Dataset

The dataset we're going to work with is the **HIGGS dataset**, a popular dataset for particle physics, particularly useful for training machine learning models in the field of High-Energy Physics (HEP). The data was collected for the purpose of identifying a specific subatomic particle known as the **Higgs boson** among other particle collision events recorded in experiments. In these experiments, features from particle collisions, such as the particle's momentum, energy, and positional data, are analyzed to see if they match the characteristics of a Higgs boson.

Label	lepton_pT	lepton_eta	lepton_phi	missing_energy_magnitude	missing_energy_phi	jet1_pt
0	1.0	0.869293	-0.635082	0.225690	0.327470	-0.689993
1	1.0	0.907542	0.329147	0.359412	1.497970	-0.313010
2	1.0	0.798835	1.470639	-1.635975	0.453773	0.425629
3	0.0	1.344385	-0.876626	0.935913	1.992050	0.882454
4	1.0	1.105009	0.321356	1.522401	0.882808	-1.205349

Table 1: Partial view of the HIGGS dataset with selected features.

Key Aspects of the Dataset:

- **Label Column**: Contains the binary target label (1 or 0), where 1 indicates events likely to be a Higgs boson, and 0 indicates events that are unlikely to be a Higgs boson.
- **Feature Columns**: These columns represent measurements of particle physics features, such as:
  - **Momentum and energy** of various particles like leptons and jets (e.g., `lepton_pT`, `jet1_pt`).
  - **Spatial features** like η and ϕ, which describe angles and directions in cylindrical coordinates, crucial for interpreting particle movement patterns.
  - **Missing energy features** that represent the undetected energy in the collision, indicating neutrinos or particles that passed undetected.

This dataset is especially valuable for binary classification problems in machine learning, where the objective is to distinguish between Higgs boson events and non-Higgs events based on the collision feature measurements.

Structures of Neural Network

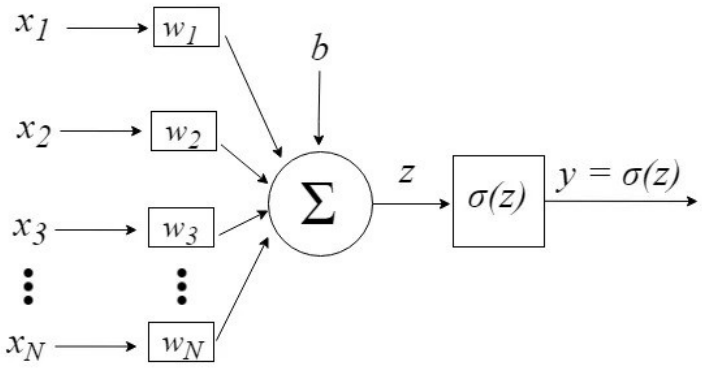

Figure 1: Computational model of a neuron

In the context of the Higgs Boson dataset, the values of array $X = [x_1, x_2, x_3, \ldots, x_n]$ correspond to the input features that describe physical properties from particle collisions. For example:

- x_1 could be **lepton_pT**, representing the transverse momentum of a lepton,
- x_2 might represent **lepton_eta**, which is the pseudorapidity of the lepton,
- x_3 could be **lepton_phi**, indicating the azimuthal angle of the lepton,

and so on, continuing through all the input features in the dataset up to **jet1_pt** (or beyond if using additional features). Each feature represents a measurable attribute of particles from the collision data and serves as an input for training or evaluating the network model.

As they are fed to the neuron, they are multiplied by their corresponding synaptic weights, which are the elements of the array $W = [w_1, w_2, w_3, \ldots, w_n]$, generating the value Z, normally called activation potential.

In neural networks, **activation potential** (often denoted as Z) is the combined input that a neuron receives after each input x_i is multiplied by its corresponding weight w_i and summed up, along with an added bias term b. This activation potential Z represents the neuron's total input, which is then passed through an **activation function** to produce the final output.

Mathematically, it's expressed as:

$$Z = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + b = \sum_{i=1}^{n} x_i w_i + b$$

This formula calculates the activation potential by combining all the weighted inputs, and it is central to determining the neuron's response to the inputs it receives.

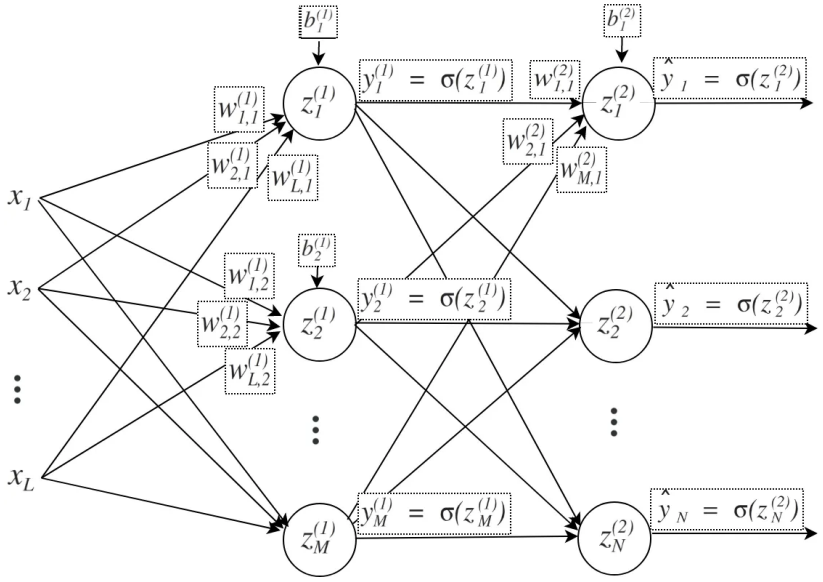

Figure 2: A Multilayer Neural Network.

Gradient Descent Method

Gradient descent is an optimization algorithm used to minimize the loss function by iteratively adjusting the model's parameters (weights and biases). The goal is to find the set of weights and biases that results in the lowest possible loss (i.e., the best predictions).

- **Loss Function:** In our context, the loss function is defined as:

$$L = (y - \hat{y})^2$$

  Here, y is our desired output, and $\hat{y}$ is the predicted output from the model. The loss measures how far off the predictions are from the actual values.

- **Weights and Biases:** Weights W and biases b are the parameters that the model learns during training. They are initially set to random values and are adjusted through the training process.

- **Gradients:** Gradients indicate the direction and rate of change of the loss function with respect to the weights and biases.

  For each weight $W^{(i)}$ and bias $b^{(i)}$, we calculate the partial derivatives:

$$\frac{\partial L}{\partial W^{(1)}}, \frac{\partial L}{\partial b^{(1)}}, \frac{\partial L}{\partial W^{(2)}}, \frac{\partial L}{\partial b^{(2)}}$$

  These derivatives tell us how much the loss L will change if we make a small change in the weights and biases.

Updating Method

Using the calculated gradients, we update the weights and biases to minimize the loss. The update rules can be expressed as follows:

$$W^{(i)} \leftarrow W^{(i)} - \alpha \cdot \frac{\partial L}{\partial W^{(i)}}$$

$$b^{(i)} \leftarrow b^{(i)} - \alpha \cdot \frac{\partial L}{\partial b^{(i)}}$$

Where:

- α is the learning rate, a parameter that controls the size of the step we take in the direction of the gradient.

- i indicates the layer (hidden or output layer).

How This Works

- **Direction of the Update:** The gradients $\frac{\partial L}{\partial W^{(i)}}$ and $\frac{\partial L}{\partial b^{(i)}}$ indicate the direction in which the loss function is increasing. By subtracting these gradients, we effectively move in the direction of the **minimum** that reduces the loss.

- **Magnitude of the Update:** The learning rate α scales the updates. If the learning rate is too high, the updates might overshoot the minimum, while if it's too low, the convergence may be slow.

- **Iterative Process:** This process is repeated for many iterations (or epochs). With each update, the weights and biases are adjusted in a way that gradually pulls the loss towards a **minimum**. The model learns to improve its predictions over time.

Forward Pass

For an input vector X:

Hidden Layer Computation:

- Linear transformation:

$$Z^{(1)} = X \cdot W^{(1)} + b^{(1)}$$

- Activation with the sigmoid function:

$$A^{(1)} = \sigma(Z^{(1)}) = \frac{1}{1 + e^{-Z^{(1)}}} = y_m^{(1)}$$

Output Layer Computation:

- Linear transformation:

$$Z^{(2)} = A^{(1)} \cdot W^{(2)} + b^{(2)}$$

- Activation with sigmoid:

$$A^{(2)} = \sigma(Z^{(2)}) = \frac{1}{1 + e^{-Z^{(2)}}} = \hat{y_n}$$

Backpropagation

To minimize the loss $L = (y - \hat{y})^2$, we calculate gradients with respect to the weights and biases.

Output Layer Gradients:

- Error signal:

$$\frac{\partial L}{\partial Z^{(2)}} = -2(y - \hat{y}) \cdot \sigma'(Z^{(2)}) = \delta^{(2)}$$

- Weight gradient:

$$\frac{\partial L}{\partial W^{(2)}} = \left(\frac{\partial L}{\partial A^{(2)}}\right)\left(\frac{\partial A^{(2)}}{\partial Z^{(2)}}\right)\left(\frac{\partial Z^{(2)}}{\partial W^{(2)}}\right) = -2(y - A^{(2)})\sigma'(Z^{(2)})A^{(1)} = \delta^{(2)}A^{(1)}$$

- Bias gradient:

$$\frac{\partial L}{\partial b^{(2)}} = \left(\frac{\partial L}{\partial A^{(2)}}\right)\left(\frac{\partial A^{(2)}}{\partial Z^{(2)}}\right)\left(\frac{\partial Z^{(2)}}{\partial b^{(2)}}\right) = -2(y - A^{(2)})\sigma'(Z^{(2)}) = \delta^{(2)}$$

Hidden Layer Gradients:

- Error signal:

$$\frac{\partial L^{(1)}}{\partial Z^{(1)}} = \delta^{(1)}$$

- Weight gradient:

$$\frac{\partial L}{\partial W^{(1)}} = \left(\frac{\partial L}{\partial A^{(2)}}\right)\left(\frac{\partial A^{(2)}}{\partial Z^{(2)}}\right)\left(\frac{\partial Z^{(2)}}{\partial A^{(1)}}\right)\left(\frac{\partial A^{(1)}}{\partial Z^{(1)}}\right)\left(\frac{\partial Z^{(1)}}{\partial W^{(1)}}\right) = \delta W^{(2)}\sigma'(Z^{(1)})X = \delta^{(1)}X$$

- Bias gradient:

$$\frac{\partial L}{\partial b^{(1)}} = \delta^{(1)}$$

Weight and Bias Updates

Using learning rate α:

- Update weights and biases in each layer:

$$W^{(2)} \leftarrow W^{(2)} - \alpha \cdot \frac{\partial L}{\partial W^{(2)}}$$

$$b^{(2)} \leftarrow b^{(2)} - \alpha \cdot \frac{\partial L}{\partial b^{(2)}}$$

$$W^{(1)} \leftarrow W^{(1)} - \alpha \cdot \frac{\partial L}{\partial W^{(1)}}$$

$$b^{(1)} \leftarrow b^{(1)} - \alpha \cdot \frac{\partial L}{\partial b^{(1)}}$$

This process repeats for each epoch. For every batch in the dataset, the forward and backward passes compute and apply these gradients, progressively adjusting weights and biases to reduce the loss. After training, predictions are made by comparing $\hat{y}$ with a threshold.

1 Result and Training Overview

```
====================800000 | Loss: 135.65968941255008
Epoch: 1/5 | Loss: 135.65968941255008
====================
====================800000 | Loss: 135.65957818181818
Epoch: 2/5 | Loss: 135.65957818181818
====================
====================800000 | Loss: 135.65957818181818
Epoch: 3/5 | Loss: 135.65957818181818
====================
====================800000 | Loss: 135.65957818181818
Epoch: 4/5 | Loss: 135.65957818181818
====================
====================800000 | Loss: 135.65957818181818
Epoch: 5/5 | Loss: 135.65957818181818
====================
Accuracy: 0.47007954545454544
```

- **Training Samples:** The dataset used for training our neural network has 800,000 samples, which means the model is being trained on this many data points.
- **Epochs:** The model went through the entire dataset for a certain number of epochs (in this case, 5 epochs). During each epoch, the model processes all 800,000 samples.
- **Loss Calculation:** The loss value reported after each epoch reflects how well the model is performing on the training dataset of 800,000 samples. It indicates how well the model is minimizing the loss function based on the current weights and biases.

And we got an accuracy of 47.01% !!!

Code Repository

One can access the **Python Script** for this project here.