Chapter 2

Neurons, Neural Network and Its Evolution History

"Talk is cheap. Show me the code."

— Linus Torvalds

2.1 McCulloch-Pitts Neuron

Creating something like us, beings or machines, is a long dream of human beings, which can be traced back to thousands of years ago. However, the first solid step in understanding the operating principle of the human brain was in 1943, when Warren McCulloch, a neurophysiologist, a young logician, Walter Pitts, published their paper on how biological neurons might work [**McCulloch-Pitts**]. They built a highly simplified model by using multi-connected artificial neurons to simulate how the brain could produce highly complex patterns. Thence, the first artificial neuron was called McCulloch-Pitts neuron (hereafter denoted only as MCP neuron), mimicking the key features of biological neurons in our brains. Though it is nowhere near creating a complex system similar to the biological neural network, their

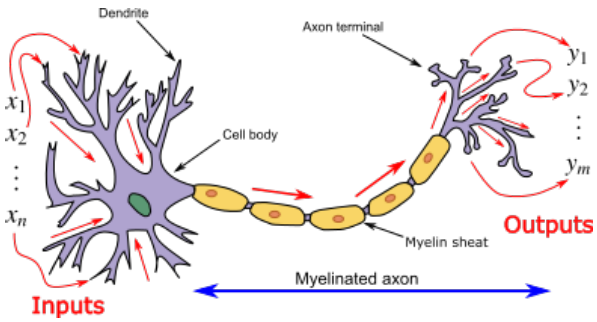

FIGURE 2.1: Schematic representation of biological neuron and myelinated axon. Image Credit: Wiki

seminal paper started the revolution in the development of artificial neural networks. The computational model includes two associated components. The first unit takes boolean inputs (often called features) and performs a summation. Then the second unit makes a binary prediction based on the summarized result and a Heaviside step function. In addition, there is one inhibitory input.The model can not be fired if the inhibitory input is on. To put the operating principle in more precise algebraic terms:

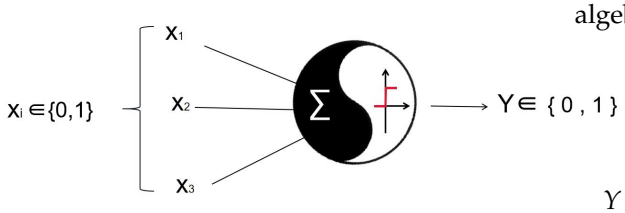

FIGURE 2.2: The schematic diagram is all there is to how a MCP Neuron works.

$$Y = \begin{cases} 1 & \text{if } \sum_i x_i > threshold \\ 0 & \text{if } \sum_i x_i \leq threshold \\ 0 & \text{if inhibitory input} \neq 0 \end{cases} \tag{2.1}$$

This is the underlying mathematical model. Let us produce a simple NOvA NC-Cosmic binary classifier to understand better what type of problems can be addressed by the MCP neuron. The input dataset is not a realistic sample, but it is easy to understand, and we will soon get to more practical examples.

2.1.1 Generating A Pseudo Dataset

To train a toy model based on the MCP neuron, let us firstly build a NOvA pseudo dataset.

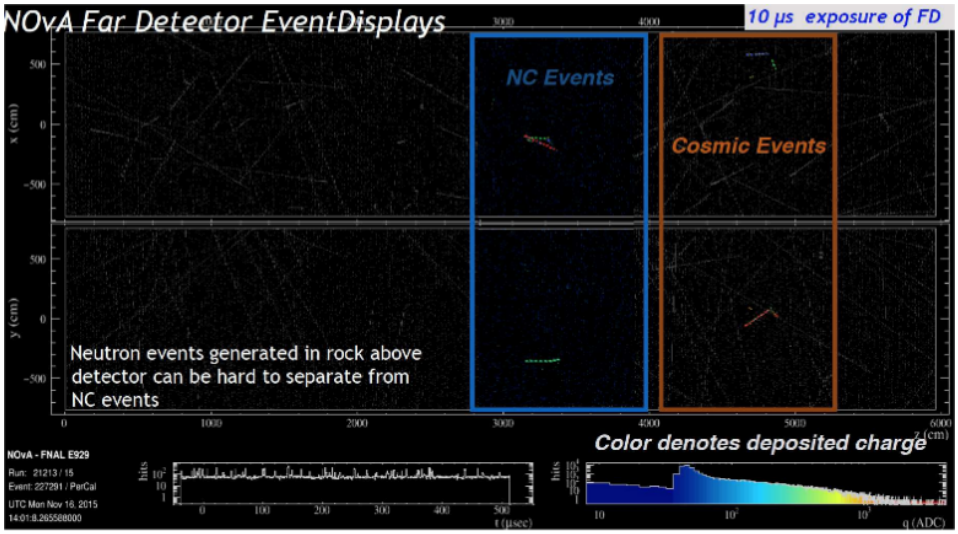

FIGURE 2.3: FD NC and cosmic ray interaction comparison.

As we introduced in the first chapter that producing and observing neutrinos require extraordinary resources and detectors. Therefore, how to extract the maximal information from the detected data is a crucial concern for neutrino-nuclear interaction related analyses. Signal selection is difficult for the neutral current (NC) interaction related analysis because several types of noises, for instance, cosmic ray relevant interactions, can mimic the signal's characteristic signature, as presented in Fig 2.3. The neutrino physics community has been developing dedicated solutions for processing raw data over decades. Each piece of information about the events has been represented as a feature. Based on NOvA feature engineering algorithms, let us produce ten data with three particular features, as individuals present in Table 2.1.

	Cosmic ID	**Shower Energy**	**Number of Prongs**	**Interaction Type**
Event One	0.50	0.27	4	Neutral Current
Event Two	0.32	0.33	3	Neutral Current
Event Three	0.21	0.65	5	Neutral Current
Event Four	0.01	0.91	5	Neutral Current
Event Five	0.001	1.10	5	Neutral Current
Event Six	0.45	0.11	4	Cosmic Ray
Event Seven	0.67	0.09	2	Cosmic Ray
Event Eight	0.91	0.05	1	Cosmic Ray
Event Nine	0.87	0.01	1	Cosmic Ray
Event Ten	0.72	0.03	2	Cosmic Ray

TABLE 2.1: NOvA NC and Cosmic pseudo events for training MCP neuron toy models

We have learned from the MCP neuron proposal that it only takes boolean features as input, so we need to project the above feature values into the boolean format.

- For Cosmic ID: if the value is less than 0.5, we set it as o, otherwise, it is 1;
- For Shower Energy: if the value is greater than 0.1, we set it as o, otherwise, it is 1;
- Number of Prongs: if the value is greater than 3, we set it as o, otherwise, it is 1;

We also set 0 as the label for neutral current events, and 1 for the cosmic events. Table 2.2 presents our input dataset.

2.1.2 Supervised Binary Classification

Supervised binary classification is a type of machine learning algorithm where the labels are predefined and are used to categorize new probabilistic observations into two labeled groups. Each feature of the input data is a dimension of the feature space. In our case, there is three-dimension, as shown in Fig 2.4. Our goal is to find the best way to put NC and Cosmic events

	Cosmic ID	Shower Energy	Number of Prongs	Interaction Type
Event One	1	0	0	0
Event Two	0	0	1	0
Event Three	0	0	0	0
Event Four	0	0	0	0
Event Five	0	0	0	0
Event Six	0	0	0	1
Event Seven	1	0	1	1
Event Eight	1	1	1	1
Event Nine	1	1	1	1
Event Ten	1	1	1	1

TABLE 2.2: Binary format of the pseudo dataset

into two separate spaces by determining the best threshold, which minimizes the misclassification probability.

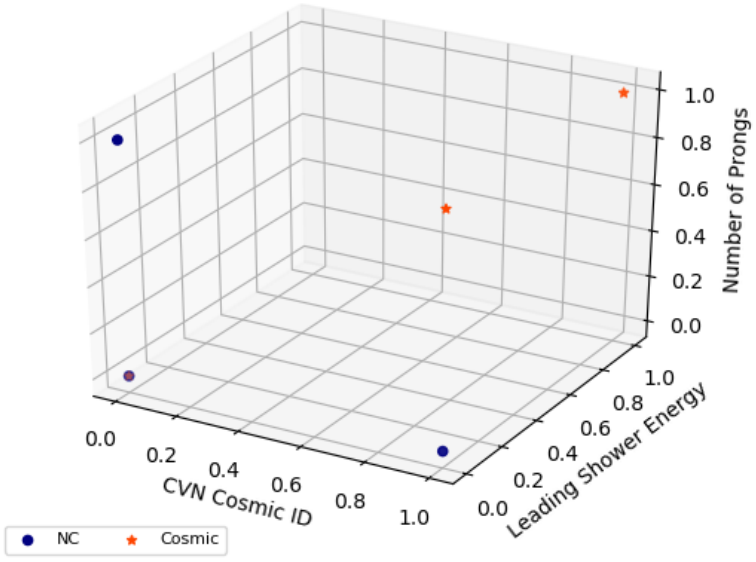

FIGURE 2.4: NC and Cosmic events distribution in feature space in the boolean format

2.1.3 Perform A MCP Neuron Classifier in Python

```
# First three rows are input features, the last row is event label.
dataset = [[1, 0, 0, 0], [0, 0, 1, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0],
           [0, 0, 0, 1], [1, 0, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]]
```

```
# Assuming no inhibitory input
def MCP(row):
     activation = 0
     for i in range(len(row) - 1):
         activation += row[i]
    # Assuming the threshold is 0
     return 1.0 if activation > 0.0 else 0.0

for row in dataset:
    prediction = MCP(row)
    print("True Information=%d, Model Prediction=%d" % (row[-1], prediction))
```

2.1.4 Limitations of The MCP Neuron Model

Running the above code in Jupyter Notebook, the result tells us that the prediction accuracy is 70%. It is not bad but not good, either. The classifier is not a complete model of human decision-making. But, the example illustrates how an artificial neuron makes decisions.

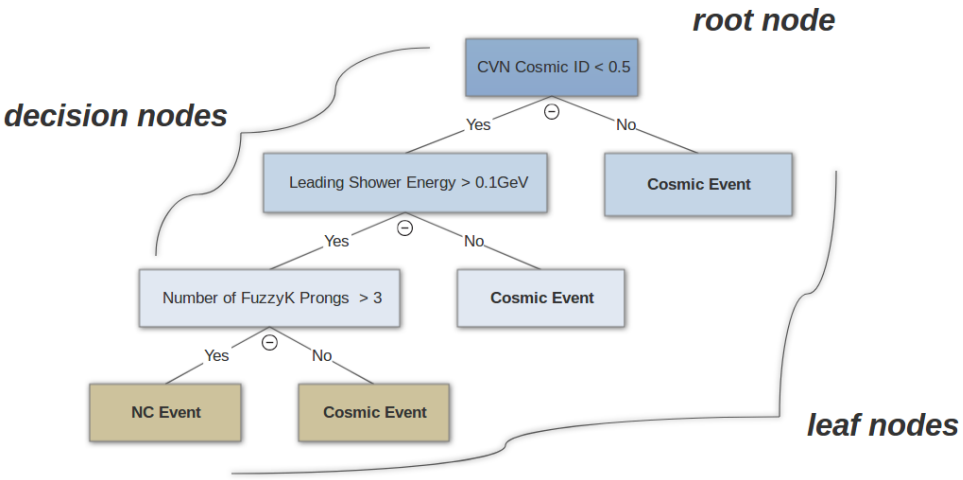

FIGURE 2.5: A binary decision tree classifier for neutral current and cosmic events

It is intriguing to compare the classification capability of neuron-based and tree-based algorithms. Based on the same splitting rules, a single MCP neuron and a decision tree, as presented in Fig 2.5, may get the same prediction accuracy. They both also suffer from the same significant limitations:

- They can not perform learning from input data;

- They can not represent the nonlinear functions;

Over the past decades, they have evolved into two distinct algorithm families to overcome the issues.

2.2 Relaxed Neurons and Perceptrons

MCP neuron, though very simple, is incredibly versatile and easy to modify. Accordingly, various variant models have produced to conquer the first defect.

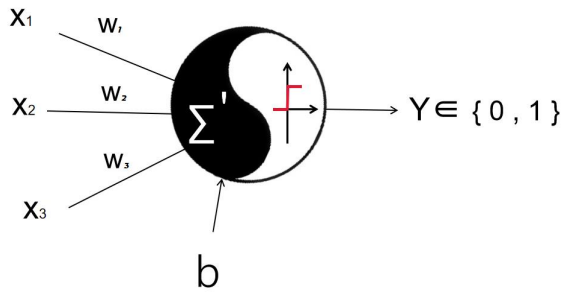

FIGURE 2.6: The schematic diagram of the Mark I Perceptron's neuron. Rosenblatt not only proposed the improved neuron model but also came up with a supervised learning algorithm which first time enable the artificial neuron to figure out the better variables (weights and bias) directly from input data by itself.

Neuronal Evolution I

A psychologist, Frank Rosenblatt, came up with the Mark I Perceptron, whose neuron is a significant improvement over the MCP neuron. By relaxing some of the MCP neuron's definitions, the artificial neuron is the first time to able to learn from data. The major differences are:

- The inhibitory input no longer applies;
- By applying each feature a different weight, features are allowed to have distinct influences onto the neuron's decision-making;
- Input feature's values are not restricted to be the boolean format;

The schematic diagram, Fig 2.6, shows the neuron has three inputs, though in general, it may take more or fewer inputs. The modified computational rules to make the decision is:

$$Y = \begin{cases} 1 & \text{if } \sum_i w_i x_i > threshold \\ \\ 0 & \text{if } \sum_i w_i x_i \leq threshold \end{cases} \tag{2.2}$$

Weights, the new variables, are real numbers that express the importance of the corresponding input features to the final decision. The neuron output is therefore decided by whether the weighted sum is higher than or less than the set threshold value. By varying the weights and the threshold, the neuron can produce different computational models of decision-making. Before diving into the deep learning fun

stuff, let us quickly simplify the above somewhat cumbersome equation by changing its notations. First, we move the threshold to the other side of the inequality and redefine it as a negative bias (b), $b \equiv -threshold$. Second, rewriting the $\sum_i w_i x_i$ as $\overrightarrow{w} \cdot \overrightarrow{x}$, where the $\overrightarrow{w}$ and $\overrightarrow{x}$ are vectors whose components are the weights and features, respectively. Based on the above modification, the neuron decision rules can be rewritten:

$$Y = \begin{cases} 1 & \text{if } \overrightarrow{w} \cdot \overrightarrow{x} + b > 0 \\ \\ 0 & \text{if } \overrightarrow{w} \cdot \overrightarrow{x} + b \leq 0 \end{cases} \tag{2.3}$$

The bias can be thought of as a measure of how easy the neuron can be fired. If the bias has a massive positive value, it is straightforward for the neuron to output a 1, and on the contrary, it is easy to get a 0 as the output. In our neuron classifier case, learning is the process of changing the two types of variables (weights and bias) to get a better classification result based on input data.

2.2.1 Perform The Improved Neuron Classifier in Python

```
# First three rows are input features, the last row is event label.
dataset = [[0.50,  0.27, 4, 0],  [0.32,  0.33, 3, 0],
           [0.21,  0.65, 5, 0],  [0.01,  0.91, 5, 0],
           [0.001, 1.10, 5, 0],  [0.45,  0.11, 4, 1],
           [0.67,  0.09, 2, 1],  [0.91,  0.05, 1, 1],
           [0.87,  0.01, 1, 1],  [0.720, 0.03, 2, 1]]


def Perceptron(row, weights):
        activation = 0
        for i in range(len(row)-1):
                activation += weights[i] * row[i]
        return 1.0 if activation > 0.0 else 0.0


# Set a weight for each input feature
weights = [0.8, -0.2, -0.1]


for row in dataset:
        prediction = Perceptron(row, weights)
        print("True Information=%d, Model Perdiction=%d" % (row[-1], prediction))
```

After trying several times, we set the bias as zero and select specific values for weights; Running the above code, we can get a better result than the MCP neuron one, the selection accuracy is 90%. Figure 2.7

presents the event distribution in the feature space. Tuning the weights and bias is the learning process, but not machine learning due to the neuron itself does not perform it. Before completing the perceptron learning, let us introduce the importance of the data representation first.

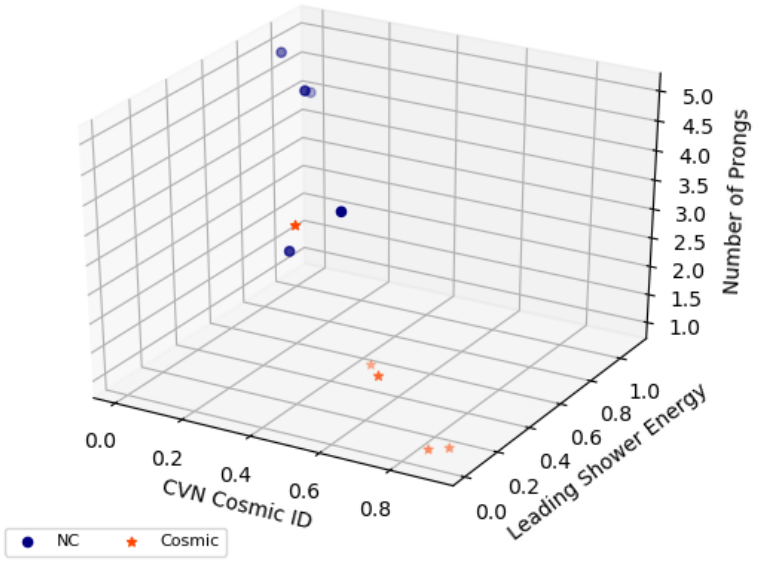

FIGURE 2.7: NC and Cosmic events distribution in three-dimension feature space

2.2.2 Representation Learning I

Comparing Fig 2.4 and Fig 2.7, we can see that the performance of these neuron models also relies on the representation of the interactions they are given. In general, classification models' performance profoundly depends on data representation due to the distinct representation methods that may highlight different valuable information, intrinsic structure, and explanatory factors of variation behind the data. We can quickly understand particle interactions in Feynman Diagrams but find it is more time-consuming when the same interactions were written in the path integral formulation. Since much of classification tasks can be solved by finding the right set of features, the majority of actual effort in applying classification algorithms go into the preprocessing pipeline design and data transformations based on various domain knowledge. This process, known as feature engineering, is essential, but hard to extract the discriminative information from the given data. To overcome this weakness, pioneers used machine learning to discover representation methods known as representation learning. Generally speaking, representation learning algorithms can find a better set of features than hand-designed representation methods; therefore, bring us better performance.