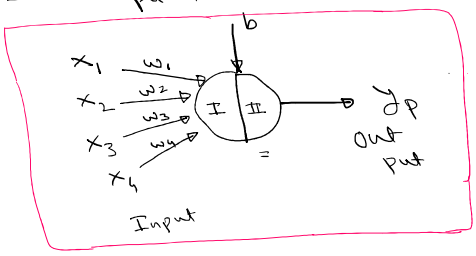
14 December 2023 21:54

Deep learning Day-2

* Neuron → Perceptron

Weight, bias → External parameter

→ it is a two stage algorithm



Stage I → Summation func^n (z)

$$z = \sum_{i=1} w_i x_i + b$$

Stage II → Activation func^n (Sigmoid)

$$\sigma(z) = \frac{1}{1+e^{-z}} = \hat{y}$$

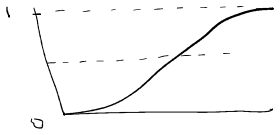
This whole process called as perceptron and perceptron it self act as a Neuron in Neural Network.

* What is Sigmoid ?

→ it is used to add Nonlinearity.

$y = mx + c$

$\sigma(y) = \frac{1}{1+e^{-y}}$



→ Sigmoid convert continous values into probability distribution.

→ Range → 0 to 1

→ it always give probabilistic value.

→ it always give probability value of any one class.

$P(x \in y=1) = P(1) = 0.7$

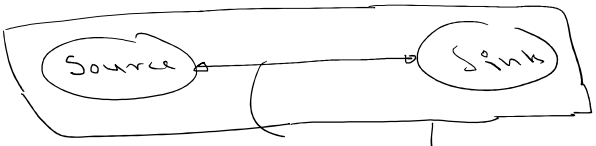
$P(x \in y=0) = P(0) = 1 - P(1)$

$= 0.3$

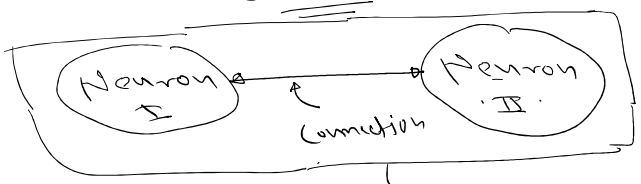
1 → 0.7 ; 0 → 0.3

threshold → 0.5

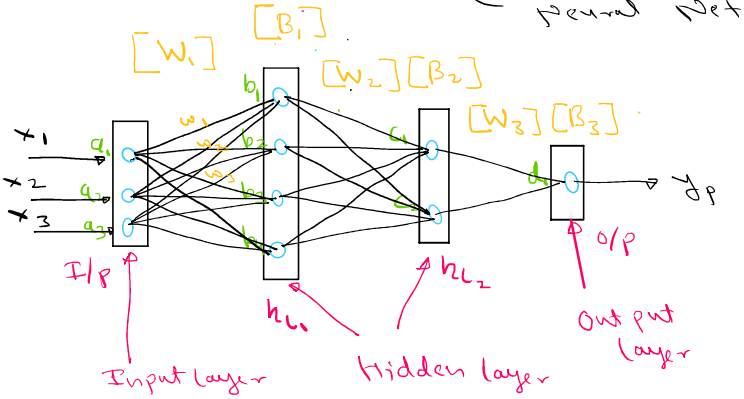
* Structure of Neural Network :-

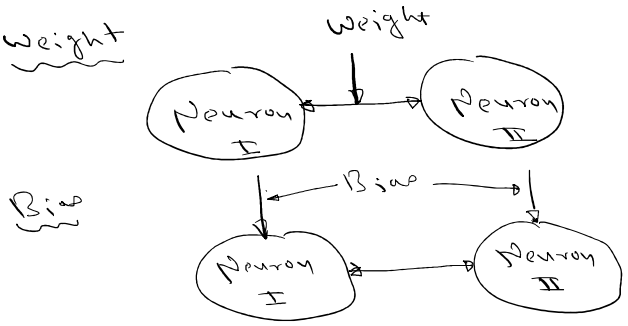


Connection → Network



Neural Network





$[W_1] = [w_1, w_2, w_3, w_4 \dots w_{12}]$

$[B_1] = [b_1, b_2, b_3, b_4]$

* Input layer → It hold the input and feed them to hidden layer.

* Hidden layer → Done all the processing/learning

* Out put layer → It shows the out put

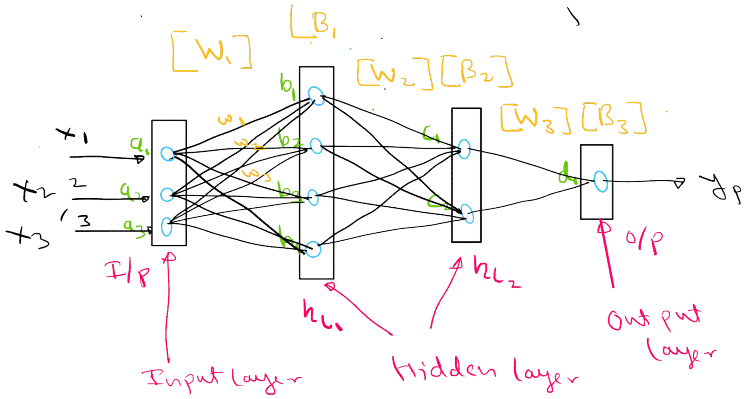
* Input layer → Should be one

* Hidden layer → One or more than one

* Hidden layer → one or more than one
  - one → Shallow Neural Network
  - more than one → Deep Neural Network

* Output layer → Should be one

* How data flow in Neural Network



* No of Neuron in Input layer = No of input features

* No of Neuron in output layer = decided by the activation fun and No of exit.

Case I → let say we are dealing with binary classification and in our out put layer we have sigmoid activation fun

No of Neuron in out put layer = One.

* Case II → let's say we are dealing with multi class classification and in our out put layer we have Soft max as a activation function. and No. of exit is 5.

No of Neuron in out put layer = 5

... We are dealing

out put layer

* Case III → Let's say we are dealing with regression problem and in our out put layer we are relu as a activation.

No of Neuron in out put = One layer

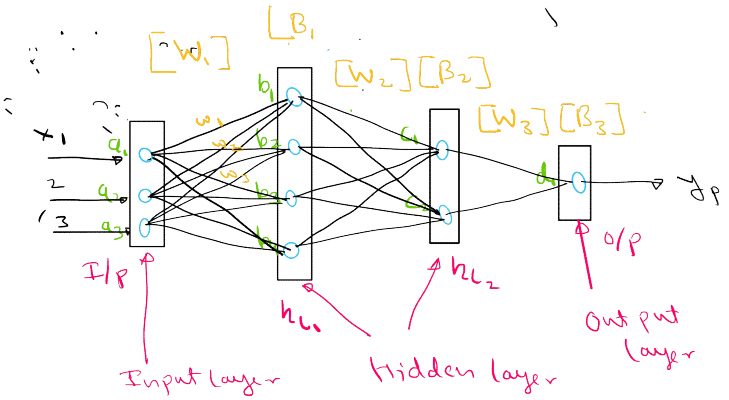
* No of Neuron in Hidden layer = Not fixed

General thumb rule (Manually build → ANN)

In Neuron
Start → input layer

Hidden layer → is between 3 to 15

3 Neuron end → out put layer



Shape of Input layer = (1, 3) (4, 3) → Stage I

Shape of Hidden layer 1 = (1, 4) (2, 4) → Stage II

Shape of Hidden layer 2 = (1, 2) (1, 2) → Stage III

Shape of out put = (1, 1)

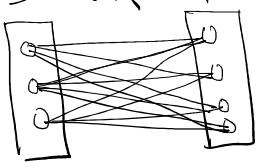
Shape of ... layer

→ How to control Shape of layers in Neural Network

→ Let's Consider W_1 (weight matrix)

We start from 3 Neuron and We end with 4 Neuron

3, 4



W_1 → 12 weights.

Shape of W_1 = (4, 3)

$$Z = \sum_{i=1} W_i x_i + b$$

Start → (1, 3)

(4, 3)

end → (1, 4)

(1, 3) · (4, 3) = (1, 4)

Transpose

(1, 3) (3, 4) = (1, 4)

(1, 4) = (1, 4)

$$Z = \sum_{i=1} W_i^T x_i + b$$

* Importance of Bias

Bias → one dimensional array that consist values of Bias.

$$Z = \sum_{i=1} W_i x_i + b$$

	x_1	x_2	x_3	y
(1)	100	0	500	cat

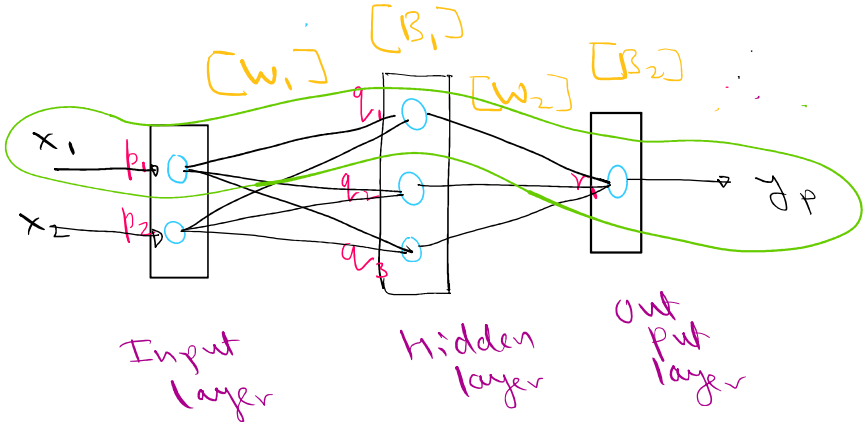
(2) 200 3 400 Dog

$z = w_1 x_1 + b_1 + w_2 x_2 + b_2 + w_3 x_3 + b_3$

$= 100 w_1 + b_1 + 0 + b_2 + 3 * 0 w_3 + b_3$

* How Neural Network learn

Chain Rule



(1) $x_1 \rightarrow p_1 \rightarrow q_1 \rightarrow r_1 \rightarrow \hat{y}_{p_1}$

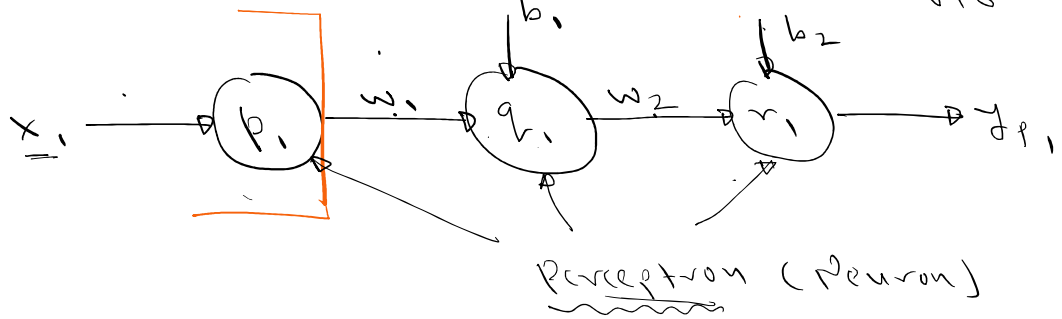
(2) $x_1 \rightarrow p_1 \rightarrow q_2 \rightarrow r_1 \rightarrow \hat{y}_{p_2}$

(3) $x_1 \rightarrow p_1 \rightarrow q_3 \rightarrow r_1 \rightarrow \hat{y}_{p_3}$

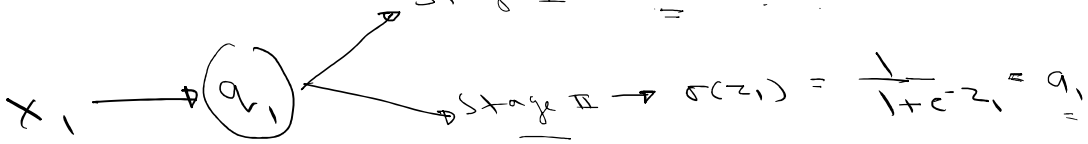
(4) $x_2 \rightarrow p_2 \rightarrow q_1 \rightarrow r_1 \rightarrow \hat{y}_{p_4}$

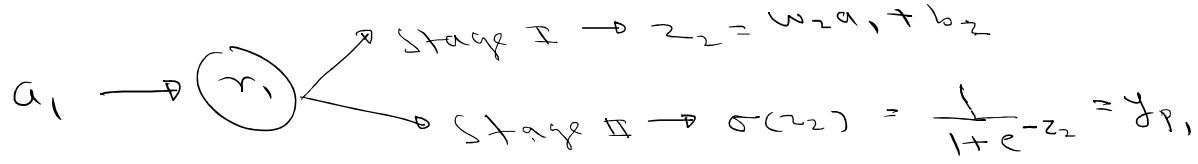
(5) $x_2 \rightarrow p_2 \rightarrow q_2 \rightarrow r_1 \rightarrow \hat{y}_{p_5}$

(6) $x_2 \rightarrow p_2 \rightarrow q_3 \rightarrow r_1 \rightarrow \hat{y}_{p_6}$



Perceptron (Neuron)





$y_a : y_{p_1}$

$\log(L_1) = y_a - y_{p_1}$

Linear Regression

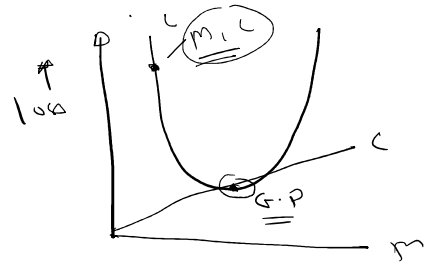
$y = mx + c$

Best value of m and c

↳ Gradient Decent algorithm

$m_{new} = m_{old} - \alpha \frac{\partial L}{\partial m}$

$c_{new} = c_{old} - \alpha \frac{\partial L}{\partial c}$



ANN

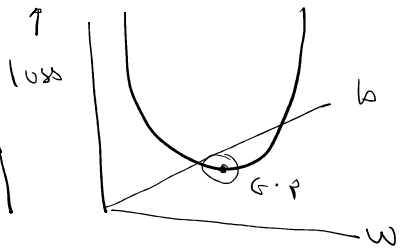
$z = \sum_{i=1}^{n} (w_i x_i + b)$

Best value of w and b

↳ Gradient Decent algorithm.

$w_{new} = w_{old} - \alpha \frac{\partial L}{\partial w}$

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



gradient

↳ Change in loss with respective weight.

$w_1' = w_1 - \alpha \frac{\partial L_1}{\partial w}$

$b_1' = b_1 - \alpha \frac{\partial L_1}{\partial b}$

$w_2' = w_2 - \alpha \frac{\partial L_1}{\partial w}$

$b_2' = b_2 - \alpha \frac{\partial L_1}{\partial b}$

$\min [L_1, L_2, L_3, L_4, L_5, L_6]$ = G.P

minimum loss

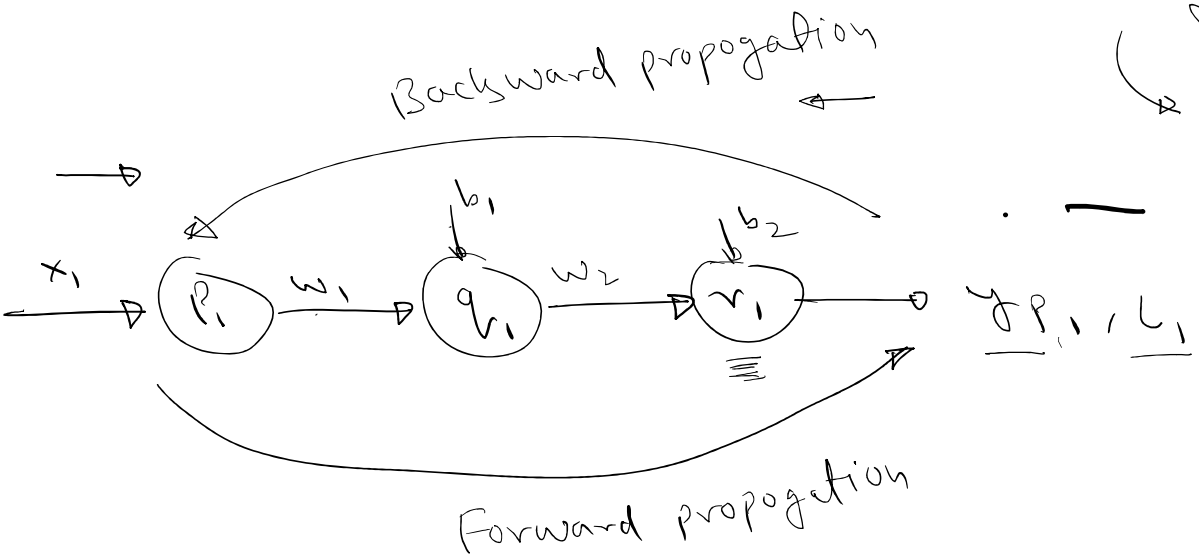
Minimum loss

Best value of W and b

Best Summation func

activation func

final $\hat{y}_p$



Forward Propogation → $\hat{y}_p$, loss

Backward propogation → to update weight and bias.

* Epoch / Iteration

let's say we have 1500 data points

→ F.P + B.P ⟹ 1 epoch

Resource → Consumption

500 → F.P$_1$ + B.P$_1$ → iteration 1

500 → F.P$_2$ + B.P$_2$ → iteration 2

500 → F.P$_3$ + B.P$_3$ → iteration 3

} 1 epoch

Parallel and distribution learning

* : Type of function use in Neural Network

(1) Optimization fun

→ it is used to minimize or optimize the error

→ it is used to find best value for weight and bias

→ it help to achieve convergence or global minima point.

(2) Activation fun

→ Governing the way a neuron behave.

→ it controls the out put of any neuron.

Neuron.

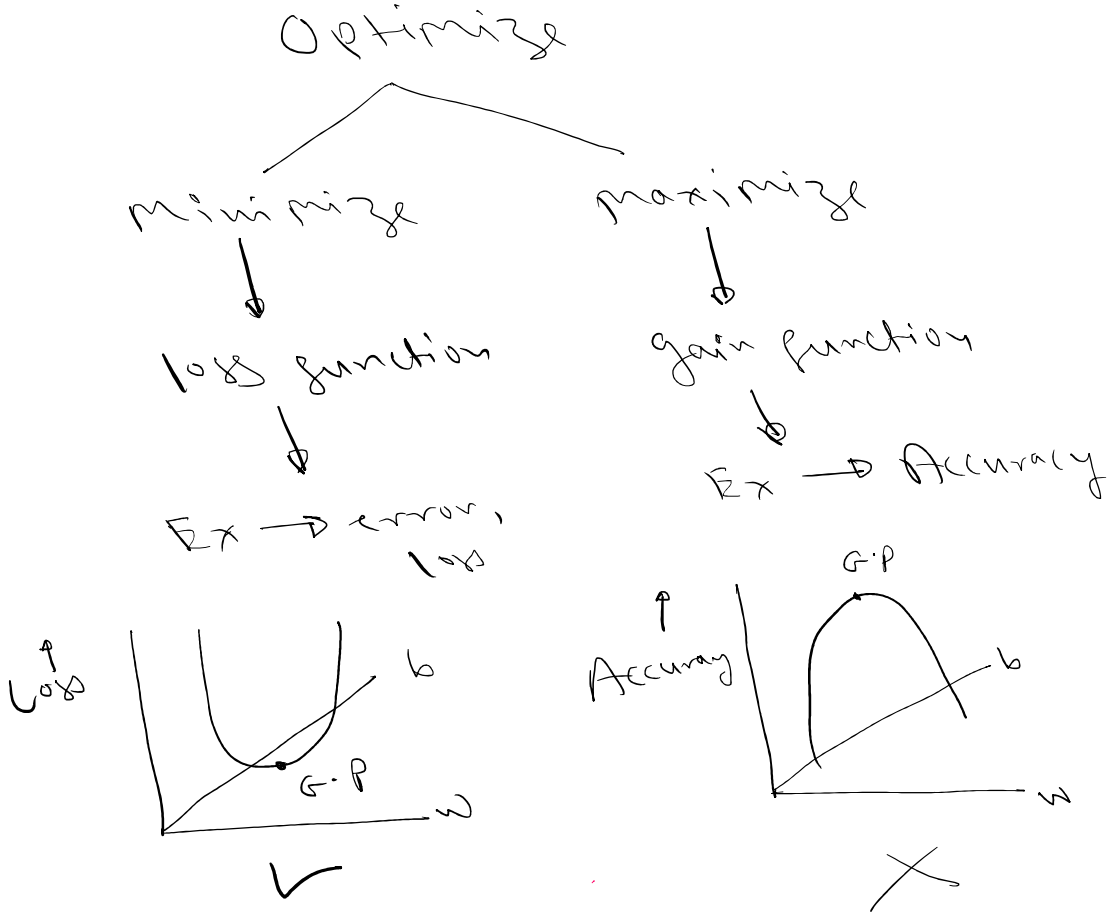
(3) Loss function :-

error ↑ error ↓

→ It give a number which represent the goodness model.

→ It helps to update the neural Network parameters.

* Optimization function



* Type of optimization fun

(1) (2) (3)

weight +

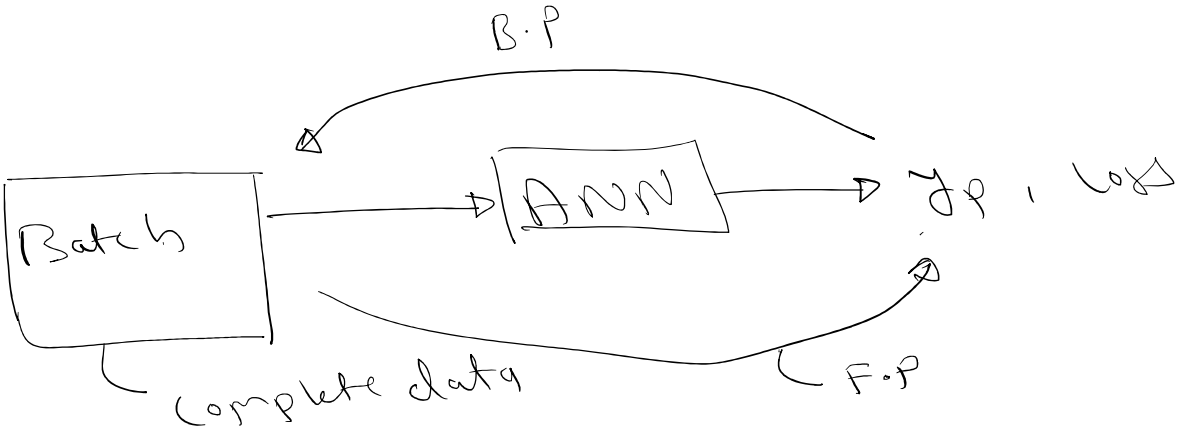
(1) Weight + Bias (Dynamically update)

(2) learning rate

(3) Weight + bias + learning

(1) Gradient Decent algorithm :-

→ Batch gradient decent algorithm (BGDA)



→ in BGDA epoch get perform on whole data, no matter how big our data set.

→ let's say we have 1000 data points and we apply BGDA, and we achieve convergence in 1 min and memory consumption is 1GB.

Consumption is 1 GB.

1000 — 1

1,00,000 — 100

→ 1 Lakh data points

time

1

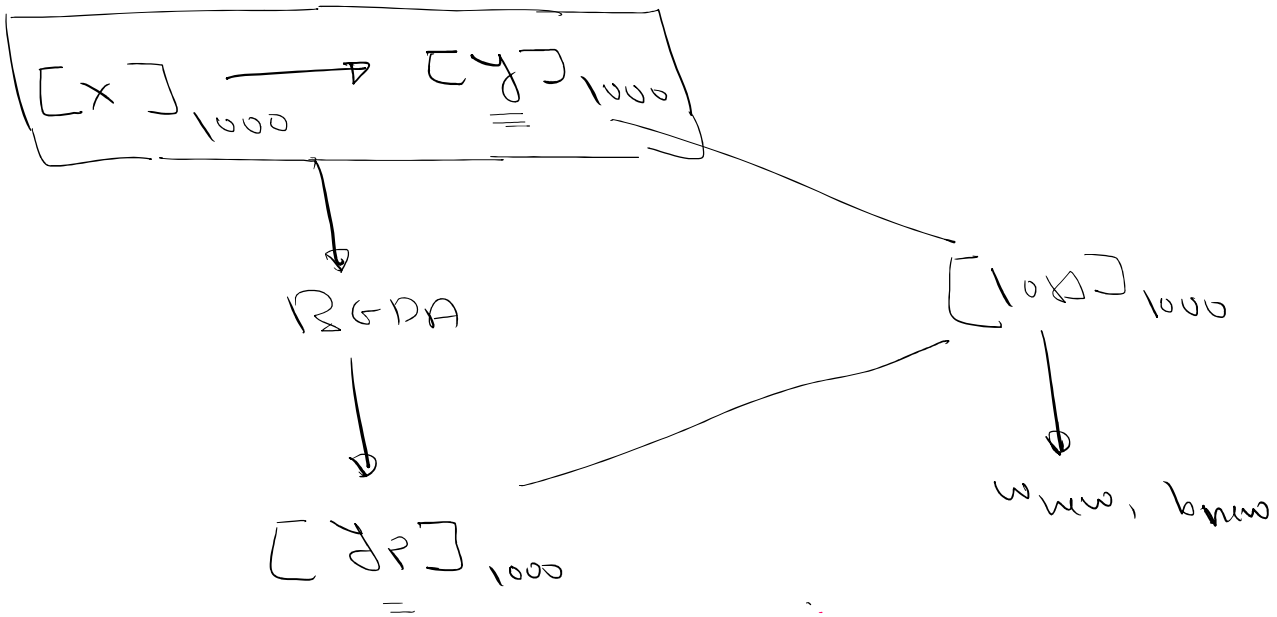
100 Min

100 G.B

* Draw back of BGDA

→ We cannot use BGDA in case of Big data because epoch get perform On whole data.

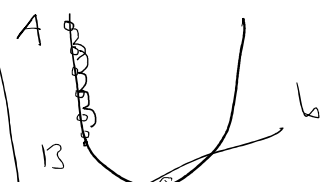
* How BGDA work in Back end :-



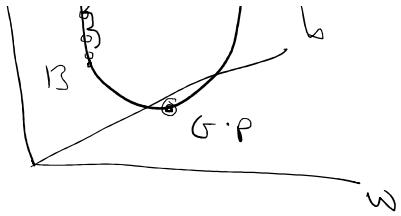
$w_{new} = w_{old} - \alpha \frac{\partial L}{\partial w}$

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

(10)



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



* Advantage of BGDA

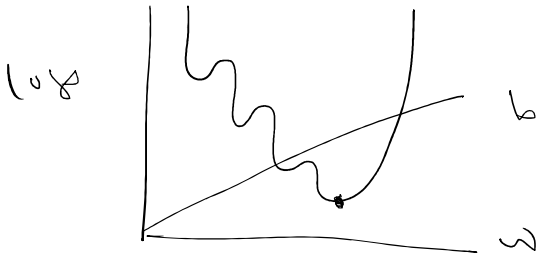
→ We get smooth learning curve in case of BGDA because we perform epoch on whole data.

(1) [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

(2) [2, 10, 12] [4, 6, 16]

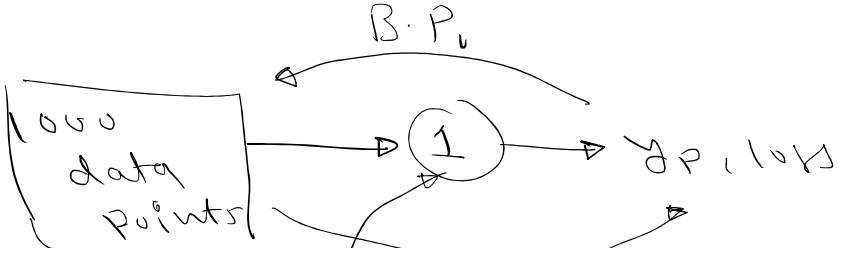
I

learning step, Learning Rate learning curve



(2) Stochastic gradient decent algorithm (SGDA)

Random or Randomness



data points

F. P

Here, we consider single data point at a time randomly

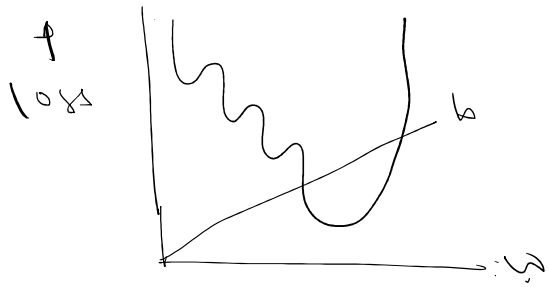
→ In SGDA → 1 epoch

↓

N no of iteration

where N → no of data point

→ In SGDA we learn from single data point at a time hence learning curve become very noisy.



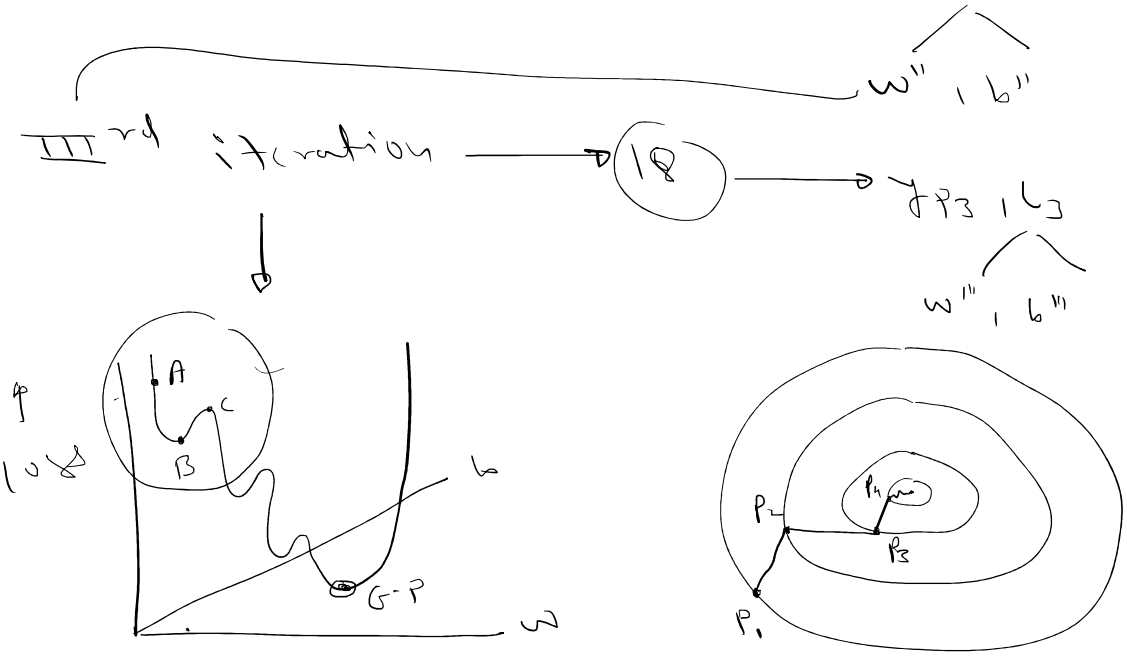
[2, 4, (6), 8, 10, (12), 14, 16, (18), 20]

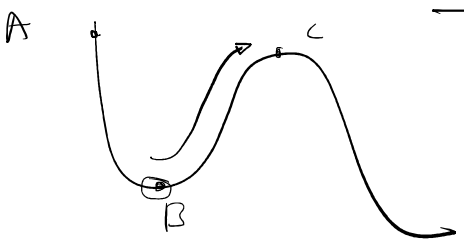
w, b

1st iteration → (6) → y_{p_1}, L_1

w_1', b_1'

IInd iteration → (16) → y_{p_2}, L_2





→ In SGDA since we are learning from single data point at a time we dont have sufficient amount of learning to overcome local minima point as a result after some time SGDA consider local minima as a global minima and that is wrong.

* Advantage of SGDA

→ Compare to BGDA, SGDA required less resources to achive convergence

* Draw back of SGDA :-

* Drawback of VNS :-

→ It is very prone to local mining because in SGD we learn from single data point at a time hence we don't have sufficient amount of learning to overcome local minima.