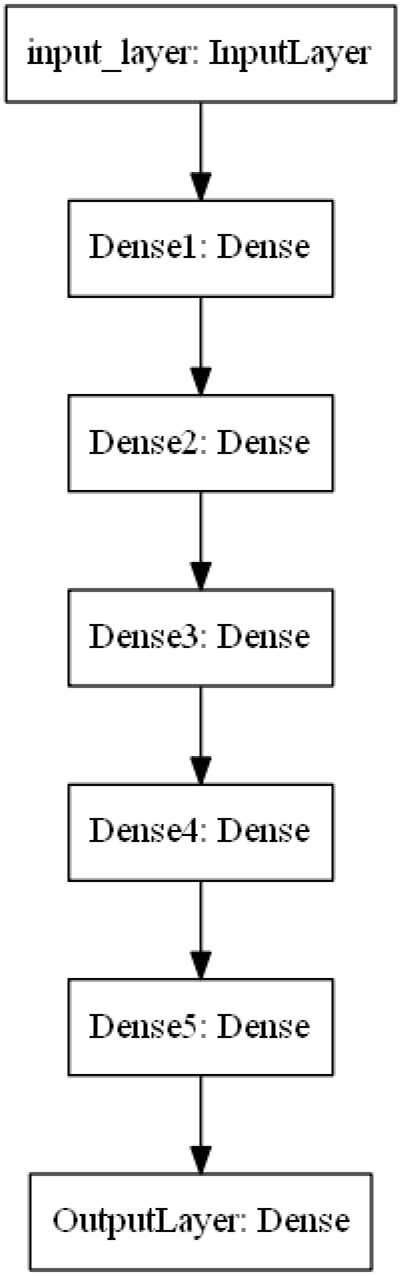
1. Download the data from here. You have to use data.csv file for this assignment

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



In [1]:
```
from google.colab import files

files = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving data.csv to data.csv
```

In [2]:
```
import pandas as pd

data = pd.read_csv('data.csv')
print(data.shape)
data.head()
```

```
(20000, 3)
```

Out[2]:

	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

In [3]:

```
from sklearn.model_selection import train_test_split

y = data['label']
X = data.drop(['label'], axis = 1)

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.33, random_state
print(X_train.shape)
print(X_test.shape)
```

```
(13400, 2)
(6600, 2)
```

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
accuracy, you have to decrese the
         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)

In [4]:

```
%load_ext tensorboard
import tensorflow as tf
import numpy as np
from tensorflow.keras.layers import Dense,Input,Activation
from tensorflow.keras.models import Model
from sklearn.metrics import f1_score, roc_auc_score
```

In [5]:

```
class custom_callback(tf.keras.callbacks.Callback):
  '''
```

```
  This class is defined to define our own custom callbacks to that has to print the micr
  after each epoch.
  '''
  def __init__(self, validation_data):
    self.x_test = validation_data[0]
    self.y_test = validation_data[1]

  def on_train_begin(self, logs ={}):
    # We are creating an instance variable history dictionary and initializing it.
    self.history = {'loss':[], 'accuracy':[], 'val_loss':[], 'val_acc':[], 'f1 score':[]

  def on_epoch_end(self, epoch, logs={}):
    #Getting the loss and checking if it is nan. If it is nan we terminate the training.
    l = logs.get('loss')
    if l is not None:
      if np.isnan(l) or np.isinf(l):
        print('Invalid Training Loss. Training terminated at epoch ',epoch)
        self.model.stop_training = True
    #Getting the weights and checking if it is nan. If it is nan we terminate the traini
    wts = self.model.get_weights()
    if wts is not None:
      if np.any([np.any(np.isnan(i)) for i in wts]):
        print('Invalid Weights. Training terminated at epoch ',epoch)
        self.model.stop_training = True

    self.history['loss'].append(l)
    #Getting accuracy, val_loss, and val_accuracy from the logs and storing in our histo
    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_acc'].append(logs.get('val_accuracy'))

    #Getting the y_pred values and calculating the micro f1_score and AUC score.
    y_pred = self.model.predict(self.x_test)
    #y_pred returns probability values. Keeping the threshold as 0.5; >=0.5 -> 1 and <0.
    #we are dividing by 0.5 and converting the values to the closest integers using rint
    y_pred_labels = np.rint(y_pred/0.5)
    self.history['f1 score'].append(f1_score(self.y_test, y_pred_labels, average = 'micr

    self.history['AUC score'].append(roc_auc_score(self.y_test, y_pred, average = 'micro

    #decaying the learning rate based on the conditions specified above.
    lr = float(self.model.optimizer.learning_rate)  #getting the current learning rate
    if (epoch>0) and ((epoch+1)%3 == 0):
      self.model.optimizer.learning_rate = 0.95*lr
      print('learning rate reduced by 5%')
    elif (epoch>1) and (self.history['val_acc'][-1]<self.history['val_acc'][-2]):
      self.model.optimizer.learning_rate = 0.9*lr
      print('learning rate reduced by 10%')
```

In [6]:
```
import os
import datetime
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import EarlyStopping
```

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.

In [7]:
```
X_train.iloc[1].shape
```
Out[7]:
```
(2,)
```
In [8]:
```
#Input Layer
input_layer = Input(shape = (2,))
#Dense hidden layer 1
layer1 = Dense(100,activation='tanh', kernel_initializer=tf.keras.initializers.RandomUni
#Dense hidden layer 2
layer2 = Dense(150,activation='tanh', kernel_initializer=tf.keras.initializers.RandomUni
#Dense hidden layer 3
layer3 = Dense(50,activation='tanh', kernel_initializer=tf.keras.initializers.RandomUnif
#Dense hidden layer 4
layer4 = Dense(40,activation='tanh', kernel_initializer=tf.keras.initializers.RandomUnif
#Dense hidden layer 5
layer5 = Dense(20,activation='tanh', kernel_initializer=tf.keras.initializers.RandomUnif
#Output Layer
output_layer = Dense(1,activation='sigmoid', kernel_initializer=tf.keras.initializers.Ra

#To save model at each epoch if validation accuracy improves compared to previous epoch
filepath="model1_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss',  verbose=1, save_bes

#To stop the training if the validation accuracy doesn't increase in consecutive 2 epoch
earlystop = EarlyStopping(monitor = 'val_loss', patience = 2)

model = Model(inputs = input_layer, outputs = output_layer)
Optimiser = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)

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

log_dir = os.path.join("logs",'fits', datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,w

custom_callbacks = custom_callback(validation_data = (X_test, y_test))

model.fit(x = X_train, y = y_train, epochs = 10, validation_data = (X_test, y_test), bat
            callbacks = [custom_callbacks, checkpoint, earlystop, tensorboard_callback])
```
```
Epoch 1/10
  1/670 [..............................] - ETA: 46:45 - loss: 4.5582 - accuracy: 0.5500
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch ti
me (batch time: 0.0017s vs `on_train_batch_end` time: 0.0033s). Check your callbacks.
653/670 [============================>.] - ETA: 0s - loss: 0.7826 - accuracy: 0.4900
Epoch 1: val_loss improved from inf to 0.69395, saving model to model1_save/weights-01-
0.4950.hdf5
670/670 [==============================] - 7s 4ms/step - loss: 0.7805 - accuracy: 0.4901
- val_loss: 0.6939 - val_accuracy: 0.4950
Epoch 2/10
660/670 [============================>.] - ETA: 0s - loss: 0.6994 - accuracy: 0.5077
Epoch 2: val_loss did not improve from 0.69395
670/670 [==============================] - 3s 4ms/step - loss: 0.6992 - accuracy: 0.5087
- val_loss: 0.7215 - val_accuracy: 0.5050
Epoch 3/10
664/670 [============================>.] - ETA: 0s - loss: 0.7010 - accuracy: 0.4963lear
ning rate reduced by 5%

Epoch 3: val_loss improved from 0.69395 to 0.69328, saving model to model1_save/weights-
03-0.5050.hdf5
670/670 [==============================] - 3s 4ms/step - loss: 0.7009 - accuracy: 0.4967
- val_loss: 0.6933 - val_accuracy: 0.5050
Epoch 4/10
669/670 [============================>.] - ETA: 0s - loss: 0.6987 - accuracy: 0.4975
Epoch 4: val_loss did not improve from 0.69328
670/670 [==============================] - 3s 5ms/step - loss: 0.6987 - accuracy: 0.4978
```

```
- val_loss: 0.7116 - val_accuracy: 0.5050
Epoch 5/10
667/670 [============================>.] - ETA: 0s - loss: 0.7007 - accuracy: 0.5017lear
ning rate reduced by 10%

Epoch 5: val_loss did not improve from 0.69328
670/670 [==============================] - 3s 4ms/step - loss: 0.7007 - accuracy: 0.5020
- val_loss: 0.6938 - val_accuracy: 0.4950
```

Out[8]:

```
<keras.callbacks.History at 0x7fb9367d96d0>
```

In [9]:

```python
custom_callbacks.history
```

Out[9]:

```
{'loss': [0.7804831266403198,
  0.6991533637046814,
  0.700865626335144,
  0.6986677646636963,
  0.7006606459617615],
 'accuracy': [0.49007463455200195,
  0.5087313652038574,
  0.4967164099216461,
  0.49776118993759155,
  0.5020149350166321],
 'val_loss': [0.6939479112625122,
  0.721451997756958,
  0.693284809589386,
  0.7115606665611267,
  0.6937859058380127],
 'val_acc': [0.4950000047683716,
  0.5049999952316284,
  0.5049999952316284,
  0.5049999952316284,
  0.4950000047683716],
 'f1 score': [0.4984848484848485,
  0.4984848484848485,
  0.4984848484848485,
  0.4984848484848485,
  0.4984848484848485],
 'AUC score': [0.49475472685699595,
  0.5047842037116961,
  0.505245273143004,
  0.5045532098550032,
  0.49475472685699595]}
```

In [10]:

```python
%tensorboard --logdir logs/fits
```

In [11]:

```python
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0

 dense (Dense)               (None, 100)               300

 dense_1 (Dense)             (None, 150)               15150

 dense_2 (Dense)             (None, 50)                7550

 dense_3 (Dense)             (None, 40)                2040

 dense_4 (Dense)             (None, 20)                820

 dense_5 (Dense)             (None, 1)                 21
```

```
=================================================================
Total params: 25,881
Trainable params: 25,881
Non-trainable params: 0
_________________________________________________________________
```

Observations for Model 1:

1. The use of 2 hidden units and softmax activation in the output layer led to errors due different shapes of the logits and the labels. For binary classification problems the better approach would be to use sigmoid activation with one hidden unit in the output dense layer.

2. The AUC score of 0.5 suggests that the model's performance is that of a random model.

3. As the epochs progress the weights of the model are not being updated as should be the case. The similar distributions of weights across the epochs suggests the same.

4. There is a decline in the accuracy when compared to that in the first epoch for train and validation sets. Which implies poor performance of the model. The decrease in the AUC score from 0.5 to 4.9 can also be observed.

5. Input layer has shape (2,) has there are 2 features.

6. No of parameters in each layer:

Layer 1:- 2(from input layer)*100(no of hidden units in this layer)+ 100(bias terms) = 300 params

Layer 2:- 100*150+150 = 15150 params

Layer 3:- 150*50+50 = 7550 params

Layer 4:- 50*40+40 = 2040 params

Layer 5:- 40*20+20 = 820 params

Output Layer:- 20*1+1 = 21 params

```
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.
```

In [12]:
```
#to reset backend and clear the logs associated with the previous model
tf.keras.backend.clear_session()
!rm -rf ./logs/
```

In [13]:
```
#Input Layer
input_layer = Input(shape = (2,))
#Dense hidden layer 1
layer1 = Dense(5,activation='relu', kernel_initializer=tf.keras.initializers.RandomUnifo
#Dense hidden layer 2
layer2 = Dense(10,activation='relu', kernel_initializer=tf.keras.initializers.RandomUnif
#Dense hidden layer 3
layer3 = Dense(15,activation='relu', kernel_initializer=tf.keras.initializers.RandomUnif
#Dense hidden layer 4
```

```
layer4 = Dense(10,activation='relu', kernel_initializer=tf.keras.initializers.RandomUnif
#Dense hidden layer 5
layer5 = Dense(5,activation='relu', kernel_initializer=tf.keras.initializers.RandomUnifo
#Output Layer
output_layer = Dense(1,activation='sigmoid', kernel_initializer=tf.keras.initializers.Ra

#To save model at each epoch if validation accuracy improves compared to previous epoch
filepath="model1_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss',  verbose=1, save_bes

#To stop the training if the validation accuracy doesn't increase in consecutive 2 epoch
earlystop = EarlyStopping(monitor = 'val_loss', patience = 2)

model2 = Model(inputs = input_layer, outputs = output_layer)
Optimiser = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)

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

log_dir = os.path.join("logs",'fits', datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,w

custom_callbacks = custom_callback(validation_data = (X_test, y_test))

model2.fit(x = X_train, y = y_train, epochs = 10, validation_data = (X_test, y_test), ba
           callbacks = [custom_callbacks, checkpoint, earlystop, tensorboard_callback])
```

```
Epoch 1/10
666/670 [============================>.] - ETA: 0s - loss: 0.7743 - accuracy: 0.5023
Epoch 1: val_loss improved from inf to 0.69478, saving model to model1_save/weights-01-
0.4985.hdf5
670/670 [==============================] - 3s 4ms/step - loss: 0.7738 - accuracy: 0.5026
- val_loss: 0.6948 - val_accuracy: 0.4985
Epoch 2/10
670/670 [==============================] - ETA: 0s - loss: 0.6936 - accuracy: 0.4973
Epoch 2: val_loss improved from 0.69478 to 0.69329, saving model to model1_save/weights-
02-0.5015.hdf5
670/670 [==============================] - 2s 4ms/step - loss: 0.6936 - accuracy: 0.4973
- val_loss: 0.6933 - val_accuracy: 0.5015
Epoch 3/10
653/670 [============================>.] - ETA: 0s - loss: 0.6936 - accuracy: 0.4983lear
ning rate reduced by 5%

Epoch 3: val_loss improved from 0.69329 to 0.69320, saving model to model1_save/weights-
03-0.4985.hdf5
670/670 [==============================] - 3s 4ms/step - loss: 0.6936 - accuracy: 0.4979
- val_loss: 0.6932 - val_accuracy: 0.4985
Epoch 4/10
652/670 [============================>.] - ETA: 0s - loss: 0.6934 - accuracy: 0.4998
Epoch 4: val_loss improved from 0.69320 to 0.69314, saving model to model1_save/weights-
04-0.5015.hdf5
670/670 [==============================] - 3s 4ms/step - loss: 0.6934 - accuracy: 0.4993
- val_loss: 0.6931 - val_accuracy: 0.5015
Epoch 5/10
650/670 [============================>.] - ETA: 0s - loss: 0.6935 - accuracy: 0.5004lear
ning rate reduced by 10%

Epoch 5: val_loss did not improve from 0.69314
670/670 [==============================] - 3s 4ms/step - loss: 0.6935 - accuracy: 0.5013
- val_loss: 0.6939 - val_accuracy: 0.4985
Epoch 6/10
652/670 [============================>.] - ETA: 0s - loss: 0.6935 - accuracy: 0.4985lear
ning rate reduced by 5%

Epoch 6: val_loss did not improve from 0.69314
670/670 [==============================] - 2s 4ms/step - loss: 0.6935 - accuracy: 0.4991
- val_loss: 0.6933 - val_accuracy: 0.5015
```

Out[13]: <keras.callbacks.History at 0x7fb9363641d0>

In [14]:
```
custom_callbacks.history
```

Out[14]:
```
{'loss': [0.7737835049629211,
  0.6935919523239136,
  0.6936379671096802,
  0.693411111831665,
  0.6934697031974792,
  0.6935043334960938],
 'accuracy': [0.5026119351387024,
  0.4973134398460388,
  0.49791043996810913,
  0.4992537200450897,
  0.501343309879303,
  0.49910447001457214],
 'val_loss': [0.6947821974754333,
  0.693292498588562,
  0.6932048201560974,
  0.6931437253952026,
  0.6938538551330566,
  0.6932554841041565],
 'val_acc': [0.49848484992980957,
  0.5015151500701904,
  0.49848484992980957,
  0.5015151500701904,
  0.49848484992980957,
  0.5015151500701904],
 'f1 score': [0.4984848484848485,
  0.4984848484848485,
  0.4984848484848485,
  0.4984848484848485,
  0.4984848484848485,
  0.4984848484848485],
 'AUC score': [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]}
```

In [15]:
```
%tensorboard --logdir logs/fits
```

```
Reusing TensorBoard on port 6006 (pid 236), started 0:02:35 ago. (Use '!kill 236' to kil
l it.)
```

In [16]:
```
model2.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0

 dense (Dense)               (None, 5)                 15

 dense_1 (Dense)             (None, 10)                60

 dense_2 (Dense)             (None, 15)                165

 dense_3 (Dense)             (None, 10)                160

 dense_4 (Dense)             (None, 5)                 55

 dense_5 (Dense)             (None, 1)                 6

=================================================================
Total params: 461
Trainable params: 461
```

```
Non-trainable params: 0
_________________________________________________________________
```

Observations for Model 2:

1. The use of 2 hidden units and softmax activation in the output layer led to errors due different shapes of the logits and the labels. For binary classification problems the better approach would be to use sigmoid activation with one hidden unit in the output dense layer.

2. The AUC score of 0.5 suggests that the model's performance is that of a random model.

3. As the epochs progress the weights of the model are not being updated as should be the case. The similar distributions of weights across the epochs suggests the same.

4. Input layer has shape (2,) has there are 2 features.

5. No of parameters in each layer:

Layer 1:- 2(from input layer)*5(no of hidden units in this layer)+ 5(bias terms) = 15 params

Layer 2:- 5*10+10 = 60 params

Layer 3:- 10*15+15 = 165 params

Layer 4:- 15*10+10 = 160 params

Layer 5:- 10*5+5 = 55 params

Output Layer:- 5*1+1 = 6 params

```
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.
```

In [17]:
```
#to reset backend and clear the logs associated with the previous model
tf.keras.backend.clear_session()
!rm -rf ./logs/
```

In [18]:
```
#Input Layer
input_layer = Input(shape = (2,))
#Dense hidden layer 1
layer1 = Dense(5,activation='relu', kernel_initializer=tf.keras.initializers.HeUniform(s
#Dense hidden layer 2
layer2 = Dense(10,activation='relu', kernel_initializer=tf.keras.initializers.HeUniform(
#Dense hidden layer 3
layer3 = Dense(15,activation='relu', kernel_initializer=tf.keras.initializers.HeUniform(
#Dense hidden layer 4
layer4 = Dense(10,activation='relu', kernel_initializer=tf.keras.initializers.HeUniform(
#Dense hidden layer 5
layer5 = Dense(5,activation='relu', kernel_initializer=tf.keras.initializers.HeUniform(s
#Output Layer
output_layer = Dense(1,activation='sigmoid', kernel_initializer=tf.keras.initializers.He

#To save model at each epoch if validation accuracy improves compared to previous epoch
filepath="model1_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss',  verbose=1, save_bes
```

```
#To stop the training if the validation accuracy doesn't increase in consecutive 2 epoch
earlystop = EarlyStopping(monitor = 'val_loss', patience = 2)

model3 = Model(inputs = input_layer, outputs = output_layer)
Optimiser = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)

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

log_dir = os.path.join("logs",'fits', datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,w

custom_callbacks = custom_callback(validation_data = (X_test, y_test))

model3.fit(x = X_train, y = y_train, epochs = 10, validation_data = (X_test, y_test), ba
           callbacks = [custom_callbacks, checkpoint, earlystop, tensorboard_callback])
```

```
Epoch 1/10
  1/670 [..............................] - ETA: 4:27 - loss: 0.6775 - accuracy: 0.4500
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch ti
me (batch time: 0.0017s vs `on_train_batch_end` time: 0.0024s). Check your callbacks.
651/670 [============================>.] - ETA: 0s - loss: 0.6862 - accuracy: 0.5604
Epoch 1: val_loss improved from inf to 0.66700, saving model to model1_save/weights-01-
0.6065.hdf5
670/670 [==============================] - 3s 4ms/step - loss: 0.6855 - accuracy: 0.5627
- val_loss: 0.6670 - val_accuracy: 0.6065
Epoch 2/10
659/670 [============================>.] - ETA: 0s - loss: 0.6481 - accuracy: 0.6251
Epoch 2: val_loss improved from 0.66700 to 0.63879, saving model to model1_save/weights-
02-0.6374.hdf5
670/670 [==============================] - 3s 4ms/step - loss: 0.6478 - accuracy: 0.6254
- val_loss: 0.6388 - val_accuracy: 0.6374
Epoch 3/10
666/670 [============================>.] - ETA: 0s - loss: 0.6258 - accuracy: 0.6495lear
ning rate reduced by 5%

Epoch 3: val_loss improved from 0.63879 to 0.62490, saving model to model1_save/weights-
03-0.6494.hdf5
670/670 [==============================] - 3s 4ms/step - loss: 0.6260 - accuracy: 0.6490
- val_loss: 0.6249 - val_accuracy: 0.6494
Epoch 4/10
659/670 [============================>.] - ETA: 0s - loss: 0.6119 - accuracy: 0.6638
Epoch 4: val_loss improved from 0.62490 to 0.61856, saving model to model1_save/weights-
04-0.6594.hdf5
670/670 [==============================] - 3s 4ms/step - loss: 0.6123 - accuracy: 0.6638
- val_loss: 0.6186 - val_accuracy: 0.6594
Epoch 5/10
663/670 [============================>.] - ETA: 0s - loss: 0.6115 - accuracy: 0.6649lear
ning rate reduced by 10%

Epoch 5: val_loss did not improve from 0.61856
670/670 [==============================] - 3s 4ms/step - loss: 0.6118 - accuracy: 0.6644
- val_loss: 0.6228 - val_accuracy: 0.6541
Epoch 6/10
669/670 [============================>.] - ETA: 0s - loss: 0.6082 - accuracy: 0.6661lear
ning rate reduced by 5%

Epoch 6: val_loss improved from 0.61856 to 0.61381, saving model to model1_save/weights-
06-0.6611.hdf5
670/670 [==============================] - 2s 4ms/step - loss: 0.6083 - accuracy: 0.6660
- val_loss: 0.6138 - val_accuracy: 0.6611
Epoch 7/10
664/670 [============================>.] - ETA: 0s - loss: 0.6070 - accuracy: 0.6671
Epoch 7: val_loss did not improve from 0.61381
670/670 [==============================] - 2s 4ms/step - loss: 0.6067 - accuracy: 0.6671
```

```
- val_loss: 0.6155 - val_accuracy: 0.6639
Epoch 8/10
654/670 [============================>.] - ETA: 0s - loss: 0.6055 - accuracy: 0.6671lear
ning rate reduced by 10%

Epoch 8: val_loss did not improve from 0.61381
670/670 [==============================] - 3s 4ms/step - loss: 0.6053 - accuracy: 0.6670
- val_loss: 0.6228 - val_accuracy: 0.6571
```

Out[18]:

```
<keras.callbacks.History at 0x7fb92267d150>
```

In [19]:

```
custom_callbacks.history
```

Out[19]:

```
{'loss': [0.6855247020721436,
  0.6477634310722351,
  0.6260292530059814,
  0.612332403659820 6,
  0.6117744445800781,
  0.6082570552825928,
  0.6067020893096924,
  0.6053016185760498],
 'accuracy': [0.562686562538147,
  0.6254477500915527,
  0.6490298509597778,
  0.6638059616088867,
  0.664402961730957,
  0.6660447716712952,
  0.667089581489563,
  0.6670148968696594],
 'val_loss': [0.6670029759407043,
  0.638794481754303,
  0.6248975396156311,
  0.6185637712478638,
  0.6228281855583191,
  0.6138055920600891,
  0.615460991859436,
  0.6227663159370422],
 'val_acc': [0.6065151691436768,
  0.6374242305755615,
  0.6493939161300659,
  0.6593939661979675,
  0.6540908813476562,
  0.661060631275177,
  0.6639394164085388,
  0.6571212410926819],
 'f1 score': [0.4984848484848485,
  0.4813636363636364,
  0.43696969696969695,
  0.519090909090909,
  0.5113636363636364,
  0.44984848484848483,
  0.42848484848484847,
  0.39939393939393936],
 'AUC score': [0.6547324126025033,
  0.6884264777454338,
  0.7080480537011359,
  0.7210524890035722,
  0.7135768464356881,
  0.7245455881137568,
  0.723322298643697 3,
  0.7177826242665222]}
```

In [20]:

```
%tensorboard --logdir logs/fits
```

```
Reusing TensorBoard on port 6006 (pid 236), started 0:04:51 ago. (Use '!kill 236' to kil
l it.)
```

In [21]:
```
model3.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0

 dense (Dense)               (None, 5)                 15

 dense_1 (Dense)             (None, 10)                60

 dense_2 (Dense)             (None, 15)                165

 dense_3 (Dense)             (None, 10)                160

 dense_4 (Dense)             (None, 5)                 55

 dense_5 (Dense)             (None, 1)                 6

=================================================================
Total params: 461
Trainable params: 461
Non-trainable params: 0
_________________________________________________________________
```

Observations for Model 3:

1. The initialization technique used plays an important role in performance of the model. HeUniform initialization definitely leads to better performance than the RandomUniform initialization. It boosted the AUC score from 0.5 in previous cases to 0.7.

2. Unlike model2 where the the distributions of weights remained similar for all epochs, we can see the distributions of weights varying as the epochs progress.

3. No of parameters in each layer:

Layer 1:- 2(from input layer)*5(no of hidden units in this layer)+ 5(bias terms) = 15 params

Layer 2:- 5*10+10 = 60 params

Layer 3:- 10*15+15 = 165 params

Layer 4:- 15*10+10 = 160 params

Layer 5:- 10*5+5 = 55 params

Output Layer:- 5*1+1 = 6 params

1. Weights in the hidden layers and output layer:-

- Layer 1:- The distribution of weights lies approx between -1.6 to 1.6. The distrubution consists of disconnected small distributions.
- Layer 2:- The distribution of weights lies approx between -1.2 to 1.3.

- Layer 3:- The distribution of weights lies approx between -0.8 to 1 for all epochs. The bell shape is more wider and more distorted (i.e less smooth) than those of the previous layer.
- Layer 4:- The distribution of weights lies approx between -0.6 to 1 for all epochs.

- Layer 5:- The distribution of weights lies approx between -0.8 to 0.8. The distrubution consists of disconnected smaller and sharper distributions.
- Output layer:- The distribution of weights lies between approx -0.2 to 1.5 and this outer bound gradually decreases to 1.3 as the epochs progress.

```
Model-4
1. Try with any values to get better accuracy/f1 score.
```

In [22]:
```
#to reset backend and clear the logs associated with the previous model
tf.keras.backend.clear_session()
!rm -rf ./logs/
```

In [23]:
```
#Input Layer
input_layer = Input(shape = (2,))
#Dense hidden layer 1
layer1 = Dense(50,activation='relu', kernel_initializer=tf.keras.initializers.HeUniform(
#Dense hidden layer 2
layer2 = Dense(40,activation='relu', kernel_initializer=tf.keras.initializers.HeUniform(
#Dense hidden layer 3
layer3 = Dense(30,activation='relu', kernel_initializer=tf.keras.initializers.HeUniform(
#Dense hidden layer 4
layer4 = Dense(20,activation='relu', kernel_initializer=tf.keras.initializers.HeUniform(
#Dense hidden layer 5
layer5 = Dense(10,activation='relu', kernel_initializer=tf.keras.initializers.HeUniform(
#Output Layer
output_layer = Dense(1,activation='sigmoid', kernel_initializer=tf.keras.initializers.He

#To save model at each epoch if validation accuracy improves compared to previous epoch
filepath="model1_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss',  verbose=1, save_bes

#To stop the training if the validation accuracy doesn't increase in consecutive 2 epoch
earlystop = EarlyStopping(monitor = 'val_loss', patience = 2)

model4 = Model(inputs = input_layer, outputs = output_layer)
Optimiser = tf.keras.optimizers.Adam(0.02)

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

log_dir = os.path.join("logs",'fits', datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1,w

custom_callbacks = custom_callback(validation_data = (X_test, y_test))

model4.fit(x = X_train, y = y_train, epochs = 10, validation_data = (X_test, y_test), ba
           callbacks = [custom_callbacks, checkpoint, earlystop, tensorboard_callback])
```

```
Epoch 1/10
  1/670 [..............................] - ETA: 6:58 - loss: 0.6297 - accuracy: 0.7500
WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch ti
me (batch time: 0.0022s vs `on_train_batch_end` time: 0.0023s). Check your callbacks.
648/670 [============================>.] - ETA: 0s - loss: 0.6420 - accuracy: 0.6299
Epoch 1: val_loss improved from inf to 0.62277, saving model to model1_save/weights-01-
0.6500.hdf5
670/670 [==============================] - 3s 4ms/step - loss: 0.6419 - accuracy: 0.6306
- val_loss: 0.6228 - val_accuracy: 0.6500
Epoch 2/10
649/670 [============================>.] - ETA: 0s - loss: 0.6167 - accuracy: 0.6560
Epoch 2: val_loss improved from 0.62277 to 0.61854, saving model to model1_save/weights-
02-0.6579.hdf5
```

```
670/670 [==============================] - 3s 4ms/step - loss: 0.6164 - accuracy: 0.6560
- val_loss: 0.6185 - val_accuracy: 0.6579
Epoch 3/10
668/670 [============================>.] - ETA: 0s - loss: 0.6092 - accuracy: 0.6628lear
ning rate reduced by 5%

Epoch 3: val_loss did not improve from 0.61854
670/670 [==============================] - 2s 4ms/step - loss: 0.6093 - accuracy: 0.6627
- val_loss: 0.6403 - val_accuracy: 0.6497
Epoch 4/10
659/670 [============================>.] - ETA: 0s - loss: 0.6094 - accuracy: 0.6664
Epoch 4: val_loss did not improve from 0.61854
670/670 [==============================] - 3s 4ms/step - loss: 0.6087 - accuracy: 0.6674
- val_loss: 0.6408 - val_accuracy: 0.6500
```

Out[23]:

```
<keras.callbacks.History at 0x7fb922542950>
```

In [24]:

```python
custom_callbacks.history
```

Out[24]:

```
{'loss': [0.6418889760971069,
  0.6163898706436157,
  0.6092991828918457,
  0.6086574792861938],
 'accuracy': [0.6305969953536987,
  0.6560447812080383,
  0.6626865863800049,
  0.6673880815505981],
 'val_loss': [0.6227667927742004,
  0.618542492389679,
  0.6403045058250427,
  0.640774130821228],
 'val_acc': [0.6499999761581421,
  0.6578788161277771,
  0.649696946144104,
  0.6499999761581421],
 'f1 score': [0.4987878787878788,
  0.4712121212121212,
  0.34984848484848485,
  0.5821212121212122],
 'AUC score': [0.7113697554614826,
  0.7176445146420078,
  0.7149295677646259,
  0.7208787959485395]}
```

In [25]:

```python
%tensorboard --logdir logs/fits
```

```
Reusing TensorBoard on port 6006 (pid 236), started 0:07:10 ago. (Use '!kill 236' to kil
l it.)
```

In [26]:

```python
model4.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0

 dense (Dense)               (None, 50)                150

 dense_1 (Dense)             (None, 40)                2040

 dense_2 (Dense)             (None, 30)                1230

 dense_3 (Dense)             (None, 20)                620

 dense_4 (Dense)             (None, 10)                210
```

```
 dense_5 (Dense)             (None, 1)                 11

=================================================================
Total params: 4,261
Trainable params: 4,261
Non-trainable params: 0
_________________________________________________________________
```

Observations for Model 4:

1. Increasing the number of units in the hidden layers than in model 3, made no significant contribution to the performance of the model.

2. Use of Adam optimizer helped converge the model in less no of epochs as expected.

3. The initialization technique used plays an important role in performance of the model. HeUniform initialization definitely leads to better performance than the RandomUniform initialization. It boosted the AUC score from 0.5 in previous cases to 0.7.

4. No of parameters in each layer:

Layer 1:- 2(from input layer)*50(no of hidden units in this layer)+ 50(bias terms) = 150 params

Layer 2:- 50*40+40 = 2040 params

Layer 3:- 40*30+30 = 1230 params

Layer 4:- 30*20+20 = 620 params

Layer 5:- 20*10+10 = 210 params

Output Layer:- 10*1+1 = 11 params

1. Weights in the hidden layers and output layer:-

- Layer 1:- The distribution of weights lies approx between -2.1 to 2.1.
- Layer 2:- The distribution of weights lies approx between -1.6 to 1.2 for all epochs and comes close to the bell shape of the normal distribution without much skewness in any direction.

- Layer 3:- The distribution of weights lies approx between -1.2 to 1.2 for all epochs. The bell shape is more wider and more distorted (i.e less smooth) than those of the previous layer.
- Layer 4:- The distribution of weights lies approx between -1 to 1 for all epochs. The bell shape is more wider and more distorted (i.e less smooth) than those of the previous 2 layers with multiple peaks.

- Layer 5:- The distribution of weights lies approx between -0.6 to 0.8. The distribution consists of disconnected samller and sharper distributions.
- Output layer:- The distribution of weights lies between approx -1 to 1.8 for all epochs and comes close to the bell shape of the normal distribution with right skewness.

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.

```
In [ ]:
```