SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107
AN AUTONOMOUS INSTITUTION
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai
DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

UNIT I

Two Marks – Part A

1. What is Deep Learning?

Deep learning is a part of machine learning with an algorithm inspired by the structure and function of the brain, which is called an artificial neural network. In the mid-1960s, Alexey Grigorevich Ivakhnenko published the first general, while working on deep learning network. Deep learning is suited over a range of fields such as computer vision, speech recognition, natural language processing, etc

2. What are the main differences between AI, Machine Learning, and Deep Learning?

AI stands for Artificial Intelligence. It is a technique which enables machines to mimic human behavior.

Machine Learning is a subset of AI which uses statistical methods to enable machines to improve with experiences.

Deep learning is a part of Machine learning, which makes the computation of multi layer neural networks feasible. It takes advantage of neural networks to simulate human-like decision making.

3. Differentiate supervised and unsupervised deep learning procedures.

Supervised learning is a system in which both input and desired output data are provided. Input and output data are labeled to provide a learning basis for future data processing.

Unsupervised procedure does not need labeling information explicitly, and the operations can be carried out without the same. The common unsupervised learning method is cluster analysis. It is used for exploratory data analysis to find hidden patterns or grouping in data.

4. What are the applications of deep learning?

There are various applications of deep learning:

Computer vision

Natural language processing and pattern recognition

Image recognition and processing

Machine translation

Sentiment analysis

Question answering system

Object Classification and Detection

Automatic Handwriting Generation

Automatic Text Generation.

5. What is scalar and vector?

A scalar is just a single number, in contrast to most of the other objects like Vectors, which are usually arrays of multiple numbers.

6.Define SVM.

SVM (Support Vector Machine) is a supervised machine learning algorithm used for classification and regression tasks. It finds the optimal hyperplane that best separates data points into different classes in a high-dimensional space.

7. Why is probability important in deep learning?

Probability is the science of quantifying uncertain things. Most of machine learning and deep learning systems utilize a lot of data to learn about patterns in the data. Whenever data is utilized in a system rather than sole logic, uncertainty grows up and whenever uncertainty grows up, probability becomes relevant.

By introducing probability to a deep learning system, we introduce common sense to the system. Otherwise the system would be very brittle and will not be useful.In deep learning, several models like Bayesian models, probabilistic graphical models, hidden markov models are used. They depend entirely on probability concepts.

8. Define Random Variable.

A random variable is a variable that can take on different values randomly. We typically denote the random variable itself with a lower case letter in plain typeface, and the values it can take on with lower case script letters. For example, x1 and x2 are both possible values that the random variable x can take on.

9. Do random variables is discrete or continuous?

continuous. A discrete random variable is one that has a finite or countably infinite number of states. Note that these states are not necessarily the integers; they can also just be named states that are not considered to have any numerical value. A continuous random variable is associated with a real value.

10. What are probability distributions?

A probability distribution is a description of how likely a random variable or set of random variables is to take on each of its possible states. The way we describe probability distributions depends on whether the variables are discrete or continuous.

11. Define Probability mass function?

A probability distribution over discrete variables may be described using a probability mass function (PMF). We typically denote probability mass functions with a capital P. Often we associate each random variable with a different probability mass function and the reader must infer which probability mass function to use based on the identity of the random variable, rather than the name of the function; P(x) is usually not the same as P(y).

12. List the properties that probability mass function satisfies?

• The domain of P must be the set of all possible states of x.

• $\forall x \in x, 0 \leq P(x) \leq 1$. An impossible event has probability 0 and no state can be less probable than that. Likewise, an event that is guaranteed to happen has probability 1, and no state can have a greater chance of occurring.

• $\sum x \in_x P(x) = 1$. We refer to this property as being normalized. Without this property, we could obtain probabilities greater than one by computing the probability of one of many events occurring.

13. List the properties that probability density function satisfies?
When working with continuous random variables, we describe probability distributions using a probability density function (PDF) rather than a probability mass function.

To be a probability density function, a function p must satisfy the following properties: • The domain of p must be the set of all possible states of x.
• $\forall x \in x$, $p(x) \geq 0$. Note that we do not require $p(x) \leq 1$.
• $\int p(x)dx = 1$.

14. What is Gradient based optimizer?

Gradient descent is an optimization algorithm that's used when training deep learning models. It's based on a convex function and updates its parameters iteratively to minimize a given function to its local minimum.
The notation used in the above Formula is given below,
In the above formula,
🎬 α is the learning rate,
🎬 J is the cost function, and
🎬 ϴ is the parameter to be updated.
As you can see, the gradient represents the partial derivative of J(cost function) with respect to ϴj

15. Why overfitting and underfitting in ML?

Factors determining how well an ML algorithm will perform are its ability to:
1. Make the training error small
2. Make gap between training and test errors small

• They correspond to two ML challenges
Underfitting - Inability to obtain low enough error rate on the training set
Overfitting - Gap between training error and testing error is too large

We can control whether a model is more likely to overfit or underfit by altering its capacity

16. What is capacity of a model?

Model capacity is ability to fit variety of functions
– Model with Low capacity struggles to fit training set
– A High capacity model can overfit by memorizing properties of training set not useful on test set
• When model has higher capacity, it overfits – One way to control capacity of a learning algorithm is by choosing the hypothesis space
• i.e., set of functions that the learning algorithm is allowed to select as being the solution

17. How to control the capacity of learning algorithms?

One way to control the capacity of a learning algorithm is by choosing its hypothesis space, the set of functions that the learning algorithm is allowed to select as being the solution. For example, the linear regression algorithm has the set of all linear functions of its input as its hypothesis space. We can generalize linear regression to include polynomials, rather than just linear functions, in its hypothesis space. Doing so increases the model's capacity

18. Define Bayes error.

Ideal model is an oracle that knows the true probability distributions that generate the data • Even such a model incurs some error due to noise/overlap in the distributions • The error incurred by an oracle making predictions from the true distribution p(x,y) is called the Bayes error

19. Why hyperparameters in ML?

Most ML algorithms have hyperparameters
– We can use to control algorithm behavior
– Values of hyperparameters are not adapted by learning algorithm itself
• Although, we can design nested learning where one learning algorithm
– Which learns best hyperparameters for another learning algorithm.

20. How to solve the overfitting problem caused by learning hyperparameters on training dataset?

• To solve the problem, we use a validation set
– Examples that training algorithm does not observe
• Test examples should not be used to make choices about the model hyperparameters • Training data is split into two disjoint parts
– First to learn the parameters
– Other is the validation set to estimate generalization error during or after training
• allowing for the hyperparameters to be updated
– Typically, 80% of training data for training and 20% for validation

21.Define Perceptrons

A Perceptron is the simplest type of artificial neural network used for binary classification. It consists of input nodes, weighted connections, a summation function, and an activation function (usually a step function) to determine the output.

22.What is the loss function?

A loss function is a mathematical function that measures the difference between the predicted output of a model and the actual target value. It helps in optimizing the model by guiding adjustments to its parameters during training. Examples include Mean Squared Error (MSE) for regression and Cross-Entropy Loss for classification.

23. Define Stochastic Gradient Descent with merits and demerits.

Stochastic Gradient Descent (SGD) is a variant of the Gradient Descent algorithm used for optimizing machine learning models. In this variant, only one random training example is used to calculate the gradient and update the parameters at each iteration. Here are some of the advantages and disadvantages of using SGD:

Advantages of Gradient Descent

Speed: SGD is faster than other variants of Gradient Descent such as Batch Gradient Descent and Mini-Batch Gradient Descent since it uses only one example to update the parameters.

Memory Efficiency: Since SGD updates the parameters for each training example one at a time, it is memory-efficient and can handle large datasets that cannot fit into memory.

Avoidance of Local Minima: Due to the noisy updates in SGD, it has the ability to escape from local minima and converges to a global minimum.

Disadvantages of Gradient Descent

Noisy updates: The updates in SGD are noisy and have a high variance, which can make the optimization process less stable and lead to oscillations around the minimum.

Slow Convergence: SGD may require more iterations to converge to the minimum since it updates the parameters for each training example one at a time.

Sensitivity to Learning Rate: The choice of learning rate can be critical in SGD since using a high learning rate can cause the algorithm to overshoot the minimum, while a low learning rate can make the algorithm converge slowly.

Less Accurate: Due to the noisy updates, SGD may not converge to the exact global minimum and can result in a suboptimal solution. This can be mitigated by using techniques such as learning rate scheduling and momentum-based updates

24. What is a deep feedforward network?

In a feedforward network, the information moves only in the forward direction, from the input layer, through the hidden layers (if they exist), and to the output layer. There are no cycles or loops in this network. Feedforward neural networks are sometimes ambiguously called multilayer perceptron.

25. What is the working principle of a feed forward neural network?

When the feed forward neural network gets simplified, it can appear as a single layer perceptron.
This model multiplies inputs with weights as they enter the layer. Afterward, the weighted input values get added together to get the sum. As long as the sum of the values rises above a certain threshold, set at zero, the output value is usually 1, while if it falls below the threshold, it is usually -1.

26.Define Neural networks are considered universal function approximators

Neural networks are considered universal function approximators, meaning they can approximate any continuous function given sufficient neurons and layers. This is based on the Universal Approximation Theorem, which states that a feedforward neural network with at least one hidden layer and a non-linear activation function can approximate any continuous function on a closed interval with arbitrary accuracy. This makes neural networks powerful tools for complex pattern recognition and machine learning tasks.

27. Brief on classification of activation function.

An activation function can be classified into three major categories: sigmoid, Tanh, and Rectified Linear Unit (ReLu).

🎬 Sigmoid:

Input values between 0 and 1 get mapped to the output values.

🎬 Tanh:

A value between -1 and 1 gets mapped to the input values.

🎬 Rectified linear Unit:

Only positive values are allowed to flow through this function. Negative values get mapped to 0.

28. What is Regularization?

Regularization is a technique used in machine learning and deep learning to prevent overfitting and improve the generalization performance of a model. It involves adding a penalty term to the loss function during training. This penalty discourages the model from becoming too complex or having large parameter values, which helps in controlling the model's ability to fit noise in the training data. Regularization methods include L1 and L2 regularization, dropout, early stopping, and more.

29. What is dropout in neural network?

Dropout is a regularization technique used in neural networks to prevent overfitting. During training, a random subset of neurons is "dropped out" by setting their outputs to zero with a certain probability. This forces the network to learn more robust and independent features, as it cannot rely on specific neurons. Dropout improves generalization and reduces the risk of overfitting.

30. Difference between regularization and optimization.

The main conceptual difference is that optimization is about finding the set of parameters/weights that maximizes/minimizes some objective function (which can also include a regularization term), while regularization is about limiting the values that your parameters can take during the optimization/learning/training, so optimization with regularisation (especially, with L1 and L2 regularization) can be thought of as constrained optimization, but, in some cases, such as dropout, it can also be thought of as a way of introducing noise in the training process.

PART C

1. Explain the working of Support Vector Machines (SVM) in detail. Discuss the role of the kernel trick in SVMs.

Support Vector Machine (SVM) Explained

SVM is a supervised learning algorithm used for classification and regression tasks. It works by finding the best boundary (hyperplane) to separate different classes in a dataset.

1. Understanding SVM Intuitively

Imagine you have two types of objects (e.g., apples and oranges) plotted on a graph. Your goal is to find a line (in 2D) or a plane (in 3D) that best separates them.

Key Concepts

- Hyperplane: The decision boundary that separates different classes.
- Support Vectors: The data points closest to the hyperplane, which influence its position.
- Margin: The distance between the hyperplane and the nearest support vectors. SVM aims to maximize this margin for better separation.

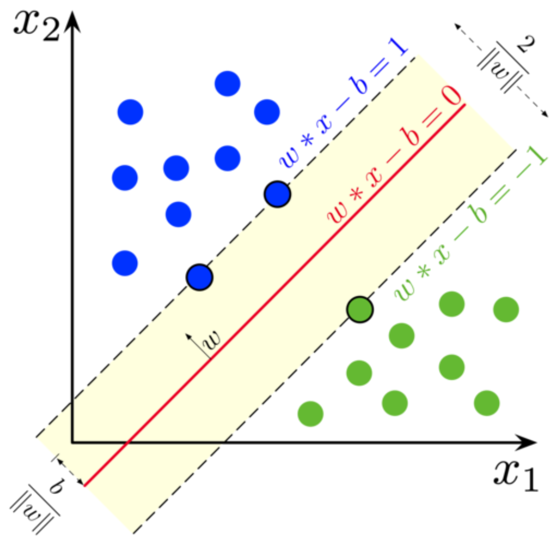
How SVM Works?

1. Find the Best Hyperplane: Among all possible lines that separate the data, SVM picks the one with the widest margin (maximizing the gap between the classes).
2. Support Vectors Influence the Decision: The closest points (support vectors) determine the hyperplane's position.
3. Soft Margin for Overlapping Data: If data isn't perfectly separable, SVM allows some misclassification while still finding a good boundary.
4. Kernel Trick for Complex Data: If the data isn't linearly separable, SVM transforms it into a higher dimension where it becomes separable.

2. Visual Representation of Key Concepts

Here are some images explaining SVM:

1. Basic SVM Hyperplane and Support Vector



1. Kernel Trick: Converting Non-Linearly Separable Data into Linearly Separable Data

3. Why Use SVM?

✅ Works well for both linear and non-linear data.
✅ Effective in high-dimensional spaces.
✅ Robust against overfitting (especially with the right kernel).
✅ Used in applications like text classification, image recognition, and bioinformatics.

2. Describe the perceptron learning algorithm and explain its convergence properties with a suitable example.

Perceptron Model

1.5.1 Simple Perceptron for Pattern Classification

Perceptron network is capable of performing pattern classification into two or more categories. The perceptron is trained using the perceptron learning rule. We will first consider classification into two categories and then the general multiclass classification later. For classification

into only two categories, all we need is a single output neuron. Here we will use bipolar neurons. The simplest architecture that could do the job consists of a layer of N input neurons, an output layer with a single output neuron, and no hidden layers. This is the same architecture as we saw before for Hebb learning. However, we will use a different transfer function here for the output neurons as given below in eq (7). Figure 7 represents a single layer perceptron network.

$$y = \begin{cases} 1 & \text{if } y\_in > \theta \\ 0 & \text{if } -\theta \leq y\_ \\ -1 & \text{if } y\_in < - \end{cases} \quad \text{eq (7)}$$

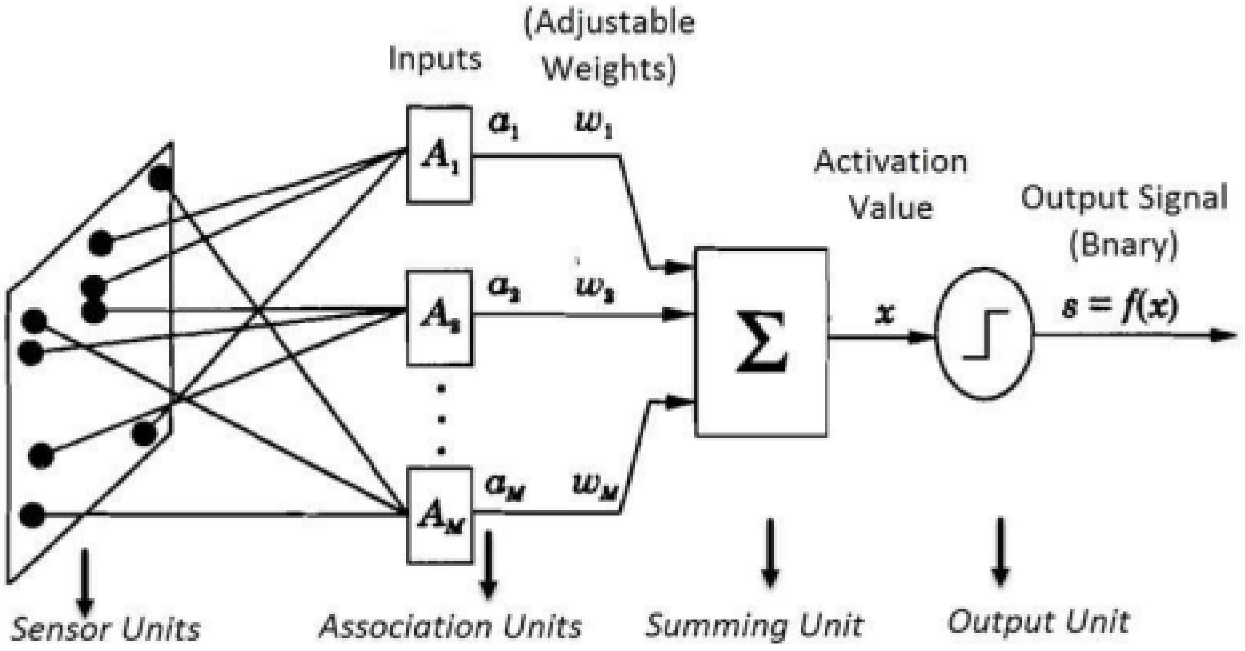


Figure 4: Single Layer Perceptron

Equation 7 gives the bipolar activation function which is the most common function used in the perceptron networks. Figure 7 represents a single layer perceptron network. The inputs arising from the problem space are collected by the sensors and they are fed to the aswociation units.Association units are the units which are responsible to associate the inputs based on their similarities. This unit groups the similar inputs hence the name association unit. A single input from each group is given to the summing unit.Weights are randomnly fixed intially and assigned to this inputs. The net value is calculate by using the expression

$$x = \Sigma\ wiai - \theta \quad eq(8)$$

This value is given to the activation function unit to get the final output response.The actual output is compared with the Target or desired .If they are same then we can stop training else the weights haqs to be updated .It means there is error .Error is given as δ = b-s , where b is the desired

/ Target output and S is the actual outcome of the machinehere the weights are updated based on the perceptron Learning law as given in equation 9.

Weight change is given as Δw= $\eta\ \delta$ ai. So new weight is given as

$$Wi\ (new) = Wi\ (old) + \text{Change in weight vector } (\Delta w) \quad eq(9)$$

1.5.2. Perceptron Algorithm

Step 1: Initialize weights and bias.For simplicity, set weights and bias to zero.Set learning rate in the range of zero to one.

- Step 2: While stopping condition is false do steps 2-6
- Step 3: For each training pair s:t do steps 3-5
- Step 4: Set activations of input units *xi = ai*
- Step 5: Calculate the summing part value Net = Σ aiwi-θ
- Step 6: Compute the response of output unit based on the activation functions • Step 7: Update weights and bias if an error occurred for this pattern(if *yis not equal to t) Weight (new) = wi(old) + atxi , & bias (new) = b(old) + at*

  Else *wi(new) = wi(old) & b(new) = b(old)*
- Step 8: Test Stopping Condition

1.5.3. Limitations of single layer perceptrons:

- Uses only Binary Activation function
- Can be used only for Linear Networks
- Since uses Supervised Learning ,Optimal Solution is provided
- Training Time is More
- Cannot solve Linear In-separable Problem

1.5.4. Multi-Layer Perceptron Model:

Figure 8 is the general representation of Multi layer Perceptron network.Inbetween the input and output Layer there will be some more layers also known as Hidden layers.

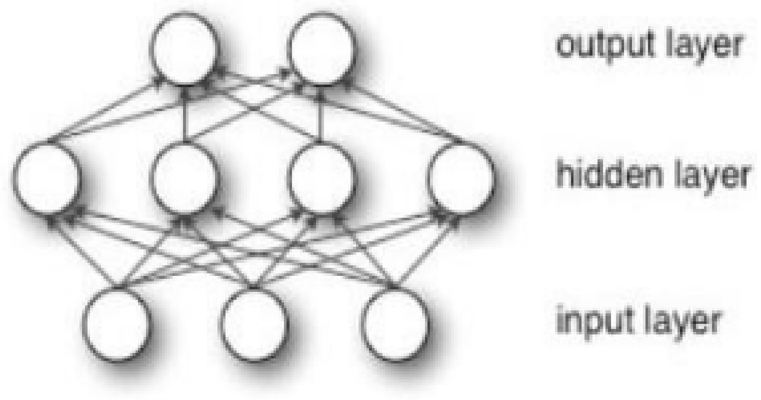


Figure 5: Multi-Layer Perceptron

1.5.5. Multi Layer Perceptron Algorithm

1. Initialize the weights (Wi) & Bias (B0) to small random values near Zero 2. Set learning rate η or α in the range of "0" to "1"

3. Check for stop condition. If stop condition is false do steps 3 to 7

4. For each Training pairs do step 4 to 7

5. Set activations of Output units: xi = si for i=1 to N

6. Calculate the output Response

$$yin = b0 + \Sigma\, xiwi$$

7. Activation function used is Bipolar sigmoidal or Bipolar Step functions For Multi Layer networks, based on the number of layers steps 6 & 7 are repeated 8. If the Targets is (not equal to) = to the actual output (Y), then update weights and bias based on Perceptron Learning Law

Wi (new) = Wi (old) + Change in weight vector

Change in weight vector = ηtixi

Where η = Learning Rate

ti = Target output of i^{th} unit

xi = i^{th} Input vector

b0(new) = b0 (old) + Change in Bias

Change in Bias = ηti

Else Wi (new) = Wi (old)

b0(new) = b0 (old)

9. Test for Stop condition

3. Explain logistic regression in detail. How does it differ from linear regression? Discuss its applications in machine learning.

Logistic Regression

Logistic regression is a probabilistic model that organizes the instances in terms of probabilities. Because the classification is probabilistic, a natural method for optimizing the parameters is to ensure that the predicted probability of the observed class for each training occurrence is as large as possible. This goal is achieved by using the notion of maximumlikelihood estimation in order to learn the parameters of the model. The likelihood of the training data is defined as the product of the probabilities of the observed labels of each training instance. Clearly, larger values of this objective function are better. By using the negative logarithm of this value, one obtains a loss function in minimization form. Therefore, the output node uses the negative log-likelihood as a loss function. This loss function replaces the squared error used in the Widrow-Hoff method. The output layer can be formulated with the sigmoid activation function, which is very common in neural network design.

10

- Logistic regression is another supervised learning algorithm which is used to solve the classification problems. In classification problems, we have dependent variables in a binary or discrete format such as 0 or 1.

- Logistic regression algorithm works with the categorical variable such as 0 or 1, Yes or No, True or False, Spam or not spam, etc.

- It is a predictive analysis algorithm which works on the concept of probability.

- Logistic regression is a type of regression, but it is different from the linear regression algorithm in the term how they are used.

- Logistic regression uses sigmoid function or logistic function which is a complex cost function. This sigmoid function is used to model the data in logistic regression. The function can be represented as:

34

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

Where f(x)= Output between the 0 and 1 value.

x= input to the function
e= base of natural logarithm.

When we provide the input values (data) to the function, it gives the S curve as follows: It uses the concept of threshold levels, values above the threshold level are rounded up to 1, and values below the threshold level are rounded up to 0.

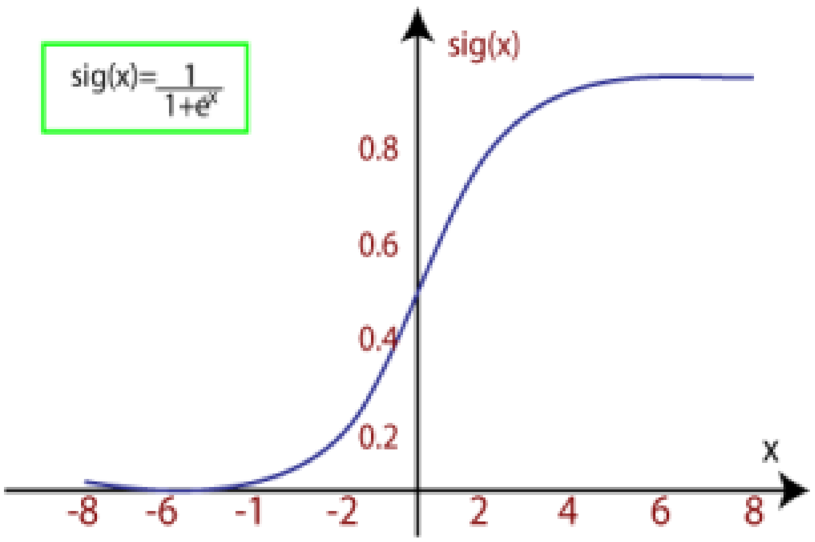


Figure 10: Circle – Logistic Function

4. Compare and contrast Support Vector Machines (SVM), Perceptrons, and Logistic Regression. Provide real-world examples of their use cases

1. Support Vector Machines (SVM)

Concept:

- SVM aims to find the optimal hyperplane that maximizes the margin (the distance between the hyperplane and the closest data points of each class, called support vectors).
- SVM works well for both linear and non-linear classification tasks, using a kernel trick for non-linearly separable data.

Key Characteristics:

- Kernel Trick: SVM can map input data into a higher-dimensional space, enabling it to classify data that isn't linearly separable in the original space.
- Margin Maximization: SVM tries to create the widest margin between classes, aiming for a more generalized model.

Advantages:

- Effective for high-dimensional spaces.
- Performs well with a clear margin of separation.
- Can handle non-linear data using kernels.
- Less prone to overfitting, especially in high-dimensional space.

Disadvantages:

- Computationally expensive, especially for large datasets.
- SVMs are harder to interpret compared to other models.
- Requires careful tuning of parameters like the regularization parameter (C) and kernel type.

Real-World Use Cases:

- Text Classification: SVMs are widely used for text classification tasks such as spam email detection and sentiment analysis.
- Image Recognition: Used in face recognition and object detection in images.
- Bioinformatics: For example, classifying proteins into categories.

2. Perceptrons

Concept:

- The Perceptron is a simple linear classifier that makes a prediction based on a weighted sum of input features, using a threshold function (activation function) to classify data into two classes.
- It is online (updates weights after seeing each training example) and can be trained with gradient descent.

Key Characteristics:

- Linear: It can only solve problems that are linearly separable.
- Binary Classifier: It predicts either class 0 or class 1.

Advantages:

- Simple and fast to train.
- Easy to understand and interpret.
- Works well when the data is linearly separable.

Disadvantages:

- Cannot handle non-linearly separable data. For example, it fails in cases like the XOR problem.
- The model is sensitive to the choice of the learning rate.
- Does not guarantee convergence if the data is not linearly separable.

Real-World Use Cases:

- Early Neural Networks: The Perceptron is the foundation of more complex neural networks like Multi-Layer Perceptrons (MLP).
- Binary Classification Tasks: Can be used for simple binary classification problems with linearly separable data, such as classifying basic geometric shapes based on features like area or perimeter.

3. Logistic Regression

Concept:

- Logistic Regression is a probabilistic classifier that models the probability of a data point belonging to class 1. It applies the sigmoid function to the linear combination of features to output values between 0 and 1.
- It estimates probabilities using logistic loss (cross-entropy), making it more suitable for classification problems than linear regression.

Key Characteristics:

- Linear: Logistic regression assumes a linear decision boundary between classes.
- Probabilistic: Outputs the probability of a class (values between 0 and 1), making it interpretable as the likelihood of a given class.
- Binary Classifier: Primarily used for binary classification, but extensions like multinomial logistic regression can handle multiclass problems.

Advantages:

- Simpler and more interpretable than SVMs.
- Can be regularized to prevent overfitting.
- Works well when the classes are linearly separable but can also handle some noise.
- Provides class probabilities, making it suitable for tasks where understanding uncertainty is important.

Disadvantages:

- Struggles with non-linear boundaries unless extended (e.g., using polynomial features or kernel tricks).
- Sensitive to outliers, which can influence the decision boundary.
- Assumes a linear relationship between the features and the log-odds of the outcome.

Real-World Use Cases:

- Medical Diagnosis: Predicting whether a patient will develop a disease (e.g., predicting diabetes or heart disease based on patient data).
- Marketing & Sales: Predicting customer churn (whether a customer will leave or stay).
- Financial Predictions: Classifying whether a loan applicant is creditworthy or not.

.

4.What is backpropagation? Explain the step-by-step working of backpropagation in training a neural network with an example.

Back Propagation Networks (BPN)

Need for Multilayer Networks

- Single Layer networks cannot used to solve Linear Inseparable problems & can only be used to solve linear separable problems
- Single layer networks cannot solve complex problems
- Single layer networks cannot be used when large input-output data set is available
- Single layer networks cannot capture the complex information's available in the training pairs

Hence to overcome the above said Limitations we use Multi-Layer

Networks.

Multi-Layer Networks

•Any neural network which has at least one layer in between input and output layers is called Multi-Layer Networks

• Layers present in between the input and out layers are called Hidden Layers • Input layer neural unit just collects the inputs and forwards them to the next higher layer

•Hidden layer and output layer neural units process the information's feed to them and produce an appropriate output

•Multi -layer networks provide optimal solution for arbitrary classification problems

•Multi -layer networks use linear discriminants, where the inputs are non linear

Back Propagation Networks (BPN)

Introduced by Rumelhart, Hinton, & Williams in 1986. BPN is a Multi layer Feedforward Network but error is back propagated, Hence the name Back Propagation Network (BPN). It uses Supervised Training process; it has a systematic procedure for training the network and is used in Error Detection and Correction. Generalized Delta Law /Continuous Perceptron Law/ Gradient Descent Law is used in this network. Generalized Delta rule minimizes the mean squared error of the output calculated from the output. Delta law has faster convergence rate when compared with Perceptron Law. It is the extended version of Perceptron Training Law. Limitations of this law is the Local minima problem. Due to this the convergence speed reduces, but it is better than perceptron's. Figure 1 represents a BPN network architecture. Even though Multi level perceptron's can be used they are flexible and efficient that BPN. In figure 1 the weights between input and the hidden portion is considered as Wij and the weight between first hidden to the next layer is considered as Vjk. This network is valid only for Differential Output functions. The Training process used in backpropagation involves three stages, which are listed as below

1. Feedforward of input training pair
2. Calculation and backpropagation of associated error
3. Adjustments of weights

The algorithm for BPN is as classified int four major steps as

follows: 1. Initialization of Bias, Weights

2. Feedforward process

3. Back Propagation of Errors

4. Updating of weights & biases

Algorithm:

```
I. Initialization of weights:
    Step 1: Initialize the weights to small random values near zero
    Step 2: While stop condition is false , Do steps 3 to 10
    Step 3: For each training pair do steps 4 to 9
II. Feed forward of inputs
    Step 4: Each input xi is received and forwarded to higher
    layers (next hidden)
    Step 5: Hidden unit sums its weighted inputs as follows
                $Zinj = Woj + \Sigma xiwij$
        Applying Activation function
                $Zj = f(Zinj)$
            This value is passed to the output layer
    Step 6: Output unit sums it's weighted inputs
        $yink= Voj + \Sigma ZjVjk$
            Applying Activation function
```

```
                $Yk = f(yink)$
III. Backpropagation of Errors
    Step 7: $\delta k = (tk – Yk)f(yink )$
```

```
            Step 8: $\delta inj = \Sigma \delta jVjk$
    IV. Updating of Weights & Biases
            Step 8: Weight correction is $\Delta wij = \alpha\delta kZj$
                    bias Correction is $\Delta woj = \alpha\delta k$
    V. Updating of Weights & Biases
          Step 9: continued:
                New Weight is
                        $Wij(new) = Wij(old) + \Delta wij$
                        $Vjk(new) = Vjk(old) + \Delta Vjk$
                New bias is
                        $Woj(new) = Woj(old) + \Delta woj$
                        $Vok(new) = Vok(old) + \Delta Vok$

          Step 10: Test for Stop Condition
```

Merits

- Has smooth effect on weight correction
- Computing time is less if weight's are small
- 100 times faster than perceptron model
- Has a systematic weight updating procedure

Demerits

- Learning phase requires intensive calculations
- Selection of number of Hidden layer neurons is an issue
- Selection of number of Hidden layers is also an issue
- Network gets trapped in Local Minima
- Temporal Instability
- Network Paralysis
- Training time is more for Complex problems

5. Discuss different types of loss functions used in neural networks. How do they influence model performance? Compare loss functions used in classification and regression problems.

Loss Functions in Neural Networks

A loss function (or cost function) is a key component of any machine learning algorithm, especially neural networks. It measures the difference between the predicted output and the true output (target), helping the model adjust its weights during training. The choice of loss function greatly influences the model's learning process, convergence speed, and final performance.

Loss functions can be broadly classified into classification loss functions and regression loss functions, depending on the nature of the problem you're solving.

---

1. Loss Functions for Classification Problems

Classification tasks involve predicting categorical labels (e.g., 0 or 1, "cat" or "dog"). These problems require loss functions that measure how well the predicted probabilities match the true class labels.

Common Loss Functions for Classification:

a. Cross-Entropy Loss (Log Loss)

Use Case: Binary and multiclass classification problems.

b. Hinge Loss (for SVMs)

- Use Case: Typically used in Support Vector Machines (SVM) and models based on margin maximization.

c. Sparse Categorical Cross-Entropy Loss

- Use Case: Multiclass classification where labels are provided as integers (not one-hot encoded).
- Formula:
  - Similar to categorical cross-entropy, but the target labels are integers instead of vectors.
  - Often used in multiclass classification problems where the output layer has multiple neurons (each representing a class).

2. Loss Functions for Regression Problems

Regression tasks involve predicting continuous values (e.g., house prices, stock prices). These problems require loss functions that measure the difference between predicted numerical values and the true target values.

Common Loss Functions for Regression:

a. Mean Squared Error (MSE) Loss

- Use Case: General regression problems.
- Explanation:
  - MSE calculates the average of the squared differences between predicted values and actual values. It penalizes larger errors more heavily, which helps in optimizing models for precise predictions.
- Influence on Model Performance:
  - MSE is sensitive to outliers because large errors are squared, making the model more likely to adjust its parameters to minimize large discrepancies.
  - It works well when the data is relatively clean and doesn’t contain significant outliers.

b. Mean Absolute Error (MAE) Loss

- Use Case: When robustness to outliers is important.
- Explanation:
  - MAE calculates the average of the absolute differences between predicted and true values. Unlike MSE, it doesn’t heavily penalize larger errors.
- Influence on Model Performance:
  - MAE is less sensitive to outliers compared to MSE, which can make it a better choice when there are significant deviations in the data.
  - However, it can lead to less stable gradients during training because it doesn't emphasize large errors as much as MSE.

c. Huber Loss

- Use Case: A combination of MSE and MAE that balances between squaring large errors and ignoring them.
- Explanation:
  - Huber loss behaves like MSE for small errors and like MAE for large errors, which helps avoid the large penalty from outliers while still maintaining smooth gradients.
- Influence on Model Performance:
  - Huber loss is a good choice for regression problems with noisy data or outliers, as it allows the model to learn in the presence of both small and large errors effectively.

6. Explain stochastic gradient descent (SGD). How does it differ from batch gradient descent? Discuss its importance in training deep learning models.

Stochastic Gradient Descent (SGD):

The word ‘stochastic‘ means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, but the problem arises when our datasets gets big. Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima is reached. Hence, it becomes computationally very expensive to perform.

7.State and explain the Universal Approximation Theorem. How does it demonstrate the power of neural networks in function approximation?

Universal Approximation Theorem (UAT)

The Universal Approximation Theorem is a fundamental result in the theory of neural networks. It states that a feedforward neural network with at least one hidden layer and a sufficient number of neurons can approximate any continuous function on a compact domain to any desired degree of accuracy, provided that the activation function is non-linear (e.g., sigmoid, ReLU).

In simpler terms, the theorem asserts that neural networks are capable of learning and approximating any function, given the right conditions, no matter how complex or intricate the function may be.

Formal Statement of the Universal Approximation Theorem:

Given:

- A continuous function fff defined on a compact domain (e.g., a closed interval).
- A feedforward neural network with one hidden layer and non-linear activation functions.

The Universal Approximation Theorem states that there exist:

Weights and biases in the network such that the neural network can approximate fff within any desired degree of accuracy.

Explanation and Key Concepts:

1. Feedforward Neural Networks: The theorem applies to networks with a structure that includes an input layer, one hidden layer, and an output layer. The hidden layer uses a non-linear activation function, which gives the network the ability to model complex relationships.
2. Non-Linear Activation Functions: The key to the power of the Universal Approximation Theorem is that the hidden layer neurons use a non-linear activation function (like sigmoid, ReLU, or tanh). This allows the network to create complex decision boundaries and non-linear mappings between inputs and outputs.
3. Compact Domain: The theorem works on functions that are continuous over a compact domain. This usually means that the domain of the input values is bounded and closed (e.g., a finite range of inputs). This condition is typically satisfied in most real-world problems.
4. Approximation with Arbitrary Accuracy: The theorem states that for any continuous function and any desired error margin ϵ\epsilonϵ, there exists a network that can approximate the function to within that margin. The number of neurons in the hidden layer may need to be very large, but theoretically, a neural network can always approximate the function with enough neurons.

Implications for Neural Networks:

1. Power of Neural Networks in Function Approximation:

- General Function Approximation: The UAT shows that neural networks have the potential to approximate any continuous function, no matter how complex, provided enough neurons and proper training. This is a cornerstone of neural network theory, highlighting that neural networks are extremely powerful tools for learning and representing a wide range of functions.
- Modeling Complex Relationships: Many real-world problems involve highly complex and non-linear relationships between inputs and outputs (e.g., image recognition, speech processing). The UAT guarantees that neural networks can model these complexities, given sufficient resources (neurons and training data).
- Flexibility: Neural networks are not limited to a particular class of functions or patterns. The UAT implies that neural networks are universal function approximators that can be used for a broad variety of tasks, from regression to classification.

2. Practical Considerations:

- Depth vs. Width: Although the UAT states that one hidden layer is sufficient for approximation, in practice, deep neural networks (with multiple layers) tend to perform better. This is because deeper networks often learn more efficient representations of complex data, and fewer neurons may be needed in each layer.
- Training and Overfitting: Even though a neural network can theoretically approximate any function, the process of training the network is non-trivial. In practice, training can be computationally expensive and prone to overfitting if the network is too large for the available data.
- Approximation Quality: The quality of the approximation depends on factors such as the size of the network, the training data, and the optimization process. If a network is not trained properly, it may fail to approximate the function well.

Real-World Example:

Function Approximation in Regression:

Imagine you have a function f(x)=sin(x)f(x) = \sin(x)f(x)=sin(x), and you want to approximate this function using a neural network. The UAT assures us that, given a network with a sufficient number of neurons in the hidden layer, the network will be able to approximate f(x)f(x)f(x) closely, even though sin(x)\sin(x)sin(x) is a non-linear function.

In practice, you would train a neural network to fit data points generated from sin(x)\sin(x)sin(x), and as the network learns the relationships, it would approximate the sine curve.

For a simple regression problem like this, one hidden layer with enough neurons would be sufficient to achieve a near-perfect fit, demonstrating the power of neural networks in function approximation.

8.How can a neural network approximate any function? Explain the role of activation functions, layers, and weights in this process.

A neural network can approximate any continuous function by using its architecture, which is made up of layers, weights, and activation functions. The Universal Approximation Theorem guarantees that a sufficiently large neural network can approximate any continuous function to an arbitrary degree of accuracy. Let's break down how this happens step by step:

1. Layers and Neurons:

A neural network is composed of layers of neurons (also known as nodes). Each layer has a different role:

- Input Layer: This layer receives the raw data (features) as input to the network.
- Hidden Layers: These are the intermediate layers between the input and output layers. The network typically has multiple hidden layers, and each hidden layer learns different representations of the input data.
- Output Layer: This layer produces the final prediction or output of the neural network.

Each neuron in a layer performs computations on the inputs and passes the result to the next layer.

2. Weights and Biases:

The weights and biases in a neural network determine the strength of the connections between neurons and the offset applied to the output of a neuron.

- Weights: Each connection between two neurons has a weight that determines how much influence one neuron has on another. The higher the weight, the stronger the connection between the neurons.
- Biases: A bias is added to the weighted sum of the inputs to a neuron. It allows the network to make predictions even when all inputs are zero, and it shifts the activation function's output.

These weights and biases are learned during training through a process called backpropagation, which adjusts the weights to minimize the difference between predicted and actual values (the loss).

3. Activation Functions:

Activation functions are mathematical functions applied to the output of a neuron. They play a key role in introducing non-linearity into the network, which is what allows neural networks to approximate complex, non-linear functions.

Why Non-Linearity is Important:

- If all activation functions were linear (e.g., just a weighted sum of inputs), the neural network would essentially behave like a single linear transformation. It wouldn't be able to capture complex relationships between input and output.
- Non-linear activation functions allow neural networks to learn complex, non-linear relationships and map inputs to outputs in a highly flexible way.

Common Activation Functions:

- Sigmoid: Produces an output between 0 and 1. Often used in binary classification problems.
- Tanh (Hyperbolic Tangent): Similar to sigmoid but produces output between -1 and 1. It's useful for centering the data around zero.
- ReLU (Rectified Linear Unit): Outputs zero for negative inputs and the input value itself for positive inputs. It's very popular for deep networks due to its simplicity and effectiveness in preventing vanishing gradients.
- Softmax: Used for multiclass classification problems to normalize the output so that it represents probabilities of different classes.

How Activation Functions Enable Function Approximation:

- The non-linearity introduced by activation functions allows the neural network to approximate non-linear functions, not just straight lines. For instance, in a deep neural network, each layer's neurons learn to extract higher-level abstractions, and the non-linear activation functions allow the network to combine these abstractions in a complex way.
- Without non-linear activation functions, the network would be limited to solving only linear problems, making it far less powerful in practice.

4. The Process of Function Approximation:

1. Initial Learning:

When the neural network starts training, it randomly initializes its weights and biases. Initially, these random values lead to random predictions.

2. Forward Propagation:

- During forward propagation, the input data is passed through the network, layer by layer. Each neuron computes a weighted sum of the inputs, adds the bias, and passes it through an activation function to generate the output for that neuron.
- This process is repeated for each layer until the final output is produced.

3. Backpropagation and Gradient Descent:

- Once the network produces an output, it calculates the error (the difference between the predicted and actual values) using a loss function (e.g., mean squared error for regression or cross-entropy for classification).
- The error is then propagated backwards through the network (using backpropagation), and the weights and biases are updated using an optimization technique like gradient descent. This minimizes the error by adjusting the weights in the direction that reduces the loss.

4. Approximation:

- Through many iterations of forward propagation and backpropagation, the network gradually learns to adjust the weights and biases so that its output becomes closer to the desired target.
- After sufficient training, the network can approximate the target function with a high degree of accuracy. For example, for a regression problem, it can learn to predict continuous values. For classification, it can learn to assign inputs to specific classes.

How Neural Networks Approximate Functions:

1. Learning Complex Mappings:
   A neural network with multiple layers can progressively learn increasingly complex representations of the data. Each hidden layer captures features or patterns from the data, and the final output layer combines these patterns to make a prediction.
2. Linear Combinations of Features:
   The neurons in the hidden layers combine the input features in non-linear ways. These combinations help the network approximate more complicated functions by effectively "splitting" the input space into regions, each corresponding to a different output.
3. Adjusting Weights:
   Through training, the network adjusts the weights to minimize the error between the predicted output and the true target. This process of tuning weights allows the neural network to approximate the function more closely as training progresses.
4. Flexibility and Expressiveness:
   The greater the number of neurons in the hidden layer, the more flexible the network becomes in approximating complex functions. With enough neurons, the network can approximate any continuous function. For highly complex functions, a deeper network (with multiple hidden layers) may be required to learn efficient representations of the data.

9.What are the limitations of neural networks despite being universal function approximators? Discuss challenges like overfitting, vanishing gradients, and computational complexity.

Despite being universal function approximators, neural networks have several limitations and challenges that can make them difficult to train and apply effectively in real-world scenarios. Some of the key challenges include overfitting, vanishing gradients, and computational complexity. Let's explore these in detail:

1. Overfitting:

Overfitting occurs when a model learns not just the underlying patterns in the data but also the noise or random fluctuations. In essence, the model becomes too complex and fits the training data too closely, which can lead to poor generalization to new, unseen data.

- Symptoms of Overfitting:
    - The model performs well on the training data but poorly on validation or test data.
    - High variance and low bias.
- Why Overfitting Happens: Neural networks, especially deep ones with many layers and parameters, are highly flexible. This flexibility allows them to learn intricate patterns in the data, but if the network is too complex for the given amount of training data, it might start memorizing specific details that are irrelevant to the general patterns. The model essentially "remembers" the noise, which results in poor performance on new, unseen data.
- Solutions to Overfitting:
    - Regularization: Techniques like L2 regularization (weight decay), dropout, and early stopping can help reduce overfitting by penalizing overly complex models and preventing them from fitting noise.
    - Data Augmentation: Increasing the amount of training data, either by collecting more data or applying transformations like rotation, scaling, and cropping, can help.
    - Cross-validation: Use cross-validation techniques to ensure the model generalizes well on unseen data.

---

2. Vanishing Gradients:

The vanishing gradient problem occurs primarily in deep neural networks when gradients become very small during backpropagation, causing the weights in earlier layers to stop changing significantly. This makes it difficult for the network to learn effectively, particularly in the early layers.

- Why It Happens:
    - In deep networks, during backpropagation, gradients are propagated backward from the output layer to the input layer. If activation functions like sigmoid or tanh are used, the gradients can shrink exponentially as they move backward through the layers.
    - For instance, if the derivative of an activation function is small (like for sigmoid, where the output is bounded between 0 and 1), the gradient will shrink exponentially as it moves backward through the layers.
- Consequences of Vanishing Gradients:
    - The network struggles to learn, especially in the early layers.
    - Weights in earlier layers receive very small updates, preventing them from

learning useful features.

- Solutions to Vanishing Gradients:
    - ReLU activation function: The Rectified Linear Unit (ReLU) is often used as an alternative to sigmoid and tanh because it has a gradient of 1 for positive inputs, which prevents gradients from vanishing.
    - He Initialization: Proper weight initialization techniques like He initialization can also help, particularly with ReLU, by ensuring that the weights are initialized in such a way that the variance of activations is maintained across layers.
    - Gradient Clipping: Clipping gradients during backpropagation can prevent extremely small gradients from vanishing or overly large gradients from exploding.

3. Exploding Gradients:

The exploding gradient problem is the opposite of vanishing gradients and occurs when gradients become too large during backpropagation, leading to excessively large updates to the weights. This can make the training process unstable and cause the network to diverge rather than converge.

- Why It Happens:
    - In deep networks, if the weights are initialized poorly or if the learning rate is too high, the gradients can grow exponentially as they are backpropagated through the layers, leading to very large weight updates.
- Consequences of Exploding Gradients:
    - Training becomes unstable, and the network fails to converge.
    - Weights can become NaN (Not a Number), causing the training process to crash.
- Solutions to Exploding Gradients:
    - Gradient Clipping: Similar to vanishing gradients, gradient clipping can be applied to prevent the gradients from growing too large.
    - Better Initialization: Proper initialization methods like Xavier or He initialization can help control the magnitude of gradients.
    - Lower Learning Rate: A smaller learning rate can prevent the updates from being too large and causing instability.

4. Computational Complexity:

Training neural networks, especially deep ones, can be computationally expensive and time-consuming. The complexity arises from the following factors:

- Large Number of Parameters: Deep neural networks, particularly those with many layers and neurons, can have millions (or even billions) of parameters. This increases the memory requirements and the amount of computation needed to update the weights during training.

- Training Time: Neural networks often require large datasets and several iterations (epochs) to converge. Training on large datasets can take a long time, especially without sufficient computational resources (like GPUs or TPUs).
- Hardware Requirements: Training deep neural networks often requires specialized hardware like Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs) to handle the large-scale computations involved in matrix multiplications and other operations.
- Solutions to Computational Complexity:
  - Efficient Architectures: Techniques like model pruning, knowledge distillation, and quantization can help reduce the size and complexity of models without sacrificing too much performance.
  - Transfer Learning: Pre-trained models can be fine-tuned on a specific task, reducing the need to train a network from scratch and saving computational resources.
  - Parallelization: Training neural networks can be distributed across multiple machines or GPUs to speed up the process.

5. Interpretability:

Despite their power, neural networks are often considered "black-box" models. This means that once a neural network has been trained, it's often difficult to understand why the model made a particular decision or prediction.

- Why This is a Problem:
  - In critical areas like healthcare, finance, and autonomous driving, understanding the reasoning behind a model's decisions is crucial.
  - Lack of interpretability can limit trust in neural networks and hinder their adoption in sensitive applications.
- Solutions:
  - Explainable AI (XAI): Techniques like LIME (Local Interpretable Model-agnostic Explanations) and SHAP (SHapley Additive exPlanations) can help explain the decisions made by neural networks.
  - Model simplification: Using simpler models or hybrid models (e.g., combining neural networks with decision trees) can offer more interpretable solutions.

6. Data Requirements:

Neural networks require a large amount of labeled data to perform well, especially for deep learning models. Insufficient data can lead to poor model performance and overfitting.

- Challenges with Data:
  - Data Scarcity: In some domains, acquiring labeled data can be costly and time-consuming (e.g., medical image labeling).

    - Bias in Data: If the data is biased or unrepresentative, the neural network will learn those biases and make biased predictions.
- Solutions:
    - Data Augmentation: Artificially increasing the dataset size through transformations can help mitigate data scarcity.
    - Transfer Learning: Pre-trained models can be adapted to new tasks with smaller datasets, helping overcome data limitations.

10. Explain the role of activation functions in neural networks. Compare different activation functions (ReLU, Sigmoid, Tanh) and their impact on network performance.

Role of Activation Functions in Neural Networks

Activation functions play a crucial role in neural networks. They introduce non-linearity into the model, allowing the network to learn complex relationships in data. Without activation functions, the network would essentially be a linear transformation, limiting its ability to solve complex tasks. The key roles of activation functions are:

1. Non-Linearity: Neural networks need non-linear activation functions to model non-linear relationships in data. If the activation function were linear, no matter how many layers you add to the network, it would behave like a single linear transformation (i.e., a linear regression model).
2. Enabling Complex Functions: By applying an activation function to the output of each neuron, networks can approximate any continuous function, which is central to their universal approximation power.
3. Controlling Output: Activation functions control the output of each neuron, determining whether or not the neuron should "fire" based on the input it receives.

Comparison of Common Activation Functions

1. ReLU (Rectified Linear Unit)

- Function: f(x)=max(0,x)f(x) = \max(0, x)f(x)=max(0,x)
    - If the input is positive, the output is the same as the input.
    - If the input is negative, the output is zero.
- Properties:
    - Non-linear: Allows the network to learn non-linear patterns.
    - Computationally Efficient: ReLU is simple to compute and less expensive than other functions (like sigmoid or tanh), especially when working with large datasets.
    - Sparsity: ReLU can produce sparse representations since it outputs zero for all negative inputs. This can lead to more efficient models and prevent overfitting.
- Impact on Network Performance:
    - Advantages:
        - Faster training: Because it doesn't saturate for positive inputs, ReLU helps

avoid the vanishing gradient problem and often leads to faster convergence during training.
        - Effective for deep networks: Works well for deep networks with many layers.
    - Disadvantages:
        - Dying ReLU problem: If a large gradient flows through a ReLU neuron, the neuron may stop updating during training and "die," i.e., its output will always be zero.
        - Not zero-centered: ReLU outputs only positive values, which can cause issues during training (e.g., gradients could be unbalanced).
- Use Cases:
    - ReLU is widely used in deep learning models, especially in convolutional neural networks (CNNs) and feedforward neural networks.

2. Sigmoid (Logistic Function)

    - The output is between 0 and 1, making it useful for binary classification problems.
- Properties:
    - Non-linear: Allows the network to model complex patterns.
    - Output Range: Produces values between 0 and 1, making it suitable for probabilities.
    - Smooth Gradient: The sigmoid function has a smooth gradient, which is helpful for optimization.
- Impact on Network Performance:
    - Advantages:
        - Probabilistic Interpretation: Sigmoid is ideal for tasks where outputs are expected to be probabilities, such as binary classification or logistic regression.
        - Differentiable: The function is differentiable, meaning it's suitable for backpropagation.
    - Disadvantages:
        - Vanishing Gradient Problem: Sigmoid suffers from the vanishing gradient problem for large positive or negative inputs, making it difficult to train deep networks. When inputs are large or small, the gradient is very close to zero, which slows down or prevents weight updates.
        - Not Zero-Centered: The output of the sigmoid function is always positive, which can make optimization more difficult as the gradients are not centered around zero.
        - Slow Convergence: Because of the vanishing gradient problem, training deep networks with sigmoid can be slow and inefficient.
- Use Cases:
    - Sigmoid is mainly used for binary classification tasks, especially in the output layer of a binary classification neural network.

3. Tanh (Hyperbolic Tangent)

    - Similar to sigmoid but produces outputs in the range of -1 to 1.
- Properties:
    - Non-linear: Like the sigmoid, tanh is non-linear and can model complex relationships.
    - Output Range: Unlike sigmoid, tanh produces outputs between -1 and 1, which can help center data and gradients during training.
- Impact on Network Performance:
    - Advantages:
        - Zero-centered: Since the output is between -1 and 1, it is zero-centered, which can help with faster convergence during optimization.
        - Smooth Gradient: The smooth nature of the gradient is useful for backpropagation.
    - Disadvantages:
        - Vanishing Gradient Problem: Like sigmoid, tanh also suffers from the vanishing gradient problem for large positive or negative inputs, causing difficulty in training deep networks.
        - Slower Convergence: Due to the vanishing gradients, tanh can be slow to train, especially for deeper networks.
- Use Cases:
    - Tanh is often used in recurrent neural networks (RNNs) for sequence modeling tasks because of its zero-centered output.