Abstractive Sentence Summarization with Attentive Deep Recurrent Neural Networks

Alex Alifimoff
aja2015@cs.stanford.edu

Author's note: I liberally use plural pronouns. Most of my projects are group projects so it is now naturally my writing style. Sadly, I was the only one who worked on this project.

Introduction

Summarization is the task of taking an input text and creating a supplementary text which contains the same meaning as the original text, but in less words. In Natural Language Processing, there are three primary approachs to summarization. The first is reducing the length of the input text by simply deleting words in the original text, but preserving word order. This is known as deletion or compressive summarization. The second type of summarization is called extractive, and refers to generating a summary using words in the input text only, but without regard to word order. Finally, the third and most general type of summarization is abstractive, which produces a summary using any words. The output in abstractive summarization is not at all constrained by the input text. Perhaps obviously, building a good abstractive system is harder than building summarization systems in either of the other two regimes.

We can formally define the task. Summarization formally consists of an input string $\boldsymbol{x} \in \mathscr{X}$, where $\mathscr{X} \subset (\{0,1\}^V, \{0,1\}^V, ...\{0,1\}^V)$ where V is the size of our vocabulary. An abstractive system finds a second string $\boldsymbol{y} \in \mathscr{X}$ such that

$$\boldsymbol{y} = \arg\max_{\boldsymbol{y} \in \mathscr{X}} s(\boldsymbol{x}, \boldsymbol{y}) \tag{1}$$

An extractive/compressive system finds a summary using only words that are extracted from the original input. This more formally expressed as:

$$\boldsymbol{y} = \arg\max_{m \in \{1,...,M\}^N} s(\boldsymbol{x}, \boldsymbol{x}_{[m_1,...,m_N]}) \tag{2}$$

A compressive summarization simply deletes words from the input sentence:

$$\boldsymbol{y} = \arg\max_{m \in \{1,...,M\}^N, m_{i-1} < m_i} s(\boldsymbol{x}, \boldsymbol{x}_{[m_1,...,m_N]}) \tag{3}$$

This paper primarily considers abstractive summarization.

Summarization can naturally be done for many different lengths of input and requested summary lengths. In this paper, we restrict the task to simple sentence summarization, meaning that we use an input text of up to 100 words, and we try to produce a single sentence summary which captures the meaning of the input text.

For notational convience for the rest of the paper, we define an input sequence of length M, $(x_1, \ldots, x_M)$ and a summary sequence of length L, $(y_1, \ldots, y_L)$.

Background and Related Work

There has been some significant work done on sentence summarization. The state-of-the-art models in the field in recent years have primarily been implemented by Alexander Rush and Sumit Chopra and their colleagues at Harvard and Facebook.

In particular, in 2015, they demonstrated state-of-the-art performance on the sentence summarization task using a feed-forward window-based neural network with an attention mechanism [1]. In 2016, they used an attentive recurrent neural network (using an Elman recurrent neural network unit) to improve their results on the same task [4].

This paper draws much inspiration from these two papers and we encourage the reader to review them as they largely serve as a basis for this work.

Finally, any paper about summarization would be amiss not to mention the excellent Document Understanding Conference (DUC), which has historically served a test-bed for comparing modern summarization systems. They offer a variety of quality baselines for evaluating summarization systems, as well as a number of different tasks to evaluate summarization systems on various tasks (since naturally, summarization can take many forms which require slightly different evaluation).

Data

One of the primary difficulties in building a modern summarization system using deep-learning based approaches is the lack of quality summaries for large datasets. There are essentially no datasets which pair human-generated summaries specifically. However, there are a number of data sets which approximate summary-input pairs.

In particular, Gigaword is a corpus which consists of news articles collected from a variety of sources. This isn't the perfect dataset, as many of the articles are editorialized or aren't really ideal summaries of the content in the article. However, for the vast majority of these articles, the headline of the article can be used as an effective summary of the content of the article.

We use 2007 Gigaword, which consists of all of the news articles published by five newspapers from 1994 to 2007. In total, there are about five million articles.

For our test set, we use 500 articles from the Document Understanding Conference's Task 1, which simply considers systems which produce short summaries after reading articles.

Data Preprocessing

However, even though there are three million articles, many of these are not suitable for using as summary-headline pairs. To simplify the task, we only consider the first 100 words of the article. [1] We filter to remove (1) articles with editing marks or editorialization in the title and (2) articles with no non-stop words in common. This leaves us with approximately 1.4 million article-headline pairs to train our model on.

Preprocessing then consists of lower-casing all tokens, and replacing all numeric characters with a digit character. Finally, we use the most frequent 60,000 words as our vocabulary and replace all unknown words with a UNK word.

Technical Approach and Models

Information Retrieval Baseline

We use the same information retrieval baseline as in Rush et al. [1] This system just uses Okapi BM-25, an IR indexing system used by some search engines. This simply indexes the

[1] We do this to reduce training time and ensure we don't need to deal with long-term dependencies in the model. Rush et al. do something similar, but only consider the first 75 words of the article

training set and chooses the title with the highest score as the summary. The idea here is to control for memorizing titles.

We refer to this system as IR BL in the results section. [2]

Feed-Forward Deep Window Model

We use this simple model as a very basic baseline. We simply take a window of the embeddings of C=4 words of the generated summary so far, concatenate those vectors with the entire 100 word embeddings corresponding to the input, and embed those down to a hidden layer of size 100, and then use that hidden layer to project a softmax-based distribution over the entire vocabulary.

Formally, this model can be expressed as

$$\begin{aligned} p(\boldsymbol{y}_{i+1}|\boldsymbol{y_c}, \boldsymbol{x}; \theta) &\propto \exp(\boldsymbol{V}\boldsymbol{h}), \\ \boldsymbol{y'_c} &= [\boldsymbol{E}\boldsymbol{y}_{i-C+1}, \ldots, \boldsymbol{E}\boldsymbol{y}_i], \\ \boldsymbol{x'} &= [\boldsymbol{F}\boldsymbol{x}_0, \ldots, \boldsymbol{F}\boldsymbol{x}_L] \\ \boldsymbol{h} &= \tanh(\boldsymbol{U}[\boldsymbol{y'_c}; \boldsymbol{x'}]). \end{aligned}$$

We refer to this as Window BL in the results section.

Feed-Forward Attention-based Neural Network

We implement the feed-forward attention-based window model in [1]. This model can be best understood as generating each successive token in the summary by conditioning it on the entire input text and some prior window of previously generated tokens. The model was described by Rush et al. as a neural network language model with an encoder. Rush et al. implement a baseline encoder, convolutional encoder, and attention-based encoder, but get the best performance out of the attention-based encoder. For this reason, we only implement that encoder and describe it formally below:

$$\begin{aligned} p(\boldsymbol{y}_{i+1}|\boldsymbol{y_c}, \boldsymbol{x}; \theta) &\propto \exp(\boldsymbol{V}\boldsymbol{h} + \boldsymbol{W}enc(\boldsymbol{x}, \boldsymbol{y_c})), \\ \boldsymbol{y'_c} &= [\boldsymbol{E}\boldsymbol{y}_{i-C+1}, \ldots, \boldsymbol{E}\boldsymbol{y}_i], \\ \boldsymbol{h} &= \tanh(\boldsymbol{U}\boldsymbol{y'_c}). \end{aligned}$$

The attention based encoder is defined as:

$$\begin{aligned} enc(\boldsymbol{x}, \boldsymbol{y}_c) &= \mathbf{p}^\top \bar{\boldsymbol{x}}, \\ \mathbf{p} &\propto \exp(\boldsymbol{x'}\boldsymbol{P}\boldsymbol{y'_c}), \\ \boldsymbol{x'} &= [\boldsymbol{F}\boldsymbol{x}_1, \ldots, \boldsymbol{F}\boldsymbol{x}_M], \\ \mathbf{y'_c} &= [\boldsymbol{G}\boldsymbol{y}_{i-C+1}, \ldots, \boldsymbol{G}\boldsymbol{y}_i], \\ \forall i \quad \bar{\boldsymbol{x}}_i &= \sum_{q=i-Q}^{i+Q} \boldsymbol{x}'_i/Q. \end{aligned}$$

Here, the trainable parameters are $\theta = (V, W, E, U, G, F)$. We use a beam-search decoder to decode the actual summary.

We refer to this model as ABS in the results section.

[2] Since we have a smaller training set than in [1], we also use this to control for our intuitions about data size. We get a slightly lower baseline score than Rush et al., confirming our intuitions

Sequence-to-Sequence RNN with Attention

We also use a sequence-to-sequence (encoder-decoder) model with attention. We use both GRUs and LSTMs as the recurrent units in this model, but omit the technical details for brevity.

We use the same implementation of an LSTM as in Vinyals et al [2] and also use a standard GRU unit (see [3] for details of implementation). However differing from the above implementation, we use a bidirectional RNN.

We follow the implementation of Vinyals et al. as well for our attention model. Specifically, they present the following model. First, consider the sequence of encoder hidden states as $(h_{x_1}, \ldots, h_{x_M})$. Likewise, the hidden states of the decoder can be written as $(d_{y_1}, \ldots, d_{y_M})$.

After we've seen word t, we can define:

$$\begin{aligned} u_i^t &= v^T \tanh(W_1 h_i + W_2 d_t) \\ a_i^t &= softmax(u_i^t) \\ d'_t &= \sum_{i=1}^{M} a_i^t h_t \end{aligned}$$

where W_1, W_2, v are the trainable parameters.

We refer to this model as ARNN-{unit type} # ### in the result, where the first # is the number of RNN units and ### is the size of the hidden layer.

Results and Analysis

Quantitative Metric

Our primary metric is ROUGE, Recall Oriented Understudy for Gisting Evaluation. In particular, we use ROUGE-1 and ROUGE-2 as our metrics for evaluating the quality of our summaries. These metrics look at n-grams of various lengths that both the reference summaries and produced summaries have in common. These are the metrics used by DUC to evaluate the performance of the systems entered in their conference. Formally, ROUGE-N is defined as:

$$\frac{\sum_{S \in ReferenceSummary} \sum_{n-gram \in S} Count_{match}(n-gram)}{\sum_{S \in ReferenceSummary} \sum_{n-gram \in S} Count(n-gram)} \tag{4}$$

Training Details

Our implementation of the ABS system was very slow. It took significantly longer to train than the system described in [1]. We believe this is a product of both access to less computational resources in addition to potentially simply a slower implementation.

We used a 1000 article Gigaword validation set, and generally our validation scores were higher than our test scores.

We differ from the training of Rush et al. in that we used AdaGrad as opposed to Stochastic Gradient Descent [9]. However, we found our best performance using parameters similar to Rush et al. Hidden size 400, context size of 5, Q = 2. However, we found better performance by using an embedding size of 300 instead of 200. We get our best performance after 14 epochs of training. This took seven days of training time on a single GPU.

We used a batch size of 64 for training of all models.

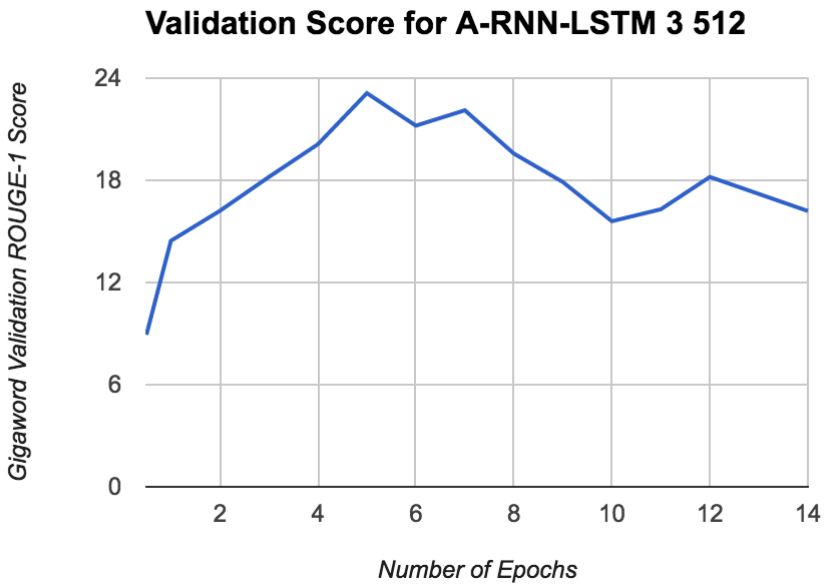
	DUC Test Dataset	
	ROUGE-1	ROUGE-2
IR BL	10.21	0.91
Window BL	9.29	1.02
ABS	15.21	2.12
A-RNN-LSTM 3 512	**21.21**	**5.01**
A-RNN-GRU 3 512	13.42	1.21
A-RNN-LSTM 3 256	20.57	4.67
A-RNN-GRU 3 256	11.84	1.09
A-RNN-LSTM 2 512	20.77	4.30
A-RNN-GRU 2 512	13.78	1.37
A-RNN-LSTM 2 256	20.79	4.36
A-RNN-GRU 2 256	10.77	1.07

We found that the RNN-based models were significantly better in terms of performance, and so we concentrated most of our time and resources training RNN-based models. [3]

We used Adagrad to train the RNN models. We tried training both 2 and 3 layer models, using both GRUs and LSTMs as the recurrent units, and tried hidden sizes of both 256 and 512.

We tended to get the best performance from each model after 4-7 epochs of training, with the exception of A-RNN-LSTM 2 256, which achieved its best performance after 10 epochs. The best performing model was A-RNN-LSTM 3 512. We include a chart of its validation score.

Generally, we saw significant improvements between the GRU-based models and the LSTM-base models. Doubling the size of the hidden layer helped in almost all models, but only yielded marginal improvements. This seems to suggest that LSTMs manage to capture something relatively significant in comparison to GRUs. However, it is important to note that the author's of [4] manage to get state-of-the-art results using the simple Elman recurrent unit (but compare their system to a nearly comparable LSTM-based translation model). [4]



[3] Quite literally: I spent over 500 dollars on AWS time on this project.

[4] An important distinction between our work and [4] is that the authors of [4] examine slightly shorter pieces of the input text, but not significantly so (100 words versus 75).

We felt that there were a couple of structural issues which hurt our score. The first was that ROUGE is a relatively unforgiving metric. There are many good ways to summarize a sentence. For each summary in DUC, we had four baseline summaries. But there were many cases where we would have very few overlapping words with any of the baseline summaries, despite producing what is clearly a reasonable summary of the content. Intuitively, we actually felt we got qualitatively better summaries from systems with slightly lower ROUGE-1 and ROUGE-2 scores.

The second issue was that our system naturally produced relatively short summaries, as the headlines we trained on had a length of, on average, 4.5 words. Adding penalties for short sentences helped to some extent, but often resulted in repeating the same key word over and over as opposed to producing better, but longer summaries. We believe this deficiency was due to the fact that our model never saw longer sentences and so understanding the grammatical structure of longer sentences was beyond its abilities.

Generally, we felt that the RNN-LSTM was significantly better than the ABS model in several regards. First of all, the model is quicker to train, both in terms of the speed per batch and in terms of number of epochs until the ROUGE score started to drop off on our validation set.

Qualitative Performance, Successes and Faults

We give some examples of the kinds of things that our system manages to summarize quite well. Generally, we got quite good performance summarizing articles that addressed topics that our system had seen before in the past. Generally, these DUC test items tended to be very news-related articles about topics that were relevant between 1994 and 2007 (the time that our data set spanned).

G: worker's perks in brazil being cut in effort to reduce deficit

G2: cardoso to unveil full deficit-cutting plan next week; part of imf deal.

RNN: brazil tries to revive economy

G: u.s. envoy tells milosevic to get out of kosovo or face nato attack

G2: milosovic must return military forces to bases to avoid nato attack.

RNN: u.s. envoy meets yugoslav leader

G: two dead, 15 missing in sydney-to-hobart race. maxi sayonara leads

G2: 15 missing, many injuries. 2 business post naiad sailors dead.

RNN: two dead in uk yacht accident

However, there were a number of things that our model did not manage to capture correctly. We noticed a couple of trends in terms of categorical errors by our system.

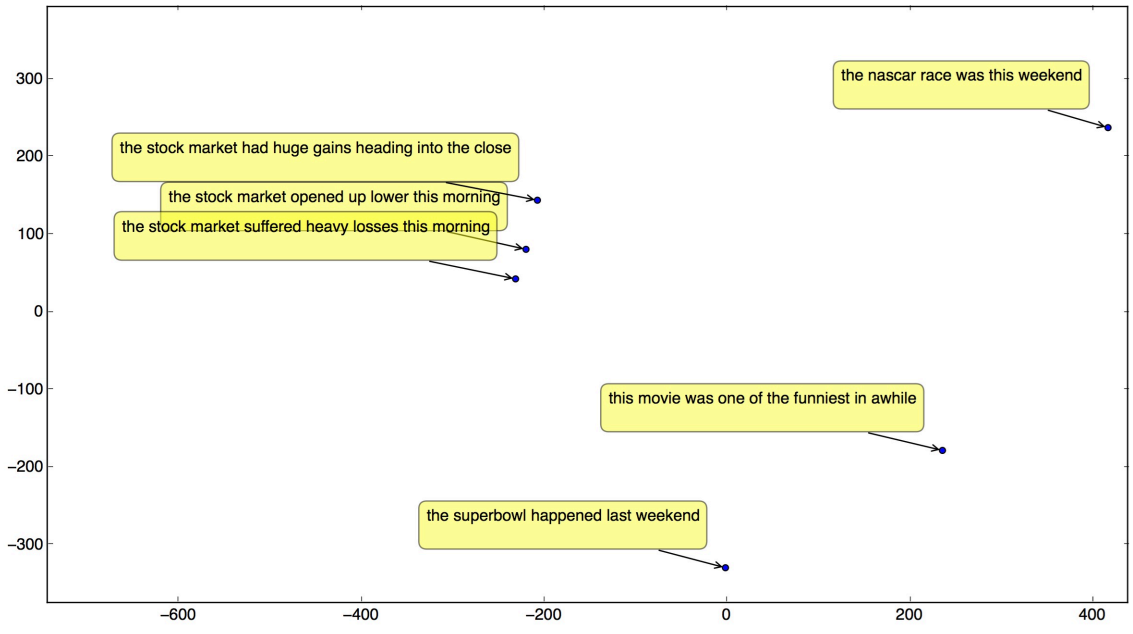
1. Our system failed when it was presented with a topic that it had seen very rarely in the training data. For example, if given paragraphs that were not news related, our model had a tendency to produce the most common starting tokens in our vocabulary. These tended to be "urgent" (an editorialized title which was not successfully removed by our filtering) or our "UNK" token.
2. Our system often had a tendency to repeat key words. For example, in a summary that was supposed to be about Hosni Mubarak meeting foreign leaders, our system produced "mubarak meets mubarak", clearly identifying that "mubarak" was an important part of the input, but failing to identify other important actors. In other, similar situations, our system would produce a good summary, but just repeat one word twice unnecessarily, often in a row. For example, in a summary that was about a political party meeting in malaysia, our system produced "malaysia's malayasia's

party meets". We believe that this systematic error points to deficiencies in the language model. We discuss this further in the conclusion.

3. Our model often fails "sufficient" semantic information, capturing a higher level summary than the gold-standard summaries capture. For example, in an article about a hurricane weakening, we see gold summaries such as "hurricane mitch weakens after widespread destruction in honduras", but our model only captures "hurricane _UNK off american coast". This is likely related to the data we used to train. The headlines capture significantly less information that the baseline summaries because they are shorter.

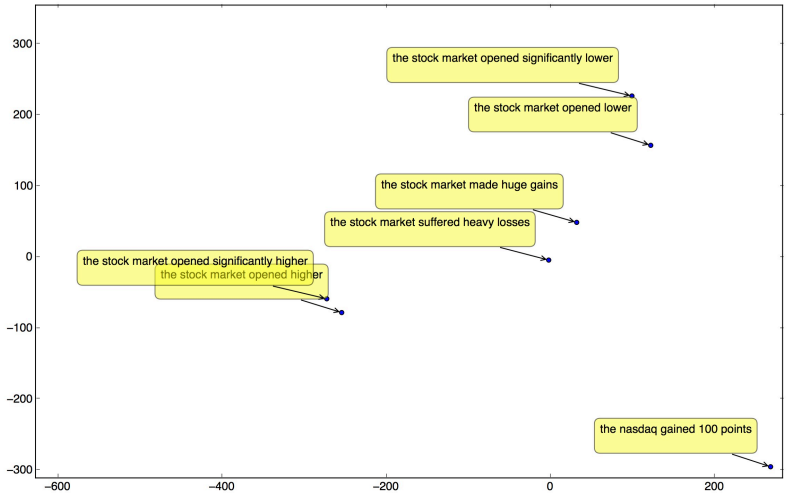
To try to get a better understanding of the inner-workings of our best model, we examined some of the sentence encodings that resulted from feeding various sentences into the network.

To produce this particular graphic, we encoded each sentence by feeding it into the encoder, collected the respective hidden states, and then used PCA to reduce the dimension of each vector so that we could represent it in a two-dimensional space.



Here we can see that the sentences which discuss similar topics are clustered together in two-dimensional space.

We were also curious about how the construction and semantics of particular sentences affected the distribution in vector space. Using the same technique, we input some simple sentences into the model which contained similar topics, but different semantic meanings projected into several different word-level constructions. We can see that while semantic differences do appear to cluster to some extent, sentences with similar constructions are much closer to one another than semantically similar constructions. This suggests that we could potentially make some significant improves to the system by coming up with some way of rewarding semantic capture more effectively.



Conclusions and Possible Future Work

In general, we felt that our project was quite successful. We managed to build and train two relatively complex architectures, and outperform reasonable baselines in the task. We wish we could have trained a more state-of-the-art model, as we believe there were minimal differences between the models we used and the current state-of-the-art abstractive summarizers (see [4] for the current best model), but we believe that much of the gap would have been made up with the larger datasets of [1] and [4].

One of the primary issues we felt existed with our sentence outputs was simply their lack of conformation to the standards of English. Consider the problem we discuss in our review of the model about repeated words. This kind of trait should be very low probability in English. However, even though we have a over a million training examples, we could significantly increase the size of our training corpus if we were to first train a recurrent language model and then use it to initialize the values for our decoder. This is similar to the concept of initializing word vectors by using GloVe or some other source of pretrained word vectors. Generally, we believe the literature supports improvements in models with smaller training sets using these pre-trained embeddings. [10]

In fact, the authors of [1] augment their decoder with a separately trained language model, and then decoder using beam search on the combined scores of the language model and the summarization model. They definitely demonstrated some improvements utilizing this set-up. We would be interested to see how effective pre-training a language model is versus augmenting the entire system in the same manner as [1].

We felt like one clear thing we could do to improve our model was increase the size of our dataset. The most recent version of Gigaword has several million more articles, and the original papers that inspired this work, [1] and [4] both used the larger version of Gigaword and also had better results. We suspect that our model would improve significantly with more data. [5]

Finally, we think it would be interesting to explore some alternative metrics, or find alternative datasets with a larger set of reference summaries. Even for relatively simple news articles, there were simply far too many possible "correct" ways to summarize a sentence without getting a particularly high ROUGE score. This would likely be improved with a larger set of reference summaries, but still potentially indicates that work exploring better metrics could yield significant improvements in system performance and comparison.

[5] Our work confirmed this. Testing with a dataset of half the size resulted in a performance decrease of over 50% on average for the RNN-LSTM models of all sizes

Acknowledgments

I would like to thank Richard Socher, who put together a truly interesting class and whose enthusiasm for the field made it completely approachable and Jencir Lee, who I worked with on a similar project in the fall and without whose advice I would have floundered considerably.

References

[1] Rush, A., Chopra, S. & Weston, J. (2015) A Neural Attention Model for Sentence Summarization. http://www.aclweb.org/anthology/D15-1044

[2] Vinyals, O., Kaiser L., et al. (2015) Grammar as a Foreign Language. https://arxiv.org/pdf/1412.7449v3.pdf

[3] Cho et al. (2014) Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. https://arxiv.org/pdf/1406.1078v3.pdf

[4] Chopra, S. et al. (2016) Abstractive Sentence Summarization with Attentive Recurrent Neural Networks. http://nlp.seas.harvard.edu/papers/naacl16_summary.pdf

[5] Bird, S, Ewan K, and E Loper (2009), Natural Language Processing with Python, O'Reilly Media.

[6] Abadi et al. (2015) TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. http://download.tensorflow.org/paper/whitepaper2015.pdf

[7] Chollet, F. (2015) Keras. https://github.com/fchollet/keras

[8] Theano Development Team. (2016) Theano: A Python framework for fast computation of mathematical expressions. https://arxiv.org/pdf/1605.02688.pdf

[9] Duchi, J. et al. (2010) Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. http://www.magicbroom.info/Papers/DuchiHaSi10.pdf

[10] Tian, Y. et al. (2016) On the convergent properties of word embeddings. https://arxiv.org/pdf/1605.03956.pdf

[11] Lin, C. (2004) ROUGE: A Package for Automatic Evaluation of Summaries. http://www.aclweb.org/anthology/W04-1013