Exploiting Background Knowledge in Compact Answer Generation for Why-Questions

Ryu Iida, Canasai Kruengkrai, Ryo Ishida,
Kentaro Torisawa, Jong-Hoon Oh, and **Julien Kloetzer**
National Institute of Information and Communications Technology, Kyoto, 619-0289, Japan
{ryu.iida, torisawa, rovellia, julien}@nict.go.jp, canasai@gmail.com, ishida.ryo@jp.panasonic.com

Abstract

This paper proposes a novel method for generating compact answers to open-domain why-questions, such as the following answer, "Because deep learning technologies were introduced," to the question, "Why did Google's machine translation service improve so drastically?" Although many works have dealt with why-question answering, most have focused on retrieving as answers relatively long text passages that consist of several sentences. Because of their length, such passages are not appropriate to be read aloud by spoken dialog systems and smart speakers; hence, we need to create a method that generates compact answers. We developed a novel neural summarizer for this compact answer generation task. It combines a recurrent neural network-based encoder-decoder model with stacked convolutional neural networks and was designed to effectively exploit background knowledge, in this case a set of causal relations (e.g., "[Microsoft's machine translation has made great progress over the last few years]$_{effect}$ *since* [it started to use deep learning.]$_{cause}$") that was extracted from a large web data archive (4 billion web pages). Our experimental results show that our method achieved significantly better ROUGE F-scores than existing encoder-decoder models and their variations that were augmented with query-attention and memory networks, which are used to exploit the background knowledge.

1 Introduction

The recent popularity of smart speakers suggests that spoken dialog systems may become a core component of future user interfaces and be used in a wide range of environments for many types of tasks. However, in the current state of language technologies, many such tasks are beyond the reach of dialog systems. One such task is non-factoid Question Answering (QA), such as *why*-question answering (why-QA) and *how to*-question answering. Although many attempts have been made to develop highly accurate non-factoid QA methods (Girju 2003; Higashinaka and Isozaki 2008; Verberne et al. 2011; Oh et al. 2013; Oh et al. 2016; Sharp et al. 2016; Tan et al. 2016; dos Santos et al. 2016; Oh et al. 2017), most developed methods retrieve from a text archive *long text passages that contain real answers* that are not suitable for dialog systems due to their lengths. Table 1 exemplifies a why-question and its answer passage retrieved with an existing why-QA method. Much of the retrieved passages is unnecessary and/or has little to do with the question. Since such redundant and long passages are inappropriate to be read aloud by spoken dialog systems and smart speakers, we need a method that condenses the essence of these answer passages into much shorter, compact answers using only a handful of words.

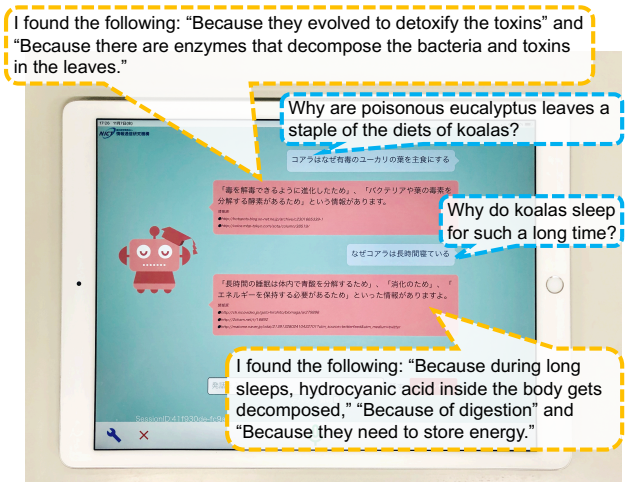

Figure 1: Screenshot of our spoken dialog system prototype WEKDA: why-questions and their compact answers were manually translated.

The aim of this work is to develop a novel method that generates non-redundant compact answers to Japanese open-domain why-questions. Actually, the method presented in this paper is already being used in our spoken dialog system prototype called WEb-based Knowledge Disseminating dialog Agent (WEKDA). The system can smoothly provide compact answers to such why-questions as "Why are poisonous eucalyptus leaves a staple of diets of koalas?" and "Why do koalas sleep for such a long time?" (Figure 1).

As exemplified by the compact answer in Table 1, these answers should be short and comprehensible (e.g., the constituents of each compact answer are limited to 25 Japanese characters in our target dataset) so that they can be read

Question: "Why did Google's machine translation service improve so drastically?"
Answer Passage: Deep learning related **technologies** have recently received much attention, and many high-tech companies are eager to integrate the results of the latest **deep learning** advancements. The Google Brain team, for example, developed such **technologies** that **were** later **introduced** into Google's translation engine. The quality of this machine translation service dramatically improved for languages such as French and Chinese, and the same can be expected in the future for many other languages. Although it remains unclear exactly how much, **deep learning technologies** may drastically change our future.
Compact Answer: Because deep learning technologies were introduced.

In the answer passage, the words in bold should be included in compact answers. The underlined words are the question's content words.

Table 1: Example of a why-question, an answer passage and a compact answer

aloud by spoken dialog systems. We follow the basic set-up of compact why-QA answer generation proposed by Ishida et al. (2018): a *single sentence* compact answer (e.g., "Because deep learning technologies were introduced.") to a given why-question (e.g., "Why did Google's machine translation service improve so drastically?") is generated from a five- or seven-sentence long passage retrieved by an existing web-based why-QA method (Oh et al. 2016). As seen in the example above, this task is complicated because the words that constitute compact answers (those in bold in Table 1) may be scattered throughout a passage and need to be organized into a well-formed sentence. Ishida et al. (2018) treated this as a summarization task and based their method on a state-of-the-art seq2seq model called a *pointer-generator network* (See, Liu, and Manning 2017), in which a long answer passage is given to an encoder and a decoder exploits the passage encoding to generate a compact answer. The content words in the passage that also appear in the question are *labeled* so that the encoder can identify which part of the passage corresponds to the question's content.

In this work, we introduce background knowledge to Ishida et al.'s method in the form of texts representing causal relations (e.g., "[Tsunami was occured]$_{effect}$ *because* [an earthquate suddenly displaced sea water.]$_{cause}$"). When we ask a why-question, we basically expect the answer to be the cause of the events expressed in the question. If we can find the causal relations where the effect part resembles a target why-question, the cause parts may include valuable clues for generating appropriate compact answers, such as important keywords to be included in the compact answers. We thus automatically extract the causal relations relevant to a target why-question from the web, such as "[Microsoft's machine translation has made great progress over the last few years]$_{effect}$ *since* [it started to use deep learning.]$_{cause}$": although this text does not express a direct answer to "Why did Google's machine translation service improve so drastically?", it helps to generate a proper answer since it includes such critical keywords as "deep learning" in its cause part. In the answer passage in Table 1, there are no explicit clue expressions such as "because" and "since" that indicate the causal relations between the part corresponding to the question ("The quality of this machine translation service was dramatically improved.") and the part containing the compact answer ("**Deep learning technologies** $\cdots$ **introduced** into Google's translation engine."). In general, proper answer passages often lack such clue expressions, and to generate proper compact answers, such words as "deep learning" that are causally associated with a given question in causal relations should be helpful.

We propose a novel neural summarizer that utilizes the causal relations extracted from the web as background knowledge. In this summarizer, we assign labels, which are distinguishable from the labels given to the words in a why-question, to the words that appear both in a passage and the cause part of causal relations, such as "deep learning" in the above example, when the passage is given to the encoder. In addition, we extend the summarizer to exploit such causality-related labels more effectively in an encoder-decoder model. We introduce a stacked version of convolutional neural networks (CNNs) (van den Oord et al. 2016; Dauphin et al. 2017; Gehring et al. 2017) that considers the surrounding context of the labeled words in the encoder and aggregate such informative encoding results as the initial decoder state for generating better compact answers. Our experimental results show that our summarizer achieved significantly better ROUGE F-scores than several baseline methods including the existing compact answer generation method (Ishida et al. 2018) and variations augmented with query-attention (Nema et al. 2017) and key-value memory networks (Miller et al. 2016), which are more sophisticated than our method that explicitly assigns labels to words.

The rest of this paper is organized as follows. We provide an overview of related works in Section 2, we present our proposed method in Section 3, and we describe our dataset and experimental results in Section 4.

2 Related Work

We follow the approach proposed in Ishida et al. (2018), in which compact answer generation is regarded as a special case of text summarization. The recent progress of neural network-based techniques, especially the seq2seq framework (Luong, Pham, and Manning 2015; Bahdanau, Cho, and Bengio 2015), has led to a significant improvement in summarization quality (Rush, Chopra, and Weston 2015; Chopra, Auli, and Rush 2016; Nallapati et al. 2016; Chen et al. 2016; Miao and Blunsom 2016; See, Liu, and Manning 2017; Zhou et al. 2017).

A notable advance in neural network-based summarization methods is the use of a *copying* function that copies the words in a source text to a summary (Nallapati et al. 2016; See, Liu, and Manning 2017). For example, See, Liu, and Manning (2017)'s pointer-generator network reduced the risk of generating irrelevant words in a fixed target vocabulary by copying the words in the source text to a summary. Ishida et al. (2018) employed the pointer-generator network as their starting point. We follow their choice.

Another research direction uses more than one input text. Nema et al. (2017) proposed a new attention mechanism called *query attention*, which was first computed for words in an additional input called *query*, and utilized it when computing the attention distribution of words in the source text. In our evaluation in Section 4, we regard background knowledge (i.e., a set of texts representing causal relations) as queries and introduce a variant of this method as a baseline method.

For why-QA, several neural methods (Sharp et al. 2016; Tan et al. 2016; dos Santos et al. 2016; Oh et al. 2017) showed significant performance improvement over methods using conventional machine learning methods such as support vector machines (Girju 2003; Higashinaka and Isozaki 2008; Verberne et al. 2011; Oh et al. 2013; Oh et al. 2016). Oh et al. (2017) used causality expressions that were automatically extracted from the Web for why-QA, like our approach. However, these methods are basically binary classifiers that predict whether a given long passage is a proper answer to a given question and cannot generate compact answers like ours.

Another work that resembles ours is a causality recognition method proposed in Kruengkrai et al. (2017). This neural method exploits web texts and why-QA answers as background knowledge for event causality recognition. But again, it is a binary classifier that judges whether a given sentence expresses event causality.

Finally, in the SQuAD machine comprehension task (Rajpurkar et al. 2016), participating systems identified short consecutive word sequences in passages as answers to questions. In compact answer generation, however, words that comprise the answer may be scattered throughout the answer passage. In this respect, our task differs from the SQuAD task and Ishida et al. (2018) actually showed that a neural network developed for the SQuAD task performed poorly on compact answer generation task. (See Ishida et al. (2018) for more details.)

3 Proposed Method

In this section, we first describe Ishida et al. (2018)'s model and then present our proposed model. We use uppercase to denote sequences (e.g., X, Y), lowercase to denote symbols in a sequence (e.g., x, y), bold uppercase to denote matrices (e.g., $\mathbf{X}$, $\mathbf{H}$) and bold lowercase to denote vectors (e.g., $\mathbf{x}$, $\mathbf{w}$). We respectively denote an answer passage, a target compact answer and a why-question by $X = (x_1, \ldots, x_S)$, $Y = (y_1, \ldots, y_T)$ and $Q = (q_1, \ldots, q_U)$.

3.1 Background: Ishida et al.'s model

Ishida et al. (2018) first represented each word x_s in the answer passage X as a feature vector $\mathbf{x}_s \in \mathbb{R}^d$. The answer passage becomes $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_S] \in \mathbb{R}^{d \times S}$. Then The feature vectors $\mathbf{x}_s (s \in \{1, \cdots, S\})$ are computed as:

$$\mathbf{x}_s = \tanh(\mathbf{W}_q[\mathbf{w}_s; \mathbf{q}_s]). \quad (1)$$

Here $\mathbf{W}_q \in \mathbb{R}^{d \times (d+e)}$ is a trainable matrix, $\mathbf{w}_s \in \mathbb{R}^d$ is a d-dimensional word embedding vector and $\mathbf{q}_s \in \mathbb{R}^e$ is a trainable e-dimensional vector. Note that $\mathbf{q}_s \in \{\mathbf{q}^q, \mathbf{q}^{nq}\}$ where $\mathbf{q}^q$ and $\mathbf{q}^{nq}$ are randomly initialized vectors. $\mathbf{q}_s$ functions as a label that is given to each content word that appears in a given question Q, as mentioned in the Introduction. $\mathbf{q}_s = \mathbf{q}^q$ if a word x_s appears in the question Q and $\mathbf{q}_s = \mathbf{q}^{nq}$ otherwise.

The encoder (a multi-layer bi-directional recurrent neural network(RNN)) then encodes $\mathbf{X}$ into hidden states $\mathbf{H}^{\mathrm{R}} = [\mathbf{h}^{\mathrm{R}}_1, \ldots, \mathbf{h}^{\mathrm{R}}_S] \in \mathbb{R}^{d \times S}$. Each $\mathbf{h}^{\mathrm{R}}_s = [\overrightarrow{\mathbf{h}}_s; \overleftarrow{\mathbf{h}}_s]$ is a concatenation of a forward $\overrightarrow{\mathbf{h}}_s$ and a backward $\overleftarrow{\mathbf{h}}_s$. According to the standard scheme, the decoder (a single-layer unidirectional RNN) initializes its initial hidden state with:

$$\mathbf{s}_0 = \mathrm{ReLU}(\mathbf{W}_r[\overrightarrow{\mathbf{h}_S}; \overleftarrow{\mathbf{h}_1}] + \mathbf{b}_r), \quad (2)$$

where $\mathbf{W}_r \in \mathbb{R}^{d \times d}$ and $\mathbf{b}_r \in \mathbb{R}^d$ are a trainable matrix and a vector.

The decoder is also set up with state-of-the-art techniques, including Bahdanau, Cho, and Bengio (2015)'s attention, copy-attention in See, Liu, and Manning (2017)'s pointer-generator network and Luong, Pham, and Manning (2015)'s input-feeding, all of which greatly strengthen Ishida et al. (2018)'s model. They used long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) but we also examine gated recurrent unit (GRU) (Cho et al. 2014) in our experiments.

The network then learns to minimize the negative conditional log-likelihood of the target compact answer:

$$\mathcal{L}_{\mathrm{R}} = -\sum_{t=1}^{T} \log p(y_t | y_{<t}, X, Q). \quad (3)$$

3.2 Proposed network model

As explained in the introduction, our method first retrieves a set of texts that represents the causal relations in which the effect part resembles a target why-question and labels the words that appear both in an answer passage and the cause parts of the causal relations. We hypothesized that these labels can be exploited more effectively by taking into account their context. For example, a text region that is densely labeled in an answer passage is more likely to be included in a proper compact answer than less densely labeled regions. Also, the combination of some labeled words and the specific syntactic patterns surrounding them may signal that the text regions should be included in a proper compact answer. We expect that such tendencies will be automatically learned by the encoder RNN, but simply adding these labels to Ishida et al.'s method only marginally improved performance in our preliminary experiments. We thus introduced a stacked version of CNNs (van den Oord et al. 2016; Dauphin et al. 2017; Gehring et al. 2017) to our encoder-decoder model to consider the contexts of the labeled words in a manner different to that in the encoder RNN. In the following, we first present how to label the words in an answer passage and then explain our stacked CNNs.

Given a target question Q (e.g., "Why did Google's machine translation service improve so drastically?"), we first retrieve *causal relation expressions* (CEs) that are relevant to Q, like "[Microsoft's machine translation made great progress in the last few years]$_{\textit{effect}}$ *since* [it started to use deep

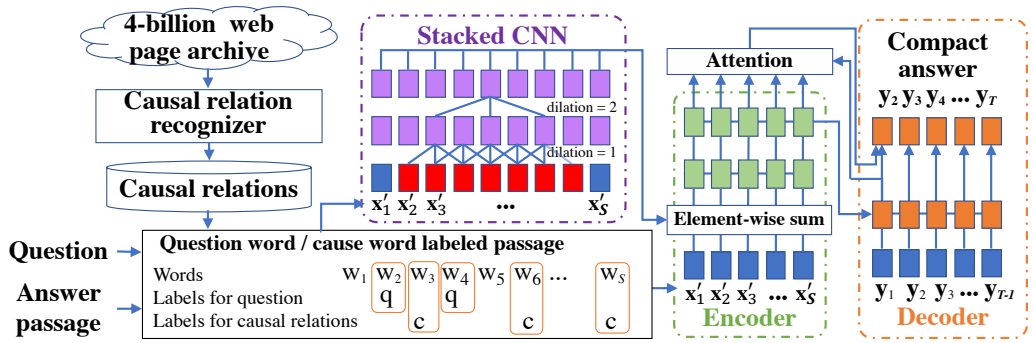

Figure 2: Proposed method

learning.]$_{cause}$." Next we label the content words in an answer passage that match words in the cause parts of the CEs. We newly introduce two trainable d-dimensional randomly initialized vectors $\mathbf{c}^c$ and $\mathbf{c}^{nc}$. To each word x_s, we associate a vector $\mathbf{c}_s \in \{\mathbf{c}^c, \mathbf{c}^{nc}\}$: $\mathbf{c}_s = \mathbf{c}^c$ if the word x_s appears in the cause parts and $\mathbf{c}_s = \mathbf{c}^{nc}$ otherwise. Given the vector $\mathbf{c}_s$, we compute the feature vector $\mathbf{x}'_s$, which represents each word x_s in the answer passage, as:

$$\mathbf{x}'_s = \mathrm{drop}(\mathbf{w}_s + \mathbf{q}_s + \mathbf{c}_s)\,, \tag{4}$$

where drop denotes the dropout operation (Hinton et al. 2012) and $\mathbf{w}_s$ and $\mathbf{q}_s$ are the embedding vectors for w_s and the question label. We use the identical dimension d for the vectors $\mathbf{w}_s, \mathbf{q}_s$ and $\mathbf{c}_s$, unlike in Ishida et al.'s original model. By replacing equation (1) with equation (4), we can avoid introducing an additional parameter, $\mathbf{W}_q$.

We extracted CEs from a 4-billion-page Japanese web text archive. To recognize them in the archive, we applied the conditional random fields (CRF)-based causal recognizer developed by Oh et al. (2013) and obtained about 100 million CEs. This method first looks up one or two consecutive sentences using Japanese clue terms that can be translated as "because" and "reason." Then the CRF judges whether the sentences represent a causal relation; if they do, we identify the regions describing the cause and effect. Following Oh et al. (2017), we indexed the effect parts of the archive CEs using a Lucene search engine[1]. We then extracted the content words (e.g., noun, verb and adjectives) in question Q and retrieved the relevant CEs by matching the content words with the effect parts of all the CEs. We ranked the retrieved CEs by the *tf-idf* score model in the Lucene search engine and used CEs with higher *tf-idf* scores than a predefined threshold θ.

As mentioned above, to consider the contexts of labeled words, we introduce a stacked version of CNNs (van den Oord et al. 2016; Dauphin et al. 2017; Gehring et al. 2017) that combines dilated convolutions (Yu and Koltun 2015) and multi-layer CNNs with residual connections (Gehring et al. 2017). As shown in Figure 2, a stacked CNN is given as inputs the labeled word embeddings of the answer passage and assigns probabilities to the words in the answer passage X that measure how likely they are to appear in the target compact answer Y using both the *labeled* feature vector $\mathbf{x}'_s$ (defined in equation (4)) and the surrounding contexts of the input word w_s and hands these probabilities to Ishida et al.'s encoder as input to the RNNs.

Let us describe the stacked CNNs more precisely. Assume that our stacked CNN has L layers. Then, a feature map $\mathbf{v}^\ell_s \in \mathbb{R}^d (\ell \in \{1, \cdots, L\})$, which corresponds to s-th word in an answer passage X and is computed at the l-th layer, is defined using the feature maps in the $(\ell-1)$-th layer as follows:

$$\mathbf{v}^\ell_s = \left(\mathbf{W}^\ell * [\mathbf{v}^{\ell-1}_{s-2^{\ell-1}}; \mathbf{v}^{\ell-1}_s; \mathbf{v}^{\ell-1}_{s+2^{\ell-1}}] + \mathbf{b}^\ell\right) + \mathbf{v}^{\ell-1}_s\,, \tag{5}$$

where $\mathbf{W}^\ell \in \mathbb{R}^{d\times 3d}$, $\mathbf{b}^\ell \in \mathbb{R}^d$ are trainable parameters and $*$ denotes the convolution operator. The i-th element of the convolutional operator is given by:

$$\mathbf{v}^\ell_s[i] = \langle \mathbf{W}^\ell[i], [\mathbf{v}^{\ell-1}_{s-2^{\ell-1}}; \mathbf{v}^{\ell-1}_s; \mathbf{v}^{\ell-1}_{s+2^{\ell-1}}]\rangle + \mathbf{b}^\ell[i]\,, \tag{6}$$

where $\langle\cdot,\cdot\rangle$ is the Frobenius inner product. $\mathbf{v}^0_s = \mathbf{x}'_s$, which is computed by equation (4). Since $\mathbf{v}^{\ell-1}_{s-2^{\ell-1}}$ and $\mathbf{v}^{\ell-1}_{s+2^{\ell-1}}$ may point to words outside the answer passage, we apply zero-padding by having $\mathbf{v}^{\ell-1}_{s-2^{\ell-1}} = \mathbf{0}$ when $s - 2^{\ell-1} < 1$ and $\mathbf{v}^{\ell-1}_{s+2^{\ell-1}} = \mathbf{0}$ when $s + 2^{\ell-1} > S$.

Note that each $\mathbf{v}^\ell_s$ is computed from just three feature maps: $\mathbf{v}^{\ell-1}_{s-2^{\ell-1}}$, $\mathbf{v}^{\ell-1}_s$ and $\mathbf{v}^{\ell-1}_{s+2^{\ell-1}}$. In the first layer, $\mathbf{v}^1_s$ is computed from $\mathbf{v}^0_{s-1}$, $\mathbf{v}^0_s$ and $\mathbf{v}^0_{s+1}$, which is actually a trigram of the feature vectors of the $(s-1)$-th, s-th and $(s+1)$-th words. As ℓ increases, the gap between $s \pm 2^{\ell-1}$ and s exponentially increases and, at the ℓ-th layer, $\mathbf{v}^\ell_s$ aggregates the information related to the (up to) $2^{\ell+1} - 1$ words surrounding the s-th word.

Finally, the probability vector $\mathbf{p_s} \in \mathbb{R}^2$, which indicates whether a word x_s should appear in the compact answer Y is computed as follows:

$$\mathbf{p_s} = \mathrm{softmax}(\mathbf{W}_p \mathbf{v}^L_s + \mathbf{b}_p)\,, \tag{7}$$

where $\mathbf{W}_p \in \mathbb{R}^{2\times d}$ and $\mathbf{b}_p \in \mathbb{R}^2$ are trainable parameters.

We train the stacked CNN in two-steps. First, we pretrain it independently without considering the encoder and the

[1]http://lucene.apache.org

decoder of the baseline model. At the pretraining step, the stacked CNN learns to minimize the negative log-likelihood of the probability $p_{s,1}$ indicating whether the words in the answer passage should appear in the compact answer:

$$\mathcal{L}_{\mathrm{C}} = -[\sum_{x_s \notin Y} (1-\alpha)\log(p_{s,0}) + \sum_{x_s \in Y} \alpha \log(p_{s,1})], \quad (8)$$

where $\mathbf{p_s} = [p_{s,0}; p_{s,1}]$ and $p_{s,1}$ is the probability of how likely x_s would appear in Y and $p_{s,0} = 1 - p_{s,1}$. α is a weight in the range of $[0, 1]$ that represents a trade-off between precision and recall.

Second, we conduct joint training of the stacked CNN and the baseline model. At the second step, we integrate the pretrained stacked CNNs to our entire architecture (Figure 2) and jointly train the stacked CNNs and the encoder-decoder model as follows. Let $\mathbf{p}_{1:S} = [p_{1,1}, \cdots, p_{S,1}] \in \mathbb{R}^S$ denote the probability vector of how likely x_s would appear in Y. The input embeddings $\mathbf{X}$ to the baseline model encoder are updated as:

$$\tilde{\mathbf{X}} = \mathrm{drop}(\mathbf{X} + \mathbf{u}_x {\mathbf{p}_{1:S}}^\top), \quad (9)$$

where $\mathbf{u}_x \in \mathbb{R}^d$ is a trainable vector. Given a dataset $\mathcal{D} = \{(X_i, Y_i)\}_{i=1}^N$, we jointly train the stacked CNN and Ishida et al.'s model by minimizing the following:

$$\mathcal{L}_{\mathrm{M}} = \frac{1}{N}\sum_{i=1}^{N}[(1-\beta)\mathcal{L}_{\mathrm{C}} + \beta\mathcal{L}_{\mathrm{R}}]. \quad (10)$$

In our experiments, we set α in equation (8) to a relatively large value during the pretraining step, favoring recall, and then change it to a smaller value in the joint training step, favoring precision. The idea is to first train a model that tries to focus on potential candidate words as much as possible and do fine-tuning in the next step, the joint training.

3.3 Decoder initialization with averaging

Thus, we described our proposed method but observed that another trick can boost the performance even further. In Ishida et al.'s method, the decoder input was the final encoder output, which is likely to focus more on the first and last words. On the other hand, we hypothesized that the decoder initial state should contain more information concerning words in the middle of answer passages because our compact answers are short sentences that often start with words located in the middle of a passage. According to this hypothesis, we used the following initial decoder hidden state, which is actually the average of all the encoder hidden states:

$$\mathbf{s}_0 = \frac{1}{S}\sum_s \mathbf{h}_s^{\mathrm{R}}. \quad (11)$$

We refer to this modification as *decoder initialization with averaging* and use it in our experiments. Note that this modification helps reduce the number of parameters to be tuned (i.e., a trainable matrix $\mathbf{W}_r$ is excluded from equation (2)). We expect that this will positively affect performance.

Dataset	#Triples	#Questions
Training set	15,130	2,060
Validation set	2,271	426
Development set	5,920	1,302
Test set	17,315	3,530

Table 2: Number of (question, answer passage, compact answer) triples and questions covered by them in each data set

4 Experiments

4.1 Data

Each of our four datasets (training, validation, development, and test) consists of triples of a why-question, an answer passage and a compact answer. Table 2 provides the number of triples in each. We confirmed that no question was found in multiple datasets. The average length of all the questions, passages and compact answers in all the datasets is 8.0, 184.4, and 9.5 words, respectively. The compact answers are much shorter than the answer passages, suggesting the difficulty of our task.

Basically, we used the datasets used in Ishida et al. (2018), which were built in the following manner. First, human annotators manually wrote *open-domain* why-questions and entered them into a publicly available web-based QA system called WISDOM X (Mizuno et al. 2016), which has an open-domain why-QA module (Oh et al. 2016). The system retrieves answer passages from a 4-billion-page-scale web archive and provides a ranked list of answer passages for each question. Each answer passage consists of five or seven consecutive sentences. For each question, the top 20 answer passages in the ranked list were chosen, and for each passage, three annotators were asked to create a compact and non-redundant answer. Restrictions given to the annotators include (1) no compact answers should exceed 25 Japanese characters in length, and (2) the compact answers' content words have to be included in the passage (see (Ishida et al. 2018) for more details). Annotators did not have to create compact answers for which the passage did not answer the question, resulting in triples without compact answers. They were removed from the training, validation and development sets, but not from the test set.

This discrepancy led to inconsistencies in the results measured for the development and test sets reported in Ishida et al.'s paper. For this reason, we removed from the test set any triple without a compact answer. Then, for more reliable experimental results, we created new triples following the same creation scheme (only using the top three answer passages for each question instead of 20) and added them to the test and development sets. The numbers given in Table 2 include these adjustments to the development and test sets.

4.2 Baseline methods

We compared our method with the following baseline methods.

Ishida-{LSTM,GRU}: This is Ishida et al.'s original method. We used pretrained word embeddings, which

were not used in the original paper, to improve its performance. We also implemented a version of Ishida's method that uses GRUs instead of LSTMs. As shown later, GRUs gave better results than LSTMs in our settings. Although Ishida et al.'s original method used automatically generated training data (AGTD), we did not use them in this comparison. (We compare our method with Ishida's method using AGTD in the end of this section.)

QryAttn: Nema et al. (2017) proposed a query-attention method that gives attention to words based on an additional text input called *query*. We implemented a variant of this query-attention scheme, which can take multiple queries as inputs, on top of the same seq2seq model as **Ishida-GRU** and used the cause part of a causal relation as a query. In this way, since the words in an answer passage that appear in the causal relation receive more attention, they are more likely to be generated in a compact answer.

Let us describe our modified version of the query attention scheme. We first encoded M cause parts of causal relations $\mathrm{CP} = \{\mathrm{CP}_1, \ldots, \mathrm{CP}_M\}$ (same as those we used for the labeling in Section 3.2) into hidden states $\mathbf{H}_i^{\mathrm{CP}} = [\mathbf{h}_{1,i}^{\mathrm{CP}}, \ldots, \mathbf{h}_{N_i,i}^{\mathrm{CP}}] \in \mathbb{R}^{d \times N_i}$ with a multi-layer bi-directional GRU, where N_i denotes the word length of the i-th cause part CP_i. Each $\mathbf{h}_{j,i}^{\mathrm{CP}} = [\overrightarrow{\mathbf{h}}_{j,i}^{\mathrm{CP}}; \overleftarrow{\mathbf{h}}_{j,i}^{\mathrm{CP}}]$ is a concatenation of a forward $\overrightarrow{\mathbf{h}}_{j,i}^{\mathrm{CP}}$ and a backward $\overleftarrow{\mathbf{h}}_{j,i}^{\mathrm{CP}}$. Second, we computed the representation of CP_i as $\mathbf{r}_i^{\mathrm{CP}} = [\overrightarrow{\mathbf{h}}_{N_i,i}^{\mathrm{CP}}; \overleftarrow{\mathbf{h}}_{1,i}^{\mathrm{CP}}]$. Then, we compute the representation $\mathbf{q}_t$ of the M cause parts at the time step t in the decoding process as:

$$a_{t,k}^q = \mathbf{z}^\top \tanh(\mathbf{W}_q \mathbf{s}_t + \mathbf{U}_q \mathbf{r}_k^{\mathrm{CP}}) \quad (12)$$

$$\alpha_{t,k}^q = \frac{\exp\left(a_{t,k}^q\right)}{\sum_m \exp\left(a_{t,m}^q\right)}, \quad (13)$$

$$\mathbf{q}_t = \sum_{k=1}^{M} \alpha_{t,k}^q \mathbf{r}_k^{\mathrm{CP}} \quad (14)$$

where $\mathbf{s}_t$ is the decoder state at time step t and $\mathbf{z} \in \mathbb{R}^{d \times 1}$, $\mathbf{W}_q \in \mathbb{R}^{d \times d}$ and $\mathbf{U}_q \in \mathbb{R}^{d \times d}$ are trainable parameters. Finally, we replaced Nema et al.'s query representation by the representation $\mathbf{q}_t$. Note that this method uses causal relations on the decoder side, unlike our proposed method that uses the relations on the encoder side.

KVMemNN: In this baseline, we integrated a key-value memory network (Miller et al. 2016) into the **Ishida-GRU**. We stored causal relations in the memory network, regarding the effect parts as *keys* and the cause parts as *values*. Our expectation here is that the information about the causal relations stored in the memory is *propagated* to the encoder input (i.e., word embeddings) and the propagated information signals the importance of the words that appear in causal relations so that those words are more likely to be generated in a compact answer.

The details are given in the Appendix. Here we only describe how the output of the memory network modifies the word embeddings. Given a why-question, we first computed the initial query representation $\mathbf{q}_s^0$, which is an input to the memory network, as:

$$\mathbf{q}_s^0 = \mathbf{A}(\sum_i \mathbf{q}_i + \mathbf{x}_s), \quad (15)$$

where $\mathbf{A} \in \mathbb{R}^{d \times d}$ is a trainable matrix and $\mathbf{q}_i$ is the word embedding vector of the i-th word in the question and $\mathbf{x}_s$ is the original input vector to the encoder. Then, the memory network iteratively updates the query representation using the representations of the causal relations stored in the key-value memory. After H iterations, which are called H hops, the updated query representation q_s^H is added to the encoder's input $\mathbf{x}_s$ as:

$$m_s = \sigma\left(\mathbf{W}_1 \mathbf{q}_s^H + b\right), \quad (16)$$

$$\tilde{\mathbf{x}}_s = \mathrm{drop}\left(\mathbf{x}_s + m_s \mathbf{u}_1\right), \quad (17)$$

where σ denotes the sigmoid activation function, drop denotes the dropout operation, and $\mathbf{W}_1 \in \mathbb{R}^{1 \times d}$, $\mathbf{u}_1 \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are trainable parameters. The new vector $\tilde{\mathbf{x}}_s$ is used as the input to the encoder RNN in Ishida et al.'s method instead of the original input vector $\mathbf{x}_s$. We employed this method as a baseline because using causal relations in a more sophisticated method might improve performance more than our simple word labeling on the encoder side.

4.3 Experimental settings

For all the methods, we used the following settings determined through preliminary experiments using the development data. Both the sizes of the word embeddings and the RNN (i.e., GRU and LSTM) hidden states[2] were set to 500. The source and target vocabulary sizes were both set to 50,000. We used Adam (Kingma and Ba 2015) with learning rate of 0.001 and mini-batches of 32 for optimization. If the validation error did not decrease after each epoch, the learning rate was divided by two. We used 1-layer RNNs as a decoder. We independently tried {1,2,3,4}-layer RNNs as encoders and chose the one that led to the optimal ROUGE-1 F-score on the development data. We ran a maximum of 20 epochs and selected the best model of each method based on the perplexity of the validation data. We pretrained the word embedding vectors on Japanese Wikipedia articles[3] using word2vec (Mikolov et al. 2013) and fixed their weight vectors during the training of each method. In the testing step, we used a beam search with a beam size of 3 and unknown word replacement (Jean et al. 2015).

For **Ishida-**{**LSTM,GRU**}, **QryAttn** and **KVMemNN**, we set the size of vectors for the word labels to 32 as done in Ishida et al. (2018). For the stacked CNNs, we used 4-layer CNNs. We set the weight α in equation (8) to 0.9 for the first (pretraining) step and changed it to 0.5 for the second training step. We set the scaling factor β in equation (10) to 0.5.

[2]The hidden state size of each RNN in the bidirectional RNNs was set to 250 and the outputs were simply concatenated.

[3]https://archive.org/details/jawiki-20150118

Method		#Layers	Development			Test		
			R-1	R-2	R-L	R-1	R-2	R-L
Baseline	Ishida-LSTM	1	54.75	40.50	54.16	52.33	37.62	51.68
	Ishida-GRU	2	56.20	42.25	55.60	52.82	38.27	52.17
	QryAttn (tfidf=2.0)	3	56.07	42.24	55.45	52.82	38.23	52.15
	KVMemNN (tfidf=1.5, hop=1)	1	55.31	41.48	54.69	52.63	38.16	51.96
Proposed (tfidf=3.0)		3	57.89	44.62	57.32	54.83*†‡	40.60*†‡	54.17*†‡
Proposed+AvgInit (tfidf=3.0)		3	58.35	45.04	57.78	**55.15***†‡	**40.94***†‡	**54.42***†‡
Proposed+AvgInit+KVMemNN (tfidf=1.5, hop=1)		2	**58.64**	**45.41**	**58.00**	54.71*†‡	40.66*†‡	54.01*†‡
Ishida-LSTM+AGTD (250K)		3	57.34	44.15	56.76	54.51*†‡	40.60*†‡	53.85*†‡

Proposed: our proposed network model. AvgInit: decoder initialization with averaging. #Layers: number of encoder layers. R-1, R-2 and R-L respectively denote ROUGE-1, ROUGE-2 and ROUGE-L F-scores. AGTD: automatically generated training data. Statistically significant improvements over Ishida-GRU, QryAttn and KVMemNN are respectively denoted by '*', '†' and '‡' (Significance was estimated by a 95% confidence interval computed by the ROUGE script).

Table 3: Results of compact answer generation

In **QryAttn**, **KVMemNN**, and our proposed method, we independently evaluated 1.5, 2.0 and 3.0 of the *tf-idf* threshold for causal relation retrieval and selected a threshold to achieve the best ROUGE-1 F-score of the development data. Furthermore, in **KVMemNN**, we independently evaluated {1,2,3,4} of the hop size in the key-value memory network and selected the one that best optimizes the ROUGE-1 F-score of the development data. For word segmentation, we used the morphological analyzer MeCab[4] (Kudo, Yamamoto, and Matsumoto 2004).

4.4 Evaluation results

We evaluated the quality of each method by measuring the averaged ROUGE-1, ROUGE-2, and ROUGE-L F-scores (Lin 2004) and conducted significance test using a 95% confidence interval computed by a ROUGE script[5]. The results of each method obtained for both the development and test data are presented in Table 3. Our proposed method (**Proposed** in Table 3) achieved significantly better ROUGE F-scores than all the baseline methods on the test data. In addition, our method with decoder initialization with averaging (**Proposed+AvgInit**) achieved slightly better ROUGE F-scores than **Proposed**. The improvement over Ishida et al.'s original method (**Ishida-LSTM**) is around 3% in all the ROUGE scores. The improvement over **Ishida-GRU**, which shows slightly better results than **Ishida-LSTM**, exceeded 2%.

In contrast, **QryAttn** and **KVMemNN** gave almost the same or slightly worse results than **Ishida-GRU** from which these methods were modified. This means that, contrary to our method, the more complex query-based summarization technique and key-value memory networks do not effectively exploit background knowledge, at least in our current settings. A possible explanation of these results is that the current encoding schemes for the cause parts in both **QryAttn** and **KVMemNN** were inadequate. **QryAttn** encodes the entire word sequences in the N cause parts using RNNs and generates a single representation vector for each cause part. **KVMemNN** also generates a single representation vector of the cause part through key addressing and value reading processes. In other words, the information in each cause part word sequence in both baselines is "condensed" into a single vector. On the other hand, the word labels are generated from cause parts without such condensation processes. If the condensation process is good enough, the two baselines may outperform the proposed method. However, this was not the case in our experiments.

Actually, a similar observation was obtained in the previous work of Ishida et al. (2018). They tried to give information concerning a given why-question using Nema et al. (2017)'s query-attention mechanism by encoding the entire word sequence in the question into a single representation. However, their experiments revealed that this method does not improve performance. On the other hand, the same previous work showed that the word labels obtained from the question words improved the performance, just like our word labels for causal relations in our work. That is, at least in our compact answer generation task settings, word labels generated from external inputs (i.e., questions and causal relations) consistently improved performance, but **QryAttn** and **KVMemNN** did not. This suggests that the encoding/condensation processes of the cause parts in those baselines were inadequate for our task settings.

Another possible explanation may be that information from the cause parts must somehow be used in the answer passage encoding processes, probably because the answer passage encoder must focus on (possibly fragmented) the text regions of the passages from which a compact answer is generated. In **QryAttn**, the cause part information does not affect the encoding process for the answer passages (i.e., it is used only for attention computation in the decoder), but word labels do affect the encoding process because they are given to the encoder for answer passages alongside the word embeddings of the answer passages. This may be another reason why the improvement by **QryAttn** was not significant.

We also tried a combination of the key-value memory networks and **Proposed+AvgInit**, both of which use background knowledge to modify the encoder input (**Proposed+AvgInit+KVMemNN**). The results on the test set were worse than **Proposed+AvgInit**, although the results

[4]http://taku910.github.io/mecab/

[5]http://www.berouge.com/

Method	#layers	Development			Test		
		R-1	R-2	R-L	R-1	R-2	R-L
Proposed+AvgInit	3	**58.35**	**45.04**	**57.78**	**55.15**	**40.94**	**54.42**
w/o word labels for causal relations	2	57.95	44.76	57.35	54.75	40.66	54.04
w/o stacked CNNs	3	57.40	43.72	56.86	54.62	40.38	53.95
w/o AvgInit (=Proposed)	3	57.89	44.62	57.32	54.83	40.60	54.17

Table 4: Results of ablation test

on the development set were slightly better than it.

We compared our method with Ishida et al. (2018)'s entire method. We obtained from them their auto-generated training data (AGTD), which was automatically generated using the same type of causal relations that we used for labeling words in Section 3.2. We re-trained **Ishida-LSTM** after adding 250K, 500K, 750K, and 1M instances of AGTD to the manually generated training data, and selected the hyper-parameters and AGTD whose size achieved the best ROUGE-1 F-score against the development data. The resulting model's performance is shown as **Ishida-LSTM+AGTD** in the bottom line of Table 3. **Proposed+AvgInit** outperformed **Ishida-LSTM+AGTD** even without AGTD. Note that **Ishida+LSTM+AGTD** achieved the best performance when AGTD was rather small (250K). This implies that using even more AGTD (more than 1M) is unlikely to improve performance. Our results suggest that using background knowledge may be more effective than using AGTD.

Table 4 shows the ablation test results to assess the effectiveness of each component in **Proposed+AvgInit**. All the components introduced in this work improved performance.

Table 5 shows some examples of compact answers that were automatically generated by each method.

Example 1

Question: Why are Japan's trains so crowded?
Answer passage: I'm visiting Japan on a sightseeing trip, but it's really irritating that so many trains are too crowded in Tokyo. It is quite obvious that they are too crowded because many company employees work in central Tokyo while living in the suburbs. I cannot fathom why no political action has alleviated this situation.
Ishida-GRU: Because I'm visiting Japan on a sightseeing trip.
QryAttn: Because sightseers live in the suburbs.
KVMemNN: Because many company employees live in the suburbs.
Proposed+AvgInit: Because many company employees live in the suburbs.

Example 2

Question: Why does bamboo charcoal deodorize smells?
Answer passage: Thank you for buying our "bamboo charcoal pillow." Bamboo charcoal's microscopic pores having absorption capabilities, it provides air purification and deodorization effects and alleviates any discomfort while sleeping. Our pillow will provide you with comfortable sleep.
Ishida-GRU: Because it alleviates discomfort while sleeping.
QryAttn: Because it alleviates discomfort while sleeping.
KVMemNN: Because it has air purification and deodorization effects.
Proposed+AvgInit: Because the microscopic pores has absorption capabilities.

Table 5: Examples of compact answers generated by each method (manually translated in English for readability)

5 Conclusion

We proposed a novel neural method for generating compact answers (e.g., "Because deep learning was introduced.") to why-questions (e.g., "Why did Google's machine translation improve so drastically?") from long multi-sentence text passages that were retrieved by existing why-QA methods. The method exploits such causal relations as "[Microsoft's machine translation has made great progress over the last few years]$_{effect}$ *since* [it started to use deep learning.]$_{cause}$", retrieved from a large web archive of 4 billion web pages. In this method, we first labeled the words that appear in the cause parts in the causal relations (e.g., deep learning) to focus on such words in generating compact answers by an encoder-decoder model. Also, we introduced stacked CNNs to consider the contexts of the labeled words in a manner different to that in the encoder RNN. Our experimental results show that our method achieved significantly better ROUGE F-scores than existing encoder-decoder models and their variations augmented with query-attention and memory networks, which exploit background knowledge.

Appendix: Details of KVMemNN baseline

This appendix details the baseline method (**KVMemNN**) that uses key-value memory networks (Miller et al. 2016), which were also used in our experiments. This method chose key-value memory networks as an alternative way to exploit causal relations as background knowledge. Instead of labeling the words, we modified the original input vector for each word that was given to an encoder using the key-value memory network in which the causal relations are encoded and stored.

The modified input vector is computed in the following four steps: *causal relation encoding*, *memory addressing*, *memory reading* and *vector updating*. An overview of the whole method is illustrated in Figure 3.

Causal relation encoding: In our KVMemNN, given J causal relations that are relevant to a target why-question, we stored the effect parts of the relations as *keys* and the cause parts as *values*. We first computed the representations of the keys and values following Miller et al. (2016)[6]. As a value stored in the network, we computed the cause part

[6]https://github.com/facebook/MemNN/tree/master/KVmemnn

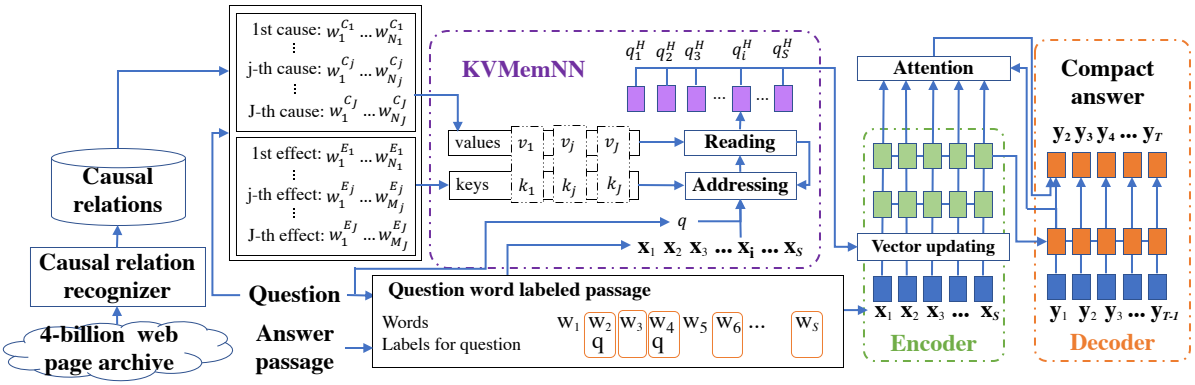

Figure 3: Baseline model augmented with KVMemNNs

representation $\mathbf{v}_j$ from the j-th cause part as the sum of the word embeddings of all the words in the cause part. As key representations, we computed the effect part representation $\mathbf{k}_j$ from the j-th effect part as the sum of the word embeddings of all the words in the effect part. These two types of representation are iteratively used for the processes of *memory addressing* and *reading*, both of which are described below. The number of iterations is referred to as *hop*, which is denoted by h in the following.

Memory addressing: At the first step of memory addressing, we compute the representation $\mathbf{q}$ of a target why-question Q as the sum of the word embedding of every word in Q. Then the *initial query vector* $\mathbf{q}_s^0$ for the s-th word's feature vector $\mathbf{x}_s$, which is the original input vector given to the encoder, is computed as:

$$\mathbf{q}_s^0 = \mathbf{A}\left(\mathbf{q} + \mathbf{x}_s\right), \tag{18}$$

where $\mathbf{A} \in \mathbb{R}^{d \times d}$ is a trainable matrix. Then at the h-th hop, the j-th causal relation is assigned a relevance probability by comparing the query vector $\mathbf{q}_s^h$ and the effect part representation:

$$p_{s,j}^h = \mathrm{softmax}\left(\mathbf{q}_s^{h\top} \mathbf{A} \mathbf{k}_j\right). \tag{19}$$

Memory reading: In the memory reading step, we computed the weighted sum of the values in the memory using the above relevance probability:

$$\mathbf{o}_s^h = \sum_j p_{s,j}^h \mathbf{A} \mathbf{v}_j. \tag{20}$$

Then we updated the query vector by combining the current query vector $\mathbf{q}_s^h$ and the above $\mathbf{o}_s^h$ as:

$$\mathbf{q}_s^{h+1} = \mathbf{R}_h\left(\mathbf{q}_s^h + \mathbf{o}_s^h\right), \tag{21}$$

where $\mathbf{R}_h \in \mathbb{R}^{d \times d}$ is a trainable matrix. We updated the query vector $\mathbf{q}_s^h$ H times using equations (19), (20), and (21), where H is the fixed number of hops.

Vector updating: Finally, we obtained a modified input vector $\tilde{\mathbf{x}}_s$ to be given to the encoder by combining the original input vector $\mathbf{x}_s$ and the final query vector $\mathbf{q}_s^H$ as:

$$m_s = \sigma\left(\mathbf{W}_1 \mathbf{q}_s^H + b\right), \tag{22}$$

$$\tilde{\mathbf{x}}_s = \mathrm{drop}\left(\mathbf{x}_s + m_s \mathbf{u}_1\right), \tag{23}$$

where σ denotes the sigmoid activation function, drop denotes the dropout operation (Hinton et al. 2012), and $\mathbf{W}_1 \in \mathbb{R}^{1 \times d}$, $\mathbf{u}_1 \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are trainable parameters.

Note that computing the modified input vectors using equations (22) and (23) is intended to function in a similar way to our stacked CNNs (especially as in equation (9) of Section 3.2). In stacked CNNs, the predicted value $\mathbf{p}_i$ is added after expanding it with the trainable vector $\mathbf{u}_x$. Analogously, in the above vector updating, we first computed the value m_s in equation (22) and then expanded it using the trainable vector u_1 in equation (23). We also tried a simpler version of vector updating in which the modified input vector $\tilde{\mathbf{x}}_s$ is computed as:

$$\tilde{\mathbf{x}}_s = \mathrm{drop}\left(\mathbf{x}_s + \mathbf{q}_s^H\right). \tag{24}$$

However, it drastically decreased the ROUGE scores and we used equations (22) and (23) for computing $\tilde{\mathbf{x}}_s$ in the baseline method using the KVMemNN.

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR 2015*.

Chen, Q.; Zhu, X.; Ling, Z.; Wei, S.; and Jiang, H. 2016. Distraction-based neural networks for modeling documents. In *IJCAI-16*, 2754–2760.

Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *EMNLP 2014*, 1724–1734.

Chopra, S.; Auli, M.; and Rush, A. M. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *NAACL 2016*, 93–98.

Dauphin, Y. N.; Fan, A.; Auli, M.; and Grangier, D. 2017. Language modeling with gated convolutional networks. In *ICML*, 933–941.

dos Santos, C. N.; Tan, M.; Xiang, B.; and Zhou, B. 2016. Attentive pooling networks. *CoRR* abs/1602.03609.

Gehring, J.; Auli, M.; Grangier, D.; Yarats, D.; and Dauphin, Y. N. 2017. Convolutional sequence to sequence learning. *CoRR* abs/1705.03122.

Girju, R. 2003. Automatic detection of causal relations for question answering. In *the ACL 2003 Workshop on Multilingual Summarization and Question Answering*, 76–83.

Higashinaka, R., and Isozaki, H. 2008. Corpus-based question answering for why-questions. In *IJCNLP 2008*, 418–425.

Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR* abs /1207.0580.

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

Ishida, R.; Torisawa, K.; Oh, J.-H.; Iida, R.; Kruengkrai, C.; and Kloetzer, J. 2018. Semi-distantly supervised neural model for generating compact answers to open-domain why questions. In *AAAI 2018*.

Jean, S.; Cho, K.; Memisevic, R.; and Bengio, Y. 2015. On using very large target vocabulary for neural machine translation. In *ACL-IJCNLP 2015*, 1–10.

Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. In *ICLR 2015*.

Kruengkrai, C.; Torisawa, K.; Hashimoto, C.; Kloetzer, J.; Oh, J.; and Tanaka, M. 2017. Improving event causality recognition with multiple background knowledge sources using multi-column convolutional neural networks. In *AAAI 2017*, 3466–3473.

Kudo, T.; Yamamoto, K.; and Matsumoto, Y. 2004. Applying conditional random fields to Japanese morphological analysis. In *EMNLP 2004*, 230–237.

Lin, C.-Y. 2004. ROUGE: A package for automatic evaluation of summaries. In *the ACL-04 Workshop on Text Summarization Branches Out*, 74–81.

Luong, M.-T.; Pham, H.; and Manning, C. D. 2015. Effective approaches to attention-based neural machine translation. In *EMNLP 2015*, 1412–1421.

Miao, Y., and Blunsom, P. 2016. Language as a latent variable: Discrete generative models for sentence compression. In *EMNLP 2016*, 319–328.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS 2013*, 3111–3119.

Miller, A.; Fisch, A.; Dodge, J.; Karimi, A.-H.; Bordes, A.; and Weston, J. 2016. Key-value memory networks for directly reading documents. In *EMNLP 2016*, 1400–1409.

Mizuno, J.; Tanaka, M.; Ohtake, K.; Oh, J.-H.; Kloetzer, J.; Hashimoto, C.; and Torisawa, K. 2016. WISDOM X, DISAANA and D-SUMM: Large-scale NLP systems for analyzing textual big data. In *COLING 2016*, 263–267.

Nallapati, R.; Zhou, B.; dos Santos, C.; Gulcehre, C.; and Xiang, B. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *CoNLL 2016*, 280–290.

Nema, P.; Khapra, M. M.; Laha, A.; and Ravindran, B. 2017. Diversity driven attention model for query-based abstractive summarization. In *ACL 2017*, 1063–1072.

Oh, J.-H.; Torisawa, K.; Hashimoto, C.; Sano, M.; Saeger, S. D.; and Ohtake, K. 2013. Why-question answering using intra- and inter-sentential causal relations. In *ACL 2013*, 1733–1743.

Oh, J.-H.; Torisawa, K.; Hashimoto, C.; Iida, R.; Tanaka, M.; and Kloetzer, J. 2016. A semi-supervised learning approach to why-question answering. In *AAAI-16*, 3022–3029.

Oh, J.-H.; Torisawa, K.; Kruengkrai, C.; Iida, R.; and Kloetzer, J. 2017. Multi-column convolutional neural networks with causality-attention for why-question answering. In *WSDM 2017*, 415–424.

Rajpurkar, P.; Zhang, J.; Lopyrev, K.; and Liang, P. 2016. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP 2016*, 2383–2392.

Rush, A. M.; Chopra, S.; and Weston, J. 2015. A neural attention model for abstractive sentence summarization. In *EMNLP 2015*, 379–389.

See, A.; Liu, P. J.; and Manning, C. D. 2017. Get to the point: Summarization with pointer-generator networks. In *ACL 2017*, 1073–1083.

Sharp, R.; Surdeanu, M.; Jansen, P.; Clark, P.; and Hammond, M. 2016. Creating causal embeddings for question answering with minimal supervision. In *EMNLP 2016*, 138–148.

Tan, M.; dos Santos, C. N.; Xiang, B.; and Zhou, B. 2016. Improved representation learning for question answer matching. In *ACL 2016*, 464–473.

van den Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A. W.; and Kavukcuoglu, K. 2016. WaveNet: A generative model for raw audio. *CoRR* abs/1609.03499.

Verberne, S.; Halteren, H.; Theijssen, D.; Raaijmakers, S.; and Boves, L. 2011. Learning to rank for why-question answering. *Information Retrieval* 14(2):107–132.

Yu, F., and Koltun, V. 2015. Multi-scale context aggregation by dilated convolutions. *CoRR* abs/1511.07122.

Zhou, Q.; Yang, N.; Wei, F.; and Zhou, M. 2017. Selective encoding for abstractive sentence summarization. In *ACL 2017*, 1095–1104.