Headline Generator

David J. Liedtka
Department of Computer Science
Stanford University
dliedtka@stanford.edu

Dane M. Hankamer
Department of Computer Science
Stanford University
dhank@stanford.edu

Abstract

Headline generation allows a writer to supplement brainstorming with computer-generated examples or a reader to quickly filter through large quantities of unnamed documents to pick those that might spark their interest. We modify an encoder-decoder recurrent neural network approach and leverage article structure to generate headlines from the content of news articles. While our results do not compare favorably to the state-of-the-art, we analyze trends that could contribute toward future work.

1 Introduction

We use a recurrent neural network to generate headlines for news articles, based on the work of Lopyrev.(4) An ideal headline summarizes the most important ideas of an article in a concise way. We aim to leverage a developed understanding of word meanings and a recognition of how articles are constructed to generate the best possible headlines and to compete against previous results. Specifically, the inputs to our algorithm are word tokens from an article. We then use a 3-layer recurrent neural network to output a predicted headline for the article. Because article lengths vary greatly, we do not input the entire article to our algorithm, but rather we input a set of tokens from the beginning of the article, a set from the end, or some combination thereof. For example, in many articles the introductory paragraph (and sometimes the final, conclusion paragraph) serve as an effective one-paragraph summary of the entire article. Keeping this in mind, combined with utilizing Keras support for word embeddings, we attempt to generate sensible headlines.

A successful implementation allows users to generate headlines for large numbers of articles, blog posts, texts, etc. that are without headlines, allowing users to quickly filter articles that they are interested in reading. Alternatively, writers might use a headline generator to spark creative thinking when generating a headline of their own. Additionally, it is easy to imagine that modifying the cost function to reward the headlines of frequently visited articles could lead to generation of more attention grabbing headlines.

2 Related Work

Lopyrev previously used a recurrent neural network to generate headlines for short news articles (using only the first 50 words).(4) The dataset we use contains much longer articles. An implementation of Lopyrev's methodology exists on Github, and we use this as our code base.(1)

Another approach to headline generation was described by Vora, in which a grammar-based extractive summarization system with a deep neural network was compared against Lopyrev's recurrent neural network based headline generation system.(7) Similar to Vora's approach, we use the BLEU evaluation metric, which compares which fraction of n-grams in expected headlines are actually output as well as the length of the generated headline compared to the true headline. It is the first metric to report high correlation with human judgment, which bodes well for our headline generation task and represents an improvement on the Levenshtein score metric employed by our code base.(5)(1) Still, Lopyrev's recurrent neural network approach is considered state-of-the-art and outperformed Vora's extractive summarization system.

In the recent work of Hayashi and Yanagimoto, another clever encoder-decoder approach with a recurrent neural network was presented, where the first sentence of an article was reformulated into a headline.(2) The results are superior to statistical machine translation, but the first sentence often yielded nonsensical headlines, especially in cases where the first sentence was a short attention-grabber. While the first sentence is often important, our work considers up to 50 word tokens, regardless of the length of the first sentence.

3 Dataset and Features

3.1 Dataset

We use the "All the news" dataset published by Andrew Thompson, which contains 187,873 news articles (with headlines) from 18 different American publications.(6) The dataset is available at the following link: https://components.one/datasets/#all-the-news. This dataset is an updated version of one posted on Kaggle, containing over 40,000 more articles from a greater number of publications. We chose to use this updated version because of this greater variety.

As noted by our lengthy python preprocessing script, parsing the dataset was more difficult than expected. The articles are stored as a CSV with comma delimiters, but the formatting of the data is inconsistent. For example, many articles have commas in their titles or in the articles themselves (or in the author field for multiple authors), so we could not simply use a comma delimiter. Fields with commas in them are usually surrounded by quotation marks, so we created an edge case to handle this, but titles and articles can also contain quotation marks. Additionally, randomly placed line breaks often break up examples without a discernable pattern. Because of this, we had to evaluate many samples on a case-by-case basis.

Despite these difficulties, we were able to process 96,543 articles total, which is a large enough dataset for our purposes.

3.2 Embedding

We tokenize all articles and headlines with the intention of creating a 40,000 word, 100-dimensional vocabulary-embedding. There are 1,322,508 unique words present in our dataset. We map the most popular 40,000 words to their GloVe embedding weights. For words outside of this top 40,000, we map them to their closest match within the 40,000 if the cosign similarity of their GloVe vectors is above a threshold of 0.5; otherwise, we map these words to an unknown token.

4 Methods

4.1 Architecture

As mentioned previously, we used a modification of Lopyrev's recurrent neural network architecture, which consists of an encoder and a decoder that each represent a recurrent neural network. (4) The encoder sequentially takes individual words from the text of an article and converts each word into a distributed representation. Next, the distributed representation is combined using a 3-layer neural network (Lopyrev uses four layers, we reduce this with an eye towards computational complexity) with the hidden layers generated after processing the previous word. Once the final word is passed through the encoder, the decoder takes as input the hidden layers created by the encoder. The decoder uses a softmax layer and a special activation layer to form each word of the generated headline. Each generated word is subsequently fed into the decoder as an input for generating the next word of the headline.

Our special activation layer computes attention weights that allow the recurrent neural network to remember certain aspects of the input better such as names and other critical words. After processing each input word, the hidden units of the last layer are split into 2 sets: one set of size 40 used to compute the attention weights, and one set of size 472 passed on to the softmax layer. For each output word, the attention weight is calculated over each of the input words. The attention weight for the input word at position *t*, computed when outputting the *t'*-th word is:

$$a_{y_{t'}}(t) = \frac{\exp(h_{x\,t}^{T} h_{y_{t'}})}{\sum_{\bar{t}}^{T} \exp(h_{x\,\bar{t}}^{T} h_{y_{t'}})}$$

where h_{x_t} represents the final hidden layer generated after processing the *t*-th input word, and $h_{y_{t'}}$ represents the final hidden layer from the current decoding step. The weights sum to 1 and represent a weighted average over the last hidden layers after processing all of the input words. Finally, the weighted average is transferred to the softmax layer along with the last hidden layer from the current decoding step. This key component allows the model to discern which words from the text are most important when generating a new headline.

We use 3 hidden layers of LSTM units with each layer containing 512 hidden units. Because our model is processed word by word, LSTM allows us to handle sequential data and helps avoid vanishing gradients. We also implemented dropout at each layer, although our results are consistent with Lopyrev's observation that dropout does not improve the model's performance. We hypothesize that dropout did not work well for our model because the recurrence in our network amplifies noise, which in turn hurts learning. However, subsequent testing using dropout on certain connections within the recurrent neural network might yield different results.

4.2 Loss Function

We employ the categorical cross entropy loss function:

$$L(\hat{y}, y) = \frac{-1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} 1_{y_i \in C_c} \log p_{model}[y_i \in C_c]$$

where *N* is the number of observations *i*, *c* is each category in *C*, $1_{y_i \in C_c}$ determines whether or not the *i*-th observation belongs to the *c*-th category, and $\log p_{model}[y_i \in C_c]$ is the probability predicted by the model for the *i*-th observation belonging to the *c*-th category. Cross entropy loss is favorable over MSE for our model because the weight adjustment factor during back-propagation will not diminish, thereby ensuring our training does not stagnate.

5 Experiments, Results, and Discussion

5.1 Hyperparameters and Training

We use a learning rate of 0.0001 along with the Adam optimizer. For Adam we use a first moment of 0.9, a second moment of 0.999, and an epsilon of 1e-8. Furthermore, we use a batch size of 64. Training these sequences is expensive, with each iteration taking approximately 50 minutes even on a GPU instance. We train 1 epoch over 114 iterations, which equates to nearly 95 hours (nearly 4 days of training time) on a p2.8xlarge Amazon EC2 instance. Figure 1 plots our loss function versus training iterations.

5.2 Testing

With nearly 96,543 articles processed, we use relatively smaller validation and test sets. We randomly shuffle the articles to ensure the training, validation, and test sets are of the same distribution, and then we choose 93,543 articles for the training set, and 1,500 articles each for the validation and test sets.

Regarding prediction, Lopyrev used the first 50 words of each article as input and capped the maximum headline length at 25 words. While we use the same maximum headline length, we hypothesize that although the introductory paragraph of an article is usually the most useful for a summarization task like headline generation, valuable information can be gained from the conclusion of the article as well. To test this hypothesis, we explore three different types of input: the first 50

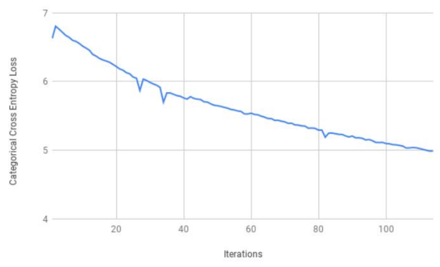


Figure 1: A plot of categorical cross entropy loss versus iterations.

words of each article (the same as Lopyrev), the first 25 and last 25 words, and the first 50 and last 25 words.

In order to quantify our ability to predict headlines, we use two metrics: BLEU score and Levenshtein score. Bilingual Evaluation Understudy (BLEU) score counts matching n-grams in the proposed headline to n-grams in the true headline, and is further defined in Papineni et al.(5) BLEU score is also the metric that Lopyrev used, allowing for a direct comparison of results. The Levenshtein score, or Levenshtein distance, is the minimum number of single-character edits (insertions, deletions or substitutions) required to change the predicted headline into the true headline. It is further defined in Levenshtein.(3) Thus, our objective is to maximize the BLEU score while minimizing the Levenshtein score.

5.3 Results

We obtain the following results:

Table 1: BLEU score of experimental runs. **50** refers to using the first 50 words of an article as input, **25+25** refers to using the first 25 and last 25 words, and **50+25** refers to using the first 50 and last 25 words.

BLEU Score	**Training**	**Validation**	**Testing**
50	0.01365756221	0.01451416069	0.01226555188
25+25	0.01334964053	0.01323200369	0.01297593721
50+25	0.01382171917	0.01177343727	0.01119147869

Table 2: Levenshtein score of experimental runs. The labels refer to the same experiments as in Table 1.

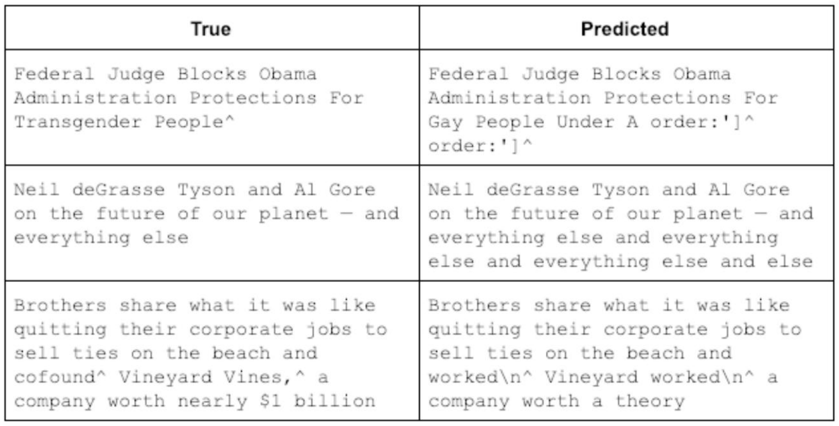
Levenshtein Score	**Training**	**Validation**	**Testing**
50	54.77231648	54.52419509	54.98365409
25+25	54.55219124	54.39320007	54.40124380
50+25	54.92491911	54.19084872	54.42568945

There is no evidence of overfitting, a result of using a large enough dataset and our shuffling of training examples. However, our BLEU score results do not compare favorably to Lopyrev's. We obtain scores similar to what he obtains early in his training, but later Lopyrev is able to eventually obtain an average BLEU score of approximately 0.09.

5.4 Analysis

A selection of predicted headlines compared to their true counterparts is presented in Figure 2.

While our results do not compare to the state of the art, we notice interesting trends that we can seek to capitalize on in our future work. The first of these trends is the difficulty of predicting the <EOS> token. Figure 2 shows two examples where the predicted headline was mostly correct, but then when the headline should have concluded the decoder continued to output miscellaneous words. We performed several small-scale intermediary runs at different stages of training, and we noticed that early headline predictions tended to be 25 words long, which was the maximum allowed headline length. With further training, we suspect our algorithm would continue to improve at outputting the <EOS> token and ending headlines in the right place.



True	Predicted
Federal Judge Blocks Obama Administration Protections For Transgender People^	Federal Judge Blocks Obama Administration Protections For Gay People Under A order:']^ order:']^
Neil deGrasse Tyson and Al Gore on the future of our planet — and everything else	Neil deGrasse Tyson and Al Gore on the future of our planet — and everything else and everything else and everything else and else
Brothers share what it was like quitting their corporate jobs to sell ties on the beach and cofound^ Vineyard Vines,^ a company worth nearly $1 billion	Brothers share what it was like quitting their corporate jobs to sell ties on the beach and worked\n^ Vineyard worked\n^ a company worth a theory

Figure 2: A (cherry-picked) selection of results: true headlines compared to their predicted counterparts.

In addition to outputting the <EOS> token, we hypothesize that our results were negatively affected by insufficient training. Training was computationally expensive, and despite training for about four days our loss had yet to converge. At a certain point, we had to stop training in order to begin running experiments, but it is unclear how much longer training would have continued to decrease loss. The cost of training also hampered our ability to experiment with different architectures and hyperparameters, as we would have had to start training from the beginning again.

Another problem that affected our results was the use of a less than optimal dataset. There is an earlier version of the dataset used on Kaggle, but we decided to use an updated version with more articles and a greater variety of publishers. However, it may have been a better choice to use the Kaggle version in hindsight, as the dataset we used was difficult to preprocess. Additionally, many articles began with text unrelated to the article (such as "please disable your ad-blocker to continue reading", etc.). We had hoped that using text from the end of the article would help combat this, but ideally we would have cleaned our dataset.

While Levenshtein score rewards character-by-character matches, BLEU scores tends to be a more punishing metric, as it rewards context (using n-grams instead, not just 1-grams). Our low BLEU scores indicate that our model struggled to generate words in the right context. We believe this to be a result of insufficient training, and given more time or resources we would expect to see improvement. Similarly, leveraging different parts of the article did not make a significant impact. However, with a better working model, we believe we would see more conclusive results about the utility of using article structure.

6 Conclusions and Future Work

Although our performance fell well short of the state of the art, we identified areas to improve on that we believe could lead to more meaningful results. In the future, we hope to further train our models, prepare a cleaner dataset, and experiment with different hyperparameters in order to gain better insights into the headline generation problem.

7 Acknowledgements

We would like to thank Andrew Ng, Kian Katanforoosh, and the entire CS 230 team for putting together a rewarding and worthwhile class, and we would like to thank our project mentor, Weini Yu, for her invaluable feedback.

8 Contributions

David Liedtka pre-processed the dataset and developed the testing script. Dane Hankamer conducted the majority of the literature review, modified the vocabulary-embedding, training, and predict scripts, and migrated the scripts onto AWS. Both David and Dane collaborated on the direction of the product, the intermediate and final reports, and the poster.

Our code can be found at the following GitHub link: https://github.com/dliedtka/cs230.

References

[1] BEN-REUVEN, E. Automatically generate headlines to short articles. `https://github.com/udibr/headlines`, 2018.

[2] HAYASHI, Y., AND YANAGIMOTO, H. Headline generation with recurrent neural network. In *New Trends in E-service and Smart Computing*. Springer, 2018, pp. 81–96.

[3] LEVENSHTEIN, V. Leveinshtein distance, 1965.

[4] LOPYREV, K. Generating news headlines with recurrent neural networks. *arXiv preprint arXiv:1512.01712* (2015).

[5] PAPINENI, K., ROUKOS, S., WARD, T., AND ZHU, W.-J. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics* (2002), Association for Computational Linguistics, pp. 311–318.

[6] THOMPSON, A. All the news. `https://components.one/datasets/#all-the-news`, 2019.

[7] VORA, D. Headline generation using deep neural networks.