A Study on Bidirectional Decoder of Neural Machine Translation

Yang Zhen

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
June, 2018

Master's Thesis

A Study on Bidirectional Decoder of Neural Machine Translation

1610200 Yang Zhen

Supervisor :	Nguyen Le Minh
Main Examiner :	Nguyen Le Minh
Examiners :	Nguyen Le Minh
	Satoshi Tojo
	Kiyoaki Shirai
	Shinobu Hasegawa

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
Information Science

June, 2018

Abstract

Machine translation is one of the most active research areas in natural language processing (NLP) field. Although the history of machine translation study can be traced back to the middle of last century, until recently, most of the research is ranged from rule-based direct translation to interlingua method to statistical machine translation (SMT) method. Neural machine translation (NMT) is similar to an idea that appeared firstly in the 1990s, but no further development was undergone because of the constraints of data and computation resources at that time.

As the rise of deep learning, we have witnessed a great number of impressive achievements in various fields, especially computer vision, speech recognition and natural language processing. Of course, it also brings a revolution to machine translation. It is the appearance of NMT, which starts from using the neural network as a component in a phrase-based SMT system, and later developed to a pure neural network machine translation architecture. Although NMT has achieved a lot of impressive results, it still has many challenges, and it needs to make more refinements. Current most successful NMT models include three types: the recurrent neural network (RNN) based, the convolutional neural network (CNN) based and the pure attention based Encoder-Decoder sequence-to-sequence model. In this thesis, we focus on the RNN based one, especially it's decoding process because the current decoding process normally uses only unidirectional information of target sentence leaves the bidirectional information unexploited. Even though there are several works about using bidirectional information of target sentence, you have to train the backward model first, which means longer training time and bigger parameter size. So it is worth to explore training forward and backward decoding in an integrated model.

This thesis presents our research on NMT and our contributions: (1) we implement an NMT model, and make it becomes a strong baseline to compare with state-of-art models in terms of BLEU score through exploring parameters; (2) we implement several multi-task learning models to make bidirectional decoding and compare these models with each other and baseline, and then analyze the real translation result; (3) we also propose a regularization way to build bidirectional decoder and compare with other models, and analyze the translation result; (4) we combine the multi-task learning model and regularization model to make a combined model, which achieve further improvement than baseline model.

At first, we implemented a prevalent NMT model, the RNN based Encoder-Decoder model with attention mechanism. In this model, the source and target sentences are just

viewed as sequences of tokens, and our model translates sequences in the source language to sequences in the target language. Based on this basic model, we explore the influence of beam size and auxiliary length and coverage penalty in the beam search decoding process to the translation result. By doing this, we hope to achieve a strong baseline model that can compete with state-of-art models. Our result shows in our model, the optimal beam size ranges from 5 to 10 and if we pass some points, the increase will not help translation and even make the results worse. The reason might be that the big beam size makes a shorter translation, which can lead to worse performance. And also for length and coverage penalty parameters, in terms of BLEU score, it can make a big improvement when compared with baseline model, especially when both of them take an approximate value of 0.33. However, in terms of NIST score, it seems there is no substantial improvement in the translation result. And for both metrics, it shows that both big length penalty and big coverage penalty does not help to improve translation results. Moreover, it can make the performance getting worse. After choosing the parameters, we achieve a strong baseline model, which has better performance than many state-of-arts models.

Our second experiment presented here is focusing on the multi-task learning (MTL) model, which takes forward and backward decoding as two related tasks and trains them together with some components shared between them. The sharable components include attention component, decoder word embedding, and generator, which outputs prediction from hidden states. According to sharing components, we proposed several models, in which some only share one component and some share multiple components at the same time. After implementing and training models, we make comparison among these models and baseline model. The result shows that the proposed models indeed make an improvement when compared with baseline model, especially for the model with sharing generator and the model with sharing both embedding and generator. When we did some deeper analysis of the real translation results, we found that the best-proposed MTL model actually captured more information than the baseline. However, this kind of sharing component model is a quite indirect way to make a connection between the forward and backward decoding. We can see that it will make worse results or no improved results when sharing some components, like sharing attention mechanism, and we do not know the reason clearly.

To make a more direct interaction between the forward and backward decoding, we can make a regularization directly between hidden states of forward and backward. We proposed two models. The first one uses$L2$ regularization loss directly between the forward and backward hidden states. But this method will make less flexibility of our model to produce hidden states. To make more flexible, we proposed the second model which add an affine transformation layer between the forward and backward hidden states. It transforms forward hidden states first before it calculates $L2$ loss. Next, we implemented these two models and trained them with weight annealing technique, then compared their results with baseline model. We find these two models have better performance than baseline model. And between two proposed model, the model which has affine layer can achieve better results. It can even achieve higher results than the MTL models. When we analyze the translation sentence, it indeed captures more information than the baseline.

However, for this model, the disadvantage is that we need to choose a good weight value for the regularization loss term.

After proposed two different ways to exploit bidirectional information of target sentences, we proposed two new models that take the best setting from MTL model, which is sharing embedding and generator model, and apply two kinds of regularization from above. We implement two models and compare them with the former proposed models and baseline model. We find it not only outperform the baseline model, but it also outperforms the former models, especially for the model with *L2* regularization have an affine layer. It becomes the strongest model among all proposed model. For deep analysis, we compare the performance of the best model and the baseline on different sentence length data. And we find the proposed model help the model to translate longer sentence a lot. Besides, for the generality of the proposed model, we also run several selected proposed models in more translation direction. It shows similar improvements. And we also run proposed models on the large dataset to further support the effectiveness of our proposed model. The result demonstrates the effectiveness of our model even training on the big dataset.

Although our work has several limitations, at least it can provide some help for other research work in this field. We wish the method we proposed here can inspire more similar ideas. The proposed methods and finding not only can be applied to NMT field, but also can be applied to many other tasks such as question answering and document summarization, or even image caption that are using RNN based Encoder-Decoder model.

Keywords: natural language processing, neural machine translation, sequence-to-sequence model, bidirectional decoding, multi-task learning, regularization

Acknowledgements

I always think that I am so lucky applied JAIST two years ago, although I know few about JAIST at that time. Because I really had a great time in JAIST, and I think it will become a precious memory in my life. Here I made a lot of friends from all over the world, here I met a lot of respectable people that I can learn a lot from them, here I got a lot of comrades to fight for dream together.

First, I would like to thank my main supervisor, Associate Professor Nguyen Le Minh, for his kindly supports. I always thanks him for accepting me into his lab in the middle of my Master program and making a very good atmosphere in our lab, which makes a very interesting seminar time. If no his guidance I would not finish my degree smoothly. I am also thanks to my previous supervisor, Associate Professor Fumihiko Asano, for taking care of me half a year and helping me made a strong foundation of mathematics.

I appreciate the feedback from Professor Satoshi Tojo, Associate Professor Kiyoshi Shirai, and Associate Professor Shinobu Hasegawa, their comments helped me much in my research. I also have benefited a lot from my internship advisor, Professor Mineo Kaneko. Not only I learned an amount of knowledge from his System Optimization class, but he also gives me many advises about my internship.

In addition to research, I would like to show my great appreciation to all the people taking care of me during the time I stayed in JAIST. First of all, I need thanks to Professor Shungo Kawanishi, I really don't know how much I learned from him. I always think he as the mentor in my life, when I stay beside him I learned a lot from him. From the Japan Study class to Global Leader Workshop to Malaysia internship, he taught me much in terms of how to become a good person, which makes me grown up and changed a lot. Of course, I also need thanks his beautiful and kind assistant Doctor Kotona Motoyama for taking care us in Malaysia. And also I need thanks Mr. Romzie and Anas at that time.

What's more, I would like to thank the staffs took care of me in JAIST, such as Shintani-san, Miyashita-san, and Yamagishi-san. Without their support, I can never live so fully in JAIST. And also I need thank residents in Nomi-city, especially the kind people from NIFA. They always organize a lot of local activities, which make my life variety and help me release stress from research.

I would like to thanks Rotary Club to select me as Rotary Yoneyama Scholarship student, which helped me much in the financial aspect. Furthermore, I need give my great appreciation for respectable people in Rotary Club, especially my counselor Takai-san, and Koyanagi-san, Muranaka-san and Yasoyama-san. They taught me a lot thing about Japan, and show me a great mind to contribute to world peace.

I would like to thank all members of our JAIST Music Circle. It is the support from your guys that we can have a lot of fun time. I still remember the first time when we went up to the stage in International Student Gathering and the memorable and crazy traveling together in Shiragawago.

Special thanks to all members in Nguyen's Laboratory. I spend a very happy time in Nguyen lab and learned a lot of things from all the members. For example, Chien-san helped me to start the research, Anh-san, and Tien-san always discussing with me about research and give advice, Viet-san gave me a lot of technique supports, etc.

One more special appreciate to my friend Papa, it is she helped me revise my poor writing thesis.

Last but not least, I need to give my deepest gratitude to my family who gives me understand and chance to study abroad. Especially, I would like to thanks my uncle who always talk with me to encourage me to have a good life.

When I look back, I find there are so many things happened and so many people I met in these two years. There are so many other people I want to mention here, but can not because of the limitation of space. I hope you can understand this.

Yang Zhen
June 2018

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we will give an introduction about the background of the research problem, the motivation to do this research, and the ultimate goals of this research.

1.1 Background

Nowadays, because of the development of the Internet and Social Network Software (SNS), everyone can share their own opinions with other people. At the same time, these things also make the distance between people become shorter, it is easy for us to access the information from other countries by ourselves. However, even if we can access the foreign websites, but a lot of them will be in foreign languages, which many of us cannot understand. Beside of Internet, for many cultures there is much knowledge cannot be shared with the world, because of the barrier of language. Although we can use the human translator to translate this information, it means high cost and low-speed (Aslerasouli and Abbasian, 2015). In this way, there is an urgent need to have an automatic translation system to break this wall, so that people can access information from other languages freely.

Machine translation is a research field in natural language processing (NLP). It can be traced down to the middle of 20th century when Warren Weaver first time published his influential Warren Weaver's memorandum (Hutchins, 1997). Machine translation aims to automatically translate the content from one source language to the target language. According to the requirement of the speed and quality, the result of machine translation can be used to three main purposes: assimilation, dissemination, and communication, which include the range from rough just for understanding the content to translate the text for publication. And because machine translation is a high-level task in NLP area, so the development of it can also boost many other tasks in NLP area. For example, current very widely used attention mechanism was first mentioned in a machine translation model (Bahdanau et al., 2014).

From the early days, many approaches have been explored in machine translation area, ranging from rule-based direct translation, to transfer methods which use more advanced way to include linguistic knowledge like morphological and syntactic informa-

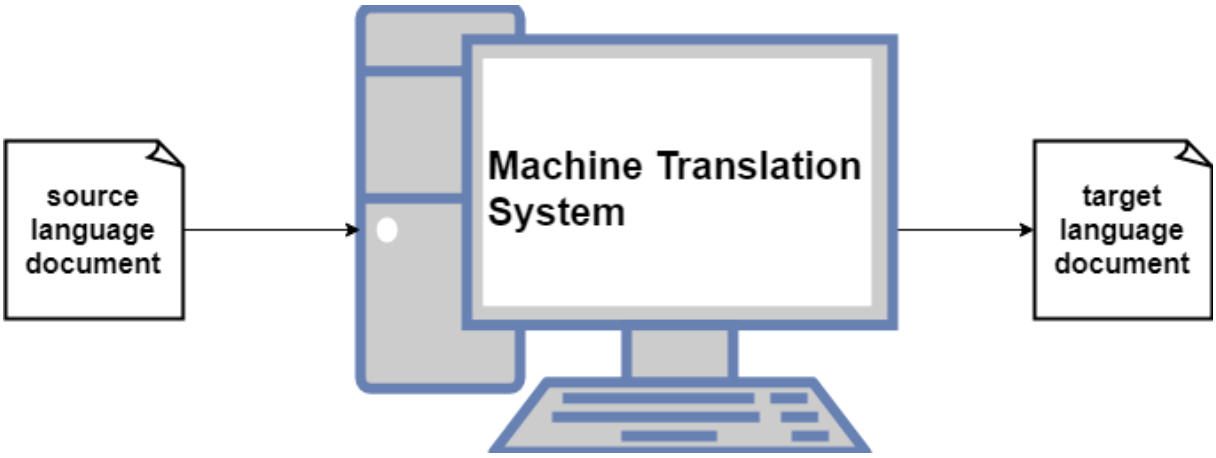


Figure 1.1: Illustration of a machine translation system

tion, and up to some interlingua methods that use an abstract meaning representation. Recently, the most widespread model of machine translation is phrase-based statistical machine translation (SMT) model, which divide the sentence into small blocks called phrase and have several separately engineered subcomponents.

However, as the era of big data and big computation come, the deep learning technique achieves a great amount of success in many fields including NLP field. One of the examples that deep learning succeeded in NLP is the neural machine translation (NMT)(Bahdanau et al., 2014; Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014), in which impressive performances can be seen. It outperformed the previously dominant phrase-based SMT method and became the state-of-art approach (Sennrich et al., 2017a; Vaswani et al., 2017; Yang et al., 2017) to machine translation task. Furthermore, NMT has already been introduced into the real world service and applications by many companies, such as Google (Wu et al., 2016), Microsoft and Yandex. In a recent paper, there is one NMT model that even achieves human parity performance on Chinese to English language pair (Hassan et al., 2018) in terms of BLEU score. Beside of better performance, another big advantage of NMT is its end-to-end approach, which means we do not need manual engineers the subcomponents of a system like SMT, and we just need to feed the input and output language pair to our system to train it.

Currently, in terms of the neural network architecture used in the models, the state-of-art NMT models can mainly be divided into three categories:

- **Recurrent neural network (RNN) based** sequence-to-sequence model. This is the most popular and vanilla model for neural machine translation and also the first proposed model. The source words will be mapped into word embeddings and then pass to RNN encoder, and the encoder will encode source sentence into a sentence representation, which will be feed into decoder RNN decoder as the initial hidden state. Latter, RNN decoder will output target sentence prediction with the help of attention mechanism to check encoder hidden states. The advantage of this model is that it has been studied longest time, and there is much-related research work. So

it is easy to find a mature model and already well-implemented model code (Klein et al., 2017; Neubig, 2015; Sennrich et al., 2017b).

- **Convolutional neural network (CNN) based** sequence-to-sequence model (Gehring et al., 2016, 2017). Compare to RNN based model, it is a relatively newer model, which has been used in many questions after it is proposed. This CNN based model replace the RNN encoder and decoder to CNN component and can achieve competitive result with RNN based model. The advantage of this kind of model is that because the CNN components can compute in parallel, the training speed will be faster than RNN based model.

- **Pure attention based** (Ahmed et al., 2017; Vaswani et al., 2017) transformer model. This one is the newest model, which was just proposed in the late of 2017. Instead of using attentional sequence-to-sequence with some form of recurrence or convolution like above two, the transformer model avoids the recurrence and convolution completely, and only depend on attention mechanism. This model has achieved state-of-art on several tasks, and now widespread among researchers and developers.

1.2 Motivation

Although the CNN based sequence-to-sequence model and attention based transformer model have achieved some good results recently, the RNN based sequence-to-sequence model is still the most prevalent model for NMT task. There are several reasons for this. First, it is the most intuitive and easiest one to understand these three models. When compared with other two, RNN is native to process sequence data, which means it already has an inductive bias in this model to process sequence data. So when you feed sequence data into RNN and get sequence data from RNN, it just as its nature. You do not even need the positional embedding like in transformer. Second, it has many mature open source research toolkit. In this way, people can easily download these toolkits and implement their own idea upon it, instead of build from scratch. Last, it still has many places worth to explore. As the research area about RNN is still developing fast, many results from it can be used into NMT model to achieve better result, for example, the new gated recurrent unit (GRU) (Cho et al., 2014) can get competitive than the old long short-term memory (LSTM) unit with simple structure, and the simple recurrent unit (SRU) (Lei and Zhang, 2017) can train as fast as CNN. Besides, we can also invent many new techniques (Xia et al., 2017) about RNN based NMT model to improve performance, like we will do here.

For RNN based sequence-to-sequence NMT model, the most well-used system will have multiple layers bidirectional RNN for the encoder, one kind of attention mechanism, and multiple layers unidirectional RNN for the decoder. Although the encoder is bidirectional, the decoder is always unidirectional, because of the natural property of decoding process for this sequence-to-sequence model. It has to translate the words one by one and

feed the output word of current time step as input for next time step. In this way, the decoding process can only translate in one direction at one time.

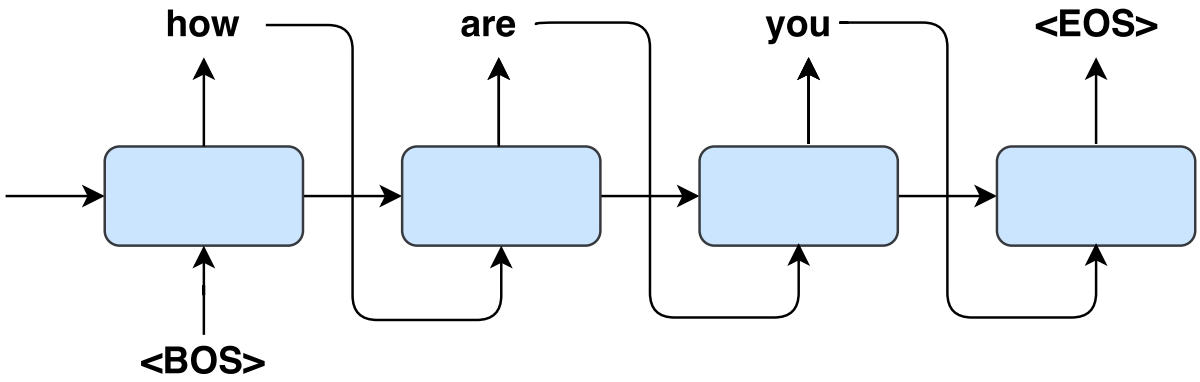


Figure 1.2: Example of unidirectional RNN decoder decoding process

On the other hand, bidirectional RNN (Schuster and Paliwal, 1997) have been proved that it normally can achieve a better result in many NLP tasks, since that bidirectional RNN can capture richer information from both directions of sequence data. In NMT area, of course, Sundermeyer et al. (2014) showed that bidirectional RNN can also achieve higher performance. However, for current popular RNN based sequence-to-sequence attention model, the only encoder is using bidirectional RNN. Therefore, if we can use the bidirectional information on the decoder side, then we can expect to achieve a better result for our NMT model.

About using bidirectional information of target language on decoder side in NMT, there are several papers have made some efforts to explore it. For example, Liu et al. (2016) tried to train two models on the same dataset, one is left-to-right and one is the right-to-left model, later they use ensemble method to rerank the result of the left-to-right model by the right-to-left model. And in another interesting recent paper, Zhang et al. (2018) use the hidden states from a pretrained right-to-left model as an additional context for attention component of the left-to-right model, and later output translate result. But for these two paper, if you want to use the backward information of target language you have to train a right-to-left model first, and then use rerank trick or attention technique. So we will ask is there any way to use the bidirectional information in one training model.

To achieve this, the first thing comes to my mind is the multi-task learning (Caruana, 1998) way. In the normal machine learning setting, we will train a single model and make it perform the desired task. Whereas when we focus on the single model, sometimes we just ignore some information might help this model get better performance. Specifically, the information comes from the training signal of the related task. So through sharing the similar representation between related tasks, we can make our model perform better. Here, we can think the left-to-right forward decoding and right-to-left backward decoding as two task, and train them together. We can let them sharing some components and representatives. In this way, the main task forward translation can get more information to help the main translation task.

Another way that we can consider to use bidirectional information of target language is regularization, which is one of the key components in machine learning area. Especially in deep neural network (Goodfellow et al., 2016), regularization allows the model to generalize well even when training on a finite training set. Therefore we can think about to use the information from the backward direction decoder as the regularization to forward decoder. In this way, the future information will become the regularizer (Serdyuk et al., 2017) to make forward translation perform better.

The last motivation for this master research is the lack of related implementation and research about this kind of end-to-end NMT. And actually, in NMT area, there are many implement detail and tricks when you want to do the research. For example, how many merge times it should be for byte pair encoding for preprocessing, and what preprocessing and post-processing technique we should use for current language pair. There are few articles about these details. So if there is no experienced people gives you suggestion and you explore by yourself, it will be quite frustrating. So we believe that if we write the detail about the whole implementation details of our research and talk about the tricks inside it, it can also help others a lot, especially for the newcomer of this topic.

1.3 Objectives

In this study, we restrict our research mainly on exploring *bidirectional decoder of RNN based sequence-to-sequence NMT model*. The goals of us will be:

- We would like to implement a baseline model for neural machine translation. It is the widespread RNN based sequence-to-sequence with attention model. We would like do some parameter exploration on this basic model, and make a strong baseline model, which can compete with the most state-of-art models.

- We would like to implement the different models based on the ideas of multi-task learning and regularization to make advantage of bidirectional information of target sentences. The models will be compare with each other and baseline. We also want to do a deeper analysis of the translation result to get a deeper understanding of the proposed models.

- We would like to combine the proposed two ideas to one model to expect see further improvement. Later, we would like to select the models with top performance among proposed model, and conduct more experiments to show the significance and credibility of proposed models. For example, we can do experiments on more translation direction and on the large dataset.

- Finally, we would like to implement the whole translation system including preprocessing and post processing. The details of these procedures will be recorded in the thesis and the implemented code will be released later. As I know, there is no such system be implemented in popular framework like PyTorch and make open-source to community.

Chapter 2

Literature review

In this chapter, we would like to introduce the basic theoretical frameworks in the RNN-based sequence-to-sequence attentional NMT model first (the other types of popular NMT models will be only mentioned roughly in the section 2.4). Then we will introduce the intuition behind the different ideas to use bidirectional information, like multi-task learning in section 2.2 and regularization in section 2.3. Last, we will introduce recent progress in neural machine translation area, especially for those have achieved great attention and make a big contribution to the community.

2.1 RNN-based Sequence-to-Sequence Attention NMT model

2.1.1 Byte Pair Encoding

Here we would like to introduce a very important technique in the preprocessing of NMT task. That is byte pair encoding (BPE), which is first proposed to data compression (Shibata et al., 1999), and later Sennrich et al. (2015) apply this to machine translation. In NMT, one of the big problems is the large vocabulary problem, especially for some rich morphological language like German. In some extreme situation, without any process, even a dataset can have millions of vocabulary. The large vocabulary of NMT model will make big problems. First, because the neural model of machine translation a large part of it is the embedding table, whose size will be decided by the vocabulary size and the embedding size. So if the vocabulary is too big, the model will also become too big to run on small chips. Second, the large vocabulary can also make the problem of sparsity. It will improve the difficulty of predicting the right word.

To solve this large vocabulary problem, the most common approach is to break the rare and long words into subword units. And BPE is the most popular way to break into subword units now. We need train BPE coding on the parallel corpus first and then apply it to the data. The detail will be, first, split the words in the corpus into characters. Second, the most frequent pair of characters is merged, and the vocabulary will increase

What makes this all the more grati@@ fying is that Members who serve on the competent parliamentary committee sometimes feel they are dealing with a mot@@ ley ass@@ ort@@ ment of un@@ connected legislative b@@ ills.
Transport, and this can never be repeated too often, cannot be com@@ part@@ ment@@ alized into separate modes.
It must always be seen in its enti@@ re@@ ty and sub@@ ordin@@ ated to over@@ riding strategic aims, such as the reduction of environmental pollution.

Figure 2.1: The example of sentences after apply byte pair encoding from WMT16 DE-EN English training data

one. Then repeat this step until the specified merge number times. The detailed algorithm will be like Algorithm 1.

Data: Training corpus
Result: The BPE encode vocabulary of this corpus
initialize the vocabulary with the character vocabulary;
$i \leftarrow 0$;
while *i less than number of bpe merge operation* **do**
    count all symbol pairs get frequency dictionary;
    sort frequency dictionary;
    add *symbol pair* with highest frequency into vocabulary;
    $i \leftarrow i + 1$
end

Algorithm 1: Byte pair encoding algorithms

There is one unintuitive thing about BPE technique. Normally, when we talking about subword units, we will come up with some words from linguistic, such as prefix and suffix. Whereas looking at BPE algorithm, we can find no place talking about linguistic knowledge but just count about the symbol pair frequency. In such easy way, BPE can even perform better than some linguistic-based subwords technique in many situations. The real example of data after BPE like Figure 2.1, in which we use @@ to divide words.

2.1.2 Encoder-Decoder approach

Almost all of the NMT model now (Gehring et al., 2017; Sennrich et al., 2017a) are using Encoder-Decoder approach. How is this Encoder-Decoder approach works? In the *encoder* phase, the encoder process input sentence and then output hidden states encode meaning of input sentence. We can think this hidden states as *input sentence embedding*. In the *decoder* phase, the encoded hidden states will be feed into the decoder to produce the translation. This procedure is illustrated in Figure 2.2.

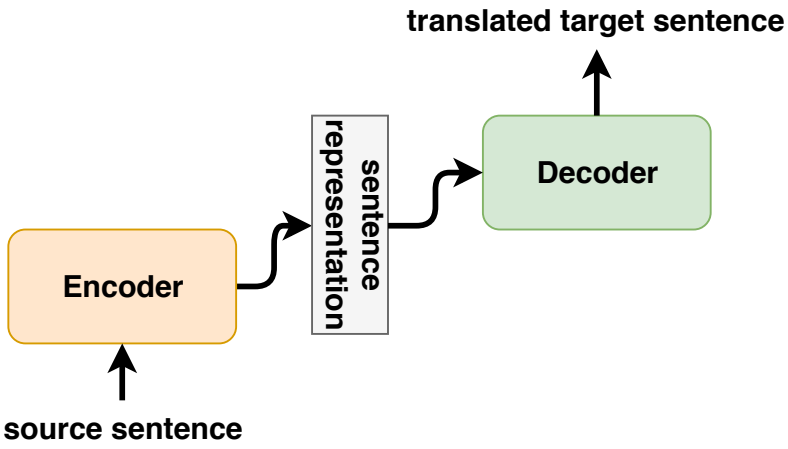


Figure 2.2: Encoder-Decoder approach of machine translation
Encoder (orange box) encode source sentence into sentence representation. Later, decoder (green box) use this representation to translate target sentence.

Obviously, when entering decoder phase, if we want to predict right translation, the quality of encoded representation from the encoder is very important. So it must remember all time step information of input sentence during encoder phase. And during decoder phase, not only we need to use the input hidden states to predict the whole target sentence, we also need to know which part of the sentence already be translated, and which part not still. All these procedures have to depend only on the operation of the fixed-size encoded representation. Of course, a problem comes with this model.

In practice, this proposed model will work for short sentences, but can not deal with long sentences. Cho et al. (2014) showed the increase of an input sentence length, the performance of this kind of basic encoder-decoder approach will deteriorate fast. Researchers have figure out some ways to solve this problem, such as feed encoded sentence embedding as input to the all-time step of the decoder. But later, it turns out the most effective way to solve this question is the star in deep learning now, the attention mechanism. We will talk about it in Section 2.1.4.

Before attention mechanism, we would like to talk about components compose encoder and decoder first. Although now if we talking about encoder-decoder or sequence-to-sequence model, many people will come up with RNN based encoder-decoder model. The real important thing behind the encoder-decoder approach is its idea. Anything can encode the input can be an encoder, and anything can do decode phase can be decoder. For example, CNN and recent pure attention based transformer model. But here we focus on RNN based one, so we will introduce the idea of RNN first.

2.1.3 Recurrent Neural Network

In machine learning area, there is no all-around model to deal with every problem, which we call it *no free lunch theorem* (Wolpert and Macready, 1997) and each model will have

its own *Inductive bias*, which means the assumption about the task in the model. For example, the main assumption of the convolutional neural network is the local invariant and parameter sharing among those local space parts. For recurrent neural network (RNN), the main assumption is the parameter sharing between data in different time step. That is why RNN is suited to deal with the sequence data like language and music.

Therefore, the main purpose of RNN is to capture the temporal dependencies between successive tokens. For example, we have an input sequence $X = \{x_1, x_2, ..., x_n\}$, and $x_t \in \mathbf{R}^d$ is input vector with d dimension. When RNN run over this input sequence, it will apply formula as bellow n times:

$$h_t = \sigma_h(W_h h_{t-1} + W_x x_t + b_h) \tag{2.1}$$

In above equation, there are two inputs. The x_t is the input vector at t time step as mentioned before, and h_{t-1} is the *hidden state* from last time step *t-1*. Here we assume $h_t \in \mathbf{R}^r$ is r dimension vector. Then $W_h \in \mathbf{R}^{r \times r}$ and $W_x \in \mathbf{R}^{r \times d}$ are the parameter matrices. $b_h \in \mathbf{R}^r$ is bias vector, and σ is a non-linear activation function, like *sigmoid* and *relu*. From the formulation, we can see current time step depends on two inputs, current input vector, and previous time step hidden state. This setting makes the RNN can delivery information from one-time step to another time step. One thing needs to pay attention is that in this formula, for a different time step, we share the weight matrices W_h and W_x. So we can think RNN as that in each time step, we feed input, and RNN calculates current hidden states from the input and the previously hidden states, then feed current hidden states to the same network, again and again until last of the input sequence. That is the reason why we call it *recurrent* neural network. A simple illustration of this kind of cyclical network as Figure 2.3.

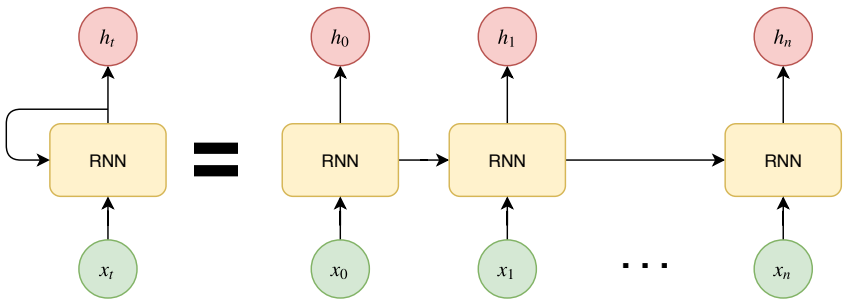


Figure 2.3: Recurrent neural network computation process.
Based on the previously hidden states and the current input vector (green circle), the RNN cell (yellow box) calculate current hidden states (red circle). Left is represent cyclic way by only using one RNN cell, and output hidden states will be feedback to the RNN cell. Right is the unrolled graph of the left one, just represent each time step explicitly. Although it looks different from left, they are actually the same thing.

This is the most vanilla RNN now people use, and its name is Elman RNN (Elman, 1990). Of course there is another variant RNN like Jordan RNN (Jordan, 1997). Although the assumption of RNN can let it works on any length sequence data theoretically, when

we apply it to the real problem, we meet a problem which is *vanishing gradient*. (Bengio et al., 1994) and (Pascanu et al., 2013) show that because vanilla RNN suffers from vanishing gradient, so it makes vanilla RNN can not learn long-term dependencies. Even though they give some solutions like regularization, the more effective and direct way is to extend RNN and add some new assumption in it. Therefore, several enhanced RNN such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) have been proposed. These units can capture longer-term dependencies.

Long Short-Term Memory

The main idea behind LSTM (Hochreiter and Schmidhuber, 1997) architecture is the gating mechanism. The LSTM unit has two kinds of hidden states, one is keeping the long-term memory of past sequence called *memory cells*, and another is the output memory of current time step depend on the long-term memory called *working memory*, or think it as the short memory. These two hidden states are controlled by *differentiable gating components*, which is logical gates simulated by smooth mathematical functions. Each LSTM unit has 3 kinds of the gate: *input*, *forget*, and *output* gate. Through the operation of these gates, LSTM will decide how much of the new input content should be written on the memory, and how much of the current memory should be forgotten. The mathematical definition of LSTM architecture is as follow:

$$
\begin{aligned}
s_t &= R_{LSTM}(s_{t-1}, x_t) = [c_t; h_t] \\
\\
c_t &= f \odot c_{t-1} + i \odot z \\
h_t &= o \odot tanh(c_t) \\
i &= \sigma(W_{xi} x_t + W_{hi} h_{t-1} + b_i) \\
f &= \sigma(W_{xf} x_t + W_{hf} h_{t-1} + b_f) \\
o &= \sigma(W_{xo} x_t + W_{ho} h_{t-1} + b_o) \\
z &= tanh(W_{xz} x_t + W_{hz} h_{t-1} + b_z)
\end{aligned}
\tag{2.2}
$$

The state of LSTM unit at time t is composed of two vectors, c_t and h_t. Here $[c_t; h_t]$ means concatenate of c_t and h_t vectors. $c_t, h_t, i, f, O, z \in \mathbf{R}^r$ are the memory component, hidden state component, input gate, forget gate, output gate, and update candidate respectively. $W^{x\circ} \in \mathbf{R}^{r \times d}$ are the weight matrices for each part that will product with current input vector, and $W^{h\circ} \in \mathbf{R}^{r \times r}$ are the weight matrices product with hidden states of previous time step. $b_\circ \in \mathbf{R}^r$ are bias vectors. From the equation we can see, at first we compute three gates based on linear combination of current input $x_t \in \mathbf{R}^d$ and previous state $h_{t-1} \in \mathbf{R}^r$, passed through a sigmoid activation function.

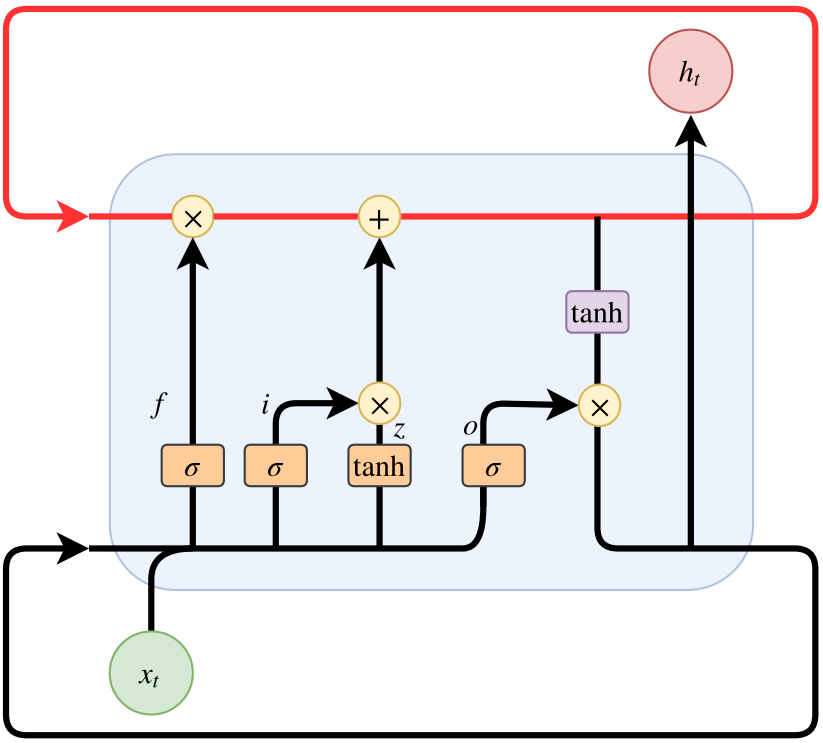


Figure 2.4: The detailed architecture inside Long Short-Term Memory unit
For current input vector (green circle) and previously hidden states, we pass through it to respective linear combination and activation function (orange box) to get f, i, z, o. f is the forget gate, which controls how much memory needs to forget through the element-wise product (yellow circle with times symbol). i is the input gate, and z is the input candidate. i controls which part of z need add to memory, then later add it to memory through element-wise plus (yellow circle with add symbol). After a tanh nonlinear transformation, o the output gate controls which part of memory should be output as the current hidden state (red circle). The core idea of LSTM is the gate mechanism and the memory. From the red line, we can clearly see why LSTM can prevent gradient vanishing problem.

Then we use tanh activation function and same kind of linear combination to calculate update candidate state z. Later, we update c_t memory: the forget gate decide how much memory from previous memory should be forgotten and remembered, and the input gate decide how much new information should be updated. Finally, we pass memory c_t through a tanh activation function and use output gate to decide which part of information should be current hidden state or output. Through these gating mechanisms, gradients related to memory c_j can keep high value across very long time. In this way, LSTM architecture can solve the problem of gradient vanishing. To get more intuitive understanding, please check Figure 2.4.

Bidirectional RNN

In a unidirectional RNN (the RNN we talk here include variants of RNN like LSTM), when computing the current hidden state, it only uses previously hidden state from one direction. In this way, when we take summarized hidden state at some time step, it only summarizes the information from one direction, the information before current time step, but without the later information. However, from our common sense, we know current word not only related to the word before it but also the word after it. So it makes sense we use two RNN run in the different directions, and then combine the information summarized by them. Then we get the bidirectional RNN (Bi-RNN) (Schuster and Paliwal, 1997): a forward RNN run over the sequence from left to right, and a backward RNN process sequence from right to left. Last, we concatenate the hidden states from both RNN. The architecture of Bi-RNN is shown in Figure 2.5.

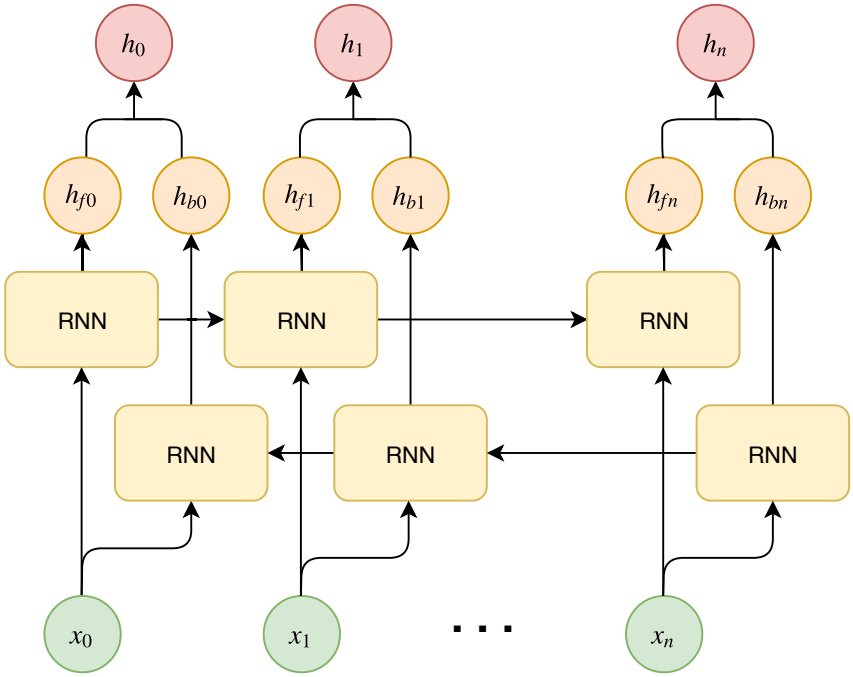


Figure 2.5: Bidirectional RNN illustration
The input vector (green circle) in each time step feed into two RNN (yellow box) in the different direction. And from each RNN, we get respective hidden states (orange circle). Then we concatenate these two hidden states to get BiRNN's hidden states (red circle).

The mathematical expression of Figure 2.5 would like:

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$$

$$\begin{aligned} \overrightarrow{h_t} &= RNN(\overrightarrow{h_t}, x_t) \\ \overleftarrow{h_t} &= RNN(\overleftarrow{h_t}, x_t) \end{aligned} \tag{2.3}$$

Using this kind of bidirectional technique, many researchers have achieved good results in many areas like machine translation (Sundermeyer et al., 2014), tagging (Wang et al., 2015) ...

2.1.4 Attention mechanism

As mentioned before, in the vanilla RNN encoder-decoder model, only the fix-sized encoded representation from encoder feed to the decoder. This in many conditions has no enough information to translate, like long length sentence. The most successful way to deal with this problem is *attention mechanism* (Bahdanau et al., 2014). Not like the traditional way just feed a sentence embedding at one time, attention mechanism let decoder check all time step hidden states of encoder when translating, and decide which words of source sentence will help translate current word most. The intuition behind it is quite clear because it like that we are doing alignment between the source sentence and the target sentence. After alignment, the translation will become easier. An example of alignment from attention is illustrated as Figure 2.6 taken from (Bahdanau et al., 2014).

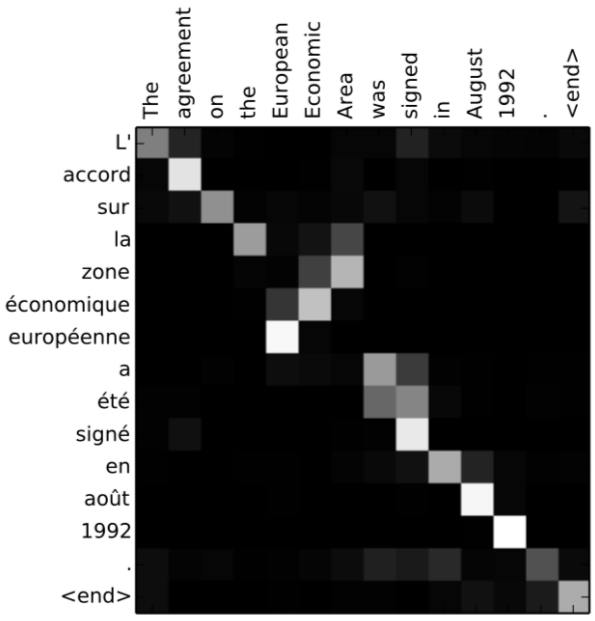


Figure 2.6: Example of alignment from attention mechanism
Language pair in the figure is the English-French pair. And white part in the alignment map means the alignment strength.

For conventional attention component in RNN encoder-decoder model, it will calculate a *context vector* based on the previous decoder hidden states and all hidden states from the encoder in each decoding time step. Later, this context vector will be used to help calculate current hidden states of decoder, and prediction words can generate from this hidden states. We compute context vector as follow.

$$
\begin{aligned}
e_{ij} &= a(s_{i-1}, h_j) \\
w_{ij} &= \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})} \\
c_i &= \sum_{j=1}^{n} w_{ij} h_j
\end{aligned}
\tag{2.4}
$$

with s is the hidden states from decoder, and h is the hidden states of the encoder. a is the score function to calculate a scalar score from s and h. In each time step i, we calculate all the score between previous decoder hidden states s_{i-1} and all n time step of encoder hidden states, and use softmax to get n weights $w_{i\circ}$ for each encoder hidden states. Later use these weights times respective encoder hidden states, and add together we can get context vector for the current time step c_i. This computation process draws in Figure 2.7.

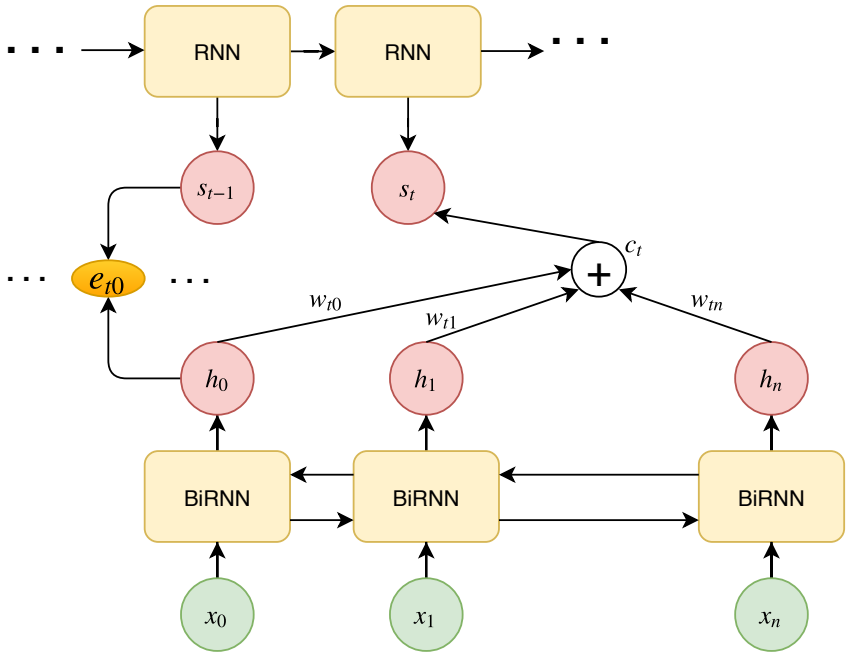


Figure 2.7: Attention mechanism in RNN encoder-decoder model
The encoder is bidirectional RNN, and it will output its hidden states (red circle with h inside) in each time step. When decoding, decoder will use previous hidden states s_{t-1} and encoder hidden states to calculate score $e_{t\circ}$ (orange oval). Then normalize these scores to get attention weights $w_{t\circ}$ for each encoder hidden states. Finally, use weights and encoder hidden states to get context vector c_t.

After we have context vector, we will use below equation to calculate current hidden states of the decoder.

$$s_t = RNN(s_{t-1}, c_t, y_t) \tag{2.5}$$

One more thing need mention is that according to the difference of score function, there are several different kinds of attention mechanism. Among them, two of the most common mechanism is additive variant (Bahdanau et al., 2014), and the multiplicative variant (Luong et al., 2015) which is less computation cost.

Present, attention mechanism has achieved impressive improvement in many tasks, such as image caption (Xu et al., 2015), machine comprehension (Hermann et al., 2015), and syntactic constituency parsing (Vinyals et al., 2015). And in NMT area, it already is the basic setting of the models.

2.2 Multi-Task Learning

In multi-task learning (MTL) (Caruana, 1998) setting, unlike the normal way that only focuses on one model and improves its performance, MTL advocates that we can train several related tasks together and let them boost each other or the main task. The reason this will work is that the related tasks will share some components between them and share representations. These tasks can help learn better-shared representations, and later these representations can help improve the performance of tasks back. Common MTL neural network model like Figure 2.8.

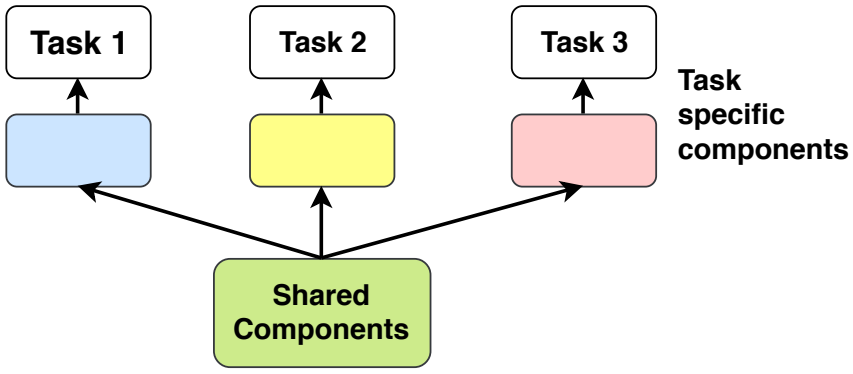


Figure 2.8: A typical multi-task learning setting
In a typical MTL setting, we have several related tasks, like task 1, task 2 and task 3. Then in the whole model, they will share some component, and also have their own specific components. When we training, just train them together.

Multi-task learning [1] has been used in many applications successfully, such as NLP (Collobert and Weston, 2008) and speech recognition (Deng et al., 2013). And recently, it gets a lot of attention and some successes(Hashimoto et al., 2016). However, which element makes the MTL model works well still need more exploration. Here are some

[1]To get an overview of MLT, please check this blog post: http://ruder.io/multi-task/

recent progress (Alonso and Plank, 2017; Bingel and Søgaard, 2017) about what makes MTL work.

There are mainly two kinds of way to sharing parameter in MTL. One is called *hard parameter sharing*, which will share the network components and multiple tasks use the totally same parameters. The another one is called *soft parameter sharing*, which will train several task on different set of network separately, but use the regularization technique to make the related component have similar parameter. The first way will be more parameter efficient, but less flexibility. Conversely, the soft way can make model more flexible, but has more parameters. The soft parameter sharing have close relation with regularization we will talk in Section 2.3.

2.3 Regularization Method

In machine learning area, the regularization method is one of the most prevalent technique to increase the generality of the learning model (Goodfellow et al., 2016). For more generality of the model means that the model not only performance good in training and validate data, but the model also can achieve a better result on the test set. Therefore we can also think regularization as a way to avoid over-fitting of our model. (Kukačka et al., 2017) definite it like:

Definition 2.3.1. Regularization is any supplementary technique that aims at making the model generalize better, i.e. produce better results on the test set

There are a lot of ways to do regularization, such as through data, through network architecture, and through regularization term (Kukačka et al., 2017), which is what we do in this thesis. Regularization also has a connection with multi-task learning. You can use multi-task learning to do regularization or use regularization to do multi-task learning in turn. In this work, actually, we can think the way that we use the future information as regularization as some kind of multi-task learning, which use soft parameter sharing between forward and backward RNN.

2.4 Related Work on Neural Machine Translation

When talking about neural machine translation, people will think it appeared recent year because of the boom of deep learning. However, the earliest similar idea about NMT maybe can trace back to the end of last century. It is amazing that Castano et al. (1997) showed an approach for machine translation quite similar to current popular NMT approach. Unfortunately, because of the constraints of the large parallel dataset and computational resources, there are few explorations about this idea later. We need to wait for 10 years to another breakthrough that uses the neural network method to machine translation.

In the first 10 years of this century, it is the data-driven approaches like phrase-based statistical machine translation (SMT) method dominant machine translation area. But at

the same time, there are some works about integrating neural method into SMT model, such as Schwenk (2007) used a neural language model to get a big improvement of SMT system. Later, some researchers try to combine more neural component for traditional SMT system (Schwenk, 2012) until the pure neural machine translation method.

In the beginning step of the pure NMT approach, researchers used convolutional model (Kalchbrenner and Blunsom, 2013) and sequence-to-sequence model without attention (Cho et al., 2014; Sutskever et al., 2014) to produce some acceptable translation result for short sentences. But, as mentioned before, the basic sequence-to-sequence model failed to translate longer sentences. Until the propose of attention mechanism, NMT finally can yield competitive results with classical SMT method (Bahdanau et al., 2014; Jean et al., 2015). Furthermore, after many techniques like byte pair encoding, dual training, and joint training with monolingual data are proposed, the NMT method becomes the new dominant method in machine translation area.

About the most recent progress, there are several very impressive works. First, one of the most interesting properties of NMT compares with the traditional SMT system is that it is easier to achieve zero-shot learning (Johnson et al., 2016). In this work, researcher achieves zero-shot learning just by using a specific language token and train several parallel data together. The model is illustrated in Figure 2.9.

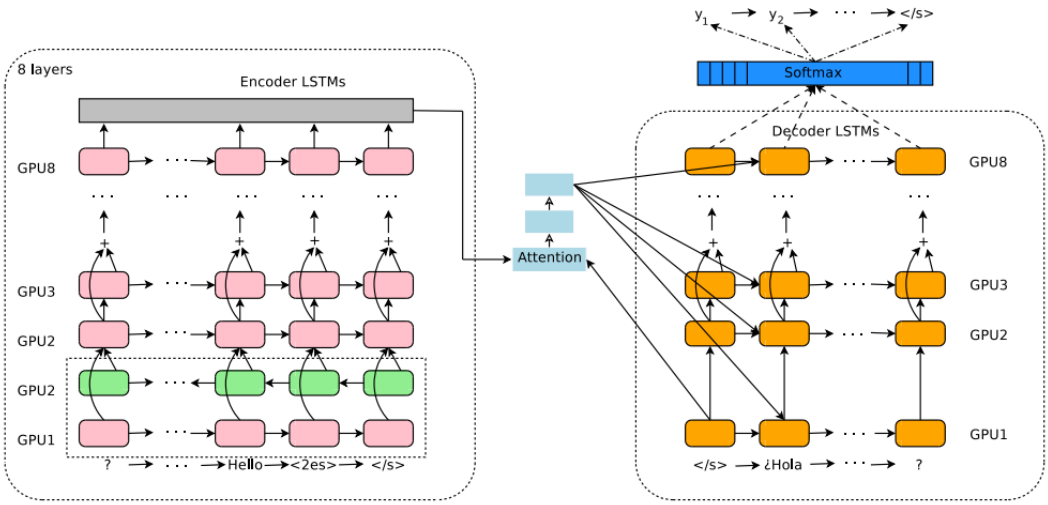


Figure 2.9: Google neural machine translation model with multi-lingual translation setting Figure from (Johnson et al., 2016). During training, we train several language pair parallel data together with a specific tag indicate the direction, for example in the picture token <2es> show the target language is Spanish.

Besides, Gehring et al. (2017) found the way to use the convolutional neural network to do sequence-to-sequence learning, which can achieve competitive performance with the RNN one. Later, Vaswani et al. (2017) from Google proposed a totally new way that without using RNN or CNN, but only attention mechanism to do machine translation. And one of most recent breakthrough is that Microsoft builds a system achieve human parity result (Hassan et al., 2018) in WMT17 Zh-En news translation dataset. The authors

applied several advanced techniques in machine translation to achieve this impressive result, these techniques are dual learning, joint training, deliberate network and agreement regularization of the left-to-right and the right-to-left model, which is the technique most related to the topic of this thesis.

One more trend in NMT area need to mention is the rise of language structure. Because the early NMT models just treat translation task as a sequence of tokens to another sequence of tokens, which indeed do not use linguistic knowledge a lot. Whereas in our common sense, we know actually language is not sequence structure but tree structure. Therefore how to use tree structure and previous linguistic knowledge to help translation become a hot topic, and we call it syntax-aware neural machine translation (Aharoni and Goldberg, 2017; Chen et al., 2017; Eriguchi et al., 2016).

Considering the using bidirectional information decoder on target language side, there are few works talk about it. Liu et al. (2016) use the method that trains two models first, left-to-right model and right-to-left model, and use the results from the right-to-left model to rerank left-to-right model results. Another Zhang et al. (2018) take the hidden states from a pretrained right-to-left model as the extra attention context to the left-to-right model.

Different with two works mentioned above, we would like to try several ways to train left-to-right and right-to-left decoder in one model, but not a separate model. So it will be more parameter efficient, and with less training time. In this research, because of our constraint time and resources, we do not aim to build a system to achieve the state-of-art performance but to explore the ways to make use of bidirectional information of target sentence and use them into standard NMT model. Later, we would like to compare the proposed model and standard baseline model.

Chapter 3

Neural Machine Translation Models

In this chapter, we would like to talk about the detail of the models that we implemented in our study. Beside of models, we also present the detail about the techniques we used in the proposed model.

3.1 RNN-based Attentional Sequence-to-Sequence Model

3.1.1 Embedding and Encoder

At first, our baseline model is implemented based on the OpenNMT, an open source NMT toolkit. The baseline model is quite similar to the one in (Bahdanau et al., 2014). In the model, we just treat the sentence as a sequence of tokens. After preprocessing, we divide words in a sentence to byte pair encoding (BPE) subword units and build a vocabulary upon these units. Later, we need feed the sentence into the encoder. But the problem is that we can not feed word directly into neural network encoder, we have to encode these token into vector first.

There are mainly two ways to encode word in the vocabulary, one is *one-hot encoding*, and another is dominant *word embedding* way. Here we will use word embedding, which is a by-product of NLP task training at the beginning because of several advantages of it. First, if we use one-hot vector only, it will be very sparse and inefficient to training, and word embedding is many compacts. What's more, word embedding can capture the semantic relationship between words, so the words with similar meaning will have the similar vectors, and this property is very useful. Last, for one-hot vector, it is fixed, but for word embedding, we can train the embeddings for our own task. Even sometimes, we can use the word embedding pretrained by other people, like famous word2vec (Mikolov et al., 2013), and Glove(Pennington et al., 2014). In this research, because we have used BPE technique, so the overlap between the vocabulary of those pretrained word embedding and our vocabulary would quite small. We do not use pretrained word embedding, but train by ourselves.

To map words into the vector, at first, we build a dictionary to map a word to an index first, later we can get word vector from *embedding table* by the corresponding index.

The procedure like:

$$\begin{aligned} i_x &= map(w_x) \\ x &= select(W_x, i_x) \end{aligned} \tag{3.1}$$

w_x means a word from source sentence, and i means index. $W_x \in \mathbf{R}^{V \times d}$ is embedding table, which is a matrix each row corresponds to a word vector with dimension d. We select the desired row after we get the index i, and get word vector x. And then we feed this vector into our encoder, the process illustrated in Figure 3.1.

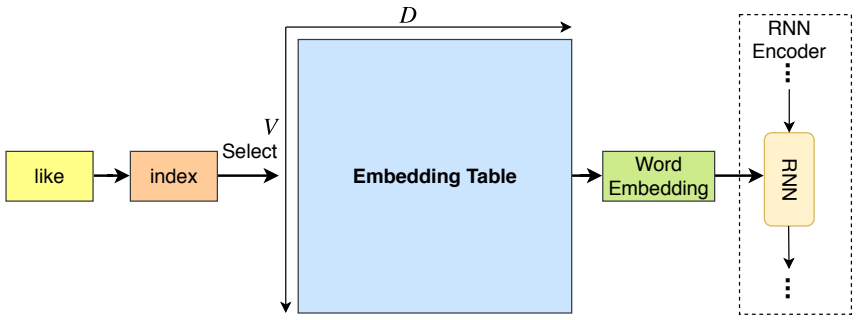


Figure 3.1: The process that how to feed a word into RNN encoder
Let's say the source side is English and we have a word *like*. To input this word into RNN encoder, we need map it to an index depend on a predefined dictionary and use this index to select the desired word embedding from embedding table. Finally, RNN encoder takes this word embedding as current input.

For encoder, we use BiRNN with LSTM cells so-called BiLSTM model to encode source sentence information. The last time step hidden states are used as the sentence embedding, which will be used as the initial hidden states for the decoder. And as mentioned before, all time step hidden states can also use to calculate the weight for attention mechanism.

3.1.2 Decoder

Decoder we also use LSTM cell as our RNN unit, and the process from the word to the LSTM unit is same as the encoder. Then after getting word embedding, we use it to calculate the hidden states of the decoder and predict target words from it:

$$\begin{aligned} i_y t &= map(w_y t) \\ y_t &= Embedding(i_y t) \\ h_t &= LSTM(h_{t-1}, y_t, c_t) \\ i'_y t &= argmax(softmax(W_g h_t + b_g)) \\ w'_y t &= map(i'_t) \end{aligned} \tag{3.2}$$

with:

- $h_t \in \mathbf{R}^r$: the hidden states at time step t. It is calculated from previous hidden states h_{t-1}, current input word embedding $y_t \in \mathbf{R}^d$ and current context vector $c_t \in \mathbf{R}^r$ from attention component.
- $W_g \in \mathbf{R}^{V \times r}$: the embedding matrix for generator to transform the hidden vector h_t to the V dimention output vector/
- $w'_{yt} \in \mathbf{R}^V$: the predicted word for current time step. It is mapped from the predicted index, which is calculated by using $argmax$ to select the index of highest probability in predicted probability vector calculated by softmax function as follow:

$$softmax(o_{ti}) = \frac{\exp(o_{ti})}{\Sigma_{j=1}^{V} \exp(o_t j)} \tag{3.3}$$

The detail of this predicting process is illustrated in Figure . Here, although we can get the highest probability without a $softmax$ layer for our generator, for the convenience to calculate the loss function later, it is better to use softmax function. For the probability that output by *softmax*, you do not need to think it as the real probability, but just something like probability and easy to calculate loss function.

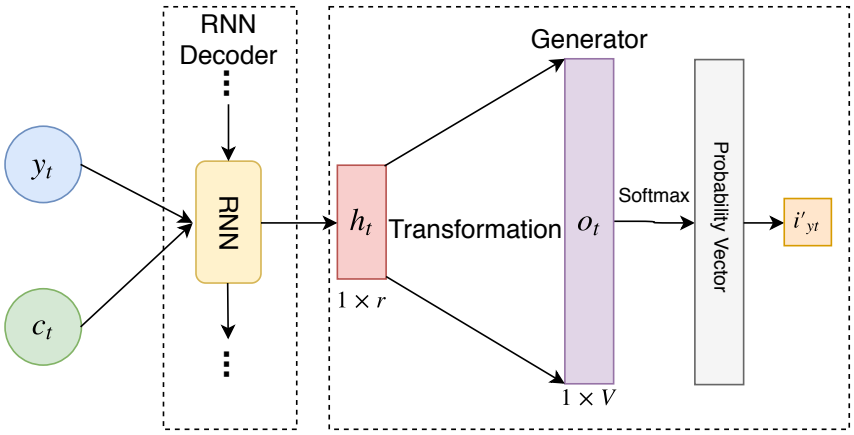


Figure 3.2: The process of how generator decoding current hidden states
Based on input and context vector of current time step and hidden states of the previous time step, the RNN unit calculates current hidden states $h_t \in \mathbf{R}^r$. Then the component we call generator in our model will at first transform hidden states from a r dimension vector to a V dimension vector o_t. Finally, *softmax* output a probability vector, and we select most possible word index.

The goal of the decoder is to maximize the log probability of correct translation sentence of target language $Y = y_1, y_2, ..., y_m$ conditioned on the correspond input sentence of source language $X = x_1, x_2, ..., x_n$:

$$\log p(Y|X) = \sum_{i=1}^{M} \log p(y_i|X, Y_{1:i-1}) \tag{3.4}$$

When we are training the model, this goal will be achieved through predict and generate a sequence of words as similar as possible with the gold translation sentence. In this way, we can define the loss function as the average categorical cross-entropy between the predicted sentence and the gold sentence.

$$\begin{aligned} H(y, y') &= -\sum_{i=1}^{V} y[i] \log y'[i] \\ H(Y, Y') &= \frac{1}{M} \sum_{i}^{M} H(y_i, y'_i) \end{aligned} \tag{3.5}$$

In the equation, for each word, we will use $y \in \mathbf{R}^V$, the gold token represented in a one-hot vector, and the predicted probability vector from softmax function to calculate the cross-entropy of this time step. $y[i]$ means the *i-th* component of the vector y. After we calculate the cross-entropy loss of each word, we sum them together and divide the length of the sentence M to take the average. That is why it named average categorical cross entropy.

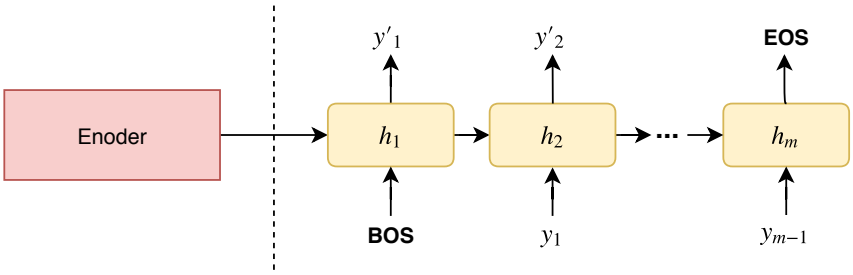
To training on the whole dataset, the objective function is the sum of the log-likelihoods of each translation pairs in the training dataset:

$$J(\theta) = \frac{1}{|D|} \sum_{(X,Y) \in D} \log p(y|x) \tag{3.6}$$

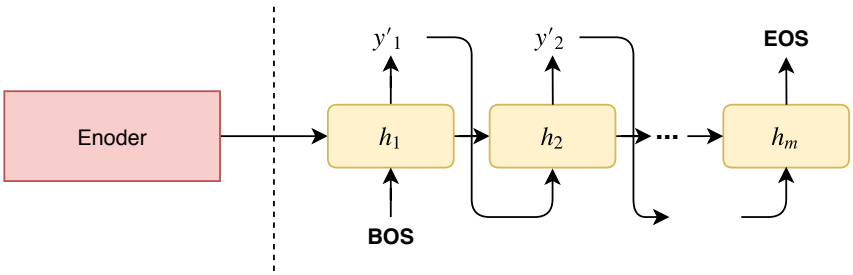
with D represent a set of parallel sentence pairs from training data. The parameter θ of this model are learned by the back-propagation algorithm.

After the object function and how to optimize it, we would like to talk the detail about the training phase and the test phase of the decoder. When we finish encoding source sentence into sentence vector, which normally set as the last hidden states of the encoder, we feed it into decoder as the initial states. And then combine it with the first input token to calculate the hidden states of the first time step. Here we need mention that the first token is a special token *<s>*, which we call it beginning of the sentence (BOS).

In training phase, we can train decoder like when we are training language model that uses the output from last time step as input for the model at current time step. However, this way has problems, such as slow convergence and model instability. To solve this, we make an assumption that previous token feed into every time step is correct, so we just use the gold token to feed into the decoder to predict the right token. This kind of training way is called *teacher forcing* because we feed gold token to predict target. It solved the training problem well. There is one variation of the teacher forcing that use the generated token from previous time step with some probability, but we only use the normal teacher forcing for simple. And in testing phase or practical using, because the gold tokens are not available, so we just use the previously generated token as input for the current time step. This process will continue until it predicts another special token *</s>*, the end of the sentence (EOS). These two phases are illustrated in Figure 3.3.



(a) In training phase, decoder inputs are gold tokens



(b) In testing phase, decoder inputs are previous predicted tokens

Figure 3.3: Different setting for decoder during training phase and test phase

3.1.3 Beam-search decoding

After we finish train our model, of course, we can just use the predicted token index with the highest value in probability vector as real predict output for each time step. However, in this way, normally we will meet the local optimal situation, so can not get a good translation result. As we know in human language, the first few words can only provide few and ambiguous information. If you decide the first several words too early, it will prevent the better output can be found later. This kind of naive decoding algorithm called *greedy search*.

There is a better way to get the predicted result and it is widely used now. It called *beam-search* method. When it is decoding, it will keep a certain number of candidates with most high probability at each time step and throw away the other prediction to increase the chance to find a better translation sentence. The constraint certain number we call it *beam size*. Normally if beam size is bigger, then the result will be better. Assume beam search is K, then we select K best partial translation. We rank the candidates by their scores assigned by a score function:

$$score(B_i) = \frac{1}{T}\sum_{t=1}^{T}\log p(w_t|X, W_{t-1}) \tag{3.7}$$

With:

- B_i: the $i-th$ beam in current step
- X: the input sentence

- w_t: the predicted token at time step t
- $W_{t-1} = [w_1, w_2, ..., w_{t-1}]$: all previous words predicted in this beam
- $\frac{1}{T}$: some kind of length normalization to make it more fair when compare candidates with different length

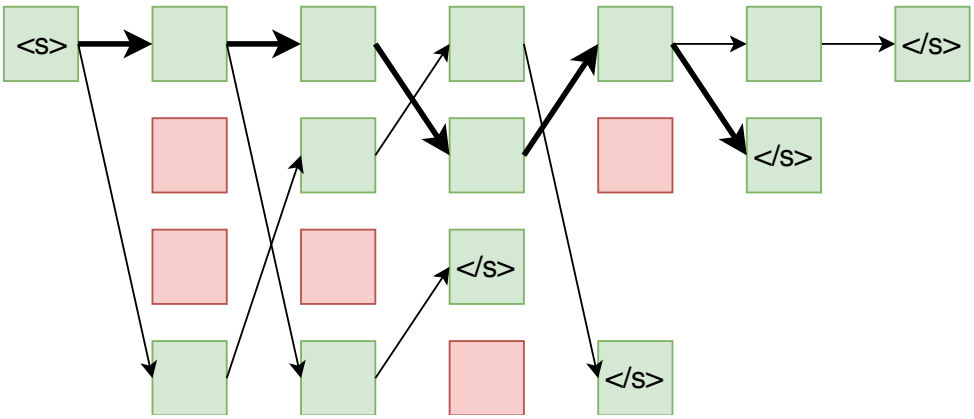


Figure 3.4: Search graph for beam search decoding in neural machine translation model At each time step, the beam size $n = 4$ partial translation with highest probability are selected. When the model predict the end of sentence token $< /s >$, an output sentence complete. We reduce one from beam size. At last, we have 4 completed sentences, and we can track down them from the arrow. The green box means the words used in the compete sentence. The red box means the candidate words have been discarded during the decoding.

Length penalty and coverage penalty

Beside of this basic beam search, in this study we use two refinements to improve beam-search decoding as (Wu et al., 2016) did. First one is the length penalty to prevent model only prefer the short sentences. Another one is the coverage penalty to favor translations that fully cover the source sentence according to the attention component. The new scoring equations are:

$$\begin{aligned} s(Y, X) &= \log(P(Y|X))/lp(Y) + cp(X; Y) \\ lp(Y) &= \frac{(5 + |Y|)^\alpha}{(5 + 1)^\alpha} \\ cp(X; Y) &= \beta * \sum_{i=1}^{|X|} \log(min(\sum_{j=1}^{|Y|}, 1.0)) \end{aligned} \tag{3.8}$$

Where:

- lp: the length penalty function

- cp: the coverage penalty function
- p_{ij}: means the attention probability of the j-th target word on the i-th source word.
- α, β: the hyper parameter to control the strength of length penalty and coverage penalty. When they are both 0, then it is the basic beam-search.

3.2 Multi-task Learning Model

This section we would like to talk about our multi-task learning models. Here we take the forward translation and backward translation as two related task. For these two tasks, our forward RNN decoder and backward RNN decoder need to capture different distribution. And they have a different objective. For forwarding translation we need to learn parameters to maximize following conditional probability:

$$P(\overrightarrow{Y}|X;\theta) = \prod_{j=1}^{M} P(y_j|Y_{[1:j-1]}, X;\theta) \tag{3.9}$$

with:

- $\overrightarrow{Y}$: the forward target sentence, $Y = [y_1, y_2, ..., y_m]$ with y the target words
- X: the source sentence
- θ: the model parameters
- $Y_{[0:j-1]}$: the target words before time step j, $[y_1, y_2, ..., y_{j-1}]$

But for backward translation we need learn parameters to maximum another probability:

$$P(\overleftarrow{Y}|X;\theta) = \prod_{j=1}^{M} P(y_j|Y_{[j+1:]}, X;\theta) \tag{3.10}$$

with:

- $\overleftarrow{Y}$: the forward target sentence, $Y = [y_1, y_2, ..., y_m]$ with y the target words
- $Y_{[j+1:]}$: the target words after time step j, as $[y_{j+1}, y_{j+2}, ..., y_m]$

So from multi-task learning setting, we need train this two task together, and we can try to share some components between them. They are the decoder *word embedding table*, the *attention component* and the *generator* that to predict tokens from hidden states. For a basic model of translation, it is illustrated in Figure 3.5.

If we want train a bidirectional decoder model and share none of three above components, the model will looks like Figure .

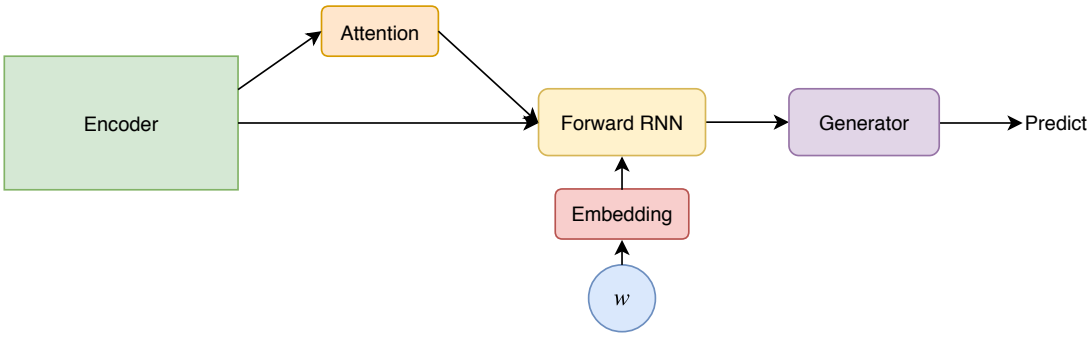


Figure 3.5: Basic model
A normal RNN based Encoder-Decoder baseline model, in which encoder encode source sentence information then feed it into attention mechanism and decoder. Decoder use sentence embedding from encoder, context vector from attention mechanism and word embedding from embedding table to produce current hidden states. Last, generate use this hidden states to calculate probability vector, and we make predict base on this vector.

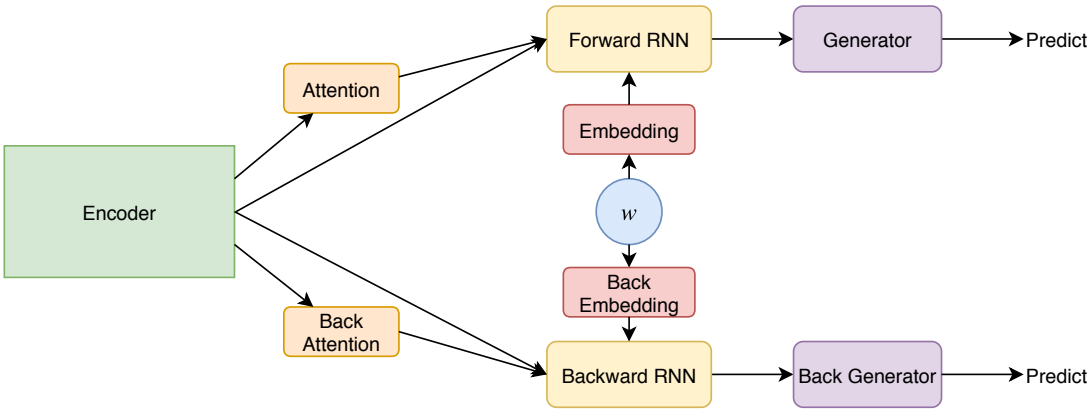


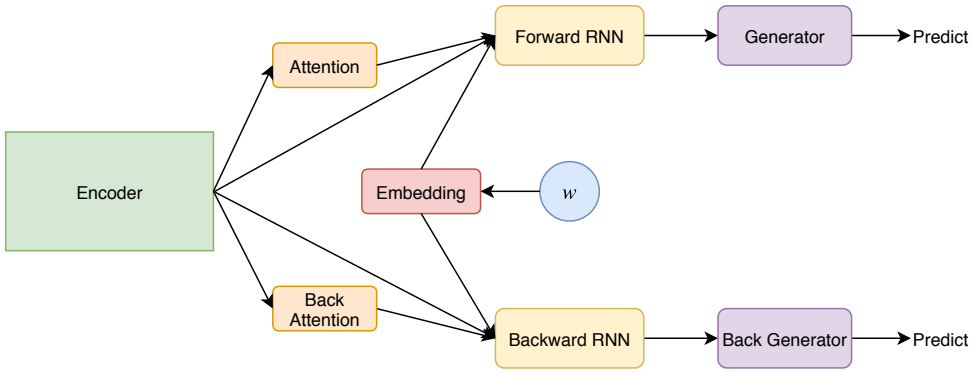
Figure 3.6: The model bidirectional decoders share no component
The core component for each direction decoder is the RNN component. In the figure, the components with the same color have the same architecture but have different parameter inside. The word *back* we put in some component just to differentiate the forward and backward components with same name and architecture.

3.2.1 Sharing decoder word embedding

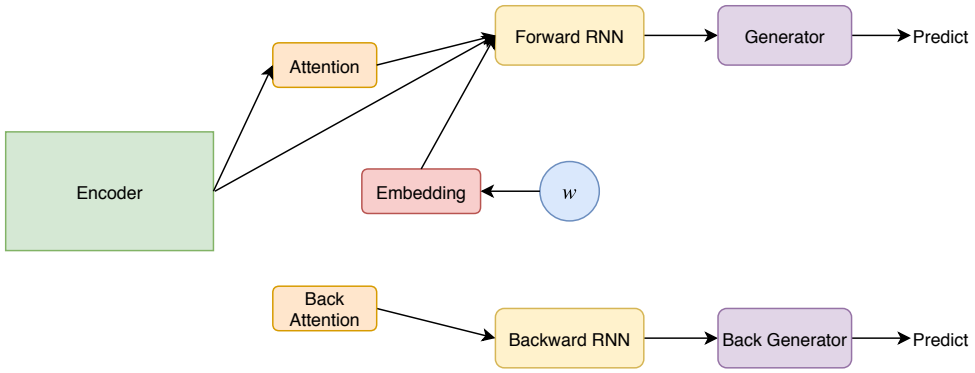
In NLP area, the first thing we can consider to share is word embedding component. Because word embedding is the most low-level layer have the elementary representation for NLP task, for almost every neural network model in NLP now, if we want to input words, we need map it to word embedding first. (Yosinski et al., 2014) showed that if we want to share representation or transfer knowledge between different two tasks, then the low-level representation is easier to share. And the widely using of pretrained word embedding Mikolov et al. (2013); Pennington et al. (2014) also support this theory, and in many tasks, it indeed boosts the task.

So at first, we try to share embedding component. And based on the loss information

from both forward and backward to update the shared embedding. The model looks like Figure 3.7.



(a) Training phase



(b) Test phase

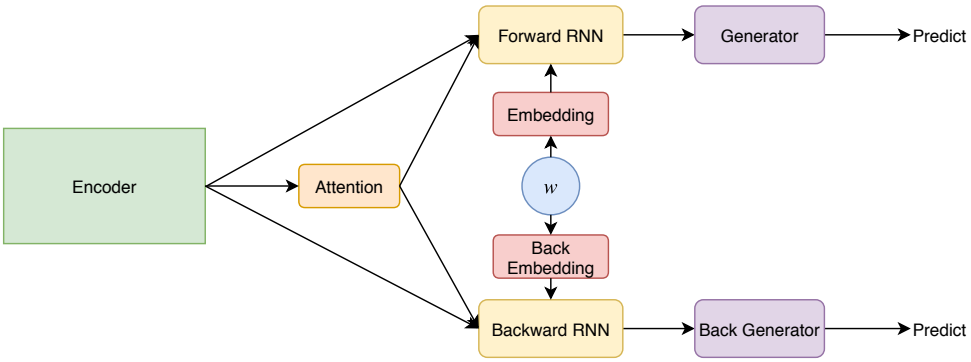
Figure 3.7: The calculate process of sharing embedding model in training and test phase

3.2.2 Sharing attention mechanism

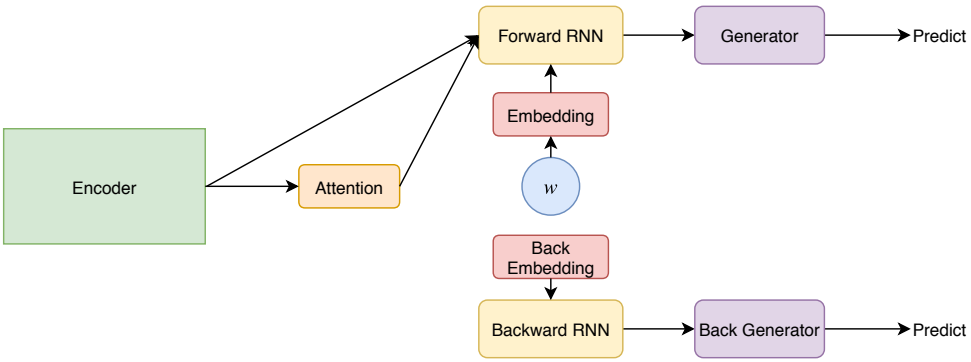
The second component we want to share is the attention mechanism, which means we input the hidden states from encoder and hidden states from both direction decoder to the same attention component. There is some work people use shared attention mechanism to achieve better performance, but for training different language task together (Firat et al., 2016).

The intuition here is that for attention mechanism, the main function is to find the alignment between the target sentence and the source sentence. No matter our model translate from front or back, for a certain sentence pair, its sentence alignment should be same. Therefore if we train them together, this two task should boost each other through shared attention.

This model is illustrated like Figure 3.8.



(a) Training phase



(b) Test phase

Figure 3.8: The calculate process of sharing attention model in training and test phase

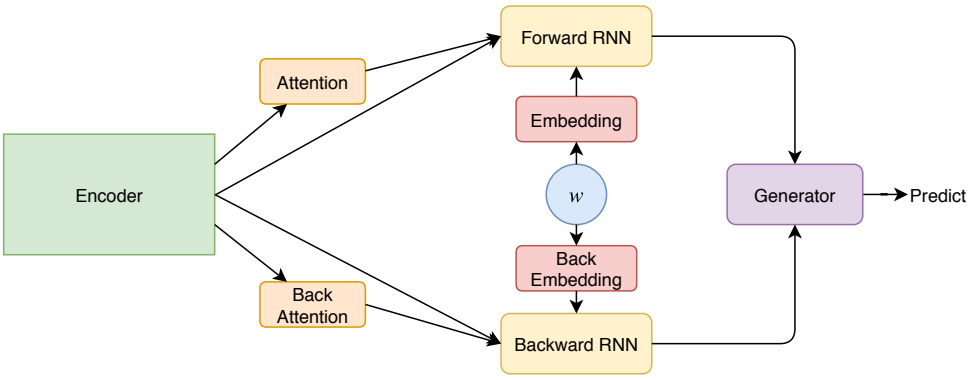
3.2.3 Sharing generator

Finally, we would share the generator component in our model. The generator component in our model mainly have two functions, one transforms the hidden states to vocabulary size V dimension output vector, another one is used softmax to output the probability vector from output vector. The intuition to share generator is that when we predict the gold token from two direction RNN, they have same gold tokens. But the hidden states from the different direction generally have different hidden states even for the same time step. After we share generator, then to predict the same gold token, the hidden states from both direction should have similar tensor value. In this way, we can think share generator as some kind of regularization to make forward hidden states and backward hidden states in the time step to become similar.

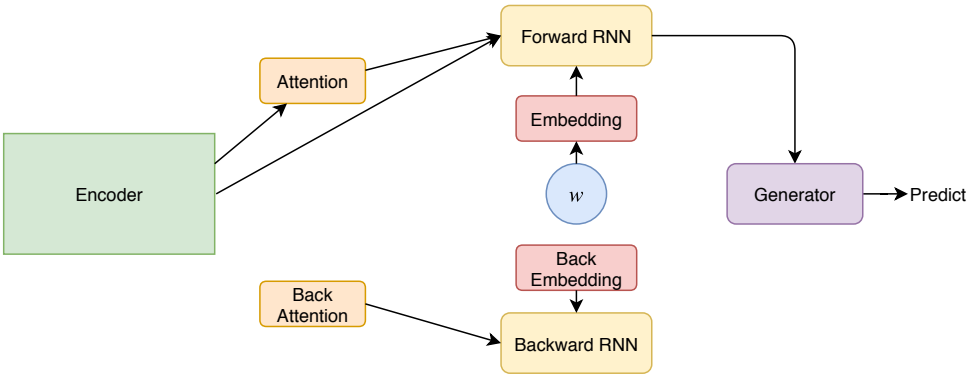
The model illustrated as Figure 3.9.

3.3 Regularization from Backward RNN

If we have two directions RNN in the decoder, beside of sharing component, another way we can get information from backward RNN is to use it as regularization for forwarding RNN. x_t is the prediction word for the current time step, from above we mentioned, we



(a) Training phase



(b) Test phase

Figure 3.9: The calculate process of sharing generator model in training and test phase

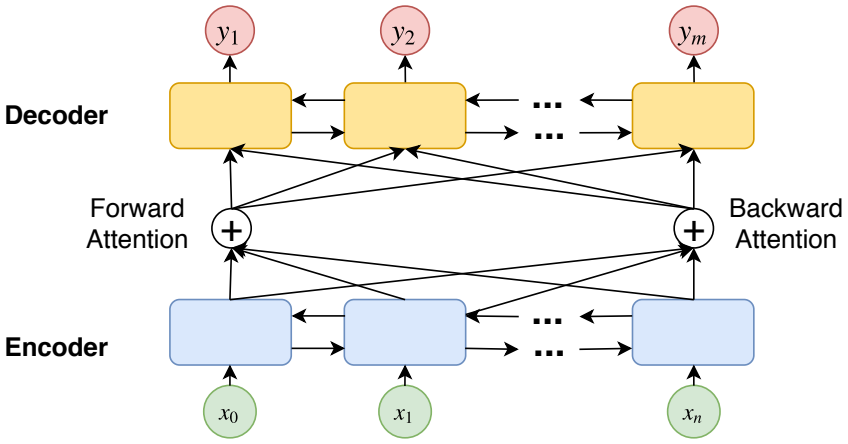
calculate if by the generator from the current hidden states h_t^f. The basic idea here is to promote the forward hidden states h_t^f contain more useful information to predict x_i. Here, the extra information we want to take is from the backward RNN, which means the information from the future of the sentence. So we need use loss function to build a connection between forwarding and backward hidden states, as illustrated in Figure 3.10b. The overview of proposed regularization model is shown as Figure 3.10a.

A naive way to do this regularization is just used $L2$ regularization loss to punish the difference between the forward and backward hidden states:

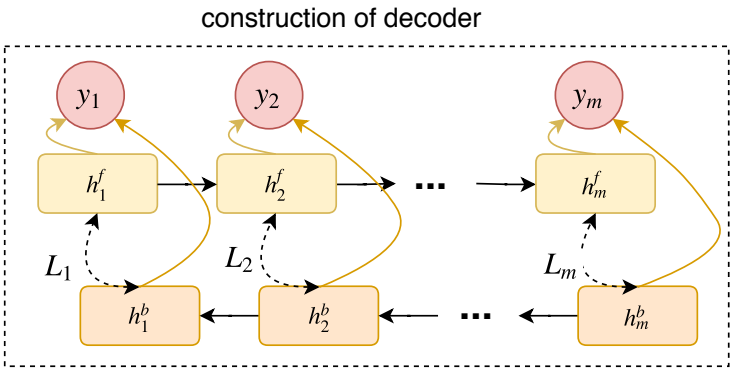
$$L_t(x) = d_t(h_t^f, h_t^b) \tag{3.11}$$

with:

- d_t: the $L2$ loss function
- h_t^f: forward hidden states
- h_t^b: backward hidden states



(a) Overview of the proposed regularization model



(b) Decoder with L2 regularization between forward and backward hidden states

Figure 3.10: (a) is the overview of proposed model. The decoder consists of forward decoder and backward decoder. They use different attention components, and share the same encoder. (b) shows the detail of the decoder. Regularization loss L is applied between forward and backward hidden states.

However, this loss is a little too strict and make the model generate forward and backward hidden states less flexible. We can add a simple affine transformation to our forward hidden states first, then calculate the $L2$ loss with backward hidden states:

$$L_t(x) = d_t(g(h_t^f), h_t^b) \tag{3.12}$$

with g the affine transformation function. We can use a linear layer to do this.

Chapter 4

Evaluation

This chapter we first introduce the statistics of the dataset and the preprocessing on it. Next, the detail experimental setting is be show carefully. Last, we present the results of different experimental settings, include the translation result, and evaluation metrics like Bilingual Evaluation Understudy Score (BLEU) score.

4.1 Dataset and Preprocessing

The dataset we used in this research is the open accessible machine translation parallel datasets from WMT16 and WMT17. Especially, we will use the DE-EN (Germany-English) and ZH-EN (Chinese-English) parallel dataset from the machine translation of news shared the task. In this research, because of the constraints of our time and computational resources, and to make the experimental iteration faster. I make two DE-EN datasets from the whole WMT16 DE-EN dataset, the big one contains sentences from three corpora: Europarl V7 (Koehn, 2005), Common Crawl corpus, and News Commentary (NC) v11; the small dataset only contains the NC v11 corpus. And one ZH-EN dataset is from the News Commentary v12 dataset.

This parallel dataset includes source language sentence and corresponds target language translation. For DE-EN translation tasks, we use the big and small dataset mentioned above as the training set, and for validation and test dataset, we use the test data in WMT16 and WMT15 and test data from WMT previous years. In detail, we use *newstest2014* as validation set, use *newstest2015* (test1) and *newstest2016* (test2) as two test datasets. For ZH-EN translation tasks, we use the NC v12 as training set. The validation and test set same as WMT17 competition that *newsdev2017* as validation set and *newstest2017* as test set. The detail statistic about train, valid and test set in Table 4.1 and Table 4.2.

For DE-EN task, after downloading these corpora from WMT16 website, we concatenate these three corpora to make the big dataset. Then we use preprocess tools from Moses statistical machine translation system [1] to do the first step of preprocessing. The

[1] https://github.com/moses-smt/mosesdecoder/tree/master/scripts

Table 4.1: Number of sentences of the DE-EN datasets

	Training	**Validation**	**newstest2015**	**newstest2016**
Big (WMT DE-EN full)	4215806	3003	2069	2999
Small (NC v11 DE-EN)	242770	3003	2069	2999

Table 4.2: Number of sentences of the ZH-EN datasets

	Training	**Validation**	**newstest2017**
NC v12 ZH-EN	227330	2002	2001

pipeline is normalized punctuation in different languages, tokenize sentences, clean empty and long sentences (we throw the sentence longer than 80 tokens), and apply true case process, which means initial word of a sentence is converted to their most probable case (The boy laugh. -> the boy laugh. I love you -> I love you). The second step, we concatenate German and English corpora together and learn BPE of it (Sennrich et al., 2016). The reason why we concatenate two corpora is to improve the consistency in the segmentation of the source and target text. The BPE merge operation we used here is 32000 for the big dataset and 8000 for the small dataset.

After the BPE is learned, we apply it to our parallel data. The last step, we will add several special tokens into our vocabulary. For target sentence, we add begin of sentence (BOS) token <*s*> in the begin of every sentence, and append end of sentence (EOS) token </s> at the end of every sentence. Besides, we also use the unknown word token <*unk*> to replace the tokens under 1 frequency and the word not in vocabulary when testing. For the reason of parallel training, we need train parallel sentence pairs in batch. However, RNN can only parallel handle sentences have the same length. so we have another special token called pad token <*blank*> to pad short sentence in a batch to the same length as the longest sentence in this batch.

For ZH-EN task, first different with English and German, there is no space between the words in Chinese. So we need segment the Chinese sentences first. Here we use the *jieba* [2] Chinese segmentation tool to segment Chinese data first. And later tokenize English, clean long sentence, and apply true case for English dataset. In terms of BPE, we also only do it in English side, and use 8000 merge operations.

4.2 Experimental Settings

For experimental settings, most of the parameter settings we reference from previous studies (Britz et al., 2017), because of the extensive time and computation resource will cost if we want to search a good hyperparameter setting by ourselves. Also one of the reasons why we don't do carefully hyperparameter searching is that the purpose of this research is not to achieve state-of-art performance, but ways to use bidirectional information of

[2]https://github.com/fxsjy/jieba

target sentence in machine translation and to see it can improve the basic model or not.

At first for vocabulary, because we learn BPE from the concatenate of two corpora, so they share a lot of vocabulary and have limited vocabulary size based on the BPE merge operations. The vocabulary size of both encoder and decoder are 37004 for the big dataset, and 8496 for the small dataset. Beside of this, there 4 more special tokens, as $< s >$, $< /s >$, $< unk >$ and $< blank >$.

And other configurations about our network are listed in Table 4.3. We take batch size 80 as mentioned in (Sennrich et al., 2016). And as (Britz et al., 2017) suggested, 512 for embedding dimension, 512 RNN hidden size, LSTM for RNN unit, 512 attention dimension, Bahdanau attention type and beam size 8. And for the layer of RNN, we choose 4 for the big dataset, and 2 for the small dataset. Beside of this, for the optimizer, we choose Adam because of its efficiency and easy to use property. We set initial learning rate as 0.001, and have a *learning rate decay*, which will decay the learning rate when the validation perplexity does not decrease and epoch has pass certain epoch. For the maximum generated sentence size, we set it as 100. Last, for the length penalty and coverage penalty parameter, we set both to 0.33 based on experimental we did.

Table 4.3: The hyper parameter setting for our experiment

Hyper Parameters	**Settings**
Batch size	80
Word embedding size	512
RNN type	LSTM
RNN hidden states	512
Encoder layers	4 (2 for small dataset)
Decoder layers	4 (2 for small dataset)
Attention size	512
Attention type	Bahdanau
Optimizer	Adam
Learning rate	0.001
Beam size	8
Length penalty	0.33
Coverage penalty	0.33
Max output length	100

Every epoch we will randomly shuffle our data and make a batch to train our model. This is a small trick to prevent over-fitting because if don't shuffle it the model will remember the repeat pattern. Another trick to prevent over-fitting is to use dropout between layers. In this research, we use default dropout rate 0.3 from OpenNMT toolkit. All the models are implemented by Pytorch, the Python version of the popular deep learning framework Torch based on Lua language originally.

In practical for training this kind of big model on such a big dataset, it needs a lot of parallel computation to train it fast, which means we need a computer with powerful

GPU. Luckily, we get the chance to use the server called *Tetsuzen (Iron Mountain)* bought recently in our lab. This server has 4 GeForce GTX 1080 GPU cards, and each of them has near 8 GB memory. It makes training of my big model come true, because if you train it on the CPU maybe it will cost several months. However, it still took quite a long time to train one epoch, especially for the model trained on the big dataset, every epoch it needs take several hours. In our experiment, we use 15 epochs as the standard epochs number for training models.

4.3 Results

To evaluate the quality of a machine translation system, it is quite ambiguous. That is why there are many metrics to evaluate machine translation systems, such as BLEU (Papineni et al., 2002), NIST (named after the US National Institute of Standards and Technology) (Doddington, 2002) and translation error rate (TER) (Snover et al., 2006). Sometimes even these metrics are not enough, we need human evaluation for some systems. Because of the convenience of evaluation tool *mteval-v13a.pl* [3] from Moses SMT system, we use NIST and BLEU score as our evaluation metrics. And beside of these two metrics, we can get accuracy and perplexity on the train and validate sets for some kind of reference to the performance of our system. So in this research, we will report BLEU, NIST, accuracy and perplexity for the valid set in each run. The definition of these four evaluation metrics are as follow:

- **BLEU**: it is computed by n-grams overlap precision between machine translation output hypotheses and reference gold translation, and it is the most popular evaluation metrics in machine translation and many other NLP tasks. Higher is better. For ZH-EN task, first, we separate all Chinese into character by *tokenizeChinese.py* [4], later calculate BLEU score based on these characters.

- **NIST**: it is a similar metric as BLEU to evaluate translation result, but it differs from the way to calculate the weight for each n-gram overlap and the way to calculate brevity penalty. Higher is better.

- **Accuracy**: it is calculated by the number of correct predict words divide total words number. Accuracy is also higher better for a model.

- **Perplexity**: it is a measurement of how well a model predicts a sample. A lower perplexity indicates the probability model is better at predicting the sample. It is calculated by the exponent of the per-word negative log probability, so can calculate it just take the exponent of loss divide words number

We run following experiments to compare the baseline system and the proposed systems:

[3] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/mteval-v13a.pl
[4] http://statmt.org/wmt17/tokenizeChinese.py

- Experiment with the baseline model: based on the recommend parameter settings, we build a strong baseline on big DE-EN dataset through exploring few parameters, such as beam search width, BPE merge operation numbers. The evaluation way is to compare it with the state-of-art machine translation model on the same dataset in terms of BLEU score.

- Experiments with the multi-task learning models: through sharing different component, we have several proposed models, like model share attention, model share embedding, or we can share multiply component at the same time. Because of the constraints of resources, so we run different setting experiments on small DE-EN dataset.

- Experiments with the bidirectional regularization model: this part we will explore different settings of regularization on our bidirectional RNN decoder, such as the direct *L2* loss and affine transformation added *L2* loss. We use an loss weight annealing technique to train our model.

- Experiments with models combined above two techniques: this part we take a best setting from the multi-task learning sharing component model, and apply two kinds of regularization upon it to build a combined model. We implement the proposed models and compare them with baseline and proposed models in former experiments.

- Experiments on more language direction and the large dataset: this part we select several good performance proposed models, and training on the ZH-EN dataset and check its performance. Then we would also run models on the big DE-EN dataset to get significant evidence of our proposed model.

4.3.1 Experiment 1: Searching a strong baseline model

In this experiment, first, we build a baseline model based on OpenNMT-py open-source NMT code based on Pytorch deep learning framework[5]. Because for a task like machine translation, it is better to reference the previously successful model to your own model. So in the beginning, we try to follow settings from some very successful models, like Edinburgh neural machine translation system (Sennrich et al., 2016), which is implemented by nematus, an NMT open-source toolkit based on Theano framework.

However, after we follow their setting, we find our model become too big to train on our machine because of its big vocabulary size 89500 and big encoder and decoder parameter number. Later, we decrease hidden size from 1024 to 512 and decrease RNN layer from 8 to 4. This time we successfully trained our model, but the result is quite bad when compared with the state-of-art on WMT16 German to English task as Table 4.4.

The reason why this result is so bad maybe the difference between two NMT toolkits or wrong is preprocess setting. Later, we try to follow setting as Britz et al. (2017) did. The main contribution of this paper I think is that we decide not to follow the BPE

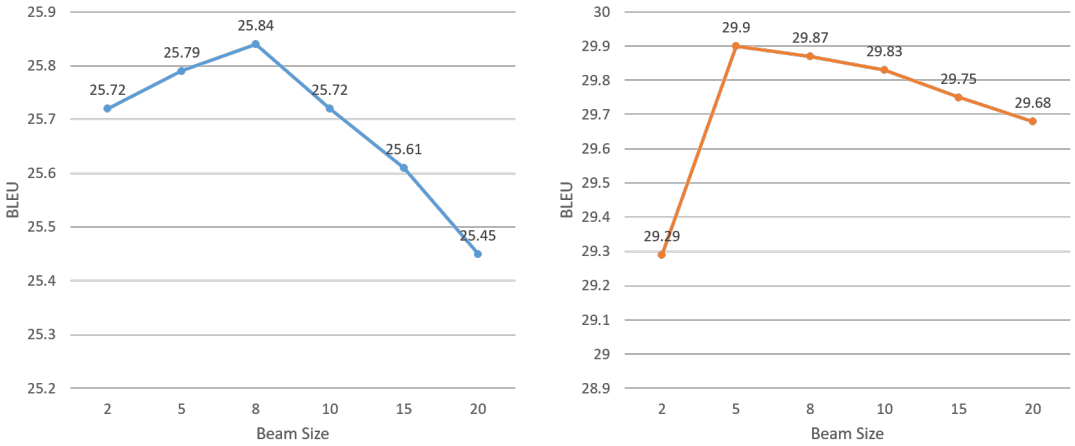
[5]https://github.com/OpenNMT/OpenNMT-py

Table 4.4: Trial baseline model translation results of WMT16 EN→DE in terms of BLEU score compare with state-of-art baseline model on big dataset (Sennrich et al., 2016)

	newstest2015	**Validation**
Baseline(trial)	16.77	18.04
Edinburgh	26.4	28.5

merge operation times 89500, but to use the less setting 32000. So our model gets a large improvement. Other parameter settings just as Section 4.2.

And later to make our translation result better, we also did some exploration of the beam size. When we are training our models, we save the model to hard disk each epoch. Then after observing the accuracy and perplexity on valid set, we select the best model for the baseline model. And use different beam size to generate our translation. When we take beam sizes: 2, 5, 8, 10, 15 and 20, the BLEU score change like Figure 4.1.

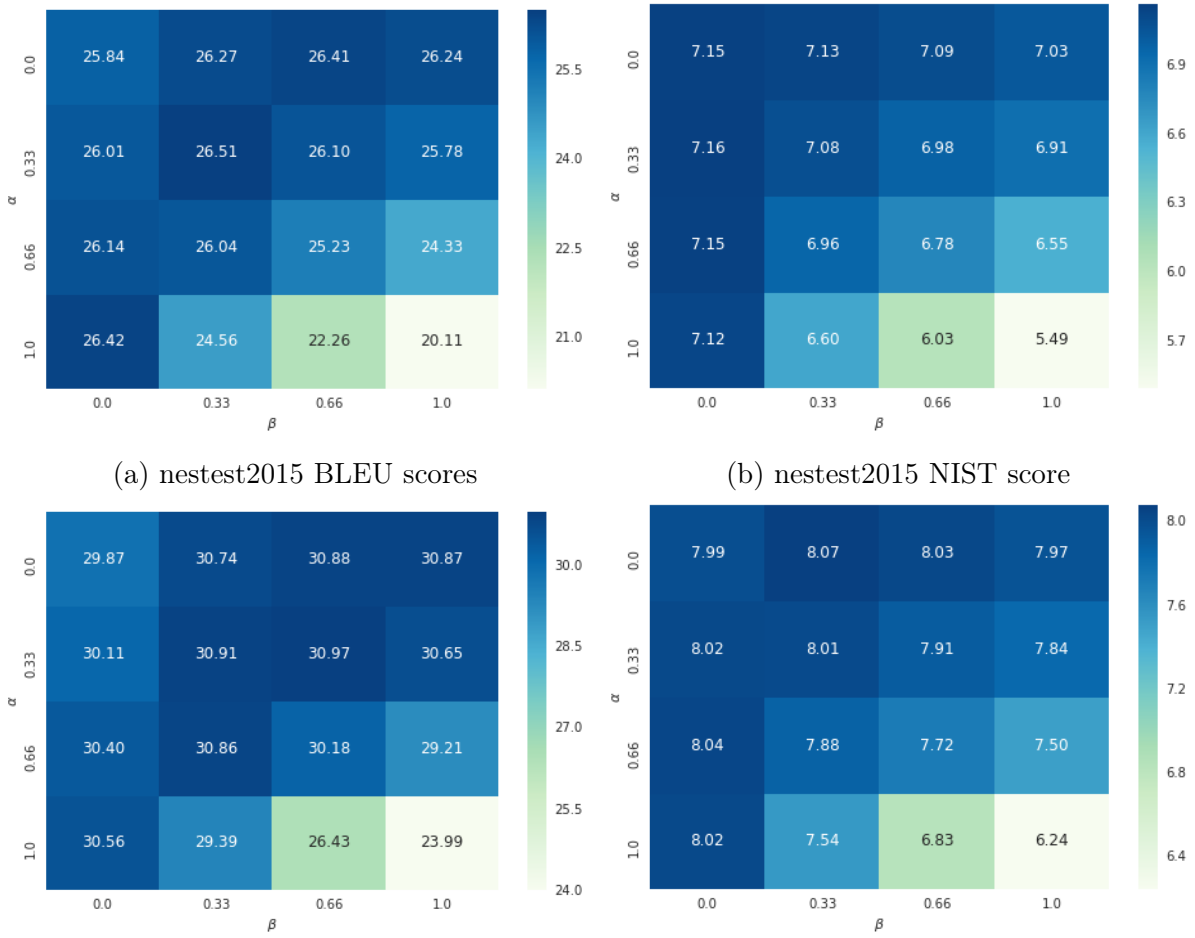


(a) Result on newstest2015 (b) Result on newstest2016

Figure 4.1: The influence of different beam size to result

Although from the intuition, increasing the beam size can make the decoding process explore more possible translations. In this way, there is also more chance to find a translation with higher model scores. However, from the experiment we find to increase the beam size does not improve translation BLEU score consistently. It reaches the peak among 5-10, and later decrease. This finding also consistent with what Koehn and Knowles (2017) find. The main reason for this phenomenon maybe wider beams makes shorter translations.

Beside of beam size, we also explore length and coverage penalty related hyperparameters, α and β. Both of them will take the range from 0 to 1. So we choose one of the beam size 8 from above and fix it, later try to modify length and coverage penalty parameters and to see the influence to our translation result. We can find the result in Figure 4.2.



(a) nestest2015 BLEU scores

(b) nestest2015 NIST score

(c) nestest2016 BLEU score

(d) nestest2016 NIST score

Figure 4.2: WMT16 En→DE NIST and BLEU scores on *newstest2015* with respect different length penalty parameter α value and coverage penalty parameter β value

When the beam search function purely depends on the sequence probability (α and β are 0), the BLEU score result on the test set is same as we show in beam size experiments. In terms of BLEU score, we find larger α can help improve BLEU score a lot, like get $+0.58$ score ($\alpha = 1.0$, $\beta = 0.0$). And for β, it have some slightly large value can help. However, when we combine them together, we find that it achieves some sweet point in the low range of both two parameters, but not in the high range. Indeed, if you set both parameters big, it achieves a very worse result with -5.73 BLEU score compare with basic beam search model. On the other hand, if we look at NIST score figure, we find that actually, these two parameters help a little. Because we take BLEU score as our main metrics, so later in our experiment, we will take $\alpha = 0.33$ and $\beta = 0.33$ as the default setting.

After explore on beam size and length and coverage penalty, we can get a strong baseline for WMT16 English to German model now. Here we use English to German

direction because there is more comparable state-of-art models' result. These state-of-art systems are:

Table 4.5: Comparison to Moses, BPE (Jean et al., 2014), BPE-Char (Jean et al., 2014), RNNSearch (Bahdanau et al., 2014), Deep-Conv (Gehring et al., 2016), Mass-EnDe (Britz et al., 2017) in terms of BLEU score one *newstest2015*

Model	**newstest2015**
Ours	**25.84**
Ours + *lcp* (with penalty)	**26.51**
Moses	20.54
BPE	20.50
BPE-Char	23.90
RNNSearch	24.88
Deep-Conv	24.30
Mass-EnDe	25.23

We can see the initial setting and some exploration, we get a very strong baseline model. It can get 25.84 BLEU score on WMT16 En→DE task in *newstest2015* test set, and even if we add proper length and coverage penalty we can even reach 26.51, which is a very high result.

4.3.2 Experiment 2: The influence of sharing components

In this section, we try to explore different settings for our multi-task learning models as mentioned before, for example sharing embedding, attention or generator. Sometimes we will even share multiple components to see is there any further improvements. The results of running each model listed as Table 4.6 and Table 4.7. We will show accuracy and perplexity of valid set in the first table, and BLEU and NIST scores on two test sets in the second table.

Table 4.6: Comparison of various proposed multi-task learning model and base model on EN-DE News Commentary v11 in terms of accuracy and perplexity of valid data *newstest2014*

	Accuracy (%)	**Perplexity**
NC-base	54.20	18.13
NC-share-embed	55.62	15.68
NC-share-atten	54.95	16.22
NC-share-gen	55.69	15.98
NC-share-embed-gen	55.61	15.12
NC-share-all	54.55	16.96

For the representation of our models in the table, *NC* means the model training on

the small dataset *News Commentary v11*. After the bar, we append several words, for example, *share* means the sharing component model. And after it, we have *embed*, *atten* and *gen*, they represent embedding, attention and generator component respectively. Except for these sharing single component model, we also have models are sharing several components. Because we want to see is there any more improvement space for our model after we sharing several components together. Therefore, *share-embed-gen* means share both embedding and generator components, and *share-all* means share all three components.

After the explanation of model representation, we can look at Table 4.6 more detail. We can find almost every proposed model get better accuracy and perplexity, for sharing single component model, especially *NC-share-embed* and *NC-share-gen* get a big improvement. So we also run a model *NC-share-embed-gen* to see the model can be better or not after we combine two good sharing components. And we find indeed it get improvement especially in terms of perplexity. However, the problem here is that sometimes accuracy and perplexity have not so big relation with the real translation quality, so it is better to just take it as a reference. BLEU and NIST score will better to evaluate a real translation result.

Table 4.7: Comparison of various proposed multi-task learning model and base model on EN-DE News Commentary v11 in terms of NIST and BLEU score

	newstest2015		**newstest2016**	
	NIST	BLEU	NIST	BLEU
NC-base	5.7381	17.64	6.3912	20.43
NC-share-embed	5.8725	17.99	6.4530	20.36
NC-share-atten	5.8677	18.10	6.3992	20.03
NC-share-gen	**5.9872**	**18.59**	6.4726	20.38
NC-share-embed-gen	5.9317	18.24	**6.5523**	**20.81**
NC-share-all	5.8994	17.86	6.3971	19.89

From Table 4.7, we can see clearly the comparison between the models in terms of NIST and BLEU scores. We can see from NIST score, all proposed multi-task learning model get improvement. Especially for *NC-share-gen* and *NC-share-embed-gen* two models, the former get highest on *newstest2015* data and the late has highest score on *newstest2016* data. What's more, they get a consistent result in terms of BLEU score. Later, we can take these two as most promised models in our proposed multi-task learning model. If we look at other models, we can find in in terms of BLEU score *NC-share-embed*, *NC-share-atten* and *NC-share-all* these three cannot beat baseline *NC-base* on *newstest2016* set. Particularly, for sharing all components model, it does not get a better result as we expect. At first, we think we share all components together and each sharing single component model can improve respectively, so maybe it can help each other by combining them, but the result is bad. One interesting observation here is that why NC-share-gen can get a better result than other two. From our point of view, we think the following result may contribute to this phenomenon:

- For embedding shared model, it may be that the dataset is a little too small. Because as we know, if we want to get a good word embedding, normally we need train the word embedding on a lot of data, for example, the wide used Glove word embedding is trained on several billion tokens. Therefore, we believe if we improve data size it can get a better result.

- For attention shared model, it may be that the task for it is a little too complex to learn. Because as we know, attention mechanism actually is doing some kind of alignment between the source sentence and the target sentences. We know no matter the forward translation or backward translation, at least they are from the same sentence pair then the alignment they need to learn is same. However, that means if the attention mechanism is same, to learn the similar alignment the hidden states on the same word need be similar. It sounds like the generator shred model, but the difference is that this alignment process is quite far from the last loss function calculation, so we think maybe there is no direct way to pass enough information back and update properly. Even it will make a lot of noise to mislead training process, this may explain why share all model perform badly.

- For generator shared model, it also needs force forward and backward hidden states become same as attention shared model. But it can get direct update signal from loss function, so train more effectively.

Besides of comparison between those metrics, we can also make some deeper analysis by using human evaluation. We take some training examples of *newstest2015* translation results from baseline model *NC-base* and the strongest proposed multi-task learning model *NC-share-embed-gen*. And we compare their translation result with each other and with gold translation to see the translation quality of each model. The translation comparison as Figure 4.3

We can see the example sentence we choose in Figure 4.3 is quite long, almost 40 tokens if we include punctuations after tokenization. The reason why we choose this is that translation of long sentence is easier to see the difference between translation result from different systems. We can find that the baseline model loss more information in its translation than the result from our proposed model. Especially, the *NC-base* model loss a big part of information like "meeting his Japanese counterpart, Shinzo Abe" and "in Tokyo to discuss economic and security ties". Compare with this, the proposed *NC-share-embed-gen* model capture this information very well, because it can have more information from backward to help translation.

4.3.3 Experiment 3: The influence of regularization from backward

Beside of the multi-task learning way that making a connection between both direction by sharing components, we can also add such kind of connection through regularization method. By adding some constraints, we can use information from backward RNN or

Source sentence:
Indiens neuer Premierminister Narendra Modi trifft bei seinem ersten wichtigen Auslandsbesuch seit seinem Wahlsieg im Mai seinen japanischen Amtskollegen Shinzo Abe in Toko, um wirtschaftliche und sicherheitspolitische Beziehungen zu besprechen.

Reference sentence:
India's new prime minister, Narendra Modi, is meeting his Japanese counterpart, Shinzo Abe, in Tokyo to discuss economic and security ties, on his first major foreign visit since winning May's election.

***NC-base* translation:**
India's new prime minister, Narendra Modi, applies to his first major visit to his first major foreign visit since his election victory in May.

***NC-share-embed-gen* translation:**
India's new Prime Minister Narendra Modi is true in his first major foreign visit, Shinzo Abe in Toko, for his Japanese counterparts in May to discuss economic and security relations.

Figure 4.3: The comparison among source sentence, gold reference sentence, *NC-base* translation and *NC-share-embed-gen* translation

future information to boost forward translation. The most direct way is to make forward and backward RNN hidden states in same time step similar to use some regularization.

The simplest function we can use here for regularization is just used Least Squares (L2 norm), which calculated by one vector subtract another vector and take the square, then sum all element. In this experiment, we have two kinds of models use L2 norm to do regularization. One is the model we take L2 norm of forwarding and backward hidden states as regularization loss, which we denote it as *NC-l2-direct*. Because if we take L2 regularization loss direct, it will become too strict and do not allow enough flexibility for our model to generate even slightly different hidden states from forward and backward (Serdyuk et al., 2017). Therefore, we try to add a transformation layer between forward and backward hidden states. That means we do a simple affine transformation of forward hidden states and use the generated vector to calculate L2 regularization loss with backward hidden states. In this way, we will have more flexibility for our forward and backward hidden states. This model we denote it as *NC-l2-affine*. Same as above, *NC* means small dataset *News Commentary v11*.

When we training our model, we can set the weight for our l2 regularization model. We have to choose a good value for this hyperparameters, after several experiments, we find the small value like 0.3 will be better. And to let the forward and backward RNN learn

useful information for a while, we use anneal technique, in which we start weight from 0.0 and add 0.00001 every iteration until it comes to its max value 0.3. The experiment result as Table 4.8 and Table 4.9.

Table 4.8: Comparison of various proposed regularization model and base model on EN-DE News Commentary v11 in terms of accuracy and perplexity of valid data *newstest2014*

	Accuracy (%)	**Perplexity**
NC-base	54.20	18.13
NC-l2-direct	54.75	15.88
NC-l2-affine	55.65	15.48

We can see in terms of accuracy and perplexity, both of our proposed models is better than baseline model. And as we expect, *NC-l2-affine* has better performance than *NC-l2-affine* model. But it is better we also check the BLEU and NIST scores.

Table 4.9: Comparison of various proposed regularization model and base model on News Commentary v11 in terms of NIST and BLEU score

	newstest2015		**newstest2016**	
	NIST	BLEU	NIST	BLEU
NC-base	5.7381	17.64	6.3912	20.43
NC-l2-direct	6.0193	18.69	6.4968	20.34
NC-l2-affine	**6.0193**	**18.63**	**6.6041**	**20.85**

From NIST score, we can still see both proposed model are better than baseline model, but in terms of BLEU score *NC-l2-direct* is a little worse than baseline on *newstest2016* set. For model *NC-l2-affine*, although it has similar scores on *newstest2015* set, it has much better performance on *newstest2016* set. Above result shows the proposed model has improvement comparing to baseline model.

And we use the same sentence as experiment 2 to analyze the real translation result. The translation comparison as Figure 4.4.

We can see the regularization models also captured more information than the baseline model. However, when we look at *NC-l2-direct*, it seems to have more information, but its translation fluency is quite bad. It is hard understood than other two models. So we think here *NC-l2-affine* is a better model.

When we compare regularization model with multi-task learning model, we find that the regularization model especially *NC-l2-affine* model is even better than the best sharing component model. The reason may be that regularization is a more direct way to pass information between backward and forward. However, the problem for regularization model is that we need to find a proper weight for our regularization loss, otherwise the performance will be worse. For small dataset, this kind of parameter tuning is still okay, but for the big dataset, it will be a problem, because every experiment needs cost a lot of time.

Source sentence:
Indiens neuer Premierminister Narendra Modi trifft bei seinem ersten wichtigen Auslandsbesuch seit seinem Wahlsieg im Mai seinen japanischen Amtskollegen Shinzo Abe in Toko, um wirtschaftliche und sicherheitspolitische Beziehungen zu besprechen.

Reference sentence:
India's new prime minister, Narendra Modi, is meeting his Japanese counterpart, Shinzo Abe, in Tokyo to discuss economic and security ties, on his first major foreign visit since winning May's election.

***NC-base* translation:**
India's new prime minister, Narendra Modi, applies to his first major visit to his first major foreign visit since his election victory in May.

***NC-l2-direct* translation:**
India's new Prime Minister Narendra Modi, in his first major foreign visit since his victory in May, Shinzo Abe, in Too Abe, is in Too to discuss economic and security relations.

***NC-l2-affine* translation:**
India's new prime minister, Narendra Modi, has seen his Japanese counterpart in Too Shinzo, in his first major foreign trip in May, to discuss economic and security relations.

Figure 4.4: The comparison among source sentence, gold reference sentence, *NC-base* translation and *NC-share-embed-gen* translation

4.3.4 Experiment 4: The influence of combine sharing components and regularization

After experiments above mentioned two techniques, multi-task learning model, and regularization model, it naturally comes to our mind that can we improve further by combining this two techniques. Therefore, we take the best setting from experiment 2, the sharing embedding and generator model. And we apply direct *L2* regularization and affine *L2* regularization from experiment 3 to this model. So we can get two models, which we call them model *NC-l2-direct-share-embed-gen* and model *NC-l2-affine-share-embed-gen* separately. We implement this two models and make a comparison with baseline and former proposed models. For the regularization part, we use same weight annealing technique to train combined model. The result as Table 4.10 and Table 4.11.

In terms of accuracy and perplexity, we find the proposed combine model have not

Table 4.10: Comparison of combined model, base model, and various former proposed models on EN-DE News Commentary v11 in terms of accuracy and perplexity of valid data *newstest2014*

	Accuracy (%)	**Perplexity**
NC-base	54.20	18.13
NC-l2-direct-share-embed-gen	54.93	15.60
NC-l2-affine-share-embed-gen	55.87	15.30
NC-share-embed-gen	55.61	15.12
NC-l2-direct	54.75	15.88
NC-l2-affine	55.65	15.48

hurt the performance of the *Nc-share-all* model. It obviously outperforms baseline model. And when compared with original regularization model *NC-l2-direct* and *NC-l2-affine*, we can find by sharing embedding and generator components it actually benefits the model performance. However, the strange point in this table is when compared with the original *NC-share-embed-gen*, we find the direct regularization hurts the performance, but the regularization with affine layer improved the performance in terms of accuracy. Although *NC-share-embed-gen* has the lowest perplexity, while we already find that it may not have big relation with metrics like BLEU, as we showed *NC-l2-affine* can beat it. So let's check other metrics.

Table 4.11: Comparison of combined model, base model, and various former proposed models on EN-DE News Commentary v11 in terms of NIST and BLEU score

	newstest2015		**newstest2016**	
	NIST	BLEU	NIST	BLEU
NC-base	5.7381	17.64	6.3912	20.43
NC-l2-direct-share-embed-gen	5.9679	18.43	6.5361	20.60
NC-l2-affine-share-embed-gen	**6.0247**	**18.67**	**6.6297**	**21.08**
NC-share-embed-gen	5.9317	18.24	6.5523	20.81
NC-l2-direct	6.0193	18.69	6.4968	20.34
NC-l2-affine	6.0193	18.63	6.6041	20.85

Then, we get the NIST and BLEU scores of our strongest proposed model *NC-l2-affine-share-embed-gen*, it get +1.03 improvement on *newstest2015* and +0.65 improvement on *newstest2016* in terms of BLEU, which is a big improvement. When looking at the comparison with other proposed models, for regularization with affine model the combine always improve performance. But for direct regularization model, the combined model sometimes helps and sometimes hurts.

Effect of sentence length

For our proposed model, because it can make advantage of bidirectional information, so we hypothesize that it can be better to translate sentences with longer length. To confirm this, at first, we concatenate the *newstest2015* and NESTEST2016 two test sets together to make a test set with totally 5168 sentences. We divide this test set into 6 buckets according to the sentence length, and calculate the BLEU [6] score for each bucket. The statistics of sentence number for the test set as Table 4.12.

Table 4.12: Number of sentences for each length buckets

	$<= 10$	$11 - 20$	$21 - 30$	$31 - 40$	$41 - 50$	> 50
Test set	1192	1998	1282	501	141	54

We take the baseline model *NC-base* and our best proposed model *NC-l2-affine-share-embed-gen*, and translate the dataset. Later, we calculate the BLEU score of the translation result. Then we compare their BLEU score. The result as Figure 4.5.

Figure 4.5 shows that proposed model outperform the baseline model on all length range. At first, the two model performance are quite similar. As the length increase, the difference of BLEU score between them become bigger. Especially, our proposed model have much better performance on longer sentences. That means our model indeed can use more information from bidirection to translate longer sentences compare with baseline model. However, as sentence length too long, there are a lot of noise so the decrease of performance is reasonable.

4.3.5 Experiment 5: Testing models on the ZH-EN data and the large DE-EN data

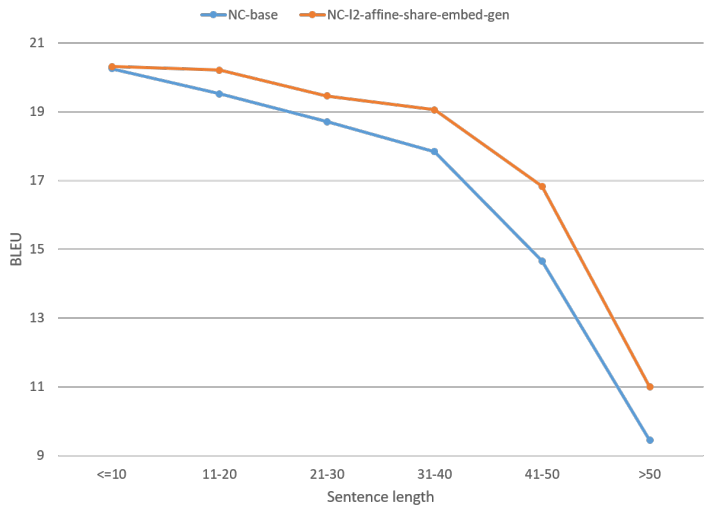
Experiments on the ZH-EN language pair

For the purpose of generality of our models, we tested the selected proposed model on another language pair, the English-Chinese pair. We run *NC-base*, *NC-share-embed-gen*, *NC-l2-affine*, and *NC-l2-affine-share-embed-gen* models on EN-ZH News Commentary v12 data just released last year. The experiment as Table 4.13 and Table 4.14.

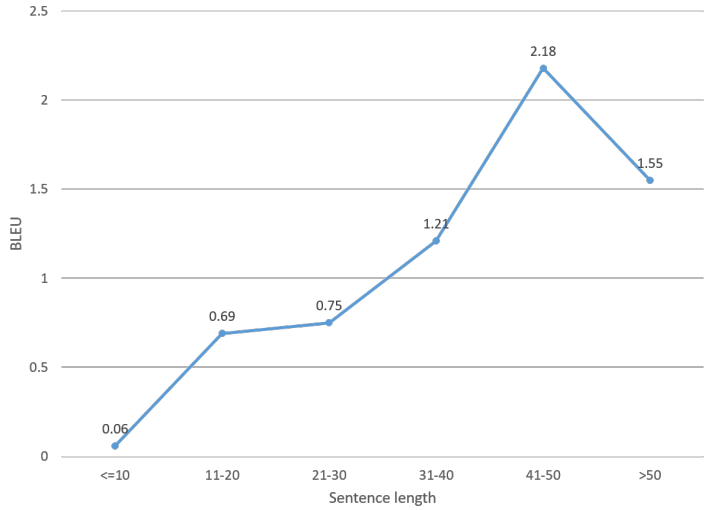
We can see in terms of accuracy and perplexity on validation set, English-Chinese pair also share the similar trend as English-German pair. Because the metrics here are calculated based on the preprocessed English and Chinese, and for NIST and BLEU scores we have one more post process step before calculation, so it is better to check them first.

In terms of NIST and BLEU scores, we can see proposed models outperform base model as demonstrated in English-German language pair. Especially, the combined model still achieve the highest performance among all the models, with *+1.18* BLEU compare with the base model.

[6] Calculate by *multi-bleu-detok.perl*: https://github.com/OpenNMT/OpenNMT-py/blob/master/tools/multi-bleu-detok.perl



(a) BLEU score on different length range



(b) BLEU score difference between two models

Figure 4.5: Compare BLEU score on different length sentences between *NC-base* and *NC-l2-affine-share-embed-gen*

Table 4.13: Comparison of various selected proposed models and base model on EN-ZH News Commentary v12 in terms of accuracy and perplexity of valid data *newsdev2017*

	Accuracy (%)	**Perplexity**
NC-base	36.46	84.87
NC-share-embed-gen	39.58	58.56
NC-l2-affine	39.73	59.83
NC-l2-affine-share-embed-gen	39.93	57.03

Table 4.14: Comparison of various selected proposed models and base model on EN-ZH News Commentary v12 in terms of NIST and BLEU score of test data *newstest2017*

	newstest2017	
	NIST	BLEU
NC-base	4.6506	14.98
NC-share-embed-gen	4.9306	15.83
NC-l2-affine	4.9901	16.10
NC-l2-affine-share-embed-gen	**5.0097**	**16.16**

Experiments on the large DE-EN data

To get more confidence about our proposed model, we test the proposed models on the DE-EN big dataset, which means the whole DE-EN dataset for WMT16 task. Because we have changed the data set, so we will also change each annotation of *NC* to *WMT*. For example, *NC-base* to *WMT-base*. For model with regularization, we also use weight annealing like before. But the limit for weight loss we set this is 0.5 as (Serdyuk et al., 2017). The accuracy and perplexity of our trained model like Table 4.15.

Table 4.15: Comparison of various selected proposed models and base model on WMT16 DE-EN data in terms of accuracy and perplexity of valid data *newstest2014*

	Accuracy (%)	**Perplexity**
WMT-base	63.79	6.17
WMT-share-embed-gen	65.17	5.80
WMT-l2-affine	65.07	5.80
WMT-l2-affine-share-embed-gen	64.90	5.85

We can see all the proposed model get better performance than baseline model in terms of accuracy and perplexity on valid set. However, one unexpected thing is that our strongest model, which share embedding and generator and have regularization with affine layer, become the worst model among selected proposed model. We think the reason might be the interaction between regularization and sharing components. The regularization loss weight 0.5 choose here is not good enough. Let's check NIST and BLEU score in Table 4.17 for further analysis.

We find that in terms of NIST and BLEU score, the *WMT-l2-affine* model become the worst model. This indicate the loss weight we choose here is indeed not good enough, and it need further exploration. The best model here become the pure MTL model *WMT-share-embed-gen* with +0.98 and +0.65 BLEU score improvement on *newstest2015* and *newstest2016* separately.

To further understand our model, we can also compare it with the state-of-art models. Because on WMT16 German to English direction there is not so many result about it. So we take the result from Edinburgh NMT system WMT16, the best model that year, and a recent syntax-aware NMT paper (Aharoni and Goldberg, 2017). We denote the

Table 4.16: Comparison of various selected proposed models and base model on WMT16 DE-EN data in terms of NIST and BLEU score of test data *newstest2015* and *newstest2016*

	newstest2015		**newstest2016**	
	NIST	BLEU	NIST	BLEU
WMT-base	7.7747	28.81	8.7081	34.12
WMT-share-embed-gen	**7.8274**	**29.79**	**8.6348**	**34.77**
WMT-l2-affine	7.7677	29.18	8.6205	34.27
WMT-l2-affine-share-embed-gen	7.7963	29.27	8.6403	34.38

Edinburgh baseline model as *Edinburgh-WMT16* and twos models from (Aharoni and Goldberg, 2017) as *bpe2bpe* and *bpe2tree*.

Table 4.17: Comparison best proposed model with stat-of-art models on WMT16 DE-EN translation task in terms of BLEU score on *newstest2015* and *newstest2016*

	newstest2015	**newstest2016**
WMT-share-embed-gen	**29.79**	**34.77**
Edinburgh-WMT16	26.4	28.5
+synthetic	29.9	36.2
+ensemble	31.5	37.5
bpe2bpe	27.33	31.19
bpe2tree	27.36	32.13
bpe2tree ensemble	28.7	33.24

We can see without using synthetic data and ensemble technique, our model is the best model on this dataset. For *bep2tree* model, even its ensemble can not beat our model performance. That means our model actually are a very competitive model.

Chapter 5

Conclusion

In this thesis, we present our research on exploring bidirectional decoding of neural machine translation. Our experiments include building a strong baseline by searching some parameters, such as beam size and length penalty, and comparison between the baseline model and our proposed models, which include multi-task learning model and regularization model. There are several points that we took it as the most important contribution in this study:

- We implemented a strong deep learning based neural machine translation baseline model. The baseline model is RNN based sequence-to-sequence model with attention mechanism. Except for reference some hyperparameters setting from previous papers, we also did some exploration on beam size and length and coverage penalty parameters. This experiment gives us a deeper understanding of how these parameters influence our translation result. And after choosing the best setting, we have a very strong baseline model, which can compete with the performance of many state-of-art models.

- We studied the influence of sharing components between forward and backward RNN in our decoder, which we call it multi-task learning model. We shared three components include attention component, target word embedding and generator separately, and how sharing component contribute to performance. We also share multiple components like embedding and generator together, it improves more. However, when we share all component it deteriorates the performance. After more analysis on real translation result, we find that the proposed model can actually capture more information than baseline model.

- We studied regularization way to do the bidirectional decoding of NMT model. We proposed two variants, one is used L2 regularization loss directly between forward hidden states and backward hidden states, another adds an affine transformation layer to get more flexibility. The result show these two regularization way all have improvement than baseline, especially the model with an affine layer. When compare with multi-task learning model, it is even better than them, because of the more

direct way to pass information. After same translation result analysis, it shows proposed model actually capture more information than baseline model.

- We studied combine multi-task learning technique and regularization technique together to do the bidirectional decoding of NMT model. We take the best sharing embedding and generator model and apply regularization on it. Then we get our strongest model multi-task learning model share embedding and generator with affine *L2* regularization, which is better than all the former proposed model. It gets a large improvement than the baseline model, especially in long length sentences.

- We selected the top performance models among the proposed models. And we run them on one more translation direction, English to Chinese. The result further support our declaration before. Besides, we run selected models on the WMT16 DE-EN dataset. First, we can see effectiveness of our proposed idea. Second, the best of our proposed model can beat almost all the state-of-arts on this dataset without synthetic data and ensemble technique.

Although there are more things that we want to explore in this research, we could not because of the limited time, resources and related knowledge. The main limitation of this research are:

- The reason for why sharing components in the multi-task learning model can get better performance has not been systematically analyzed. We could only give some intuitive assumption about why sharing generator is better than sharing word embedding and attention. And also the interaction between several components should be explored deeper to see why some multiple components shared model are worse than others.

- Although regularization way is quite successful, if we want it combine with the multi-task learning model to achieve a better result, then we need more time and experiments to explore the proper loss weight of it. And there are a lot of thing about their interaction that we can explore.

- We could not investigate more advanced regularization way to do the bidirectional decoding, such as variational auto-encoder (VAE) technique. Actually, we have already implemented the VAE model, however, the result is not good and the model can not learn more information from backward RNN. Although we spend a lot of time and use various techniques, such as weight annealing and auxiliary loss, want to solve this question, we could not solve it.

- Because of the limited time and resource, we could not run more experiments on more language pairs on big datasets, and combine some technique like synthetic data and ensemble model to achieve further improvement. We also have not do more careful comparison with related techniques.

Neural machine translation is a very interesting research area and still in high speed developing. Because using deep learning in machine translation is just a recent thing, so there are many techniques we can explore and research to improve neural machine translation system. Later, after combining all those techniques, it is possible we can achieve a machine translation system which is comparable to real human translator just as some recent paper shows (Hassan et al., 2018).

Bibliography

Aharoni, R. and Goldberg, Y. (2017). Towards string-to-tree neural machine translation. *arXiv preprint arXiv:1704.04743*.

Ahmed, K., Keskar, N. S., and Socher, R. (2017). Weighted transformer network for machine translation. *arXiv preprint arXiv:1711.02132*.

Alonso, H. M. and Plank, B. (2017). When is multitask learning effective? semantic sequence prediction under varying data conditions. In *15th Conference of the European Chapter of the Association for Computational Linguistics*.

Aslerasouli, P. and Abbasian, G. R. (2015). Comparison of google online translation and human translation with regard to soft vs. hard science texts. *Journal of Applied Linguistics and Language Research*, 2(3):169–184.

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

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Bingel, J. and Søgaard, A. (2017). Identifying beneficial task relations for multi-task learning in deep neural networks. *arXiv preprint arXiv:1702.08303*.

Britz, D., Goldie, A., Luong, T., and Le, Q. (2017). Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.

Caruana, R. (1998). Multitask learning. In *Learning to learn*, pages 95–133. Springer.

Castano, M. A., Casacuberta, F., and Vidal, E. (1997). Machine translation using neural networks and finite-state models. *Theoretical and Methodological Issues in Machine Translation (TMI)*, pages 160–167.

Chen, H., Huang, S., Chiang, D., and Chen, J. (2017). Improved neural machine translation with a syntax-aware encoder and decoder. *arXiv preprint arXiv:1707.05436*.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Deng, L., Hinton, G., and Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8599–8603. IEEE.

Doddington, G. (2002). Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, pages 138–145. Morgan Kaufmann Publishers Inc.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Eriguchi, A., Hashimoto, K., and Tsuruoka, Y. (2016). Tree-to-sequence attentional neural machine translation. *arXiv preprint arXiv:1603.06075*.

Firat, O., Cho, K., and Bengio, Y. (2016). Multi-way, multilingual neural machine translation with a shared attention mechanism. *arXiv preprint arXiv:1601.01073*.

Gehring, J., Auli, M., Grangier, D., and Dauphin, Y. N. (2016). A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Hashimoto, K., Xiong, C., Tsuruoka, Y., and Socher, R. (2016). A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*.

Hassan, H., Aue, A., Chen, C., Chowdhary, V., Clark, J., Federmann, C., Huang, X., Junczys-Dowmunt, M., Lewis, W., Li, M., et al. (2018). Achieving human parity on automatic chinese to english news translation. *arXiv preprint arXiv:1803.05567*.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701.

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

Hutchins, J. (1997). From first conception to first demonstration: the nascent years of machine translation, 1947–1954. a chronology. *Machine Translation*, 12(3):195–252.

Jean, S., Cho, K., Memisevic, R., and Bengio, Y. (2014). On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*.

Jean, S., Firat, O., Cho, K., Memisevic, R., and Bengio, Y. (2015). Montreal neural machine translation systems for wmtâĂŹ15. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 134–140.

Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., et al. (2016). Google's multilingual neural machine translation system: enabling zero-shot translation. *arXiv preprint arXiv:1611.04558*.

Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709.

Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. (2017). Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.

Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86.

Koehn, P. and Knowles, R. (2017). Six challenges for neural machine translation. *arXiv preprint arXiv:1706.03872*.

Kukačka, J., Golkov, V., and Cremers, D. (2017). Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*.

Lei, T. and Zhang, Y. (2017). Training rnns as fast as cnns. *arXiv preprint arXiv:1709.02755*.

Liu, L., Utiyama, M., Finch, A., and Sumita, E. (2016). Agreement on target-bidirectional neural machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 411–416.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Neubig, G. (2015). lamtram: A toolkit for language and translation modeling using neural networks.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Schwenk, H. (2007). Continuous space language models. *Computer Speech & Language*, 21(3):492–518.

Schwenk, H. (2012). Continuous space translation models for phrase-based statistical machine translation. *Proceedings of COLING 2012: Posters*, pages 1071–1080.

Sennrich, R., Birch, A., Currey, A., Germann, U., Haddow, B., Heafield, K., Barone, A. V. M., and Williams, P. (2017a). The university of edinburgh's neural mt systems for wmt17. *arXiv preprint arXiv:1708.00726*.

Sennrich, R., Firat, O., Cho, K., Birch, A., Haddow, B., Hitschler, J., Junczys-Dowmunt, M., Läubli, S., Barone, A. V. M., Mokry, J., et al. (2017b). Nematus: a toolkit for neural machine translation. *arXiv preprint arXiv:1703.04357*.

Sennrich, R., Haddow, B., and Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Sennrich, R., Haddow, B., and Birch, A. (2016). Edinburgh neural machine translation systems for wmt 16. *arXiv preprint arXiv:1606.02891*.

Serdyuk, D., Ke, R. N., Sordoni, A., Pal, C., and Bengio, Y. (2017). Twin networks: Using the future as a regularizer. *arXiv preprint arXiv:1708.06742*.

Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., and Arikawa, S. (1999). Byte pair encoding: A text compression scheme that accelerates pattern matching. Technical report, Technical Report DOI-TR-161, Department of Informatics, Kyushu University.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, volume 200.

Sundermeyer, M., Alkhouli, T., Wuebker, J., and Ney, H. (2014). Translation modeling with bidirectional recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 14–25.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

Vinyals, O., Kaiser, Ł., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. (2015). Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781.

Wang, P., Qian, Y., Soong, F. K., He, L., and Zhao, H. (2015). A unified tagging solution: Bidirectional lstm recurrent neural network with word embedding. *arXiv preprint arXiv:1511.00215*.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xia, Y., Tian, F., Wu, L., Lin, J., Qin, T., Yu, N., and Liu, T.-Y. (2017). Deliberation networks: Sequence generation beyond one-pass decoding. In *Advances in Neural Information Processing Systems*, pages 1782–1792.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057.

Yang, Z., Chen, W., Wang, F., and Xu, B. (2017). Improving neural machine translation with conditional sequence generative adversarial nets. *arXiv preprint arXiv:1703.04887*.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.

Zhang, X., Su, J., Qin, Y., Liu, Y., Ji, R., and Wang, H. (2018). Asynchronous bidirectional decoding for neural machine translation. *arXiv preprint arXiv:1801.05122*.