Can basic RNN/LSTM correctly predict WSJ article category?

1. Import WSJ articles

In [2]:
```
from bs4 import BeautifulSoup
import requests
import re
from selenium import webdriver
import tensorflow as tf
import os
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import pickle
from collections import Counter
import itertools
import pandas as pd
from tensorflow.contrib.tensorboard.plugins import projector
import matplotlib.pyplot as plt
```

1.1 import all article urls in the market tab

In [3]:
```
#For reloading data
#with open( "article.pkl", 'rb') as pickle_file:
#    article = pickle.load(pickle_file)
#with open("category.pkl", 'rb') as pickle_file:
#    category = pickle.load(pickle_file)
#final_embeddings = np.load("./my_final_embeddings.npy")
```

In [ ]:
```
url_path = f'https://www.wsj.com/search/term.html?KEYWORDS=Markets&page=1'
response = requests.get(url_path)
soup = BeautifulSoup(response.text,'lxml')
total_page_count = int(soup.find_all(class_='results-count')[-1].text.split()[-1])
root_url = 'https://www.wsj.com'
url = []
page_length = 20 # len(soup.find_all(class_='headline')) include video urls
for i in range(total_page_count):
    url_path = f'https://www.wsj.com/search/term.html?KEYWORDS=Markets&page={i+1}'
    response = requests.get(url_path)
    soup = BeautifulSoup(response.text,'lxml')
    for j in range(page_length):
        if soup.find_all(class_='headline')[j].find('a')['href'].find("https")== -1:
            url.append(root_url + soup.find_all(class_='headline')[j].find('a')['href'])
        else:
            url.append(soup.find_all(class_='headline')[j].find('a')['href'])
```

In [192]:
```
pickle.dump(url, open( "url.pkl", "wb" ) )
```

1.2 Scrape all articles from the urls.

Note: Since WSJ article is available by subscription, I used selenium for login

In [282]:
```
chromedriver = "/Applications/chromedriver" # path to the chromedriver e
xecutable
os.environ["webdriver.chrome.driver"] = chromedriver
# Start the driver
driver = webdriver.Chrome(chromedriver)
article = defaultdict(list)
category = defaultdict(list)
driver.get(url[0])
html = driver.page_source
soup = BeautifulSoup(html,'lxml')
if soup.find(role="button"):
    (WebDriverWait(driver,1000).until(EC.presence_of_element_located\
                                      ((By.CLASS_NAME,"style__login-butt
ons_2AlXz2kN-GyfeoGvsttrZh"))))
    (driver.find_element_by_class_name("style__login-buttons_2AlXz2kN-Gy
feoGvsttrZh")\
     .find_elements_by_tag_name("a")[1].click())
elif soup.find(text="Sign In"):
    WebDriverWait(driver,1000).until(EC.presence_of_element_located((By.
LINK_TEXT,"Sign In")))
    driver.find_element_by_link_text("Sign In").click()
driver.implicitly_wait(10)
driver.find_element_by_name("username").send_keys('xxxx@email.com') # se
nsitive information deleted
driver.find_element_by_name("password").send_keys('xxxx') # sensitive in
formation deleted
driver.find_element_by_class_name("solid-button").click()
WebDriverWait(driver,1000).until(EC.presence_of_element_located((By.CLAS
S_NAME,"title")))


for i in range(len(url)):
    driver.get(url[i])
    html = driver.page_source
    soup = BeautifulSoup(html,'lxml')
    if soup.find(class_="article-breadCrumb-wrapper"): # if there is a c
ategory
        if  soup.find(class_="login-buttons"):
            continue
        article[i]+=[driver.title[:-6]]
        #article[i]+=[driver.find_element_by_class_name("sub-head").tex
t]
        if soup.find(class_="article-content"):
            paragraph = driver.find_element_by_class_name("article-conte
nt").find_elements_by_tag_name("p")
        elif soup.find(id="wsj-article-wrap"):
            paragraph = driver.find_element_by_id("wsj-article-wrap").fi
nd_elements_by_tag_name("p")
        for j in range(len(paragraph)-1):
            if paragraph[j].text.find(".")>0:
                article[i] += [paragraph[j].text]
        category_js = (driver.find_element_by_class_name\
                       ("article-breadCrumb-wrapper").find_elements_by_t
ag_name("span"))
        for k in range(len(category_js)):
            category[i] += [category_js[k].text]
```

```
        if i%10==0:
            print(f'article{i+1} processed.')
driver.quit()
```

```
article6051 processed.
article6061 processed.
article6071 processed.
article6081 processed.
article6091 processed.
article6101 processed.
article6111 processed.
article6131 processed.
```

In [283]:
```
pickle.dump(article, open( "article.pkl", "wb" ) )
pickle.dump(category, open( "category.pkl", "wb" ) )
```

In [7]:
```
article[1]
```

Out[7]:
```
['Australian Dollar Declines After RBA Signals Possible Rate Cut',
 'The dollar strengthened Tuesday, buoyed by gains against the Australi
an dollar after Reserve Bank of Australia cleared the path to an intere
st-rate cut in June.',
 'The Australian dollar declined 0.4% against the U.S. dollar to 68.82
U.S. cents, while the New Zealand dollar also slid 0.4% to 65.05 U.S. c
ents.',
 'Both currencies fell after the release of minutes from the RBA’s May
7 meeting indicated that a deterioration in Australia’s job market woul
d warrant the first rate reduction since 2016. RBA Governor Philip Lowe
later said that officials would “consider the case for lower interest r
ates” at the central bank’s next meeting.',
 'Those comments sapped demand for the Australian dollar because lower
interest rates make currencies less attractive to yield-seeking investo
rs.',
 'Meanwhile, the dollar was also up 0.5% against the Japanese yen, as i
nvestors showed greater appetite for riskier assets after the Trump adm
inistration said it would grant temporary exemptions to an export black
list against Huawei Technologies.',
 'That move helped ease concerns about U.S.-China trade tensions, which
have escalated in recent weeks and sparked fresh concerns about the glo
bal growth outlook.',
 'The yen is a popular destination for investors during times of politi
cal or economic uncertainty.',
 'The WSJ Dollar Index, which measures the U.S. currency against a bask
et of 16 others, rose for the sixth time in seven days, advancing 0.1%
to 91.24.']
```

In [6]:
```
category[1]
```

Out[6]:
```
['MARKETS', 'CURRENCIES', 'FOREIGN EXCHANGE']
```

2. Word Embedding using scraped article

The rationale for this step primarily is two-fold.

First, it significantly reduces dimension from sparse one-hot-vectors.

Second, similar words are represented similarly by the dimension reduction. In this project, the definition of similarity follows the skip-gram model. Briefly, it is a way of highlighting the relationship between a context word and its neighboring words. For example, we want to predict "I" or "happy" from the context word "am" from the string "I am happy".

2.1 Transform the collection of articles into sequence of words

In [10]:
```
flatten_article = list(itertools.chain(*article.values()))
article_to_string = ' '.join(flatten_article)
```

In [13]:
```
article_to_string[:200]
```

Out[13]:
```
'Should Uber Bar Felons From Becoming Drivers? Roughly 30% of Americans
have a criminal record, potentially disqualifying millions of people wi
th past traffic violations, theft and drug-related offense'
```

In [41]:
```
query = re.compile("[a-zA-Z]*")
words = query.findall(article_to_string)
```

In [42]:
```
words = list(filter(lambda x: x!='',words))
words = list(map(lambda x: x.lower(),words))
```

```
 'at',
 'agresource',
 'with',
 'the',
 'usda',
 's',
 'crop',
 'progress',
 'continuing',
 'to',
 'report',
 'slow',
 'planting',
 'particularly',
 'in',
 'corn',
 'traders',
 'expect',
 'prices',
 'to',
 'rise',
 'further',
 'i',
 'm',
 'bullish',
 'at',
 'these',
 'prices',
 'said',
 'john',
 'payne',
 ...]
```

In [61]:

```
# Less frequently appearing words are dumped into the word "UNK"
vocabulary_size = 50000
vocabulary = [("UNK", None)] + Counter(words).most_common(vocabulary_siz
e - 1)
vocabulary = np.array([word for word, _ in vocabulary])
dictionary = {word: code for code, word in enumerate(vocabulary)}
data = np.array([dictionary.get(word, 0) for word in words])
```

2.2 Implement Skip-gram

The codes are from Tensorflow tutorial applied to WSJ articles but I added more comments to explain each step

In [321]:
```
from collections import deque
data_index = 0
def generate_batch(batch_size, num_skips, skip_window):
    """
    batch_size: length of a given batch
    num_skips: the number of times context word (center word) is duplica
ted in the batch
    skip_window: How many left neighbors (or right neighbors) will be co
nsidered for each context word

    In this partciular implementation, neighbors for each context word w
ill be chosen randomly within the
    skip_window (uniquely). num_skips determine the number of neighbors
 to be drawn from the skip_window

    """
    global data_index
    assert batch_size % num_skips == 0 # if num_skips=2, batch_size=4: b
atch = ['a','a', 'bv','bv']
    assert num_skips <= 2 * skip_window
    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 = deque(maxlen=span)
    for _ in range(span):
        buffer.append(data[data_index]) #data_index initially = 0
        # if data_index exceeds length of words vector restart data_inde
x from 0
        data_index = (data_index + 1) % len(data)
    for i in range(batch_size // num_skips): # number of unique elements
in the batch
        # target label at the center of the buffer. i.e. if skip_window
 =2 then span=5
        # so buffer=[x1,x2,x3,x4,x5] and target[skip_window]=x3, the cen
ter value
        target = skip_window  # target label at the center of the buffer
        targets_to_avoid = [ skip_window ]
        for j in range(num_skips):
            # i.e. buffer=[x1,x2,x3,x4,x5] then initially, randomly sele
ct
            # index other than that for x3 i.e. don't want target = 2
            # suppose target = 1. Then, targets_to_avoid = [2,1]
            # batch will be filled with center word of the buffer. Same
 number will be appeneded
            # to the same batch by the num_skips. i.e. batch = [x3,x3, x
4,x4,x5,x5,x6,x6]
            # labels consist of unique neighbor of each center word with
in buffer
            while target in targets_to_avoid:
                target = np.random.randint(0, span)
            targets_to_avoid.append(target)
            batch[i * num_skips + j] = buffer[skip_window]
            labels[i * num_skips + j, 0] = buffer[target]
        buffer.append(data[data_index])
        data_index = (data_index + 1) % len(data)
    return batch, labels
```

In [319]:
```
batch_size = 128 # element of batch is the context word and element of l
abel is its neighbor (of the same size)
skip_window = 1       # line 18 in the cell above
num_skips = 2         # line 14 in the cell above
```

In [70]:
```
def reset_graph(seed=42):
    tf.reset_default_graph()
    tf.set_random_seed(seed)
    np.random.seed(seed)
```

In [341]:
```
reset_graph()
```

In [342]:
```
# Input data.
train_labels = tf.placeholder(tf.int32, shape=[batch_size, 1])
valid_dataset = tf.constant(valid_examples, dtype=tf.int32)
```

In [30]:
```
vocabulary_size = 50000
embedding_size = 150

# Look up embeddings for inputs.
init_embeds = tf.random_uniform([vocabulary_size, embedding_size], -1.0,
1.0)
embeddings = tf.Variable(init_embeds)
```

In [344]:
```
train_inputs = tf.placeholder(tf.int32, shape=[None])
embed = tf.nn.embedding_lookup(embeddings, train_inputs)
```

In [345]:
```
# Construct the variables for the NCE loss

learning_rate = 0.01

nce_weights = tf.Variable(
    tf.truncated_normal([vocabulary_size, embedding_size],
                        stddev=1.0 / np.sqrt(embedding_size)))
nce_biases = tf.Variable(tf.zeros([vocabulary_size]))

# Compute the average NCE loss for the batch.
# tf.nce_loss automatically draws a new sample of the negative labels ea
ch
# time we evaluate the loss.
loss = tf.reduce_mean(
    tf.nn.nce_loss(nce_weights, nce_biases, train_labels, embed,
                   num_sampled, vocabulary_size))

# Construct the Adam optimizer
optimizer = tf.train.AdamOptimizer(learning_rate)
training_op = optimizer.minimize(loss)

# Compute the cosine similarity between minibatch examples and all embed
dings.
norm = tf.sqrt(tf.reduce_sum(tf.square(embeddings), axis=1, keepdims=Tru
e))
normalized_embeddings = embeddings / norm
valid_embeddings = tf.nn.embedding_lookup(normalized_embeddings, valid_d
ataset)
similarity = tf.matmul(valid_embeddings, normalized_embeddings, transpos
e_b=True)

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

```
WARNING: Logging before flag parsing goes to stderr.
W0523 10:49:37.144406 4447532480 deprecation.py:323] From /Users/gimdon
g-geon/python3_cooking/lib/python3.7/site-packages/tensorflow/python/op
s/nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from tensorflo
w.python.ops.array_ops) is deprecated and will be removed in a future v
ersion.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

In [19]:
```
# Following parameters are not used for training the model
# However, these are for observing cosine similarity of randomly chosen
 words
# at the beginning and end of training

valid_size = 16     # Random set of words to evaluate similarity on.
valid_window = 100  # Only pick dev samples in the head of the distribut
ion.
valid_examples = np.random.choice(valid_window, valid_size, replace=Fals
e) # random index of 16 words that range

# from 0 to valid_window = 100
num_sampled = 64    # Number of negative examples to sample.
# Computing cross-entropy from the entire corpus every iteration would b
e costly. NCE is a way of coming up
# with a loss function that shares similar characteristics with the cros
s entropy.
```

In [346]:
```
num_steps = 10001

with tf.Session() as session:
    init.run()

    average_loss = 0
    for step in range(num_steps):
        print("\rIteration: {}".format(step), end="\t")
        batch_inputs, batch_labels = generate_batch(batch_size, num_skip
s, skip_window)
        feed_dict = {train_inputs : batch_inputs, train_labels : batch_l
abels}

        # We perform one update step by evaluating the training op (incl
uding it
        # in the list of returned values for session.run()
        _, loss_val = session.run([training_op, loss], feed_dict=feed_di
ct)
        average_loss += loss_val

        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 5
00 steps)
        if step % 10000 == 0:
            sim = similarity.eval()
            for i in range(valid_size):
                valid_word = vocabulary[valid_examples[i]]
                top_k = 8 # number of nearest neighbors
                nearest = (-sim[i, :]).argsort()[1:top_k+1] # argsort is
argmin so -sim to get argmax
                log_str = "Nearest to %s:" % valid_word
                for k in range(top_k):
                    close_word = vocabulary[nearest[k]]
                    log_str = "%s %s," % (log_str, close_word)
                print(log_str)

    final_embeddings = normalized_embeddings.eval()
```

```
Iteration: 0    Average loss at step  0 :  290.19189453125
Nearest to than: starkly, disrupts, exercised, respects, expressly, rus
t, rumpel, btus,
Nearest to sales: toying, likeliest, engage, poetic, overhauled, mastec
tomy, masterpieces, practitioners,
Nearest to been: ths, phasing, maeghan, sonntags, intercourse, stressfu
l, kirsch, fenders,
Nearest to u: abdelmoumen, rise, bingo, faxed, unreleasable, manet, eur
omonitor, restive,
Nearest to but: bucking, steepen, wti, pretty, undermined, gardasil, co
urt, arrondissements,
Nearest to may: consciousness, screwed, videogame, randgold, recyclable
s, succinctly, luspatercept, greats,
Nearest to some: carlsberg, vml, singlethread, muscles, preventive, fin
emark, sunseeker, depositors,
Nearest to have: canberra, dink, admitted, journey, path, gilbert, refi
ne, pacvue,
Nearest to growth: rug, vcg, sassanian, meh, heteroskedasticity, regula
tory, barba, flagler,
Nearest to to: meituan, strada, irvine, valuations, helvea, nook, chakr
abarti, contingent,
Nearest to a: jim, yearbook, ayatollahs, rationalizable, upsides, retro
spective, heel, imprudent,
Nearest to has: illumination, mented, sutures, arborists, sock, csx, fu
ndaci, perpetually,
Nearest to would: strappatelle, childress, kfw, crime, trainer, retesti
ng, detelina, biogen,
Nearest to prices: harrods, fellow, detract, hiball, throat, subsidize,
realms, disturbances,
Nearest to you: archard, membership, dazzled, risibly, secularism, fian
cee, undersupply, reassessing,
Nearest to s: ftr, scalefactor, butter, smudged, beme, rehearsal, rocki
ng, handcuffs,
Iteration: 2000 Average loss at step  2000 :  130.73070597457885
Iteration: 4000 Average loss at step  4000 :  60.272551845312115
Iteration: 6000 Average loss at step  6000 :  38.34326530981064
Iteration: 8000 Average loss at step  8000 :  29.55954886651039
Iteration: 10000        Average loss at step  10000 :  23.1665637874603
26
Nearest to than: crashed, detained, civica, corpus, crohn, blows, more,
achleitner,
Nearest to sales: chlorides, sycamore, laurel, geisel, panicked, tivit
y, rerating, ntsb,
Nearest to been: abraaj, blankfein, duperreault, tendency, is, made, sw
orn, are,
Nearest to u: china, mellanox, blazer, uk, truckers, dermatology, chatt
er, shiller,
Nearest to but: that, barratt, and, reynolds, oram, uefa, elkins, supr
a,
Nearest to may: eisbrenner, yaccarino, should, might, could, astor, inf
ringed, mamdani,
Nearest to some: krw, arnault, waste, pinot, elfers, vulnerabilities, x
fl, nepal,
Nearest to have: has, had, ira, gop, posenenske, foley, missiles, by,
Nearest to growth: growing, response, hikma, results, envelope, carillo
n, deportation, nebraska,
Nearest to to: would, tivity, that, and, elfers, cariclub, stronach, pa
```

```
ssword,
Nearest to a: the, blazer, cosco, radicchio, israelis, mng, unnamable,
abraaj,
Nearest to has: have, s, had, in, broux, noncompetes, by, sewell,
Nearest to would: to, the, that, can, shiller, nontransparent, could, a
nd,
Nearest to prices: injunction, sculptural, brooklyn, to, yields, kuber,
spurs, pyongyang,
Nearest to you: malls, we, murdered, proactive, planner, bodes, sycamor
e, uniform,
Nearest to s: the, and, in, seuss, norsk, sayer, has, streamers,
```

In [347]:

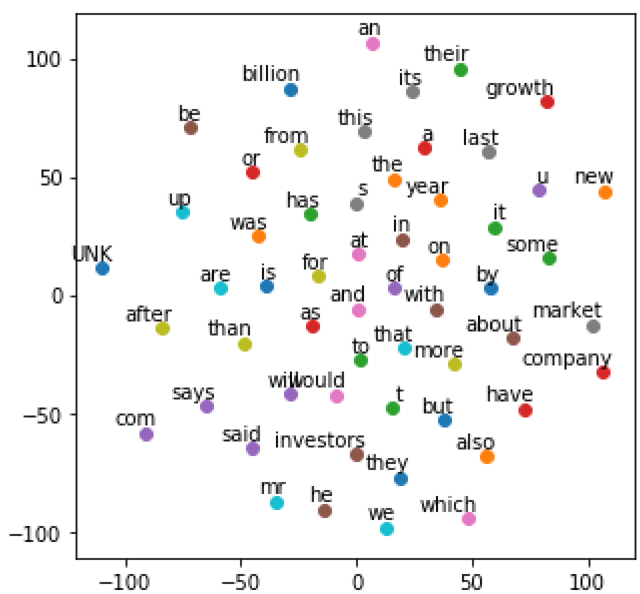
```python
np.save("./my_final_embeddings.npy", final_embeddings)
```

In [590]:

```python
def plot_with_labels(low_dim_embs, labels):
    assert low_dim_embs.shape[0] >= len(labels), "More labels than embed
dings"
    plt.figure(figsize=(5, 5))  #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')
```

In [596]:

```python
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
%matplotlib inline
tsne = TSNE(perplexity=30, n_components=2, init='pca', n_iter=5000)
plot_only = 56
low_dim_embs = tsne.fit_transform(final_embeddings[:plot_only,:])
labels = [vocabulary[i] for i in range(plot_only)]
plot_with_labels(low_dim_embs, labels)
```



In [9]:
```
final_embeddings.shape
```

Out[9]:
```
(50000, 150)
```

3. Predict WSJ article category

3.1 Preprocess articles

Transform each word in each article into unique index assigned during embeddings. Then, the word embedding associated with a particular word will be the row of the embedding matrix where row index equals the unique index.

In [21]:
```
query = re.compile("[a-zA-Z]*")
words = query.findall(article_to_string)
words = list(filter(lambda x: x!='',words))
words = list(map(lambda x: x.lower(),words))
```

In [22]:
```
# Restrict attention to articles that have category assignments
index = list(article.keys() & category.keys())
article_intersection = dict({k: article[k] for k in index})
string_by_article = [' '.join(x) for x in article_intersection.values()]
string_alphabet_by_article = [query.findall(x) for x in string_by_articl
e]
word_by_article = [list(filter(lambda x: x!='',y)) for y in string_alpha
bet_by_article]
word_by_article = [list(map(lambda x: x.lower(),y)) for y in word_by_art
icle]
```

In [28]:
```
k=0 #k is the article index
word_by_article[k][:20]
```

Out[28]:
```
['australian',
 'dollar',
 'declines',
 'after',
 'rba',
 'signals',
 'possible',
 'rate',
 'cut',
 'the',
 'dollar',
 'strengthened',
 'tuesday',
 'buoyed',
 'by',
 'gains',
 'against',
 'the',
 'australian',
 'dollar']
```

In [58]:
```
def padder(x,max_len=50):
    """
    padder function ensures that we use at most max_len number of words
 per article to predict category
    and it transforms each word to its unique index
    x is the collection of words per article (list)
    """
    if len(x)>=max_len:
        return list(map(lambda y: dictionary[y] if y in dictionary.keys
() else 0,x[:max_len]))
    else:
        tmp = list(np.zeros(max_len,int))
        tmp[:len(x)]=list(map(lambda y: dictionary[y] if y in dictionary
.keys() else 0,x))
        return tmp
```

In [62]:
```
max_len=50
word_label_by_article = [padder(x,max_len) for x in word_by_article]
```

In [63]:
```
# each article becomes one input. Each input is a concatenation of word
 embeddings per article
n_inputs = embedding_size * max_len
prev_data = final_embeddings[word_label_by_article[0]].reshape(-1,n_inpu
ts)
for i in range(1,len(word_label_by_article)):
    new_data = np.concatenate([prev_data, final_embeddings[word_label_by
_article[i]].reshape(-1,7500)])
    prev_data = new_data
X_batch=new_data
```

In [64]:
```
X_batch.shape
```

Out[64]:
```
(4903, 7500)
```

In [34]:
```
# Categories of articles to which we pay attention. This is what this no
tebook wants to predict
category_intersection = dict({k:category[k] for k in index})
list(filter(lambda x: x!=[],category_intersection.values()))
```

```
 'PRO BANKRUPTCY DISTRESS': 171,
 'PRO VC PEOPLE': 172,
 'PRO VC NEWSLETTER': 173,
 'TRI-STATE AREA': 174,
 'DECO SUMMARY (PLAIN)': 175,
 'PRO PE LIMITED PARTNERS': 176,
 'U.K.': 177,
 'RUSSIA': 178,
 'NFL': 179,
 'WORLD NEWS': 180,
 'JAPAN': 181,
 'YOUR HEALTH': 182,
 'PRO VC VC FUNDS': 183,
 'DESIGN': 184,
 'RELATIVE VALUES': 185,
 'MUSIC': 186,
 'GOLF': 187,
 'POTOMAC WATCH': 188,
 'PRO CENTRAL BANKS NEWSLETTER': 189,
 'NATIONAL SECURITY': 190,
 'UPWARD MOBILITY': 191,
 'EUROPE MARKETS': 192,
 'CENTRAL BANKS COMMENTARY': 193,
 'TECHNOLOGY ESSENTIALS': 194,
 "WHAT'S YOUR WORKOUT?": 195,
 'CRIME': 196,
 'THE SATURDAY ESSAY': 197,
 'OLYMPICS': 198,
 'THE ARTIST': 199,
 'ARTS & ENTERTAINMENT': 200,
 'EAST IS EAST': 201}
```

In [42]:
```
# If an article has category Markets then output should be [0., 1., 0.,
 ..., 0., 0., 0.]
one_hot_embedding = np.eye(202)
one_hot_embedding
```

Out[42]:
```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.]])
```

In [424]:
```
# What is the maximum number of categories assigned to one article
sorted([len(category[i]) for i in range(1,len(category)+1)])[-1]
```

Out[424]:
```
3
```

In [43]:
```
def y_padder(x,max_len=3):
    """
    Idea is similar to that of padder. This time, padding is done to out
put.
    """
    if len(x)>=max_len:
        return list(map(lambda y: y_dict[y] if y in y_dict.keys() else 0
,x[:max_len]))
    else:
        tmp = list(np.zeros(max_len,int))
        tmp[:len(x)]=list(map(lambda y: y_dict[y] if y in y_dict.keys()
else 0,x))
        return tmp
```

In [45]:
```
# These are actual categories per article. I want to transform these to
 sum of one hot vectors per article
y_padded = [y_padder(x) for x in category_intersection.values()]
y_padded[:10]
```

Out[45]:
```
[[1, 2, 3],
 [1, 4, 0],
 [5, 0, 0],
 [6, 7, 0],
 [1, 8, 0],
 [9, 10, 0],
 [11, 0, 0],
 [1, 12, 0],
 [6, 13, 0],
 [1, 14, 0]]
```

In [176]:
```
for i in range(3):
    exec(f'y_padded_{i}=np.array(y_padded)[:,{i}]')
    exec(f'y_{i} = one_hot_embedding[y_padded_{i}]')
y_data = y_0.reshape(-1,1,202) +  y_1.reshape(-1,1,202) +  y_2.reshape(-
1,1,202)
y_data
```

Out[176]:
```
array([[[0., 1., 1., ..., 0., 0., 0.]],

       [[1., 1., 0., ..., 0., 0., 0.]],

       [[2., 0., 0., ..., 0., 0., 0.]],

       ...,

       [[2., 0., 0., ..., 0., 0., 0.]],

       [[2., 0., 0., ..., 0., 0., 0.]],

       [[2., 1., 0., ..., 0., 0., 0.]]])
```

In [198]:
```
for i in range(3):
    exec(f'y_padded_{i}=np.array(y_padded)[:,{i}]')
    exec(f'y_{i} = one_hot_embedding[y_padded_{i}]')
y_data = y_0[:,1:].reshape(-1,1,201) +  y_1[:,1:].reshape(-1,1,201) +  y
_2[:,1:].reshape(-1,1,201)
y_data
```

Out[198]:
```
array([[[1., 1., 1., ..., 0., 0., 0.]],

       [[1., 0., 0., ..., 0., 0., 0.]],

       [[0., 0., 0., ..., 0., 0., 0.]],

       ...,

       [[0., 0., 0., ..., 0., 0., 0.]],

       [[0., 0., 0., ..., 0., 0., 0.]],

       [[1., 0., 0., ..., 0., 0., 0.]]])
```

In [199]:
```
X_data = X_batch.reshape(-1,1,n_inputs)
```

3.2 Basic LSTM implementation

In [264]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test
_size=0.33, random_state=42)

# Reset graph
reset_graph()
n_steps=1
n_neurons = 201
n_outputs=201
learning_rate = 0.001

# Input data.
X = tf.placeholder(tf.float32, shape=[None, 1, n_inputs])
y = tf.placeholder(tf.float32, shape=[None,1,n_outputs])

# Specify model
cell = tf.nn.rnn_cell.BasicLSTMCell(num_units=n_neurons, activation=tf.n
n.relu)
outputs, states = tf.nn.dynamic_rnn(cell, X, dtype=tf.float32)

# Specify optimization routine
loss = tf.reduce_mean(tf.square(outputs - y)) # MSE
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

init = tf.global_variables_initializer()
saver = tf.train.Saver()

n_iterations = 1500
LSTM_MSE = []
with tf.Session() as sess:
    init.run()
    for iteration in range(n_iterations):
        #X_batch, y_batch = next_batch(batch_size, n_steps)
        sess.run(training_op, feed_dict={X: X_train, y: y_train})
        if iteration % 100 == 0:
            mse = loss.eval(feed_dict={X: X_train, y: y_train})
            print(iteration, "\tMSE:", mse)
            LSTM_MSE += [mse]

    saver.save(sess, "./LSTM_model") # not shown in the book
```

```
0       MSE: 0.0069662314
100     MSE: 0.0022923048
200     MSE: 0.0018345
300     MSE: 0.0017527033
400     MSE: 0.0017314882
500     MSE: 0.0017257062
600     MSE: 0.001724131
700     MSE: 0.0017236957
800     MSE: 0.0017235681
900     MSE: 0.001723527
1000    MSE: 0.0017235369
1100    MSE: 0.0017592217
1200    MSE: 0.0017235128
1300    MSE: 0.0017235104
1400    MSE: 0.0017235033
```

In [272]:

```
with tf.Session() as sess:
    saver.restore(sess, "./LSTM_model")
    y_lstm_pred = sess.run(outputs, feed_dict={X: X_test})
    lstm_test_mse = loss.eval(feed_dict={X: X_test, y: y_test})
print(lstm_test_mse)
```

```
0.005999455
```

In [266]:

```
predict=[]
for i in range(y_lstm_pred.shape[0]):
    n_positive = len(y_lstm_pred[i][(y_pred[i])>0])
    if n_positive < 3:
        predict+=[list((-y_lstm_pred[i]).argsort()[0][:n_positive])+[0*x
for x in range(3-n_positive)]]
    else:
        predict+=[list((-y_lstm_pred[i]).argsort()[0][:3])]

result_exc_pad = []
result_bool_exc_pad = []
result_bool = []
result = []
for i in range(y_test.shape[0]):
    result_exc_pad += [set(predict[i]) & set(y_padded[i]) ^ {0}]
    result_bool_exc_pad += [len(set(predict[i]) & set(y_padded[i]) ^ {0
})>0]
    result += [set(predict[i]) & set(y_padded[i])]
    result_bool += [len(set(predict[i]) & set(y_padded[i]))>0]
```

In [267]:

```
np.array(result_bool).sum()/y_test.shape[0]
```

Out[267]:

```
0.8275648949320148
```

In [268]:

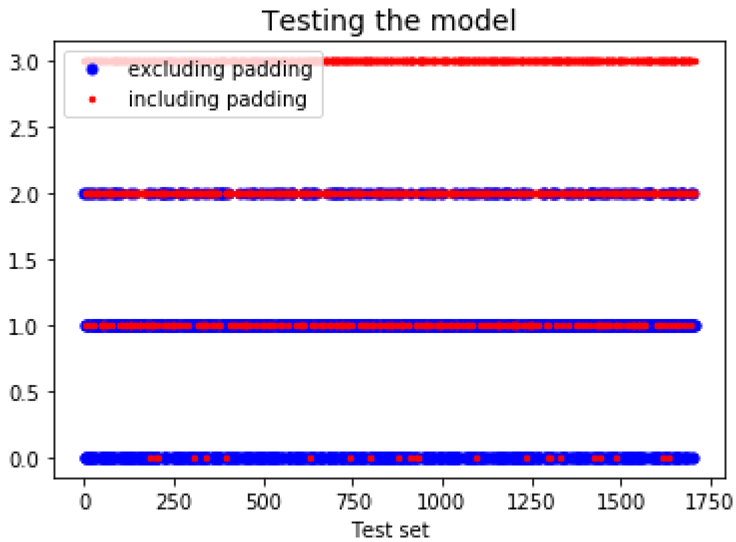
```
np.array(result_bool_exc_pad).sum()/y_test.shape[0]
```

Out[268]:

```
0.23300370828182942
```

In [672]:
```
plt.title("Testing the model", fontsize=14)
plt.plot(x,sum1, "bo", markersize=5, label="excluding padding")
plt.plot(x, sum2, "r.", markersize=5, label="including padding")
plt.legend(loc="upper left")
plt.xlabel("Test set")

plt.show()
```



3.3 Basic RNN Implementation

In [203]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test
_size=0.33, random_state=42)

# Reset graph
reset_graph()
n_steps=1
n_neurons = 201
n_outputs=201
learning_rate = 0.001

# Input data.
X = tf.placeholder(tf.float32, shape=[None, 1, n_inputs])
y = tf.placeholder(tf.float32, shape=[None,1,n_outputs])

# Specify model
cell = tf.nn.rnn_cell.BasicRNNCell(num_units=n_neurons, activation=tf.nn
.relu)
outputs, states = tf.nn.dynamic_rnn(cell, X, dtype=tf.float32)

# Specify optimization route
loss = tf.reduce_mean(tf.square(outputs - y)) # MSE
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

init = tf.global_variables_initializer()
saver = tf.train.Saver()

n_iterations = 1500
LSTM_MSE = []
with tf.Session() as sess:
    init.run()
    for iteration in range(n_iterations):
        #X_batch, y_batch = next_batch(batch_size, n_steps)
        sess.run(training_op, feed_dict={X: X_train, y: y_train})
        if iteration % 100 == 0:
            mse = loss.eval(feed_dict={X: X_train, y: y_train})
            print(iteration, "\tMSE:", mse)
            RNN_MSE += [mse]

    saver.save(sess, "./RNN_model") # not shown in the book
```

```
0       MSE: 0.0072795944
100     MSE: 0.005528547
200     MSE: 0.005113464
300     MSE: 0.0049441718
400     MSE: 0.004867676
500     MSE: 0.004831889
600     MSE: 0.0048150523
700     MSE: 0.004807169
800     MSE: 0.0048034983
900     MSE: 0.0048017907
1000    MSE: 0.0048009926
1100    MSE: 0.0048006154
1200    MSE: 0.004800432
1300    MSE: 0.0048003397
1400    MSE: 0.0048002903
```

In [259]:

```
with tf.Session() as sess:                          # not shown in the b
ook
    saver.restore(sess, "./RNN_model")    # not shown
    y_rnn_pred = sess.run(outputs, feed_dict={X: X_test})
```

In [260]:

```
predict=[]
for i in range(y_rnn_pred.shape[0]):
    n_positive = len(y_rnn_pred[i][(y_pred[i])>0])
    if n_positive < 3:
        predict+=[list((-y_rnn_pred[i]).argsort()[0][:n_positive])+[0*x
for x in range(3-n_positive)]]
    else:
        predict+=[list((-y_rnn_pred[i]).argsort()[0][:3])]

result_exc_pad = []
result_bool_exc_pad = []
result_bool = []
result = []
for i in range(y_test.shape[0]):
    result_exc_pad += [set(predict[i]) & set(y_padded[i]) ^ {0}]
    result_bool_exc_pad += [len(set(predict[i]) & set(y_padded[i]) ^ {0
})>0]
    result += [set(predict[i]) & set(y_padded[i])]
    result_bool += [len(set(predict[i]) & set(y_padded[i]))>0]
```

In [261]:

```
np.array(result_bool).sum()/y_test.shape[0]
```

Out[261]: 0.9375772558714462

In [262]:

```
np.array(result_bool_exc_pad).sum()/y_test.shape[0]
```

Out[262]: 0.0723114956736712

In [ ]:

In [ ]: