DEEP LEARNING

Module No: 02

Training, Optimization and Regularization of Deep Neural Network

Forward propagation

Forward propagation

- Forward propagation in deep learning refers to the process of passing input data through the neural network to get the output or prediction.
- It involves a series of computations, where the input data is transformed as it passes through the layers of the network.
- Process of passing the input forward through the network, involving weighted sums, biases, and activation functions, is forward propagation.
- The network learns the optimal weights and biases during the training phase to make accurate predictions.

Deep Leaning

Forward propagation

Simple Analogy

- Preparing a recipe (making predictions)
- Ingredients (Input)
- Ingredients importance or preference (Weights and Biases)
- Mixing Ingredients (Weighted Sums)
- Taste recipe (Activation Function)
- Final Dish as per desire (Output : 0/1)

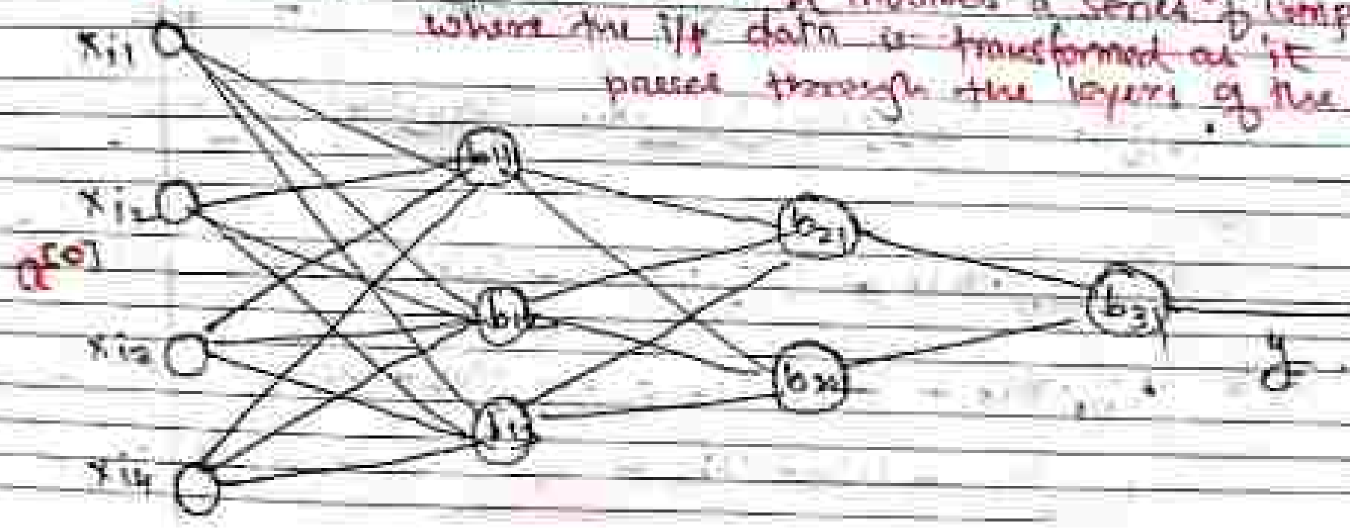
Deep Leaning

forward propogation.

organized way of doing mathematics.

cgpa	iq	10th marks	12 marks	placed?
7.2	75	69	81	1
8.1	92	50	40	0.

Forward propagation in DL refers to the process of passing i/p data through the nn to get the o/p or prediction. * It involves a series of computations where the i/p data is transformed as it passes through the layers of the network.



Weight Matrix for layer 1.

$$\begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \\ w_{41}^1 & w_{42}^1 & w_{43}^1 \end{bmatrix}$$

$$\text{prediction} = \sigma(W^T \cdot x + b)$$

for layer 1

$$= \sigma \underset{3\times4}{\begin{bmatrix} w_{11}^1 & w_{21}^1 & w_{31}^1 & w_{41}^1 \\ w_{12}^1 & w_{22}^1 & w_{32}^1 & w_{42}^1 \\ w_{13}^1 & w_{23}^1 & w_{33}^1 & w_{43}^1 \end{bmatrix}} \cdot \underset{4\times1}{\begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ x_{i4} \end{bmatrix}} + \underset{3\times1}{\begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix}}$$

Deep Leaning

$$= \sigma \left(\begin{bmatrix} W^1_{11} & W^1_{21} & W^1_{31} & W^1_{41} \\ W^1_{12} & W^1_{22} & W^1_{32} & W^1_{42} \\ W^1_{13} & W^1_{23} & W^1_{33} & W^1_{43} \end{bmatrix}_{3\times 4} \cdot \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ x_{i4} \end{bmatrix}_{4\times 1} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix}_{3\times 1} \right)$$

$$= \sigma \left(\begin{bmatrix} W^1_{11} x_{i1} & W^1_{21} x_{i2} & W^1_{31} x_{i3} & W^1_{41} x_{i4} \\ W^1_{12} \cdot x_{i1} & W^1_{22} \cdot x_{i2} & W^1_{32} x_{i3} & W^1_{42} \cdot x_{i3} \\ W^1_{13} \cdot x_{i1} & W^1_{23} \cdot x_{i2} & W^1_{33} \cdot x_{i3} & W^1_{43} x_{i4} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix} \right)$$

$$= \sigma \begin{bmatrix} W^1_{11} \cdot x_{i1} + W^1_{21} \cdot x_{i2} + W^1_{31} \cdot x_{i3} + W^1_{41} \cdot x_{i4} + b_{11} \\ W^1_{12} \cdot x_{i1} + W^1_{22} \cdot x_{i2} + W^1_{32} \cdot x_{i3} + W^1_{42} \cdot x_{i4} + b_{12} \\ W^1_{13} \cdot x_{i1} + W^1_{23} \cdot x_{i2} + W^1_{33} \cdot x_{i3} + W^1_{43} \cdot x_{i4} + b_{13} \end{bmatrix}$$

Deep Leaning

$$= \begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix} \quad \cdots\cdots \quad a[1]$$

Now weight matrix for layer #2

$$\begin{bmatrix} W^2_{11} & W^2_{12} \\ W^2_{21} & W^2_{22} \\ W^2_{31} & W^2_{32} \end{bmatrix}$$

prediction for layer 2 = $\sigma\, W^T \cdot O + b$.

Date

$$= \sigma \begin{bmatrix} W^2_{11} & W^2_{21} & W^2_{31} \\ W^2_{12} & W^2_{22} & W^2_{32} \end{bmatrix} \cdot \begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix} + \begin{bmatrix} b_{21} \\ b_{22} \end{bmatrix}$$

Deep Leaning

$$= \sigma \begin{bmatrix} W_{11}^2 \cdot O_{11} & W_{21}^2 \cdot O_{12} & W_{31}^2 \cdot O_{13} \\ W_{12}^2 \cdot O_{11} & W_{22}^2 \cdot O_{12} & W_{32}^2 \cdot O_{13} \end{bmatrix} + \begin{bmatrix} b_{21} \\ b_{22} \end{bmatrix}$$

$$= \sigma \begin{bmatrix} W_{11}^2 \cdot O_{11} + W_{21}^2 \cdot O_{12} + W_{31}^2 \cdot O_{13} + b_{21} \\ W_{12}^2 \cdot O_{11} + W_{22}^2 \cdot O_{12} + W_{32}^2 \cdot O_{13} + b_{22} \end{bmatrix}$$

$$= \begin{bmatrix} O_{21} \\ O_{22} \end{bmatrix} \quad \ldots\ldots \quad a[2]$$

Now Weight Matrix for layer 3

$$\begin{bmatrix} W_{11}^3 \\ W_{21}^3 \end{bmatrix}$$

prediction for layer 3 $= \sigma\, W^T \cdot O + b$.

Deep Leaning

$$= \sigma \begin{bmatrix} w_{11}^{3} & w_{21}^{3} \end{bmatrix} \begin{bmatrix} O_{21} \\ O_{22} \end{bmatrix} + \begin{bmatrix} b_{13} \end{bmatrix}$$

$$= \sigma \begin{bmatrix} w_{11}^{3} \cdot O_{21} \cdot & w_{21}^{3} \cdot O_{22} \end{bmatrix} + \begin{bmatrix} b_{13} \end{bmatrix}$$

$$= \sigma \left[w_{11}^{3} \cdot O_{21} + w_{21}^{3} \cdot O_{22} + b_{13} \right]$$

$$= \begin{bmatrix} O_{31} \end{bmatrix} \quad \ldots\ldots \quad a[3]$$

$$= y$$

We will simplify above

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

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

$$a^{[3]} = \sigma \left[a^{[2]} \cdot w^{[3]} + b^{[3]} \right) = y$$

Loss Function

Loss function

- In deep learning, a loss function is a measure of how well a model's predictions match the actual target values.
- The goal during the training of a model is to minimize this loss function.
- It quantifies the difference between predicted values and true values, providing a way to assess how well the model is performing.
- Different types of problems (classification, regression, etc.) and algorithms use different loss functions.

What is Loss Function?

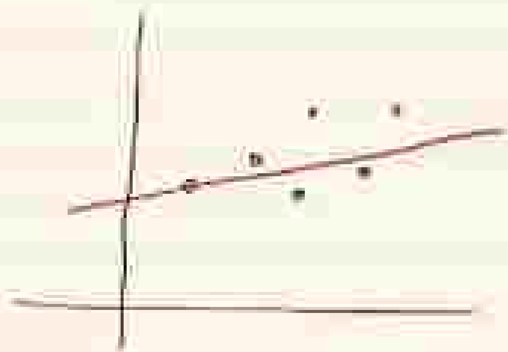
Loss function is a method of evaluating how well your algorithm is modelling your dataset.

high → poor

small → great

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

$L(\text{parameters})$



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

$m x_i + b$

$L(m, b) = \min$

GD

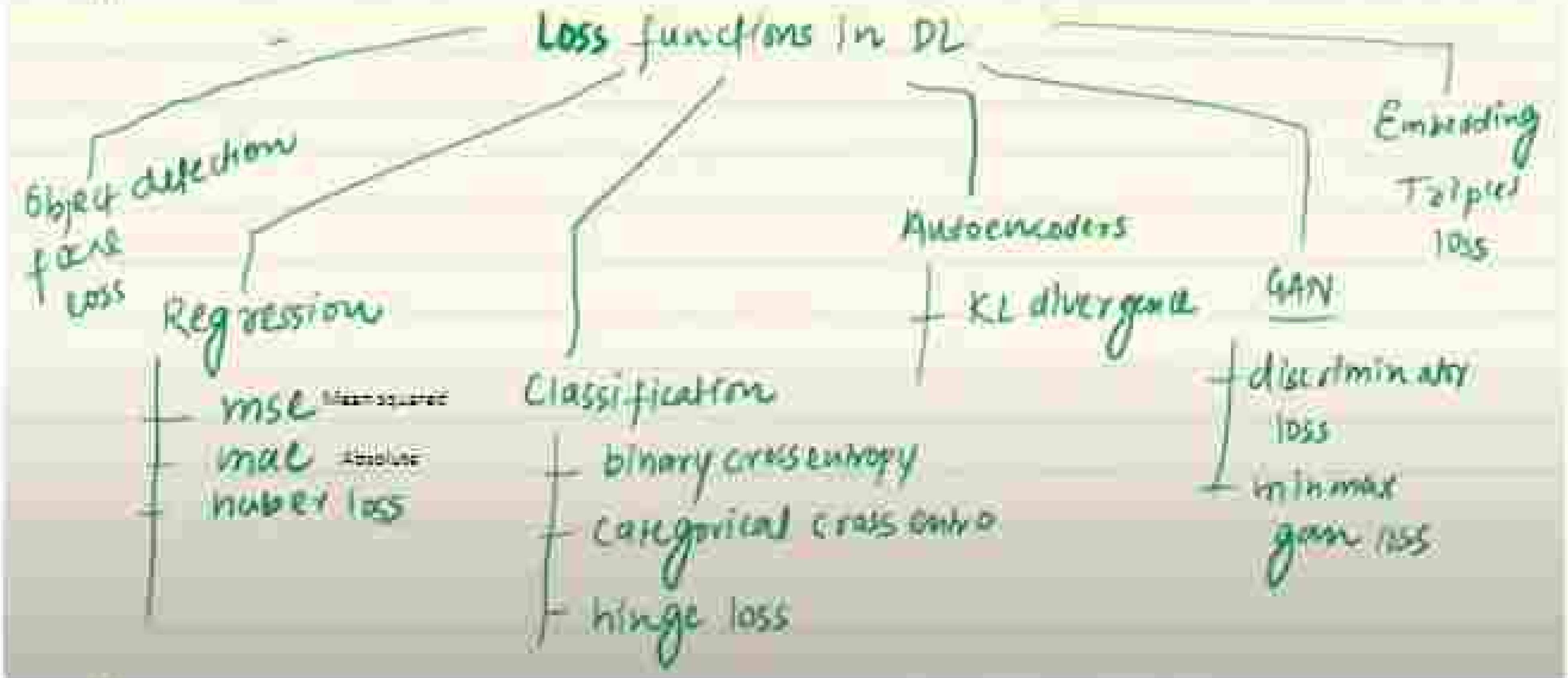
Why is Loss function important?

[You can't improve what you can't measure.]

Peter Drucker

Deep Learning

Loss Function



Loss Function

Squared Error Loss (Mean Squared Error - MSE):

- **Use Case:** Typically used for regression problems, where the goal is to predict a continuous variable.
- **Calculation:** It calculates the average of the squared differences between the predicted and actual values.
- **Formula:**

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

n is the number of data points, y_i is the actual value, and $\hat{y}_i$ is the predicted value.

1. Mean squared Error (MSE)

squared loss   L2 Loss

$(y_i - \hat{y}_i)^2$

(true − predict)^2

$(6.3 - 6.1)^2 = -$

$\boxed{(y_i - \hat{y}_i)^2}$

Adam   DBAAS

mae

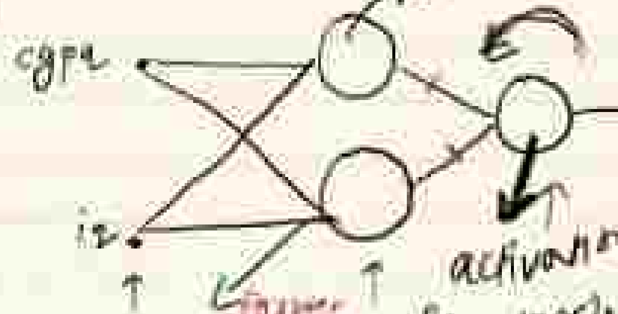
$(y_i - \hat{y}_i)^2$

quadratic

^2

punish

cgpa	iq	package (y_i)	Prediction ($\hat{y}_i$)	error $(y_i - \hat{y}_i)^2$
6.3	100	6.3	6.1	0.2
7.1	91	4.1	4	0.1
8.5	83	3.5	3.7	−0.2
9.2	102	7.2	7	0.2
8.1	100	50	8	overall error



$(42)^2$

drastic

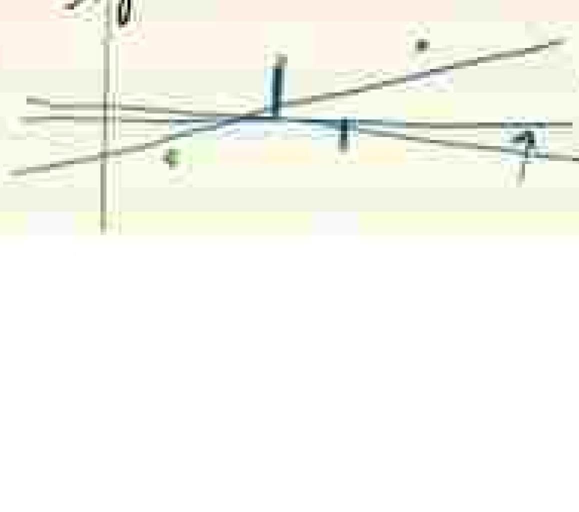
big

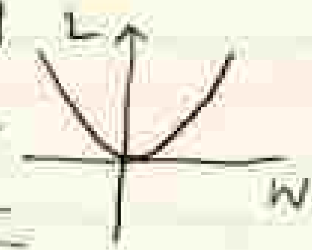
true − predic → 1 unit → 1 unit
2 unit → 4 unit
4 unit → 16 unit

magnify

$(y_i - \hat{y}_i)^2$

$(w, b) \rightarrow L$ min





Squared Error Loss (Mean Squared Error - MSE)

Problem:

Suppose you have a dataset with actual values and predicted values as follows:

Actual Values (y): [5, 10, 15, 20]

Predicted Values ($\hat{y}$): [7, 9, 14, 18]

Deep Learning

Squared Error Loss (Mean Squared Error - MSE)

Solution:

Calculate Squared Differences:

$(y_i - \hat{y}_i)^2$

$(5 - 7)^2 = 4$

$(10 - 9)^2 = 1$

$(15 - 14)^2 = 1$

$(20 - 18)^2 = 4$

Calculate Mean Squared Error (MSE):

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

$MSE = \frac{4+1+1+4}{4} = \frac{10}{4} = 2.5$

Squared Error Loss (Mean Squared Error - MSE)

Suppose you have a dataset with actual values and predicted values as follows:

- True values: $[5, 10, 15]$
- Predicted values: $[7, 8, 13]$

Now, we calculate the squared differences for each pair:

$$(5-7)^2 = 4$$
$$(10-8)^2 = 4$$
$$(15-13)^2 = 4$$

Then, we take the average:

$$\text{MSE} = \frac{1}{3} \times (4+4+4) = \frac{12}{3} = 4$$

Cross Entropy Loss (Binary Cross Entropy and Categorical Cross Entropy)

Binary Cross Entropy Loss:

- **Use Case:** Commonly used for classification problems.
- **Calculation:** For binary classification problems (two classes).
- **Formula:**

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)]$$

n is the number of data points, y_i is the true label (0 or 1), $\hat{y}_i$ is the predicted probability.

4. Binary Cross Entropy

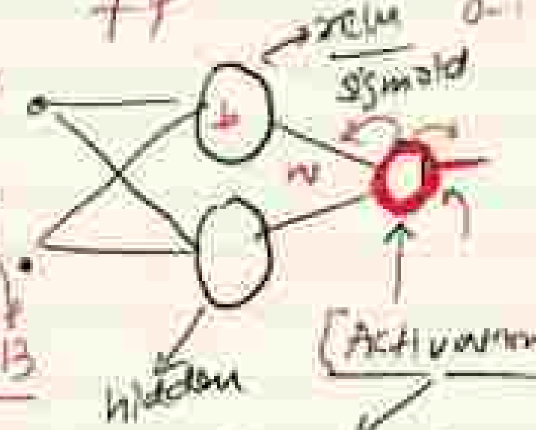
→ classification ✓

→ Two classes ✓

$\rightarrow$ 1 0 GD FP

cgpa	iq	placement	
8	80	1	0.73
7	70	0	0.25
6	60	0	0.13

0.12



{sigmoid}

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

$y \rightarrow$ actual value / target

$\hat{y}$ − NN prediction

0.12 $\quad -(1-0)\log(1-0.25)$

$-1\log 0.75 \quad -0.12$

$-1\log(0.73)$

$-1\times -0.13 = 0.13$

cost function $= -\frac{1}{n}\left[\sum_{i=1}^{n} y_i\log\hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right]$

Cross Entropy Loss
(Binary Cross Entropy and Categorical Cross Entropy)

Binary Cross Entropy Loss:

consider a binary classification problem where the true class label (ground truth) is 1, and the model prediction is 0.8.

Here, Binary Cross Entropy (BCE) Loss is calculated as:

$$\text{BCE Loss} = -[y \log(p) + (1-y)\log(1-p)]$$

Where:

- y is the true label (1 for the positive class, 0 for the negative class).
- p is the predicted probability of the positive class.

Cross Entropy Loss
(Binary Cross Entropy and Categorical Cross Entropy)

Binary Cross Entropy Loss:

Let's substitute the values into the formula:

$$\text{BCE Loss} = -[1 \cdot \log(0.8) + (1-1) \cdot \log(1-0.8)]$$

$$\text{BCE Loss} = -[\log(0.8)]$$

Now, calculate the numerical value:

$$\text{BCE Loss} = -[-0.09691]$$

$$\text{BCE Loss} \approx 0.09691$$

Cross Entropy Loss
(Binary Cross Entropy and Categorical Cross Entropy)

Exercise:

Suppose the true label (ground truth) is 0 (negative class), and the model predicts a probability of 0.3 for the positive class.

Let's substitute the values into the formula:

$$\text{BCE Loss} = -[0 \cdot \log(0.3) + (1-0) \cdot \log(1-0.3)]$$

$$\text{BCE Loss} = -[\log(0.7)]$$

Now, calculate the numerical value:

$$\text{BCE Loss} = -[-0.1549]$$

$$\text{BCE Loss} \approx 0.1549$$

Cross Entropy Loss
(Binary Cross Entropy and Categorical Cross Entropy)

Categorical Cross Entropy Loss:

- **Use Case:** Commonly used for classification problems.
- **Calculation:** For multi-class classification problems (more than two classes).
- **Formula**

$$CCE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{i,j} \cdot \log(\hat{y}_{i,j})$$

n is the number of data points, m is the number of classes, $y_{i,j}$ is the true label, $\hat{y}_{i,j}$ is the predicted probability for class j.

5. Categorical Cross Entropy [used in Softmax Regression]

→ [Multi-class] Classification

1 point

$$\boxed{L = -\sum_{j=1}^{k} y_j \log(\hat{y}_j)}$$

cgpa	iq	placed?		Yes	No	Maybe
8	80	Yes	1	1	0	0
6	60	No	2	0	1	0
7	70	Maybe	3	0	0	1

where k is # classes in the data

k → 3

1 point

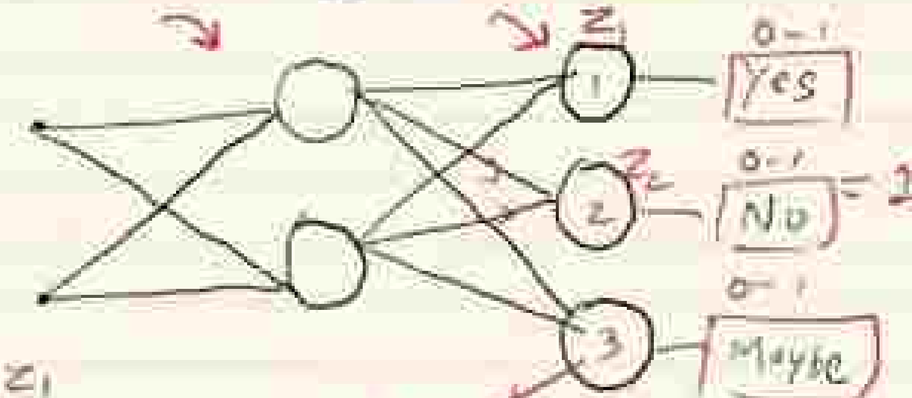
$$L = -y_1 \log(\hat{y}_1) - y_2 \log(\hat{y}_2) - y_3 \log(\hat{y}_3)$$

$$\frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_1}}$$

$$f(z) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$\frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Softmax



Activation

neurons

class ← categories

→ Multi-class classification

1 point

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

8	80
6	60
7	70

Yes 1
No 2
Maybe 3

$$\begin{array}{c|l} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{array}{l} \text{Yes} \\ \text{No} \\ \text{Maybe} \end{array} \end{array}$$

where (k) is # classes in the data → 3

1 point

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

[0.2 0.3 0.5]

[1 0 0]

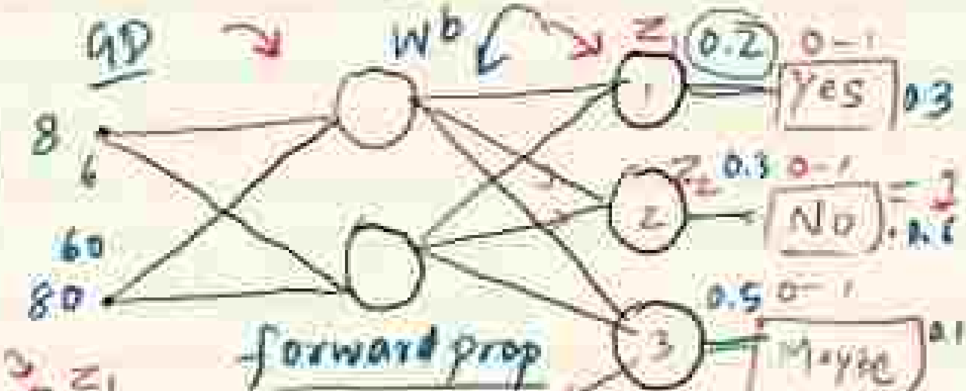
$$\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}$$

$$\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}$$

$L = -1 \times \log(0.2)$

$$f(z) = \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}$$

softmax



Activation

neurons

class ← categories

[0.3 0.6 0.1]

[

Cross Entropy Loss
(Binary Cross Entropy and Categorical Cross Entropy)

Categorical Cross Entropy Loss:

consider a scenario where we have a classification problem with three classes (C1, C2, and C3). We have a set of true class probabilities and predicted class probabilities for each instance. The Categorical Cross-Entropy Loss is commonly used in such multi-class classification problems.

Suppose we have three instances, and the true class probabilities and predicted class probabilities are as follows:

Cross Entropy Loss
(Binary Cross Entropy and Categorical Cross Entropy)

Instance 1:

- True class probabilities: $[1, 0, 0]$ (belongs to C1)
- Predicted class probabilities: $[0.8, 0.15, 0.05]$

Instance 2:

- True class probabilities: $[0, 1, 0]$ (belongs to C2)
- Predicted class probabilities: $[0.2, 0.7, 0.1]$

Instance 3:

- True class probabilities: $[0, 0, 1]$ (belongs to C3)
- Predicted class probabilities: $[0.05, 0.1, 0.85]$

Deep Learning

Cross Entropy Loss (Binary Cross Entropy and Categorical Cross Entropy)

The Categorical Cross-Entropy Loss for each instance is calculated as:

$$L_i = -\sum_j y_{ij} \cdot \log(\hat{y}_{ij})$$

Where:

- y_{ij} is the true probability of class j for instance i.
- $\hat{y}_{ij}$ is the predicted probability of class j for instance i.

Let's calculate the loss for each instance:

Cross Entropy Loss
(Binary Cross Entropy and Categorical Cross Entropy)

Instance 1:

$$L_1 = -(1 \cdot \log(0.8) + 0 \cdot \log(0.15) + 0 \cdot \log(0.05))$$

Instance 2:

$$L_2 = -(0 \cdot \log(0.2) + 1 \cdot \log(0.7) + 0 \cdot \log(0.1))$$

Instance 3:

$$L_3 = -(0 \cdot \log(0.05) + 0 \cdot \log(0.1) + 1 \cdot \log(0.85))$$

Now, we can calculate the average loss over all instances:

$$\text{Average Loss} = \frac{L_1 + L_2 + L_3}{3}$$

Deep Learning

Cross Entropy Loss
(Binary Cross Entropy and Categorical Cross Entropy)

Instance 1:

$$L_1 = -(1 \cdot \log(0.8) + 0 \cdot \log(0.15) + 0 \cdot \log(0.05))$$

Instance 2:

$$L_2 = -(0 \cdot \log(0.2) + 1 \cdot \log(0.7) + 0 \cdot \log(0.1))$$

Instance 3:

$$L_3 = -(0 \cdot \log(0.05) + 0 \cdot \log(0.1) + 1 \cdot \log(0.85))$$

Now, we can calculate the average loss over all instances:

$$\text{Average Loss} = \frac{L_1 + L_2 + L_3}{3}$$

This average loss gives us an indication of how well the predicted probabilities match the true class probabilities across all instances.
Lower values indicate better performance.

What is activation function?

- The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.
- In artificial neural networks, an activation function is one that outputs a smaller value for tiny inputs and a higher value if its inputs are greater than a threshold. An activation function "fires" if the inputs are big enough; otherwise, nothing happens.
- An activation function, then, is a gate that verifies how an incoming value is higher than a threshold value.
- The activation function is a fundamental component of neural networks that introduces non-linearity, enabling them to learn complex relationships, adapt to various data patterns, and make sophisticated decisions.

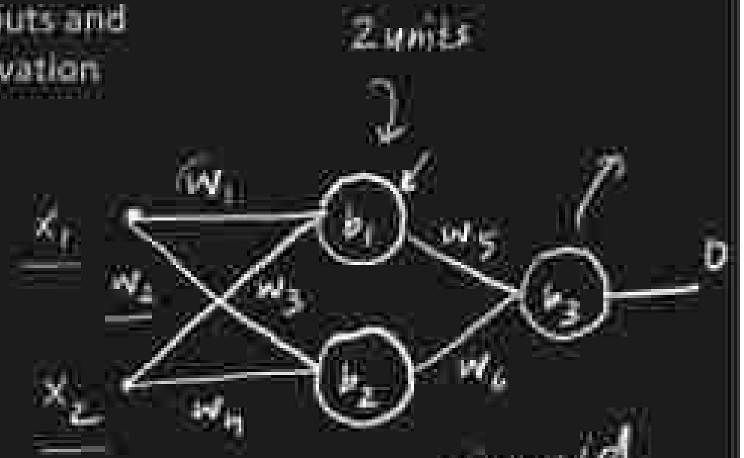
What are Activation Functions?

31 May 2022 14:49

In artificial neural networks, each neuron forms a weighted sum of its inputs and passes the resulting scalar value through a function referred to as an activation function or transfer function. If a neuron has *n* inputs then the output or activation of a neuron is

$$a = g(w_1x_1 + w_2x_2 + w_3x_3 + \ldots w_nx_n + b)$$

This function g is referred to as the activation function.



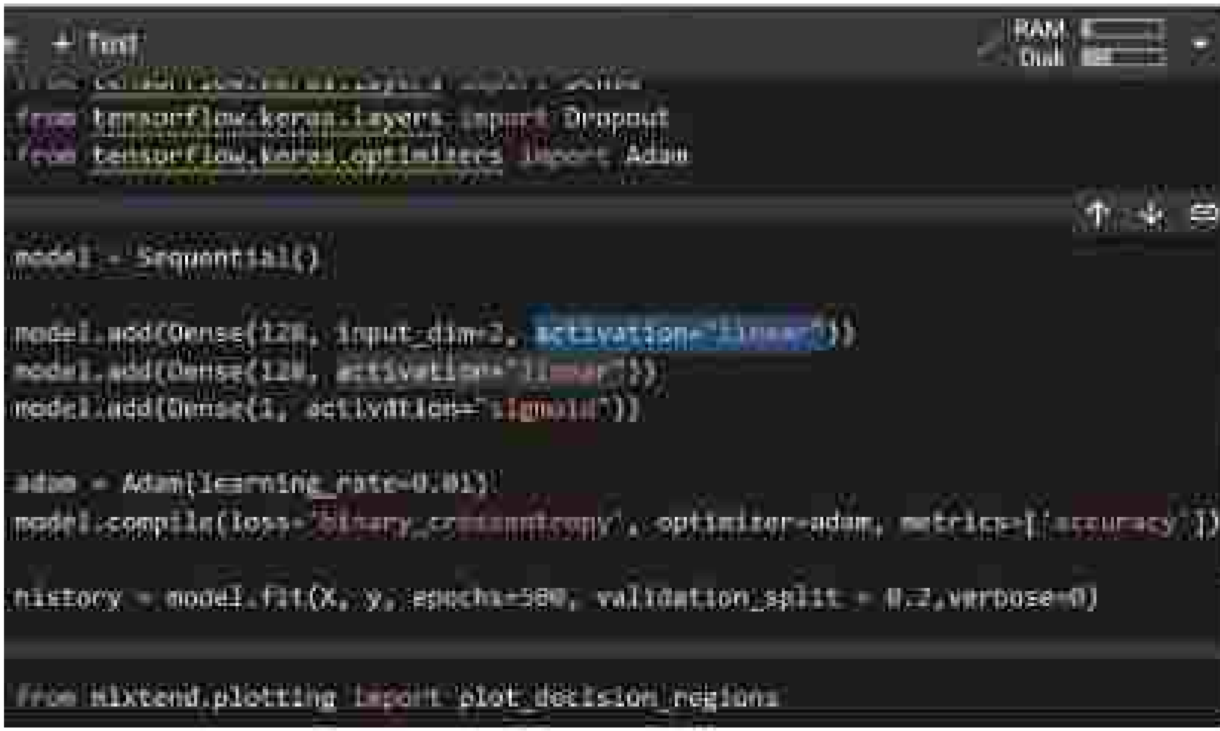
$$g(w_1x_1 + w_2x_2 + b_1)$$

sigmoid

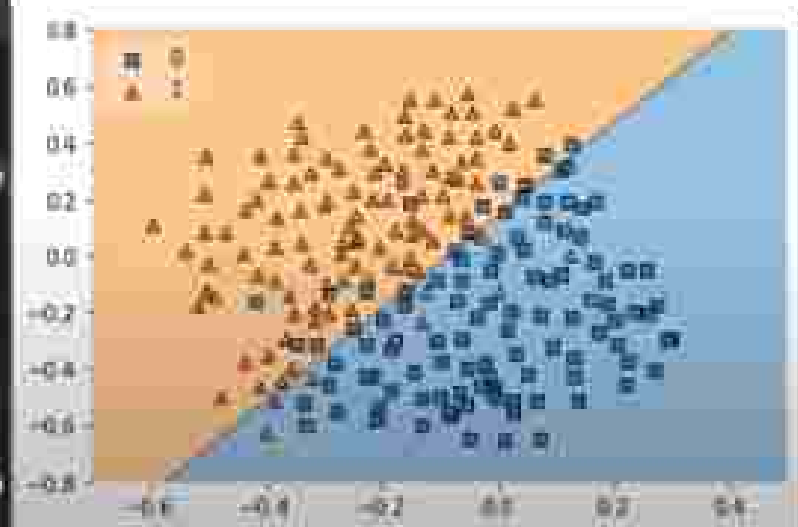
relu

softmax

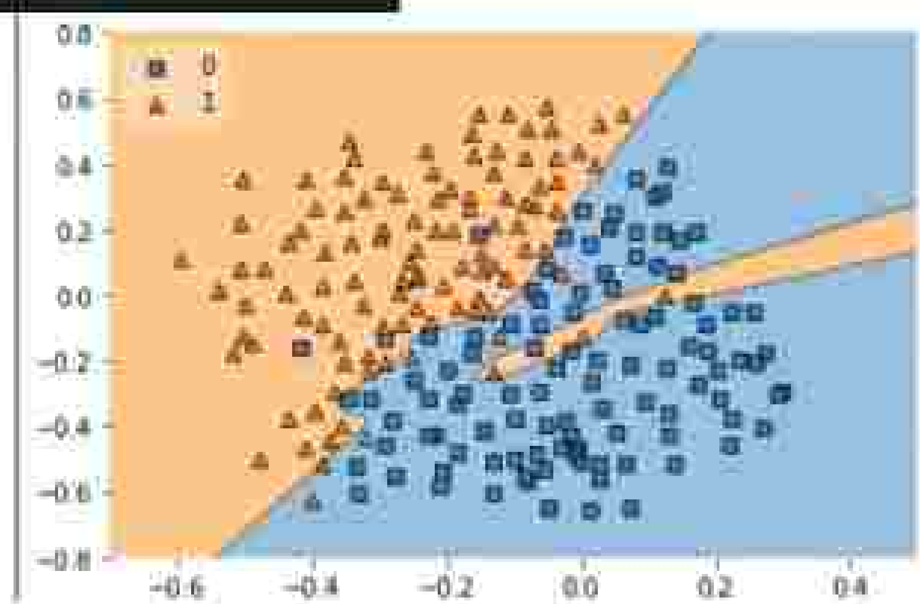
activation



```
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam

model = Sequential()

model.add(Dense(128, input_dim=2, activation='linear'))
model.add(Dense(128, activation='linear'))
model.add(Dense(1, activation='sigmoid'))

adam = Adam(learning_rate=0.01)
model.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])

history = model.fit(X, y, epochs=500, validation_split = 0.2,verbose=0)

from mlxtend.plotting import plot_decision_regions
```



```python
model = Sequential()

model.add(Dense(128, input_dim=2, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dense(1, activation="sigmoid"))

adam = Adam(learning_rate=0.01)
model.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])

history = model.fit(X, y, epochs=500, validation_split = 0.2,verbose=0)

from mlxtend.plotting import plot_decision_regions
```



Why there is a need of activation function?

Introducing Non-linearity:

- Without activation functions, the entire neural network would behave like a linear model.
- The stacking of multiple linear operations would result in a linear combination, limiting the network's ability to learn and represent complex, non-linear patterns in the data.

Capturing Complex Relationships:

- Many real-world problems involve intricate and non-linear relationships.
- Activation functions allow the neural network to model and capture these complex patterns, making it more powerful in representing diverse data.

Enabling Neural Network to Learn:

- The non-linear transformations introduced by activation functions enable the network to learn and adapt to intricate patterns in the input data during the training process. This is crucial for the network to generalize well to unseen data.

Why there is a need of activation function?

Thresholding and Output Scaling:

- Activation functions often introduce thresholding effects, where the neuron activates or not based on certain conditions.
- This helps in decision-making and provides a level of abstraction. Additionally, activation functions like sigmoid and softmax scale the output to represent probabilities in classification tasks.

Avoiding Vanishing or Exploding Gradients:

- Activation functions play a role in mitigating issues like vanishing or exploding gradients during backpropagation, especially in deep neural networks.
- Well-designed activation functions help in the stable training of deep networks.

Why there is a need of activation function?

Introducing Sparsity:

- Some activation functions, like ReLU (Rectified Linear Unit) and its variants, introduce sparsity in the network by setting negative values to zero.
- This can be beneficial in certain scenarios.

Facilitating Backpropagation:

- Activation functions provide derivatives or gradients that are essential for the backpropagation algorithm, which is used to update the weights of the network during training.
- This enables the network to learn and improve its performance over time.

Types of activation function

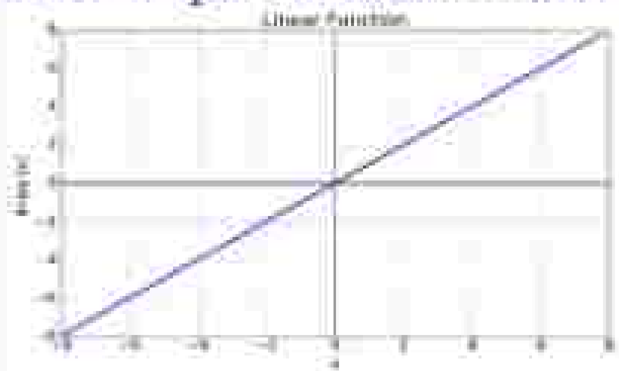
In a perceptron or a neural network, activation functions play a crucial role by introducing non-linearity to the model.
Here are some common types of activation functions used in perceptrons:

1. Linear activation function
2. Logistic activation function
3. Tanh activation function
4. Softmax activation function
5. ReLU activation function
6. Leaky ReLU activation function

Types of activation function

1. **Linear Function:**
   - Description: The linear activation function, also known as the identity activation function, is a straightforward and simple function. It is defined as:
   - Mathematical Form:

$$f(x) = x$$



   - Advantages:
     - Simplicity
     - Ease of Interpretation:
       - Direct proportionality between input and output.
       - Straightforward interpretation.
     - Compatibility with Linear Models (Well-suited for tasks with linear relationships)

Types of activation function

Disadvantages:

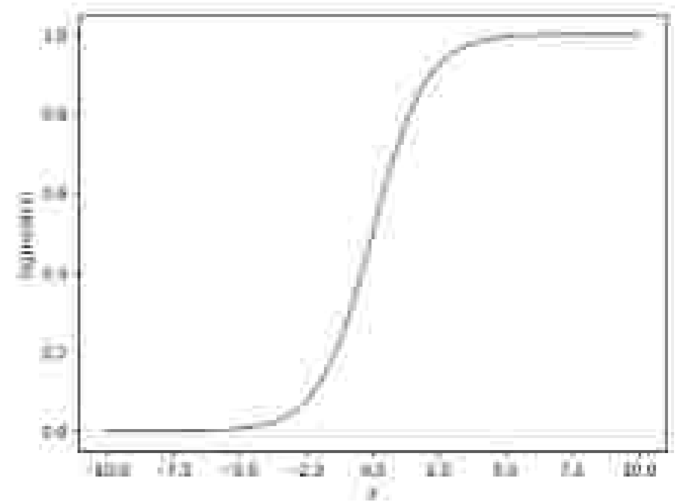
- Limited Expressiveness:
  - Inability to model complex, non-linear relationships.
  - Stacking linear layers results in a linear model.
- Vanishing Gradient Problem:
  - Prone to vanishing gradients, especially in deep networks.
  - May lead to slow learning.
- Not Suitable for Classification Problems:
  - Challenging for binary classification tasks.
  - Output not squashed into a specific range.
- Not Used in Hidden Layers of Deep Networks:
  - Rarely used in deep networks' hidden layers.
  - Non-linear activations preferred.

Types of activation function

2. Logistic Activation Function :

- is also commonly referred to as the Sigmoid Activation Function.
- Description: The sigmoid (logistic) function squashes input values to the range (0, 1). It is commonly used in the output layer of binary classification models.
- Mathematical Form:

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



Types of activation function

- It is a function which is plotted as 'S' shaped graph.
- Nature : Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- Value Range : 0 to 1
- Uses : Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.

Types of activation function

Question:

Calculate the output of a neuron with the given parameters using the sigmoid activation function:

- $x_1 = 0.7$
- $x_2 = -0.3$
- $w_1 = 0.5$
- $w_2 = -1$
- $b = 0.2$

Types of activation function

Given $x_1 = 0.7$, $x_2 = -0.3$, $w_1 = 0.5$, $w_2 = -1$, and $b = 0.2$, the weighted sum z is calculated as:

$$z = (0.5 \cdot 0.7) + ((-1) \cdot (-0.3)) + 0.2$$

Simplifying this:

$$z = 0.35 + 0.3 + 0.2 = 0.85$$

Now, applying the sigmoid activation function:

$$\sigma(0.85) = \frac{1}{1+e^{-0.85}}$$

Calculating this:

$$\sigma(0.85) \approx \frac{1}{1+e^{-0.85}} \approx \frac{1}{1+0.427} \approx \frac{1}{1.427} \approx 0.7$$

So, the output of the neuron for the given inputs, weights, and bias is approximately 0.7.

Types of activation function

Problem:

Calculate the output of a neuron with the given data using the sigmoid activation function:

- $x_1 = -0.6$
- $x_2 = 0.4$
- $w_1 = 1.2$
- $w_2 = -0.8$
- $b = 0.5$

Deep Learning

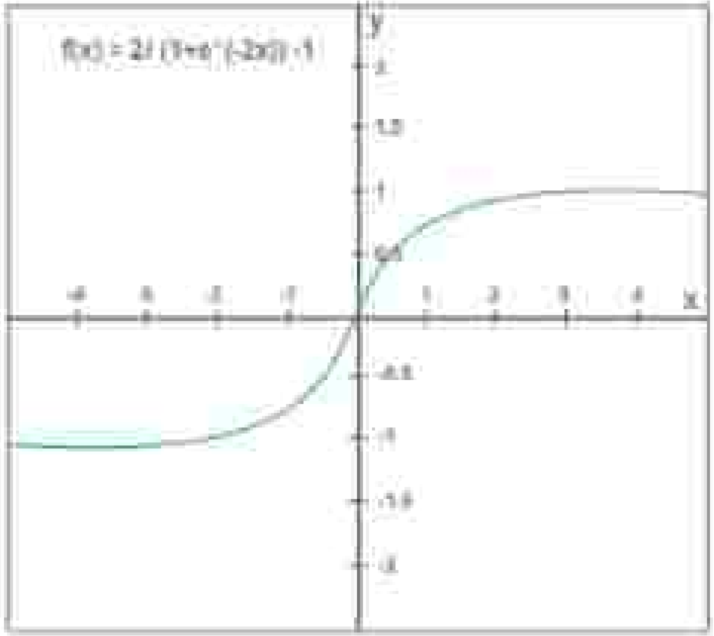
Types of activation function

3. Tanh (Hyperbolic Tangent) Function:

- Description: Similar to the sigmoid function, the tanh function maps input values to the range (-1, 1). It is often used in hidden layers of neural networks.
- Mathematical Form:

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

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$



Types of activation function

- The activation that works almost always better than sigmoid function is Tanh function also known as Tangent Hyperbolic function. It is actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
- Value Range : -1 to +1
- Nature :- non-linear
- Uses :- Usually used in hidden layers of a neural network as it is values lies between -1 to 1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier.

Deep Learning

Types of activation function

Problem:

Calculate the output of a neuron with the given data using the hyperbolic tangent (tanh) activation function:

- $x_1 = 0.8$
- $x_2 = -0.2$
- $w_1 = 0.6$
- $w_2 = 1.1$
- $b = -0.4$

Types of activation function

Solution:

1. Calculate the weighted sum (z):

$$z = (0.6 \cdot 0.8) + (1.1 \cdot -0.2) - 0.4$$
$$z = 0.48 - 0.22 - 0.4$$
$$z = -0.14 - 0.4$$
$$z = -0.54$$

2. Apply the hyperbolic tangent (tanh) activation function:

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

Substitute $z = -0.54$ into the tanh function:

$$\tanh(-0.54) = \frac{e^{-0.54} - e^{0.54}}{e^{-0.54} + e^{0.54}}$$

Calculating the numerator and denominator separately:

$$\tanh(-0.54) = \frac{0.5806 - 1.7147}{0.5806 + 1.7147}$$
$$\tanh(-0.54) = \frac{-1.1341}{2.2953}$$

The final result is:

$$\tanh(-0.54) \approx -0.4929$$

Deep Learning

Types of activation function

Problem:

Calculate the output of a neuron using the hyperbolic tangent (tanh) activation function for the following data:

$$x_1 = -0.6, \quad x_2 = 0.2, \quad w_1 = 0.8, \quad w_2 = -1.3, \quad b = 0.5$$

Types of activation function

1. Calculate the weighted sum (z):

   $z = (-0.6 \cdot 0.8) + (0.2 \cdot -1.3) + 0.5$

   $z = -0.48 - 0.26 + 0.5$

   $z = -0.74 + 0.5$

   $z = -0.24$

2. Apply the hyperbolic tangent (tanh) activation function:

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

   Substitute $z = -0.24$ into the tanh function:

   $\tanh(-0.24) = \frac{e^{-0.24} - e^{0.24}}{e^{-0.24} + e^{0.24}}$

   Calculating the numerator and denominator separately:

   $\tanh(-0.24) = \frac{0.7878 - 1.2689}{0.7878 + 1.2689}$

   $\tanh(-0.24) = \frac{-0.4811}{2.0567}$

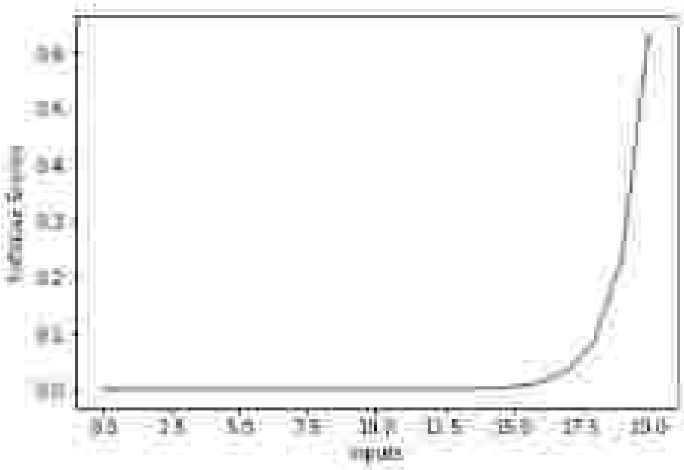
   The final result is:

   $\tanh(-0.24) \approx -0.2345$

Types of activation function

4. Softmax Function:

- Description: Often used in the output layer of a neural network for multi-class classification problems. It transforms the raw output scores (logits) into a probability distribution over multiple classes. The Softmax function is particularly useful when dealing with problems where an input can belong to one of several exclusive classes.
- Mathematical Form:



For a given input vector $\mathbf{z}$ with K elements (where K is the number of classes), the Softmax function is defined as follows:

$$\text{Softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, 2, \ldots, K$$

Here:

- $\text{Softmax}(\mathbf{z})_i$ is the i-th element of the Softmax output.
- e^{z_i} is the exponential of the i-th element of the input vector.
- The denominator is the sum of the exponentials of all elements in the input vector.

Types of activation function

- The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems.
- Nature :- non-linear
- Uses :- Usually used when trying to handle multiple classes. the softmax function was commonly found in the output layer of image classification problems.The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
- If your output is for binary classification then, sigmoid function is very natural choice for output layer.
- If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.

Types of activation function

Problem:

Calculate the output probabilities of a neural network using the Softmax activation function for the following data:

$$x_1 = 1.2, \quad x_2 = -0.5, \quad w_1 = 1.5, \quad w_2 = -2.3, \quad w_3 = 1.3, \quad b = 0.5$$

Types of activation function

1. Calculate the weighted sum:

$$z_1 = (1.2 \times 1.5) + (-0.5 \times -2.3) + 0.5$$
$$z_2 = (1.2 \times 2.3) + (-0.5 \times 3.1) + 0.5$$
$$z_3 = (1.2 \times -1.3) + (-0.5 \times 2.1) + 0.5$$

Simplifying these equations:

$$z_1 = 1.8 + 1.15 + 0.5 \approx 3.45$$
$$z_2 = 2.76 + 1.55 + 0.5 \approx 4.81$$
$$z_3 = -1.56 - 1.05 + 0.5 \approx -2.11$$

Types of activation function

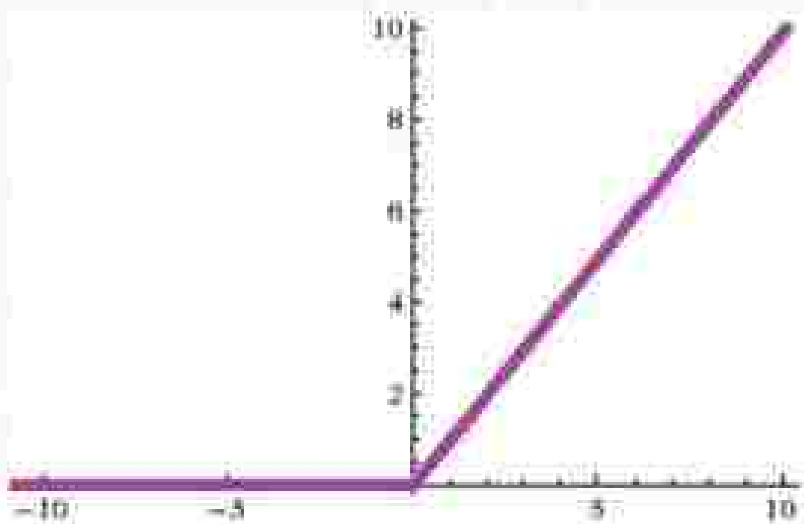
2. Apply the Softmax function:
   - Calculate the unnormalized exponentials:
     $e^{3.45} \approx 31.91, \quad e^{4.81} \approx 121.46, \quad e^{-2.11} \approx 0.123$
   - Calculate the sum of the exponentials:
     $\text{Sum} = 31.91 + 121.46 + 0.123 \approx 153.493$
   - Calculate the probabilities:
     $P(\text{class}_1) = \frac{31.91}{153.493} \approx 0.208$
     $P(\text{class}_2) = \frac{121.46}{153.493} \approx 0.791$
     $P(\text{class}_3) = \frac{0.123}{153.493} \approx 0.001$
3. Final Output:
   - The calculated probabilities for each class using the Softmax function are approximately:
     $P(\text{class}_1) \approx 0.208, \quad P(\text{class}_2) \approx 0.791, \quad P(\text{class}_3) \approx 0.001$

Deep Learning

Types of activation function

5. Rectified Linear Unit (ReLU):

- Description: ReLU is a popular activation function that outputs the input for positive values and zero for negative values. It introduces non-linearity and is computationally efficient.
- Mathematical Form: $f(x) = \max(0, x)$



Types of activation function

- It Stands for Rectified linear unit. It is the most widely used activation function. Chiefly implemented in hidden layers of Neural network.
- Value Range :- [0, inf)
- Nature :- non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- Uses :- ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

Types of activation function

Problem:

Calculate the output of a neuron using the Rectified Linear Unit (ReLU) activation function for the following data:

$$x_1 = 0.9, \quad x_2 = -0.5, \quad w_1 = 0.6, \quad w_2 = 0.8, \quad b = -0.2$$

Types of activation function

1. Calculate the weighted sum (z):

$$z = (0.9 \cdot 0.6) + (-0.5 \cdot 0.8) - 0.2$$

$$z = 0.54 - 0.4 - 0.2$$

$$z = -0.06$$

2. Apply the ReLU activation function:

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

Substitute $z = -0.06$ into the ReLU function:

$$\text{ReLU}(-0.06) = \max(0, -0.06)$$

Since -0.06 is less than zero, the ReLU function returns zero:

$$\text{ReLU}(-0.06) = 0$$

3. Final Output:
   - The calculated weighted sum (z) is approximately -0.06.
   - The output after applying the ReLU function is 0.

Types of activation function

Problem:

Calculate the output of a neuron using the Rectified Linear Unit (ReLU) activation function for the following data:

$$x_1 = 0.6, \quad x_2 = 0.3, \quad w_1 = 0.8, \quad w_2 = -0.5, \quad b = 0.2$$

Types of activation function

1. Calculate the weighted sum (z):

$$z = (0.6 \cdot 0.8) + (0.3 \cdot (-0.5)) + 0.2$$

$$z = 0.48 - 0.15 + 0.2$$

$$z = 0.53$$

2. Apply the ReLU activation function:

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

Substitute $z = 0.53$ into the ReLU function:

$$\text{ReLU}(0.53) = \max(0, 0.53)$$

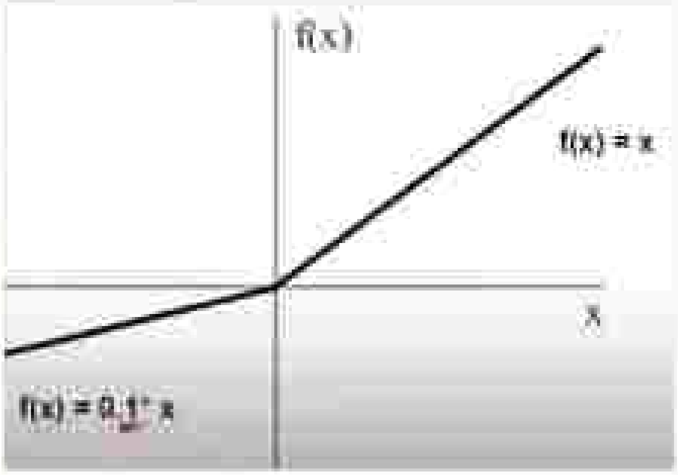
Since 0.53 is greater than zero, the ReLU function returns the input value:

$$\text{ReLU}(0.53) = 0.53$$

Types of activation function

5. Leaky ReLU (Rectified Linear Unit) Function:

- Description: Leaky ReLU is an activation function used in artificial neural networks to introduce nonlinearity among the outputs between layers of a neural network. This activation function was created to solve the dying ReLU problem using the standard ReLU function that makes the neural network die during training.
- Mathematical Form:



$$LeakyReLU(x) = max(alpha * x, x)$$

In the equation, we see that for every parameter x, the Leaky ReLU function will return the maximum value between x or $alpha * x$, where $alpha$ is a small positive constant.

Deep Learning

Types of activation function

- Using this function, we can convert negative values to make them close to 0 but not actually 0, solving the dying ReLU issue that arises from using the standard ReLU function during neural network training.
- The Leaky ReLU is a popular activation function that is used to address the limitations of the standard ReLU function in deep neural networks by introducing a small negative slope for negative function inputs, which helps neural networks to maintain better information flow both during its training and after.

Types of activation function

In a perceptron or a neural network, activation functions play a crucial role by introducing non-linearity to the model.
Here are some common types of activation functions used in perceptrons:

1. Linear activation function
2. Logistic activation function
3. Tanh activation function
4. Softmax activation function
5. ReLU activation function
6. Leaky ReLU activation function