Assignment No. 7

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

Objective: To understand and implement an ANN for classification, 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 :

Artificial Neural Networks are computational models inspired by the structure and functioning of biological neural networks—such as those in the human brain. They consist of interconnected processing units called neurons, which work together to process data and learn patterns. ANNs have become a cornerstone of modern machine learning, especially in tasks where data relationships are complex and non-linear.

1. Biological Inspiration and Computational Analogy

Biological Neurons vs. Artificial Neurons

- **Biological Inspiration:**
  Biological neurons receive electrical signals through dendrites, process them in the cell body, and transmit output through the axon. They are interconnected in complex networks that enable learning, memory, and decision making.
- **Artificial Neurons:**
  An artificial neuron mimics this behavior by taking numerical inputs, applying a weight to each input, summing them up along with a bias term, and then passing the result through an activation function. This activation determines whether the neuron "fires" (i.e., transmits a signal) to subsequent layers.

2. Architecture of an ANN

An ANN is typically organized into layers:

- **Input Layer:**
  This is the first layer that receives the raw input features from the dataset. Each neuron in this layer represents a feature.
- **Hidden Layers:**
  One or more layers that transform the input into something the output layer can use. Each hidden layer applies weights, biases, and non-linear activation functions (like ReLU, Sigmoid, or Tanh) to capture complex patterns.
- **Output Layer:**
  For multiclass classification, the output layer usually consists of one neuron per class, and a softmax activation function is commonly used to produce probabilities that sum to one.

Interconnectedness and Weights

- **Weighted Connections:**
  Every connection between neurons is associated with a weight. These weights are the parameters learned during training.
- **Bias Terms:**
  Each neuron (except in the input layer) also has a bias term, which allows the activation function to be shifted left or right.

3. Working of an ANN in Multiclass Classification

3.1 Forward Propagation

- **Data Flow:**
  In forward propagation, the input data is passed layer by layer. Each neuron computes a weighted sum of its inputs, adds a bias, and applies an activation function.
- **Output Computation:**
  For multiclass tasks, the final output layer uses the softmax activation function, which converts raw scores (logits) into a probability distribution over multiple classes.

3.2 Loss Calculation

- **Purpose of Loss:**
  The loss function quantifies the error between the predicted outputs and the actual class labels.
- **Categorical Crossentropy:**
  In multiclass classification, categorical crossentropy is typically used. It measures the divergence between the predicted probability distribution and the true distribution (often one-hot encoded).

3.3 Backpropagation and Optimization

- **Backpropagation:**
  After computing the loss, the network uses backpropagation to calculate the gradient of the loss function with respect to each weight. This gradient information tells us how to adjust the weights to reduce the error.
- **Optimizers:**
  Algorithms like Adam, RMSprop, or SGD use these gradients to update the weights. Adam is particularly popular because it adapts the learning rate during training and converges efficiently.

3.4 Iterative Learning and Epochs

- **Epochs:**
  The entire training dataset is passed through the network multiple times (epochs) to progressively refine the weights.
- **Batch Processing:**
  Training is often done in mini-batches (small subsets of the dataset) to strike a balance between computational efficiency and stable convergence.

4. Advantages and Disadvantages of ANNs for Multiclass Classification

Advantages

- **Learning Complex Relationships:**
  ANNs can model non-linear and complex relationships between features and classes.
- **Flexibility:**
  They can be applied to a wide range of problems, including image recognition, text classification, and more.
- **Simultaneous Multi-Class Predictions:**
  The softmax function in the output layer allows the network to predict probabilities for multiple classes simultaneously.

Disadvantages

- **Data Requirements:**
  Effective training often requires large datasets to avoid underfitting and ensure that the model generalizes well.
- **Computational Expense:**
  Training deep networks can be computationally intensive and time-consuming, particularly with large architectures.
- **Overfitting Risks:**
  Without proper regularization (like dropout or early stopping), ANNs can overfit, meaning they perform well on training data but poorly on unseen data.

Code & Output :

[22]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
```

[14]:
```
file_path = "/Users/pranavashokdivekar/this_mac/Machine Learning/diabetes.csv"
df = pd.read_csv(file_path)
```

[12]:
```
df
```

[12]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

[15]:
```
print(df.isnull().sum()) #Check for missing values
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[16]:
```
# Step 4: Define Features (X) and Target (Y)
X = df.drop(columns=["Outcome"])  # Input features
y = df["Outcome"]  # Target variable (0 or 1)
```

[23]:
```
#Step 5: Split dataset into Training and Testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[24]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[25]:
```
model = keras.Sequential([
    keras.layers.Dense(16, activation="relu", input_shape=(X_train.shape[1],)),  # Input layer
    keras.layers.Dense(8, activation="relu"),  # Hidden layer
    keras.layers.Dense(1, activation="sigmoid")  # Output layer (Sigmoid for binary classification)
])
```

```
/Users/pranavashokdivekar/this_mac/venv/lib/python3.11/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[26]:
```
# Step 8: Compile the Model
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
```

[27]:
```
# Step 9: Train the Model
model.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5837 - loss: 0.6826 - val_accuracy: 0.5779 - val_loss: 0.6448
Epoch 2/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6544 - loss: 0.6235 - val_accuracy: 0.6494 - val_loss: 0.6016
Epoch 3/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6760 - loss: 0.5920 - val_accuracy: 0.6948 - val_loss: 0.5701
Epoch 4/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7162 - loss: 0.5656 - val_accuracy: 0.7403 - val_loss: 0.5479
Epoch 5/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7110 - loss: 0.5342 - val_accuracy: 0.7532 - val_loss: 0.5291
Epoch 6/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7027 - loss: 0.5287 - val_accuracy: 0.7532 - val_loss: 0.5160
Epoch 7/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7808 - loss: 0.4711 - val_accuracy: 0.7532 - val_loss: 0.5053
Epoch 8/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7380 - loss: 0.4911 - val_accuracy: 0.7662 - val_loss: 0.4983
Epoch 9/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7797 - loss: 0.4788 - val_accuracy: 0.7727 - val_loss: 0.4934
Epoch 10/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7681 - loss: 0.4651 - val_accuracy: 0.7727 - val_loss: 0.4889
Epoch 11/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7675 - loss: 0.4713 - val_accuracy: 0.7597 - val_loss: 0.4855
Epoch 12/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7738 - loss: 0.4841 - val_accuracy: 0.7597 - val_loss: 0.4863
Epoch 13/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7895 - loss: 0.4718 - val_accuracy: 0.7662 - val_loss: 0.4859
Epoch 14/50
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7781 - loss: 0.4653 - val_accuracy: 0.7662 - val_loss: 0.4877
```

[28]:
```
# Step 10: Evaluate the Model
y_pred_prob = model.predict(X_test)  # Get probabilities
y_pred = (y_pred_prob > 0.5).astype(int)  # Convert to binary labels
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
```

[29]:
```
# Step 11: Print Performance Metrics
print("\nAccuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy Score: 0.7662337662337663

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.82      0.82        99
           1       0.67      0.67      0.67        55

    accuracy                           0.77       154
   macro avg       0.75      0.75      0.75       154
weighted avg       0.77      0.77      0.77       154
```

[ ]:

Github: https://github.com/dnyaneshwardhere/ML

Conclusion :

In this assignment, we used an ANN with ReLU activation in the input and hidden layers and a Sigmoid output layer for binary classification. The performance metrics, as shown in the confusion matrix and classification report, indicate about 76% accuracy, suggesting moderate success. While the model is capturing important patterns, further tuning such as adjusting hyperparameters, experimenting with additional layers, or applying stronger regularization could improve its predictive performance.