⊙ we know that MLP builds model by finding right parameters of $w_0, w_1, \ldots w_n$ (weights). But, how does it find them? simply put,

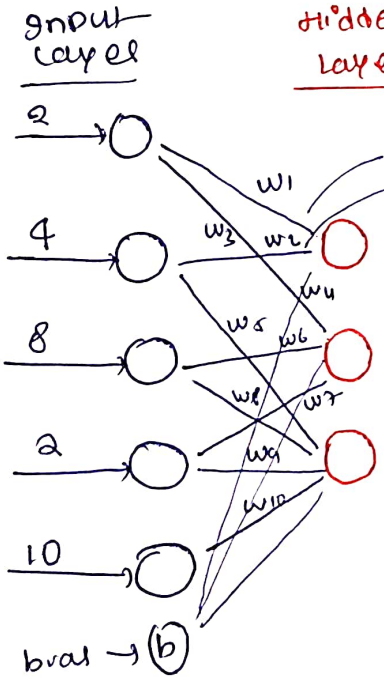
**How does training happen in MLP?**

→ To understand this clearly, let's take an example, say we have 5 features and a target variable.

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | label |
|---|---|---|---|---|---|
| 2 | 4 | 8 | 8 | 10 | I |
| 8 | 9 | 10 | 2 | 1 | I |
| 10 | 20 | 11 | 13 | 12 | II |
| 10 | 18 | 11 | 19 | 13 | II |

Let's try to solve this using MLP:

First, we need to create input layer, which has basically the same no. of neurons as dimension of feature. As we have 5 features, we need 5 neurons in input.




input to the neuron from this = $2w_1 + b$

input from this = $4w_2 + b$

total input to 1st hidden neuron = $2w_1 + 4w_2 + 2b$

→ If we take linear regression, it would try to find the relation b/w only y & x. But here, these hidden layers enable us to find the relation b/w each feature & the target. And these connections are labelled as $w_1, w_2 \ldots w_{10}$, which are also weights for the neurons.

→ So here we take various combinations of inputs to the neurons in hidden layers.

→ And if we take the first neuron in hidden layer, overall it is receiving input - $2w_1 + 4w_2 + b$,

Now, this input will be passed to step function, and then we will get the output from this neuron

If we think about this step, we wonder why pass it to a step function?

→ well, it doesn't necessarily has to be a step function, infact the function commonly used in this place is a sigmoid function.

→ Now, these are called activation functions, and there are many such functions such as ReLU, hypertangent etc. which will discussed further.

But here, we can think of what is the purpose of having one such activation function?

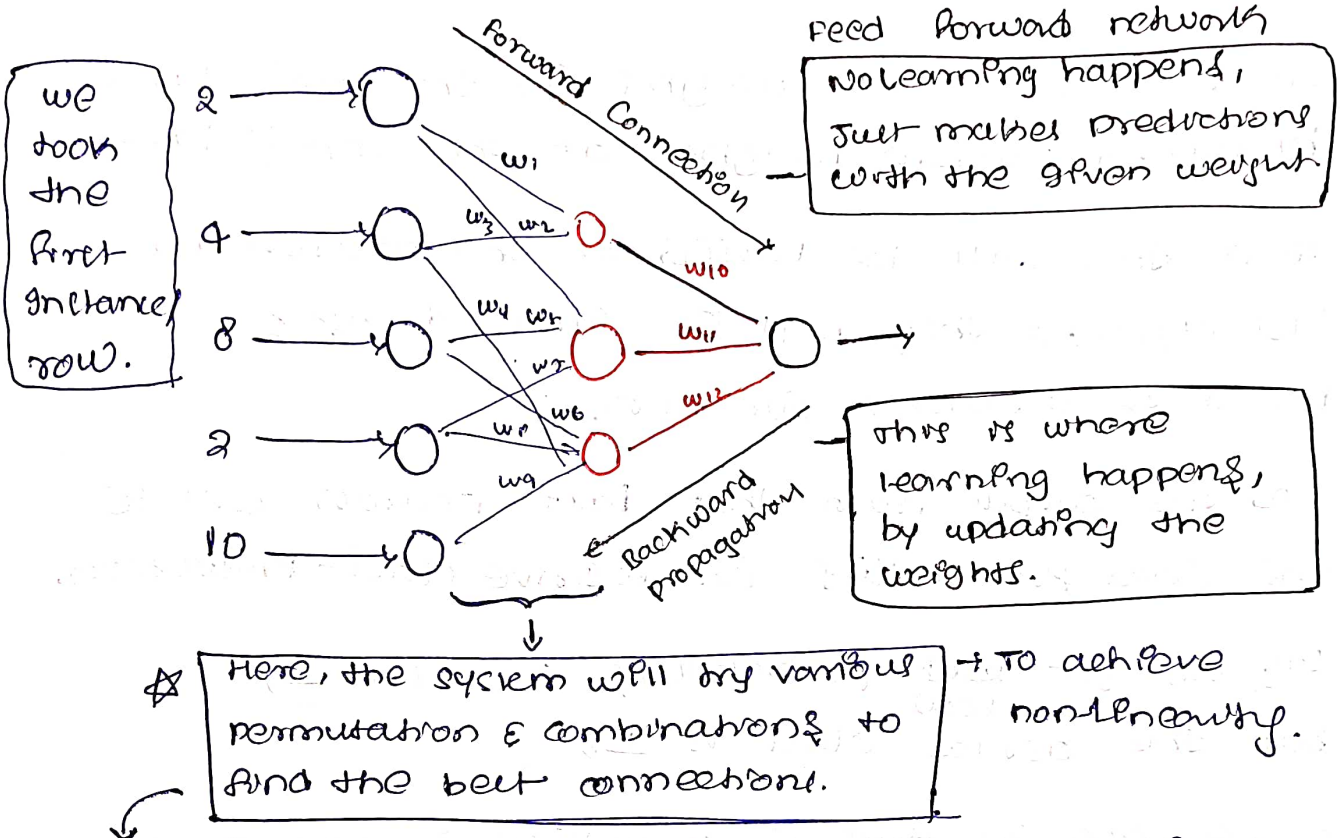
① It ensures that the output stays within a manageable range, just like normalization it keeps the output from reaching extremely high / low values.

② By keeping values in manageable range, so it allows the network to easily understand the complex or non-linear patterns.

③ output from the activation function will be passed as input to neurons in next layers. Basically, it activates the output of this neuron.

⊙ Now that we understood the purpose behind activation function, let's get back to solving/ understanding the example.

→ So the output from activation function, will become the output of this neuron & this output will be passed as input to the next layer.




✱ Here, the system will try various permutation & combinations to find the best connections. → To achieve non-linearity.

→ unlike LR, here we aim to find even the non-linear relations that some feature have with target. Some find need to try all combinations.

→ To proceed with the problem, we need weights. So, initially we will take all random weights & will the problem.

→ So let's take random weights to begin with, very say, $w_1 = 0.2$, $w_2 = 0.4$, $b = 1$.

Then, Eqn $\Rightarrow 2(0.2) + 4(0.4) + 1 = 3.7$ — ✱ will be passed through activation function.

→ say we got 3, 2, 4 at sum. Now we pass it to activation function.

$$S(3) = \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-3}} = 0.95 \rightarrow w_{10}$$

$S(2) = 0.88 \rightarrow w_{11}$

$S(4) = 0.98 \rightarrow w_{12}$

→ on this we can get weights for the next layer, by taking random weights at the input layer.

→ then this will be passed to the neuron in last layer, & they will be summed up & passed to activation function.

→ And the output from that last neuron will be the final probability for positive class - Prediction.

→ say, we got ~~high~~ low probability for class I, but the actual class is I.

→ so, we can say this is a mistake, and now we need to find the loss due to this mistake. we know that the because of the weights we ~~got~~ taken, we got this kind of issue, if the weights were properly given we wouldn't get mistake.

→ And the process so far is called the 'Forward propagation' where there isn't any learning happening, but just the predictions with random weights.

## ⊙ Backward Propagation

→ So far, we know that through forward propagation, we can make predictions using random weights, and with the obtained prediction we can find the loss.

→ Now, we need to find new value of w with by reducing the loss. Then by using optimizers we try to reduce the loss, that is by finding the best parameters for w.

→ So starting from last output, it will go back and find the weights for last layer say $w_{10}, w_{11}, w_{12}$. And so on., until it reaches the input layer.

→ Again, we send the data in forward direction, and now the predictions are made using the new weights.

→ If any mistake found, then again it will do backward propagation. In such way, the process continues until the loss become least.

→ Finally, we can simplify backward propagation as,

> It efficiently measures the error gradient across all the connection weights in the network by propagating the error gradient backward through the network.

Then, the algorithm performs a Gradient descent step to update all the connection weights in the network, using the error gradients it computed.