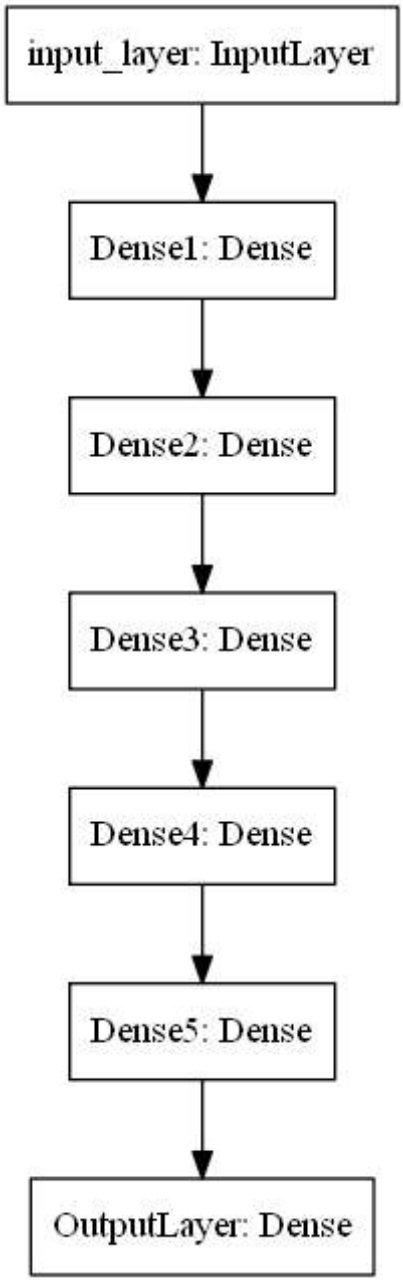
1. Download the data from here. You have to use data.csv file for this assignment

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




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

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

```
import pandas as pd
import numpy as np
```

```
path = "/content/drive/MyDrive/Assignment_data/data.csv"
df = pd.read_csv(path)

df.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

```
from sklearn.model_selection import train_test_split
y = df['label'].values
X = df.drop(['label'], axis=1)

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.20, stratify=y)
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

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

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 epoch a
       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)

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 tensorflow as tf
from tensorflow.keras.layers import Dense,Input,Activation
from tensorflow.keras.models import Model
import random as rn


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

class metric_callback(tf.keras.callbacks.Callback):
    def __init__(self,validation_data):
      self.x_test = validation_data[0]
      self.y_test= validation_data[1]
    def on_train_begin(self, logs={}):
        ## on begin of training, we are creating a instance varible called history
        ## it is a dict with keys [loss, acc, val_loss, val_acc]
        self.history={'loss': [],'accuracy': [],'val_loss': [],'val_accuracy': [],'val_f1s

    def on_epoch_end(self, epoch, logs={}):
        true_positives=0
        ## on end of each epoch, we will get logs and update the self.history dict
        self.history['loss'].append(logs.get('loss'))
        self.history['accuracy'].append(logs.get('accuracy'))

        if logs.get('val_loss', -1) != -1:
            self.history['val_loss'].append(logs.get('val_loss'))
        if logs.get('val_accuracy', -1) != -1:
            self.history['val_accuracy'].append(logs.get('val_accuracy'))
        # we can get a list of all predicted values at the end of the epoch
        # we can use these predicted value and the true values to calculate any custom eva
        # Here we are taking log of all true positives and then taking average of it
        y_pred= self.model.predict(self.x_test)
        y_label_pred=np.argmax(y_pred,axis=1)

        #we can also calcualte predefined metrics such as precison, recall, etc. using cal
        f1score = round(f1_score(self.y_test,y_label_pred,average='micro'),4)
        self.history['val_f1score'].append(f1score)
        auc = round(roc_auc_score(self.y_test,y_label_pred),4)
        self.history['val_AUC'].append(auc)
```

```
class modelSave_callback(tf.keras.callbacks.Callback):
    def __init__(self,path):
      self.filepath_name = path

    def on_train_begin(self, logs={}):
        ## on begin of training, we are creating a instance varible called history
        ## it is a dict with keys [loss, acc, val_loss, val_acc]
        self.history={'loss': [],'accuracy': [],'val_loss': [],'val_accuracy': [],'val_f1s

    def on_epoch_end(self, epoch, logs={}):
        true_positives=0
        ## on end of each epoch, we will get logs and update the self.history dict
        self.history['loss'].append(logs.get('loss'))
        self.history['accuracy'].append(logs.get('accuracy'))

        if logs.get('val_loss', -1) != -1:
            self.history['val_loss'].append(logs.get('val_loss'))
        if logs.get('val_accuracy', -1) != -1:
            self.history['val_accuracy'].append(logs.get('val_accuracy'))

        if epoch >= 2:
          if self.history.get('val_accuracy')[-1] > self.history.get('val_accuracy')[-2]:
            self.model.save(self.filepath_name)

class learning_rate_callback(tf.keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        ## on begin of training, we are creating a instance varible called history
        ## it is a dict with keys [loss, acc, val_loss, val_acc]
        self.history={'loss': [],'accuracy': [],'val_loss': [],'val_accuracy': [],'val_f1s

    def on_epoch_end(self, epoch, logs={}):
        true_positives=0
        ## on end of each epoch, we will get logs and update the self.history dict
        self.history['loss'].append(logs.get('loss'))
        self.history['accuracy'].append(logs.get('accuracy'))

        if logs.get('val_loss', -1) != -1:
            self.history['val_loss'].append(logs.get('val_loss'))
        if logs.get('val_accuracy', -1) != -1:
            self.history['val_accuracy'].append(logs.get('val_accuracy'))

    def on_epoch_begin(self, epoch, logs={}):
      if epoch > 2:
        if self.history.get('val_accuracy')[-1] < self.history.get('val_accuracy')[-2] or
          if self.history.get('val_accuracy')[-1] < self.history.get('val_accuracy')[-2]:
            self.model.optimizer.lr = self.model.optimizer.lr*0.9
          else:
            self.model.optimizer.lr = 0.95*self.model.optimizer.lr

class terminateNaN_callback(tf.keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        ## on begin of training, we are creating a instance varible called history
        ## it is a dict with keys [loss, acc, val_loss, val_acc]
        self.history={'loss': [],'accuracy': [],'val_loss': [],'val_accuracy': [],'val_f1s
```

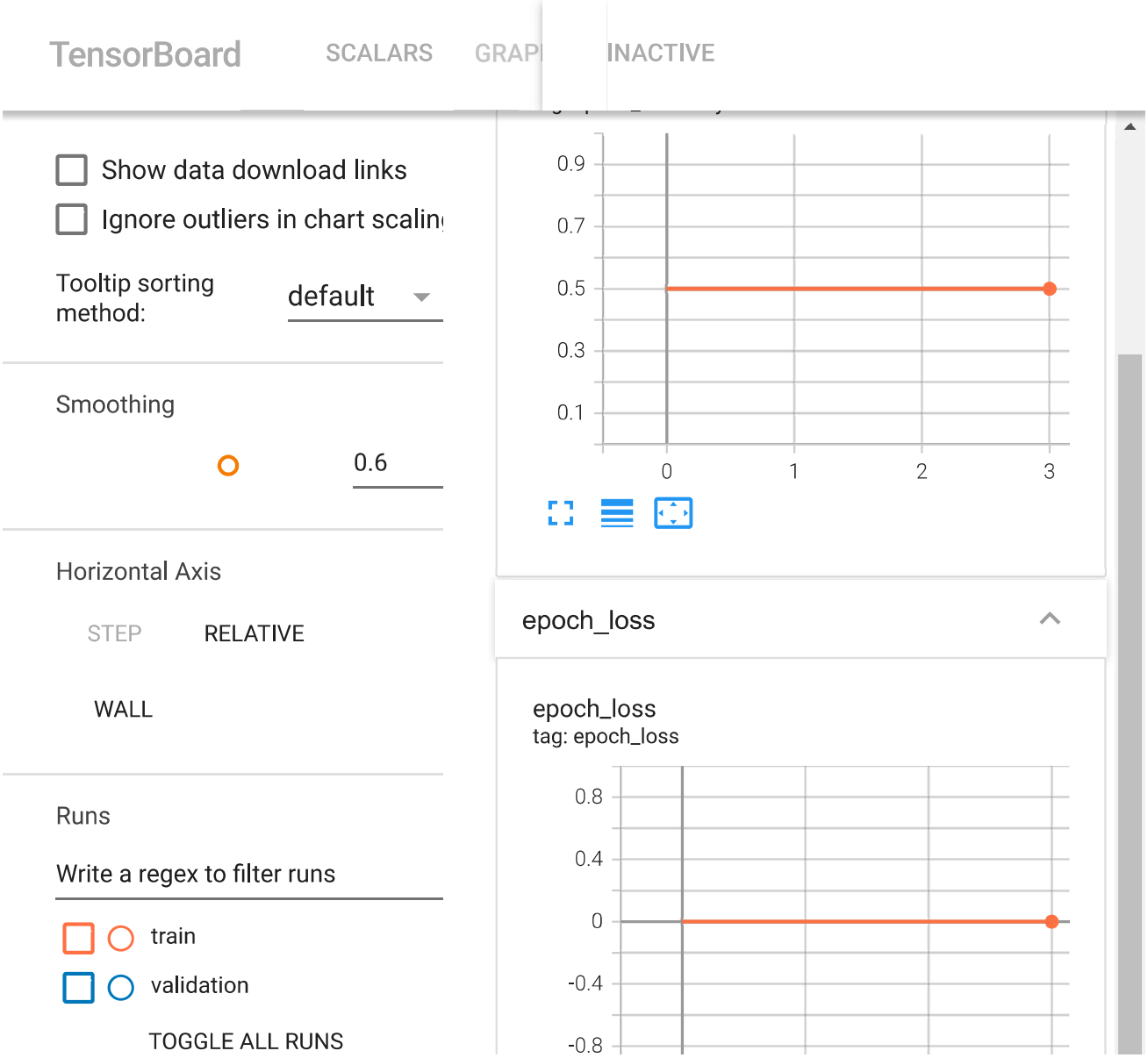
```
        def on_epoch_end(self, epoch, logs={}):
            true_positives=0
            ## on end of each epoch, we will get logs and update the self.history dict
            self.history['loss'].append(logs.get('loss'))
            self.history['accuracy'].append(logs.get('accuracy'))

            if logs.get('val_loss', -1) != -1:
                self.history['val_loss'].append(logs.get('val_loss'))
            if logs.get('val_accuracy', -1) != -1:
                self.history['val_accuracy'].append(logs.get('val_accuracy'))

            model_weights = self.model.get_weights()
            if model_weights is not None:
                if np.any([np.any(np.isnan(x)) for x in model_weights]):
                  print("Invalid model weights and terminated at epoch {}".format(epoch))
                  self.model.stop_training = True
            loss = logs.get('loss')
            if loss is not None:
              if np.isnan(loss) or np.isinf(loss):
                print("Invalid loss and terminated at epoch {}".format(epoch))
                self.model.stop_training = True

    class modelcheck_callback(tf.keras.callbacks.Callback):
        def on_train_begin(self, logs={}):
            ## on begin of training, we are creating a instance varible called history
            ## it is a dict with keys [loss, acc, val_loss, val_acc]
            self.history={'loss': [],'accuracy': [],'val_loss': [],'val_accuracy': [],'val_f1s

        def on_epoch_end(self, epoch, logs={}):
            true_positives=0
            ## on end of each epoch, we will get logs and update the self.history dict
            self.history['loss'].append(logs.get('loss'))
            self.history['accuracy'].append(logs.get('accuracy'))

            if logs.get('val_loss', -1) != -1:
                self.history['val_loss'].append(logs.get('val_loss'))
            if logs.get('val_accuracy', -1) != -1:
                self.history['val_accuracy'].append(logs.get('val_accuracy'))

            if epoch >= 3:
              if self.history.get('val_accuracy')[-1] <= self.history.get('val_accuracy')[-2]
                print("Val_accuracy is not improved from last 2 epochs")
                self.model.stop_training = True



    # there are other ways of doing this: https://www.dlology.com/blog/quick-guide-to-run-tens
    %load_ext tensorboard
    # Clear any logs from previous runs
    !rm -rf ./logs/


    from tensorflow.keras.callbacks import ModelCheckpoint,TerminateOnNaN,EarlyStopping,Learni

    #Input layer
```

```
input_layer = Input(shape=(2,))
#Dense hidden layer
layer1 = Dense(32,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform
layer2 = Dense(16,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform
layer3 = Dense(8,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(
layer4 = Dense(4,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(
layer5 = Dense(2,activation='tanh',kernel_initializer=tf.keras.initializers.RandomUniform(
#output layer
output = Dense(1,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_norm
#Creating a model
model1 = Model(inputs=input_layer,outputs=output)

optimizer = tf.keras.optimizers.SGD(learning_rate=0.1)

model1.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy'])

#Callbacks
mc=metric_callback(validation_data=[X_test,Y_test])
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
ms = modelSave_callback(filepath)
lr = learning_rate_callback()
tn = terminateNaN_callback()
mcheck = modelcheck_callback()
t_c = tf.keras.callbacks.TensorBoard(log_dir="./logs")

callback_lst = [mc,ms,lr,tn,mcheck,t_c]

model1.fit(X_train,Y_train,epochs=10,validation_data=(X_test,Y_test),batch_size=16,callbac
```

```
Epoch 1/10
   1/1000 [..............................] - ETA: 6:54 - loss: 0.0000e+00 - accuracy
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0000e+00 - accurac
Epoch 2/10
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0000e+00 - accurac
Epoch 3/10
1000/1000 [==============================] - 1s 1ms/step - loss: 0.0000e+00 - accurac
Epoch 4/10
 993/1000 [============================>.] - ETA: 0s - loss: 0.0000e+00 - accuracy: 0
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0000e+00 - accurac
<keras.callbacks.History at 0x7fdf72564910>
```

```
%tensorboard --logdir logs
```

```
Reusing TensorBoard on port 6008 (pid 2411), started 0:06:30 ago. (Use '!kill 2411' t
```



1. Acuuracy & AUC are same i.e. 0.5
2. Training terminated at epoch 4 because of no improvement in the "Val_Accuracy"
3. Both Train and test accuracies are same

evaluation_accuracy_vs_iterations

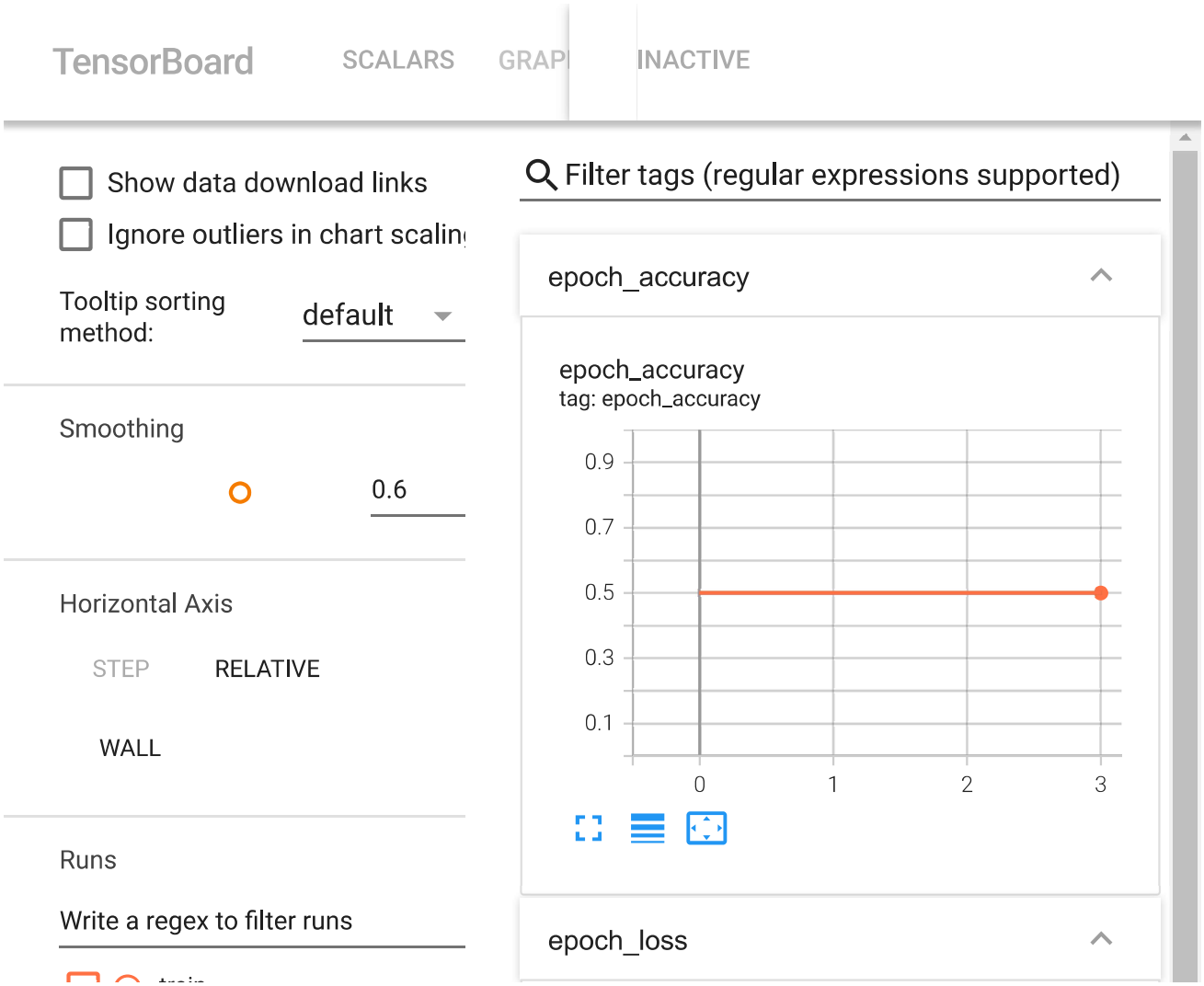
Model-2

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.

```
#Input layer
input_layer = Input(shape=(2,))
#Dense hidden layer
layer1 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform
layer2 = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform
layer3 = Dense(8,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(
layer4 = Dense(4,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(
layer5 = Dense(2,activation='relu',kernel_initializer=tf.keras.initializers.RandomUniform(
#output layer
output = Dense(1,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_norm
#Creating a model
model2 = Model(inputs=input_layer,outputs=output)

optimizer = tf.keras.optimizers.SGD(learning_rate=0.1, momentum=0.9)

model2.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy'])

#Callbacks
mc=metric_callback(validation_data=[X_test,Y_test])
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
ms = modelSave_callback(filepath)
lr = learning_rate_callback()
tn = terminateNaN_callback()
mcheck = modelcheck_callback()
t_c = tf.keras.callbacks.TensorBoard(log_dir="./logs")

callback_lst = [mc,ms,lr,tn,mcheck,t_c]

model2.fit(X_train,Y_train,epochs=10,validation_data=(X_test,Y_test),batch_size=16,callbac
```

```
Epoch 1/10
   1/1000 [..............................] - ETA: 10:21 - loss: 0.0000e+00 - accuracy
 981/1000 [============================>.] - ETA: 0s - loss: nan - accuracy: 0.4992In
Invalid loss and terminated at epoch 0
1000/1000 [==============================] - 2s 2ms/step - loss: nan - accuracy: 0.50
<keras.callbacks.History at 0x7fdf72242b90>
```

```
%tensorboard --logdir logs_model2
```



1. Acuuracy & AUC are same i.e. 0.5
2. Training terminated at epoch 4 because of no improvement in the "Val_Accuracy"
3. Both Train and test accuracies are same

Model-3

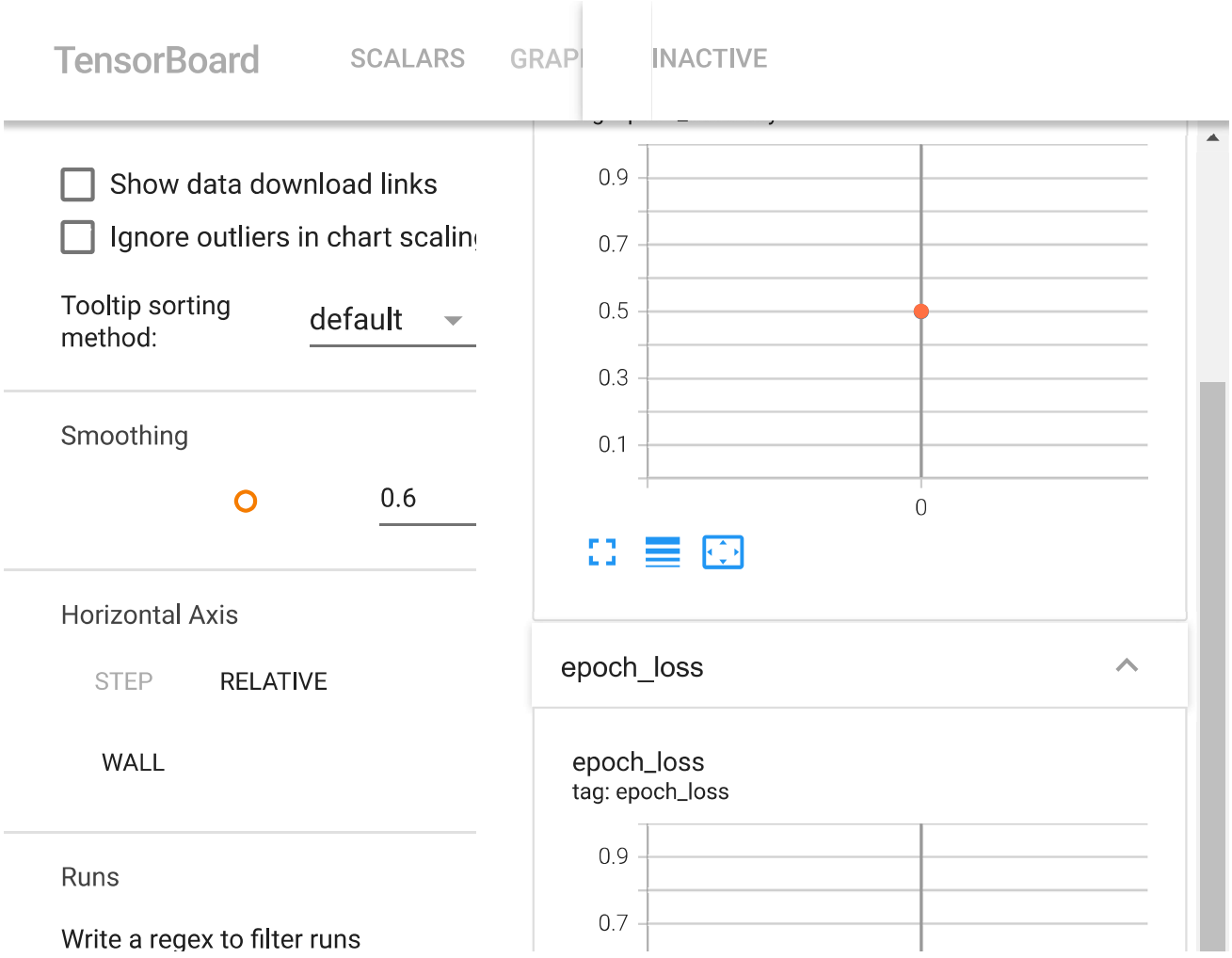
1. Use relu as an activation for every layer except output layer.
2. use SGD with momentum as optimizer.
3. use he_uniform() as initilizer.
3. Analyze your output and training process.

```
#Input layer
input_layer = Input(shape=(2,))
#Dense hidden layer
layer1 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform(see
layer2 = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform(see
layer3 = Dense(8,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform(seed
```

```
layer4 = Dense(4,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform(seed
layer5 = Dense(2,activation='relu',kernel_initializer=tf.keras.initializers.HeUniform(seed
#output layer
output = Dense(1,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_norm
#Creating a model
model3 = Model(inputs=input_layer,outputs=output)

optimizer = tf.keras.optimizers.SGD(learning_rate=0.1, momentum=0.9)

model3.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy'])

#Callbacks
mc=metric_callback(validation_data=[X_test,Y_test])
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
ms = modelSave_callback(filepath)
lr = learning_rate_callback()
tn = terminateNaN_callback()
mcheck = modelcheck_callback()
t_c = tf.keras.callbacks.TensorBoard(log_dir="./logs")

callback_lst = [mc,ms,lr,tn,mcheck,t_c]

model3.fit(X_train,Y_train,epochs=10,validation_data=(X_test,Y_test),batch_size=16,callbac
```

```
Epoch 1/10
   1/1000 [..............................] - ETA: 6:02 - loss: 0.0000e+00 - accuracy
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0000e+00 - accurac
Epoch 2/10
1000/1000 [==============================] - 1s 1ms/step - loss: 0.0000e+00 - accurac
Epoch 3/10
1000/1000 [==============================] - 1s 1ms/step - loss: 0.0000e+00 - accurac
Epoch 4/10
 975/1000 [============================>.] - ETA: 0s - loss: 0.0000e+00 - accuracy: 0
1000/1000 [==============================] - 1s 1ms/step - loss: 0.0000e+00 - accurac
<keras.callbacks.History at 0x7fdf6b929210>
```

```
%tensorboard --logdir logs_model3
```



1. Acuuracy & AUC are same i.e. 0.5013
2. Training terminated because of invalid loss(NaN) at epoch 0
3. Both Train and test accuracies are same

Model-4

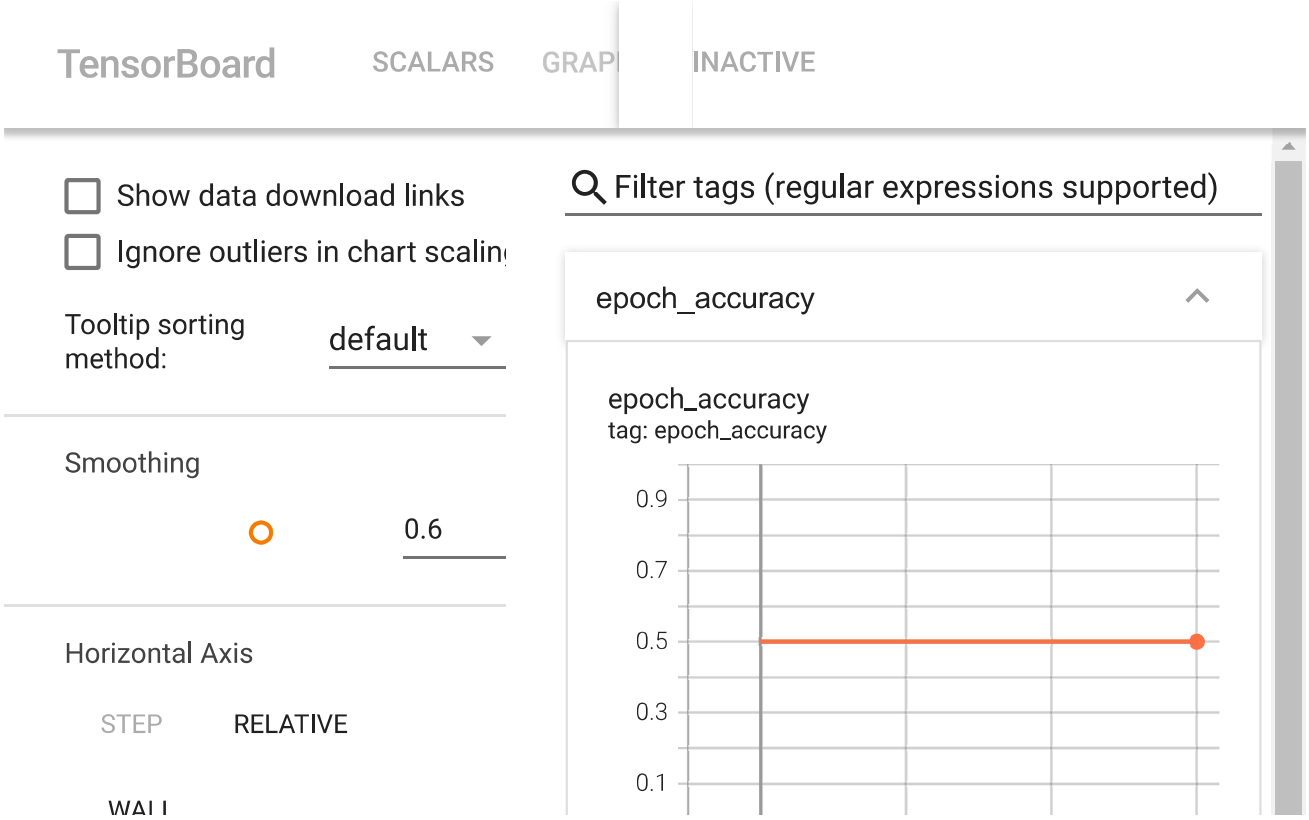
1. Try with any values to get better accuracy/f1 score.

```
#Input layer
input_layer = Input(shape=(2,))
#Dense hidden layer
layer1 = Dense(32,activation='relu',kernel_initializer=tf.keras.initializers.HeNormal(seed
layer2 = Dense(16,activation='relu',kernel_initializer=tf.keras.initializers.HeNormal(seed
layer3 = Dense(8,activation='relu',kernel_initializer=tf.keras.initializers.HeNormal(seed=
layer4 = Dense(4,activation='relu',kernel_initializer=tf.keras.initializers.HeNormal(seed=
layer5 = Dense(2,activation='relu',kernel_initializer=tf.keras.initializers.HeNormal(seed=
#output layer
output = Dense(1,activation='softmax',kernel_initializer=tf.keras.initializers.glorot_norm
#Creating a model
```

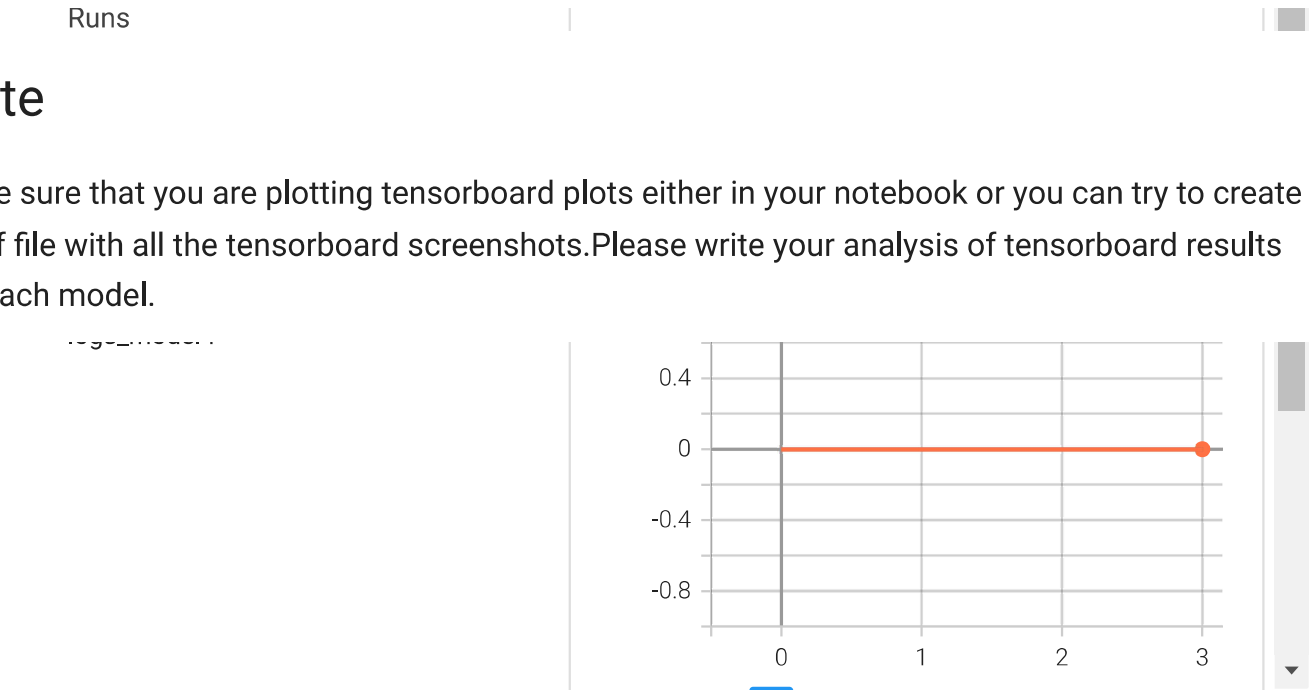
```
model4 = Model(inputs=input_layer,outputs=output)

optimizer = tf.keras.optimizers.Adam(learning_rate=0.1)

model4.compile(optimizer=optimizer, loss='categorical_crossentropy',metrics=['accuracy'])

#Callbacks
mc=metric_callback(validation_data=[X_test,Y_test])
filepath="model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
ms = modelSave_callback(filepath)
lr = learning_rate_callback()
tn = terminateNaN_callback()
mcheck = modelcheck_callback()
t_c = tf.keras.callbacks.TensorBoard(log_dir="./logs")

callback_lst = [mc,ms,lr,tn,mcheck,t_c]

model4.fit(X_train,Y_train,epochs=10,validation_data=(X_test,Y_test),batch_size=16,callbac
```

```
Epoch 1/10
   1/1000 [..............................] - ETA: 16:23 - loss: 0.0000e+00 - accuracy
1000/1000 [==============================] - 4s 3ms/step - loss: 0.0000e+00 - accurac
Epoch 2/10
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0000e+00 - accurac
Epoch 3/10
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0000e+00 - accurac
Epoch 4/10
 958/1000 [===========================>..] - ETA: 0s - loss: 0.0000e+00 - accuracy: 0
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0000e+00 - accurac
<keras.callbacks.History at 0x7fdf6b635490>
```

```
%tensorboard --logdir logs_model4
```



1. There is no improvement in either accuracy or f1_score



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.

✓ 14s completed at 11:08 PM