**Assignment No: - 1**

**Feed-Forward Neural Network**

**Problem Statement:**

Implementing Feedforward neural networks in Python using Keras and TensorFlow.

**Objectives:**

* To understand the basic structure of feedforward neural networks.
* To learn how to preprocess data for training neural networks.
* To implement a feedforward neural network model using Keras and TensorFlow.
* To evaluate model performance using validation data.
* To visualize training loss and validation loss over epochs.

**S/W Packages and H/W apparatus used:**

* **Operating System:** Windows/Linux/MacOS
* **Kernel:** Python 3.x
* **Tools:** Jupyter Notebook, Anaconda, or Google Colab
* **Hardware:** CPU with minimum 4GB RAM; optional GPU for faster training
* **Libraries and packages used:** TensorFlow, Keras, NumPy, Pandas, Matplotlib, Scikit-Learn

**Theory:**

A **Feedforward Neural Network (FNN)** is a type of artificial neural network where connections between the nodes do not form cycles. The information moves in one direction only, from the input layer to the output layer, through hidden layers (if any).

**Structure:**

* **Input Layer:** Receives input features.
* **Hidden Layers:** One or more layers where computation occurs. Neurons in a layer are connected to all neurons in the subsequent layer.
* **Output Layer:** Produces the final output.
* **Activation Functions:** Commonly used functions include ReLU (Rectified Linear Unit), Sigmoid, and Softmax, introducing non-linearity into the network.

**Training via Backpropagation:**

* **Backpropagation** is used to update weights by minimizing the error using gradient descent and adjusting weights based on the error propagated backward.

**Methodology:**

**Step 1: Data Acquisition**

* Load the dataset (e.g., winequality-red.csv) using Pandas for analyzing the chemical properties of red wine and their quality.

**Step 2: Data Preparation**

* Split the dataset into training (75%) and validation (25%) sets.
* Normalize the feature values between 0 and 1 using min-max scaling to ensure faster convergence.

**Step 3: Model Architecture**

* **Sequential model** created using Keras:
  + Input layer with 64 units and ReLU activation function.
  + Hidden layer with 64 units and ReLU activation function.
  + Output layer with a single unit for regression tasks.

**Step 4: Model Compilation**

* Compile the model using **Adam** optimizer and **Mean Absolute Error (MAE)** as the loss function.

**Step 5: Model Training**

* Fit the model on the training data and validate on the validation data.
* Track loss metrics for each epoch.

**Step 6: Model Evaluation**

* Use the trained model to predict wine quality on the validation set and compare predictions with actual values.

**Step 7: Visualization of Loss**

* Plot the training and validation loss over epochs to visualize model performance and monitor overfitting.

**Advantages:**

* **Non-linearity Handling:** With activation functions like ReLU, FNNs capture complex relationships in data.
* **Flexibility:** Architecture can be easily adjusted for various applications.
* **Scalability:** Adding more hidden layers can improve learning from large datasets.
* **Parallel Processing:** Suitable for GPU-accelerated hardware, making training faster.

**Limitations:**

* **Data Requirements:** Requires a large amount of labeled data for effective learning.
* **Computational Cost:** Deep networks demand significant time and resources.
* **Overfitting Risk:** Can overfit if the model complexity exceeds the complexity of the data.
* **Hyperparameter Sensitivity:** The performance relies heavily on choosing the right hyperparameters (e.g., learning rate, layers).

**Working / Algorithm:**

# Step 1: Import Libraries

import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt

# Step 2: Load Dataset

data = pd.read\_csv('winequality-red.csv')

# Step 3: Data Preprocessing

X = data.drop(columns='quality') # Features

y = data['quality'] # Target

# Step 4: Normalize the data

scaler = MinMaxScaler()

X\_scaled = scaler.fit\_transform(X)

# Step 5: Split the dataset

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_scaled, y, test\_size=0.25, random\_state=42)

# Step 6: Build the Model

model = tf.keras.Sequential([

tf.keras.layers.Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)),

tf.keras.layers.Dense(64, activation='relu'),

tf.keras.layers.Dense(1) # Regression output

])

# Step 7: Compile the Model

model.compile(optimizer='adam', loss='mae')

# Step 8: Train the Model

history = model.fit(X\_train, y\_train, epochs=100, validation\_data=(X\_val, y\_val))

# Step 9: Visualize Loss

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Loss Over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

**Conclusion:**

Feedforward Neural Networks (FNNs) are powerful tools for both classification and regression tasks, effectively capturing complex relationships in data. When trained appropriately, they can provide accurate predictions. However, the choice of architecture, amount of data, and hyperparameter tuning play crucial roles in determining the success of the model.