ROLL NUMBER: 210701067

Exp No: 2

BUILD A SIMPLE NEURAL NETWORKS

AIM:

To build a simple neural network using Keras/TensorFlow.

PROCEDURE:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

PROGRAM:

```
import pandas as pd
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

dataset = loadtxt('pima-indians-diabetes-data.csv', delimiter = ',')

X = dataset[:,0:8]
y = dataset[:,8]

model = Sequential()
model.add(Dense(12, input_shape=(8,), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])model.fit(X,

y, epochs=150, batch_size=10)

_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

OUTPUT

```
[1]: import pandas as pd
     from numpy import loadtxt
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense

[2]: dataset = loadtxt('pima-indians-diabetes-data.csv', delimiter = ',')

[3]: X = dataset[:,0:8]
     y = dataset[:,8]

[4]: model = Sequential()
     model.add(Dense(12, input_shape=(8,), activation='relu'))
     model.add(Dense(8, activation='relu'))
     model.add(Dense(1, activation='sigmoid'))

     D:\Softwares\Anaconda\envs\ML\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a la
     yer. 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)

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

[25]: model.fit(X, y, epochs=150, batch_size=10)

      Epoch 1/150
      77/77 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6337 - loss: 20.1332
      Epoch 2/150
      77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5327 - loss: 3.0242
      Epoch 3/150
      77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5500 - loss: 1.6982
      Epoch 4/150
      77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5913 - loss: 1.1001
      Epoch 5/150
      77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5897 - loss: 1.1504
      Epoch 6/150
      77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6220 - loss: 0.9522
      Epoch 7/150
      77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6655 - loss: 1.0650
      Epoch 8/150
      77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6231 - loss: 1.0535
      Epoch 9/150
      77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6301 - loss: 0.8142

[26]: _, accuracy = model.evaluate(X, y)
      print('Accuracy: %.2f' % (accuracy*100))

      24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 739us/step - accuracy: 0.7159 - loss: 0.5300
      Accuracy: 71.22
```