Controlling Output Length in Neural Encoder-Decoders

Yuta Kikuchi[1]
kikuchi@lr.pi.titech.ac.jp

Graham Neubig[2*]
gneubig@cs.cmu.edu

Ryohei Sasano[1]
sasano@pi.titech.ac.jp

Hiroya Takamura[1]
takamura@pi.titech.ac.jp

Manabu Okumura[1]
oku@pi.titecjh.ac.jp

[1]Tokyo Institute of Technology, Japan
[2]Carnegie Mellon University, USA

Abstract

Neural encoder-decoder models have shown great success in many sequence generation tasks. However, previous work has not investigated situations in which we would like to control the length of encoder-decoder outputs. This capability is crucial for applications such as text summarization, in which we have to generate concise summaries with a desired length. In this paper, we propose methods for controlling the output sequence length for neural encoder-decoder models: two decoding-based methods and two learning-based methods.[1] Results show that our learning-based methods have the capability to control length without degrading summary quality in a summarization task.

1 Introduction

Since its first use for machine translation (Kalchbrenner and Blunsom, 2013; Cho et al., 2014; Sutskever et al., 2014), the encoder-decoder approach has demonstrated great success in many other sequence generation tasks including image caption generation (Vinyals et al., 2015b; Xu et al., 2015), parsing (Vinyals et al., 2015a), dialogue response generation (Li et al., 2016a; Serban et al., 2016) and sentence summarization (Rush et al., 2015; Chopra et al., 2016). In particular, in this paper we focus on sentence summarization, which as its name suggests, consists of generating shorter versions of sentences for applications such as document summarization (Nenkova and McKeown, 2011) or headline generation (Dorr et al., 2003). Recently, Rush et al. (2015) automatically constructed large training data for sentence summarization, and this has led to the rapid development of neural sentence summarization (NSS) or neural headline generation (NHG) models. There are already many studies that address this task (Nallapati et al., 2016; Ayana et al., 2016; Ranzato et al., 2015; Lopyrev, 2015; Gulcehre et al., 2016; Gu et al., 2016; Chopra et al., 2016).

One of the essential properties that text summarization systems should have is the ability to generate a summary with the desired length. Desired lengths of summaries strongly depends on the scene of use, such as the granularity of information the user wants to understand, or the monitor size of the device the user has. The length also depends on the amount of information contained in the given source document. Hence, in the traditional setting of text summarization, both the source document and the desired length of the summary will be given as input to a summarization system. However, methods for controlling the output sequence length of encoder-decoder models have not been investigated yet, despite their importance in these settings.

In this paper, we propose and investigate four methods for controlling the output sequence length for neural encoder-decoder models. The former two methods are decoding-based; they receive the desired length during the decoding process, and the training process is the same as standard encoder-decoder models. The latter two methods are learning-based; we modify the network architecture to receive the desired length as input.

[*] This work was done when the author was at the Nara Institute of Science and Technology.

[1] Available at https://github.com/kiyukuta/lencon.

In experiments, we show that the learning-based methods outperform the decoding-based methods for long (such as 50 or 75 byte) summaries. We also find that despite this additional length-control capability, the proposed methods remain competitive to existing methods on standard settings of the DUC2004 shared task-1.

2 Background

2.1 Related Work

Text summarization is one of the oldest fields of study in natural language processing, and many summarization methods have focused specifically on sentence compression or headline generation. Traditional approaches to this task focus on word deletion using rule-based (Dorr et al., 2003; Zajic et al., 2004) or statistical (Woodsend et al., 2010; Galanis and Androutsopoulos, 2010; Filippova and Strube, 2008; Filippova and Altun, 2013; Filippova et al., 2015) methods. There are also several studies of abstractive sentence summarization using syntactic transduction (Cohn and Lapata, 2008; Napoles et al., 2011) or taking a phrase-based statistical machine translation approach (Banko et al., 2000; Wubben et al., 2012; Cohn and Lapata, 2013).

Recent work has adopted techniques such as encoder-decoder (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Cho et al., 2014) and attentional (Bahdanau et al., 2015; Luong et al., 2015) neural network models from the field of machine translation, and tailored them to the sentence summarization task. Rush et al. (2015) were the first to pose sentence summarization as a new target task for neural sequence-to-sequence learning. Several studies have used this task as one of the benchmarks of their neural sequence transduction methods (Ranzato et al., 2015; Lopyrev, 2015; Ayana et al., 2016). Some studies address the other important phenomena frequently occurred in human-written summaries, such as copying from the source document (Gu et al., 2016; Gulcehre et al., 2016). Nallapati et al. (2016) investigate a way to solve many important problems capturing keywords, or inputting multiple sentences.

Neural encoder-decoders can also be viewed as statistical language models conditioned on the target sentence context. Rosenfeld et al. (2001) have proposed whole-sentence language models that can consider features such as sentence length. However, as described in the introduction, to our knowledge, explicitly controlling length of output sequences in neural language models or encoder-decoders has not been investigated.

Finally, there are some studies to modify the output sequence according some meta information such as the dialogue act (Wen et al., 2015), user personality (Li et al., 2016b), or politeness (Sennrich et al., 2016). However, these studies have not focused on length, the topic of this paper.

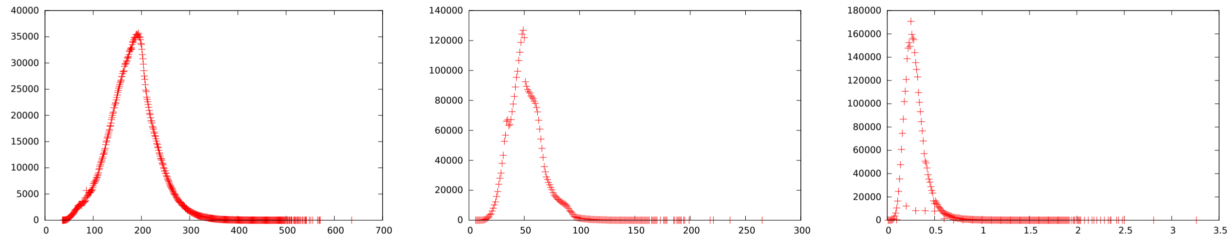
2.2 Importance of Controlling Output Length

As we already mentioned in Section 1, the most standard setting in text summarization is to input both the source document and the desired length of the summary to a summarization system. Summarization systems thus must be able to generate summaries of various lengths. Obviously, this property is also essential for summarization methods based on neural encoder-decoder models.

Since an encoder-decoder model is a completely data-driven approach, the output sequence length depends on the training data that the model is trained on. For example, we use sentence-summary pairs extracted from the Annotated English Gigaword corpus as training data (Rush et al., 2015), and the average length of human-written summary is 51.38 bytes. Figure 1 shows the statistics of the corpus. When we train a standard encoder-decoder model and perform the standard beam search decoding on the corpus, the average length of its output sequence is 38.02 byte.

However, there are other situations where we want summaries with other lengths. For example, DUC2004 is a shared task where the maximum length of summaries is set to 75 bytes, and summarization systems would benefit from generating sentences up to this length limit.

While recent NSS models themselves cannot control their output length, Rush et al. (2015) and others following use an ad-hoc method, in which the system is inhibited from generating the end-of-sentence (EOS) tag by assigning a score of $-\infty$ to the tag and



(a) first sentence (181.87)

(b) article headline (51.38)

(c) ratio (0.30)

Figure 1: Histograms of first sentence length, headline length, and their ratio in Annotated Gigaword English Gigaword corpus. Bracketed values in each subcaption are averages.

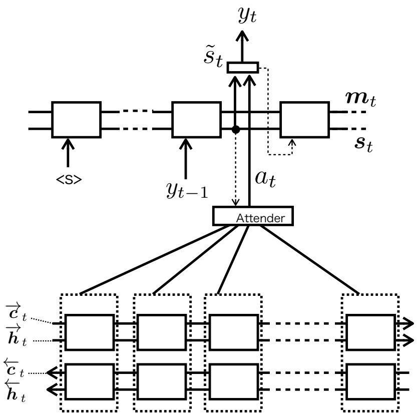


Figure 2: The encoder-decoder architecture we used as a base model in this paper.

generating a fixed number of words[2], and finally the output summaries are truncated to 75 bytes. Ideally, the models should be able to change the output sequence depending on the given output length, and to output the EOS tag at the appropriate time point in a natural manner.

3 Network Architecture: Encoder-Decoder with Attention

In this section, we describe the model architecture used for our experiments: an encoder-decoder consisting of bi-directional RNNs and an attention mechanism. Figure 2 shows the architecture of the model.

Suppose that the source sentence is represented as a sequence of words $\boldsymbol{x} = (x_1, x_2, x_3, ..., x_N)$. For a given source sentence, the summarizer generates a shortened version of the input (i.e. $N > M$), as summary sentence $\boldsymbol{y} = (y_1, y_2, y_3, ..., y_M)$. The model estimates conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$ using parameters trained on large training data consisting of sentence-summary pairs. Typically, this conditional probability is factorized as the product of conditional probabilities of the next word in the sequence:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \prod_{t=1}^{M} p(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x}),$$

where $\boldsymbol{y}_{<t} = (y_1, y_2, y_3, ..., y_{t-1})$. In the following, we describe how to compute $p(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x})$.

3.1 Encoder

We use the bi-directional RNN (BiRNN) as encoder which has been shown effective in neural machine translation (Bahdanau et al., 2015) and speech recognition (Schuster and Paliwal, 1997; Graves et al., 2013).

A BiRNN processes the source sentence for both forward and backward directions with two separate RNNs. During the encoding process, the BiRNN computes both forward hidden states $(\overrightarrow{\boldsymbol{h}}_1, \overrightarrow{\boldsymbol{h}}_2, ..., \overrightarrow{\boldsymbol{h}}_N)$ and backward hidden states $(\overleftarrow{\boldsymbol{h}}_1, \overleftarrow{\boldsymbol{h}}_2, ..., \overleftarrow{\boldsymbol{h}}_N)$ as follows:

$$\overrightarrow{\boldsymbol{h}}_t = g(\overrightarrow{\boldsymbol{h}}_{t-1}, x_t),$$
$$\overleftarrow{\boldsymbol{h}}_t = g(\overleftarrow{\boldsymbol{h}}_{t+1}, x_t).$$

While g can be any kind of recurrent unit, we use long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) networks that have memory cells for both directions ($\overrightarrow{\boldsymbol{c}}_t$ and $\overleftarrow{\boldsymbol{c}}_t$).

[2] According to the published code (https://github.com/facebook/NAMAS), the default number of words is set to 15, which is too long for the DUC2004 setting. The average number of words of human summaries in the evaluation set is 10.43.

After encoding, we set the initial hidden states $\boldsymbol{s}_0$ and memory-cell $\boldsymbol{m}_0$ of the decoder as follows:

$$\begin{aligned} \boldsymbol{s}_0 &= \overleftarrow{\boldsymbol{h}}_1, \\ \boldsymbol{m}_0 &= \overleftarrow{\boldsymbol{c}}_1. \end{aligned}$$

3.2 Decoder and Attender

Our decoder is based on an RNN with LSTM g:

$$\boldsymbol{s}_t = g(\boldsymbol{s}_{t-1}, x_t).$$

We also use the attention mechanism developed by Luong et al. (2015), which uses $\boldsymbol{s}_t$ to compute contextual information $\boldsymbol{d}_t$ of time step t. We first summarize the forward and backward encoder states by taking their sum $\bar{\boldsymbol{h}}_i = \overrightarrow{\boldsymbol{h}}_i + \overleftarrow{\boldsymbol{h}}_i$, and then calculate the context vector $\boldsymbol{d}_t$ as the weighted sum of these summarized vectors:

$$\boldsymbol{d}_t = \sum_i \boldsymbol{a}_{ti} \bar{\boldsymbol{h}}_i,$$

where $\boldsymbol{a}_t$ is the weight at the t-th step for $\bar{\boldsymbol{h}}_i$ computed by a softmax operation:

$$a_{ti} = \frac{\exp(\boldsymbol{s}_t \cdot \bar{\boldsymbol{h}}_i)}{\sum_{\bar{\boldsymbol{h}}'} \exp(\boldsymbol{s}_t \cdot \bar{\boldsymbol{h}}')}.$$

After context vector $\boldsymbol{d}_t$ is calculated, the model updates the distribution over the next word as follows:

$$\begin{aligned} \tilde{\boldsymbol{s}}_t &= \tanh(W_{hs}[\boldsymbol{s}_t; \boldsymbol{d}_t] + \boldsymbol{b}_{hs}), \\ p(y_t | \boldsymbol{y}_{<t}, \boldsymbol{x}) &= \mathrm{softmax}(W_{so}\tilde{\boldsymbol{s}}_t + \boldsymbol{b}_{so}). \end{aligned}$$

Note that $\tilde{\boldsymbol{s}}_t$ is also provided as input to the LSTM with y_t for the next step, which is called the *input feeding* architecture (Luong et al., 2015).

3.3 Training and Decoding

The training objective of our models is to maximize log likelihood of the sentence-summary pairs in a given training set D:

$$L_t(\theta) = \sum_{(\boldsymbol{x},\boldsymbol{y}) \in D} \log p(\boldsymbol{y}|\boldsymbol{x}; \theta),$$

$$p(\boldsymbol{y}|\boldsymbol{x}; \theta) = \prod_t p(y_t | \boldsymbol{y}_{<t}, \boldsymbol{x}).$$

Once models are trained, we use beam search to find the output that maximizes the conditional probability.

4 Controlling Length in Encoder-decoders

In this section, we propose our four methods that can control the length of the output in the encoder-decoder framework. In the first two methods, the decoding process is used to control the output length without changing the model itself. In the other two methods, the model itself has been changed and is trained to obtain the capability of controlling the length. Following the evaluation dataset used in our experiments, we use bytes as the unit of length, although our models can use either words or bytes as necessary.

4.1 $fixLen$: Beam Search without EOS Tags

The first method we examine is a decoding approach similar to the one taken in many recent NSS methods that is slightly less ad-hoc. In this method, we inhibit the decoder from generating the EOS tag by assigning it a score of $-\infty$. Since the model cannot stop the decoding process by itself, we simply stop the decoding process when the length of output sequence reaches the desired length. More specifically, during beam search, when the length of the sequence generated so far exceeds the desired length, the last word is replaced with the EOS tag and also the score of the last word is replaced with the score of the EOS tag (*EOS replacement*).

4.2 $fixRng$: Discarding Out-of-range Sequences

Our second decoding method is based on discarding out-of-range sequences, and is not inhibited from generating the EOS tag, allowing it to decide when to stop generation. Instead, we define the legitimate range of the sequence by setting minimum and maximum lengths. Specifically, in addition to the normal beam search procedure, we set two rules:

- If the model generates the EOS tag when the output sequence is shorter than the minimum length, we discard the sequence from the beam.

- If the generated sequence exceeds the maximum length, we also discard the sequence from the beam. We then replace its last word with the EOS tag and add this sequence to the beam

(EOS replacement in Section 4.1).[3]

In other words, we keep only the sequences that contain the EOS tag and are in the defined length range. This method is a compromise that allows the model some flexibility to plan the generated sequences, but only within a certain acceptable length range.

It should be noted that this method needs a larger beam size if the desired length is very different from the average summary length in the training data, as it will need to preserve hypotheses that have the desired length.

4.3 $LenEmb$: Length Embedding as Additional Input for the LSTM

Our third method is a learning-based method specifically trained to control the length of the output sequence. Inspired by previous work that has demonstrated that additional inputs to decoder models can effectively control the characteristics of the output (Wen et al., 2015; Li et al., 2016b), this model provides information about the length in the form of an additional input to the net. Specifically, the model uses an embedding $e_2(l_t) \in \mathbb{R}^D$ for each potential desired length, which is parameterized by a length embedding matrix $W_{le} \in \mathbb{R}^{D \times L}$ where L is the number of length types. In the decoding process, we input the embedding of the *remaining length* l_t as additional input to the LSTM (Figure 3). l_t is initialized after the encoding process and updated during the decoding process as follows:

$$\begin{aligned} l_1 &= length, \\ l_{t+1} &= \begin{cases} 0 & (l_t - byte(y_t) \leq 0) \\ l_t - byte(y_t) & \text{(otherwise)}, \end{cases} \end{aligned}$$

where $byte(y_t)$ is the length of output word y_t and $length$ is the desired length. We learn the values of the length embedding matrix W_{le} during training. This method provides additional information about the amount of length remaining in the output sequence, allowing the decoder to "plan" its output based on the remaining number of words it can generate.

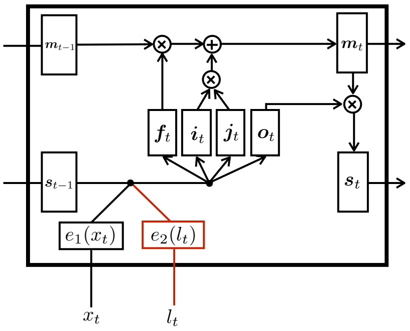


Figure 3: $LenEmb$: *remaining length* is used as additional input for the LSTM of the decoder.

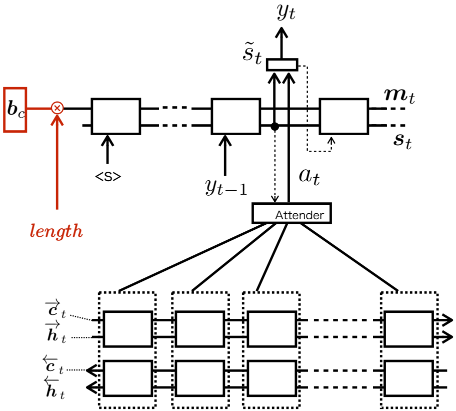


Figure 4: $LenInit$: initial state of the decoder's memory cell m_0 manages output length.

4.4 $LenInit$: Length-based Memory Cell Initialization

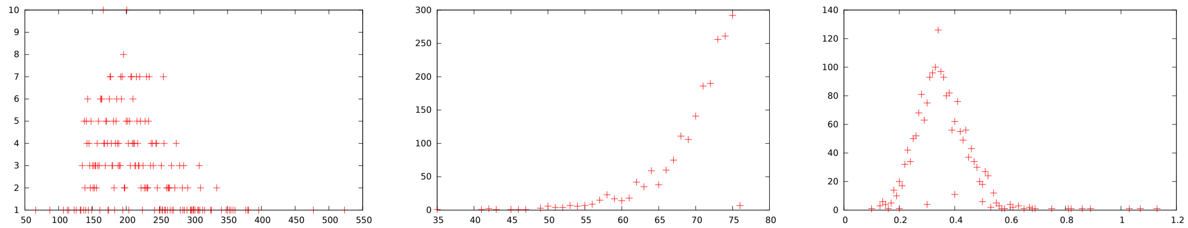
While $LenEmb$ inputs the *remaining length* l_t to the decoder at each step of the decoding process, the $LenInit$ method inputs the desired length once at the initial state of the decoder. Figure 4 shows the architecture of $LenInit$. Specifically, the model uses the memory cell $\boldsymbol{m}_t$ to control the output length by initializing the states of decoder (hidden state $\boldsymbol{s}_0$ and memory cell $\boldsymbol{m}_0$) as follows:

$$\begin{aligned} \boldsymbol{s}_0 &= \overleftarrow{\boldsymbol{h}}_1, \\ \boldsymbol{m}_0 &= \boldsymbol{b}_c * length, \end{aligned} \tag{1}$$

where $\boldsymbol{b}_c \in \mathbb{R}^H$ is a trainable parameter and $length$ is the desired length.

While the model of $LenEmb$ is guided towards the appropriate output length by inputting the remaining length at each step, this $LenInit$ attempts to provide the model with the ability to manage the output length on its own using its inner state. Specifically, the memory cell of LSTM networks is suitable for this endeavour, as it is possible for LSTMs

[3] This is a workaround to prevent the situation in which all sequences are discarded from a beam.



(a) first sentence (206.91) **(b)** summary (70.00) **(c)** ratio (0.35)

Figure 5: Histograms of first sentence length, summary length, and their ratio in DUC2004.

to learn functions that, for example, subtract a fixed amount from a particular memory cell every time they output a word. Although other ways for managing the length are also possible,[4] we found this approach to be both simple and effective.

5 Experiment

5.1 Dataset

We trained our models on a part of the Annotated English Gigaword corpus (Napoles et al., 2012), which Rush et al. (2015) constructed for sentence summarization. We perform preprocessing using the standard script for the dataset[5]. The dataset consists of approximately 3.6 million pairs of the first sentence from each source document and its headline. Figure 1 shows the length histograms of the summaries in the training set. The vocabulary size is 116,875 for the source documents and 67,564 for the target summaries including the beginning-of-sentence, end-of-sentence, and unknown word tags. For $LenEmb$ and $LenInit$, we input the length of each headline during training. Note that we do not train multiple summarization models for each headline length, but a single model that is capable of controlling the length of its output.

We evaluate the methods on the evaluation set of DUC2004 task-1 (generating very short single-document summaries). In this task, summarization systems are required to create a very short summary for each given document. Summaries over the length limit (75 bytes) will be truncated and there is no bonus for creating a shorter summary. The evaluation set consists of 500 source documents and 4 human-written (reference) summaries for each source document. Figure 5 shows the length histograms of the summaries in the evaluation set. Note that the human-written summaries are not always as long as 75 bytes. We used three variants of ROUGE (Lin, 2004) as evaluation metrics: ROUGE-1 (unigram), ROUGE-2 (bigram), and ROUGE-L (longest common subsequence). The two-sided permutation test (Chinchor, 1992) was used for statistical significance testing ($p \leq 0.05$).

5.2 Implementation

We use Adam (Kingma and Ba, 2015) (α=0.001, β_1=0.9, β_2=0.999, eps=10^{-8}) to optimize parameters with a mini-batch of size 80. Before every 10,000 updates, we first sampled 800,000 training examples and made groups of 80 examples with the same source sentence length, and shuffled the 10,000 groups.

We set the dimension of word embeddings to 100 and that of the hidden state to 200. For LSTMs, we initialize the bias of the forget gate to 1.0 and use 0.0 for the other gate biases (Józefowicz et al., 2015). We use Chainer (Tokui et al., 2015) to implement our models. For $LenEmb$, we set L to 300, which is larger than the longest summary lengths in our dataset (see Figure 1-(b) and Figure 5-(b)).

For all methods except $fixRng$, we found a beam size of 10 to be sufficient, but for $fixRng$ we used a beam size of 30 because it more aggressively discards candidate sequences from its beams during decoding.

6 Result

6.1 ROUGE Evaluation

Table 1 shows the ROUGE scores of each method with various length limits (30, 50 and 75 byte). Regardless of the length limit set for the summariza-

[4] For example, we can also add another memory cell for managing the length.

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

	30 byte			50 byte			75 byte		
model	R-1	R-2	R-L	R-1	R-2	R-L	R-1	R-2	R-L
$fixLen$	**14.34**	3.10*	**13.23**	20.00*	5.98	18.26*	25.87*	7.93	23.07*
$fixRng$	13.83*	3.08*	12.88	20.08*	5.74	18.19*	26.01	7.69*	22.77*
$LenEmb_{(0,L)}$	14.23	3.21	13.02	20.78	5.97	18.57	**26.73**	**8.39**	**23.88**
$LenInit_{(0,L)}$	14.31	3.27	13.19	**20.87**	6.16	19.00	25.87	8.27	23.24
$LenEmb_{(0,\infty)}$	13.75	3.30	12.68	20.62	**6.22**	18.64	26.42	8.26	23.59
$LenInit_{(0,\infty)}$	13.92	**3.49**	12.90	20.87	6.19	**19.09**	25.29*	8.00	22.71*

Table 1: ROUGE scores with various length limits. The scores with $*$ are significantly worse than the best score in the column (bolded).

source	five-time world champion michelle kwan withdrew from the #### us figure skating championships on wednesday , but will petition us skating officials for the chance to compete at the #### turin olympics .
reference	injury leaves kwan 's olympic hopes in limbo
$fixLen$ (30)	kwan withdraws from us gp
(50)	kwan withdraws from us skating championships
(75)	kwan pulls out of us figure skating championships for turin olympics
$fixRng$ (30)	kwan withdraws from us gp
(50)	kwan withdraws from figure skating championships
(75)	kwan pulls out of us figure skating championships for turin olympics bid
$LenEmb$ (30)	kwan withdraws from us skating
(50)	kwan withdraws from us figure skating championships
(75)	world champion kwan withdraws from #### olympic figure skating championships
$LenInit$ (30)	kwan quits us figure skating
(50)	kwan withdraws from #### us figure skating worlds
(75)	kwan withdraws from #### us figure skating championships for #### olympics

Table 2: Examples of the output of each method with various specified lengths.

tion methods, we use the same reference summaries. Note that, $fixLen$ and $fixRng$ generate the summaries with a hard constraint due to their decoding process, which allows them to follow the hard constraint on length. Hence, when we calculate the scores of $LenEmb$ and $LenInit$, we impose a hard constraint on length to make the comparison fair (i.e. $LenEmb_{(0,L)}$ and $LenInit_{(0,L)}$ in the table). Specifically, we use the same beam search as that for $fixRng$ with minimum length of 0.

For the purpose of showing the length control capability of $LenEmb$ and $LenInit$, we show at the bottom two lines the results of the standard beam search without the hard constraints on the length[6]. We will use the results of $LenEmb_{(0,\infty)}$ and $LenInit_{(0,\infty)}$ in the discussions in Sections 6.2 and 6.3.

The results show that the learning-based methods ($LenEmb$ and $LenInit$) tend to outperform decoding-based methods ($fixLen$ and $fixRng$) for the longer summaries of 50 and 75 bytes. However, in the 30-byte setting, there is no significant difference between these two types of methods. We hypothesize that this is because average compression rate in the training data is 30% (Figure 1-(c)) while the 30-byte setting forces the model to generate summaries with 15.38% in average compression rate, and thus the learning-based models did not have enough training data to learn compression at such a steep rate.

6.2 Examples of Generated Summaries

Tables 2 and 3 show examples from the validation set of the Annotated Gigaword Corpus. The tables show that all models, including both learning-based methods and decoding-based methods, can often generate well-formed sentences.

We can see various paraphrases of "#### us figure

[6] $fixRng$ is equivalence to the standard beam search when we set the range as $(0, \infty)$.

source	at least two people have tested positive for the bird flu virus in eastern turkey , health minister recep akdag told a news conference wednesday .
reference	two test positive for bird flu virus in turkey
$fixLen$ (30)	two infected with bird flu
(50)	two infected with bird flu in eastern turkey
(75)	two people tested positive for bird flu in eastern turkey says minister
$fixRng$ (30)	two infected with bird flu
(50)	two more infected with bird flu in eastern turkey
(75)	two people tested positive for bird flu in eastern turkey says minister
$LenEmb$ (30)	two bird flu cases in turkey
(50)	two confirmed positive for bird flu in eastern turkey
(75)	at least two bird flu patients test positive for bird flu in eastern turkey
$LenInit$ (30)	two cases of bird flu in turkey
(50)	two people tested positive for bird flu in turkey
(75)	two people tested positive for bird flu in eastern turkey health conference

Table 3: More examples of the output of each method.

championships"[7] and "withdrew". Some examples are generated as a single noun phrase ($LenEmb$(30) and $LenInit$(30)) which may be suitable for the short length setting.

6.3 Length Control Capability of Learning-based Models

Figure 6 shows histograms of output length from the standard encoder-decoder, $LenEmb$, and $LenInit$. While the output lengths from the standard model disperse widely, the lengths from our learning-based models are concentrated to the desired length. These histograms clearly show the length controlling capability of our learning-based models.

Table 4-(a) shows the final state of the beam when $LenInit$ generates the sentence with a length of 30 bytes for the example with standard beam search in Table 3. We can see all the sentences in the beam are generated with length close to the desired length. This shows that our method has obtained the ability to control the output length as expected. For comparison, Table 4-(b) shows the final state of the beam if we perform standard beam search in the standard encoder-decoder model (used in $fixLen$ and $fixRng$). Although each sentence is well-formed, the lengths of them are much more varied.

6.4 Comparison with Existing Methods

Finally, we compare our methods to existing methods on standard settings of the DUC2004 shared task-1. Although the objective of this paper is not to obtain state-of-the-art scores on this evaluation set, it is of interest whether our length-controllable models are competitive on this task. Table 5 shows that the scores of our methods, which are copied from Table 1, in addition to the scores of some existing methods. ABS (Rush et al., 2015) is the most standard model of neural sentence summarization and is the most similar method to our baseline setting ($fixLen$). This table shows that the score of $fixLen$ is comparable to those of the existing methods. The table also shows the $LenEmb$ and the $LenInit$ have the capability of controlling the length without decreasing the ROUGE score.

7 Conclusion

In this paper, we presented the first examination of the problem of controlling length in neural encoder-decoder models, from the point of view of summarization. We examined methods for controlling length of output sequences: two decoding-based methods ($fixLen$ and $fixRng$) and two learning-based methods ($LenEmb$ and $LenInit$). The results showed that learning-based methods generally outperform the decoding-based methods, and the learning-based methods obtained the capability of controlling the output length without losing ROUGE score compared to existing summarization methods.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number JP26280080. We are grateful to have the

[7] Note that "#" is a normalized number and "us" is "US" (United States).

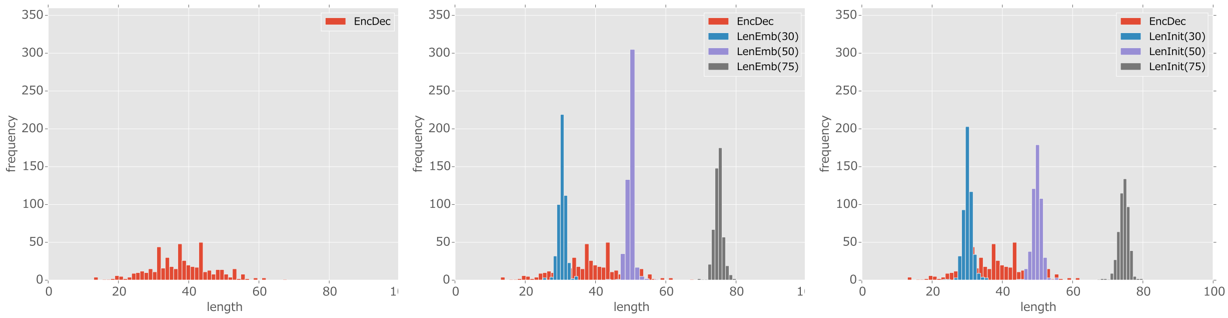
log$p(\boldsymbol{y}\|\boldsymbol{x})$	byte	candidate summary
-4.27	31	two cases of bird flu in turkey
-4.41	28	two bird flu cases in turkey
-4.65	30	two people tested for bird flu
-5.25	30	two people tested in e. turkey
-5.27	31	two bird flu cases in e. turkey
-5.51	29	two bird flu cases in eastern
-5.55	32	two people tested in east turkey
-5.72	30	two bird flu cases in turkey :
-6.04	30	two people fail bird flu virus

(a) the beam of $LenInit$

log$p(\boldsymbol{y}\|\boldsymbol{x})$	byte	candidate summary
-5.05	57	two people tested positive for bird flu in eastern turkey
-5.13	50	two tested positive for bird flu in eastern turkey
-5.30	39	two people tested positive for bird flu
-5.49	51	two people infected with bird flu in eastern turkey
-5.52	32	two tested positive for bird flu
-5.55	44	two infected with bird flu in eastern turkey
-6.00	49	two more infected with bird flu in eastern turkey
-6.04	54	two more confirmed cases of bird flu in eastern turkey
-6.50	49	two people tested positive for bird flu in turkey

(b) the beam of the standard encoder-decoder

Table 4: Final state of the beam when the learning-based model is instructed to output a 30 byte summary for the source document in Table 3.



(a) encoder-decoder **(b)** $LenEmb$ **(c)** $LenInit$

Figure 6: Histograms of output lengths generated by (a) the standard encoder-decoder , (b) $LenEmb$, and (c) $LenInit$. For $LenEmb$ and $LenInit$, the bracketed numbers in each region are the desired lengths we set.

model	R-1	R-2	R-L
$fixLen$	25.88	7.93	23.07
$fixRng$	26.02	7.69	22.78
$LenEmb$	26.73	8.40	23.88
$LenInit$	25.87	8.28	23.25
ABS(Rush et al., 2015)	26.55	7.06	22.05
ABS+(Rush et al., 2015)	28.18	**8.49**	23.81
RAS-Elman(Chopra et al., 2016)	**28.97**	8.26	**24.06**
RAS-LSTM(Chopra et al., 2016)	27.41	7.69	23.06

Table 5: Comparison with existing studies for DUC2004. Note that top four rows are reproduced from Table 1.

opportunity to use the Kurisu server of Dwango Co., Ltd. for our experiments.

References

Ayana, S. Shen, Z. Liu, and M. Sun. 2016. Neural Headline Generation with Minimum Risk Training. *CoRR*, abs/1604.01904.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR15*.

Michele Banko, Vibhu O. Mittal, and Michael J. Witbrock. 2000. Headline generation based on statistical translation. In *Proceedings of ACL00*, pages 318–325.

Nancy Chinchor. 1992. The statistical significance of the muc-4 results. In *Proceedings MUC4 '92*, pages 30–50.

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 EMNLP14*, pages 1724–1734.

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

Trevor Cohn and Mirella Lapata. 2008. Sentence compression beyond word deletion. In *Proceedings of COLING08*, pages 137–144.

Trevor Cohn and Mirella Lapata. 2013. An abstractive approach to sentence compression. *ACM TIST13*, 4(3):41:1–41:35, July.

Bonnie Dorr, David Zajic, and Richard Schwartz. 2003. Hedge trimmer: A parse-and-trim approach to headline generation. In *Proceedings of the HLT-NAACL 03 Text Summarization Workshop*, pages 1–8.

Katja Filippova and Yasemin Altun. 2013. Overcoming the lack of parallel data in sentence compression. In *Proceedings of EMNLP13*, pages 1481–1491.

Katja Filippova and Michael Strube. 2008. Dependency tree based sentence compression. In *Proceedings of INLG08*, pages 25–32.

Katja Filippova, Enrique Alfonseca, Carlos A. Colmenares, Lukasz Kaiser, and Oriol Vinyals. 2015. Sentence compression by deletion with lstms. In *Proceedings of EMNLP15*, pages 360–368.

Dimitrios Galanis and Ion Androutsopoulos. 2010. An extractive supervised two-stage method for sentence compression. In *Proceedings of NAACL-HLT10*, pages 885–893.

A. Graves, N. Jaitly, and A. r. Mohamed. 2013. Hybrid speech recognition with deep bidirectional lstm. In *Proceedings of IEEE Workshop on ASRU13*, pages 273–278.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of ACL16*, pages 1631–1640.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. In *Proceedings of ACL16*, pages 140–149.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *Proceedings of ICML15*, pages 2342–2350.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proceedings of EMNLP13*, pages 1700–1709, Seattle, Washington, USA, October. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of ICLR15*.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016a. A diversity-promoting objective function for neural conversation models. In *Proceedings of NAACL-HLT16*, pages 110–119.

Jiwei Li, Michel Galley, Chris Brockett, Georgios Spithourakis, Jianfeng Gao, and Bill Dolan. 2016b. A persona-based neural conversation model. In *Proceedings of ACL16*, pages 994–1003.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Proceedings of the ACL04 Workshop*, pages 74–81.

Konstantin Lopyrev. 2015. Generating news headlines with recurrent neural networks. *CoRR*, abs/1512.01712.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP15*, pages 1412–1421.

Ramesh Nallapati, Bing Xiang, and Bowen Zhou. 2016. Sequence-to-sequence rnns for text summarization. *CoRR*, abs/1602.06023.

Courtney Napoles, Chris Callison-Burch, Juri Ganitkevitch, and Benjamin Van Durme. 2011. Paraphrastic sentence compression with a character-based metric: Tightening without deletion. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, pages 84–90.

Courtney Napoles, Matthew Gormley, and Benjamin Van Durme. 2012. Annotated gigaword. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, pages 95–100.

Ani Nenkova and Kathleen McKeown. 2011. Automatic summarization. In *Foundations and Trends® in Information Retrieval*, volume 2-3, pages 103–233.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *CoRR*, abs/1511.06732.

Ronald Rosenfeld, Stanley F. Chen, and Xiaojin Zhu. 2001. Whole-sentence exponential language models: a vehicle for linguistic-statistical integration. *Computer Speech & Language*, 15(1):55–73.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of EMNLP15*, pages 379–389.

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

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Controlling politeness in neural machine translation via side constraints. In *Proceedings of NAACL-HLT16*, pages 35–40.

Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of AAAI16*, pages 3776–3784.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of NIPS14*, pages 3104–3112.

Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. 2015. Chainer: a next-generation open source framework for deep learning. In *Proceedings of NIPS15 Workshop on LearningSys*.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2015a. Grammar as a foreign language. In *Proceedings of NIPS15*, pages 2773–2781.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015b. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of EMNLP15*, pages 1711–1721, Lisbon, Portugal, September. Association for Computational Linguistics.

Kristian Woodsend, Yansong Feng, and Mirella Lapata. 2010. Title generation with quasi-synchronous grammar. In *Proceedings of the EMNLP10*, pages 513–523.

Sander Wubben, Antal van den Bosch, and Emiel Krahmer. 2012. Sentence simplification by monolingual machine translation. In *Proceedings of ACL12*, pages 1015–1024.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In David Blei and Francis Bach, editors, *Proceedings of ICML15*, pages 2048–2057. JMLR Workshop and Conference Proceedings.

David Zajic, Bonnie J Dorr, and R. Schwartz. 2004. Bbn/umd at duc-2004: Topiary. In *Proceedings of NAACL-HLT04 Document Understanding Workshop*, pages 112 – 119.