Case Study 6 (Unit 12): Evaluate Neural Nets using Keras

Team: Hieu Nguyen, Nithya Devadoss, Ramesh Simhambhatla, Ramya Mandava

Date: 11/18/2018

Abstract

In Artifical Intelligence, Artificial Neural Networks (ANN) are computing systems vaguely inspired by the biological neural networks that constitute animal brains. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. In this case study, we will leverage Keras API, a python package, capable of fast experimentation for deep learning training and testing. We will experiment with a subset of Higgs Boson experimental data acquired from UCI Machine Learning repository. We will use ROC AUC (Area under ROC Curve, which provides an aggregate measure of performance across all possible classification thresholds) as our model performance metric to measure and compare our models. We will experiment with varying number of neurons, hidden layers, activation functions, kernel initializers, optimizers and various other hyper parameters such as epochs, batch sizes, learning rates etc. We will compare the performance results between each of these model in order to extract best model and conlcude our findings.

Introduction

The Higgs boson is an elementary particle in the Standard Model of particle physics, produced by the quantum excitation of the Higgs field, one of the fields in particle physics theory [1].

The data set is acquired from UCI Machine Learning repository: https://archive.ics.uci.edu/ml/datasets/HIGGS (https://archive.ics.uci.edu/ml/datasets/HIGGS)

Data Set Information:

The data has been produced using Monte Carlo simulations. The first 21 features (columns 2-22) are kinematic properties measured by the particle detectors in the accelerator. The last seven features are functions of the first 21 features; these are high-level features derived by physicists to help discriminate between the two classes. There is an interest in using deep learning methods to obviate the need for physicists to manually develop such features. Benchmark results using Bayesian Decision Trees from a standard physics package and 5-layer neural networks are presented in the original paper. The last 500,000 examples are used as a test set.

Attribute Information:

The first column is the class label (1 for signal, 0 for background), followed by the 28 features (21 low-level features then 7 high-level features): lepton pT, lepton eta, lepton phi, missing energy magnitude, missing energy phi, jet 1 pt, jet 1 eta, jet 1 phi, jet 1 b-tag, jet 2 pt, jet 2 eta, jet 2 phi, jet 2 b-tag, jet 3 pt, jet 3 eta, jet 3 phi, jet 3 b-tag, jet 4 pt, jet 4 eta, jet 4 phi, jet 4 b-tag, m_jj, m_jjj, m_lv, m_jlv, m_bb, m_wbb, m_wwbb. For more detailed information about each feature see the original paper.

In [41]:
```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import SGD
from keras.optimizers import Adam
from keras.optimizers import Adagrad
from keras.optimizers import RMSprop
from keras.optimizers import Adamax
from sklearn.metrics import roc_auc_score
```

We will use the data set of 1100000 observations to train and use 60000 observations to test the model

In [42]:
```
N=1100000. #Change this line adjust the number of rows.
data=pd.read_csv("HIGGS.csv",nrows=N,header=None)
test_data=pd.read_csv("HIGGS.csv",nrows=60000,header=None,skiprows=1100000)
```

In [43]:
```
def size_it (df):
    total=0
    for i in range(len(data.memory_usage())):
        total=total+data.memory_usage()[i]
    size=total/(2**20)
    print("%.2f Megabytes" % size)
```

In [44]:
```
size_it(data)
data.shape
```

```
251.77 Megabytes
```

Out[44]:
```
(1100000, 29)
```

In [45]:
```
size_it(test_data)
test_data.shape
```

```
251.77 Megabytes
```

Out[45]:
```
(60000, 29)
```

In [46]:
```
data.columns=['label','lepton pT', 'lepton eta', 'lepton phi', 'missing energy magnitude', 'missing e
nergy phi', 'jet 1 pt',
             'jet 1 eta', 'jet 1 phi', 'jet 1 b-tag', 'jet 2 pt', 'jet 2 eta', 'jet 2 phi', 'jet 2 b-
tag', 'jet 3 pt',
             'jet 3 eta', 'jet 3 phi', 'jet 3 b-tag', 'jet 4 pt', 'jet 4 eta', 'jet 4 phi', 'jet 4 b-
tag', 'm_jj',
             'm_jjj', 'm_lv','m_jlv', 'm_bb', 'm_wbb', 'm_wwbb']
```

In [47]:
```
test_data.columns=['label','lepton pT', 'lepton eta', 'lepton phi', 'missing energy magnitude', 'missing energy phi', 'jet 1 pt',
             'jet 1 eta', 'jet 1 phi', 'jet 1 b-tag', 'jet 2 pt', 'jet 2 eta', 'jet 2 phi', 'jet 2 b-tag', 'jet 3 pt',
             'jet 3 eta', 'jet 3 phi', 'jet 3 b-tag', 'jet 4 pt', 'jet 4 eta', 'jet 4 phi', 'jet 4 b-tag', 'm_jj',
             'm_jjj', 'm_lv','m_jlv', 'm_bb', 'm_wbb', 'm_wwbb']
```

In [48]:
```
data.tail()
```

Out[48]:

	label	lepton pT	lepton eta	lepton phi	missing energy magnitude	missing energy phi	jet 1 pt	jet 1 eta	jet 1 phi	jet 1 b-tag	...	jet 4 eta
1099995	0.0	1.147101	-0.290297	-0.502390	0.787117	-0.115922	1.501992	-0.961539	-0.940719	2.173076	...	0.785724
1099996	1.0	1.078290	0.090525	-1.113295	0.828900	0.153260	1.389315	-0.565447	0.124895	0.000000	...	0.542540
1099997	1.0	0.915960	0.174286	-0.096232	1.543762	0.596144	0.664335	-0.476326	-1.245072	1.086538	...	-0.108728
1099998	1.0	0.585263	-0.882470	-1.682583	0.990881	0.796417	1.032413	-0.100039	-0.312609	0.000000	...	-0.585103
1099999	0.0	1.175833	0.074941	1.691634	0.293551	-0.324434	0.932928	2.726078	0.404855	0.000000	...	1.120519

5 rows × 29 columns

In [49]:
```
test_data.tail()
```

Out[49]:

	label	lepton pT	lepton eta	lepton phi	missing energy magnitude	missing energy phi	jet 1 pt	jet 1 eta	jet 1 phi	jet 1 b-tag	...	jet 4 eta	j
59995	0.0	0.370959	1.178448	-0.839749	0.772050	1.058658	0.807243	0.743637	-0.260498	2.173076	...	0.321009	0.:
59996	0.0	0.971595	-1.077263	1.713273	0.314970	0.231261	0.641067	-0.844692	0.510741	0.000000	...	-0.909071	1.:
59997	1.0	1.003622	-0.219197	1.164516	0.555159	0.221953	1.710307	0.027701	1.581794	0.000000	...	-0.401049	-0
59998	0.0	1.891765	0.042800	-0.227179	0.876925	0.322577	0.782142	2.035887	-1.363153	0.000000	...	1.459478	1.(
59999	1.0	0.989347	-1.482435	-0.797579	0.924406	0.210223	0.481304	-1.085317	1.071214	2.173076	...	0.146949	1.:

5 rows × 29 columns

In [50]:
```
data.iloc[:,0:10].describe()
```

Out[50]:

	label	lepton pT	lepton eta	lepton phi	missing energy magnitude	missing energy phi	jet 1 pt	jet 1 e
count	1.100000e+06	1.100000e+06	1.100000e+06	1.100000e+06	1.100000e+06	1.100000e+06	1.100000e+06	1.100000e+0
mean	5.295173e-01	9.913837e-01	9.533126e-04	-7.382432e-04	9.981465e-01	-7.666476e-04	9.906847e-01	-7.511484e-0
std	4.991282e-01	5.649399e-01	1.008487e+00	1.005848e+00	5.991570e-01	1.006687e+00	4.751833e-01	1.010139e+0
min	0.000000e+00	2.746966e-01	-2.434976e+00	-1.742508e+00	6.259872e-04	-1.743944e+00	1.386017e-01	-2.969725e+(
25%	0.000000e+00	5.907533e-01	-7.363746e-01	-8.719308e-01	5.762637e-01	-8.717909e-01	6.788095e-01	-6.882352e-0
50%	1.000000e+00	8.535544e-01	9.198132e-04	9.714414e-04	8.915848e-01	-1.158754e-03	8.942697e-01	-1.015666e-0
75%	1.000000e+00	1.236592e+00	7.391881e-01	8.693294e-01	1.293202e+00	8.711392e-01	1.170740e+00	6.871941e-01
max	1.000000e+00	8.711782e+00	2.434868e+00	1.743236e+00	9.900929e+00	1.743257e+00	8.382610e+00	2.969674e+0

In [51]:
```
data.iloc[:,10:20].describe()
```

Out[51]:

	jet 2 pt	jet 2 eta	jet 2 phi	jet 2 b-tag	jet 3 pt	jet 3 eta	jet 3 phi	jet 3 b-ta
count	1.100000e+06	1.100000e+06	1.100000e+06	1.100000e+06	1.100000e+06	1.100000e+06	1.100000e+06	1.100000e+(
mean	9.930826e-01	1.694100e-03	2.226422e-04	1.000497e+00	9.924060e-01	2.097878e-03	6.820779e-05	9.992596e-(
std	5.000212e-01	1.008805e+00	1.006862e+00	1.049272e+00	4.867730e-01	1.008249e+00	1.005805e+00	1.193635e+(
min	1.889811e-01	-2.913090e+00	-1.742372e+00	0.000000e+00	2.636076e-01	-2.729663e+00	-1.742069e+00	0.000000e+(
25%	6.573421e-01	-6.925291e-01	-8.707339e-01	0.000000e+00	6.512039e-01	-6.970776e-01	-8.711343e-01	0.000000e+(
50%	8.906413e-01	1.031646e-03	3.514990e-04	0.000000e+00	8.977762e-01	2.903640e-03	-7.519117e-04	0.000000e+(
75%	1.202001e+00	6.965352e-01	8.715371e-01	2.214872e+00	1.222500e+00	7.019747e-01	8.708400e-01	2.548224e+(
max	1.164708e+01	2.913210e+00	1.743175e+00	2.214872e+00	8.864838e+00	2.730009e+00	1.742884e+00	2.548224e+(

In [52]:
```
types={}
for i in data.columns:
    types[i]=data[i].dtype
```

In [53]:
```
types
```

Out[53]:
```
{'jet 1 b-tag': dtype('float64'),
 'jet 1 eta': dtype('float64'),
 'jet 1 phi': dtype('float64'),
 'jet 1 pt': dtype('float64'),
 'jet 2 b-tag': dtype('float64'),
 'jet 2 eta': dtype('float64'),
 'jet 2 phi': dtype('float64'),
 'jet 2 pt': dtype('float64'),
 'jet 3 b-tag': dtype('float64'),
 'jet 3 eta': dtype('float64'),
 'jet 3 phi': dtype('float64'),
 'jet 3 pt': dtype('float64'),
 'jet 4 b-tag': dtype('float64'),
 'jet 4 eta': dtype('float64'),
 'jet 4 phi': dtype('float64'),
 'jet 4 pt': dtype('float64'),
 'label': dtype('float64'),
 'lepton eta': dtype('float64'),
 'lepton pT': dtype('float64'),
 'lepton phi': dtype('float64'),
 'm_bb': dtype('float64'),
 'm_jj': dtype('float64'),
 'm_jjj': dtype('float64'),
 'm_jlv': dtype('float64'),
 'm_lv': dtype('float64'),
 'm_wbb': dtype('float64'),
 'm_wwbb': dtype('float64'),
 'missing energy magnitude': dtype('float64'),
 'missing energy phi': dtype('float64')}
```

We further examined the data types of all the features to optimize the memory consumption in roder to improve computational process. We found all the features are of type float, and so we did not change them any futher.

Splitting the data into features and output as well test data.

In [54]:
```
y = np.array(data.iloc[:,0])
x = np.array(data.iloc[:,1:])
```

In [55]:
```
x_test = np.array(test_data.iloc[:,1:])
y_test = np.array(test_data.iloc[:,0])
```

1. Pick 3 or more different architectures (add/subtract layers+neurons) and run the model + score.

1.1 Architecture1: 1 hidden layer with 100 nodes - Base Model

We will begin with our starter model with 1 hidden layer that has 100 neurons and signmoid activation functions

In [56]:
```
test_results = {}
arch_name = "sigmoid-1layer-100nodes"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('sigmoid'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 6s 5us/step - loss: 0.6627 - acc: 0.5928
Epoch 2/5
1100000/1100000 [==============================] - 5s 5us/step - loss: 0.6454 - acc: 0.6213
Epoch 3/5
1100000/1100000 [==============================] - 5s 4us/step - loss: 0.6356 - acc: 0.6349
Epoch 4/5
1100000/1100000 [==============================] - 4s 4us/step - loss: 0.6279 - acc: 0.6447
Epoch 5/5
1100000/1100000 [==============================] - 5s 4us/step - loss: 0.6211 - acc: 0.6532

ROC AUC Score for sigmoid-1layer-100nodes is: 0.7320651934415888
```

1.2 Architecture 2: 2 hidden layers with 100 nodes each

In [57]:
```
arch_name = "sigmoid-2layers-100nodes"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('sigmoid'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('sigmoid'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.6923 - acc: 0.5248
Epoch 2/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.6841 - acc: 0.5484
Epoch 3/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.6557 - acc: 0.6062
Epoch 4/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.6473 - acc: 0.6184
Epoch 5/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.6421 - acc: 0.6234

ROC AUC Score for sigmoid-2layers-100nodes is: 0.6863232135063109
```

1.3 Architecture 3: 3 hidden layers with 100 nodes each

In [58]:
```
arch_name = "sigmoid-3layers-100nodes"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('sigmoid'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('sigmoid'))
model.add(Dropout(0.10))

#hidden layer 3: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('sigmoid'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 9s 9us/step - loss: 0.6923 - acc: 0.5249
Epoch 2/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 0.6920 - acc: 0.5262
Epoch 3/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.6919 - acc: 0.5274
Epoch 4/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.6918 - acc: 0.5282
Epoch 5/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.6918 - acc: 0.5288

ROC AUC Score for sigmoid-3layers-100nodes is: 0.5060564617261423
```

2. With those same 3 architectures, run the SAME architecture but with 2 different (from sigmoid) activation functions. Google the Keras documentation for a look at different available activations.

Architecture 2 using ReLU

2.1.1 Architecture1: 1 hidden layer with 100 nodes using ReLU

In [59]:
```
arch_name = "relu-1layer-100nodes"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 6s 5us/step - loss: 0.6171 - acc: 0.6559
Epoch 2/5
1100000/1100000 [==============================] - 5s 4us/step - loss: 0.5892 - acc: 0.6878
Epoch 3/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.5799 - acc: 0.6959
Epoch 4/5
1100000/1100000 [==============================] - 6s 5us/step - loss: 0.5745 - acc: 0.7010
Epoch 5/5
1100000/1100000 [==============================] - 5s 5us/step - loss: 0.5708 - acc: 0.7037

ROC AUC Score for relu-1layer-100nodes is: 0.7868069467401482
```

2.1.2 Architecture 2: 2 hidden layers with 100 nodes each using ReLU

In [60]:
```
arch_name = "relu-2layers-100nodes"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.6165 - acc: 0.6521
Epoch 2/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.5760 - acc: 0.6955
Epoch 3/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.5630 - acc: 0.7067
Epoch 4/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.5542 - acc: 0.7131
Epoch 5/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.5469 - acc: 0.7181

ROC AUC Score for relu-2layers-100nodes is: 0.8089000030353879
```

2.1.3 Architecture 3: 3 hidden layers with 100 nodes each using ReLU

In [61]:
```
arch_name = "relu-3layers-100nodes-batch1000"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 3: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 11s 10us/step - loss: 0.6331 - acc: 0.6282
Epoch 2/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.5784 - acc: 0.6937
Epoch 3/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 0.5640 - acc: 0.7048
Epoch 4/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.5526 - acc: 0.7137
Epoch 5/5
1100000/1100000 [==============================] - 11s 10us/step - loss: 0.5449 - acc: 0.7196

ROC AUC Score for relu-3layers-100nodes-batch1000 is: 0.8123238731526745
```

Architecture 3 using Tanh

2.2.1 Architecture1: 1 hidden layer with 100 nodes using Tanh

In [62]:
```
arch_name = "tanh-1layer-100nodes"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('tanh'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.6432 - acc: 0.6279
Epoch 2/5
1100000/1100000 [==============================] - 5s 5us/step - loss: 0.6304 - acc: 0.6466
Epoch 3/5
1100000/1100000 [==============================] - 6s 6us/step - loss: 0.6172 - acc: 0.6594
Epoch 4/5
1100000/1100000 [==============================] - 6s 5us/step - loss: 0.6072 - acc: 0.6695
Epoch 5/5
1100000/1100000 [==============================] - 6s 5us/step - loss: 0.6012 - acc: 0.6747

ROC AUC Score for tanh-1layer-100nodes is: 0.7599418643787034
```

2.2.2 Architecture 2: 2 hidden layers with 100 nodes each using tanh

In [63]:
```
arch_name = "tanh-2layers-50nodes"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('tanh'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('tanh'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.6454 - acc: 0.6238
Epoch 2/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.6301 - acc: 0.6459
Epoch 3/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.6103 - acc: 0.6644
Epoch 4/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.5959 - acc: 0.6789
Epoch 5/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.5898 - acc: 0.6844

ROC AUC Score for tanh-2layers-50nodes is: 0.7703052432297541
```

2.2.3 Architecture 3: 3 hidden layers with 100 nodes each using tanh

In [64]:
```
arch_name = "tanh-3layers-100nodes"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('tanh'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('tanh'))
model.add(Dropout(0.10))

#hidden layer 3: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('tanh'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 10s 9us/step - loss: 0.6480 - acc: 0.6186
Epoch 2/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 0.6300 - acc: 0.6469
Epoch 3/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.6089 - acc: 0.6660
Epoch 4/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.5983 - acc: 0.6757
Epoch 5/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.5920 - acc: 0.6813

ROC AUC Score for tanh-3layers-100nodes is: 0.7688310275395528
```

Compare all tests

In [65]:
```python
testSeries = pd.Series(test_results)
testSeries.sort_values(ascending=False, inplace=True)
testSeries
```

Out[65]:
```
relu-3layers-100nodes-batch1000    0.812324
relu-2layers-100nodes              0.808900
relu-1layer-100nodes               0.786807
tanh-2layers-50nodes               0.770305
tanh-3layers-100nodes              0.768831
tanh-1layer-100nodes               0.759942
sigmoid-1layer-100nodes            0.732065
sigmoid-2layers-100nodes           0.686323
sigmoid-3layers-100nodes           0.506056
dtype: float64
```

3. Take your best model from parts 1&2 and vary the batch size by at least 2 orders of magnitude

Our best model from previous tests was using ReLU activation functions with 3 layers and 100 neurons each. We will now experiment changing the batch sizes of 100, 10000, and 100000 - and compare results

In [66]:
```
test_results_batch = {}

test_results_batch["relu-3layers-100nodes-batch1000"] = testSeries[0]

arch_name = "relu-3layers-100nodes-batch10K"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 3: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=10000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results_batch[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.6914 - acc: 0.5290
Epoch 2/5
1100000/1100000 [==============================] - 7s 6us/step - loss: 0.6911 - acc: 0.5295
Epoch 3/5
1100000/1100000 [==============================] - 6s 5us/step - loss: 0.6821 - acc: 0.5568
Epoch 4/5
1100000/1100000 [==============================] - 6s 6us/step - loss: 0.6362 - acc: 0.6351
Epoch 5/5
1100000/1100000 [==============================] - 6s 5us/step - loss: 0.6159 - acc: 0.6594

ROC AUC Score for relu-3layers-100nodes-batch10K is: 0.7454849185402521
```

In [67]:
```
arch_name = "relu-3layers-100nodes-batch100K"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 3: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=100000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results_batch[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 10s 9us/step - loss: 0.6922 - acc: 0.5258
Epoch 2/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 0.6915 - acc: 0.5295
Epoch 3/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 0.6914 - acc: 0.5295
Epoch 4/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 0.6914 - acc: 0.5295
Epoch 5/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.6913 - acc: 0.5295

ROC AUC Score for relu-3layers-100nodes-batch100K is: 0.5542685851098367
```

In [68]:
```
arch_name = "relu-3layers-100nodes-batch100"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 3: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=100)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results_batch[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 29s 26us/step - loss: 0.6035 - acc: 0.6694
Epoch 2/5
1100000/1100000 [==============================] - 32s 29us/step - loss: 0.5743 - acc: 0.6984
Epoch 3/5
1100000/1100000 [==============================] - 29s 26us/step - loss: 0.5639 - acc: 0.7072
Epoch 4/5
1100000/1100000 [==============================] - 27s 25us/step - loss: 0.5572 - acc: 0.7130
Epoch 5/5
1100000/1100000 [==============================] - 27s 25us/step - loss: 0.5526 - acc: 0.7169

ROC AUC Score for relu-3layers-100nodes-batch100 is: 0.8095547108313648
```

In [69]:
```
testSeries = pd.Series(test_results_batch)
testSeries.sort_values(ascending=False, inplace=True)
testSeries
```

Out[69]:
```
relu-3layers-100nodes-batch1000    0.812324
relu-3layers-100nodes-batch100     0.809555
relu-3layers-100nodes-batch10K     0.745485
relu-3layers-100nodes-batch100K    0.554269
dtype: float64
```

4. Take your best model (score) from parts 1&2 and use 3 different kernel initializers. Use a reasonable batch size.

In this section, we will use our best model (ReLU activation function, with 3 hidden layers of 100 neurons each w/batch size of 1000 inputs), and experiment with kernel initilizers such as 'random_uniform', 'glorot_uniform' and 'he_uniform'

In [70]:
```
test_results_kernals = {}

test_results_kernals["relu-3layers-100nodes-batch1000-uniform"] = testSeries[0]

arch_name = "relu-3layers-100nodes-batch1000-randomuniform"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 3: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results_kernals[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 9s 9us/step - loss: 0.6333 - acc: 0.6266
Epoch 2/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 0.5775 - acc: 0.6944
Epoch 3/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 0.5612 - acc: 0.7078
Epoch 4/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.5507 - acc: 0.7155
Epoch 5/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.5428 - acc: 0.7213

ROC AUC Score for relu-3layers-100nodes-batch1000-randomuniform is: 0.813445948173916
```

In [71]:
```
arch_name = "relu-3layers-100nodes-batch1000-glorotuniform"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='glorot_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='glorot_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 3: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='glorot_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results_kernals[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 10s 9us/step - loss: 0.6161 - acc: 0.6547
Epoch 2/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 0.5771 - acc: 0.6946
Epoch 3/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.5627 - acc: 0.7065
Epoch 4/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.5538 - acc: 0.7132
Epoch 5/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 0.5471 - acc: 0.7186

ROC AUC Score for relu-3layers-100nodes-batch1000-glorotuniform is: 0.8124337469460416
```

In [72]:
```
arch_name = "relu-3layers-100nodes-batch1000-heuniform"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='he_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='he_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 3: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='he_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results_kernals[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 9s 9us/step - loss: 0.6363 - acc: 0.6320
Epoch 2/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.5961 - acc: 0.6776
Epoch 3/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.5764 - acc: 0.6955
Epoch 4/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.5656 - acc: 0.7043
Epoch 5/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.5582 - acc: 0.7105

ROC AUC Score for relu-3layers-100nodes-batch1000-heuniform is: 0.8014568339834331
```

In [73]:
```
testSeries = pd.Series(test_results_kernals)
testSeries.sort_values(ascending=False, inplace=True)
testSeries
```

Out[73]:
```
relu-3layers-100nodes-batch1000-randomuniform    0.813446
relu-3layers-100nodes-batch1000-glorotuniform    0.812434
relu-3layers-100nodes-batch1000-uniform          0.812324
relu-3layers-100nodes-batch1000-heuniform        0.801457
dtype: float64
```

5. Take your best results from #3 and try 3 different optimizers.

Our best model userd SGD optimizer in the previous experiment. In this section, we will use our best model to experiment with 3 different optimizers such as adagrad, adam, rmsprop to compare the model peformances

In [80]:
```
test_results_optimizers = {}

test_results_optimizers["relu-3layers-100nodes-batch1000-randomuniform-sgd"] = testSeries[0]
test_results_optimizers
```

Out[80]:
```
{'relu-3layers-100nodes-batch1000-randomuniform-sgd': 0.813445948173916}
```

In [81]:
```
arch_name = "relu-3layers-100nodes-batch1000-randomuniform-adagrad"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 3: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
# sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)

adagrad = Adagrad(lr=0.01, epsilon=None, decay=1e-6)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=adagrad)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results_optimizers[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 11s 10us/step - loss: 0.6171 - acc: 0.6538
Epoch 2/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.5854 - acc: 0.6870
Epoch 3/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.5758 - acc: 0.6956
Epoch 4/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.5692 - acc: 0.7010
Epoch 5/5
1100000/1100000 [==============================] - 8s 7us/step - loss: 0.5640 - acc: 0.7057

ROC AUC Score for relu-3layers-100nodes-batch1000-randomuniform-adagrad is: 0.7925072607508825
```

In [82]:
```
arch_name = "relu-3layers-100nodes-batch1000-randomuniform-adam"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 3: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
# sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
adam = Adam(lr=0.1, beta_1=0., epsilon=None, decay=1e-6)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=adam)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results_optimizers[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 11s 10us/step - loss: 8.5255 - acc: 0.4705
Epoch 2/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 8.5348 - acc: 0.4705
Epoch 3/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 8.5348 - acc: 0.4705
Epoch 4/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 8.5348 - acc: 0.4705
Epoch 5/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 8.5348 - acc: 0.4705

ROC AUC Score for relu-3layers-100nodes-batch1000-randomuniform-adam is: 0.5
```

In [83]:
```
arch_name = "relu-3layers-100nodes-batch1000-randomuniform-rmsprop"
model = Sequential()

#hidden layer 1: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 2: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 3: 100 nodes
model.add(Dense(100, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
# sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
# adamax = Adamax(lr=0.1, beta_1=0.9, epsilon=None, decay=1e-6)
#adadelta = Adadelta(lr=0.1, rho=0.95, epsilon=None, decay=1e-6)
rmsprop = RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=rmsprop)
model.fit(x, y, epochs=5, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
test_results_optimizers[arch_name] = score
```

```
Epoch 1/5
1100000/1100000 [==============================] - 11s 10us/step - loss: 0.6228 - acc: 0.6474
Epoch 2/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.5847 - acc: 0.6874
Epoch 3/5
1100000/1100000 [==============================] - 8s 8us/step - loss: 0.5695 - acc: 0.7004
Epoch 4/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.5599 - acc: 0.7086
Epoch 5/5
1100000/1100000 [==============================] - 9s 8us/step - loss: 0.5528 - acc: 0.7142

ROC AUC Score for relu-3layers-100nodes-batch1000-randomuniform-rmsprop is: 0.8082289253285946
```

In [84]:

```
testSeries = pd.Series(test_results_optimizers)
testSeries.sort_values(ascending=False, inplace=True)
testSeries
```

Out[84]:

```
relu-3layers-100nodes-batch1000-randomuniform-sgd        0.813446
relu-3layers-100nodes-batch1000-randomuniform-rmsprop    0.808229
relu-3layers-100nodes-batch1000-randomuniform-adagrad    0.792507
relu-3layers-100nodes-batch1000-randomuniform-adam       0.500000
dtype: float64
```

6. Take all that you've learned so far and give your best shot at producing a score.

After experimenting with various parameters, we have achieved best peformance with ReLU activation function for 3 hidden layers with 100 neurons each, with a batch size of 1000, SGD optimizer and random_uniform kernel initializer. We will now experiment changing some of these key parameters and find out if there will be further improvement in the model performance.

In [85]:

```
arch_name = "relu-3layers-100nodes-batch1000-randomuniform-sgd-THE BEST"
model = Sequential()

#hidden layer 1: 200 nodes
model.add(Dense(200, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 2: 200 nodes
model.add(Dense(200, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 3: 200 nodes
model.add(Dense(200, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#hidden layer 4: 200 nodes
model.add(Dense(200, input_dim=x.shape[1], kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.10))

#output layer: 1 node
model.add(Dense(1, kernel_initializer='uniform'))
model.add(Activation('sigmoid'))

#optimizer
sgd = SGD(lr=0.15, decay=1e-8, momentum=0.95, nesterov=True)

#compile
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=sgd)
model.fit(x, y, epochs=20, batch_size=1000)

score = roc_auc_score(y_test,model.predict(x_test))

print("")
print("ROC AUC Score for " + arch_name + " is: " + str(score))
# test_results_optimizers[arch_name] = score
```

```
Epoch 1/20
1100000/1100000 [==============================] - 23s 21us/step - loss: 0.6116 - acc: 0.6545
Epoch 2/20
1100000/1100000 [==============================] - 19s 18us/step - loss: 0.5627 - acc: 0.7058
Epoch 3/20
1100000/1100000 [==============================] - 20s 18us/step - loss: 0.5478 - acc: 0.7178
Epoch 4/20
1100000/1100000 [==============================] - 20s 18us/step - loss: 0.5365 - acc: 0.7261
Epoch 5/20
1100000/1100000 [==============================] - 21s 19us/step - loss: 0.5289 - acc: 0.7317
Epoch 6/20
1100000/1100000 [==============================] - 21s 19us/step - loss: 0.5239 - acc: 0.7352
Epoch 7/20
1100000/1100000 [==============================] - 20s 18us/step - loss: 0.5200 - acc: 0.7375
Epoch 8/20
1100000/1100000 [==============================] - 20s 19us/step - loss: 0.5170 - acc: 0.7397
Epoch 9/20
1100000/1100000 [==============================] - 20s 18us/step - loss: 0.5149 - acc: 0.7412
Epoch 10/20
1100000/1100000 [==============================] - 20s 19us/step - loss: 0.5124 - acc: 0.7429
Epoch 11/20
1100000/1100000 [==============================] - 21s 19us/step - loss: 0.5106 - acc: 0.7443
Epoch 12/20
1100000/1100000 [==============================] - 21s 19us/step - loss: 0.5087 - acc: 0.7452
Epoch 13/20
1100000/1100000 [==============================] - 20s 18us/step - loss: 0.5074 - acc: 0.7461
Epoch 14/20
1100000/1100000 [==============================] - 21s 19us/step - loss: 0.5062 - acc: 0.7468
Epoch 15/20
1100000/1100000 [==============================] - 20s 18us/step - loss: 0.5052 - acc: 0.7478
Epoch 16/20
1100000/1100000 [==============================] - 21s 19us/step - loss: 0.5039 - acc: 0.7489
Epoch 17/20
1100000/1100000 [==============================] - 21s 19us/step - loss: 0.5029 - acc: 0.7491
Epoch 18/20
1100000/1100000 [==============================] - 22s 20us/step - loss: 0.5022 - acc: 0.7495
Epoch 19/20
1100000/1100000 [==============================] - 21s 19us/step - loss: 0.5012 - acc: 0.7503
Epoch 20/20
1100000/1100000 [==============================] - 21s 19us/step - loss: 0.5005 - acc: 0.7507

ROC AUC Score for relu-3layers-100nodes-batch1000-randomuniform-sgd-THE BEST is: 0.84338237923623
```

10 points - Q1: What was the effect of adding more layers/neurons.

During this exercise, we have gamed with our best model by adjusting layers and neurons. We have tested the model with 100 and 200 neurons in each of the layers, and also tested with 3, 4 and 5 hidden layers. We found that adding additional neurons to the layers increased the ROC AUC Score by about 2% (from ~0.8127 to ~0.8267), and additional 1% increase by adding 4th layer (from ~0.8267 to ~0.83) - with number of epochs at 10. However, the score slighly decreased by adding 5th layer. We reran our final model with 20 epochs and achieved the score of 0.8433. We conclude 4 layers with 200 neurons each yielded best results with all other parameters unchanged.

10 points - Q2: Which parameters gave you the best result and why (in your opinion) did they work.

We have achieved the best results with following key parameters:

Activation function for Hidden Layers: ReLU. The ReLu function is defined as A(x) = max(0,x). It gives an output x if x is positive and 0 otherwise. ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. That is a good point to consider when we are designing deep neural nets [2].

Kernel Initializer: random_uniform. Initializer that generates tensors with a uniform distribution.

Optimizer: SGD (with learning rate=0.15; decay=1e-8, momentum=0.95) (we have increased learning rate from 0.1 to 0.15, decreased decay from 1e-6 to 1e-8 and increased momentum from 0.9 to 0.95 compared to the base model)

Number of Epochs: We started the base model with 5 epochs, and gradually increased to 10 and 20. We have observed that the score significantly increased as we increase the epochs.

Batch size: We tested the batch sizes of 100, 1000, 10000 and 100000. We have observed the best scores with the batch size at 1000.

In our opinion, Rectified Linear Units (ReLU) activation function is appropriate for the dataset as it non-linear and is computationally less expensive from other functions considering binary classification of the problem; Random_Uniform aids at removing bias while uniformly distributing initial weights; and Stochastic Gradient Descent (SGD) does batch processing that performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time.

20 points - Q3: For #6, how did you decide that your model was ‘done’

We have started our base model with 'sigmoid' activation function with 'uniform' Kernel Initializer, 'SGD' Optimizer with 1 hidden layer, with 5 epochs and a batch size of 1000. We have used ROC AUC score as our metric to measure the performance of the model. When we gamed changing these parameters, we have observed the ROC AUC score ranging from 0.5 to 0.7 with our base model. We continued to tune the paramerts as decribed in the previous sections, we have certainly improved the model performance significantly with our BEST model (activation funtion: ReLU, Optimizer: SGG, Kernal Initializer: random_uniform, epochs:20, batch size:1000) with ROC AUC score of 0.8428. The improvement of the scores can still be possible with further tuning of number of epochs, batch size and other parameters we did not test. We conclude the exercise given the time and resource constraints, and achieved satisfactory results, and we call it done.

References

[1] Higgs Boson Experiment (Wikipedia) https://en.wikipedia.org/wiki/Higgs_boson (https://en.wikipedia.org/wiki/Higgs_boson)

[2] https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 (https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0)

[3] Prof Slater's sample code and presentations

Thank you