Name - Vasu Kalariya
RoU - PE29
Sub - AI

Lab Assignment - 5

Aim: Implement Neural Network for any application

Objective:
→ To study and implement neural network for any application

Theory:

→ Neural Networks Architecture
Neural Networks are ~~computer~~ complex structures made of artificial neurons that can take in multiple inputs to produce a single output. This is the primary job of a Neural Network to transform input into a meaningful output. Usually, a neural network consist of an input and output layer with one or multiple hidden layers within. In Neural Networks all the neurons ~~in~~ influence each other, and hence they are all connected. The network can acknowledge and observe every aspect of the dataset at hand and how the different parts of data may or may not relate to each other. This is how neural networks are capable of finding extremely complex pattern in vast volume of data

→ Deep learning framework
Deep learning frameworks offers building blocks for designing, training and validat^n deep neural network, through a high level programming interface. This eliminates the need to manage packages and dependencies or build deep learning framework from source

→ Commonly used activation function
(i) Sigmoid function
(ii) Softmax function
(iii) Hyperbolic function

FAQ's

1 Which algorithm is used to train neural network
→ 1] Gradient descent
2] Newton method
3] Conjugate gradient
4] One dimensional optimization
5] Multi dimensional optimization

2 How to decide number of hidden layers in neural network?
→ For most problems one could probably get descent performance by setting the hidden layer configuration using just two rules ① Number of hidden layers equals one and ② the number of neurons is that layers is the mean of the neurons in the input and output layers.

3 What is the drawback of deep learning?
→ It requires very large amount of data in order to perform better than other techniques. It is extremely expensive to train due to complex data models. Moreover deep learning requires expensive GPUs and hundreds of mechanics. It increases the cost to the users.

Artificial Neural Network

```
import numpy as np
import pandas as pd
import tensorflow as tf
```

```
dataset = pd.read_csv('dataset.csv')
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values
```

```
print(X)
```

```
[[619 'France' 'Female' ... 1 1 101348.88]
 [608 'Spain' 'Female' ... 0 1 112542.58]
 [502 'France' 'Female' ... 1 0 113931.57]
 ...
 [709 'France' 'Female' ... 0 1 42085.58]
 [772 'Germany' 'Male' ... 1 0 92888.52]
 [792 'France' 'Female' ... 1 0 38190.78]]
```

```
print(y)
```

```
[1 0 1 ... 1 1 0]
```

Encoding categorical data

Label Encoding the "Gender" column

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
```

```
print(X)
```

```
[[619 'France' 0 ... 1 1 101348.88]
 [608 'Spain' 0 ... 0 1 112542.58]
 [502 'France' 0 ... 1 0 113931.57]
 ...
 [709 'France' 0 ... 0 1 42085.58]
 [772 'Germany' 1 ... 1 0 92888.52]
 [792 'France' 0 ... 1 0 38190.78]]
```

One Hot Encoding the "Geography" column

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

```
print(X)
```

```
[[1.0 0.0 0.0 ... 1 1 101348.88]
 [0.0 0.0 1.0 ... 0 1 112542.58]
 [1.0 0.0 0.0 ... 1 0 113931.57]
 ...
 [1.0 0.0 0.0 ... 0 1 42085.58]
 [0.0 1.0 0.0 ... 1 0 92888.52]
 [1.0 0.0 0.0 ... 1 0 38190.78]]
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Initializing the ANN

```
ann = tf.keras.models.Sequential()
```

Adding the input layer and the first hidden layer

```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

Adding the second hidden layer

```
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

Adding the output layer

```
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Compiling the ANN

```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the ANN on the Training set

```
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 71/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3349 - accuracy: 0.8622
Epoch 72/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3346 - accuracy: 0.8626
Epoch 73/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3347 - accuracy: 0.8634
Epoch 74/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3348 - accuracy: 0.8633
Epoch 75/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3346 - accuracy: 0.8626
Epoch 76/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3342 - accuracy: 0.8626
Epoch 77/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3345 - accuracy: 0.8634
Epoch 78/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3344 - accuracy: 0.8654
Epoch 79/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3336 - accuracy: 0.8636
Epoch 80/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3342 - accuracy: 0.8629
Epoch 81/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3342 - accuracy: 0.8616
Epoch 82/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3340 - accuracy: 0.8620
Epoch 83/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3338 - accuracy: 0.8629
Epoch 84/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3337 - accuracy: 0.8641
Epoch 85/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3338 - accuracy: 0.8633
Epoch 86/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3333 - accuracy: 0.8633
Epoch 87/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3336 - accuracy: 0.8644
Epoch 88/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3331 - accuracy: 0.8624
Epoch 89/100
```

```
Epoch 89/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3325 - accuracy: 0.8637
Epoch 90/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3328 - accuracy: 0.8640
Epoch 91/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3333 - accuracy: 0.8626
Epoch 92/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3332 - accuracy: 0.8637
Epoch 93/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3330 - accuracy: 0.8636
Epoch 94/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3329 - accuracy: 0.8650
Epoch 95/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3327 - accuracy: 0.8648
Epoch 96/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3325 - accuracy: 0.8641
Epoch 97/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3323 - accuracy: 0.8644
Epoch 98/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3320 - accuracy: 0.8646
Epoch 99/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3323 - accuracy: 0.8640
Epoch 100/100
250/250 [==============================] - 0s 1ms/step - loss: 0.3321 - accuracy: 0.8639
```

```
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 1]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[1523   72]
 [ 198  207]]
0.865
```