State-of-the-art abstractive summarization

Peter Li
Stanford University
peter888@stanford.edu

Apurva Pancholi
Stanford University
apurva03@stanford.edu

Stylianos Serghiou
Stanford University
sstelios@stanford.edu

Abstract

We hereby detail our attempt to replicate the paper "A deep reinforced model for abstractive summarization" by Romain Paulus, Caiming Xiong and Richard Socher. This paper presents a model for abstractive summarization using an RNN-based encoder-decoder model with two main innovations. First, it attempts to reduce repetition in summarizing long texts using what the original authors called "intra-attention". Second, it combines a maximum likelihood objective with a reinforcement learning objective in an attempt to improve readability. The combined effect of these two innovations led to a state-of-the-art performance of 41.16% in ROUGE-1 F_1 score in the CNN/Daily Mail dataset. In this project we concentrate on replicating the first of these two innovations.

1 Introduction

Creating summaries of long texts, whether that pertains to medical research articles or patient records, is an essential, yet arduous and time-consuming aspect of medicine. We are interested in the prospect of automating at least some of those summarization tasks using state-of-the-art algorithms for automated summarization. To develop the appropriate knowledge and expertise, we agreed with Richard Socher to work on replicating a preprint on arXiv for which he is the senior author [1].

Automated summarization is the process of shortening a full text into a summary of its salient points. Extractive summarization creates such summaries by synthesizing salient phrases from the full text verbatim [2, 3]. In contrast, abstractive summarization creates an internal semantic representation of the text, which it translates into a summary much like the human process [4, 5]. We are interested in the latter task, which even though more challenging, can eventually lead to more coherent and concise summaries.

High quality summaries with abstractive summarization were only obtained very recently [5], by implementing the encoder-decoder model with attention for machine translation [6] to summarization. Nevertheless, early models could only achieve good performance with short inputs of one or two sentences being summarized into even smaller summaries [7]. Instead, attempts at analyzing longer full texts with longer summaries using the the CNN/Daily Mail dataset [8] led to summaries with substantial repetition of phrases.

The first model for abstractive summarization trying to address this problem [9] used an implementation of the coverage vector [10], which was first proposed in the neural machine translation literature. Coverage attempts to remind each subsequent decoder time-step of parts from the full text that have already been covered. The Paulus et al. [1] preprint that we hereby attempt to replicate addresses the same task by using novel intra-temporal and intra-decoder attention mechanisms, together known as "intra-attention", which keep track of previous aspects of the full text that have been attended to and of words that have already been generated.

Our aim in this replication study is to implement an encoder-decoder model with intra-temporal and intra-decoder attention, i.e. intra-attention, as detailed by Paulus et al. [1]. We also aim to replicate their pointer mechanism, using which will make our model capable of copying words of the full text

by a mechanism known as pointing [11] and as such compensate for words it would like to use but which are not found in its vocabulary. We will compare our performance to the study we are trying to replicate, previous models of extractive and abstractive summarization and to that of See et al. [9] with no coverage and no pointer mechanism.

2 Model Design

In this section we describe our baseline model and the following aspects of the Paulus et al. [1] study we are trying to replicate: (1) intra-temporal attention, (2) intra-decoder attention, (3) token generation and pointer and (4) repetition avoidance at test time. Our implementation differs from the original in the following aspects: (1) our pointer mechanism does not make use of named-entity recognition (NER), (2) we are using randomly initialized word embeddings instead of GLoVe vectors, (3) we are not sharing decoder weights and (4) we are using a maximum-likelihood objective rather than a combined maximum likelihood and reinforcement learning objective.

2.1 Baseline Sequence-to-Sequence Model with Attention

Our baseline model is that which was presented by See et al. [9] with no coverage and no pointer mechanism. First, each token of a full-text article was matched to a randomly initialized embedding matrix $E \in \mathbb{R}^{|V| \times d_x}$ and fed sequentially into a single-layer bidirectional LSTM-RNN encoder. This led to a sequence of encoder hidden states $h_i^e \in \mathbb{R}^{d_h}$, where $i \in \{1, 2, \ldots, n\}$. n was fixed and was taken to be 400, as per See et al. [9]. Full texts with less tokens than 400 were padded and our loss was masked to this padding. The final hidden state h_n^e was used to initialize the first decoder hidden state, such that $h_0^d = h_n^e$.

The decoder was a single-layer unidirectional LSTM-RNN, which at each time-step t computes hidden states $h_t^d \in \mathbb{R}^{d_h}$, where $t \in \{1, 2, \ldots, n'\}$. In training mode, each token of the target summary was fed sequentially into successive time-steps of the decoder. The decoder in training mode had a maximum sequence length of 100 and used the same embedding matrix E as the encoder. Decoder inputs with less than 100 tokens in length were padded and our loss was masked to this padding. In testing mode, we used beam search with a beam size of 5.

The attention distribution $\alpha_b^t \in \mathbb{R}^n$ was calculated as per Bahdanau et al. [6]. Attention can be seen as a way of forcing each time-step of the decoder to attend to appropriate encoder hidden states. We first calculated attention scores $e_{ti}^b \in \mathbb{R}^n$, which we normalized into the attention distribution and which we then used to weigh encoder hidden states into the context vector $c_t^b \in \mathbb{R}^{d_h}$.

$$e_{ti}^b = v^T \tanh(W_{\text{enc}} h_i^e + W_{\text{dec}} h_t^d + b_{\text{attn}}) \tag{1}$$

$$\alpha_t^b = \text{softmax}(e_t^b) \tag{2}$$

$$c_t^b = \sum_i \alpha_{ti}^b h_i^e \tag{3}$$

where W_{enc}, W_{dec} and b_{attn} are trainable parameters. The context vector was then used to create our predictions probability distribution $p(y_t) \in \mathbb{R}^{|V|}$ by,

$$p(y_t) = \text{softmax}(W_{\text{outr}}(W_{\text{innr}}[h_t^d \;\; c_t^b] + b_{\text{innr}}) + b_{\text{outr}}) \tag{4}$$

where, W_{outr}, W_{innr}, b_{outr} and b_{innr} are trainable parameters.

2.2 Intra-Temporal Attention

Intra-temporal attention was first introduced by Sankaran et al. [12]. Unlike attention used in our baseline model, this version of attention not only helps the decoder focus on particular parts of the source sequence, but also penalizes input tokens that have obtained high attention scores in past decoding steps. As such, intra-temporal attention is thought to be particularly helpful in avoiding repetition, especially when summarizing longer texts [5].

Attention scores, $e_{ti} \in \mathbb{R}$, for each input i at each decoder time-step t are a function of the encoder hidden state at input i, $h_i^e \in \mathbb{R}^{d_h}$, and the decoder hidden state at time-step t, $h_t^d \in \mathbb{R}^{d_h}$, such that,

$$e_{ti} = (h_t^d)^T W_{\text{attn}}^e h_i^e \tag{5}$$

where, $W_{\text{attn}}^e \in \mathbb{R}^{d_h \times d_h}$ is a trainable parameter. Input tokens that have obtained high attention scores in past decoder steps are penalized by normalizing current attention scores by previous attention scores. This leads to the new temporal scores $\tilde{e}_{ti} \in \mathbb{R}$, such that,

$$\tilde{e}_{ti} = \begin{cases} \exp(e_{ti}) & t = 1 \\ \frac{\exp(e_{ti})}{\sum_{j=1}^{t-1} \exp(e_{ji})} & t > 1. \end{cases} \tag{6}$$

The above temporal scores are normalized into the intra-temporal attention distribution, $\alpha_{ti}^e \in \mathbb{R}$, such that,

$$\alpha_{ti}^e = \frac{e'_{ti}}{\sum_{j=1}^{n} \tilde{e}_{tj}} \tag{7}$$

The temporal attention distribution is used to weigh the encoder hidden states into the encoder context vector, $c_t^e \in R^{d_h}$, such that,

$$c_t^e = \sum_{i=1}^{n} \alpha_{ti}^e h_i^e \tag{8}$$

2.3 Intra-Decoder Attention

Intra-temporal attention helps different time-steps of the decoder focus on different aspects of the encoded sequence. However, with longer summaries, repetition can still occur because in our baseline model subsequent decoder time-steps have no information about what aspects of the encoded sequence have already been decoded. Intra-decoder attention attempts to enhance our model's ability to avoid repetition by helping each subsequent decoder time-step attend to all previous decoder time-steps.

First, we calculate intra-decoder attention scores, $e_{tt'}^d \in \mathbb{R}$, where $t' \in [1, 2, \ldots, t-1]$, such that,

$$e_{tt'}^d = h_t^d W_{\text{attn}}^d h_{t'}^d \tag{9}$$

where W_{attn}^d is a trainable parameter. Then, we normalize the above intra-decoder attention scores into the intra-decoder attention distribution, $\alpha_{tt'}^d \in \mathbb{R}$, by dividing current attention scores with the total previous attention to a specific decoder step, such that,

$$\alpha_{tt'}^d = \frac{\exp(e_{tt'}^d)}{\sum_{j=1}^{t-1} \exp(e_{tj}^d)} \tag{10}$$

Finally, the intra-decoder attention distribution is used to weigh the decoder hidden states into the decoder context vector, $c_t^d \in \mathbb{R}^{d_h}$, such that,

$$c_t^d = \begin{cases} 0 & t = 1 \\ \sum_{j=1}^{t-1} \alpha_{tj}^d h_j^d & t > 1. \end{cases} \tag{11}$$

2.4 Token Generation and Pointer

We used a slight modification of the pointer-generator mechanism detailed in See et al. [9], which was inspired from Vinyals et al. [11], rather than the one in Paulus et al. [1], which was first presented

in Nallapati et al. [5] and Gülçehre et al. [13]. There are two main differences between the See et al. [9] and the Paulus et al. [1] pointers. First, the former predicts whether an out-of-vocabulary (OOV) word should be used at each time-step t conditional on attention distribution c_t^b, current state h_t^d and current decoder input $x_t \in \mathbb{R}^{d_x}$, whereas the latter computes the same probability conditional on intra-temporal context vector c_t^e, intra-decoder attention vector c_t^d and current hidden state h_t^d. Second, the former uses parameters which it trains against overall cross entropy loss, whereas the latter uses parameters which it trains directly to predict use of OOV or presence of named entity, versus not, as well as α_{ti}^e. It is unclear which of the two pointer mechanisms works better.

In our implementation, we substituted c_t^b by c_t^e and c_t^d. As such, we first calculated the scalar generation probability for time-step t, $p(u_t = 1) \in [0, 1]$, as,

$$p(u_t = 1) = \sigma(W_u[h_t^d \;\; c_t^e \;\; c_t^d] + b_u) \tag{12}$$

where, W_{gen} and b_{gen} are trainable parameters. Our token-generation layer generates the probability distribution, $p(y_t \mid u_t = 0) \in \mathbb{R}^{|V_e|}$, where V_e is the original vocabulary extended to include all OOV words seen in the full text and abstract, such that,

$$p(y_t \mid u_t = 0) = \begin{cases} 0 & \text{OOV} \\ \text{softmax}(W_{\text{out}}[h_t^d \;\; c_t^e \;\; c_t^d] + b_{\text{out}}) & \text{otherwise.} \end{cases} \tag{13}$$

where W_{out} and b_{out} are trainable parameters. The pointer mechanism uses the intra-temporal attention weights α_{ti}^e as the probability distribution, $p(y_t \mid u_t = 1) \in \mathbb{R}^{|V_e|}$, to copy the embedding x_i of input token w_i, such that,

$$p(y_t = x_i \mid u_t = 1) = \begin{cases} \alpha_{ti}^e & w_i \notin V \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

As such, the final distribution $p(y_t) \in \mathbb{R}^{|V_e|}$ is,

$$p(y_t) = p(u_t = 0)\, p(y_t \mid u_t = 0) + p(u_t = 1) \sum_{i:w_i=w} p(y_t = x_i \mid u_t = 1) \tag{15}$$

2.5 Loss

Our objective was to maximize the likelihood of observing ground-truth at time-step t, given all tokens of previous time-steps $1, \ldots, t-1$ and current decoder input x_t. We did so by minimizing the cross-entropy objective,

$$J_{\text{CE}} = -\sum_{t=1}^{T} \log p(y_t) \tag{16}$$

2.6 Repetition Avoidance at Test Time

In addition to the intra-attention mechanism detailed above, we further avoided repetition at test time by setting $p(y_t) = 0$ during beam search if outputting y_t would create a tri-gram that already exists in the previously decoded sequence of the current beam. This is based on the observation of the original authors that a ground-truth summary rarely includes repeating tri-grams.

2.7 Training and Hyperparameters

We used two 200-dimensional LSTMs for the bidirectional encoder and one 400-dimensional LSTM for the decoder. We limited the input vocabulary size $|V|$ to 50,000 tokens (unlike 150,000 from the paper) due to memory constraints, which was also identical to the output vocabulary. Input word embeddings were 128-dimensional (unlike the paper, where they were 100-dimensional) and were initialized using the truncated Normal distribution with a standard deviation of 0.0001 (unlike

using GloVe vectors). We trained all models with Adam [14] with a batch size of 50 and a learning rate $\alpha = 0.001$. We used gradient clipping with a maximum gradient Euclidean norm threshold of 2. We did not use any form of regularization, as we did not observe a reduction in validation set performance with further training. At test time, we used beam search of width 5 on all our models to generate our final predictions. We trained our final model on a Standard NV6 virtual machine by Microsoft Azure for a total of 3 days. Unlike the study under replication, we always trained against ground-truth summaries instead of choosing the previously generated token with 25% probability to reduce exposure bias. In training, all weight parameters were initialized using Xavier initialization and all bias terms were initialized to zero.

3 Data

We trained, evaluated and tested our model on the non-anonymized version of the CNN/Daily Mail dataset [5, 8] as found online[1]. This dataset contains online news articles (mean number of tokens, 781) alongside their summaries (mean number of sentences, 3.75; mean number of tokens, 56) categorized into 287,226 training pairs, 13,368 validation pairs and 11,490 test pairs.

4 Experiments

4.1 Experimental Setup

All experiments were done using parameters detailed in 2.7. We evaluated the impact of our modelling decisions by first fitting our baseline model, then replacing its attention mechanism by intra-temporal attention, then adding intra-decoder attention (collectively we refer to these as intra-attention) and finally adding the pointer mechanism. All comparisons were made at 3 epochs (about 55,000 iterations), which was approximately equivalent to training for 12 hours. We evaluated our performance in the validation and test set using the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) score [15]. We are using F_1 scores for ROUGE-1 (here taken to be the number of common unigrams (i.e. tokens) between reference summaries and generated summary irrespective of sequence), ROUGE-2 (as per ROUGE-1 for bigrams) and ROUGE-L (here taken to be the length of the longest number of in-sequence matches between reference and generated summaries). These were obtained using the `pyrouge` package[2]. In addition to the original paper we are providing 95% Confidence Intervals (CI) to express uncertainty in our estimates. We also completed many other experiments to test modelling decisions that are not presented in this report, such as masking for padding versus not and passing both decoder input and intra-attention as input to LSTM cells versus only decoder input.

4.2 Results

Our results in the test set are shown in Table 1. We are also comparing our results to recently published extractive and abstractive summarization attempts, as per Paulus et al. [1], the results of Paulus et al. [1] and the results of See et al. [9] (without coverage and pointer-generator). We are presenting results at 12 hours because we were unfortunately unable to retrieve standardized results of the final models and hyperparameters for a longer period of time. Progressive implementation of methods as per the study under replication did not predictably lead to better performance and our implementations did not manage to outperform our baseline. It also appears that combination of intra-temporal and intra-decoder attentions into a unified intra-attention, hindered rather than facilitated our model and that combination of Bandanau attention with our Intra-Decoder attention did not lead to an appreciable change in performance. Lastly, it appears that our model was not able to derive benefit from longer training and that almost half of the performance seen in our final model was due to the pointer mechanism.

[1] `https://github.com/JafferWilson/Process-Data-of-CNN-DailyMail`

[2] `pypi.python.org/pypi/pyrouge/0.1.3`

Table 1: ROUGE F_1 scores in percentages (%) and 95% CIs in percentages (%) on the test set. The models in the top section of the graph have already been published. The middle section of the table details our results for progressive implementations of Paulus et al. [1]. The final section presents our results for the fully-trained model.

Model	**ROUGE-1**		**ROUGE-2**		**ROUGE-L**	
	F_1	95% CI	F_1	95% CI	F_1	95% CI
Lead-3 [3]	39.2	-	15.7	-	35.5	-
SummaRuNNer [3]	39.6	-	16.2	-	35.3	-
words-lvt2k-temp-att [5]	35.46	-	13.30	-	32.65	-
ML, no intra-attention, pointer [1]	37.86	-	14.69	-	34.99	-
ML, with intra-attention, pointer [1]	38.30	-	14.81	-	35.49	-
Baseline [9]	31.33	-	11.81	-	28.83	-
Baseline	23.16	$\pm$0.80	7.63	$\pm$0.79	21.03	$\pm$0.77
Intra-Temporal	14.48	$\pm$0.56	3.93	$\pm$0.31	13.43	$\pm$0.52
Baseline + Intra-Decoder	22.72	$\pm$0.75	7.27	$\pm$0.70	20.67	$\pm$0.74
Intra-Attention	9.78	$\pm$0.44	2.41	$\pm$0.19	9.14	$\pm$0.40
Intra-Temporal + Pointer	20.11	$\pm$0.64	6.71	$\pm$0.43	18.61	$\pm$0.59
Baseline + Intra-Decoder + Pointer	25.96	$\pm$0.76	9.42	$\pm$0.63	23.36	$\pm$0.73
Intra-Attention + Pointer	17.60	$\pm$0.64	5.96	$\pm$0.39	16.41	$\pm$0.60
Intra-Attention + Pointer (3 days)	17.58	$\pm$0.67	5.85	$\pm$0.42	16.17	$\pm$0.61

4.3 Error Analysis

First, we set out to confirm that our models are minimizing cross-entropy loss by making convergence plots (Figure 1). Convergence plots for increasingly complex model specifications illustrate exponentially decreasing cross-entropy loss for the first 20,000 iterations followed by a plateau. The intra-temporal attention model displayed the least loss within the first 63,000 iterations, whereas the model with intra-attention and pointer generation displayed the most. The rest of our modelling choices did similarly well, even though based on Table 1 we expected less loss reduction with intra-attention. We would have also liked to plot loss for the seq2seq $\pm$ intra-decoder model.

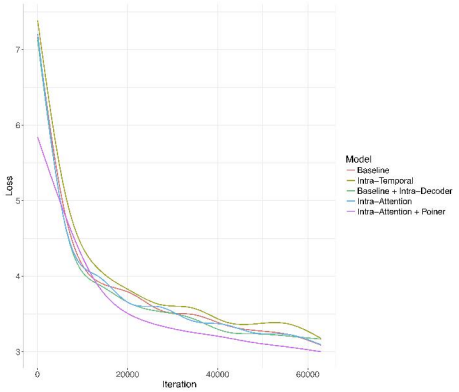

Figure 1: Loss for varying model specifications.

Second, we attempted to re-tune hyperparameters presented in Paulus et al. [1] for our full model against loss; loss was a more convenient proxy to tuning against ROUGE scores. We concentrated on tuning learning rate, which in view of time-constraints and indications of performance boosts detailed in See et al. [9], we believed was the most important hyperparameter to tune. We hereby present a 6 hour plot for some of the learning rates we tested against loss (Figure 2). Even though all presented learning rates led to a similar loss within the displayed iterations, our hyperparameter search confirmed the learning rate suggested in study under replication (alpha = 0.001).

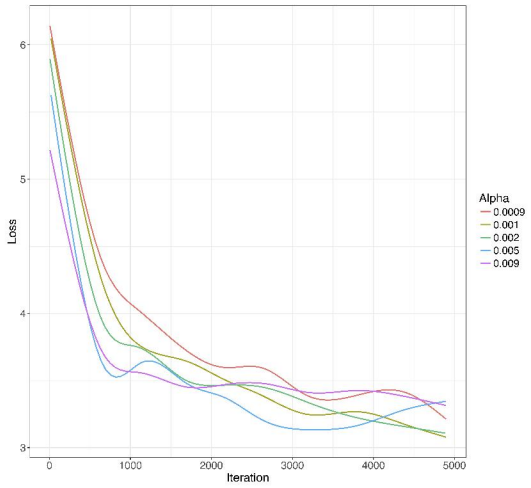

Figure 2: Hyperparameter search for learning rate (alpha).

Third, we attempted to quantify the ability of our model to overfit the train set and the extent to which this may hurt its performance in the validation set across iterations (Figure 3). Our model performed better in the training set than in the validation set at all times, as expected, and we did not observed a drop in performance within the validation set across epochs for which we were able to train our model. However, it did confirm that our model's performance does not improve significantly beyond roughly 30,000 iterations (i.e. almost 2 epochs) and that our model was unable to overfit the training set. We would have liked to repeat this experiment with (1) only one mini-batch of training examples and (2) variable learning rates across the run as previously suggested See et al. [9].

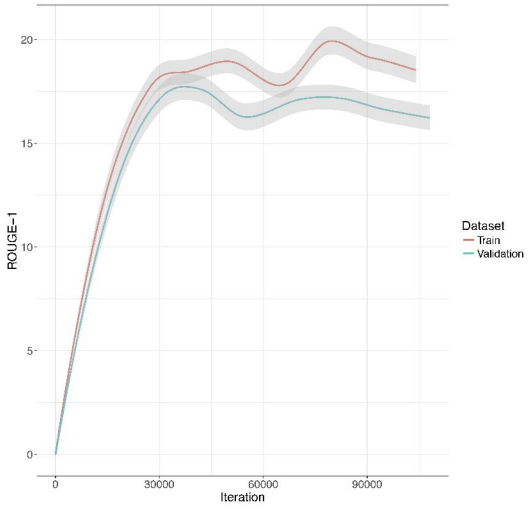

Figure 3: ROUGE-1 F_1 score for train versus validation set.

Lastly, we also explored the quality of summaries we obtained with different model specifications and we hereby present an example (Figure 4); these summaries were created after 12 hours of

training for each of the models shown. As shown, even though our model is a model for abstractive summarization, the grand majority of our summary content was extractive, regardless of model. All models were prone to repetition, even though we observed this becoming limited to phrase and sentence repetition after implementation of beam search as per Section 2.6. Models with intra-decoder were less prone to repetition and made more semantic sense. Interestingly, our model seemed capable of skipping phrases within " tokens and most summaries were surprisingly readable, despite the low ROUGE F_1 score.

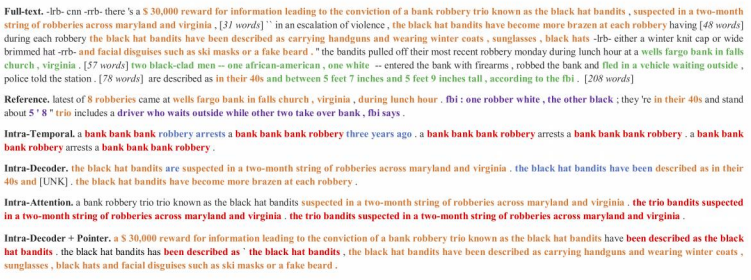

Figure 4: An example summary across different model specifications: **orange** for text extraction; **green** for information from the full text that were also seen in a summary; **purple** for clear paraphrasing of the full text; **blue** for words or phrases that were not found in the full text; **red** for repeated text.

5 Discussion

In this project we attempted to replicate a preprint by Paulus et al. [1]. We started off using data and a baseline model with no coverage and no pointer made available by See et al. [9] and we then attempted to build into it intra-attention and pointer from the study we are attempting to replicate. Unfortunately, our results did not appear to improve on our baseline and aspects of our model, such as intra-temporal attention, led to a significant decrease in baseline performance. Nevertheless, our error analysis identified that our model was able to learn and that important hyperparameters we were using were well-tuned. Analysis of summaries we obtained revealed that most of our performance was derived from extractive summarization and that intra-attention was not as effective as we had hoped in repetition avoidance.

Given further time we would also like to observe the distribution of attention in our model to identify whether this was allocated appropriately. We would also like to systematically ablate parts of our model to discover which parts of it led to the observed decrease in performance. Even though we ran multiple such ablations, the sheer number of combinations of what may be buggy did not let us run all possible such ablations.

Upon appropriate replication, we would be very interested in (1) automating the quantification of repetition in generated summaries, (2) exploring methods of helping our model pay attention to all parts of a full text rather than preferentially attending to its beginning (as shown in Figure 4), (3) identifying a dataset in which ground-truth summaries are not as extractive as the ones in our current dataset and (4) exploring the impact of feeding many more information to the model in the quality of its summarization, e.g. the article title, name of author and date of publication.

Acknowledgements

We would like to acknowledge help we received from our mentor Richard Socher. We would also like to acknowledge help from Abigail See via email and through her GitHub page[3] and help from Romain Paulus in troubleshooting and pointer implementation.

[3] `https://github.com/abisee/pointer-generator/blob/master/model.py`

References

[1] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *CoRR*, abs/1705.04304, 2017. URL http://arxiv.org/abs/1705.04304.

[2] Bonnie Dorr, David Zajic, and Richard Schwartz. Hedge trimmer: A parse-and-trim approach to headline generation, 2003.

[3] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 3075–3081, 2017. URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14636.

[4] Sumit Chopra, Michael Auli, and Alexander M. Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *HLT-NAACL*, 2016.

[5] Ramesh Nallapati, Bing Xiang, and Bowen Zhou. Sequence-to-sequence rnns for text summarization. *CoRR*, abs/1602.06023, 2016. URL http://arxiv.org/abs/1602.06023.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

[7] Wenyuan Zeng, Wenjie Luo, Sanja Fidler, and Raquel Urtasun. Efficient summarization with read-again and copy mechanism. *CoRR*, abs/1611.03382, 2016. URL http://arxiv.org/abs/1611.03382.

[8] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1693–1701. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5945-teaching-machines-to-read-and-comprehend.pdf.

[9] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368, 2017. URL http://arxiv.org/abs/1704.04368.

[10] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Coverage-based neural machine translation. *CoRR*, abs/1601.04811, 2016. URL http://arxiv.org/abs/1601.04811.

[11] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5866-pointer-networks.pdf.

[12] Baskaran Sankaran, Haitao Mi, Yaser Al-Onaizan, and Abe Ittycheriah. Temporal attention model for neural machine translation. *CoRR*, abs/1608.02927, 2016. URL http://arxiv.org/abs/1608.02927.

[13] Çaglar Gülçehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words. *CoRR*, abs/1603.08148, 2016. URL http://arxiv.org/abs/1603.08148.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[15] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In Stan Szpakowicz Marie-Francine Moens, editor, *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.