Neural-based Natural Language Generation in Dialogue using RNN Encoder-Decoder with Semantic Aggregation

Van-Khanh Tran[1,2], Le-Minh Nguyen[1,⋆] and Satoshi Tojo[1]
[1]Japan Advanced Institute of Science and Technology, JAIST
1-1 Asahidai, Nomi, Ishikawa, 923-1292, Japan
{tvkhanh, nguyenml, tojo}@jaist.ac.jp
[2]University of Information and Communication Technology, ICTU
Thai Nguyen University, Vietnam
tvkhanh@ictu.edu.vn

Abstract

Natural language generation (NLG) is an important component in spoken dialogue systems. This paper presents a model called Encoder-Aggregator-Decoder which is an extension of an Recurrent Neural Network based Encoder-Decoder architecture. The proposed Semantic Aggregator consists of two components: an Aligner and a Refiner. The Aligner is a conventional attention calculated over the encoded input information, while the Refiner is another attention or gating mechanism stacked over the attentive Aligner in order to further select and aggregate the semantic elements. The proposed model can be jointly trained both text planning and text realization to produce natural language utterances. The model was extensively assessed on four different NLG domains, in which the results showed that the proposed generator consistently outperforms the previous methods on all the NLG domains.

1 Introduction

Natural Language Generation (NLG) plays a critical role in a Spoken Dialogue System (SDS), and its task is to convert a meaning representation produced by the dialogue manager into natural language sentences. Conventional approaches to NLG follow a *pipeline* which typically breaks down the task into *sentence planning* and *surface realization*. *Sentence planning* decides the order and structure of a sentence, followed by a *surface realization* which converts the sentence structure into final utterance. Previous approaches to NLG still rely on extensive hand-tuning templates and rules that require expert knowledge of linguistic representation. There are some common and widely used approaches to solve NLG problems, including rule-based (Cheyer and Guzzoni, 2014), corpus-based n-gram generator (Oh and Rudnicky, 2000), and a trainable generator (Ratnaparkhi, 2000).

Recurrent Neural Network (RNN)-based approaches have recently shown promising results in NLG tasks. The RNN-based models have been used for NLG as a joint training model (Wen et al., 2015a,b) and an end-to-end training network (Wen et al., 2016c). A recurring problem in such systems requiring annotated corpora for specific dialogue acts[*] (DAs). More recently, the attention-based RNN Encoder-Decoder (AREncDec) approaches (Bahdanau et al., 2014) have been explored to tackle the NLG problems (Wen et al., 2016b; Mei et al., 2015; Dušek and Jurčíček, 2016b,a). The AREncDEc-based models have also shown improved results on various tasks, *e.g.*, image captioning (Xu et al., 2015; Yang et al., 2016), machine translation (Luong et al., 2015; Wu et al., 2016).

To ensure that the generated utterance represents the intended meaning of the given DA, the previous RNN-based models were conditioned on a 1-hot vector representation of the DA. Wen et al. (2015a) proposed a Long Short-Term Memory-based (HLSTM) model which introduced a heuristic gate to guarantee that the slot-value pairs were accurately captured during generation. Subsequently, Wen et al. (2015b) proposed an LSTM-based generator (SC-LSTM) which jointly learned the controlling signal and language model. Wen et al. (2016b) proposed an AREncDec based generator (ENCDEC) which applied attention mechanism on the slot-value pairs.

[⋆]Corresponding author.

[*]A combination of an action type and a set of slot-value pairs). *E.g. inform(name='Piperade'; food='Basque')*.

Proceedings of the SIGDIAL 2017 Conference, pages 231–240,
Saarbrücken, Germany, 15-17 August 2017. ©2017 Association for Computational Linguistics

Table 1: Order issue in natural language generation, in which an incorrect generated sentence has wrong ordered slots.

Input DA	**Compare**(name=***Triton 52***; ecorating=***A+***; family=***L7***; name=***Hades 76***; ecorating=***C***; family=***L9***)
INCORRECT	The ***Triton 52*** has an ***A+*** eco rating and is in the ***L9*** product family, the ***Hades 76*** is in the ***L7*** product family and has a ***C*** eco rating.
CORRECT	The ***Triton 52*** is in the ***L7*** product family and has an ***A+*** eco rating, the ***Hades 76*** is in the ***L9*** product family and has a ***C*** eco rating.

Although these RNN-based generators have worked well, however, they still have some drawbacks, and none of these models significantly outperform the others in solving NLG tasks. While the HLSTM cannot handle cases such as the binary slots (*i.e.*, *yes* and *no*) and slots that take *don’t_care* value in which these slots cannot be directly delexicalized, the SCLSTM model is limited to generalize to the unseen domain, and the ENCDEC model has difficulty to prevent undesirable semantic repetitions during generation.

To address the above issues, we propose a new architecture, *Encoder-Aggregator-Decoder*, an extension of the AREncDec model, in which the proposed Aggregator has two main components: (i) an Aligner which computes the attention over the input sequence, and (ii) a Refiner which are another attention or gating mechanisms to further select and aggregate the semantic elements. The proposed model can learn from unaligned data by jointly training the sentence planning and surface realization to produce natural language sentences. We conduct comprehensive experiments on four NLG domains and find that the proposed method significantly outperforms the previous methods regarding BLEU (Papineni et al., 2002) and slot error rate ERR scores (Wen et al., 2015b). We also found that our generator can produce high-quality utterances with correctly ordered than those in the previous methods (see Table 1). To sum up, we make two key contributions in this paper:

- We present a semantic component called *Aggregator* which is easy integrated into existing (attentive) RNN encoder-decoder architecture, resulting in an end-to-end generator that empirically improved performance in comparison with the previous approaches.

- We present several different choices of attention and gating mechanisms which can be effectively applied to the proposed semantic Aggregator.

In Section 2, we review related works. The proposed model is presented in Section 3. Section 4 describes datasets, experimental setups and evaluation metrics. The results and analysis are presented in Section 5. We conclude with a brief of summary and future work in Section 6.

2 Related Work

Conventional approaches to NLG traditionally divide the task into a pipeline of sentence planning and surface realization. The conventional methods still rely on the handcrafted rule-based generators or rerankers. Oh and Rudnicky (2000) proposed a class-based n-gram language model (LM) generator which can learn to generate the sentences for a given dialogue act and then select the best sentences using a rule-based reranker. Ratnaparkhi (2000) later addressed some of the limitation of the class-based LMs by proposing a method based on a syntactic dependency tree. A phrase-based generator based on factored LMs was introduced by Mairesse and Young (2014), that can learn from a semantically aligned corpus.

Recently, RNNs-based approaches have shown promising results in the NLG domain. Vinyals et al. (2015); Karpathy and Fei-Fei (2015) applied RNNs in setting of multi-modal to generate caption for images. Zhang and Lapata (2014) also proposed a generator using RNNs to create Chinese poetry. For task-oriented dialogue systems, Wen et al. (2015a) combined two TNN-based models with a CNN reranker to generate required utterances. Wen et al. (2015b) proposed SC-LSTM generator which proposed an additional ”reading” cell to the traditional LSTM cell to learn the gating mechanism and language model jointly. A recurring problem in such systems lacking of sufficient domain-specific annotated corpora. Wen et al. (2016a) proposed an out-of-domain model which is trained on counterfeited datasets by using semantic similar slots from the target-domain dataset instead of the slots belonging to the out-of-domain dataset. The empirical results indicated

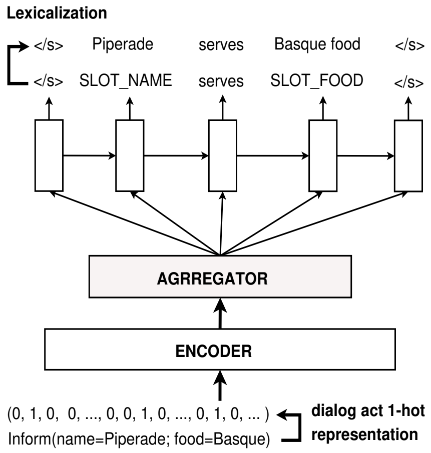
Lexicalization
</s> Piperade serves Basque food </s>
</s> SLOT_NAME serves SLOT_FOOD </s>
AGRREGATOR
ENCODER
(0, 1, 0, 0, ..., 0, 0, 1, 0, ..., 0, 1, 0, ...)
Inform(name=Piperade; food=Basque)
dialog act 1-hot representation

Figure 1: Unfold presentation of the RNN-based neural language generator. The encoder part is subject to various designs, while the decoder is an RNN network.

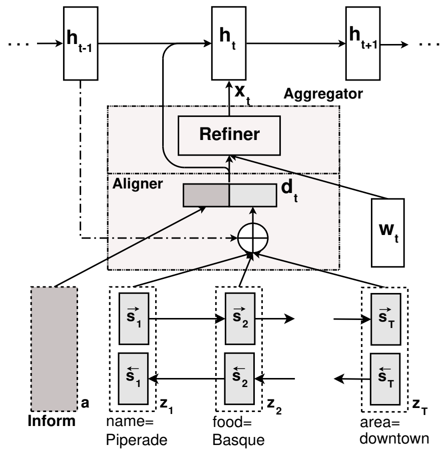

Figure 2: The RNN Encoder-Aggregator-Decoder for NLG proposed in this paper. The output side is an RNN network while the input side is a DA embedding with aggregation mechanism. The Aggregator consists of two parts: an Aligner and a Refiner. The lower part Aligner is an attention over the DA representation calculated by a Bidirectional RNN. Note that the action type embedding **a** is not included in the attention mechanism since its task is controlling the style of the sentence. The higher part Refiner computes the new input token $\mathbf{x}_t$ based on the original input token $\mathbf{w}_t$ and the dialogue act attention $\mathbf{d}_t$. There are several choices for Refiner, *i.e.*, gating mechanism or attention mechanism.

that the model can obtain a satisfactory results with a small amount of in-domain data by fine-tuning the target-domain on the out-of-domain trained model.

More recently, attentional RNN encoder-decoder based models (Bahdanau et al., 2014) have shown improved results in a variety of tasks. Yang et al. (2016) presented a review network in solving the image captioning task, which produces a compact thought vector via reviewing all the input information encoded by the encoder. Mei et al. (2015) proposed attentional RNN encoder-decoder based model by introducing two layers of attention to model content selection and surface realization. More close to our work, Wen et al. (2016b) proposed an attentive encoder-decoder based generator, which applied the attention mechanism over the slot-value pairs. The model indicated a domain scalability when a very limited proportion of training data is available.

3 Recurrent Neural Language Generator

The recurrent language generator proposed in this paper is based on a neural net language generator (Wen et al., 2016b) which consists of three components: an encoder to incorporate the target meaning representation as the model inputs, an aggregator to align and control the encoded information, and a decoder to generate output sentences. The generator architecture is shown in Figure 1. While the decoder typically uses an RNN model, there is a variety of ways to choose the encoder because it depends on the nature of the meaning representation and the interaction between semantic elements. The encoder first encodes the input meaning representation, then the aggregator with a feature selecting or an attention-based mechanism is used to aggregate and select the input semantic elements. The input to the RNN decoder at each time step is a 1-hot encoding of a token[†] and the aggregated input vector. The output of RNN decoder represents the probability distribution of the next token given the previous token, the dialogue act representation, and the current hidden state. At generation time, we can sample from this conditional distribution to obtain the next token in a generated sentence, and feed it as the next input to the RNN decoder. This process finishes when a stop sign is generated (Karpathy and Fei-Fei, 2015), or some constraint is reached (Zhang and Lapata, 2014). The network can generate a sequence of tokens which can be lexicalized[‡] to form the required utterance.

[†] Input texts are delexicalized in which slot values are replaced by its corresponding slot tokens.

[‡] The process in which slot token is replaced by its value.

3.1 Gated Recurrent Unit

The encoder and decoder of the proposed model use a Gated Recurrent Unit (GRU) network proposed by Bahdanau et al. (2014), which maps an input sequence $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, .., \mathbf{w}_T]$ to a sequence of states $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, .., \mathbf{h}_T]$ as follows:

$$\begin{aligned}
\mathbf{r}_i &= \sigma(\mathbf{W}_{rw}\mathbf{w}_i + \mathbf{W}_{rh}\mathbf{h}_{i-1}) \\
\mathbf{u}_i &= \sigma(\mathbf{W}_{uw}\mathbf{w}_i + \mathbf{W}_{uh}\mathbf{h}_{i-1}) \\
\tilde{\mathbf{h}}_i &= \tanh(\mathbf{W}_{hw}\mathbf{w}_i + \mathbf{r}_i \odot \mathbf{W}_{hh}\mathbf{h}_{i-1}) \\
\mathbf{h}_i &= \mathbf{u}_i \odot \mathbf{h}_{i-1} + (1 - \mathbf{u}_i) \odot \tilde{\mathbf{h}}_i
\end{aligned} \tag{1}$$

where: $\odot$ denotes the element-wise multiplication, $\mathbf{r}_i$ and $\mathbf{u}_i$ are called the reset and update gates respectively, and $\tilde{\mathbf{h}}_i$ is the candidate activation.

3.2 Encoder

The encoder uses a separate parameterization of the slots and values. It encodes the source information into a distributed vector representation $\mathbf{z}_i$ which is a concatenation of embedding vector representation of each slot-value pair, and is computed by:

$$\mathbf{z}_i = \mathbf{o}_i \oplus \mathbf{v}_i \tag{2}$$

where: $\mathbf{o}_i$ and $\mathbf{v}_i$ are the i-th slot and value embedding, respectively. The *i* index runs over the given slot-value pairs. In this study, we use a Bidirectional GRU (Bi-GRU) to encode the sequence of slot-value pairs[§] embedding. The Bi-GRU consists of forward and backward GRUs. The forward GRU reads the sequence of slot-value pairs from left-to-right and calculates the forward hidden states $(\overrightarrow{s_1}, .., \overrightarrow{s_K})$. The backward GRU reads the slot-value pairs from right-to-left, resulting in a sequence of backward hidden states $(\overleftarrow{s_1}, .., \overleftarrow{s_K})$. We then obtain the sequence of hidden states $\mathbf{S} = [\mathbf{s}_1, \mathbf{s}_2, .., \mathbf{s}_K]$ where $\mathbf{s}_i$ is a sum of the forward hidden state $\overrightarrow{s_i}$ and the backward one $\overleftarrow{s_i}$ as follows:

$$\mathbf{s}_i = \overrightarrow{s_i} + \overleftarrow{s_i} \tag{3}$$

3.3 Aggregator

The Aggregator consists of two components: an Aligner and a Refiner. The Aligner computes the dialogue act representation while the choices for Refiner can be varied.

Firstly, the Aligner calculates dialogue act embedding $\mathbf{d}_t$ as follows:

$$\mathbf{d}_t = \mathbf{a} \oplus \sum\nolimits_i \alpha_{t,i}\mathbf{s}_i \tag{4}$$

where: $\mathbf{a}$ is vector embedding of the action type, $\oplus$ is vector concatenation, and $\alpha_{t,i}$ is the weight of *i*-th slot-value pair calculated by the attention mechanism:

$$\begin{aligned}
\alpha_{t,i} &= \frac{\exp(e_{t,i})}{\sum_j \exp(e_{t,j})} \\
e_{t,i} &= a(\mathbf{s}_i, \mathbf{h}_{t-1}) \\
a(\mathbf{s}_i, \mathbf{h}_{t-1}) &= \mathbf{v}_a^\top \tanh(\mathbf{W}_a\mathbf{s}_i + \mathbf{U}_a\mathbf{h}_{t-1})
\end{aligned} \tag{5}$$

where: $a(.,.)$ is an alignment model,$\mathbf{v}_a, \mathbf{W}_a, \mathbf{U}_a$ are the weight matrices to learn.

Secondly, the Refiner calculates the new input $\mathbf{x}_t$ based on the original input token $\mathbf{w}_t$ and the DA representation. There are several choices to formulate the Refiner such as gating mechanism or attention mechanism. For each input token $\mathbf{w}_t$, the selected mechanism module computes the new input $\mathbf{x}_t$ based on the dialog act representation $\mathbf{d}_t$ and the input token embedding $\mathbf{w}_t$, and is formulated by:

$$\mathbf{x}_t = f_R(\mathbf{d}_t, \mathbf{w}_t) \tag{6}$$

where: f_R is a refinement function, in which each input token is refined (or filtered) by the dialogue act attention information before putting into the RNN decoder. By this way, we can represent the whole sentence based on this refined input using RNN model.

Attention Mechanism: Inspired by work of Cui et al. (2016), in which an attention-over-attention was introduced in solving reading comprehension tasks, we place another attention applied for Refiner over the attentive Aligner, resulting in a model Attentional Refiner over Attention (ARoA).

- ARoA with Vector (*ARoA-V*): We use a simple attention where each input token representation is weighted according to dialogue act attention as follows:

$$\begin{aligned}
\beta_t &= \sigma(\mathbf{V}_{ra}^\top \mathbf{d}_t) \\
f_R(\mathbf{d}_t, \mathbf{w}_t) &= \beta_t * \mathbf{w}_t
\end{aligned} \tag{7}$$

  where: $\mathbf{V}_{ra}$ is a refinement attention vector which is used to determine the dialogue act attention strength, and σ is sigmoid function to normalize the weight β_t between 0 and 1.

[§] We treat the set of slot-value pairs as a sequence and use the order specified by slot's name (*e.g.*, slot *area* comes first, *price* follows *area*). We have tried treating slot-value pair sequence as natural order as appear in the DA, which even yielded worse results.

- ARoA with Matrix (*ARoA-M*): ARoA-V uses only a vector $\mathbf{V}_{ra}$ to weight the DA attention. It may be better to use a matrix to control the attention information. The Equation 7 is modified as follows:

$$\begin{aligned} \mathbf{V}_{ra} &= \mathbf{W}_{aw}\mathbf{w}_t \\ \beta_t &= \sigma(\mathbf{V}_{ra}^\top \mathbf{d}_t) \\ f_R(\mathbf{d}_t, \mathbf{w}_t) &= \beta_t * \mathbf{w}_t \end{aligned} \tag{8}$$

  where: $\mathbf{W}_{aw}$ is a refinement attention matrix.

- ARoA with Context (*ARoA-C*): The attention in ARoA-V and ARoA-M may not capture the relationship between multiple tokens. In order to add context information into the attention process, we modify the attention weights in Equation 8 with additional history information $\mathbf{h}_{t-1}$:

$$\begin{aligned} \mathbf{V}_{ra} &= \mathbf{W}_{aw}\mathbf{w}_t + \mathbf{W}_{ah}\mathbf{h}_{t-1} \\ \beta_t &= \sigma(\mathbf{V}_{ra}^\top \mathbf{d}_t) \\ f_R(\mathbf{d}_t, \mathbf{w}_t, \mathbf{h}_{t-1}) &= \beta_t * \mathbf{w}_t \end{aligned} \tag{9}$$

  where: $\mathbf{W}_{aw}, \mathbf{W}_{ah}$ are parameters to learn, $\mathbf{V}_{ra}$ is the refinement attention vector same as above, which contains both DA attention and context information.

Gating Mechanism: We use simple element-wise operators (multiplication or addition) to gate the information between the two vectors $\mathbf{d}_t$ and $\mathbf{w}_t$ as follows:

- Multiplication (*GR-MUL*): The element-wise multiplication plays a part in word-level matching which learns not only the vector similarity, but also preserve information about the two vectors:

$$f_R(\mathbf{d}_t, \mathbf{w}_t) = \mathbf{W}_{gd}\mathbf{d}_t \odot \mathbf{w}_t \tag{10}$$

- Addition (*GR-ADD*):

$$f_R(\mathbf{d}_t, \mathbf{w}_t) = \mathbf{W}_{gd}\mathbf{d}_t + \mathbf{w}_t \tag{11}$$

3.4 Decoder

The decoder uses a simple GRU model as described in Section 3.1. In this work, we propose to apply the DA representation and the refined inputs deeper into the GRU cell. Firstly, the GRU reset and update gates can be further influenced on the DA attentive information $\mathbf{d}_t$. The reset and update gates are modified as follows:

$$\begin{aligned} \mathbf{r}_t &= \sigma(\mathbf{W}_{rx}\mathbf{x}_t + \mathbf{W}_{rh}\mathbf{h}_{t-1} + \mathbf{W}_{rd}\mathbf{d}_t) \\ \mathbf{u}_t &= \sigma(\mathbf{W}_{ux}\mathbf{x}_t + \mathbf{W}_{uh}\mathbf{h}_{t-1} + \mathbf{W}_{ud}\mathbf{d}_t) \end{aligned} \tag{12}$$

where: $\mathbf{W}_{rd}$ and $\mathbf{W}_{ud}$ act like background detectors that learn to control the style of the generating sentence. By this way, the reset and update gates learn not only the long-term dependency but also the attention information from the dialogue act and the previous hidden state. Secondly, the candidate activation $\tilde{\mathbf{h}}_t$ is also modified to depend on the DA representation as follows:

$$\begin{aligned} \tilde{\mathbf{h}}_t = \tanh(&\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{r}_t \odot \mathbf{W}_{hh}\mathbf{h}_{t-1} \\ &+\mathbf{W}_{hd}\mathbf{d}_t) + \tanh(\mathbf{W}_{dc}\mathbf{d}_t) \end{aligned} \tag{13}$$

The hidden state is then computed by:

$$\mathbf{h}_t = \mathbf{u}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \odot \tilde{\mathbf{h}}_t \tag{14}$$

Finally, the output distribution is computed by applying a softmax function g, and the distribution is sampled to obtain the next token,

$$\begin{aligned} P(w_{t+1} \mid w_t, w_{t-1}, ...w_0, \mathbf{z}) &= g(\mathbf{W}_{ho}\mathbf{h}_t) \\ w_{t+1} &\sim P(w_{t+1} \mid w_t, w_{t-1}, ...w_0, \mathbf{z}) \end{aligned} \tag{15}$$

3.5 Training

The objective function was the negative log-likelihood and simply computed by:

$$F(\theta) = -\sum_{t=1}^{T} \mathbf{y}_t^\top \log \mathbf{p}_t \tag{16}$$

where: $\mathbf{y}_t$ is the ground truth word distribution, $\mathbf{p}_t$ is the predicted word distribution, T is length of the input sequence. The proposed generators were trained by treating each sentence as a mini-batch with l_2 regularization added to the objective function for every 10 training examples. The pre-trained word vectors (Pennington et al., 2014) were used to initialize the model. The generators were optimized by using stochastic gradient descent and back propagation through time (Werbos, 1990). To prevent over-fitting, we implemented early stopping using a validation set as suggested by Mikolov (2010).

3.6 Decoding

The decoding consists of two phases: (i) over-generation, and (ii) reranking. In the over-generation, the generator conditioned on the given

Table 2: Comparison performance on four datasets in terms of the BLEU and the error rate ERR(%) scores; **bold** denotes the best and ***italic*** shows the second best model. The results were produced by training each network on 5 random initialization and selected model with the highest validation BLEU score. $^\sharp$ denotes the Attention-based Encoder-Decoder model.

Model	**Restaurant**		**Hotel**		**Laptop**		**TV**	
	BLEU	ERR	BLEU	ERR	BLEU	ERR	BLEU	ERR
HLSTM	0.7466	0.74%	0.8504	2.67%	0.5134	1.10%	0.5250	2.50%
SCLSTM	0.7525	0.38%	0.8482	3.07%	0.5116	0.79%	0.5265	2.31%
ENCDEC$^\sharp$	0.7398	2.78%	0.8549	4.69%	0.5108	4.04%	0.5182	3.18%
GR-ADD$^\sharp$	0.7742	0.59%	0.8848	1.54%	***0.5221***	***0.54%***	0.5348	0.77%
GR-MUL$^\sharp$	0.7697	0.47%	0.8854	1.47%	0.5200	1.15%	0.5349	0.65%
ARoA-V$^\sharp$	0.7667	***0.32%***	0.8814	**0.97%**	0.5195	0.56%	***0.5369***	0.81%
ARoA-M$^\sharp$	**0.7755**	**0.30%**	**0.8920**	***1.13%***	**0.5223**	**0.50%**	**0.5394**	**0.60%**
ARoA-C$^\sharp$	***0.7745***	0.45%	***0.8878***	1.31%	0.5201	0.88%	0.5351	***0.63%***

Table 3: Comparison performance of variety of the proposed models on four dataset in terms of the BLEU and the error rate ERR(%) scores; **bold** denotes the best and ***italic*** shows the second best model. The first two models applied gating mechanism to Refiner component while the last three models used attention over attention mechanism. The results were averaged over 5 randomly initialized networks.

Model	**Restaurant**		**Hotel**		**Laptop**		**TV**	
	BLEU	ERR	BLEU	ERR	BLEU	ERR	BLEU	ERR
GR-ADD	0.7685	0.63%	***0.8838***	1.67%	***0.5194***	***0.66%***	***0.5344***	0.75%
GR-MUL	0.7669	***0.61%***	0.8836	1.40%	0.5184	1.01%	0.5328	0.73%
ARoA-V	0.7673	0.62%	0.8817	***1.27%***	0.5185	0.73%	0.5336	0.68%
ARoA-M	**0.7712**	**0.50%**	**0.8851**	**1.14%**	**0.5201**	**0.62%**	**0.5350**	**0.62%**
ARoA-C	***0.7690***	0.70%	0.8835	1.44%	0.5181	0.78%	0.5307	***0.64%***

DA uses a beam search to generate a set of candidate responses. In the reranking, the cost of the generator is computed to form the reranking score R as follows:

$$R = -\sum_{t=1}^{T} \mathbf{y}_t^\top \log \mathbf{p}_t + \lambda ERR \tag{17}$$

where λ is a trade off constant and is set to a large value in order to severely penalize nonsensical outputs. The slot error rate ERR, which is the number of slots generated that is either redundant or missing, and is computed by:

$$ERR = \frac{p+q}{N} \tag{18}$$

where: N is the total number of slots in DA, and p, q is the number of missing and redundant slots, respectively. Note that the ERR reranking criteria cannot handle arbitrary slot-value pairs such as *binary* slots or slots that take the *dont_care* value because these slots cannot be delexicalized and matched.

4 Experiments

We conducted an extensive set of experiments to assess the effectiveness of our model using several metrics, datasets, and model architectures, in order to compare to prior methods.

4.1 Datasets

We assessed the proposed models using four different NLG domains: finding a restaurant, finding a hotel, buying a laptop, and buying a television. The Restaurant and Hotel were collected in (Wen et al., 2015b) which contain around 5K utterances and 200 distinct DAs. The Laptop and TV datasets have been released by Wen et al. (2016a). These datasets contain about 13K distinct DAs in the Laptop domain and 7K distinct DAs in the TV. Both Laptop and TV datasets have a much larger input space but only one training example for each DA so that the system must learn partial realization of concepts and be able to recombine and apply them to unseen DAs. As a result, the NLG tasks for the Laptop and TV datasets become much harder.

4.2 Experimental Setups

The generators were implemented using the TensorFlow library (Abadi et al., 2016) and trained by partitioning each of the datasets into training, validation and testing set in the ratio 3:1:1. The hidden layer size was set to be 80 for all cases, and the generators were trained with a 70% of dropout rate. We perform 5 runs with different random initialization of the network and the training is terminated by using early stopping as described in Section 3.5. We select a model that yields the highest BLEU score on the validation set as shown in Table 2. Since the trained models can differ depending on the initialization, we also report the results which were averaged over 5 randomly initialized networks. Note that, except the results reported in Table 2, all the results shown were averaged over 5 randomly initialized networks. The decoder procedure used beam search with a beam width of 10. We set λ to 1000 to severely discourage the reranker from selecting utterances which contain either redundant or missing slots. For each DA, we over-generated 20 candidate utterances and selected the top 5 realizations after reranking. Moreover, in order to better understand the effectiveness of our proposed methods, we (1) trained the models on the Laptop domain with a varied proportion of training data, starting from 10% to 100% (Figure 3), and (2) trained general models by merging all the data from four domains together and tested them in each individual domain (Figure 4) .

4.3 Evaluation Metrics and Baselines

The generator performance was assessed by using two objective evaluation metrics: the BLEU score and the slot error rate ERR. Both metrics were computed by adopting code from an open source benchmark NLG toolkit[¶]. We compared our proposed models against three strong baselines from the open source benchmark toolkit. The results have been recently published as an NLG benchmarks by the Cambridge Dialogue Systems Group[¶], including *HLSTM*, *SCLSTM*, and *ENCDEC* models.

5 Results and Analysis

5.1 Results

We conducted extensive experiments on the proposed models with varied setups of Refiner and compared against the previous methods. Overall, the proposed models consistently achieve the better performances regarding both evaluation metrics across all domains.

Table 2 shows a comparison between the *AREncDec* based models (the models with $\sharp$ symbol) in which the proposed models significantly reduce the slot error rate across all datasets by a large margin about 2% to 4% that are also improved performances on the BLEU score when comparing the proposed models against the previous approaches. Table 3 further shows the stable strength of our models since the results' pattern stays unchanged compared to those in Table 2. The *ARoA-M* model shows the best performance over all the four domains, while it is an interesting observation that the *GR-ADD* model with simple addition operator for Refiner obtains the second best performance. All these prove the importance of the proposed component Refiner in aggregating and selecting the attentive information.

Figure 3 illustrates a comparison of four models (*ENCDEC*, *SCLSTM*, *ARoA-M*, and *GR-ADD*) which were trained from scratch on the laptop dataset in a variety of proportion of training data, from 10% to 100%. It clearly shows that the BLEU increases while the slot error rate decreases as more training data was provided. Figure 4 presents a comparison performance of general models as described in Section 4.2. Not surprisingly, the two proposed models still obtain higher the BLEU score, while the *ENCDEC* has difficulties in reducing the ERR score in all cases. Both the proposed models show their ability to generalize in the unseen domains (TV and Laptop datasets) since they consistently outperform the previous methods no matter how much training data was fed or how training method was used. These indicate the relevant contribution of the proposed component Refiner to the original AREncDec architecture, in which the Refiner with gating or attention mechanism can effectively aggregate the information before putting them into the RNN decoder.

Figure 5 shows a different attention behavior of the proposed models in a sentence. While all the three models could attend the slot tokens and their surrounding words, the *ARoA-C* model with context shows its ability in attending the consecutive words. Table 4 shows comparison of responses generated for some DAs between different models.

[¶] https://github.com/shawnwun/RNNLG

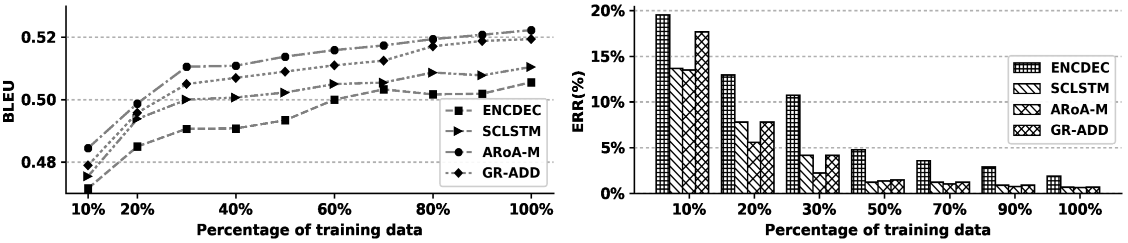

Figure 3: Performance comparison of the four models trained on Laptop (unseen) domain.

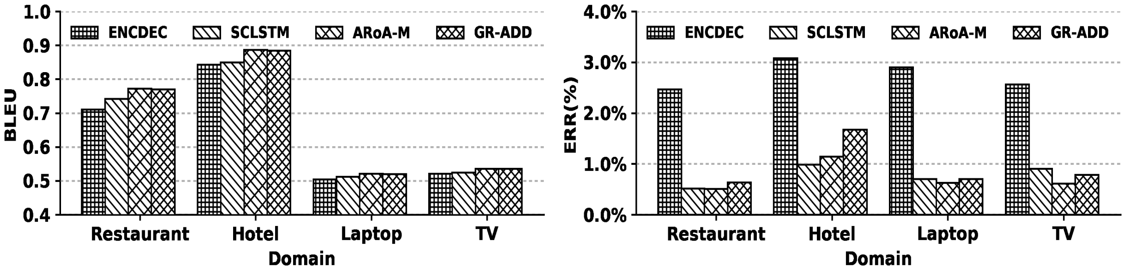

Figure 4: Performance comparison of the general models on four different domains.

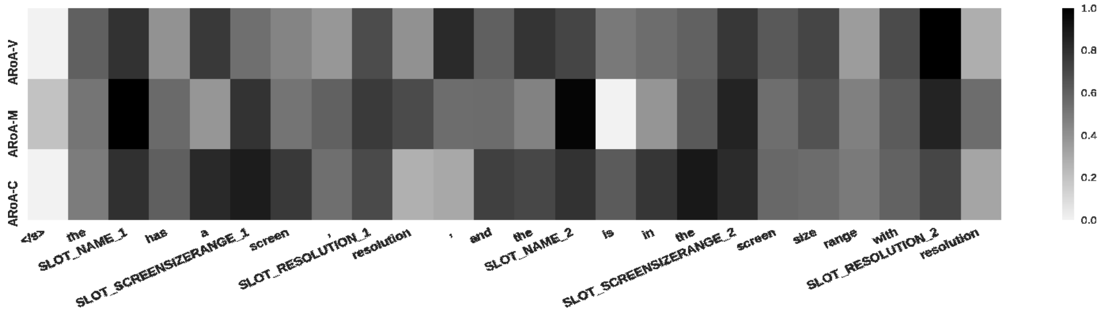

Figure 5: A comparison on attention behavior of three models in a sentence on given ***DA*** with sequence of slots [*Name_1, ScreenSizeRange_1, Resolution_1, Name_2, ScreenSizeRange_2, Resolution_2*].

The previous approaches (*ENCDEC*, *HLSTM*) still have missing and misplaced information, whereas the proposed models can generate complete and correct-order sentences.

6 Conclusion and Future Work

We present an extension of an Attentional RNN Encoder-Decoder model named Encoder-Aggregator-Decoder, in which a Refiner component is introduced to select and aggregate the semantic elements produced by the encoder. We also present several different choices of gating and attention mechanisms which can be effectively applied to the Refiner. The extension, which is easily integrated into an RNN Encoder-Decoder, shows its ability to refine the inputs and control the flow information before putting them into the RNN decoder. We evaluated the proposed model on four domains and compared to the previous generators. The proposed models empirically show consistent improvement over the previous methods in both BLEU and ERR evaluation metrics. In the future, it would be interesting to further investigate hybrid models which integrate gating and attention mechanisms in order to leverage the advantages of both mechanisms.

Acknowledgements

This work was supported by JSPS Kakenhi Grant Number JP15K16048 and the grant for the research cooperation between JAIST and TIS. The first author would like to thank the Vietnamese Government Scholarship (911 project).

Table 4: Comparison of top responses generated for some input dialogue acts between different models. Errors are marked in color (missing, misplaced slot-value pair). † and $^{\natural}$ denotes the baselines and the proposed models, respectively.

Model	Generated Responses in Laptop domain
Input DA	*compare(name='aristaeus 59'; screensizerange='large'; resolution='1080p'; name='charon 61'; screensizerange='medium'; resolution='720p')*
Reference	*Compared to aristaeus 59 which is in the large screen size range and has 1080p resolution, charon 61 is in the medium screen size range and has 720p resolution. Which one do you prefer?*
ENCDEC†	the aristaeus 59 has a large screen , the charon 61 has a medium screen and 1080p resolution [1080p, 720p]
HLSTM†	the aristaeus 59 has a large screen size range and has a 1080p resolution and 720p resolution [720p, charon 61, medium]
SCLSTM†	the aristaeus 59 has a large screen and 1080p resolution , the charon 61 has a medium screen and 720p resolution
GR-ADD$^{\natural}$	the aristaeus 59 has a large screen size and 1080p resolution , the charon 61 has a medium screen size and 720p resolution
GR-MUL$^{\natural}$	the aristaeus 59 has a large screen size and 1080p resolution , the charon 61 has a medium screen size and 720p resolution .
ARoA-V$^{\natural}$	the aristaeus 59 has a large screen size and 1080p resolution , the charon 61 has a medium screen size , and has a 720p resolution
ARoA-M$^{\natural}$	the aristaeus 59 has a large screen and 1080p resolution , the charon 61 has a medium screen and 720p resolution
ARoA-C$^{\natural}$	the aristaeus 59 has a large screen size and 1080p resolution , the charon 61 has a medium screen size range and 720p resolution

References

Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* .

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

A. Cheyer and D. Guzzoni. 2014. Method and apparatus for building an intelligent automated assistant. US Patent 8,677,377. https://www.google.com/patents/US8677377.

Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. 2016. Attention-over-attention neural networks for reading comprehension. *arXiv preprint arXiv:1607.04423* .

Ondřej Dušek and Filip Jurčíček. 2016a. A context-aware natural language generator for dialogue systems. *arXiv preprint arXiv:1608.07076* .

Ondřej Dušek and Filip Jurčíček. 2016b. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. *arXiv preprint arXiv:1606.05491* .

Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference CVPR*. pages 3128–3137.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* .

François Mairesse and Steve Young. 2014. Stochastic language generation in dialogue using factored language models. *Computational Linguistics* .

Hongyuan Mei, Mohit Bansal, and Matthew R Walter. 2015. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. *arXiv preprint arXiv:1509.00838* .

Tomas Mikolov. 2010. Recurrent neural network based language model.

Alice H Oh and Alexander I Rudnicky. 2000. Stochastic language generation for spoken dialogue systems. In *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational systems-Volume 3*. Association for Computational Linguistics, pages 27–32.

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

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*. volume 14, pages 1532–43.

Adwait Ratnaparkhi. 2000. Trainable methods for surface natural language generation. In *Proceedings* *of the 1st North American Chapter of the Association for Computational Linguistics Conference*. Association for Computational Linguistics, Stroudsburg, PA, USA, NAACL 2000, pages 194–201. http://dl.acm.org/citation.cfm?id=974305.974331.

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

Tsung-Hsien Wen, Milica Gašić, Dongho Kim, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015a. Stochastic Language Generation in Dialogue using Recurrent Neural Networks with Convolutional Sentence Reranking. In *Proceedings SIGDIAL*. Association for Computational Linguistics.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina M Rojas-Barahona, Pei-Hao Su, David Vandyke, and Steve Young. 2016a. Multi-domain neural network language generation for spoken dialogue systems. *arXiv preprint arXiv:1603.01232* .

Tsung-Hsien Wen, Milica Gašic, Nikola Mrkšic, Lina M Rojas-Barahona, Pei-Hao Su, David Vandyke, and Steve Young. 2016b. Toward multi-domain language generation using recurrent neural networks .

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015b. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of EMNLP*. Association for Computational Linguistics.

Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. 2016c. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562* .

Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* 78(10):1550–1560.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* .

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*. volume 14, pages 77–81.

Zhilin Yang, Ye Yuan, Yuexin Wu, William W Cohen, and Ruslan R Salakhutdinov. 2016. Review networks for caption generation. In *Advances in Neural Information Processing Systems*. pages 2361–2369.

Xingxing Zhang and Mirella Lapata. 2014. Chinese poetry generation with recurrent neural networks. In *EMNLP*. pages 670–680.