Effective Attention Mechanisms for Sequence Learning

LI, Jian

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
July 2020

Thesis Assessment Committee

Professor LO Chi Lik Eric (Chair)

Professor LYU Rung Tsong Michael (Thesis Supervisor)

Professor LEUNG Kwong Sak (Committe Member)

Professor GUO Song (External Examiner)

Abstract of thesis entitled:

Effective Attention Mechanisms for Sequence Learning

Submitted by LI, Jian

for the degree of Doctor of Philosophy

at The Chinese University of Hong Kong in July 2020

Sequence learning aims to process sequential data such as text, speech, and video, and discover valuable knowledge from them. Attention mechanism, as an effective method for dependency modeling, has become an indispensable component in deep sequence models such as recurrent neural networks and self-attention networks. However, we believe that the expressiveness of attention mechanism is not fully exploited, due to either the application domains or model design deficiencies. In this thesis, we explore effective ways to improve attention mechanisms for sequence learning from multiple perspectives. Our exploration ranges from shallow attention to deep self-attention, from sequence prediction to sequence generation, and engages applications from programming languages to natural language processing.

Firstly, we explore the conventional shallow attention (i.e., RNN-based attention) applied to source code completion. We treat the code completion task as a language modeling problem and propose a tailored attention mechanism that can exploit the structure information on program's abstract syntax tree. To deal with the out-of-vocabulary (OoV) words in a program, we further propose a pointer mixture network that learns to copy OoV words from local context based on the attention weights. Experiments on two benchmarked datasets demonstrate the effectiveness of our proposed methods.

Secondly, we focus on multi-head attention, one of the key components in self-attention mechanism. Multi-head attention is appealing for its ability to jointly extract different types of information from multiple representation subspaces. We propose two approaches to better exploit such diversity to improve multi-head attention. On one hand, we introduce a disagreement regularization to explicitly encourage the diversity among multiple attention heads. On the other hand, we propose to better aggregate the information distributed in the extracted partial-representations with the routing-by-agreement algorithm. We apply our approaches to the Transformer architecture and validate their effectiveness on both machine translation and language representation tasks.

Thirdly, we continue to explore the popular deep self-attention, i.e., the Transformer architecture for sequence-to-sequence learning. The strength of Transformer lies in its ability to capture different linguistic properties of the input sentence by different layers and different attention heads. Rather than using the last layer or linearly combining all attention heads, we study how to effectively aggregate the representations learned by different components. Specifically, we propose a bilinear pooling-based approach with low-rank approximation and first-order extension. Experiments on machine translation tasks show superior performances over the baselines.

Lastly, we study how to apply pre-trained attention models such as BERT to the downstream semantic parsing task, which generates executable code directly from natural language utterances. Current semantic parsers are mostly syntax-specific thus cannot generalize. We propose a model called BERT-LSTM that employs a pre-trained BERT encoder and a general-purpose LSTM decoder, to accomplish both effectiveness and generalization. We further incorporate a pointer network for copying code tokens from the inputs. We validate BERT-LSTM on four code generation datasets where the model achieves state-of-the-art on three of them.

In summary, this thesis targets at designing effective and customized solutions for improving attention mechanisms in sequence learning. Extensive experiments on various datasets across different applications demonstrate the effectiveness of our proposed methods.

論文題目：序列學習中有效的注意力機制
作者　　：李建
學校　　：香港中文大學
學系　　：計算機科學與工程學系
修讀學位：哲學博士
摘要　　：

序列學習旨在處理序列數據例如文本、語音和視頻，并從中發現有用的知識。注意力機制作為一種有效的依賴建模方法，已被廣泛應用於深度序列模型中比如循環神經網絡和自注意力網絡。然而，由於不同應用場景的需求和模型本身的缺陷，我們認為當下的注意力機制並沒有被充分利用。在本論文中，我們探索了各種改進序列學習中注意力機制的有效方法。我們從各個方面進行探索，包括淺層注意力和深層自注意力，序列預測和序列生成，以及編程語言和自然語言處理上的應用。

首先，我們關注傳統基於循環神經網絡的淺層注意力并將其應用在源代碼補全任務上。我們將代碼補全看做一個自然語言建模問題并提出一種定制的注意力機制以利用程序抽象語法樹上的結構信息。為了解決程序中的未登錄詞問題，我們進一步提出了一個混合指針網絡模型，讓其從局部上下文中基於注意力權重複製未登錄詞作為預測結果。我們在兩個基準數據集上進行實驗并驗證了我們方法的有效性。

其次，我們關注自注意力機制中的一個重要模塊：多頭注意力。多頭注意力的好處在於能同時從多個表示子空間中抽取到不同的信息。我們提出兩種方法來更好地利用這種差異性來改進多頭注意力。一方面，我們提出一種差異性約束用來顯示地

鼓勵多個注意力頭中的差異性。另一方面，我們提出使用協同路由算法來更好地融合各個子空間中的不同信息。我們將所提出的方法應用到Transformer模型上，并在機器翻譯和語言表示實驗上證明了我們方法的有效性。
再其次，我們繼續探索深層自注意力機制，即Transformer模型應用於序列到序列的學習。Transformer的優點在於能從不同的層次或不同的注意力頭中抽取到有關輸入語句的不同的語言學特性。不同於傳統方法只用最後一層表示或者線性結合多頭表示，我們研究如何有效地融合不同模塊中學習到的表示。具體地，我們提出一種基於雙線性池化的方法，并做了低階近似和擴展。在機器翻譯任務上進行的實驗展示了我們方法的有效性。
最後，我們研究如何將預訓練注意力模型比如BERT應用到下游的代碼生成任務上。當前的代碼生成模型大多數具有任務專一性，所以並不能泛化。我們提出一個BERT-LSTM模型用來同時兼顧代碼生成中的有效性和泛化性。該模型使用一個預訓練好的BERT編碼器，和一個通用的LSTM解碼器，并進一步嵌入了指針網絡用來從輸入語句中拷貝代碼。我們在四個公開的代碼生成數據集上驗證BERT-LSTM，並在其中三個取得了當前最優成績。
綜上所述，本論文的目標是設計有效和定制的方案來改進序列學習中的注意力機制。在不同應用和不同數據集上廣泛進行的實驗證明了我們方法的有效性。

Acknowledgement

I feel highly privileged to take this opportunity to express my sincere gratitude to the people who have been instrumental and helpful on my way to pursuing my Ph.D. degree.

First and foremost, I would like to thank my supervisor, Prof. Michael R. Lyu, for his kind supervision of my Ph.D. study at CUHK. He has provided inspiring guidance and incredible help in every aspect of my research. From maturing a research idea to working on the experiments, from technical writing to conference presentation, I have learned so much from him not only on knowledge but also the attitude towards doing research. Besides, he gives me a lot of freedom to select my research topic and study necessary techniques. I will always be grateful for his advice, encouragement, and support at all levels.

I am grateful to my thesis assessment committee members, Prof. Eric Lo and Prof. Kwong Sak Leung, for their constructive comments and valuable suggestions to this thesis and all my term reports. Great thanks to Prof. Song Guo from the Hong Kong Polytechnic University who kindly serves as the external examiner for this thesis.

I would like to thank my overseas supervisor, Prof. Monica S. Lam, for her support for my research visit to Stanford University. During the visit, Prof. Lam has provided insightful discussion and constructive feedback to my research. I also thank Silei Xu, Giovanni Campagna, Mehrad Moradshahi, Sina Semnani, and Michael Fischer, for the nine-month wonderful memories at Stanford. The experience to work with world-leading researchers at Stanford University will always inspire me.

I would like to greatly thank Dr. Zhaopeng Tu, my mentor during the internship at Tencent AI Lab. He unreservedly provides me valuable support for my research in this thesis. I also thank friends met there, Baosong Yang, Zi-Yi Dou, Yong Jiang, Jie Hao, Xinwei Gen, Yong Wang, Xing Wang, Longyue Wang, for their discussion and help.

I would like to thank my very good friends, Ruilin Ye, Xiang Xi, Peng Su, Minglin Li, Ming Liao, Zhaorui Wang, and Zhi Lu, for their encouragement and inspiration along my Ph.D. journey.

I sincerely thank Yue Wang, Zhaopeng Tu, Baosong Yang, Zi-Yi Dou, Xing Wang, and Pinjia He, for their valuable guidance and contribution to the research work in this thesis. I am also thankful for my other fantastic groupmates, Zibin Zheng, Yangfan Zhou, Yu Kang, Jieming Zhu, Cuiyn Gao, Hui Xu, Yuxin Su, Jichuan Zeng, Shilin He, Zhuangbin Chen, Haoli Bai, Wenxiang Jiao, Pengpeng Liu, Hou Pong Chan, Jiani Zhang, Jingjing Li, Yifan Gao, Tong Zhao, Hongyi Zhang, Shenglin Zhao, Xixian Chen, and Xiaotian Yu, who give me encouragement and kind help.

Last but not least, I would like to thank my parents. Without their deep love and constant support, this thesis would never have been completed.

To my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis presents our research on designing effective attention mechanisms for sequence learning, which is an important field of machine learning with a wide range of applications. We provide a brief overview of the research problems under study in Section 1.1, and highlight the main contributions of this thesis in Section 1.2. Section 1.3 outlines the thesis structure.

1.1 Overview

Sequential data is prevalent in our everyday life. Any data that arises through the measurement of time series is sequential data, for example, the rainfall on successive days at a particular location, the daily values of a stock price, and the acoustic features at consecutive time frames used for speech recognition, among others. Sequential data also exists in domains other than time series such as the sequence of characters and words in an English sentence, or the sequence of nucleotide base pairs along a strand of DNA [16].

We are interested in discovering knowledge from sequential data for various applications, for example, forecasting. Forecasting

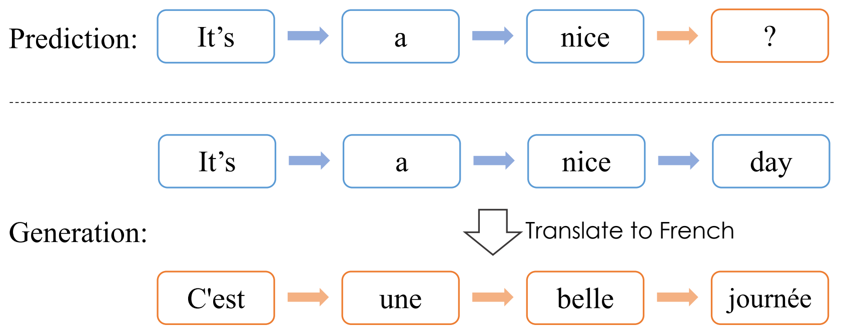


Figure 1.1: The two types of sequence learning tasks.

is the process of predicting the future based on current and previous data. The major challenge is extracting the patterns in the sequence of data and then using these patterns to analyze the future. If we were to hand-code the patterns, it would be inefficient and error-prone for the large amounts of sequential data in today's big data era. As a result, *sequence learning* arises as a research topic which automatically finds statistically relevant patterns among sequential data examples. There are generally two types of tasks in sequence learning, i.e., sequence prediction and sequence generation, which are illustrated in Figure 1.1. Sequence prediction attempts to predict the next immediate element of a sequence based on all the preceding elements, such as sentence completion. Sequence generation is basically the same as sequence prediction but attempts to piece together a sequence one by one as the way it naturally occurs, such as machine translation.

A great number of research efforts have been devoted to sequence learning. As pointed out by Agrawa et al. [1], the core research problem of sequence learning is *dependency modeling*, which can influence the features and meaning of a sequence to a large extent. To this end, Markov models [16] are proposed

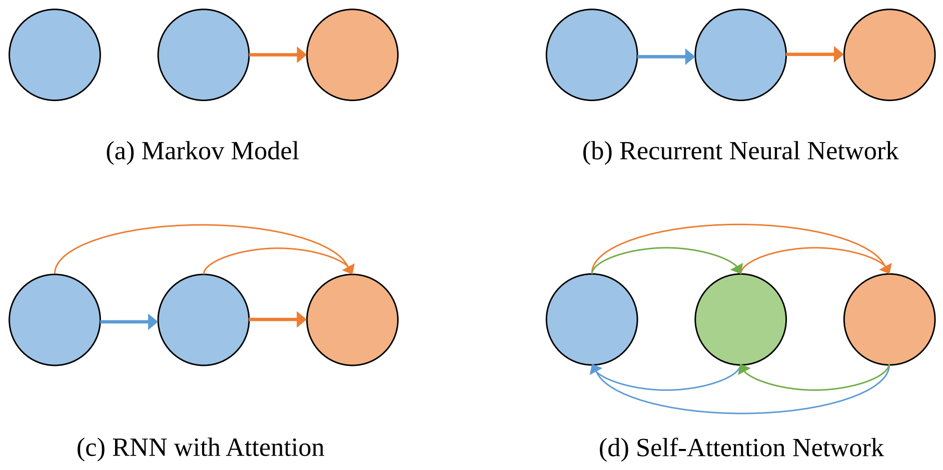


Figure 1.2: Different approaches to dependency modeling. (a) A first-order Markov chain where the prediction depends only on its prior element. (b) RNN naturally models the sequentiality. (c) Attention mechanism eases the hidden state bottleneck in RNN. (d) Self-attention parallelly captures the dependencies among all elements.

to probabilistically model the dependencies in sequential data. There is an assumption in Markov models that predictions only rely on most recent observations, as illustrated in Figure 1.2(a). As a result, Markov models can effectively capture short-term dependencies but fail on the long-term [96]. With the recent success of deep learning in all kinds of areas, Recurrent Neural Network (RNN) [50] has been adopted for sequence learning. As shown in Figure 1.2(b), RNN naturally models the sequentiality thus is able to deal with long sequences. Based on RNN, Long Short-Term Memory (LSTM) network and encoder-decoder framework have been developed and are widely employed in various applications. However, RNN has a so-called *hidden state bottleneck* [23], i.e., all prior observations are encoded in the last hidden vector which suffers from information loss as the sequence becomes longer. This problem restricts RNN's performance on capturing long-term dependencies.

Attention mechanism [7] is proposed to enhance RNN and mitigate the hidden state bottleneck. Figure 1.2(c) depicts how attention works: when predicting the current element, we use attention weights to denote how strongly it is correlated with (or attends to) previous elements and take the weighted sum of previous values to assist current prediction. These *shortcut connections* with all prior elements can effectively capture long-term dependencies. Besides, attention mechanism increases the interpretability of RNN models by explicitly showing how much each element contributes to the prediction with the attention weights. Nowadays, attention mechanism has become an indispensable component in deep sequence models to improve dependency modeling.

Despite its effectiveness, RNN with attention still processes the elements in sequential order which prohibits *parallel computation*. This quickly becomes problematic when training on large-scale sequential data. Self-attention network (SAN) [142] is proposed to discard the recurrent architecture and rely solely on attention mechanisms, resulting in increased parallelization and reduced training time. As illustrated in Figure 1.2(d), SAN simultaneously learns the dependencies among all elements through the attention weights, where colored arrows denote parallel operations. Since there is no sequential architecture, to encode the order information, SAN also adds positional encoding to the inputs. Due to its effectiveness and efficiency, SAN has achieved state-of-the-art on many sequence learning tasks such as speech recognition and machine translation. As SAN normally stacks in deep layers, to distinguish it from RNN-based attention, we term the former as *deep attention* and the latter as *shallow attention*.

Although conceptually simple and empirically powerful, attention mechanisms still face challenges when applied for sequence learning. First, since attention mechanisms are originally proposed for text data, we may need special designs for other domains of data. In this thesis, we focus on source code data. Compared to text data, source code is highly structured since it has well-defined grammar and syntax. Utilizing the underlying syntactic rules can help us further improve the learning performance. Besides, source code has larger vocabularies than text data as programmers can define arbitrary variable names, causing the Out-of-Vocabulary problem. Specifically-designed solutions are required to tackle such a problem.

Second, attention mechanisms still have design deficiencies. Previous research has improved the shallow attention from various aspects, for example, enhancing attention scopes with global and local attention [99]. Deep self-attention, however, is not fully explored as the concept is relatively new. Self-attention generally involves multiple components working together, e.g., multiple attention heads and multiple layers. In this thesis, we explore how to *coordinate* these components towards better performance, which is rarely studied by previous work.

Built on deep self-attention, pre-trained attention models (e.g., BERT [32], GPT [117]) have recently gained growing popularity. The core idea is to first train those models on large-scale text corpus to learn universal language representations, then fine-tune them on downstream tasks with supervised training. While considerable efforts have been dedicated to building better models in the pre-training phase, the fine-tuning phase is insufficiently investigated as different downstream tasks have different characteristics.

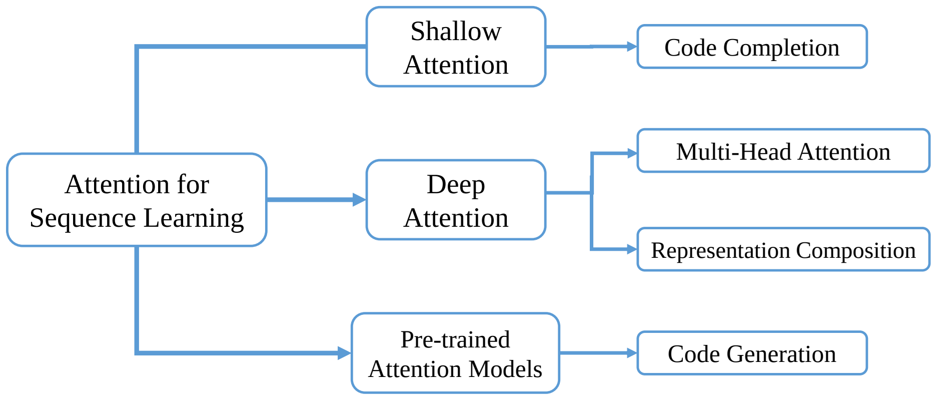

Shallow Attention
Code Completion
Multi-Head Attention
Attention for Sequence Learning
Deep Attention
Representation Composition
Pre-trained Attention Models
Code Generation

Figure 1.3: Overview of the research in this thesis.

Therefore, the research of this thesis comprises three parts, as depicted in Figure 1.3. In the first part, we apply shallow attention (i.e., RNN-based attention) to the source code completion task by designing tailored structures to exploit the properties of code. In the second part, we focus on improving deep attention (i.e., self-attention) based on two design deficiencies, namely multi-head attention and representation composition. In the third part, we explore how to apply pre-trained attention models such as BERT to the downstream code generation task.

1.2 Thesis Contributions

In this thesis, we make contributions to design effective attention mechanisms for sequence learning in the following ways:

1. **Neural Attention and Pointer Networks for Code Completion**

   Intelligent code completion has become an essential research task to accelerate modern software development. To facilitate effective code completion for dynamically-typed programming languages, we apply neural language models by learning from large codebases, and develop a tailored attention mechanism for code completion. To deal with the out-of-vocabulary (OoV) words in programs, we further propose a *pointer mixture network* which is inspired by the prevalence of locally repeated terms in program source code. Based on the context, the pointer mixture network learns to either generate a within-vocabulary word through an RNN component, or copy an OoV word from local context through a pointer component. Experiments on two benchmarked datasets demonstrate the effectiveness of our attention mechanism and pointer mixture network on the code completion task [87].

2. **Improving Multi-Head Attention via Exploiting Diversity**

   Multi-head attention is one of the key components in self-attention mechanism, with the appealing ability to jointly attend to information from different representation subspaces at different positions. In this thesis, we propose two approaches to better exploit such diversity for multi-head attention. First, we introduce three *disagreement regularizations* to explicitly encourage the diversity among multiple attention heads [85]. Specifically, we respectively encourage the subspace, the attended positions, and the output representations associated with each attention head to be different from other heads. Second, we propose to better aggregate the diverse information distributed in the extracted partial-representations with the *routing-by-agreement* algorithm [88]. The routing algorithm iteratively updates the proportion of how much a part should be assigned to a whole based on the agreement between parts and wholes. Experimental results on both machine translation tasks and sentence encoding tasks demonstrate the effectiveness and universality of the proposed approaches.

3. **Improving Self-Attention Networks via Representation Composition**

   Built on stacking of self-attention networks, the Transformer architecture has achieved state-of-the-art on many NLP tasks. The strength of Transformer lies in its ability to capture different linguistic properties of the input sentence by different layers and different attention heads. Rather than using the last layer or linearly combining all attention heads, in this thesis, we study how to effectively *compose (or aggregate) the representations* learned by different components (i.e., multi-layer networks or multi-head attention) [86]. Specifically, we leverage bilinear pooling to model pairwise multiplicative interactions among individual neurons, and a low-rank approximation to make the model computationally feasible. We further propose extended bilinear pooling to incorporate first-order representations. Experiments on machine translation tasks show that our model consistently improves performances over the baseline. Further analyses demonstrate that our approach indeed captures more syntactic and semantic information.

4. **Pre-trained Attention Models for Code Generation**

   Semantic parsing is the task to map natural language utterances to logical forms or executable code. The state-of-the-art semantic parsing models are mostly syntax-specific and heavily-engineered, thus they are not generalizable. In this thesis, we explore how to apply the powerful pre-trained attention models such as BERT to build semantic parsers that are both *effective* and *generalizable*. Specifically, We propose a novel BERT-LSTM model that employs a pre-trained BERT encoder followed by an LSTM decoder. We also adopt a pointer-generator network to learn to copy code tokens from the input. We demonstrate the effectiveness and universality of our model on three code generation tasks, where BERT-LSTM achieves state-of-the-are on three of the four datasets. We also highlight several design principles for code generation, such as the use of LSTM decoder and greedy decoding, and fine-tuning BERT parameters.

1.3 Thesis Organization

The remainder of this thesis is organized as follows:

- **Chapter 2**

  In this chapter, we give a systematic review of the background knowledge and related work on sequence learning and attention mechanisms. Firstly, we briefly introduce traditional sequence learning methods in Section 2.1, including TF-IDF and Markov models. Then, we review neural sequence learning (i.e., deep learning-based) approaches in Section 2.2, including RNN and LSTM, sequence-to-sequence learning and attention mechanisms, as well as pointer networks. After that, we introduce the self-attention mechanism and the Transformer architecture in Section 2.3. We also provide recent related studies on self-attention. Finally, in Section 2.4, we review pre-trained attention models and provide the related work.

- **Chapter 3**

  This chapter presents our study on adapting shallow attention for the source code completion task. We develop a parent attention to exploit the structure information and a pointer mixture network to solve the OoV problem. In particular, Section 3.1 introduces the code completion task and the problems it encounters. We then present our proposed approaches in Section 3.2 including the parent attention and pointer mixture network. Section 3.3 shows the evaluation details covering the datasets, experimental setup, experimental results, discussions, and case study. We show some related work on statistical code completion in Section 3.4 and summarize the chapter in Section 3.5.

- **Chapter 4**

  In this chapter, we explore how to improve multi-head attention, the core component in deep self-attention. To this end, we propose disagreement regularizations and routing-by-agreement algorithms to better exploit the diversity among multiple attention heads. Specifically, Section 4.1 presents our motivation for exploiting the diversity for multi-head attention. Section 4.2 describes some background knowledge. We elaborate the details of the two proposed approaches and how to combine them in Section 4.3. Section 4.4 and Section 4.5 present the experiments on machine translation and sentence encoding tasks. Section 4.6 shows some related work and Section 4.7 concludes this chapter.

- **Chapter 5**

  In this chapter, we continue to explore the deep self-attention. We propose a low-rank bilinear pooling-based approach to effectively aggregate the representations learned by different attention heads or layers. To be specific, we introduce our motivation for exploiting neuron interactions with bilinear pooling in Section 5.1. We then present some background knowledge in Section 5.2. Section 5.3 details our bilinear pooling approach for information aggregation. Section 5.4 presents evaluation including system setup, experimental results and Section 5.5 presents analysis. Section 5.6 illustrates some related work on bilinear pooling. Finally, we summarize this chapter in Section 5.7.

- **Chapter 6**

  This chapter presents our study on fine-tuning pre-trained attention models on the semantic parsing (i.e., code generation) task. We design a novel model called BERT-LSTM that are shown both effective and generalizable across several datasets. Specifically, Section 6.1 introduces the background and motivation for building a generalizable semantic parser. Section 6.2 elaborates the details about our BERT-LSTM model, including the BERT encoder, LSTM decoder, and pointer network in between. Section 6.3 details our evaluation on four code generation datasets, consisting of the explanations of datasets and system setup, the experimental results, and code generation case studies. We present some related work on the semantic parsing task in Section 6.4 and conclude this chapter in Section 6.5 with several empirical finds.

- **Chapter 7**

  The last chapter summarizes this thesis and provides some potential future directions for sequence learning with attention mechanisms that deserve further exploration.

□ **End of chapter.**

Chapter 2

Background Review

This chapter reviews some background knowledge and related work. We first introduce traditional sequence learning methods such as Markov models. Then we explain neural (i.e., deep learning-based) sequence learning approaches including RNN and attention mechanisms in Section 2.2. Next, we describe self-attention mechanisms and pre-trained attention models in Section 2.3 and Section 2.4, respectively. In each subsection, we first present the technical details, then the related studies.

2.1 Traditional Sequence Learning

A great number of research efforts have been devoted to sequence learning. Among them, the simplest method is to treat the sequential data as independent elements and ignore their correlations. In this way, the method only counts the elements' frequencies, which has inspired models like Bag-of-Words and TF-IDF [119]. TF-IDF refers to term frequency–inverse document frequency, which is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. TF-IDF has been widely used by search engines as a central tool in scoring and ranking a document's relevance given a user query [17, 132]. Though effective to certain extents, such methods fail to exploit the sequential dependencies in sequential data.

As pointed out by Agrawa et al. [1], the core research problem of sequence learning is *dependency modeling*, which can influence the features and meaning of a sequence to a large extent. To this end, Markov models [16] are proposed to probabilistically model the dependencies in sequential data. From a probabilistic view, we can adopt the product rule to denote the joint distribution for a sequence of elements in the following form:

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \prod_{n=1}^{N} p(\mathbf{x}_n | \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}). \tag{2.1}$$

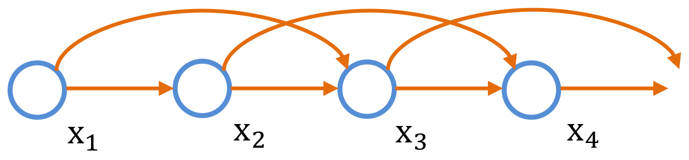
As the above equation is generally intractable, to simplify it, we can assume that each of the right side conditional distributions is independent of all prior elements except the most recent one. In this way, we get the *first-order Markov chain*, which can express Equation 2.1 as following:

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = p(\mathbf{x}_1) \prod_{n=2}^{N} p(\mathbf{x}_n | \mathbf{x}_{n-1}). \tag{2.2}$$

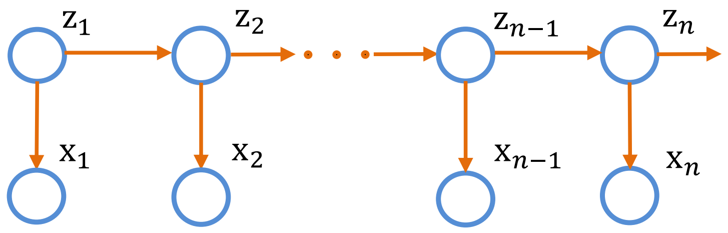
Therefore, the conditional distribution for element $\mathbf{x}_n$ given all the prior elements up to time n is conditioned only on the immediately preceding element $\mathbf{x}_{n-1}$.

To further relax the independent assumption, for example, if we allow the predictions to depend on previous two elements, we can obtain the *second-order Markov chain* as illustrated in Figure 2.1(a). The joint distribution is now expressed by:

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1) \prod_{n=3}^{N} p(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{x}_{n-2}). \tag{2.3}$$



(a) second-order Markov chain



(b) hidden Markov model

Figure 2.1: Graphical illustration of (a) a second-order Markov chain where the prediction depends on the two previous observations, and (b) a hidden Markov model with $\{z_n\}$ being hidden variables.

Similarly, we can extend the above models to an M^{th} order Markov chain where the conditional distribution for a particular element depends on the previous M elements. However, the cost for such increased flexibility is the much larger number of parameters in the model. The number of parameters increases exponentially with M, making the model impractical for larger values of M.

To overcome the above contradiction between expressiveness and complexity, *hidden Markov model (HMM)* [41] is proposed where the observations are not limited by the Markov assumption to any order. The core idea is to introduce additional latent variables in addition to the explicit observation. Specifically, for each observation $\mathbf{x}_n$, a corresponding latent variable $\mathbf{z}_n$ is introduced. The assumption is that the latent variables form a Markov chain rather than the observations, as the graphical structure depicted in Figure 2.1(b). As a result, the joint distribution for this model is expressed as:

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{z}_1, \ldots, \mathbf{z}_N) = p(\mathbf{z}_1) \left[\prod_{n=2}^{N} p(\mathbf{z}_n | \mathbf{z}_{n-1} \right] \prod_{n=1}^{N} p(\mathbf{x}_n | \mathbf{z}_n). \tag{2.4}$$

From the d-separation property, we see that the path connecting any two observed variables $\mathbf{x}_n$ and $\mathbf{x}_m$ always exists via the latent variables. Therefore, the distribution $p(\mathbf{x}_n | \mathbf{x}_1, \ldots, \mathbf{x}_{n-1})$ for prediction $\mathbf{x}_n$ given all prior observations does not show any conditional independence properties. Note that in HMM, the latent variables $\{z_n\}$ should be discrete.
The HMM has been widely employed in various applications such speech recognition [67, 116], natural language modeling [101], computational finance [130], musical score following [112], on-line handwriting recognition [108], transportation forecasting [168], and biological sequential data analysis like proteins and DNA [80, 40, 8], etc.

2.2 Neural Sequence Learning

Neural sequence learning refers to deep learning-based approaches such as RNN and attention mechanisms.

2.2.1 RNN and LSTM

In the last few years, as deep learning achieves success in all kinds of areas, Recurrent Neural Networks (RNNs) have been extensively adopted for sequence learning although proposed many years ago [122]. One of the core challenges for sequence learning is the variable sequence lengths, as it is impractical to design models with separate parameters for each length of the sequence. RNN overcomes this challenge through *parameter*

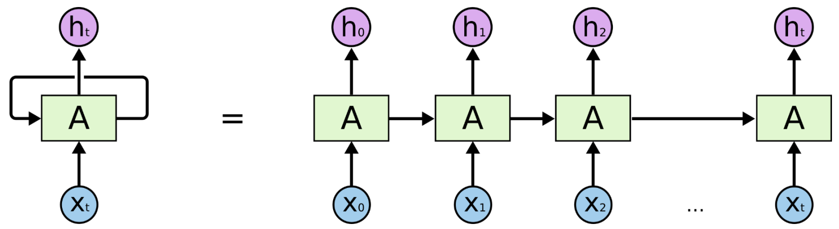


Figure 2.2: Illustration of an unrolled RNN. Figure is from [28].

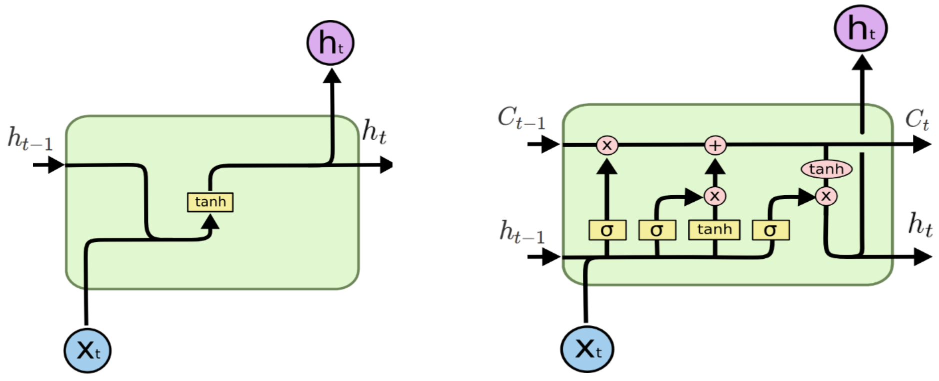
sharing, which makes it possible to extend and apply the model to sequences of different lengths and generalize across them. As illustrated in Figure 2.2, RNN has a cycle connection that can be unrolled along time steps for arbitrary long sequences. For each time step, RNN shares the same parameters. Formally, an RNN cell can be expressed as:

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta), \tag{2.5}$$

where $\mathbf{x}^{(t)}$ is the system input and $\mathbf{h}^{(t)}$ is the system state (it is generally called hidden state as it is the hidden units of the network), transition function $f()$ maps the state at t to the state at $t+1$. The same parameters, i.e., the values of θ used to parameterize f, are used for all time steps.

Though the chain-like RNNs naturally model sequentiality and are empirically effective, RNNs still suffer from long-range dependencies. The basic problem is that during gradient descent training, the gradients propagated over many time steps tend to either vanish (most of the cases) or explode [50]. Training on long-term dependencies will produce exponentially smaller weights (as multiplying many Jacobians) in comparison to the short-term ones.

The above problem has been explored in depth by many studies [62, 38, 12] and many solutions are developed. Among them, Long Short-Term Memory networks (LSTMs) [63] are



(a) RNN Cell (b) LSTM Cell

Figure 2.3: Illustration of a standard RNN cell and an LSTM cell. [28]

the most effective one that work tremendously well on a large variety of problems. The secret ingredient of LSTM is the *gating mechanism*, which dynamically decides how much information should be removed or added to the cell state. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer which outputs numbers between zero and one, and a point-wise multiplication operation. An LSTM cell has three of these gates, namely *forget gate, input gate, and output gate*, to protect and control the cell state. We depict a standard RNN cell and an LSTM cell in Figure 2.3. We can see that the standard RNN cell only contains a single *tanh* layer, while there are four interacting layers in the LSTM cell. C_t denotes the LSTM cell states.
Formally, an LSTM cell with these gates is expressed as:

$$\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t
\end{aligned}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

where σ refers to the sigmoid function. f_t, i_t and o_t denote the forget gate, input gate and output gate, respectively.

Besides the above LSTM, there are also many LSTM variants in the literature. Gated Recurrent Units (GRU) [24] are the most popular one. It combines the forget gate and input gate as a single update gate, and merges the cell state and hidden state. In addition, Gers et al. [48] propose to add "peephole connections" to LSTM and let the gate layers look at the cell state. There are other variants like Depth Gated RNNs by Yao et al. [161] and Clockwork RNNs by Koutnik et al. [79].

Several investigations have been conducted in terms of these LSTM variants. Greff et al. [52] conduct a nice comparison across popular variants and find that they are all about the same. Jozefowicz et al. [71] test more than ten thousand RNN architectures and find that adding a bias of 1 to the LSTM forget gate making LSTM the best explored architecture.

2.2.2 Seq2Seq Learning and Attention

With a single RNN/LSTM, we can map an input sequence to a fixed-size vector or an output sequence of the same length. However, for many sequence generation applications like machine translation and question answering, the input and output sequences are generally not of the same length (though might be related). In this context, sequence-to-sequence (Seq2Seq) learning [135] is proposed in which RNNs map a variable-length sequence to another variable-length sequence, as shown in Figure 2.4.

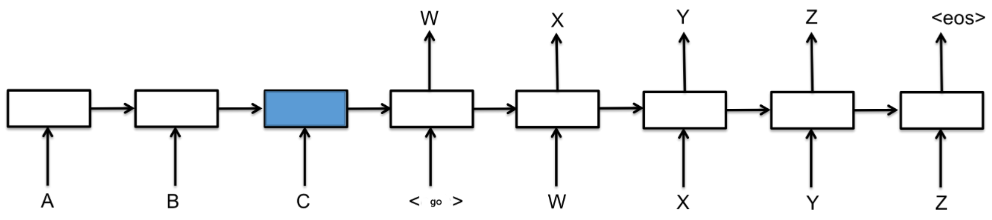


Figure 2.4: An encoder-decoder architecture for Seq2Seq learning. The blue-colored cell denotes the context vector produced by the encoder and serves as input to the subsequent decoder.

Sequence-to-sequence learning depends on the encoder-decoder architecture, composed of an encoder RNN and a decoder RNN. The encoder (or reader) RNN processes and summarizes the input sequence as a context vector C. The decoder (or writer) RNN generates the output sequence conditioned on that fix-length context vector. The last hidden state of the encoder RNN is typically used as the context vector that served as input to the decoder RNN, as the blue-colored cell in Figure 2.4. The two RNNs are trained jointly to maximize the probability of generating the correct output sequence based on the input sequence. There is no constraint that the encoder and decoder must have the same size of hidden layer.

The limitation of such encoder-decoder architecture is obvious: the context vector C output by the encoder RNN may have a dimension that is too small to adequately summarize a long sequence and tend to forget the earlier inputs. This problem is also called *hidden state bottleneck*, which restricts the performance on capturing long-term dependencies. *Attention mechanism* [7] is therefore developed to solve the problem in the context of machine translation.

As illustrated in Figure 2.5, attention mechanisms add shortcut connections from each decoder step S_t to all the encoder hidden

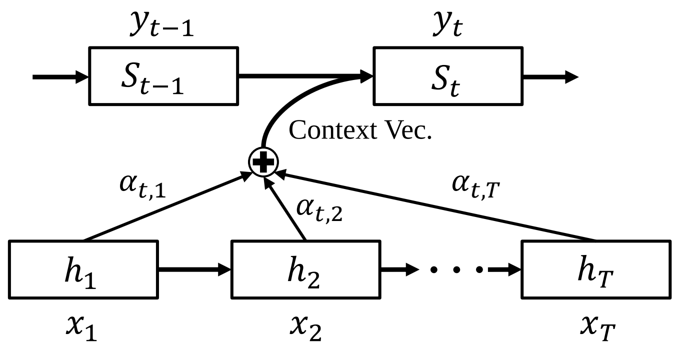


Figure 2.5: Illustration of attention mechanism in an encoder-decoder architecture. $\alpha_{t,i}$ denotes the attention weights.

states h (in addition to the original connection to prior decoder step), and use attention weights $\alpha_{t,i}$ to denote the connection strengths. Then the context vector C_t is computed by taking the sum of all encoder hidden states weighted by the attention weights. By doing so, the model searches for a set of positions in the encoder hidden states where the most relevant information is available, thus effectively captures long-term dependencies.
The above process can be formally expressed by:

$$\begin{aligned} C_t &= \sum_{i=1}^{T} \alpha_{t,i} h_i \\ \alpha_{t,i} &= align(y_t, x_i) \\ &= \frac{exp(score(s_{t-1}, h_i))}{\sum_{i'=1}^{n} exp(score(s_{t-1}, h_{i'}))} \end{aligned}$$

where C_t is the context vector and $\alpha_{t,i}$ denotes the alignment score between output y_t and input x_i.
Another benefit of attention mechanism is the increased interpretability by explicitly showing how much each input element contributes to the prediction with the attention weights.
The encoder-decoder architecture with attention has been extremely successful in many sequence-to-sequence learning tasks,

for example, machine translation [7, 99], summarization [109, 124], reading comprehension [126], question answering [139], semantic parsing [34], and among others. Besides the natural language processing field, the model also achieves great success in other areas such as speech recognition [25], image caption [152, 72, 164] and visual question answering [151, 98].

Machine translation is a typical sequence-to-sequence learning task and has received a great number of research efforts. The above Seq2Seq learning and attention mechanism are both originally proposed for machine translation [135, 7]. Armed with deep learning techniques, neural machine translation (NMT) has demonstrated ground-breaking performances over traditional phrase-based statistical machine translation (SMT) [18]. Many research works try to further improve the attention mechanism for NMT. Luong et al. [99] propose local attention to refine the attention scope, which calculates the relevance with a subset of the source sentence. Yang et al. [160] enhance the attention structure by modeling the relationship of a word with its previous and subsequent attention. Feng et al. [43] propose recurrent attention mechanism to attain more accurate alignments. Tu et al. [140] propose the coverage mechanism into the decoder which adjusts the context vector in NMT by adding coverage information during attention calculation, thus improves the adequacy of translation. Combining several advanced techniques, Google develops GNMT [148], an industry-level model applied in Google Translation. To solve the OOV (i.e., rare words) problem in NMT, Sennrich et al. [125] propose Byte Pair Encoding (BPE) to split each word into sub-word units during preprocessing, which are now widely adopted. Following the same idea, wordpiece tokenization is proposed in GNMT [148]. Besides, Ling et al. [95] propose character-level NMT as an alternative to deal with the OOV problem. Convolutional sequence-to-sequence learning is proposed by Gehring et al. [47] to discard RNNs and only utilize convolution operations for machine translation.

2.3 Self-Attention Networks

The sequential nature of RNNs precludes parallelization within the training samples, which becomes critical for long sequences and large-scale datasets.

Self-attention networks (SANs) [93, 142], as a variant of the above attention mechanisms, have recently drawn increasing interests due to their flexibility in parallel computation and the capability to model both long and short-term dependencies. As illustrated in Figure 1.2(d), SANs simultaneously calculate the attention weights between each pair of the tokens in a sequence, thus capturing long-range dependencies more directly than their RNN counterpart.

Formally, given an input layer $\mathbf{H} = \{h_1, \dots, h_n\}$ which is generally the embedded representations of the input tokens, the output hidden states of SAN are constructed by correlating (or attending) to the states of input layer. Specifically, SAN first transforms the input layer $\mathbf{H} \in \mathbb{R}^{n\times d}$ into queries $\mathbf{Q} \in \mathbb{R}^{n\times d}$, keys $\mathbf{K} \in \mathbb{R}^{n\times d}$, and values $\mathbf{V} \in \mathbb{R}^{n\times d}$:

$$\begin{bmatrix}\mathbf{Q}\\ \mathbf{K}\\ \mathbf{V}\end{bmatrix} = \mathbf{H}\begin{bmatrix}\mathbf{W}_Q\\ \mathbf{W}_K\\ \mathbf{W}_V\end{bmatrix}, \tag{2.6}$$

where $\{\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V\} \in \mathbb{R}^{d\times d}$ are trainable parameter matrices with d being the dimensionality of input states. The output layer $\mathbf{O} \in \mathbb{R}^{n\times d}$ of SAN is constructed by

$$\mathbf{O} \;=\; \textsc{Att}(\mathbf{Q},\mathbf{K})\ \mathbf{V}, \tag{2.7}$$

where $\textsc{Att}(\cdot)$ is an attention function, which can be implemented as either additive attention [7] or dot-product attention [99, 142]. In this thesis, we employ scaled dot-product attention, which has proved in similar performance with its additive counterpart but much faster and more space-efficient in practice [142]:

$$\textsc{Att}(\mathbf{Q},\mathbf{K}) \;=\; softmax(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}), \tag{2.8}$$

where scaling factor d is the dimensionality of layer states.

Transformer Architecture Based on SANs, Vaswani et al. [142] propose the Transformer architecture, which has achieved record-setting performances in various applications such as machine translation and constituency parsing. We illustrate the architecture of Transformer in Figure 2.6, which is in essence an encoder-decoder framework. We can see that both the encoder and decoder are stacked in N identical layers. Each layer has two kinds of sub-layers, i.e., multi-head self-attention mechanism and fully connected feed-forward network. Residual connection [56] and layer normalization [6] are added around each sub-layer to facilitate better training. A recent analysis paper [33] has revealed that the most contributing components in Transformer are the *multi-head attention* and *multi-layer networks*. Many follow-up research has been conducted. Therefore, in the following sections, we present more details and related studies of the two components.

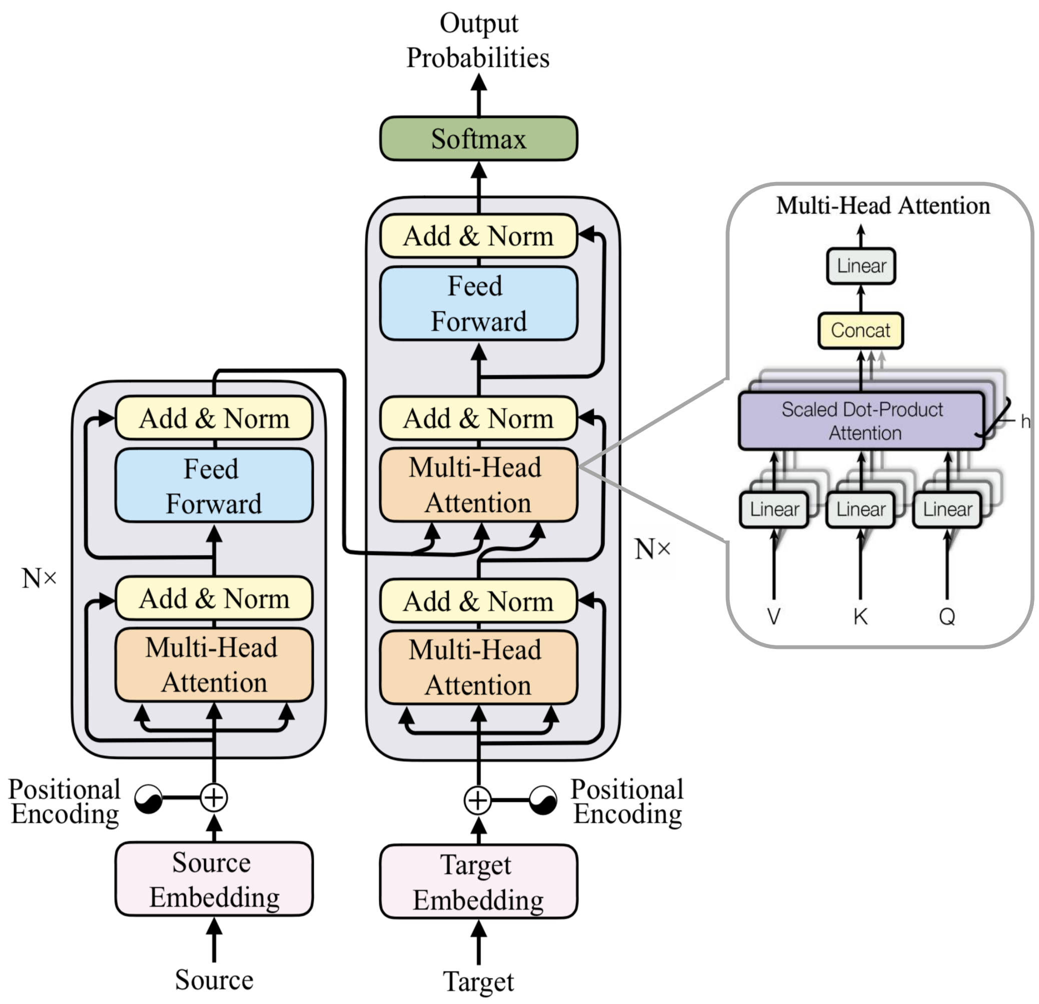


Figure 2.6: Illustration of the Transformer architecture.

2.3.1 Multi-Head Attention

Instead of performing a single attention function as in Equation 2.7, Vaswani et al. [142] find it beneficial to capture different context features with multiple individual attention functions. Specifically, multi-head attention transforms $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ into H subspaces with different, trainable linear projections:

$$\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h = \mathbf{Q}\mathbf{W}_h^Q, \mathbf{K}\mathbf{W}_h^K, \mathbf{V}\mathbf{W}_h^V, \tag{2.9}$$

where $\{\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h\}$ are respectively the query, key, and value representations of the h-th head. Then individual attention heads *parallelly* perform Equation 2.7 to get the output of each head $\mathbf{O}_h$. The multiple outputs are further combined together as the final output.

Multi-head attention has shown promising results in many natural language processing (NLP) tasks, such as machine translation [142, 33], semantic role labeling [131], dialog [137], subject-verb agreement task [136]. Previous work shows that multi-head attention can be further enhanced by encouraging individual attention heads to extract distinct information. For example, Lin et al. [93] introduce a penalization term to reduce the redundancy of attention weights among different attention heads, and Yang et al. [157] model the interactions among attention heads through convolution. Yang et al. [155] also propose local self-attention and show that different attention heads have different attention scopes. Shen et al. [127] explicitly use multiple attention heads to model different dependencies of the same word pair, and Strubell et al. [131] employ different attention heads to capture different linguistic features of the sentence. In this thesis, we aim to improve multi-head attention by exploring more about the diversity.

2.3.2 Multi-Layer Networks

As shown in Figure 2.6, the Transformer encoder consists of a stack of N identical layers. Each layer has two sub-layers, the first is a self-attention network and the second is a fully connected feed-forward network. In the n-th layer, the outputs of the first sub-layer $\mathbf{C}_e^n$ and the second sub-layer $\mathbf{H}_e^n$ are formally expressed as:

$$\begin{aligned} \mathbf{C}_e^n &= \textsc{Ln}(\textsc{Att}(\mathbf{Q}_e^n, \mathbf{K}_e^{n-1}, \mathbf{V}_e^{n-1}) + \mathbf{H}_e^{n-1}), \\ \mathbf{H}_e^n &= \textsc{Ln}(\textsc{Ffn}(\mathbf{C}_e^n) + \mathbf{C}_e^n), \end{aligned} \tag{2.10}$$

where $\textsc{Att}(\cdot)$, $\textsc{Ln}(\cdot)$, and $\textsc{Ffn}(\cdot)$ are self-attention network, layer normalization, and feed-forward network, respectively. $\{\mathbf{Q}_e^n, \mathbf{K}_e^{n-1}, \mathbf{V}_e^{n-1}\}$ are query, key and value vectors transformed from the (n-1)-th encoder layer $\mathbf{H}_e^{n-1}$. The Transformer decoder is basically the same but has one more sub-layer to perform attention between encoder and decoder.

Exploiting multi-layer representations has been well studied in the computer vision field. He et al. [56] propose a residual learning framework that adds shortcuts to layers and encourages gradient flow. Huang et al. [64] further extend the idea by introducing densely connected layers. Recently, Yu et al. [165] propose deep layer aggregation strategies to fuse more information across layers. Exploiting multi-layer representations has also been studied in the NLP community. Peters et al. [113] have found that linearly combining different layers is helpful and improves their performances on various NLP tasks. In the context of NMT, several neural network-based approaches to fuse information across historical layers have been proposed. Shen et al. [128] propose dense information flow and dense attention structure to build a densely connected NMT. Dou et al. [36] propose to aggregate the NMT layers with iterative and hierarchical aggregation strategies, as well as routing-by-agreement algorithm [37]. Bapna et al. [9] propose transparent attention to ease the optimization of deep NMT models. They all demonstrate that aggregating deep layers is beneficial for NMT, which will also be explored in this thesis.

2.4 Pre-trained Attention Models

Recently, the emergence of pre-trained models (PTMs) has transformed the field of NLP research. Since most PTMs are built on self-attentional Transformer architecture (e.g., the Transformer encoder), to make this thesis more consistent, we also call PTMs *pre-trained attention models*.

The core of deep learning research is representation learning. When it comes to natural language, a good representation of text data should capture the underlying linguistic properties and common sense knowledge. To this end, researchers have built increasingly deep and wide neural networks to process text data, producing large numbers of trainable parameters. However, it is notoriously hard to train those large neural models given the limited amounts of labeled text data. *Pre-training and fine-tuning* arise as a promising solution to such problem by making use of the huge unlabeled text data. The main idea is to first pre-train large models on large-scale unlabeled text data to learn universal language representations, then fine-tune those models on downstream specific tasks with limited labeled data, as illustrated in Figure 2.7. In this way, we can effectively transfer language knowledge from large unannotated corpus and prevent model overfitting on limited data.

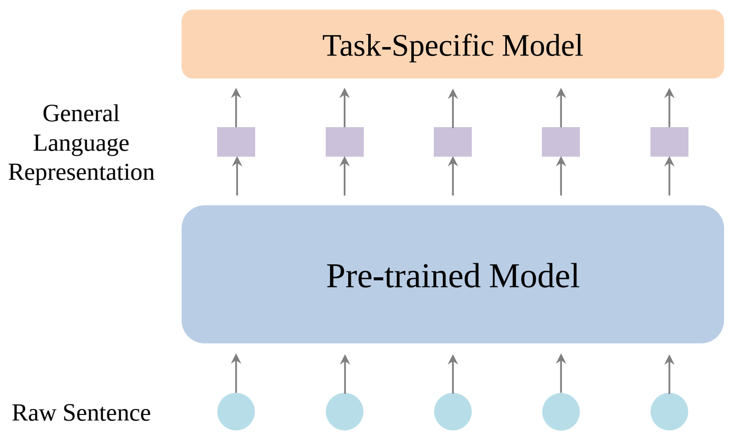


Figure 2.7: The use of Pre-trained models in NLP.

In the field of computer vision, pre-training has already been widely adopted. Most models are first pre-trained on the huge ImageNet dataset, and then fine-tuned further on smaller data for different tasks. In the field of NLP, we have recently seen a surge in interests towards PTMs. With pre-training on the huge text corpus, the model can learn universal language representations and help with the downstream tasks. Pre-training also offers the model a better parameter initialization, which normally speeds up the convergence during training.

In the NLP literature, the first-generation PTMs are Pre-trained Word Embeddings which are context-independent. The representative models are Continuous Bag-of-Words (CBOW) and Skip-Gram (SG) proposed by Mikolov et al. [104].

More recently, with the advancement of computing power and model architectures (i.e., Transformer), the very deep PTMs have been proposed to effectively learn universal language representations. We call them second-generation PTMs as they are all contextual-dependent. The representative models include ELMo (Embeddings from Language Models) [113], OpenAI GPT (Generative Pre-training) [117], BERT (Bidirectional Encoder Representation from Transformer) [32], ERNIE (Enhanced Representation through kNowledge IntEgration) [134], etc. They mainly differ at the pre-training tasks on large text corpus. BERT employs Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), while GPT adopts a normal Language Modeling (LM) objective which predicts the next word given a sentence. The different pre-training tasks make the models suitable for different downstream tasks, for example, BERT for language understanding such as classification while GPT for language generation.

□ **End of chapter.**

Chapter 3

Neural Attention and Pointer Networks for Code Completion

This chapter presents our study on applying LSTM and attention mechanisms for the source code completion task, which is essential research to accelerate modern software development. Specifically, we target at dynamically-typed programming languages (e.g., Python) and apply neural language models by learning from large codebases. The main points of this chapter are as follows. (1) We develop an attention mechanism for code completion, which makes use of the structure information (specially, the parent-children information) on abstract syntax tree. (2) We propose a pointer mixture network for better predicting OoV words in code completion, which learns to generate next word from either the global vocabulary or the local context. (3) We evaluate our models on two benchmarked datasets (JavaScript and Python). The experimental results show great improvements upon the state-of-the-arts.

3.1 Introduction

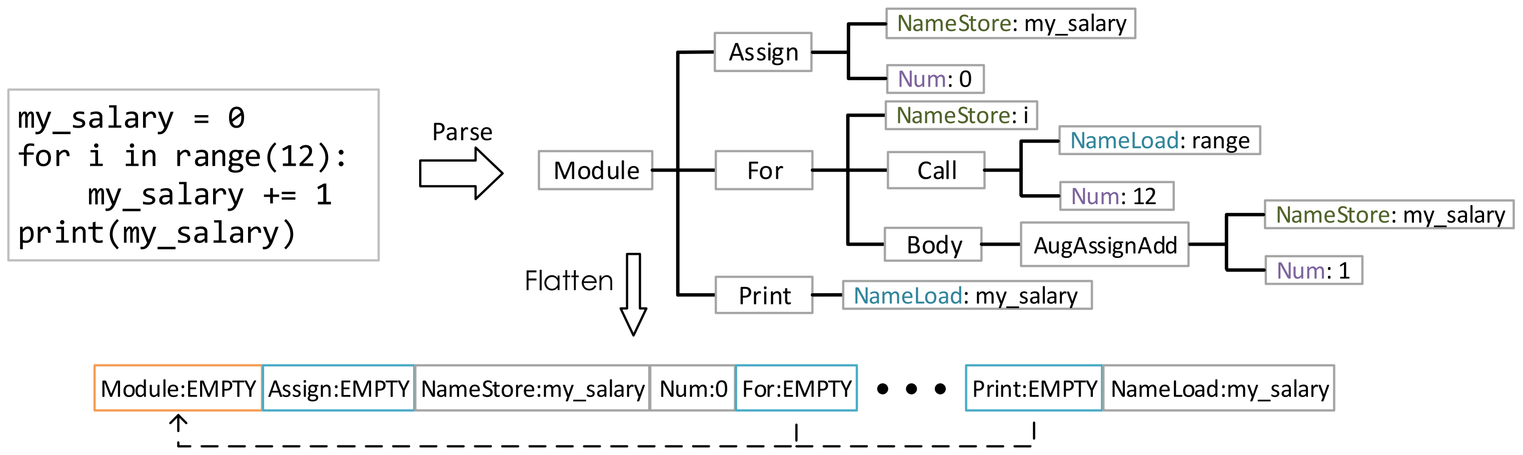
Integrated development environments (IDEs) have become essential paradigms for modern software engineers, as IDEs provide a set of helpful services to accelerate software development. Intelligent code completion is one of the most useful features in IDEs, which suggests next probable code tokens, such as method calls or object fields, based on existing code in the context. Traditionally, code completion relies heavily on compile-time type information to predict next tokens [141]. Thus, it works well for statically-typed programming languages like Java. Yet code completion is harder and less supported for dynamically-typed programming languages such as JavaScript and Python, due to the lack of type annotations.

To render effective code completion for dynamically-typed languages, recently, researchers turn to learning-based language models [58, 146, 15]. They treat programming languages as natural languages, and train code completion systems by learning from large codebases (e.g., GitHub). In particular, neural language models such as Recurrent Neural Networks (RNNs) can capture sequential distributions and deep semantics, hence become very popular. However, these standard neural language models are limited by the so-called *hidden state bottleneck*: all the information about current sequence is compressed into a fixed-size vector. The limitation makes it hard for RNNs to deal with long-range dependencies, which are common in program source code such as a class identifier declared many lines before it is used.

Attention mechanism [7] provides one solution to this challenge. With attention, neural language models learn to retrieve and make use of relevant previous hidden states, thereby increasing the model's memorization capability and providing more paths for back-propagation. To deal with long-range dependencies in code completion, we develop a tailored attention mechanism that can exploit the structure information on program's *abstract syntax tree* (AST, see Figure 3.1), which will further be described in the following.

But even with attention, there is another critical issue called *unknown word problem*. In general, the last component of neural language models is a softmax classifier, with each output dimension corresponding to a unique word in the predefined vocabulary. As computing high-dimensional softmax is computational expensive, a common practice is to build the vocabulary with only K most frequent words in the corpus and replace other out-of-vocabulary (OoV) words with a special word, i.e., *UNK*. Intuitively, standard softmax based neural language models cannot correctly predict OoV words. In code completion, simply recommending an *UNK* token offers no help to the developers. The *unknown word problem* restricts the performance of neural language models, especially when there are a large number of unique words in the corpus like program source code.

For our code completion task, we observe that when writing programs, developers tend to repeat locally. For example, the variable name `my_salary` in Figure 3.1 may be rare and marked as *UNK* with respect to the whole corpus. But within that specific code block, it repeats several times and has a relatively high frequency. Intuitively, when predicting such unknown words, our model can *learn to choose* one location in local context and copy the word at that location as our prediction. Actually, the recently proposed Pointer Networks [143] can do



```
my_salary = 0
for i in range(12):
    my_salary += 1
print(my_salary)
```

Figure 3.1: A Python program and its corresponding abstract syntax tree (AST). The dashed arrow starts from a child node and points to a parent node.

so, which employ attention scores to select a word from the input sequence as output. Although pointer networks can make better predictions on unknown words or rare words, they are unable to predict words beyond the current input sequence, i.e., lacking the global view. Therefore they may not work well in our code completion.

In this chapter, to facilitate effective code completion, we propose a *pointer mixture network*, which can predict the next word by either generating one from the global vocabulary or copying a word from the local context. For the former, we apply a standard RNN with attention, which we call the *global RNN component*. For the latter, we employ a pointer network which we call the *local pointer component*. Actually, the two components share the same RNN architecture and attention scores. Our pointer mixture network is a weighted combination of the two components. At each prediction, a switcher is learned based on the context information, which can guide the model to choose one component for generating the next word. In this way, our model learns *when and where* to copy an OoV word from the local context as the final prediction. Therefore, the main contributions of this chapter are the tailored attention mechanism and pointer mixture network. We evaluate our models on two benchmarked datasets, namely JavaScript and Python. The experimental results show great improvements upon the state-of-the-arts.

3.2 Methodology

3.2.1 Program Representation

In our corpus, each program is represented in the form of *abstract syntax tree* (AST). Any programming language has an unambiguous context-free grammar, which can be used to parse source code into an AST. Further, the AST can be converted back into source code in a one-to-one correspondence. Processing programs in the form of ASTs is a typical practice in *Software Engineering* (SE) [107, 84].
Figure 3.1 shows an example Python program and its corresponding AST. We can see that each AST node contains two attributes: the *type* of the node and an optional *value*. For each leaf node, ":" is used as a separator between type and value. For each non-leaf node, we append a special `EMPTY` token as its value. As an example, consider the AST node `NameLoad:my_salary` in Figure 3.1 where `NameLoad` denotes the type and `my_salary` is the value. The number of unique types is relative small (hundreds in our corpus), with types encoding the program structure, e.g., `Identifier,IfStatement, SwitchStatement`, etc. There are infinite possibilities for values, which encode the program text. A value may be any program identifier (e.g. `jQuery`), literal (e.g. `66`), program operator (e.g., `+,-,*`), etc.
Representing programs as ASTs rather than plain text enables us to predict the structure of the program, i.e., type of each AST node. See the example in Figure 3.1 again, when the next token is a keyword `for`, the corresponding next AST node is `For(:EMPTY)`, which corresponds to the following code block:

```
for __ in __:
  ## for loop body
```

In this way, successfully predicting next AST node completes not only the next token `for`, but also the whole code block including some trivial tokens like `in` and "`:`". Such structure completion enables more flexible code completion at different levels of granularity.
To apply statistical sequence models, we flatten each AST as a sequence of nodes in the in-order depth-first traversal. To make sure the sequence can be converted back to the original tree structure thus converted back to the source code, we allow each node type to encode two additional bits of information about whether the AST node has a child and/or a right sibling. If we define a word as $w_i = (T_i, V_i)$ to represent an AST node, with T_i being the *type* and V_i being the *value*, then each program can be denoted as a sequence of words $w_{i=1}^{n}$. Thus our code completion problem is defined as: given a sequence of words $w_1, ..., w_{t-1}$, our task is to predict the next word w_t. Obviously, we have two kinds of tasks: predicting the next node *type* T_t and predicting the next node *value* V_t. We build one model for each task and train them separately. We call this AST-based code completion.

3.2.2 Neural Language Model

The code completion task can be regarded as a language modeling problem, where recurrent neural networks (RNNs) have achieved appealing success in recent years. LSTM [63] is proposed to mitigate the gradients vanishing/exploding problem in RNNs, by utilizing gating mechanisms. A standard LSTM cell is defined as $h_t = f(x_t, h_{t-1})$. At each time step t, an LSTM cell takes current input vector x_t and previous hidden state h_{t-1} as inputs, then produces the current hidden state h_t which will be used to compute the prediction at time step t.

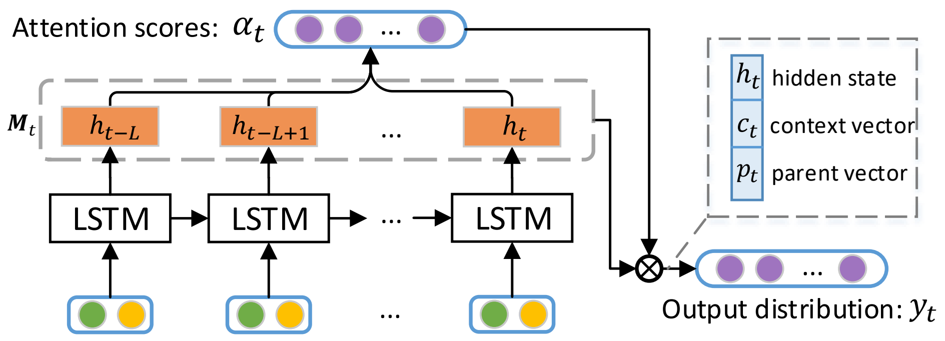


Figure 3.2: The attentional LSTM. The inputs fed to each LSTM cell are composed of two kinds of embeddings (green for Type and yellow for Value). Here $\otimes$ represents the element-wise multiplication.

3.2.3 Attention Mechanisms

Standard neural language models suffer from hidden state bottleneck [21]. To alleviate the problem, attention mechanism is proposed to retrieve and make use of *relevant* previous hidden states. It is incorporated into standard LSTM which we call attentional LSTM in this chapter, as illustrated in Figure 3.2.

Context Attention Traditional attention mechanism makes use of previous hidden states within a context window [7], which we call the *context attention*. Formally, we keep an external memory of L previous hidden states, which is denoted as $M_t = [h_{t-L}, ..., h_{t-1}] \in \mathbb{R}^{k*L}$. At time step t, the model uses an attention layer to compute the relation between h_t and hidden states in M_t, represented as attention scores α_t, and then produces a summary context vector c_t. We design our context attention for code completion as follows:

$$A_t = v^T \tanh(W^m M_t + (W^h h_t)1_L^T), \tag{3.1}$$

$$\alpha_t = softmax(A_t), \tag{3.2}$$

$$c_t = M_t \alpha_t^T, \tag{3.3}$$

where $W^m, W^h \in \mathbb{R}^{k*k}$ and $v \in \mathbb{R}^k$ are trainable parameters. k is the size of the hidden state, i.e. dimension of h_t. 1_L represents an L-dimensional vector of ones.

Parent Attention Besides the traditional context attention, we also propose a *parent attention* for the AST-based code completion. Intuitively, different hidden states within the context window should have different degrees of relevance to the current prediction. As our sequence is flattened from a tree (i.e., AST, see Figure 3.1), a parent node should be of great relevance to a child node. But the flattened AST has lost the parent-children information. To exploit such structure information, when flattening the AST, we record the parent location *pl* of each AST node, i.e., how many nodes before it. Then at time step t, our model retrieves a parent vector p_t from the external memory M_t, which is the hidden state at the parent location, i.e., h_{t-pl} [1]. The information of parent code segments can benefit our model to make more confident predictions.
When predicting next word at time step t, we condition the decision on not only the current hidden state h_t but also the context vector c_t and parent vector p_t. The output vector G_t encodes the information about next token which is then projected into the vocabulary space, followed by a softmax function to produce the final probability distribution $y_t \in \mathbb{R}^V$:

$$G_t = \tanh(W^g[h_t; c_t; p_t]), \tag{3.4}$$

$$y_t = softmax(W^v G_t + b^v), \tag{3.5}$$

where $W^g \in \mathbb{R}^{k*3k}$ and $W^v \in \mathbb{R}^{V*k}$ are two trainable projection matrices and $b^v \in \mathbb{R}^V$ is a trainable bias vector. Note

[1] If *pl* is larger than L, we set *pl* as 1.

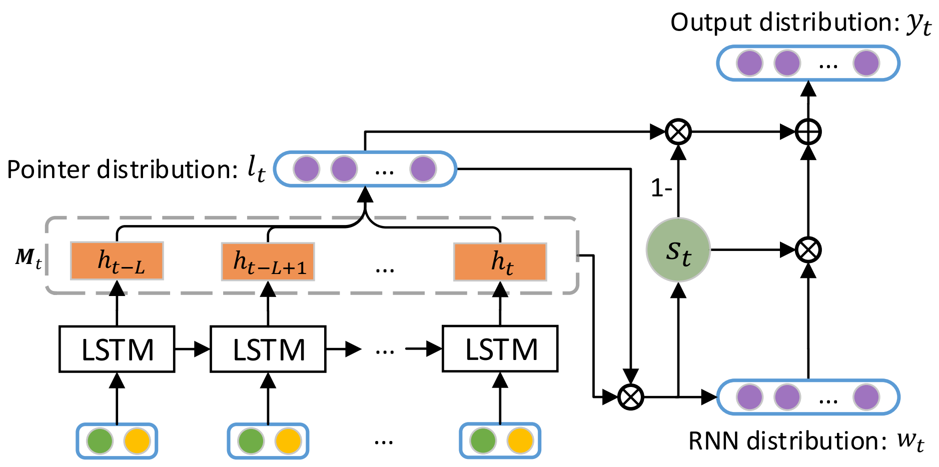


Figure 3.3: The pointer mixture network. We reuse the attention scores α_t (see Figure 3.2) as the pointer distribution l_t. The switcher produces $s_t \in [0, 1]$ to balance l_t and w_t. The final distribution is generated by concatenating the two scaled distributions. Here $\oplus$ indicates the concatenation operation.

that V represents the size of vocabulary and ";" denotes the concatenation operation.

3.2.4 Pointer Mixture Network

Inspired by the prevalence of locally repeated tokens in program source code, we propose to leverage the pointer networks to predict OoV tokens in code completion, by *copying* a token from previous input sequence. Specifically, we propose a *pointer mixture network* that combines a standard RNN and a pointer network, as shown in Figure 3.3.

Our pointer mixture network consists of two major components (global RNN component and local pointer component), and one switcher to strike a balance between them. For the global RNN component, it is an attentional LSTM that predicts the next token from a predefined global vocabulary. For the local pointer component, it points to previous locations in local context according to the learned location weights. Our pointer mixture network combines the two components by concatenating the two components' output vectors. Before concatenation, the two individual outputs are scaled by a learned switcher based on the context, thus our model learns how to *choose* a certain component at each prediction. Specifically, the switcher produces a scalar $s_t \in [0, 1]$ which indicates the probability to use the global RNN component, and then $1-s_t$ is the probability to use the local pointer component.

After concatenating the two scaled vectors, we pick one output dimension with the highest probability. If this dimension belongs to the RNN component, then the next token is generated from the global vocabulary. Otherwise, the next token is copied from the local context.

Formally, at time step t, the global RNN component produces a probability distribution $w_t \in \mathbb{R}^V$ for the next token x_t within the vocabulary according to Equation 3.5. The local pointer component points to the locations inside a memory according to the distribution $l_t \in \mathbb{R}^L$, where L is the length of the memory. In order to reduce the parameters and accelerate the training, we reuse the attention scores (see Equation 3.2) as l_t in practice. The switcher is a sigmoid function conditioned on the current hidden state h_t and context vector c_t:

$$s_t = \sigma(W^s[h_t; c_t] + b^s), \tag{3.6}$$

where $W^s \in \mathbb{R}^{2k*1}$ and $b^s \in \mathbb{R}^1$ are trainable weights. $s_t \in [0, 1]$ is a scalar to balance w_t and l_t. Finally, the model completes by concatenating the two scaled distributions to produce the final prediction:

$$y_t = [s_t w_t; (1 - s_t) l_t]. \tag{3.7}$$

	JS	PY
Training Queries	$10.7 * 10^7$	$6.2 * 10^7$
Test Queries	$5.3 * 10^7$	$3.0 * 10^7$
Type Vocabulary	95	330
Value Vocabulary	$2.6 * 10^6$	$3.4 * 10^6$

Table 3.1: Dataset Statistics

3.3 Evaluation

3.3.1 Dataset

We evaluate different approaches on two benchmarked datasets: JavaScript (JS) and Python (PY), which are summarized in Table 3.1. Collected from GitHub, both two datasets are publicly available[2] and used in previous work [15, 120, 97]. Both datasets contain 150,000 program files that are stored in their corresponding AST formats, with the first 100,000 used for training and the remaining 50,000 used for testing. After serializing each AST in the in-order depth-first traversal, we generate multiple *queries* used for training and evaluation, one per AST node, by removing the node (plus all the nodes to the right) from the sequence and then attempting to predict back the node.

The numbers of unique node *types* in JS and PY are 44 and 181 originally. By adding information about children and siblings as discussed in Section 3.2.1, we increase the numbers to 95 and 330 respectively. As shown in Table 3.1, the number of unique node *values* in both datasets are too large to directly apply neural language models, thus we only choose K most frequent values in each training set to build the global vocabulary, where K is a

[2]http://plml.ethz.ch

free parameter. We further add three special values: UNK for out-of-vocabulary values, EOF indicating the end of each program, and EMPTY being the value of non-leaf AST nodes.

3.3.2 Experimental Setup

Configuration Our base model is a single layer LSTM network with unrolling length of 50 and hidden unit size of 1500. To train the model, we use the cross entropy loss function and mini-batch SGD with the Adam optimizer [74]. We set the initial learning rate as 0.001 and decay it by multiplying 0.6 after every epoch. We clip the gradients' norm to 5 to prevent gradients exploding. The size of attention window is 50. The batch size is 128 and we train our model for 8 epochs. Each experiment is run for three times and the average result is reported.
We divide each program into segments consisting of 50 consecutive AST nodes, with the last segment being padded with EOF if it is not full. The LSTM hidden state and memory state are initialized with h_0, c_0, which are two trainable vectors. The last hidden and memory states from the previous LSTM segment are fed into the next one as initial states if both segments belong to the same program. Otherwise, the hidden and memory states are reset to h_0, c_0. We initialize h_0, c_0 to be all-zero vectors while all other variables are randomly initialized using a uniform distribution over [-0.05, 0.05]. We employ *accuracy* as our evaluation metric, i.e., the proportion of correctly predicted next node types/values.

Preprocessing ad Training Details As each AST node consists of a type and a value, to encode the node and input it to the LSTM, we train an embedding vector for each type (300 dimensions) and value (1200 dimensions) respectively, then concatenate the two embeddings into one vector. Since the number of unique types is relatively small in both datasets, there is no *unknown word problem* when predicting next AST node type. Therefore, we only apply our *pointer mixture network* on predicting next AST node value.

For each dataset, we build the global vocabulary for AST node values with K most frequent values in the training set, and mark all out-of-vocabulary node values in training set and test set as OoV values. Before training, if an OoV value appears exactly the same as another previous value within the attention window, then we label that OoV value as the corresponding *position* in the attention window. Otherwise, the OoV value is labeled as `UNK`. If there are multiple matches in the attention window, we choose the position label as the last occurrence of the matching value in the window, which is the closest one. For within-vocabulary values, we label them as the corresponding IDs in the global vocabulary. During training, whenever the ground truth of a training query is `UNK`, we set the loss function to zero for that query such that our model does not learn to predict `UNK`. In both training and evaluation, all predictions where the target value is `UNK` are treated as *wrong* predictions, i.e., decrease the overall accuracy.

We implement our models using *Tensorflow* and run our experiments on a Linux server with one NVIDIA GTX TITAN GPU. Unless otherwise stated, each experiment is run for three times and the average result is reported.

3.3.3 Experimental Results

For each experiment, we run the following models for comparison, which have been introduced in Section 3.2:

- **Vanilla LSTM**: A standard LSTM network without any attention or pointer mechanisms.
- **Attentional LSTM**: An LSTM network equipped with our (context and parent) attention mechanism which attends to the last 50 hidden states at each time step.
- **Pointer Mixture Network**: Our proposed mixture network which combines the above attentional LSTM and the pointer network.

OoV Prediction We first evaluate our pointer mixture network's ability to ease the *unknown word problem* when predicting next AST node value. For each of the two datasets, we create three specific datasets by varying the global vocabulary size K for node values to be 1k, 10k, and 50k, resulting in different out-of-vocabulary (OoV) rates. We also measure how often OoV values can occur in previous context window thus be labeled as the corresponding positions. We call this measure as *localness*, which is the upper-bound of the performance gain we can expect from the pointer component. We run the above models on each specific dataset. Table 3.2 lists the corresponding statistics and experimental results.
As Table 3.2 shows in the column, on each specific dataset, the vanilla LSTM achieves the lowest accuracy, while the attentional LSTM improves the performance upon the vanilla LSTM, and our pointer mixture network achieves the highest accuracy.

Vocabulary Size (JS) OoV Rate / Localness	JS_1k 20% / 8%	JS_10k 11% / 3.7%	JS_50k 7% / 2%
Vanilla LSTM	69.9%	75.8%	78.6%
Attentional LSTM (ours)	71.7%	78.1%	80.6%
Pointer Mixture Network (ours)	**73.2%**	**78.9%**	**81.0%**
Vocabulary Size (PY)	PY_1k	PY_10k	PY_50k
OoV Rate / Localness	24% / 9.3%	16% / 5.2%	11% / 3.2%
Vanilla LSTM	63.6%	66.3%	67.3%
Attentional LSTM (ours)	64.9%	68.4%	69.8%
Pointer Mixture Network (ours)	**66.4%**	**68.9%**	**70.1%**

Table 3.2: Accuracies on next *value* prediction with different vocabulary sizes. The out-of-vocabulary (OoV) rate denotes the percentage of AST nodes whose value is beyond the global vocabulary. *Localness* is the percentage of values who are OoV but occur in the context window.

Besides, we can see that by increasing the vocabulary size in JS or PY dataset, the OoV rate decreases, and the general accuracies on different models increase due to more available information. We also notice a performance gain by our pointer mixture network over the attentional LSTM, and the gain is the largest with 1k vocabulary size. We attribute this performance gain to correctly predicting some OoV values through the local pointer component. Therefore, the results demonstrate the effectiveness of our pointer mixture network to predict OoV values, especially when the vocabulary is small and the OoV rate is large.

State-of-the-Art Comparison As there are already prior investigations conducting code completion on the two benchmarked datasets, to validate the effectiveness of our proposed approaches, we need to compare them against the state-of-the-art.

	JS		**PY**	
	TYPE	VALUE	TYPE	VALUE
Vanilla LSTM	87.1%	78.6%	79.3%	67.3%
Attentional LSTM (no parent)	88.1%	80.5%	80.2%	69.8%
Attentional LSTM (ours)	**88.6%**	80.6%	**80.6%**	69.8%
Pointer Mixture Network (ours)	-	81.0%	-	**70.1%**
LSTM [97]	84.8%	76.6%	-	-
Probabilistic Model [120]	83.9%	**82.9%**	76.3%	69.2%

Table 3.3: Comparisons against the state-of-the-arts. The upper part is the results from our experiments while the lower part is the results from the prior work. TYPE means next node type prediction and VALUE means next node value prediction.

Particularly, Liu et al. [97] employ a standard LSTM on the JS dataset, without attention or pointer mechanisms. Raychev et al. [120] build a probabilistic model for code based on probabilistic grammars and achieve the state-of-the-art accuracies for code completion on the two datasets.

Specifically, we conduct experiments on next AST node type prediction and next AST node value prediction respectively. For the former, there is no *unknown word problem* due to the small type vocabulary, so we only use the vanilla LSTM and the attentional LSTM. For the latter, we set the value vocabulary size to 50k to make the results comparable with [97], and employ all the three models. The results are shown in Table 3.3.

The upper part of Table 3.3 shows our results in this work, while the lower part lists the results from the prior work. Note that Liu et al. [97] only apply LSTM on the JS dataset, so they do not have results on the PY dataset. For next type prediction, our attentional LSTM achieves the highest accuracy on both datasets, significantly improving the best records of the two datasets. For next value prediction on JS dataset, our pointer mixture network achieves comparable performance with Raychev et al.'s [120], which is a probabilistic model based on domain-specific grammars. However, our approaches outperform Liu et al. [97] that is also based on neural networks. On PY dataset, our pointer mixture network for next value prediction outperforms the previous best record. Therefore, we conclude that our attentional LSTM and pointer mixture network are effective for code completion, achieving three state-of-the-art performances out of the four tasks.

3.3.4 Discussion

More explanations about experimental results. From Table 3.2 and Table 3.3, we observe that the accuracies produced in JS dataset are consistently higher than the accuracies in PY dataset, whether in type prediction or value prediction. We attribute this difference to the fact that JS dataset contains more data to train the model and fewer categories to predict than the PY dataset, as shown in Table 3.1. The work by Raychev et al. [120] also confirms this accuracy gap between the two datasets. Besides, from Table 3.3 we notice that our vanilla LSTM outperforms the work by Liu et al. [97] a lot who also apply a simple LSTM. We think the main reason lies in the different formulation of loss function, where Liu et al. define a loss function for both type and value prediction and train a model for them together. On the contrary, we define one loss function for each task and train them separately, which is much easier to train.

	JS_1k	PY_1k
Pointer Random Network	71.4%	64.8%
Attentional LSTM	71.7%	64.9%
Pointer Mixture Network	**73.2**%	**66.4**%

Table 3.4: Showing why pointer mixture network works.

Why attention mechanism works? When writing programs, it is quite common to refer to a variable identifier declared many lines before. In this work, the mean program length (i.e., the number of AST nodes) is around 1000 in JS dataset and 600 in PY dataset. Therefore in our code completion task, we need the attention mechanism to capture the long dependencies. Furthermore, we measure how our proposed *parent attention* influence the final prediction by only using the context attention (see Equation 3.4). As shown in Table 3.3, parent attention can effectively contribute to the type prediction while has little effect on the value prediction.

Why pointer mixture network works? In both training and evaluation, all predictions where the target value is `UNK` are treated as *wrong* predictions. After incorporating the pointer network, we predict OoV values by copying a value from local context and that copied value may be the correct prediction. Thus we observe a performance gain in our pointer mixture network. However, one may argue that no matter how capable the pointer component is, the accuracy will definitely increase as long as we get chances to predict OoV values.

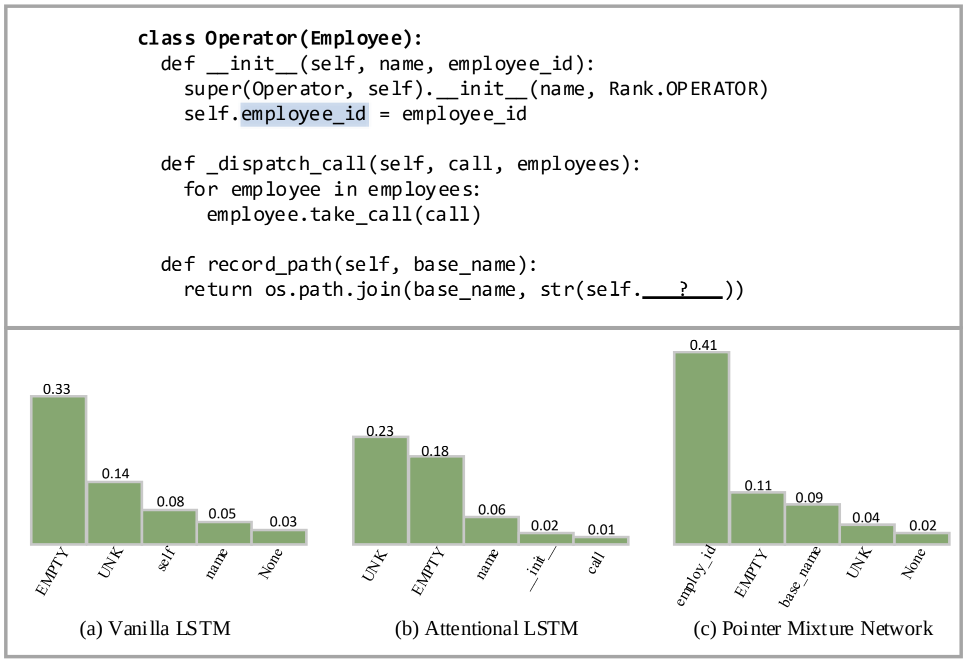
To verify the copying ability of our pointer component, we develop a *pointer random network* where the pointer distribution l_t (see Figure 3.3) is a random distribution instead of reusing the learned attention scores. We conduct comparisons on value prediction in JS and PY datasets with 1k vocabulary size. The results are listed in Table 3.4, where the pointer random network achieves lower accuracies than the pointer mixture network. Thus we demonstrate that our pointer mixture network indeed learns *when and where* to copy some OoV values. However, the pointer random network performs even worse than the attentional LSTM. We think the reason lies in the switcher which is disturbed by the random noise and cannot always choose the correct component (i.e., the RNN component), thus influencing the overall performance.

3.3.5 Case Study

We depict a code completion example in Figure 3.4. In this example, the target prediction `employee_id` is an OoV value with respect to the whole training corpus. We show the top five predictions of each model. For vanilla LSTM, it just produces `EMPTY` which is the most frequent node value in our corpus. For attentional LSTM, it learns from the context that the target has a large probability to be `UNK`, but fails to produce the real value. The pointer mixture network successfully identifies the OoV value from the context, as it observes the value appearing in the previous code.

3.4 Related Work

Statistical Code Completion has been a long-term research topic. Much of this work is inspired by Hindle et al. [58], who are the first to demonstrate that real programs written by real people have rich statistical properties and employed n-gram models to



```
class Operator(Employee):
  def __init__(self, name, employee_id):
    super(Operator, self).__init__(name, Rank.OPERATOR)
    self.employee_id = employee_id

  def _dispatch_call(self, call, employees):
    for employee in employees:
      employee.take_call(call)

  def record_path(self, base_name):
    return os.path.join(base_name, str(self.___?___))
```

Figure 3.4: A code completion example showing predicting an OoV value.

accomplish a code completion task. Tu et al. [141] extend Hindle et al.'s work by adding a cache mechanism. There is a body of recent work that explores the application of statistical learning and sequence models on the code completion task, such as probabilistic grammars [4, 15, 120]. Recently, neural networks become very popular to model source code [121, 146, 3]. In particular, Bhoopchand et al. [14] propose a sparse pointer mechanism for RNN, to better predict identifiers in Python source code. Nevertheless, their pointer component targets at identifiers in Python source code, rather than OoV tokens in our work. The OoV tokens include not only identifiers but also other types such as VariableDeclarator. Besides, they directly serialize each program as a sequence of code tokens, while in our corpus each program is represented as a sequence of AST nodes to facilitate more intelligent structure prediction.

3.5 Summary

In this chapter, we apply neural language models on the code completion task, and develop an attention mechanism that exploits the parent-children information on the program's AST. To deal with the OoV values in code completion, we propose a pointer mixture network which learns to either generate a new value through an RNN component or copy an OoV value from local context through a pointer component. Experimental results demonstrate the effectiveness of our approaches.

□ **End of chapter.**

Chapter 4

Exploiting Diversity for Multi-Head Attention

Multi-head attention is one of the key components in the self-attention mechanism, with the appealing ability to jointly attend to information from *different* representation subspaces at *different* positions. In this chapter, we explore effective approaches to better exploit such *diversity* to improve multi-head attention. The main points of this chapter are as follows. (1) We propose three disagreement regularizations to explicitly encourage the diversity among multiple attention heads, which are respectively applied to the subspace, the attention positions, and the output representations associated with each head. (2) We propose to employ the routing-by-agreement algorithm to better aggregate the diverse information distributed in the extracted partial-representations. (3) We evaluate our proposed approaches on machine translation tasks and sentence encoding tasks where we achieve promising results.

4.1 Introduction

Attention model has become a standard component of today's deep learning networks, contributing to impressive results in machine translation [7, 99], image captioning [152], speech recognition [25], among many other applications. Recently, the performance of attention is further improved by multi-head mechanism [142], which concurrently performs the attention functions on different representation subspaces of the input sequence. Consequently, different attention heads are able to capture distinct properties of the input, which are embedded in different subspaces [118]. Subsequently, a linear transformation is generally employed to aggregate the partial representations extracted by different attention heads [142, 2], producing the final output representation.

However, the conventional multi-head mechanism may not fully exploit the *diversity* among attention heads. First, one strong point of multi-head attention is the ability to jointly attend to information from *different* representation subspaces at *different* positions. But currently there is no mechanism to guarantee that different attention heads indeed capture distinct information. Second, we believe that information extraction and information aggregation are both important to produce an informative representation. We argue that the straightforward linear transformation is not expressive enough to fully capture the rich information distributed in the extracted partial-representations. In this chapter, we propose two strategies to better exploit the diversity of multi-head attention, namely **disagreement regularization** and **advanced aggregation function**.

In response to the first problem, we introduce a disagreement regularization term to explicitly encourage the diversity among multiple attention heads. The disagreement regularization serves as an auxiliary objective to guide the training of the related attention component. Specifically, we propose three types of disagreement regularization, which are applied to the three key components that refer to the calculation of information vector using multi-head attention. Two regularization terms are respectively to maximize cosine distances of the input subspaces and output representations, while the last one is to disperse the positions attended by multiple heads with element-wise multiplication of the corresponding attention matrices. The three regularization terms can be either used individually or in combination.

To address the second problem, we replace the standard linear transformation in conventional multi-head attention [142] with an advanced routing-by-agreement algorithm, to better aggregate the diverse information distributed in the extracted partial-representations. Specifically, we cast information aggregation as the *assigning-parts-to-wholes* problem [60], and investigate the effectiveness of the routing-by-agreement algorithm, which is an appealing alternative to solving this problem [123, 61]. The routing algorithm iteratively updates the proportion of how much a part should be assigned to a whole, based on the agreement between parts and wholes.

In addition, it is natural to combine the two types of approaches and apply them simultaneously, as the former focuses on extracting more diverse information while the latter aims to better aggregate the extracted information.

We evaluate the performance of the proposed approaches on both machine translation tasks as well as sentence encoding tasks. For machine translation, we validate our approaches on top of the advanced Transformer model [142] on both WMT14 English⇒German and WMT17 Chinese⇒English data. Experimental results show that our approaches consistently improve the translation performance across language pairs while keeping computational efficiency. For sentence encoding, we evaluate with the linguistic probing tasks [29], which consist of 10 classification problems to study what linguistic properties are captured by input representations. Probing analysis shows that our approaches indeed produce more informative representation, which embeds more syntactic and semantic information. Precisely, our study reveals that:

- Directly applying disagreement regularization on the output representations of multiple attention heads is most effective.
- The EM routing algorithm shows its superiority on information aggregation over the standard linear transformation and other aggregation algorithms.
- Disagreement regularization and advanced aggregation function are complementary to each other, as indicated from analyses in machine translation and sentence encoding.

4.2 Multi-Head Attention

Attention mechanism aims at modeling the relevance between representation pairs, thus a representation is allowed to build a direct relation with another representation. Instead of performing a single attention function, Vaswani et al. [142] found it is beneficial to capture different context features with multiple

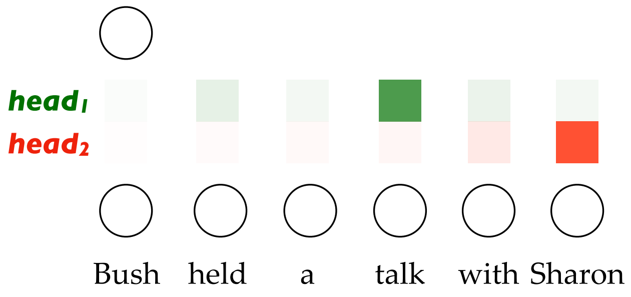


Figure 4.1: Illustration of the multi-head attention, which jointly attends to different representation subspaces (colored boxes) at different positions (darker color denotes higher attention probability).

individual attention functions, namely multi-head attention. Figure 4.1 shows an example of a two-head attention model. For the query word "Bush", green and red heads pay attention to different positions of "talk" and "Sharon".
Formally, attention function maps a sequence of query $\mathbf{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$ and a set of key-value pairs which are denoted by $\{\mathbf{K}, \mathbf{V}\} = \{(\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_m, \mathbf{v}_m)\}$ to outputs, where $\mathbf{Q} \in \mathbb{R}^{n \times d}$, $\{\mathbf{K}, \mathbf{V}\} \in \mathbb{R}^{m \times d}$. More specifically, multi-head attention model first transforms $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ into H subspaces with different, learnable linear projections:

$$\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h = \mathbf{Q}\mathbf{W}_h^Q, \mathbf{K}\mathbf{W}_h^K, \mathbf{V}\mathbf{W}_h^V, \tag{4.1}$$

where $\{\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h\}$ are respective the query, key, and value representations of the h-th head. $\{\mathbf{W}_h^Q, \mathbf{W}_h^K, \mathbf{W}_h^V\} \in \mathbb{R}^{d \times \frac{d}{H}}$ denote parameter matrices associated with the h-th head, where d represents the dimensionality of the model hidden states. Furthermore, H attention functions are applied in parallel to produce the output states $\{\mathbf{O}_1, \ldots, \mathbf{O}_H\}$, among them:

$$\mathbf{O}_h = \textsc{Att}(\mathbf{Q}_h, \mathbf{K}_h)\mathbf{V}_h, \tag{4.2}$$

where $\mathbf{O}_h \in \mathbb{R}^{n \times \frac{d}{H}}$, $\textsc{Att}(\cdot)$ is an attention model to produce

the alignment matrix A. In this chapter, we use scaled dot-product attention [99], which achieves similar performance with its additive counterpart [7] while is much faster and more space-efficient in practice [142].
Finally, the H output states are concatenated and linear transformed to produce the final state:

$$\begin{aligned} \text{Concat:} \quad \widehat{\mathbf{O}} &= [\mathbf{O}_1, \dots, \mathbf{O}_H], && (4.3) \\ \text{Linear:} \quad \mathbf{O} &= \widehat{\mathbf{O}}\mathbf{W}^O, && (4.4) \end{aligned}$$

where $\mathbf{O} \in \mathbb{R}^{n \times d}$ denotes the final output states, $\mathbf{W}^O \in \mathbb{R}^{d \times d}$ is a trainable parameter matrix.

4.3 Methodology

In this chapter, we propose to better exploit the diversity of multi-head attention from two perspectives:

- *Disagreement Regularization*: Conventional multi-head attention conducts multiple attention functions in parallel (Equation 4.2), while there is no mechanism to guarantee that different attention heads indeed capture distinct information. In response to this problem, we introduce disagreement regularizations to explicitly encourage different attention heads to extract distinct information (Section 4.3.1);

- *Advanced Aggregation Function*: As shown in Equations 4.3 and 4.4, the standard multi-head attention uses a straightforward concatenation and linear mapping to aggregate the partial-representations captured by multiple attention heads. We argue that this straightforward strategy may not fully exploit the expressiveness of multi-head attention, which can benefit from advanced information aggregation. In this chapter, we exploit a more advanced routing-by-agreement method to aggregate the information extracted by different attention heads (Section 4.3.2).

The disagreement regularization encourages multiple attention functions to extract different information, and advanced aggregation function helps better aggregate the extracted information. Therefore, the two approaches are complementary to each other and can be employed simultaneously, which we will describe in Section 4.3.3.

4.3.1 Disagreement Regularization

Multi-head attention allows the model to jointly attend to information from *different* representation subspaces at *different* positions. To further guarantee the diversity, we enlarge the distances among multiple attention heads with disagreement regularization. To this end, we introduce an auxiliary regularization term in order to encourage the diversity among multiple attention heads. Taking the machine translation task as example, the training objective is revised as:

$$J(\theta) = \arg\max_{\theta} \Big\{ \underbrace{L(\mathbf{y}|\mathbf{x};\theta)}_{likelihood} + \lambda * \underbrace{D(\mathbf{a}|\mathbf{x},\mathbf{y};\theta)}_{disagreement} \Big\},$$

where $\mathbf{a}$ is the referred attention matrices, λ is a hyper-parameter and is empirically set to 1.0 in this chapter. The auxiliary regularization term $D(\cdot)$ guides the related attention component to capture different features from the corresponding projected subspaces. Note that the introduced regularization term works like $L1$ and $L2$ terms, which do not introduce any new parameters and only influence the training of the standard model parameters.

Specifically, we propose three types of disagreement regularization to encourage each head vector $\mathbf{O}_h$ to be different from other heads:

- **Disagreement on Subspaces (Sub.)** This disagreement is designed to maximize the cosine distance between the projected values. Specifically, we first calculate the cosine similarity $\cos(\cdot)$ between the vector pair V^i and V^j in different value subspaces, through the dot product of the normalized vectors[1], which measures the cosine of the angle between V^i and V^j. Thus, the cosine distance is defined as negative similarity, i.e, $-\cos(\cdot)$. Our training objective is to enlarge the average cosine distance among all head pairs. The regularization term is formally expressed as:

$$D_{subpace} = -\frac{1}{H^2} \sum_{i=1}^{H} \sum_{j=1}^{H} \frac{V^i \cdot V^j}{\|V^i\| \|V^j\|}. \tag{4.5}$$

- **Disagreement on Attended Positions (Pos.)** Another strategy is to disperse the attended positions predicted by multiple heads. Inspired by the agreement regularization [90, 22] which encourages multiple alignments to be similar, in this chapter, we deploy a variant of the original term by introducing an alignment disagreement regularization. Formally, we employ the sum of element-wise multiplication of corresponding matrix cells[2], to measure

[1]We did not employ the Euler Distance between vectors since we do not care the absolute value in each vector.

[2]We also used the squared element-wise subtraction of two matrices in our preliminary experiments, and found it underperforms its multiplication counterpart, which is consistent with the results in [22].

the similarity between two alignment matrices A^i and A^j ($\textsc{Att}(\cdot)$ in Equation 4.2) of two heads:

$$D_{position} = -\frac{1}{H^2}\sum_{i=1}^{H}\sum_{j=1}^{H}|A^i \odot A^j|. \tag{4.6}$$

- **Disagreement on Outputs (Out.)** This disagreement directly applies regularization on the outputs of each attention head, by maximizing the difference among them. Similar to the *subspace* strategy, we employ negative cosine similarity to measure the distance:

$$D_{output} = -\frac{1}{H^2}\sum_{i=1}^{H}\sum_{j=1}^{H}\frac{O^i \cdot O^j}{\|O^i\|\|O^j\|}. \tag{4.7}$$

4.3.2 Advanced Aggregation Function

Information aggregation in multi-head attention (e.g. Equations 4.3 and 4.4) aims at composing the partial representations captured by different attention heads to a final representation. Recent work shows that representation composition benefits greatly from advanced functions beyond simple concatenation or mean/max pooling [44, 11, 36]. In this chapter, we cast information aggregation in multi-head attention as the problem of *assigning-parts-to-wholes*, to which an appealing solution is the *routing-by-agreement* algorithm, as shown in Figure 4.2.

The routing algorithm consists of two layers: *input capsules* and *output capsules*. The input capsules are constructed from the transformation of the partial representations extracted by different attention heads. For each output capsule, each input capsule proposes a distinct "voting vector", which represents the proportion of how much the information is transformed from this input capsule (i.e. parts) to the corresponding output capsule

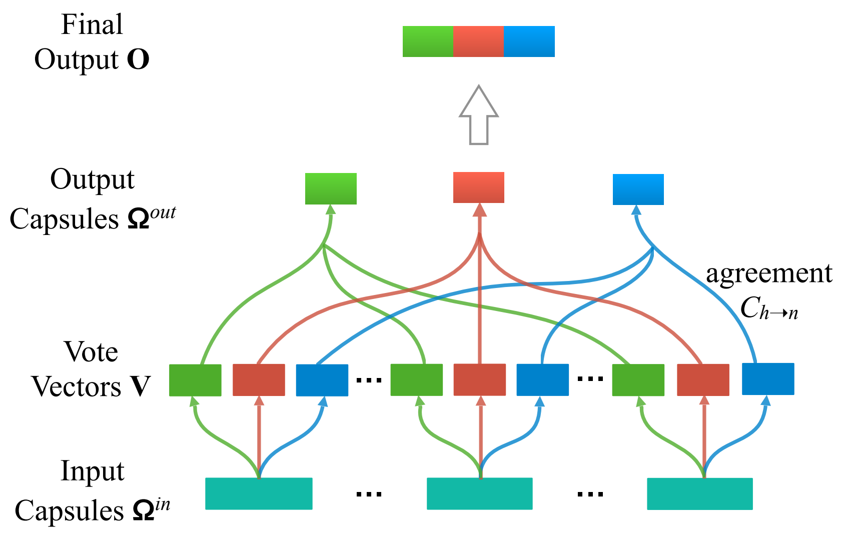


Figure 4.2: Illustration of routing-by-agreement.

(i.e. wholes). The proportion is iteratively updated based on the agreement between the voting vectors and the output capsule. Finally, all output capsules are concatenated to form the final representation.

Mathematically, the input capsules $\mathbf{\Omega}^{in} = \{\mathbf{\Omega}^{in}_1, \dots, \mathbf{\Omega}^{in}_H\}$ with $\mathbf{\Omega}^{in} \in \mathbb{R}^{n\times d}$ are constructed from the outputs of multi-head attention:

$$\mathbf{\Omega}^{in}_h = f_h(\widehat{\mathbf{O}}), \tag{4.8}$$

where $f_h(\cdot)$ is a distinct non-linear transformation function associated with the input capsule $\mathbf{\Omega}^{in}_h$. Given N output capsules, each input capsule $\mathbf{\Omega}^{in}_h$ propose N "vote vectors" $\mathbf{V}_{h\rightarrow *} = \{\mathbf{V}_{h\rightarrow 1}, \dots, \mathbf{V}_{h\rightarrow N}\}$, which is calculated by

$$\mathbf{V}_{h\rightarrow n} = \mathbf{\Omega}^{in}_h \mathbf{W}_{h\rightarrow n}. \tag{4.9}$$

Each output capsule $\mathbf{\Omega}^{out}_n$ is calculated as the normalization of its total input, which is a weighted sum over all the incoming

Algorithm 1 Iterative Simple Routing.

```
1: procedure ROUTING($\mathbf{V}$, $T$):
2:     $\forall \mathbf{V}_{h\to *}$: $B_{h\to n} = 0$
3:     for $T$ iterations do
4:         $\forall \mathbf{V}_{h\to *}$: $C_{h\to n} = \frac{\exp(B_{h\to n})}{\sum_{n'=1}^{N}\exp(B_{h\to n'})}$
5:         $\forall \mathbf{\Omega}_n^{out}$: compute $\mathbf{\Omega}_n^{out}$ by Eq. 4.10
6:         $\forall \mathbf{V}_{h\to *}$: $B_{h\to n}$ += $\mathbf{\Omega}_n^{out} \cdot \mathbf{V}_{h\to n}$
    return $\mathbf{\Omega}$
```

"vote vectors" $\mathbf{V}_{*\to n}$:

$$\mathbf{\Omega}_n^{out} = \frac{\sum_{h=1}^{H} C_{h\to n}\mathbf{V}_{h\to n}}{\sum_{h=1}^{H} C_{h\to n}}. \tag{4.10}$$

The weight $C_{h\to n}$ with $\sum_n C_{h\to n} = 1$ measures the agreement between vote vector $\mathbf{V}_{h\to n}$ and output capsule $\mathbf{\Omega}_n^{out}$, which is determined by the iterative routing as described in the next section. Note that $\sum_{h=1}^{H} C_{h\to n}$ is not necessarily equal to 1. After the routing process, we concatenate the N output capsules to form the final representation: $\mathbf{O} = [\mathbf{\Omega}_1^{out}, \ldots, \mathbf{\Omega}_N^{out}]$. To make the dimensionality of the final output be consistent with that of hidden layer (i.e. d), we set the dimensionality of each output capsule be $\frac{d}{N}$.
In this chapter, we explore two representative routing mechanisms, namely *simple routing* [123] (Section 4.3.2) and *EM routing* [61] (Section 4.3.2), which differ at how the agreement weights $C_{h\to n}$ are calculated.

Simple Routing

Algorithm 1 lists a straightforward implementation of routing. $B_{h\to n}$ measures the degree that the input capsule $\mathbf{\Omega}_h^{in}$ should be coupled to the output capsule $\mathbf{\Omega}_h^{in}$, which is initialized

as all 0 (Line 2). The agreement weights $C_{h\to n}$ are then iteratively refined by measuring the agreement between the vote vector $\mathbf{V}_{h\to n}$ and the output capsule $\mathbf{\Omega}_n^{out}$ (Lines 4-6), which is implemented as a simple scalar product $\mathbf{\Omega}_n^{out} \cdot \mathbf{V}_{h\to n}$ (Line 5).
To represent the probability that the output capsule $\mathbf{\Omega}_n^{out}$ is activated, we follow Sabour et al. [123] use a non-linear "squashing" function:

$$\mathbf{\Omega}_n^{out} = \frac{||\mathbf{\Omega}_n^{out}||^2}{1 + ||\mathbf{\Omega}_n^{out}||^2} \frac{\mathbf{\Omega}_n^{out}}{||\mathbf{\Omega}_n^{out}||}, \tag{4.11}$$

The scalar product $\mathbf{\Omega}_n^{out} \cdot \mathbf{V}_{h\to n}$ saturates at 1, which makes it insensitive to the difference between a quite good agreement and a very good agreement. In response to this problem, Hinton et al. [61] propose a novel Expectation-Maximization (EM) routing algorithm.
Comparing with simple routing, EM routing has two modifications. First, it explicitly assigns an activation probability A to represent the probability of whether each output capsule is activated, rather than the length of vector calculated by a squashing function (Equation 4.11). Second, it casts the routing process as fitting a mixture of Gaussians using EM, where the output capsules play the role of Gaussians and the means of the input capsules play the role of the datapoints. Accordingly, EM routing can better estimate the agreement by allowing activated output capsules to receive a cluster of similar votes.

EM Routing

Algorithm 2 lists the EM routing, which iteratively adjusts the means, variances, and activation probabilities $(\boldsymbol{\mu}, \boldsymbol{\sigma}, A)$ of the output capsules, as well as the agreement weights C of the input

Algorithm 2 Iterative EM Routing.

```
1: procedure EM ROUTING($\mathbf{V}$, $T$):
2:     $\forall \mathbf{V}_{h\to *}$: $C_{l\to n} = 1/N$
3:     for $T$ iterations do
4:         $\forall \mathbf{\Omega}_n^{out}$: M-STEP($\mathbf{V}$, $C$)    ▷ hold $C$ constant, adjust $(\boldsymbol{\mu}_n, \boldsymbol{\sigma}_n, A_n)$
5:         $\forall \mathbf{V}_{h\to *}$: E-STEP($\mathbf{V}$, $\boldsymbol{\mu}, \boldsymbol{\sigma}, A$) ▷ hold $(\boldsymbol{\mu}, \boldsymbol{\sigma}, A)$ constant, adjust $C_{h\to *}$
6:     $\forall \mathbf{\Omega}_n^{out}$: $\mathbf{\Omega}_n^{out} = A_n * \boldsymbol{\mu}_n$
       return $\mathbf{\Omega}$
```

capsules (Lines 4-5). The representation of output capsule $\mathbf{\Omega}_n^{out}$ is calculated as

$$\mathbf{\Omega}_n^{out} = A_n * \boldsymbol{\mu}_n = A_n * \frac{\sum_{h=1}^{H} C_{h\to n}\mathbf{V}_{h\to n}}{\sum_{h=1}^{H} C_{h\to n}}, \tag{4.12}$$

The EM algorithm alternates between an E-step and an M-step. The E-step determines, for each datapoint (i.e. input capsule), the probability of agreement (i.e. C) between it and each of the Gaussians (i.e. output capsules). The M-step holds the agreement weights constant, and for each Gaussian (i.e. output capsule) consists of finding the mean of these weighted datapoints (i.e. input capsules) and the variance about that mean.

M-Step for each Gaussian (i.e. $\mathbf{\Omega}_n^{out}$) consists of finding the mean $\boldsymbol{\mu}_n$ of the votes from input capsules and the variance $\boldsymbol{\sigma}_n$ about that mean:

$$\boldsymbol{\mu}_n = \frac{\sum_{h=1}^{H} C_{h\to n}\mathbf{V}_{h\to n}}{\sum_{h=1}^{H} C_{h\to n}}, \tag{4.13}$$

$$(\boldsymbol{\sigma}_n)^2 = \frac{\sum_{h=1}^{H} C_{h\to n}(\mathbf{V}_{h\to n} - \boldsymbol{\mu}_n)^2}{\sum_{h=1}^{H} C_{h\to n}}. \tag{4.14}$$

The incremental cost of using an active capsule $\mathbf{\Omega}_n^{out}$ is

$$\chi_n = \sum_i \big(\log(\boldsymbol{\sigma}_n^i) + \frac{1+\log(2\pi)}{2}\big)\sum_{h=1}^{H} C_{h\to n},$$

where $\boldsymbol{\sigma}_n^i$ denotes the i-th dimension of the variance vector $\boldsymbol{\sigma}_n$. The activation probability of capsule $\mathbf{\Omega}_n^{out}$ is calculated by

$$A_n = logistic\big(\lambda(\beta_A - \beta_\mu \sum_{h=1}^{H} C_{h\to n} - \chi_n)\big),$$

where β_A is a fixed cost for coding the mean and variance of $\mathbf{\Omega}_n^{out}$ when activating it, β_μ is another fixed cost per input capsule when not activating it, and λ is an inverse temperature parameter set with a fixed schedule. We refer the readers to [61] for more details.

E-Step adjusts the assignment probabilities $C_{h\to *}$ for each input $\mathbf{\Omega}_h^{in}$. First, we compute the negative log probability density of the vote $\mathbf{V}_{h\to n}$ from $\mathbf{\Omega}_h^{in}$ under the Gaussian distribution fitted by the output capsule $\mathbf{\Omega}_n^{out}$ it gets assigned to:

$$P_{h\to n} = \sum_i \frac{1}{\sqrt{2\pi(\boldsymbol{\sigma}_n^i)^2}} \exp(-\frac{(\mathbf{V}_{h\to n}^i - \boldsymbol{\mu}_n^i)^2}{2(\boldsymbol{\sigma}_n^i)^2}).$$

Again, i denotes the i-th dimension of the vectors $\{\mathbf{V}_{h\to n}, \boldsymbol{\mu}_n, \boldsymbol{\sigma}_n\}$. Accordingly, the agreement weight is re-normalized by

$$C_{h\to n} = \frac{A_n P_{h\to n}}{\sum_{n'=1}^{N} A_{n'} P_{h\to n'}}. \tag{4.15}$$

4.3.3 Combining Together

Coupling different representations with diversity is a well-known technique to improve the performance [69]. While disagreement regularization focuses on adjusting the training objective, i.e.

the loss function, advanced aggregation function aims at modifying the network architecture. In terms of functionality, they are also complementary to each other as one improves information extraction and the other benefits information aggregation. Therefore, it is natural to combine the two approaches and apply them simultaneously. In consideration of computation cost, we first respectively choose the best strategy from the two kinds of approach, and then apply them simultaneously by modifying both training objective and network architecture.

4.4 Evaluation on Machine Translation

In following sections, we evaluate the performance of our approaches on both machine translation tasks (Section 4.4) and sentence encoding tasks (Section 4.5). We conduct evaluation study of the proposed approaches on the benchmark machine translation tasks, and carry out final evaluation on both translation and sentence encoding tasks.

4.4.1 Setup

Data We conduct experiments on the widely-used WMT2014 English⇒German (En⇒De) and WMT2017 Chinese⇒English (Zh⇒En) translation tasks. For the En⇒De task, the dataset consists of 4.6M sentence pairs. We use newstest2013 as the development set and newstest2014 as the test set. For the Zh⇒En task, we use all of the available parallel data with maximum length limited to 50, consisting of about 20.6M sentence pairs. We use newsdev2017 as the development set and newstest2017 as the test set. We employ byte-pair encoding [125] with 32K merge operations for both language pairs. We use the case-sensitive 4-gram NIST BLEU score [111] as evaluation metric, and bootstrap resampling [77] for statistical significance test.

Models We implement the proposed approaches on top of the advanced TRANSFORMER model [142]. We follow Vaswani et al. [142] to set the configurations and have reproduced their reported results on the En⇒De task. The *Base* and *Big* models differ at hidden size (512 vs. 1024) and number of attention heads (8 vs. 16). All the models are trained on eight NVIDIA P40 GPUs where each is allocated with a batch size of 4096 tokens.
TRANSFORMER consists of three attention components: encoder self-attention, decoder self-attention and encoder-decoder attention, all of which are implemented as multi-head attention. For the information aggregation in multi-head attention, we replace the standard linear transformation with the proposed routing mechanisms. We experimentally set the number of iterations to 3 and the number of output capsules as model hidden size, which outperform other configurations during our investigation.

4.4.2 Evaluation Study on Disagreement

Effect of Regularization Terms In this section, we evaluate the impact of different regularization terms on the Zh⇒En task using TRANSFORMER-BASE. For simplicity and efficiency, here we only apply regularizations on the encoder side. As shown in Table 4.1, all the models with the proposed disagreement regularizations (Rows 2-4) consistently outperform the vanilla TRANSFORMER (Row 1). Among them, the *Output* term performs best which is +0.65 BLEU score better than the baseline model, the *Position* term is less effective than the other

#	**Regularization**			**Speed**	**BLEU**
	Sub.	*Pos.*	*Out.*		
1	×	×	×	1.21	24.13
2	✓	×	×	1.15	24.64
3	×	✓	×	1.14	24.42
4	×	×	✓	1.15	**24.78**
5	✓	×	✓	1.12	24.73
6	✓	✓	×	1.11	24.38
7	✓	✓	✓	1.05	24.60

Table 4.1: Effect of regularization terms, which are applied to the encoder self-attention only. "Speed" denotes the training speed (steps/second). Results are reported on the WMT17 Zh⇒En translation task using TRANSFORMER-BASE.

two. In terms of training speed, we do not observe obvious decrease, which in turn demonstrates the advantage of our disagreement regularizations.
However, the combinations of different disagreement regularizations fail to further improve translation performance (Rows 5-7). One possible reason is that different regularization terms have overlapped guidance, and thus combining them does not introduce too much new information while makes training more difficult.

Effect on Attention Components The TRANSFORMER consists of three attention networks, including encoder self-attention, decoder self-attention, and encoder-decoder attention. In this experiment, we investigate how each attention network benefits from the disagreement regularization. As seen from Table 4.2, all models consistently improve upon the baseline model. When applying disagreement regularization to all three attention networks, we achieve the best performance, which is +0.72 BLEU

Applying to			**Speed**	**BLEU**
Enc	*Enc-Dec*	*Dec*		
×	×	×	1.21	24.13
✓	×	×	1.15	24.78
✓	✓	×	1.10	24.67
✓	×	✓	1.11	24.69
✓	✓	✓	1.06	**24.85**

Table 4.2: Effect of regularization on different attention networks, i.e., encoder self-attention ("*Enc*"), encoder-decoder attention ("*Enc-Dec*"), and decoder self-attention ("*Dec*"). We use *Output Disagreement* as the regularization term. Results are reported on the WMT17 Zh⇒En translation task using Transformer-Base.

score better than the baseline model. The training speed decreases by 12%, which is acceptable considering the performance improvement.

In the following sections, we apply the *Output Disagreement* to all the three attention networks, which we term "Disagreement".

4.4.3 Evaluation Study on Aggregation

Table 4.3 lists the results on the En⇒De task with Transformer-Base. As seen, the proposed routing mechanism outperforms the standard aggregation in all cases, demonstrating the necessity of advanced aggregation functions for multi-head attention.

Routing Mechanisms (Rows 3-4) We first apply simple routing and EM routing to encoder self-attention. Both strategies perform better than the standard multi-head aggregation (Row 1), verifying the effectiveness of the non-linear aggregation mechanisms. Specifically, the two strategies require comparable parameters and computational speed, but EM routing achieves better performance on translation qualities. Considering the

#	**Applying to ...**			**Routing**	**Para.**	**Speed**	**BLEU**	△
1	*Enc*	*E-D*	*Dec*					
2	×	×	×	n/a	88.0M	1.92	27.31	–
3	✓	×	×	Simple	+12.6M	1.23	27.98	+0.67
4	✓	×	×	EM	+12.6M	1.20	28.28	+0.97
5	×	✓	×	EM	+12.6M	1.20	27.94	+0.63
6	×	×	✓	EM	+12.6M	1.21	28.15	+0.84
7	✓	✓	×	EM	+25.2M	0.87	28.45	+1.14
8	✓	✓	✓	EM	+37.8M	0.66	28.47	+1.16

Table 4.3: Effect of information aggregation on different attention components, i.e., encoder self-attention ("*Enc*"), encoder-decoder attention ("*E-D*"), and decoder self-attention ("*Dec*"). "Para." denotes the number of parameters, and "Speed" denotes the training speed (steps/second). Results are reported on the WMT14 En⇒De translation task using TRANSFORMER-BASE.

training speed and performance, *EM routing* is used as the default multi-head aggregation method in subsequent experiments.

Effect on Attention Components (Rows 4-8) Concerning the individual attention components (Rows 4-6), we found that the encoder and decoder self-attention benefit more from the routing-based information aggregation than the encoder-decoder attention. This is consistent with the finding in [136], which shows that self-attention is a strong semantic feature extractor. Encouragingly, applying EM routing in the encoder (Row 4) significantly improve the translation quality with almost no decrease in decoding speed, which matches the requirement of online MT systems. We find that this is due to the auto-regressive generation schema, modifications on the decoder influence the decoding speed more than the encoder.

Compared with individual attention components, applying routing to multiple components (Rows 7-8) marginally improves translation performance, at the cost of a significant decrease of the training and decoding speeds. Possible reasons include that the added complexity makes the model harder to train, and the benefits enjoyed by different attention components are overlapping to some extent. To balance translation performance and efficiency, we only apply EM routing to aggregate multi-head self-attention at the *encoder* in subsequent experiments.

Encoder Layers As shown in Row 4 of Table 4.3, applying EM routing to all encoder layers significantly decreases the training speed by 37.5%, which is not acceptable since TRANSFORMER is best known for both good performance and quick training. We expect applying to fewer layers can alleviate the training burden. Recent studies show that different layers of NMT encoder can capture different levels of syntax and semantic features [129, 113]. Therefore, an investigation to study whether EM routing works for multi-head attention at different layers is highly desirable.
As shown in Table 4.4, we respectively employ EM routing for multi-head attention at the high-level three layers (Row 3) and low-level three layers (Row 4). The translation quality marginally drops while parameters are fewer and training speeds are quicker. This phenomena verifies that it is unnecessary to apply the proposed model to all layers. We further reduce the applied layers to low-level two (Row 5), the above phenomena still holds. However, a big drop on translation quality occurs when the number of layer is reduced to 1 (Rows 6-7). Accordingly, to balance translation performance and efficiency, we only apply EM routing for multi-head aggregation at the *low-level*

#	**Layers**	**Para.**	**Train**	**BLEU**
1	None	88.0M	1.92	27.31
2	[1-6]	+12.6M	1.20	28.28
3	[4-6]	+6.3M	1.54	28.26
4	[1-3]	+6.3M	1.54	28.27
5	[1,2]	+4.2M	1.67	28.26
6	[6]	+2.1M	1.88	27.68
7	[1]	90.1M	1.88	27.75

Table 4.4: Evaluation of different layers in the encoder, which are implemented as multi-head self-attention with the EM routing based information aggregation. “1” denotes the bottom layer, and “6” the top layer. Results are reported on the WMT14 En⇒De translation task using TRANSFORMER-BASE.

two layers of the encoder, which we term “Aggregation” in the following sections.

4.4.4 Combining Together and Main Results

Finally, we validate the proposed disagreement regularization and advanced information aggregation for multi-head attention on both WMT14 En⇒De and WMT17 Zh⇒En translation tasks. The results are concluded in Table 4.5. Our baseline models, both TRANSFORMER-BASE and TRANSFORMER-BIG, outperform all existing NMT systems on the same data, and match the results of TRANSFORMER reported in previous works, which we believe make the evaluation convincing.

As seen, incorporating disagreement regularization and advanced information aggregation consistently improve translation performance for both base and big TRANSFORMER models across language pairs, demonstrating the efficiency and universality of the proposed approaches. Combining them together further improves translation performances, which confirms our

Architecture	**En⇒De**		**Zh⇒En**	
	# Para.	BLEU	# Para.	BLEU
Existing NMT systems				
Rnn with 8 layers [148]	n/a	26.30	n/a	n/a
Cnn with 15 layers [47]	n/a	26.36	n/a	n/a
Transformer-Base [142]	65M	27.3	n/a	n/a
Transformer-Big [142]	213M	28.4	n/a	n/a
Transformer-Big [54]	n/a	n/a	n/a	24.2
Our NMT systems				
Transformer-Base	88M	27.31	108M	24.13
+ Disagreement	88M	28.20$^{\Uparrow}$	108M	24.85$^{\Uparrow}$
+ Aggregation	92M	28.26$^{\Uparrow}$	112M	24.68$^{\Uparrow}$
+ Both	92M	28.41$^{\Uparrow}$	112M	24.90$^{\Uparrow}$
Transformer-Big	264M	28.58	304M	24.56
+ Disagreement	264M	28.96$^{\uparrow}$	304M	25.08$^{\Uparrow}$
+ Aggregation	297M	28.96$^{\uparrow}$	337M	25.00$^{\uparrow}$
+ Both	297M	29.09$^{\Uparrow}$	337M	25.12$^{\Uparrow}$

Table 4.5: Comparing with existing NMT systems on WMT14 English⇒German and WMT17 Chinese⇒English tasks. "$\uparrow$ / $\Uparrow$": significantly better than the baseline counterpart ($p < 0.05/0.01$), tested by bootstrap resampling.

conjecture that the two approaches are complementary to each other as one improves information extraction and the other benefits information aggregation. It is encouraging to see that Transformer-Base with both approaches even achieves comparable performance to Transformer-Big, with about two thirds fewer parameters, which further demonstrates that our performance gains are not simply brought by additional parameters.

4.5 Evaluation on Sentence Encoding

Although we have shown that our proposed disagreement regularization and advanced information aggregation can improve NMT systems with respect to the translation quality, we still have a poor understanding of what they are capturing and changing from the linguistic perspective. Recently, Conneau et al. [29] designed 10 probing tasks to study what linguistic properties are captured by input representations. We conduct these probing tasks here to study whether our proposed approaches can benefit multi-head attention to produce more informative representations.

4.5.1 Setup

Tasks A probing task is a classification problem that focuses on simple linguistic properties of sentences. "SeLen" is to predict the length of sentences in terms of number of words. "WC" tests whether it is possible to recover information about the original words given its sentence embedding. "TrDep" checks whether an encoder infers the hierarchical structure of sentences. In "ToCo" task, sentences should be classified in terms of the sequence of top constituents immediately below the sentence node. "Bshif" tests whether two consecutive tokens within the sentence have been inverted. "Tense" asks for the tense of the main-clause verb. "SubNm" focuses on the number of the subject of the main clause. "ObjNm" tests for the number of the direct object of the main clause. In "SOMO", some sentences are modified by replacing a random noun or verb with another noun or verb and the classifier should tell whether a sentence has been modified. "CoIn" benchmark contains sentences made of two coordinate clauses. Half of the sentences are inverted the order of the clauses and the task is to tell whether a sentence is intact or modified.

Data and Models The models on each classification task are trained and examined using the open-source dataset provided by Conneau et al. [29], where each task is assigned 100k sentences for training and 10k sentences for validating and testing. Each of our probing model consists of 6 encoding layers followed by a MLP classifier. For each encoding layer, we employ a multi-head self-attention block and a feed-forward block as in Transformer-Base, which have achieved promising results on several NLP tasks [32]. The mean of the top encoding layer is served as the sentence representation passed to the classifier. The difference between the compared models merely lies in the disagreement or aggregation mechanism of multiple attention heads. As we have conduct evaluation study on translation task, here we merely evaluate the representative models in each category. "Disagreement" and "Aggregation" are assigned output disagreement regularization and EM routing algorithms respectively, while "Combine" denotes employing the two mechanisms simultaneously.

4.5.2 Results on Linguistic Probing

Table 4.6 lists the classification accuracies of the three models on the 10 probing tasks. We highlight the best accuracies under each category (i.e., "Surface", "Syntactic", and "Semantic") in bold. Obviously, the proposed models outperform the baseline system on almost all the probing tasks, verifying that more informative representations are produced by enhancing multi-

Task		Baseline	Disagreement	Aggregation	Combine
Surface	SeLen	95.35	96.47	96.02	96.55
	WC	98.03	98.65	98.31	98.87
	Ave.	96.69	*97.56*	97.15	**97.71**
Syntactic	TrDep	44.40	46.54	45.77	46.93
	ToCo	83.48	84.24	84.05	84.17
	BShif	51.45	53.54	50.97	54.26
	Ave.	59.77	*61.44*	60.26	**61.78**
Semantic	Tense	84.57	85.03	85.56	86.07
	SubNm	82.80	83.15	85.47	85.84
	ObjNm	80.31	80.49	82.46	83.38
	SOMO	49.87	49.58	50.09	50.13
	CoIn	69.39	68.48	70.21	69.99
	Ave.	73.38	73.34	*74.76*	**75.08**

Table 4.6: Classification accuracies on 10 probing tasks of evaluating the linguistic properties ("Surface", "Syntactic", and "Semantic") embedded in the encoding representation produced by each model. "Ave." denotes the averaged accuracy in each type of linguistic tasks."Disagreement" denotes the disagreement regularization, "Aggregation" denotes the advanced information aggregation, and "Combine" is the combination of the two mechanisms.

head attention networks with disagreement regularization and advanced information aggregation. Besides, several interesting observations can be made here.

First, disagreement regularization gains better results on surface and syntactic tasks than advanced information aggregation, as indicated with the italic numbers in Table 4.6. Advanced information aggregation, on the contrary, performs better on semantic tasks, especially on "SubNm" and "ObjNm" tasks which are the benchmarks for examining the semantic consistency of the model. This empirical result is consistent with the conclusion in [29]: as a model captures deeper linguistic properties, it will tend to forget about some superficial features.

Second, the combination of the two mechanisms achieves the best accuracies on almost all tasks, which is on par with the results in machine translation task. Concerning the three main categories, the relative improvements over the baseline are respectively 1.05%, 3.36%, and 2.31%. Together with the first observation, we can conclude that the two types of approaches are complementary to each other concerning extracting linguistic information of the input sentence.

4.6 Related Work

The disagreement regularizations are inspired by *agreement learning* in prior works, which encourages alignments or hidden variables of multiple models to be similar. Liang et al. [90] assign agreement terms for jointly training word alignment in phrase-based statistic machine translation. The idea is further extended into other natural language processing tasks such as grammar induction [91]. Levinboim et al. [83] extend the agreement for general bidirectional sequence alignment models with model inevitability regularization. Cheng et la. [22] further explore the agreement on modeling the source-target and target-source alignments in NMT model. In contrast to the mentioned approaches which assign *agreement* terms into loss function, we deploy an alignment *disagreement* regularization by maximizing the distance among multiple attention heads.
The *routing-by-agreement algorithm* origins from the capsule networks [60]. The majority of existing work on capsule networks has focused on computer vision tasks, such as MNIST tasks [123, 61], CIFAR tasks [149], and object segmentation task [82]. The applications of capsule networks in NLP tasks,

however, have not been widely investigated to date. Zhao et al. [158] testify capsule networks on text classification tasks and Gong et al. [49] propose to aggregate a sequence of vectors via dynamic routing for sequence encoding. Inspired by these successes, we apply the routing algorithms to multi-head attention.

4.7 Summary

In this chapter, we propose to better exploit the diversity of multi-head attention by incorporating disagreement regularization and employing advanced aggregation function. To this end, we propose several effective and efficient strategies to implement the disagreement regularization and advanced aggregation function. We find that the output disagreement term and EM routing algorithm yield the best performances, and are complementary to each other. Experimental results on machine translation tasks and linguistic probing tasks demonstrate the effectiveness and universality of the proposed approaches, suggesting that our models produce more informative representation of the input sentence.

□ **End of chapter.**

Chapter 5

Representation Composition for Self-Attention Networks

Built on stacking of self-attention networks, the Transformer architecture has achieved state-of-the-art on many NLP tasks. The strength of Transformer lies in its ability to capture different linguistic properties of the input sentence by different layers and different attention heads. Rather than using the last layer or linearly combining all attention heads, in this chapter, we study how to effectively *compose (i.e., aggregate) the representations* learned by different components. The main points of this chapter are as follows. (1) We propose a bilinear pooling-based approach with low-rank approximation for information aggregation. (2) We further extend the bilinear pooling approach with first-order representations. (3) Experiments on machine translation tasks show that our model consistently improves performances over the baseline. Further analyses demonstrate that our approach indeed captures more syntactic and semantic information.

5.1 Introduction

Deep neural networks (DNNs) have advanced the state of the art in various natural language processing (NLP) tasks, such as machine translation [142], semantic role labeling [131], and language representations [32]. The strength of DNNs lies in their ability to capture different linguistic properties of the input by different layers [129, 118], and composing (i.e., aggregating) these layer representations can further improve performances by providing more comprehensive linguistic information of the input [113, 36].

Recent NLP studies show that single neurons in neural models which are defined as individual dimensions of the representation vectors, carry distinct linguistic information [10]. A follow-up work further reveals that simple properties such as coordinating conjunction (e.g., "but/and") or determiner (e.g., "the") can be attributed to individual neurons, while complex linguistic phenomena such as syntax (e.g., part-of-speech tag) and semantics (e.g., semantic entity type) are distributed across neurons [30]. These observations are consistent with recent findings in neuroscience, which show that task-relevant information can be decoded from a group of neurons interacting with each other [106]. One question naturally arises: *can we better capture complex linguistic phenomena by composing/grouping the linguistic properties embedded in individual neurons?*

The starting point of our approach is an observation in neuroscience: *stronger neuron interactions* – directly exchanging signals between neurons, enable more information processing in the nervous system [75]. We believe that simulating the neuron interactions in nervous system would be an appealing alternative to representation composition, which can potentially better learn the compositionality of natural language with subtle operations at a smaller granularity. Concretely, we propose to employ bilinear pooling [92], which executes pairwise multiplicative interactions among individual representation elements, to achieve *strong* neuron interactions. We also introduce a low-rank approximation to make the original bilinear models computationally feasible [73]. Furthermore, as bilinear pooling only encodes multiplicative second-order features, we propose *extended bilinear pooling* to incorporate first-order representations, which can capture more comprehensive information of the input sentences.

We validate the proposed neuron interaction-based (NI-based) representation composition on top of multi-layer multi-head self-attention networks (MlMhSans). The reason is two-fold. First, MlMhSans are critical components of various SOTA Dnns models, such as Transformer [142], Bert [32], and Lisa [131]. Second, MlMhSans involve in compositions of both multi-layer representations and multi-head representations, which can investigate the universality of NI-based composition. Specifically,

- First, we conduct experiments on the machine translation task, a benchmark to evaluate the performance of neural models. Experimental results on the widely-used WMT14 English⇒German and English⇒French data show that the NI-based composition consistently improves performance over Transformer across language pairs. Compared with existing representation composition strategies [113, 36], our approach shows its superiority in efficacy and efficiency.

- Second, we carry out linguistic analysis [29] on the learned representations from NMT encoder, and find that NI-based composition indeed captures more syntactic and semantic information as expected. These results provide support for our hypothesis that modeling strong neuron interactions helps to better capture complex linguistic information via advanced composition functions, which is essential for downstream NLP tasks.

This paper is an early step in exploring neuron interactions for representation composition in NLP tasks, which we hope will be a long and fruitful journey. We make the following contributions:

- Our study demonstrates the necessity of modeling neuron interactions for representation composition in deep NLP tasks. We employ bilinear pooling to simulate strong neuron interactions.
- We propose *extended bilinear pooling* to incorporate first-order representations, which produces a more comprehensive representation.
- Experimental results show that representation composition benefits the widely-employed MlMhSans by aggregating information learned by multi-layer and/or multi-head attention components.

5.2 Multi-Layer Multi-Head SAN

In the past two years, MlMhSans based models establish the SOTA performances across different NLP tasks. The main strength of MlMhSans lies in the powerful representation learning capacity provided by the multi-layer and multi-head architectures. MlMhSans perform a series of nonlinear transformations from the input sequences to final output sequences. Specifically, MlMhSans are composed of a stack of L identical layers (*multi-layer*), each of which is calculated as

$$\mathbf{H}^l \; = \; \textsc{Self-Att}(\mathbf{H}^{l-1}) + \mathbf{H}^{l-1}, \tag{5.1}$$

where a residual connection is employed around each of two layers [56]. $\textsc{Self-Att}(\cdot)$ is a self-attention model, which captures dependencies among hidden states in $\mathbf{H}^{l-1}$:

$$\textsc{Self-Att}(\mathbf{H}^{l-1}) = \textsc{Att}(\mathbf{Q}^l, \mathbf{K}^{l-1}) \; \mathbf{V}^{l-1}, \tag{5.2}$$

where $\{\mathbf{Q}^l, \mathbf{K}^{l-1}, \mathbf{V}^{l-1}\}$ are the query, key and value vectors that are transformed from the lower layer $\mathbf{H}^{l-1}$, respectively.
Instead of performing a single attention function, Vaswani et al. [142] found it is beneficial to capture different context features with multiple individual attention functions (*multi-head*). Concretely, multi-head attention model first transforms $\{\mathbf{Q}, \mathbf{K}, \mathbf{V}\}$ into H subspaces with different, learnable linear projections:[1]

$$\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h = \mathbf{Q}\mathbf{W}_h^Q, \mathbf{K}\mathbf{W}_h^K, \mathbf{V}\mathbf{W}_h^V, \tag{5.3}$$

where $\{\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h\}$ are respectively the query, key, and value representations of the h-th head. $\{\mathbf{W}_h^Q, \mathbf{W}_h^K, \mathbf{W}_h^V\}$ denote parameter matrices associated with the h-th head. H self-attention functions (Equation 5.2) are applied in parallel to produce the output states $\{\mathbf{O}_1, \dots, \mathbf{O}_H\}$. Finally, the H outputs are concatenated and linearly transformed to produce a final representation:

$$\mathbf{H} = [\mathbf{O}_1, \dots, \mathbf{O}_H] \; \mathbf{W}^O, \tag{5.4}$$

[1]Here we skip the layer index for simplification.

where $\mathbf{W}^O \in \mathbb{R}^{d\times d}$ is a trainable matrix.

Representation Composition Composing (i.e. aggregating) representations learned by different layers or attention heads has been shown beneficial for MlMhSans [36, 2]. Without loss of generality, from here on, we refer to $\{\mathbf{r}_1, \ldots, \mathbf{r}_N\} \in \mathbb{R}^d$ for the representations to compose, where $\mathbf{r}_i$ can be a layer representation ($\mathbf{H}^l$, Equation 5.1) or head representation ($\mathbf{O}_h$, Equation 5.4). The composition is expressed as:

$$\widetilde{\mathbf{H}} = \textsc{Compose}(\mathbf{r}_1, \ldots, \mathbf{r}_N), \tag{5.5}$$

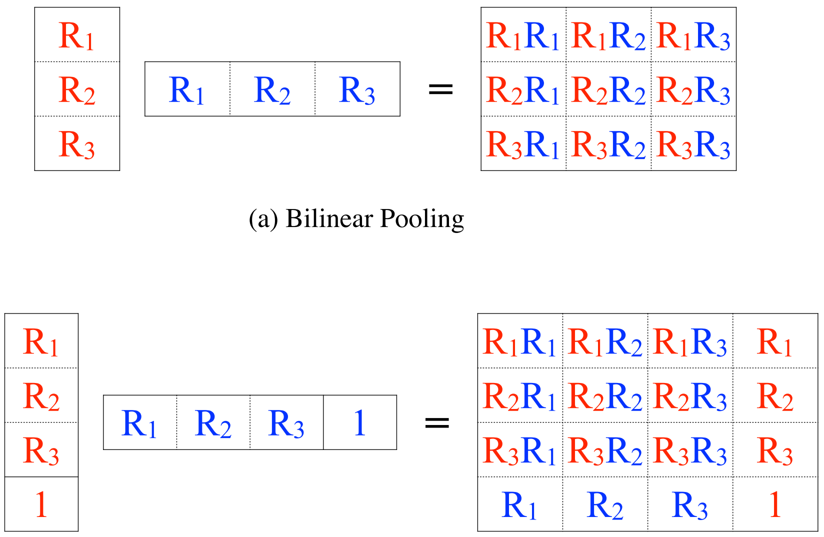
where $\textsc{Compose}(\cdot)$ can be arbitrary functions, such as linear combination[2] [113, 2] and hierarchical aggregation [36]. Though effective to some extent, these approaches do not model neuron interactions among the representation vectors, which we believe is valuable for representation composition in deep NLP models.

5.3 Methodology

Different types of neurons in the nervous system carry distinct signals [27]. Similarly, neurons in deep NLP models – individual dimensions of representation vectors, carry distinct linguistic information [10, 30]. Studies in neuroscience reveal that stronger neuron interactions bring more information processing capability [75], which we believe also applies to deep NLP models.

In this chapter, we explore the strong neuron interactions provided by bilinear pooling for representation composition. Bilinear pooling [92] is a recently proposed feature fusion

[2]The linear composition of multi-head representations (Equation 5.4) can be rewritten in the format of weighted sum: $\mathbf{O} = \sum_{h=1}^{H} \mathbf{O}_h \mathbf{W}_h^O$ with $\mathbf{W}_h^O \in \mathbb{R}^{\frac{d}{H}\times d}$.



(a) Bilinear Pooling

(b) Extended Bilinear Pooling

Figure 5.1: Illustration of (a) *bilinear pooling* that models fully neuron-wise multiplicative interaction, and (b) *extended bilinear pooling* that captures both second- and first-order neuron interactions.

approach in the vision field. Instead of linearly combining all representations, bilinear pooling executes pairwise multiplicative interactions among individual representations, to model *full* neuron interactions as shown in Figure 5.1(a).
Note that there are many possible ways to implement the neuron interactions. The aim of this paper is not to explore this whole space but simply to show that one fairly straightforward implementation works well on a strong benchmark.

5.3.1 Bilinear Pooling

Bilinear pooling [138] is defined as an *outer product* of two representation vectors followed by a linear projection. As illustrated in Figure 5.1(a), all elements of the two vectors have direct multiplicative interactions with each other. However, in the scenario of multi-layer and multi-head composition, we generally have more than two representation vectors to compose (i.e., L layers and H attention heads). To utilize the full second-order (i.e. multiplicative) interactions in bilinear pooling, we concatenate all the representation vectors and feed the concatenated vector twice to the bilinear pooling. Concretely, we have:

$$\mathbf{R} = |\widehat{\mathbf{R}}\widehat{\mathbf{R}}^\top|\mathbf{W}^B, \tag{5.6}$$

$$\widehat{\mathbf{R}} = [\mathbf{r}_1, \ldots, \mathbf{r}_N], \tag{5.7}$$

where $|\widehat{\mathbf{R}}\widehat{\mathbf{R}}^\top| \in \mathbb{R}^{Nd\times Nd}$ is the outer product of the concatenated representation $\widehat{\mathbf{R}}$, $|\cdot|$ denotes serializing the matrix into a vector with dimensionality $(Nd)^2$. In this way, all elements in the partial representations are able to interact with each other in a multiplicative way.

However, the parameter matrix $\mathbf{W}^B \in \mathbb{R}^{(Nd)^2\times d}$ and computing cost cubically increases with dimensionality d, which becomes problematic when training or decoding on a GPU with limited memory. For example, a regular TRANSFORMER model requires a huge amount of 36 billion ($(Nd)^2 \times d$) parameters for $d = 1000$ and $N = 6$. There have been a few attempts to reduce the computational complexity of the original bilinear pooling. Gao et al. [45] propose *compact bilinear pooling* to reduce the quadratic expansion of dimensionality for image classification. Kim et al. [73] and Kong et al. [78] propose *low-rank bilinear pooling* for visual question answering and image classification respectively, which further reduces the parameters to be learned and achieves comparable effectiveness with full bilinear pooling. In this chapter, we focus on the low-rank approximation for its efficiency, and generalize from the original model for deep representations.

5.3.2 Low-Rank Approximation

In the full bilinear models, each output element $R_i \in \mathbb{R}^1$ can be expressed as

$$\begin{aligned} R_i &= \sum_{j=1}^{Nd}\sum_{k=1}^{Nd} w^B_{jk,i}\widehat{R}_j\widehat{R}_k^\top \\ &= \widehat{\mathbf{R}}^\top \mathbf{W}_i^B \widehat{\mathbf{R}}, \end{aligned} \tag{5.8}$$

where $\mathbf{W}_i^B \in \mathbb{R}^{Nd\times Nd}$ is a weight matrix to produce output element R_i. The low-rank approximation enforces the rank of $\mathbf{W}_i^B$ to be low-rank $r \leq Nd$ [114], which is then factorized as $\mathbf{U}_i\mathbf{V}_i^\top$ with $\mathbf{U}_i \in \mathbb{R}^{Nd\times r}$ and $\mathbf{V}_i \in \mathbb{R}^{Nd\times r}$. Accordingly, Equation 5.8 can be rewritten as

$$\begin{aligned} R_i &= \widehat{\mathbf{R}}^\top \mathbf{U}_i\mathbf{V}_i^\top \widehat{\mathbf{R}} \\ &= (\widehat{\mathbf{R}}^\top \mathbf{U}_i \odot \widehat{\mathbf{R}}^\top \mathbf{V}_i)\mathbf{1}_r, \end{aligned} \tag{5.9}$$

where $\mathbf{1}_r$ is a r-dimensional vector of ones, $\odot$ represents element-wise product. By replacing $\mathbf{1}_r$ with $\mathbf{P} \in \mathbb{R}^{r\times d}$, and redefining $\mathbf{U} \in \mathbb{R}^{Nd\times r}$ and $\mathbf{V} \in \mathbb{R}^{Nd\times r}$, the low-rank approximation can be defined as

$$\mathbf{R} = (\widehat{\mathbf{R}}^\top \mathbf{U} \odot \widehat{\mathbf{R}}^\top \mathbf{V})\mathbf{P}. \tag{5.10}$$

In this way, the computation complexity is reduced from $O(d^3)$ to $O(d^2)$. And the parameter matrices $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{P}$ are now feasible to fit in GPU memory.

5.3.3 Extended Bilinear Pooling

Previous work in information theory has proven that second-order and first-order representations encode different types of information [51], which we believe also holds on NLP tasks. As bilinear pooling only encodes second-order (i.e., multiplicative) interactions among individual neurons, we propose the *extended bilinear pooling* to inherit the advantages of first-order representations and form a more comprehensive representation.

Specifically, we append **1**s to the representation vectors. As illustrated in Figure 5.1(b), we respectively append **1** to the two **R** vectors, then the outer product of them produces both second-order and first-order interactions among the elements. According to Equation 5.10, the final representation is revised as:

$$\mathbf{R_f} = (\begin{bmatrix} \widehat{\mathbf{R}} \\ 1 \end{bmatrix}^\top \mathbf{U} \odot \begin{bmatrix} \widehat{\mathbf{R}} \\ 1 \end{bmatrix}^\top \mathbf{V})\, \mathbf{P}, \tag{5.11}$$

where $\widehat{\mathbf{R}}$ is the concatenated representation as in Equation 5.7. As a result, the final representation $\mathbf{R_f}$ preserves both multiplicative bilinear features (as in Equation 5.10) and first-order linear features (as in Equation 5.4).

5.3.4 Applying to Transformer

Transformer [142] consists of an encoder and a decoder, each of which is stacked in 6 layers where we can apply multi-layer composition (excluding the embedding layer) to produce the final representations of the encoder and decoder. Besides, each layer has one (in encoder) or two (in decoder) multi-head attention component with H heads, to which we can apply multi-head composition to substitute Equation 5.4. The two sorts of representation composition can be used individually, while combining them is expected to further improve the performance.

5.4 Evaluation

5.4.1 Setup

Dataset We conduct experiments on the WMT2014 English to German (En⇒De) and English to French (En⇒Fr) translation tasks. The En⇒De dataset consists of about 4.56 million sentence pairs. We use newstest2013 as the development set and newstest2014 as the test set. The En⇒Fr dataset consists of 35.52 million sentence pairs. We use the concatenation of newstest2012 and newstest2013 as the development set and newstest2014 as the test set. We employ BPE [125] with 32K merge operations for both language pairs. We adopt the case-sensitive 4-gram NIST BLEU score [111] as our evaluation metric and bootstrap resampling [77] for significance test.

Models We evaluate the proposed approaches on the advanced Transformer model [142], and implement on top of an open-source toolkit – THUMT. We follow Vaswani et al. [142] to set the configurations and have reproduced their reported results on the En⇒De task. The parameters of the proposed models are initialized by the pre-trained Transformer model. We have tested both *Base* and *Big* models, which differ at hidden size (512 vs. 1024) and number of attention heads (8 vs. 16). Concerning the low-rank parameter (Equation 5.9), we set low-rank dimensionality r to 512 and 1024 in *Base* and *Big* models respectively. All models are trained on eight NVIDIA P40 GPUs where each is allocated with a batch size of 4096 tokens. In consideration of computation cost, we study model variations with *Base* model on the En⇒De task, and evaluate overall performance with *Big* model on both En⇒De and En⇒Fr tasks.

#	Model	# Para.	Train	BLEU
1	TRANSFORMER-BASE	88.0M	2.02	27.31
	Existing representation composition			
2	+ Multi-Layer: Linear Combination	+3.1M	1.98	27.77
3	+ Multi-Layer: Hierarchical Agg.	+23.1M	1.62	28.32
4	+ Multi-Head: Hierarchical Agg.	+13.6M	1.74	28.13
5	+ Both (3+4)	+36.7M	1.42	28.42
	This work: neuron-interaction based representation composition			
6	+ Multi-Layer: *NI-based Composition*	+16.8M	1.93	28.31
7	+ Multi-Head: *NI-based Composition*	+14.1M	1.92	28.29
8	+ Both (6+7)	+30.9M	1.87	**28.54**

Table 5.1: Translation performance on WMT14 English⇒German translation task. "# Para." denotes the number of parameters, and "Train" denotes the training speed (steps/second). We compare our model with linear combination [113] and hierarchical aggregation [36].

5.4.2 Comparison to Existing Approaches

In this section, we evaluate the impacts of different representation composition strategies on the En⇒De translation task with TRANSFORMER-BASE, as listed in Table 5.1.

Existing Representation Composition (Rows 1-5) For the conventional TRANSFORMER model, it adopts multi-head composition with linear combination but only uses top-layer representation as its default setting. Accordingly, we keep the linear multi-head composition (Row 1) unchanged, and choose two representative multi-layer composition strategies (Rows 2 and 3): the widely-used linear combination [113] and the effective hierarchical aggregation [36]. The hierarchical aggregation merges states of different layers through a CNN-like tree structure with the filter size being two, to hierarchically preserve and combine feature channels.

As seen, linearly combining all layers (Row 2) achieves +0.46 BLEU improvement over TRANSFORMER-BASE with almost the same training and decoding speeds. Hierarchical aggregation for multi-layer composition (Row 3) yields larger improvement in terms of BLEU score, but at the cost of considerable speed decrease. To make a fair comparison, we also implement hierarchical aggregation for multi-head composition (Rows 4 and 5), which consistently improves performances at the cost of introducing more parameters and slower speeds.

The Proposed Approach (Rows 6-8) Firstly, we apply our NI-based composition, i.e. *extended bilinear pooling*, for multi-layer composition with the default linear multi-head composition (Row 6). We find that the approach achieves almost the same translation performance as hierarchical aggregation (Row 3), while keeps the training and decoding speeds as *efficient* as linear combination. Then, we apply the NI-based approach for multi-head composition with the default top layer exploitation (Row 7). We can see that our approach gains +0.98 BLEU point over TRANSFORMER-BASE and achieves more improvement than hierarchical aggregation (Row 4). The two results demonstrate that our NI-based approach can be effectively applied to different representation composition scenarios.
At last, we simultaneously apply the NI-based approach to the multi-layer and multi-head composition (Row 8). Our model achieves further improvement over individual models and the hierarchical aggregation (Row 5), showing that TRANSFORMER can benefit from the complementary composition from multiple heads and historical layers. In the following experiments, we adopt NI-based composition for both the multi-layer and multi-

Architecture	**EN⇒DE**		**EN⇒FR**	
	# Para.	BLEU	# Para.	BLEU
Existing NMT systems: [142]				
Transformer-Base	65M	27.3	n/a	38.1
Transformer-Big	213M	28.4	n/a	41.8
Our NMT systems				
Transformer-Base	88M	27.31	95M	39.28
+ NI-Based Composition	118M	$28.54^{\Uparrow}$	125M	$40.15^{\Uparrow}$
Transformer-Big	264M	28.58	278M	41.41
+ NI-Based Composition	387M	$29.17^{\Uparrow}$	401M	$42.10^{\Uparrow}$

Table 5.2: Comparing with existing NMT systems on WMT14 English⇒German ("EN⇒DE") and English⇒French ("EN⇒FR") translation tasks. "$\Uparrow$": significantly better than the baseline ($p < 0.01$) using bootstrap resampling [77].

head compositions as the default strategy.

5.4.3 Main Results on Machine Translation

In this section, we validate the proposed NI-based representation composition on both WMT14 En⇒De and En⇒Fr translation tasks. Experimental results are listed in Table 5.2. The performances of our implemented Transformer match the results on both language pairs reported in previous work [142], which we believe makes the evaluation convincing.

Incorporating NI-based composition consistently and significantly improves translation performance for both base and big Transformer models across language pairs, demonstrating the effectiveness and universality of the proposed NI-based representation composition. It is encouraging to see that Transformer-Base with NI-based composition even achieves competitive performance as that of Transformer-Big in the En⇒De task, with only half fewer parameters and the training speed is twice faster. This further demonstrates that our performance gains are not simply brought by additional parameters. Note that the improvement on En⇒De task is larger than En⇒Fr task, which can be attributed to the size of training data (4M vs. 35M).

5.5 Analysis

In this section, we conduct extensive analysis to deeply understand the proposed models in terms of 1) the linguistic properties learned by the NMT encoder; 2) the influences of first-order representation and low-rank constraint; and 3) the translation performances on sentences of varying lengths.

5.5.1 Linguistic Evaluation on NMT Encoder

Machine translation is a complex task, which consists of both the understanding of input sentence (encoder) and the generation of output conditioned on such understanding (decoder). In this probing experiment, we evaluate the understanding part using Transformer encoders that are trained on the EN⇒DE NMT data, and are fixed in the probing tasks with only MLP classifiers being trained on probing data.

Recently, Conneau et al. [29] designed 10 probing tasks to study what linguistic properties are captured by representations from sentence encoders. A probing task is a classification problem that focuses on simple linguistic properties of input sentences, including surface information, syntactic information, and semantic information. We have given detailed introduction of the 10 probing tasks in Section 4.5 of the previous chapter. However, our setting here is a little different from the setting in Section 4.5

Task		Base	Ours	△
Surface	SeLen	92.20	92.11	-0.1%
	WC	63.00	63.50	+0.8%
	Ave.	77.60	77.81	+0.3%
Syntactic	TrDep	44.74	44.96	+0.5%
	ToCo	79.02	81.31	**+2.9%**
	BShif	71.24	72.44	**+1.7%**
	Ave.	65.00	66.24	**+1.9%**
Semantic	Tense	89.24	89.26	+0.0%
	SubNm	84.69	87.05	**+2.8%**
	ObjNm	84.53	86.91	**+2.8%**
	SOMO	52.13	52.52	+0.7%
	CoIn	62.47	64.93	**+3.9%**
	Ave.	74.61	76.13	**+2.0%**

Table 5.3: Classification accuracies on 10 probing tasks of evaluating the linguistic properties ("Surface", "Syntactic", and "Semantic"). "Ave." denotes the averaged accuracy in each category. "△" denotes the relative improvement, and we highlight the numbers $\geq 1\%$.

as we freeze the parameters in NMT encoders. We conduct probing tasks to examine whether the NI-based representation composition can benefit the Transformer encoder to produce more informative representation.

Table 5.3 lists the results. The NI-based composition outperforms that by the baseline in most probing tasks, proving that our composition strategy indeed helps Transformer encoder generate more informative representation, especially at the syntactic and semantic level. The averaged gains in syntactic and semantic tasks are significant, showing that our strategy makes San capture more high-level linguistic properties. Note that the lower values in surface tasks (e.g., SeLen), are consistent with the conclusion in [29]: as model captures deeper linguistic properties, it will tend to forget about these superficial features.

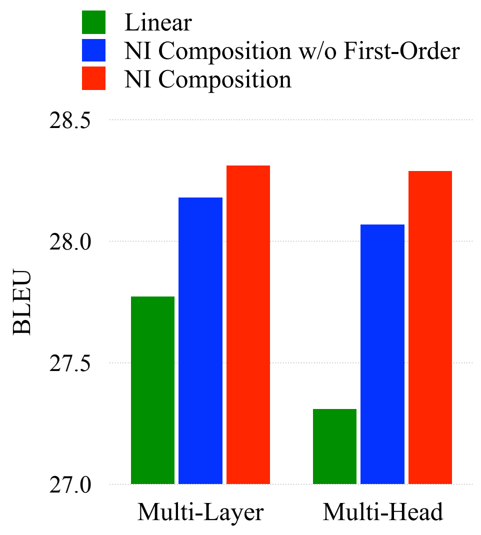


Figure 5.2: Effect of first-order representation on WMT14 En⇒De translation task.

5.5.2 Effect of First-Order Representation

As aforementioned, we extend the conventional bilinear pooling by appending **1**s to the representation vectors thus incorporate first-order representations (i.e. linear combination), and capture both multiplicative bilinear features and additive linear features. Here we conduct ablation study to validate the effectiveness of each component. We respectively experiment on multi-layer and multi-head representation composition, and the results are shown in Figure 5.2.

Several observations can be made. First, we notice that by replacing linear combination with mere bilinear pooling ("NI-based composition w/o first-order" in Figure 5.2), the translation performance significantly improves both in multi-layer and multi-head composition, demonstrating the effectiveness of full neuron interaction and second-order features. We further observe that it is indeed beneficial to extend bilinear pooling

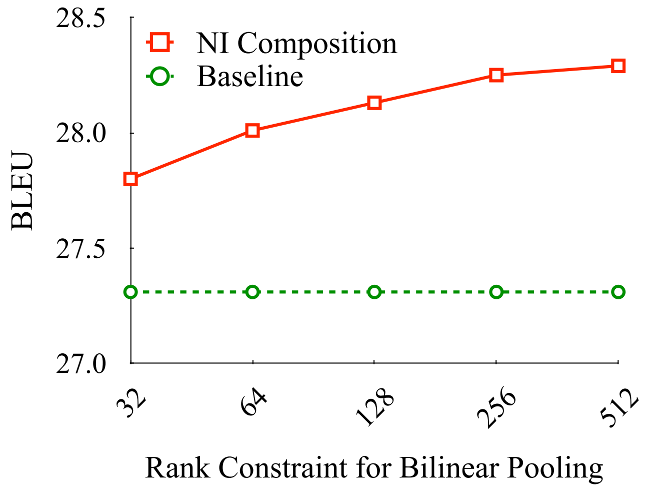


Figure 5.3: BLEU scores on the En⇒De test set with different rank constraints for bilinear pooling. "Baseline" denotes TRANSFORMER-BASE.

with linear combination ("NI composition" in Figure 5.2) which captures the complementary information among them and forms a more comprehensive representation of the input.

5.5.3 Effect of Low-Rank Constraint

In this experiment, we study the impact of low-rank constraint r (Equation 5.9) on bilinear pooling, as shown in Figure 5.3. It is interesting to investigate whether the model with a smaller setting of r can also achieve considerable results. We examine groups of multi-head composition models with different r on the En⇒De translation task. From Figure 5.3, we can see that the translation performance increases with larger r value and the model with $r = 512$ achieves best performance[3]. Note that even when the dimensionality r is reduced to 32, our model can still consistently outperform the baseline with only 0.9M

[3]The maximum value of r is 512 since the rank of a matrix $\mathbf{W} \in \mathbb{R}^{Nd \times Nd}$ is bounded by Nd.

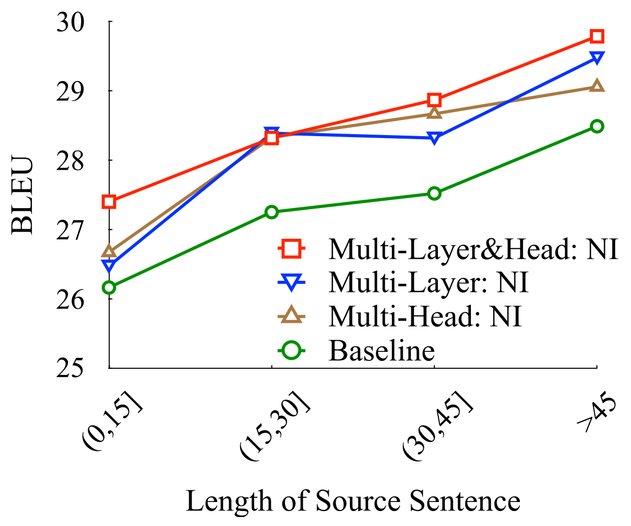


Figure 5.4: BLEU scores on the En⇒De test set with respect to various input sentence lengths. "Baseline" denotes TRANSFORMER-BASE.

parameters added (not shown in the figure). This reconfirms our claim that the improvements on the BLEU score could not be simply attributed to the additional parameters.

5.5.4 Length Analysis

We group sentences of similar lengths together and compute the BLEU score for each group, as shown in Figure 5.4. Generally, the performance of TRANSFORMER goes up with the increase of input sentence lengths, which is different from the results on single-layer RNNSearch models (i.e., performance decreases on longer sentences) as shown in [140]. We attribute this phenomenon to the advanced TRANSFORMER architecture including multiple layers, multi-head attention and feed-forward networks.

Clearly, our NI-based approaches outperform the baseline TRANSFORMER in all length segments, including only using multi-layer composition or multi-head composition, which verifies our contribution that representation composition indeed benefits SANs. Moreover, multi-layer composition and multi-head composition are complementary to each other regarding different length segments, and simultaneously applying them achieves further performance gain.

5.6 Related Work

Bilinear pooling has been well-studied in the computer vision community, which is first introduced by Tenenbaum et al. [138] to separate style and content. Bilinear pooling has since then been considered to replace fully-connected layers in neural networks by introducing second-order statistics, and applied to fine grained recognition [92]. While bilinear models provide richer representations than linear models [51], bilinear pooling produces a high-dimensional feature of quadratic expansion, which may constrain model structures and computational resources. To address this challenge, Gao et al. [45] propose compact bilinear pooling through random projections for image classification, which is further applied to visual question answering [44]. Kim et al. [73] and Kong et al. [78] independently propose low-rank approximation on the transformation matrix of bilinear pooling, which aims to reduce the model size and corresponding computational burden. Their models are applied to visual question answering and fine-grained image classification, respectively.

While most work focus on computer vision tasks, our work is among the few studies [39, 31], which prove the idea of bilinear pooling can have promising applications on NLP tasks.

5.7 Summary

In this chapter, we propose NI-based representation composition for MLMHSANS, by modeling strong neuron interactions in the representation vectors generated by different layers and attention heads. Specifically, we employ bilinear pooling to capture pairwise multiplicative interactions among individual neurons, and propose *extended bilinear pooling* to further incorporate first-order representations. Experiments on machine translation tasks show that our approach effectively and efficiently improves translation performance over the TRANSFORMER model, and multi-head composition and multi-layer composition are complementary to each other. Further analyses reveal that our model makes the encoder of TRANSFORMER capture more syntactic and semantic properties of input sentences.

□ **End of chapter.**

Chapter 6

Pre-trained Attention Models for Code Generation

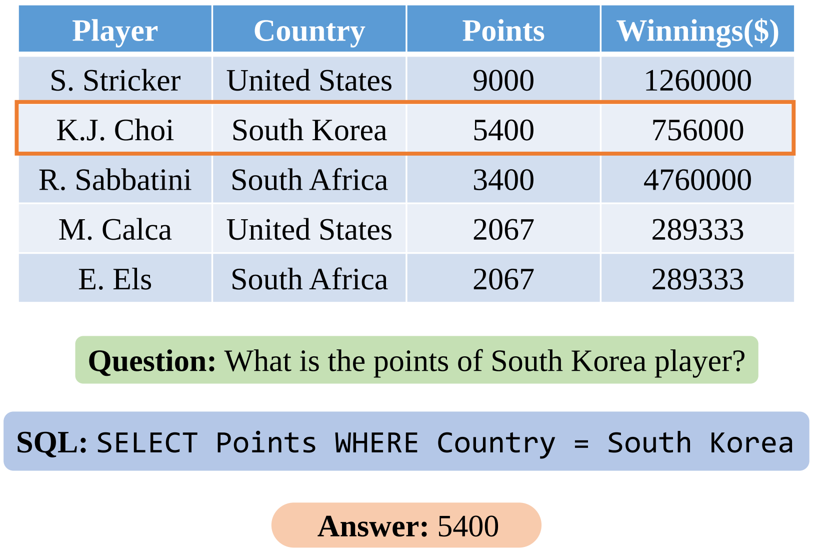
Semantic parsing is the task to map natural language utterances to logical forms or executable code. The state-of-the-art semantic parsers are mostly syntax-specific and heavily-engineered, thus they are not generalizable. In this chapter, we explore how to apply the powerful pre-trained attention models such as BERT [32] to build semantic parsers that are both *effective* and *generalizable*. The main points of this chapter are as follows. (1) We propose a novel BERT-LSTM model that employs a pre-trained BERT encoder followed by an LSTM decoder with a pointer-generator network. (2) We demonstrate the effectiveness and universality of our model on three code generation tasks, where BERT-LSTM achieves state-of-the-art on three of the four datasets. (3) We also highlight several design principles for code generation, such as the use of LSTM decoder and greedy decoding, and fine-tuning BERT parameters.

6.1 Introduction

Digital virtual assistants have become increasingly powerful and popular, giving us the ability to instruct computing devices using natural language. One of the key challenges for such virtual assistants is natural language understanding: how to interpret human languages into machine-understandable representations? Works in semantic parsing try to approach this challenge by mapping natural language to some formal domain-specific programming languages which are executable by computers. An illustrative example is shown in Figure 6.1. These programming languages can be formalisms for querying databases or knowledge bases [170, 13, 171, 167], commands for robots or virtual assistants [5, 19], instructions to manipulate spreadsheets [53], and general-purpose programming languages like Python and Java [94, 162], among others.

The natural language processing community has been developing data-driven approaches for semantic parsing, such as grammar-based probabilistic models [26]. Recently, with the great success of neural networks, deep learning has been applied to semantic parsing. Specifically, researchers treat semantic parsing as a sequence-to-sequence learning problem like neural machine translation. Attentional neural encoder-decoder models are generally employed, where the source side learns to encode the semantics of natural language and the target side learns to generate the corresponding programming code.

Conventionally, bi-directional LSTM or Transformer network is adopted as the encoder. As for the decoder, many task-specific designs have been proposed. Compared to natural language, program source code has well-defined syntax thus allowing us



Player	Country	Points	Winnings($)
S. Stricker	United States	9000	1260000
K.J. Choi	South Korea	5400	756000
R. Sabbatini	South Africa	3400	4760000
M. Calca	United States	2067	289333
E. Els	South Africa	2067	289333

Question: What is the points of South Korea player?

SQL: `SELECT Points WHERE Country = South Korea`

Answer: 5400

Figure 6.1: An example of the WiKiSQL task. The natural language question is parsed into a SQL query, and then executed on corresponding table to get the result.

to design syntax-specific decoders that are tailored to the target programming languages. For example, Yin et al. [162] propose to utilize the Abstract Syntax Tree (AST) in Python programming language and design a tree-structured decoder which reflects the recursive structure of Python AST. Though effective, such specifically-designed semantic parsers lose universality and are hard to generalize to other programming languages.

Meanwhile, the natural language processing community has recently witnessed a rapid advancement in pre-trained models (e.g., BERT [32], GPT [117], XLNet [159]), which have proved to be extremely effective for most language processing tasks. The core idea is to first train those models on large-scale text corpus to learn universal language representations, then fine-tune them on downstream tasks with supervised training. Since the pre-trained models are mostly built on self-attentional Transformer networks, we also call them pre-trained attention models. Given the universality of those pre-trained models, one natural question arises: *can we utilize the powerful pre-trained attention models to build neural semantic parsing models that are both effective and generalizable on many programming languages?*

In this chapter, we propose a model called BERT-LSTM to achieve such a goal, which is designed with three principles in mind: simplicity, extensibility, and effectiveness. We try to add a minimal amount of additional parameters on top of BERT, to make the most use of pre-training. We also design our decoder without considering the underlying syntax of target programming languages to accomplish model generalization. Specifically, our model uses a pre-trained BERT encoder with a pooling layer, followed by an LSTM decoder with a pointer-generator network that can either copy from the input or generate from a vocabulary.

We evaluate our proposed model on three code generation tasks (i.e., Almond, Django, WikiSQL), and demonstrate that BERT-LSTM is capable of generalizing to different domains and different programming languages while registering strong performances, achieving state-of-the-art on three of the four datasets. We also highlight several design principles for better code generation performances: use LSTM decoder rather than Transformer decoder, use greedy decoding rather than beam search, and fine-tune BERT rather than freezing the BERT parameters.

6.2 Methodology

Our neural semantic parsing model is a simple yet novel architecture called BERT-LSTM. It is an encoder-decoder framework composed of the pre-trained BERT encoder and an LSTM decoder with attention in between. In this section, we will elaborate on our model and the rationale for its design. We depict the overall architecture of the model in Figure 6.2.

6.2.1 Encoder

We adopt the BERT model [32] as the sentence encoder. We try to add a minimal amount of additional parameters on top of BERT, in order to make the best use of pre-trained natural language knowledge.

BERT is a deep Transformer network [142] with only self-attention mechanism. To encode a sentence, it first splits it into *word-piece* sub-tokens $x_0 \ldots x_n$, then feeds them to a 12-layer (base model) or 24-layer (big model) Transformer network, to compute the final contextualized representations of each token $h_{\mathrm{E},t}$. BERT is pre-trained on large-scale general English corpus like Wikipedia with the *masked language model* objective. We take the publicly released BERT model and fine-tune it on our code generation tasks.

We also compute the sentence representation $\bar{h}_{\mathrm{E}}$ as a single vector by averaging the token representations from the top BERT layer and feeding it to a single-layer feed-forward network:

$$\bar{h}_{\mathrm{E}} = W_{\mathrm{pool,out}} \mathrm{relu}(W_{\mathrm{pool,in}} \mathrm{avg}(h_{\mathrm{E}}) + b_{\mathrm{pool}})$$

(where W and b are learnable parameters).

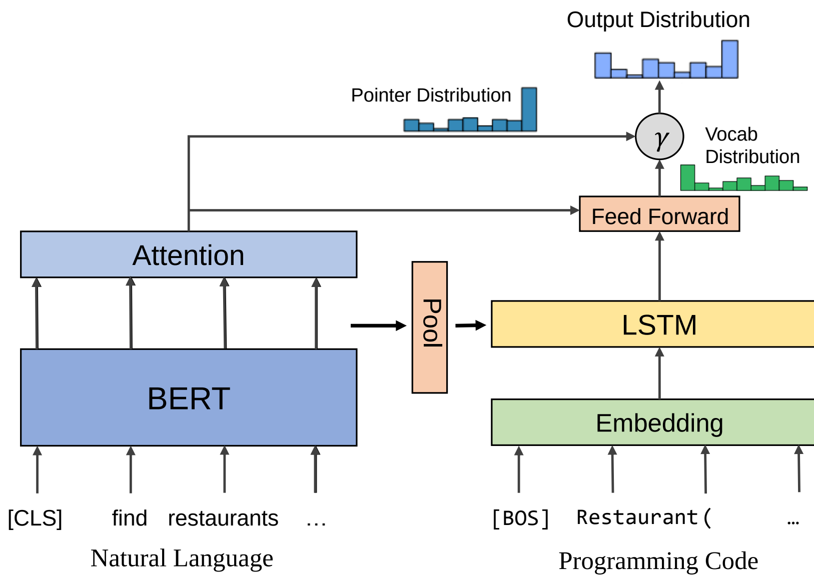


Figure 6.2: The proposed BERT-LSTM model.

To improve the learning process and regularization, normalization and dropout layers are added around the pooling layer.

6.2.2 Decoder

At the time of decoding, our model produces one token of the executable query or program y_t at a time, given the last time produced code token y_{t-1}. During training, we employ teacher forcing and set y_{t-1} as the ground truth of the prior time step. It is worth noting that our decoder does not contain any syntax-specific components, thus our model can generalize to any programming languages, for example, ThingTalk [20], Python, and SQL, which we will show in the experiments part. Specifically, the previous code token is first transformed to a continuous vector through an embedding layer. We subsequently feed the embedded token to an LSTM cell to compute the LSTM hidden state $h_{\mathrm{D},t}$, and then compute the attention scores s_t

(which is a list) against each token in the encoder, as well as the attention value vector v_t and the attention context vector c_t, which are formally expressed as follows:

$$\begin{aligned}
h_{\mathrm{D},0} &= \bar{h}_{\mathrm{E}} \\
c_0 &= \mathbf{0} \\
y_{\mathrm{emb},t} &= W_{\mathrm{emb}} y_{t-1} \\
h_{\mathrm{D},t} &= \mathrm{LSTM}(h_{\mathrm{D},t-1}, [y_{\mathrm{emb},t}; c_{t-1}]) \\
s_t &= \mathrm{softmax}(h_{\mathrm{D},t} h_{\mathrm{E}}^T) \\
v_t &= \sum_{t'} s_{t,t'} h_{\mathrm{E},t'} \\
c_t &= \tanh(W_{\mathrm{att}} \, [v_t; h_{\mathrm{D},t}])
\end{aligned}$$

(“;” indicates concatenation).
With the hidden state $h_{\mathrm{D},t}$ and context vector c_t, the model can predict an output token over a vocabulary that is built on the training data.

6.2.3 Pointer Network

However, program source code is often very diverse and the model may face many entities or variable names that are unseen during training. To cope with this problem, we adopt a *pointer network* [143] and enable the model to either copy from the input sentence or generate from the vocabulary. The use of such pointer network allows the model to be mostly agnostic to specific entity names mentioned in the question, which are copied verbatim in the generated program code.
Specifically, we employ the pointer-generator network proposed by See et al. [124]. As shown in following equations, the choice of whether to copy from the input or to generate from the

vocabulary is decided by a switch probability γ_t:

$$\begin{aligned} \gamma_t &= \sigma(W_\gamma\,[y_{\text{emb},t}; h_{\text{D},t}; c_t]) \\ p_{t,w} &= \gamma_t \sum_{t', x_{t'}=w} s_{t,t'} + (1-\gamma_t)\text{softmax}(W_o c_t) \\ y_t &= \arg\max_w p_{t,w} \end{aligned}$$

where the final distribution p_w predicts the output code token. During training, our model is trained to maximize the likelihood of the program code for a given question using teacher forcing. During inference, the model greedily chooses the token with the highest probability at each time step.

6.2.4 Discussion

Though BERT-LSTM looks very simple, there are still many design choices that need to be discussed. First, we choose LSTM as our decoder. As in the Transformer network for machine translation [142], a Transformer decoder can also be the alternative. Second, we employ the greedy decoding strategy. However, the most popular decoding strategy for text generation is beam search, which may also work for code generation. Lastly, there are two ways to exploit BERT representation: fine-tune it with downstream tasks or freeze it as word embedding while we adopt the former. In the following experiment part, we will evaluate all these design choices on the code generation task.

6.3 Evaluation

In this part, we first introduce our datasets and experimental settings. Then we evaluate the design choices proposed in Section 6.2.4 on the ALMOND dataset. Finally, we validate our BERT-LSTM on all the four datasets. Note that all experimental results are averaged over three runs with different random seeds.

6.3.1 Datasets and Metrics

We conduct experiments on three code generation tasks with four datasets, i.e., Almond-Restaurant, Almond-People, Django, and WiKiSQL. The statistics of the four datasets are shown in Table 6.1, including the train/dev/test set size and the average question/code length.

Almond This dataset [153] is released by the Almond virtual assistant [19] team at Stanford University. Each example in the dataset is composed of a natural language question and the corresponding *ThingTalk* code, which is a specially designed programming language to manipulate the Almond virtual assistant. This dataset is further split into two domains, namely Almond-Restaurant and Almond-People. Almond-Restaurant contains user questions about restaurant cuisines, ratings, locations, etc., while Almond-people is about people's profiles such as work and education. It is worth noting all questions in Almond dataset are compositional questions involving multiple properties of the restaurant or people. So it is quite challenging.

Django This dataset [110] is a collection of lines of Python code from the Django web framework that each line is paired with a natural language description. The dataset is very diverse, exhibiting a wide variety of use cases, such as string manipulation, iteration, and exception handling.

Dataset	Almond-Res	Almond-Ppl	Django	WiKiSQL
Train	364,193	420,253	15,967	56,324
Development	378	315	996	8,419
Test	415	429	1,801	15,873
Avg. Ques Len	13	15	14	24
Avg. Code Len	23	25	8	6

Table 6.1: Statistics of the datasets. Almond-Res and Almond-Ppl denote the restaurant and people datasets, respectively. Avg. Ques Len refers to the average length of questions.

WiKiSQL This dataset [171] contains examples of natural language questions and annotated SQL queries extracted from 24,241 tables on Wikipedia. Different from prior datasets, each example here is paired with a table on which the SQL query is executed to get the answer. An example is shown in Figure 6.1. However, only table schema (i.e., table column names) is allowed to use but not the table content. Therefore, we concatenate all the table column names and the question as a long sentence for input to the model.

Metrics As is standard in semantic parsing, we use **accuracy** as our evaluation metric which is the percentage of correctly generated program code that exactly matches the ground truth.

6.3.2 Setup

Preprocessing To work with the BERT model, we employ the WordPiece tokenizer [148] on all natural language questions with a 30,000 token vocabulary. Note that it is also necessary to apply the tokenizer on program code since our model tries to copy input tokens as the output. We adopt the preprocessed Django dataset provide by Yin et al. [162] where quoted strings in the input are substituted with place holders. The preprocessed WIKISQL dataset is from Wang et al. [144] with annotations.

Configuration We adopt the uncased BERT-BASE model as our encoder which has 12 layers and 768 hidden sizes. We also set the LSTM hidden size as 768 and feed the averaged encoder state as the initial LSTM state. We set the trainable decoder embedding size as 50. We adopt Adam optimizer [74] and the standard Transformer learning rate strategy [142]. We set the training batch tokens for ALMOND, DJANGO, and WIKISQL as 9000, 3000, and 500, respectively. We train all the models for 60000 iterations and choose the best-performing model on the development set.

6.3.3 Evaluation on Design Choices

As discussed in Section 6.2.4, there are several design choices need to be evaluated, i.e., LSTM decoder vs. Transformer decoder, greedy decoding vs. beam search, and fine-tune BERT vs. freeze BERT. We set LSTM decoder + greedy decoding + fine-tune BERT as the standard system. We conduct comparative experiments on the ALMOND-RESTAURANT dataset and show the results in Figure 6.3.

As the results indicate, our standard system achieves the highest accuracy among the other strategies. We observe that when changing the LSTM decoder to Transformer, the model converges much slower. It is important for the training to converge quickly as too many updates to the encoder would reduce the effectiveness of pre-training. We also conjecture the reason why greedy decoding performs better than beam search

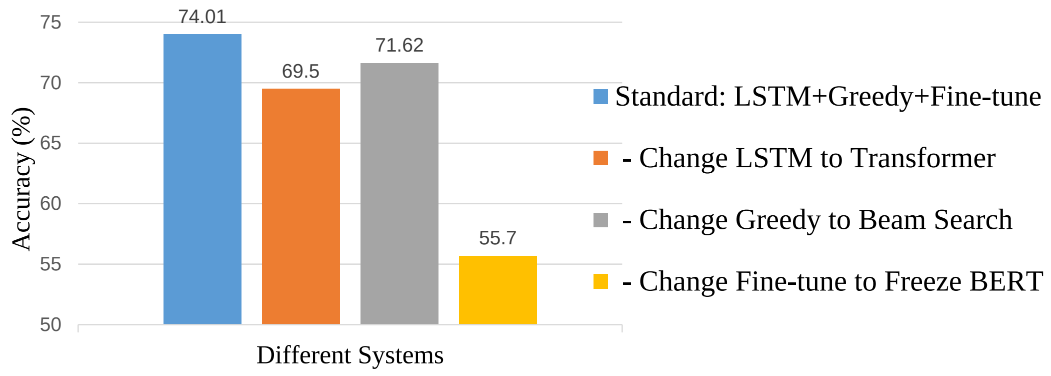


Figure 6.3: The evaluation results on the design choices.

is due to the fact that program code is less diverse than natural language. Therefore, in the following experiments, we all adopt the standard system.

In addition, we conduct another experiment which changes the fine-tuning BERT to randomly initialized BERT. The decreased accuracy (66.31%) demonstrates that what benefits in BERT is the pre-trained natural language knowledge (i.e., the initial parameters) rather than the large number of trainable parameters.

6.3.4 Results on All Datasets

In this section, we compare our BERT-LSTM model against several previously published systems as well as some baselines on the four datasets. To demonstrate the effectiveness of copying mechanism (i.e., the pointer network in Section 6.2.3), we also report the model results without copying by setting γ_t to 0.

Almond Table 6.2 presents results on the two ALMOND datasets. MQAN denotes Multi-Task Question Answering Network [102], which is the previous state-of-the-art semantic parsing model

Model	ALMOND-RESTAURANT	ALMOND-PEOPLE
MQAN [102]	68.92%	75.65%
BERT-LSTM	**74.01**%	**81.93**%
– copying	56.76%	59.78%

Table 6.2: Code generation accuracies on the two ALMOND datasets.

employed by the Almond virtual assistant [20]. MQAN is an encoder-decoder network consisting of a stack of self-attention, co-attention, and LSTM layers with Glove and character word embeddings. It also uses the pointer-generator network for copying from the inputs. As Table 6.2 shows, our BERT-LSTM registers 5.09% and 6.28% absolute improvements over MQAN in accuracy on ALMOND-RESTAURANT and ALMOND-PEOPLE, respectively. Given that MQAN is a more complex network, BERT-LSTM yields better performances by utilizing the pre-trained natural language knowledge. Without the copying mechanism, our model drops 17% and 22% accuracy on the two datasets. The results also indicate that BERT-LSTM is able to generalize to different domains.

Django Table 6.3 reports results on DJANGO where BERT-LSTM also achieves state-of-the-art result. Previous methods in the literature can be divided into two categories: sequence-to-sequence models and sequence-to-tree models. Except for the Neural Machine Translation [162] and Latent Predictor Network [94], all existing methods in Table 6.3 belong to sequence-to-tree models which implicity or explicitly utilize the Abstract Syntax Tree of Python language. Our BERT-LSTM, though very effective, is a sequence-to-sequence model thus generalizable to other programming languages.

Model	**Accuracy**
Sequence-to-Tree Network [34]	39.4%
Neural Machine Translation [162]	45.1%
Latent Predictor Network [94]	62.3%
Syntax Neural Model [162]	71.6%
Transition-Based Syntax Parser [163]	73.7%
Coarse-to-Fine Decoding [35]	74.1%
BERT-LSTM	**76.48**%
– copying	54.07%

Table 6.3: Python code generation accuracies on DJANGO. BERT-LSTM achieves state-of-the-art result.

Model	**Accuracy**
Sequence-to-Sequence [171]	23.4%
Sequence-to-SQL [171]†	48.3%
SQLNet [154]†	61.3%
Transition-based Syntax Parser [163]†	68.6%
Coarse-to-Fine Decoding [35]†	71.7%
Multi-Task QA Network [102]	75.4%
SQLova [65]$^{\dagger *}$	83.6%
X-SQL [57]$^{\dagger *}$	86.0%
HydraNet [100]$^{\dagger *}$	**86.5**%
BERT-LSTM*	78.49%
– copying	35.99%

Table 6.4: SQL code generation accuracies on WIKISQL. "†" denotes syntax-specific models. "*" indicates that the model employs pre-trained BERT.

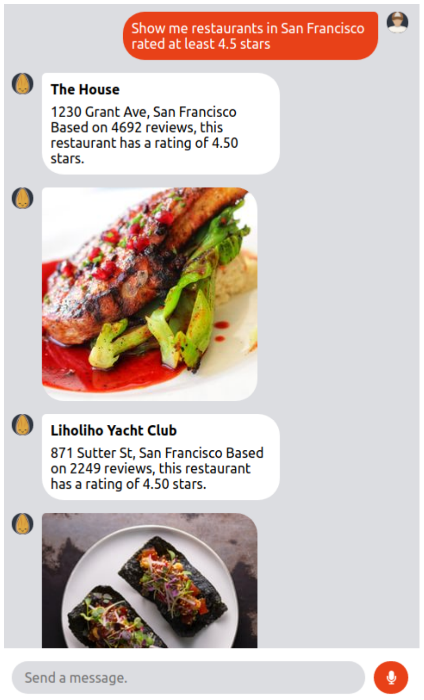
WiKiSQL The experimental results on WiKiSQL are shown in Table 6.4. The SQL queries in this dataset are highly structured. Specifically, WiKiSQL queries have the format "`SELECT agg_op agg_col WHERE (cond_col cond_op cond) AND ...`" that is a subset of the SQL syntax. `SELECT` indicates which column to be included in the calculation when applying the aggregation operation `agg_op` on column `agg_col`. There maybe zero or multiple `WHERE` clauses, which specifies the constraints on column `cond_col` by the operation `cond_op` and the condition value `cond`. In this way, most of the existing methods are syntax-specific and yield decent accuracies, as in Table 6.4. Besides, the state-of-the-art methods also employ the pre-trained BERT as ours which are shown in the second block.
Our BERT-LSTM registers strong performances on WiKiSQL compared to existing methods. Without considering the underlying syntax of SQL, BERT-LSTM even outperforms many syntax-specific models that explicitly reflect the SQL query structure. Moreover, BERT-LSTM is a neat and simple model compared to the state-of-the-art ones (in the second block) which are all heavily-engineered. Therefore, BERT-LSTM is much easier to be adopted by other researchers.
Note that when removing the copying mechanism, the code generation accuracy significantly decreases to 35.99%. This result indicates that, on WiKiSQL, the pointer network dominates the decoding process and most of the output tokens are copied from the input questions.

6.3.5 Case Study

We present several code generation examples in Table 6.5. From top to down, we respectively show examples from Almond,

Input	*which restaurants have italian food?*
Pred.	`now => (@org.schema.Restaurant.Restaurant) filter param:servesCuisine:String =~ "italian" => notify` ✓
Input	*name all the people who have won scholarships.*
Pred.	`now => (@org.schema.Person.Person) filter param:award:Array(String) contains~ " => notify` ✗
Ref.	`now => (@org.schema.Person.Person) filter param:award:Array(String) contains~ "scholarships" => notify`
Input	*activate is a lambda function which returns None for any argument x.*
Pred.	`activate = lambda x :  None` ✓
Input	*self.plural is a lambda function with an argument n, which returns result of boolean expression n not equal to integer 1.*
Pred.	`self.plural = lambda n :  len(n)` ✗
Ref.	`self.plural = lambda n :  int(n!=1)`
Input	*Table has columns: Ballarat_FL Wins Byes Losses Draws Against, how many Byes have Against of 1076 and Wins smaller than 13?*
Pred.	`SELECT count(Byes) FROM table WHERE Against = 1076 AND Wins < 13` ✓
Input	*Table has columns: Conference Division Team City Home_Arena, which team is in the southeast with a home at Philips?*
Pred.	`SELECT (Team) FROM table WHERE Conference = southeast AND Home_Arena = Philips` ✗
Ref.	`SELECT (Team) FROM table WHERE Division = southeast AND Home_Arena = Philips`

Table 6.5: Code generation examples. Each dataset has one correct example and one incorrect example. Copied contents with copying probability > 0.9 are highlighted.


Show me restaurants in San Francisco rated at least 4.5 stars

The House
1230 Grant Ave, San Francisco Based on 4692 reviews, this restaurant has a rating of 4.50 stars.

Liholiho Yacht Club
871 Sutter St, San Francisco Based on 2249 reviews, this restaurant has a rating of 4.50 stars.

Figure 6.4: Screenshot of the Almond virtual assistant.

Django, and WikiSQL, each with one correct instance and one incorrect instance. We also highlight the predicted contents with copying probability larger than 0.9. From the table, we observe that BERT-LSTM is able to learn the correct syntax and structure of the three programming languages. Most of the failed cases are due to partial implementation errors (e.g., the Almond example) or incorrect understanding of the input semantics (e.g., the Django and WikiSQL examples). We also find that the prediction on WikiSQL heavily relies on the pointer network to copy column names and values from input descriptions, which is consistent with our results in Table 6.4. BERT-LSTM has been integrated into the Almond virtual assistant [19] for parsing complex user questions to code which is subsequently executed. One use case is shown in Figure 6.4.

6.4 Related Work

Semantic Parsing has been a long-term research topic in the NLP field, which aims to map natural language utterances into their structured meaning representations (e.g., executable queries or logical forms). Previous systems typically learn lexicalized mapping rules and are guided by grammatical formalisms [169, 46, 147, 81]. Recently, neural sequence-to-sequence systems have been applied on semantic parsing with promising results [34, 68, 105]. Several approaches have been proposed to enhance the performance of these models, for example, data augmentation guided by the grammar [68, 76], transfer learning [42], utilizing user feedbacks [66], and guiding generation with information retrieval [55]. Besides, there are efforts developing structured decoders that exploit the underlying syntax of meaning representations. Dong et al. [34] propose a model that generates tree-structure in a top-down manner. Xiao et al. [150] utilize the grammar to constrain the decoding process. Yin et al. [162, 163] design grammar models for generating the abstract syntax trees. Dong et al. [35] propose a two-stage decoding method which first generates a rough sketch and then fills the missing details.

Researchers have extended semantic parsing to generate domain-specific programming languages [115] with widespread interests on the WikiSQL dataset [171]. Since the SQL queries are uniformly composed of a `SELECT` clause and a `WHERE` clause, SQLNet [154] independently generates the two components with a sequence-to-set model which eliminates the order issue in `WHERE` conditions. TypeSQL [166] also adopts the sequence-to-set structure but with an additional "type" information.

6.5 Summary

In this chapter, we try to utilize the powerful pre-trained attention models to build neural semantic parsing models that are both *effective* and *generalizable*. We propose BERT-LSTM which uses a pre-trained BERT encoder followed by an LSTM decoder with a pointer-generator network. Experimental results on three code generation tasks demonstrate that BERT-LSTM can generalize to different domains and different programming languages. We also show the importance of copying mechanism in the code generation task through ablation studies. The findings concluded in this chapter can facilitate future semantic parsing and code generation studies.

□ **End of chapter.**

Chapter 7

Conclusion and Future Work

7.1 Conclusion

As we enter the big data era, there are large amounts of sequential data generated everyday. We rely on sequence learning to discover valuable knowledge from sequential data for various applications, for example, weather forecast. Attention mechanisms, as an effective method for dependency modeling, has been widely employed for deep learning-based sequence models. However, we believe that the expressiveness of attention mechanisms is not fully exploited, due to either the application domains or model design deficiencies. In this thesis, we propose customized solutions to improve attention mechanisms for the source code domain. We also improve the deep self-attention by designing better coordination mechanisms for the multiple attention heads and multiple layers.

In particular, in Chapter 3, we explore the conventional shallow attention (i.e., RNN-based attention) applied to source code completion. We treat the code completion task as a language modeling problem and propose a tailored attention mechanism that can exploit the structure information on program's abstract syntax tree. To deal with the out-of-vocabulary (OoV) words in a program, we further propose a pointer mixture network that learns to copy OoV words from local context based on the attention weights.

In Chapter 4, we study how to improve multi-head attention, the core component in deep self-attention. To this end, we propose three disagreement regularizations and two routing-by-agreement algorithms to better exploit the diversity among multiple attention heads. Experiments on both machine translation and sentence encoding tasks show that our models consistently outperform the baselines.

In Chapter 5, we continue to explore the deep self-attention. To effectively aggregate the representations learned by different attention heads or layers, we propose a low-rank bilinear pooling-based approach with first-order extension. We also extensively analyze our model to show that it indeed captures more linguistic information of the input sentences.

In Chapter 6, we study how to apply pre-trained attention models such as BERT to the downstream semantic parsing task, which generates executable code directly from natural language utterances. We propose a BERT-LSTM model that employs a pre-trained BERT encoder and a general-purpose LSTM decoder. We also adopt pointer network for copying code tokens. The model achieves state-of-the-are on three of the four code generation datasets.

In summary, this thesis targets at designing effective and customized solutions for improving attention mechanisms in sequence learning. Extensive experiments on various datasets across different applications demonstrate the effectiveness of our proposed methods.

7.2 Future Work

Sequence learning with attention mechanisms has been extensively studied in recent years, and it is a promising research topic. Although we have proposed a number of novel techniques that advance the state-of-the-art solutions, there are still many interesting research directions which can be considered as future work.

7.2.1 Multi-Modal Attention Models

In this thesis, we have mainly applied attention mechanisms and sequence learning to process textual data (including source code). Besides NLP, sequence learning is also employed in many cross-modal applications where most of them involve visual and textual data, for example, image captioning. Image captioning refers to the process of generating textual description based on the contents of an image which we call image-to-sequence. Attention mechanisms have been adopted in such a problem, e.g., hard visual attention [152], to focus on certain parts of the image when generating the words. However, hard attention is non-differentiable and requires more complicated techniques such as variance reduction or reinforcement learning. Therefore, there is still a lot of work to do to make visual attention smoother and easier to train.

Recently, there is a surgent interest in cross-modal pre-trained attention models [89, 133]. The models are pre-trained on some huge corpus of multi-modal data such as images or videos with captions, and then fine-tuned on downstream cross-modal tasks like visual question answering. To adapt the pre-trained models from unimodal to multi-modal, many research questions need to be investigated, for example, how to design the masked language modeling objective with visual features, how to effectively match and aggregate visual and textual features, etc.

7.2.2 Model Compression on Large Attention Models

Nowadays, the deep self-attention models like Transformer [142] and BERT [32] have achieved remarkable success in various NLP tasks. However, the increasingly deep and large attention models usually come with hundreds of millions of parameters, making them difficult to be deployed in real-life applications that need quick responses and resource-limited devices like smartphones. Model compression, as a potential approach to reducing the model size and enhancing computational efficiency, has drawn growing attention.

There are several directions worth trying. First, model pruning, which removes less important components (e.g., weights, layers, and attention heads) of neural networks. Michel et al. [103] prove that many attention heads in the Transformer architecture can be pruned during inference time without significantly impacting performances. We can potentially extend their method to the BERT model for pruning unnecessary attention heads. We can also draw lessons from computer vision which guide the model to learn to skip some of the layers [145]. Second, knowledge distillation [59], in which a small student model is trained to mimic the behaviors of a large teacher model through some optimization objectives. Distillation techniques have been well-studied in the computer vision field and we can extend them to the large pre-trained attention models.

7.2.3 Interpretability and Reliability of Attention Models

Though attention models reach impressive performances in various sequence learning tasks, like most deep learning models, the black-box property and non-linear architecture make them non-transparent for human decision making. Recently, model interpretation has become a hotspot in deep learning research which tries to explain the behaviors of deep learning models. In terms of self-attention mechanism, Yang et al. [156] design a word order detection task and find that self-attention can effectively capture word orders in a sentence despite no recurrent connections. Given the popularity of pre-trained attention models, their interpretability can be a promising direction. However, unlike CNNs for images, interpreting pre-trained models is challenging due to the complex transformer architecture and language properties. More research efforts are required to better understand the linguistic and word knowledge captured by pre-trained models.

Besides, pre-trained attention models like BERT have been proved vulnerable to adversarial attacks [70]. With the wide adoption of pre-trained models in production systems, their reliability issue becomes critical. In particular, we are interested in designing effective defense mechanisms against adversarial attacks thus improving the robustness of pre-trained models.

□ **End of chapter.**

Chapter 8

List of Publications

1. Silei Xu, Giovanni Campagna, **Jian Li**, Monica S. Lam. *Schema2QA: Answering Complex Queries on the Structured Web with a Neural Model.* ArXiv Technical Report, arXiv:2001.05609, 2020.

2. **Jian Li**, Xing Wang, Baosong Yang, Shuming Shi, Michael R. Lyu, Zhaopeng Tu. *Neuron Interaction Based Representation Composition for Neural Machine Translation.* The 34th AAAI Conference on Artificial Intelligence (AAAI), 2020.

3. **Jian Li**, Baosong Yang, Zi-Yi Dou, Xing Wang, Michael R. Lyu, Zhaopeng Tu. *Information Aggregation for Multi-Head Attention with Routing-by-Agreement.* The 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), pages 3566-3575, 2019.

4. Baosong Yang, **Jian Li**, Derek Wong, Lidia Chao, Xing Wang, Zhaopeng Tu. *Context-Aware Self-Attention Networks.* The 33rd AAAI Conference on Artificial Intelligence (AAAI), pages 381-394, 2019.

5. **Jian Li**, Zhaopeng Tu, Baosong Yang, Michael R. Lyu and Tong Zhang. *Multi-Head Attention with Disagreement Regularization.* The 2018 International Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 2897-2903, 2018.

6. **Jian Li**, Yue Wang, Michael R. Lyu, Irwin King. *Code Completion with Neural Attention and Pointer Networks.* The 28th International Joint Conference on Artificial Intelligence (IJCAI), pages 4159-4165, 2018.

7. **Jian Li**, Pinjia He, Jieming Zhu, Michael R. Lyu. *Software Defect Prediction via Convolutional Neural Network.* The 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), pages 318-328, 2017.

8. Pinjia He, Jieming Zhu, Shilin He, **Jian Li**, Michael R. Lyu. *Towards Automated Log Parsing for Large-Scale Log Data Analysis.* IEEE Transactions on Dependable and Secure Computing (TDSC), Vol. 15, Issue 6, pages 931-944, 2018.

9. Pinjia He, Jieming Zhu, Shilin He, **Jian Li**, Michael R. Lyu. *An Evaluation Study on Log Parsing and Its Use in Log Mining.* The 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 654-661, 2016.

□ **End of chapter.**

Bibliography

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE)*, pages 3–14, 1995.

[2] K. Ahmed, N. S. Keskar, and R. Socher. Weighted Transformer Network for Machine Translation. In *arXiv preprint arXiv:1711.02132*, 2018.

[3] M. Allamanis, H. Peng, and C. Sutton. A convolutional attention network for extreme summarization of source code. In *Proceedings of the 2016 International Conference on Machine Learning (ICML)*, pages 2091–2100, 2016.

[4] M. Allamanis and C. Sutton. Mining idioms from source code. In *Proceedings of the 2014 International Symposium on Foundations of Software Engineering (FSE*, pages 472–483, 2014.

[5] Y. Artzi and L. Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics (TACL)*, 1:49–62, 2013.

[6] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[7] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[8] P. Baldi, S. Brunak, and F. Bach. *Bioinformatics: the machine learning approach.* MIT press, 2001.

[9] A. Bapna, M. X. Chen, O. Firat, Y. Cao, and Y. Wu. Training deeper neural machine translation models with transparent attention. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLLP)*, pages 3028–3033, 2018.

[10] A. Bau, Y. Belinkov, H. Sajjad, N. Durrani, F. Dalvi, and J. Glass. Identifying and controlling important neurons in neural machine translation. In *Proceedings of the 2019 International Conference on Learning Representations (ICLR)*, 2019.

[11] H. Ben-Younes, R. Cadene, M. Cord, and N. Thome. Mutan: Multimodal tucker fusion for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision (CVPR)*, pages 2612–2620, 2017.

[12] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[13] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in*

Natural Language Processing (EMNLP), pages 1533–1544, 2013.

[14] A. Bhoopchand, T. Rocktäschel, E. Barr, and S. Riedel. Learning python code suggestion with a sparse pointer network. *arXiv preprint arXiv:1611.08307*, 2016.

[15] P. Bielik, V. Raychev, and M. Vechev. Phog: probabilistic model for code. In *Proceedings of the 2016 International Conference on Machine Learning (ICML)*, pages 2933–2942, 2016.

[16] C. M. Bishop. *Pattern recognition and machine learning.* springer, 2006.

[17] J. Boyan, D. Freitag, and T. Joachims. A machine learning architecture for optimizing web search engines. In *Proceedings of the 1996 AAAI Workshop on Internet Based Information Systems*, pages 1–8, 1996.

[18] P. F. Brown, V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

[19] G. Campagna, R. Ramesh, S. Xu, M. Fischer, and M. S. Lam. Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*, pages 341–350, 2017.

[20] G. Campagna, S. Xu, M. Moradshahi, R. Socher, and M. S. Lam. Genie: A generator of natural language semantic parsers for virtual assistant commands. In *Proceedings*

of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 394–410, 2019.

[21] J. Cheng, L. Dong, and M. Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[22] Y. Cheng, S. Shen, Z. He, W. He, H. Wu, M. Sun, and Y. Liu. Agreement-based joint training for bidirectional attention-based neural machine translation. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2761–2767, 2016.

[23] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of the Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST)*, pages 103–111, 2014.

[24] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[25] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 577–585, 2015.

[26] S. Clark and J. R. Curran. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics*, 33(4):493–552, 2007.

[27] J. Y. Cohen, S. Haesler, L. Vong, B. B. Lowell, and N. Uchida. Neuron-type-specific signals for reward and punishment in the ventral tegmental area. *Nature*, 482(7383):85, 2012.

[28] colah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[29] A. Conneau, G. Kruszewski, G. Lample, L. Barrault, and M. Baroni. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 2126–2136, 2018.

[30] F. Dalvi, N. Durrani, H. Sajjad, Y. Belinkov, A. Bau, and J. Glass. What is one grain of sand in the desert? analyzing individual neurons in deep nlp models. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[31] J.-B. Delbrouck and S. Dupont. Multimodal compact bilinear pooling for multimodal neural machine translation. *arXiv preprint arXiv:1703.08084*, 2017.

[32] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Annual Conference of the North American Chapter of the Associa-*

tion for Computational Linguistics (NAACL), pages 4171–4186, 2019.

[33] T. Domhan. How much attention do you need? a granular analysis of neural machine translation architectures. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1799–1808, 2018.

[34] L. Dong and M. Lapata. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pages 33–43, 2016.

[35] L. Dong and M. Lapata. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pages 731–742, 2018.

[36] Z.-Y. Dou, Z. Tu, X. Wang, S. Shi, and T. Zhang. Exploiting deep representations for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4253–4262, 2018.

[37] Z.-Y. Dou, Z. Tu, X. Wang, L. Wang, S. Shi, and T. Zhang. Dynamic layer aggregation for neural machine translation with routing-by-agreement. In *Proceedings of the 2019 AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 86–93, 2019.

[38] K. Doya. Bifurcations of recurrent neural networks in gradient descent learning. *IEEE Transactions on Neural Networks*, 1(75):218, 1993.

[39] T. Dozat and C. D. Manning. Deep biaffine attention for neural dependency parsing. In *Proceedings of the 2017 International Conference on Learning Representations (ICLR)*, 2017.

[40] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids.* Cambridge university press, 1998.

[41] R. J. Elliott, L. Aggoun, and J. B. Moore. *Hidden Markov Models: Estimation and Control*, volume 29. Springer Science & Business Media, 1995.

[42] X. Fan, E. Monti, L. Mathias, and M. Dreyer. Transfer learning for neural semantic parsing. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 48–56, 2017.

[43] S. Feng, S. Liu, N. Yang, M. Li, M. Zhou, and K. Zhu. Improving attention modeling with implicit distortion and fertility for machine translation. In *Proceedings of the 26th International Conference on Computational Linguistics (COLING)*, pages 3082–3092, 2016.

[44] A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 457–468, 2016.

[45] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell. Compact bilinear pooling. In *Proceedings of the IEEE Conference on*

Computer Vision and Pattern Recognition (CVPR), pages 317–326, 2016.

[46] R. Ge and R. Mooney. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL)*, pages 9–16, 2005.

[47] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1243–1252, 2017.

[48] F. A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN)*, volume 3, pages 189–194, 2000.

[49] J. Gong, X. Qiu, S. Wang, and X.-J. Huang. Information aggregation via dynamic routing for sequence encoding. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING)*, pages 2742–2752, 2018.

[50] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning.* MIT press, 2016.

[51] M. W. Goudreau, C. L. Giles, S. T. Chakradhar, and D. Chen. First-order versus second-order single-layer recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(3):511–513, 1994.

[52] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE*

Transactions on Neural Networks and Learning Systems, 28(10):2222–2232, 2016.

[53] S. Gulwani and M. Marron. Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 803–814, 2014.

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

[55] S. A. Hayati, R. Olivier, P. Avvaru, P. Yin, A. Tomasic, and G. Neubig. Retrieval-based neural code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 925–930, 2018.

[56] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[57] P. He, Y. Mao, K. Chakrabarti, and W. Chen. X-sql: reinforce schema representation with context. *arXiv preprint arXiv:1908.08113*, 2019.

[58] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu. On the naturalness of software. In *Proceedings of the 2012 International Conference on Software Engineering (ICSE)*, pages 837–847, 2012.

[59] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[60] G. E. Hinton, A. Krizhevsky, and S. D. Wang. Transforming auto-encoders. In *Proceedings of the 2011 International Conference on Artificial Neural Networks (ICANN)*, 2011.

[61] G. E. Hinton, S. Sabour, and N. Frosst. Matrix Capsules with EM Routing. In *Proceedings of the 2018 International Conference on Learning Representations (ICLR)*, 2018.

[62] S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

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

[64] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017.

[65] W. Hwang, J. Yim, S. Park, and M. Seo. A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*, 2019.

[66] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pages 963–973, 2017.

[67] F. Jelinek. *Statistical methods for speech recognition*. MIT press, 1997.

[68] R. Jia and P. Liang. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pages 12–22, 2016.

[69] W. Jiang, L. Ma, Y.-G. Jiang, W. Liu, and T. Zhang. Recurrent fusion network for image captioning. In *Proceedings of the 2018 European Conference on Computer Vision (ECCV)*, pages 499–515, 2018.

[70] D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits. Is bert really robust? natural language attack on text classification and entailment. *arXiv preprint arXiv:1907.11932*, 2019.

[71] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 2015 International Conference on Machine Learning (ICML)*, pages 2342–2350, 2015.

[72] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

[73] J.-H. Kim, K.-W. On, W. Lim, J. Kim, J.-W. Ha, and B.-T. Zhang. Hadamard product for low-rank bilinear pooling. In *Proceedings of the 2017 International Conference on Learning Representations (ICLR)*, 2017.

[74] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[75] C. Koch, T. Poggio, and V. Torre. Nonlinear interactions in a dendritic tree: localization, timing, and role in information processing. *Proceedings of the National Academy of Sciences*, 80(9):2799–2802, 1983.

[76] T. Kočiskỳ, G. Melis, E. Grefenstette, C. Dyer, W. Ling, P. Blunsom, and K. M. Hermann. Semantic parsing with semi-supervised sequential autoencoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1078–1087, 2016.

[77] P. Koehn. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 388–395, 2004.

[78] S. Kong and C. Fowlkes. Low-rank bilinear pooling for fine-grained classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 365–374, 2017.

[79] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

[80] A. Krogh, M. Brown, I. S. Mian, K. Sjolander, and D. Haussler. Hidden markov models in computational biology. applications to protein modeling. *Journal of Molecular Biology*, 235(5):1501–1531, 1994.

[81] T. Kwiatkowski, E. Choi, Y. Artzi, and L. Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1545–1556, 2013.

[82] R. LaLonde and U. Bagci. Capsules for object segmentation. *arXiv preprint arXiv:1804.04241*, 2018.

[83] T. Levinboim, A. Vaswani, and D. Chiang. Model invertibility regularization: Sequence alignment with or without parallel data. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 609–618, 2015.

[84] J. Li, P. He, J. Zhu, and M. R. Lyu. Software defect prediction via convolutional neural network. In *Proceedings of the 2017 International Conference on Software Quality, Reliability and Security (QRS*, pages 318–328, 2017.

[85] J. Li, Z. Tu, B. Yang, M. R. Lyu, and T. Zhang. Multi-head attention with disagreement regularization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2897–2903, 2018.

[86] J. Li, X. Wang, B. Yang, S. Shi, M. R. Lyu, and Z. Tu. Neuron interaction based representation composition for neural machine translation. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[87] J. Li, Y. Wang, M. R. Lyu, and I. King. Code completion with neural attention and pointer networks. In *Proceedings*

of the 27th International Joint Conference on Artificial Intelligence (IJCAI), pages 4159–4165, 2018.

[88] J. Li, B. Yang, Z.-Y. Dou, X. Wang, M. R. Lyu, and Z. Tu. Information aggregation for multi-head attention with routing-by-agreement. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), Volume 1 (Long and Short Papers)*, pages 3566–3575, 2019.

[89] L. H. Li, M. Yatskar, D. Yin, C.-J. Hsieh, and K.-W. Chang. Visualbert: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*, 2019.

[90] P. Liang, B. Taskar, and D. Klein. Alignment by agreement. In *Proceedings of the 2006 Conference of the North American Chapter of the Association of Computational Linguistics (NAACL)*, pages 104–111, 2006.

[91] P. S. Liang, D. Klein, and M. I. Jordan. Agreement-based learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 913–920, 2008.

[92] T.-Y. Lin, A. RoyChowdhury, and S. Maji. Bilinear cnn models for fine-grained visual recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV*, pages 1449–1457, 2015.

[93] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio. A structured self-attentive sentence embedding. In *Proceedings of the 2017 International Conference on Learning Representations (ICLR)*, 2017.

[94] W. Ling, P. Blunsom, E. Grefenstette, K. M. Hermann, T. Kočiskỳ, F. Wang, and A. Senior. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pages 599–609, 2016.

[95] W. Ling, I. Trancoso, C. Dyer, and A. W. Black. Character-based neural machine translation. *arXiv preprint arXiv:1511.04586*, 2015.

[96] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[97] C. Liu, X. Wang, R. Shin, J. E. Gonzalez, and D. Song. Neural code completion. 2016.

[98] J. Lu, J. Yang, D. Batra, and D. Parikh. Hierarchical question-image co-attention for visual question answering. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 289–297, 2016.

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

[100] Q. Lyu, K. Chakrabarti, S. Hathi, S. Kundu, J. Zhang, and Z. Chen. Hybrid ranking network for text-to-sql. *Technical Report*, 2020.

[101] C. D. Manning, C. D. Manning, and H. Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[102] B. McCann, N. S. Keskar, C. Xiong, and R. Socher. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*, 2018.

[103] P. Michel, O. Levy, and G. Neubig. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 14014–14024, 2019.

[104] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3111–3119, 2013.

[105] D. Misra and Y. Artzi. Neural shift-reduce ccg semantic parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1775–1786, 2016.

[106] A. S. Morcos and C. D. Harvey. History-dependent variability in population dynamics during evidence accumulation in cortex. *Nature neuroscience*, 19(12):1672, 2016.

[107] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1287–1293, 2016.

[108] R. Nag, K. Wong, and F. Fallside. Script recognition using hidden markov models. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 11, pages 2071–2074, 1986.

[109] R. Nallapati, B. Zhou, C. dos Santos, C. Gulcehre, and B. Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, pages 280–290, 2016.

[110] Y. Oda, H. Fudaba, G. Neubig, H. Hata, S. Sakti, T. Toda, and S. Nakamura. Learning to generate pseudo-code from source code using statistical machine translation (t). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 574–584. IEEE, 2015.

[111] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL)*, pages 311–318, 2002.

[112] B. Pardo and W. Birmingham. Modeling form for on-line following of musical performances. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 1018, 2005.

[113] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for*

Computational Linguistics (NAACL), pages 2227–2237, 2018.

[114] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Bilinear classifiers for visual recognition. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2009.

[115] C. Quirk, R. Mooney, and M. Galley. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP), Volume 1: Long Papers*, pages 878–888, 2015.

[116] L. R. Rabiner and B.-H. Juang. *Fundamentals of speech recognition.* Tsinghua University Press, 1999.

[117] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[118] A. Raganato and J. Tiedemann. An Analysis of Encoder Representations in Transformer-Based Machine Translation. In *EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 2018.

[119] A. Rajaraman and J. D. Ullman. *Mining of massive datasets.* Cambridge University Press, 2011.

[120] V. Raychev, P. Bielik, and M. Vechev. Probabilistic model for code with decision trees. In *Proceedings of the 2016 International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 731–747, 2016.

[121] V. Raychev, M. Vechev, and E. Yahav. Code completion with statistical language models. In *ACM SIGPLAN Notices*, volume 49, pages 419–428, 2014.

[122] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[123] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3856–3866, 2017.

[124] A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pages 1073–1083, 2017.

[125] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1715–1725, 2016.

[126] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[127] T. Shen, T. Zhou, G. Long, J. Jiang, S. Pan, and C. Zhang. Disan: Directional self-attention network for rnn/cnn-free language understanding. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[128] Y. Shen, X. Tan, D. He, T. Qin, and T.-Y. Liu. Dense information flow for neural machine translation. In *Pro-*

ceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), pages 1294–1303, 2018.

[129] X. Shi, I. Padhi, and K. Knight. Does string-based neural mt learn source syntax? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1526–1534, 2016.

[130] I. R. Sipos, A. Ceffer, and J. Levendovszky. Parallel optimization of sparse portfolios with ar-hmms. *Computational Economics*, 49(4):563–578, 2017.

[131] E. Strubell, P. Verga, D. Andor, D. Weiss, and A. McCallum. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5027–5038, 2018.

[132] K. Sugiyama, K. Hatano, M. Yoshikawa, and S. Uemura. Refinement of tf-idf schemes for web pages using their hyperlinked neighboring pages. In *Proceedings of the 14th ACM Conference on Hypertext and hypermedia (Hypertext)*, pages 198–207, 2003.

[133] C. Sun, A. Myers, C. Vondrick, K. Murphy, and C. Schmid. Videobert: A joint model for video and language representation learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7464–7473, 2019.

[134] Y. Sun, S. Wang, Y. Li, S. Feng, X. Chen, H. Zhang, X. Tian, D. Zhu, H. Tian, and H. Wu. Ernie: Enhanced representation through knowledge integration. *arXiv preprint arXiv:1904.09223*, 2019.

[135] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3104–3112, 2014.

[136] G. Tang, M. Müller, A. R. Gonzales, and R. Sennrich. Why self-attention? a targeted evaluation of neural machine translation architectures. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4263–4272, 2018.

[137] C. Tao, S. Gao, M. Shang, W. Wu, D. Zhao, and R. Yan. Get the point of my utterance! learning towards effective responses with multi-head attention mechanism. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4418–4424, 2018.

[138] J. B. Tenenbaum and W. T. Freeman. Separating style and content with bilinear models. *Neural Computation*, 12(6):1247–1283, June 2000.

[139] N. K. Tran and C. Niedereée. Multihop attention networks for question answer matching. In *Proceedings of the The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR)*, pages 325–334, 2018.

[140] Z. Tu, Z. Lu, Y. Liu, X. Liu, and H. Li. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 76–85, 2016.

[141] Z. Tu, Z. Su, and P. Devanbu. On the localness of software. In *Proceedings of the 22nd ACM SIGSOFT International*

Symposium on Foundations of Software Engineering (FSE, pages 269–280, 2014.

[142] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

[143] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2692–2700, 2015.

[144] C. Wang, K. Tatwawadi, M. Brockschmidt, P.-S. Huang, Y. Mao, O. Polozov, and R. Singh. Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100*, 2018.

[145] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[146] M. White, C. Vendome, M. Linares-Vásquez, and D. Poshyvanyk. Toward deep learning software repositories. In *Proceedings of the 2015 Working Conference on Mining Software Repositories (MSR)*, pages 334–345, 2015.

[147] Y. W. Wong and R. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 960–967, 2007.

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

[149] E. Xi, S. Bing, and Y. Jin. Capsule network performance on complex data. *arXiv preprint arXiv:1712.03480*, 2017.

[150] C. Xiao, M. Dymetman, and C. Gardent. Sequence-based structured prediction for semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pages 1341–1350, 2016.

[151] H. Xu and K. Saenko. Ask, attend and answer: Exploring question-guided spatial attention for visual question answering. In *Proceedings of The 2016 European Conference on Computer Vision (ECCV)*, pages 451–466, 2016.

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

[153] S. Xu, G. Campagna, J. Li, and M. S. Lam. Schema2qa: Answering complex queries on the structured web with a neural model. *arXiv preprint arXiv:2001.05609*, 2020.

[154] X. Xu, C. Liu, and D. Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.

[155] B. Yang, Z. Tu, D. F. Wong, F. Meng, L. S. Chao, and T. Zhang. Modeling localness for self-attention networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4449–4458, 2018.

[156] B. Yang, L. Wang, D. F. Wong, L. S. Chao, and Z. Tu. Assessing the ability of self-attention networks to learn word order. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3635–3644, 2019.

[157] B. Yang, L. Wang, D. F. Wong, L. S. Chao, and Z. Tu. Convolutional self-attention networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 4040–4045, 2019.

[158] M. Yang, W. Zhao, J. Ye, Z. Lei, Z. Zhao, and S. Zhang. Investigating capsule networks with dynamic routing for text classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3110–3119, 2018.

[159] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5754–5764, 2019.

[160] Z. Yang, Z. Hu, Y. Deng, C. Dyer, and A. Smola. Neural machine translation with recurrent attention modeling. In *Proceedings of the 15th Conference of the European*

Chapter of the Association for Computational Linguistics (EACL), pages 383–387, 2017.

[161] K. Yao, T. Cohn, K. Vylomova, and K. Duh. Depth-gated recurrent neural networks. *arXiv preprint arXiv:1508.03790*, 9, 2015.

[162] P. Yin and G. Neubig. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pages 440–450, 2017.

[163] P. Yin and G. Neubig. Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations*, pages 7–12, 2018.

[164] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo. Image captioning with semantic attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4651–4659, 2016.

[165] F. Yu, D. Wang, E. Shelhamer, and T. Darrell. Deep layer aggregation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2403–2412, 2018.

[166] T. Yu, Z. Li, Z. Zhang, R. Zhang, and D. Radev. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational*

Linguistics (NAACL), Volume 2 (Short Papers), pages 588–594, 2018.

[167] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3911–3921, 2018.

[168] F. E. Zarwi, A. Vij, and J. Walker. Modeling and forecasting the evolution of preferences over time: A hidden markov model of travel behavior. *arXiv preprint arXiv:1707.09133*, 2017.

[169] L. Zettlemoyer and M. Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687, 2007.

[170] L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 2005 Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[171] V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.