A DEEP REINFORCED MODEL FOR ABSTRACTIVE SUMMARIZATION

Romain Paulus, Caiming Xiong & Richard Socher
Salesforce Research
172 University Avenue
Palo Alto, CA 94301, USA
{rpaulus,cxiong,rsocher}@salesforce.com

ABSTRACT

Attentional, RNN-based encoder-decoder models for abstractive summarization have achieved good performance on short input and output sequences. For longer documents and summaries however these models often include repetitive and incoherent phrases. We introduce a neural network model with a novel intra-attention that attends over the input and continuously generated output separately, and a new training method that combines standard supervised word prediction and reinforcement learning (RL). Models trained only with supervised learning often exhibit "exposure bias" – they assume ground truth is provided at each step during training. However, when standard word prediction is combined with the global sequence prediction training of RL the resulting summaries become more readable. We evaluate this model on the CNN/Daily Mail and New York Times datasets. Our model obtains a 41.16 ROUGE-1 score on the CNN/Daily Mail dataset, an improvement over previous state-of-the-art models. Human evaluation also shows that our model produces higher quality summaries.

1 INTRODUCTION

Text summarization is the process of automatically generating natural language summaries from an input document while retaining the important points. By condensing large quantities of information into short, informative summaries, summarization can aid many downstream applications such as creating news digests, search, and report generation.

There are two prominent types of summarization algorithms. First, extractive summarization systems form summaries by copying parts of the input (Dorr et al., 2003; Nallapati et al., 2017). Second, abstractive summarization systems generate new phrases, possibly rephrasing or using words that were not in the original text (Chopra et al., 2016; Nallapati et al., 2016).

Neural network models (Nallapati et al., 2016) based on the attentional encoder-decoder model for machine translation (Bahdanau et al., 2014) were able to generate abstractive summaries with high ROUGE scores. However, these systems have typically been used for summarizing short input sequences (one or two sentences) to generate even shorter summaries. For example, the summaries on the DUC-2004 dataset generated by the state-of-the-art system by Zeng et al. (2016) are limited to 75 characters.

Nallapati et al. (2016) also applied their abstractive summarization model on the CNN/Daily Mail dataset (Hermann et al., 2015), which contains input sequences of up to 800 tokens and multi-sentence summaries of up to 100 tokens. But their analysis illustrates a key problem with attentional encoder-decoder models: they often generate unnatural summaries consisting of repeated phrases.

We present a new abstractive summarization model that achieves state-of-the-art results on the CNN/Daily Mail and similarly good results on the New York Times dataset (NYT) (Sandhaus, 2008). To our knowledge, this is the first end-to-end model for abstractive summarization on the NYT dataset. We introduce a key attention mechanism and a new learning objective to address the repeating phrase problem: (i) we use an intra-temporal attention in the encoder that records previous attention weights for each of the input tokens while a sequential intra-attention model in the decoder

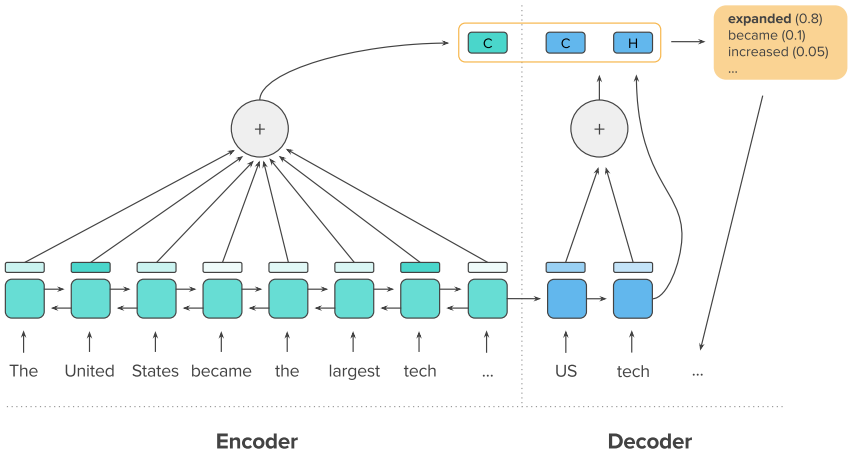

Figure 1: Illustration of the encoder and decoder attention functions combined. The two context vectors (marked "C") are computed from attending over the encoder hidden states and decoder hidden states. Using these two contexts and the current decoder hidden state ("H"), a new word is generated and added to the output sequence.

takes into account which words have already been generated by the decoder. (ii) we propose a new objective function by combining the maximum-likelihood cross-entropy loss used in prior work with rewards from policy gradient reinforcement learning to reduce exposure bias.

Our model achieves 41.16 ROUGE-1 on the CNN/Daily Mail dataset. Moreover, we show, through human evaluation of generated outputs, that our model generates more readable summaries compared to other abstractive approaches.

2 Neural Intra-Attention Model

In this section, we present our intra-attention model based on the encoder-decoder network (Sutskever et al., 2014). In all our equations, $x = \{x_1, x_2, \ldots, x_n\}$ represents the sequence of input (article) tokens, $y = \{y_1, y_2, \ldots, y_{n'}\}$ the sequence of output (summary) tokens, and $\|$ denotes the vector concatenation operator.

Our model reads the input sequence with a bi-directional LSTM encoder $\{\text{RNN}^{e\_\text{fwd}}, \text{RNN}^{e\_\text{bwd}}\}$ computing hidden states $h_i^e = [h_i^{e\_\text{fwd}} \| h_i^{e\_\text{bwd}}]$ from the embedding vectors of x_i. We use a single LSTM decoder RNN^d, computing hidden states h_t^d from the embedding vectors of y_t. Both input and output embeddings are taken from the same matrix W_{emb}. We initialize the decoder hidden state with $h_0^d = h_n^e$.

2.1 Intra-Temporal Attention on Input Sequence

At each decoding step t, we use an intra-temporal attention function to attend over specific parts of the encoded input sequence in addition to the decoder's own hidden state and the previously-generated word (Sankaran et al., 2016). This kind of attention prevents the model from attending over the sames parts of the input on different decoding steps. Nallapati et al. (2016) have shown that such an intra-temporal attention can reduce the amount of repetitions when attending over long documents.

We define e_{ti} as the attention score of the hidden input state h_i^e at decoding time step t:

$$e_{ti} = f(h_t^d, h_i^e), \tag{1}$$

where f can be any function returning a scalar e_{ti} from the h_t^d and h_i^e vectors. While some attention models use functions as simple as the dot-product between the two vectors, we choose to use a bilinear function:

$$f(h_t^d, h_i^e) = h_t^{dT} W_{\text{attn}}^e h_i^e. \tag{2}$$

We normalize the attention weights with the following temporal attention function, penalizing input tokens that have obtained high attention scores in past decoding steps. We define new temporal scores e'_{ti}:

$$e'_{ti} = \begin{cases} exp(e_{ti}) & \text{if } t = 1 \\ \frac{exp(e_{ti})}{\sum_{j=1}^{t-1} \exp(e_{ji})} & \text{otherwise.} \end{cases} \tag{3}$$

Finally, we compute the normalized attention scores α^e_{ti} across the inputs and use these weights to obtain the input context vector c^e_t:

$$\alpha^e_{ti} = \frac{e'_{ti}}{\sum_{j=1}^{n} e'_{tj}} \tag{4}$$

$$c^e_t = \sum_{i=1}^{n} \alpha^e_{ti} h^e_i. \tag{5}$$

2.2 Intra-decoder attention

While this intra-temporal attention function ensures that different parts of the encoded input sequence are used, our decoder can still generate repeated phrases based on its own hidden states, especially when generating long sequences. To prevent that, we can incorporate more information about the previously decoded sequence into the decoder. Looking back at previous decoding steps will allow our model to make more structured predictions and avoid repeating the same information, even if that information was generated many steps away. To achieve this, we introduce an intra-decoder attention mechanism. This mechanism is not present in existing encoder-decoder models for abstractive summarization.

For each decoding step t, our model computes a new decoder context vector c^d_t. We set c^d_1 to a vector of zeros since the generated sequence is empty on the first decoding step. For $t > 1$, we use the following equations:

$$e^d_{tt'} = h^{dT}_t W^d_{\text{attn}} h^d_{t'} \tag{6}$$

$$\alpha^d_{tt'} = \frac{exp(e^d_{tt'})}{\sum_{j=1}^{t-1} exp(e^d_{tj})} \tag{7}$$

$$c^d_t = \sum_{j=1}^{t-1} \alpha^d_{tj} h^d_j \tag{8}$$

Figure 1 illustrates the intra-attention context vector computation c^d_t, in addition to the encoder temporal attention, and their use in the decoder.

A closely-related intra-RNN attention function has been introduced by Cheng et al. (2016) but their implementation works by modifying the underlying LSTM function, and they do not apply it to long sequence generation problems. This is a major difference with our method, which makes no assumptions about the type of decoder RNN, thus is more simple and widely applicable to other types of recurrent networks.

2.3 Token generation and pointer

To generate a token, our decoder uses either a token-generation softmax layer or a pointer mechanism to copy rare or unseen from the input sequence. We use a switch function that decides at each decoding step whether to use the token generation or the pointer (Gulcehre et al., 2016; Nallapati et al., 2016). We define u_t as a binary value, equal to 1 if the pointer mechanism is used to output y_t, and 0 otherwise. In the following equations, all probabilities are conditioned on $y_1, \ldots, y_{t-1}, x$, even when not explicitly stated.

Our token-generation layer generates the following probability distribution:

$$p(y_t|u_t = 0) = \text{softmax}(W_{\text{out}}[h^d_t \| c^e_t \| c^d_t] + b_{\text{out}}) \tag{9}$$

On the other hand, the pointer mechanism uses the temporal attention weights α^e_{ti} as the probability distribution to copy the input token x_i.

$$p(y_t = x_i|u_t = 1) = \alpha^e_{ti} \tag{10}$$

We also compute the probability of using the copy mechanism for the decoding step t:

$$p(u_t = 1) = \sigma(W_u[h^d_t \| c^e_t \| c^d_t] + b_u), \tag{11}$$

where σ is the sigmoid activation function.

Putting Equations 9 , 10 and 11 together, we obtain our final probability distribution for the output token y_t:

$$p(y_t) = p(u_t = 1)p(y_t|u_t = 1) + p(u_t = 0)p(y_t|u_t = 0). \tag{12}$$

The ground-truth value for u_t and the corresponding i index of the target input token when $u_t = 1$ are provided at every decoding step during training. We set $u_t = 1$ either when y_t is an out-of-vocabulary token or when it is a pre-defined named entity (see Section 5).

2.4 Sharing decoder weights

In addition to using the same embedding matrix W_{emb} for the encoder and the decoder sequences, we introduce some weight-sharing between this embedding matrix and the W_{out} matrix of the token-generation layer, similarly to Inan et al. (2017) and Press & Wolf (2016). This allows the token-generation function to use syntactic and semantic information contained in the embedding matrix.

$$W_{\text{out}} = \tanh(W_{\text{emb}}W_{\text{proj}}) \tag{13}$$

2.5 Repetition avoidance at test time

Another way to avoid repetitions comes from our observation that in both the CNN/Daily Mail and NYT datasets, ground-truth summaries almost never contain the same trigram twice. Based on this observation, we force our decoder to never output the same trigram more than once during testing. We do this by setting $p(y_t) = 0$ during beam search, when outputting y_t would create a trigram that already exists in the previously decoded sequence of the current beam.

3 Hybrid learning objective

In this section, we explore different ways of training our encoder-decoder model. In particular, we propose reinforcement learning-based algorithms and their application to our summarization task.

3.1 Supervised learning with teacher forcing

The most widely used method to train a decoder RNN for sequence generation, called the teacher forcing" algorithm (Williams & Zipser, 1989), minimizes a maximum-likelihood loss at each decoding step. We define $y^* = \{y_1^*, y_2^*, \ldots, y_{n'}^*\}$ as the ground-truth output sequence for a given input sequence x. The maximum-likelihood training objective is the minimization of the following loss:

$$L_{ml} = -\sum_{t=1}^{n'} \log p(y_t^*|y_1^*, \ldots, y_{t-1}^*, x) \tag{14}$$

However, minimizing L_{ml} does not always produce the best results on discrete evaluation metrics such as ROUGE (Lin, 2004). This phenomenon has been observed with similar sequence generation tasks like image captioning with CIDEr (Rennie et al., 2016) and machine translation with BLEU (Wu et al., 2016; Norouzi et al., 2016). There are two main reasons for this discrepancy. The first one, called exposure bias (Ranzato et al., 2015), comes from the fact that the network has knowledge of the ground truth sequence up to the next token during training but does not have such supervision when testing, hence accumulating errors as it predicts the sequence. The second reason is due to the large number of potentially valid summaries, since there are more ways to arrange tokens to produce paraphrases or different sentence orders. The ROUGE metrics take some of this flexibility into account, but the maximum-likelihood objective does not.

3.2 Policy learning

One way to remedy this is to learn a policy that maximizes a specific discrete metric instead of minimizing the maximum-likelihood loss, which is made possible with reinforcement learning. In our model, we use the self-critical policy gradient training algorithm (Rennie et al., 2016).

For this training algorithm, we produce two separate output sequences at each training iteration: y^s, which is obtained by sampling from the $p(y_t^s|y_1^s, \ldots, y_{t-1}^s, x)$ probability distribution at each decoding time step, and $\hat{y}$, the baseline output, obtained by maximizing the output probability distribution at each time step, essentially performing a greedy search. We define $r(y)$ as the reward function for an output sequence y, comparing it with the ground truth sequence y^* with the evaluation metric of our choice.

$$L_{rl} = (r(\hat{y}) - r(y^s)) \sum_{t=1}^{n'} \log p(y_t^s|y_1^s, \ldots, y_{t-1}^s, x) \tag{15}$$

We can see that minimizing L_{rl} is equivalent to maximizing the conditional likelihood of the sampled sequence y^s if it obtains a higher reward than the baseline $\hat{y}$, thus increasing the reward expectation of our model.

3.3 Mixed training objective function

One potential issue of this reinforcement training objective is that optimizing for a specific discrete metric like ROUGE does not guarantee an increase in quality and readability of the output. It is possible to game such discrete metrics and increase their score without an actual increase in readability or relevance (Liu et al., 2016). While ROUGE measures the n-gram overlap between our generated summary and a reference sequence, human-readability is better captured by a language model, which is usually measured by perplexity.

Since our maximum-likelihood training objective (Equation 14) is essentially a conditional language model, calculating the probability of a token y_t based on the previously predicted sequence $\{y_1, \ldots, y_{t-1}\}$ and the input sequence x, we hypothesize that it can assist our policy learning algorithm to generate more natural summaries. This motivates us to define a mixed learning objective function that combines equations 14 and 15:

$$L_{mixed} = \gamma L_{rl} + (1 - \gamma) L_{ml}, \tag{16}$$

where γ is a scaling factor accounting for the difference in magnitude between L_{rl} and L_{ml}. A similar mixed-objective learning function has been used by Wu et al. (2016) for machine translation on short sequences, but this is its first use in combination with self-critical policy learning for long summarization to explicitly improve readability in addition to evaluation metrics.

4 Related work

4.1 Neural encoder-decoder sequence models

Neural encoder-decoder models are widely used in NLP applications such as machine translation (Sutskever et al., 2014), summarization (Chopra et al., 2016; Nallapati et al., 2016), and question answering (Hermann et al., 2015). These models use recurrent neural networks (RNN), such as long-short term memory network (LSTM) (Hochreiter & Schmidhuber, 1997) to encode an input sentence into a fixed vector, and create a new output sequence from that vector using another RNN. To apply this sequence-to-sequence approach to natural language, word embeddings (Mikolov et al., 2013; Pennington et al., 2014) are used to convert language tokens to vectors that can be used as inputs for these networks. Attention mechanisms (Bahdanau et al., 2014) make these models more performant and scalable, allowing them to look back at parts of the encoded input sequence while the output is generated. These models often use a fixed input and output vocabulary, which prevents them from learning representations for new words. One way to fix this is to allow the decoder network to point back to some specific words or sub-sequences of the input and copy them onto the output sequence (Vinyals et al., 2015). Gulcehre et al. (2016) and Merity et al. (2017) combine this pointer mechanism with the original word generation layer in the decoder to allow the model to use either method at each decoding step.

4.2 Reinforcement learning for sequence generation

Reinforcement learning (RL) is a way of training an agent to interact with a given environment in order to maximize a reward. RL has been used to solve a wide variety of problems, usually when

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.