CS671 Fall 2017 - Deep Learning for Hedge Fund Stock investments

Arshpreet Singh Pabla
MSCS Dept, Rutgers University
Piscataway, NJ, USA
Email: asp229@rutgers.edu

***Abstract*—This project presents a deep learning platform built using artificial neural networks (ANNs) that would allow trading hedge funds to analyze large patterns of stock market data and detect possible permutations so as to make an informed decision on investments. This would allow to increase returns and reduce risk of bearing a loss.**

I. Introduction

A Hedge Fund is an investment, possibly in various types of securities like stocks, bonds, commodities and real estate, that aim to earn active return for their investors. In this project we consider investments in stocks only. There are multiple risks involved with a hedge fund investment, such as:

- hedge funds are exposed to potentially huge losses due to concentrated investment strategy
- hedge funds lock up the investors money for a couple of years
- the minimum initial investment required in a typical hedge fund is very high
- due to market risk, inflation risk and liquidity risk, stock market is very volatile to invest such huge amounts of money

Hence, it is a big catastrophe if the investment goes wrong. To mitigate this, we have developed a predictive model that uses deep learning to capture the influence of various factors on the stock prices and make an informed decision if the hedge fund should invest in that stock or not. Through the use of in-memory Big data processing of Apache Spark and adaptive learning, self-organization of a neural network we will learn the patterns embedded in our financial data and extrapolate them in an intelligent manner.

II. Data and preprocessing

Stock Market data is available on many free data sources like Yahoo!Finance, however that is only a very tiny fraction of the actual data generated through stocks. A high quality stock market data would include the factors that affect the price of a stock. However, explicit publication of such data is not permitted. Hence, for this project we use a dataset downloaded from NUMERAI, with 500,000 tuples and a set of 50 features. Each feature is responsible for affecting the value of a stock, directly or indirectly. There is a target column with binary values- 0 or 1, indicating if a user should invest in that particular stock or not. The stock name and the feature set are generated using structure-preserving encryption schemes such that we are blind to the raw data yet the structure of the dataset is preserved so as to allow the deep learning algorithms to process it efficiently. The training and validation dataset has the following format-

id, era, datatype, feature1...feature50, target

- id- encrypted name of the stock we are dealing with
- era- an identifier to help give insights to when an observation occurred
- datatype- [train, validation, test, live]
- feature1...feature50- set of relevant features
- target- the binary class: 0 or 1

III. Data Exploration

Each feature in the dataset corresponds to a normalized value of a factor that directly or indirectly influences the performance of the stock. Fig. 1 reflects how the average values of each feature for a particular ”target” value stands against each other. The feature average values are grouped by the ”target” values. Here we have taken just 25 features from a total of 50, so that the graph is discernible. The manner in which a feature affects the ”target” value is evident from Fig. 2. These features are plotted as a line chart against the ”target” values.

IV. Neural Network for Pattern Classification

A. *General overview of Neural Networks*

Neural Networks are flexible, nonparametric modeling tools. They can perform any complex function mapping with a good accuracy. An artificial Neural Network (ANN) is typically composed of several layers of many computing elements called neurons. Each neuron receives an input signal from other neurons or external inputs and after processing the signal locally through a transfer function, it outputs a transformed signal to other neurons or final result. As in any statistical model, the parameters (weights) of neural network model need to be estimated before the network can be used for prediction purposes. The process of determining these weights is called training. The training process aims at changing the weights so as to minimize the error between the observed and the predicted outcomes, traditionally given by the sum of their squared differences across all observations (patters) scrutinized in one iteration (epoch) of training. During training, the calculated error is backpropagated to the network and the weights are adjusted accordingly so as to improve the network prediction.

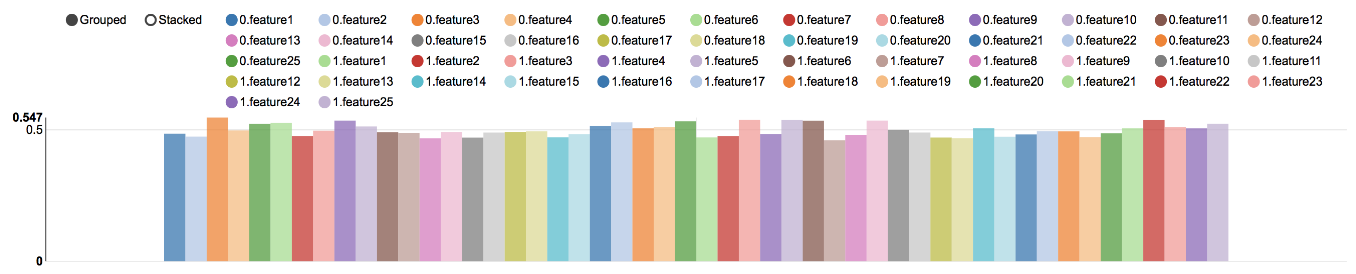

Fig. 1. Bar Chart of the average feature values

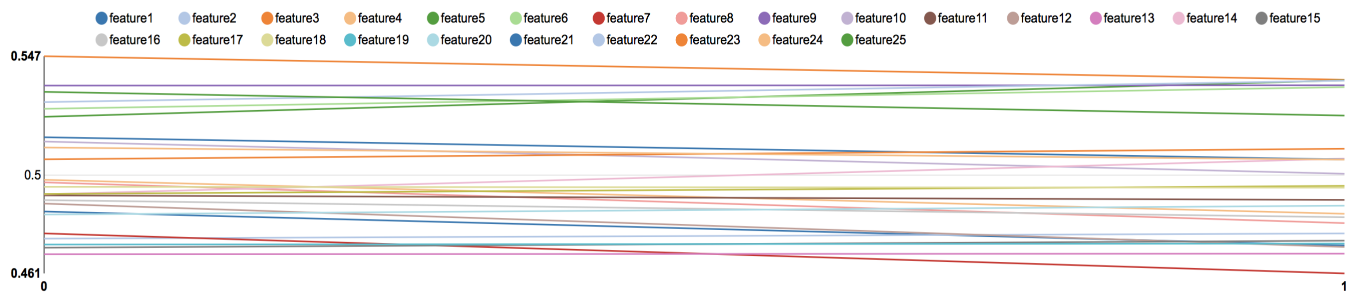

Fig. 2. Variation of features with the target value

The backpropagation algorithm is an iterative method based on gradient descent on the error surface reaching the minimal error possible. The weights are continually modified as a function of the change in the error, and the amount of weight change is determined by a learning parameter. The network training is a supervised learning method in which the desired or target response of the network for each input pattern is always known a-priori.

B. The Multilayer Perceptron Model

In Multilayer Perceptron (MLP) Neural Network, all neurons and layers are arranged in a feed-forward manner. The first layer is called the input layer where external information is received. The last layer is called the output layer where the network produces the model solution. In between, there are one or more hidden layers which are critical to the model's ability to identify the complex patterns in the data. An example of an MLP with an input layer of M neurons, one hidden layer of N neurons and an output layer with K neurons is shown in Fig. 3. There is one neuron in the input layer for each feature variable.

1) Input Layer: A vector of predictor variable values is presented to the input layer. At the input layer, the values are distributed to each of the neurons in the hidden layer. In addition to the predictor variables, there is a constant input called the bias that is fed to each of the hidden layers. The bias is multiplied by a weight and added to the sum going into the neuron.

2) Hidden Layer: At the hidden layer, the value from each input neuron is multiplied by a weight, and the resulting weighted values are added together producing a combined value. The weighted sum is fed into a transfer function. The outputs from the hidden layer are distributed to the output layer.

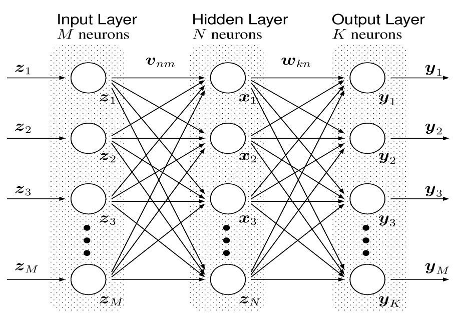

Fig. 3. A Multilayer Perceptron Neural Network Model

3) Output Layer: On arrival at the output layer, the value from each hidden layer neuron is multiplied by a weight, and the resulting weighted values are added together producing a combined value. Thereafter, the weighted sum is fed into a transfer function, which outputs its own value.

C. Training Multilayer Perceptron Networks

The goal of the training process is to find the set of weight values that will cause the output from the neural network to match the actual target values as closely as possible. There are several issues involved in designing and training a MLP network such as:

- Selecting the number of hidden layers to use in the network.
- Deciding how many neurons to use in each hidden layer.

- Finding a globally optimal solution that avoids local minima.
- Converging to a optimal solution in a reasonable period of time
- Validating the neural network to test for overfitting

V. Implementation

1) Measures and Sample: This model utilizes a total of 50 features as predictor variables. The predictor variables are certain features like inflation levels, interest rates, etc. These features influence the stock market in some manner, directly or indirectly. All columns values have been normalized such that any instance of a feature is between (0,1). The Target is binary coded; a 1 represents that the stock price is most probable to rise in the future and hence is worth investing in, while a 0 represents that no investment should be made on the stock as it would eventually lead to a loss. Prior to training the neural network, the data columns were portioned into either the input features or desired output label. The data rows were segmented into two groups; training and testing, in the ratio 60:40. The model is trained on the 60% of the data based on the output labels known. It's preformance is then tested on the remaining 40% of the dataset.

2) Design of the Neural Network model: An Artificial Neural Network is characterized by its architecture. The architecture refers to the number of layers, number of neurons in each layer and the degree of connections. The network implemented in this project has three hidden layers; hidden layer 1 has 64 neurons, hidden layer 2 has 32 neurons and hidden layer 3 has 16 neurons. The number of input neurons is the number of predictor features, which is 50 in our case. The output layer has just a single output neuron which returns the class of the input tuple, which can be either 0 or 1. The layers are fully connected with each other, which means that from each neuron of one layer, there is an outgoing connection to every neuron of the next layer. Using Tensorboard, Fig. 4 is generated which shows the network graph of the ANN model designed for this project.

3) Training and Loss: Input weights for each of the layers are initialized as random values from the normal distribution of the shape: [input-layer-dimension, output-layer-dimension]. The loss function is defined by the Mean Squared error which measures the average amount that the model's predictions vary from the correct target values. The cost is higher when the model is performing poorly on the training set. The objective of the learning algorithm then is to find the weight values which give the minimum possible cost. This is achieved by using the Gradient descent optimizer which is a first-order optimization algorithm that attempts to find a local minimum of a function by taking steps proportional to the negative of the gradient of the function at the current point. Fig. 5 shows the training optimization component of our model generated via Tensorboard. Hence, iteratively the value of the loss function is reduced which can be seen in Fig. 6.

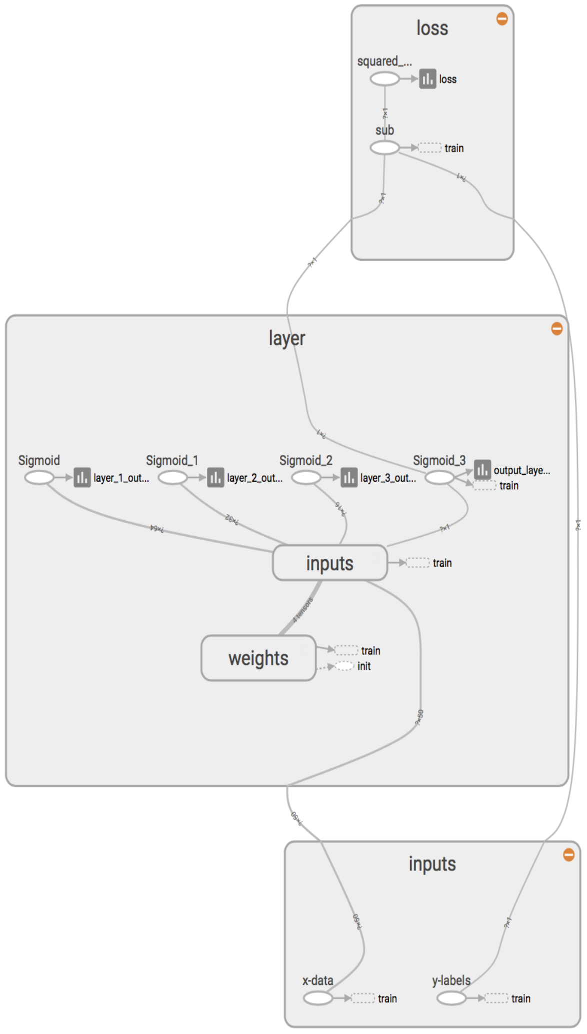

Fig. 4. Network Graph

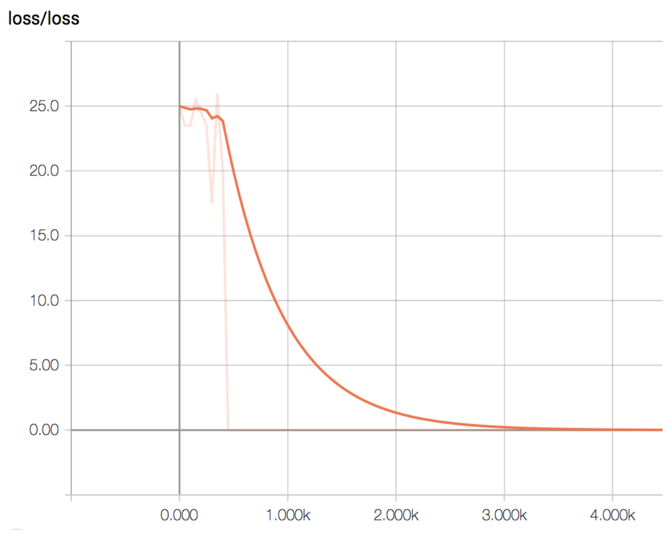

Fig. 5. Loss function

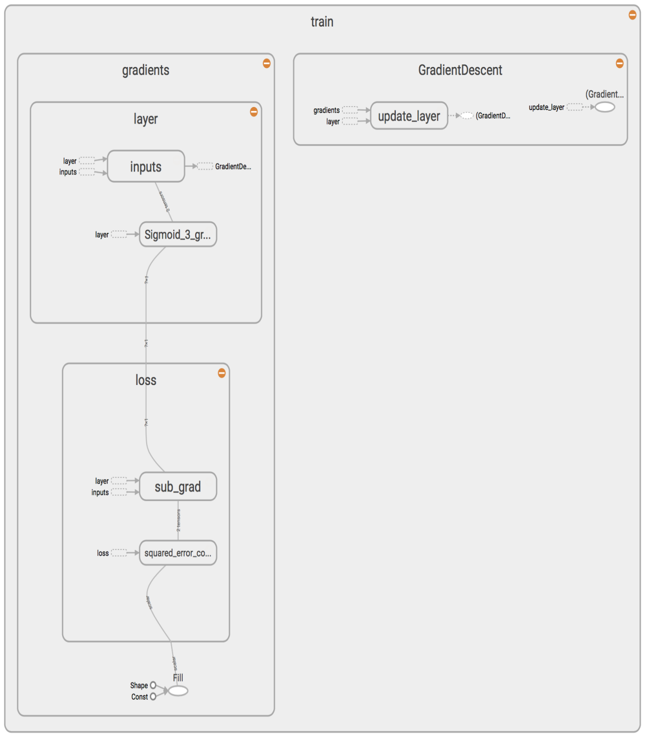

Fig. 6. Training Optimization component

VI. Results

Once the weights are randomly initialized, the algorithm works to converge them to an optimum value using Gradient descent such that the loss function can be minimized. The distribution of 'weights' across each layer can be seen in Fig. 7. The darker region shows that most of the weight values were in that range, while the lighter shade depicts that there were few weights assigned within that range. The maximum range of values for hidden layer 1 is (3.6,-3.0), for hidden layer 2 is (4.2,-5.0) and for hidden layer 3 is (3.0,-3.7). Each neuron of every layer receives input values from the previous layer neurons. Then it calculates the weighted sum of its inputs and then decides if it should fire or not based on the Sigmoid activation function. Sigmoid activation function is given as $A = \frac{1}{1+e^{-x}}$. It allows the neurons to respond to changes in any weight in the inputs in a smooth manner instead of a drastic manner like a step function. Also, Sigmoid function does not have a jerk on its curve. It is smooth and has a very nice simple derivative that is differentiable everywhere on the curve. Hence, with this function we introduce non-linearity into our neural network which ultimately allows for non-linear decision boundary. Eventually, with every output of a hidden layer, the classification improves step-by-step. This learning procedure is evident from Fig. 8. After each layer, the distinction between the two classes- 0 and 1, becomes more and more evident and hence the model's performance improves. Finally, the output layer produces the classified elements, the left and right shaded areas represent those set of

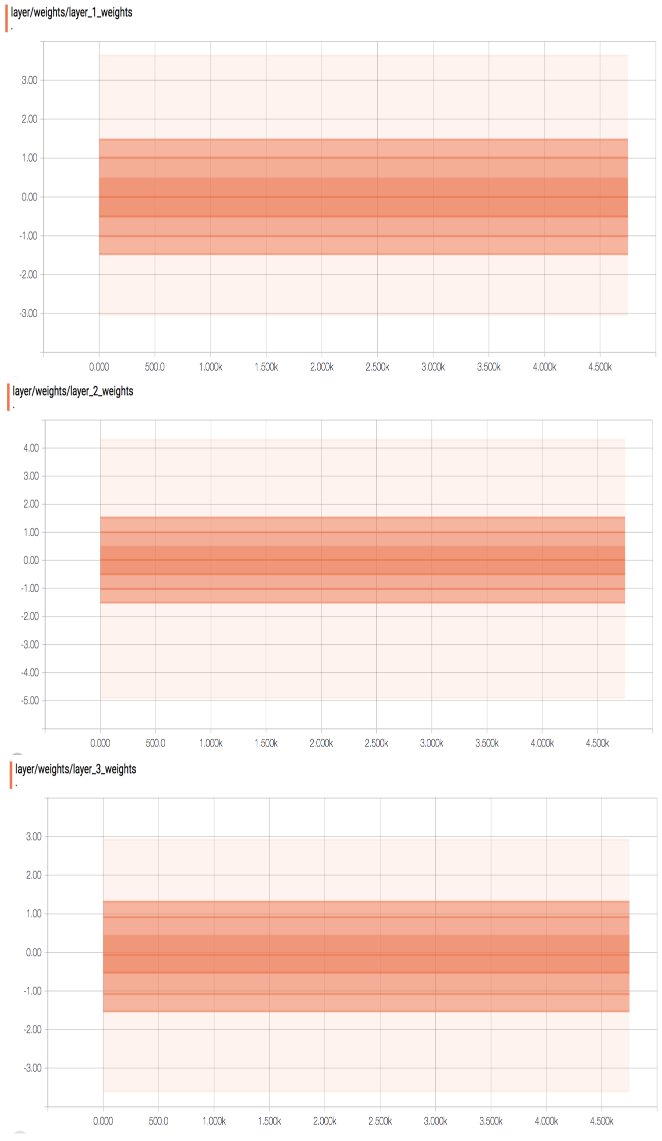

Fig. 7. Weight Histogram

elements. After training and testing the model, it returned an accuracy of 75% on the test dataset.

VII. Future Work

As mentioned in the previous section, the accuracy achieved by this model is 75%, which may lead to investments that may not return great profits in the future. Hence, it is crucial that the model is further improved. We can experiment with activation function used for the layers. Rectified Linear unit can converge faster than other activation functions. In addition to that, the number of neurons in each layer can be altered. However, there is no rule of thumb in choosing the number of neurons. If an inadequate number of neurons are used, the network will be unable to model complex data and the resulting fit will be poor. On the other hand, if too many neurons are used, the training time may become excessively long and the network may over-fit the data too. Hence, it is something that needs to

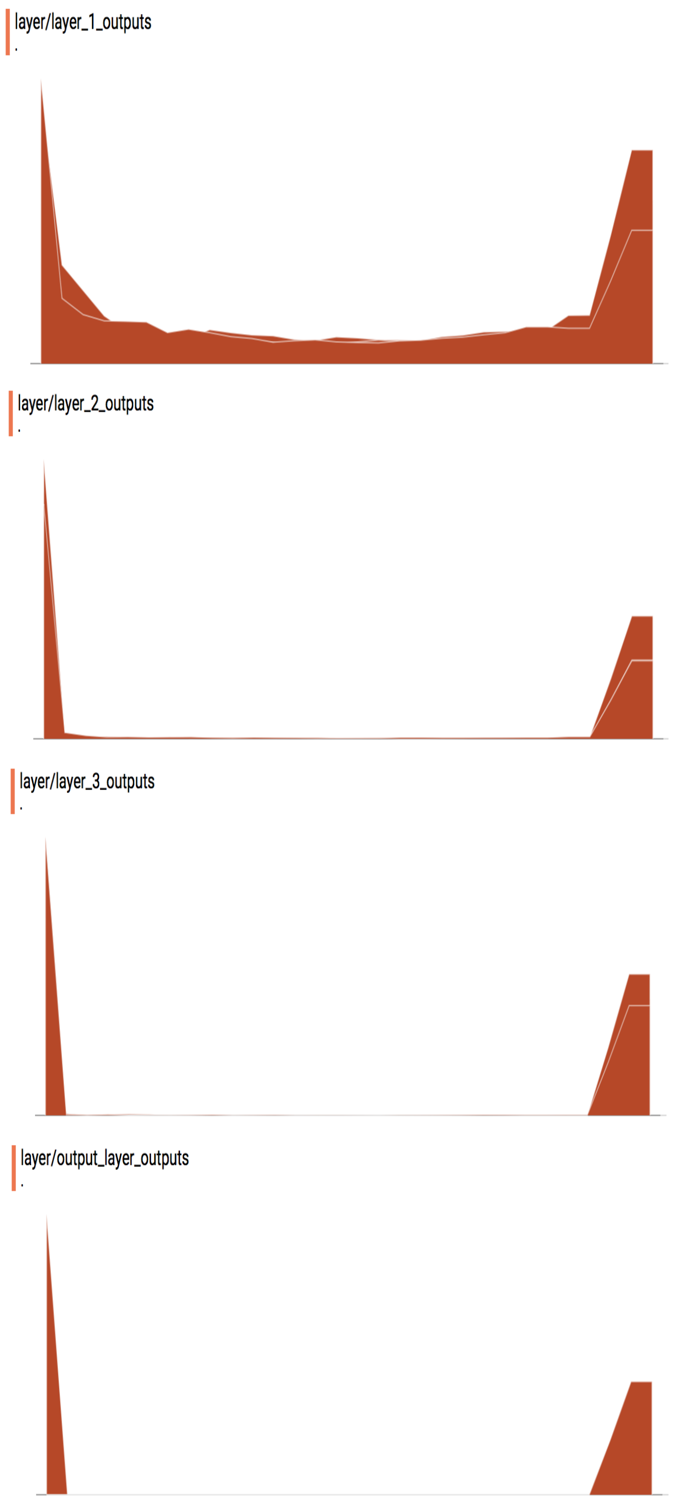

Fig. 8. Output Histogram

be tried and tested multiple times. The random values of the initial synaptic weights may cause the model perform poorly. Hence, we should try with different random seed to generate different random weights and then choose the seed number which works well for our problem. In the future, we can also implement a meta-neural network by using stacking ensemble method. In this, we will train multiple learners, each uses its own subset of the dataset. Then a combiner algorithm will be trained to make the ultimate predictions using the predictions of the other constituent algorithms. This way we can get the best out of each of the algorithms while trying to avoid their quirks.

References

[1] *https://www.barclayhedge.com/research/educational-articles/hedge-fund-strategy-definition/what-is-a-hedge-fund.html*
[2] *https://finance.yahoo.com/news/must-know-3-key-risks-210152682.html*
[3] *https://medium.com/numerai/encrypted-data-for-efficient-markets-fffbe9743ba8*
[4] *https://numer.ai/help*
[5] *http://www.investopedia.com/terms/h/hedgefund.asp*
[6] *http://mccormickml.com/2014/03/04/gradient-descent-derivation/*
[7] *http://research.ijcaonline.org/volume35/number5/pxc3976106.pdf*
[8] *https://www.quora.com/What-is-an-intuitive-explanation-of-gradient-descent*
[9] *https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f*
[10] *http://deeplearning.net/tutorial/mlp.html*
[11] *https://machinelearningmastery.com/neural-networks-crash-course/*
[12] *http://ischlag.github.io/2016/06/04/how-to-use-tensorboard/*
[13] *https://www.datacamp.com/community/tutorials/tensorflow-tutorial*
[14] *https://d4datascience.wordpress.com/2016/09/29/fbf/*