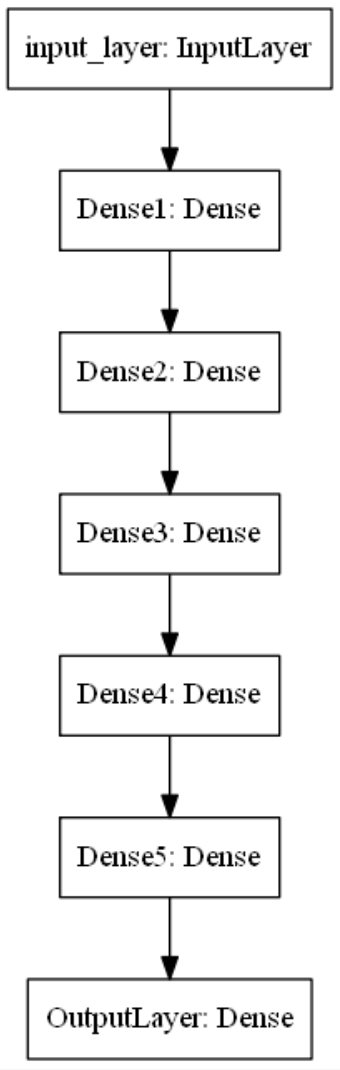
1. Download the data from here

2. Code the model to classify data like below image



3. Write your own callback function, that has to print the micro F1 score and AUC score after each epoch.

4. Save your model at every epoch if your validation accuracy is improved from previous epoch.

5. you have to decay learning based on below conditions
   Cond1. If your validation accuracy at that epoch is less than previous epoch accuracy, you have to decrese the learning rate by 10%.
   Cond2. For every 3rd epoch, decay your learning rate by 5%.

6. If you are getting any NaN values(either weigths or loss) while training, you have to terminate your training.

7. You have to stop the training if your validation accuracy is not increased in last 2 epochs.

8. Use tensorboard for every model and analyse your gradients. (you need to upload the screenshots for each model for evaluation)

9. use cross entropy as loss function

10. Try the architecture params as given below.

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.

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.

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.

Model-4

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

1. Loading dataset and Visualize

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

```
data = pd.read_csv("data.csv")
```

```
data.head()
```

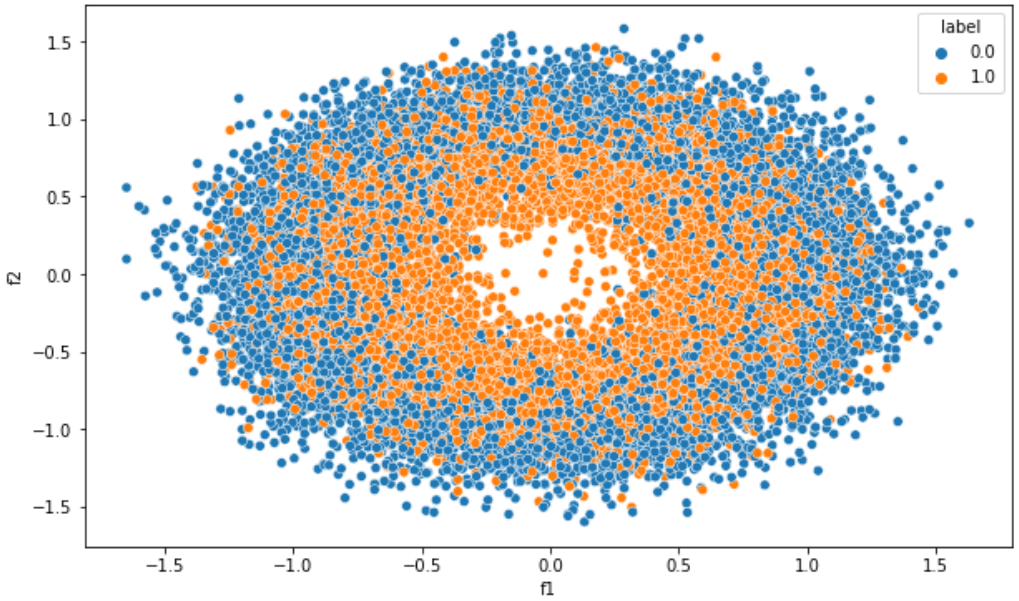
	f1	f2	label
0	0.450564	1.074305	0.0
1	0.085632	0.967682	0.0
2	0.117326	0.971521	1.0
3	0.982179	-0.380408	0.0
4	-0.720352	0.955850	0.0

```
data.describe()
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
plt.figure(figsize=(10,6))
sns.scatterplot(x='f1', y='f2', hue='label', data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f827e4a78d0>
```



Observations

1. data is in circular shape
2. positive datapoints residing towards inside of the circle
3. Negative datapoints residing towards outside of the circle.
4. There is a lot of overlap between the dataset classes.
5. Linear Algorithms can't classify it better
6. Even traditional ML Algorithms will survive to classify it better.
7. DL Algorithms is the best option for this

reshaping the data

```
X, Y = data[['f1','f2']], data['label']
X.shape, Y.shape
```

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

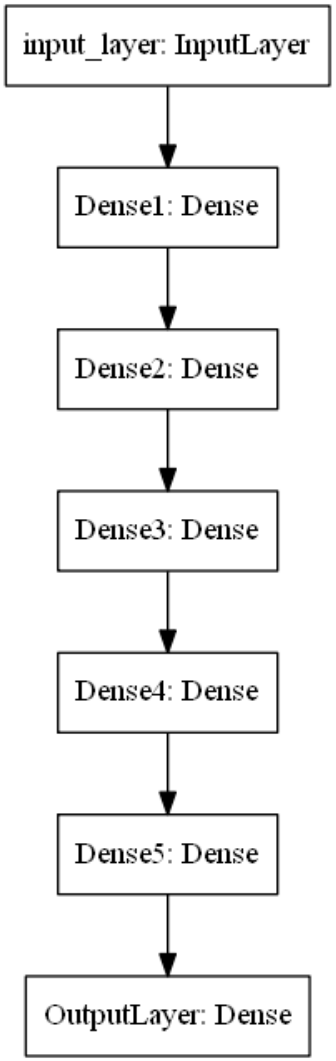
```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2, random_state=42)
```

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

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

2. Code the model to classify data like below image




```
tf.keras.backend.clear_session()
```

```
def create_model():
    return tf.keras.models.Sequential([
        tf.keras.layers.InputLayer(input_shape=(2,)),
        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(84, activation='relu'),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])
```

```
model = create_model()
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               1536
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 84)                10836
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2720
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 179,349
Trainable params: 179,349
Non-trainable params: 0
_________________________________________________________________
```

3. Write you own Callback function

that has to print the micro F1 score and AUC score after each epoch.

```
from sklearn.metrics import f1_score, roc_auc_score
class GetScore(tf.keras.callbacks.Callback):
    def __init__(self, validation_data, batch_size=64):
        super().__init__()
        self.validation_data = validation_data
```

```
        self.batch_size = batch_size

    def on_train_begin(self, logs= {}):
        self.mf1s = []
        self.auc = []

    def on_epoch_end(self, epoch, logs= {}):
#         print("self : ",dir(self))
#         print("self.model : ",dir(self.model))
#         print(self.validation_data)
        y_pred  = (np.asarray(self.model.predict(self.validation_data[0]))).round()

        y_true  = (np.asarray(self.validation_data[1]).reshape((-1,1))).round()

        y_score = (np.asarray(self.model.predict_proba(self.validation_data[0])).reshape((-1,1))).round()
        # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
        val_mf1s = f1_score(y_true, y_pred, average='micro')
        val_auc  = roc_auc_score(y_true, y_score)

        self.mf1s.append(val_mf1s)
        self.auc.append(val_auc)

        print(' - micro f1 score = {} - auc = {}'.format(val_mf1s,val_auc))
        return
```

```
score = GetScore((X_test,y_test))
```

```
# print(score.mf1s, score.auc)
```

```
optimizer = tf.keras.optimizers.Adam(0.1)
model.compile(optimizer=optimizer,
             loss='binary_crossentropy',
             metrics=['accuracy']
             )
```

```
X_test.shape, y_test.shape
```

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

```
model.fit(X_train, y_train,
         validation_data=(X_test,y_test),
         epochs=10,
         batch_size=64,
         callbacks=[score]
         )
```

```
Epoch 1/10
250/250 [==============================] - 2s 7ms/step - loss: 6.6625 - accuracy: 0.5151 - val_loss: 0.6934 - val_accuracy: 0.5008
```

```
 - micro f1 score = 0.50075 - auc = 0.5
Epoch 2/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6940 - accuracy: 0.5041 - val_loss: 0.6937 - val_accuracy: 0.5008
 - micro f1 score = 0.50075 - auc = 0.5
Epoch 3/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6941 - accuracy: 0.5048 - val_loss: 0.6934 - val_accuracy: 0.5008
 - micro f1 score = 0.50075 - auc = 0.5
Epoch 4/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6944 - accuracy: 0.4994 - val_loss: 0.6934 - val_accuracy: 0.4992
 - micro f1 score = 0.49925 - auc = 0.5
Epoch 5/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6939 - accuracy: 0.4967 - val_loss: 0.6939 - val_accuracy: 0.5008
 - micro f1 score = 0.50075 - auc = 0.5
Epoch 6/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6945 - accuracy: 0.4992 - val_loss: 0.6950 - val_accuracy: 0.5008
 - micro f1 score = 0.50075 - auc = 0.5
Epoch 7/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6939 - accuracy: 0.4997 - val_loss: 0.6934 - val_accuracy: 0.4992
 - micro f1 score = 0.49925 - auc = 0.5
Epoch 8/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6944 - accuracy: 0.4970 - val_loss: 0.6935 - val_accuracy: 0.4992
 - micro f1 score = 0.49925 - auc = 0.5
Epoch 9/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6944 - accuracy: 0.4952 - val_loss: 0.6944 - val_accuracy: 0.4992
 - micro f1 score = 0.49925 - auc = 0.5
Epoch 10/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6942 - accuracy: 0.4971 - val_loss: 0.6935 - val_accuracy: 0.5008
 - micro f1 score = 0.50075 - auc = 0.5
<tensorflow.python.keras.callbacks.History at 0x7f826ffd6490>
```

```
print("AUC scores:")
score.auc
```

```
AUC scores:
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
```

```
print('Micro f1 score for each epoch')
score.mf1s
```

```
Micro f1 score for each epoch
[0.50075,
 0.50075,
 0.50075,
 0.49925,
 0.50075,
 0.50075,
 0.49925,
 0.49925,
 0.49925,
 0.50075]
```

4. Save your model at every epoch

If your validation accuracy is improved from previous epoch.

```
from tensorflow.keras.callbacks import ModelCheckpoint
```

```
filepathw = 'model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5'

checkpoint = ModelCheckpoint(
    filepath = filepathw,
    monitor ='val_accuracy',
    verbose = 1,
    save_best_only=True,
    mode = 'max' # max in case of accuracy # min=> loss # auto=> detect automatic
)
```

```
optimizer = tf.keras.optimizers.Adam(0.11)
model.compile(optimizer=optimizer,
             loss='binary_crossentropy',
             metrics=['accuracy']
             )
```

```
model.fit(X_train, y_train,
         validation_data=(X_test,y_test),
         epochs=15,
         batch_size=32,
         callbacks=[checkpoint]
         )
```

```
Epoch 1/15
500/500 [==============================] - 3s 6ms/step - loss: 0.6949 - accuracy: 0.5028 - val_loss: 0.6941 - val_accuracy: 0.5008

Epoch 00001: val_accuracy improved from -inf to 0.50075, saving model to model_save/weights-01-0.5008.hdf5
Epoch 2/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6943 - accuracy: 0.4981 - val_loss: 0.6937 - val_accuracy: 0.4992

Epoch 00002: val_accuracy did not improve from 0.50075
Epoch 3/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6955 - accuracy: 0.4974 - val_loss: 0.6942 - val_accuracy: 0.5008

Epoch 00003: val_accuracy did not improve from 0.50075
Epoch 4/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6942 - accuracy: 0.5098 - val_loss: 0.6931 - val_accuracy: 0.5008

Epoch 00004: val_accuracy did not improve from 0.50075
Epoch 5/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6947 - accuracy: 0.4982 - val_loss: 0.6940 - val_accuracy: 0.5008

Epoch 00005: val_accuracy did not improve from 0.50075
```

```
Epoch 6/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6954 - accuracy: 0.4979 - val_loss: 0.6996 - val_accuracy: 0.5008

Epoch 00006: val_accuracy did not improve from 0.50075
Epoch 7/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6957 - accuracy: 0.5056 - val_loss: 0.6953 - val_accuracy: 0.5008

Epoch 00007: val_accuracy did not improve from 0.50075
Epoch 8/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6965 - accuracy: 0.4956 - val_loss: 0.6961 - val_accuracy: 0.5008

Epoch 00008: val_accuracy did not improve from 0.50075
Epoch 9/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6960 - accuracy: 0.4916 - val_loss: 0.6931 - val_accuracy: 0.5008

Epoch 00009: val_accuracy did not improve from 0.50075
Epoch 10/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6958 - accuracy: 0.4912 - val_loss: 0.6932 - val_accuracy: 0.4992

Epoch 00010: val_accuracy did not improve from 0.50075
Epoch 11/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6946 - accuracy: 0.5103 - val_loss: 0.6957 - val_accuracy: 0.5008

Epoch 00011: val_accuracy did not improve from 0.50075
Epoch 12/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6945 - accuracy: 0.5067 - val_loss: 0.6962 - val_accuracy: 0.4992

Epoch 00012: val_accuracy did not improve from 0.50075
Epoch 13/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6961 - accuracy: 0.4940 - val_loss: 0.6949 - val_accuracy: 0.5008

Epoch 00013: val_accuracy did not improve from 0.50075
Epoch 14/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6944 - accuracy: 0.5080 - val_loss: 0.7029 - val_accuracy: 0.5008

Epoch 00014: val_accuracy did not improve from 0.50075
Epoch 15/15
500/500 [==============================] - 3s 5ms/step - loss: 0.6974 - accuracy: 0.4903 - val_loss: 0.6936 - val_accuracy: 0.4992
```

5. Decay learning

you have to decay learning based on below conditions

cond1. If your validation accuracy at that epoch is less than previous epoch accuracy, you have to decrease the learning rate by 10%.
cond2. For every 3rd epoch, decay your learning rate by 5%.

```
from tensorflow.keras.callbacks import LearningRateScheduler
```

```
class decayLR(tf.keras.callbacks.Callback):
    """
    This is custom callback class to implement decay the Learning Rate.
    Author: Muhammad Iqbal Bajmi @AppliedAICourse.com
    """
    def on_train_begin(self, logs={}):
        self.valid_acc = {'accuracy':[]}
    def on_epoch_begin(self, epoch, logs={}):
        print(epoch)
        print("Learning rate is : {}".format(float(self.model.optimizer.learning_rate)))
        if epoch>1:
            if (epoch+1)%3==0:
                lr = self.model.optimizer.learning_rate
                self.model.optimizer.learning_rate = self.model.optimizer.learning_rate - (lr*0.05)
    def on_epoch_end(self, epoch, logs={}):
        self.valid_acc['accuracy'].append(logs.get('val_accuracy'))
        if epoch>0: # because epoch starts from 0
            if self.valid_acc['accuracy'][epoch] < self.valid_acc['accuracy'][epoch-1]:
                lr = self.model.optimizer.learning_rate
                self.model.optimizer.learning_rate = self.model.optimizer.learning_rate - (lr*0.1)
```

```
decaylr = decayLR()
```

```
optimizer = tf.keras.optimizers.Adam(0.1)
model.compile(optimizer=optimizer,
             loss='binary_crossentropy',
             metrics=['accuracy']
             )
```

```
model.fit(X_train, y_train,
         validation_data=(X_test,y_test),
         epochs=20,
         batch_size=64,
         callbacks=[decaylr]
         )
```

```
Epoch 1/20
0
Learning rate is : 0.10000000149011612
250/250 [==============================] - 2s 7ms/step - loss: 0.6940 - accuracy: 0.5157 - val_loss: 0.6939 - val_accuracy: 0.4992
Epoch 2/20
1
Learning rate is : 0.10000000149011612
250/250 [==============================] - 2s 6ms/step - loss: 0.6945 - accuracy: 0.5081 - val_loss: 0.6933 - val_accuracy: 0.4992
Epoch 3/20
2
Learning rate is : 0.10000000149011612
250/250 [==============================] - 2s 7ms/step - loss: 0.6942 - accuracy: 0.5039 - val_loss: 0.6934 - val_accuracy: 0.5008
Epoch 4/20
3
Learning rate is : 0.0949999988079071
```

```
250/250 [==============================] - 2s 6ms/step - loss: 0.6944 - accuracy: 0.5080 - val_loss: 0.6931 - val_accuracy: 0.5008
Epoch 5/20
4
Learning rate is : 0.0949999988079071
250/250 [==============================] - 2s 6ms/step - loss: 0.6947 - accuracy: 0.4991 - val_loss: 0.6932 - val_accuracy: 0.5008
Epoch 6/20
5
Learning rate is : 0.0949999988079071
250/250 [==============================] - 2s 6ms/step - loss: 0.6939 - accuracy: 0.5017 - val_loss: 0.6980 - val_accuracy: 0.5008
Epoch 7/20
6
Learning rate is : 0.09025000035762787
250/250 [==============================] - 2s 7ms/step - loss: 0.6949 - accuracy: 0.4985 - val_loss: 0.6932 - val_accuracy: 0.4992
Epoch 8/20
7
Learning rate is : 0.08122500032186508
250/250 [==============================] - 2s 6ms/step - loss: 0.6941 - accuracy: 0.4960 - val_loss: 0.6942 - val_accuracy: 0.5008
Epoch 9/20
8
Learning rate is : 0.08122500032186508
250/250 [==============================] - 2s 6ms/step - loss: 0.6941 - accuracy: 0.4924 - val_loss: 0.6932 - val_accuracy: 0.4992
Epoch 10/20
9
Learning rate is : 0.06944737583398819
250/250 [==============================] - 2s 6ms/step - loss: 0.6941 - accuracy: 0.4949 - val_loss: 0.6939 - val_accuracy: 0.4992
Epoch 11/20
10
Learning rate is : 0.06944737583398819
250/250 [==============================] - 2s 6ms/step - loss: 0.6942 - accuracy: 0.4885 - val_loss: 0.6932 - val_accuracy: 0.4992
Epoch 12/20
11
Learning rate is : 0.06944737583398819
250/250 [==============================] - 2s 6ms/step - loss: 0.6940 - accuracy: 0.5051 - val_loss: 0.6932 - val_accuracy: 0.5008
Epoch 13/20
12
Learning rate is : 0.06597501039505005
250/250 [==============================] - 2s 6ms/step - loss: 0.6938 - accuracy: 0.4930 - val_loss: 0.6935 - val_accuracy: 0.4992
Epoch 14/20
13
Learning rate is : 0.059377510100603104
250/250 [==============================] - 2s 6ms/step - loss: 0.6938 - accuracy: 0.4935 - val_loss: 0.6952 - val_accuracy: 0.5008
Epoch 15/20
14
Learning rate is : 0.059377510100603104
```

6.Terminate training if getting NaN

6. If you are getting any NaN values(either weights or loss) while training, you have to terminate your training.

```
from tensorflow.keras.callbacks import TerminateOnNaN
```

```
terminate = TerminateOnNaN()
```

```
class TerminateNaN(tf.keras.callbacks.Callback):
    def __init__(self):
        self.we = 0
    def on_epoch_end(self, epoch, logs={}):
        import numpy as np
        loss = logs.get('loss')
        w = self.model.get_weights()
        w = np.array(w)
        self.we = w
        if loss is not None:
            if np.isnan(loss) or np.isinf(loss):
                print("Invalid loss and terminated at epoch {}".foramt(epoch))
                self.model.stop_training = True

        for i in range(6):
            layer   = self.model.layers[i]
            weights = layer.get_weights()[0]
            biases  = layer.get_weights()[1]

            if np.isnan(weights).any() or np.isinf(weights).any():
                print("Invalid weights and terminated at epoch ={}".format(epoch))
                self.model.stop_training = True
```

```
terminate = TerminateNaN()
```

```
optimizer = tf.keras.optimizers.Adam(0.1)
model.compile(optimizer=optimizer,
             loss='binary_crossentropy',
             metrics=['accuracy']
             )
```

```
model.fit(X_train, y_train,
         validation_data=(X_test,y_test),
         epochs=10,
         batch_size=64,
         callbacks=[terminate]
         )
```

```
Epoch 1/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6945 - accuracy: 0.5010 - val_loss: 0.6945 - val_accuracy: 0.4992
Epoch 2/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6951 - accuracy: 0.4875 - val_loss: 0.6960 - val_accuracy: 0.5008
Epoch 3/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6960 - accuracy: 0.4984 - val_loss: 0.6967 - val_accuracy: 0.4992
Epoch 4/10
```

```
250/250 [==============================] - 2s 6ms/step - loss: 0.6946 - accuracy: 0.4932 - val_loss: 0.6932 - val_accuracy: 0.4992
Epoch 5/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6945 - accuracy: 0.5006 - val_loss: 0.6947 - val_accuracy: 0.5008
Epoch 6/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6944 - accuracy: 0.5030 - val_loss: 0.6940 - val_accuracy: 0.4992
Epoch 7/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6947 - accuracy: 0.5020 - val_loss: 0.6935 - val_accuracy: 0.5008
Epoch 8/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6944 - accuracy: 0.4985 - val_loss: 0.6943 - val_accuracy: 0.4992
Epoch 9/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6942 - accuracy: 0.4972 - val_loss: 0.7005 - val_accuracy: 0.5008
Epoch 10/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6953 - accuracy: 0.4955 - val_loss: 0.6934 - val_accuracy: 0.5008
<tensorflow.python.keras.callbacks.History at 0x7f826ebc6a50>
```

7. Stop training if validation_accuracy not improved

You have to stop the training if your validation accuracy is not increased in last 2 epochs.

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
earlystop = EarlyStopping(
    monitor  = 'val_accuracy',
    mode     = 'max',
    patience = 2
)
```

```
optimizer = tf.keras.optimizers.Adam(0.1)
model.compile(optimizer=optimizer,
             loss='binary_crossentropy',
             metrics=['accuracy']
             )
```

```
model.fit(X_train, y_train,
         validation_data=(X_test,y_test),
         epochs=10,
         batch_size=64,
         callbacks=[earlystop]
         )
```

```
Epoch 1/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6949 - accuracy: 0.4940 - val_loss: 0.6952 - val_accuracy: 0.5008
Epoch 2/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6941 - accuracy: 0.5007 - val_loss: 0.6932 - val_accuracy: 0.4992
Epoch 3/10
```

```
250/250 [==============================] - 2s 6ms/step - loss: 0.6948 - accuracy: 0.5008 - val_loss: 0.6932 - val_accuracy: 0.4992
<tensorflow.python.keras.callbacks.History at 0x7f826e9c54d0>
```

8. Use tensorboard

```
import datetime
import tensorflow as tf
```

```
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=True)
```

```
!rm -rf ./logs/
```

```
optimizer = tf.keras.optimizers.Adam(0.1)
model.compile(optimizer=optimizer,
              loss='binary_crossentropy',
              metrics=['accuracy']
              )
```

```
model.fit(X_train, y_train,
          validation_data=(X_test,y_test),
          epochs=10,
          batch_size=64,
          callbacks=[tensorboard_callback]
          )
```

```
Epoch 1/10
250/250 [==============================] - 3s 9ms/step - loss: 0.6943 - accuracy: 0.5023 - val_loss: 0.6937 - val_accuracy: 0.5008
Epoch 2/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6946 - accuracy: 0.5000 - val_loss: 0.6956 - val_accuracy: 0.4992
Epoch 3/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6955 - accuracy: 0.4922 - val_loss: 0.6933 - val_accuracy: 0.5008
Epoch 4/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6947 - accuracy: 0.4915 - val_loss: 0.6950 - val_accuracy: 0.5008
Epoch 5/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6951 - accuracy: 0.4926 - val_loss: 0.6980 - val_accuracy: 0.4992
Epoch 6/10
250/250 [==============================] - 2s 7ms/step - loss: 0.6953 - accuracy: 0.5016 - val_loss: 0.6937 - val_accuracy: 0.4992
Epoch 7/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6955 - accuracy: 0.4931 - val_loss: 0.6933 - val_accuracy: 0.5008
Epoch 8/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6942 - accuracy: 0.5022 - val_loss: 0.6936 - val_accuracy: 0.5008
Epoch 9/10
```

```
250/250 [==============================] - 2s 6ms/step - loss: 0.6943 - accuracy: 0.5064 - val_loss: 0.6952 - val_accuracy: 0.4992
Epoch 10/10
250/250 [==============================] - 2s 6ms/step - loss: 0.6951 - accuracy: 0.4945 - val_loss: 0.6933 - val_accuracy: 0.4992
<tensorflow.python.keras.callbacks.History at 0x7f826e93d8d0>
```

```
%load_ext tensorboard
```

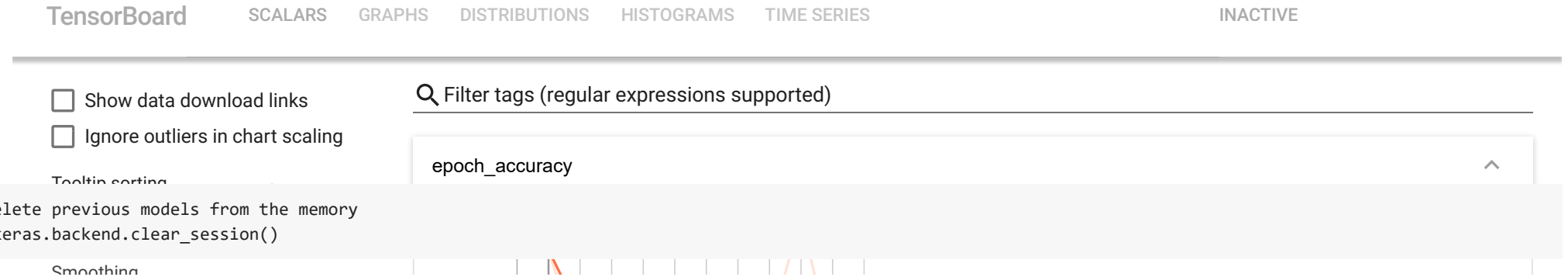
```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
!kill 1179
```

```
/bin/bash: line 0: kill: (1179) - No such process
```

```
%tensorboard --logdir logs/fit/
```

```
Reusing TensorBoard on port 6006 (pid 1135), started 5:24:17 ago. (Use '!kill 1135' to kill it.)
```




```
# delete previous models from the memory
tf.keras.backend.clear_session()
```

10. Try different architectures

```
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 initializer.
4. Analyze your output and training process.
```

```
# defining model 1
initializer = tf.keras.initializers.RandomUniform(0,1)
def create_model1():
    return tf.keras.models.Sequential([
        tf.keras.layers.Dense(512,
                              input_shape=(2,),
                              activation='tanh',
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(256,
                              activation='tanh',
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(128,
                              activation='tanh',
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(84,
                              activation='tanh',
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(32,
                              activation='tanh',
                              kernel_initializer=initializer
                             ),
```

```
                            );
        tf.keras.layers.Dense(1,
                              activation='sigmoid',
                              kernel_initializer=initializer
                             )
    ])
```

```
model1 = create_model1()
```

```
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               1536
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 84)                10836
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2720
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 179,349
Trainable params: 179,349
Non-trainable params: 0
_________________________________________________________________
```

```
# remove previous logs
! rm -rf logs
```

```
score = GetScore((X_test, y_test))# => Mf1s, auc score

filepathw = 'model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5'
checkpoint = ModelCheckpoint(
    filepath = filepathw,
    monitor ='val_accuracy',
    verbose = 1,
    save_best_only=True,
    mode = 'max' # max in case of accuracy # min=> loss # auto=> detect automatic
)
decay_lr = decayLR() # => decaying learning rate
terminate_nan = TerminateNaN()# => terminating on NaN
earlystop = EarlyStopping(
    monitor  = 'val_accuracy',
```

```
    mode     = 'max',
    patience = 2
)
# stop_training = StopTraining() # stop training on some conditions ,,,, Not gonna use custom stop training instead use Early
# save_model = SaveModel() # to save model=>  ,,,, Not gonna use custom save_model instead use Checkpoint

log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=True)
```

```
callback_list = [score, checkpoint, decay_lr, terminate_nan, earlystop, tensorboard_callback]
```

```
# remove all previous saved models
!rm -rf model_save # comment this if you don't want to remove previous saved models
```

```
import warnings
```

```
warnings.filterwarnings('ignore')
```

```
optimizer = tf.keras.optimizers.SGD(learning_rate=0.1, momentum=0.9)
model1.compile(optimizer=optimizer,
            loss='binary_crossentropy',
            metrics=['accuracy']
            )
model1.fit(X_train, y_train,
         validation_data=(X_test,y_test),
         epochs=20,
         batch_size=64,
         callbacks=[callback_list]
         )
```

```
Epoch 1/20
0
Learning rate is : 0.10000000149011612
  3/250 [..............................] - ETA: 10s - loss: 9.1194 - accuracy: 0.4505 WARNING:tensorflow:Callback method `on_train_batch_end` is slow compare
250/250 [==============================] - 2s 7ms/step - loss: 1.8425 - accuracy: 0.4981 - val_loss: 0.6952 - val_accuracy: 0.5008
 - micro f1 score = 0.50075 - auc = 0.5

Epoch 00001: val_accuracy improved from -inf to 0.50075, saving model to model_save/weights-01-0.5008.hdf5
Epoch 2/20
1
Learning rate is : 0.10000000149011612
250/250 [==============================] - 1s 6ms/step - loss: 0.7153 - accuracy: 0.5102 - val_loss: 0.7026 - val_accuracy: 0.5113
 - micro f1 score = 0.51125 - auc = 0.5112462753041195

Epoch 00002: val_accuracy improved from 0.50075 to 0.51125, saving model to model_save/weights-02-0.5113.hdf5
Epoch 3/20
2
Learning rate is : 0.10000000149011612
250/250 [==============================] - 1s 6ms/step - loss: 0.7435 - accuracy: 0.5039 - val_loss: 0.6939 - val_accuracy: 0.5113
```

```
 - micro f1 score = 0.51125 - auc = 0.5112462753041195

Epoch 00003: val_accuracy did not improve from 0.51125
Epoch 4/20
3
Learning rate is : 0.0949999988079071
250/250 [==============================] - 1s 6ms/step - loss: 0.7456 - accuracy: 0.5055 - val_loss: 0.7673 - val_accuracy: 0.4888
 - micro f1 score = 0.48875 - auc = 0.4887537246958806

Epoch 00004: val_accuracy did not improve from 0.51125
<tensorflow.python.keras.callbacks.History at 0x7f827ea00610>
```

Summary of model 1:

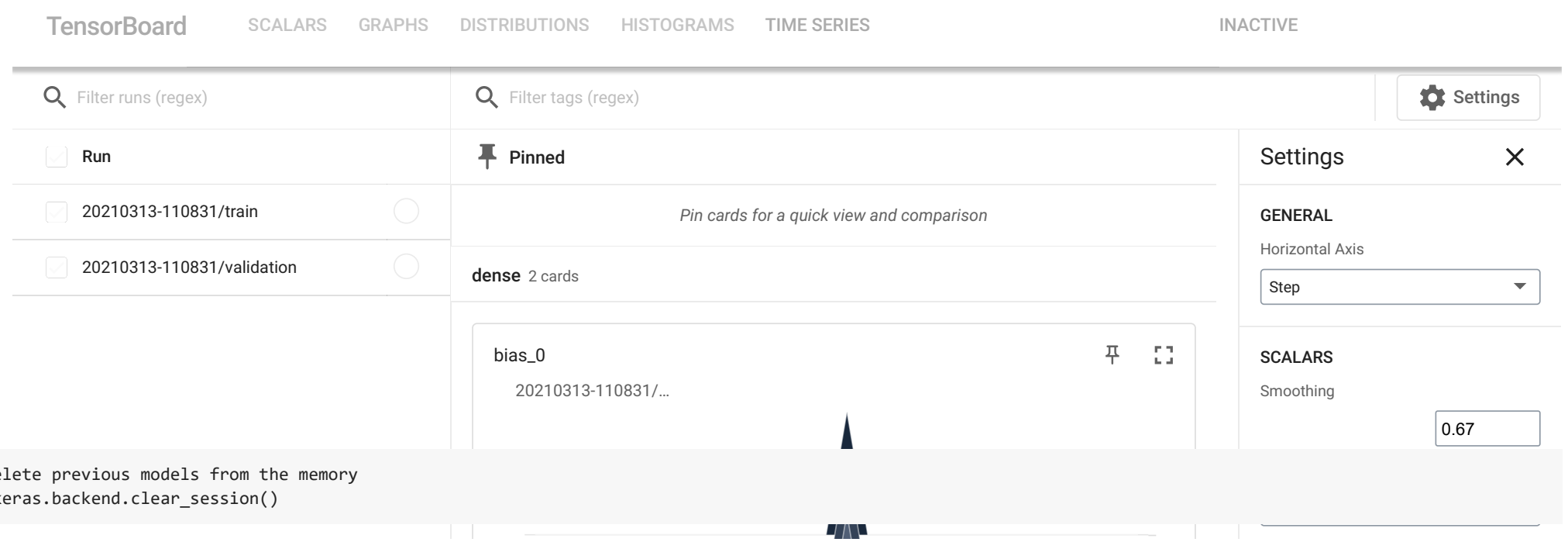
1. Model 1 ran upto 4 epochs
2. best Validation accuracy : 0.5113
3. best micro f1 score : 0.5112

```
%load_ext tensorboard
```

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

```
%tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6007 (pid 1489), started 5:08:56 ago. (Use '!kill 1489' to kill it.)
```




```
# delete previous models from the memory
tf.keras.backend.clear_session()
```

```
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 initializer.
4. Analyze your output and training process.
```

```
# defining model 2
initializer = tf.keras.initializers.RandomUniform(0,1)
def create_model2():
    return tf.keras.models.Sequential([
        tf.keras.layers.Dense(512,
                              input_shape=(2,),
                              activation='relu',
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(256,
                              activation='relu',
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(128,
                              activation='relu',
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(84,
```

```
                            activation='relu',
                            kernel_initializer=initializer
                           ),
        tf.keras.layers.Dense(32,
                            activation='relu',
                            kernel_initializer=initializer
                           ),
        tf.keras.layers.Dense(1,
                            activation='sigmoid',
                            kernel_initializer=initializer
                           )
    ])
```

```
model2 = create_model2() # creating model
model2.summary() # model summary
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               1536
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 84)                10836
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2720
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 179,349
Trainable params: 179,349
Non-trainable params: 0
_________________________________________________________________
```

```
# remove previous logs
!rm -rf logs
# remove previous saved files
!rm -rf model_save

# initializing optimizer
optimizer = tf.keras.optimizers.SGD(learning_rate=0.1, momentum=0.9)
# compiling the model
model2.compile(optimizer=optimizer,
             loss='binary_crossentropy',
             metrics=['accuracy']
             )
# Fit the model
```

```
# Fit the model
model2.fit(X_train, y_train,
         validation_data=(X_test,y_test),
         epochs=20,
         batch_size=64,
         callbacks=[callback_list]
         )
```

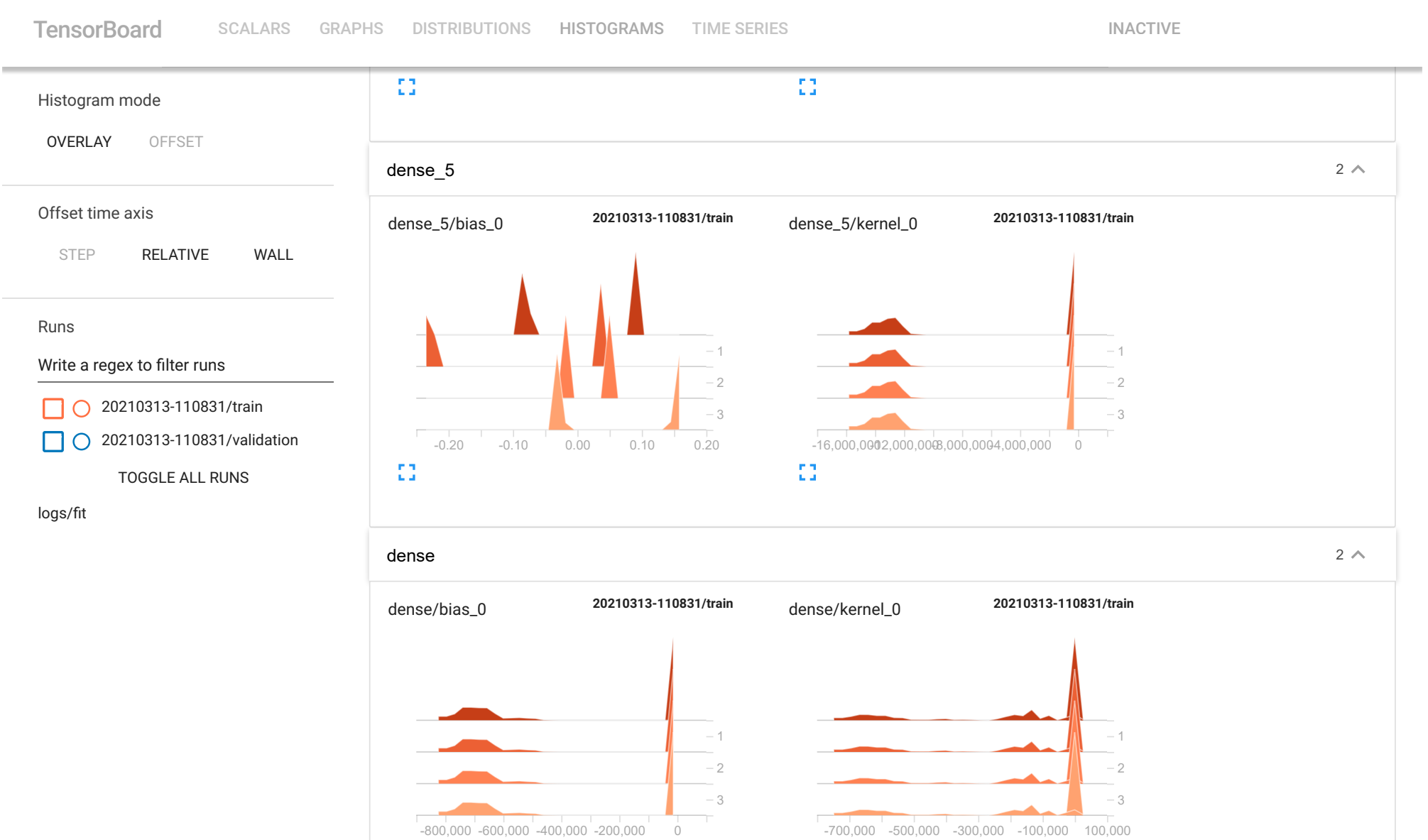
```
Epoch 1/20
0
Learning rate is : 0.10000000149011612
  3/250 [..............................] - ETA: 10s - loss: 121816249.3333 - accuracy: 0.5321 WARNING:tensorflow:Callback method `on_train_batch_end` is slow
250/250 [==============================] - 2s 7ms/step - loss: 4848125.8745 - accuracy: 0.5032 - val_loss: 0.6943 - val_accuracy: 0.4992
 - micro f1 score = 0.49925 - auc = 0.5

Epoch 00001: val_accuracy did not improve from 0.51125
Epoch 2/20
1
Learning rate is : 0.10000000149011612
250/250 [==============================] - 2s 6ms/step - loss: 0.6935 - accuracy: 0.5110 - val_loss: 0.6995 - val_accuracy: 0.5008
 - micro f1 score = 0.50075 - auc = 0.5

Epoch 00002: val_accuracy did not improve from 0.51125
Epoch 3/20
2
Learning rate is : 0.10000000149011612
250/250 [==============================] - 2s 6ms/step - loss: 0.6938 - accuracy: 0.5075 - val_loss: 0.6935 - val_accuracy: 0.4992
 - micro f1 score = 0.49925 - auc = 0.5

Epoch 00003: val_accuracy did not improve from 0.51125
Epoch 4/20
3
Learning rate is : 0.08550000190734863
250/250 [==============================] - 2s 6ms/step - loss: 0.6939 - accuracy: 0.5003 - val_loss: 0.6932 - val_accuracy: 0.5008
 - micro f1 score = 0.50075 - auc = 0.5

Epoch 00004: val_accuracy did not improve from 0.51125
<tensorflow.python.keras.callbacks.History at 0x7f82776a6c90>
```

Summary of model 2:

1. Model 2 ran upto 4 epochs
2. Validation accuracy : 0.51125
3. micro f1 score : 0.50075

```
%tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6007 (pid 1489), started 5:19:42 ago. (Use '!kill 1489' to kill it.)
```




```
# clear all sessions
tf.keras.backend.clear_session()
```

Model-3

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

```
# defining model 3
initializer = tf.keras.initializers.he_uniform()
activation = 'relu'
def create_model3():
    return tf.keras.models.Sequential([
        tf.keras.layers.Dense(512,
                              input_shape=(2,),
                              activation=activation,
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(256,
                              activation=activation,
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(128,
                              activation=activation,
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(84,
                              activation=activation,
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(32,
                              activation=activation,
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(1,
                              activation='sigmoid',
                              kernel_initializer=initializer
                             )
    ])

# creating an object of model 3
model3 = create_model3()
model3.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               1536
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 84)                10836
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2720
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
```

```
Total params: 179,349
Trainable params: 179,349
Non-trainable params: 0
_________________________________________________________________
```

```
# remove previous logs
!rm -rf logs
# remove previous saved files
!rm -rf model_save

# initializing optimizer
optimizer = tf.keras.optimizers.SGD(learning_rate=0.1, momentum=0.9)
# compiling the model
model3.compile(optimizer=optimizer,
             loss='binary_crossentropy',
             metrics=['accuracy']
             )
# Fit the model
model3.fit(X_train, y_train,
         validation_data=(X_test,y_test),
         epochs=20,
         batch_size=64,
         callbacks=[callback_list]
         )
```

```
Epoch 1/20
0
Learning rate is : 0.100000000149011612
  3/250 [..............................] - ETA: 10s - loss: 0.8119 - accuracy: 0.4861 WARNING:tensorflow:Callback method `on_train_batch_end` is slow compare
250/250 [==============================] - 2s 7ms/step - loss: 0.6923 - accuracy: 0.5687 - val_loss: 0.6534 - val_accuracy: 0.6018
 - micro f1 score = 0.60175 - auc = 0.6013482280335131

Epoch 00001: val_accuracy improved from 0.51125 to 0.60175, saving model to model_save/weights-01-0.6018.hdf5
Epoch 2/20
1
Learning rate is : 0.100000000149011612
250/250 [==============================] - 1s 6ms/step - loss: 0.6434 - accuracy: 0.6344 - val_loss: 0.7136 - val_accuracy: 0.4997
 - micro f1 score = 0.49975 - auc = 0.5004992511233151

Epoch 00002: val_accuracy did not improve from 0.60175
Epoch 3/20
2
Learning rate is : 0.090000000357627869
250/250 [==============================] - 2s 6ms/step - loss: 0.6358 - accuracy: 0.6316 - val_loss: 0.6393 - val_accuracy: 0.6553
 - micro f1 score = 0.65525 - auc = 0.6553298494921613

Epoch 00003: val_accuracy improved from 0.60175 to 0.65525, saving model to model_save/weights-03-0.6553.hdf5
Epoch 4/20
3
Learning rate is : 0.085500000190734863
250/250 [==============================] - 1s 6ms/step - loss: 0.6205 - accuracy: 0.6572 - val_loss: 0.6256 - val_accuracy: 0.6587
 - micro f1 score = 0.65875 - auc = 0.6586663569993032

Epoch 00004: val_accuracy improved from 0.65525 to 0.65875, saving model to model_save/weights-04-0.6587.hdf5
```

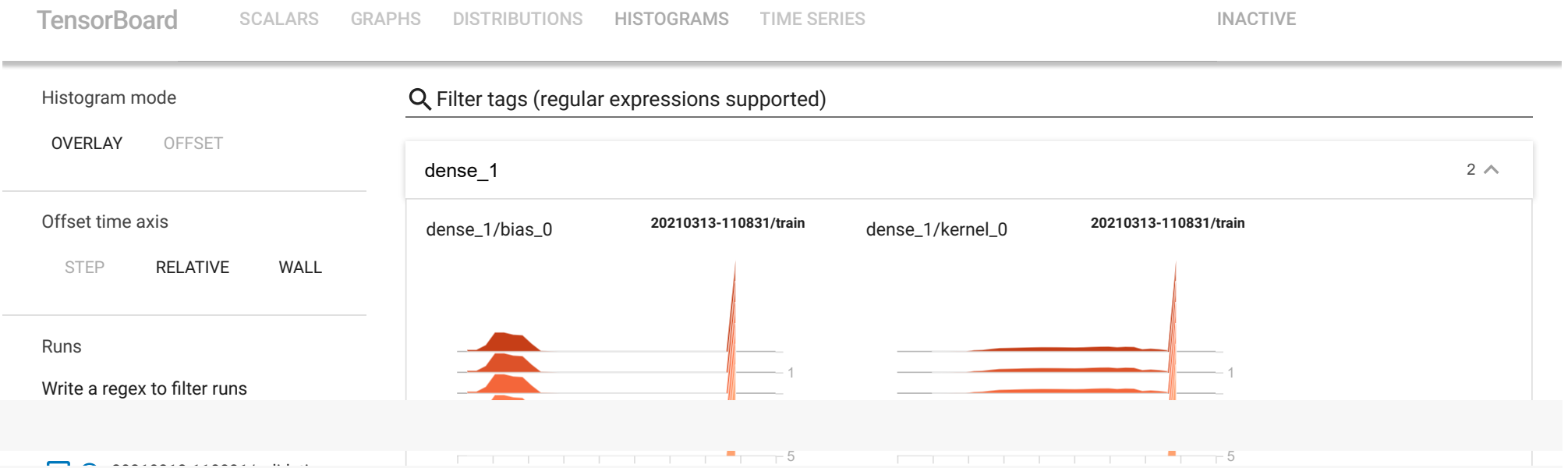
```
Epoch 5/20
4
Learning rate is : 0.08550000190734863
250/250 [==============================] - 2s 6ms/step - loss: 0.6168 - accuracy: 0.6586 - val_loss: 0.6288 - val_accuracy: 0.6405
 - micro f1 score = 0.6405 - auc = 0.6402599405848665

Epoch 00005: val_accuracy did not improve from 0.65875
Epoch 6/20
5
Learning rate is : 0.07694999873638153
250/250 [==============================] - 2s 6ms/step - loss: 0.6151 - accuracy: 0.6635 - val_loss: 0.6263 - val_accuracy: 0.6513
 - micro f1 score = 0.65125 - auc = 0.6511498400871402

Epoch 00006: val_accuracy did not improve from 0.65875
<tensorflow.python.keras.callbacks.History at 0x7f826e63c690>
```

Summary of model 3:

1. Model 3 ran upto 7 epochs
2. Validation accuracy : 0.6587
3. micro f1 score : 0.6515

```
%tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6007 (pid 1489), started 5:25:49 ago. (Use '!kill 1489' to kill it.)
```




```
# clear all sessions
tf.keras.backend.clear_session()
```

Model-4

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

```
# clear all sessions
tf.keras.backend.clear_session()

# defining model 4
initializer = tf.keras.initializers.he_normal()
activation = 'relu'
def create_model4():
    return tf.keras.models.Sequential([
        tf.keras.layers.Dense(512,
                              input_shape=(2,),
                              activation=activation,
                              kernel_initializer=initializer
                             ),
        tf.keras.layers.Dense(256,
                              activation=activation,
                              kernel_initializer=initializer
                             ),
        #tf.keras.layers.Dropout(.2),
        tf.keras.layers.Dense(128,
                              activation=activation,
```

```
                            kernel_initializer=initializer
                           ),
        tf.keras.layers.Dense(84,
                            activation=activation,
                            kernel_initializer=initializer
                           ),
        tf.keras.layers.Dense(32,
                            activation=activation,
                            kernel_initializer=initializer
                           ),
        tf.keras.layers.Dense(1,
                            activation='sigmoid',
                            kernel_initializer=initializer
                           )
    ])

# creating an object of model 3
model4 = create_model4()
model4.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               1536
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 84)                10836
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2720
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 179,349
Trainable params: 179,349
Non-trainable params: 0
_________________________________________________________________
```

```
# remove previous logs
!rm -rf logs
# remove previous saved files
!rm -rf model_save

# initializing optimizer
optimizer = tf.keras.optimizers.Adam(0.1)
# compiling the model
model4.compile(optimizer=optimizer,
             loss='binary_crossentropy',
             metrics=['accuracy']
             )
# Fit the model
```

```
# Fit the model
model4.fit(X_train, y_train,
         validation_data=(X_test,y_test),
         epochs=20,
         batch_size=64,
         callbacks=[callback_list]
         )
```

```
Epoch 1/20
0
Learning rate is : 0.10000000149011612
  3/250 [..............................] - ETA: 11s - loss: 289.7768 - accuracy: 0.4887WARNING:tensorflow:Callback method `on_train_batch_end` is slow compar
250/250 [==============================] - 2s 8ms/step - loss: 22.2963 - accuracy: 0.5709 - val_loss: 0.6495 - val_accuracy: 0.6382
 - micro f1 score = 0.63825 - auc = 0.6380073105164487

Epoch 00001: val_accuracy did not improve from 0.65875
Epoch 2/20
1
Learning rate is : 0.10000000149011612
250/250 [==============================] - 2s 7ms/step - loss: 0.6122 - accuracy: 0.6633 - val_loss: 0.6382 - val_accuracy: 0.6398
 - micro f1 score = 0.63975 - auc = 0.6395568140028316

Epoch 00002: val_accuracy did not improve from 0.65875
Epoch 3/20
2
Learning rate is : 0.10000000149011612
250/250 [==============================] - 2s 7ms/step - loss: 0.6163 - accuracy: 0.6582 - val_loss: 0.6255 - val_accuracy: 0.6497
 - micro f1 score = 0.64975 - auc = 0.6497240868791955

Epoch 00003: val_accuracy did not improve from 0.65875
Epoch 4/20
3
Learning rate is : 0.0949999988079071
250/250 [==============================] - 2s 7ms/step - loss: 0.6050 - accuracy: 0.6705 - val_loss: 0.6250 - val_accuracy: 0.6545
 - micro f1 score = 0.6545 - auc = 0.6544947226131259

Epoch 00004: val_accuracy did not improve from 0.65875
Epoch 5/20
4
Learning rate is : 0.0949999988079071
250/250 [==============================] - 2s 7ms/step - loss: 0.6122 - accuracy: 0.6629 - val_loss: 0.6530 - val_accuracy: 0.6382
 - micro f1 score = 0.63825 - auc = 0.638168560879262

Epoch 00005: val_accuracy did not improve from 0.65875
Epoch 6/20
5
Learning rate is : 0.08550000190734863
250/250 [==============================] - 2s 7ms/step - loss: 0.6108 - accuracy: 0.6600 - val_loss: 0.6243 - val_accuracy: 0.6543
 - micro f1 score = 0.65425 - auc = 0.6541708468844055

Epoch 00006: val_accuracy did not improve from 0.65875
<tensorflow.python.keras.callbacks.History at 0x7f8267af8bd0>
```

Summary of model 4:

1. Model 4 ran upto 6 epochs
2. Validation accuracy : 0.65875
3. micro f1 score : 0.65425

slightly better than all other model

```
%tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6007 (pid 1489), started 5:33:54 ago. (Use '!kill 1489' to kill it.)
```

Histogram mode

OVERLAY OFFSET

Offset time axis

STEP RELATIVE WALL

Runs

Write a regex to filter runs

- 20210313-110831/train
- 20210313-110831/validation

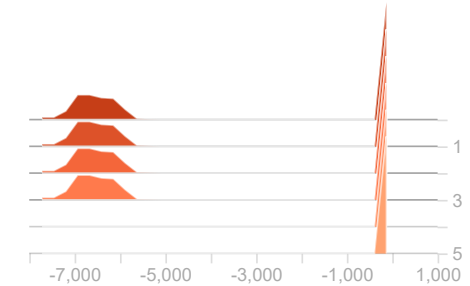
TOGGLE ALL RUNS

logs/fit

Filter tags (regular expressions supported)

dense_1 2

dense_1/bias_0 20210313-110831/train



dense_1/kernel_0 20210313-110831/train