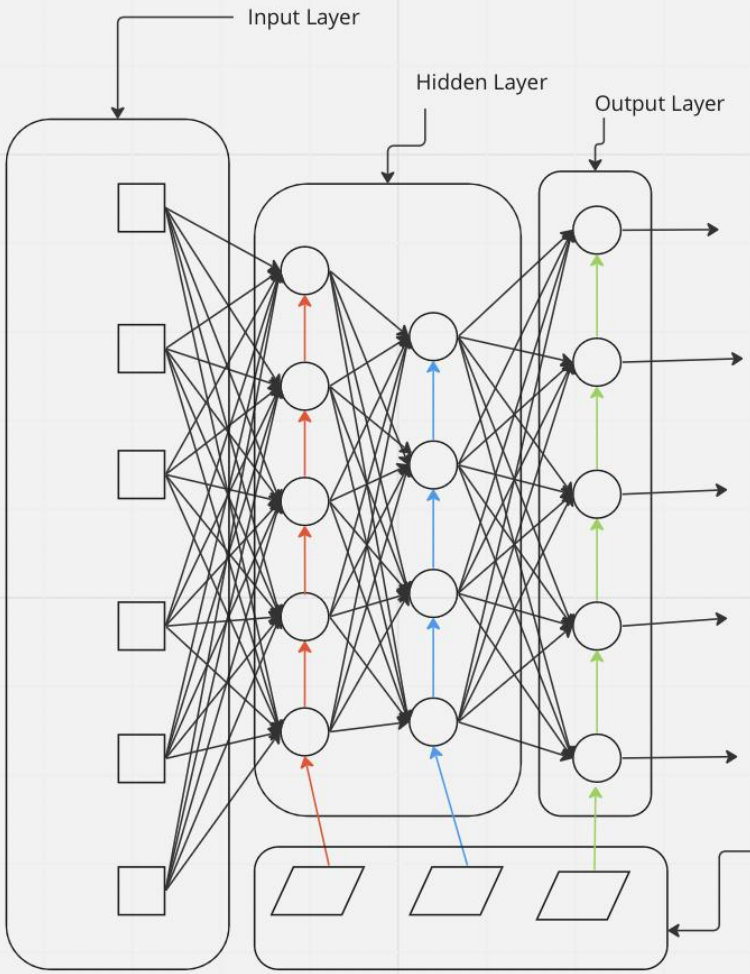
Artificial Neural Networks w/ Deep Layers (Fully Connected Layers)

ie a Multi-Layer Perceptron Model (MLP Model)

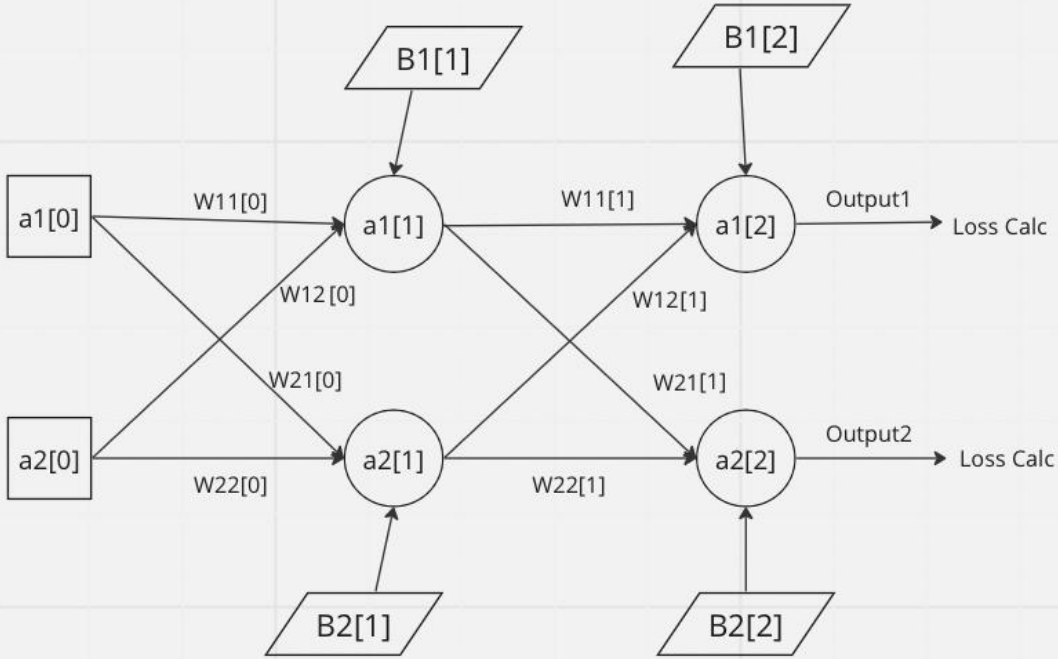
Feed Forward NN



Loss Calculation

Bias (to each neuron) in hidden & output layer

A simpler version of the NN above:



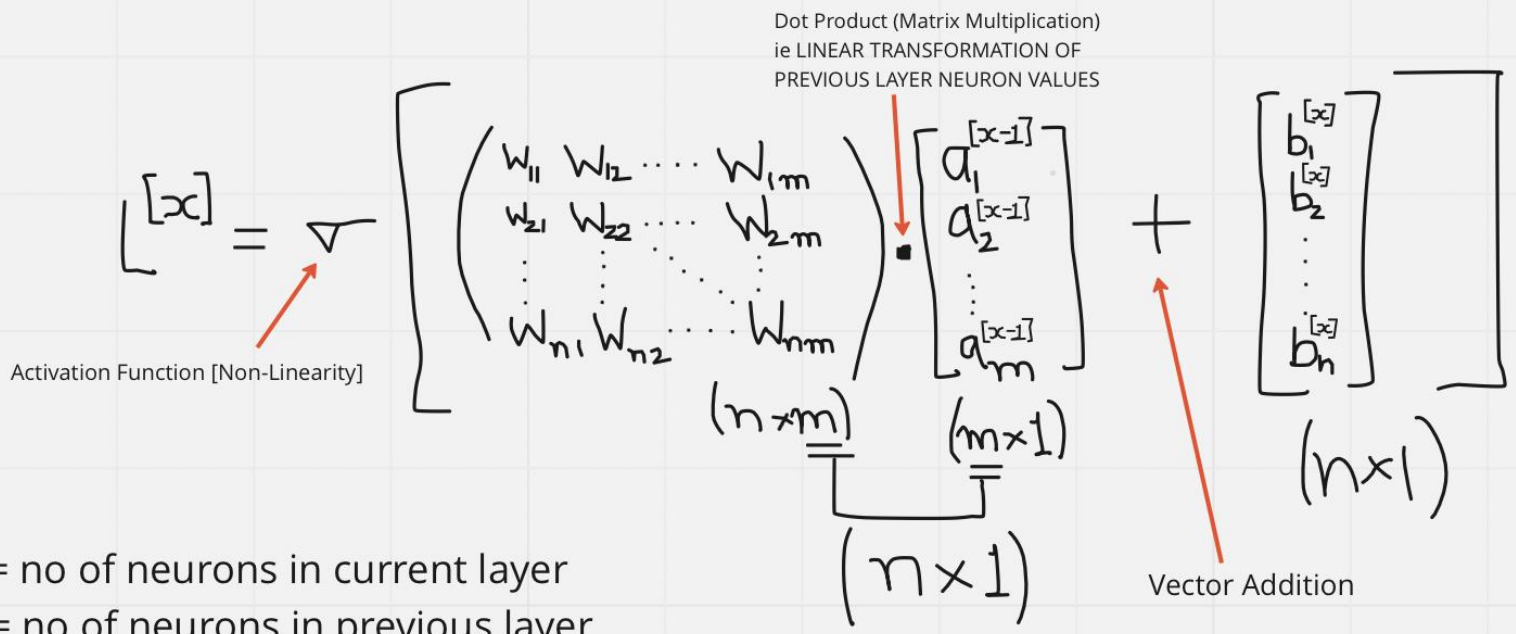
m=2 Input nodes in input layer, corresponding to 1 hidden layer with n1=2 neurons and 1 output layer with n2=2 neurons, each node in hidden and bias with it's own Bias node

Forward Pass

1. In ANNs with Deep Layers ie Fully connected Layers, each neuron in hidden/output layer is connected to all the neuron in the previous layer
2. At each neuron in Hidden & output, a linear value is calculated ie for N input values from previous layer, we get
   Neuron= weight1*input1+weight2*input2+.....+weightN*inputN + Bias
3. Bias node for provides a constant input value to the neurons in the hidden/output layers, allowing them to learn and model offsets or biases in the data. It helps the network in shifting the decision boundary and controlling the activation threshold of the neurons [It is usually the same value for each node in a layer]
4. At each neuron (in hidden/output layer only), an activation function is applied to the calculated value in the neuron to introduce non-linearity in the NN, so
   Neuron=activation funtion(weight1*input1+w2*i2+.....+weightN*inputN + Bias)
5. Non-Linearity (provided by Activation functions) is important because it allows us to find non-linear relations between input and final model output as well, otherwise Neural Network will only calculate a linear relation between them always
6. Hidden layer activation functions include sigmoid, hyperbolic tangent (tanh) and ReLU and Leaky ReLU etc, Output layer only activation function is Softmax, other functions like Sigmoid, tan h and ReLU can be used at output too
7. Activation functions decides whether the respective node will fire or not (ie if it's value will matter in further steps of output calculation or not), if it doesn't surpass a certain threshold, its value would be zeroed down by the activation function, killing its involvement in further calculation [weightX*0=0]
8. This process of Linear Transformations and Activation Function application is continued till we receive certain outputs on output layer where a single instance of Forward passing ends and we get our initial prediction/inference value

Forward Pass Calculation:

This is how values at layer L[x] is calculated ("x" here represents layer no in order, ie it's layer number x's value being calculated in our Forward Pass currently)



Here,

n = no of neurons in current layer

m = no of neurons in previous layer

Wij means weight between neuron “i” in current layer [x], and neuron “j” in previous layer [x-1]

$\sigma(z)$ here, can be any of the following activation functions

Perceptron $\phi = \begin{cases} 0, & z<0 \\ 1, & z \geq 0 \end{cases}$

Sigmoid $S = \frac{1}{1+e^{(-z)}} = \frac{e^z}{e^z+1}$

Hyperbolic Tangent $\tanh = \frac{e^z - e^{(-z)}}{e^z + e^{(-z)}}$

better than Sigmoid, but costlier to calculate

Rectified Linear Unit $ReLU = \max(0, z)$

the most common activation function in the hidden layers of NN

Leaky ReLU $\text{Leaky ReLU} = \max(\alpha z, z)$ for $0<\alpha<1$

when more outliers/noise in data

Exponential Linear Unit $ELU = \begin{cases} \alpha(e^z - 1), & z<0 \\ z, & z>0 \end{cases}$ for $0<\alpha<1$

Softmax $Softmax = \frac{e^{z_j}}{\sum_i e^{z_i}}$ For any arbitary Zj

FOR OUTPUT LAYER ONLY

Softplus $Softplus = \log(1+e^z)$

only gives +ve values