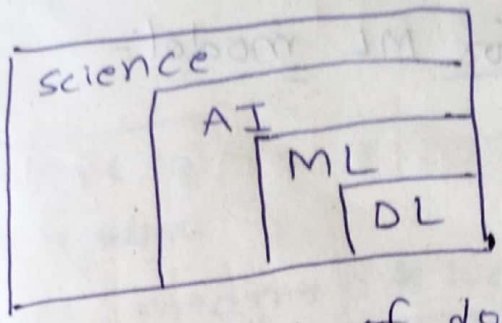
29/12/22

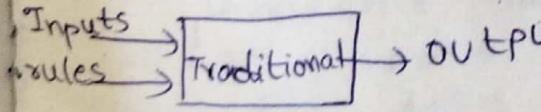
Deep Learning

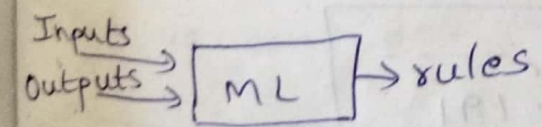


- AI is one way of doing science.
- ML is one way of doing AI
- DL is one way of doing ML
- Deep learning - learning using neural networks
- ML - learning using other than neural networks (except neural networks)
  Ex: SVM, KNN, DTs, RFs etc...
- AI - It is not just learning, but also doing reasoning, planning and solving problems.
- Science - doing science we develop intelligence and knowledge

- ML :- A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E

- Traditional Programming Vs ML program :-





Tasks
1. Sorting & searching
2. Playing chess

} we can describe these tasks using clear set of rules (Traditional algorithms)

3. Recognising speaker listening to voice

④ Understanding what is being spoken
⑤ Recognizing objects.

- Algorithmic components for ML model:-
  - Dataset
  - Model
  - Optimization Algorithm
  - Cost function
  - Training strategies.

- Understanding Feature space:- typical ML algorithm

Q. Define input data of a

Ans Input data is a collection of features that have been quantitatively measured from some object or event that we want the ML system to process.

Linear Regression

Task To prepare a smoothie and serve

Goal To predict the calories

Experience ?

Performance ?

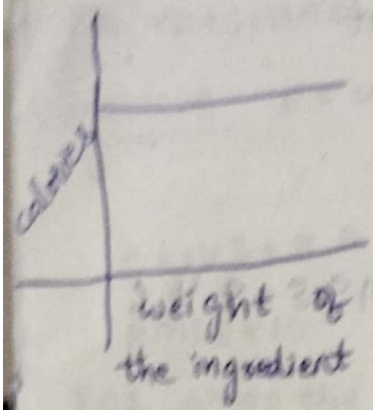
Smoothie = 1 cup of yogurt + 1 secret ingredient + some colour dyes

Calories
300
400

15 smoothies [265, 285, 207, 148, ----, 191]

Smoothie	calories
1	265
2	285
⋮	⋮
15	191
16 ⇒	237

Mean = 236.9
Median = 240



smoothie	weight	calories
1	199	253
2	111	211
⋮	⋮	⋮
15	⋮	141
16		

calories = mean + β * weight
↳ Intercept ↳ slope

Task or Goal :- Predict calories

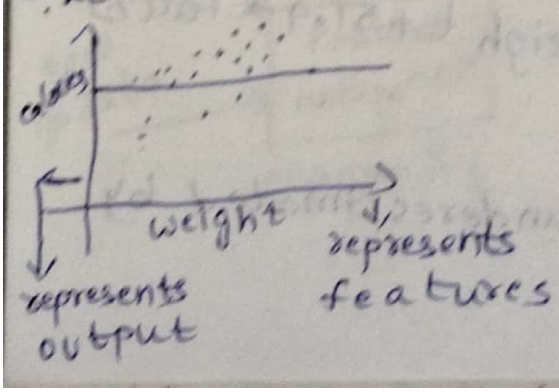
Model 1 :- Wild guess
↳ very bad model
↳ we are not having any experience

Model 2 :- We use experience
If 15 previously made smoothies
253, ------ 141 ⇒ mean = 236.9
median = 240

↳ No features are used
↳ Prediction for 16th smoothie is 238
but actual calories = 445
↳ ingredient quich of 88 gm
↳ underestimation by 208

Model 3 :- more data
↳ doesn't mean moderate points
↳ It means more information about the data.

- Let us consider weight of ingredient

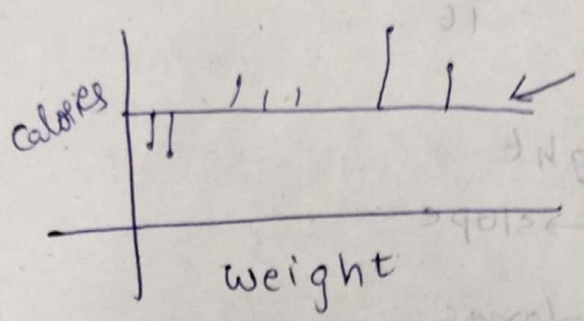


Data points

weights	calories

Model 2 calories = 236.9 + 0 * weight

Model 3 cal = $w_0 + w_1$ * weight



Model 2 RMSE = 84.6

~~for~~ model 2

Model 3: c = 236.9 + 5 * weight

Model 4: c = 236.9 + 1 * weight

Model 5: c = ~~236.9~~ 200 + 1 * weight

RMSE = 69.3

= 69.2

= 52.8

Finally: c = 175 + 1.35 * weight

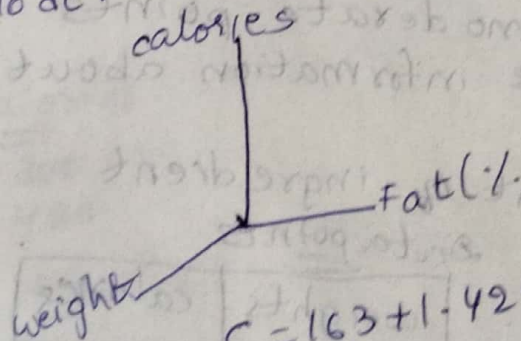
RMSE = 48.8

~~To make it much better~~, Model 6:

For 16th smoothie, the prediction = 293.8

↳ underestimated by 445 − 293.8 = 151.2

Model 7: c = $w_0 + w_1$ * weight + w_2 * Fat(%)



c = 163 + 1.42 * weight + 51.9 * Fat(%)

RMSE = 46.9

For 16th smoothie pred underestimated by an amount 143

So, consider Fat(Y), Carbohydrates, Protein (9-4-4)

model: $c = w_0 + w_1 * Fat(Y) + w_2 * Carbohydrates + w_3 * Protein$

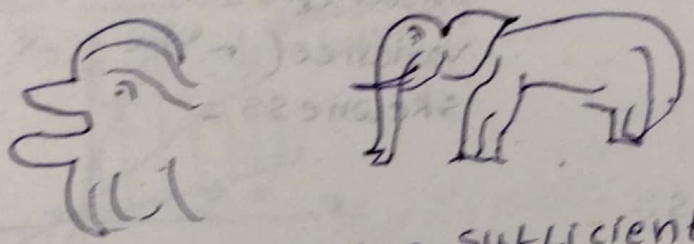
$C = 163 + 8.9 * Fat + 3.9 * Carbohydrates + 4.3 * Protein$

RMSE < 10

For smoothe pred = 431

Classification - Computer vision

Task1: Lion or Elephant



Shape features are sufficient

Task 2 :- Horse vs Zebra

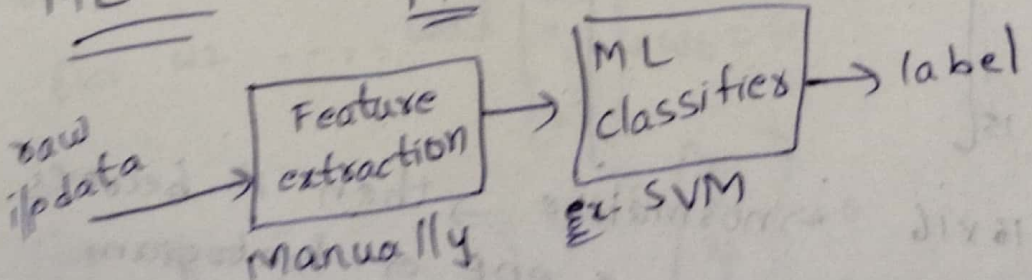
Shape features are not useful

but regional (Texture) features are useful

Elephant

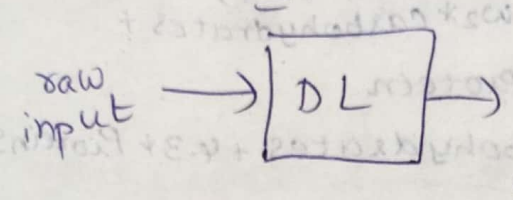
ML vs DL

ML

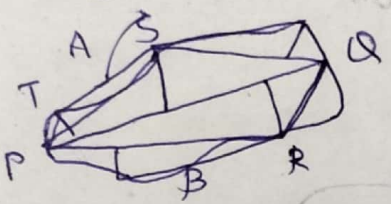


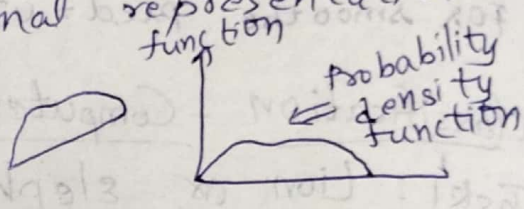
Model 2 calorie s = 236.9 + 0 * weight

DL



Shape descriptors : polygonal representation





mean(μ) =
variance(σ^2) =
skewness =

30/12/22

*Regional features :-

① Texture

Texture can be represented using the cooccurence matrix $A_p(i,j)$
- j → gray level j
- i → gray level i
- p → some position vector

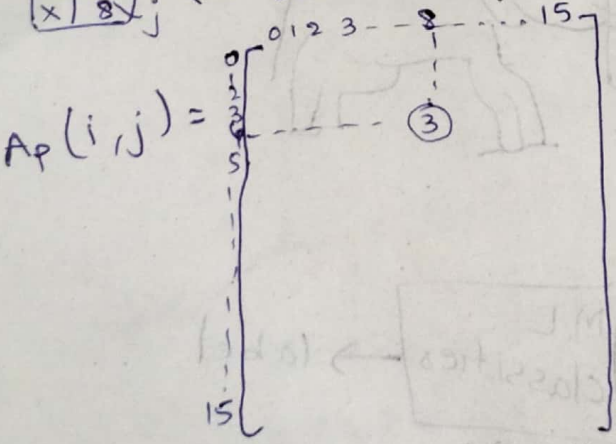
4-bit image of size 8×8

i = 4 ; j = 8

$p = \langle d, \theta \rangle = \langle 1, 45° \rangle$
- θ → orientation
- d → distance

4	x
x	8

j ⇐ count



$A_p(i,j)$ =

Raw image

10	9	7	9	5	8	11	9
65	5	15	12	4	6	3	2
9	3	2	10	6	8	4	5
8	2	4	3	7	5	6	1
8	0	11	8	10	9	8	2
8	4	7	1	6	0	7	2
8	3	8	9	11	6	3	9
2	7	2	8	8	6	12	6	7

$A_p(i,j)$ 16×16 → normalize → then it becomes a PDF or histogram denoted as C_{ij} 16×16

$A_p(i,j)$ ↑ given $\langle 1, 45° \rangle$

① Max prob : $\max_{i,j} c_{ij}$

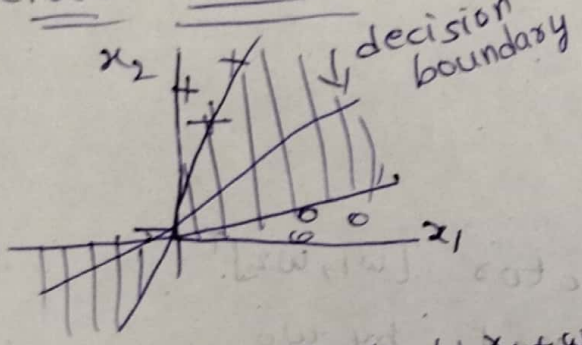
② Elementwise difference moment : $\sum_i \sum_j (i-j)^k c_{ij}$

③ Inverse elementwise diff : $\sum_i \sum_j c_{ij} / (i-j)^k$

④ Uniformity : $\sum_i \sum_j c_{ij}^2$

⑤ Entropy : $-\sum_i \sum_j c_{ij} \log(c_{ij})$

*Linear classification



decision boundary $w_1 x_1 + w_2 x_2 + w_0 = 0$
↳ bias

Decision boundary is the normal of the vector
$[w_1, w_2]^T$

$$\sum_{i=1}^{d} w_i x_i + w_0 = 0$$

$$\boxed{w^T x + w_0 = 0}$$

Decision rule is
If $w^T x + w_0 > 0$; then $x \in C_1$
$w^T x + w_0 < 0$; then $x \in C_2$
we want to unify them into one rule

$$[w_1 \; w_2 \; \dots \; w_d] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} + w_0 = 0$$

becomes

$$\underbrace{[w_1 \; w_2 \; \dots \; w_d \; w_0]}_{a} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ 1 \end{bmatrix}}_{y} = 0$$

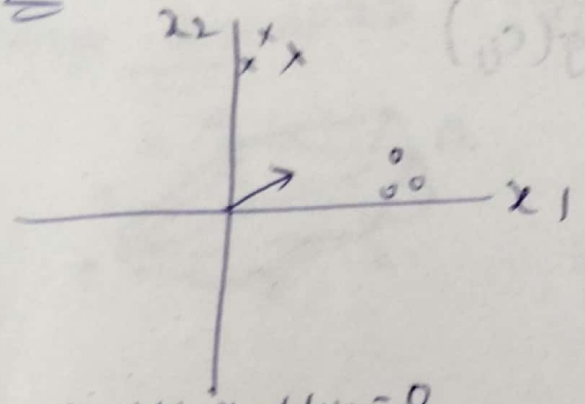
Decision rules

$a^T y > 0 \; ; \; y \in C_1$

$a^T y < 0 \; ; \; y \in C_2$

Now, negate y of class 2

DR :- $a^T y > 0$



$w_1 x_1 + w_2 x_2 + w_0 = 0$

Decision boundary

↳ is normal to the vector $[w_1, w_2]^T$

↳ Position is determined by w_0

2/1/23

↳ Input data
↳ Feature space
→ Model Architecture
→ Training strategy
→ Optimization algorithm
→ Loss function (RMSE)

Finally performance on test set.

Regressing → Fitting the hyperplane

Classification → Finding the decision boundary which seperates two classes.

Regression

calories = $w_0 + w_1 *$ Fat (1) $+ w_2 *$ carbo $+ w_3 *$ protein

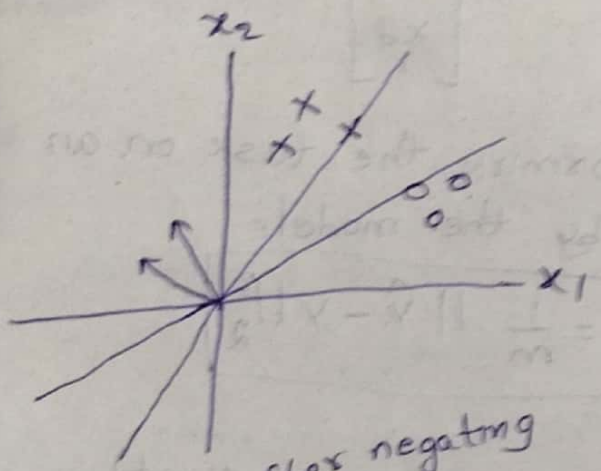
prediction = $w_0 + w_1 * feat_1 + w_2 * feat_2 + w_3 * features$

classification

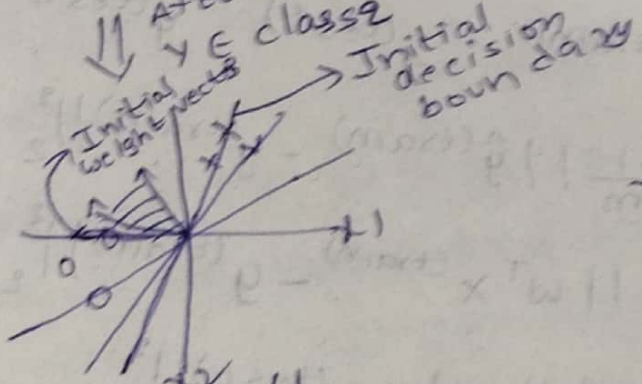
decision boundary $w_1 * f_1 + w_2 * f_2 + w_0 = 0$ is also a hyperplane

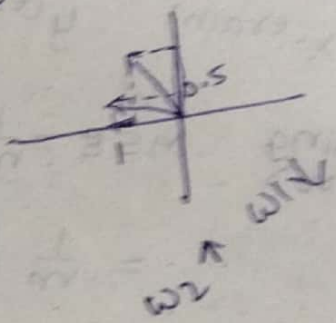
Decision rule $\boxed{a^T y > 0}$

$$a = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_0 \end{bmatrix} \qquad y = \begin{bmatrix} x_1 \\ x_2 \\ 1 \\ x_4 \\ 1 \end{bmatrix}$$



After negating $y \in$ class2





Training Algorithm:

1) $a(0) \leftarrow$ Arbitrarily initialize

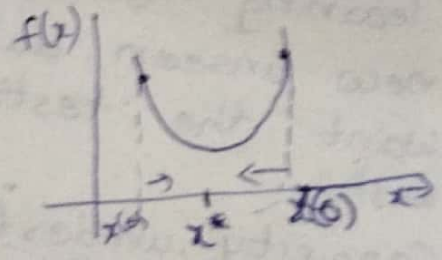
2a) $a(k) \leftarrow a(k-1)$ in such a way that $a(k)$ minimizes some performance index.

2)b) Error $J = -\sum_{\forall y \text{ misclassified}} a^T y$

$$\boxed{\nabla J = -\sum y}$$

2) c) $a(k) = a(k-1) - \eta \nabla J$

$$a(k) = a(k-1) + \eta \sum_{\substack{\forall y \\ \text{misclassified}}} y$$



$$x^* = \arg\min_x f(x)$$

$$x(k) = x(k-1) - \eta \frac{\partial f(x)}{\partial x}$$

* Inference:-

Regression $w_1 x_1 + w_2 x_2 + w_0 = 0$

classification $a^T y = 0$

$w^T x = 0$

where $W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ w_0 \end{bmatrix}_{(d+1)\times 1}$ $\quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ 1 \end{bmatrix}$

- Inference means performing the task on an unseen data so far by the model.

Training error $\boxed{MSE = \frac{1}{m} \| \hat{y} - y \|_2^2}$

m samples

$x^{(train)}$ $y^{(train)}$ $x^{(test)}$ $y^{(test)}$

Training error $MSE = \frac{1}{m} \| \hat{y}^{(train)} - y^{(train)} \|_2^2$

$= \frac{1}{m} \| w^T x^{(train)} - y^{(train)} \|_2^2$

Test error $= \| \hat{y}^{(test)} - y^{(test)} \|_2^2$

↳ Generalization error

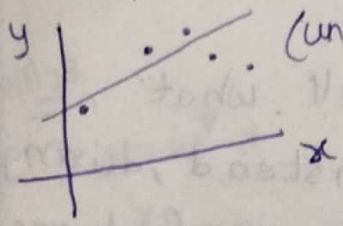
* Generalization:- It is the ability of machine-learning systems to perform well on the new unseen data. That means to say we want the testing error also too small as well.

R - Capacity, underfitting and overfitting:-

$pred = w_1 x_1 + w_2 x_2 + w_0$

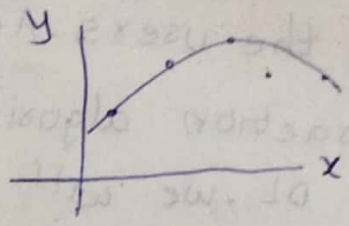
Different weight vector values gives us different functions.

Range of functions which the model can learn is called the capacity of that model. The goal then becomes selecting one among those functions which will best fit the data (or) which will well-seperate the data corresponding to two categories respectively.



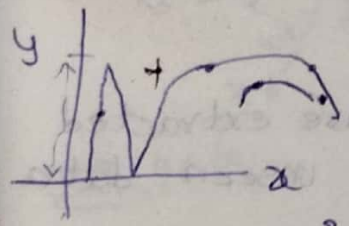
(underfitting)

$y = w_1 x + w_0$



(Best fitting)

$y = w_1 x + w_2 x^2 + w_0$



(Overfitting)

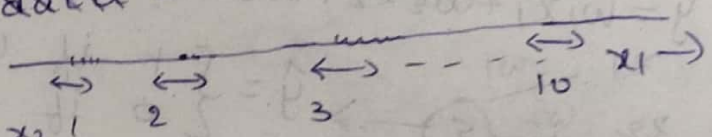
(Training error = 0)

$y = w_1 x + w_2 x^2 + \cdots + w_9 x^9 + w_0$

- Challenges of ML driving us to DL :-

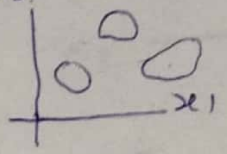
① The course of ~~data~~ dimensionality

one feature



+second feature

$10^2 = 100$

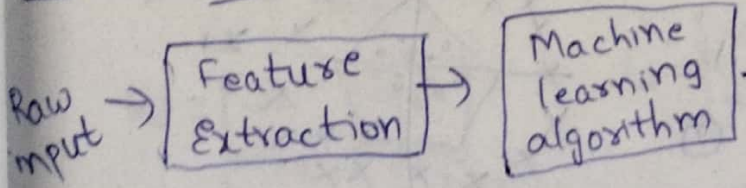


+third feature

② Local constancy and smoothness regularization

③ Manifold learning.

ML Approach



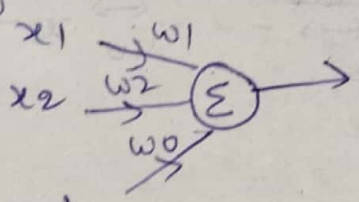
DL Approach

Raw i/p → [Machine Learning Algorithm] → Decision on i/p

- In traditional ML, the features are to be decided by the users. Accordingly, we need feature extraction algorithm.
- In case of DL, we will not tell what features are to be analyzed. Instead, during the training phase the model learns features from the training samples that are useful in taking decisions.
- During inference, similar features are extracted by the model from the so far unseen data in order to make a decision.

* Perceptron :-

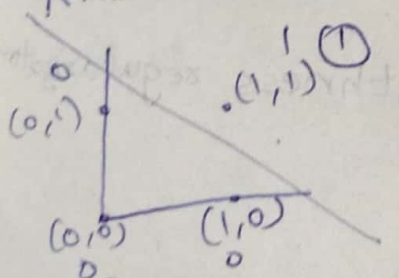
$y = w_1x_1 + w_2x_2 + w_0$

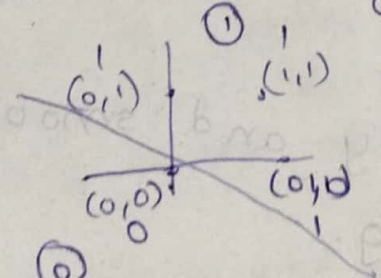


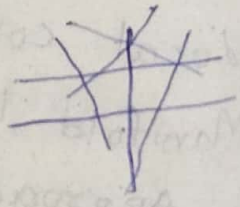
$$\hat{y} = \begin{cases} 1 & \text{if } w^Tx > 0 \\ 0 & \text{if } w^Tx < 0 \end{cases}$$

x_1	x_2	AND	OR	NOT
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	0	0

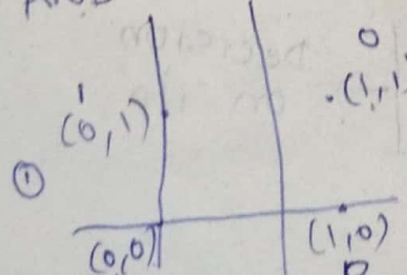
* AND OR NOT



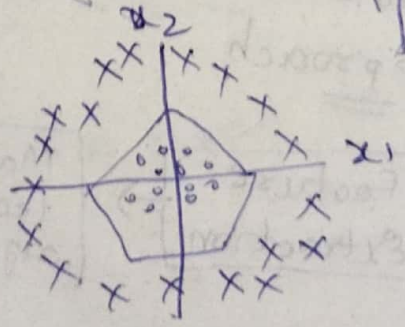


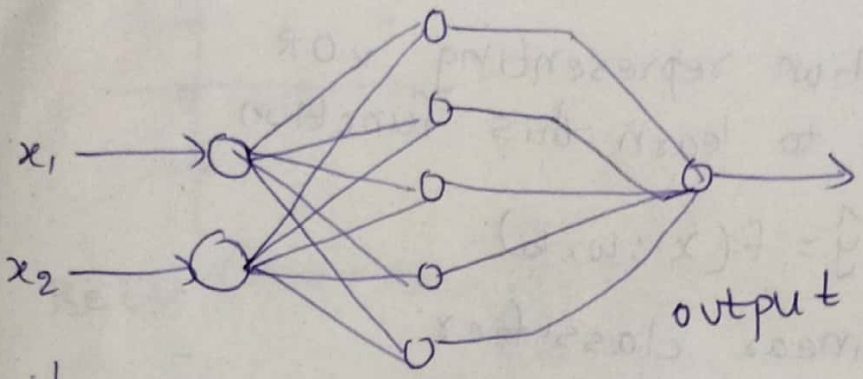


AND



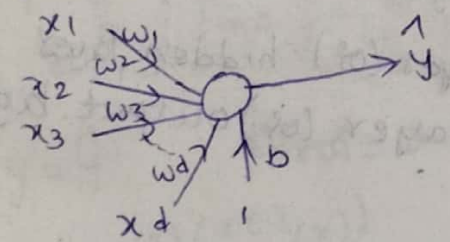
OR



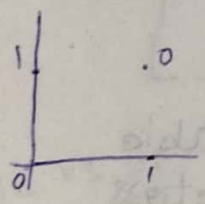


9/11/23

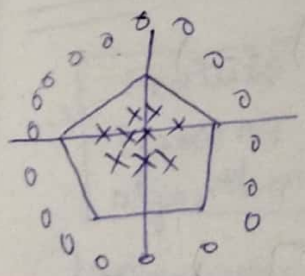
$y = w^T x$

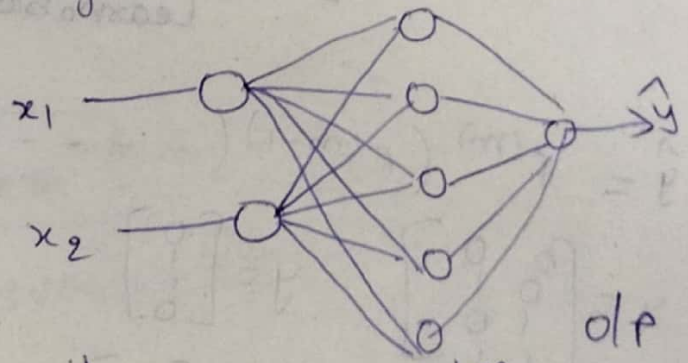


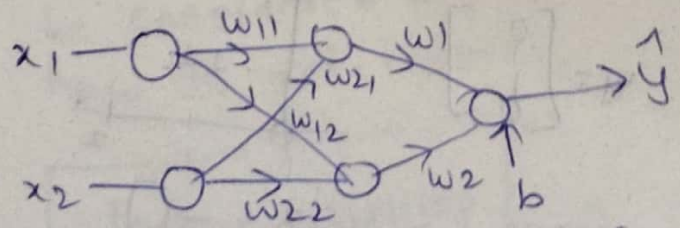
XOR gate



Q Single perceptron can classify AND, OR, NOT with linear decision boundary but not XOR gate.







$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}_{2\times 2} \qquad w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}_{2\times 1} \qquad C = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}_{2\times 1} ; b_{1\times 1}$$

$y = f^*(x)$

- The function representing XOR
- we want to learn this function

we tried $\hat{y} = f(x; w, b)$

- linear classifier
- This failed

$\hat{y} = f^{(2)}(f^{(1)}(x)) = f(x)$

- $f^{(1)}$ — Input layer
- first layer (or) hidden layer
- second layer (or) output layer

$\hat{y} = f(x; w, c, w, b)$

θ (or)

- θ → parameters (or) Learnable (or) trainable parameters

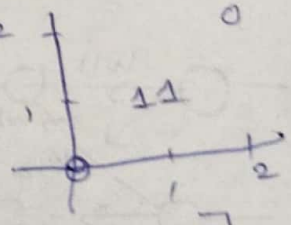
$\hat{y} = f^{(m)}(f^{(m-1)}(\dots f^{(1)}(x)) \dots)$

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

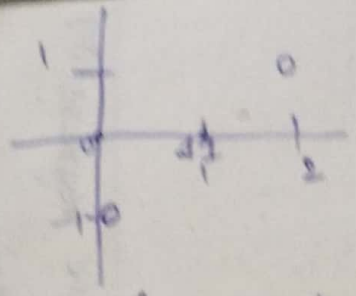
$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad w = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad c = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad b = 0.$$

$$XW = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}_{4\times 2} \quad (4\times 2 \;\; 2\times 2)$$

$$y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$



$$XW + c = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix} + \begin{bmatrix} 0 & -1 \\ 0 & -1 \\ 0 & -1 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

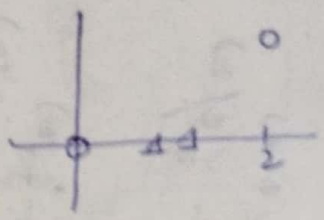


$$ReLU(XW+c) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$$ReLU(x) = \max\{0, x\}$$

$$ReLU(XW+c)\cdot w + b$$

$$= \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \hat{y}$$

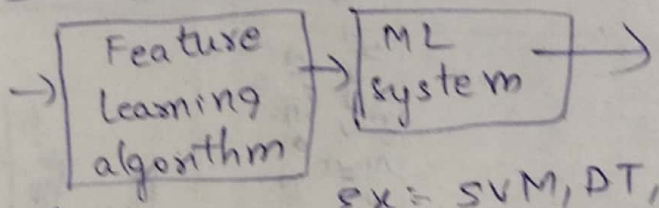


$y = \hat{y}$ Error = 0

$f(x) = f^{(2)}(f^{(1)}(x))$

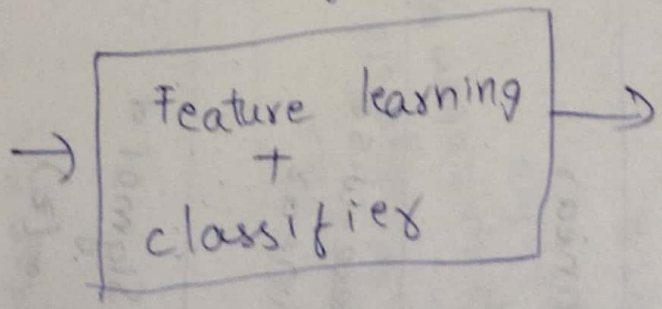
- $f^{(1)}$ → feature learning layer
- $f^{(2)}$ → Linear classifier

ML vs DL

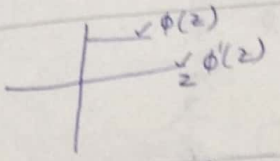
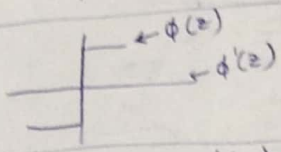
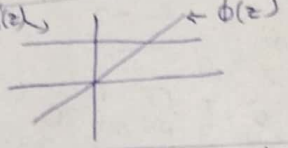
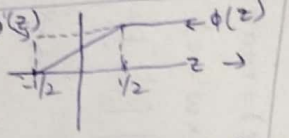
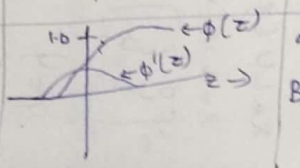
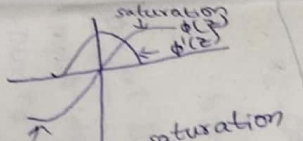
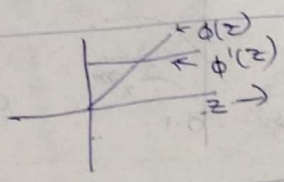
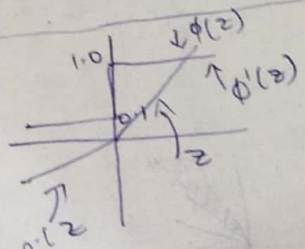


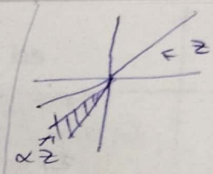
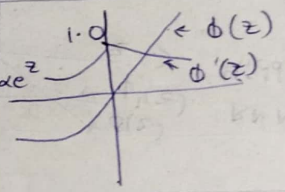
Ex: SVM, DT, RF

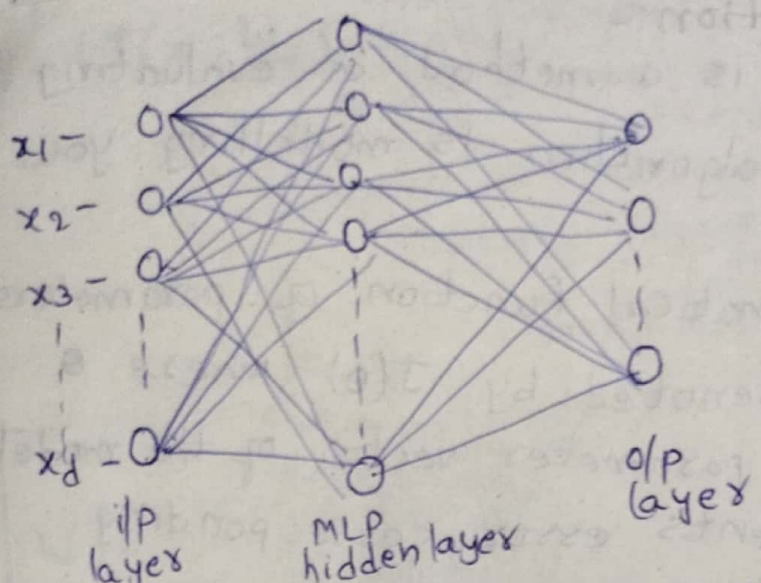
↳ Designer has to decide which features to be considered & how to get them.



Types of Activation functions:-

Activation function	Equation	Derivation	1D Graphs	Examples
① Binary step (or) unit step (or) Heaviside	$\phi(z) = \begin{cases} 0 & z<0 \\ 0.5 & z=0 \\ 1 & z>0 \end{cases}$	$\phi'(z) = 0$		MLP
② Signum	$\phi(z) = \begin{cases} -1 & z<0 \\ 0 & z=0 \\ 1 & z>0 \end{cases}$	$\phi'(z) = 0$		MLP
③ Linear	$\phi(z) = z$	$\phi'(z) = 1$		
④ Piece-wise linear	$\phi(z) = \begin{cases} 1 & z \geq 1/2 \\ z+1/2 & -1/2 < z < 1/2 \\ 0 & z \leq -1/2 \end{cases}$	$\phi'(z) = \begin{cases} 1 & ; -1/2 < z < 1/2 \\ 0 & ; \text{otherwise} \end{cases}$		
⑤ Sigmoid $\sigma(z)$	$\phi(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z}$	$\phi'(z) = \sigma(z)[1-\sigma(z)]$		Logistic regression Binary-classification
⑥ tanh	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$\phi'(z) = 1 - \tanh^2(z)$		
⑦ Soft plus Note:- Sigmoid = differentiation of softplus	$\phi(z) = \log(1+e^z)$	$\phi'(z) = \frac{e^z}{1+e^z}$		
⑧ ReLU	$\phi(z) = \max(0,z)$ (or) $\phi(z) = \begin{cases} z & ; z \geq 0 \\ 0 & ; z < 0 \end{cases}$	$\phi'(z) = \begin{cases} 1 & ; z \geq 0 \\ 0 & ; z < 0 \end{cases}$		RNN, CNN, Hidden layers
⑨ Leacky ReLU	$\phi(z) = \max(0.1z, z)$ $\phi(z) = \begin{cases} z & ; z \geq 0 \\ 0.1z & ; z < 0 \end{cases}$	$\phi'(z) = \begin{cases} 1 & ; z \geq 0 \\ 0.1 & ; z < 0 \end{cases}$		

(10) Parametric ReLU	$\phi(z) = \max(\alpha z, z)$ $\phi(z) = \begin{cases} z & ; z \geq 0 \\ \alpha z & ; z < 0 \end{cases}$	$\phi'(z) = \begin{cases} 1 & ; z \geq 0 \\ \alpha & ; z < 0 \end{cases}$		
(11) Softmax	$\phi(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$			Multi class classification
(12) Exponential linear unit	$\phi(z) = \begin{cases} z & ; z \geq 0 \\ \alpha(e^z - 1) & ; z < 0 \end{cases}$	$\phi'(z) = \begin{cases} 1 & ; z \geq 0 \\ \alpha e^z & ; z < 0 \end{cases}$		



Output layer :- It can see labels/targets i.e., input data it can see.

- During training phase, parameters are updated such that the dp becomes close to targets.

Hidden layers :-

- They don't have access to labels or targets i.e., training datasets.
- obviously, the hidden layers won't get any direct guidance from dataset in updating its parameters.

↳ solution is backpropagation.

→ Thus, they are called hidden layers.

For output layer, the activation function is decided choosen based on type of tasks. The tasks can be

① Regression - Linear Activation function

② Binary classification - sigmoid activation function (gives conditional probability)

③ Multi class classification - Softmax

④ Multi label classification - sigmoid

Hidden layers - The activation functions are chosen based on the type of neural network architecture

① CNN - ReLU

② RNN - Tanh (or) sigmoid

Cost / Loss function:-

① Cost function is a method of evaluating how well your algorithm is modelling your dataset.

- It is a mathematical function of parameters of the model denoted by $J(\theta)$ where θ represents the parameter vector of the model.
- Loss → represents error corresponding one training sample
  cost – represents error over the entire dataset.

Linear regression loss :-

① MSE (Mean-scale-error) ② MAE (Mean-absolute-error) ③ Huber loss

Classification loss :-

① Binary cross-entropy
② Categorical cross-entropy

Auto encoders :-

① KL divergence loss

GAN :-

① Discriminative loss
② Minmax GAN loss

Object Detection :-

① Focal loss

Word Embeddings :-

① Triplet loss

Problem setting

$x, y \sim P_{data}$ ⇒ the entire dataset coming from the distribution P_{data}

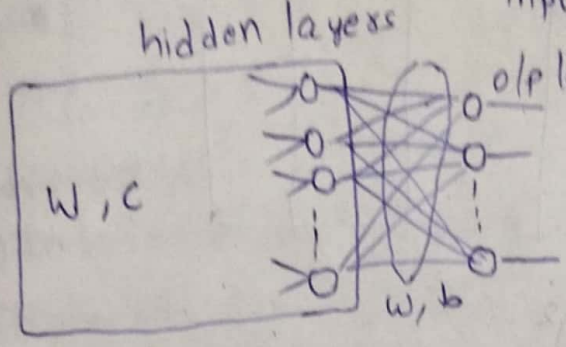
$\hat{y} \sim P_{model}$ ⇒ predictions produced by the model ($\hat{y}$) according its own learned distribution P_{model}

$\hat{y} = f^*(x; W, c, w, b)$ → set of parameters corresponding o/p layer.

θ → complete set of model parameters.

↳ set of parameters of all the remaining layers of model

input layer — all hidden layers.



$z = W^T h + b$

↳ feature vector from last hidden layer

↳ linear output of the output layer.

$\hat{y} = \phi(z) = \phi(W^T h + b)$

↳ Activation function

↳ decides what is the type of output units.

eg: if you choose ϕ to be sigmoid, then output unit is said to be of sigmoid type.

Problem setting

Goal: we want $\hat{y}_i$ (prediction) as close to y_i (target) as possible $\forall i$ for all training samples.

→ If they are not equal, then there is error given by $y_i - \hat{y}_i$ (+ve / -ve)

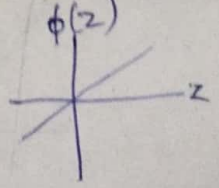
→ we define cost function $J(\theta)$ which is a function of model parameters, and we try to update 'θ' such that $J(\theta)$ gets minimized.

Linear Regression Loss:-

→ which $\phi(z)$ is to be used?

Ans $\phi(z)$ must be linear function

$\phi(z) = z$

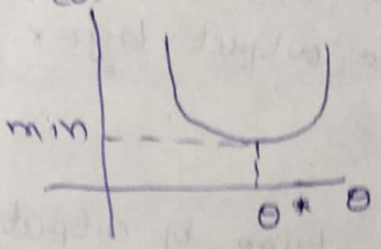


$\rightarrow \hat{y} = \phi(w^T h + b) = w^T h + b$

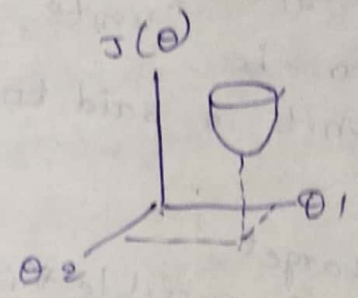
S. No	features					x_d	y	$\hat{y}$
	x_1	x_2						
1								
2								
-								
-								
-								
N								

① MSE cost

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$



$$\theta^* = \arg\min_{\theta} J(\theta)$$



Advantages:-

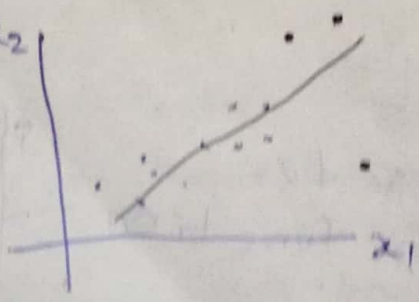
① Easy to interpret.

② Always differentiable. because it is a quadratic function.

③ Only one local minima present

Disadvantages:-

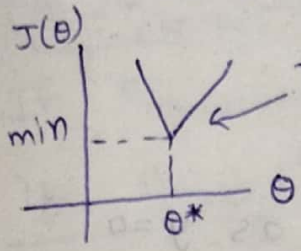
① MSE is not robust to outliers



② MAE

$\hat{y} = \phi(w^T h + b) = w^T h + b$

$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$



this is a discontinuity
∴ not differentiable at that point

Advantages :-

① Intuitive and easy to implement.

② Robust to outliers

Disadvantage :-

① Not differentiable. So gradient descent cannot be used directly but we can use subgradient descent algorithm.

③ Huber loss

$$J_{huber}(\theta) = \begin{cases} \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \frac{1}{N} \sum_{i=1}^{N} \delta \left(|y_i - \hat{y}_i| - \frac{1}{2} \delta \right) & \text{if } |y_i - \hat{y}_i| > \delta \end{cases}$$

δ - hyper parameter

- the hyperparameter δ defines the point where the cost fn(J(θ)) transitions from quadratic to linear.

Advantages :-

~~Note :- MSE performance~~

① Robust to outliers

② Its performance lies blw MAE & MSE

Disadvantage :-

① Associated complexity. The hyper parameter δ which also needed to be optimized which is an additional requirement on the training algorithm.

Note:- For linear regression, the output layers contains only one output unit with linear activation.

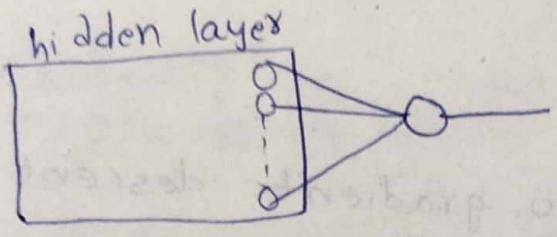
Classification losses:-

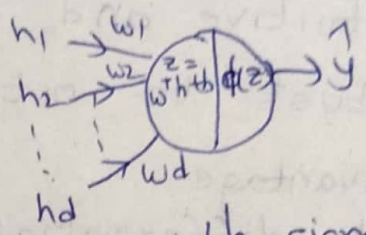
① Binary cross-entropy

#categories (OR) #classes = 2

Let us say class 1 represented as $y=0$

class 2 represented as $y=1$



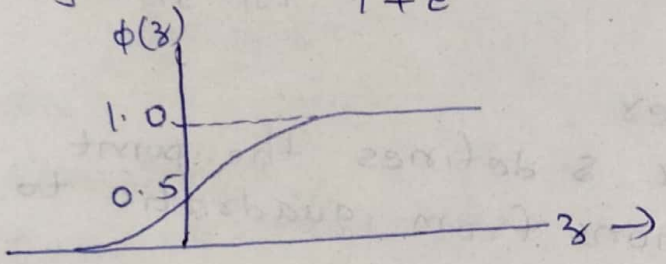


dp layer having only one o/p unit with sigmoid function

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

$z = w^T h + b$

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



① $-\infty < z < \infty$

② Min value of $\sigma(z) = 0$

③ Max value of $\sigma(z) = 1$

④ $\therefore 0 \le \hat{y} \le 1$ ⇒ can be interpreted as probability

$\sigma(z)$ is converting 'z' into probabilities.

→ we know that 'y' has one of the two possible values $y=0$ or $y=1$

$y=0$ if $x \in$ class 1

$y=1$ if $x \in$ class 2

→ But $\hat{y}$ can take any real number in between

0 and 1 $\Rightarrow \hat{y}$ is a probability

$\hat{y} = P_{model}(y=1/x)$

~~$\hat{y} = P_{model}(y=0/x)$~~ (or) ~~x~~

If $P_{model}(y=1/x) = \hat{y} > 0.5$ then $x \in$ class 2

If $P_{model}(y=1/x) = \hat{y} < 0.5$ then $x \in$ class 1

$$J(\theta) = -\sum_{i=1}^{N} y_i \log \hat{y_i} + (1-y_i) \log (1-\hat{y_i})$$

S.No		y	$\hat{y}$
1			
...			
N			

Suppose the accuracy = 1

Loss $= -y_i \log \hat{y_i} - (1-y_i) \log (1-\hat{y_i})$

Case I :- $y_i = 1$ & the model prediction also $\hat{y_i} = 1$

Loss $= -1 \cdot \log 1 - (1-1) \log (1-1)$

$= 0$

case II :- $y_i = 0$ and $\hat{y_i} = 0$

Loss $= -0 \cdot \log 0 - (1-0) \log (1-0) = 0$

case III :- $\hat{y_i}$ is other than 0 (or) 1 when y_i is 0 (or) 1 respectively.

Eg $\hat{y_i} = 0.21, 0.77, 0.98$

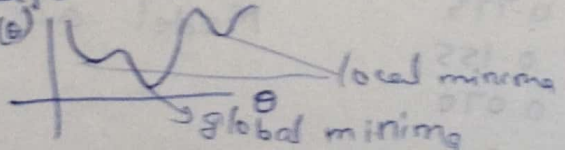
→ Then there will be non-zero loss, and adding losses corresponding to all training samples gives us $J(\theta)$

Advantages :-

① The cost function is differentiable

Disadvantages :-

① the loss landscape will have multiple local minima



② It is not intuitive

② Multiclass classification loss

#categories = #classes = k

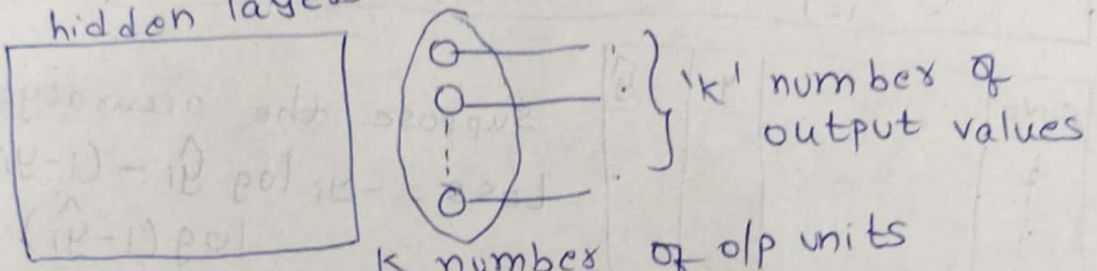
class 1 $y=0$

class 2 $y=1$

⋮

class k $y=k-1$

hidden layers



S.No		y	$\hat{y}$
1		2	[, , , ,]$_{1\times k}$
2		4	[, , , ,]$_{1\times k}$
⋮		⋮	⋮
N			

Softmax $\phi(z)_i = \dfrac{\phi(z_i)}{\sum_{j=1}^{k} \phi(z_j)}$

Ex Iris dataset

$k=3$

		one-hot encoding
setosa	0	[1, 0, 0]
virginica	1	[0, 1, 0]
versicolor	2	[0, 0, 1]

i	y_i	$\hat{y}_i$
1	1	0.775
2	0	0.155
3	0	0.070

⟶ Loss = 0.3675

Model accuracy = 63.25 %.

$$loss = -\sum_{j=1}^{k} y_j \log \hat{y}_j$$

$$J(\theta) = -\sum_{i=1}^{N} \sum_{j=1}^{k} y_{ij} \log \hat{y}_{ij}$$

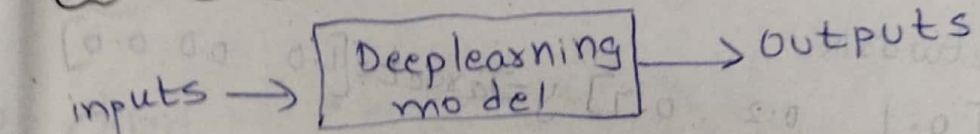
After few epochs

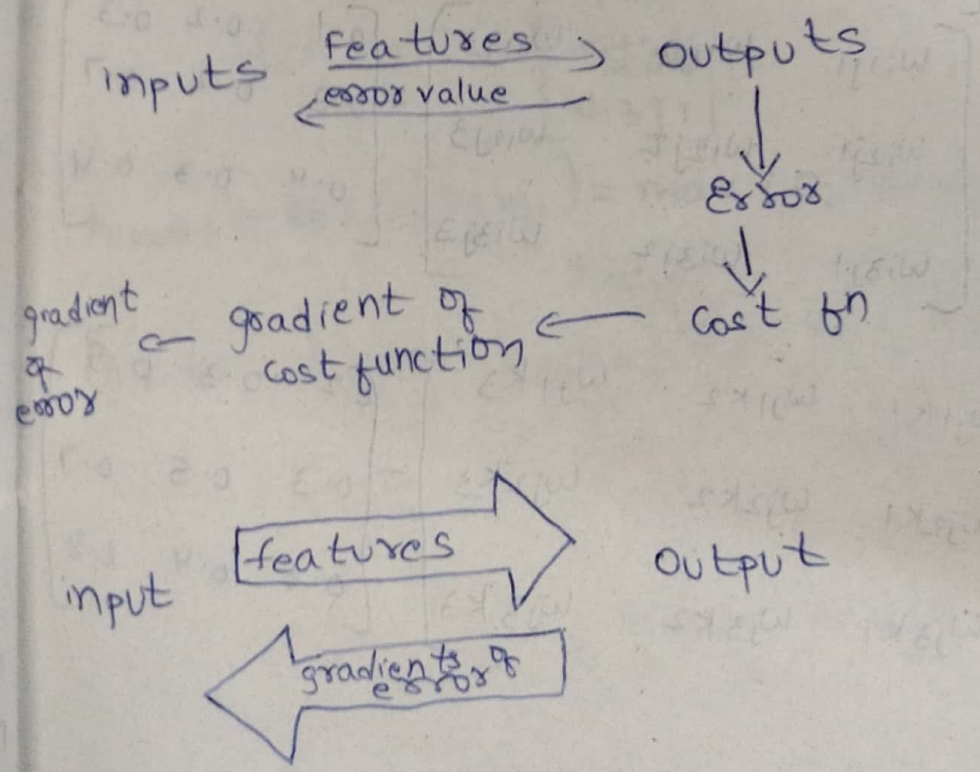
$\hat{y}_i$
0.938
0.028
0.036

→ loss = 0.095
Accuracy = 99.05 %

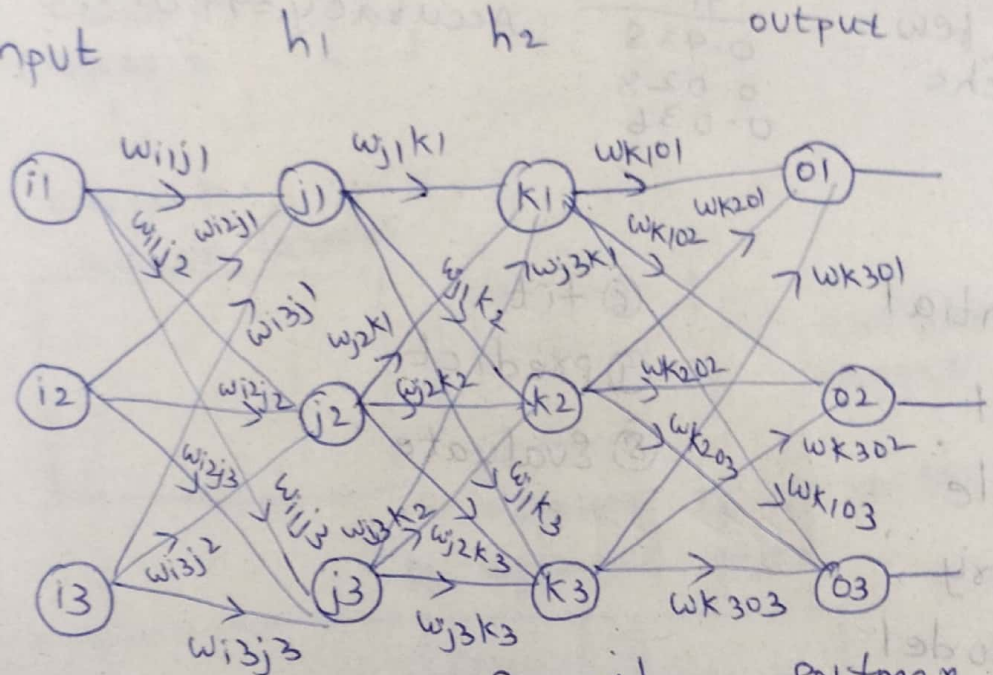
① Sequential
② Model
③ Compile
④ Summary
⑤ plot_model
⑥ fit
⑦ predict
⑧ Evaluate

Back propagation algorithm :-





- 4 layers: i/p, h_1, h_2, o/p
- 3 neurons for each layer
- For simplicity all biases = 0



Relu Sigmoid softmax

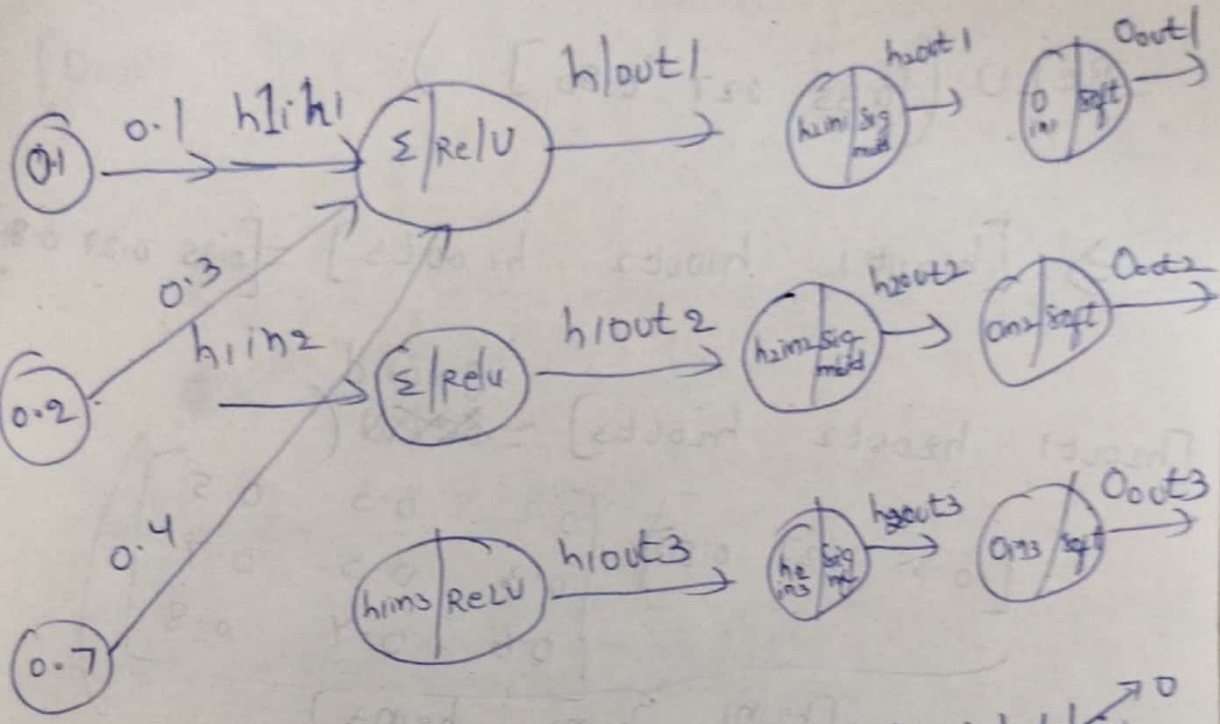
This is a feed forward fully connected neural network

Input = $[0.1 \quad 0.2 \quad 0.7]^T$ y = Target = $[1.0 \quad 0.0 \quad 0.0]^T$

$$w_{ij} = \begin{bmatrix} w_{i1j1} & w_{i1j2} & w_{i1j3} \\ w_{i2j1} & w_{i2j2} & w_{i2j3} \\ w_{i3j1} & w_{i3j2} & w_{i3j3} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.3 & 0.2 & 0.7 \\ 0.4 & 0.3 & 0.4 \end{bmatrix}$$

$$w_{jk} = \begin{bmatrix} w_{j1k1} & w_{j1k2} & w_{j1k3} \\ w_{j2k1} & w_{j2k2} & w_{j2k3} \\ w_{j3k1} & w_{j3k2} & w_{j3k3} \end{bmatrix} = \begin{bmatrix} 0.2 & 0.3 & 0.5 \\ 0.3 & 0.5 & 0.7 \\ 0.6 & 0.4 & 0.8 \end{bmatrix}$$

$$W_{K1} = \begin{bmatrix} w_{K1I1} & w_{K1I2} & w_{K1I3} \\ w_{K2I1} & w_{K2I2} & w_{K2I3} \\ w_{K3I1} & w_{K3I2} & w_{K3I3} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.4 & 0.8 \\ 0.3 & 0.7 & 0.2 \\ 0.5 & 0.2 & 0.9 \end{bmatrix}$$



$$h_{1in1} = i_1 * w_{i1j1} + i_2 * w_{i2j1} + i_3 * w_{i3j1} + b_{j1}$$ (0)

$$h_{1in1} = \begin{bmatrix} i_1 & i_2 & i_3 \end{bmatrix} \begin{bmatrix} w_{i1j1} \\ w_{i2j1} \\ w_{i3j1} \end{bmatrix}$$

$$h_{1out1} = ReLU(h_{1in1}) = \max(0, h_{1in1})$$

$$\begin{bmatrix} h_{1out1} & h_{1out2} & h_{1out3} \end{bmatrix} = Relu\left(\begin{bmatrix} i_1 & i_2 & i_3 \end{bmatrix} \begin{bmatrix} w_{i1j1} & w_{i1j2} & w_{i1j3} \\ w_{i2j1} & w_{i2j2} & w_{i2j3} \\ w_{i3j1} & w_{i3j2} & w_{i3j3} \end{bmatrix} \right)$$

$$= ReLU\left(\begin{bmatrix} 0.1 & 0.2 & 0.7 \end{bmatrix} \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.3 & 0.2 & 0.7 \\ 0.4 & 0.3 & 0.9 \end{bmatrix} \right)$$

$$= ReLU\left(\begin{bmatrix} 0.35 & 0.27 & 0.80 \end{bmatrix} \right)$$

$$\Rightarrow \begin{bmatrix} h_{1out1} & h_{1out2} & h_{1out3} \end{bmatrix} = \begin{bmatrix} 0.35 & 0.27 & 0.80 \end{bmatrix}$$

$$\begin{bmatrix} h_{2out1} & h_{2out2} & h_{2out3} \end{bmatrix} = \text{sigmoid}\left(\underbrace{\begin{bmatrix} 0.35 & 0.27 & 0.80 \end{bmatrix} \begin{bmatrix} 0.2 & 0.3 & 0.5 \\ 0.3 & 0.5 & 0.7 \\ 0.6 & 0.4 & 0.8 \end{bmatrix}}_{\begin{bmatrix} h_{2in1} & h_{2in2} & h_{2in3} \end{bmatrix}} \right)$$

$$h_{2out1} = \frac{1}{1 + e^{-h_{2in1}}}$$

$$\begin{bmatrix} h_{2out1} & h_{2out2} & h_{2out3} \end{bmatrix} = \text{sigmoid}\left(\begin{bmatrix} h_{2in1} & h_{2in2} & h_{2in3} \end{bmatrix} \right)$$

$$= \text{sigmoid}\left(\begin{bmatrix} 0.631 & 0.56 & 1.004 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 0.6527 & 0.6364 & 0.7318 \end{bmatrix}$$

$$\begin{bmatrix} O_{out1} & O_{out2} & O_{out3} \end{bmatrix} = Softmax\left(\underbrace{\begin{bmatrix} 0.6527 & 0.6364 & 0.7318 \end{bmatrix} \begin{bmatrix} 0.1 & 0.4 & 0.8 \\ 0.3 & 0.7 & 0.2 \\ 0.5 & 0.2 & 0.9 \end{bmatrix}}_{\begin{bmatrix} O_{in1} & O_{in2} & O_{in3} \end{bmatrix}} \right)$$

$$= Softmax\left(\begin{bmatrix} \end{bmatrix} \right)$$

softmax =

$O_{out1} = softmax([O_{in1} \quad O_{in2} \quad O_{in3}])$

$= \frac{e^{O_{in1}}}{e^{O_{in1}} + e^{O_{in2}} + e^{O_{in3}}}$

$[O_{out1} \quad O_{out2} \quad O_{out3}] = \left[\frac{e^{O_{in1}}}{e^{O_{in1}} + e^{O_{in2}} + e^{O_{in3}}} \quad \frac{e^{O_{in2}}}{e^{O_{in1}} + e^{O_{in2}} + e^{O_{in3}}} \quad \frac{e^{O_{in3}}}{e^{O_{in1}} + e^{O_{in2}} + e^{O_{in3}}} \right]$

$= [0.223 \quad 0.3064 \quad 0.470]$

$output^T = softmax(sigmoid(ReLU(Input^T w_{ij}) w_{jk}) w_{kl})$

$output = f(x; \theta)$

$S.GD = \theta^{(2)} = \theta^{(1)} - \eta \nabla_\theta J(\theta)$

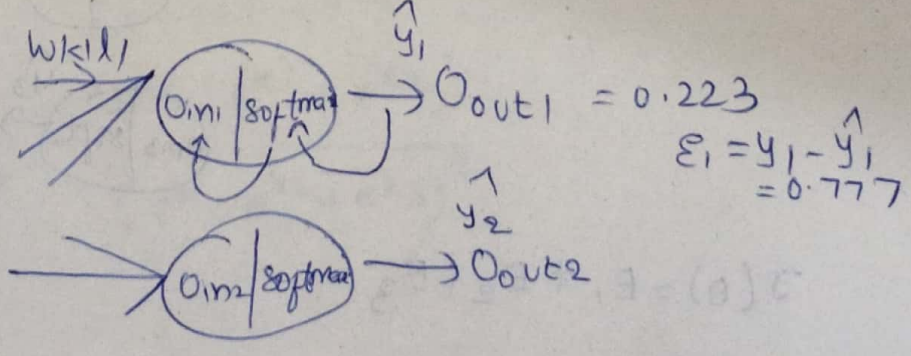
Optimizer

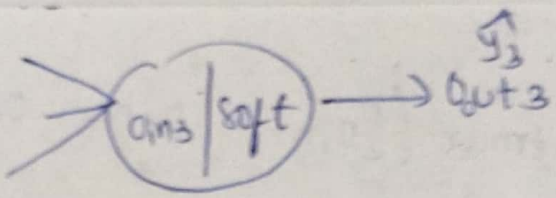
Gradient descent: $\theta^{(2)} = \theta^{(1)} - \eta \nabla_\theta J(\theta)$; $\eta = 0.01$

Cost function: Categorical cross-entropy

$J(\theta) = -\sum_{i=1}^{3} Target_i \log(Out_i)$

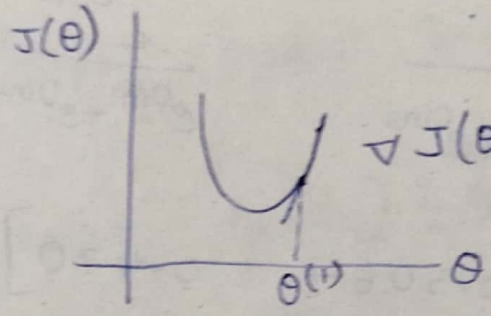
$J(\theta) = -\sum_{i=1}^{3} y_i \log(\hat{y_i})$





$$\frac{\partial \hat{y}_1}{\partial w_{kil1}} = \frac{\partial \hat{y}_1}{\partial \phi} \times \frac{\partial \phi}{\partial o_{in1}} \times \frac{\partial o_{in1}}{\partial w_{kil1}}$$

$$O_{out1} = \hat{y}_1 = \text{softmax}(h_{2out1} w_{ki11} + h_{2out2} w_{k211} + h_{2out3} w_{k311})$$

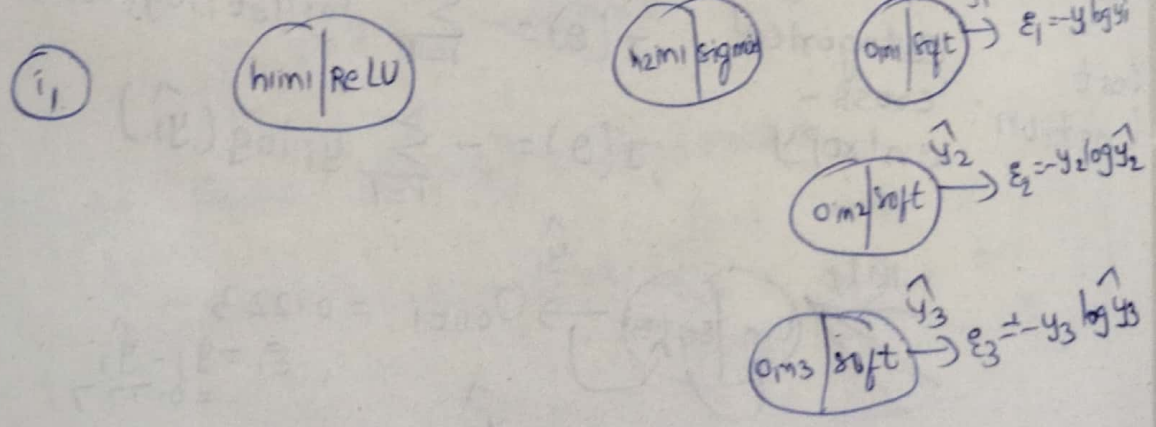


$$\theta^{(2)} = \theta^{(1)} - \eta \nabla J(\theta^{(1)})$$

$$\frac{\partial J(\theta)}{\partial \hat{y}_1} = \frac{\partial}{\partial \hat{y}_1}\left[-y_1 \log \hat{y}_1 - y_2 \log \hat{y}_2 - y_3 \log \hat{y}_3\right]$$

$$= \left[-y_1 \left(\frac{1}{\hat{y}_1}\right) - 0 - 0\right] = -y_1/\hat{y}_1$$

$$\frac{\partial J(\theta)}{\partial w_{kil1}} = \frac{\partial J(\theta)}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial \phi} * \frac{\partial \phi}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial w_{kil1}}$$



$$J(\theta) = E_1 + E_2 + E_3$$

$J(\theta) = E_1 + E_2 + E_3$

$= -y_1 \log \hat{y}_1 - y_2 \log \hat{y}_2 - y_3 \log \hat{y}_3$

$J(\theta)$: how the cost value changes when all the parameters gets updated.

$f(x;\theta) =$ ~~ReLU(InputTWij)~~

softmax (sigmoid (ReLU (Input^T W_{ij}) W_{jk}) W_{kl})

$$\frac{\partial J(\theta)}{\partial W_{kl}} = \frac{\partial J(\theta)}{\partial \text{softmax}} * \frac{\partial \text{Softmax}}{\partial O_{in_1}} * \frac{\partial O_{in_1}}{\partial W_{kl}}$$

Derivative of softmax:-

$$y_1 = \frac{e^{x_1}}{e^{x_1} + e^{x_2} + e^{x_3}}$$

$$y_2 = \frac{e^{x_2}}{e^{x_1} + e^{x_2} + e^{x_3}}$$

$$y_3 = \frac{e^{x_3}}{e^{x_1} + e^{x_2} + e^{x_3}}$$

$$\frac{\partial y_1}{\partial x_1} = \frac{\partial}{\partial x_1}\left(\frac{e^{x_1}}{e^{x_1} + e^{x_2} + e^{x_3}}\right) = \frac{(e^{x_1} + e^{x_2} + e^{x_3})e^{x_1} - e^{x}(e^{x_1})}{(e^{x_1} + e^{x_2} + e^{x_3})^2}$$

$$\frac{\partial y_1}{\partial x_1} = \frac{e^{x_1}(e^{x_2} + e^{x_3})}{(e^{x_1} + e^{x_2} + e^{x_3})^2}$$

$$\frac{\partial y_2}{\partial x_2} = \frac{e^{x_2}(e^{x_1} + e^{x_3})}{(e^{x_1} + e^{x_2} + e^{x_3})^2}$$

$$\frac{\partial y_3}{\partial x_3} = \frac{e^{x_3}(e^{x_1} + e^{x_2})}{(e^{x_1} + e^{x_2} + e^{x_3})^2}$$

derivative of sigmoid

$$y = \frac{1}{1+e^{-x}}$$

$$\frac{\partial}{\partial x}\left(\frac{1}{x}\right) = -\frac{1}{x^2}$$

$$y = \frac{e^x}{1+e^x}$$

$$\frac{\partial y}{\partial x} = \frac{(1+e^x)e^x - e^x e^x}{(1+e^x)^2}$$

(eq)

$$\frac{\partial y}{\partial x} = -\frac{1}{(1+e^{-x})^2} \cdot e^{-x}(-1)$$

$$= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{(1+e^{-x})}\left[\frac{e^{-x}}{1+e^{-x}} + 1 - 1\right]$$

$$= \frac{1}{(1+e^{-x})}\left[1 - \frac{1}{1+e^{-x}}\right] = \text{sigmoid}(1-\text{sigmoid})$$

Derivation of ReLU

$$\text{ReLU} = \max(0, x)$$

$$= \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\frac{\partial \text{ReLU}}{\partial x} = \begin{cases} 1 & x > 0 \\ 0 & \text{elsewhere} \end{cases}$$

$$w_{kl}' = w_{kl} - \eta \delta w_{kl}$$

$$w_{jk}' = w_{jk} - \eta \delta w_{jk}$$

$$w_{ij}' = w_{ij} - \eta \delta w_{ij}$$

$$\delta w_{kl} = \begin{bmatrix} \frac{\partial E1}{\partial w_{k1l1}} & \frac{\partial \varepsilon_2}{\partial w_{k1l2}} & \frac{\partial \varepsilon_3}{\partial w_{k1l3}} \\ \frac{\partial \varepsilon_1}{\partial w_{k2l1}} & \frac{\partial \varepsilon_2}{\partial w_{k2l2}} & \frac{\partial \varepsilon_3}{\partial w_{k2l3}} \\ \frac{\partial \varepsilon_1}{\partial w_{k3l1}} & \frac{\partial \varepsilon_2}{\partial w_{k3l2}} & \frac{\partial \varepsilon_3}{\partial w_{k3l3}} \end{bmatrix}$$

chain-rule

$$\frac{\partial E_1}{\partial w_{k11}} = \frac{\partial E_1}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial w_{k11}}$$

↓ w.r.t activation function

$$E_1 = -y_1 \log \hat{y}_1$$

$$\frac{\partial E_1}{\partial \hat{y}_1} = \frac{\partial}{\partial \hat{y}_1}(-y_1 \log \hat{y}_1) = -y_1\left(\frac{1}{\hat{y}_1}\right)$$

$$\frac{\partial \hat{y}_1}{\partial o_{in1}} = \frac{e^{\hat{y}_1}\left(e^{\hat{y}_2} + e^{\hat{y}_3}\right)}{\left(e^{\hat{y}_1} + e^{\hat{y}_2} + e^{\hat{y}_3}\right)^2}$$

$$\frac{\partial o_{in1}}{\partial w_{k11}} = \frac{\partial}{\partial w_{k11}}\left(h_{2out1} * w_{k11} + h_{2out2} * w_{k21} + h_{2out2} * w_{k31}\right)$$

$$\frac{\partial o_{in1}}{\partial w_{k11}} = h_{2out1}$$

$$\delta w_{jk} = \begin{cases} \dfrac{\partial E}{\partial w_{j1k1}} \\ \\ \\ \end{cases}$$

$$\delta w_{k1} = \begin{cases} \dfrac{\partial E_1}{\partial w_{k11}} \\ \\ \\ \end{cases}$$

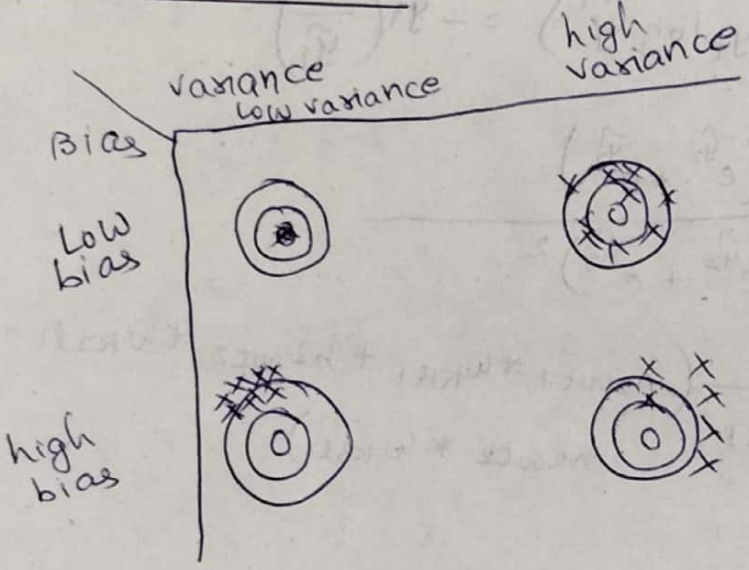
UNIT-II

*Regularization & Optimization:-

Regularization

what is meant by Generalisation?

Bias and Variance



-Generalisation is the ability of model to perform well on training & testing data.

- Regularization reduce the generalisation/test error possibly at the expense of training error

-Regularization is the strategy to reduce the test error possibly at the expense of increased training error.

-It is any modification that we make to any learning algorithm that is intended to reduce its generalisation error but not its training error.

-There are mainly 3 strategies used by regularisation

① Constraints based regularization

In this we put an extra constraint on

the model such as adding restrictions on the parameter values.

② Penalties based regularization
In this we add extra terms in the objective/cost function

③ Ensemble methods
In this we combine multiple hypothesis that explains the training data.

- Any of these regularization methods works by trading increased bias for reduced variance
- An effective regularizer is one that makes a profitable trade off such as reducing variance significantly while not overly increasing the bias

Parameter Norm Penalties

Many regularisation approaches

J - cost fn
θ - parameters wt
x - training set
y - labels

$$\tilde{J}(\theta; x, y) = J(\theta; x, y) + \alpha \Omega(\theta)$$

$\tilde{J}$ → cost function of the regularised model

J → cost function of an unregularized model

$\Omega(\theta)$ → Penalty term which is a function of only the model parameters. This term is nothing to do with training data set

α → hyperparameter that weights the contribution of penalty term

when $\alpha = 0$ ⇒ no regularization

higher the value of α ⇒ more the regularization

Consider an unregularized model

→ We try to calculate w^* at which $J(\theta; x, y)$ is min

In regularised model,

Our goal becomes

→ find $\tilde{w}$ at which $\tilde{J}(\theta; x, y)$ is min

- this may increase data term
- decreased by reducing penalty term.

Data & penalty terms are independent. So that they can ↑ or ↓.

- Data → depends on features
- penalty → depends on parameter

Ultimately over goal is to minimize $\tilde{J}(\theta; x, y)$

Different Regularisation functions:-

P-Norm Lp (or) $L^p = \left(\sum_i |\theta_i|^p\right)^{1/p} = \|\theta\|_p$

where $p = 0.5, 1, 2, 3, \ldots$ any positive number

L^2 Norm Regularization (OR) Ridge Regularization

$$\|\theta\|_2 = \left(\sum_i |\theta_i|^2\right)^{1/2} = \theta\theta^T$$

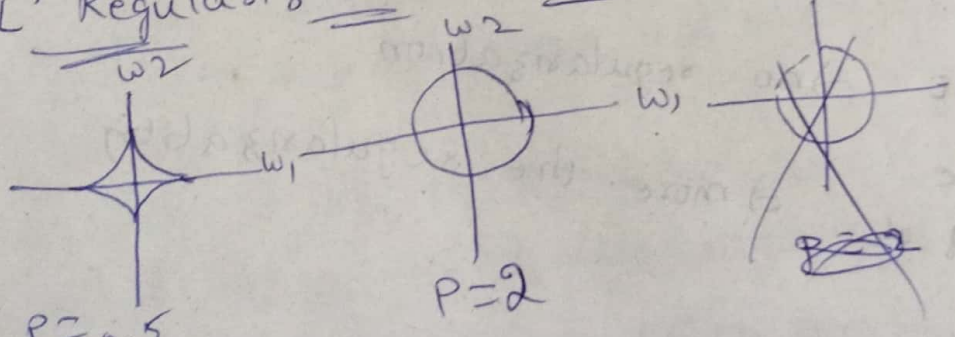
$$\tilde{J}(\theta; x, y) = J(\theta; x, y) + \alpha\theta\theta^T$$

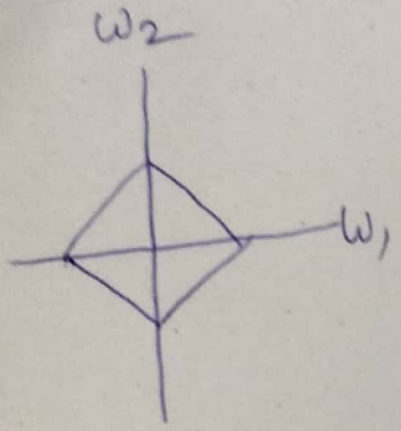
For simplicity ignore the bias parameter

$$\tilde{J}(w; x, y) = J(w; x, y) + \alpha w w^T$$

GD: $w \leftarrow w - \nabla_w \tilde{J}(w; x, y)$

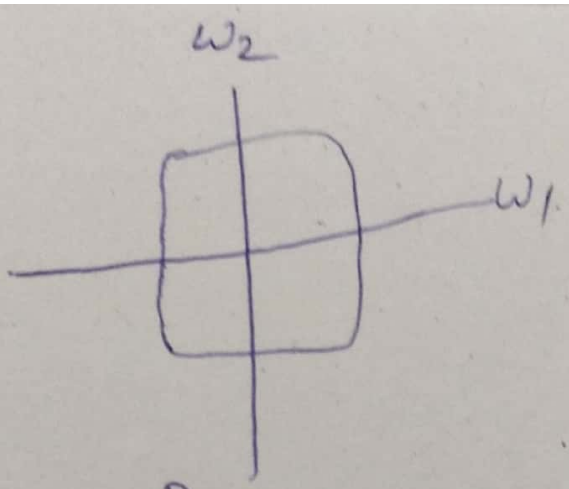
L^1 Regularization (OR) Lasso Regularization





P = 1

Lasso



P = 4