An Operation Network for Abstractive Sentence Compression

Naitong Yu Jie Zhang[*] Minlie Huang[†] Xiaoyan Zhu
State Key Laboratory of Intelligent Technology and Systems
Tsinghua National Laboratory for Information Science and Technology
Dept. of Computer Science and Technology, Tsinghua University, Beijing, PR China
[*]Microsoft Search Technology Center
ynt12@mails.tsinghua.edu.cn zhanjie@microsoft.com
aihuang@tsinghua.edu.cn zxy-dcs@tsinghua.edu.cn

Abstract

Sentence compression condenses a sentence while preserving its most important contents. Delete-based models have the strong ability to delete undesired words, while generate-based models are able to reorder or rephrase the words, which are more coherent to human sentence compression. In this paper, we propose Operation Network, a neural network approach for abstractive sentence compression, which combines the advantages of both delete-based and generate-based sentence compression models. The central idea of Operation Network is to model the sentence compression process as an editing procedure. First, unnecessary words are *deleted* from the source sentence, then new words are either *generated* from a large vocabulary or *copied* directly from the source sentence. A compressed sentence can be obtained by a series of such edit operations (*delete*, *copy* and *generate*). Experiments show that Operation Network outperforms state-of-the-art baselines.

1 Introduction

Sentence compression is the natural language generation (NLG) task of condensing a sentence while preserving its most important contents. It can also be viewed as a sentence-level summarization task. With the rapid growth of the web contents in recent years, summarization techniques such as sentence compression are becoming more and more important since these techniques can greatly reduce the information overload on the web. Sentence compression can benefit a wide range of applications, especially those on mobile devices which have restricted screen spaces. Sentence compression models can be broadly classified into two categories: delete-based models and abstractive models. Delete-based approaches remove unimportant words from the source sentence and generate a shorter sentence by stitching the remaining fragments together. On the contrary, abstractive models consider operations beyond word deletion, such as reordering, substitution and insertion. Abstractive sentence compression models produce a reform of the source sentence from scratch, thus the results produced by abstractive sentence compression models are more expressive. Obviously, abstractive sentence compression is much harder than delete-based sentence compression, since it needs deeper understanding of the source sentence.

Delete-based sentence compression treats the task as a word deletion problem: given an input *source* sentence x $= x_1, x_2, ..., x_n$ (where x_i stands for the ith word in the sentence x), the goal is to produce a *target* sentence by removing any subset of words in the source sentence x (Knight and Marcu, 2002). Delete-based sentence compression has been widely explored across different modeling paradigms, such as noisy-channel model (Knight and Marcu, 2002; Turner and Charniak, 2005), large-margin learning (McDonald, 2006; Cohn and Lapata, 2007), integer linear programming (Clarke and Lapata, 2008) and variational auto-encoder (Miao and Blunsom, 2016). In delete-based sentence compression models, only *delete* operations are allowed, thus the order of the remaining words can not be changed. These constraints make delete-based sentence compression a relatively easier task. However, in spite of the strong ability of deleting undesired words, delete-based models are not able to rephrase the words, which is far

[†]Corresponding author: Minlie Huang (aihuang@tsinghua.edu.cn)

from human sentence compression. For example, in human sentence compression, the word "remove" can replace the phrase "get rid of" under some particular circumstance, but this substitution can not be accomplished by delete-only models.

Abstractive sentence compression has the ability to reorder, substitute words or rephrase, thus it is more coherent to human sentence compression. Due to the difficulty of abstractive sentence compression, there was only a limited number of work on the task (Cohn and Lapata, 2008; Cohn and Lapata, 2013; Galanis and Androutsopoulos, 2011; Coster and Kauchak, 2011a). However, with the recent success of the sequence-to-sequence (Seq2Seq) model, the task of abstractive sentence compression has become viable. Seq2Seq has an encoder-decoder architecture where the encoder encodes the input sequence into hidden states, and the decoder then generates the output sequence from the hidden states. The attention mechanism (Bahdanau et al., 2014), which can align the output sequence with the input sequence automatically, boosts the performance of Seq2Seq significantly. A number of abstractive sentence compression work has been built upon the Seq2Seq architecture with attention mechanism, such as (Chopra et al., 2016; Wubben et al., 2016; Nallapati et al., 2016; See et al., 2017). These abstractive models (which will be termed *generate-based* models hereafter) have the ability to reorder words or rephrase. However, none of these models consider explicit word deletion. As Coster and Kauchak (2011b) pointed out, deletion is a frequently occurring phenomena in sentence compression dataset. Coster and Kauchak (2011a) imposed delete operation on their sentence compression model and improved the performance significantly. Thus, deletion is also very important for abstractive sentence compression task.

Inspired by previous work, we propose an Operation Network for abstractive sentence compression, which combines the advantages of both delete-based and generated-based sentence compression. The central idea is to model the sentence compression process as an editing procedure. We not only enable the model with a strong ability of word deletion inspired from delete-based models, but also endow the model with the ability of word reordering and word rephrasing inspired from generate-based models.

With a series of editing operations, the source text can be transformed into a condensed version of itself. There are three kinds of editing operations in our model: *delete*, *copy* and *generate*. Given a source text as input, first the *delete* operations remove unnecessary words from the source text yet retain the important content. Then, the summary is constructed by the *copy* and *generate* operations. *Copy* operations duplicate words directly from the selected source text, while *generate* operations produce words which are not in the source texts. Our model is built upon the Seq2Seq framework, and can be trained in an end-to-end fashion.

To summarize, the contributions of this paper are as follows:

- We propose Operation Network, a neural framework for abstractive sentence compression, which models the sentence compression task as a series of editing operations.
- Our Operation Network combines the advantages of both delete-based and generate-based sentence compression. The model is equipped with a strong ability of not only word deletion, but also word reordering and word rephrasing. Experiments show that our model outperforms the baselines.

2 Task Definition

We formulate the problem of abstractive sentence compression discussed in this paper as follows: Given a source sentence $X = (x_1, x_2, ..., x_n)$ as input, the goal is to generate a target sentence $Y = (y_1, y_2, ..., y_m)$, $m < n$, which is a condensed version of X with good readability and retains the most important information in X. Essentially, the model estimates the conditional probability as follows:

$$P(\boldsymbol{Y}|\boldsymbol{X}) = \prod_{t=1}^{m} P(y_t|y_{<t}, \boldsymbol{X}). \tag{1}$$

3 Background: Seq2Seq Model

Our model is built upon the general Seq2Seq model (Sutskever et al., 2014), which is also called the encoder-decoder model. The model consists of an encoder and a decoder. The encoder takes as input a source sequence $\boldsymbol{X} = (x_1, x_2, ..., x_n)$, and outputs a sequence of hidden states $\boldsymbol{h} = (h_1, h_2, ..., h_n)$. The decoder takes $\boldsymbol{h}$ as input and outputs the target sequence $\boldsymbol{Y} = (y_1, y_2, ..., y_m)$.

In both encoder and decoder, we use gated recurrent unit (GRU) (Cho et al., 2014; Chung et al., 2014) as the basic unit. We use a bi-directional RNN (Schuster and Paliwal, 1997) as encoder. A bi-directional RNN contains two distinct RNNs, a forward RNN and a backward RNN. Given the input $\boldsymbol{X} = (x_1, x_2, ..., x_n)$ and the embedding lookup table e, the forward RNN reads the input in the left-to-right direction, resulting a sequence of forward hidden states $\overrightarrow{\boldsymbol{h}} = (\overrightarrow{h}_1, \overrightarrow{h}_2, ..., \overrightarrow{h}_n)$, where $\overrightarrow{h}_t = \text{GRU}(\overrightarrow{h}_{t-1}, e(x_t))$. Similarly, the backward RNN reads the input in the reversed direction and outputs $\overleftarrow{\boldsymbol{h}} = (\overleftarrow{h}_1, \overleftarrow{h}_2, ..., \overleftarrow{h}_n)$. At each time step, we concatenate the hidden states of the corresponding forward and backward RNNs and obtain the encoder hidden states $\boldsymbol{h} = (h_1, h_2, ..., h_n)$, where $h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]$, $t = 1, ..., n$ and $[A; B]$ denotes vector concatenation.

In the decoder for general Seq2Seq model, another GRU is used for updating the decoder hidden states. Given a context vector c_t and the previously decoded word y_{t-1}, the decoder hidden states are updated as follows:

$$s_t = \text{GRU}(s_{t-1}, [c_t; e(y_{t-1})]) \tag{2}$$

The context vector c_t is designed to dynamically attend on key information of the input sentence during the decoding procedure (Bahdanau et al., 2014). c_t is calculated as a weighted sum of the encoder hidden states:

$$c_t = \sum_{k=1}^{n} \alpha_{tk} h_k \tag{3}$$

$\boldsymbol{\alpha}_t = (\alpha_{t1}, \alpha_{t2}, ..., \alpha_{tn})$ is called the *attention distribution*. It can be viewed as a probability distribution over the source words, which tells the decoder where to attend to produce the next word. The attention distribution is calculated as follows:

$$\alpha_{tk} = \text{softmax}(e_{tk}) = \frac{\exp(e_{tk})}{\sum_{j=1}^{n} \exp(e_{tj})} \tag{4}$$

$$e_{tk} = v_a^\top \tanh(W_a s_{t-1} + U_a h_k + b_a) \tag{5}$$

where v_a, W_a, U_a and b_a are trainable parameters.

The vocabulary distribution P_t^{voc} at time step t is calculated as follows:

$$P_t^{voc} = \text{softmax}(W_o'(W_o[s_t; c_t] + b_o) + b_o') \tag{6}$$

where W_o', W_o, b_o and b_o' are trainable parameters. P_t^{voc} is a probability distribution over all words in the vocabulary V, and the output word y_t at time step t is calculated as follows:

$$y_t = \arg\max_w P_t^{voc}(w), w \in V \tag{7}$$

During training, the loss for time step t is the negative log likelihood of the target word y_t^*: $\text{loss}_t = -\log P_t^{voc}(y_t^*)$. The overall loss function for the whole word sequence is: $\text{loss} = \frac{1}{T_Y} \sum_{t=1}^{T_Y} \text{loss}_t$, where T_Y is the length of the target word sequence $\boldsymbol{Y}$.

4 Operation Network

In this section, we introduce our model, called as the Operation Network, to address the task of abstractive sentence compression. In Operation Network, we consider the transformation from the source sentence to the target sentence as a series of operations. We use three kinds of different operations: delete, copy and generate. To enable the model to perform these three types of edit operations, we use two distinct decoders, one for delete and the other for copy and generate. Specifically, we use the same encoder as in the general Seq2Seq model, but employ a delete decoder and a copy-generate decoder. After the input sentence is encoded by the encoder, the delete decoder will first delete unnecessary words from the input sentence, then the copy-generate decoder will produce the output sentence either by copying from the choices of the delete decoder, or generating from a fixed vocabulary. The overall architecture of our model is shown in Figure 1.

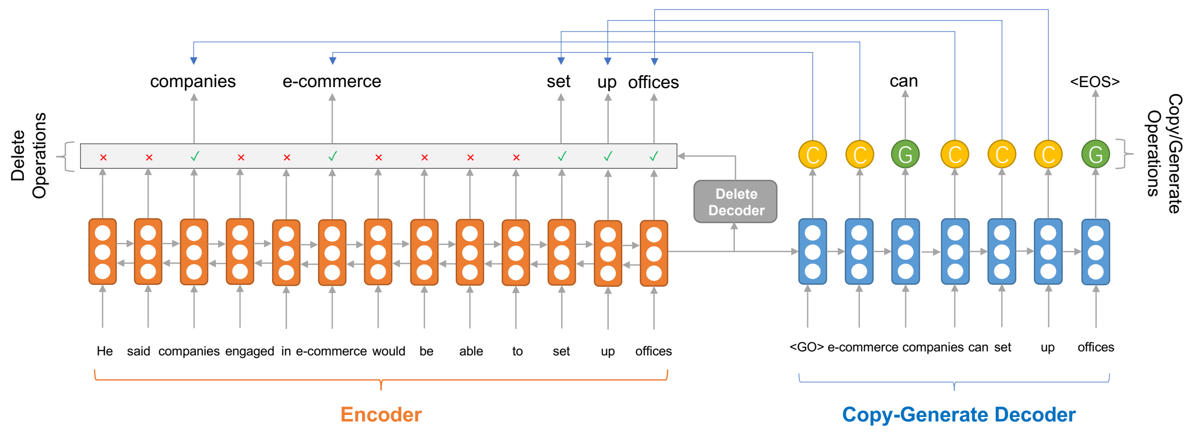

Figure 1: Operation Network, which consists of an encoder, a delete decoder and a copy-generate decoder.

4.1 Delete Decoder

The delete decoder "deletes" all unimportant words from the sentence. It takes as input the sequence of the encoder hidden states $\boldsymbol{h} = (h_1, h_2, ..., h_n)$, the text sequence $\boldsymbol{X} = (x_1, x_2, ..., x_n)$, and outputs a sequence $\boldsymbol{d} = (d_1, d_2, ..., d_n)$, which has the same length n. $d_i \in [0, 1], i \in 1, 2, ..., n$. If d_i is close to 0, then the corresponding word x_i tends to be deleted. Otherwise if d_i is close to 1, then the corresponding word x_i tends to be kept. The output sequence $\boldsymbol{d}$ will be used together with the copy-generate decoder which we will discuss in Section 4.2. The architecture of the delete decoder is shown in Figure 2.

For decoding step t in the delete decoder, we feed the decoder with the embedding of word x_t, the previous decoder output d_{t-1} and the context vector c_t. The delete decoder states are updated as follows:

$$s_t = \text{GRU}(s_{t-1}, [c_t; e(x_t); d_{t-1}]) \tag{8}$$

$$d_t = \sigma(W_d s_t^d + b_d) \tag{9}$$

where σ is the sigmoid function, c_t is the context vector calculated in the same way as Equation 3, e denotes the word embedding table, and W_d and b_d are trainable parameters.

The output of the delete decoder, $\boldsymbol{d} = (d_1, d_2, ..., d_n)$, is fed into the copy-generate decoder for further calculation.

4.2 Copy-Generate Decoder

The Copy-Generate decoder produces output the compressed sentence word by word, and the output words are either copied from the input words which are filtered by the delete decoder, or generated with a fixed vocabulary. In other words, our model integrates copy, generate, delete operations together to produce the output sequence.

We implement the Copy-Generate decoder as a hybrid network between the basic Seq2Seq network and a pointer network (Vinyals et al., 2015). The structure of the Copy-Generate decoder is close to CopyNet (Gu et al., 2016), Pointer Softmax (Gulcehre et al., 2016) and Pointer-Generator (See et al., 2017). However, the Copy-Generate decoder model has some unique characteristic designed for abstractive sentence compression task. The major difference is that our Copy-Generate decoder incorporates the result of the delete decoder, to make sure that unnecessary words will not be copied back by accident. Another difference from the other models is that our model generates explicit operations. During the

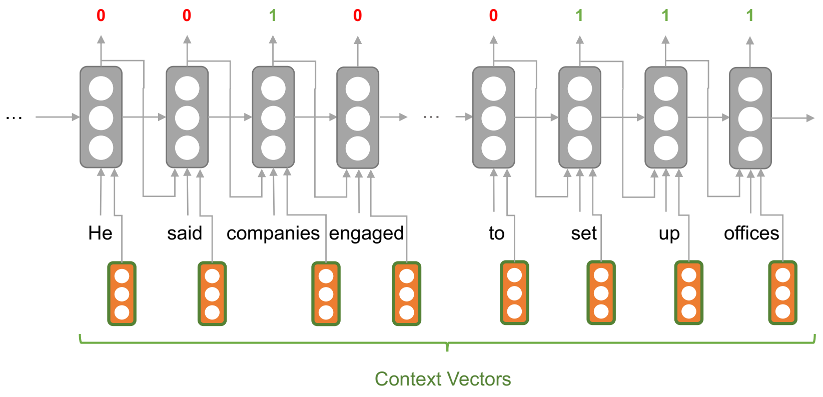

Figure 2: Delete Decoder.

training procedure, we also add supervision on these operations (see Section 4.3). We find that explicit operation supervision can lead to better results. Last, unlike Gu et al. (2016) and Gulcehre et al. (2016), where copy operations are only used for handling UNK tokens (special tokens which stand for out-of-vocabulary words) or named entities, in our model, copy operations are triggered much frequently as long as the output word can be copied from the input sentence. This is useful especially when the training dataset is relatively small.

The Copy-Generate decoder takes as input the encoder hidden states $\boldsymbol{h} = (h_1, h_2, ..., h_n)$ and the attention distributions $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_n)$, and outputs the target sentence as a sequence of words $\boldsymbol{Y} = (y_1, y_2, ..., y_m)$. At each step of decoding, a switch network is used to decide whether to use copy mode or generate mode. If copy mode is chosen, the word is copied directly from the source words (filtered by the delete decoder). We use the attention distributions to sample the source words. The output of the delete decoder is used to modify the attention distributions to filter out unwanted words. Otherwise, if generate mode is chosen, the word is generated from the fixed vocabulary. Specifically, for words that are neither in the source sentence nor in the vocabulary, the switch network will choose generate mode and UNK tokens will be produced from the decoder.

The Copy-Generate decoder use another distinct GRU to update its decoder hidden states as follows:

$$s'_t = \mathrm{GRU}(s'_{t-1}, [c_t; e(y_{t-1})]) \tag{10}$$

where s'_t is the decoder hidden state at time step t, c_t is the context vector, e denotes the word embedding table, and y_{t-1} is the previous decoded output.

At each time step t, the switch network calculates *generate probability* as follows:

$$g_t^{gen} = \sigma(W_z[s'_t; c_t] + b_z) \tag{11}$$

g_t^{gen} acts as a soft switch to choose between generate mode and copy mode.

In generate mode, the target word is sampled from the vocabulary distribution P_t^{voc} (see Equation 6).

In copy mode, first we use the outputs of the delete decoder $\boldsymbol{d}$ (see Section 4.1) to mask and re-normalize the attention distribution $\boldsymbol{\alpha_t}$:

$$\boldsymbol{\alpha'_t} = \frac{1}{\sum_{i=1}^{n} d_i \alpha_{ti}} (d_1\alpha_{t1}, d_2\alpha_{t2}, ..., d_n\alpha_{tn}) = (\alpha'_{t1}, \alpha'_{t2}, ..., \alpha'_{tn}). \tag{12}$$

Then the target word is sampled from the modified attention distribution $\boldsymbol{\alpha'_t}$.

For each sentence, let the *extended vocabulary* V' denotes the union of the vocabulary V and all words appearing in the source sentence, then the final probability distribution over the extended vocabulary V' is:

$$P_t(w) = g_t^{gen} P_t^{voc}(w) + (1 - g_t^{gen}) \sum_{k:w=x_k} \alpha'_{tk}, w \in V' \tag{13}$$

where x_k denotes the kth words in the source sequence $\boldsymbol{X}$. The output word y_t at time step t is sampled from the final distribution P_t:

$$y_t = \arg\max_{w} P_t(w), w \in V' \tag{14}$$

4.3 Loss Function

The total loss consists of two parts: the loss from delete decoder and the loss from copy-generate decoder.

4.3.1 Delete Decoder Loss

The delete decoder loss measures the difference between the target deletion sequence and the actual deletion sequence predicted by the delete decoder. The delete decoder loss is calculated as follows:

$$delete\_loss = \frac{1}{T_X} \sum_{t=1}^{T_X} \Big(-\hat{d}_t \log d_t - (1 - \hat{d}_t) \log(1 - d_t) \Big) \tag{15}$$

where T_X is the length of the source word sequence X, and $\hat{\boldsymbol{d}}$ is the target deletion sequence, $\hat{d}_t = 0$ if word x_t should be deleted, otherwise, if word x_t should be kept, $\hat{d}_t = 1$.

4.3.2 Copy-Generate Decoder Loss

At each time step t, the copy-generate decoder loss can be divided into three parts: the switch loss, the copy loss and the generate loss. The switch loss measures the difference between the target switch and the predicted switch by the model. The copy loss measures the difference between the target attention distribution and the actual attention distribution predicted by the model. And the generate loss measures the loss between the target vocabulary distribution and the generated vocabulary distribution at that time step. The losses at time step t is calculated as follows:

$$\begin{aligned} switch\_loss_t = & - \hat{g}_t^{gen} \log g_t^{gen} \\ & - (1 - \hat{g}_t^{gen}) \log(1 - g_t^{gen}) \end{aligned} \tag{16}$$

$$copy\_loss_t = - (1 - \hat{g}_t^{gen}) \log \sum_{k:y_t^*=x_k} \alpha'_{tk} \tag{17}$$

$$gen\_loss_t = - \hat{g}_t^{gen} \log P_t^{voc}(y_t^*) \tag{18}$$

where $\hat{g}_t^{gen} = 1$ if target is in *generate mode* at time step t, otherwise, if target is in *copy mode*, $\hat{g}_t^{gen} = 0$. y_t^* denotes the target word at time step t.

The overall copy-generate loss function for the target compressed sentence is as follows:

$$copy\_gen\_loss = \frac{1}{T_Y} \sum_{t=1}^{T_Y} (switch\_loss_t + copy\_loss_t + gen\_loss_t) \tag{19}$$

where T_Y is the length of the target word sequence $\boldsymbol{Y}$.

5 Experiments

In this section, we introduce the experiments for abstractive sentence compression with our proposed Operation Network. First, we present a brief description of the dataset, and the pre-processing procedure in our experiments. Then, we introduce baselines that are compared with our model. Next, we introduce parameters of our models in the experiments. At last we present and analyze the experiment results.

5.1 Dataset

We adopt the dataset provided by Toutanova et al. (2016) for our experiments. It's a manually-created, multi-reference dataset for sentence and short paragraph compression. It contains 6,169 source texts with multiple compressions (26,423 pairs of source and compressed texts), consisting of business letters, news journals, and technical documents sampled from the Open American National Corpus (OANC[1]). Of all the source texts, 3,769 are single sentences and the rest are 2-sentence short paragraphs. Each pair of the source and compressed text is aligned by the state-of-the-art monolingual aligner Jacana (Yao et al., 2013). The dataset is split into a training set (21,145 pairs), a validation set (1,908 pairs) and a test set (3,370 pairs). The dataset is available online[2].

The alignment information together with the pair of source text and compressed text are used for generating operation sequences for our experiments. Specifically, for each pair of the source and compressed text, a delete operation sequence and a copy/generate sequence are generated. The delete operation sequence is a sequence of *delete*/*retain* tokens which has the same length with the source text. For each word in the source text, if the word exists in the compressed text or is aligned by the aligner, the corresponding token in the delete operation sequence is *retain*. On the other hand, if the word neither exists in the compressed text nor aligned by the aligner, the corresponding token in the delete operation sequence should be *delete*. The copy/generate operation sequence is a sequence of *copy*/*generate* tokens which has the same length with the compressed text. For each word in the compressed text, if the word is aligned by the aligner and is the same as its counterpart, then the corresponding token in the sequence is *copy*, which means this word is copied from the source text. If the word is not aligned or not the same as its counterpart, then the corresponding token in the sequence is set to *generate*, which means this word is generated from the vocabulary. The reason we didn't put *delete*, *copy* and *generate* operations in a single sequence is that *delete* operations are source-text-based operations while *copy* and *generate* operations are target-text-based operations. It is consistent with that the delete operation sequence has the same length as the source text and the copy/generate operation sequence has the same length as the compressed text.

5.2 Baselines

We compared the abstractive sentence compression results generated by our model (Operation Network) with those by two baselines. These baselines consist of a generate-only model (Seq2Seq) and a generate + copy model (Pointer-Generator). The details of the baselines are described as follows:

- **Seq2Seq:** Seq2Seq is an generate-only model similar to the model described by Nallapati et al. (2016). It uses the same bi-directional RNN encoder and attention mechanisms as our model. The decoder of Seq2Seq model can only generate words from the fixed vocabulary.

- **Pointer-Generator:** Pointer-Generator is a model proposed by See et al. (2017). It uses a hybrid pointer-generator network that can copy words from source text or generate words directly from a large vocabulary. This model also use coverage to keep track of what has been summarized, which discourage repetition. Refer to See et al. (2017) for more details.

5.3 Experiment Parameters

For all experiments, we use 300-dimensional word embeddings. We also use the pre-trained *GoogleNews* vectors to initialize the embeddings. For words do not exist in *GoogleNews* vectors, we initialize them randomly. The vocabulary size is set to 20,000. We also use the large vocabulary tricks described by Jean et al. (2014). The sampled vocabulary size is set to 4,096. The hidden state of a single GRU is 256-dimensional. To overcome overfitting, we also imposed dropout on the input, output and state vector of the GRUs. The dropout probability is set to 0.5.

We trained the models on a single Nvidia Titan X GPU with a batch size of 32. During training, we used Stochastic Gradient Descent with an initial learning rate of 0.15, and applied the exponential decay to the learning rate as the training process proceeds. At test time, the output sentences are decoded using beam search with beam size 4.

[1]`http://www.anc.org/data/oanc`

[2]The dataset can be downloaded from the project's website
`https://www.microsoft.com/en-us/research/project/intelligent-editing/`

5.4 Experiment Metric

In the experiments, we compare our model and the baselines with the following metrics.

Compression Ratio: The common assumption in compression research is that the system can make the determination of the optimal compression length. Thus, compression ratios can vary drastically across systems. Different systems can be compared only when they are compressing at similar ratios (Napoles et al., 2011). Compression ratio is defined as:

$$\text{CompRatio} = \frac{\text{\# of tokens in compressed text}}{\text{\# of tokens in source text}} \tag{20}$$

ROUGE: We evaluated our models with the standard ROUGE metric proposed by Lin (2004). ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. It is commonly used for measuring the quality of the summary by comparing computer-generated summaries to reference summaries generated by humans. The basic idea of ROUGE is to count the number of overlapping units such as n-grams, word sequences, and word pairs between computer-generated summaries and the reference summaries. In our experiments, we considered ROUGE-1, ROUGE-2 and ROUGE-L (which respectively measures the word-overlap, bigram-overlap, and longest common sequence between the reference summary and the summary to be evaluated).

BLEU: We also report BLEU scores of the baseline models and Operation Network on the test dataset. BLEU is proposed by Papineni et al. (2002), and is usually used for automatic evaluation of statistical machine translation systems. However, it can also be used for evaluating sentence compression task (Napoles et al., 2011). We use the multi-bleu script[3] for BLEU score calculation.

5.5 Result Analysis

We present the average ratios of the operations in Operation Network's outputs and the human-written references on the test dataset in Figure 3. Note that the delete ratio is calculated as the number of delete operations divide by the number of tokens in *source* text, and the copy and generate ratio is calculated as the number of copy and generate operations divided by the number of tokens in *compressed* text, respectively. From the figure we can see that Operation Network can utilize the operation information to help accomplish the sentence compression task. The average delete ratio in Operation Network is lower than in human reference, since there are multiple references for one source text in the dataset, and the Operation Network learns to delete a token only it is deleted in most of the references. The average copy ratio is higher in Operation Network than in human reference, this may suggest that Operation Network tends to copy words directly from the source text whenever it is possible.

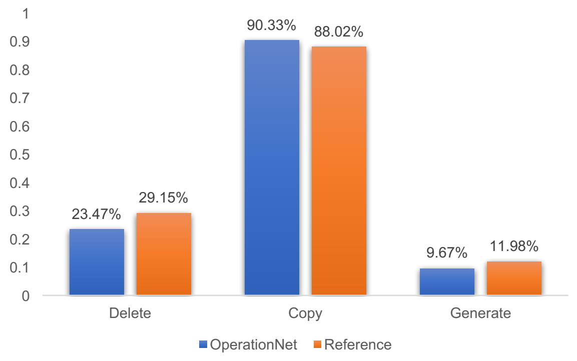

Figure 3: The average ratios of *delete*, *copy* and *generate* operations of Operation Network and human-written references on the test dataset.

Table 1 shows the average compression ratios (%), ROUGE and BLEU scores of the baseline models and Operation Network on the test dataset. The second column shows the average compression ratios.

[3]https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl

The average compression ratio is calculated as the average of the compression ratios of all the compressed text outputs in the test dataset. We can see that the average compression ratios of both our model and the other two baselines are similar. Thus, the comparison between our model and the baseline models is fair. Since we have employed the delete decoder in Operation Network, the average compression ratio of our model is lower than the other baselines.

	CompRatio (%)	ROUGE-1	ROUGE-2	ROUGE-L	BLEU
Seq2Seq	69.39	30.05	10.42	26.87	18.32
Pointer-Generator	66.59	35.34	16.57	32.56	25.09
OperationNet	65.53	**36.21**	**17.43**	**33.72**	**26.30**

Table 1: Average compression ratios (%), ROUGE and BLEU scores of the models on the test dataset. Statistically significant improvements ($p < 0.01$) over the baselines are demonstrated by bold fonts.

From Table 1 we can see that the generate-only model (Seq2Seq) performs poorly compared to its counterparts. This is due to the fact that the Seq2Seq model can only generate words from a fixed vocabulary. When the training dataset is relatively small (about 21k pairs of source and compressed text), the model may not be trained adequately. We tried the same models with random-initialized word embeddings and with *GoogleNews*-vector-initialized word embeddings and it turned out that random-initialized word embeddings lead to worse performance. The copy-and-generate model (Pointer-Generator) performs much better than the generate-only model. This is because the copy operations allow to copy words directly from the source texts and the attention distribution is much smaller than the vocabulary distribution and easier to train. With the addition of delete operation, our model (Operation Network) outperforms the other baselines in terms of all three ROUGE metrics and the BLEU metric. Results show that the delete operations can effectively filter out unnecessary words from the source texts and lead to better performance. This can also show that the Operation Network is suitable for the abstractive sentence compression task.

In order to evaluate the quality of the summaries produced by our model and the other baselines, we ask annotators to do a score evaluation on some aspects of the summaries. Specifically, we ask them to score the grammaticality and non-redundancy of the summaries. Annotators are instructed to read the summary carefully and rate each aspect with scores matching the quality of the corresponding aspect. Each aspect is rated with a score from 0 (bad) to 5 (excellent). We randomly sample 100 samples from the test set for evaluation. Each summary generated by our model or baselines is rated by at least 5 annotators. The summaries are randomly reordered and model information is anonymous to the annotators. The evaluation result is shown in Table 2.

	Grammaticality	Non-Redundancy
Seq2Seq	2.53	2.24
Pointer-Generator	**3.46**	3.58
OperationNet	3.23	**3.64**

Table 2: Results of aspect evaluation.

The results in Table 2 show that combined with the delete decoder, our model can effectively delete unimportant contents without losing much grammar quality of the text. This exactly matches what we expect from our model. Pointer-Generator model performs better on grammaticality aspect than our model, however, our model outperforms the other baselines on non-redundancy aspect. The basic Seq2Seq model performs poorly on both grammaticality aspect and non-redundancy aspect.

6 Related Work

Delete-based sentence compression. A large number of work is devoted to delete-based sentence compression. Jing (2000) presented a system that used multiple sources of knowledge to decide which phrases in a sentence can be removed. Knight and Marcu (2000) proposed statistical approaches to mimic the sentence compression process, they used both noisy-channel and decision-tree to solve the problem. McDonald (2006) presented a discriminative large-margin learning framework coupled with a feature set and syntactic representations for sentence compression. Clarke and Lapata (2006) compared different models for sentence compression across domains and assessed a number of automatic evaluation measures. Clarke and Lapata (2008) used integer linear programming to infer globally optimal compression with linguistically motivated constraints. Berg-Kirkpatrick et al. (2011) proposed a joint model of sentence extraction and compression for multi-document summarization. Filippova and Altun (2013) presented a method for automatically building delete-based sentence compression corpus and proposed an compression method which used structured prediction.

Abstractive sentence compression. Abstractive sentence compression extends delete-based compression methods with additional operations, such as substitution, reordering and insertion. Cohn and Lapata (2008) proposed a discriminative tree-to-tree transduction model which incorporated a grammar extraction method and used a language model for coherent output. Galanis and Androutsopoulos (2011) presented a dataset for extractive and abstractive sentence compression and proposed a SVR based abstractive sentence compressor which utilized additional PMI-based and LDA-based features. Shafieibavani et al. (2016) proposed a word graph-based model which can improve both informativeness and grammaticality of the sentence at the same time.

Neural sentence compression. Filippova et al. (2015) proposed a delete-based sentence compression system which took as input a sentence and output a binary sequence corresponding to word deletion decisions in the sentence. The model was trained on a set of 2 millions sentence pairs which was constructed by the same approach used in Filippova and Altun (2013). There are also some neural approaches for abstractive sentence compression. Rush et al. (2015) proposed a fully data-driven approach which utilized neural language models for abstractive sentence compression. They tried different kinds of encoders to encode the input sentence into vector representation of fixed dimensions. Chopra et al. (2016) further improved the model with Recurrent Neural Networks. However, both works used vocabularies of fixed size for target sentence generation. Wubben et al. (2016) used a Seq2Seq model with bi-directional LSTMs for abstractive compression of captions. Toutanova et al. (2016) manually created a multi-reference dataset for sentence and short paragraph compression and studied the correlations between several automatic evaluation metrics and human judgment.

7 Conclusion

In this paper, we propose Operation Network, a neural approach for abstractive sentence compression, which combines the advantages of both delete-based and generate-based abstractive sentence compression. The central idea of Operation Network is to model the sentence compression process as an editing procedure. With 3 kinds of operations (*delete*, *copy* and *generate*), Operation Network can transform the source sentence into the condensed target. Operation Network is implemented based on the neural Seq2Seq network and pointer network, which consists of a delete decoder and a copy-generate decoder. Given a source sentence as input, first the delete decoder *deletes* unnecessary words from the source sentence, then new words are either *generated* from a large vocabulary or *copied* from the source sentence with the copy-generate decoder. The model is equipped with a strong ability of not only word deletion, but also word reordering and word rephrasing. Experiments show that the model outperforms the baselines.

Acknowledgements

This work was partly supported by the National Science Foundation of China under grant No.61272227/61332007 and the National Basic Research Program (973 Program) under grant No. 2013CB329403.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Taylor Berg-Kirkpatrick, Dan Gillick, and Dan Klein. 2011. Jointly learning to extract and compress. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 481–490. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734. Association for Computational Linguistics.

Sumit Chopra, Michael Auli, and Alexander M Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. *Proceedings of NAACL-HLT16*, pages 93–98.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*. Presented at the Deep Learning workshop at NIPS2014.

James Clarke and Mirella Lapata. 2006. Models for sentence compression: A comparison across domains, training requirements and evaluation measures. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 377–384. Association for Computational Linguistics.

James Clarke and Mirella Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 31:399–429.

Trevor Cohn and Mirella Lapata. 2007. Large margin synchronous generation and its application to sentence compression. In *EMNLP-CoNLL*, pages 73–82.

Trevor Cohn and Mirella Lapata. 2008. Sentence compression beyond word deletion. In *Proceedings of the 22nd International Conference on Computational Linguistics*, pages 137–144. Association for Computational Linguistics.

Trevor Cohn and Mirella Lapata. 2013. An abstractive approach to sentence compression. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 4(3):41.

William Coster and David Kauchak. 2011a. Learning to simplify sentences using wikipedia. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, MTTG '11, pages 1–9, Stroudsburg, PA, USA. Association for Computational Linguistics.

William Coster and David Kauchak. 2011b. Simple english wikipedia: a new text simplification task. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 665–669. Association for Computational Linguistics.

Katja Filippova and Yasemin Altun. 2013. Overcoming the lack of parallel data in sentence compression. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1481–1491. Association for Computational Linguistics.

Katja Filippova, Enrique Alfonseca, Carlos Colmenares, Lukasz Kaiser, and Oriol Vinyals. 2015. Sentence compression by deletion with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Dimitrios Galanis and Ion Androutsopoulos. 2011. A new sentence compression dataset and its use in an abstractive generate-and-rank sentence compressor. In *Proceedings of the UCNLG+Eval: Language Generation and Evaluation Workshop*, pages 1–11. Association for Computational Linguistics.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1631–1640. Association for Computational Linguistics.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 140–149. Association for Computational Linguistics.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*.

Hongyan Jing. 2000. Sentence reduction for automatic text summarization. In *Proceedings of the sixth conference on Applied natural language processing*, pages 310–315. Association for Computational Linguistics.

Kevin Knight and Daniel Marcu. 2000. Statistics-based summarization - step one: Sentence compression. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 703–710. AAAI Press.

Kevin Knight and Daniel Marcu. 2002. Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1):91–107.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop*, volume 8. Barcelona, Spain.

Ryan McDonald. 2006. Discriminative sentence compression with soft syntactic evidence. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 297–304.

Yishu Miao and Phil Blunsom. 2016. Language as a latent variable: Discrete generative models for sentence compression. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, pages 280–290. Association for Computational Linguistics.

Courtney Napoles, Benjamin Van Durme, and Chris Callison-Burch. 2011. Evaluating sentence compression: Pitfalls and suggested remedies. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, pages 91–97. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389. Association for Computational Linguistics.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Abigail See, Christopher Manning, and Peter Liu. 2017. Get to the point: Summarization with pointer-generator networks. In *Association for Computational Linguistics*.

Elaheh Shafieibavani, Mohammad Ebrahimi, Raymond K Wong, Fang Chen, Robert J Durrant, and Kee-Eung Kim. 2016. An efficient approach for multi-sentence compression. In *Proceedings of the 8th Asian Conference on Machine Learning*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Kristina Toutanova, Chris Brockett, Ke M. Tran, and Saleema Amershi. 2016. A dataset and evaluation metrics for abstractive compression of sentences and short paragraphs. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 340–350. Association for Computational Linguistics.

Jenine Turner and Eugene Charniak. 2005. Supervised and unsupervised learning for sentence compression. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 290–297, Stroudsburg, PA, USA. Association for Computational Linguistics.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Neural Information Processing Systems*.

Sander Wubben, Emiel Krahmer, Antal van den Bosch, and Suzan Verberne. 2016. Abstractive compression of captions with attentive recurrent neural networks. In *Proceedings of the 9th International Natural Language Generation Conference*. Association for Computational Linguistics.

Xuchen Yao, Benjamin Van Durme, Chris Callison-Burch, and Peter Clark. 2013. A lightweight and high performance monolingual word aligner. In *ACL*, pages 702–707.