Name: Souri Yaswanth Krishna

Reg.No:19bci7070

LAB-2

Question 1: Use the above code of "KerasClassifier" on new dataset

mount drive

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

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

load the data

```
import pandas as pd
```

```
data=pd.read_csv('/content/drive/MyDrive/Deep Learning Lab/L2D2.csv',delimiter=',',
data.head()
```

	0	1	2	3	4	5	6	7	8	9	10
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	0.1609
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	0.4918
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	0.6333
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	0.0881
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	0.4152

```
data.dtypes
```

```
0 object
1 object
2 object
3 object
4 object...
56 object
57 object
58 object
59 object
60 object
Length: 61, dtype: object
```

Pre Process data set

```
data[60].replace(['R','M'],['0','1'],inplace=True)
data[60]
```

```
0 0
1 0
2 0
3 0
4 0 ..
203 1
204 1
205 1
206 1
207 1
Name: 60, Length: 208, dtype: object
```

```
## convert all the data columns into int64,float64
data=data.apply(pd.to_numeric,errors='coerce').fillna(0)
data.dtypes
```

```
0 float64
1 float64
2 float64
3 float64
4 float64 ...
56 float64
57 float64
58 float64
59 float64
60 int64
Length: 61, dtype: object
```

```
labels=['R','M']
labels
```

```
['R', 'M']
```

```
labels[data.iat[0,60]]
```

```
'R'
```

```
data.head()
```

	0	1	2	3	4	5	6	7	8	9	10
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	0.1609
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	0.4918
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	0.6333
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	0.0881
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	0.4152

Split data into X & y

```
X=data.drop(60,axis=1)
y=data[60]
```

```
X.head(),y.head()
```

```
( 0 1 2 3 ... 56 57 58 59 0 0.0200 0.0371 0.0428 0.0207 ... 0.0180
0.0084 0.0090 0.0032 1 0.0453 0.0523 0.0843 0.0689 ... 0.0140 0.0049 0.0052
0.0044 2 0.0262 0.0582 0.1099 0.1083 ... 0.0316 0.0164 0.0095
0.0078 3 0.0100 0.0171 0.0623 0.0205 ... 0.0050 0.0044 0.0040 0.0117
4 0.0762 0.0666 0.0481 0.0394 ... 0.0072 0.0048 0.0107 0.0094

 [5 rows x 60 columns], 0 0
 1 0
 2 0
 3 0
 4 0
 Name: 60, dtype: int64)
```

Split X & y into train and test data set

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

```
x_tr,x_te,y_tr,y_te=train_test_split(X,y,test_size=0.33,random_state=9)
```

```
x_tr.head(),y_tr.head()
```

```
( 0 1 2 3 ... 56 57 58 59 155 0.0211 0.0128 0.0015 0.0450 ... 0.0024
0.0039 0.0051 0.0015 116 0.0094 0.0333 0.0306 0.0376 ... 0.0078 0.0055
0.0091 0.0067 68 0.0195 0.0142 0.0181 0.0406 ... 0.0058 0.0042 0.0067
0.0012  67 0.0368 0.0403 0.0317 0.0293 ... 0.0135 0.0067 0.0078 0.0068
          101 0.0335 0.0134 0.0696 0.1180 ... 0.0032 0.0377 0.0126 0.0156

 [5 rows x 60 columns], 155 1
 116 1
 68 0
 67 0
 101 1
 Name: 60, dtype: int64)
```

KerasClassifier Model (MODEL-1)

```
import numpy as np
```

```
from keras.models import Sequential from keras.layers
import Dense from keras.wrappers.scikit_learn import
KerasClassifier from sklearn.model_selection import
StratifiedKFold from sklearn.model_selection import
cross_val_score import numpy
```

```
seed = 7

numpy.random.seed(seed)

def create_model():
model = Sequential()
    model.add(Dense(12, input_dim=60, kernel_initializer='uniform', activation='rel
    model.add(Dense(8, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy
    return model
    kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
model_1 = KerasClassifier(build_fn=create_model, epochs=15,
batch_size=10) results = cross_val_score(model_1, x_tr, y_tr, cv=kfold)
```

```
 Epoch 1/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6931 - accuracy
 Epoch 2/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6926 - accuracy
 Epoch 3/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6918 - accuracy
 Epoch 4/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6910 - accuracy
 Epoch 5/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6892 - accuracy
 Epoch 6/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6871 - accuracy

 Epoch 7/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6839 - accuracy
 Epoch 8/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6809 - accuracy
 Epoch 9/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6758 - accuracy
 Epoch 10/15
 Epoch 10/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6708 - accuracy
 Epoch 11/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6663 - accuracy
 Epoch 12/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6618 - accuracy
 Epoch 13/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6599 - accuracy
 Epoch 14/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6525 - accuracy
 Epoch 15/15
 13/13 [==============================]  - 0s 2ms/step - loss: 0.6472 - accuracy
 2/2 [==============================] - 0s 8ms/step - loss: 0.6758 - accuracy:
 Epoch 1/15
 13/13 [==============================] - 0s 2ms/step - loss: 0.6929 - accuracy
 Epoch 2/15
 13/13 [==============================] - 0s 2ms/step - loss: 0.6921 - accuracy
 Epoch 3/15
 13/13 [==============================] - 0s 2ms/step - loss: 0.6910 - accuracy
 Epoch 4/15
 13/13 [==============================] - 0s 2ms/step - loss: 0.6893 - accuracy
 Epoch 5/15
 13/13 [==============================] - 0s 2ms/step - loss: 0.6873 - accuracy
 Epoch 6/15
 13/13 [==============================] - 0s 2ms/step - loss: 0.6853 - accuracy
```

```
Epoch 7/15
13/13 [==============================]  - 0s 2ms/step - loss: 0.6827 - accuracy
Epoch 8/15
13/13 [==============================]  - 0s 2ms/step - loss: 0.6797 - accuracy
Epoch 9/15
13/13 [==============================]  - 0s 2ms/step - loss: 0.6768 - accuracy
Epoch 10/15
13/13 [==============================]  - 0s 3ms/step - loss: 0.6749 - accuracy
Epoch 11/15
13/13 [==============================]  - 0s 2ms/step - loss: 0.6725 - accuracy
Epoch 12/15
13/13 [==============================]  - 0s 2ms/step - loss: 0.6702 - accuracy
Epoch 13/15
13/13 [==============================]  - 0s 2ms/step - loss: 0.6682 - accuracy
Epoch 14/15
13/13 [==============================]  - 0s 2ms/step - loss: 0.6664 - accuracy
E h 15/15
```

Evaluate MODEL-1

```
score_1=print(results.mean()*100)
score_1
```

```
57.582420110702515
```

Conclusion:

The Accuracy obtained from MODEL-1 is : 57.5%*

Question 2: Check whether accuracy will improve with the larger network

The Previous model's mean accuracy is **55%** so in order to improve model-1

I consider following steps:

for the existing hidden layer i've increased the hidden units by 5x .

added another hidden layer with 20 hidden units

MODEL-2

```
## SET UP RANDOM

SEED

np.random.seed(9)

## CREATE MODEL-2

model_2=Sequential([           Dense(60, input_dim=60,
kernel_initializer='uniform', activation='relu'),
          Dense(40, kernel_initializer='uniform', activation='relu'),
          Dense(20, kernel_initializer='uniform', activation='relu'),
          Dense(1, kernel_initializer='uniform', activation='sigmoid') ])
```

```
## COMPILE THE MODEL model_2.compile(
loss='binary_crossentropy',
          optimizer='adam',
          metrics=['accuracy']
)

## fit the model
model_2.fit(x_tr,
y_tr,
           epochs=150)
     5/5 [ ] 0s 4ms/step loss: 0.2123 accuracy:  Epoch 121/150 5/5
     [==============================] - 0s 3ms/step - loss: 0.2134 - accuracy:
     Epoch 122/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.2112 - accuracy:
     Epoch 123/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.2098 - accuracy:
     Epoch 124/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.2132 - accuracy:
     Epoch 125/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.2101 - accuracy:
     Epoch 126/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.2126 - accuracy:
     Epoch 127/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.2027 - accuracy:
              /
     Epoch 128/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.1996 - accuracy:
     Epoch 129/150 5/5 [==============================] - 0s 3ms/step - loss:
      0.1994 - accuracy:

     Epoch 130/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.1984 - accuracy:
     Epoch 131/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.2007 - accuracy:
     Epoch 132/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.1965 - accuracy:
     Epoch 133/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.1933 - accuracy:
     Epoch 134/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.1916 - accuracy:
     Epoch 135/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.1908 - accuracy:
     Epoch 136/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.1918 - accuracy:
     Epoch 137/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.1916 - accuracy:
     Epoch 138/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.1997 - accuracy:
     Epoch 139/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.1948 - accuracy:
     Epoch 140/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.1837 - accuracy:
     Epoch 141/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.1826 - accuracy:
     Epoch 142/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.1795 - accuracy:
     Epoch 143/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.1847 - accuracy:
     Epoch 144/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.1786 - accuracy:
     Epoch 145/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.1791 - accuracy:
```

```
Epoch 146/150 5/5 [==============================] - 0s 5ms/step - loss:
0.1774 - accuracy:
Epoch 147/150 5/5 [==============================] - 0s 4ms/step - loss:
0.1746 - accuracy:
Epoch 148/150 5/5 [==============================] - 0s 4ms/step - loss:
0.1778 - accuracy:
Epoch 149/150 5/5 [==============================] - 0s 4ms/step - loss:
0.1738 - accuracy:
Epoch 150/150
```

Evaluate MODEL-2

```
score_2=model_2.evaluate(x_te,y_te)
print("%s: %.2f%%" % (model_2.metrics_names[1], score_2[1]*100))
```

```
3/3 [==============================] - 0s 4ms/step - loss: 0.4453 - accuracy:
accuracy: 78.26%
```

```
model_2.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
dense_30 (Dense) (None, 60) 3660
_________________________________________________________________
dense_31 (Dense) (None, 40) 2440
_________________________________________________________________
dense_32 (Dense) (None, 20) 820
_________________________________________________________________
dense_33 (Dense) (None, 1) 21
=================================================================
Total params: 6,941
Trainable params: 6,941
Non-trainable params: 0
_________________________________________________________________
```

Conclusion:

MODEL-2 accuracy is **78.26%**

With the increase of hidden units as well as addition of new layer the model's performance improved

Question 3: Check whether accuracy will be improved after Dropout or not

MODEL-3

This model is similar to MODEL-2 but here i am additionally adding drop out after each hidden layer.

```
from keras.layers import Dropout


## SET UP RANDOM

SEED

np.random.seed(9) ##

CREATE MODEL-3

model_3=Sequential([           Dense(60, input_dim=60,
kernel_initializer='uniform', activation='relu'),
          Dense(40, kernel_initializer='uniform', activation='relu'),
          Dropout(0.2,seed=9),
          Dense(20, kernel_initializer='uniform', activation='relu'),
          Dropout(0.2,seed=9),
          Dense(1, kernel_initializer='uniform', activation='sigmoid')
])
## COMPILE THE MODEL model_3.compile(
loss='binary_crossentropy',
           optimizer='adam',
           metrics=['accuracy']
)

## fit the model
model_3.fit(x_tr,
y_tr,
            epochs=150)
     5/5 [==============================] - 0s 5ms/step - loss: 0.2553 - accuracy:
     Epoch 121/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.2864 - accuracy:
     Epoch 122/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.2576 - accuracy:
     Epoch 123/150
     5/5 [==============================] - 0s 4ms/step - loss: 0.2672 - accuracy:
     Epoch 124/150 5/5 [==============================] - 0s 5ms/step - loss:
     0.2535 - accuracy:
     Epoch 125/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.2373 - accuracy:
     Epoch 126/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.2573 - accuracy:
     Epoch 127/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.2697 - accuracy:
     Epoch 128/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.2294 - accuracy:
     Epoch 129/150 5/5 [==============================] - 0s 5ms/step - loss:
     0.2237 - accuracy:
     Epoch 130/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.2236 - accuracy:
     Epoch 131/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.2507 - accuracy:
     Epoch 132/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.2138 - accuracy:
     Epoch 133/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.2083 - accuracy:
     Epoch 134/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.2229 - accuracy:
     Epoch 135/150 5/5 [==============================] - 0s 4ms/step - loss:
     0.2158 - accuracy:
```

```
Epoch 136/150 5/5 [==============================] - 0s 5ms/step - loss:
0.2361 - accuracy:
Epoch 137/150 5/5 [==============================] - 0s 4ms/step - loss:
0.2240 - accuracy:
Epoch 138/150 5/5 [==============================] - 0s 4ms/step - loss:
0.2418 - accuracy:
Epoch 139/150 5/5 [==============================] - 0s 3ms/step - loss:
0.2219 - accuracy:
Epoch 140/150 5/5 [==============================] - 0s 3ms/step - loss:
0.2186 - accuracy:
Epoch 141/150 5/5 [==============================] - 0s 4ms/step - loss:
0.2338 - accuracy:
Epoch 142/150 5/5 [==============================] - 0s 4ms/step - loss:
0.1992 - accuracy:
Epoch 143/150 5/5 [==============================] - 0s 4ms/step - loss:
0.2208 - accuracy:
Epoch 144/150 5/5 [==============================] - 0s 3ms/step - loss:
0.2365 - accuracy:
         /
Epoch 145/150 5/5 [==============================] - 0s 4ms/step - loss:
0.2068 - accuracy:
Epoch 146/150 5/5 [==============================] - 0s 4ms/step - loss:
 0.2010 - accuracy:
Epoch 147/150 5/5 [==============================] - 0s 4ms/step - loss:
0.2064 - accuracy:
Epoch 148/150 5/5 [==============================] - 0s 4ms/step - loss:
0.2021 - accuracy:
Epoch 149/150 5/5 [==============================] - 0s 3ms/step - loss:
0.2005 - accuracy:
Epoch 150/150
```

Evaluate MODEL-3

```
model_3.evaluate(x_te,y_te)
     3/3 [==============================] - 0s 4ms/step - loss: 0.4361 - accuracy:
     [0.43607544898986816, 0.8115941882133484]


model_3.summary()

     Model: "sequential_11"
     _________________________________________________________________
     Layer (type) Output Shape Param #
     =================================================================
     dense_34 (Dense) (None, 60) 3660
     _________________________________________________________________
     dense_35 (Dense) (None, 40) 2440
     _________________________________________________________________
     dropout (Dropout) (None, 40) 0
     _________________________________________________________________
     dense_36 (Dense) (None, 20) 820
     _________________________________________________________________
     dropout_1 (Dropout) (None, 20) 0
     _________________________________________________________________
     dense_37 (Dense) (None, 1) 21
     =================================================================
     Total params: 6,941
     Trainable params: 6,941
     Non-trainable params: 0
```

Conclusion:

MODEL-3's accuracy is **~81.6%**

After Implementing **Dropout** "MODEL-3's" accuracy slightly increased.

Question 4: Check whether the accuracy improved or not with decay learning rate; USe SGD optimizer

Decay = Learning Rate/ Number of epochs

MODEL-4

```
import numpy as np import
tensorflow as tf from
tensorflow import keras


## SET UP RANDOM

SEED

np.random.seed(9) ##

CREATE MODEL-4

model_4=Sequential([
          Dense(60, input_dim=60, kernel_initializer='uniform', activation='relu'),
Dense(40, kernel_initializer='uniform', activation='relu'),
          Dropout(0.2,seed=9),
          Dense(20, kernel_initializer='uniform', activation='relu'),
          Dropout(0.2,seed=9),
          Dense(1, kernel_initializer='uniform', activation='sigmoid')
])

## Learning rate scheduler
for epoch in range(1,150):
learning_rate = 0.1
  decay_rate = learning_rate / epoch
  sgd = tf.keras.optimizers.SGD(lr=learning_rate,decay=decay_rate)

## COMPILE THE MODEL model_4.compile(
loss='binary_crossentropy',
           optimizer = sgd,
           metrics=['accuracy']
)
```

```
## fit the model
history_4=model_4.fit(x_tr,
y_tr,
                      epochs=150)
        [ ] p y Epoch 19/150 5/5 [==============================] - 0s 3ms/step
    - loss: 0.6819 - accuracy:
    Epoch 20/150 5/5 [==============================] - 0s 3ms/step - loss:
    0.6821 - accuracy:
    Epoch 21/150 5/5 [==============================] - 0s 4ms/step - loss:
    0.6820 - accuracy:
    Epoch 22/150 5/5 [==============================] - 0s 3ms/step - loss:
    0.6819 - accuracy:
    Epoch 23/150 5/5 [==============================] - 0s 3ms/step - loss:
     0.6818 - accuracy:
    5/5 [ ] 0s 3ms/step loss: 0.6818 accuracy:  Epoch 24/150 5/5
    [==============================] - 0s 4ms/step - loss: 0.6818 - accuracy:
    Epoch 25/150 5/5 [==============================] - 0s 4ms/step - loss:
    0.6824 - accuracy:
    Epoch 26/150 5/5 [==============================] - 0s 4ms/step - loss:
    0.6819 - accuracy:
    Epoch 27/150 5/5 [==============================] - 0s 6ms/step - loss:
    0.6819 - accuracy:
    Epoch 28/150 5/5 [==============================] - 0s 4ms/step - loss:
    0.6819 - accuracy:
    Epoch 29/150 5/5 [==============================] - 0s 3ms/step - loss:
    0.6820 - accuracy:
    Epoch 30/150 5/5 [==============================] - 0s 4ms/step - loss:
    0.6821 - accuracy:
    Epoch 31/150 5/5 [==============================] - 0s 3ms/step - loss:
    0.6817 - accuracy:
    Epoch 32/150 5/5 [==============================] - 0s 4ms/step - loss:
    0.6820 - accuracy:
    Epoch 33/150 5/5 [==============================] - 0s 3ms/step - loss:
    0.6819 - accuracy:
    Epoch 34/150 5/5 [==============================] - 0s 3ms/step - loss: 0.6820
      - accuracy:
    Epoch 35/150 5/5 [==============================] - 0s 4ms/step - loss:
    0.6829 - accuracy:
    Epoch 36/150 5/5 [==============================] - 0s 3ms/step - loss:
    0.6819 - accuracy:
    Epoch 37/150 5/5 [==============================] - 0s 3ms/step - loss:
    0.6821 - accuracy:
    Epoch 38/150 5/5 [==============================] - 0s 4ms/step - loss:
    0.6818 - accuracy:
    Epoch 39/150 5/5 [==============================] - 0s 3ms/step - loss:
    0.6821 - accuracy:
    Epoch 40/150 5/5 [==============================] - 0s 3ms/step - loss:
    0.6817 - accuracy:
    Epoch 41/150 5/5 [==============================] - 0s 4ms/step - loss:
    0.6818 - accuracy:
    Epoch 42/150 5/5 [==============================] - 0s 3ms/step - loss:
    0.6818 - accuracy:
    Epoch 43/150 5/5 [==============================] - 0s 2ms/step - loss:
    0.6819 - accuracy:
    Epoch 44/150 5/5 [==============================] - 0s 4ms/step - loss:
    0.6821 - accuracy:
    Epoch 45/150 5/5 [==============================] - 0s 3ms/step - loss:
    0.6818 - accuracy:
    Epoch 46/150 5/5 [==============================] - 0s 3ms/step - loss:
    0.6820 - accuracy:
    Epoch 47/150 5/5 [==============================] - 0s 5ms/step - loss:
    0.6817 - accuracy:
```

```
Epoch 48/150
```

Evaluate the model

```
model_4.evaluate(x_te,y_te)
```

```
3/3 [==============================] - 0s 4ms/step - loss: 0.7204 - accuracy:
[0.7204493284225464, 0.4492753744125366]
```

```
model_4.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
dense_42 (Dense) (None, 60) 3660
_________________________________________________________________
dense_43 (Dense) (None, 40) 2440
_________________________________________________________________
dropout_4 (Dropout) (None, 40) 0
_________________________________________________________________
dense_44 (Dense) (None, 20) 820
_________________________________________________________________
dropout_5 (Dropout) (None, 20) 0
_________________________________________________________________
dense_45 (Dense) (None, 1) 21
=================================================================
Total params: 6,941
Trainable params: 6,941
Non-trainable params: 0
_________________________________________________________________
```

Conclusion:

MODEL-4's accuracy is **~45%**

The accuracy dropped when decay learning rate implemented.

check 0s completed at 22:38