Open Domain Event Text Generation*

Zihao Fu,[1] Lidong Bing,[2] Wai Lam[1]
[1]Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong
[2]DAMO Academy, Alibaba Group
{zhfu, wlam}@se.cuhk.edu.hk; l.bing@alibaba-inc.com

Abstract

Text generation tasks aim at generating human-readable text from different kinds of data. Normally, the generated text only contains the information included in the data and its application is thus restricted to some limited scenarios. In this paper, we extend the task to an open domain event text generation scenario with an entity chain as its skeleton. Specifically, given an entity chain containing several related event entities, the model should retrieve from a trustworthy repository (e.g. Wikipedia) the detailed information of these entities and generate a description text based on the retrieved sentences. We build a new dataset called WikiEvent[1] that provides 34K pairs of entity chain and its corresponding description sentences. To solve the problem, we propose a wiki augmented generator framework that contains an encoder, a retriever, and a decoder. The encoder encodes the entity chain into a hidden space while the decoder decodes from the hidden space and generates description text. The retriever retrieves relevant text from a trustworthy repository which provides more information for generation. To alleviate the overfitting problem, we propose a novel random drop component that randomly deletes words from the retrieved sentences making our model more robust for handling long input sentences. We apply the proposed model on the WikiEvent dataset and compare it with a few baselines. The experimental results show that our carefully-designed architecture does help generate better event text, and extensive analysis further uncovers the characteristics of the proposed task.

Introduction

In recent years, many natural language generation (NLG) tasks have been proposed to generate human-readable text based on different kinds of data. (Gardent et al. 2017a; 2017b) propose the WebNLG task to generate text descriptions based on a group of knowledge base triples. The E2E (Novikova, Dušek, and Rieser 2017) task aims to generate restaurant reviews with respect to the given attributes. The WikiBio (Lebret, Grangier, and Auli 2016) task generates the first sentence of a person's biography based on its Wikipedia Infobox.

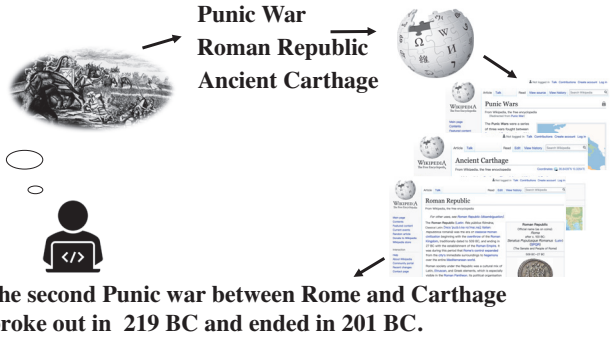

Figure 1: Illustration of the Open Domain Event Text Generation task. The event description is generated based on a given entity chain.

The current NLG tasks always assume that the generated text only contains the information included in the given data. However, this assumption is very rigid and sort of less practical in the real world. For example, in the novel writing scenario as shown in Fig. 1, when a writer wants to describe the "Punic War", what comes into his mind first is only some entity words such as "Punic War", "Roman Republic" and "Ancient Carthage". Then, he digs his own or turns to certain resources for help to acquire knowledge of these entities to compose a complete sentence. It would be very useful if such event text description can be automatically generated.

In this paper, we propose a new task named Open Domain Event Text Generation (ODETG). In this task, we are given a sequence of entities called an entity chain. The goal is to generate a sentence with this entity chain as its skeleton. We assume that we only know the entity chain and require the model to retrieve from a trustworthy repository (e.g. Wikipedia) to find the detailed information of these entities and generate a description text based on the retrieved sentences. The ODETG task can be applied to more scenarios where the traditional generation task cannot handle eas-

*The work described in this paper is substantially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project Code: 14204418).

[1]Available at https://github.com/fuzihaofzh/WikiEvent

ily. Traditional tasks are restricted since they assume that the given data contain all the needed information and the generated sentences can only describe the given information. Different from traditional generation tasks, the ODETG task requires to retrieve information from a trustworthy repository. Therefore, such nature allows our new task to be used in much broader scenarios, meanwhile, it also makes the task more challenging.

We propose a novel Wiki Augmented Generator (WAG) framework to solve this problem of generating sentences based on a given entity chain. Our framework firstly retrieves sentences from a trustworthy sentence repository with queries formed with the entities, and then it generates an event description with the retrieved sentences. Accordingly, the core part of our framework contains an encoder, a retriever and a decoder. The encoder encodes the given entity chain into a hidden space. The retriever retrieves the related information for the given entity chain. Precisely, it retrieves the sentences describing the entities from the sentence repository. The decoder generates text based on the encoder output and the retrieved text. We observe that since the retrieved sentences can be extremely long, the training procedure of the generation model can easily get overfitted. We employ a random drop component to alleviate the overfitting problem.

We build a real-world dataset called WikiEvent to evaluate the performance of our model. The dataset is built based on Wikipedia and provides 34K pairs of entity chains and their corresponding description text. We also build a sentence repository based on Wikipedia for the retriever to fetch useful text information for the generation sub-task. Presumably, it can be regarded as a trustworthy one and, of course, it could be replaced by other resources such as daily news. In summary, our contributions are as follows. (1) We propose a new task, namely, Open Domain Event Text Generation (ODETG), which is much more challenging and practical. (2) We contribute a new dataset called WikiEvent which can be used to evaluate the models for this task. (3) We propose a novel model tackling the characteristics of this task to solve the ODETG problem. (4) We propose an effective random drop component to tackle the overfitting problem encountered in the training process.

Related Works

Nowadays, many data-to-text generation tasks have been proposed to generate text aiming at explaining different kinds of data in a more human-readable form. For example, the WebNLG task (Gardent et al. 2017a; 2017b) generates sentences corresponding to a group of related triples sampled from DBpedia (Auer et al. 2007; Lehmann et al. 2015). A similar task is to generate a description text from the attributes of an entity, such as generating a short biography with person attributes in the WikiBio dataset (Lebret, Grangier, and Auli 2016), or a review from the restaurant attributes in the E2E dataset (Novikova, Dušek, and Rieser 2017). (Chen and Mooney 2008; Wiseman, Shieber, and Rush 2017) propose to generate a summarization of a competition. All these tasks strive to generate text exactly explaining the data without adding extra information to help understand the data.

On the other hand, many works have been proposed to generate text from different kinds of prompts. Fan, Lewis, and Dauphin (2018) use reddit's WritingPrompts data to expand a short story to a long one. (Park and Ahn 2018; Feng et al. 2018; Peng et al. 2018) propose to generate sentences from keywords. Yan (2016) proposes to generate a poem from given keywords. Drissi, Watkins, and Kalita (2018) propose to generate text by learning from the output of a summarization model. Wang, Chen, and Lee (2019) propose to generate text from hidden explainable topics. (Li et al. 2013; Li, Luong, and Jurafsky 2015; Martin et al. 2018) propose to generate text based on an abstract state translation. However, instead of using a trustworthy repository to provide information, these tasks generate text solely based on the prompts and knowledge from the training set. Different from them, our framework expands the semantic of the input entity chain by retrieving rich content from a trustworthy repository, i.e. Wikipedia.

WikiEvent Dataset Construction

We build our WikiEvent dataset based on the English Wikipedia dump[2] with three steps: (1) selecting articles describing some events; (2) extracting hyperlinked entities to form (entity chain, text) pairs; (3) filtering the candidates to build the final dataset. In addition, we also build a trustworthy sentence repository from the Wikipedia dump.

Article Selection

In order to find articles describing some events, we determine some keywords (namely Battle, Revolution, Revolt, Campaign, Rebellion, Siege, War, Conflict, Invasion, Incident, Conference, Treaty, Affair, Uprising, Expedition) that may refer to some events and choose the articles containing any keyword in their titles. We ignore the functional articles that contain words such as "Category", "List of ", "disambiguation", "Image:", "File:" in the title.

Candidate Extraction

We split the article text into sentences with NLTK (Bird and Loper 2004). After we get the tokenized sentences, we extract the entities and corresponding text by regular expressions. For example, given the text in Wikipedia markup language "[[Washington, D.C.|Washington]] is the capital of the [[United States|U.S.]]", an entity is marked in the square brackets and it is separated into two parts by "|". The first part in the brackets is the formal entity name while the second part is the surface text shown in the rendered Wikipedia page. Therefore, the extracted entity chain is "Washington, D.C. — United States", while the extracted text description is "Washington is the capital of the U.S.". In such manner, one entity in different entity chains keeps the same form and its different surface forms may occur in different description text. After the extraction, we get a candidate set of (entity chain, text) pairs, and each of them contains an entity chain as the skeleton and a piece of text as the target description.

[2]https://dumps.wikimedia.org/enwiki/20190320/

	train	test	dev
Battle	9,770	644	629
Revolution	1,221	64	83
Revolt	328	23	22
Campaign	1,748	113	98
Rebellion	425	31	23
Siege	1,689	109	112
War	8,187	551	558
Conflict	548	34	33
Invasion	584	40	41
Incident	506	43	48
Conference	2,371	154	166
Treaty	1,261	106	83
Affair	410	23	34
Uprising	299	19	25
Expedition	653	46	45
Total	30,000	2,000	2,000

Table 1: Dataset

Filtering

We use a rule-based filter to further clean the candidate pairs obtained above since some of them may not be suitable for our task. We require the number of entities in each pair to be less than 10 and the target sentence length in the range of 20 to 50 words. We also filter out the text containing certain keywords (such as "Category", "Portal", "Timeline of") since we observe that such text is unlikely to be a desirable event description. After some other filtering steps, we finally obtain a set of 34,000 (entity chain, event text) pairs, and split it into train, development (dev), and test sets. The statistics of the final dataset is shown in Table 1.

Trustworthy Sentence Repository

We also build a trustworthy sentence repository for storing the raw information supporting the process of retrieving useful related information for text generation. This repository is a collection of information from trustworthy source (including Wikipedia, news, etc.). We split all the articles in the dump into sentences similar to Section . By importing the sentences into Elasticsearch[3], they can be retrieved with search queries, i.e. entity names.

Our Proposed Framework

Overview

Formally, we denote e_i as the ith entity in an entity chain $E_c = [e_1, e_2, \cdots, e_n]$. Given an entity chain E_c, the goal of the model is to generate a target sequence $G = [g_1, g_2, \cdots, g_l]$ where g_i is the ith word and G maximizes the conditional probability:

$$\max_G P(G|E_c).$$

Our proposed Wiki Augmented Generator (WAG) framework is illustrated in Fig. 2. It is composed of three components, i.e. an encoder, a retriever and a decoder. The encoder, on the bottom left, contains a typical S2S encoder to encode the entity chain into hidden vectors. In order to utilize more detailed information, we require the model to retrieve related information for each entity from the trustworthy repository. The retriever on the bottom right uses the input entity as the query words to retrieve some sentences from the trustworthy repository. After that, it will go through a random drop (RD) component which randomly removes some words when training. The retriever only depends on the entity chain and thus can be invoked as a pre-processing step. We denote s_i as the ith sentence in the retrieved sentences list $S_w = [s_1, s_2, \cdots, s_n]$. The decoder on the top generates a hidden state vector q for each generation step. Then, the decoder uses q as a query vector to impose an attention weight on each word of the retrieved text for calculating an aggregated vector. The aggregated vector and the hidden state vector are then used to generate the final output.

Encoder

We develop the encoder based on a common S2S framework. It encodes the entity chain into a hidden representation which will be used by the decoder. Each entity in the entity chain E_c is firstly concatenated one after another to form an input vector $E'_c = [w_{e_1}^{(1)}, \cdots, w_{e_1}^{(n_{e_1})}, \cdots, w_{e_n}^{(1)}, \cdots, w_{e_n}^{(n_{e_n})}]$, in which $w_{e_i}^{(j)}$ stands for the jth word in the ith entity in E_c. Subsequently, E'_c is sent to an embedding layer to get the embedding of each word:

$$E_h = Emb(E'_c),$$

where $E_h \in \mathbb{R}^{d_e \times n_c}$. d_e stands for the dimension of the embedding while n_c is the total word count of E'_c. The hidden representation E_h is then sent to an RNN layer (Here, we use multi-layer multi-directional LSTM) to get the hidden representation for each word:

$$H = RNN(E_h),$$

where $H \in \mathbb{R}^{d_h \times n_c}$. d_h is the size of each hidden vector. The final state and cell state of this RNN network will be used to initialize the decoder's RNN network.

Retriever

The retriever retrieves sentences from the sentence repository with traditional retrieve methods. We use the existing Elasticsearch tool to retrieve the relevant sentences from the trustworthy repository. It gives each sentence a score and returns the highly scored sentences. The score[4] in Elasticsearch is calculated as follows:

$$\mathrm{s}(e,d) = \frac{1}{\|e\|_q} \cdot C(e,d) \cdot \sum_{t \in e} (\mathrm{tf}(t,d) \cdot \mathrm{idf}(t)^2 \cdot B(t) \cdot \|d\|_f),$$

where e is the query. $\frac{1}{\|e\|_q}$ is the query normalization factor. $C(e,d)$ is the coordination factor. $\mathrm{tf}(t,d)$ is the term frequency for the term t in the document d. $\mathrm{idf}(t)$ is the inverse document frequency for the term t. $B(t)$ is the boosted

[3] https://www.elastic.co/

[4] https://www.elastic.co/guide/en/elasticsearch/guide/current/practical-scoring-function.html

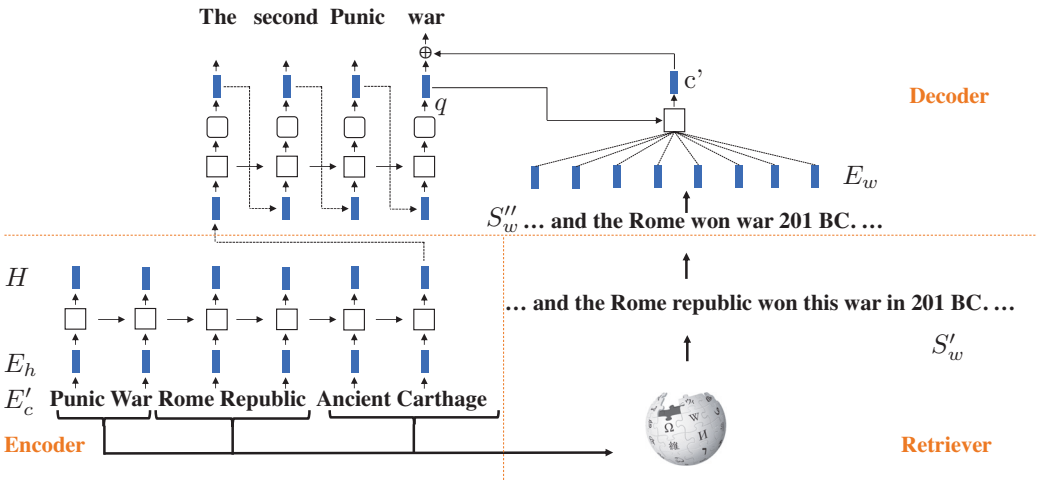

Figure 2: Our proposed Wiki Augmented Generator (WAG).

value for t. $\|d\|_f$ is the field-length norm of the document d. The detailed definition can be found in the reference guide[4]. When retrieving related sentences for an entity chain E_c, we retrieve sentences for each entity respectively and combine them together. For the entity e_i, the score function can be expressed as:

$$\text{score}(e_i, d) = \mathbb{1}(e_i \in d)\text{s}(e_1 + e_2 + \cdots + e_n, d),$$

where $\mathbb{1}$ is an indicator function. It equals to 1 when $e_i \in d$ and equals to 0 otherwise. $e_1 + e_2 + \cdots + e_n$ denotes the concatenation of the entities into a long sequence. Subsequently, we select top-n_r sentences for each entity to make the retrieved sentences list S_w. It is then concatenated into a sequence $S'_w = [w^{(1)}_{s_1}, \cdots, w^{(n_{s_1})}_{s_1}, \cdots, w^{(1)}_{s_m}, \cdots, w^{(n_{s_m})}_{s_m}]$, where $w^{(j)}_{s_i}$ stands for the jth word in the ith sentence in S_w.

Since the retriever only depends on the entity chain, it can be conducted as a pre-processing step. Meanwhile, the retriever component is only based on statistics of words and thus it has no parameters to learn.

Decoder

The decoder first applies a random drop component on the retriever output and then it utilizes the remaining token sequence combined with the encoder output to generate the target text.

It has been observed that the retrieved sentences can be extremely long. Consequently, the training procedure can easily get overfitted since it can just ad-hocly focus on a particular portion of the retrieved text, i.e. S'_w, and uses it to generate the target. To alleviate this problem, we propose a random drop (RD) mechanism that randomly removes words in S'_w with a drop ratio. Different from the traditional dropout layer, RD directly deletes the words while the dropout layer just sets the random dimensions to 0. Let $S''_w = \text{RD}(S'_w)$ denote the sequence after the RD operation. The RD component is only applied in the training stage, while set ineffective in the testing stage, since its purpose is for preventing overfitting of training.

The hidden state and cell state of each decoding step ($t > 0$) are calculated by passing the previous step's states to an RNN cell:

$$h_t, c_t = RNN([h_{t-1}, c_{t-1}]),$$

where $h_t \in \mathbb{R}^{d_h}$ is the hidden output of an RNN cell and $c_t \in \mathbb{R}^{d_h}$ is the cell state. The encoder's final state is used h_0. h_t is then sent to an attention layer to get the traditional attention context vector, called query vector q here, from H weighted by the relatedness to the current state h_t:

$$q = Attn(h_t, H),$$

where $Attn$ is the traditional attention function (Bahdanau, Cho, and Bengio 2014). In the traditional S2S framework, the output of the attention layer, i.e. q, is directly used to predict the next output word. In our framework, since we have extra retrieved text information, i.e. S''_w, q will be used as a query vector to fetch additional context information from S''_w.

Specifically, we firstly embed S''_w as:

$$E_w = Emb(S''_w),$$

where $E_w \in \mathbb{R}^{d_e \times |S''_w|}$ and $|\cdot|$ denotes the sequence length. It has been observed that $|S''_w|$ can always be very large even if we use a random drop component to remove some words. If we directly use E_w for the decoder, a lot of irrelevant information may be introduced and it thus hurts the performance. To alleviate this problem, we use an attention mechanism

to select relevant information for the decoder. For the vector $q \in \mathbb{R}^{d_h}$, it first goes through a linear layer to get the transformed query vector:

$$q' = W_e q + b_e,$$

where $W_e \in \mathbb{R}^{d_h \times d_e}$ and $b_e \in \mathbb{R}^{d_e}$ is the transform matrix and bias vector respectively. The aligning weight is calculated by a simple matrix product with the embedding E_w and q:

$$s = E_w^T q',$$

where $s \in \mathbb{R}^{|S''_w|}$ is the alignment vector. Afterwards, s is passed through a softmax layer to get a probability over all the words:

$$a = \frac{e^s}{\sum_{j=1}^{|S''_w|} e^{s_j}},$$

where $a \in \mathbb{R}^{|S''_w|}$ and the sum of all elements in a equals to 1. The probability gives each word in the retrieved sentences a weight. Then, a context vector is calculated by the weighted sum of the E_w matrix with the weight a:

$$c' = E_w a,$$

where $c' \in \mathbb{R}^{d_e}$. It contains the information we need from the retrieved text.

The decoder generates the output text from the context vector and the query vector jointly. The context vector c' is concatenated with the input query vector. Then it goes through a linear layer and a tanh layer to get the final context vector.

$$c = \tanh(W_c[q; c'] + b_c),$$

where $c \in \mathbb{R}^{d_h}, W_c \in \mathbb{R}^{d_h \times (d_h + d_e)}, b_c \in \mathbb{R}^{d_h}$. c is the context vector and is used to calculate the probability for all the words as:

$$y_t = Softmax(W_y c + b_y),$$

where $y_t \in \mathbb{R}^{|V|}$ and $|V|$ is the size of the target vocabulary. $W_y \in \mathbb{R}^{|V| \times d_h}, b_y \in \mathbb{R}^{|V|}$. Copy mechanism (He et al. 2017) is utilized to solve the out-of-vocabulary problem.

Experiments

Experiment Setup

The output is evaluated with several metrics, namely BLEU (Papineni et al. 2002), NIST (Doddington 2002), METEOR (Banerjee and Lavie 2005), ROUGE$_L$ (Lin 2004), and CIDEr (Vedantam, Lawrence Zitnick, and Parikh 2015). The evaluation is conducted with the package provided by (Novikova, Dušek, and Rieser 2017) and we follow the setting in the package where the ROUGE$_L$ is a harmonic mean of precision and recall with $\beta = 1.2$. We build our model based on the traditional sequence-to-sequence (S2S) model (Sutskever, Vinyals, and Le 2014; Cho et al. 2014; Klein et al. 2017). We set both the embedding size and the hidden size to 500, the dropout rate to 0.3, the optimization algorithm to SGD with the initial learning rate of 1.0 and the decay rate of 0.5. The random drop ratio is set to 0.5 and the retrieved sentence number is set to 1 for each entity. All the hyper-parameters are tuned on the dev set and some details are presented later.

Comparison Models

We employ a few baselines that are developed with the S2S framework and its variants that use our entity chain and the retrieved sequence as input.

S2S-E utilizes the traditional S2S (Sutskever, Vinyals, and Le 2014; Cho et al. 2014; Klein et al. 2017) framework with the standard attention (Bahdanau, Cho, and Bengio 2014; Luong, Pham, and Manning 2015) and the copy mechanism (He et al. 2017). We flatten the entity chain as a simple sequence and separate each entity with a special token. It is then used as the input of the S2S framework.

S2S-W has the same structure as S2S-E model. We firstly retrieve Wikipedia sentences from the repository with the entity chain as the key. Then, we directly use the retrieved text as the input of the S2S framework.

S2S-E-W-1 uses the entity chain as the S2S input sequence and uses the retrieved text as an additional feature vector. It encodes the retrieved text by an RNN network and averages the output states of all time steps to convert the text as a feature vector. This vector is then concatenated to the input vector at each decoding step to generate the output sequence.

S2S-E-W-2 firstly retrieves sentences from the trustworthy repository and concatenates the retrieved text at the end of the flattened entity chain. The concatenation is then sent into the S2S framework.

S2S-E-W-2' uses the same input of S2S-E-W-2 but adds a random drop (RD) component on the text before it is concatenated at the end of the flattened entity chain.

S2S-E-W-2" is similar to S2S-E-W-2'. The difference is that S2S-E-W-2' drops words randomly while S2S-E-W-2" drops words with a probability depending on whether the current word is near a word in the entity chain. Words far from the entity words are more likely to be dropped.

WAG w/o RD is an ablation variant of our WAG model, by removing the RD component.

	BLEU	NIST	METEOR	ROUGE$_L$	CIDEr
S2S-E	0.200	4.38	0.200	0.377	1.47
S2S-W	0.157	3.92	0.156	0.316	1.11
S2S-E-W-1	0.210	4.84	0.201	0.377	1.51
S2S-E-W-2	0.224	5.16	0.204	0.385	1.62
S2S-E-W-2'	0.224	5.11	0.209	0.394	1.64
S2S-E-W-2"	0.231	5.51	0.209	0.387	1.62
WAG w/o RD	0.231	5.43	0.207	0.383	1.69
WAG	0.238	5.64	0.216	0.397	1.76

Table 2: Main results

Experimental Results

Main Results. The experimental result is shown is Table 2. It can be observed that our WAG model outperforms all baseline models, and presumably we can draw the following

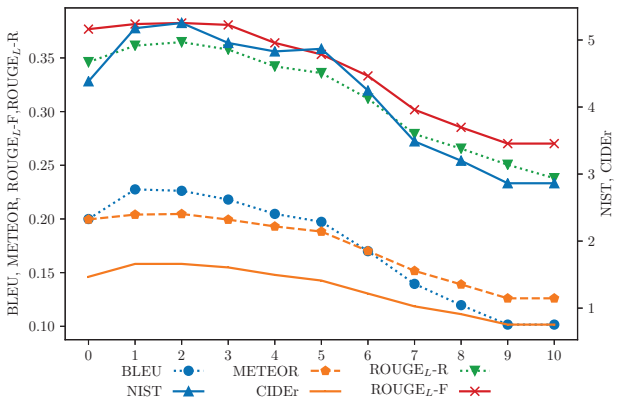

Figure 3: The effect of the number of retrieved sentence.

three conclusions. (1) Comparing with S2S-E and S2S-W, WAG utilizes both types of input information and thus it can naturally outperform the two baselines. (2) Comparing with the four baselines of S2S-E-W-X's, WAG can more appropriately utilize the two types of information to generate better text. (3) The comparison with WAG w/o RD shows that our proposed RD component does help train a more robust model so that it can perform better for testing.

Specifically, S2S-W performs very poor because without giving the entity chain as skeleton, the model does not have sufficient guidance regarding what aspect to talk more about and it is even worse than only using the entity chain, i.e. S2S-E. S2S-E-W-X baselines can easily outperform S2S-E and S2S-W, no matter how they combine the two types of information, even in a rather simple manner as used by S2S-E-W-1. S2S-E-W-2 can further improves S2S-E-W-1 by concatenating the entity chain and the text, which produces a more favourable input for the S2S model. By applying the drop operation for preventing training overfitting, S2S-E-W-2' and S2S-E-W-2" get even better results. Overall, S2S-E-W-2" performs slightly better than S2S-E-W-2' (i.e. more advantages on BLEU and NIST, even on METEOR, less disadvantages on ROUGE and CIDE), probably because its drop mechanism is more sophisticated than RD, i.e. deleting more words that are less likely to be used in the target text. Note that WAG adopts the RD component for drop operation since it is more straightforward and efficient. Even though, WAG can still outperform S2S-E-W-2" on all metrics, which can definitely be attributed to the tailor-made model architecture of WAG. Concretely, WAG utilizes the fetched information at each decoding step by querying the retrieved text (i.e. its embedding matrix) with a query vector of that step, and thus it can generate text that complies better with the skeleton and meanwhile provides more appropriate semantic context from the fetched information.

Effect of the Number of Retrieved Sentence. In the above main result, we retrieve 1 sentence for each entity. To study the influence of the retrieval sentence number, we conduct an experiment by setting the number ranging from 0 to 10. The result is illustrated in Fig. 3. It can be observed that under both the precision-oriented metric (i.e. BLEU) and the recall-oriented metric (i.e. ROUGR$_L$-R), introducing the retrieved text do help improve the result. In general, retrieving one or two sentences for each query entity achieves the best performance. As the increasing of the retrieved sentences, the sentence quality is decreasing and some irrelevant sentences could be involved. Thus, it enlarges the difficulty for the model to distill the useful information.

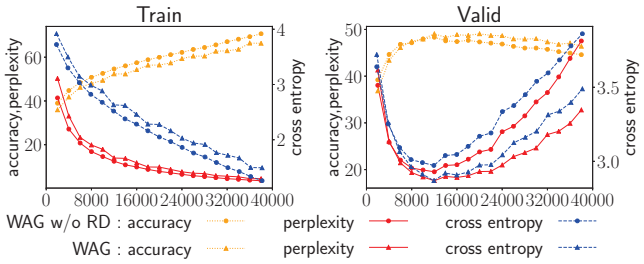

Figure 4: Training curve on every 2,000 steps.

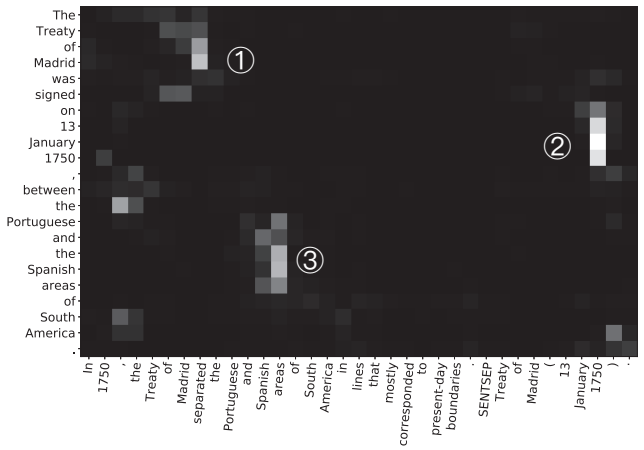

Figure 5: Attention map for the decoder. The entity chain is "South America — Treaty of Madrid (1750)". The y-axis is the generated text, while the x-axis is the retrieved text where "SENTSEP" is the separator between two sentences.

Effect of Random Drop. To have a closer study on the effect of our random drop component, we plot 3 metrics [5] namely accuracy, perplexity and cross entropy every 2,000 steps during the training stage in Fig. 4. The accuracy is defined as the ratio of the correct word count over the total word count. The perplexity is defined as the exponential of the loss over the total word count. The cross-entropy is defined as the loss over the total word count. Higher accuracy with lower perplexity and cross entropy represents a better performance for the model. It can be observed that in training, WAG w/o RD component outperforms WAG with RD. The reason is that randomly dropping words makes it harder to learn, and of course, the other side of the coin is

[5] http://opennmt.net/OpenNMT-py/index.html

Entity Chain	Retrieved Sentences	Generated Text	Gold Text
Western Desert Campaign — Second World War — Battle of El Agheila	During the Second World War the north-eastern desert between El Agheila and the Egyptian border was the scene of heavy fighting between the Axis powers and the Western Allies , a period known as the Western Desert Campaign . <SENTSEP> The " Siege of Tobruk " lasted for 241 days in 1941 , after Axis forces advanced through Cyrenaica from El Agheila in Operation Sonnenblume against Allied forces in Libya , during the Western Desert Campaign (1940–1943) of the Second World War . <SENTSEP> The defence of "Outpost Snipe" in Egypt , took place in the Second Battle of El Alamein , part of the Western Desert Campaign during the Second World War .	The Battle of El Agheila was a battle of the Western Desert Campaign of the Second World War .	The Battle of El Agheila was a brief engagement of the Western Desert Campaign of the Second World War .
Tsardom of Russia — Constantinople — Istanbul — Ottoman Empire — Ottoman	It ended the Russo-Turkish War of 1686-1700 . <SENTSEP> The " Vilayet of Constantinople " or " Istanbul " was a first-level administrative division (vilayet) of the Ottoman Empire , encompassing the imperial capital , Constantinople (Istanbul) . <SENTSEP> The treaty was concluded on 3 July (O.S .) 13 July 1700 in Constantinople . The Tsardom of Russia and the Ottoman Empire agreed on a truce set to expire in thirty years . <SENTSEP>" Garabet Yazmaciyan " (1868 Constantinople , Ottoman Empire ; Istanbul , Turkey 1929) was a prominent Ottoman painter of Armenian descent. <SENTSEP> birth place = Istanbul , Ottoman Empire	The Treaty of Constantinople was signed on 3 July 1700 between the Tsardom of Russia and the Ottoman Empire.	The Treaty of Constantinople or Istanbul was signed on 13 July 1700 between the Tsardom of Russia and the Ottoman Empire .
Lo Giang — Battle of Lo Giang	During the Tet Offensive of 1968 , the People 's Army of Vietnam (PAVN) 2nd Division tried to capture Da Nang but they were defeated in the Battle of Lo Giang . <SENTSEP> " ' Lo Khac Tam " ' is a former Vietnamese officer and Lieutenant General who fought for the army of North Vietnam during the Vietnam War .	The Battle of Lo Giang was fought during the Vietnam War.	The Battle of Lo Giang was a battle during the Vietnam War.
Dongnae — Siege of Dongnae	It resulted in the capture of Dongnae , a mountain castle on the way to Hanseong , by the Japanese . <SENTSEP> In 1592 , after Japan ' s request for aid conquering Ming China was rebuffed , approximately 200,000 Japanese soldiers invaded Joseon , and the Japanese invasions of Korea (1592–98) began .	The Siege of Dongnae was a battle of the Japanese invasions of Korea (1592–98).	The Siege of Dongnae was a siege that occurred on 25 May 1592 during the Japanese invasions of Korea (1592–98).

Table 3: Examples for generated text.

the model is less likely to get overfitted. Therefore, in the validation, we observe that our WAG using RD suffers less from the overfitting problem. Specifically, both models get overfitted after 12,000 steps, but WAG w/o RD gets worse more quickly after that.

Case Study

In order to show that the decoder can focus on the correct position in the retrieved text, we draw the attention map for each generated word as shown in Fig. 5. Brighter stands for higher attention. It can be observed that during the generation procedure, three major areas have been focused on. The decoder pays more attention to the portion of "the Treaty of Madrid separated" in the retrieved text, i.e. area ①, when it generates the subject of the output. Then it utilizes the information fetched from area ② for outputting the time of the event. Finally, it outputs the parties of the treaty with the information fetched from area ③. It can be concluded that the decoder focuses on specific information and utilizes corresponding information for generating.

Table 3 presents some generated sentences to give an intuitive understanding of the task and the model. It can be observed that the generated sentences contain more specific information with the given entity chain as the skeleton. For example, in the second case, our model retrieved the information "the treaty of Constantinople was signed on 13 July" which is not given by the entity chain. It shows that our proposed model can successfully retrieve related information from the trustworthy repository and can give a more detailed description of the entity chain.

Conclusions

We propose a new task namely Open Domain Event Text Generation (ODETG). This task can be applied in scenarios where traditional generation tasks are not applicable. Besides, we contribute a new dataset, namely, WikiEvent, to evaluate models that solving the ODETG problem. It contains 34K (entity chain, text) pairs composed of an entity chain and the corresponding description text. Moreover, we propose a novel framework containing an encoder, a retriever and a decoder. The encoder encodes the entity chain into hidden representation. The retriever retrieves related information to enhance the generation procedure. The decoder with a random drop component successfully decodes a sequence depicting the entity chain with more detailed information. Our proposed model on the WikiEvent dataset outperforms a few baselines, which shows that our carefully-designed architecture does help generate better event text.

Extensive analysis further uncovers the characteristics of the new task and our proposed model.

In the future, we will explore the following directions: 1) We will try to incorporating more event articles to enrich the trustworthy repository. 2) We will explore using reinforcement learning to further refining the retrieved text.

References

Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; Cyganiak, R.; and Ives, Z. G. 2007. Dbpedia: A nucleus for a web of open data. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference*, 722–735.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Banerjee, S., and Lavie, A. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 65–72.

Bird, S., and Loper, E. 2004. NLTK: The natural language toolkit. In *The Companion Volume to the Proceedings of 42st Annual Meeting of the Association for Computational Linguistics*, 214–217. Barcelona, Spain: Association for Computational Linguistics.

Chen, D. L., and Mooney, R. J. 2008. Learning to sportscast: a test of grounded language acquisition. In *Proceedings of the 25th international conference on Machine learning*, 128–135. ACM.

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 *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734.

Doddington, G. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the Second International Conference on Human Language Technology Research*, 138–145.

Drissi, M.; Watkins, O.; and Kalita, J. 2018. Hierarchical text generation using an outline. *arXiv preprint arXiv:1810.08802*.

Fan, A.; Lewis, M.; and Dauphin, Y. 2018. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 889–898.

Feng, X.; Liu, M.; Liu, J.; Qin, B.; Sun, Y.; and Liu, T. 2018. Topic-to-essay generation with neural networks. In *IJCAI*, 4078–4084.

Gardent, C.; Shimorina, A.; Narayan, S.; and Perez-Beltrachini, L. 2017a. Creating training corpora for nlg micro-planners. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 179–188.

Gardent, C.; Shimorina, A.; Narayan, S.; and Perez-Beltrachini, L. 2017b. The webnlg challenge: Generating text from rdf data. In *Proceedings of the 10th International Conference on Natural Language Generation*, 124–133.

He, S.; Liu, C.; Liu, K.; and Zhao, J. 2017. Generating natural answers by incorporating copying and retrieving mechanisms in sequence-to-sequence learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 199–208.

Klein, G.; Kim, Y.; Deng, Y.; Senellart, J.; and Rush, A. 2017. OpenNMT: Open-source toolkit for neural machine translation. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, System Demonstrations* 67–72.

Lebret, R.; Grangier, D.; and Auli, M. 2016. Neural text generation from structured data with application to the biography domain. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1203–1213.

Lehmann, J.; Isele, R.; Jakob, M.; Jentzsch, A.; Kontokostas, D.; Mendes, P. N.; Hellmann, S.; Morsey, M.; van Kleef, P.; Auer, S.; et al. 2015. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web* 6(2):167–195.

Li, B.; Lee-Urban, S.; Johnston, G.; and Riedl, M. 2013. Story generation with crowdsourced plot graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Li, J.; Luong, M.-T.; and Jurafsky, D. 2015. A hierarchical neural autoencoder for paragraphs and documents. *arXiv preprint arXiv:1506.01057*.

Lin, C.-Y. 2004. Rouge: A package for automatic evaluation of summaries. *Workshop on Text Summarization Branches Out*.

Luong, T.; Pham, H.; and Manning, C. D. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1412–1421.

Martin, L. J.; Ammanabrolu, P.; Wang, X.; Hancock, W.; Singh, S.; Harrison, B.; and Riedl, M. O. 2018. Event representations for automated story generation with deep neural nets. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Novikova, J.; Dušek, O.; and Rieser, V. 2017. The e2e dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, 201–206.

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the Annual meeting on Association for Computational Linguistics*, 311–318.

Park, D., and Ahn, C. W. 2018. Lstm encoder-decoder with adversarial network for text generation from keyword. In *International Conference on Bio-Inspired Computing: Theories and Applications*, 388–396. Springer.

Peng, N.; Ghazvininejad, M.; May, J.; and Knight, K. 2018. Towards controllable story generation. In *Proceedings of the First Workshop on Storytelling*, 43–49.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 3104–3112.

Vedantam, R.; Lawrence Zitnick, C.; and Parikh, D. 2015. Cider: Consensus-based image description evaluation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 4566–4575.

Wang, Y.-S.; Chen, Y.-N.; and Lee, H.-Y. 2019. TopicGAN: Unsupervised text generation from explainable latent topics.

Wiseman, S.; Shieber, S.; and Rush, A. 2017. Challenges in data-to-document generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2253–2263.

Yan, R. 2016. i, poet: Automatic poetry composition through recurrent neural networks with iterative polishing schema. In *IJCAI*, 2238–2244.