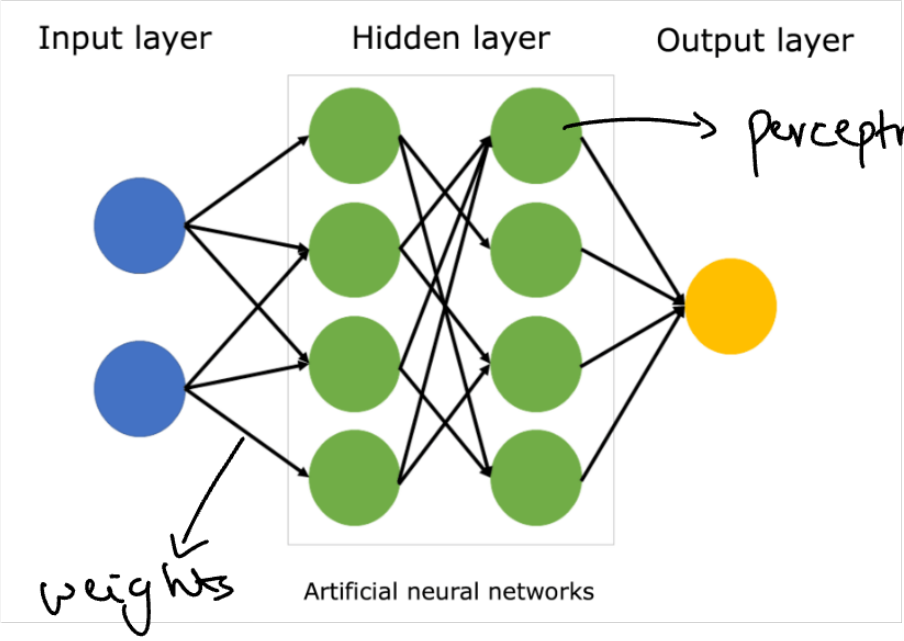
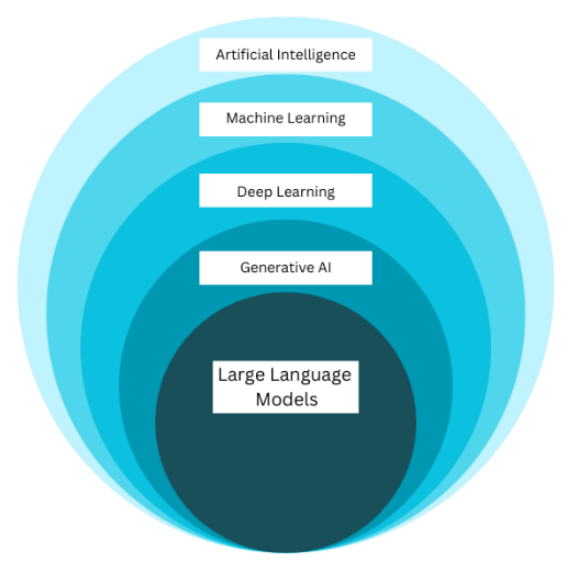
Deep Learning

deep learning is a subfield of AI and ML that is inspired by structure of human brain.

Deep learning algorithm attempt to draw similar conclusions as humans would by continually analyzing data with a given logical structure called Neural Network.





DL is part of a broader family of ml methods based on ANN with representation learning

↓

auto feature extraction

Deep learning algorithms uses multiple layers to progressively extract higher-level features from the raw input. for ex, in image processing, lower layers may identify edges (primitive features), while higher layers may identify concepts relevant to a human such as digits or letters or faces (complex features)

In DL, we do not need feature extraction because the architecture is designed in a way the we dont need to extract feature.

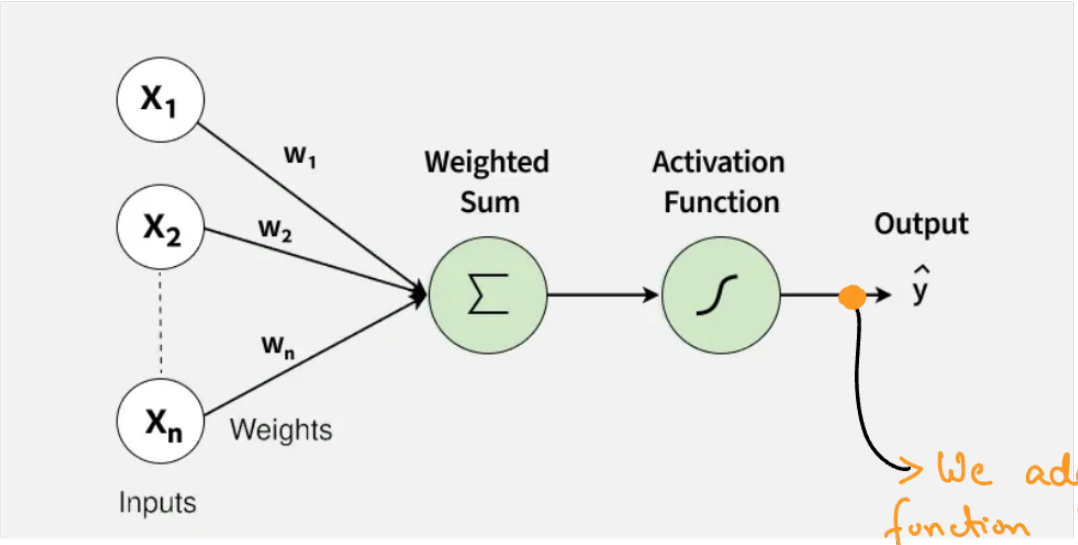
Deep Learning vs Machine Learning

1. Data Dependency : The more data provided to dL, the more better it gets whereas in mL, after some threshold, the performance remains stegnant.
2. Hardware Dependency : GPU are required for dL model training because of alot of complex matrix operations whereas, in ML, it's not required
3. Training Time : DL has very high training time while mL is very less
4. Feature Selection : No need of this in dL because of representative learning where dL layers automatically extract features from data whereas for mL, features are provided prior training.
5. Interpratibility : DL models are generally black box models which is a downside.

Perceptron

A perceptron is a very simple model of a neuron used for binary classification.

Simply, perceptron takes several inputs, multiplies each by a weight, adds them up together (including bias) and then passes the result through an activation function to output either 0 or 1.



The biggest drawback of a perceptron is that it only is valid for binary output which is why it only works on

Linear data.

Perceptron trick

The simple mental model:

1. Randomly pick up a point x, compute score $z = w \cdot x + b$
2. Check sign: $z \geq 0 \longrightarrow$ predict +1 else 0
3. Compare it to true label y:
   - if correct then skip
   - Wrong then change the boundary (shift line)
4. Repeat for all points until mostly correct.

∴ Start with a random line

↓

pick a point

↓

if misclassified point

↓

push line towards correct side

MODEL	LOSS FUNCTION	FORMULA	ACTIVATION	OUTPUT	TASK
Perceptron	Hinge Loss	$L = \max(0, 1 - y \cdot \hat{y})$	Step	{-1, +1} Hard labels	Binary Classification
Logistic Regression	Log-Loss (Binary Cross-Entropy)	$L = -[y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})]$	Sigmoid	{0, 1} Probability → class	Binary Classification
Softmax Regression	Categorical Cross-Entropy	$L = -\Sigma\, y_k \cdot \log(\hat{y}_k)$	Softmax	$[0, 1]^K$ Probability distribution	Multiclass Classification
Linear Regression	MSE (Mean Squared Error)	$L = (1/n)\ \Sigma\ (y - \hat{y})^2$	Linear	$\mathbb{R}$ Continuous values	Regression

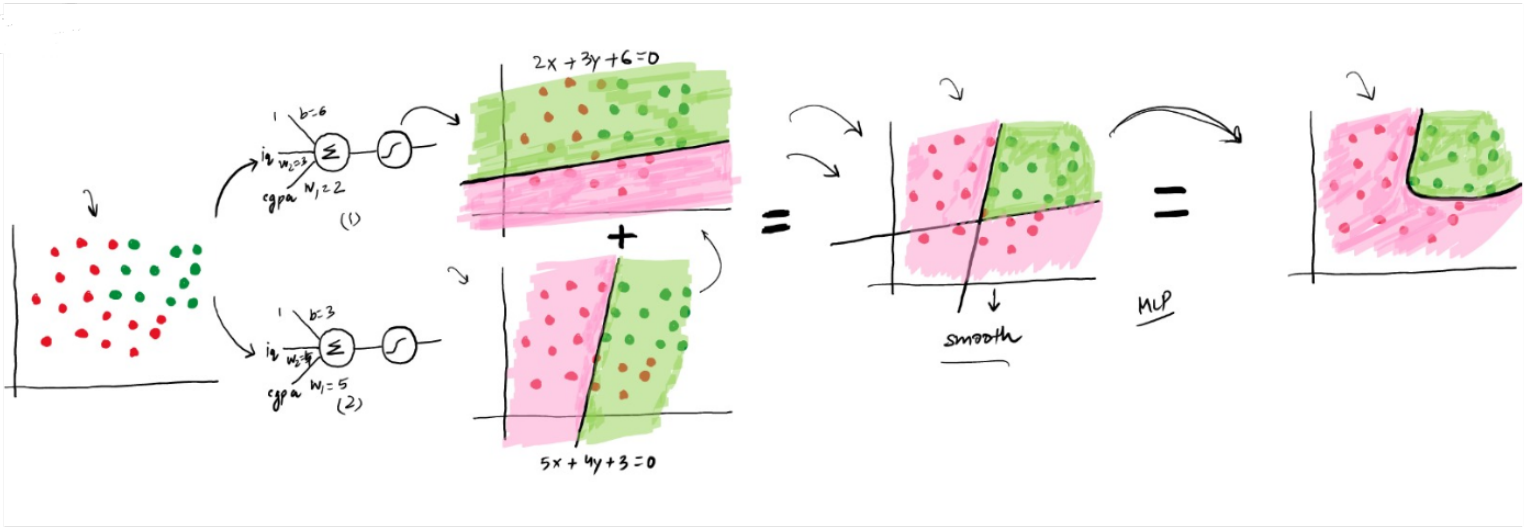
If we change the activation function or loss function or both, perceptron will behave differently.

What's the problem with perceptron?

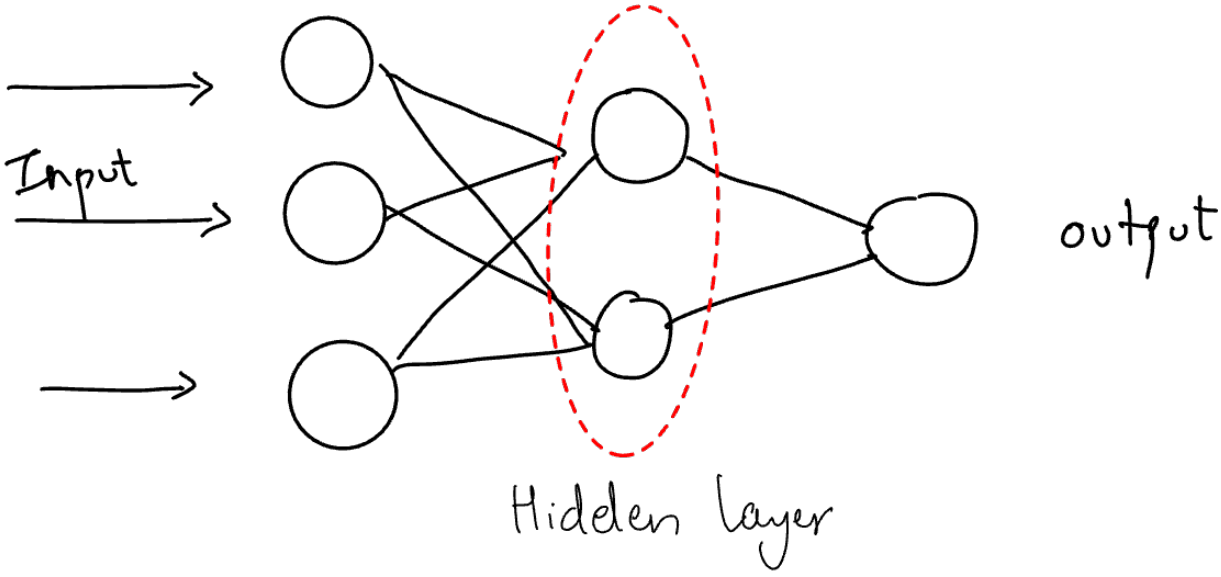
→ It only works on linear data and not on non-linear data

Multi-Layer Perceptron : MLP

Multi-layer perceptrons, or MLPs, are a powerful member of the Artificial Neural Networks family that can be used to solve complex problems that a single perceptron alone cannot. At their core, MLPs are complex and it has a collection of interconnected single perceptrons, also known as neurons or nodes, working together to process and analyze data.



Here as we can see, outputs of perceptrons are again fed as an input to another perceptron. This layering is called MLP



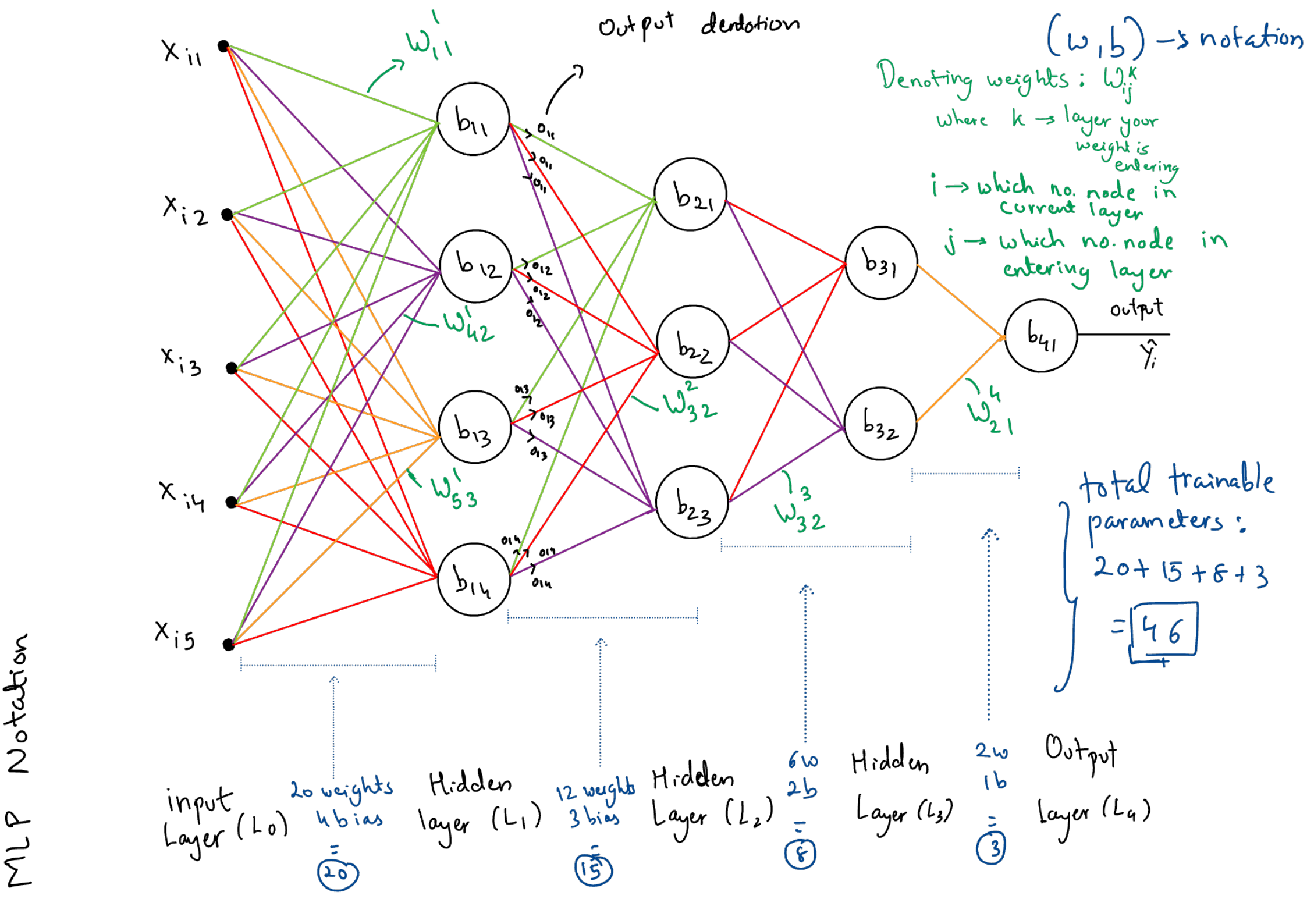
Each perceptron is called nodes when stacked in a neural network architecture.

There are four ways to improve the outputs.

1. Add more nodes to the Hidden Layer
2. Add more nodes to the input layer (This is when there are more input cols, more nodes are present by default)
3. Adding Nodes in Output
4. Adding More Hidden Layers

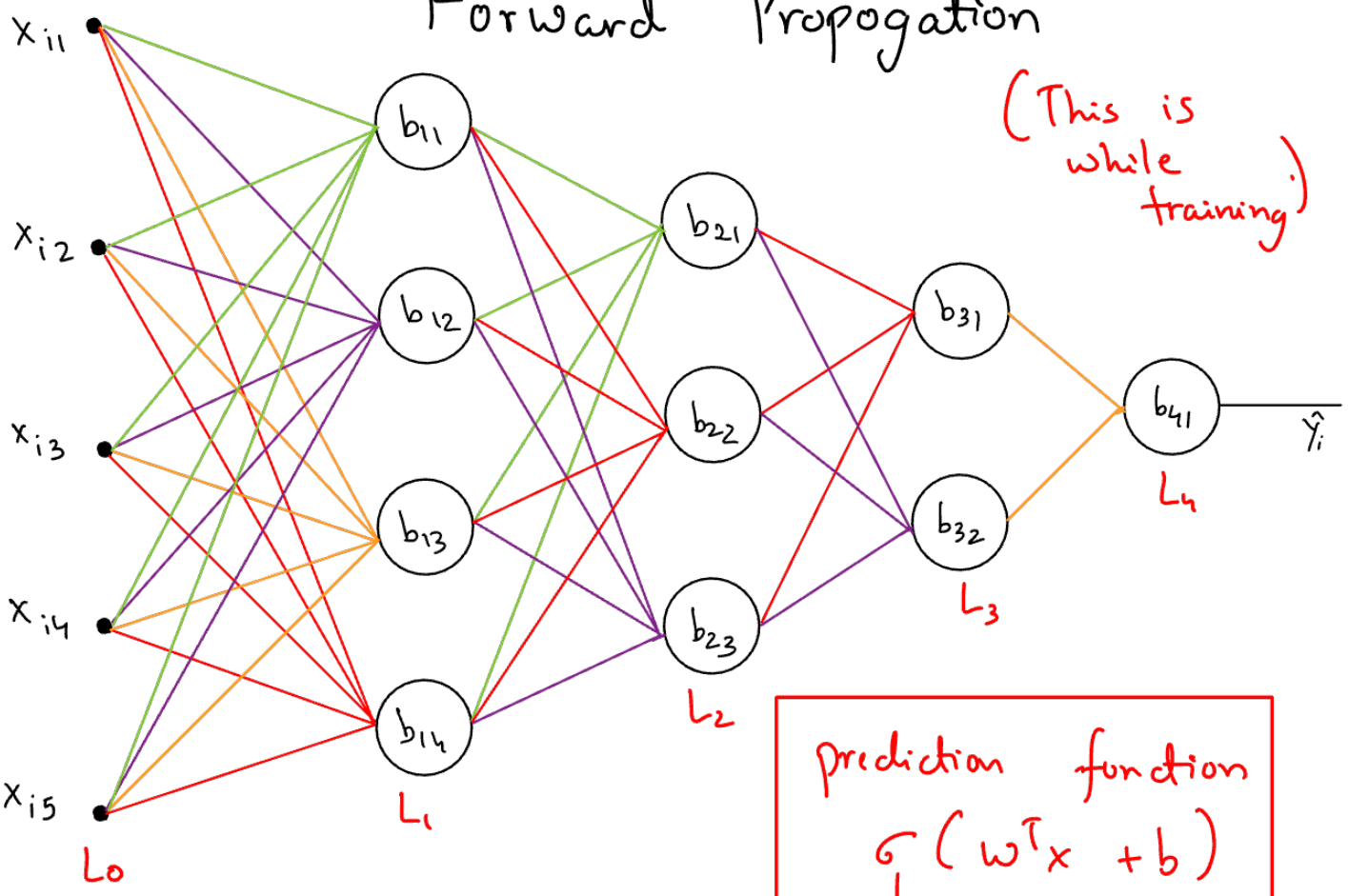
↓

Deep Neural Network


MLP Notation
Output denotation
(w, b) -> notation
Denoting weights : W_{ij}^{k}
where k -> layer your weight is entering
i -> which no. node in current layer
j -> which no. node in entering layer
x_{i1}
x_{i2}
x_{i3}
x_{i4}
x_{i5}
b_{11}
b_{12}
b_{13}
b_{14}
b_{21}
b_{22}
b_{23}
b_{31}
b_{32}
b_{41}
output
$\hat{y}_i$
W_{11}^{1}
W_{42}^{1}
W_{53}^{1}
W_{32}^{2}
W_{32}^{3}
W_{21}^{4}
o_{11} o_{12} o_{13} o_{14}
o_{21} o_{22} o_{23}
o_{31} o_{32}
total trainable parameters :
20 + 15 + 6 + 3
= 46
input Layer (L_0)
20 weights 4 bias = 20
Hidden layer (L_1)
12 weights 3 bias = 15
Hidden Layer (L_2)
6w 2b = 8
Hidden Layer (L_3)
2w 1b = 3
Output layer (L_4)

Forward Propogation

(This is while training)



prediction function

$\sigma(w^T x + b)$

Sigmoid activation func

(first row is passed)

Layer 1:

$$\underset{5\times4}{\begin{bmatrix} w'_{11} & w'_{12} & w'_{13} & w'_{14} \\ w'_{21} & w'_{22} & w'_{23} & w'_{24} \\ w'_{31} & w'_{32} & w'_{33} & w'_{34} \\ w'_{41} & w'_{42} & w'_{43} & w'_{44} \\ w'_{51} & w'_{52} & w'_{53} & w'_{54} \end{bmatrix}}_{w} \underset{5\times1}{\begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ x_{i4} \\ x_{i5} \end{bmatrix}}_{x} + \underset{4\times1}{\begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \\ b_{14} \end{bmatrix}}_{b}$$

Transposing $w \longrightarrow w^T$

$$\underset{4\times5}{\begin{bmatrix} w'_{11} & w'_{21} & w'_{31} & w'_{41} & w'_{51} \\ w'_{12} & w'_{22} & w'_{32} & w'_{42} & w'_{52} \\ w'_{13} & w'_{23} & w'_{33} & w'_{43} & w'_{53} \\ w'_{14} & w'_{24} & w'_{34} & w'_{44} & w'_{54} \end{bmatrix}}$$

Multiplying W^T and $x \rightarrow W_T X$

$[4\times3]\ [5\times1] \rightarrow [4\times1]$

$$\begin{bmatrix} w'_{11}x_{i1} + w'_{21}x_{i2} + w'_{31}x_{i3} + w'_{41}x_{i4} + w'_{51}x_{i5} \\ w'_{12}x_{i1} + w'_{22}x_{i2} + w'_{32}x_{i3} + w'_{42}x_{i4} + w'_{52}x_{i5} \\ w'_{13}x_{i1} + w'_{23}x_{i2} + w'_{33}x_{i3} + w'_{43}x_{i4} + w'_{53}x_{i5} \\ w'_{14}x_{i1} + w'_{24}x_{i2} + w'_{34}x_{i3} + w'_{44}x_{i4} + w'_{54}x_{i5} \end{bmatrix}_{4\times1} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}_{4\times1}$$

$$\begin{bmatrix} w'_{11}x_{i1} + w'_{21}x_{i2} + w'_{31}x_{i3} + w'_{41}x_{i4} + w'_{51}x_{i5} + b_1 \\ w'_{12}x_{i1} + w'_{22}x_{i2} + w'_{32}x_{i3} + w'_{42}x_{i4} + w'_{52}x_{i5} + b_2 \\ w'_{13}x_{i1} + w'_{23}x_{i2} + w'_{33}x_{i3} + w'_{43}x_{i4} + w'_{53}x_{i5} + b_3 \\ w'_{14}x_{i1} + w'_{24}x_{i2} + w'_{34}x_{i3} + w'_{44}x_{i4} + w'_{54}x_{i5} + b_4 \end{bmatrix}$$

$\downarrow$

$W^T x + b \rightarrow$ Now apply Sigmoid activation function $\sigma(W^T x + b)$

$$\sigma\left(\begin{bmatrix} w'_{11}x_{i1} + w'_{21}x_{i2} + w'_{31}x_{i3} + w'_{41}x_{i4} + w'_{51}x_{i5} + b_1 \\ w'_{12}x_{i1} + w'_{22}x_{i2} + w'_{32}x_{i3} + w'_{42}x_{i4} + w'_{52}x_{i5} + b_2 \\ w'_{13}x_{i1} + w'_{23}x_{i2} + w'_{33}x_{i3} + w'_{43}x_{i4} + w'_{53}x_{i5} + b_3 \\ w'_{14}x_{i1} + w'_{24}x_{i2} + w'_{34}x_{i3} + w'_{44}x_{i4} + w'_{54}x_{i5} + b_4 \end{bmatrix}\right)$$

$\downarrow$

$$\begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \\ O_{14} \end{bmatrix}$$

outputs

Layer 2 :

$$\underset{4\times3}{\underbrace{\begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 \\ w_{41}^2 & w_{42}^2 & w_{43}^2 \end{bmatrix}}}_{W} \underset{4\times1}{\underbrace{\begin{bmatrix} o_{11} \\ o_{12} \\ o_{13} \\ o_{14} \end{bmatrix}}}_{X} + \underset{3\times1}{\underbrace{\begin{bmatrix} b_{21} \\ b_{22} \\ b_{23} \end{bmatrix}}}_{b}$$

Transposing $W \longrightarrow W^T$

$$\underset{3\times4}{\begin{bmatrix} w_{11}^2 & w_{21}^2 & w_{31}^2 & w_{41}^2 \\ w_{12}^2 & w_{22}^2 & w_{32}^2 & w_{42}^2 \\ w_{13}^2 & w_{23}^2 & w_{33}^2 & w_{43}^2 \end{bmatrix}}$$

Multiplying with X (outputs) and adding bias and then σ

$$\sigma\left(\begin{bmatrix} w_{11}^2 o_{11} + w_{21}^2 o_{12} + w_{31}^2 o_{13} + w_{41}^2 o_{14} + b_{21} \\ w_{12}^2 o_{11} + w_{22}^2 o_{12} + w_{32}^2 o_{13} + w_{42}^2 o_{14} + b_{22} \\ w_{13}^2 o_{11} + w_{23}^2 o_{12} + w_{33}^2 o_{13} + w_{43}^2 o_{14} + b_{23} \end{bmatrix}\right)$$

$$\downarrow$$

$$\begin{bmatrix} o_{21} \\ o_{22} \\ o_{23} \end{bmatrix}$$

Layer 3 :

$$\begin{bmatrix} w^3_{11} & w^3_{12} \\ w^3_{21} & w^3_{22} \\ w^3_{31} & w^3_{32} \end{bmatrix} \begin{bmatrix} o_{21} \\ o_{22} \\ o_{23} \end{bmatrix} + \begin{bmatrix} b_{31} \\ b_{32} \end{bmatrix}$$

W^T

$$\begin{bmatrix} w^3_{11} & w^3_{21} & w^3_{31} \\ w^3_{12} & w^3_{22} & w^3_{32} \end{bmatrix} \begin{bmatrix} o_{21} \\ o_{22} \\ o_{23} \end{bmatrix} + \begin{bmatrix} b_{31} \\ b_{32} \end{bmatrix}$$

$$\sigma(w^T x + b)$$

$$\sigma\left(\begin{bmatrix} w^3_{11} o_{21} + w^3_{21} o_{22} + w^3_{31} o_{23} + b_{31} \\ w^3_{12} o_{21} + w^3_{22} o_{22} + w^3_{32} o_{23} + b_{32} \end{bmatrix}\right)$$

$$\downarrow$$

$$\begin{bmatrix} o_{31} \\ o_{32} \end{bmatrix}$$

Layer 4 : $\begin{bmatrix} w^4_{11} \\ w^4_{21} \end{bmatrix} \begin{bmatrix} o_{31} \\ o_{32} \end{bmatrix} + \begin{bmatrix} b_{41} \end{bmatrix}$

$\sigma(w^T x + b)$

$\sigma\left(\begin{bmatrix} w^4_{11} o_{31} + w^4_{21} o_{32} + b_{41} \end{bmatrix}\right)$

$\downarrow$

$[o_{41}]$

final output.

The outputs of current layer are given as inputs to the next layer's perceptrons.

This process keeps on following until we end up with final output layer output.

This is how training is done.

Back Propagation

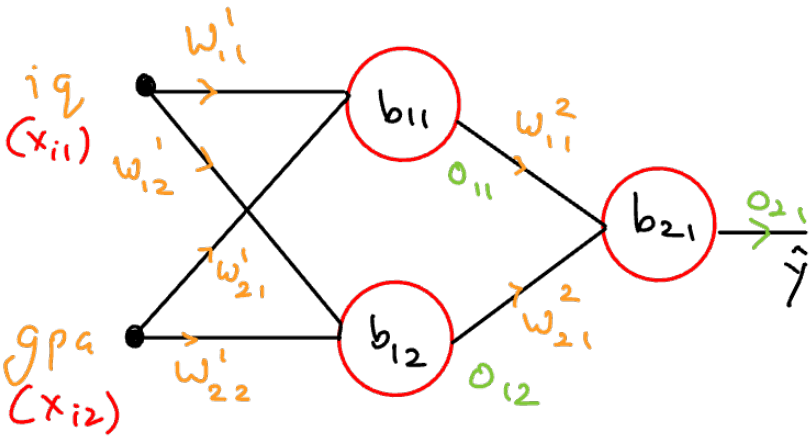
Back propagation (Backward propagation of Errors) is the core algorithm for training neural networks. It works by performing a forward pass (forward propagation) to compute predictions calculating the loss between predicted and actual outputs, and then executing a backward pass to compute gradients using the chain rule of calculus. These gradients are used with optimization methods like gradient descent to update weights and biases, reducing prediction error over multiple epochs.

Key steps in the process:

1. **Forward Pass** – Inputs are multiplied by weights, biases are added, and activation functions (e.g., Sigmoid, ReLU, Softmax) produce outputs layer by layer.
2. **Loss Calculation** – A loss function (e.g., Mean Squared Error, Cross-Entropy) measures prediction error.
3. **Backward Pass** – Gradients of the loss with respect to each parameter are computed from output to input layers.
4. **Weight Update** – Parameters are adjusted in the opposite direction of the gradient, scaled by the learning rate.

Step by Step implementation

iq	gpa	salary (100k)
80	3.5	1.8
60	4	2.5
70	3.2	2



1. Initialize the values for w, b

There are different initilization techniques:

- Random values for w, b
- $w \rightarrow 1$, $b \rightarrow 0$
- And many more

We will take $w \rightarrow 1$ and $b \rightarrow 0$ for simplicity.

Here, we have used "linear" as the activation function.
Cuz this is a regression problem. ✓
(Remember, linear is not mse, because this is activation function not loss function

2. Select a point (from row of dataset) (student). Now we use the variables and predict the salary of 1st student. This is forward propagation. (ex, we got $\hat{y} = 5$)

3. Choose a loss function (MSE)

$$L = (y - \hat{y})^2$$

L = (ground truth − predicted value)^2

$= (1.8 - 5)^2$

$= \underbrace{10.2}_{\text{error}}$ (This is very wrong).

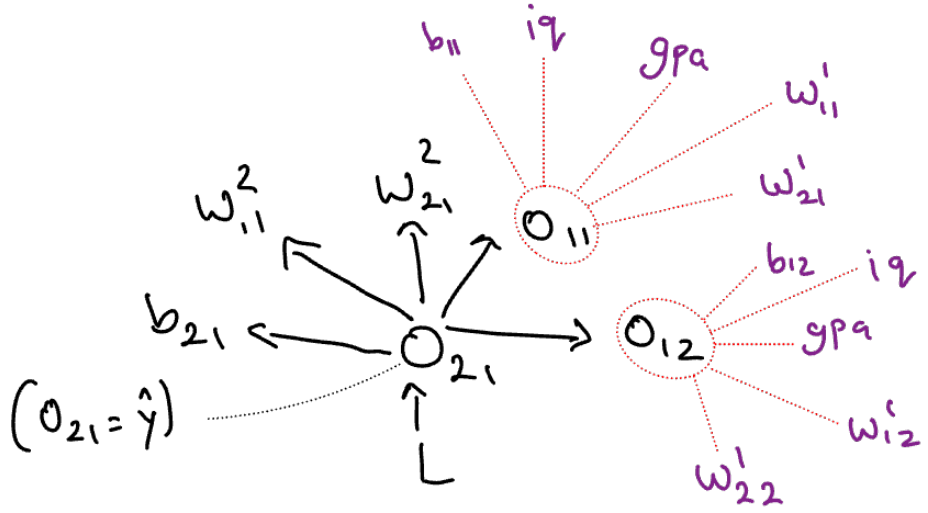
So, now we try to change the value of $\hat{y}$ (we cannot change y because it is the ground truth)

To change the value of $\hat{y}$, we have to change all the values that resulted in the current value. Those are:

$$\underset{(\hat{y})}{O_{21}} = w_{11}^2 O_{11} + w_{21}^2 O_{12} + b_{21}$$

(note that there's no activation function at the end (final output) that's why we are not applying any!)

Now again, O_{11} and O_{12}'s output depends on other weights too.

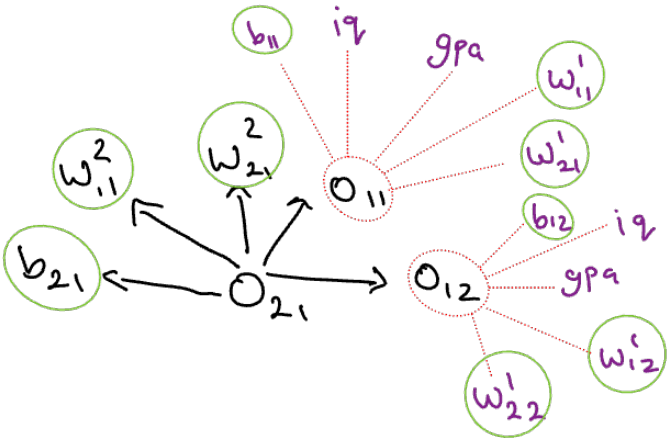


4) Final step : Updating Weights and Bias using Gradient Descent

↓ formulas

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

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



Here, we have a total of 9 trainable parameters that we can change. (ofc! we cant change the actual data like gpa, iq.)

example formula for one of the trainable parameter

$$b_{21_{new}} = b_{21_{old}} - \eta \frac{\partial L}{\partial b_{21_{old}}}$$

We have to apply this same formula for each of those 9 trainable parameters.

We already have value of b_{21} (we took 0) and if its weight w_1 we take $w = 1$ initially.

We also have the value of η (learning rate) (0.1)

But we don't have the value for $\frac{\partial L}{\partial b_{21_{old}}}$.

So we have to calculate the values of all parameters w.r.t. Loss.

$$\frac{\partial L}{\partial w_{11}^2}, \frac{\partial L}{\partial w_{21}^2}, \frac{\partial L}{\partial b_{21}} \quad \Big| \quad \frac{\partial L}{\partial w_{11}^1}, \frac{\partial L}{\partial w_{21}^1}, \frac{\partial L}{\partial b_{11}}$$

$$\frac{\partial L}{\partial w_{12}^1}, \frac{\partial L}{\partial w_{22}^1}, \frac{\partial L}{\partial b_{12}}$$

Lets calculate the first set's 1st parameter:

$\frac{\partial L}{\partial w_{11}^2}$: There might be a question that why are we calculating Loss function wrt w_{11}^2 when it's not even directly connected?

The reason is that we want to see what will be the change in value of O_{21} if we change w_{11}^2 and then based on that, what will be getting the change of L (loss) when O_{21} gets changed.

We will be using chain rule for differentiation to break down

$$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_{11}^2}$$

($\hat{y}$ is O_{21})

① $$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_{11}^2}$$

We need to find value of $\frac{\partial L}{\partial \hat{y}}$ and $\frac{\partial \hat{y}}{\partial w_{11}^2}$

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial (y-\hat{y})^2}{\partial \hat{y}} \rightarrow \boxed{-2(y-\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial w_{11}^2} = \frac{\partial}{\partial w_{11}^2}(o_{11}w_{11}^2 + o_{21}w_{21}^2 + b_{21}) \rightarrow \boxed{o_{11}}$$

$$\therefore \boxed{\frac{\partial L}{\partial w_{11}^2} = -2(y-\hat{y})\,o_{11}}$$

② $$\frac{\partial L}{\partial w_{21}^2} = \frac{\partial L}{\partial \hat{y}} \times \underbrace{\frac{\partial \hat{y}}{\partial w_{21}^2}}$$

$$\frac{\partial \hat{y}}{\partial w_{21}^2} = \frac{\partial (o_{11}w_{11}^2 + o_{12}w_{21}^2 + b_{21})}{\partial w_{21}^2}$$

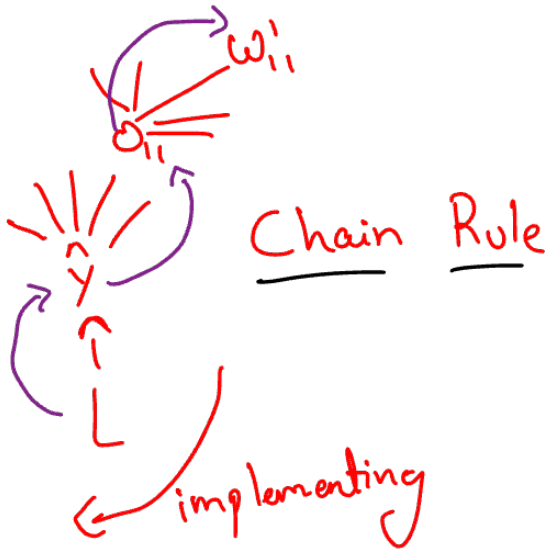
$$= o_{12}$$

$$\therefore \boxed{\frac{\partial L}{\partial w_{21}^2} = -2(y-\hat{y})\,o_{12}}$$

(3) $$\frac{\partial L}{\partial b_{21}} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial b_{21}}$$

$$\frac{\partial \hat{y}}{\partial b_{21}} = 1$$

$$\therefore \boxed{\frac{\partial L}{\partial b_{21}} = -2(y-\hat{y})}$$



Next Set ⟶

(4) $$\frac{\partial L}{\partial w^1_{11}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{11}} \frac{\partial o_{11}}{\partial w^1_{11}}$$

already have it from above

$$\frac{\partial \hat{y}}{\partial o_{11}} = \frac{\partial (w^2_{11} o_{11} + w^2_{21} o_{21} + b_{21})}{\partial o_{11}} = w^2_{11}$$

$$\frac{\partial o_{11}}{\partial w^1_{11}} = \frac{\partial}{\partial w^1_{11}} (w^1_{11} x_{i1} + w^1_{21} x_{i2} + b_{11})$$

$$= x_{i1}$$

$$\therefore \boxed{\frac{\partial L}{\partial w^1_{11}} = -2(y-\hat{y}) w^2_{11} x_{i1}}$$

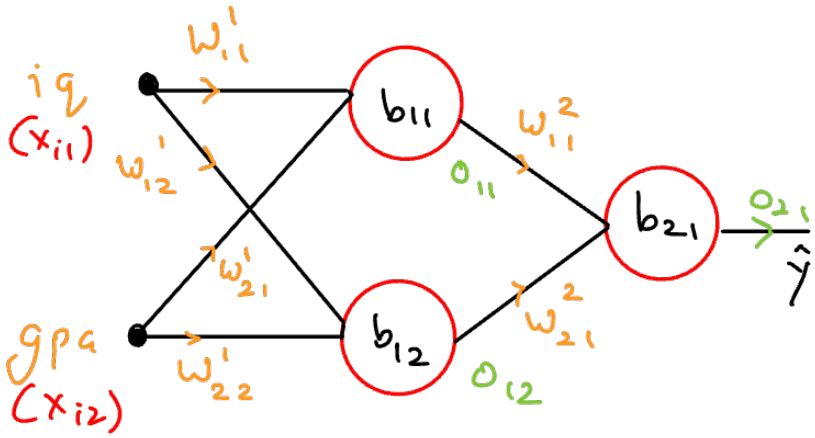
(5) $$\frac{\partial L}{\partial w^1_{21}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{11}} \frac{\partial o_{11}}{\partial w^1_{21}}$$

$$\frac{\partial o_{11}}{\partial w^1_{21}} = \frac{\partial}{\partial w^1_{21}} (w^1_{21} x_{i2} + w^1_{11} x_{i1} + b_{11})$$

$$= x_{i2}$$

$\therefore$

$$\frac{\partial L}{\partial w^1_{21}} = -2(y-\hat{y})\, w^2_{11}\, x_{i2}$$



⑥ $$\frac{\partial L}{\partial b_{11}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{11}} \frac{\partial o_{11}}{\partial b_{11}}$$

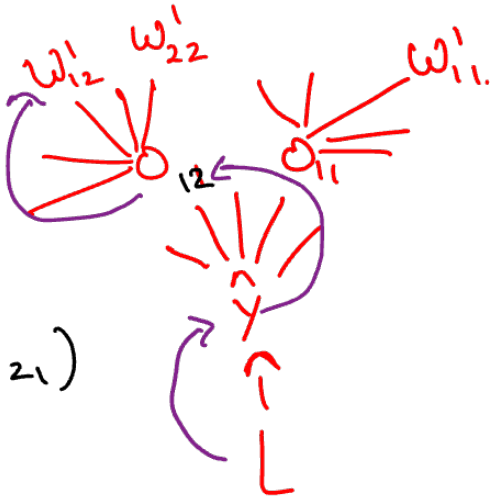
$$\frac{\partial o_{11}}{\partial b_{11}} = \frac{\partial (w^1_{11} x_{i1} + w^1_{21} x_{i2} + b_{11})}{\partial b_{11}}$$

$$= 1$$

$$\therefore \frac{\partial L}{\partial b_{11}} = -2(y-\hat{y})\, w^2_{11}$$

⑦ $$\frac{\partial L}{\partial w^1_{12}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{12}} \frac{\partial o_{12}}{\partial w^1_{12}}$$

have it



$$\frac{\partial \hat{y}}{\partial o_{21}} = \frac{\partial (w^2_{11} o_{11} + w^2_{21} o_{12} + b_{21})}{\partial o_{12}}$$

$$= w^2_{21}$$

$$\frac{\partial o_{12}}{\partial w^1_{12}} = \frac{\partial}{\partial w^1_{12}} (w^1_{12} x_{i1} + w^1_{22} x_{i2} + b_{12})$$

$$= x_{i1}$$

$$\therefore \frac{\partial L}{\partial w^1_{12}} = -2(y-\hat{y})\, w^2_{21}\, x_{i1}$$

(8) $$\frac{\partial L}{\partial w^1_{22}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_{12}} \frac{\partial O_{12}}{\partial w^1_{22}}$$

we have it

$$\frac{\partial O_{12}}{\partial w^1_{22}} = \frac{\partial}{\partial w^1_{22}} (w^1_{12} x_{i1} + w^1_{22} x_{i2} + b_{12})$$

$$= x_{i2}$$

$$\therefore \frac{\partial L}{\partial w^1_{22}} = -2(y-\hat{y})\, w^2_{21}\, x_{i2}$$

(9) $$\frac{\partial L}{\partial b_{12}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_{12}} \frac{\partial O_{12}}{\partial b_{12}}$$

we have it

$$\frac{\partial O_{12}}{\partial b_{12}} = \frac{\partial}{\partial b_{12}} (w^1_{12} x_{i1} + w^1_{22} x_{i2} + b_{12})$$

$$= 1$$

$$\therefore \frac{\partial L}{\partial b_{12}} = -2(y-\hat{y})\, w^2_{21}$$

Going over the steps once again

Of Weight and bias value initialize

for i in epochs: (usually 100-1000)

for i in range(size):

i) 1 student → forward propagation → predict

ii) Loss calculate (mse)

iii) Adjust all weight and biases

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

iv) Calculate the average loss for epochs

Now, lets apply this algorithm to a real world dataset from scratch.

We already have the gradient descent formulas:

$$\frac{\partial L}{\partial w^2_{11}} = -2(y-\hat{y})\, o_{11}$$

$$\frac{\partial L}{\partial w^2_{21}} = -2(y-\hat{y})\, o_{12}$$

$$\frac{\partial L}{\partial b_{21}} = -2(y-\hat{y})$$

$$\frac{\partial L}{\partial w^1_{11}} = -2(y-\hat{y})\, w^2_{11}\, x_{i1}$$

$$\frac{\partial L}{\partial w^1_{21}} = -2(y-\hat{y})\, w^2_{11}\, x_{i2}$$

$$\frac{\partial L}{\partial b_{11}} = -2(y-\hat{y})\, w_{11}{}^2$$

$$\frac{\partial L}{\partial w^1_{12}} = -2(y-\hat{y})\, w^2_{21}\, x_{i1}$$

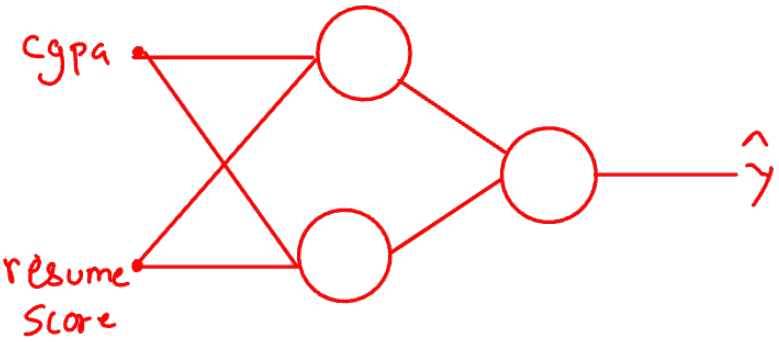
$$\frac{\partial L}{\partial w^1_{22}} = -2(y-\hat{y})\, w^2_{21}\, x_{i2}$$

$$\frac{\partial L}{\partial b_{12}} = -2(y-\hat{y})\, w^2_{21}$$

Regression Problem

Applying the algorithm from scratch to custom dataset.

cgpa	resume score	salary (lpa)
8	8	4
7	9	5
6	10	6
5	12	7



This dataset is applied and coded in one of the files in my repo.
Same goes for classification.

fun fact

We take derivation as $\frac{dy}{dx}$ when the derivative has one variable that changes w.r.t to other but when we have more than one, we take $\frac{\partial x}{\partial y}$ and this is simply gradient.

Back propagation Intuition (Last time)

We know that in back propagation the main formula that updates the values of weights and biases is

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

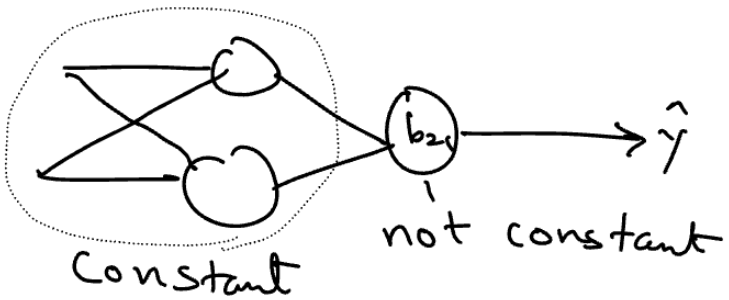
But why does this formula work?

⟶ for now, assume $\eta = 1$

$$\downarrow$$

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

And for now, just assume that all weights and biases are constant except the bias of $\hat{y}$ (just for explanation purpose)



$$\therefore L(b_{2,1})$$

$b_{21} = b_{21} \ominus \boxed{\frac{\partial L}{\partial b_{21}}}$ → Why are we substracting this value and why is it substraction?

(? over the ⊖)

So, the -ve sign is there because:

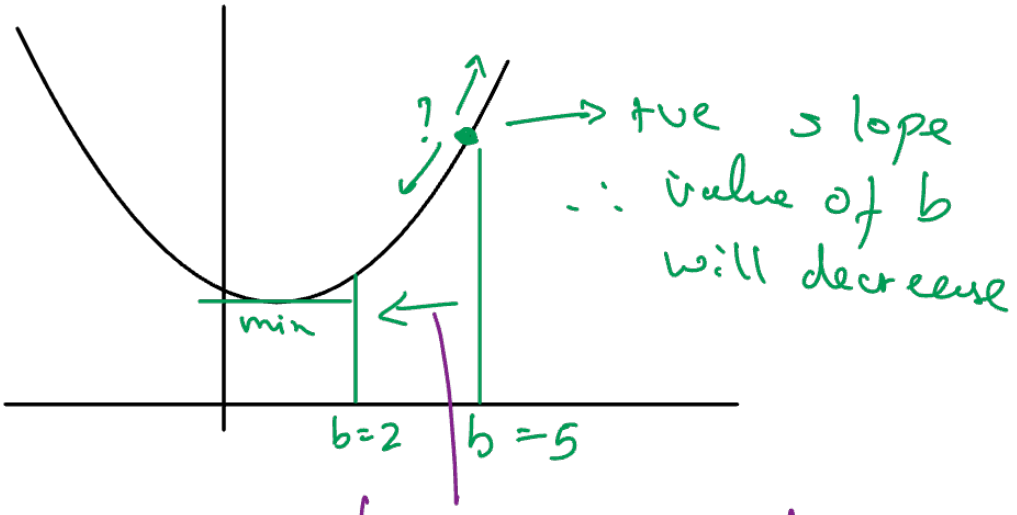
1) We know, if we change value of b_{21} then value of L will also change. $\frac{\partial L}{\partial b_{21}}$

If value of $\frac{\partial L}{\partial b_{21}} = +ve$, then we increase the value of b_{21}, the value of L will also increase, but we know the we have to decrease the value of L (as much possible) but if $\frac{\partial L}{\partial b_{21}}$ is -ve, then when we increase b_{21}, loss decreases, which is what we want!

So, the -ve sign in $b_{21} - \frac{\partial L}{\partial b_{21}}$ is very smartly placed there that it handles both the cases.

Therefore, simply when the loss is at x, it knows either to move in -ve or +ve direction that ends up with minima of GD.

Our algorithm will simply calculate the derivative (slope).



because we decreased value of b due to +ve slope.

Same goes for the other way.

That's why, gradient descent always goes towards the minima.

And, how much change should the value take? the same as the current value

therefore, we decrease the value from its own:

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_0}$$

What is Convergence?

Simply when we reach our minima, it's called convergence.

So, when W_n is roughly close to W_0 it's convergence.

$$W_{new} \approx W_{old}$$

The standard way is generally to run external loop which we refer to epochs.

We usually dont hard code it that when it reaches convergence it should stop because it's not always that W_{new} is exactly same as W_{old}.

Memoization

Memoization is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again.

When a neural network has more than two hidden layer, the derivation that we calculate at every step becomes more and more complex, which significantly increases the time. To solve this, Backpropagation uses memoization to store the results of differentiations that were done during the process.

In simple words, Back propagation is simply Chain role (derivatives) + Memoization.

Gradient Descent

We know that GD is an iterative optimization algorithm used to minimize a cost function by adjusting model's parameters in the steepest descent of the function's gradient.

There are three variants of GD, which differ in how much data we use to compute the gradient of the objective function. Depending on the amount of data, we make trade-off between the accuracy of the parameter update and the time it takes to perform an update.

Batch GD (vanilla GD) ↓ default: updates parameters using the gradient computed over the entire training dataset each step.

Stochastic GD: updates parameters using the gradient from a single randomly chosen example each step

Mini-batch GD: updates parameters using the gradient from a small random subset of examples each step, balancing speed and stability.

The default backpropagation uses stochastic GD where we update weights n times (rows) x epochs.

Batch GD is faster than stochastic because in batch GD, we update the weight epoch times while in stochastic GD, we run for n x epochs.

Stochastic GD is more faster to reach convergence since it updates the weights way more times than it Batch GD.

The Loss function's reduseness is very unstable in stochastic GD while very smooth /stable in batch GD. because we pick one random point in stochastic GD multiple times. The pro for this is that stochastic may sometims move out of local minima to reach global but the con is that it does not converge at the exact solution. (it is approximate)

Batch GD is very faster because it uses vectorization (dot product) rather than running loops.

The only problem is that if the dataset is very big, it wont work because we have to load the entire dataset into the ram.

Mini-Batch GD is the middle ground.
(generally MBGD is used in keras)

Usually, in keras when we use the batch size parameter in keras, we generally use the 2^n numbers as parameters.

2, 4, 8, 32, 64, 128, 256.

This is because in giving two multiple input research has proven that it is very effective way to handle the RAM.

Vanishing Gradient Problem

The **vanishing gradient problem** is a significant challenge in training deep neural networks. It occurs when the gradients of the loss function with respect to the network's weights become exceedingly small during backpropagation. This issue is particularly prevalent in deep networks with many layers, where the gradients diminish as they propagate backward through the network.

When using activation function, specifically tanh or sigmoid, vanishing gradient problem can occur where the initial weights become too much small (after repetitive Back propagation) that the model never reaches convergence, in other words, model does not get trained.

There are two ways to identify if this problem is occuring or not.

1. Observing the Loss function's value.
   If it does not change for multiple epochs.
2. plotting a graph for weights.
   epoch VS value graph.

Solution to handle Vanishing GP

1. Reduce model complexity
   - Decrease the number of hidden layers.
2. Using different activation functions
   - RELU is the most commonly used.
3. Proper Weight Initialization.
4. Batch Norm → layer
5. Residual Network

} Covered in next topics!