# Learning the Language of Reddit Using RNNs

Budmonde Duinkharjav
Kerberos: budmonde
Massachusetts Institute of Technology

Danny Tang
Kerberos: data1013
Massachusetts Institute of Technology

***Abstract*—This paper explores the effectiveness of the RNN and LSTM models in their ability to generate text in the context of Reddit comments; it also analyzes whether these models are capable of emulating the Reddit language style.**

***Keywords* – LSTM, RNN, NLP, Reddit**

## I. Introduction

Reddit.com is a social news aggregation, web content rating, and discussion website (description from Google). It features a variety of different communities called subreddits, each centered around a different theme or topic. For example, the "pics" subreddit features photographs and pictures, and the "politics" subreddit features discussion about American politics. In these subreddits, users can make posts called threads to start conversations, and other users may comment on these threads to express their own thoughts on the matter at hand. Threads and comments may also be upvoted or downvoted by other users, allowing users a quick way to rate the content they see.

As Reddit is an online community, it is rich in Internet lingo and Reddit specific jargon. For our project, we will be using generative models such as basic Recurrent Neural Networks (RNNs) and Long Short Term Memory networks (LSTMs) to "learn" the language specific to Reddit. To do so, we will be training our networks on comments from threads across a variety of subreddits. Analysis on our results will be made with a combination of quantitative (ie. comparing character distributions of our generated text to the training set and to randomly generated text) and "common sense" (ie. do these outputs sound like comments?) arguments.

## II. Related Work

In this paper we refer mainly to the implementations of RNNs and LSTMs in "The Unreasonable Effectiveness of Recurrent Neural Networks" [1], with the original LSTM algorithm as proposed in "Long Short Term Memory" [2]. We also refer to the blog post "Understanding LSTM Networks" [3] for background information.

## III. Dataset and Features

In this section, we will discuss the dataset we used for our models as well as the feature space that was used for analysis.

### *A. Dataset*

We collected our data from Reddit's "top" page which is a ranked listing of the most upvoted threads of all time, containing threads with up to 200,000 upvotes. These threads are a good measure of what kind of content the Reddit community appreciates and what type of people make up the Reddit culture. By examining the large variety of comments these Reddit threads have, we are able to sample the types of conversations that are most likely to occur within the Reddit community. To obtain these comments we used PRAW (Python Reddit API Wrapper) to scrape every single comment in the top 21 threads that were on this listing, resulting in a total of 95,964 comments

with each comment containing about 23 words on average. Below, we demonstrate a sample comment from the data.

> *"I really don't understand how reddit works sometimes*
> *Edit: Thank you, kind stranger; for my first gold and further confusing me about what reddit likes and does not like."*

### B. Features

The basic implementation of the RNN and LSTM algorithms we will discuss below uses a set vocabulary as the space of possible inputs. We use the comments we collected as a stream of characters as an input to the models by:

- stripping a comment of non-ASCII characters to keep the feature space tractable
- stripping the comments of multiple spaces and new-line characters to eliminate possible noise within comments.

## IV. Method

In this section we will discuss the motivation and architectures that RNNs and LSTMs are built upon as well as how these models fit our problem. We will also discuss how we will quantitatively evaluate our results.

### A. The RNN Model

*1) Motivation:* The Recurrent Neural Network (RNN) Model is a type of neural network that aims to incorporate past information into its outputs – it can be compared to a Markov Chain version of a neural network. Its ability to persist information through time allows it to decide how future predictions are affected by current ones. In Figure 1, we show a diagram of the recurrent structure of RNNs: The output of each query is passed into the next iteration, allowing for a flow of information from *any* time step $t$ to another timestep $t'$ given that $t < t'$. With such an recurrent structure, the input is passed into the network as a stream until some end-of-input marker is passed into the network at which point the recurrence halts. Notice that this structure allows us to pass in inputs of variable size, an advantage the RNN has over conventional neural networks where the number of layers has to be explicitly specified. This quality is what ultimately provides the RNN the flexibility to be applied to various NLP problems where the input length is almost always varied.

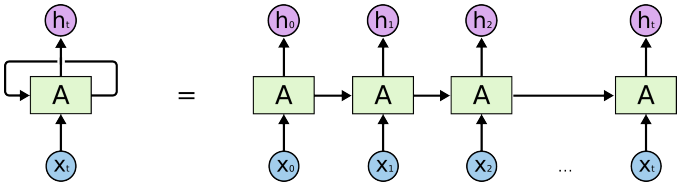


Fig. 1. RNN. We can see how the recurrent structure allows the RNN to unroll to a variable length depending on the length of the input. Picture from "Understanding LSTM Networks" [3].

As such, in the context of text generation, RNNs are a fitting choice because they allow for outputs of varying sizes; we may generate as many outputs as we want because we can control when the network stops.

*2) Implementation:* The most basic RNN consists of a hidden state $h$, which contains the history of previous calculations. It takes the form of a $n$ dimensional vector in the range $[-1, 1]^n$. This state is updated after each recurrence with the update function:

$$h_t = \tanh(W_h h_{t-1} + W_x x_t). \tag{1}$$

Here, the subscripts $t$ and $t-1$ denote the temporal progression of the input $x$ and the hidden state $h$. The $W_h$ and $W_x$ are weight matrices that correspond to the hidden state along with $W_y$ which we will use for the output function below:

$$y_t = \text{logsoftmax}(W_y h_t) \tag{2}$$

This RNN structure is now able to fully compute forward-propagations as well as back-propagations using cross-entropy since the output activation function is a Log-Softmax output. Using this basic building block we are able to construct more complex, deeper structures by feeding the output $y_t$ of the RNN into another RNN to produce deeper outputs, $z_t$'s, and so on.

As such, we are given control of two adjustable hyper-parameters which affect how the network behaves during training: the dimensionality of the hidden state and the number of RNN layers.

### B. The LSTM Model

*1) Motivation:* Unfortunately, there is a caveat to using RNNs to predict text. If we look at Equation 1, we see that over many iterations of updates, the contribution of the earliest updates die out due to the tanh normalization of the hidden state at each iteration. This is not good, because there are many occasions in natural languages where information needs to be preserved long-term. An example of this phenomenon is the following query:

> *"We created a Git repository for the 6.867 project... Danny cloned..."*

We argue that if the entire context of the input that we had was *"Danny cloned"*, we would not be so inclined to predict *"repository"* as if we had also read the earlier part about *"creat[ing] a Git repository"*. Similarly, natural languages often require the reference of long-term memories of states to persist over many recurrences to be able to predict the most likely output. In order to achieve this, we look at the LSTM model as proposed by "Long Short Term Memory" [2]. This model solves the issue RNNs have by introducing a second hidden state that allows for long-term memory persistence.

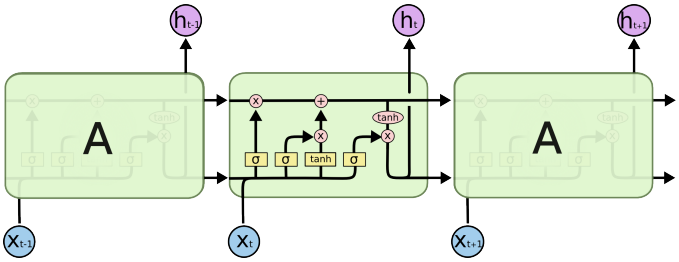


Fig. 2. The LSTM structure. Every yellow box is a neural netowrk layer with trainable weights while the red circles are pointwise operations. Picture from "Understanding LSTM Networks" [3].

In Figure 2, we look at the basic LSTM network. On the macro scale, the structure is very similar to RNNs in that the LSTM passes hidden states onto the next recurrence. However, in this case, instead of a single hidden state, there are two hidden states, $h$ and $c$. The use of $h$ is the same as in a basic RNN, while the long-term memory hidden state, $c$, is used to allow data to persist over long chains of recurrences. $c$ makes this possible because it is updated with addition, as opposed to multiplications by values in the range [0, 1].

*2) Implementation:* Implementations of LSTMs may be different: for example, they may use slightly different ways of connecting their neural network nodes, or they may copy weights across different layers for compactness. The model we used for our experiments is the one shown in Figure 2: all of its network layers have independent weights. We can see that just like with regular RNNs, there are arrows from inputs to outputs, each following linear combinations of the internal neural network layers to determine the new states $c$ and $h$. The information flow within the network represents the following intuitions behind them:

- We should store the hidden state $h$ and input it to the long-term memory $c$
- We need to decide which parts of $c$ should

stay relevant at each recurrence
- We should allow $h$ (which ultimately decides the output $y$) to be affected by $c$

Since an LSTM has these three qualities, we should be able to perform better than an RNN in the context of text generation. We will discuss the results of this hypothesis in Section V.

### C. Evaluation Scheme

In order to judge the effectiveness of our models, we will employ two main strategies: a quantitative analysis comparing our models' outputs to their training sets and expected random outputs, as well as an appeal to the reader's understanding of the English language through provided examples of outputs.

For the quantitative analysis on our models, we will use two main metrics together: Kullback-Leibler divergence (KL divergence) and "real word ratios."

The KL divergence values we use will be taken between the probability distributions of characters in two files of text. Our first distribution will be that of the training set text used to train the model of interest. The second distribution will either be that of our test file, containing 20,000 character outputs of text from the model, or a uniform distribution of the characters in the training text, representing a random output. In general, the KL divergence will tell us the difference between two probability distributions $P$ and $Q$, when we use $Q$ to estimate $P$. In the context of our analysis, we are trying to measure how different our model outputs are from our training text.

Then, let $C_1$ be the set of unique characters in file 1, $C_2$ be the set of unique characters in file 2, $S = C_1 \cap C_2$, $P_1$ be the probability distribution of characters in file 1, and $P_2$ be the probability distribution of characters in file 2. Values from $P_1$ and $P_2$ are in the range [0, 1]. The KL divergence between $P_1$ and $P_2$, written as $D_{KL}(P_1||P_2)$, is then given by:

$$D_{KL}(P_1||P_2) = \sum_{c \in S} P_1(c) \log \frac{P_1(c)}{P_2(c)} \qquad (3)$$

We also note that $\sum_{c \in S} P_1(c)$ and $\sum_{c \in S} P_2(c)$ may not equal 1 if $C_1 \neq C_2$, but this is okay for our analysis since we are trying to measure the relative effectiveness of different $P_2$'s on the same $P_1$.

| KL Divergences From a Reddit Thread | |
|---|---|
| Character Dist. From... | KL Divergence |
| Other Reddit Thread 1 | 0.00521 |
| Other Reddit Thread 2 | 0.00593 |
| Wikipeida Article (Ball) | 0.07091 |
| Wikipedia Article (Dance) | 0.06298 |
| Alice in Wonderland | 0.10111 |
| Hamlet | 0.03268 |
| Romeo and Juliet | 0.01514 |
| The Book of Genesis | 0.06518 |
| Obama Acceptance Speech | 0.01927 |
| Trump Acceptance Speech | 0.00229 |
| Uniform Dist. | 1.07563 |

Fig. 3. Table showing the KL Divergences from the character distribution of a Reddit thread, of character distributions from other sources.

In order to check that KL divergence is a reasonable metric of analysis, we calculated the KL divergences from one of our scraped Reddit threads against text from other sources. The results are shown in figure 3. As we would expect, other Reddit threads are in general closer to our original thread than text from other sources. This difference in proximity is quite noticeable, as it is generally equal to about an order of magnitude. Variance among these other sources could be attributed to various factors, such as a difference in writing styles, dialects, or word complexity. As a final sanity check, we measured the KL divergence with a uniform distribution of characters, which returned a really high value relative to

our other sources. This result makes sense, as we would not expect a random selection of characters to well represent a language.

Meanwhile, the ”real word ratios” metric measures the ratio of ”real” to total words in the output text generated by our models. Specifically, it is given by:

$$\frac{\sum_{w \in W_o} 1\{w \in W_t\}}{|W_o|} \tag{4}$$

where $W_0$ is the set of unique words in the output text and $W_t$ is the set of unique words in the training text. We use this in conjunction with the KL divergence in order to ensure that we are not misled by character distributions alone - we have to make sure that our output gives us readable English words as well.

We'll also briefly explore one other metric: the character probability distributions themselves, of shared characters between training text and output text. This will be used to better visualize some of our results.

## V. Experiment Results and Analysis

Our experimentation consisted of three main parts: training different basic RNNs on a smaller dataset, training different LSTMs on a smaller dataset, and training the network with the best results from the previous two parts on a larger dataset. The smaller dataset consisted of a randomly chosen Reddit thread containing 142,805 characters, and the larger dataset consisted of multiple Reddit threads, containing a total of 4,570,667 characters. The KL divergence between the large dataset and the small dataset is 0.00218, so we hoped that results on the small dataset would translate well to the larger one. In general, we trained our models with varying hidden states, $n$, and layers, $l$, with a fixed batch size of 50 over 50 epochs.

In the first two parts, we will analyze KL divergence, real word ratios, and samples of text outputs from our models. We will also briefly explore the character distributions themselves, of our training set text and model outputs for select LSTM parameters. For the last part, we will mainly look at samples of text outputs, and determine whether or not our model ”works.”

| KL Divergences From Basic RNNs | | | |
|---|---|---|---|
| $n \backslash l$ | 2 | 3 | 4 |
| 128 | 0.0023 | 0.0036 | 0.0025 |
| 256 | 0.0051 | 0.0060 | 0.0042 |
| 512 | 0.0559 | 0.0117 | 0.1095 |

Fig. 4. Table showing the KL Divergences from the character distribution of a Reddit thread, of 20,000 character outputs from RNNs with varying hidden states, $n$, and layers, $l$.

### A. Basic RNNs

Figure 4 shows us the KL divergences between the small dataset and outputs from different basic RNNs. On initial glance, it looks like all of the RNNs with $n = 128$ and $n = 256$ performed well; these results are all better than or comparable to the values from other Reddit threads, as seen in figure 3. Meanwhile, the RNNs with $n = 512$ appear to perform poorly. There is an obvious increasing trend in KL divergences as we increase $n$, which was surprising to us. We had hypothesized that more hidden states in the RNN would allow the network to better model itself after our inputs, causing the KL divergences to follow a decreasing trend instead. The actual result may have occurred because of incompleteness in model training. It is possible that by adding more hidden states, we increased the number of parameters to the point where they could not all be developed in the same number of epochs. The results from smaller $n$ may also be an indicator that we do not need so many hidden states in our model at all.

There also isn't a clear trend as we increase $l$. For $n = 128$ and $n = 256$, our values increase and then decrease, while for $n = 512$ our values decrease and then increase. This may

| Real Word Ratios From RNNs | | | |
|---|---|---|---|
| $n \backslash l$ | 2 | 3 | 4 |
| 128 | 0.5859 | 0.6338 | 0.6446 |
| 256 | 0.6381 | 0.5343 | 0.1027 |
| 512 | 0.1194 | 0.1095 | 0.1027 |

Fig. 5. Table showing the ratio of real to total words in the 20,000 character outputs from RNNs with varying hidden states, $n$, and layers, $l$.

| KL Divergences From LSTMs | | | |
|---|---|---|---|
| $n \backslash l$ | 2 | 3 | 4 |
| 128 | 0.0018 | 0.0040 | 0.0034 |
| 256 | 0.0025 | 0.0044 | 0.0051 |
| 512 | 0.0122 | 0.0057 | 0.0023 |

Fig. 6. Table showing the KL Divergences from the character distribution of a Reddit thread, of 20,000 character outputs from LSTMs with varying hidden states, $n$, and layers, $l$.

be because different information is persisted throughout each RNN layer - since each layer may behave differently in how they adjust their parameters, we may run into inconsistencies when we combine them.

Figure 5 shows us the real word ratios of our outputs from basic RNNs. Roughly following the pattern of results in figure 4, we have better ratios for $n = 128$ and $n = 256$, and worse ratios for $n = 512$. The best ratios come from the $(n, l)$ pairs of $(128, 3), (128, 4)$, and $(256, 2)$, which seems to suggest that for this training set, RNNs with less parameters are better able to model our Reddit comments. This makes sense, because with more parameters, we may overfit on the training set's character distributions, resulting in less real words. Meanwhile, the worse ratios from $n = 512$ supports our previous claim that these models may not have been completely developed.

Similar to the KL divergence, there is no consistent trend as we increase $l$. We believe the reason for this to be the same as what we discussed while we were analyzing figure 4.

Having considered both tables for basic RNNs, it seems that our worst model is given by hyperparameter pair $(512, 4)$, and our best model is given by $(128, 4)$. We provide one sample comment made by each model that we think well represents the entire output from them:

> *"abennttSgenaislWnhs aptI 'fi? ht ltoy eiaih a oo ahnti/dctssobd Gaddhoiy"*

> *"Bernie find that bely be the teally have about not just voting cromination is post and deciabs"*

We can see that the first model's output is completely nonsensical in the context of the English language. It does, however, have some resemblance of a sentence, as it appears to split the characters into words with spaces. The second model's output appears to be much better: it contains actual words and context - the reader can infer that the original thread may have been about the US presidential election. These outputs support our quantitative analysis and act as a means of keeping us on the right track as we test other models.

### B. LSTMs

Figure 6 shows us the KL divergences between the small dataset and outputs from different LSTMs. It seems as if these values do not differ much from those in figure 4 for $n = 128$ and $n = 256$. There is, however, an improvement in results for when $n = 512$, compared to the basic RNN models. This may be attributed to how LSTMs are better able to persist information: they are able to get rid of noise more quickly than basic RNNs are.

Once again, we are unable to pinpoint a trend of behavior in varying $l$. We will follow the same line of reasoning as before and attribute this to inconsistencies within updates among the RNN layers.

| Real Word Ratios From LSTMs | | | |
|---|---|---|---|
| $n \backslash l$ | 2 | 3 | 4 |
| 128 | 0.6062 | 0.4343 | 0.4455 |
| 256 | 0.6191 | 0.6513 | 0.6521 |
| 512 | 0.7637 | 0.8227 | 0.1024 |

Fig. 7. Table showing the ratio of real to total words in the 20,000 character outputs from LSTMs with varying hidden states, $n$, and layers, $l$.

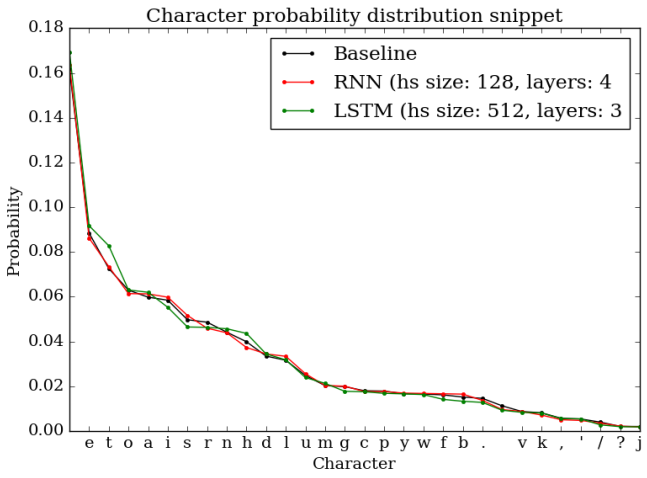


Fig. 8. Probability distribution of sampled characters in a model (top 30 most likely). The LSTM and RNN models are able to emulate the character distribution with the baseline text well and neither model is distinctly better than the other. The distinction in quality of results can be inferred when we compare the real word ratios.

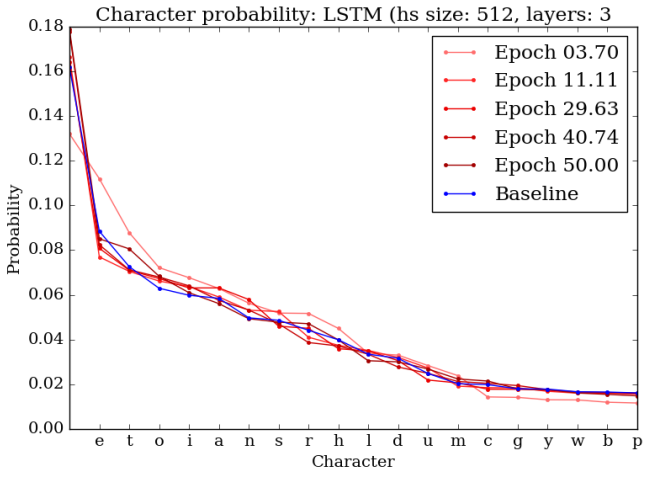


Fig. 9. Probability distribution of sampled characters in the LSTM model at different epochs of training. We can see that the model improves in terms of the probability distribution, approaching the baseline distribution

Figure 7 shows us the real word ratios of our outputs from LSTMs. We see that compared to basic RNNs, we are able to reach a much better value with LSTMs, with a maximum of 0.8227 with hyperparameters $(512, 3)$ compared to 0.6446 achieved with $(128, 4)$ in a basic RNN. This, coupled with the graph in Figure 8 and our KL divergences gives us an important distinction between LSTMs and basic RNNs: while their KL divergences may be comparable, LSTMs give us better real word ratios. In other words, while both basic RNNs and LSTMs are able to model the character distributions of comments in our Reddit threads, LSTMs are better at maintaining the structure behind these characters. This gives us a better model when we want to generate the language of Reddit.

As we have already seen what a nonsensical output looks like, this time we will only look at outputs from the best LSTM model, and compare them to the output from the best RNN model. Upon examining the LSTM tables, it seems like our best model is from hyperparameters $(512, 3)$. Here are three sample comments from this model:

> *"Congratulations; you've found the karmic singularity."*

> *"That post his culring merit!"*

> *"Birdie Sanders 2016] (https://i.ytimg.com/vi/vhJVNLYq4bA/hqdefault.jpg)"*

This is quite impressive! The first comment makes complete grammatical sense, and even manages to capture some Reddit-specific language (karma refers to the number of upvotes or downvotes users have acquired from posts and comments). The second comment makes less sense, but has perfect sentence structure, ending with an exclamation mark. The third comment contains what looks like a political joke, as well as a perfectly structured url (that I wouldn't recommend

going to, because it is a random link from Reddit). From an English standpoint, these results are clearly preferable to the models given by our basic RNNs.

As a last measurement of effectiveness before we train a model with these hyperparameters on our larger dataset, we'll look at the character distributions of the intermediate models during training. This is shown in figure 9. The blue line represents the character distribution of the training set itself, while all of the red lines belong to models at different epochs of training. We see that the training indeed causes the character distributions to converge to the training set. Since training on these hyperparameters behaves as expected, we'll use them to train on our larger dataset.

### C. A Larger Dataset

To recap, we will use the network and hyperparameters that gave us the best results in the previous two parts on our larger dataset - this is the LSTM network with $n = 512$ and $l = 3$. As a result, we end up with a KL divergence of 0.0071 and a real word ratio of 0.9404. We see that the KL divergence is slightly higher here than it was for our smaller dataset (0.0071 compared to 0.0057), while the real word ratio has improved by 0.1177. After seeing these two metrics, we predicted that our final model's generated comments would be better than our previous models, in the context of the English language. Here are the first 6 outputs from the model:

> *"arger_carge there" my jaf at here. Google Pics, for example."*
>
> *"That is a strange to office your time."*
>
> *"Honestly, that's actually more than going on you just canaring light now."*
>
> *"Can I turn it on that picture, like the guy deleted the election."*
>
> *"That is a staged stock market."*
>
> *"Lol why did you even read that? I'm so gonna be aware off with this much. Islam is actually :)"*

It's hard to tell whether or not the output is truly better than before, but this definitely isn't bad! We can see that the model has learned to end all sentences with some sort of punctuation, and most of the words in these comments are actual words. The last comment is of particular interest: it starts with "lol", which is well known to mean "laugh out loud", and ends with ":)", which is a smiley face. In general, these comments don't make complete sense, but are really close.

## VI. Conclusion

After training many generative models on our comments from Reddit, we conclude that it is *probably* possible for RNNs (specifically LSTMs) to learn the language of Reddit. While our final results' comments don't make perfect sense, they are closer than not to sounding like real sentences from users. This leads us to believe that if tweaked even further, LSTMs should be able to achieve even greater results and generate real sounding comments.


## Acknowledgment

The authors would like to thank...

the 6.867 staff, for providing us with the tools and resources necessary to learn so much about machine learning.


## References


[1] Andrej Karpathy. *"The Unreasonable Effectiveness Of Recurrent Neural Networks"*. 2015. Accessed December 13 2016. karpathy.github.io/2015/05/21/rnn-effectiveness/.

[2] Sepp Hochreiter and Jurgen Schmidhuber. *"Long Short Term Memory"*. 1997. Accessed December 13 2016. deeplearning.cs.cmu.edu/pdfs/Hochreiter97_lstm.pdf.

[3] Christopher Olah. *"Understanding LSTM Networks"*. 2015. Accessed December 13 2016. colah.github.io/posts/2015-08-Understanding-LSTMs/.