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 :

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 ANN Classifier

1. **Input Layer:** Accepts features from the dataset.
2. **Hidden Layers:** Performs computations using weighted sums, activation functions, and backpropagation for learning.
3. **Output Layer:** Produces classification results (e.g., probabilities for different classes).
4. **Training Process:**
    - Forward propagation: Computes the predicted output.
    - Loss calculation: Measures error between predicted and actual values.
    - Backpropagation: Adjusts weights using an optimizer (e.g., SGD, Adam).
    - Repetition: Trains for multiple epochs to improve accuracy.

Choosing the Right Parameters

- Number of Layers & Neurons: More layers capture complex patterns but increase computation.
- Activation Function: Common choices include ReLU, Sigmoid, Softmax.
- Optimizer: Adam, SGD, RMSprop for weight updates.

- Loss Function: Categorical Crossentropy (for multi-class) or Binary Crossentropy (for binary classification).

Advantages of ANN

Handles complex patterns in data.
Learns non-linear relationships.
Can improve accuracy with sufficient training.

Disadvantages of ANN

Computationally expensive (requires more processing power).
Sensitive to overfitting (requires regularization).
Requires large amounts of labeled data for effective training.

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 :

[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_sh
ape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model ins
tead.
  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/Pranav-Divekar/Machine-learning-**

Conclusion:

The ANN classifier achieved 76.62% accuracy, meaning it correctly predicted 77% of cases.

- Class 0: Good performance (82% precision & recall).
- Class 1: Weaker performance (67% precision & recall), likely due to class imbalance.
- The model works well but struggles with the minority class.
- Adjusting data balance or tuning parameters can improve results.