Predicting Consumer Price Index Performance: An Ensemble Neural Network Approach

Nicholas Dorogy

Washington and Lee University

Lexington, VA

ABSTRACT

I investigate the utility of an ensemble of recurrent neural networks used to predict fluctuations of the Consumer Price Index (CPI). Twenty technical analysis indicators are used as inputs to the network. An individual networks attempt to model a single scenario of the CPI's behavior. Models from each scenario were then averaged to produce a unique prediction. My results show that an average ensemble model consistently outperforms the prediction made by a single network.

1. INTRODUCTION

The automotive industry is a critical indicator of economic performance. In the United States alone, there are 1.7 million jobs directly created by the industry.[1] These jobs are estimated to contribute to a further 8 million total private sector employment positions, $500 billion in annual compensation, and $70 billion in personal tax revenues.[1] The employment multiplier for the industry is measured to be 4: for every job directly created by the industry, four more employment opportunities are opened. Moreover, the automotive industry has historically accounted for between 3% and 3.5% of the overall Gross Domestic Product (GDP) in the United States.[1] It is estimated that approximately 4.5% of all U.S. jobs are supported by the auto industry's presence in the United States economy.[1] With such an intricate relationship to American income, the industry is vital to both the U.S. and international economies. Thus, analyzing trends in the auto industry is a pragmatic

mechanism to predict economic progression and overall health. For this reason, automotive-related economic metrics can reliably be used as significant influencers when estimating the economy's performance.

However, producing predictions of stock performance is notoriously difficult. There is a tremendous number of inputs that influence a stock's performance. In addition, the lack of data ensures inaccuracy in results. Lag time between the occurrence of events and their resulting impact on the market further complicates economic forecasts. A potential solution to overcome these issues can be found through the use of **neural networks**.

Neural networks function almost identically to their namesake (the neuron) which serves as the fundamental tool driving activity in the brain. In this model, neurons are interconnected in a large array. When a certain activation value, or threshold, is achieved, the neuron will discharge an electric signal.[2] However, this analogy can be extended beyond a biological resemblance. In quantum mechanics, the potential function describes the behavior of a bound particle.

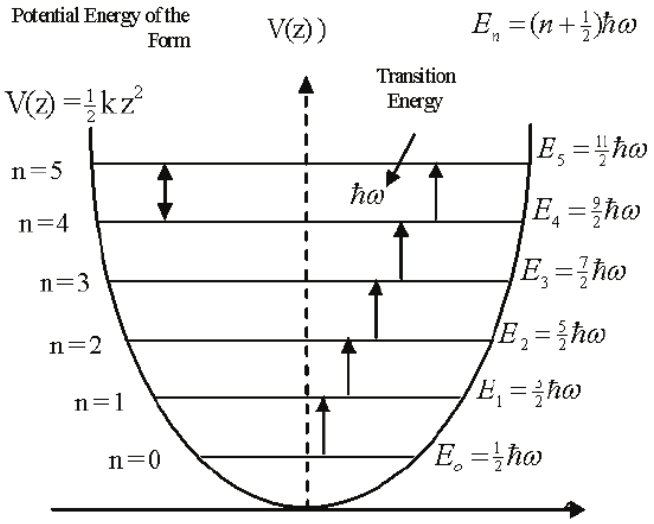

Figure 1. The quantum harmonic oscillator potential function[3]

In this case, the function (like a square function, harmonic function, or delta function) describes whether or not a particle is bound. When the particle's energy overcomes the potential energy, it transitions from a bound state to a unbound state: a behavior that

is quite similar to the activation threshold in a neural network. Even within the bound states of the system, the quantum wave function is restricted to discrete energies ensuring that transitions in energy levels only occur when a certain transition energy threshold is satisfied.

Furthermore, neural networks mature similarly to Ising models. In this scenario, each spin in a lattice is dictated by the directions of the neighboring spins. That is, as the model evolves over time, spins are flipped when the influences of their neighbors reach a certain threshold. Because neighboring spins with the same spin have a lower energy than those with opposite spin directions, the system can migrate towards an energy minimum.

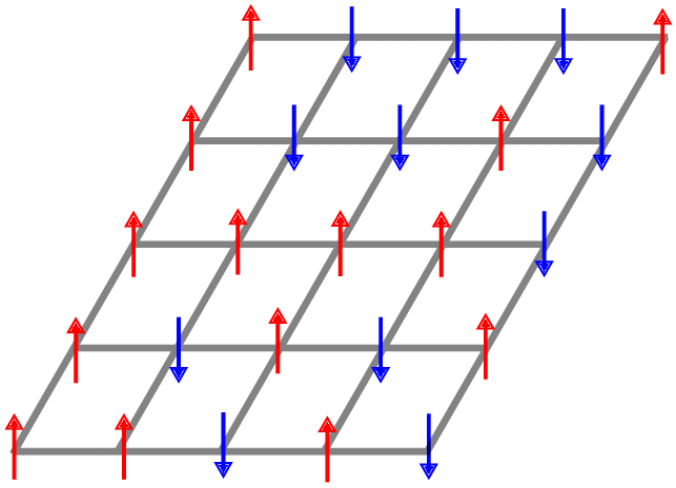

Figure 2. Ising model spin lattice[4]

For a more a specific application of the relationship between Ising models and neural networks, consider spin glasses: magnetic systems with randomly distributed ferromagnetic and anti ferromagnetic interactions.[5] The evolution of the state of this system behaves like a neural network. When a certain threshold value is reached, a node (or neuron in the biological example) will act on a neighboring node influencing its state. The goal of the output system is to reach a stable equilibrium which corresponds to an energy minimum. Again, this parallels the optimizer's loss function deployed in neural networks where the function, often gradient descent, seeks to ascertain the minimum of the system. In both scenarios, an input will reach a certain value and subsequently alter the state of the following

node. This process will continue through many "layers" or connections of nodes until it reaches an output state in which the influences of previous layers are compiled into a single output for the stable system.

Such networks are also rooted in the field of statistical physics when considering ensemble techniques. Ensemble averaging simply requires calculating averages over all possible microstates or configurations of a system. In classical statistical physics, ensembles are generally employed to calculate average quantities in order to replace the time average. This is advantageous when trying to understand the properties of a complex system without knowing all of its specific characteristics. In this scenario, one specific system has an observed macrostate that can be represented by an very large number of microstates. As this system evolves with time, the macrostate representing the system will change, where the changes in the microstates represent the average behavior of the system (the macrostate). However, instead of looking at the evolution of the microstates in time, we can consider a particular instant and average over the total number of microstates possible. Taking the average of these microstates is equivalent to the average behavior of the system over time. We can then apply this predictive technique to create models. In this methodology, multiple predictive sub-models, each of which generates a unique prediction, contributes equally to a combined forecast. That is, this model is run repeatedly to produce several different outputs that are then averaged into a single result. This reduces the variance in the performance in the model, while increasing the confidence in a particular final result; a distinctly useful return when considering stock evaluation. Moreover, it lessens the network's dependence on highly specific data required to train the model.

The deeply intertwined relationship between physics and economics has given rise to a notable sub-discipline: econophysics. This field seeks to marry the analytical tools used by physicists with the market forecasts performed by economists. My network specifically employed probabilistic and statistical methods adapted from statistical physics to perform a prediction of the CPI's behavior.

2. LITERATURE REVIEW

Shadra and Patil proved the validity of neural networks used for time series forecasting.[6] The authors tested annual, quarterly, and monthly data using neural network and Box-Jenkins forecasting models, a well established model capable of handling time series evolution. Their results showed that neural networks performed as well as the Box-Jenkins forecasting model. They also found that neural networks provided more accurate forecasting with irregular time series, thus offering a useful alternative to typical time series forecasting. Kara, Boyacioglu, and Baykan compare artificial neural network (ANN) and support vector machines (SVM) in predicting the daily movement of the Istanbul Stock Exchange.[7] They selected ten key drivers of stock performance on the market to be used as inputs for the models. They found that ANN models significantly outperformed SVM models. The average prediction performance of the ANN model was 75.74%, while the SVM model scored 71.52%. They showed that neural networks can be used for accurate daily stock trading. Panda and Narasimhan also prove the utility of stock predictions using neural networks for daily trading on the Bombay Stock Exchange Sensitive Index returns.[8] Using six measurements of stock performance as their inputs, they compared the accuracy of ANN models against random walk and linear auto-regressive models. They found that artificial neural networks outperform or are comparable to the other models used in their study. They further conclude that neural networks can successfully incorporate non-linearities contained in stock returns, and thus produce improved return predictions. Borovkova and Tsiamas show that an ensemble of long-short-term memory neural networks can be used for intraday stock predictions when introducing a large number of technical analysis indicators.[9] This system creates daily predictions and weights the individual models proportionally based on their continued performance. They evaluate their model on data from several US large-cap stocks and benchmark the models performance against lasso and ridge logistic classifiers. They found that their ensemble method using a variety of inputs outperforms the traditional benchmarks' performances. Perrone and Cooper show that a ensemble method outperforms any other estimator in mean square accuracy score.[10] Using optical character recognition tasks, they demonstrate that an ensemble significantly improves neural network performance even for

real-world problems because it incorporates all networks of a population, avoids over-fitting, and overcomes pitfalls present in a single network by using minimum values to produce improved estimates. Wang, Xu, Huang, and Yang also use the concept of an ensemble of networks to identify fraudulent stock trading on the China Securities Regulatory Commission.[11] Using a variety of parameters contained within trading records they found that the ensemble technique outperformed current methodologies by 29.8% in terms of area under curve value. Using the S&P 500 Index as a test case, ensemble recurrent neural networks are also used to predict future stock movement by Li and Pan.[12] They achieved a 57.55% reduction in mean square error when comparing their ensemble of networks against the best existing prediction model using the same dataset. Furthermore, they increased precision rate by 40% and movement direction accuracy by 33.4% confirming the ensemble method's applicability in enhancing stock price trends.

3. METHODOLOGY

I designed the neural network to draw upon twenty distinct variables to serve as the input data for the network. After the sample set undergoes pre-processing, I feed the data to the neural network so that it may generate the desired output: a prediction of the behavior of the Consumer Price Index. I then stored this prediction and ran the neural network repeatedly producing numerous new, unique outputs each of which is recorded and saved. I then averaged these outputs to produce the final prediction and further compared the resulting error against the error of a classic neural network and the error of the linear regression performed on the entire dependent training dataset.

3.1 Selection of Input Parameters

To train this neural network, I utilized the Consumer Price Index as the dependent variable. That is, the Consumer Price Index serves as the reference on which the network will compare the value it produces in order to assess and improve its accuracy. The output of the network is a forecast of the dependent variables behavior in the future. Although CARZ ETF would have served as a more appropriate dependent variable, limited data as a consequence of its inception in 2011 necessitated that the Consumer Price Index serve as a proxy.

The independent variables (totalling twenty parameters) served as the input data for the network. Figure 2 lists these inputs.

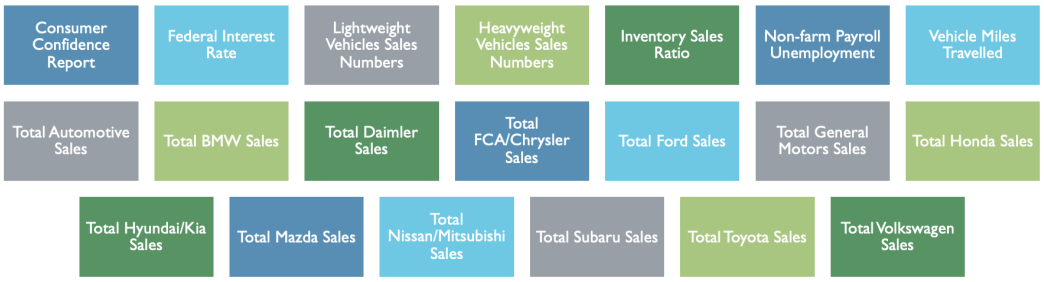

Figure 3. The 20 input parameters used by the neural network

The parameters were chosen for their significant influence on the automotive industry's success, and therefore the greater economy as a whole. Thus, it is reasonable to assume that their behavior is indicative of the performance of the Consumer Price Index.

The amount of data fed into the network was limited by its availability. Although certain inputs contained publicly available data from the early 1900's, others were restricted to the mid 2000's. Each of the selected inputs is reported on a monthly basis from January 1, 2005 until August 1, 2021. In turn, I provided 4,200 data points to the network for training and testing.

3.2 Data Acquisition and Cleaning

Data was imported from a variety of different sources as shown in Figure 3. In order to use

Figure 4. Data sources for the input parameters and CPI

this data for the neural network, I had to standardize and rescale every value. Standardization ensures that all inputs can be treated equally. Without standardization, the networks prediction is meaningless since data would be reported in different units where a particular category could dominate because it is inherently larger. For example, the raw volume of cars sold and the raw interest rate cannot be treated as equal inputs. Rescaling the inputs forces them between a value of 0 and 1 and therefore removes the ability of outlying categories to over exert their influence on the network. Moreover, it adjusts the training parameters so that they are not set to an initial maximum of 1 or minimum of 0. Had input parameters been set at a maximum or minimum, the output would not be able to surpass these initial values and would thus be restricted to an artificially small range. The code provided in Appendix 7.2, written in R, contains the process by which the data was standardized and rescaled.

3.3 Neural Network Creation

I used Python to construct the network because the Keras and TensorFlow libraries provide the framework for the simple construction of a network. Keras offers a variety of built-in functions that ease the process of tuning a *single* network, whereas TensorFlow is the base library needed to run the network. Thus, using Keras, I first constructed one particular neural network to perform one prediction. This prediction was stored and the network was called again to perform another calculation for the specified number of attempts. Once the network had created the desired number of predictions, the results were averaged to produce a final output. The code in Appendix 7.3 was used to generate the ensemble prediction.

It is important to note that because the Consumer Price Index displays linear growth, the model must be classified as a regression problem. For this reason, the hyperparameters are tuned to best fit a regression analysis.

3.4 Tuning the Network & Algorithm Selection

To train the network, there were a variety of different hyperparameters that had to be manually set. These included the optimizer or optimization function, activation function, loss function, metric, number of nodes, number of epochs, batch size, and test size.

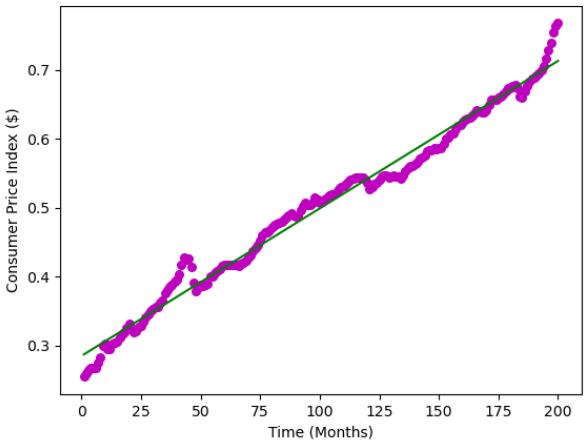

Figure 5. Regression of the Consumer Price Index data

3.4.1 The Optimizer

Perhaps the most important of these is the optimizer. The optimizer is used to alter the attributes of a network (like the weights of the inputs or the learning rate) in order to reduce the loss. In the same way that Sompolinsky and Hopfield use a loss function to achieve an energy minimum and thus a stable equilibrium, the optimizer is tasked with finding a minimum error.[2,5] Although Stochastic gradient descent is traditionally used, I chose to employ the adaptive moment estimation (Adam, for short) because of its improved ability to train efficiently, accurately, and quickly.[13] Adam is designed to slow the descent of the error correction (or the loss) in order to avoid missing the minimum. This is accomplished by combining features of two other extensions of stochastic gradient descent: Adaptive gradient algorithm (AdaGrad) and Root Mean squared propagation (RMSProp). More specifically, Adam calculates an exponential moving average of the gradient and the squared gradient. The equation governing Adam's update rule is given by,

$$\theta_{t+1} = \theta_t - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{1}$$

Appendix 7.1 provides greater detail on Adam's functionality.

3.4.2 Activation Function

The activation function is the equation that determines how the weighted sum of the input is to be transformed into an output. This is equivalent to Sompolisnky's "action potential" or Hopfield's spin-glass activation threshold which determine if a neuron will fire or a spin will flip based on whether or not the potential has exceeded a particular threshold.[2,5] For this network, the activation function was selected to be the Sigmoid function. Whereas Somplinsky uses a membrane potential that is the linear sum of potentials induced by the activity of neighboring neurons, I exploited the Sigmoid function because its non-linearity permits the model to learn a non-linear solution. This function also pairs particularly well with data that was rescaled between values of 0 and 1 during normalization; the same process that was applied to this dataset. The Sigmoid function, given by,

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

returns values in the range 0 to 1. The larger or more positive the input is, the closer the output value will be to 1. The smaller or more negative the input is, the closer the output will be to 0.

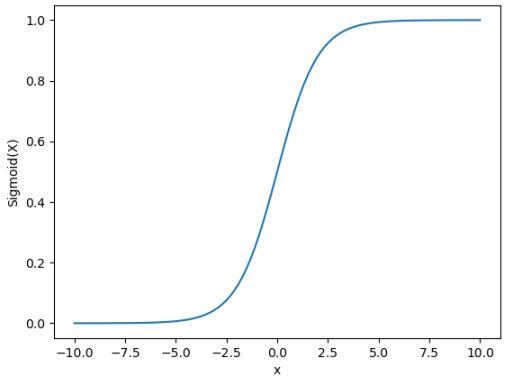

Figure 6. The Sigmoid function

3.4.3 Loss Function

The loss function is designed to abstract all aspects of the model into a single number so that improvements in that number are a sign of a better model. More simply, the loss function

is the method employed during network training in which the error between the network's prediction and the actual value is calculated. For problems where a real-value quantity is to be predicted (regression problems) through a single output node, mean-squared-error (MSE) or the average of the squared differences between the predicted and actual values, is the most common choice. MSE was also the loss function I chose to utilize in my program. The equation for MSE is,

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{3}$$

where $(Y_i - \hat{Y}_i)^2$ is the difference between the actual and predicted values squared and N is the sample size. More simply, MSE is the average squared difference between the estimated values and the actual value.

3.4.4 Metric

A networks metric is a loss function used to judge the performance of the model. They behave quite similarly to loss functions, except that the results of the metric are not used when training the model. The metric is a function that returns a scalar measure of the "fitness" of the model to the available data. Because this network is best classified as a regression problem, I chose root mean square error (RMSE) as the metric to be displayed. It is calculated by,

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x_i})^2}{N}} \tag{4}$$

where, once again, $(x_i - \hat{x_i})^2$ is the difference between the actual and predicted values squared, while N is the sample size.

3.4.5 Number of Nodes

A general rule for determining the number of hidden nodes in the network is that,

$$N_h = \frac{N_s}{(\alpha \cdot (N_i + N_o))} \tag{5}$$

where N_i is the number of input neurons, N_o is the number of output neurons, N_s is the number of samples in the training dataset, and α is an arbitrary scaling factor between 2 and 10. Therefore, this network consisted of a single input node with 40 neurons, a single

output node with 40 neurons, and a single hidden layer of 26 nodes as dictated by above equation.

3.4.6 Number of Epochs

The number of epochs dictates the number of *complete* passes through the training dataset before an output is produced. For example, if the epoch hyperparameter is set to ten, each sample in the training dataset updates the internal model parameters ten times. Generally, a model's accuracy improves with more epochs, but will eventually plateau in accuracy. Thus, I set the number of epochs to be 50– a compromise between speed and performance.

3.4.7 Batch Size

The batch size determines the number of training samples the network utilizes before the model's internal parameters are updated. This network relied on stochastic gradient descent meaning the batch contains only one sample from the dataset. This means that I have chosen a batch size of 1, a surprising benefit of having a relatively small number of samples in the dataset.

3.4.8 Test Size

The test size simply regulates the amount of data devoted to testing (and therefore, training). For this network, I chose the common 80:20 split so that 80% of the samples are used for training the network, while only 20% is used to test and evaluate the performance of the network.

4. RESULTS

I found that an average ensemble of neural networks consistently outperformed a single neural network's prediction. I began by comparing a single network's error against the error of an average ensemble of two networks. I found that despite the addition of just one network, there was an appreciable difference in the error between the single and ensemble accuracy scores. I then continued to increase the number of networks being averaged while comparing their resulting differences in error. The accuracy, of each ensemble is listed in Table 1.

Overall Accuracy			
# Networks	Total Error	Mean Error	Std. Dev. Error
1	0.5168	0.013	0.014
2	0.4129	0.010	0.008
3	0.3799	0.009	0.007
4	0.3711	0.009	0.008
5	0.3561	0.009	0.008
10	0.3623	0.009	0.008
25	0.3674	0.009	0.008
50	0.3384	0.008	0.007
75	0.3687	0.009	0.008
100	0.3639	0.009	0.008
500	0.3571	0.009	0.007

Table 1. Accuracy of the output for various number of networks being averaged

Clearly, the ensemble approach consistently results in less error than a single network. However, averaging increasingly more networks returns exponentially decaying improvements in accuracy while requiring progressively longer computation times. The total error plateaus at 0.36 with an ensemble of only five networks. Thus, the number of networks used in the ensemble could be limited to five unique inputs. Moreover, a regression analysis was performed on the network and the resulting error was reported to be 2.4896– a value significantly higher than the error produced by a single or ensemble network.

5. DISCUSSION

Each time the network runs, a unique output is produced. Therefore, Table 1 compares the ensembles' accuracy against the accuracy of a single, random network. This is evidenced by the fact that the error of a single network changes in Figures 2-11. But, this also indicates the power of constructing such ensembles of neural networks. This method accounts for a

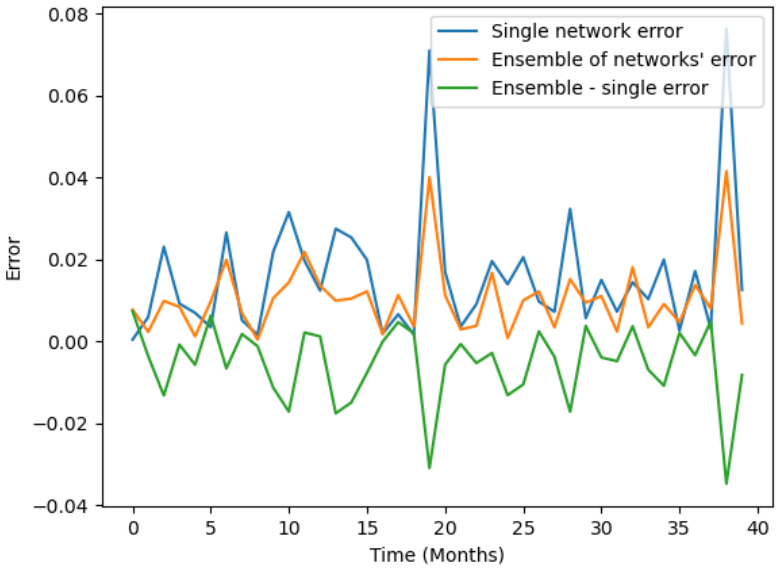

Figure 7. Performance results for an average ensemble of two networks

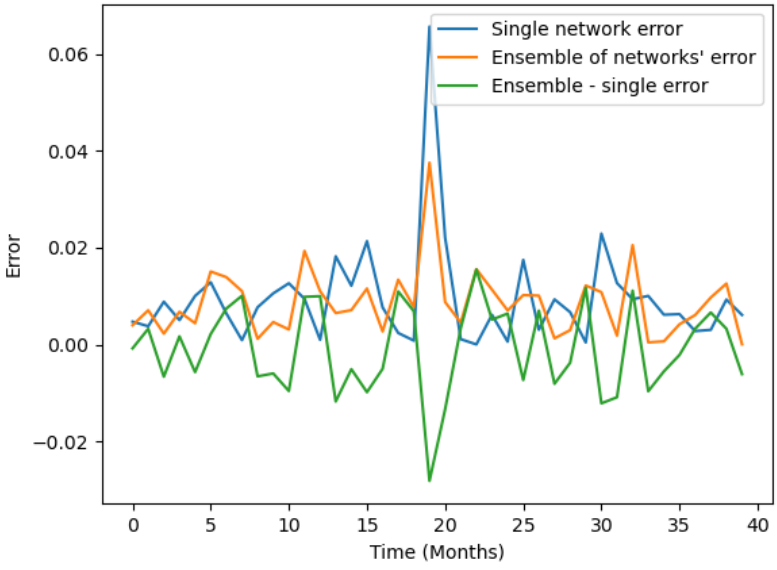

Figure 8. Performance results for an average ensemble of three networks

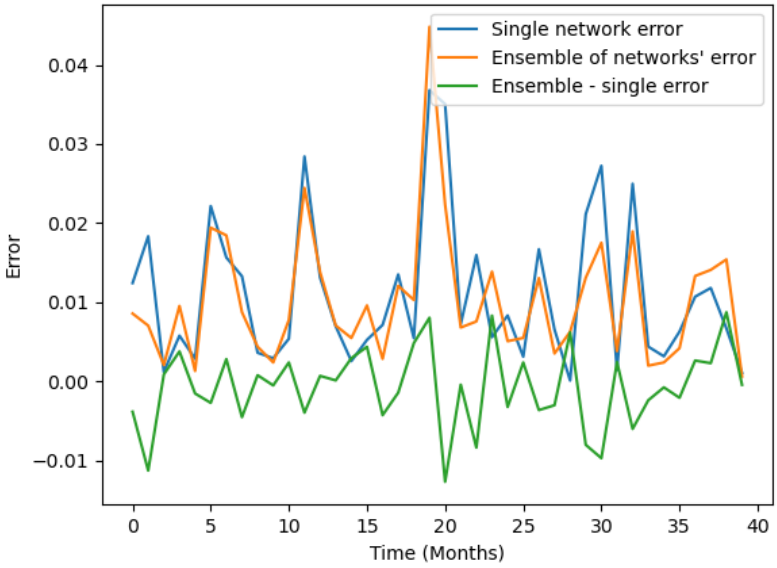

Figure 9. Performance results for an average ensemble of four networks

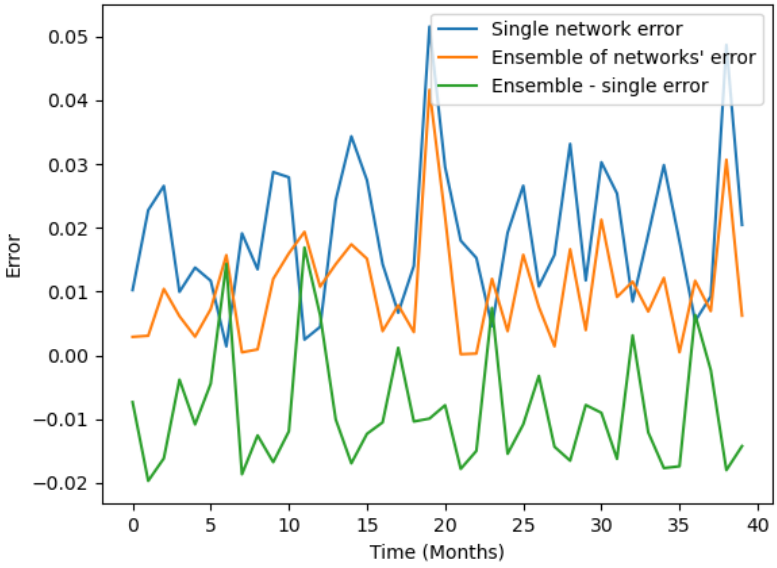

Figure 10. Performance results for an average ensemble of five networks

wide range of possible outputs and thus improves overall accuracy by diminishing the affect of outliers.

6. CONCLUSION

The first step to improving this model is to acquire a significantly larger dataset. Of course, more samples would allow for larger testing and training sets. An increased historical sample size would also ensure more variance in the data due to a larger variability of values for the input parameters of the network. Therefore, more strenuous testing of the ensembles prediction over time would occur. Although this network is restricted to monthly samples between January 1, 2005 and August 8, 2021, using samples from 1980 or earlier would more accurately assess the model's performance. Moreover, simply adding new input parameters could also improve the model's accuracy by incorporating unaccounted for influences on the Consumer Price Index's behavior.

Further improvements could be made in the type of ensemble used. For example, a weighted average ensemble could replace the simple average ensemble deployed in this investigation. The weighted average ensures that the predictions with the lowest total error contribute more to the final prediction than predictions with a higher error. This could increase the model's accuracy, but necessarily sacrifices the output's tolerance or its ability to predict sizable, acute changes.

A change in the independent input parameter could also improve the model's score. Replacing the Consumer Price Index with an automotive index would more accurately train the data used in this model. However, the first automotive index (CARZ ETF) was not launched until 2011, so its theoretical historical performance would need to be approximated.

Although this program is tailored to learn a regression problem, the general idea of an ensemble network can be applied to a broad number of problems. For any non-classifying network in which a single output prediction is made, an average ensemble network could be substitute in the place of a single network. This particular model is designed as recurrent neural network in order to handle time series data. Therefore, this general model can be used to improve any single network handling time series data.

7. APPENDIX

7.1 The Adam Optimizer

Adam's creators, Ba and Kingma describe Adam as combining the advantages of Adaptive Gradient Algorithm and Root Mean Square Propagation: two similar functions dedicated to evaluating stochastic gradient descent.[14] The Adaptive Gradient Algorithm (AdaGrad) maintains a per-parameter learning rate which improves performance for problems with sparse gradients. Root Mean Square Propagation (RMSProp) also maintains per-parameter learning rates that are altered depending on the average of recent magnitudes of the gradients of the weight. This is particularly useful for noisy problems. Adam combines the benefits of both AdaGrad and RMSProp. Instead of changing the parameter learning rates depending on the average first moment as is done in RMSProp, Adam uses the average of the second moments of the gradients. That is, the algorithm determines an exponential moving average of the gradient and the squared gradient, where the parameters β_2 and β_2 control the decay rates of the moving averages. The initial values of the moving averages and β_1 and β_2 close to 1.0 result in a bias of moment estimates towards zero. This bias is challenged by calculating the biased estimates before and subsequently calculating the bias-corrected estimates.

The decaying averages of past gradients (m_t) and past squared gradients (v_t) are,

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{6}$$

and

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{7}$$

Moreover, the biased first and second moment estimates are given by,

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{8}$$

and

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t} \tag{9}$$

Therefore, the update rule is then given by,

$$\theta_{t+1} = \theta_t - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v_t}} + \epsilon} \tag{10}$$

Ba and Kingma have chosen $\beta_1 = 0.90$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. η, the default learning rate, is generally assumed to be 0.001.

7.2 Standardization and Scaling Code

```
##All data scaled & standardized

#import statements
library(readr)
library(caret)

#read in the data, transform it to usable form, and removes certain
    values
myData <- read_csv("~/Desktop/school/GitHub/CapstoneCode-/Capstone/Non
    -standardized/Sales Numbers/BY MANUFACTURER/FCA-Chrysler/Chrysler_
    monthly_05-21.csv") #reading in raw data
transformData <- as.data.frame(as.vector(t(myData[-c(1)]))) #Removes
    the first column, transposes the data, vectorizes it, creates a
    data frame
transformData <- transformData[-c(202,203,204),] #removes rows

#reformatting, adding dates back, and fine tuning
dates <- seq(from=as.Date("2005-01-01"), to=as.Date("2021-09-01"), by=
    'month') #creating dates
newData <- transform(transformData, newcol=paste(dates, sep="_")) #
    combining dates and newData columns
newData <- as.data.frame(newData) #creating a data fram out of the 2
    columns
colnames(newData) <- c("Cars sold","Date") #setting column names
newData <- newData[c('Date','Cars sold')] #reordering columns by their
     name in the specified order

#standardize & rescale the data
```

```
standardDev <- sd(newData[,2]) #find standard deviation
average = mean(newData[,2]) #find the average of the data
newData[,2] = (newData[,2]-average)/standardDev # find the distance of
    every data point from the average and divide it by the stdev
max(newData[,2])  #print max values of new data
min(newData[,2]) #print min values of new data
newData[,2] <- (newData[,2]+5)/10 #standardize data on a scale from -5
    to 5

#Use these values adjust max and min to be roughly between .3 and .7;
    allows healthy room for movement in NN
max(newData[,2]) #print max values of new data
min(newData[,2]) #print min values of new data

#write the "new" data to a new csv file
write.csv(newData,"~/Desktop/school/GitHub/CapstoneCode-/Capstone/
    Standardized/Sales Numbers/BY MANUFACTURER/FCA-Chrysler/
    STANDARDIZEDChrysler_monthly_05-21.csv", row.names = FALSE) #
    output new datatset
```

7.3 Ensemble Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 26 16:43:48 2021

@author: nickdorogy
"""

##Import statements
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from sklearn.ensemble import RandomForestClassifier
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from matplotlib import pyplot
import numpy as np
from numpy import mean
from numpy import std
from numpy import array
from numpy import argmax
from numpy import tensordot
from numpy.linalg import norm
from itertools import product


##Initial data load-in and splitting data into X and y
##X is the raw data and y is the standaridzed Consumer Price Index
X = np.genfromtxt("/Users/nickdorogy/Desktop/school/GitHub/
    CapstoneCode-/Capstone/Standardized/!ALL_COMBINED/
    ALL_COMBINED_STANDARDIZED_DATA.csv", delimiter=',', skip_header=1,
     usecols=(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21))
print(X.shape)
y = np.genfromtxt("/Users/nickdorogy/Desktop/school/GitHub/
    CapstoneCode-/Capstone/Standardized/Consumer Price Index/
    STANDARDIZEDConsumerPriceIndex.csv", delimiter=',', skip_header=1)


##Splitting data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

```
    =0.20, random_state=33) #select size of test data (and therefore
    size of train)
y_test = np.reshape(y_test, (40,1)) #reshaping so everything is matrix
     of same dimensions


##Fit model on dataset
def fit_model(X_train, y_train):
    baseline_model = models.Sequential()
    baseline_model.add(layers.Dense(26, activation='sigmoid')) #input
    baseline_model.add(layers.Dense(1)) #output-- obviously want one
    output
    #choose optimizer, loss, & metric
    baseline_model.compile(optimizer='adam',
                  loss=tf.keras.losses.MeanSquaredError(reduction="
    auto", name="mean_squared_error"),
                  metrics=['RootMeanSquaredError'])
    baseline_model.fit(X_train, y_train, epochs=50,batch_size=1) #
    Select number of epochs & batch size
    return baseline_model


##Make an ensemble prediction
def ensemble_predictions(members, X_test):
  #make predictions
    y_predict = [model.predict(X_test) for model in members]
    #reshaping so they are matrices of the same dimensions
    y_predict = [np.reshape(pred, (40,1)) for pred in y_predict]
    #averaging
    result=np.sum(y_predict,axis=0)/len(members)
    return result

```

```
65
66 ##Evaluate accuracy of the model
67 def evaluate_members(members, X_test, y_test):
68     y_predict = ensemble_predictions(members, X_test)
69     accuracy = np.abs(y_predict - y_test)
70     return accuracy
71
72
73 ##Fit all models
74 n_members = 500 # number of models to be averaged
75 members = [fit_model(X_train, y_train) for i in range(n_members)]
76
77
78 ##Calculate & display error
79 ensemble_score = evaluate_members(members, X_test, y_test)
80 single_score = np.abs(np.reshape(members[0].predict(X_test), (40,1))-
       y_test)
81 print("----------------------------------")
82 print("Ensemble Error:", sum(ensemble_score))
83 print("First Network's Error:", sum(single_score))
84
85
86 ##Summarize average accuracy of a single final model
87 print('Accuracy of single score || Mean: %.3f, Std Dev:, (%.3f) ||' %
       (mean(single_score), std(single_score)))
88 print('Accuracy of ensemble score || Mean: %.3f, Std Dev:, (%.3f) ||'
       % (mean(ensemble_score), std(ensemble_score)))
89
90
91 ##Plot scores
92 plt.plot(np.abs(single_score), label="Single network error")
93 plt.plot(np.abs(ensemble_score), label="Ensemble of networks' error")
```

```
plt.plot(ensemble_score-single_score, label="Ensemble - single error")
plt.legend()
plt.xlabel('Time (Months)')
plt.ylabel('Error')
pyplot.show()
```

7.4 Further Average Ensemble Performance Scores

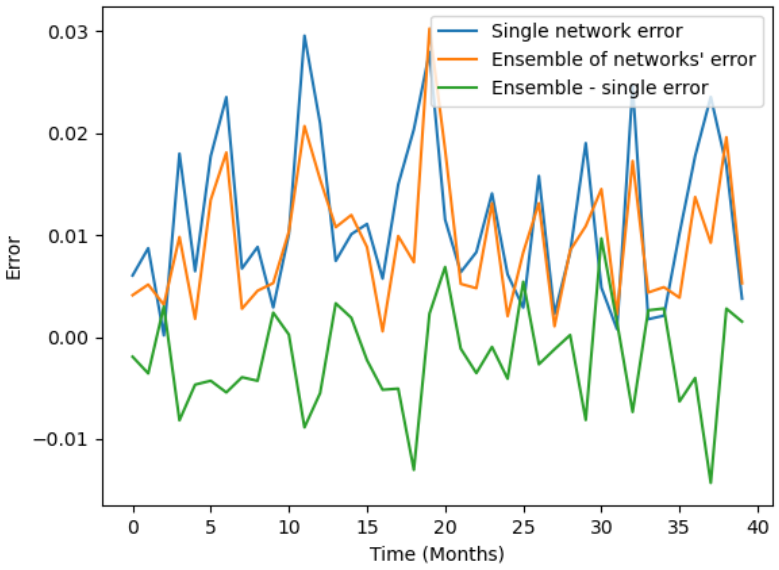

Figure 11. Performance results for an average ensemble of 10 networks

REFERENCES

[1] Hill, K., Cooper, A., and Menk, D., "Contribution of the automotive industry to the economies of all fifty state and the united states," Ann Arbor, MI (2010).

[2] Sompolinsky, H., "Statistical mechanics of neural networks," *American Institute of Physics* **41**, 70 (1988).

[3] Shara, S. A., "Energy levels of the one-dimensional harmonic oscillator," (2011).

[4] Wald, S., "Schematic representation of a configuration of the 2d ising model on a square lattice.," (2017).

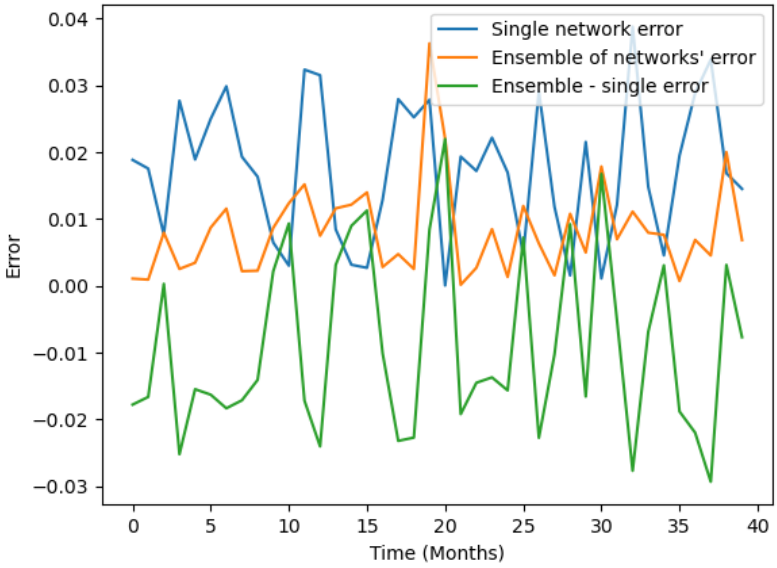

Figure 12. Performance results for an average ensemble of 25 networks

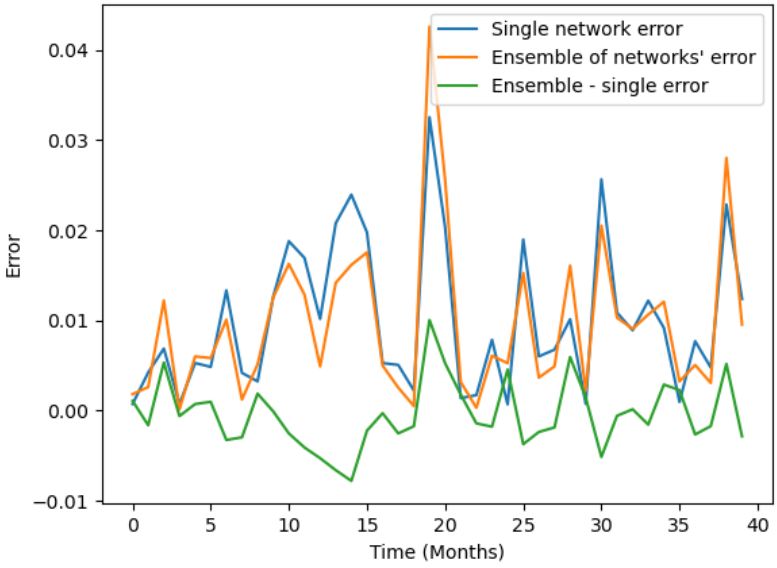

Figure 13. Performance results for an average ensemble of 50 networks

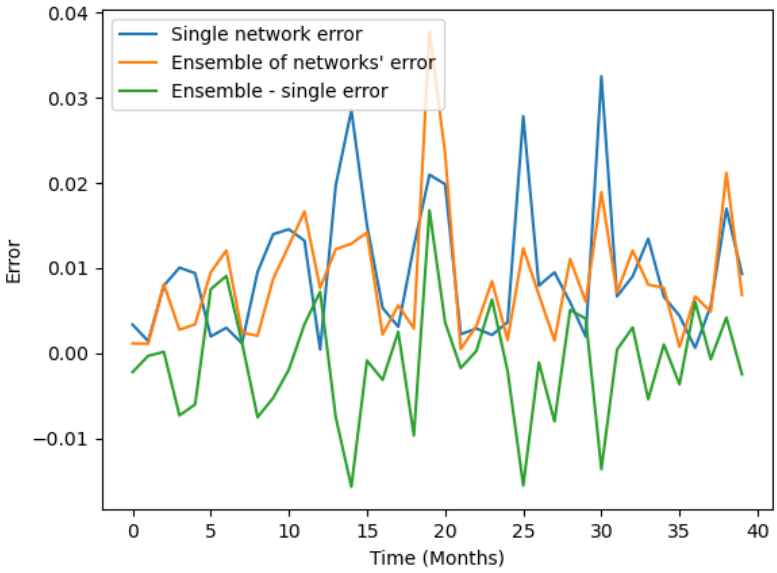

Figure 14. Performance results for an average ensemble of 75 networks

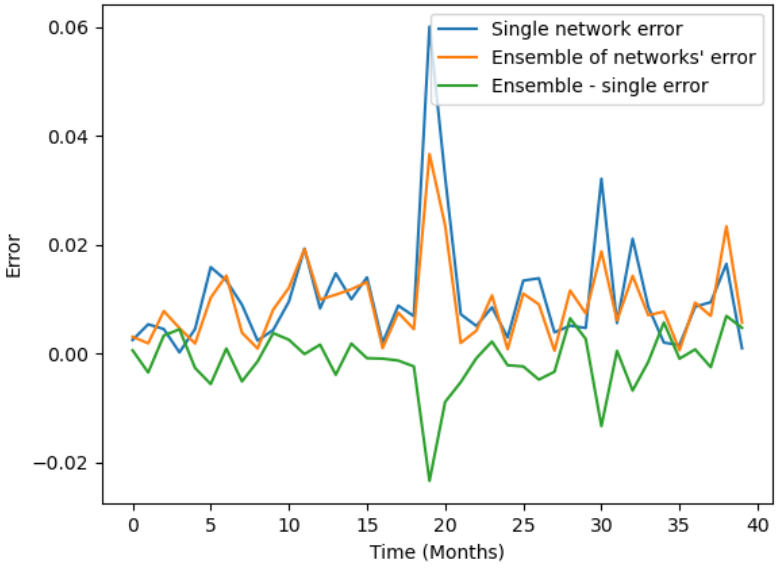

Figure 15. Performance results for an average ensemble of 100 networks

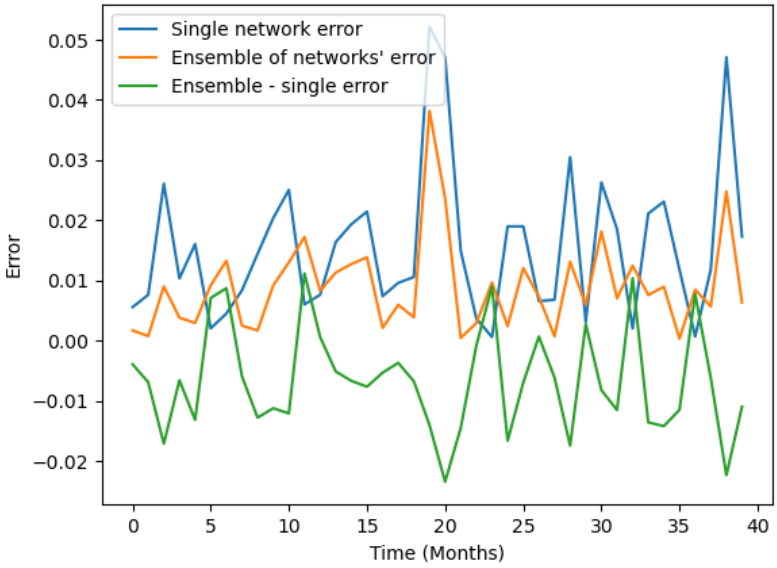

Figure 16. Performance results for an average ensemble of 500 networks

[5] Hopfield, J. J., "Neural networks and physical systems with emergent collective computational abilities," *Proceedings on the National Academy of Sciences* **79**, 2554–2558 (1982).

[6] Patel, S. and Patil, R. B., "Connectionist approach to time series prediction: An empirical test," *Journal of Intelligent Manufacturing* **3**, 317–323 (1992).

[7] Kara, Y., Boyacioglu, M. A., and Ömer Kaan Baykan, "Predicting direction of stock index movement using artificial neural networks and support vector machines: The sample of the istanbul stock exchange," *Expert Systems with Applications* **38**, 5311–5319 (2011).

[8] Panda, C. and Narasimhan, V., "Predicting stock returns: An experiment of the artificial neural network in indian stock market," *South Asia Economic Journal* **7**, 205–218 (2006).

[9] Borovkova, S. and Tsiamas, I., "An ensemble of lstm neural networks for high-frequency stock market classification," *Journal of Forecasting* **38**, 600–619 (2019).

[10] Perrone, M. P. and Cooper, L. N., "When networks disagree: Ensemble methods for hybrid neural networks," 126–142, Chapman and Hall (1993).

[11] Wang, W., Xu, W., Huang, W., and Yang, K., "Enhancing intraday stock price manipulation detection by leveraging recurrent neural," *Neurocomputing* **347**, 46–58 (2019).

[12] Li, Y. and Pan, Y., "A novel ensemble deep learning model for stock prediction based on stock prices and news," *International Journal of Data Science and Analytics* , 1–11 (2021).

[13] Doshi, S., "Various optimization algorithms for training neural network," (January 12, 2019).

[14] Kingma, D. P. and Ba, J., "Adam: A method for stochastic optimization," in [*3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*], Bengio, Y. and LeCun, Y., eds. (2015).