Reddit WSB Language Model

ELEC-E5550 Statistical Natural Language Processing
Project Report

Nhan Phan (913346), Ryoko Noda (875031)
May 7, 2021

1 Introduction

Our objective in this project is to create a language model that replicates Reddit posts, posts from WallStreetBets in particular. WallStreetBets (WSB) is a discussion forum on Reddit, a social media platform. Members of WSB are famous for crushing a 8 billion dollar hedge fund with their investment strategies[1]. Those members, however, have a very different way of talking that is very different from a professional investment analyst. They call themself “retards”, and talk with slang, emojis and memes.

We chose to do this work because a language model is a fundamental part of any advanced natural language processing (NLP) task. It is used in translation models, speech recognition, and description tasks to name a few. And what kind of language these language models are wired to understand will have a direct impact on what kind of language it can generate or interpret.

Usually language models are trained using datasets from whatever environment and purpose it is expected to be used in. For instance, language models used for conversation agents should be trained on human conversation and not Shakespeare (or else whatever the language model is used in would start talking like and expecting to hear Shakespeare). So in this project’s case, if we wanted to make a language model that sounds like Reddit posts, we must train the model on posts from Reddit.

That is exactly what we did. Our work involves collecting discussions from the forum and building a corpus. Then from the corpus we tried creating four types of language models: one the traditional n-gram model, and the other three using deep learning. Then we compare the models on their ability to produce WSB posts.

2 Method

The project had three steps, 1. collecting text from Reddit, 2. creating language models from it, and 3. comparing the models. For the data collection we used Pushshift API. For the language models we used n-grams and 3 recurrent neural networks (RNN): Gated Recurrent Unit (GRU) and two types of Long Short-Term Memory model (LSTM) (normal and bi-LSTM) to create four different models. Then we used perplexity and visual inspection to evaluate the models.

2.1 Pushshift API

For the text collection, we used the Pushshift Reddit Dataset. This is a dataset that is collected by Jason Baumgartner et al. It is an ever-growing collection of posts from Reddit that can be accessed through a free API [2].

According to Baumgartner’s article[3], the Pushshift’s architecture is composed of four parts: The "ingest engine" that scrapes content from webpages, the " PostgreSQL database" that allows queries, "Elastic Search document store" to aggregate the collected posts, and the "API" that allows users to sort through and extract from the monthly data dumps. We show an image of the architecture from the original Pushshift dataset article in Figure 1.

2.2 N-Grams with absolute smoothing

As covered in the Statistical Natural Language Processing (SNLP) lectures, n-grams are a way of predicting a word that comes after a history of words using conditional probability. For example, if we are trying to predict using trigrams, we need to create a model that can give us a high probability

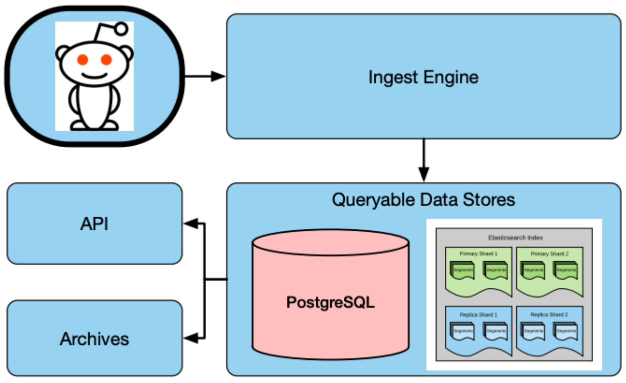


Figure 1: Pushshift platform architecture from [3]

for $P(\text{word1}|\langle s\rangle\ \langle s\rangle)$, $P(\text{word2}|\ \langle s\rangle\text{word1})$, all the way down to $P(\langle /s\rangle|\text{word}n\ .)$, where word$n$ is the last word in the sentence and $\langle s\rangle$ and $\langle /s\rangle$ indicate sentence beginning and end.

N-grams use maximum likelihood estimates as a way of approximating the conditional probabilities, which is simply the count of a particular n-gram divided by the count if its history. So for example, the estimate for $P(\text{word2}|\ \langle s\rangle\text{word1})$would be:

$$P(\text{word2}|\ \langle s\rangle\text{word1}) \tag{1}$$
$$= \frac{Count(\langle s\rangle\ \text{word1word2})}{Count(\langle s\rangle\ \text{word1})} \tag{2}$$

The probability of an entire word sequence can be estimated by the chain rule of probability. Below we use the expression $w_{1:n}$ for expressing a word sequence from word 1 to word n and show the sequence probability for trigrams:

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_k|w_{k-2:k-1})$$

N-grams are a simple but powerful tool, and our course textbook demonstrates that it can successfully generate Shakespeare-like texts[4]. However, it does have a problem when implemented as pure conditional probabilities over the training corpus: it cannot predict new words it has never seen, or known words in different contexts.

The former problem is solved by introducing a psuedo-word, <unk>, to our vocabulary. We used the method introduced in the course assignment to do this, by keeping the N most common words as they are, and convert the rest to <unk>s.

The latter problem is solved by smoothing, which is a way of allocating a small amount of probability mass to n-grams that are possible (can be created from the vocabulary) but have not happened yet. This prevents the model from dismissing an unfamiliar context as "impossible". We used the same smoothing method as the SNLP assignment, which is absolute smoothing:

$$P_{ABS}(w_i|w_{i-n+1:i-1}) = \frac{\max(C(w_{i-n+1:i}) - \delta, 0)}{C(w_{i-n+1:i-1})} + \lambda(w_{i-n+1:i-1})P(w_i|w_{i-n+1:i-1}).$$

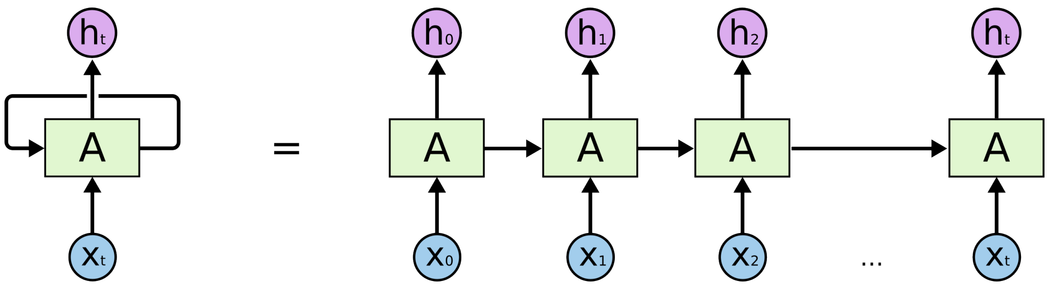


Figure 2: A generic image of what a RNN network looks like from [5]

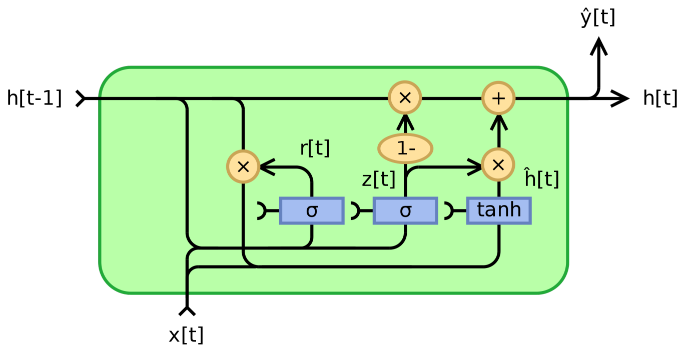


Figure 3: An image of a GRU from Wikipedia [6]

2.3 GRU language model

N-grams have another obvious weakness. It cannot use words outside of its scope of context to predict words (trigrams cannot look five words back). A recurrent neural network language model addresses this problem by using a hidden "state" that passes information of what the model has processed in previous iterations. This can be interpreted as a memory of what kind of words the model has seen in the past in a language model's case.

A Gated Recurrent Unit (GRU) is a core component of RNN that accepts and produces one hidden state. RNNs using GRU can predict from longer context than n-grams, but its memory is relatively short term compared to the LSTM model. The basic structure of the GRU is shown in 3.

The computations of a GRU proceeds as follows, where x[t] is the input at time t, h[t-1] is the hidden state at time t-1, h[t] is the hidden state at time t, and r[t], z[t], and $\hat{h}$[t] are the reset, update, and new gates respectively. W_{i*} indicates the weight matrix for the input vector and W_{h*} indicates the weight matrix for the hidden vector.

$$
\begin{aligned}
&\text{r[t]} = \text{sigmoid}(W_{ir}\text{x[t]} + b_{ir} + W_{hr}\text{h[t-1]} + b_{hr}) \\
&\text{z[t]} = \text{sigmoid}(W_{iz}\text{x[t]} + b_{iz} + W_{hz}\text{h[t-1]} + b_{hz}) \\
&\hat{h}\text{[t]} = \tanh(W_{i\hat{h}}\text{x[t]} + b_{i\hat{h}} + \text{r[t]} * (W_{h\hat{h}}\text{h[t-1]} + b_{h\hat{h}})) \\
&\text{h[t]} = (1 - \text{z[t]}) * \hat{h}\text{[t]} + \text{z[t]} * \text{h[t-1]}
\end{aligned}
$$

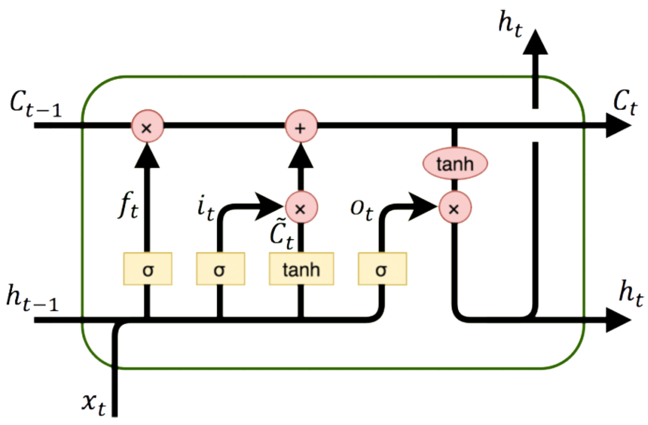


Figure 4: A image of a LSTM unit from an APMonitor online course [7]

2.4 LSTM language model

The Long Short Term Memory network (LSTM) is another variation of RNN, with two hidden states instead of one. One of these states represents a "long term memory" and holds context from many iterations back. The other represents "short term memory" and holds context from the last several iterations. In theory, LSTM models can do predictions with a significantly larger history compared to GRU models. The LSTM unit is shown in 4.

The computations of a LSTM unit is shown below. h_t, c_t and h_{t-1}, c_{t-1} are the hidden states and cell states at time t and t-1 respectively, x_t is the input, and i_t, f_t, $\tilde{C}_t$, o_t are the input gate, forget gate, cell gates, and output gates respectively. W_{i*} indicates the weight matrix for the input vector and W_{h*} indicates the weight matrix for the hidden vector.

$$\begin{aligned}
i_t &= \text{sigmoid}(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \text{sigmoid}(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
\tilde{C}_t &= \tanh(W_{i\tilde{c}}x_t + b_{i\tilde{c}} + W_{h\tilde{c}}h_{t-1} + b_{h\tilde{c}}) \\
o_t &= \text{sigmoid}(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t * c_{t-1} + i_t * \tilde{C}_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}$$

There is another variant of LSTM called bi-directional LSTM (bi-LSTM). The unit and the computation remains the same, but bi-LSTM has one additional layer of the LSTM unit that processes information backwards, starting from the last set of words to the first set of words. This has proven to be an effective for sequence classification problems.

2.5 Perplexity

Perplexity is an evaluation metric for language models, and in the n-gram's case is defined as below, where W is a sequence of words $w_1, w_2, \cdots, w_n$.

$$\text{Perplexity}(W) = \sqrt{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 w_2 \cdot w_n)}}$$
$$= \sqrt{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-n+1:i-1})}}$$

In a neural network model's case, the perplexity is calculated as, below, where H indicates the average cross-entropy per word [8] [9]. The perplexity in our textbook [4] uses 2 as the base, but e seems to be used for practical purposes so we use the version with e.

$$\text{Perplexity}(W) = e^H$$

Perplexity has a low value when the conditional probabilities of words are high, which means the model with the lowest perplexity would be considered the best.

3 Experiment

3.1 Download data from Reddit's WallStreetBets

Reddit provide users with an API to download their data. Initially, we used PRAW, the Python wrapper, to use the API with Python [10]. However, official API from Reddit has several flaws, mostly due to Reddit's restriction. Reddit only allow maximum of 1,000 comments per request, which does not allow us to obtain a reasonable amount of data for our project.

We first tried to filter the comments by time, so we can download 1,000 comments per day, but we found that Reddit no longer support that function. Another option we can do is create a software that automatically download comments every 30 minutes. By our estimate, letting it run for one month could at least give us 1 millions comments. This plan, however, was abandoned due to the low quality of the data we got. The comments downloaded by this method include a lot of spams. Normally, spam comments will be deleted by moderators or will be downvoted by Reddit users. Because Reddit API download all comments in recent order, most of comments downloaded were new and were not marked as spam or did not get a proper score. Furthermore, Reddit API does not allow to filter by time, and we could not get data from the period before the surge of members in WallStreetBets, this results in almost all of the comments we get are from new members and does not correctly represent the original WallStreetBets (see figure 5 on page 6).

Eventually, we found another solution. Jason Baumagartner, a Reddit user, has been downloading data from Reddit and provide Pushshift API to download those data for free [3]. His data contain all comments, submission from all subreddit, including WallStreetBets. Pushshift API offer a wide range of option for us. We can select the period we want for our data, the number of score a comment or a submission have, and allow sorting those data. All filter options can be applied before download, and there's not limit of the number of data request, therefore, even though Pushshift's server is not reliable as Reddit's, we can finally download our data we want. Pushshift use RESTful API so we use PMAW, a light wrapper for Pushshift API so we can download our data with Python [2].

To maintain a high quality data source for our text generation model, we carefully selected what data to download from Pushshift. Each submission must has a minimum score of 40, and each comment must has a minimum score of 10. While the minimum score is low, this is a reasonable score for data from Pushshift. Because Pushshift download data very quickly after it was posted in Reddit, the score stored in Pushshift is not the final score. For example, the submission from http://redd.it/bdsx91 have a real score of 228, but in Pushshift, its score is only 210. We also filtered any submissions or comments that were deleted by moderators.

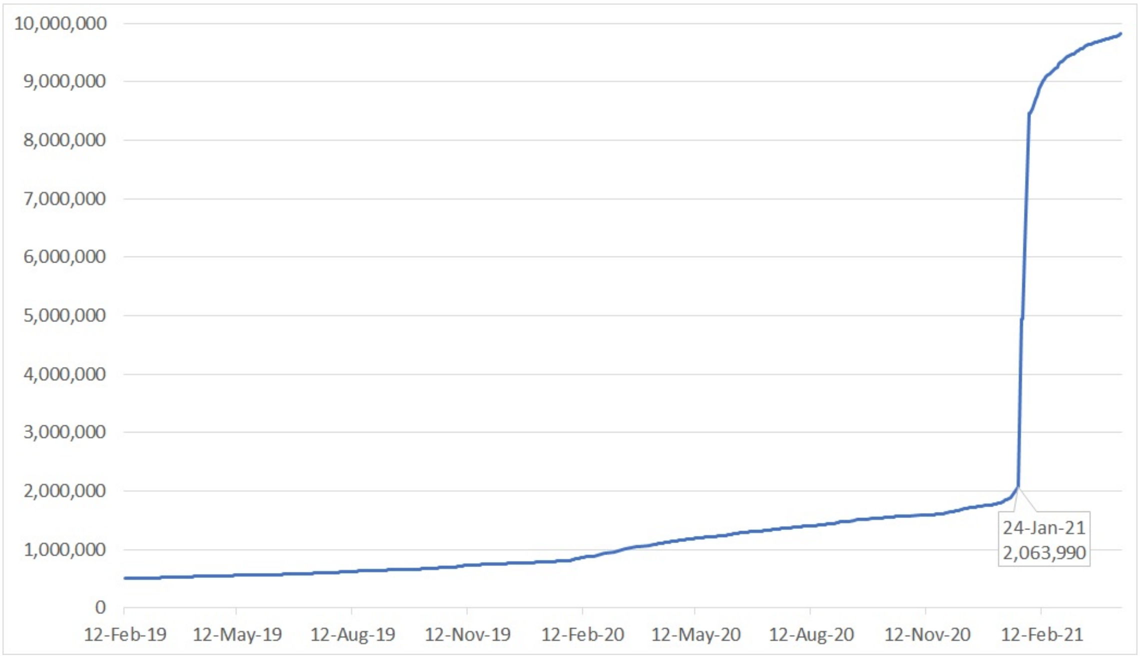


Figure 5: Number of members of WallStreetBets from 12 February 2019 to 16 April 2021

The last point we have to consider when download data from WallStreetBets is the GameStop short squeeze event in January 2021. This event make WallStreetBets become famous and attract 6 millions members to the subreddit, quadruped its number of members [11]. New member's submissions and comments are not unique and will reduce the quality of our data, but there's no option to filter out those new members. To avoid those new members, based on figure 5, we select data from 12 January 2021, before the surge of new members, back to 12 February 2019.

3.2 Data preprocessing

We first consider checking spelling errors in our data set, since typo is a common errors in forum discussions. However, our goal is to generate a sentence with unique style that only from WallStreetBets, and their style intentionally have typos. For example, "stonks" in "stonks only go up" meme is a real word in WallStreetBets. It becomes difficult to determine whether a typos is intentional or not, and we need to use our human judgment on case-by-case basic. Therefore, we skip the typo check in the preprocessing part. In addition, number also have special meaning to members in WallStreetBets. Members, for example, often set their stock price target at $420. For that reason, we also did not remove any numbers in our dataset.

Most of our preprocessing are for cleaning website link in discussions, which could be a Graphics Interchange Format (GIF) meme, link to a news article outside Reddit or a link from another discussion in Reddit. The reason is we did not have the resource to processing those link into a meaningful text for our data.

WallStreetBets subreddit use a lot of emoji, especially the rocket emoji and moon emoji, which indicate the stock price will go up much higher (as in rocket fly to the moon). We try to preserve the emoji in our data, so that our prediction can predict emoji in output. As a result, our data intentionally contain special characters. For example, the rocket emoji was stored as "ðŸš€" in our data file.

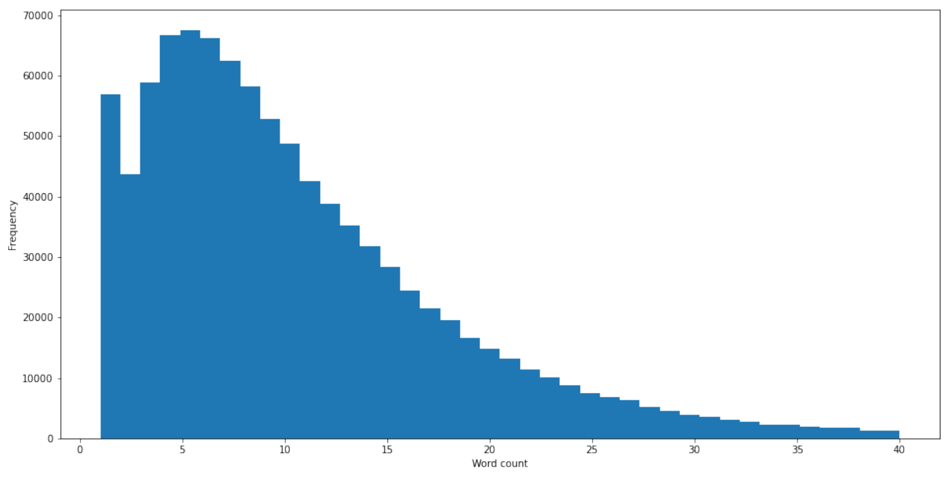


Figure 6: Number of words in a sentence from WallStreetBets' data

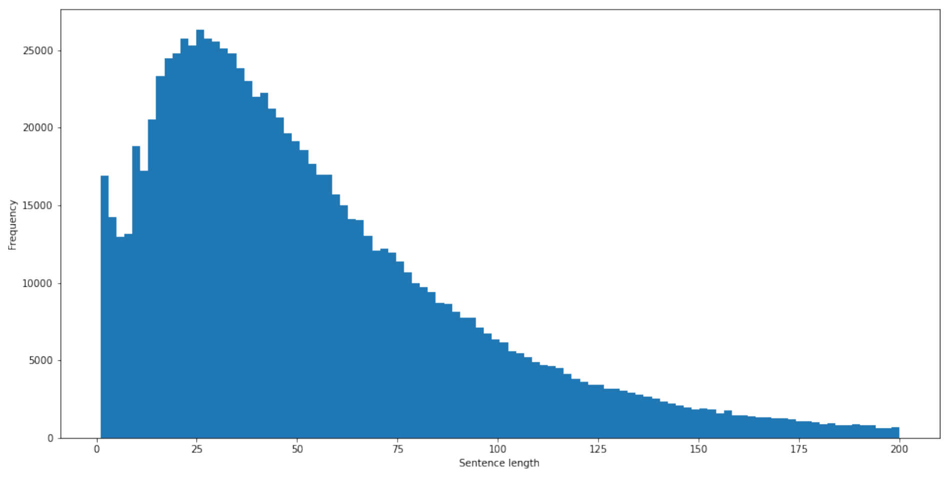


Figure 7: Sentence length from WallStreetBets' data

Because WallStreetBets is a discussion forum without any restriction on the length of users' comments, many discussion only have a few words, some even only have one character (e.g "k") or just an emoji. As indicated by figure 7, most of the sentences in our data is short with length less than 50 characters. To make sure our model generate more meaningful sentences, we decided to remove any sentence that has length less than 15 characters.

Our raw sample contains 967,968 sentences, and our final cleaned data contains 851,344 sentences.

3.3 N-grams model

We created two n-gram models with different vocabulary sizes, which were set by keeping the N most frequent words and replacing the rest with the unknown word token <unk>. One model has $N = 5,000$, which was the maximum vocabulary size we could set with our resources. The other model has half the vocabulary size at $N = 2,500$. Both used the absolute smoothing method with the maximum order n-gram set as 5-grams.

Also, to calculate perplexity, we first set aside 1% of the corpus as a test set. We settled with this split because anything larger could not give us a perplexity value in reasonable time, and because 1% of the original dataset still contained 8,513 sentences.

3.4 RNN model

At first, we try to use all of our dataset for RNN model. However, we quickly found that we did not have the computational hardware for such heavy task. Instead, we just use a sample of our dataset, which contain 40,000 sentences drew randomly from our dataset. While sample dataset is roughly 4.7% of our dataset, it took at least 5 hours to train a epoch on computer with 16GB RAM CPU, and about 20 minutes to train a epoch on computer with 16GB GPU. Therefore, all our RNN model only use sample data for training, optimization and prediction.

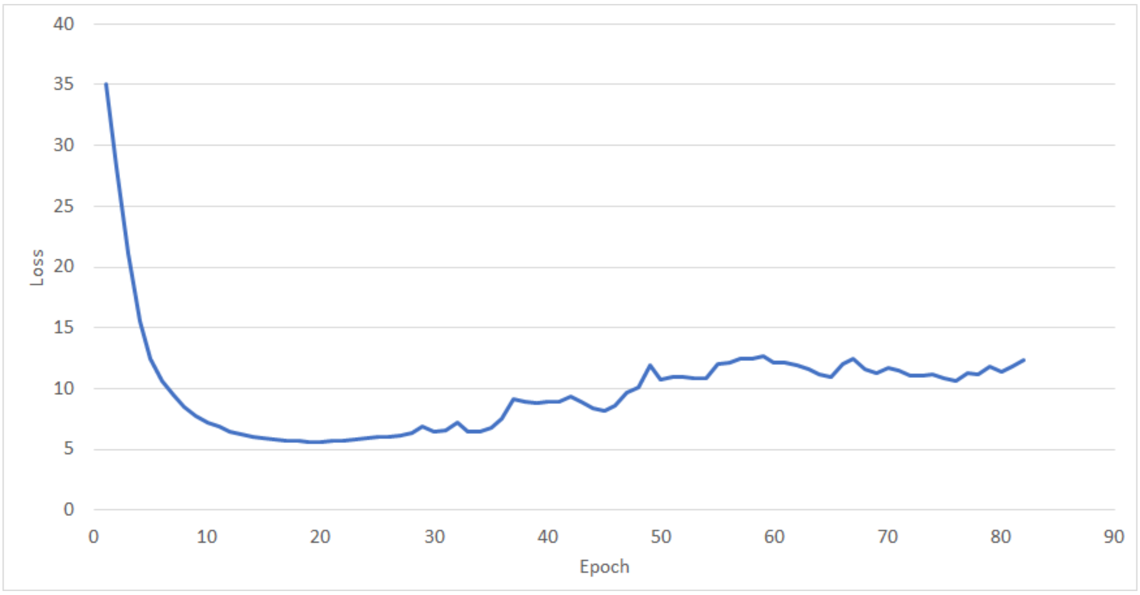


Figure 8: Loss with learning rate 0.001 and 20,000 sentences without preprocessing

We first use learning rate of 0.001 and sample size of 20,000 raw sentences without any detail data preprocessing. However, from figure 8, we found that 0.001 learning rate is large and it lead to the model move away from the local minimal after 20 epochs. We optimized the model by used a larger sample size of 40,000 sentences, a smaller loss of 0.0005 and perform data preprocessing to get a higher quality sample data.

4 Results

For n-gram models, the size of vocabulary is important. With vocabulary size of 2,500 words, the model is relatively quick to predict a new sentence, but the predicted sentences had as many as three unknown word tokens in a sentence. By increasing our vocabulary size to 5,000 words we got more meaningful predicted sentences, but each prediction took about 30 minutes. Also, even with a 5,000 word vocabulary, many of the sentences still contained unknown words. Some of the sentences generated from the 5,000-word vocabulary n-gram model are:

Seed: "stonk": *Stonk go - $ 2k immediately at open?*

Seed: "stock market" : *Stock market is still free enough to realize that gold is nowhere near <unk>*.

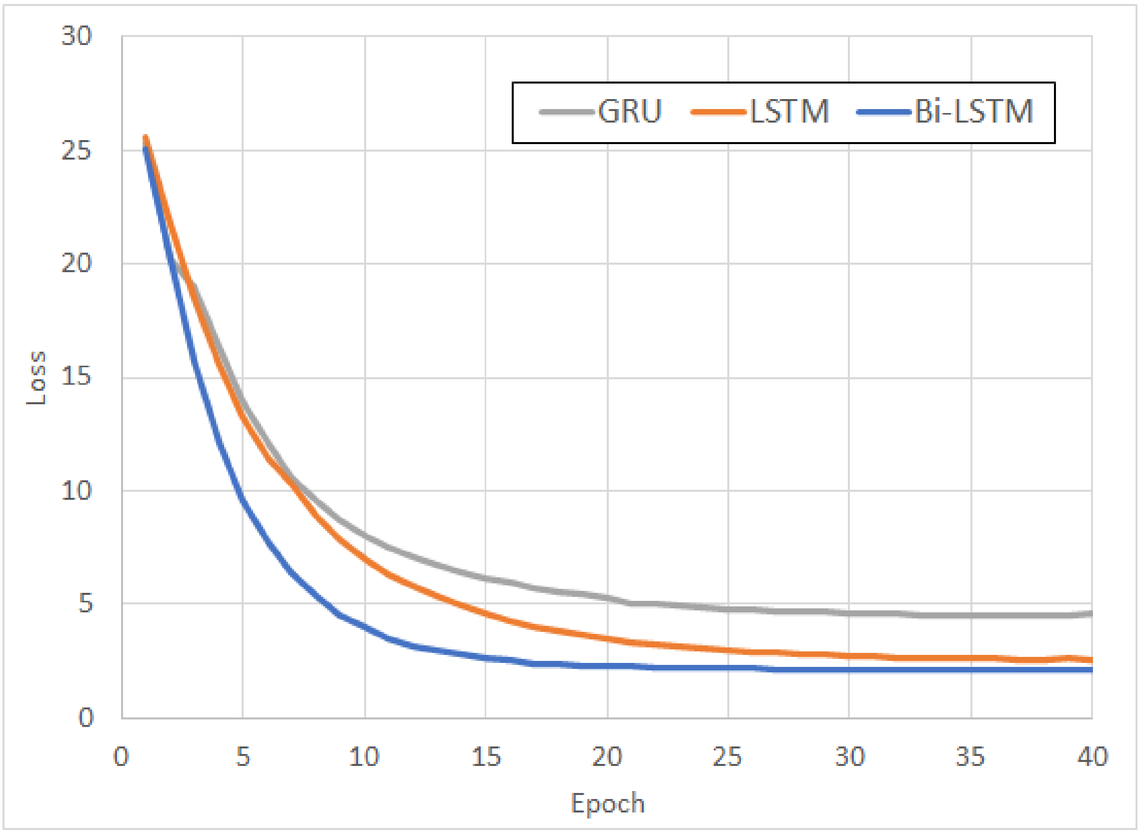


Figure 9: Loss with learning rate 0.0005 and 40,000 cleaned sentences

Theoretically, LSTM model should perform better with large dataset and long sentence, while GRU model supposed to compute faster because its architecture is simpler than LSTM. In practise, both models could have similar performance [12]. Our training show that GRU was faster than LSTM, GRU need 12 minutes to train one epoch, while LSTM need at least 20 minutes. Without filtered out the shorter sentences, both GRU and LSTM have the same performance. This was our expectation, because the sample dataset is smaller, and most of discussions in WallStreetBets are generally short sentences. By remove any sentences that shorter than 15 characters in the dataset, from the loss and perplexity score in table 1, we can see that the performance of LSTM is slightly better.

The Bi-LSTM model outperformed both GRU and LSTM model. It took longer time to train one epoch with roughly 30 minutes, but the loss converge faster and model achieved smaller loss compare with GRU or LSTM model.

Seed: "sir this": *sir this is the motel car park, we've called the cops.* Longer seed would result in *sir this is a casino.*, which is a unique way to tell other people they are in the wrong place in WSB.

Seed: "this is the kind of" : *this is the kind of degenerate lunacy this sub used to promote.*

Seed: "so risky" : *so risky to pick small gains lmao*

Seed: "gay" : *gay bear gang: faangt stocks were the market leaders through the covid recovery.* Notes that "gay" in WSB is not a negative term, "gay bear" mean a person who want to make profit when the stock market going down, as opposed to people believe that "stonks only go up".

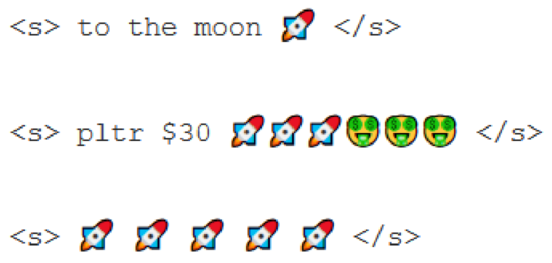
Seed: "covid affect" : *covid affect the few weeks , i was losing a bunch of money day trading options, so figured i'd buy a late january call and just let it marinate.*

Seed: "elon musk" : *elon musk made 7 billion dollars in the day now so we can make 2x the money on a lot of work is a shitty in.*

Parameters	N-grams 2,500	N-grams 5,000	GRU	LSTM	Bi-LSTM
Loss	N/A	N/A	4.26	2.54	2.13
Perplexity	57.32	69.38	70.81	12.73	8.45

Table 1: Model performance comparison.

From table 1, we can see that, overall, the perplexity score are low in our models. This could partly due to perplexity score favors short sentences and sentences in our dataset are short. We can see that n-gram model with 2,500 vocabulary size has perplexity score of 57.32, while the 5,000 vocabulary size has perplexity score of 69.38. This is because the perplexity score gets artificially low with many unknown words in the vocabulary. We can also see that Bi-LSTM model outperformed other model with a very low perplexity score.



```
<s> to the moon 🚀 </s>

<s> pltr $30 🚀🚀🚀🤑🤑🤑 </s>

<s> 🚀 🚀 🚀 🚀 🚀 </s>
```

Figure 10: Models can also predict sentences with emoji

We store our data with emoji, and as a result, N-grams and RNN models prediction can also include emoji in their result (see figure 10). It's interesting to see that if the seed is just one rocket emoji, the result will generate five rocket emoji. While other discussion forums try to mark emoji as spam, WSB encourage people using emoji instead of text. And usually if you type rocket emoji, it's better to spam several rocket emoji instead of one.

5 Conclusions/Discussion

In this project we built four variants of language models, n-grams, GRU, LSTM, and bi-LSTM to replicate forum discussions in Reddit's WallStreetBets. Then we compared these models with the perplexity score and by visually inspecting the generated sentences.

We found that the n-gram model starts to generate natural sentences when we set a large vocabulary size. However, with the vocabulary size we could set, the model still had many unknown tokens, which drove the perplexity score up from where it should have been. Additionally we realized that better predictions came at a cost of longer computation times.

We also found that the neural network models can produce sentences that look more like a conversation on WallStreetBets. In theory, the LSTM and Bi-LSTM models should have generated better sentences than GRU, but we found in our experiment that they actually created similar sentences. This may have been caused by our relatively small dataset, and another factor may be our computational limitations (our model can only run 40 epochs with less than 5% of our dataset).

During our experiment with the neural language models, we quickly realized we needed a GPU to train the network. We used Google Colab for this purpose, but ran out of usage allowance before training the models to the loss value we wanted. We also tried Google Cloud Computing and Amazon Web Service, but we quickly found that free tier, even with $300 credits for new customer, couldn't access GPU instances. Therefore our results are not as attractive as we want them to be. If we had enough resources, we would like to continue the training to see the models' true potential.

We also found that our predicted sentences are short, due to the nature of WSB's discussions. Another idea for future project would be to generate text to form a full submission (which contain several sentences). With each sample in the dataset containing a submission, the predicted result will retain more context than just a sentence.

6 Division of labor

Nhan collected and preprocessed the WallStreetBets text so it can be used as a training dataset. He was the one who discovered the Pushshift API after hours of struggling with the tool officially recommended by Reddit. He also transformed the SNLP assignment notebooks so it can run on the dataset and added LSTM and Bi-LSTM versions.

Ryoko mainly did research on methodology, and experimented with ways to train models on Google Colab with more GPU resources, which was a difficult and critical task because it frequently disconnected and erased all data. She also analyzed how a LSTM model works from a KDnuggets post [13], since both members had never seen one implemented before.

7 Acknowledgments

Though we did not seek help directly from any of these people, we would like to thank Jason Baumgartner and the developers of Pushshift API, the teaching assistants from the SNLP course, Alexander Ilin and the staff from the Deep Learning course at Aalto (CS-E4890), and Google Colab.

Without Pushshift API, we would have been stuck at the very start of this project without a way to effectively collect our dataset. We ended up creating our models from modifying the SNLP course assignments, so we feel very fortunate to have such well organized notebooks to use as building blocks. And the Deep Learning course made us familiar enough with Pytorch so that we could quickly modify and build variants of the RNN language models. Finally, we did not have enough computing resources to run RNN models, and we are thankful for the free 12GB GPU computing Google Colab provided us.

8 Appendix

We created a GitHub repository for our work, though it is not organized or documented like most public repositories found out there. The full raw dataset and cleaned dataset, which are 63,728KB and 56,756KB respectively, are too big to store in GitHub repository and can be shared with any person interesting in the project by contact the team members. The trained GRU, LSTM, Bi-LSTM models are even bigger in size, and also available upon request.

https://github.com/Usin2705/RetardBot

References

1. Darbyshire M, Fletcher L, Smith C, and Mackenzie M. Hedge funds rush to get to grips with retail message boards. Available from: `https://www.ft.com/content/04477ee8-0af2-4f0f-a331-2987444892c3`
2. Podolak M. PMAW: Pushshift Multithread API Wrapper. Available from: `https://pypi.org/project/pmaw/`
3. Baumgartner J, Zannettou S, Keegan B, Squire M, and Blackburn J. The Pushshift Reddit Dataset. 2020. arXiv: `2001.08435 [cs.SI]`
4. Jurafsky D and Martin JH. Speech and Language Processing (3rd ed. draft), Chapter 3. Available from: `http://web.stanford.edu/~jurafsky/slp3/3.pdf`
5. Olah C. Understanding LSTM Networks. Available from: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`
6. Gated recurrent unit. Available from: `https://en.wikipedia.org/wiki/Gated_recurrent_unit`
7. Hedengren J. LSTM Networks. Available from: `https://apmonitor.com/do/index.php/Main/LSTMNetwork`
8. Kawthekar P. Evaluating Generative Models for Text Generation. 2017
9. Cross Entropy. Available from: `https://en.wikipedia.org/wiki/Cross_entropy`
10. Boe B. PRAW: The Python Reddit API Wrapper. Available from: `https://praw.readthedocs.io/en/latest/`
11. r/wallstreetbets stats. Available from: `https://subredditstats.com/r/wallstreetbets`
12. Junyoung C, Caglar G, KyungHyun C, and Bengio Y. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. 2014 Dec. Available from: `https://arxiv.org/pdf/1412.3555v1.pdf`
13. PyTorch LSTM: Text Generation Tutorial. Available from: `https://www.kdnuggets.com/2020/07/pytorch-lstm-text-generation-tutorial.html`