Final Project

Case Study in Sentiment Analysis

Cheruvu, Ria

CSCI E-89 Deep Learning

Harvard University Extension School

Prof. Zoran B. Djordjević

Problem Statement

- In this project, I will build a convolutional neural network that can recognize individuals' emotions/opinions on self-driving cars from tweets, textual messages users post on the social media website Twitter, using **sentiment analysis**.
- I will use pre-trained Word2Vec and GloVe word embeddings, which will be inputted into a convolutional neural network built using the Keras API for recognizing emotions from sample tweets.
- I will use the dataset from Crowdflower available at https://data.world/crowdflower/sentiment-self-driving-cars to generate the Word2Vec embeddings and train the model, and pre-trained GloVe word vectors generated using a large Twitter corpus, available on the https://nlp.stanford.edu/projects/glove/webpage.
- I chose this problem because companies, researchers, etc. can learn more about individuals' emotions/opinions on self-driving cars using this type of technology.

Description of Software and Technology

- The Anaconda platform/distribution (version 4.4.10), downloaded from this URL: https://www.anaconda.com/download/.
- Jupyter Notebook
- Keras, a high-level neural networks API
- Word2Vec
- GloVe
- TensorFlow
- Packages:
  - pandas
  - matplotlib
  - h5py
  - nltk
  - genism
  - scikit-learn
  - The Natural Language Toolkit (NLTK)

My Hardware Environment

- Operating Systems: Windows 10 Home

Device specifications

HP ENVY Curved All-in-One PC 34-b0xx

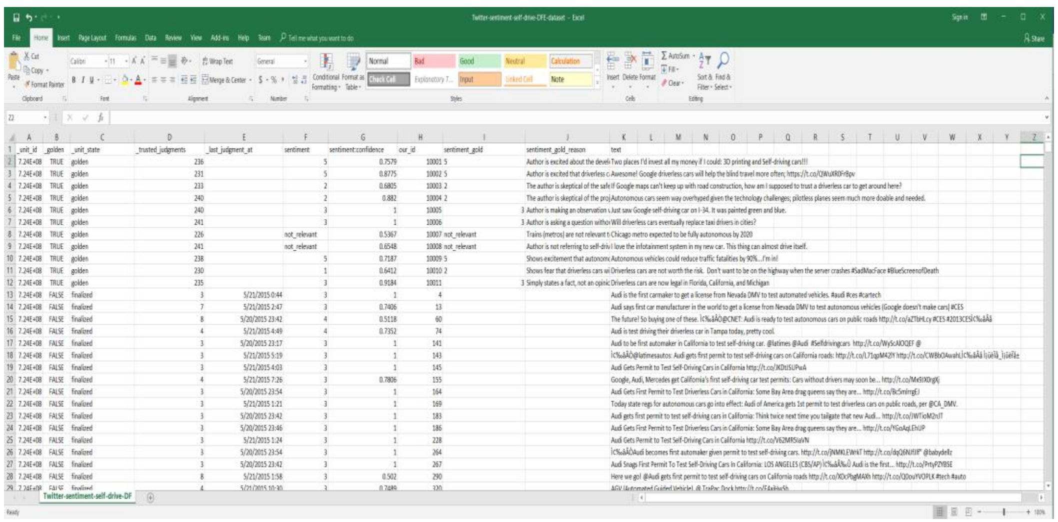
Device name	DESKTOP-0FG6UFD
Processor	Intel(R) Core(TM) i7-7700T CPU @ 2.90GHz 2.90 GHz
Installed RAM	16.0 GB (15.9 GB usable)
Device ID	07ADA2D2-C555-4A47-A00B-F28130E9FFCF
Product ID	00325-96260-70851-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Windows specifications

Edition	Windows 10 Home

Description of Data

- The dataset, titled Twitter sentiment analysis: Self-driving cars, is downloaded from https://data.world/crowdflower/sentiment-self-driving-cars
- There are 11 columns in the original dataset, with 7156 rows
- The size of the dataset is 1.15 MB
- I am using two columns: "Sentiment" column contains values and "Text" column contains the tweets.



	sentiment	text
0	5	Two places I'd invest all my money if I could: 3D printing and Self-driving cars!!!
1	5	Awesome! Google driverless cars will help the blind travel more often; https://t.co/QWuXR0FrBpv
2	2	If Google maps can't keep up with road construction, how am I supposed to trust a driverless car to get around here?
3	2	Autonomous cars seem way overhyped given the technology challenges; pilotless planes seem much more doable and needed.
4	3	Just saw Google self-driving car on I-34. It was painted green and blue.
5	3	Will driverless cars eventually replace taxi drivers in cities?
8	5	Autonomous vehicles could reduce traffic fatalities by 90%...I'm in!
9	1	Driverless cars are not worth the risk. Don't want to be on the highway when the server crashes #SadMacFace #BlueScreenofDeath
10	3	Driverless cars are now legal in Florida, California, and Michigan
11	3	Audi is the first carmaker to get a license from Nevada DMV to test automated vehicles. #audi #ces #cartech
12	3	Audi says first car manufacturer in the world to get a license from Nevada DMV to test autonomous vehicles (Google doesn't make cars) #CES
13	4	The future! So buying one of these. Ì¢‰âÂÒ@CNET: Audi is ready to test autonomous cars on public roads http://t.co/aZTbHLcy #CES #2013CESÌ¢‰âÂå
14	4	Audi is test driving their driverless car in Tampa today, pretty cool.
15	3	Audi to be first automaker in California to test self-driving car. @latimes @Audi #Selfdrivingcars http://t.co/WyScAlOQEF @
16	3	Ì¢‰âÂÒ@latimesautos: Audi gets first permit to test self-driving cars on California roads: http://t.co/L71qpM42IY http://t.co/CWBbOAwahLÌ¢‰âÂå Ì¡üèÏå_Ì¡üèÏå±
17	3	Audi Gets Permit to Test Self-Driving Cars in California http://t.co/JKDtJSUPwA
18	3	Google, Audi, Mercedes get California's first self-driving car test permits: Cars without drivers may soon be... http://t.co/Mx9JX0rgXj
19	3	Audi Gets First Permit to Test Driverless Cars in California: Some Bay Area drag queens say they are... http://t.co/Bc5mlrrgEJ
20	3	Today state regs for autonomous cars go into effect: Audi of America gets 1st permit to test driverless cars on public roads, per @CA_DMV.
21	3	Audi gets first permit to test self-driving cars in California: Think twice next time you tailgate that new Audi... http://t.co/JWTioM2nJT
22	3	Audi Gets First Permit to Test Driverless Cars in California: Some Bay Area drag queens say they are... http://t.co/YGoAqLEhUP
23	3	Audi Gets Permit to Test Self-Driving Cars in California http://t.co/V62MR5laVN
24	3	Ì¢‰âÂÒAudi becomes first automaker given permit to test self-driving cars. http://t.co/jNMKLEWrkT http://t.co/dqQ6NJfJlf" @babydellz
25	3	Audi Snags First Permit To Test Self-Driving Cars In California: LOS ANGELES (CBS/AP) Ì¢‰âÂ‰Û Audi is the first... http://t.co/PrtyPZYBSE

Overview of Steps

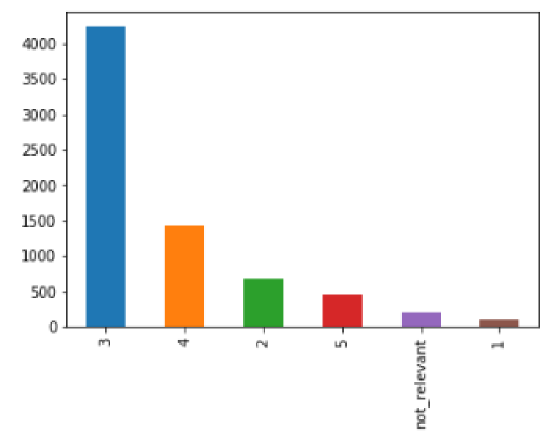
1. Install Anaconda, TensorFlow, Jupyter Notebook, Pandas, matplotlib, h5py, nltk, keras, genism, and scikit-learn
2. Preprocess the data
3. Generate Word2Vec embeddings
4. Input Word2Vec embeddings and GloVe embeddings into Convolutional Neural Network built with Keras API

Data Set Overview

- Values of the “Sentiment” column:
  - Very positive – 5
    - Example tweet: “Two places I'd invest all my money if I could: 3D printing and Self-driving cars!!!”
  - Slightly positive – 4
    - Example tweet: “Audi is test driving their driverless car in Tampa today, pretty cool.”
  - Neutral – 3
    - Example tweet: “Driverless cars are now legal in Florida, California, and Michigan”
  - Slightly negative – 2
    - Example tweet: “If i need to constantly supervise the car it's not autonomous #vlabauto”
  - Very negative – 1
    - Example tweet: “Driverless cars are not worth the risk. Don't want to be on the highway when the server crashes #SadMacFace #BlueScreenofDeath”

In [4]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
senti['sentiment'].value_counts().plot(ax=ax, kind='bar')
```

Out[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1df12bbbdd8>
```



Data Set Preparation

- Remove “not_relevant” values
- Use regex to remove:
  - URL links,
  - @username links,
  - #links,
  - additional whitespaces,
  - numbers/digits,
  - special characters,
  - and stop words with regular expressions (in conjunction with NLTK).
- I took and modified code for the regex operations
- I took and modified code for expanding on Contractions
- Expand on Contractions
- Tokenize

The future! So buying one of these. Ì¢‰âÂÒ@CNET: Audi is ready to test autonomous cars on public roads http://t.co/aZTbHLcy #CES #2013CESÌ¢‰âÂå•

future buying one audi ready test autonomous cars public roads ces

```
#Removing the URL Links with a regular expresion.
text_col = text_col.replace(r'((www\.[^\s]+)|(https?://[^\s]+))', '', regex=True)
#Removing the @username Links with a regular expresion.
text_col = text_col.replace('@[^\s]+', '', regex=True)
#Some tweets involve the use of the at (@) symbol, whitespace, and then a word, su
#Here we are removing the @ username Links with a regular expresion.
text_col = text_col.replace('@ [^\s]+', '', regex=True)
#Replacing the #Links with a regular expresion.
text_col = text_col.replace(r'#([^\s]+)', r'\1', regex=True)
```

```
#Removing additional whitespaces
text_col = text_col.replace('[\s]+', ' ', regex=True)
#Remove all digits
text_col = text_col.replace(r'\w*\d\w*', '', regex=True)
```

```
#Expanding contractions
#(code taken from https://stackoverflow.com/questions/43018030/replace-appostrophe-
text_col = text_col.replace(r"won't", "will not", regex=True)
text_col = text_col.replace(r"can\'t", "can not", regex=True)
text_col = text_col.replace(r"n\'t", " not", regex=True)
text_col = text_col.replace(r"\'re", " are", regex=True)
text_col = text_col.replace(r"\'s", " is", regex=True)
text_col = text_col.replace(r"\'d", " would", regex=True)
text_col = text_col.replace(r"\'ll", " will", regex=True)
text_col = text_col.replace(r"\'t", " not", regex=True)
text_col = text_col.replace(r"\'ve", " have", regex=True)
text_col = text_col.replace(r"\'m", " am", regex=True)
text_col
```

Generating Word2Vec embeddings

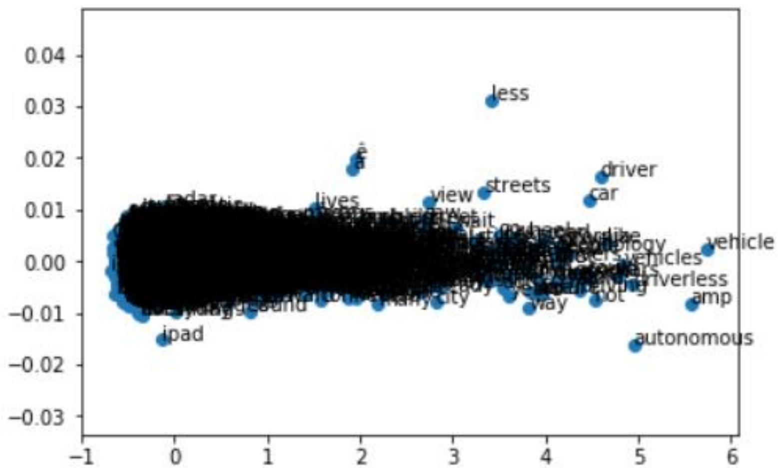
- Generate Word2Vec embeddings:

  model = Word2Vec(array, size=100, window=5, workers=8, **min_count**=2)

- Vocabulary size of 3882
- I varied the min_count parameter, which had an impact on the cosine similarity scores and the vocabulary size

```
In [33]: model.wv.most_similar(negative=['markets'])

Out[33]: [('chart', 0.44268137216567993),
          ('predictions', 0.4416291415691376),
          ('diver', 0.38253650069236755),
          ('distraction', 0.013044580817222595),
          ('predator', 0.0037798210978507996),
          ('showing', -0.1869221180677414),
          ('backtothefuturefan', -0.2363729178905487),
          ('finest', -0.26595252752304077),
          ('autonomus', -0.27073991298675537),
          ('opt', -0.3551793098449707)]
```



Using the pre-trained Word2Vec embeddings

- Preparing the train and test datasets
  - Split train and test datasets
  - Created and fit the Keras Tokenizer on the training dataset
  - Encode and pad the sequences of the **training and test datasets**.
  - One hot encode the output labels
- Preparing the train and test datasets
  - Loading the model for computing an index mapping words to known embeddings
  - Creating a weight matrix for the Embedding layer and loading the embedding matrix into the Embeddings layer
  - Build the CNN using the Keras API, which incorporates 3 Conv1d layers, 3 Max Pooling layers, 1 Flatten layer, 1 Dropout layer, and 2 Dense layers (one of which has L2 regularization).

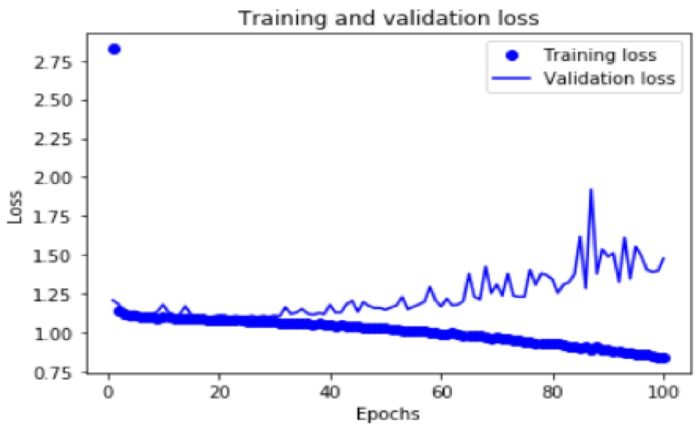
```
from keras import regularizers
model2 = Sequential()
model2.add(embedding_layer)
model2.add(Conv1D(150, 3, activation='relu'))
model2.add(MaxPooling1D(pool_size=2))
model2.add(Conv1D(150, 3, activation='relu'))
model2.add(MaxPooling1D(pool_size=2))
model2.add(Conv1D(150, 3, activation='relu'))
model2.add(MaxPooling1D(pool_size=1))
model2.add(Flatten())
model2.add(Dropout(0.8)) #I added this
model2.add(Dense(150, kernel_regularizer=regularizers.l2(0.1), activation='relu'))
model2.add(Dense(6, activation='softmax'))
```

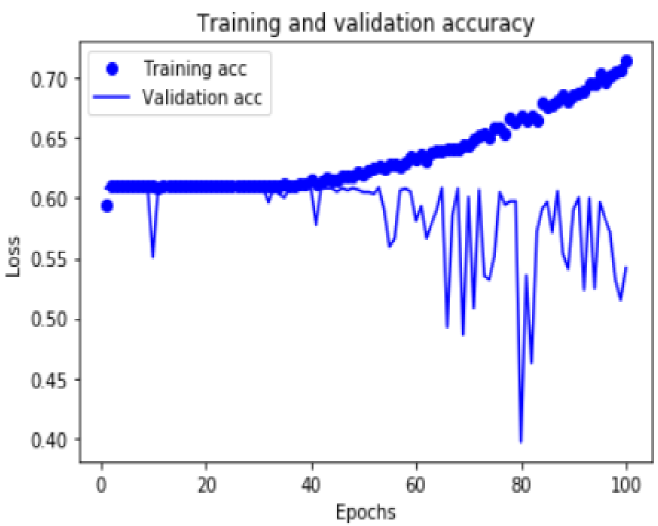
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 21, 100)           770400
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 19, 150)           45150
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 9, 150)            0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 7, 150)            67650
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 3, 150)            0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 1, 150)            67650
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 1, 150)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 150)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_1 (Dense)              (None, 150)               22650
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 906
=================================================================
Total params: 974,406
Trainable params: 204,006
Non-trainable params: 770,400
_________________________________________________________________
None
```

Using the pre-trained Word2Vec embeddings

```
Epoch 83/100
 - 1s - loss: 0.9166 - acc: 0.6654 - val_loss: 1.3242 - val_acc: 0.5720
Epoch 84/100
 - 1s - loss: 0.9119 - acc: 0.6798 - val_loss: 1.3778 - val_acc: 0.5910
Epoch 85/100
 - 1s - loss: 0.9037 - acc: 0.6764 - val_loss: 1.6205 - val_acc: 0.5970
Epoch 86/100
 - 1s - loss: 0.9107 - acc: 0.6775 - val_loss: 1.2860 - val_acc: 0.5715
Epoch 87/100
 - 1s - loss: 0.8967 - acc: 0.6809 - val_loss: 1.9224 - val_acc: 0.6060
Epoch 88/100
 - 1s - loss: 0.9075 - acc: 0.6854 - val_loss: 1.3789 - val_acc: 0.5540
Epoch 89/100
 - 1s - loss: 0.8958 - acc: 0.6815 - val_loss: 1.5366 - val_acc: 0.5410
Epoch 90/100
 - 1s - loss: 0.8929 - acc: 0.6868 - val_loss: 1.4905 - val_acc: 0.5905
Epoch 91/100
 - 1s - loss: 0.8863 - acc: 0.6885 - val_loss: 1.5111 - val_acc: 0.6005
Epoch 92/100
 - 1s - loss: 0.8812 - acc: 0.6894 - val_loss: 1.3258 - val_acc: 0.5235
Epoch 93/100
 - 1s - loss: 0.8736 - acc: 0.6953 - val_loss: 1.6120 - val_acc: 0.6000
Epoch 94/100
 - 1s - loss: 0.8732 - acc: 0.6964 - val_loss: 1.3467 - val_acc: 0.5245
Epoch 95/100
 - 1s - loss: 0.8655 - acc: 0.7037 - val_loss: 1.5539 - val_acc: 0.5965
Epoch 96/100
 - 1s - loss: 0.8631 - acc: 0.6978 - val_loss: 1.4950 - val_acc: 0.5830

Epoch 97/100
 - 1s - loss: 0.8579 - acc: 0.7023 - val_loss: 1.4056 - val_acc: 0.5715
Epoch 98/100
 - 1s - loss: 0.8547 - acc: 0.7054 - val_loss: 1.3901 - val_acc: 0.5320
Epoch 99/100
 - 1s - loss: 0.8467 - acc: 0.7071 - val_loss: 1.3978 - val_acc: 0.5150
Epoch 100/100
 - 1s - loss: 0.8428 - acc: 0.7147 - val_loss: 1.4786 - val_acc: 0.5420
[1.3294213887510582, 0.5802735782853985]
Test Accuracy: 58.027358
```





Using the pre-trained Word2Vec embeddings

Experimental tweet text: 'Unnecessary waste of time. Creepy and freaky. Not worth it.'
Predicted: Slightly Negative

Experimental tweet text: ‘Whoever did this completely wasted their time. #sadmacface’
Predicted: Neutral

Experimental tweet text: ‘GENIUS! AWESOME! Who thought of this? They are amazing people.’
Predicted: Very Positive

Experimental tweet text: 'Self-driving cars are murderers that kill people. I HATE them, they suck, and I am NOT driving them.'
Predicted: Very Negative

Using the pre-trained GloVe embeddings

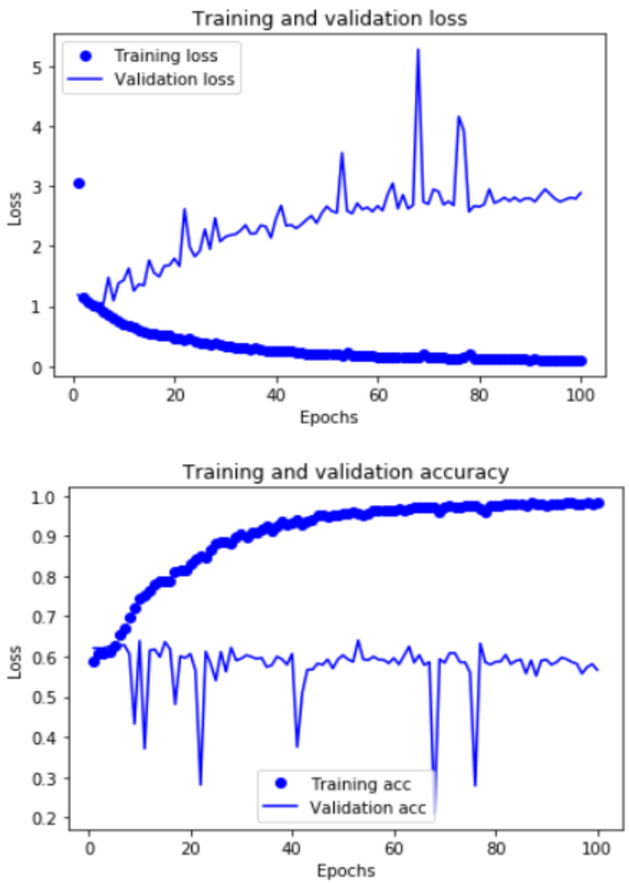
- The GloVe algorithm was **not** trained on the dataset obtained from CrowdFlower
- The creators of GloVe provided pre-trained word vectors obtained by generating GloVe word embeddings on a large Twitter corpus.
- The same code used for using the pre-trained Word2Vec embeddings (with minor modifications) was used for:
  - Loading the model for computing an index mapping words to known embeddings
  - Creating a weight matrix for the Embedding layer and loading the embedding matrix into the Embeddings layer
  - Build the CNN using the Keras API

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 26, 100)           762300
_________________________________________________________________
conv1d_16 (Conv1D)           (None, 24, 150)           45150
_________________________________________________________________
max_pooling1d_16 (MaxPooling (None, 12, 150)           0
_________________________________________________________________
conv1d_17 (Conv1D)           (None, 10, 150)           67650
_________________________________________________________________
max_pooling1d_17 (MaxPooling (None, 5, 150)            0
_________________________________________________________________
conv1d_18 (Conv1D)           (None, 3, 150)            67650
_________________________________________________________________
max_pooling1d_18 (MaxPooling (None, 3, 150)            0
_________________________________________________________________
flatten_6 (Flatten)          (None, 450)               0
_________________________________________________________________
dense_11 (Dense)             (None, 150)               67650
_________________________________________________________________
dense_12 (Dense)             (None, 6)                 906
=================================================================
Total params: 1,011,306
Trainable params: 249,006
Non-trainable params: 762,300
_________________________________________________________________
None
```

Using the pre-trained GloVe embeddings

```
Epoch 83/100
 - 1s - loss: 0.1242 - acc: 0.9795 - val_loss: 2.7155 - val_acc: 0.5810
Epoch 84/100
 - 1s - loss: 0.1170 - acc: 0.9806 - val_loss: 2.7539 - val_acc: 0.5895
Epoch 85/100
 - 1s - loss: 0.1246 - acc: 0.9781 - val_loss: 2.8073 - val_acc: 0.5925
Epoch 86/100
 - 1s - loss: 0.1170 - acc: 0.9772 - val_loss: 2.7469 - val_acc: 0.5580
Epoch 87/100
 - 1s - loss: 0.1123 - acc: 0.9820 - val_loss: 2.8107 - val_acc: 0.5905
Epoch 88/100
 - 1s - loss: 0.1148 - acc: 0.9800 - val_loss: 2.7433 - val_acc: 0.5520
Epoch 89/100
 - 1s - loss: 0.1157 - acc: 0.9797 - val_loss: 2.7899 - val_acc: 0.5900
Epoch 90/100
 - 1s - loss: 0.1098 - acc: 0.9772 - val_loss: 2.7956 - val_acc: 0.5925
Epoch 91/100
 - 1s - loss: 0.1138 - acc: 0.9809 - val_loss: 2.7391 - val_acc: 0.5800
Epoch 92/100
 - 1s - loss: 0.1046 - acc: 0.9809 - val_loss: 2.8433 - val_acc: 0.5860
Epoch 93/100
 - 1s - loss: 0.1092 - acc: 0.9795 - val_loss: 2.9459 - val_acc: 0.5975
Epoch 94/100
 - 1s - loss: 0.1050 - acc: 0.9828 - val_loss: 2.8629 - val_acc: 0.5935
Epoch 95/100
 - 1s - loss: 0.1065 - acc: 0.9820 - val_loss: 2.7827 - val_acc: 0.5865
Epoch 96/100
 - 1s - loss: 0.1055 - acc: 0.9811 - val_loss: 2.7373 - val_acc: 0.5820

Epoch 97/100
 - 1s - loss: 0.1046 - acc: 0.9817 - val_loss: 2.7743 - val_acc: 0.5575
Epoch 98/100
 - 1s - loss: 0.1002 - acc: 0.9823 - val_loss: 2.8067 - val_acc: 0.5735
Epoch 99/100
 - 1s - loss: 0.1006 - acc: 0.9817 - val_loss: 2.7863 - val_acc: 0.5815
Epoch 100/100
 - 1s - loss: 0.1005 - acc: 0.9828 - val_loss: 2.8821 - val_acc: 0.5670
[2.8469228559127377, 0.5593952484015855]
Test Accuracy: 55.939525
```



Using the pre-trained GloVe embeddings

In [74]:
```
# load json and create model
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("model.h5")
print("Loaded model from disk")

labels = [[''], ['Very negative'], ['Slightly negative'], ['Neutral'], ['Slightly positive'], ['Very positive']]

exp_tests = ['Unnecessary waste of time. Creepy and freaky. Not worth it.']
encoded_exp = tokenizer.texts_to_sequences(exp_tests)
Xexp = pad_sequences(encoded_exp, maxlen=max_length, padding='post')

pred = model3.predict_classes(Xexp)
print(pred)
print(labels[pred[0]])
```

```
Loaded model from disk
[5]
['Very positive']
```

In [75]:
```
exp_tests = ['Whoever did this completely wasted their time. #sadmacface']
encoded_exp = tokenizer.texts_to_sequences(exp_tests)
Xexp = pad_sequences(encoded_exp, maxlen=max_length, padding='post')

pred = model3.predict_classes(Xexp)
print(pred)
print(labels[pred[0]])
```

```
[4]
['Slightly positive']
```

In [76]:
```
exp_tests = ['GENIUS! AWESOME! Who thought of this? They are amazing people.']
encoded_exp = tokenizer.texts_to_sequences(exp_tests)
Xexp = pad_sequences(encoded_exp, maxlen=max_length, padding='post')

pred = model3.predict_classes(Xexp)
print(pred)
print(labels[pred[0]])
```

```
[4]
['Slightly positive']
```

In [77]:
```
exp_tests = ['Self-driving cars are murderers that kill people. I HATE them, they suck, and I am NOT driving them.']
encoded_exp = tokenizer.texts_to_sequences(exp_tests)
Xexp = pad_sequences(encoded_exp, maxlen=max_length, padding='post')

pred = model3.predict_classes(Xexp)
print(pred)
print(labels[pred[0]])
```

```
[2]
['Slightly negative']
```

Lessons Learned & Pros/Cons

- It appears the model using the GloVe embeddings predicts negative sentiments as positive. Therefore, it seems that using Word2Vec models generated using the Crowdflower dataset, as opposed to GloVe embeddings generated on another large Twitter corpus, leads to better accuracy and more accurate predictions. **<u>However, this is not true in all cases.</u>** There are times when I found the Word2Vec model predicted these experimental tweets as Neutral, so these predictions are highly volatile and solid interpretations cannot based on them due to the unbalanced nature of the dataset.
- The steps used for pre-processing tweets for sentiment analysis are debated
- A larger corpus would benefit model performance.
- Using the Keras API allowed easy inputting of the Word2Vec embeddings and GloVe embeddings into the CNN. I personally preferred Keras' text-preprocessing functions compared to NLTK's text-preprocessing functions.
- Next steps include more extensive tuning of the CNN to achieve higher model accuracy, training/testing the model on a larger dataset, and comparing the differences between Word2Vec and GloVe.

References

- "Step-by-Step Twitter Sentiment Analysis: Visualizing United Airlines' PR Crisis" (http://ipullrank.com/step-step-twitter-sentiment-analysis-visualizing-united-airlines-pr-crisis/)
- "Using pre-trained word embeddings in a Keras model" (https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html)
- https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/
- "Emotion Detection on Twitter Data using Knowledge Base Approach" (https://pdfs.semanticscholar.org/6698/5a996eab1e680ffdd88a4e92964ac4e7dd56.pdf)
- **Twitter sentiment analysis: Self-driving cars dataset available at:** https://data.world/crowdflower/sentiment-self-driving-cars.
- **Pre-trained GloVe word vectors that GloVe creators provided, available at:** https://nlp.stanford.edu/projects/glove/webpage.