Keras and Tensorflow

```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

Dataset: Credit Card Customer Churn Prediction

- 14 Columns
- 10,000 rows

Problem: Binary Classification

```
df = pd.read_csv(r'https://raw.githubusercontent.com/hamzanasirr/Exploratory-Data-Analysis-on-Bank-Customer-Churn-data/refs/heads/master/Churn_Modelling.csv')

df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

```
df.drop(columns = ['RowNumber','CustomerId','Surname'],inplace=True)
df.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CreditScore      10000 non-null  int64
 1   Geography        10000 non-null  object
 2   Gender           10000 non-null  object
 3   Age              10000 non-null  int64
 4   Tenure           10000 non-null  int64
 5   Balance          10000 non-null  float64
 6   NumOfProducts    10000 non-null  int64
 7   HasCrCard        10000 non-null  int64
 8   IsActiveMember   10000 non-null  int64
 9   EstimatedSalary  10000 non-null  float64
 10  Exited           10000 non-null  int64
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

- **No missing values**

```
df.duplicated().sum()

0
```

- No dup rows

```
df.Exited.value_counts()
```

```
Exited
0    7963
1    2037
Name: count, dtype: int64
```

```
df['Geography'].value_counts()
```

```
Geography
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
```

```
df['Gender'].value_counts()
```

```
Gender
Male      5457
Female    4543
Name: count, dtype: int64
```

One-Hot Encoding of CAT columns

```
df = pd.get_dummies(df,columns=['Geography','Gender'],drop_first=True)
df.head()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Geography_Germany	Geography_Spain	Gender_Male
0	619	42	2	0.00	1	1	1	101348.88	1	False	False	False
1	608	41	1	83807.86	1	0	1	112542.58	0	False	True	False
2	502	42	8	159660.80	3	1	0	113931.57	1	False	False	False
3	699	39	1	0.00	2	0	0	93826.63	0	False	False	False
4	850	43	2	125510.82	1	1	1	79084.10	0	False	True	False

Train-test Split

```
X = df.drop(columns=['Exited'])
y = df['Exited']

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)
```

Scale the columns

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_trf = scaler.fit_transform(X_train)
X_test_trf = scaler.transform(X_test)
```

```
X_train_trf
✓ 0.0s
```

```
array([[ 0.16958176, -0.46460796,  0.00666099, ..., -0.5698444 ,
         1.74309049, -1.09168714],
       [-2.30455945,  0.30102557, -1.37744033, ...,  1.75486502,
        -0.57369368,  0.91601335],
       [-1.19119591, -0.94312892, -1.031415  , ..., -0.5698444 ,
        -0.57369368, -1.09168714],
       ...,
       [ 0.9015152 , -0.36890377,  0.00666099, ..., -0.5698444 ,
        -0.57369368,  0.91601335],
       [-0.62420521, -0.08179119,  1.39076231, ..., -0.5698444 ,
         1.74309049, -1.09168714],
       [-0.28401079,  0.87525072, -1.37744033, ...,  1.75486502,
        -0.57369368, -1.09168714]])
```

TensorFlow

Install:

```
!pip install tensorflow
```

Import Libraries:

```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

1. `import tensorflow` :

- **TensorFlow** is an open-source library developed by Google for machine learning and deep learning tasks. It provides a framework to design, train, and deploy machine learning models, especially neural networks.

2. `from tensorflow import keras` :

- **Keras** is a high-level neural networks API, built on top of TensorFlow, that simplifies the process of creating and training deep learning models. It provides easy-to-use tools for building neural networks, like layer structures, optimizers, loss functions, etc.

3. `from tensorflow.keras import Sequential` :

- **Sequential** is a model type in Keras that allows you to build a neural network layer by layer. You simply stack layers in a linear order, where each layer's output is the next layer's input. This is typically used for simpler, feedforward networks.

4. `from tensorflow.keras.layers import Dense` :

- **Dense** is a fully connected layer in a neural network. Each neuron in a Dense layer is connected to every neuron in the previous layer. It is the most common type of layer used in many deep learning models.

Steps to build a model:

1. **Create an Object**

```
model = Sequential()
```

2. **Add Layers:**

Hidden Layer:

```
model.add(Dense(3,activation='sigmoid', input_dim=11))
```

`3` → 3 Nodes (3 perceptrons)

`activation='sigmoid'` → For binary classification

`input_dim=11` → It will get 11 inputs

Output Layer:

```
model.add(Dense(1,activation='sigmoid'))
```

- `1` → 1 Layer

```
model = Sequential()

model.add(Dense(3,activation='sigmoid', input_dim=11))
model.add(Dense(1,activation='sigmoid'))
```

```
model.summary()
```

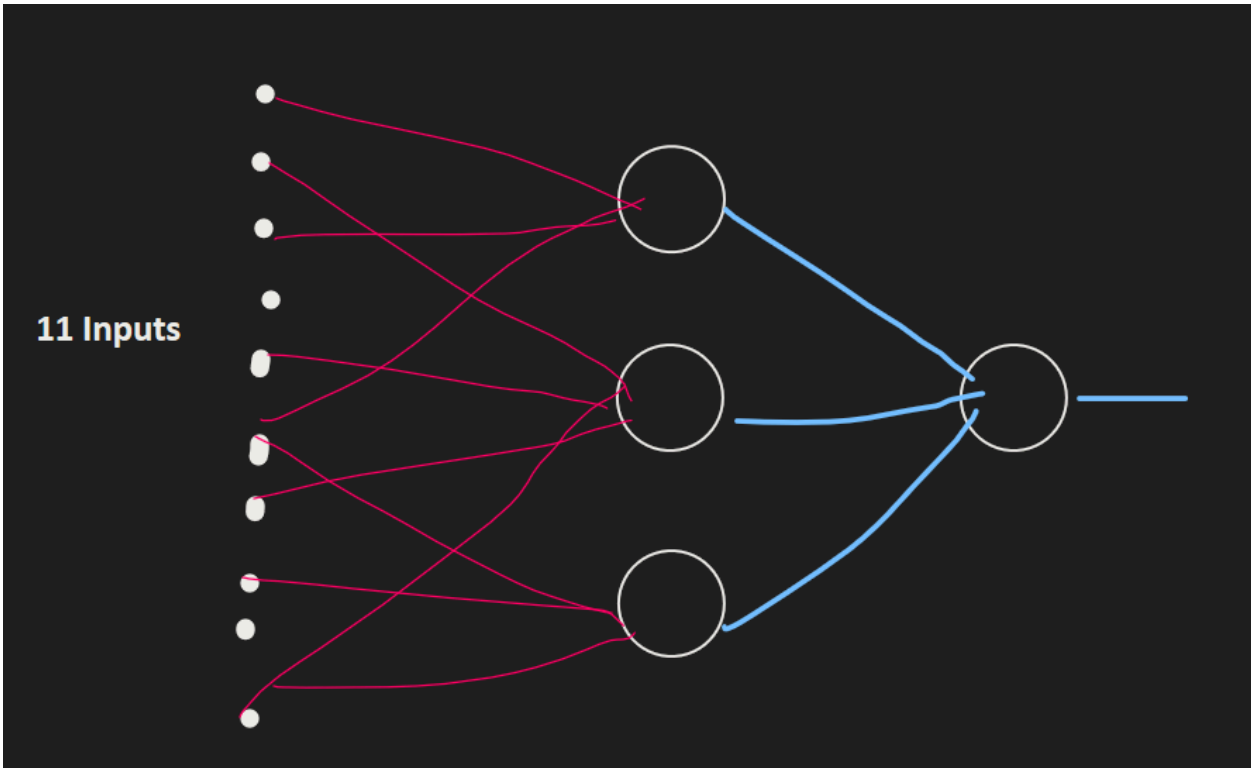
```
Model: "sequential"

| Layer (type)          | Output Shape    | Param # |
|-----------------------|-----------------|---------|
| dense (Dense)         | (None, 3)       |      36 |
| dense_1 (Dense)       | (None, 1)       |       4 |

Total params: 40 (160.00 B)

Trainable params: 40 (160.00 B)

Non-trainable params: 0 (0.00 B)
```



Compile the Model

- Specify the **loss function, optimizer**, etc

```
model.compile(optimizer='Adam',loss='binary_crossentropy')
```

Fit the model

```
model.fit(X_train_trf, y_train, epochs=10)
```

`epochs=10`

- One epoch is one full iteration through the entire training dataset.
- In each epoch, the model makes predictions based on the current weights, compares the predictions to the actual targets (ground truth), and then updates the weights based on the error (using backpropagation and optimization techniques like gradient descent).

> 💡 Here, we calculate **weights & biases**.

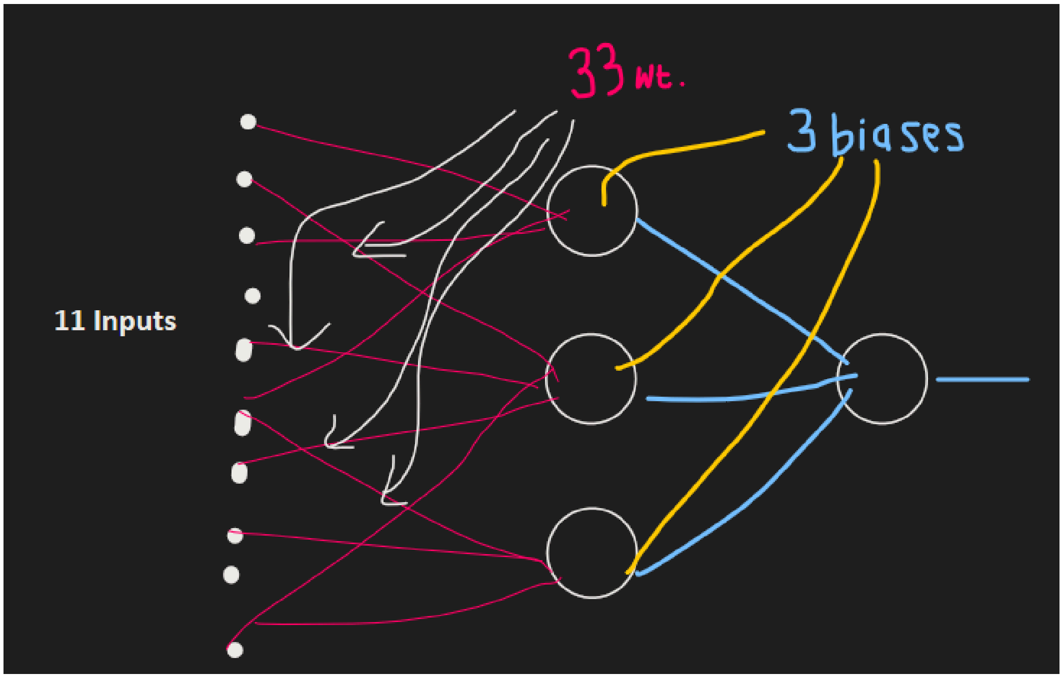
weights:

```
model.layers[0].get_weights()
```

```
[array([[ 0.23044327,  0.10794672,  0.31715706],
        [ 0.6596899 , -1.1845328 , -1.4722042 ],
        [ 0.20926282,  0.21099435,  0.1484614 ],
        [ 0.4485632 , -0.6650833 ,  0.06855625],
        [-0.8331062 , -0.36040655,  0.5141464 ],
        [-0.02098621,  0.16122589, -0.08359355],
        [-0.99188805,  0.6205362 ,  0.5666368 ],
        [-0.2822069 , -0.4446395 ,  0.11166077],
        [ 0.29588562, -0.8850321 , -0.39562535],
        [ 0.38293236,  0.05097587,  0.09171998],
        [-0.71892524,  0.51194906,  0.29488176]], dtype=float32),
 array([-0.4649123,  0.6746617,  0.6777313], dtype=float32)]
```

33 wt.

3 biases




```
model.layers[1].get_weights()
```

```
[array([[ 0.6422341],
        [-1.0281332],
        [-1.1344699]], dtype=float32),
 array([-0.47845036], dtype=float32)]
```

- 👆 3 Weights & 1 bias for layer 2.

Predict:

```
y_log =model.predict(X_test_trf)
```

```
array([[0.14862315],
       [0.24334534],
       [0.20895432],
       ...,
       [0.14640807],
       [0.09730779],
       [0.51069075]], dtype=float32)
```

💡 **The output is not 0/1 because we're using sigmoid function.**

- We have to convert these values into 0/1 by using a threshold.

Threshold=0.5

```
np.where(y_log>0.5,1,0)
```

- If `y_log>0.5` → Return `1`
  - Else: return `0`

- Store the above values in `y_pred`

```
y_pred = np.where(y_log>0.5,1,0)
```

Calculate the Accuracy

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)

0.812
```

Improve the Accuracy

- Increase the no. of epochs
- Change the **activation function** to `relu`
- Increase the number of nodes from 3 to → 10, 20, 30, etc.
- Increase the number of layers
    - More layers can cause overfitting

```
model = Sequential()

model.add(Dense(11,activation='relu',input_dim=11))
model.add(Dense(11,activation='relu')) #Added Hidden layer
model.add(Dense(1,activation='sigmoid'))
```

Add accuracy metric:

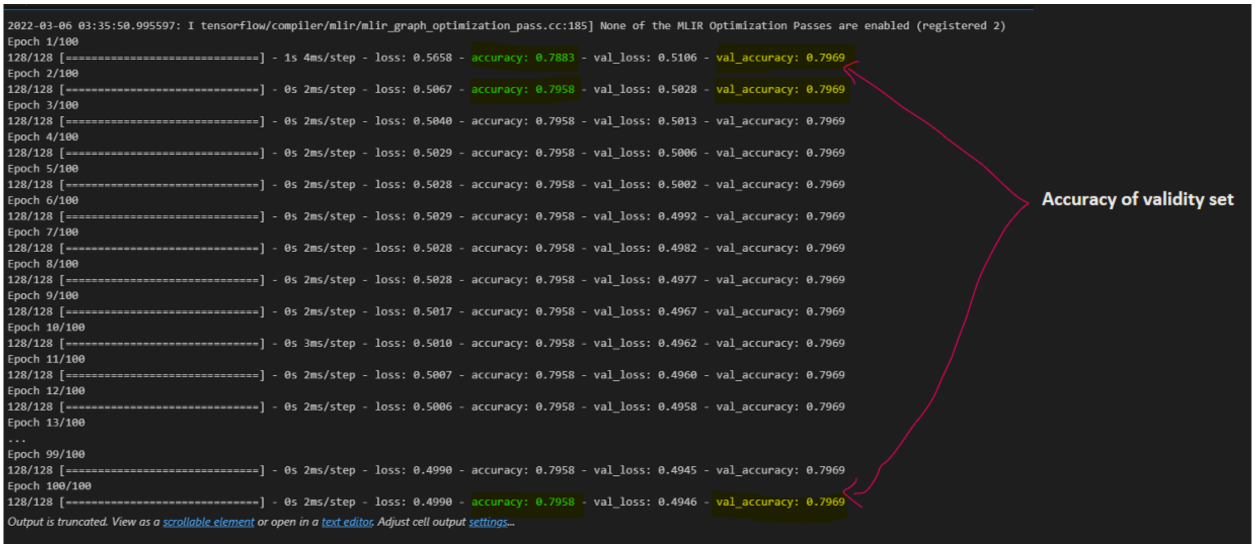
```
model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['accuracy'])
```

Validation split:

```
model.fit(X_train,y_train,batch_size=50,epochs=100,verbose=1,validation_split=0.2)
```

`validation_split=0.2` → This will separate 20% of training data (20% of 80%) for testing.

- It'll give accuracy score of the 20% data.



```
2022-03-06 03:35:50.995597: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/100
128/128 [==============================] - 1s 4ms/step - loss: 0.5658 - accuracy: 0.7883 - val_loss: 0.5106 - val_accuracy: 0.7969
Epoch 2/100
128/128 [==============================] - 0s 2ms/step - loss: 0.5067 - accuracy: 0.7958 - val_loss: 0.5028 - val_accuracy: 0.7969
Epoch 3/100
128/128 [==============================] - 0s 2ms/step - loss: 0.5040 - accuracy: 0.7958 - val_loss: 0.5013 - val_accuracy: 0.7969
Epoch 4/100
128/128 [==============================] - 0s 2ms/step - loss: 0.5029 - accuracy: 0.7958 - val_loss: 0.5006 - val_accuracy: 0.7969
Epoch 5/100
128/128 [==============================] - 0s 2ms/step - loss: 0.5028 - accuracy: 0.7958 - val_loss: 0.5002 - val_accuracy: 0.7969
Epoch 6/100
128/128 [==============================] - 0s 2ms/step - loss: 0.5029 - accuracy: 0.7958 - val_loss: 0.4992 - val_accuracy: 0.7969
Epoch 7/100
128/128 [==============================] - 0s 2ms/step - loss: 0.5028 - accuracy: 0.7958 - val_loss: 0.4982 - val_accuracy: 0.7969
Epoch 8/100
128/128 [==============================] - 0s 2ms/step - loss: 0.5028 - accuracy: 0.7958 - val_loss: 0.4977 - val_accuracy: 0.7969
Epoch 9/100
128/128 [==============================] - 0s 2ms/step - loss: 0.5017 - accuracy: 0.7958 - val_loss: 0.4967 - val_accuracy: 0.7969
Epoch 10/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5010 - accuracy: 0.7958 - val_loss: 0.4962 - val_accuracy: 0.7969
Epoch 11/100
128/128 [==============================] - 0s 2ms/step - loss: 0.5007 - accuracy: 0.7958 - val_loss: 0.4960 - val_accuracy: 0.7969
Epoch 12/100
128/128 [==============================] - 0s 2ms/step - loss: 0.5006 - accuracy: 0.7958 - val_loss: 0.4958 - val_accuracy: 0.7969
Epoch 13/100
...
Epoch 99/100
128/128 [==============================] - 0s 2ms/step - loss: 0.4990 - accuracy: 0.7958 - val_loss: 0.4945 - val_accuracy: 0.7969
Epoch 100/100
128/128 [==============================] - 0s 2ms/step - loss: 0.4990 - accuracy: 0.7958 - val_loss: 0.4946 - val_accuracy: 0.7969
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```


Plot a graph

- Store the `model.fit()` in a variable

```
history= model.fit(X_train,y_train,batch_size=50,epochs=100,verbose=1,validation_split=0.2)
```

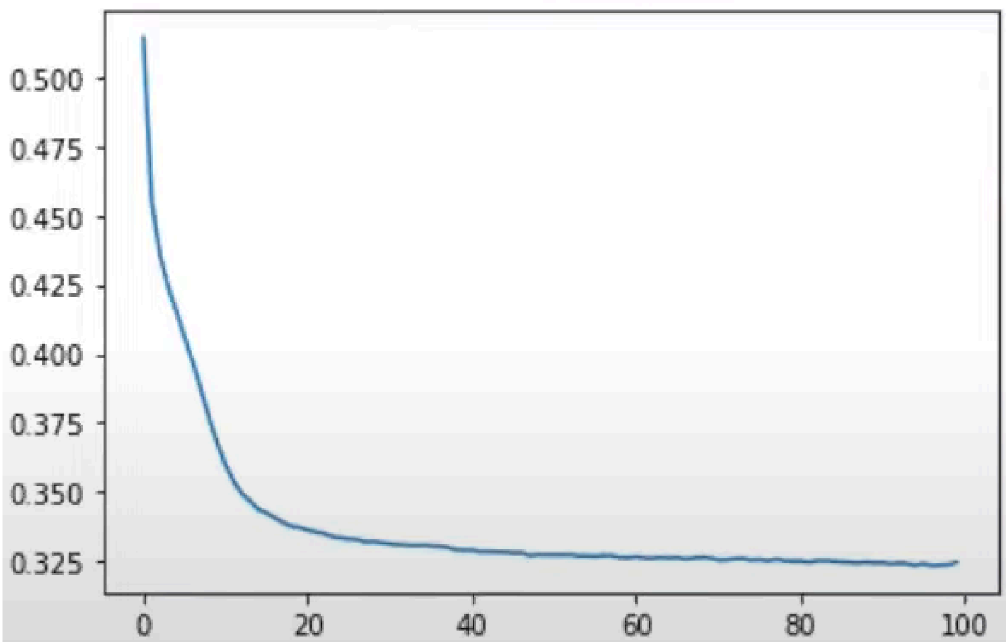
```
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
```

```
plt.plot(graph.history['accuracy'])
plt.plot(graph.history['val_accuracy'])
```

```
plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7eff63137790>]
```



```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x7eff6300bcd0>]
```