Deep Learning Examples.

1. Self driving car
2. Digital Assistants : Alexa, Siri
3. Text to speech Synthesis
4. Language identificatn
5. Optical character recognition
6. Translation.
7. Image Caption Generation - Surveillance camera.
8. Video to textual description (Annotatn to video).
9. Synthetic parsing for NLP

Artificial Intelligence.

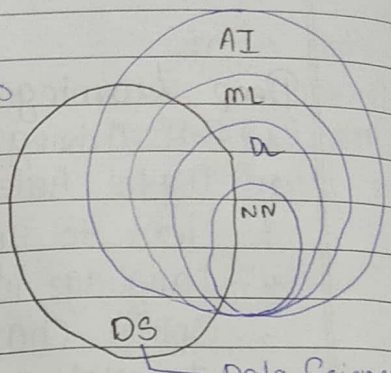
- It is just a program or machines ability to learn & think for itself, in such a way that it is capable of developing skills as it practices & gains experience. Its purpose is to support & enhance human activities.
- Enables machine to think w/o human intervention & take decision.
- eg. are given above.

Machine Learning.

- Subset of AI.
- Provides statistical tools to explore & analyze data
- Method of data analysis that automates analytical model building
- Branch of AI based on the idea that systems can learn from data, identify patterns etc. & make decision

Neural Networks.

1) makes use of neurons to transmit data as input & output values.
2) They are used to transfer data by network of connection.
3) 3 layers
- Input layer
- Hidden layer
- o/p layer



Deep Learning.

1. Composed of several hidden layers while neural networks consists upto 3 layers.
2. subset of ML.
3. This got created for like can machine to learn to think how actually human brain learn to think.
4. In DL, we create architecture called multi-neural network architecture. Idea of Deep neural network is to mimic human brain.

Data Science.

1. will work on above AI, ML, DL, using mathematical tools such as statistics, probability, Linear algebra, Differential calculus etc.

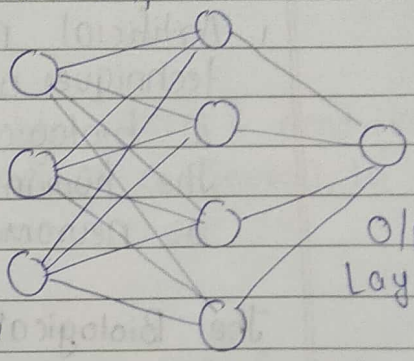
Some more examples of applied deep learning include the foll.

1. Automated image sharpening.
2. Automating image upscaling.
3. Wavenet: generating human speech that can imitate anyone's voice.
4. Speech reconstruction from silent video.
5. Image autofill for missing regions.
6. Automated image captioning (process of generating textual description).

7. Turning hand-drawn doodles into stylized artwork.

- In Deep learning, there are 3 techniques.
  1. ANNs : Problems that involved with the numbers are solved with the help of ANN.

fig. Basic NN.



  2. Convolution NN (CNN): If the i/p is in images, then we use CNN. Advanced CNN known as Transfer learning.
  3. Recurrent Neural N/w : If data is time series of kind of data, we use RNN.

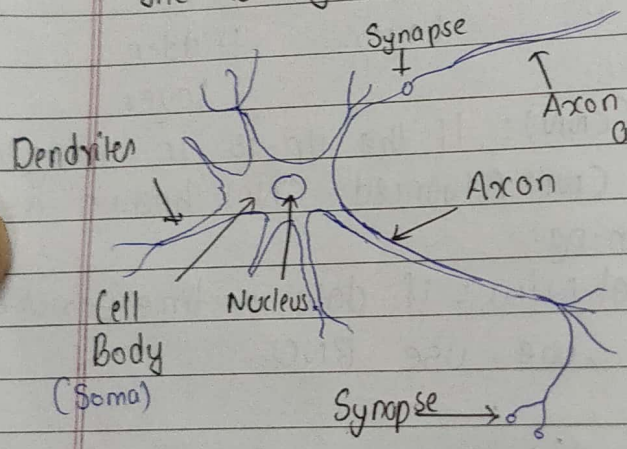
Tue.
07/01/20

INTRODUCTION TO NEURAL NETWORKS

* Introduction (Josh).

1. Artificial neural nlw (ANNs) are mlc learning techniques which simulate the mech of learning in biological organisms.
The human nervous system contains cells, referred as neurons.

The Biological Neuron.



Dendrites - i/p to neuron (Synapse).

- The biological neuron is a nerve cell that provides the fundamental functional unit for the nervous systems of all animals.
- Neurons exist to communicate info & pass electrochemical impulses.

- The neuron is made up of a nerve cell consisting of soma that has many dendrites but only one axon.

- Axons are nerve cell fibers with special cellular extensions that comes from the cell body. The axon can branch hundreds of times.
- Dendrites are thin structures that extend from main body.

- Synapses
  1. Synapses - connecting jn blw axon & dendrites.
  2. Sends sig from the axon of a neuron to dendrite of another neuron.

- Dendrites.
  1. Dendrites have fibers branching out from the soma in a bushy nlw around the nerve cell.
  2. Allows the cell to receive signals from connected neighbouring neurons.
  3. Able to perform multiplication by the dendrites weight value.
  4. ↑ or ↓ in ratio of neuro transmitters.

- Axon.
  1. Single, long fiber extending from the main soma. (1 cm in length & 100 times dia of soma).
  2. Branches 100s of times & connects to other dendrites.

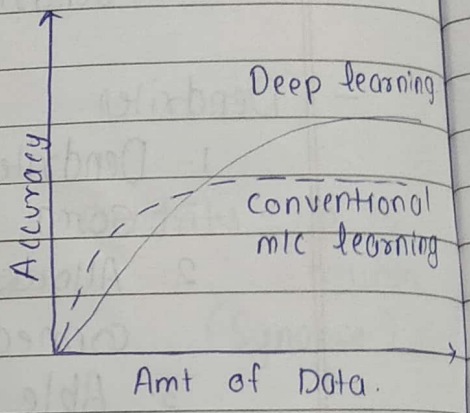
- Information Flow across the biological neuron.
  1. Synapses that ↑ the potential & those that decrease the potential.
  2. Neurons also have been shown to form new connections over time & even migrate.
  3. There combined mechanics

Wed. 08/01/20

- Total connections in the human brain.
  1. Biological NN (brain) are composed of roughly 86 billion neurons connected to many other neurons.
  2. More than 500 trillion connectn blw neurons in the human brain.

* Humans Vs Computers : Stretching the limits of Artificial Intelligence. (Charu).

- for smaller data mlc learing is better coz of more choise, ease of model interpretation.



* Neural Networks (from Josh.)
  1. NN nlws are computational model that shares some properties with animal brain in which many simple units are working in parallel with no centralized control unit.
  2. The behaviour of NN is shaped by its nlw arch.
  3. Networks archi can be defined by.

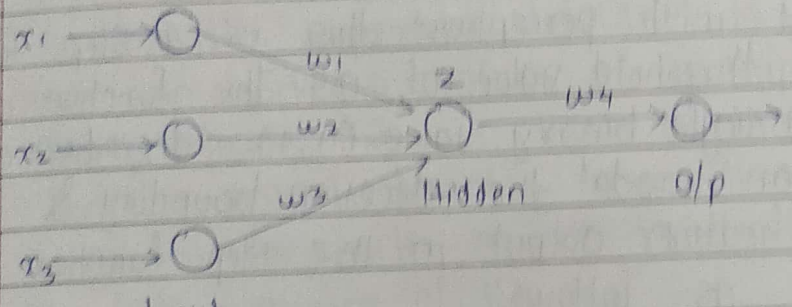
     No. of neurons
     No. of layers
     Types of connectn blw them.

From biological to artificial.

1. Animal brain is fundamental components of the mind
2. Research has shown ways of mapping functionality of human brain to track signals as they move through neurons

The Perceptron

The perceptron is a linear model used for binary classification. In the field of neural networks the perceptron is considered an artificial neuron.



$y = x_1w_1 + x_2w_2 + x_3w_3 + \text{bias}$

$= Act(y)$

↳ Activation fn.

$= z \cdot w_4$

- Types of Activatn fn.

Act(y) → Sigmoid $= \frac{1}{1+e^{-y}}$

↳ ReLU

The perceptron is a linear-model binary classifier with a simple input-output relationship as depicted.

It shows we're summing n no. of inputs times their associated weights & then sending this "net input" to a step function with a defined threshold.

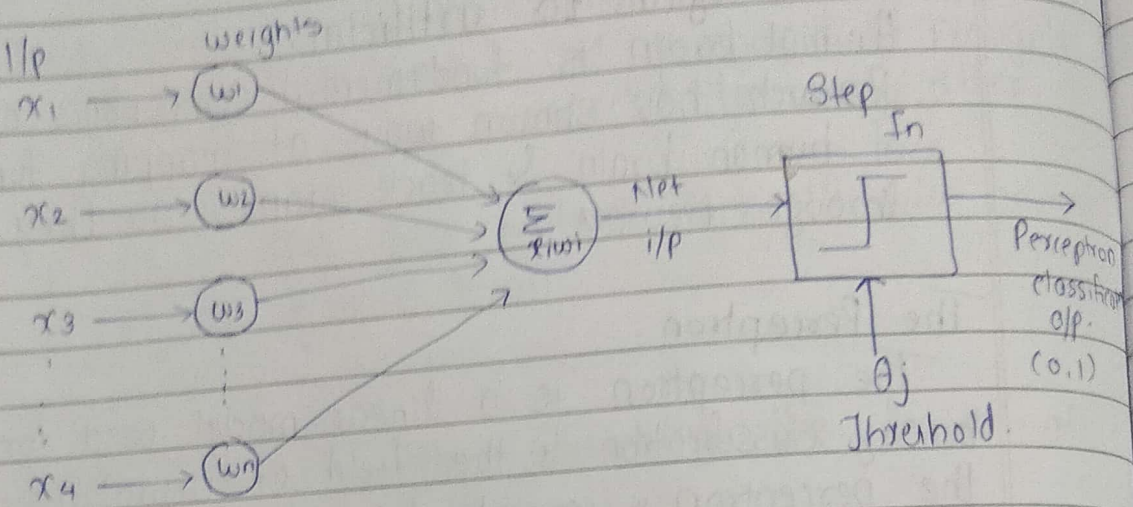


fig. Single layer perceptron.

- Typically with perceptrons, this is a step function with a threshold value of 0.5. This function will output a single binary value (0,1), depending on input.
- We can model the decesion boundary & the classification output in the step function equation as follows:

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x >= 0. \end{cases}$$

- To produce the net input Act fn, we take dot product of i/p & connectn wts.

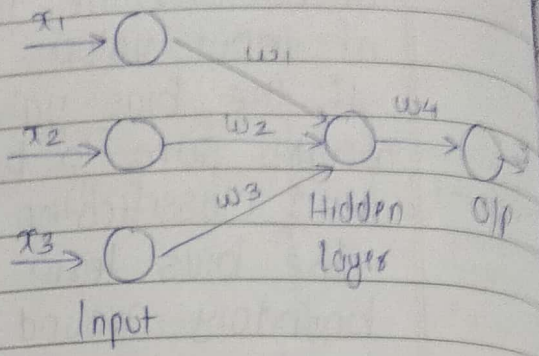
Function Parameter		Description
w	-	Vector of real valued weights on the connectn.
w.x	-	Dot product ($\sum_{i=1}^{n} w_i x_i$)
n.	-	No. of i/ps to perceptron
b	-	The bias term (input values does not affect the bias.

- The o/p of the step fn (activation fn) is the output for the perceptron & gives us a classification of input values.
- If the bias value is -ve, it forces the learned weights sum to be a much greater value to get a 1 classification o/p.
- The bias term in this capacity moves the decision boundary around for the model.
- Input values dont affect the bias, but bias is learned through perceptron learning algo.
- The algo will not terminate if the learning input is not linearly seperable.
- A linearly seperable dataset is one for which we can find the values of a hyperplane that will cleanly divide the two classes of the data set.
- The perceptron learning algorithm initializes the weight vector with small random values or 0.0s at the beginning of training.
- The perceptron learning algorithm takes each input record as we can see in the fig. & computes the output classification to check against the actual classification label.
- To produce the classification, the columns (features) are matched up to weights where n is the no. of dimensions in both our inputs & weights
- The first input value is the bias input, which is always 1.0 because we dont affect the bias input.
- The first weight is our bias term in this diag.
- The dot product of the input vector & weight vector gives us the i/p to our actn fn.
-

Sat
11/01/20

Types of Activation fn.

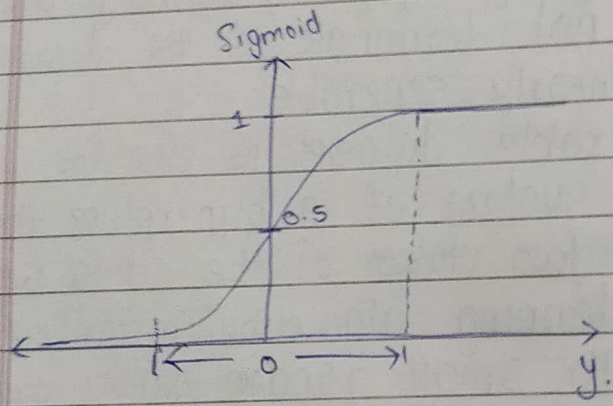
1. Linear
2. Non-Linear
3. Sigmoid (Logistic)
4. Tanh
5. ReLU



Sigmoid fn. $= \frac{1}{1+e^{-y}}$

$y = x_1w_1 + x_2w_2 + x_3w_3 + bias$

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

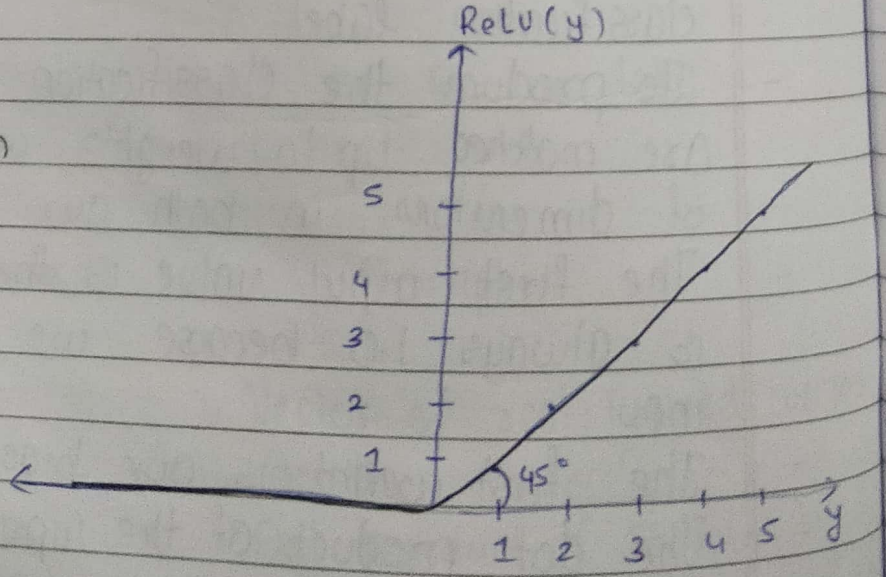


Mostly used in o/p layer.

ReLU → max(y, 0)

y = -ve , max(-ve, 0) = 0

y = +ve , max (+ve, 0) = +ve = y.



Mostly used in hidden layer

* The Basics Architecture of Neural Networks (Charu Agg)

Single Computational Layer : The Perceptron.

1. In any type of application, one would have historical cases in which the class variable is observed & other (current) cass in which the class variable has not yet been observed but needs to be predicted.

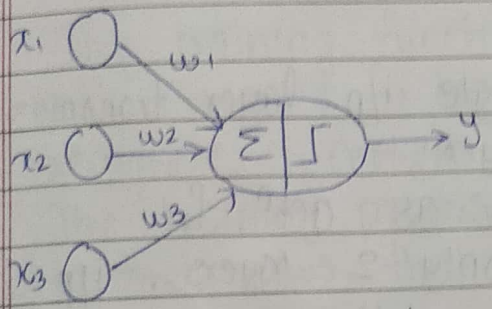


fig Perceptron w/o bias

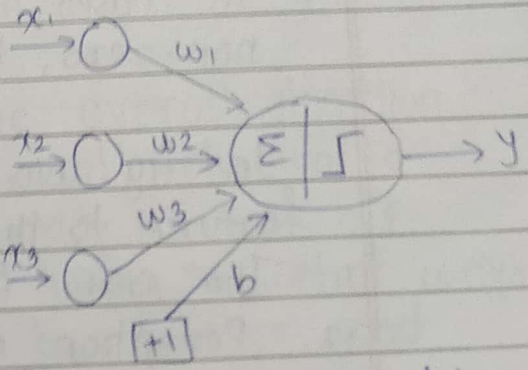


fig with bias.

Fig. basic Arch of Perceptron.

2. The input layer contains d nodes that transmit the d features $\bar{X} = [x1 \ldots x_d]$ with edges of weight $\bar{w} = [w1 \ldots w_d]$ to an o/p node.
3. The input layer does not perform any computation in its own.
4. The linear function $\bar{W} \cdot \bar{X} = \sum_{i=1}^{d} w_i \cdot x_i$ is computed at the output node.
5. Subsequently, the sign of this real value is used in order to predict the dependent variable of X. Therefore, the prediction $\hat{y}$ is computed as follows.

$$\hat{y} = \text{sign}\{\bar{w} \cdot \bar{x}\} = \text{sign}\{\sum_{j=1}^{d} w_j x_j\}$$

6. The sign function maps a real value to either

+1 or -1, which is appropriate for binary classification.

7. Note the cap mark on top of the variable y to indicate that it is predicted value rather than an observed value.

8. The error of the prediction is therefore $\varepsilon(x) = y - \hat{y}$

9. In cases where the error value $\varepsilon(x)$ is non-zero, the weights in the neural

10. In fig w/o bias, single i/p layer transmits the features to the o/p layer.

11. The sign fn serves as actn fn.

12. Perceptron contains only 2 layers, i/p layer only transmits hence not included.

13. Therefore it has only one computational layer & it is called as single layer feed forward nlw.

14. In many settings, there is an invariant part of prediction which is referred to as bias.

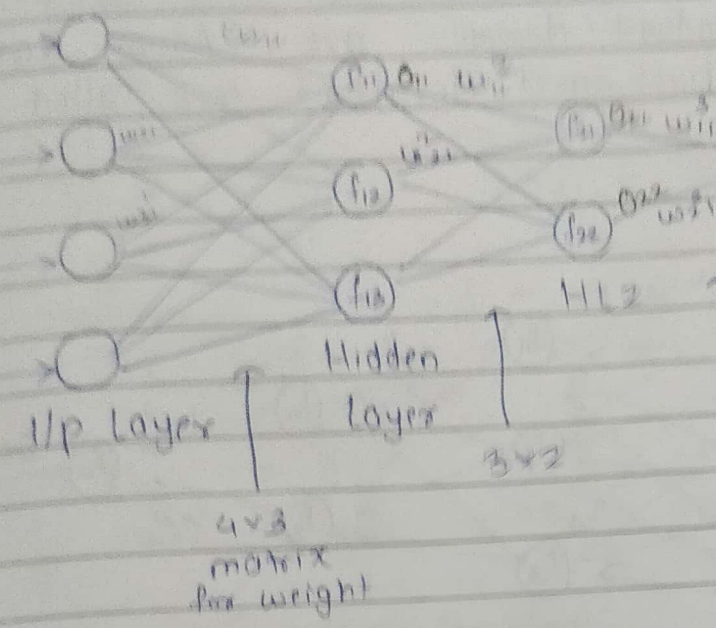
eg. consider a setting in which the feature variables are mean centered, but the mean of the binary class prediction from {-1, +1} is not 0.

15. This will tend to occur in situations in which the binary classifica distribution is highly imbalanced

Invariant definition: an entity, quantity etc that is unaltered by a particular transformation.

Use of additional input, which is called bias, of neuron improves properties of the neuron. It

- The bias value allows the actn fn to be shifted left or right, to better fit the data. It is always 1.
- A weight represent the strength of connectn blw units. If the weight from node 1 to node 2 has greater magnitude, it means neuron 1 has greater influence over neuron 2.
- Weights in NN can, & will, become +ve or -ve depending on the training data.
-
- The training function is the overall algorithm that is used to train the neural network to recognize a certain input & map it to an o/p.
- The training process involves finding a set of weights in the nlws that proves to be good or good enough at solving specific problem.
- A learning fn deals with individual wts & thresholds & decides how these would be manipulated.



$$w_{new} = w_{old} + \eta \cdot \frac{dL}{dw}$$

η → learning rate.

Loss fn = $(y - \hat{y})^2$ → reduce optimizer.

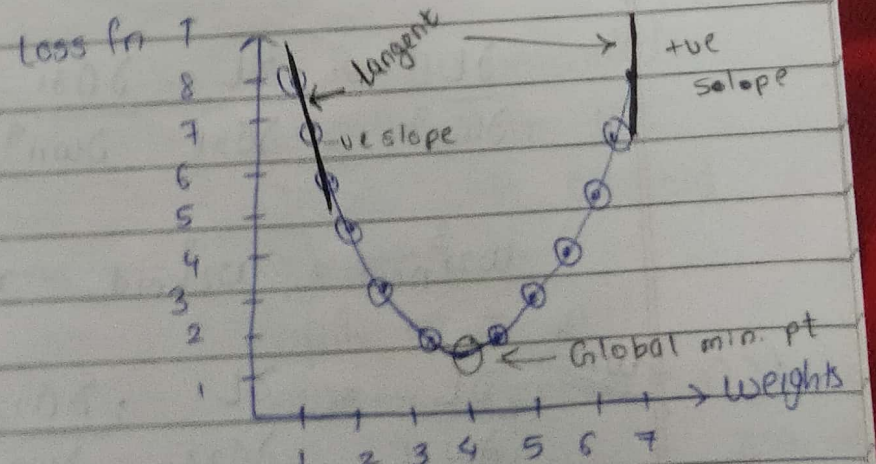
$\hat{y}$ → predicted value.

$$\Rightarrow \frac{1}{m}(y - \hat{y})^2$$

where m = no. of o/p neurons.

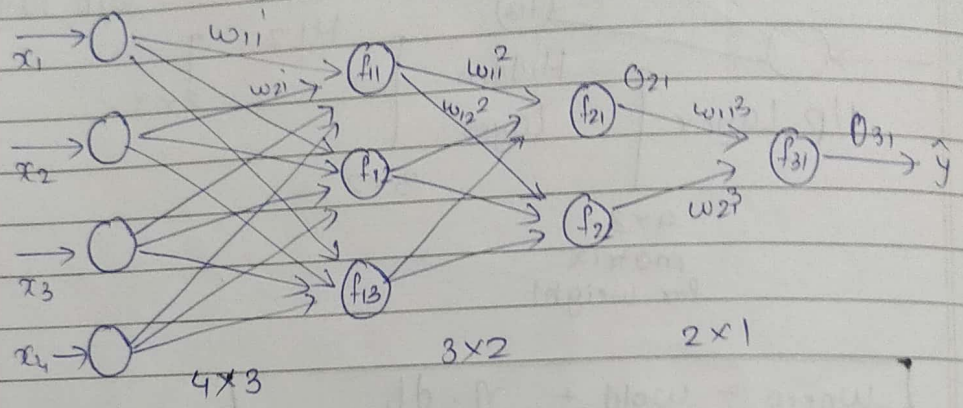
Gradient descent

update the weights such a way that $\hat{y}$ to be $\approx$ to y. find that max efficient weights to be updated. to get the best predicted value $\hat{y}$.



- Optimizer will reduce the loss value.
- Some gradient decents are - Adam, SGD
- Forward backward iterations are called as epoch

MNN



I/p Layer | Hidden Layer 1 | Hidden Layer 2 | O/p Layer

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

$$w_{11}^3 \text{new} = w_{11}^3 \text{old} - \eta \left[\frac{\partial L}{\partial w_{11}^3}\right]$$

w_{11}^3 $O_{31} = w_{11}^3 . O_{21} + w_{21}^3 . O_{22}$ Chain rule on derivative in NN

$$\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{11}^3}$$

$$w_{21}^3 \text{new} = w_{21}^3 \text{old} - \eta \frac{\partial L}{\partial w_{21}^3}$$

w_{21}^3 $$\frac{\partial L}{\partial w_{21}} = \frac{\partial L}{\partial O_{22}} \cdot \frac{\partial O_{22}}{\partial w_{21}}$$

w_{11}^2

$$w_{11}^2{}_{new} = w_{11}^2{}_{old} - \eta \frac{\partial L}{\partial w_{11}^2}$$

$$O_{21} = w_{11}^2 \cdot O_{11}$$

$$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial O_{31}} \times \frac{\partial O_{31}}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial w_{11}^2}$$

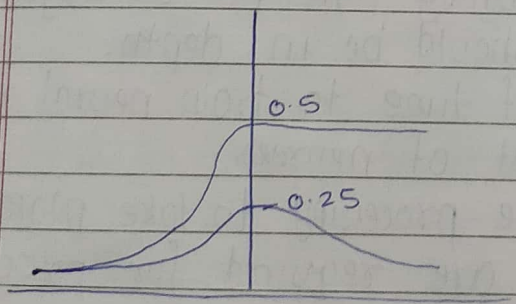
$$w_{12}^2{}_{new} = w_{12}^2{}_{old} - \eta \frac{\partial L}{\partial w_{12}^2}$$

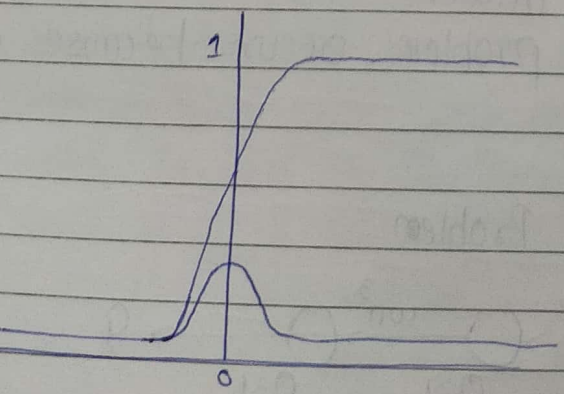
$$O_{22} = w_{12}^2 \cdot O_{12}$$

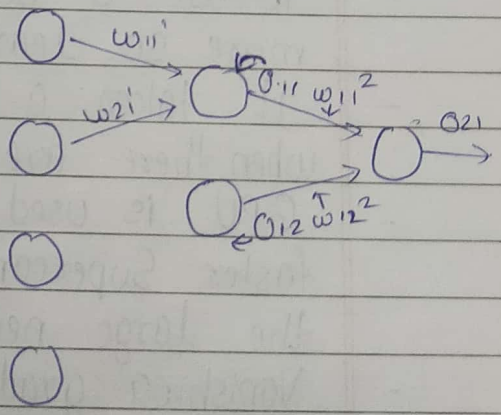
$$\frac{\partial L}{\partial w_{12}^2} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{22}} \cdot \frac{\partial O_{22}}{\partial w_{12}^2}$$

Vanishing Gradient Problem.

$$w_{11}'{}_{new} = w_{11}'{}_{old} - \eta \frac{\partial L}{\partial w_{11}'}$$







- It is an unwanted situation in NN
- Occurs with sigmoid activation fn
- updation of w_{11}

$$w_{11}{}_{new} = w_{11}{}_{old} - \eta \frac{\partial L}{\partial w_{11}{}_{old}}$$

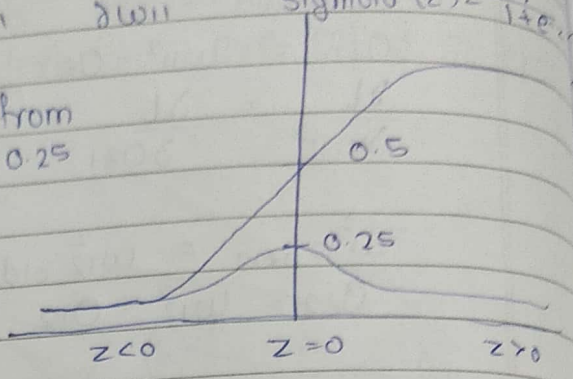
Page

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

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial w_{11}}$$

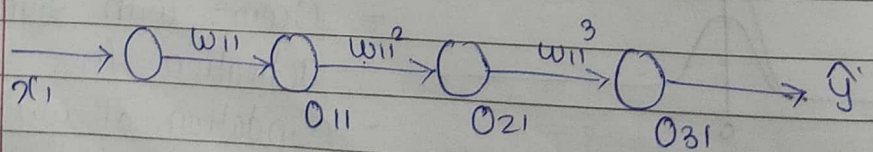
$\frac{\partial (\text{sigmoid})}{\partial z}$ varies from $0 - 0.25$

sigmoid $(z) = \frac{1}{1+e^{-z}}$



- Since $\frac{\eta \partial L}{\partial w_{11}}$ is a very small value that is subtracted with w_{11}.
- Using ReLU function the $\frac{\eta \partial L}{\partial w_{old}}$ will give a higher value which gives the difference.
- Incase of deep learning, the no. of layers are more, so learning should be in depth.
- It takes a lot of time to train neural n/w when there are a lot of neurons.
- GPU is used for the processing to take place faster. Supercomputer are required for processing the large neural networks.
- Vanishing gradient problem occurs because of activatn function.

Exploding Gradient Problem.



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

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial O_{11}} \cdot \frac{\partial O_{11}}{\partial w_{11}}$$

$O_{21} = Act(z)$

$z = w_{11}^2 \times O_{11} + b_2$

result of step 1.

bias to then 2nd neuron.

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial w_{11}}$$

$$\frac{\partial O_{21}}{\partial O_{11}} = \frac{\partial [Act(z)]}{\partial O_{11}} = \frac{\partial (Act(z))}{\partial (z)} \times \frac{\partial (z)}{\partial O_{11}} \quad \text{(chain rule)}$$

(0 to 0.25)

Substituting for z,

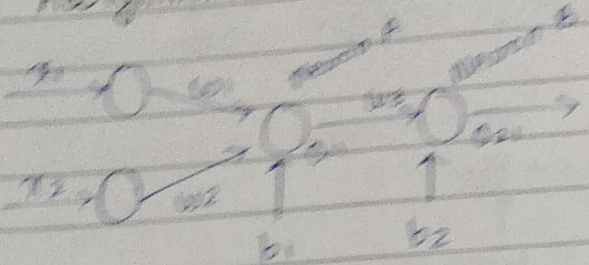
$$[0 \text{ to } 0.25] \times \frac{\partial [w_{11}^2 \times O_{11} + b_2]}{\partial O_{11}}$$

$$(0 \text{ to } 0.25) \times [w_{11}^2 + 0]$$

$$(0 \text{ to } 0.25) \times w_{11}^2$$

This will usually be a large value when compared to vanishing gradient problem. So there will be a variation in w_{11} new & w_{11} old. So it will never reach a global minima region.

Assignment Question.



$x_1 = 0.05$ $b_1 = 0.35$
$x_2 = 0.1$ $b_2 = 0.6$

Actual o/p $(y) = 0.01$
$\eta = 0.01$
$w_1 = 0.15$, $w_2 = 0.2$
$w_3 = 0.4$.

Analyse the given NN for one epoch with
1) Forward Propagation 2) Backward Propagation

Neuron A

$$z_1 = w_1 x_1 + w_2 x_2 + b_1 = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35$$
$$= 0.3775$$

$$o_{11} = Act(z_1) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-0.3775}} = \frac{1}{1.6856} = 0.5933$$

Neuron B

$$z_2 = w_3 o_{11} + b_2 = 0.4 \times 0.5933 + 0.6 = 0.8373.$$
$$o_{21} = Act(z_2) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-0.8373}} = 0.411 = \hat{y}$$

$$Error = (y - \hat{y}) = (0.01 - 0.411) = -0.401$$

$$Loss = (y - \hat{y})^2 = (-0.401)^2 = 0.1608.$$

To update the weight w_3,

$$w_{3new} = w_{3old} - \eta \frac{\partial L}{\partial w_3} \quad \text{..... (1)}$$

Now,

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial w_3} \quad \text{..... (2)}$$

we know,

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

$$\frac{\partial L}{\partial O_{21}} = \frac{\partial [(y - O_{21})^2]}{\partial O_{21}} = 2(y - O_{21}) \times (-1)$$

$$= -2(y - O_{21})$$

$$= -2(0.01 - 0.411) = 0.8$$

$$\frac{\partial O_{21}}{\partial w_3} = \frac{\partial O_{21}}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_3}$$

we know, $O_{21} = Act(z_2) = \frac{1}{1 + e^{-z_2}}$

$$\frac{\partial O_{21}}{\partial z_2} = \frac{\partial (Act(z_2))}{\partial z_2} = Act(z_2) \times (1 - Act(z_2))$$

$$= 0.411 \times (1 - 0.421) = 0.242$$

Now, $z_2 = w_3 O_{11} + b_2$

$$\frac{\partial z_2}{\partial w_3} = O_{11} + 0 = 0.5933$$

In eq (2.)

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial O_{21}} \times \left\{ \frac{\partial O_{21}}{\partial z_2} \times \frac{\partial z_2}{\partial w_3} \right\} = 0.802 \times 0.242 \times 0.5933 = 0.1153$$

In eqn (1)

$w_{3new} = 0.4 - 0.01 \times 0.1153 = 0.3988$

Now to update the weight w_1,

$$w_{1new} = w_{1old} - \eta \frac{\partial L}{\partial w_1} \quad \ldots (3)$$

Now,

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial z_2} \times \frac{\partial z_2}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}$$

$\frac{\partial}{\partial w_1}[w_1 x + w_2 x_2 + b_1] = x_1 + 0 + 0 + 0$

$\frac{\partial}{\partial z_1}(Act(z_1)) = Act(z_1) \times (1 - Act(z_1))$

$= 0$

$\frac{\partial}{\partial O_{11}}[w_3 O_{11} + b_2] = w_3 + 0$

$= 0.4$

0.802 0.242

Now, $= 0.802 \times 0.242 \times 0.4 \times 0.2413 \times 0.05 = 0.94$

$w_{1new} = 0.15 - 0.01 \times 0.93 \times 10^{-3} = \underline{0.14999}$

next class — 04/02/20

Wed.
05/02/20

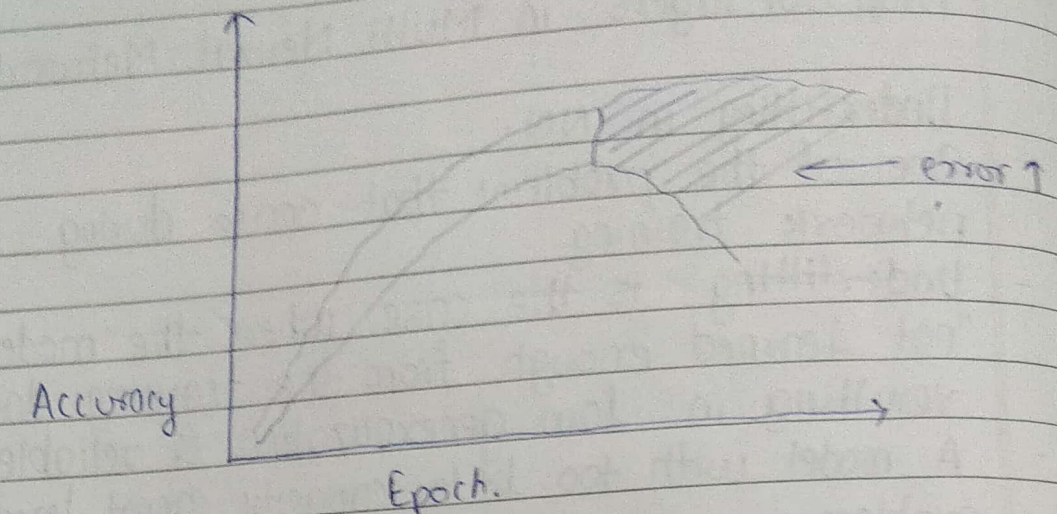
Drop out layers in Multi Neural Network.

Underfitting in NN.

- One of the problems that occur during neural network training.
- Underfitting is the case where the model has 'not learned enough' from the training data, resulting in low generalization & reliable predictn
- A model with too little capacity cannot learn the problem.
- Too little learning will make the model to perform poorly on the training dataset & on new data.
- An underfit model has high bias & low variance

Overfitting in NN

- One of the problems that occur during neural network training is called overfitting.
- The error on the training set is driven to a very small value, but when new data is presented to the network the error is large.
- The network has memorized the training examples, but it has not learned to generalize to new situations.
- A model with too much capacity can learn it too well & overfit the training dataset.
- With too much learning, the model will perform well on the training dataset & poorly on new data.
- An overfit model has low bias & high variance. The model learns the training data too well & performance varies widely with new unseen.



Good fit Model.

A model that suitably learns the training dataset & generalizes well to the old ~~outdoor~~ dataset

How to handle underfitting problem?

Underfitting can easily be addressed by increasing the capacity of the nlw.

How do you handle overfitting problem?

(i) Reduce the nlw's capacity by removing layers or reducing the number of neuron elements in the hidden layers.

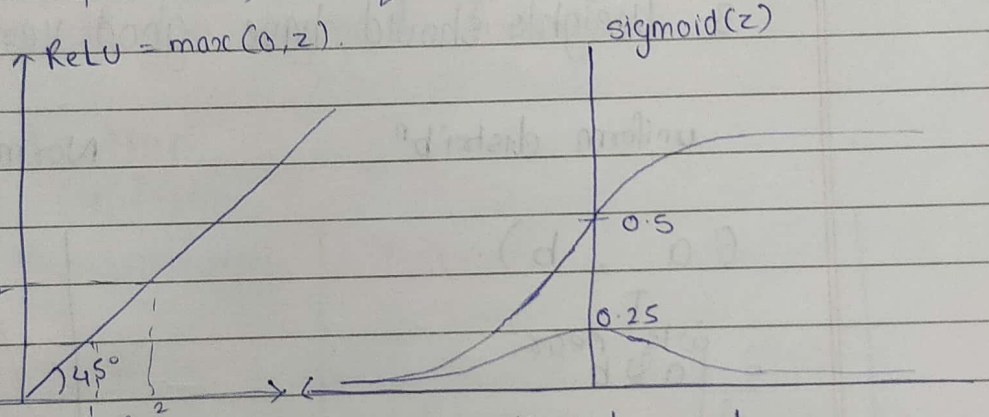
(ii) Use Dropout layers which will randomly remove certain features by setting them to zero

- Dropout layers are an easy & effective way to prevent overfitting in NN.
- A dropout layer randomly drops some of the connections blw layers. This helps to prevent overfitting We reduce the weights of the model to small during learning.

These techniques not only reduce overfitting but they can also lead to faster optimization of the model & better overall performance.

- In deep NN, there are many weights & bias parameters. When we have huge weights & bias parameters, the NN tends to overfit the data set problem on a particular use scales.
- In deep NN, underfit will never happen. Usually underfitting happens with one layer NN.

To fix this problem, technique used called dropout.



ReLU = max(0,z)

$\frac{d}{d}(ReLU) = 1$

transformation value = 1

sigmoid $\frac{1}{1+e^{-z}}$

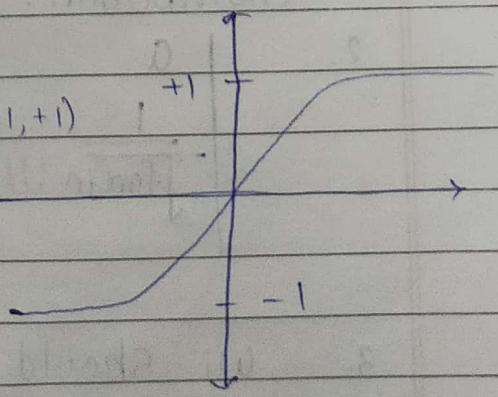
$\frac{d}{d}(sigmoid) = 0.5 \cdot (1 - 0.5)$

$= 0.25$

transformation value.

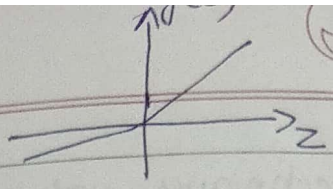
tanh Activation fn.

$$\tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}} = (-1, +1)$$



d(tanh) ⇒ 0 to 1.

Tue.
11/02/20



Leaky ReLu

$$f(x) = \begin{cases} z & \text{if } x > 0 \\ 0.01z & \text{otherwise.} \end{cases}$$

Till here in Session

Weight Initialization Techniques in NN

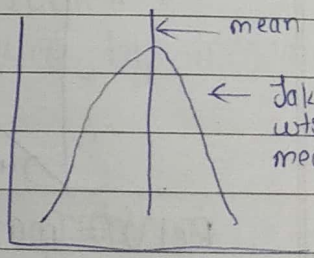
Key Points while initialising the weights.
1. Weights should be small.
2. Weights should not be same.
3. Weights should have good variance.

uniform distribⁿ

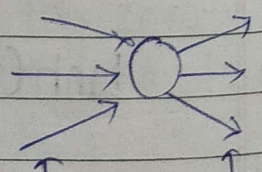
(a, b).

↑
wts near a & b

Normal distributⁿ



1. Uniform distributⁿ technique.
   1. wts are taken & sampled from uniform distribution.
   2. $\left| \frac{a}{\frac{1}{\sqrt{\text{fan-in}}}} + \frac{b}{\frac{1}{\sqrt{\text{fan-in}}}} \right|$



fan-in ⇊ no. of inputs.

fan-out ⇊ no. of o

   3. Wi should be in the above range.
   4. Sigmoid actⁿ fn works well with this distributⁿ.

GD → Error = $(y-\hat{y})$

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

cost = $\sum_{i=1}^{100}(y_i-\hat{y})^2$

2. Xaviers / Gorat distribution.

(i) Xavier / Gorat uniform distribution.

- wi will be in the range from.

$$\left[-\frac{\sqrt{6}}{\sqrt{fan\_in + fan\_out}} \quad , \quad +\frac{\sqrt{6}}{\sqrt{fan\_in + fan\_out}} \right]$$

(ii) Xavier / Gorat normal distribution.

- w_i = Normal (mean = 0, std dev = σ)

$$\sigma = \sqrt{\frac{2}{fan\_in + fan\_out}}$$

- Works well for sigmoid function.

3. He initialization.

(i) He uniform

w_i = uniform. $\left[-\sqrt{\frac{6}{fan\_in}} \quad , \quad +\sqrt{\frac{6}{fan\_in}} \right]$

(ii) He normal

- w_i = normal (mean = 0, std dev = σ)

$$\sigma = \sqrt{\frac{2}{fan\_in}}$$

- Works well with ReLU act^n fn.

Optimizers

Gradient Descent	Stoachastic Gradient Descent
Considers all the data sets at once	

Wed.
12/02/20

Stochastic Gradient Descent.

Error → $(y-\hat{y})$
Loss → $(y-\hat{y})^2$
Cost → $(y-\hat{y})^2$

} Instead of considering all data like GD, it only considers one data at a time.

Note: speed (GD) < speed (SGD)
accuracy (GD) > accuracy (SGD)

- To increase the speed, consider group of data at a time (i.e subset of data). This maintains the accuracy (slightly less than b4) but ~~less~~ much faster than before.
This is called as Minibatch SGD.
error = $(y-\hat{y})$
Loss = $(y-\hat{y})^2$
Cost = $\sum_{i=1}^{k}(y-\hat{y})^2$ where $k<n$

- Minibatch SGD is faster than GD.
It is a bad optimizer but converges to global minima faster than G.D.

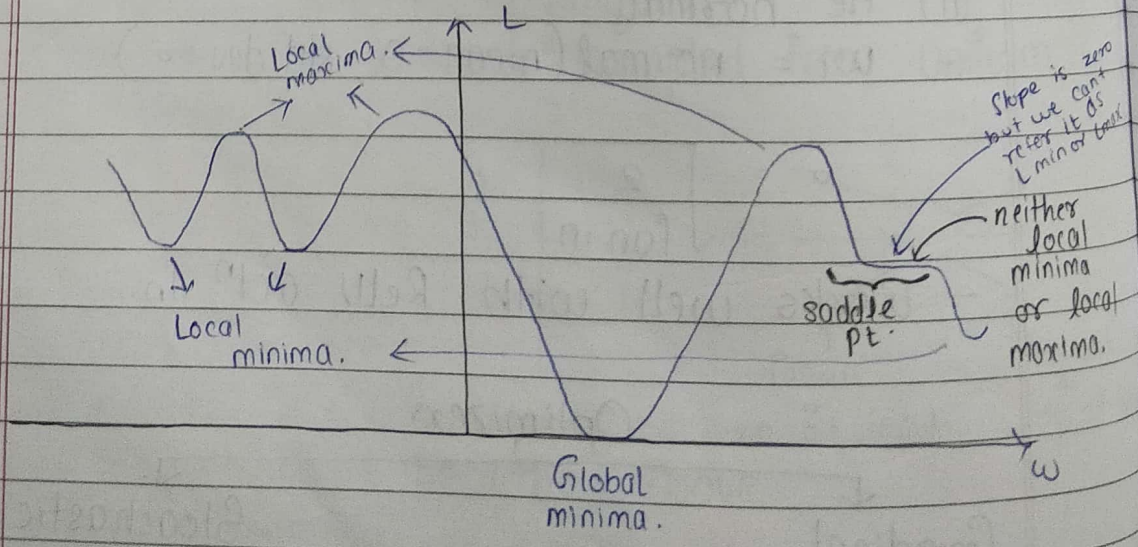
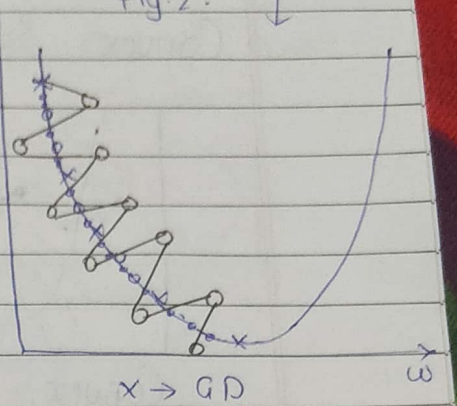


Fig. 1. Gradient Descent curve.

Here noise is getting introduced in data pt/assmate
This is an unwanted data which should be removed in NN
-noisy data

- In GD the pts will be far away & on joining a smooth curve isn't obtained.
- In SGD there will be a large no. of pts ∴ more accurary & on joining we get a smoother curve.

Fig.2.



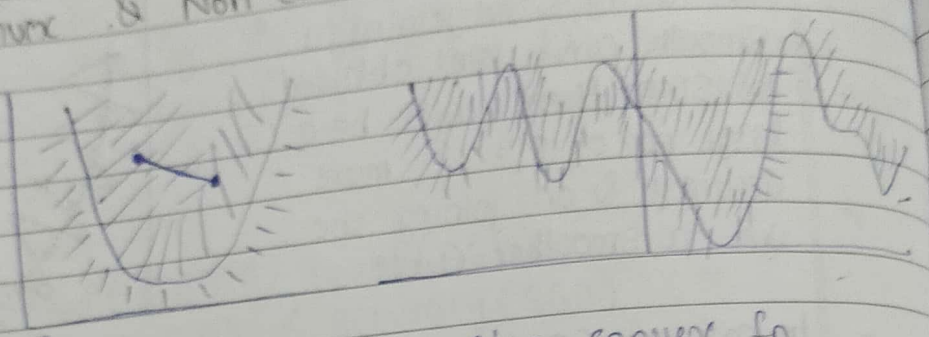
- In case of GD, memory & computational power reqd is more when compared to MSGD
- In that sense MSGD is faster than GD
- This is when the total data is very huge.

- In fig-1. the loss ↑↓ ↑↓ ... At a certain point it reaches global minima (where $y - \hat{y} \approx 0$) (Here the slope will be zero).
The remaining ones are local minima.
- The global minima is a point where the the gradient descent curve completely convergen
There is only one in each graph.

- The local minima are the points in which only in that locality, that is the best converging point.
- There will be advanced techniques with optimizer where these problems can be solved.
- The peak on the top are called local minima
- The saddle point occurs occurs blw local minima & local maxima. The slope here is zero. It isn't global minima either.

$E = \frac{1}{2}(y_1 - \bar{y})^2 + \frac{1}{2}(y_2 - \bar{y}_2)^2$

Convex & Non-convex function.



Convex function

Non-convex fn

- Area is divided into 2.
- Mark 2 pts & join them.
- The line remains in same region.
- Most Deep Neural Nlw will have non-convex fn
- Even if it crosses, it will be just once

- Here the pts will cross more than 1 region, multiple times.

Exponentially moving average.

time	t_1	t_2	t_3	t_4	t_n
data pts	b_1	b_2	b_3	b_4 ----	b_n

$V_1 = b_1$

$V_2 = \gamma V_1 + b_2 \quad 0 < \gamma \leq 1$

let $\gamma = 0.5$

$\therefore V_2 = 0.5 V_1 + b_2$

$V_3 = \gamma V_2 + b_3$

$= \gamma(\gamma V_1 + b_2) + b_3$

$V_3 = \gamma^2 b_1 + \gamma b_2 + b_3$

$V_3 = 0.25 b_1 + 0.5 b_2 + b_3$

This is included in NN to come out of noise data.

momentum.

Now, $$w_{new} = w_{old} - \left(\gamma V_{t-1} + \eta \frac{\partial L}{\partial old}\right)$$

$$V_{t-1} = \frac{\partial L}{\partial w}\Big|_t + \gamma \frac{dL}{dw}\Big|_{t-1} + \gamma^2 \frac{dL}{dw}\Big|_{t-2} + \cdots\cdots$$

SGD with momentum.

Smoothens the curve by removing noise from data points.

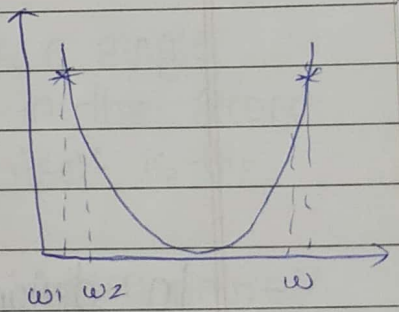
Adaptive gradient descent optimizer. (Adagrad)

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

Dense → non-zero

Sparse → zero.

x_1	x_2	x_3	.....	x_n
0	1	1	1	0
1	1	0	1	0
0	0	0	1	1
0	0	1	0	0
1	1	1	0	1
0	0	1	1	1



$$w_t = w_{t-1} - \eta_t \frac{\partial L}{\partial w_{t-1}}$$

$$\eta_t = \frac{\eta \to 0.01}{\sqrt{\alpha_t + \epsilon}}$$

← small +ve no. (so that if $\alpha_t = 0$, η_t shld not become ∞)

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_i}\right)^2$$

Exponentially weighted avg :

$$w_{avg} = \gamma w_{avg_{(t-1)}} + (1-\gamma) \frac{\partial L}{\partial w_{avg_{(t-1)}}}$$

$\gamma = 0$ to L

Wed
26/02/20

Hyperparameter:
Parameter that is set before training the NN.

eg { I/P
weights
tuning hyperparameter.

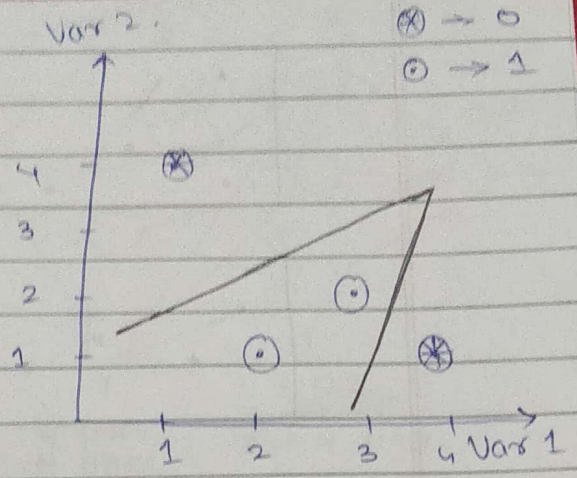
- These are used to take care of learning learning rate
- if these are ~~y~~ are not used then we may have a problem of vanishing gradient.

eg.

Variable 1	Variable 2	Class label
1	4	0
4	1	0
3	2	1
2	1	1

- In designing ANN archi,
  - start by selecting the no. of neurons in the input & output layers.
  - 2 variables as inputs & hence 2 input nodes.
  A binary classification problem, 1 output neuron
- for more than 2 classes, we'd need to create an output neuron for each class.
- To deducing the best number of hidden layers & neurons is to visualize the samples in a 2D graph.
  "What is the minimum no. of connected lines to separate the classes?"
- No. of lines represents the no. of hidden neurons across all hidden layers.

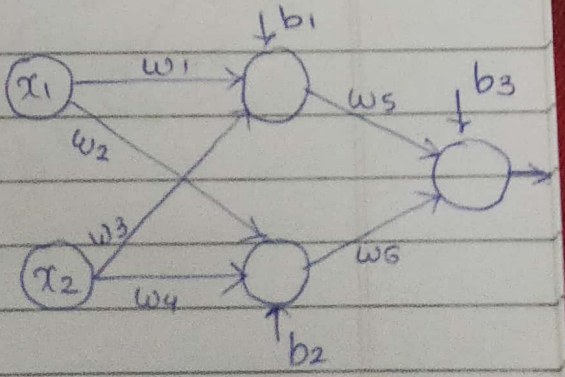
- In this example, there is overlap blw the two classes
- Thus we cannot use just a single line to split the data. Minimum no. of lines is 2.



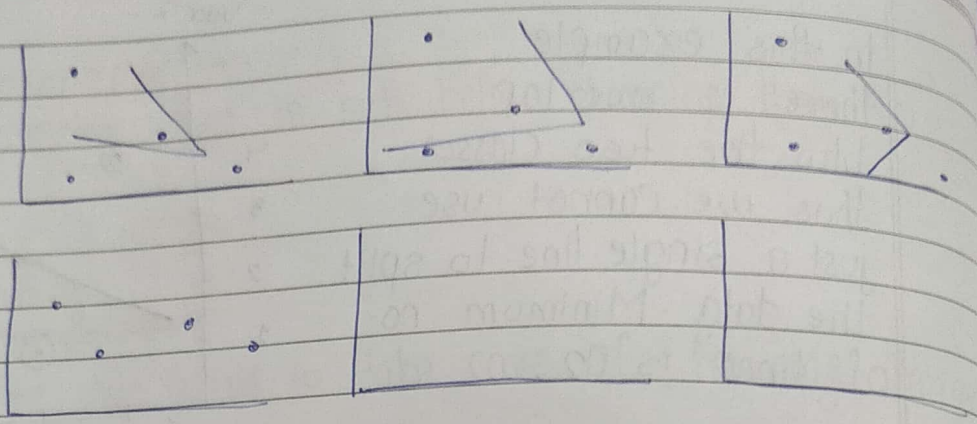
- No. of required lines is 2, NN will have 2 hidden neurons in the first hidden layer, where each neuron produces a line.

"How many hidden layers are required?"

- The preceding hidden layer connects these lines.
- The no. of connections defines the number of hidden neurons in the next hidden layer.
- Because there are just 2 lines, with a single connection (ie just single hidden neuron in the second hidden layer). The second hidden layer itself is the output layer.
- To conclude, 1 hidden layer with 2 hidden neuron.

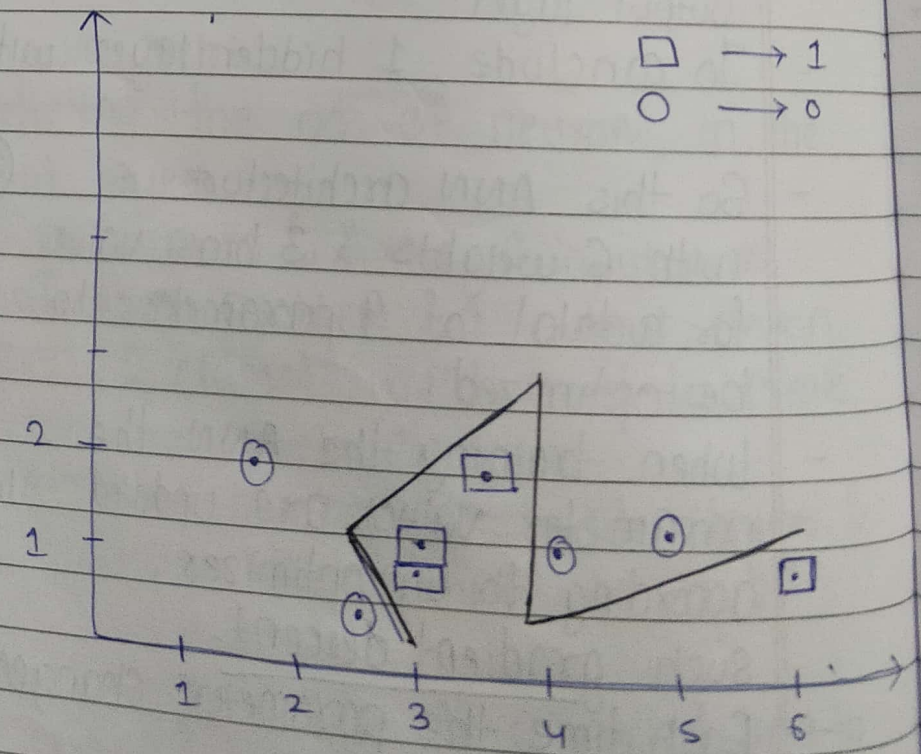


- So this ANN architecture is with 6 weights & 3 bias values for a total of 9 parameters to be optimized.
- When training the ANN, the parameter values are updated according to an optimizer, such gradient descent.
- Each time the parameters changes, the lines are rearranged.

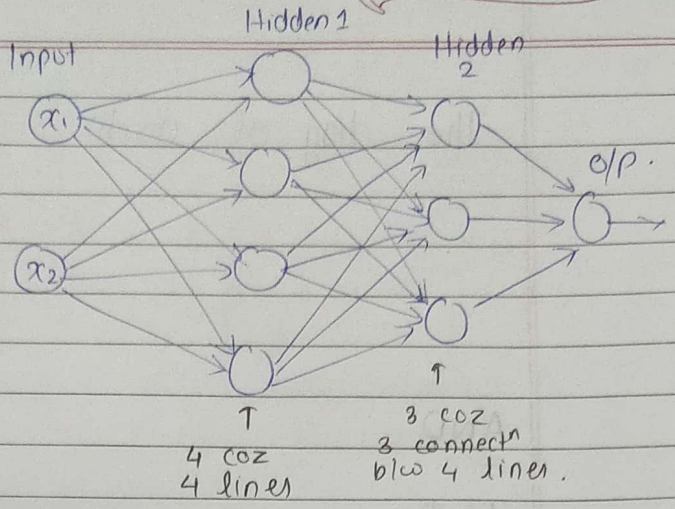


eg. 2.

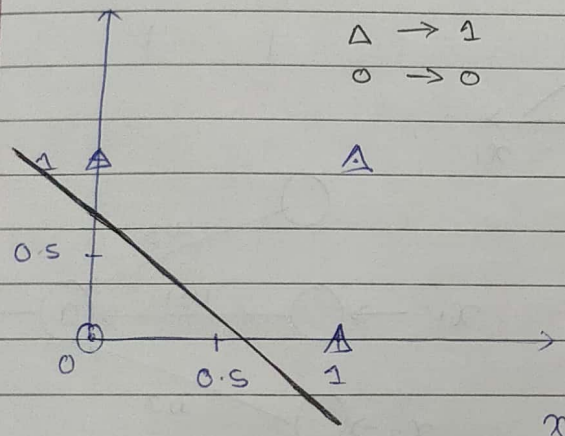
Var 1	Var 2	Class Label.
5	1.5	0
4	1.25	0
1.5	2	0
2.5	0.5	0
6	1	1
3.5	2	1
3	1.5	1
3	1	1



∴ The archi



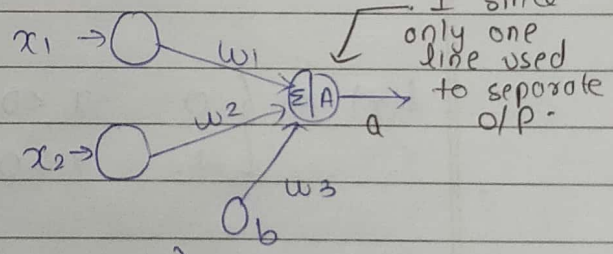
OR gate implementatn in NN



Δ → 1
o → o

OR gate

x_1	x_2	o/p
0	0	0
0	1	1
1	0	1
1	1	1



The o/p will be

o/p = (a)

$(a) = Activatn(w_1 \cdot x_1 + w_2 x_2 + w_3 b)$.

let $b = 1$

$w_3 = -0.5$

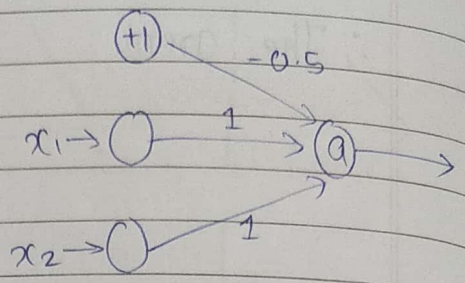
$w_1 = w_2 = 1$.

$\therefore \quad a = Act(x_1 + x_2 - 0.5)$

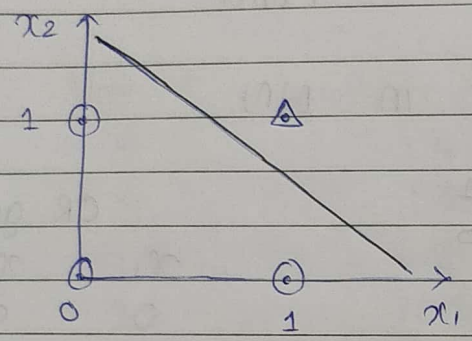
∴

x_1	x_2	x_1 OR x_2	$x_1+x_2-0.5$	a.
0	0	0	-0.5	0
0	1	1	0.5	1
1	0	1	0.5	1
1	1	1	1.5	1

The diag of archi

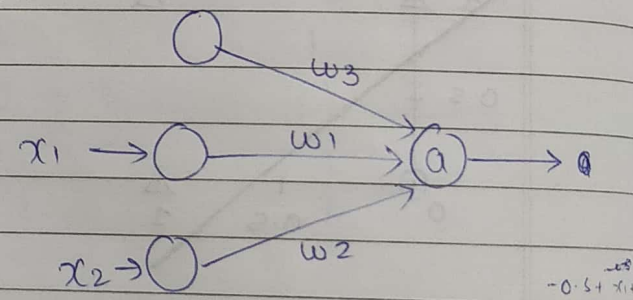


AND

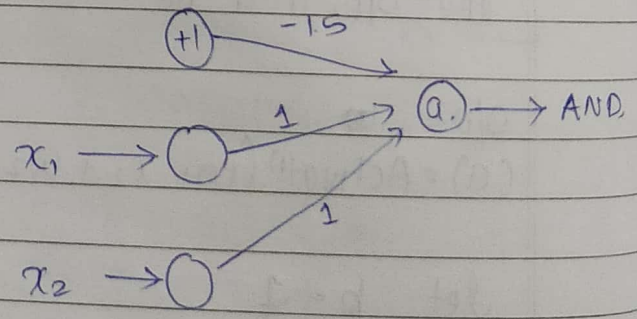


x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	1
1	1	1

a = Act



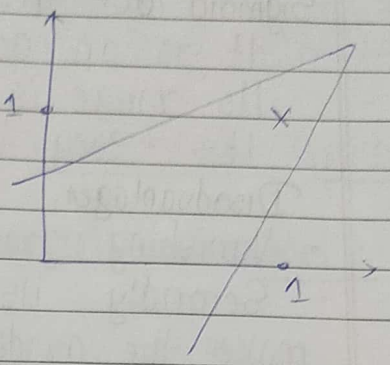
$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$



X	y	X AND y	$-1.5 + X_1 + X_2$	$a = f(w_1x_1 + w_2x_2 + w_3b)$
0	0	0	-1.5	0
0	1	0	-0.5	0
1	0	0	-0.5	0
1	1	1	0.5	1

XNOR

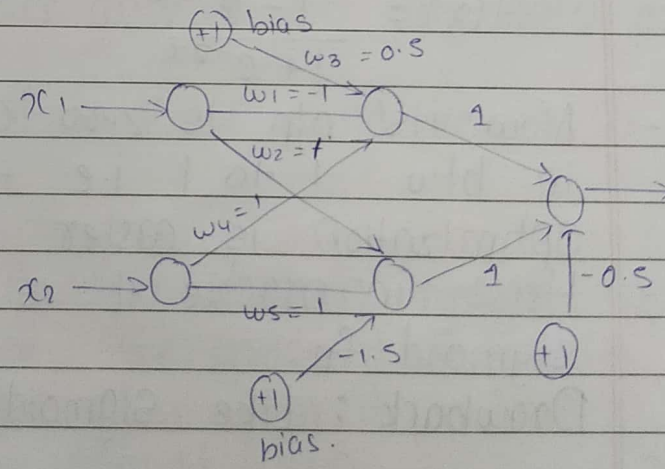
X_1	X_2	X_1 XNOR X_2
0	0	1
0	1	0
1	0	0
1	1	1



$$XNOR = \overline{X_1 \bar{X}_2 + \bar{X}_1 X_2}$$

$$X_1 \text{ XNOR } X_2 = \text{NOT}(X_1 \text{ XOR } X_2) = \text{NOT}\left[(A+B)\cdot(\bar{A}+\bar{B})\right]$$
$$= (\bar{A}+B)' + (A+\bar{B})' = (A\cdot\bar{B}) + (\bar{A}\cdot B)$$

Inputs → $A\cdot\bar{B}$, $\bar{A}B$ → OR → XNOR.



— Activation fn decides whether a neuron should be activated or not by calculating weighted sum & further adding bias with it.

Sigmoid actn fn.:
- It is an actn fn of form, $f(x) = \frac{1}{1+e^{-x}}$.
- Its range is blw 0 & 1.
- Its easy to understand & apply.

Disadvantages.
- Vanishing gradient problem.
- Secondly its o/p is not zero centered So it makes the gradient updates go too far in different direct^n. $0 < o/p < 1$ & it makes optimization harder.
- Sigmoids saturate & kill gradients.
- Sigmoids have slow convergence.
- Computationally expensive.

Hyperbolic Tangent fn - Tanh.
Its mathematical formula is
$$f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$$
- Now its o/p is zero centered because its range in blw -1 to 1 i.e $-1 < o/p < 1$. Hence optimization is easier in this method.
- Hence in practice it is always preferred over sigmoid fn.

Drawback: Like sigmoid fn only.

ReLU - Rectified Linear Units.
- It has become very popular in the past couple of years.
- It was recently proved that it had 6 time improvement in convergence from Tanh fn.
- Its just $R(x) = \max(0, x)$ i.e if $x<0$, $R(x)=0$ & if $x>=0$, $R(x)=x$.

- Hence as seeing the mathematical form of this fn, we can see that it is very simple & efficient.
- It avoids & rectifies vanishing gradient problem.
- Almost all deep learning models use ReLu nowadays.
- Drawback:
  Its limitation is that it should only be used within hidden layers of a neural nlw model.
- Hence for o/p layers we should use a Softmax fn for a classification problem to compute the probabilities for the classes.
- Another problem with ReLU is that some gradients can be fragile during training & can die, ie it can cause a weight update which will make it never activate on any data point again.
- Simply saying that ReLu could result in dead neurons.

- The Dying ReLU Problem: when inputs approach zero, or are -ve, the gradient of the fn becomes zero, the nlw cannot perform backpropagation & cannot learn.
- To fix this problem another modification was introduced called Leaky ReLU to fix the problem of dying Neurons. It introduces a small slope to keep the updates alive.

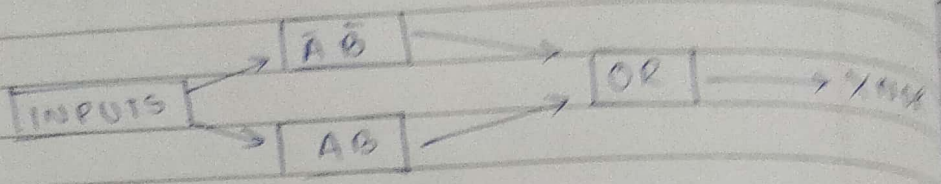
Drawback.

In Leaky ReLU results are not consistent & does not provide consistent predictions for -ve i/p values.

Tue
03/03/20

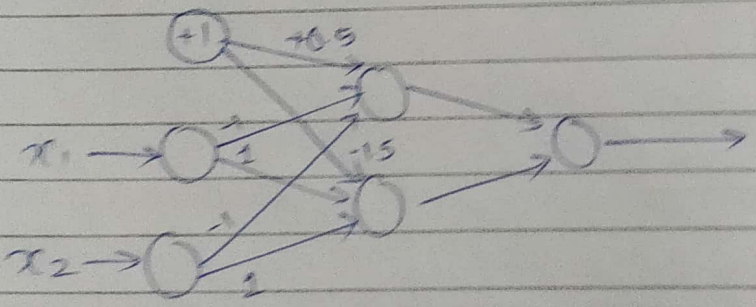
XNOR Gate

X_1 XNOR X_2 = NOT (X_1 XOR X_2)
= NOT $[(A+B) \cdot (A'+B')]$
= $(A+B)' + (A+B')$
= $A' \cdot B' + A \cdot B$



		AND	XNOR
1		0	1
0	OR	0	0
0		0	0
0		1	1

this can be 0 or 1 but if 1 it will be XNOR .·. its 0



Conclusion.

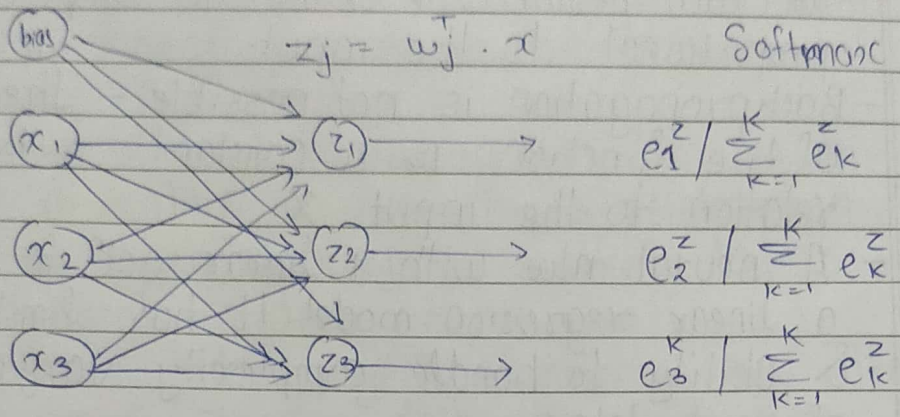
Sigmoid & tanh should not be used due to the vanishing gradient problem which causes a lot of problem, degrades the accuracy & performance of deep neural n/w model.

- Nowadays we use ReLU which should only be applied to hidden layers.
- And if your model suffers from dead neuron during training, we should use leaky ReLU fn.

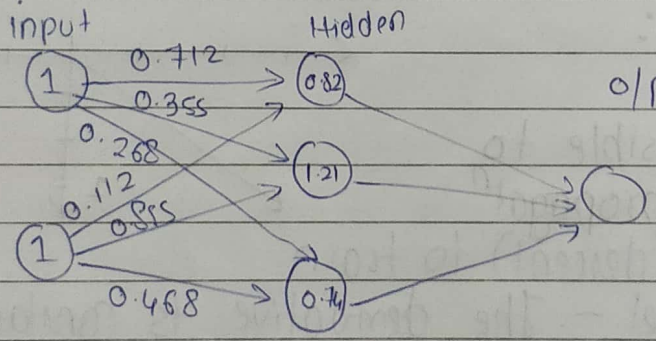
Other activation functions.

1. Softmax : $\frac{e^{a_i}}{\sum_k e^{a_k}}$

Multiclass Classificatn with NN & Softmax fn



What would be the Softmax activated value of this neuron ?



↑

exp of these [2.27, 3.35, 2.096].

Divide by sum (Softmax) we get [0.29, 0.43, 0.27]

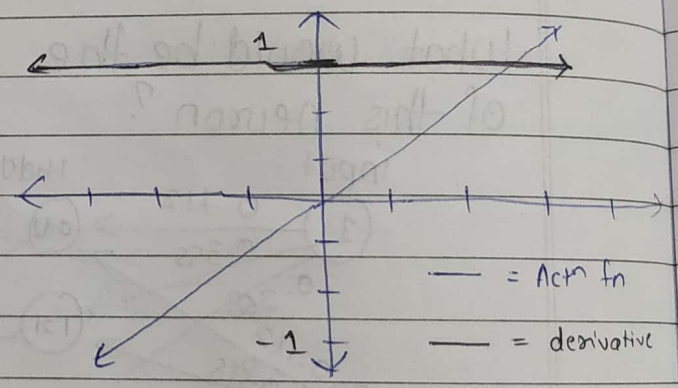
0.65
0.242
0.098

eg. LOGITS scores.

$$y = \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix} \Rightarrow \text{Softmax} \Rightarrow \begin{matrix} 0.65 \approx 0.7 \\ 0.242 \approx 0.2 \\ 0.098 \approx 0.1. \end{matrix}$$

2. Linear Activat^n fn.
- Takes the form : Act^n() = cx.
- Tanh, Sigmoid are <u>not</u> linear

Linear or Identity Act^n fn.
- It takes the input, multiplied by the weights for each neuron, & creates an output signal proportional to the input
- Backpropagation is not possible - The derivative of the function is a constant, & has no relation to the input, X.
- A neural nlw with a linear act^n fn is simply a linear regression model. It has limited power & ability to handle complexity varying parameters of input data.



Problems :

1. Not possible to use backpropagat^n (gradient descent) to train the model -. The derivative is constant & has no relat^n to input X. So its not possible to go back & understand which weights can provide better predict^n.

2. All layers of neural nlws collapse into one. With linear act^n fn, no matter how many layers in the NN, the last layer will be a linear fn of the first layer. (because linear combinat^n of linear fn is still linear fn). Turns NN into just one layer.
∴ Not much used.

3. Non-Linear Actⁿ fn.

- Todays NN models use non-linear actⁿ fn.
- They allow the model to create complex mappings blw the nlw's inputs & outputs, which are essential for learning & modelling complex data such as images, video, audio & data sets which are non-linear or have high dimensionality.
- Non-linear fn address the problems of a linear actⁿ fn.
- They allow backpropagation because they have a derivative fn which is related to the inputs.
- They allow "stacking" of multiple layers of neurons to create a deep neural nlw.
- Multiple hidden layers of neurons are needed to learn complex data sets with high level of accuracy.

Wed.
04/03/20

Chp 2: Shallow Neural Networks.

Batch Normalization.

One of the important issues in NN is that the training of the network takes a long time for effectively deep networks.

By backpropagation, the NN learn how much error it did & correct themselves, i.e correct their "weights" & "biases"

By this, NN learn the problem to produce outputs for given inputs.

Back propagation involves computing gradients for each layer & propagating it backward, hence the name.

But during backpropagation of errors to the weights & biases, we'll face a undesired property of Internal Covariate Shift. This makes the nlw too long to train

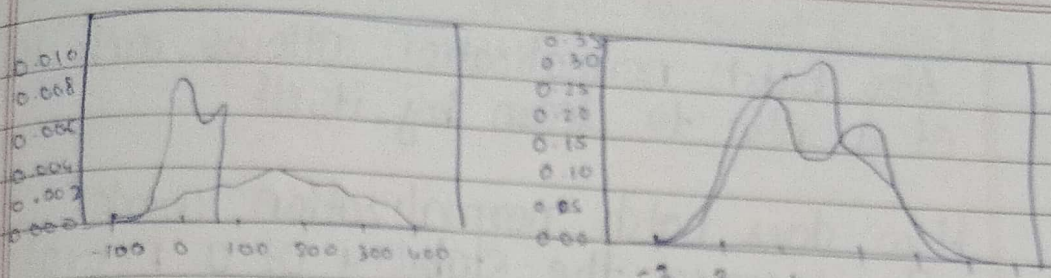
Internal Covariate Shift

During training, each layer is trying to correct itself for error made up during the forward propagation. But every single layer acts separately, trying to correct itself for the error made up.

More specifically, due to changes in weights of previous layers, the distribution of input values for current layer changes, forcing it to learn from new "input distribution".

Normalization.

In a dataset, all the features may not be in same range. It takes a lot of time to train for these kind of datasets.



Raw & Normalized data.

We normalize the input layer by adjusting & scaling the features.

eg. When we have features from 0 to 1 & some from 1 to 1000, we should normalize them to speed up learning.

Same thing also for the values in the hidden layers, that are changing all the time, & get 10 times more improvement in training speed.

Batch normalization reduces the amount by what the hidden unit values shift around.

To know shift around, lets have a deep network on cat detection. We train our data on only black cats' images.

If we now try to apply this network to data with colored cats, it is obvious, we're not going to do well.

The training set & the prediction set are both cats images but they differen a little bit.

In other words, if an algorithm learned some X to y mapping.

If the distribution of x changes, then we might need to ~~train~~ retrain the learning algo by trying to align the distribution of x with the distribution of y.

- Also batch normalization allows each layer of a nlw to learn by itself.

How does batch normalization work?

To increase the stability of a neural nlw batch normalization normalizes the output of a previous actn layer by subtracting the batch mean & dividing by the batch standard deviation.

After this scale of actn olps, the weights in the next layer may be optimal. SGD undo this normalization if its a way for it to maximize the loss fn.

If so, batch normalization adds two trainable parameters to each layer, so the normalized olp is multiplied by a standard deviation & add a mean

In other words, batch normalization lets SGD do the normalization by changing only these two weights for each actn, instead of losing stability of the nlw by changing all the weights.