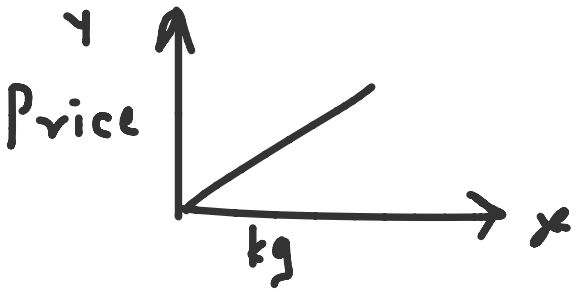
Deep Learning : ANN

Monday, September 6, 2021 5:44 PM



Y = mx + c where m-slope & c - intercept

$m = \frac{\Delta y}{\Delta x}$

If m = 0, y = c

W.I + b

In case of deep learning,

W - weights, I - inputs, b - bias based on Psychology

Inputs in human beings : See | Sound

↓ ↓

Image Text

If you don't know any question's answer, weight will be 0

Answer will be b(bias) based on Psychology

We pass this function through Activation function I_0^1 which activates thoughts between 0 & 1

Vector and Matrix

Types of Matrices

Row Matrix

$$\begin{pmatrix} a & b & c \end{pmatrix}$$

Column Matrix
Vector Matrix

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Zero Matrix
Null Matrix

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Diagonal Matrix

$$\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix}$$

Scalar Matrix

$$\begin{pmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & a \end{pmatrix}$$

Unit Matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Upper Triangular Matrix

$$\begin{pmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{pmatrix}$$

Lower Triangular Matrix

$$\begin{pmatrix} a & 0 & 0 \\ b & c & 0 \\ d & e & f \end{pmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

A B C

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2

$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}_{2 \times 3} \qquad A^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}_{3 \times 2}$$

If A = A^T : Symmetric Matrix

A^T = -A : Skew Symmetric

PROPERTIES OF MATRIX ARITHMETIC

(a) $A + B = B + A$ **(Commutative law for addition)**

(b) $A + (B + C) = (A + B) + C$ **(Associative law for add.)**

(c) $A(BC) = (AB)C$ **(Associative law for multiplication)**

(d) $A(B + C) = AB + AC$ **(Left distributive law)**

(e) $(A + B)C = AC + BC$ **(Right distributive law)**

(f) $A(B - C) = AB - AC$

(g) $(A - B)C = AC - BC$

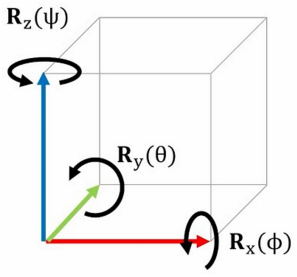
(h) $a(B + C) = aB + aC$

Elementary Rotations

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Jon Woolfrey

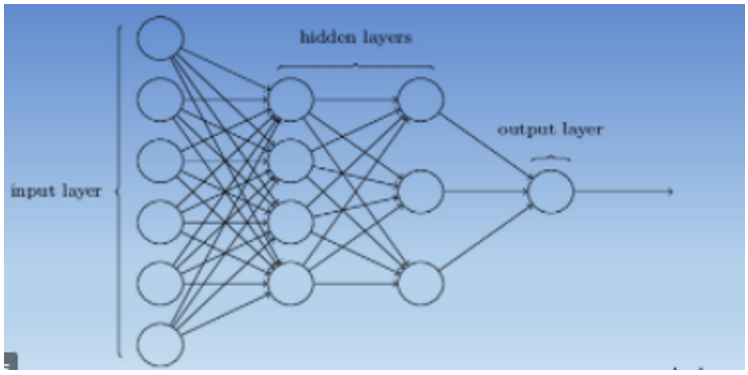
Deep Learning :

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain , allowing it to “learn” from large amounts of data.

Unlike Machine Learning, it automatically extract features from text/image data and train and predict the result.

NN = a(WI + b)

The NN should get large amount of quality data to perform well.



Hidden Layers are used for Feature Extraction. O/P layers contain Prediction Algorithm. You can add multiple features in hidden layer.

Loss = $(\boldsymbol{y_{actual}} - \boldsymbol{Y_{predicted}})2$

Cost Function = $\mathbf{1/n} * \sum(y_{actual} - \mathbf{Y_{predicted}})^2$

We cannot tune bias much because ultimately weights will not work much and bias is an additional component.

$W_{new} = W_{old} - \eta \frac{dL}{dW_{old}}$

This is updating weight equation where dL is loss.

$$\frac{dL}{dW11} = \frac{dL}{03} \cdot \frac{d03}{dW31} \cdot \frac{dw31}{d02} \cdot \frac{d02}{dw21} \quad \frac{dw21}{d01} \cdot \frac{d01}{dw11} \cdot \frac{dw11}{dI}$$
$$+ \quad path\ 2 + path\ 3 + path\ 4$$

This is called Back Propagation, using Chain rule.
We trying to reduce loss using updating weight equation. Eventually it will reduces weights using equation.

Vanishing Gradient Descent :
It occurs when we have less variants in Data or less spread of Data, and the network is dense then the losses will distributed among the multiple neurons and Change is Weights will be very less and we not get good accuracy. It stuck in some local Minima. This occurs due to similar data variants in Data. For Example, we have less amount of image and we extract more amount of Features.

Solutions for VGD :

1. Increase Variation in Data or spread of Data
2. Tuned Activation Function
3. Residual Connections

Exploding Gradient Descent :

If we have good amount of data, but extraction of features is very low such that it skips some hidden layers. So that loss value is very high and Weight values will be high. Then it stuck in some local Minima. This problem is called Exploding Gradient Descent.

Solutions for EGD :

1. Use Min-Max Scaler or Standard-scaler to normalize the data.

Neuron Die :

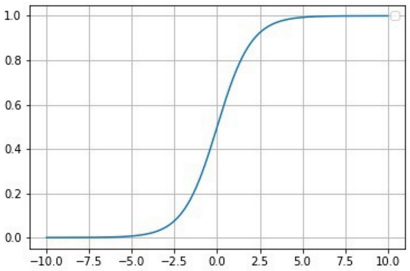
A unit which doesn't update during training by any of its neighbor and basically weights will not get updated and hence we will not reach near global minima.

Activation Function:

is used for activating/ triggering thought process. It can be Linear or Non-Linear.

Sigmoid Function : also called Non Linear or Squashing Function.

f(x) = $\frac{1}{1+e^{-x}}$ where X = W.I + b



Its value always lies between 0 and 1.

Types of Sigmoid :

1. Hard Sigmoid : f(x) = max(0, min(1, $\frac{x+1}{2}$)

   We consider hard sigmoid better than conventional sigmoid because sigmoid takes higher computational cost than hard sigmoid.
2. Sigmoid Weightage Linear Unit : $a^k = z^k . \propto z^k$ where $\propto$ is weight and z^k – W.I + b
3. Derivative of Sigmoid Weightage Linear Unit :

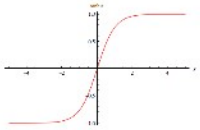
   $a^{k'} = \propto z^k \left(1 + z^k \left(1 - \propto z^k\right)\right)$
4. Tanh : (Hyperbolic Tangent activation function)

   $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ = {-1, 1}
5. Hard Tanh : uses less computational resources.

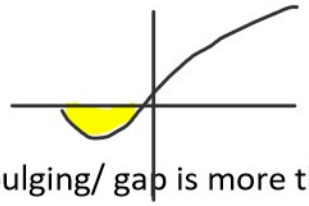
   $f(x) =$ -1 x <-1

   x -1 < x < 1

1 x > 1333333...

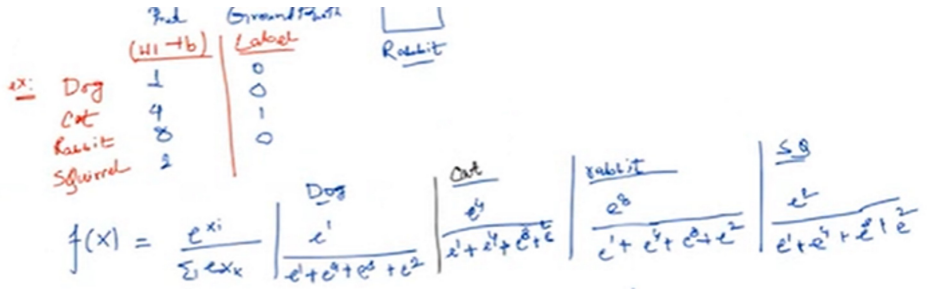


6. ReLu (Rectifier Linear Unit) : $f(x) = \max(0, x)$, it selects only positive values neglects negative values.
   If we normalizing / squashing every feature using above Activation Functions there is huge chance that some important feature is loosing.
7. Leaky ReLu : $f(x) = \begin{matrix} x & if\ x > 0 \\ \propto x & if\ x < 0 \end{matrix}$
   If we have -ve values, then ReLu will skipped that values and the vanishing gradient problems occurs which results in **neuron die**. So we have to use Leaky ReLu or Parametric. Parametric ReLu uses multiple $\propto$ values which will be adjusting while back propagation.
8. Randomized ReLu :
   $$f(x) = \begin{matrix} x & if\ x > 0 \\ \propto x & if\ x < 0 \end{matrix}$$
   here $\propto$ is taken from uniform distribution of u(a,b) $\propto = a + b/2$
9. Softplus AF : Smoother version of ReLu. It is computationally heavy but training time is less.
   $f(x) = \log(1 + e^{-x})$
10. ELU (Exponential Linear Unit) :
    $$f(x) = \begin{matrix} x & if\ x>0 \\ \propto(e^x - 1) & if\ x \leq 0 \end{matrix}$$
    where $\propto$ - hyperparameter for controlling saturation point of -ve value.

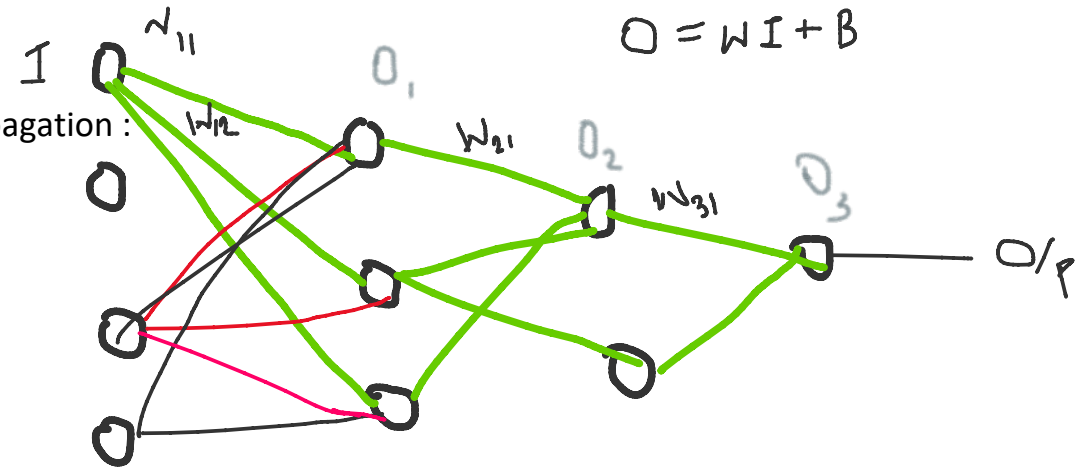


Here the bulging/ gap is more than leaky ReLu. It removes -ve values that are out of bulge/ gap. It consider more range than Leaky ReLu.

11. Swish :
    $f(x) = x.\sigma(x)$
    It increases the lesser value of x.
12. Softmax AF : In other Af, there is some range, softmax don't have any range.
    $$f(x) = \frac{e^x}{\sum e^{Xk}}$$
    It calculates f(x) for every label and selects the highest value. X is W. I. + b



Cross entropy = - $\sum label_i * \log(i)$



Back Propagation :

Types of Gradient Descent :

1. Stochastic GD : It randomly select one path in network, update weights and then takes another so that overall journey is smooth. It takes only one path while iteration. It can useful for binary classification. It fluctuates while converging.
   1 Epoch = n records = n iterations
   Advantages : (a) It will reduce parameters
   (b) Reduces Computational cost
   Disadvantages : (a) Due to random weight updates, overfitting might happens
   For which we have to increase epochs.
2. Mini Batch SGD : Divides Data into batches, every time it takes one batch and updates the weights. It basically fluctuates along the curve to reach Global minima. To solve fluctuation problem, we use momentum.
   1 Epoch = n/k iterations
3. Batch GD : It takes all paths in single iteration. It takes whole data in 1 epoch. Computationally expensive.
   Momentum : It is basically smoothens the fluctuations produced while converging in GD. We calculate Exponential weighted average
   $V_t = \beta V_{t-1} + (1 - \beta) S_t$
   S_t $-$ *previous gradient*
   V_{t-1} $-$ *previous value*
   β - *Coeffient/ hyperparameter for Momentum*

$\underline{Wnew = W - V_t}$

Step : It is difference between current weight and previous weight while iteration.

Adagrad(Adaptive Gradient) :

$$W_{new} = W_{old} - \eta' \frac{dL}{dW_{old}}$$

It uses different n values for different features. It helpful in Multiclass classification. If there are less features, n will diminish.

$$n'_t = \frac{n_{t-1}}{\sqrt{\propto_t + \in}}$$

Where $\propto_t = \sum_{i=1}^{t} \left(\frac{dL^2}{dW_i}\right)$, sum of squared of previous gradient

$\in = small + ve\ value\ for\ balancing\ n$

Adagrad helps to optimize the step size.

But if $\propto_t$ is high then n' will low and weights will not get updated.

So, the solution for this is RMSProp

RMSProp(Root Mean Square Propagation):

$V_t = \beta V_{t-1} + (1 - \beta) \sum_{i=1}^{t} \left(\frac{dL^2}{dW_i}\right)$

$\propto -hyperparameter\ \ for\ tuning\ V_t$

$V_{t-1}\ - previous\ value$

So here we are giving more weightage to current gradient and less weightage to previous gradient.

$$W_{new} = W_{old} - \frac{n_{t-1}}{\sqrt{V_t + \in}}$$

Adam : Adagrad + RMSProp + Momentum

$$W_{new} = W_{old} - \frac{n_{t-1}}{\sqrt{V_t + \in}} * M$$

M - Momentum for smoothening

Bias correction : we use after calculating Momentum and RMS Prop

M' $= \frac{M}{1-\beta}$ and $V'_t = \frac{V_t}{1-\beta}$

We uses this updated values in

$$W_{new} = W_{old} - \frac{n_{t-1}}{\sqrt{V'_t + \in}} * M'$$

Adadelta : is a more robust extension of Adagrad that adapts learning rates based on a restricting previous gradients. It uses RMS prop in num & den.

For Ex, $\beta = 0.95, so\ every\ time\ it\ restricts\ \sum_{i=1}^{t}\left(\frac{dL}{dW_i}\right)^2$

$or\ sum\ of\ squared\ of\ previous\ grad.$

Update equations

Method	Update equation
SGD	$g_t = \nabla_\theta J(\theta_t)$ $\Delta\theta_t = -\eta \cdot g_t$ $\theta_t = \theta_t + \Delta\theta_t$
Momentum	$\Delta\theta_t = -\gamma\, v_{t-1} - \eta g_t$
NAG	$\Delta\theta_t = -\gamma\, v_{t-1} - \eta \nabla_\theta J(\theta - \gamma v_{t-1})$
Adagrad	$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$
Adadelta	$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$
RMSprop	$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
Adam	$\Delta\theta_t = -\frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$

TensorFlow :
TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It has its own data structures.

Keras :
Keras is the high-level API of TensorFlow 2: an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning.

Theano : is a Python library that allows us to evaluate mathematical operations including multi-dimensional arrays so efficiently. It is mostly used in building Deep Learning Projects. It works a way more faster on Graphics Processing Unit (GPU) rather than on CPU