**Lecture Notes on Deep Learning**

**Neural Networks**

**Jawwad Shamsi and Rizwan Qureshi**

**Spring 2021**

**Multi-layer Perceptron**

Recap: Single output. previous lecture.

|  |
| --- |
|  |

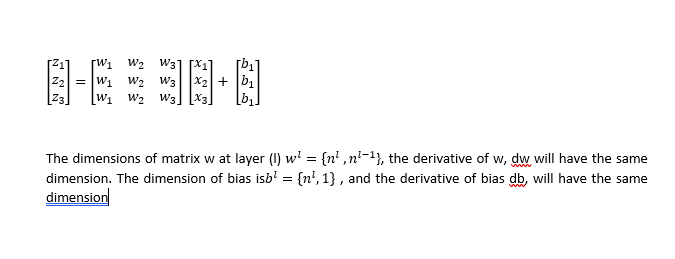
**Multi Output Perceptron**

All inputs are densely connected to all outputs.

It is a fully connected NN. Layers are also called as dense layers.

|  |
| --- |
|  |

Forward Propagation: In neural networks, we first calculate a linear value by multiplying the input layer with randomly initialized weights in the layer and than add a bias to it. After that an activation function is applied (sigmoid / relu) to add the non-linearity. This whole process is called the forward propagation.

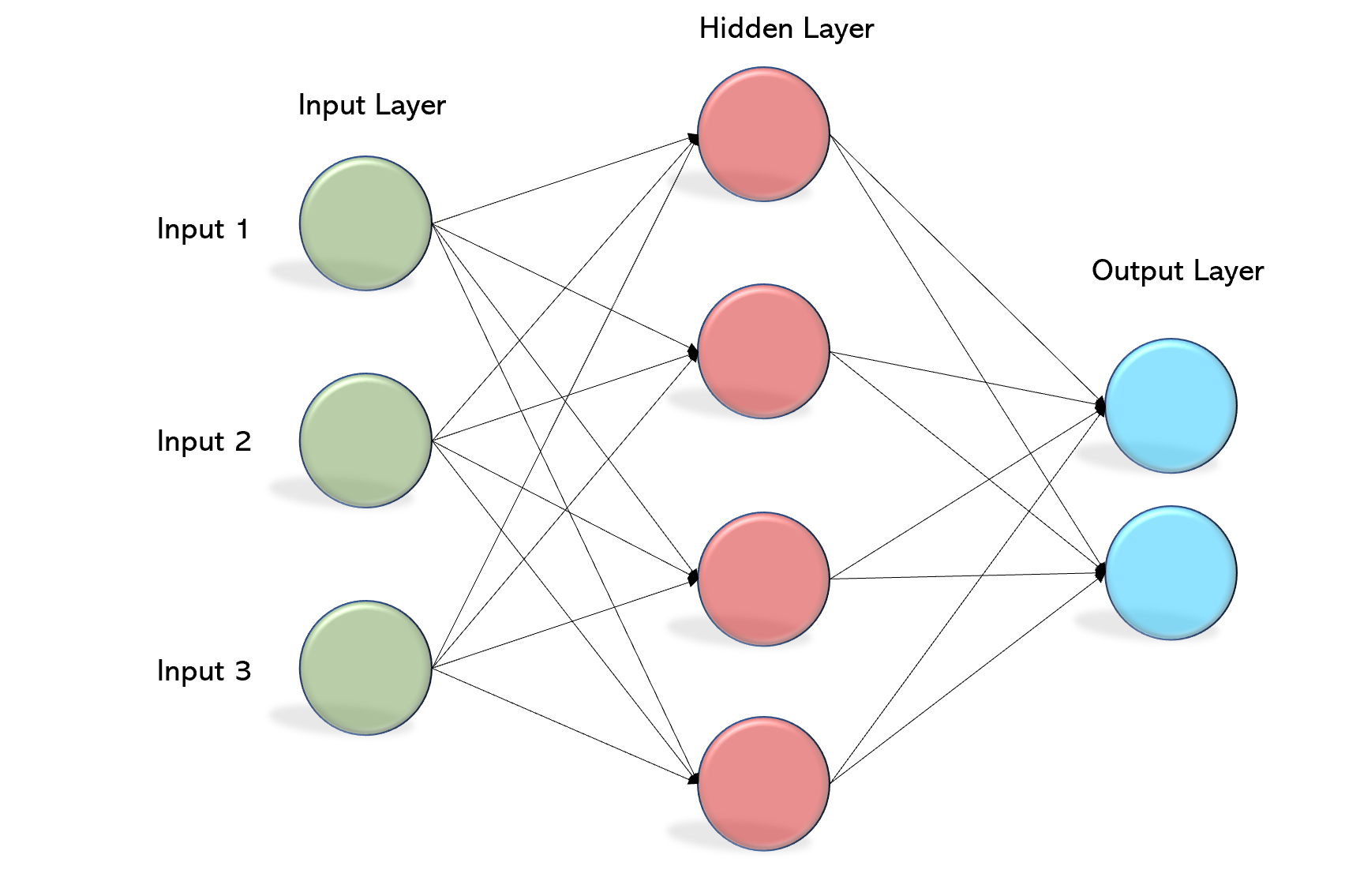


Once a forward propagation value is calculated it icompared with the actual value

Ȥi = W0,i +

initialize the two components. Weights and biases

dot product of inputs with weights , add a bias, activation function and compute output.



W(1) w(2)

Ȥi = W0,i + y=g(Ȥi = W0,i + )

w1 = weight matrices for the first layer and w2 = weight matrics for the second layer.

**How to tune parameters**

**Back Propagation**

Back propagation is a technique in which we adjust weights in the network in proportion to their contribution in determining the overall error. The main theme is that each weight can be adjusted such that the total error is minimal.

**Back propagation vs forward propagation**

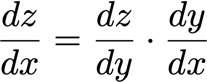
Forward propagation estimates output in correspondence with inputs and weights (and bias).

Backward propagation estimates weights in correspondence with error (actual output - estimated output).

**Chain rule:**







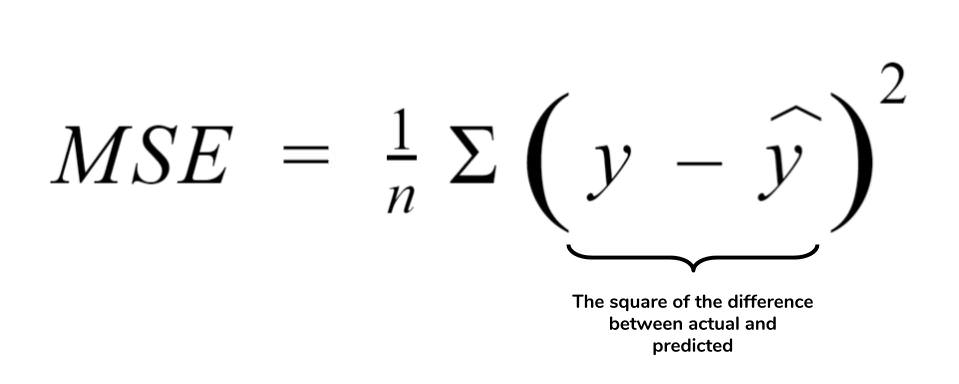
**Loss Function/ objective function/Cost function**

Minimize the error between the estimated output and actual output

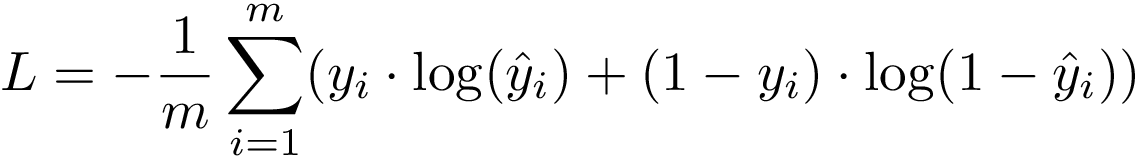
J(w), function of weights. We have to adjust weights in order to reduce the cost.

**Loss Functions:**

1. Mean Square Error (Regression)



1. Cross Entropy (Discrete Values)



where ‘m’ is the total no. of samples

**How to adjust weights**

Gradient Descent Algorithm

W ←W - ƞ

where

- ve sign, because we have to move in the opposite direction

next weight update is current weight – small amount of the learning rate multiply by the gradient.

learning rate : ƞ (Eta) how fast we want to do learning

how large the each step we take in practice to adjust the gradient.

The gradient tells us the direction. ƞ tells us the magnitude of direction.

Setting one value could be difficult.

**Example:**

Convex function: One minima.

Non-Convex function : more than one minima

Concave function: one maxima

Non Convex loss landcape J(W).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Local vs Global Minimum

**SGD:**

**Stochastic gradient Descent Algorithm**

For a large dataset, numerous resources are needed.

We divide data into small batches and compute local minima using SGD

Batch size represents data in the small batch

An epoch represents one complete iteration.

Scenarios : Size of Learning rate

1) We have a local minima. If we select learning rate to a too low value then the model can get stuck in local minima. It optimizes itself but it may optimizes to a non-optimal minima.

Converge very slowly as well.

2) select the learning rate too much. We may overshoot minima and diverge and escape the training process.

Challenge:

large enough to escape the local minima problem and small enough to converge.

Options

1) Try lots of different rates and see what works

2) Design an adaptive learning rate that adapts to the landscape

Forward Propagation vs Backward Propagation

Feed Forward Neural Network