Neural network based task oriented dialogue system

Di Wang

Doctor of Philosophy

University of York

Computer Science

April 2021

Abstract

This thesis focuses on building a goal-oriented dialogue system with deep learning models. Current commercial dialogue systems such as Apple Siri or Google Alexa, although they are able to make simple daily conversations and finish tasks like open an app or play music, they are still obviously quite far away from being able to make complex conversations and doing difficult tasks such as replacing manual customer services. Dialogue systems in commercial use requires a massive amount of manually defined rules and expert knowledge.

The aim of this Thesis is explore the possibility of building a goal-oriented dialogue system by deep learning models, and the task is divided into three subtasks: 1. Sentence encoding, which tries to encode the text data into numerical vectors that can be input into neural networks and preserve as much information as possible, we will use tools such as word embedding, GRU, BERT, etc. 2. Dialogue state tracking tracks the topics and goals of dialogue, we designed a memory network-based dialogue state tracker. 3. Sentence generator, which generates system response to the user.

As a result of this thesis, we designed an end-to-end task-oriented dialogue system based on a memory network structure, which includes a memory-based dialogue state tracker and text generator. Our experiment shows that memory architecture can boost performance in both dialogue state tracking tasks and dialogue text generation tasks.

In conclusion, we find that deep learning approaches are useful for dialogue system tasks. Compared with the rule-based method, the deep learning method does not require much expert work and has better generality. But as shown in our experiment, the overall success rate of dialogue task completion is around 60%, so there is still a big gap between the state-of-the-art and dialogue system addthat is good enough for commercial use. We conclude that the current statistical NLP approach is not enough to achieve a significantly better dialogue system, unless more sophisticated approaches such as logic and reasoning are more thoroughly introduced to the model.

Contents

List of Figures

List of Tables

Acknowledgements

I am very grateful to my supervisor Dr. Simon O'Keefe, for his careful advice in the past year and a half. When my first supervisor left the university, and I was perplexed about the next step of my program, he was willing to be my new supervisor and gave me helpful guidance in a field he was unfamiliar with, which helped me out. I also would like to thank my first supervisor, Dr. Suresh Manandhar, who taught me a lot about research in the first two years of my program. Thanks to my colleagues at the University of York, they all gave me much help in research and life. Finally, thanks to my family, who gave me tremendous support, I would never go this far without them.

Declaration

I declare that this thesis is a presentation of original work, and I am the sole author. This work has not previously been presented for an award at this or any other university. All sources are acknowledged as references.

Publications

1. *Paterson, C., Calinescu, R., Wang, D., Manandhar, S. (2019, May). Using unstructured data to improve the continuous planning of critical processes involving humans. In Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (pp. 25-31).*

2. *Wang, D., O'Keefe, S. (2021). Memory State Tracker: A Memory Network based Dialogue State Tracker. In proceedings of 13th International Conference on Agents and Artificial Intelligence (1) (pp. 533-538), Vienna, Austria*

Chapter 1

Introduction

Those most challenging problems in artificial intelligence are sometimes unofficially called "AI-complete" or "AI-hard" problems, similar with NP-hard problem. Solving these problems are considered equivalent to achieve strong AI[24], which means to build an AI system that is as intelligent as a human. Dialogue system is undoubtedly one of them.

A dialogue system, also known as a conversational agent, is a computer system intended to converse with a human with a coherent structure. It is one of the core tasks in the Natural Language Processing (NLP) field and has been studied and commercially applied for many years. In the past, dialogue system was mainly built in rule-based approaches, such as ELIZA [62] or A.L.I.C.E. (Artificial Linguistic Internet Computer Entity [61]). In recent years, neural networks based end-to-end approaches have become the most popular tools for dialogue systems [54] [63].

There are mainly two types of dialogue system tasks: open dialogue and

task-oriented dialogue systems. The open domain dialogue system aims to give proper response to any user input, including day-to-day dialogue, greetings, etc. The training data for an open dialogue system is usually large unlabelled dialogue corpus, such as [33]. On the other hand, the task-oriented dialogue system aims to solve specific requests from users, such as restaurant/hotel booking. Its training data set is usually labeled and domain-specific. Question answering task is also a sort of dialogue system, and the underlying techniques are similar.

In this thesis, we mainly focus on a task-oriented dialogue system. We choose it because the open dialogue system is a much tougher challenge compared with task-oriented dialogue system, and is very difficult to evaluate. There are no benchmarks and labels for day-to-day conversations, and different expressions can have exactly the same meaning. Automatic evaluation for this task may only be possible after strong AI is developed when the machine can fully understand the meaning of a text, which is an AI-hard problem. On the other hand, the task-oriented dialogue system is more specific, has labels for automatic evaluation, therefore the target of the model is more explicit.

A typical task-oriented dialogue systems usually consist of an encoder, a dialogue state tracker, and a response generator. The first step is feeding the previous conversations into the encoder and converting the text conversation into a numerical vector representation. The dialogue state tracker then predicts current *dialogue state* which represents the topic or user. In the end, generate the output sentence by the generator.

The main contributions of this thesis are: we first introduced memory architecture into both dialogue state tracking task and response generation task, with our end-to-end task-oriented dialogue system model, and the result of our experiments shows that it increases the model's performance.

We discussed in three chapters the three main components of a task-oriented dialogue system, sentence encoder on Twitter post to predict accidents, memory-based dialogue system, and memory-based text generation.

The structure of this thesis is as follows: **Chapter 2** introduces core elements, tools, and techniques in the current NLP field, which will be used in our experiments. **Chapter 3** introduces a novel experiment of sentence encoder on the task of event predictions on Twitter data. **Chapter 4** presents our experiments on dialogue state tracker. **Chapter 5** introduces dialogue response generator, and the experiments on end-to-end task-oriented dialogue systems. The Final **Chapter 6** summarises the entire thesis.

Chapter 2

Fundamental elements of current NLP pipelines

This chapter will introduce some of the fundamental elements in the current NLP area, including neural networks, word embeddings, recurrent neural networks, BERT, attention mechanisms, Seq2Seq model, and end-to-end dialogue system.

The material in this chapter can be split into three parts: basics of neural networks, word and sentence embeddings, and neural network structure for dialogue systems.

The structure of neural networks is becoming more and more complex, and the size is growing even faster. Together with the rapid development of computing hardware, neural networks achieved remarkable performance in many areas. However, no matter how giant a neural network might be, the fundamental element of the network is still basic neural unit, connected

with each other by its parameters. Therefore, developing a complex neural network requires a thorough understanding of the basic neural unit. We will start with an introduction to the simplest neural unit, the perceptron, and then develop more complex structures based on it.

Following that, we will introduce different approaches to word and sentence embeddings. This part focuses on converting text data into numerical vectors, which is essential for neural network-based NLP as we can not directly input text into the neural network. Vector representation of words and sentences will undoubtedly lose part of the original information. The goal of a better word and sentence embeddings is to keep as much information as possible. Finding good embedding is one of the core sub-task for all neural network-based NLP tasks. We will go through different popular approaches, from skip-gram to BERT.

In the end, based on previously introduced elements, we will introduce some essential neural network models which are useful for dialogue system tasks and review some state-of-the-art dialogue system models.

2.1 Neural networks

Neural networks are the most fundamental element of the current rapidly developing field of artificial intelligence, or more precisely, it is the foundation of the statistics approach of artificial intelligence[30]. Intrinsically, a neural network is a giant function consisting of many neurons that input the data we are working on and output desired structured data such as class labels, dimension reduced data, etc. The function can be modified by adding or

reducing the number of neurons or changing how neurons connect, depending on the different tasks the neural network is dealing with.

Neural networks have shown their remarkable ability to solve various tasks. If classified by the type of input data, two of the main categories of tasks are Machine Vision, which deals with image data, and Natural Language Processing, which deals with human language. In this thesis, our focus is dialogue system, we will concentrate on NLP-related neural network structure.

2.1.1 Perceptron

What is a neuron in neural networks? To answer this question, let us start with the first and trivial artificial neuron: the perceptron, developed by Frank Rosenblatt[51] in 1958. A perceptron is basically a one neuron network, takes as input several numerical variables $x_1, x_2, ..., x_n$, and output a binary value y. For each input x the perceptron has a corresponding weight that indicate how important this input is, and the final binary output $y \in \{0, 1\}$is determined by $\sum_i x_i w_i$, if the sum is greater than a threshold $y = 1$, otherwise 0.

As shown in figure 2.1, a formula form of perceptron is:

$$y = \begin{cases} 0, & if \ \sum_i w_i x_i > threshold \\ 1, & if \ \sum_i w_i x_i \leqslant threshold \end{cases} \tag{2.1}$$

It is easy to see that a perceptron is just a special form of a first-order polynomial. This simple perceptron, together with activation functions that we will come up with later, is the basic unit of a neural network. The output

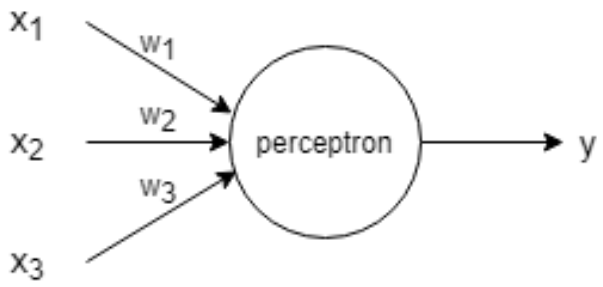

Figure 2.1: An example of a perceptron with three inputs.

of a perceptron can be new input to another perceptron. Like human brains, A neural network may contain millions of these neurons connected with each other, constructed in a complicated structure.

Now we have a basic structure of neural network, it contains multiple layers and each layer contains several neuron units. The output of each unit is a hard classification value zero or one, which is not good enough. A soft range of values can pass more information, not only class itself but the extent of it, closer to 0 or 1 represent high belief on the class. However, if we modify Formula 2.1 and only use $\sum_i w_i x_i$ as the output, although we could have soft value output, it turns out that no matter how many layers we have in our network, it is equivalent to one perceptron. In other words, it is equivalent to one linear equation, and it only has the ability to fit linear function with graph being a straight line or plane and so on. To solve this problem, we need to introduce non-linearity into our model. This can be achieved by simply applying a nonlinear function to each neuron, and this nonlinear function is also called the activation function.

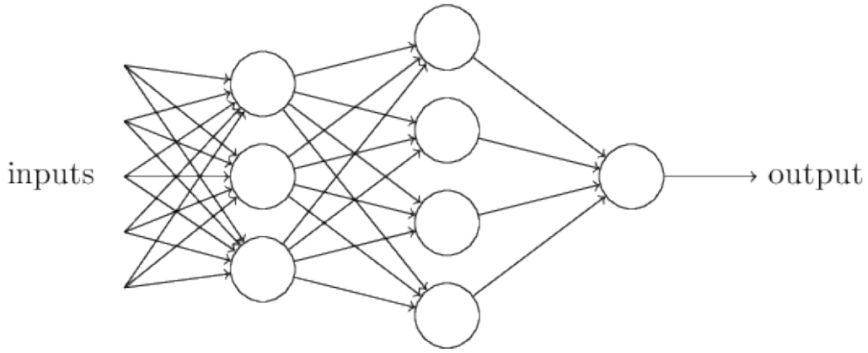

Figure 2.2: A neural network consist of multiple perceptrons[42]

2.1.2 Activation function

One of the most commonly used activation functions is the sigmoid function.

$$Sigmoid(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \tag{2.2}$$

Its figure is as shown in Figure 2.3. Applying the activation function to our neurons is also simple. We apply a sigmoid to the sum:

$$y = sigmoid(\sum_i w_i x_i) \tag{2.3}$$

Compared with the hard decision made by the plain perceptron $y \in \{0, 1\}$, now each neuron will output a soft confidence $y \in [0, 1]$.

The sigmoid function gives our model non-linearity, and thus, George Cybenko proved in 1989[13] that a single layer perceptron with sigmoid as activation function could fit all functions.

As the activation function's goal is to add non-linearity to our model, we can use even simpler functions as the activation function, as long as it is a

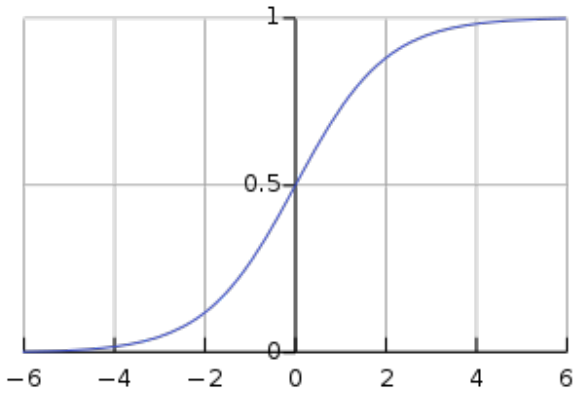

Figure 2.3: The graph of sigmoid function.

non-linear function. One of the most notable alternatives is ReLU[41], which is a straightforward function:

$$y = \begin{cases} x, & if\ x > 0 \\ 0, & if\ x < 0 \end{cases} \tag{2.4}$$

And research has shown that in many cases, ReLU can keep similar performance with sigmoid while reducing the training time needed[48]. Other commonly used activation function includes *tanh* where $y \in (-1, 1)$, and GeLU [22] and ELU[11] which are modified Relu function.

Another commonly used activation function is softmax

$$softmax(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.5}$$

This function takes as input a set of numbers and converts them into a probabilistic distribution that sums to one. Softmax activation is usually used at the last layer activation for classification tasks.

2.1.3 Gradient decent

We have a neural network ready to use, and the next step is to introduce data into our model and update model parameters according to the data. In other words, let the neural networks learn from the data.

To have a sense of the big picture, figure 2.4 shows how the model parameters $w_1, w_2, ..., w_3$ updated with the training loss, the loss represents how different between our model prediction y' differs from the ground truth label y. In this example, the loss function is simply the squared error loss y and y'. We will introduce some more delicate loss functions later on.

The goal to train our model is clear now, and we need to minimize the total loss created by our model for the data set. That is, to find a set of parameters $\theta = w_1, w_2, ..., w_n$ that minimize $\sum_i (y_i - y'_i)^2$. It is usually challenging to find an analytical solution for a large neural network. It is not applicable to compute an analytical solution as we need to compute a new solution even with a very slight modification of the network. We instead use gradient descent to approximate the solution gradually. The mathematical formula for a gradient descent algorithm is:

$$\theta' = \theta - \gamma \frac{\delta}{\delta \theta} L(\theta) \tag{2.6}$$

where θ is the original parameters, θ' is the updated parameters with one step of training. γ is the learning rate that determine how much we plan to update the parameter from the loss of current training sample. $L(\theta)$ is the loss function of our model $L(\theta) = \sum (y - y')^2$ where $y' = \sum_i w_i x_i$

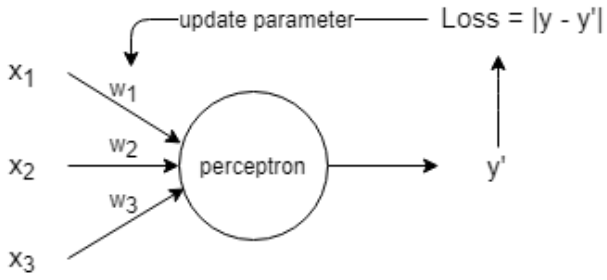

Figure 2.4: Update network parameter with loss.

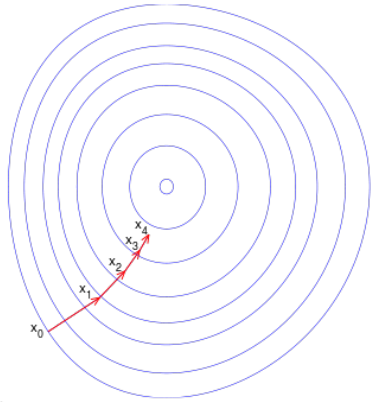

Figure 2.5: gradient descent steps.

The idea behind the gradient descent algorithm is that, as we are dealing with the optimization problem(minimize loss function), we will eventually reach the minimal point if we move at every step in the reverse direction of the gradient of the loss function.

Learning rate γ decides how long each step moves. In practice, a proper γ is essential for successful training, if γ is too large, then one step may jump over the global minimum, and if too small, it will cause much more time to finish the training.

2.1.4 Back propagation

Gradient descent requires the first-order gradient of the entire model, it works for simple networks such as previous example $y' = \sum_i w_i x_i$. But for larger multi-layer networks, it isn't easy to compute the gradient, and also, it would be too much work if we need to compute the gradient for every slightly modified version of the network. To solve this problem, the Backpropagation algorithm was invented. It only requires the partial derivatives of each neuron with its neighbors. The loss will propagate through the entire network and update the weight of each neuron by the chain rule.

Figure 2.6 shows a simple example of back propagation algorithm for one neuron. This algorithm has two steps, the forward propagation simply compute the value of $b = a_1 w_1 + a_2 w_2 + a_3 w_3$. Backward propagation step propagate inversely the error signal, which at this point it is $\frac{\delta L}{\delta a_b}$. Once we know $\frac{\delta L}{\delta a_b}$ from previous step (if current layer is not the last layer) we can easily compute the partial derivatives of w_i and L by chain rule :

$$\frac{\delta L}{\delta w_i} = \sum_{j \in J} \frac{\delta L}{\delta w_j} \frac{\delta w_j}{\delta w_i} \tag{2.7}$$

where $w_{j \in J}$ is the weights of all outwards connection of current neuron.

We now know how it works for one neuron. All we need to do next is to apply the same approach throughout all networks, update each parameter w_i with the gradient descent formula

$$w_i' = w_i - \gamma \frac{\delta L}{\delta w_i} w_i \tag{2.8}$$

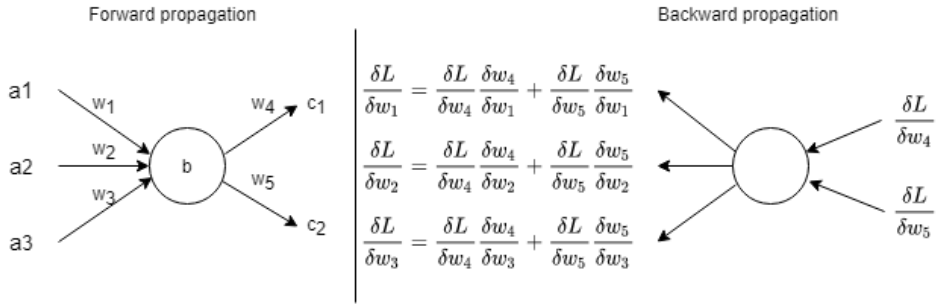

Figure 2.6: An example of back propagation algorithm.

The neural network will then update towards minimum loss and thus learn from the data set.

2.1.5 Loss function

The loss function is used to measure the difference between the model outputs and ground truth. Most trivial loss functions are previously introduced mean squared error(MSE) loss $\sum_i (y_i - y_i')^2$, this is also called L2 loss. Similarly we have L1 loss $\sum_i |y_i - y_i'|$. L2 works well in most cases, but if the data contains extreme outliers, it will have worse performance as it took a square of the difference.

For classification tasks, MSE loss has a disadvantage which it gives each class the same weights regardless of whether it is the true label. If we want our loss function to only take into account the true label, we could use Cross entropy loss

$$L = \sum_i y_i log(p_i) \tag{2.9}$$

where p_i is the probability our model predicted for ith class, generated by a

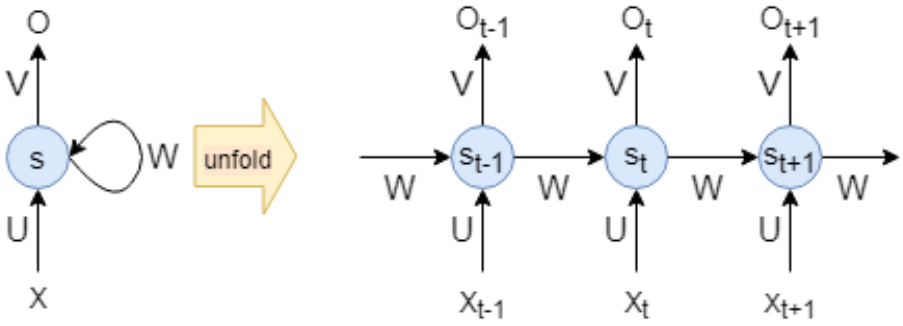

Figure 2.7: Recurrent neural network. U, V, W are trainable parameters, x is one unit of the input sequence, o is the output at each time step.

softmax function.

2.2 RNNs

We have introduced basic ideas of regular feed-forward neural networks. One of the disadvantages of it is that it requires fixed-size input. But for NLP tasks, we are dealing with text data, and it is ubiquitous to have a sentence with a different length. The Recurrent neural network (RNN) structure was introduced to solve this problem, which takes as input a sequence data. The input sequence is fed into RNN one by one, and the RNN works as a loop over itself, with each new input, the network will extend one extra layer for it.

2.2.1 RNN

Figure 2.7 shows the basic structure of an RNN. It is, in its essence, a loop of the same neuron. It consists of three trainable parameters U, V, W, and

a hidden state s, which keeps the information from the previous input of the input sequence. Each input x_t will be processed by the same parameters U, and then update the hidden state by [c1]W. Thus, at each time step, the network will create an output O_t by the parameter [c2]V. The corresponding equations are:

[c1] ~~V~~

[c2] ~~W~~

$$s_t = \sigma(Ux_t + Ws_{t-1}) \tag{2.10}$$

$$o_t = softmax(Vs_t) \tag{2.11}$$

where σ is the sigmoid activation function.

Intuitively, the hidden state s works like human short-term memory. With each new word inputted into the model, the hidden state will be updated to keep new information. Thus, after the last word has been inputted, the hidden state will have all information from the input sentence. This is why the final hidden state is usually used as the vector representation of the sentence. We will discuss this later in detail.

The idea of RNN is reasonable and makes clear sense, but in practice, it suffers from a vanishing gradient problem. If the input sequence is too long, the gradient at each layer will vanish if one layer has a small gradient because of its multiplicative nature. To deal with it, the long short-term memory network (LSTM)[23] was invented by Sepp Hochreiter in 1997, which not only can remember(by a hidden state like RNN) but also have the ability to forget.

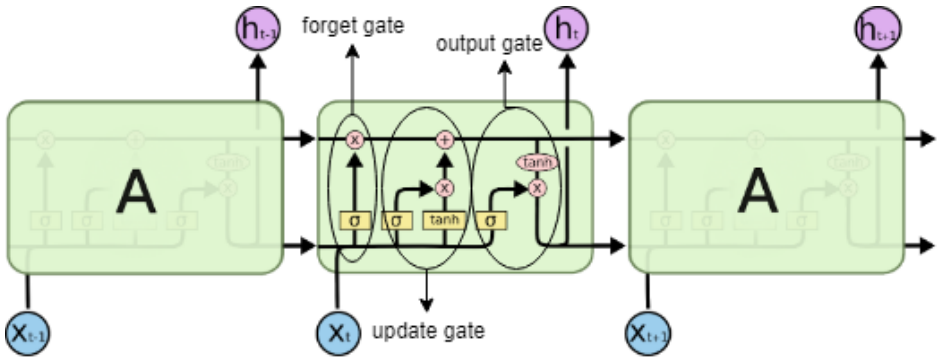

Figure 2.8: Architecture of Long short memory machine.[43]

2.2.2 LSTM

Figure 2.8 shows the architecture of a LSTM model. It consists of three parts: the forget gate, update gate, and output gate, which controls the model to forget, update, and output.

The equation for the forget gate is:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.12}$$

where W_f and U_f are trainable parameters, b_f is trainable bias vector, h_{t-1} is the hidden state of last step. This gate used the sigmoid activation function to push values into $(0, 1)$, indicate how much the model decides to forget, based on the current input and last hidden state.

Next is the update gate:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.13}$$

where W_i and U_i are trainable parameters, b_i is trainable bias vector. Same with the forget gate, this update gate decides how much the model decides

to update, but this is only half the update gate. The model still needs to know which direction to update:

$$\widetilde{c}_t = tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{2.14}$$

where W_c and U_c are trainable parameters, b_c is trainable bias vector. Note we choose to use *tanh* instead of *sigmoid* as the activation function, as *tanh* output to $(-1, 1)$ so it decides which direction to update. The final update of the model would then be $i_t \circ \widetilde{c}_t$, where $\circ$ is pointwise product.

We are now able to update the cell state vector, usually noted as c_t, as follows

$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t \tag{2.15}$$

Finally, the output gate, similar to the update gate, the output is based on hidden state c_t, but we need first to decide what and which direction to output:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.16}$$

where W_o and U_o are trainable parameters, b_o is trainable bias vector.

$$h_t = o_t \circ tanh(c_t) \tag{2.17}$$

2.2.3 GRU

Gated recurrent unit(GRU)[10] is a variant of LSTM, as shown in figure 2.9. It combines the forget gate and update gate, and combines cell state and hidden state, reducing the number of parameters by around a half. The mathematical equation for GRU is:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.18}$$

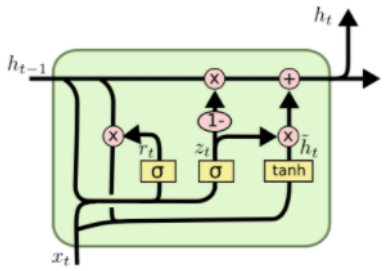

Figure 2.9: Architecture of gated neural unit.[43]

This gate controls how much to forget and how much to learn.

As LSTM, we compute how much and which direction to update:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.19}$$

$$\widetilde{h}_t = o_t \circ tanh(c_t) \tag{2.20}$$

Finally, the hidden state is computed by:

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \widetilde{h}_t \tag{2.21}$$

It is a more straightforward setting to use z_t to determine how much to learn and $1 - z_t$ to decide how much to forget. Research has shown GRU can significantly save training time while keeping similar performance with LSTM[10].

2.3 Word embedding

We have introduced the basic idea of a neural network. However, we can not directly use the neural network to deal with human language. This is because the original neural network can only take as input the numerical data; there

is no way we can feed the text directly into our neural network. To solve this problem, we need to convert textual data into numerical representations and then feed it into the neural network. This approach is known as a distributed representation of words or word embedding.

The trivial method to convert text data into numerical data is to use the one-hot vector. Each independent word in the vocabulary set will have a unique one hot vector $(0, 0, ..., 1, ..., 0, 0)$ as its vector representation. However, this simple method does not preserve any semantic information of each word, we all understand that *man* and *woman* are closely related word, but if we use the one-hot vector, they are just two independent indexes, there is no difference between (*man*, *woman*) and (*man*, *computer*).

To preserve the semantic information, we want to make the representation of related words such as *man* and *woman* similar, whereas *love* and *hate* very different. Skip-gram and Glove embeddings are the two most popular tasks to train the word embedding. We will introduce both below. Note that these tasks are just specifically designed to learn word embeddings. In fact, for any NLP task, during the training process, the word embedding can be updated to fit the task better, or each task can learn its own word embedding starts with one-hot representation, and this is what the embedding layer will do in the two most popular AI platform Pytorch and Tensorflow if the word embedding is not pre-defined. But as the Skip-gram and Glove task are trained with a much larger data set, the quality of the word embedding is usually much better compared with embedding trained on a specific task with smaller data set. Word embedding is usually referred to as a form of

transfer learning[53].

2.3.1 Skip gram

The most commonly used method to convert text into embeddings is called the skip-gram algorithm introduced by Mikolov in 2013[37]. The basic idea of the skip-gram algorithm is to find vector representations of words that predict their nearby words in a sentence.

Given a sequence of training words $w_1, w_2, ..., w_n$, we first define a window size s, then generate pairs of words.

$(w_1, w_2), (w_1, w_3), ..., (w_1, w_s), (w_2, w_3), ..., (w_2, w_{s+1}), ...$

These pairs of words are used as training samples and are fed into a simple neural network that contains input, hidden, and softmax output layer. At the input layer, each word is represented as a one-hot vector of dimension $1 \times V$, where V is the size of the vocabulary set. Then it is multiplied with a hidden layer matrix with dimension $V \times D$, where D is the hidden dimension and will be the dimension of learned embeddings. In the end, it passes to the output layer with a matrix of dimension $D \times V$ with softmax, each element of the output is the prediction of a nearby word of the input word, and the training target will be a one-hot vector of the nearby word in the training pairs.

As mentioned before, our goal is not the model itself. Instead, we will use the hidden layer matrix with dimension $V \times D$ as the trained embedding lookup table the matrix is just the learned weight of the neural network. Each row represents a D dimensional vector representation of a word.

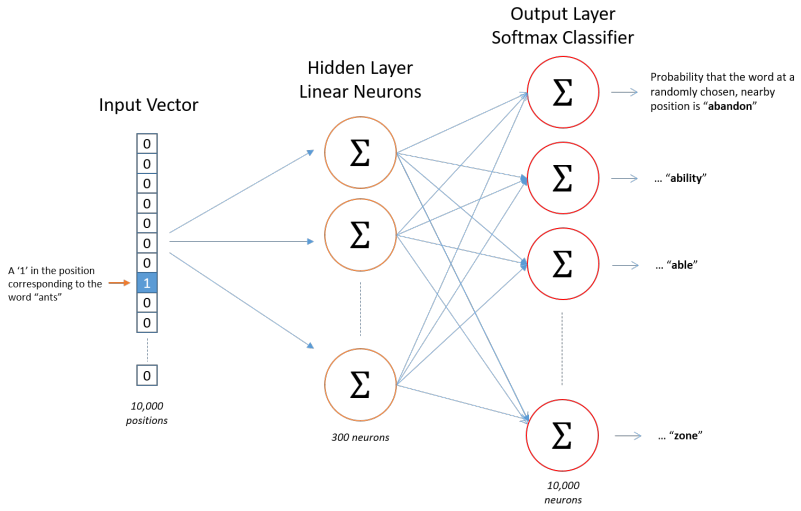

Figure 2.10: The simple neural network used for transfer learning the word embeddings[37]

Sub-sampling

One problem with the current model is that there will be many training instances for common words. Therefore the output probability will be evenly distributed for many words, and thus it will not contain much useful information for learning good vector representation. The Mikolov paper[39] deals with this problem by a sub-sampling scheme, where each word w_i in the training set is discarded with a probability computed by the formula:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{2.22}$$

where $f(w_i)$ is the frequency of words w_i and t is a chosen threshold, typically around 10^{-5}.

Negative sampling

There will be a massive amount of training pairs generated by the window size to train the current model. Each training instance will update $V \times D$ parameters, which is very computationally expensive. To deal with this, the paper introduced a negative sampling mechanism. Intuitively speaking, it is to update only a small amount of word embeddings each time where negative samples are randomly selected.

2.3.2 Glove: Global Vectors for Word Representation

Similar to Skip-gram models, the Glove embeddings [46] also learns the word vector representation by the co-occurrence information. The difference is that while the skip-gram learns the vector to improve the predictive model, the Glove model first builds a co-occurrence matrix based on the counts of co-occurrence and then makes dimension reduction on it. In detail, for each word w_i in the vocabulary (one row of the co-occurrence matrix), each column represents the counts you see the two words appear in near position. Then normalize the counts and log-smoothing it and find a lower-dimensional representation by minimizing a reconstruction loss.

The glove embedding was trained on a corpus of 42B words and is available freely on the internet[c0].

2.4 Sentence embedding

Similar to word embedding, sentence embedding is the vector representation of a whole sentence. While word embedding tries to preserve as much se-

[c0]https://nlp.stanford.edu/projects/glove/

mantic information of a word as possible, sentence embedding needs to keep semantic information and some structured information such as simple logic and reasoning of a sentence. It hugely improves the difficulty of the task.

The state-of-the-art approaches of sentence embedding fall into three categories: average word embedding [2], recursive neural network [55], skip thought [28]. We will take a look at each of them and introduce some state-of-the-art models in detail.

2.4.1 Average embedding

Unweighted averaging of word embeddings has been found to perform well in presenting short phase [39], [67] introduced an approach that learns the sentence embedding by starting with standard word embeddings and modifying them based on supervision from the Paraphrase pairs dataset(PPDB), and constructing sentence embeddings by training a simple word averaging model. This model is further simplified by [2] which claimed to achieve the state-of-the-art result.

Their approach is straightforward in structure, first compute the weighted average of a sentence from standard word embeddings, then remove the projections of the average vectors on their first principal component. The weight of a word w, which they called *smooth inverse frequency*(SIF) according to the paper, is:

$$\frac{a}{a + p(w)} \tag{2.23}$$

where a is a parameter and $p(w)$ is the estimated word frequency.

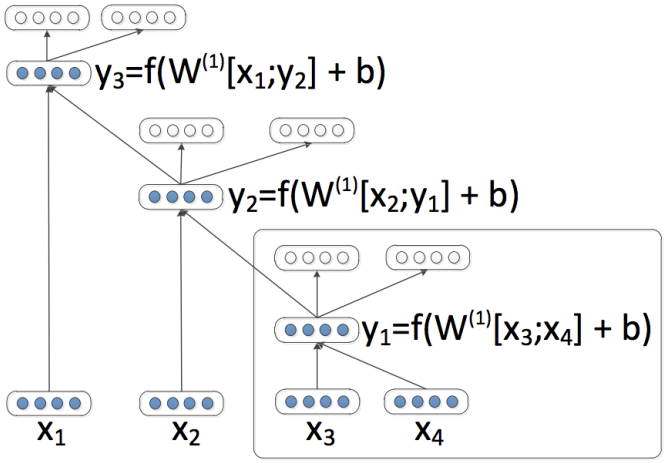

Figure 2.11: An illustration of how recursive neural network works. Nodes that are not filled are only used to compute the reconstruction error.[55]

The steps to find a vector representation v_s given word embeddings $v_w : w \in V$, parameter a and word frequency $p(w) : w \in V$ is:

1. Find weighted sentence embedding:

$$v_s = \frac{1}{|s|} \sum_{w \in s} \frac{a}{a + p(w) v_w} \tag{2.24}$$

2. Remove first principal component of the averaged vector:

$$v_s = v_s - u u^\top v_s \tag{2.25}$$

This model trained on Glove embeddings achieves the state-of-the-art result on the Semantic Text Similarity database, beats Deep average network, Tf-IDF weighted average, and many other models.[46]

2.4.2 Recursive neural network

Richard Socher[55] introduced Recursive auto-encoder (RAE) to find the vector representation of a sentence, as shown in 2.11. The idea is to first generate a tree structure of the sentence by repeatedly choose two continuous words to merge, and the way of picking words is by minimizing the reconstruction loss. Then combine embedding of words according to the tree structure.

In more detail, given a sequence of words $s = w_1, w_2, w_3, ..., w_n$, the first step is to merge each continuous pairs of words embeddings to a parent vector p by:

$$p = f(W^{(1)}[c_1; c_2] + b^{(1)}) \tag{2.26}$$

Where $c_1; c_2$ are either an input word vector or a non-terminal node in the tree, $[c_1; c_2]$ is the concatenation of the two vectors. $W^1 and b^1$ are parameters of neural network.

After merging each pair, we instantly decode the obtained vector p via:After merge each pairs, we instantly decode the obtained vector p via:

$$[c_1^{'}; c_2^{'}] = W^{(2)}p + b^{(2)} \tag{2.27}$$

And compute the reconstruction error by:

$$E_{rec} = \frac{1}{2}||[c_1; c_2] - [c_1^{'}; c_2^{'}]|| \tag{2.28}$$

We merge the pairs that give the slightest reconstruction error and repeat this step until we merge the whole sentence to one vector representation.

2.4.3 Skip thought

The other popular approach is the Skip Thought vector proposed by [28]. Similar to skip-gram models for obtaining word embeddings, the skip thought model tries to get sentence embeddings by encoding a sentence to predict sentences around it.

The model is in the framework of encoder-decoder models. That is, giving a sentence in the corpus it is first encoded to a vector representation and then a decoder is used to generate the surrounding sentences. In detail, suppose we are given a tuple s_{i-1}, s_i, s_{i+1}, let w_i^t be the t-th word of sentence s_i and $\mathbf{x}_i^t$ be its word embedding.

The first step is to encode the sentence by a GRU, following the equation 2.18-2.21, and then use the final hidden state $\mathbf{h}_i^N$ as the vector representation of the whole sentence s_i.

At decoder time, two analogous modified GRU are used to generate the next and previous sentences. It is similar to the encoder except they introduced $\mathbf{C}_r, \mathbf{C}_z$ and $\mathbf{C}$ to bias the reset gate, the update gate, and hidden state computation.

$$\mathbf{r}^t = \sigma(\mathbf{W}_r \mathbf{x}^t + \mathbf{u}_r \mathbf{h}^{t-1} + \mathbf{C}_r \mathbf{h}^i) \tag{2.29}$$

$$\mathbf{z}^t = \sigma(\mathbf{W}_z \mathbf{x}^t + \mathbf{u}_z \mathbf{h}^{t-1} + \mathbf{C}_z \mathbf{h}^i) \tag{2.30}$$

$$\overline{\mathbf{h}}^t = tanh(\mathbf{W}\mathbf{x}^t + \mathbf{U}\mathbf{r}^t \odot \mathbf{h}^{t-1} + \mathbf{C}\mathbf{h}^i) \tag{2.31}$$

$$\mathbf{h}_{i+1}^t = (1 - \mathbf{z}^t) \odot \mathbf{h}^{t-1} + \mathbf{z}^t \odot \overline{\mathbf{h}}^t \tag{2.32}$$

Given $\mathbf{h}_{i+1}^t$, the probability of word w_{i+1}^t is:

$$P(w_{i+1}^t | w_{i+1}^{<t}, \mathbf{h}_i) \propto exp(\mathbf{v}_{w_{i+1}^t} h_{i+1}^t) \tag{2.33}$$

Where $\mathbf{v}_{w_{i+1}^t}$ denotes the row of V corresponding to the word of w_{i+1}^t, and V is the weight matrix represent a distribution over words based on the hidden states.

The objective function of this model given s_{i-1}, s_i, s_{i+1} is then optimize the sum of log-probabilities for the previous and next sentences conditioned on the encoder representation:

$$\sum_t log(P(w_{i+1}^t | w_{i+1}^{<t}, \mathbf{h}_i)) + \sum_t log(P(w_{i-1}^t | w_{i-1}^{<t}, \mathbf{h}_i)) \tag{2.34}$$

The paper[28] evaluated the skip-thought approach on eight different tasks: text classification, semantic-relatedness, and paraphrase detection. In each task, they first convert a textual sentence to skip thought vectors, freeze the vector, use it as features in downstream classifications, and obtain competitive results.

2.4.4 BERT based word and sentence embedding

One of the most significant breakthroughs in NLP in recent years is BERT, Pre-training of Deep Bidirectional transformers for Language Understanding[16]. It is a language model based on Transformer encoder [59].

Figure 2.12 shows one step of the transformer encoder. The input sentence is first converted to a sequence of word embeddings, then add positional embedding to each word embedding to add positional information, as the transformer is not an RNN-like structure. Then the word embeddings are fed to a multi-head attention layer and MLP afterward with the residual connection. Each step is a normal process except the multi-head attention, and we will discuss more on it next.

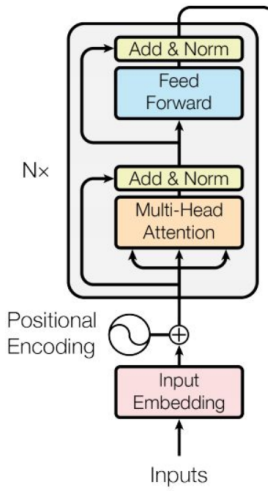

Figure 2.12: One step of the transformer encoder. [59]

The attention here in the encoder is a self-attention that measures how strongly each other word is related to the current word. In detail, for word w_i in sentence $w_1, ..., w_n$, we will have three matrices Q, V, K that project each word embedding to queries, keys, and values, which is all vectors. E.g., for word w_i it is converted to query q_i and a similarity score is computed by dot product of q_i and keys for other word $k_{j,j\in 1:n,j\neq i}$. The score is the attention score of how much other word is related to w_i. Then the word embedding of w_i is the weighted sum of values of other words $v_{j,j\in 1:n,j\neq i}$. This way, each word will contain information from the whole sentence. The parameter of each project matrix Q, K, V is randomly initialized and updated during training.

Multi-head attention is a set of the independent self-attention process and concatenates all of the final embeddings. BERT is simply doing the Transformer multiple times and use it as an encoder to do a language model

task.

Similar to skip-gram, BERT trains to predict unseen words in a sentence. It marks 15% of the word in the original sentence as a special $[Mask]$, and the task is to predict it based on surrounding words. During training, one sentence in the data set will be sent to the model repeatedly. There is an 80% chance for the masked word that it is replaced by $[Mask]$ token, and 10% chance to be replaced with a random word and a 10% chance to stay unchanged. This is to prevent the model from memorizing the sentence.

As an example, given the sentence *Joe Biden is the new president of US*. The first step is to convert each word into word embedding. It could be randomly initialized or generated with an algorithm such as skip-gram.

$$e_{Joe}, e_{Biden}, e_{is}, e_{the}, e_{new}, e_{president}, e_{of}, e_{US} \tag{2.35}$$

Then add each embedding with the positional encoding by bitwise addition to add position information into the word embedding. It is computed by:

$$p_i = \begin{bmatrix} sin(w_1 t) \\ cos(w_1 t) \\ sin(w_2 t) \\ cos(w_2 t) \\ ... \\ sin(w_{d/2} t) \\ cos(w_{d/2} t) \end{bmatrix} \tag{2.36}$$

where d is the desired dimension of positional encoding, which is divisible by 2, t is the position of the current word we are computing, and:

$$w_k = \frac{1}{10000^{(2k/d)}} \tag{2.37}$$

Then, each word embedding is processed with three independent matrices Q, K, V, all with dimension d by h where d is the embedding dimension, and h is the hidden size. The Q, K, V are trainable parameters of the model and are randomly initialized.

$$e^q_{Joe}, e^q_{Biden}, e^q_{is}, e^q_{the}, e^q_{new}, e^q_{president}, e^q_{of}, e^q_{US}$$
$$e^k_{Joe}, e^k_{Biden}, e^k_{is}, e^k_{the}, e^k_{new}, e^k_{president}, e^k_{of}, e^k_{US}$$
$$e^v_{Joe}, e^v_{Biden}, e^v_{is}, e^v_{the}, e^v_{new}, e^v_{president}, e^v_{of}, e^v_{US}$$

Here the k, q, v stands for key, query, and value, respectively. The keys and queries are used to compute the attention score, and values are used as the output value.

For the word Joe, firstly, we use its query e^q_{Joe} to do dot product with all keys of the sentence. We have attention scores for Joe with each word. Then, we multiply the value of each word with the attention scores and sum all of the multiplied values together, and the sum is the new word embedding of Joe. And we repeat this process to all words in the sentence.

For sentence embedding, there are several options, we can use the start of the input token $[CLS]$ as the representation of the whole sentence, or we can use the average of all word embeddings of the sentence, or we could use the maximum value at each position of all word embedding as the sentence embedding. Experiments[50] shows that the average embedding performs best.

We have introduced several different approaches for word and sentence embeddings, including skip-gram, Glove, average embedding, recursive neural network, skip-thought and BERT. In our experiments, we choose to use Glove embedding as the word embedding, as it is reportedly that it outperforms the skip-gram[46]. For sentence embedding on the other hand, the average embedding is the most trivial approach and performance is weaker compared with later work such as skip-thought[28], the rest approaches can be classified into two categories: GRU based and BERT based. Both Skip-thought and Recursive neural network used the last hidden state of a GRU as the sentence embedding, and BERT, as introduced before, used a special attention model together with average of word vector as the sentence embedding. In our experiments, we will try both the GRU based and BERT based sentence embedding.

2.5 Memory

While word and sentence embedding trying is using numerical vectors to preserve semantic information of text, syntactic information and logic within multiple sentences can also be very important. As introduced before, a group of recurrent neural networks (RNN) was proposed to deal with sequential data, including long short term memory machine (LSTM) [23] and gated recurrent unit (GRU) [10]. The idea is to have a hidden state vector that is updated by each word of the sentence. This hidden vector will then catches some structure information of the sentence. This model has been applied on many NLP tasks and gained popularity for its expertise in dealing with sequential data. And in a more extensive perspective, the hidden state vector

can be regarded as the first attempt to represent memory/syntactic information of a sentence.

However RNN models have difficulties dealing with long sequence, it suffers from gradient vanishing problem as mentioned before. Also, it only has a minimal ability to represent complex syntax/logical structures, especially when there are multiple sentences. It takes each input one by one with the same process. BERT does not have a gradient vanishing problem, but still, all it does to multiple sentences is to add $[SEP]$ token between sentences and left to the model as a black box to process the input.

Jason Weston et al.[66] proposed memory network models to deal with question answering tasks from conversations with multiple sentences. They introduced a memory vector which is updated multiple times following the conversations. The model contains for separate modules:

- **Input module**: Convert input text and question to vector representation.
- **Generalization module**: Update the memory vector based on attention mechanism.
- **Output module**: Output the memory vector.
- **Prediction module**: Make predictions based on the memory vector.

These four modules are trained independently.

2.5.1 MemN2N

Sainbayar Sukhbaatar et al.[57] proposed MemN2N, an end to end trainable version of memory networks on question answering tasks. Its basic structure is similar to a memory network.

Given an input passage set $x_1, ..., x_i$ where each x_i is a sentence in the passage, and a query q. The first step is to convert each x_i to a vector representation m_i with fixed dimension d, and the query q to vector u with same dimension d. This can be done by summing up each word embeddings in the passage and query respectively from an embedding matrix A of size $d \times V$, where V is vocabulary size.

The next step is to compute the attention vector, which is the inner product of m_i and u, and do a softmax over it.

$$p_i = Softmax(u^\top m_i) \tag{2.38}$$

which is, in fact, the match of m_i and u, this attention vector will take the duty of pointing out which part of the sentence should be focused on given the query.

Now we convert each $x_1, ..., x_i$ to vector representation $c_1, ..., c_i$ again based on the other embedding matrix C, and compute the memory vector o by multiply each c_i with attention p_i and sum over all sentences in the passage:

$$o = \sum_i p_i c_i \tag{2.39}$$

The final step is to make predictions by a simple neural network with softmax based on the memory vector o and query vector u:

$$a = Softmax(W(o + u)) \tag{2.40}$$

A loop version is also proposed to catch more structured information. After obtaining o_k, u_k we update the query vector by:

$$u^{k+1} = u^k + o^k \tag{2.41}$$

And then, based on the updated query vector u^{k+1} we do the previous step again to obtain an updated memory vector o^{k+1}, and do this sufficient times and make a prediction.

In the paper, they evaluated a three layers MemN2N model on the Facebook Babi dataset [65], shown that the memory network architecture beats LSTM and reaches a competitive result.

2.5.2 Dynamic memory network

Kumar et al. [29] proposed a more sophisticated memory network model, where they called dynamic memory network (DMN).

While keeping the basic idea of learning a memory vector and make a prediction based on it, their main contributions are:

1. Using GRU to form the vector representation of sentences, rather than simply the sum of word embeddings, use GRU to update the memory vector, rather than sum overall sentences.

2. The attention is computed by a two-layer feed-forward neural network taking a feature vector of:

$$[c, m, q, c \circ m, c \circ q, |c - q|, |c - m|, c^\top W^{(b)} m, c^\top W^{(b)} q]$$

where c is the input vector representation, m is the memory vector representation, q is the question vector representation,$\circ$ is the element-wise product.

They also evaluated the Facebook Babi dataset, and the DMN did outperform the MemN2N slightly.

2.5.3 Summary of Memory architecture

We have discussed the original memory network, the MemN2N, and the Dynamic memory network, the essence of the memory architecture is a *memoryvector* that represents current knowledge the model knows about a specific task, and attentions updated this memory to different sentences of input context.

Compared with sentence representation obtained by RNN, memory architecture have more potential ability to preserve syntactic information of the context, as it has sentence level attentions and more complex structures to handle sentence representations. This is particularly useful for context with multiple sentences, especially when these sentences have complex logical relations.

In BERT, if the input context includes multiple sentences, each sentence is separated by a special token $[SEP]$, and then feed all as a long sequence into the following massive network. Then the network works like a big black box, with 12 layers of transformer encoder. It can preserve syntactic information

more than RNN, but it still does not have a special structure, particularly for catching sentence-level logic and information.

2.6 Dialogue system

The dialogue system is one of the core tasks of natural language processing and probably the hardest one. It aims to establish a computer agent that can communicate with a human by natural language. In the past, dialogue systems were usually established by rule-based approaches [62][61], that is, to manually define responses to a set of possible queries made by experts. This approach is still widely applied in commercial systems, as it is very reliable when the query to the system is on the list. But apparently, this approach suffers from low generality, and it can not respond to any queries that are not on the list.

With the development of deep learning algorithms, neural network-based dialogue system, or more generally, statistical dialogue system, has attracted more and more attention. This approach requires massive dialogue text as training data instead of the predefined responses manually made by experts. As long as there are similar expressions in the training data, the model will know how to reply, and therefore its generality is much better than the rule-based approach. However, as the model itself generates the responses, it is less stable than the predefined responses in the rule-based approach.

As mentioned in the introduction chapter, there are mainly two types of dialogue system tasks: the open dialogue system and the task-oriented dialogue system. The open domain dialogue system aims to respond to any

Table 2.1: An example of dialogue states at each turn of the dialogue.

utterance	dialogue state
User:I would like to find a cheap restaurant that serves tuscan food **System**: nothing is matching your request. I'm sorry.	(restaurant, food, tuscan) (restaurant, price range, cheap) (restaurant, name, not mentioned) (restaurant, area, not mentioned)
User:Could you help me find some cheap Italian food then? **System**: If you do not have a preference of area, I recommend La Margherita in the west.	(restaurant, food, italian) (restaurant, price range, cheap) (restaurant, name, not mentioned) (restaurant, area, not mentioned)

user input, including day-to-day dialogue, greetings, etc. The task-oriented dialogue system aims to solve a specific type of request from the user, such as restaurant/hotel booking. Its training data set is usually labeled and domain-specific. In this thesis, we will focus on a task-oriented dialogue system.

2.6.1 Dialogue state tracking

One of the essential intermediate step of dialogue system is the dialogue state tracking (DST), or belief tracking. In this step, the task is to recognize the user's goal as *dialogue state*, which will then be used to guide the system response correspondingly[6]. Table 2.1 shows an example of a dialogue with the corresponding dialogue states at each turn of the dialogue, which will be updated at every user utterance.

DST systems can be classified into two types: ontology-based and ontology-free. Ontology-based DST [49, 71] requires a set of pre-defined possible values for the dialogue state, and the model selects the most likely one from the ontology set. The Ontology-free DST [69, 19], on the other hand, does not

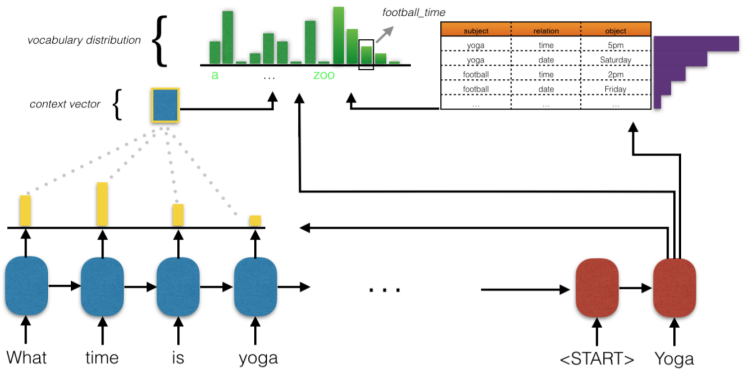

Figure 2.13: Key-value retrieval network [18]

require such an ontology set, and will choose the most likely phrase from the dialogue history and vocabulary set.

The ontology-based DST will require a manually defined set of values for all slots, which will be expensive for large-scale dialogue systems and makes it difficult to generalize the model. On the other hand, the ontology-free DST is easy to generalize for large-scale dialogue systems but can not utilize expert knowledge in the model.

The two types of DST can be incorporated by using a gate function that determines if the current slot should be filled by choosing from an ontology or the dialogue history [47].

2.6.2 Key-value network

One of the limitations current NLP models faced is that the model only has access to its text corpus data set; therefore, all the model can process is a long sequence of language symbols. Everything it learns is just the

rules of how those symbols are ordered. Human, instead, learns language by connecting those language symbols with the real-world, linguist Saussure called the language symbol the *signifier* and the relative stuff in real-world the *signified* [15].

As an example, when we first learn the word *apple*, we will see and touch the apple, taste it, cook it, and so on. Therefore, we establish the signifier and signified of apple in our mind. NLP algorithm, however, only have access to its signifier, it probably learns that the word *apple* is very likely following with another word 'pie' if the corpus is large enough, but it can never answer questions such as 'what is the taste of the apple ?' unless there are such question-answer pairs in the corpus, as it does not link to the real world.

As a solution, incorporate NLP models to the Knowledge graph of the Knowledge base (KB) has been introduced, originally in question answering tasks [64]. A KB consists of a large amount of object-relation-subject triples such as $(meeting, time, 3pm)$

For the dialogue system, Mihail Eric et al. introduced the key-value retrieval network [18], which connected each conversation in a task-oriented dialogue system with a small knowledge base, which all triples in the small KB are related to the current dialogue.

The key-value retrieval network has an encoder-decoder structure as a normal Seq2Seq dialogue system, but at the output step of the decoder, it not only selects candidate word from the corpus vocabulary set but also selects from the small KB. As an example, a triple $(meeting, time, 3pm)$ is

firstly converted to object as the value, and subject relation pairs as the key, say, $key : meeting_time, value : 3pm$. When making predictions, the model not only searches from the vocabulary but also searches all the keys and gives a probability distribution over all vocabulary plus all the keys. If one of the keys has the largest output probability, then its relevant value will be output. E.g., *The meeting is at* _, and the model select key *meeting_time* as output, it will actually output value *3pm*.

Some other networks were trying to connect the model with knowledge, e.g., Andrea Madotto et al. introduced Mem2Seq model[35], which also deals with task-oriented dialogue system with a small relative KB. The difference is that for each triple of the KB, a feature vector is extracted instead of key-value pairs.

The limitation of the current mechanism is the KB is too small, and it has to be manually selected to be related to the current conversation. Connecting the NLP model with KB is one of the initial efforts to add external real-world information to the model. Other attempts to do so, such as multi-model NLP[27] which gives the NLP model a direct link to the real world, such as sense of sight and hearing. Also, there are attempts to introduce common sense into the NLP model[56].

2.6.3 Seq2Seq network

One of the building blocks for a dialogue system is the Seq2Seq model. Intrinsically, any model given a sequence of text as input and then produces a sequence of responses can be regarded as a Seq2Seq model, and dialogue system task naturally fit in this schema.

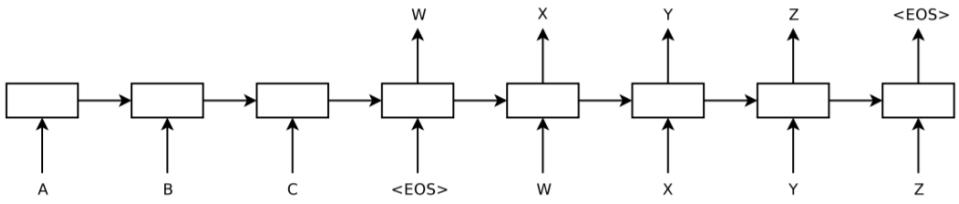

Figure 2.14: Sequence to sequence for machine translation[58].

Seq2Seq model was introduced by Ilya Sutskever et al.[58] for machine translation task. In a machine translation task, we have a source language sentence, say, ABC, and target language sentence $WXYZ$. The base model is a long short term memory network (LSTM)[23], and at each step of the LSTM, it takes as input one word from the source sentence. After the whole source sentence is fed into the model, it starts to predict the first word of the target sentence, and the predicted word is then fed into the network at the next step and so the model recursively generate a long sentence until a special token $< eos >$ stands for the end of the sentence is predicted.

Dzmitry Bahdanau et al. [4] introduced an attention mechanism into the normal Seq2Seq model. In their work, the translation model is split into an encoder and a decoder, where the encoder is a bi-directional RNN that the last hidden state of the RNN is used as the vector representation of the source sentence, and fed as the initial hidden state of the decoder RNN, which then predicts the whole target sentence. The attention mechanism is applied at the decoder stage, at each step of the decoder, it takes the previous predicted word y_t, the previous hidden state s_t and attention-based context vector c_t as input. We will discuss this in more detail next.

Encoder: The encoder is essentially a Bi-directional RNN. The forward RNN reads the input sequence as it is ordered, and generate a sequence of forward ordered hidden states $h_{1,forward}, h_{2,forward}, ..., h_{n,forward}$. The backward RNN read the input sequence in reverse order and generate a sequence of backward ordered hidden states $h_{1,backward}, h_{2,backward}, ..., h_{n,backward}$. For each word of the input sentence, we now have a forward state and a backward state, and the final representation h_i will be the concatenation of the two states $[h_{i,forward}, h_{i,backward}]$. This representation is later used to compute context vectors and is used during the decoding process.

Decoder: The difference with a normal decoder is that here at each step, the decoder has access to the vector representations of input words with an attention mechanism. As shown in figure 2.15, at step i, for jth hidden state, the attention weight is computed by:

$$\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k=1}^{n} exp(e_{ik})} \tag{2.42}$$

Where

$$e_{ij} = a(s_{i-1}, h_j) \tag{2.43}$$

is a similarity score indicate how well the word representation and the decoder hidden state match.

Then the content vector c_i is computed via

$$c_i = \sum_{j=1}^{n} \alpha_{ij} h_j \tag{2.44}$$

Intuitively, the context vector is just the weighted sum of the word embeddings of each word in the context, and the weight for each word is the

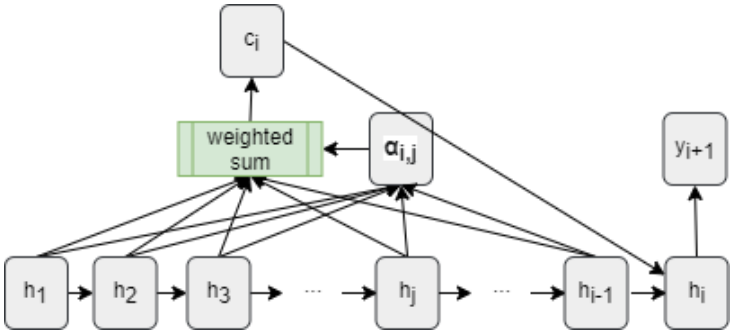

Figure 2.15: Bahdanau attention mechanism.

similarity score between its embedding and current decoder hidden state, which represent how related is the word with current state.

Seq2Seq model as a generator suffers from the exposure bias problem, like all generators that generate words one by one. The generation of the current word will depend on previously generated words, therefore once the model made a mistake, it will affect all the following generations, and the bias will be accumulated and amplified with the generation going on. Efforts to solve this problem have been made, such as to use partly ground truth target and partly generated target when training the model[70], but so far, this is still an open question to the field. In our work, we will adopt the Seq2Seq model as the skeleton of our generator. More detail will be introduced in chapter 5.

2.6.4 Pointer network and copy mechanism

In human conversation, quite often, the phrase we used can be found in a dialogue history. For example, A:*Do you like Peaky Blinders?* B: *Yes, Peaky Blinders is my favorite.* In this case, the phrase $PeakyBlinders$ is directly copied from question to response. To better deal with this situation in the

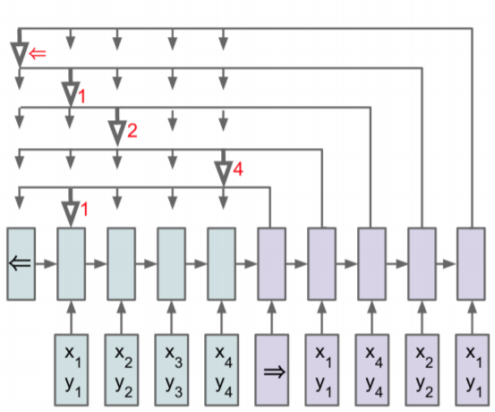

Figure 2.16: Pointer network [60]

NLP model, Oriol Vinyals et al. introduced pointer network [60], a modified Seq2Seq model that each element of the output sequence is directly from the input sequence. Their paper chooses Convex Hull as an example, and a simpler example is sorting a sequence of numbers. As shown in figure2.16, at each time step of decoding, the network points to one element of the input sequence, so it is named as pointer network.

Pointer network in NLP tasks is also known as Copy mechanism [72], it outputs either word from vocabulary, or copy from input sentence, which might be a single word or a phrase of continuous words. In our experiments we will use this approach as part of our sentence generator.

2.6.5 End-to-End dialogue system

The next step is to build an end-to-end neural network-based dialogue system with the state tracker and generator ready. Tsung-Hsien Wen, et al. [63] proposed a neural network-based model for task-oriented dialogue systems.

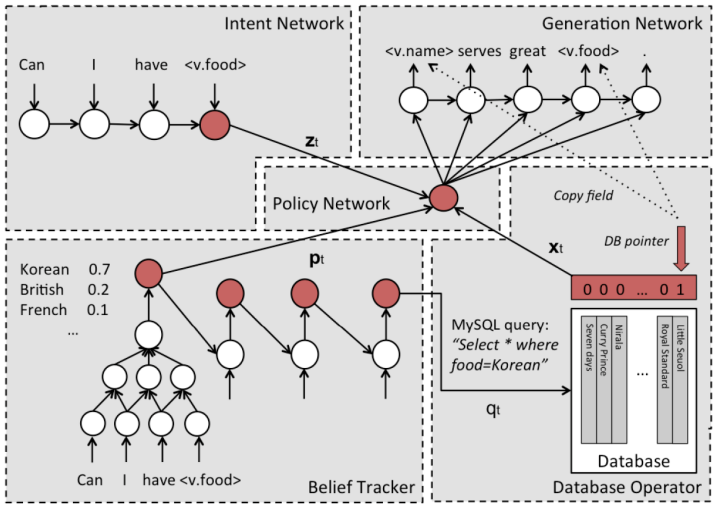

Figure 2.17: neural network-based model for task-oriented dialogue systems.[63]

The model consists of a belief network, an intent network, a database operator, a policy network, and a generation network. As shown in Figure 2.17 This architecture has three essential components:

Sentence encoder: the intent network encodes the query sentences into sentence representations.

Dialogue state tracking: the belief tracker predicts current dialogue states.

Response generator: the generation network generates system output. Note the database operator enables the generator output text in the knowledge base.

Given a sequence of words $w_1^t, w_2^t, ..., w_N^t$ from user (the query sentence), the system first converts them into two internal representations. One is

distributed representation of the sentence $\mathbf{z}$ generated by the intent network by an LSTM:

$$\mathbf{z}^t = LSTM(w_1, w_2, ..., w_N) \tag{2.45}$$

Another is the belief tracker, which tries to maintain a multinomial distribution p over $v \in V_s$ for each informable slot and binary distribution for each requestable slot. A slot-value pair s-v is a state indicator of conversation. The informable slots are the user's constraints, such as the food types or prices; the requestable slots are slots that the user can request value for, such as the address. So in the restaurant situation, the belief tracker is basically predicting what type of food the dialogue is currently talking about, the range of accepted price, etc. Each slot will have its tracker, and the probability is updated by:

$$f_v^t = f_{v,cnn}^t \oplus p_v^{t-1} \oplus p_\phi^{t-1} \tag{2.46}$$

$$g_v^t = \mathbf{w}_s \cdot sigmoid(\mathbf{W}_s \mathbf{f}_v^t + \mathbf{b}_s) + b_s' \tag{2.47}$$

$$p_v^t = \frac{exp(g_v^t)}{exp(g_{\phi,s}) + \Sigma_{v' \in V_s} exp(g_{v'}^t)} \tag{2.48}$$

where $\oplus$ is vector concatenation, $f_{v,cnn}^t$ denote the sentence feature obtained by a CNN [40], p_v^t is the probability of all possible values from ontology for current slot, and p_ϕ^t is the probability that the user has not mentioned that slot up to turn t.

Based on the output p_s^t, a query to the knowledge base q_t is formed by:

$$q_t = \cup_{s' \in S_I} argmax_v P_{s'}^t \tag{2.49}$$

Where S_I is the set of informable slots. Then the query is searched on the knowledge base and generates a binary value vector x_t over entities of the knowledge base, where a 1 indicates that the corresponding entity is consistent with the query. If x_t has a non-zero element, then an entity pointer identifies one of the matching entities selected at random.

Then $\mathbf{z}_t, \mathbf{q}_t, \mathbf{x}_t$ are feed into a one layer feed forward neural network to form a single action vector representation:

$$o_t = tanh(W_{zo}z_t + W_{po}p_t + W_{xo}x_t) \tag{2.50}$$

where $\mathbf{q}_t$ is a concatenation of all belief vectors p_v^t.

The last step is to generate the output response sentence with a LSTM:

$$P(w_{j+1}^t | w_j^t, h_{j-1}^t, o_t) = LSTM_j(w_j^t, h_{j-1}^t, o_t) \tag{2.51}$$

In this thesis, we adopt this three-step approach. Compared with the trivial Seq2Seq model, the intermediate state tracker can better extract information from context and enables the model to query KB. We will add a memory mechanism into the model and see if it can further improve the performance on state tracking and generation tasks.

2.7 Summary

In this chapter, we introduced: 1. Basics of neural networks. 2. Word and sentence embeddings. 3. Dialogue systems and useful neural network structures for a dialogue system. These elements together can be constructed to

a pipeline process for NLP tasks: first, convert each word into word embeddings, then encode these word embeddings into sentence representations using skip-thought or BERT. Finally, feed the sentence representations into a predictor network, the type of predictor networks depends on the kind of tasks. E.g., for classification tasks, the predictor can be a simple feed-forward neural network with a softmax layer, but for generation tasks such as dialogue system, the predictor can be a Seq2Seq model with pointer network and copying mechanism.

The essence of this pipeline is to find a proper vector representation of the input text data. We want the vector to preserve as much information of the original input as possible. Current techniques might be able to deal with input text if it is a single sentence, but for tasks with multiple sentences as input, especially if the sentences have strong logical relations with each other such as dialogue systems, if we still treat them as a single long sentence and feed them in one go by concatenating every sentence, there are probably information lost which is not negligible. Therefore, in this thesis, we designed models with memory mechanisms for dialogue state tracking and dialogue generation tasks and evaluate our model with benchmark models without memory mechanisms.

Chapter 3 will introduce research on event detection tasks with Twitter data, and the methodology in this research will follow the NLP pipeline we just mentioned. The data of this chapter is a single tweet, one sentence at a time, so it fits well with the pipeline.

In chapters 4 and 5, we will introduce research on dialogue state tracking

and dialogue generation. Both tasks have an input of dialogue instead of a single sentence. We will present our novel models with memory mechanisms and corresponding experiments of the models.

Chapter 3

Crisis prediction based on Twitter data

In this chapter we will introduce experiments about sentence encoder. The goal of this research is to evaluate the possibility of establishing an NLP model that can predict critical events happening based on monitoring the stream of Twitter data. This research is part of the project "Continuous Planning of Operational Processes Applied to Non-combatant Evacuation Operations (Dstl, 2017-2018, PI)" sponsored by Defence Science and Technology Laboratory.

The aim of this project is to use a combination of run-time natural-language processing (to update a stochastic model of the target process based on unstructured data such as Twitter) and stochastic model synthesis (to generate Pareto-optimal plans for the process), to derive operational plans of human-centric critical processes such as and rescue, disaster relief operations, and emergency management.

The methodology in this research is the NLP pipeline we introduced in Chapter 2. First, we encode input Twitter data into vector representations

and then feed them into downstream predictors. The task is event detection, so the predictor we used is a neural network with a softmax layer at the end.

The result of this research has been published as part of the paper **1** in the publication page. My contribution in this research is that I developed a data pipeline to collect stream data, and labeled thousands of tweets with relevant events such as a gunshot, flood, earthquake, and so on. And I designed and trained a natural language processing model to update the stochastic model of a human-centric critical process by exploiting information encoded in unstructured data streams such as Twitter. Based on the predictions of this NLP component the planner is then making updates on the operational plans.

3.1 Continuous planning of operational processes involving humans (COPE)

The success of human-centric critical processes such as search and rescue, disaster relief operations, and emergency management relies on the dependable use of relevant information to support effective decision-making. A very challenging decision-making task in this context is the continuous planning needed because of the uncertainty and frequent changes in the environment and goals of operational processes[3].

When models are used to derive operational plans, it is paramount to ensure that these models are both reliable and up to date. Accordingly, the models are updated at runtime, with their parameters (and sometimes their structure) derived from observations of data sources that record model fea-

Event type	Event Location & Date	#Tweets†	Training	Testing
Gunshot	Las Vegas Oct 2017	16,000	Yes	-
Gunshot	Lower Manhattan Oct 2017	50,000	-	Yes
Earthquake	Oklahoma Oct 2017	17,000	Yes	-
Earthquake	California Oct 2017	14,000	-	Yes
Flood	Louisiana Aug 2016	18,000	Yes	-
Flood	Colorado Sep 2013	31,000	-	Yes

†Different numbers of tweets were posted at locations and times close to where and when each of the events occurred (including many tweets unrelated to the event).

Figure 3.1: Table of collected Twitter data.[45]

tures. The techniques used for this purpose typically require structured data sources (e.g., application logs or software-generated events). As such, these techniques cannot be applied to critical processes with human participants who generate relevant unstructured data through channels like social media, web-based forums, and verbal reporting of their observations.

3.2 Data collection

We have collected over 100,000 tweets that were posted at locations and times close to an incident that occurred. The incident types we are interested in in this research include Gunshot, Earthquake, and Flood, as shown in Figure 3.1. For each event type, we collected data of two independent events as training and testing data set, respectively.

Each data set was gathered using GPS location information and search radius around the event of interest. We selected 1000 tweets from each of the training sets by hand and labeled them as being related to the event or being normal tweets. This subset was then used as a training set for the classifier.

Data sets marked 'testing' are used as test sets and evaluations of the COPE simulator.

3.3 Model

The model to detect the occurrence of incidents from Twitter data consists of two components, a predictor that predicts if a single tweet is related to an incident or just a normal tweet, and a detector that keep detecting all tweets posted in small time windows, and trigger the alarm if the detector finds significant percentage of tweets in a certain period are related to one type of incident.

The predictor includes a sentence encoder that encodes each tweet into sentence representation and a softmax classifier that predicts if this tweet is about a certain event or just a normal tweet. We used GRU as the sentence encoder. Each tweet is firstly pre-processed such as cleaning the text and removing retweets. Then we convert each word w_i $i \in 1, ..., N$ in the sentence into glove embedding. After that, the embedding sequence is fed into a GRU, and the last hidden state is used as the sentence vector representation. The final layer is a feed-forward neural network ended with a softmax layer to make the prediction. In the mathematical form, the model is:

$$x_t = embedding(w_t) \tag{3.1}$$

x_t is the vector representation, w_t is the original input word.

$$s_{t+1} = GRU(s_t, x_t) \tag{3.2}$$

s_t stands for the hidden state of the GRU.

$$p = softmax(\sigma(W s_N + b)) \tag{3.3}$$

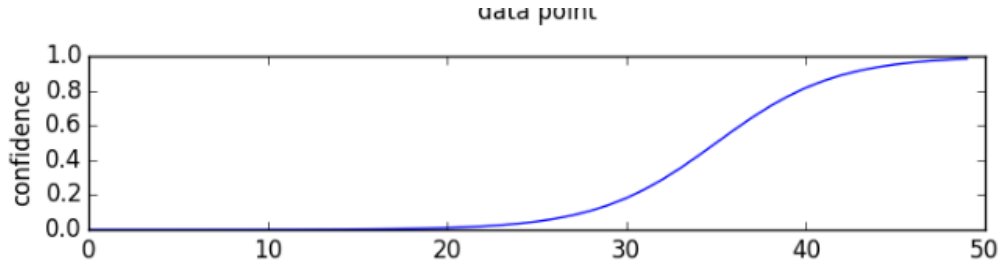

Figure 3.2: The graph of the confidence function.

p is the output probability, W and b are trainable parameters, and s_N is the last hidden state of the GRU. If the value of p for a tweet is larger than a threshold, then the model will predict that this tweet talks about an incident.

The model is now able to predict if a single tweet is indicating an incident or not, but as the Twitter data is very noisy, and the model is unable to make all prediction correctly, We introduced a confidence function to monitor all tweets in a time window, if only a sufficient amount of tweets are about the incident then our model raise the alarm, to prevent false positive. The confidence function is a logistic form function:

$$c = \frac{e^{r(n-t)}}{1 + e^{r(n-t)}} \tag{3.4}$$

where n is the percentage of tweets in the batch for which the probability of the event having occurred is over the threshold; t is a threshold above which we assume an event to have occurred; and r is a scaling factor allowing for the gradient of the confidence function to be controlled.

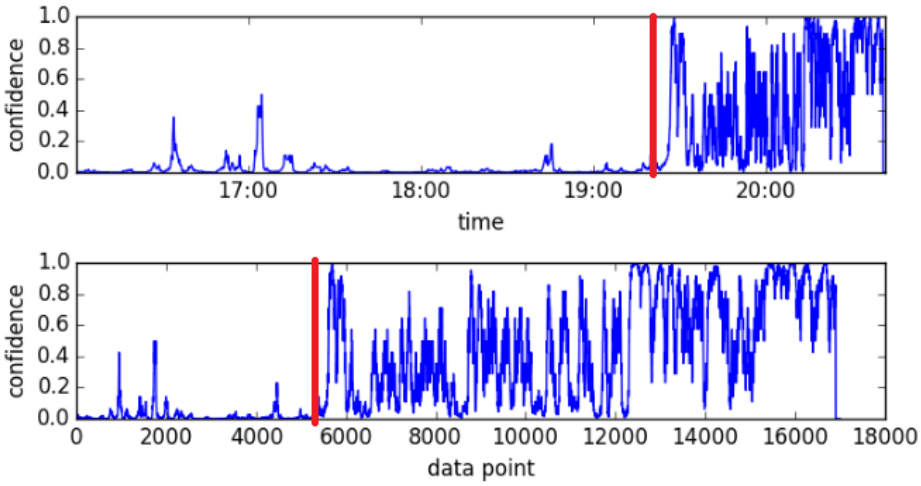

Figure 3.3: Apply confidence function and predictor on the test set of Gunshot(Lower Manhattan data set). The graph on top is the corresponding confidence level by time, and the bottom graph is the confidence level by data point. The vertical red line indicates the time that the gunshot really happens, according to the news report.

3.4 Experiment

We trained our model for each of the events independently. The hyperparameter setting is word embedding and hidden state dimension: 300, learning rate 0.001. As mentioned before, we used one event as the training set and another same type of event as the test set, as example, for gunshot event, we used 1000 labelled tweets from Las Vegas gunshot as the training data set,

Table 3.1: Accuracy of event predictions for single tweets, with three different sentence encoders.

	Gunshot	Earthquake	Flood
GRU	0.927	0.883	0.855
LSTM	0.925	0.891	0.847
Conv1d	0.894	0.886	0.821

and 1000 tweets from Lower Manhattan as the test set. This is to make sure our model does not have any information of test set during training process and so we could best test our model's performance for completely new events.

Table 3.1 shows the result we get for each type of the events, with three different sentence encoders. We used three different types of sentence encoder: GRU, LSTM and Conv1d. The first two have been introduced in Chapter 2, and the last Conv1d stands for one dimensional convolutional neural network[26], we choose it so that we could compare the performance between RNN structure and CNN structure for NLP task. Usually convolutional neural network(CNN) is used for machine vision tasks, and is usually two-dimensional as image data is two-dimensional, Conv1d used a one dimensional kernel in order to process text data which is one-dimensional. As a result, GRU encoder performs the best for gunshot and flood event and LSTM for earthquake. 1d Convolutional neural network has generally worse performance compared with RNN structured network.

The next step is to apply the confidence function and the predictor on the whole original test data set of Gunshot(Lower Manhattan data set). Figure 3.3 shows the confidence level our model has over time and data point. The vertical red line is the time point where the gunshot really happens, according to the new report. The figure shows clearly that the confidence of our model is significantly higher than usual after the gunshot happens. The graph at the bottom is based on data points, we can see that the red line indicating theafter the gunshot, people tweet much more than at the normal time.

3.5 Summary

In this chapter, we present our research on Twitter data based event detection task. The methodology we used is the NLP pipeline introduced in the previous chapter, which is the word and sentence embedding plus downstream predictor. Our contribution in this research is that we created a novel event detection data set based on Twitter data[c0], and the model we proposed on the data set has shown its capability to extract useful information from the Twitter data stream.

The result of the experiment indicates that, in the case of single sentence input, the encoded vector representation of each tweet preserved enough information so that it can be used to predict the occurrence of events by following predictors. This shows that the NLP pipeline we introduced is working, at least for single sentence input, and it is a possible direction of solving more challenging NLP tasks. In the next two chapters, we will present our research on dialogue system tasks, where the input will be multiple sentences.Crisis prediction is a typical classification tasks, the predictor choose from a binary target space (related, not related). Although dialogue system looks much more complicated than crisis prediction, it is intrinsically just a complex version of classification task: for dialogue state tracker, the task is to choose from all possible dialogue state; for dialogue generator, at each time step, the task is to choose the right word from the vocabulary set.

[c0]This project is sponsored by Defence Science and Technology Laboratory, due to copyright issue, this data set is not publicly available.

Chapter 4

Dialogue state tracker

Task-oriented dialogue systems, such as Apple Siri or Amazon Alexa, address one of the major tasks in the NLP field. While the complete accomplishment of a dialogue system may still have a long way to go, a step-by-step approach that includes Dialogue Representation, State Tracking, and Text Generation has been proposed. The main component of the middle step is State Tracking, which predicts at each turn of the dialogue what the topics or user requirements are. The state is represented as values in certain predefined slots. For example, given one of the sentences of a dialogue: *Is there any restaurant in the city center?* then the corresponding state values could be *[Task: Restaurant][Location: Center]*.

Various task-oriented dialogue data sets have been released, including NegoChat [52] and Car assistance [18]. Eric et al. introduced the MultiWOZ 2.1 data set [17], which is a multi-domain data set in which the conversation domain may switch over time. The user may start a conversation by asking to reserve a restaurant table, then go on to request a taxi ride to that restaurant. In this case, the state tracker has to determine the corresponding domain, slots and values at each turn of the dialogue, taking into account the history

of the conversation if necessary.

In this chapter, we introduce a memory-based dialogue state tracker, which consists of two components: a memory encoder which encodes the dialogue history into a memory vector, and a pointer network which points to the set of possible values of states, including words in the dialogue history and ontology of the data set. The memory vector will be updated at each turn of the dialogue, and it will then be passed to the pointer network, where prediction is made based on the current memory. For each dialogue, there are multiple slots to be filled, each with a set of possible values. Both the slots and their possible values are predefined. The prediction is a two-step procedure. Firstly, predict the domains the current utterance lies in, then for the specific domain, fill values into corresponding slots.

This work has been accepted as a conference paper **2** in the publication page.

4.1 Introduction to dialogue state tracker

Dialogue state tracking (DST), or Belief tracking, was introduced as an intermediate step in dialogue systems. In this step, the task is to recognize the user's goal as *state*, which will then be used to guide the system response correspondingly[6]. Table 2.1 shows an example of a dialogue with the corresponding dialogue states at each turn of the dialogue, which will be updated at every user utterance.

DST systems can be classified into two types: ontology-based and ontology-free. Ontology-based DST [49, 71] requires a set of pre-defined possible values

for the dialogue state, and the model selects the most likely one from the ontology set. The Ontology-free DST [69, 19], on the other hand, does not require such an ontology set, and will choose the most likely phrase from the dialogue history and vocabulary set.

The ontology-based DST will require a manually defined set of values for all slots, which will be expensive for large-scale dialogue systems and makes it difficult to generalize the model. On the other hand, the ontology-free DST is easy to generalize for large-scale dialogue systems but can not utilize expert knowledge in the model.

The two types of DST can be incorporated by using a gate function that determines if the current slot should be filled by choosing from an ontology or from the dialogue history [47].

4.1.1 Multi domain Dialogue state tracking

The MultiWOZ 2.1 data set released by [17] is one of the largest task-oriented dialogue system data sets so far. It includes nine different domains, and at each turn of the dialogue, there can be more than one domain active. As the dialogue continues, the active domain and corresponding slot may be changed.

Benchmark methods of dialogue state tracking on this data set include FJST, HJST, DST, HyST and TRADE[17]. FJST, or Flat Joint State Tracker, refers to the trivial approach which uses bi-directional LSTM to encode the full dialogue history and then feed into separate feed-forward neural

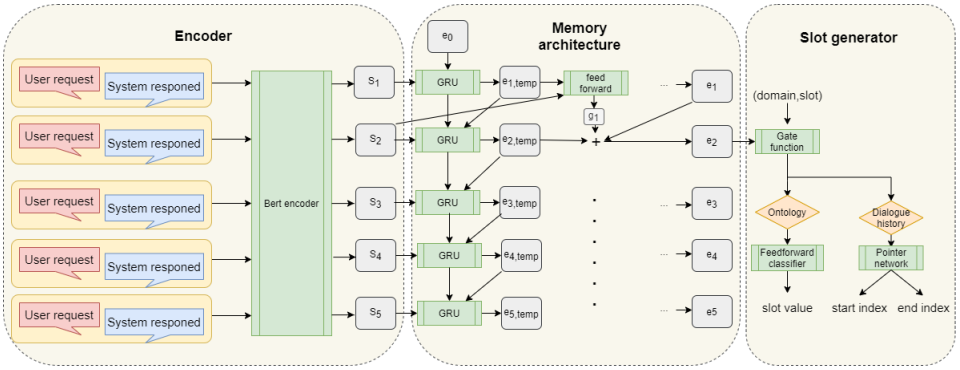

Figure 4.1: Memory state tracking system. This graph shows how does the memory mechanism work on the second turn of the dialogue

network independently trained for each state slot. HJST refers to Hierarchical Joint State Tracker, which is similar with FJST but instead enodes the dialogue history in a hierarchical structure. TRADE[68] uses bi-directional LSTM as encoder, and uses generative state tracker with a copy mechanism. DST[19] model frames the dialogue state tracking task as a reading comprehension task and extract values of each slot from the dialogue history, this utilizes the idea of pointer network. HyST[20]aims to extend the generality of the model by adding an external vocabulary set as possible slot value.

While those benchmark methods varies mainly on the predictor part, our model uses a novel memory architecture as the dialogue history encoder, and used a combination of copy mechanism and pointer network as the predictor with a gate function decides which network is used to generate value of slot.

4.2 Memory state tracker model

In this chapter, we provide a detailed description of the proposed Memory State Tracking model, as shown in figure 4.1.

Each sentence in the input dialogue is first encoded to vector representation by the sentence encoder. Then each sentence vector is fed into an RNN structured Memory network in turn, and the memory network will output a memory vector at each turn of the dialogue. The final step is to make predictions based on the memory vector, which is a three-step procedure for each state slot to be filled: firstly, through a binary 'mention' gate to determine if the current state is mentioned or not in the dialogue. If the gate predicts it is not mentioned, then the state slot will be marked as "not mentioned". If it is mentioned, two predictors will make independent predictions of possible slot values in the ontology and dialogue history, respectively, and another gate function will be used to decide which prediction will be used as the final predicted value of the slot.

4.2.1 Encoder

We used two types of sentence encoding models as our model's encoder. The first one used Glove word embedding[46] and feed each word into a GRU, and used the last hidden state as the sentence representation. The second one used the BERT model for language understanding [16] as the sentence encoder. For each turn of the dialogue, the input sequence is the current user request and system response separated by [SEP] token, and then padded to a fixed length and fed into the encoder, the output s_i is the vector representation of sentence i.

4.2.2 Memory network

For each dialogue, a memory vector e_i is used to represent all information from the dialogue history. At each turn of the dialogue, the memory vector is updated using an RNN, where the input is the sentence representation s_i of the utterance at current turn. At the beginning of the training process, a hidden parameter H_0 is initialized with all values set to zero, then for each turn of the dialogue, the memory is updated with

$$e_{temp},\ H_i = GRU(s_i, H_{i-1}) \tag{4.1}$$

$$e_i = g_i \cdot e_{temp} + (1 - g_i) \cdot e_{i-1} \tag{4.2}$$

where the e_i is the ith memory representation of the dialogue, representing the whole dialogue until the ith turn.

g_i is an attention score computed by a simple gate function with two layer feed-forward neural networks

$$g_i = sigmoid(W_2 \cdot tanh(W_1 \cdot (s_i,\ e_{i-1})) \tag{4.3}$$

The purpose of this gate function is to measure how important the current turn of the dialogue is. If the gate function gives a high score, it means the current turn is informative, and by equation 4.2 it will update the memory vector greatly, and vice versa.

4.2.3 Pointer predictor

We adopt the Pointer network architecture [36] as the predictor in our model. The prediction model makes predictions based only on the memory vector e_i

at each turn. The prediction is a two-step procedure. First, we have a set of mention gate functions for each of the slots to be filled:

$$m_{i,j} = G_{j \in J}(e_i) \tag{4.4}$$

which decide if the jth (domain, slot) pair is mentioned or not at ith. J is the set of all (domain, slot) pairs to be filled, $m_{i,j}$ is the probability of the mention gate, and $G_{j \in J}$ is a fully connected layer with sigmoid activation.

$$G_{j \in J}(e_i) = sigmoid(W_j e_i + b_j) \tag{4.5}$$

Different from the three-way gate in TRADE [68], we do not add "don't care" in this gate, as the number of "don't care" slots is relatively small compared with "not mentioned", and it can be predicted later in the categorical predictor.

If the mention gate predicts the jth (domain, slot) pair is not mentioned, then the value of this slot will be filled with "not mentioned". If the gate predicts that it is mentioned, the value will be filled by the pointer network.

For (domain, slot) pairs that are predicted to be "mentioned", there are two independent predictors to predict its values. One is a pointer network point to words in the dialogue history.

$$Index_{start,j} = argmax_k(sigmoid(W_{j,1} w_k + b_{j,1})) \tag{4.6}$$

$$Index_{end,j} = argmax_k(sigmoid(W_{j,2} w_k + b_{j,2})) \tag{4.7}$$

where w_k is the word embedding of the kth word in the dialogue history. $Index_{start,j}$ represents the start index of predicted slot values and $Index_{end,j}$ represents the end index, with words in between being the final prediction of pointer network.

Another predictor is a categorical classifier that chooses a word from possible values for current (domain, slot) pairs, with a feed-forward neural network.

As an example, given a dialogue *Is there any hotel in the city centre with low-price?*, slot to be filled is hotel-price and slot ontology is(*cheap, medium, expensive*), the pointer predictor will pick the most likely phrase from the dialogue history and output *low-price*, whereas the categorical classifier will choose *cheap* from the ontology.

A gate function is employed to determine which predictor is used, with a structure similar to the mention gate.

$$G_{pred}(e_i) = sigmoid(W_j e_i + b_j) \tag{4.8}$$

4.3 Experiment

This chapter discusses the experiment of our model trained on the MultiWOZ 2.0 [8] and MultiWOZ2.1 [17] data sets, including experiment setups, parameters and results.

4.3.1 Data set

MultiWOZ 2.0 and 2.1 are large goal oriented dialogue system data sets, which consist of around 10,000 dialogues, 113, 556 turns of dialogues with multiple domains including *restaurant*, *hotel*, *hospital*, *taxi*, *police train*, *attraction*, and *bus*. Table 2.1 is an example of a dialogue about restaurant booking.

MultiWOZ 2.1 is a refined version of MultiWOZ 2.0, modifying around 2% of the slots.

4.3.2 Experiment setup

The hospital and police domains are ignored in this thesis as they contain only very few dialogues, following [68]. We use the validation and test set supplied by the data set.

4.3.3 Model details and parameters

Our model is trained on a single GTX 2080 GPU with Pytorch environment.

Word embedding dimension is set to 300 using Glove embedding [46]. The maximum sentence length is set to 30, in order to train the model in batches. This means words in an input sentence after the 30th word will be discarded, if the sentence is longer than 30, and if shorter, a special token of [**PAD**] will be added until the length of input sentence is 30. Similarly, the max dialogue length is set to be 10.

The training batch size is 128, hidden size of the encoder, and all feed-forward neural networks is 256. We also used Gradient Clipping [25] with

clip parameter 50, to prevent gradient explosion.

We used different learning rates for encoder and pointer networks, which shows to have better performance. The encoder learning rate is 0.001 and predictor learning rate is 0.0001.

4.4 Experimental Results

Table 4.1: Joint goal accuracy on MultiWOZ 2.0 and MultiWOZ 2.1 data set

	MWOZ 2.0	MWOZ 2.1
HJST[17]	38.4	35.55
FJST[17]	40.2	38.0
TRADE[68]	48.6	45.6
DST[19]	39.41	36.4
HyST[20]	42.33	38.10
MST(ours)	**49.16**	**47.27**

We used joint goal accuracy to test our model. The slot accuracy is the accuracy of each single state value. For the joint accuracy, only if all the slots in one turn are correctly predicted will the prediction be marked as correct, otherwise, it is incorrect.

Table 4.1 shows the joint goal accuracy of our model on MultiWOZ2.0 and MultiWOZ2.1 data set, compared with benchmark models. Our model beats these baseline models in both data sets, compared with state-of-the-art TRADE, our model outperforms it by 0.56% on MultiWOZ 2.0 and 1.67% on MultiWOZ 2.1 dataset.

Table 4.2 shows the accuracy of each domain for MultiWOZ 2.1 data set. The joint accuracy indicates how much our model got all correct for all slot

at one turn, and the slot accuracy indicates how much our model got right for a single slot, so the joint accuracy is generally much lower. Particularly, the joint accuracy for hotel and taxi domain is relatively lower than others, according to Table 6.1 in appendix, this could be because the possible value of taxi is much more than others, and for hotel domain it has more slots to be filled so it is more difficult to predict them all correct.

Table 4.2: Domain-Specific Accuracy on MultiWOZ 2.1 data set

Domain	Joint Accuracy	Slot Accuracy
Restaurant	66.41	98.22
Hotel	48.13	97.14
Taxi	39.50	94.85
Attraction	66.47	98.43
Train	63.83	94.98

4.5 Ablation Test

Table 4.3 shows the ablation test on the MultiWOZ 2.1 data set. We tested the performance with different encoders, and with and without the memory mechanism. The test shows that the choice of the encoder does not make much difference, Bert encoder very slightly outperforms the GRU encoder. However, the memory mechanism improves the performance for both encoder by more than 0.6%, indicates the memory mechanism did extract more information from dialogue history.

Table 4.3: Ablation test on MultiWOZ 2.1 data set.

Feature	Joint accuracy
GRU encoder(no memory)	46.55
Bert encoder(no memory)	46.62
GRU+Memory mechanism	47.19
Bert+Memory mechanism	47.27

4.6 Summary and Discussion

In this chapter, we introduced a novel memory mechanism for a dialogue state tracking system. The core contribution of our work is to incorporate the memory architecture into the dialogue state tracking system. We used a vector that will be updated at each turn of the dialogue to preserve useful historical information in the model. This model outperforms a set of benchmark models with joint goal accuracy on both MultiWOZ 2.0 and MultiWOZ 2.1 data sets.

Compared with existing works such as TRADE[68], our memory-based model enables the network to have sentence-level links between sentence embeddings and update the memory vector correspondingly. This gives our model at least the possibility of learning sentence-level logic information. The model performance is improved with memory setting, indicating that the memory vector could preserve more information than no memory setting.

In the domain-specific accuracy table, we can see that the Hotel and Taxi domains are shown to be more difficult compared with other domains. The Hotel domain has 11 slots to be filled which is the largest of all domains, so it is reasonable that the Hotel domain has a lower joint goal accuracy. For the Taxi domain, as shown in the Appendix, the number of possible values for its state slot is the highest among all domains, which may lead to the low joint goal accuracy.

Chapter 5

Sentence generator for dialogue systems

In this chapter, we will introduce another important element of the dialogue system: the sentence generator. We already have embeddings to convert user utterance to vector representation and a state tracker to predict the user's goal. The next step is using the sentence generator to generate human language response based on the vector and states. We will introduce experiments that we have implemented related to sentence generator. We will start with the language model, and discuss different approaches of sentence generator and introduce our model. Then we will discuss the evaluation of the sentence generator and the experiment result.

5.1 Sentence generator

A sentence generator is a statistic model that, given a start of sentence token [SOS], it gives the next word of the sentence by giving a probability distribution over the vocabulary set V, and making this prediction repeatedly until the predicted word is the end of sentence token [EOS]. The essence of this model is a language model.

5.1.1 Language model

Given a sequence of words $s = w_1, w_2, ..., w_n$, a language model is a probability distribution that determines how likely this sequence is a proper sentence in a language. For example, "*I like playing football.*" should have a higher probability compared with "*I football playing like.*".

Formally, a language model is defined by:

$$P(w_1, w_2, ..., w_n) = \prod_{i \in n} P(w_i | w_1, ..., w_{i-1}) \tag{5.1}$$

where $P(w_i|w_1, ..., w_{i-1})$ is the probability that the upcoming word is w_i given the previous word being $w_1, ..., w_{i-1}$).

In practice, the language model is usually simplified to the uni-gram or n-gram model, which only considers one or n previous words. But to build a sentence generator, what we are interested in here is not the probability of the whole sentence but the single conditioned probability of the upcoming word. If we have this probability model, we can generate a sentence by repeatedly picking the word with the highest probability. In language model, given a large corpus, this uni-gram probability is computed by:

$$P(w_i|w_{i-1}) = \frac{Count(w_{i-1}, w_i)}{Count(w_i)} \tag{5.2}$$

where $P(w_i|w_{i-1})$ is the conditional probability that current word is w_i given the previous word isw_{i-1}, $Count(w_{i-1}, w_i)$ is the total amount that current to have phrase (w_{i-1}, w_i) in the corpus, and $Count(w_i)$ is total amount of word w_i in the corpus.

This probabilistic model can be regarded as a simple sentence generator. However, this generator is far from functional. It will generate the same sentence with the same start, and it does not have the ability to consider the context of the dialogue. If we have a knowledge base, this generator is unable to connect with it.

To solve these problems, we need to update this language model. Bengio[5] first introduced a neural network based language model, it firstly convert each previous word into word embeddings, then used a feed-forward neural network to learn the probability of the upcoming word. this model has greater generalization ability compared with the trivial language model, but still suffer from lacking the ability to handle long term information in a long sequence.

Mikolov[38] introduced a recurrent neural network-based language model, the word embeddings are fed into an RNN, and the output of this RNN at each time step is then fed into a softmax function to generate the probability distribution. This network has a better ability to deal with long-term information.Most commonly used RNN includes LSTM, Bi-LSTM and GRU, generally speaking, Bi-LSTM has the greatest amount of trainable parameters, so Bi-LSTM will require more computational resources compared with other two, and performs slightly better[14]. Based on this RNN model, we can now construct a simple sentence generator.

5.1.2 RNN generator and encoder-decoder model

As shown in figure 5.1, an RNN sentence generator is very similar to a normal RNN model. The difference is that:

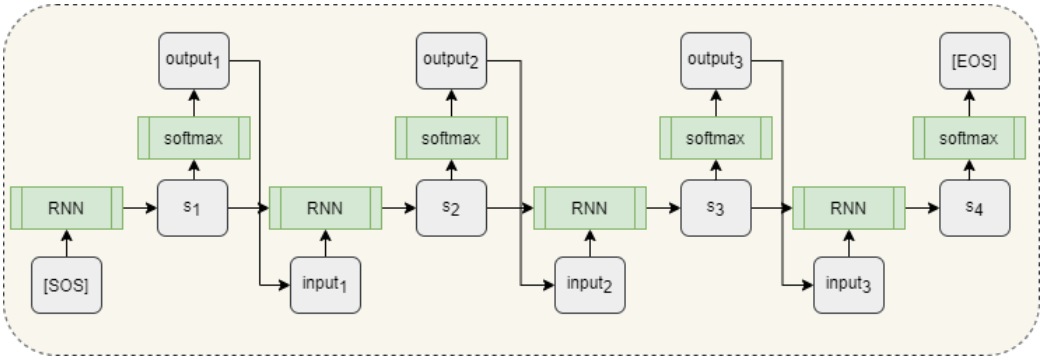

Figure 5.1: Trivial sentence generator.

1. The output of the RNN generator at each time step is connected with a softmax over the vocabulary set, which predicts the next word of the current sequence.

$$y_t = softmax(o_t) \tag{5.3}$$

where y_t is the generated word, o_t is the RNN output with dimension equals to size of vocabulary, represents confidence of the model to each word in the vocabulary.

2. at each time step of the RNN generator, the input is previously generated word.

$$s_t = \sigma(Uy_{t-1} + Vs_{t-1}) \tag{5.4}$$

where s_t is the RNN hidden state at time t, y_{t-1} is the previously generated word, s_{t-1} is the previous hidden state and U and V are model parameters.

Here the first input will be the start of the sentence token [SOS]. We could use any type of RNN such as LSTM or GRU, as long as at each time

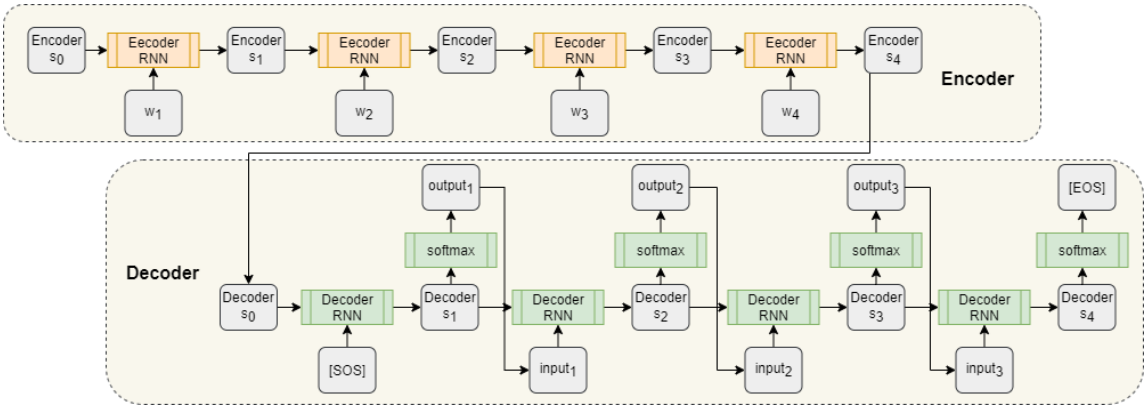

Figure 5.2: Seq2Seq sentence generator.

step we feed in the previous prediction and predict the upcoming word at the output.

Now we have a working sentence generator, but as mentioned, it still cannot cope with the context of a dialogue. Users may ask "*what is the weather like today?*", and there is no way to inform this generator about this query. To deal with this, we adopted the Seq2Seq model or encoder-decoder model, as shown in figure 5.2, which was invented for machine translation task[58].

The idea of Seq2Seq model is simple. For the sentence in the previous dialogue, we simply add an RNN to learn it and use the last hidden state of this RNN as the initial hidden state of the generator RNN. In this way, the generator will have access to the dialogue history.

5.1.3 Attention based decoder

The standard Seq2Seq model generally performs not well for long sequences, as it only uses the last hidden state as the context vector, other than that

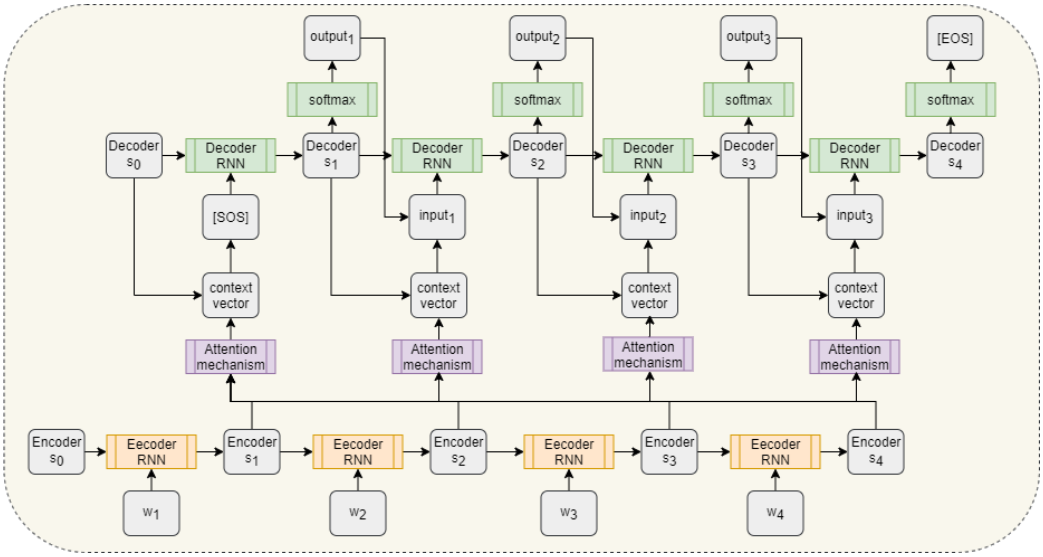

Figure 5.3: attention based sentence generator.

it does not have any other connection with the context. This means all information that is lost in the context vector during the generation process is permanently lost.

To deal with this, Dzmitry Bahdanau et al. [4] and Luong et al. [34] introduced attention mechanism into the Seq2Seq model, respectively. The idea of their work is very similar, as shown in figure 5.3, at the encoder step, the output of the encoder RNN at each time step h_j is used as the vector representation of the word w_j. At each step of the decoder, when the model is trying to output the next predicted word, the input of the decoder RNN is not only the previously generated word embedding but also a weighted context vector computed by the weighted sum of the word vector representation in the encoder part. The weight of each word is the similarity score a_{ij} between the current decoder hidden state and each word vector representation. The final

input is the concatenation of previous word embedding and weighted context vector, and the similarity score here works as the attention mechanism, if a word in the context get higher similarity score with the decoder hidden state than other words, it indicates that this word is more related to current state of generator and we shoud put more attention on this word, so we give it higher weight when computing the context vector. In this way, the decoder could trace back to the context at every step of output and could know which part of the context is important for the current step by the attention mechanism.

Formally, for Bahdanau's attention, the context vector at each step of decoder is:

$$c_i = \sum_{j=1}^{n} \alpha_{ij} h_j \tag{5.5}$$

where

$$\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k=1}^{n} exp(e_{ik})} \tag{5.6}$$

and

$$e_{ij} = a(s_{i-1}, h_j) \tag{5.7}$$

s_{i-1} is the decoder RNN hidden state at current step and h_j is the jth word vector representation.

The Luong's attention is developed based on Bahdanau's attention. The difference is that it uses different similarity metrics, computes the current decoder hidden state first, then uses it and the context vector to compute the output of the current step. In practice, the two approaches have very similar performance.

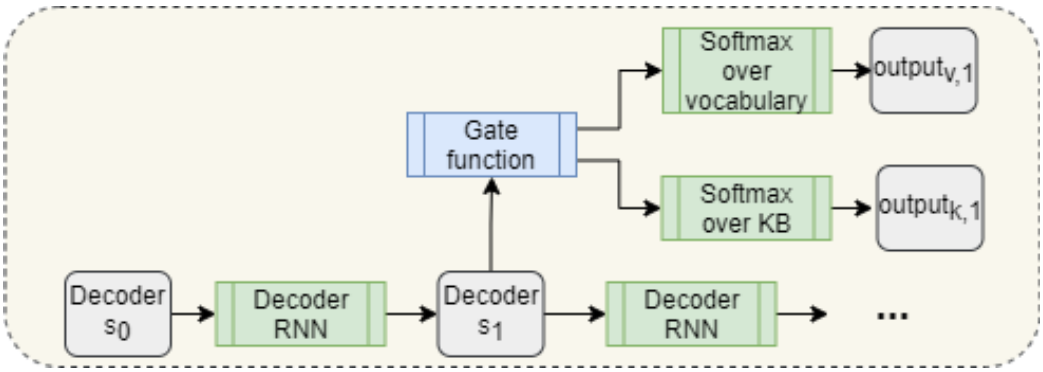

Figure 5.4: Knowledge enhanced sentence generator.

5.2 Knowledge enhanced generator

We now have a sentence generator which is able to generate sentence based on the context. This model works well for machine translation tasks, but it is still not good enough for the dialogue system. For task-oriented dialogue systems, there will usually be a knowledge base containing important information about the dialogue. The current generator does not have any access to the knowledge base.

For example, in MultiWOZ data set, there is a knowledge base about detailed information of many restaurants, hotels, attractions, and so on, including their name, address, area, etc. This knowledge base is important for a task-oriented dialogue system intended to respond to customers' requests and queries.

Here are some sample of the knowledge bases of MultiWOZ data set:

```
{
    "address": "Regent Street City Centre",
    "area": "centre",
    "food": "italian",
```

```
    "id": "19210",
    "introduction": "Pizza hut is a large chain with restaurants
    nationwide offering convenience pizzas pasta and salads to eat
    in or take away",
    "location": [
        52.20103,
        0.126023
    ],
    "name": "pizza hut city centre",
    "phone": "01223323737",
    "postcode": "cb21ab",
    "pricerange": "cheap",
    "type": "restaurant"
}
```


Not connecting with this knowledge base will undoubtedly make our model perform worse. As the generator now only makes predictions of the upcoming word based on the vocabulary set of context, there will be many *out of vocabulary* (OOV) problems. For example, in the training data set, the ground truth agent response might be *the number is 01223337766* ., but the telephone number *01223337766* never appears in the training data but the knowledge base, then it is not possible for the generator to make a correct prediction in this case.

An intuitive approach to solve this problem is adding the vocabulary of knowledge base V_k into context vocabulary V. This solves the OOV problem but plenty of structured information is lost. As shown in the example, the element in knowledge base is in form of slot-value pairs, such as *"area": "centre", "food": "italian"*. We are mainly interested in the values, but the slot name also contains plenty of semantic information.

Key-value retrieval network[18] was introduced to utilize the structured data in the knowledge base. At each time step in the decoder, a gate function

g firstly decided to select the upcoming word in context vocabulary V or in knowledge base vocabulary V_k. If the gate function chooses to generate from V_k, then each slot-value pairs will be given a score based on the slot name or key. And it will output the value of the slot-value pairs with the highest score.

As an instance, if the decoder has generated "The phone number is :", ideally the gate function will choose to generate from knowledge base and "phone" will have higher score compared with other slot name "area" or "address", and then the final output is the value of "phone", say 01223323737.

This key-value retrieval model can preserve part of the structured information of the knowledge base, but it still has limitations for a larger and more complex knowledge base. For the knowledge base of MultiWOZ data set, its first level structure is a set of restaurants, hotels, and so on. Each restaurant or hotel then has a set of attributes; therefore, if we convert the whole knowledge base into slot-value pairs, there will be many phone number pairs, then it is impossible for the network to decide which one should be selected. Also, when using Seq2Seq models for a dialogue system, usually the dialogue history is concatenated as one long sequence and fed into the encoder, but they are separate utterances in turn, this structured information may be lost as well.

5.3 Our model: Memory based sentence generator

In order to better utilize information from the knowledge base and dialogue history, we proposed a memory-based sentence generator. As shown in figure

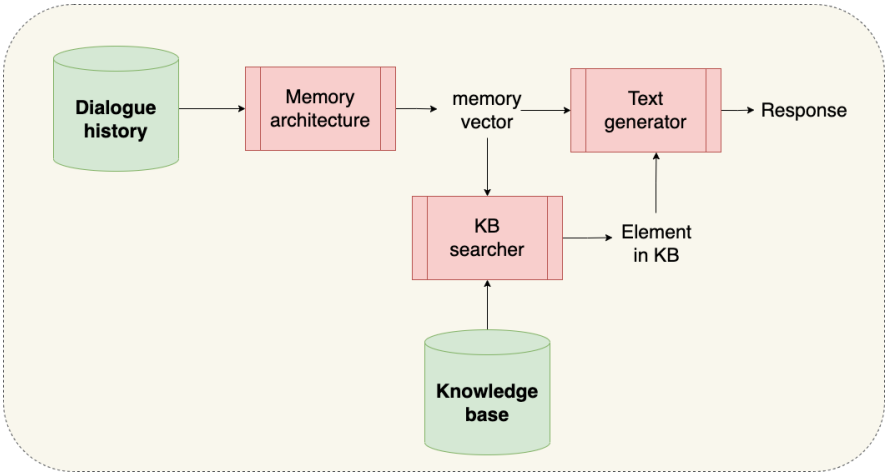

Figure 5.5: A high level architecture of the our model.

5.5 it mainly consists of three parts: the memory architecture, the knowledge base searcher, and the text generator. All words are firstly converted into Glove embedding as input of the model. The memory architecture is to compute a memory vector based on dialogue history and current utterance. This memory vector is then used as input of decoder RNN and generator. The knowledge base searcher is to find elements in the knowledge base that match the dialogue state generated by state tracker, and then used them as the potential candidates of upcoming words in the generator. The generator then predicts the upcoming word based on memory vectors.

Memory architecture

As shown in figure 5.6, in order to better utilize information from the knowledge base, the essence of the memory architecture of our model is the short term memory and long term memory vector. The short-term memory is computed by the attention mechanism over the current utterance and

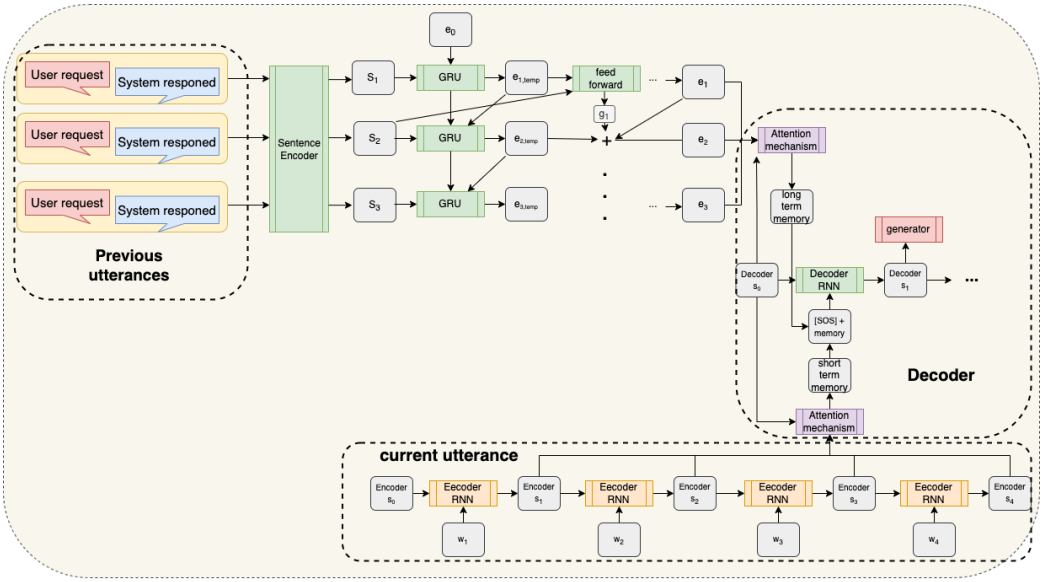

Figure 5.6: Memory architecture of our model.

words that have been generated at this time step, and the long-term memory vector is computed over the dialogue history. Then the long-term memory vector and short term memory vector are concatenated with the previously generated word as the input of decoder RNN, and the output of the decoder RNN, together with long and short memory vector, are fed into the text generator model.

In detail, to compute the long term memory, given a dialogue history $u_1, ..., u_k$, we first compute their sentence representations $s_1, ..., s_k$ by a sentence encoder, then compute memory vector $e_1, ..., e_k$ for each utterance by an RNN, just like what we did in the memory state tracker model in chapter 4, with equation 4.1-4.3. Now we have the memory vector $e_1, ..., e_k$ for each utterance in the dialogue history. The next step is to introduce an attention mechanism into the model to compute the long term memory vector ltm_i at

the current time step of the decoder:

$$ltm_i = \sum_{j=1}^{n} \alpha_{ij} e_j \tag{5.8}$$

where

$$\alpha_{ij} = softmax(score_{ij}) = \frac{exp(score_{ij})}{\sum_{k=1}^{n} exp(score_{ik})} \tag{5.9}$$

here $score_{ij}$ is the similarity score between the current decoder hidden state (at time step i) and memory vector of jth utterance:

$$score_{ij} = a(s_{i-1}, e_j) \tag{5.10}$$

s_{i-1} is the decoder RNN hidden state at current step and e_j is the jth memory vector representation. a can be any similarity metric, here we used dot product.

Similarly, the short term memory vector stm is computed by:

$$stm_i = \sum_{j=1}^{n} \alpha_{ij} h_j \tag{5.11}$$

where

$$\alpha_{ij} = softmax(score_{ij}) = \frac{exp(score_{ij})}{\sum_{k=1}^{n} exp(score_{ik})} \tag{5.12}$$

here $score_{ij}$ is the similarity score between the current decoder hidden state (at time step i) and jth word vector representation computed by encoder:

$$score_{ij} = a(s_{i-1}, h_j) \tag{5.13}$$

s_{i-1} is the decoder RNN hidden state at current step and h_j is the jth word vector representation.

Intuitively speaking, the short term memory is a weighted sum of all word embeddings of current sentence, where the weight (or attention) is the similarity score between each wording embedding and current decoder hidden state, and long term memory is weighted sum of sentence embeddings.

The long term memory ltm_i and short term memory stm_i are then concatenated with the previous predicted word (initially [SOS]) as the input of the decoder at the current time step.

Knowledge base searcher

The purpose of the knowledge base searcher is to find desired elements in the knowledge base, such as a restaurant or hotel, that matches the dialogue state we predicted with our state tracker. The search result can be a set of many elements, or empty, which means no elements in the knowledge base matches the current dialogue state.

The amount of search results is also important information for the generator. In the data set, we can often see agent replies such as "*There are several, do you have a specific area you are interested in?*" or "*There are no restaurants that fit your desired criteria. Is there another cuisine type or price range?*" Therefore, at each turn of the dialogue, we have a search result state vector to inform our model what we got from the knowledge base. The first three positions of the vector is a one-hot indicator, (1,0,0) represent there are multiple candidates in the search result, (0,1,0) stands for a single candidate, and (0,0,1) stands for no candidate. The fourth position is the exact amount of candidates, this vector is concatenated with the short-term memory when training.

Text generator

The main structure of our text generator is shown in figure 5.4, the generator takes as input the long and short memory vector, the search result state vector, and the decoder hidden state at the current step. Then we used a feed-forward neural network as the gate function, and two independent feed-forward neural network as predictors for vocabulary and knowledge base:

$$g_i = sigmoid(W_2 \cdot tanh(W_1 \cdot (stm_i,\ ltm_i,\ s_i)) \tag{5.14}$$

$$predictor_{V,i} = softmax_V(W_2 \cdot tanh(W_1 \cdot (stm_i,\ ltm_i,\ s_i)) \tag{5.15}$$

$$predictor_{kb,i} = softmax_{kb}(W_2 \cdot tanh(W_1 \cdot (stm_i,\ ltm_i,\ s_i)) \tag{5.16}$$

5.3.1 Objective function

The most intuitive objective function for text generation is the maximum likelihood estimator(MLE) loss:

$$\widehat{y} = \underset{y}{argmax}(log(P(y|X))) \tag{5.17}$$

X is the previous words. The idea is to maximize the probability of the target word given by our model. The MLE loss is the most commonly used loss function for many classification tasks because it is intuitively trivial, But for Language model research has shown that model trained with simple MLE loss tends to output the most common expressions, and lacks diversity.[31] To deal with this problem, instead of using MLE, here we adopted the maximum mutual information(MMI) as objective function[31].

$$\widehat{y} = \underset{y}{argmax}(log(P(y|X)) - \lambda log(U(y)) + \gamma_{length} N_y) \tag{5.18}$$

where

$$U(y) = \prod_{i=1}^{N} P(y_i|y_1, y_2, ..., y_{i-1}) \cdot g(i) \tag{5.19}$$

and

$$g(i) = \begin{cases} 1 \; if \; i \leqslant \gamma \\ 0 \; if \; i > \gamma \end{cases} \tag{5.20}$$

In formula 5.21, the $log(P(y|X))$ is the conditioned language model term, $\lambda log(U(y))$ is the anti language model(LM) term that punishes popular expressions controlled by hyper-parameter λ, $\gamma_{length} N_y$ is the term to encourage long sentence. The idea is to discourage the model from having common expressions by the anti-LM term, but if we use the original language model term $log(P(y))$, it will cause the model to produce sentences not follow the language model. Therefore we used a parameter λ to control its weight in the objective function and use modified $log(U(y))$ as the language model to make this term only works at first γ_{length} words in the generated sentence. In our experiment, the λ, γ_{length} and γ is set to 0.3, 5 and 0.15 respectively. For example, phrase *Apple is a healthy* will very likely followed by *fruit*, and therefore the language model will give very high output probability to *fruit*. However, it is also possible that the following word is actually *company*, if we only have a language model term, then our model will rarely or even never output *company*. With the anti-LM term, our model gives a penalty to very common expressions and so the confidence to *fruit* will be relatively lower and therefore the diversity of output will increase.

5.3.2 Teacher forcing

One of the challenging problems for Seq2Seq text generation is the so-called exposure bias. As each word generated by the model is based on previously generated words, if there are mistakes or biases at the first few steps of generation, then it will be largely amplified with the process of the generation. This problem is related deeply to the structure of the Seq2Seq generation approach, so it is difficult to cope with.

One of the effects of exposure bias is it makes the training process hard to converge. At the beginning of the training, when all parameters are random numbers, the start of generated words is completely random, and it makes the training of the rest of the sentence meaningless, as our objective function is based on the conditioned probability over previous words. One way to deal with this is teacher forcing or professor forcing[21], and the idea is very intuitive: we use the ground truth word instead of the generated word as the model input.

It is clear that this setting will decrease the model's generality, as we do not have the ground truth text at test time. To prevent it, we set this process as a stochastic approach. At each time step, there is a teacher forcing ratio λ determines if using ground truth word or generated word. This ratio is high at the beginning and will decrease with the process of training:

$$\lambda = \frac{1}{1 + epoch} \tag{5.21}$$

where *epoch* is the current training epoch count.

5.4 Experiment result

5.4.1 Evaluation metric

Automatic evaluation of text generation is very challenging and is still an open question to the NLP community[9]. It is difficult because two completely different expressions may have exactly the same meaning. "I am happy to do so.", "No problem." and "I got it." represent the same semantic information under certain circumstances, and there are many more different expressions as well. But in text generation, if the target is "I am happy to do so." and model prediction is "no problem.", the model will mark it as a big error and learns against it. On the other hand, two similar expressions might have very different or even opposite meaning, e.g., "There are many French restaurant near the city center." and "There are no French restaurant near the city center" are two opposite sentence, but model will give it a very tiny loss as the majority of words are the same. However, to solve this problem requires the model to fully understand the semantic meaning of each sentence, which is equivalent to strong AI, and there is still a long way to go[9].

Bleu score is firstly introduced to evaluate machine translation tasks[44], but it can be used in any text generation task with ground truth target text. It gives a score between 0 and 1. The higher the better. The idea is to find how many N-gram phrases in the predicted sentence are also in the target sentence. For example, if $N = 1$, the bleu score is equivalent to the percentage of words that are correctly predicted. Usually, Bleu score is the average of scores with a set of $N = 1, 2, ...$ Basically, short N evaluates word

precision, and large N evaluates the fluency of the generated sentence.

Compare with evaluation by the accuracy at each position, the Bleu score can better cover more correct predictions. But it is not perfect, the previous example “I am happy to do so.” and “no problem.” still gives a zero score.

As we are dealing with a task-oriented dialogue system, we also evaluated the model with “inform” rate and “success” rate, the “inform” rate indicates whether the output response has provided an appropriate entity, and “success” rate indicates whether the response answered all requested attributes. For example, if requested attributes of current utterance is *place: centre* and *postcode: YO14RT*, then whether the response includes *centre* and *YU74RT* are two independent sample of “inform” rate, and only if the response includes both attributes then it is a positive sample of “success” rate[8].

5.4.2 Result

Table 5.1: Experiment result on MultiWOZ 2.0 data set

	Bleu score	Inform rate	Success rate
Baseline[7](With oracle BS)	0.189	71.33	60.96
Our model(With predicted BS)	0.174	71.15	60.28
Our model(With oracle BS)	0.178	78.81	67.47

Table 5.1 shows the experiment results on MultiWOZ 2.0 data set, BS is the belief state, with oracle BS means using the ground truth state provided by the data set, with predicted BS means using belief state tracker in chapter 4 to predict the state.

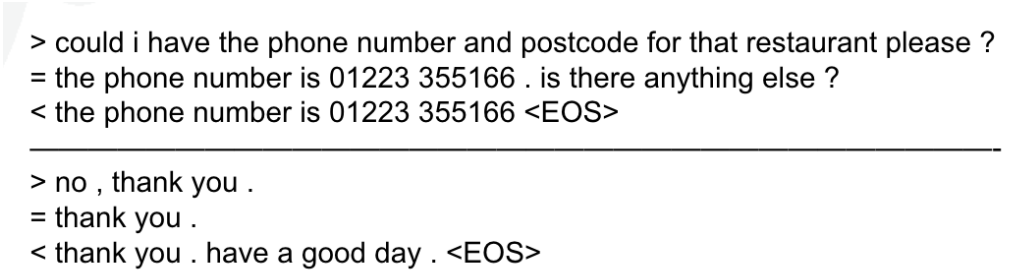

Figure 5.7: An example generated dialogue of our model, the symbol $>, =, <$ indicate user query, response in data set and generated response respectively.

The baseline model we used here for evaluation is a trivial seq2seq model with a discrete database accessing component as additional features[7], it uses the annotations of the dialogue state are used as an oracle tracker. For comparison, we tested our model with and without oracle dialogue state. As shown in table 5.1, our model gets close performance even without oracle state, and beat the baseline model for around 7% at both inform rate and success rate. This indicates that with our memory architecture the model is able to provide more useful information extracted from dialogue history and knowledge base. For the bleu score, we can see that our model get slightly worse score compared with the baseline model, this could be caused we used MMI instead of MLE as the loss function, which encourages the model to output diverse expressions and discourages common expressions. As discussed before, the bleu score evaluates how fluent a sentence is and since we are dealing with task oriented dialogue system, it is more important to be able to successfully provide correct information than output fluent but meaningless response.

Figure 5.7 shows an example of responses generated by our model, the

average response time is around 40-50 ms, which means it can give almost instant response when deployed for business service.

5.5 Summary

In this chapter, we introduced sentence generation in a dialogue system, starting with a language model. We proposed a novel memory-based sentence generator that outperforms the baseline model. Our experiment shows that memory architecture can improve the generation performance of a dialogue system. Also, the connection with KB can hugely improve dialogue system performance, and the quality of the dialogue state is important.

Chapter 6

Conclusions

6.1 Conclusions

This thesis focused on memory architecture for task-oriented dialogue state tracking and text generation. We used three steps approach to build a task-oriented dialogue system, namely the encoder, dialogue state tracker, and sentence generator. We implemented three independent experiments in Chapter 3,4,5 respectively.

Chapter 3 creates a novel data set for Twitter event detection and builds an event detector based on a sentence encoder. This predictor is further used as part of the evacuation planning system. Our experiment shows that the sentence encoder can preserve enough information for downstream tasks.

In Chapter 4, we proposed a novel memory-based dialogue state tracker which outperforms a set of benchmark models. The core contribution is to incorporate the memory architecture into the dialogue state tracking system.

In Chapter 5, we introduced our experiments on memory-based sentence generation for a dialogue system. The generator is based on the Seq2Seq

model, consists of memory architecture, copying mechanism, KB searcher by dialogue state, and text generator. As a result, our model outperforms benchmark models.

6.2 Why memory helps

Our experiments have shown that the memory architecture is useful for dialogue system tasks. The essence of memory architecture is to build extra connections on the sentence level. Why would this helpful?

Deep learning approaches for the natural language process are intrinsically statistical models and learn mainly the grammatical rules of language. Word embeddings bring part of the semantic information of text, but certainly, it can not preserve all information, and in downstream processes such as language models, it learns mainly the dependencies between words. For instance, it is simple for a language model to predict the upcoming word after "thanks very" is "much." But it will be much difficult to predict what is the next word after "I do not like apple pie because," and without explicit information, it is impossible to predict the word after "my name is." These kinds of predictions require logic and reasoning over context and knowledge and even common sense. Current NLP approaches are very weak on this aspect.

The memory architecture attempts to build a logic structure over the usual NLP approach. It predicts the next word based on both long-term memory and short-term memory and queries the knowledge base if needed. This is intuitively similar to the thinking of human beings. Practically, there

are neurons in charge of establishing memory and link them to the model's attention, and so there are more robust connections between information and knowledge-based with the model predictor, so it is a reasonable result to see a boost in performance.

6.3 Limitation on pure statistical model

BERT as the state-of-the-art language model has been very successful recently. However, although BERT was trained on a gigantic data set, it is still a purely statistical model. Our experiment in Chapter 4 shows that a Bert-based sentence encoder does not make much difference compared with RNN sentence encoder in a dialogue state tracking task. The improvement comes from the downstream processes. This shows that the semantic information might have reached its limit, and there is still space to be improved for syntactic information. Also, this result indicates that a massive neural network that works as a black box may not be the final answer for natural language understanding. We need a more delicate structure design for each specific task to make logic and reasoning possible.

In chapter 5, our experiment shows that connection with KB can hugely improve the model's performance, but this is only just a beginning to connect the statistical model to the real world. The knowledge base has to be manually created, so it is expensive and lacks generality and is expensive to update as well. Also, information is stored in KB as triples, therefore it is very difficult, perhaps not possible, to cover all information in the real world. Compared with the machine, when human learns language, we have much more access to the real world, we could interact with the real world by vision,

hearing, smelling, touching and tasting. When we learn the word "apple," we have the opportunity to see its color and shape, could touch it and feel its hardness. We first learned that if we yelled the word "apple" to our parents, we could eat it, then we will learn imperative sentence such as "*give me that apple.*", and later we may learn to comment "*this apple tastes good, I like it.*" These interactions are called *language games* by Wittgenstein, and by these language games, we are able to establish a comprehensive picture of the word "apple." The machine, on the other hand, have only access to the order of the alphabet, "a,p,p,l,e", the model might know that apple can make pies because it has seen the phrase "apple pie" several times, but it is very likely that it does not know what an apple pie tastes like. There is no mental subject to participate in the language games.

Connection to the KB is one attempt to add external information to the statistical model. Another approach is the multi-model approach such as visual question answering[1], it gives the model access to both an image and a text question. In conclusion, to develop towards strong AI, more parameters, bigger models, and a larger data set are not enough. We have to put more attention on more carefully designed network structures and give models more connections with the real world.

6.4 Future work

1. Reinforcement learning model is a different learning methodology for dialogue systems compared with the supervised approach, and it is more like when human learning language. We speak with a purpose and will strengthen the learning when the purpose is achieved. Wittgenstein called this the

"language games." Research has shown RL algorithm can be useful for text generation task[32][12]. Applying RL to dialogue systems is intuitively worth trying. For example, in this thesis we used supervised learning model to build task oriented dialogue system, and the goal of our model is just to output sentence that as close to the response given in original data set, in reinforcement learning setting, we could treat the dialogue system as a game, once the output of the model successfully provided correct information (the inform rate metric in chapter 5), we give our model a reward, and if not success, we give penalty. This process is more directly linked to the task itself and more intuitively reasonable.

2. Another possible future work is human evaluation. As we mentioned before, automatic evaluation of sentence generation is a very tough task, and manual review could better evaluate the quality of the generated sentences. However, human evaluation is not useful for training the models, as nobody can work like a machine and give instant loss for all training instances. So with limitations on time and resources, we have not performed a human evaluation for the generator.

3. Our current dialogue model is mainly in question answering schema, and is trained only on task oriented dialogue data set. When human learns language, we start with simple day-to-day dialogues and usually does not have very specific purpose, following this line of thought, it could be a reasonable idea to incorporate external day-to-day dialogue data set and train our model with it at first as a sort of transfer learning. The data set specifically related to the task is small and expensive but open dialogue data set is much more

than that as every dialogue data set can be used as pre-training data set.

4. In a task oriented dialogue system there are several clear routine format of response, such as *the location is at xxx, post code is xxx and telephone number is xxx. Do you need anything else?* So it is reasonable to manually build a collection of skeletons of response with key information left blank, every time our model generate a response it first pick a skeleton and then fill in missing key information. Intuitively, this may improve the bleu score of the model as the skeleton of the response perfectly follows routine response.

Bibliography

[1] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.

[2] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *5th International Conference on Learning Representations, ICLR, Palais des Congrès Neptune, Toulon, France*, 2017.

[3] Clara Ayora, Victoria Torres, Manfred Reichert, Barbara Weber, and Vicente Pelechano. Towards run-time flexibility for process families: open issues and research challenges. In *International Conference on Business Process Management*, pages 477–488. Springer, 2012.

[4] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA*, 2015.

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jan-

vin. A neural probabilistic language model. *The journal of machine learning research*, 3:1137–1155, 2003.

[6] Dan Bohus and Alexander Rudnicky. A" k hypothteses+ other" belief updating model. In *Proc. AAAI Workshop on Statistical and Empirical Approaches to Spoken Dialogue Systems*, 2006.

[7] Pawel Budzianowski, Iñigo Casanueva, Bo-Hsiang Tseng, and Milica Gasic. Towards end-to-end multi-domain dialogue modelling, (cued/f-infeng/tr.706), https://doi.org/10.17863/cam.31464. 2018.

[8] Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Ultes Stefan, Ramadan Osman, and Milica Gašić. Multi-woz - 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)*, 2018.

[9] Asli Celikyilmaz, Elizabeth Clark, and Jianfeng Gao. Evaluation of text generation: A survey. *arXiv preprint arXiv:2006.14799*, 2020.

[10] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December,Montreal, Quebec, Canada*, 2014.

[11] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[12] Heriberto Cuayáhuitl. Simpleds: A simple deep reinforcement learning dialogue system. In *Dialogues with social robots*, pages 109–118. Springer, 2017.

[13] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[14] Wim De Mulder, Steven Bethard, and Marie-Francine Moens. A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language*, 30(1):61–98, 2015.

[15] Ferdinand De Saussure. *Course in general linguistics*. Columbia University Press, 2011.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Minneapolis*, 2019.

[17] Mihail Eric, Rahul Goel, Shachi Paul, Abhishek Sethi, Sanchit Agarwal, Shuyang Gao, and Dilek Hakkani-Tür. Multiwoz 2.1: Multi-domain dialogue state corrections and state tracking baselines. *Computing Research Repository*, abs/1907.01669, 2019.

[18] Mihail Eric, Lakshmi Krishnan, Francois Charette, and Christopher D Manning. Key-value retrieval networks for task-oriented dialogue. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, Antwerp, Belgium*, pages 37–49, 2017.

[19] Shuyang Gao, Abhishek Sethi, Sanchit Agarwal, Tagyoung Chung, and Dilek Hakkani-Tur. Dialog state tracking: A neural reading comprehension approach. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue, Stockholm, Sweden*, pages 264–273, 2019.

[20] Rahul Goel, Shachi Paul, and Dilek Hakkani-Tür. Hyst: A hybrid approach for flexible and accurate dialogue state tracking. *arXiv preprint arXiv:1907.00883*, 2019.

[21] Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: a new algorithm for training recurrent networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems, Centre Convencions Internacional Barcelona, Barcelona SPAIN*, pages 4608–4616, 2016.

[22] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

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

[24] Minlie Huang, Xiaoyan Zhu, and Jianfeng Gao. Challenges in building intelligent open-domain dialog systems. *ACM Transactions on Information Systems (TOIS)*, 38(3):1–32, 2020.

[25] Sekitoshi Kanai, Yasuhiro Fujiwara, and Sotetsu Iwamura. Preventing gradient explosions in gated recurrent units. In *Advances in neural information processing systems*, pages 435–444, 2017.

[26] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J Inman. 1d convolutional neural networks and applications: A survey. *Mechanical systems and signal processing*, 151:107398, 2021.

[27] Ryan Kiros, Ruslan Salakhutdinov, and Rich Zemel. Multimodal neural language models. In *International conference on machine learning*, pages 595–603. Proceedings of Machine Learning Research, 2014.

[28] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.

[29] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International conference on machine learning*, pages 1378–1387, 2016.

[30] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[31] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. In *The 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologie, San Diego, California*, 2016.

[32] Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. Deep reinforcement learning for dialogue generation. In *Conference on Empirical Methods in Natural Language Processing, Texas, USA*, 2016.

[33] Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. Dailydialog: A manually labelled multi-turn dialogue dataset. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Taipei, Taiwan*, pages 986–995, 2017.

[34] Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Conference on Empirical Methods in Natural Language Processing, EMNLP, Lisbon, Portugal*, 2015.

[35] Andrea Madotto, Chien-Sheng Wu, and Pascale Fung. Mem2seq: Effectively incorporating knowledge bases into end-to-end task-oriented dialog systems. In *The 56th Annual Meeting of the Association for Computational Linguistics, Melbourne, Australia*, 2018.

[36] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[37] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[38] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association, Makuhari, Chiba, Japan*, 2010.

[39] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[40] Nikola Mrksic, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gasic, Pei-hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J Young. Multi-domain dialog state tracking using recurrent neural networks. In *proceedings of the Association for Computational Linguistics, Beijing, China*, 2015.

[41] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *proceedings of the 27th International Conference on Machine Learning, Haifa, Israel*, 2010.

[42] Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, 2015.

[43] Christopher Olah. Understanding lstm networks,. *https://colah.github.io/posts/2015-08-Understanding-LSTMs/*, 2015.

[44] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceed-*

ings of the 40th annual meeting of the Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, pages 311–318, 2002.

[45] Colin Paterson, Radu Constantin Calinescu, Di Wang, and Suresh Kumar Manandhar. Using unstructured data to improve the continuous planning of critical processes involving humans. In *14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. York, 2019.

[46] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), Doha, Qatar*, pages 1532–1543, 2014.

[47] Lin Qiu, Yunxuan Xiao, Yanru Qu, Hao Zhou, Lei Li, Weinan Zhang, and Yong Yu. Dynamically fused graph network for multi-hop reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy*, pages 6140–6150, 2019.

[48] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[49] Osman Ramadan, Paweł Budzianowski, and Milica Gasic. Large-scale multi-domain belief tracking with knowledge sharing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Melbourne, Australia*, pages 432–437, 2018.

[50] Nils Reimers, Iryna Gurevych, Nils Reimers, Iryna Gurevych, Nandan Thakur, Nils Reimers, Johannes Daxenberger, Iryna Gurevych, Nils Reimers, Iryna Gurevych, et al. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Hong Kong, China*. Association for Computational Linguistics, 2019.

[51] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[52] Avi Rosenfeld, Inon Zuckerman, Erel Segal-Halevi, Osnat Drein, and Sarit Kraus. Negochat: a chat-based negotiation agent. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, Paris France*, pages 525–532, 2014.

[53] Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials, Minneapolis*, pages 15–18, 2019.

[54] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, Arizona USA*, 2016.

[55] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and

Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the conference on empirical methods in natural language processing, Edinburgh, UK*, pages 151–161. Association for Computational Linguistics, 2011.

[56] Shane Storks, Qiaozi Gao, and Joyce Y Chai. Commonsense reasoning for natural language understanding: A survey of benchmarks, resources, and approaches. *arXiv preprint arXiv:1904.01172*, pages 1–60, 2019.

[57] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems, Montréal CANADA*, pages 2440–2448, 2015.

[58] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems, Montréal CANADA*, pages 3104–3112, 2014.

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems, Long Beach Convention Center, Long Beach*, pages 5998–6008, 2017.

[60] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems, Montréal CANADA*, pages 2692–2700, 2015.

[61] Richard S Wallace. The anatomy of alice. In *Parsing the Turing Test*, pages 181–210. Springer, 2009.

[62] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[63] Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina Maria Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve J Young. A network-based end-to-end trainable task-oriented dialogue system. In *proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, Valencia, Spain*, 2017.

[64] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In *Proceedings of the 23rd international conference on World wide web, Bloomington Indiana USA*, pages 515–526. ACM, 2014.

[65] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

[66] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[67] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Towards universal paraphrastic sentence embeddings. *arXiv preprint arXiv:1511.08198*, 2015.

[68] Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. Transferable multi-domain state generator for task-oriented dialogue systems. *arXiv preprint arXiv:1905.08743*, 2019.

[69] Puyang Xu and Qi Hu. An end-to-end approach for handling unknown slot values in dialogue state tracking. *arXiv preprint arXiv:1805.01555*, 2018.

[70] Wen Zhang, Yang Feng, Fandong Meng, Di You, and Qun Liu. Bridging the gap between training and inference for neural machine translation. *arXiv preprint arXiv:1906.02448*, 2019.

[71] Victor Zhong, Caiming Xiong, and Richard Socher. Global-locally self-attentive encoder for dialogue state tracking. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Melbourne, Australia*, pages 1458–1467, 2018.

[72] Qingyu Zhou, Nan Yang, Furu Wei, and Ming Zhou. Sequential copying networks. In *Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA*, 2018.

APPENDIX

Table 6.1: Number of possible values for each (domain, slot) pairs, for MultiWOZ 2.0 and MultiWOZ 2.1 data set.

Slot Name	MultiWOZ2.0	MultiWOZ2.1
taxi-leaveAt	119	108
taxi-destination	277	252
taxi-departure	261	254
taxi-arriveBy	101	97
restaurant-people	9	9
restaurant-day	10	10
restaurant-time	61	72
restaurant-food	104	109
restaurant-pricerange	11	5
restaurant-name	183	190
restaurant-area	19	7
hotel-people	11	8
hotel-day	11	13
hotel-stay	10	10
hotel-name	89	89
hotel-area	24	7
hotel-parking	8	4
hotel-pricerange	9	8
hotel-stars	13	9
hotel-internet	8	4
hotel-type	18	5
attraction-type	37	33
attraction-name	137	164
attraction-area	16	7
train-people	14	12
train-leaveAt	134	203