Natural Language Processing for Stock Price Prediction

Baptiste Aubert supervised by Umut Simsekli

February 8, 2019

Abstract

The financial services industry is among the most competitive in the world as clients can access any provider from any place in the world. This results in a constant race for innovation in the whole range of financial services that are provided. The Financial Industry thus feeds itself from all the technological breakthrough that emerge in the society. Over the past few years it is thus no surprise that Machine Learning has made a sensational start in the industry. In this paper we will analyse the impact of a subset of this: the application of Natural Language Processing for stock price predictions.

Contents

1 Introduction **3**

2 10-K Reports, Rush for Gold? **4**
2.1 Presentation of the data 4
2.2 Acquisition of the data 4
2.3 Construction of our Scrapper 4
2.4 Loading and Processing of the Data to train word vectors 7
2.4.1 Skip-Gram Algorythm 7
2.4.2 Reducing the computational cost: the input of Morin and Bengio 8
2.4.3 Negative Sampling 8
2.5 Regressing the Stock Returns on Report release 17
2.5.1 Getting the Financial Data 17
2.5.2 Loading the Data 17
2.5.3 Building the X matrix 18
2.5.4 Building the dictionaries 18
2.5.5 Discussion of the Drivers 27

3 ANNEX **29**

4 References **33**

1 Introduction

As a mirror and a mean of human economy, finance shares very strong links with technological innovations. For centuries it used to provide security only to wealthy customers who could not offered keeping their money under the mattress. It then aimed at financing agriculture, trade and industry successively.
But technological innovations brought a whole new range of services in the financial industry. In the 1990s the mathematical modelisation of stock price trajectories and the digitalization of the world unabled a great expansion of the financial industry. More customers entered the market, more investment followed.

Those investments aims at making the industry more reliable and more accurate. Financial Markets are very much about providing clients with economic visions to navigate the market. It this later sector Machine Learning and Big Data are having a huge impact. Investment banks, hedge funds and start-ups lead massive investments in those areas. The most dazzling revolution has gone through computer vision. More and more market actors try to use satellite images to forecast economic indicators (using images of parking lots, oil tankers, flaring camps...) and trade ahead of them. Some players, like Kayrros in France became highly recognized leaders in this indstry, growing from a small corporates client portfolio to massive multi national companies, hedge funds and asset managers clients in a few years.

If Natural Language Processing techniques can be arguably considered lacking development to compute vision in the world of machine learning, they definitely are in the world of financial markets. The main underlying reason is data availability. While satellite images are readily available through providers like NASA or USGS, data for NLP signals generation is way more sparse.

This sparsity makes the analysis much harder when it comes to Natural Language Processing relatively to Computer Vision. Retail investors actually tend to discuss investment strategies on specialized forums (like Boursorama in France). Generating an aggregate sentiment is thus very hard. The knowledge of institutional investors behaviors could help tackle this issue. Institutional investors actually tend to subscribe to private and specialized news providers. They thus tend to trade based on the publication of some specific articles or news.

We can thus distinguish the providers in two categories. The first are the one dedicated to retail investors. They can be free or for sale. One would find Twitter, Les Echos or the FT in this category. Other providers focus exclusively on institutional investors (Investment Banks, Hedge Funds or Asset Managers). They are never free (up to several thousands euros per month and per person) and their impact on financial markets is substantial.

2 10-K Reports, Rush for Gold?

2.1 Presentation of the data

The American equities market regulator is known as the SEC. Every single company listed on the US market is therefore overseen by the SEC and have to produce an annual report known as the 10-K Report. The document is very focused on by analysts all over the world who try to derive financial indicators for the year ahead. Predicting its impact on the company share price therefore represent a big objective for investors.
This report is divided into 9 sections. Some of them convey way more sentiment than others that remain very standardised. Section 7 is of crucial interest. In section 7: "Management's Discussion and Analysis of Financial Condition and Results of Operations" the management discusses its view of the performance for the past year and provides guidance for the exercise on the year ahead.

2.2 Acquisition of the data

The reason why we decided to go with the above source of data is that it is public information. 10-K reports are actually readily available on a DataBase provided by the SEC. This DataBase is named "Electronic Data Gathering, Analysis, and Retrieval System". We will call it EGDAR DataBase.
Although freely available on the SEC website, the information is far from being standardised. Over the past few years those reports have gained additional attention from investors and one scrapper publicly available on GitHub has been developed. We nevertheless decided to go with our own version. This stands for two reasons. One, the existing version struggles to capture most of the data only achieving a 60% retrieval rate. It is actually a hard job to scrap those data because of the lack of standardization that exists in the structure of those reports. As we will see, we invested a significant amount of time developing an advanced methodology for scrapping those data. The second reason is that we want the DataBase where we store those reports to be scalable and very fast. We chose MongoDB to store those data and used PyMongo to make the python scrapper communicate with it.

2.3 Construction of our Scrapper

The scrapper that we built is coded in Python. It fetches the data from the SEC website and retrieve the douments from their assoicated ticker. This ticker is provided by the SEC on the EDGAR DataBase. We display the correspondance table that we used to guide our scrapper in this work.

```
// SEC EDGAR Scrapper
1.1) Load Correspondance Table Between Company Name and SEC Ticker

crs = open("companylist.txt", "r")
companies_dict = {}
for raw in crs:
```

Company Name	SEC Ticker
Apple	AAPL
Accenture plc	ACN
Adobe Systems Incorporated	ADBE
Advanced Micro Devices Inc.	AMD
Akamai Technologies Inc.	AKAM
Altera Corporation	ALTR
Analog Devices Inc.	ADI
Autodesk Inc.	ADSK
Automatic Data Processing Inc.	ADP
Broadcom Corp.	BRCM
CA Technologies	CA
Cisco Systems Inc.	CSCO
Citrix Systems Inc.	CTXS
Dell Inc.	DELL
Electronic Arts Inc.	EA
EMC Corp.	EMC
First Solar Inc.	FSLR
Facebook	FCBK

```
    companies_dict[raw.strip().split()[0]] = raw.strip().split()[1]
```

1.2) Class that calls SEC Scrapper

```
import os
import sys
import re
from pymongo import MongoClient
import requests
from bs4 import BeautifulSoup

class SecCrawler():

    def __init__(self, company, limit_year):
        client = MongoClient('localhost', 27017)
        self.company = company
        self.db = client['SEC_Datas_2']
        self.collection = self.db[company]
        self.correspondance = companies_dict
        self.limit = limit_year

    def get_year_all_doc(self, max_date, count, company):
        '''Return the presentation page where all the document for this
            year are posted. Among them is the 10-K'''
        count=100
        cik = self.correspondance[company]
        base_url =
            "http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK="+str(cik)+"&type
        print(base_url)
```

Company Name	SEC Ticker
Google (Alhpabet)	GOOG
Hewlett-Packard Co	HPQ
Intel Corp	INTC
International Business Machines Corp	IBM
Juniper Networks Inc	JNPR
Microsoft Corporation	MSFT
Motorola Solutions Inc.	MSI
NetApp Inc.	NTAP
NVIDIA Corp	NVDA
Oracle Corp.	ORCL
Qualcomm Inc.	QCOM
Salesforce.com Inc.	CRM
SanDisk Corp.	SNDK
Seagate Technology	STX
Symantec Corp.	SYMC
Teradata	TDC
Texas Instruments Inc.	TXN
VeriSign Inc.	VRSN
Western Digital Corp.	WDN
Xerox Corp	XRX
Xilinx Inc.	XLNX
Yahoo! Inc.	YHOO
Amazon.com Inc.	AMZN
LinkedIn	LNKD
Verizon Communications Inc	VZ
VMware Inc.	VMW
Zynga Inc.	ZNGA

```
    r = requests.get(base_url)
    data = r.text
    soup = BeautifulSoup(data,"html5lib")
    link_list=[]

    for link in soup.find_all('filinghref'):
        URL = link.string
        link_list.append(URL)
    self.link_list = link_list

def save_in_mongoDB(self):
    for year_links in self.link_list:
        date, K_10_link = get_K_10_link(year_links)
        print(K_10_link)
        print(date)
        if int(date[0:4]) < self.limit:
            print('DATE IS ' + date + "we STOP")
            break

        r=requests.get(K_10_link)
```

```
            data = r.text
            soup = BeautifulSoup(data,"html5lib")

            text = soup.getText(strip=True)
            self.collection.insert_one({'date':date, '10_K':text})


def get_K_10_link(documents_link):
    '''The 10-K report is always the first item of the first table'''
    r = requests.get(documents_link)
    data = r.text
    soup = BeautifulSoup(data,"html5lib")
    date = soup.findAll('div',{'class':'info'})[0].text
    tables = soup.findAll('table')
    table = tables[0]
    K_10_link =
        'http://www.sec.gov/'+table.find('tr').nextSibling.nextSibling.find('td').nextSiblin
    return date, K_10_link

1.3) Call SEC Scrapper to store Reports in MongoDB

for company in companies_dict.keys():#companies_dict.keys():
    SecCrawler_object = SecCrawler(company, 2015)
    SecCrawler_object.get_year_all_doc(2018, 1000, company)
    SecCrawler_object.save_in_mongoDB()
```

As the reader can see, we decided to go for with the MongoDB technology as to storing the SEC reports. This is driven by two main reasons. First this NoSql DataBase is much faster than classic SQL DataBase is terms of accessing data. The reports we are storing have an important size, ranging around 40,000 words on average and we want to store (then load) 200 of them. A BigData, NoSQL DataBase is thus necessary.
Bu we could have gone with any NoSQL DataBase. Here our choice was mainly motivated by the format of the data we are using. For each iteration (unique combination of company and report) we are storing meta-data such as dates, text etc. The Json organisation of Mongo makes it algoritmically very easy to access and process.

2.4 Loading and Processing of the Data to train word vectors

Now that we have the data stored locally in MongoDB we can call it and start training a model on the reports. The model we are going to use is a very simple one. We are going to process our data through a Skip-Gram algorythm.

2.4.1 Skip-Gram Algorythm

Word2vec aims at transforming text data ie: words (our reports) into some kind of vectors that we can subsequently use for regressions, classifications and predictions. It is a shallow two-layers neural network but can feed into any kind of deep learning algorithm and bigger neural network.

The whole point of WordtoVec is to group the vectors of similar words together

in a vector space, thus in a mathematical and systematic way. Once the algorithm has finished running, we aim at having each item attached to vector, where this will be further feed to another algorithm to perform the final classification task.

More formally, if we consider the following sequence of words in our training set: $w_1, w_2, w_3, ..., w_t$ the Skip Gram algorithm aims at maximising the log probability/

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c,j\neq 0} logp(w_{t+j}|w_t) \tag{1}$$

where c represents the size of the training context. Calibrating the c is part of the trade off between higher accuracy and higher computational time. What we define by $p(w_{t+j}|w_t)$ relates to the softmax function:

$$p(w_O|w_I) = \frac{exp(v'^T_{wo} v_{wi})}{\sum_{w=1}^{W} exp(v'^T_w v_{wi})} \tag{2}$$

where $v^{'}$ represents the vector representation of the context words (the ones we try to predict) ie: all the words but the skip one whose vector representation is represented by v_{wI}. W represents the number of words in the vocabulary.

In practice our vocabulary size is of around 50,000 words, and is is thus unrealistic to compute the softmax function with 50,000 words for each word.

2.4.2 Reducing the computational cost: the input of Morin and Bengio

To reduce this computational cost that we cannot afford when training our own word vectors, we are going to use an implementation of the algorythm that uses the hierarchical softmax. Instead of evaluating W output nodes at the neural netword final layer, we would just evaluate $log_2(W)$ of them.

If we consider the neural network as a binary tree (from a graphic presepective) then we can define for each node the relative probability of each child nodes.

2.4.3 Negative Sampling

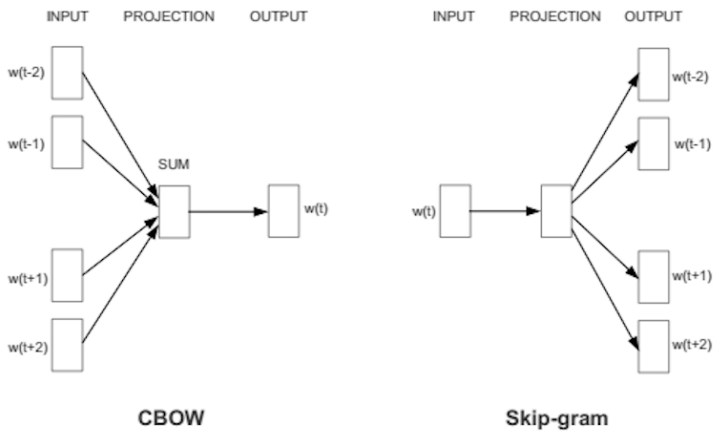
Noise Contrastive Estimation was invented by Gutmann and Hyvarinen and first applied to natural language processing by Minh and Teh. The idea is that it is mostly important to generate few strong similarities between words than an important number of weak one. To do so, we can reduce the number of vectors (word representations effectively) when computing the soft max function. Doing so we can define the Negative Sampling objective function as:

$$log\sigma(v'^T_{wo} v_{w1}) + \sum_{i=1}^{k} \mathbb{E}_{wi\ \ P_n(w)}\left[log\sigma(-v'^T_{wi} v_{wI})\right] \tag{3}$$

This formula basically replaces all the $logP(w_O|w_I)$ term in the Skip-gram objective. Thus we don't distinguish the target word from $vocabulary_Size - 1$ word vectors representation anymore but from k samples, which basically are our negative samples.

NB: They are a lot of theories in terms of choosing the negative sampling parameters. In Distributed Representations of Words and Phrases and their Compositionality,

Tomas Mikolov and his team (Google) don't define this k as a function of the vocabulary size but rather give directions, stating that the k parameter should be no smaller than 20 for small datasets. We subjectively used a 64 one for computational efficiency after noticing that any variation above 50 provides little change in our dataset.



```
// Data Processing and Building Word Vectors
1.1) Load Data from Mongo and Build clean Vocabulary Universe

from bson.objectid import ObjectId

def clean_10_K(report):
    #clean linebreaks that pollutes our documents
    clean_report = report.replace(u'\xa0', u' ')
    #clean introduction part before table of content that has no interest
        in the price prediction
    if 'TABLE OF CONTENTS' in clean_report:
        clean_report = clean_report.split('TABLE OF CONTENTS')[1]
    clean_report = clean_report.split()
    return clean_report

vocabulary = []
for company in companies_dict.keys():
    dates = []
    collection = db[company]
    id_list = collection.distinct('_id')
    for id_ in id_list:
        date = collection.find_one({'_id': ObjectId(id_)})['date']
        if date not in dates:
            dates.append(date)
            vocabulary.append(clean_10_K(collection.find_one({'_id':
                ObjectId(id_)})['10_K']))

new_voc = [word for sublist in vocabulary for word in sublist] #This the
    list containing our vocabulary Universe
len(new_voc) >>> 6,832,521
```

1.2) Processing text data to Numerical Data Set, Sorting Most Common Words

```
vocabulary_size = 50000
def build_dataset(words, n_words):
  """Process raw inputs into a dataset."""
  count = [['UNK', -1]]
  count.extend(collections.Counter(words).most_common(n_words - 1))
  dictionary = dict()
  for word, _ in count:
    dictionary[word] = len(dictionary)
  data = list()
  unk_count = 0
  for word in words:
    index = dictionary.get(word, 0)
    if index == 0: # dictionary['UNK']
      unk_count += 1
    data.append(index)
  count[0][1] = unk_count
  reversed_dictionary = dict(zip(dictionary.values(), dictionary.keys()))
  return data, count, dictionary, reversed_dictionary

data, count, dictionary, reverse_dictionary = build_dataset(new_voc,
    vocabulary_size)
#del vocabulary # Hint to reduce memory.
print('Most common words (+UNK)', count[:5])
print('Sample data', data[:10], [reverse_dictionary[i] for i in data[:10]])

import numpy as np
import random
```

1.3) Function to Process Words in Batch for Skip-Gram (Bi Gram) Model

```
# Batch Generation for Skip Gram ie: Bigram model. We use context word to
    predict before and after
data_index = 0

def generate_batch(batch_size, num_skips, skip_window):
  global data_index
  batch = np.ndarray(shape=(batch_size), dtype=np.int32)
  labels = np.ndarray(shape=(batch_size, 1), dtype=np.int32)
  span = 2 * skip_window + 1 # [ skip_window target skip_window ]
  buffer = collections.deque(maxlen=span)
  if data_index + span > len(data):
    data_index = 0
  buffer.extend(data[data_index:data_index + span])
  data_index += span
  for i in range(batch_size // num_skips):
    context_words = [w for w in range(span) if w != skip_window]
    words_to_use = random.sample(context_words, num_skips)
    for j, context_word in enumerate(words_to_use):
      batch[i * num_skips + j] = buffer[skip_window]
```

```
      labels[i * num_skips + j, 0] = buffer[context_word]
    if data_index == len(data):
      buffer.extend(data[0:span])
      data_index = span
    else:
      buffer.append(data[data_index])
      data_index += 1
  # Backtrack a little bit to avoid skipping words in the end of a batch
  data_index = (data_index + len(data) - span) % len(data)
  return batch, labels


batch, labels = generate_batch(batch_size=8, num_skips=2, skip_window=1)
for i in range(8):
  print(batch[i], reverse_dictionary[batch[i]], '->', labels[i, 0],
        reverse_dictionary[labels[i, 0]])
```

1.4) Run Bi Gram Algorythm on our DataSet

```
import numpy as np
from six.moves import urllib
from six.moves import xrange
import tensorflow as tf
import math
from tensorflow.contrib.tensorboard.plugins import projector
import argparse
import sys
from tempfile import gettempdir


#Store Results in Log
current_path = os.path.dirname(os.path.realpath(sys.argv[0]))

parser = argparse.ArgumentParser()
parser.add_argument(
    '--log_dir',
    type=str,
    default=os.path.join(current_path, 'log'),
    help='The log directory for TensorBoard summaries.')
FLAGS, unparsed = parser.parse_known_args()


#Build and train a skip-gram model.

batch_size = 128
embedding_size = 128 # Dimension of the embedding vector.
skip_window = 1 # How many words to consider left and right.
num_skips = 2 # How many times to reuse an input to generate a label.
num_sampled = 64 # Number of negative examples to sample.

valid_size = 16 # Random set of words to evaluate similarity on.
```

```
valid_window = 100 # Only pick dev samples in the head of the distribution.
valid_examples = np.random.choice(valid_window, valid_size, replace=False)

graph = tf.Graph()

with graph.as_default():

  # Input data.
  with tf.name_scope('inputs'):
    train_inputs = tf.placeholder(tf.int32, shape=[batch_size])
    train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
    valid_dataset = tf.constant(valid_examples, dtype=tf.int32)

  # Ops and variables pinned to the CPU because of missing GPU
      implementation
  with tf.device('/cpu:0'):
    # Look up embeddings for inputs.
    with tf.name_scope('embeddings'):
      embeddings = tf.Variable(
          tf.random_uniform([vocabulary_size, embedding_size], -1.0, 1.0))
      embed = tf.nn.embedding_lookup(embeddings, train_inputs)

    # Construct the variables for the NCE loss
    with tf.name_scope('weights'):
      nce_weights = tf.Variable(
          tf.truncated_normal(
              [vocabulary_size, embedding_size],
              stddev=1.0 / math.sqrt(embedding_size)))
    with tf.name_scope('biases'):
      nce_biases = tf.Variable(tf.zeros([vocabulary_size]))

  with tf.name_scope('loss'):
    loss = tf.reduce_mean(
        tf.nn.nce_loss(
            weights=nce_weights,
            biases=nce_biases,
            labels=train_labels,
            inputs=embed,
            num_sampled=num_sampled,
            num_classes=vocabulary_size))

  # Add the loss value as a scalar to summary.
  tf.summary.scalar('loss', loss)

  # Construct the SGD optimizer using a learning rate of 1.0.
  with tf.name_scope('optimizer'):
    optimizer = tf.train.GradientDescentOptimizer(1.0).minimize(loss)

  # Compute the cosine similarity between minibatch examples and all
      embeddings.
  norm = tf.sqrt(tf.reduce_sum(tf.square(embeddings), 1, keepdims=True))
```

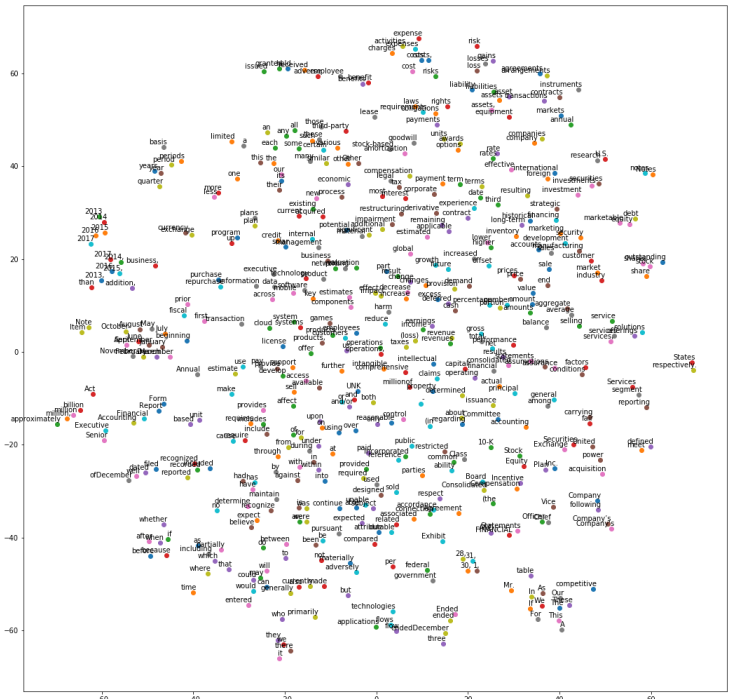
```
  normalized_embeddings = embeddings / norm
  valid_embeddings = tf.nn.embedding_lookup(normalized_embeddings,
                                            valid_dataset)
  similarity = tf.matmul(
      valid_embeddings, normalized_embeddings, transpose_b=True)

  # Merge all summaries.
  merged = tf.summary.merge_all()

  # Add variable initializer.
  init = tf.global_variables_initializer()

  # Create a saver.
  saver = tf.train.Saver()

# Begin training.
num_steps = 100001

with tf.Session(graph=graph) as session:
  # Open a writer to write summaries.
  writer = tf.summary.FileWriter(FLAGS.log_dir, session.graph)

  # We must initialize all variables before we use them.
  init.run()
  print('Initialized')

  average_loss = 0
  for step in xrange(num_steps):
    batch_inputs, batch_labels = generate_batch(batch_size, num_skips,
                                                skip_window)
    feed_dict = {train_inputs: batch_inputs, train_labels: batch_labels}

    # Define metadata variable.
    run_metadata = tf.RunMetadata()

    # We perform one update step by evaluating the optimizer op (including
       it
    # in the list of returned values for session.run()
    # Also, evaluate the merged op to get all summaries from the returned
       "summary" variable.
    # Feed metadata variable to session for visualizing the graph in
       TensorBoard.
    _, summary, loss_val = session.run(
        [optimizer, merged, loss],
        feed_dict=feed_dict,
        run_metadata=run_metadata)
    average_loss += loss_val

    # Add returned summaries to writer in each step.
    writer.add_summary(summary, step)
    # Add metadata to visualize the graph for the last run.
```

```
    if step == (num_steps - 1):
      writer.add_run_metadata(run_metadata, 'step%d' % step)

    if step % 2000 == 0:
      if step > 0:
        average_loss /= 2000
      # The average loss is an estimate of the loss over the last 2000
          batches.
      print('Average loss at step ', step, ': ', average_loss)
      average_loss = 0

    # Note that this is expensive (~20% slowdown if computed every 500
        steps)
    if step % 10000 == 0:
      sim = similarity.eval()
      for i in xrange(valid_size):
        valid_word = reverse_dictionary[valid_examples[i]]
        top_k = 8 # number of nearest neighbors
        nearest = (-sim[i, :]).argsort()[1:top_k + 1]
        log_str = 'Nearest to %s:' % valid_word
        for k in xrange(top_k):
          close_word = reverse_dictionary[nearest[k]]
          log_str = '%s %s,' % (log_str, close_word)
        print(log_str)
  final_embeddings = normalized_embeddings.eval()

  # Write corresponding labels for the embeddings.
  with open(FLAGS.log_dir + '/metadata.tsv', 'w') as f:
    for i in xrange(vocabulary_size):
      f.write(reverse_dictionary[i] + '\n')

  # Save the model for checkpoints.
  saver.save(session, os.path.join(FLAGS.log_dir, 'model.ckpt'))

  # Create a configuration for visualizing embeddings with the labels in
      TensorBoard.
  config = projector.ProjectorConfig()
  embedding_conf = config.embeddings.add()
  embedding_conf.tensor_name = embeddings.name
  embedding_conf.metadata_path = os.path.join(FLAGS.log_dir,
      'metadata.tsv')
  projector.visualize_embeddings(writer, config)

writer.close()


1.5) Visualisation of our Embedings

def plot_with_labels(low_dim_embs, labels, filename):
  assert low_dim_embs.shape[0] >= len(labels), 'More labels than
      embeddings'
```

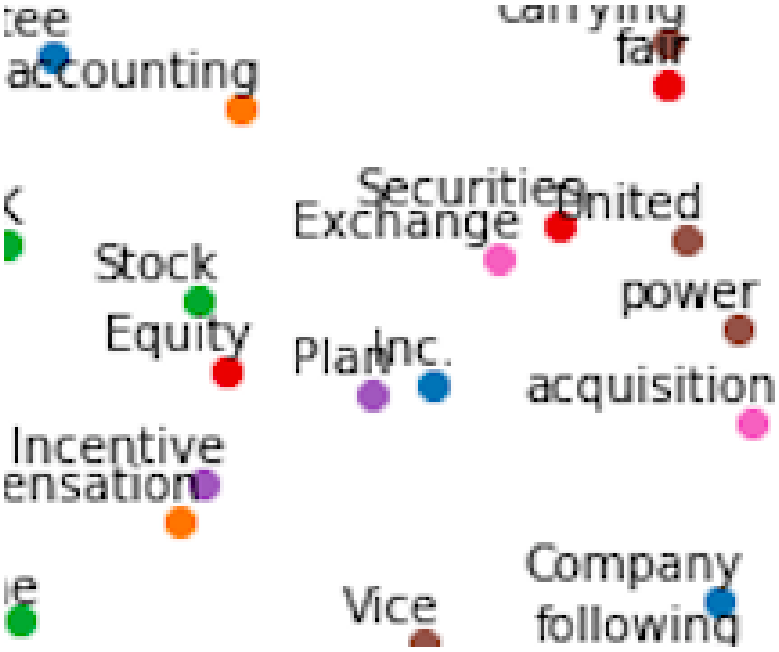
```
  plt.figure(figsize=(18, 18)) # in inches
  for i, label in enumerate(labels):
    x, y = low_dim_embs[i, :]
    plt.scatter(x, y)
    plt.annotate(
        label,
        xy=(x, y),
        xytext=(5, 2),
        textcoords='offset points',
        ha='right',
        va='bottom')

  plt.savefig(filename)


try:
  # pylint: disable=g-import-not-at-top
  from sklearn.manifold import TSNE
  import matplotlib.pyplot as plt

  tsne = TSNE(
      perplexity=30, n_components=2, init='pca', n_iter=5000,
          method='exact')
  plot_only = 500
  low_dim_embs = tsne.fit_transform(final_embeddings[:plot_only, :])
  labels = [reverse_dictionary[i] for i in xrange(plot_only)]
  plot_with_labels(low_dim_embs, labels, 'tsne.png')

except ImportError as ex:
  print('Please install sklearn, matplotlib, and scipy to show
      embeddings.')
  print(ex)
```

This entire algorithm processes in around 15 minutes on a MacBook Air labtop with 8Go RAM and Intel Core I5.

Now let's have a look at visualization we get. I only plot 500 words out of 50,000 for visibility purposes.



The word embedings seem to reflect their specific meanings in the financial industry jargon. If take a closer look at some part of the graph:



In this part of the plot, we clearly see that Stock and Equity which have a very similar meaning in many ways in the world of corporate finance are represented in a similar part of the vector space. Securities and Exchange are obviously very similar

as well given the fact that the exchange is the place were Securities trading takes place. Finally the algorithm produces some more arbitrary association but arguably correct such as acquisition and power. We can understand that in the sense that every MergersAcquisition operation involves a relationship of power between the two protagonist.

2.5 Regressing the Stock Returns on Report release

2.5.1 Getting the Financial Data

Getting the financial data regarding stock returns is done by querying the Quandl API. This API is free and just requires authentication using a personal token. First we need a new correspondence table between the SEC ticker that we used for scrapping and the quandl ticker that we will use to retrieve prices from the API.

Company Name	SEC Ticker	Quandl Ticker
Apple	AAPL	AAPL
Accenture plc	ACN	ACN
Adobe Systems Incorporated	ADBE	ADBE
Advanced Micro Devices Inc.	AMD	AMD
Akamai Technologies Inc.	AKAM	AKAM
Altera Corporation	ALTR	ALTR
Analog Devices Inc.	ADI	ADI
Autodesk Inc.	ADSK	ADSK
Automatic Data Processing Inc.	ADP	ADP
Broadcom Corp.	BRCM	BRCM
CA Technologies	CA	CA
Cisco Systems Inc.	CSCO	CSCO
Citrix Systems Inc.	CTXS	CTXS
Dell Inc.	DELL	DELL
Electronic Arts Inc.	EA	EA
EMC Corp.	EMC	EMC
First Solar Inc.	FSLR	FSLR
Facebook	FCBK	FB

Those prices that we get will from the free Quandl API. Rather than performing a regression of the price move following the publication of the report, we are going to try to classify its impact. Hence if a business day after the report the stock has gone up, our Y vector will take value 1, alternatively if the stock has gone down we will display value -1.

2.5.2 Loading the Data

The data set we consider is the one we scrapped using our Python Scrapper from the SEC EDGAR Database. At this point of the study the data is stored locally in MongoDB and we will access it using the python Library PyMongo.

Company Name	SEC Ticker	
Google (Alhpabet)	GOOG	GOOG
Hewlett-Packard Co	HPQ	HPQ
Intel Corp	INTC	INTC
International Business Machines Corp	IBM	IBM
Intuit Inc	INTU	INTU
Juniper Networks Inc	JNPR	JNPR
Microsoft Corporation	MSFT	MSFT
Motorola Solutions Inc.	MSI	MSI
NetApp Inc.	NTAP	NTAP
NVIDIA Corp	NVDA	NVDA
Oracle Corp.	ORCL	ORCL
Qualcomm Inc.	QCOM	QCOM
Salesforce.com Inc.	CRM	CRM
SanDisk Corp.	SNDK	SNDK
Seagate Technology	STX	STX
Symantec Corp.	SYMC	SYMC
Teradata	TDC	TDC
Texas Instruments Inc.	TXN	TXN
VeriSign Inc.	VRSN	VRSN
Western Digital Corp.	WDN	WDC
Xerox Corp	XRX	XRX
Xilinx Inc.	XLNX	XLNX
Yahoo! Inc.	YHOO	YHOO
Amazon.com Inc.	AMZN	AMZN
LinkedIn	LNKD	LNKD
Verizon Communications Inc	VZ	VZ
VMware Inc.	VMW	VMW
Zynga Inc.	ZNGA	ZNGA

2.5.3 Building the X matrix

As we have seen before the 10-K reports that are published by the companies regulated by the SEC have between a few hundreds and 50,000 words. It is unrealistic to try to apply a convolutional neural network to a matrix with such dimensions. Besides the sample that we were able to gather from the scrapper leave with a sample of 151 reports.
Those two reasons lead us to consider resampling the data. The transformation is very simple. We transform the reports in batches of 500 words. Each sub sample will of course keep the label of the bigger text it is derived from. Also we need each of our corpus to have the same length. We are thus going to *pad* the samples with $< /s >$ until they all have a length of 500 words.

This process leaves us with a complete 13737 samples, each 500 in length.

2.5.4 Building the dictionaries

To process our data we are going to transform our words data to numerical data. *management* becomes 740, *power*: 2033 and *Debentures*: 18400.
This process also allows two things. First we are going to apply a filter on those

words. We have no interest in providing very common words such as *The*, *a* or *of*. Those words occur very often and have absolutely no discriminating power, they will be deleted from our dataset. Secondly, we are able to free a lot of memory space, let's remind ourselves that we are looking at around 7 million words.

Having defined our process we can now implement it:

```
1.1) Associate each Document with its unique return

#Buil the Correspondance Matrix from SEC Ticker to Quandl Ticker
correspd_SEC_quandl = {'AAPL':'AAPL', 'ACN':'ACN','ADBE':'ADBE',
    'AMD':'AMD', 'AKAM':'AKAM','ALTR':'ALTR', 'ADI':'ADI',
                       'ADSK':'ADSK', 'ADP':'ADP', 'BRCM':'BRCM', 'CA':'CA',
                           'CSCO':'CSCO', 'CTXS':'CTXS', 'DELL':'DELL',
                       'EBAY':'EBAY', 'EA':'EA', 'EMC':'EMC', 'FSLR':'FSLR',
                           'FCBK':'FB', 'GOOG':'GOOG', 'HPQ':'HPQ',
                           'INTC':'INTC',
                       'IBM':'IBM', 'INTU':'INTU', 'JNPR':'JNPR',
                           'MSFT':'MSFT', 'MSI':'MSI', 'NTAP':'NTAP',
                           'NVDA':'NVDA',
                       'ORCL':'ORCL', 'QCOM':'QCOM', 'CRM':'CRM',
                           'SNDK':'SNDK', 'STX':'STX', 'SYMC':'SYMC',
                           'TDC':'TDC',
                       'TXN':'TXN', 'VRSN':'VRSN', 'WDN':'WDC', 'XRX':'XRX',
                           'XLNX':'XLNX', 'YHOO':'YHOO', 'AMZN':'AMZN',
                       'LNKD':'LNKD', 'VZ':'VZ', 'VMW':'VMW', 'ZNGA':'ZNGA'}


#Set API Key
quandl.ApiConfig.api_key = '[redacted]

#Function to Get Return From Quandl
#Quandl Return Price Tables, we only want the return on the period
#We are only interesting in Banking Trading Days
def get_return(company, date, delta):
    report_date = pd.to_datetime(date)
    return_date = report_date + timedelta(days=delta)
    if return_date.dayofweek == 5:
        return_date += timedelta(days=2)
    elif return_date.dayofweek == 6:
        return_date += timedelta(days=1)
    data = quandl.get_table('WIKI/PRICES', ticker = [company],
                        qopts = { 'columns': ['ticker', 'date', 'adj_close']
                            },
                        date = { 'gte': date, 'lte':
                            str(return_date).split(' ')[0] }, paginate=True)
    n = data.shape[0]
    if tbl.iloc[0]["adj_close"] < tbl.iloc[n-1]["adj_close"] :
        return -1
    else:
        return 1
```

```
#Dictionary that holds our results: Associates Company and Date with
    Results
info_holder = {}
for company in companies_dict.keys():
    temp = []
    dates = []
    collection = db[company]
    id_list = collection.distinct('_id')
    for id_ in id_list:
        date = collection.find_one({'_id': ObjectId(id_)})['date']
        if date not in dates:
            dates.append(date)
            temp.append({'date':date,
                'return':get_return(correspd_SEC_quandl[company], date, 1)})
                #'text':clean_10_K(collection.find_one({'_id':
                ObjectId(id_)})['10_K']),
    info_holder[company] = temp


1.2) Build our X Matrix and Label Vector

#Establish Connection with Mongo
client = MongoClient('localhost', 27017)
db = client['SEC_Datas_2']

#Text Cleaning Function
def clean_10_K(report):
    #clean linebreaks that pollutes our documents
    clean_report = report.replace(u'\xa0', u' ')
    #clean introduction part before table of content that has no interest
        in the price prediction
    if 'TABLE OF CONTENTS' in clean_report:
        clean_report = clean_report.split('TABLE OF CONTENTS')[1]
    clean_report = clean_report.split()
    return clean_report


sentiment_words_big = []
sentiment_labels_big = []

#Parse all Mongo Json Documents
for company in info_holder.keys():
    for document in info_holder[company]:
        sentiment_labels_big.append(document['return'])
        collection = db[company]
        sentiment_words_big.append(clean_10_K(collection.find_one({'date':document['date']}))

#Resample because texts are too big
#We prefer to split them in smaller, assigning same label as big one
sentiments_words, sentiment_labels = [], []
for counter, text in enumerate(sentiment_words_big):
```

```
    if len(text) > 500:
        for i in range(len(text) // 500):
            sentiments_words.append(text[i*500:(i+1)*500]) # update X
            sentiment_labels.append(sentiment_labels_big[counter]) # update
                Y
        if len(text) > (i+1)*500:
            sentiments_words.append(text[(i+1)*500:len(text)]) # update
                last X
            sentiment_labels.append(sentiment_labels_big[counter]) # update
                last Y

#Standardise the matrix
for sentiment_words in sentiments_words:
    if len(sentiment_words) < 500:
        sentiment_words.extend(['</s>' for _ in range(500 -
            len(sentiment_words))])

words = [word for sublist in sentiments_words for word in sublist]
print(len(words))
print(len(sentiments_words))
print(len(sentiments_words[0]))

1.3) Building our Dictionaries

# We set max vocabulary to this
vocabulary_size = 20000

def build_dataset(words):
  global vocabulary_size
  count = [['UNK', -1]]

  # Sorts words by their frequency
  count.extend(collections.Counter(words).most_common(vocabulary_size - 1))

  # Define IDs for special tokens
  dictionary = dict({'<unk>':0, '</s>':1})

  # Crude Vocabulary Control
  # We ignore the most commone (words like a , the , ...)
  # and most rare (having a repetition of less than 10)
  # to reduce size of the vocabulary
  count_dict = collections.Counter(words)

  for word in words:
    # Add the word to dictionary if not already encountered
    if word not in dictionary:
        if count_dict[word]<50000 and count_dict[word] > 10:
            dictionary[word] = len(dictionary)

  data = []
  unk_count = 0
```

```
  # Replacing word strings with word IDs
  for word in words:
    if word in dictionary:
      index = dictionary[word]
    else:
      index = 0 # dictionary['UNK']
      unk_count = unk_count + 1
    data.append(index)
  count[0][1] = unk_count

  # Create a reverse dictionary with the above created dictionary
  reverse_dictionary = dict(zip(dictionary.values(), dictionary.keys()))

  # Update the vocabulary
  vocabulary_size = len(dictionary)
  return data, count, dictionary, reverse_dictionary

data, count, dictionary, reverse_dictionary = build_dataset(words)
# Print some statistics about the data
print('Most common words (+UNK)', count[:25])
print('Sample data', data[:10])
print('Vocabulary size: ',vocabulary_size)
#del words # Hint to reduce memory.

1.4) Jumping from Strings to Numerical Data

def build_sentiment_dataset(sentiment_words, sentiment_labels):
  '''
  This function takes in reviews and labels, and then replace
  all the words in the reviews with word IDs we assigned to each
  word in our dictionary
  '''
  data = [[] for _ in range(len(sentiment_words))]
  unk_count = 0
  for sent_id,sent in enumerate(sentiment_words):
    for word in sent:
        if word in dictionary:
          index = dictionary[word]
        else:
          index = 0 # dictionary['UNK']
          unk_count = unk_count + 1
        data[sent_id].append(index)

  return data, sentiment_labels

# Run the operation
sentiment_data, sentiment_labels =
    build_sentiment_dataset(sentiments_words, sentiment_labels)
print('Sample data')
for rev in sentiment_data[:10]:
```

```
    print('\t',rev)
```

1.5) Batch Generator

```
# Shuffle the data
sentiment_data, sentiment_labels = shuffle(sentiment_data,
    sentiment_labels)

sentiment_data_index = -1

def generate_sentiment_batch(batch_size, region_size, is_train):
  global sentiment_data_index

  # Number of regions in a single review
  # as a single review has 100 words after preprocessing
  num_r = 100//region_size

  # Contains input data and output data
  batches = [np.ndarray(shape=(batch_size, vocabulary_size),
      dtype=np.int32) for _ in range(num_r)]
  labels = np.ndarray(shape=(batch_size), dtype=np.int32)

  # Populate each batch index
  for i in range(batch_size):
    # Choose a data point index, we use the last 300 reviews (after
        shuffling)
    # as test data and rest as training data
    if is_train:
      sentiment_data_index = np.random.randint(len(sentiment_data)-300)
    else:
      sentiment_data_index = max(len(sentiment_data)-300,
          (sentiment_data_index + 1)%len(sentiment_data))
    #print(sentiment_data_index)
    # for each region
    for reg_i in range(num_r):
        batches[reg_i][i,:] = np.zeros(shape=(1, vocabulary_size),
            dtype=np.float32) #input
        # for each word in region
        for wi in
            sentiment_data[sentiment_data_index][reg_i*num_r:(reg_i+1)*num_r]:

            # if the current word is informative (not <unk> or </s>)
            # Update the bow representation for that region
            if wi != dictionary['<unk>'] and wi != dictionary['</s>']:
              batches[reg_i][i,wi] += 1

    labels[i] = sentiment_labels[sentiment_data_index]

  return batches, labels
```

1.6) Neural Network Architecture

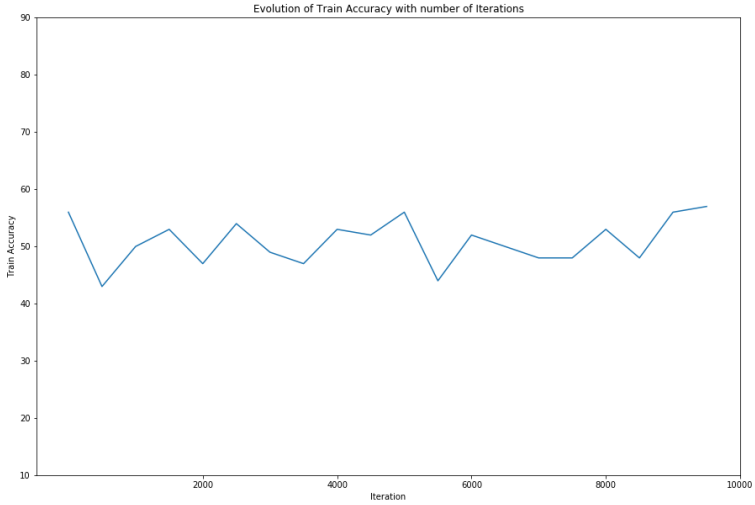
```
batch_size = 50

tf.reset_default_graph()
graph = tf.Graph()

region_size = 10
conv_width = vocabulary_size
conv_stride = vocabulary_size

num_r = 100//region_size

with graph.as_default():

  # Input/output data.
  train_dataset = [tf.placeholder(tf.float32, shape=[batch_size,
      vocabulary_size]) for _ in range(num_r)]
  train_labels = tf.placeholder(tf.float32, shape=[batch_size])

  # Testing input/output data
  valid_dataset = [tf.placeholder(tf.float32, shape=[batch_size,
      vocabulary_size]) for _ in range(num_r)]
  valid_labels = tf.placeholder(tf.int32, shape=[batch_size])

  with tf.variable_scope('sentiment_analysis'):
      # First convolution layer weights/bias
      sent_w1 = tf.get_variable('conv_w1', shape=[conv_width,1,1],
          initializer = tf.contrib.layers.xavier_initializer_conv2d())
      sent_b1 = tf.get_variable('conv_b1',shape=[1], initializer =
          tf.random_normal_initializer(stddev=0.05))

      # Concat all the train data and create a tensor of [batch_size,
          num_r, vocabulary_size]
      concat_train_dataset = tf.concat([tf.expand_dims(t,0) for t in
          train_dataset],axis=0)
      concat_train_dataset = tf.transpose(concat_train_dataset, [1,0,2]) #
          make batch-major (axis)

      concat_train_dataset = tf.reshape(concat_train_dataset, [batch_size,
          -1])

      # Compute the convolution output on the above transformation of inputs
      sent_h = tf.nn.relu(
          tf.nn.conv1d(tf.expand_dims(concat_train_dataset,-1),filters=sent_w1,stride=conv_s
              padding='SAME') + sent_b1
      )

      # Do the same for validation data
      concat_valid_dataset = tf.concat([tf.expand_dims(t,0) for t in
          valid_dataset],axis=0)
      concat_valid_dataset = tf.transpose(concat_valid_dataset, [1,0,2]) #
```

```
        make batch-major (axis)
    concat_valid_dataset = tf.reshape(concat_valid_dataset, [batch_size,
        -1])

    # Compute the validation output
    sent_h_valid = tf.nn.relu(
        tf.nn.conv1d(tf.expand_dims(concat_valid_dataset,-1),filters=sent_w1,stride=conv_s
            padding='SAME') + sent_b1
    )

    sent_h = tf.reshape(sent_h, [batch_size, -1])
    sent_h_valid = tf.reshape(sent_h_valid, [batch_size, -1])

    # Linear Layer
    sent_w = tf.get_variable('linear_w', shape=[num_r, 1], initializer=
        tf.contrib.layers.xavier_initializer())
    sent_b = tf.get_variable('linear_b', shape=[1], initializer=
        tf.random_normal_initializer(stddev=0.05))

    # Compute the final output with the linear layer defined above
    sent_out = tf.matmul(sent_h,sent_w)+sent_b
    tr_train_predictions = tf.nn.sigmoid(tf.matmul(sent_h, sent_w) +
        sent_b)
    tf_valid_predictions = tf.nn.sigmoid(tf.matmul(sent_h_valid, sent_w)
        + sent_b)

    # Calculate valid accuracy
    valid_pred_classes =
        tf.cast(tf.reshape(tf.greater(tf_valid_predictions,
        0.5),[-1]),tf.int32)

    # Loss computation and optimization
    naive_sent_loss =
        tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=tf.expand_dims(train
        logits=sent_out))
    naive_sent_optimizer = tf.train.AdamOptimizer(learning_rate =
        0.0005).minimize(naive_sent_loss)



1.7) Run Training / Testing

num_steps = 10001

naive_valid_ot = []
with
    tf.Session(graph=graph,config=tf.ConfigProto(allow_soft_placement=True))
    as session:
  tf.global_variables_initializer().run()
  print('Initialized')
  average_loss = 0
```

```
for step in range(num_steps):
  if (step+1)%100==0:
      print('.',end='')
  if (step+1)%1000==0:
      print('')

  batches_data, batch_labels = generate_sentiment_batch(batch_size,
      region_size,is_train=True)

  feed_dict = {}
  #print(len(batches_data))
  for ri, batch in enumerate(batches_data):
      feed_dict[train_dataset[ri]] = batch

  feed_dict.update({train_labels : batch_labels})

  _, l, tr_batch_preds = session.run([naive_sent_optimizer,
      naive_sent_loss, tr_train_predictions], feed_dict=feed_dict)

  if np.random.random()<0.002:
      print('\nTrain Predictions:')
      print(tr_batch_preds.reshape(-1))
      print(batch_labels.reshape(-1))
  average_loss += l

  if (step+1) % 500 == 0:
    sentiment_data_index = -1
    if step > 0:
      average_loss = average_loss / 500
    # The average loss is an estimate of the loss over the last 2000
       batches.
    print('Average loss at step %d: %f' % (step+1, average_loss))
    average_loss = 0

    valid_accuracy = []
    for vi in range(2):
      batches_data, batch_labels = generate_sentiment_batch(batch_size,
          region_size,is_train=False)


      feed_dict = {}
      #print(len(batches_data))
      for ri, batch in enumerate(batches_data):

          feed_dict[valid_dataset[ri]] = batch
      feed_dict.update({valid_labels : batch_labels})

      batch_pred_classes, batch_preds =
          session.run([valid_pred_classes,tf_valid_predictions],
          feed_dict=feed_dict)
      valid_accuracy.append(np.mean(batch_pred_classes==batch_labels)*100.0)
```

```
        print(batch_pred_classes.reshape(-1))
        print(batch_labels)
        print()
    print('Valid accuracy: %.5f'%np.mean(valid_accuracy))
    naive_valid_ot.append(np.mean(valid_accuracy))
```

This algorithm is fully available is the notebook provided with the pdf. Unfortunately the performance is not very good and we could not achieve a better prediction than random. This is due to the complexity of the drivers of financial markets.



2.5.5 Discussion of the Drivers

The results that we found are poor. While most analysts tend to consider SEC 10-K reports as a reliable (out of many other) source of information we failed proving a relationship between the positivity of the reports and the effect on the stock price. This work is still in progress and we are investigating how to achieve better results with Quants and Traders in London and Hong Kong. Out of the discussion we have had with the different market actors it seems we can identify a few issues:

- First it seems that quantifying an absolute positivity of the news is a wrong way of going in its core logic. The future price of a stock is not determined by the absolute future performance of the company. It is determined by the future performance of the company compared to the expected performance. That is, when investors by the stock they have a view of the future performance of the company. If the company performs well but not as well as it was expected to do, then the stock price will actually deteriorates.

- A very obvious objection to our study is that we placed ourselves in a simplistic world where this report is only source of information that investors get. Unfortunately the information we consider here is very naive. Although very anticipated, it is also very incomplete. Its incompleteness actually derives from anticipation: as the publication date gets closer, financial news providers, brokers and Investment Banks are going to produce a lot of notes and price

predictions. The effect of the report on the stock price is thus often incorporated before the publication of the report itself. Each quarter hundreds of analysts will thus make predictions regarding AAPL revenues, costs and profits. Empirically it even easy to see that the volatility of the stock around the publication date is actually not much greater than on most days.

- Institutional Investors manage a portfolio of investments. Their investment horizon is much more complex than a binary world where they decide to either invest or withdraw from a specific company. They do so comparing hundreds if not thousands of companies and then make their investment decisions. And since the amount of money they can invest is not unlimited they assess the performance of one company against the performance of another. Such a universe make the study of an isolated report irrelevant and we have to find a way to assess a company 10-K reports against the one of its competitors. What we are working on currently is to come up with a ranking the of the reports of companies in a given sector where investors are likely to compare the performance of companies between them. We would then predict a positive impact on the stock price of one company if she comes higher in the ratings of its competitive pool.

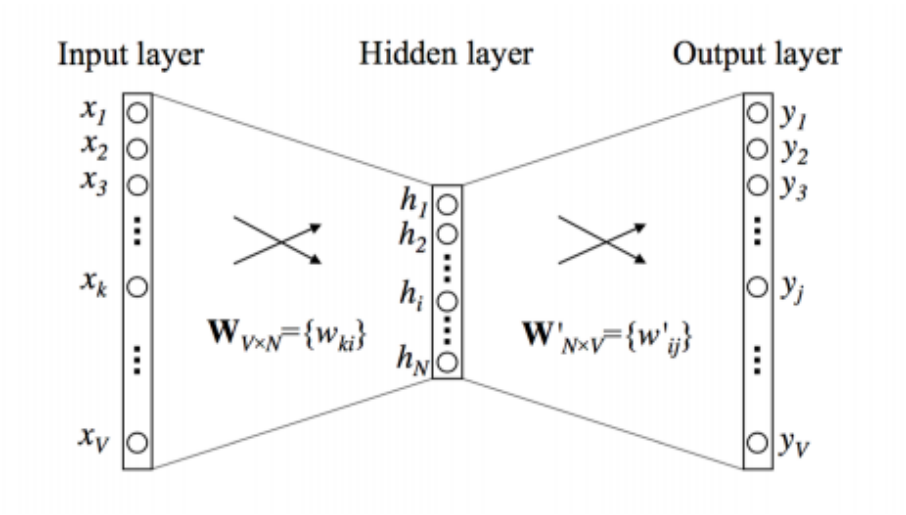
- Of interest is also the time frame that we consider. For this study, we have decided to focus on a one day window. But reports that are published one time of year can arguably produce an impact for a period longer than a day. Fitting this time window is quiet hard because it really comes down to providing the model the level of over fitting that you want. What we have tried instead of adding some pure bias by choosing the time windows where we get the best results is to actually multiply the number of samples that we have by three, ie: consider a classification on one day, one month and one year (right before the next report is released). Unfortunately we did not get better results. (The code remains the same, only the query of the Quandl API is different).

- Another point that was raised by the traders is inherent to the nature of the source of information. Going back to the start, what we try to do is a Sentiment Analysis of the text. The problem that look at here is the level of standardization that those documents have. Not only are those documents are written on weeks, they are also put together by many different analysts and top management. We thus end up with a document that is very different from a simple speech and the task becomes much more complex than assessing if a speech is positive or not.

- We don't loose faith is the algorithm though as its performance on movies review classification for instance is actually very high.

3 ANNEX

We did not want to make the heavy algebra part of our actual report but rather display aside in the annex so the interested reader can have a look and the less interested reader can just skip it.
We are going to derive the mathematical computation of the algorithm we implemented as to training word vectors in the case of one context vector. Extension can then be derived easily to any number of context words. Recall that one context word means that we predict one word using its immediate predecessor.



We denote by V the vocabulary size and N the hidden layer size.
As we said before, we have chosen a certain vocabulary size of 20,000 words. In the universe, an input word is represented by a vector $x = x_1, x_2....x_v$, where $x_k = 1 and all others values are equal to 0. Hence the common notation one hot vector.$

We dnote the weights transitioning from the input layer to the hidden layer by W whose dimension is $V * N$. Each row of this transition matrix can be seen as the context representation of the words. Recall that x input vectors are actually one hot vectors, then the product of x by W basically copies the row k^{th} row of the matrix where $x_k = 1$ and $x_j = 0$ for all $j \neq k$. We have:

$$h = x^T W = W_{(k,.)} := v_{wI} \tag{4}$$

where we call v_{wI} the context vector representation of of the input word.
Recall that :

$$\mathrm{W}_{V*N} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1N} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{v1} & w_{v2} & w_{v3} & \dots & w_{vN} \end{bmatrix}$$

and

$$\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_V \end{bmatrix}$$

then

$$h = x^T W = [x_1 x_2 ... x_k ... x_V] * \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1N} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{v1} & w_{v2} & w_{v3} & \dots & w_{vN} \end{bmatrix} \tag{5}$$

$$h = [x_k w_{k1} x_k w_{k2} ... x_k w_{kN}] \tag{6}$$

$$h = [w_{k1} w_{k2} ... w_{kN}] \tag{7}$$

$$h = W_{k,)} \tag{8}$$

$$h = v_{wI} \tag{9}$$

Now, on the output side on the shallow neural network, we have a transition matrix $W^{'}$ out puting to the word vectors. This allows us to compute a score for each word that we consider in the vocabulary:

$u_j = {v^{'}_{wj}}^T . h$

where u_j is thus of dimension 1 and can be seen as the match between the context word and the output word.

At this step we thus end up with V scalars, each representing when we can consider as a matching score of context word to output word.

We then use the softmax classification function:

$$p(w_O|w_I) = y_j = \frac{exp(u_j)}{\sum_{j=V}^{j} exp(u_{j'})} \tag{10}$$

and if we replace y_j in the above expression we can see the impact of the context Matrix W and the output matrix $W^{'}$.

$$p(w_O|w_I) = \frac{exp(v^{'T}_{wo} v_{wi})}{\sum_{w=1}^{W} exp(v^{'T}_{w} v_{wi})} \tag{11}$$

Now if we recall that we aim at maximizing equation 11, we can write:

$$maxp(w_O|w_I) = maxy_j \tag{12}$$

which comes to maximizing the log value:

$$logmaxp(w_O|w_I) = maxlogy_j \tag{13}$$

and replacing from equation 11:

$$maxp(w_O|w_I) = u_j = log \sum_{j'=1}^{V} exp(u_{j'}) := -E \tag{14}$$

Let us consider one single training example. The objective is to maximize the above quantity ie: to find the most probable word output w_O having the word w_I

as input. If we want to jump back to a minimisation problem then we can consider the above E as our cost function, to minimize.
Now we can see that:

$$\frac{\partial E}{\partial u_j} = y_j - t_j = e_j \tag{15}$$

Also we can remark that $t_j = 1(j = j^*$

Now we have to compute the update parameters of the output matrix:

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial uj} . \frac{\partial u_j}{\partial w'_{ij}} = e_j . h_j \tag{16}$$

from:

$$\frac{\partial E}{\partial uj} = \frac{\partial (u_{j*} - log \sum_{j'=1}^{V} exp(uj'))}{\partial u_{j*}} \tag{17}$$

$$= \frac{\partial u_j}{\partial uj*} = \frac{(log \sum_{j'=1}^{V} exp(u_{j'})}{\partial u_j} \tag{18}$$

$$= t_j - \frac{exp(u_j)}{\sum_{j'=1}^{V} exp(u_{j'})} \tag{19}$$

$$= t_j - y_j \tag{20}$$

We can now process the data in batch, we write the example using stochastic gradient descent:

$$w'^{new}_{ij} = w'^{old}_{ij} - \eta . e_j . h_j \tag{21}$$

which is equivalent in our first notation using rows of matrix W to:

$$v'^{new}_{wj} = w'^{old}_{vj} - \eta . e_j . h \tag{22}$$

where we define of course $\eta by the learning rate we want to apply to our algorithm.$
Now that we have updated the wiehgts for W' we need to update them for W.

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} . \frac{\partial E}{\partial u_j} . \frac{\partial u_j}{\partial h_i} \tag{23}$$

$$= \sum_{j=1}^{V} e_j . w'_{ij} \tag{24}$$

$$EH_i \tag{25}$$

where as we defined before, h_i is the output of the i-th unit in the hidden layer (before transformation with the softmax) and $\mathrm{u}_j is a scalar and e_j = y_j - t_j$ is the prediction error for our j-th word. Finally the vector EH lives in N.

Now we can finally compute our the derivative of E matrix with respect to our first weight matrix: W.

Recall that that the hidden layer performs a linear computation by computing the dot product of the input vector x (as we have seen before those are one hot). Thus we had for each vector h:

$$h_i = \sum_{k=1}^{V} x_k.w_{ki} \tag{26}$$

$$= EH_i.x_k \tag{27}$$

eventually we get the updating equation for W:

$$v_{wI}^{'new} = v^{'old} - \eta.EH \tag{28}$$

4 References

Michael U. Gutmann, Aapo Hyvärinen 2012. Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics.

Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. *Domain adaptation for large-scale sentiment classification: A deep learning approach.* In ICML, 513–520, 2011.

Michael U Gutmann and Aapo Hyv¨arinen. *Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics.* The Journal of Machine Learning Research, 13:307–361,2012.

Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. *Extensions of recurrent neural network language model. In Acoustics, Speech and Signal Processing* (ICASSP), 2011 IEEE International Conference on, pages 5528–5531. IEEE, 2011.

Andriy Mnih and Geoffrey E Hinton. *A scalable hierarchical distributed language model. Advances in neural information processing systems*, 21:1081–1088, 2009.

Andriy Mnih and Yee Whye Teh. *A fast and simple algorithm for training neural probabilistic language models.* arXiv preprint arXiv:1206.6426, 2012.

Frederic Morin and Yoshua Bengio. *Hierarchical probabilistic neural network language model. In Proceedings of the internationalworkshop on artificial intelligence and statistics,* pages 246–252, 2005.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. *Word representations: a simple and general method for semi-supervised learning.* 2010.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. *Semantic Compositionality Through Recursive Matrix-Vector Spaces.*

Richard Socher, Cliff C. Lin, Andrew Y. Ng, and Christopher D. Manning. *Parsing natural scenes and natural language with recursive neural networks.*