Sr. No.	Title	Page No.	Sign./Remarks

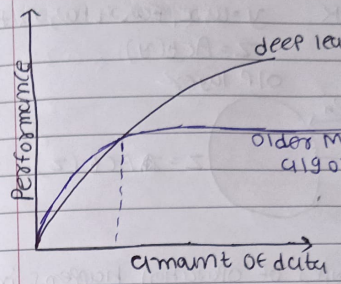
⇒ DL is part of a broader family of ML methods based on artificial neural network (ANN) with representation learning

⇒ Representation learning: ML we need to give Feature while AI detect feature automatically from given data

Ex Dog or cat ML need features like size, colours (Need feature eng)
AI ~~detect extract~~ feature automatically (no need to feature engineering)

⇒ DL algo uses multiple layers to progressively extract higher-level features from raw inputs Ex in image lower layers may identify edges while higher level may identify the concepts relevent to human such as digits or letters or face

ML & DL is used for find relation betn input and output mostly
⇒ ML uses statistic technique for this while DL follow Logical structure called neural networks

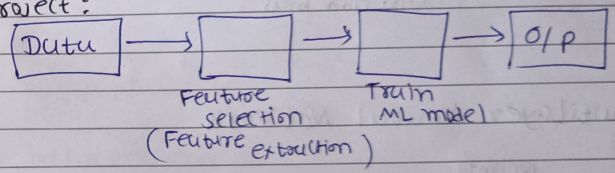


⇒ deep learning can solve complex problems like Image & object detection, Neural network problems, speech recognition, Chat box

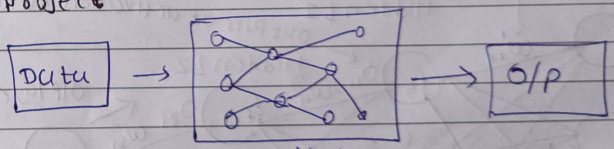
⇒ DL needs more data to do high performance

How do data science techniques scale with amount of data?

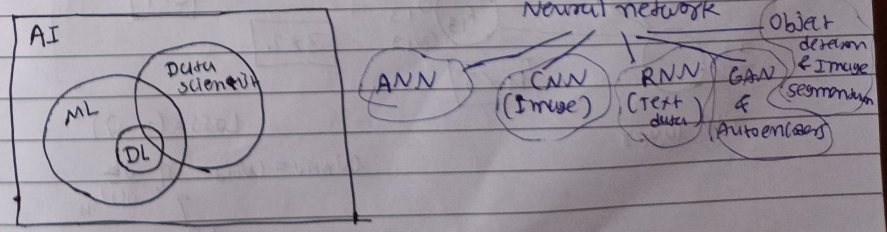
ML Project:



DL Project



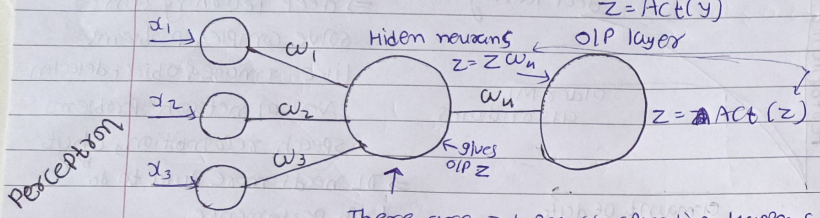
This combine Feature extraction & Classification or Regression



✱ How neural network work

$y = w_1x_1 + w_2x_2 + w_3x_3 + bias$

$z = Act(y)$



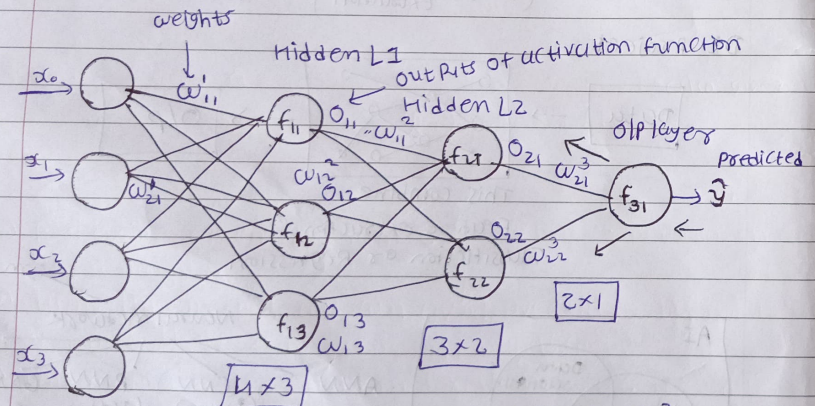
There are 2 types of operation happens inside hiden neurons

Step 1: ~~⊖~~ → Step 2

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

and add bias

✱ MultiLayer Neural Network



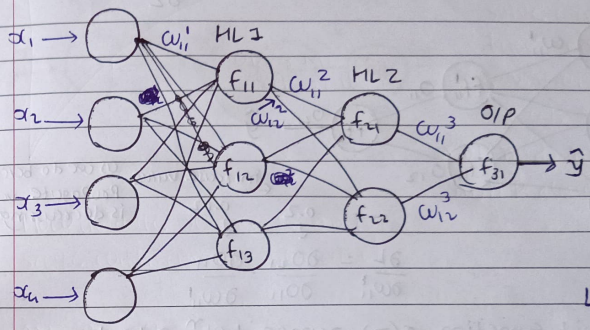
$Loss = (y - \hat{y})^2$

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

η ← LR

✱ Chain Rule

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



$Loss = \sum_{i=1}^{n} (y - \hat{y})^2$

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

→ w_{11} is impact of O_{31}

→ w_{21} is impacting O_{31} so $\frac{\partial L}{\partial w_{21}^3} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{21}^3}$

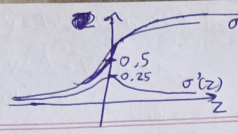
→ $w_{11}^2 = w_{11\,old}^2 - \eta \frac{\partial L}{\partial w_{11}^2}$

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

one const + second's derivative + second const 1st derivative

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

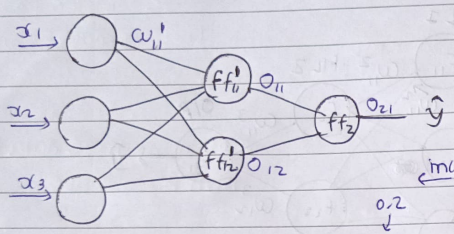
$\frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{12}^2} = \frac{\partial L}{\partial w_{12}^2}$



✱ Vanishing gradient Problem

$0 \leq \sigma(z) \leq 1$

$0 < \frac{\partial \sigma(z)}{\partial z} \leq 0.25$



$\frac{\partial L}{\partial w_{11}'} = \frac{\partial o_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w_{11}'}$

0.2 ← increase value ← 0.02

as we do back Propagate value is decreasing

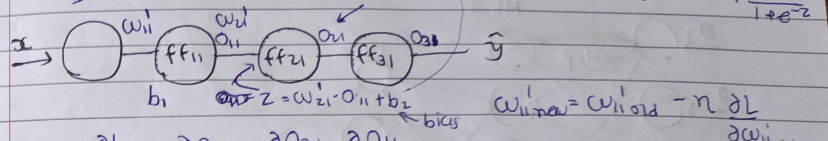
⇒ For sigmoid function $\sigma(z)$ range bet^n 0 to 1
and $\sigma'(z)$ 0 to 0.25
So as the hidden layer increase $\frac{\partial L}{\partial w_{11}'}$ decreases

and $w_{11\,new} \approx w_{11\,old}$

so sigmoid function gives vanishing gradient problem and many other also gives same Problem Ex tanh $0 < z < 1$

✱ Exploding gradient Problem

activation function
let activation fun is sigmoid fun $\frac{1}{1+e^{-z}}$



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

$\frac{\partial L}{\partial w_{11}'} = \frac{\partial o_{31}}{\partial o_{21}} \frac{\partial o_{21}}{\partial o_{11}} \frac{\partial o_{11}}{\partial w_{11}'}$

$\frac{\partial o_{21}}{\partial o_{11}} = \frac{\partial \phi(z)}{\partial z} \frac{\partial z}{\partial o_{11}}$

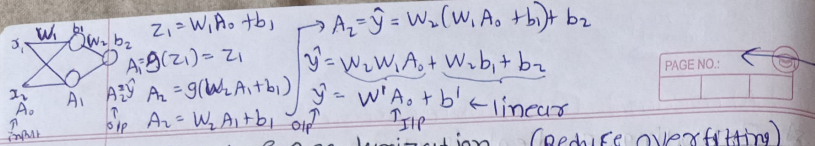
$0 < \sigma'(z) < 0.25$

$0 < \phi'(z) < 0.25$

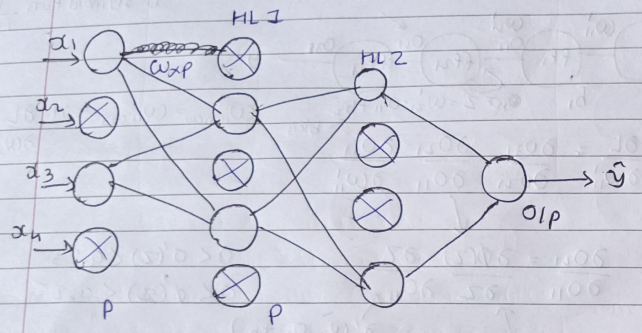
$0 < \sigma'(z) < 0.25$

$\frac{\partial (w_{21} o_{11} + b_2)}{\partial o_{11}} = w_{21}$

when we have higher weight exploding gradient Problem will occure and gradient decent never converge to global minima



★ Drop out & Regularization (Reduce overfitting)



Drop out ratio p ⇒ to improve overfitting we use drop out method

$0 \leq p \leq 1$

In this method we deactivate some neurons & also activate and deactivate input features.

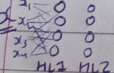
⇒ For test data we activate all features and all neurons and multiply $w(1-p)$ with every weight

⇒ For p value we can use hyperparameter optimization (Ex cross validation)

⇒ For every epoch we remove p fraction of neuron in each HL randomly.

⇒ using this method we can avoid overfitting by not focusing of single thing & focusing on every things.

⇒ this method is like random forest here we uses diff neural networks instead of diff dicision trees

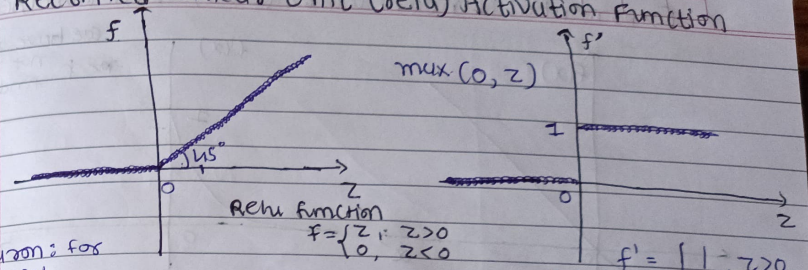
Ex  $\rightarrow \hat{y}$ Total possible neural network 2^8 (without using p)

using epoch 100 Approximate 100 diff neural network possible for 100 epochs

HL1 HL2

$g(z) = z$ linear function ⇒ without or linear activation function only use on linear regression and logistic regression model can not work on non linear data without non linear fun we can not do non linear regression or classification problems that are not linearly seperable

Rectified Linear Unit (relu) Activation Function



dead neuron: for any given input op is zero

Reason for dying neuron Problem: $z_1 = w_1 x_1 + w_2 x_2 + b_1$

$z_1 < 0$

If high LR High -ve bias

$\theta = \theta - \eta \frac{\partial L}{\partial \theta}$

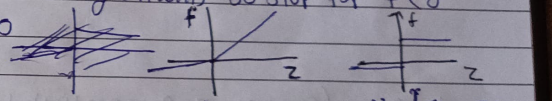
$w_{old} = w_{new} - \eta \frac{\partial L}{\partial w}$

derivative of each is either 0 or 1

⇒ when one of derivate is zero then $w_{old} = w_{new}$

This create dead neuron or dead activation function

⇒ to overcome this problem we use Leaky relu In this we will add some value (very small) so slop for $f' < 0$ have some value > 0

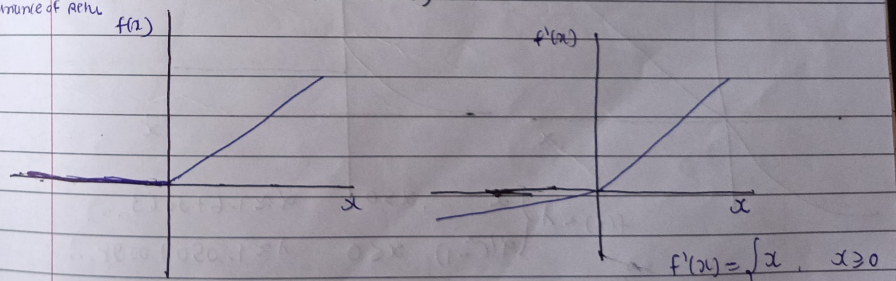


⇒ once neuron is dead then it is not recover because if $z_1 < 0$ that means w_1 and $w_2 < 0$ or $b_1 < 0$ and since our data is normalized so x_1 & x_2 are small and for any input $z_1 < 0$

⇒ Solution low LR +ve bias use variant of relu

Leaky relu → $f = \begin{cases} z, & z>0 \\ 0.01(z), & z<0 \end{cases}$ $f' = \begin{cases} 1, & z>0 \\ 0.01, & z<0 \end{cases}$

PRelu (Parametric ReLU)



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

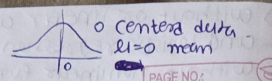
$f'(x) = \begin{cases} x, & x \geq 0 \\ ax, & x < 0 \end{cases}$

Learning Parameter

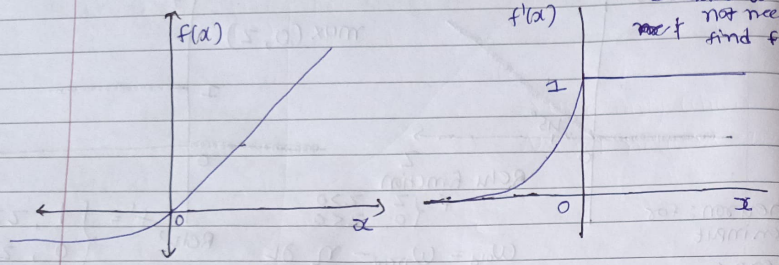
when $a = 0.01$ It's become Leaky relu

If $a = 0$ Relu

If a is learnable parameter it become PRELU



Exponential Linear units (ELU) function (Activation)



~~Gradient does not~~ not need to find f'(0)

$$f(x) = \begin{cases} x & , \text{if } x \geq 0 \\ \alpha(e^x - 1) & , x < 0 \end{cases}$$

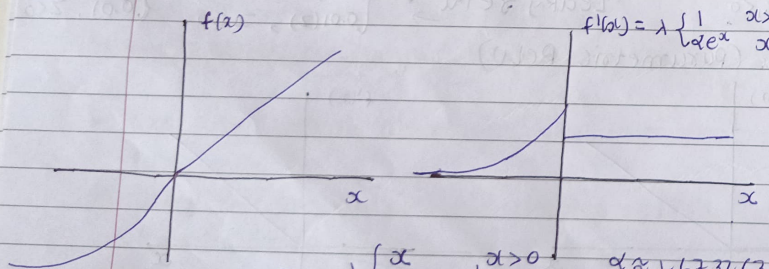
⇒ No dead ReLu issues, continuous & differentiable ← convergence faster

⇒ mean of OP is close to 0, zero centered

⇒ computationally expensive because $\frac{\partial(e^x - 1)}{\partial x}$ take time

⇒

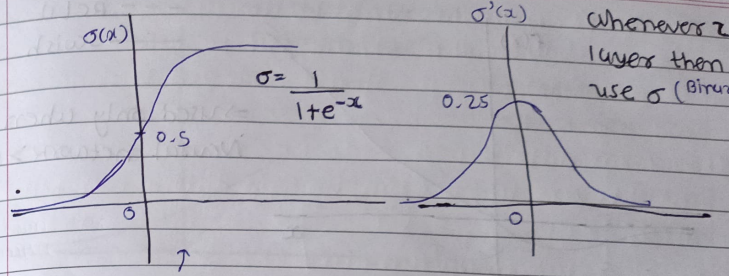
★ SELU (Scaled Exponential Linear unit)



$$f(x) = \lambda \begin{cases} x & , x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases} \qquad \alpha \approx 1.6732632\ldots, \quad \lambda \approx 1.0507009\ldots$$

⇒ self normalizing (OP is $\sigma = 1$, mean $= 0$) so converge faster

sigmoid activation function

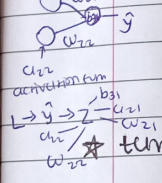


whenever 2 OP layer then we use σ (Binary)

⇒ Not zero centered data will we reduce the efficiency of weight update.

⇒ computationally expensive ($\because \frac{\partial(\frac{1}{1+e^{-x}})}{\partial x}$)

⇒ σ'(x) is range to 0 to 0.25 so vanishing gradient problem occures.



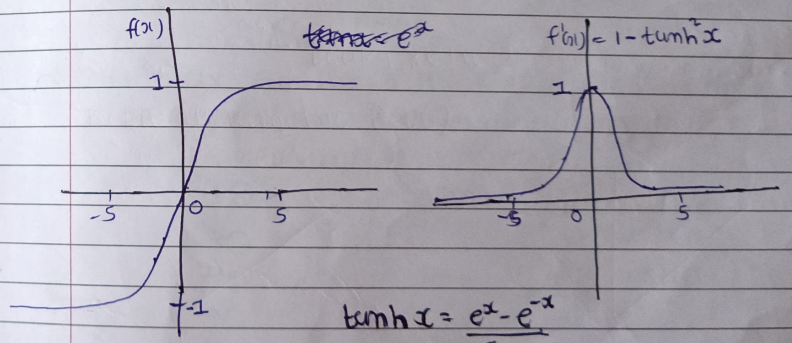
$$\frac{\partial L}{\partial w_{21}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_{31}} a_{21} \qquad 0 < a_{21}, a_{22} < 1$$

$$\frac{\partial L}{\partial w_{22}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_{31}} a_{22}$$

⇒ So both gradient are +ve or -ve and for all weight we increase or decrease it all together and converge slowly

⇒ OP from activation not normalized (mean ≠ 0) hence this problem occure (not zero centred)

★ tanh activation function



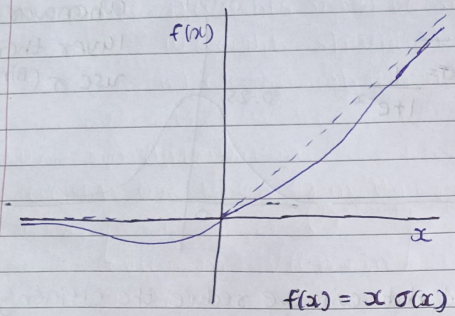
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

⇒ almost like sigmoid but f'(x) range to 0 to 1

⇒ data is zero centered which is better than sigmoid

⇒ same ~~differ~~ vanishing gradient occurs.

Swish (A self-Gated) activation Function



--- ReLU
—— Swish

⇒ used only when Neural network > 40 layers

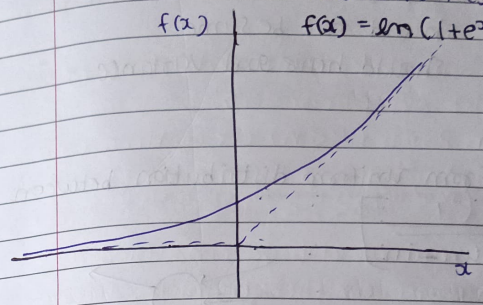
$f(x) = x\,\sigma(x)$

$f(x) = x \dfrac{1}{1+e^{-x}}$

Softplus activation function

$f(x) = \ln(1+\exp x)$

$f(x) = \ln(1+e^{x})$



—— Softplus
---- Rectifier

☆ Softmax

$$S(x_i) = \frac{e^{x_i}}{\sum_{K=1}^{K} e^{x_K}} = \frac{e^{x_i}}{\sum_{K=1}^{K} e^{x_K}}, \quad i = 1, 2, 3 \ldots K$$

$$0 \leq S(x_i) \leq 1$$

⇒ Soft max function is used when we have more than 2 OP layers (For 2 OP layers σ)

O/P layers

$x_1 \rightarrow \bigcirc$
$x_2 \rightarrow \bigcirc$
$x_3 \rightarrow \bigcirc$
$x_4 \rightarrow \bigcirc$

$$S(x_1) = \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}+e^{x_4}}$$

$$S(x_2) = \frac{e^{x_2}}{11}, \quad S(x_3) = \frac{e^{x_3}}{11}, \quad S(x_4) = \frac{e^{x_4}}{11}$$

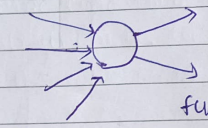
⇒ $\sum_{i=1}^{K} S(x_i) = 1$

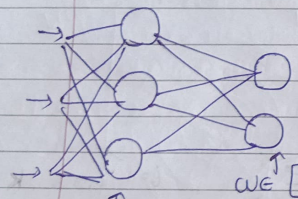
Various weight initialization Techniques

Key points i) weights should be small
ii) weights should not be same
iii) weights should have good variance

1) uniform distribution

weights are taken from uniform distribution between range $\left[\frac{-1}{\sqrt{fan\_in}}, \frac{1}{\sqrt{fan\_in}}\right]$





For this column weights are in range of

$w \in \left[\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right]$

⇒ work nicely with σ

2) Xavier / Gorut — work nicely with σ & tanh

i) xavier / Gorut Normal

$w_{ij} \in N(0, \sigma)$ (normal distribution)

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

ii) Xavier / Gorut Uniform

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

(uniform distribution)

3) He init — work nicely with ReLU

i) He uniform

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

ii) He Normal

$w_{ij} \in (0, \sigma)$

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

Optimizer

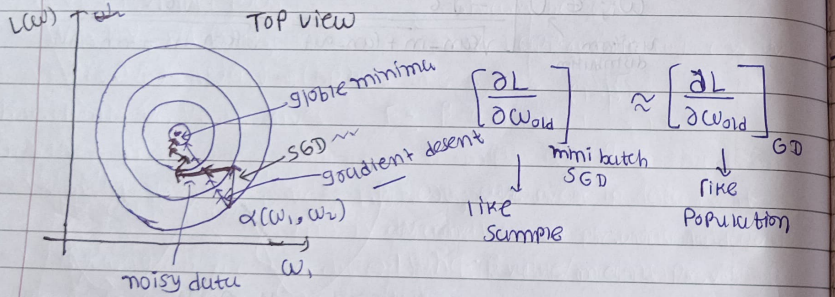
Stochastic Gradient Descent (SGD)

$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$ Total n data points

⇒ If we consider only one data point at a time and finding $\hat{y}$ and update the weight is called SGD.

⇒ If we consider K data points ($K<n$) at a time and finding Loss = $\sum_{i=1}^{K}(y-\hat{y})^2$ and update the weights is called mini SGD (no of updates per epoch = batch)

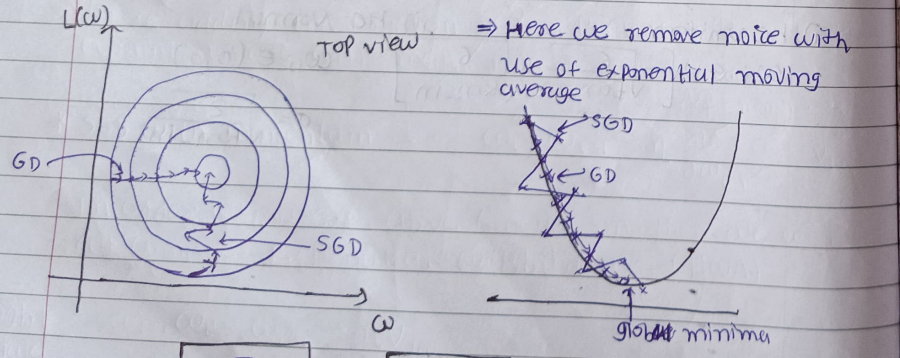
⇒ If we take all n data at a time and find Loss = $\sum_{i=1}^{n}(y-\hat{y})^2$ and updates weights then its called gradient descent (GD)



$\left[\frac{\partial L}{\partial w_{old}}\right]_{\text{mini batch SGD}} \approx \left[\frac{\partial L}{\partial w_{old}}\right]_{GD}$

like sample — like Population

SGD with momentum



Next →

⇒ Epoch can be understood as the number of times the algo scans entire data whereas iteration is number of times a certain batch passed via algorithm. Ex a dataset having 20 samples, batch size = 4, epoch = 5

Ex There will be 5 batches (20/4) each batch will passed by the algo there will be 5 iteration per epoch. since epoch is set as 5 number of iterations will be 5 × 5 = 25 (5 batches × epoch)

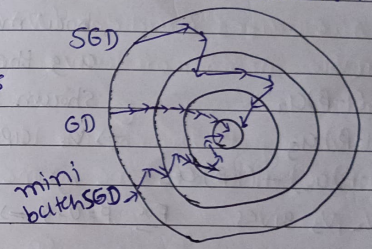
Ex For Batch size = 1000
and total data 10000
let we have 20 epochs
So in 1 epoch → iteration will be $\frac{10000}{1000} = 10$ iteration
So total iteration = 10 × 20 = 200

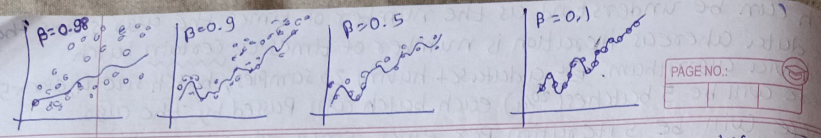
(to forward & backward Propagation happens)

⇒ In case of GD there are 1 iteration in one epoch $\sum_{i=1}^{n}(y-\hat{y})^2$

⇒ For SGD
iteration = no of records
$(y-\hat{y})$

In 1 epoch = n iterations





PAGE NO.:

⇒ we have some date point at different-different time intervels

time	t_1	t_2	t_3	...
date	a_1	a_2	a_3	...
after denoising	V_1	V_2	V_3	...

$V_t = \beta V_{t-1} + (1-\beta) S_t$ ← date point at point t; hyper parameter

$(0 < \beta < 1)$ $\beta = 0.9$ in most of cases

Smoothen value at point t

⇒ Choice of β will determine how many date point that we avg the value of V_t as shown below

$V_1 = a_1$

$V_2 = \beta V_1 + (1-\beta) a_2$

$V_3 = \beta V_2 + (1-\beta) a_3$

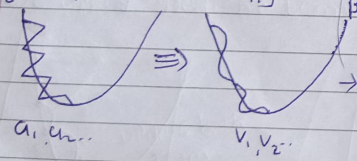
$= \beta(\beta V_1 + (1-\beta) a_2) + (1-\beta) a_3$

⇒ V_t approximating avg over $\frac{1}{(1-\beta)}$

Ex $\beta = 0.9 \Rightarrow \frac{1}{1-0.9} = 10$

These points V_1, V_2, V_3 gives smoothen line

$V_n = (1-\beta)[\beta^{n-1} a_1 + \beta^{n-2} a_2 + \dots + a_n]$



$\beta = 0.9$, ave over 10 date point

Normal avg of previous 10 date point

→ Higher β value means previous point take more importance

⇒ Here adding the fraction of the previous update to the current update will make process a bit faster.

* SGD with momentum $\quad w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$ ← For moment it replace by V_{dw_t}

In order to make above function smoothen we will compute Exponential weighted avg (EWA)

weight update $\quad w_t = w_{t-1} - \eta V_{dw_t}$

where $V_{dw_t} = \beta V_{dw_{t-1}} + (1-\beta) \frac{\partial L}{\partial w_{t-1}}$ ← exponential smoothing fun

V_{dw_t} called as velocity and it accelerates gradients in the direction that leads to convergence.

Convex Fun occures For ML problems

Ex Linear regression, Logistic regression

Non-convex Fun occures For DL prob

$\beta \approx 0.95$ for smoothening of curve

⇒ β is showing the how fast velocity exact decay with time

⇒ For $\beta = 0.9$ current velocity is equal to $\left(\frac{1}{1-\beta} = 10\right)$ past 10 velocity avg of

bias update $\quad b_t = b_{t-1} - \eta V_{db_t}$

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

⇒ Instead of $\frac{\partial L}{\partial w_{t-1}}$ we replace it as V_{dw_t}

~~⇒ Initially For every iteration $V_{dw} = 0$, $V_{db} = 0$ and over For every batch we compute~~

⇒ on iteration t inside epoch compute derivative of weights and bais on the current mini batch and we compute V_{dw} and V_{db} and updates weight and bias $w = w - \eta V_{dw}$, $b = b - \eta V_{db}$

initially $V_{dw} = 0$, $V_{db} = 0$

⇒ In case of mini batch gradient descent when we update the model parameters after iterating through all the date points in the given batch, thus the direction of the update will have some variance which leads to oscillations.

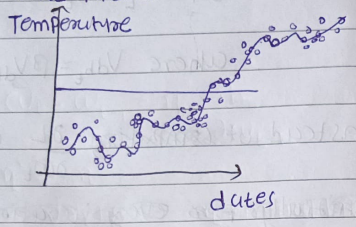
⇒ Due to this oscillation it is hard to reach convergence and it slow down the process of attaining it. To combat this we use momentum.

⇒ Momentum helps us in not taking the direction that does not lead us to convergence.

⇒ EWA is technique to find hidden trend in time series base data

⇒ Recent point has more weight then older one

⇒ at any fix point, with time weight is decreses.



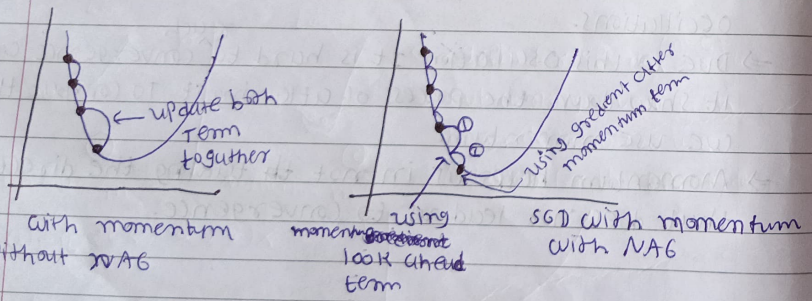
Nesterov Accelurated Gradient (NAG)

⇒ $w_t = w_{t-1} - \eta V_{dw_t}$ depends on two terms.

① history of velocity + ② gradient at that point

⇒ For NAG updatation occure one by one instead of both together.

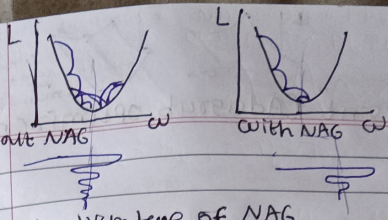
⇒ First ① There is momentum term ① is applied and then we apply gradient term at that point



$$w_{ea} = w_t - \beta V_{t-1}$$

$$V_t = \beta V_{t-1} + \eta \frac{\partial L}{\partial w_{ea}} \left(= (w_t - w_{ea}) + \eta \frac{\partial L}{\partial w_{ea}} \right)$$

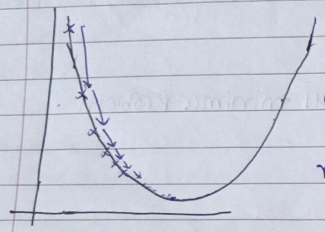
$$w_{t+1} = w_t - V_t$$



Disadvantage of NAG

⇒ some time it stuck with local minima problem.

Adaptive gradient descent (Adagrad optimizers)



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

$$\eta'_t = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

diff for every weight

η = initial LR

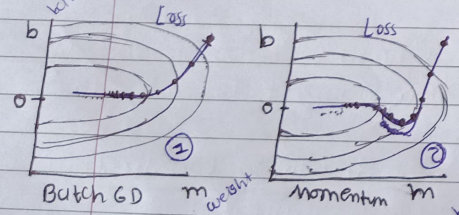
$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_i}\right)^2$$

ε = very small +ve number to avoid $\alpha_t = 0$ effect

⇒ η'_t is decreases as time goes

⇒ after some time α_t very large and η'_t decrease and become very small so $w_t \approx w_{t-1}$ this is disadvantage of Adagrab optimizers.

⇒ Adagrab is word best for sparse data (more zero) and data with different scales (for without normalization)



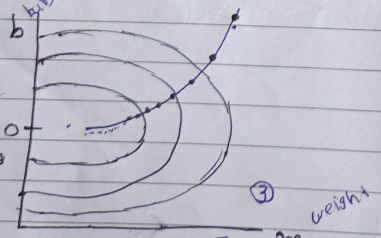
⇐ graphs for sparse (more zeros) data

⇒ not working well with sparsh data so we use Adagrab for this type of data

$$w = w - \eta \frac{\partial L}{\partial w} \leftarrow -2(y-\hat{y})x$$

$$b = b - \eta \frac{\partial L}{\partial b} \leftarrow -2(y-\hat{y}).1$$

For Batch GD we calculate $\frac{\partial L}{\partial w}$ and add for batch



⇒ For sparse data x_i mostly zero and we get $\theta_{old} = \theta_{new}$ and for b There are No zero(I) so b update each time so movement is given in graph ①

⇒ To avoid this we use Variable η

Adadelta and RMSProp

(Root mean square prop)

main ideal behind Adadelta and RMS Prop is same to deal with vanishing LR by taking the weighted avg of gredient step

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

$$\eta'_t = \frac{\eta}{\sqrt{S_{dw} + \varepsilon}}$$

$\beta = 0.9$ or 0.95

$$S_{dw} = \beta S_{dw_{t-1}} + (1-\beta)\left(\frac{\partial L}{\partial w}\right)^2$$ ← Exponetially weighted avg (EWA)

Initially $S_{dw} = 0$

⇒ we restricting to increase S_{dw} to very high value by adding $(1-\beta)$ term

⇒ on Iteration t in mini batch compute $\frac{\partial L}{\partial w}$, $\frac{\partial L}{\partial b}$ on current mini batch

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

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

then we update weight and bias

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

$$w_t = w_{t-1} - \underbrace{\frac{\eta}{\sqrt{\beta S_{dw} + (1-\beta)\left(\frac{\partial L}{\partial w}\right)^2 + \varepsilon}}}_{\eta'_t} \frac{\partial L}{\partial w_{t-1}}$$

same for $b_t = b_{t-1} - \eta'_t \frac{\partial L}{\partial b_{t-1}}$

$\eta'_t = \frac{\eta}{\sqrt{S_{db} + \varepsilon}}$ same as above

⇒ Adagrab is not working on non-convex optimization but RMS works for

RMS has no disadvantage.

Adam optimizer (Adoptive momentum Estimation)

⇒ initially $V_{dw}=0$, $V_{db}=0$, $S_{dw}=0$, $S_{db}=0$

on iteration t

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

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

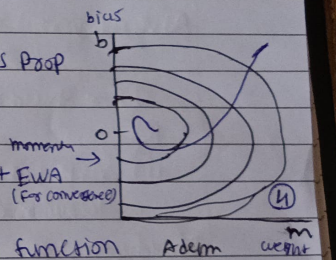
$$V_{db} = \beta_1 V_{db} + (1-\beta_1)\frac{\partial L}{\partial b}$$

} momentum exponential smoothing functions

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

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

} RMS Prop



$$w_t = w_{t-1} - \frac{\eta \cdot V_{dw}}{\sqrt{S_{dw}+\varepsilon}}$$

$$b_t = b_{t-1} - \frac{\eta \cdot V_{db}}{\sqrt{S_{db}+\varepsilon}}$$

combine smoothing function + RMSProp → Adam

Bias correction on momentum: $V_{dw}^{correct} = \frac{V_{dw}}{1-\beta_1^{(t)}}$ ← not power, same for V_{db}

Bias correction on S_{dw}: $S_{dw}^{correct} = \frac{S_{dw}}{1-\beta_2^{(t)}}$ same for S_{db}

Different types of Loss functions

⇒ Loss function : For only one data point $(y-\hat{y})^2$ is Loss fn

⇒ cost function : For batch size $\sum_{i=1}^{t}(y-\hat{y})$ is cost function

⇒ Error function : Loss & cost function called as error fun

i) squared error ~~function~~ Loss

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

Cost function $= J=\sum_{i=1}^{t}\frac{(y-\hat{y})^2}{t}$ ← MSE

mean squared error Loss

Advantages: In form of quadratic eqn

ax^2+bx+c

i) For quadratic eqn has only one global minima

ii) MSE Loss penalize the model for making larger errors by squaring them.

Disadvantage :

i) It is not robust to outliers (unnecessary penalize model)

ii) Absolute error Loss

$L=|y-\hat{y}|$ $J=\sum_{i=1}^{t}\frac{|y-\hat{y}|}{t}$ ← MAE

Modulus operator is mathematically computationally very difficult.

advantages: i) The MAE is robust to outliers compare to MSE

disadvantage: i) may have local minima, not differentiable at 0 (for this need to calculate subgradient)

iii) Huber Loss (combination of MSE & MAE)

$$\text{Loss} = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & , |y-\hat{y}| \leq \delta \leftarrow \text{hyper parameter} \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2 & , \text{otherwise} \end{cases}$$

⇒ In huber Loss If $y-\hat{y}>\delta$ means diff is greater than some value hyper parameter then It is outlier and we avoid unnecessary penalization and we use MAE instead of MSE

iv) Cross entropy loss Logistic Regression

$\text{Loss} = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$ for binary cross entropy

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

v) multiclass cross entropy Loss

$L(x_i, y_i) = -\sum_{j=1}^{c} y_{ij}\log(\hat{y}_{ij})$ ← No of categories

categorical cross entropy

y_{ij} is one hot encoded target vector

o/p multiclass

$y_i = [y_{i1}, y_{i2}, y_{i3}, \ldots y_{ic}]$ ← total num of features

row column

total number of data

$y_{ij} = \begin{cases} 1, & \text{if } i^{th} \text{ element is in class } j \\ 0, & \text{otherwise} \end{cases}$

PAGE NO.:

IIP			OIP		
f_1	f_2	f_3	y_1	y_2	y_3
2	3	4	[1	0	0] ← 3 bits
4	5	6	[0	1	1]
7	8	9	[0	0	1]

$\hat{y}_{ij} \Rightarrow$ Soft max function

$$f(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \leftarrow \text{probability value}$$

Cost function For multiclass Cross entropy

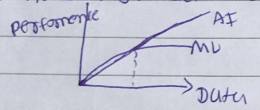
$$= -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{K} y_{ij} \log(\hat{y}_{ij})$$

PAGE NO.:

Deep Learning Vs Machine learning

1) Data dependency
2) hardware dependency
3) training time
4) Feature selection
5) Interpretability

1) Data dependency
AI need high data compare to ML



2) AI need GPU (∵ high computations)
ML does not need GPU It can be runs on CPU

3) DL is very slow but prediction time is very high

4) DL used Representation Learning and extract features automatically while ML doesn't

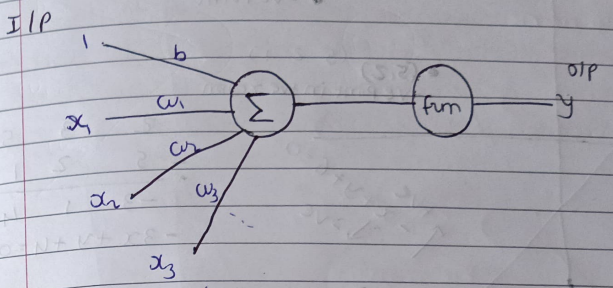
5) AI model is not interpretable
Ex for cat-dog model AI does not ans why It has selelt cat or dog
but ML can explain why by using weights of ~~that~~ features

2015 → google → Tensorflow ← difficult to use
Keras ← use as front-end
user → Keras → Tensorflow
industry use Tensorflow most

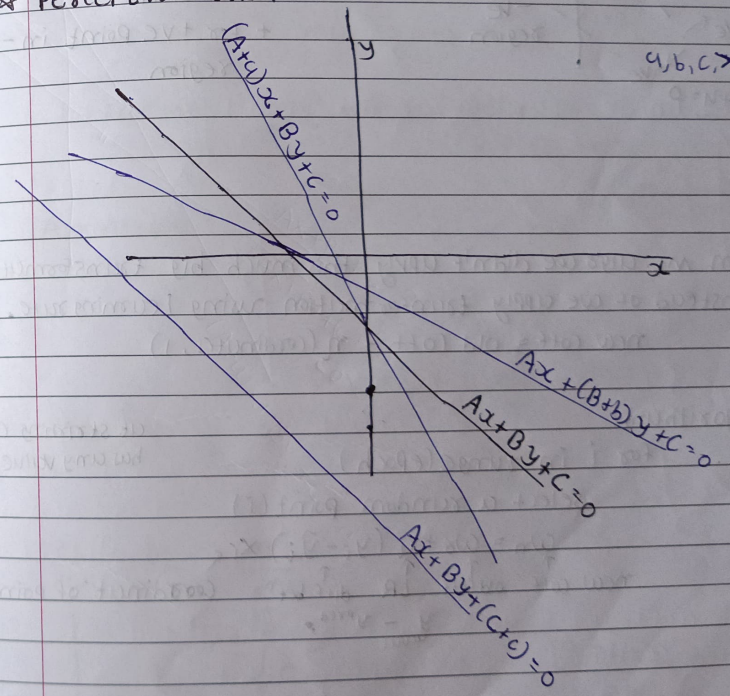
2016 → Face book → Pytorch
AI Researcher uses most
merge with Pytorch and after some time FB made complet framework Pytorch
2018 → caffe2 diploy Pytorch model into sever & run

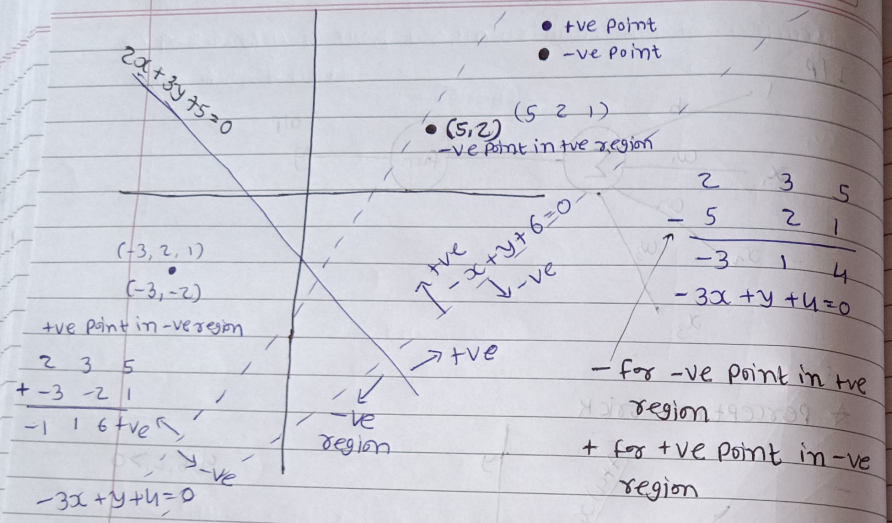
drop down Aplication that convert Nueral network into GUI base
Tensor flow & Pytorch code
Ex Auto AI (google)
Apple (create ML)
Microsoft (Custom Vision AI)

Perceptron



☆ perceptron trick





- for -ve point in +ve region
+ for +ve point in -ve region

⇒ In ML algo we didn't apply this much big transformation instead of we apply transformation using Learning rate.

new coff = old coff - η (cordinates, 1)

Algorithum:

for i in range(epoch)
  select a random point(i)
  $w_n = w_o + \eta (y_i - \hat{y}_i) x_i$
  (new coff, old coff, LR, diff bet^n $y_{actual} - y_{pred}$, coordinat of point)

at starting w_o has any value

⇒ This perceptron trick is not work in all cases this is Juguadu method

⇒ If this trick choose only those points which are correctly classified then line does not move and ~~go~~ gives us wrong result

⇒ Here there are random fun used so every time we get diff line and we can't decide which is give more pricise o/p.

⇒ so we use loss function to know how precise our solution is

~~too~~ loss function for Perceptron

$$\text{loss} = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) = L(w_1, w_2, b)$$

$$L(y_i, f(x_i)) = \max(0, -y_i f(x_i))$$

(y_i: actual op; $f(x_i)$: pred op $w_1x_1 + w_2x_2 + b$)

$$\text{Loss} = \frac{1}{n}\sum_{i=1}^{n} \max(0, -y_i f(x_i))$$

$$L = \underset{w_1, w_2, b}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} \max(0, -y_i f(x_i))$$

where $f(x_i) = w_1 x_{i1} + w_2 x_{i2} + b_i$

(x_{ij}: row (data), column (features))

Algorithum:

for in epochs:

$$w_1 = w_1 + \eta \frac{\partial L}{\partial w_1}$$

$$w_2 = w_2 + \eta \frac{\partial L}{\partial w_2}$$

$$b = b + \eta \frac{\partial L}{\partial b}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f(x_i)} \frac{\partial f(x_i)}{\partial w_1}$$

$$\frac{\partial L}{\partial f(x_i)} = \begin{cases} 0, & y f(x_i) \geq 0 \\ -y_i, & y f(x_i) < 0 \end{cases}$$

$$\frac{\partial f(x_i)}{\partial w_1} = x_{i1}$$

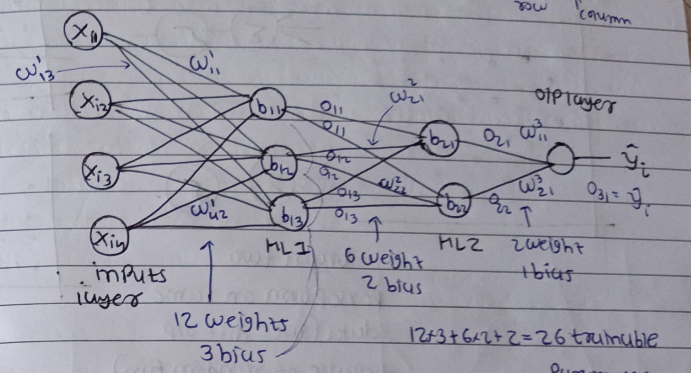
$$\frac{\partial L}{\partial w_1} = \begin{cases} 0, & y_i f(x_i) \geq 0 \\ -y_i x_{i1}, & y_i f(x_i) < 0 \end{cases}$$

same for w_2 & b

$$\frac{\partial L}{\partial \alpha} = \begin{cases} 0, & y_i f(x_i) \geq 0 \\ -y_i \frac{\partial f(x_i)}{\partial \alpha}, & " \ < 0 \end{cases}$$

⇒ perseptron is used for linear data only for nonlinear it doesn't work

Multilayer Perseptron (MLP) Notation

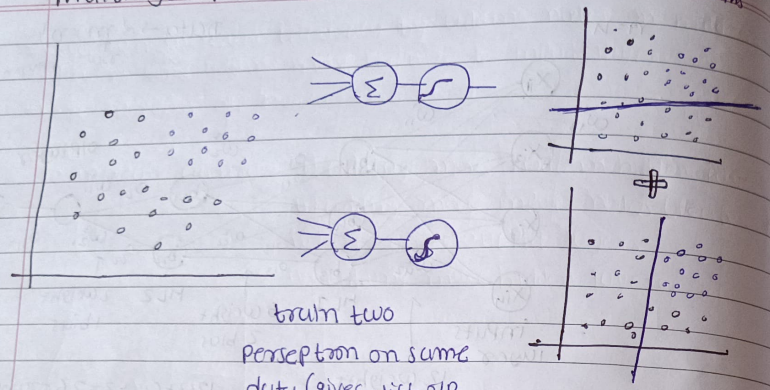


12+3+6+2+2 = 26 trainable Parameters

⇒ bias b_{ij} (layer, node), output o_{ij}, input x_{ij} (j^{th} feature, row of i^{th} data (feature of i^{th} data))

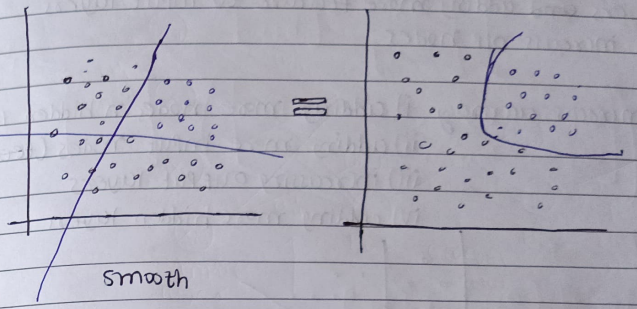
⇒ weight w^k_{ij} (k^{th} input layer; output of i^{th} node in pre layer; input for j^{th} node in k^{th} layer)

Multilayer perceptron intuition



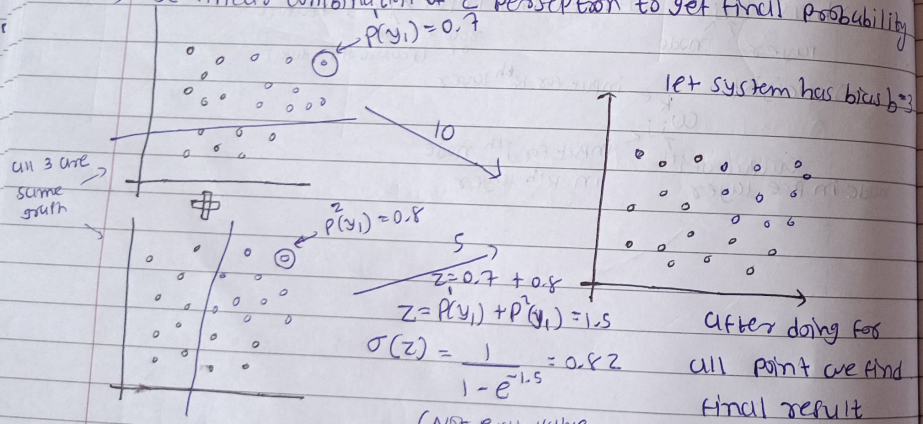
all three are same graphs

train two perceptron on same data (gives diff o/p because of random fun)



smooth

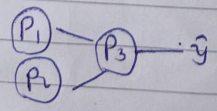
⇒ use linear combination of 2 perceptron to get final probability



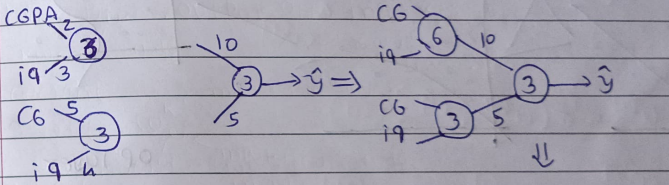
⇒ If perceptron 1's op is dominating then weighted addition is used and we can also use bias

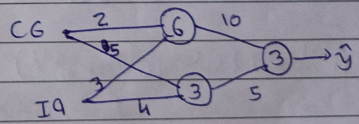
$z = (0.7)(10) + (0.8)(5) + 3$

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

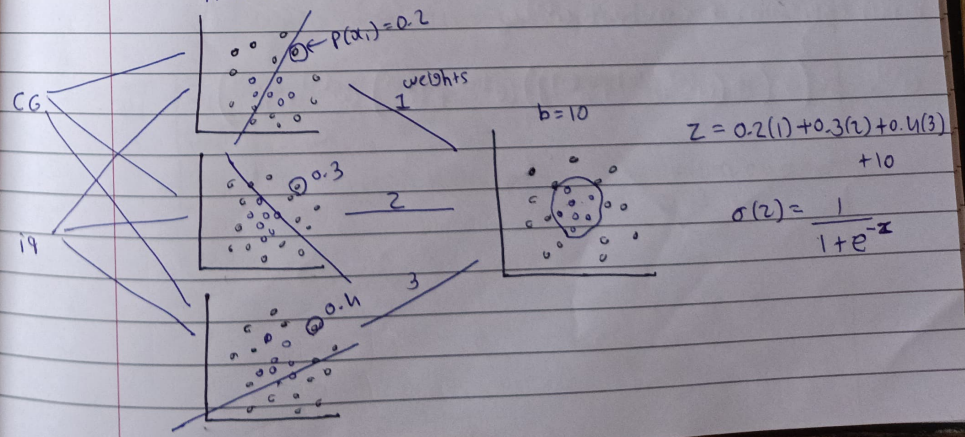


Ex





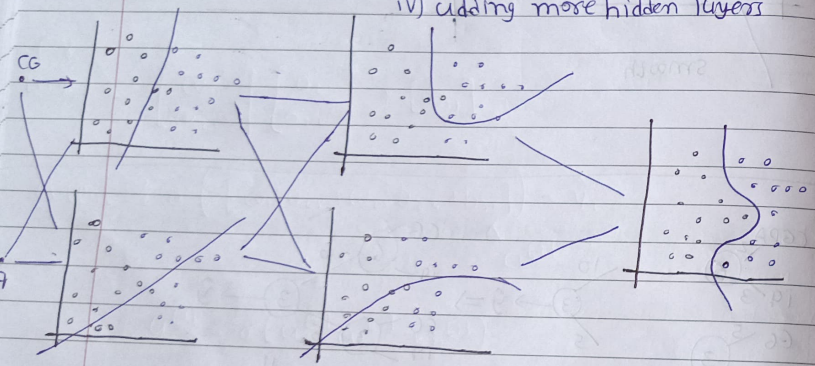
*All graphs are same



⇒ we can increase accuracy by adding nodes on hidden layers OR adding more feature to input layers

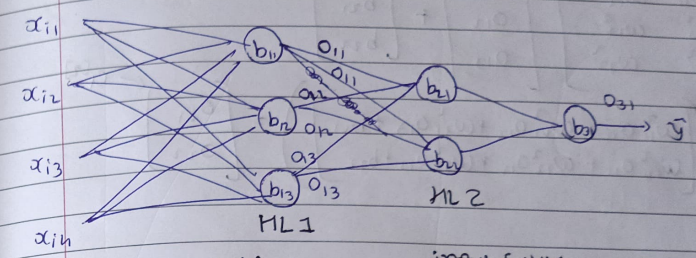
⇒ OR increase o/p nodes

To increase accuracy i) adding more nodes in hidden layer
ii) adding more input nodes (features)
iii) increasing output layers
iv) adding more hidden layers



HL 1 HL 2 OP layer

Forward Propogation



For layer 1

weights (transpose) × input for HL1 + bias of HL1

$$\begin{bmatrix} w'_{11} & w'_{12} & w'_{13} \\ w'_{21} & w'_{22} & w'_{23} \\ w'_{31} & w'_{32} & w'_{33} \\ w'_{41} & w'_{42} & w'_{43} \end{bmatrix}^T \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ x_{i4} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix}$$

$4\times3 \xrightarrow{T} 3\times4$; 4×1 ; 3×1

$$= \begin{bmatrix} w'_{11}x_{i1} + w'_{21}x_{i2} + w'_{31}x_{i3} + w'_{41}x_{i4} \\ w'_{12}x_{i1} + w'_{22}x_{i2} + w'_{32}x_{i3} + w'_{42}x_{i4} \\ w'_{13}x_{i1} + w'_{23}x_{i2} + w'_{33}x_{i3} + w'_{43}x_{i4} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix}$$

Output $= \sigma\left(\begin{bmatrix} w'_{11}x_{i1} + w'_{21}x_{i2} + w'_{31}x_{i3} + w'_{41}x_{i4} + b_{11} \\ w'_{12}x_{i1} + w'_{22}x_{i2} + w'_{32}x_{i3} + w'_{42}x_{i4} + b_{12} \\ w'_{13}x_{i1} + w'_{23}x_{i2} + w'_{33}x_{i3} + w'_{43}x_{i4} + b_{13} \end{bmatrix} \right)$

$$= \begin{bmatrix} o_{11} \\ o_{12} \\ o_{13} \end{bmatrix}$$ ← activation of layer 1

$a^{[1]}$

input layer $\begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix}$ is activation of zeroth layer $a^{[0]}$

PAGE NO.:

weights Input for HL2 bias of HL2

For Layer 2

$$\begin{bmatrix} w_{11}^{2} & w_{12}^{2} \\ w_{21}^{2} & w_{22}^{2} \\ w_{31}^{2} & w_{32}^{2} \end{bmatrix} \begin{bmatrix} o_{11} \\ o_{12} \\ o_{13} \end{bmatrix} + \begin{bmatrix} b_{21} \\ b_{22} \end{bmatrix}$$

op $\Rightarrow$

$$\sigma\left(\begin{bmatrix} w_{11}^{2} o_{11} + w_{21}^{2} o_{12} + w_{31}^{2} o_{13} + b_{21} \\ w_{12}^{2} o_{11} + w_{22}^{2} o_{12} + w_{32}^{2} o_{13} + b_{22} \end{bmatrix}\right) = \begin{bmatrix} o_{21} \\ o_{22} \end{bmatrix} \leftarrow a^{[2]}$$

For Layer 3

$$\begin{bmatrix} w_{11}^{3} \\ w_{21}^{3} \end{bmatrix} \begin{bmatrix} o_{21} \\ o_{22} \end{bmatrix} + [b_{31}]$$

output $\Rightarrow \sigma\left([w_{11}^{3} o_{21} + w_{21}^{3} o_{22} + b_{31}]\right) = \hat{y}_i$

$\uparrow a^{[3]}$

$$a^{[1]} = \sigma(a^{[0]} w^{[1]} + b^{[1]})$$

$$a^{[2]} = \sigma(a^{[1]} w^{[2]} + b^{[2]})$$

$$a^{[3]} = \sigma(a^{[2]} w^{[3]} + b^{[3]})$$

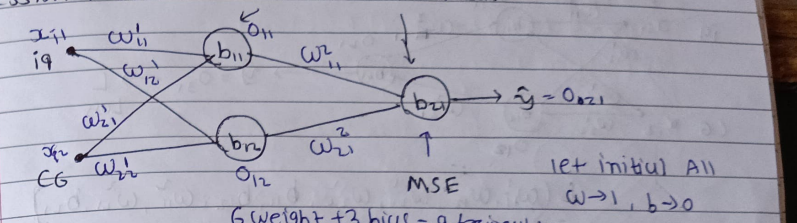
Final expression for output $a^{[3]}$

$$a^{[3]} = \sigma\left(\sigma\left(\sigma(a^{[0]} w^{[1]} + b^{[1]})\right) w^{[2]} + b^{[2]}\right) w^{[3]} + b^{[3]}\Big)$$

PAGE NO.:

Back Propogation

For regression Problem : activation fun = linear



let initial All $w \to 1$, $b \to 0$

6 weight + 3 bias = 9 trainable parameters

Steps

loop { epochs × Row times

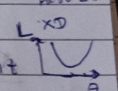
i) select point (row)

ii) predict op (forward prop) (Dot Product)

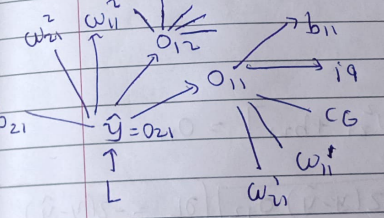
iii) choose loss fun (MSE here)

iv) update weights and biases using gradient Desent

Here 2D



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



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

same for w_{21}^2 & b

$o_{21} = w_{11}^2 o_{11} + w_{21}^2 o_{12} + b_{21}$

$o_{11} = w_{11}^1 x_{i1} + w_{21}^1 x_{i2} + b_{11}$

we have to calculate $\frac{\partial L}{\partial \theta}$ for all 9 parameters

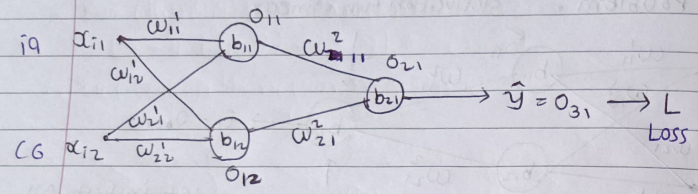
$\theta = w_{11}^2, w_{21}^2, w_{11}^1, w_{21}^1, w_{12}^1, w_{22}^1, b_{21}, b_{11}, b_{12}$

}

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

$\hat{y} = w_{11}^2 [w_{11}^1 x_{i1} + w_{21}^1 x_{i2} + b_{11}] + w_{21}^2 [w_{12}^1 x_{i1} + w_{22}^1 x_{i2} + b_{12}] + b_{12}$

only x_{i1}, x_{i2} is const other 9 are variable



$L(w_{11}^1, w_{12}^1, w_{21}^1, w_{22}^1, b_{11}, b_{12}, w_{11}^2, w_{21}^2, b_{21})$

Now we need

$\frac{\partial L}{\partial w_{11}^2}, \frac{\partial L}{\partial w_{21}^2}, \frac{\partial L}{\partial b_{21}} \mid \frac{\partial L}{\partial w_{11}^1}, \frac{\partial L}{\partial w_{21}^1}, \frac{\partial L}{\partial b_{11}} \mid \frac{\partial L}{\partial w_{12}^1}, \frac{\partial L}{\partial w_{22}^1}, \frac{\partial L}{\partial b_{12}}$

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

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

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

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

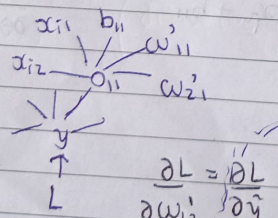
$\frac{\partial L}{\partial w_{21}} = -2(y-\hat{y}) o_{12}$

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

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

$\frac{\partial L}{\partial w_{21}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{11}} \frac{\partial o_{11}}{\partial w_{21}^1}$

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



② $\frac{\partial L}{\partial w_{12}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{12}} \frac{\partial o_{12}}{\partial w_{12}^1}$

$\frac{\partial L}{\partial w_{21}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{12}} \frac{\partial o_{12}}{\partial w_{22}^1}$

$\frac{\partial L}{\partial b_{12}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{12}} \frac{\partial o_{12}}{\partial b_{12}}$

we have calculated $\frac{\partial L}{\partial \hat{y}}$ already

① $\frac{\partial \hat{y}}{\partial o_{11}} = \frac{\partial}{\partial o_{11}} [w_{11}^2 o_{11} + w_{21}^2 o_{21} + b_{21}] = w_{11}^2$

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

o_{11} $\frac{\partial o_{11}}{\partial w_{11}^1} = \frac{\partial}{\partial w_{11}^1} [x_{i1} w_{11}^1 + x_{i2} w_{21}^1 + b_{11}]$

$\frac{\partial o_{11}}{\partial w_{11}^1} = x_{i1}$, $\frac{\partial o_{11}}{\partial w_{21}^1} = x_{i2}$, $\frac{\partial o_{11}}{\partial b_{11}} = 1$

o_{12} $\frac{\partial o_{12}}{\partial w_{12}^1} = \frac{\partial}{\partial w_{12}^1} [w_{12}^1 x_{i1} + w_{22}^1 x_{i2} + b_{12}]$

$\frac{\partial o_{12}}{\partial w_{12}^1} = x_{i1}$, $\frac{\partial o_{12}}{\partial w_{22}^1} = x_{i2}$, $\frac{\partial o_{12}}{\partial b_{12}} = 1$

Finally we get

$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial \hat{y}} -2(y-\hat{y}) w_{11}^2 x_{i1}$ same for w_{21}^1 & b_{11}

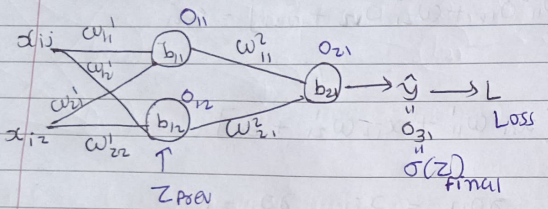
$\frac{\partial L}{\partial w_{12}^1} = -2(y-\hat{y}) w_{21}^2 x_{i1}$ same for w_{22}^1 & o_{12}

⇒ Here we calculate minima for 9 diff parameters

For classification problem :

$L = -y \log(\hat{y}) - (1-y)\log(1-\hat{y})$

activation function is sigmoid



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

last node's z

$z_{final} = w_{11}^2 o_{11} + w_{21}^2 o_{11} + b_{21}$

$\hat{y}$ → L ; w_{11}^2

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}\left[-y\log(\hat{y}) - (1-y)\log(1-\hat{y})\right]$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{-(y-\hat{y})}{\hat{y}(1-\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial(\sigma(z))}{\partial z} = \sigma(1-\sigma)$$

(derivative of σ is $\sigma(1-\sigma)$)

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$$

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

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

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

already calculated: $\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_f} = -(y-\hat{y})$

$$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_f} \frac{\partial z_f}{\partial o_{11}} \frac{\partial o_{11}}{\partial z_{prev}} \frac{\partial z_{prev}}{\partial w_{11}^1}$$

$L \to \hat{y} \to z_{final} \to o_{11} \to z_{prev} \to w_{11}^1$

$$\frac{\partial L}{\partial w_{11}^1} = \underbrace{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_f}}_{-(y-\hat{y})} \underbrace{\frac{\partial z_f}{\partial o_{11}}}_{w_{11}^2} \underbrace{\frac{\partial o_{11}}{\partial z_{prev}}}_{o_{11}(1-o_{11})} \underbrace{\frac{\partial z_{prev}}{\partial w_{11}^1}}_{x_{i1}}$$

$z_{prev} = w_{11}^1 x_{i1} + w_{21}^1 x_{i2} + b_{11}$

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

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

$$\boxed{\frac{\partial L}{\partial b_{11}} = -(y-\hat{y})\, w_{11}^2\, o_{11}(1-o_{11})}$$

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

$L \to \hat{y} \to z_f \to o_{12} \to z_p \to w_{12}^1$

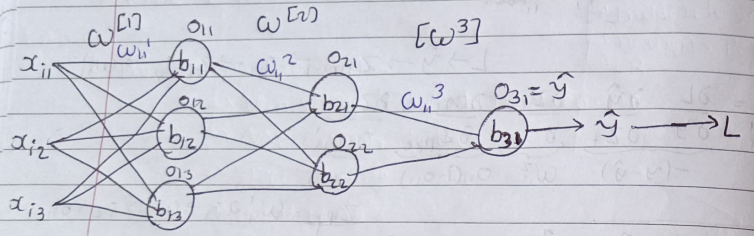
$$\boxed{\frac{\partial L}{\partial w_{12}^1} = -(y-\hat{y}) \cdot w_{21}^2\, o_{12}(1-o_{12}) \cdot x_{i1}}$$

$$\boxed{\frac{\partial L}{\partial w_{22}^1} = -(y-\hat{y})\, w_{21}^2\, o_{12}(1-o_{12})\, x_{i2}}$$

$$\boxed{\frac{\partial L}{\partial b_{12}} = -(y-\hat{y})\, w_{21}^2\, o_{12}(1-o_{12})}$$

multi layer perceptron

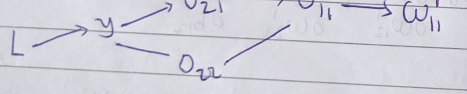
For 2 layers ⇒ MLP memoization



$$\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_{11}^3} \qquad L \to \hat{y} \to w_{11}^3$$

$$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{21}} \frac{\partial o_{21}}{\partial w_{11}^2} \qquad L \to \hat{y} \to o_{21} \to w_{11}^2$$

For $\frac{\partial L}{\partial w_{11}^1}$



$$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial \hat{y}} \left[\frac{\partial \hat{y}}{\partial o_{21}} \frac{\partial o_{21}}{\partial o_{11}} \frac{\partial o_{11}}{\partial w_{11}^1} + \frac{\partial \hat{y}}{\partial o_{22}} \frac{\partial o_{22}}{\partial o_{11}} \frac{\partial o_{11}}{\partial w_{11}^1} \right]$$

How to improve a neural network

1) Fine tuning ~~a~~ neural network → batch size, Optimizer, Activation, Hyper parameter → epoch, hidden layer, neurons per layer, LR

2) By solving Problems:
- → Vanishing / Exploding gradient
- → Not enough data
- → slow traing
- → overfitting

1) Fine tuning Hyperparameter

i) => more HL with few neurons is better then few HL with more neuron (∵ Representation Learning)
=> extend HL untill overfitting occurs

ii) Neuron/layer
=> nodes should be sufficient so It capture primitive Features

iii) Batch size

Smaller (8 to 32)	large (8192) ← depends on GPU Ram
=> give generalize model	=> traing fast
=> better result	=> LR varies small to high (warming up LR)

iv) Epoch value
=> early stopping : is mechanism which is intelligent to understand when weight updation should be stop

2)

i) Vanishing/exploding gradient
- → weigh initialization
- → Activation function
- → Batch normalization
- → gradient clipping, for exploding gradient

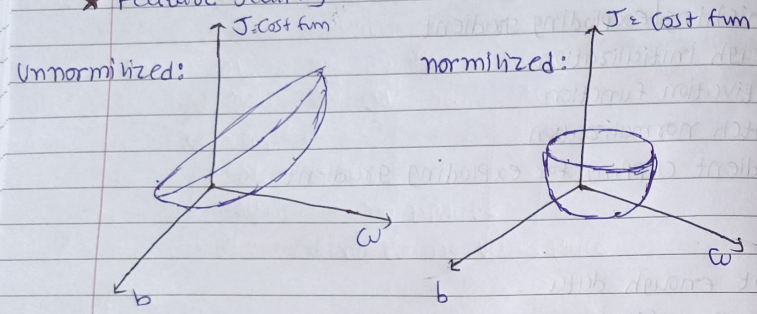
ii) Not enough data
- → use Transfer learning
- → In Transfer learning we use model which made by others in our model
- → unsupervise pretraing

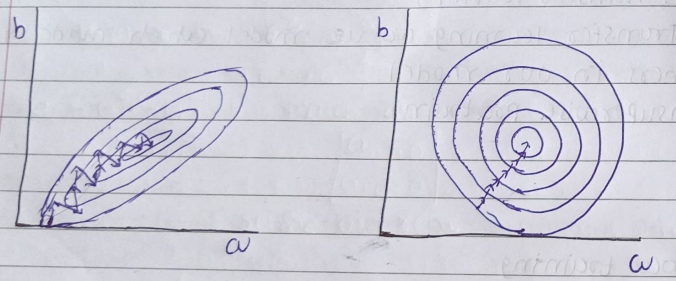
iii) slow training
- → optimizer
- → LR scheduling

iv) overfitting
- → L1, L2 regularization
- → dropouts
- → Reduse complexity
- → increases data
- → early stopping

v) Optimizer
- → Momentum
- → Adagrad
- → RMSProp
- → Adam

★ Feature sculing





★ ways to solve overfitting

→ adding more rows
→ data augmentation

Reducing complexity
↳ drop out
↳ early stopping
↳ Regularization

⇒ Regularization (hyper para)

Lasso L2 → $Loss = L_{old} + \frac{\lambda}{2n} \sum_{i=1}^{k} \|w_i\|^2$ ← Penalty term

Ridge L1 → $Loss = L_{old} + \frac{\lambda}{2n} \sum_{i=1}^{k} \|w_i\|$

$\lambda \geq 0$

λ Very high = under fitting

λ low = overfitting

$$\left(\sum_{i=1}^{k} \|w_i\|^2 = \sum_{l=1}^{L} \sum_{i=1} \sum_{j=1} \|w_{ij}^{l}\|^2 \right)$$

⇒ by adding penalty term we can reduce values of weight and avoid overfitting

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

Here after regularization L become

$$L = L' + \frac{1}{2} \sum_{i=1}^{n} \|w_i\|^2$$

$$\frac{\partial L}{\partial w_{old}} = \frac{\partial L'}{\partial w_{old}} + \frac{1}{2}(2 w_{old})$$

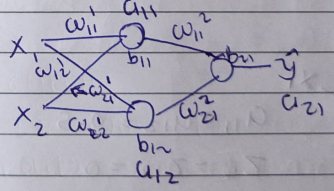
$\|w_i\|^2 = w_1^2 + w_2^2 + \dots + w_n^2$ only one w remain when we integrad others zero

$$w_{new} = w_{old} - \eta \left(\frac{\partial L'}{\partial w_{old}} + \lambda w_{old} \right)$$

$$w_{new} = (1 - \eta\lambda) w_{old} - \eta \frac{\partial L'}{\partial w_{old}}$$

< 1

★ weight initialization



⇒ wrong weight initialization cause
→ Vanishing GP (gradient problem)
→ Exploding GP
→ slow convergence

case 1 zero initialization $W = 0, b = 0$

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

$z_{11} = w_{11}^1 x_1 + w_{21}^1 x_2 + b_{11} = 0$

$z_{12} = w_{12}^1 x_1 + w_{22}^1 x_2 + b_{12} = 0$

since $z_{11} z_{12} = 0$ so

$$a_{11} = a_{12} = 0$$

$w_{11}^1 = w_{11}^1 - \eta \frac{\partial L}{\partial w_{11}^1}$ $\quad L \to \hat{y} \to a_{11} \to o_{11} \to w_{11}^1$

$$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{11}} \frac{\partial a_{11}}{\partial o_{11}} \frac{\partial o_{11}}{\partial w_{11}^1}$$

$a_{11} = a_{12} = 0$

$\frac{\partial L}{\partial w_{11}^1} = 0$ no update & no traing take place

for tanh

$$\tanh x = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$a_{11} = \frac{a^{z_{11}} - a^{-z_{11}}}{a^{z_{11}} + a^{-z_{11}}} \quad \text{since } z_{11} = 0$$

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

same as relu

for sigmoid

$$\sigma = \frac{1}{1+e^{-z}}$$

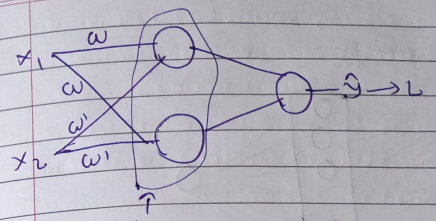
$\boxed{a_{11} = a_{12} = 0.5}$

$$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{11}} \frac{\partial a_{11}}{\partial z_{11}} \frac{\partial z_{11}}{\partial w_{11}^1} \leftarrow x_1 \qquad a_{11} = a_{12} = 0.5, \quad z_{11} = z_{12} = 0$$

$$\frac{\partial L}{\partial w_{12}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{12}} \frac{\partial a_{12}}{\partial z_{12}} \frac{\partial z_{12}}{\partial w_{12}^1} \leftarrow x_1$$

$$\frac{\partial L}{\partial w_{21}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{11}} \frac{\partial a_{11}}{\partial z_{11}} \frac{\partial z_{11}}{\partial w_{21}^1} \leftarrow x_2$$

$$\frac{\partial L}{\partial w_{22}^1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{12}} \frac{\partial a_{12}}{\partial z_{12}} \frac{\partial z_{12}}{\partial w_{22}^1} \leftarrow x_2$$



behave as 1 single neuron

(If there are 1000 neurons then also that act as single)

Case 2 Non zero constent value to W & b $\quad W = b = \text{const}$

For ReLU, tanh, sigmoid

$a_{11} = \max(0, z_{11})$ or $\tanh z_{11}$ or $\frac{1}{1+e^{-z_{11}}}$

$z_{11} = w_{11}^1 x_1 + w_{21}^1 x_2 + b_{11} \neq 0$

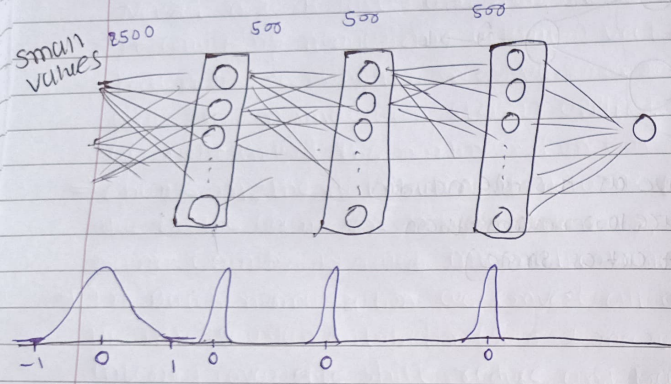
$z_{12} = w_{12}^1 x_1 + w_{22}^1 x_2 + b_{12}$

since $w_{12}^1 = w_{11}^1 \neq w_{21}^1 = w_{22}^1$ & $b_{11} = b_{12}$

so $\boxed{z_{11} = z_{12}}$ $\boxed{a_{11} = a_{12}}$

$\Rightarrow$ act same as sigmoid with zero initialize

Case 3 Random initialization



let bias = 0, np.random.randn(500,500) × 0.01 (0 to 1) × 0.01

and since weights are very small and x_i (input) $\Rightarrow -1 < x_i < 1$ so

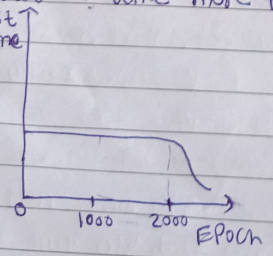
$Z = \sum w_i x_i$ is small numer

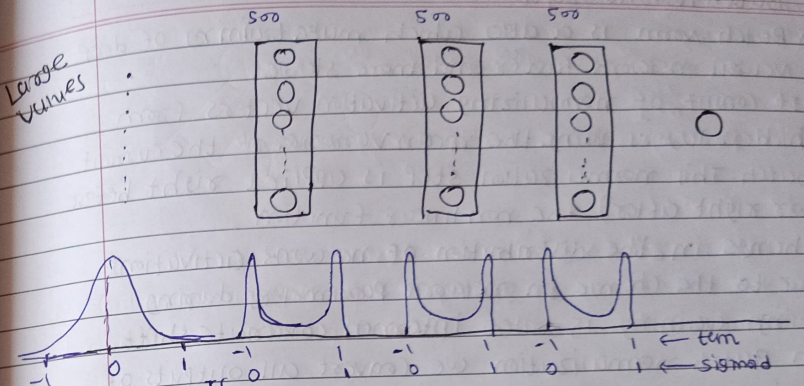
and activation of all node very close to zero

For activation fun = tanh x, singmoid (For deep network), ReLU (no vanishing gradient but training is very slow)

activation is close to zero so during back propogation gradient is very small so $w_{old} \approx w_{new}$ and vanishing gradient occurs

⇒ In case of ReLu due to small gradient vanishing not occure but it take more time and converge at very high epoch



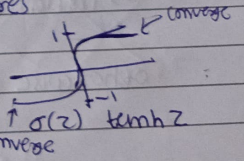


If $-1 < w_i, x_i < 1$

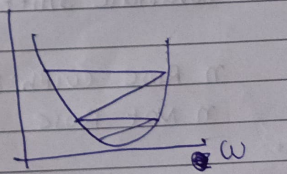
then $Z = 100 \leq \sum w_i x_i \leq 500$

np.random.randn(500,500) × 1 (0 to 1) × 1

⇒ due to high value of Z for activation fun like sigmoid and tanh x value converges so traing will be slow or vanishing gradient can be also occures



⇒ For relu If Z is high then activation function also high due to this $w = w - \eta \frac{\partial L}{\partial w}$ very high and we get unstable gradient



due to high activation fumction $\frac{\partial L}{\partial w}$ high and we get this graph

⇒ to avoid this kind of Problem use uniform distribution, xavier/Gort and He init

Batch Normalization

⇒ Batch Norm is a algo which make training of deep Neural network faster and more stable

⇒ It consist of normalizing activation vectors from hidden layers using the mean variance of the current batch. This normalization step is applied right before (or right after) the non linear function.

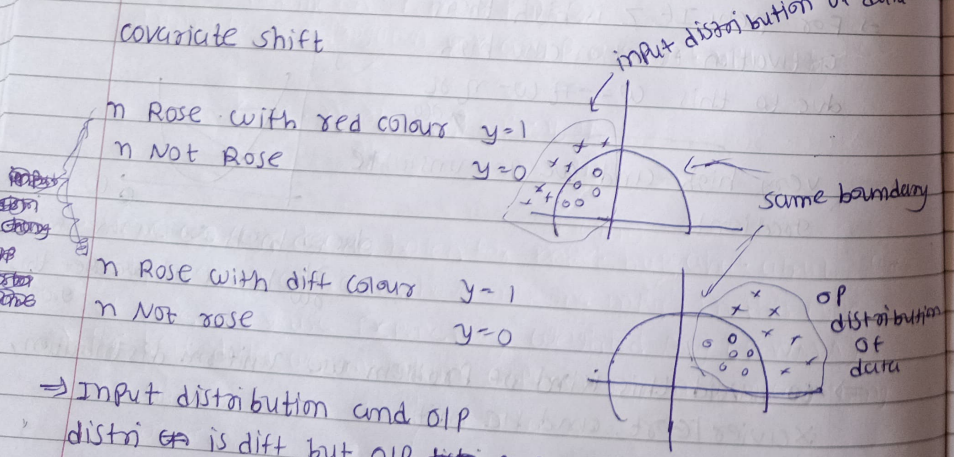
⇒ Change in the distribution of network activations due to the change in network parameters during ~~training~~ training is called internal covariate shift.

⇒ In Batch normalization we convert all outputs of HL into gausian distribution (so $\sigma=1$, mean$=0$) so internal covariate shift reduce and training will be fast and stable

⇒ If we dont use Batch Norm then
- → LR should be low
- → initialization of Parameters need to done carefully (like weights)

otherwise train is very unstable. so we use Batch Norm

covariate shift



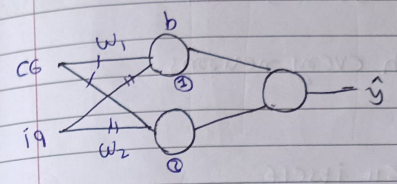
⇒ Input distribution and OIP distri ~~of~~ is diff but OIP ~~distri are same~~ and input relation are same (given by curve ◯)

⇒ IS is called covariate shift and we need to train

⇒ Batch Norm is applied with ⇆ Mini batch GD

⇒ applied Layer by Layer (can applied on any numbers of layers)

our model again.



$z_{11} = w_1 CG + w_2 ig + b$

$g(z_{11}) = a_{11}$ For batch Norm 2 ways

1) $z_{11} \to z_{11}^{\sim} \to g(z_{11}^{\sim}) \Rightarrow a_{11}$

2) $z_{11} \to g(z_{11}) \to a_{11} \to a_{11}^{\sim}$

let batch$=4$

$(4,2) \cdot (2,2) = (4,2)$

input, weight, bias

$(4,2) + (1,2)$

we input 4 rows so get 4 activation on ① and 4 on ②

For ① neuron $\mu_B = \frac{1}{m} \sum_{i=1}^{m} z_{11}^{i}$ ← activation for i^{th} input row, $m=4$

Here batch$=4$ So 4 inputs and 4 op are there for ①

$\sigma_B = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (z_{11}^{i} - \mu_B)}$ for every activation $z_{11}^{i} = \frac{z_{11}^{i} - \mu_B}{\sigma_B + \varepsilon}$ ← +ve small to avoid zero's effect

For ② neuron $\mu_B = \frac{1}{m} \sum_{i=1}^{m} z_{12}^{i}$, $m=4$

$\sigma_B = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (z_{12}^{i} - \mu_B)}$

all $0 < z_{11}^{i} < 1$

$i = 1, 2, \ldots$, batch size

$i = 1, 2, 3, 4$

⇒ Here we done with $z_{11} \to z_{11}^{\sim}$

$\gamma z_{11}^{\sim} + \beta$ ← scale and shift by γ, β

γ, β are learnable parameter (intial value of $\gamma=1$, $\beta=0$ in ~~Keras~~ Keras)

every neuron has its own γ & β parameter

$\gamma Z_{11}^{N} + \beta$

$Z_{11} \rightarrow Z_{11}^{N} \rightarrow Z_{11}^{BN} \rightarrow g(Z_{11}^{BN}) \rightarrow a_{11}$

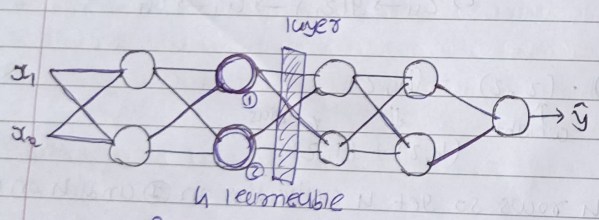
($\mu=0$, $\sigma=1$)

⇒ This process is done with every neurons

If $\gamma = \sigma + \varepsilon$, $\beta = \mu$

then $Z_{11}^{N} \rightarrow Z_{11}^{BN} = Z_{11}$ itself

we give flexibility by giving γ & β If neural network has no need of Normilization then γ, β are set accordingly



4 learnable Parameters : γ, β for ① & ②

during back propogation γ and β also get updated by

$$\gamma = \gamma - \eta \frac{\partial L}{\partial \gamma}, \quad \beta = \beta - \eta \frac{\partial L}{\partial \beta}$$

⇒ Here we can decide batch for training but what about for test testing where we have only one input In this type of case we use Exponetially weighted Avg (EWMA)

⇒ for test data we maintain exponentially moving avg using of $\mu_{B1}, \mu_{B2}, \mu_{B3} \dots \mu_{Bn}$ for n batch

$\sigma_1, \sigma_2, \sigma_3 \dots \sigma_n$

and use final μ_B and σ for further calculation

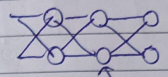
4 parameters $\gamma, \beta, \mu, \sigma$ (learnable / non learnable)

during batch normalization every neuron has 4 Parameters 2 are learnable γ, β and 2 are non learn μ, σ

Advantage: → Stable so we can choose hyper para in large range

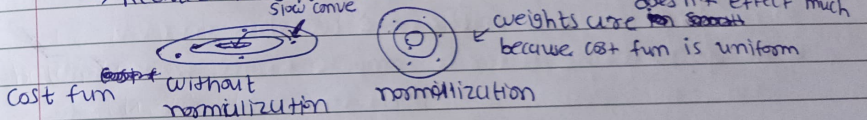
→ training faster (we can set High LR)

→ has regularization effect



μ, σ calculate for every batch which is dependent on batch it self and changing of batch value also change activation also change and due to change on activation system get some randomness and overfitting reduce slightly. It has not very much effect so we need to use regularization also.

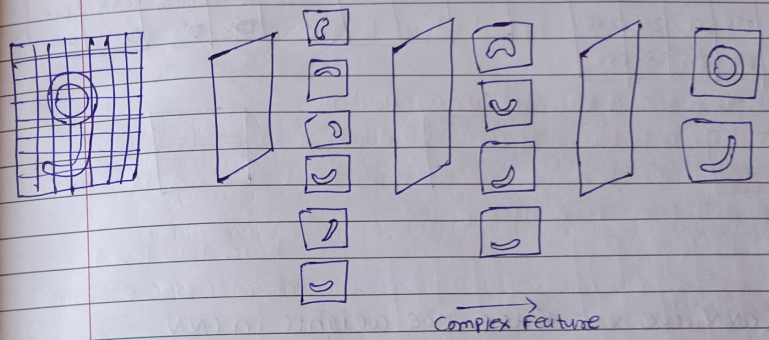
→ Reduce weight initialization impact



ANN = Matrix multiplication
CNN = Convolution

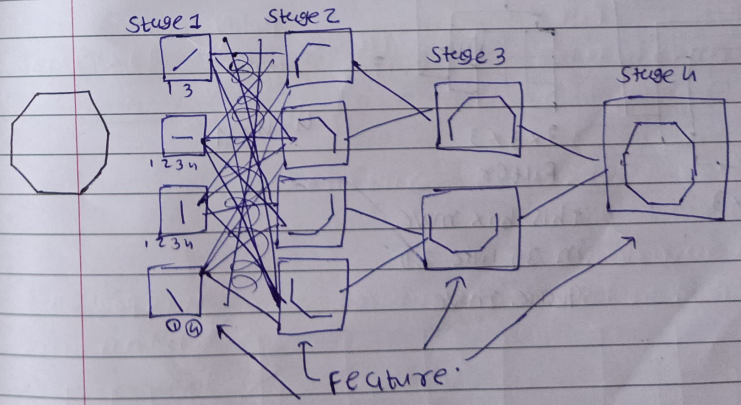
CNN (Convolutional Neural Network)

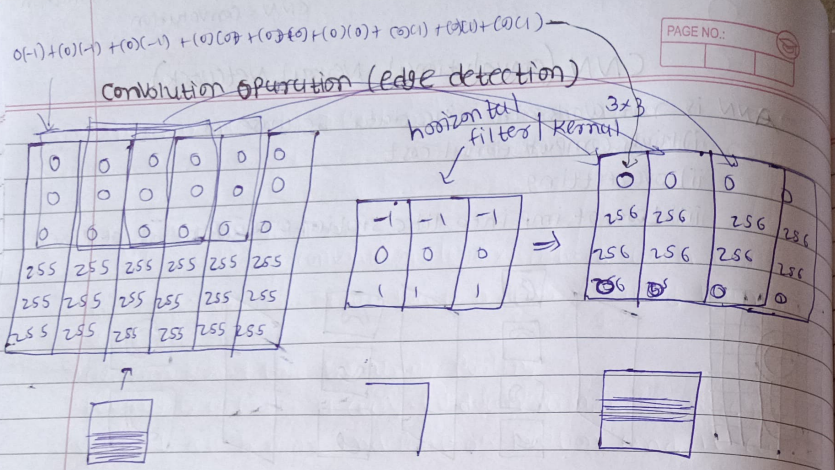
⇒ ANN is not works in image data because of
i) High computational cost
ii) overfitting
iii) Lose of imp info like spatial arrangement of pixels



⇒ CNN extract features from given data as we go ahead
It detect complex features

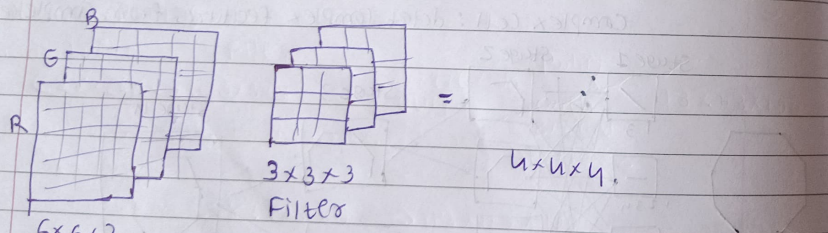
⇒ There are two types of cell
simple cell : detect edges
complex cell : detect complex features from simple cell





⇒ In CNN we use filter like weights in CNN
⇒ values inside filter is compute from back propagation

Working with RGB Images



Padding and Strides

⇒ In Padding we added boundray with value zero
⇒ Strides decide that how a filter is moving (how many rows and column should be skip)
⇒ Generaly values in padding is zero.

Image n×n → (n-f+1)
Filter f×f

$(n+2p-f+1)$ (If there is padding p)

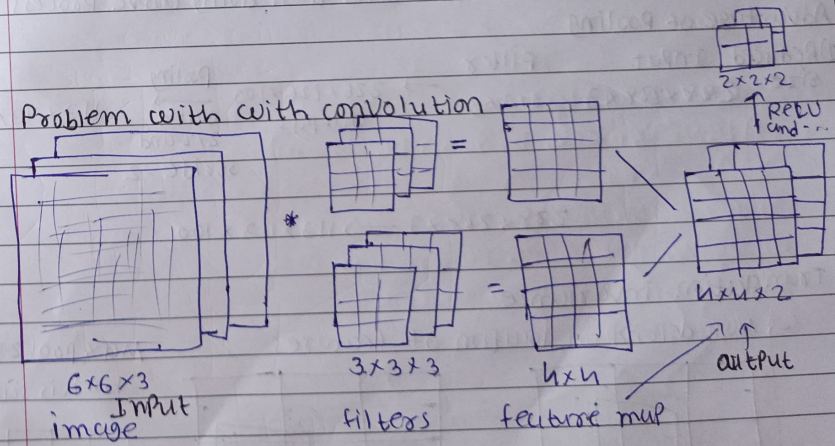
$\left[\frac{n-f}{s}+1\right]$ (If strides is s)

(⌊.⌋ floor operation)

$\left[\frac{n+2p-f}{2}+1\right]$ (If padding p & strides s)

Why strides required
→ If we need high level feature
→ Computing time low

Problem with with convolution



⇒ We need to reduce feature map otherwise size of output is very big. This can be done by increasing strides value

⇒ There is majorly 2 problems 1) memory issue
2) Translation Variance

Translation variance

⇒ After applying convolution layer feature with is find by CNN is dependent on location and is called translation variance

Filter

3×3×3 → 226×226×

228×228×3 ⟹ 113×

variance

on location of featur

3) using (max Pooling) Pooling Enhanced Features
(only for max Pooling)

4) No Need of training

⇒ In Keras There are 3 types of Pooling

1) max pooling
2) Avg pooling ← take avg of numbers this 4 numbers
3) global pooling ← max or avg of whole matrix

global max | global avg

vantage of Pooling

⇒ can't be we in image segmentation task
⇒ loss lose information 4×4 → 2×2
lose 75% information

⇒ There is majorly 2 Problems 1) memory issue
2) Translation Variance

Translation Variance

⇒ After applying convolution layer feature wich is find by CNN is dependent on location this is called translation variance

⇒ we can avoid this by using Pooling

Pooling is actually a way to down sample feature map so features become indepent of location and it called Translation invariance

Ex

3	1	1	3
2	5	0	2
1	4	2	1
4	7	2	4

after ReLU →

5	3
7	4

here low level details are eleminated

Type : max pooling
size = 2×2
Stride = 2

⇒ In this Process ~~first~~ first we apply ReLU on output feature map and then doing above process

Advantage of pooling

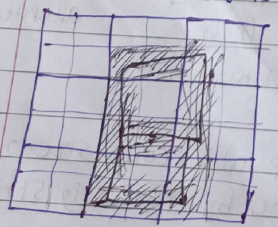
1) Reduced Input size

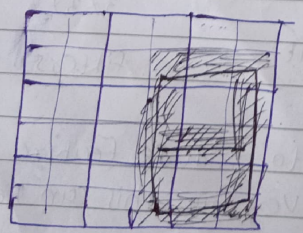
228×228×3 → Filter 3×3×3 → 226×226×100 output → Pooling with 2×2 and stride = 2 → 113×113×100

228×228×3 ⟹ 113×113×100

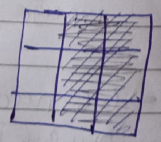
2) Translation invariance

⇒ Not depend on location of features





Max pool2D for both is same



3) using (max pooling) Pooling Enhanced Features (only for max pooling)

4) No Need of training

⇒ In Keras There are 3 types of pooling

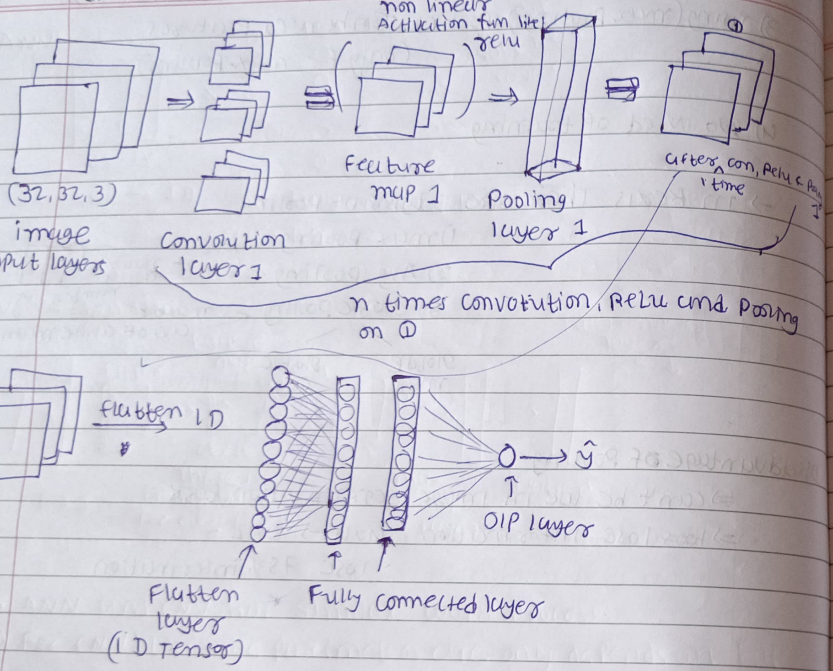
1) max pooling
2) Avg pooling ← take avg of numbers → this is number
3) Global pooling ← max or avg of whole matrix
   - Global max
   - Global avg

Disadvantage of Pooling

⇒ can't be we in image segmentation task

⇒ ~~loss~~ lose information 4×4 → 2×2
lose 75% information

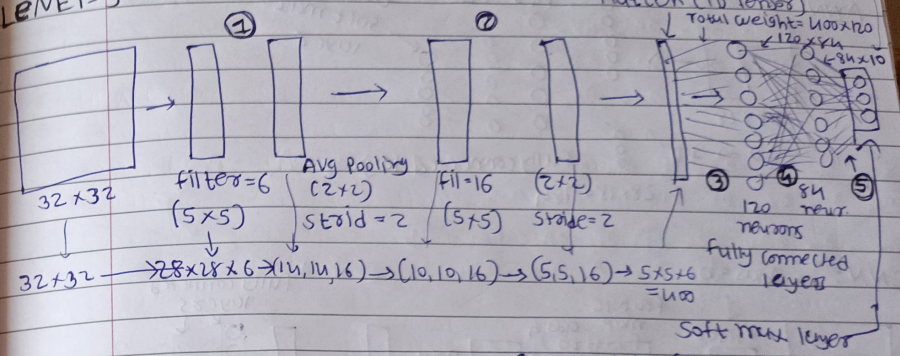
CNN Architecture



CNN Architecture

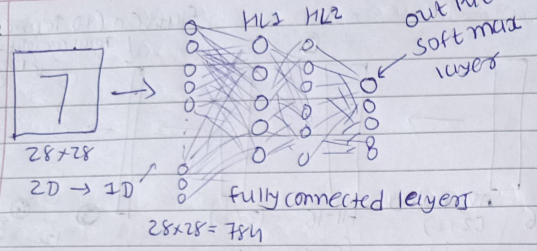
1) LeNET-5 made by Yann Le Cunn
2) Alex NET
3) GoogleLeNET
4) VggNET
5) ResNTE
6) Inception

LeNET-5



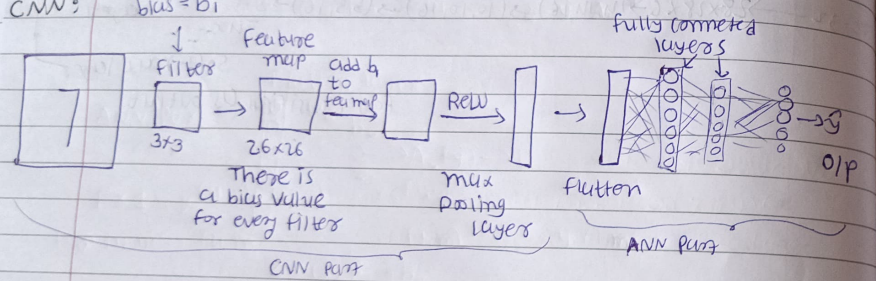
CNN & ANN

ANN:



28×28 = 784

CNN:



⇒ ANN and CNN has similar work Process.

⇒ For CNN trainable parameters are only depends on filter size and number of filters

For Ex (3, 3, 3) 50 filter

Input (1080, 1080, 3)

learnable parameter is = ((3×3×3)×50) + 50 (every filter have bias value)

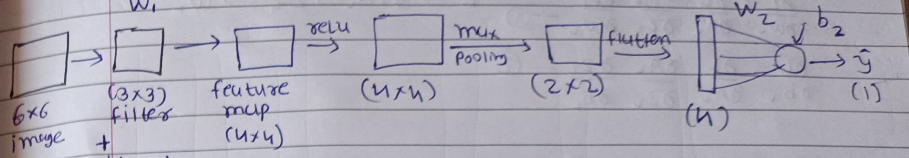
= (27)(50) + 50

= 1350 + 50 = 1400

⇒ while ANN learnable parameter is depends on size of input learnable parameters are

= 1080 × 1080 × 3

Bake Propagation in CNN



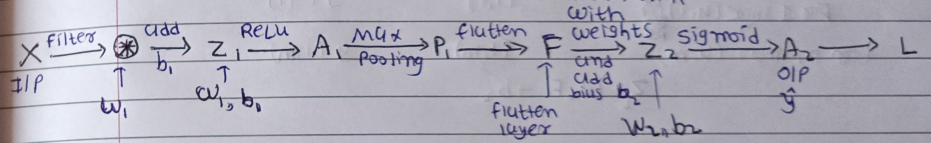
Trainable Parameter

$W_1 = (3,3)$ $W_2 = (1,4)$ = 10 + 5

$b_1 = (1,1)$ $b_2 = (1,1)$ = 15 trainable parameter

⇒ After forward Propagation we will calculate Loss

Logical flow



Forward Propagation

$Z_1 = \text{conv}(X, W_1) + b_1$

$A_1 = \text{Relu}(Z_1)$

$P_1 = \text{max pool}(A_1)$

$F = \text{flatten}(P_1)$

$Z_2 = W_2 F + b_2$

$A_2 = \sigma(Z_2)$

Gradient decent (matrix (1,4))

$W_1 = W_1 - \eta \frac{\partial L}{\partial W_1}$ $W_2 = W_2 - \eta \frac{\partial L}{\partial W_2}$

$b_1 = b_1 - \eta \frac{\partial L}{\partial b_1}$ $b_2 = b_2 - \eta \frac{\partial L}{\partial b_2}$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial A_2}\frac{\partial A_2}{\partial z_2}\frac{\partial z_2}{\partial w_2} \quad -①$$

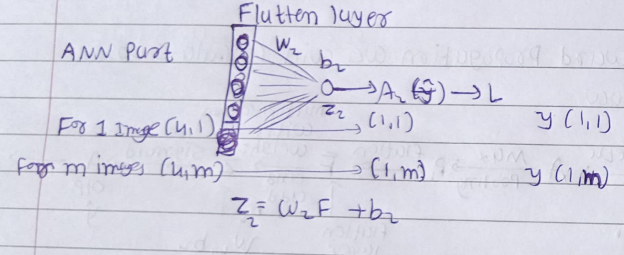
$L \to \hat{y} = A_2 \to z_2 \to w_2$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial A_2}\frac{\partial A_2}{\partial z_2}\frac{\partial z_2}{\partial b_2} \quad -②$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial A_2}\frac{\partial A_2}{\partial z_2}\frac{\partial z_2}{\partial F}\frac{\partial F}{\partial P_1}\frac{\partial P_1}{\partial A_1}\frac{\partial A_1}{\partial z_1}\frac{\partial z_1}{\partial w_1} \quad -③$$

$w_1 \to z_1 \to A_1 \to P_1 \to F \to z_2 \to A_2 = \hat{y} \to L$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial A_2}\frac{\partial A_2}{\partial z_2}\frac{\partial z_2}{\partial F}\frac{\partial F}{\partial P_1}\frac{\partial P_1}{\partial A_1}\frac{\partial A_1}{\partial z_1}\frac{\partial z_1}{\partial b_1} \quad -④$$



$$z_2 = w_2 F + b_2$$

$$A_2 = \sigma(z_2)$$

For ① $\frac{\partial L}{\partial w_2} = \frac{\partial [-y_i \log(a_2) - (1-y_i)\log(1-a_2)]}{\partial a_2}\frac{\partial a_2}{\partial w_2}$

For single image

$$\frac{\partial L}{\partial a_2} = \frac{a_2 - y_i}{a_2(1-a_2)}$$

For single instance we take a_2

$$\frac{\partial A_2}{\partial z_2} = \sigma(z_2)\cdot[1-\sigma(z_2)] = a_2(1-a_2)$$

$$\frac{\partial z_2}{\partial w_2} = F$$

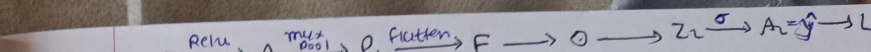
$$\frac{\partial z_2}{\partial b_2} = 1$$

$$\frac{\partial L}{\partial w_2} = (a_2 - y_i)F$$

replace a_2 by A_2 matrix and y_i to y matrix

$$\frac{\partial L}{\partial w_2} = \underset{(1,1)}{(A_2 - y)}\ \underset{(1,1)}{F^T} \quad \text{(1,4)}$$

$$\frac{\partial L}{\partial b_2} = (A_2 - y)$$



For m images

$$\frac{\partial L}{\partial w_2} = \underset{(1,m)}{(A_2 - y)}\ \underset{(m,4)}{F^T} \quad (1,4)$$

$w_2 \to (1,4)$

$(1,4)$

For ③ & ④ $Z_1 = \text{conv}(X, w_1) + b_1$

$A_1 = \text{Relu}(z_1)$

$F = \text{flatten}(P_1)$

$A_2 = \sigma(z_2)$

$L = \frac{1}{m}\sum_{i=1}^{m}[-y_i \log(A_2) - (1-y_i)\cdot\log(1-A_2)]$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial A_2}\frac{\partial A_2}{\partial z_2}\frac{\partial z_2}{\partial F}\frac{\partial F}{\partial P_1}\frac{\partial P_1}{\partial A_1}\frac{\partial A_1}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial A_2}\frac{\partial A_2}{\partial z_2}\frac{\partial z_2}{\partial F}\frac{\partial F}{\partial P_1}\frac{\partial P_1}{\partial A_1}\frac{\partial A_1}{\partial z_1}\frac{\partial z_1}{\partial b_1}$$

already calculated $(A_2 - y)$ — need to calculate

$\frac{\partial z_2}{\partial F} = w_2$, $\frac{\partial F}{\partial P_1}$ no trainable parameter

reshape(P_1, shape)

$$\frac{\partial L}{\partial A_2}\frac{\partial A_2}{\partial z_2}\frac{\partial z_2}{\partial F} = (A_2 - y)\,w_2 \cdot \text{reshape}(P_1 \text{ shape})$$

$$\frac{\partial L}{\partial A_1} = \begin{cases} \frac{\partial L}{\partial P_1}_{xy} & \text{if Amn is the max element} \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial A}{\partial Z_1} = \begin{cases} 1, & \text{If } Z_{1\,x,y} > 0 \\ 0, & \text{if } Z_{1\,x,y} \leq 0 \end{cases}$$

BACK PROPAGATION ON CONVOLUTION:

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial Z_1}\frac{\partial Z_1}{\partial b_1}$$

$X \longrightarrow (*) \longrightarrow Z$

(6×6) $\quad$ $\uparrow$ $\quad$ $\frac{\partial L}{\partial Z_1} \rightarrow (4,4)$

W_1, b_1

$(3,3)$

For easy calculation perspective

3×3 $\quad$ 2×2 $\quad$ 2×2 $\quad$ $\frac{\partial L}{\partial Z_1} = 2\times2$

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \qquad W_1 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \qquad Z_{2\times2} = Z_1 = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix}$$

$X * W_1 + b$

Convoluting X & W_1 and by adding bias we get

$$z_{11} = x_{11}w_{11} + x_{12}w_{12} + x_{21}w_{21} + x_{22}w_{22} + b_1$$
$$z_{12} = x_{12}w_{11} + x_{13}w_{12} + w_{22}w_{21} + x_{23}w_{22} + b_1$$
$$z_{21} = x_{21}w_{11} + x_{22}w_{12} + x_{31}w_{21} + x_{32}w_{22} + b_1$$
$$z_{22} = x_{22}w_{11} + x_{23}w_{12} + x_{32}w_{21} + x_{33}w_{22} + b_1$$

$$\frac{\partial L}{\partial Z_1} = \begin{bmatrix} \frac{\partial L}{\partial z_{11}} & \frac{\partial L}{\partial z_{12}} \\ \frac{\partial L}{\partial z_{21}} & \frac{\partial L}{\partial z_{22}} \end{bmatrix}$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_{11}}\frac{\partial z_{11}}{\partial b_1} + \frac{\partial L}{\partial z_{12}}\frac{\partial z_{12}}{\partial b_1} + \frac{\partial L}{\partial z_{21}}\frac{\partial z_{21}}{\partial b_1} + \frac{\partial L}{\partial z_{22}}\frac{\partial z_{22}}{\partial b_1}$$

($\frac{\partial z_{11}}{\partial b_1} = 1$, $\frac{\partial z_{12}}{\partial b_1} = 1$, $\frac{\partial z_{21}}{\partial b_1} = 1$, $\frac{\partial z_{22}}{\partial b_1} = 1$)

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_{11}} + \frac{\partial L}{\partial z_{12}} + \frac{\partial L}{\partial z_{21}} + \frac{\partial L}{\partial z_{22}} = \text{sum}\left(\frac{\partial L}{\partial Z_1}\right)$$

$$\boxed{\frac{\partial L}{\partial b_1} = \text{sum}\left(\frac{\partial L}{\partial Z_1}\right)} \leftarrow \text{scalar}$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial Z_1}\frac{\partial Z_1}{\partial W_1}$$

2×2

$$\frac{\partial L}{\partial W_1} = \begin{bmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} \end{bmatrix} \qquad \frac{\partial L}{\partial Z_1} = \begin{bmatrix} \frac{\partial L}{\partial z_{11}} & \frac{\partial L}{\partial z_{12}} \\ \frac{\partial L}{\partial z_{21}} & \frac{\partial L}{\partial z_{22}} \end{bmatrix}$$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial z_{11}}\frac{\partial z_{11}}{\partial w_{11}} + \frac{\partial L}{\partial z_{12}}\frac{\partial z_{12}}{\partial w_{11}} + \frac{\partial L}{\partial z_{21}}\frac{\partial z_{21}}{\partial w_{11}} + \frac{\partial L}{\partial z_{22}}\frac{\partial z_{22}}{\partial w_{11}}$$

$$\frac{\partial L}{\partial w_{12}} = w_{11} \longrightarrow w_{21} \quad \text{replace}$$

same for all $\frac{\partial L}{\partial w_{ij}}$ $\quad i=1,2$ $\quad j=1,2$

From eqns of z_{xy} we can find $\frac{\partial z_{xy}}{\partial w_{ij}}$ $\quad x=1,2$. $y=1,2$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial z_{11}}x_{11} + \frac{\partial L}{\partial z_{12}}x_{12} + \frac{\partial L}{\partial z_{21}}x_{21} + \frac{\partial L}{\partial z_{22}}x_{22}$$

$$\frac{\partial L}{\partial w_{12}} = \frac{\partial L}{\partial z_{11}}x_{12} + \frac{\partial L}{\partial z_{12}}x_{13} + \frac{\partial L}{\partial z_{21}}x_{22} + \frac{\partial L}{\partial z_{22}}x_{23}$$

$$\frac{\partial L}{\partial w_{21}} = \frac{\partial L}{\partial z_{11}}x_{21} + \frac{\partial L}{\partial z_{12}}x_{22} + \frac{\partial L}{\partial z_{21}}x_{31} + \frac{\partial L}{\partial z_{22}}x_{32}$$

$$\frac{\partial L}{\partial w_{22}} = \frac{\partial L}{\partial z_{11}}x_{22} + \frac{\partial L}{\partial z_{12}}x_{23} + \frac{\partial L}{\partial z_{21}}x_{32} + \frac{\partial L}{\partial z_{22}}x_{33}$$

we can simplify this into ~~max~~ convolution operation

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \qquad \frac{\partial L}{\partial Z_1} = \begin{bmatrix} \frac{\partial L}{\partial Z_{11}} & \frac{\partial L}{\partial Z_{12}} \\ \frac{\partial L}{\partial Z_{21}} & \frac{\partial L}{\partial Z_{22}} \end{bmatrix}$$

$$\frac{\partial L}{\partial \omega_1} = \text{conv}\left(X, \frac{\partial L}{\partial Z_1}\right)$$

$$\frac{\partial L}{\partial \omega_1} = \text{conv}\left(X, \frac{\partial L}{\partial Z_1}\right)$$

&

$$\frac{\partial L}{\partial b_1} = \text{sum}\left(\frac{\partial L}{\partial Z_1}\right)$$

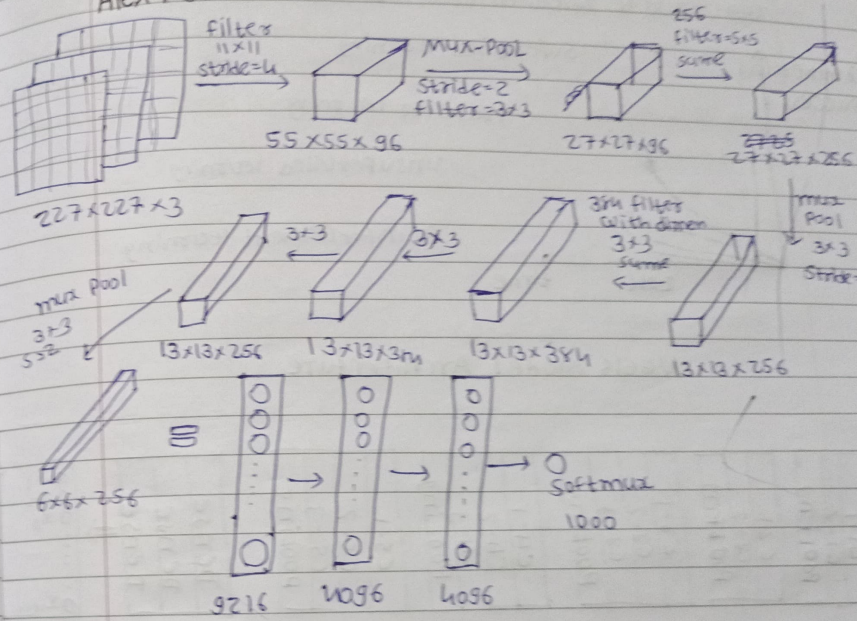
pretrained models in CNN

⇒ why

→ Data hungry

→ Time

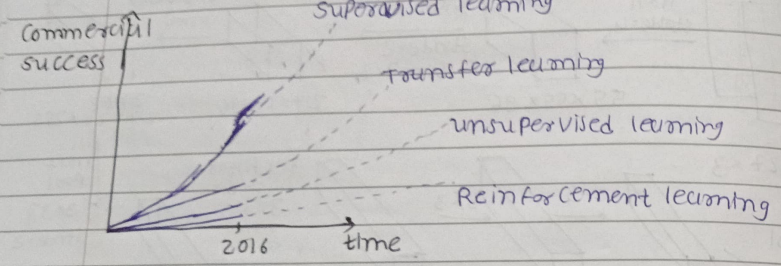
Alex Net



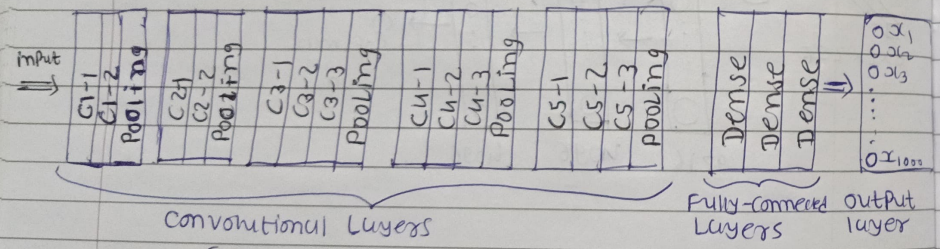
Transfer Learning

⇒ Tranfer Learning is a research problem in machine learning that focuses on applying knowledge gained while solving one task to a related task

For ex Knowledge gained while learning to recognize car could be used in recognization of truck.



VGG16 Model Architecture



⇒ convolution Layers find relation between pixel in 2D matrix and fullyconnected part do classification

⇒ In this we freeze convolution base and only train Fully-connected layers a

ways of tranfer learning

Feature extraction

⇒ In this Dense layers is replace by our dense layers and trainning of convolutional layers freeze trainning occures only on our dense layer

fine tuning

⇒ In fine tuning we also train some last convolution layers

RNN (Recurrunt neural network)

⇒ RNN is type of sequential model to work on sequential data

Ex: Prediction of animals: sequente of input like shape of nose, eyes, ears ect.
sequental data: text (Hi, my name is Vatsal), time series base data, speech (~~~)

why use RNN?

	Input	Output
5 →	Hi my name is Vatsal	0
3 →	I love cammux	0
4 →	India can the match	1

12 unique words

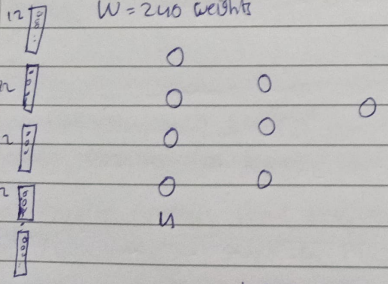
size 12

Hi → [1 0 0 ... 0]
my → [0 1 0 0 ... 0]
name → [0 0 1 0 0 ... 0]

Vectorization

ANN Text classification

W = 240 weights



For 1st input

60

For 2nd input 36×4

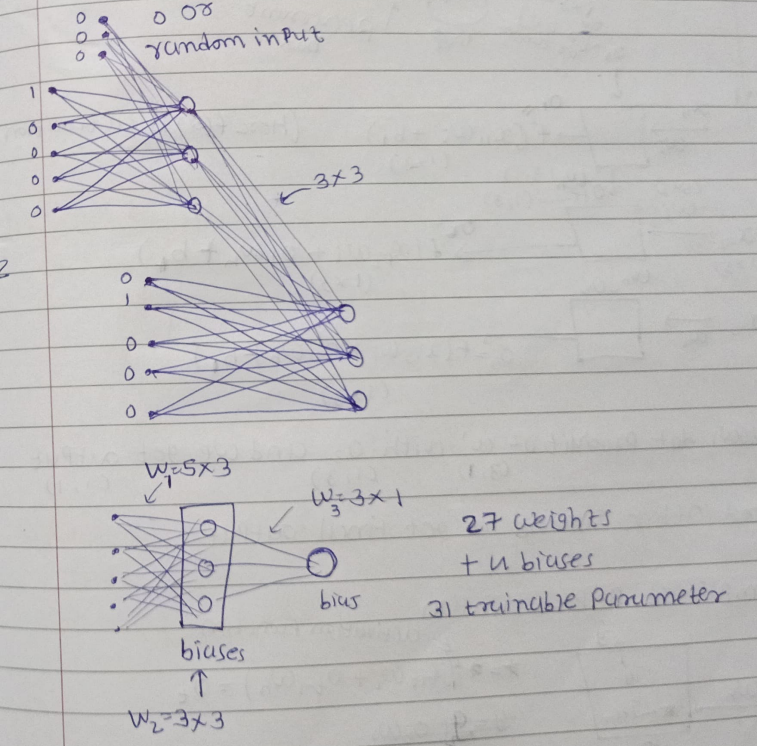
3rd " 48×4

⇒ Here input size is varies but for ANN there should be const input size so for ANN we we zero padding but it is not efficient

Forward Propogation in RNNs

Review	Sentiment
x_1: movie (x_{11}) was (x_{12}) good (x_{13})	1
x_2: movie was bad	0
x_3: movie was not good	0

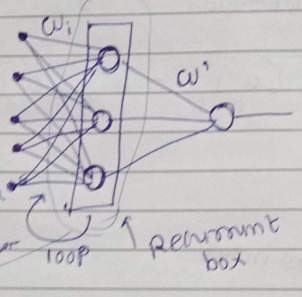
[1 0 0 0 0] movie [0 1 0 0 0] was [0 0 1 0 0] good [0 0 0 1 0] bad [0 0 0 0 1] not



27 weights
+ 4 biases
31 trainable parameter

review	sentiment
$x_{11}\ x_{12}\ x_{13}$	1
$x_{21}\ x_{22}\ x_{23}$	0
$x_{31}\ x_{32}\ x_{33}\ x_{34}$	0

x_{11} → vectors 5D
[1 0 0 0 0]



$O_3 = [0\ 0\ 0]$ or random

t=1 x_{11} → w_i (5×3) → O_1 $= f(x_{11}w_i + b_1)$ (1×3)
(Here f(a) is activation fun)
O_1 → w_h (3×3) (1,3)

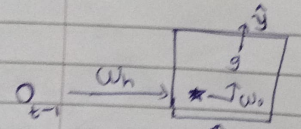
t=2 x_{12} (1×5) → w_i (5×3)
$O_2 = f(x_{12}w_i + O_1 w_h + b_2)$ (1×3)
O_2 → w_h

t=3 x_{13} → w_i
$O_3 = f(x_{13}w_i + O_2 w_h + b_3)$ (1×3)

Now, dot product of w' with O_3 and we get output
(3,1) (1,3) (1,1)

and after sigmoid we get final output

⇒ Simplified Representation

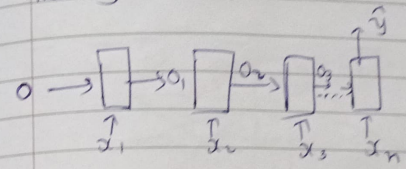


activation function
$* = * f_h(x_{it}w_i + O_{t-1}w_h) = O_t$
~~$g = g f_h O_t w_o$~~
If O_t is last OP
then we pass it through activation fun
$g = g(w_o O_t)$

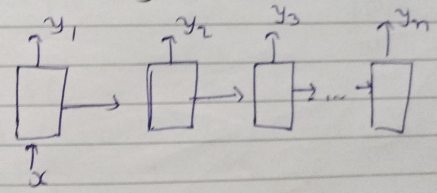
i = row
t = time step

Types of RNN

i) many to one
input → sequence
output → non sequence
Ex sentiment analysis
Rating Prediction

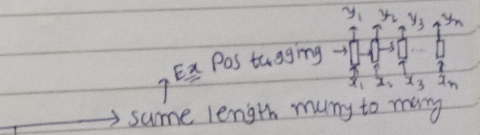


ii) one to many



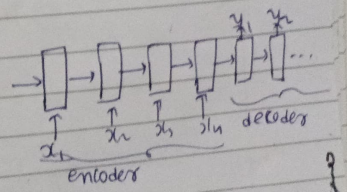
Ex describe image,
music generation

iii) many to many
input → sequence
output → sequence



Ex Pos tagging

Ex machin translation
Eng to hindi



iv) one to one (ANN, CNN)
~~or~~ Input = non sequensial
output = " "

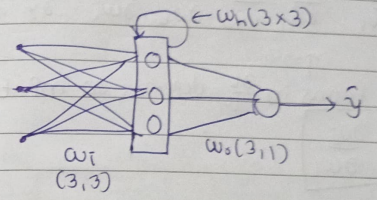
Ex: Image classification

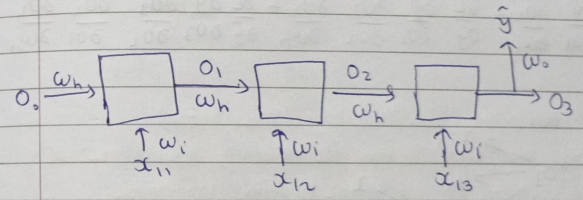
This is simple ~~not~~ neural network

Back Propogation in RNN

Text

			y
cat mat rat	x_1	[1 0 0] [0 1 0] [0 0 1]	1
rat rat mat	x_2	[0 0 1] [0 0 1] [0 1 0]	1
mat mat cat	x_3	[0 1 0] [0 1 0] [1 0 0]	0





$O_1 = f(x_{11} w_i + O_0 w_h)$

$O_2 = f(x_{12} w_i + O_1 w_h)$

$O_3 = f(x_{13} w_i + O_2 w_h)$

$\hat{y} = \sigma(O_3 w_o)$

$L = -y_i \log \hat{y}_i - (1-y_i) \log(1-\hat{y}_i)$

minimiz loss using gradient decent

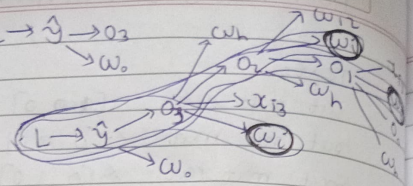
w_i, w_h, w_o

$w_i = w_i - \eta \frac{\partial L}{\partial w_i}$

$w_h = w_h - \eta \frac{\partial L}{\partial w_h}$

$w_o = w_o - \eta \frac{\partial L}{\partial w_o}$

$$\frac{\partial L}{\partial w_o} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_o}$$

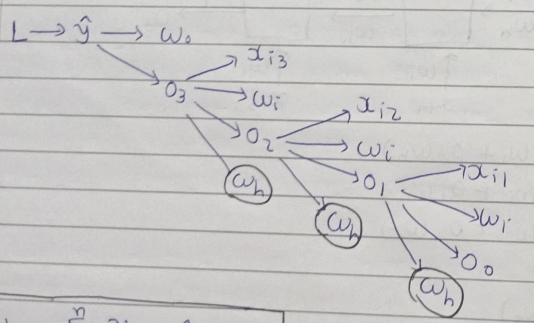


$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial w_i} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial w_i} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial O_1} \frac{\partial O_1}{\partial w_i}$$

↑ Total terms in $\frac{\partial L}{\partial w_i}$ depends on input

$$\frac{\partial L}{\partial w_i} = \sum_{j=1}^{t=3} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_j} \frac{\partial O_j}{\partial w_i}$$

$$\frac{\partial L}{\partial w_h} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial w_h} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial w_h} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial O_1} \frac{\partial O_1}{\partial w_h}$$



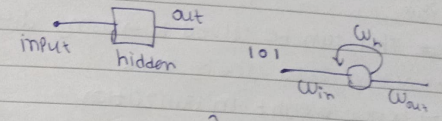
$$\frac{\partial L}{\partial w_h} = \sum_{j=1}^{n} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_j} \frac{\partial O_j}{\partial w_h}$$

problems with RNN

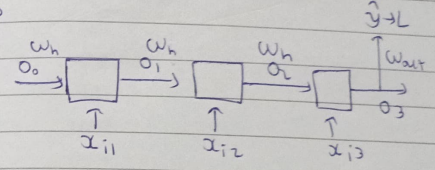
⇒ i) Problem of long term dependency (vanishing gradient problem)
ii) unstable gradients (exploding gradient)

i) Problem of long Term dependency (vanishing)

input			out
1	0	1	
0	0	1	0
0	0	0	0
1	1	1	1



3 time step



$$w = w - \eta \frac{\partial L}{\partial w}$$

$$\frac{\partial L}{\partial w_{in}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial w_{in}} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial w_{in}} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_1}{\partial O_1} \frac{\partial O_1}{\partial w_{in}}$$

↑ Short term dependency

↑ long term dependency

long Term dependency look like

$$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_{100}} \frac{\partial O_{100}}{\partial O_{99}} \cdots \frac{\partial O_2}{\partial O_1} \frac{\partial O_1}{\partial w_{in}}$$

$$= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_{100}} \prod_{t=2}^{100} \left(\frac{\partial O_t}{\partial O_{t-1}} \right) \frac{\partial O_1}{\partial w_{in}}$$

$$O_1 = \tanh(x_{i1} w_{in} + O_0 w_h)$$

$$O_2 = \tanh(x_{i2} w_{in} + O_1 w_h)$$

$$O_t = \tanh(x_{it} w_{in} + O_{t-1} w_h)$$

$$\frac{\partial O_t}{\partial O_{t-1}} = \tanh'(x_{it} w_{in} + O_{t-1} w_h) \, w_h$$

$$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{100}} \left\{ \prod_{t-2}^{100} \left[\tanh'(a_{ft} W_{in} + W_{t-1} W_h) W_h \right] \right\} \frac{\partial o_1}{\partial w_{in}}$$

0 to 1

So above term ≈ 0

$\frac{\partial L}{\partial w_{in}}$ only short term contribution is consider.

Solution ⇒ This can be solved by → using different activation function relu, leaky relu
→ better weight initialization
→ Skip RNNs
→ LSTM

ii) unstable Training (Exploding gradients)

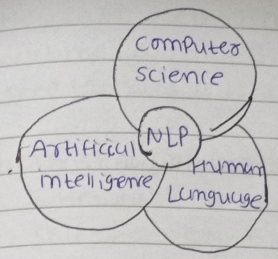
⇒ In this problem long term dependency became very large so gradient and weights become infinite and model does not train.

⇒ If LR is large then same problem occures.

Solution: → gradient clipping (set max value of gradient)
→ controlled Learning Rate (LR)
→ LSTM

Natural Language Processing (NLP)

Introduction to NLP



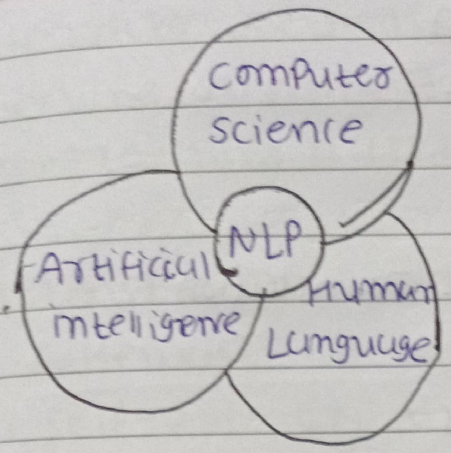
⇒ In neuropsychology, linguistics and the philosophy of language, a natural language or ordinary language is any language that has evolved naturally in humans through use and repetition without conscious planning or premeditation. Natural languages can take different forms, such as speech or signing. They are distinguished from constructed and formal languages such as those used to program computers or to study logic.

Real world applications

⇒ Contextual Advertisements
⇒ Email clients - Spam filtering, smart reply
⇒ social media - removing adult content, opinion mining
⇒ search engines
⇒ chatbots

(NLP)

Introduction to NLP



⇒ In neuropsychology, linguistics and the philosophy of language, a natural language or ordinary language is any language that has evolved naturally in humans through use and repetition without conscious planning or premeditation. Natural languages can take different forms, such as speech or signing. They are distinguished from constructed and formal languages such as those used to program computers or to study logic.

Real world applications

⇒ Contextual Advertisements
⇒ Email clients - spam filtering, smart reply
⇒ social media - removing adult content, opinion mining
⇒ search engines
⇒ chatbots

Common NLP Tasks

⇒ Text / Document classification
⇒ Sentiment Analysis
⇒ Information Retrieval
⇒ Part of speech Tagging
⇒ Language detection & Machine Translation
⇒ Conversational Agents
⇒ Knowledge graph & QA systems
⇒ Text summarization
⇒ Topic modeling
⇒ Text generation
⇒ Spell checking and grammar correction
⇒ Text Parsing
⇒ Speech to text

Approaches to NLP

i) Heuristic methods
ii) Machine Learning Base methods
iii) Deep learning Based methods

i) Heuristic approaches (युक्ति)

A Heuristic is any approach to problem solving or self-discovery that employs a practical method that is not guaranteed to be optimal perfect or rational but is nevertheless (तो भी) sufficient for reaching an immediate short-term goal or approximation.

Examples
→ Regular Expressions
→ wordnet
→ open mind common sense

ii) Machine learning approaches

⇒ Advantage of this approaches is when some time its not possible to make rules by human so in this kind of scenario ML approaches used because in ML rules are made by machine itself.
⇒ Algorithams used
- Naive Bayes
- Logistic regression
- SVM
- L D.A
- Hidden Markov models

iii) Deep learning approaches

→ ML approaches loss sequential information so DL approaches is very useful

→ Architectures used
- RNN
- LSTM (long short term memory)
- CRU / CNN
- Transformers
- Autoencoders

☆ Challenges in NLP

→ Ambiguity (I saw the boy on the beach with my binoculars)

→ Contextual words (I ran to the store because we ran out of milk)

→ Colloquialisms and slang (piece of cake) (easy work)

→ Synonyms

→ Irony, sarcasm and tonal difference

→ Spelling Errors

→ Creativity (poems, dialogue, scripts)

→ Diversity

What is NLP Pipeline

⇒ NLP pipeline is a set of steps followed to build an end to end NLP software.

⇒ NLP software consists of the following steps:

→ Data Acquisition

→ Text preparation
- Text clean up
- Basic Preprocessing
- Advance preprocessing

→ Feature engineering

→ modelling
- model building
  - ML → Naive Bayes, Random forest
  - DL → RNN → LSTM, CNN, BERT
- Evaluation
  - accuracy score
  - Precision & Recall
  - ROC, AUC

→ Deployment
- Deployment
- monitoring
- Model update

Points to remember

→ It's not universal

→ Deep learning pipelines are slightly different

→ Pipe line is non-leniar

i) Data Acquisition

- data Available
  - on your table
  - database
    - data engineers can provide data from database
  - less data
    - Data augmentation
- others have data
  - Public dataset
  - web scraping
    - API
    - Pdf
    - image
    - Audio
- Nobody has data

ii) ~~Text PreProcessing~~
Text - preparation

Cleaning
- HTML tag cleaning
- Unicode normalization (emoji to machine language)
  Ex 😊 = \xf0\x9f\x98\x93
- speling checker

Basic PreProcessing
- Basic
  - tokenization
    - sentence
    - word
- Optional
  - Stop word removal
  - Stemming
  - Removing digits, Punctuations
  - lowercasing
  - language detection

Advance PreProcessing
- Pos tagging
- parsing
- coreference resolution

iii) Feature Engineering

⇒ Feature convert to number

Ex

	review text	sentiment
1	≡≡≡	0
2	≡≡≡	1
.	.	.
.	.	.
.	.	.

⇒ convert Text feature to number features

tve words	-ve words	neutral	senti
5	2	6	0
3	1	6	1
.	.	.	.

Text vectorization

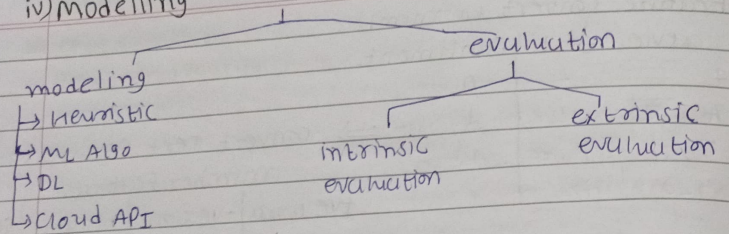
more Techniques
- Bag of words
- Tfidf
- OHE
- ~~word Lover~~
- word2vec
- Custom Features

⇒ Feature engineering by

ML algo
- buse on domain knowledge need to do feature engi. and create feature
- Advantage: can understood How result are come
- Disadv: need to have knowledge about domain

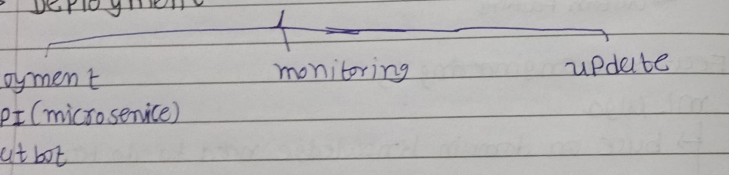
AL algo
- AI algo do feature eng ~~self~~ it self
- Adva: without domain knowledge can make model
- disadv: can not know how actually work

iv) Modelling



which approch should be used is depend on 2 things
- Amount of data
- Nature of Problem

v) ⭐ Deployment



Feature Engineering

⇒ common terms

Corpus (C): Combination of all word of dataset which also contain repeted word

Vocabulary (V): Contain unite words of dataset

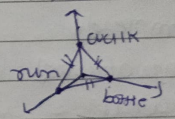
Document (D): Eg each review is called Document

word (w) : word

* One Hot Encoding

In this ~~Technique~~ Technique Text is convert into vector of dimention V (Vocabulary)

Problems with OHE → Sparsity (contain large number of 0)
→ No fixed size
→ OOV (out of vocabulary)
If input is not in vocabulary
→ No capturing of semantic



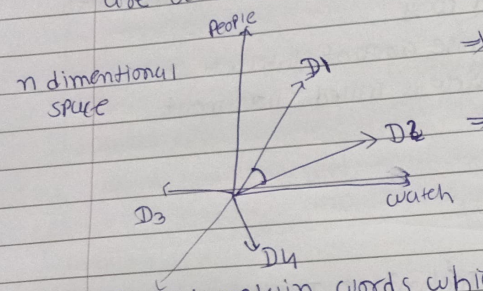
⇒ run and walk are more similar than walk and bottle but all 3 are equal distance
⇒ so OHE cannot capture actual meaning of the sentence

* Bag of words

V = 5

		People	watch	campusx	write	comment
D1	People watch campusx	1	1	1	0	0
D2	campusx watch campusx	0	1	2	0	0
D3	people write comment	1	0	0	1	1
D4	campusx write comment	0	0	1	1	1

⇒ Bag of words note that how many time specific word are occures not note order of words



⇒ angle between two vector denote similarity between them

⇒ sklearn → feature_extraction.text

↓

CountVectorizer

⇒ If Input contain words which not in vocabulary then Bow ignore those words

Advantages: simple and intuitive
fix sized
can handle OOV for some extent
capture semantic little bit

Disadv: sparsity
can't handle OOV completely
can't capture ordering of word
can't do semantic well
For Ex, This is a very good movie
This is not a very good movie
→ Bow denote these two sentence are very related but they are completely opposite.

* N-grams (Bag of n-grams)

Bigram ⇒	People watch	watch campus	campus watch	People write	write comment	campus write
	1	1	0	0	0	0
	0	1	1	0	0	0
	0	0	0	1	1	0
	0	0	0	0	1	1

Adv: able to capture semantic of the sentence
easy to implement

Disadv: dimention of vocabulary is large so it slow the algo.
out of vocabulary

* Tf-Idf

⇒ If-Idf assign weight to every vocabulary. Its give more weight which are more time in document but less in corpus

$$Tf(t,d) = \frac{\text{Number of occurrence of term t in document d}}{\text{Total number of terms in d}}$$

$$IDf(t) = \log_e\left(\frac{\text{total number of doc in corpus}}{\text{number of doc with term t in them}} + 1\right)$$

normalize IDf using $\log_e$

add 1 because when $\log_e(1)$ so we do $\log(1+1) = 2$

Tf ← Particular term how important for That Document

IDf ← Particular Term How important for corpus

adv: used in information retrival

Disadv: sparsity
OOV
dimention high

Tf × IDf

People	watch	camp	write	comm
$\frac{1}{3}\times 0.3$	$\frac{1}{3}\times 0.3$	$\frac{1}{3}\times 0.125$	0	0
0	$\frac{1}{3}\times 0.3$	$\frac{2}{3}\times 0.125$	0	0
$\frac{1}{3}\times 1.3$	0	0	$\frac{1}{3}\times 0.3$	$\frac{1}{3}\times 0.3$
0	0	$\frac{1}{3}\times 0.125$	$\frac{1}{3}\times 0.3$	$\frac{1}{3}\times 0.3$

* Custom Features

For Ex number of +ve reviews, number of -ve reviews
Posive / Negative reviews, number of words

* word2vec ← deep learning base technique

⇒ word embedding: In NLP word embedding is a term for the representation of words for text analysis, typically in the ~~form~~ form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning. Ex BoW, Tf-Idf Techniques

embedding types

- Frequency base
  - BoW
  - Tf-Idf
  - Glove (Matrix factorization)
- Prediction base
  - word2vec

Adv: semantic meaning
low dimention vector
dense vector (Non zero values in matrix) (Not sparse)

⇒ we can use pretrain model and self-train model
⇒ we will use the pretrained weights of word2vec that was trained on Google new corpus containing 3 billion words. this model consists of 300-dimentional vectors for 3 millions words and Phrases.

intuition

	King	Queen	man	woman	monkey
gender	1	0	1	0	1
wealth	1	1	0.3	0.3	0
Power	1	0.7	0.2	0.2	0
weigth	0.8	0.4	0.6	0.5	0.3
speak	1	1	1	1	0

King - man + woman ≈ queen

Types of word2vec

- CBOW (continuous Bag of words)
- Skip-gram

⇒ CBOW

Fake problem / dummy problem → solve → vector (byproduct)

context word → Target ← context word

watch (campusx) for, data science

⇒ our target is to convert above sentence into 3D vector

— — target — — ← for window size 5

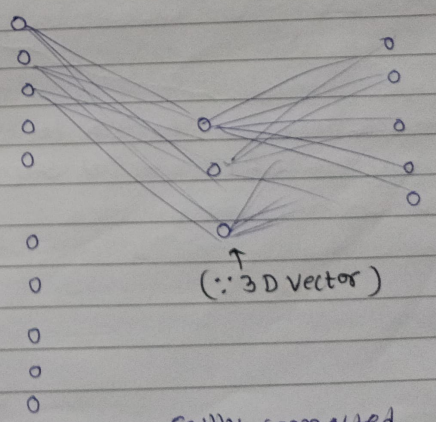
context word

training of deep learning model

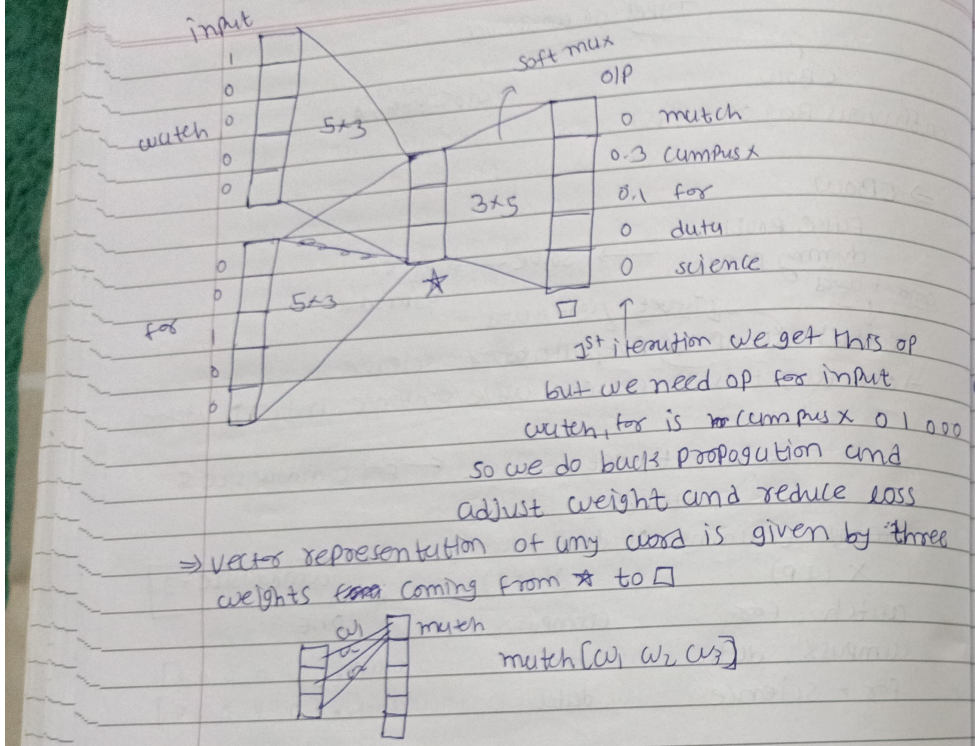
window size = 3

X (IP)	Y (OP)	OHE
watch, for	campusx	watch [1 0 0 0 0]
campusx, data	for	campusx [0 1 0 0 0]
for, science	data	

Creating neural network for above training data



Fully connected



1st iteration we get this op but we need op for input watch, for is campus x 0 1 0 0 0
So we do back propogation and adjust weight and reduce loss

⇒ vector representation of any word is given by three weights coming from ☆ to □

watch $[w_1, w_2, w_3]$

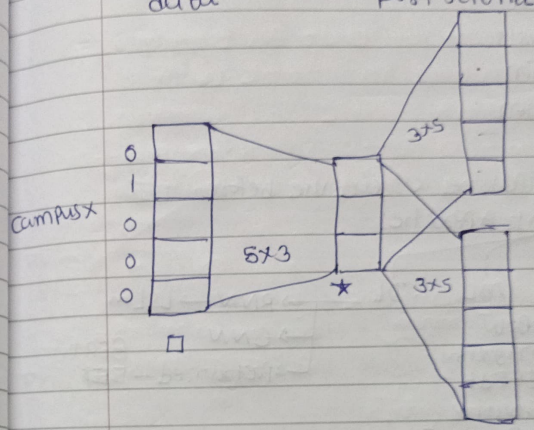
small data = CBOW
large data = skip gram gives best result

⇒ skip gram

x (inputs) y (outputs)

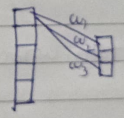
campus x → watch, for
for → campus x, data
data → for, science

window size = 3



⇒ for input campus x o/p should be watch, for in every iteration updates weights and minimize loss

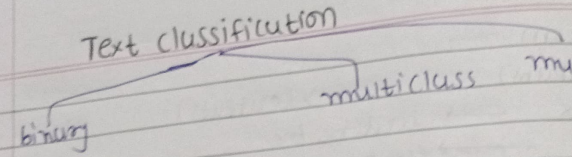
⇒ vector form of word is ~~same~~ corresponding weight of ☆ to □



for watch $[w_1, w_2, w_3]$

⇒ to improve word2vec quality
- increases training data
- increases dimention of vector
- increase window size

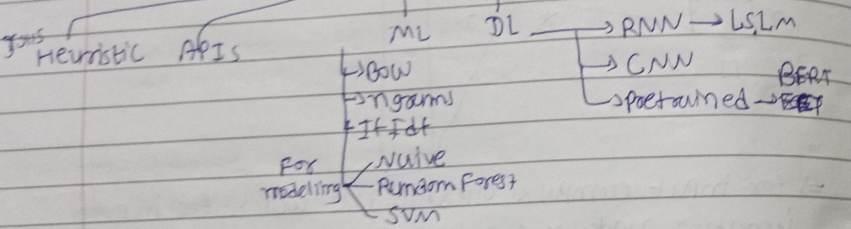
Text classification



Applications: Email spam
Customer support
Sentiment Analysis
Language detection
Fake news detection

⇒ Here we we same pipeline as we use before

Different Approaches



i) Heuristic Approach

Used when less data

Ex: +ve words, -ve words, +ve, -ve, Total words

ii) APIs

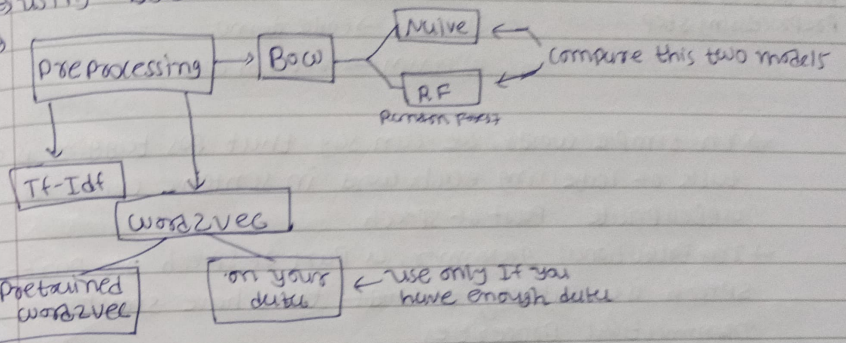
⇒ Amazon, Google have built in solution using their API and can do Classification of text.

disADV: Those company cost money for this task

iii) Machine learning Approaches

⇒ using BoW

⇒



Practical advice: ensemble technique
Heuristic features
Deep learning
make sure you have balance dataset
Some As many Project as you can

Part of speech Tagging in NLP (POS Tagging)

Plane of attack
1) What is POS tagging
2) Applications
3) Spacy → code demo
4) HMM → Viterbi algo

Pos tagging is Preprocessing step

⇒ In simple words, we can say that pos tagging is a task of labelling each word in sentence with its appropriate Part of speech.

⇒ In traditional grammar, a part of speech or part-of-speech is a category of words that have similar grammatical properties

Why not tell someone ?
adverb adverb verb noun punctuation mark, sentence closer

Application: 1) name entity Recognition
2) Question Answering system
3) word sense disambiguation
4) Chatbox

How pos tagging works

⇒ first of all we have ~~some~~ dataset which are label by human itself, and using this we will train our HMM model

Noun Nitish loves Campusx
Verb
Model
Can Nitish google campusx
will Ankita google Campusx
Ankita loves will
will loves google
will will google CampusX

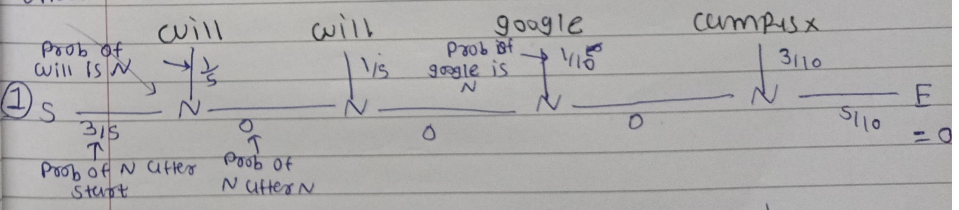
⇒ Emission Probability

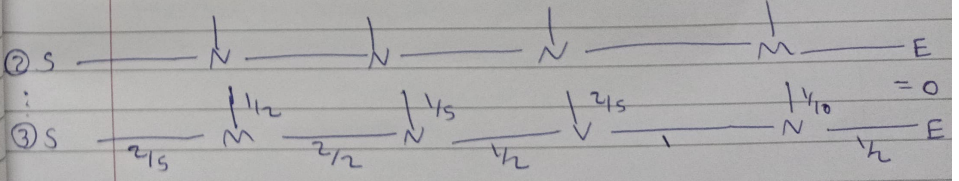
word	N	M	V	
Nitish	2/10	0	0	← given N, M or V
loves	0	0	3/5	Probability of word
campusx	3/10	0	0	
google	1/10	0	2/5	
will	2/10	1/2	0	
ankita	2/10	0	0	
can	0	1/2	0	
	(1)	(1)	(1)	

⇒ Transition Probability (what is prob of next.)

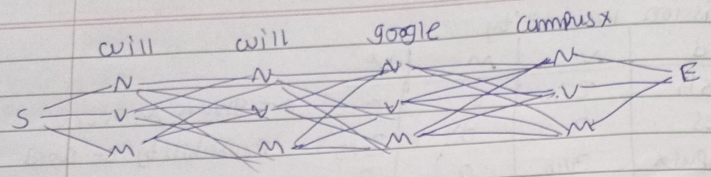
	N	M	V	End
Start	3/5 (S→N)	2/5 (S→M)	0 (S→V)	0 (S→End)
N	0 (N→N)	0 (N→M)	5/10 (N→V)	5/10 (N→E)
M	2/2 (M→N)	0 (M→M)	0 (M→V)	0 (M→E)
V	5/5 (V→N)	0 (V→M)	0 (V→V)	0 (V→E)
	—(1)	—(1)	—(1)	—(1)

Check for every combinations





Total combination = $3^4 = 81$



⇒ Viterbi Algorithum