Complete Road Map to Prepare Deep learning

- Py Torch
- Keras
- Tensor Flow

Deployment Projects → HuggingFace, Ktrain, Transformers, BERT

Encoders And Decoders, Attention Models

Word Embeddings, Word 2vec

LSTM, GRU, Bidirectional LSTM

RNN

Deployment Projects → Transfer Learning Eg: Vgg 16, Alexnet

RCNN, Masked RCNN, SSD, YOLO

Object Detection

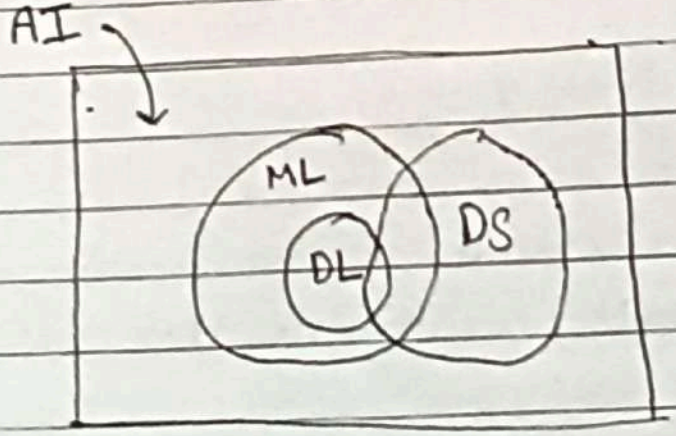
Artificial Neural Network

Convolutional Neural Network

Introduction to Neural Network, Loss Function, Optimizers, Gradient Descent, SGD, Adograd, RMSprop, Adam

Deep learning

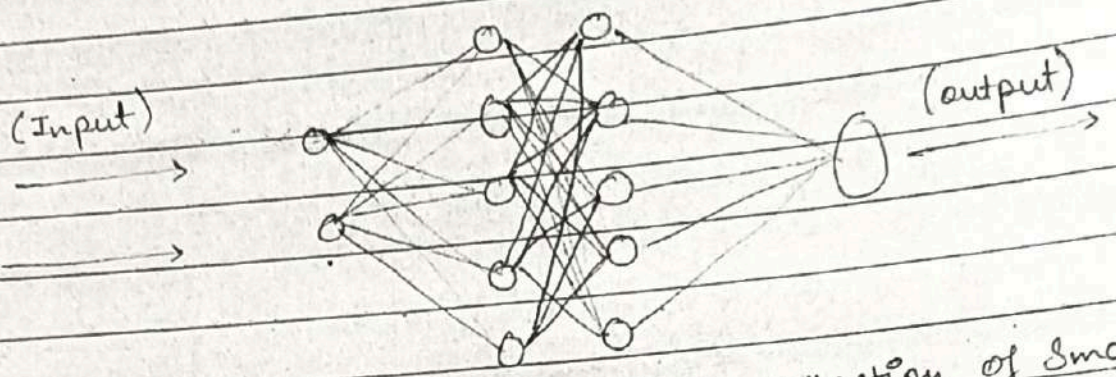
- ANN → Artificial Neural Network (Tabular kind of dataset)
- CNN → Convolutional neural Network (Image, Videos)
- Image classification → CNN, Transfer Learning.
- object detection → RCNN, FAST RCNN, FACTSR, RCNN, SSD YOLO.
- object Segmentation → Self driving car
- RNN → Recurrent Neural Network → Text, Time Saver, Data.



AI → Artificial intelligence
ML → Machine learning
DL → Deep learning
DS → Data Science.

A Simple Neuron

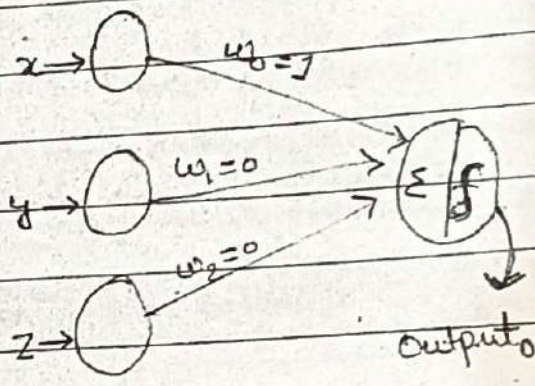
Deep learning is a specialized subset of Machine Learning that uses layered neural networks to stimulate human decision-making.



A Neural Networks in AI is a collection of small computing units called neurons that take incoming data and learn to make decisions over time.

Perceptron

A perceptron is a neural network unit (an artificial neuron) that does certain computation to detect features or business intelligence in the input data

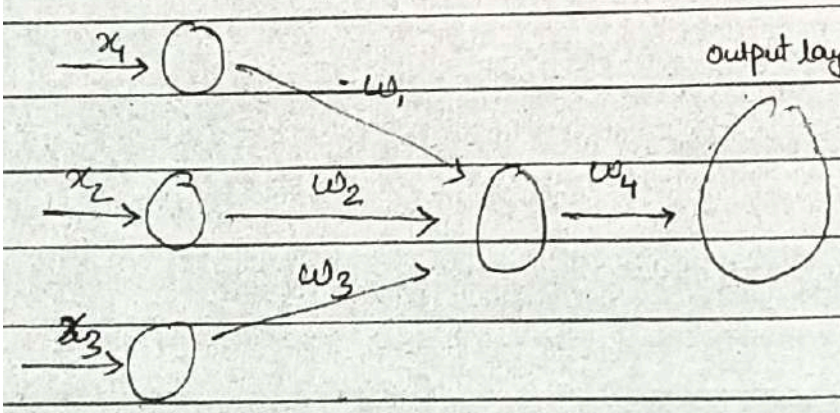


Step 1 $y = w_1x_1 + w_2x_2 + w_3x_3 + \text{bias}$

$Z = \text{Act}(y)$

Neuron

Input layer



Step 1 Step 2

$\sum_{i=1}^{n} W_i X_i$ $\text{Act}\left(\sum_{i=1}^{n} w_i x_i\right)$

- The function output is not centered on 0, which will reduce the efficiency of weight update
- The Sigmoid function performs exponential operations, which is slower for computers.

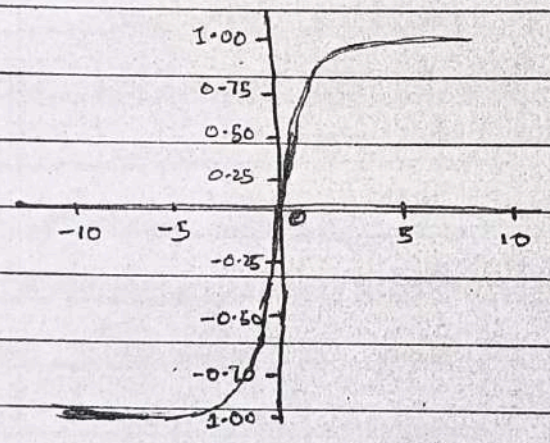
⇒ Advantages of Sigmoid Function

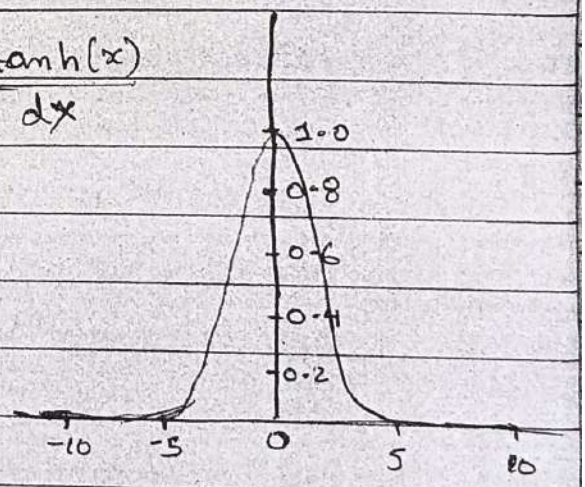
1) Smooth gradient, preventing "jumps" in output values.
2) Output values bound b/w 0 and 1, normalizing the output of each neuron
3) clear predictions, i.e very close to 1 or 0

⇒ Sigmoid has three major disadvantages.

1) Prone to gradient vanishing.
2) Function output is not zero-centered
3) Power operations are relatively time consuming.

(2) Tanh function





→ Tanh is a hyperbolic tangent function. The curves of tanh function and Sigmoid function are relatively similar.

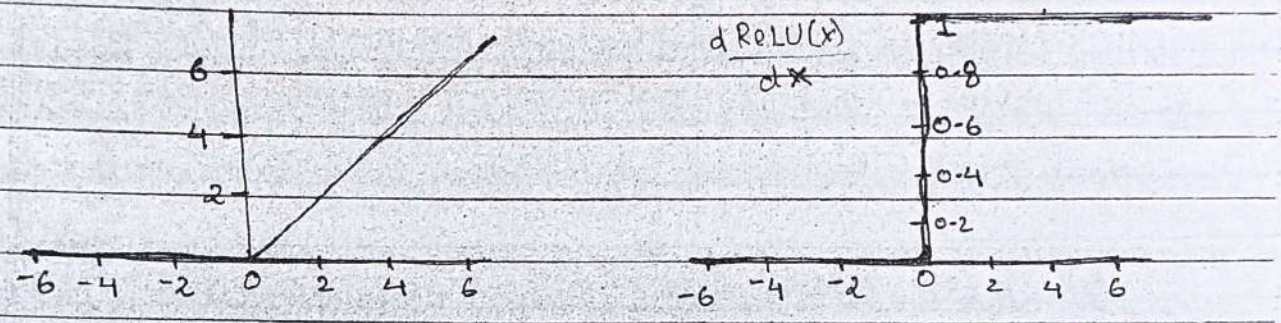
→ when the input is large or small, the output is almost smooth & the gradient is small, which is not conducive to weight update.

→ The difference is the output interval. The output interval of tanh is 1

→ The whole function is 0-centric, which is better than Sigmoid

→ In general binary classification problems, the tanh function is used for the hidden layer and the Sigmod function is used for the output layer. However, these are not static, and the specific activation function to be used must be analyzed according to the specific problem, or it depends on debugging.

③ ReLU function [max(0,x)]



→ whenever your (x) ~~value~~ > 0, then the output will be that particular x. If x < 0, then the output will be 0.

→ This is mostly use in hidden layers because it solves the vanishing Gradient problem.

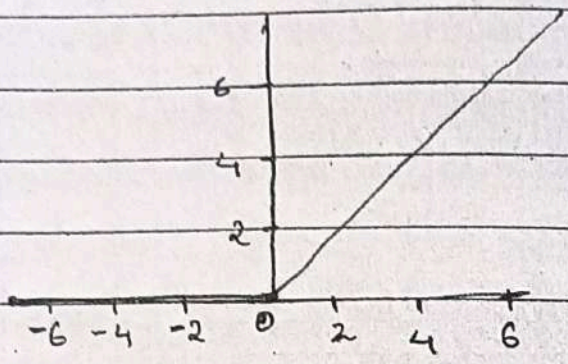
→ Compared to the Sigmod function and the tanh function, it has the following advantages:

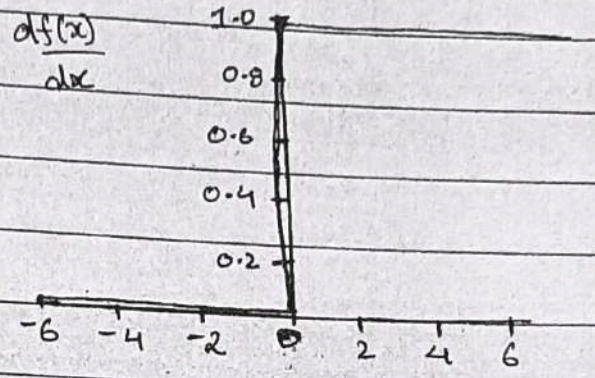
- when the input is positive, there is no gradient saturation Problem.
- The calculation speed is much faster.

→ Disadvantages

- when the input is -ve, ReLU is completely inactive.
- The output of the ReLU function is either 0 or a positive number, which means that the ReLU function is not a 0-centric function.

④ Leaky ReLU function $f(x) = \max(0.01x, x)$



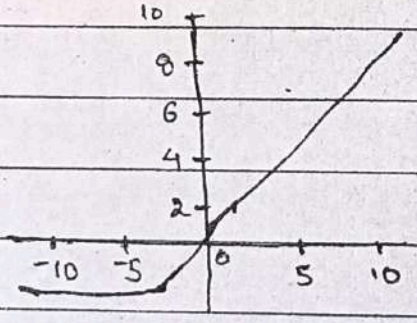


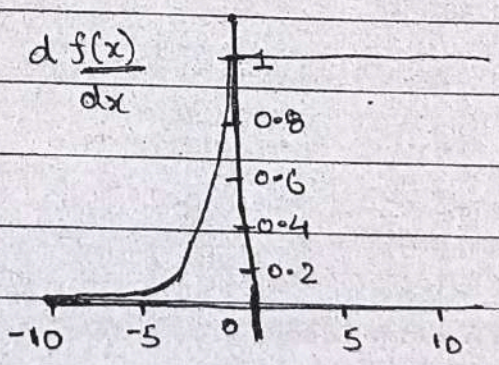
→ Leaky ReLU has all the advantages of ReLU, Plus there will be no problems with Dead ReLU.

→ But in actual operation, it has not been fully proved that Leaky ReLU is always better than ReLU.

→ People proposed to set the first half of ReLU 0.01x instead of 0. Another intuitive idea is a parameter-based method, parametric ReLU : $f(x) = \max(\text{alpha}\, x, x)$, which alpha can be learned from back propagation.

⑤ ELU (Exponential Linear Units) function

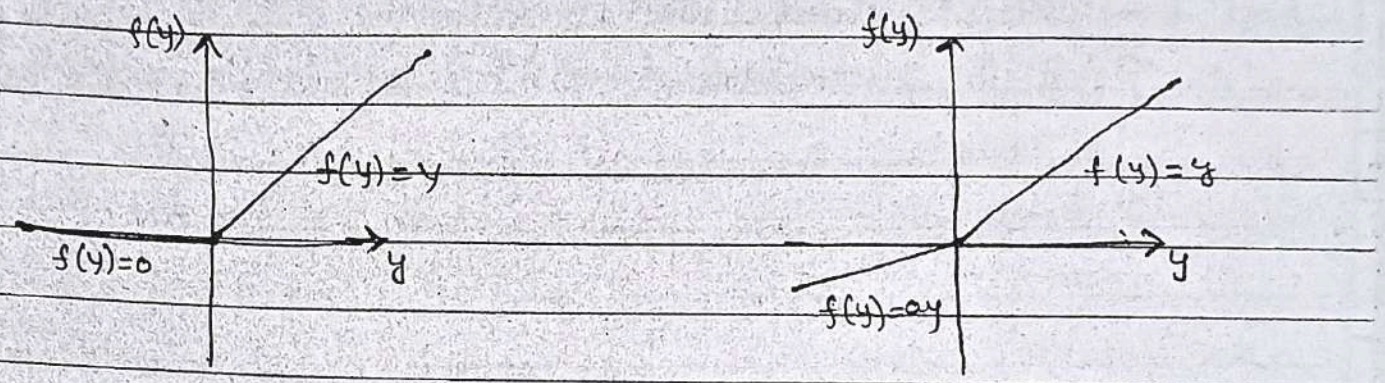




- No Dead ReLU issues
- The mean of the output is close to 0, zero-centered.

→ One small problem is that it is slightly more computationally intensive. Similar to Leaky ReLU, although theoretically better than ReLU, there is currently no good evidence in practice that ELU is always better than ReLU.

⑤ PReLU (Parametric ReLU)



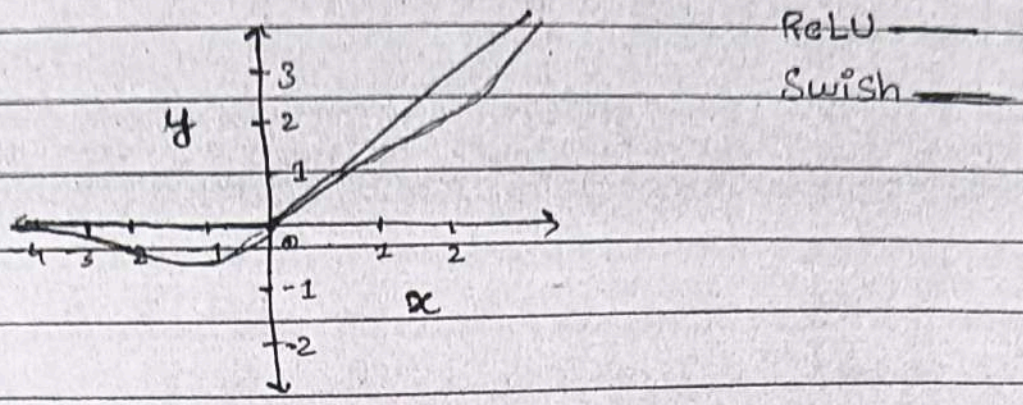
→ PReLU is also an improved version of ReLU.
→ In the negative region, PReLU has a small slope, which can also avoid the problem of ReLU death.

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}$$

above, y_i is any input on the i^{th} channel and a_i is the negative slope which is a learnable parameter.

- if $a_i = 0$, f becomes ReLU
- if $a_i > 0$, f becomes leaky ReLU
- if a_i is a learnable parameter, f becomes PReLU.

⑦ Swish (A Self-Gated) Function



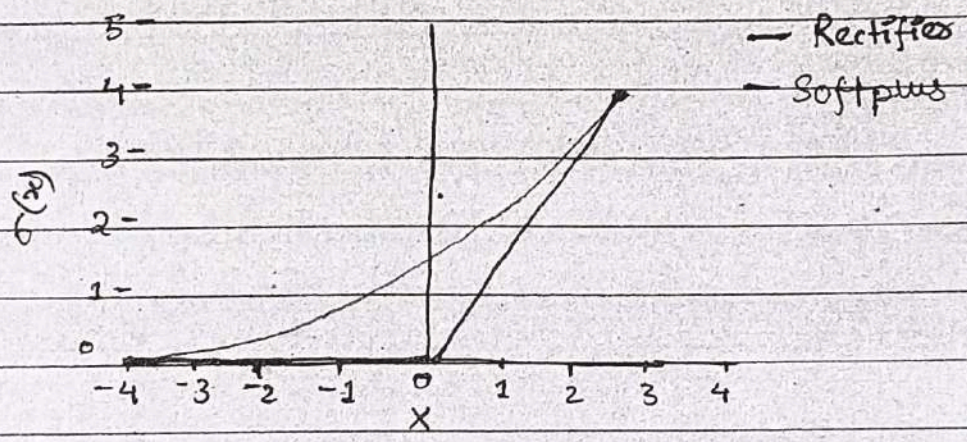
→ $y = x * \text{Sigmoid}(x)$ ← formula, and this is called Self-gated

→ This is used in LSTMs

→ we use this type of activation function when we have our neural network > 40 layers

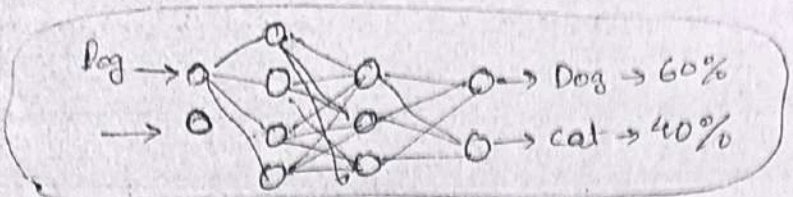
→ It is zero centric and it also solves the ded-relu

⑧ Softplus



→ we know that derivative of zero is not possible in real world Scenario, most of the time, but we may get Sometimes when we Solving, So if you get zero in that Specific way, using a relu will not Serve the purpose. because we dont know the derivative of 0. So we can use Softplus instead.

→ $f(x) = \ln(1 + \exp x)$



⑤ Softmax → we use this when we have more than 2 outputs.

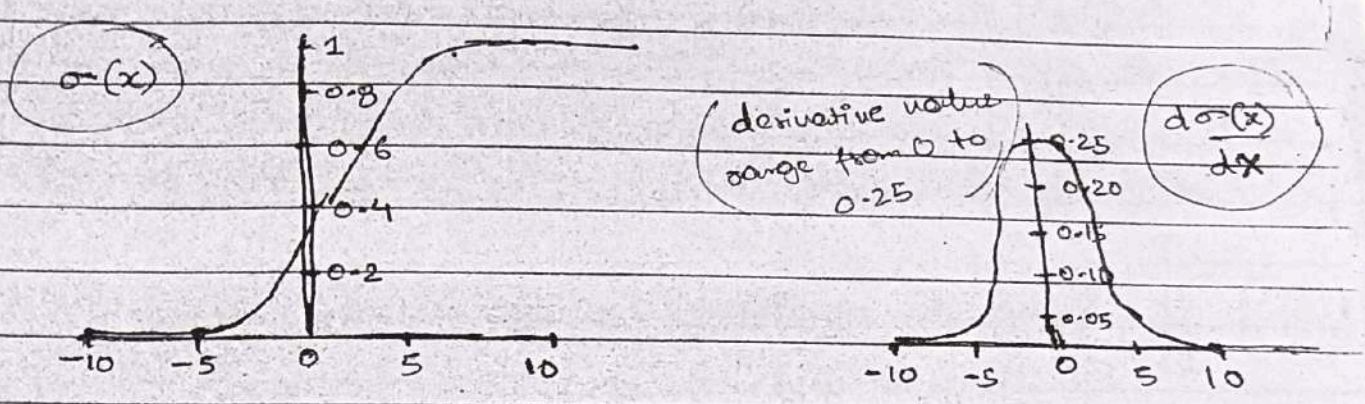
$$\rightarrow \boxed{S(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{xk}}}, \quad j = 1, 2, \ldots, K$$

Note Generally Speaking, these activation functions have their own advantages and disadvantages, There is no Statement that indicates which ones are not working, and which activation functions are good. All the good and bad must be obtained by experiments.

Name	Plot	Equation
Identity		$f(x) = x$
Binary Step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a Sigmoid / Soft Step)		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{(e^{x} - e^{-z})}{(e^{x} + e^{-z})}$
Rectified Linear unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Leaky (ReLU)		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Softmax		for $i = 1, \ldots, J$ $f_i(\vec{x}) = \frac{e^{xi}}{\sum_{j=1}^{J} e^{xj}}$
Maxout		$f(\vec{x}) = \max x_i$

☆ Commonly used activation functions

① Sigmod function



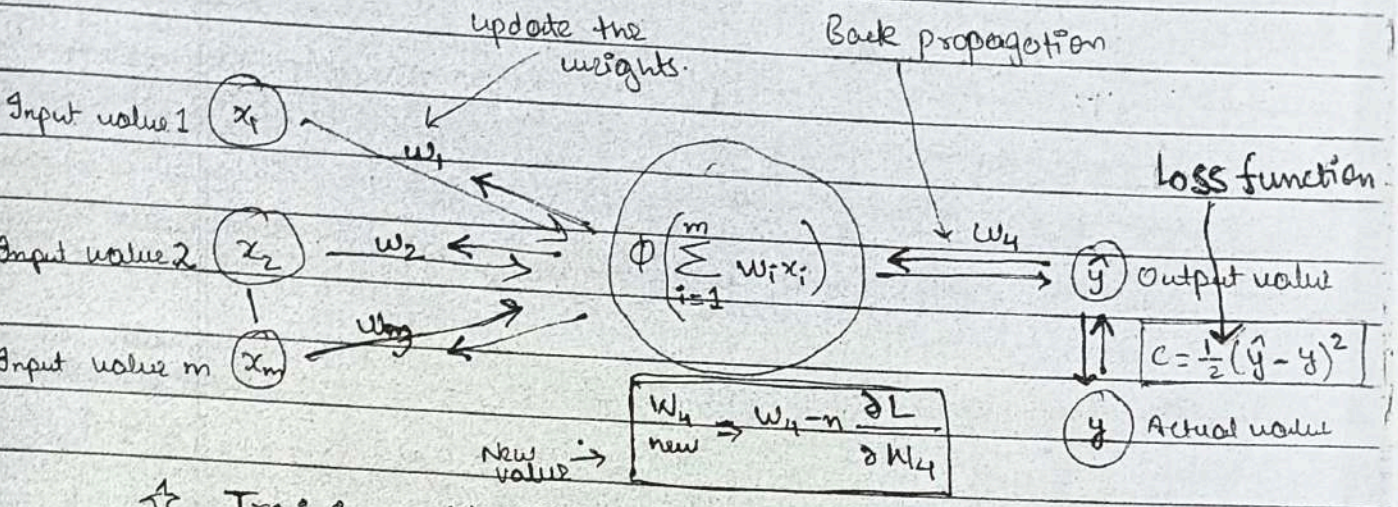
⇒ The Sigmoid function is the most frequently used activation function in the beginning of deep learning. It is a smoothing function that is easy to derive.

⇒ The function itself have certain defects :-

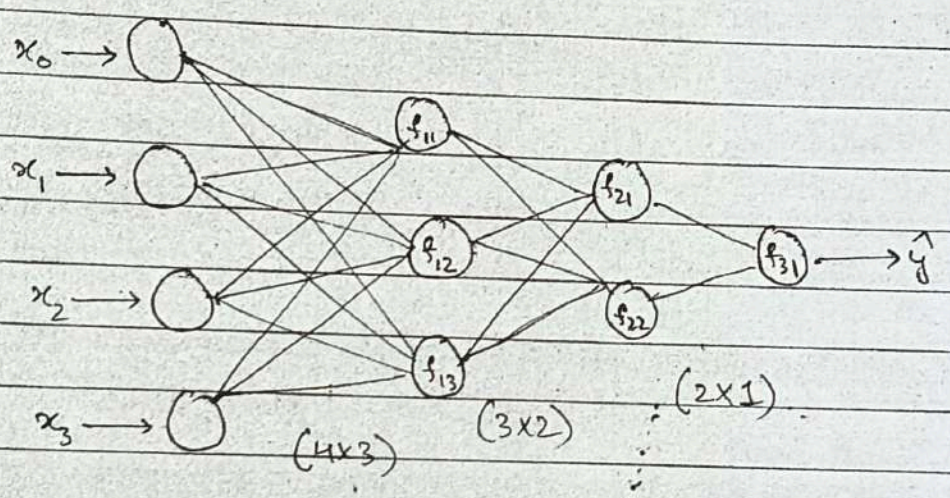
- when the input is slightly away from the co-ordinate origin, the gradient of the function becomes very small, almost zero. In the process of neural network backpropa-gation, we all use the chain rule of differential to calculate the differential of each weight (W). when the backpropagation passes through the Sigmod function. the differential on this chain is very small. Moreover, it may pass through many Sigmod functions, which will eventually cause the weight (W) to have little effect on the loss function. which is not conductive to the optimization of the weight. This problem is called gradient Saturation or gradient dispersion.

In case of Back propagation, our $\hat{y}$ and y values are different so we take Back propagation. here we update the w_4 value first the w_1, w_2 & w_3. Once we get the new value, again the front propagation will go ahead, this process will keep on happening until the Loss value is completely reduced i.e $\boxed{\hat{y} = y}$.

☆ Back propagation



☆ Training Multilayer Neural Network & Gradient Descent



→ for comparing $\hat{y}$ and y we use loss function $(L = (y - \hat{y})^2)$

→ here we are trying to see the difference b/w y and $\hat{y}$ and our main aim is to reduce that loss value. then only we'll be getting similar results for $\hat{y}$ & y

→ If we want to reduce the loss function, we basically use an optimizer. such as gradient descent.

→ Gradient descent work is to update ~~the~~ all the weights in such a way the should be able to get a $\hat{y}$ similar to y

→ This updating the weights to each layer is known as

for -ve slope → $W_{new} = W_{old} + \eta \frac{\partial L}{\partial w}$

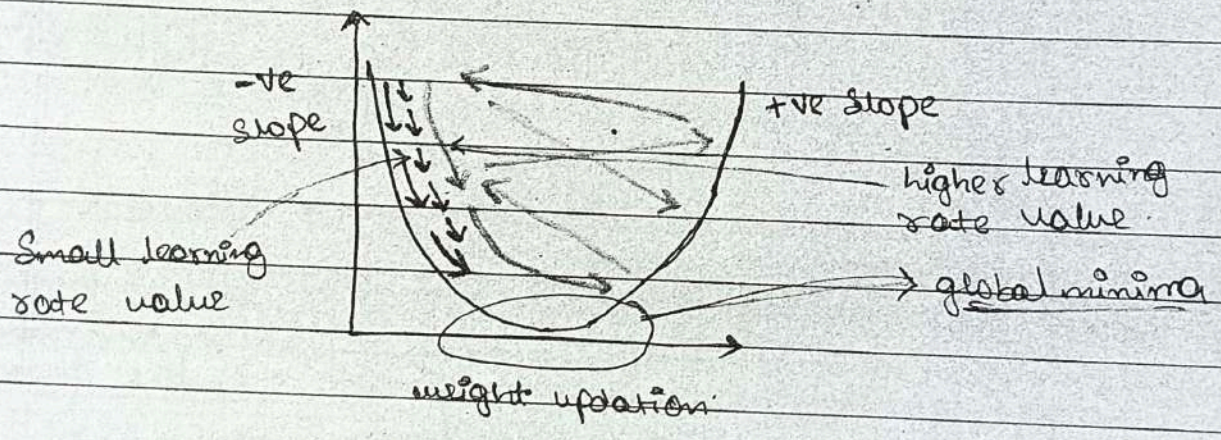
formula for updating the weights

for +ve slope → $W_{new} = W_{old} - \eta \frac{\partial L}{\partial w}$

→ W_{new} = new weight
→ W_{old} = old weight
→ η = Learning rate
→ ∂L → derivative of the loss
→ ∂w → derivative of that specific weight

Q) why do we need to find derivative?
Ans) derivative is nothing but finding the slope

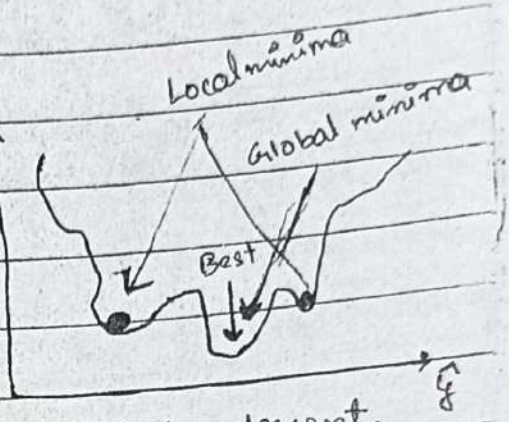
* The main aim of gradient Descent is to reach in "global minima". Unless and until we dont reach to global minima, the updation of the weights will happen with the help of back propagation.
* If the loss value is decreasing, that will also indicate that we are reaching towards global minima.



→ If we take a Small learning rate value, then it will take more time to reach global minima point.
→ If we take a high learning rate value, then it will take less time to reach, but most of the time it will start jumping here & there b/w -ve slope and +ve slope. and it may never reach the global minima point.

→ So that's why we should select a "learning value" in such a way that it should be a value that is suitable with respect to gradient descent.

☆ Local Minima and Global Minima

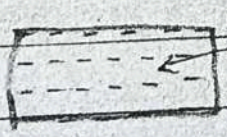
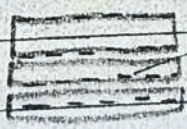
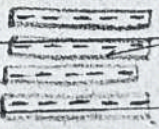


→ The point at which a function takes the minimum value is called global minima. However, when the goal is to minimize the function and solved using optimization algorithms such as gradient descent, it may so happen that function may appear to have a minimum value at different points.

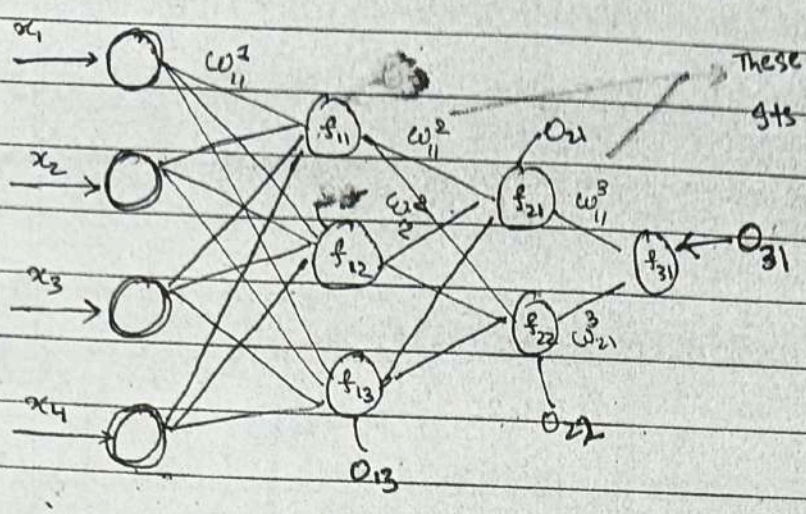
→ Those several points which appear to be minima but are not the point where the function actually takes the minimum values are called local minima.

→ Machine learning algorithms such as gradient descent algorithms may get stuck in local minima during the training of the models.

→ In the order to avoid this we use stochastic gradient descent.

Batch - GD	minibatch - GD	Stochastic - GD
		
→ May end up in local minima → Slow → more deterministic	→ If stuck in local minima, some noisy steps can lead the way out of them.	→ Local minima problem can be resolved → Fast → Less deterministic

☆ Chain Rule in Back Propagation



* If we want to update w_{11}^3 (1st layer from back side)

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

$$\rightarrow w_{11}^{3\,new} = w_{11}^{3}{}_{old} - \eta \frac{\partial L}{\partial w_{11}^3}$$

we can also write as

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

where ∂O_{31} → is the derivative of the output of the particular layer

* If we want to update w_{21}^3

$$w_{21}^{3\,new} \rightarrow w_{21}^{3}{}_{old} - \eta \frac{\partial L}{\partial w_{21}^3}$$

can be written as

$$\frac{\partial L}{\partial w_{21}^3} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{21}^3}$$

* If we want to find the derivative of w_{11}^2

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

→ $$\frac{\partial L}{\partial w_{11}^2} = \left[\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{21}} \times \frac{\partial o_{21}}{\partial w_{11}^2}\right] + \left[\frac{\partial L}{\partial o_{31}} \times \frac{\partial o_{31}}{\partial o_{22}} \times \frac{\partial o_{22}}{\partial w_{12}^2}\right]$$

☆ **Vanishing Gradient Problem**

→ When the elements of the gradient become exponentially small so that the update of the parameters with the gradient becomes almost insignificant.

→ As the number of layers increases, the derivative value decreases and at one situation you will see that the value of 'old weight' and 'new weight' are matching. [Predicted output ($\hat{y}$) and given output (y) are different. they should be same) I/c

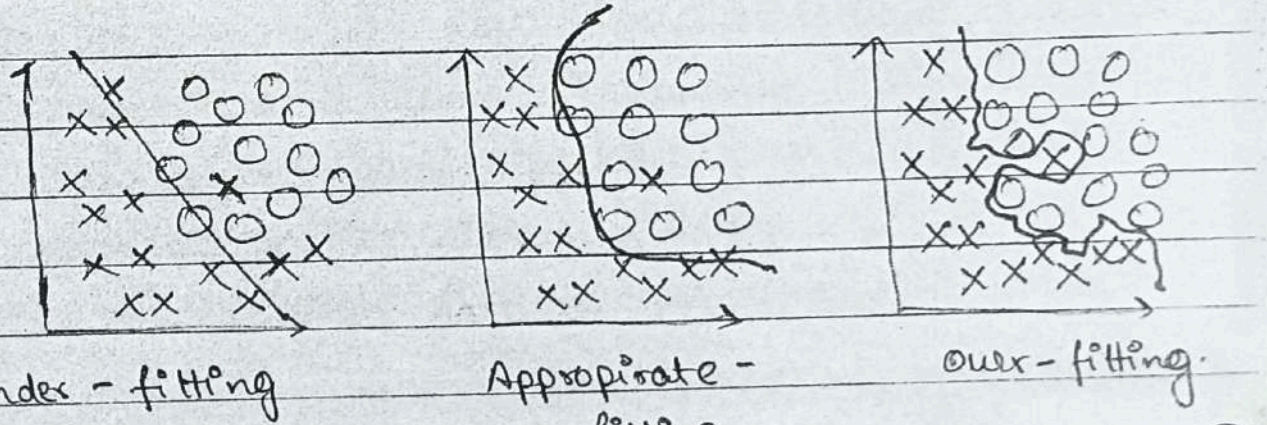
→ So this the problem we face in Sigmoid & tanh

→ This is the reason why Sigmoid is bad and its shouldn't be use in each and every layer.

→ This is where ~~Relu~~ ReLU comes into existance.

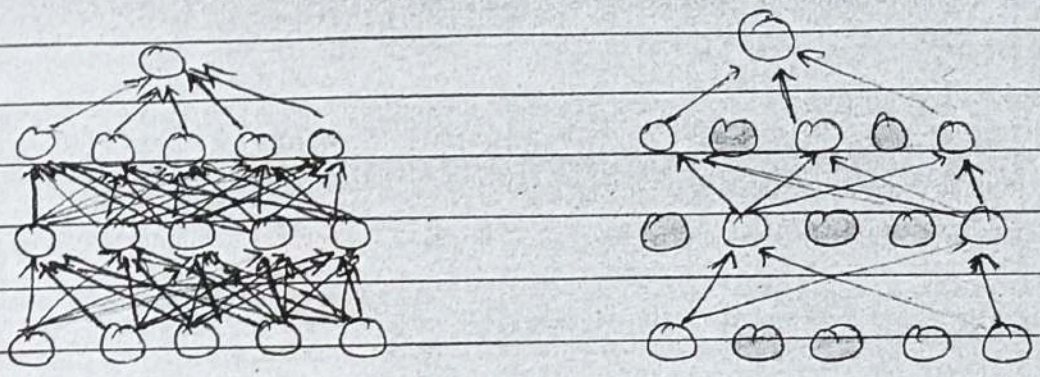
→ Sigmoid function ranges b/w 0 to 1 & Derivative of Sigmoid functio ranges b/w 0 to 0.25

☆ **Dropout Regularization**



under-fitting | Appropirate-fitting | over-fitting.

→ 'Dropout' in machine learning refers to the process of randomly ignoring certain nodes in a layer during training.



a) Standard Neural Network

b) After applying dropout.

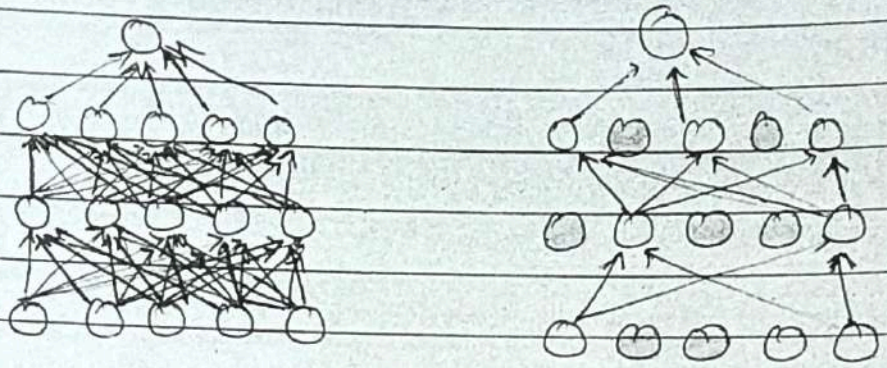
→ Dropout is used as a regularization technique — it prevents overfitting by ensuring that no units are codependent

Q) why will dropout help with overfitting?
→ It cant rely on one input as it might be dropped out at random
→ Neurons will not learn redundant details of inputs.

☆ Dead neuron
→ A neuron that cannot be activated, i.e, contributes nothing to feature extraction or classification. In practise, when you see a feature map is almost 0 no matter what kind of input you feed, there is very likely a dead neuron.
→ The problem with ReLU is that Sigmoid function always converts it into the range 0 to 1 which creates dead neuron.

→ Activation function → Activation functions are mathematical equations that determine the output of a neural network.

→ 'Dropout' in machine learning refers to the process of randomly ignoring certain nodes in a layer during training.



a) Standard Neural Network

b) After applying dropout.

→ Dropout is used as a regularization technique — it prevents overfitting by ensuring that no units are codependent

Q) why will dropout help with overfitting?

→ It cant rely on one input as it might be dropped out at random

→ Neurons will not learn redundant details of inputs.

☆ Dead neuron

→ A neuron that cannot be activated, i.e, contributes nothing to feature extraction or classification. In practise, when you see a feature map is almost 0 no matter what kind of input you feed, there is very likely a dead neuron.

→ The problem with ReLU is that Sigmoid function always converts it into the range 0 to 1 which creates dead neuron.

→ Thats why we use Leaky ReLU instead. In order to solve the Dead ReLU Problem, people proposed to set the final half of ReLU (0.01) ~~instead~~ instead of (0).

GD, SGD, Minibatch SGD → These all optimizers were helping us to reduce the loss function with respect to the weights

☆ Weight initialization

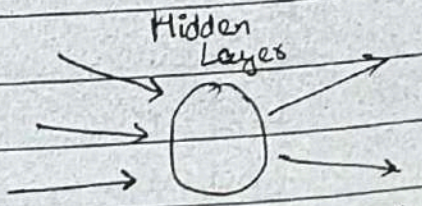
→ weights should be small

→ weights should not be same

→ weights should have good variance

☆ There are three weight initialization techniques :
(we dont have to remember this, just understant)

① Uniform Distribution



$$\rightarrow W_{ij} \sim \text{Uniform}\left[\frac{-1}{\sqrt{fan\_in}}, \frac{1}{\sqrt{fan\_in}}\right]$$

② Xavier / Gorat → This works good with Sigmoid AcV

→ Xavier Normal

$$w_{ij} \sim N(0, \sigma)$$

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

→ Xavier Uniform

$$w_{ij} \sim U\left[\frac{-\sqrt{6}}{\sqrt{fan\_in + fan\_out}}\right]$$

④ He init

$$w_{ij} \approx U\left[-\frac{\sqrt{6}}{\sqrt{fan\_in}}, \frac{\sqrt{6}}{\sqrt{fan\_in}}\right]$$

⇒ He uniform

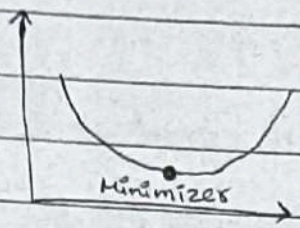
$$w_{ij} \approx N(0, \sigma)$$

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

⇒ He normal

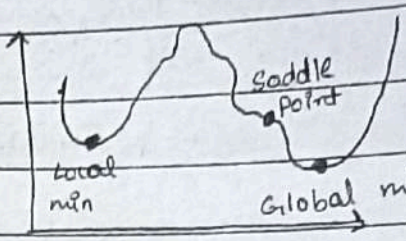
If all the points present in the same curve

☆ Difference b/w convex & Non-convex function



Convex function

→ we usually see this in ML Problems

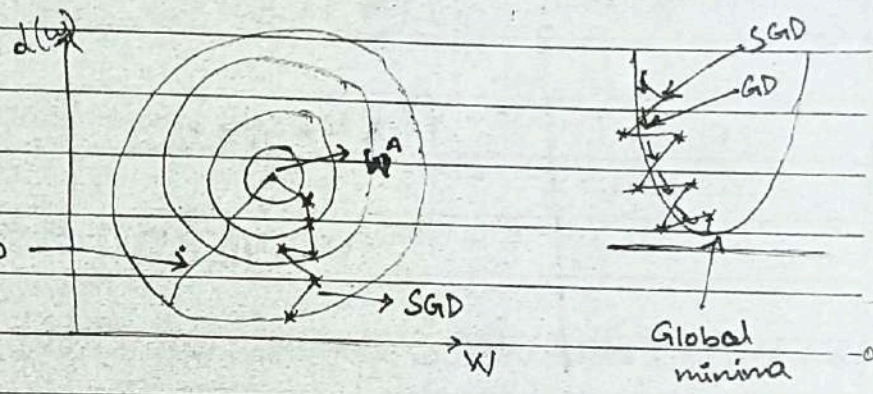


Non-convex function.

→ we find this in DL problems

SGD with momentum

This shows how your weights are getting updated to best minima weight



b
$\sqrt{6}$
$\sqrt{fan\_in + fan\_out}$

⇒ we can see that we ~~have~~ always have a noisy data incase of SGD, thus for the convergence it will take more time. So everytime we use SGD, we should try to remove that noise from the data. In the oder to remove that noise, there is an concept called 'Exponential moving average'

$(b_1, b_2 \cdots b_n)$

⇒ Suppose ~~have~~ have data points at different time intervals $(t_1, t_2 \cdots t_n)$

⇒

t_1	$\rightarrow b_1$
t_2	$\rightarrow b_2$
⋮	⋮

So, the Exponential moving average formula

⇒ $V = \gamma \times V_1 + b_2$

V = variable

γ = gama

Need not have to learn,

just remember

t_1	t_2	- - -	t_n
b_1	b_2	- - -	b_n

finding the Exponential moving average :- ($\gamma = 0.5$)

→ $V_{t_1} = b_1$ — (1)

→ $V_{t_2} = \gamma \times V_1 + b_2$ — (2)

$= 0.5 \times b_1 + b_2$ [from 1]

→ $V_{t_3} = \gamma + V_2 + b_3$

$= \gamma (\gamma \times V_1 + b_2) + b_3$ [from (2)]

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

$= 0.25 b_1 + 0.5 b_2 + 1 \times b_3$

This is how moving average is calculated b/w b_1, b_2 & b_3

⇒ Now how do we apply the same formula in our G.D

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

This is momentum

we write this instead in this case

→ $w_{new} = w_{old} - \left[\gamma v_{t-1} + \eta \frac{\partial L}{\partial w_{old}}\right]$ ($\because \gamma$ = gama)

→ $\therefore V_{t-1} = 1 \times \left[\frac{\partial L}{\partial w_{old}}\right]_t + \gamma \left[\frac{\partial L}{\partial w_{old}}\right]_{t-1} + \gamma^2 \left[\frac{\partial L}{\partial w_{old}}\right]_{t-2}$

∵ this is how we should be able to reduce the noise

Implimentation details

weight

bias

Initially $V_{dw} = 0$ $V_{db} = 0$

On iteration t inside epoch, compute dw, db on the corrent mini batch.

$$\rightarrow V_{dw_t} = \beta V_{dw_{t-1}} + (1-\beta) \frac{\partial L}{\partial w_{t-1}}$$

$$\rightarrow V_{db_t} = \beta V_{db_{t-1}} + (1-\beta) \frac{\partial h}{\partial b_{t-1}}$$

$$\Rightarrow \boxed{\begin{aligned} w &= w - \eta V_{dw} \\ b &= b - \eta V_{db_t} \end{aligned}}$$

← by this we will achieve smoothening of convergence.

GRADIENT DESCENT WITH MOMENTUM

Implementation details

on iteration t in minibatch

compute $\frac{\partial h}{\partial w}$, $\frac{\partial L}{\partial w}$ on current minibatch

→ $S_{dw} = \beta S_{dw} + (1-\beta)\left(\frac{\partial h}{\partial w}\right)^2$

→ $S_{db} = \beta S_{db} + (1-\beta)\left(\frac{\partial h}{\partial b}\right)^2$

$$w_t = w_{t-1} - \eta' \frac{\partial h}{\partial w_{t-1}}$$

$$b_t = b_{t-1} - \eta' \frac{\partial h}{\partial b_{t-1}}$$

Algorithm for weight updation

BoW → [illegible] NLP

☆ AdaGrad optimizer → Adaptive Gradient Descent

→ It is an algorithm for gradient-based optimization
→ In AdaGrad optimizer, there is no momentum concept So, it is much simpler compared to SGD with momentum.
→ The idea behind this is to use different learning rates for each parameter based on iteration.
→ The reason behind the need for different learning rates is that the learning rate for Sparse features parameters needs to be higher compared to dense features parameter because the frequency of occurrence of Sparse features is lower.

[Sparse → (0,1) (most of the values are 0)] , [dense → most of the values are non-zero]

So the main concept behind AdaGrad optimizer is that, "can we use different learning rates with respect to number of feature, different weights and different itration.

$$\Rightarrow w_t = w_{t-1} - \eta \times \frac{\partial h}{\partial w_{t-1}}$$

$$\Rightarrow w_t = w_{t-1} - \eta_t^1 \times \frac{\partial h}{\partial w_{t-1}}$$

Initial learning rate

$$\Rightarrow \eta_t^1 = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

∵ ε → Small +ve number

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

∵ $\alpha_t \uparrow \Leftrightarrow \eta_t \downarrow$

☆ Advantages of using Ada Grad

→ It eliminates the need to manually tune the learn rate

→ Convergence is faster and more reliable - than SimSGD when the scaling of the weights is unequal

→ It is not very sensitive to the size of the master step

☆ disadvantages of using Ada Grad

→ Its optimization method can result in aggressive, monotonically decreasing learning rates.

→ It is incredibly slow, this is because the sum of gradient squared only grows and never shrinks. (when we square the α_t value).

→ $\alpha_t \uparrow \longleftrightarrow \eta_t' \downarrow$

→ Ada Delta, RMSProp (for Root Mean Square Propagation) fixes this issue by adding a decay factor.

☆ Adadelta & RMSPROP

Adagrad → $W_{new(t)} = W_{old(t-1)} - \eta_t' \frac{\partial h}{\partial W_{old}}$

→ $\eta_t' = \frac{\eta}{\sqrt{W_{avg} + \epsilon}}$

gama

→ $W_{avg(t)} = \gamma \times W_{avg(t-1)} + (1-\gamma) \times \left(\frac{\partial L}{\partial W_t}\right)^2$

→ It restrict (α_t) to become a very very big number.

→ It should become Small, but not very very Small.

→ Since it prevent (α_t) to become very very big numbers, So your learning rate will not be very very Small and Convergence will not take that much time.

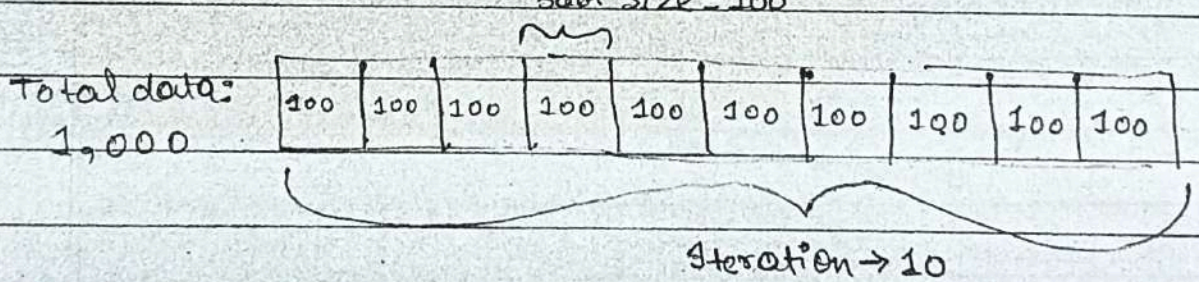
☆ Difference b/w Epoch and Iteration

→ An epoch is defined as the number of times an algor-ithm visits the data set. In other words, epoch is one backward and one forward pass for all the traing.

→ Iteration is defined as the number of times a batch of data has passed through the algorithm. In other words, it is the number of passes, one pass consists of one forward and one backward pass.

→ For example :-

* consider a Set of 1000 images having a batch size of 10 one each. iteration would process 10 images, it will take 100 Such iteration to complete 1 epoch.



- Total Dataset/No. of records → 10K
- Batch size → 1K

☆ Epoch 1 $\underset{\leftarrow}{\overset{GD}{\rightarrow}}$ Iteration 1

☆ Epoch $\underset{\leftarrow}{\overset{SGD}{\rightarrow}}$ Iteration = No. of records

☆ Epoch $\underset{\leftarrow}{\overset{\text{mini batch SGD}}{\rightarrow}}$ Iteration = $\frac{\text{Total Dataset}}{\text{batch size}}$

☆ Adam optimizer → Adaptive Moment Estimation

⇒ Adam combines the best properties of the AdaGrad (momentum) and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

⇒ with the help of momentum, you will get Smoothing, and with the help of RMS PROP, you will be able to change your learning rate in an efficient manner. Such that α_t value will also not go high.

⇒ ALGORITHM IMPLIMENTATION

$V_{dw} = 0 \quad V_{db} = 0 \quad S_{dw} = 0 \quad S_{db} = 0$

On iteration t

⇒ compute $\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}$ using current mini batch

$$V_{dw} = \beta_1 V_{dw} + (1-\beta)\frac{dL}{dw}$$

$$V_{db} = \beta_1 V_{dw} + (1-\beta)\frac{\partial L}{dw}$$

} Momentum → Smoothening factor

$$S_{dw} = \beta_2 S_{dw} + (1-\beta)\left(\frac{\partial L}{\partial w}\right)^2$$

$$S_{db} = \beta_2 S_{db} + (1-\beta)\left(\frac{\partial L}{\partial b}\right)^2$$

} RMS problem

$$w_b = w_{t-1} - \frac{\eta \times V_{dw}}{\sqrt{S_{dw} + \epsilon}}$$

$$b_t = b_{t-1} - \frac{\eta * V_{db}}{\sqrt{S_{db} * \epsilon}}$$

→ initial Learning rate

Adam optimizer

→ Smoothening factor

→ RMS properties

☆ Bias Correction

$$V_{dw}^{correction} = \frac{V_{dw}}{(1-\beta_1^t)}$$

$$V_{db}^{correction} = \frac{V_{db}}{(1-\beta_1^t)}$$

} Momentum → Smoothening factor

$$S_{dw}^{correction} = \frac{S_{dw}}{(1-\beta_2^t)}$$

$$S_{db}^{correction} = \frac{S_{db}}{(1-\beta_2^t)}$$

} RMS problem

$$\Rightarrow \quad w_b = w_{t-1} - \frac{\eta \times V_{dw}^{correction}}{\sqrt{S_{dw}^{correction} + \epsilon}}$$

$$b_t = b_{t-1} - \frac{\eta \times V_{db}^{correction}}{\sqrt{S_{dw}^{correction} + \epsilon}}$$

⊛ Multi class Cross Entropy Loss

$$L(x_i, y_i) = -\sum_{j=1}^{c} y_{ij} * \log(\hat{y}_{ij})$$

↳ categorical Cross Entropy

→ we use this when we have more than 2 outputs.

☆ Training ANN

Step 1 → Randomly initialise the weights to small numbers close to 0 (but not 0).

Step 2 → Input the first observation of your dataset in the input layer, each feature in one input node.

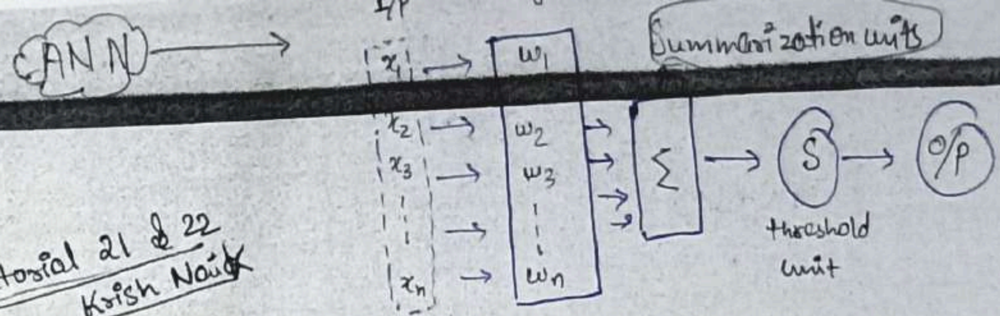
Step 3 → Forward-Propagation: from left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the predicted result y.

Step 4 → Compare the predicted result to the actual result. Measure the generated error.

Step 5 → Back-propagation: from right to left, the error is back-propagated. update the weights according to how much they are responsible for the error. The learning rate decides by how much we update the weights.

Step 6 → Repeat Steps 1 to 5 and update the weights after each observation (Reinforcement learning) or
Repeat Steps 1 to 5 but update the weights only after a batch of observations (Batch learning)

Step 7 → when the whole training set passed through the ANN, that makes an epoch. Redo more epochs.



Refer Tutorial 21 & 22 Krish Naik

☆ Convolution

→ Convolution is a mathematical operation on two function (f and g) that produces a third function ( ) that expresses how the shape of one is modified by the other. The term convolution refers to both the result function and to the process of computing it.

(6×6)

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

Suppose → white (0), black (1)

$*$ (3×3)

+1	0	-1
+2	0	-2
+1	0	-1

=

0	-4	-4	0
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0

→ Retransforming this image using min-max scalar

(4×4)

255	0	0	255
255	0	0	255
255	0	0	255
255	0	0	255

☆ Padding in CNN

→ Padding basically extends the area of an image in which a convolutional neural network processes.

→ In order to work the kernel/ filter with processing in the image, padding is added to the outer frame of the image to allow for more space for the filter to cover in the image.

→ Adding padding to an image processed by an CNN allows for a more accurate analysis of image.

before padding

→ $n-f+1$

$n=6-3+1$

If we want to get my output dimension as 6

→ $n-f+1=6$

→ $n=6+3+1$

→ ...

after padding

→ $p=1$, $n+2p-f+1$

→ $6+2\times1-3+1$

used for image recognition and tasks that involve the processing of pixel data.

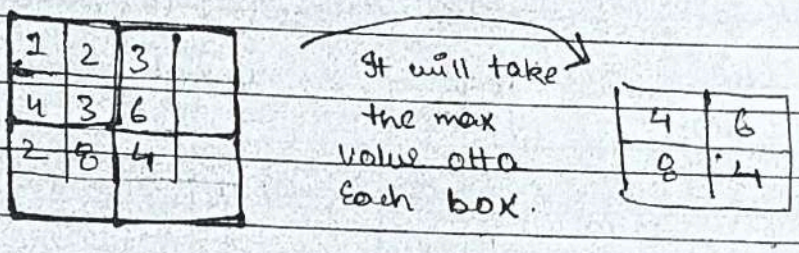
ANN	CNN → Convolutional neural networks
→ uses fully connected layers	→ Usses Partially connected layers, Size of connection depends on Size of the filter
→ can be used only for small images	→ can be used for any image
→ considers entire Image for learning the Patterns	→ Considers only the part of the image, which is in Receptive Field of the Filter..
→ Number of parameters will be very high	→ Number of Parameters will be very less.
→ Doesn't learn the patterns in Spatial Hierarchy i.e., Higher layers of ANN doesnt combine the lower layers outputs	→ Learns Spatial Hierarchy of Patterns i.e., Higher Layers of CNN are formed by combining Lower Layers. This helps in identifying the Patterns more effectively than ANN
→ Success rate is high	→ Success rate is more than ANN
→ Scalability is lower	→ Scalability is higher
→ neurons r fully connected	→ neurons are partially connected
→ has 2 basic layers → input & output	→ It has input, output, convolution and Pooling layers
→ It has 1D to 3D	→ It has 3D Neuron Volume
→ Not translation Invariant: Once a regular DNN has learned to recognize a pattern in one location, it can recognize it only in that location. So we have to train the image again (cant reuse the previous weights) if it is Rotated or Shifted.	→ translation Invariant: Once the CNN has learned to recognize a pattern in one location, it can recognize it in any location. So learning (weights) can be reused even if the image is Rotated or Shifted.

☆ Max Pooling

→ Max Pooling is a Pooling operation that calculates the maximum value for patches of a feature map, and uses it to create a downsampled (pooled) feature map. It is usually used after a convolutional layer.

→ It is used to reduce the dimensions of the feature maps. Thus it reduces the Number of parameters to learn and the amount of computation performed in the network.

→ This makes the model more robust to variations in the position of the features in the input image.



☆ Data Augmentation

→ Data augmentation in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created Synthetic data from existing data. It acts as a regularizer and helps reduce overfitting when training a machine learning model.

→ It basically flip, horizontal Shifts, vertical Shifts or zoom the data in the oder to give more variance.

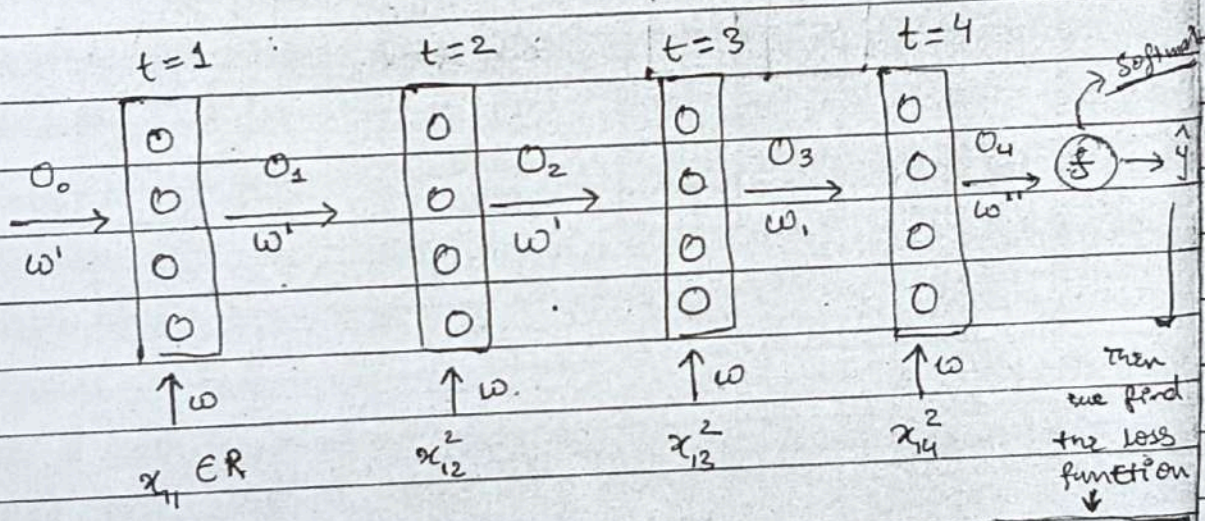
→ Because even if we provid flipped image to CNN, it shold be able to predict what image it is.

use of convolutional layers which basically use filters to help recognise important feature in an image. There are so many feature which can help distinguish a particular image. All these features are being learned internally. When we use a deep CNN, weights can be millions

RNN Deep Learning

→ Recurrent neural networks (RNN) are the state of the art algorithm for Sequential data and are used by Apple's Siri and Google's voice Search. It is one of the algorithms behind the scenes of the amazing achievements seen in deep learning over the past few years.

→ Recurrent Neural Networks (RNN) are a type of Neural Network where the output from the previous step is fed as input to the current step. RNN's are mainly used for, Sequence classification – Sentiment classification & video classification. Sequence Labelling – Part of Speech tagging & named entity recognition.



→ $O_1 = f(x_{11}\omega + O_0\omega')$

→ $O_2 = f(x_{12}\omega + O_1\omega_1)$

→ $O_3 = f(x_{13}\omega + O_2\omega_1)$

→ $O_4 = f(x_{14}\omega + O_3\omega_1)$

⇒ forward propagation has happened from left to right. just with respect to one Statement. Each and every words can be represented into vectors of some (d) dimentions

Transfer learning

whenever we train CNN on a bunch of Images, convolution network make use of convolution layers which basically use filters to help recognise important feature in an image. There are so many feature which can help distinguish a particular Image All these features are being learned internally, when we use a deep CNN, weights can be million when million of parameter need to be learn, its gon take a lot of time. So transfer learning is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.

for eg → knowledge gained while learning to recognize cars could be applied when trying to recognize trucks.

→ weight updation is important so that we can reach global minima in the particular gradient descent.

when million of parameter need to be learn, its gon take a lot of time. So transfer lea is a technique

and that depends on the types of vocablary, No. words that are present in that vocabulary.

→ The output (4), depends on x_{14} and output (3)
→ output (3) depends on x_{13} and output (2)
→ output (2) depends on x_{12} and output (1)

⇒ So by this way, the sequence information is alwa maintained., • It cant be discarded. which was th problem in bow, TF-IDF.

→ weight only gon assign during backward Propagation, & not forward.

☆ Backward propagation in RNN

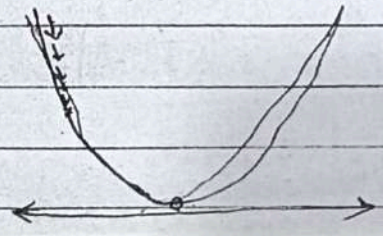
→ when the back propagation is getting applied, we ju finding the derivative and updating the wights from backward.

→ As Soon as we find desivative, which is actually a Slope, we will be able to find out the weights, we u update the weights and this will go on for man iterations.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial O_4} \cdot \frac{\partial O_4}{\partial w}$$ → for 1^{st} w from back.

☆ Problems in Simple RNN

RNN Suffer from the problem of vanishing gradients. The gradients carry information used in the RNN parameter update and when the gradient becomes Smaller and Smaller, the parameter updates become insignificant which means no real learning is done.



NLP

Complete Road Map to prepare NLP

→ PyTorch
→ Keras
→ Tensor Flow

→ SpaCy
→ Natural Language Analyses with NLTK

Bert
Transformers

Bidirectional LSTM RNN, Encoders / Decoders, Attention Models

Text Preprocessing Level 3 - word Embeddings, word 2vec

Understanding Recurrent Neural Networks, LSTM, GRU

Get the Understanding of Artificial Neural Network

Solve Machine Learning Usecases

Text preprocessing - Gensim, Klord2vec, Avg. word 2vec

Text Pre-processing Level-2, Bag of words, TFIDF, Unigram, Bi, n-Gram

Text preprocessing Level I - Tokenization, Lemmatization, Stopword, tenning, Part of Speech (POS) Tagging

Natural Language processing helps computers communicate with humans in their own language and scales other language-related tasks. For example, NLP makes it possible for computers to read text, hear speech, interpret it, measure sentiment and determine which parts are important.

⇒ Tokenization

- The first thing you need to do in any NLP project is text preprocessing.
- Tokenization can Separate Sentences, words, characters, or Subwords. when we Split the text into Sentences, we call it Sentence tokenization. For words, we call it words tokenization.

⇒ Stemming and Lemmatization

- Stemming and Lemmatization both generate the foundation Sort of the inflected words and therefore the only difference is that Stem may not be an actual word whereas, lemma is an actual language word.
- Lemmatization takes more time.

word	Stemming (↑ PorterStemmer)	Lemmatization (↑ wordNetLemmatizer)
information	inform	information
informative	inform	informative
computers	comput	computer
feet	feet	foot

- Problem with respect to Stemming is that it produces intermediate representation of the word may not have any meaning.
- Stopwords → words that actually does not put much value in the paragraph.

☆ Bag of words

A bag of words is a representation of text that describes the occurrence of words within a document. It involves two things: A vocabulary of known words. A measure of the presence of known words.

Eg →

Sentence 1 = He is a good boy

Sentence 2 = She is a good girl

Sentence 3 = Boy & girl are good

(i) Vectors

(BOW)

	good	boy	girl	o/p
Sent 1	1	1	0	-
Sent 2	1	0	1	-
Sent 3	1	1	1	-

→ Disadvantages

- It gives the same preference to all the words.
- If the new sentences contain new words, then our vocabulary size would increase and thereby, the length of the vectors would increase too
- Additionally, the vectors would also contain many 0s, thereby resulting in a sparse matrix (which is what we would like to avoid).
- It is messy and techniques like machine learning algorithms prefer well defined fixed-length inputs & outputs.

TF - IDF

→ TF (Term frequency) ⇒ $\frac{\text{No. of repetation of words in Sentence}}{\text{No. of words in Sentence}}$

→ IDF (Inverse Document Frequency) ⇒ $\log\left(\frac{\text{No. of Sentences}}{\text{No. of Sentences containing words}}\right)$

→ So we have to multiply TF & IDF to quantify words in a set of documents.

Eg → Sentence 1 → good boy

Sentence 2 → good girl

Sentence 3 → boy girl good

⇓

TF

	good	boy	girl
Sent-1	1/2	1/2	0
Sent-2	1/2	0	1/2
Sent-3	1/3	1/3	1/3

(3×3)

×

IDF

words	IDF
good	$\log(3/3) = 0$
boy	$\log(3/2)$
girl	$\log(3/2)$

(3×1)

	f_1 good	f_2 boy	f_3 gir
Sent-1	0	$\frac{1}{2} \times \log(3/2)$	0
Sent-2	0	0	$\frac{1}{2} \times \log(\frac{3}{2})$
Sent-3	0	$\frac{1}{3} \times \log(\frac{3}{2})$	$\frac{1}{3} \times \log(\frac{3}{2})$

⇒ when we were using BOW, we just had values like 0 and 1. So, TF-IDF gives Symantic meaning to the data.

☆ word 2Vec

Bag of words and TF-IDF Problems

⇒ Both BOW and TF-IDF approach Semantic information is not stored. TF-IDF gives importance to uncommon words.

⇒ There is definitely chance of over fitting.

→ Solution — Word 2 Vec

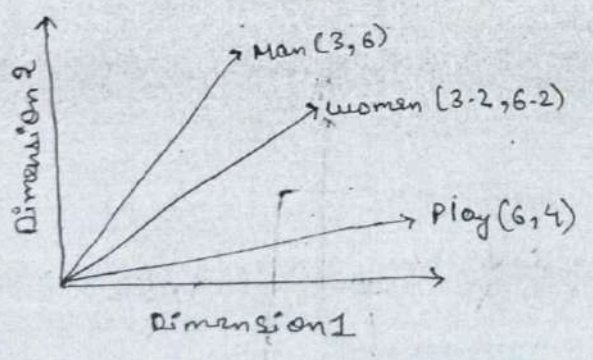
⇒ In this specific model, each word is basically represented as a vector of 32 or more dimension instead of a single number.

⇒ Here the Semantic information and relation b/w different words is also preserved.

⇒ Another great advantage of word 2 Vec. approach is that the size of the embedding vector is very small.

Stopwords → words that actually does not put much value in the paragraph.

Visual Representation - word 2 Vec



⇒ King - Man + Women = Queen

Steps to create word2vec

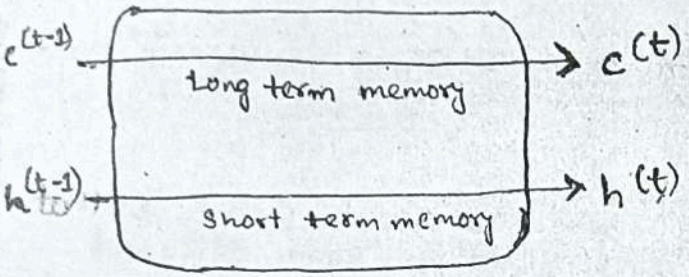
→ Tokenization of the Sentences
→ Create Histograms
→ Take most frequent words
→ Create a matrix with all the unique words. It also represent the occurrence relation between the words

∵ we use PorterStemmer() for Stemming purpose

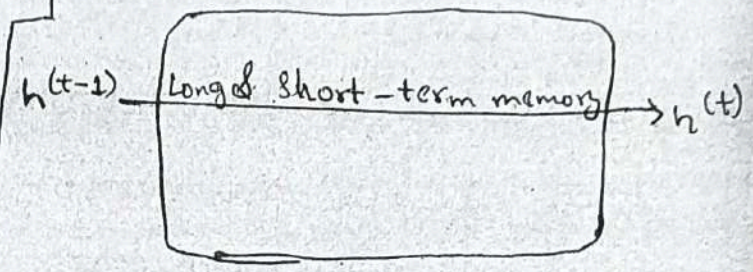
Gated Recurrent Unit (GRU)

LSTM	GRU
→ 3 Gates: Input, output, forget	→ 2 Gates: reset, update
→ More accurate on longer sequence less efficient	→ More efficient computation wise. Getting more popular.
→ Invented: 1995 - 1997	→ Invented: 2014

LSTM

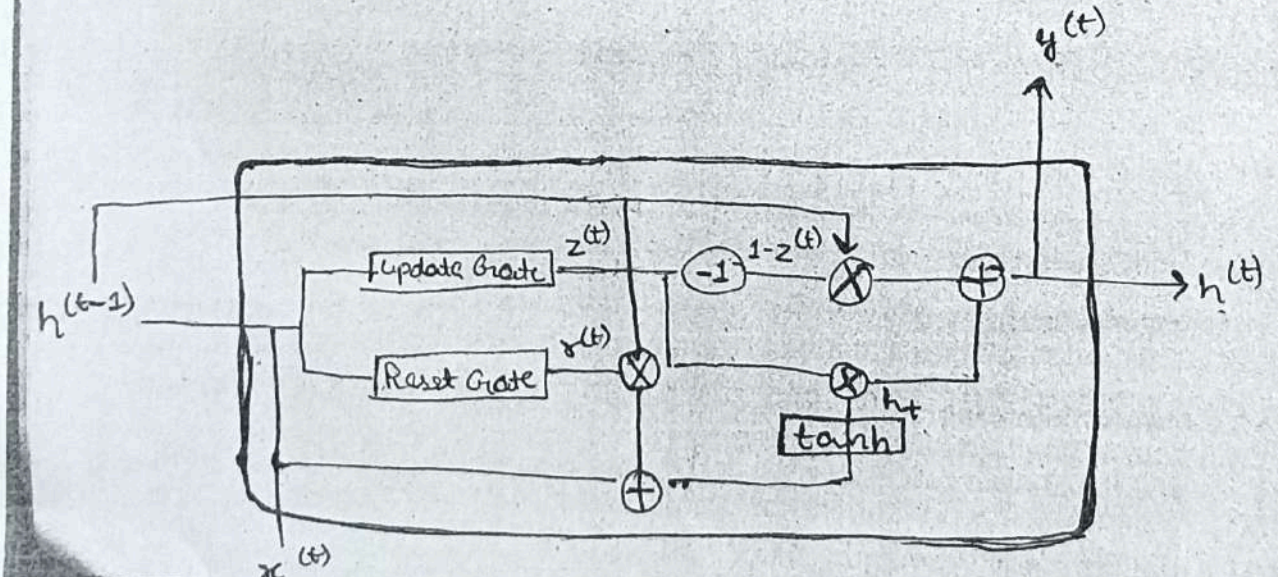


GRU



→ GRU is a lightweight version of lstm where it combines long and short term memory into its hidden state.

→ Lstm has two cell state and hidden state here it has only hidden state which can combine both long & short term memory



→ $z_t = \sigma_g (W_z x_t + U_z h_{t-1} + b_z)$

→ $r_t = \sigma_g (W_r x_t + U_r h_{t-1} + b_r)$

→ $\hat{h}_t = \phi_h (W_h x_t + U_h (r_t \odot h_{t-1}) + b_h)$

→ $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$

Variables :-

- x_t → input vector
- z_t → update gate vector
- h_t → output vector
- r_t → reset gate vector
- $\hat{h}_t$ → candidate activation vector
- W, U and b: parameter matrices & vector.

The crux of this whole GRU unit is, It combines short-term memory and long term memory into one state and its also little bit lightweight than lstm.

Seq2Seq learning with Neural Networks

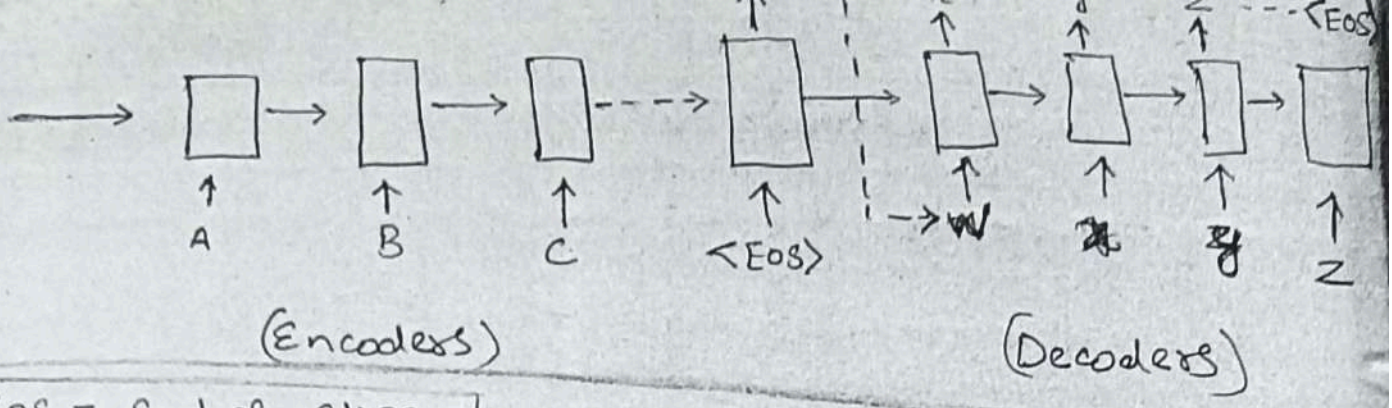
→ In Seq2Seq, my model takes a sequence of information, and it output some sequence of information.

→ Suppose we want to convert english to french, or converting image to text. (Image captioning).

→ for Encoders, we dont have the output at every timestand.

→ when we get an 'context vector' out of 'Encoder', this vector is then passed to the 'decoders'. For decoders, we do have the output at every timestand.

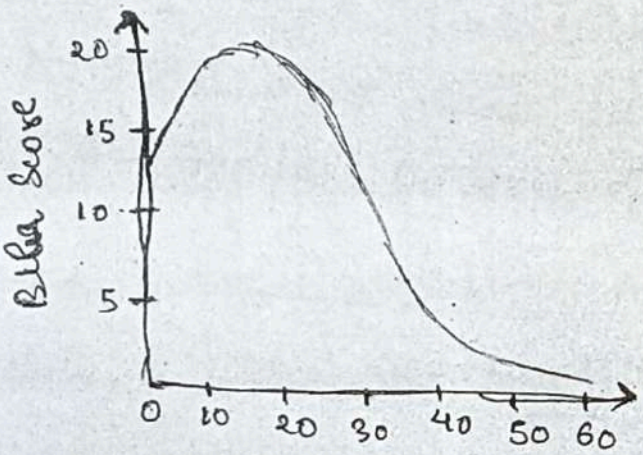
→ The input sequence and the output sequence need not



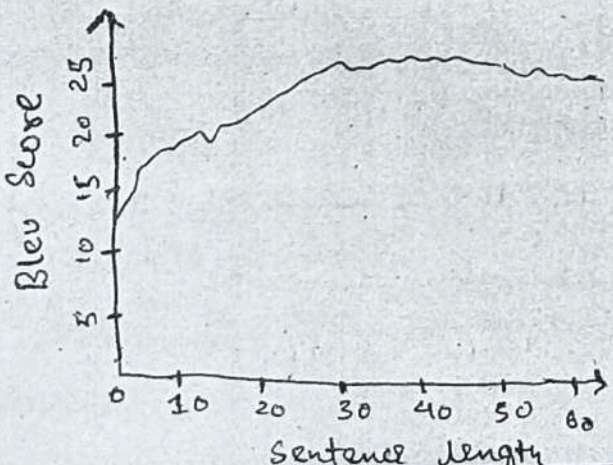
→ EOS = End of Stream

→ One disadvantage is that, whenever you have long sentences, the accuracy is not that good. → disadvantage of Encoders & Decoders

• BLEU → To measure the machine translation effectiveness, we will evaluate the closeness of the machine translation to human reference translation using a metric known as BLEU - Bilingual Evaluation Understudy.

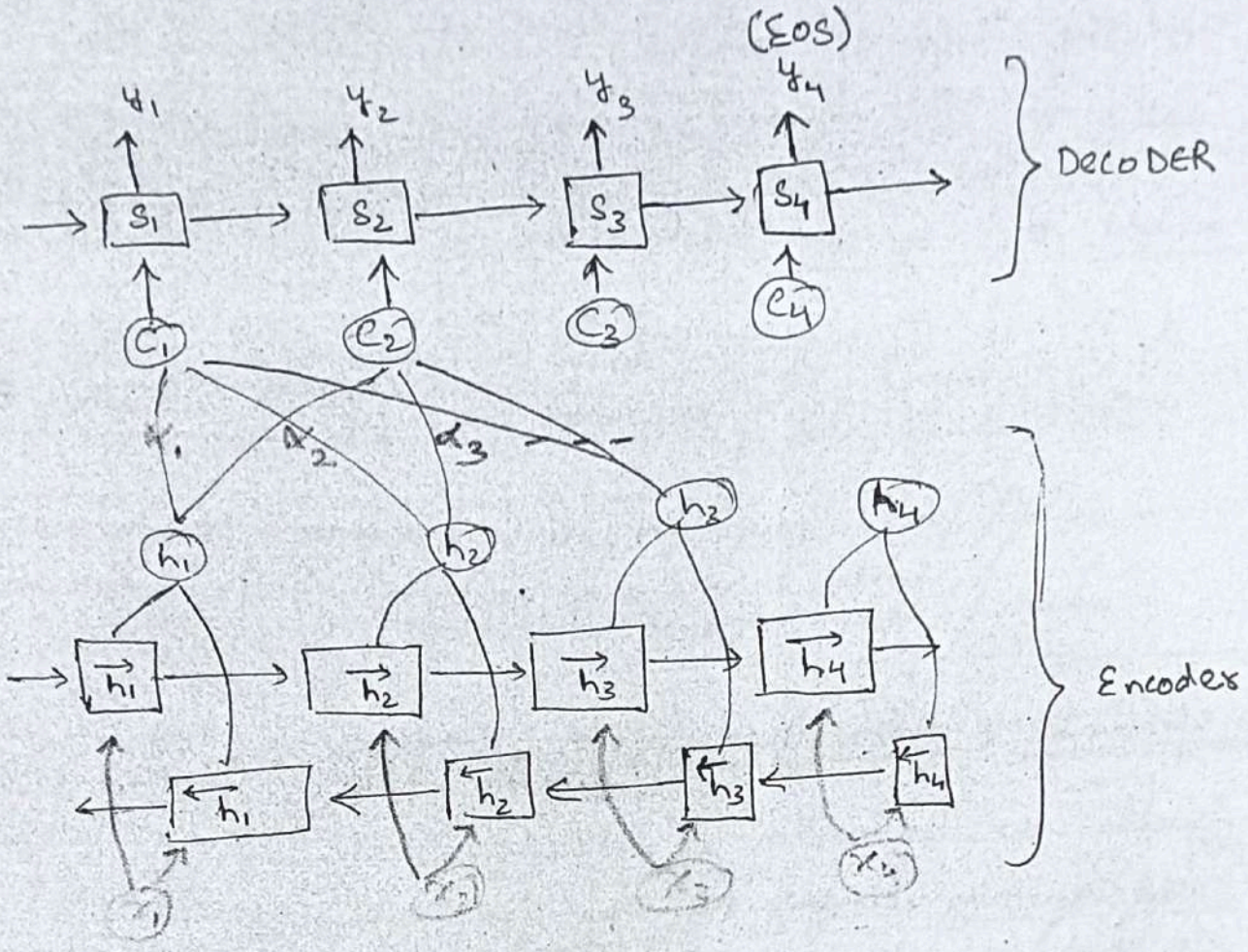


→ It shows that, as the sentence length increasing the bleu score is coming down.



→ In the odes to solve the problems faced by seq2seq we use Bidirectional - LSTM RNN. as we can see in this graph, the accuracy has improved a lot.

☆ Bi-LSTM RNN (Bidirectional LSTM RNN)



→
- x = Input
- h = ouput of various hidden layers
- c = contex vector
- y = output
- S = Sate of the decoders
- α = (alpha) Sigmoid activation functio

→ Conditions with respect to α_i

(1) The Sum of all the alphas within a time Spane Should be Equal to 1.

(2) we can calculate the contex' vector by multiplying the output of bidirectional LSTM (h) with the alpha values. like how in deep learning, we multiply inputs with the weights, we get Some kind of output. In the Similar way.

Feedforward NEural Network → $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$

- The reasons researchers may have applied this feed forward neural network, is basically for the back propagation. because we need to update these values of $\alpha_1, \alpha_2 - - - - \alpha_n$. Unless and until we dont updates the weights, you wont get good accuracy.
- By applying Softmax function, we can make sure that the value of (α_{ij}) is equal to 1.
- The weight (α_{ij}) of each annotation (h_j) is computed by :-

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

where $e_{ij} = a(S_{i-1}, h_j)$ (feed-forward neural network)

- **Transformer : Attention is all you need**

→ It adopts the mechanism of self-attention.

→ It differentially weight the significance of each part of the input data.

→ Like rnn, transformers are designed to handle sequential input data, such as natural language, for tasks such as translation and text summarization.

→ However, unlike RNNs, transformers do not necessarily process the data in order. ~~the~~ rather, the attention mechanism provides context for any position in the input sequence.

- **Advantages of transformers**

① They hold the potential to understand the relationship between sequential elements that are far from each other.

② They are way more accurate.

③ They pay equal attention to all the elements in the sequence.

(4) Transformers boast of faster processing. In other words, they can process more data in lesser time.

(5) They could work with virtually any kind of sequential data

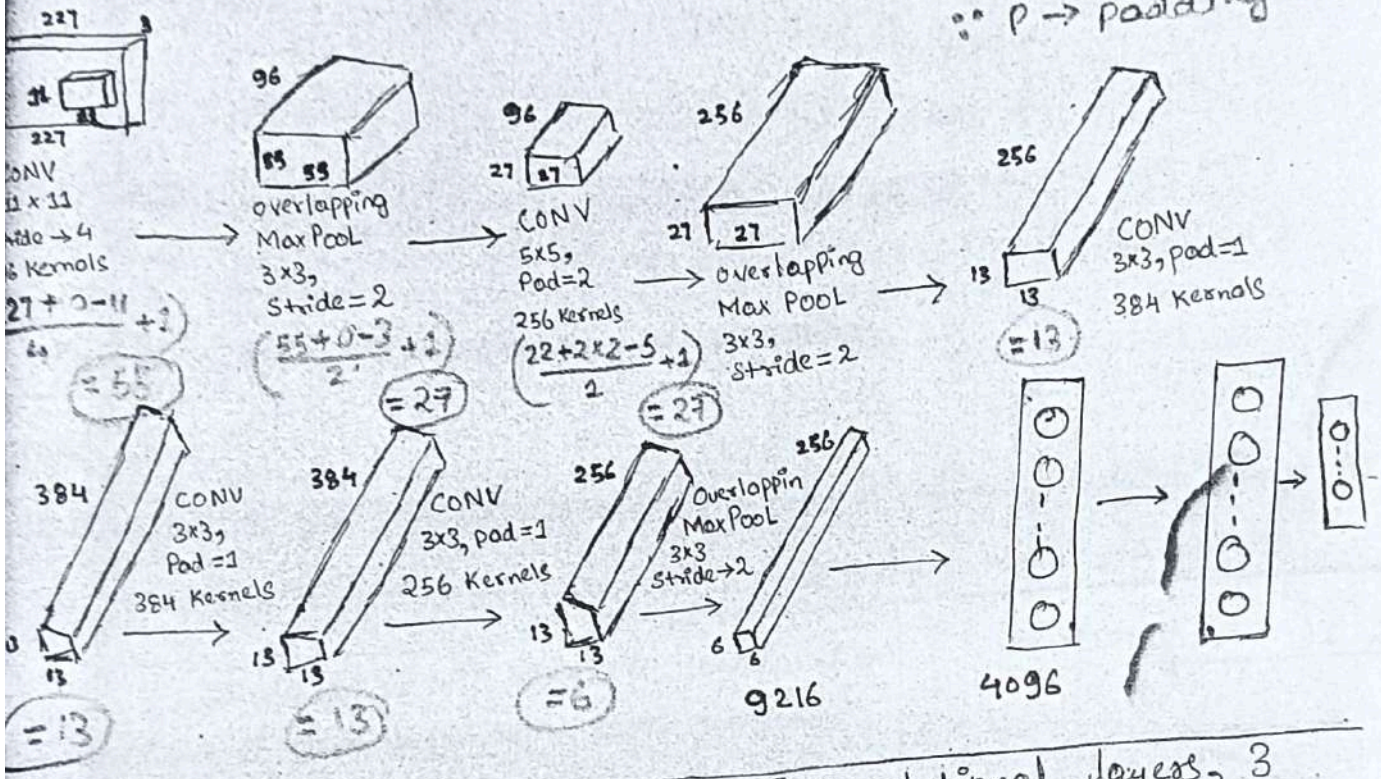
(6) They can predict what could happen next.

(7) Transformers seem to be helpful in anomaly detection.

ALEXNET ARCHITECTURE

formula → $\left(\frac{n+2p-f}{s}+1\right)$

∵ p → padding



---

(1) Alexnet architecture consists of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 Softmax layer.

(2) Each convolutional layer consists of convolutional filters & a nonlinear activation function ReLU.

(3) The pooling layers are used to perform max pooling.

(4) Input size is fixed due to the presence of fully connected layers

(5) The input size is mentioned at most of the places as $224 \times 224 \times 3$ but due to some padding which happens it works out to be $227 \times 227 \times 3$. It has 60 million parameters.

→ Alex Net is a leading architecture for any object-detection task and may have huge applications in the coputer vision sector of artificial intelligence problems. In the future, AlexNet may be adopted more than CNNs for image tasks.

Important things to learn in Reinforcement

→ Deep Q Networks

→ Deep Deterministic Policy Gradient

→ A3C Algo

⇒ Q Learning

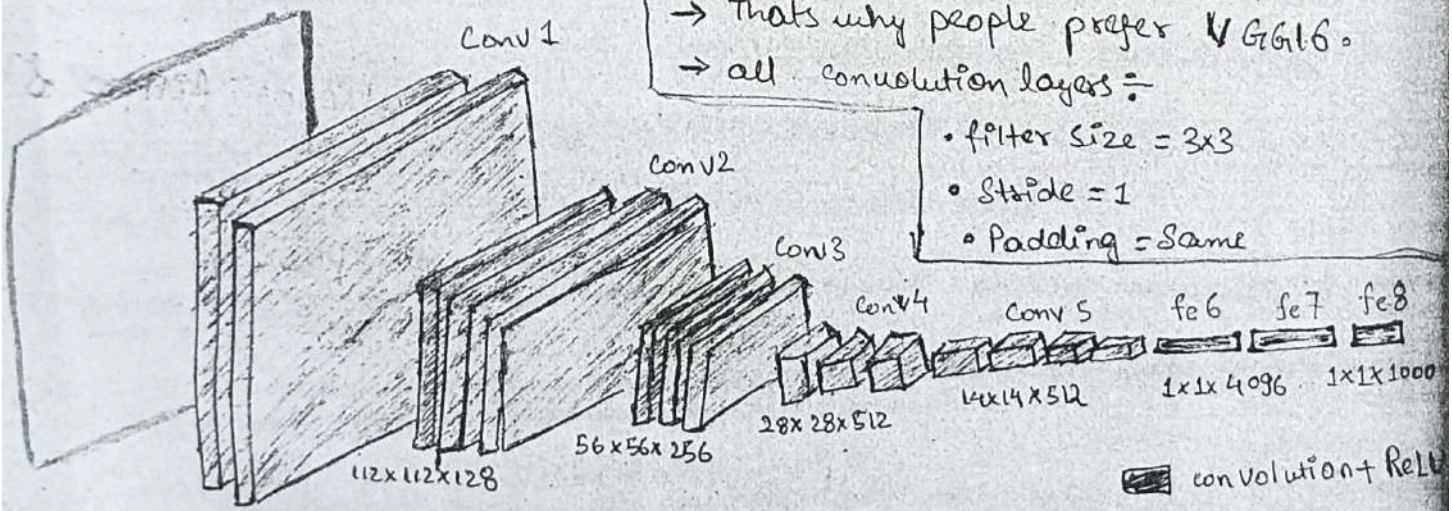
VGG 16 Architecture

→ In alexnet, we have lot of variation, Something that you dont find in VGG16

→ Thats why people prefer VGG16.

→ all convolution layers :-

- filter size = 3x3
- Stride = 1
- Padding = Same



Hummingbird

- Hummingbird is a library for compiling trained tradition ML models into tensor computations.
- Hummingbird allows users to seamlessly leverage neural network framework (such as PyTorch) to accelerate traditional ML models.
-
- In general, Hummingbird syntax is very intuitive and minimal. To run your traditional ML model on DNN frameworks, you only need to (import hummingbird-ml) and add to ('dnn_framework') to your code.

TensorDash

TensorDash is an application that lets you remotely monitor your deep learning model's metrics and notifies you when your model training is completed or crashed.