ISSN: 2997-2795

Research Article

Annals of Computational Physics and Material Science

# Implementing an Artificial Quantum Perceptron

**Ashutosh Hathidara[1]* and Lalit Pandey[2]**

[1]*SAP AI, USA*

[2]*Informatics Department, Indiana University, Bloomington, Indiana, USA*

***Corresponding Author**
Ashutosh Hathidara, SAP AI, USA.

**Submitted:** 2024, Dec 05; **Accepted:** 2024, Dec 24; **Published:** 2025, Jan 03

**Citation:** Hathidara, A., Pandey, L. (2025). Implementing an Artificial Quantum Perceptron. *Ann Comp Phy Material Sci, 2*(1), 01-05.

**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 perceptron's with distinct mechanisms, and developed a quantum version of one of those perceptron's. As a part of this modelling, 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 at https://github.com/ashutosh1919/quantum-perceptron*

**Keywords:** Quantum Perceptron, Quantum Computing, Quantum Machine Learning

## 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 perceptron's 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 [1]. Decades of research in the field of classical deep learning have given rise to state-of-the-art systems that mimic human-level intelligence [2,3]. 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 and implement the quantum circuit using Qiskit quantum simulator to simulate the training of a single perception [4,5].

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 attempt 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. Previous works like introduce a novel architecture and quantum algorithm to design a quantum version of a perceptron [4]. 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 behaviour as discussed in [4]. Following the training process, we conduct a comprehensive analysis to confirm the trained perceptron's ability to accurately classify patterns.

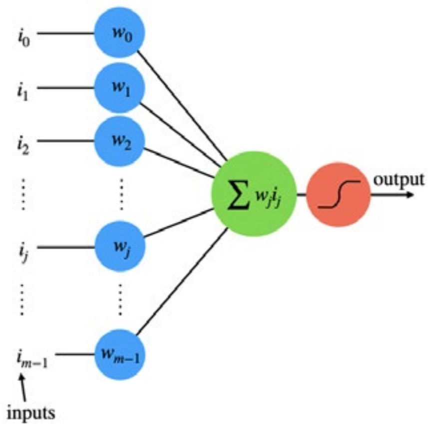



**Figure 1:** A classical perceptron used in deep learning systems. The perceptron takes multiple input values $\{i_0, i_1, ..., i_{n-1}\}$. Internally, it initializes random weight values $\{w_0, w_1, ..., w_{n-1}\}$ corresponding to each of the input values. The perceptron computes the dot product 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 function which computes the probability [6]. 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 and updating the weights [4].

### 1.1. Related Work

The concept of a perceptron was first introduced in [7], 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., introduced a theoretical notion of quantum perceptron for supervised and unsupervised learning [8]. Such perceptron's require generalized values and use qRAM to store values [9]. This study contributes to the theoretical literature of quantum computing. In 2014, Schuld, et al., introduced the concept of simulating perceptron's using tools [10]. They used the same simulation tools used in to implement the quantum circuit of a perceptron [4]. The terminology and the approach are similar too. However, [10] 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, [4] 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 [11], utilize bias vectors in addition to weight vectors for their perceptron's. 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. 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.

$$|\psi_i\rangle = \frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} i_j \,|j\rangle \tag{1}$$

$$|\psi_w\rangle = \frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} w_j \,|j\rangle \tag{2}$$

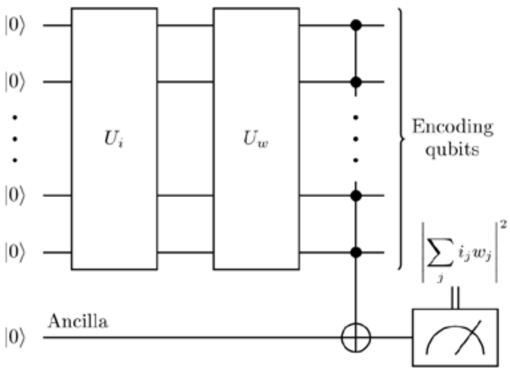



**Figure 2:** A Quantum Version of Perceptron

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 by applying the $U_i$ function as shown in equation 1, while the Uf function transforms the weight vector into a weighted state as shown in equation 2. 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.

## 2.2. Dataset Generation

We used the same quantum perceptron to generate the dataset consisting of value-label pairs. Following Mcculloch et al., we replaced all the classical bits containing 1 with -1 and 0 bits with 1 [1]. For instance, if the input value is 12, then the transition from classical to quantum vector will look as 12 → [1,1,0,0] → [−1,−1,1,1].

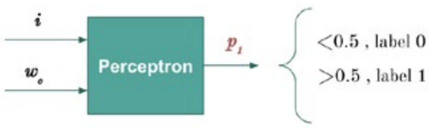



**Figure 3**: Generating Dataset using Single Perceptron

The overall implementation of dataset generation is described in algorithm 1. The algorithm was tested using varying numbers of qubits, resulting in 16 possible input values when using 2 qubits and 216 possible input values when using 4 qubits.

**Algorithm 1** Data Generation

**Require:** Optimal weight $w_o$, Number of qubits $n$, Number of iterations $N$

$data \leftarrow \{\}$ ▷ Initializing empty list
$p \leftarrow Perceptron(n)$ ▷ Initializing perceptron
**for** $i \in [0, 2^{2^n} - 1]$ **do**
    $p.input \leftarrow i$
    $p.weight \leftarrow w_0$
    $p_1 \leftarrow p.measure(N)$ ▷ Probability of measuring 1
    **if** $p_1 < 0.5$ **then**
        $data.add((i, 0))$ ▷ Assigning label 0
    **else if** $p_1 \geq 0.5$ **then**
        $data.add((i, 1))$ ▷ Assigning label 1
    **end if**
**end for**

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

## 2.3. 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. The quantum perceptron training is described in the algorithm 2.

**Algorithm 2** Training Perceptron

**Require:** Optimal weight $w_o$, Number of qubits $n$, Number of iterations $N$, $data$

$w_t \leftarrow U(0, 2^{2^n} - 1)$ ▷ Randomly initialize weight for training
$p \leftarrow Perceptron(n)$ ▷ Initializing perceptron
**for** $i, l \leftarrow data$ **do**
  $p.input \leftarrow i$
  $p.weight \leftarrow w_t$
  $p_1 \leftarrow p.measure(N)$ ▷ Probability of measuring 1
  **if** $p_1 < 0.5$ and $l = 1$ **then**
    FLIP-NON-MATCHING-BITS($w_t, i$) ▷ Flip non-matching bits of $w_t$ w.r.t $i$
  **else if** $p_1 \geq 0.5$ and $l = 0$ **then**
    FLIP-MATCHING-BITS($w_t, i$) ▷ Flip matching bits of $w_t$ w.r.t $i$
  **end if**
  converged if $w_t = w_o$
**end for**

During the training phase, 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. In case when the misprediction happens, we need to penalize the loss such that the weights are updated. Here, we have two cases of misprediction. Below, we describe the details to handle the misprediction to update weights.

**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. Note that we do not need to update the weights when the prediction is correct since the loss in such cases would be zero. Finally, we check if $w_t = w_o$ and stop the training if satisfied.

## 3. 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.

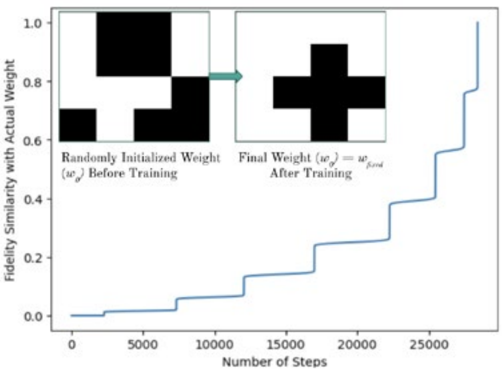
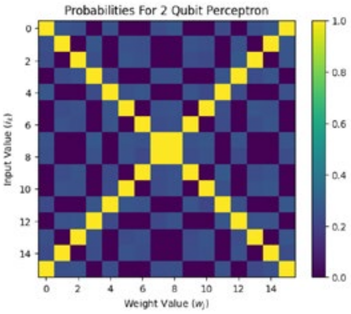


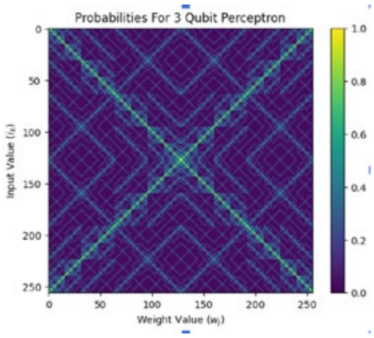

**Figure 4:** Training Procedure for the Generated Data

• **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

## 4. Conclusion and Future Work

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 perceptron's. This will lead to the development of an advanced quantum network for classification purposes.

## Acknowledgment

We thank Dr. Mohsen Heidari, professor at Indiana University Bloomington, for being our instructor, guiding us throughout the project, and thereby supporting our work.

## References

1. McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics, 5*, 115-133.
2. Liu, Y., Han, T., Ma, S., Zhang, J., Yang, Y., Tian, J., ... & Ge, B. (2023). Summary of chatgpt-related research and perspective towards the future of large language models. *Meta-Radiology*, 100017.
3. Shanahan, M. (2024). Talking about large language models. *Communications of the ACM, 67*(2), 68-79.
4. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by back-propagating errors in Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Eds.
5. Qiskit.
6. Narayan, S. (1997). The generalized sigmoid activation function: Competitive supervised learning. *Information sciences, 99*(1-2), 69-82.
7. Fitch, F. B. (1944). 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.
8. Lloyd, S., Mohseni, M., & Rebentrost, P. (2013). Quantum algorithms for supervised and unsupervised machine learning. *arXiv preprint arXiv:1307.0411*.
9. Giovannetti, V., Lloyd, S., & Maccone, L. (2008). Quantum random access memory. *Physical review letters, 100*(16), 160501.
10. Schuld, M., Sinayskiy, I., & Petruccione, F. (2015). Simulating a perceptron on a quantum computer. *Physics Letters A, 379*(7), 660-663.
11. Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks, 61*, 85-117.