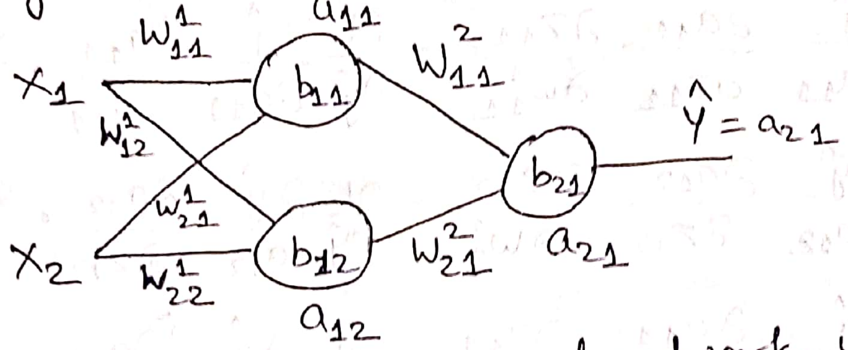
# Weight initialization

Weight initialization is a crucial step in training deep neural networks because it sets the starting point for learning. Proper initialization can help avoid problems like gradient exploding/vanishing and ensure faster convergence. Here are some common ways in which weights can be initialized—

(1) Zero initialization ⇒ when we initialize the weights with a constant value 0. Then for different activation functions, different scenarios can occur.
For eg. let's take a neural network as shown below.




(i) Let's suppose for this neural network, we take activation function as **ReLU**, we know for ReLU,

$$a_{11} = \max(0, z_{11})$$

where, $z_{11} = W_{11}^1 x_1 + W_{21}^1 x_2 + b_{11}$ {∵ weights and bias are initialized with 0}

$$z_{11} = 0$$

similarly, $a_{12} = \max(0, z_{12})$

where, $z_{12} = W_{12}^1 x_1 + W_{22}^1 x_2 + b_{12} = 0$

Thus $\boxed{a_{11} = a_{12} = 0}$

In our weight update rule,

$$\boxed{W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}}$$

For weight, $W_{11}^1$

$$W_{11new}^1 = W_{11old}^1 - \eta \frac{\partial L}{\partial W_{11old}^1}$$

Now, $\frac{\partial L}{\partial W_{11old}^1} = 0$, since its activation is 0.

Hence, $W_{11new}^1 = W_{11old}^1$

Thus, no training happens when we initialize weights with 0 for ReLU activation $f^n$.

(ii) Let's suppose, we make use of the tanh activation function

$$a_{11} = \frac{e^{z_1} - e^{-z_1}}{e^{z_1} + e^{-z_1}} = \frac{e^0 - e^0}{e^0 + e^0} = \frac{1-1}{1+1} = 0$$

Similarly, $a_{12} = 0$

Here again, no training will happen, similar to ReLU.

(iii) Now, we may use the sigmoid activation function.

$$a_{11} = \sigma(z_{11}) = 0.5 \quad , \quad a_{12} = \sigma(z_{12}) = 0.5$$

$$\Rightarrow \boxed{a_{11} = a_{12} = 0.5}$$

Now, $$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{11}} \frac{\partial a_{11}}{\partial z_{11}} \frac{\partial z_{11}}{\partial w_{11}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{11}} \frac{\partial a_{11}}{\partial z_{11}} . X_1$$

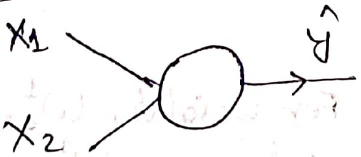
$$\frac{\partial L}{\partial w_{12}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{12}} \frac{\partial a_{12}}{\partial z_{12}} \frac{\partial z_{12}}{\partial w_{12}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{a_{12}} . \frac{\partial a_{12}}{\partial z_{12}} . X_2$$

$$\frac{\partial L}{\partial w_{21}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{11}} \frac{\partial a_{11}}{\partial z_{11}} \frac{\partial z_{11}}{\partial w_{21}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{12}} . \frac{\partial a_{11}}{\partial z_{11}} . X_2$$

$$\frac{\partial L}{\partial w_{22}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{12}} \frac{\partial a_{12}}{\partial z_{12}} \frac{\partial z_{12}}{\partial w_{22}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{12}} . \frac{\partial 12}{\partial z_{12}} . X_2$$

Thus, $\frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial w_{12}^1}$ and similarly, $\frac{\partial L}{\partial w_{21}^1} = \frac{\partial L}{\partial w_{22}^1}$

Hence, both the branches from first input act like one and both the branches from second input acts like one.




Thus, no matter how many neurons we have in the hidden layer, it acts as a single neuron and thus it behaves like a perceptron and can only learn linear relationship between data.

Thus, in case we use zero weights and bias with sigmoid activation function. Our neural network works as a perceptron and won't be able to learn any non-linear pattern in the data no matter how many neurons we use in the hidden layer.

(2) Constant equal weights initialization

In case we initialize the weights and biases with equal non-zero weights & biases, again the neural networks starts acting as a perceptron as we saw in the last example.

(3) Random Initialization

(i) When we use very small weights and biases initialized randomly then in case of tanh it suffers from vanishing gradient problem, for sigmoid as well it may suffer from vanishing gradient
its convergance will be very slow.
-t problem or otherwise its convergance will be very low.
Similarly for ReLU, convergance speed will be very low.

(ii) When we use very large weights and biases initialized randomly then in case of tanh and sigmoid the activation value saturates in either positive or negative directions because of this either training/convergence will be slow or in worst case vanishing gradient problem will occur.
In case of ReLU activation function, since it is non-saturating in the positive direction, the value of activation in positive direction will be large thus, gradients will also be larger and hence during weight update it will cause larger jumps because of which weight update will be unstable and we will not be able to converge at the global minima.

(4) Xavier (Glorot) Initialization

In this weights are drawn from a distribution with variance dependent on the number of input and output units.

Xavier Uniform $\Rightarrow W \sim U\left[-\sqrt{\frac{6}{fan\_in + fan\_out}}, \sqrt{\frac{6}{fan\_in + fan\_out}}\right]$

Xavier normal $\Rightarrow W \sim N\left(0, \frac{2}{fan_{in} + fan_{out}}\right)$

→ It is mostly used with tanh and sigmoid activation $f^n$.

(5) He (Kaiming) Initialization

It is mostly used with ReLU activation $f^n$ and its variants.

He normal $\Rightarrow W \sim N\left(0, \frac{2}{fan_{in}}\right)$

(6) LeCun Initialization It is used with Leaky ReLU.

LeCun normal $\Rightarrow W \sim N\left(0, \frac{1}{fan_{in}}\right)$