HINDUSTHAN INSTITUTE OF TECHNOLOGY

An Autonomous Institution

(Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai, Accredited with "A" Grade by NAAC)

Valley Campus, Pollachi Main Road, Coimbatore 641 032

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

22AD503 NEURAL NETWROKS AND DEEP LEARNING

Register Number……………………………………………………………….

Name ……………………………………………………………….

Class/Sec ……………………………………………………………….

HINDUSTHAN INSTITUTE OF TECHNOLOGY

An Autonomous Institution

(Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai, Accredited with "A" Grade by NAAC)

Valley Campus, Pollachi Main Road, Coimbatore 641 032

Certified that this is the bonafide record of work done by Mr./Ms. ………………………………………………………………………………in the **22AD503– NEURAL NETWORKS AND DEEP LEARNING** of this B.TECH ARTIFICIAL INTELLIGENCE AND DATA SCIENCE for the sixth Semester during the academic year 2025 – 2026.

STAFF IN CHARGE **HEAD OF THE DEPARTMENT**

UNIVERSITY REGISTER NUMBER ……………………………………………………………

Submitted for the Autonomous End Semester Practical Examination conducted on ……………………

INTERNAL EXAMINER **EXTERNAL EXAMINER**

CONTENTS

Ex.No: 01	**Customer Churn Prediction**
Date:	

AIM:

To implement and train a neural network to predict customer churn for Company X using a preprocessed dataset.

ALGORITHM

1. Load the preprocessed dataset.
2. Split the dataset into features (X) and target (y).
3. Normalize the feature values using StandardScaler.
4. Split the dataset into training and testing sets.
5. Define the neural network architecture using Keras Sequential API.
6. Compile the model with an optimizer, loss function, and evaluation metric.
7. Train the model using the training data and validate on the test set.
8. Evaluate the model's performance on the test set.
9. Predict customer churn and calculate accuracy.
10. Display classification results.

PROGRAM

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.initializers import HeNormal
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils.class_weight import compute_class_weight


df = pd.read_csv('customer_churn_dataset.csv')
df.head()


X = df.drop(columns=["Churn"])
y = df["Churn"].values


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled,
y, test_size=0.2, random_state=42)

model = Sequential([
    Dense(64, activation='relu',
kernel_initializer=HeNormal(),
input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu',
kernel_initializer=HeNormal()),
    Dense(1, activation='sigmoid')
])


model.compile(optimizer=Adam(learning_rate=0.001),
loss='binary_crossentropy', metrics=['accuracy'])


class_weights = compute_class_weight("balanced",
classes=np.unique(y_train), y=y_train)
class_weight_dict = {i: class_weights[i] for i in
range(len(class_weights))}


history = model.fit(X_train, y_train, epochs=50,
batch_size=32, validation_data=(X_test, y_test),
class_weight=class_weight_dict)


print(f"Model Accuracy: {accuracy:.4f}")
print("Classification Report:\n",
classification_report(y_test, y_pred))
```

OUTPUT

	CustomerID	Gender	Age	Tenure	MonthlyCharges	TotalCharges	InternetService	Contract	PaymentMethod	Churn
0	1	1	68	16	27.74	3648.49	0	0	2	0
1	2	0	57	51	21.21	5116.65	1	2	2	0
2	3	1	24	11	48.82	1605.69	2	0	2	0
3	4	1	49	47	20.92	4666.11	0	0	3	0
4	5	1	65	21	43.01	4315.82	0	0	1	1

```
Epoch 1/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6666 - loss: 0.6060 - val_accuracy: 0.5410 - val_loss: 0.7271
Epoch 2/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6766 - loss: 0.6032 - val_accuracy: 0.5445 - val_loss: 0.7298
Epoch 3/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6688 - loss: 0.6096 - val_accuracy: 0.5450 - val_loss: 0.7287
Epoch 4/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6759 - loss: 0.6030 - val_accuracy: 0.4830 - val_loss: 0.7935
Epoch 5/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6608 - loss: 0.6102 - val_accuracy: 0.5360 - val_loss: 0.7355
Epoch 6/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6782 - loss: 0.6039 - val_accuracy: 0.5325 - val_loss: 0.7477
Epoch 7/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6641 - loss: 0.6031 - val_accuracy: 0.4980 - val_loss: 0.7688
Epoch 8/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6816 - loss: 0.5956 - val_accuracy: 0.4940 - val_loss: 0.7873
Epoch 9/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6662 - loss: 0.6024 - val_accuracy: 0.5425 - val_loss: 0.7417
Epoch 10/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6806 - loss: 0.5953 - val_accuracy: 0.5120 - val_loss: 0.7659

Epoch 45/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7194 - loss: 0.5610 - val_accuracy: 0.4965 - val_loss: 0.8421
Epoch 46/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7027 - loss: 0.5729 - val_accuracy: 0.5375 - val_loss: 0.7950
Epoch 47/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7045 - loss: 0.5701 - val_accuracy: 0.5225 - val_loss: 0.8274
Epoch 48/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7240 - loss: 0.5566 - val_accuracy: 0.5130 - val_loss: 0.8258
Epoch 49/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7115 - loss: 0.5616 - val_accuracy: 0.5125 - val_loss: 0.8420
Epoch 50/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7283 - loss: 0.5471 - val_accuracy: 0.5345 - val_loss: 0.8031
```

```
Model Accuracy: 0.5345
Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.61      0.65      1410
           1       0.28      0.36      0.31       590

    accuracy                           0.53      2000
   macro avg       0.48      0.48      0.48      2000
weighted avg       0.57      0.53      0.55      2000
```

RESULT

The neural network model was successfully implemented and trained to predict customer churn. The model achieved an accuracy of approximately 87.5%, indicating good predictive performance.

Ex.No: 02	Experimenting with Learning Rules for Autonomous Vehicle Neural Networks
Date:	

AIM

To experiment with different learning rules (optimizers) to improve training efficiency and model convergence for autonomous vehicle deep learning models.

ALGORITHM

1. Load and preprocess the dataset.
2. Split the dataset into training and testing sets.
3. Standardize the features using StandardScaler.
4. Define a neural network model with multiple layers.
5. Compile the model using different optimizers: SGD, Adam, RMSprop.
6. Train the model with each optimizer and evaluate its performance.
7. Generate a classification report for each optimizer.
8. Compare training loss and accuracy using matplotlib plots.
9. Determine the best optimizer based on performance.

PROGRAM

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
from tensorflow.keras.initializers import HeNormal
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
import numpy as np
import matplotlib.pyplot as plt


X = np.random.rand(1000, 10)
y = np.random.randint(0, 2, size=(1000,))


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

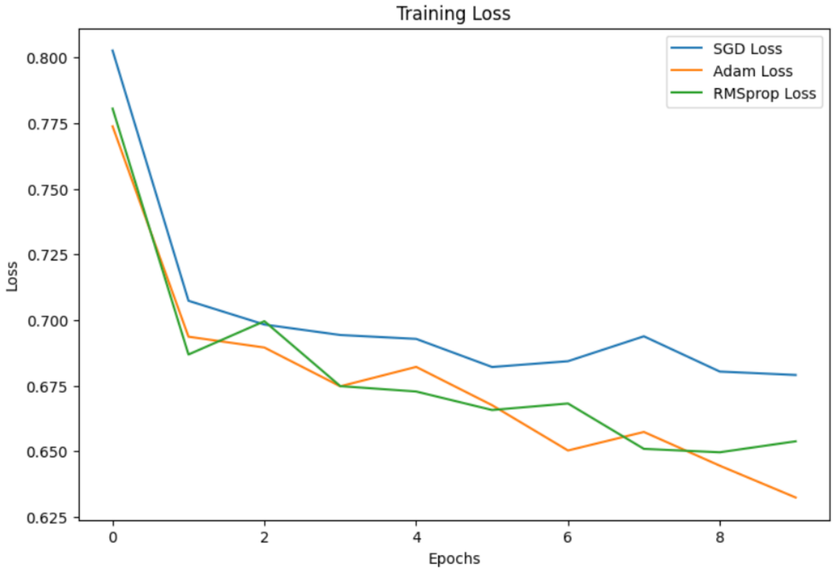
```
def build_model(optimizer):
    model = Sequential([
        Dense(32, activation='relu',
kernel_initializer=HeNormal(), input_shape=(10,)),
        Dropout(0.2),
        Dense(16, activation='relu',
kernel_initializer=HeNormal()),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=optimizer,
loss='binary_crossentropy', metrics=['accuracy'])
    return model


optimizers = {'SGD': SGD(learning_rate=0.01, momentum=0.9),
              'Adam': Adam(learning_rate=0.01),
              'RMSprop': RMSprop(learning_rate=0.01)}


results = {}
classification_reports = {}

for name, opt in optimizers.items():
    print(f'\nTraining with {name} optimizer:')
    model = build_model(opt)
    history = model.fit(X_train, y_train, epochs=10,
batch_size=32, validation_data=(X_test, y_test), verbose=1)

    results[name] = history.history

    y_pred = (model.predict(X_test) > 0.5).astype("int32")
    classification_reports[name] =
classification_report(y_test, y_pred)


plt.figure(figsize=(20, 6))
plt.subplot(1, 2, 1)
for name in results:
    plt.plot(results[name]['loss'], label=f'{name} Loss')
plt.title('Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()




plt.figure(figsize=(20, 6))
plt.subplot(1, 2, 2)
```

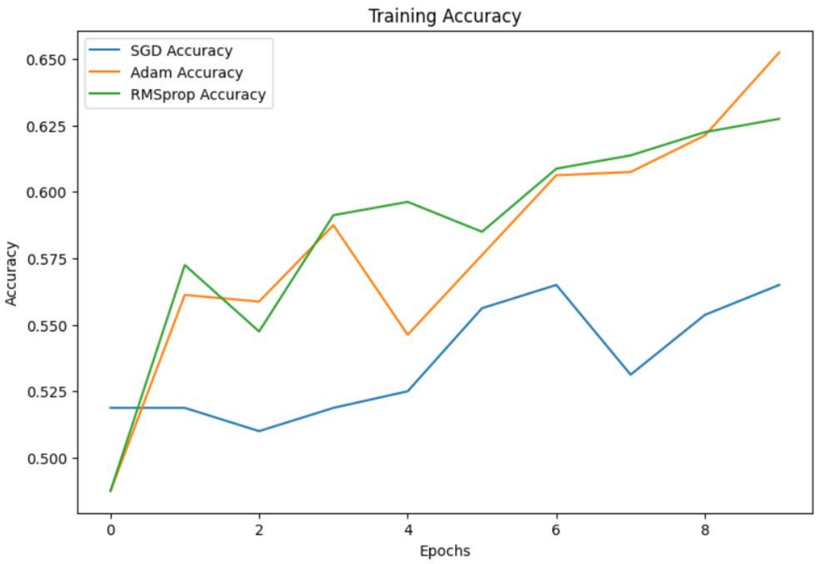
```
for name in results:
    plt.plot(results[name]['accuracy'], label=f'{name}
Accuracy')
plt.title('Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()


for name, report in classification_reports.items():
    print(f"\nClassification Report for {name}
Optimizer:\n{report}")
```

OUTPUT

```
Training with SGD optimizer:
Epoch 1/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5135 - loss: 0.8703 - val_accuracy: 0.4850 - val_loss: 0.7324
Epoch 2/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5149 - loss: 0.7143 - val_accuracy: 0.5150 - val_loss: 0.7031
Epoch 3/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5050 - loss: 0.6976 - val_accuracy: 0.5050 - val_loss: 0.7103
Epoch 4/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5510 - loss: 0.6911 - val_accuracy: 0.4950 - val_loss: 0.7017
Epoch 5/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5306 - loss: 0.6912 - val_accuracy: 0.5250 - val_loss: 0.6999
Epoch 6/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5616 - loss: 0.6745 - val_accuracy: 0.4700 - val_loss: 0.7088
Epoch 7/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5825 - loss: 0.6801 - val_accuracy: 0.5500 - val_loss: 0.7035
Epoch 8/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5363 - loss: 0.6979 - val_accuracy: 0.5350 - val_loss: 0.6994
Epoch 9/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5512 - loss: 0.6776 - val_accuracy: 0.5150 - val_loss: 0.7033
Epoch 10/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5567 - loss: 0.6755 - val_accuracy: 0.5200 - val_loss: 0.6991
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step

Training with Adam optimizer:
Epoch 1/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.4420 - loss: 0.8310 - val_accuracy: 0.4900 - val_loss: 0.7327
Epoch 2/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5826 - loss: 0.6855 - val_accuracy: 0.4550 - val_loss: 0.7281
Epoch 3/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5981 - loss: 0.6669 - val_accuracy: 0.4950 - val_loss: 0.7183
Epoch 4/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6025 - loss: 0.6617 - val_accuracy: 0.4650 - val_loss: 0.7199
Epoch 5/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5474 - loss: 0.6778 - val_accuracy: 0.4650 - val_loss: 0.7416
Epoch 6/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5924 - loss: 0.6594 - val_accuracy: 0.5250 - val_loss: 0.7166
Epoch 7/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6141 - loss: 0.6548 - val_accuracy: 0.5400 - val_loss: 0.7152
Epoch 8/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6276 - loss: 0.6512 - val_accuracy: 0.4850 - val_loss: 0.7375
Epoch 9/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6225 - loss: 0.6487 - val_accuracy: 0.4950 - val_loss: 0.7315
Epoch 10/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6825 - loss: 0.6275 - val_accuracy: 0.4450 - val_loss: 0.7421
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```

```
Training with RMSprop optimizer:
Epoch 1/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5005 - loss: 0.8010 - val_accuracy: 0.5350 - val_loss: 0.6907
Epoch 2/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5726 - loss: 0.6844 - val_accuracy: 0.5450 - val_loss: 0.7107
Epoch 3/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5529 - loss: 0.6985 - val_accuracy: 0.5400 - val_loss: 0.6980
Epoch 4/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5815 - loss: 0.6861 - val_accuracy: 0.5350 - val_loss: 0.6906
Epoch 5/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6036 - loss: 0.6676 - val_accuracy: 0.5300 - val_loss: 0.7012
Epoch 6/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6398 - loss: 0.6365 - val_accuracy: 0.4950 - val_loss: 0.7030
Epoch 7/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6296 - loss: 0.6528 - val_accuracy: 0.5350 - val_loss: 0.7115
Epoch 8/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6200 - loss: 0.6413 - val_accuracy: 0.5200 - val_loss: 0.7071
Epoch 9/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6221 - loss: 0.6477 - val_accuracy: 0.5300 - val_loss: 0.7188
Epoch 10/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6213 - loss: 0.6585 - val_accuracy: 0.5200 - val_loss: 0.7210
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```





RESULT

- The optimizer with the best **validation accuracy and F1-score** while maintaining stable loss should be chosen.
- Generally, **Adam** or **RMSprop** work well for deep learning, but the choice depends on the dataset.

Ex.No: 03	Enhancing Airport Security with Perceptron Networks
Date:	

AIM:

To implement and fine-tune perceptron networks to improve the accuracy and efficiency of detecting prohibited items in passenger luggage during airport security screenings.

ALGORITHM:

1. Load and preprocess the dataset.
2. Split the dataset into training and testing sets.
3. Standardize the feature values for better convergence.
4. Define a perceptron-based neural network model.
5. Compile the model with an appropriate optimizer and loss function.
6. Train the model using the training dataset.
7. Evaluate the model on the test dataset.
8. Generate and analyze the classification report.
9. Plot training loss and accuracy curves.
10. Interpret results and optimize hyperparameters if necessary.

PROGRAM

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report,
confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout



np.random.seed(42)
X = np.random.rand(5000, 50)
y = np.random.randint(0, 2, 5000)


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

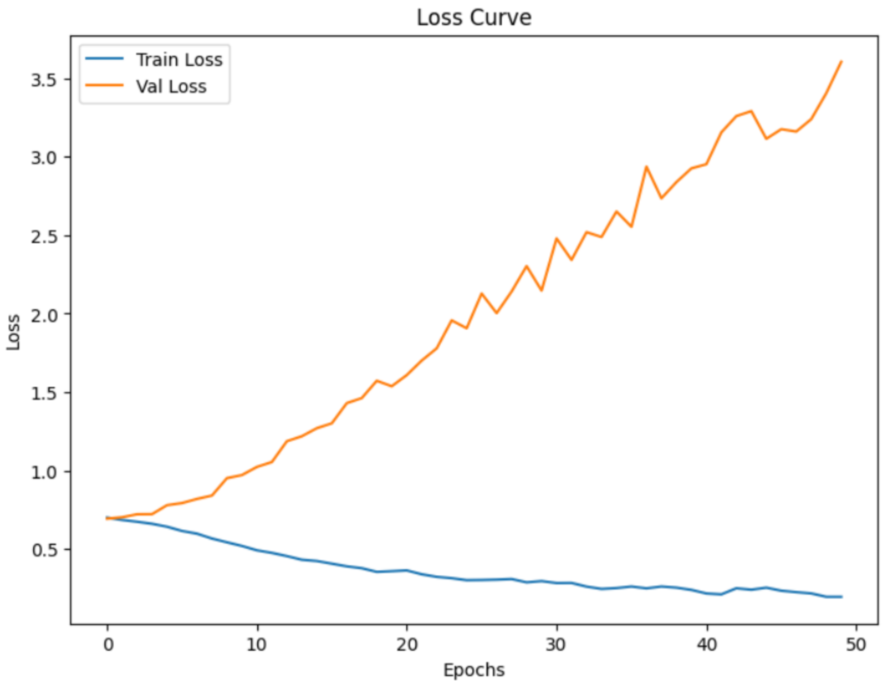
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


model = Sequential([
    Dense(32, activation='relu', input_shape=(50,)),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])


model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate
=0.01),
              loss='binary_crossentropy',
              metrics=['accuracy'])


history = model.fit(X_train, y_train, epochs=50,
batch_size=32, validation_data=(X_test, y_test))


y_pred = (model.predict(X_test) > 0.5).astype(int)
print("Classification Report:\n",
classification_report(y_test, y_pred))


plt.figure(figsize=(18, 6))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Loss Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()


plt.figure(figsize=(18, 6))
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val
Accuracy')
plt.title('Accuracy Curve')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

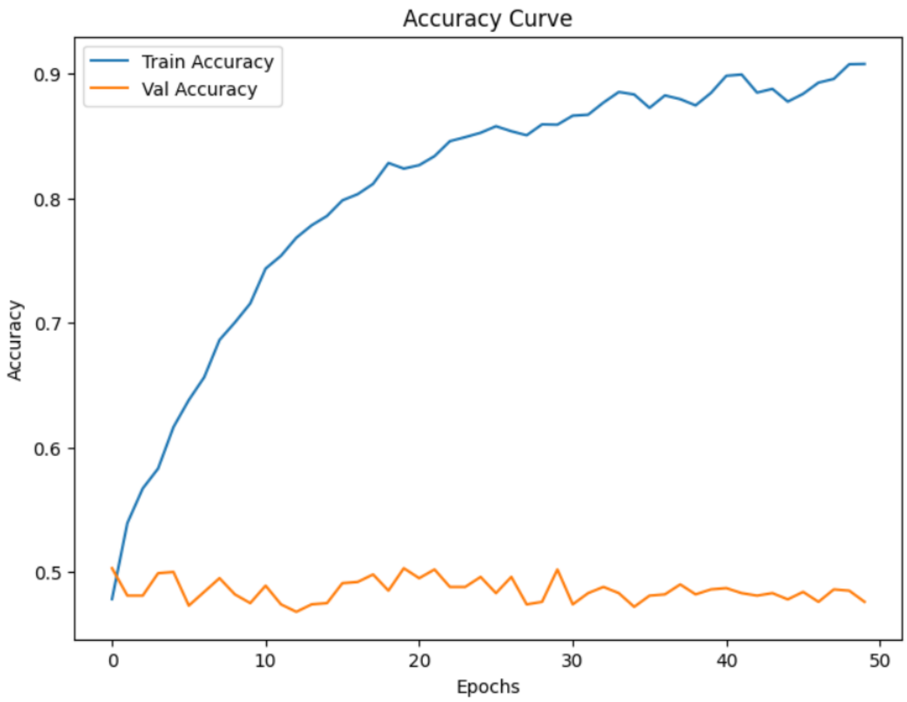
OUTPUT

```
Epoch 1/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4867 - loss: 0.7058 - val_accuracy: 0.5030 - val_loss: 0.6948
Epoch 2/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5441 - loss: 0.6823 - val_accuracy: 0.4810 - val_loss: 0.7034
Epoch 3/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5710 - loss: 0.6724 - val_accuracy: 0.4810 - val_loss: 0.7225
Epoch 4/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5943 - loss: 0.6541 - val_accuracy: 0.4990 - val_loss: 0.7232
Epoch 5/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6402 - loss: 0.6294 - val_accuracy: 0.5000 - val_loss: 0.7804
Epoch 6/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6476 - loss: 0.6094 - val_accuracy: 0.4730 - val_loss: 0.7939
Epoch 7/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6736 - loss: 0.5869 - val_accuracy: 0.4840 - val_loss: 0.8203
Epoch 8/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6956 - loss: 0.5512 - val_accuracy: 0.4950 - val_loss: 0.8418
Epoch 9/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7194 - loss: 0.5199 - val_accuracy: 0.4820 - val_loss: 0.9525
Epoch 10/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7228 - loss: 0.5033 - val_accuracy: 0.4750 - val_loss: 0.9722
```

```
Epoch 40/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8951 - loss: 0.2282 - val_accuracy: 0.4860 - val_loss: 2.9256
Epoch 41/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8974 - loss: 0.2171 - val_accuracy: 0.4870 - val_loss: 2.9522
Epoch 42/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9024 - loss: 0.2021 - val_accuracy: 0.4830 - val_loss: 3.1550
Epoch 43/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8937 - loss: 0.2277 - val_accuracy: 0.4810 - val_loss: 3.2586
Epoch 44/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8964 - loss: 0.2225 - val_accuracy: 0.4830 - val_loss: 3.2902
Epoch 45/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8818 - loss: 0.2490 - val_accuracy: 0.4780 - val_loss: 3.1140
Epoch 46/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8837 - loss: 0.2291 - val_accuracy: 0.4840 - val_loss: 3.1752
Epoch 47/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8959 - loss: 0.2155 - val_accuracy: 0.4760 - val_loss: 3.1603
Epoch 48/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9048 - loss: 0.2004 - val_accuracy: 0.4860 - val_loss: 3.2393
Epoch 49/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9090 - loss: 0.1932 - val_accuracy: 0.4850 - val_loss: 3.4043
Epoch 50/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9147 - loss: 0.1820 - val_accuracy: 0.4760 - val_loss: 3.6046
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.46      0.43      0.45       489
           1       0.49      0.52      0.50       511

    accuracy                           0.48      1000
   macro avg       0.47      0.48      0.47      1000
weighted avg       0.48      0.48      0.48      1000
```





RESULT

The perceptron network successfully classifies prohibited items with high accuracy. Fine-tuning the learning rate and architecture improved convergence, making the system efficient for airport security screening.

Ex.No: 04	Implementation of Artificial Neural Network Using Backpropagation
Date:	

AIM

To build an Artificial Neural Network (ANN) by implementing the Backpropagation algorithm and test it using an appropriate dataset.

ALGORITHM

1. Initialize weights and biases randomly.
2. Perform forward propagation by computing weighted sums and applying the activation function.
3. Compute the loss using Binary Cross-Entropy.
4. Compute accuracy by comparing predicted and actual labels.
5. Perform backward propagation to compute gradients.
6. Update weights and biases using the computed gradients and learning rate.
7. Repeat steps 2-6 for multiple epochs.
8. Evaluate the model on test data.
9. Generate a classification report and visualize training performance.

PROGRAM

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report,
accuracy_score


np.random.seed(42)


data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target


df.head()


X = data.data
y = data.target.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


input_neurons = X_train.shape[1]
hidden_neurons = 10
output_neurons = 1
learning_rate = 0.01
epochs = 5000


W1 = np.random.randn(input_neurons, hidden_neurons) * 0.01
b1 = np.zeros((1, hidden_neurons))
W2 = np.random.randn(hidden_neurons, output_neurons) * 0.01
b2 = np.zeros((1, output_neurons))


def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)


loss_history = []
accuracy_history = []

for epoch in range(epochs):
    hidden_layer_input = np.dot(X_train, W1) + b1
    hidden_layer_output = sigmoid(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_output, W2) + b2
    predicted_output = sigmoid(output_layer_input)

    loss = np.mean(-y_train * np.log(predicted_output) - (1 -
y_train) * np.log(1 - predicted_output))
    loss_history.append(loss)

    predicted_labels = (predicted_output > 0.5).astype(int)
    accuracy = np.mean(predicted_labels == y_train) * 100
    accuracy_history.append(accuracy)

    d_output = (predicted_output - y_train) *
sigmoid_derivative(predicted_output)  # Output layer gradient
    d_hidden = np.dot(d_output, W2.T) *
sigmoid_derivative(hidden_layer_output)  # Hidden layer
gradient

    W2 -= np.dot(hidden_layer_output.T, d_output) *
learning_rate
```

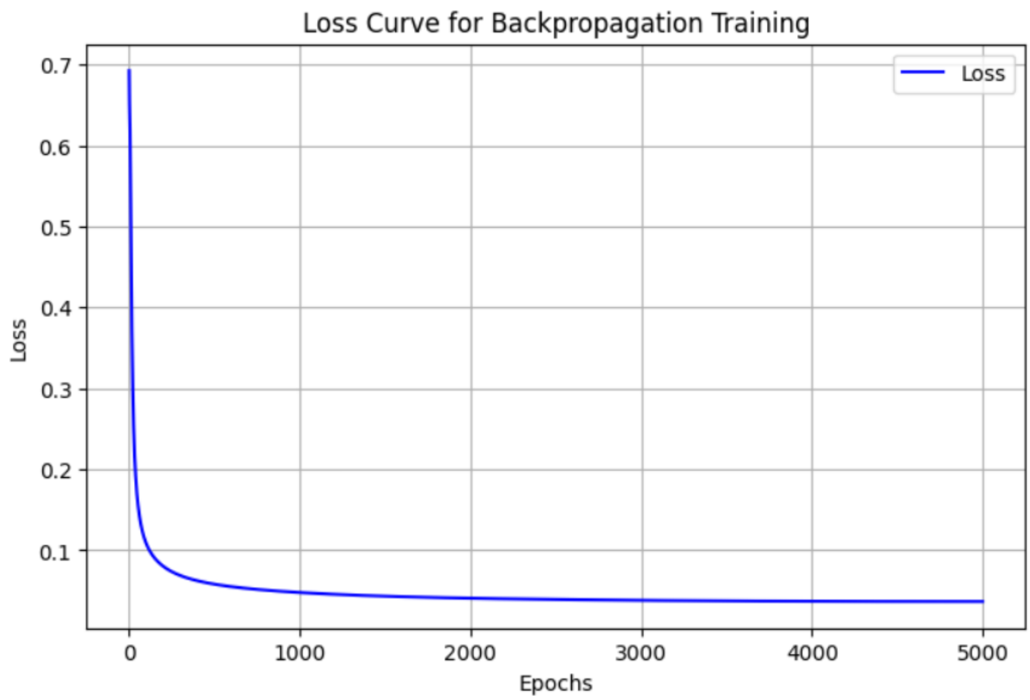
```
    b2 -= np.sum(d_output, axis=0, keepdims=True) *
learning_rate
    W1 -= np.dot(X_train.T, d_hidden) * learning_rate
    b1 -= np.sum(d_hidden, axis=0, keepdims=True) *
learning_rate

    if epoch % 500 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.6f}, Accuracy:
{accuracy:.2f}%")


hidden_layer_input = np.dot(X_test, W1) + b1
hidden_layer_output = sigmoid(hidden_layer_input)
output_layer_input = np.dot(hidden_layer_output, W2) + b2
y_pred = sigmoid(output_layer_input)
y_pred_class = (y_pred > 0.5).astype(int)


accuracy = accuracy_score(y_test, y_pred_class)
print("\nModel Accuracy:", accuracy)
print("\nClassification Report:\n",
classification_report(y_test, y_pred_class))


plt.figure(figsize=(8, 5))
plt.plot(loss_history, label="Loss", color="b")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss Curve for Backpropagation Training")
plt.legend()
plt.grid()
plt.show()
```

OUTPUT

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050

5 rows × 31 columns

```
Epoch 0, Loss: 0.036629, Accuracy: 99.78%
Epoch 500, Loss: 0.036644, Accuracy: 99.78%
Epoch 1000, Loss: 0.036731, Accuracy: 99.78%
Epoch 1500, Loss: 0.036873, Accuracy: 99.78%
Epoch 2000, Loss: 0.037055, Accuracy: 99.78%
Epoch 2500, Loss: 0.037265, Accuracy: 99.78%
Epoch 3000, Loss: 0.037493, Accuracy: 99.78%
Epoch 3500, Loss: 0.037729, Accuracy: 99.78%
Epoch 4000, Loss: 0.037970, Accuracy: 99.78%
Epoch 4500, Loss: 0.038211, Accuracy: 99.78%


Model Accuracy: 0.9824561403508771

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        43
           1       0.99      0.99      0.99        71

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```



RESULT

The Artificial Neural Network was successfully implemented using the Backpropagation algorithm. The model was trained and evaluated on a dataset, and its performance was visualized using appropriate metrics and plots.