qRAM [5] to store values. This study contributes to the theoretical literature of quantum computing. In 2014, Schuld et. al. [15] introduced the concept of simulating perceptrons using tools. They used the same simulation tools used in [12] to implement the quantum circuit of a perceptron. The terminology and the approach are similar too. However, [15] utilizes QFT to create intermediate oracle circuits to prepare the input and weight states which operates on an exponential number of gates. On the other hand, [12] make use

of hypergraph states to construct these oracles. This allows them to operate with a polynomial number of quantum gates. The most recent classical deep learning models, as described in [14], utilize bias vectors in addition to weight vectors for their perceptrons. As implementing perceptron algorithms in the quantum field is a relatively new concept, we omit the bias vector and exclusively focus on training the weight vector.

2 Methods

2.1 An Artificial Neuron Implemented on an Actual Quantum Processor

2.1.1 Architecture

Unlike a classical perceptron, a quantum perceptron has quantum gates that prepare the inputs and weights for the system to process. Unitary transformation functions are used to pre-process the input and weight vectors. A Unitary transformation function, also known as an Oracle, houses quantum gates which act upon the input vectors to perform operations such as phase shift, imposing superposition, entanglement, etc. Akin to classical neurons, a quantum perceptron takes an input vector and a weight vector and outputs a probability of the outcome.

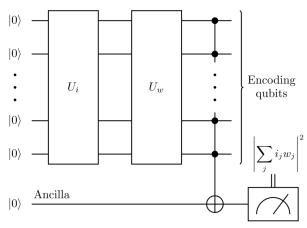


Figure 2: A Quantum Version of Perceptron.

Figure 2 illustrates the internal structure of a perceptron architecture. Two Unitary transformation functions namely, U_i and U_f, are used to perform quantum operations. The input vector is transformed into an input state ($|\psi_i\rangle = \frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} i_j |j\rangle$) by applying the U_i function, while the U_f function transforms the weight vector into a weighted state ($|\psi_w\rangle = \frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} w_j |j\rangle$). After applying the transformation functions, the dot product is calculated between the input and the weight state ($\langle\psi_w|\psi_i\rangle$). This entire series of operations are carried out until the model converges and we obtain the optimal weight.

2.1.2 Dataset Generation

We used the same quantum perceptron to generate the dataset consisting of value-label pairs. Following Mcculloch et. al. [9], we replaced all the classical bits containing 1 with -1 and 0 bits with 1. For instance, if the input value is 12, then the transition from classical to quantum vector will look as $12 \rightarrow [1, 1, 0, 0] \rightarrow [-1, -1, 1, 1]$. Algorithm 1 was tested using varying numbers of qubits, resulting in 16 possible input values when using 2 qubits and 2^{16} possible input values when using 4 qubits.

A neural network requires a dataset to operate upon and to update the network's parameters. To generate the dataset, first, we take a fixed optimal weight $w_0 = 626$ as shown in figure 3. Second, we passed sequential input values and weight w_o to the perceptron. Finally, we compute the output probability and based on that label the data. If the probability was less than 0.5, the input value was classified as 0, and if it was 0.5 or greater, the input value was classified as 1. The weight was constant and did not update throughout

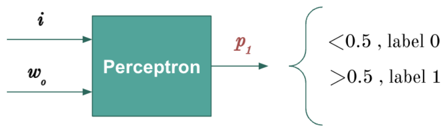


Figure 3: Generating dataset using single perceptron

Algorithm 1 Data Generation

```
Require: Optimal weight $w_o$, Number of qubits $n$, Number of iterations $N$
1:  $data \leftarrow \{\}$                          ▷ Initializing empty list
2:  $p \leftarrow Perceptron(n)$                    ▷ Initializing perceptron
3:  for $i \in [0, 2^{2^n} - 1]$ do
4:      $p.input \leftarrow i$
5:      $p.weight \leftarrow w_0$
6:      $p_1 \leftarrow p.measure(N)$               ▷ Probability of measuring 1
7:      if $p_1 < 0.5$ then
8:          $data.add((i, 0))$                      ▷ Assigning label 0
9:      else if $p_1 \geq 0.5$ then
10:         $data.add((i, 1))$                      ▷ Assigning label 1
11:     end if
12: end for
```

the data collection process. This approach is similar to supervised learning in the case of classical deep learning systems.

2.1.3 Training

Classical deep learning systems need an enormous amount of training to achieve convergence. In contrast, quantum computing offers the advantage of rapidly converging models. During the training phase as shown in algorithm 2, each perceptron is initialized with a random weight which is updated after each training iteration. Here, for a system with 4 qubits, we initialize a random weight w_t. The goal of training the perceptron is to update its weights, such that it can correctly classify the input values as per their labels. The two cases of misprediction, while the perceptron is under training, are:

Case 1: Predicted label = 0, Actual label = 1. In this case, we first find the number of non-matching bits between the input and weight sequence. Next, we multiply the learning rate by the number of non-matching bits and round down to obtain a product. Finally, we randomly flip the resulting number (product obtained in the above step) of bits in the weight, bringing it closer to the input sequence and facilitating faster convergence of the model.

Case 2: Predicted label = 1, Actual label = 0. In this case, instead of finding non-matching bits, we look for the matching bits between the input and weight sequence. The rest of the steps remain the same as in case 1.

The weight of the perceptron is updated after each training iteration (epoch) based on the above two cases. Finally, we check if $w_t = w_o$ and stop the training if satisfied.

Algorithm 2 Training Perceptron

```
Require: Optimal weight $w_o$, Number of qubits $n$, Number of iterations $N$, $data$
1:  $w_t \leftarrow U(0, 2^{2^n} - 1)$                    ▷ Randomly initialize weight for training
2:  $p \leftarrow Perceptron(n)$                          ▷ Initializing perceptron
3:  for $i, l \leftarrow data$ do
4:      $p.input \leftarrow i$
5:      $p.weight \leftarrow w_t$
6:      $p_1 \leftarrow p.measure(N)$                     ▷ Probability of measuring 1
7:      if $p_1 < 0.5$ and $l = 1$ then
8:          FLIP-NON-MATCHING-BITS($w_t, i$) ▷ Flip non-matching bits of $w_t$ w.r.t $i$
9:      else if $p_1 \geq 0.5$ and $l = 0$ then
10:         FLIP-MATCHING-BITS($w_t, i$)          ▷ Flip matching bits of $w_t$ w.r.t $i$
11:     end if
12:     converged if $w_t = w_o$
13: end for
```

2.1.4 Results

Pattern Classification: We trained a quantum perceptron and visualized its optimal weights after training. Figure 4 shows the training steps and the transformation of randomly initialized weight into a complete pattern. Through our experiments, we found that a single quantum perceptron can successfully classify simple patterns of horizontal and vertical lines. Here, we report one such pattern after training the perceptron.

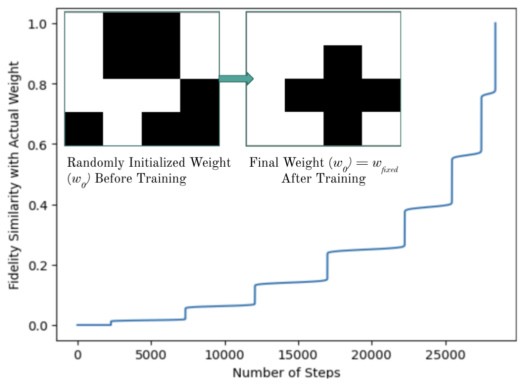
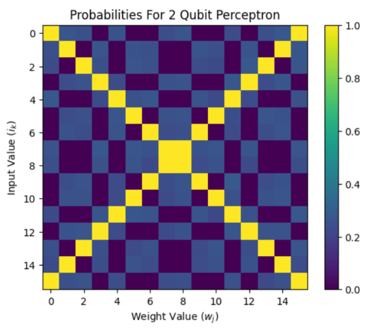


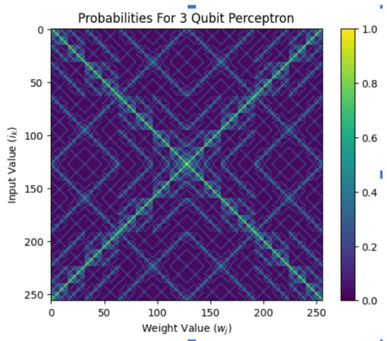
Figure 4: Training procedure for the generated data

Faster Convergence: Compared to classical deep learning systems, a quantum perceptron can achieve optimal performance faster and has the ability to terminate training once the optimal weight has been reached. We found that a four-qubit system converged and reached the optimal weight before the training was completed.

Identical Input and Weight: Finally, we only get a probability of 1 when the input and the weight have the same value. The geometrical patterns in Figure 5 denote the perceptron probability for all combinations of input and weight values.



(a) Simulation for 2 qubit system



(b) Simulation for 3 qubit system

Figure 5: Simulation of perceptron on all combinations of input and weight values

2.2 An Artificial Spiking Quantum Neuron

This subsection focuses on describing quantum spiking neurons as illustrated in [6]. For classical systems, Moore's law is slowing down [3] as the design of the computers gets more complex. Essentially, it is the reason why researchers started looking into quantum computing to improve the speed of computing. One of the objectives of looking into quantum is that it processes information as superpositions of all states and due to this, it gives an illusion of parallel processing of data. This paper illustrates quantum spin models which we can think of as a small component. We can combine these components to create a more complex model.

2.2.1 Architecture

In the classical implementation of neuron, the neuron gets activated based on the weighted sum of inputs as shown in figure 1. Inspired by this, the quantum spiking neuron is built using two components. The first component flips the output spin based on how many input states are active ($|\uparrow\rangle$) spin. The second component compares the relative phases of the smaller components on a computational basis. Let us look at the architecture of both of these components.

Counting Excitations: From the classical analogy, the first neuron in this model tries to activate ($|\uparrow\rangle$) the output spin depending on the number of input states which are in the active spin. Hamiltonian operator [2] corresponds to the total energy of the quantum system. It can be used to derive the time evolution of the quantum system. The authors of this paper have created the Hamiltonian operator which can count the number of activated input states. The Hamiltonian is given by:

$$H_{Exc} = \frac{J}{2}(\sigma_1^x\sigma_2^x + \sigma_1^y\sigma_2^y + \sigma_1^z\sigma_2^z) + \beta\sigma_2^z\sigma_3^z + Acos(\frac{2\beta}{\hbar}t)\sigma_3^x$$

Here σ^* represents corresponding basis vectors of different qubits. The important thing to note here is that 1 and 2 are input qubits and 3 is an output qubit of the system. The intuition behind the above Hamiltonian is that interaction between input states creates energy differences between four bell states and we can detect these differences based on the output spin state of the system. Skipping the smaller details for simplicity, this component provides the below result on passing different bell states as inputs.

$$\left|\psi^{\pm}\right\rangle|\downarrow\rangle \rightarrow \left|\psi^{\pm}\right\rangle|\downarrow\rangle$$
$$\left|\phi^{\pm}\right\rangle|\downarrow\rangle \rightarrow \left|\phi^{\pm}\right\rangle|\uparrow\rangle$$

The first ket corresponds to the input state and the second ket corresponds to the output states in the above expressions. The outputs are either activated or not activated, it is not the superposition. Thus, it mimics the functionality of classical spiking neurons [11].

Comparing Relative Phases: The first component discussed above has a very similar behavior as classical spiking neurons. But the input quantum states are not like classical input values. The quantum states also have phases. We also need to handle the relative phase of the input states. The goal of the second neuron is to detect the phase difference between the positive bell states and negative bell states in the basis. Similar to the above component, phase detection can also be described by another Hamiltonian:

$$H_{Phase} = \frac{J}{2}(\sigma_1^x\sigma_2^x + \sigma_1^y\sigma_2^y + \sigma_1^z\sigma_2^z) + \delta\sigma_2^z\sigma_3^z + B\sigma_3^x$$

The fundamental objective of this second neuron is to correct the relative phase of the input qubits in the case when one qubit is a positive bell state and another is a negative bell state.

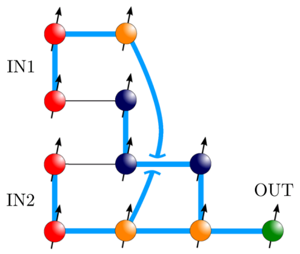


Figure 6: Quantum spiking neuron

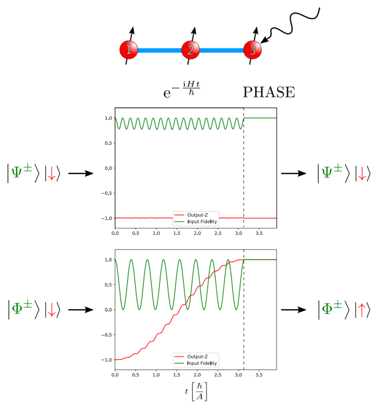
Figure 6 describes the overall model for quantum spiking neurons. The first layer represented by red nodes is an input layer. Subsequent layers contain multiple neurons of type 1 and 2 depending on the use case. The orange nodes represent neurons of type 1 which count excitations. The blue nodes represent neurons of type 2 which detect and correct relative phase. We get an output state at the end. From the above-described Hamiltonian operators, the generated output state will be entangled with the input states.

2.2.2 Results

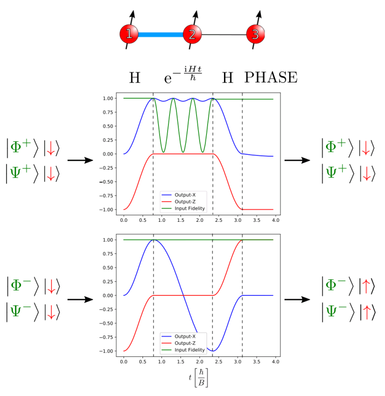
The architecture described above mimics the behavior of the classical spiking neurons in the quantum systems. Figure 7 illustrates the results of two different types of neurons described above. After creating the quantum system shown in 6, we pass different bell states as inputs.

Figure 7(a) demonstrates the effect of output states for neuron 1. Whenever we pass $\left|\psi^{\pm}\right\rangle$ as inputs the output state doesn't get activated $|\downarrow\rangle$. But when we pass $\left|\phi^{\pm}\right\rangle$, the output state gets activated $|\uparrow\rangle$.

Figure 7(b) demonstrates the effect of output states for neuron 2. Whenever we pass $\left|\psi^{+}\right\rangle$ and $\left|\phi^{+}\right\rangle$, it doesn't detect any phase and the output state of the neuron doesn't get



(a) Behavior of Neuron 1



(b) Behavior of Neuron 2

Figure 7: The behavior of two types of neurons on different input bell states

activated. But when we pass $\left|\psi^-\right\rangle$ and $\left|\phi^-\right\rangle$, it detects the relative phase between two inputs, and the output state of the neuron gets activated.

The spiking neurons can be used for quantum preparation tasks. Specifically, when we need to prepare two entangled quantum states, it is useful since the input and the output states of the spiking neuron described above are entangled. Moreover, this type of neuron can be used for quantum communication. The first neuron either gets activated or not and thus, it imitates the behavior of classical bits and can be used in communication tasks.

3 Conclusion

We implemented a quantum version of a perceptron and tested the algorithm's efficacy. Upon analysis, a single perceptron was able to classify patterns after training. The results suggest that a quantum perceptron converges faster than a classical perceptron. This faster convergence highlights the parallel processing of the inputs present in the superposition states. One of the limitations of this work is the use of a single perceptron to design a classifier. Another limitation is the absence of bias vectors in addition to the weight vectors in the training process. We also confine the input values (only -1 and 1) when training the perceptron. Future work will focus on designing and implementing an advanced network with more interconnected perceptrons. This will lead to the development of an advanced quantum network for classification purposes.

We also reviewed [6] which introduces quantum spiking neurons. The objective of the quantum neuron is to mimic the spiking behavior of classical spiking neurons. The quantum system usually outputs a superposition state which generates the probabilistic output. The probabilistic outputs distort the learning scheme in machine learning models and thus, spiking behavior is important. Spiking behavior outputs the activated or non-activated spin quantum states which is beneficial. Although this spiking model looks very promising, it is easily affected by the environment and hence is unstable. It can be used for quantum preparation and quantum communication.

Contributions

For this study, Lalit implemented the dataset generation code and generated the dataset for training the perception. Ashutosh implemented the perceptron and trained the model. The report and presentation have equal contributions from Lalit and Ashutosh.

References

[1] Qiskit.

[2] Mario Castagnino and Olimpia Lombardi. The role of the Hamiltonian in the interpretation of quantum mechanics. *Journal of physics*, 128:012014, 8 2008.

[3] Lieven Eeckhout. Is Moore's Law Slowing Down? What's Next? *IEEE Micro*, 37(4):4–5, 7 2017.

[4] Frederic B. Fitch. Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. *Journal of Symbolic Logic*, 9(2):49–50, 6 1944.

[5] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical Review Letters*, 100(16), apr 2008.

[6] Lasse Sommer Kristensen, Matthias Degroote, Peter Wittek, Alán Aspuru-Guzik, and Nikolaj Thomas Zinner. An artificial spiking quantum neuron. *npj Quantum Information*, 7(1), 4 2021.

[7] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, Zihao Wu, Dajiang Zhu, Xiang Li, Ning Qiang, Dingang Shen, Tianming Liu, and Bao Ge. Summary of chatgpt/gpt-4 research and perspective towards the future of large language models, 2023.

[8] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning, 2013.

[9] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of mathematical biophysics*, 5(4):115–133, 1 1990.

[10] Sridhar Narayan. The generalized sigmoid activation function: Competitive supervised learning. *Information Sciences*, 99(1-2):69–82, 6 1997.

[11] Michael Pfeiffer and Thomas Pfeil. Deep Learning With Spiking Neurons: Opportunities and Challenges. *Frontiers in Neuroscience*, 12, 10 2018.

[12] David E. Rumelhart. Learning internal representations by back-propagating errors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362, 1 1986.

[13] David E. Rumelhart. Learning internal representations by back-propagating errors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362, 1 1986.

[14] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[15] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Simulating a perceptron on a quantum computer. *Physics Letters A*, 379(7):660–663, mar 2015.

[16] Murray Shanahan. Talking about large language models, 2023.