Predicting and Generating Discussion Inspiring Comments

Yunhe (John) Wang
SUNetID: yunhe

Junwon Park
SUNetID: junwonpk

Abstract

Can an artificial intelligence understand and inspire discussions in online communities? To answer this question, we will attempt to predict whether a given comment on Reddit will produce descendant comments, and then generate comments that lead to the most follow-up comments.

1 Introduction

It is possible but non-trivial for human participants of online communities to read and lead the discussions. In other words, a human participant can understand which comments to respond to, and how to write comments to attract the comments of others. Our project attempts to build models that understand the subtleties that make a comment popular, and both predict and generate comments that have most descendant comments.

We define a descendant d of a comment c to be a comment that is "eventually a reply" to c. By "eventually a reply", we mean that d could be a direct reply to comment c, or a direct reply to a direct reply to comment c, etc. Thus, comments that generate the most discussion can be thought of as the ones that have the most descendant comments.

2 Background/Related Work

For prediction, no papers have been found that directly address predicting the number of descendant comments in Reddit. However, CNN models were seen to be successful in predicting whether or not a tweet will be retweeted by a user in [1], which is an inspiration for our use of CNN models, in addition to LSTM models that are now standard in NLP.

For generation, there is one relevant paper. Using the vanilla RNN to generate a text suffers the vanishing gradient problem, which makes it difficult for a model to use information from many time steps ago to predict the word at a given time step. The paper Long Short-Term Memory introduces an improvement to RNN that provides multiple paths for a gradient to flow through hidden layers, thereby preventing a gradient from multiple time steps ago from being vanished [2].

3 Prediction

3.1 Dataset

We use the Reddit Comments Dataset, which contains comments from every subreddit from the last 15 years. From this dataset, we've extracted comments from the "news" subreddit from between January 2016 and March 2017. Each comment comes with relevant metadata such as author, post time, a link to the article etc. In addition, we processed the data and annotated them with further information, including how many descendant comments each one has, the parent comment, the response time, the time of day it was posted, and the day of week it was posted. Here, the parent comment refers to the comment that the given comment was a reply to, and the response time refers to the time between the posting of the given comment and the posting of its parent comment (ie how long it took for the comment to respond to its parent comment). Furthermore, we used the VADER sentiment analyzer [3] to extract rule-based sentiment features.

After cleaning and pairing comments with their parent comments, we split the dataset into about 2.3 million comments train, 280 thousand comments dev, and 280 thousand comments test. Figure 1 shows some distributions for the train comments (the dev comments are distributed similarly). As can be seen from the left chart of Figure 1, most comments have 0 descendant comments, followed by 1-2 descendant comments, etc. Thus, it is rare for comments to have high number of descendant comments. From the right chart of Figure 1, we can see that over 98% of the comments have length (in number of words, including punctuation) at most 250, and over 99% of comments have length at most 350. Thus, most comments are not very long, which allows us to make our models more time efficient by truncating the (negligible amount of) very long comments.

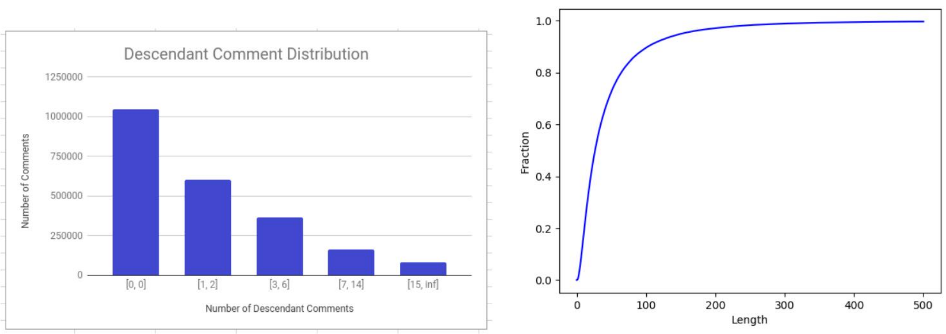


Figure 1: Distributions for the train dataset. Left: The number of descendant comments each comment has divided into 5 buckets. Right: The cumulative distribution of the length of the comments.

We then formulate the problem as a binary classification problem in which given a comment, output the label 0 if the comment has no descendant comments, and output the label 1 if the comment has at least 1 descendant comment. With this formulation, 53.83% of the dev comments have at least 1 descendant comment - this is used for our **naive baseline**, in which we just predict the majority class.

3.2 Approach

We then have the following models in working on this task:

- A **logistic baseline model** consisting of logistic regression on unigram and bigram features from the data.
- A **human oracle** in which we tasked 6 humans to predict whether or not comments will have descendant comments. In order to ensure fair advantage compared to the other models, each comment was annotated with is parent comment as well as its response time before being presented to the human.
- An **LSTM model** [2] [4] using pretrained 300 dimensional GloVe word embeddings [5] on Common Crawl to embed the comments. A self-attention layer was added on top of the LSTM outputs in order learn to weigh important words in comments more, modified from [6]. The outputs of this layer are concatenated with additional features (the response time and the length of the comment) and input into another layer (termed layer 2). The result of layer 2 is finally inputted into a layer that fed into a softmax function to produce the final labels. The optimizer used was the Adam Optimizer [7] and Dropout regularization [8] was added between fully connected layers.

- A **CNN model** [9] with a similar architecture to the LSTM model with the following differences: The LSTM layer was replaced with a CNN layer that feeds outputs from a set of filters into a max pool layer. Positive and negative sentiment intensity scores from the VADER sentiment analyzer applied to the comment itself and its parent comment were concatenated with the other additional features to be input into layer 2 (these were empirically found to have a negligible if not negative impact on the LSTM model). No attention layer was applied.

3.3 Experiments

3.3.1 Comparing Models

We first compare different models and their performance on a subset of the comments in order to find the best model to proceed with hyperparameter search. The results are reproduced in Table 1. To ensure fairness, all models were run on 200,000 train and 40,000 dev comments (except the human oracle), and the iteration with the best dev accuracy was reported, along with its corresponding train accuracy. Additional configurations for the LSTM w/ Attention are 128 LSTM units, 32 attention units, 32 layer 2 units, empirically annealed learning rates, and a Dropout keep probability of 0.9. Additional configurations for the CNN are filter (window) sizes of [1, 2, 3], 100 filters per filter size, 64 layer 2 units, empirically annealed learning rates, and a Dropout keep probability of 0.9. In both we use minibatches of size 64. In Figure 2, we present the learning curves for the LSTM w/ Attention model and the CNN model.

Model	Train Accuracy	Dev Accuracy
Naive Baseline	-	0.5383
Logistic Baseline	0.6715	0.5780
Human Oracle	-	0.625
LSTM w/o Attention	0.6347	0.6439
LSTM w/ Attention	0.6469	0.6492
CNN	0.6382	0.6504

Table 1: Performance of Various Models

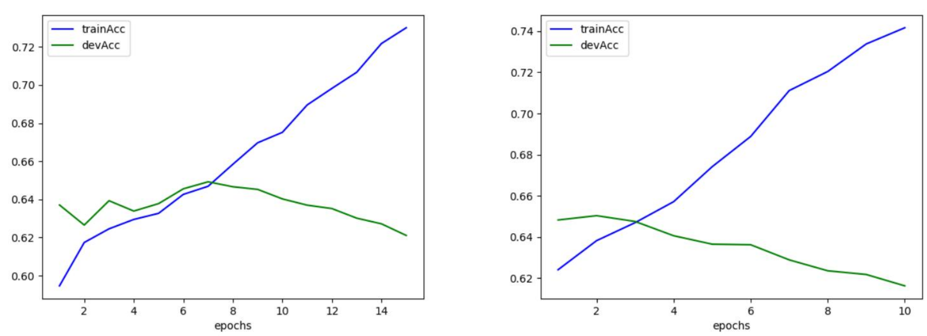


Figure 2: Learning curves. Left: LSTM w/ Attention. Right: CNN.

Observations:

- Table 1 shows that the Logistic Baseline overfits, but the best dev accuracy was achieved while overfitting. More training data (1 million comments) evaluated on less dev data (10,000 comments) has been shown to increase dev accuracy to a max of 0.5871, and while this demonstrates that more data will help, is unfair to compare against the other models in the table due to the different train/dev dataset sizes.

- As the Table 1 and Figure 2 demonstrates, for the LSTM and CNN models, the best dev accuracy was achieved early on when the train accuracy is quite close to the dev accuracy, before the model overfits and dev accuracy drops. Lower dropout keep rates were empirically tested to increase the amount of epochs needed to overfit, but obtain less optimal dev accuracies. Here, the LSTM model performed better with Attention, and the CNN model performed the best. The performance difference is not huge between the CNN and LSTM models, however, the CNN model trains far faster (more than 10x faster) and thus was chosen for Hyperparameter Search.
- While the CNN model was chosen, LSTM with Attention provides more intuition about how important a word may be in determining whether a comment gets a response. The most common heavily weighted token was "?" among those that the model predicted correctly have response comments, which makes sense since questions tend to elicit responses. On the other hand, not all questions elicit responses, as "?" was also the most common heavily weighted token among those that the model predicted incorrectly to have response comments.
- We chose to compare the models based off of accuracy (number of correct predictions) rather than F1 score because of the difficulty of predicting descendant comments. Just because a comment is likely to provoke a response does not mean Redditors will respond. The average human accuracy only beat the simpler models (Table 1), and the best result a single human achieved was only 70% accuracy. As a comparison, the Naive Baseline with an accuracy of 53.83% would obtain an unenlighteningly high F1 score of 70.13%. In addition, the sizes of the two classes are relatively balanced so accuracy is not a bad measure.
- In terms of additional features, the response time was the most useful accounting for a 3-4% improvement, while the VADER sentiment features only accounted for 0.1% improvement in CNN performance at most (and did not help the LSTM), indicating that the models captured most of the correlation between VADER sentiment features and response comments.

3.3.2 Hyperparameter Search

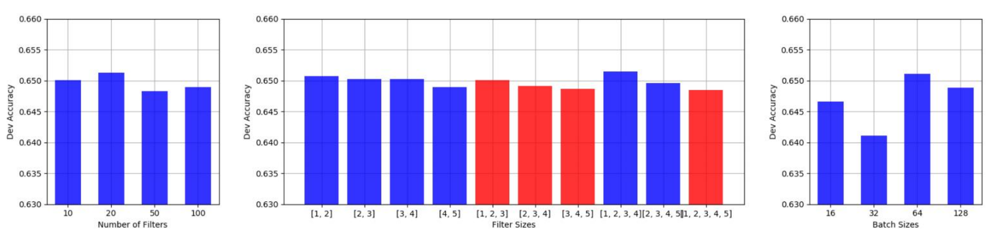


Figure 3: Hyperparameter Search. Left: Dev Performance on different number of filters. Center: Dev Performance on different filter sizes. Right: Dev Performance on different minibatch sizes.

Hyperparameter Search was done via grid search on the number of filters per filter size, the filter sizes, and the minibatch size. Each hyperparameter was individually increased until dev accuracy dropped. The parameters chosen for the final model were 20 filters per filter size, filter sizes of [1, 2, 3, 4], and minibatch sizes of 64.

3.3.3 Final Model and Analysis

With the given hyperparams on the full 2 million comment training set, our CNN model obtained a best dev accuracy of 0.6515 on the full dev set (at a training accuracy of 0.6502), which is comparable to our performance on a subset of the dataset. The Learning Curve in Figure 4 demonstrates that we no longer drastically overfit the training data, which attests to the benefits of more data, however enough epochs were run to demonstrate that the dev accuracy has maxed out. The same model obtained a test accuracy of 0.6499, which is only slightly lower than our dev accuracy. For completeness, the confusion matrix is displayed in Figure 4, the Test precision is 0.6605, and recall

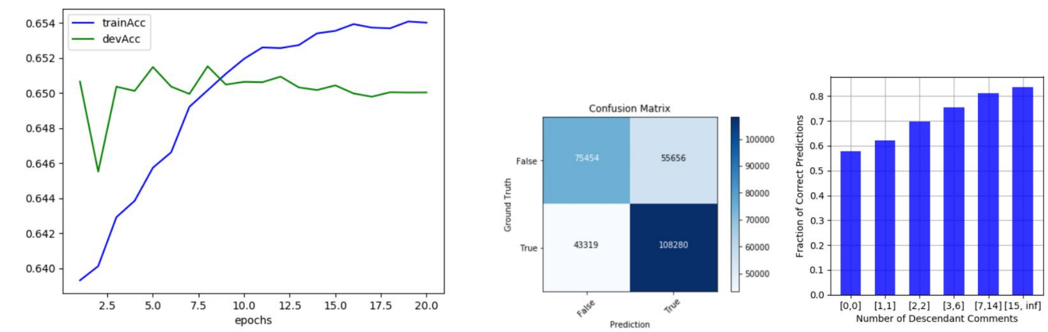


Figure 4: Left: Learning curve for CNN on full dataset for Train and Dev. Center: Confusion Matrix as evaluated on Test where false refers to label 0 and true refers to label 1. Right: Chart of what fraction of comments were predicted to be of the correct class in Dev dataset bucketed by the number of descendant comments

is 0.7143. The predictions are somewhat skewed towards predicting a label of 1 (as noted by the higher recall and lower precision), which can be partially explained by the fact that it is the majority class.

The last chart in Figure 4 demonstrates that even on Dev, our worst accuracy comes from false positives. This reinforces the notion that even if a comment provokes a response, no one may respond due to reasons such as low visibility of the comment or that people may be bored or tired of responding to certain provocations. Meanwhile, the more descendant comments a comment has, the more likely we are to (correctly) give it the label 1, with about 83% accuracy in labeling comments with 15 or more descendant comments - a positive sign that we captured some patterns for popular comments.

Of the comments with label 1 that we predicted correctly, the following is one of the top 5 comments that we predicted correctly (and that is short enough to reproduce here). It is characterized by the high presence of "?" and mention of "lawsuits":

```
Oh like the dozen that have already been published? The ones
that were used in successful lawsuits of the Russian government
in the last 5 years? Cablegate? HBGary? Those?
```

Of the comments with label 1 that we predicted incorrectly, the following is the comment that we thought we most likely predicted correctly. This comment is difficult to understand and seems to require some sort of background knowledge to know what "the pun" is referring to, and is hard for even a human to predict whether or not it would get a response.

```
It didn't take *long* for the *pigs* to get to the scene.\n\n
^normally ^I ^wouldn't ^call ^police ^pigs ^it's ^for ^the ^pun
```

Of the comments with label 0 that we predicted correctly, the following is the one we're most confident in. It is characterized by being short and only using words that tend to be in an aside:

```
Only the finest marihuanas
```

Of the comments with label 0 that we predicted incorrectly, the following is the comment that we though we most likely predicted correctly. Perhaps our model was confused by the many unknown words in the comment, despite the comment not directly asking for a response:

```
As an anatomy major I don't like it because it's the same acronym
that I use for the forearm muscle Extensor Digiti Minimi.
```

Finally, more enlightening may be attention weighted sentences from the LSTM (despite it not being chosen for the final model), which demonstrate some of the words that the model found helpful in

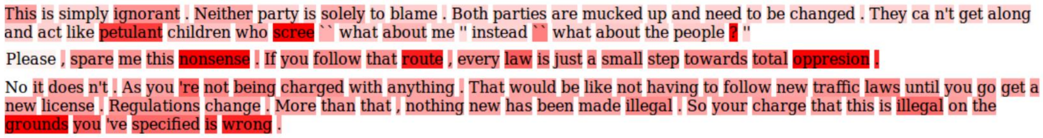


Figure 5: Visualizations from LSTM w/ Attention

predicting whether or not a comment had descendant comments. For example, in the first sentence of Figure 5, which was correctly predicted to have descendant comments (probably from being directly confrontational comments), medium sophistication words of strong emotion tend to have higher weights, such as “ignorant” and “petulant”, as well as the “?” as discussed previously (even though it was not used in this sentence to ask a question of other Redditors). The second comment is similar, but with strong nouns like “nonsense” and “oppresion” highlighted more (even though “oppresion” is spelled wrong, it appeared a few times in our training set as a common misspelling). Finally, to illustrate how social media can be unpredictable, the third comment is a clearly confrontational one that unlike most received no descendant comments, and which we predicted to have label 1 having highlighted words such as “illegal” and “wrong”.

4 Generation

4.1 Dataset

We used the same Reddit comments dataset as for our Prediction model, but processed it differently. We ran through every comment from 2005 December to 2017 March, and added comments from top 20 subreddits that qualify for one of our two criteria features (number of descendant comments is above 20 or score point is above 1000) to respective files (year-month-subreddit-feature.json).

The total processed dataset for text generation include 17,941,413 comments with a vocabulary of 1,544,449 words. The top 20 most frequent words are ’the’, ’to’, ’a’, ’I’, ’and’, ’of’, ’it’, ’that’, ’in’, ’is’, ’you’, ’s’, ’t’, ’on’, ’for’, ’was’, ’with’, ’have’, ’are’, ’be’. If we exclude stop words, top 20 most frequent words are ’like’, ’one’, ’people’, ’beer’, ’bottles’, ’time’, ’http’, ’think’, ’wall’, ’know’, ’com’, ’around’, ’really’, ’great’, ’much’, ’go’, ’good’, ’never’, ’still’, ’back’.

4.2 Approach

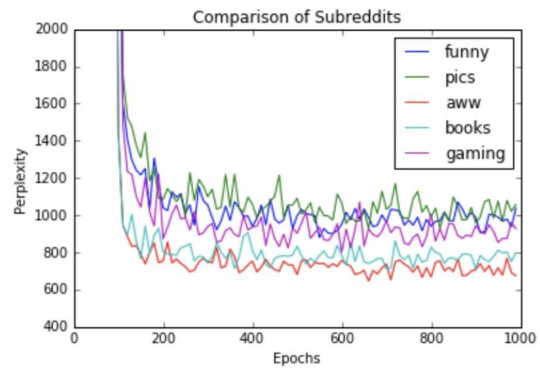
Text Generation part of our project uses a recurrent neural network (RNN) with long-short-term-memory (LSTM) architecture to learn a language model based on the inputted text, and generates a post by predicting the next token at each word. More concretely, it learns a probability distribution of each word in the vocabulary that can come after each word by minimizing cross entropy loss during training. Then it randomly chooses a start word from a set of start words it has seen from the examples, then recursively generates the token at step (t+1) by randomly choosing a token from the vocabulary based on the probability each token can come after the token at step (t) which the language model has learnt during the training. We built our software after forking off of an existing implementation of RNN Text Generation using TensorFlow on Github [10].

We investigated the impact of different parameters and features. First of all, we taught the model comments from six different subreddits such as r/funny and r/news to study how comments differ among online communities. Then, we taught the model the comments that have been chosen for high numChildren, high upvote score, and high controversiality (upvotes + downvotes) to study whether the three indicators of engagement are produced by similar comments or not. Then, we ran multiple iterations of LSTM with different hyper-parameters, namely altering size of each batch, altering number of total batches, number of time steps to stop after in RNN, number of layers, and learning rate. The results are below.

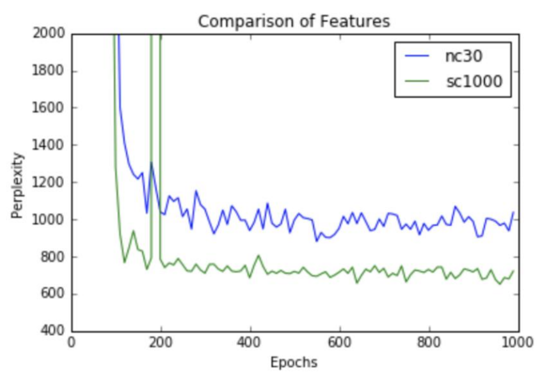
4.3 Experiments

4.3.1 Quantitative Analysis

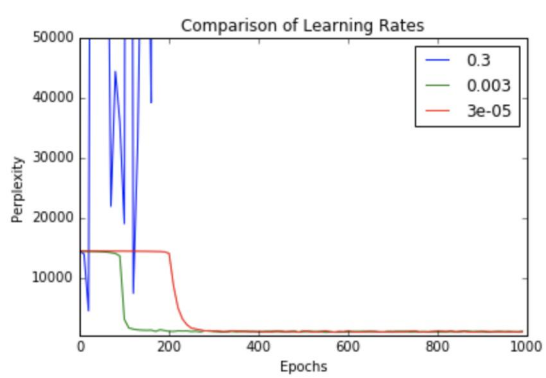
For quantitative analysis of text generation, we run training while varying target subreddits, target features, number of epochs, number of layers, and learning rate. Then we compute perplexity score of our model in training step, and compare them to understand how different factors affect our model's performance.



Some subreddits are easier to learn than others. Relative difficulty across subreddits stayed consistent through the different epochs, and r/aww had lower perplexity than r/pics at every epoch.



We also found that achieving a lower perplexity score on comments that have received scores of over 1000 was easier than on comments that have 30 or more descend comments.



Learning rate was also an impactful factor to training of the model. We found that learning rate of 0.3 was too big for the model to converge. Learning rates 0.003 and 0.00003 both converged to a similar perplexity score, but 0.003 achieved the convergence in about half the number of epochs. Therefore, 0.003 was discovered to be the best learning rate.

4.3.2 Qualitative Analysis

Unfortunately, the model did not produce comments that were grammatically correct or semantically meaningful. One example is the following:

- r/news: Price you need you were taken down black over being black.
- r/todayilearned: Remember one reasoned already massive gone. Woman worries to all voyage soup. We deal guys dead computing. Apparently keep time on it month baby cult there dangers.
- r/pics: Not solid shadows. Free memes play seat to amsterdam event. Somehow lemon lightsaber where best used before that is but amateur. So pick employee in women.

We found that the language model is hard to learn, because the vocabulary size is large on Reddit which tolerates typos, both intentional and not. For example, vocabulary included nice, nieeeeeeeee, and niceeeeeeeeeeeeeee. The context they are used in and the semantic meaning they convey are not significantly different, but the model understood them to be completely separate words, and could not generalize the learning from nice to predicting nieeeeeeeee.

5 Conclusion

5.1 Future Work for Comment Prediction

In analyzing features for comment predictions we tried various ways to include the parent comment, including passing the comment through a separate LSTM before concatenating, and using the parent comment to weigh the comment's words through an attention mechanism. None of these were too effective, and intuition dictates that there should be some interaction between a parent comment and a response that may make people respond to the response, and future work would be to find a better way of incorporating the parent comment (beyond using VADER sentiment features from it).

5.2 Future Work for Comment Generation

We planned to, but did not get the time to, build a sequence-to-sequence model that takes a parent comment and generates a child comment. This would have provided the advantage of producing comments that are topically related to the parent comment, thereby being more engaged in the conversation and inducing more future engagements.

We could try updating the weights of a word when the model sees a misspelling of it. For example, the model would update the weight for ”nice” when it sees ”nieeeeeeeee”. However, identifying the correct spelling of a word is not a trivial task, and many misspellings on the community, such as ”doge”, has a specific meaning that differs from its correct spelling, such as ”dog”.

Another path worth exploring is using pre-trained word vectors as a beginning point, instead of training word vectors from zero. There are only so many comments on Reddit that have 30 or more child comments, or 1000 or more score points, so the corpus was too small for the model to gather a robust understanding of language.

References

[1] Q. Zhang, Y. Gong, J. Wu, H. Huang, and X. Huang, “Retweet prediction with attention-based deep neural network,” *CIKM*, 2016.

[2] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, 1997.

[3] C. Hutto and E. Gilbert, “Vader: A parsimonious rule-based model for sentiment analysis of social media text,” *ICWSM-14*, 2014.

[4] A. Deshpande, “Perform sentiment analysis with lstms, using tensorflow.” `https://github.com/adeshpande3/LSTM-Sentiment-Analysis`, 2017.

[5] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[6] I. Ivanov, "Tf rnn attention." https://github.com/ilivans/tf-rnn-attention, 2017.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv*, vol. arXiv:1412.6980, 2014.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[9] D. Britz, "Implementing a cnn for text classification in tensorflow." http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/, 2015.

[10] Spiglerg, "Rnn text generation tensorflow." https://github.com/spiglerg/RNN_Text_Generation_Tensorflow/graphs/contributors, 2017.