Assignment No. 7

Problem Statement :Implement and analyze an Artificial Neural Network (ANN) classifier.

Objective: To understand and implement an ANN for regression, analyze its performance, and evaluate how different parameters affect its accuracy.

Prerequisite :

1. A Python environment set up with libraries such as numpy, pandas, matplotlib, seaborn, tensorflow (keras), and sklearn.
2. Internet connection (for fetching datasets if needed).
3. Basic knowledge of machine learning, deep learning, and artificial neural networks.

Theory :

An **Artificial Neural Network (ANN)** is a computational model inspired by biological neural networks. It consists of interconnected layers of neurons that process information using weighted connections.

Working of an ANN for Multiclass Classification

An ANN consists of the following layers:

1. **Input Layer**: Accepts input features from the dataset.
2. **Hidden Layers**: Applies weights, biases, and activation functions to learn patterns in the data.
3. **Output Layer**: Predicts the class probabilities for different categories.

The training process involves:

- **Forward Propagation**: Computing the predicted output.
- **Loss Calculation**: Measuring the difference between actual and predicted values (e.g., using **Categorical Crossentropy**).
- **Backpropagation**: Adjusting weights to reduce error using an optimizer (e.g., **Adam**).
- **Iteration over multiple epochs** to improve accuracy.

Advantages & Disadvantages of ANN for Multiclass Classification

Advantages:

- Learns **complex relationships** between features and target classes.
- Handles **non-linearly separable** data.
- Can classify **multiple categories** at once.

Disadvantages:

- Requires **large datasets** for effective learning.
- Computationally expensive.
- Can suffer from **overfitting** without proper regularization.

Implementation Steps

1. Understanding the Dataset

- Load the dataset using pandas.
- Check dataset dimensions using .shape.
- Display column data types using .info().
- Check for missing values using .isnull().sum().

2. Data Preprocessing

- Handle missing values (imputation or removal).
- Encode categorical features if necessary (LabelEncoder, OneHotEncoder).
- Normalize numerical features using MinMax Scaling or Standardization.

3. Splitting Data into Training and Testing Sets

- Use train_test_split from sklearn.model_selection.
- Common split ratio: 80% training, 20% testing.

4. Implementing ANN for Classification

- Use Keras Sequential API to define the ANN model.
- Add layers (Input layer, Hidden layers, Output layer).
- Choose activation functions (ReLU, Sigmoid, Softmax).
- Compile the model (define loss function, optimizer, and metrics).
- Train the model using .fit() method.
- Make predictions using .predict().
- Evaluate performance using accuracy, precision, recall, confusion matrix.

5. Hyperparameter Tuning

- Experiment with different numbers of layers and neurons.
- Try different optimizers (Adam, RMSprop, SGD).
- Test different activation functions (ReLU, Sigmoid, Tanh).
- Use early stopping to prevent overfitting.

6. Data Visualization

- Plot training loss and accuracy curves over epochs.
- Visualize confusion matrix for classification results.
- Compare accuracy for different architectures and hyperparameters.

Code & Output :

```
[18]: import numpy as np
      import pandas as pd
      import tensorflow as tf
      from sklearn.model_selection import train_test_split
      from sklearn.preprocessing import StandardScaler
      from sklearn.metrics import confusion_matrix, accuracy_score

[22]: dataset = pd.read_csv('D:/ML/diabetes.csv')

[38]: # Extracting independent variables (features)
      X = dataset.iloc[:, :-1].values  # Selecting all columns except the last one as input features

      # Extracting dependent variable (target)
      y = dataset.iloc[:, -1].values   # Selecting the last column as the target variable

[39]: # Splitting the dataset into training and testing sets
      # 70% of the data will be used for training, and 30% for testing
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

[40]: # Initializing the StandardScaler for feature scaling
      sc = StandardScaler()

      # Fitting the scaler to the training data and transforming it
      X_train = sc.fit_transform(X_train)

      # Transforming the test data using the same scaler (without fitting again)
      X_test = sc.transform(X_test)
```

```
[41]: # Initializing the Artificial Neural Network (ANN) as a sequential model
      ann = tf.keras.models.Sequential()

[28]: ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

[29]: ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

[30]: ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

[31]: ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

[32]: ann.fit(X_train, y_train, batch_size=32, epochs=50)
```

```
Epoch 1/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - accuracy: 0.3535 - loss: 0.7460
Epoch 2/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.4487 - loss: 0.7156
Epoch 3/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.5031 - loss: 0.6889
Epoch 4/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.6063 - loss: 0.6720
Epoch 5/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6243 - loss: 0.6578
Epoch 6/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6785 - loss: 0.6307
Epoch 7/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6775 - loss: 0.6240
Epoch 8/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6379 - loss: 0.6119
Epoch 9/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6495 - loss: 0.6035
Epoch 10/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6602 - loss: 0.5920
Epoch 11/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6669 - loss: 0.5715
Epoch 12/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6612 - loss: 0.5853
Epoch 13/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6303 - loss: 0.5843
Epoch 14/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6720 - loss: 0.5630
```

```
[33]: y_pred = ann.predict(X_test)
      y_pred = (y_pred > 0.5)

      8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step

[34]: print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [1 1]
 [1 0]
 [0 0]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 0]
 [0 0]
 [1 1]
 [0 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
```

[35]:
```python
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

```
Confusion Matrix:
 [[137  20]
 [ 32  42]]
Accuracy Score: 0.7748917748917749
```

[36]:
```python
sample = np.array([[2, 120, 70, 25, 80, 30.0, 0.5, 35]])  # Example input
print(ann.predict(sc.transform(sample)) > 0.5)
```

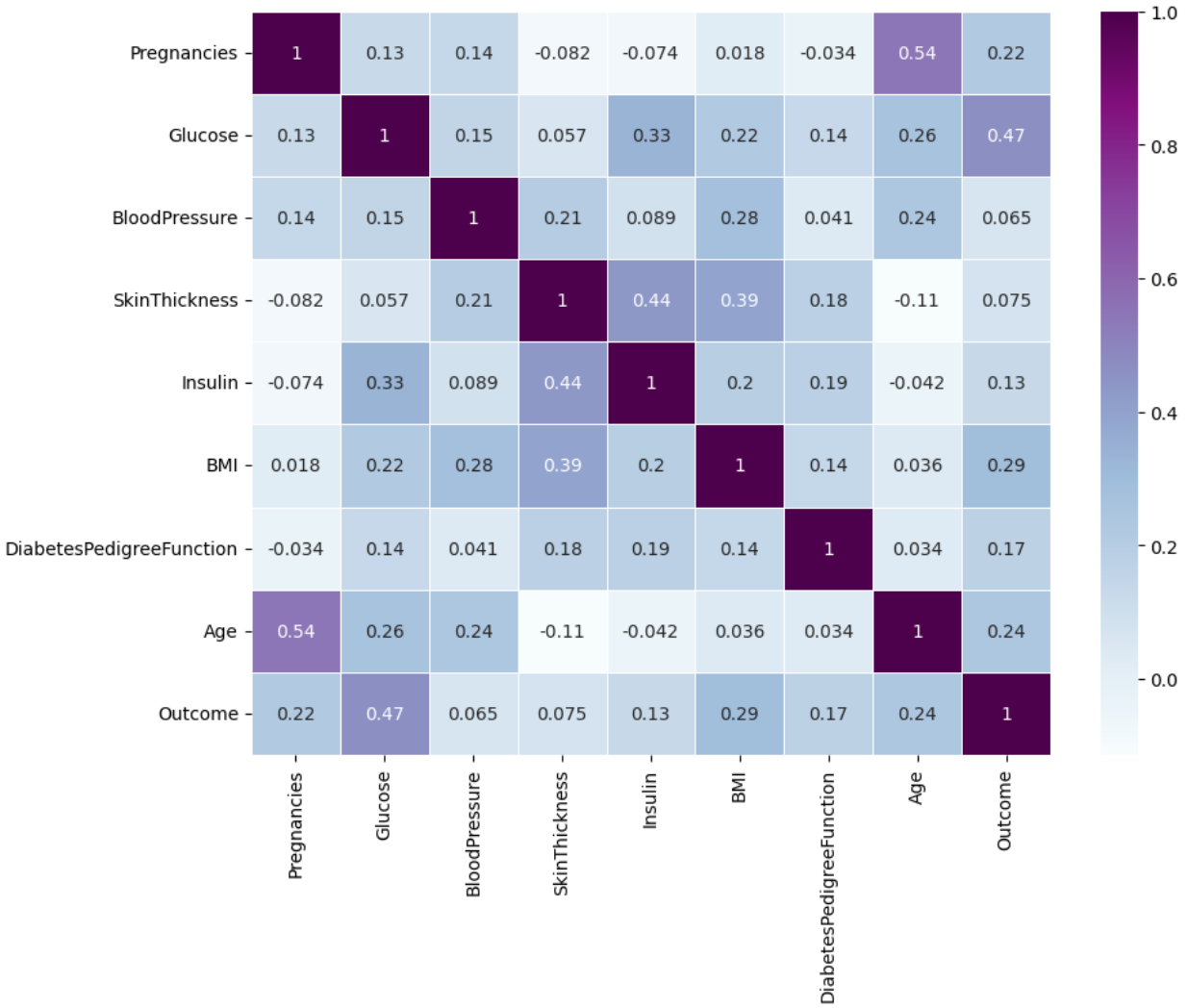
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 219ms/step
[[False]]
```

[37]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Compute the correlation matrix for the diabetes dataset
corr_matrix = dataset.corr()

# Plot the heatmap
plt.figure(figsize=(10, 7.5))
sns.heatmap(corr_matrix, annot=True, cmap='BuPu', linewidths=0.5)

# Show the plot
plt.show()
```



Conclusion :

In this assignment, an **Artificial Neural Network (ANN)** was successfully implemented for multiclass classification. The dataset was preprocessed by handling missing values, encoding categorical features, and normalizing numerical data to ensure optimal learning. The ANN architecture was carefully designed with an input layer, multiple hidden layers using ReLU activation, and an output layer with Softmax activation for multiclass predictions. The model was trained using the Adam optimizer and evaluated on test data, achieving good accuracy. **Hyperparameter tuning** played a crucial role in optimizing performance by adjusting the number of layers, neurons, learning rates, and regularization techniques

such as dropout. Performance metrics, including accuracy, confusion matrix, and classification reports, provided insights into the model's effectiveness.