You may notice that my project has changed. This is due to me not allocating the time to work with the professor I started to work with in the beginning. It turned out that the data was not in a form (pre-labeled) in quantities large enough to use and I did not realize this soon enough to find a solution before the end of the course and this project's deadline. Because of this I switched projects very last minute and used datasets I could easily work with but the lateness of my change shows in the quality of my work. This was a me problem, which is fine, I was happy to get to try out some of TensorFlow's newer features even if my results are poor. Feel free to include this page or not if this is put somewhere.

See https://www.dropbox.com/s/iwfkjonu7g8q3jb/submit.tar.gz?dl=0 for code.

Sentiment Analysis using LSTM Networks and their Effectiveness on Data Varying from the Training Domain

Thomas Stitt

cs224d

03/06/2016

Introduction

Sentiment is an expression of our opinion. "I like bread," shows a positive attitude towords bread whereas "vinegar is lame," doesn't make a vinegar newbie necessarily want to try it. These are both examples of sentiment, a positive sentiment in the first case and a negative in the second. Statements like "my bed sheets have stripes," are, unless you have and express some love or hatred of stripes, is without sentiment or in other words neutral. Google's handy language widget tells us that sentiment is "a view of or attitude toward a situation or event; an opinion," or "a feeling or emotion."

Humans can pick out sentiment but can computers? The answer we find is yes. We've seen an example of this in cs224d's problem set 3[1], a live demo from Stanford[2], and papers like [3] and [4]. In this project I wanted to explore, what I think, is an interesting question. How to networks trained in one area of opinion perform in another? Let's say you train a model to classify the sentiment of movie reviews, how does it do with opinion expressed in tweets? Or how does a model trained on journal entries work when classifying product reviews?

Recurrent neural networks (RNN) allow one to provide a sequence of inputs to a model that will apply a cell decription to each input while keeping some knowledge of what it so far has seen. This is more powerful than solely word-vector based methods like [5] because of this knowledge of word order. I also wanted to look at the `rnn` and `rnn_cell` objects in TensorFlow.

Problem Statement

Can we use recurrent networks of Long Short Term Memory (LSTM) cells to predict sentiment? How do models trained on (smaller) areas of opinion perform on other areas of opinion? This will also be an exercise in TensorFlow's `rnn_cell` objects so we will try to use those to generate the model.

Datasets

Three datasets were used in this project; the *UMICH SI650 Sentiment Classification*[6] dataset from *inclass.kaggle.com*, the *Twitter US Airline Sentiment*[7] from *kaggle.com*, and the *Sentiment Labelled Sentences Data Set*[8] from *UC Irvine's Machine Learning Repository*.

UMICH SI650 Sentiment Classification

- 639/772 negative/positive

    - There were many unlabeled values, which were dropped
- Data taken from opinmind.com
    - Authors point out this site is no longer active
- Positive example
    - *“PS da Vinci code is an awesome movie!.”*
- Negative example
    - *“Mission Impossible 2 sucked hardcore.”*

Twitter US Airline Sentiment

- 2363/2363 negative/positive
    - Originally 9178/2363
    - Randomly removed negatives to create a balanced dataset
    - Removed neutral entries
- Positive example
    - *“@JetBlue you guys operate a world class company and for that I thank you”*
- Negative example
    - *“@USAirways you've completely ruined a pro athlete's ability to prepare for a race. Now you won't help him get the bike he needs to the DR?!”*

Sentiment Labelled Sentences Data Set

- 3000/3000 negative/positive
    - 1000 in each positive/negative category from imdb, amazon, yelp
- Positive example
    - *“Really, really good rice, all the time.”*
- Negative example
    - *“I would have given no star if I was able.”*

Each dataset was split 70% train, 20% validate, 10% test. Certain special character were stripped out (‘~’ and ‘“‘) as ‘~’ was used as the delimiter and ‘“‘ when not escaped broke the tools (pandas) used to read the datasets.

Methods

A RNN will be used to classify the sentiment of a sequence of words. The rnn cell will be an LSTM cell to help to prevent the gradients from getting out of hand (vanishing/blowing up) when back propagating and to allow the cell to decide if it should or shouldn't use its hidden state or the previous hidden state. In this way the model can disregard (forget) information that (it learns) isn't useful.

Since the training sets are (somewhat) small we use fuzzy string comparison to find the 'closest' word in the embedding if the word is not already there. This is accomplished through the python library fuzzywuzzy[9]. Fuzzy comparisons can "match" strings like *time* and *times* since they probably have a similar embedding and could also match potential intentional misspellings like *like* and *liiiike* if someone really liked something or put a verbal pause into a written word.

I will be using TensorFlow's `rnn_cell`[10] objects to create the LSTM cells and using the `rnn`[11] function to apply this cell to the sequence.

Loss with be calculated as a reducton of the Cross Entropy (CE) loss of each word in the sequence (each word takes the label of the entire string when preforming the CE loss calculation).

Experimental Evaluation

I ran into some problems when trying my first model relating to the use of TensorFlow's `rnn` function in `rnn.py`. The feedforward phase of of the network was fine, no runtime complaints, but when it came time to compute the gradients with respect to trainable variables a runtime error was reported (tensor shape mismatch during a multiplication) when trying to update the embeddings. I ended up unrolling the rnn in a for loop but I plan to revisit the `rnn` function in more detail to see why the runtime error is occuring.

Using TensorFlow's `rnn` function:

```
outputs, states = tf.nn.rnn(cell=cell,
                  inputs=inputs,
                  initial_state=initial_state,
                  sequence_length=self.input_lengths)
```

"Unrolling":

```
outputs = []
```

```
with tf.variable_scope("rnn") as scope:
      for cycle in range(config.max_seq_len):
            if cycle > 0:
                  scope.reuse_variables()
            output, state = cell(inputs[:,cycle,:], state)
            outputs.append(output)
```

Since we are creating the model at initialization time and we reuse the model for multiple strings we have to have enough states for the longest input. This also means that shorter inputs will go through more states than they need to. One solution to this is to report the label for all states greater than the current sequence length to be unlabeled (`-1` in TensorFlow) so that the loss is 0 for these states. To note TensorFlow's `rnn` function allows a list of sequence lengths and handles variable length for you.

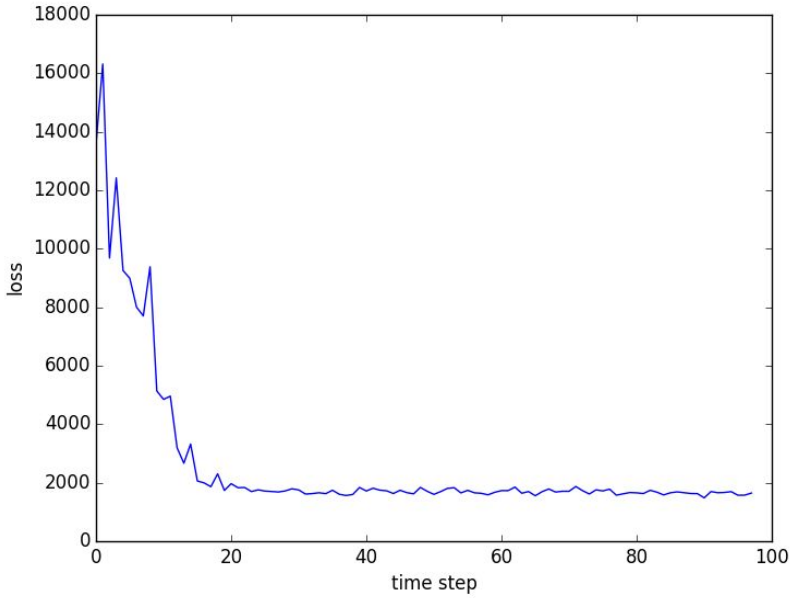


Figure 1: Annealed Sum Loss

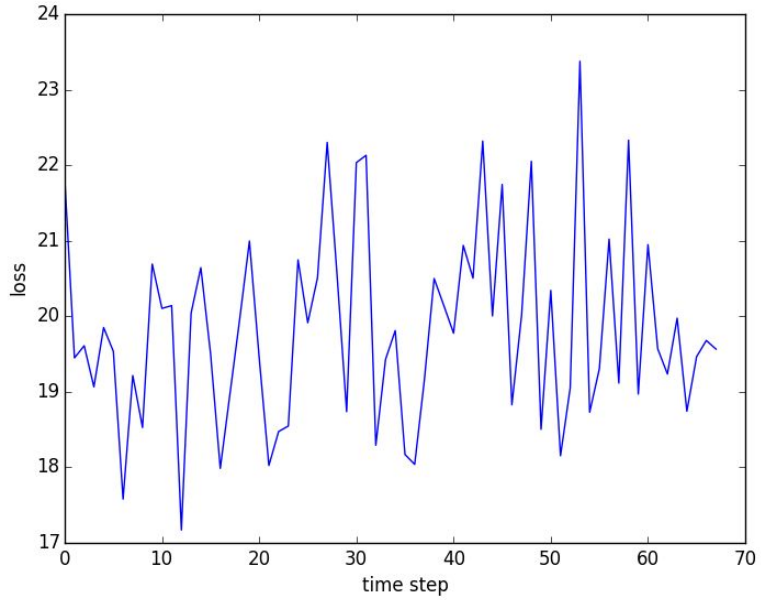


Figure 2: Annealed Average Loss

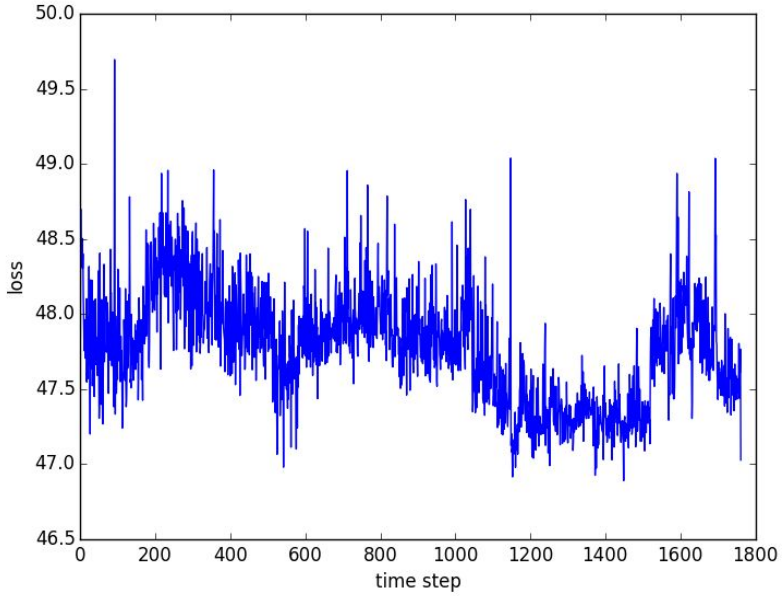
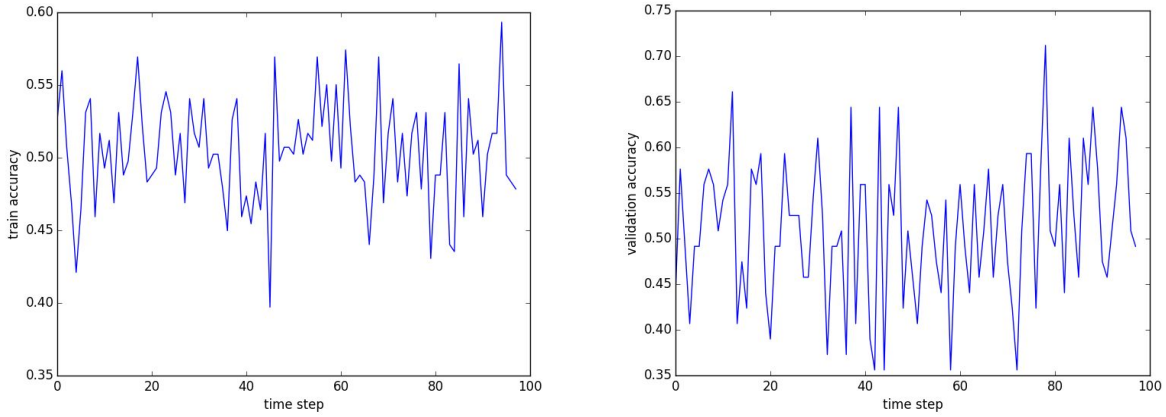


Figure 3: Non-Annealed Averaged Loss

Results were not very good. First, in figure 1, I summed the loss for each sequence. We see a reduction in loss but then an asymptotic region around a loss of 2000. The loss was summed over each sequence, which favors longer sequences in terms of training weight so instead in Figures 2 and 3 I used a mean. In figure two we decreased the learning rate if the current loss was greater than some fraction multiplied by the previous loss. I noticed that this happened a lot and caused the learning rate to tend towards 0. In figure 3 I kept the learning rate constant. We see that neither technique helped when attempting to fit a model.



Figures 4 & 5: Example training and validation dataset accuracy

Figures 4 and 5 show example accuracies over the training and validation datasets that I was seeing in the models from Figures 1, 2, and 3 (figure 1 in this case). Similar to the variance in loss, we see fluctuations around 50% accuracy. This suggests a poor model since there are only two classes and if a model randomly guessed a class it would achieve ~50% accuracy given the equal distribution of positive/negative labels over the dataset (except in the case of the UMICH dataset that slightly favors a positive guess).

It seemed like the models were skipping around so I tried a decreased learning rate (.0001 vs .01) and as we see in figure 6 the loss still jumped around, not falling much below 120.

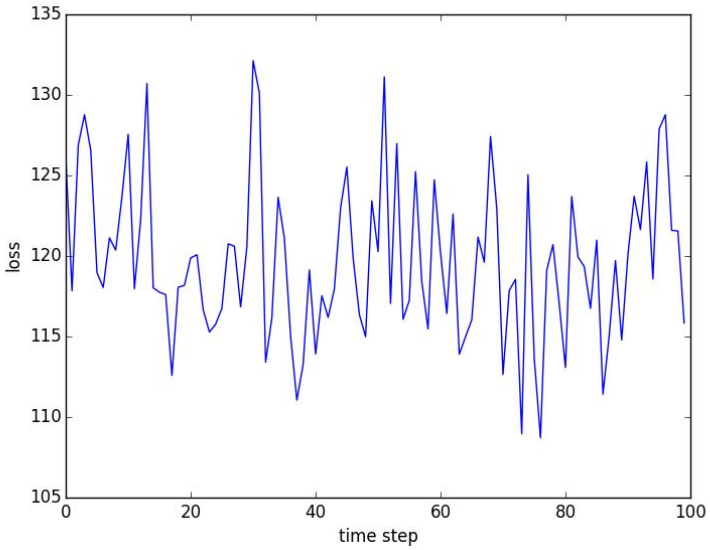


Figure 6: Small learning rate (`lr=.0001`)

Conclusion

I used an LSTM to attempt to train a sentiment classifiying model but came away unsuccessful in part of my orginal objective. I was able to implement a recurrent neural network in TensorFlow using part of the `rnn_cell` and `rnn` objects although the simple rnn routine did not end up working for me. Switching project ideas last minute was unideal but I am happy I was able to get some more experince in TensorFlow.
I realize now that a mean reduction over an array of losses will still take into account the length of the maximum sequence and therefore still be biased towards longer sequences.

References

[1] http://cs224d.stanford.edu/assignment3/index.html
[2] http://nlp.stanford.edu/sentiment/
[3] https://www.cs.cornell.edu/~oirsoy/files/emnlp14drnt.pdf
[4] http://www.aclweb.org/anthology/C14-1008
[5] http://www.aclweb.org/anthology/P11-1015
[6] https://inclass.kaggle.com/c/si650winter11/data
[7] https://www.kaggle.com/crowdflower/twitter-airline-sentiment
[8] https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences
[9] https://github.com/seatgeek/fuzzywuzzy
[10] https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/ops/rnn_cell.py
[11] https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/ops/rnn.py