Date : 12/02/2024

DL Defn : Computational Graph

Visible layer (pixels) → 1st hidden layer (edges) → 2nd hidden layer (corners contours edges)

Output (Object Identity) ← 3rd hidden layer (object parts)

NN evolved in the year 1952.

1952 - SGD

1958 - Perceptron
- Learnable weights

Block region of AI (N-L Problem - XOR)

1986 - Backpropagation
- Multi-Layer Perceptron

1995 - Deep convolutional NN
- Digit Recognition.

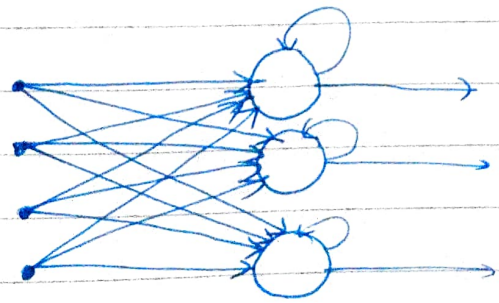
Issues at that time : Database availability and computationally efficient machines.

1. Big Data
- Larger datasets
- Easier collection & storage

2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software
- Improved Techniques
- New models
- Toolboxes

Recurrent Neural Network :



Eg :- LSTM (Long Short-Term Memory)

Gated Recurrent Unit (GRU)

Gradient Descent (GD)

→ Iterative, first-order, optimization algorithm

Function requirements:
→ Differentiable $w \leftarrow w - \eta \cdot \frac{\partial}{\partial w}(x \in w)$ → differentiable
→ convex.
↳ line segment connecting two segments fns.'s points lies on or above the curve.

$f(\lambda x_1 + (1-\lambda) x_2) =$

Mathematical way to find: second derivative > 0.

$f(x) = x^2 - x + 3$

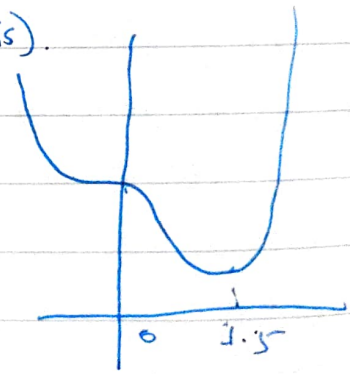
$f'(x) = 2x - 1$

$f''(x) = 2 > 0$ (convex fn.).

→ Saddle Points

→ Quasi-convex functions (minimax points).

eg:- $f(x) = x^4 - 2x^3 + 2$.



$\frac{d}{dx}(f(x)) = 4x^3 - 6x^2$

$= 4x^2(x - 1.5)$

$\frac{d^2}{dx^2}(f(x)) = 12x^2 - 12x = 12x(x-1)$

$x = 0, 1.5$

$x = 0$, $f''(x) = 0$.

$x = 1.5$, $f''(x) = 9$.

$x < 0$: Convex

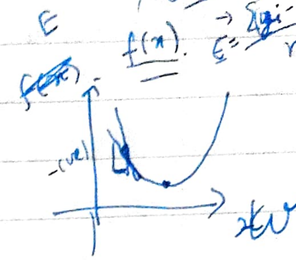
$0 < x < 1$: concave.

$x > 1$: Convex.

Date :

Intuitively, gradient is the slope of the function at a particular direction.

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_d}(p) \end{bmatrix}$$ for n-dimensional functions.



→ opp. direction to max. change.

$$W = W - \eta \cdot \frac{\partial E}{\partial w}$$

$\frac{\partial E}{\partial w} =$

$$w = w - \eta \cdot \frac{\partial E}{\partial w}$$

Algorithm Gradient Descent :

Basic / Vanilla Gradient Descent

13/02/2024

1) Initialize random weight, w
2) Do until converge
3) $w \leftarrow w - \eta \cdot \frac{\partial E}{\partial w}$
4) Until error is minimized.

criterion

$$W_{new} \leftarrow W_{old} - \eta \cdot \frac{\partial J(w)}{\partial w}$$

5) Return 'w'

Stochastic Gradient Descent (SGD)

1) Initialize w
2) Pick sample 'i' out of 'n' samples
3) Do
4) $w \leftarrow w - \eta \cdot \frac{\partial J_i(w)}{\partial w}$
5) Until $J(w)$ is minimized.
6) Return 'w'

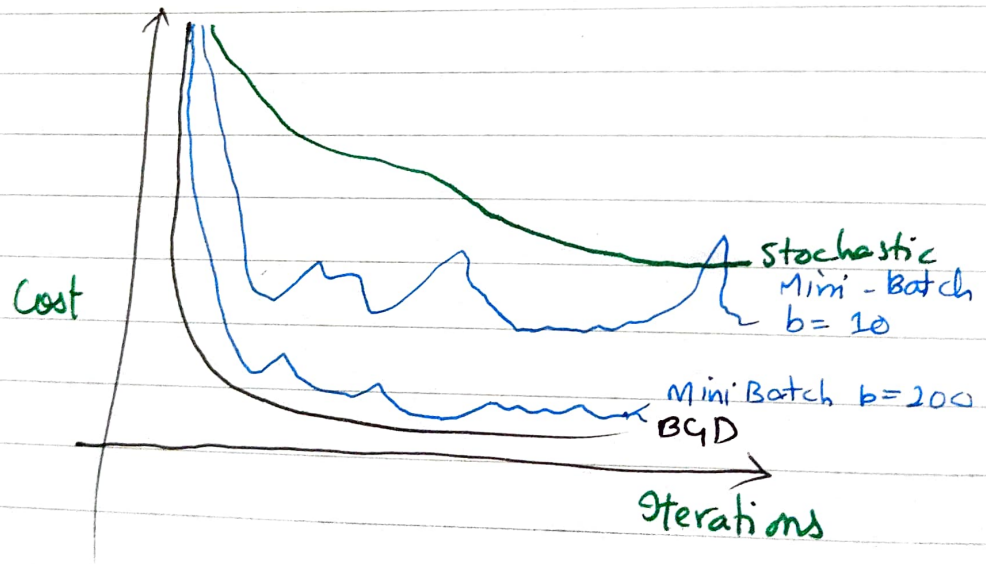
Mini Batch Gradient Descent

1) Initialize 'w'
2) Pick some sample of batch 'B'
3) Do
4) $w \leftarrow w - \eta \cdot \sum_{i=1}^{B} \frac{\partial J_i(w)}{\partial w}$
5) Until $J(w)$ is minimized
6) Return 'w'

BGD : $\frac{\partial J(w)}{\partial w}$ → computationally intensive to compute

SGD: Faster to compute gradient, but noisy.

Error minimization with GD :-



LOGISTIC REGRESSION :

$y = mx + c$.

$h_\theta(x) = \theta_0 + \theta_1 x$

$x = x_1, x_2, \ldots$

$h_\theta(x) = \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$

$\theta_1 = m$, $\theta_2 = c$.

X = given I.V.

$\hat{y}$ = Predicted O/P

y = Actual O/P

n = no. of samples.

$h_\theta(x) = \theta_0 + \theta_1 x \Rightarrow$ univariate linear regression.

14/02/2024

Hypothesis function θ Cost Function θ

$$y = mx + c$$

$$\rightarrow h_\theta(x) = \theta_0 + \theta_1 x_1$$

x - Input

y - True Label

$h_\theta(x)$ - Predicted value

$\theta = ? \mid h_\theta(x) = y.$

$$\frac{\partial E}{\partial \theta} = \frac{\partial}{\partial \theta} E(\theta_0, \theta_1) = \frac{1}{2}\sum_{i=1}^{n} (h_\theta(\dot{x}) - y^i)^2$$

$$= \frac{1}{2}\sum_{i=1}^{n} (h_\theta(x^i) - y^i)^2$$

$$h_\theta(x_i) = \theta_0 + \theta_1 x_i$$

Date : 11/02/2024

$$\frac{\partial E}{\partial \theta_0} = \frac{2}{2} \sum_{i=1}^{n} (h_\theta(x^i) - y^i) \cdot \frac{\partial}{\partial \theta_0} (h_\theta(x^i) - y^i)$$

(y^i: True label)

$$\boxed{\frac{\partial E}{\partial \theta_0} = \sum_{i=1}^{n} (h_\theta(x^i) - y^i)}$$

$$\frac{\partial E}{\partial \theta_1} = \frac{2}{2} \sum_{i=1}^{n} (h_\theta(x^i) - y^i) \cdot \frac{\partial}{\partial \theta_1} (\theta_0 + \theta_1 x^i - y^i)$$

$$\boxed{\frac{\partial E}{\partial \theta_1} = \sum_{i=1}^{n} (h_\theta(x^i) - y^i) \cdot x^i}$$

Multi-variate case: 'd' number of features:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \ldots + \theta_d x_d$$

Use vectorized approach: Compute $\vec{\theta}^T \vec{X}$.

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \qquad \vec{X} = \begin{bmatrix} \boxed{1} \\ x_1 \\ \vdots \\ x_d \end{bmatrix} \quad (x_0)$$

$$\theta^T X = \begin{bmatrix} \theta_0 & \theta_1 & \ldots & \theta_d \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = h_\theta(x).$$

Gradient Descent Algorithm:

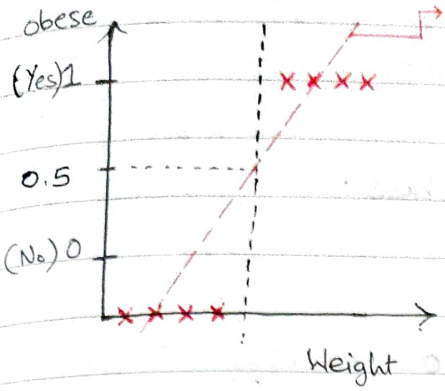
Repeat until convergence {

$$\theta_i := \theta_i - \alpha \cdot \frac{\partial}{\partial \theta_i} (E(\theta_0, \theta_1, \ldots, \theta_d)), \quad 0 \le i \le d, \; i \in Z^+$$

}

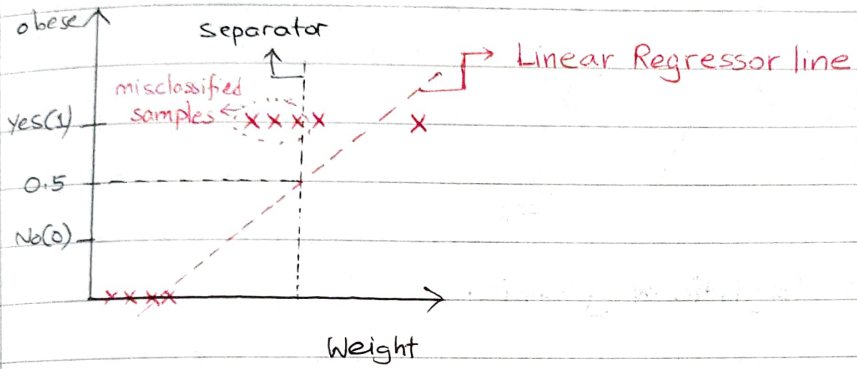
Date :

Issue with Linear regression: Outliers



If $h_\theta(x) \geq 0.5$, predict $y = 1$.

If $h_\theta(x) < 0.5$, predict $y = 0$.



Classification $y = 0$ or 1.

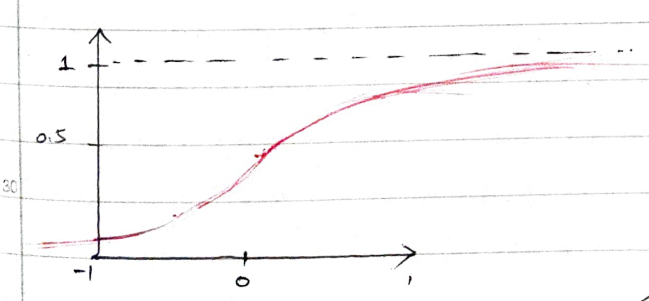
But here $h_\theta x$ can be > 0 or < 1.

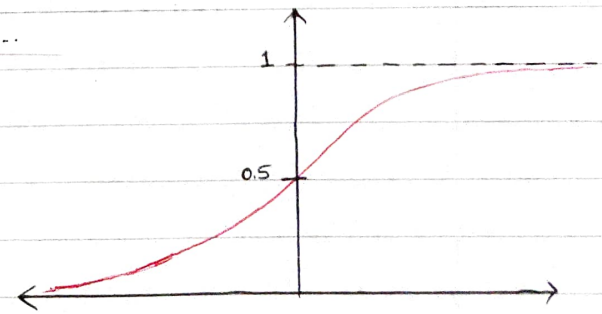
So, Linear Regression is not very suitable for classification.

Hypothesis of Logistic Regression : $0 \leq h_\theta(x) \leq 1$.

Previously, $h_\theta(x) = \theta^T x$.

Now, $h_\theta(x) = g(h_\theta(x))$ and $g(z) = \frac{1}{1+e^{-z}}$





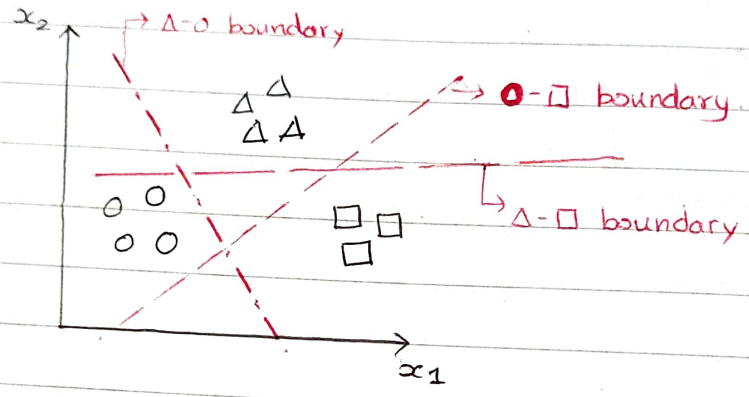
Date : 14/02/2024

Trained weight is also known as "Frozen weight".

For testing, compute $P(y=1 \mid x, \theta)$

Also, $P(y=0 \mid x, \theta) = 1 - P(y=1 \mid x, \theta)$

Logistic Regression : Multi-class :



Linear vs Logistic Regression

Linear Regression	Logistic Regression
1) Continuous variables	

Date : 15/02/2024

How many features (N) = 5

Target (C) = {0, 1} # = 2.

Confusion Matrix : where the model gets confused.

Predicted \ Actual	T	F
T	TP	FP
F	FN	TN

$$\text{Accuracy} = \frac{\text{Total correctly classified samples}}{\text{Total samples}}$$

$$= \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

Spam Detection: Precision is an important metric.

TN : 570

TP : 350

FP : 50

FN : 30

$$\text{Accuracy} = \frac{570 + 350}{}$$

$$\text{Precision} = \frac{350}{350 + 50}, \quad \text{Recall} = \frac{350}{30 + 350}.$$

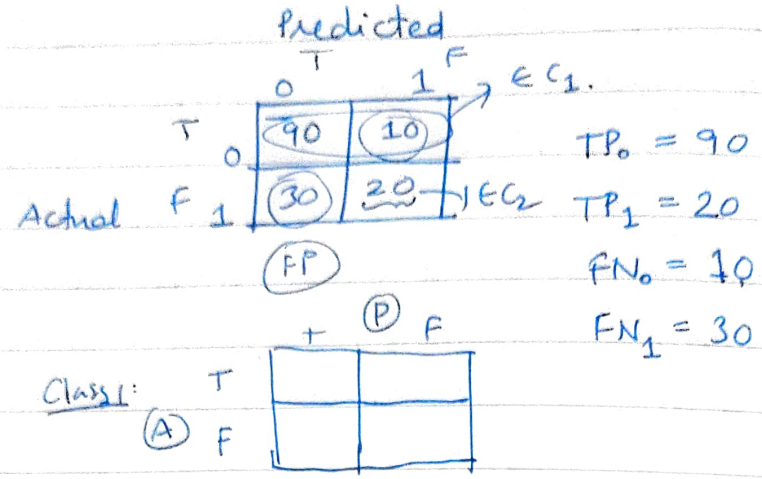
Multi-class :

Three types of averaging.

macro:

compute for each class and take weighted average of them.

Date :



$P_0 = \frac{90}{100}$, ~~$P_1 = \frac{20}{100}$~~ $= 0.9$

$P_1 = \frac{20}{30+20} = 0.4$

$P_{macro} = \frac{0.9 + 0.4}{2} = 0.65$

$P_{weighted} = P_0 \times support_0 + P_1 \times support_1$

$= .9 \times \frac{100}{150} + 0.4 \times \frac{50}{150}$

(i) ROC (ii) Cohen's kappa (iii) Quad. weight kappa

$TP_A = 10$, $TP_B = 15$, $TP_C = 8$, $TP_D = 5$.

$FP_A = 3$, $FP_B = 4$, $FP_C = 2$, $FP_D = 1$.

$FN_A = 3$, $FN_B = 3$, $FN_C = 2$, $FN_D = 2$.

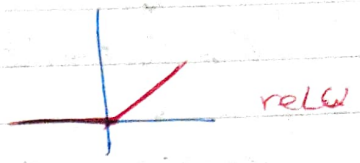
$TN_A = 32$, $TN_B = 25$, $TN_C = 35$, $TN_D = 40$.

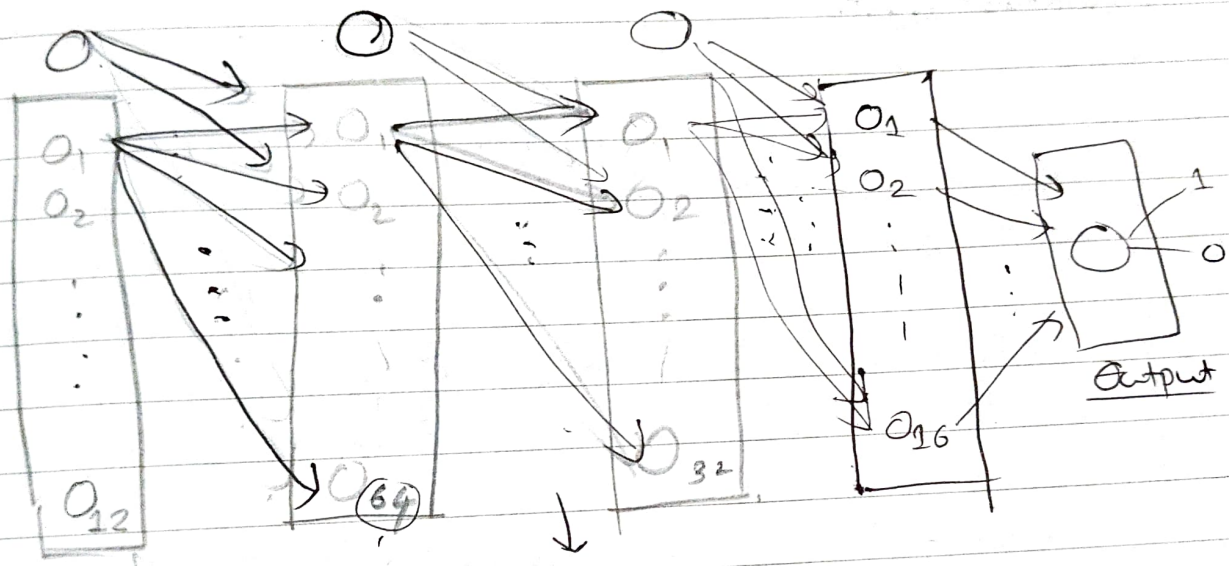
Date :

Hyperparameters - values are predetermined prior to start of learning process.

Eg:-

1) Learning Rate
2) Batch size — high → Comp. complexity ↑
   — small → Time ~~complexity~~ ↑ for execution.
3) epochs
4) Activation fns.
5) No. of layers / No. of neurons





# Trainable Parameters	# weight params $= 12 \times 64 + 1 \times 64$ $= 832$	$64 \times 32 + 1 \times 32$ $= 2080$	528	17

Date : 19/02/202

Vanishing and Exploding gradient Problem :-

activation function : non-linear, differentiable

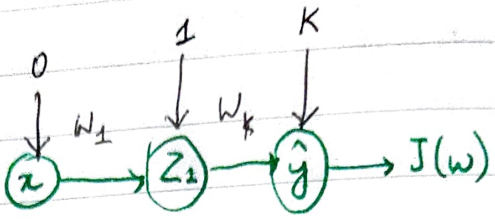
deep learning

⇓

Deep Neural Network

⇓

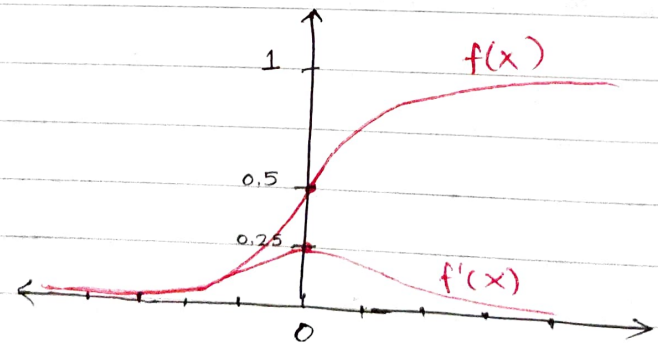
100 - 150 layers



$$w_{new} \leftarrow w_{old} - \eta \cdot \frac{\partial E}{\partial w}$$ → $\partial J(w)$

$$\frac{\partial J(w)}{\partial w_1} = \frac{\partial J(w)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

$$f(x) = \frac{1}{1+e^{-x}}, \quad f'(x) = f(x)(1 - f(x))$$



Maximum value of derivative of sigmoidal function is

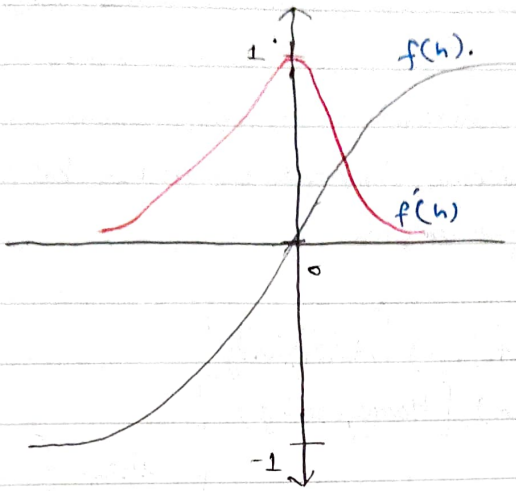
Say $\frac{\partial L}{\partial w} = 0.015$ $(0.2 \times 0.15 \times 0.05)$

$w_{new} = w_{old} - (1).\ 0.015 = 2.4985$

Date : 19/02/2024

→ The gradient of activation function is small, that's why the issue occurs.

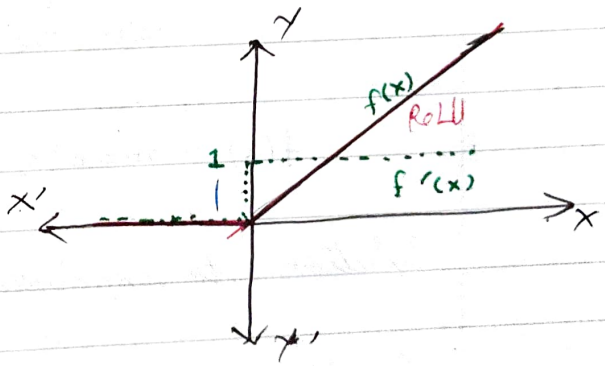
→ tanh.



Again, it can result in vanishing gradient problem.

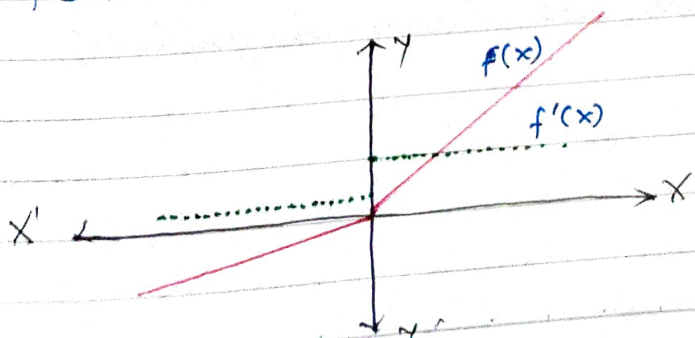
→ ReLU (Rectified Linear Unit), Leaky ReLu, Parametric ReLU.
(saturates in one direction).

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



If $f(x) \leq 0$, then, $w_{old} = w_{new}$.
This condition is called as dead neuron.

Modification done to solve dead neuron problem:- Leaky ReLU.
(Not Vanishing Gradient)



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

$\alpha \sim 0.01$

Date : 19/02/2024

$O_1 = W_k \cdot z_1$

$\hat{y} = \underbrace{f(W_k \cdot z_1)}_{\text{sigmoid}}$

→ keeps oscillating, doesn't converge.

→ The weight vector changes too much because of large value of weight vector.

20/02/2024

$f(x+y) = f(x) + f(y)$ [Additivity]

$f(ax) = a f(x)$. [Homogeneity]

$f(ax+by) = af(x) + bf(y)$.

} Linear function

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$ ⇒ Step fn. ⇒ not continuous/differentiable

(Also gradient = 0 since value in constant in the two intervals.)

$f(x) = \max(0, x)$.

Solution to exploding gradient :

Gradient clipping :

If $\|g\| >$ threshold

$g \leftarrow \dfrac{\text{threshold} \times g}{\|g\|}$

[Same direction but lesser magnitude].

Date : 20/02/2024

In Leaky ReLU, $\alpha = 0.01$,
If $\alpha \neq 0.01$, then it is parameterized ReLU.

Heaviside step fn. $f(n)$

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

$f'(x) = \delta(x)$, where $\delta(x)$ is the dirac delta function.

ReLU function

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

$$f'(x) = \begin{cases} 0, & x < 0 \\ \text{undefined}, & x = 0 \\ 1, & x = 0, \end{cases}$$

For practical purposes, $f'(x) = 0$ or 1 at $x = 0$ or subgradients are used (value in between 0 and 1).

Activation Functions

The purpose of activation fn. is to add non-linearity of the neuron.

Characteristics of activation functions:

- Continuous and differentiable
- Non-decreasing function
- Not always constant

Sigmoid - used in o/p layer for binary classification.

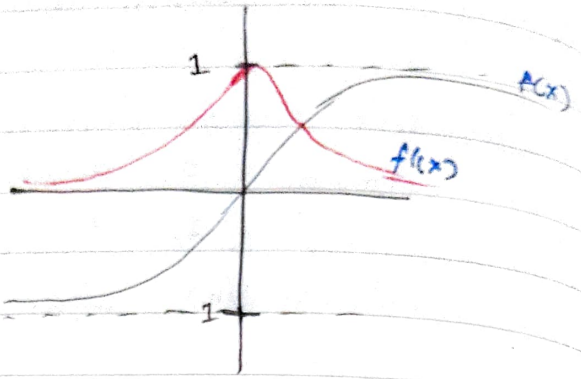
Date :

→ Hyperbolic tangent

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

$$f'(x) = 1 - f(x)^2$$

Vanishing gradient.



→ Inverse tangent

$$f(x) = \tan^{-1}(x)$$

$$f'(x) = \frac{1}{1+x^2}$$

→ ReLU

Most popular activation function since 2017.
It can result in dead neurons.

→ Leaky ReLU

Fixes the problem of dead neuron
Used only in hidden layers of ~~NN~~.

$$f(x) = \begin{cases} 0.01x, & x < 0 \\ x, & x \geq 0 \end{cases}$$

→ Parametric ReLU

$$f(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}, \quad \alpha \neq 0.01$$

Date :

Maxout function

$$f(\vec{x}) = \max_i x_i$$

$$\frac{\partial f}{\partial x_j} = \begin{cases} 1 & \text{for } j = \arg\max_i x_i \\ 0 & \text{for } j \neq \arg\max_i x_i \end{cases}$$

$[0.5 \quad 0.7, \quad 0.1] \longrightarrow [0 \quad 0 \quad 1] \Rightarrow$ One Hot vector.

Exponential Linear Unit (ELU).

Function

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

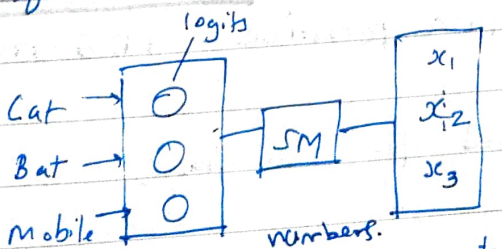
→ No dead neurons
→ O/P is zero centered (Mean = 0).

$$f'(\alpha, x) = \lambda \begin{cases} f(\alpha, x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

Identity function – for O/P layer

Use: Regression.

Softmax function



$0 \leq x_1, x_2, x_3 \leq 1$ s.t. $x_1 + x_2 + x_3 = 1$.

O/P from summation are called "logits"

$$f_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad i = 1 \ldots J.$$

Σ of f

Σ of f

$$\frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$, for n classes.

$f'(x) = J(\vec{x})(f_1 \ldots f_J(\vec{x}))$

Date :

Regression Model (0-1) (S.L.)

- → Prediction ($\hat{y}$)
- → Loss function.

  $E = (y - \hat{y})$

  (L_2) $MSE = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})^2$

  (L_1) $MAE = \frac{1}{n} \sum_{i=1}^{n} |(y - \hat{y})|$

- → Huber Loss function

Classification Model (S.L.)

- → classify / assign discrete value for the sample y_i: 0/1
- → Loss function

  Cross "Entropy" Loss

  KL-divergence: $\hat{y} \log(\hat{y}/y)$.

  Bhattacharya (dist b/w prob. dist).

  Cross Entropy Loss:

  $-\left(\sum_{i=1}^{c} y \log(\hat{y})\right)$ — y → ground truth.

  CCEL (M)
  BCEL (B)
  KL Div. (AE)

- → Hingle loss function.

For Binary Classification,

$$-\left(\sum_{i=1}^{c} y_i \log(\hat{y_i})\right)$$

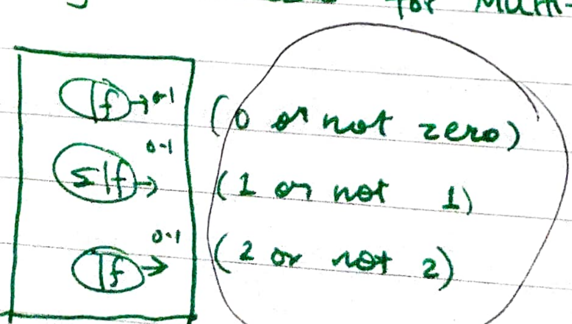
$$-\left(\sum_{i=1}^{2} y_i \log(\hat{y_i})\right) = -\left(y_1 \log(\hat{y_1}) + y_2 \log(\hat{y_2})\right)$$

$$= -\left(y_1 \log(\hat{y_1}) + (1 - y_1) \log(\hat{y_2})\right)$$

$$BCEL = -\left(y_1 \log(\hat{y_1}) + (1 - y_1) \log(1 - \hat{y_1})\right)$$

Robust outlier - Less Penalize - MAE
Misclassified Sample - More penalize - MSE - More quick weight updation.

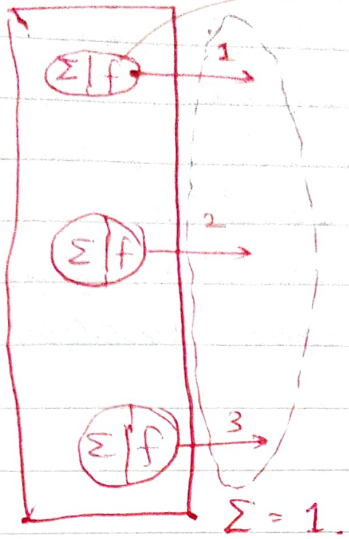
If sigmoid is used for Multi-class,



$\Sigma \neq 1$ always.

Date :

If softmax is used → S.M. (Softmax).

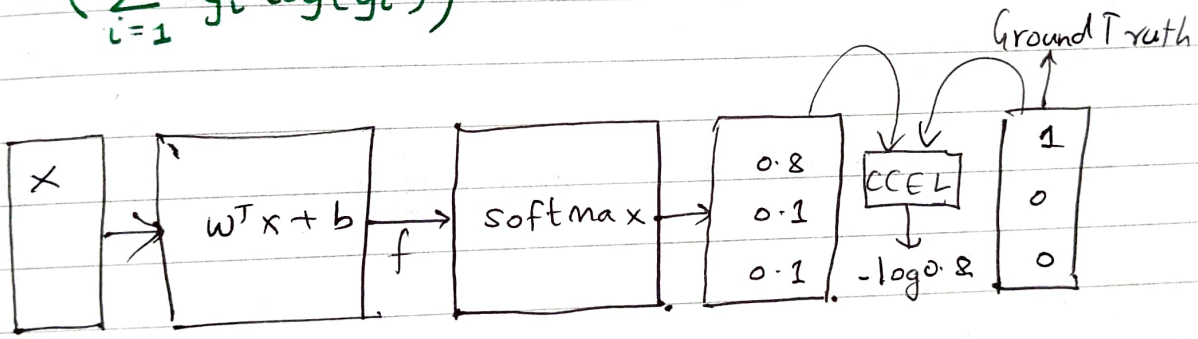


$$\frac{e^i}{\sum_{j=1}^{c} e^j} \quad (0-1).$$

$$\sum_{i=1}^{c} p_i = 1.$$

CCEL (Categorical Cross Entropy Loss).

$$-\left(\sum_{i=1}^{c} y_i \log(\hat{y}_i)\right)$$



Why cross entropy is used most commonly as compared to MSE?

Date : 26/02/2024

Optimization techniques

↳ Gradient Descent ⇒ Optimization Tech./Algo. to max./min. the criterion function $J(w)$

Loss function

→ $L/E = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$ MSE

→ $L/E = \sum_{i=1}^{n} |y_i - \hat{y}_i|$ MAE

→ $L/E = \sum_{i=1}^{n} y_i \log(\hat{y}_i)$

$\frac{\partial L}{\partial w}$

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

→ Batch G.D. (Computationally complex for large I/P size)
→ Stochastic G.D. (Longer Time to converge) [Image: → oscillating towards solution]
→ Mini Batch G.D.

BGD - 1 epoch = 1 iteration
SGD - 1 epoch = 10 k iterations
MBGD - 1000 - Mini Batch Size
1 epoch - $\frac{10k}{1000}$ = 10 iterations

[Assume I/P size - 10 k]

GD optimization with momentum:-

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

$$w_t \leftarrow w_{t-1} - \eta \cdot \frac{\partial L}{\partial w_{t-1}}$$

exponential weighted average.

t_1	t_2	. . .	t_n
a_1	a_2	. . .	a_n

$$V_{t1} = a_1$$

$$V_{t2} = \beta . V_{t1} + (1-\beta)\, a_2 . \quad \text{Usually, } \beta = 0.95.$$

More weight to previous than current if $\beta = 0.95$.

$$V_{t3} = \beta \cdot V_{t2} + (1-\beta) a_3$$

$$= \beta . (\beta . V_{t1} + (1-\beta) . a_2) + (1-\beta) a_3$$

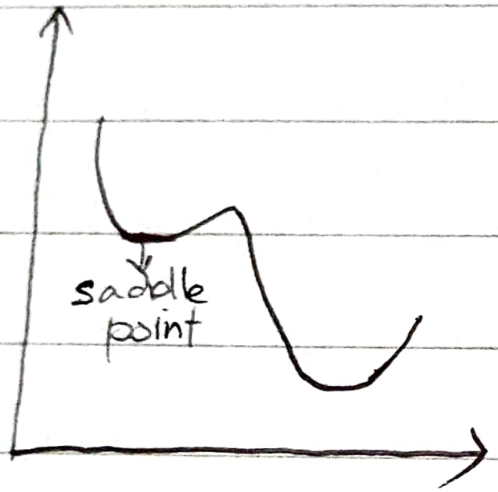
$$V_{t3} = \beta^2 V_{t1} + \beta (1-\beta)\, a_2 + (1-\beta)\, a_3$$

$$\boxed{V_{t3} = \beta^2 a_1 + \beta . (1-\beta)\, a_2 + (1-\beta) a_3}$$

$$w_t \leftarrow w_{t-1} - \eta . V_{\partial w_t}$$

$$V_{dw_t} \leftarrow \beta (V_{dw_{t-1}}) + (1-\beta) . \frac{\partial L}{\partial V_{dw_t}}$$

$$V_{dw_t} \leftarrow \beta (V_{db_{t-1}}) + (1-\beta) \left(\frac{\partial L}{\partial V_{db_t}} \right)$$



① Change in gradient component
② Change in learning rate
③ Change in both GC and LR.

11 Gradient Descent with Momentum.

$$W_t = W_{t-1} - \eta \cdot \frac{\partial L}{\partial W_{t-1}}$$

$$W_t = W_{t-1} - \eta \cdot m_t$$

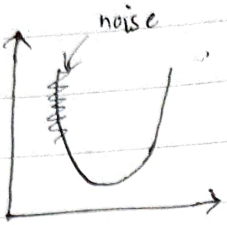
$$m_t = \underset{0.95}{\beta} \cdot m_{t-1} + \overbrace{(1-\beta)}^{0.05} \cdot \frac{\partial L}{\partial W_{t-1}}$$

It helps to avoid getting stuck

$$W_t \leftarrow W_{t-1} - \eta \cdot V_{dW_t}$$

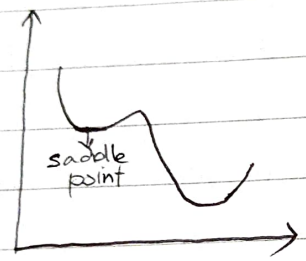
$$b \leftarrow b_{t-1} - \eta \cdot V_{\partial b_t}$$

Optimization Technique : w ; $J(w)$



$$W_t = W_{t-1} - \eta \cdot \frac{\partial L}{\partial W}$$

Saddle points (when multiple local minima are present)



① Change in gradient component
② Change in learning rate
③ Change in both GC and LR.

1.1 Gradient Descent with Momentum:

$$W_t = W_{t-1} - \eta \cdot \frac{\partial L}{\partial W_{t-1}}$$

$$W_t = W_{t-1} - \eta \cdot m_t$$

$$m_t = \underbrace{\beta}_{0.95} \cdot m_{t-1} + \underbrace{(1-\beta)}_{0.05} \cdot \frac{\partial L}{\partial W_{t-1}}$$

It helps to avoid getting stuck in saddle points.

Date :

This computation occurs at time $t_0, t_1, t_2, \ldots, t_n$.

How it comes?

1 epoch ⇒ more than one iteration

1.2 Nesterov Accelerated Gradient (NAG)

Not commonly used.