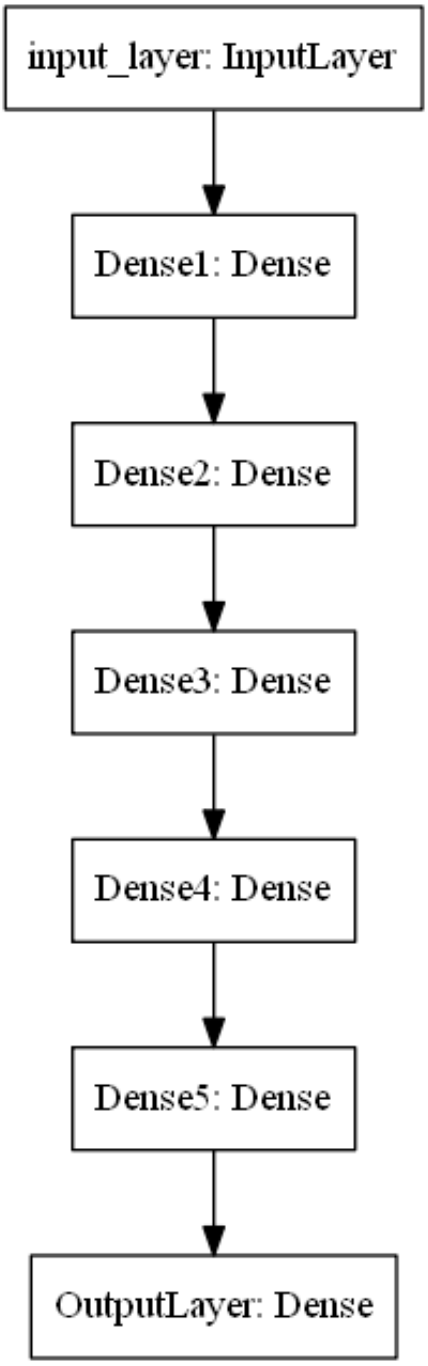
1. Download the data from here (https://drive.google.com/file/d/15dCNcmKskcFVjs7R0ElQkR61Ex53uJpM/view?usp=sharing)

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 sco re.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall
import tensorflow as tf
from tensorflow.keras.layers import Dense,Input,Activation, Dropout
from tensorflow.keras.models import Model, Sequential
import random as rn
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnP
import tensorflow.keras.backend as K
from tensorflow.keras.metrics import AUC
import datetime
import warnings; warnings.simplefilter('ignore')
```

In [2]:
```
data=pd.read_csv('data.csv', index_col=False)
```

In [3]:
```
data
```

Out[3]:

	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
...	...	...	...
19995	-0.491252	-0.561558	0.0
19996	-0.813124	0.049423	1.0
19997	-0.010594	0.138790	1.0
19998	0.671827	0.804306	0.0
19999	-0.854865	-0.588826	0.0

20000 rows × 3 columns

In [4]:
```
Y = data['label'].values
X = data.drop(['label'], axis=1)
```

In [5]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size
# X_train, X_cv, Y_train, Y_cv = train_test_split(X_train, Y_train,
```

In [6]:
```
print(X_train.shape)
print(Y_train.shape)
```

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

In [7]:
```
class Callback(object):

    """Abstract base class used to build new callbacks.
      Attributes:
          params: dict. Training parameters
              (eg. verbosity, batch size, number of epochs...).
          model: instance of `keras.models.Model`.
              Reference of the model being trained.
          validation_data: Deprecated. Do not use.
      The `logs` dictionary that callback methods
      take as argument will contain keys for quantities relevant to
      the current batch or epoch.
      Currently, the `.fit()` method of the `Model` class
      will include the following quantities in the `logs` that
      it passes to its callbacks:
          on_epoch_end: logs include `acc` and `loss`, and
          optionally include `val_loss`
          (if validation is enabled in `fit`), and `val_acc`
          (if validation and accuracy monitoring are enabled).
          on_batch_begin: logs include `size`,
          the number of samples in the current batch.
          on_batch_end: logs include `loss`, and optionally `acc`
            (if accuracy monitoring is enabled).
      """

    def __init__(self):
        self.validation_data = None
        self.model = None
        # Whether this Callback should only run on the chief worker
        # Multi-Worker setting.
        # TODO(omalleyt): Make this attr public once solution is st
        self._chief_worker_only = None

    def set_params(self, params):
        self.params = params

    def set_model(self, model):
        self.model = model

    def on_batch_begin(self, batch, logs=None):
        """A backwards compatibility alias for `on_train_batch_begi

    def on_batch_end(self, batch, logs=None):
        """A backwards compatibility alias for `on_train_batch_end`

    def on_epoch_begin(self, epoch, logs=None):
        """Called at the start of an epoch.
```

```
        Subclasses should override for any actions to run. This fun
        be called during TRAIN mode.
        Arguments:
            epoch: integer, index of epoch.
            logs: dict. Currently no data is passed to this argumen
              but that may change in the future.
        """

    def on_epoch_end(self, epoch, logs=None):
        """Called at the end of an epoch.
        Subclasses should override for any actions to run. This fun
        be called during TRAIN mode.
        Arguments:
            epoch: integer, index of epoch.
            logs: dict, metric results for this training epoch, and
              validation epoch if validation is performed. Validati
              are prefixed with `val_`.
        """

    def on_train_batch_begin(self, batch, logs=None):
        """Called at the beginning of a training batch in `fit` met
        Subclasses should override for any actions to run.
        Arguments:
            batch: integer, index of batch within the current epoch
            logs: dict. Has keys `batch` and `size` representing th
              number and the size of the batch.
        """
        # For backwards compatibility.
        self.on_batch_begin(batch, logs=logs)

    def on_train_batch_end(self, batch, logs=None):
        """Called at the end of a training batch in `fit` methods.
        Subclasses should override for any actions to run.
        Arguments:
            batch: integer, index of batch within the current epoch
            logs: dict. Metric results for this batch.
        """
        # For backwards compatibility.
        self.on_batch_end(batch, logs=logs)

    def on_test_batch_begin(self, batch, logs=None):
        """Called at the beginning of a batch in `evaluate` methods
        Also called at the beginning of a validation batch in the `
        methods, if validation data is provided.
        Subclasses should override for any actions to run.
        Arguments:
            batch: integer, index of batch within the current epoch
            logs: dict. Has keys `batch` and `size` representing th
                  number and the size of the batch.
        """

    def on_test_batch_end(self, batch, logs=None):
        """Called at the end of a batch in `evaluate` methods.
        Also called at the end of a validation batch in the `fit`
```

```
        methods, if validation data is provided.
        Subclasses should override for any actions to run.
        Arguments:
            batch: integer, index of batch within the current epoch
            logs: dict. Metric results for this batch.
        """

    def on_predict_batch_begin(self, batch, logs=None):
        """Called at the beginning of a batch in `predict` methods.
        Subclasses should override for any actions to run.
        Arguments:
            batch: integer, index of batch within the current epoch
            logs: dict. Has keys `batch` and `size` representing th
                  number and the size of the batch.
        """

    def on_predict_batch_end(self, batch, logs=None):
        """Called at the end of a batch in `predict` methods.
        Subclasses should override for any actions to run.
        Arguments:
            batch: integer, index of batch within the current epoch
            logs: dict. Metric results for this batch.
        """

    def on_train_begin(self, logs=None):
        """Called at the beginning of training.
        Subclasses should override for any actions to run.
        Arguments:
            logs: dict. Currently no data is passed to this argumen
                  but that may change in the future.
        """

    def on_train_end(self, logs=None):
        """Called at the end of training.
        Subclasses should override for any actions to run.
        Arguments:
            logs: dict. Currently no data is passed to this argumen
                  but that may change in the future.
        """

    def on_test_begin(self, logs=None):
        """Called at the beginning of evaluation or validation.
        Subclasses should override for any actions to run.
        Arguments:
            logs: dict. Currently no data is passed to this argumen
              but that may change in the future.
        """

    def on_test_end(self, logs=None):
        """Called at the end of evaluation or validation.
        Subclasses should override for any actions to run.
        Arguments:
            logs: dict. Currently no data is passed to this argumen
              but that may change in the future.
```

```
        """

    def on_predict_begin(self, logs=None):
        """Called at the beginning of prediction.
        Subclasses should override for any actions to run.
        Arguments:
            logs: dict. Currently no data is passed to this argumen
              but that may change in the future.
        """

    def on_predict_end(self, logs=None):
        """Called at the end of prediction.
        Subclasses should override for any actions to run.
        Arguments:
            logs: dict. Currently no data is passed to this argumen
              but that may change in the future.
    """
```

In [13]:

```
class LossHistory(tf.keras.callbacks.Callback):

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

    def on_epoch_end(self, epoch, logs={}):
        ## on end of each epoch, we will get logs and update the se
        self.history['loss'].append(logs.get('loss'))
        self.history['acc'].append(logs.get('acc'))
        loss = logs.get('loss')
        if logs.get('val_loss', -1) != -1:
            self.history['val_loss'].append(logs.get('val_loss'))
        if logs.get('val_acc', -1) != -1:
            self.history['val_acc'].append(logs.get('val_acc'))
        if loss is not None:
          if np.isnan(loss) or np.isinf(loss):
            print("Invalid loss and terminated at epoch {}".format(
            self.model.stop_training = True

        val_predict = (np.asarray(self.model.predict(X_test))).roun
        val_target = Y_test
        _val_f1 = f1_score(val_target, val_predict)
        val_predict = (np.asarray(self.model.predict(X_test)))
        _val_auc = roc_auc_score(val_target, val_predict)
        self.val_f1s.append(_val_f1)
        self.val_aucs.append(_val_auc)
        print ("— val_f1:", _val_f1, "— val_auc:", _val_auc)
        return

history_own=LossHistory()
```

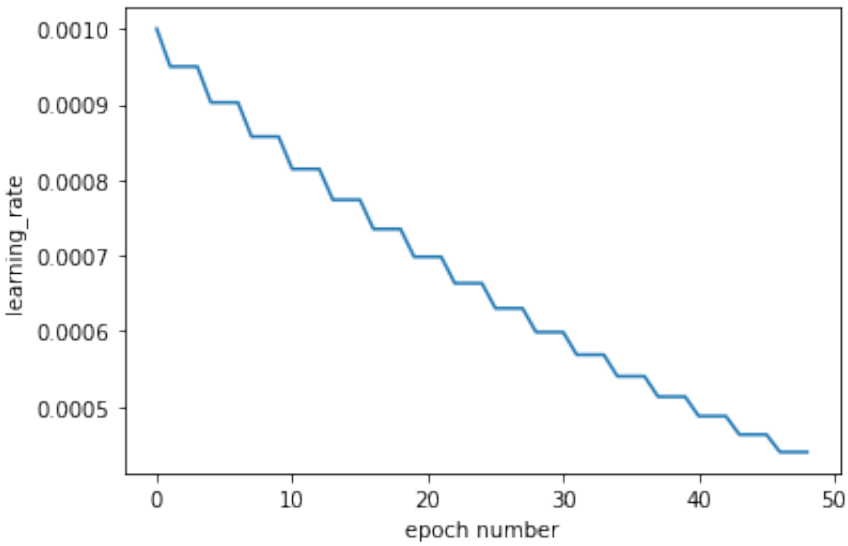
In [14]:
```
## Learning Rate Scheduler
from tensorflow.keras.callbacks import LearningRateScheduler

def changeLearningRate(epoch, lr):
    if((epoch+1)%3==0):
      lr=0.95*lr
    return lr
```

In [10]:
```
changed_lr = []
lr = 0.001
for i in range(1,50):
    lr = changeLearningRate(i, lr)
    changed_lr.append(lr)
```

In [11]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(changed_lr)
plt.ylabel('learning_rate')
plt.xlabel('epoch number')
```

Out[11]:
```
Text(0.5, 0, 'epoch number')
```



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.
4. Analyze your output and training process.

In [15]:
```
def create_model1():
  ## weights initialization
  weights = tf.keras.initializers.RandomUniform(0,1)
  return tf.keras.models.Sequential([
                                     Input(shape=(2,)),
```

```
                                        Dense(48, activation='tanh', k
                                        Dense(35, activation='tanh', k
                                        Dense(24, activation='tanh', k
                                        Dense(13, activation='tanh', k
                                        Dense(7, activation='tanh', ke
                                        Dense(1, activation='sigmoid',
  ])

model = create_model1()

#Callbacks
history_own = LossHistory()
filepath="content/model_save/weights-{epoch:02d}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accura
lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.9, p
es=EarlyStopping(monitor='val_accuracy',patience=2)

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_d

# here we are creating a list with all the callbacks we want
callback_list = [tensorboard_callback, history_own, checkpoint, lrs
optimizer = tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0

model.compile(optimizer=optimizer, loss='binary_crossentropy',metri
model.fit(X_train,Y_train,epochs=50, validation_data=(X_test, Y_tes
```

```
Epoch 1/50

Epoch 00001: LearningRateScheduler reducing learning rate to 0.001
0000000474974513.
 3/32 [=>............................] - ETA: 1s - loss: 1.5882 -
accuracy: 0.4900 WARNING:tensorflow:Callback method `on_train_batc
h_end` is slow compared to the batch time (batch time: 0.0032s vs
`on_train_batch_end` time: 0.0109s). Check your callbacks.
32/32 [==============================] - 1s 11ms/step - loss: 1.50
45 - accuracy: 0.5002 - val_loss: 1.2743 - val_accuracy: 0.4882
- val_f1: 0.4858075860336599 - val_auc: 0.48805671736969447

Epoch 00001: val_accuracy improved from -inf to 0.48825, saving mo
del to content/model_save/weights-01.hdf5
Epoch 2/50

Epoch 00002: LearningRateScheduler reducing learning rate to 0.001
0000000474974513.
32/32 [==============================] - 0s 4ms/step - loss: 1.155
7 - accuracy: 0.5020 - val_loss: 0.9518 - val_accuracy: 0.4882
- val_f1: 0.4870959659233275 - val_auc: 0.4885928815699808

Epoch 00002: val_accuracy did not improve from 0.48825

Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.0009000
000427477062.
```

```
Epoch 3/50

Epoch 00003: LearningRateScheduler reducing learning rate to 0.000
8550000406103208.
32/32 [==============================] - 0s 3ms/step - loss: 0.878
8 - accuracy: 0.5069 - val_loss: 0.7954 - val_accuracy: 0.4888
- val_f1: 0.4873401855101529 - val_auc: 0.48523660370988614

Epoch 00003: val_accuracy improved from 0.48825 to 0.48875, saving
model to content/model_save/weights-03.hdf5
Epoch 4/50

Epoch 00004: LearningRateScheduler reducing learning rate to 0.000
85500004934147.
32/32 [==============================] - 0s 4ms/step - loss: 0.761
3 - accuracy: 0.5063 - val_loss: 0.7314 - val_accuracy: 0.4888
- val_f1: 0.4873401855101529 - val_auc: 0.48847059411944904

Epoch 00004: val_accuracy did not improve from 0.48875

Epoch 00004: ReduceLROnPlateau reducing learning rate to 0.0007695
00044407323.
Epoch 5/50

Epoch 00005: LearningRateScheduler reducing learning rate to 0.000
7695000385865569.
32/32 [==============================] - 0s 3ms/step - loss: 0.718
2 - accuracy: 0.5013 - val_loss: 0.7083 - val_accuracy: 0.4882
- val_f1: 0.4876095118898623 - val_auc: 0.48823677251158165

Epoch 00005: val_accuracy did not improve from 0.48875

Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.0006925
500347279012.
```

Out[15]:
```
<tensorflow.python.keras.callbacks.History at 0x7f2610a56190>
```

In [17]:
```
%load_ext tensorboard
```

In [18]:
```
%tensorboard --logdir /content/logs/fit/20210225-074028
#logs\fit\20
```

```
<IPython.core.display.Javascript object>
```

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.
4. Analyze your output and training process.

In [19]:
```
def create_model2():
  ## weights initialization
  weights = tf.keras.initializers.RandomUniform(0,1)
  return tf.keras.models.Sequential([
                                     Input(shape=(2,)),
                                     Dense(48, activation='relu', k
                                     Dense(34, activation='relu', k
                                     Dense(20, activation='relu', k
                                     Dense(14, activation='relu', k
                                     Dense(6, activation='relu', ke
                                     Dense(1, activation='sigmoid',
  ])

model = create_model2()

#Callbacks
history_own = LossHistory()
filepath="content/model_save/weights-{epoch:02d}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accura
lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.9, p
es=EarlyStopping(monitor='val_accuracy',patience=2)


log_dir="logs_2/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H
tensorboard = TensorBoard(log_dir=log_dir,histogram_freq=1, write_g

# here we are creating a list with all the callbacks we want
callback_list = [tensorboard, history_own, checkpoint, lrschedule,

optimizer = tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0

model.compile(optimizer=optimizer, loss='binary_crossentropy',metri

model.fit(X_train,Y_train,epochs=50, validation_data=(X_test, Y_tes
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0
for the `TensorBoard` Callback.
Epoch 1/50

Epoch 00001: LearningRateScheduler reducing learning rate to 0.001
0000000474974513.
 3/32 [=>............................] - ETA: 1s - loss: 6591.0667
- accuracy: 0.4875 WARNING:tensorflow:Callback method `on_train_ba
tch_end` is slow compared to the batch time (batch time: 0.0025s v
s `on_train_batch_end` time: 0.0112s). Check your callbacks.
32/32 [==============================] - 1s 12ms/step - loss: 1417
.9297 - accuracy: 0.4952 - val_loss: 0.6931 - val_accuracy: 0.5088
- val_f1: 0.0 - val_auc: 0.5

Epoch 00001: val_accuracy improved from -inf to 0.50875, saving mo
del to content/model_save/weights-01.hdf5
Epoch 2/50
```

```
Epoch 00002: LearningRateScheduler reducing learning rate to 0.001
0000000474974513.
32/32 [==============================] - 0s 4ms/step - loss: 0.693
1 - accuracy: 0.5020 - val_loss: 0.6931 - val_accuracy: 0.5088
- val_f1: 0.0 - val_auc: 0.5

Epoch 00002: val_accuracy did not improve from 0.50875

Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.0009000
000427477062.
Epoch 3/50

Epoch 00003: LearningRateScheduler reducing learning rate to 0.000
8550000406103208.
32/32 [==============================] - 0s 3ms/step - loss: 0.693
1 - accuracy: 0.4953 - val_loss: 0.6931 - val_accuracy: 0.5088
- val_f1: 0.0 - val_auc: 0.5

Epoch 00003: val_accuracy did not improve from 0.50875

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.0007695
00044407323.
```

Out[19]: <tensorflow.python.keras.callbacks.History at 0x7f260ae35750>

In [20]:
```
%tensorboard --logdir /content/logs_2/fit/20210225-074215
```

```
<IPython.core.display.Javascript object>
```

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.
4. Analyze your output and training process.

In [21]:
```
def create_model3():
  ## weights initialization
  weights = tf.keras.initializers.HeUniform()
  return tf.keras.models.Sequential([
                                     Input(shape=(2,)),
                                     Dense(45, activation='relu', k
                                     Dense(31, activation='relu', k
                                     Dense(25, activation='relu', k
                                     Dense(12, activation='relu', k
                                     Dense(5, activation='relu', ke
                                     Dense(1, activation='sigmoid',
  ])

model = create_model3()

#Callbacks
```

```
#callbacks
history_own = LossHistory()
filepath="content/model_save/3/weights-{epoch:02d}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accura
lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.9, p
es=EarlyStopping(monitor='val_accuracy',patience=2)


log_dir="logs_3/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H
tensorboard = TensorBoard(log_dir=log_dir,histogram_freq=1, write_g

# here we are creating a list with all the callbacks we want
callback_list = [tensorboard, history_own, checkpoint, lrschedule,

optimizer = tf.keras.optimizers.SGD(learning_rate=0.1, momentum=0.5

model.compile(optimizer=optimizer, loss='binary_crossentropy',metri

model.fit(X_train,Y_train,epochs=10, validation_data=(X_test, Y_tes
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0
for the `TensorBoard` Callback.
Epoch 1/10

Epoch 00001: LearningRateScheduler reducing learning rate to 0.100
00000149011612.
  3/125 [..............................] - ETA: 3s - loss: 0.7181
- accuracy: 0.5451 WARNING:tensorflow:Callback method `on_train_ba
tch_end` is slow compared to the batch time (batch time: 0.0022s v
s `on_train_batch_end` time: 0.0096s). Check your callbacks.
125/125 [==============================] - 1s 4ms/step - loss: 0.6
777 - accuracy: 0.5733 - val_loss: 0.6253 - val_accuracy: 0.6640
- val_f1: 0.6525336091003102 - val_auc: 0.7262446624278684

Epoch 00001: val_accuracy improved from -inf to 0.66400, saving mo
del to content/model_save/3/weights-01.hdf5
Epoch 2/10

Epoch 00002: LearningRateScheduler reducing learning rate to 0.100
00000149011612.
125/125 [==============================] - 0s 2ms/step - loss: 0.6
202 - accuracy: 0.6634 - val_loss: 0.6097 - val_accuracy: 0.6643
- val_f1: 0.6585303839308415 - val_auc: 0.7274447799638639

Epoch 00002: val_accuracy improved from 0.66400 to 0.66425, saving
model to content/model_save/3/weights-02.hdf5
Epoch 3/10

Epoch 00003: LearningRateScheduler reducing learning rate to 0.095
00000141561031.
125/125 [==============================] - 0s 2ms/step - loss: 0.6
102 - accuracy: 0.6671 - val_loss: 0.6073 - val_accuracy: 0.6635
- val_f1: 0.6687992125984251 - val_auc: 0.7313593538021018
```

```
Epoch 00003: val_accuracy did not improve from 0.66425

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.0854999
9892711639.
Epoch 4/10

Epoch 00004: LearningRateScheduler reducing learning rate to 0.085
50000190734863.
125/125 [==============================] - 0s 2ms/step - loss: 0.6
067 - accuracy: 0.6647 - val_loss: 0.6047 - val_accuracy: 0.6687
- val_f1: 0.6656573303053243 - val_auc: 0.7335313939894093

Epoch 00004: val_accuracy improved from 0.66425 to 0.66875, saving
model to content/model_save/3/weights-04.hdf5
Epoch 5/10

Epoch 00005: LearningRateScheduler reducing learning rate to 0.085
50000190734863.
125/125 [==============================] - 0s 2ms/step - loss: 0.6
066 - accuracy: 0.6709 - val_loss: 0.6139 - val_accuracy: 0.6725
- val_f1: 0.6196283391405343 - val_auc: 0.7358628580002626

Epoch 00005: val_accuracy improved from 0.66875 to 0.67250, saving
model to content/model_save/3/weights-05.hdf5
Epoch 6/10

Epoch 00006: LearningRateScheduler reducing learning rate to 0.081
2250018119812.
125/125 [==============================] - 0s 2ms/step - loss: 0.6
007 - accuracy: 0.6719 - val_loss: 0.6036 - val_accuracy: 0.6735
- val_f1: 0.6604264170566823 - val_auc: 0.7343108827078293

Epoch 00006: val_accuracy improved from 0.67250 to 0.67350, saving
model to content/model_save/3/weights-06.hdf5
Epoch 7/10

Epoch 00007: LearningRateScheduler reducing learning rate to 0.081
22500032186508.
125/125 [==============================] - 0s 2ms/step - loss: 0.6
060 - accuracy: 0.6631 - val_loss: 0.6064 - val_accuracy: 0.6637
- val_f1: 0.6789209835282883 - val_auc: 0.734356896799645

Epoch 00007: val_accuracy did not improve from 0.67350

Epoch 00007: ReduceLROnPlateau reducing learning rate to 0.0731025
0028967857.
Epoch 8/10

Epoch 00008: LearningRateScheduler reducing learning rate to 0.073
10250401496887.
125/125 [==============================] - 0s 2ms/step - loss: 0.6
017 - accuracy: 0.6720 - val_loss: 0.6037 - val_accuracy: 0.6665
```

```
- val_f1: 0.6735193343122859 - val_auc: 0.7353853367593826

Epoch 00008: val_accuracy did not improve from 0.67350

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.0657922
5361347199.
```

Out[21]: <tensorflow.python.keras.callbacks.History at 0x7f2610a24190>

In [22]:
```
%tensorboard --logdir /content/logs_3/fit/20210225-074314
```

```
<IPython.core.display.Javascript object>
```

Model 4

In [23]:
```
def create_model4():
  ## weights initialization
  weights = tf.keras.initializers.HeUniform()
  return tf.keras.models.Sequential([
                                     Input(shape=(2,)),
                                     Dense(70, activation='relu', k
                                     Dense(57, activation='relu', k
                                     Dense(46, activation='relu', k
                                     Dense(27, activation='relu', k
                                     Dense(15, activation='relu', k
                                     Dense(1, activation='sigmoid',
  ])

model = create_model4()

#Callbacks
history_own = LossHistory()
filepath="content/model_save/4/weights-{epoch:02d}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accura
lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.9, p
es=EarlyStopping(monitor='val_accuracy',patience=2)


log_dir="logs_4/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H
tensorboard = TensorBoard(log_dir=log_dir,histogram_freq=1, write_g

# here we are creating a list with all the callbacks we want
callback_list = [tensorboard, history_own, checkpoint, lrschedule,

optimizer = tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.

model.compile(optimizer=optimizer, loss='binary_crossentropy',metri

model.fit(X_train,Y_train,epochs=50, validation_data=(X_test, Y_tes
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0
```

```
for the `TensorBoard` Callback.
Epoch 1/50

Epoch 00001: LearningRateScheduler reducing learning rate to 0.001
0000000474974513.
 3/16 [====>.........................] - ETA: 0s - loss: 0.7298 -
accuracy: 0.4979WARNING:tensorflow:Callback method `on_train_batch
_end` is slow compared to the batch time (batch time: 0.0052s vs `
on_train_batch_end` time: 0.0114s). Check your callbacks.
16/16 [==============================] - 1s 24ms/step - loss: 0.70
79 - accuracy: 0.5042 - val_loss: 0.6855 - val_accuracy: 0.5160
- val_f1: 0.22061191626409016 - val_auc: 0.6042681821307776

Epoch 00001: val_accuracy improved from -inf to 0.51600, saving mo
del to content/model_save/4/weights-01.hdf5
Epoch 2/50

Epoch 00002: LearningRateScheduler reducing learning rate to 0.001
0000000474974513.
16/16 [==============================] - 0s 9ms/step - loss: 0.686
4 - accuracy: 0.5118 - val_loss: 0.6796 - val_accuracy: 0.5560
- val_f1: 0.2896 - val_auc: 0.645264737325806

Epoch 00002: val_accuracy improved from 0.51600 to 0.55600, saving
model to content/model_save/4/weights-02.hdf5
Epoch 3/50

Epoch 00003: LearningRateScheduler reducing learning rate to 0.000
9500000451225787.
16/16 [==============================] - 0s 8ms/step - loss: 0.677
6 - accuracy: 0.5520 - val_loss: 0.6724 - val_accuracy: 0.5932
- val_f1: 0.4141159524666907 - val_auc: 0.6840379866334065

Epoch 00003: val_accuracy improved from 0.55600 to 0.59325, saving
model to content/model_save/4/weights-03.hdf5
Epoch 4/50

Epoch 00004: LearningRateScheduler reducing learning rate to 0.000
9500000160187483.
16/16 [==============================] - 0s 8ms/step - loss: 0.668
5 - accuracy: 0.5928 - val_loss: 0.6618 - val_accuracy: 0.5847
- val_f1: 0.3087806908031627 - val_auc: 0.7209039018199324

Epoch 00004: val_accuracy did not improve from 0.59325

Epoch 00004: ReduceLROnPlateau reducing learning rate to 0.0008550
000144168735.
Epoch 5/50

Epoch 00005: LearningRateScheduler reducing learning rate to 0.000
8549999911338091.
16/16 [==============================] - 0s 8ms/step - loss: 0.660
8 - accuracy: 0.6094 - val_loss: 0.6544 - val_accuracy: 0.6465
```

```
– val_f1: 0.5985235661555934 – val_auc: 0.7034736888172003

Epoch 00005: val_accuracy improved from 0.59325 to 0.64650, saving
model to content/model_save/4/weights-05.hdf5
Epoch 6/50

Epoch 00006: LearningRateScheduler reducing learning rate to 0.000
8122499915771186.
16/16 [==============================] - 0s 8ms/step - loss: 0.644
9 - accuracy: 0.6532 - val_loss: 0.6376 - val_accuracy: 0.6633
– val_f1: 0.6523870967741935 – val_auc: 0.7188227644716194

Epoch 00006: val_accuracy improved from 0.64650 to 0.66325, saving
model to content/model_save/4/weights-06.hdf5
Epoch 7/50

Epoch 00007: LearningRateScheduler reducing learning rate to 0.000
812250014860183.
16/16 [==============================] - 0s 8ms/step - loss: 0.628
6 - accuracy: 0.6604 - val_loss: 0.6217 - val_accuracy: 0.6685
– val_f1: 0.6570098292809105 – val_auc: 0.7260646072859813

Epoch 00007: val_accuracy improved from 0.66325 to 0.66850, saving
model to content/model_save/4/weights-07.hdf5
Epoch 8/50

Epoch 00008: LearningRateScheduler reducing learning rate to 0.000
812250014860183.
16/16 [==============================] - 0s 8ms/step - loss: 0.617
3 - accuracy: 0.6623 - val_loss: 0.6126 - val_accuracy: 0.6695
– val_f1: 0.6489644184811472 – val_auc: 0.7290407687354252

Epoch 00008: val_accuracy improved from 0.66850 to 0.66950, saving
model to content/model_save/4/weights-08.hdf5
Epoch 9/50

Epoch 00009: LearningRateScheduler reducing learning rate to 0.000
7716375141171738.
16/16 [==============================] - 0s 8ms/step - loss: 0.608
6 - accuracy: 0.6685 - val_loss: 0.6128 - val_accuracy: 0.6660
– val_f1: 0.6546018614270941 – val_auc: 0.7249810254390407

Epoch 00009: val_accuracy did not improve from 0.66950

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.0006944
737862795592.
Epoch 10/50

Epoch 00010: LearningRateScheduler reducing learning rate to 0.000
6944737979210913.
16/16 [==============================] - 0s 8ms/step - loss: 0.606
4 - accuracy: 0.6704 - val_loss: 0.6070 - val_accuracy: 0.6680
– val_f1: 0.6391304347826087 – val_auc: 0.7346891235440854
```

```
Epoch 00010: val_accuracy did not improve from 0.66950

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.0006250
264181289822.
```

Out[23]:
```
<tensorflow.python.keras.callbacks.History at 0x7f260ccbffd0>
```

In [24]:
```
%tensorboard --logdir /content/logs_4/fit/20210225-074432
```

```
<IPython.core.display.Javascript object>
```

Observation

- On changing activation function from tanh to relu, the loss has increased from 0.7182 to 0.6931
- On changing weights distribution from random to He, there is improvement in all the scores; loss, accuracy, F1 score and AUC score
- On changing optimizer as Adam and increasing batch size, there is improvement in validation accuracy and F1 score, and the smoothness has improved.