## **Introduction**

With the growing size of data in recent years, data scientists have taken advantage of the power and abilities of neural networks. Neural networks are systems comprised of a large number of layered, interconnected processing nodes. The data then passes through the layers of the network to yield the output layer. With each layer, the weights and thresholds of the network are adjusted to produce the most accurate result as possible. This is why neural networks perform so well on very large data sets [1].

Although the common use of neural networks is a relatively new trend in the data science community, they have been around since 1944. The researchers, Warren McCullough and Walter Pitts researched neural networks, which were roughly inspired by the functioning details of the human brain. The use of neural networks in mathematics has gone in and out of style since then, becoming extremely popular in recent years due to the development of computers’ processing power [1].

There are several types of neural networks that are commonly used to complete machine learning tasks today including Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), and Modular Neural Networks (MNN) to name a few. Each type of neural network requires slightly different mathematical operations and parameters to produce the output layer [2].

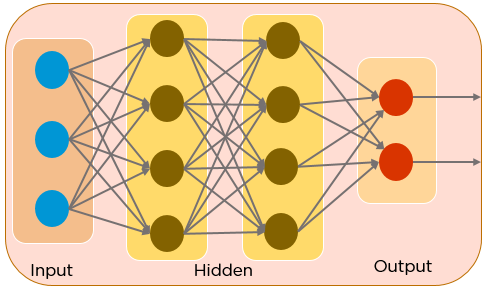
This analysis will focus on describing the effects that changing portions of the neural network has on the output. The HIGGS data used in this analysis ( <https://archive.ics.uci.edu/ml/machine-learning-databases/00280/?C=D;O=D>) is a classification problem to distinguish between a signal process which produces Higgs bosons and a background process which does not. The goal of this study is to determine the effect of adding more layers and to determine the parameters necessary to achieve the best result.

## **Background**

Artificial neural networks are important tools used in deep learning. The structure of an artificial neural network is modeled loosely on the design of the human brain. The human brain is comprised of neurons that connect with other neurons through signals that transmit information. Based on the input to the neuron in the brain, an output is produced through an axon connected to another neuron [3].

Similar to the structure of neural networks in a human brain, an artificial neural network works by sending the data through layers (neurons) to produce an output. The data begins as input layers, which performs best when the data is extremely large. Then the data is passed through a number of hidden layers that transform the data using set parameters and operations to produce the output layer, or the desired result [3]. This process is displayed in figure 1. There can be several hidden layers within the neural network.

**Figure 1. Visual representation of an artificial neural network**

****

The network is refined using set parameters and hyperparameters that can affect the result of the model. These include weights, biases, epochs, the learning rate, the batch size, the number of layers, the activation functions, the kernel initializers, and the optimizers [3]. Transfer functions are calculated based on the weights of the nodes, which are then summed using the weights and a bias is added. This result is used as the input to the activation functions that determine which nodes to fire. The nodes that are fired are used to produce the output layer.

Calculating the cost function along with the use of the back-propagation algorithm can help improve the accuracy of neural networks. The cost function is a calculation of the error, which can be used to improve the accuracy in subsequent models if the neural network learns from the prior mistakes. The back-propagation algorithm is used to randomly adjust the weights of the nodes to minimize the error. This process is repeated until the error is minimized [3].

## **Method**

**Data Description**

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; The last 500,000 examples of the total 10,500,000 rows are used as a test set.

For 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.

**Implementation Environment**

Data processing was done using Python 3.6. Keras with TensorFlow backend was used to build up and train a deep neural network. Intel 8th GEN CORE-i7- 8750H processor, Graphic card NVIDIA GPU (GeForce GTX 1070) with cuDNN 7.3.1 for CUDA 10.0) and 16 GB DDR4-2666MHz of RAM were used to perform the computations.

**Fine-tuning Model Development**

There is an interest in using deep learning methods to find important features in order to meet the need for physicists. There are huge number of parameters for deep learning model construction. In this case study, we will optimize the CNN architectures by choosing different convolutional layers, adjust activation function, select different batch size, compare different kernels with reasonable batch size, and decide the best optimizer. Finally, we will evaluate the optimized CNN architectures for model performance. For baseline model, we will use 3 convolutional layers with sigmoid activation.

**1. Use different numbers of layers and neurons in architecture**

Three architectures were chosen to compare the effects that different numbers of layers on the accuracy. These three architectures included networks with 2, 3, and 5 layers by using Sigmoid activation function. **Table 1** displays the accuracy score along with the Area Under the Receiver Operating Characteristic Curve (ROC AUC) score from the prediction scores. **Table 1** reveals that the model with 3 layers performs the best in terms of accuracy and ROC AUC.

**Table 1. Accuracy and ROC AUC scores for the three models**

|  |  |  |
| --- | --- | --- |
| Number of Layers | Accuracy Score | ROC AUC Score |
| **2** | 0.6955 | 0.7832 |
| **3** | 0.6960 | 0.7844 |
| **5** | 0.5300 | 0.5203 |

In **Figure 2**, the true and false positive rates are plotted in the ROC curves, as shown with these 3 models with 2, 3, and 5 layers. The areas under ROC curves are used to show the tradeoff between sensitivity and specificity. Consistent with model accuracy scores, the model with 3 layers are the best model in performance.

|  |
| --- |
| **Figure 2. ROC curves for different numbers of layers** |
|  |

In **Figure 3**, we plot the model accuracy (**Figure 3A**) and model loss (**Figure 3B**) per epoch for the model with 2 layers in neural network architecture. Model accuracy and loss are two different metrics to evaluate the model performance. The model accuracy from applied perspective is usually determined after the model parameters are learned and fixed and no learning is taking place. Loss is often used to optimize in the by updating weights in neural network models. In **Figure** 2A, we find there the model accuracy is less in train, but higher in validation, indicating the model needs to be optimized to let both values converge. Obviously, there are no overfitting issues in this model. In **Figure 3B**, we find the model loss in higher in train data than in validation data. As expected, there is the reduction of loss after each epoch, suggesting model optimization is needed.

|  |
| --- |
| **Figure 3. Model accuracy and loss for 2 layers in neural network architecture** |
| |  |  | | --- | --- | | **A** | **B** | |

**2. Use different activation functions (“ReLU” and “eLU”) in the same architecture**

A nonlinear activation function is typically applied after each convolutional layer. The rectified linear unit (ReLU) tends to be the default activation for deep networks due to its simplicity and its ability to enable fast training in recent years among various nonlinear functions. ReLU has many variations, such as leaky exponential LU (eLU).

Running these same architectures with different activation functions returns different results. **Table 2** displays the overall ROC AUC score for these three models using a Relu and Elu activation functions. The ROC AUC scores decreased for all of the models for both new activations functions. Overall, using the Elu activation function produced slightly higher ROC AUC scores than the Relu function, but both of these were a decrease from the score produced using the Sigmoid function.

**Table 2. ROC AUC scores for the three models using different activation functions**

|  |  |  |
| --- | --- | --- |
| Number of Layers | Activation Function | ROC AUC Score |
| **2** | Relu | 0.4745 |
| **2** | Elu | 0.4846 |
| **3** | Relu | 0.4656 |
| **3** | Elu | 0.4709 |
| **5** | Relu | 0.4642 |
| **5** | Elu | 0.5 |

**3. Vary the batch size by at least 2 orders of magnitude in the CNN architecture**

Batch size is the total number of training examples present in a single batch. Varying the batch size (10, 1000, 100,000 in separation) can also alter the accuracy of the neural network.

**Table 3. ROC AUC scores for the 3-layer network with various batch sizes**

|  |  |
| --- | --- |
| Batch Size | ROC AUC Score |
| **10** | 0.8252 |
| **1000** | 0.7789 |
| **100,000** | 0.5006 |

**Table 3** presents the model scores of the top-performing model from above using various batch sizes. This was the network created with three layers using a Sigmoid activation function. Overall, as the batch size increased in this model, the ROC AUC score decreased. This is evidence that a smaller batch size produces better results using this data.

**4. Try different kernel initializers with batch number 10**

This model was then tested with different kernel initializers, including ‘uniform,’ ‘random uniform,’ and ‘normal.’ The results of varying this parameter can be seen in **Table 4**. The kernel initializer did not have a huge effect on the ROC AUC score for this model, although the model created using a ‘normal’ kernel initializer had a slightly higher score than the others.

**Table 4. ROC AUC scores for the 3-layer network with various kernel initializers**

|  |  |
| --- | --- |
| Kernel Initializer | ROC AUC Score |
| **Uniform** | 0.7808 |
| **Random Uniform** | 0.7809 |
| **Normal** | 0.7820 |

**5. Use different optimizers (“SGD”, “Adamax”, “Nadam”)**

The effects of changing the optimizer on this same model are shown in **Table 5**. The Adamax optimizer produced the highest ROC AUC score with a value of 0.8228.

**Table 5. ROC AUC scores for the 3-layer network with various optimizers**

|  |  |
| --- | --- |
| Optimizer | ROC AUC Score |
| **SGD** | 0.7811 |
| **Adamax** | 0.8228 |
| **Nadam** | 0.7802 |

**6. Model with the best performance**

Using the results from the various parameter adjustments, a few neural network models were then created to produce the most accurate results. Using parameter combination (3 layers, a ‘random uniform’ kernel initializer, a Sigmoid activation function, the Adamax optimizer), the models were created with a batch size of 10, 500, 1000, 10000. The batch size at 1000 has the best score 0.8235 over other batch sizes, and the scores are summarized in **Table 6**.

**Table 6. ROC AUC scores for the 3-layer network using a ‘random uniform’ kernel initializer, a Sigmoid activation function, and the Adamax optimizer with various batch sizes**

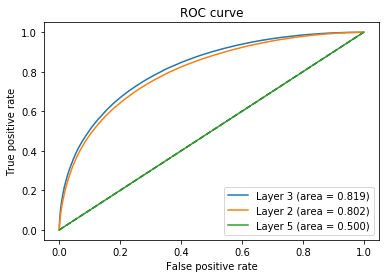
|  |  |
| --- | --- |
| Batch Size | ROC AUC Score |
| **10** | 0.7875 |
| **500** | 0.8175 |
| **1000** | 0.8235 |
| **10,000** | 0.8214 |

## **Results**

1. **Adding more convolutional layers does not improve performance**

In **Figure 4**, the true and false positive rates in the models with optimized parameters except the number of layers are plotted in the ROC curves, as shown with these 3 models with 2, 3, and 5 layers. The areas under ROC curves are used to show the tradeoff between sensitivity and specificity. Consistent with model accuracy scores, the model with 3 layers are the best model in performance with ROC AUC score 0.819. This score is close to 2-layer model, but far better than 5 -layer model.

|  |
| --- |
| **Figure 4. ROC curves for different numbers of layers** |



In addition, as shown in **Table 7**, the summarized results of increasing the number of layers for different activation functions are compared to address if more convolutional layers would be helpful in boosting model performance.

**Table 7. ROC AUC scores by activation function and number of layers**

|  |  |  |
| --- | --- | --- |
| Number of Layers | Activation Function |  |
| **2** | Sigmoid | 0.7792 |
| **3** | Sigmoid | 0.7808 |
| **5** | Sigmoid | 0.4443 |
| **2** | Elu | 0.4846 |
| **3** | Elu | 0.4709 |
| **5** | Elu | 0.5 |
| **2** | Relu | 0.4745 |
| **3** | Relu | 0.4656 |
| **5** | Relu | 0.4642 |

The color of the ROC AUC score identifies if the score increased (green) or decreased (red) from the previous model, one with fewer layers. For the Sigmoid models, the ROC AUC score increases and then decreases rather substantially. This gives evidence to overfitting, by increasing the number of layers the model becomes too good at predicting that training data so when it is asked to predict a different dataset it does not perform as well. The Elu model behaved in the opposite manner as the sigmoid with at first seeing a decrease and then an increase in ROC AUC score. Lastly, the Relu models decreased each time the number of layers was increased.

Overall increasing the number of layers had three different impacts using three different activation functions. However, if one looks at each time the number of layers was increased it appears that increasing the number of layers decreasing performance. The number of layers was increased six times and four of those times the performance decreased. It should be noted that most of these decreases in performance were very small, but performance did still do down.

1. **Interpreting why some parameters are important for best model performance**

The best parameter combination was a 3-layer network using a random uniform kernel initializer, Sigmoid activation function, Adamax optimizer and a batch size of 1,000 which had a ROC AUC score of 0.8235. Three layers isn’t enough layers to overfit the model but provides a greater level of detail than just two layers. This same argument can be made for the batch size of 1,000. The batches were large enough to get a diversify selection but not large enough to overfit. The dataset did not have many variables, only 28, which could explain why the Sigmoid activation function was determined to be the best. This is one of the simplest activation functions so some of the other functions might have provided a level of complexity that was not needed which caused them to not perform as well.

1. **Evaluate the optimized morels**

Multiple different approaches were tested in order to arrive at the optimum model. A value of 0.82 (**Figure 4**) for a ROC AUC score is very high due to the fact that the max value is 1. A trial and error process were carried out to tune this model, but no model outperformed the model mentioned about. While it may be possible that there is a model that would beat our ideal model the effort to find it is not worth the gain associated. The ideal parameters that cause large changes in ROC AUC, mainly number of layers, have been determined. Other parameters could be tuned but the gain from these will only be minimal, if at all. Due to this fact and the few models attempted that did not beat our model it is reasonable to assume this is the best model possible.

Compared with the baseline model accuracy and model loss as shown in **Figure 3**, we also plot the model accuracy (**Figure 5A**) and model loss (**Figure 5B**) per epoch for the model with 3 layers in the optimized neural network architecture. In **Figure** 5A, we find the model accuracies have been improved up to 0.74 (**Figure 5A**) from 0.71 (**Figure 3A**) and the loss values have been dropped to 0.52 (**Figure 5B**) from 0.56 (**Figure 3B**), both for training and validation data. This closeness in model accuracy and model loss suggest model optimization is completed.

|  |
| --- |
| **Figure 5. Model accuracy and loss for 3 layers in neural network architecture** |
| |  |  | | --- | --- | | **A** | **B** | |

## **Conclusions**

The parameters selected can be the difference between an excellent model and a poor model. One of the main tasks as a data scientist is too fine tune a model, change the parameters, until an optimum result is reached. In this case study it was shown how large of an effect changing just one parameter can have on the model. Another important note is that bigger is not always better, which was soon when choosing the number of layers in the model. If the model uses too many layers there arises a potential for overfitting. This would cause the model to perform poorly when given a test dataset as seen in the Sigmoid activation function example.

Lastly, it should be noted that parameters interact with each other differently, so it is imperative to try many combinations until the optimum model is reached. When different activation functions were used the resulting ideal number of layers was different in each case. For sigmoid the best performing model had three layers, Elu had five and Relu had two. This shows that what works for one model is not the best choice for every model. This is also true for datasets. While we found evidence that generally increasing the number of layers decreased the model’s performance that finding is not applicable to all datasets. This is true for all parameters and even types of models. Due to this fact when starting the modeling process, one should always try a vast array of model types and model parameters to determine the ideal model.

## **Future Work**

One area that could be expanded upon is doing a similar analysis on multiple datasets. While the findings of this analysis are interesting and helpful, they cannot necessarily be applied to the next dataset that is encountered. It would be something to note when starting the model process on a new dataset that in the past these were our findings but those cannot be applied with certainty. While they can never be truly applied with absolute certainty the level of certainty can be increased. If we saw the same results when doing this analysis on ten datasets or twenty or one hundred the level of certainty increases dramatically. This would be an interesting study to see if the findings in this case study are out of the norm or in line with other results.

It would also be interesting to make a “cheat sheet” using these results. This would involve documenting different characteristics about the dataset being analyzed and which parameters worked best most of the time. Then in the future one could reference this document and find the dataset characteristics that most closely match the dataset they are modeling. This would allow one to have a starting point as opposed to starting from scratch.

## **Ethical considerations**

The power and performance abilities of neural networks on large data is extremely promising and exciting to data scientists. However, with a tool this powerful, some considerations should be taken into account. First, neural networks and deep learning has the potential to learn things that the creators do not necessarily want the model to learn. An example of this would be the model knowingly or unknowingly producing a result that discriminates against certain sexes, races, or abilities. This could happen due to an inherent bias in the dataset. In a recent example, certain algorithms used to select which ads to display to users would only display ads to some groups of people but not to other groups. Educational ads that are excluded from being displayed to certain groups of people can lead to unintentional discrimination or even segregation if the admission reviewers are not careful [4]. There are several examples of racist bots that learned from comments left by users online. A bias in the dataset can have extreme effects, whether intentional or not. It is for this reason that data scientists must pay extra attention to the data that is used to train these neural networks or deep learning models to ensure that the data is unbiased enough to produce a reliable and unbiased output. Additionally, data scientists should not display results or outcomes as useful or even use the output if the results are sexist, racist, or discriminate against any group of people.

**References**

1. Hardesty, L., & MIT News Office. (2017, April 14). Explained: Neural networks. Retrieved March 25, 2019, from http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414
2. Maladkar, K. (2018, November 20). 6 Types of Artificial Neural Networks Currently Being Used in ML. Retrieved March 25, 2019, from https://www.analyticsindiamag.com/6-types-of-artificial-neural-networks-currently-being-used-in-todays-technology/
3. How do artificial neural networks work? (n.d.). Retrieved March 25, 2019, from https://www.quora.com/How-do-artificial-neural-networks-work
4. Ethics for neural networks. (n.d.). Retrieved March 25, 2019, from <https://aleph.se/andart2/ethics/ethics-for-neural-networks/>

## **Appendix - Python code**

## Python codes for Case study 6 (Unit 12)

# data source: https://archive.ics.uci.edu/ml/machine-learning-databases/00280/

import pandas as pd

N=10500000. ## 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=500000,header=None,skiprows=10500000)

from keras.models import Sequential

from keras.layers.core import Dense, Dropout, Activation

from keras.optimizers import SGD

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc

import numpy as np

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

## 2xlayers, roc\_auc\_score = 0.7791760135160337

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q1\_2 = Sequential()

model\_q1\_2.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q1\_2.add(Activation('sigmoid'))

model\_q1\_2.add(Dropout(0.10))

model\_q1\_2.add(Dense(1, kernel\_initializer='uniform'))

model\_q1\_2.add(Activation('sigmoid'))

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

model\_q1\_2.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q1\_2.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q1\_2.predict(x\_test))

## 2xlayers, show model accuracy and model loss

epochs = 5

batch\_size = 1000

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q1\_2 = Sequential()

model\_q1\_2.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q1\_2.add(Activation('sigmoid'))

model\_q1\_2.add(Dropout(0.10))

model\_q1\_2.add(Dense(1, kernel\_initializer='uniform'))

model\_q1\_2.add(Activation('sigmoid'))

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

model\_q1\_2.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

print(model\_q1\_2.summary())

h = model\_q1\_2.fit(

x,

y,

batch\_size=batch\_size,

epochs=epochs,

validation\_data=(x\_test, y\_test),

shuffle=True

)

print('Saving model1 to the file "model1.h5"')

model1.save("model1.h5")

model\_q1\_2.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q1\_2.predict(x\_test))

show\_scores(model\_q1\_2, h, x, y, x\_test, y\_test)

## 3xlayers, roc\_auc\_score = 0.7807769854026723

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q1\_3 = Sequential()

model\_q1\_3.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q1\_3.add(Activation('sigmoid'))

model\_q1\_3.add(Dropout(0.10))

model\_q1\_3.add(Dense(50, kernel\_initializer='uniform'))

model\_q1\_3.add(Activation('sigmoid'))

model\_q1\_3.add(Dropout(0.10))

model\_q1\_3.add(Dense(1, kernel\_initializer='uniform'))

model\_q1\_3.add(Activation('sigmoid'))

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

model\_q1\_3.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q1\_3.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q1\_3.predict(x\_test))

## 5xlayers, roc\_auc\_score = 0.4442846595315575

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q1\_5 = Sequential()

model\_q1\_5.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q1\_5.add(Activation('sigmoid'))

model\_q1\_5.add(Dropout(0.10))

model\_q1\_5.add(Dense(50, kernel\_initializer='uniform'))

model\_q1\_5.add(Activation('sigmoid'))

model\_q1\_5.add(Dropout(0.10))

model\_q1\_5.add(Dense(50, kernel\_initializer='uniform'))

model\_q1\_5.add(Activation('sigmoid'))

model\_q1\_5.add(Dropout(0.10))

model\_q1\_5.add(Dense(50, kernel\_initializer='uniform'))

model\_q1\_5.add(Activation('sigmoid'))

model\_q1\_5.add(Dropout(0.10))

model\_q1\_5.add(Dense(1, kernel\_initializer='uniform'))

model\_q1\_5.add(Activation('sigmoid'))

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

model\_q1\_5.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q1\_5.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q1\_5.predict(x\_test))

## ROC curves for Figure 2

## from sklearn.metrics import roc\_curve, auc

y\_pred\_q1\_l2 = model\_q1\_2.predict(x\_test).ravel()

fpr\_keras\_q1\_l2, tpr\_keras\_q1\_l2, thresholds\_keras\_q1\_l2 = roc\_curve(y\_test, y\_pred\_q1\_l2)

auc\_keras\_q1\_l2 = auc(fpr\_keras\_q1\_l2, tpr\_keras\_q1\_l2)

y\_pred\_q1\_l3 = model\_q1\_3.predict(x\_test).ravel()

fpr\_keras\_q1\_l3, tpr\_keras\_q1\_l3, thresholds\_keras\_q1\_l3 = roc\_curve(y\_test, y\_pred\_q1\_l3)

auc\_keras\_q1\_l3 = auc(fpr\_keras\_q1\_l3, tpr\_keras\_q1\_l3)

y\_pred\_q1\_l5 = model\_q1\_5.predict(x\_test).ravel()

fpr\_keras\_q1\_l5, tpr\_keras\_q1\_l5, thresholds\_keras\_q1\_l5 = roc\_curve(y\_test, y\_pred\_q1\_l5)

auc\_keras\_q1\_l5 = auc(fpr\_keras\_q1\_l5, tpr\_keras\_q1\_l5)

plt.figure(1)

plt.plot([0, 1], [0, 1], 'k--')

plt.plot(fpr\_keras\_q1\_l2, tpr\_keras\_q1\_l2, label='Q1\_L2 (area = {:.3f})'.format(auc\_keras\_q1\_l2))

plt.plot(fpr\_keras\_q1\_l3, tpr\_keras\_q1\_l3, label='Q1\_L3 (area = {:.3f})'.format(auc\_keras\_q1\_l3))

plt.plot(fpr\_keras\_q1\_l5, tpr\_keras\_q1\_l5, label='Q1\_L5 (area = {:.3f})'.format(auc\_keras\_q1\_l5))

plt.xlabel('False positive rate')

plt.ylabel('True positive rate')

plt.title('ROC curve for different layers')

plt.legend(loc='best')

plt.show()

## 2. With those same 3 architectures, run the SAME architecture but with 2 different (from sigmoid) activation function

## http://cs231n.github.io/neural-networks-1/

## 2xlayers, activation ='relu', roc\_auc\_score = 0.47453227907167295

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q2\_2 = Sequential()

model\_q2\_2.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q2\_2.add(Activation('relu'))

model\_q2\_2.add(Dropout(0.10))

model\_q2\_2.add(Dense(1, kernel\_initializer='uniform'))

model\_q2\_2.add(Activation('relu'))

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

model\_q2\_2.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q2\_2.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q2\_2.predict(x\_test))

## 3xlayers, activation ='relu', roc\_auc\_score = 0.46561285961535454

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q2\_3 = Sequential()

model\_q2\_3.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q2\_3.add(Activation('relu'))

model\_q2\_3.add(Dropout(0.10))

model\_q2\_3.add(Dense(50, kernel\_initializer='uniform'))

model\_q2\_3.add(Activation('relu'))

model\_q2\_3.add(Dropout(0.10))

model\_q2\_3.add(Dense(1, kernel\_initializer='uniform'))

model\_q2\_3.add(Activation('relu'))

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

model\_q2\_3.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q2\_3.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q2\_3.predict(x\_test))

## 5xlayers, activation ='relu', roc\_auc\_score = 0.4642447063750198

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q2\_5 = Sequential()

model\_q2\_5.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q2\_5.add(Activation('relu'))

model\_q2\_5.add(Dropout(0.10))

model\_q2\_5.add(Dense(50, kernel\_initializer='uniform'))

model\_q2\_5.add(Activation('relu'))

model\_q2\_5.add(Dropout(0.10))

model\_q2\_5.add(Dense(50, kernel\_initializer='uniform'))

model\_q2\_5.add(Activation('relu'))

model\_q2\_5.add(Dropout(0.10))

model\_q2\_5.add(Dense(50, kernel\_initializer='uniform'))

model\_q2\_5.add(Activation('relu'))

model\_q2\_5.add(Dropout(0.10))

model\_q2\_5.add(Dense(1, kernel\_initializer='uniform'))

model\_q2\_5.add(Activation('relu'))

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

model\_q2\_5.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q2\_5.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test, model\_q2\_5.predict(x\_test))

## 2xlayers, activation ='elu', roc\_auc\_score = 0.484558208847519

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q2\_2e = Sequential()

model\_q2\_2e.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q2\_2e.add(Activation('elu'))

model\_q2\_2e.add(Dropout(0.10))

model\_q2\_2e.add(Dense(1, kernel\_initializer='uniform'))

model\_q2\_2e.add(Activation('elu'))

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

model\_q2\_2e.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q2\_2e.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q2\_2e.predict(x\_test))

## 3xlayers, activation ='elu', roc\_auc\_score = 0.4709237031307237

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q2\_3e = Sequential()

model\_q2\_3e.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q2\_3e.add(Activation('elu'))

model\_q2\_3e.add(Dropout(0.10))

model\_q2\_3e.add(Dense(50, kernel\_initializer='uniform'))

model\_q2\_3e.add(Activation('elu'))

model\_q2\_3e.add(Dropout(0.10))

model\_q2\_3e.add(Dense(1, kernel\_initializer='uniform'))

model\_q2\_3e.add(Activation('elu'))

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

model\_q2\_3e.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q2\_3e.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q2\_3e.predict(x\_test))

## 5xlayers, activation ='elu', roc\_auc\_score = 0.5

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q2\_5e = Sequential()

model\_q2\_5e.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q2\_5e.add(Activation('elu'))

model\_q2\_5e.add(Dropout(0.10))

model\_q2\_5e.add(Dense(50, kernel\_initializer='uniform'))

model\_q2\_5e.add(Activation('elu'))

model\_q2\_5e.add(Dropout(0.10))

model\_q2\_5e.add(Dense(50, kernel\_initializer='uniform'))

model\_q2\_5e.add(Activation('elu'))

model\_q2\_5e.add(Dropout(0.10))

model\_q2\_5e.add(Dense(50, kernel\_initializer='uniform'))

model\_q2\_5e.add(Activation('elu'))

model\_q2\_5e.add(Dropout(0.10))

model\_q2\_5e.add(Dense(1, kernel\_initializer='uniform'))

model\_q2\_5e.add(Activation('elu'))

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

model\_q2\_5e.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q2\_5e.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test, model\_q2\_5e.predict(x\_test))

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

## 3xlayers, batch\_size=1000, roc\_auc\_score = 0.7788914810564554

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q3\_1k = Sequential()

model\_q3\_1k.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q3\_1k.add(Activation('sigmoid'))

model\_q3\_1k.add(Dropout(0.10))

model\_q3\_1k.add(Dense(50, kernel\_initializer='uniform'))

model\_q3\_1k.add(Activation('sigmoid'))

model\_q3\_1k.add(Dropout(0.10))

model\_q3\_1k.add(Dense(1, kernel\_initializer='uniform'))

model\_q3\_1k.add(Activation('sigmoid'))

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

model\_q3\_1k.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q3\_1k.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q3\_1k.predict(x\_test))

## 3xlayers, batch\_size=100000, roc\_auc\_score: 0.5006365070937373

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q3\_100k = Sequential()

model\_q3\_100k.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q3\_100k.add(Activation('sigmoid'))

model\_q3\_100k.add(Dropout(0.10))

model\_q3\_100k.add(Dense(50, kernel\_initializer='uniform'))

model\_q3\_100k.add(Activation('sigmoid'))

model\_q3\_100k.add(Dropout(0.10))

model\_q3\_100k.add(Dense(1, kernel\_initializer='uniform'))

model\_q3\_100k.add(Activation('sigmoid'))

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

model\_q3\_1k.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q3\_1k.fit(x, y, epochs=5, batch\_size=100000)

roc\_auc\_score(y\_test,model\_q3\_100k.predict(x\_test))

## 3xlayers, batch\_size=10, roc\_auc\_score: 0.7807750274417308 (the best)

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q3\_10 = Sequential()

model\_q3\_10.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q3\_10.add(Activation('sigmoid'))

model\_q3\_10.add(Dropout(0.10))

model\_q3\_10.add(Dense(50, kernel\_initializer='uniform'))

model\_q3\_10.add(Activation('sigmoid'))

model\_q3\_10.add(Dropout(0.10))

model\_q3\_10.add(Dense(1, kernel\_initializer='uniform'))

model\_q3\_10.add(Activation('sigmoid'))

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

model\_q3\_10.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q3\_10.fit(x, y, epochs=5, batch\_size=10)

roc\_auc\_score(y\_test,model\_q3\_10.predict(x\_test))

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

## 4 kernel\_initializer='uniform', roc\_auc\_score: 0.7807750274417308

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q4\_u = Sequential()

model\_q4\_u.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q4\_u.add(Activation('sigmoid'))

model\_q4\_u.add(Dropout(0.10))

model\_q4\_u.add(Dense(50, kernel\_initializer='uniform'))

model\_q4\_u.add(Activation('sigmoid'))

model\_q4\_u.add(Dropout(0.10))

model\_q4\_u.add(Dense(1, kernel\_initializer='uniform'))

model\_q4\_u.add(Activation('sigmoid'))

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

model\_q4\_u.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q4\_u.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q4\_u.predict(x\_test))

## 4 kernel\_initializer='random\_uniform', roc\_auc\_score: 0.7808735534914519

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q4\_ru = Sequential()

model\_q4\_ru.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='random\_uniform')) # X\_train.shape[1] == 28 here

model\_q4\_ru.add(Activation('sigmoid'))

model\_q4\_ru.add(Dropout(0.10))

model\_q4\_ru.add(Dense(50, kernel\_initializer='random\_uniform'))

model\_q4\_ru.add(Activation('sigmoid'))

model\_q4\_ru.add(Dropout(0.10))

model\_q4\_ru.add(Dense(1, kernel\_initializer='random\_uniform'))

model\_q4\_ru.add(Activation('sigmoid'))

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

model\_q4\_ru.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q4\_ru.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q4\_ru.predict(x\_test))

## 4 kernel\_initializer='normal' roc\_auc\_score: 0.7820724352084556

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q4\_n = Sequential()

model\_q4\_n.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='normal')) # X\_train.shape[1] == 28 here

model\_q4\_n.add(Activation('sigmoid'))

model\_q4\_n.add(Dropout(0.10))

model\_q4\_n.add(Dense(50, kernel\_initializer='normal'))

model\_q4\_n.add(Activation('sigmoid'))

model\_q4\_n.add(Dropout(0.10))

model\_q4\_n.add(Dense(1, kernel\_initializer='normal'))

model\_q4\_n.add(Activation('sigmoid'))

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

model\_q4\_n.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q4\_n.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q4\_n.predict(x\_test))

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

## 5 optimizer=sgd, roc\_auc\_score: 0.7811275278944515

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q5\_sgd = Sequential()

model\_q5\_sgd.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q5\_sgd.add(Activation('sigmoid'))

model\_q5\_sgd.add(Dropout(0.10))

model\_q5\_sgd.add(Dense(50, kernel\_initializer='uniform'))

model\_q5\_sgd.add(Activation('sigmoid'))

model\_q5\_sgd.add(Dropout(0.10))

model\_q5\_sgd.add(Dense(1, kernel\_initializer='uniform'))

model\_q5\_sgd.add(Activation('sigmoid'))

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

model\_q5\_sgd.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=sgd)

model\_q5\_sgd.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q5\_sgd.predict(x\_test))

## 5 optimizer=Adamax, roc\_auc\_score: 0.8227544567440018

from keras.optimizers import Adamax

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q5\_adam = Sequential()

model\_q5\_adam.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q5\_adam.add(Activation('sigmoid'))

model\_q5\_adam.add(Dropout(0.10))

model\_q5\_adam.add(Dense(50, kernel\_initializer='uniform'))

model\_q5\_adam.add(Activation('sigmoid'))

model\_q5\_adam.add(Dropout(0.10))

model\_q5\_adam.add(Dense(1, kernel\_initializer='uniform'))

model\_q5\_adam.add(Activation('sigmoid'))

adam = Adamax(lr=0.1, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-6)

model\_q5\_adam.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=adam)

model\_q5\_adam.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q5\_adam.predict(x\_test))

## 5 optimizer=Nadam, roc\_auc\_score: 0.7801730074105127

from keras.optimizers import Nadam

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q5\_nag = Sequential()

model\_q5\_nag.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q5\_nag.add(Activation('sigmoid'))

model\_q5\_nag.add(Dropout(0.10))

model\_q5\_nag.add(Dense(50, kernel\_initializer='uniform'))

model\_q5\_nag.add(Activation('sigmoid'))

model\_q5\_nag.add(Dropout(0.10))

model\_q5\_nag.add(Dense(1, kernel\_initializer='uniform'))

model\_q5\_nag.add(Activation('sigmoid'))

nadam = Nadam(lr=0.1, beta\_1=0.9, beta\_2=0.999, epsilon=None, schedule\_decay=1e-6)

model\_q5\_nag.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=nadam)

model\_q5\_nag.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q5\_nag.predict(x\_test))

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

## 3xlayers, kernel\_initializer='random\_uniform, activation= 'sigmoid', optimizer= Adamax, batch\_size=10

## roc\_auc\_score: 0.7875221799347585

# from keras.optimizers import Adamax

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q5\_adam = Sequential()

model\_q5\_adam.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='random\_uniform')) # X\_train.shape[1] == 28 here

model\_q5\_adam.add(Activation('sigmoid'))

model\_q5\_adam.add(Dropout(0.10))

model\_q5\_adam.add(Dense(50, kernel\_initializer='random\_uniform'))

model\_q5\_adam.add(Activation('sigmoid'))

model\_q5\_adam.add(Dropout(0.10))

model\_q5\_adam.add(Dense(1, kernel\_initializer='random\_uniform'))

model\_q5\_adam.add(Activation('sigmoid'))

adam = Adamax(lr=0.1, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-6)

model\_q5\_adam.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=adam)

model\_q5\_adam.fit(x, y, epochs=5, batch\_size=10)

roc\_auc\_score(y\_test,model\_q5\_adam.predict(x\_test))

## 3xlayers, kernel\_initializer='random\_uniform, activation= 'sigmoid', optimizer= Adamax, batch\_size=500

## roc\_auc\_score: 0.8174560479875594

## from keras.optimizers import Adamax

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q5\_adam = Sequential()

model\_q5\_adam.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='random\_uniform')) # X\_train.shape[1] == 28 here

model\_q5\_adam.add(Activation('sigmoid'))

model\_q5\_adam.add(Dropout(0.10))

model\_q5\_adam.add(Dense(50, kernel\_initializer='random\_uniform'))

model\_q5\_adam.add(Activation('sigmoid'))

model\_q5\_adam.add(Dropout(0.10))

model\_q5\_adam.add(Dense(1, kernel\_initializer='random\_uniform'))

model\_q5\_adam.add(Activation('sigmoid'))

adam = Adamax(lr=0.1, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-6)

model\_q5\_adam.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=adam)

model\_q5\_adam.fit(x, y, epochs=5, batch\_size=500)

roc\_auc\_score(y\_test,model\_q5\_adam.predict(x\_test))

## 3xlayers, kernel\_initializer='random\_uniform, activation= 'sigmoid', optimizer= Adamax, batch\_size=1000

## roc\_auc\_score: 0.8235113259509186

## from keras.optimizers import Adamax

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q5\_adam = Sequential()

model\_q5\_adam.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q5\_adam.add(Activation('sigmoid'))

model\_q5\_adam.add(Dropout(0.10))

model\_q5\_adam.add(Dense(50, kernel\_initializer='uniform'))

model\_q5\_adam.add(Activation('sigmoid'))

model\_q5\_adam.add(Dropout(0.10))

model\_q5\_adam.add(Dense(1, kernel\_initializer='uniform'))

model\_q5\_adam.add(Activation('sigmoid'))

adam = Adamax(lr=0.1, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-6)

model\_q5\_adam.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=adam)

model\_q5\_adam.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q5\_adam.predict(x\_test))

## 3xlayers, kernel\_initializer='random\_uniform, activation= 'sigmoid', optimizer= Adamax, batch\_size=10000

## roc\_auc\_score:0.8213663083670677

## from keras.optimizers import Adamax

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q5\_adam = Sequential()

model\_q5\_adam.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q5\_adam.add(Activation('sigmoid'))

model\_q5\_adam.add(Dropout(0.10))

model\_q5\_adam.add(Dense(50, kernel\_initializer='uniform'))

model\_q5\_adam.add(Activation('sigmoid'))

model\_q5\_adam.add(Dropout(0.10))

model\_q5\_adam.add(Dense(1, kernel\_initializer='uniform'))

model\_q5\_adam.add(Activation('sigmoid'))

adam = Adamax(lr=0.1, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-6)

model\_q5\_adam.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=adam)

model\_q5\_adam.fit(x, y, epochs=5, batch\_size=1000)

roc\_auc\_score(y\_test,model\_q5\_adam.predict(x\_test))

## Questions to be answered (These are loaded questions—be warned they are there to test your understanding):

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

## final 3 layers

from keras.optimizers import Adamax

epochs = 15

batch\_size = 1000

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q7 = Sequential()

model\_q7.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q7.add(Activation('sigmoid'))

model\_q7.add(Dropout(0.10))

model\_q7.add(Dense(50, kernel\_initializer='uniform'))

model\_q7.add(Activation('sigmoid'))

model\_q7.add(Dropout(0.10))

model\_q7.add(Dense(1, kernel\_initializer='uniform'))

model\_q7.add(Activation('sigmoid'))

adam = Adamax(lr=0.1, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-6)

model\_q7.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=adam)

h = model\_q7.fit(

x,

y,

batch\_size=batch\_size,

epochs=epochs,

validation\_data=(x\_test, y\_test),

shuffle=True

)

show\_scores(model\_q7, h, x, y, x\_test, y\_test)

roc\_auc\_score(y\_test, model\_q7.predict(x\_test))

## final 5 layers

from keras.optimizers import Adamax

epochs = 15

batch\_size = 1000

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

model\_q8 = Sequential()

model\_q8.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform')) # X\_train.shape[1] == 28 here

model\_q8.add(Activation('sigmoid'))

model\_q8.add(Dropout(0.10))

model\_q8.add(Dense(50, kernel\_initializer='uniform'))

model\_q8.add(Activation('sigmoid'))

model\_q8.add(Dropout(0.10))

model\_q8.add(Dense(50, kernel\_initializer='uniform'))

model\_q8.add(Activation('sigmoid'))

model\_q8.add(Dropout(0.10))

model\_q8.add(Dense(50, kernel\_initializer='uniform'))

model\_q8.add(Activation('sigmoid'))

model\_q8.add(Dropout(0.10))

model\_q8.add(Dense(1, kernel\_initializer='uniform'))

model\_q8.add(Activation('sigmoid'))

adam = Adamax(lr=0.1, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-6)

model\_q8.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=adam)

h = model\_q8.fit(

x,

y,

batch\_size=batch\_size,

epochs=epochs,

validation\_data=(x\_test, y\_test),

shuffle=True

)

print('Saving model\_q8 to the file "model8.h5"')

model\_q8.save("model\_q8.h5")

show\_scores(model\_q8, h, x, y, x\_test, y\_test)

roc\_auc\_score(y\_test, model\_q8.predict(x\_test))

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

y = np.array(data.loc[:,0])

x = np.array(data.loc[:,1:])

x\_test = np.array(test\_data.loc[:,1:])

y\_test = np.array(test\_data.loc[:,0])

from keras.optimizers import Adamax

def build\_model8():

model\_q8 = Sequential()

model\_q8.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform'))

model\_q8.add(Activation('sigmoid'))

model\_q8.add(Dropout(0.10))

model\_q8.add(Dense(1, kernel\_initializer='uniform'))

model\_q8.add(Activation('sigmoid'))

adam = Adamax(lr=0.1, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-6)

model\_q8.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=adam)

return model\_q8

from keras.wrappers.scikit\_learn import KerasClassifier

keras\_model1 = build\_model8()

keras\_model1.fit(x, y, epochs=5, batch\_size=1000, verbose=1)

def build\_model9():

model\_q9 = Sequential()

model\_q9.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform'))

model\_q9.add(Activation('sigmoid'))

model\_q9.add(Dropout(0.10))

model\_q9.add(Dense(50, kernel\_initializer='uniform'))

model\_q9.add(Activation('sigmoid'))

model\_q9.add(Dropout(0.10))

model\_q9.add(Dense(1, kernel\_initializer='uniform'))

model\_q9.add(Activation('sigmoid'))

adam = Adamax(lr=0.1, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-6)

model\_q9.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=adam)

return model\_q9

from keras.wrappers.scikit\_learn import KerasClassifier

keras\_model = build\_model9()

keras\_model.fit(x, y, epochs=5, batch\_size=1000, verbose=1)

def build\_model10():

model\_q10 = Sequential()

model\_q10.add(Dense(50, input\_dim=x.shape[1], kernel\_initializer='uniform'))

model\_q10.add(Dropout(0.10))

model\_q10.add(Dense(50, kernel\_initializer='uniform'))

model\_q10.add(Activation('sigmoid'))

model\_q10.add(Dropout(0.10))

model\_q10.add(Dense(50, kernel\_initializer='uniform'))

model\_q10.add(Activation('sigmoid'))

model\_q10.add(Dropout(0.10))

model\_q10.add(Dense(50, kernel\_initializer='uniform'))

model\_q10.add(Activation('sigmoid'))

model\_q10.add(Dropout(0.10))

model\_q10.add(Dense(1, kernel\_initializer='uniform'))

model\_q10.add(Activation('sigmoid'))

adam = Adamax(lr=0.1, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-6)

model\_q10.compile(loss='binary\_crossentropy', metrics=['accuracy'], optimizer=adam)

return model\_q10

keras\_model2 = build\_model10()

keras\_model2.fit(x, y, epochs=5, batch\_size=1000, verbose=1)

from sklearn.metrics import roc\_curve

y\_pred\_keras = keras\_model.predict(x\_test).ravel()

fpr\_keras, tpr\_keras, thresholds\_keras = roc\_curve(y\_test, y\_pred\_keras)

y\_pred\_keras1 = keras\_model1.predict(x\_test).ravel()

fpr\_keras1, tpr\_keras1, thresholds\_keras1 = roc\_curve(y\_test, y\_pred\_keras1)

y\_pred\_keras2 = keras\_model2.predict(x\_test).ravel()

fpr\_keras2, tpr\_keras2, thresholds\_keras2 = roc\_curve(y\_test, y\_pred\_keras2)

from sklearn.metrics import auc

auc\_keras = auc(fpr\_keras, tpr\_keras)

auc\_keras1 = auc(fpr\_keras1, tpr\_keras1)

auc\_keras2 = auc(fpr\_keras2, tpr\_keras2)

from matplotlib import pyplot as plt

plt.figure(1)

plt.plot([0, 1], [0, 1], 'k--')

plt.plot(fpr\_keras, tpr\_keras, label='Layer 3 (area = {:.3f})'.format(auc\_keras))

plt.plot(fpr\_keras1, tpr\_keras1, label='Layer 2 (area = {:.3f})'.format(auc\_keras1))

plt.plot(fpr\_keras2, tpr\_keras2, label='Layer 5 (area = {:.3f})'.format(auc\_keras2))

plt.xlabel('False positive rate')

plt.ylabel('True positive rate')

plt.title('ROC curve')

plt.legend(loc='best')

plt.show()

roc\_auc\_score(y\_test, model\_11\_adam.predict(x\_test))