Abstractive Summarization using a Feed-Forward Neural Attention Model

Alex Alifimoff
aalifimoff@stanford.edu

Jencir Lee
jli14@stanford.edu

Abstract

We implement a model from Rush et al. which performs abstractive sentence summarization. We train the model over a series of text, summary pairs scraped from Wikipedia. We then try to combine this model with distributed vector representations of sentences from Kiros et al. to try to perform summarization over larger texts.

1 Introduction

1.1 The Summarization Problem

Summarization is largely considered an unsolved problem in natural language processing. The goal of the summarization task is to produce a condensed version of the input text which preserves the meaning of the original as much as possible.

We focus specifically on abstractive summarization, as opposed to extractive or compression summarization. Abstractive summarization formally consists of an input string $\boldsymbol{x} \in \mathcal{X}$, where $\mathcal{X} \subset (\{0,1\}^V, \{0,1\}^V, ...\{0,1\}^V)$ where V is the size of our vocabulary. An abstractive system finds a second string $\boldsymbol{y} \in \mathcal{X}$ such that

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

An extractive 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:

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

Naturally, the abstractive summarization task is more difficult from an algorithmic standpoint, but does allow for much more expressive output.

1.2 Scoring

The first obstacle in approaching the summarization task is defining a score function. We work with an approximate scoring function that takes into account a fixed window of context, abbreviated $\boldsymbol{y_c}$. Thus we have,

$$s(\boldsymbol{x}, \boldsymbol{y}) \approx \sum_{i=0}^{N-1} g(\boldsymbol{y}_{i+1}, \boldsymbol{x}, \boldsymbol{y_c}) \tag{4}$$

We turn to the log probability as a realization of our score function:

$$s(\boldsymbol{x}, \boldsymbol{y}) = \log p(\boldsymbol{y}|\boldsymbol{x}; \theta) \tag{5}$$

$$\log p(\boldsymbol{y}|\boldsymbol{x}; \theta) \approx \sum_{i=0}^{N-1} \log p(\boldsymbol{y}_i + 1, \boldsymbol{x}, \boldsymbol{y_c}) \tag{6}$$

This allows us to focus our attention on modeling the probability of a particular word given its context and the input sentence, or $p(\boldsymbol{y}_{i+1}, \boldsymbol{x}, \boldsymbol{y_c})$.

2 Modeling

2.1 Language Model

We chose to reimplement a state-of-the-art model originally designed by Rush et al.[1] The model uses a feed-forward neural network to directly predict $p(\boldsymbol{y}_{i+1}, \boldsymbol{x}, \boldsymbol{y_c})$. The model makes use of word embeddings to augment the predictive power of the network. While the original model trains these word embeddings while training the standard weight matrices,

[1] http://arxiv.org/pdf/1509.00685v2.pdf

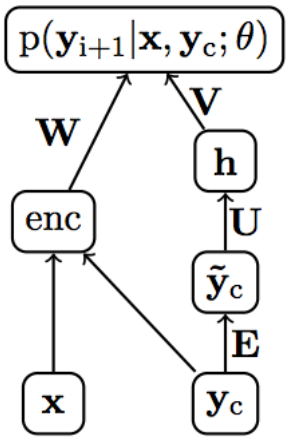

Figure 1: Network diagram for the Rush et al. summarization model

our model makes use of pre-trained word vectors to reduce the amount of training necessary to get a well-functioning model.

The model is:

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

where *enc* is an encoder. Figure 1 provides a network diagram of the model.

This model is paramaterized by $\theta = (\boldsymbol{E}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W})$. $\boldsymbol{E} \in \mathbb{R}^{DxV}$ is a word embedding matrix. The weight matrices are $\boldsymbol{U} \in \mathbb{R}^{(CD)xH}$, $\boldsymbol{V} \in \mathbb{R}^{VxH}$, and $\boldsymbol{W} \in \mathbb{R}^{VxH}$. H is the size of our hidden layer, V is the size of our vocabulary, D is the size of the word-embeddings and C is the length of our context.

2.2 Word Embeddings

To reduce the training time needed to train word embeddings from scratch, as well as to reduce how complicated the model was to implement, we chose to use pre-trained word embeddings. For this, we selected the commonly used Global Vectors for Word Representation, or GloVe embedding set. [2]

2.3 Encoders

In the original Rush et al. paper, they choose to implement three encoders: a convolutional neural model, an attention-based neural model, and a baseline bag-of-words model. The most effective encoder in the Rush et al. paper was the attention-based encoder, so we chose to simply implement the attention-based encoder and the baseline.

For the purpose of representing unknown words, we chose to average the 1,000 least common word vectors. We used this average vector each time an unknown word appeared in our training and test sets.

Bag-of-Words Encoder

The bag-of-words encoder is defined as:

$$\begin{aligned} enc_1(\boldsymbol{x}, \boldsymbol{y_c}) &= \mathbf{p}^\top \boldsymbol{x'} \\ \mathbf{p} &= [1/M, \ldots, 1/M], \\ \boldsymbol{x'} &= [\boldsymbol{F}\boldsymbol{x}_1, \ldots, \boldsymbol{F}\boldsymbol{x}_M]. \end{aligned}$$

This encoder works very simply. It has an internal embedding (in our implementation, we again used the GloVe set to approximate this embedding) to get an H dimensional representation, and then assumes a uniform distribution over the relationship to the input words. The new parameters of this encoder are $\boldsymbol{F} \in \mathbb{R}^{HxV}$.

Attention-based Encoder

The attention-based encoder is defined as:[3]

$$\begin{aligned} enc_3(\boldsymbol{x}, \boldsymbol{y_c}) &= \mathbf{p}^\top \bar{\boldsymbol{x}}, \\ \mathbf{p} &\propto \exp(\boldsymbol{x'}\boldsymbol{P}\boldsymbol{y'_c}), \\ \boldsymbol{x'} &= [\boldsymbol{F}\boldsymbol{x}_1, \ldots, \boldsymbol{F}\boldsymbol{x}_M], \\ \mathbf{y'_c} &= [\boldsymbol{G}\boldsymbol{y}_{i-C+1}, \ldots, \boldsymbol{G}\boldsymbol{y}_i], \\ \forall i \quad \bar{\boldsymbol{x}}_i &= \sum_{q=i-Q}^{i+Q} \boldsymbol{x}'_i/Q. \end{aligned}$$

In this model, $\boldsymbol{G} \in \mathbb{R}^{DxV}$ and $\boldsymbol{P} \in \mathbb{R}^{Hx(CD)}$. $\boldsymbol{F}$ is as defined in the Bag-of-Words encoder (note the parallels).

The motivation behind the attention-based encoder stems from reducing the complexity of the convolutional encoder, which was required to produce an output conditioned on the entire input sentence, as opposed to a particular context. Effectively, what the attention-based encoder does is augment the bag-of-words model

[2] We considered using word2vec, but the omission of a number of tokens necessary for our task led us to use GloVe instead

[3] We try to remain consistent with the notation in Rush et al. by labeling this as enc_3 to emphasize the ommission of the convolutional encoder

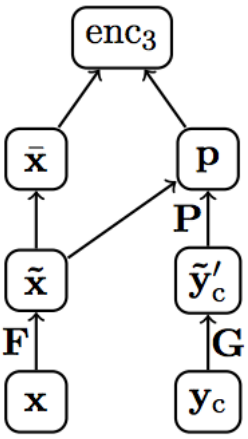

Figure 2: Network diagram for the Rush et al. attention-based encoder

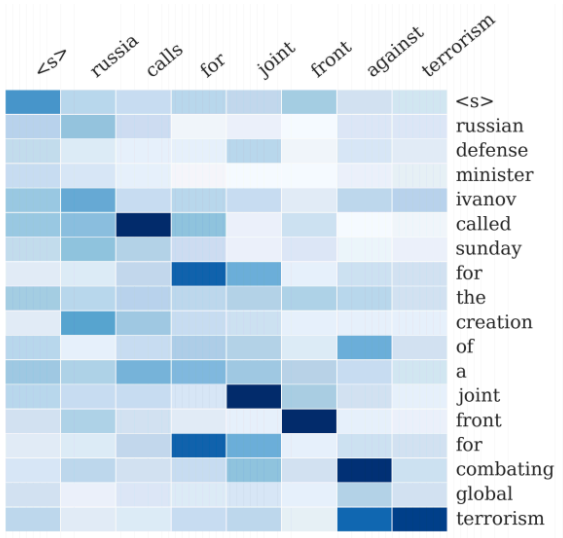

Figure 3: An example alignment (from Rush et al.)

with a learned soft-alignment, which essentially allows it to concentrate on a particular part of the input context, as opposed to building a representation over the entire input.

2.4 Training

Now that we have our method for arriving at $p(\boldsymbol{y}_{i+1}, \boldsymbol{x}, \boldsymbol{y}_c)$, we need some way to set the weights of the network. Here, Rush et al. choose to use negative log-likelihood. We can easily define the log-likelihood over a set of n input pairs, $(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})$.

$$\begin{aligned} \text{NLL}(\theta) &= -\sum_{j=1}^{J} \log p(\boldsymbol{y}^{(j)} | \boldsymbol{x}^{(j)}; \theta), \\ &= -\sum_{j=1}^{J} \sum_{i=1}^{N-1} \log p(\boldsymbol{y}_{i+1}^{(j)} | \boldsymbol{x}^{(j)}, \boldsymbol{y}_c; \theta) \end{aligned}$$

As is evident, this conveniently factors into the log of our predictions over each output word, conditioned on its context, the input, and our parameters. We then follow in the footsteps of Rush et al. and can minimize the negative log-likelihood using mini-batch stochastic gradient descent.

2.5 Search & Summary Generation

Although we have a convenient model for determining the score of a particular summary and a mechanism for scoring and training our model, we do not yet have a mechanism for generating the actual summary itself. We follow work in machine translation and instead of opting for an exact or greedy search, we opt to take the middle ground between the two. We implement a beam search which maintains the entire vocabulary while only considering K possible words as it generates each word of the summary.

However, our approach does not require nearly as complicated of a search algorithm as machine translation, as we can simply move from left-to-right generating words with no constraint about using each word in the input. We use the same simple beam search algorithm as Rush et al. and encourage the reader to turn there for more information.

3 Datasets

Because of the difficulty associated with obtaining the standard DUC dataset [4], commonly used as the benchmark in text summarization, we elected to collect our own dataset to train our model.

3.1 English Wikipedia / Simple Wikipedia

We found a wealth of potential text - summary pairs on Wikipedia. Simple English Wikipedia offered us a mirror of a large number of wikipedia pages. We made the reasonable assumption that the first sentence of the

[4]This dataset is composed of a wide variety of news articles. The Rush et al paper used the first sentence of these news articles in addition to the headline to train their model

Simple English wikipedia article would be a summary of the first paragraph of the standard English Wikipedia article.

We collected a total of 10,500 pairs of text/summary pairs from Wikipedia. We performed some filtering to remove articles that were not suitable for training for a variety of reasons - these included inability to programmatically find matches between the wikipedias, and similar problems (like hitting disambiguation pages).

4 Implementation

Our implementation was done in Theano, a symbolic toolkit for Python which is widely used by the deep-learning community. Theano is designed to work with CUDA, allowing for faster training of our neural network on distributed infrastructure. All of the code for our model implementation is available and included with submission. Additionally, we include the code for scraping Wikipedia, which was also written in Python.

4.1 Evaluation Metric

We turn to the standard metrics for evaluating sentence generation tasks. One popular metric for evaluating summaries is ROUGE, or Recall-Oriented Understudy for Gisting Evaluation, which has a variety of instantiations, based on the length of the n-gram used to identify similarity. We report ROUGE-1 and ROUGE-2. The metric is defined for some N as:

$$\frac{\sum_{S \in ReferenceSummary} \sum_{n-gram \in S} Count_{match}(n-gram)}{\sum_{S \in ReferenceSummary} \sum_{n-gram \in S} Count(n-gram)} \quad (7)$$

5 Limitations

Before we turn to our results, we want to consider some the limitations on our system that prevent it from being state-of-the-art.

The first major handicap is the lack of an extremely large database. Sadly, the difficulty to obtain the rather substantial DUC database that is typically used to train and evaluate this task majorly handicaps our model, as it inevitably lacks a substantial amount of training data that most algorithms operating in this space can utilize.

Second, our simplifying assumption to replace the embedding matrices in the model with the pre-trained set of GloVe vectors also comes at the cost of more accurate translations, as we can't train word embeddings that are specific to our task. This is linked to the first issue, as part of the motivation for us to use pre-trained embeddings stemmed from the lack of a huge training set that would be necessary to develop highly accurate word embeddings.

In particular, the potential limitations of using pre-trained embeddings is evident just in examining the model. There are a variety of embedding matrices using in the model (in fact, even the bag of words approach can potentially use an embedding separate from the one trained in the core language model). Having different matrices for different parts of the model allows us to develop highly specific embeddings for particular subtasks. Obviously, our model misses some of this nuance that was captured in the original Rush et al model.

Additionally, using pre-trained embeddings also required us to develop some novel approaches for dealing with inadequacies in our vector set. For example, GloVe does not include a vector for unknown words. We chose to approximate this vector by averaging the 1,000 least commonly seen vectors in the GloVe dataset. [5]

6 Results

Sadly, while we managed to create a functioning implementation of the model in Rush et al, we were handicapped by the amount of training time we could perform to build an effective model. Our best functioning model was trained with the following hyperparameters:

- GloVe Vector Size: 300
- Hidden Layer Size: 1000
- L2 Penalty Coefficient: 0.0
- Context Length: 5
- Summary Output Length: 20
- Search Beam Size: 10

[5] As noted previously, we chose to use GloVe instead of word2vec because of missing end-of-sentence tokens.

- Vocabulary Size: 50,000
- Mini-batch Size: 25

In order to speed up training, we had to reduce the size of the vocabulary significantly from the entire set available to us in GloVe. Additionally, we trained over 6.6k training examples and a test set of 200 training pairs and 600 training epochs. The training of this iteration of the model took approximately six hours. Additionally, we utilized Theano's CUDA support in addition to GPU's on the author's desktop and Amazon Web Services to train the model.

We tried several other iterations of the model, but as the training took such a long time and required most of the computing resources at our disposal, it was difficult to test a wide variety of variations of the model to find the best functioning one.

For context, the Rush et al. model managed to obtain a ROUGE-1 of 26.55 with an attention-based encoder and no additional tuning. With additional tuning, their model achieved ROUGE-1 of 28.18. We ran the pretrained Rush et al. model on our data as well and achieved a ROUGE-1 of 26.12.

6.1 Error Analysis

Mostly, our summaries only included unigram matches to the reference summaries, causing our ROUGE scores to be particularly low. In particular, our bag-of-words encoder did not manage to produce any bigram matches. We omit specific examples as they are mostly nonsense.

Our attention-based encoder was slightly better, and managed to produce a slim number of bigram matches, but, like the bag of words encoder, mostly produced non-sensical summaries. Largely, we managed to match a number of common phrases like "in the" and "for the". Interestingly, one common bigram match was "Gregorian calendar", likely due the prevalence of Wikipedia pages about particular years in our training set.

One of the particular problems that was introduced in our effort to reduce training time was reducing the size of our embedding matrix by removing uncommon words. However, one problem that this introduced is that there is significant information lost in a number of our summaries which have a large number of unknown words. In these summaries, we would frequently have no unigram matches.

Encoder	ROUGE-1	ROUGE-2
Bag-of-Words	1.23	0.00
Attention-Based	5.34	1.28

Table 1: Results on Wikipedia data set using both encoders

7 Extension: General Text Summarization

We propose a possible refinement of the general summarization algorithm to extend its application to general text summarization.

Some interesting work has surfaced in generating distributed vector representations of sentences, as opposed to simply works, to ideally encapsulate some amount of meaning. Primarily, we look to the work of Ryan Kiros' skip-thought vectors. [6] The skip-thoughts model abstracts the same methodology of the standard skip-gram model to the sentence level: that is, given a particular sentence in a sequence of sentences s_i, skip-thoughts vectors are encoded to predict the context of the sentence, or sentences s_{i-1}, s_{i+1}

7.1 Naive Approach

We modify the approach of Rush et al. to build an approach to paragraph level summarization using skip-thought vectors. The natural approach in this situation is to replace both embedding matrices $\boldsymbol{F}$ (in the encoders) and $\boldsymbol{E}$ (in the language model) and $\boldsymbol{G}$ (only in the attention-based encoder) with skip-thought sentence embedding matrices instead, and generalize the training set to the paragraph level.

This allows us to build a model to predict the probability of the next sentence given the context (which is now defined as a trailing number of sentences) and the input (now a vectorized sentence representation). In this case, we are generate some output $\boldsymbol{\gamma} = \boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_k}$ with each $\boldsymbol{y} \in \mathcal{X}$, where $\mathcal{X}$ is the set previously defined. Likewise, we have an input $\boldsymbol{\xi} = \boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_k}$ with each $\boldsymbol{x} \in \mathcal{X}$.

[6] http://arxiv.org/abs/1506.06726

In this case, we now have matrices $\boldsymbol{E} \in \mathbb{R}^{DxS}$, $\boldsymbol{F} \in \mathbb{R}^{HxS}$, and $\boldsymbol{G} \in \mathbb{R}^{DxV}$ with S being the cardinality of the set of all sentences.

Here, we gain significantly by having static embedding matrices, as representing a matrix with size S when considering any natural language is simply impossible. We can generate vectorized sentence representation with the approach from Kiros, and because we don't need to train those representations. Provided that the Skip-Thought vectors representation we have used are trained on a sufficiently general dataset. [7]

7.2 Refinement

However, this immediately poses some significant problems when we begin to consider the problem of generation. Applying a simple beam search to generate $\boldsymbol{\gamma}$ doesn't work as it did previously, as we have to generate entire sentences before we can apply the simple beam search used in the work of Rush et al. This makes the search space very large, and effectively prevents us from heuristically pruning during our generation phase, rendering our beam search useless. We try to develop an approach to still allow effective generation of sentences while making effective use of the skip-thoughts representation.

If we redefine our embedding matrix to its original manifestation $\boldsymbol{E} \in \mathbb{R}^{DxV}$ (in the language model), $\boldsymbol{F} \in \mathbb{R}^{HxV}$ for our context (in the encoder), we see we can use the same generation approach as before - using beam search to generate and score our sentences one at a time. In this sense, we use vectorized word representations to augment our language model, while using the vectorized sentence representations to augment context. This theoretically allows us to expand the length of the input context that we can use while allowing us to leverage the same algorithms.

Sadly, we were not able to test this approach due to the significant amount of training time required to get a functioning model, however we've successfully implemented the framework for training and testing the model. Because of the large size of the vectors associated with this model, we found there were substantial increases in the time needed to train an epoch to the point that it was not worth our limited time to train instead of the simpler model.

[7] We use the Skip-Thought vectors Python implementation provided by Kiros, which is trained on the significantly vast BookCorpus dataset.

8 Conclusions

Naturally, we were a little disappointed in the results of our summarizer, as we set relatively ambitious goals to expand the horizon of the field, however slightly. Largely, we believe that the main handicap of our model was the inability to train the model for a particularly long period of time. Even though a particular iteration of training only took on the order of 6 hours, it was difficult to experiment with a wide variety of hyperparameters.

However, it is possible that decisions along the design process also influenced the effectiveness of our model. Naturally, one advantage of the Rush et al. model is the ability to produce specific word embedding matrices for both input and context in the encoder, as well as a completely separate embedding matrix for the language model.

However, on the bright side, we developed an adaptation of the implementation of the Rush et al. model which opens the door for potentially more powerful summarization (given sufficient training time). We ensure that this model can be applied in the same generation framework as the general Rush et al. model.

8.1 Possible Future Work

Sadly, since such a large portion of our time was focused on implementing the model, substantial opportunities for extending the model weren't explored. We present a couple of our ideas on how to further refine the model as a possible inspiration for future work in the field of summary generation.

Markovization of Word Embeddings

The word embedding matrices provide a substantial number of opportunities for extension. One possible idea that we had was to extend the embedding matrix by considering not only the word, but the part of speech of the particular word being considered. Provided a substantially large enough training corpus, this could significantly augment the power of the transformation similarly to how markovization in parsing allows for refined accuracy in parse

trees. Similarly, markovization could be directly applied to the embedding matrix to generate embedding for n-grams, as opposed to unigrams.

Phrase Embeddings

One feature we implemented while considering word2vec vector representations of words was to do phrase detection, as the word2vec corpus includes a significant number of bigram and trigram phrases in addition to simple unigram words. Since we don't have the ability to do this while using the word embeddings from GloVe, it is likely that our algorithm suffers from this as well. We would borrow a page from the book of machine translation and consider using phrases in addition to simple words.

Coreference and Named Entities

Summary generation in particular could benefit from utilizing coreference resolver and named entity recognition to augment the beam search. There is a significant amount of tuning to particular metrics performed by Rush et al in their original paper. In future work, we expect there would be significant benefit in setting up a system built off this probability model and several other models designed to address specific deficiencies of our model, linked in a linearly-optimizable regression, similar to how work in machine translation scores translations across several models, combined with beam search. We believe this would easily help address some clear deficiencies in both our summaries and the summaries of Rush et al.