Identifying Russian Trolls on Reddit with Deep Learning and BERT Word Embeddings

Henry Weller
Department of Mechanical Engineering
Stanford University
hweller@stanford.edu

Jeffrey Woo
Department of Computer Science
Stanford University
jbwoo@stanford.edu

Abstract

Spreading misinformation, internet trolls seek to create chaos on popular channels of social media communication. We propose a deep learning solution to performing Reddit troll detection on a comment level basis. We have chosen comment level detection because account-level approaches require increased amounts of user data, whereas comments can be analyzed using Natural Language Processing techniques. We propose a three layer neural network architecture that leverages transfer learning through BERT word embeddings to successfully detect trolls with an extremely limited supervised dataset. We show that our best model containing a Recurrent Convolutional Neural Network (RCNN)(1) outperforms current machine learning practices with an AUC of 84.6% on our test set.

1 Introduction

In recent years, troll accounts have become prevalent on numerous social media platforms[1]. Russian trolls in particular are known for potentially influencing American elections and destabilizing international affairs(2), and they remain at large on Reddit(3). Given that these Russian trolls often operate under singular organizations, it raises two questions: do these trolls operate in similar ways, and can these patterns be identified? In today's changing political climate, developing better methods for troll detection is imperative. Social media has a tremendous influence on how people get their information(4), and truthful discourse will become increasingly difficult with corrupt interference.

While troll and bot detection is being explored on other high profiles platforms such as Twitter with deep learning(5)(6), there is extremely limited work on troll detection applications on Reddit. Furthermore, there appears to be no applications of deep learning on Reddit comments, despite Reddit being one of the most popular sites for information in the world(7). Comment level troll detection offers a way to detect signals within the comment alone that indicate troll-like behavior without any noise from user account data or increased overhead in data processing. We focus our paper on this type of detection for the above benefits and because NLP techniques can be applied to comments.

In this paper, we present our work to apply deep learning to Reddit in an exploration of its viability in comment level troll detection with a limited supervised dataset. We circumvent our limited data by leveraging transfer learning through BERT (Bidirectional Encoder Representations from Transformers)(8) to build better word representations that our dataset could not generate normally. We compare our best model, the RCNN, to two baselines: a non-deep learning approach to comment level troll detection by Punturo (9), and our basic one-layer neural network. With our best model achieving an AUC of 84.6%, we outperform our two baselines of 74% and 66% respectively, indicating a promising potential for deep learning applications in this field of detection.

[1]The definition of a troll is nuanced on the internet, and often used interchangeably in existing literature. We focus on troll detection from a high level standpoint due to limited data.

2 Background and Related Work

Troll and bot detection has already been explored in other platforms such as Twitter. Griffin and Bickel discuss unsupervised methods to gather Russian troll tweets(10) for sufficient dataset creation. Additionally, a student paper from CS230 at Stanford University explores Tweet classification of trolls with LSTMs that achieved 95.07% accuracy(11). Finally, Kudugunta and Ferrara built a state of the art model that performs Twitter bot detection on a comment level with a fairly simple LSTM model, achieving over 96% AUC(6). These findings led us to believe that usage of an LSTM on comment level detection would be an excellent starting point for our model analysis. Ultimately, the breadth of exploration into troll and bot detection on Twitter suggests that there is potential to explore this challenge in other social information mediums.

Although Reddit has far fewer labeled troll datasets compared to Twitter, there is growing concern on this issue, and Reddit has increased their efforts to address it. In Reddit's 2017 transparency report, the CEO released the data of over 900 Russian troll accounts that posted over 7000 total comments, yielding an extremely limited, but viable supervised dataset(12). Currently, the only existing related work on comment level detection is a Medium project by Brandon Punturo in which he utilized optimized random forest classifiers on this dataset to yield an AUC of about 74%(9). We use this attempt as inspiration that our challenge is feasible and as a comparative baseline for our paper.

While limited datasets are not ideal, there has been considerable research in navigating this challenge through transfer learning. Applications of transfer learning achieve significant success by leveraging pretrained models to reduce the total amount of supervised data required to train neural networks(13). BERT (Bidirectional Encoder Representations from Transformers) from Google is a type of pretrained contextual word embedding model that can be utilized when there is not enough labeled data to identify word embeddings(8).Using BERT, we now have the capacity to perform deep learning on our limited dataset to generate classification results.

3 Approach

3.1 Non-Deep Learning Baseline

"Predicting Russian Trolls Using Reddit Comments"(9) is a Medium post that features an Optimized Random Forest Classifier on the same dataset we used to determine trolls from non-trolls. They managed to achieve 74% AUC on their test set.

3.2 Neural Baseline

For our neural baseline, we implemented a one-layer neural network with ReLU activation. The pooled outputs of BERT are sent through a shallow classification layer after performing ReLU activation and dropout to introduce non-linearity and reduce overfitting on the training data respectively. We pool the outputs of our BERT model to create a representation that fits our linear layer.

$$comment = [a_1, a_2, ..., a_c] \in \mathbb{R}^c$$

Reddit comments are represented as vectors of words that is padded to be 'c' words long.

$$x = BERT_{pooled}(comment) \in \mathbb{R}^E$$
$$x = Dropout(ReLU(x)) \in \mathbb{R}^E$$

'x' is the intermediate output with dimension BERT embedding size 'E' that will be sent through the binary classification layer with bias.

$$logits = xW + b \in \mathbb{R}^2$$
$$\hat{y} = Softmax(logits) \in \mathbb{R}^2$$

We use these scores to compute the Cross Entropy Loss to backpropogate through the model.

3.3 Neural Network Architecture Overview

For the remainder of our models, the neural architecture consists of four primary components; the BERT Pretrained Embeddings Layer(14)(15), a neural "middle layer", a dropout layer, and a classification layer as displayed in Figure 1. Each Reddit comment from the dataset is first run through a BERT layer to create an embedding representation of the sentence.

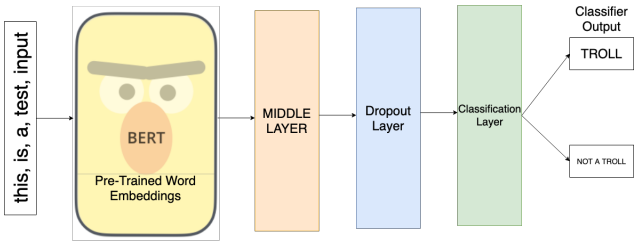


Figure 1: High Level Overview of Neural Architecture

This is run through various different middle layers such as an LSTM. After performing dropout, we use a classification layer that outputs logits that are fed into a softmax function to perform binary classification; whether the comment was written by a troll or not.

Our implementation of BERT is taken from the huggingface repository(14). The remainder of our model draws inspiration from what we have seen in various CS224N assignments and research on each model. However, our architecture design decisions, custom pytorch neural net modules, and modifications are our own. Finally, our run.py file that trains or tests our code draws inspiration from assignments 3 and 5 from this class. We used these files as starting points, modified them to fit our specific problem, and implemented our own evaluation metric.

3.4 Recurrent Neural Network

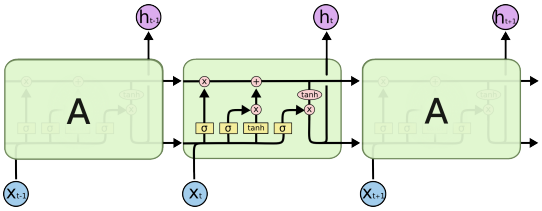


Figure 2: Repeating Module in an LSTM (16)

We implemented an LSTM RNN as the middle layer of our neural architecture. LSTMs excel at analyzing the relationships between the words in the sentence in sequential order(17) and LSTMs circumvent the vanishing gradient problem that makes loss computations difficult with regular RNNs(18). We chose this model as we are evaluating comments, which have an inherent sequential structure.

We compute the BERT embeddings representation of the comment, producing an output of comment length 'c' by embeddings size 'E'.

$$x_{bert} = BERT(comment) \in \mathbb{R}^{c \times E}$$

This is passed through the LSTM layer to produce hidden states 'h' and cell states 'c', each with dimensions of hidden size.

$$h_t, c_t = LSTM(x_{bert}), h_t \in \mathbb{R}^h, c_t \in \mathbb{R}^h$$

We perform dropout on the final hidden state, h_n and then use this as the input for the classification layer. The resulting logits are used to compute Cross Entropy Loss.

3.5 Convolutional Neural Network

Although LSTMs are effective at capturing sequential sentence information, they are known for not being as strong at classification problems because they tend to capture too much of information from the final state(19). We decided to implement a CNN for its classification advantages.

We used a 1-Dimensional Convolutional Layer which has two hyperparameters. The kernel size 'k' determines the windows size used to compute features over comments of length 'c', and the filter size 'f' is similar to the hidden size from our LSTM layer in that it defines the number of output features.

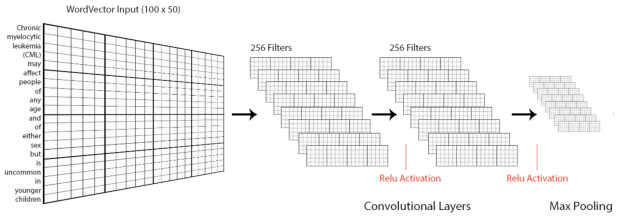


Figure 3: An Example of a CNN(20)

$$x_{conv} = Conv1D(x_{bert}) \in \mathbb{R}^{f \times (c-k+1)}$$

These convolutions represent values for k-grams of the comment. We then apply a ReLU activation function and max pooling to get the most significant signals from each convolution.

$$x_{conv-out} = MaxPool(ReLU(x_{conv})) \in \mathbb{R}^{f}$$

Just like the LSTM layer, we feed this through our dropout and classification layer to get the logits.

3.6 Recurrent Convolutional Neural Network

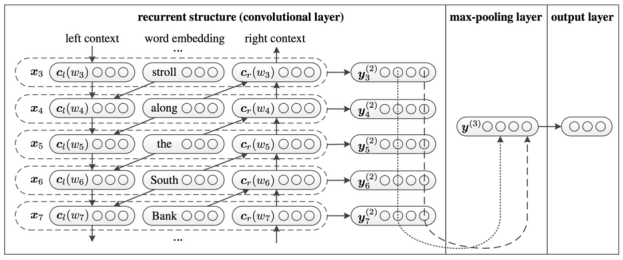


Figure 4: RCNN Architecture(1)

While both LSTMs and CNNs have their merits for sentence level text classification, both have drawbacks. LSTMs tend to be weaker at classification, and CNNs have difficulty in optimizing window size(1). While exploring past 224N projects for inspiration on better text classification models, we discovered an application of Recurrent Convolutional Networks (RCNNs) on subreddit classification, and decided to perform our own research on its efficacy(21).

The RCNN, proposed by Lai, Xu, Liu, and Zhao in 2015 seeks to "learn more contextual information than [CNNs] and represent the semantic of texts more precisely for text classification"[2](1). This model aims to utilize advantages from both RNNs and CNNs to excel even further at text classification.

The RCNN model first performs a pseudo-convolutional layer using a bidirectional LSTM to create an updated representation of each word x_i consisting of its left context (c_l), the BERT embedding, and the right context (c_r).

$$x_i = [c_l(w_i); e(w_i); c_r(w_i)] \in \mathbb{R}^{E+2h}$$

'E' is the BERT embedding size and 'h' is hidden size of the left or right context. We process this representation with a linear layer and tanh activation.

$$x_{intermediate_i} = tanh(Wx_i + b) \in \mathbb{R}^{h}$$

Once we have the representations of every word in the comment, we perform max pooling to capture the most important signals from the sentence and convert this comment into a fixed-length vector.

$$x_{rcnn-out} = MaxPool(x_{intermediate}) \in \mathbb{R}^{h}$$

[2]Quotation can be found on page 2268 of their paper.

Finally, we run the output through our dropout and classification layer to get the logits. Our implementation of the RCNN model was started on our own. We then drew inspiration from another implementation of an RCNN(22), but disagreed with some of their design decisions.

4 Experiments

4.1 Data and Evaluation Metrics

We evaluated our model using two datasets that were created by Brandon Puntoro(23). One dataset consists of approximately 7,000 comments written by Russian Troll users that were banned in 2017(12). The other is a Google BigQuery dataset of normal Reddit user comments from 2015 to 2018. It is worth noting that these BigQuery comments are random samples, and we are assuming that none of them are actual trolls. With more time, we would seek to surpass this limitation and curate a more rigorous dataset. We randomly selected about 7000 comments from the BigQuery dataset as the non-troll dataset. We then merged and shuffled the datasets, creating an 80-20 split of train/validation to test data, and a further 80-20 split of the train/validation data respectively. This led to 64% of the data being used for training, 16% for validation, and 20% for testing.

The metric we used to classify our model's success is our classifier's Area Under the Curve - Receiver Operating Characteristics (AUC-ROC). AUC is a consistent metric that circumvents the situation where accuracy can be misleading with unbalanced datasets(24). Furthermore, AUC illuminates our model's confidence in its classifications, and it is a commonly utilized metric in other text classification papers(25). We will also measure model accuracy, but will not prioritize it for optimization.

4.2 Initial Experimental Details

Our model is given "vanilla" parameters to generate initial results:

	Value
BERT Embedding Size	768
Hidden Size	256
Max Comment Length	40
Dropout	0.2

Table 1: "Vanilla" parameters to generate initial results

Stochastic gradient descent updates are extended using an Adam Optimizer with an initial learning rate of .001. Our model runs for a maximum of 30 epochs, but we utilize an early stopping mechanism to reduce overfitting by occasionally reverting training back to the best model seen so far. Finally, during training, we uniformly initialize every parameter that is not associated with BERT.

On an Azure GPU, our model takes approximately 30 to 60 minutes to train and less than a minute to test because our training set has less than 10,000 entries. Every thirty batch iterations during training, we validate the model with the development data. Finally, our model is seeded so that results are reproducible.

4.3 Optimization

After testing each model with "vanilla" parameters, we achieved promising results from both the CNN and RCNN, each producing AUC scores on the test set of roughly 83%. We selected the RCNN as the model to optimize hyperparameters for since it had slightly better performance than the CNN and fewer hyperparameters since beam size was not a consideration.

Rather than performing babysitting optimization(26), we wanted to push ourselves to find the best hyperparameters possible. According to Bengio and Bergsta, Random Search is a highly effective method of optimization(27), and with 60 iterations, there is a 95% probability of having a combination of hyperparameters that are in the top 5% of all hyperparameters(28).

We optimized over hidden size, dropout rate, and comment length. After 60 trials with hyperparameters randomly selected from windows of [64, 512], [0.35, 0.65] and [10,80], respectively, we chose

the models that tied for best dev AUC and ran them on the test set. The best model achieved a test AUC of 0.846 with hyperparameters of 415 (hidden size), 0.445 (dropout), and 70 (comment length).

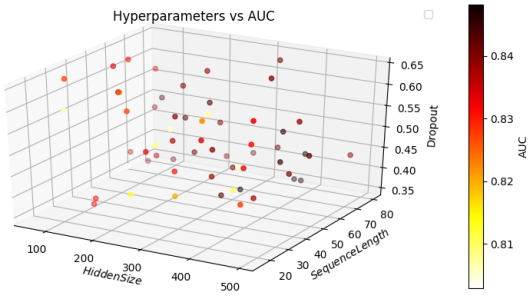


Figure 5: Randomized Hyperparameter Search Results

The comment length optimum seems to be the most intuitive of the three values, since about 93% of our test dataset has 70 words or less (see Figure 8). The dropout optimum of 0.45 also aligns with dropout numbers we have seen in the paper "Understanding Dropout"(29), which suggest that a dropout rate around 0.5 increases variance in the training set distribution and the effectiveness of regularization. Surprisingly, increasing hidden size beyond 400 causes performance to degrade. It demonstrates that increasing the number of features in our hidden state does not always mean greater performance. For other optimization experiments, please see appendix.

4.4 Results

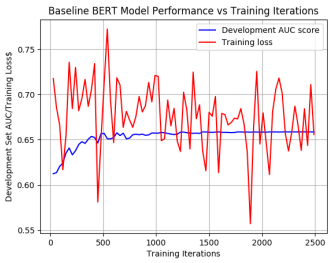
	AUC	Accuracy
Optimized Forest Classifier Baseline	0.74	0.74
BERT Baseline	0.659	0.625
BERT + LSTM	0.805	0.727
Vanilla BERT + CNN	0.826	0.741
Optimized BERT + CNN	0.843	0.756
Vanilla BERT + RCNN	0.836	0.746
Optimized BERT + RCNN	**0.846**	0.749

Table 2: Experiment results showing AUC and accuracy per model classifier

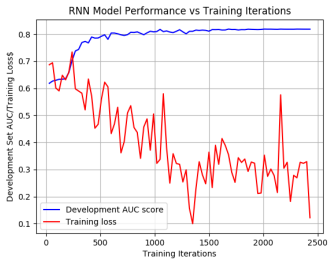
Our results greatly exceeded our expectations. We knew that BERT was a powerful tool for Natural Language Processing, but did not expect that combining it with the CNN and RCNN would handily beat both baselines. We were not surprised that each additional model performed better than the previous. The RCNN sought to optimize the best parts of the LSTM and CNN, so it would make sense that it would perform slightly better.

We are also excited to see how well our CNN performed once optimized. Although we weren't prioritizing accuracy, it was interesting that it scored the best on this metric. This also reaffirms the lectures that CNNs tend to be better at classification than LSTMs. It also shows that "standard" models such as CNNs can perform comparatively well to more complex architectures like RCNNs.

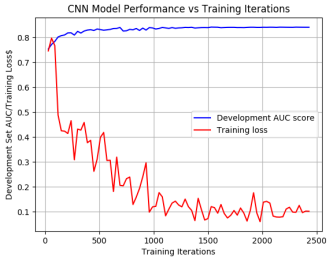
BERT with only a simple classifier was not enough to effectively accomplish troll detection. Unlike the other training curves, the training loss never properly decreases over time for the baseline model, and it reaches a maximum development AUC of 0.659, well below any of the scores of the other models. This suggests that though BERT is a state-of-the-art tool for building contextual word embeddings, classification solely based on BERT transfer learning is insufficient for specific NLP tasks.



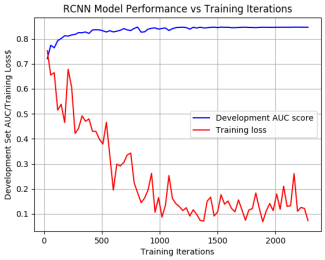
(a) BERT + Classifier



(b) BERT + LSTM + Classifier



(c) BERT + CNN + Classifier



(d) BERT + RCNN + Classifier

Figure 6: Training and Validation Performance of BERT with Different Neural Architectures

4.5 Output Analysis

We examined three failure modes for error analysis on the best performing RCNN model: number of comments posted per troll, number of words per comment, and trigger words within the dataset.

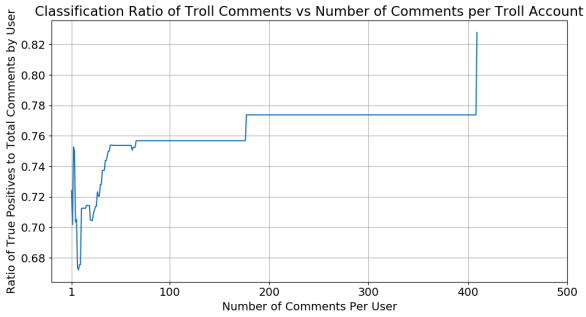
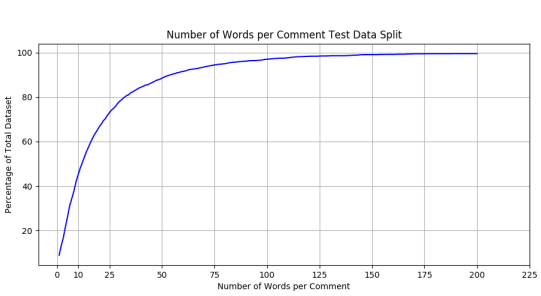


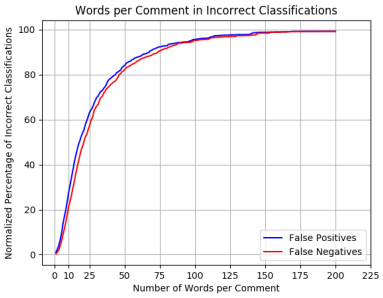
Figure 7: Distribution of Number of Comments Per User in Test Results

To examine diversity of the dataset, we looked at how well our model performs at classifying trolls who comment anywhere from a few times to over 400 times. Figure 7 shows how accuracy generally improves as the number of troll comments per user increases. This suggests that the model's evaluation metric success might be biased towards the users who contribute significantly more to our dataset. However, the model still appears moderately successful at identifying trolls with less comments, so it is beneficial to explore our model's efficacy in more detail.

Figure 8 shows the cumulative distribution of number of words per comment in our test dataset. This figure also shows the false positive and false negative comment length distributions, normalized by the number of total comments with a given word length. Almost 50% of the dataset has 10 words or less, and these account for roughly 20% of the error in our model, suggesting that our model performs better on longer comments. This could be because longer comments have more signals and nuance for the model to interpret. The plot also shows pretty clearly that the model is just as likely to misclassify false positives as it is false negatives based on comment length alone.



(a) Distribution of Test Data



(b) Distribution of Words per Comment in False Positives and Negatives produced by RCNN Model

Figure 8: Distribution of Comment Size in Dataset and False Classifications

Given that our model makes many errors on smaller comments, we took a closer look at the specific classifications our model made to better understand how our model succeeds and fails on certain text samples. Italicized sentences are originals, bold words are changes.

Example	Test Sentence	Prediction	True
1	*Hillary won the popular vote*	1	0
	Sam won the popular vote	0	0
	Cats won the popular vote	1	0
	Hillary **played** the popular vote	0	0
2	*Wow I hope that's not me*	0	0
	Cats Wow I hope that's not me	1	0
3	*Hillary is 68, she just can't be all healthy*	1	1
	Hillary is 68, **he** just can't be all healthy	1	1
	Hillary is 68, she just can't be all **there**	1	1
	Hillary is 68, **he** just can't be all **there**	0	1

Table 3: Example with predictions and gold standards modified to show classification changes.

With smaller comments, our model seems to value "trigger" words in making its prediction. Notably, the model often categorizes comments with "Hillary" or "Cats" as troll comments. We hypothesize these are triggers because Hillary Clinton was a regular target of Russian trolls on social media and cats are mentioned frequently by trolls in the training data. This suggests that our model might be making mistakes on smaller comments because it relies too heavily on triggers when there isn't enough other information to process. However, examples 1 and 3 also show that triggers alone are not enough for a troll classification. When certain context words are changed in sentences with trigger words, the classification can also change. Perhaps the relationship between the trigger and context words is critical towards correctly classification.

5 Conclusion

Our initial concept for this project was to develop a neural network that could detect Russian trolls on Reddit more effectively than conventional machine learning models could. Our findings showed that that the usage of BERT and transfer learning alongside standard LSTM, CNN and non-standard RCNN architectures can beat the baseline by a wide margin.

We recognize two main limitations in our work. First, it is difficult to make strong claims about our findings because of our small dataset. Secondly, our dataset is skewed where many of the troll comments come from a small subset of users, which has introduced some bias into our model. Despite these limitations, we assert that our models offer the best Reddit troll detection capabilities of what currently exist today. As next steps, we recommend that more work is done in proper data collection so that we can test our models again for reproducible and scalable success.

6 Acknowledgements

Custom Project Mentor: Xiaoxue Zhang

References

[1] Lai, S., Xu, L., Liu, K., & Zhao, J. (2015). Recurrent Convolutional Neural Networks for Text Classification. AAAI.

[2] Romano, A. (2018, October 19). Twitter released 9 million tweets from one Russian troll farm. Here's what we learned. Retrieved from https://www.vox.com/2018/10/19/17990946/twitter-russian-trolls-bots-election-tampering

[3] Yeates, W. (2019, February 07). Russian Troll Farms Continue To Have A Presence On Reddit. Retrieved from https://www.valuewalk.com/2019/02/russian-troll-farms-presence-reddit/

[4] Marr, B. (2018, March 21). How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read. Retrieved from https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#3473dafe60ba

[5] Fornacciari, P., Mordonini, M., Poggi, A., Sani, L., & Tomaiuolo, M. (2018). A holistic system for troll detection on Twitter [Abstract]. Computers in Human Behavior, 89, 258-268. doi:10.1016/j.chb.2018.08.008

[6] Kudugunta, S., & Ferrara, E. (2018). Deep neural networks for bot detection. Information Sciences, 467, 312-322. doi:10.1016/j.ins.2018.08.019

[7] Nguyen, C. (2018, May 30). Reddit Is the Third-Most-Popular Destination on the Internet. Retrieved from https://www.digitaltrends.com/computing/reddit-more-popular-than-facebook-in-2018/

[8] Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). A New Pre-Training Method for Training Deep Learning Models with Application to Spoken Language Understanding. CoRR. Retrieved from https://arxiv.org/abs/1810.04805.

[9] Punturo, B. (2019, January 08). Predicting Russian Trolls Using Reddit Comments – Towards Data Science. Retrieved from https://towardsdatascience.com/predicting-russian-trolls-using-reddit-comments-57a707653184.

[10] Griffin, C., & Bickel, B. (2018). Unsupervised Machine Learning of Open Source Russian Twitter Data Reveals Global Scope and Operational Characteristics [Abstract]. CoRR. Retrieved from https://arxiv.org/abs/1810.01466.

[11] Brown, E., Edelson, B., & Taylor, E. (2018). Identifying Tweets Written by Russian Troll Accounts. CS230 2018 Spring. Retrieved from https://cs230.stanford.edu/projects_spring_2018/reports/8289223.pdf

[12] Huffman, S. (2018, April 10). R/announcements - Reddit's 2017 transparency report and suspect account findings. Retrieved from https://www.reddit.com/r/announcements/comments/8bb85p/reddits_2017_transparency_report_and_suspect/.

[13] Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018). A Survey on Deep Transfer Learning [Abstract]. CoRR. Retrieved from https://arxiv.org/abs/1808.01974.

[14] Huggingface. (2018). PyTorch Pretrained BERT: The Big & Extending Repository of pretrained Transformers. Retrieved from https://github.com/huggingface/pytorch-pretrained-BERT.

[15] Alammar, J. (2018). The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning). Retrieved from http://jalammar.github.io/illustrated-bert/

[16] Olah, C. (2015, August 27). Understanding LSTM Networks. Retrieved from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[17] See, A. (2019). Lecture 6: Language Models and Recurrent Neural Networks [Powerpoint Slides]. Retrieved from Stanford University CS224N, delivered 24 Jan 2019.

[18] See, A. (2019). Lecture 7: Vanishing Gradients and Fancy RNNs [Powerpoint Slides]. Retrieved from Stanford University CS224N, delivered 29 Jan 2019.

[19] Manning, C. (2019). Lecture 11: ConvNets for NLP [Powerpoint Slides]. Retrieved from Stanford University CS224N, delivered 12 Feb 2019.

[20] Hughes, M., Li, I., Kotoulas, S., & Suzumura, T. (2017). Medical Text Classification using Convolutional Neural Networks. Student Health Technology Informatics. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/28423791.

[21] Tam, I. (2017). Classifying Reddit comments by subreddit. CS224N 2017 Winter. Retrieved from https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/reports/2735436.pdf

[22] Prakashpandey9. (2018). Text classification using deep learning models in Pytorch. Retrieved from https://github.com/prakashpandey9/Text-Classification-Pytorch/blob/master/models/RCNN.py.

[23] Punturo, B. (2019). Reddit Trolls. Retrieved from https://github.com/brandonjpunturo/Reddit-Trolls.

[24] Aidankmcl, Indico, & Pascallv. (2014). Advantages of AUC vs standard accuracy. Retrieved from https://datascience.stackexchange.com/questions/806/advantages-of-auc-vs-standard-accuracy

[25] Narkhede, S. (2018, June 26). Understanding AUC- ROC Curve – Towards Data Science. Retrieved from https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[26] Gozzoli, A. (2018, September 05). Practical guide to hyperparameter searching in Deep Learning. Retrieved from https://blog.floydhub.com/guide-to-hyperparameters-search-for-deep-learning-models/

[27] Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization [Abstract]. Journal of Machine Learning Research. Retrieved from http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

[28] Zheng, A. (2015). Evaluating Machine Learning Models. Retrieved from https://www.oreilly.com/ideas/evaluating-machine-learning-models/page/5/hyperparameter-tuning

[29] Baldi, P., & Sadowski, P. (2013). Understanding Dropout. NIPS. Retrieved from http://papers.nips.cc/paper/4878-understanding-dropout.pdf

7 Appendix

7.1 CNN Hyperparameter Tuning

We made the assumption that the optimized hyperparameters for RCNN would work well for the CNN model, and performed babysitting tuning for the beam size hyperparameter. This produced a CNN model with an AUC of 0.843 and beam size of 3.

We found it interesting that the optimum point was at beam size of 3. Typically convolutions are more effective at capturing n-gram relationships when the window size is larger. However, 13% of our dataset has are fewer than 3 words, which might cause our performance to taper out when the beam size exceeds 3. This reveals an issue with our dataset in that a preponderance of short comments can make it difficult to build in accurate CNN model with a reasonable beam size of 5 or more.

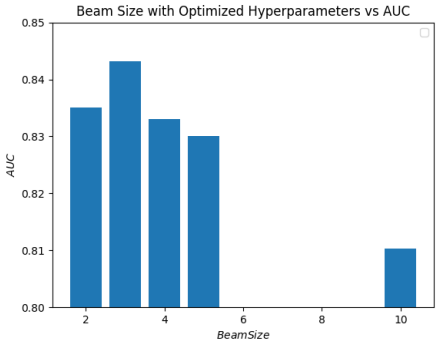


Figure 9: Beam Search Results

7.2 BERT Transfer Learning Basis

We also experimented with changing our transfer learning basis from Bert-base-uncased to Bert-large-uncased to see if increasing to 24 hidden BERT layers and 1024 BERT embeddings improved the model. However, this actually reduced performance; RCNN AUC decreased to 0.815.