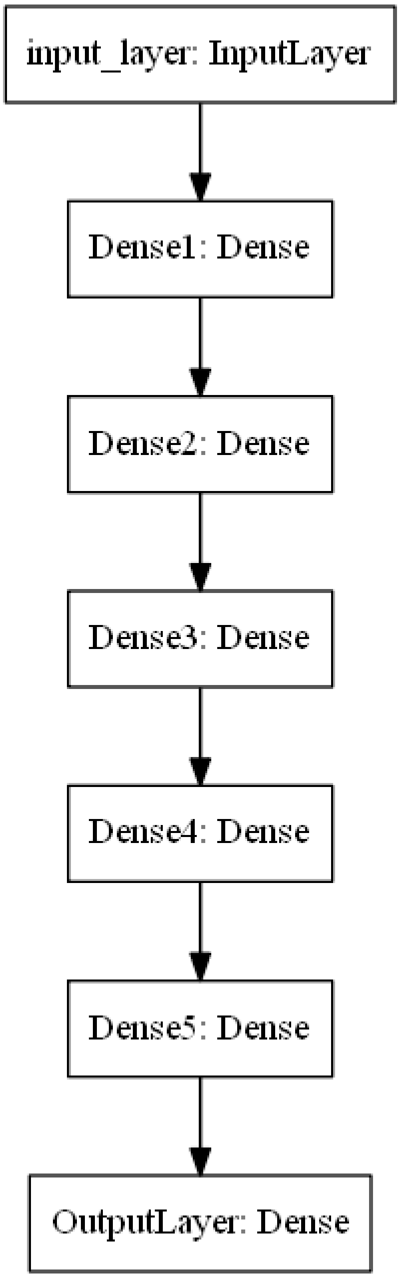
1. Download the data from here. You have to use data.csv file for this assignment

2. Code the model to classify data like below image. You can use any number of units in your Dense layers.



3. Writing Callbacks

You have to implement the following callbacks

- Write your own callback function, that has to print the micro F1 score and AUC score after each epoch.Do not use tf.keras.metrics for calculating AUC and F1 score.

- Save your model at every epoch if your validation accuracy is improved from previous epoch.
- You have to decay learning based on below conditions

```
Cond1. If your validation accuracy at that epoch is less than previous e
       learning rate by 10%.
Cond2. For every 3rd epoch, decay your learning rate by 5%.
```

- If you are getting any NaN values(either weigths or loss) while training, you have to terminate your training.
- You have to stop the training if your validation accuracy is not increased in last 2 epochs.
- Use tensorboard for every model and analyse your scalar plots and histograms. (you need to upload the screenshots and write the observations for each model for evaluation)

Data Preprocessing

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

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

```
import pandas as pd
path = "/content/drive/MyDrive/Named_Assignments/Callbacks/data.csv"
data = pd.read_csv(path)
data.head()
```

	f1	f2	label
0	0.450564	1.074305	0.0
1	0.085632	0.967682	0.0
2	0.117326	0.971521	1.0
3	0.982179	-0.380408	0.0
4	-0.720352	0.955850	0.0

If you are running this noteboook in local machine Comment the above two cells and uncomment the below cell.

```
# import pandas as pd
# path_local = "/Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_Assignment
```

```
# data = pd.read_csv(path_local)
# data.head()
```

```
Y= data['label']
X = data.drop('label',axis =1)
print(X.shape)
print(Y.shape)
```

```
(20000, 2)
(20000,)
```

Checking for null or nan values

```
data.isna().sum()
```

```
f1       0
f2       0
label    0
dtype: int64
```

Splitting data into train and test data

```
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, strat
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(16000, 2)
(16000,)
(4000, 2)
(4000,)
```

Data Normalization

Before we apply data to Neural Networks, we must normalize the data. Normalizing the data using z-score normalization.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train['f1'].values.reshape(-1, 1))

f1_train = scaler.transform(X_train['f1'].values.reshape(-1, 1))
f1_test = scaler.transform(X_test['f1'].values.reshape(-1, 1))
print(f1_train.shape,f1_test.shape)
```

```
(16000, 1) (4000, 1)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train['f2'].values.reshape(-1, 1))

f2_train = scaler.transform(X_train['f2'].values.reshape(-1, 1))
f2_test = scaler.transform(X_test['f2'].values.reshape(-1, 1))
print(f2_train.shape,f2_test.shape)
```

```
(16000, 1) (4000, 1)
```

Encoding the output classes into labels rather than float values.

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
encoder.fit(y_train.values)

y_train = encoder.transform(y_train.values)
y_test = encoder.transform(y_test.values)
print(y_train.shape,y_test.shape)
print(y_train[:5])
print(y_test[:5])
```

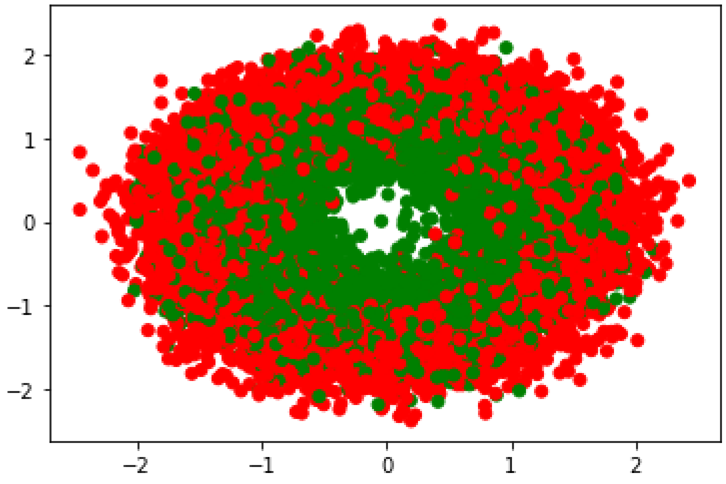
```
(16000,) (4000,)
[1 0 0 1 1]
[1 0 0 1 0]
```

Stacking all features

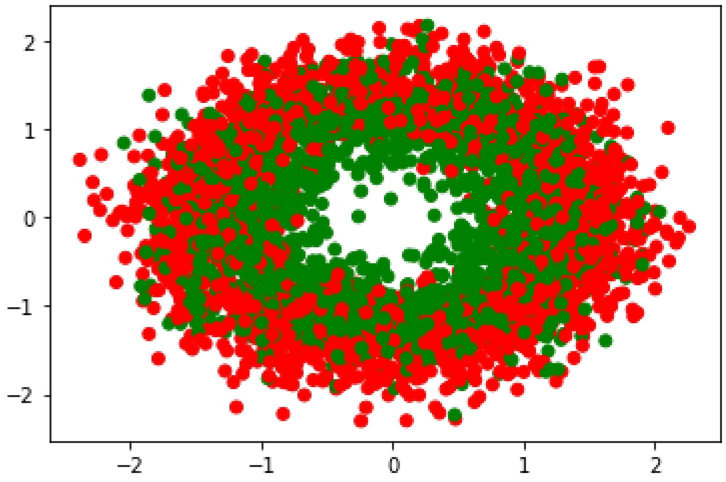
```
import numpy as np

X_train_ftr = np.hstack((f1_train, f2_train))
X_test_ftr = np.hstack((f1_test, f2_test))
y_train = y_train.reshape((-1,1))
y_test = y_test.reshape((-1,1))

print("Final Data matrix")
print(X_train_ftr.shape, y_train.shape)
print(X_test_ftr.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(16000, 2) (16000, 1)
(4000, 2) (4000, 1)
================================================================================
```

```
import matplotlib.pyplot as plt
plt.figure()
color= ['red' if l == 0 else 'green' for l in y_train]
plt.scatter(X_train_ftr[:,0],X_train_ftr[:,1],c =color)
plt.show()
```



```
import matplotlib.pyplot as plt
plt.figure()
color= ['red' if l == 0 else 'green' for l in y_test]
plt.scatter(X_test_ftr[:,0],X_test_ftr[:,1],c =color)
plt.show()
```



Observation :

1. This is highly linearly unseparable data in 2D.

Applying Different models

```
from tensorflow import keras

def create_model(activation_function, weights_intializer):
  model = keras.Sequential()
  model.add(keras.layers.InputLayer(input_shape=(2,),name='input'))
```

```
  model.add(keras.layers.Dense(128, activation = activation_function, kernel_ir
  model.add(keras.layers.Dense(64, activation = activation_function, kernel_ini
  model.add(keras.layers.Dense(32, activation = activation_function, kernel_ini
  model.add(keras.layers.Dense(16, activation = activation_function, kernel_ini
  model.add(keras.layers.Dense(8, activation = activation_function, kernel_init
  model.add(keras.layers.Dense(1, activation = 'sigmoid', kernel_initializer= v

  return model

```

```
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn import metrics

def fptp(k,y_pred,y_actual):
  cnf_matrix=np.zeros((2,2),dtype=int)
  y_label = np.array(list(map(lambda y: (1 if y>=k else 0) , y_pred))).reshape(
  for i in range(2):
    for j in range(2):
      cnf_matrix[i,j] = sum((y_label==i)&(y_actual == j))

  tpr=(cnf_matrix[1,1]/(cnf_matrix[0,1]+cnf_matrix[1,1]))
  fpr=(cnf_matrix[1,0]/(cnf_matrix[1,0]+cnf_matrix[0,0]))
  return tpr,fpr

def calcfprtpr(x,y_actual):
    tpr_array=[]
    fpr_array=[]
    for i in x:
      tpr,fpr=fptp(i[0],x,y_actual)
      tpr_array.append(tpr)
      fpr_array.append(fpr)

    tpr,fpr=fptp(np.max(x)+1,x,y_actual)
    tpr_array.append(tpr)
    fpr_array.append(fpr)

    return tpr_array,fpr_array

def micro_f1_auc(y_pred,y_actual):
  labels = np.unique(y_actual).size
  TP = np.zeros((labels,1))
  FP = np.zeros((labels,1))
  FN = np.zeros((labels,1))
  y_label = np.array(list(map(lambda y: (1 if y>=0.5 else 0) , y_pred))).reshap

  for i in range(labels):
    TP[i,0] = sum((y_label == i)&(y_actual == i))
    FP[i,0] = sum((y_label == i)&(y_actual != i))
    FN[i,0] = sum((y_label != i)&(y_actual == i))

  prec=sum(TP)/(sum(TP)+sum(FP))
  recall=sum(TP)/(sum(TP)+sum(FN))
  mu_F1=(2*prec*recall)/(prec+recall)
```

```
  # mu_F1 = sum(TP)/(sum(TP)+(0.5*(sum(FP)+sum(FN))))
  mu_F1_Sklearn = f1_score(y_actual, y_label,average='micro')

  print("\nMicro F1 by custom_function is ", mu_F1[0])
  print("Micro F1 by sklearn is", mu_F1_Sklearn)
  # fpr, tpr, thresholds = metrics.roc_curve(y_actual, y_pred, pos_label=1)

  tpr_array,fpr_array=calcfprtpr(y_pred,y_actual)
  AUC = np.trapz(sorted(tpr_array), sorted(fpr_array))

  # print("Micro F1 score is ", mu_F1)
  # print("AUC by sklearn auc is ",metrics.auc(fpr, tpr))
  print("AUC by sklearn is ",roc_auc_score(y_actual, y_pred))
  print("AUC by custom_function is",AUC)
```

```
from tensorflow import keras

class print_f1_auc(keras.callbacks.Callback):
  #custom class for printing the micro f1 score and auc score on end of epoch
  def __init__(self,validation_data):
    self.x = validation_data[0]
    self.y = validation_data[1]
    super().__init__()

  def on_epoch_end(self, epoch, logs = None):
    model = self.model
    y_pred = model.predict(self.x)
    micro_f1_auc(y_pred,self.y)


```

```
from tensorflow import keras

class model_save(keras.callbacks.Callback):
  #custom class for saving the model if validation accuracy is improved from pr
  def __init__(self, filepath):
    super().__init__()
    self.prev_val_acc = 0
    self.filepath = filepath
  def on_epoch_end(self, epoch, logs = None):
    model = self.model
    cur_val_acc = logs['val_binary_accuracy']
    if(cur_val_acc>self.prev_val_acc):
      # print("\n Current validation accuracy is {0} \n Prev validation accurac
      print("saving the model as val_acc > prev_val_acc")
      model.save(filepath=self.filepath,overwrite=True)
      self.prev_val_acc = cur_val_acc


```

```
from tensorflow import keras

def epoch_3_scheduler(epoch,lr):
  if((epoch+1)%3==0):
```

```
    print("Setting the learning rate as epoch is multiple of 3")
    lr=0.95*lr
    return lr
  else:
    return lr

class decay_learning(keras.callbacks.Callback):
  #custom class for decaying the learning rate based on conditions.
  def __init__(self):
    super().__init__()
    self.prev_val_acc = 0

  def on_epoch_end(self, epoch, logs = None):
    model = self.model
    cur_val_acc = logs['val_binary_accuracy']
    lr = float(keras.backend.get_value(self.model.optimizer.learning_rate))

    if(cur_val_acc<self.prev_val_acc):
      print("Setting the learning rate as val_acc < prev_val_acc")
      scheduled_lr = lr*0.9
      # Set the value back to the optimizer before new epoch starts
      keras.backend.set_value(self.model.optimizer.lr, scheduled_lr)

    # Call schedule function to get the scheduled learning rate.
    new_lr = epoch_3_scheduler(epoch,lr)
    # Set the value back to the optimizer before new epoch starts
    keras.backend.set_value(self.model.optimizer.lr, new_lr)
    self.prev_val_acc = cur_val_acc



```

```
from tensorflow import keras
import numpy as np
# weights
class stop_train_on_nan(keras.callbacks.Callback):
  #custom class for printing the micro f1 score and auc score on end of epoch
  def __init__(self):
    super().__init__()

  def on_train_batch_end(self, batch, logs = None):
    train_loss = logs['loss']
    weights = self.model.get_weights()
    # print(len(weights))
    if(np.isnan(train_loss)):
      print("found nan values in loss.")
      self.model.stop_training = True
    sam = np.any([np.isnan(np.sum(matrix)) for matrix in weights])
    if(sam):
      print("found nan values in weights or biases")
      self.model.stop_training = True


```

```
from tensorflow import keras

class stop_train_on_valacc(keras.callbacks.Callback):
  #custom class for printing the micro f1 score and auc score on end of epoch
  def __init__(self,patience):
    super().__init__()
    self.patience = 0
    self.wait = patience
    self.prev_val_acc = 0

  def on_epoch_end(self, epoch, logs = None):
    model = self.model
    cur_val_acc = logs['val_binary_accuracy']
    if(cur_val_acc<=self.prev_val_acc):
      self.patience += 1
      if(self.patience == self.wait):
        # print("\n Current validation accuracy is {0} \n Prev validation accu
        print("Stopping the model Training as val_acc <= prev_val_acc for cont
        self.model.stop_training = True
    else :
      self.patience = 0
    self.prev_val_acc = cur_val_acc

```

Model-1

1. Use tanh as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use RandomUniform(0,1) as initilizer.
3. Analyze your output and training process.

```
import os
import datetime
from itertools import combinations
```

```
# path_local = "/Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_Assignment
# model_saving_path = os.path.join(path_local,'model_1', "weights")
# graph_saving_dir = os.path.join(path_local,'model_1', "Tensorboard_graphs")
```

Uncomment the above cell and comment the below cell if you are running this notebook in local machine.

```
path = "/content/drive/MyDrive/Named_Assignments/Callbacks/"
model_saving_path = os.path.join(path,'model_1', "weights")
graph_saving_dir = os.path.join(path,'model_1', "Tensorboard_graphs")
```

```
f1_auc_callback = print_f1_auc(validation_data= [X_test_ftr,y_test])
model_save_callback = model_save(model_saving_path)
decay_learning_callback = decay_learning()
patience = 2
stop_train_on_valacc_callback = stop_train_on_valacc(patience)
stop_train_on_nan_callback = stop_train_on_nan()
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=graph_saving_dir,his

callbacksList = [f1_auc_callback, model_save_callback, decay_learning_callback,
```

```
initializer = keras.initializers.RandomUniform(minval=0, maxval=1)
model = create_model('tanh',initializer)
eta = 0.0001
optimizer = keras.optimizers.SGD(learning_rate=eta,momentum=0.9)
model.compile(optimizer=optimizer,
              metrics = [keras.metrics.BinaryAccuracy()],
              loss=keras.losses.BinaryCrossentropy())
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 128)               384

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 16)                528

 dense_5 (Dense)             (None, 8)                 136

 output (Dense)              (None, 1)                 9

=================================================================
Total params: 11,393
Trainable params: 11,393
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(X_train_ftr, y_train, epochs = 10,validation_data= (X_test_ftr,y_test
```

```
Epoch 1/10
  1/500 [..............................] - ETA: 28:02 - loss: 2.1459 - binary_
498/500 [============================>.] - ETA: 0s - loss: 1.7188 - binary_acc
Micro F1 by custom_function is  0.49325
Micro F1 by sklearn is 0.49325
AUC by sklearn is  0.49349974999999996
AUC by custom_function is 0.49349974999999996
saving the model as val_acc > prev_val_acc
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Named_Assignments/Ca
500/500 [==============================] - 88s 169ms/step - loss: 1.7170 - bir
Epoch 2/10
494/500 [============================>.] - ETA: 0s - loss: 1.0329 - binary_acc
```

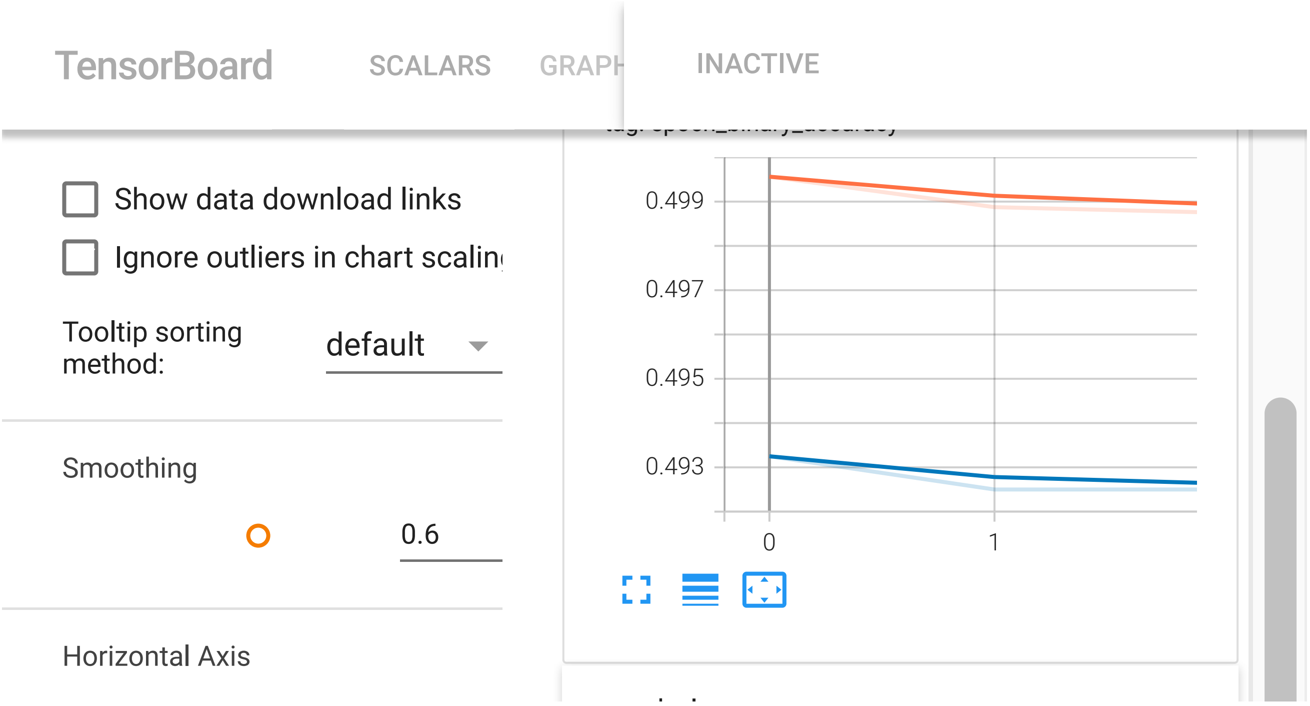
```
Micro F1 by custom_function is  0.4925
Micro F1 by sklearn is 0.4925
AUC by sklearn is  0.4923820000000001
AUC by custom_function is 0.4923820000000001
Setting the learning rate as val_acc < prev_val_acc
500/500 [==============================] - 78s 157ms/step - loss: 1.0307 - bir
Epoch 3/10
493/500 [============================>.] - ETA: 0s - loss: 0.7574 - binary_acc
Micro F1 by custom_function is  0.4925
Micro F1 by sklearn is 0.4925
AUC by sklearn is  0.4923820000000001
AUC by custom_function is 0.4923820000000001
Setting the learning rate as epoch is multiple of 3
Stopping the model Training as val_acc <= prev_val_acc for continuous 2 epochs
500/500 [==============================] - 78s 157ms/step - loss: 0.7568 - bir
<keras.callbacks.History at 0x7f1940049950>
```

```
# there are other ways of doing this: https://www.dlology.com/blog/quick-guide-
%load_ext tensorboard
```

```
# %tensorboard --logdir /Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_As
```

Uncomment the above cell and comment the below cell if you are running this notebook in local machine.

```
%tensorboard --logdir /content/drive/MyDrive/Named_Assignments/Callbacks/model_
```



1. The above model stopped as the validation accuracy is same for 2 continuous epochs.
2. As we have intialized the weights with Random uniform(0,1), weight matrices are almost maintaining the same distribution even after some epochs.
3. From weight and bias distributions we can observe that change in the w,b values is very insignificant.
4. As we have used `tanh` activation function, the updates in weights are insignificant. So we are not getting much improvement in the results.
5. We have achieved these results

- Micro F1 score - **0.4925**
- Area under the curve, AUC - **0.49238**
- Training Accuracy - **0.4988**
- Validation Accuracy - **0.4925**

```
Model-2

 1. Use relu as an activation for every layer except output layer.
 2. use SGD with momentum as optimizer.
 3. use RandomUniform(0,1) as initilizer.
 3. Analyze your output and training process.
```

```
import os
import datetime
from itertools import combinations
```

```
# path_local = "/Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_Assignment
# model_saving_path = os.path.join(path_local,'model_2', "weights")
# graph_saving_dir = os.path.join(path_local,'model_2', "Tensorboard_graphs")
```

Uncomment the above cell and comment the below cell if you are running this notebook in local machine.

```
path = "/content/drive/MyDrive/Named_Assignments/Callbacks/"
model_saving_path = os.path.join(path,'model_2', "weights")
graph_saving_dir = os.path.join(path,'model_2', "Tensorboard_graphs")
```

```
  f1_auc_callback = print_f1_auc(validation_data= [X_test_ftr,y_test])
  model_save_callback = model_save(model_saving_path)
  decay_learning_callback = decay_learning()
  patience = 2
  stop_train_on_valacc_callback = stop_train_on_valacc(patience)
  stop_train_on_nan_callback = stop_train_on_nan()

  tensorboard_callback = keras.callbacks.TensorBoard(log_dir=graph_saving_dir,hi

  callbacksList = [f1_auc_callback, model_save_callback, decay_learning_callback
```

```
initializer = keras.initializers.RandomUniform(minval=0, maxval=1)
model = create_model('relu',initializer)
eta = 0.0001
optimizer = keras.optimizers.SGD(learning_rate=eta,momentum=0.9)
model.compile(optimizer=optimizer,
              metrics = [keras.metrics.BinaryAccuracy()],
              loss=keras.losses.BinaryCrossentropy())
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_1 (Dense)             (None, 128)               384       
                                                                 
 dense_2 (Dense)             (None, 64)                8256      
                                                                 
 dense_3 (Dense)             (None, 32)                2080      
                                                                 
 dense_4 (Dense)             (None, 16)                528       
                                                                 
 dense_5 (Dense)             (None, 8)                 136       
                                                                 
 output (Dense)              (None, 1)                 9         
                                                                 
=================================================================
Total params: 11,393
Trainable params: 11,393
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(X_train_ftr, y_train, epochs = 10,validation_data= (X_test_ftr,y_test
```

```
Epoch 1/10
  1/500 [..............................] - ETA: 3:51 - loss: 257696.3281 - bir
495/500 [============================>.] - ETA: 0s - loss: 660.5337 - binary_a
Micro F1 by custom_function is  0.5
Micro F1 by sklearn is 0.5
AUC by sklearn is  0.5
AUC by custom_function is 0.5
saving the model as val_acc > prev_val_acc
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Named_Assignments/Ca
500/500 [==============================] - 82s 163ms/step - loss: 653.9352 - b
Epoch 2/10
498/500 [============================>.] - ETA: 0s - loss: 0.6932 - binary_acc
Micro F1 by custom_function is  0.5
Micro F1 by sklearn is 0.5
AUC by sklearn is  0.5
AUC by custom_function is 0.5
500/500 [==============================] - 78s 155ms/step - loss: 0.6932 - bir
Epoch 3/10
499/500 [============================>.] - ETA: 0s - loss: 0.6932 - binary_acc
Micro F1 by custom_function is  0.5
Micro F1 by sklearn is 0.5
AUC by sklearn is  0.5
AUC by custom_function is 0.5
Setting the learning rate as epoch is multiple of 3
Stopping the model Training as val_acc <= prev_val_acc for continuous 2 epochs
500/500 [==============================] - 78s 156ms/step - loss: 0.6932 - bir
<keras.callbacks.History at 0x7f18b611fbd0>
```

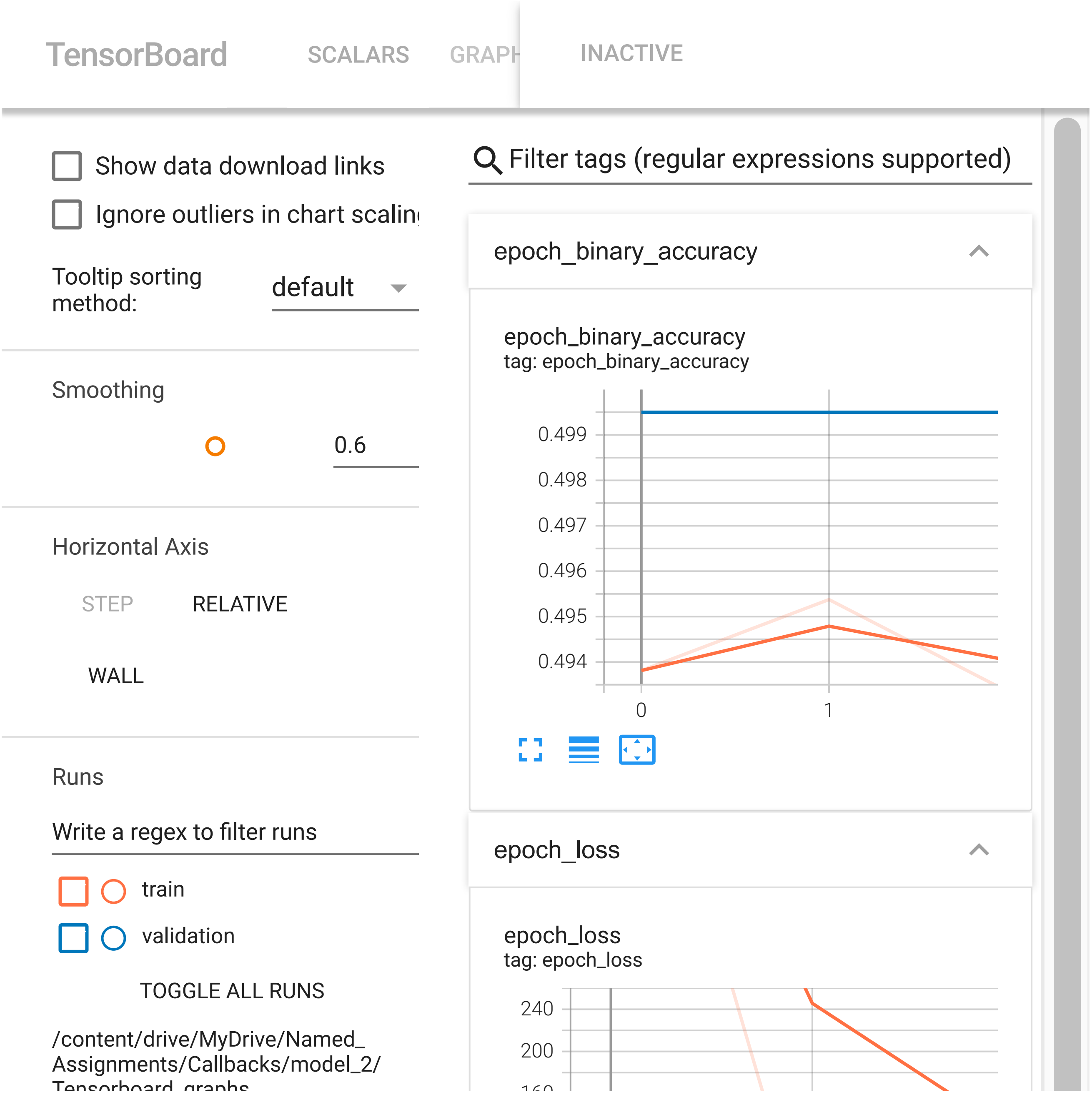
```
# there are other ways of doing this: https://www.dlology.com/blog/quick-guide-
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
# %tensorboard --logdir /Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_As
```

Uncomment the above cell and comment the below cell if you are running this notebook in local machine.

```
%tensorboard --logdir /content/drive/MyDrive/Named_Assignments/Callbacks/model_
```



1. As we have intialized the weights with Random uniform(0,1), weight matrices are almost maintaining the same distribution even after some epochs.
2. Even though we have used `relu` activation function, the updates in weights and biases are insignificant due to random uniform intialization. So we are not getting much improvement in the results.
3. We have achieved these results

- Micro F1 score - **0.5**
- Area under the curve, AUC - **0.5**
- Training Accuracy - **0.4938**
- Validation Accuracy - **0.5**

```
Model-3
```

```
1. Use relu as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use he_uniform() as initilizer.
3. Analyze your output and training process.
```

```
import os
import datetime
from itertools import combinations
```

```
# path_local = "/Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_Assignment
# model_saving_path = os.path.join(path_local,'model_3', "weights")
# graph_saving_dir = os.path.join(path_local,'model_3', "Tensorboard_graphs")
```

Uncomment the above cell and comment the below cell if you are running this notebook in local machine.

```
path = "/content/drive/MyDrive/Named_Assignments/Callbacks/"
model_saving_path = os.path.join(path,'model_3', "weights")
graph_saving_dir = os.path.join(path,'model_3', "Tensorboard_graphs")
```

```
f1_auc_callback = print_f1_auc(validation_data= [X_test_ftr,y_test])
model_save_callback = model_save(model_saving_path)
decay_learning_callback = decay_learning()
patience = 2
stop_train_on_valacc_callback = stop_train_on_valacc(patience)
stop_train_on_nan_callback = stop_train_on_nan()

tensorboard_callback = keras.callbacks.TensorBoard(log_dir=graph_saving_dir,his

callbacksList = [f1_auc_callback, model_save_callback, decay_learning_callback,
```

```
initializer = keras.initializers.HeUniform()
model = create_model('relu',initializer)
eta = 0.0001
optimizer = keras.optimizers.SGD(learning_rate=eta,momentum=0.9)
model.compile(optimizer=optimizer,
              metrics = [keras.metrics.BinaryAccuracy()],
              loss=keras.losses.BinaryCrossentropy())
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 128)               384
```

```
 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 16)                528

 dense_5 (Dense)             (None, 8)                 136

 output (Dense)              (None, 1)                 9

=================================================================
Total params: 11,393
Trainable params: 11,393
Non-trainable params: 0
_________________________________________________________________
```

```
1 model.fit(X_train_ftr, y_train, epochs = 10,validation_data= (X_test_ftr,y_test
```

```
500/500 [==============================] - 79s 158ms/step - loss: 0.6861 -
Epoch 5/10
499/500 [============================>.] - ETA: 0s - loss: 0.6841 - binary_
Micro F1 by custom_function is  0.517

Micro F1 by sklearn is 0.517
AUC by sklearn is  0.6699229999999999
AUC by custom_function is 0.669923
saving the model as val_acc > prev_val_acc
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Named_Assignment
500/500 [==============================] - 78s 157ms/step - loss: 0.6841 -
Epoch 6/10
498/500 [============================>.] - ETA: 0s - loss: 0.6825 - binary_
Micro F1 by custom_function is  0.52525
Micro F1 by sklearn is 0.52525
AUC by sklearn is  0.683200875
AUC by custom_function is 0.683200875
saving the model as val_acc > prev_val_acc
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Named_Assignment
Setting the learning rate as epoch is multiple of 3
500/500 [==============================] - 81s 161ms/step - loss: 0.6824 -
Epoch 7/10
500/500 [==============================] - ETA: 0s - loss: 0.6808 - binary_
Micro F1 by custom_function is  0.53275
Micro F1 by sklearn is 0.53275
AUC by sklearn is  0.70442425
AUC by custom_function is 0.70442425
saving the model as val_acc > prev_val_acc
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Named_Assignment
500/500 [==============================] - 78s 156ms/step - loss: 0.6808 -
Epoch 8/10
494/500 [============================>.] - ETA: 0s - loss: 0.6790 - binary_
Micro F1 by custom_function is  0.5315
Micro F1 by sklearn is 0.5315
AUC by sklearn is  0.7137515
AUC by custom_function is 0.7137515
Setting the learning rate as val_acc < prev_val_acc
500/500 [==============================] - 78s 156ms/step - loss: 0.6791 -
Epoch 9/10
492/500 [============================>.] - ETA: 0s - loss: 0.6775 - binary_
Micro F1 by custom_function is  0.55425
Micro F1 by sklearn is 0.55425
```

```
Micro F1 by sklearn is 0.55425
AUC by sklearn is  0.7150812499999999
AUC by custom_function is 0.71508125
saving the model as val_acc > prev_val_acc
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Named_Assignment
Setting the learning rate as epoch is multiple of 3
500/500 [==============================] - 81s 162ms/step - loss: 0.6776 -
Epoch 10/10
494/500 [============================>.] - ETA: 0s - loss: 0.6760 - binary_
Micro F1 by custom_function is  0.54875
Micro F1 by sklearn is 0.54875
AUC by sklearn is  0.721301
AUC by custom_function is 0.721301
Setting the learning rate as val_acc < prev_val_acc
500/500 [==============================] - 77s 155ms/step - loss: 0.6759 -
<keras.callbacks.History at 0x7f18b71819d0>
```

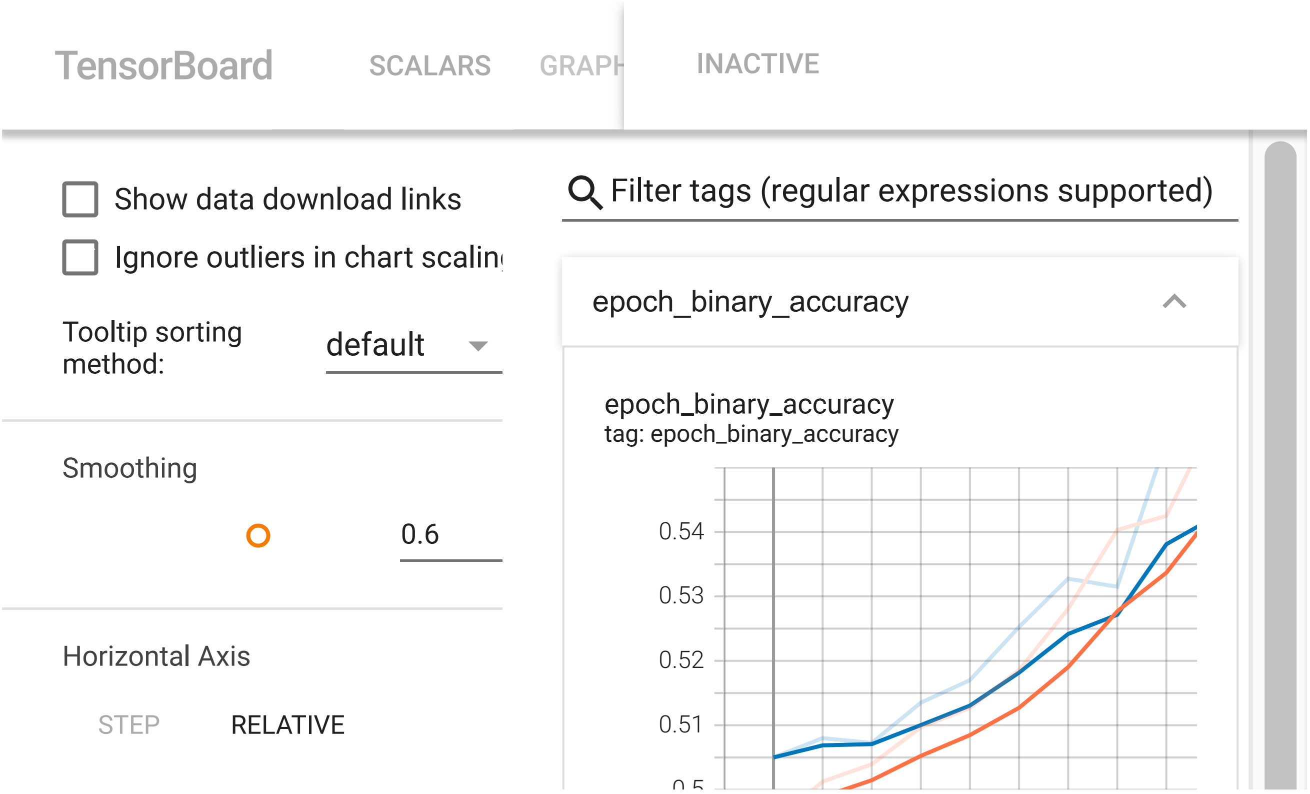
```
# there are other ways of doing this: https://www.dlology.com/blog/quick-guide-
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
# %tensorboard --logdir /Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_As
```

Uncomment the above cell and comment the below cell if you are running this notebook in local machine.

```
%tensorboard --logdir /content/drive/MyDrive/Named_Assignments/Callbacks/model_
```



1. As we have intialized the weights with He uniform, weight matrices are almost maintaining the same distribution even after many epochs.
2. From the distribution of weight matrices and bias vectors, we can observe a significant change in all w,b matirces of all layers except 1st dense layer from first epoch to last epoch.
3. As we have used `relu` activation function, we got more update in the weights and biases and there is an improvemnt shown in the accuracy as well.
4. We have achieved these results

- Micro F1 score - **0.54875**
- Area under the curve, AUC - **0.721301**
- Training Accuracy - **0.5579**
- Validation Accuracy - **0.5487**

From Random Uniform to He uniform intialization, we can obseve these two differences.

1. In Random Uniform intialization we are restricting the values of weight and bias matrices to come from only [0,1] range.
2. But in He uniform we are intializing as sqrt(6/fan_in). As we have different layers and as layers have different fan_ins. We get different ranges of weights and bias for each layer. Due to this model will be able to learn different patterns of the data.
3. Conclusion from the above is we should always try to intialize the weights and biases with different values, so that model can learn difffent patterns of the data.

Model-4

1. Try with any values to get better accuracy/f1 score.

```
import os
import datetime
from itertools import combinations
```

```
# path_local = "/Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_Assignment
# model_saving_path = os.path.join(path_local,'model_4', "weights")
# graph_saving_dir = os.path.join(path_local,'model_4', "Tensorboard_graphs")
```

Uncomment the above cell and comment the below cell if you are running this notebook in local machine.

```
path = "/content/drive/MyDrive/Named_Assignments/Callbacks/"
model_saving_path = os.path.join(path,'model_4', "weights")
graph_saving_dir = os.path.join(path,'model_4', "Tensorboard_graphs")
```

```
f1_auc_callback = print_f1_auc(validation_data= [X_test_ftr,y_test])
model_save_callback = model_save(model_saving_path)
decay_learning_callback = decay_learning()
patience = 2
stop_train_on_valacc_callback = stop_train_on_valacc(patience)
stop_train_on_nan_callback = stop_train_on_nan()

tensorboard_callback = keras.callbacks.TensorBoard(log_dir=graph_saving_dir,his

callbacksList = [f1_auc_callback, model_save_callback, decay_learning_callback,
```

```
initializer = keras.initializers.HeUniform()
model = create_model('relu',initializer)
eta = 0.0001
optimizer = keras.optimizers.Adam(learning_rate=eta)
model.compile(optimizer=optimizer,
              metrics = [keras.metrics.BinaryAccuracy()],
              loss=keras.losses.BinaryCrossentropy())
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_1 (Dense)             (None, 128)               384       
                                                                 
 dense_2 (Dense)             (None, 64)                8256      
                                                                 
 dense_3 (Dense)             (None, 32)                2080      
```

```
 dense_4 (Dense)             (None, 16)                528

 dense_5 (Dense)             (None, 8)                 136

 output (Dense)              (None, 1)                 9

=================================================================
Total params: 11,393
Trainable params: 11,393
Non-trainable params: 0
_________________________________________________________________
```

```
1 model.fit(X_train_ftr, y_train, epochs = 10,validation_data= (X_test_ftr,y_test
```

```
Epoch 5/10
495/500 [============================>.] - ETA: 0s - loss: 0.6516 - binary
Micro F1 by custom_function is  0.657
Micro F1 by sklearn is 0.657
AUC by sklearn is  0.719096375
AUC by custom_function is 0.7190963750000001
saving the model as val_acc > prev_val_acc
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Named_Assignment
500/500 [==============================] - 79s 157ms/step - loss: 0.6515 -
Epoch 6/10
500/500 [==============================] - ETA: 0s - loss: 0.6395 - binary
Micro F1 by custom_function is  0.66575
Micro F1 by sklearn is 0.66575
AUC by sklearn is  0.7326167499999999
AUC by custom_function is 0.73261675
saving the model as val_acc > prev_val_acc
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Named_Assignment
Setting the learning rate as epoch is multiple of 3
500/500 [==============================] - 78s 157ms/step - loss: 0.6395 -
Epoch 7/10
500/500 [==============================] - ETA: 0s - loss: 0.6289 - binary
Micro F1 by custom_function is  0.668
Micro F1 by sklearn is 0.668
AUC by sklearn is  0.7292628750000001
AUC by custom_function is 0.7292628750000001
saving the model as val_acc > prev_val_acc
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Named_Assignment
500/500 [==============================] - 81s 162ms/step - loss: 0.6289 -
Epoch 8/10
499/500 [============================>.] - ETA: 0s - loss: 0.6205 - binary
Micro F1 by custom_function is  0.67625
Micro F1 by sklearn is 0.67625
AUC by sklearn is  0.7369996249999999
AUC by custom_function is 0.7369996249999999
saving the model as val_acc > prev_val_acc
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Named_Assignment
500/500 [==============================] - 81s 162ms/step - loss: 0.6205 -
Epoch 9/10
497/500 [============================>.] - ETA: 0s - loss: 0.6138 - binary
Micro F1 by custom_function is  0.67375
Micro F1 by sklearn is 0.67375
AUC by sklearn is  0.736329
AUC by custom_function is 0.736329
Setting the learning rate as val_acc < prev_val_acc
Setting the learning rate as epoch is multiple of 3
```

```
Setting the learning rate as epoch is multiple of 3
500/500 [==============================] - 78s 156ms/step - loss: 0.6141 -
Epoch 10/10
496/500 [============================>.] - ETA: 0s - loss: 0.6109 - binary_
Micro F1 by custom_function is  0.66825
Micro F1 by sklearn is 0.66825
AUC by sklearn is  0.73635125
AUC by custom_function is 0.73635125
Setting the learning rate as val_acc < prev_val_acc
Stopping the model Training as val_acc <= prev_val_acc for continuous 2 ep
500/500 [==============================] - 78s 156ms/step - loss: 0.6108 -
<keras.callbacks.History at 0x7f18b8abaf10>
```

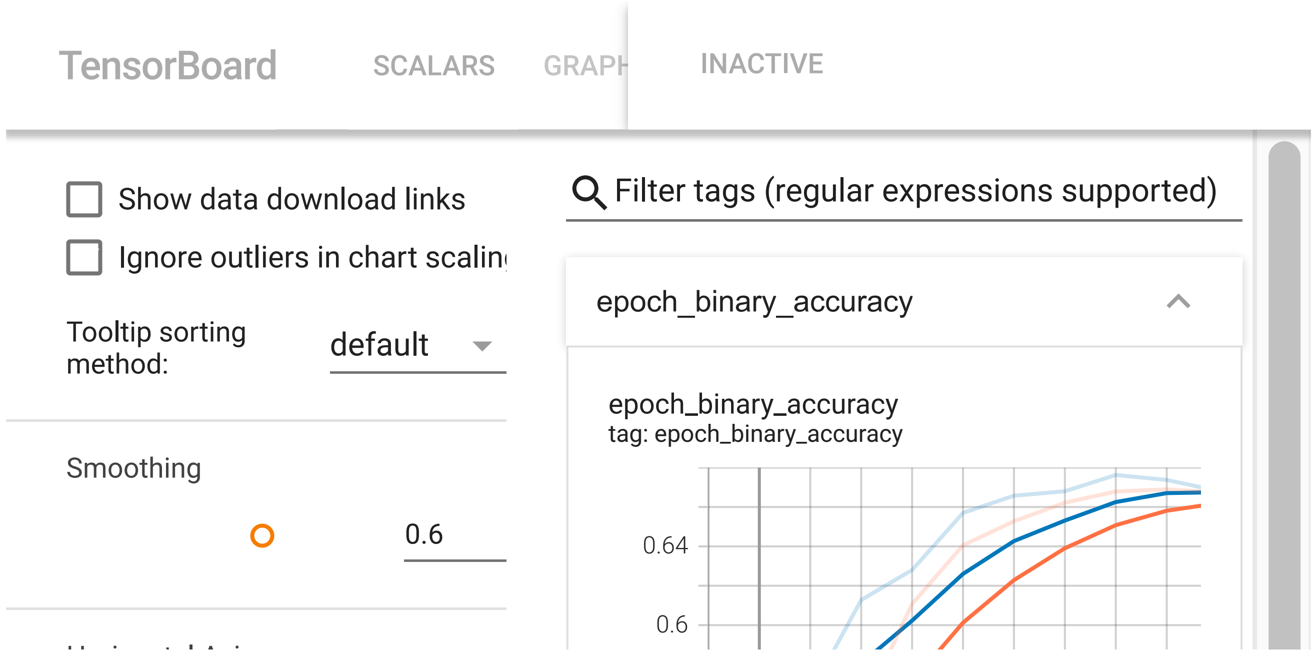
```
# there are other ways of doing this: https://www.dlology.com/blog/quick-guide-
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
# %tensorboard --logdir /Users/yamasanimanoj-kumarreddy/Documents/AAIC/Named_As
```

Uncomment the above cell and comment the below cell if you are running this notebook in local machine.

```
%tensorboard --logdir /content/drive/MyDrive/Named_Assignments/Callbacks/model_
```



1. As we have intialized the weights with He uniform, weight matrices are almost maintaining the same distribution even after many epochs.
2. From the distribution of weight matrices and bias vectors, we can observe a significant change in all w,b matirces of all layers except 1st dense layer from first epoch to last epoch.
3. As we have used `relu` activation function, we got more update in the weights and biases and there is an improvemnt shown in the accuracy as well.
4. We have achieved these results

- Micro F1 score - **0.66825**
- Area under the curve, AUC - **0.73635**
- Training Accuracy - **0.6675**
- Validation Accuracy - **0.6683**

1. From the above we can observe that AUC = 0.73635 which implies model a good classifier.
2. F1 score = 0.66825 also represents model as a good classifier.

From Model_3 to Model_4 we can observe these differences.

1. Even though we have taken care of decreasing learning rate for every 3 epochs, SGD with momentum could not acheive better results than Adam optimizer. This is due to adapting change in the learning rate in Adam optimizer.

Note

Make sure that you are plotting tensorboard plots either in your notebook or you can try to create a pdf file with all the tensorboard screenshots.Please write your analysis of tensorboard results for each model.

To achieve best model's performance, we can observe From above all models that

1. By trying to have different weight and bias values we can make our model to learn different patterns of data.
2. By decreasing the learning rate based on the above conditions, we could achieve better optimization of loss function.
3. Always try with multiple optimizers to get better results.