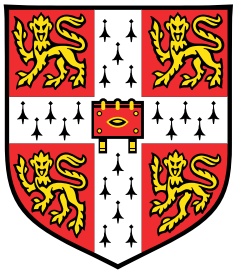
Augmenting Natural Language Generation with external memory modules in Spoken Dialogue Systems



Minglong Sun

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Darwin College

August 2018

Declaration

I, Minglong Sun of Darwin College, being a candidate for the MPhil in Machine Learning, Speech and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 14285

Minglong Sun
August 2018

Acknowledgements

I would like to thank my supervisor Dr Milica Gašić who is really responsible and supportive throughout the whole project and kindly provides me the opportunity to work in the friendly and collaborative Dialogue System Group laboratory. I have benefited a lot from the great meetings with her on project ideas and writing thesis. I would also like to thank Bo-Hsiang Tseng for his valuable ideas and guidance for this project and for his patience in answering my questions about Pytorch and my models. I also want to thank Yan-chen Wu and Yinpei Dai for their kind and inspiring discussions with me.

Abstract

Semantically Conditioned LSTM (SC-LSTM) is one of the state-of-the-art models in the Natural Language Generation of the Spoken Dialogue Systems. Though it has a Dialogue Act (DA) cell which enables the generations to condition on DA information efficiently, its generations are still imperfect when the input DA has multiple dialogue act types and semantic slots. Additionally, SC-LSTM lacks the ability to exploit the cross-sentence information which is shared between dialogue turns of a dialogue. Therefore, we study and compare the use of three different types of memory modules on this SC-LSTM system with the aim to generate more accurate and natural sentences. We find the memory module which exploits the context information from recent dialogue turns can achieve the best performances in terms of slot error rate and BLEU score. This indicates that an external memory module with better model designs can further improve generations' quality for a multi-turn dialogue generation task.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Natural Language Generation (NLG) is an important task in Spoken Dialogue Systems in which input Dialogue Acts (DA) are transformed into human text sentences. Recurrent Neural Network (RNN) model is a type of Deep Neural Network (DNN) unfolded through time and has the potential to model long-range dependencies hence it is a suitable sequence model for language modelling (RNNLM [16]). On grounds of this, Long short-term memory (LSTM) model is introduced to mitigate the vanishing gradient problem in RNN [8] and Semantically-controlled LSTM (SC-LSTM) [26] which is an LSTM model conditioning on DA information through control gates has achieved state-of-art performances in NLG tasks.

While LSTM-based models excel in sequence-to-sequence generation tasks and can be viewed as a kind of sequential memory models [30], it has been proven that LSTM-based models have difficulties in learning non-sequential contexts in a long sentence on their own [13] which means more attention will be placed on local/more recent context information when generating word tokens at each time step. Specifically, in our NLG task which uses SC-LSTM based decoders, we observe that when an input DA has multiple semantic slots or contains different types of dialogue actions, the generations for the DA are not accurate enough and may lack important semantic information.

To mitigate this problem and to augment the generation system, we can exploit the cross-sentence information or recent generations given that some contextual semantic information is shared between dialogue turns in a dialogue. Hence, storing and using this document-level contextual information has the potential to augment current LSTM-based models.

Recently, external-memory based models have been studied extensively in augmenting generation models since a neural network system can retrieve useful information from memory modules. Hence we study the use of memory modules in our NLG task with the aim to make our baseline models more robust and to make generated sentences adapt to their recent histories such that the generated sentences are diverse while accurate.

Chapter 2

Background

2.1 Spoken Dialogue Systems

A Spoken Dialogue Systems (SDS) is a computer system which enables interactions with human users through speech. It has a wide range of applications such as requesting travel information, making reservations and even some advanced tasks such as medical diagnosis. It has become a promising field and has been studied extensively recently. The architecture of an SDS can be summarized in the figure below:

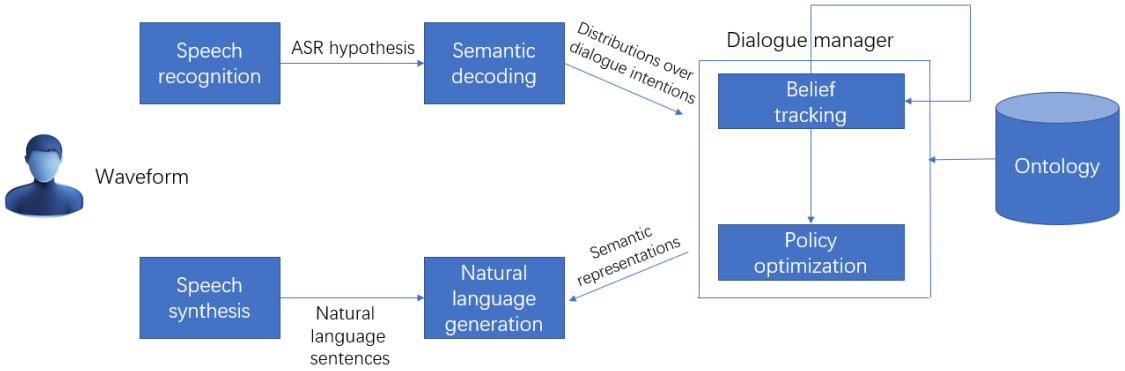


Fig. 2.1 An example figure showing SDS architecture

Each module from above will be briefly discussed in the following subsections.

Ontology

In Spoken Dialogue Systems (SDS), Ontology summarizes the domain knowledge by specifying the properties of domain knowledge and is often used for goal-oriented dialogue systems. It defines the concepts that the system can talk about and can be viewed as a structured

database which contains informable/ requestable semantic slots and their corresponding values for each domain. The informable slots refer to the semantic slots through which users can constrain their search (*acceptscards*, *kids-allowed* for example) while requestable slots correspond to the properties through which users can request more information about a domain (such as *postcode* and *phonenumber*).

Ontology plays an important role in several components in the SDS. For example, in dialogue management, domain knowledge from ontologies can be used to check whether a user has provided responses/queries which are fit with system design. Various ontology building approaches have been discussed and compared in papers such as [17] and [19].

Front-end of Spoken Dialogue Systems

The front end of an SDS includes Automatic Speech Recognition (ASR) and text-to-speech (TTS) synthesis:

Given the input speech which consists of a sequence of changing time-signal, the ASR will transform it into a sequence of symbols such as texts in the dialogue systems. Hence, various sequence-to-sequence models are commonly used for this task. The output from ASR will then be the input to the Semantic Decoding module.

Given the model parameters from the trained models, the Viterbi algorithm is commonly used in finding the most likely state sequence (top hypothesis). However, since SDS is often applied in environments with background noise (busy street, noisy cars for example), the top hypothesis may not be correct. Hence, keeping a set of good candidate hypothesis to be jointly considered in Semantic Decoding will achieve more robust performances. To achieve this, several decoding strategies such as N-best list, word lattice and confusion network have been studied. The N-best approach saves the results as a list of candidates while the word lattice approach can have a more compact representation and is more preferable in large vocabulary recognition task [28]. In addition, Confusion network which based on weighted directed word-lattices is another popular approach which provides a way of compact representation while maintains model robustness in terms of recognition accuracy [28]. These recognition results are then passed to the Semantic Decoding module.

Text-to-speech synthesis is the last step in the system which transforms human text sentences from NLG module into speech waveforms. There are mainly two types of TTS synthesis: concatenative speech synthesis which concatenates segments of recorded speech and statistical speech synthesis which generates speech through statistical models.

Semantic decoding

Semantic decoding (or known as Spoken Language Understanding) is the process of transforming the recognition results from ASR into more compact representations of users' intentions and it is the guidance of system's response. The input data can be top ASR hypothesis, N-best hypothesis or a confusion network as discussed above.

Semantic decoding can be viewed as a multi-class classification task and hence a set of Support Vector Machine classifiers can be used. In addition, it can also be viewed as a sequence-to-sequence learning task where linear-chain Conditional Random Fields methods have been shown to be capable of this task [14]. However, both SVM and CRF based approaches are discriminative models which rely heavily on the semantically annotated training data whose collecting process can be expansive and time-consuming [14].

On these grounds, use of RNN in semantic decoding has been studied and has achieved competitive results in [15]. Though RNN methods excel in learning context information, they suffer from vanishing gradient problem and hence a more advanced RNN-based Long Short-Term Memory network is introduced with outstanding model performances in the SLU task [29].

Dialogue Management

Dialogue management is a crucial component in the SDS which controls the system's behaviour at each turn. Two key elements of dialogue management are belief tracking and policy optimization which will be briefly discussed here.

We first introduce the Dialogue State which refers to a full representation of what a user wants at any point in a dialogue. For instance, a dialogue state can be interpreted as a combination of three components: goal, requested slot and method. Since the output hypothesis from ASR may contain errors as discussed above and that the Semantic Decoding can be imperfect, sometimes it is difficult for the system to decide users' real intentions. Hence, a distribution over all possible dialogue states is computed to get better system performance and this is known as 'belief state'. Furthermore, the system can update this distribution (belief state) after a new turn is observed such that users' intentions can be tracked over sequences of turns, and this process is called belief state tracking.

Apart from belief tracking, we also talk about the policy optimization in the dialogue management. The dialogue can be thought to be a control problem where the input is the belief state from the belief state tracker and the control is the actions that the system takes (what system replies to users) with the aim to automatically optimize the system actions (to find the optimal policy). One influential approach for the tasks of belief tracking and policy

optimization is the Partially Observable Markov Decision Process (POMDP) [27] which has been shown to have the capability for automatic policy optimization and robust performance in coping with speech recognition errors [27]. Considering the fact that exact belief states update with POMDP is intractable, Hidden Information State[31] and Bayesian Update of Dialogue State[22] approaches using approximate belief tracking based on POMDP have been proposed with competitive model performances.

Natural Language Generation

In the process of Natural Language Generation, the output Dialogue Acts (DA) from the Dialogue Management module are transformed into human text sentences. The input DA encode intentions of either the system or the users and can be viewed as approximate representations of dialogue utterances. A DA is made up of two components: dialogue act types (user/system general intention) and semantic slots with corresponding values (from the ontology). An example of a simple DA from the SF dataset is listed below which has *inform* dialogue act type and two semantic slots (*pricerange* and *kidsallowed*):

inform(pricerange=cheap; kidsallowed=yes)

In an NLG task, they are encoded into 1-hot representation vectors (Semantic Representation vectors) which are the input to an NLG model. For each Semantic Representation (SR), the NLG model will output a sequence of 1-hot representation vectors which correspond to delexicalized tokens (explained as below). These delexicalized tokens will then be lexicalized into words to form a human text sentence [26]. The aim of an NLG model is to generate natural, diverse and accurate sentences given the input SR vectors. This project focuses on improving the NLG process and we further discuss this in later sections.

The delexicalized tokens mentioned above refer to the generic notations for the semantic slots from the ontology and their corresponding values in a sentence. For example, *Tapas restaurant is in the centre of the town* will be delexicalized to *slot-name is in the slot-area of the town*. And the delexicalized tokens can be lexicalized back through the vocabulary dictionary. Delexicalization enables the transfer learning from one example to another [7], which means the model can exploit the similarities between training examples from the same 'template' hence it is popular in the NLG task of an SDS.

2.2 State-of-the-art Natural Language Generation models in Spoken Dialogue Systems

Traditionally, two steps should be followed in Natural Language Generation (NLG), namely, sentence planning and surface realization. Sentence planning guides the general order and structure of sentence generations while surface realization realizes the sentence structure into the output utterances. Previous approaches including rule-based (template-based) methods , corpus-based methods and trainable generator approach have been studied for this task. However, rule-based approaches fail to generate natural and diverse response due to the frequent repetition of identical output and corpus-based approach rely on annotated alignments which limits the scalability of the dialogue system [26]. Hence, RNN based generation models have been proposed with the aim to cope with above problems. We describe two state-of-the-art LSTM based models following the main idea of [23] and [26] respectively as below since they will serve as baseline models in our project.

2.2.1 Semantically Conditioned LSTM

Semantically Conditioned LSTM (SC-LSTM) [26] is an LSTM based language model which exploits the semantic information when generating word tokens at each time step. It consists of two modules and the first one is a common LSTM module which is demonstrated in fig 2.2.

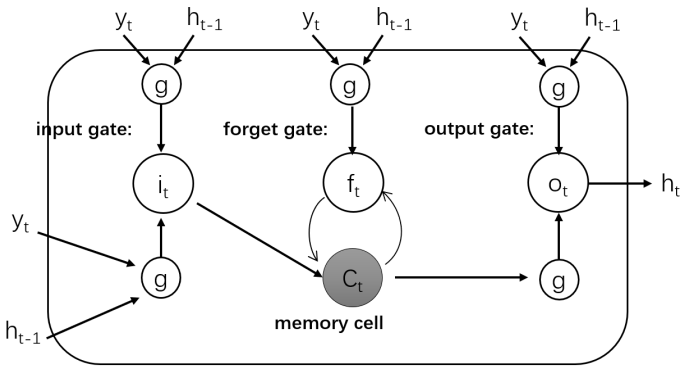


Fig. 2.2 LSTM cell in language modelling

As can be seen, there are three gates which control the information flow: input gate (i_t), forget gate (f_t) and output gate (o_t). And c_t denotes the memory cell which is the key unit in the LSTM cell to control what information is stored and what information is passed to next

time step. Suppose the word token and hidden state vector at time t are denoted as y_t and h_t respectively, then the LSTM mechanism for each time step can be explained as below:

$$i_t = \sigma(\mathbf{W}_{yi} y_t + \mathbf{W}_{hi} h_{t-1}) \tag{2.1}$$
$$f_t = \sigma(\mathbf{W}_{yf} y_t + \mathbf{W}_{hf} h_{t-1}) \tag{2.2}$$
$$o_t = \sigma(\mathbf{W}_{yo} y_t + \mathbf{W}_{ho} h_{t-1}) \tag{2.3}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(\mathbf{W}_{hc} y_t + \mathbf{W}_{ht} h_{t-1}) \tag{2.4}$$
$$h_t = o_t \odot tanh(c_t) \tag{2.5}$$

where $\odot$ denotes the element-wise multiplication, σ denotes the logistic sigmoid functions. W_{yi}, W_{hi} are the weight matrices for the input gates i_t and so are other weight matrices for other gates and memory cells (bias terms have been eliminated for simplicity).

Apart from the LSTM module, semantic information about the Dialogue Acts (DA) is used as the guidance of the generations at each time step and it is realized by a DA cell in SC-LSTM (fig 2.3).

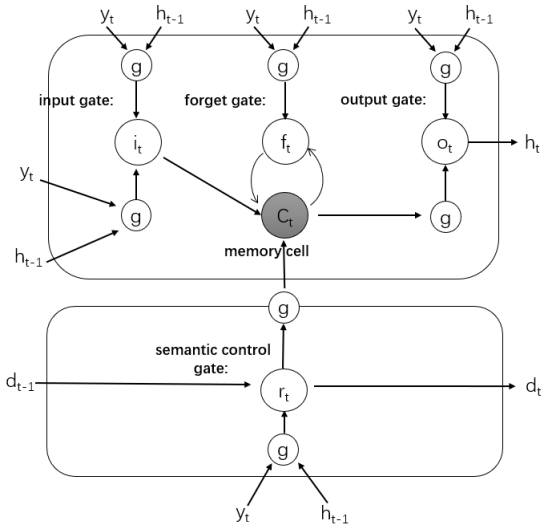


Fig. 2.3 A demonstration of the model architecture of SC-LSTM in [26]

From figure 2.3, the memory cell (c_t) in LSTM cell exploits the information from the bottom DA cell through a control gate. d_t refers to the DA feature vector at time t and the initial DA feature vector d_0 is the 1-hot representation of a dialogue act. To control the information flow for DA feature vectors over different time steps, a read gate (r_t) is employed at each time step. The mechanism of generating new hidden state vectors and DA feature vectors can be further explained by below equations:

$$r_t = \sigma(\mathbf{W}_{yr} y_t + \mathbf{W}_{hr} h_{t-1}) \tag{2.6}$$

$$d_t = r_t \odot tanh(d_{t-1}) \tag{2.7}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(\mathbf{W}_{hc} y_t + \mathbf{W}_{ht} h_{t-1}) + tanh(\mathbf{W}_{dc} d_t) \tag{2.8}$$

The first two equations show the update of DA feature vectors over time and equation 2.8 is the new equation for calculating the memory cell c_t.

The sequence of hidden state vectors h_1, h_2...h_t will then be used to generate the word tokens. This is realized by first mapping h_t to a logits vector whose size is equal to the vocabulary size through a fully connected neural network (for example one layer is used in below function), and then the softmax function g is used on the logits such that the output is a distribution over the word tokens from vocabulary.

$$P(y_{t+1}|y_t, y_{t-1}, ...y_0, y_t) = g(\mathbf{W}_{ho} h_t) \tag{2.9}$$

where $\mathbf{W}_{ho}$ denotes the weight matrix for the fully connected layer.

The last step to obtain a word token is sampling from this distribution (as below) and the words generated at each time step can be combined into the final human text sentence.

$$y_{t+1} \sim P(y_{t+1}|y_t, y_{t-1}, ...y_0, d_t) \tag{2.10}$$

SC-LSTM can effectively use the semantic information through the gating mechanism. It enables the sentence planning and surface realization processes jointly [26] through the bottom DA cell and the top LSTM cell respectively. The main advantage of SC-LSTM is that it can effectively generate reasonably good while diverse and natural sentences hence we develop our memory models based on this model.

Apart from the basic model architecture described above, multiple LSTM cells can be stacked with the aim of better learning high-level features and has been realized through techniques such as skip connections and dropout in [26]. In addition, bidirectional LSTM model has been proven to be more robust since it can better capture the forward and backward contexts. This has been verified in [26] that training two SC-LSTM models for forward and backward context separately can obtain more stable performance in the case of random initialization. Though these advanced architectures can outperform the basic SC-LSTM model, we will mainly focus on using memory modules on the basic SC-LSTM since training on a basic SC-LSTM can be much faster and we can still have a general idea of how different memory models behave in a generation task.

2.2.2 Conditional VAE for SC-LSTM

Recently, semantically conditioned VAE (SCVAE) [23] has been proposed to further improve the SC-LSTM model. The figure from [23] demonstrating model architecture is presented as below:

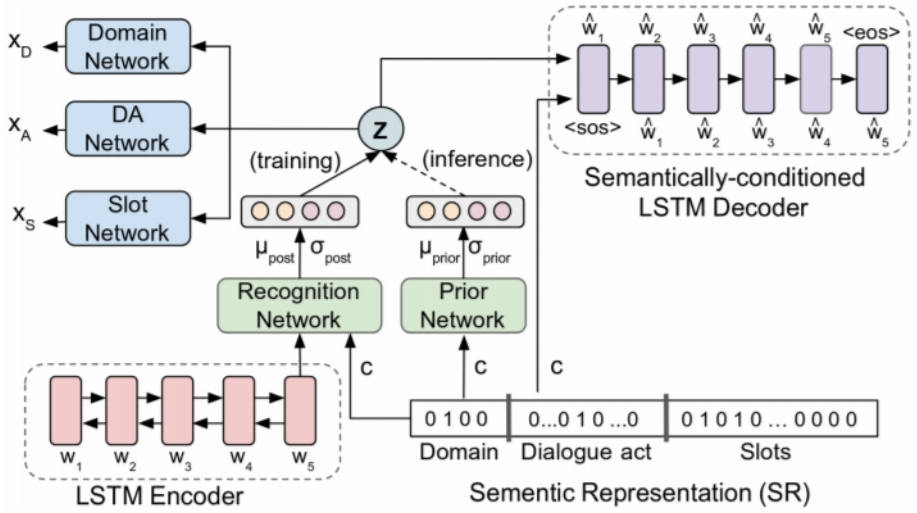


Fig. 2.4 Model structure of combining conditional VAE with SC-LSTM decoder (source: [23])

Variational auto-encoder [11] is an influential generative model which has been widely used in tasks such as image-reconstruction, generating digits etc. In a VAE model, the generation process of generating x from the latent variable z can be viewed as below:

$$P(x|\lambda) = \int P(x|z,\lambda)P(z)dz$$

where λ denotes the model parameters. During inference, we would want to know the true distribution of z given x and by Bayes's rule we get:

$$P(z|x) = \frac{P(x|z,\lambda)P(z,\lambda)}{P(x|\lambda)} = \frac{P(x|z,\lambda)P(z,\lambda)}{\int P(x|z,\lambda)P(z)dz}$$

However, the denominator is intractable which makes calculating exact $P(z|x)$ intractable. Hence, VAE introduces an approximating distribution $Q(z|\theta)$ (θ denotes the parameters of the distribution) for $P(z|x)$ and Kullback-Leibler Divergence between the two distributions $KL(Q(z|\theta)||P(z|x))$ is introduced as a similarity measure for the two distributions. This is included in the model as training loss l to encourage that the approximating distribution (also known as prior) to be close to the true distribution (posterior). Hence a VAE model will be

trained to minimize this l which can be derived as below:

$$l = min_\theta \quad KL[Q(z|\theta)||P(z|x))] \tag{2.11}$$

$$= min_\theta \quad \int Q(z|\theta) log \frac{Q(z|\theta)}{P(z|x)} dz \tag{2.12}$$

$$= min_\theta \quad \int Q(z|\theta) log \frac{Q(z|\theta)}{P(z)P(x|z)} dz \tag{2.13}$$

$$= min_\theta \quad \int Q(z|\theta) log \frac{Q(z|\theta)}{P(z)} dz - \int Q(z|\theta) log P(x|z) dz \tag{2.14}$$

$$= min_\theta \quad KL[Q(z|\theta)||P(z))] - E_{Q(z|\theta)}[log P(x|z)] \tag{2.15}$$

where from equation 2.12 to 2.13 we use the fact that:

$$P(z|x) = \frac{P(x|z)P(z)}{P(x)} \tag{2.16}$$

$$\approx P(x|z)P(z) \tag{2.17}$$

since $P(x)$ is a constant

On grounds of this, a VAE model can further condition on another description of data x and this is known as conditional VAE (CVAE) [20]. In [23], CVAE is combined with the SC-LSTM decoder and the condition in CVAE is the DA information which is encoded into Semantic Representation (SR) vectors. In addition, the 1-hot representation vectors of the words in a target sentence will be passed to an LSTM module and the last hidden state from the LSTM decoder contains the information of the sentence- hence it is called sentence information x which will be the observations in CVAE.

During training, SR and the corresponding generated sentences are available and they are jointly passed through a deep neural network to get the posterior $P(z|x,SR)$. During inference, only SR information is passed through another DNN to get the prior $Q(z|SR)$. In [23], these DNN are referred to as recognition network and prior network respectively. Similar to VAE, a KL-divergence is used to encourage the prior distribution during inference to be as close as possible to the true posterior distribution. The model training loss then becomes:

$$l = min_\theta \quad KL[Q(z|\theta,SR)||P(z|SR))] - E_{Q(z|\theta,SR)}[log P(x|z,SR)] \tag{2.18}$$

In VAE, models are trained through back-propagation but it is hard for models to pass the gradients through the distributions in VAE since z has to be randomly sampled from the distributions. Hence, reparameterization trick is used to ensure that the gradients can pass

through the random node. Mathematically, reparameterization trick is:

$$z \sim N(\mu, \sigma^2) \rightarrow z = \mu + L\varepsilon, \quad \varepsilon \sim N(0, I), \quad \sigma^2 = LL^T$$

where μ, σ are model parameters.

In [23], several regularization techniques are also introduced for a more robust CVAE model. Reconstruction loss from sampled z to domain information and dialogue act information are all included in the total training loss to encourage that z can encode as much information as possible from these resources. Furthermore, KL-annealing technique [2] for improving KL-divergence is also introduced in this model. Interestingly, we did not observe significant improvements using KL-annealing regularization when replicating the experiments hence we will not make further discussions regarding this in the future.

The latent variable z from above will then be passed to the SC-LSTM decoder as the first hidden state. Latent variable z wraps SR information with the sentence information x during training and the CVAE model ensures that z can encode the appropriate sentence information even though only SR information is available during inference. This mechanism enables z to be semantically meaningful [23] and contains useful information for generating sentences. Therefore, this better initialization of the SC-LSTM system can further improve the generations' quality of the SC-LSTM hence it will be our baseline model in section 3.1 .

2.3 Evaluating metrics in NLG

We would want to compare the performances of different NLG models so good evaluation metrics are needed to assess the quality of the generations from the models. Generally, the generated sentences should be natural and accurate. During testing, the generated sentences will be compared to the target sentences. Good generations should be similar to the target sentences while they are also expected to accurately convey the semantic information in the input DA. Ideally, human evaluation should be employed since it will be the best evaluation metric which is in line with human judgment and can accurately reflect the overall quality of a sentence. However, using human evaluation can be time-consuming and expensive [10]. Hence, in this project, we use three popular automatic metrics to measure models' performances, namely slot error rate (ERR), BLEU score and Perplexity following our baseline models. We describe them as below.

Slot error rate

Slot error rate (ERR) is used to measure the generation quality in terms of conveying general semantic information and it measures the semantic slots in a sentence similarly as word error rate (WER):

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

where p is the number of missing slots in a generation, q is the number of redundant slots in a generation and N is the total number of slots in a correct generation (target sentence). An example is presented below:

*Target: the address is **slot-inform-addr**. postal code **slot-inform-post**. and it is located in the **slot-inform-area**.*
*Generation: it is located in the **slot-recommend-area**. the address is **slot-inform-addr**, and it is in the **slot-inform-area**.*

Clearly, the generated sentence incorrectly generates a *recommend-area* slot instead of *inform-post*, hence the slot error rate for this example will be $\frac{1+1}{3} = 66.7\%$.

Perplexity

Perplexity measures the probability of an example is generated from a distribution and is typically used for language models. Since the word tokens are sampled from distributions in our task, we calculate the perplexity over each sentence and then average over all generated examples. A lower perplexity usually shows the model is more confident in its generations.

BLEU score

BLEU (bilingual evaluation understudy) is a popular metric in natural language processing to evaluate model performance in transforming one form of text to another. It is based on n-gram precision and is a number between 0 and 1. The closer the generated sentence is to the target, the higher the BLEU score will be. Additionally, to evaluate model performance at corpus-level, the BLEU scores can be averaged over all sentences. Normally, n is set to be 4 which means the metric considers the similarity from unigrams to four-grams between two sentences and hence we report it as BLEU-4 in the future.

Comparisons of the evaluating metrics

Each of the above metrics can examine the language models at corpus-level and generally shows how well the models perform. BLEU score has been proved to be closely related to human evaluation but a model with higher BLEU score will not necessarily be a better model. For example, in our task, the generated sentence can be almost identical to the target with the only difference being a wrong substitution from *slot-price* to *slot-area*. In this case, there will be a high BLEU score but the important semantic information is not correctly presented.

Perplexity can indicate whether a model is confident with its generations, while it is not directly correlated with the quality of the generations in our task. Firstly, Perplexity is unable to measure the accuracy of important semantic slots in the generations as ERR does. Secondly, it fails to evaluate whether the generations are similar to the target. More importantly, Perplexity is more suitable for a Language Model (LM) in recognition task than an LM model in a generation task since we would want diverse generations as discussed before.

Though ERR can accurately measure whether the dialogue act information is conveyed correctly in the model generations, it is poor in measuring the similarity between the generations and the target sentence. A sentence with lower ERR can still be inconsistent with the target sentence and problems such as low intelligibility and incorrect grammar can still exist.

In view of this, the three metrics will be used jointly to evaluate the generations' quality. However, we focus more on slot error rate (ERR) since we care about whether the system has conveyed the semantic information encoded in dialogue act vectors which is the core task of the NLG module.

Chapter 3

Related work

3.1 Designing a memory module

A memory module can be viewed as a kind of information buffer consisting of a number of memory slots which can be trained to store and pass useful information to the system through designed interaction mechanisms. Generally, using memory modules requires read and write operations. Reading is the process of retrieving relevant and desired information from a memory buffer. There will be input to the memory module in the process of reading and this input is known as 'query'. Writing refers to the update of a memory module and usually involves the addition or replacement operation for some memory slots in the buffer. Furthermore, a system normally needs to learn the weights of using the information obtained from a memory module. The influential Neural Turning Machines (NTM) [6] is one of the pioneers in using external memory to augment a neural network system and it has inspired the recent models with external memory modules through a basic model structure illustrated in figure 3.1.

Specifically, a few things can be considered when designing and using a memory module. *Ways to address the memory*: content-based addressing accesses memory module through a defined similarity measure between memory slots and a query hence a key matching distribution can be computed to show the correlations between memory slots and the query; location-based addressing accesses memory module based on the location of each memory slot. A combination of the two addressing methods is also applicable and is popularly used in [6] and [9].

When a memory module is interacting with the environment, the system generally needs to decide which memory slots are more informative thus it can store and use information from memory more efficiently (this process is commonly known as attention mechanism). Given a computed similarity distribution (also known as key matching distribution), there are

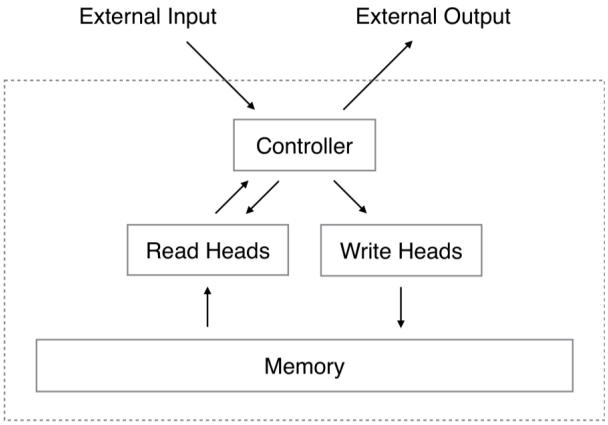


Fig. 3.1 NTM model structure (source: [6]): using an external memory to augment the controller network

two *types of attention mechanism*, namely soft and hard attention mechanisms to retrieve the information from the memory buffer. A soft attention uses a weighted sum of memory slots through the similarity distribution while a hard attention instead usually uses a single memory slot either through sampling from the similarity distribution or taking the memory slot with maximum similarity to the query. Additionally, *different ways of memory initialization* also needs to be considered when designing a memory buffer with variable memory content. The first and most popular approach is to initialize the memory module randomly such that more flexibility is given to the model. While in models such as [1], the initial memory contents are sampled from the training data with the propose of better model initialization. Most importantly, *where to use the memory in a system and the use of information retrieved from a memory module* should be carefully studied and thought about before using an external memory. This is a broad topic and is the focus of various models with memory. We briefly discuss a few memory models below since they motivate the models studied in this project.

3.2 Memory modules as database

Memory modules can be used as a database providing information to be selected by the system. In [21], a large memory module is employed to store information from the input context sentences in the question-answering task. A question will interact with this memory module through an inner product similarity measure, while the memory module will also interact with the context sentences to generate the appropriate response. In [32], an external memory is used as a database to store the vector form representations of emotional and

generic vocabulary. A vocabulary type selector which decides the probability of using an emotional or generic word is trained such that a probability distribution over the whole vocabulary is obtained and is used to generate emotional sentences.

3.3 Memory modules as augmentation of neural network systems

Additionally, memory modules are also popular in augmenting generation models by capturing and using certain information from the system. In [4], a memory module as a trainable matrix (model parameters) can pass local invariant information and detailed feature information [4] from one layer to another in deep generative models (DGMs). [9] presents a large life-long memory module to improve models' performances in terms of dealing with rare events in machine learning tasks (a rare word in machine translation task for example). The long-term memory buffer can store and retrieve information about training examples from a fairly early stage of the training. Therefore, the memory module in [9] is successful in the task of one-shot or k-shot learning in models such as the Convolutional Network and Google Neural Machine Translation. One drawback of their approach is the high computational cost even though the fast nearest-neighbour algorithm is used to improve the model efficiency.

To mitigate this problem, some memory modules inherit the main idea of [9] but without computing the 'memory loss' in [9] and without computing and storing a large amount of 'nearest neighbours' [9] memory slots such that the models are more scalable in terms of memory size. These models are well received in sequence-to-sequence generation tasks, especially Neural Machine Translation (NMT). Before we discuss these memory models, we first briefly introduce the NMT task. An NMT system maps the input text to another language (output text) and it usually consists of three components: an encoder, a decoder and attention mechanisms (attention vectors). An example figure of the attention mechanism in NMT models is presented in figure 3.2.

In [30], one memory module studied is 'random access memory' [30] where a memory buffer is used to store k most recent hidden states in an LSTM decoder. Given a new hidden state h_t, a weighted sum m_t of the stored hidden states is calculated through the similarity distribution and is combined with the original hidden state through trained parameter matrices $W_{h,h}$ and $W_{h,m}$: $\hat{h}_t = W_{h,h}h_t + W_{h,m}m_t$. The motivation here is to help an LSTM decoder retrieve useful hidden state level information (m_t) based on recent histories at each time step. This m_t is further used to improve the original h_t. The main disadvantage of this approach is that this memory module still performs in a sequential way like the LSTM mechanism

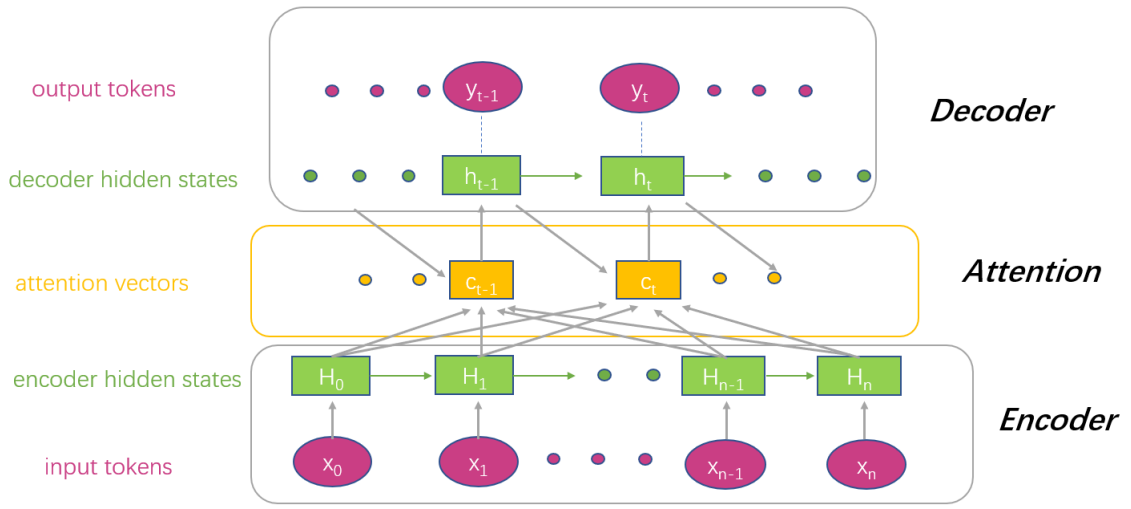


Fig. 3.2 Figure of attention mechanism in NMT models

and the model is unable to use information beyond the current sentence. This limitation is verified by results in [30] that only marginal improvement over the translated sentences can be obtained using this approach.

In [5] where a memory module is used to improve the NMT task, the memory consists of key-value pairs where memory keys are vectors consisting of hidden state information and memory values are word tokens. The correspondence between a hidden state and its output token is augmented by previous generation histories such that more appropriate word tokens will be generated.

In addition, a memory module can also be used to take advantage of information from a deeper level. By deeper, we mean the context vectors which guide the behaviours of hidden states in the decoder, although a hidden state itself serves as context for generating a token. In NMT models, the attention vectors can be seen as contexts for the hidden states in the LSTM decoder. In [24], a memory module is used to store both attention vectors and hidden state vectors. Using memory module in this scenario aims to augment the relations between attention vectors from the source side and hidden states for the target side [24]. This memory mechanism has been shown to have better performance in terms of BLEU and Perplexity scores than memory models mentioned earlier since it enables better use of translation history while exploits both source-side and target-side information during generation.

3.4 Similarities between NMT and SC-LSTM system

The semantically conditioned LSTM (SC-LSTM) model for NLG in Spoken Dialogue System is intrinsically similar to the task of NMT. They are both sequence-to-sequence

generation tasks transforming one form of text sentences to another. Besides, they both have the module to encode input information into context vectors. In NMT, it is the attention mechanism that transforms source-side hidden states in the encoder into attention vectors while the bottom Dialogue Act (DA) cell in SC-LSTM transforms the source-side input DA into feature vectors $d_0, d_1, ..., d_t$ which serve as sentence contexts for the LSTM hidden states above them. Though the DA feature vectors in SC-LSTM have sequential dependencies between each other which differs from the attention mechanism in NMT, we can still take advantage of these deeper context vectors in memory models.

With these in mind, we first investigate the use of memory modules in the conditional VAE (CVAE) to see whether we can further improve the initialization of the SC-LSTM system. Then we design two memory modules inside the LSTM cell of the SC-LSTM system which have similar ideas with memory modules in NMT. The main goal of these models is to augment the current SC-LSTM system such that reduced average slot error rate (ERR) can be achieved.

Chapter 4

Model architectures

4.1 Memory modules inside the conditional VAE

The conditional VAE has the advantage of encoding sentence information into the latent variable z. In this section, we study the various usage of memory modules inside this CVAE. In the CVAE, a semantic representation (SR) of a dialogue act (DA) has been trained to have a stronger relation with its target generation (which is passed through an LSTM module to get the sentence information vector X). During inference, the VAE mechanism ensures the sampled z exploit this SR-X pair information. Since it remains to be implicit what exact information from X or SR should be stored in the memory, we let the memory module decide by itself. Therefore, we introduce each memory module as a trainable matrix of model parameters in this section.

Reading from the memory

The mechanism of the memory module used in this section is shown in Fig 4.1.

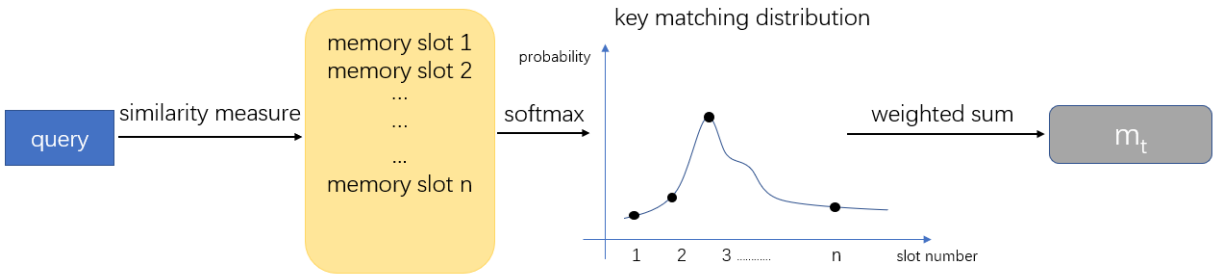


Fig. 4.1 Reading from a memory module

The memory module is designed as a number of memory slots where each slot is a row vector of model parameters whose dimension is fixed to be the same as the query vector (this is to ensure that we can take inner product between the two vectors).

Given a query q, a similarity measure is used to measure the correlation between q and each memory slot m_a in the model. There are a few inner-product based similarity measures such as plain inner product $< q \cdot m_a >$ and normalized inner product $\frac{<q \cdot m_a>}{|q| \cdot |m_a|}$. We didn't observe significant differences between using these measures in a few scenarios hence we just use $\frac{<q \cdot m_a>}{|q| \cdot |m_a|}$ as our similarity measure in this section. The next step is to apply the softmax function on the computed similarity matching quantities to get the final key matching distribution over the memory slots:

$$p_{match}(a) = \frac{exp(\frac{<q \cdot m_a>}{|q| \cdot |m_a|})}{\sum_{j=1}^{k} exp(\frac{<q \cdot m_j>}{|q| \cdot |m_j|})}$$

A few ways can be used to retrieve m_t from the memory buffer which will be discussed along with different memory usage.

Using the information retrieved from the memory buffer

We first apply the memory module to augment the latent variable z as shown in fig 4.2 and use the memory module to augment the SR as in fig 4.3. We refer them as 'memory module for z' and 'memory module for SR' respectively in this section.

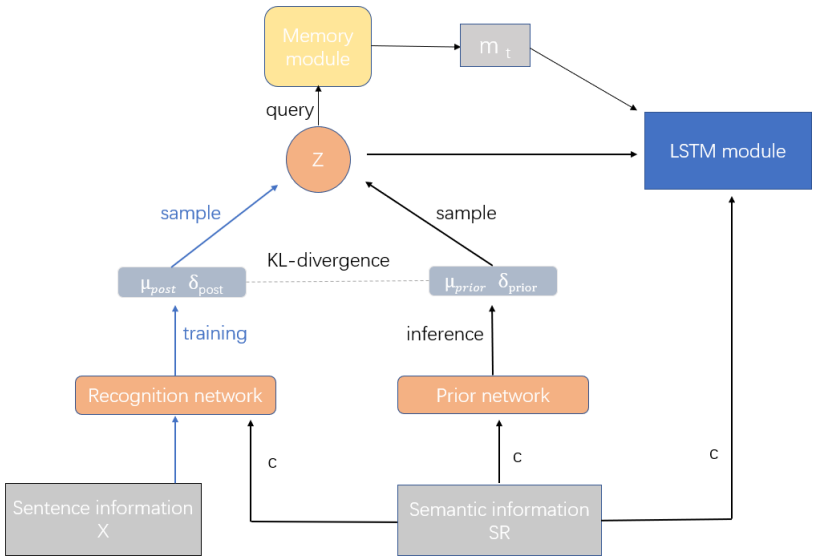


Fig. 4.2 Memory module for z

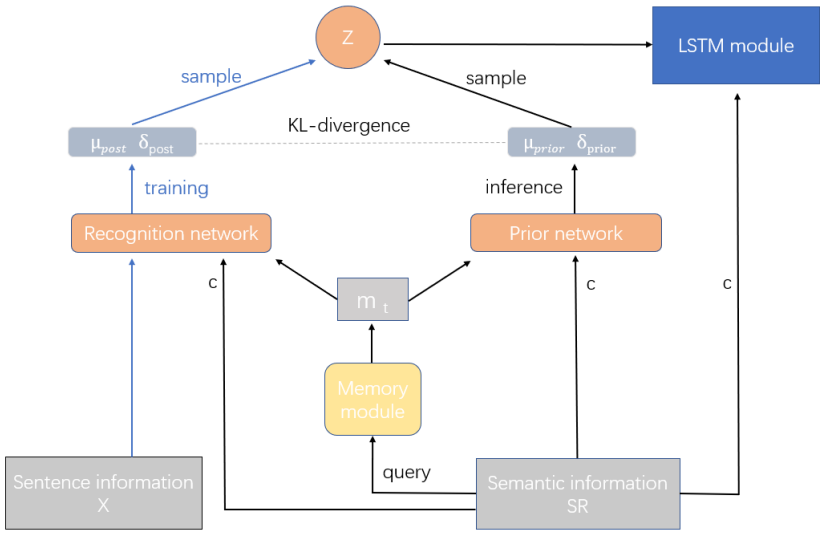


Fig. 4.3 Memory module for SR

As shown in the figures, the memory modules are used to pass information between the layers (modules). Given the queries, m_t will be retrieved from the memory and then be passed to next module of the system. The memory modules are automatically trained such that it can store some useful information from previous training examples and given a new input during inference, it will find the most similar information from the memory module through the similarity measure. This m_t can be viewed as a kind of supplementary information from the input layer to the following layer.

One limitation of the above model architectures is that memories are used to encode only one form of information without binding the relations between different modules in the CVAE. The CVAE module has the advantage of encoding Semantic Representation and its corresponded target generations information into a latent variable. Inspired by this, we study a memory model where the memory module can interact with both SR and X to see whether better improvements can be achieved. The general architecture using memory is shown in fig 4.4 .

As in fig 4.4 and fig 4.5, we use X and SR jointly to interact with the memory module during training and we use SR only to interact with memory module during inference (since X is not available during inference). The information retrieved from the memory module during training (m_{train}) and inference (m_{test}) will then be passed to recognition network and prior network respectively. To achieve this, we concatenate X with SR to form a new vector C. To ensure that queries in training and inference have the same size as the dimension of a memory slot, we passed the C (concatenation vector) and SR through fully connected layers to get the queries during training and inference respectively.

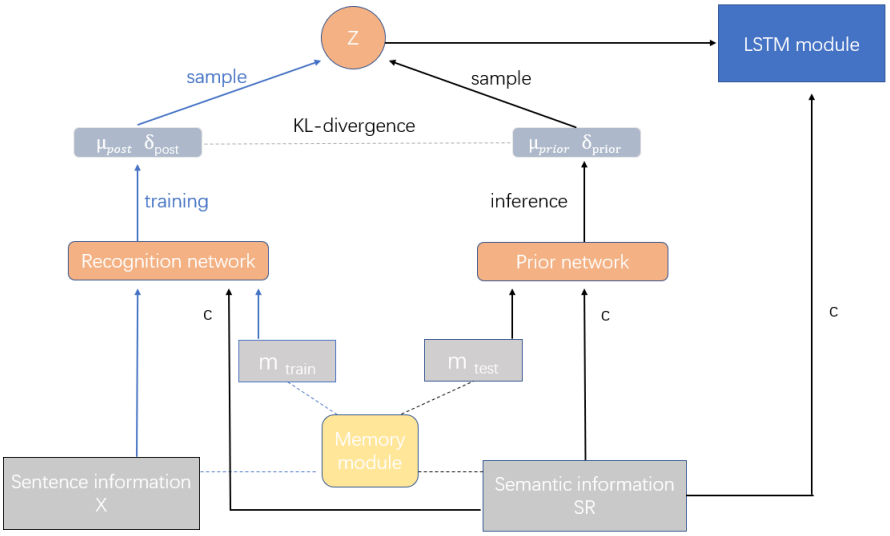


Fig. 4.4 Memory module which interacts with X and SR jointly (dash lines around memory module show the general information flow and interactions between the modules which are further explained in fig 4.5)

During training:

$$q = U \cdot C$$

During inference:

$$q = V \cdot SR$$

where q is the query to the memory module, SR and C are the semantic representation and the concatenation vector respectively. U, V are weight matrices such that q in training and inference have same sizes. Suppose dimensions of SR, X and a memory slot are $\mathbb{R}^{a\times 1}$, $\mathbb{R}^{b\times 1}$ and $\mathbb{R}^{h\times 1}$ respectively, then dimensions of C, U, V will be $\mathbb{R}^{(a+b)\times 1}$, $\mathbb{R}^{h\times(a+b)}$ and $\mathbb{R}^{h\times a}$.

Given the queries, we can compute key matching distributions $q(a)$ and $p(a)$ through the similarity measure discussed above during training and inference respectively. There are generally two methods (as shown in 4.5) to retrieve memory information given $q(a)$ and $p(a)$ which are discussed below:

1. A soft way: using weighted sum to compute m_{train} and m_{test} during training and inference respectively.

2. A hard way: sampling the memory address variable a from $q(a)$ and $p(a)$ during training and inference respectively. The final memory information will then be m_a.

The first soft way of accessing memory has the advantage of considering all memory slots instead of a single memory slot. More importantly, this soft way of using memory does

not involve a sampling process. This is a major advantage since the gradients regarding the model parameters (such as memory contents and U, V) can be back-propagated through the memory model without any difficulties. On the contrary, the hard way of using memory introduces a non-differentiable node which means the gradients can not be passed through the non-differentiable memory address variable a. Hence, the memory module and the fully connected layers (U, V) will not be properly optimized using this sampling method. It is also verified by our experimental results that using the sampling method without any trick to help gradients pass through the discrete variable a will lead to significantly worse results.

Considering this, here we use the weighted sum method to compute the memory information. Additionally, we employ the similarity measure KL-divergence between $q(a)$ and $p(a)$ which is used to encourage the $p(a)$ during inference to be closer to the $q(a)$ during training. To achieve this, the KL-divergence $KL(q(a)||p(a))$ is added to the total training loss of the CVAE-SCLSTM (SCVAE) system. This way, the parameter matrix of the fully connected layer V will be trained along with the system such that $KL(q(a)||p(a))$ will gradually decrease to a small number. Hence, the information retrieved from the memory module during inference will be similar to the information stored in the memory during training even if the sentence information X is not available during inference. We hope that such m_{test} which is similar to information from previous examples will be helpful for the generation task.

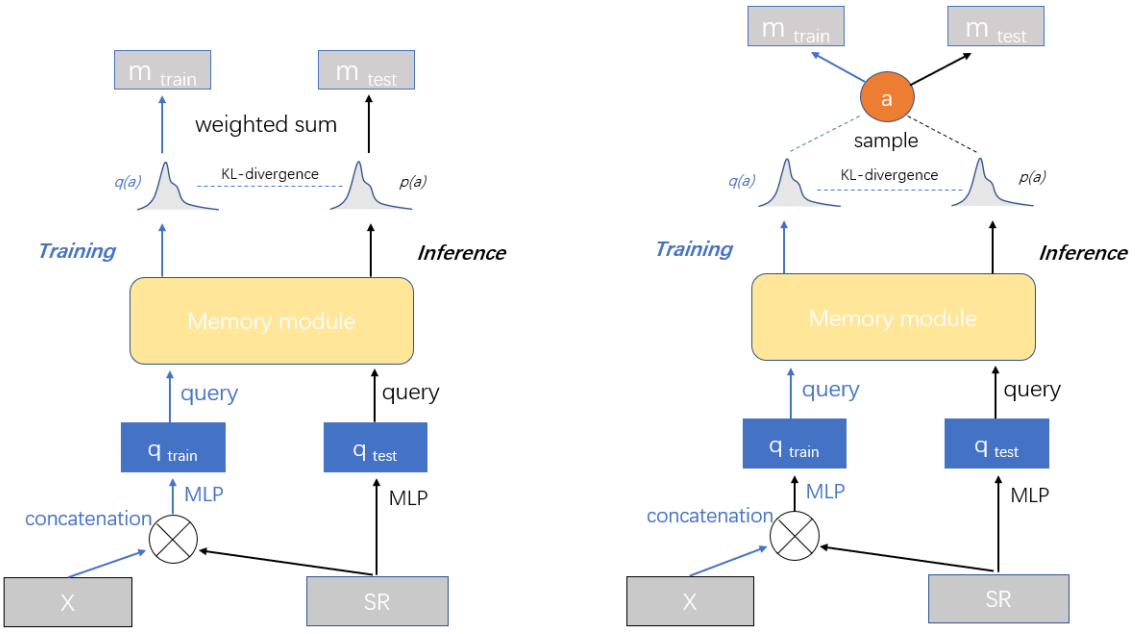


Fig. 4.5 Reading from the memory module using a soft way (left) and a hard way (right)

Training

For the *memory model for z* and the *memory model for SR*, the total training loss will remain the same. For the *memory module which interacts with both X and SR*, the KL-divergence between the key matching distributions in training and inference is added to the total training loss and it can guide the optimization of U, V as discussed above. The memory modules here are model parameters and a natural way of memory initialization is to initialize the memories randomly. Since we always use weighted sum method to retrieve memory information in all memory models, the memory buffer and the parameter matrices of the fully connected layers for the memory modules can be trained automatically along with the CVAE and SC-LSTM modules through back propagation. This is a major advantage of these kinds of models as we do not need to design the memory update rules. Additionally, though we expect that the computation cost of adding these memory modules will scale linearly with the memory size, we found that generally adding these memory modules will not lead to significantly longer training time.

4.2 Word level external memory module inside the SC-LSTM

Instead of using memory buffers inside the CVAE, we can consider using external memory modules inside the SC-LSTM model such that the system can interact with the memory at each time step. We first study a word level memory buffer to augment the correspondence between hidden states and the generated word tokens which is inspired by the memory model proposed in an NMT task in [5].

The internal architecture of SC-LSTM decoder

The internal SC-LSTM model architecture can be shown in fig 4.6. As discussed in the Background section, the hidden state vector is passed through a deep neural network and then a softmax function to get a distribution over the vocabulary word tokens. The final output token is sampled from this distribution. The generated token at time t-1 (y_{t-1}) is the input token at the next time step.

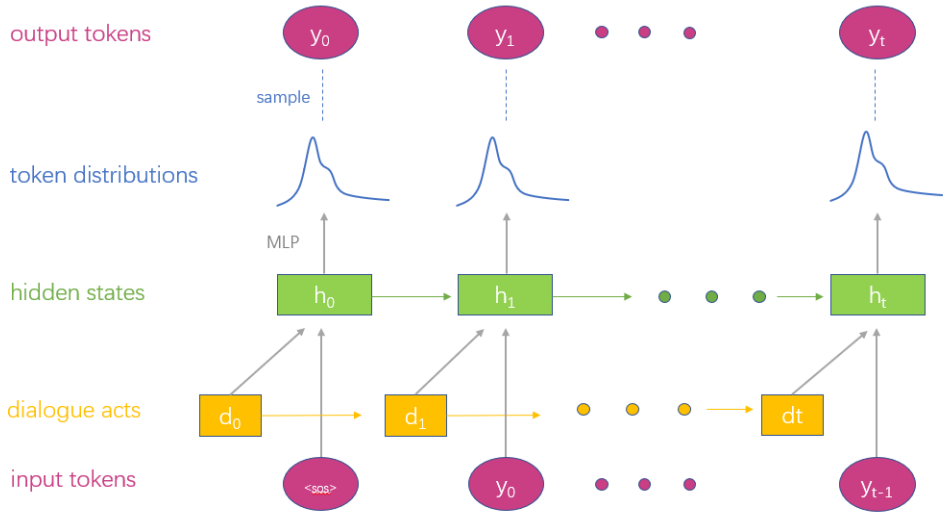


Fig. 4.6 The internal architecture of SC-LSTM when generating output tokens at each time step

Reading from memory module

The memory buffer is composed of memory slots of key-value pairs with the keys being hidden state information and the values being the word tokens from the dataset vocabulary. The memory module architecture and the usage of memory information m_t are presented in fig 4.7 .

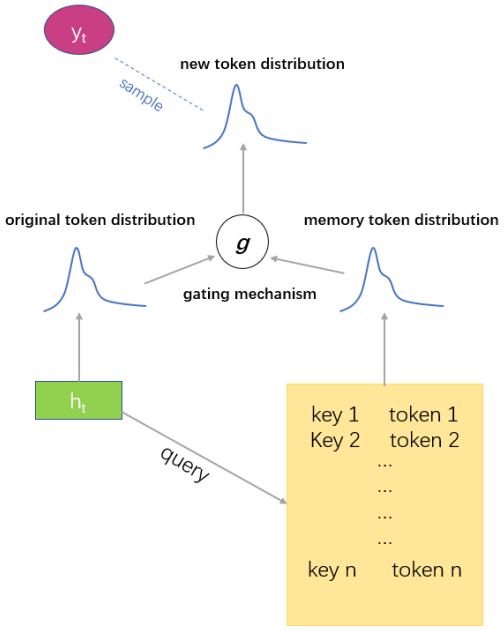


Fig. 4.7 Word-level memory module architecture and reading from the memory buffer at time t

As in figure 4.7, the query to the memory module is the hidden state vector h_t. Key matching quantities between a query and memory keys can be computed through similarity measures. Here we use the simple inner product as our similarity measure. As before, the key matching quantities are passed through the softmax to get the key matching distribution over all memory slots. Since each memory value corresponds to a word token, the key matching distribution can be seen as a distribution over the vocabulary words and this distribution over the word tokens is the information retrieved from the memory buffer at time t (m_t). We then combine the distribution from memory (m_t) with the distribution from SC-LSTM language model at each time step to form a new distribution over words from the vocabulary. The final output word token can then be sampled from the new distribution as usual.

Gating mechanism

Here we use the gating mechanism which is similar to Gated Recurrent Units (GRUs) and LSTM models:

$$\tilde{P}_{LM}(w_t) = (1-\lambda_t)\cdot P_{LM}(w_t) + \lambda_t \cdot P_{memory}(w_t) \tag{4.1}$$

$$\lambda_t = Sigmoid(U \cdot h_t + V \cdot d_t + W \cdot m_t) \tag{4.2}$$

where $P_{LM}(w_t)$, $P_{memory}(w_t)$, $\tilde{P}_{LM}(w_t)$ are original probability from language model, probability from the memory buffer and new probability of generating word token w_t at time t respectively. And λ_t controls how much information from the memory should be used in forming the new probability distribution over word tokens $\tilde{P}_{LM}(w_t)$. h_t, d_t, m_t refer to the hidden state vector, dialogue act information and key matching distribution from memory respectively. U, V, W are matrices to control the information flow from the h_t, d_t, m_t respectively and are implemented as model parameters to be trained along with the whole system. Since hidden state, dialogue act vector and memory module can have different sizes, the dimensions of U, V, W should be $\mathbb{R}^{1\times h}$, $\mathbb{R}^{1\times d}$, $\mathbb{R}^{1\times m}$ respectively to ensure the combination of three terms is a scalar. The scalar is passed through the sigmoid function such that the final weight λ_t is a number between 0 and 1. In [5], they use a fixed weight λ to combine the memory distribution and the original LM distribution at each time step. However, this weight has to be tuned carefully and lacks the flexibility and variability between different time steps. Additionally, our experimental results show that using a dynamic gated weight outperforms a fixed weight in our task so we use this gating mechanism to compute the weight in our model. The gating parameters need to be trained and hence we apply this memory module during both training and inference which is different from [5] where their memory is only used during inference.

Update of the memory module

Since the word distribution from memory will be combined with the word distribution from the language model, we set the size of the memory buffer to be equal to the size of the vocabulary of the dataset. Given this, each generated word can find a corresponding memory slot to be updated hence we do not need to replace any exiting slot when updating the memory. Hence we use a simple update rule for the memory module: Given a generated word token y_t with ID n and corresponding hidden state h_t, the updated key of memory slot n ($\tilde{k}_n$) will become: $\tilde{k}_n = \frac{k_n+h_t}{2}$. Since some word tokens will appear multiple times in a single sentence, the first appearance of the word in the sentence will have more influence on the later generations than the words from previous sentences since it is more recent if memory buffer is updated at each time step. We want to focus on the effects of considering cross-sentence information since the dependencies within a sentence can be modelled by the LSTM model, hence we update the memory buffer after a whole sentence is generated instead of updating the memory buffer at each time step.

Though the hidden state information from the memory buffer is not directly used to concatenate with the language model, they serve as keys to access the word tokens. Given a

new example, the probability of sampling a token is changed as its memory key is updated by new hidden state information thus the model can exploit the previous generation histories.

Training

The training objective will stay the same as the baseline SC-LSTM training objective. The memory keys and memory values are updated according to the rules discussed above which means back-propagation for memory contents is not needed. The memory values are word tokens which are fixed from the beginning as discussed above and the memory keys are initialized as random vectors. The reading and updating mechanisms for the memory buffer remain the same during both training and inference. During training, U, V, W in equation 4.2 are weight matrices for the gating mechanism hence they are treated as model parameters to be trained along with the whole system through back-propagation. During inference, the memory keys will be initialized randomly again but the trained U, V, W from training will be fixed. At each time step, the weight vector λ_t which controls the memory information usage is computed depending on h_t, d_t and m_t: $\lambda_t = Sigmoid(U \cdot h_t + V \cdot d_t + W \cdot m_t)$.

Since λ_t should be a scalar here, U, V, W are just row vectors hence the total number of new model parameters and the computational cost for updating the memory buffer are relatively small. This is verified in practice that adding this memory buffer will not significantly increase the computational complexity of the system which indicates that this kind of memory buffer can be easily scaled for tasks with even bigger vocabulary sizes.

4.3 Context level external memory module inside the SC-LSTM

Apart from augmenting the process of generating word tokens from hidden states as in section 4.2, we can also improve the correspondence between the deeper context vector (DA feature vector d_t) and the hidden state vector h_t using the recent generation history. Since an NMT model and SC-LSTM model both have deep context vectors as discussed in section 3.4, we study a memory module which inherits the main idea from [24].

Memory module architecture

Similar to the word-level memory model proposed in section 4.2, here we use a memory buffer inside the SC-LSTM such that memories are accessed at each time step. The memory slots are also in the form of key-value pairs with the memory keys being the mixed information from the DA feature vectors and the memory values being the mixed information from hidden states. Additionally, for the convenience of memory update, we also store the generated word tokens and the ages of each memory slot. The architecture of the memory buffer and the reading process is shown in fig 4.8. We describe the memory read and write operations below.

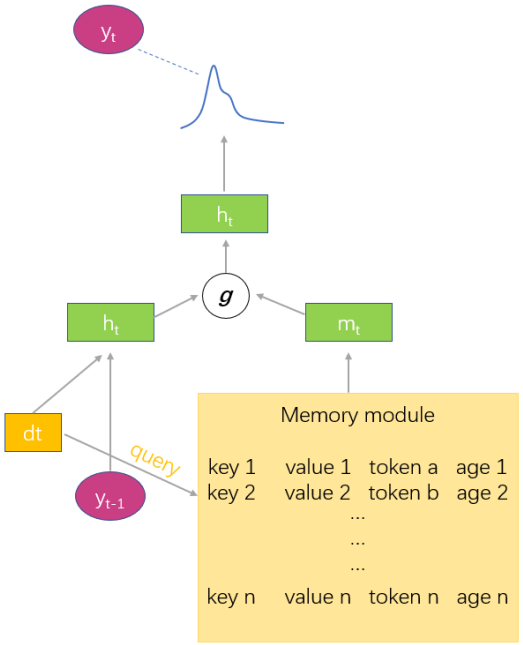


Fig. 4.8 Context-level memory module architecture and reading from the memory buffer at time t

Reading from the memory module

The reading process is straightforward and it is similar to previous models. The DA feature vector d_t will be the query to the memory. Under some similarity measure and softmax function, we can obtain the key matching distribution over all memory slots. The information retrieved from the memory module is then taken to be a weighted sum of the memory values (hidden state information) stored in the memory:

$$m_t = \sum_{j=1}^{n} P_{match}(j) \cdot v_j \tag{4.3}$$

$$= \sum_{j=1}^{n} \frac{exp(d_t \cdot k_j)}{\sum_{i=1}^{n} exp(d_t \cdot k_i)} \cdot v_j \tag{4.4}$$

where d_t, k_j, v_j denotes the query (DA feature vector), memory keys and memory values respectively.
This memory information m_t is then combined with the hidden state h_t from the SC-LSTM system using a similar gating mechanism as in section 4.2:

$$\tilde{h}_{LM}(t) = (1 - \lambda_t) \cdot h_{LM}(t) + \lambda_t \cdot m_t \tag{4.5}$$

$$\lambda_t = Sigmoid(U \cdot h_t + V \cdot d_t + W \cdot m_t) \tag{4.6}$$

Unlike in section 4.2, λ_t here is a vector of weights instead of a single scalar as we use information from memory differently for different dimensions of a hidden state. Interestingly, we found that eliminating the last term in 4.6 can lead to slightly better model performance through control experiments. Hence, we use equation 4.3 instead to obtain the dynamic weight vectors since this introduces fewer parameters with better results:

$$\lambda_t = Sigmoid(U \cdot h_t + V \cdot d_t) \tag{4.7}$$

Given the new hidden state vector $\tilde{h}_{LM}(t)$, a distribution over words and the final sampled word token will be generated similarly as in the baseline LM model.

Updating the memory module

As the memory buffer here is not used to obtain a distribution over all words from vocabulary, the size of memory (number of slots) does not have to be fixed to be equal to the vocabulary size. Consequently, when a new DA example shows up, we need to replace some memory slots. When implementing the memory module, each memory slot consists of 4 components:

memory key, memory value, corresponding word token and age. It can be denoted as: $slot[n] = [k_n, h_n, y_n, a_n]$. Hence for a new generated token y_t with corresponding DA feature vector d_t and hidden state h_t, the memory update mechanism can be summarized as below:

if y_t in memory:

find n such that $y_t = y_n$

update memory slot n:

$$\tilde{slot}_n = (\tfrac{k_n + k_t}{2}, \tfrac{h_n + h_t}{2}, y_n, a_n = 0)$$

update the age of other memory slots:

$$\tilde{slot}_j = (k_j, h_j, y_j, a_j + 1) \text{ for } j \neq n$$

if y_t not in memory:

randomly choose n such that $a_n \geq a_j$ for $j \neq n$

replace memory slot n:

$$\tilde{slot}_n = (k_t, h_t, y_t, a_n = 0)$$

update the age of other memory slots:

$$\tilde{slot}_j = (k_j, h_j, y_j, a_j + 1) \text{ for } j \neq n$$

We choose to replace the oldest memory slot since we believe the sentence context information from most recent generations will be more important. Additionally, we still update the memory buffer after a whole sentence is generated to investigate the influence of cross-sentence context information.

Training

The training objective of the context-level memory model will stay the same as the baseline SC-LSTM training objective. Again, training and inference follow the same memory reading and updating processes as described above. Suppose the size of the memory module is n, then we randomly sample n word tokens from the vocabulary during initialization. For each memory slot which contains a random token, we initialize the memory key and memory value as random vectors. The ages of these n memory slots are initialized as 0. The initial word tokens stored in the memory slots will not matter due to the quick replacement (overwrite) mechanism during memory updating. During training, weight matrices U, V are the only parameters to be trained along with the system since the memory contents will be updated through the defined rule. During inference, the memory will be initialized again while U, V from training will be fixed. Then for each time step at testing, a weight vector λ_t will be computed depending on h_t and d_t: $\lambda_t = Sigmoid(U \cdot h_t + V \cdot d_t)$.

In [24] and [5], pre-training technique is used to deal with the problem of training difficulty in their complicated NMT task. To reduce the computational complexity and for better model performance, we also use the pre-train technique when adding the memory module to the SC-LSTM system. Pre-training means we first train the SC-LSTM system and then we pass these trained weights to our new model. Then the new parameters introduced to the system such as weight matrices U, V will be trained jointly with the parameters from the SC-LSTM model. With pre-training technique, our new memory model will have a better initialization which means the gradients of model parameters are less likely to be trapped in local maxima/minima thus it generally leads to better model performance.

Different from models in section 3.1 and 3.2, we found adding this memory buffer will significantly increase the computational cost and generally the training time scales linearly as memory size increases. There are a few possible reasons: First, our implementation might not be efficient enough and needs improvement in terms of training efficiency. Second, the number of new parameters introduced by the gating mechanism to the system is significantly higher than that of the model in section 3.2 since λ_t is a vector instead of a scalar. Third, the updating mechanism for the memory module is more complicated than models from previous sections which may possibly increase the training time.

Chapter 5

Experiments

5.1 Datasets and setup

5.1.1 Data

In this project, experiments are performed on two datasets which consist of system-side text dialogue turn sentences with corresponding annotated semantic representations.

The first dataset is SF dataset [26] and is also used in models from [26] and [23] which serve as baseline models in our project. The dataset has 4 domains: Restaurant, Laptop, TV and Hotel. The statistics of each domain is demonstrated in table fig 5.1.

	Domain			
	Hotel	Laptop	Restaurant	TV
total number of dialogue turns	5192	5373	7035	13202
number of different dialogue acts	9	9	14	14
number of different semantic slots	12	12	15	12

Table 5.1 Statistics for each domain in SF dataset

For SF dataset, we first perform experiments on all 4 domains and then we also perform experiments on the single Restaurant domain since it is the most challenging domain with the highest slot error rates (ERR) among all 4 domains. An example DA with its target generation from SF Restaurant domain is shown as below:

Example DA: *inform(kidsallowed=yes; count=0; name=opera plaza)*
Target generation: *there are 0 restaurant -s where kid -s are allowed in the opera plaza area*

Apart from this relatively simple dataset, we also train our models on the second dataset Multi-domain Wizard-of-Oz (MultiWOZ) which was recently collected by Cambridge Dialogue System Group [3]. Similar to the first dataset, it has multiple types of dialogue acts and semantic slots across several topics and it has been annotated by Mechanical Turks. The statistics of MultiWOZ dataset along with the SF dataset is shown in table 5.2:

Statistic	Dataset	
	SF	MultiWOZ
total number of dialogue turns	30802	127669
total number of different dialogue acts	14	13
total number of different semantic slots	42	24
vocabulary size	750	2250

Table 5.2 Statistics for SF and MultiWOZ datasets

Compared to the first dataset, it has much more dialogue turns for every single domain which enables the effective training of more complicated models. Additionally, the DA in MultiWOZ tend to be more complicated (sentences are longer; more dialogue act types and semantic slots in each DA) than those from SF dataset and the vocabulary size in MultiWOZ is much bigger. Therefore, MultiWOZ is a better dataset to test model performances for the word-level memory model and the context-level memory model. An example of DA with its target generation in MultiWOZ is shown below:

Example DA: *general-reqmore-none|Book-Name-1|Book-People-1|Book-Ref-1|Book-Stay-1*
Target generation: *i was able to book you for **slot-book-stay** night at **slot-book-name** for **slot-book-people** people . your reference number is **slot-book-ref** . is there anything else i can do for you ?*

Both datasets are split with ratio 3:1:1 to form the training set, validation set and testing set respectively. We first perform experiments for our models on the first dataset since it takes less training time while can give us general impressions of how models behave and it is easier to debug on this simple dataset.

5.1.2 Setup

All models were implemented through Pytorch [18] and we developed our memory models based on the source code of SC-LSTM and CVAE from [23]. The size of hidden state vectors in LSTM modules, the size of semantic representation vector in SC-LSTM decoder and the size of sentence information vector in the CVAE module are set to be 128, 153 and

128 respectively. For memory models in section 3.1, we keep the batch-size to be 64. For memory models in section 3.2 and 3.3, the batch-size is set to be 1 since memory models are implemented to deal with only one sentence at each batch. We follow other parameter settings as in [23] since they have already been tuned carefully.

Since the generation process in decoders involves sampling word tokens from the output distribution in SC-LSTM and sampling latent variable z in CVAE, experiment results can vary slightly over different seeds due to the randomness. Hence we run at least 5 seeds and average the results for each experiment setting. Besides, we follow [23] and [26] to use the over-generation mechanism during inference which refers to generating multiple sentences for each testing DA and averaging model performances over these generations. Firstly, over-generation mechanism tests models' capability to generate diverse while accurate sentences. Secondly, models such as CNN and backward RNN can be further used to rerank these generations as in [25]. Table 5.3 demonstrates an example of top 5 generations for an example input DA from SF dataset. We set the number of over-generation to be 10 for all decoders in all experiments.

Example DA: *inform(kidsallowed=yes; count=0; name=opera plaza)*
Target generation: *there are 0 restaurant -s where kid -s are allowed in the opera plaza area*

Gen 1: *there are 0 restaurant -s in opera plaza that do not allow kid -s*
Gen 2: *there are 0 restaurant -s in the area in opera plaza that do not care the food*
Gen 3: *there are no restaurant -s near opera plaza that are not allow child -s*
Gen 4: *there are 0 restaurant -s that are near opera plaza that do not allow kid -s*
Gen 5: *i am sorry but there are no restaurant -s that allow child -s near opera plaza*

Table 5.3 An example of top 5 generations for an example DA from SF dataset

5.2 Memory modules in CVAE

5.2.1 Some preliminary results

As previously discussed, the latent variable z from the CVAE module is passed to the SC-LSTM decoder as the initial hidden state which can be seen as a better initialization to the decoder. We are also curious about whether this z is helpful for a vanilla LSTM decoder (without conditioning on semantic representation SR) and the difference between z and plain SR information as input to the LSTM-based models. Hence, some preliminary results are summarized in table 5.4. By plain SR we mean passing the SR vector directly to decoders as the initial hidden state.

Initialization of LSTM	Decoder Type	
	vanilla LSTM	SC-LSTM
0 initialization	Fail to learn(124%)	5.49%
random initialization	Fail to learn(121%)	5.41%
plain SR	12.21%	5.96%
z from CVAE	31.43%	4.89%

Table 5.4 Averaged slot error rates for SF Restaurant domain in different LSTM initialization

As can be seen, the vanilla LSTM model will fail to learn without any guidance from the SR information. Interestingly, the plain SR information as model initialization outperforms the latent variable z from the CVAE module. While for the SC-LSTM system, we can observe that ERR will be around 5.5% without any additional information as the initial hidden state (model 1, 2). The SC-LSTM achieves the best performance using the sampled z from the CVAE module while it performs worst with plain SR information. Since z encodes the correlation between an SR and its target sentence, SC-LSTM seems to favour this supplementary information instead of the redundant plain SR information. While for a vanilla LSTM which is not conditioning on any SR information, the plain SR information will be much more helpful than information from z. The SC-LSTM models can always outperform the vanilla LSTM decoders because they can exploit the SR information at each time step.

5.2.2 Main results

Here we mention the information retrieved from the memory as m_t, sentence information encoded by an LSTM as X and the sampled latent variable as z as before. The CVAE-SC-LSTM (SCVAE) model will be our baseline model in this section.

We first present the results of memory modules augmenting z and SR in table 5.5

Model Architectures	Metrics	
	Slot error rate	BLEU score
baseline SCVAE	4.89%	0.539
memory module for z	4.78%	0.539
memory module for SR	4.93%	0.5387

Table 5.5 Best Performances of memory models on SF Restaurant Domain

For memory models in table 5.5, we report their best model performances as we vary the memory module size between [10, 100, 256, 512, 1024]. The reading process uses the weighted sum to compute m_t and the similarity measure used is the normalized inner product. We noticed that there are no significant changes even though we scale the memory size up to 1024. Furthermore, results from the table indicate that generally there are no statistically significant improvements when using the memory modules to augment z or SR.

Additionally, instead of passing m_t as supplementary information for the queries as discussed in section 3.1, we experimented on replacing the queries by m_t such that only m_t is passed to next layer/module in the system and we observed even worse model performance. This is reasonable since m_t can be viewed as general information taking account of previous training examples hence replacing z or SR with m_t will lead to loss of specific information for each DA. Consequently, we always use m_t as supplementary information in the memory models.

We next report the performance of the memory module which interacts with both X and SR during training in table 5.6 and table 5.7.

Model Architectures	Metrics	
	Slot error rate	BLEU score
baseline SCVAE	4.89%	0.539
memory module which interacts with X and SR	5.24%	0.536

Table 5.6 Best performance of the memory model which interacts with z and SR jointly on SF Restaurant Domain

Contrary to our expectations, the memory module which interacts with X and SR has achieved worse model performance. One possible explanation is that the SC-LSTM system favours z with clear dialogue act type and domain information provided by the CVAE module while the current mechanism using the $KL(q(a)||p(a))$ will encourage the sampled z to be closer to each other regardless of the DA information and domain information encoded in each test DA (m_{test} will always be similar to the information stored in the memory since

Model Architectures	Metrics	
	Slot error rate	BLEU score
baseline SCVAE	2.27%	0.434
memory module which interacts with X and SR	2.36%	0.4337

Table 5.7 Best performance of the memory model which interacts with z and SR jointly on SF Multi-Domain

$p(a)$ will be close to $q(a)$ after a few training epochs due to the KL-divergence). This does not provide a clean separation of dialogue act types or domains information for each input DA which is contradictory to the main idea of the CVAE module. Hence worse model performance is obtained due to the worse sampled z.

5.2.3 Models' performances on unseen test data

While we are curious about whether the memory modules have improved system's performances on unseen data, we compared our models' performances on unique and non-overlapping test datasets inspired by [12]. The SF dataset is divided into training, validation and testing sets with ratio 3:1:1 while some of the dialogue acts (DA) in testing set also exist in the training and validation sets. Besides, same DA can appear multiple times in the test set which may lead to biased results. Considering this, we picked the unique test set (each DA only appear once) and the non-overlapping test set (DA which are unseen in training and validation sets) from the original SF dataset as in [12]. The statistics of unique and non-overlap test sets are summarized in table 5.8.

Type of test set	Domain			
	Hotel	Laptop	Restaurant	TV
whole test set	1075	2649	1039	1407
unique test set	78	1378	130	680
non-overlap test set	5	33	7	21

Table 5.8 Numbers of DA in unique and non-overlap test sets across the domains in SF dataset

We noticed that the exact numbers for Restaurant and Hotel domains in our unique/non-overlapping set differ slightly from the numbers in [12]. The models are trained as before but are also tested on the unique test set and the non-overlap test set. The ERR of the baseline model and the memory models on new test sets across 4-domains are shown in table 5.9.

Clearly, all models are less robust on non-overlapping set which indicates decayed generation accuracies on unseen test data. Again, we observe some slight improvements

Type of test set	Model type		
	baseline SCVAE	Memory module for z and SR	Memory module for z
whole test set	2.27%	2.36%	2.19%
unique test set	2.63%	2.82%	2.52%
non-overlap test set	3.49%	3.56%	3.28%

Table 5.9 Models' performances on unique set and non-overlap set across 4 domains of SF dataset

using the best memory module on the test sets. Disappointingly, though the ERR in the best memory model (memory size 256) is slightly lower, the improvement is not statistically significant given the limited data in the non-overlapping set (only 66 DA in total in the non-overlapping set) which suggests that the memory module will not make the system more robust in terms of unseen data on the SF dataset.

5.2.4 Discussions

As we experiment on various memory modules inside the CVAE, improvements of the ERR are generally marginal. Moreover, we aware that there are a few limitations of using these memory modules inside the CVAE. First, memory contents in the memory buffer are designed to be model parameters. Though this mechanism lets the memory to flexibly decide what forms of information should be learned, the memories inside are not interpretable intrinsically (since they are vectors of numbers) which makes it hard to visualize the memories and to know what helpful information from memory is used. In addition, the improvements are relatively marginal even with the best memory architecture given the parameters it has introduced. For example, the memory module for z with size 256 actually increases the number of parameters by $256 \times 128 = 32768$ (when the hidden state vector dimension is 128) while only increases the ERR by 0.11%. It is possible that the generation task on the SF dataset is not challenging enough (SRs are too simple; the baseline model is already good enough) and the unique training data are limited such that a model with far more parameters will not have significantly better results. But more importantly, the memory modules are used in the CVAE to obtain a better z which is the initialization of the SC-LSTM decoder. This indicates that memory modules are only used to provide a better initialization for the LSTM instead of interacting with the decoder at each time step, hence the memory modules will not effectively mitigate the problem with long-range dependencies in LSTM based models. With this in mind, we move on to experiments where memories are used inside the LSTM cell of the SC-LSTM system.

5.3 Word-level and context-level memory modules inside the SC-LSTM

In this section, we compare two kinds of memory modules which can interact with the LSTM cell at each time step. We mention models described in section 3.2 and 3.3 as word-level memory model and context-level memory model respectively from now on. Since we want to investigate the effects of memory modules inside the LSTM cell, here the SC-LSTM system is taken to be our baseline model instead of the SCVAE in section 5.2 . We first experiment on the simple SF dataset.

5.3.1 Experiments on SF dataset

We used the pre-train technique to add our memory modules to the baseline SC-LSTM model as discussed in section 3.3. Results on SF Restaurant domain and multi-domain are summarized in table 5.10 and 5.11 respectively.

Model Architectures	Metrics		
	Slot error rate	BLEU score	Perplexity
baseline SC-LSTM	5.41%	0.537	2.86
word-level memory	5.26%	0.5352	2.92
context-level memory	5.07%	0.5384	2.86

Table 5.10 Performances of two kinds of memory modules inside the SC-LSTM on SF Restaurant domain

Model Architectures	Metrics		
	Slot error rate	BLEU score	Perplexity
baseline SC-LSTM	2.45%	0.4341	3.48
word-level memory	2.43%	0.4335	3.42
context-level memory	2.25%	0.4344	3.49

Table 5.11 Performances of two kinds of memory modules inside the SC-LSTM on SF multi-domain

From the tables, the performance of the word-level memory model is similar to that of the baseline model while the context-level memory model slightly reduces the ERRs to 5.07% on Restaurant and 2.25% on multi-domain. There are two reasons which may possibly explain the limited improvement using the context-level memory model: Firstly, input DA in the SF test set are independent of each other since the order of dialogue turns in

each dialogue is not preserved in the SF dataset. Consequently, it will be less likely that a testing sentence contains useful contexts for its neighbour sentences. Secondly, SF dataset is relatively simple (sentences are shorter; input DA are less complicated as discussed before) compared to MultiWOZ and we can see that the baseline SC-LSTM has already achieved fairly good performance on SF multi-domain test set which means there is not much room for improvement in terms of ERR. In view of this, we should test the models' performances on MultiWOZ dataset to better compare the models.

Before moving on to MultiWOZ dataset, we first take advantage of the over-generation mechanism (as discussed in section 5.1.2) in the decoders to test whether memory modules can exploit cross-sentence information. Since multiple sentences will be generated during decoding (over-generation mechanism), we can update the memory module after each generation to see whether the former generations can influence the later generations for each single DA. The averaged results over the generations are shown in table 5.12.

Model Architectures	Metrics		
	Slot error rate	BLEU score	Perplexity
baseline SC-LSTM	5.41%	0.537	2.86
word-level memory	5.20%	0.5359	2.86
context-level memory	4.35%	0.542	2.85

Table 5.12 Performances of memory models which are updated at every generation (during over-generation) for each DA in SF Restaurant domain

As can be seen, the context-level memory module can significantly reduce the averaged slot error rates of generations while the word-level memory module still fails to give any further improvements. One possible explanation for the improvement is that the contexts in all generations for a DA are similar to each other which means later generations can easily use previous context information stored in the memory buffer, hence reduced averaged ERR over all generations for one DA can be achieved. This shows that the memory module does have the potential to exploit the cross-sentence information and can be used to improve the overall quality of all generations of a DA. However, we are more curious about the use of memory module between different test DA instead of different generations of a single DA. Hence, we will focus on the use of memory modules between different input DA in later experiments on MultiWOZ dataset.

5.3.2 Experiments on MultiWOZ dataset

As discussed before, MultiWOZ has much more dialogue turns than the naive SF dataset for each domain. Besides, the dialogue acts (DA) tend to be more challenging in MultiWOZ as

discussed before. More importantly, the MultiWOZ dataset preserves the order of DA in the dialogue so the information from previous DA can be useful for the current DA. Hence, both word-level and context-level memory modules can make use of the corpus-level information in the experiments. An example of a dialogue consisting of turns of DA from the machine side is shown in table 5.13:

*1: 'there are **slot-inform-choice** options in cambridge , do you have any tastes in particular ?'*
*2: 'there are **slot-inform-choice** results for **slot-inform-food** restaurants . is there a certain area or price range you 're looking at ?"*
*3: '**slot-inform-name** fits your request perfectly . they are located at **slot-inform-addr** . would you like to make a booking ?'*
*4: 'the phone number is **slot-inform-phone** . the address is **slot-inform-addr** , and the post code is **slot-inform-post** . is there anything i can help you with ?'*

Table 5.13 An example dialogue which consists of dialogue turns from the machine side. Numbers in the front show the order of each dialogue turn in the dialogue.

We can observe that the ***slot-inform-choice*** is mentioned in both turn 1 and turn 2 which shows the overlapped contextual information in the two turns. Hence, we expect better model performance using the two memory modules. We avoided the influence of the over-generation mechanism by fixing the memory buffer during the over-generation for a DA and updating the memory buffer when a new DA shows up. Results are summarized in table 5.14.

Model Architectures	Metrics		
	Slot error rate	BLEU score	Perplexity
baseline SC-LSTM	8.52%	0.6083	4.20
word-level memory	8.56%	0.6082	4.76
context-level memory	7.46%	0.6132	4.249

Table 5.14 Performances of memory models on MultiWOZ Restaurant Domain

Though the word-level memory model still does not show any improvement over the baseline model, the context-level memory model has remarkably outperformed the baseline model in terms of slot error rates. The Perplexity in all three models are generally the same which may due to the fact that the perplexity in baseline system is already small enough. In addition, we can also observe that BLEU score in context-level memory model is slightly higher than that of the baseline model which can be explained by more accurate generations of semantic slots and hence more similar generations to the target in the memory model.

From the experiments on SF and MultiWOZ datasets, we can see there are no significant improvements using the word-level memory model. One possible reason is that using this

word-level memory model will encourage a word from previous sentences to show up in the current generation. This might be helpful in some tasks such as NMT (since words show up in the previous translation history are more likely to show up again [5]) while the primary problem in our task is missing/ replacement of semantic slots and most words will not show up multiple times in a dialogue (apart from the generic word tokens such as 'can', 'will', 'thank' etc) . Furthermore, this kind of memory module is popular in dealing with 'rare events' (in language model tasks refers to rare words) since this memory module is in a life-long manner. But this will not be helpful for our task since the semantic slots are not 'rare' and vocabulary sizes of both datasets are relatively small (775 and 2250) which makes problems with rare words less severe. Another possible reason is that hidden states and DA feature vectors serve as surface realization and sentence planning respectively in SC-LSTM system hence it may be more helpful to augment the model from a 'higher' level (deeper contexts instead of hidden state vectors).

To better explain the improvement of the context-level memory model, we choose an example input dialogue act whose generations have been improved:
general-reqmore-none|Book-Name-1|Book-People-1|Book-Ref-1|Book-Stay-1
And its top 5 generations by the baseline model are shown in table 5.15:

*Gen1 (0,1,4): i have booked a room at **slot-book-name** for **slot-book-people** person . your reference number is : **slot-book-ref** . can i assist you with anything else ?*
*Gen2 (0,0,4): i have booked **slot-book-name** for **slot-book-stay** nights for **slot-book-people** people. your reference number is **slot-book-ref**. can i help you with anything else ?*
*Gen3 (1,1,4): i have booked **slot-book-name** for you for **slot-book-people** people for **slot-book-people** nights . your reference number is **slot-book-ref**. is there any other information i can assist you with today ?*
*Gen4 (1,1,4): i was able to book **slot-book-name** , for **slot-book-stay** nights starting on **slot-book-day** . your reference number is : **slot-book-ref**. is there anything else i can help you with ?*
*Gen5 (1,0,4): your reservation for **slot-book-people** at the **slot-book-name** for **slot-book-stay** days **slot-book-people** people was successful . your reference number is : **slot-book-ref** . is there anything else i can help you with ?*

Table 5.15 Top 5 generations for the example DA in the **baseline SC-LSTM** model: numbers (a,b,c) in the front correspond to the number of redundant slots, number of incorrect slots and total number slots in target sentence respectively.

As can be seen, one major problem in the generations of the baseline model is the missing and replacement of ***slot-book-stay***. This indicates the limitations of SC-LSTM in capturing long-range dependencies. While in our context-level memory module, this can be significantly improved by taking advantage of similar contextual information from the

recent dialogue turns. The dialogue consisting of dialogue turns from the machine side which contains the example DA is shown in table 5.16.

1: "i 'm sorry . i don't have that information . can i help you with anything else ?"
2: "i 'm sorry , the booking was unsuccessful . would you like to try for another date or a shorter stay ?"
3: 'can you find me another hotel that accommodates 6 people'
4: "i 'm not able to reserve that hotel for ***slot-nobook-stay*** *days for* ***slot-nobook-people*** *people . would you like to try another hotel ?"*
5: 'yes , we can do ***slot-book-stay*** *night at* ***slot-book-name*** *.'*
6: 'i was able to book you for ***slot-book-stay*** *night at* ***slot-book-name*** *for* ***slot-book-people*** *people . your reference number is* ***slot-book-ref****. is there anything else i can do for you ?'*

Table 5.16 Dialogue in the testing data which contains the example DA (number 6)

From table 5.16, we can observe that ***slot-book-stay*** exists in both sentences 5 and 6. In addition, ***yes, we can do*** in sentence 5 generally has similar contextual intention as ***I was able to book*** in sentence 6. Hence this provides an opportunity for the memory buffer to store information from sentence 5 and to pass the information to sentence 6. We show our top 5 generations in table 5.17. As can be seen, the problem with generating ***slot-book-stay*** has been fixed which leads to a smaller ERR for the test DA. In addition, we demonstrate an example grayscale figure of key matching distribution over the memories in the memory buffer for the third generation of the example DA (fig 5.1).

Gen1 (0,0,4): i was able to book ***slot-book-name*** *for* ***slot-book-people*** *people for* ***slot-book-stay*** *nights . your reference is* ***slot-book-ref*** *. can i help you with anything else ?*
Gen2 (0,0,4): i 've booked you a room at the ***slot-book-name*** *for* ***slot-book-people*** *people for* ***slot-book-stay*** *nights . your reference number is* ***slot-book-ref*** *. is there anything else i may help you with ?*
Gen3 (0,0,4): your booking for ***slot-book-people*** *at* ***slot-book-name*** *was successful for* ***slot-book-stay*** *nights . your reference number is :* ***slot-book-ref*** *. can i be of further help ?*
Gen4 (0,0,4): i 've made a booking at the ***slot-book-name*** *. for* ***slot-book-stay*** *nights for* ***slot-book-people*** *people . your reservation number is* ***slot-book-ref*** *. is there anything else i can help you with today ?*
Gen5 (0,0,4): i have booked a room at the ***slot-book-name*** *for* ***slot-book-stay*** *nights ,* ***slot-book-people*** *people . your reference number is* ***slot-book-ref*** *. is there anything else i can help you with ?*

Table 5.17 Top 5 generations for the example DA in the **context-level memory model**

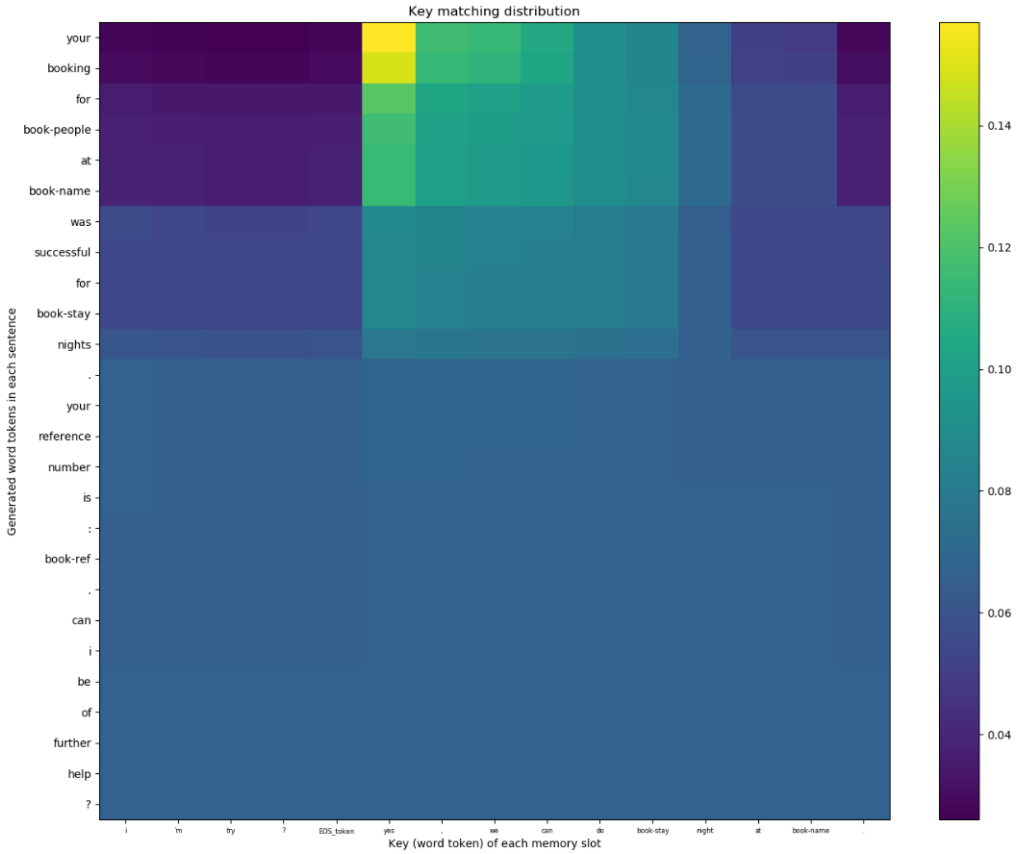


Fig. 5.1 The third generation for the example DA: the y-axis shows the delexicalized word tokens in the generated sentence (top to bottom) while the x-axis shows the memories with corresponding keys (word tokens) in the memory buffer.

Figure 5.1 shows the magnitude of key matching probabilities through the brightness of grids. As can be seen, the memories with 'keys' (word tokens) being ***'yes', 'we', 'can', 'do', 'slot-book-stay'*** generally have higher matching probabilities with the words in the generation. This is in good agreement with the similarities between sentence 5 and sentence 6 (example DA) discussed above which suggests that the new generations retrieve the corresponding memory values (hidden state information) from the memory buffer and correctly generate ***slot-book-stay*** through the gating mechanism described in section 3.3. We also present the weight vector λ_t of the gating mechanism (from below equation) for each word in the generations which shows the usage of the memory information at each time step (as in figure 5.2).

$$\tilde{h}_{LM}(t) = (1 - \lambda_t) \cdot h_{LM}(t) + \lambda_t \cdot m_t \lambda_t$$

Nevertheless, a few things should be noted in the key matching distribution figure (fig 5.1). The correlations between word tokens and the keys (word tokens) are descending as

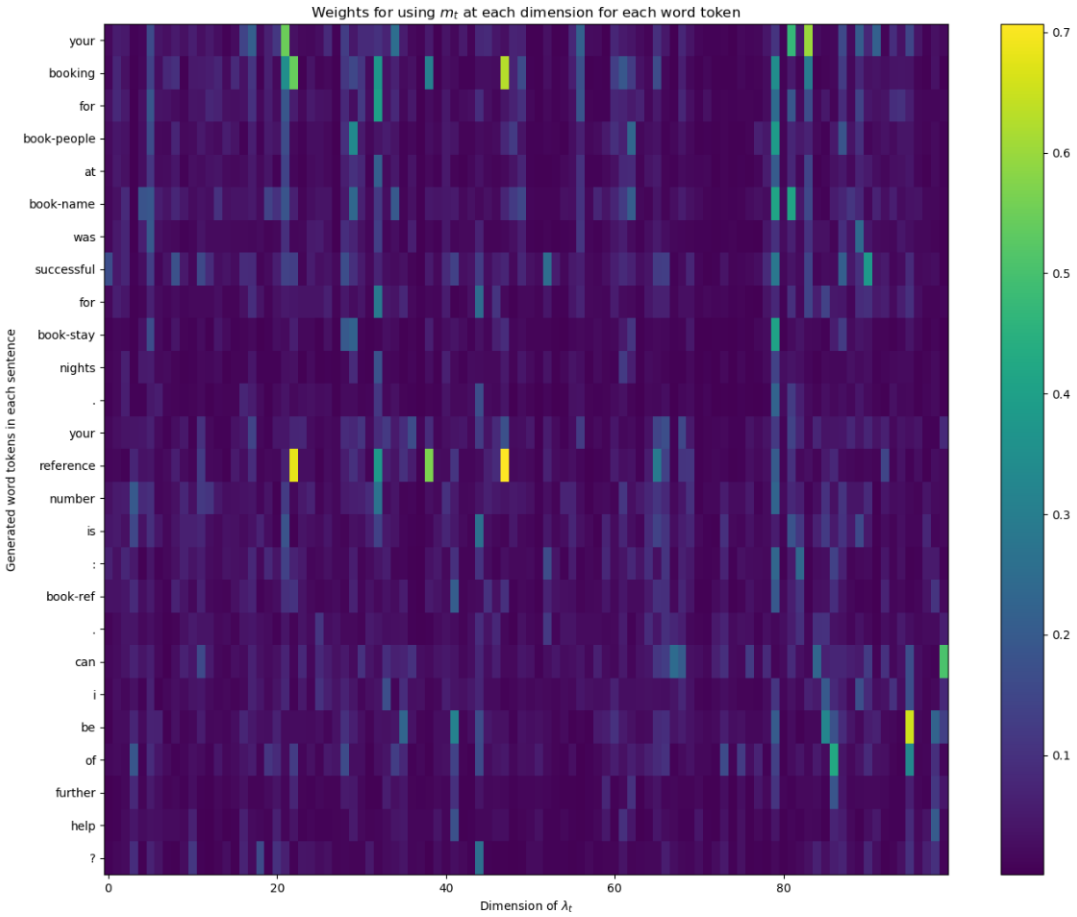


Fig. 5.2 λ_t for each word in the generated sentence

the words are generated (from top to bottom, the grids are becoming darker which indicates less matching probabilities). A possible reason is that we are using DA vectors (d_0, d_1, ..., d_t) as the queries to the memory module whose dependencies on each other are sequential. In addition, we can notice that after ***slot-book-stay*** is generated, the correlations between following word tokens and the memories immediately decay to a small number. Again, this can be explained by the mechanisms of DA vectors. The control gates for DA in SC-LSTM are designed in a way such that values in DA vectors which correspond to the certain semantic slot will decay to 0 after that slot is generated. Since the usage of memory information depends on the queries (DA vectors), we observe the decayed correlations between current and previous contexts after the semantic slot is generated.

5.3.3 Context-level memory models of different sizes

The memory module reported in the last section has memory size 15. We scale the memory size up to 75 to see whether better model performances can be achieved. Results are reported in table 5.18.

Model Architectures	Metrics		
	Slot error rate	BLEU score	Perplexity
baseline SC-LSTM	8.52%	0.6083	4.20
memory with size=15	7.46%	0.6132	4.249
memory with size=25	**7.34%**	**0.6146**	4.170
memory with size=50	7.72%	0.6125	4.23
memory with size=75	7.70%	0.6129	4.23

Table 5.18 Performances of memory modules of different sizes on MultiWOZ Restaurant Domain

From table 5.18, the context-level memory module can achieve the best model performance when the memory size is 25. This finding slightly contradicts our expectation that memory modules with larger sizes will generally have better performances since they will store more contextual information. There are a few possible explanations: Firstly, we noticed that the number of memory slots in a memory buffer which have strong correlations with the generated words is usually below 10 and that the most matched memories in memory modules with different sizes are nearly the same in most cases. This indicates that a small memory size (such as 15 and 25) will usually be enough for a memory module to store the useful context information. Secondly, a larger memory buffer will tend to store the context information from 'older' histories which are less likely to be relevant to the current generation (in our MultiWOZ dataset, the most relevant context information will usually be in the previous two turns before the current dialogue turn in a dialogue). This can be a 'distraction' for the system to select the most helpful information from the memory buffer hence further improvements using larger memory buffers are not observed.

5.3.4 Performance of context-level memory module on limited training data

In this section, we compare the performance of our context-level memory model with the performance of the baseline model on limited training data. We randomly choose dialogues from the training set such that the total dialogue turns in the dialogues are approximately 1/4, 1/2 or 3/4 of the original training data. Note we must ensure that the two models are

trained on the same sampled data during each comparison to make the results comparable. As discussed in the last section, the memory buffer will achieve relatively good results when the memory size reaches 25 hence we present the model performances for memory module size 15 and 25 respectively. Again, we run five seeds for each experiment setting and the averaged results are presented in figure 5.3.

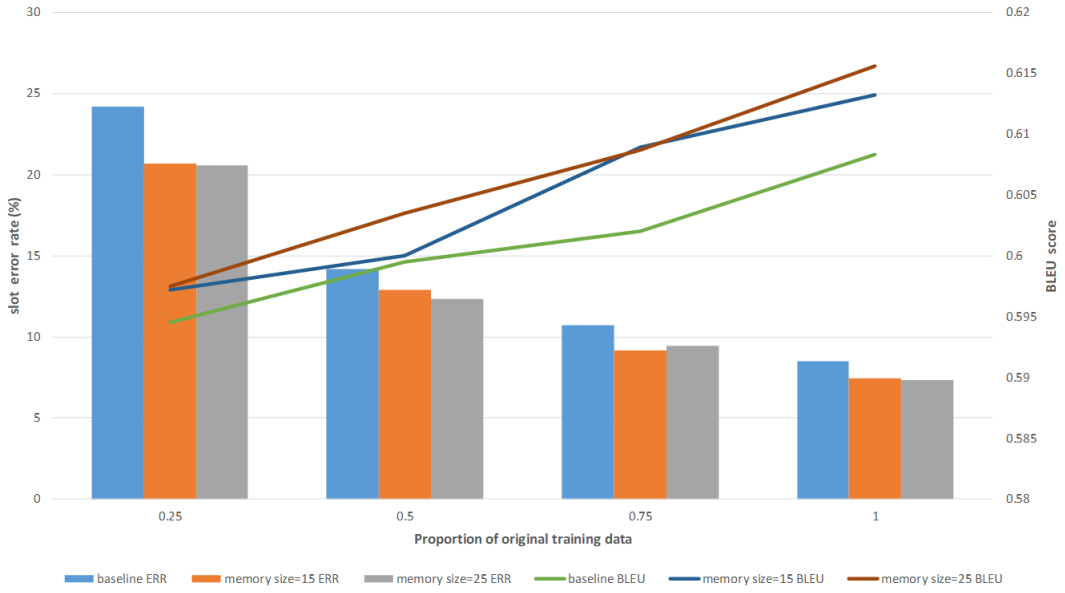


Fig. 5.3 Models' performance on limited training data (MultiWOZ)

As can been seen, when the training data is less, all models will have worse performance in terms of ERR and BLEU score. Besides, clearly the memory models (size 15 and 25) can have even more improvement over the baseline SC-LSTM model in terms of ERR when the training data is less. In addition, generally the memory model of size 25 can slightly outperform the memory model of size 15 though the difference is not significant which again suggests a memory buffer with size 25 is more suitable for the MultiWOZ dataset.

5.3.5 Discussions

Though the word-level memory module introduces fewer parameters and the computational cost of adding it is negligible due to its simple memory updating mechanism, it lacks the ability to improve generations' quality through previous training examples which indicates that the deeper contextual information plays a more important role in conveying correct semantic information.

Context-level memory model can achieve better generations since it can exploit the contexts of previous generations from a higher level. Additionally, a system with such

memory module can perform much stronger than the baseline model when the training data is limited which shows the potential of this memory model on challenging generation tasks. However, a number of limitations of the context-level memory module need to be considered. Firstly, our memory module excels in storing and retrieving information of recent generations but it fails to store information from the distant past under current updating rule. Though we have shown that memory buffers which can store more contextual information may not necessarily lead to better model performances on our MultiWOZ dataset, there may be generation tasks where a life-long memory module is more suitable. Secondly, using DA feature vectors as queries to the memory buffer means it is harder for later word tokens in a generation to retrieve useful information from the buffer since dependencies between DA feature vectors are sequential. Hence, it is possible that the memory module will fail to improve the slot errors at the end of generations when the sentences are fairly long. Finally, the large number of new parameters and the relatively complicated updating rule limit the scalability of this context-level memory buffer. Hence, a more flexible and efficient memory updating mechanism needs to be explored.

Chapter 6

Summary and Future work

6.1 Summary

In this project, we compared three types of memory modules for the state-of-the-art SC-LSTM system. Using memory modules inside the CVAE will generally fail to achieve significant improvements in the generations since memory information is only used during the initialization of LSTM-based models. This indicates that it is more attempting to let a memory module interact with the LSTM cell at each time step. Word-level and context-level memory modules can exploit the cross-sentence information in a dialogue through designed reading and updating rules of the memory buffer, and the retrieved information from a memory buffer can be used to augment the LSTM-based system at each time step through a flexible gating mechanism. We found that the word-level memory module is unable to improve our NLG task in the SDS which suggests that the DA feature vectors are more important in conveying semantic information. Additionally, we have shown that the context-level memory module can exploit contextual information from recent generation histories to augment the correlations between DA feature vectors and hidden state vectors in an LSTM cell such that more accurate and natural generations can be obtained. This shows the potential of using the memory to exploit corpus-level information in a dialogue and to augment the current LSTM-based models when the input dialogue acts are complicated.

6.2 Future work

The extensions of this project can be summarized as below:

Memory modules in the deep SC-LSTM system

Apart from the basic SC-LSTM system, multiple LSTM cells can be stacked through skip connections and drop-out technique to get a deep SC-LSTM decoder [26] such that better performances can be achieved. Hence, it is promising to study the use of context-level memory modules inside this deep version of SC-LSTM. Since there are multiple layers of DA feature vectors and hidden states in this deep SC-LSTM system, it will be interesting to study which correspondence between them should be augmented by the memory module.

Comparisons with models which directly use the previous context information

The current memory module is mainly used to augment the correspondence between context vectors and hidden state vectors given the generation histories, and the SC-LSTM system can exploit the memory information flexibly through the gating mechanism. Hence, it is worth investigating whether this memory module can outperform models where the previous context information is directly passed to the current generation. One way to achieve this might be encoding the previous sentence into a context vector for the current sentence through LSTM-based modules. Then we can discover which approach using the previous context information is better.

Using attention mechanism over the input DA

We noticed that using the DA feature vectors as memory queries still suffer from the problem of decaying sequential dependencies between each other which limits the use of memory information. One possible solution to be studied in the future might be using the attention mechanism over the input DA. The semantic representation of a DA can be passed through an LSTM module and the hidden states of this LSTM can be used to compute the attention vectors. For one thing, attention vectors may serve as better context vectors for the LSTM module on the top. For another, they can be used as better memory queries since the problem of sequential dependencies between memory queries can be mitigated.

References

[1] Bornschein, J., Mnih, A., Zoran, D., and Rezende, D. J. (2017). Variational memory addressing in generative models. In *Advances in Neural Information Processing Systems*, pages 3920–3929.

[2] Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2015). Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*.

[3] Budzianowski, P., Wen, T.-H., Tseng, B.-H., Casanueva, I., Stefan, U., Osman, R., and Gašić, M. (2018). Multiwoz - a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[4] Chevillon, G. and Mavroeidis, S. (2017). Learning can generate long memory. *Journal of Econometrics*, 198(1):1–9.

[5] Grave, E., Joulin, A., and Usunier, N. (2016). Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*.

[6] Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

[7] Henderson, M., Thomson, B., and Young, S. (2014). Robust dialog state tracking using delexicalised recurrent neural networks and unsupervised adaptation. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 360–365. IEEE.

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

[9] Kaiser, Ł., Nachum, O., Roy, A., and Bengio, S. (2017). Learning to remember rare events. *arXiv preprint arXiv:1703.03129*.

[10] Kalyani, A., Kumud, H., Singh, S. P., Kumar, A., and Darbari, H. (2014). Evaluation and ranking of machine translated output in hindi language using precision and recall oriented metrics. *arXiv preprint arXiv:1404.1847*.

[11] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[12] Lampouras, G. and Vlachos, A. (2016). Imitation learning for language generation from unaligned data. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1101–1112.

[13] Linzen, T., Dupoux, E., and Goldberg, Y. (2016). Assessing the ability of lstms to learn syntax-sensitive dependencies. *arXiv preprint arXiv:1611.01368*.

[14] Mairesse, F., Gasic, M., Jurcicek, F., Keizer, S., Thomson, B., Yu, K., and Young, S. (2009). Spoken language understanding from unaligned data using discriminative classification models. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 4749–4752. IEEE.

[15] Mesnil, G., Dauphin, Y., Yao, K., Bengio, Y., Deng, L., Hakkani-Tur, D., He, X., Heck, L., Tur, G., Yu, D., et al. (2015). Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539.

[16] Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

[17] Milward, D. and Beveridge, M. (2003). Ontology-based dialogue systems. In *Proc. 3rd Workshop on Knowledge and reasoning in practical dialogue systems (IJCAI03)*, pages 9–18.

[18] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

[19] Pinto, H. S. and Martins, J. P. (2004). Ontologies: How can they be built? *Knowledge and information systems*, 6(4):441–464.

[20] Sohn, K., Lee, H., and Yan, X. (2015). Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pages 3483–3491.

[21] Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

[22] Thomson, B. and Young, S. (2010). Bayesian update of dialogue state: A pomdp framework for spoken dialogue systems. *Computer Speech & Language*, 24(4):562–588.

[23] Tseng, B.-H., Kreyssig, F., Budzianowski, P., Casanueva, I., Wu, Y.-c., Ultes, S., and Gašic, M. (2018). Variational cross-domain natural language generation for spoken dialogue systems. In *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*, pages 338–343.

[24] Tu, Z., Liu, Y., Shi, S., and Zhang, T. (2017). Learning to remember translation history with a continuous cache. *arXiv preprint arXiv:1711.09367*.

[25] Wen, T.-H., Gasic, M., Kim, D., Mrksic, N., Su, P.-H., Vandyke, D., and Young, S. (2015a). Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. *arXiv preprint arXiv:1508.01755*.

[26] Wen, T.-H., Gasic, M., Mrksic, N., Su, P.-H., Vandyke, D., and Young, S. (2015b). Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*.

[27] Williams, J. D. and Young, S. (2007). Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2):393–422.

[28] Woodland, P. (2017). Lecture notes in lvcsr search and system design.

[29] Yao, K., Peng, B., Zhang, Y., Yu, D., Zweig, G., and Shi, Y. (2014). Spoken language understanding using long short-term memory neural networks. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*, pages 189–194. IEEE.

[30] Yogatama, D., Miao, Y., Melis, G., Ling, W., Kuncoro, A., Dyer, C., and Blunsom, P. (2018). Memory architectures in recurrent neural network language models.

[31] Young, S., Gašić, M., Keizer, S., Mairesse, F., Schatzmann, J., Thomson, B., and Yu, K. (2010). The hidden information state model: A practical framework for pomdp-based spoken dialogue management. *Computer Speech & Language*, 24(2):150–174.

[32] Zhou, H., Huang, M., Zhang, T., Zhu, X., and Liu, B. (2017). Emotional chatting machine: Emotional conversation generation with internal and external memory. *arXiv preprint arXiv:1704.01074*.