Neural Networks – Identifying HIGGS boson particles

Jeffery Lancon, Manisha Pednekar, David Stroud
MSDS 7333 - Quantifying the World - Case Study #12
04/02/2019

Abstract

The purpose of this study is to explore neural networks and the effects parameter selections have on the accuracy of a model. The study is an extension of work previously conducted by the authors of *Searching for exotic particles in high-energy physics with deep learning (BSW_HIGGS*), Baldi, P., P. Sadowski, and D. Whiteson [1][2]. BSW_HIGGS paper focused on identifying collisions at high-energy colliders the produce exotic particles (HIGGS bosons). Finding these particles requires identification of signature signals through background noise, classification problem, utilizing machine-learning methodology. We will employ the same benchmark data sets used for the BSW_HIGGS article, which is publicly available through UC Irvine's machine learning repository [3].

Introduction

Experimental physics is dedicated to discovering the fundamental structures of matter and how it relates to our understanding of the universe. To discover and identify the sub atomic particles, physicists use accelerators to collide proton/antiprotons to produce exotic particles. By studying these particles, physicists hope to gain critical insight into how our universe works at the most basic levels.

Since these exotic particles are too small to be observed, finding these particles requires identification of signature signals through background noise utilizing machine-learning methodologies, a classic classification problem. Machine-leaning classifiers, such as neural networks, can be instrumental in being able to distinguish noise from signals generate by exotic particles with a reasonable level of accuracy.

The most famous particle accelerator/collider is Large Hadron Collider (LHC) at the European Center for Nuclear Research (CERN). The LHC collides charged particles traveling at nearly lightspeed. The collisions break the particles into their subatomic constituents. The LHC produces nearly 10^{11} collisions per hour. Of those collisions, approximately 300 result in the creation of a HIGGS boson particle. With the shear volume of data being generated and the low success rate of creating HIGGS particles, accurate data analysis techniques are required to correctly identify and measure these particles.

Data Set

We will employ the same benchmark data sets used for the BSW_HIGGS article, which is publicly available through UC Irvine's machine learning repository [3]. The first variable in the data set is a classification variable, indicating if a HIGGS boson was detected. The next 21 features are kinematic properties measured by the particle accelerator's detectors. An additional 7 features are high level features derived by physicists to discriminate between the two classes. The first variable in the data set is a classification variable, indicating if a HIGGS boson was detected.

Note: 47% of the observations are classified as detecting a HIGGS particle, while 53% is classified as not detecting a HIGGS particle. This is much higher 'positive' (47%) rate than is present in actual data gathering scenarios (.0000003%). The data sets observations were chosen to have a higher 'positive' rate, to help improve the accuracy of the training models.

Neural Networks – Identifying HIGGS boson particles

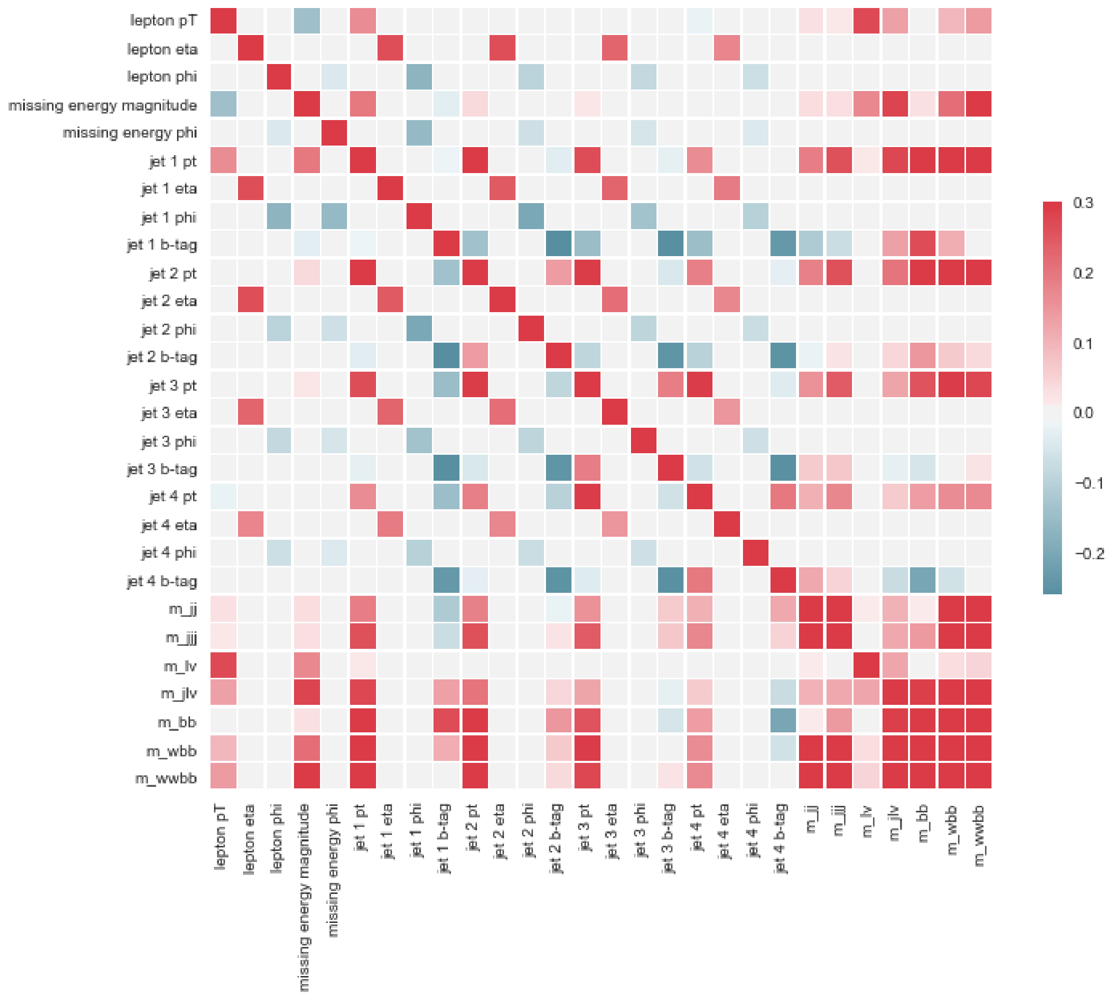
Jeffery Lancon, Manisha Pednekar, David Stroud
MSDS 7333 - Quantifying the World - Case Study #12
04/02/2019

The data set contains 11M observations X 28 attributes. 10.5M observations are training observations with last 500,000 observations being used as a test set.

The 28 attributes are named as follows: (21 low-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; followed by (7 high-level features): m_jj, m_jjj, m_lv, m_jlv, m_bb, m_wbb, m_wwbb

Since the seven high-level features are derived from the 21 low-level features, it would be expected that these features would be highly correlated with the other variables (Figure 1).

Figure – 1 Variable Correlation Matrix



The BSW_HIGGS paper performed preliminary analysis on the effect of the correlated variable on the overall model accuracy. BSW_HIGGS found that their models were more accurate when both the low-level features and high-level features were included in the training data set. Our analysis will also include both low-level and high-level features.

The BSW_HIGGS analysis did not modify, transform, add and/or remove any attributes from their model. To keep our base data set as like BSW_HIGGS', our analysis will also keep all attributes and will not perform any transformations/modifications.

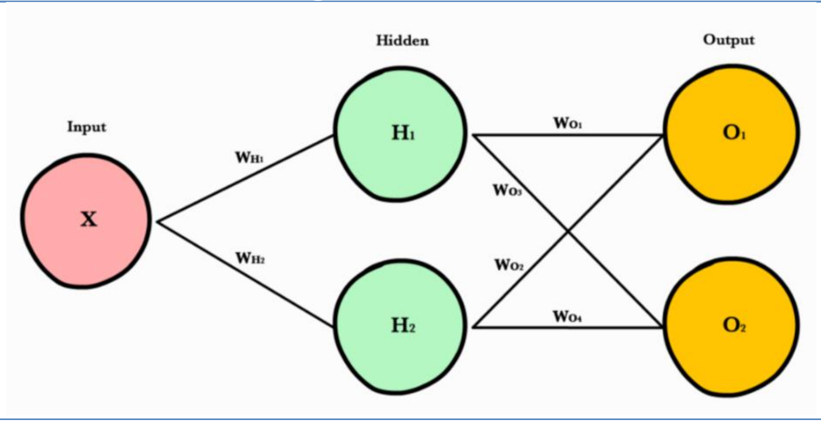
In this study, we will be using a subset of the HIGGS data set [3]. We will randomly select 2 Million observation, from the 10.5Million observation, to create our training dataset. We will use the last 500,000 observations as our test dataset.

Methods

Neural Networks

Neural Networks (NN) are a class of machine learning algorithms that use multiple hidden layers and non-linear activation functions that model complex patterns in data sets. They consist of an Input layer, multiple Hidden layers, and an Output layer, Figure 2. A NN takes the input and passes it through multiple layers of hidden neutrons allowing the input to learn at each of the hidden layers. The culmination of this process is a prediction that combines the inputs of all the neurons.

Figure – 2 Neural Network



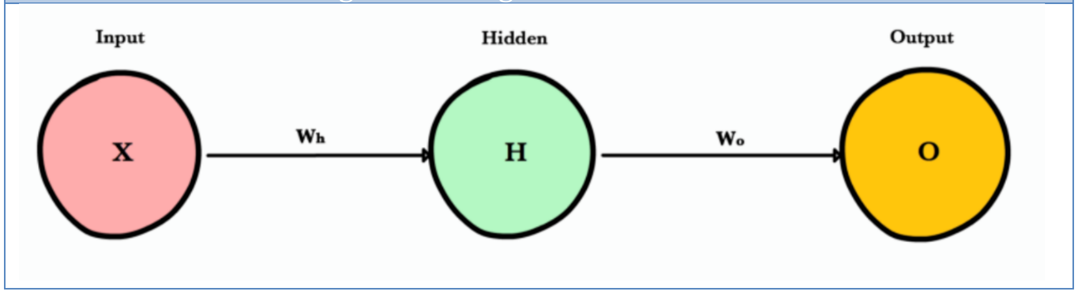
To grasp a firm understanding of the analysis, it is necessary to cover the nomenclature associated with NN's. We will cover the framework of a NN, as well as provide definitions for the major components. This review is designed to make our analysis more intuitive.

Layers

The Input layer hold the data your model will train on. A unique attribute in the datasets is represented with each neuron in the input layer. See Figure 3 for a visual representation of a neural network.

The Hidden layer is situated between the Input and Output layers. An application function is applied while the neuron is in the hidden layer and this is done prior to passing along to the output layer. Often, there are multiple hidden layers in traditional networks and each neural receives an updated weight prior to being sent to the next layer. In theory, this is supposed to give a more accurate output and works particularly well with non-linear functions.

Figure – 3 Single Neural Network

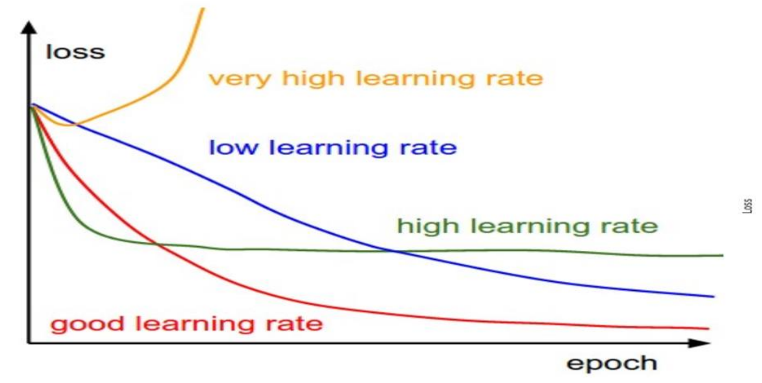


The Output layer is the last layer and contains the results of the Input layer, as well as the updates in the Hidden layer.

Sequential layer ~ the model needs to know what input shape is expected and this is determined at the sequential layer. This is the first and only sequential layer as the following layers do automatic shape inference.

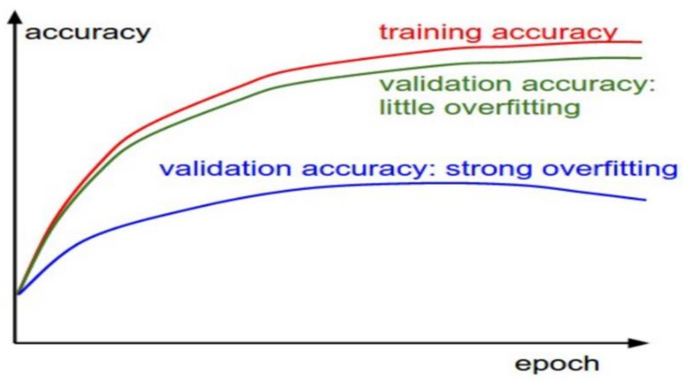
Loss ~ This is the first quantity that is useful to track and the lower the loss the better the model, see Figure 4. The loss value gives you an indication of how well or poorly your model performs after each iteration. Notice that loss have various levels of desired learning rates. The shape of the loss can provide information about the learning rate.

Figure – 4 Loss vs Epoch



Accuracy ~ If the number of test samples is 1,000,000 and the model classifies 742,000 of those correctly, then the model's accuracy is 74.20 percent, see Figure 5. Just as with the loss function, notice that the shape of the curve can provide information about the level of accuracy and the overfitted associated with accuracy.

Figure – 5 Accuracy vs Epoch



Activation Functions

Rectified Linear Units (ReLu) ~ A non-linear activation function that provides the same benefits as Sigmoid, but better performance. ReLu avoids and rectifies the vanishing gradient problem and is less computationally expensive than tanh and sigmoid due to simpler math associated with the function.

Sigmoid ~ Outputs a value between 0 and 1 as it takes a real value as input. Sigmoid is non-linear in nature, has a smooth gradient and works well as a classifier. Some of the drawbacks of the Sigmoid function is that Y values tend to respond less to changes in X toward either end of the sigmoid function. This can give rise to problems with 'vanishing gradients.'

Tanh ~ A non-linear, real number in the range of [-1, 1]. The derivatives are steeper for tanh and the gradient is stronger than sigmoid. Tanh suffers from a 'vanishing gradient' problem.

Adaptive Moment Estimation (Adam) ~ The Adam algorithm is used as a method for stochastic optimization and only requires first-order gradients. The method is computationally efficient, has little memory requirements and is well suited toward large datasets. Adam also realizes the benefits of both AdaGrad and RMSprop.

AdaDelta ~ An optimizer that implements the Adaptive Learning Rate method. This method is a per-dimension learning rate method for gradient descent and only relies on first order information.

Stochastic Gradient Descent ~ An optimization algorithm that performs a parameter update or each training example. Gradient is the calculation of a slope of error and descent is moving down along the slope towards some minimum level of error.

Batch ~ A hyperparameter that defines the number of samples to go through before updating the internal model parameters.

Epoch ~ Gives each sample in the training dataset an opportunity to update the model's parameters.

Root Mean Square Prop (RMSprop) ~ An optimization algorithm designed for NN. This algorithm resolves the problem of gradients that may vary widely in magnitudes.

Loss Functions

Cross-entropy loss measures the performance of a classification model where the output is a probability value between 0 and 1. As the predicted probability diverges from the actual label, the cross-entropy will increase.

TensorFlow

TensorFlow is an opensource platform used for Machine Learning and Deep Learning. Models are easy and intuitive to build with high-level API's that allow for easy debugging and model iteration. Our analysis was done with the TensorFlow and Kera's libraries.

Analysis

Our analyses begin with the development of three base neural network models, Model 1, Model 2, and Model 3. Model 1 is a shallow neural network with only one hidden layer. Model 2 is a medium and Model 3 is a deep neural network, consisting of 3 and 6 hidden layers. Table 1 provides a detailed description of the design parameters for each model.

Table – 1 Neural Network Model Parameters

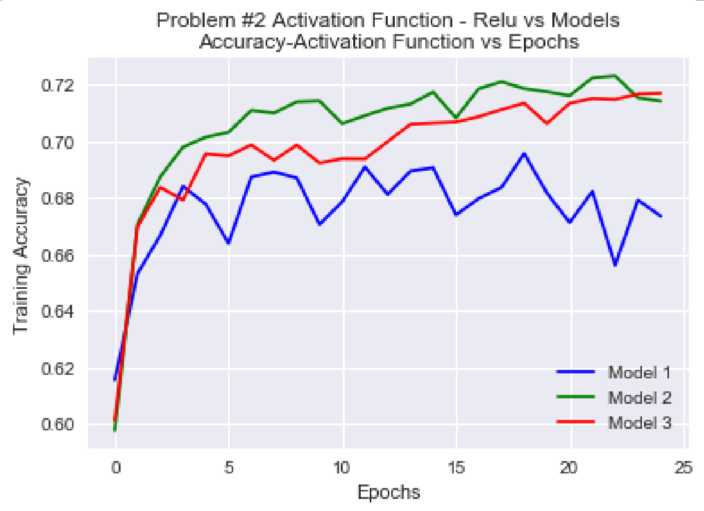
Global Parameters				
# of Observations:	2,000,000			
# of Attributes:	28			
Model Type:	Classification			
Neural Network				
Parameters		Model 1	Model 2	Model 3
Input Layer: Neurons		50	50	50
Hidden Layer (HL): Neurons / Dropout				
HL1		50 / 0.2	50 / 0.2	50 / 0.2
HL2			50 / 0.2	50 / 0.2
HL3			100 / 0.2	100 / 0.2
HL4				100 / 0.2
HL5				80 / 0.2
HL6				80 / 0.2
Output Layer: Neurons		1	1	1
Total Parameters:		4051	11751	36391
Analysis Parameters				
Epochs	25			
Batch Size	1000			
Model Metrics	accuracy			
Model Loss	binary_crossentropy			

Activation Functions	Sigmoid, tanh, relu
Kernel Initializers	Uniform, RandomNormal, Orthogonal, VarianceScaling
Optimizer	Adam, RMSprop, SGD, adadelta

Analysis-Base Model Comparison

Our first model has three different models, using the ReLu activation function (Figure 6). This first chart serves as a baseline to compare with our other models.

Figure – 6 Neural Network Base Model Comparison



This is a baseline of the three models using ReLu as activation functions and twenty five Epochs. Model 1 and 2 have similar accuracy rates in the approximate range of 72 percent.

Analysis-Activation Function Comparison

The three models, Figure 7A,7B,7C, contain three different architectures where we added and subtracted layers and neurons to show the variation in training accuracy.

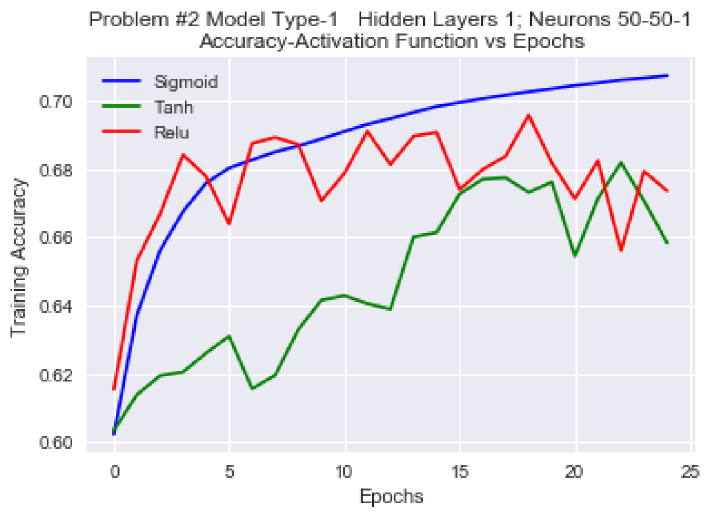
Figure – 7A Activation Function Comparison (Model 1)	
	The first model for this problem is reduced to one hidden layer with the fifty neurons. As expected, this is our worst performing model. This makes sense intuitively, as there is less analysis in the hidden layers and less learning for the neurons as they were passing from layer to layer in the previous models.

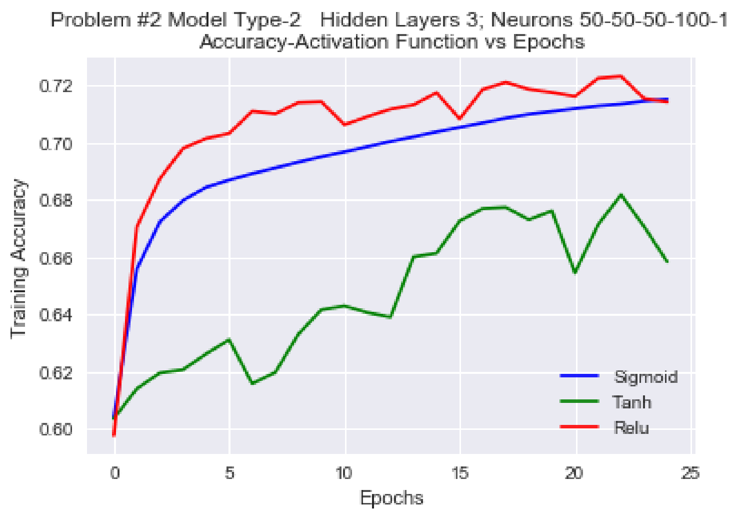
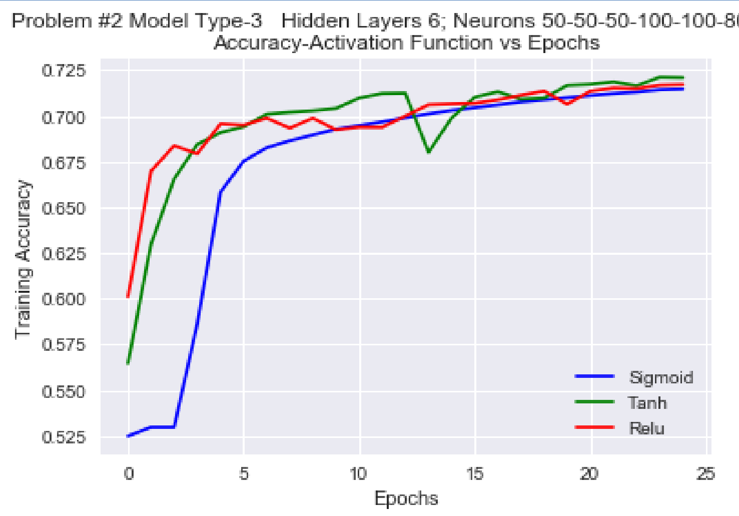
Figure – 7B Activation Function Comparison (Model 2)	
	This model increases the hidden layers from one to three and keeps the same approximate amount of neurons, proportional to the layers. ReLu and Sigmoid both end up with an approximate .71 per cent training accuracy. Tanh, however, ends up substantially lower with a training accuracy of .66 percent. This appears to be a case where Tanh suffers from the 'vanishing gradient' problem.

Figure – 7C Activation Function Comparison (Model 3)

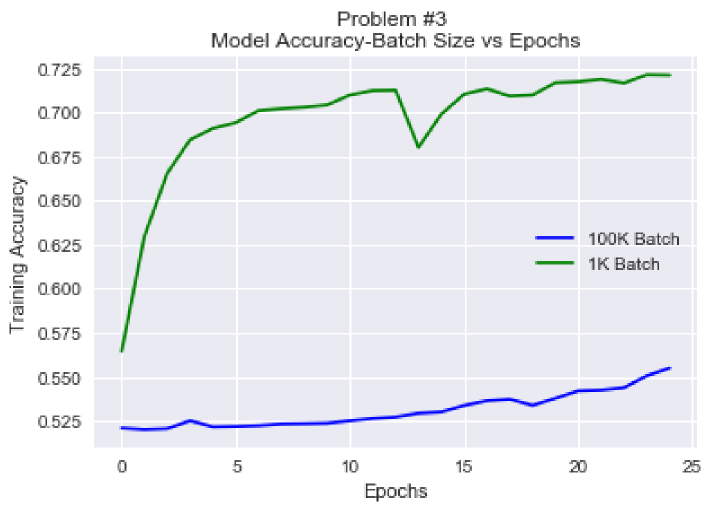


With 6 hidden layers and the largest amount of neurons, all three models ended up in the .72 percent range. Thus, from this model we are not really able to ascertain the differences in the three activation functions.

Analysis-Batch Size Comparison

The analysis below takes the best models from parts one and two and varies the batch size by at least two orders of magnitude, see Figure 8.

Figure – 8 Batch Size Comparison – (Model 3)

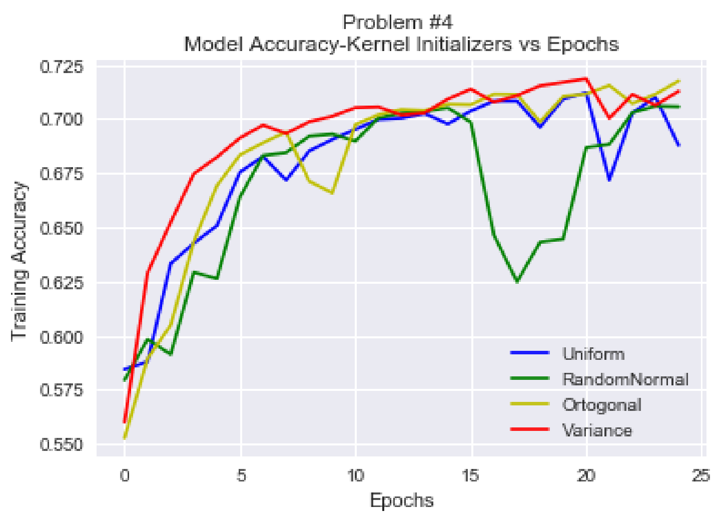


The clear winner in this model is the 1,000 batch. The model compares a substantially large batch of 100,000 training samples to a more reasonalby amount of 1,000 training sample per batch. It is made abundantly clear by the training model that 100K is an inferior batch size.

Analysis-Kernel Initializer Comparison

The model below takes the best score from parts one and two and uses three different kernel initializers, see Figure 9.

Figure – 9 Kernel Initializer Comparison – (Model 3)

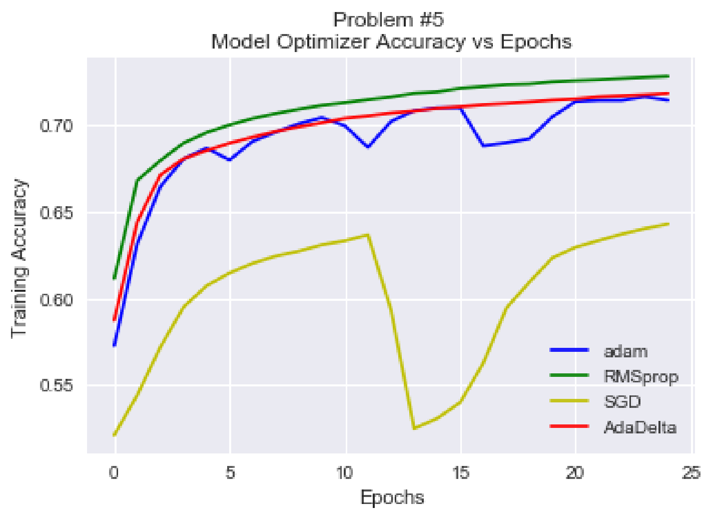


Initial weights of neural networks models are set by an initialization function i.e. Kernel_ initializer. Kernel_initializers can be fixed values, or statistical distribution. The initial weights will affect the rate at which the models learn and stabilize, as can be seen by the RandomNormal accuracy trend line, but the weights are continually updated each epoch and should not affect the overall outcome if an adequate number of epochs are executed.

Analysis-Kernel Initializer Comparison

We took the best results from our previous models and used three different optimizers, see Figure 10.

Figure – 10 Model Optimizer Comparison – (Model 3)



Interesting comparison of four different activation functions. SGD is the inferior funtion in this model. The other three functions are superior and their results are closely allinged with one another. RMSprop performs slightly better than the adam and AdaDelta. One could surmise that RMSprop has an advantage in dealing with the wideness of the gradients magnitudes.

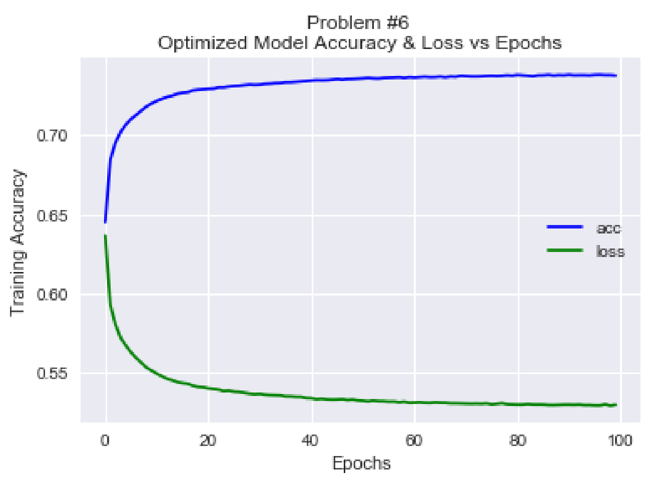
Analysis-Best Accuracy/AUC Score

Our last model is a culmination of all the models and is the model that produces the best score. This model encompasses all the lessons learned from the original models and parameter adjustments and placed them in a single model, see Table 2, Figure 11. The overall loss 0.5139 and accuracy 0.7469 were better than the any other model. The AUC was a respectable 0.827.

Table – 2 Neural Network Model Optimal Parameters

Global Parameters	
# of Observations:	2,000,000
# of Attributes:	28
Model Type:	Classification
Neural Network	
Parameters	Optimal Model
Input Layer: Neurons	50
Hidden Layer (HL): Neurons / Dropout	
HL1	50 / 0.2
HL2	50 / 0.2
HL3	50 / 0.2
HL4	50 / 0.2
HL5	5 / 0.2
HL6	50 / 0.2
Output Layer: Neurons	1
Total Parameters:	14251
Analysis Parameters	
Epochs	100
Batch Size	300
Model Metrics	accuracy
Model Loss	binary_crossentropy
Activation Functions	Tanh
Kernel Initializers	VarianceScaling
Optimizer	RMSprop

Figure – 11 Optimal Model



The goal of any Deep Learning problem is to maximize the training accuracy and minimize the loss. Given our best parameters from Problem 5, we analyze the difference in number of epochs that were run. With respect to loss and accuracy, both number go parabolic from 1 epoch to approximately 25 epochs. Past 40 epochs, we see only gradual amounts of improvements. This observation can save a substantial of time in your analysis, as each epoch run adds more time to running your model.

Discussions

The Higgs Boson Machine Learning Challenge is primarily a binary classification problem. From machine learning point of view this challenge problem can be resolved by many supervised classification algorithms. We used an ensemble of deep neural network solution which is computationally expensive. The neural network learns by adjusting its weights and bias (threshold) iteratively to yield the desired output. We think that we should explore decision tree algorithm with feature engineering as a possible solution next.

In this case study, we explored many hyperparameters of the deep neural network architecture using different activation functions, adding and subtracting the neuron layers, using different optimization functions with different learning rates, varying batch-sizes and epochs. Initially we developed build-train-evaluate-predict-score pipeline to explore majority of neural network hyperparameters. the training data is input to the network, and the desired output is known weights are adjusted until production yields desired value. We change the hyperparameters once we got a feel of how the data behaved, reacted to these changes and only attempted to increase complexity of our models as the classification accuracy improved.

Future Work

Neural networks (NN) in general is a field that is evolving, and new methods, procedures, and algorithms are constantly being developed. There are an endless number of leavers and knobs to tune when establishing the parameters for NN models. Time and compute resources often become the limiting factor on developing a NN model.

More Advanced Models

Due to time and compute resources restrictions, the models developed for this study were lacking in complexity and iterations.
All the models we analyzed were limited in the number of hidden layers and neurons we were able to develop due to compute resource restrictions. In all our models, it was noted that the accuracy and loss rates had not fully stabilized (stopped changing). This is a clear indication that the models developed had not been fully optimized. Previous studies [2] which used this data set analyzed 200 – 1000 epochs before their models reached their minimum error rates, which they defined as the change in error rate by a factor of 0.00001 over 10 epochs.
For future studies, we would like to expand the number of hidden layers and neurons within those layers by using cloud compute resources that are tailor made for these types of applications. This would allow for a more in-depth exploration of the models and their parameters as well as optimization of model by increasing the epochs to minimize the model error rates.

Data Source

The HIGGS data set was developed in hopes that researchers outside of CERN would be able to develop algorithms and models that would help physicist better detect HIGGS particles. In a way, it was developed for an open-ended research project that anyone could contribute to. Because of this, the data used has been 'cherry-picked and sanitized' and presented in a way that is conducive to novice and expert users being able to easily work with it. An example of this 'cherry-picking' is

the % of positive observations are well outside the normal level of observations that would be expected in an actual data set.
We would like to obtain raw data and test our algorithms on them. While we fully expect the accuracy of our models to be negatively affected by this new data set, we believe that it would be disingenuous to develop algorithms on 'false' data set, which when applied to a raw data set may or may-not be useful.

Variable Creation/Manipulation/Elimination

Our analyses were conducted using all 28 attributes: (21 low-level features, 7 high-level features). Future analysis work should include the exploration of these attributes and determine if they are beneficial in helping identify the classification of a HIGGS particle and determine if they should be included as an input to the classification algorithm.
Attribute manipulation should also be investigated, to determine if it would be beneficial in detecting HIGGS particles. Investigating each attribute, to identify patterns within the attribute data that might be exploited, to better differentiate the classification of HIGGS vs no HIGGS. This could be in the form of mathematical manipulation, square root, logarithm, inverse function, etc.
Like the current 7 high-level features, which are derived from the 21 low-level features, additional feature creation may aid in this classification analysis.

Conclusion

The purpose of this study is to explore neural networks and the effects parameter selections have on the accuracy of a model. The study is an extension of work previously conducted by the authors of *Searching for exotic particles in high-energy physics with deep learning (BSW_HIGGS).* We explored many hyperparameters of the deep neural network architecture, adding and subtracting the neuron layers, varying batch-sizes and epochs, all to understand the effects of these parameters have on the models themselves. Neural Networks has one of the largest 'learning curves' in both tuning of models and understanding the results, mainly due to the large number of 'leavers and dials' that can be tweaked and manipulated.

While manipulation of these parameters did have substantial effects on models that had relatively low number of epochs, it appears that most models converged to some sort of 'maximum' value when the models are trained using a substantial number of epochs (100+).

Deep neural networks do appear to be a valid methodology/algorithm to assist the physicist at CERN to identify HIGGS boson particles, but additional work will be needed to fully optimize the models.

References

[1] Baldi, P., P. Sadowski, and D. Whiteson. "Searching for Exotic Particles in High-energy Physics with Deep Learning." Nature Communications 5 (July 2, 2014)
[2] https://www.nature.com/articles/ncomms5308#discussion
[3] https://archive.ics.uci.edu/ml/datasets/HIGGS
https://arxiv.org/pdf/1212.5701.pdf

http://cs231n.github.io/neural-networks-3/#loss
https://arxiv.org/pdf/1609.04747.pdf
https://arxiv.org/pdf/1412.6980.pdf
https://keras.io/getting-started/sequential-model-guide/
https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html
https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a

Neural Networks – Identifying HIGGS boson particles

Jeffery Lancon, Manisha Pednekar, David Stroud
MSDS 7333 - Quantifying the World - Case Study #12
04/02/2019

Appendix A

```
"""
#  MSDS 7333 - Quantifying the World - Case Study #12
#  Neural Networks
#  Team Members:
#         Jeffery Lancon, Manisha Pednekar, David Stroud
#  Date: 04/02/2019
"""
import os
import tensorflow as tf
import pandas as pd
import numpy as np
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt

N=2000000 #Chosen as a decent compromise between sample size and limitations of hardware
input_data=pd.read_csv("C:/Users/Prodigy/Documents/Personal Info/SMU/MSDS 7333 - Quantifying the
World/Unit_12/HIGGS.csv",nrows=10500000,header=None)

cc = input_data.iloc[:,1:]

# Creating a random sample of observations from the 10.5M observations
np.random.seed(42)
data = input_data.sample(n=N)

# Selecting the 500K test data set (last 500K rows in the data set)
test_data=pd.read_csv("C:/Users/Prodigy/Documents/Personal Info/SMU/MSDS 7333 - Quantifying the
World/Unit_12//HIGGS.csv",nrows=500000,header=None,skiprows=10500000)

###############################
##----  Correlation Matrix
#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):
cc.columns = ['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']

corr = cc.corr()
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)
```

```
# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, cmap=cmap, vmax=.3, center=0,
        square=True, linewidths=.5, cbar_kws={"shrink": .5})

################################################


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
from keras import optimizers

y_train = np.array(data.loc[:,0])
y_train = y_train.reshape(-1,1) # nessecarily for method 2
x_train = np.array(data.loc[:,1:])
x_test = np.array(test_data.loc[:,1:])
y_test = np.array(test_data.loc[:,0])



# Model #1 Hidden Layers-1
# Number of neurons;  50
# Dropout % 0.20
act = tf.nn.relu
np.random.seed(42)
model1 = tf.keras.models.Sequential([
  tf.keras.layers.Dense(50, input_shape=(28,), activation=act, kernel_initializer='random_uniform'),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(1, activation=act)
])

model1.compile(optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy'])

model1.fit(x_train, y_train, epochs=25,batch_size=1000)
# See Sheet
model1.evaluate(x_test, y_test)
  #sigmoid 500000/500000 [==============================] - 10s 20us/sample - loss: 0.5463 - acc: 0.7192
  #Out[67]: [0.5462561998291016, 0.719156]
    #tanh 500000/500000 [==============================] - 12s 24us/sample - loss: 0.6074 - acc: 0.6611
    #Out[109]: [0.6074042194004059, 0.66107]
      #relu 500000/500000 [==============================] - 13s 25us/sample - loss: 0.5833 - acc: 0.6935
      #Out[136]: [0.58332225872612, 0.693536]
roc_auc_score(y_test,model1.predict(x_test))
  #sigmoid Out[68]: 0.7955977870008317
    #tanh Out[110]: 0.7491576370383417
      #relu Out[137]: 0.7656150409797512
model1.summary()
  #Layer (type)              Output Shape            Param #
  #=================================================================
  #dense_13 (Dense)          (None, 50)              1450
  #_________________________________________________________________
  #dense_14 (Dense)          (None, 50)              2550
  #_________________________________________________________________
  #dropout_7 (Dropout)       (None, 50)              0
  #_________________________________________________________________
  #dense_15 (Dense)          (None, 1)               51
  #=================================================================
  #Total params: 4,051
  #Trainable params: 4,051
  #Non-trainable params: 0
  #_________________________________________________________________

#Saving models for future use
model1_2_2_relu_25epoch = model1.history.history
model1.save('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Models/my_model
1_2_2_relu_25epoch.h5')
#model1_sigmoid_25epoch['acc'][1]



# Model #2 Hidden Layers-3
# Number of neurons;  50,50,100
# Dropout % 0.20
model2 = tf.keras.models.Sequential([
 tf.keras.layers.Dense(50, input_shape=(28,), activation=act),
 tf.keras.layers.Dense(50, activation=act),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(50, activation=act),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(100, activation=act),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(1, activation=act)
])
model2.compile(optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy'])
model2.fit(x_train, y_train, epochs=25,batch_size=1000)
```

```
# See Spreadsheet
model2.evaluate(x_test, y_test)
  #sigmoid 500000/500000 [==============================] - 16s 32us/sample - loss: 0.5434 - acc: 0.7204
  #Out[123]: [0.5434014899806976, 0.720364]
    #tanh 500000/500000 [==============================] - 15s 29us/sample - loss: 0.5446 - acc: 0.7214
    #Out[115]: [0.5446132134284973, 0.721358]
      #relu 500000/500000 [==============================] - 15s 30us/sample - loss: 0.5746 - acc: 0.7117
      #Out[141]: [0.574634434967041, 0.711668]
roc_auc_score(y_test,model2.predict(x_test))
  #sigmoid Out[73]: 0.7909795835368331
    #tanh Out[116]: 0.7974080490996827
      #relu Out[142]: 0.7836337294987561
model2.summary()
  #_________________________________________________________________
  #Layer (type)                 Output Shape              Param #
  #=================================================================
  #dense_16 (Dense)             (None, 50)                1450
  #_________________________________________________________________
  #dense_17 (Dense)             (None, 50)                2550
  #_________________________________________________________________
  #dropout_8 (Dropout)          (None, 50)                0
  #_________________________________________________________________
  #dense_18 (Dense)             (None, 50)                2550
  #_________________________________________________________________
  #dropout_9 (Dropout)          (None, 50)                0
  #_________________________________________________________________
  #dense_19 (Dense)             (None, 100)               5100
  #_________________________________________________________________
  #dropout_10 (Dropout)         (None, 100)               0
  #_________________________________________________________________
  #dense_20 (Dense)             (None, 1)                 101
  #=================================================================
  #Total params: 11,751
  #Trainable params: 11,751
  #Non-trainable params: 0
  #_________________________________________________________________


#Saving models for future use
model2_2_2_relu_25epoch = model2.history.history
model2.save('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Models/my_model
2_2_2_relu_25epoch.h5')



# Model #3 Hidden Layers-6
# Number of neurons;  50,50,100,100,80,80
# Dropout % 0.20
model3 = tf.keras.models.Sequential([
  tf.keras.layers.Dense(50, input_shape=(28,), activation=act),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(100, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(100, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(80, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(80, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(1, activation=act)
])
model3.compile(optimizer='adam',
          loss='binary_crossentropy',
          metrics=['accuracy'])

model3.fit(x_train, y_train, epochs=25,batch_size=1000)

model3.evaluate(x_test, y_test)
   #Sigmoid 500000/500000 [==============================] - 17s 34us/sample - loss: 0.5463 - acc: 0.7191
   #Out[86]: [0.5463351220207214, 0.719114]
      #tanh 500000/500000 [==============================] - 19s 38us/sample - loss: 0.5283 - acc: 0.7342
      #Out[130]: [0.5283023863420486, 0.734198]
         #relu 500000/500000 [==============================] - 19s 38us/sample - loss: 0.5584 - acc: 0.7094
         #Out[146]: [0.5583731457977295, 0.70944]

roc_auc_score(y_test,model3.predict(x_test))
   #sigmoid Out[87]: 0.7951442546686375
      #tanh Out[131]: 0.8124479478926048
         #relu Out[147]: 0.8005893624291227

model3.summary()
   #Layer (type)              Output Shape              Param #
   #=================================================================
   #dense_36 (Dense)           (None, 50)                1450
   #_________________________________________________________________
   #dense_37 (Dense)           (None, 50)                2550
   #_________________________________________________________________
   #dropout_16 (Dropout)       (None, 50)                0
   #_________________________________________________________________
   #dense_38 (Dense)          (None, 50)            2550
   #_________________________________________________________________
   #dropout_17 (Dropout)       (None, 50)            0
   #_________________________________________________________________
   #dense_39 (Dense)          (None, 100)           5100
   #_________________________________________________________________
   #dropout_18 (Dropout)       (None, 100)           0
   #_________________________________________________________________
   #dense_40 (Dense)          (None, 100)           10100
   #_________________________________________________________________
   #dropout_19 (Dropout)       (None, 100)           0
   #_________________________________________________________________
   #dense_41 (Dense)          (None, 80)            8080
   #_________________________________________________________________
   #dropout_20 (Dropout)       (None, 80)            0
   #_________________________________________________________________
   #dense_42 (Dense)          (None, 80)            6480
   #_________________________________________________________________
   #dropout_21 (Dropout)       (None, 80)            0
   #_________________________________________________________________
   #dense_43 (Dense)          (None, 1)             81
   #=================================================================
   #Total params: 36,391
   #Trainable params: 36,391
   #Non-trainable params: 0
   #_________________________________________________________________

model3_2_2_relu_25epoch = model3.history.history
model3.save('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Models/my_model
3_2_2_relu_25epoch.h5')




Prob11_SigmoidDF = pd.DataFrame.from_dict(model1_sigmoid_25epoch)
Prob12_SigmoidDF = pd.DataFrame.from_dict(model2_sigmoid_25epoch)
Prob13_SigmoidDF = pd.DataFrame.from_dict(model3_sigmoid_25epoch)
Prob1_SigmoidDF = pd.concat([Prob11_SigmoidDF,Prob12_SigmoidDF,Prob13_SigmoidDF],axis=1)
Prob1_SigmoidDF.columns = ['model1_sigmoid_25epoch_loss','model1_sigmoid_25epoch_acc',
                 'model2_sigmoid_25epoch_loss','model2_sigmoid_25epoch_acc',
                 'model3_sigmoid_25epoch_loss','model3_sigmoid_25epoch_acc']
Prob1_SigmoidDF.to_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Prob1_
SigmoidDF.csv')

########################################
# Problem #2 graphics - Activation Functions - Sigmoid
```

```
Prob2A_DF_plot=pd.read_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Pr
ob1_SigmoidDF.csv')
Prob2B_DF_plot=pd.read_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Pr
ob2_tanhDF.csv')
Prob2C_DF_plot=pd.read_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Pr
ob2_reluDF.csv')

fig, ax = plt.subplots()
ax.plot(Prob2A_DF_plot.model1_sigmoid_25epoch_acc, '-b', label='Sigmoid')
ax.plot(Prob2B_DF_plot.model1_tanh_25epoch_acc, '-g', label='Tanh')
ax.plot(Prob2C_DF_plot.model1_relu_25epoch_acc, '-g', label='Relu')
plt.title('Problem #2 Model Type-1   Hidden Layers 1; Neurons 50-50-1\nAccuracy-Activation Function vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Training Accuracy')
leg = ax.legend();

##########################################




Mod1_2_1_tanhDF = pd.DataFrame.from_dict(model1_2_1_tanh_25epoch)
Mod2_2_1_tanhDF = pd.DataFrame.from_dict(model2_2_1_tanh_25epoch)
Mod3_2_1_tanhDF = pd.DataFrame.from_dict(model3_2_1_tanh_25epoch)
Prob2_tanhDF = pd.concat([Mod1_2_1_tanhDF,Mod2_2_1_tanhDF,Mod3_2_1_tanhDF],axis=1)
Prob2_tanhDF.columns = ['model1_tanh_25epoch_loss','model1_tanh_25epoch_acc',
                'model2_tanh_25epoch_loss','model2_tanh_25epoch_acc',
                'model3_tanh_25epoch_loss','model3_tanh_25epoch_acc']
Prob2_tanhDF.to_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Prob2_tan
hDF.csv')


Mod1_2_2_reluDF = pd.DataFrame.from_dict(model1_2_2_relu_25epoch)
Mod2_2_2_reluDF = pd.DataFrame.from_dict(model2_2_2_relu_25epoch)
Mod3_2_2_reluDF = pd.DataFrame.from_dict(model3_2_2_relu_25epoch)
Prob2_reluDF = pd.concat([Mod1_2_2_reluDF,Mod2_2_2_reluDF,Mod3_2_2_reluDF],axis=1)
Prob2_reluDF.columns = ['model1_relu_25epoch_loss','model1_relu_25epoch_acc',
                'model2_relu_25epoch_loss','model2_relu_25epoch_acc',
                'model3_relu_25epoch_loss','model3_relu_25epoch_acc']
Prob2_reluDF.to_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Prob2_relu
DF.csv')


#########################################
# Problem #2 graphics - Activation Functions - Model1
```

```
Prob2A_DF_plot=pd.read_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Pr
ob1_SigmoidDF.csv')
Prob2B_DF_plot=pd.read_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Pr
ob2_tanhDF.csv')
Prob2C_DF_plot=pd.read_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Pr
ob2_reluDF.csv')

fig, ax = plt.subplots()
ax.plot(Prob2A_DF_plot.model1_sigmoid_25epoch_acc, '-b', label='Sigmoid')
ax.plot(Prob2B_DF_plot.model1_tanh_25epoch_acc, '-g', label='Tanh')
ax.plot(Prob2C_DF_plot.model1_relu_25epoch_acc, '-r', label='Relu')
plt.title('Problem #2 Model Type-1   Hidden Layers 1; Neurons 50-50-1\nAccuracy-Activation Function vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Training Accuracy')
leg = ax.legend();

#------
# graphics - Activation Functions - Model2

fig, ax = plt.subplots()
ax.plot(Prob2A_DF_plot.model2_sigmoid_25epoch_acc, '-b', label='Sigmoid')
ax.plot(Prob2B_DF_plot.model2_tanh_25epoch_acc, '-g', label='Tanh')
ax.plot(Prob2C_DF_plot.model2_relu_25epoch_acc, '-r', label='Relu')
plt.title('Problem #2 Model Type-2   Hidden Layers 3; Neurons 50-50-50-100-1\nAccuracy-Activation Function vs
Epochs')
plt.xlabel('Epochs')
plt.ylabel('Training Accuracy')
leg = ax.legend();

#------
#graphics - Activation Functions - Model3

fig, ax = plt.subplots()
ax.plot(Prob2A_DF_plot.model3_sigmoid_25epoch_acc, '-b', label='Sigmoid')
ax.plot(Prob2B_DF_plot.model3_tanh_25epoch_acc, '-g', label='Tanh')
ax.plot(Prob2C_DF_plot.model3_relu_25epoch_acc, '-r', label='Relu')
plt.title('Problem #2 Model Type-3   Hidden Layers 6; Neurons 50-50-50-100-100-80-80-1\nAccuracy-Activation
Function vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Training Accuracy')
leg = ax.legend();


#------
#graphics - Activation Functions - tanh - comparing Models
```

```
fig, ax = plt.subplots()
ax.plot(Prob2C_DF_plot.model1_relu_25epoch_acc, '-b', label='Model 1')
ax.plot(Prob2C_DF_plot.model2_relu_25epoch_acc, '-g', label='Model 2')
ax.plot(Prob2C_DF_plot.model3_relu_25epoch_acc, '-r', label='Model 3')
plt.title('Problem #2 Activation Function - Relu vs Models\nAccuracy-Activation Function vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Training Accuracy')
leg = ax.legend();



##########################################################################
##########################################################################
#Problem #3

# Best Model: Model 3; Activation Function - tanh

# Running Model with batch size at least 2 orders of magnitude

act = tf.nn.tanh
np.random.seed(42)
# Model #3 Hidden Layers-6
# Number of neurons;  50,50,100,100,80,80
# Dropout % 0.20
model3 = tf.keras.models.Sequential([
  tf.keras.layers.Dense(50, input_shape=(28,), activation=act),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(100, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(100, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(80, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(80, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(1, activation=act)
])
model3.compile(optimizer='adam',
          loss='binary_crossentropy',
          metrics=['accuracy'])

model3.fit(x_train, y_train, epochs=25,batch_size=1000)
```

```
model3.evaluate(x_test, y_test)
  #500000/500000 [==============================] - 17s 33us/sample - loss: 0.6703 - acc: 0.5843
  #Out[9]: [0.6703165130882264, 0.584286]
roc_auc_score(y_test,model3.predict(x_test))
  #Out[10]: 0.634416375272185
model3.summary()
  #_________________________________________________________________
  #Layer (type)             Output Shape            Param #
  #=================================================================
  #dense (Dense)             (None, 50)              1450
  #_________________________________________________________________
  #dense_1 (Dense)           (None, 50)              2550
  #_________________________________________________________________
  #dropout (Dropout)         (None, 50)              0
  #_________________________________________________________________
  #dense_2 (Dense)           (None, 50)              2550
  #_________________________________________________________________
  #dropout_1 (Dropout)       (None, 50)              0
  #_________________________________________________________________
  #dense_3 (Dense)           (None, 100)             5100
  #_________________________________________________________________
  #dropout_2 (Dropout)       (None, 100)             0
  #_________________________________________________________________
  #dense_4 (Dense)           (None, 100)             10100
  #_________________________________________________________________
  #dropout_3 (Dropout)       (None, 100)             0
  #_________________________________________________________________
  #dense_5 (Dense)           (None, 80)              8080
  #_________________________________________________________________
  #dropout_4 (Dropout)       (None, 80)              0
  #_________________________________________________________________
  #dense_6 (Dense)           (None, 80)              6480
  #_________________________________________________________________
  #dropout_5 (Dropout)       (None, 80)              0
  #_________________________________________________________________
  #dense_7 (Dense)           (None, 1)               81
  #=================================================================
  #Total params: 36,391
  #Trainable params: 36,391
  #Non-trainable params: 0
  #_________________________________________________________________

model3_tanh_25epoch_100Kbatch = model3.history.history
model3.save('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Models/my_model
_Prob3_tanh_25epoch_100Kbatch.h5')
```

```
Prob3_tanh_25epoch_100kbatchDF = pd.DataFrame.from_dict(model3_tanh_25epoch_100Kbatch)
Prob3_tanh_25epoch_100kbatchDF.columns =
['Prob3_tanh_25epoch_100kbatchDF_loss','Prob3_tanh_25epoch_100kbatchDF_acc']
Prob3_tanh_25epoch_100kbatchDF.to_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/Case
Study_12/Prob3_tanh_25epoch_100kbatchDF.csv')

#########################################
# Problem #3 graphics

Prob3A_DF_plot=pd.read_csv("C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Pr
ob3_tanh_25epoch_100kbatchDF.csv")
Prob3B_DF_plot=pd.read_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Pr
ob2_tanhDF.csv')
fig, ax = plt.subplots()
ax.plot(Prob3A_DF_plot.Prob3_tanh_25epoch_100kbatchDF_acc, '-b', label='100K Batch')
ax.plot(Prob3B_DF_plot.model3_tanh_25epoch_acc, '-g', label='1K Batch')
plt.title('Problem #3 \nModel Accuracy-Batch Size vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Training Accuracy')
leg = ax.legend();

###########################################


#########################################################################
#########################################################################
#  Problem #4 Take best model from 3 and try 3 different kernal initializers


act = tf.nn.tanh
kern = 'VarianceScaling'
np.random.seed(42)
# Model #3 Hidden Layers-6
# Number of neurons;  50,50,100,100,80,80
# Dropout % 0.20
model4 = tf.keras.models.Sequential([
  tf.keras.layers.Dense(50, input_shape=(28,), activation=act,kernel_initializer=kern),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(100, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(100, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(80, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(80, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(1, activation=act)
])
model4.compile(optimizer='adam',
          loss='binary_crossentropy',
          metrics=['accuracy'])

model4.fit(x_train, y_train, epochs=25,batch_size=1000)

model4.evaluate(x_test, y_test)
  #uniform 500000/500000 [==============================] - 18s 36us/sample - loss: 0.5617 - acc: 0.7117
  #Out[21]: [0.5616875746669769, 0.711664]
    #RandomNormal 500000/500000 [==============================] - 18s 36us/sample - loss: 0.5615 - acc:
0.7146
    #Out[27]: [0.5614667445354462, 0.714616]
      #Orthogonal 500000/500000 [==============================] - 18s 37us/sample - loss: 0.5388 - acc:
0.7249
      #Out[33]: [0.5388194880161286, 0.724886]
        #VarianceScaling 500000/500000 [==============================] - 19s 38us/sample - loss: 0.5324 -
acc: 0.7296
        #Out[38]: [0.5324102998027801, 0.729626]

roc_auc_score(y_test,model4.predict(x_test))
  #uniform Out[22]: 0.7825732460190485
    #RandomNormal Out[28]: 0.7897035163009849
      #Ortogonal Out[34]: 0.8037352701470606
        #VarianceScaling Out[39]: 0.807966897119839

model4.summary()
  #Layer (type)             Output Shape          Param #
  #=================================================================
  #dense_16 (Dense)          (None, 50)            1450
  #_________________________________________________________________
  #dense_17 (Dense)          (None, 50)            2550
  #_________________________________________________________________
  #dropout_12 (Dropout)        (None, 50)            0
  #_________________________________________________________________
  #dense_18 (Dense)          (None, 50)            2550
  #_________________________________________________________________
  #dropout_13 (Dropout)        (None, 50)            0
  #_________________________________________________________________
  #dense_19 (Dense)          (None, 100)           5100
  #_________________________________________________________________
  #dropout_14 (Dropout)        (None, 100)           0
```

```
    #_________________________________________________________________
    #dense_20 (Dense)          (None, 100)            10100
    #_________________________________________________________________
    #dropout_15 (Dropout)       (None, 100)            0
    #_________________________________________________________________
    #dense_21 (Dense)          (None, 80)             8080
    #_________________________________________________________________
    #dropout_16 (Dropout)       (None, 80)             0
    #_________________________________________________________________
    #dense_22 (Dense)          (None, 80)             6480
    #_________________________________________________________________
    #dropout_17 (Dropout)       (None, 80)             0
    #_________________________________________________________________
    #dense_23 (Dense)          (None, 1)              81
    #=================================================================
    #Total params: 36,391
    #Trainable params: 36,391
    #Non-trainable params: 0
    #_________________________________________________________________


model4_4_VarianceScaling = model4.history.history
model4.save('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Models/my_model
4_4_VarianceScaling.h5')



Prob4_uniformDF = pd.DataFrame.from_dict(model4_1_uniform)
Prob4_RandomNormalDF = pd.DataFrame.from_dict(model4_2_RandomNormal)
Prob4_OrtogonalDF = pd.DataFrame.from_dict(model4_3_Ortogonal)
Prob4_VarianceScalingDF = pd.DataFrame.from_dict(model4_4_VarianceScaling)

Prob4_DF =
pd.concat([Prob4_uniformDF,Prob4_RandomNormalDF,Prob4_OrtogonalDF,Prob4_VarianceScalingDF],axis=1)
Prob4_DF.columns = ['uniform_loss','uniform_acc',
                   'RandomNormal_loss','RandomNormal_acc',
                   'Ortogonal_loss','Ortogonal_25epoch_acc',
                   'Variance_loss','Variance_acc']
Prob4_DF.to_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Prob4_DF.csv')

##########################################
# Problem #4 graphics

Prob4_DF_plot=pd.read_csv("C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Pro
b4_DF.csv")
```

```
fig, ax = plt.subplots()
ax.plot(Prob4_DF_plot.uniform_acc, '-b', label='Uniform')
ax.plot(Prob4_DF_plot.RandomNormal_acc, '-g', label='RandomNormal')
ax.plot(Prob4_DF_plot.Ortogonal_25epoch_acc, '-y', label='Ortogonal')
ax.plot(Prob4_DF_plot.Variance_acc, '-r', label='Variance')
plt.title('Problem #4 \nModel Accuracy-Kernel Initializers vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Training Accuracy')
leg = ax.legend();

##############################################







#################################################################################
#################################################################################
#  Problem #5 Take best model from 3 and try 3 different optimizers


act = tf.nn.tanh
np.random.seed(42)
# Model #3 Hidden Layers-6
# Number of neurons;  50,50,100,100,80,80
# Dropout % 0.20
model5 = tf.keras.models.Sequential([
  tf.keras.layers.Dense(50, input_shape=(28,), activation=act),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(100, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(100, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(80, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(80, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(1, activation=act)
])
model5.compile(optimizer='adadelta',
          loss='binary_crossentropy',
          metrics=['accuracy'])

model5.fit(x_train, y_train, epochs=25,batch_size=1000)

model5.evaluate(x_test, y_test)
  #adam 500000/500000 [==============================] - 20s 40us/sample - loss: 0.5433 - acc: 0.7274
  #Out[51]: [0.5433235804748535, 0.727376]
    #RMSprop 500000/500000 [==============================] - 20s 40us/sample - loss: 0.5189 - acc:
0.7406
    #Out[62]: [0.5189299085597991, 0.74063]
      #sgd 500000/500000 [==============================] - 19s 37us/sample - loss: 0.6297 - acc: 0.6502
      #Out[68]: [0.6296809244155884, 0.650238]
        #adadelta 500000/500000 [==============================] - 19s 39us/sample - loss: 0.5366 - acc:
0.7273
        #Out[75]: [0.5366370195713043, 0.727302]

roc_auc_score(y_test,model5.predict(x_test))
  #adam Out[52]: 0.8069738775920415
    #RMSprop Out[63]: 0.8201887933972172
      #sgd Out[69]: 0.6956203140084707
        #adadelta Out[76]: 0.8047374342268395

model5.summary()
  #_________________________________________________________________
  #Layer (type)                 Output Shape              Param #
  #=================================================================
  #dense_96 (Dense)             (None, 50)                1450
  #_________________________________________________________________
  #dense_97 (Dense)             (None, 50)                2550
  #_________________________________________________________________
  #dropout_72 (Dropout)         (None, 50)                0
  #_________________________________________________________________
  #dense_98 (Dense)             (None, 50)                2550
  #_________________________________________________________________
  #dropout_73 (Dropout)         (None, 50)                0
  #_________________________________________________________________
  #dense_99 (Dense)             (None, 100)               5100
  #_________________________________________________________________
  #dropout_74 (Dropout)         (None, 100)               0
  #_________________________________________________________________
  #dense_100 (Dense)            (None, 100)               10100
  #_________________________________________________________________
  #dropout_75 (Dropout)         (None, 100)               0
  #_________________________________________________________________
  #dense_101 (Dense)            (None, 80)                8080
  #_________________________________________________________________
```

```
    #dropout_76 (Dropout)        (None, 80)             0
    #_________________________________________________________________
    #dense_102 (Dense)           (None, 80)             6480
    #_________________________________________________________________
    #dropout_77 (Dropout)        (None, 80)             0
    #_________________________________________________________________
    #dense_103 (Dense)           (None, 1)              81
    #=================================================================
    #Total params: 36,391
    #Trainable params: 36,391
    #Non-trainable params: 0
    #_________________________________________________________________


model5_4_adadelta = model5.history.history
model5.save('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Models/my_model
5_4_adadelta.h5')

Prob5_adamDF = pd.DataFrame.from_dict(model5_1_adam)
Prob5_RMSpropDF = pd.DataFrame.from_dict(model5_2_RMSprop)
Prob5_sgdDF = pd.DataFrame.from_dict(model5_3_sgd)
Prob5_adadeltaDF = pd.DataFrame.from_dict(model5_4_adadelta)

Prob5_DF = pd.concat([Prob5_adamDF,Prob5_RMSpropDF,Prob5_sgdDF,Prob5_adadeltaDF],axis=1)
Prob5_DF.columns = ['adam_loss','adam_acc',
                    'RMSprop_loss','RMSprop_acc',
                    'sgd_loss','sgd_acc',
                    'adadelta_loss','adadelta_acc']
Prob5_DF.to_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Prob5_DF.csv')

########################################
# Problem #5 graphics

Prob5_DF_plot=pd.read_csv("C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Pro
b5_DF.csv")

fig, ax = plt.subplots()
ax.plot(Prob5_DF_plot.adam_acc, '-b', label='adam')
ax.plot(Prob5_DF_plot.RMSprop_acc, '-g', label='RMSprop')
ax.plot(Prob5_DF_plot.sgd_acc, '-y', label='SGD')
ax.plot(Prob5_DF_plot.adadelta_acc, '-r', label='AdaDelta')
plt.title('Problem #5 \nModel Optimizer Accuracy vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Training Accuracy')
leg = ax.legend();
```

```
##############################################


# Problem #6 - Final push for a high AUC score.
act = tf.nn.tanh
kern = 'VarianceScaling'
np.random.seed(42)
# Model #3 Hidden Layers-6
# Number of neurons;  50,50,100,100,80,80
# Dropout % 0.20
model6 = tf.keras.models.Sequential([
  tf.keras.layers.Dense(50, input_shape=(28,), activation=act,kernel_initializer=kern),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(50, activation=act),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(1, activation=act)
])
model6.compile(optimizer='RMSprop',
          loss='binary_crossentropy',
          metrics=['accuracy'])

model6.fit(x_train, y_train, epochs=100,batch_size=300)

model6.evaluate(x_test, y_test)
   #500000/500000 [==============================] - 20s 41us/sample - loss: 0.5139 - acc: 0.7469
   #Out[176]: [0.5139309719181061, 0.746896]

roc_auc_score(y_test,model6.predict(x_test))
   #roc_auc_score(y_test,model6.predict(x_test))
   #Out[178]: 0.8274673123171997


model6.summary()
   # _________________________________________________________________
   #Layer (type)              Output Shape            Param #
   #=================================================================
   #dense_105 (Dense)         (None, 50)              1450
   #_________________________________________________________________
   #dense_106 (Dense)         (None, 50)              2550
```

```
    #_________________________________________________________________
    #dropout_78 (Dropout)        (None, 50)              0
    #_________________________________________________________________
    #dense_107 (Dense)           (None, 50)              2550
    #_________________________________________________________________
    #dropout_79 (Dropout)        (None, 50)              0
    #_________________________________________________________________
    #dense_108 (Dense)           (None, 50)              2550
    #_________________________________________________________________
    #dropout_80 (Dropout)        (None, 50)              0
    #_________________________________________________________________
    #dense_109 (Dense)           (None, 50)              2550
    #_________________________________________________________________
    #dropout_81 (Dropout)        (None, 50)              0
    #_________________________________________________________________
    #dense_110 (Dense)           (None, 50)              2550
    #_________________________________________________________________
    #dropout_82 (Dropout)        (None, 50)              0
    #_________________________________________________________________
    #dense_111 (Dense)           (None, 1)               51
    #=================================================================
    #Total params: 14,251
    #Trainable params: 14,251
    #Non-trainable params: 0
    #_________________________________________________________________


model6_ultimate = model6.history.history

model6.save('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Models/my_model6_ultimate.h5')
Prob6_ultimate = pd.DataFrame.from_dict(model6_ultimate)
Prob6_ultimate.columns = ['loss','acc']
Prob6_ultimate.to_csv('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Prob6_DF.csv')

########################################
# Problem #6 graphics

Prob6_DF_plot=pd.read_csv("C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Prob6_DF.csv")

fig, ax = plt.subplots()
ax.plot(Prob6_DF_plot.acc, '-b', label='acc')
ax.plot(Prob6_DF_plot.loss, '-g', label='loss')
plt.title('Problem #6 \nOptimized Model Accuracy & Loss vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Training Accuracy')
leg = ax.legend();

#########################################


##### Example - Retrieving saved model #####
new_model =
tf.keras.models.load_model('C:/Users/Prodigy/Documents/GitRepositories/MSDS_7333_QTW/CaseStudy_12/Mod
els/my_model3_2_2_relu_25epoch.h5')
new_model_history = new_model.history.history
new_model.summary()
loss, acc = new_model.evaluate(x_test, y_test)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))
############################################
```