CSCI-E63 Final Project Report
Tensorflow for Algorithmic Trading

Summary:
One of the hottest areas in finance these days is Algorithmic Trading, which uses artificial intelligence to sift through massive troves of data to identify signals that humans can't see. Algorithmic trading (automated trading, black-box trading, or simply algo-trading) is the process of using computers programmed to follow a defined set of instructions for placing a trade in order to generate profits at a speed and frequency that is impossible for a human trader. It uses natural-language processing to find keywords like company names, and measures when a story is rising up the media food chain, such as from blogs to newswires, to indicate that it may be important enough to act on. The goal of the project is to use sentiment analysis on news data to predict stock price.

Specific areas covered in the project:
- Retrieving data from an API
- Reading JSON files into Python Pandas Data Frame.
- Data analysis techniques to clean-up and prepare malformed data.
- Serializing and De-Serializing Python object structures using Pickle module.
- Python Natural Language Tool Kit (NLTK).
- Neural Network setup and Hyper-parameter tuning.
- Visualization

Problem Statement:
Create a Deep Neural Network that analyzes news feeds for a stock and makes price predictions based on the sentiment.

Current Setup and Future Work:
This project currently runs on static news articles from 2007-2016 made available by New York Times and makes predictions on the price of the Dow Jones Industrial Average (DJI). The Jupyter Notebook downloads the data sets for you using an API call. The DJI open and close prices are used from a csv provided. Future work involves integrating a dynamic feed of news articles from multiple sources and parametrizing the stock to be analyzed with real time price information.

Results:
The predicted prices follow the known prices very closely. With a little more optimization in the network architecture, including using different optimizers or loss functions, it seems possible to tune this model to predict with market-worthy accuracy.

Detailed implementation for reference and future work:
https://github.com/llSourcell/Stock_Market_Prediction

YouTube URL of 2 minute presentation: https://youtu.be/ZtwWGCC7z64

YouTube URL of full presentation: https://youtu.be/eGu36PADdzc

CSCI-E63 Final Project Report
Tensorflow for Algorithmic Trading

→ Setting up system on macOS

Follow instructions from: https://www.tensorflow.org/install/install_mac

For Installing with Anaconda:
Follow the instructions on the Anaconda download site to download and install Anaconda

```
[karanbhandarkar@Karans-MacBook-Air:~$ python -V
 Python 3.6.3 :: Anaconda, Inc.
```

Check version of pip3. Pip3 version 8.1+ is recommended.

```
[karanbhandarkar@Karans-MacBook-Air:~$ pip3 -V
pip 9.0.1 from /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages (python 3.6)
```

Create a conda env by issuing the following command: `condo create –n tensorflow python=3.5.2`

```
[karanbhandarkar@Karans-MacBook-Air:~$ conda create -n tensorflow python=3.5.2
Fetching package metadata ...........
Solving package specifications: .

Package plan for installation in environment /anaconda3/envs/tensorflow:

The following NEW packages will be INSTALLED:

    ca-certificates: 2017.08.26-ha1e5d58_0
    certifi:         2017.7.27.1-py35h0fdde5e_0
    openssl:         1.0.2m-h86d3e6a_0
    pip:             9.0.1-py35h33ce766_4
    python:          3.5.2-0
    readline:        6.2-2
    setuptools:      36.5.0-py35h52cde6a_0
    sqlite:          3.13.0-0
    tk:              8.5.18-0
    wheel:           0.29.0-py35ha7aa5c4_1
    xz:              5.2.3-h0278029_2
    zlib:            1.2.11-hf3cbc9b_2

Proceed ([y]/n)? y

ca-certificate 100% |################################################################################| Time: 0:00:00   2.47 MB/s
readline-6.2-2 100% |################################################################################| Time: 0:00:00   3.62 MB/s
sqlite-3.13.0- 100% |################################################################################| Time: 0:00:00   3.58 MB/s
tk-8.5.18-0.ta 100% |################################################################################| Time: 0:00:00   3.74 MB/s
xz-5.2.3-h0278 100% |################################################################################| Time: 0:00:00   4.41 MB/s
zlib-1.2.11-hf 100% |################################################################################| Time: 0:00:00   4.16 MB/s
openssl-1.0.2m 100% |################################################################################| Time: 0:00:00   3.75 MB/s
python-3.5.2-0 100% |################################################################################| Time: 0:00:03   3.64 MB/s
certifi-2017.7 100% |################################################################################| Time: 0:00:00   3.95 MB/s
setuptools-36. 100% |################################################################################| Time: 0:00:00   3.67 MB/s
wheel-0.29.0-p 100% |################################################################################| Time: 0:00:00   5.15 MB/s
pip-9.0.1-py35 100% |################################################################################| Time: 0:00:00   3.77 MB/s
#
# To activate this environment, use:
# > source activate tensorflow
```

Activate the condo env by issuing the following command(your prompt should change):
`source activate tensorflow`

Issue the following command to install TensorFlowin inside your conda env to install CPU-only version:

CSCI-E63 Final Project Report
Tensorflow for Algorithmic Trading

```
(tensorflow) karanbhandarkar@Karans-MacBook-Air:~$ pip install --ignore-installed --upgrade tensorflow
Collecting tensorflow
  Downloading tensorflow-1.4.0-cp35-cp35m-macosx_10_11_x86_64.whl (38.8MB)
    100% |████████████████████████████████| 38.8MB 29kB/s
Collecting tensorflow-tensorboard<0.5.0,>=0.4.0rc1 (from tensorflow)
  Using cached tensorflow_tensorboard-0.4.0rc2-py3-none-any.whl
Collecting six>=1.10.0 (from tensorflow)
  Using cached six-1.11.0-py2.py3-none-any.whl
Collecting protobuf>=3.3.0 (from tensorflow)
  Using cached protobuf-3.4.0-py2.py3-none-any.whl
Collecting enum34>=1.1.6 (from tensorflow)
  Using cached enum34-1.1.6-py3-none-any.whl
Collecting numpy>=1.12.1 (from tensorflow)
  Downloading numpy-1.13.3-cp35-cp35m-macosx_10_6_intel.macosx_10_9_intel.macosx_10_9_x86_64.macosx_10_10_intel.macosx_10_10_x86_64.whl (4.5M
B)
    100% |████████████████████████████████| 4.5MB 268kB/s
Collecting wheel>=0.26 (from tensorflow)
  Using cached wheel-0.30.0-py2.py3-none-any.whl
Collecting bleach==1.5.0 (from tensorflow-tensorboard<0.5.0,>=0.4.0rc1->tensorflow)
  Using cached bleach-1.5.0-py2.py3-none-any.whl
Collecting html5lib==0.9999999 (from tensorflow-tensorboard<0.5.0,>=0.4.0rc1->tensorflow)
  Using cached html5lib-0.9999999.tar.gz
Collecting markdown>=2.6.8 (from tensorflow-tensorboard<0.5.0,>=0.4.0rc1->tensorflow)
  Using cached Markdown-2.6.9.tar.gz
Collecting werkzeug>=0.11.10 (from tensorflow-tensorboard<0.5.0,>=0.4.0rc1->tensorflow)
  Using cached Werkzeug-0.12.2-py2.py3-none-any.whl
Collecting setuptools (from protobuf>=3.3.0->tensorflow)
  Downloading setuptools-36.7.0-py2.py3-none-any.whl (482kB)
    100% |████████████████████████████████| 491kB 1.6MB/s
Building wheels for collected packages: html5lib, markdown
  Running setup.py bdist_wheel for html5lib ... done
  Stored in directory: /Users/karanbhandarkar/Library/Caches/pip/wheels/6f/85/6c/56b8e1292c6214c4eb73b9dda50f53e8e977bf65989373c962
  Running setup.py bdist_wheel for markdown ... done
  Stored in directory: /Users/karanbhandarkar/Library/Caches/pip/wheels/bf/46/10/c93e17ae86ae3b3a919c7b39dad3b5ccf09aeb066419e5c1e5
Successfully built html5lib markdown
Installing collected packages: six, html5lib, bleach, markdown, numpy, setuptools, protobuf, wheel, werkzeug, tensorflow-tensorboard, enum34,
 tensorflow
Successfully installed bleach-1.5.0 enum34-1.1.6 html5lib-0.9999999 markdown-2.6.9 numpy-1.13.3 protobuf-3.4.0 setuptools-36.7.0 six-1.11.0 t
ensorflow-1.4.0 tensorflow-tensorboard-0.4.0rc2 werkzeug-0.12.2 wheel-0.30.0
```

`pip install --ignore-installed --upgrade tensorflow`

Validate the installation:

Start a terminal.

Activate the created the Anaconda env.

Invoke python.

Import the tensorflow package.

```
(tensorflow) karanbhandarkar@Karans-MacBook-Air:~$ python
Python 3.5.2 |Continuum Analytics, Inc.| (default, Jul  2 2016, 17:52:12)
[GCC 4.2.1 Compatible Apple LLVM 4.2 (clang-425.0.28)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>>
```

Install Jupyter Notebook in your Anaconda environment using command:

`conda install jupyter`

Installation ends as:

```
jupyter_client 100% |################################################################################| Time: 0:00:00   3.91 MB/s
nbformat-4.4.0 100% |################################################################################| Time: 0:00:00   3.67 MB/s
prompt_toolkit 100% |################################################################################| Time: 0:00:00   3.98 MB/s
ipython-6.1.0- 100% |################################################################################| Time: 0:00:00   3.82 MB/s
nbconvert-5.3. 100% |################################################################################| Time: 0:00:00   3.88 MB/s
ipykernel-4.6. 100% |################################################################################| Time: 0:00:00   4.77 MB/s
jupyter_consol 100% |################################################################################| Time: 0:00:00   4.30 MB/s
notebook-5.2.1 100% |################################################################################| Time: 0:00:01   3.61 MB/s
qtconsole-4.3. 100% |################################################################################| Time: 0:00:00   3.52 MB/s
widgetsnbexten 100% |################################################################################| Time: 0:00:00   3.65 MB/s
ipywidgets-7.0 100% |################################################################################| Time: 0:00:00   4.58 MB/s
jupyter-1.0.0- 100% |################################################################################| Time: 0:00:00   3.88 MB/s
    If this is your first install of dbus, automatically load on login with:
        mkdir -p ~/Library/LaunchAgents
        cp /anaconda3/envs/tensorflow/org.freedesktop.dbus-session.plist ~/Library/LaunchAgents/
        launchctl load -w ~/Library/LaunchAgents/org.freedesktop.dbus-session.plist

    If this is an upgrade and you already have the org.freedesktop.dbus-session.plist loaded:
        launchctl unload -w ~/Library/LaunchAgents/org.freedesktop.dbus-session.plist
        cp /anaconda3/envs/tensorflow/org.freedesktop.dbus-session.plist ~/Library/LaunchAgents/
        launchctl load -w ~/Library/LaunchAgents/org.freedesktop.dbus-session.plist

(tensorflow) karanbhandarkar@Karans-MacBook-Air:~$
```

Install matplotlib using command: `pip install matplotlib`

```
(tensorflow) karanbhandarkar@Karans-MacBook-Air:~$ pip install matplotlib
Collecting matplotlib
  Downloading matplotlib-2.1.0-cp35-cp35m-macosx_10_6_intel.macosx_10_9_intel.macosx_10_9_x86_64.macosx_10_10_intel.macosx_10_10_x86_64.whl (13.2MB
)
    100% |████████████████████████████████| 13.2MB 98kB/s
Requirement already satisfied: six>=1.10 in /anaconda3/envs/tensorflow/lib/python3.5/site-packages (from matplotlib)
Requirement already satisfied: python-dateutil>=2.0 in /anaconda3/envs/tensorflow/lib/python3.5/site-packages (from matplotlib)
Collecting pytz (from matplotlib)
  Downloading pytz-2017.3-py2.py3-none-any.whl (511kB)
    100% |████████████████████████████████| 512kB 1.4MB/s
Collecting pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 (from matplotlib)
  Downloading pyparsing-2.2.0-py2.py3-none-any.whl (56kB)
    100% |████████████████████████████████| 61kB 2.7MB/s
Requirement already satisfied: numpy>=1.7.1 in /anaconda3/envs/tensorflow/lib/python3.5/site-packages (from matplotlib)
Collecting cycler>=0.10 (from matplotlib)
  Downloading cycler-0.10.0-py2.py3-none-any.whl
Installing collected packages: pytz, pyparsing, cycler, matplotlib
Successfully installed cycler-0.10.0 matplotlib-2.1.0 pyparsing-2.2.0 pytz-2017.3
```

Start Jupyter Notebook and run `NewsFeedStockPrediction.ipynb`

→ Import Required Modules

```
# First time installs:
# sudo pip install requests
# sudo pip install newsapi
# sudo pip install pandas
# sudo pip install nltk
# sudo pip install twython
# sudo pip install h5py

import sys, csv, json
import requests
from newsapi.articles import Articles
from newsapi.sources import Sources
import numpy as np
import csv, json
import pandas as pd
from nltk.classify import NaiveBayesClassifier
from nltk.corpus import subjectivity
from nltk.sentiment import SentimentAnalyzer
from nltk.sentiment.util import *
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import unicodedata
import math
import h5py
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.python.framework import ops
```

The packages mentioned above need to be installed as shown in the commented out section.

CSCI-E63 Final Project Report
Tensorflow for Algorithmic Trading

→ Collect NY Times data

Documentation for this can be found at:

https://github.com/SlapBot/newsapi
https://github.com/llSourcell/Stock_Market_Prediction/blob/master/Collecting%20NYTimes%20Data.py

The implementation is done as below. This retrieves news articles from the NY Times API for all months from 2007 to 2016.

```
class ArchiveAPI(object):
    def __init__(self, key=None):
        self.key = key
        self.root = 'http://api.nytimes.com/svc/archive/v1/{}/{}.json?api-key={}'
        if not self.key:
            nyt_dev_page = 'http://developer.nytimes.com/docs/reference/keys'
            exception_str = 'Warning: API Key required. Please visit {}'
            raise NoAPIKeyException(exception_str.format(nyt_dev_page))

    def query(self, year=None, month=None, key=None,):
        """
        Calls the archive API and returns the results as a dictionary.
        :param key: Defaults to the API key used to initialize the ArchiveAPI class.
        """
        if not key:
            key = self.key

        if (year < 1882) or not (0 < month < 13):
            # currently the Archive API only supports year >= 1882
            exception_str = 'Invalid query: See http://developer.nytimes.com/archive_api.json'
            raise InvalidQueryException(exception_str)

        url = self.root.format(year, month, key)
        r = requests.get(url)
        return r.json()


api = ArchiveAPI('0ba6dc04a8cb44e0a890c00df88c393a')
years = [2016, 2015, 2014, 2013, 2012, 2011, 2010, 2009, 2008, 2007]
months = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

for year in years:
    for month in months:
        mydict = api.query(year, month)
        file_str = '/Users/karanbhandarkar/Projects/PythonProjects/NewsFeedStockPrediction/Data' + str(year) + '-' + '{
        with open(file_str, 'w') as fout:
            try:
                json.dump(mydict, fout)
            except:
                pass
        fout.close()
```

This process takes a while to run. Dataset is 2.2GB(~400MB after compression) and so was not submitted separately.

CSCI-E63 Final Project Report
Tensorflow for Algorithmic Trading

→ Preparing Stock Price

This process uses a CSV downloaded from the internet(https://github.com/llSourcell/Stock_Market_Prediction/blob/master/Data/DJIA%20indices%20data.csv).
The file(DJIA_indices_data.csv) has been submitted as part of the data of the implemented project.
To make this project trading-ready, this should be replaced by an API to retrieve data from Quandl in real-time.

```
with open('/Users/karanbhandarkar/Projects/PythonProjects/NewsFeedStockPrediction/Data/DJIA_indices_data.csv', 'r',enco
    spamreader = csv.reader(csvfile, delimiter=',')
    # Converting the csv file reader to a lists
    data_list = list(spamreader)
```

Separating header from the data

```
header = data_list[0]
data_list = data_list[1:]

data_list = np.asarray(data_list)
```

```
print(data_list)
```

```
[['2016-12-30' '19833.169922' '19852.550781' ..., '19762.599609'
  '271910000' '19762.599609']
 ['2016-12-29' '19835.460938' '19878.439453' ..., '19819.779297'
  '172040000' '19819.779297']
 ['2016-12-28' '19964.310547' '19981.109375' ..., '19833.679688'
  '188350000' '19833.679688']
 ...,
 ['2007-01-04' '12473.160156' '12510.410156' ..., '12480.69043' '259060000'
  '12480.69043']
 ['2007-01-03' '12459.540039' '12580.349609' ..., '12474.519531'
  '327200000' '12474.519531']
 ['2006-12-29' '12500.480469' '12526.030273' ..., '12463.150391'
  '161560000' '12463.150391']]
```

Selecting date and close value and adj close for each day
Volume will be added in the next model

```
selected_data = data_list[:, [0, 4, 6]]
```

Convert it into dataframe
index = date

```
df = pd.DataFrame(data=selected_data[0:,1:],
             index=selected_data[0:,0],
                                columns=['close', 'adj close'],
                                        dtype='float64')
```

```
print (df.tail())
```

```
                  close     adj close
2007-01-08  12423.490234  12423.490234
2007-01-05  12398.009766  12398.009766
2007-01-04  12480.690430  12480.690430
2007-01-03  12474.519531  12474.519531
2006-12-29  12463.150391  12463.150391
```

Interpolating data

```
df1 = df
idx = pd.date_range('12-29-2006', '12-31-2016')
df1.index = pd.DatetimeIndex(df1.index)
df1 = df1.reindex(idx, fill_value=np.NaN)
# df1.count() # gives 2518 count
interpolated_df = df1.interpolate() # Fill in the gap
interpolated_df.count() # gives 3651 count
```

```
close        3656
adj close    3656
dtype: int64
```

```
print (df1.head(25))
```

```
                  close     adj close
2006-12-29  12463.150391  12463.150391
2006-12-30           NaN           NaN
2006-12-31           NaN           NaN
2007-01-01           NaN           NaN
2007-01-02           NaN           NaN
2007-01-03  12474.519531  12474.519531
2007-01-04  12480.690430  12480.690430
2007-01-05  12398.009766  12398.009766
2007-01-06           NaN           NaN
2007-01-07           NaN           NaN
2007-01-08  12423.490234  12423.490234
2007-01-09  12416.599609  12416.599609
2007-01-10  12442.160156  12442.160156
2007-01-11  12514.980469  12514.980469
2007-01-12  12556.080078  12556.080078
2007-01-13           NaN           NaN
2007-01-14           NaN           NaN
2007-01-15           NaN           NaN
2007-01-16  12582.589844  12582.589844
2007-01-17  12577.150391  12577.150391
2007-01-18  12567.929688  12567.929688
2007-01-19  12565.530273  12565.530273
2007-01-20           NaN           NaN
2007-01-21           NaN           NaN
2007-01-22  12477.160156  12477.160156
```

Interpolating is a useful technique while dealing with time series data. For this project, we need our data to be continuous. As you can see above, there are dates that don't contain any data. Interpolation is used to fill in data for those dates using data from available dates.

```
# Removing extra date rows added in data for calculating interpolation
interpolated_df = interpolated_df[3:]
```

```
print (interpolated_df.head())
```

```
                  close     adj close
2007-01-01  12469.971875  12469.971875
2007-01-02  12472.245703  12472.245703
2007-01-03  12474.519531  12474.519531
2007-01-04  12480.690430  12480.690430
2007-01-05  12398.009766  12398.009766
```

→ Merging NY Times Date

Now we merge the articles and price data frames created above to prepare the dataset we're going to work with.

Function to parse and convert date format
Try 2 formats for date or raise error

```
date_format = ["%Y-%m-%dT%H:%M:%SZ", "%Y-%m-%dT%H:%M:%S+%f"]

def try_parsing_date(text):
    for fmt in date_format:
        try:
            return datetime.strptime(text, fmt).strftime('%Y-%m-%d')
        except ValueError:
            pass
    raise ValueError('no valid date format found')
```

```
years = [2016, 2015, 2014, 2013, 2012, 2011, 2010, 2009, 2008, 2007]
months = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
dict_keys = ['pub_date', 'headline'] #, 'lead_paragraph']
articles_dict = dict.fromkeys(dict_keys)
```

Filtering to read only the following categories of news

```
# Filtering list for type_of_material
type_of_material_list = ['blog', 'brief', 'news', 'editorial', 'op-ed', 'list','analysis']
# Filtering list for section_name
section_name_list = ['business', 'national', 'world', 'u.s.' , 'politics', 'opinion', 'tech', 'science',  'health']
news_desk_list = ['business', 'national', 'world', 'u.s.' , 'politics', 'opinion', 'tech', 'science',  'health', 'forei
```

Data from the NY Times API is categorized. Using the above defined lists, we are going to work with data that is categorized into these sections.

```
current_date = '2016-10-01'
from datetime import datetime

current_article_str = ''
```

Adding article column to dataframe

```
interpolated_df["articles"] = ''
count_articles_filtered = 0
count_total_articles = 0
count_main_not_exist = 0
count_unicode_error = 0
count_attribute_error = 0
```

Refer to the ipynb for detailed implementation.

See a summary of our prepared dataset.

```
print (count_articles_filtered)
print (count_total_articles)
print (count_main_not_exist)
print (count_unicode_error)
```

```
461738
1248084
16
0
```

We save the prepared dataset as a pickle file. The pickle module implements a fundamental, but powerful algorithm for serializing and de-serializing a Python object structure.

```
# Saving the data as pickle file
interpolated_df.to_pickle('/Users/karanbhandarkar/Projects/PythonProjects/NewsFeedStockPrediction/Data/pickled_ten_year_


# Save pandas frame in csv form
interpolated_df.to_csv('/Users/karanbhandarkar/Projects/PythonProjects/NewsFeedStockPrediction/Data/sample_interpolated_
                       sep='\t', encoding='utf-8')



# Reading the data as pickle file
dataframe_read = pd.read_pickle('/Users/karanbhandarkar/Projects/PythonProjects/NewsFeedStockPrediction/Data/pickled_te
```

The generated file along with an easy-to-read csv version have been submitted as part of data of the implemented project.

CSCI-E63 Final Project Report
Tensorflow for Algorithmic Trading

→ Deep Neural Network

Read the pickle file created in the previous step and created a dataframe for our model.
We are interested in seeing the impact of news articles on the adjusted closing price.

```
df_stocks = pd.read_pickle('/Users/karanbhandarkar/Projects/PythonProjects/NewsFeedStockPrediction/Data/pickled_ten_yea
```

```
print (df_stocks.head())
```

```
                  close     adj close  \
2007-01-01  12469.971875  12469.971875
2007-01-02  12472.245703  12472.245703
2007-01-03  12474.519531  12474.519531
2007-01-04  12480.690430  12480.690430
2007-01-05  12398.009766  12398.009766

                                                articles
2007-01-01  . Estimates of Iraqi Civilian Deaths. Romania ...
2007-01-02  . For Dodd, Wall Street Looms Large. Ford's Lo...
2007-01-03  . Ethics Changes Proposed for House Trips, K S...
2007-01-04  . I Feel Bad About My Face. Bush Recycles the ...
2007-01-05  . Macworld Bingo. Anti-Surge Protests Against ...
```

```
df_stocks['prices'] = df_stocks['adj close'].apply(np.int64)
```

```
# selecting the prices and articles
df_stocks = df_stocks[['prices', 'articles']]
```

```
df_stocks.head()
```

	prices	articles
2007-01-01	12469	. Estimates of Iraqi Civilian Deaths. Romania ...
2007-01-02	12472	. For Dodd, Wall Street Looms Large. Ford's Lo...
2007-01-03	12474	. Ethics Changes Proposed for House Trips, K S...
2007-01-04	12480	. I Feel Bad About My Face. Bush Recycles the ...
2007-01-05	12398	. Macworld Bingo. Anti-Surge Protests Against ...

At this point, steps, like removing trailing punctuations, are taken to clean up the data.

The NLTK package's SentimentIntensityAnalyzer is applied to the news articles to generate sentiment scores. Positive values are positive valence, negative value are negative valence.

```
sid = SentimentIntensityAnalyzer()
for date, row in df_stocks.T.iteritems():
    try:
        sentence = unicodedata.normalize('NFKD', df_stocks.loc[date, 'articles'])
        ss = sid.polarity_scores(sentence)
        df.set_value(date, 'compound', ss['compound'])
        df.set_value(date, 'neg', ss['neg'])
        df.set_value(date, 'neu', ss['neu'])
        df.set_value(date, 'pos', ss['pos'])
    except TypeError:
        print (df_stocks.loc[date, 'articles'])
        print (date)
```

```
df.head()
```

	prices	compound	neg	neu	pos
2007-01-01	12469	-0.9881	0.176	0.723	0.102
2007-01-02	12472	-0.397	0.117	0.778	0.105
2007-01-03	12474	-0.9994	0.196	0.749	0.055
2007-01-04	12480	-0.9975	0.112	0.835	0.053
2007-01-05	12398	-0.9882	0.118	0.8	0.082

Going to use numpy normalize to replace this formula soon

```
datasetNorm = (df - df.mean()) / (df.max() - df.min())
datasetNorm.reset_index(inplace=True)
del datasetNorm['index']
datasetNorm['next_prices'] = datasetNorm['prices'].shift(-1)
datasetNorm.head(5)
```

	prices	compound	neg	neu	pos	next_prices
0	-0.085546	-0.0671946	0.133644	-0.0519895	0.0656875	-0.085323
1	-0.085323	0.228814	-0.0932795	0.0108676	0.0820809	-0.085174
2	-0.085174	-0.0728534	0.210567	-0.0222752	-0.191143	-0.084727
3	-0.084727	-0.0719019	-0.11251	0.0760105	-0.202072	-0.090834
4	-0.090834	-0.0672447	-0.0894333	0.0360105	-0.0436022	-0.090239

Neural networks can have many hyperparameters, including those which specify the structure of the network itself and those which determine how the network is trained.

The gradient descent hyperparameters are defined as:

```
num_epochs = 1000

batch_size = 1

total_series_length = len(datasetNorm.index)

truncated_backprop_length = 3 #The size of the sequence

state_size = 12 #The number of neurons

num_features = 4
num_classes = 1 #[1,0]

num_batches = total_series_length//batch_size//truncated_backprop_length

min_test_size = 100

print('The total series length is: %d' %total_series_length)
print('The current configuration gives us %d batches of %d observations each one looking %d steps in the past'
      %(num_batches,batch_size,truncated_backprop_length))
```

```
The total series length is: 3653
The current configuration gives us 1217 batches of 1 observations each one looking 3 steps in the past
```

The model hyperparameters are defined as:

Weights and biases
Because is a 3 layer net:
Input
Hidden Recurrent layer
Output
We need 2 pairs of W and b

```
W2 = tf.Variable(initial_value=np.random.rand(state_size,num_classes),dtype=tf.float32)
b2 = tf.Variable(initial_value=np.random.rand(1,num_classes),dtype=tf.float32)
```

The dataset created for the model needs to be split into training data and test data. This is done as:

Train-Test split

```
datasetTrain = datasetNorm[datasetNorm.index < num_batches*batch_size*truncated_backprop_length]


for i in range(min_test_size,len(datasetNorm.index)):

    if(i % truncated_backprop_length*batch_size == 0):
        test_first_idx = len(datasetNorm.index)-i
        break

datasetTest =  datasetNorm[datasetNorm.index >= test_first_idx]
```

```
xTrain = datasetTrain[['prices','neu','neg','pos']].as_matrix()
yTrain = datasetTrain['next_prices'].as_matrix()
```

```
xTrain.shape
```

```
(3651, 4)
```

```
xTest = datasetTest[['prices','neu','neg','pos']].as_matrix()
yTest = datasetTest['next_prices'].as_matrix()
```

```
yTest.shape
```

```
(102,)
```

We perform a forward pass to calculate the output and a backward pass to calculate the gradient, while tracing back to the input.

Forward pass - Unroll the cell

```
cell = tf.contrib.rnn.BasicLSTMCell(num_units=state_size)

states_series, current_state = tf.nn.dynamic_rnn(cell=cell,inputs=batchX_placeholder,dtype=tf.float32)
```

```
states_series = tf.transpose(states_series,[1,0,2])
```

Backward pass - Output

```
last_state = tf.gather(params=states_series,indices=states_series.get_shape()[0]-1)
last_label = tf.gather(params=labels_series,indices=len(labels_series)-1)
```

CSCI-E63 Final Project Report
Tensorflow for Algorithmic Trading

Backward pass - Output

```
weight = tf.Variable(tf.truncated_normal([state_size,num_classes]))
bias = tf.Variable(tf.constant(0.1,shape=[num_classes]))
```

Backward pass - Output

```
prediction = tf.matmul(last_state,weight) + bias
prediction
```

```
<tf.Tensor 'add:0' shape=(?, 1) dtype=float32>
```

We use the root mean squared error as the loss function here and use an AdamOptimizer to train our model.

```
loss = tf.reduce_mean(tf.squared_difference(last_label,prediction))

train_step = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)
```

The model is trained and tested for the hyperparameters configured earlier:

```
loss_list = []
test_pred_list = []

with tf.Session() as sess:

    tf.global_variables_initializer().run()

    for epoch_idx in range(num_epochs):

        print('Epoch %d' %epoch_idx)

        for batch_idx in range(num_batches):
            start_idx = batch_idx * truncated_backprop_length
            end_idx = start_idx + truncated_backprop_length * batch_size


            batchX = xTrain[start_idx:end_idx,:].reshape(batch_size,truncated_backprop_length,num_features)
            batchY = yTrain[start_idx:end_idx].reshape(batch_size,truncated_backprop_length,1)

            #print('IDXs',start_idx,end_idx)
            #print('X',batchX.shape,batchX)
            #print('Y',batchX.shape,batchY)

            feed = {batchX_placeholder : batchX, batchY_placeholder : batchY}

            #UPDATE
            loss_list.append(_loss)



            if(batch_idx % 50 == 0):
                print('Step %d - Loss: %.6f' %(batch_idx,_loss))

    #TEST


    for test_idx in range(len(xTest) - truncated_backprop_length):

        testBatchX = xTest[test_idx:test_idx+truncated_backprop_length,:].reshape((1,truncated_backprop_length,num_feat
        testBatchY = yTest[test_idx:test_idx+truncated_backprop_length].reshape((1,truncated_backprop_length,1))


        #_current_state = np.zeros((batch_size,state_size))
        feed = {batchX_placeholder : testBatchX,
            batchY_placeholder : testBatchY}

        #Test_pred contains 'window_size' predictions, we want the last one
        _last_state,_last_label,test_pred = sess.run([last_state,last_label,prediction],feed_dict=feed)
        test_pred_list.append(test_pred[-1][0]) #The last one
```

The results for the final epoch:

```
Epoch 999
Step 0 - Loss: 0.000056
Step 50 - Loss: 0.000006
Step 100 - Loss: 0.000140
Step 150 - Loss: 0.000001
Step 200 - Loss: 0.000020
Step 250 - Loss: 0.000027
Step 300 - Loss: 0.000024
Step 350 - Loss: 0.000190
Step 400 - Loss: 0.000015
Step 450 - Loss: 0.000071
Step 500 - Loss: 0.000046
Step 550 - Loss: 0.000003
Step 600 - Loss: 0.000325
Step 650 - Loss: 0.000004
Step 700 - Loss: 0.000078
Step 750 - Loss: 0.000000
Step 800 - Loss: 0.000012
Step 850 - Loss: 0.000018
Step 850 - Loss: 0.000018
Step 900 - Loss: 0.000031
Step 950 - Loss: 0.000073
Step 1000 - Loss: 0.000001
Step 1050 - Loss: 0.000448
Step 1100 - Loss: 0.000002
Step 1150 - Loss: 0.000011
Step 1200 - Loss: 0.000014
```

Matplotlib is used to visualize the predictions against the closing prices as:

```
plt.figure(figsize=(21,7))
plt.plot(yTest,label='Price',color='blue')
plt.plot(test_pred_list,label='Predicted',color='red')
plt.title('Price vs Predicted')
plt.legend(loc='upper left')
plt.show()
```