1. Explain deep belief network.

A Deep Belief Network (DBN) is a type of neural network that is composed of multiple layers of latent variables, or hidden layers, that are used to extract increasingly abstract features from input data. It is a generative model that can be used for unsupervised learning, which means that it can learn to identify patterns in data without being explicitly told what those patterns are.

The DBN architecture consists of multiple layers of Restricted Boltzmann Machines (RBMs), which are a type of neural network that can learn to represent the probability distribution of a set of input data. Each RBM layer takes the output of the previous layer as input and uses it to learn a new set of features that are more abstract than the previous layer. The first layer of the DBN takes the raw input data and learns a set of basic features, such as edges or corners, while the subsequent layers learn more complex features by building upon the features learned by the previous layer.

Once the DBN has been trained, it can be used for a variety of tasks, such as classification or generation. For example, it can be used for image recognition by training on a large dataset of images and then using the learned features to classify new images. It can also be used for image generation by starting with a random noise vector and passing it through the network to generate a new image.

Overall, DBNs have been shown to be effective at learning complex, hierarchical representations of data, and have been used in a wide range of applications, including computer vision, speech recognition, and natural language processing.

2. Write a python code to plot sigmoid function

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
x = np.linspace(-10, 10, 1000)
y = 1 / (1 + np.exp(-x) )
plt.figure(figsize=(10, 5))
plt.plot(x, y)
plt.legend(['sigmoid function'])
plt.show()
```

3. Describe data preprocessing with an example.

Data preprocessing is a crucial step in data analysis and machine learning, as it involves cleaning, transforming, and preparing raw data into a format that is suitable for further analysis. The goal of data preprocessing is to improve the quality of the data, reduce errors, and make it more consistent and standardized.

Here is an example of data preprocessing:

Suppose we have a dataset of customer reviews for a restaurant, which includes columns for the date of the review, the reviewer's name, the rating on a scale of 1 to 5, and the text of the review. Before we can analyze this data, we need to preprocess it.

Data cleaning: The first step is to clean the data by removing any irrelevant or redundant information, such as the reviewer's name, which is not necessary for our analysis. We also need to handle missing or null values by either removing the corresponding rows or filling them with appropriate values. For example, if the rating column is missing for a particular review, we can fill it with the average rating for that restaurant.

Data transformation: Next, we need to transform the data to make it more suitable for analysis. One common transformation is to convert categorical variables, such as the rating, into numerical values. In this case, we can map the rating scale from 1 to 5 to a range of 0 to 1 for easier analysis.

Data normalization: We also need to normalize the data to ensure that all features have a similar scale. For example, the length of the review text may vary widely, so we can normalize it by scaling it to a range of 0 to 1 based on the maximum length in the dataset.

Data encoding: Finally, we may need to encode the data into a format that can be easily processed by machine learning algorithms. For example, we can use a bag-of-words model to convert the review text into a vector of word frequencies, which can be used as input for a machine learning model.

Overall, data preprocessing is a crucial step in data analysis and can greatly affect the accuracy and reliability of the results. By cleaning, transforming, and preparing data in a consistent and standardized way, we can ensure that our analysis is based on high-quality data that accurately represents the real-world phenomenon we are studying.

4. Write a python program to find the derivation of f(x) = x^3-4*x^2+2 and plot f(x).

```
import matplotlib.pyplot as plt
import sympy as sp
x = sp.Symbol('x')
f = x*3 - 4*x*2 + 2
fprime = f.diff(x)
print(fprime)
x = np.linspace(-5, 5, 100)
plt.plot(x, y, label='f(x)')
plt.plot(x, dy, label='f\'(x)')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Function and its derivative')
plt.show()
```

5. Explain ReLU activation function.

ReLU, or Rectified Linear Unit, is an activation function commonly used in neural networks. It is a simple, yet effective, non-linear activation function that has been found to work well in many applications.

The ReLU activation function takes an input x and returns the maximum between 0 and x. In mathematical notation, it is defined as:

f(x) = max(0, x)

The ReLU function is piecewise linear and has a constant slope of 1 for positive values of x, and a slope of 0 for negative values of x. This means that ReLU can introduce non-linearity into the network, which is important for learning complex patterns in data.

One of the main advantages of ReLU over other activation functions is that it is computationally efficient to calculate, and has a simple gradient function. The gradient of ReLU is 1 for positive values of x and 0 for negative values of x, which makes it easy to compute the gradient of the cost function with respect to the network weights during backpropagation.

Another advantage of ReLU is that it helps to prevent the problem of vanishing gradients, which can occur when the gradients of the activation function become very small. ReLU ensures that the gradients are always non-zero for positive values of x, which can help to maintain the flow of information during training.

However, one potential drawback of ReLU is that it can cause a problem called "dying ReLU", where some neurons can become stuck at zero and stop contributing to the network output. This can be addressed by using variants of ReLU such as leaky ReLU or parametric ReLU, which introduce a small slope for negative values of x to prevent dying ReLU.

Overall, ReLU is a popular and effective activation function that is widely used in deep learning.

6. Write any four difference between RNN and MLP.

	RNN	MLP
Handling sequential data	RNNs are specifically designed to handle sequential data, such as time-series data or natural language text. They have loops in their architecture that allow information to be passed from one time step to the next, making them well-suited for processing sequential inputs.	MLPs, are designed for processing fixed-size inputs and cannot naturally handle sequential data.
Memory	RNNs have a form of memory that allows them to keep track of past inputs and use them to influence the current output. This makes them particularly useful for tasks that require context or long-term dependencies, such as language modeling or speech recognition.	M LPs, in contrast, do not have memory and only operate on the current input.

Architecture	RNNs have a unique architecture that includes recurrent connections, which allow information to flow backwards through the network. This means that the output of the network at a given time step can depend on both past and future inputs.	MLPs, in contrast, have a feedforward architecture and do not have any connections that allow information to flow backwards through the network.
Training	Training RNNs can be more challenging than training MLPs, due to the nature of the recurrent connections. Backpropagation through time (BPTT), the standard algorithm for training RNNs, can suffer from the vanishing or exploding gradients problem, which can make it difficult to learn long-term dependencies.	MLPs, in contrast, are typically easier to train and do not suffer from these problems to the same extent.

7. Explain model selection and bias variance trade-off.

Model selection is the process of choosing the best machine learning model from a set of candidate models that have been trained on a given dataset. The goal of model selection is to find a model that performs well on new, unseen data, while avoiding overfitting to the training data.

Overfitting occurs when a model is too complex and fits the noise in the training data, leading to poor generalization performance on new data. One way to avoid overfitting is to use regularization techniques that penalize the model for having too many parameters. Another way is to use a separate validation set to evaluate the performance of the model during training and select the best model based on the performance on the validation set.

The bias-variance trade-off is a fundamental concept in machine learning that relates to the trade-off between underfitting and overfitting. Bias refers to the error that is introduced by approximating a real-world problem with a simplified model. Variance refers to the error that is introduced by the model's sensitivity to small fluctuations in the training data.

A model with high bias and low variance is said to be underfitting, because it is too simple to capture the complexity of the problem. A model with low bias and high variance is said to be overfitting, because it is too complex and fits the noise in the training data. The optimal model is the one that achieves a balance between bias and variance, by fitting the underlying patterns in the data without overfitting to the noise.

The bias-variance trade-off can be visualized using a learning curve, which plots the training and validation error as a function of the number of training examples or model complexity. The learning curve can help to identify whether the model is underfitting, overfitting, or achieving an optimal trade-off between bias and variance.

In summary, model selection is the process of choosing the best model from a set of candidates, while the bias-variance trade-off is a fundamental concept that relates to the trade-off between underfitting and overfitting. Achieving a good trade-off between bias and variance is crucial for building accurate and robust machine learning models that generalize well to new data.

8. Describe the working of RBM.

A Restricted Boltzmann Machine (RBM) is a type of artificial neural network that is commonly used for unsupervised learning tasks, such as dimensionality reduction, feature learning, and collaborative filtering.

The RBM consists of two layers of nodes: a visible layer and a hidden layer. The visible layer represents the input data, while the hidden layer represents learned features. The nodes in each layer are connected to nodes in the other layer by a set of weights, which are learned during training.

The RBM is trained using a process called contrastive divergence, which is a type of stochastic gradient descent. During training, the RBM receives an input vector in the visible layer and updates the activations of the hidden layer nodes based on the input and the weights. The activations of the visible layer nodes are then updated based on the activations of the hidden layer nodes and the weights.

The training process aims to learn the weights that allow the RBM to capture the underlying structure of the data. Specifically, the RBM learns a set of features that are able to capture the correlations and patterns in the input data.

Once the RBM has been trained, it can be used for a variety of tasks, such as dimensionality reduction, feature extraction, and recommendation systems. For example, in a recommendation system, the RBM can be used to learn features that represent user preferences and item attributes, and then predict the likelihood that a user will be interested in a particular item.

In summary, the RBM is a type of artificial neural network that is used for unsupervised learning tasks. It consists of two layers of nodes, and is trained using a process called contrastive divergence to learn a set of features that capture the underlying structure of the data. The RBM can then be used for a variety of tasks, such as dimensionality reduction and recommendation systems.

9. Write different types of auto encoders and explain.

Auto encoders are a type of artificial neural network that can learn to encode and decode data in an unsupervised manner. There are several different types of auto encoders, each with their own unique characteristics and use cases.

Standard Auto encoder: This is the simplest type of auto encoder, consisting of an encoder network that maps the input data to a lower-dimensional representation, and a decoder network that maps the lower-dimensional representation back to the original input data. The goal of the standard auto encoder is to minimize the difference between the input and the output.

Convolutional Auto encoder: This type of auto encoder is designed for image data, and uses convolutional layers in the encoder and decoder networks to learn spatial features in the input data. The

convolutional auto encoder is often used for tasks such as image denoising, image compression, and image generation.

Denoising Auto encoder: This type of auto encoder is designed to remove noise from input data. During training, the denoising auto encoder is trained to reconstruct the original input from a noisy version of the input. The denoising auto encoder is often used for tasks such as speech denoising, image denoising, and signal denoising.

Variational Auto encoder: This type of auto encoder is designed to learn a probabilistic representation of the input data. The variational auto encoder consists of an encoder network that maps the input data to a distribution in latent space, and a decoder network that maps the latent distribution back to the original input data. The variational auto encoder is often used for tasks such as image generation, anomaly detection, and data compression.

Recurrent Auto encoder: This type of auto encoder is designed for sequential data, such as time series or natural language. The recurrent auto encoder consists of an encoder network that maps the input sequence to a lower-dimensional representation, and a decoder network that maps the lower-dimensional representation back to the original input sequence. The recurrent autoencoder is often used for tasks such as time series prediction, text generation, and speech recognition.

In summary, there are several different types of autoencoders, each with their own unique characteristics and use cases. Standard autoencoders are the simplest type, while convolutional, denoising, variational, and recurrent autoencoders are designed for specific types of data and tasks.