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

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df = pd.read_csv("/content/Churn_Modelling.csv")
```

```
df.head()
```

	RowNumber	CustomerId	Surname	Credi
0	1	15634602	Hargrave	
1	2	15647311	Hill	
2	3	15619304	Onio	
3	4	15701354	Boni	
4	5	15737888	Mitchell	

```
# for X, take 4th column to untill last second column as independent variable and exited as dependent variable
X = df.iloc[:, 3:-1].values
y = df.iloc[:, -1].values
```

```
X
```

```
array([[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]], dtype=object)
```

```
y
```

```
array([1, 0, 1, ..., 1, 1, 0])
```

```
#lebel encodding for gender column
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
```

```
#one hot encoding for country column, for this first import all necesary moddule
#Then, define the transformer function which has list of transformer which transform value into one hot
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

```
#define column transfer function where instruct which column need to apply the function
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')
```

```
#apply the CT to X
X = np.array(ct.fit_transform(X))
```

```
#feature scaling with std_scaler
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
```

```
X
```

```
array([[ 0.99720391, -0.57873591, -0.57380915, ...,  0.64609167,
         0.97024255,  0.02188649],
       [-1.00280393, -0.57873591,  1.74273971, ..., -1.54776799,
```

```
          0.97024255,  0.21653375],
        [ 0.99720391, -0.57873591, -0.57380915, ...,  0.64609167,
         -1.03067011,  0.2406869 ],
        ...,
        [ 0.99720391, -0.57873591, -0.57380915, ..., -1.54776799,
          0.97024255, -1.00864308],
        [-1.00280393,  1.72790383, -0.57380915, ...,  0.64609167,
         -1.03067011, -0.12523071],
        [ 0.99720391, -0.57873591, -0.57380915, ...,  0.64609167,
         -1.03067011, -1.07636976]])
```

```
#spliting the dataset intotrain and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Double-click (or enter) to edit

```
Building **MNN**
```

Building **MNN**

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

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

```
#adding 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 third hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

# Adding the fourth hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

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

Adam stands for Adaptive Moment Estimation, and it's an extension of the stochastic gradient descent (SGD) algorithm. Adam adapts the learning rates of each parameter based on the past gradients and their squared gradients.

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

```
#training the ANN with training set with epoch 120
ann.fit(X_train, y_train, batch_size = 32, epochs = 120)
```

```
Epoch 1/120
235/235 [==============================] - 2s 2ms/step - loss: 0.5646 - accuracy:
Epoch 2/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4568 - accuracy:
Epoch 3/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4362 - accuracy:
Epoch 4/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4252 - accuracy:
Epoch 5/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4162 - accuracy:
Epoch 6/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4076 - accuracy:
Epoch 7/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3996 - accuracy:
Epoch 8/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3902 - accuracy:
Epoch 9/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3808 - accuracy:
Epoch 10/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3741 - accuracy:
```

```
Epoch 11/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3688 - accuracy:
Epoch 12/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3641 - accuracy:
Epoch 13/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3611 - accuracy:
Epoch 14/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3586 - accuracy:
Epoch 15/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3563 - accuracy:
Epoch 16/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3553 - accuracy:
Epoch 17/120
235/235 [==============================] - 1s 2ms/step - loss: 0.3544 - accuracy:
Epoch 18/120
235/235 [==============================] - 1s 3ms/step - loss: 0.3531 - accuracy:
Epoch 19/120
235/235 [==============================] - 1s 3ms/step - loss: 0.3518 - accuracy:
Epoch 20/120
235/235 [==============================] - 1s 3ms/step - loss: 0.3509 - accuracy:
Epoch 21/120
235/235 [==============================] - 1s 2ms/step - loss: 0.3500 - accuracy:
Epoch 22/120
235/235 [==============================] - 1s 2ms/step - loss: 0.3498 - accuracy:
Epoch 23/120
235/235 [==============================] - 1s 3ms/step - loss: 0.3488 - accuracy:
Epoch 24/120
235/235 [==============================] - 1s 3ms/step - loss: 0.3482 - accuracy:
Epoch 25/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3469 - accuracy:
Epoch 26/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3467 - accuracy:
Epoch 27/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3465 - accuracy:
Epoch 28/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3466 - accuracy:
Epoch 29/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3470 - accuracy:
```

nobel barua
04:07 Today

Take 2 minute for epoch 120 and MNN

```
# predict with x test and see the orginal y test and ypred line by line
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))
```

```
79/79 [==============================] - 0s 2ms/step
[[0 0]
 [0 1]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1873  118]
 [ 243  266]]
```

✓ 0s completed at 04:08