Reddit Comment Generator - Project Report

Braulio Chavez
braulioc@stanford.edu

Abstract

In this work I describe the implementation of a recurrent neural network language model (RNN-LM) to solve the problem of automatic generation of logical comments for a specific context. The RNN makes use of LSTM cells to help keep long term data dependencies, improving the performance of the model. For completeness I compared the training evaluation of the LSTM cell vs the GRU cell under the same conditions. Then I discuss the reactions that the generated comments got from the Reddit community and how to improve the model in order to give it more context and understanding of sentences.

1 Introduction

Recurrent neural networks (RNN) have been used in a wide array of machine learning problems. They are known to be a generative model which is a particularly interesting property. Recently there have been various attempts to increase the creativity of these models. These approaches are applied to generate a varied amount of media like images, music or poems.

In this project I make use of the generative properties of the RNN to try to automatize the generation of comments specific to a provided context. The applications of this are many: FAQ answering, forum like discussion, chatbots that help accomplish a task, gmail's auto-reply feature.

The potential of this is huge, it can create be a whole new human computer interaction paradigm on how users deal with technology. If implemented correctly people won't even be able to discern a bot from a human on the other side of the conversation (Turing test). This applies to any form of communication. Companies wouldn't need to spend as many resources to generate a valuable product or experience to their customers, think of customer service and internal processes, even increase the usefulness and intelligence of their current products.

The approach to solve this problem was to use already proved RNN language models. I used the help of LSTMs to help alleviate the long term data dependency loss issue, this practice improved the performance of the model. In order to have a baseline I also experimented with a GRU which in practical terms it is a modified LSTM with different keep and forget functions and internal mechanisms which I'll describe later. Once these two models were trained I evaluated them with perplexity.

2 Background/Related Work

Previous approaches have been taken to solve this problem. This project is highly influenced by the work *Recurrent neural network based language model* by Mikolov et al. 2010 and can be thought as a practical application of the RNN language model with small variations. The main modification was the use of LSTM cells on the model structure. Such cell was first proposed by Hochreiter and Schmidhuber 1997 in the paper *Long Short-Term Memory*

3 Approach

After some experimentation (described in the next section) with different hyperparameter configurations, and by comparing the use of LSTM and GRU cells. I arrived to the following language model used for comment generation. The model is a stacked RNN with two hidden layer LSTM cells to help with the long term data dependency loss problem. Each hidden layer has 200 nodes. You can see a graphic representation in figure 1.

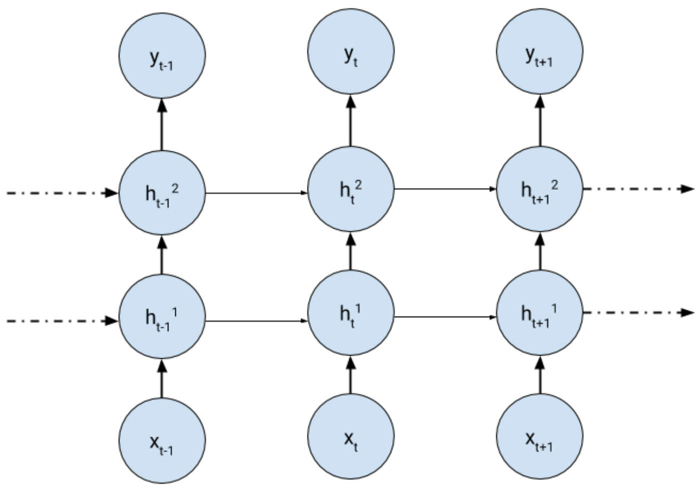


Figure 1: Stacked RNN.

Let's start defining the equations for the LSTM cells. (Zaremba el al. 2014) The memory cell c_t contains the information of a unit. The gates control how much of that information should be memorized or forgotten to be sent to the next step.

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \widetilde{c}_t^j \tag{1}$$

where

$$\widetilde{c}_t = \tanh\left(W_c x_t + U_c h_{t-1}\right) \tag{2}$$

Input i_t and forget f_t gates are calculated from previous hidden states. h_{t-1}

$$i_t = \sigma(W_i x_t + U_i h_{t-1}) \tag{3}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1}) \tag{4}$$

$\sigma(.)$ is an element-wise logistic sigmoid function. x_t is the input vector.

When we have the result of the memory content. The hidden state h_t^j of the j-th LSTM unit is calculated as:

$$h_t^j = o_t^j \tanh\left(c_t^j\right) \tag{5}$$

The output gate o_t controls how much memory is exposed:

$$o_t = \sigma(W_o x_t + U_o h_{t-1}) \tag{6}$$

LSTM cells help to memorize long term data dependencies.

The model's prediction $\hat{y}_t$ is obtained by:

$$\hat{y}_t = \text{softmax}(h_t^2) \tag{7}$$

Notice that *softmax* is is being applied only to the topmost hidden layer on the stack. In our rnn diagram (see figure 1) $h_t^2 = y^t$

The evaluation of the model was done with a technical metric: *perplexity*, and by engagement metrics: *users' replies* and *upvotes*. You can think of perplexity as a measure of how perplex or surprised is the model of comparing the results it created with a validation dataset. To obtain perplexity we must first calculate *loss*. Loss allows the model to determine how far away from the correct result is its prediction, generally speaking the less the distance the better. The model optimizes for minimizing this loss. Note that optimizing too much can cause *overfitting*, making our model incapable of generalize its guesses.

$$\text{loss} = -\frac{1}{N}\sum_{i=1}^{N} \ln(\hat{y}) \tag{8}$$

Then perplexity is defined as:

$$\text{perplexity} = e^{\text{loss}} \tag{9}$$

4 Experiment

The dataset used in the experiment was 31 GB of an entire month's of reddit comments. It has comments metadata such as: score, subreddit, body and author. I sectioned it to use a single subreddit (theme/topic). I decided to use the "funny" subreddit, which is one of the most popular subreddits. This subset arrived to 450 MB of comments. Then to bias the model towards better quality comments pre-evaluated by Reddit's community with the score field, I filtered the comments to only keep the ones with score bigger than 10 (see figure 2). After this sectioning the dataset used for training was about 13% of the comments for the "funny" subreddit. Then I generated a file to contain all of the comment's body, did a further separation to create a train, test and validation datasets with 80%, 10% and 10% original body file size correspondingly.

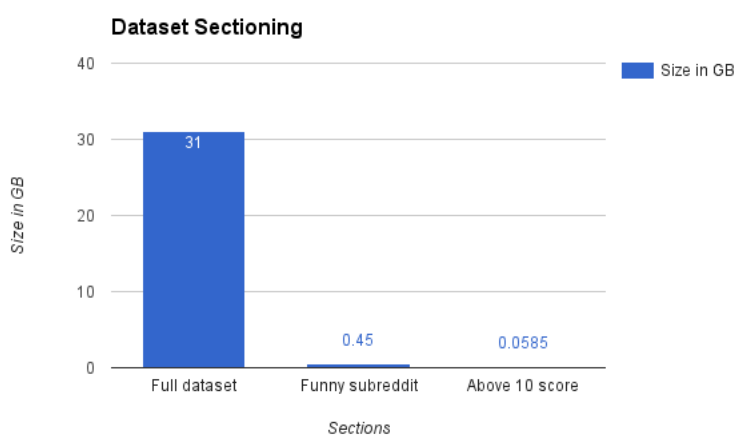


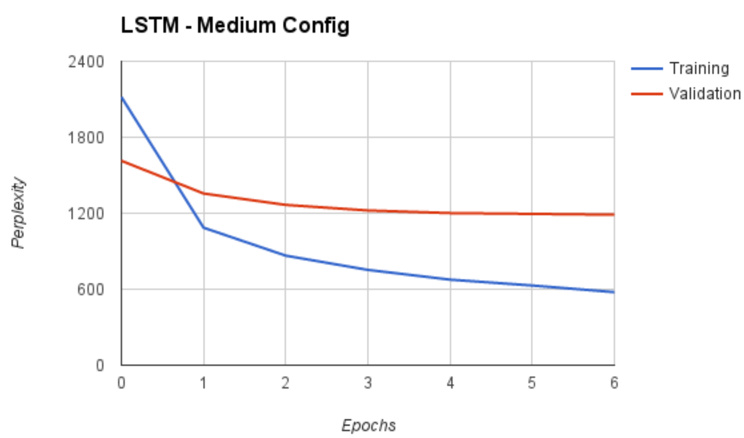
Figure 2: Dataset sectioning.

With the dataset ready I proceeded to train two models. One with LSTM and one with GRU cells. This helped me compare the two models and choose the one that performed the better based solely on perplexity. I used two set of hyperparameter configuration to train the models. You can see the hyperparameter sets used in Table 1.

Table 1: Used hyperparameter configurations for testing.

Parameter	Small	Medium
batch_size	64	64
embed_size	50	50
hidden_size	100	200
num_steps	10	20
max_epochs	16	26
early_stopping	2	2
dropout	0.5	0.5
learning_rate	0.001	0.001
num_layers	2	2

One of the main differences in both set of hyperparameters was the increase of the hidden_size parameter. By increasing it I arrived to better performance in perplexity in both models. You can see the comparison of performance of both models on figure 3.



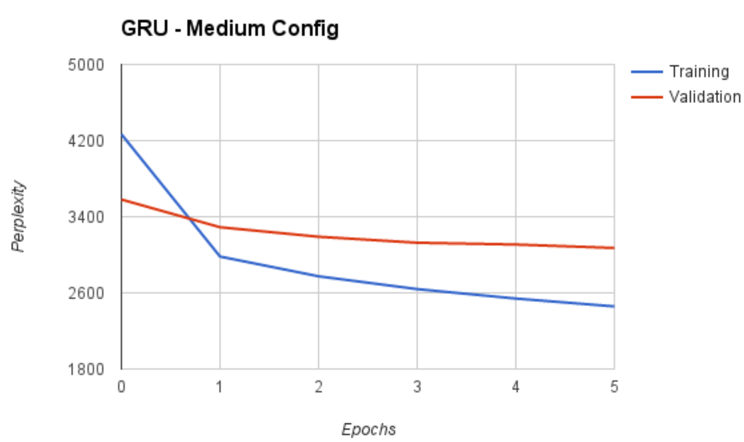


Figure 3: LSTM vs GRU models in training and validation perplexity.

We can clearly see that the model using LSTM cells was better than the one using GRU cells at the given task with the same parameters. Given that conclusion I decided to use the LSTM model's to generate test sentences.

To distribute the comments generated by my RNN I created a Reddit account called Roy_Nexus. I deliberately avoided to give any clear indication that this was a bot, just to see people's reactions to it.

The bot worked as follows: It retrieved the top result of the feed "rising" under the subreddit "funny". The decision to select this specific feed was because it has posts that are not yet popular but have the potential to be at the front page, by posting in here the bot had a chance of being relevant in the case of selecting a future popular post. Once the bot had downloaded the comment's information, it used the comment's title to give it as a seed of the RNN's generative function. This way I provided some context to the RNN in hopes of receiving back a sentence that was within the boundaries of the theme. Once the RNN gave the output words, the bot uploaded them in a comment under the previously selected post. Then the program slept for 15 minutes before repeating the whole process again. The bot never commented twice on the same post.

The reaction obtained by the community was very clear: the comments generated by the RNN were mostly non-sense or out of context. This resulted in a negative response measured by the score the content received and by replies saying things like "are you having a stroke?" or "somebody has a case of Google translate it seems". You can see some screenshots of the generated comments on figure 4. And you can see the some of the replies on figure 5. Notice the reply of the user that identified what I was doing to generate the sentences.

Figure 4: Sample of the generated comments.

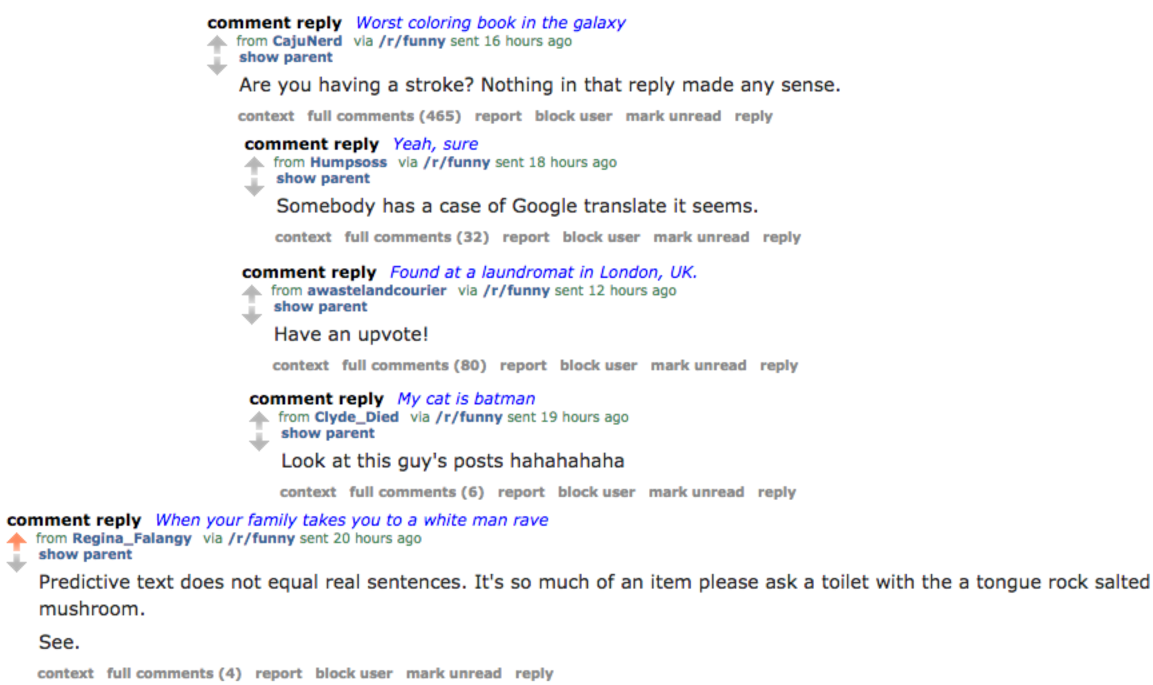


Figure 5: Sample of the replies by users.

Table 2: Engagement metrics.

Concept	Value
Comments	41
Replies	20
Score	-65

After some days of publishing content on Reddit the the bot got tracked down and banned from the subreddit "funny" by the moderators, giving end to the experiment. On Table 2 you can see a summary of the number of comments, replies and final comment score the bot got when finished the experiment.

5 Conclusion

After such a sudden forced end of my experiment. I concluded that the RNN didn't have any sufficient consideration for the context of which it was being asked to generate text. The title seed provided wasn't enough to guide the RNN to an specific theme, much less to a set of themes. This task proved to be a very hard problem.

I also think that the perplexity metric can be improved by first using more data to train the model, second giving it more time to train and third increasing the size of the hidden layer (as experiments concluded).

For further extensions it would be illustrative to use a Gated Feedback RNN (GF-RNN), this model had shown to be superior to the basic stacked RNN-LSTM, RNN-GRU models in previous experiments and are promising. Is also interesting to mix different models to aid in the problem of context and sentence understanding. For example, integrating to a syntactic parser, that works by passing the input sentences to tag each word with a part-of-speech (POS) tag that describes the word's syntactic function, and determines the syntactic relationship between words in the sentence, represented by a tree (see figure 6). This will help to understand the input and output sentence of the RNN.

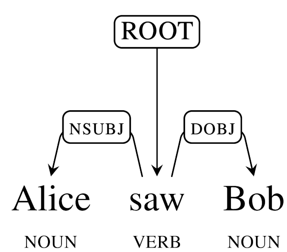


Figure 6: Parse tree of the phrase: "Alice saw Bob"

6 References

Mikolov, Martin Karafit, Luks Burget, Jan Cernock, Sanjeev Khudanpur: Recurrent neural network based language model. INTERSPEECH 2010: 1041048

Sepp Hochreiter, Jürgen Schmidhuber: Long Short-Term Memory. 1997.

Zaremba, Wojciech and Sutskever, Ilya. Learning to execute. arXiv preprint arXiv:1410.4615, 2014.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, Yoshua Bengio: Gated Feeback Recurrent Neural Networks. arXiv:1502.02367. 2015

Reddit. `www.reddit.com`

Reddit dataset used `https://www.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment`

Announcing SyntaxNet: The Worlds Most Accurate Parser Goes Open Source. `http://googleresearch.blogspot.com/2016/05/announcing-syntaxnet-worlds-most.html`