A Hierarchical Long Short-Term Memory Encoder-Decoder Model for Abstractive Summarization

Khuong Nguyen-Ngoc[1,3], Anh-Cuong Le[2⋆], and Viet-Ha Nguyen[1]

[1] VNU University of Engineering and Technology
144 Xuan Thuy, Cau Giay, Ha Noi, Viet Nam
[2] Faculty of Information and Technology, Ton Duc Thang University
19 Nguyen Huu Tho Str., Tan Phong Ward, District 7, Ho Chi Minh City
[3] Faculty of Information and Technology, Hai Phong University
171 Phan Dang Luu Str., Ngoc Son Ward, Kien An District, Hai Phong City
{khuongnn@dhhp.edu.vn, leanhcuong@tdt.edu.vn, hanv@vnu.edu.vn}

Abstract. Encoder-Decoder Sequence-to-Sequence Models have been proved to be powerful in many tasks of natural language generation (NLG). Text summarization is one of the most important tasks in NLG. Summarization of coherent long or very long text is challening due to the difficulties of modeling various relationships. In this work, we propose a novel Hierarchical Long Short-term Memory(HLSTM) to built Encoder-Decoder model for the task of abstractive text summarization. Extensive experiments on the two most popular corpora (Gigaword and Amazon Review) show that our proposed model performs better than several strong baseline models.

1 Introduction

Abstractive text summarization refers to the task of generating a short summary that retains the core meaning of the original text. Summarization is an important challenge towards natural language understanding [21,3]. Most successful summarization systems uses extractive approaches [7]. Different from the common extraction-based, abstractive methods are supposed to generated summaries from the source files, although they may not appear as part of the original [6].

Some recently research works use deep learning methods which have been proven to be promising in generating representations and language models [6,13,16,18]. Recurrent neural network (RNN) are powerful in natural language processing especially in learning representations of texts. However, the understanding of long documents is still far from satisfactory [23].

In this paper, we propose an novel encoder-decoder model that use to summarize a news article into its title. More specifically, we built model by combining a number of best practices in recent deep learning literature. Following Liu et al., [13], we use hierarchical Long-Short-Term-Memory(LSTM) to built an encoder - decoder model for abstractive text summarization. However, we differ by replacing unidirectional LSTMs by bidirectional LSTMs(Bi-LSTMs), in which the representations of sentences are learned by a Bi-LSTMs model whose inputs are words, and the representation of the document is learned by another Bi-LSTMs model whose inputs are sentences representations. A normal unidirectional LSTMs is used as a decoder.

2 Related Work

A lot of works in summarization are extractive methods, which are using word frequency to determined the importance of the word, in order to select senteces [15,8,22,9]. Abstractive methods, however, are more similar to the way of human beings generating summaries. Abstractive sentence summarization has been traditionally connected to the task of headline generation. Banko et al. [2] showed work using statistical machine translation directly for abstractive summarization. Cohn and Lapata [5] give a tree transduction compression method with a maxmargin learning algorithm.

Deep learning methods provided a framework for data-driven approach of generating summaries. In [4], a recurrent neural network with attention mechanism was built to generate summaries. Rush et al. [18] use convolutional models with attention mechanism, showing state-of-art performance on DUC tasks. Hu et al. [10] proposed a large data set for Chinese summarization, with recurrent neural network as encoder and decoder.

⋆ Corresponding author.

3 Model Architecture

3.1 Neural Encoder-Decoder Model

In sequence-to-sequence generation tasks, each input $X = x_1, x_2, ..., x_T$ is paired with a sequence of outputs to predict: $Y = y_1, y_2, ..., y_L$. Our task is to generate a summary sequence $Y = y_1, y_2, ..., y_L$, of L words, where $L < T$ such that the meaning of X is preserved: $Y = argmax_Y P(Y|X)$, where Y is a random variable denoting a sequence of L words. Typically the conditional probability is modeled by a parametric function with parameters θ: $P(Y|X) = P(Y|X;\theta)$. Training involves finding the θ which maximizes the conditional probability of sequence-summary pairs in the training corpus. If the model is trained to generate the next word of the summary, given the previous words, then the above conditional can be factorized into a product of individual conditional probabilities:

$$P(Y|X;\theta) = \prod_{t\in[1,L]} p(y_t|y_1, y_2, ..., y_{t-1}, x_1, x_2, ..., x_T;\theta) \tag{1}$$

in which:

$$P(Y|X) = \prod_{t\in[1,L]} p(y_t|y_1, y_2, ..., y_{t-1}, x_1, x_2, ..., x_T) = \prod_{t\in[1,L]} \frac{exp(f(h_{t-1}, e_{y_t}))}{\sum_{y'} exp(f(h_{t-1}, e_{y'}))} \tag{2}$$

$f(h_{t-1}, e_{y_t})$ denotes the activation function between h_{t-1} and e_{y_t}, where h_{t-1} is the representation hidden state from the LSTM at time $t-1$. Note that each sentence ends up with a special end-of sentence symbol $< eos >$. Commonly, the input and output use two different LSTMs with different sets of convolutional parameters for capturing different compositional patterns.

In this work, we used the Long-Short-Term-Memory (LSTM) cell which's basic architecture is shown in (Figure 1). This cell is composed of five main elements: an input gate i, a forget gate f, an output gate o, a recurring cell state c and hidden state output h. In this variant of LSTM cell, Apaszke[4] used a different approach for computing h_t. In the concrete, at every time step t, an internal memory cell $c_t \in R^n$ is added for computing h_t. Particularly, at each time step t, an LSTM cell uses the previous hidden state h_{t-1}, the input state x_t to produce the temp internal memory state c_in_t, then using the internal memory state c_{t-1} and the temp internal memory state c_in_t to produce the internal memory state c_t. By using the addition of gradient in these LSTM cells to minimize the gradient explosion(GE). Sundermeyer et al [19] showed that LSTM outperforms vanilla RNN in almost NLP tasks.

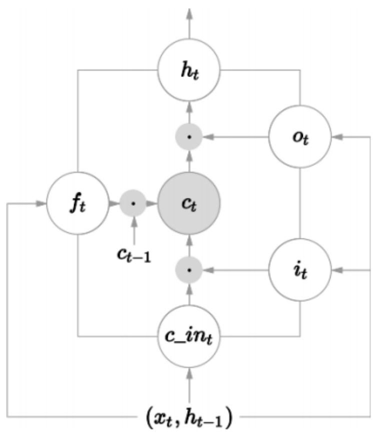

Fig. 1: LSTM cell [17]

Sutskever et al,.[20] proposed a neural sequence to sequence modeling approach which combines the encoder and decoder components as Figure 2. In which the LSTM Encoder takes the source sequence and maps it in to an encoded low-dimensional representation. The encoded low-dimensional representation is then used by the LSTM Decoder to generate a high-dimensional target sequence. In abstractive text summarization, the LSTM Encoder encodes meaning of the

[4] https://apaszke.github.io/lstm-explained.html

source sentence into a low-dimensional vector then the LSTM Decoder can use that low-dimensional vector and generate a summarization. While generating the summarization, the generation of each new word depends on the current word in source sentence, features of the model and the textual of preceding generated words.

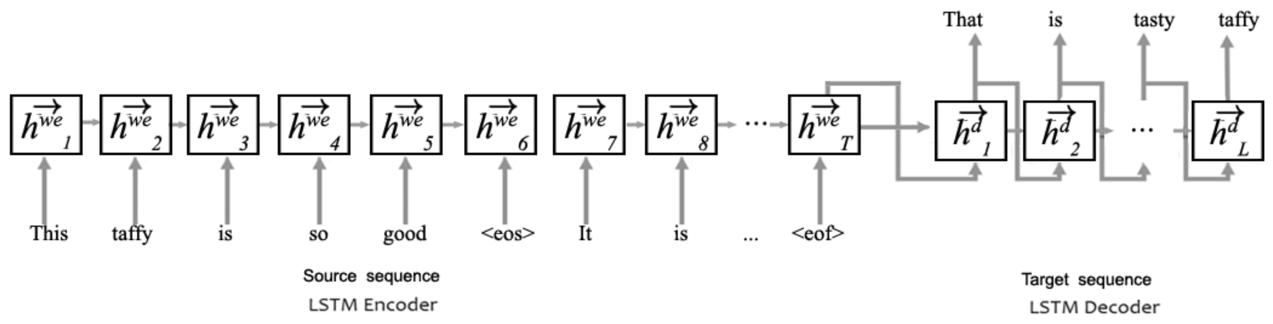

Fig. 2: Neural Encoder-Decoder framework.

Recently, Liwei et al.,[11] have been shown as one the best models for natural language generation of coherent long texts like paragraphs or longer documents. In addition, they also show that hierarchical LSTM models can play some role in enabling such more sophisticated generation tasks like summarization. In our work, we employ hierarchical LSTM models for text summarization where input of the sentence LSTM layer uses output of the previous(word level) layer. Note that, our model can be seen as an extension of their model.

3.2 Hierarchical Encoder-Decoder

Let D denote a paragraph or a document, which is comprised of a sequence of N_D sentences, $D = s_1, s_2, ..., s_{N_D}, < eod >$. An additional "$< eod >$" token is appended to each document. Each sentence s is comprised of a sequence of tokens $s = w_1, w_2, ..., w_{N_s}$ where N_s denotes the length of the sentence, each sentence ending with an "$< oes >$" token. The word w is associated with a K-dimensional embedding $x_w, x_w = x_w^1, x_w^2, ..., x_w^K$. Let V denote vocabulary size. Each sentence s is associated with a K dimensional representation e_s.

An encoder is a neural model where output units are directly connected with or identical to input units. Typically, inputs are compressed into a representation using neural models (encoding), which is then used to reconstruct it back (decoding). The encoder first compresses document D into a vector representation e_D and then reconstructs D based on e_D.

For clarification, we first describe the following notations used in encoder and decoder:

– h_t^{ew} and h_t^{es} denote hidden vectors from LSTM models in encoder component; h_t^{dw} denotes hidden vectors from LSTM models in decoder component, the subscripts of which indicate timestep t, the second character of superscripts of which indicate operations at word level (w) or sequence level (s).
– x_t^{ew} and x_t^{es} denote word-level and sentence-level embedding for word and sentence at position t in terms of its residing sentence or document at encoding stage.
– y_t^{dw} denotes word-level embedding for word at position t decoding stage.

Model 1: Standard Encoder-Decoder The whole input and output are treated as one sequence of tokens. Following Sutskever et al. [20] and Bahdanau et al. [1], we trained an encoder that first maps input documents into vector representations from a $LSTM_{encode}$ and then reconstructs inputs by predicting tokens within the document sequentially from a $LSTM_{decode}$. Two separate LSTMs are implemented for encoding and decoding with no sentence structures considered. Illustration is shown in Figure 2.

Model 2: Hierarchical LSTM Encoder-Decoder The overall architecture of the Hierarchical Long Short-Term Memory Encoder-Decoder model is show in Figure 3. Which draws on the intuition that just as the juxtaposition of words creates a joint meaning of a sentence, the juxtaposition of sentences also creates a joint meaning of a paragraph or a document.

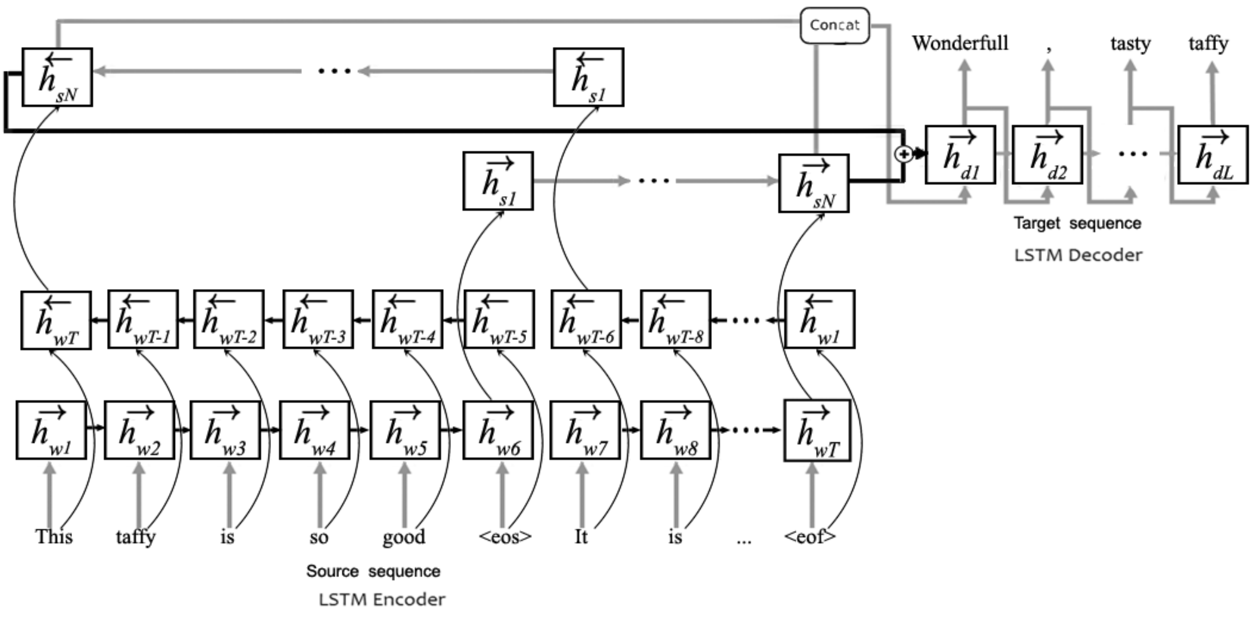

Fig. 3: Hierarchical Encoder-Decoder framework.

Encoder We first obtain representation vectors at the sentence level by putting one layer of Bidirectional LSTM (denoted as $B - LSTM^{ew}$) on top of its containing words. At word level representation, we also use bidirectional LSTM to exploit information both from the past and the future. The describes of basic computations in an LSTM cell in our encoder component was rewritten as follow:

$$\begin{aligned}
i_t^{ew} &= \sigma(U_{xi}^{ew} x_t^{ew} + W_{hi}^{ew} h_{t-1}^{ew} + b_i^{ew}) \\
f_t^{ew} &= \sigma(U_{xf}^{ew} x_t^{ew} + W_{hf}^{ew} h_{t-1}^{ew} + b_f^{ew}) \\
o_t^{ew} &= \sigma(U_{xo}^{ew} x_t^{ew} + W_{ho}^{ew} h_{t-1}^{ew} + b_o^{ew}) \\
c\_in_t^{ew} &= tanh(U_{xc}^{ew} x_t^{ew} + W_{hc}^{ew} h_{t-1}^{ew} + b_{c\_in}^{ew}) \\
c_t^{ew} &= f_t^{ew} \odot c_{t-1}^{ew} + i_t^{ew} \odot c\_in_t^{ew} \\
h_t^{ew} &= o_t^{ew} \odot tanh(c_t^{ew})
\end{aligned} \tag{3}$$

The vector output at the ending time-step is used to represent the entire sentence as $x_t^{se} = h_{eos_t}^{ew}$. To build representation e_D for the current document/paragraph D, another layer of LSTM (denoted as $LSTM^{es}$) is placed on top of all sentences, computing representations sequentially for each timestep:

$$\begin{aligned}
i_t^{es} &= \sigma(U_{xi}^{es} x_t^{es} + W_{hi}^{es} h_{t-1}^{es} + b_i^{es}) \\
f_t^{es} &= \sigma(U_{xf}^{es} x_t^{es} + W_{hf}^{es} h_{t-1}^{es} + b_f^{es}) \\
o_t^{es} &= \sigma(U_{xo}^{es} x_t^{es} + W_{ho}^{es} h_{t-1}^{es} + b_o^{es}) \\
c\_in_t^{es} &= tanh(U_{xc}^{es} x_t^{es} + W_{hc}^{es} h_{t-1}^{es} + b_{c\_in}^{es}) \\
c_t^{es} &= f_t^{es} \odot c_{t-1}^{es} + i_t^{es} \odot c\_in_t^{es} \\
h_t^{es} &= o_t^{es} \odot tanh(c_t^{es})
\end{aligned} \tag{4}$$

Representation e_D computed at the final time step is used to represent the entire document: $e_D = h_{eod}^{es}$. Thus one LSTM operates at the token level, leading to the acquisition of sentence-level representations that are then used as inputs into the second LSTM that acquires document-level representations, in a hierarchical structure.

Decoder The decoding algorithm operates on one layer of LSTMs. The basic computations of each LSTM cell was used in the work of prakash et al.,[17]. The initial time step $h_0^{ds} = e_D$, the end-to-end output from the encoding procedure. h_{t-1}^{dw} is used as the original input into $LSTM^{dw}$ for subsequently predicting tokens within word t. According to [17] the following describes the basic computations in an LSTM cell:

$$\begin{aligned} i_t^d &= \sigma(U_{xi}^d x_t + W_{hi}^d h_{t-1}^d + b_i^d) \\ f_t^d &= \sigma(U_{xf}^d x_t + W_{hf}^d h_{t-1} + b_f^d) \\ o_t^d &= \sigma(U_{xo}^d x_t + W_{ho}^d h_{t-1} + b_o^d) \\ c\_in_t^d &= tanh(U_{xc}^d x_t + W_{hc}^d h_{t-1} + b_{c\_in}^d) \\ c_t^d &= f_t^d \odot c_{t-1}^d + i_t^d \odot c\_in_t^d \\ h_t^d &= o_t^d \odot tanh(c_t^d) \end{aligned} \tag{5}$$

The output of the network at time t is:

$$o(t) = \beta(W_{do} h_t^d) \tag{6}$$

The training stage aims to learn the parameters U_{x*}, W_{h*} for x and h respectively; $\sigma(.)$ is usually the $sigmoid$ function; $tanh(.)$ denotes hyperbolic tangent functions; β refer to $softmax$ function; $\odot$ is the multiplication operator; and b denotes the bias.

3.3 Training

Given a training corpus $S = (x^i, y^i)_i^S = 1$ of S document-summary pairs, the above model can be trained end-to-end using stochastic gradient descent by minimizing the negative conditional log likelihood of the training data with respect to θ:

$$\mathcal{L} = -\sum_{i=1}^{S}\sum_{t=1}^{N} logP(y_t^i | y_1^i, ...y_{t-1}^i, x^i; \theta) \tag{7}$$

where the parameters θ constitute the parameters of the decoder and the encoder.

Like other monolingual machine translation tasks, the main objective of training is determining model parameters for maximizing the log probability of the summarization sequences given the source sequences. For that reason, the desire decoded summarization sequence is selected in the set of all candidate sequences that has the biggest score. To find best summarization sequence, the definitely of beam size is used as a small set of hypotheses (candidate set).The total score for all and each of these hypotheses need to be computed. In their work, Sutskever et al,. [20] have showed that although a beam size of 1, it usually achieves good results while with a higher beam size it is always better. Once the parametric model is trained we generate a summary for a new document x through a wordbased beam search such that $P(y|x)$ is maximized, $argmaxP(y_t|y_1, ..., y_{t-1}, x)$. The search is parameterized by the number of paths k that are pursued at each time step.

For all the models in the experiment, we used 200 dimensional word2vec vectors [14] trained on the same corpus to initialize the model embeddings, but we allowed them to be updated during training. The hidden state dimension of the encoder and decoder was fixed at 400 in all our experiments. When we used only the first sentence of the document as the source, as done in Rush et al. (2015), the encoder vocabulary size was 119,505 and that of the decoder stood at 68,885. We used Adadelta[24] for training, with an initial learning rate of 0.001. We used a batch-size of 50 and randomly shuffled the training data at every epoch, while sorting every 10 batches according to their lengths to speed up training. We did not use any dropout or regularization, but applied gradient clipping. We used early stopping based on the validation set and used the best model on the validation set to report all test performance numbers. For all our models, we employ the large-vocabulary trick, where we restrict the decoder vocabulary size to 2,0005, because it cuts down the training time per epoch by nearly three times, and helps this and all subsequent models converge in only 50%-75% of the epochs needed for the model based on full vocabulary.

4 Experiments

4.1 Dataset

We obtained Amazon reviews from Stanford Network Analysis Project (SNAP). The raw dataset includes 34,686,770 reviews spanning all kinds of products: books, video games, toys, music, etc, where each entry contains product, user information, ratings, a plain text review and a summary.

Because the dataset is so large, it is unlikely that we will be able to feed all of them into our models. In order to iterate fast, We selected one specific product, book, for testing and debugging our model. We utilized 20,626 reviews for training and 3278 reviews for validation. For test please see multi dataset.

However, one product is unlikely to suffice; we are also interested in understanding how our models perform for various products. For this purpose, we mix and match multiple products into one dataset. We utilized 440,324 reviews for training, 66,893 for validation and 14,836 for test. Multi-product data include the following products: ”Books,” ”Electronics,” ”Clothing Shoes & Jewelry,” ”Movies & TV” and ”Home & Kitchen.” Please note that the test data also include two additional out-of domain products: ”Sports & Outdoors” and ”Health & Personal Care.”

It is useful to know a few statistics of the dataset beforehand. We use those statistics to tweak bucketing for RNN, as you will see in section 6. By querying data using Google BigQuery, we found that interquartile range of number of words per review is 26-64 words, and interquartile range of number of words per summary is 4-7 words. Additionally, the average number of reviews is more than 100 words, which means the review dataset is skewed.

A lot of preliminary work was devoted to preparing the data. We first exclude titles that contain encode non-words like HTML tags and titles that get accidentally truncated to 128 characters. We then split them into training, validation and test set, and massage them into the format we want. Specifically, we remove question marks, semicolon, colon and punctuation, and convert all the characters to lower cases.

We also evaluate our model on the annotated Gigaword corpus as described in Rush et al. [18]. We used the scripts made available by the authors of this work[5] to preprocess the data, which resulted in about $3.8M$ training examples. The script also produces about $400K$ validation and test examples, but we created a randomly sampled subset of 2000 examples each for validation and testing purposes, on which we report our performance. Further, we also acquired the exact test sample used in Rush et al. [18] to make precise comparison of our models with theirs. We also made small modifications to the script to extract not only the tokenized words, but also system generated parts-of-speech and named-entity tags.

4.2 Evaluation

For evaluation, we use ROUGE-1, ROUGE-2 and ROUGE-L [12]. ROUGE is a package of standard metrics for summarization tasks; it measures similarity between reference summary and test summary. **ROUGE-N** is recall-oriented measure; it divides the number of matching n-grams by the number of total n-grams in the reference summary.

$$ROUGE - N = \frac{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} Count(gram_n)} \tag{8}$$

ROUGE-L compares the longest common sequence (LCS) between reference summary and test summary. It eliminates the need to pre-define n-gram length. The intuition is that the longer the LCS is, the more similar those two text are. Given X reference summary X of length m and test summary Y of length n, it computes an F-measure:

$$\begin{aligned} R_{lcs} &= \frac{LCS(X,Y)}{m} \\ P_{lcs} &= \frac{LCS(X,Y)}{n} \\ F_{lcs} &= \frac{1+\beta^2 R_{lcs} P_{lcs}}{R_{lcs}+\beta^2 P_{lcs}} \end{aligned} \tag{9}$$

[5] https://github.com/facebook/NAMAS

5 Results

5.1 ROUGE score on Gigaword Dataset

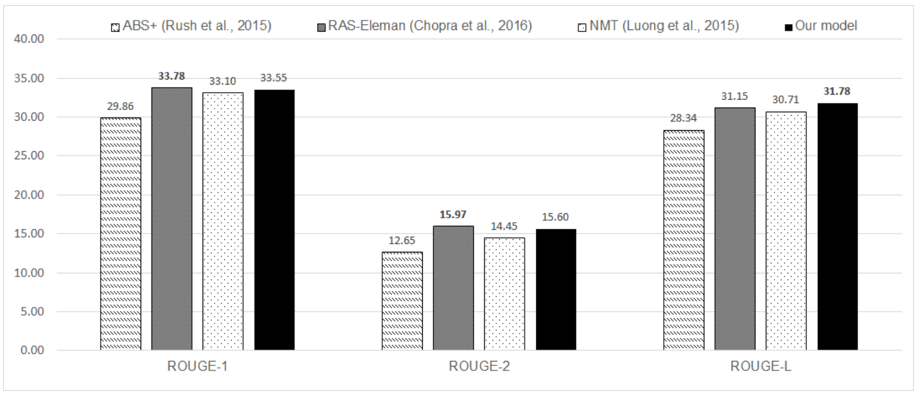

Fig. 4: F1 ROUGE scores on the Gigaword (smaller than 150 words).

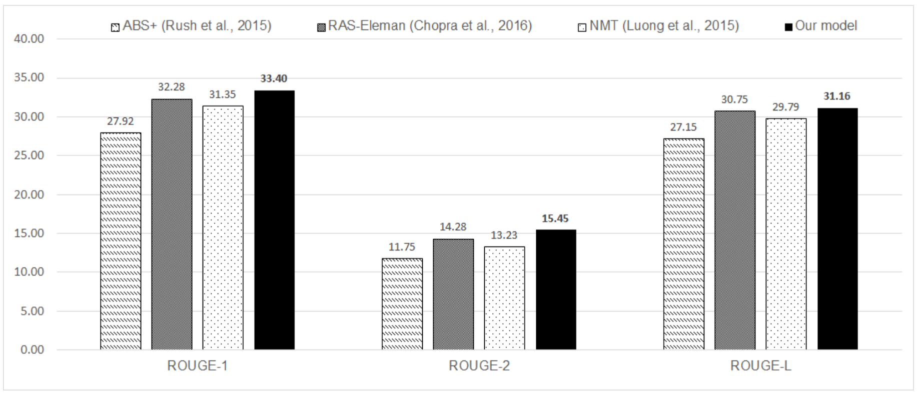

Fig. 5: F1 ROUGE scores on the Gigaword (larger than 150 words).

Figure 4 and 5 show that our propose model achieves better performance than all other models on all metrics with long text (larger than 150 words) but performs slightly worse than RAS-Eleman model [4] as well as one of the best model with short text (smaller than 150 words) on ROUGE-1 metric.

5.2 ROUGE score on Amazon Review Dataset

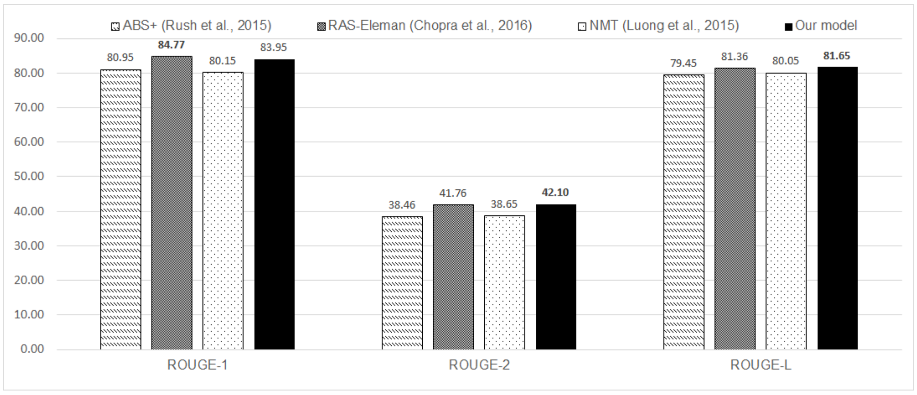

Fig. 6: F1 ROUGE scores on the Amazon Review (smaller than 150 words).

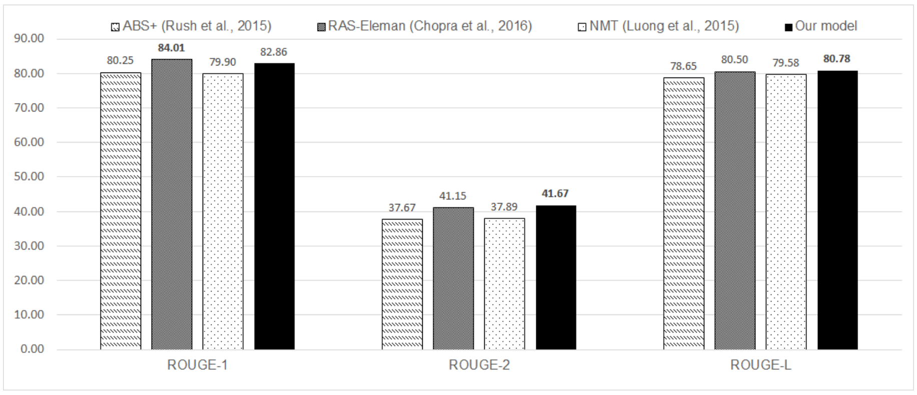

Fig. 7: F1 ROUGE scores on the Amazon Review (larger than 150 words).

The ROUGE scores of the test set is shown in Figure 6 and 7, which show that the our model is almost better than other model, but outperforms the state-of-the-art model(RAS-Eleman) [4] on two of the three variants of ROUGE, while being competitive on ROUGE-1.

6 Conclusion

In terms of natural language processing, most problems can be transformed to the problem of getting representations of language elements (words, sentences, paragraphs, and documents). The work of word to vector was well developed. However, the sentence to vector is still far from satisfactory. RNN models on natural language processing is still not powerful enough like CNN models on computer vision. This might because the LSTM models are not good enough to model sentences, or the sentences can be modeled by LSTM, but the neural network is not "deep" enough to understand them. The former problem might be solved by adding hidden variables to neural networks, like combining RNN and hidden Markov models. In this work, we apply the hirarchical LSTM encoder-decoder model for the task of abstractive long text summarization with very promising results. This model is also capable of generating paragraph and document representation. Extensive experiments on two different benchmark datasets show that our model achives improvements over strong baseline models. As part of our future work, attention mechanism could be added to the model to achieve better performance, and more LSTM layers could be applied to understand the sentence better. In other direction for our work is considering to combine residue learing with LSTM, to achieve a deep RNN for better understanding of sentences.

References

1. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
2. Michele Banko, Vibhu O Mittal, and Michael J Witbrock. Headline generation based on statistical translation. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 318–325. Association for Computational Linguistics, 2000.
3. Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.
4. Sumit Chopra, Michael Auli, and Alexander M Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98, 2016.
5. Trevor Cohn and Mirella Lapata. Sentence compression beyond word deletion. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 137–144. Association for Computational Linguistics, 2008.
6. Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.
7. Misha Denil, Alban Demiraj, and Nando de Freitas. Extraction of salient sentences from labelled documents. *arXiv preprint arXiv:1412.6815*, 2014.
8. Günes Erkan and Dragomir R Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22:457–479, 2004.

9. Katja Filippova and Yasemin Altun. Overcoming the lack of parallel data in sentence compression. In *EMNLP*, pages 1481–1491, 2013.
10. Baotian Hu, Qingcai Chen, and Fangze Zhu. Lcsts: A large scale chinese short text summarization dataset. *arXiv preprint arXiv:1506.05865*, 2015.
11. Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. A hierarchical neural autoencoder for paragraphs and documents. *arXiv preprint arXiv:1506.01057*, 2015.
12. Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop*, volume 8. Barcelona, Spain, 2004.
13. Pengfei Liu, Xipeng Qiu, Xinchi Chen, Shiyu Wu, and Xuanjing Huang. Multi-timescale long short-term memory neural network for modelling sentences and documents. In *EMNLP*, pages 2326–2335, 2015.
14. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. word2vec, 2014.
15. Joel Neto, Alex Freitas, and Celso Kaestner. Automatic text summarization using a machine learning approach. *Advances in Artificial Intelligence*, pages 205–215, 2002.
16. G PadmaPriya and K Duraiswamy. An approach for text summarization using deep learning algorithm. *Journal of Computer Science*, 10(1):1–9, 2014.
17. Aaditya Prakash, Sadid A Hasan, Kathy Lee, Vivek Datla, Ashequl Qadir, Joey Liu, and Oladimeji Farri. Neural paraphrase generation with stacked residual lstm networks. *arXiv preprint arXiv:1610.03098*, 2016.
18. Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.
19. Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Interspeech*, pages 194–197, 2012.
20. Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.
21. Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.
22. Kam-Fai Wong, Mingli Wu, and Wenjie Li. Extractive summarization using supervised and semi-supervised learning. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 985–992. Association for Computational Linguistics, 2008.
23. Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.
24. Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.