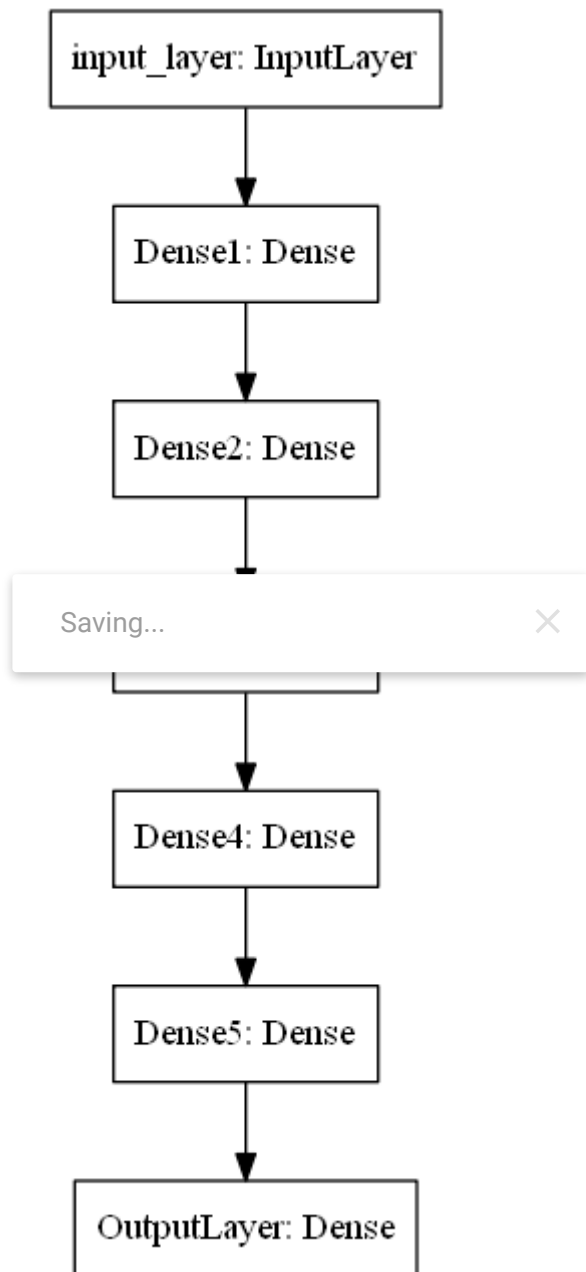


1. Download the data from [here](#). You have to use data.csv file for this assignment
2. Code the model to classify data like below image. You can use any number of units in your Dense layers.



### ▼ 3. Writing Callbacks

You have to implement the following callbacks

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

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

Cond1. If your validation accuracy at that epoch is less than previous epoch accuracy, decay your learning rate by 10%.

Cond2. For every 3rd epoch, decay your learning rate by 5%.

- If you are getting any NaN values(either weights 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)

```
1 pip install tensorflow scikeras scikit-learn
```

Saving...

Requirement already satisfied: tensorflow in /usr/local/lib/python3.7/dist-packages (2.9.0rc0)  
 Requirement already satisfied: scikeras in /usr/local/lib/python3.7/dist-packages (0.0.0)  
 Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (1.0.2)  
 Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.7/dist-packages (4.1.1)  
 Requirement already satisfied: tensorflow-estimator<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3.7/dist-packages (2.9.0rc0)  
 Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.7/dist-packages (0.4.0)  
 Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-packages (0.2.0)  
 Requirement already satisfied: tensorboard<2.10,>=2.9 in /usr/local/lib/python3.7/dist-packages (2.9.0rc0)  
 Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.7/dist-packages (1.24.3)  
 Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.7/dist-packages (0.23.1)  
 Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packages (3.3.0)  
 Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-packages (1.24.3)  
 Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages (3.10.0)  
 Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.7/dist-packages (1.4.0)  
 Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (21.3)  
 Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (59.0.0)  
 Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (1.14.0)  
 Requirement already satisfied: keras<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3.7/dist-packages (2.9.0rc0)  
 Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.7/dist-packages (1.1.2)  
 Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-packages (1.6.3)  
 Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (1.1.0)  
 Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.7/dist-packages (3.19.6)  
 Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.7/dist-packages (1.16.0)  
 Requirement already satisfied: flatbuffers<2,>=1.12 in /usr/local/lib/python3.7/dist-packages (1.12.0)  
 Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.7/dist-packages (13.0.0)  
 Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist-packages (0.23.0)  
 Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (1.5.2)  
 Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dist-packages (1.6.3)  
 Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (0.4.6)  
 Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (3.4.3)  
 Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.7/dist-packages (2.2.3)  
 Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.7/dist-packages (0.6.0)  
 Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (1.6.0)

```
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyasn1-modules<0.3,>=0.2.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: requests-oauthlib<2,>=0.7.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: importlib-metadata<4.4,>=4.4 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: zipp<0.5,>=0.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: certifi<2017.4.17 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: urllib3!=1.25.0,!<1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: oauthlib<3.0.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: threadpoolctl<2.0.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scipy<1.1.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib<0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages
```

```
1 import warnings
2 warnings.filterwarnings("ignore")
3 import pandas as pd#pandas to create small dataframes
4 # if numpy is not installed already : pip3 install numpy
5 import numpy as np#Do arithmetic operations on arrays
6 # matplotlib: used to plot graphs
```

Saving...

```
9 import seaborn as sns#PLOTS
10
11 from sklearn.model_selection import train_test_split
12 import tensorflow as tf
13 from tensorflow.keras.layers import Dense,Input,Activation
14 from tensorflow.keras.models import Model,Sequential
15 import random as rn
16 from tensorflow import keras
17 import datetime, os
18
19 from keras.callbacks import Callback
20 from keras.initializers import RandomUniform,HeUniform
21 from sklearn.metrics import roc_auc_score, f1_score
22 from scikeras.wrappers import KerasClassifier
23 from keras.optimizers import SGD
24 from tensorflow.keras.callbacks import TerminateOnNaN,ReduceLROnPlateau,EarlyStopping,L
25 from keras.callbacks import TensorBoard
```

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

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount(

```
1 data=pd.read_csv("drive/My Drive/Working-with-callbacks/data.csv")
2 data.head()
```

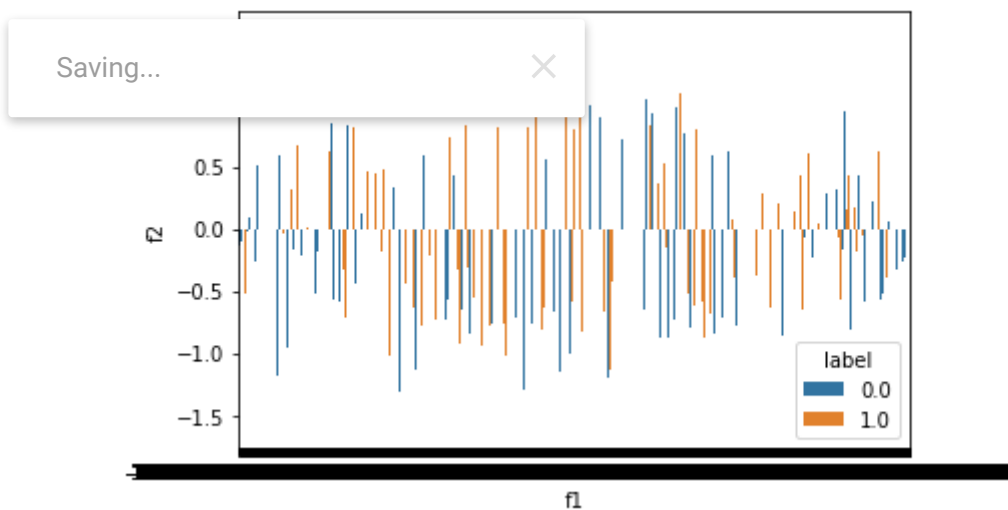
	f1	f2	label	
0	0.450564	1.074305	0.0	
1	0.085632	0.967682	0.0	
2	0.117326	0.971521	1.0	
3	0.982179	-0.380408	0.0	
4	-0.720352	0.955850	0.0	

```
1 y=data['label'].values
2 X=data[["f1","f2"]].values
```

```
1 set(y)
```

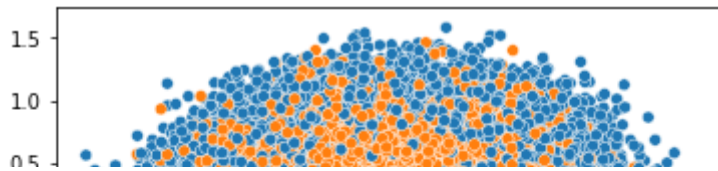
```
{0.0, 1.0}
```

```
1 sns.barplot(x = "f1",y= "f2",hue = "label",data = data)
2 plt.figure(figsize = (15,8))
3 plt.show()
```



<Figure size 1080x576 with 0 Axes>

```
1 sns.scatterplot(x = "f1",y= "f2",hue = "label",data = data)
2 plt.figure(figsize = (15,10))
3 plt.show()
```



```
1 data.shape
```

```
(20000, 3)
```



```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```



```
1 y_test.shape
```

```
(5000,)
```

### 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.

g process.

Saving...



```
1 class Metrics(tf.keras.callbacks.Callback):
2     def __init__(self):
3         self.validation_data=(X_test,y_test)
4     def on_train_begin(self, logs={}):
5         self.val_f1s = []
6     def on_epoch_end(self, epoch, logs={}):
7         val_predict = (np.asarray(self.model.predict(self.validation_data[0])))
8         val_target = self.validation_data[1]
9         val_predict_prob = np.argmax(val_predict,axis=1)
10        val_f1 = f1_score(val_target, val_predict_prob,average='micro')
11        roc_val = roc_auc_score(val_target, val_predict_prob)
12        self.val_f1s.append(val_f1)
13        print("f1 score :",val_f1,"AUC value :", roc_val)
14
15
```

```
1 # Stop training if NaN is encountered
2 class TerminateNaN(tf.keras.callbacks.Callback):
3     def on_epoch_end_loss(self, epoch, logs={}):
4         loss = self.model.get_loss()
5         if loss is not None:
6             if np.isnan(loss) or np.isinf(loss):
7                 print("Invalid loss and terminated at epoch {}".format(epoch))
```

```

8         self.model.stop_training = True
9
10    def on_epoch_end_weight(self,epoch,logs={}):
11        model_weights = self.model.get_weights()
12        if model_weights is not None:
13            if np.any([np.any(np.isnan(x)) for x in model_weights]):
14                print("Invalid weight and terminated at epoch {}".format(epoch))
15                self.model.stop_training = True
16
17    NanStop = TerminateNaN()

1 #For every 3rd epoch, decay your learning rate by 5%.
2 def schedule(epoch,lr):
3     if epoch % 3 == 0:
4         lr = lr - (lr*.05)
5         return lr
6     return lr
7
8 # Decrease learning rate by 5% for every 3rd epoch
9 LrScheduler = LearningRateScheduler(schedule,verbose=0)
10
11 # Decrease learning rate by 10%
12 LrValAccuracy = ReduceLROnPlateau(monitor='val_accuracy', patience=1, factor= 0.9, mode
13
14
15
16
17 #Save your model at every epoch if your validation accuracy is improved from previous e
18 filepath="drive/My Drive/Working-with-callbacks/best_model.hdf5"
19 CheckPoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy', verbose=1, sav
20

```

Saving...



your validation accuracy is not increased in last 2 e  
 = 'val\_accuracy', min\_delta=0.35, patience=2, verbose=1

Double-click (or enter) to edit

```

1 def build_model(activation_fun,initializer_val):
2     model = Sequential()
3     model.add(Dense(units = 2,kernel_initializer = initializer_val, activation = activati
4     model.add(Dense(units = 20,kernel_initializer = initializer_val, activation = activat
5     model.add(Dense(units = 20,kernel_initializer = initializer_val, activation = activat
6     model.add(Dense(units = 20,kernel_initializer = initializer_val, activation = activat
7     model.add(Dense(units = 20,kernel_initializer = initializer_val, activation = activat
8     model.add(Dense(units = 20,kernel_initializer = initializer_val, activation = activat
9     model.add(Dense(units = 1,kernel_initializer = initializer_val, activation = "sigmoid
10
11     return model
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

5 optimizer_1 = SGD(lr=0.01, momentum=0.9, nesterov=False, name='SGD')
6 model_1.compile(optimizer_1, loss='binary_crossentropy', metrics=['accuracy'])
7
8 metric_vals = Metrics()
9
10 tb = TensorBoard(log_dir='./Graph', histogram_freq=2, write_graph=True, write_images=True)
11 tb.set_model(model_1)
12
13 cb = [NanStop, LrValAccuracy, LrScheduler, CheckPoint, metric_vals, tb, EarlyStop]
14
15 model_1.fit(x = X_train, y = y_train, epochs=10, validation_data=(X_test, y_test), callback
16

```

Epoch 1/10

```

1/469 [.....] - ETA: 6:05 - loss: 3.7213 - accuracy: 0.562
454/469 [=====>.] - ETA: 0s - loss: 0.7765 - accuracy: 0.5205
Epoch 1: val_accuracy improved from -inf to 0.49460, saving model to drive/My Drive/v
157/157 [=====] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5

```

```

469/469 [=====] - 3s 5ms/step - loss: 0.7739 - accuracy: 0.5205

```

Epoch 2/10

```

468/469 [=====>.] - ETA: 0s - loss: 0.6964 - accuracy: 0.5051
Epoch 2: val_accuracy did not improve from 0.49460
157/157 [=====] - 0s 2ms/step
f1 score : 0.4942 AUC value : 0.5

```

```

469/469 [=====] - 2s 3ms/step - loss: 0.6964 - accuracy: 0.5051

```

Saving...

```

468/469 [=====>.] - ETA: 0s - loss: 0.6953 - accuracy: 0.4993
Epoch 3: val_accuracy improved from 0.49460 to 0.50960, saving model to drive/My Drive/v
157/157 [=====] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [=====] - 2s 4ms/step - loss: 0.6955 - accuracy: 0.5096
Epoch 3: early stopping
<keras.callbacks.History at 0x7fc077f75290>

```

```

1 print("TensorFlow version: ", tf.__version__)
2 # Load the TensorBoard notebook extension.
3 %load_ext tensorboard

```

TensorFlow version: 2.9.2

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

```
%reload_ext tensorboard
```

```
1 %tensorboard --logdir Graph
```



TensorBoard

SCALARS

GRAPHS

INACTIVE

- ☐ Show data download links
- ☐ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing

0.6

Horizontal Axis

STEP RELATIVE

WALL

Saving...

☐ ○ train

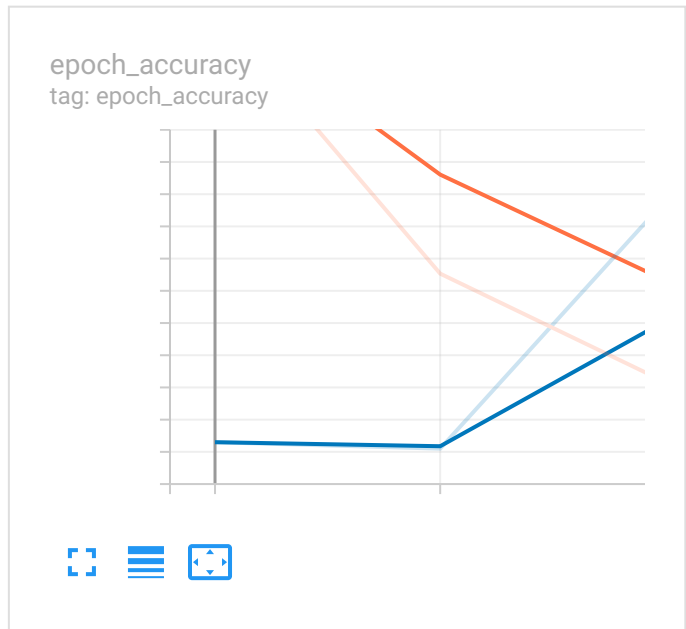
☐ ○ validation

TOGGLE ALL RUNS

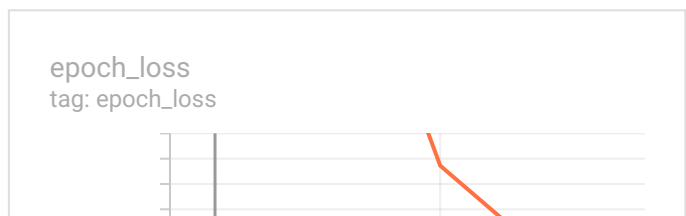
Graph

Filter tags (regular expressions supported)

epoch\_accuracy



epoch\_loss



```
1 train_acc_model_1 = model_1.evaluate(X_train, y_train, verbose=0)
2 test_acc_model_1 = model_1.evaluate(X_test, y_test, verbose=0)
3
4 train_accuracy_model_1 = train_acc_model_1[1]
5 test_accuracy_model_1 = test_acc_model_1[1]
6 train_loss_model_1 = train_acc_model_1[0]
7 test_loss_model_1 = test_acc_model_1[0]
8
9 print("Train data accuracy:", train_accuracy_model_1)
10 print("Test data accuracy:", test_accuracy_model_1)
```

Train data accuracy: 0.5006666779518127  
Test data accuracy: 0.5095999836921692

## 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.
3. Analyze your output and training process.

```

1 activation_fun_2 = "relu"
2 initializer_val_2 = RandomUniform(0,1)
3
4 # create model
5 model_2 = build_model(activation_fun_2,initializer_val_2)
6 optimizer_2 = SGD(lr=0.01, momentum=0.9, nesterov=False,name='SGD')
7 model_2.compile(optimizer_2,loss='binary_crossentropy', metrics=['accuracy'])
8
9 metric_vals = Metrics()
10
11 tb = TensorBoard(log_dir='./Graph', histogram_freq=2,write_graph=True, write_images=True)
12 tb.set_model(model_2)
13
14 cb = [NanStop,LrValAccuracy,LrScheduler,CheckPoint,metric_vals,tb,EarlyStop]
15
16 model_2.fit(x = X_train, y = y_train,epochs=10,validation_data=(X_test,y_test),callback

```

Saving...



```

Epoch 1/10
 1/469 [.....] - ETA: 5:22 - loss: 7665.0679 - accuracy: 0
459/469 [=====>.] - ETA: 0s - loss: 17.3913 - accuracy: 0.5046
Epoch 1: val_accuracy did not improve from 0.50960
157/157 [=====] - 0s 2ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [=====] - 3s 4ms/step - loss: 17.0440 - accuracy: 0
Epoch 2/10
456/469 [=====>.] - ETA: 0s - loss: 0.6933 - accuracy: 0.4985
Epoch 2: val_accuracy did not improve from 0.50960
157/157 [=====] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [=====] - 2s 4ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 3/10
465/469 [=====>.] - ETA: 0s - loss: 0.6933 - accuracy: 0.4985
Epoch 3: val_accuracy did not improve from 0.50960
157/157 [=====] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [=====] - 2s 4ms/step - loss: 0.6933 - accuracy: 0.4
Epoch 3: early stopping
<keras.callbacks.History at 0x7fc0595ec6d0>

```

```

1 train_acc_model_2 = model_2.evaluate(X_train, y_train, verbose=0)
2 test_acc_model_2 = model_2.evaluate(X_test, y_test, verbose=0)
3
4 train_accuracy_model_2 = train_acc_model_2[1]
5 test_accuracy_model_2 = test_acc_model_2[1]
6 train_loss_model_2 = train_acc_model_2[0]

```

```

7 test_loss_model_2 = test_acc_model_2[0]
8
9 print("Train data accuracy:",train_accuracy_model_2)
10 print("Test data accuracy:",test_accuracy_model_2)

```

```

Train data accuracy: 0.5019333362579346
Test data accuracy: 0.4941999912261963

```

### 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.

```

1 activation_fun_3 = "relu"
2 initializer_val_3 = HeUniform()
3 optimizer_3 = SGD(lr=0.01, momentum=0.9, nesterov=False, name='SGD')
4 model_3.compile(optimizer_3, loss='binary_crossentropy', metrics=['accuracy'])
5
6 metric_vals = Metrics()
7
8 tb = TensorBoard(log_dir='./Graph', histogram_freq=2, write_graph=True, write_images=True)
9 tb.set_model(model_3)
10
11 cb = [NanStop, LrValAccuracy, LrScheduler, CheckPoint, metric_vals, tb, EarlyStop]
12
13 model_3.fit(x = X_train, y = y_train, epochs=10, validation_data=(X_test, y_test), callback=cb)

```

Epoch 1/10

1/469 [.....] - ETA: 5:40 - loss: 0.9259 - accuracy: 0.437

463/469 [=====>.] - ETA: 0s - loss: 0.6760 - accuracy: 0.5723

Epoch 1: val\_accuracy improved from 0.50960 to 0.57840, saving model to drive/My Drive

157/157 [=====>.] - 0s 1ms/step

f1 score : 0.4942 AUC value : 0.5

469/469 [=====>.] - 3s 5ms/step - loss: 0.6756 - accuracy: 0.5723

Epoch 2/10

456/469 [=====>.] - ETA: 0s - loss: 0.6672 - accuracy: 0.5874

Epoch 2: val\_accuracy improved from 0.57840 to 0.58500, saving model to drive/My Drive

157/157 [=====>.] - 0s 1ms/step

f1 score : 0.4942 AUC value : 0.5

469/469 [=====>.] - 2s 4ms/step - loss: 0.6674 - accuracy: 0.5874

Epoch 3/10

460/469 [=====>.] - ETA: 0s - loss: 0.6645 - accuracy: 0.5916

Epoch 3: val\_accuracy improved from 0.58500 to 0.59280, saving model to drive/My Drive

157/157 [=====>.] - 0s 1ms/step

f1 score : 0.4942 AUC value : 0.5

```
469/469 [=====] - 2s 4ms/step - loss: 0.6644 - accuracy: 0.5
Epoch 3: early stopping
<keras.callbacks.History at 0x7fc058dc3650>
```

```
1 train_acc_model_3 = model_3.evaluate(X_train, y_train, verbose=0)
2 test_acc_model_3 = model_3.evaluate(X_test, y_test, verbose=0)
3
4 train_accuracy_model_3 = train_acc_model_3[1]
5 test_accuracy_model_3 = test_acc_model_3[1]
6 train_loss_model_3 = train_acc_model_3[0]
7 test_loss_model_3 = test_acc_model_3[0]
8
9 print("Train data accuracy:",train_accuracy_model_3)
10 print("Test data accuracy:",test_accuracy_model_3)
```

```
Train data accuracy: 0.5935999751091003
Test data accuracy: 0.5928000211715698
```

#### Model-4

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

Saving...

```
1 activation_fun_4 = "relu"
2 initializer_val_4 = RandomUniform(0,1)
3
4 # create model
5 model_4 = build_model(activation_fun_4,initializer_val_4)
6 optimizer_4 = SGD(lr=0.01, momentum=0.9, nesterov=False,name='adam')
7 model_4.compile(optimizer_4,loss='binary_crossentropy', metrics=['accuracy'])
8
9 metric_vals = Metrics()
10
11 tb = TensorBoard(log_dir='./Graph', histogram_freq=2,write_graph=True, write_images=True)
12 tb.set_model(model_4)
13
14 cb = [NanStop,LrValAccuracy,LrScheduler,CheckPoint,metric_vals,tb,EarlyStop]
15
16 model_4.fit(x = X_train, y = y_train,epochs=10,validation_data=(X_test,y_test),callback
```

Epoch 1/10

```
1/469 [.....] - ETA: 5:26 - loss: 14089.4746 - accuracy: 0.0
463/469 [=====] - ETA: 0s - loss: 31.1226 - accuracy: 0.4984
Epoch 1: val_accuracy did not improve from 0.59280
157/157 [=====] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [=====] - 3s 4ms/step - loss: 30.7494 - accuracy: 0
Epoch 2/10
```

```

454/469 [=====>.] - ETA: 0s - loss: 0.6934 - accuracy: 0.4986
Epoch 2: val_accuracy did not improve from 0.59280
157/157 [=====] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [=====] - 2s 3ms/step - loss: 0.6934 - accuracy: 0.4986
Epoch 3/10
455/469 [=====>.] - ETA: 0s - loss: 0.6934 - accuracy: 0.4917
Epoch 3: val_accuracy did not improve from 0.59280
157/157 [=====] - 0s 1ms/step
f1 score : 0.4942 AUC value : 0.5
469/469 [=====] - 2s 4ms/step - loss: 0.6934 - accuracy: 0.4917
Epoch 3: early stopping
<keras.callbacks.History at 0x7fc059586810>

```

```

1 train_acc_model_4 = model_4.evaluate(X_train, y_train, verbose=0)
2 test_acc_model_4 = model_4.evaluate(X_test, y_test, verbose=0)
3
4 train_accuracy_model_4 = train_acc_model_4[1]
5 test_accuracy_model_4 = test_acc_model_4[1]
6 train_loss_model_4 = train_acc_model_4[0]
7 test_loss_model_4 = test_acc_model_4[0]
8
9 print("Train data accuracy:",train_accuracy_model_4)
10 print("Test data accuracy:",test_accuracy_model_4)

```

Saving...



62579346

Test data accuracy: 0.4941999912261963

```

1 pretty_table = pd.DataFrame(columns = ['Optimizer','Initializer','Activation_fun','Train-Loss','Train-Accuracy','Test-Loss'])
2 pretty_table['Optimizer'] = ["SGD","SGD","SGD","Adam"]
3 pretty_table['Initializer'] = ["Random_Uniform","Random_Uniform","He_Uniform","Random_Uniform"]
4 pretty_table['Activation_fun'] = [activation_fun_1,activation_fun_2,activation_fun_3,activation_fun_4]
5 pretty_table['Train-Loss'] = [train_loss_model_1,train_loss_model_2,train_loss_model_3,train_loss_model_4]
6 pretty_table['Test-Loss'] = [test_loss_model_1,test_loss_model_2,test_loss_model_3,test_loss_model_4]
7 pretty_table['Train-Accuracy'] = [train_accuracy_model_1,train_accuracy_model_2,train_accuracy_model_3,train_accuracy_model_4]
8 pretty_table['Test-Accuracy'] = [test_accuracy_model_1,test_accuracy_model_2,test_accuracy_model_3,test_accuracy_model_4]
9 pretty_table

```

	Optimizer	Initializer	Activation_fun	Train-Loss	Train-Accuracy	Test-Loss
0	SGD	Random_Uniform	tanh	0.694091	0.500667	0.692691
1	SGD	Random_Uniform	relu	0.693317	0.501933	0.693668
2	SGD	He_Uniform	relu	0.663075	0.593600	0.664845
3	Adam	Random_Uniform	relu	0.693146	0.501933	0.693249

## 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.

After careful observations on each model we can say that adam (final model) gives good accuracy results than other optimizers because it can speed up the optimization by accelerating the gradient descent.

Instead of sigmoid, using an activation function such as ReLU to solve vanishing gradient problem in deep learning.

After clear observation model 2 and model 3 weight initializers with activation functions are most important consideration to get better accuracy of the model because combination can solve vanishing and exploding gradient problem in deep learning.

In model 2 we used random initializer with relu and SGD optimizer it gives around 50 percent accuracy. In model 3 we used he\_uniform initializer with relu and SGD optimizer it gives around 59 percent accuracy.

plots:

<https://tensorboard.dev/experiment/bE729AL8RdSqAMya>

Saving...



Colab paid products - [Cancel contracts here](#)

✓ 0s completed at 11:14 AM

