Deployment and Hyper-Parameter Optimization of Chatbots

Michael Read and Pablo Rivas
Department of Computer Science, School of Computer Science and Mathematics
Marist College, 3399 North Road Poughkeepsie, New York 12601, United States

Abstract— *Chatbots are a specific application of a set of machine learning algorithms belonging to the family of natural language processing (NLP). Recently, NLP algorithms have gained attention as we are closer to passing the Turing test when they are applied to human-computer interaction-based systems. In this thesis project we will model chatbots using NLP-based machine learning algorithms based on datasets of people. Based on sentences and text from a specific person, we measure how well the chatbot models such person's writing. In theory, NLP algorithms of the Long Short Term Memory (LSTM) type are capable of remembering, summarizing, and learning patterns of speech, style, and forms of any sequences of text. Results indicate that an LSTMs is capable of generating novel sentences using as a case study Donald Trump's tweets.*

Keywords: Sequence-to-Vector, Neural Networks, LSTM, Chatbots, Hyper-Parameter Optimization

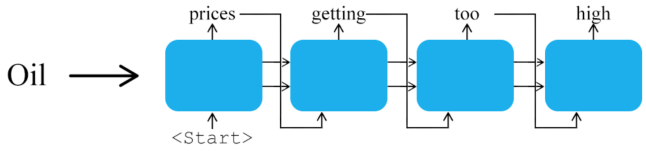

Fig. 1: Sequential representation of LSTM recurrences.

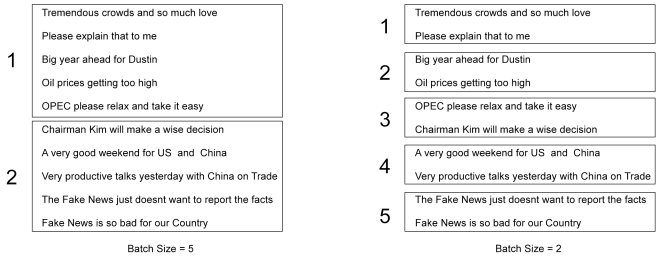

Fig. 2: Example of batch selection w. batch sizes of 2 & 5.

1. Introduction and Background

LSTMs are a subset of Recursive Neural Networks (RNNs) [1], and their recurrence can be visually explained as sequential items until the model is made to stop, as depicted in Figure 1.

LSTMs are capable of learning long-term dependencies, as well as handling the vanishing gradient problem that plagues non-LSTM RNNs [2]. The vanishing gradient problem refers to an issue in neural networks that utilize gradient-based methods of training, in addition to backwards propagation. As the number of layers increase, the update to the weights of the front layers gets compounded more and more, sometimes to the point where training is no longer possible for the front layers. LSTMs solve this problem by allowing the gradient, instead of being compounded from layer to layer, to flow unchanged through the LSTM.

In this paper we, thus, attempted to modeled a chatbot using an LSTM, which has proven to be a challenging task in machine learning [3]. In our experiments, we evaluated the performance effects of changing three parameters: batch size, model size, and dropout. The first parameter, batch size, refers to the number of lines of the input the training algorithm looks through at any given time, illustrated in Figure 2. The best batch size would be the entire length of the training input, however due to hardware limitations it is incredibly unlikely to get to that batch size outside of specially built machines dedicated to AI training. The smaller the batch size, the easier the process is on the hardware, but the more time it will take per pass, and the worse the results will be. However, overstressing the hardware with batch sizes that are too large will also result in poor results, so it is important to find the batch size that best suits the hardware that training is being performed on.

The second parameter we have tested is model size, which refers to the length of the word vector encodings. When performing NLP, it is common to convert the words from the input into vectors, to help the algorithm better understand the relationship between words [4], described by Figure 3. When two vectors end up close to one another, this represents some sort of relationship between those words, that there is some similarity between the words. The longer these vectors are, the more information there is that can be encoded into these vectors, allowing for more, better relationships between words. The trade-off between model sizes is that with larger model sizes, more information can be encoded yielding better results, while taking more time to train these word vector encodings.

The third parameter we have tested thus far is dropout. Dropout refers to the random dropping of a number of nodes in the network, signified by a percentage, depicted in Figure 4. We utilize dropout in neural networks to mimic the way our own brains work, as we constantly have brain cells dying and being created. This causes us to forget the less important pieces of information, while still remembering the important

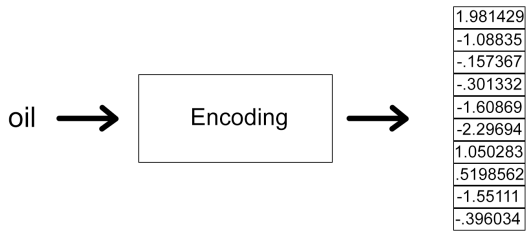

Fig. 3: Example of word encoding into a vector of size 10

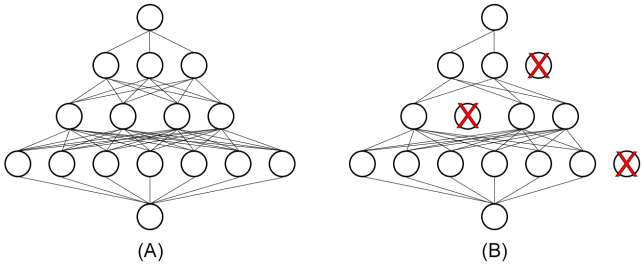

Fig. 4: Example of dropout applied to a neural network. (A) shows the network before dropout, (B) shows the network after dropout.

information. The theory of dropout is that, by randomly dropping a certain percentage of nodes, we will be likely to lose the less important patterns that may only come up a few times in the inputs, while being less likely to lose the big patterns that consistently show up in the input [5].

2. Methods

2.1 Dataset

Our first step in getting our dataset ready for our chatbot was to acquire the data. We did this through the website trumptwitterarchive.com, which contains all of the tweets from Donald Trumps twitter account from 2016 and earlier that have not been deleted, as well as almost all of his tweets after 2016.

The next step was to perform the preprocessing, to make sure that we didn't have any characters that might mess with the chatbot, while keeping emojis, and making sure that the data we got was in the format we wanted it in. We did this by first running the data we received through a python script which replaced symbols such as '%' and '&' with the words 'percent' and 'and', and removing quotes and other similar special characters, as well as hyperlinks. While this python script worked for most of the unwanted symbols, some symbols were not able to be removed by the python script so we removed the rest by hand.

After dealing with the special characters, we then broke the tweets down into sentences, as the goal of this project was to generate sentences and not tweets. To break down the tweets we ran the preprocessed tweets through another python script which added new lines after each period,

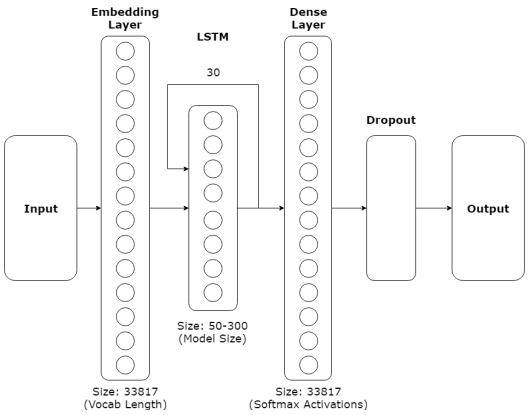

Fig. 5: Depiction of our designed architecture. The dense layer is a Softmax layer.

exclamation mark, and question mark. While this worked for the most part, we did have a few issues in cases like abbreviations where periods do not mark the end of a sentence. These issues were difficult to attempt to handle, however we tried to at least handle common cases such as "Donald J. Trump" and "U.S.A.".

Finally, we performed basic sentence analysis on the formatted dataset to determine what the average length of Trumps sentences is, which we used as the length of sentences our chatbot would form.

2.2 Architecture

After getting our dataset prepared and performing our analysis, we began working on the chatbot itself. The first step here was to construct and train our own word vector encoding based on our dataset. While it is possible to find already pretrained word vector encodings, we decided that it would be best for our purposes to create our own. This allows our encoding to best fit our dataset, as most pretrained encodings that can be found will not include the hashtags, twitter handles, and emojis that our dataset contains. For our specific case of Donald Trump, this also allowed us to include his common misspellings and nicknames to get an even better fit for our chatbot.

The next step was to construct our neural network and begin testing. We decided to go with a 32-layer, LSTM-based neural network. We chose the Adam optimization algorithm for our training, with a softmax-based sparse categorical cross entropy loss function. We chose the Adam optimization algorithm because it tends to yield extremely fast results; however, it can sometimes get stuck in a loop for a while, which slows down that particular run. While this is not common, it is something that should be taken into account when thinking about how you want to design your architecture. A depiction of our architecture is shown in Fig. 5.

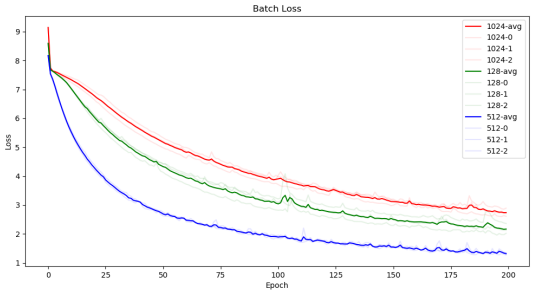

Fig. 6: Batch Size test results mapping loss over time

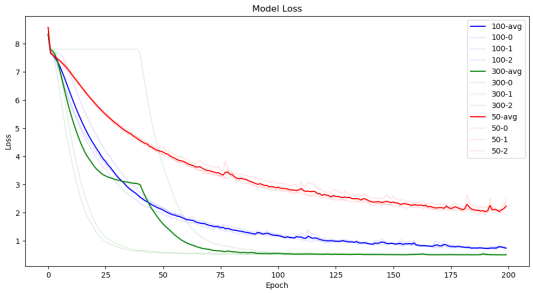

Fig. 7: Model Size test results mapping loss over time

We performed three tests for each parameter value, beginning with batch size testing. Our starting parameters were a batch size of 128, a model size of 50, and a dropout value of 0%. We then changed our batch size to 512 and 1024. We then chose the best batch size, and moved on to modifying our model size. For model size testing, we used values of 50, 100, and 300. Choosing the best model size, we moved on to dropout testing. We tested dropout values of 0%, 10%, 15%, and 20%.

3. Results and Conclusions

3.1 Batch Size

The first experiment was carried on batch size. Fig. 6 shows the Batch Size test results mapping loss over time. As it can be seen, the batch size that performed the best was 512, and the batch size that performed the worst was 1024. The batch size of 128 fell between the two. What we would expect to see is an increase in batch size correlating with better performance, however we found that the highest batch size performed the worst. We believe that this is due to hardware limitations, where our hardware was not able to optimally use the larger batch size. Other than that, the outcome is what we expected.

3.2 Model Size

The second experiment was with model size. Fig. 7 shows the Model Size test results mapping loss over time. For these tests, we continued with our batch size of 512. In our testing, we found that a model size of 300 performed the best, with the worst performance coming from the model size of 50. The model size of 100 performed slightly worse than the model size of 300, but better than a model size of 50. These results line up with what we would expect from these tests. The corner in the graph on the model size 300 line around epoch 50 comes from one of the tests having the optimizer get stuck in a loop, which happens to the Adam optimizer from time to time, as evidenced by the flat line between epochs 0 and 50 on one of the three tests. There is not a huge difference between the model sizes of 100 and 300 compared to 100 and 50, so the decision between the two of them is a question of the increase in performance over the added time to train the word to vector encoding. It is important to note, however, that while the model sizes of 100 and 300 seemed to converge to a similar loss rate, the model size of 300 was able to converge much faster than the model size of 100.

3.3 Dropout Rate

The third experiment was on the proportion of dropout. Fig. 8 shows the Dropout test results mapping loss over time. For these tests, we continued with our batch size of 512 and model size of 300. In our testing, we found that a 0% dropout produced the best results, with each increase in dropout percentage producing worse results, meaning that the worst results came from a 20% dropout. While we were not sure what percentage would work best, we did not predict that the 0% dropout would be the best. We believe that this is due to either the size of the network not being large enough to benefit from the dropout, or the size of the dataset not being large enough to benefit from dropout. To better optimize dropout, it may be best to increase the size of the network, and add in more layers, to see if network size is the parameter holding back dropout. The corners in this graph are also produced by certain tests having their optimizer not converging. This experiment elucidates the consequences of selectively ignoring (dropping) neurons associated with specific vocabulary words; one might observe that, while the rest (surviving) neurons become stronger, still the performance of the network does not increase. This is especially interesting if we consider that many words in the vocabulary consist of hashtags and user names that may not contribute much to the structure of sentences from a grammatical standpoint.

3.4 Quality of Results

After selecting the optimal set of hyper-parameters, we collected sample sentences of the model at different stages of training. Table 1 shows some examples of these collected sentences. As can be seen, the early results seem to be a random string of words from the vocab with no meaning. The late results, on the other hand, seem to make much more sense. While they are not perfect results, they are much better

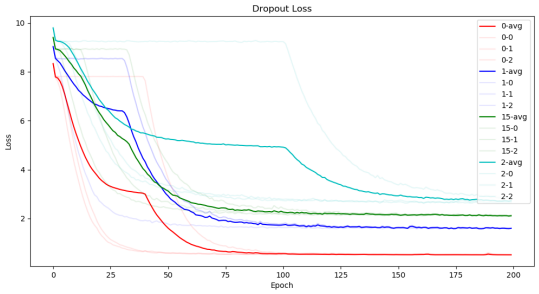

Fig. 8: Dropout test results mapping loss over time

Table 1: Performance comparison of LSTM during training

Ep	Prompt	Output
5	bad	`bad hp heroic contractor @dhsgov #oscars @souperfan2013 @kevweezy5 worst-the @yewkalaylee lifetime`
5	democrats	`democrats syria @buckleybro40 michael @greenerag @samanthaviner @mstrbass2000 @pamplinfilmco @business @thecccowanshow prison`
100	crooked	`crooked hillary presidential beyond belief approximately hampshire packed house #potus7 #usa`
100	fake	`fake source gov agree me supporters @holzmdk radar better off @garrett`

than the early results. An issue that is brought forward by these results is that the vocab contains an incredibly large number of twitter handles, which causes them to show up a lot even in the later sentences.

3.5 Discussion

While these results are not bad, there are a few things that we believe may be having a negative impact on our model. First, the vocab that we have is incredibly large. This is somewhat due to a myriad of misspellings and typos in the dataset, and also somewhat due to a large number of twitter handles, some of which only appear a few times at most in the corpus. Another issue that may be impacting the model is the size of the model. It may be that to get the best results, more layers are needed in our model. The problems also may just be coming from the writing style of the dataset. It may be that the tweets are too inconsistent to be able to pull any meaningful data about how they are constructed.

However, we believe that these results are a good start. While the sentences generated could be better, there is a clear improvement between the early output and the late output which is overall a success. Also, each parameter that we optimized, provided an increase in performance, which shows that we are moving in the right direction. Finally, the fact that we were able to produce these results on a machine that was not specifically designed to perform heavy ML training shows how far technology has come in the last few decades, and so theoretically even better results could be produced on a machine built with ML in mind.

4. Conclusions

We conclude that it is possible to develop and deploy a chatbot, when taking special care in the choice of dataset, preprocessing, and architecture. We chose a dataset that was both large and freely available, processed it to make sure that there were no characters that would impede the performance of the chatbot, and then chose the paramters of the chatbot that would yield the best results we could achieve. We determined that, for our setup, a batch size of 512, combined with a model size of 300 and dropout rate of 0%, gave us the best results. While our vocabulary was too large to easily produce coherent sentences, it would be a trivial task to remove uncommon handles and other unwanted words from the vocab to help streamline the model. Finally, while our results were not bad, we believe that this process could be repeated on a dedicated ML machine to produce even better results. The next steps for this project are to look into different parameters to try and better optimize our chatbot. The parameters that we are currently looking into are network size, and optimizer. For network size, we want to slowly add layers to see how that affects the performance of the chatbot, as well as whether or not a larger network changes the dropout curves. For the optimizer, we are looking into whether changing the loss function from the softmax-based sparse categorical cross-entropy function that we are currently using to a sigmoid-based binary cross-entropy function will yield better results. If you would like more information about this project, we have made the code available in a github repository which can be found at:

`github.com`
`/MichaelInAction/Senior_Thesis_LSTM_Chatbot`

References

[1] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.

[2] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[3] M.-H. Su, C.-H. Wu, K.-Y. Huang, Q.-B. Hong, and H.-M. Wang, "A chatbot using lstm-based multi-layer embedding for elderly care," in *2017 International Conference on Orange Technologies (ICOT)*. IEEE, 2017, pp. 70–74.

[4] Q. Zhou, N. Yang, F. Wei, and M. Zhou, "Selective encoding for abstractive sentence summarization," *arXiv preprint arXiv:1704.07073*, 2017.

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.