EXPT NO : 5 **A python program to implement Multi Layer**

DATE: 20/09/24 **Perceptron With Backpropagation**

AIM:

To write a python program to implement Multilayer perceptron with backpropagation .

PROCEDURE:

Implementing Multilayer perceptron with backpropagation using the Keras dataset involve the following steps:

Step 1: Import Necessary Libraries

First, import the libraries that are essential for data manipulation, visualization, and model building.

```
# importing modules
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
import matplotlib.pyplot as plt
```

Step 2: Load the Keras Dataset

The Keras dataset can be loaded.

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

OUTPUT :

Step 3: Data Preprocessing

Ensure the data is clean and ready for modeling. Since the Iris dataset is clean, minimal preprocessing is needed.

231501075

```
# Cast the records into float values
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# normalize image pixel values by dividing
# by 255
gray_scale = 255
x_train /= gray_scale
x_test /= gray_scale

print("Feature matrix:", x_train.shape)
print("Target matrix:", x_test.shape)
print("Feature matrix:", y_train.shape)
print("Target matrix:", y_test.shape)
```

OUTPUT :

Step 4 : Train a Model

```
model = Sequential([

 # reshape 28 row * 28 column data to 28*28 rows
 Flatten(input_shape=(28, 28)) ,

 # dense layer 1
 Dense(256, activation='sigmoid') ,

 # dense layer 2
 Dense(128, activation='sigmoid') ,

 # output layer
```

```
Feature matrix: (60000, 28, 28)
Target matrix: (10000, 28, 28)
Feature matrix: (60000,)
Target matrix: (10000,)
```

```
 Dense(10, activation='sigmoid') ,
])
```

OUTPUT:

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning:
  super().__init__(**kwargs)
```

Step 5 : Make Predictions

Use the model to make predictions based on the independent variable.

```
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10,
batch_size=2000,
validation_split=0.2)
```

OUTPUT:

```
Epoch 1/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 5s 115ms/step - accuracy: 0.3546 - loss: 2.1596 - val_accuracy: 0.68
Epoch 2/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 53ms/step - accuracy: 0.7116 - loss: 1.3743 - val_accuracy: 0.826
Epoch 3/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 53ms/step - accuracy: 0.8221 - loss: 0.8221 - val_accuracy: 0.872
Epoch 4/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 65ms/step - accuracy: 0.8720 - loss: 0.5676 - val_accuracy: 0.892
Epoch 5/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 99ms/step - accuracy: 0.8907 - loss: 0.4444 - val_accuracy: 0.902
Epoch 6/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 102ms/step - accuracy: 0.8993 - loss: 0.3852 - val_accuracy: 0.91
Epoch 7/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 104ms/step - accuracy: 0.9088 - loss: 0.3416 - val_accuracy: 0.91
Epoch 8/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 92ms/step - accuracy: 0.9119 - loss: 0.3188 - val_accuracy: 0.922
Epoch 9/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 92ms/step - accuracy: 0.9191 - loss: 0.2911 - val_accuracy: 0.926
Epoch 10/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 99ms/step - accuracy: 0.9245 - loss: 0.2704 - val_accuracy: 0.929
<keras.src.callbacks.history.History at 0x7d9ca1406a40>
```

Step 6 : Evaluate the Model

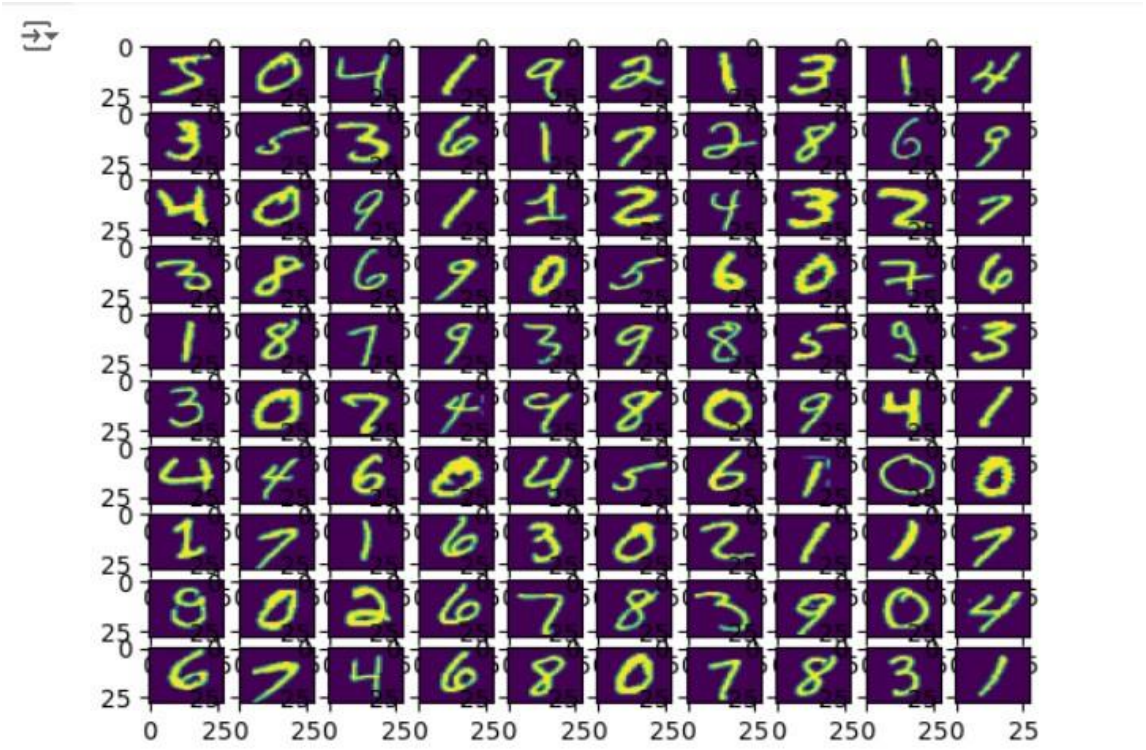
231501075

Evaluate the model performance.

```
results = model.evaluate(x_test, y_test, verbose = 0)
print('test loss, test acc:', results)
fig, ax = plt.subplots(10, 10)
k = 0
for i in range(10):
for j in range(10):
  ax[i][j].imshow(x_train[k].reshape(28, 28) ,
            aspect='auto')
   k += 1
plt.show()
```

OUTPUT :

```
test loss, test acc: [0.2589016258716583, 0.9277999997138977]
```



RESULT:

This step-by-step process will help us to implement MultiLayer Perceptron with Backpropagation models using the Keras dataset and analyze their performance.