Implementing a Differentially Private ANN with TensorFlow Privacy, on the Churn Modelling Dataset

Importing Libraries

In [2]:

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import tensorflow as tf

import numpy as np
import pandas as pd
```

Install TensorFlow Privacy.

In [3]:

```
!pip install tensorflow_privacy

from tensorflow_privacy.privacy.analysis import compute_dp_sgd_privacy
from tensorflow_privacy.privacy.optimizers.dp_optimizer_keras import DPKerasSGDOptimizer
from tensorflow_privacy.privacy.optimizers.dp_optimizer_keras import DPKerasAdamOptimizer
```

```
Collecting tensorflow_privacy
  Downloading tensorflow_privacy-0.7.3-py3-none-any.whl (251 kB)
     |████████████████████████████████| 251 kB 5.3 MB/s
Requirement already satisfied: mpmath in /usr/local/lib/python3.7/dist-packa
ges (from tensorflow_privacy) (1.2.1)
Requirement already satisfied: tensorflow-probability>=0.13.0 in /usr/local/
lib/python3.7/dist-packages (from tensorflow_privacy) (0.15.0)
Requirement already satisfied: scipy>=0.17 in /usr/local/lib/python3.7/dist-
packages (from tensorflow_privacy) (1.4.1)
Requirement already satisfied: dm-tree~=0.1.1 in /usr/local/lib/python3.7/di
st-packages (from tensorflow_privacy) (0.1.6)
Requirement already satisfied: attrs>=21.2.0 in /usr/local/lib/python3.7/dis
t-packages (from tensorflow_privacy) (21.2.0)
Requirement already satisfied: tensorflow-estimator>=2.3.0 in /usr/local/li
b/python3.7/dist-packages (from tensorflow_privacy) (2.7.0)
Collecting tensorflow-datasets>=4.4.0
  Downloading tensorflow_datasets-4.4.0-py3-none-any.whl (4.0 MB)
     |████████████████████████████████| 4.0 MB 34.1 MB/s
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-
packages (from dm-tree~=0.1.1->tensorflow_privacy) (1.15.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dis
t-packages (from scipy>=0.17->tensorflow_privacy) (1.19.5)
Requirement already satisfied: promise in /usr/local/lib/python3.7/dist-pack
ages (from tensorflow-datasets>=4.4.0->tensorflow_privacy) (2.3)
Requirement already satisfied: tensorflow-metadata in /usr/local/lib/python
3.7/dist-packages (from tensorflow-datasets>=4.4.0->tensorflow_privacy) (1.
4.0)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.7/
dist-packages (from tensorflow-datasets>=4.4.0->tensorflow_privacy) (2.23.0)
Requirement already satisfied: termcolor in /usr/local/lib/python3.7/dist-pa
ckages (from tensorflow-datasets>=4.4.0->tensorflow_privacy) (1.1.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.
7/dist-packages (from tensorflow-datasets>=4.4.0->tensorflow_privacy) (3.10.
0.2)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packa
ges (from tensorflow-datasets>=4.4.0->tensorflow_privacy) (0.16.0)
Requirement already satisfied: protobuf>=3.12.2 in /usr/local/lib/python3.7/
dist-packages (from tensorflow-datasets>=4.4.0->tensorflow_privacy) (3.17.3)
Requirement already satisfied: dill in /usr/local/lib/python3.7/dist-package
s (from tensorflow-datasets>=4.4.0->tensorflow_privacy) (0.3.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-package
s (from tensorflow-datasets>=4.4.0->tensorflow_privacy) (4.62.3)
Requirement already satisfied: importlib-resources in /usr/local/lib/python
3.7/dist-packages (from tensorflow-datasets>=4.4.0->tensorflow_privacy) (5.
4.0)
Requirement already satisfied: absl-py in /usr/local/lib/python3.7/dist-pack
ages (from tensorflow-datasets>=4.4.0->tensorflow_privacy) (0.12.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist
-packages (from requests>=2.19.0->tensorflow-datasets>=4.4.0->tensorflow_pri
vacy) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
7/dist-packages (from requests>=2.19.0->tensorflow-datasets>=4.4.0->tensorfl
ow_privacy) (2021.10.8)
```

```
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /u
sr/local/lib/python3.7/dist-packages (from requests>=2.19.0->tensorflow-data
sets>=4.4.0->tensorflow_privacy) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.
7/dist-packages (from requests>=2.19.0->tensorflow-datasets>=4.4.0->tensorfl
ow_privacy) (3.0.4)
Requirement already satisfied: gast>=0.3.2 in /usr/local/lib/python3.7/dist-
packages (from tensorflow-probability>=0.13.0->tensorflow_privacy) (0.4.0)
Requirement already satisfied: cloudpickle>=1.3 in /usr/local/lib/python3.7/
dist-packages (from tensorflow-probability>=0.13.0->tensorflow_privacy) (1.
3.0)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-pa
ckages (from tensorflow-probability>=0.13.0->tensorflow_privacy) (4.4.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.7/dist-
packages (from importlib-resources->tensorflow-datasets>=4.4.0->tensorflow_p
rivacy) (3.6.0)
Requirement already satisfied: googleapis-common-protos<2,>=1.52.0 in /usr/l
ocal/lib/python3.7/dist-packages (from tensorflow-metadata->tensorflow-datas
ets>=4.4.0->tensorflow_privacy) (1.53.0)
Installing collected packages: tensorflow-datasets, tensorflow-privacy
  Attempting uninstall: tensorflow-datasets
    Found existing installation: tensorflow-datasets 4.0.1
    Uninstalling tensorflow-datasets-4.0.1:
      Successfully uninstalled tensorflow-datasets-4.0.1
Successfully installed tensorflow-datasets-4.4.0 tensorflow-privacy-0.7.3
```

Load pre-processed Churn Modelling Dataset and Create Test and Train Datasets with Split

In [4]:

```
df = pd.read_csv('Churn_Modelling.csv') #Reading the Churn Modelling Dataset we have for th
df.head()
```

Out[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balanc
0	1	15634602	Hargrave	619	France	Female	42	2	0.0
1	2	15647311	Hill	608	Spain	Female	41	1	83807.8
2	3	15619304	Onio	502	France	Female	42	8	159660.8
3	4	15701354	Boni	699	France	Female	39	1	0.0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.8

In [5]:

```
df=pd.get_dummies(df, columns=['Geography','Gender']) #Using One Hot Encoding
```

In [6]:

```
# Dropping columns which either don't help in prediction or can be used to identify individ

df.drop(columns = ['RowNumber', 'CustomerId', 'Surname'], axis = 1, inplace = True)
df.head()
```

Out[6]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estimate
0	619	42	2	0.00	1	1	1	10
1	608	41	1	83807.86	1	0	1	11
2	502	42	8	159660.80	3	1	0	11
3	699	39	1	0.00	2	0	0	9
4	850	43	2	125510.82	1	1	1	7

Load the created Training Dataset

In [7]:

```
X = df.loc[:, df.columns != 'Exited'] #Getting the Training Features (X)
```

In [8]:

```
X.head()
```

Out[8]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estimate
0	619	42	2	0.00	1	1	1	10
1	608	41	1	83807.86	1	0	1	11
2	502	42	8	159660.80	3	1	0	11
3	699	39	1	0.00	2	0	0	9
4	850	43	2	125510.82	1	1	1	7

In [11]:

```
y = df["Exited"].astype('float')
```

In [12]:

```
y.head()
```

Out[12]:

```
0    1.0
1    0.0
2    1.0
3    0.0
4    0.0
Name: Exited, dtype: float64
```

Train-Test Split

In [13]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.01, random_state =
```

In [14]:

```
X_train
```

Out[14]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estim
861	548	32	2	172448.77	1	1	0	
3285	735	43	9	98807.45	1	0	0	
7532	760	33	1	118114.28	2	0	1	
1845	737	36	9	0.00	1	0	1	
8051	605	56	1	74129.18	2	1	1	
...	...	...	...	...	...	...	...	
6738	573	35	9	134498.54	2	1	1	
6278	602	72	3	0.00	2	1	1	
3361	602	48	7	76595.08	2	0	0	
9357	418	46	9	0.00	1	1	1	
8407	726	28	2	0.00	1	0	0	

9900 rows × 13 columns

In [15]:

```
# Scaling data

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
#X_val = sc.transform(X_val)
X_test = sc.transform(X_test)
# For final test scale and predict with train scaler
```

Defining model hyperparameters

In [16]:

```
epochs = 100
batch_size = 128
```

In [17]:

```
l2_norm_clip = 1.5
noise_multiplier = 1.3
num_microbatches = 64
learning_rate = 0.25

if batch_size % num_microbatches != 0:
  raise ValueError('Batch size should be an integer multiple of the number of microbatches'
```

Building our DP ANN

In [38]:

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import BatchNormalization
callback = keras.callbacks.EarlyStopping(patience=40, restore_best_weights=True)
# initializing ann
model = Sequential()

# adding the first input layer and the first hidden layer
model.add(Dense(18, kernel_initializer = 'normal', activation = 'relu', input_shape = (13,

# adding batch normalization and dropout layer
model.add(Dropout(rate = 0.1))
model.add(BatchNormalization())

# adding the third hidden layer
model.add(Dense(14, kernel_initializer = 'normal', activation = 'relu'))

# adding batch normalization and dropout layer
model.add(Dropout(rate = 0.1))
model.add(BatchNormalization())

model.add(Dense(10, kernel_initializer = 'normal', activation = 'relu'))

# adding batch normalization and dropout layer
model.add(Dropout(rate = 0.1))
model.add(BatchNormalization())

model.add(Dense(6, kernel_initializer = 'normal', activation = 'relu'))

# adding batch normalization and dropout layer
model.add(Dropout(rate = 0.05))
model.add(BatchNormalization())

# adding the output layer
model.add(Dense(1, kernel_initializer = 'normal', activation = 'sigmoid'))
```

Define the optimizer and loss function for the learning model. We compute the loss by using a vector of losses per-example, so we can support gradient manipulation over each training point.

This is different compared to the standard mean over a minibatch

In [39]:

```
optimizer = DPKerasSGDOptimizer(
    l2_norm_clip=l2_norm_clip,
    noise_multiplier=noise_multiplier,
    learning_rate=learning_rate)

loss = tf.keras.losses.BinaryCrossentropy(reduction=tf.losses.Reduction.NONE)
```

Compiling and training the DP ANN Model

In [40]:

```
import time, datetime #Used for training time
```

In [41]:

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


start = datetime.datetime.now()
model_history = model.fit(X_train, y_train,
          epochs=epochs,
          validation_split=0.1,
          batch_size = batch_size, callbacks = [callback])
end = datetime.datetime.now()
print("Overall Training Time: ", (end - start))
```

```
Epoch 1/100
70/70 [==============================] - 9s 43ms/step - loss: 0.5793 - acc
uracy: 0.7175 - val_loss: 0.5234 - val_accuracy: 0.7889
Epoch 2/100
70/70 [==============================] - 3s 41ms/step - loss: 0.5791 - acc
uracy: 0.7744 - val_loss: 0.5408 - val_accuracy: 0.7889
Epoch 3/100
70/70 [==============================] - 3s 40ms/step - loss: 0.6255 - acc
uracy: 0.8067 - val_loss: 0.6497 - val_accuracy: 0.7889
Epoch 4/100
70/70 [==============================] - 3s 40ms/step - loss: 0.6387 - acc
uracy: 0.8159 - val_loss: 0.6442 - val_accuracy: 0.7970
Epoch 5/100
70/70 [==============================] - 3s 42ms/step - loss: 0.6523 - acc
uracy: 0.8263 - val_loss: 0.5962 - val_accuracy: 0.8222
Epoch 6/100
70/70 [==============================] - 3s 42ms/step - loss: 0.6116 - acc
uracy: 0.8354 - val_loss: 0.6108 - val_accuracy: 0.8131
Epoch 7/100
```

In [21]:

```
acc = model.evaluate(X_test, y_test)[1]

print(f'Accuracy of model is {acc}')
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.3476 - accurac
y: 0.9000
Accuracy of model is 0.8999999761581421
```

In [22]:

```
threshold = 0.5
y_prob = model.predict(X_test)
y_pred = np.where(y_prob >= threshold, 1,0)
```

In [28]:

```
 from sklearn.metrics import confusion_matrix
 from sklearn.metrics import precision_recall_fscore_support
 import matplotlib.pyplot as plt
 import seaborn as sns
```

In [29]:

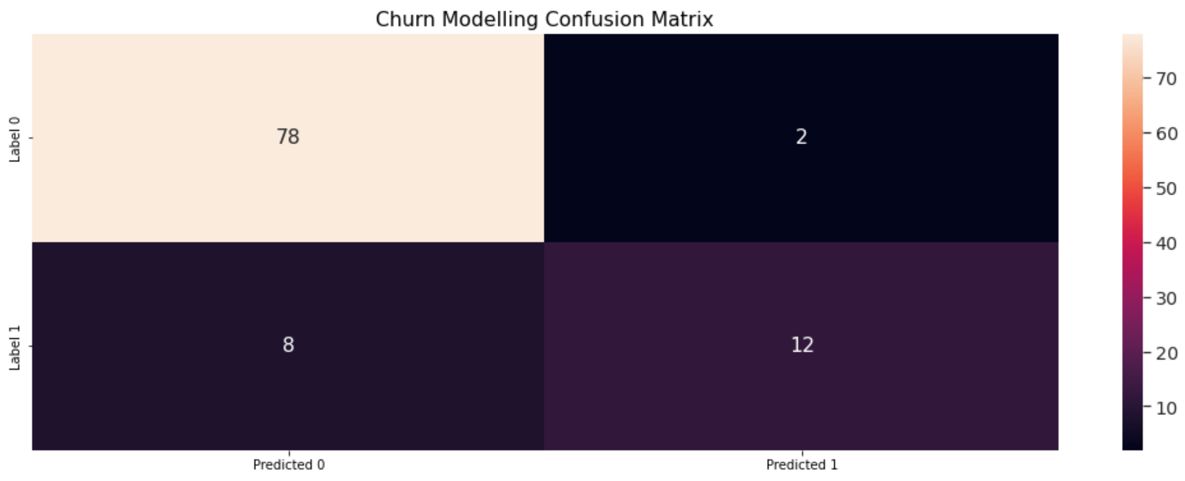
```
# Confusion Matrix for Basic Logistic Regression Model
conf_Log = confusion_matrix(y_test, y_pred)
fig, axes = plt.subplots(figsize=(18,6))

new_conf = pd.DataFrame(conf_Log, index = ['Label 0','Label 1'],columns = ['Predicted 0','P


sns.set(font_scale=1.3)
sns.heatmap(new_conf.iloc[-1::-1,:], annot=True,fmt= 'g', ax = axes)
axes.set_ylim([0,2]);
axes.set_title('Churn Modelling Confusion Matrix')
```

Out[29]:

```
Text(0.5, 1.0, 'Churn Modelling Confusion Matrix')
```



In [30]:

```
# Precision: Whenever Model Predicts True, how many times it is actually true
# Recall: Out of all True, how many did the Model find out

# Precision:   True Positives/(True Positives + False Positives)
# Recall:   True Positives/(True Positives + False Negatives)

TP = conf_Log[1][1]
FP = conf_Log[0][1]
TN = conf_Log[0][0]
FN = conf_Log[1][0]
```

In [31]:

```
Precision_Metric = TP/(TP + FP)
print("Precision is:", Precision_Metric)
```

```
Precision is: 0.8571428571428571
```

In [32]:

```
Recall_Metric = TP/(TP + FN)
print("Recall is:", Recall_Metric)
```

```
Recall is: 0.6
```

In [33]:

```
F1_Metric = (2*Precision_Metric*Recall_Metric)/(Precision_Metric + Recall_Metric)
print("F1 Score is:", F1_Metric)
```

```
F1 Score is: 0.7058823529411764
```

In [34]:

```
model.save_weights("model_DP_ANN.h5")
```

In [48]:

```
compute_dp_sgd_privacy.compute_dp_sgd_privacy(n=9900, batch_size=128, noise_multiplier=1.3,
```

```
DP-SGD with sampling rate = 1.29% and noise_multiplier = 1.3 iterated over 2
321 steps satisfies differential privacy with eps = 2.58 and delta = 1e-05.
The optimal RDP order is 8.0.
```

Out[48]:

```
(2.5776593476915757, 8.0)
```

In [47]:

```
compute_dp_sgd_privacy.compute_dp_sgd_privacy(n=9900, batch_size=128, noise_multiplier=1.3,
```

```
DP-SGD with sampling rate = 1.29% and noise_multiplier = 1.3 iterated over 7
735 steps satisfies differential privacy with eps = 4.97 and delta = 1e-05.
The optimal RDP order is 5.0.
```

Out[47]:

```
(4.968762552641338, 5.0)
```