Document Summarization (draft)

Jeffrey Ling

February 16, 2017

Abstract

We propose a novel hierarchical hard attention method to modify the standard sequence-to-sequence (seq2seq) attention model. By organizing the source sequence into a 2-dimensional image, we hierarchically apply attention, using a stochastic mechanism for the first layer to select a row sequence for the second soft attention layer. While the computation for training standard seq2seq models scales linearly with source sequence length, our method is invariant to length and thus can scale arbitrarily.

We evaluate our model on the CNN/Dailymail document summarization task. As of this report we are able to match the performance of soft attention baselines using our new hierarchical method.

1 Introduction

Text summarization is an important problem for compressing large bodies of natural text into a more easily digestable form. Document summarization is one of the most challenging formulations of this problem, where given a document with several sentences of text, the goal is to produce a coherent summary that captures most or all of its salient points. To accomplish this, we use the most recent advances in deep learning to automatically produce summaries by iterating over a large dataset of given examples.

We can frame summarization as a sequence to sequence task: given a news article (the source sequence of words), we desire to produce a summary (the target sequence). Existing methods in deep learning have been developed and proven to be highly effective for this kind of task, especially the sequence-to-sequence (seq2seq) model applied to machine translation (**??**).

Existing seq2seq methods are limited by the length of source and target sequences; for a problem such as document summary, the source sequence is too long for seq2seq models to be effective (in both a computational sense and correctness sense). In order to scale these methods for this application, we are experimenting with reinforcement learning methods that aim to prune down the length of the source sequence in an intelligent way.

2 Related Work

2.1 Summarization

Originated with **?**

Summarization has historically been a difficult problem. To standardize the task, NIST released data for DUC (Document Understanding Conferences) between 2001-2007 (**?**). The DUC

tasks involved producing summaries for both single- and multiple-document sets of news articles. DUC 2001 and 2002 ask for general summaries of these articles documents and summaries, while DUC 2003-2006 also evaluate summaries based on their usefulness for certain question-answer tasks. While a single most effective metric for summarization may not exist, the DUC conferences established several important criteria, including grammaticality, non-redundancy, and content coverage (for which metrics like ROUGE (**?**) were created).

2.1.1 Methods

A variety of approaches have been used to solve summarization. Some of the most popular methods have been *extractive*, where sentences and words are pulled from documents based on relevance and pasted together (usually in document order). Some examples are **?**, which uses a simple information metric for ranking sentences, and **?**, which uses a simple neural network for the same purpose.

Extractive: CRF **?**

Another approach has been to relax general document summarization into the easier problem of sentence compression, where the source document consists of a single source. **?** employs a noisy channel model, similar to machine translation, to deduce the "most probable" compression, while **?** uses an integer linear program. **?** extend the tree-based methods to allow for insertions and substitutions during compression, whereas prior methods were purely deletion based. **?** successfully use a sentence compression algorithm along with an unsupervised topic model on the DUC 2004 task.

While extraction has proven to be successful, the method is inherently limited in its ability to summarize. The more challenging method, and also the closest to what humans do, is *abstractive* summarization. Instead of strictly requiring that all words of the summary come from the source document, any coherent text is allowed. In the past, such methods were limited to sentence compression tasks, as capturing inter-sentence dependencies and using a general vocabulary were beyond the scope of most statistical methods.

2.1.2 Deep Learning

With the onset of deep learning and sequence-to-sequence models, the abstractive approach has been reconsidered, as the only bottleneck for these methods is the quantity of data. **?** proposed a data-driven, completely abstractive model for summarizing short sentences by training a seq2seq model with attention. More recent work (**??**) scale the models to full documents, demonstrating that such methods are feasible.

These new models require a large amount of supervised training data, which now exist thanks to annotated CNN and Dailymail news stories (DUC data are limited in size and only suitable for evaluation). The task has not yet been fully standardized in this context, and researchers continue to debate what datasets are useful and what evaluation metrics to use (**?**).

2.2 Neural Architectures

Sequence-to-sequence methods have been highly successful in many tasks, including machine translation (**??**), question answering (**?**), dialogue (**?**), and in particular summarization (**?**).

? is one the first to envision a hierarchical LSTM system, where a neural autoencoder combines sentence representations into a paragraph representation. The idea of separating text into levels of abstraction is also used for hierarchical attention, which has been applied to summarization (**?**) and found to perform well.

The problem of full document summarization, however, is still very open. In order to make the models work, **?** use a variety of tricks: limiting the decoder vocabulary to the document, the use of pointer attention for `<unk>` tokens, etc. Our goal in this paper is not necessarily to optimize performance, but to understand how to scale up existing seq2seq methods in an efficient way, and so we attempt to eliminate the use of these ad-hoc tricks whenever possible. We aim to find the best implementation of "hard" attention, in which a discrete subset of the source document is selected at any given time, to satisfy the scalability condition.

Many techniques have been recently invented to handle the problem of scale. **?** propose the sparsemax function as a sparse alternative to softmax, and **?** use a nearest neighbors approach in "sparse access memory" to train a large-scale neural Turing machine.

These methods are designed to be fully differentiable and hence compatible with standard backpropagation. Reinforcement learning, however, is an alternative method for obtaining "hard" attention. In traditional reinforcement learning, an agent is trained to maximize the total expected reward for performing a series of actions. Such methods can be applied to deep neural networks by backpropagating an unbiased gradient at the action step, which is commonly known as the REINFORCE algorithm or policy gradient (**??**).

Deep reinforcement learning has been tried in the context of NLP (**???**) with varying degrees of success so far. We experiment with this in our paper.

Inspired by these ideas, we aim to improve on previous work for document summarization by (1) strengthening the hierarchical assumption with coarse features at the sentence level, and (2) including reinforcement learning, or one of recent sparse attention methods, for sparse attention. Below we describe our models in detail.

3 Models

3.1 seq2seq

We first describe the neural network architecture of the seq2seq models, also known as encoder-decoder models.

As described in **?**, an *encoder* recurrent neural network (RNN) reads the source sequence as input to produce a vector known as the *context*, and a *decoder* RNN generates the output sequence using the context as input. One popular RNN choice is the long-short term memory (LSTM) network (**?**).

More formally, a given sentence $w_1, \ldots, w_n \in \mathcal{V}$ is transformed into a sequence of vectors $x_1, \ldots, x_n \in \mathbb{R}^{d_{in}}$ through a word embedding matrix $E \in \mathbb{R}^{|\mathcal{V}| \times d_{in}}$ as $x_t = Ew_t$. An RNN is given by a parametrizable function f and a hidden state $h_t \in \mathbb{R}^{d_{hid}}$ at each time step t with $h_t = f(x_t, h_{t-1})$. This sequence of hidden states $h_1, \ldots, h_n$ jointly forms the context $c_{enc} = [h_1, \ldots, h_n]$ that is passed to the decoder.

The decoder is another RNN f_{dec} that generates output words $y_t \in \mathcal{V}$. It keeps hidden state $h_t^{dec} \in \mathbb{R}^{d_{hid}}$ as $h_t^{dec} = f_{dec}(y_{t-1}, h_{t-1}^{dec})$. Each word is predicted using another function g as

$$p(y_t | y_{t-1}, \ldots, y_1, c) = g(h_t^{dec}, c_{enc})$$

The models are trained to maximize the log probability of getting the sequences in the dataset correct. As the model is fully differentiable with respect to its parameters, we can train it end-to-end with stochastic gradient descent and the backpropagation algorithm.

The details of the function g will be described next.

3.2 Model 0: Standard Attention

In **?**, the function g is implemented with an *attention network*. We compute attention weights for each encoder hidden state h_i as follows:

$$\beta_i = h_i^T W h_t^{dec}$$

$$\alpha_i = \frac{\exp(\beta_i)}{\sum_{j=1}^{n} \exp(\beta_j)}$$

$$\widetilde{c} = \sum_{i=1}^{n} \alpha_i h_i$$

We normalize the α_i to sum to 1 over the source sentence words.

g is then implemented as

$$g(h_t^{dec}, c) = W_{out}(\widetilde{c}^T W_2 h_t^{dec}) + b_{out}$$

In essence, g computes a probability distribution α over the encoder hidden states, then takes the expectation of the encoder hidden state under α. The idea behind attention is to select the most relevant words of the source (by assigning higher attention weights) when generating output word y_t at time t.

Going forward, we call this *Model 0*.

3.3 Model 1 and 2: Hierarchical seq2seq

For a large source input like a document, it may be computationally inefficient to run an RNN over the entire source. Instead, we can consider organizing the document into distinct sentences and run an RNN separately over each. Specifically, assuming we have sentences $s_1, \ldots, s_m$ with words $w_{i,1}, \ldots, w_{i,n_i}$ for sentence s_i, we can apply an RNN to get corresponding hidden states $h_{i,j}$.

For attention, we then have two options. We can follow Model 0 and compute attention weights $\alpha_{i,j}$ for each hidden state $h_{i,j}$ by normalizing over all states. We call this *Model 1*.

Alternatively, rather than taking attention over the entire document, we can instead have a two-layered hierarchical attention mechanism: first, we have weights $\alpha_1, \ldots, \alpha_m$ for each sentence, and then for each sentence s_i, we have another set of weights $\widetilde{\alpha}_{i,1}, \ldots, \widetilde{\alpha}_{i,n_i}$.

The sentence level attention is computed using a different method: we first produce a representation of each sentence given the words $x_{i,1}, \ldots, x_{i,n_i}$ of the sentence. Our first option is *bag of words*: we simply take the sentence representation $u_i = \sum_{j=1}^{n_i} E x_{i,j} \in \mathbb{R}^{d_{in}}$, i.e. the sum of the word embeddings.

Alternatively, we can use a convolutional method: as in **?**, we perform a convolution over each window of words in the sentence. We use max-over-time pooling to obtain a fixed-dimensional sentence representation in $\mathbb{R}^{d_f}$ where d_f is the number of filters.

The final attention computed for word j in sentence i will thus be

$$\alpha_{i,j} = \alpha_i \cdot \widetilde{\alpha}_{i,j}$$

following the interpretation of $\alpha_{i,j}$ as the probability mass on $w_{i,j}$.

We call this method of attention *Model 2*.

4 Model 3: Hard Attention

With hierarchical attention, we still do not obtain significant gains in efficiency, since we have to compute RNN states for all words in the source document. Therefore, the idea that underlies this project is to apply stochastic sampling methods to the attention distribution α.

Specifically, rather than computing the context $\widetilde{c} = \sum_i \alpha_i h_i$, we can sample from the probability distribution α_i to obtain a single state h_i, and we set $\widetilde{c} = h_i$ as the sampled hidden state.

Known in the literature as "hard attention" (**?**), this model loses the property of being end-to-end differentiable and thus cannot be trained with standard backpropagation. However, reinforcement learning provides a way to circumvent this issue, as described below.

We take Model 2 and apply hard attention at the sentence level, but keep the word level attention per sentence as is. That is, we sample from the attention weights $\alpha_1, \ldots, \alpha_m$ to obtain a one-hot encoding for the sentence attention, and apply the same multiplication with this one-hot vector on the word-level attention weights $\widetilde{\alpha}_{i,1}, \ldots, \widetilde{\alpha}_{i,n_i}$. We call this *Model 3*.

4.1 Multiple Samples

From our initial experiments with Model 3, we found that taking a single sample was not very effective. However, we discovered that sampling multiple times from the distribution α significantly improves performance.

We sample based on the multinomial distribution $\text{Mult}(k, \{\alpha_i\}_{i=1}^n)$ to produce the sentence-level attention vector α of length n, with $\alpha_i = x_i/k$, where x_i is the number of times index i was sampled. k is a hyperparameter which can be tuned, and we found that $k = 5$ works well in our experiments.

The intuition here is for the hard attention model to more closely approximate the soft attention model, as it can select more sentences to produce the context.

5 Reinforcement Learning

To train Model 3, we must apply techniques from the reinforcement learning literature. We cast our learning problem with the neural network as stochastic agent and log probability as reward, and thus are able to apply reinforcement learning to train the network.

In our setup, where our agent is a parameterized model, computing the gradients for the model in this setup is known as the REINFORCE algorithm (**?**) or policy gradient, and has been well-explained in recent work (**???**).

Specifically, assuming we have a probability distribution $p(\alpha)$ from which we sample, and subsequently receive reward r, the gradient that is backpropagated from $p(\alpha)$ is

$$\nabla_\theta \mathcal{L}(\theta) = r \nabla_\theta \log p(\alpha) \tag{1}$$

i.e. the reward multiplied by the gradient of the log probability. A proof of this can be found in **?**.

In our framework, we use the log probability of the correct word at each time step as the reward r_t. Since samples at time t of the RNN decoder can also affect future rewards, we use a discount factor of $\gamma = 0.5$, so that the reward is $r = \sum_{s=t}^{n} \gamma^{n-s} r_s$ for the decoding sampler at time t.

5.1 Variance Reduction

While the policy gradient of equation **??** is proven to be unbiased, in practice it has such high variance that training converges far too slowly.

One of the most common ways to reduce the variance of the gradient estimator is to introduce a baseline reward b, which we subtract from our reward. Including a baseline is proven to reduce the variance of the estimator (**?**). We also normalize the rewards to a common scale by dividing by the reward variance in a given minibatch.

Our policy gradient equation then becomes

$$\nabla_\theta \mathcal{L}(\theta) = \frac{r - b}{\sigma} \nabla_\theta \log p(\alpha) \tag{2}$$

There are a few methods for producing the baseline; as in **?**, we can keep an exponentially moving average of the reward

$$b_j = (1 - \beta) b_{j-1} + \beta r_j$$

where r_j is the average minibatch reward and β is a hyperparameter (set to 0.9). Similarly, we keep a moving average of the variance for normalization:

$$\sigma_j^2 = (1 - \beta) \sigma_{j-1}^2 + \beta v_j$$

where v_t is the variance of the minibatch rewards for batch j. Since we have rewards at each time step of the decoder LSTM, we keep a separate moving average for the baseline for each time step, but we keep a single moving variance for all time steps.

While several papers suggest using a learned baseline from the RNN state (e.g. **?**), we have not found this to be more effective.

We also use an entropy term to reduce the variance. Our policy gradient equation then becomes

$$\nabla_\theta \mathcal{L}(\theta) = \frac{r - b}{\sigma} \nabla_\theta \log p(\alpha) - \lambda_{ent} \nabla_\alpha (\alpha \log \alpha) \tag{3}$$

where λ_{ent} is a hyperparameter. This has the effect of increasing the entropy of our sampling distribution, hence encouraging more exploration and faster convergence of learning.

5.2 Curriculum

Since training using policy gradients tends to be noisy and slow to converge, we experimented with a curriculum that starts training with soft attention and in epoch t, trains a minibatch using hard attention with probability $p_t = 1 - 1/\sqrt{t}$ (**?**).

While we found this to be helpful for single sample hard attention, it was not necessary for effective training with multisampled hard attention. We prefer to train solely with hard attention when possible, as we are able to save computation at training time.

6 Experiments

6.1 Methods and Models

Dataset: Experiments were performed on the CNN/Dailymail dataset from **?**. While the dataset was created for a question-answering task, the dataset format is suited for summary. Each data point is a news document accompanied by up to 4 "highlights", and we take the first of these as our target summary. Train, validation, and test splits are provided along with document tokenization and sentence splitting. We do additional preprocessing by replacing all numbers with # and appending end of sentence tokens to each sentence. We limit our vocabulary size to 50000 most frequent words, replacing the rest with `<unk>` tokens. We dropped the documents which had an empty source (which came from photo articles).

Table **??** lists statistics for the CNN/Dailymail dataset.

Dataset	CNN	Dailymail
Train size	90267	196962
Valid size	1221	12149
Average words per doc	794	832
Average sents per doc	21	29
Average words per sent	36	27
Average words per summary	13	15

Table 1: Statistics for CNN/Dailymail data.

6.2 Synthetic Pretraining

We found that unsupervised pretraining on the given dataset is beneficial to learning. For each document, we randomly sample 2 sentences and concatenate to form the target sentence in a new synthetic dataset. We can sample multiple times to have multiple targets for a given source document, and we found that 5 samples was most beneficial to learning (performance drops with significantly more samples). We then train on the synthetic dataset for 5 epochs and initialize future training with the learned weights.

6.3 Implementation Details

A few implementation details were necessary to make minibatch training possible. First, instead of taking attention over each individual sentence, we arrange the first 400 words of the document into a 10 by 40 image, and take each row to be a sentence. Second, we pad short documents to the maximum length with a special padding word, and allow the model to attend to it. However, we zero out word embeddings for the padding states and also zero out their corresponding LSTM states. We found in practice that very little of the attention ended up on the padding words.

Ideally, we would prefer to not truncate documents, especially since later context can be important for summarizing the document. Due to memory issues, this is a problem we still have to resolve.

Models: We ran experiments with Models 0 to 3 as described above. Model 0 serves as the baseline.

- Model 0: Soft attention.
- Model 1: Hierarchical LSTM, soft attention over all.
- Model 2: Hierarchical LSTM, soft hierarchical attention.
- Model 3: Hierarchical LSTM, hard attention over sentences.
- Model 3+multisampling: We include multisampling with $k = 5$.

Training: We train with minibatch stochastic gradient descent (SGD) with batch size 32 for 13 epochs, renormalizing gradients to be below norm 5. We initialize the learning rate to 1, and begin decaying it by 0.5 each epoch after the validation perplexity stops decreasing.

We use 2 layer LSTMs with 500 hidden units, and we initialize word embeddings with 300-dimensional word2vec embeddings (**?**). For convolutional layers, we use a kernel width of 6 and 600 filters.

7 Results

7.1 Evaluation

We report metrics for best validation perplexity and ROUGE scores (**?**). We use the ROUGE balanced F-scores with ROUGE-1 (unigrams), ROUGE-2 (bigrams), and ROUGE-L (longest common substring). We chose F-scores since recall is biased towards longer sentences.

To generate summaries for evaluation, we run beam search with a beam size of 5.

7.2 Baselines

For a baseline, we take the first 15 words of the document (chosen as the average length of a sentence in the training dataset). We call this FIRST.

7.3 Discussion

We notice that Model 2 has the best performance, while multisampling is comparable. We hypothesize that by sampling multiple times, the model learns to approximate the soft attention distribution

Model	PPL	ROUGE-1	ROUGE-2	ROUGE-L
FIRST	-	23.1	9.8	20.5
BERKELEY				
MODEL 0				
MODEL 1				
MODEL 2	16.2	24.5	12.0	22.9
MODEL 2+SYNTHPRE	16.0	23.7	11.5	22.0
MODEL 3				
MODEL 3+SAMPLE5	17	23.9	11.3	22.4
MODEL 3+SAMPLE5+SYNTHPRE	14.6	24.1	11.7	22.5

Table 2: Summarization results for CNN/Dailymail.