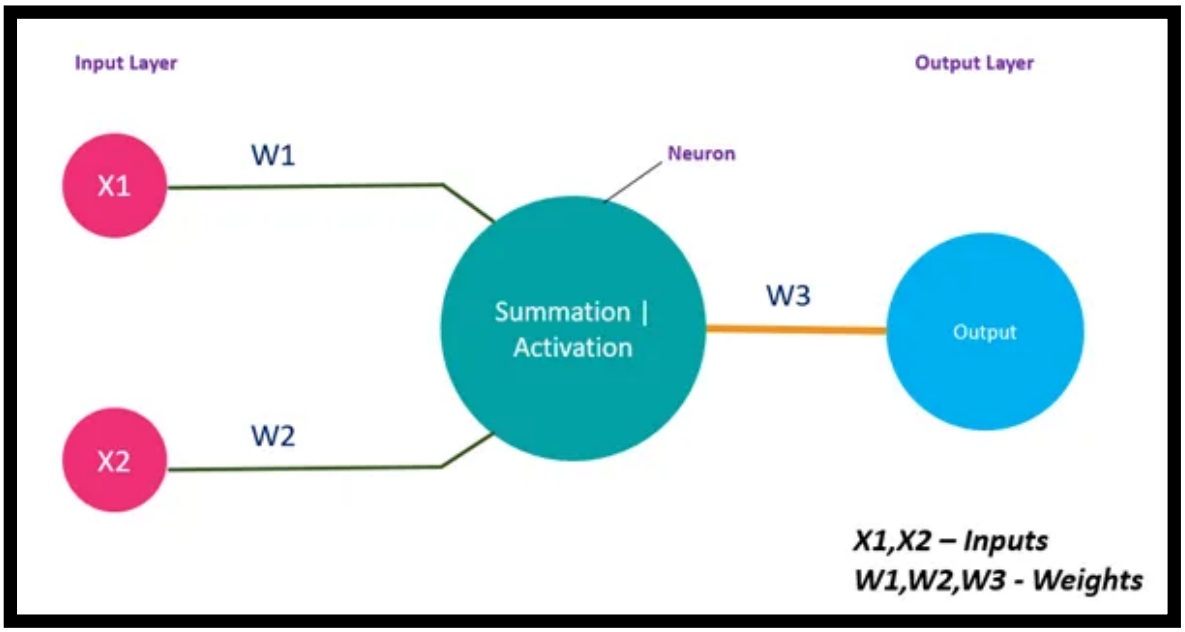
ANN NOTES

Perceptron:

A simple artificial neuron having an input layer and output layer is called a perceptron.
What does this Neuron contain?

- Summation function
- Activation function

The inputs given to a perceptron are processed by Summation function and followed by activation function to get the desired output.



After this we will look into the Mathematical representation of the inputs and outputs for the neuron

Mathematical Representation:

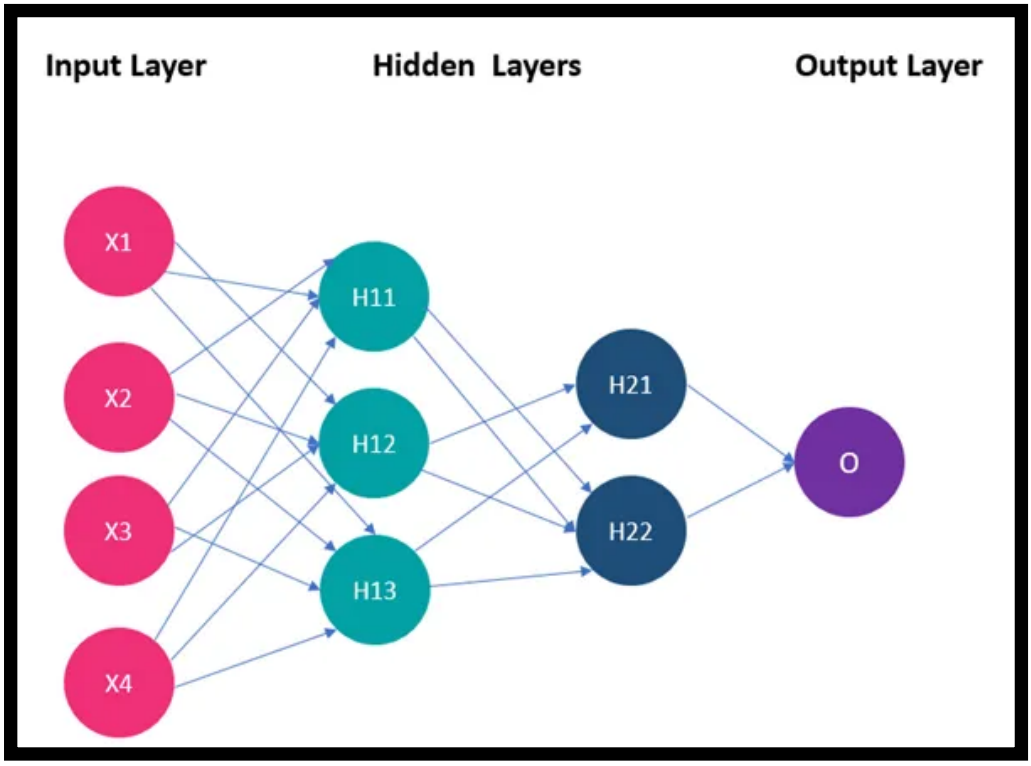
$$z = \sum_{i=1}^{n} w_i x_i + b$$

where w_i are weights, x_i are inputs, and b is the bias.

Example: For a neuron with inputs $x_1 = 0.5$, $x_2 = 0.3$, weights $w_1 = 0.8$, $w_2 = 0.4$, and bias $b = 0.2$:

$$z = (0.8 \cdot 0.5) + (0.4 \cdot 0.3) + 0.2 = 0.4 + 0.12 + 0.2 = 0.72$$

The entire structure of neural network would be:



Working of ANN:

1st STEP: Forward propagation:

- Process of passing inputs through the network to get the output.
- Involves matrix multiplications and activation functions.
- **Example** for a single layer:

$$a = \sigma(Wx + b)$$

where W is the weight matrix, x is the input vector, b is the bias vector, and σ is the activation function.

- **Example**: For input vector $x = [0.5, 0.3]$, weight matrix $W = \begin{bmatrix} 0.8 & 0.4 \\ 0.6 & 0.9 \end{bmatrix}$, and bias vector $b = [0.2, 0.1]$:

$$z = Wx + b = \begin{bmatrix} 0.8 & 0.4 \\ 0.6 & 0.9 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.3 \end{bmatrix} + \begin{bmatrix} 0.2 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0.8 \cdot 0.5 + 0.4 \cdot 0.3 + 0.2 \\ 0.6 \cdot 0.5 + 0.9 \cdot 0.3 + 0.1 \end{bmatrix} = \begin{bmatrix} 0.72 \\ 0.79 \end{bmatrix}$$

ACTIVATION FUNCTION:

The main purpose of the activation function is to convert the weighted sum of input signals of a neuron into the output signal. And this output signal is served as input to the next layer.

We have talked about about activation function in very detail in the previous notes although in very brief they are of types:

Sigmoid

- Ranges from 0 and 1.
- A small change in x would result in a large change in y.
- Usually used in the output layer of binary classification.

Tanh

- Ranges between -1 and 1.
- Output values are centered around zero.
- Usually used in hidden layers.

RELU (Rectified Linear Unit)

- Ranges between 0 and max(x).
- Computationally inexpensive compared to sigmoid and tanh functions.
- Default function for hidden layers.
- It can lead to neuron death, which can be compensated by applying the Leaky RELU function.

WEIGHTS AND BIASES ADAPTATION

- **Initialization**: Weights are often initialized randomly, but certain strategies (like He, Xavier initialization) help improve convergence.
- **Learning**: Weights and biases are updated during training to minimize the loss function.

2nd STEP: Calculate Loss Function

After the forward propagation we would get the predicted values based on the preset weights now we would like to calculate errors based on the true values, that show how far are our predictions with respect to true values

Loss Function

Measures the difference between the predicted and actual values.

Common Loss Functions:

- **Mean Squared Error (MSE)**: For regression tasks.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  - **Example**: For actual values $y = [1, 0]$ and predicted values $\hat{y} = [0.8, 0.2]$,

$$\text{MSE} = \frac{1}{2}((1 - 0.8)^2 + (0 - 0.2)^2) = \frac{1}{2}(0.04 + 0.04) = 0.04$$

- **Cross-Entropy Loss**: For classification tasks.

$$\text{Cross-Entropy} = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

  - **Example**: For actual values $y = [1, 0]$ and predicted probabilities $\hat{y} = [0.8, 0.2]$,

$$\text{Cross-Entropy} = -(1 \cdot \log(0.8) + 0 \cdot \log(0.2)) = -\log(0.8) \approx 0.223$$

3**rd Step:** Backpropagation

- Algorithm for updating weights by calculating the gradient of the loss function w.r.t each weight. Like choosing the values of parameter that could give the minimum slope for example I had weight value ‘w’ of 4 and after the forward pass I learnt that it yielded some error so after that in backward propagation, I would update my ‘w’ to reduce the error in next iteration. After updating I would move even backwards to the previous neuron and repeat the same.

SO LETS RECALL THE STEPS:

- **Steps**:
  1. **Forward Pass**: Compute the output of the network.
  2. **Compute Loss**: Calculate the loss using the loss function.
  3. **Backward Pass**: Compute the gradient of the loss w.r.t each weight (using chain rule).
  4. **Update Weights**: Adjust weights using a learning rate.

$$W = W - \alpha \frac{\partial L}{\partial W}$$

$$b = b - \alpha \frac{\partial L}{\partial b} \qquad (\alpha - \textit{Learning Rate})$$

Optimisation Algorithms

- **Gradient Descent**: Standard method for updating weights.
  - Variants:
    - Stochastic Gradient Descent (SGD): Updates weights for each training example.
    - Mini-batch Gradient Descent: Updates weights for a batch of training examples.
    - Momentum: Accelerates SGD by adding a fraction of the previous update.
    - Adam (Adaptive Moment Estimation): Combines advantages of RMSProp and Momentum.