Formatting Instructions for NIPS 2013

Kyle G Griswold Department of Computer Science
Stanford University
Stanford, CA 94305
`kggriswo@stanford.edu`

Abstract

This paper attempts to construct a model to predict the helpfulness of online product reviews, specifically those coming from Amazon. I generated word vectors for each word, then averaged the word vectors together for each review and use a multi-layer neural network to model the helpfulness of a given review. I experimented with a variety of topologies and hyperparameters for this neural network, and include the results of these experiments within this paper.

1 Introduction

The ”helpfulness” of a review for an online product has become a standard way for customers to communally ward against irrelevent reviews, spammers, plants, and griefers. Without this metric, it is difficult for an online marketplace (like Amazon) to be able to show customers reviews that will actually help the customer decide whether they want a given product or not, which thus increases customer satisfaction, and the probability that the customer will come back to the marketplace. Unfortunatly, the reliability of the helpfulness metric is dependent upon a large number of people actually voting on the helpfulness of a given review, which leaves lesser known products and any reviews that are not shown close to the top of the list without a reliable helpfulness metric. This paper aims to solve this problem by constructing a model to predict the helpfulness of any given review.

In order to do this, we first generate word vectors from our corpous using standard natural language processing algorithms for doing so. We then average the word vectors for a given review together to get a representative vector for the review. We then run a multi-layer neural network on this vector to get a rating of the helpfulness of the review from 0 to 1. We train this neural network using standard deep-learning training techniques and the large data set of amazon reviews at [1].

2 Background/Related Work

2.1 Related Model Designs

With respect to the model itself, this paper doesn't use any techniques outside of the standard deep learning algorithms - it uses linear, fully-connected layers with established activation functions for the network and an established criterion for training purposes. It does use a more recent word-vector representation for generating the word vectors called Glove though, which is described in [2].

2.2 Related Problem Models

The problem as a whole is very commonly attempted. There are many previous papers that attempt to model the helpfulness of reviews, however, I have been unable to find one that both uses deep learning as the model and attempts to model the exact helpfulness rating as opposed to descritizing the helpfulness rating. By using both of these techniques, this paper will hopefully push other group's models to be more expressive and more fine-grained in their modeling, which should improve the accuracy of the state-of-the art models for this problem.

3 Approach

3.1 Initial Recurrent Neural Net Model

My initial plan was to use a multi-layer recurrent neural network to model the reviews. I was able to get it working on my laptop, but after many days of effort and several all-nighters, I was unable to get torch [4] to work on Amazon's EC2 instances, which was needed because my laptop didn't have enough memory to handle anything except the smallest of RNNs. This forced me to revert to my secondary plan, which was to use the model I am about to detail.

3.2 Generating Representitive Vectors

We start by taking our data and tokenizing it with the Stanford Tokenizer located at [3]. We then convert the tokens to lower case and run the GloVe word vector generator [2] on our training corpous with the following parameters:

- memory: 4.0
- vocab_min_count: 5
- vector_size: 64
- max_iter: 30
- window_size: 15
- binary: 2
- num_threads: 8
- x_max = 10

The details of what each of these parameters do is detailed in the GloVe documentation at [2].

We used this configuration to generate vectors for the 999,998 most common words, along with an additional two for the start and end tokens, as well as for the unknown token, which comprised the remaining words other than the 999,998 words we generated vectors for. After this, we took the review text (and only the review text - we want the machine learning algorithm to learn the model with as little help from a human as possible), converted every word in the review into it's corresponding word vector, and then averaged all of the word vectors together to get a representitive vector for the review as a whole.

3.3 Results to Model

The original dataset gives us both the number of helpfulness ratings and the number of positive helpfulness ratings for each review. We use this information to calculate the percentage of ratings that said that the review was helpful, and we attempt to model this metric. There are, of course, many other ways to represent how helpful a review is with this data (eg. by taking into account the total number of ratings, or descrititizing the helpfulness, etc.), but this is how we chose to model the helpfulness.

3.4 Deep Learning Model

Since we now have our inputs and outputs, we need a model to do the translation. I decided to use a standard fully-connected neural-network architecture for the models. The network has some number of hidden layers, each of which is composed of a linear layer with the sigmoid function as the activation function. At the end of the network, we have a final linear layer attached to a sigmoid function, which is what we use to generate the predicted output.

3.5 Training

I used the standard torch training algorithm for training with a learning rate that varies among my experiments, Dropout at every hidden layer as the regularization, and MSE (mean-squared error) as the loss function for my model. I used this training algorithm in batches of training examples as the training methodology for my models.

4 Experiment

4.1 Dataset

We are using the amazon review dataset located at [1] for our data. This dataset is far too large to work with on the machines that I have available, so I cut it down to 1,000,000 examples for training purposes and 10,000 examples for testing purposes (Note that since I am reporting the results of all of my tests, I don't need a validation set since I am not optimizing the hyperparameters). After making sure all the examples had at least one helpfulness review, I used the training set to generate the word vectors and train the models on, and then evaluated the models on every element of the test set to get the accuracy of the models.

4.2 Experimental Methodology

My experiments took the above methodology and varied the number of hidden layers, the size of those hidden layers, and the learning rate of the model to get a good distribution of the hyper parameters. We used a batch size of 512 examples and 10,000 batches for each period of training, and then varied the above hyperparameters among the following values:

- Number of Hidden Layers: 1, 3, 5
- Hidden Layer Size: 32, 64, 128, 256, 512
- Learning Rate: 1e0, 1e-1, 1e-2, 1e-3, 1e-4

We experimented with every combination of the above values to give us a total of 75 different models. We evaluated each model on the average absolute difference between the predicted helpfulness rating and the actual helpfulness rating. The following subsections describe our results:

4.3 Comparing the Number of Hidden Layers

Figure 1 shows how the number of hidden layers affected the accuracy of the models. Here the results seem odd - since networks with more layers are more expressive, we would expect that they would also be more accurate - but this does not seem to be the case. This graph shows that the best performing network is the 1 hidden layer network by a significant amount, which indicates that the extra hidden layers actually hurt the accuracy instead of help it. The usual explaination for this is that the extra layers are over-fitting the data, but this doesn't make much sense either since that would imply that the best network for predicting helpfulness of a review (which is an extremely complex task) is a shallow network, which doesn't seem to fit. We will continue our analysis below to see if looking at the data in a different way can help us understand this better.

4.4 Comparing the Size of the Hidden Layers

Figure 2 shows us more odd results - the lowest error of any size of hidden layers is the lowest hidden layer size of 32. The problems with this are similar to those of the number of hidden layers from above: bigger hidden layers should be more expressive and thus more accurate, but our data shows that the most accurate model only has a hidden size of 32. This is counter-intuitive for the same reason as in the first section: evaluating helpfulness is a very complex task, and we would not expect 32 neurons to be the best for accomplishing this task. Let's compare the various learning rates to see if that illuminates anything for us.

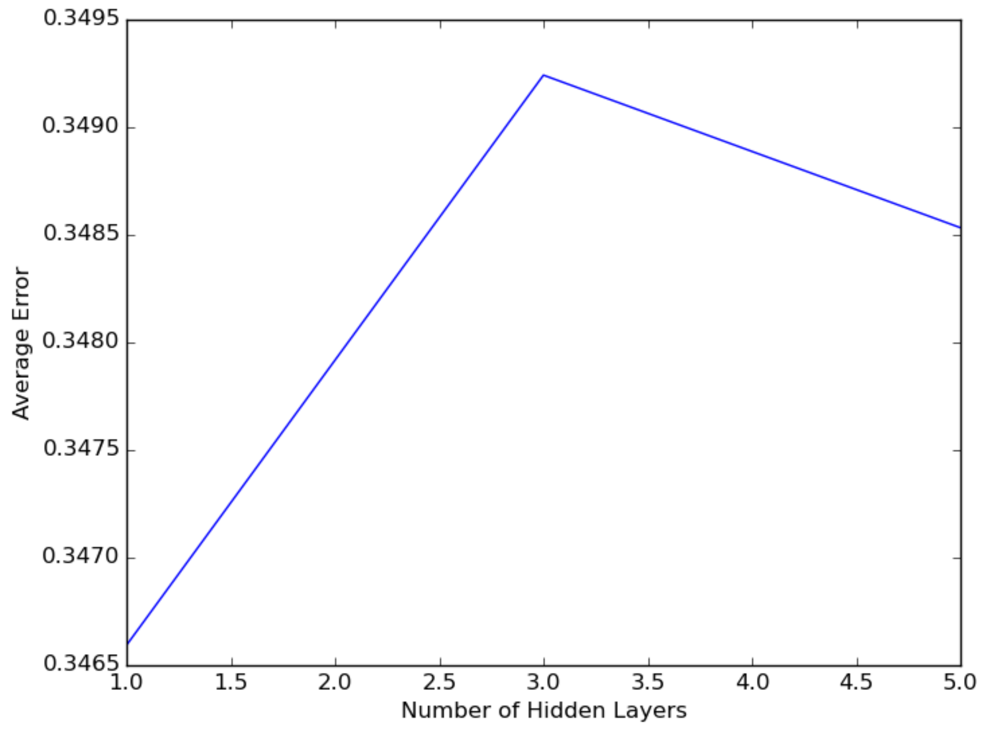

Figure 1: Number of Hidden Layers vs. Average Error

4.5 Comparing the Learning Rate

Figure 3 confuses the matter further - the results are odd this time not so much because of the absolute value of the optimum learning rate, but because of the shape of the curve. Specifically, it would be fine if the accuracy went down when lowering the learning rate from 1e0, because that would indicate that the lower learning rates were not high enough to get to the optimum result within the number of iterations we had. Similarly, if the accuracy went up when raising the learning rate from 1e-4, that would make sense as well since that would indicate that the higher learning rates were causing the model to jump around the optimum result which only a slower, more methodical learning rate would be able to reach. Having both in the same graph is the problem though - this indicates that the middle learning rates are somehow so large they are jumping around the optimum, and yet so small that they don't have time to reach it. A possible explaination is that the high and low learning rates found different optimum points in the space of possible parameters, but this still seems unsatisfactory. Next, we will analyze both the number and size of the hidden layers at once to see if seeing more trends that are dependent on the size of the network will help us analyze our results better.

4.6 Comparing the Size and Number of Hidden Layers

At first glance, Figure 4 looks like it won't help us much because the lines of each number of layers is very erratic and seems to have no pattern (the 3 layer case seems to be increasing, the five layer case seems to be decreasing, and the 1 layer case is simply alternating whether it increases or decreases), but this may be what helps us answer our questions. Specifically, the chaotic nature of the graphs may simply be caused by a high variance in the accuracy over the hyperparameters we are looking at. This could cause our confusing results above simply through random chance. However, we still don't know why our data has such a high amount of randomness in it - most datasets wouldn't have

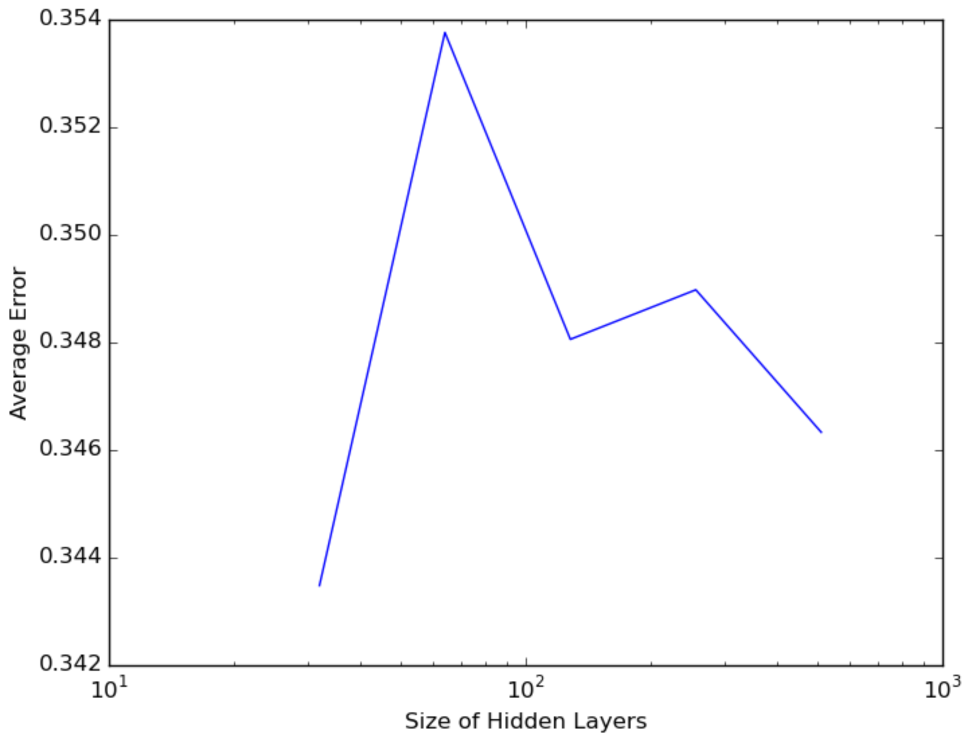

Figure 2: Size of Hidden Layers vs. Average Error

such a high randomness, especially when averaged over multiple learning rates like our data has been, which means that we have not answered all of our questions yet.

5 Conclusion

5.1 Our Results

In all of our analysis above, we focused on the relative performance of the hyperparameters to determine which was best, but we never looked at the absolute accuracy of our models, which is just as important as the relative performance of hyperparameters. Looking at the absolute accuracies, we see that the best accuracy we have achieved is about 30-35 percentage points, which is awful. This means that the average entry can easily vary in a range of 60-70 percentage points around the value we expect, which is nearly the whole range of values. This could easily be the cause of the random variation we noticed above - since our model is not very accurate for any of our hyperparameter combinations, this indicates that our model does not have enough information to make an accurate prediction, which means that it makes mostly random guesses, which will give us a high variance.

We now want to know why our model doesn't have enough information to make an accurate deduction. One possibility is the size of the word vectors - specifically the size is only 64 values long. This is relatively small compared to the size of the hidden layers we used, which means that increasing the size of the hidden layers isn't actually allowing us to store any more information, because there wasn't much information in the vectors to begin with. A far more likely reason though is the architecture of the model itself. Our model simply averages the word vectors together to get the vector for the review text. The problem with this architecture is that it makes us lose the information about the order of the words and a lot of information about which words are actually in the review since we average them all together. Since this information is usually important in determining whether a review is helpful or not, this causes us to lose a lot of information that is vital to understanding

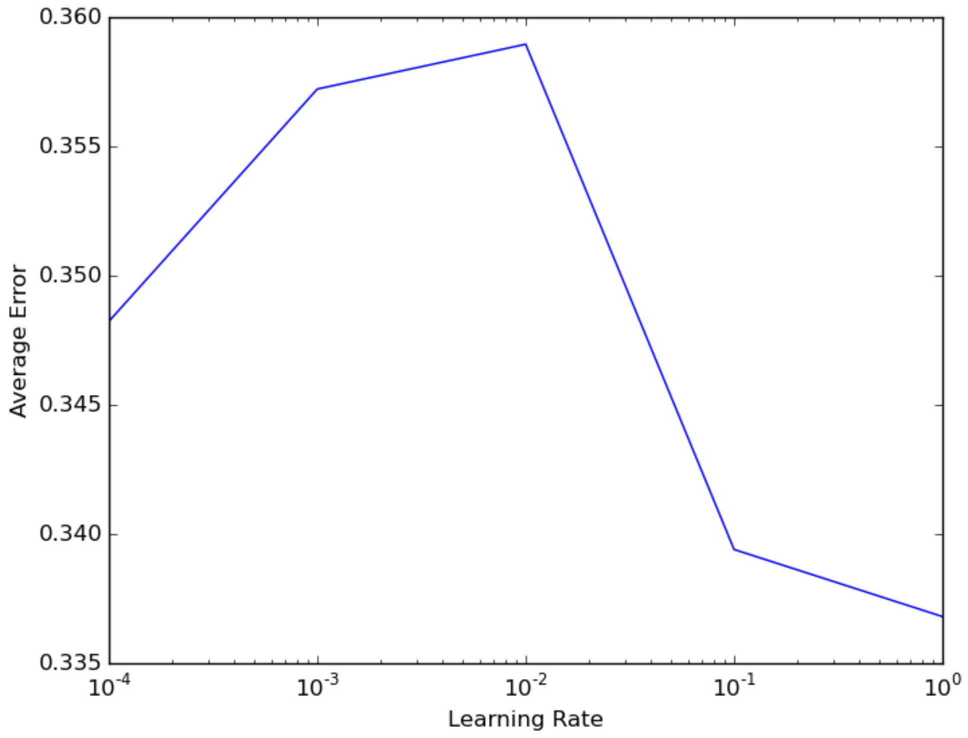

Figure 3: Learning Rate vs. Average Error

the review, which causes us to not give our model enough information, which causes the model to perform poorly. We will use our understanding of the failings of our model to decide what to do with our future work.

5.2 Future Work

Considering that what seems to be the main problem with our model is that it is losing too much of the information about the order and individual words in our review to be effective, the first step to improving the model would be to utilize modern NLP techniques, like recurrent neural nets, convolutional neural nets, or even the dynamic neural net that Richard talked about in class. These models are all specifically designed to maintain the information about the order of the words and which exact words are in the sentence to begin with, and they have already been tested exensively and have been shown to work well on similar tasks, which seems to indicate that they would performe well on this task as well. I suspected that this would be the case when I started the project, which is why I initially attempted to implement these models, but due to the difficulties in getting Torch to work properly on Amazon's EC2 instances, I was unable to get the RNN working and was forced to implement my backup plan, which was the model outlined in this paper.

After we use an RNN, CNN, DNN, or any other more complex model we choose though, we will still want to improve the accuracy further. One way to do this is to take into account the number of ratings per review. The current model only considers the fraction of positive ratings, but doesn't consider the fact that most reviews only have a few ratings (we know this because this was one of the problems we set out to solve). This means that even if our model gets the helpfulness rating exactly right, we might not have the resolution in the percentage of positive ratings to be able to know that. For example, if a review's rating should be around 60%, but only has one positive rating, even if our model gets the 60% helpfulness exactly right, it will look like our model is off by 40 percentage points, which doesn't look good at all. One way to account for this is to round the output

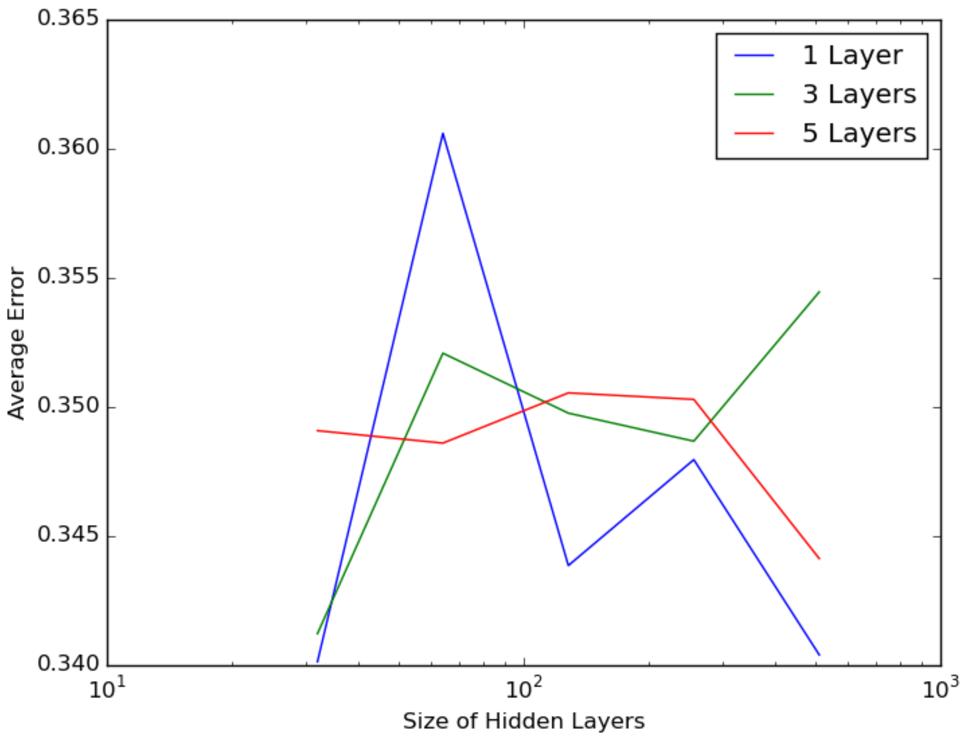

Figure 4: Size of Hidden Layers vs. Average Error

of the model to the nearest fraction that the number of raters could represent (that is, if there are 2 reviewers, round it to 0, 0.5, or 1, whichever is closest), and then apply MSE to the result. Another way is to have the neural net simply model both the number of positive raters and the total number of raters and calculate the percentage based on that. Either of these methods would allow us to better account for the number of raters for any given review.

References

[1] J. McAuley, C. Targett, J. Shi, A. van den Hengel SIGIR, 2015, http://jmcauley.ucsd.edu/data/amazon/links.html

[2] J. Pennington, R. Socher, C. Manning. GloVe: Global Vectors for Word Representation. http://nlp.stanford.edu/projects/glove/glove.pdf

[3] http://nlp.stanford.edu/software/tokenizer.shtml

[4] https://github.com/torch/torch7/wiki/Cheatsheet