1. **What are the 3 components of the neural network?**
   * The three components of a neural network are:
     1. **Input layer**: The layer that receives the input data.
     2. **Hidden layer(s)**: Layers where computations and transformations of the input data occur. These layers have neurons that apply activation functions.
     3. **Output layer**: The final layer that produces the output of the neural network, typically the prediction or classification result.
2. **Why is Multilayer Perceptron (MLP) better than a single layer?**
   * A **Multilayer Perceptron (MLP)** is better than a single-layer perceptron because it can capture **non-linear relationships** in the data. Single-layer perceptrons can only solve linearly separable problems, while MLP with multiple layers (also called hidden layers) can model more complex relationships by using activation functions and learning from multiple levels of abstraction.
3. **What is the use of multi-layer neural networks?**
   * Multi-layer neural networks, or **deep neural networks**, are used to model complex relationships in data that cannot be captured by shallow networks. They are commonly used in tasks such as:
     1. Image and speech recognition
     2. Natural language processing
     3. Time series forecasting
     4. Autonomous systems (e.g., self-driving cars)
4. **What is the advantage of basis function over a multilayer feed-forward neural network?**
   * **Basis functions** (such as Radial Basis Functions or RBFs) are often used in simpler networks because they are computationally efficient. Compared to a multilayer feed-forward neural network (MLP), which requires training with many layers and numerous weights, networks using basis functions can often achieve better performance with fewer parameters and a more interpretable structure. Basis function networks tend to be better for interpolation tasks and are less prone to overfitting with small datasets.
5. **Why is MLP called a universal approximator?**
   * The **Multilayer Perceptron (MLP)** is called a **universal approximator** because it can approximate any continuous function to any degree of accuracy, given enough hidden neurons and layers. This is a result of the **Universal Approximation Theorem**, which states that an MLP with at least one hidden layer can approximate any continuous function on a compact input space.
6. **How many hidden layers are present in Multi-Layer Perceptron?**
   * The number of hidden layers in an MLP is **variable** and depends on the specific problem being solved. Typically, a basic MLP consists of **one or two hidden layers**, but in more complex models (like deep learning models), it can have many hidden layers (hence the term **deep neural networks**). There is no fixed number, as it is determined by the complexity of the data and the task.
7. **What is an Epoch in Machine Learning?**
   * An **Epoch** in machine learning refers to one complete pass through the entire training dataset by the model during training. After each epoch, the model updates its parameters (weights) to minimize the loss function. The training process often involves multiple epochs to allow the model to converge to an optimal solution.

**Possible Oral Questions (Excluding the Above):**

1. **What is the difference between supervised and unsupervised learning?**
   * Supervised learning uses labeled data to train the model, while unsupervised learning works with unlabeled data, aiming to find hidden patterns.
2. **Explain overfitting and underfitting in machine learning.**
   * Overfitting occurs when the model learns the noise in the training data, performing well on training data but poorly on new data. Underfitting occurs when the model is too simple to capture the patterns in the data, resulting in poor performance on both training and test data.
3. **What is the purpose of activation functions in neural networks?**
   * Activation functions introduce non-linearity into the model, allowing it to learn complex patterns in the data. Common activation functions include ReLU, Sigmoid, and Tanh.
4. **What is the difference between a perceptron and a neural network?**
   * A perceptron is a single-layer neural network used for binary classification. A neural network, however, can have multiple layers (hidden layers) and can solve more complex problems, including regression and multi-class classification.
5. **What is gradient descent?**
   * Gradient descent is an optimization algorithm used to minimize the loss function by adjusting the weights of the model in the direction of the steepest descent of the gradient of the loss function.
6. **Explain the difference between Batch Gradient Descent, Stochastic Gradient Descent (SGD), and Mini-Batch Gradient Descent.**
   * Batch Gradient Descent computes the gradient using the entire dataset, while Stochastic Gradient Descent (SGD) uses a single sample to compute the gradient. Mini-Batch Gradient Descent is a compromise, using a small batch of samples.
7. **What is backpropagation in neural networks?**
   * Backpropagation is the process used to update the weights of the network by calculating the gradient of the loss function with respect to each weight and propagating this gradient backward through the network.
8. **What is a convolutional neural network (CNN)?**
   * A **CNN** is a type of neural network primarily used for image processing, which uses convolutional layers to detect patterns such as edges and textures in images.
9. **What is the role of the loss function in machine learning?**
   * The loss function measures how well the model's predictions match the actual labels in the training data. It guides the training process by quantifying the error, which the model then tries to minimize.
10. **What is regularization in machine learning?**

* Regularization is a technique used to prevent overfitting by adding a penalty to the loss function based on the complexity of the model (e.g., L1 and L2 regularization).

\*\*import pandas as pd\*\*

\*\*from keras.models import Sequential\*\*

\*\*from keras.layers import Dense\*\*

# Imports the pandas library for data manipulation, and Keras libraries for building and training a neural network.

\*\*df = pd.read\_csv('5.csv')\*\*

# Reads the dataset from the CSV file named '5.csv' into a pandas DataFrame.

\*\*df.info()\*\*

# Displays a summary of the DataFrame, including the number of entries and data types.

\*\*df.columns\*\*

# Displays the column names of the DataFrame.

\*\*x = df.iloc[:, 0:-1].values\*\*

# Selects all columns except the last one as input features (x).

\*\*y = df.iloc[:, 8].values\*\*

# Selects the 9th column (index 8) as the target variable (y).

\*\*from ann\_visualizer.visualize import ann\_viz;\*\*

# Imports the ANN visualizer library to visualize the neural network architecture.

\*\*model = Sequential()\*\*

# Initializes a Sequential neural network model.

\*\*model.add(Dense(12, input\_dim=8, activation='relu'))\*\*

# Adds a fully connected (Dense) layer with 12 units, using 'relu' (rectified linear unit) activation, and an input dimension of 8.

\*\*model.add(Dense(8, activation='relu'))\*\*

# Adds another Dense layer with 8 units, using 'relu' activation.

\*\*model.add(Dense(1, activation='sigmoid'))\*\*

# Adds the final Dense layer with 1 unit, using 'sigmoid' activation for binary classification.

\*\*model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])\*\*

# Compiles the model with binary cross-entropy loss, Adam optimizer, and accuracy as the evaluation metric.

\*\*model.fit(x, y, epochs=100, batch\_size=10)\*\*

# Trains the model using the input features (x) and target variable (y) for 100 epochs with a batch size of 10.

\*\*\_, accuracy = model.evaluate(x, y)\*\*

# Evaluates the trained model on the data and stores the accuracy.

\*\*print('Accuracy: %2.f' % (accuracy \* 100))\*\*

# Prints the accuracy of the model as a percentage.

\*\*ann\_viz(model, title="My first neural network")\*\*

# Visualizes the model architecture using ANN Visualizer. This will generate a '.gv' file that can be opened with Graphviz.

# In case of an error, the following alternative approach is suggested:

# \*\*model.save('model.h5')\*\*

# \*\*import netron\*\*

# \*\*netron.start('model.h5')\*\*

# These steps save the model in H5 format and visualize it using the Netron tool.

\*\*import numpy as np\*\*

# Imports numpy for numerical operations, specifically for rounding the predictions.

\*\*predictions = np.round\_(model.predict(x))\*\*

# Makes predictions on the input data (x) and rounds the predicted values to either 0 or 1.

\*\*print(predictions)\*\*

# Prints the predictions made by the model.

\*\*model.summary()\*\*

# Displays a summary of the model's architecture, including the number of layers, parameters, and the output shapes.