Cullen Hauser
CSCI 490 – Data Science
3/9/2018

Introduction

With the rise of technology companies find themselves sitting on massive amounts of data describing all facets of their enterprise. To stay competitive companies now need to figure out how to interpret this data and use it to gain an upper hand on their competitors. Sometimes companies use this data to predict whether investments will succeed or fail, others use more generalized data to determine whether their millions of customers are happy or upset with a proposed business venture. To do this most use what is called a neural network. In the case of gauging public responses companies use a special neural network called an LSTM.

LSTM stands for long-short-term-memory, and is a form of recurrent neural network (RNN). Unlike typical neural networks RNNs use a string of inputs, with one input being fed in to a "cell" at a time. This cell has two outputs, a cell memory and a cell output. The cell memory is passed to the next cell in the line while the cell output is used to calculate the output of the overall network. You can see a depiction of an LSTM in figure 1.

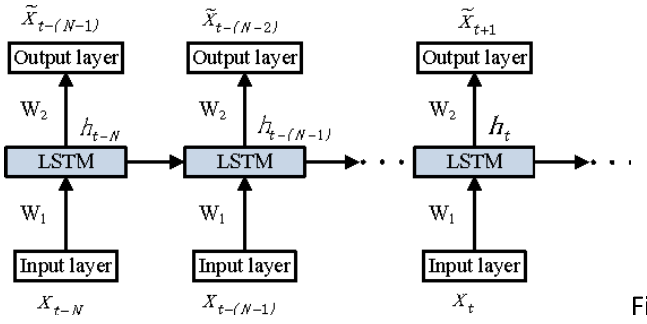


Figure 1

As described by Hong-Fang LSTMs are particularly adept at "remembering" features over long stretches of data. This means that LSTMs are very good at processing text because they remember meanings of words they read one or more steps prior. As you read this sentence you are not only processing each word individually, you are processing all the words before it to give the sentence its overall meaning. This is essentially what an LSTM does, except not quite as well as humans can.

The data that I will be using is from Kaggle and contains thousands of hotel reviews. Each entry provides the text of the review as well as the rating that the customer gave the hotel. I will be attempting to create a model capable of predicting the overall sentiment of the review, positive or negative, based solely on the text of the review. The goal is to create a model capable of at least 80% accuracy on a validation set. The reviews come with a rating of 1-5 stars but because classifying arbitrary reviews in to 5 'buckets' accurately is difficult, I will be classifying the reviews as either positive (1) or negative (0).

Cleaning the data

It's worth mentioning that this particular data set has almost twice as many positive reviews than it has negative. prevent the lopsided data from ruining the model I pared down the data set to ensure there are roughly as many positive as negative reviews before I even began to clean the data.

Some of the reviews in the data either have a rating on a scale of 1-10 or they have no rating at all. Those are very easy to filter out. Those were the first ones I eliminated as I transferred the data from a .csv to a .txt.

Due to the inherently large amount of words in the English language it is only natural to limit the network to only caring about the most frequent words in its data set. For this exercise I will be using a vocabulary of list of the top 15% most frequent words in the data set, with the rest of the words being replaced with 'UNKNOWN'. Some of the reviews in the data set are, however, not in English for these reviews I used the latitude and longitude to remove the data points that aren't from the United States. Some of the reviews posted in the United States aren't in English anyways, these reviews were removed by the program when it built its training set. The program removes any data point that has more than 50% of its text replaced with the unknown token.

To further clean the data I used the NLTK library in python to tokenize the sentences and then remove the stop words from each data set. This left each review with only the words that had the most effect on the overall meaning of the review. Once each data point was tokenized and cleaned, the words in its vocabulary were replaced with integers which translated to a vector that describes the meaning of that word to the network.

After cleaning the data, the program then generates its training set and its validation set. The training set ends up being 18,032 reviews, and the validation set is 3,183 reviews. The model uses a vocabulary size of 3,860 words. The least frequent word that it knows is 'google' which appears 12 times in the data set.

Training

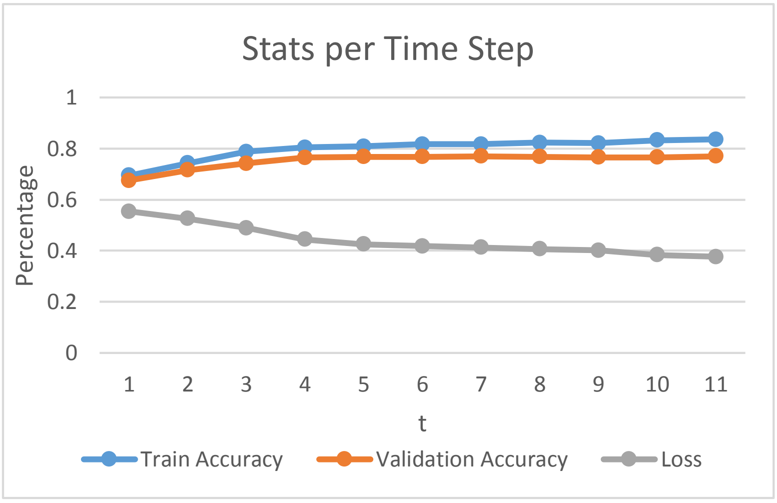
Learning Rate	.00075
Optimizer	Adam
Activation	softsign
Dimensions	72
Layers	2
Batch size	1

In addition to the information included to the left I used several other techniques in an attempt to boost the performance of the model. Most importantly instead of only using the output of the last perceptron in the LSTM I took the average of the outputs, multiplied that by an output weight matrix, and added a bias vector before finally performing logistic regression on the result to obtain the probability vector for each class. This results in a better representation of the overall sentiment of the review instead of being weighted towards whatever the last few words were. A similar version of this method is described here.

During the early stages of testing I discovered that the AI was somewhat prone to overfitting of the training set. To combat this I introduced input dropout. Basically what I did was gave each word in a given data point a 30% chance of being dropped during each epoch. So, instead of seeing a review like "There were cockroaches and mold everywhere" the model might be given "--- were cockroaches and ---- everywhere" and on the next epoch it may see "There were ---- and ---- everywhere". The dropout did not distinguish between key words and non-key words but instead gave each word an equal chance of being dropped. This proved to be a very effective method as instead of beginning to overfit and lose validation accuracy the model continued to reduce loss while maintaining or slowly gaining in validation accuracy. This is likely because instead of the model essentially memorizing the training set it is given an almost random data set each epoch. I'm positive there is a better method for using input dropout that doesn't include sometimes removing the most important words from an input but I'm very happy with the results that my little 10 line dropout function produced.

Results

I let the model train overnight with it set to stop training and save the model if the loss were to ever drop below .35. It never made it there and I manually stopped it the next morning with a loss of .37. While the model failed to reach my 80% validation accuracy goal, it got close with the ending validation accuracy being 77.03 %.



At around t = 3 the model began to lose some of its momentum in terms of accuracy gained. However it continued to steadily reduce its loss. I'm very happy with how close the validation accuracy and training accuracy ended up being as that's indicative of a model that has very little overfitting. I would have preferred them to be within one or two percentage points of each other rather than the actual gap of 5%. But this is acceptable.

Comparing my results with the results of others shows how my model stacks up against theirs. Luke Piglisi on Kaggle analyzed the same set using an SVM and a Naïve Bayes classifier, you can find his analysis here. However, instead of classifying the reviews as either positive or negative, he classified them in to their true ratings. He admits that most error occurred from the model predicting some 4 star reviews as 5 stars and vice versa. Correcting for that using our classification method we get the following validation accuracies:

	SVM (Stopwords)	SVM (No stopwords)	Naïve Bayes	My LSTM
Accuracy	80.72%	81.00%	65.11%	77.03%

While my LSTM model isn't bad per se, it could be improved by a lot. With a more robust dropout method, I could possibly squeeze out another percentage or two. One of the biggest issues with the data is the large amount of spelling errors and abbreviations. The most common misspelling of 'the', 'tge', appears in the data numerous times. The reviewers also alternated between spelling out renovations and abbreviating it to reno. Perhaps by messing around with some of the parameters, changing the hidden dimensions I could also get some more accuracy, but for now I'm happy with the results.