

StockPredictor

June 21, 2016

1 Project 5: Capstone Project

1.0.1 Stock Predictor

1.1 Definition

1.1.1 1. Project Overview

In this section, look to provide a high-level overview of the project in layman's terms. Questions to ask yourself when writing this section:

- Has an overview of the project been provided, such as the problem domain, project origin, and related datasets or input data?
- Has enough background information been given so that an uninformed reader would understand the problem domain and following problem statement?

Background information: “Stock” is a general term used to describe the ownership certificates of any company, in general, and “shares” refers to a the ownership certificates of a particular company. When investors say they own stocks, they are generally referring to their overall ownership in one or more companies. The owners of [Stock and Shares](#) earn money through the process of buying and trading stocks (trading). In addition, many of the more established stocks pay out a dividend quarterly, which can also provide the stockholder with an income. [How to Buy Stocks for Beginners](#)

A stock predictor is a reliable barometer to measure the economic condition of companies as well as can help to assess the performance of the stock market. But, how much should we worry when share prices fall? How does it impact on the average consumer? and how does it affect the economy?

1. Wealth effect

The first impact is that people with shares will see a fall in their wealth. If the fall is significant it will affect their financial outlook. If they are losing money on shares they will be more hesitant to spend money; this can contribute to a fall in consumer spending. However, the effect should not be given too much importance. Often people who buy shares are prepared to lose money; their spending patterns are usually independent of share prices, especially for short term losses. The wealth effect is more prominent in the housing market. In Dec 2014, the value of the UK stock market was US \$6.06 trillion so it has a big impact on wealth.

2. Effect on pensions

Anybody with a private pension or investment trust will be affected by the stock market, at least indirectly. Pension funds invest a significant part of their funds on the stock market. Therefore, if there is a serious fall in share prices, it reduces the value of pension funds. This means that future pension payouts will be lower. If share prices fall too much, pension funds can struggle to meet their promises. The important thing is the long term movements in the share prices. If share prices fall for a long time then it will definitely affect pension funds and future payouts.

3. Confidence

Often share price movements are reflections of what is happening in the economy. E.g. a fear of a recession and global slowdown could cause share prices to fall. The stock market itself can affect consumer confidence. Bad headlines of falling share prices are another factor which discourage people from spending.

On its own it may not have much effect, but combined with falling house prices, share prices can be a discouraging factor. However, there are times when the stock market can appear out of step with the rest of the economy. In the depth of a recession, share prices may rise as investors look forward to a recovery two years in the future.

4. Investment

Falling share prices can hamper firms ability to raise finance on the stock market. Firms who are expanding and wish to borrow often do so by issuing more shares – it provides a low cost way of borrowing more money. However, with falling share prices it becomes much more difficult.

5. Bond market

A fall in the stock market makes other investments more attractive. People may move out of shares and into government bonds or gold. These investments offer a better return in times of uncertainty. Though sometimes the stock market could be falling over concerns in government bond markets (e.g. Euro fiscal crisis)

How does the stock market effect ordinary people?

Most people, who do not own shares, will be largely unaffected by short term movements in the stock market. However, ordinary workers are not completely unaffected by the stock market.

a) Pension funds. Many private pension funds will invest in the stock market. A substantial and prolonged fall in the stock market could lead to a fall in the value of their pension fund, and it could lead to lower pension payouts when they retire. Similarly, if the stock market does well, the value of pension funds could increase. Even if people don't own shares, it is quite likely people with a private pension will have some connection to the stock market.

b) Business investment. The stock market could be a source of business investment, e.g. firms offering new shares to finance investment. This could lead to more jobs and growth. The stock market can be a source of private finance when bank finance is limited. However, the stock market is not usually the first source of finance. Most investment is usually financed through bank loans rather than share options. The stock market only plays a limited role in determining investment and jobs.

c) Short-termism. It could be argued workers and consumers can be adversely affected by the short-termism that the stock market encourages. Shareholders usually want bigger dividends. Therefore, firms listed on the stock market can feel under pressure to increase short-term profits. This can lead to cost cutting which affects workers (e.g. zero contract hours) or the firm may be more tempted to engage in collusive practises which push up prices for consumers. It has been argued that UK firms are more prone to short-termism because the stock market plays a bigger role in financing firms. In Germany, firms are more likely to be financed by long-term loans from banks. Typically, banks are more interested in the long-term success of firms and are willing to encourage more investment, rather than short-term profit maximisation.[Source](#)

Problem domain: The problem domain of this project are the daily stock prices of companies.

Project origin: In this case the project origin is the need to predict the stock prices to make brokers obtain more profit in [daily trading](#). Day traders buy and sell a variety of instruments including stocks, currencies, futures, commodities, indices and ETFs. In this case we will be focusing on predicting stocks in a daily manner.

Goal: Our goal is to help traders to negotiate (buy/sell their stocks) with confidence.

Related datasets or input data: Some of the companies that provide information about stock prices are [Yahoo](#), [Google](#) and [Bloomberg](#). For this project I have chosen to retrieve stocks data through [Yahoo Finance](#).

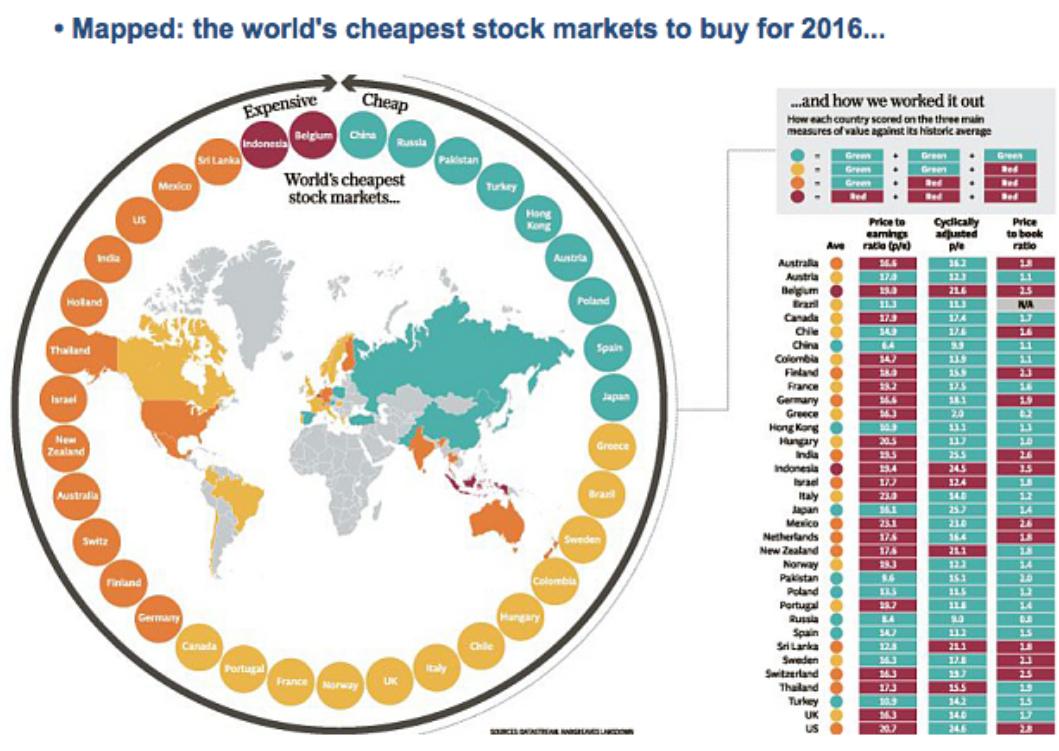
For a [more detailed explanation](#) of what stocks are I recommend you to take a look at the following links: * [History of Money](#) * [History of Stock Market](#) * [25 Basic Stock Market Trading Terms You Should Know](#) * [Glossary of Stock Market terms](#) * [Day Trading Strategies](#) * [Day Trading strategies](#)

In [1]: [#Image1](#)

#Citation:

*#Oxlade, Andrew."Time to 'sell Everything'? No, This Is When a 'hold Everything' Strategy Works
The Telegraph. Telegraph Media Group, 15 Jan. 2016. Web. 5 May 2016.
<[http://www.telegraph.co.uk/finance/personalfinance/investing/12101740/
Time-to-sell-everything-No-this-is-when-a-hold-everything-strategy-works.html](http://www.telegraph.co.uk/finance/personalfinance/investing/12101740/Time-to-sell-everything-No-this-is-when-a-hold-everything-strategy-works.html)>.*

Out [1] :



1.1.2 2. Problem Statement

In this section, you will want to clearly define the problem that you are trying to solve, including the strategy (outline of tasks) you will use to achieve the desired solution. You should also thoroughly discuss what the intended solution will be for this problem. Questions to ask yourself when writing this section:

- Is the problem statement clearly defined? Will the reader understand what you are expecting to solve?
- Have you thoroughly discussed how you will attempt to solve the problem?
- Is an anticipated solution clearly defined? Will the reader understand what results you are looking for?

Considering the volatility of the stock market but knowing the stock prices of previous days the problem is to choose and correctly train a model to predict the stock price of the following day. The solution consists of using an ANN (Artificial Neural Network) of 3 inputNodes, 3 hiddenNodes and 1 outputNode that receives the result of trainingData (outcome from getTrainingData in optimized_analyzer.py). trainingData is an array composed of inputNode[array] and outputNode[array]. It consists on a set of 3 stock prices in total (averages, minimum value and maximum value from the most recent 10 previous dates starting from the given date; at first 20 dates in total are passed) to train the Neural Network, etc.(more details in the following sections) to predict the opening price of the stock for the following day.

In [2]: #Image2

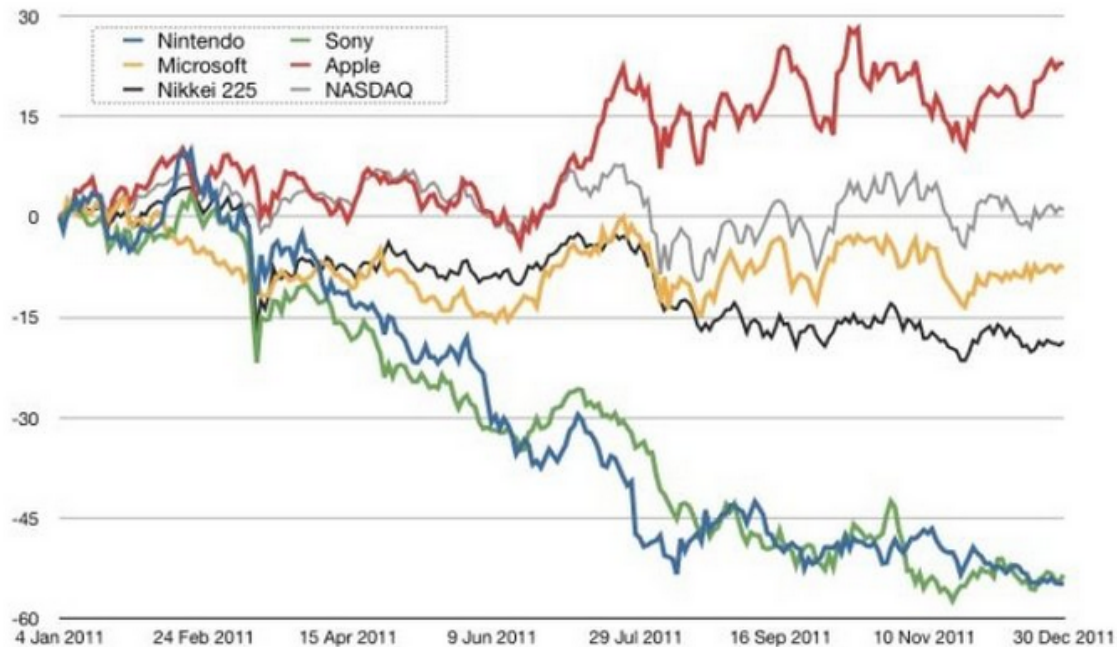
#Citation:

#Fahey, Rob. "Stock Ticker: 2011 In Review."

Gamesindustry.biz. Gamer Network, 09 Jan. 2012. Web. 7 May 2016.

<<http://www.gamesindustry.biz/articles/2012-01-09-stock-ticker-2011-in-review?page=1>>.

Out [2] :



1.1.3 3. Metrics

In this section, you will need to clearly define the metrics or calculations you will use to measure performance of a model or result in your project. These calculations and metrics should be justified based on the characteristics of the problem and problem domain. Questions to ask yourself when writing this section:

- Are the metrics you've chosen to measure the performance of your models clearly discussed and defined?
- Have you provided reasonable justification for the metrics chosen based on the problem and solution?

To measure the performance of the model mainly I compared two arrays of values. The real values against the predictions from the model. Some of the metrics that I chose are:

- A) **Mean Absolute Percent Error:** One problem with the MAE is that the relative size of the error is not always obvious. Sometimes it is hard to tell a big error from a small error. To deal with this problem, we can find the mean absolute error in percentage terms. [More](#).
- B) **Cumulative Forecast Error:** Cumulative gain/loss through the error. It is the sum of all the errors [More](#)

I included other metrics too: 1. **Mean Absolute Error:** The absolute error is the absolute value of the difference between the forecasted value and the actual value. MAE tells us how big of an error we can expect from the forecast on average. [More](#) 2. **Mean Absolute Deviation:** The Mean Absolute Deviation (MAD) of a set of data is the average distance between each data value and the mean. Helps to get a sense of how "spread out" a data set is. [More](#)

3. **r²-score:** R² (coefficient of determination) is a statistical measure of how well the regression line approximates the real data points. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R² score of 0.0. [More](#)

4. **Spearman correlation:** Spearman's correlation coefficient, measures the strength of association between two ranked variables. If there are no repeated data values (which is the case), a perfect Spearman correlation of +1 or -1 occurs when each of the variables is a perfect monotone function of the other.[More](#)
5. **metric#6:** It is the sum of the negative errors from the error list. The errors are the difference between the real values and the predicted values. [More](#)
6. **complement of metric#6:** It is the sum of the positive errors from the error list. The errors are the difference between the real values and the predicted values.[More](#)
7. **average gain through error per stock price:** The average gain through the error in every prediction. It is the sum of the positive errors divided by the number of predictions made.
8. **average loss through error per stock price:** The average loss through the error in every prediction. It is the sum of the negative errors divided by the number of predictions made.
9. **average total gain/loss per stock price:** The average error in every prediction. It is the cumulative forecast error divided by the number of predictions made.
10. **Maximum loss (negative error):** It is the lowest negative error, the lowest value from the error list.
11. **Maximun gain (positive error):** It is the greatest positive error.
12. **Probability of negative error:** It is the probability of having negative errors.
13. **Probability of positive error:** It is the probability of having positive errors.

In [3]: `#Image3`

```
#Citation:
#McGee, Andrea. "Forecasting. "
#SlideShare. SlidePlayer.com Inc.,
#2015. Web. 21 June 2016.
#<http://www.slideshare.net/rushdishams/syntax-27514305>.
```

Out [3]:

Choosing a Forecasting Method

- Mean Absolute Percent Error (MAPE)
 - relative error, scale independent
 - $MAPE = (\sum |\text{forecast error}| / \text{Actual Data}) / n$

1.2 Analysis

1.2.1 4. Data Exploration

In this section, you will be expected to analyze the data you are using for the problem. This data can either be in the form of a dataset (or datasets), input data (or input files), or even an environment. The type of data should be thoroughly described and, if possible, have basic statistics and information presented (such as discussion of input features or defining characteristics about the input or environment). Any abnormalities or interesting qualities about the data that may need to be addressed have been identified (such as features that need to be transformed or the possibility of outliers). Questions to ask yourself when writing this section:

- If a dataset is present for this problem, have you thoroughly discussed certain features about the dataset? Has a data sample been provided to the reader?
- If a dataset is present for this problem, are statistics about the dataset calculated and reported? Have any relevant results from this calculation been discussed?
- If a dataset is not present for this problem, has discussion been made about the input space or input data for your problem?
- Are there any abnormalities or characteristics about the input space or dataset that need to be addressed? (categorical variables, missing values, outliers, etc.)

After choosing a [company ticker symbol](#) and a date (the previous day of the day that you want to predict), the data is retrieved. The **input data** is a set of 10 stock prices in total (from the 20 previous dates starting from the given date) to predict the opening price of the stock for the following day.

Details: Considering the volatility of the stock market but knowing the stock prices of previous days the problem is to choose and correctly train a model to predict the stock price of the following day. The solution

consists of using an ANN (Artificial Neural Network) of 3 inputNodes, 3 hiddenNodes and 1 outputNode that receives the result of trainingData (outcome from getTrainingData in optimized_analyzer.py). trainingData is an array composed of inputNode[array] and outputNode[array]. It consists on a set of 3 stock prices in total (averages, minimum value and maximum value from the most recent 10 previous dates starting from the given date; at first 20 dates in total are passed) to train the Neural Network, etc.(more details in the following sections) to predict the opening price of the stock for the following day.

Some **abnormalities** about the input data are that whenever the date is a holiday, an observance day or a weekend day there is no stock price value for that day because those days the [New York Stock Exchange](#) does not operate. So, the stock prices for those days are omitted. For this reason I just retrieved a larger dataset of stock prices, the 20 most recent previous dates to the date that we want to predict.

```
In [4]: from __future__ import division #import must be at the beginning of the file
import urllib

#This is the "ystockquote" module.
#This module provides a Python API for retrieving stock data from Yahoo Finance.
#Adjusted to receive two inputs:
# symbol and chosen_date instead of symbol, start_date, end_date
def initial_get_historical_prices(symbol, chosen_date):

    #Get historical prices for the given ticker symbol.
    #Date format is 'YYYY-MM-DD'
    #Returns a nested list.

    import datetime
    from datetime import date
    from datetime import timedelta

    #Test
    # >>> chosen_date = '2016-05-10'
    # >>> year = int(chosen_date[:4])
    # >>> month = int(chosen_date[5:7])
    # >>> day = int(chosen_date[8:])
    # >>> end_date = datetime.date(year, month, day)
    # >>> start_date = str(end_date - datetime.timedelta(days=2))

    past_n_days = 10 #fixed
    year = int(chosen_date[:4])
    month = int(chosen_date[5:7])
    day = int(chosen_date[8:])

    end_date = datetime.date(year, month, day)

    if end_date > datetime.date.today():
        statement = "Choose any date before today: " + str(datetime.date.today())
        d0 = end_date
        d1 = datetime.date.today()
        delta = d0 - d1
        past_n_days += delta.days
        # from datetime import date
        # d0 = date(2008, 8, 18)
        # d1 = date(2008, 9, 26)
        # delta = d0 - d1
        # print delta.days
    if end_date == datetime.date.today():
```

```

    past_n_days += 1

    #assert end_date < datetime.date.today(),
    # "chosen date must be any previous day from today: %r" % end_date
    #assert num == 4, "len of set is not 4: %r" % num #example

    #List of dates:
    date_list = [end_date - datetime.timedelta(days=x) for x in range(0, 3)]

    # >>> date_list = [end_date - datetime.timedelta(days=x) for x in range(0, 3)]
    # >>> print date_list
    # [datetime.date(2016, 5, 10), datetime.date(2016, 5, 9), datetime.date(2016, 5, 8)]

    #start_date = str(end_date - datetime.timedelta(days=past_n_days))
    #previous code doesn't work when we previously put from datetime import datetime
    start_date = str(end_date - timedelta(days=past_n_days)) #code is always functional
    end_date = chosen_date

    #month, day and year
    url = 'http://ichart.yahoo.com/table.csv?s=%s&' % symbol + \
        'd=%s&' % str(int(end_date[5:7]) - 1) + \
        'e=%s&' % str(int(end_date[8:10])) + \
        'f=%s&' % str(int(end_date[0:4])) + \
        'g=d&' + \
        'a=%s&' % str(int(start_date[5:7]) - 1) + \
        'b=%s&' % str(int(start_date[8:10])) + \
        'c=%s&' % str(int(start_date[0:4])) + \
        'ignore=.csv'

    print "url"
    print url
    days = urllib.urlopen(url).readlines()
    data = [day[:-2].split(',') for day in days]
    return data

    #Expecting 10 stock prices, but only get 9
    print initial_get_historical_prices('GOOG', '2016-05-27')

```

Out[4]: url

```

http://ichart.yahoo.com/table.csv?s=GOOG&d=4&e=27&f=2016&g=d&a=4&b=17&c=2016&ignore=.csv
[['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Adj Clos'],
 ['2016-05-27', '724.01001', '733.935974', '724.00', '732.659973', '1974000', '732.65997'],
 ['2016-05-26', '722.869995', '728.330017', '720.280029', '724.119995', '1542900', '724.11999'],
 ['2016-05-25', '720.76001', '727.51001', '719.705017', '725.27002', '1629200', '725.2700'],
 ['2016-05-24', '706.859985', '720.969971', '706.859985', '720.090027', '1920400', '720.09002'],
 ['2016-05-23', '706.530029', '711.478027', '704.179993', '704.23999', '1320900', '704.2399'],
 ['2016-05-20', '701.619995', '714.580017', '700.52002', '709.73999', '1816000', '709.7399'],
 ['2016-05-19', '702.359985', '706.00', '696.799988', '700.320007', '1656300', '700.32000'],
 ['2016-05-18', '703.669983', '711.599976', '700.630005', '706.630005', '1763400', '706.63000'],
 ['2016-05-17', '715.98999', '721.52002', '704.109985', '706.22998', '1999500', '706.2299']]

```


1.2.2 5. Exploratory Visualization

In this section, you will need to provide some form of visualization that summarizes or extracts a relevant characteristic or feature about the data. The visualization should adequately support the data being used. Discuss why this visualization was chosen and how it is relevant. Questions to ask yourself when writing this section:

- Have you visualized a relevant characteristic or feature about the dataset or input data?
- Is the visualization thoroughly analyzed and discussed?
- If a plot is provided, are the axes, title, and datum clearly defined?

Some **relevant characteristic** about the **input data** to visualize would be a stock ticker

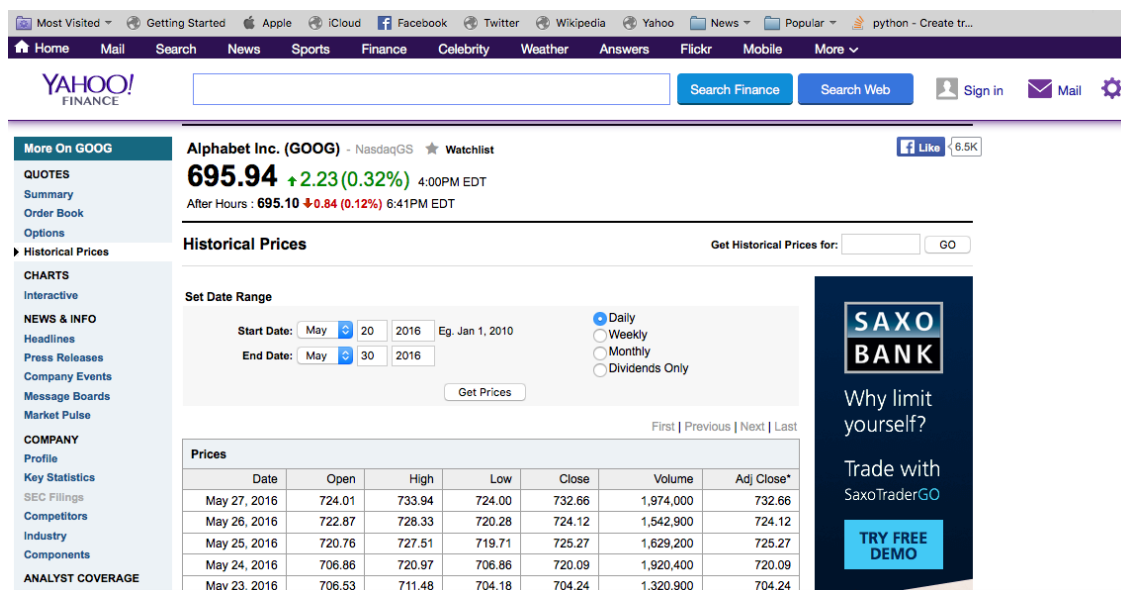
Once that we choose a stock ticker, for example: 'GOOG', and we choose a specific date (a date different from "Today"), then we obtain 20 arrays of stock prices. Later we keep the most recent 10 arrays. Each array has detailed information related to a date. As seen in the array of the labels, each array contains: (1) Date, (2) Open [opening price of the stock], (3) High [highest price of the stock], (4) Low [lowest price of the stock], (5) Close [closing price of the stock], (6) Volume *volume of stocks traded that day*, (7) Adj Clos *adjusted closing price of the stock*.

In [5]: *#Image5*

#Citation:

*#"GOOGLE Inc. Stock Ticker, Historical Prices." Yahoo! Finance. Yahoo! Inc.,
#21 June 2016. Web. 21 June 2016. <http://finance.yahoo.com/q?s=goog&ql=1>.*

Out [5]:



```
In [5]: #Expecting 10 stock prices but only get 6  
print initial_get_historical_prices('GOOG', '2016-05-30')
```

```
Out[5]: url  
http://ichart.yahoo.com/table.csv?s=GOOG&d=4&e=30&f=2016&g=d&a=4&b=19&c=2016&ignore=.csv  
[['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Adj Clos'],  
 ['2016-05-27', '724.01001', '733.935974', '724.00', '732.659973', '1974000', '732.65997'],
```

```
['2016-05-26', '722.869995', '728.330017', '720.280029', '724.119995', '1542900', '724.11999']
['2016-05-25', '720.76001', '727.51001', '719.705017', '725.27002', '1629200', '725.2700'],
['2016-05-24', '706.859985', '720.969971', '706.859985', '720.090027', '1920400', '720.09002']
['2016-05-23', '706.530029', '711.478027', '704.179993', '704.23999', '1320900', '704.2399'],
['2016-05-20', '701.619995', '714.580017', '700.52002', '709.73999', '1816000', '709.7399']
```

1.2.3 6. Algorithms and Techniques

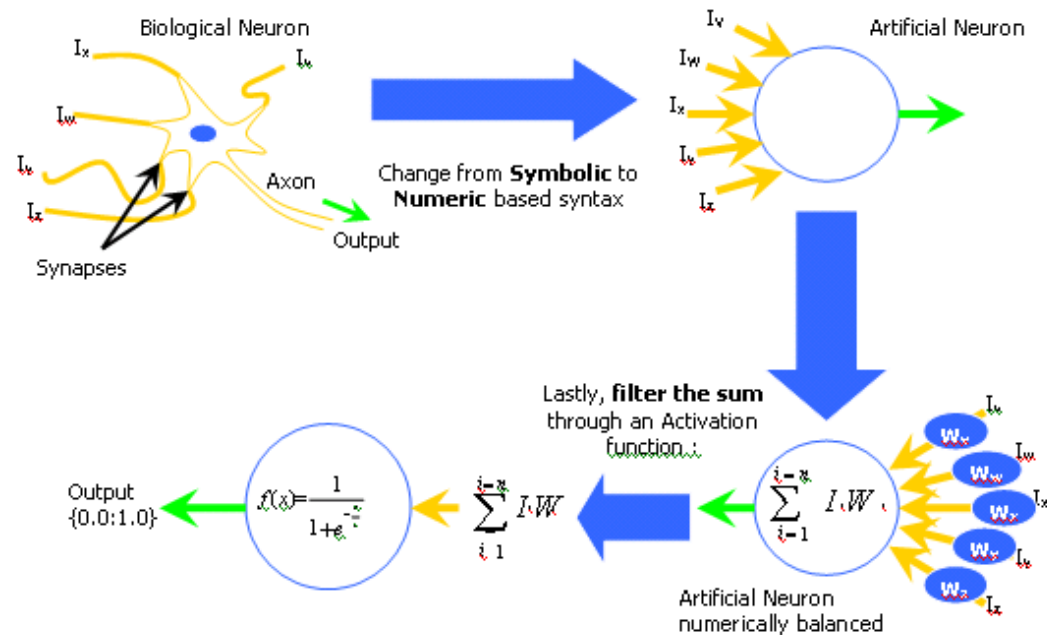
In this section, you will need to discuss the algorithms and techniques you intend to use for solving the problem. You should justify the use of each one based on the characteristics of the problem and the problem domain. Questions to ask yourself when writing this section:

- Are the algorithms you will use, including any default variables/parameters in the project clearly defined?
- Are the techniques to be used thoroughly discussed and justified?
- Is it made clear how the input data or datasets will be handled by the algorithms and techniques chosen?

In [6]: #Image6-a

```
#Citation:
#Jeerge, Gururaj. "Artificial Neural Networks Technology."
# Sasta Servers. Sasta Servers, 25 May 2015. Web. 10 May 2016.
# <http://www.sastaservers.com/blog/artificial-neural-networks-technology/>.
```

Out [6]:



This is how an ANN works:

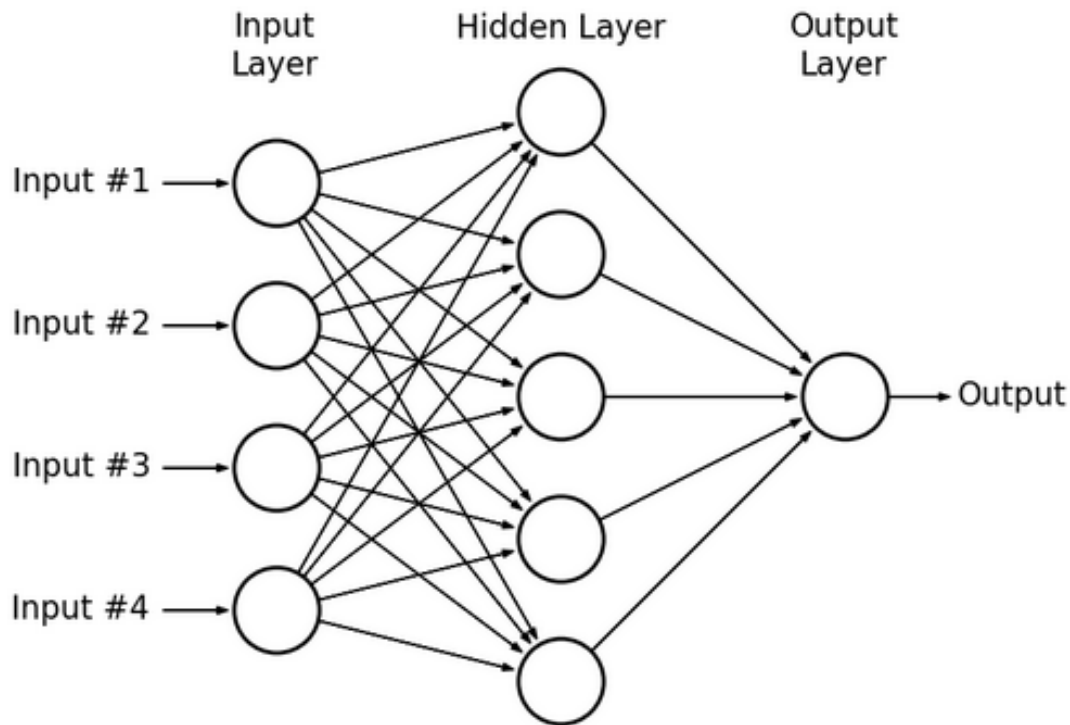
1. Initialize the network, with all weights set to random numbers between 1 and +1.
2. Present the first training pattern, and obtain the output.
3. Compare the network output with the target output.
4. Propagate the error backwards.

5. Calculate the error, by taking the average difference between the target and the output vector.
6. Repeat from 2 for each pattern in the training set to complete one epoch.
7. Repeat from step 2 for a specified number of epochs, or until the error ceases to change. [Details](#)

In [6]: `#Image6-b`

```
#Citation:
#Vanderplas, Jake. "Neural Network Diagram."
#AstroML. AstroML, 2013. Web. 21 June 2016.
#Wonderful code to visualize an ANN diagram
```

Out [6]:



```
In [6]: #!/usr/bin/env python
#
# Copyright (c) 2012, Jake Marsh (http://jakemmarsh.com)
#
# license: GNU LGPL
#
# This library is free software; you can redistribute it and/or
# modify it under the terms of the GNU Lesser General Public
# License as published by the Free Software Foundation; either
# version 2.1 of the License, or (at your option) any later version.
#
# As a guide I took the following code:
```

```

# * http://iamtrask.github.io/2015/07/12/basic-python-network/
# Add visualization of neural network :)

import math, random, string

random.seed(0)

#2nd Change: Lines 365, 368
#3rd Change: Lines 366, 377, 368

## =====

# calculate a random number a <= rand < b
def rand(a, b):
    return (b-a)*random.random() + a

def makeMatrix(I, J, fill = 0.0):
    m = []
    for i in range(I): #loops 4 times
        m.append([fill]*J) #[0.0, 0.0, 0.0]
    return m

# m = [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]

####Value of the activation function parameter: 0.5
def sigmoid(x):
    # tanh is a little nicer than the standard 1/(1+e^-x)
    return math.tanh(x)

# derivative of our sigmoid function, in terms of the output (i.e. y)
def dsigmoid(y):
    return 1.0 - y**2

## =====

class initial_NeuralNetwork:
    def __init__(self, inputNodes, hiddenNodes, outputNodes):
        # number of input, hidden, and output nodes
        self.inputNodes = inputNodes + 1 # +1 for bias node
        self.hiddenNodes = hiddenNodes
        self.outputNodes = outputNodes

        # activations for nodes
        self.inputActivation = [1.0]*self.inputNodes
        #>>> len(c) is 4; c = [1.0]*4 gives c = [1.0, 1.0, 1.0, 1.0]
        self.hiddenActivation = [1.0]*self.hiddenNodes
        #>>> len(d) is 3; d = [1.0]*3 gives d = [1.0, 1.0, 1.0]
        self.outputActivation = [1.0]*self.outputNodes
        #>>> len(e) is 1; e = [1.0]*1 gives e = [1.0]

        # create weights
        self.inputWeight = makeMatrix(self.inputNodes, self.hiddenNodes)
        #len(m) is 4; m = [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]
        self.outputWeight = makeMatrix(self.hiddenNodes, self.outputNodes)

```

```

#len(n) is 1; n = [[0.0], [0.0], [0.0]]
# set them to random vaules
for i in range(self.inputNodes): #loops 4 times
    for j in range(self.hiddenNodes): #loops 3 times
        self.inputWeight[i][j] = rand(-0.2, 0.2) #it's m but with rand(-0.2, 0.2)

#print "self.inputWeight fed with rand(-0.2, 0.2): ", self.inputWeight #len(m) is 4

#self.inputWeight fed with rand(-0.2, 0.2):
#[[0.13776874061001926, 0.10318176117612099, -0.031771367667662004],
# [-0.09643329988281467, 0.004509888547443414, -0.03802634501983429],
# [0.11351943561390904, -0.07867490956842903, -0.009361218339057675],
# [0.03335281578201249, 0.16324515407813406, 0.0018747423269561136]]

for j in range(self.hiddenNodes): #loops 3 times
    for k in range(self.outputNodes): #loops 1 time
        self.outputWeight[j][k] = rand(-2.0, 2.0) #it's m but with rand(-0.2, 0.2)
#print "self.outputWeight fed with rand(-0.2, 0.2): ", self.outputWeight #len(n) is 1
# self.outputWeight fed with rand(-0.2, 0.2):
#[[-0.8726486224011847], [1.0232168166288957], [0.4734759867013265]]

# last change in weights for momentum
self.ci = makeMatrix(self.inputNodes, self.hiddenNodes)
#m = [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]
#print "self.ci last change for momentum: ", self.ci
# self.ci last change for momentum:
#[[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]

self.co = makeMatrix(self.hiddenNodes, self.outputNodes)
#n = [[0.0], [0.0], [0.0]]
#print "self.co last change for momentum: ", self.co
# self.co last change for momentum:
#[[0.0], [0.0], [0.0]]

# Results:
# self.inputWeight fed with rand(-0.2, 0.2):
# [[0.13776874061001926, 0.10318176117612099, -0.031771367667662004],
# [-0.09643329988281467, 0.004509888547443414, -0.03802634501983429],
# [0.11351943561390904, -0.07867490956842903, -0.009361218339057675],
# [0.03335281578201249, 0.16324515407813406, 0.0018747423269561136]]
# self.outputWeight fed with rand(-0.2, 0.2):
# [[-0.8726486224011847], [1.0232168166288957], [0.4734759867013265]]
# self.ci last change for momentum: [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0],
#                                     [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]
# self.co last change for momentum: [[0.0], [0.0], [0.0]]

#[Link](http://cs231n.github.io/neural-networks-1/)

#Analysis: train calls update(Line 290)
#         update uses self.inputWeight and self.outputWeightchanges
#         update changes self.inputActivation, self.hiddenActivation, self.outputActivation
#         update returns self.outputActivation (as an array)
#         train calls backPropagate
#         backPropagate modifies self.inputWeight and self.outputWeight

```

```

#         self.inputWeight and self.outputWeight are used in update
#         repeats itself 1000 times. Then in the analyzer code we call test
#         test calls update (Line 260)
#         update uses self.inputWeight and self.outputWeightchanges (self.inputWeight and
#         self.outputWeight were modified by backpropagate)
#         update changes self.inputActivation, self.hiddenActivation, self.outputActivation
#         update returns self.outputActivation (as an array)
#That is how train and test relate to each other, through self.inputWeight and
# self.outputWeight
#because backpropagate inside train modifies self.inputWeight and self.outputWeight
#and then self.inputWeight and self.outputWeight are used by update which is called inside test

def update(self, inputs):
    #input is [531.9904153999998, 524.052386, 539.172466]
    if len(inputs) != self.inputNodes-1: #if len(inputs) != 3
        raise ValueError('wrong number of inputs')

    # input activations
    for i in range(self.inputNodes-1): #loops 3 times (4-1 = 3 times)
        self.inputActivation[i] = inputs[i]
        #we modify all the values except the default value of the bias input node
    #print "self.inputActivation inside update function: ", self.inputActivation[:]
    #self.inputActivation inside update function:
    #[531.9904153999998, 524.052386, 539.172466, 1.0]
    #before, self.inputActivation was [1.0, 1.0, 1.0, 1.0]

    # hidden activations
    for j in range(self.hiddenNodes): #loops 3 times
        sum = 0.0
        for i in range(self.inputNodes): #loops 4 times
            sum = sum + self.inputActivation[i] * self.inputWeight[i][j]
            #len(m) is 4;
            #m = [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0],
            #      [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]
            #self.inputActivation inside update function:
            #[531.9904153999998, 524.052386, 539.172466, 1.0]

            #self.inputWeight fed with rand(-0.2, 0.2):
            #[[0.13776874061001926, 0.10318176117612099, -0.031771367667662004],
            # [-0.09643329988281467, 0.004509888547443414, -0.03802634501983429],
            # [0.11351943561390904, -0.07867490956842903, -0.009361218339057675],
            # [0.03335281578201249, 0.16324515407813406, 0.0018747423269561136]]
            #Extract (summary)
            #j, loops 3 times
            #i, loops 4 times
            ##j=0
            # Then:
            #self.inputActivation = sia #len(sia) is 4
            # sia[0], sia[1], sia[2], sia[3]
            # m[0][0], m[1][0], m[2][0], m[3][0]
            #self.hiddenActivation[0] = sigmoid(sum) #part of the result
            ##j=1
            # Then:
            #self.inputActivation = sia #len(sia) is 4

```



```

# sia[0], sia[1], sia[2], sia[3]
# m[0][1], m[1][1], m[2][1], m[3][1]
#self.hiddenActivation[1] = sigmoid(sum) #part of the result
##j=2
# Then:
#self.inputActivation = sia #len(sia) is 4
# sia[0], sia[1], sia[2], sia[3]
# m[0][2], m[1][2], m[2][2], m[3][2]
#self.hiddenActivation[2] = sigmoid(sum) #part of the result

self.hiddenActivation[j] = sigmoid(sum)
#>>> len(d) is 3; d = [1.0]*3 gives d = [1.0, 1.0, 1.0]
#before, self.inputActivation was [1.0, 1.0, 1.0]
#print "self.hiddenActivation inside update function: ", self.hiddenActivation[:]
#self.hiddenActivation inside update function:
#[1.0, 0.9999999999998125, -1.0]

# output activations
#print "range(self.outputNodes): ", range(self.outputNodes)
#range(self.outputNodes): [0]
for k in range(self.outputNodes): #loops 1 time, k=0
    sum = 0.0
    for j in range(self.hiddenNodes): #loops 3 times
        sum = sum + self.hiddenActivation[j] * self.outputWeight[j][k]
    self.outputActivation[k] = sigmoid(sum)
#>>> len(e) is 1; e = [1.0]*1 gives e = [1.0]

#print "This is full self.outputWeight: ", self.outputWeight[:]
#This is full self.outputWeight:
#[[-0.8726486224011847], [1.0232168166288957], [0.4734759867013265]]
#before the update self.outputWeight was [[0.0], [0.0], [0.0]]
# range(self.outputNodes): [0], k is always [0], loops 1 time
# This is [j][k]: 0 0 #k from the outer loop, j from the inner loop
# This is [j][k]: 1 0
# This is [j][k]: 2 0
# n[0][0], n[1][0], n[2][0] #len(n) is 1; n = [[0.0], [0.0], [0.0]]

#print "self.outputActivation inside update function: ", self.outputActivation[:]
#self.outputActivation inside update function: [0.9606026038505812]
#before update self.outputActivation was [1.0]
return self.outputActivation[:] #len(self.outputActivation) is 1;
#self.outputActivation = e, e = [1.0]*1 gives e = [1.0]

#Note: self.inputWeight and self.outputWeight are
#modified in backPropagate (also self.co and self.ci)
def backPropagate(self, targets, N, M):
    if len(targets) != self.outputNodes:
        #len(targets) is 1, self.outputNodes = 1
        raise ValueError('wrong number of target values')

    # calculate error terms for output
    output_deltas = [0.0] * self.outputNodes
    #output_deltas = [0.0]*1 or [0.0]
    for k in range(self.outputNodes):

```

```

#loops 1 time, k is always [0]
    error = targets[k]-self.outputActivation[k]
    #self.outputActivation[0] is [1.0]
    output_deltas[k] = dsigmoid(self.outputActivation[k]) * error
    #output_deltas = [0.0] with a different value rather than 0.0
#print "output deltas from backPropagate function: ", output_deltas
#because of the loop with 1000 iterations
#output deltas appears 1000 times with different values
#example
#output deltas from backPropagate function: [0.0025952033592540847]

# calculate error terms for hidden
hidden_deltas = [0.0] * self.hiddenNodes
#hidden_deltas = [0.0]*3 or [0.0, 0.0, 0.0]
for j in range(self.hiddenNodes): #loops 3 times
    error = 0.0
    for k in range(self.outputNodes): #loops 1 time, k is always [0]
        error = error + output_deltas[k]*self.outputWeight[j][k]
        #self.outputWeight is [[0.0], [0.0], [0.0]]
        #of course as update function is called it gives
        #self.outputWeight different values rather than 0.0
    hidden_deltas[j] = dsigmoid(self.hiddenActivation[j]) * error
    #hidden_deltas = [0.0, 0.0, 0.0] (result from [0.0]*3)
    #with different values rather than 0.0
#hidden_Activation is [1.0, 1.0, 1.0] but with different values,
# from [1.0]*3 gives [1.0, 1.0, 1.0]
#print "hidden deltas from backPropagate function: ", hidden_deltas
#because of the loop with 1000 iterations
#hidden deltas appears 1000 times with different values
#example
#hidden deltas from backPropagate function: [-0.0, 1.7789468019844285e-15, -0.0]

# update output weights
for j in range(self.hiddenNodes): #loops 3 times
    for k in range(self.outputNodes): #loops 1 time, k is always [0]
        change = output_deltas[k]*self.hiddenActivation[j]
        #output_deltas = [0.0] with a different value rather than 0.0
        #print "change for self.co :", change
        self.outputWeight[j][k] = self.outputWeight[j][k] + N*change + M*self.co[j][k]
        #self.outputWeight is [[0.0], [0.0], [0.0]]
        #of course as update function is called it gives
        #self.outputWeight different values rather than 0.0
        self.co[j][k] = change #self.co is [[0.0], [0.0], [0.0]]
        #and the float type zeros are substituted by new values, change
#print "self.co from backPropagate function: ", self.co
#because of the loop with 1000 iterations
#self.co appears 1000 times with different values
#example
#self.co from backPropagate function:
#[[0.00013728964264772642], [0.00013728964264770067], [-0.00013728964264772642]]

# update input weights
for i in range(self.inputNodes): #loops 4 times (3+1=4)
    for j in range(self.hiddenNodes): #loops 3 times

```

```

        change = hidden_deltas[j]*self.inputActivation[i]
        #hidden_deltas = [0.0, 0.0, 0.0]
        #(result from [0.0]*3) with different values rather than 0.0
        #print "change for self.ci: ", change
        self.inputWeight[i][j] = self.inputWeight[i][j] + N*change + M*self.ci[i][j]
        #self.inputWeight has length 4 and is:
        # [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0],
        # [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]].
        #The float type zeros are substituted by new values
        self.ci[i][j] = change
        #self.ci has length 4 and is:
        # [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0],
        # [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]].
        #The float type zeros are substituted by new values, change
        #print "self.ci from backPropagate function: ", self.ci
        #because of the loop with 1000 iterations
        #self.ci appears 1000 times with different values
        #example
        #self.ci from backPropagate function:
        # [[0.0, 5.599362764540085e-14, -0.0],
        # [0.0, 5.515812563334366e-14, -0.0],
        # [0.0, 5.674956056333596e-14, -0.0],
        # [0.0, 1.0525307604141634e-16, -0.0]]

        # calculate error
        error = 0.0
        for k in range(len(targets)): #loops 1 time, k is always [0]
            error = error + 0.5*(targets[k] - self.outputActivation[k])**2
        #print "error, return value from backPropagate function: ", error
        return error #because of the loop with 1000 iterations
        #error appears 1000 times with different values
        #example (last value, 1000th value)
        #error, return value from backPropagate function: 3.42294633473e-05

def test(self, inputNodes):
    #print "This is self from input from test from NeuralNetwork: ", self
    #print "This is the input from test from NeuralNetwork: ", inputNodes
    #print(inputNodes, '->', self.update(inputNodes))
    #print "test1 (self.update(inputNodes): ", self.update(inputNodes)
    #prints an array with only one value
    #print "test2 (self.update(inputNodes)[0])", self.update(inputNodes)[0]
    #prints the value (only) without being inside the array
    return self.update(inputNodes)[0]
    #returns the value (only) without being inside the array
    #example: -0.3763290104856086

#from function **test** from neuralNetwork.py. Line 134
# ([699.7640014000001, 692.359985, 711.119995], '->', [-0.3763290104856086])

def weights(self):
    print('Input weights:')
    for i in range(self.inputNodes):
        print(self.inputWeight[i])
    print()

```

```

print('Output weights:')
for j in range(self.hiddenNodes):
    print(self.outputWeight[j])

# Momentum factor:
# To avoid oscillating weight changes the momentum factor is defined.
# Therefore the calculated weight change would not be the same always.
# alpha (Greek letter) is called the momentum factor, and
# typically take values in the range [0.7, 0.95]: 0.1 #change it

# #Momentum
# Momentum basically allows a change to the weights to persist
# for a number of adjustment cycles.
# The magnitude of the persistence is controlled by the momentum factor.
# If the momentum factor is set to 0, then the equation reduces to that of Equation 3.
# If the momentum factor is increased from 0, then increasingly greater persistence of
# previous adjustments is allowed in modifying the current adjustment.
# This can improve the learning rate in some situations,
# by helping to smooth out unusual conditions
# in the training set. [Link](http://www.cheshireeng.com/Neuralyst/nmbg.htm)
# Another technique that can help the network out of local minima is the use of a momentum term
# This is probably the most popular extension of the backprop algorithm;
# it is hard to find cases where this is not used.
# With momentum  $m$ , the weight update at a given time  $t$  becomes
# (the equation where  $M$  is included. Check the link below)
# where  $0 < m < 1$  is a new global parameter which must be determined by trial and error.
# Momentum simply adds a fraction  $m$  of the previous weight update to the current one.
# When the gradient keeps pointing in the same direction,
# this will increase the size of the steps taken towards the minimum.
# It is therefore often necessary to reduce the global learning rate  $\mu$  (Greek letter)
# when using a lot of momentum ( $m$  close to 1).
# If you combine a high learning rate with a lot of momentum,
# you will rush past the minimum with huge steps!

# When the gradient keeps changing direction, momentum will smooth out the variations.
# This is particularly useful when the network is not well-conditioned.
# In such cases the error surface has substantially different curvature along different
# directions,
# leading to the formation of long narrow valleys. For most points on the surface,
# the gradient does not point towards the minimum,
# and successive steps of gradient descent can oscillate from one side to the other,
# progressing only very slowly to the minimum (Fig. 2a).
# Fig. 2b shows how the addition of momentum helps to speed up convergence to the minimum
# by damping these oscillations.
# Link: https://www.willamette.edu/~gorr/classes/cs449/momrate.html

#Learning rate
# With momentum  $m$ , the weight update at a given time  $t$  becomes
# (the equation where  $M$  is included. Check the link below)
# where  $0 < m < 1$  is a new global parameter which must be determined by trial and error.
# Momentum simply adds a fraction  $m$  of the previous weight update to the current one.
# When the gradient keeps pointing in the same direction,
# this will increase the size of the steps taken towards the minimum.
# It is therefore often necessary to reduce the global learning rate  $\mu$  (Greek letter)

```

```

# when using a lot of momentum (m close to 1).
# If you combine a high learning rate with a lot of momentum,
# you will rush past the minimum with huge steps!
# [Link] (https://www.willamette.edu/~gorr/classes/cs449/momrate.html)

#Iterations
#Ya!
# Minimum performance gradient: #1e-7 [Link]
# (http://www.mathworks.com/help/nnet/ref/trainlm.html)
# The network training terminates when either the maximum number of iterations is reached or
# the performance gradient falls below 10-6.
# Furthermore, the generalization performance is evaluated using
# classification accuracy for the classification problem.
# Example:
# *the maximum number of training iterations is: 500
# *the minimum performance gradient is 10-6;and
# *the learning rate is 0.01
# [Link] (https://books.google.com.mx/books?id=rD2FCwAAQBAJ&pg=PA79&lpg=PA79&dq=artificial+neural+network+minimum+performance+gradient&source=bl&ots=Dhj2ag-SMv&sig=oUeCenJ4zbYU5YBMMgjdK0d1ZvA&hl=en&sa=X&ved=0ahUKEwioodKS7fPMAhUSE1IKHaJ3BXYQ6AEIQzAF#v=onepage&q=artificial%20neural%20network%20minimum%20performance%20gradient&f=false)

def initial_train(self, patterns, iterations = 1000, N = 0.5, M = 0.1):
    # N: learning rate, M: momentum factor
    #print "Test1 This is patterns: ", patterns
    #This is patterns (input of train function from NeuralNetwork):
    #[[[531.9904153999998, 524.052386, 539.172466], [1.0000000000000075]]]
    for i in range(iterations):
        error = 0.0
        #print "Test2 This is patterns: ", patterns
        for p in patterns: #an array of 5 arrays (each array has two arrays)
            #print "Test3 This is patterns: ", patterns
            #print "This is p from patterns: ", p
            inputs = p[0] #three items
            #print "This is p[0]: ", inputs
            targets = p[1] #one item
            #print "This is targets: ", targets
            self.update(inputs)
            error = error + self.backPropagate(targets, N, M)
        #if i % 100 == 0:
            #print('error %-.5f' % error)

print "Artificial Neural Network test"
ANN_test = initial_NeuralNetwork(3,3,1)
print "ANN_test: ", ANN_test
print "input Nodes: ", ANN_test.inputNodes
print "hidden Nodes: ", ANN_test.hiddenNodes
print "output Nodes: ", ANN_test.outputNodes
print "input Activation: ", ANN_test.inputActivation
print "hidden Activation: ", ANN_test.hiddenActivation
print "output Activation: ", ANN_test.outputActivation
print "input Weight: ", ANN_test.inputWeight
print "output Weight: ", ANN_test.outputWeight

```

```
print "last change in weight for momentum, input-and-hidden: ", ANN_test.ci
print "last change in weight for momentum, hidden-and-output: ", ANN_test.co
```

Out[6]: Artificial Neural Network test

ANN_test: <__main__.NeuralNetwork instance at 0x101b91f80>

input Nodes: 4

hidden Nodes: 3

output Nodes: 1

input Activation: [1.0, 1.0, 1.0, 1.0]

hidden Activation: [1.0, 1.0, 1.0]

output Activation: [1.0]

input Weight: [[0.13776874061001926, 0.10318176117612099, -0.031771367667662004],
[-0.09643329988281467, 0.004509888547443414, -0.03802634501983429],
[0.11351943561390904, -0.07867490956842903, -0.009361218339057675],
[0.03335281578201249, 0.16324515407813406, 0.0018747423269561136]]

output Weight: [[-0.8726486224011847], [1.0232168166288957], [0.4734759867013265]]

last change in weight for momentum, input-and-hidden:

[[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]

last change in weight for momentum, hidden-and-output: [[0.0], [0.0], [0.0]]

In [6]: #Image6-c

#Citation:

#Dejan Tanikić and Vladimir Despotovic (2012). Artificial Intelligence Techniques

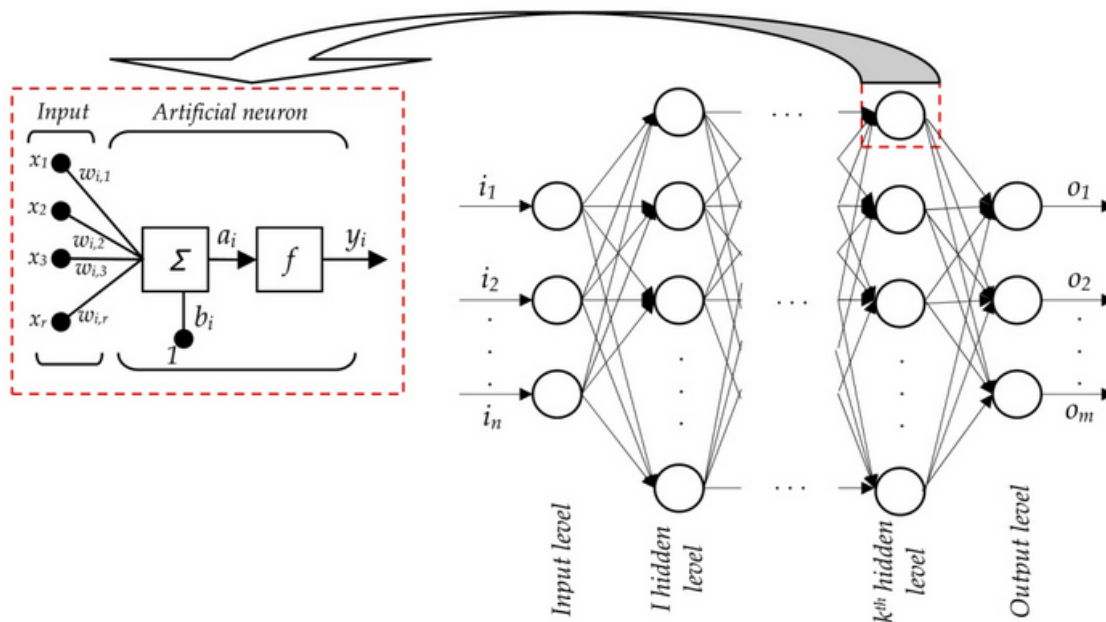
#for Modelling of Temperature in the Metal Cutting Process, Metallurgy - Advances in Materials
#and Processes,

#Dr. Yogiraj Pardhi (Ed.), InTech, DOI: 10.5772/47850.

#Available: <http://www.intechopen.com/books/metallurgy-advances-in-materials-and-processes/>

#artificial-intelligence-techniques-for-modelling-of-temperature-in-the-metal-cutting-process

Out[6]:



In [6]: #Image6-d

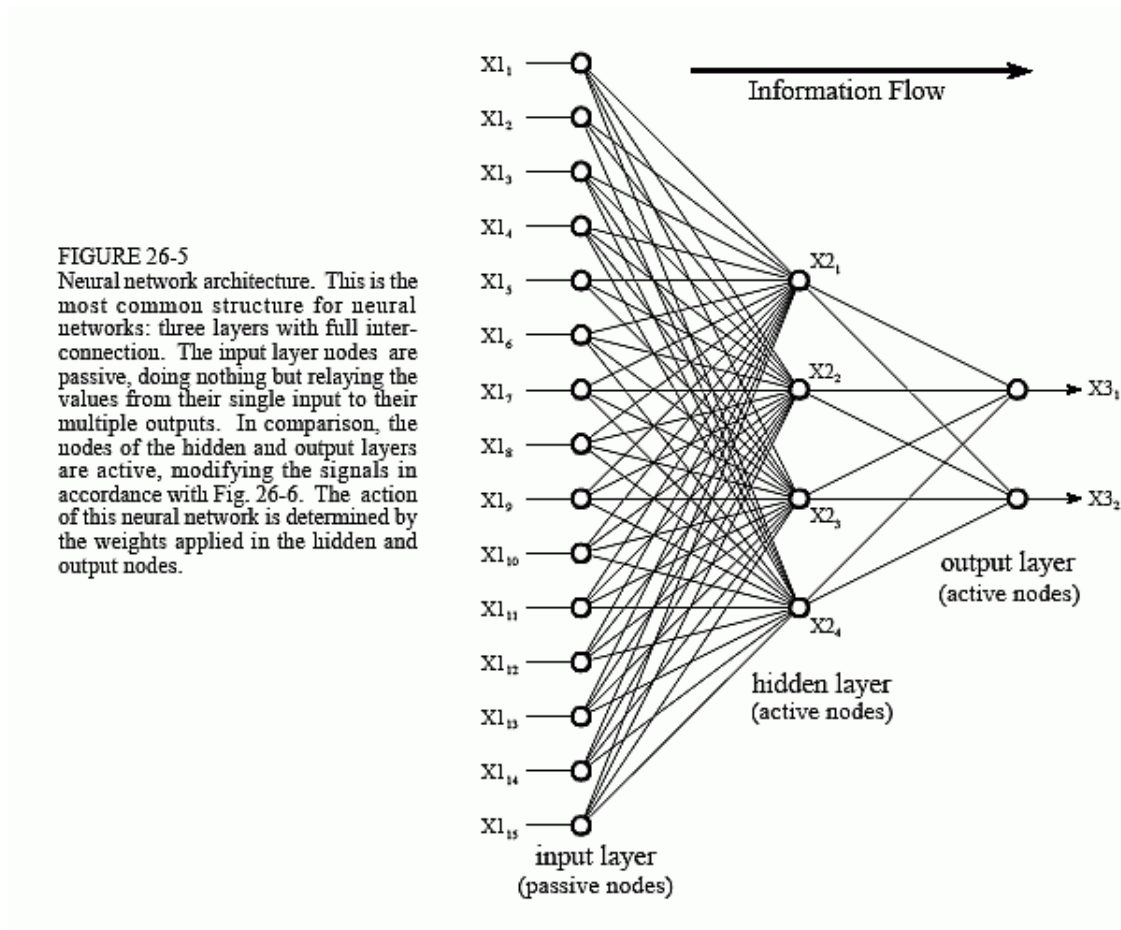
#Citation:

#Smith, Steven W. "Chapter 26/Section2."

#The Scientist and Engineer's Guide to Digital Signal Processing.

#1st ed. N.p.: California Technical, 1997.

Out[6]:



In [6]: #Image6-e

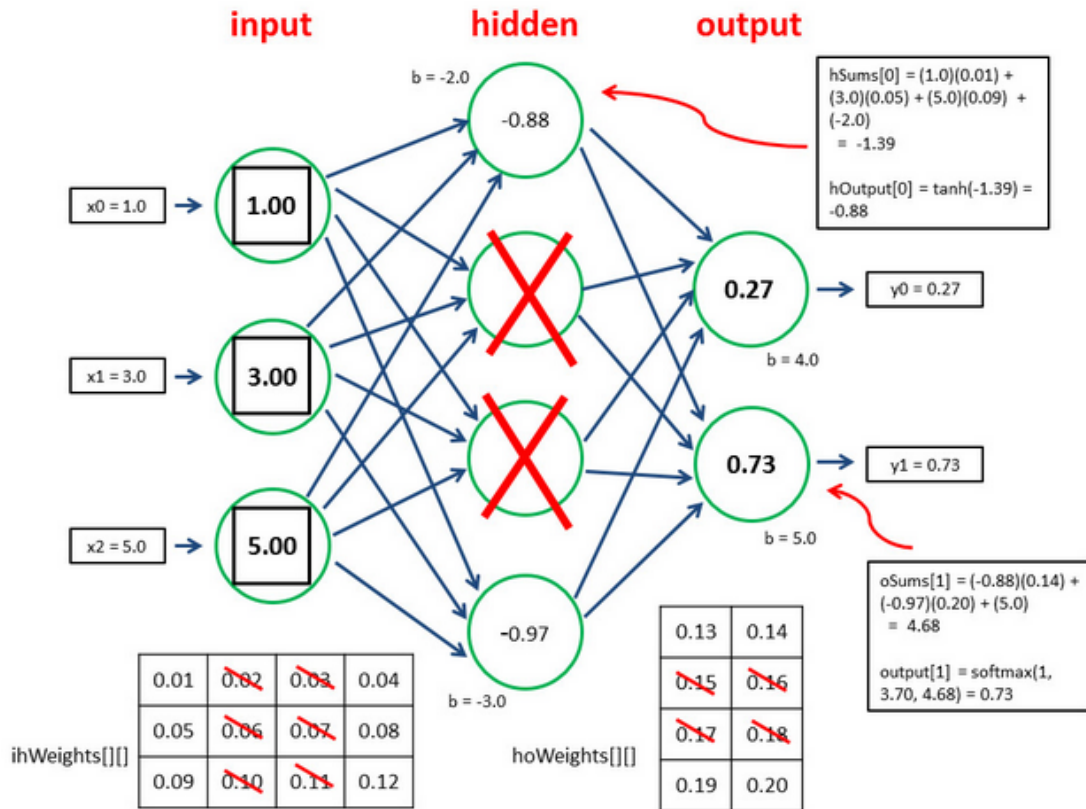
#Citation:

#McCaffrey, James. "Neural Network Dropout Training."

Visual Studio Magazine. Media, Inc., 13 May 2014. Web. 9 May 2016.

<<https://visualstudiomagazine.com/articles/2014/05/01/neural-network-dropout-training.aspx>>

Out[6]:



1.2.4 7. Benchmark

In this section, you will need to provide a clearly defined benchmark result or threshold for comparing across performances obtained by your solution. The reasoning behind the benchmark (in the case where it is not an established result) should be discussed. Questions to ask yourself when writing this section:

- Has some result or value been provided that acts as a benchmark for measuring performance?
- Is it clear how this result or value was obtained (whether by data or by hypothesis)?

My main benchmarks are **Mean Absolute Percent Error** and **Cumulative Forecast Error**.

Mean Absolute Percent Error allows us to compare forecasts of different series in different scales and gives me understanding about the accuracy of my predictions, while Cumulative Forecast Error is the sum of all the forecast errors and is a quick way to see whether my StockPredictor is overestimating the future stock prices or underestimating their value.

Note: A positive Cumulative Forecast Error gives **preference** to the **seller** in daily trading.

In [7]: `#Image7-a`

```
#Citation:
#"Demand Management, Forecast Error & Tracking."
#Apics Forum. ApicsForum.com, 7 Jan. 2012. Web. 6 June 2016.
#<http://www.apicsforum.com/ebook/forecast_error_%2526_tracking>.
```

Out [7]:

Measuring Accuracy, Forecast Errors

- ◆ To compare different time series techniques or to select the “best” set of initial values for the parameters, use a combination of the the following four metrics:

- ◆ Mean Absolute Deviation

- ◆ Most popular *but*

$$\text{MAD} = \frac{\sum_{i=1}^n |A_i - F_i|}{n}$$

- ◆ Mean Absolute Percent Error

- ◆ Should be used in tandem with MAD

$$\text{MAPE} = \frac{100}{n} \sum_{i=1}^n \frac{|A_i - F_i|}{A_i}$$

- ◆ Mean Square Error

$$\text{MSE} = \frac{\sum_{i=1}^n (A_i - F_i)^2}{n}$$

- ◆ Root Mean Square Error

$$\text{RMSE} = \sqrt{\text{MSE}}$$

In [7]: #Image7-b

#Citation:

#Almazrouei, Abdulrahman. "Forecasting." SlideShare.

#LinkedIn, 11 May 2014. Web. 21 June 2016.

#<<http://www.slideshare.net/abdulrahmanalmazrooei5/chapter-13-2>>.

Out [7]:

Measures of Forecast Error

- **Cumulative sum of forecast errors (CFE):** A measurement of the total forecast error that assesses the bias in a forecast. $CFE = \sum E_t$
- **Mean squared error (MSE):** A measurement of the dispersion of forecast errors. $MSE = \frac{\sum E_t^2}{n}$
- **Mean absolute deviation (MAD):** A measurement of the dispersion of forecast errors. $MAD = \frac{\sum |E_t|}{n}$
- **Standard deviation (σ):** A measurement of the dispersion of forecast errors. $\sigma = \sqrt{\frac{\sum (E_t - \bar{E})^2}{n-1}}$

© 2007 Pearson Education



1.3 Methodology

1.3.1 8. Data Preprocessing

In this section, all of your preprocessing steps will need to be clearly documented, if any were necessary. From the previous section, any of the abnormalities or characteristics that you identified about the dataset will be addressed and corrected here. Questions to ask yourself when writing this section:

- If the algorithms chosen require preprocessing steps like feature selection or feature transformations, have they been properly documented?
- Based on the Data Exploration section, if there were abnormalities or characteristics that needed to be addressed, have they been properly corrected?
- If no preprocessing is needed, has it been made clear why?

The algorithms chosen only require us to select **one stock ticker** [Example: 'GOOG'] and **one date** [Format: '2014-04-03'] (one day previous to the date that we want to predict). The date that is chosen as an input along with the ticker symbol must be different from "today" because based on the Data Exploration, I found that there is no closing stock price for "today" until the day has ended [Closing Stock Price](#).

Tons of Detail: `getHistoricalData` (from `optimized.ystockquote.py`) removes the array with the labels and returns the closing stock prices of the 10 most recent days. `getTrainingData` makes sure that we only use the closing prices of five days and then calls `getTimeSeriesValues` which calls `getMovingAverage` which calls `rollingWindow`; `rollingWindow` returns 6 arrays, each array containing 5 closing stock prices in different order. `getMovingAverage` returns 6 values, **the averages of the 5 closing prices in each of the 6 arrays**. `getTimeSeriesValues` also calls `getMinimums` and `getMaximums`. `getMinimums` returns 6 values, **the minimum values from each of the 6 arrays**. `getMaximums` returns 6 values, **the maximum**

values from each of the 6 arrays. Then the values collected (minimums, maximums and averages) are normalized using `normalizePrice` function. In other words: The first minimum value, the first maximum value and the first average value are the input of `normalizePrice` function, the result is stored as an array of length 1 named `outputNode` (inside `getTimeSeriesValues` function). On the other hand the average, the minimum value, the maximum value are stored in an array (reassuring: it's an array of length 3) named `inputNode`. In the end we have 6 arrays of length 3 ordered by pairs (each pair is named `tempItem`): `[inputNode, outputNode]`, `[inputNode, outputNode]`, etc. All the `tempItem` arrays are stored in a variable named `trainingData`. `trainingData` is super important for it carries the data to train the Neural Network.

```
In [8]: #Improved function to retrieve the data
def get_historical_prices(symbol, chosen_date):
    #Get historical prices for the given ticker symbol.
    #Date format is 'YYYY-MM-DD'
    #Returns a nested list.

    import datetime
    from datetime import date
    from datetime import timedelta

    #Test
    # >>> chosen_date = '2016-05-10'
    # >>> year = int(chosen_date[:4])
    # >>> month = int(chosen_date[5:7])
    # >>> day = int(chosen_date[8:])
    # >>> end_date = datetime.date(year, month, day)
    # >>> start_date = str(end_date - datetime.timedelta(days=2))

    past_n_days = 20 #fixed

    year = int(chosen_date[:4])
    month = int(chosen_date[5:7])
    day = int(chosen_date[8:])

    end_date = datetime.date(year, month, day)

    if end_date >= datetime.date.today():
        statement = "Choose any date before today: " + str(datetime.date.today())
        return statement

    #assert end_date < datetime.date.today(),
    #"chosen date must be any previous day from today: %r" % end_date
    #assert num == 4, "len of set is not 4: %r" % num #example

    #start_date = str(end_date - datetime.timedelta(days=past_n_days))
    #previous code doesn't work when we previously put from datetime import datetime
    start_date = str(end_date - timedelta(days=past_n_days)) #code always functional
    end_date = chosen_date

    # #month, day and year
    url = 'http://ichart.yahoo.com/table.csv?s=%s&' % symbol + \
        'd=%s&' % str(int(end_date[5:7]) - 1) + \
        'e=%s&' % str(int(end_date[8:10])) + \
        'f=%s&' % str(int(end_date[0:4])) + \
        'g=d&' + \
```

```

'a=%s&' % str(int(start_date[5:7]) - 1) + \
'b=%s&' % str(int(start_date[8:10])) + \
'c=%s&' % str(int(start_date[0:4])) + \
'ignore=.csv'

#print "url"
#print url
days = urllib.urlopen(url).readlines()
data = [day[:-2].split(',') for day in days]
return data[0:11] #returns a set of 10 stock prices

#Returns the 10 stock prices that we expect
print get_historical_prices('GOOG', '2016-05-30')

```

Out[8]:

```

[['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Adj Clos'],
 ['2016-05-27', '724.01001', '733.935974', '724.00', '732.659973', '1974000', '732.65997'],
 ['2016-05-26', '722.869995', '728.330017', '720.280029', '724.119995', '1542900', '724.11999'],
 ['2016-05-25', '720.76001', '727.51001', '719.705017', '725.27002', '1629200', '725.2700'],
 ['2016-05-24', '706.859985', '720.969971', '706.859985', '720.090027', '1920400', '720.09002'],
 ['2016-05-23', '706.530029', '711.478027', '704.179993', '704.23999', '1320900', '704.2399'],
 ['2016-05-20', '701.619995', '714.580017', '700.52002', '709.73999', '1816000', '709.7399'],
 ['2016-05-19', '702.359985', '706.00', '696.799988', '700.320007', '1656300', '700.32000'],
 ['2016-05-18', '703.669983', '711.599976', '700.630005', '706.630005', '1763400', '706.63000'],
 ['2016-05-17', '715.98999', '721.52002', '704.109985', '706.22998', '1999500', '706.2299'],
 ['2016-05-16', '709.130005', '718.47998', '705.650024', '716.48999', '1316200', '716.4899']]

```

In [8]: #Image8

```


#Citation:
#"NYSE Holidays and Trading Hours (2016)."  

#NYSE: Holidays and Trading Hours. Intercontinental Exchange, Inc., 2016. Web. 21 June 2016.  

#<https://www.nyse.com/markets/hours-calendars>.

```

Out[8]:



[ICE](#)
[NYSE](#)
[ABOUT](#)
[CONTACT](#)

[LOGIN | CONNECT](#)
[LIST](#)
[TRADE](#)
[DATA](#)
[GOVERNANCE SERVICES](#)
[INSIGHTS](#)
[REGULATION](#)

MARKETS

HOLIDAYS AND TRADING HOURS

HOLIDAYS - ALL MARKETS

All NYSE markets observe U.S. holidays as listed below for 2016 and 2017.

Please note that although NYSE Arca Equities follows the same holiday schedule as the NYSE, on days when the NYSE closes early, NYSE Arca Equities hours may vary due to its extended trading day. Review the NYSE Arca Equities section below for more details.

TRADING DAYS

HOLIDAY	2016	2017
New Years Day	January 1	January 1 (Observed Monday, January 2)
Martin Luther King, Jr. Day	January 18	January 16
Washington's Birthday	February 15	February 20
Good Friday	March 25	April 14
Memorial Day	May 30	May 29
Independence Day	July 4	July 4**

NYSE

NYSE ARCA EQUITIES

NYSE MKT

NYSE AMEX OPTIONS

NYSE ARCA OPTIONS

NYSE BONDS

LIQUIDITY PROGRAMS

1.3.2 9. Implementation

In this section, the process for which metrics, algorithms, and techniques that you implemented for the given data will need to be clearly documented. It should be abundantly clear how the implementation was carried out, and discussion should be made regarding any complications that occurred during this process. Questions to ask yourself when writing this section:

- Is it made clear how the algorithms and techniques were implemented with the given datasets or input data?
- Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution?
- Was there any part of the coding process (e.g., writing complicated functions) that should be documented?

Continuing with the previous section, section 8, once that we have the trainingData values ready (inputNode[array] and outputNode[array] stored in an array), we are begin working with our Neural Network. We start by creating a new object of the NeuralNetwork class. To create it we must indicate the number of inputNodes, hiddenNodes and outputNodes. In this case 3 inputNodes, 3 hiddenNodes and 1 outputNode. Then, except for those attributes every attribute gets default values.

The next step is **training the Neural Network** by calling the **train** function and feeding it with the trainingData (inputNode[array] and outputNode[array] stored in an array), to start the replacing of default values from the attributes of the Neural Network. Part of the process of training the Neural Network involves backpropagation. **backpropagate** calculates the error between the outputNode value and the outputActivation value from our neural network. Inside the train function, backpropagate function gets called 1500 times (the value that I chose for iterations).

Going back to optimized_analyzer.py (inside **analyzeSymbol** function), after calling **getPredictionData**, which gives us an array with 3 values: (1) average value from the 6 inputNodes, (2) minimum value of the 6 inputNodes and (3) maximum value of the 6 inputNodes (the 6 arrays mentioned are of this type: [inputNodes, outputNode] and came from rollingWindow), **we proceed to test our Neural Network** by calling **test** (function from the NeuralNetwork class) and we feed it with the return value of getPredictionData. Inside test function, update function (also from NeuralNetwork) gets called. **Update receives the inputNode** (an array with 3 values), and using **the activation function**, sigmoid, calculates the new value of outputActivation (NeuralNetwork class attribute). The very initial value of outputActivation class attribute is 1.

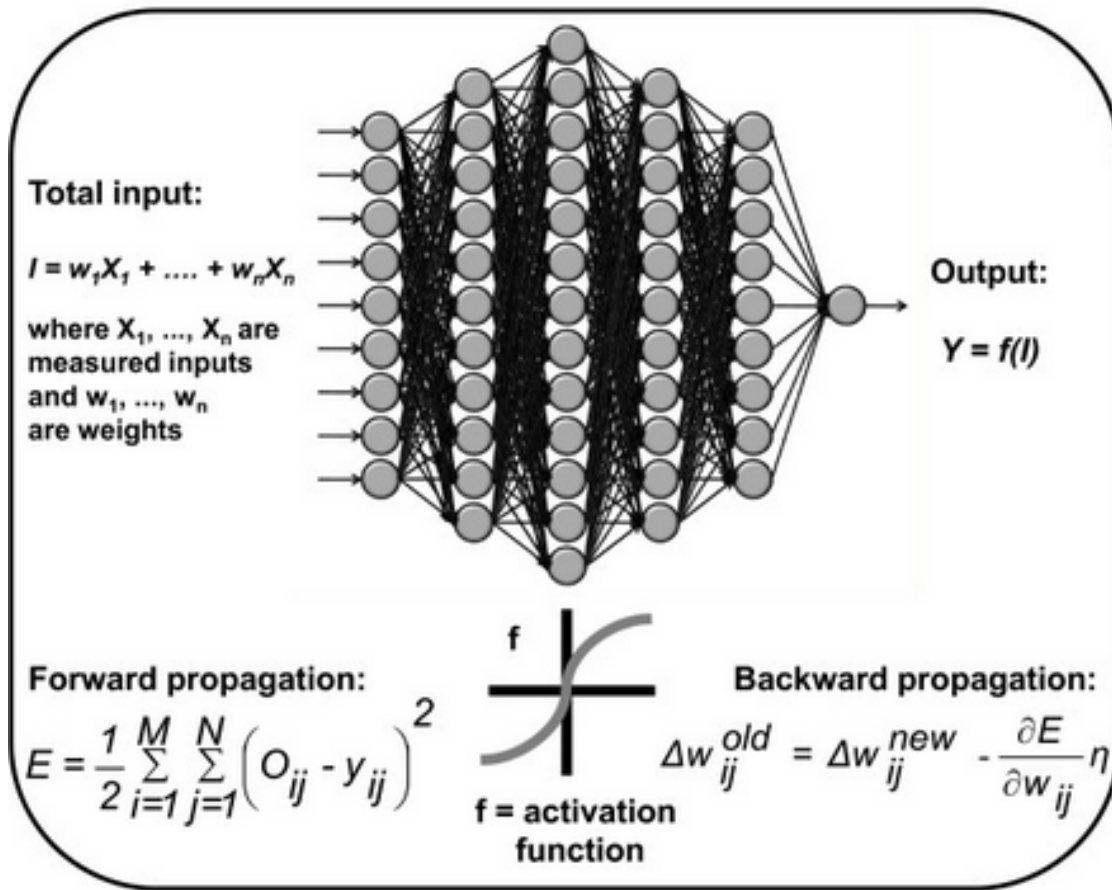
The output of calling test having fed it with predictionData is an **updated value of the outputActivation** which is stored in a variable **named returnPrice**. Next, inside analyzeSymbol (from optimized_analyzer.py), we use the **denormalize** function (we feed it with returnPrice and with the minimum and maximum value from predictionData). The **final result** is the **predictedStockPrice**.

analyzeSymbol returns **the predictedStockPrice** and **processing time** (time that the program takes to perform the final prediction from the start).

In [9]: #Image9

```
#Citation:
#Røe, Kathrine, Manish Kakar, Therese Seierstad, Anne H. Ree, and Dag R. Olsen.
#Illustration of the Architecture of the Back Propagation Neural Network (BPNN).
#Digital image. Yale Image Finder. Krauthammer Lab, Yale University,
#8 June 2011. Web. 9 May 2016.
#<http://krauthammerlab.med.yale.edu/imagefinder/Figure.
# external?sp=SPMC3130663%2F1748-717X-6-65-3
# &state:Figure=Br00ABXcRAAAAAQAACmRvY3VtZW50SWR
# zcgARamF2YS5sYW5nLkludGVnZXIS4qCk94GHOAIAAUkABXZhH
# VleHIAEGphdmEubGFuZy50dW1iZXKGrJUdC5TgiwIAAHwAAryyg%3D%3D>.
```

Out [9]:



1.3.3 10. Refinement

In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Questions to ask yourself when writing this section:

- Has an initial solution been found and clearly reported?
- Is the process of improvement clearly documented, such as what techniques were used?
- Are intermediate and final solutions clearly reported as the process is improved.

The initial solution to this problem was predicting the stock price of the following day using a neural Network with 3 input neurons (plus 1 bias input neuron), 3 hidden neurons and 3 output neurons. The code is found in the files: my_first_analyzer.py and my_first_neuralNetwork.py

The improvements that I made were the following:

- 1) **The process of extracting the data from Yahoo Finance:** I modified the main function in the file analyzer.py and using other helper functions I made sure to “always” (no matter if the previous dates are holidays or weekends or observed days) return a set of 10 stock prices. The outcome from this

improvement can be seen mainly by comparing the cumulative forecast error of the previous prediction with this new prediction set.

- Previous mean absolute percent error is: 2.56467188276
- Previous cumulative forecast error: 114.429396164
- New mean absolute percent error is: 3.07061480364
- New cumulative forecast error: 98.222619522

2) **Momentum factor, Iterations and Learning Rate:** I changed the Momentum factor (M), the Learning Rate (N) and the number of Iterations in the file neuralNetwork.py. The new number of Iterations was 2500. And the Momentum Factor and the Learning rate were the same than in the previous model, but in the loop (iterations: 2500) I decreased the Learning Rater and increased the Momentum factor. The outcome from this improvement can be seen by comparing the mean absolute percent error is and the cumulative forecast error of the previous prediction with this new prediction set.

- Previous mean absolute percent error is: 3.07061480364
- Previous cumulative forecast error: 98.222619522
- New mean absolute percent error is: 3.07427704404
- New cumulative forecast error: 93.1659690816

3) **Momentum Factor:** I modified the Momentum factor's value inside the loop. Instead of increasing it by a constant value, I chose random constant values in a specific range. The outcome from this improvement can be seen by comparing the mean absolute percent error is and the cumulative forecast error of the previous prediction with this new prediction set.

- Previous mean absolute percent error is: 3.07427704404
- Previous cumulative forecast error: 93.1659690816
- New mean absolute percent error is: 3.06091931524
- New cumulative forecast error: 91.5888261064

As the small value of the **Mean Absolute Percent Error** indicates, my Stock Predictor tends to overfit the actual data.

Through the **Cumulative Forecast Error** (positive values) we can observe that the predictions tends to be a bit larger compared to the actual values.

I think that way more data (more and **random** stock tickers and dates) must be brought to play in order to choose better values for number of iterations, learning rate and momentum factor. This will give the client full confidence to use my Stock Predictor.

In [10]: *#Image10-a*

```
#Citation:  
#"Momentum and Learning Rate Adaptation (Momentum)."  
#CS449. Willamette University, 2015. Web. 21 June 2016.  
#<https://www.willamette.edu/~gorr/classes/cs449/momrate.html>.
```

Out [10]:

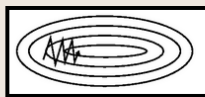
Momentum

Another technique that can help the network out of local minima is the use of a **momentum** term. This is probably the most popular extension of the backprop algorithm; it is hard to find cases where this is not used. With momentum m , the weight update at a given time t becomes

$$\Delta w_{ij}(t) = \mu_i \delta_i y_j + m \Delta w_{ij}(t-1) \quad (1)$$

where $0 < m < 1$ is a new global parameter which must be determined by trial and error. Momentum simply adds a fraction m of the previous weight update to the current one. When the gradient keeps pointing in the same direction, this will increase the size of the steps taken towards the minimum. It is therefore often necessary to reduce the global learning rate μ when using a lot of momentum (m close to 1). If you combine a high learning rate with a lot of momentum, you will rush past the minimum with huge steps!

When the gradient keeps changing direction, momentum will smooth out the variations. This is particularly useful when the network is not well-conditioned. In such cases the error surface has substantially different curvature along different directions, leading to the formation of long narrow valleys. For most points on the surface, the gradient does not point towards the minimum, and successive steps of gradient descent can oscillate from one side to the other, progressing only very slowly to the minimum (Fig. 2a). Fig. 2b shows how the addition of momentum helps to speed up convergence to the minimum by damping these oscillations.



(Fig. 2a)

steep valley 2

(Fig. 2b)

To illustrate this effect in practice, we trained 20 networks on a simple problem (4-2-4 encoding), both with and without momentum. The mean training times (in epochs) were

momentum	Training time
0	217
0.9	95

In [10]: #Image10-b

#Citation:

#"Momentum and Learning Rate Adaptation (Learning Rate Adaptation)."

#CS449. Willamette University, 2015. Web. 21 June 2016.

#<<https://www.willamette.edu/~gorr/classes/cs449/momrate.html>>.

Out [10]:

Learning Rate Adaptation

In the section on [preconditioning](#), we have employed simple heuristics to arrive at reasonable guesses for the global and local learning rates. It is possible to refine these values significantly once training has commenced, and the network's response to the data can be observed. We will now introduce a few methods that can do so *automatically* by adapting the learning rates during training.

Bold Driver

A useful batch method for adapting the global learning rate μ is the **bold driver** algorithm. Its operation is simple: after each epoch, compare the network's loss $E(t)$ to its previous value, $E(t-1)$. If the error has decreased, increase μ by a small proportion (typically 1%-5%). If the error has increased by more than a tiny proportion (say, 10^{-10}), however, undo the last weight change, and decrease μ sharply - typically by 30%. Thus bold driver will keep growing μ slowly until it finds itself taking a step that has clearly gone too far up onto the opposite slope of the error function. Since this means that the network has arrived in a tricky area of the error surface, it makes sense to reduce the step size quite drastically at this point.

Annealing

Unfortunately bold driver cannot be used in this form for online learning: the stochastic fluctuations in $E(t)$ would hopelessly confuse the algorithm. If we keep μ fixed, however, these same fluctuations prevent the network from ever properly converging to the minimum - instead we end up randomly dancing around it. In order to actually reach the minimum, and stay there, we must **anneal** (gradually lower) the global learning rate. A simple, non-adaptive annealing schedule for this purpose is the **search-then-converge** schedule

$$\mu(t) = \mu(0)/(1 + t/T) \quad (2)$$

Its name derives from the fact that it keeps μ nearly constant for the first T training patterns, allowing the network to find the general location of the minimum, before annealing it at a (very slow) pace that is known from theory to guarantee convergence to the minimum. The characteristic time T of this schedule is a new free parameter that must be determined by trial and error.

Local Rate Adaptation

If we are willing to be a little more sophisticated, we go a lot further than the above global methods. First let us define an online weight update that uses a local, time-varying learning rate for each weight:

$$w_{ij}(t+1) = w_{ij}(t) + \mu_{ij}(t) \Delta w_{ij}(t) \quad (3)$$

The idea is to adapt these local learning rates by gradient descent, while simultaneously adapting the weights. At time t , we would like to change the learning rate (before changing the weight) such that the loss $E(t+1)$ at the next time step is reduced. The gradient we need is

$$\frac{\partial E(t+1)}{\partial \mu_{ij}(t)} = \frac{\partial E(t+1)}{\partial w_{ij}(t+1)} \frac{\partial w_{ij}(t+1)}{\partial \mu_{ij}(t)} = -\Delta w_{ij}(t) \Delta w_{ij}(t-1) \quad (4)$$

1.4 Results

1.4.1 11. Model Evaluation and Validation

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should

be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model's solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:

- Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?
- Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?
- Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?
- Can results found from the model be trusted?

The final model consists on an ANN with 3 inputNodes, 3 hiddenNodes and 1 outputNode.

The ANN receives trainingData: inputNodes [3 values], outputNode [1 value].

*The ANN's input layers are: 3 plus the bias, 4 in total.

*The ANN's hidden layers are: 4 times 3, 12 in total, (it's like the information flows back and forward through the layers).

*The ANN's output layers are: 3

*The ANN's activation function is: Sigmoid function (found in optimized_neuralNetwork.py). It is the hyperbolic tangent function (tanh).

*The number of Iterantions (epochs) are: 1500

*The Learning rate is: 0.5 (decreases up to 0.0015, approx)

*The Momentum Factor is: 0.1 (increases up to 0.945, approx)

Glance @ the Benchmark:

* Current model's mean absolute percent error is: 2.58488413024

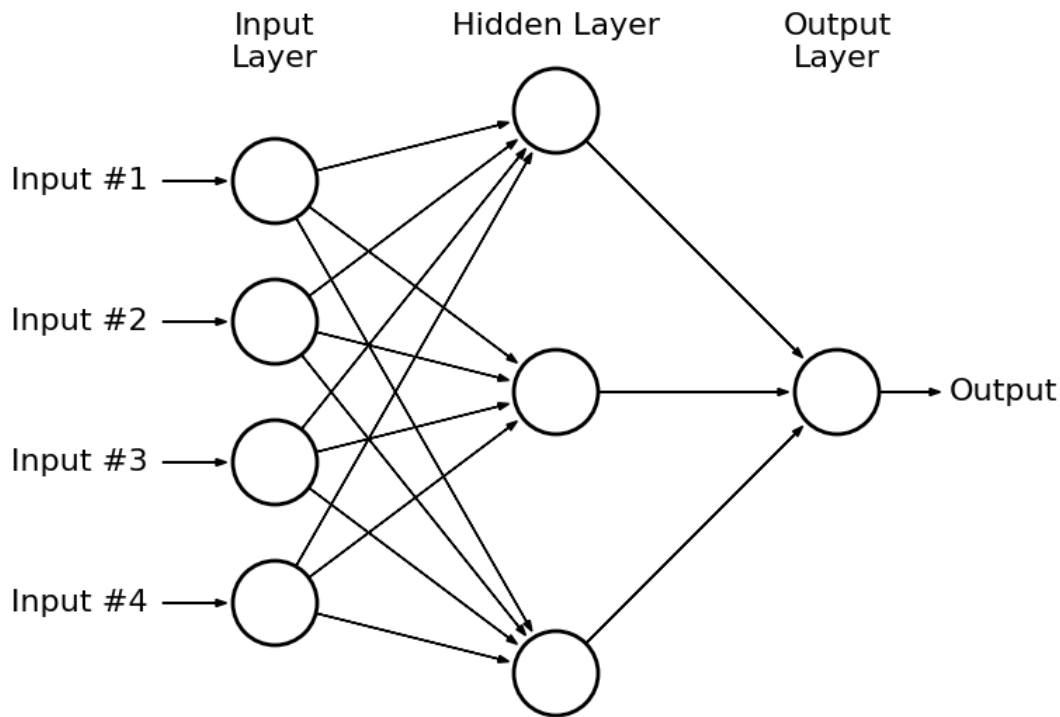
* Current model's cumulative forecast error is: 77.2415129785

In general this model outputs reasonable predictions for calculating the stock price of the following day. But because of the small values from Mean Absolute Percent Error, it tends to overfit the real data. And well, as seen through the Cumulative Forecast Error values, always positive values, this Stock Predictor tends to benefit more the seller than the buyer in daily trading. So the predictions must be used with care.

In [11]: *#Image11*

#Details found in ANN_diagram.py

Out[11]:



1.4.2 12. Justification

In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

- Are the final results found stronger than the benchmark result reported earlier?
- Have you thoroughly analyzed and discussed the final solution?
- Is the final solution significant enough to have solved the problem?

After the model improvements (making sure that we count with a uniform set of input data and using an Artificial Neural Network with three input neurons) and choosing an optimal value for the Momentum factor (M) and Iterations, the final results found became stronger than the first benchmark. We obtained a larger value of **Mean Absolute Percent Error**, which indicates that the model is a little less overfitting. And the positive value of the **Cumulative Forecast Error** decreased, which says that the predictions became fairer for the stocks **buyer** in daily trading.

First Benchmark:

- * First model's mean absolute percent error is: 2.56467188276
- * First model's cumulative forecast error is: 114.429396164

New Benchmark:

- * Current model's mean absolute percent error is: 2.58488413024
- * Current model's cumulative forecast error is: 77.2415129785

Note: For a more detailed report on the Benchmarks check the file `outputComparison.txt`

The predictions from the final models are sufficiently reasonable to be trusted if one is the seller in daily trading. But if one is the buyer, then you should take this into account before accepting any daily stock trading offer:

- 1) `get_price_earnings_ratio(symbol)`
- 2) `get_price_earnings_growth_ratio(symbol)`
- 3) `get_price_book_ratio(symbol)`
- 4) D/E ratio, [Debt to Equity](#) also called Gearing ratio
- 5) FCE, [Free Cashflow](#)

Sources: Taken from [Fundamental stock metrics](#) and [Yahoo finance ticker symbols](#)

In [12]: `#Image12`

```
#Citation:  
#Oti, Hellen. "New Study Reveals Surprising Difference  
#Between Buyer and Seller Perspectives." MarketingWorld. MarketingWorld,  
#15 Apr. 2014. Web. 21 June 2016.  
#<http://www.marketing-world.net/new-study-reveals-surprising-difference-  
# between-buyer-and-seller-perspectives/2014/04/>.
```

Out[12]:



1.5 Conclusion

1.5.1 13. Free-Form Visualization

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

- Have you visualized a relevant or important quality about the problem, dataset, input data, or results?
- Is the visualization thoroughly analyzed and discussed?
- If a plot is provided, are the axes, title, and datum clearly defined?

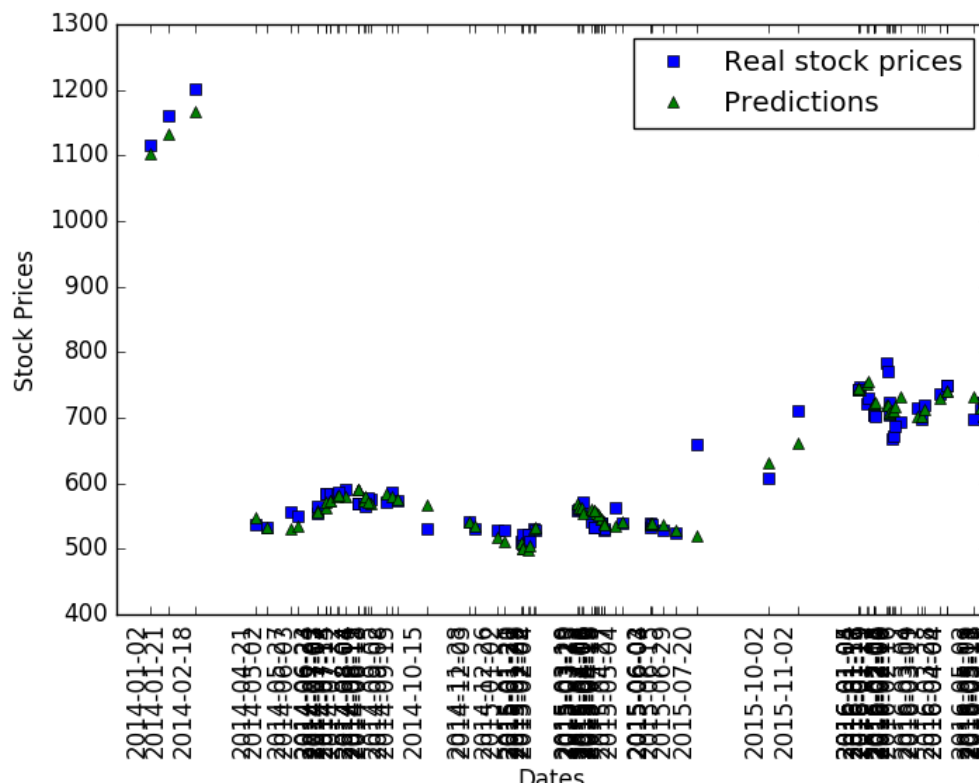
For the **Free-Form Visualization**, I am showing:

- 1) A **plot** of my **Stock Predictions** vs the **Actual Stock values**
- 2) The **Residual plot** from the Predictions.

In [13]: *#Image13-a (Main Plot)*

```
#The Predictions vs The Actual Stock Prices  
#Details found in ANN_diagram.py
```

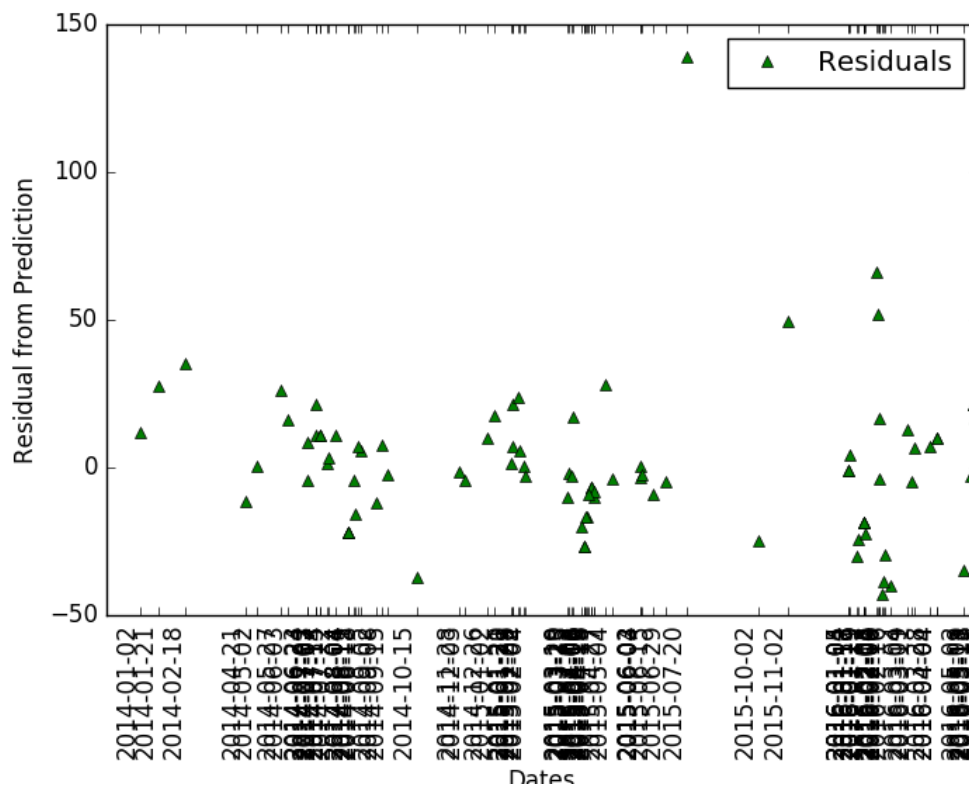
Out[13]:



In [13]: *#Image13-b (Residual Plot)*

```
#The Residuals (difference between Predictions and the Actual Stock Prices)  
#Details found in ANN_diagram.py
```

Out [13]:



1.5.2 14. Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

- Have you thoroughly summarized the entire process you used for this project?
- Were there any interesting aspects of the project?
- Were there any difficult aspects of the project?
- Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?

This project was about predicting stock prices. I did it by creating an Artificial Neural Network. The ANN (Artificial Neural Network) learns from the input data and does multiple re-calibrations for each linkage weights. The input weights of the ANN are the stock prices of the current day and the two previous days. The main functions used by the ANN are the activation function and the error function (this function helps to re-calibrate the weights). And of course the output of the ANN is the predicted stock price for the following day [How ANNs works](#).

Beyond focusing only in the activity of the New York Stock Exchange, some relevant or important qualities about the problem that involve the input data and the results are the time zones and the trading times. Although the trading hours of the New York Stock Exchange market go from 9:30 in the morning to

4 in the afternoon (Eastern Time), there are stocks that happen before 9:30 a.m. and after 4 p.m. There are other exchanges across the globe, in other countries, other US cities, and even in New York itself.

While in New York (USA - New York) it is Thursday, May 5, 2016 at 9:30:00 AM EDT UTC-4 hours, in Saint-Petersburg (Russia - Saint Petersburg) it is Thursday, May 5, 2016 at 4:30:00 PM. It is clear to notice how the time zones play a special role in daily trading. Because of Time Zones, with the exception of the weekends, the Stock Market never stops operating. For this reason additional data can be selected and included as input data if one wishes to be way more accurate predicting the closing stock price of the day (careful with overfitting). A better prediction prevents traders from paying more for stocks than they should, along the day. Interesting to know: “morning time is when daily traders lose most money”, [Why Morning Is the Worst Time to Trade Stocks](#).

Also, after studying the behavior of the Stock Market, I identified the volatility of the Stock Market as challenge in the task of predicting future daily stock prices. With this I mean that besides all metrics and numbers there are other big and important factors that are decisive in the future value of stocks. New technology can replace the current technology, which means that some companies will disappear and new ones will be brought to the market. For example: [Apple Inc.](#), [Google Inc.](#), [Samsung\(\)](#) and other companies* that virtually (software) or physically (electronics) developed and improved Android and iOS platform technologies in the cellphones* replaced and took a big chunk of money from the digital camera and video camera companies like [Kodak](#) and others. Another example is how virtual streaming companies like [Netflix](#) replaced the service provided by [Blockbuster](#), the physical rent of movies. More variables should be taken into account to predict stock prices with more accuracy. Doing that this model can be used in a general setting to predict daily stock prices.

In [14]: `#Image14-a`

#Citation:

#Rijo, Dahiana. "MENSAJE DETRÁS DE STAY HUNGRY, STAY FOOLISH."

DR. Dahiana Rijo, 28 Feb. 2015. Web. 9 May 2016.

<<http://dahianarijo.com/mensaje-detras-de-stay-hungry-stay-foolish/>>.

Out [14]:



In [14]: #Image14-b

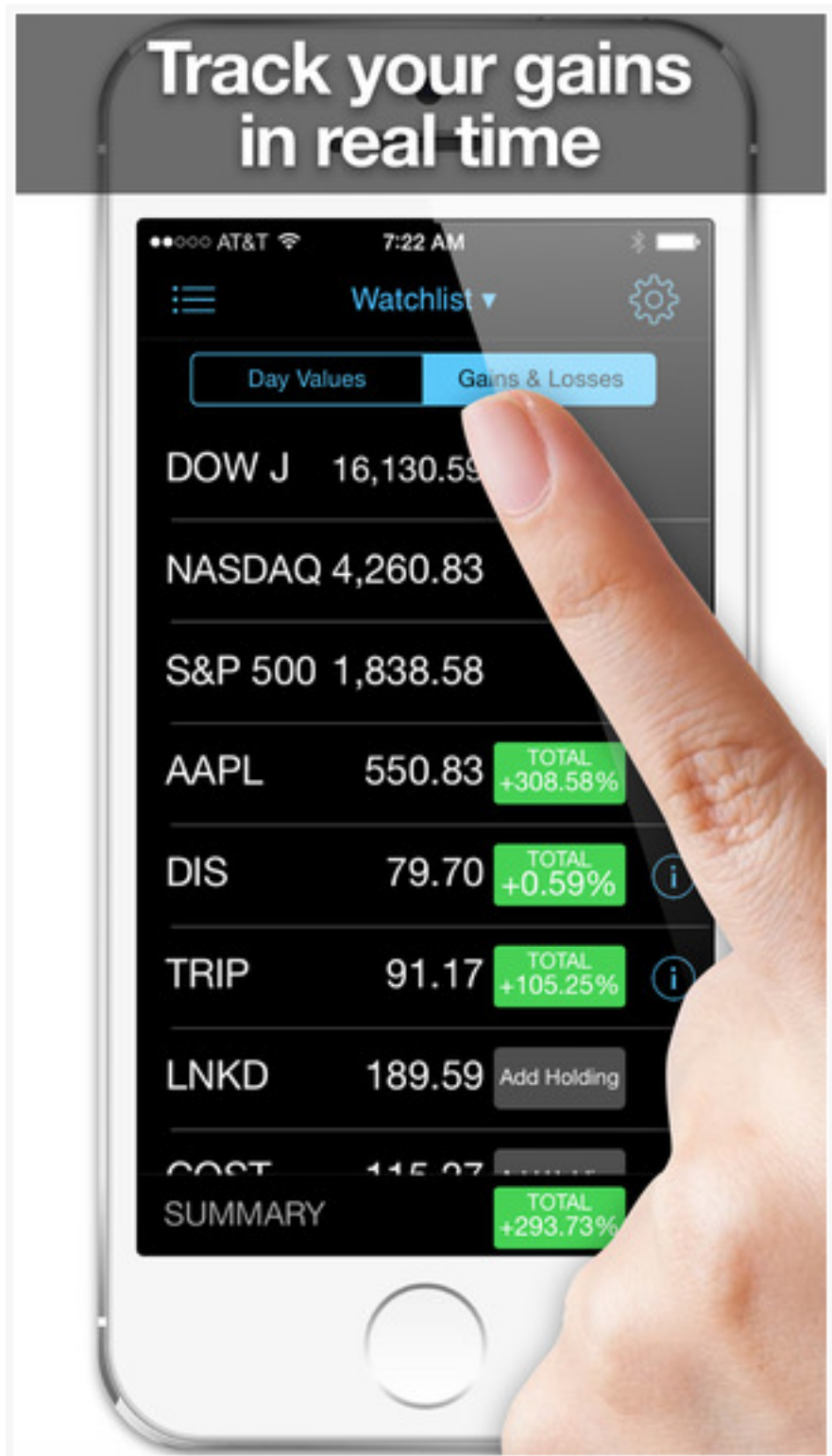
#Citation:

#"Stock Market HD: Stocks & Shares." iTunes Preview.

#Apple Inc, n.d. Web. 10 May 2016.

#<<https://itunes.apple.com/us/app/stock-market-hd-stocks-shares/id469405092?mt=8>>.

Out [14]:



In [14]: #Image14-c


```
#Citation:  
#The Best Way to Predict the Future Is to Invent It. - Alan Kay.  
# Digital image. Pixteller. Pixteller, n.d. Web. 9 May 2016.  
# <http://www.pixteller.com/img/3079>.
```

Out[14]:



1.5.3 15. Improvement

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:

- Are there further improvements that could be made on the algorithms or techniques you used in this project?
- Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?
- If you used your final solution as the new benchmark, do you think an even better solution exists?

Some of the further improvements that could be made on the algorithms or techniques used in this project are decreasing having a number of activation functions implemented along with options for crossvalidation sets to achieve more accuracy on the dataset. [Stock Prediction using Artificial Neural Networks](#).

Other algorithms that I did not implement but consider useful are: **Genetic algorithms**.

For example, a trading rule may involve the use of parameters like Moving Average Convergence-Divergence (MACD), Exponential Moving Average (EMA) and Stochastics. A genetic algorithm would then input values into these parameters with the goal of maximizing net profit. Over time, small changes are introduced and those that make a desirably impact are retained for the next generation.

There are three types of genetic operations that can then be performed:

- Crossovers represent the reproduction and biological crossover seen in biology, whereby a child takes on certain characteristics of its parents.
- Mutations represent biological mutation and are used to maintain genetic diversity from one generation of a population to the next by introducing random small changes.
- Selections are the stage at which individual genomes are chosen from a population for later breeding (recombination or crossover).

These three operators are then used in a five-step process:

1st Initialize a random population, where each chromosome is n-length, with n being the number of parameters. That is, a random number of parameters are established with n elements each.

2nd Select the chromosomes, or parameters, that increase desirable results (presumably net profit).

3rd Apply mutation or crossover operators to the selected parents and generate an offspring.

4th Recombine the offspring and the current population to form a new population with the selection operator.

5th Repeat steps two to four.

Over time, this process will result in increasingly favorable chromosomes (or, parameters) for use in a trading rule. The process is then terminated when a stopping criteria is met, which can include running time, fitness, number of generations or other criteria. For more on MACD (Moving average convergence divergence), read [Trading The MACD Divergence](#)

Source: * [Using Genetic Algorithms To Forecast Financial Markets] (<http://www.investopedia.com/articles/financial-theory/11/using-genetic-algorithms-forecast-financial-markets.asp#ixzz473WOYOWV>)

A better solution can be adding more parameters to improve the model. Some parameters that can improve the accuracy of the model are [Eight Factors that Influence Day Trading](#). Some ways to measure those eight factors that Influence Day Trading could be the following:

Factor#1, Overseas Market/Economic Action: Stock prices worldwide. [Find Information About Pre-And After-Hours Trading](#). If you see major negative activity in a “foreign” market that affects your sector, the prices of a stocks in that sector could fall. We could assign Factor#1 binary values.

Factor#2, Economic Data: This is mostly expressed by “interest rates”. For example, as interest rates in the U.K. rise, investors in that market may flee for better opportunities. Often, U.S. stocks will reap the benefit. [How Interest Rates affect the Stock Market](#). Factor#2 can be measured using binary values.

Factor#3, Futures Data: [Index futures](#) cover the major market indexes. They start trading before the stock market and are a very good indicator of what the stock market opening will look like. The reason for this is that index futures prices are closely linked with the actual level of the Dow Jones industrial average (DJIA). [“Are ETF Futures The Wave of The Future”](#). In other words Factor#3 can be measured by using the DJIA (a negative/positive numeric value/percentage)

Factor#4, Buying At the Open: We can consider the difference between the opening stock prices and the average day stock price to estimate the maximum price of the stock. We can measure Factor#4 with a positive/negative percentage

Factor#5, Midday Trading Lull: When this happens, stocks can be purchased at a cheaper price at 1 p.m. than they could at, say, 11 a.m. Again, this is important to know, as this can affect both entry and exit points. Observing the stock price values at these hours of the day and compare it to the average day

stock price can help us to better estimate the minimum value of the stock during the day. Factor#5 can be measured with a positive/negative percentage

Factor#6, Analyst Upgrades/Downgrades: We could classify the words from comments of top analysts. If an analyst mentions more words that backup Upgrades vs Downgrades we give the company a score of 1. In the opposite case we give it a score of 0. We count the sets of 1s and divide it by the number of analyst who reviewed the company. The highest score that could be achieved is 1. Then we will have to measure the correlation between the scores and the stock price value along different days. This way we can make it possible for the analysts to contribute in the prediction of stock prices. [What You Need to Know About Financial Analysts](#). Factor#6 can be measured using binary values.

Factor#7, Web-Related Articles: We can do the same than what we did for Factor#6 but only if the websites are reliable. First we should search which websites have a significant (massive) number of readers/viewers who trade in the Stock Market. Factor#7 will be measuring using binary values.

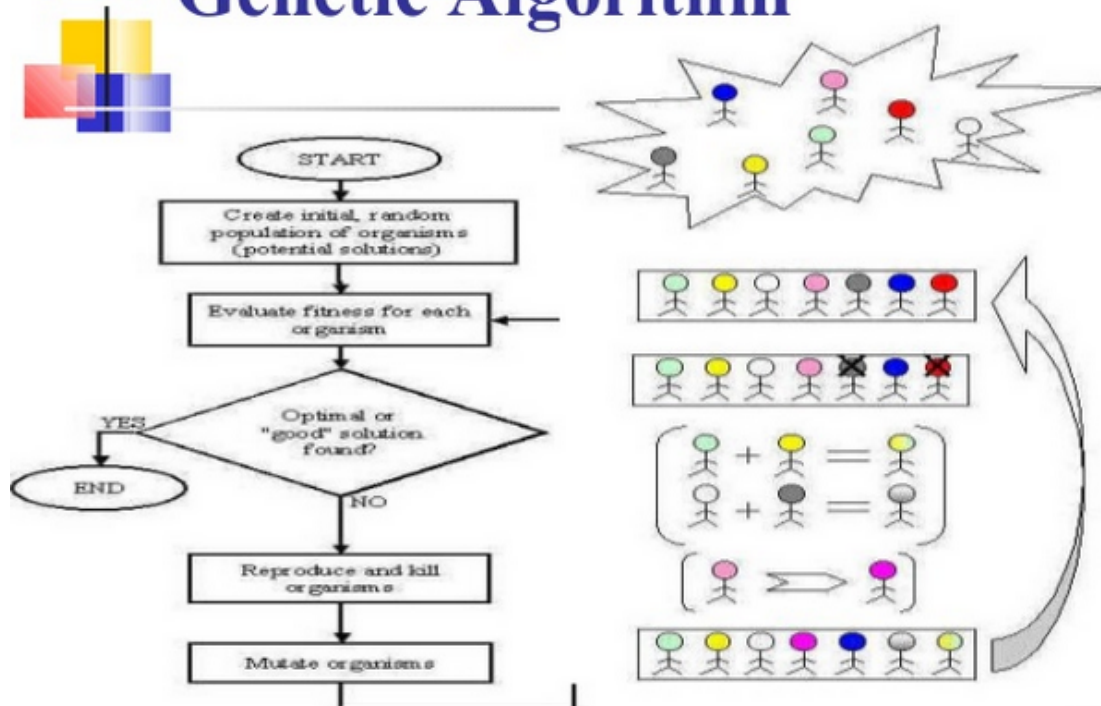
Factor#8, Friday Trading: Factor#8 can be helpful if we find that there is a correlation between the closing prices of Friday (numeric values) and the Analyst Upgrades/Downgrades during the weekend (binary values). Factor#8 can be measured by using that correlation. The [correlation coefficient](#), or Pearson product-moment correlation coefficient (PMCC) is a numerical value between -1 and 1 that expresses the strength of the linear relationship between two variables. When r is closer to 1 it indicates a strong positive relationship. This might be helpful to predict the opening stock prices on Mondays.

In [15]: *#Image15*

```
#Citation:  
#Tief, Amr Abd Ella, Allam Sheahata Hassanien Allab, and Abdullah Shoukry Nagaty.  
#"Stock Market Analysis Using Ga and Neural Network." SlideShare.  
#LinkedIn Corporation, 24 Feb. 2016. Web. 9 May 2016.  
#<http://www.slideshare.net/Amr_abd_ellatief/stock-market-  
# analysis-using-ga-and-neural-network>.
```

Out [15]:

Genetic Algorithm



<http://ib-poland.virtualave.net/ee/genetic1/3geneticalgorithms.htm>

10