Neural Networks for Text Matching

Chunlin Xu

Faculty of Computing, Engineering and Built Environment

Ulster University

This thesis is submitted for the degree of

Doctor of Philosophy

April 2021

Declaration

Chunlin Xu

April 2021

Acknowledgements

First and foremost, I would like to express deepest appreciation to my supervisors. To Professor Hui Wang, for providing careful guidance on my research. To Dr. Shengli Wu, for encouraging me a lot when I was struggling. To Dr. Zhiwei Lin, for providing me valuable suggestions on both scientific research and daily life.

I would also like to thank Ulster University for giving me scholarship for living and education. I am grateful for my colleagues and friends. Thanks to Huan Wan, Xin Wei, Mengyuan Wang, Zhi Chen, Lingkai Yang, Fayas and Jojo, for great help during my stay at university. To ShuangShuang Kong and Mingzhou Yang, for great company.

Lastly, thanks to my parents, for their unconditional support. To Xing Tian, for cheering me up when I felt depressed.

Abstract

Text matching is a task of comparing two texts to identify their relationship. It is an important component of a variety of natural language processing (NLP) tasks. Recently, neural networks provide a new paradigm for text matching. In this thesis, we seek to improve text matching in the following two perspectives: (1) text representation and (2) text interaction.

Tree-Structured Long Short-Term Memory (TreeLSTM) has shown its effectiveness in text representation. To further improve the expressive power of TreeLSTM, we propose two new text representation models: TreeLSTM with Tag-aware Hypernetwork (TagHyperTreeLSTM) and Tag-Enhanced Dynamic Compositional Neural Network (TE_DCNN), respectively. These two models are both devised to alleviate the inability of distinguishing different syntactic compositions of standard TreeLSTM with the aid of Part-of-Speech (POS) tags. TagHyperTreeLSTM contains two separate TreeLSTMs, a tag-aware hypernetwork TreeLSTM to generate parameters of the sentence encoder TreeLSTM dynamically, and a sentence encoder TreeLSTM to generate the final sentence representation. TE_DCNN shares similar framework with TagHyperTreeLSTM, but it extends the standard TreeLSTM with binarized constituency tree to a novel structure, named ARTreeLSTM with general constituency tree in which non-leaf nodes can have any number of child nodes.

In addition, the ability of capturing matching features between two texts is of great significance for improving text matching performance. Two new interaction-based matching models are proposed in this thesis: Multi-Level Compare-Aggregate model (MLCA) and

Multi-Level Matching Network (MMN). MLCA matches each word in one text against the other text at three different levels of granularity – word level (word-by-word matching), phrase level (word-by-phrase matching) and sentence level (word-by-sentence matching). MMN utilises multiple levels of word representations such as word embedding, contextualized word representation, to obtain multiple word level matching results for final text level matching decision.

In summary, this thesis proposes four novel neural network based models for text matching. As text matching is a fundamental operation in various NLP tasks and applications including paraphrase identification, natural language inference, machine comprehension, information retrieval and so on, we believe that models presented in this thesis would promote the performance of the above NLP tasks and applications containing text matching. Moreover, text representation models described in this thesis would also be helpful for some other NLP applications such as text classification.

Table of contents

List of figures

List of tables

List of Abbreviations

AI	Artificial Intelligence
BiLSTM	Bidirectional Long Short-Term Memory
BRNN	Bidirectional Recurrent Neural Network
CNN	Convolutional Neural Network
GNN	Graph Neural Network
LSTM	Long Short-Term Memory
MLP	Multi Layer Perceptron
NLP	Natural Language Processing
POS	Part-of-Speech
RecNN	Recursive Neural Network
RNN	Recurrent Neural Network
TreeLSTM	Tree-Structured Long Short-Term Memory

Chapter 1

Introduction

1.1 Overview of Text Matching

Natural language is filled with ambiguity, e.g., a certain meaning can be associated with several different word expressions and a single word can have multiple meanings, which makes it incredibly difficult for a machine to accurately determine the intended meaning of text or voice data. The ability for a computer to read and comprehend human language, i.e., Natural Language Processing (NLP), is a significant research area in Artificial Intelligence (AI). A wide spectrum of applications need high level of natural language understanding, including information retrieval [35, 36], recommender systems [161, 153], machine translation [26, 113], question answering [137, 144], etc.

Text matching aiming to identify the relationship between two texts is a fundamental operation in a variety of NLP tasks and applications including paraphrase identification, natural language inference, machine comprehension, information retrieval and answer selection. Specifically, in paraphrase identification [145], text matching is used to recognize whether two texts have the same meaning. In natural language inference [7], text matching is utilised to identify the logical relationship between a hypothesis sentence and a premise sentence. In answer selection [133], text matching is used to judge whether a candidate answer sentence is related to a question. In machine comprehension [94], text matching is used to extract the best answer span for a given question from a matched paragraph. In information retrieval [81], text matching is employed to find the most relevant text to a given query. Table 1.1 gives examples of three different text matching tasks.

Table 1.1 *Examples of three different text matching tasks. S is the source sentence. For paraphrase identification, "+" denotes a paraphrase of S, "-" otherwise. For natural language inference, E, C and N mean entailment, contradiction and neutral, respectively. For answer selection, A_+ is an answer of question Q, A_- otherwise.*

Paraphrase Identification:
S: How do I learn English step by step?
+: What is step by step guide to learn English?
-: How to learn Chinese step by step?
Natural Language Inference:
S: A dog jumping over a beam.
N: An animal is outdoors.
C: An animal is lying down.
E: An animal is jumping.
Answer Selection:
Q: What is the meaning of the plus-minus sign?
A_+: It indicates a value that can be of either sign.
A_-: The sign is normally pronounced plus or minus.

1.2 Deep Neural Networks for Text Matching

Text matching is a challenging problem due to the following two factors. Firstly, natural language is filled with ambiguity, making it incredibly difficult for a computer to accurately recognize its intended meaning. For example, different expressions may represent the same meaning, e.g., words “grapefruit”, “pomelo” and “shaddock” refer to the same fruit. Similarly, the same word may have different meanings, e.g., word “apple” may refer to a fruit or a brand in different contexts. Secondly, the order of words in a text is of great importance for text matching. For example, phrase pairs “in all” and “all in” are exactly the same for a matching model which takes no order information into consideration. However, they are totally different in meaning.

Traditional text matching models mainly involves two steps. The first step is to extract discriminative features. The second step is to determine the relationship between two texts based on features from the first step through learning [24, 115, 95, 42, 62, 11]. The performances of traditional text matching models heavily rely on the features extracted. However, these features are devised manually for a specific task, thus the generalization ability of traditional text matching models is limited.

Deep neural networks have made astonishing progress in the field of artificial intelligence in the past decade, providing a powerful new tool for natural language modeling. A deep neural network involves a collection of neurons which are organized in layers. The layered structure makes deep neural network models have the ability of taking raw data as input and learn features in higher levels gradually. Recently, the research on text matching has shifted from traditional text matching models to deep neural network based models. Deep neural network based text matching models have two advantages. Firstly, deep neural network based text matching models can extract features from raw data directly, which avoids the process of designing features manually. Moreover, since the feature extraction process is a part of the model, a deep neural network based text matching model can be easily applied to various text matching tasks. Secondly, combining with word embedding technique [79, 78], deep neural network based text matching models can eliminate word level ambiguity effectively.

Existing deep neural network based text matching models can be generally partitioned into two types, i.e., representation-based models and interaction-based models, respectively. The representation-based models encode two texts into two representation vectors independently by using deep neural networks such as recurrent neural networks (RNNs) [20], convolutional neural networks (CNNs) [92], recursive neural networks (RecNNs) [59] and self-attention based neural networks [104]. Then a matching function is used to aggregate the above two representation vectors into a fixed-size vector. Finally, a classifier network is used to obtain the final matching decision. Recursive neural networks have achieved impressive results for text representation on several downstream NLP tasks including text matching, due to its ability of obtaining the structural information of a sentence explicitly. To further enhance the expressive power of existing RecNNs, this thesis proposes two text representations models (Chapter 3 and Chapter 4) which can distinguish different compositions. By contrast, the interaction-based models aim to capture direct matching features and two texts are interacted before obtaining the final text representations. This includes matching-aggregation models [87, 15, 55], attentive representation models [74, 119], dependent reading models [100, 32] and other models over interaction matrix [86, 45, 34]. In this thesis, we focus on matching-aggregation framework where smaller units such as words in two texts are matched firstly. We propose two new models for text matching under this framework (Chapter 5 and Chapter 6).

1.3 Thesis Motivation and Contributions

This thesis aims to improve text matching in the following two perspectives: (1) text representation: to enhance the expressive power of tree-structured neural networks by dynamic composition, and (2) text interaction: to extract more matching information by performing matching at multiple levels of granularity and using multiple levels of word representations.

Recursive neural networks (RecNNs) or tree-structured neural networks learn sentence representation by exploiting syntactic structures. Based on the pre-obtained syntactic parse tree, a RecNN model converts each word at a leaf node of the tree to a representation vector, and then uses a composition function to compose word/phrase pairs to get representations of the intermediate nodes in the tree. Finally, the representation of the root node is viewed as a representation of the sentence. However, a major limitation is that all kinds of compositions share the same parameters in a RecNN model, neglecting the fact that different syntactic compositions exist which require different parameters for the RecNN model to handle precisely.

To further enhance the expressive power of RecNN models, we propose two new dynamic composition models for text representation: TreeLSTM with Tag-aware Hypernetwork (TagHyperTreeLSTM) and Tag-Enhanced Dynamic Compositional Neural Network (TE_DCNN). These two models are both devised to alleviate the inability of distinguishing different syntactic compositions of standard TreeLSTM with the aid of Part-of-Speech tags. TagHyperTreeLSTM consists of two separate TreeLSTMs, a tag-aware hypernetwork TreeLSTM to generate parameters of the sentence encoder TreeLSTM dynamically, and a sentence encoder TreeLSTM to generate the final sentence representation. The TE_DCNN model shares similar framework with the TagHyperTreeLSTM, but it extends the standard TreeLSTM with binarized constituency tree to ARTreeLSTM with general constituency tree in which each non-leaf node can have any number of child nodes.

In addition to performing text matching based on semantic representation vectors of texts, another way is to let smaller units (e.g., words) in two texts interact or match firstly. Then the matching results of these small units are aggregated for final matching decision. This is called the matching-aggregation or compare-aggregate framework. Although existing matching-aggregation based models have made impressive progress in text matching, there is still room for further improvement.

One limitation of previous works is that they mainly focus on matching at single granularity, i.e., word level, without considering matching at other levels of granularity such as word-by-phrase, word-by-sentence matching and so on. In this thesis, we propose Multi-Level Compare-Aggregate model (MLCA) which conducts matching at different granularities. MLCA matches each word in one text against the other text at three different levels of granularity, i.e., word level (word-by-word matching), phrase level (word-by-phrase matching) and sentence level (word-by-sentence matching).

Another limitation of previous models is that only the final representations of words are used to obtain the word level matching results for text level matching decision without considering other levels of word representations. To alleviate this limitation, we propose Multi-Level Matching Network (MMN), which utilises multiple levels of word representations such as word embedding and contextualized word representation, to obtain multiple word level matching results for final text level matching decision.

1.4 Organization

The rest of this thesis is structured as follows: related works are presented in Chapter 2. Two proposed representation-based models TagHyperTreeLSTM and TE_DCNN are presented in Chapter 3 and Chapter 4, respectively. The other two interaction-based models MLCA and MMN are described in Chapter 5 and Chapter 6, respectively. Finally, Chapter 7 concludes this thesis and future work.

1.5 Publications

The work of this thesis is supported by the following four research papers:

1. Chunlin Xu, Zhiwei Lin, Hui Wang, and Shengli Wu. "Multi-Level Compare-Aggregate Model for Text Matching." In 2019 International Joint Conference on Neural Networks (IJCNN), pp. 1-7. IEEE, 2019. (Chapter 5)

2. Chunlin Xu, Zhiwei Lin, Shengli Wu, and Hui Wang. "Multi-Level Matching Networks for Text Matching." In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 949-952. 2019. (Chapter 6)

3. Chunlin Xu, Hui Wang, Shengli Wu, Zhiwei Lin. "TreeLSTM with tag-aware hypernetwork for sentence representation." Neurocomputing, Volume 434, pp. 11-20, 2021. (Chapter 3)

4. Tag-Enhanced Dynamic Compositional Neural Network over Arbitrary Tree Structure for Sentence Representation (accepted by Journal - Expert System with Applications, Elsevier B.V.) (Chapter 4)

Chapter 2

Literature Review

In this chapter, we firstly introduce three widely used deep neural networks, i.e., recurrent neural networks (RNNs), convolutional neural networks (CNNs) and recursive neural networks (RecNNs). Then, a detailed literature review on deep neural network based text matching models is presented.

2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [30] allow the output from previous timestep to be used as the input to the current timestep. The idea behind RNNs is to make use of sequential information and is suitable for modeling sequence data such as audio, video, time-series, and text. RNNs conduct the same work at each timestep and the output of current timestep is dependent on its previous timestep.

2.1.1 Simple RNN

Figure 2.1 introduces a simple RNN [30] and its unfolding in time. Unfolding means to spread out the network for the entire sequence. Taking a sequence of text with 10 words as an example, the RNN will be unfolded into a network with 10 layers and each layer corresponds to a single word.

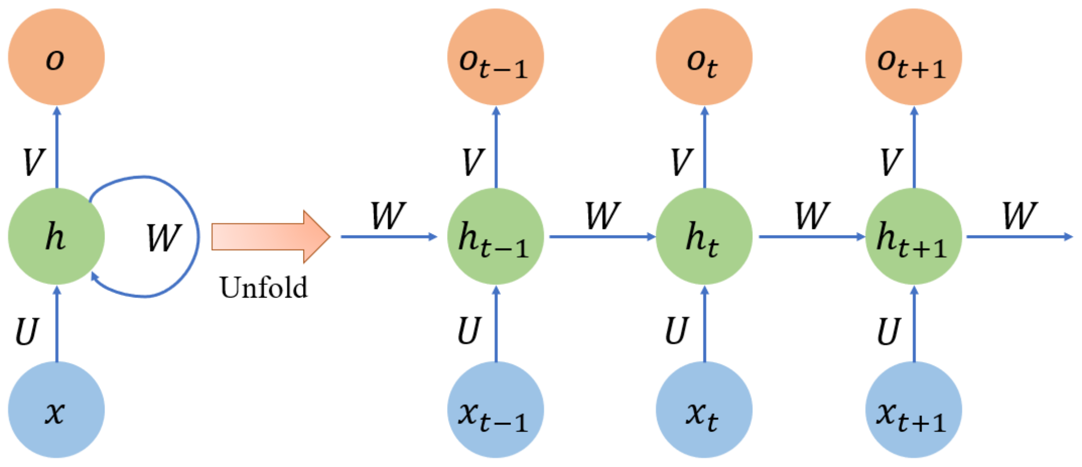


Fig. 2.1 Illustration of a simple RNN and its unfolding in time.

The mathematic representation of a simple RNN unit is as follows:

$$\mathbf{h}_t = f(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}) \tag{2.1}$$

$$\mathbf{o}_t = g(\mathbf{V}\mathbf{h}_t) \tag{2.2}$$

where $\mathbf{x}_t$ is the input vector at timestep t. Taking $\mathbf{x}_0$ as an example, it could be the representation vector for the first element in inputs. $\mathbf{h}_t$ is the hidden state at timestep t which is obtained based on current input $\mathbf{x}_t$ and previous hidden state $\mathbf{h}_{t-1}$. f is an activation function which is usually a non-linear function such as ReLU, tanh or Sigmoid. $\mathbf{U}, \mathbf{W}$ and $\mathbf{V}$ are parameter matrice to be learned. $\mathbf{o}_t$ indicates the output vector at timestep t which depends on a specific task, and g is an activation function. For example, if the task is next word prediction for a sentence, then the activation function g would be softmax function and $\mathbf{o}_t = \mathbf{softmax}(\mathbf{V}\mathbf{h}_t)$ would be a list of probabilities for all words in vocabulary.

2.1.2 Bidirectional RNN

Simple RNN has limitations as current state can only read previous inputs but not future inputs. Bidirectional Recurrent Neural Network (BRNN) [99] can read a sequence of data in both forward and backward directions. In other words, both forward and backward states information can be read by the output layer of BRNN at the same time. BRNN is particularly helpful when the contextual information of the input is needed.

Figure 2.2 gives an illustration of a BRNN which contains two hidden layers at each timestep t, for computing forward states and backward states, respectively. A simple BRNN unit can be defined as the following equations:

$$\mathbf{h}_t^f = f(\mathbf{U}^f\mathbf{x}_t + \mathbf{W}^f\mathbf{h}_{t-1}^f + \mathbf{a}^f) \tag{2.3}$$

$$\mathbf{h}_t^b = f(\mathbf{U}^b\mathbf{x}_t + \mathbf{W}^b\mathbf{h}_{t-1}^b + \mathbf{a}^b) \tag{2.4}$$

$$\mathbf{o}_t = g(\mathbf{V}[\mathbf{h}_t^f, \mathbf{h}_t^b] + \mathbf{c}) \tag{2.5}$$

where $\mathbf{h}_t^f$ denotes hidden state of the forward direction at timestep t and $\mathbf{h}_t^b$ is the hidden state of the backward direction. $\mathbf{o}_t$ denotes the output at timestep t which is generated based on the combination of the above two hidden states. $\mathbf{U}^f, \mathbf{U}^b, \mathbf{W}^f, \mathbf{W}^b, \mathbf{V}, \mathbf{a}^f, \mathbf{a}^b$ and $\mathbf{c}$ are parameters to be learned. f and g are two activation functions.

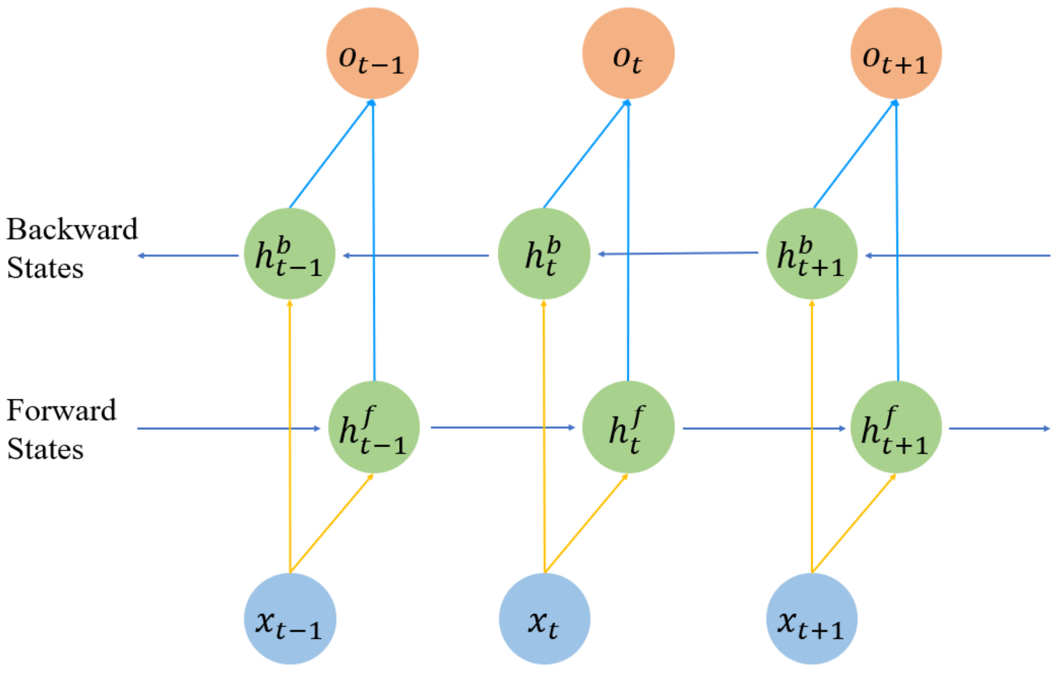


Fig. 2.2 Illustration of a bidirectional RNN.

2.1.3 Long Short-Term Memory Unit

Although simple RNN can theoretically make use of sequences with arbitrary length, it can solely look back a few timesteps due to the gradient vanishing and exploding problems [3], so it can not capture the long-term dependencies from very long sequences. Thus, the long short-term memory (LSTM) unit [44] was devised to alleviate the above limitation.

Figure 2.3 gives the inner structure of a LSTM unit. LSTM uses a memory cell to store information throughout the processing of the sequence, and three gates to control information in and out of the memory cell. The transition functions of LSTM are defined as follows:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_i) \tag{2.6}$$

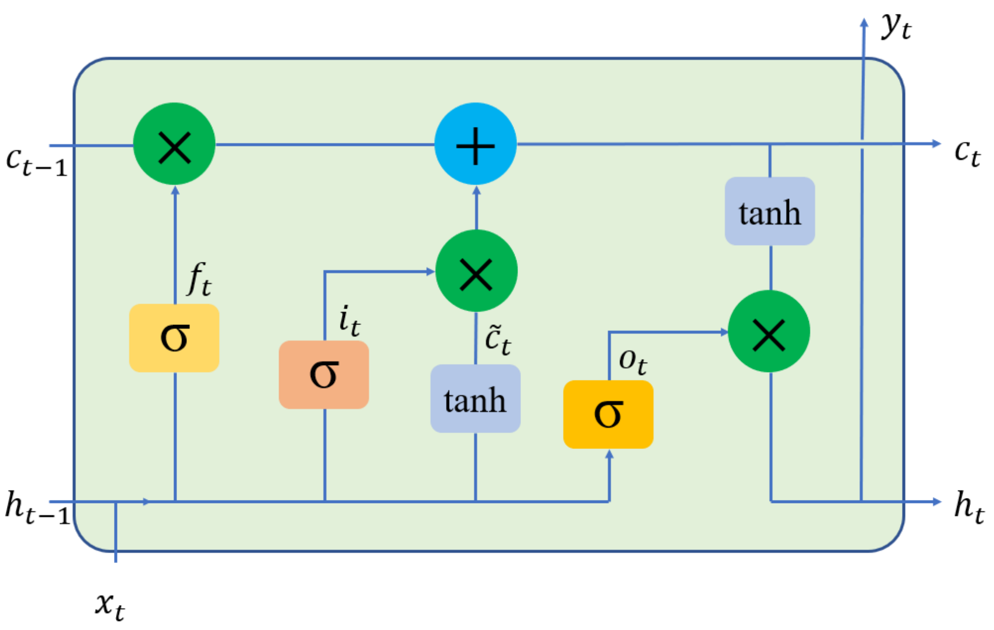


Fig. 2.3 Architecture of an LSTM unit

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{x}_t;\mathbf{h}_{t-1}] + \mathbf{b}_f) \tag{2.7}$$

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c[\mathbf{x}_t;\mathbf{h}_{t-1}] + \mathbf{b}_c) \tag{2.8}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{x}_t;\mathbf{h}_{t-1}] + \mathbf{b}_o) \tag{2.9}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t \tag{2.10}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{2.11}$$

where $\mathbf{x}_t$ is the input vector at timestep t. $\mathbf{h}_t$ and $\mathbf{h}_{t-1}$ denote the hidden states at timestep t and $t-1$, respectively. $\mathbf{i}_t, \mathbf{f}_t$, $\mathbf{o}_t$ and $\mathbf{c}_t$ are input gate, forget gate, output gate and cell state at timestep t, respectively. $\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o, \mathbf{W}_c, \mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o, \mathbf{b}_c$ are trainable parameters.

2.2 Recursive Neural Networks

Recursive neural networks (RecNNs) or tree-structured neural networks are a kind of artificial neural networks which utilise the same set of parameters recursively over a structured input.

2.2.1 Vanilla RecNN

The idea of RecNNs for NLP is to train a deep neural network with a syntactic tree structure [90] that can be employed to model phrases and sentences. The simplest member of RecNNs is the vanilla RecNN [110], in which the representation vector of the parent node is computed by weighted linear combination of the representation vectors of its child nodes.

Figure 2.4 shows a vanilla RecNN applied to a parsed sentence "She studies English". $\mathbf{e_1}, \mathbf{e_2}$ and $\mathbf{e_3}$ are representation vectors for words "She", "studies" and "English" in the sentence, respectively. The RecNN firstly concatenates $\mathbf{e_1}$ and $\mathbf{e_2}$ to form a representation vector $\mathbf{h_1}$ for the phrase "studies English":

$$\mathbf{h}_1 = \tanh(\mathbf{W}[\mathbf{e}_2; \mathbf{e}_3] + \mathbf{b}) \tag{2.12}$$

After that, the word representation vector $\mathbf{e}_1$ and the phrase vector $\mathbf{h}_1$ are concatenated to get a new vector $\mathbf{h}_2$ that represents the full sentence "She studies English".

$$\mathbf{h}_2 = \tanh(\mathbf{W}[\mathbf{e}_1; \mathbf{h}_1] + \mathbf{b}) \tag{2.13}$$

$\mathbf{W}$ and $\mathbf{b}$ are parameters to be trained. The obtained sentence representation vector can be applied to various NLP tasks including text matching.

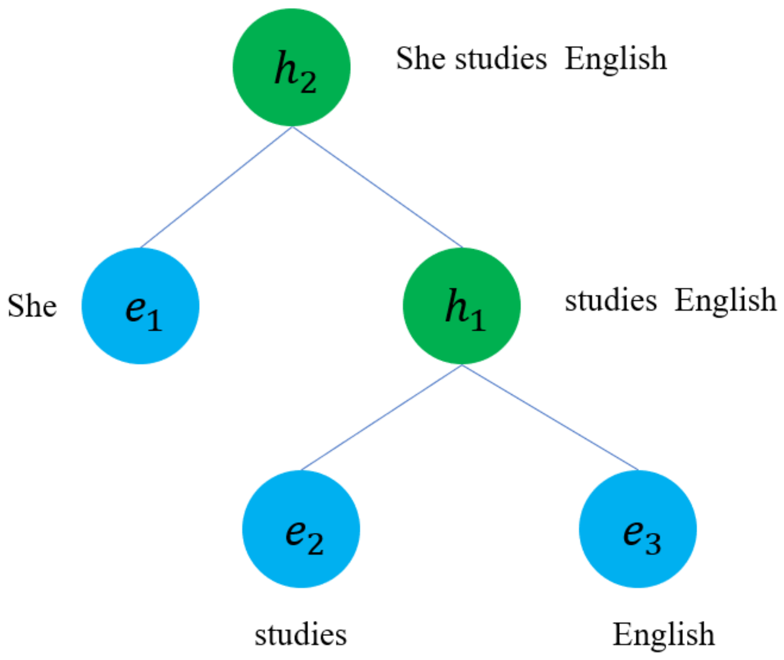


Fig. 2.4 A vanilla RecNN applied to a parsed sentence “She studies English”.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a special kind of feed-forward neural network and have been applied successfully for image processing [63, 53]. Unlike traditional feed-forward neural network, neurons in CNN can respond to a small part of neurons in the next layer. Generally, a CNN is usually composed of an input layer, an output layer and a stack of hidden layers. Each hidden layer is composed of a series of convolutional and pooling layers.

2.3.1 Convolutional Layer

While modeling natural language such as texts, a CNN takes as inputs a sequence of word representation vectors, and then creates representation vectors for all sub-phrases, and finally aggregates them together for future tasks.

Suppose we have a sequence of text with m words, denoted as $(x_1, x_2, ..., x_m)$, and word embeddings of these words, denoted as $(\mathbf{v}_1, \ldots \mathbf{v}_m)$ where $\mathbf{v}_i \in \mathbb{R}^d$, d indicates the length of each word embedding vector. Then the concatenation of words $\mathbf{v}_i, \ldots, \mathbf{v}_{i+1}, \ldots, \mathbf{v}_j$ can be denoted as $\mathbf{v}_{i:j}$. Finally, a convolution operation is used to run over the entire input text to generate new features. The convolution filter is denoted as $\mathbf{W} \in \mathbb{R}^{kd}$ where k denotes the window size of the convolution filter. A feature p_i is generated from a sequence of words within a window $x_{i:i+k-1}$ by the following equation:

$$\mathbf{p}_i = f(\mathbf{v}_{i:i+k-1}\mathbf{W}^T + b) \tag{2.14}$$

where f is an activation function and $b \in \mathbb{R}$ denotes the bias term. After the convolution filter $\mathbf{W}$ being applied to each possible text sequence with k words $x_{1:k}, x_{2:k+1}, \ldots, x_{m-k+1:m}$, a feature vector is generated as follows:

$$\mathbf{p} = [p_1, p_2, \ldots, p_{m-k+1}] \tag{2.15}$$

where $\mathbf{p} \in \mathbb{R}^{m-k+1}$.

2.3.2 Pooling Layer

Normally, a pooling layer is always used after a convolutional layer in CNNs. The reasons of using a pooling layer are two-fold: firstly, it provides a fixed-size output, which is typically required for further layers such as feed-forward neural network. As aforementioned, the convolutional output feature vector **p** has $m-k+1$ elements that is dependent on the input size m if the input text is not padded. Therefore, the outputs of the convolutional layer would be in variable lengths. By using pooling layers [52] the size of the outputs of the convolutional layer can be fixed, which equals to the number of filters used. There are several ways of doing pooling such as max-pooling and average-pooling. Taking max-pooling as an example, it simply takes the maximum element of the output feature vector **p**:

$$\hat{p} = \max\left[\mathbf{p}\right] \tag{2.16}$$

For example, if there are 1000 filters, then we will get a 1000-dimensional output after using max-pooling. Secondly, a pooling layer can reduce the output dimensionality by keeping the most salient information.

2.4 Neural Networks for Text Matching

Existing neural network based text matching models can be generally partitioned into two types, i.e., representation-based models and interaction-based models, respectively. The representation-based models aim to encode two texts to two representation vectors independently, and there is no interaction between two texts until the corresponding representations are obtained. The interaction-based models aim to make matching decision based on direct matching features and two texts are interacted before obtaining final text representations.

2.4.1 Representation-based Neural Networks for Text Matching

The main idea of the representation-based models is to encode two texts into two representation vectors independently using deep neural networks. Figure 2.5 shows the framework of the representation-based text matching model which is composed of four layers. The first is the input layer which is used to represent each word in a text into a vector using bag-of-words [40] or word embedding [79]. The second layer accepts word representation vectors from the first layer to generate two text representation vectors. The third layer aggregates the above two text representation vectors into a single vector using a matching function such as element-wise product. Finally, the aggregated vector is fed into the final classifier network to make prediction. The text representation layer is the core of the representation-based model. How to represent text into a vector with rich semantic information has great impact on the performance of the model.

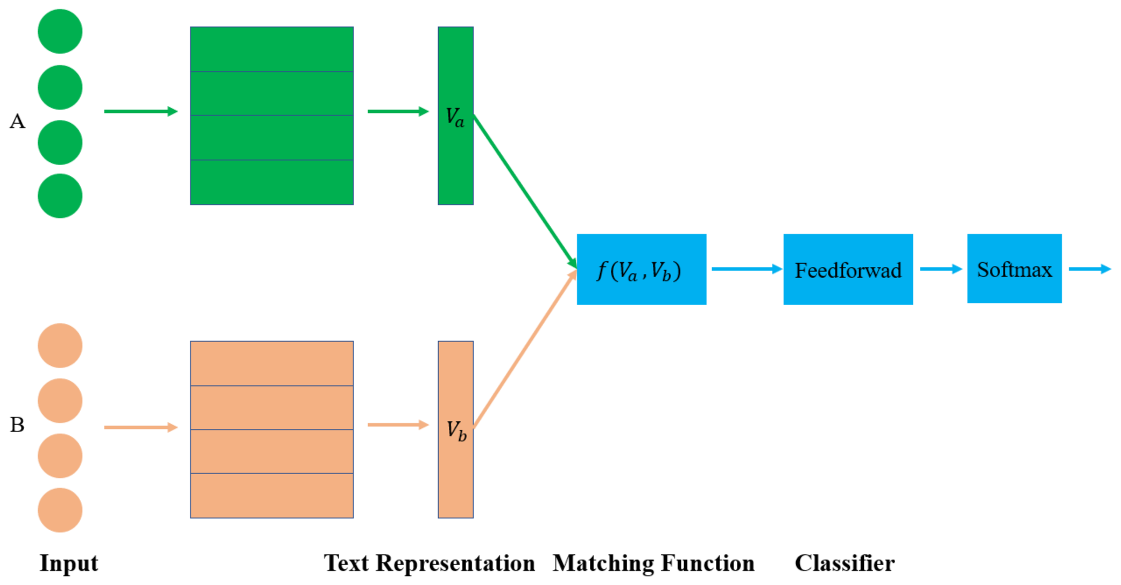


Fig. 2.5 Illustration of the framework of representation-based text matching model. V_a and V_b are text representations encoded by deep neural networks for text A and text B respectively. $f(V_a, V_b)$ is a matching function that used to aggregate V_a and V_b into a single vector for the final classifier.

2.4.1.1 Recurrent Neural Networks for Text Representation

Recurrent neural networks (RNNs) are a class of neural architectures which have been widely used to obtain text representations [126]. Generally, a RNN takes one word from the text as its input at each timestep and aggregates current word with its previous state. Then the compositional result over the whole text can be seen as the representation of the text. Among several variants of the simple RNN [30], the long short-term memory (LSTM) [44] has become a typical choice for RNNs recently. Benefiting from the well-designed gate mechanism in LSTM, gradient from distant terms can be delivered to the current term, thus it has the ability of tackling the gradient vanishing and exploding problem.

Although the complex gate mechanism in LSTM unit is helpful with the express power of RNNs, it sacrifices the computational efficiency of the models. Thus, how to simplify computation but keep the capacity of RNN models need to be further explored. One way is to simplify the inner structure of the LSTM unit. Cho et al. simplified LSTM unit by removing its cell state and proposed the gated recurrent unit (GRU) with only two gates, a reset gate to decide how much past information to forget, and an update gate to determine what new information to add [18]. Lee et al. removed the output gate and the recurrent non-linearity from LSTM and proposed the recurrent additive network (RAN), which generates hidden states with linear transformed inputs directly [64]. [151] devised a novel twin-gated mechanism and proposed the addition-subtraction twin-gated recurrent unit (ATR) with fewer matrix operations. The twin-gated mechanism can generate two gates with the same parameters through linear addition and subtraction operation.

Another way is to remove recurrent connections in order to promote parallelism of RNNs. [9] proposed the quasi-recurrent network (QRNN), which contains two components, a convolutional component and a pooling component. The convolutional component computes across both minibatches and spatial dimensions in parallel to get local input features. And the pooling component allows fully parallel computation across minibatch and feature dimensions to obtain global input features. [65] proposed a simple recurrent unit (SRU) which consists of a light recurrence component and a highway network. The recurrence component is strengthened via the highway network component. Zhang and Sennrich made use of ATR [151] and SRU [65] and proposed a lightweight recurrent network (LRN) with two gates, an input gate to decide how much information from the current input to add and a forget gate to determine what information from previous hidden state to forget, respectively.

Besides, to further increase the expressivity of recurrent neural networks, some works propose to stack multiple recurrent layers on top of each other in a hierarchical manner to capture hierarchical features [98, 29, 88, 43]. Nie and Bansal [83] proposed a stacked bidirectional LSTM-RNN with shortcut connections where hidden states from all previous layers are concatenated so as to make the backpropagation path short. Cellaware stacked LSTM (CAS-LSTM) [20] utilises both hidden states and cell states from the previous layer to ensure information flow to be delivered in both horizontal recurrence and vertical connections. [125] presented a recurrently controlled recurrent network (RCRN) which learns gating functions using recurrent networks. RCRN consists of two key components, a controller RNN to control the information flow and compositionality of the listener RNN, and a listener RNN to generate final output of the model.

2.4.1.2 Recursive Neural Networks for Text Representation

RNN models take a sentence as a flat sequence, without considering the structural information of the sentence. In contrast, tree-structured neural networks or recursive neural networks (RecNNs) take a sentence as a recursive structure [111]. Based on the pre-obtained syntactic parse tree, a RecNN model converts each word at a leaf node of the tree to a representation vector, and then uses a composition function to compose word/phrase pairs to get representations of the intermediate nodes of the tree. Finally, the representation of the root node is viewed as a representation of the sentence.

Earlier research on RecNNs mainly focuses on investigating effective composition functions. Socher et al. [110] firstly proposed the RecNN architecture and used a simple feed-forward neural network as the composition function of the model. Latterly, some more complex composition functions such as matrix-vector multiplication [109], tensor computation [111], TreeGRU [159] and TreeLSTM [117, 160] are proposed to improve the performance of the basic RecNN. TreeLSTM works in a bottom-up manner, which means the learned representation of a node is based on its child nodes, without considering information from higher up in the tree. By analogy with the bidirectional sequential network such as BiLSTM, bidirectional recursive networks have been proposed [61, 127, 13, 155].

Benefiting from considering the syntactic structure of sentences, RecNNs achieve impressive performance on many NLP tasks. However, a major limitation of these RecNN models is that the same composition function is used recursively over the syntactic tree, thus lacking the ability of distinguishing different syntactic compositions.

To alleviate this problem, a straightforward method is to utilise multiple composition functions. Socher et al. [108] selected a suitable composition function for each word/phrase pair according to its syntactic categories. Similarly, Dong et al. [27] introduced AdaRNN, which adaptively selects composition functions according to tags and child vectors. However, the predefined composition functions are usually designed manually for some specific tasks, thus the generalization ability of these models is limited. Recently, researchers proposed some models which can automatically perform dynamic composition with the aid of tags [91, 138, 46, 56]. Generally, tags are usually used as supplementary inputs for RecNNs in these models. For example, Wang et al. [138] and Huang et al. [46] employed tag embeddings as additional inputs to control the gates of TreeLSTM to conduct dynamic composition.

Other works such as [72, 101] took advantage of recent works on dynamic parameter prediction [39] and proposed to use a hypernetwork to generate parameters for RecNNs dynamically. DC-TreeLSTM [72] consists of two separate TreeLSTMs with similar structure but different number of parameters. The smaller TreeLSTM is employed to calculate weights of the bigger TreeLSTM. The two TreeLSTMs in a DC-TreeLSTM model share the same input, i.e., word embeddings. Shen et al. [101] proposed TG-HTreeLSTM, which improves the performance of DC-TreeLSTM by designing a complex information fusion layer which incorporates tag information and word information for hypernetwork TreeLSTM.

Finally, some latent tree-structured models have been proposed [148, 21, 75, 142], which accept only a sequence of words as inputs and learn sentence representation as well as its syntactic tree jointly from downstream tasks including text matching.

2.4.1.3 Other Neural Networks for Text Representation

The deep semantic structured model (DSSM) [47] is one of the earliest neural network based text matching model which uses a feed-forward neural network to map a query and a document to its representation vector. However, DSSM treats its inputs as a bag of words, which ignores the contextual structures within the input texts. To capture the contextual structures of the input texts, convolutional neural networks have been used for text representation [107, 45, 92, 31, 67]. CNN based representation models take as inputs representation vectors of input words sequentially, and then create representation vectors for all sub-phrases using a stack of convolution and pooling layers, and finally aggregates them together to generate a fixed-size representation vector in the final layer.

Recently, self-attention based architectures have been widely investigated in encoding texts due to their highly parallelizable computation [102, 103, 105, 121, 38, 122, 1]. Among them, the Transformer architecture [130] has attracted extensive attention and has made tremendous impact in the field of NLP. A Transformer encoder consists of several identical layers, and each layer is composed of a multi-head self-attention layer and a position-wise feed-forward network. Moreover, a positional encoding layer is used to model the order of the sequence.

In addition, graph neural networks (GNNs) [97] are also used for text representation [154, 73]. Zhang et al. proposed a sentence-state LSTM (S-LSTM) [154] which investigates a graph RNN for encoding sentences. S-LSTM obtains hidden states for all words simultaneously at each recurrent step rather than only one word at a time. [73] improved the performance of S-LSTM by introducing a depth-adaptive mechanism which allows the model to learn how many computational steps to conduct for different words as required.

2.4.1.4 Matching Function for Vector Aggregation

In spite of text representation component as shown in Figure 2.5, another important component is the matching function. The matching function is used to aggregate two text representation vectors into a feature vector which will be fed into the final classifier network for making matching decision. An appropriate matching function should have the ability of reflecting the nature of a task and providing sufficient information for the classifier network. [50] improved the performance of paraphrase identification by using element-wise sum and absolute element-wise difference as the matching function. Mou et al. [82] concatenated three matching heuristics including element-wise multiplication, element-wise difference and concatenation of the two sentence vectors to improve the performance of natural language inference. Chen et al. [16] modified the heuristic matching function in [82] by changing element-wise difference to absolute element-wise difference. [92] used a non-linear tensor layer as the matching function in order to capture more complicated interactions. [19] proposed a matching function ElBiS which is in bilinear form and can express different kinds of element-wise relations including element-wise product, sum, difference and absolute difference.

2.4.2 Interaction-based Neural Networks for Text Matching

The representation-based models encode two texts separately, therefore, there is no explicit interaction between two texts until the corresponding representation is obtained. The interaction-based models have been proposed which aim to capture direct matching features, and interaction between two texts is permitted before obtaining the final representation of each text.

2.4.2.1 Matching-Aggregation based Models

A number of recent studies on text matching are under the matching-aggregation or compare-aggregate framework [135]. Most of these models consist of the following five layers: input layer, contextual encoding layer, matching layer, aggregation layer and prediction layer, respectively.

The input layer is to represent words in input texts to the corresponding representation vectors. The contextual encoding layer encodes the contextual information into word representations. The matching layer matches smaller units such as words in one text against the other text. The aggregation layer aggregates all matching results from the previous layer into a fixed-size vector. Finally, the prediction layer is to make final matching decision.

Wang and Jiang [136] proposed the mLSTM model for NLI which is the earliest model under this framework. The mLSTM employs a LSTM to conduct word-by-word matching sequentially, and at each position, it tries to match the current word in the hypothesis with an attention-aware representation of the premise. Unlike mLSTM which only performs matching from a single direction, BiMPM [139] uses a BiLSTM to encode the contextual information into each word representations in two texts and then match them in two directions. Chen et.al. [15] proposed an enhanced sequential inference model (ESIM) which enhances the comparison approach of comparison layer by calculating the element-wise difference and product between the contextual representation and the attention-weighted representation of each word in two texts. Bian et.al. [4] pointed out that attention mechanism used in pervious matching-aggregation models would introduce noisy information of irrelevant words. Thus, they proposed a dynamic-clip attention mechanism, which can help filter irrelevant words by clipping their attention weights to 0. They also gave two implementations of the dynamic-clip attention mechanism, i.e., $K-max$ (save the largest k attention weights and set the others to 0) and $K-threshold$ (attention weights less than k are set to 0), respectively.

To capture more matching information, Tan et.al. [118] introduced a multiway attention network (MwAN) for text matching, which employs four types of attention functions to match two texts at word level, including minus attention, concat attention, dot attention and bilinear attention. Duan et.al. [28] pointed out that the network structures of previous models are too shallow to model complex relations. They presented an attention fused deep matching network (AF-DMN), which learns the attention-aware representations for two texts based on multi-level interactions. [68] proposed a multi-turn inference mechanism for NLI, which only focuses on a particular matching feature such as element-wise product and difference at each inference turn. Liu et al. [70] proposed the SAT-LSTMs matching model, which integrates syntax structure into attention model without using sequence based attention. [15] improved the ESIM model by explicitly considering syntactic parsing information of the input text.

Finally, external knowledge bases such as WordNet [80] and ConceptNet [112] have been used to improve text matching. [14] presented a knowledge-based inference model (KIM), which enriches the ESIM model [15] by incorporating semantic relation information of each word pair in two texts. And the semantic relations such as synonymy, antonymy, hypernymy, hyponymy and cohyponyms are extracted from WordNet [80]. Jiang et.al. [51] represented the aforementioned semantic relations in WordNet in the form of distributed relation features to argument word embedding. [152] adaptively incorporated knowledge from both WordNet and ConceptNet into neural network using a knowledge absorption gate.

2.4.2.2 Other Interaction-based Models

In addition to matching-aggregation based models, a number of recent studies on text matching take full advantage of the interaction matrix between two texts [34]. These models usually obtain a word-by-word interaction or co-attention matrix between two texts firstly, and then extract semantic features from the interaction matrix for final matching decision.

Hu et al. [45] proposed the architecture-II (ARC-II) model, which generates the interaction matrix by concatenating embedding words in two sentences and fed this matrix into a CNN to obtain semantic features. [86] viewed text matching as an image recognition problem and proposed the MatchPyramid model. In this model, the matching matrix containing the similarity information between words is regarded as an image. Different levels of matching patterns are learned in a hierarchical manner using a CNN. Densely interactive inference network (DIIN) [34] uses a linear layer to get a dense interaction tensor between two texts which contains more words interaction information. Yin et al. [147] proposed the attention-based convolutional neural network (ABCNN) which is also the first attempt to combine attention and CNNs for NLP task. Yin et al. [146] proposed MultiGranCNN, which learns representations for a sentence of different granularities such as words, phrases and sentence. Then the interaction matrix between two texts is constructed based on these three different level of representations. Similar with [146], [12] proposed the MIX model which fuses CNNs at multiple granularities.

Wan et al. [131] proposed the MV-LSTM model, which utilises a BiLSTM to generate multiple positional sentence representations firstly. The matching degree between two sentences are computed by aggregating interactions between generated positional sentence representations. Shen et al. [106] designed a gated relevance network (GRN) to capture complex semantic interactions, which incorporates bilinear model [116, 49] and single layer neural network [22] through the gate mechanism. Wan et al. [132] viewed text matching as a longest common subsequence problem and proposed the Match-SRNN model which uses a spatial RNN to integrate the local interactions over the word interaction tensor recursively.

Besides, attentive representation models are also used for text matching [74, 119]. The key to attentive representation models is also to represent texts, but this is quite different with aforementioned representation-based models. The representation-based models encode each text separately, while the attentive representation models encode one text by considering the impact of the other text.

Finally, there are some works based on dependent reading. [96] used two LSTMs with different parameters to read premise sentences and hypothesis sentences respectively for natural language inference. The last state of the premise LSTM is used to initialize the cell state of the hypothesis LSTM. Sha et al. [100] proposed a Re-read LSTM (rLSTM) for NLI. While processing the hypothesis sentence, rLSTM takes word representations of all words in premise as its additional inputs. The average of the outputs of rLSTM is used as the representation of the premise-hypothesis pair to make final matching decision. Ghaeini et al. [32] improved the performance of rLSTM by exploring dependency aspects of the premise-hypothesis during both encoding and inference stages.

2.5 Summary

This chapter firstly introduced some preliminary knowledge of three deep neural network architectures including recurrent neural networks, convolutional neural networks and recursive neural networks. Then, a detailed literature review on neural networks for text matching is presented. Table 2.1 gives a summary of two types of existing neural based text matching models reviewed in this thesis. The representation-based models focus on investigating neural architectures with strong expressive power to encode texts into a dense vector. The interaction-based models aim to construct neural networks with the ability of capturing more matching features between two texts. The merit of the representation-based models is that the final representation vectors of texts can be used for other purpose such as text classification [120], text clustering [134]. Moreover, most of representation-based models are based on a "Siamese" architecture [10], which have less parameters and can be trained easily. However, the interaction-based models usually outperform representation-based models in text matching.

Table 2.1 *A summary of existing neural network based text matching models reviewed in this thesis.*

Category		Summaries	References
Representation-based models	RNNs for text representation	• RNNs with LSTM to capture long-distance dependency • Various LSTM variants to simplify the inner structure of LSTM to increase computational efficiency • Stacked RNNs to increase the depth of the networks to improve expressive power of the model	[44, 18, 64, 151, 9, 65, 150] [20, 125, 83]
	RecNNs for text representation	• Take syntactic information into consideration while encoding each text into a representation vector by using various RecNNs such as TreeLSTM, TreeGRU and so on • Dynamic compositional models to distinguish different compositions, for example, a *verb-noun* (VP) composition and a *determiner-noun* (NP) composition should be treated differently in RecNNs • Latent tree-structured models to learn text representation and its syntactic tree jointly	[111, 110, 109, 159, 117, 160, 61, 127, 13, 155] [108, 27, 91, 138, 46, 56, 72, 101] [148, 21, 75, 142]
	Other networks for text representation	• CNNs to obtain local lexical connections such as *n*-gram expressions • Self-attention based models to encode text into a vector solely based on attention mechanism • Graph neural networks such as graph RNNs for representing texts which obtain the hidden states for all words simultaneously at each recurrent step, instead of only one word at a time	[107, 45, 92, 31, 67, 102, 103, 105, 121, 38, 122, 154, 73]
Interaction-based models	Matching-aggregation based models	• Matching results between representation and corresponding attention-aware representation of smaller units such as words are aggregated for making matching decision	[135, 136, 139, 15, 4, 118, 28, 68, 70, 14, 51, 152]
	Other interaction-based models	• Attentive representation models to encode one text by considering the impact of the other text • Matching models over interaction matrix which usually obtain a word-by-word interaction or co-attention matrix between two texts firstly, and then extract semantic features from the interaction matrix for final matching decision • Dependent reading models to explicitly consider the dependency of one text on the information of the other text	[74, 119] [34, 45, 86, 34, 147, 146, 12, 131, 132] [96, 100, 32]

Chapter 3

TreeLSTM with Tag-Aware Hypernetwork for Sentence Representation

This thesis aims to improve text matching in the following two perspectives: text representation and text interaction. This chapter focuses on investigating effective text representation model. Specifically, this chapter is to enhance the expressive power of the tree-structured neural network by introducing a hypernetwork into standard TreeLSTM to perform dynamic composition.

3.1 Introduction

Recursive neural networks (RecNNs) or tree-structured neural networks is one type of neural architecture which learns sentence representation by exploiting syntactic structures. Based on the pre-obtained syntactic parse tree, a RecNN model converts each word at a leaf node of the tree to a representation vector, and then uses a composition function to compose word/phrase pairs to get representations of the intermediate nodes of the tree. Finally, the representation of the root node is viewed as a representation of the sentence. However, a major limitation is that all kinds of compositions share the same parameters in a RecNN model as shown in Figure 3.1, neglecting the fact that different syntactic compositions exist which require different parameters for the RecNN model to handle precisely.

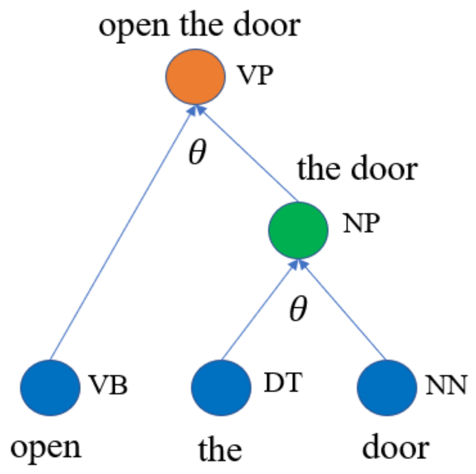


Fig. 3.1 A RecNN model where the parameter θ is shared by different kinds of syntactic compositions such as *verb-noun* (VP) composition and *determiner-noun* (NP) composition.

In order to distinguish different syntactic compositions, some dynamic compositional models are proposed, in which different composition functions are used for different compositions. One way of performing dynamic composition is with the aid of POS tags attached to nodes in the syntactic tree of a sentence [91, 46, 138, 101]. Usually, a low-dimensional distributed vector is used to represent each tag, namely tag embedding. Tag embeddings are learned and used together with word embeddings as inputs of the model. For example, Huang et al. [46] proposed TE-LSTM which takes tag embeddings as additional inputs to the gate functions of the TreeLSTM. A limitation of this kind of model is that the learned tag embeddings in these models are too simple to reflect the rich information that tags provide in different syntactic structures.

Another way of conducting dynamic composition is to take advantage of hypernetworks [39] which use a small network (i.e., "hypernetwork") to generate the weights for a larger network (i.e., main network). Liu et al. [72] proposed DC-TreeLSTM which is composed of two separate TreeLSTMs with similar structures but different numbers of parameters. The smaller TreeLSTM is employed to calculate the weights of the bigger TreeLSTM. A limitation of DC-TreeLSTM is that the two TreeLSTMs share the same inputs, i.e., word embeddings, thus, the model can only extract semantic information and lacks the ability of capturing syntactic information which is useful for dynamic composition. Although Shen et al. [101] considered syntactic information and improved the performance of DC-TreeLSTM by incorporating tag information and word information in the hypernetwork TreeLSTM, tag information is only a supplement to the word information. However, comparing with words, tags can help the model to distinguish different syntactic compositions more explicitly. Thus, how to use tags in a more efficient way for dynamic composition still need to be further explored.

In short, the syntactic information of a sentence has not been fully explored for dynamic composition in previous studies. To alleviate this limitation, we propose a new model, TagHyperTreeLSTM, which is composed of a tag-aware hypernetwork and a sentence encoder. The purposes of the tag-aware hypernetwork are two-fold: (1) to extract much more syntactic information by encoding some structural information into tag embeddings; and (2), to dynamically generate parameters for the sentence encoder. Specifically, the tag-aware hypernetwork is a standard TreeLSTM which only accepts tag embeddings at each node of a tree, and outputs a new tag representation of the node. Then these new tag representations, which encode structural information of nodes, will be used to generate parameters for the sentence encoder to perform dynamic composition. The sentence encoder is another TreeLSTM which accepts words as inputs and outputs the final sentence representation. The proposed TagHyperTreeLSTM model is evaluated on two typical NLP tasks: text classification and text semantic matching. The results show that TagHyperTreeLSTM is more expressive than previous models due to its ability of capturing both semantic and syntactic information.

The contributions of this work can be summarised as follows:

- We propose a new perspective on the usage of tags in a syntactic tree and devise a novel dynamic compositional model TagHyperTreeLSTM for sentence representation.
- Experimental results show that the proposed model achieves state-of-the-art performance among tree-structured models on six benchmark datasets with fewer parameters.
- An elaborate qualitative analysis is presented, giving an intuitive explanation of why our model works.

3.2 The Model

In this section, we present a novel dynamic compositional neural architecture, named **TagHyperTreeLSTM**, which consists of two components, i.e., tag-aware hypernetwork and sentence encoder. The tag-aware hypernetwork is a standard TreeLSTM which only accepts tag embeddings as inputs and outputs new tag representation of each node in a parse tree. The sentence encoder accepts word representations as inputs and outputs sentence representations. The parameters of sentence encoder are calculated based on the outputs of tag-aware hypernetwork. Figure 3.2 illustrates the proposed model.

In subsection 3.2.1, the standard TreeLSTM architecture is outlined. Subsection 3.2.2 describes the tag-aware hypernetwork and subsection 3.2.3 presents the sentence encoder.

3.2.1 TreeLSTM

LSTM is proposed to deal with the vanishing and exploding gradient problems of RNN, which can capture long-distance dependencies for sequential data due to its well-designed gate mechanism. TreeLSTM [117] extends LSTM to tree-structured topology and achieves impressive performance in sentence representation.

For each node j in a binary constituency tree of a sentence, let $\mathbf{x}_j = [x_1, \ldots, x_{d_e}]^T$ be an input vector, $\mathbf{h}^l_j = [h^l_1, \ldots, h^l_d]^T$ and $\mathbf{h}^r_j = [h^r_1, \ldots, h^r_d]^T$ be the hidden states of left child and right child, respectively. $\mathbf{c}^l_j = [c^l_1, \ldots, c^l_d]^T$ and $\mathbf{c}^r_j = [c^r_1, \ldots, c^r_d]^T$ be the memory cells of left child and right child, respectively. The composition function of a TreeLSTM unit can be described as follows:

$$\mathbf{i}_j = \sigma(\mathbf{W}_i[\mathbf{x}_j; \mathbf{h}^l_j; \mathbf{h}^c_j] + \mathbf{b}_i) \tag{3.1}$$

$$\mathbf{f}^l_j = \sigma(\mathbf{W}_l[\mathbf{x}_j; \mathbf{h}^l_j; \mathbf{h}^c_j] + \mathbf{b}_l) \tag{3.2}$$

$$\mathbf{f}^r_j = \sigma(\mathbf{W}_r[\mathbf{x}_j; \mathbf{h}^l_j; \mathbf{h}^c_j] + \mathbf{b}_r) \tag{3.3}$$

$$\mathbf{g}_j = \tanh(\mathbf{W}_g[\mathbf{x}_j; \mathbf{h}^l_j; \mathbf{h}^c_j] + \mathbf{b}_g) \tag{3.4}$$

$$\mathbf{o}_j = \sigma(\mathbf{W}_o[\mathbf{x}_j; \mathbf{h}^l_j; \mathbf{h}^c_j] + \mathbf{b}_o) \tag{3.5}$$

$$\mathbf{c}_j = \mathbf{f}^l_j \odot \mathbf{c}^l_j + \mathbf{f}^r_j \odot \mathbf{c}^r_j + \mathbf{i}_j \odot \mathbf{g}_j \tag{3.6}$$

$$\mathbf{h}_j = \mathbf{o}_j \odot \tanh(\mathbf{c}_j) \tag{3.7}$$

where $\mathbf{c}_j, \mathbf{h}_j \in \mathbb{R}^d$ refer to the memory cell and hidden state of node j. $\mathbf{i}_j, \mathbf{f}^l_j, \mathbf{f}^r_j, \mathbf{o}_j \in \mathbb{R}^d$ represent input gate, two forgot gates (left child and right child), and output gate, respectively. $\mathbf{g}_j \in \mathbb{R}^d$ is the newly composed input for the memory cell. $\mathbf{W}_i, \mathbf{W}_l, \mathbf{W}_r, \mathbf{W}_g, \mathbf{W}_o \in \mathbb{R}^{d\times(2d+d_e)}$ and $\mathbf{b}_i, \mathbf{b}_l, \mathbf{b}_r, \mathbf{b}_g, \mathbf{b}_o \in \mathbb{R}^d$ are trainable parameters. $[;]$ denotes the concatenation operation, tanh is the hyperbolic tangent, σ denotes the sigmoid function, and $\odot$ represents element-wise multiplication.

For simplicity, we describe the computation of the hidden state of node j at a high level with Equation (3.8) to facilitate references later in this chapter, and the detailed computation refers to Equations (3.1-3.7).

$$[\mathbf{h}_j; \mathbf{c}_j] = \mathbf{TreeLSTM}(\mathbf{x}_j, \mathbf{h}_j^l, \mathbf{h}_j^r, \mathbf{c}_j^l, \mathbf{c}_j^r) \tag{3.8}$$

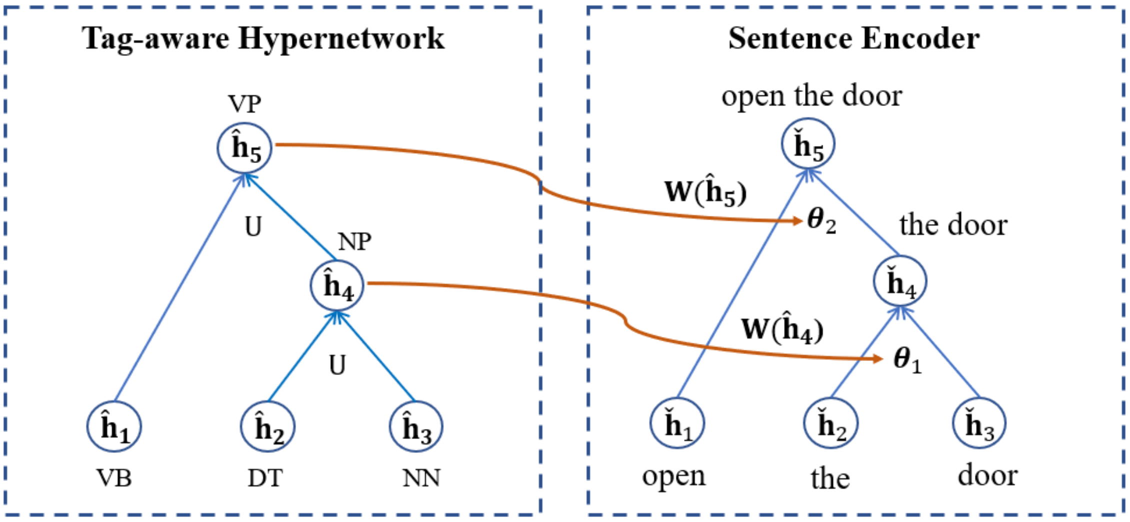


Fig. 3.2 An overview of the TagHyperTreeLSTM. The tag-aware hypernetwork is a s-tandard TreeLSTM which only accepts tag embeddings as inputs. $\mathbf{U}$ denotes parameters of tag-aware hypernetwork. $(\hat{\mathbf{h}}_1, \dots, \hat{\mathbf{h}}_5)$ and $(\check{\mathbf{h}}_1, \dots, \check{\mathbf{h}}_5)$ are hidden states of nodes in tag-aware hypernetwork and sentence encoder, respectively. $\mathbf{W}(\hat{\mathbf{h}}_4)$ and $\mathbf{W}(\hat{\mathbf{h}}_5)$ are inter-mediate hidden vectors computed based on $\hat{\mathbf{h}}_4$ and $\hat{\mathbf{h}}_5$. Parameters of the sentence encoder at each non-leaf node, i.e., θ_1 and θ_2, are not static, but are changed dynamically by the hidden vectors $\mathbf{W}(\hat{\mathbf{h}}_4)$ and $\mathbf{W}(\hat{\mathbf{h}}_5)$, respectively, and the detailed computation refers to subsection 3.2.3

3.2.2 Tag-aware Hypernetwork

In this subsection, the tag-aware hypernetwork in Figure 3.2 is described in detail. It is a standard TreeLSTM but accepts only tag embeddings as inputs. The purpose of the tag-aware hypernetwork is to generate parameters dynamically for the sentence encoder (the right side of Figure 3.2).

Formally, we denote tag embedding for the tag attached to each node j in a binary constituency tree as $\mathbf{e}_j = [e_1, \ldots, e_{d_t}]^T$. Then the hidden state $\hat{\mathbf{h}}_j \in \mathbb{R}^{d_t}$ and memory cell $\hat{\mathbf{c}}_j \in \mathbb{R}^{d_t}$ of node j are defined in the following way. If node j is a leaf node:

$$[\hat{\mathbf{h}}_j; \hat{\mathbf{c}}_j] = \tanh(\mathbf{V}\mathbf{e}_j + \mathbf{a}) \tag{3.9}$$

If node j is a non-leaf node:

$$[\hat{\mathbf{h}}_j; \hat{\mathbf{c}}_j] = \mathbf{TreeLSTM}(\mathbf{e}_j, \hat{\mathbf{h}}_j^l, \hat{\mathbf{h}}_j^r, \hat{\mathbf{c}}_j^l, \hat{\mathbf{c}}_j^r) \tag{3.10}$$

where **TreeLSTM** refers to Equation (3.8). $\mathbf{V} \in \mathbb{R}^{2d_t \times d_t}$ and $\mathbf{a} \in \mathbb{R}^{2d_t}$ are trainable parameters. The remaining notation follows Equations (3.1-3.7).

3.2.3 Sentence Encoder

In this subsection, we introduce the sentence encoder (the right side of Figure 3.2) which is used to compose word/phrase pair recursively over a binary constituency tree.

Formally, we denote sentence as a sequence of words $(w_1, w_2, ..., w_m)$ where m is the length of the sentence. Word embeddings of the sentence are denoted as $(\mathbf{v}_1, \ldots \mathbf{v}_m)$ where $\mathbf{v}_i = [v_1, \ldots, v_{d_w}]^T, i \in [1, m]$. Note that these words are leaf nodes in the constituency tree generated by a parser. Instead of using word embeddings directly, we firstly use a LSTM [44] on them, and then use each hidden state and memory cell of the LSTM as inputs for leaf nodes in the sentence encoder, which is effective for performance improvements on several NLP tasks [21].

Thus, for leaf node t, the hidden state $\bar{\mathbf{h}}_t \in \mathbb{R}^{d_h}$ and memory cell $\bar{\mathbf{c}}_t \in \mathbb{R}^{d_h}$ are computed in sequential order, with the corresponding input, i.e., word embedding $\mathbf{v}_t \in \mathbb{R}^{d_m}$ by the following equation:

$$[\bar{\mathbf{h}}_t; \bar{\mathbf{c}}_t] = \mathbf{LSTM}(\bar{\mathbf{h}}_{t-1}, \bar{\mathbf{c}}_{t-1}, \mathbf{v}_t) \tag{3.11}$$

where $\bar{\mathbf{h}}_{t-1} \in \mathbb{R}^{d_h}$ and $\bar{\mathbf{c}}_{t-1} \in \mathbb{R}^{d_h}$ refer to the hidden state and memory cell of LSTM at $(t-1)^{th}$ time-step. The hidden state $\bar{\mathbf{h}}_t$ and memory cell $\bar{\mathbf{c}}_t$ can be utilised as inputs to the sentence representation TreeLSTM, with the left (right) child of the target node j corresponding to the t^{th} word in the input sentence as follows:

$$[\check{\mathbf{h}}_j^{\{l,r\}}; \check{\mathbf{c}}_j^{\{l,r\}}] = [\bar{\mathbf{h}}_t; \bar{\mathbf{c}}_t] \tag{3.12}$$

Then, for each none-leaf node j, a TreeLSTM with dynamic parameters is used to obtain its hidden state $\check{\mathbf{h}}_j \in \mathbb{R}^{d_h}$ and memory cell $\check{\mathbf{c}}_j \in \mathbb{R}^{d_h}$ as follows:

$$\mathbf{W}(\hat{\mathbf{h}}_j) = \mathbf{W}_d \hat{\mathbf{h}}_j + \mathbf{b}_d \tag{3.13}$$

$$\check{\mathbf{i}}_j = \sigma(\mathbf{W}(\hat{\mathbf{h}}_j) \odot \mathbf{W}_i [\check{\mathbf{h}}_j^l; \check{\mathbf{h}}_j^c] + \mathbf{W}(\hat{\mathbf{h}}_j) \odot \check{\mathbf{b}}_i) \tag{3.14}$$

$$\check{\mathbf{f}}_j^l = \sigma(\mathbf{W}(\hat{\mathbf{h}}_j) \odot \mathbf{W}_l [\check{\mathbf{h}}_j^l; \check{\mathbf{h}}_j^c] + \mathbf{W}(\hat{\mathbf{h}}_j) \odot \check{\mathbf{b}}_l) \tag{3.15}$$

$$\check{\mathbf{f}}_j^r = \sigma(\mathbf{W}(\hat{\mathbf{h}}_j) \odot \mathbf{W}_r [\check{\mathbf{h}}_j^l; \check{\mathbf{h}}_j^c] + \mathbf{W}(\hat{\mathbf{h}}_j) \odot \check{\mathbf{b}}_r) \tag{3.16}$$

$$\check{\mathbf{g}}_j = \sigma(\mathbf{W}(\hat{\mathbf{h}}_j) \odot \mathbf{W}_g [\check{\mathbf{h}}_j^l; \check{\mathbf{h}}_j^c] + \mathbf{W}(\hat{\mathbf{h}}_j) \odot \check{\mathbf{b}}_g) \tag{3.17}$$

$$\check{\mathbf{o}}_j = \sigma(\mathbf{W}(\hat{\mathbf{h}}_j) \odot \mathbf{W}_o [\check{\mathbf{h}}_j^l; \check{\mathbf{h}}_j^c] + \mathbf{W}(\hat{\mathbf{h}}_j) \odot \check{\mathbf{b}}_o) \tag{3.18}$$

$$\check{\mathbf{c}}_j = \check{\mathbf{f}}_j^l \odot \check{\mathbf{c}}_j^l + \check{\mathbf{f}}_j^r \odot \check{\mathbf{c}}_j^r + \check{\mathbf{i}}_j \odot \check{\mathbf{g}}_j \tag{3.19}$$

$$\check{\mathbf{h}}_j = \check{\mathbf{o}}_j \odot \tanh(\check{\mathbf{c}}_j) \tag{3.20}$$

where $\mathbf{W}_d \in \mathbb{R}^{d_h \times d_h}$ and $\mathbf{b}_d \in \mathbb{R}^{d_h}$ are trainable parameters. $\mathbf{W}(\hat{\mathbf{h}}_j) \in \mathbb{R}^{d_h}$ is an intermediate hidden vector computed based on the output of the tag-aware hypernetwork at node

j, which modifies the corresponding static parameters $\mathbf{W}_i, \mathbf{W}_l, \mathbf{W}_r, \mathbf{W}_g, \mathbf{W}_o \in \mathbb{R}^{d_h \times 2d_h}$ and $\check{\mathbf{b}}_i, \check{\mathbf{b}}_l, \check{\mathbf{b}}_r, \check{\mathbf{b}}_g, \check{\mathbf{b}}_o \in \mathbb{R}^{d_h}$ through linearly scaling each row in the weight matrix. The remaining notation follows Equations (3.1-3.7). Finally, the hidden state of the root node $\check{\mathbf{h}}^{\mathbf{root}} \in \mathbb{R}^{d_h}$ is used as the representation for the given sentence.

3.3 Applications of TagHyperTreeLSTM

This section presents the application of TagHyperTreeLSTM for two typical NLP tasks.

Text classification. Given a sentence s and a pre-defined class set $\mathscr{Y}$, text classification is to predict a label $\hat{y} \in \mathscr{Y}$ for s. A single layer MLP followed by a softmax classifier is applied on the sentence representation $\check{\mathbf{h}}^{\mathbf{root}}$ to obtain the final predicted probability distribution of class y given sentence s as follows:

$$\mathbf{h}_s = \mathbf{Relu}(\mathbf{W}_s \check{\mathbf{h}}^{\mathbf{root}} + \mathbf{b}_s) \tag{3.21}$$

$$p(y|s) = \mathbf{softmax}(\mathbf{W}_c \mathbf{h}_s + \mathbf{b_c}) \tag{3.22}$$

where $\mathbf{h}_s \in \mathbb{R}^{d_s}$ is the intermediate feature vector for the softmax classifier. $\mathbf{W}_s \in \mathbb{R}^{d_s \times d_h}, \mathbf{b_s} \in \mathbb{R}^{d_s}, \mathbf{W}_c \in \mathbb{R}^{d_c \times d_s}, \mathbf{b_c} \in \mathbb{R}^{d_c}$ are trainable parameters.

Text matching. Text matching is to predict the relationship $\hat{l} \in \mathscr{L}$ between a given sentence pair $s1$ and $s2$ from a pre-defined label set $\mathscr{L}$. Firstly, the same TagHyperTreeLSTM is used to encode $s1$ and $s2$ into two sentence representation vectors $\check{\mathbf{h}}^{\mathbf{root}}_{\mathbf{s1}}, \check{\mathbf{h}}^{\mathbf{root}}_{\mathbf{s2}} \in \mathbb{R}^{d_h}$. Next, some matching heuristics [82] are used to combine the two sentence vectors together to generate a intermediate feature vector for the final classifier in the following way:

$$\mathbf{h}_{st} = [\check{\mathbf{h}}^{root}_{s1}, \check{\mathbf{h}}^{root}_{s2}, \check{\mathbf{h}}^{root}_{s1} \odot \check{\mathbf{h}}^{root}_{s2}, |\check{\mathbf{h}}^{root}_{s1} - \check{\mathbf{h}}^{root}_{s2}|] \tag{3.23}$$

$$\mathbf{h}_m = \mathbf{Relu}(\mathbf{W}_m \mathbf{h}_{st} + \mathbf{b}_m) \tag{3.24}$$

where $\mathbf{h}_m \in \mathbb{R}^{d_m}$ is the intermediate feature vector for the following classifier. $\mathbf{W}_m \in \mathbb{R}^{d_m \times 4d_h}, \mathbf{b}_m \in \mathbb{R}^{d_m}$ are trainable parameters. Finally, the probability distribution of label l given sentence pair $s1$ and $s2$ is obtained using a softmax classifier:

$$p(l|(s1, s2)) = \mathbf{softmax}(\mathbf{W}_l \mathbf{h}_m + \mathbf{b}_l) \tag{3.25}$$

where $\mathbf{W}_l \in \mathbb{R}^{d_l \times d_m}, \mathbf{b}_l \in \mathbb{R}^{d_l}$ are again trainable parameters. The parameters of the model are learned to minimise the cross-entropy of the distributions between predicted and true labels.

3.4 Experiments

3.4.1 Datasets

The proposed model is evaluated on four benchmark datasets for text classification (SST2, MR, SUBJ, TREC) and two datasets (SICK and SNLI) for text matching:

- SST2: Stanford Sentiment Treebank consisting of movie reviews with positive or negative label [111].
- MR: The movie reviews with positive or negative label [85].
- SUBJ: Sentences grouped as being either subjective or objective [84].
- TREC: A dataset which groups questions into six different question types [66].
- SICK: A textual entailment dataset with three classes (entailment, neutral, contradiction) [76].

- SNLI: The stanford natural language inference dataset with three classes (entailment, neutral, contradiction) [7].

Tabel 3.1 shows statistics of the six datasets used in this work.

Table 3.1 *Statistics of six benchmark datasets for two tasks. Train, Dev and Test are the size of training, validation and test dataset, respectively. CV means 10-fold cross validation is used. L_{avg} is the average number of words in sentences. $|V|$ is the size of vocabulary. $|T|$ is the number of tags. Class is the size of label/class set.*

Dataset	**Train**	**Dev**	**Test**	$\mathbf{L_{avg}}$	**|V|**	**|T|**	**Class**
SST2	6920	872	1821	18	15K	73	2
MR	10662	-	CV	22	19K	73	2
SUBJ	10000	-	CV	21	21K	72	2
TREC	75952	-	500	10	10K	67	6
SICK	4500	500	4927	10	2K	41	3
SNLI	549K	9800	9800	10	36K	72	3

3.4.2 Experimental Setup

For all the datasets, sentences are tokenized and parsed by Stanford PCFG parser[1] [60]. Word embeddings are initialized with the 300-dimensional GloVe word vectors [89], and embeddings of out-of-vocabulary words and tags are initialized by randomly sampling from the uniform distribution $[-0.005, 0.005]$. Tag embeddings are fine-tuned during training procedure while word embeddings are fixed. Hidden size for tag-aware hypernetwork and the dimension of tag embeddings are fine-tuned from $[25, 50]$. Hidden size for sentence representation TreeLSTM is 100 for all datasets. Batch size is selected from $[32, 64]$. Weights of the model are trained by minimizing the cross-entropy of the training dataset

[1]https://nlp.stanford.edu/software/lex-parser.shtml

by the Adadelta [149] optimizer. The initial learning rate is 1. The accuracy metric is used in this work to measure the performance of the proposed and all comparison models on six datasets.

3.4.3 Results

Text Classification. The proposed model is compared with two kinds of models, i.e. tree-structured models and other neural models. Table 3.2 shows test accuracies of the proposed TagHyperTreeLSTM and all comparison models on four text classification datasets. Generally, compared with all baseline models, the proposed TagHyperTreeLSTM achieves superior or competitive performance on all four text classification datasets.

From table 3.2, we can observe that, firstly, compared with previous tree-structured models, TagHyperTreeLSTM sets a new state of the art on all four datasets - SST2, MR, SUBJ and TREC with accuracies of 91.2%, 83.7%, 95.2% and 96.0%, respectively.

Secondly, compared with DC-TreeLSTM and TG-HTreeLSTM which are two models very related to our work, TagHyperTreeLSTM outperforms them on all four datasets. DC-TreeLSTM is the first work that employs a hypernetwork to generate parameters dynamically for a TreeLSTM which is responsible for sentence representation. Note that the hypernetwork and the sentence representation TreeLSTM share the same input (i.e., word embeddings) in DC-TreeLSTM. Different from DC-TreeLSTM, the hypernetwork in our model is based on tag embeddings without using any word information. TagHyper-TreeLSTM is superior to the DC-TreeLSTM on four datasets SST2, MR, SUBJ and TREC with 3.4%, 2%, 1.5% and 2.2% improvements, respectively. A possible reason is that tag information is more useful for distinguishing different syntactic compositions than word information. Moreover, TagHyperTreeLSTM is slightly better than TG-HTreeLSTM on all four datasets SST2, MR, SUBJ and TREC with 0.8%, 1.1%, 0.3% and 0.2% improvements, respectively. TG-HTreeLSTM uses tag information as a supplement to word information

Table 3.2 *Accuracies of previous neural models and proposed model on four different text classification datasets. The symbol * indicates models that are pre-trained with large external corpora. The symbol † indicates our implementations.*

Model	**SST2**	**MR**	**SUBJ**	**TREC**
Tree-structured neural models				
RecNN [110]	82.4	76.4	91.8	90.2
RNTN [111]	86.4	-	-	-
AdaMC-RNTN [27]	87.1	-	-	-
TE-RNN [91]	86.5	77.9	-	-
TreeLSTM [117]	88.0	-	-	-
AdaHT-LSTM [71]	87.8	81.9	94.1	-
TE-LSTM [46]	89.6	82.2	-	-
BiTreeLSTM [128]	90.3			94.8
DC-TreeLSTM [72]	87.8	81.7	93.7	93.8
Gumbel Tree-LSTM [21]	90.7	-	-	-
TG-HTreeLSTM [101]	90.4	82.6	94.9	95.8
Other neural models				
CNN [57]	88.1	81.5	93.4	93.6
LSTM [117]	84.9	-	-	-
BCN + Char + CoVe* [77]	90.3	-	-	95.8
byte-mLSTM* [93]	<u>91.8</u>	<u>86.9</u>	94.6	
DiSAN [102]	-	-	94.2	94.2
DARLM [158]	-	83.2	94.1	96.0
WSAN [48]	-	83.2	94.6	95.0
Transformer† [130]	86.9	80.2	94.1	91.9
star-Transformer† [37]	87.1	80.7	93.6	93.0
Bert*† [25]	90.8	85.8	<u>96.0</u>	<u>96.8</u>
TagHyperTreeLSTM (proposed)	**91.2**	**83.7**	**95.2**	**96.0**

and devises a complex information fusion layer for the hypernetwork. Compared with TG-HTreeLSTM, the proposed model achieves better performance with fewer parameters thus is more effective and efficient.

Thirdly, compared with other neural models, TagHyperTreeLSTM shows its consistently better performance on four datasets. Althogh Bert achieves better performance than TagHyperTreeLSTM, it is pre-trained with large external corpora while TagHyperTreeLSTM do not perform any pre-training. As table 3.2 shows, TagHyperTreeLSTM outperforms all the models that do not perform pre-training, including recently published transformer based model such as star-Transformer.

Text Matching. To evaluate the proposed TagHyperTreeLSTM on other NLP tasks, we also conduct an experiment on text matching. Different from text classification requiring only one sentence at a time, each example in a text semantic matching dataset consists of two sentences. In this work, we evaluate the performance of TagHyperTreeLSTM on SICK and SNLI datasets.

Table 3.3 shows experimental results on SICK dataset. The performance of RNN, LSTM, RecNN, and RNTN are reported in [7]. The performance of MV-RNN is reported in [72], and performance of the other models come from respective papers. We can find that TagHyperTreeLSTM again demonstrates its superior performance compared against baseline models with an accuracy of 83.9%, and is slightly better than the best baseline model TG-HTreeLSTM with an improvement of 0.6%. Table 3.4 gives experimental results on SNLI dataset. For fair comparison, we only consider sentence-encoding based models. The performance of TagHyperTreeLSTM is on par with the previous tree-structured models. This comparison shows that TagHyperTreeLSTM is also effective in text sematic matching task, and has generalization ability in different NLP tasks.

Table 3.3 *Accuracies of previous neural models and proposed model on the SICK dataset.*

Model	**Acc (%)**
RNN [7]	72.2
LSTM [7]	77.6
RecNN[8]	74.9
RNTN [8]	76.9
MV-RNN [72]	75.5
DC-TreeLSTM [72]	82.3
TG-HTreeLSTM [101]	83.3
TagHyperTreeLSTM	<u>83.9</u>

Table 3.4 *Accuracies of previous neural models and proposed model on the SNLI dataset.*

Model	**Acc (%)**
LSTM [7]	80.6
Tree-based CNN [82]	82.1
SPINN-PI [6]	83.2
Gumbel Tree-LSTM [21]	<u>85.6</u>
TagHyperTreeLSTM	85.5

Table 3.5 *Comparison of number of parameters of different models on SST2 dataset.*

Model	**# Params**	**Acc (%)**
CNN [57]	$360K$	88.1
LSTM [117]	$316K$	84.9
TreeLSTM [117]	$317K$	88.0
TE-LSTM [46]	$919K$	89.6
Gumbel Tree-LSTM [21]	$1M$	90.7
TG-HTreeLSTM [101]	$486K$	90.4
TagHyperTreeLSTM	**418K**	**91.2**

3.4.4 Analysis

3.4.4.1 An Observation on Model Complexity

In this subsection, a comparison of the number of parameters between the proposed TagHyperTreeLSTM and some typical neural models on the SST2 dataset is presented in Table 3.5. Firstly, compared with basic TreeLSTM, the performance of TagHyperTreeLSTM is improved by 3.2%, but the number of parameters are only increased by about $100K$. Secondly, compared with previous models which also use tag information, such as TE-LSTM and TG-HTreeLSTM, the proposed TagHyperTreeLSTM outperforms them with fewer parameters. This comparison demonstrates that TagHyperTreeLSTM is more effective and efficient than previous models.

3.4.4.2 Configuration Study

In this section, we present a configuration study on key modules of the proposed model to explore their effectiveness. As shown in Table 3.6, if we use word information instead of tag information as the input of the hypernetwork (the left side of Figure 3.2), the accuracy of the model drops to 95.6% on TREC dataset and 83.3% on SICK dataset, respectively.

Table 3.6 *A configuration study on key modules of TagHyperTreeLSTM. Test accuracies on TREC and SICK datasets are reported. word: only use word information in hypernetwork. word + tag: use both word and tag information in hypernetwork. -LSTM: remove LSTM layer in sentence encoder.*

Model	**TREC**	**SICK**
TagHyperTreeLSTM	96.0	83.9
word	95.6	83.3
word + tag	96.0	84.0
-LSTM	95.2	83.3

A possible reason for this performance degradation is that tag information is more explicit and useful in distinguishing different syntactic structures than word information. Moreover, we do not get much improvement (no improvement on TREC dataset and only 0.1% improvement on SICK dataset) using both word and tag information in the hypernetwork. This may be because syntactic tags have provided enough information in distinguishing different syntactic structures, thus the contribution of word information is very limited. If we remove LSTM layer in sentence encoder (subsection 3.2.3), the accuracy of the model drops on both datasets with more than 0.5% degradation. Thus, using LSTM to get contextualized word representations as the inputs of the sentence encoder is crucial for performance improvement.

3.4.4.3 Qualitative Analysis

As described in previous sections, in order to distinguish different syntactic compositions, parameters of the sentence encoder at each non-leaf node j are not static, but are changed dynamically by a latent vector $\mathbf{W}(\hat{\mathbf{h}}_j)$, which is computed based on the hidden state of the tag-aware hypernetwork of node j. To get an intuitive understanding on how the latent vector $\mathbf{W}(\hat{\mathbf{h}}_j)$ works, we design an experiment to explore behaviours of neurons in $\mathbf{W}(\hat{\mathbf{h}}_j)$.

Table 3.7 *Some interpretable neurons and the phrases/clauses captured by these neurons. Symbol + splits the left child and right child of current node.*

	Node tags	**Neurons**	**Child nodes tags**	**Examples**
Phrase Level	WHADJP	39*th*	WRB+JJ	how+fast, how+much how+tall, how+far
	WHNP	77*th*	WDT+NN	what+continent, what+year what+color, what+province
	PP	88*th*	IN+NP	in+1913, of+yugoslavia for+june, in+algeria
		95*th*	IN+NP	in+a galon, in+a ton, in+a mile in+the neuschwanstein castle
	NP	65*th*	NNP+NNP	euphrates+river, eiffel+tower national+forest
		15*th*	JJ+NN	acid+rain, compounded+interest nuclear+power, trivial+pursuit
		30*th*	DT+NN	the+calculator, a+thyroid a+carcinogen, an+earthquake
			NN+NN	fuel+cell, state+flower yak+milk, spirometer+test
Clause Level	SQ	91*st*	VBZ+NP	is+acid rain is+john wayne airport
			VBP+NP	are+the rocky mountains are+in the troposphere
			VBD+NP	founded+american red cross invented+the hula hoop
		37*th*	VBZ+NP	is+the population of seattle is+the capital of mongolia

We randomly sample some sentences on the test set from the TREC dataset. Table 3.7 presents some interpretable neurons and some representative phrases or clauses captured by these neurons. By analyzing the maximum activation neurons in $\mathbf{W}(\hat{\mathbf{h}}_j)$ at each non-leaf node j, we find different neurons focus on capturing different syntactic compositions. For example, the $65th$, $15th$ and $30th$ neurons are more sensitive to *noun phrases* (NP) compositions, while the $88th$ and $95th$ neurons are more sensitive to *prepositional phrases* (PP) compositions. Moreover, note that although the $65th$, $15th$ and $30th$ neurons are sensitive to the same phrase type i.e., NP, the child node tags of the phrase are quite different. For example, the $65th$ neuron is sensitive to noun phrases composed of two *proper nouns* (NNP), while the $15th$ is sensitive to noun phrases composed of an *adjective* (JJ) and a *noun* (NN).

Moreover, neurons with large value in $\mathbf{W}(\hat{\mathbf{h}}_j)$ are dominated by nodes with specific syntactic structure or semantic basis which are significant for text classification and text semantic matching. Figures 3.3 and 3.4 show two examples. In Figure 3.3, the $9th$ neuron of $\mathbf{W}(\hat{\mathbf{h}}_j)$ is sensitive to sentences starting with “Who”, which is crucial for the model to classify these sentences to the label “HUM”. Figure 3.4 gives an example for text semantic matching. The 73*rd* neuron of $\mathbf{W}(\hat{\mathbf{h}}_j)$ monitors verb phrases such as “running down” and “standing still" which are more helpful for recognizing the relation between the sentence pair “A man is standing still” and “A man is running down the road”.

To have a better understanding of how the syntactic and dynamic composition parameters improve the performance, we analyse some examples (see table 3.8) from the SST2 test set where our TagHyperTreeLSTM predicts correctly while TreeLSTM without using syntactic tags fails. By analyzing the maximum activation of the latent vector$\mathbf{W}(\hat{\mathbf{h}}_j)$, we find that the $10th$ neuron in $\mathbf{W}(\hat{\mathbf{h}}_j)$ is sensitive to emotional terms, which can be regarded as a sentinel, and more attention should be paid in the process of composition. Figure 3.5 gives a visualization of the parser tree and values of the $10th$ neuron of $\mathbf{W}(\hat{\mathbf{h}}_j)$ for sentence “a whole lot foul , freaky and funny .”. We can see that the $10th$ neuron in

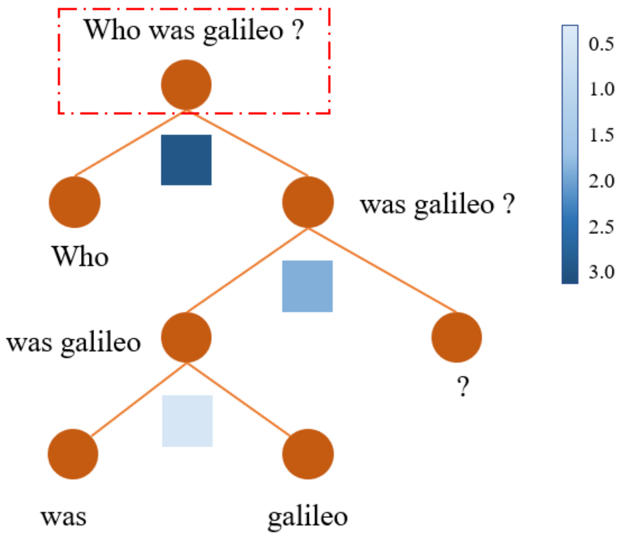
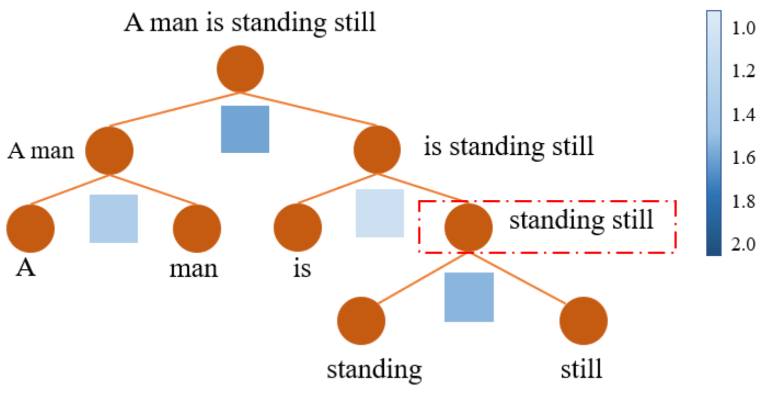


Fig. 3.3 A visualization of values of the $9th$ neuron in $\mathbf{W}(\hat{\mathbf{h}}_j)$ for sentence "Who was galileo ?". The color of the square below non-leaf nodes indicates the value of the $9th$ neuron of $\mathbf{W}(\hat{\mathbf{h}}_j)$.

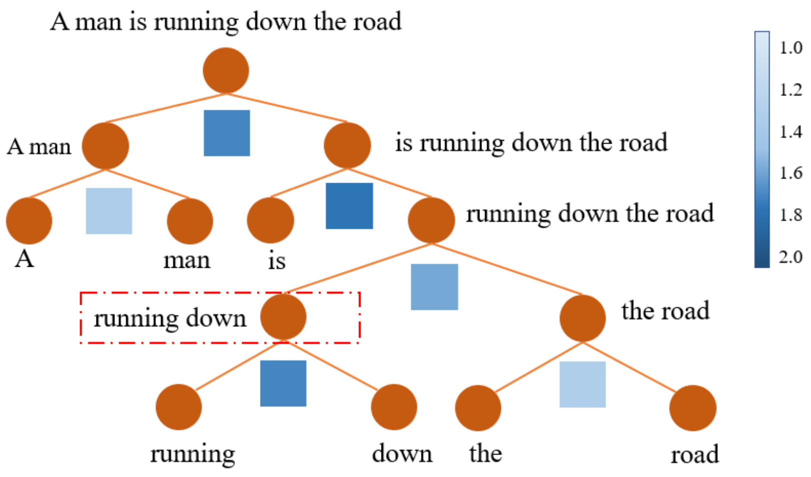
$\mathbf{W}(\hat{\mathbf{h}}_j)$ has the biggest value at phrase "foul, freaky and funny", which indicates that the model has realized that the adjective phrase "foul, freaky and funny" is important for the final sentiment classification. This suggests TagHyperTreeLSTM has the ability of distinguishing informative phrases whilst encoding sentence, and the expressive powder of TagHyperTreeLSTM is better.

3.5 Summary

In this chapter, we have presented a novel dynamic compositional architecture, named TagHyperTreeLSTM, for learning sentence representation, which has better expressive power due to its ability of distinguishing different syntactic compositions. The model consists of two components, i.e., sentence encoder and tag-aware hypernetwork. The purpose of the tag-aware hypernetwork is to extract syntactic information and to generate parameters dynamically for the sentence encoder. The sentence encoder is to output the



(a) A visualization of values of the 73*rd* neuron in $\mathbf{W}(\hat{\mathbf{h}}_j)$ for sentence "A man is standing still ".



(b) A visualization of values of the 73*rd* neuron in $\mathbf{W}(\hat{\mathbf{h}}_j)$ for sentence "A man is running down the road ".

Fig. 3.4 A visualization of values of the 73*rd* neuron in $\mathbf{W}(\hat{\mathbf{h}}_j)$ for sentence pair "A man is running down the road/ A man is standing still". The color of the square below non-leaf nodes indicates value of the 73*rd* neuron in $\mathbf{W}(\hat{\mathbf{h}}_j)$.

Table 3.8 *Examples from the SST2 test set, where the proposed TagHyperTreeLSTM predicts correctly while TreeLSTM without using syntactic tags fails. **P** indicates a predicted label by TreeLSTM and **G** indicates gold-standard label.*

Sentences	**G**	**P**
don't waste your money .	Negtive	Postive
a whole lot foul , freaky and funny .	Positive	Negtive
the episodic film makes valid points about the depersonalization of modern life .	Positive	Negtive
the whole damn thing is ripe for the jerry springer crowd .	Negtive	Postive
one scarcely needs the subtitles to enjoy this colorful action farce .	Postive	Negtive
like old myths and wonder tales spun afresh .	Positive	Negtive
but it pays a price for its intricate intellectual gamesmanship .	Negtive	Positive

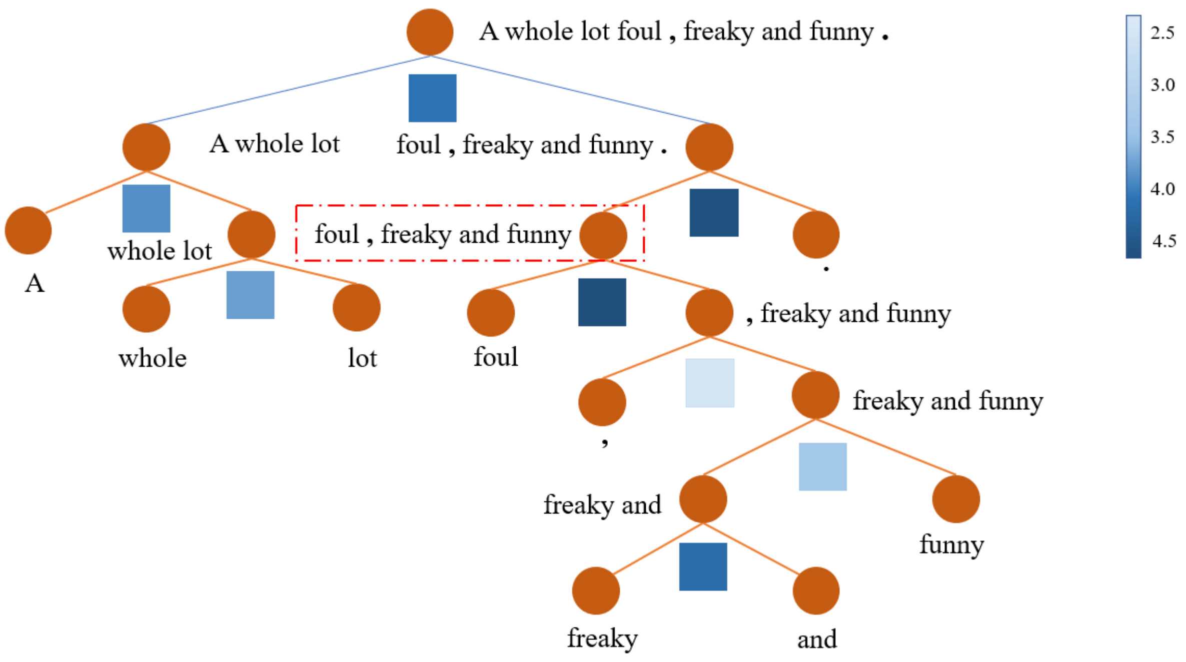


Fig. 3.5 A visualization of values of the 10th neuron in $\mathbf{W}(\hat{\mathbf{h}}_j)$ for sentence "a whole lot foul , freaky and funny .". The color of the square below non-leaf nodes indicates the value of the 10th neuron in $\mathbf{W}(\hat{\mathbf{h}}_j)$.

final sentence representation. Experimental results on six datasets in two NLP tasks have demonstrated the superiority and the generalization ability of the proposed model.

Chapter 4

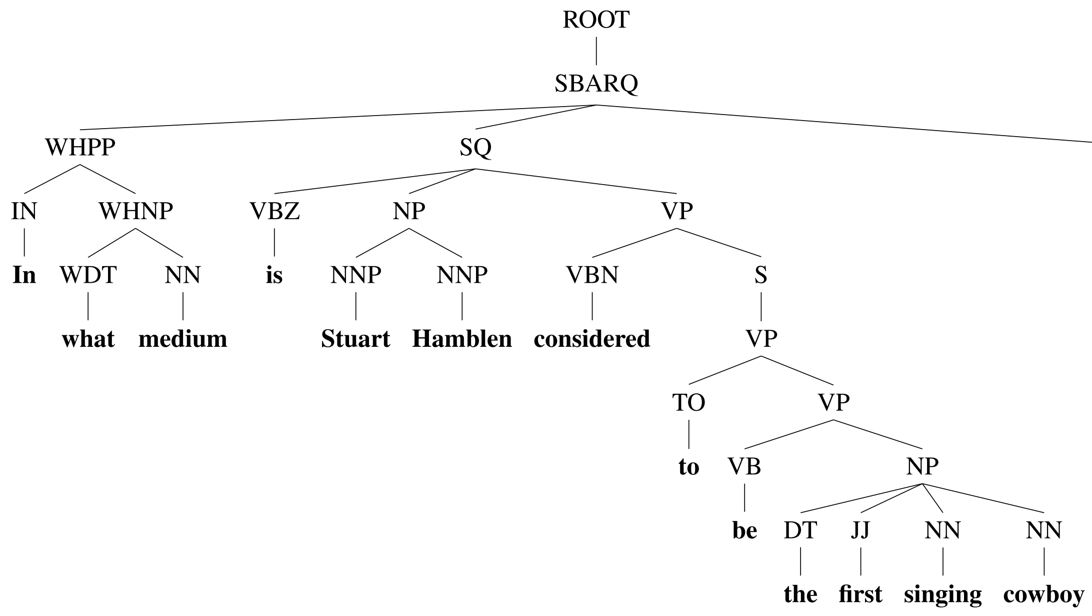
Tag-Enhanced Dynamic Compositional Neural Network over Arbitrary Tree Structure for Sentence Representation

In the previous chapter, we improved the expressive power of the standard TreeLSTM by performing dynamic composition using a tag-aware hypernetwork. However, existing dynamic compositional models are based on a binarized constituency tree which is different from the original constituency tree generated by a parser and cannot obtain the inherent structural information of a sentence effectively. This chapter extends the standard TreeLSTM with binarized constituency tree to a novel structure, i.e., ARTreeLSTM, with general constituency tree in which each non-leaf node can have any number of child nodes. And then a dynamic compositional model is proposed based on this ARTreeLSTM.

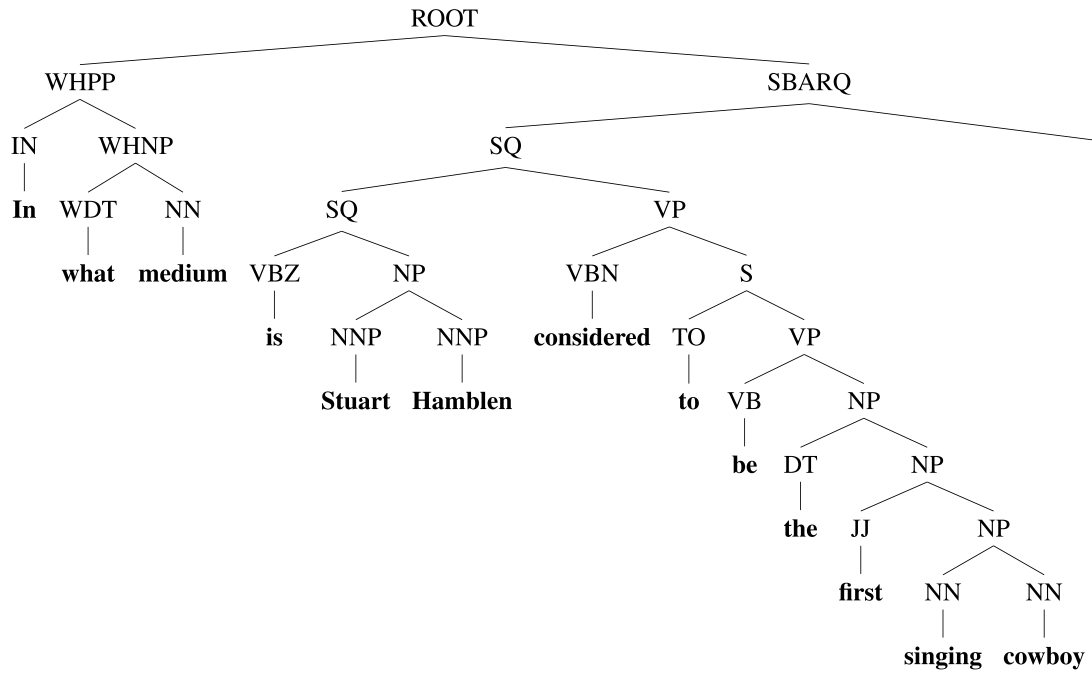
4.1 Introduction

Recursive neural networks (RecNNs) or tree-structured neural networks have made impressive progress for sentence representation on several downstream NLP tasks [143, 117, 138]. However, a major limitation is that all kinds of syntactic compositions share the same parameters in RecNN models, neglecting the fact that different compositions require different composition rules. For example, the *adjective-noun* composition (e.g., phrase "important meeting") is significantly different from the *adverb-adjective* composition (e.g., phrase "extremely important"). Thus, some dynamic compositional models have been proposed to model the diversity of different syntactic compositions [41, 27, 91, 138, 72, 46, 56, 101]. For example, Shen et al. [101] proposed a hypernetwork RecNN which employs Part-of-Speech (POS) tag information and semantic information of word/phrase jointly as inputs to generate parameters of different syntactic compositions dynamically, and achieves impressive performance on text classification and text semantic matching tasks.

In spite of the impressive performance, there are at least two limitations with previous dynamic compositional models. The first limitation is that existing dynamic compositional models are based on a binarized constituency tree which is different from the original constituency tree generated by a parser and cannot obtain the inherent structural information of a sentence effectively. Figure 4.1 shows a binarized constituency tree and the original constituency tree for a sentence from the TREC dataset [66]. It is clear that, different from the binarized constituency tree, the number of child nodes of an inner node of the original constituency tree is arbitrary. Furthermore, the original constituency tree is more shallow and so can represent a sentence in a more compact format than its corresponding binarized tree. In addition, recently proposed TreeNet [17] has also demonstrated the effectiveness of using the original constituency tree for sentence representation. However, similar to early RecNN models, TreeNet cannot capture the diversity of different syntactic compositions.



(a) Original constituency tree



(b) Binarized constituency tree

Fig. 4.1 Examples of original constituency tree and binarized constituency tree. The explanations of all tags in the figure is presented in Table 4.1.

Table 4.1 *Explanations of tags in a constituency tree.*

Tags	**Explanations**
DT	Determiner
IN	Preposition
JJ	Adjective
NN	Noun
NNP	Proper Noun
NP	Noun Phrases
PP	Prepositional Phrases
S	Simple Declarative Clause
SBARQ	Direct question introduced by a wh-word or a wh-phrase
SQ	Inverted yes/no question, or main clause of a wh-question
TO	to
VBD	Verb, past tense
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
VP	Verb Phrase
WDT	WH-Determiner
WHADJP	WH-Adjective Phrase
WHNP	WH-Noun Phrase
WHPP	WH-Prepositional Phrase
WRB	WH-Adverb

The second limitation is that previous models only use the tag embedding of the current phrase while composing the representation of the phrase, which is not reasonable in some cases. For example, in Figure 4.1 (a), although phrase *"Stuart/NNP Hamblen/NNP"* and phrase *"the/DT first/JJ singing/NN cowboy/NN"* both have the same NP tag, these two phrases have different internal structure because they vary greatly in size and the tags of their child nodes are quite different. Therefore, it is not reasonable to guide the composition of these two phrases solely based on the simple tag embedding of the NP tag. Thus, more complicated tag representations are needed to obtain structural information. Moreover, tag representations of child nodes are important for distinguishing different syntactic compositions.

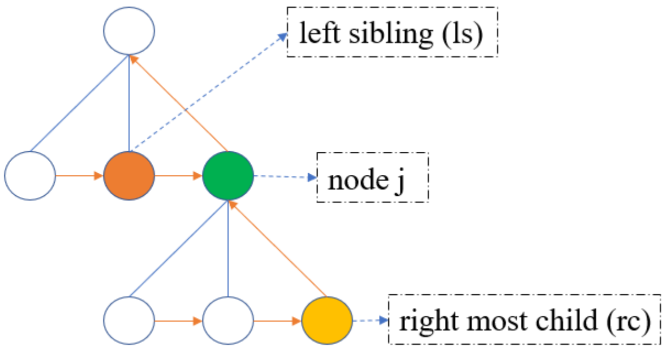
To alleviate the above two limitations, we firstly propose a variant of LSTM, ARTreeLSTM, to handle arbitrary tree structure, in which each node within a tree can learn from its left sibling and right child from left to right and bottom to top direction. Based on ARTreeLSTM, Tag-Enhanced Dynamic Compositional Neural Network (TE-DCNN) is proposed for sentence representation learning, which contains two ARTreeLSTMs, i.e., tag-level ARTreeLSTM and word-level ARTreeLSTM. The tag-level ARTreeLSTM accepts original tag embeddings at each node of a constituency tree and outputs the computed tag representations for the nodes, and then these tag representations will be used to control the gates of the word-level ARTreeLSTM to conduct dynamic semantic composition. For word-level ARTreeLSTM, a gate-memory encoder [17] is used to obtain the word representation of each leaf node in the constituency tree. For each non-leaf nodes, its representation is based on the states and tag representations of its left sibling and right child. The representation of the root node is used to represent the entire sentence. To the best of our knowledge, the proposed TE-DCNN is the first work that has the ability of both handling arbitrary tree structures such as the original constituency tree and capturing the richness of compositionality.

4.2 ARTreeLSTM

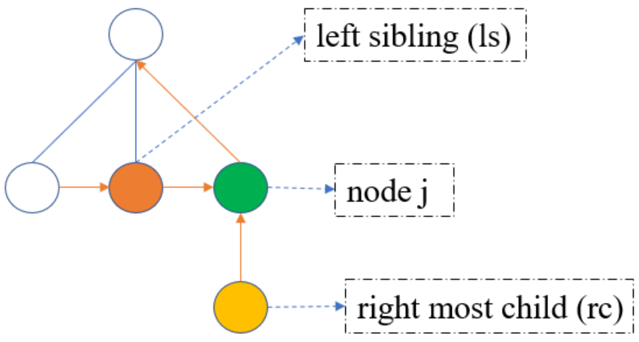
LSTM [44] is proposed to deal with the vanishing and exploding gradient problems of RNN [30], which can capture long-distance dependencies for sequential data due to its well-designed gate mechanism. Tai et al. [117] and Zhu et al. [160] applied LSTM unit into tree structures and achieved impressive performance for sentence representation. Previous dynamic compositional models assume that there are only two child nodes for non-leaf nodes in TreeLSTM, thus it can only handle a binarized constituency tree which is different from the original constituency tree generated by a standard parser tool, and cannot reflect the inherent structure of a sentence effectively. Inspired by TreeNet [17], in which each inner node can learn from its left sibling and right child, we extend the original TreeLSTM to ARTreeLSTM to handle the original constituency tree, in which the number of child nodes of non-leaf nodes is arbitrary. Instead of learning from left and right child nodes as in TreeLSTM, each node in ARTreeLSTM will learn from its left sibling and right most child. Child nodes with the same parent node are processed sequentially from left to right in a recurrent manner, thus the right most child can learn from all its siblings. Figure 4.2 shows the left sibling and right most child of a node j in two different cases. For each node j, the left sibling can be seen as its previous state and the right most child can be viewed as the representation of its descendants.

Figure 4.3 shows the architecture of ARTreeLSTM unit. For each node j in a constituency tree of a sentence, let $\mathbf{x}_j = [x_1, \ldots, x_{d_e}]^T$ be an input vector, $\mathbf{h}_j^{ls} = [h_1^{ls}, \ldots, h_d^{ls}]^T$ and $\mathbf{h}_j^{rc} = [h_1^{rc}, \ldots, h_d^{rc}]^T$ be the hidden states of left sibling and right most child at node j, respectively. $\mathbf{c}_j^{ls} = [c_1^{ls}, \ldots, c_d^{ls}]^T$ and $\mathbf{c}_j^{rc} = [c_1^{rc}, \ldots, c_d^{rc}]^T$ be the cell states of left sibling and right most child of node j, respectively.

Firstly, a new vector $\mathbf{z}_j$ is created by concatenating $\mathbf{x}_j, \mathbf{h}_j^{ls}, \mathbf{h}_j^{rc}$ where $\mathbf{z}_j = [x_1, \ldots, x_{d_e}, h_1^{ls}, \ldots, h_d^{ls}, h_1^{rc}, \ldots, h_d^{rc}]^T$. Then the hidden state of each node j in ARTreeLSTM is com-



(a) node *j* with multiple child nodes



(b) node *j* with only one child

Fig. 4.2 Left sibling and right most child of a node *j* in two different cases. If node *j* has multiple child nodes, then the last child is regarded as right most child. Otherwise, the only child is regarded as right most child.

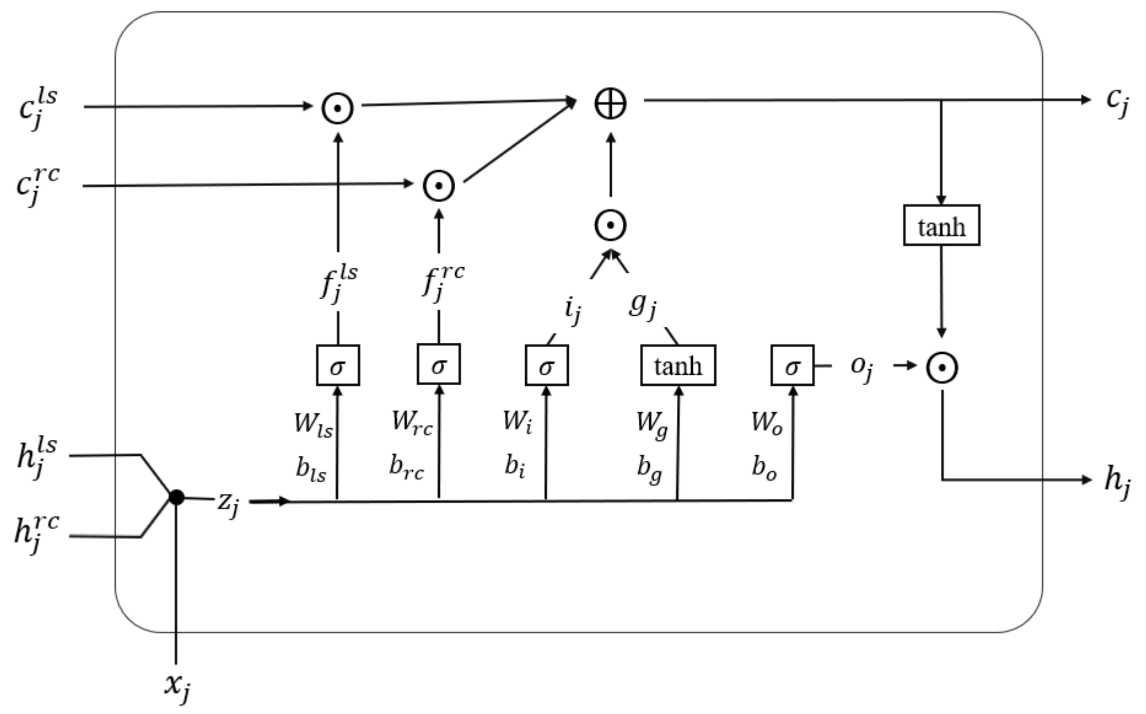


Fig. 4.3 Illustration of ARTreeLSTM architecture.

puted by the following equations:

$$\mathbf{i}_j = \sigma(\mathbf{W}_i \mathbf{z}_j + \mathbf{b}_i) \tag{4.1}$$

$$\mathbf{f}_j^{ls} = \sigma(\mathbf{W}_{ls} \mathbf{z}_j + \mathbf{b}_{ls}) \tag{4.2}$$

$$\mathbf{f}_j^{rc} = \sigma(\mathbf{W}_{rc} \mathbf{z}_j + \mathbf{b}_{rc}) \tag{4.3}$$

$$\mathbf{g}_j = \tanh(\mathbf{W}_g \mathbf{z}_j + \mathbf{b}_g) \tag{4.4}$$

$$\mathbf{o}_j = \sigma(\mathbf{W}_o \mathbf{z}_j + \mathbf{b}_o) \tag{4.5}$$

$$\mathbf{c}_j = \mathbf{f}_j^{ls} \odot \mathbf{c}_j^{ls} + \mathbf{f}_j^{rc} \odot \mathbf{c}_j^{rc} + \mathbf{i}_j \odot \mathbf{g}_j \tag{4.6}$$

$$\mathbf{h}_j = \mathbf{o}_j \odot \tanh(\mathbf{c}_j) \tag{4.7}$$

where $\mathbf{c}_j, \mathbf{h}_j \in \mathbb{R}^d$ refer to the memory cell and hidden state of node j. $\mathbf{i}_j, \mathbf{f}_j^{ls}, \mathbf{f}_j^{rc}, \mathbf{o}_j \in \mathbb{R}^d$ represent input gate, two forgot gates (left sibling and right most child), and output gate, respectively. $\mathbf{g}_j \in \mathbb{R}^d$ is the newly composed input for the memory cell. $\mathbf{W}_i, \mathbf{W}_{ls}, \mathbf{W}_{rc}, \mathbf{W}_g, \mathbf{W}_o \in \mathbb{R}^{d\times(2d+d_e)}$ and $\mathbf{b}_i, \mathbf{b}_{ls}, \mathbf{b}_{rc}, \mathbf{b}_g, \mathbf{b}_o \in \mathbb{R}^d$ are trainable parameters. tanh is the hyperbolic tangent, σ denotes the sigmoid function, and $\odot$ represents element-wise multiplication.

In particular, child nodes with the same parent node are processed sequentially from left to right. If node j is a leaf node (has no child nodes), a zero vector is employed to initialize $\mathbf{h}_j^{rc}$ and $\mathbf{c}_j^{rc}$. If node j is the first child node (has no left sibling), a zero vector is used to initialize $\mathbf{h}_j^{ls}$ and $\mathbf{c}_j^{ls}$.

For simplicity, we describe the computation of the hidden state of node j at a high level with Equation (4.8) to facilitate references later in this chapter, and the detailed computation refers to Equations (4.1-4.7).

$$\mathbf{h}_j = \mathbf{ARTreeLSTM}(\mathbf{x}_j, \mathbf{h}_j^{ls}, \mathbf{h}_j^{rc}, \mathbf{c}_j^{ls}, \mathbf{c}_j^{rc}) \tag{4.8}$$

All notations in Equation (4.8) follow Equations (4.1-4.7).

4.3 Model

In this section, a novel dynamic compositional neural architecture, named TE-DCNN (**T**ag-**E**nhanced **D**ynamic **C**ompositional **N**eural **N**etwork) is presented. The proposed TE-DCNN contains two separate ARTreeLSTMs: tag-level ARTreeLSTM and word-level ARTreeLSTM. The tag-level ARTreeLSTM is employed to guide the composition of word-level ARTreeLSTM which is responsible for constructing sentence representations. For a node j in the constituency parse tree, instead of using its own tag representation, tag representations of its left sibling and right most child are jointly used to control the gate functions of the word-level ARTreeLSTM to perform dynamic composition. Figure 4.4 illustrates the proposed model.

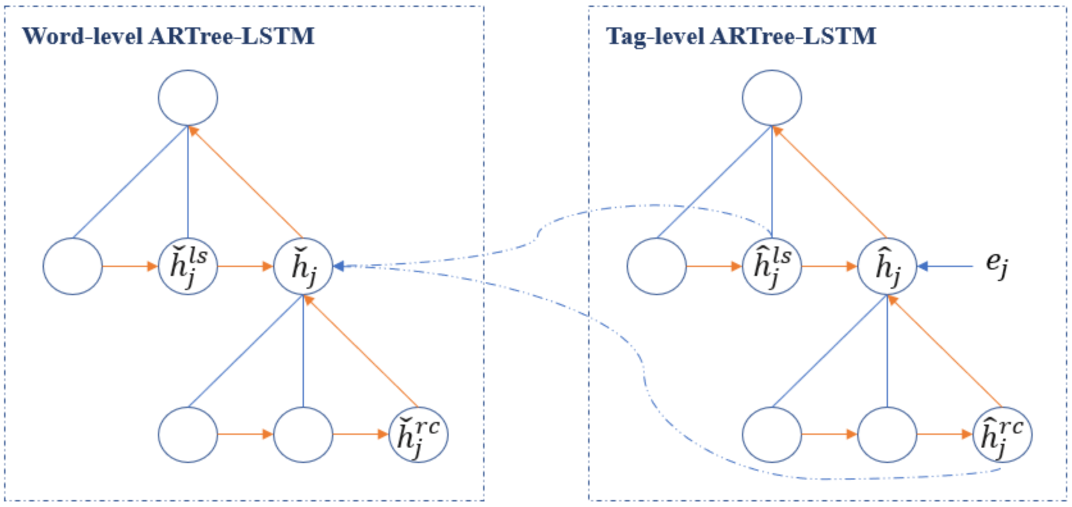


Fig. 4.4 An overview of TE-DCNN at node j. Tag-level ARTreeLSTM only accepts tag embeddings as inputs. $\hat{\mathbf{h}}_j$ denotes the hidden state at node j in the tag-level ARTreeLSTM, which is computed based on tag embeddings $\mathbf{e}_j$, the hidden states of its right most child $\hat{\mathbf{h}}_j^{rc}$, and the left sibling $\hat{\mathbf{h}}_j^{ls}$. Word-level ARTreeLSTM accepts word representations and tag representations generated by tag-level ARTreeLSTM as inputs. $\check{\mathbf{h}}_j$ denotes the hidden state at node j in the word-level ARTreeLSTM which is computed based on its right most child $\check{\mathbf{h}}_j^{rc}$, left sibling $\check{\mathbf{h}}_j^{ls}$, the corresponding tag presentations $\hat{\mathbf{h}}_j^{rc}$, and $\hat{\mathbf{h}}_j^{ls}$.

Formally, we denote sentence as a sequence of words $(w_1, w_2, ..., w_m)$ where m denotes the length of the sentence. Word embeddings are denoted as $(\mathbf{v}_1, \ldots \mathbf{v}_m)$ where $\mathbf{v}_i = [v_1, \ldots, v_{d_w}]^T, i \in [1, m]$. Tag embedding for the tag attached to each node j in the constituency tree is denoted by $\mathbf{e}_j = [e_1, \ldots, e_{d_t}]^T$. For each word w_t ($t \in [1, m]$) within a sentence, we use a gate-memory encoder [17] onto its word embedding $\mathbf{v}_t$ and tag embedding $\mathbf{e}_t$ to obtain a new word representation $\bar{\mathbf{h}}_t$ by the following equations:

$$\bar{\mathbf{i}}_t = \sigma(\mathbf{U}_{vi}\mathbf{v}_t + \mathbf{U}_{ei}\mathbf{e}_t + \mathbf{a}_i) \tag{4.9}$$

$$\bar{\mathbf{o}}_t = \sigma(\mathbf{U}_{vo}\mathbf{v}_t + \mathbf{U}_{eo}\mathbf{e}_t + \mathbf{a}_o) \tag{4.10}$$

$$\bar{\mathbf{g}}_t = \sigma(\mathbf{U}_{vg}\mathbf{v}_t + \mathbf{U}_{eg}\mathbf{e}_t + \mathbf{a}_g) \tag{4.11}$$

$$\bar{\mathbf{c}}_t = \bar{\mathbf{i}}_t \odot \bar{\mathbf{g}}_t \tag{4.12}$$

$$\bar{\mathbf{h}}_t = \bar{\mathbf{o}}_t \odot \tanh(\bar{\mathbf{c}}_t) \tag{4.13}$$

where $\bar{\mathbf{h}}_t, \bar{\mathbf{c}}_t \in \mathbb{R}^{d_h}$ represent the hidden state and memory cell of the gate-memory encoder. $\bar{\mathbf{i}}_t, \bar{\mathbf{o}}_t \in \mathbb{R}^{d_h}$ are two gates of the gate-memory encoder. $\mathbf{U}_{vi}, \mathbf{U}_{ei}, \mathbf{U}_{vo}, \mathbf{U}_{eo}, \mathbf{U}_{vg}, \mathbf{U}_{eg} \in \mathbb{R}^{d_h \times (d_w + d_t)}$ and $\mathbf{a}_i, \mathbf{a}_o, \mathbf{a}_g \in \mathbb{R}^{d_h}$ are trainable parameters. The rest of the notations follow that of the ARTreeLSTM above. The representation $\bar{\mathbf{h}}_t$ is used as the input of the word-level ARTreeLSTM.

Then, the final representation for the input sentence can be obtained in two steps:

First, the tag-level ARTreeLSTM which only accepts tag embeddings as inputs is employed to obtain the structure-aware tag representations (the right side of Figure 4.4). Tags are usually represented as low-dimensional tag embeddings in most previous works. Kim et al. [56] used a TreeLSTM on original tag embeddings to obtain the structure-aware tag representations which have proven to be more effective than simple tag embeddings. Similarly, we use a separate ARTreeLSTM to obtain the structure-aware tag representation

$\hat{\mathbf{h}}_j \in \mathbb{R}^{d_h}$ for each node j in a constituency tree in the following way:

$$\hat{\mathbf{h}}_j = \mathbf{ARTreeLSTM}(\mathbf{e}_j, \hat{\mathbf{h}}_j^{ls}, \hat{\mathbf{h}}_j^{rc}, \hat{\mathbf{c}}_j^{ls}, \hat{\mathbf{c}}_j^{rc}) \tag{4.14}$$

where **ARTreeLSTM** refers to Equation (4.8). $\mathbf{e}_j \in \mathbb{R}^{d_t}$ is tag embedding for the tag attached to each node j. $\hat{\mathbf{h}}_j^{ls}, \hat{\mathbf{h}}_j^{rc}, \hat{\mathbf{c}}_j^{ls}, \hat{\mathbf{c}}_j^{rc} \in \mathbb{R}^{d_h}$ represent the hidden states and memory cells of the left sibling and the right most child at the j-th node, respectively. The new tag representation $\hat{\mathbf{h}}_j$ will be used to control the gates of the word-level ARTreeLSTM to conduct dynamic composition.

Second, the word-level ARTreeLSTM (the left side of Figure 4.4) which accepts word representations as inputs is used to obtain the final sentence representation. Note that for each non-leaf nodes, its representation is computed based on the states and tag representations of its left sibling and right most child. Let $\check{\mathbf{h}}_j^{ls} = [\check{h}_1^{ls}, \ldots, \check{h}_{d_h}^{ls}]^T, \check{\mathbf{c}}_j^{ls} = [\check{c}_1^{ls}, \ldots, \check{c}_{d_h}^{ls}]^T, \check{\mathbf{h}}_j^{rc} = [\check{h}_1^{rc}, \ldots, \check{h}_{d_h}^{rc}]^T, \check{\mathbf{c}}_j^{rc} = [\check{c}_1^{rc}, \ldots, \check{c}_{d_h}^{rc}]^T$ be the hidden states and memory cells of left sibling and right most child of the j-th node in the word-level ARTreeLSTM, respectively. $\bar{\mathbf{h}}_j = [\bar{h}_1, \ldots, \bar{h}_{d_h}]^T$ is the input vector at the j-th node, which is a zero vector at non-leaf node and word representation vector computed by Equations (4.9-4.13) at leaf nodes. $\hat{\mathbf{h}}_j^{ls} = [\hat{h}_1^{ls}, \ldots, \hat{h}_{d_h}^{ls}]^T, \hat{\mathbf{h}}_j^{rc} = [\hat{h}_1^{rc}, \ldots, \hat{h}_{d_h}^{rc}]^T$ are the tag representations of the left sibling and right most child at node j, which are computed by Equation (4.14). Vectors $\mathbf{p}_j \in \mathbb{R}^{5d_h}$ and $\mathbf{q}_j \in \mathbb{R}^{3d_h}$ are created by concatenating $(\bar{\mathbf{h}}_j, \check{\mathbf{h}}_j^{ls}, \check{\mathbf{h}}_j^{rc}, \hat{\mathbf{h}}_j^{ls}, \hat{\mathbf{h}}_j^{ls})$ and $(\bar{\mathbf{h}}_j, \check{\mathbf{h}}_j^{ls}, \check{\mathbf{h}}_j^{rc})$, respectively. The hidden state of each node j in word-level ARTreeLSTM is defined by the following equation:

$$\check{\mathbf{i}}_j = \sigma(\mathbf{V}_i \mathbf{p}_j + \mathbf{r}_i) \tag{4.15}$$

$$\check{\mathbf{f}}_j^{ls} = \sigma(\mathbf{V}_{ls} \mathbf{p}_j + \mathbf{r}_{ls}) \tag{4.16}$$

$$\check{\mathbf{f}}_j^{rc} = \sigma(\mathbf{V}_{rc} \mathbf{p}_j + \mathbf{r}_{rc}) \tag{4.17}$$

$$\check{\mathbf{g}}_j = \tanh(\mathbf{V}_g \mathbf{q}_j + \mathbf{r}_g) \tag{4.18}$$

$$\check{\mathbf{o}}_j = \sigma(\mathbf{V}_o \mathbf{p}_j + \mathbf{r}_o) \tag{4.19}$$

$$\check{\mathbf{c}}_j = \check{\mathbf{f}}_j^{ls} \odot \check{\mathbf{c}}_j^{ls} + \check{\mathbf{f}}_j^{rc} \odot \check{\mathbf{c}}_j^{rc} + \check{\mathbf{i}}_j \odot \check{\mathbf{g}}_j \tag{4.20}$$

$$\check{\mathbf{h}}_j = \check{\mathbf{o}}_j \odot \tanh(\check{\mathbf{c}}_j) \tag{4.21}$$

where $\check{\mathbf{h}}_j, \check{\mathbf{c}}_j \in \mathbb{R}^{d_h}$ refer to the hidden state and memory cell of node j in the word-level ARTreeLSTM. $\mathbf{V}_i, \mathbf{V}_{ls}, \mathbf{V}_{rc}, \mathbf{V}_o \in \mathbb{R}^{d_h \times 5d_h}, \mathbf{V}_g \in \mathbb{R}^{d_h \times 3d_h}$ and $\mathbf{r}_i, \mathbf{r}_{ls}, \mathbf{r}_{rc}, \mathbf{r}_g, \mathbf{r}_o \in \mathbb{R}^{d_h}$ are learned parameters. The remaining notation follows Equations (4.1-4.7). Finally, the hidden state of the root node $\check{\mathbf{h}}^{\mathbf{root}} \in \mathbb{R}^{d_h}$ is used as the representation for the given sentence.

4.4 Applications of TE-DCNN

This section describes the applications of TE-DCNN for two typical NLP tasks.

Text classification. Given a sentence s and a pre-defined class set $\mathscr{Y}$, text classification is to predict a label $\hat{y} \in \mathscr{Y}$ for s. A softmax classifier is applied directly on the sentence representation $\check{\mathbf{h}}^{\mathbf{root}} \in \mathbb{R}^{d_h}$ in this work. The final predicted probability distribution of class y given sentence s is defined as follows:

$$p(y|s) = \mathbf{softmax}(\mathbf{W}_c \check{\mathbf{h}}^{\mathbf{root}} + \mathbf{b_c}) \tag{4.22}$$

where $\mathbf{W}_c \in \mathbb{R}^{d_c \times d_h}, \mathbf{b_c} \in \mathbb{R}_c^d$ are trainable parameters, d_c is the size of class set $\mathscr{Y}$.

Text matching. Text matching is to predict the relationship $\hat{l} \in \mathscr{L}$ between a given sentence pair $s1$ and $s2$ from a pre-defined label set $\mathscr{L}$. Firstly, the same TE-DCNN is used to encode $s1$ and $s2$ into two sentence representation vectors $\check{\mathbf{h}}_{\mathbf{s1}}^{\mathbf{root}}, \check{\mathbf{h}}_{\mathbf{s2}}^{\mathbf{root}} \in \mathbb{R}^{d_h}$. Next, some matching heuristics [82] are used to combine the two sentence vectors together in the following way:

$$\mathbf{h}_{st} = [\check{\mathbf{h}}_{s1}^{root}, \check{\mathbf{h}}_{s2}^{root}, \check{\mathbf{h}}_{s1}^{root} \odot \check{\mathbf{h}}_{s2}^{root}, |\check{\mathbf{h}}_{s1}^{root} - \check{\mathbf{h}}_{s2}^{root}|] \tag{4.23}$$

Then, a single layer network is applied on the above concatenated vector $\mathbf{h}_{st}$:

$$\mathbf{h}_m = \mathbf{Relu}(\mathbf{W}_m \mathbf{h}_{st} + \mathbf{b}_m) \tag{4.24}$$

where $\mathbf{h}_m \in \mathbb{R}^{d_m}$ is the intermediate feature vector for the following classifier. $\mathbf{W}_m \in \mathbb{R}^{d_m \times 4d_h}, \mathbf{b}_m \in \mathbb{R}_m^d$ are trainable parameters. Finally, the probability distribution of label l given sentence pair $s1$ and $s2$ is obtained using a softmax classifier:

$$p(l|(s1,s2)) = \mathbf{softmax}(\mathbf{W}_l \mathbf{h}_m + \mathbf{b}_l) \tag{4.25}$$

where $\mathbf{W}_l \in \mathbb{R}^{d_l \times d_m}, \mathbf{b}_l \in \mathbb{R}^{d_l}$ are again trainable parameters. The parameters of the model are learned to minimise the cross-entropy of the distributions between predicted and true label.

4.5 Experiments

4.5.1 Datasets

The proposed model are evaluated on three benchmarks for text classification (MR, SUBJ, TREC) and one dataset (SICK) for text semantic matching:

- MR: The movie reviews with positive or negative label [85].
- SUBJ: Sentences grouped as being either subjective or objective [84].
- TREC: A dataset which groups questions into six different question types [66].

- SICK: A textual entailment dataset with three classes (entailment, neutral, contradiction) [76].

Tabel 4.2 shows the detailed statistics about the above four datasets.

Table 4.2 *Statistics of four benchmark datasets for two tasks. Train, Dev and Test are the size of train, validation and test dataset, respectively. CV means 10-fold cross validation is used.* L_{avg} *is the average number of words in sentences.* $|V|$ *is the size of vocabulary.* $|T|$ *is the number of tags. Class is the size of label/class set.*

Dataset	**Train**	**Dev**	**Test**	$\mathbf{L_{avg}}$	**|V|**	**|T|**	**Class**
MR	10662	-	CV	22	19K	73	2
SUBJ	10000	-	CV	21	21K	72	2
TREC	75952	-	500	10	10K	67	6
SICK	4500	500	4927	10	2K	41	3

4.5.2 Experimental Setup

In all experiments, sentences in datasets are tokenized and parsed by Stanford PCFG parser[1] [60]. Word embeddings are initialized with the 300-dimensional GloVe word vectors [89], and out-of-vocabulary words are randomly sampled from the uniform distribution $[-0.05, 0.05]$. Tag embeddings are initialized by the normal distribution $[0, 1]$. Tag embeddings are fine-tuned during training procedure while word embeddings are fixed. Hidden size for ARTreeLSTMs is fine-tuned from $[100, 200, 300]$. The dimension of tag embedding is selected from $[20, 25]$. Batch size is selected from $[32, 64]$. Weights of the model are trained by minimizing the cross-entropy of the training dataset by Adam optimizer [58] and learning rate is fine-tuned in the range of $[1e^{-2}, 1e^{-3}]$. The accuracy

[1] https://nlp.stanford.edu/software/lex-parser.shtml

metric is used in this work to measure the performance of the proposed and all comparison models on four datasets and two tasks, and the results of all comparison models come from respective papers.

4.5.3 Results

Text Classification. The proposed TE-DCNN is compared with two types of models. The first is RecNN based models which are based on tree structure, and the other is non-RecNN based models. Tables 4.3 and 4.4 show the test accuracy of the proposed TE-DCNN and comparison models on each dataset.

Comparison with RecNN based Models. We classified RecNN based models into two categories: Non-dynamic models and dynamic compositional models. Non-dynamic models are RecNN based models that share the same parameters for all kinds of syntactic compositions, while dynamic compositional models can distinguish different syntactic compositions. Table 4.3 shows test accuracies of the proposed TE-DCNN and previous RecNN based models. **Firstly**, compared with all previous RecNN based models, TE-DCNN achieves superior or competitive performance on all three text classification datasets. Moreover, it sets a new state of the art among RecNN based models on two out of three datasets - MR and TREC with accuracies of 84.1% and 97%, respectively. Specifically, the proposed TE-DCNN outperforms the best non-dynamic model and dynamic compositional model on MR dataset by a margin of 0.5% and 0.3%; and on TREC dataset by a margin of 0.9% and 0.8%, respectively. **Secondly**, the proposed model is superior to dynamic compositional models ignoring tag information (i.e., AdaHT-LSTM, iTLSTM, DC-RecNN and DC-TreeLSTM) on all three datasets MR, SUBJ and TREC with 2.4%, 0.6% and 3.2% improvements, respectively. The consistency suggests these improvements are due to the usage of tag information which is helpful for composing sentence representation. **Thirdly**, compared with five dynamic models leveraging tag information (i.e., TE-RNN, TE-LSTM,

Table 4.3 *Accuracies of previous RecNN based models and the proposed TE-DCNN on three text classification datasets.*

Model	**MR**	**SUBJ**	**TREC**
Non-Dynamic Models			
RecNN [110]	76.4	91.8	90.2
TreeLSTM [117]	81.2	93.2	93.6
BiTreeLSTM [128]	-	-	94.8
TreeNet [17]	83.6	<u>95.9</u>	96.1
Dynamic Compositional Models			
AdaHT-LSTM [71]	81.9	94.1	-
iTLSTM [69]	82.5	94.5	-
DC-RecNN [72]	80.2	93.5	91.2
DC-TreeLSTM [72]	81.7	93.7	93.8
TE-RNN [91]	77.9	-	-
TE-LSTM [46]	82.2	-	-
TG-HRecNN [101]	80.9	93.7	93.6
TG-HTreeLSTM [101]	82.6	94.9	95.8
SATA TreeLSTM[56]	83.8	95.4	96.2
TE-DCNN (proposed)	<u>**84.1**</u>	**95.1**	<u>**97.0**</u>

TG-HRecNN, TG-HTreeLSTM and SATA Tree-LSTM), TE-DCNN outperforms these models on MR and TREC datasets by a margin of 0.3% and 0.8%, respectively, and achieves competitive performance with the state-of-the-art model SATA Tree-LSTM on SUBJ dataset.

Table 4.4 *Accuracies of Non-RecNN based models and the proposed TE-DCNN on three text classification datasets. The symbol * indicates models which are pre-trained with large external corpora.*

Model	**MR**	**SUBJ**	**TREC**
CNN [57]	81.5	93.4	93.6
BLSTM-2DCNN [157]	82.3	94.0	96.1
byte-mLSTM* [93]	<u>86.9</u>	94.6	-
BCN + Char + CoVe* [77]	-	-	95.8
DARLM [158]	83.2	94.1	96.0
3W-CNN [156]	82.3	93.5	-
WSAN [48]	83.2	94.6	95.0
TE-DCNN (proposed)	**84.1**	**<u>95.1</u>**	**<u>97.0</u>**

Comparison with Non-RecNN based Models. Table 4.4 shows a comparison with some non-RecNN based models on three text classification datasets. We observe that the proposed TE-DCNN has consistently strong performance on three datasets, and it outperforms all comparison models in this group on SUBJ and TREC datasets by a margin of 0.5% and 0.9%, respectively. Although byte-mLSTM achieves the state-of-the-art performance on MR dataset with an accuracy of 86.9%, better than TE-DCNN with an accuracy of 84.1%, it is pre-trained with large external corpora while TE-DCNN do not perform any pre-train procedure.

Text Matching. To evaluate the proposed TE-DCNN on other NLP tasks, we also conducted an experiment on text semantic matching task. Different from text classification

Table 4.5 *Accuracies of previous RecNN based models and the proposed TE-DCNN on the SICK dataset.*

Model	**SICK**
Non-Dynamic Models	
RecNN[110]	74.9
MV-RNN [109]	75.5
RNTN [111]	76.9
TreeLSTM [117]	77.5
Dynamic Compositional Models	
DC-RecNN [72]	80.2
DC-TreeLSTM [72]	82.3
TG-HRecNN [101]	77.5
TG-HTreeLSTM [101]	83.3
TE-DCNN (proposed)	**83.4**

which works by classifying one sentence at a time, text semantic matching works by comparing two sentences at a time. Therefore, in a text classification dataset, a sample is a single sentence whereas in a text semantic matching dataset, a sample is a pair of sentences.Text semantic matching experiments were conducted on the SICK [76] dataset, and the experimental results are shown in Table 4.5. The results of TreeNet and SATA TreeLSTM come from our implementation while results of other comparison models come from respective papers. We can see that TE-DCNN has again demonstrated its superior performance compared against the previous RecNN based models. Specifically, TE-DCNN outperforms all compared non-dynamic models by a margin of 1.2% and is slightly better than previous dynamic compositional models with 0.1% improvements. This comparison shows that TE-DCNN is also effective in text sematic matching task, and has generalization ability in different NLP tasks.

4.5.4 Ablation Study

This section presents an ablation study on key modules of the proposed TE-DCNN to explore their effectiveness. An ablation study refers to removing some components or modules of the model and seeing how that affects performance. We focus on two modules, the ARTreeLSTM and tag representations. TREC dataset is used in this experiment, and the target module is replaced with other candidates while keeping the other settings fixed. In the first case, the ARTreeLSTM is replaced with a basic TreeLSTM. In the second case, we do not employ any tag information in the model.

The experimental results are shown in figure 4.5. As the chart shows, TE-DCNN outperforms the other two options we considered. Specially, if we do not use any tag information in TE-DCNN, the accuracy of the model drops to 95.6%, with 1.4% performance degradation. A possible reason for this performance degradation is that the model cannot distinguish different syntactic compositions while neglecting tag information, because tag representations are used to control the gates of the word-level ARTreeLSTM to conduct dynamic composition in TE-DCNN. If we replace ARTreeLSTM with a basic TreeLSTM in TE-DCNN, the accuracy of the model drops to 96.1%, with 0.9% performance degradation. Possible reason for this performance degradation is because basic TreeLSTM can only handle a binarized constituency tree which is different from the original constituency tree. In contrast, ARTreeLSTM can handle the original constituency tree of a sentence, thus can capture the inherent structural information of a sentence effectively and have better expressive power than a basic TreeLSTM.

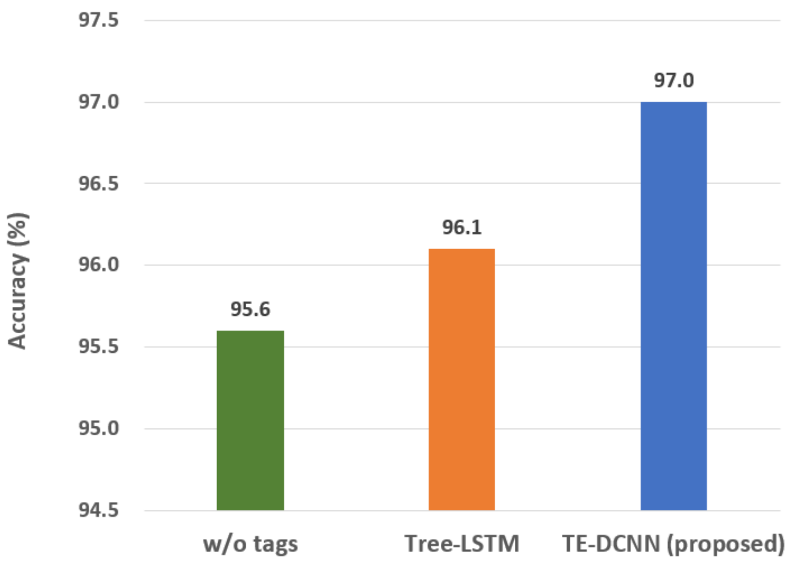


Fig. 4.5 An ablation study on key modules of TE-DCNN. Test accuracies on TREC dataset is reported. TreeLSTM: TreeLSTM is used instead of ARTreeLSTM in TE-DCNN. w/o tags: Tag information is not used.

4.5.5 Error Analysis

In this section, we analyze the predictions of the proposed TE-DCNN model and another two state-of-the-art models, TreeNet [17] and SATA Tree-LSTM [56]. SATA TreeLSTM and our model are dynamic compositional models while TreeNet is a non-dynamic model. Table 4.6 gives a breakdown of accuracy for classes on test sets of TREC and SICK datasets. We observe that, for TREC dataset, the proposed TE-DCNN matches the other two models on all classes. For SICK dataset, most of our gains stem from Neutral, while most losses come from Entailment pairs. Figure 4.6 presents the distribution of errors of our TE-DCNN and other two models on SICK dataset. There are six types of error in total. We can see that the most frequent error type of the proposed model is E$\rightarrow$N which means our model tend to classify an Entailment sentence pair Neutral. While the most frequent type of error of TreeNet and SATA TreeLSTM is N$\rightarrow$E. For all the three models, the least frequent error type is E$\rightarrow$C. Moreover, in most cases, dynamic compositional models such as SATA TreeLSTM and the propsoed TE-DCNN have smaller error rate than non-dynamic model TreeNet except N$\rightarrow$C.

Table 4.7 shows some example wins and losses of the proposed TE-DCNN compared to other models on SICK dataset. Examples 1 and 2 are cases where the proposed TE-DCNN is correct while both TreeNet and SATA TreeLSTM are incorrect. In these two examples, both sentences contain phrases that are either the same or highly lexically related (e.g., “a dog", “toddler/baby" and “a toy/a ball"). Our TE-DCNN correctly favors Neutral in these cases, while TreeNet and SATA TreeLSTM prefer to Entailment. Due to this characteristic of TE-DCNN, it fails in Examples 3 and 4 while TreeNet and SATA TreeLSTM succeed. Examples 5 and 6 are cases where the proposed TE-DCNN and SATA TreeLSTM are correct and TreeNet is incorrect. In these two examples, the key point to predict correctly is to conclude that “a white and brown dog/no white and brown dog" and “no girl/one girl" are contradictions. A possible reason for the success of TE-DCNN and SATA TreeLSTM

Table 4.6 *Breakdown of accuracy with respect to classes on TREC and SICK test sets. TE-DCNN is the proposed model while TreeNet [17] and SATA TreeLSTM [56] are two previously developed models with state-of-the-art performance (see Tables 4.3 and 4.5).*

Datasets	**Class**	**TreeNet**	**SATA TreeLSTM**	**TE-DCNN**
TREC	NUM	94.7	95.6	**96.5**
	HUM	98.5	98.5	98.5
	ENTY	89.4	**93.6**	**93.6**
	LOC	95.1	96.3	**97.5**
	DESC	96.4	97.8	**98.6**
	ABBR	88.9	88.9	**98.6**
	Overall	96.1	96.2	**97.0**
SICK	Entailment	84	**84.9**	77.9
	Neutral	81.8	82.2	**86.2**
	Contradiction	78.4	**85.3**	83.3
	Overall	82.1	83.3	**83.4**

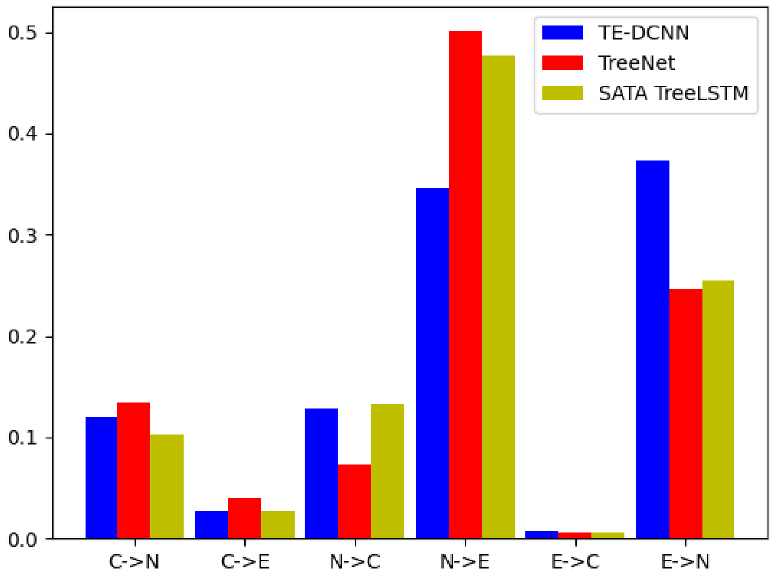


Fig. 4.6 Distribution of errors on SICK dataset. C, N and E refer to Contradiction, Neutral and Entailment, respectively. C→N means the true label is Contradiction while the prediction is Neutral. and so on.

Table 4.7 *Example wins and losses on SICK dataset of the proposed TE-DCNN compared to two state-of-the-art models TreeNet [17] and SATA TreeLSTM [56]. E, C and N refer to Entailment, Contradiction and Neutral, respectively.*

ID	**Text pair**	**TreeNet**	**SATA TreeLSTM**	**TE-DCNN**	**Gold**
1	a dog is licking a toddler a dog is licking a baby	E	E	N	N
2	a dog is running towards a ball a dog is running through a field and is chasing a toy	E	E	N	N
3	a deer is jumping over a fence a deer is jumping over the enclosure	E	E	N	E
4	a man is playing soccer a man is playing a game with a ball	E	E	N	E
5	a white and brown dog is walking through the water with difficulty there is no white and brown dog pacing with through the water difficulty	N	C	C	C
6	there is no girl jumping into the car one girl is jumping on the car	N	C	C	C

in these two cases is that these two models conduct dynamic composition with the aid of syntactic tags, so they can find the differences between above two phrase pairs easily.

4.6 Summary

This chapter presented a novel dynamic compositional model based on tree structure for sentence representation. A newly introduced ARTreeLSTM is employed to handle the original constituency tree firstly, in which an inner node can have arbitrary child nodes. Then a tag-level ARTreeLSTM and a word-level ARTreeLSTM are jointly used for sentence representation. Experimental results on four datasets and two NLP tasks have shown the superiority and the generalization ability of the proposed model.

Chapter 5

Multi-Level Compare-Aggregate Model for Text Matching

As aforementioned, this thesis aims to improve text matching in two perspectives: (1) text representation: to generate informative text representation, and (2) text interaction: to capture more interactive features between two texts. In Chapter 3 and Chapter 4, we focused on the first perspective, i.e., text representation, and proposed two new text representation models. In this chapter, we focus on another perspective, i.e., text interaction. Specially, to extract more matching information, this chapter propose to perform matching at multiple levels of granularity.

5.1 Introduction

Text matching is important for many NLP tasks and has attracted wide attention. Recent studies have achieved very promising results under the matching-aggregation or compare-aggregate framework, where smaller units (e.g., words) in two texts are matched firstly, and then the matching results of these small units are aggregated into a fixed-size vector for final matching decision. Although matching-aggregation based models have made impressive progress in text matching, there is still room for further improvement. Previous works mainly focus on matching at word level [135, 96, 87, 15], without considering matching at other levels such as word-by-phrase, word-by-sentence matching and so on. For example, *"UN"* stands for *"United Nations"* and *"007"* stands for *"James Bond"*. Under these circumstances, conducting matching at phrase level would be helpful for making matching decision.

In this chapter, we propose a multi-level compare-aggregate model (MLCA), which matches each word in one text against the other text at three different levels, word level (word-by-word matching), phrase level (word-by-phrase matching) and sentence level (word-by-sentence matching). Then these three types of matching results are aggregated for making final matching decision. The contributions of this work include:

- A new multi-level compare-aggregate (MLCA) model is proposed to improve the performance of original compare-aggregate model by incorporating matching information at three levels.
- The model is evaluated on two text matching tasks and experimental results show that the model outperforms baselines.

5.2 The MLCA Model

Figure 5.1 shows the overall framework of the proposed MLCA model, which has six layers. The first layer is the input layer, which consists of two matching texts. The second layer is the word representation layer, which is used to convert each word in either of the two texts into a semantic vector. The third layer is the contextual encoding layer, implemented by BiLSTM to encode contextual information into each word representation. The fourth layer is the matching layer, which is the core part of the model. We conduct matching at two directions and three levels (details in Section 5.2.3). The fifth layer is the aggregation layer, in which all the matching results are aggregated into a fixed-size vector in each direction by another BiLSTM. And the last layer is the prediction layer – a multilayer perceptron (MLP) classifier, for decision making.

5.2.1 Word Representation Layer

The purpose of this layer is to convert each word in texts P and Q into a d-dimensional representation vector, denoted as $\mathbf{p}_i \in \mathbb{R}^d$ $(i \in [1,m])$ and $\mathbf{q}_j \in \mathbb{R}^d$ $(j \in [1,n])$, which consists of a word-level embedding and a character-level embedding. The word-level embedding is obtained from a pre-trained word vectors. The character-level embedding is learned using a BiLSTM which takes as inputs characters within a word.

5.2.2 Contextual Encoding Layer

The goal of this layer is to encode the contextual information into each word representation. The new contextualized representations $\overline{\mathbf{p}}_1, \ldots, \overline{\mathbf{p}}_m$ and $\overline{\mathbf{q}}_1, \ldots, \overline{\mathbf{q}}_n$ for words in texts P and Q are generated using BiLSTM. A BiLSTM is composed of two LSTMs [44], one for capturing information from the first timestep to the last timestep and the other for capturing information from the reverse direction. Outputs of two LSTMs are concatenated to obtain a new representation. The following equations describe this computation ($i \in [1,m]$):

$$\overrightarrow{\mathbf{h}}_i = \mathbf{LSTM}(\overrightarrow{\mathbf{h}}_{i-1}, \mathbf{p}_i) \tag{5.1}$$

$$\overleftarrow{\mathbf{h}}_i = \mathbf{LSTM}(\overleftarrow{\mathbf{h}}_{i-1}, \mathbf{p}_i) \tag{5.2}$$

$$\overline{\mathbf{p}}_i = [\overrightarrow{\mathbf{h}}_i; \overleftarrow{\mathbf{h}}_i] \tag{5.3}$$

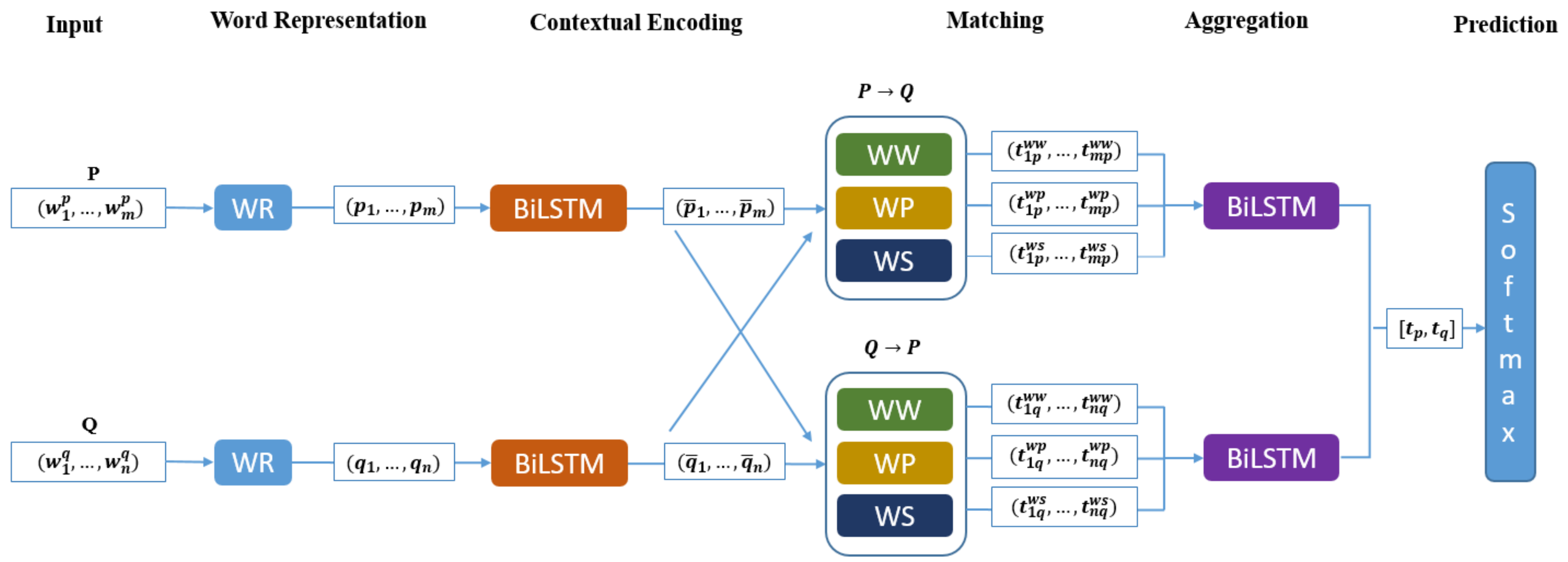

Input
Word Representation
Contextual Encoding
Matching
Aggregation
Prediction
P
$(w_1^p, \ldots, w_m^p)$
WR
$(p_1, \ldots, p_m)$
BiLSTM
$(\bar{p}_1, \ldots, \bar{p}_m)$
$P \to Q$
WW
WP
WS
$(t_{1p}^{ww}, \ldots, t_{mp}^{ww})$
$(t_{1p}^{wp}, \ldots, t_{mp}^{wp})$
$(t_{1p}^{ws}, \ldots, t_{mp}^{ws})$
BiLSTM
Q
$(w_1^q, \ldots, w_n^q)$
WR
$(q_1, \ldots, q_n)$
BiLSTM
$(\bar{q}_1, \ldots, \bar{q}_n)$
$Q \to P$
WW
WP
WS
$(t_{1q}^{ww}, \ldots, t_{nq}^{ww})$
$(t_{1q}^{wp}, \ldots, t_{nq}^{wp})$
$(t_{1q}^{ws}, \ldots, t_{nq}^{ws})$
BiLSTM
$[t_p, t_q]$
Softmax

Fig. 5.1 The overall framework of multi-level compare-aggregate (MLCA) model. *WR* means word representation layer. $P\to Q$ means matching text P against text Q, $Q\to P$ means matching text Q against text P. *WW*, *WP* and *WS* indicate word-by-word matching, word-by-phrase matching and word-by-sentence matching, respectively.

where $\overrightarrow{\mathbf{h}}_{i-1}$ and $\overleftarrow{\mathbf{h}}_{i-1}$ are the hidden states of LSTM at $(i-1)$-th timestep from two directions, $\mathbf{p}_i \in \mathbb{R}^l$ is the input of LSTM at i-th timestep, $\overline{\mathbf{p}}_i \in \mathbb{R}^l$, l is the dimension of the word embedding, which is equals to the hidden size of the BiLSTM. Meanwhile, we use another BiLSTM to encode each $\mathbf{q}_j$ to a new representation $\overline{\mathbf{q}}_j$.

5.2.3 Matching Layer

In this layer, each word in one text is matched against to the other text. The matching process is conducted in two directions: match P against Q, and match Q against P, respectively. Furthermore, we perform three types of matching: word-by-word (*WW*) matching, word-by-phrase (*WP*) matching and word-by-sentence (*WS*) matching, respectively. For simplicity, we only introduce the matching process in one direction (i.e., P against Q as an example). The reverse direction is performed in the same manner.

Word-by-Word Matching (WW). For word level matching, the attention weights are firstly computed as the similarity between words in two texts ($i \in [1,m], j \in [1,n]$):

$$\alpha_{ij} = (\overline{\mathbf{p}}_i)^T \cdot \overline{\mathbf{q}}_j \tag{5.4}$$

where α_{ij} is the similarity between the i-th word $\overline{\mathbf{p}}_i$ of text P and j-th word $\mathbf{q}_j$ of text Q. Then, for each word in text P, its relevant semantics in text Q are computed based on attention matrix α_{ij} as follows ($i \in [1,m]$):

$$\widetilde{\mathbf{p}}_i = \sum_{j=1}^{n} \frac{\exp(\alpha_{ij})}{\sum_{k=1}^{n} \exp(\alpha_{ik})} \overline{\mathbf{q}}_j \tag{5.5}$$

where $\widetilde{\mathbf{p}}_i$ denotes the contents in $\{\overline{\mathbf{q}}_j\}_{j=1}^{n}$ related to $\overline{\mathbf{p}}_i$. Then, a vector matching function is used on each pair of $<\widetilde{\mathbf{p}}_i, \overline{\mathbf{p}}_i>$ to obtain the word level matching results$\{\mathbf{t}_i^{ww}\}_{i=1}^{m}$ between two texts as follows ($i \in [1,m]$):

$$\mathbf{t}_{ip}^{ww} = f(\widetilde{\mathbf{p}}_i, \overline{\mathbf{p}}_i) \tag{5.6}$$

where f is the vector matching function.

Word-by-Phrase Matching (WP). In order to obtain phrase level representations from text Q, we use a convolutional layer to compose n-grams. A convolution filter slide over the text like a sliding window. For each window of k words in Q, we then generate the representation $\overline{\mathbf{g}}_j \in \mathbb{R}^l$ of phrase $\mathbf{q}_{j-k+1}, \ldots, \mathbf{q}_j$ by the following equation ($j \in [1, n-k+1]$):

$$\overline{\mathbf{g}}_j = \tanh(\mathbf{W}_g \mathbf{c}_j + \mathbf{b}_g) \tag{5.7}$$

where $\mathbf{c}_j \in \mathbb{R}^{kl}$ is the concatenated embeddings of words $\mathbf{q}_{j-k+1}, \ldots, \mathbf{q}_j$. $\mathbf{W}_g \in \mathbb{R}^{l \times kl}$ and $\mathbf{b}_g \in \mathbb{R}^l$ are the learnable parameters. For word $\overline{\mathbf{p}}_i$ in text P and phrase $\overline{\mathbf{g}}_j$ in text Q, we use the above Equations (5.4-5.5) to obtain $\check{\mathbf{p}}_i$ which represents the contents in $\{\overline{\mathbf{g}}_j\}_{j=1}^{n-k+1}$related to $\overline{\mathbf{p}}_i$. Then the word-by-phrase level matching result is obtained by the following equation ($i \in [1, m]$):

$$\mathbf{t}_{ip}^{wp} = f(\check{\mathbf{p}}_i, \overline{\mathbf{p}}_i) \tag{5.8}$$

where f is the same vector matching function as Equation (5.6).

Word-by-Sentence Matching (WS). Words in a sentence have different importance for the semantic composition. Therefore, this work uses a self-attention mechanism to assign an importance weight to each word in a sentence as following equations ($j \in [1, n]$):

$$\mathbf{u}_j = \tanh(\mathbf{W}_q \overline{\mathbf{q}}_j + \mathbf{b}_q) \tag{5.9}$$

$$\alpha_j = \frac{\exp(\mathbf{u}_j)}{\sum_{k=1}^{n} \exp(\mathbf{u}_k)} \tag{5.10}$$

where $\mathbf{W}_q \in \mathbb{R}^{l \times l}$ and $\mathbf{b}_q \in \mathbb{R}^l$ are the learnable parameters. α_j denotes the importance weight for the j-th word in text Q. Then, we compute the weighted sum of all words to form the sentence representation as follows ($j \in [1, n]$):

$$\mathbf{s}_q = \sum_{j=1}^{n} \alpha_j \overline{\mathbf{q}}_j \tag{5.11}$$

where $\mathbf{s}_q \in \mathbb{R}^l$ is the representation vector of text Q. Then the matching results of each word $\overline{\mathbf{p}}$ in text P against the sentence representation $\mathbf{s}_q$ of text Q is obtained using the following equation $(i \in [1,m])$:

$$\mathbf{t}_{ip}^{ws} = f(\overline{\mathbf{p}}_i, \mathbf{s}_q) \tag{5.12}$$

where f is the same vector matching function as Equation (5.6) and (5.8).

The same procedure is used to obtain these three levels of matching results while matching Q against P. We use $\mathbf{t}_{jq}^{ws}$, $\mathbf{t}_{jq}^{wp}$ and $\mathbf{t}_{jq}^{ws}$ $(j \in [1,n])$ to denote the matching results at word level, phrase level and sentence level, respectively.

5.2.4 Aggregation and Prediction Layer

To aggregate all the matching results into a fixed-size matching vector, we first concatenate all three types of matching results, and then feed the concatenated matching results in each direction into another BiLSTM individually. The last timestep outputs of the BiLSTMs are utilised as the aggregation results at two directions. The aggregation results for matching P against Q and matching Q against P are denoted as $\mathbf{t}_p \in \mathbb{R}^l$ and $\mathbf{t}_q \in \mathbb{R}^l$, respectively. Finally, we concatenate $\mathbf{t}_p$ and $\mathbf{t}_q$ and then put the concatenated vector into a MLP classifier which includes a tanh activation and softmax output layer to obtain the final matching decision.

5.3 Experiments and Results

In this section, the proposed MLCA model is evaluated on two different tasks, i.e., natural language inference and paraphrase identification, respectively. Details of the datasets and the settings for experiments are shown in Section 5.3.1 and 5.3.2. In Section 5.3.3, four different vector matching methods are compared in order to choose the most efficient one for the model. Finally, the proposed model is compared with several baseline models on two benchmark datasets in Sections 5.3.4 and 5.3.5. Ablation studies on our model are presented in Section 5.3.6.

5.3.1 Dataset

The proposed model is tested on two tasks, i.e., paraphrase identification and natural language inference. The details of datasets for these two tasks are as follows.

Quora question pair dataset[1]. It contains over $400K$ question pairs. A binary annotation is provided for each question pair which indicates whether two questions share the same meaning or not. The same split ratio is used in our experiments as mentioned in [139].

Scitail. It is an entailment classification dataset created from science domain, which contains over $27K$ examples with entailment label or neutral label. The dataset is composed of $23K$, $1.3K$ and $2K$ text pairs for training, validation and testing, respectively.

5.3.2 Experimental Setup

We use the development dataset to select the models for testing. We run 30 epochs each time and select the best performer on the development dataset as the final model for testing. Word embeddings are initialized using the 300-dimensional GloVe word vectors [89]. The window size k for convolution operation in word-by-phrase matching step is set as 2. The out-of-vocabulary words are initialized randomly within $[-0.01, 0.01]$. Hidden size of all BiLSTMs of the model is 200. We apply dropout to all layers to avoid over fitting, and the dropout ratio is 0.1. The weights are learned by minimizing the cross entropy of the

[1]https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs

training dataset by Adam optimizer [58]. The learning rate is 0.0005 and the batch size is 60 for Quora question pair dataset and 32 for Scitail dataset.

5.3.3 Vector Matching Methods Comparison

In text matching tasks, a vector matching method is usually used to obtain the matching results between the representation vectors of two texts. Previous research shows that some simple matching functions based on element-wise operations can work better than standard neural network and neural tensor network [135]. Recently, some new element-wise based vector matching methods have been proposed [15, 117], but the effectiveness of these vector matching methods for text matching have not been systematically investigated. In this section, we compare four different element-wise based vector matching methods used in [15, 117, 82]: elemen-twise subtraction (**Ele_Sub**), element-wise product (**Ele_Pro**), concat (**Concat**) method and heuristic (**Heuristic**) method. Suppose we have two representation vectors $\vec{\mathbf{a}}$ and $\vec{\mathbf{b}}$, the above four matching methods can be described by the following equations:

$$\textbf{Ele\_Sub} : \mathbf{s} = \mathbf{f}(\vec{\mathbf{a}}, \vec{\mathbf{b}}) = \vec{\mathbf{a}} - \vec{\mathbf{b}} \tag{5.13}$$

$$\textbf{Ele\_Pro} : \mathbf{s} = \mathbf{f}(\vec{\mathbf{a}}, \vec{\mathbf{b}}) = \vec{\mathbf{a}} \odot \vec{\mathbf{b}} \tag{5.14}$$

$$\textbf{Concat} : \mathbf{s} = \mathbf{f}(\vec{\mathbf{a}}, \vec{\mathbf{b}}) = [\vec{\mathbf{a}} - \vec{\mathbf{b}}, \vec{\mathbf{a}} \odot \vec{\mathbf{b}}] \tag{5.15}$$

$$\textbf{Heuristic} : \mathbf{s} = \mathbf{f}(\vec{\mathbf{a}}, \vec{\mathbf{b}}) = [\vec{\mathbf{a}}, \vec{\mathbf{b}}, \vec{\mathbf{a}} - \vec{\mathbf{b}}, \vec{\mathbf{a}} \odot \vec{\mathbf{b}}] \tag{5.16}$$

In our experiments, the above vector matching methods are used to obtain the matching results of each word in one text against the other text as described in Equation (5.6), (5.8) and (5.12). We run the proposed model MLCA with these four vector matching methods on the Scitail dataset, respectively. Furthermore, we compare the performance of the above four vector matching methods on another effective text matching model BiMPM[2]

[2]https://github.com/zhiguowang/BiMPM

[139]. We use the above vector matching methods instead of the multiple perspective cosine matching function originally used in BiMPM. Table 5.1 shows the performances of BiMPM and MLCA with different vector matching methods. We can see that both MLCA and BiMPM achieve the best performance with element-wise product.

Table 5.1 *Performances of MLCA and BiMPM with different vector matching methods on the Scitail dataset*

Model / Method	**MLCA**	**BiMPM**
Ele_Sub	73.38	77.75
Ele_Pro	**77.90**	**77.89**
Concat	75.40	75.16
Heuristic	76.87	76.81

5.3.4 Results on Paraphrase Identification

We compare our model with several baselines shown in Table 5.2. S-CNN and S-LSTM [140] belong to the representation based model. M-CNN and M-LSTM improve the performances of S-CNN and S-LSTM by using multiple perspective cosine matching function [139]. We also compare our MLCA model with several compare-aggregate based models, such as pt-DECATTword [129], pt-DECATTchar [129], L.D.C [141] and BiMPM [139]. pt-DECATTword and pt-DECATTchar [129] are variants of another compare-aggregate model proposed in [87], which are pre-trained on a noisy dataset of automatically collected question paraphrases. From Table 5.2, we can see that our model achieves 88.54% accuracy, which outperforms all the baseline models.

Table 5.2 *Performances for paraphrase identification task on the Quora question pair dataset*

Model	**Accuracy**
S-CNN [140]	79.60
M-CNN[139]	81.38
S-LSTM[140]	82.58
M-LSTM[139]	83.21
L.D.C[141]	85.55
BiMPM[139]	88.17
pt-DECATTword[129]	87.54
pt-DECATTchar[129]	88.40
MLCA	**88.54**

5.3.5 Results on Natural Language Inference

Table 5.3 shows the performances of the proposed MLCA and some baseline models on the Scitail dataset. Decomposable Attention Model (DecompAtt) [87] and ESIM [15] are two advanced models under compare-aggregate framework. N-gram is a word overlap baseline and DGEM is the decomposed graph entailment model proposed in [54]. Comparing with the two compare-aggregate baselines, i.e., DecompAtt and ESIM, which only conduct word-by-word matching, the proposed MLCA outperforms them by a large margin over 5%. Moreover, it is slightly better than DGEM with an accuracy of 77.9%.

5.3.6 Ablation Study on Scitail Dataset

In this section, we conduct an ablation on our model to examine the effectiveness of each major component. Table 5.4 shows the ablation study results on the development set of Scitail. MLCA-WW, MLCA-WP and MLCA-WS mean remove the word level

Table 5.3 *Performances for natural language inference task on the Scitail dataset*

Model	**Accuracy**
Majority[54]	60.3
DecompAtt[87]	72.3
ESIM[15]	70.6
N-gram[54]	70.6
DGEM[54]	77.3
MLCA	**77.9**

matching, phrase level matching and sentence level matching from the full model MLCA, respectively. From Table 5.4, we can see that the proposed MLCA model achieves 82.06% accuracy. If we eliminate any level of the matching from matching layer would hurt the performance especially word-by-word matching. It shows the effectiveness of matching at different levels.

Table 5.4 *Ablation study results on the development set of Scitail. T(s)/epoch: average time (second) per epoch*

Model	**Dev Acc**	**T(s)/epoch**
MLCA	82.06	39
MLCA -WW	80.29	33
MLCA -WP	81.83	36
MLCA -WS	81.37	37

5.4 Summary

This chapter presented a novel text matching model (MLCA) which matches each word in one text against the other text at three levels: word level, phrase level and sentence level. We evaluate our model on natural language inference and paraphrase identification, and experimental results show that the proposed model achieves impressive performance on both tasks due to its ability in matching at different levels. Moreover, we systematically compared four vector matching methods and found that the element-wise product method is the best choice.

Chapter 6

Multi-Level Matching Networks for Text Matching

This chapter still focuses on improving text matching by constructing neural network with the ability of capturing matching information between two texts. We focus on matching-aggregation framework and propose to use multiple levels of word representations such as word embeddings and contextualized word representations to obtain multiple word level matching results for final matching decision.

6.1 Introduction

Deep neural networks have been widely applied to text matching in recent years [7, 87, 15] and the existing deep models for text matching can be mainly categorized into two approaches. The first approach models two texts by encoding each text separately and then predicts their relationship based on the two extracted representations, taking no account of interaction between two texts [7, 23]. The second approach aims to capture direct matching features and two texts are interacted before obtaining the final representation vectors. The matching-aggregation or compare-aggregate framework [15, 139, 33] is a kind of model in line with this approach, in which words in two texts are matched firstly. Then these word level matching results are aggregated into a fixed-size vector for making final text level matching decision. Combining with bidirectional long short-term memory (BiLSTM) [44], previous matching-aggregation based models have achieved state-of-the-art performances on text matching [139, 15].

However, the problem with matching-aggregation models lies in the fact that previous models only use the final representations of words to obtain the word level matching results for text level matching decision without considering other levels of word representations. For example, the state-of-the-art model ESIM [15], firstly uses the low level pre-trained word embeddings [89] as inputs to a BiLSTM layer to generate high level contextualized word representations for representing words and their contextual information. Then an attention mechanism is employed to conduct word level matching solely based on high level contextualized word representations without considering low level representations, which can not capture sufficient information for modeling complex matching relations. For example, obviously, "I went to London yesterday" and "I went to Beijing yesterday" have different meanings and they should not be matched. However, because the contextual information of 'London' and 'Beijing' are very similar, the high level contextualized representations of these two words generated by BiLSTM layer will be very close in word representation space, which may not be sufficient to differentiate the two words, thus leading to incorrect matching decision. If the low-level word embeddings of these two words, which may be far from each other in embedding space, are also considered, the model would be aware of the difference between words 'London' and 'Beijing', which is helpful for making correct matching decision. Therefore, it is important to have multiple levels of matching to capture more matching information, hence yielding correct matching decision.

In order to address the above limitation, this work presents a multi-level matching network (MMN) for text matching, which utilises multiple levels of word representations to obtain multiple word level matching results for final text level matching decision. In each matching level, an attention mechanism is firstly used to learn the attention-aware representation of each word in two texts and to make word level matching at current level. Next, a fusion gate is used to combine the attention-aware representation with original representation of each word for word representation refinement. Then, a BiLSTM encoder is employed to generate new word representations which will be used as the inputs for next matching level. The above process is repeated for *k* times. Finally, the matching results of *k* matching levels are aggregated for final decision. The contributions of this work are summarized as follows:

- A new multi-level matching network (MMN) is proposed for text matching. The model can capture more matching information by utilising multiple levels of word representations.
- An attention aware representation fusion (AARF) layer is devised to refine word representations in each matching level.
- The model is evaluated on two popular benchmarks, SNLI and Scitail. Experimental results show that the model outperforms state-of-the-art baselines.

6.2 The MMN Model

The overall framework of the proposed MMN model is shown in Figure 6.1. It consists of five layers:

(1) the input layer for a pair of texts;

(2) the word embedding layer for representing each word in the two texts as a vector;

(3) the multi-level matching layer whose architecture is shown in Figure 6.2;

(4) the aggregation layer, where matching results from all matching levels are aggregated into a fixed-size vector;

(5) the prediction layer.

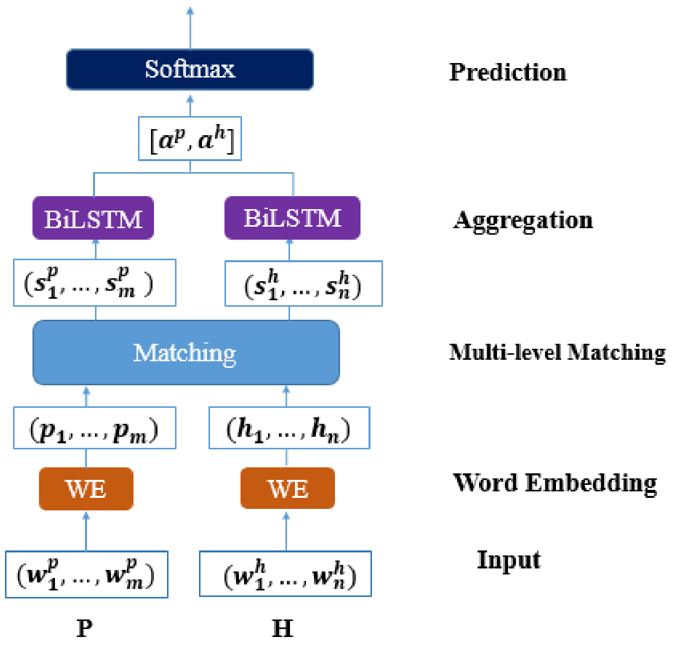


Fig. 6.1 The framework of the proposed MMN.

6.2.1 Word Embedding Layer

Given a pair of texts $P = (w_1^p, \dots, w_m^p)$ with m words and $H = (w_1^h, \dots, w_n^h)$ with n words, the purpose of this layer is to convert each word in text P and text H into a d-dimensional vector denoted as $\mathbf{p}_i \in \mathbb{R}^d$ and $\mathbf{h}_j \in \mathbb{R}^d$. The d-dimensional column vector is composed of two parts: a word-level embedding and a character-level embedding. The word-level embedding is obtained from a pre-trained word embedding matrix Glove [89]. Then we feed each character within a word into a BiLSTM, and the last time-step output of the BiLSTM is used as the character-level embedding, which is the same as [139].

6.2.2 Multi-level Matching Layer

This layer obtains the matching results based on different levels of word representations. During the k-th matching level, given the representations of two texts P and H computed in the previous matching level: $(\mathbf{p}_1^k, \ldots, \mathbf{p}_m^k)$ and $(\mathbf{h}_1^k, \ldots, \mathbf{h}_n^k)$. A word-by-word matching layer is firstly employed to obtain the word level matching results at current level.

Word-by-Word Matching. To perform word level matching, co-attention matrix $\mathbf{A}^k = (\alpha_{ij}^k)_{m\times n}$ between two texts are computed by the following equations:

$$\alpha_{ij}^k = \mathbf{p}_i^{k^T} \cdot \mathbf{h}_j^k \tag{6.1}$$

where α_{ij}^k indicates the relevance between the i-th word $\mathbf{p}_i^k$ of text P and j-th word $\mathbf{h}_j^k$ of text H, and $\alpha_{ji}^k = \alpha_{ij}^{k^T}$ otherwise. Next, for each word in one text, the relevant semantics in the other text is extracted and composed based on the co-attention matrices α_{ij}^k and α_{ji}^k by the following equations:

$$\overline{\mathbf{p}}_i^k = \sum_{j=1}^{n} \frac{\exp\left(\alpha_{ij}^k\right)}{\sum_{r=1}^{n} \exp\left(\alpha_{ir}^k\right)} \mathbf{h}_j^k \tag{6.2}$$

$$\overline{\mathbf{h}}_j^k = \sum_{i=1}^{m} \frac{\exp\left(\alpha_{ji}^k\right)}{\sum_{r=1}^{m} \exp\left(\alpha_{rj}^k\right)} \mathbf{p}_i^k \tag{6.3}$$

where $\overline{\mathbf{p}}_i^k$ and $\overline{\mathbf{h}}_j^k$ are the attention-aware representations of $\mathbf{p}_i^k$ and $\mathbf{h}_j^k$, representing the contents in $\{\mathbf{h}_j^k\}_{j=1}^n$ related to $\mathbf{p}_i^k$ and the contents in $\{\mathbf{p}_i^k\}_{i=1}^m$ related to $\mathbf{h}_j^k$, respectively. Then we use a vector matching function on each pair of $< \mathbf{p}_i^k, \overline{\mathbf{p}}_i^k >$ and $< \mathbf{h}_j^k, \overline{\mathbf{h}}_j^k >$ to obtain the word level matching results at current level between two texts.

$$\mathbf{t}_{ki}^p = \mathbf{p}_i^k \odot \overline{\mathbf{p}}_i^k \tag{6.4}$$

$$\mathbf{t}_{kj}^h = \mathbf{h}_j^k \odot \overline{\mathbf{h}}_j^k \tag{6.5}$$

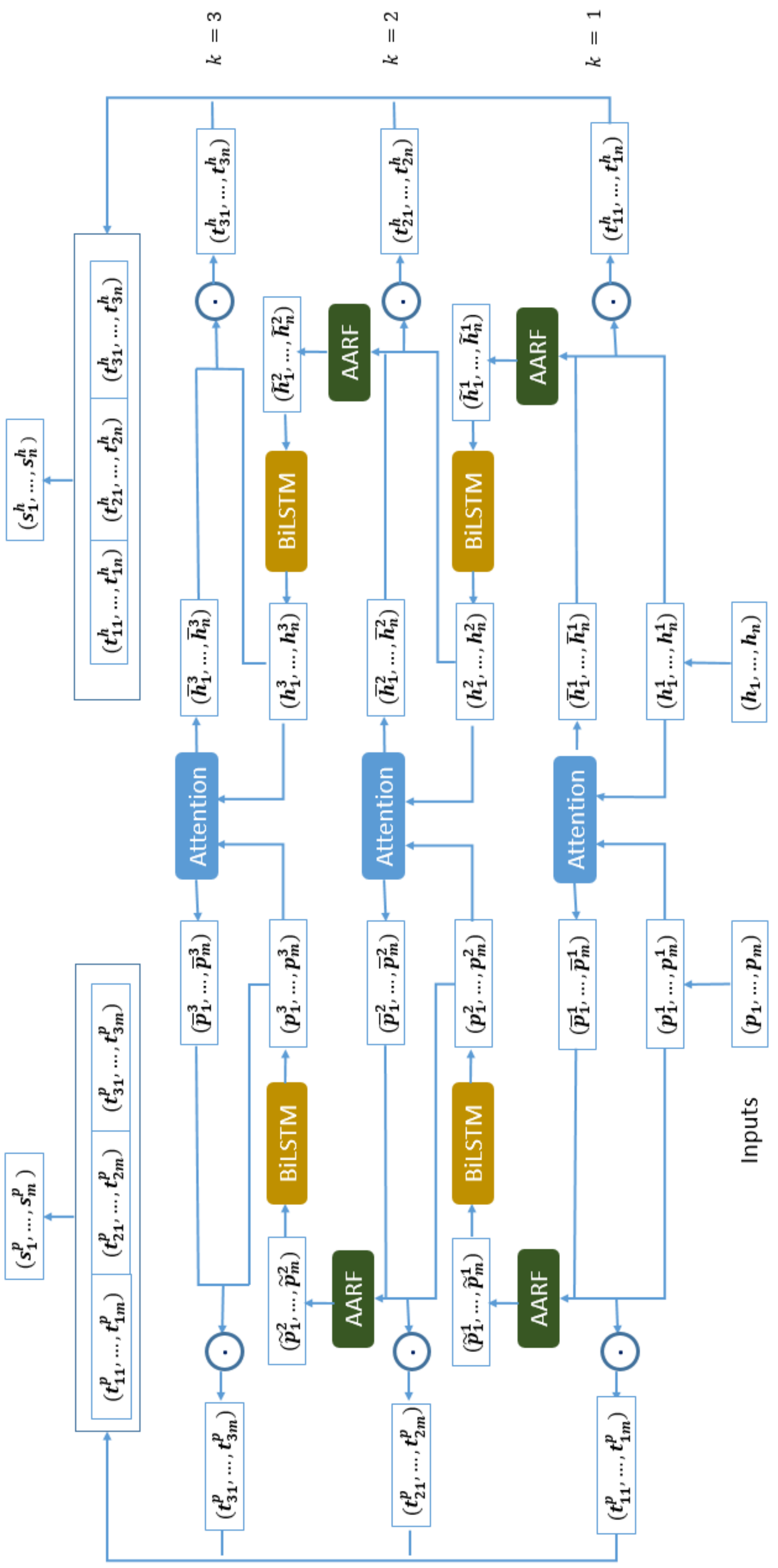


Fig. 6.2 Architecture of multi-level matching layer (taking $k = 3$ as an example).

where $\odot$ is the element-wise product operation, t^p_{ki} and t^h_{kj} are the matching results at k-th level of matching P against H and matching H against P, respectively.

Attention Aware Representaion Fusion (AARF). This layer is utilised to refine the word representation vectors. In this work, a fusion gate is used to incorporate the attention-aware representations into original representations of each word in two texts for word representation refinement.

$$\mathbf{F_p} = sigmoid(\mathbf{W}_{p1} p^k_i + \mathbf{W}_{p2} \overline{\mathbf{p}}^k_i + \mathbf{b}_p) \tag{6.6}$$

$$\mathbf{F_h} = sigmoid(\mathbf{W}_{h1} h^k_j + \mathbf{W}_{h2} \overline{\mathbf{h}}^k_j + \mathbf{b}_h) \tag{6.7}$$

$$\widetilde{\mathbf{p}}^k_i = \mathbf{F_p} \odot \mathbf{p}^k_i + (1 - \mathbf{F_p}) \odot \overline{\mathbf{p}}^k_i \tag{6.8}$$

$$\widetilde{\mathbf{h}}^k_j = \mathbf{F_h} \odot \mathbf{h}^k_j + (1 - \mathbf{F_h}) \odot \overline{\mathbf{h}}^k_j \tag{6.9}$$

where $\mathbf{W}_{p1}, \mathbf{W}_{p2}, \mathbf{W}_{h1}, \mathbf{W}_{h2} \in \mathbb{R}^{d_l \times d_l}$ and $\mathbf{b}_p, \mathbf{b}_h \in \mathbb{R}^{d_l}$ are the learable parameters of the fusion gate. $\odot$ is the element-wise product operation. $\widetilde{\mathbf{p}}^k_i, \widetilde{\mathbf{h}}^k_j \in \mathbb{R}^{d_l}$ are the refined word representation vectors after fusion.

BiLSTM Encoder. Next, a BiLSTM encoder layer is employed to encode the contextual information into the above refined representation vectors to generate new word representations.

$$\mathbf{p}^{k+1}_i = \mathbf{BiLSTM}(\widetilde{\mathbf{p}}^k_i, \mathbf{p}^{k+1}_{i-1}, \mathbf{p}^{k+1}_{i+1}) \tag{6.10}$$

$$\mathbf{h}^{k+1}_j = \mathbf{BiLSTM}(\widetilde{\mathbf{h}}^k_j, \mathbf{h}^{k+1}_{j-1}, \mathbf{h}^{k+1}_{j+1}) \tag{6.11}$$

where $\mathbf{p}^{k+1}_i, \mathbf{h}^{k+1}_j \in \mathbb{R}^{d_l}$ will be used as the inputs for next matching level.

Finally, the matching results from all matching levels are concatenated and used as the outputs of the multi-level matching layer.

$$\mathbf{s}^p_i = [\mathbf{t}^p_{1i}; \ldots; \mathbf{t}^p_{ki}] \tag{6.12}$$

$$\mathbf{s}_j^h = [\mathbf{t}_{1j}^h; \ldots; \mathbf{t}_{kj}^h] \tag{6.13}$$

where $\mathbf{s}_i^p, \mathbf{s}_j^h \in \mathbb{R}^{kd_l}$ are the concatenated matching results of matching P against H and matching H against P, respectively.

6.2.3 Aggregation and Prediction Layer

To aggregate all the matching results into a fixed-size vector, we pass the above concatenated matching results into another BiLSTM.

$$\mathbf{u}_i^p = \mathbf{BiLSTM}(\mathbf{s}_i^p, \mathbf{u}_{i-1}^p, \mathbf{u}_{i+1}^p) \tag{6.14}$$

$$\mathbf{u}_j^h = \mathbf{BiLSTM}(\mathbf{s}_i^h, \mathbf{u}_{j-1}^h, \mathbf{u}_{j+1}^h) \tag{6.15}$$

Then a mean pooling method is used to obtain the fixed-size vectors.

$$\mathbf{a}^p = \frac{1}{m}\sum_{i=1}^{m} \mathbf{u}_i^p \tag{6.16}$$

$$\mathbf{a}^h = \frac{1}{n}\sum_{j=1}^{n} \mathbf{u}_j^h \tag{6.17}$$

Finally, the above fixed-size vectors $\mathbf{a}^p$ and $\mathbf{a}^h$ are concatenated and then passed to a MLP classifier which includes a tanh activation and softmax output layer to obtain the final prediction.

6.3 Experiments and Results

6.3.1 Dataset and Experimental Setup

We evaluate our model on two datasets: SNLI [7], and Scitail dataset [54]. SNLI contains over $570K$ human annotated sentence pairs, each labeled with one of the following relationships: *entailment, contradiction, neutral.* Scitail is constructed from science domain, which contains about $27K$ sentence pairs. Unlike the SNLI dataset, Scitail uses only two labels: *entailment, neutral.*

In this work, word embeddings are initialized with the 300d GloVe word vectors [89]. The dimensions of the BiLSTM encoders are set as 400 in multi-level matching layer and 600 for aggregation layer. The number of aggregation BiLSTM layers is set as 2. The number of matching levels is tuned from $[1,4]$. Batch sizes are 32 for Scitail dataset and 128 for SNLI dataset. The Adam optimizer [58] is used for training, and the initial learning rate is set as 0.001. To avoid overfitting, we apply dropout to all layers of the model and the dropout ratio is set as 0.2.

6.3.2 Experimental Results

The accuracy metric is used to evaluate the performance of the proposed MMN and baseline models on datasets SNLI and Scitail. The performance of all baseline models come from respective papers.

SNLI. Table 6.1 shows the results of different models on the training and test sets of SNLI. DecompAtt [87] divides the text matching task into several sub-tasks using soft attention. BiMPM [139] performs matching at multi-perspective and two directions. ESIM [15] enhances the local inference procedure and achieves the state-of-the-art performance. DIIN [34] and CIN[33] are two advanced models based on CNN. From Table 6.1 we can see that the proposed MMN achieves an accuracy of 88.2% in the test sets, which outperforms all the baselines and achieves the state-of-the-art performance.

Table 6.1 *Performances on the SNLI dataset*

Model	**Train**	**Test**
DecompAtt [87]	89.5	86.3
BiMPM [139]	90.9	87.5
ESIM [15]	92.6	88.0
DIIN [34]	91.2	88.0
CIN[33]	93.2	88.0
MMN	89.3	**88.2**

Scitail. Table 6.2 shows results of the proposed MMN model and baselines on the Scitail dataset. DGEM is the decomposed graph entailment model proposed in [54]. HCRN [124] obtain the attention matrix using the complex-valued inner product (Hermitian products). CAFE [123] utilises the word level matching results for augmentation of the base word representation instead of aggregating them for prediction. From Table 6.2 we can see our model MMN outperforms all baselines and achieves the state-of-the-art performance with an accuracy of 84.8%. Comparing with ESIM which achieves the state-of-the-art performance on SNLI dataset, the proposed MMN outperforms it by a large margin over 14%.

6.3.3 Ablation Study

We perform an ablation study on the MMN model to examine the effectiveness of each major component. Table 6.3 shows the ablation study results on Scitail and SNLI datasets. First, if we remove the attention aware representation fusion (AARF) layer from the model, the performance of the model has dropped slightly on two datasets, from 84.8% to 84.24%

Table 6.2 *Performances on the Scitail dataset*

Model	**Accuracy**
DecompAtt[87]	72.3
ESIM[15]	70.6
DGEM[54]	77.3
HCRN [124]	80.0
CAFE [123]	83.3
MMN	**84.8**

on Scitail, and from 88.2% to 87.9% on SNLI. This indicates the AARF layer is helpful for improving the performance of the model. Second, if we do not use multiple levels of word representations (only use the final word representations to get word level matching results), the accuracy drops by over 1% on both datasets. According to the results, all components are effective for performance improvement.

Table 6.3 *Ablation study on Scitail and SNLI datasets*

Model	**Scitail**	**SNLI**
MMN	84.8	88.2
- AARF	84.24	87.9
- Multi-level matching	83.34	87.8

6.3.4 Effect of Number of Matching Levels

Figure 6.3 shows the effect of number of matching levels on SNLI and Scitail datasets. We observe that the optimal performance is 3 matching levels for SNLI. However, the performance of SNLI declines after 3 matching levels. Similarly, Scitail achieves its best performance at level 2 and then declines after 2 matching levels.

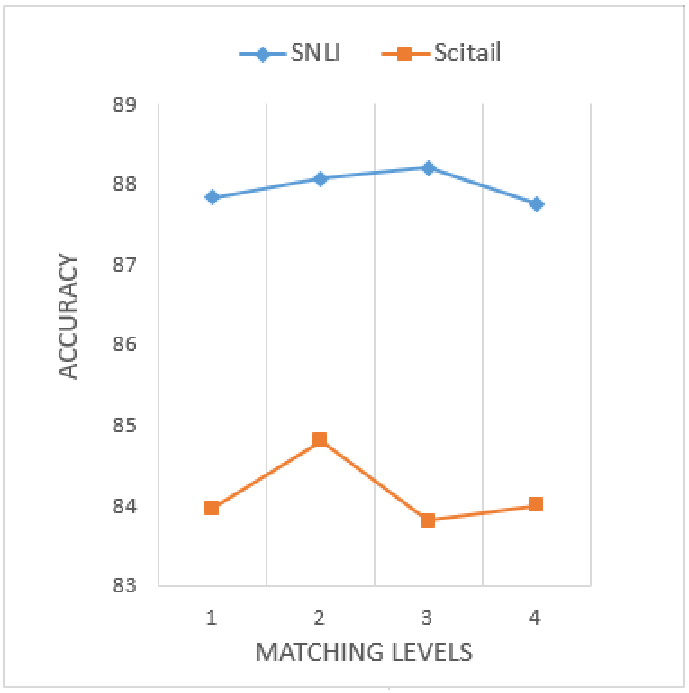


Fig. 6.3 Effect of number of matching levels.

6.4 Summary

In this chapter, we have presented a novel multi-level matching network (MMN) for text matching, which obtains word level matching results based on multiple levels of word representations to capture more matching information. The MMN model achieves the state-of-the-art performance on SNLI and Scitail datasets and demonstrates its effectiveness.

Chapter 7

Conclusions and Future Work

In this chapter, we summarise the work presented in this thesis, considers its major contributions, and discusses directions for future work.

7.1 Conclusions

The target of text matching is to identify the relationship between two texts. A variety of NLP tasks and applications involve text matching such as natural language inference, paraphrase identification, answer selection and so on. The main work in this thesis is to develop effective deep neural network models to enhance the performance of text matching. We have sought for the improvement in the following two perspectives: (1) text representation: to enhance the express power of tree-structured neural networks by dynamic composition, and (2) text interaction: to extract more matching information by performing matching at multiple granular and using multiple levels of word representations.

In Chapter 3, we proposed the TagHyperTreeLSTM model for better text representation. TagHyperTreeLSTM was devised to alleviate the inability of distinguishing different syntactic compositions of standard TreeLSTM with the aid of Part-of-Speech tags. TagHyperTreeLSTM is based on the standard TreeLSTM, which contains two separate TreeLSTMs, a tag-aware hypernetwork TreeLSTM to generate parameters of the sentence encoder TreeLSTM dynamically, and a sentence encoder TreeLSTM to generate the final representation of a sentence. Our experimental results demonstrated that the proposed model outperforms existing tree-structured neural network models with fewer parameters. Moreover, we have done an elaborate qualitative analysis on the proposed model, which gives an intuitive explanation of why our model works. Specifically, we explored behaviours of neurons in the latent vector which is computed based on the hidden state of the tag-aware hypernetwork. We found that the occurrence of large value neurons in the latent vector is dominated by nodes with specific syntactic structure or semantic basis.

In Chapter 4, we pointed out that existing dynamic compositional networks are mostly based on binarized constituency trees which cannot represent the inherent structural information of sentences effectively. To fill this research gap, we proposed a new tree-structured model TE_DCNN for text representation. We firstly introduced a novel LSTM structure, ARTree-LSTM, which was proposed to handle general constituency trees in which each inner node can have arbitrary number of child nodes. Based on ARTree-LSTM, a novel network model, Tag-Enhanced Dynamic Compositional Neural Network (TE-DCNN), was proposed for sentence representation learning, which contains two ARTree-LSTMs, i.e., tag-level ARTree-LSTM and word-level ARTree-LSTM. The tag-level ARTree-LSTM guides the word-level ARTree-LSTM in conducting dynamic composition. Experimental results showed that the TE_DCNN model improves the performance of text matching and text classification tasks on several benchmark datasets.

In addition to performing text matching based on semantic representation vectors of texts (Chapter 3 and Chapter 4). Another way is using the matching-aggregation framework which is to let smaller units (e.g., words) in two texts interact or match firstly. Then the matching results of these small units are aggregated for final matching decision (Chapter 5 and Chapter 6).

In Chapter 5, we presented the Multi-Level Compare-Aggregate model (MLCA). Previous works mainly focus on matching at word level, without considering matching at other levels such as word-by-phrase, word-by-sentence matching and so on, which is not reasonable under some circumstances. MLCA matches each word in one text against the other text at three different levels of granularity, word level (word-by-word matching), phrase level (word-by-phrase matching) and sentence level (word-by-sentence matching). The experimental results showed that the MLCA model can extract matching information at multiple granularity and outperform models performing matching at single granularity.

In Chapter 6, we described the Multi-level Matching Network (MMN). A drawback of existing methods is that word level matching results are based on the high level contextualized word representation only, while other levels of word representations are ignored. Thus, incorrect matching decisions would be made, when two words have different semantic meanings but are close in high level contextualized word representation space. To tackle this issue, our MMN model utilises multiple levels of word representations to obtain multiple word level matching results for final text level matching decision. An attention aware representation fusion layer was devised to refine word representations in each matching level. Experimental results demonstrated that the proposed MMN model is effective for text matching.

As text matching is a vital ingredient for various NLP tasks and applications, we believe that models presented in this thesis would advance the performance of these tasks and applications.

7.2 Future Directions

This thesis is focusing on investigating effective neural architectures for text matching. This section provides several future directions which have large potential and have not been resolved well at current stage.

Jointly Learning Sentence Representation and Syntactic Tree The two TreeLSTM based text representation models in this thesis both rely on syntactic parse trees which are obtained using Stanford PCFG parser [60]. This causes an extra procedure for data preparation. Although some recent works propose to learn sentence representation and its syntactic tree jointly from a downstream task like text classification or text matching [21, 75, 142], none of them achieve state-of-the-art performance. Moreover, trees they learned do not conform to the PTB grammar. As such, how to maintain the performance of the model and ensure the accuracy of the generated syntactic parse tree need to be further explored.

Knowledge enhanced Text Matching Incorporating external knowledge into neural network models has been proven meaningful in variety of NLP tasks. Currently, only a few works tried to use external knowledge to improve text matching performance [14, 51, 152]. All these models use WordNet [80] as external knowledge resource. Other knowledge resources such as DBpedia[1] [2], Freebase [2] [5], YAGO [3] [114] and so on are not well investigated in text matching.

Long Text Matching Recent research including this thesis mainly focus on matching two short texts (e.g., sentences). Comparing with short texts, long texts such as web

[1]https://wiki.dbpedia.org/
[2]https://developers.google.com/freebase
[3]https://yago-knowledge.org/

pages, legal documents or academic articles have more complicated syntactic structures and contains more information. It is difficult for LSTM or TreeLSTM based representation models to keep these information in its internal memory. Moreover, as for interaction based models, attention mechanism used in them may face bottleneck. Therefore, this remains a highly technical challenge that need to be resolved in the future.

Bibliography

[1] Ahmed, M. and Mercer, R. E. (2019). Efficient transformer-based sentence encoding for sentence pair modelling. In *Canadian Conference on Artificial Intelligence*, pages 146–159. Springer.

[2] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer.

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

[4] Bian, W., Li, S., Yang, Z., Chen, G., and Lin, Z. (2017). A compare-aggregate model with dynamic-clip attention for answer selection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1987–1990.

[5] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250.

[6] Bowman, S., Gauthier, J., Rastogi, A., Gupta, R., Manning, C. D., and Potts, C. (2016). A fast unified model for parsing and sentence understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1466–1477.

[7] Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015a). A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642.

[8] Bowman, S. R., Potts, C., and Manning, C. D. (2015b). Recursive neural networks can learn logical semantics. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 12–21.

[9] Bradbury, J., Merity, S., Xiong, C., and Socher, R. (2016). Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*.

[10] Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1994). Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*, pages 737–744.

[11] Chang, M.-W., Goldwasser, D., Roth, D., and Srikumar, V. (2010). Discriminative learning over constrained latent representations. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 429–437.

[12] Chen, H., Han, F. X., Niu, D., Liu, D., Lai, K., Wu, C., and Xu, Y. (2018a). Mix: Multi-channel information crossing for text matching. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 110–119.

[13] Chen, H., Huang, S., Chiang, D., and Chen, J. (2017a). Improved neural machine translation with a syntax-aware encoder and decoder. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1936–1945.

[14] Chen, Q., Zhu, X., Ling, Z.-H., Inkpen, D., and Wei, S. (2018b). Neural natural language inference models enhanced with external knowledge. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2406–2417.

[15] Chen, Q., Zhu, X., Ling, Z. H., Wei, S., Jiang, H., and Inkpen, D. (2017b). Enhanced lstm for natural language inference. In *Proceedings of the 55th Annual Meeting of the*

Association for Computational Linguistics (Volume 1: Long Papers), volume 1, pages 1657–1668.

[16] Chen, Q., Zhu, X., Ling, Z.-H., Wei, S., Jiang, H., and Inkpen, D. (2017c). Recurrent neural network-based sentence encoder with gated attention for natural language inference. In *Proceedings of the 2nd Workshop on Evaluating Vector Space Representations for NLP*, pages 36–40.

[17] Cheng, Z., Yuan, C., Li, J., and Yang, H. (2018). Treenet: Learning sentence representations with unconstrained tree structure. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4005–4011, Stockholm, Sweden.

[18] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). 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.

[19] Choi, J., Kim, T., and Lee, S.-g. (2018a). Element-wise bilinear interaction for sentence matching. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 107–112.

[20] Choi, J., Kim, T., and Lee, S.-g. (2019). Cell-aware stacked lstms for modeling sentences. In *Asian Conference on Machine Learning*, pages 1172–1187. PMLR.

[21] Choi, J., Yoo, K. M., and Lee, S.-g. (2018b). Learning to compose task-specific tree structures. In *Thirty-Second AAAI Conference on Artificial Intelligence*, pages 5094–5101.

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

[23] Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680.

[24] Das, D. and Smith, N. A. (2009). Paraphrase identification as probabilistic quasi-synchronous recognition. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 468–476.

[25] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.

[26] Ding, Y., Liu, Y., Luan, H., and Sun, M. (2017). Visualizing and understanding neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1150–1159.

[27] Dong, L., Wei, F., Tan, C., Tang, D., Zhou, M., and Xu, K. (2014). Adaptive recursive neural network for target-dependent twitter sentiment classification. In *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 2: Short papers)*, pages 49–54.

[28] Duan, C., Cui, L., Chen, X., Wei, F., Zhu, C., and Zhao, T. (2018). Attention-fused deep matching network for natural language inference. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4033–4040.

[29] El Hihi, S. and Bengio, Y. (1996). Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pages 493–499.

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

[31] Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252.

[32] Ghaeini, R., Hasan, S. A., Datla, V., Liu, J., Lee, K., Qadir, A., Ling, Y., Prakash, A., Fern, X., and Farri, O. (2018). Dr-bilstm: Dependent reading bidirectional lstm for natural language inference. In *Proceedings of the 2018 Conference of the North*

American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 1460–1469.

[33] Gong, J., Qiu, X., Chen, X., Liang, D., and Huang., X. (2018). Convolutional interaction network for natural language inference. In *EMNLP*, pages 1576–1585.

[34] Gong, Y., Luo, H., and Zhang, J. (2017). Natural language inference over interaction space.

[35] Guo, J., Fan, Y., Ai, Q., and Croft, W. B. (2016). A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 55–64.

[36] Guo, J., Fan, Y., Pang, L., Yang, L., Ai, Q., Zamani, H., Wu, C., Croft, W. B., and Cheng, X. (2020a). A deep look into neural ranking models for information retrieval. *Information Processing & Management*, 57(6):102067.

[37] Guo, Q., Qiu, X., Liu, P., Shao, Y., Xue, X., and Zhang, Z. (2019). Star-transformer. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1315–1325.

[38] Guo, Q., Qiu, X., Liu, P., Xue, X., and Zhang, Z. (2020b). Multi-scale self-attention for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7847–7854.

[39] Ha, D., Dai, A., and Le, Q. V. (2016). Hypernetworks. *arXiv preprint arXiv:1609.09106*.

[40] Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

[41] Hashimoto, K., Miwa, M., Tsuruoka, Y., and Chikayama, T. (2013). Simple customization of recursive neural networks for semantic relation classification. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1372–1376.

[42] Heilman, M. and Smith, N. A. (2010). Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *Human Language Technologies:*

The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pages 1011–1019.

[43] Hermans, M. and Schrauwen, B. (2013). Training and analysing deep recurrent neural networks. In *Advances in neural information processing systems*, pages 190–198.

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

[45] Hu, B., Lu, Z., Li, H., and Chen, Q. (2014). Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*, pages 2042–2050.

[46] Huang, M., Qian, Q., and Zhu, X. (2017). Encoding syntactic knowledge in neural networks for sentiment classification. *ACM Transactions on Information Systems (TOIS)*, 35(3):26.

[47] Huang, P.-S., He, X., Gao, J., Deng, L., Acero, A., and Heck, L. (2013). Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338.

[48] Huang, T., Deng, Z., Shen, G., and Chen, X. (2020). A window-based self-attention approach for sentence encoding. *Neurocomputing*, 375:25–31.

[49] Jenatton, R., Roux, N. L., Bordes, A., and Obozinski, G. R. (2012). A latent factor model for highly multi-relational data. In *Advances in neural information processing systems*, pages 3167–3175.

[50] Ji, Y. and Eisenstein, J. (2013). Discriminative improvements to distributional sentence similarity. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 891–896.

[51] Jiang, S., Li, B., Liu, C., and Yu, D. (2018). Knowledge augmented inference network for natural language inference. In *China Conference on Knowledge Graph and Semantic Computing*, pages 129–135. Springer.

[52] Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665.

[53] Khan, A., Sohail, A., Zahoora, U., and Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516.

[54] Khot, T., Sabharwal, A., and Clark, P. (2018). Scitail: A textual entailment dataset from science question answering. In *Proceedings of 32nd AAAI Conference on Artificial Intelligence*.

[55] Kim, S., Kang, I., and Kwak, N. (2019a). Semantic sentence matching with densely-connected recurrent and co-attentive information. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 6586–6593.

[56] Kim, T., Choi, J., Edmiston, D., Bae, S., and Lee, S.-g. (2019b). Dynamic compositionality in recursive neural networks with structure-aware tag representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6594–6601.

[57] Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar.

[58] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[59] Kleenankandy, J. and Nazeer, K. (2020). An enhanced tree-lstm architecture for sentence semantic modeling using typed dependencies. *arXiv preprint arXiv:2002.07775*.

[60] Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430, Sapporo, Japan.

[61] Kokkinos, F. and Potamianos, A. (2017). Structural attention neural networks for improved sentiment analysis. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 586–591.

[62] Lai, A. and Hockenmaier, J. (2014). Illinois-lh: A denotational and distributional approach to semantics. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 329–334.

[63] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

[64] Lee, K., Levy, O., and Zettlemoyer, L. (2017). Recurrent additive networks. *arXiv preprint arXiv:1705.07393*.

[65] Lei, T., Zhang, Y., Wang, S. I., Dai, H., and Artzi, Y. (2018). Simple recurrent units for highly parallelizable recurrence. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4470–4481.

[66] Li, X. and Roth, D. (2002). Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7, Taipei, Taiwan.

[67] Liu, B., Niu, D., Wei, H., Lin, J., He, Y., Lai, K., and Xu, Y. (2019). Matching article pairs with graphical decomposition and convolutions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6284–6294.

[68] Liu, C., Jiang, S., Yu, H., and Yu, D. (2018). Multi-turn inference matching network for natural language inference. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 131–143. Springer.

[69] Liu, P., Qian, K., Qiu, X., and Huang, X. (2017a). Idiom-aware compositional distributed semantics. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 1204–1213, Copenhagen, Denmark.

[70] Liu, P., Qiu, X., and Huang, X. (2016a). Syntax-based attention model for natural language inference. *arXiv preprint arXiv:1607.06556*.

[71] Liu, P., Qiu, X., and Huang, X. (2017b). Adaptive semantic compositionality for sentence modelling. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 4061–4067, Melbourne, Australia.

[72] Liu, P., Qiu, X., and Huang, X. (2017c). Dynamic compositional neural networks over tree structure. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 4054–4060. AAAI Press.

[73] Liu, Y., Meng, F., Chen, Y., Xu, J., and Zhou, J. (2020). Depth-adaptive graph recurrent network for text classification. *arXiv preprint arXiv:2003.00166*.

[74] Liu, Y., Sun, C., Lin, L., and Wang, X. (2016b). Learning natural language inference using bidirectional lstm model and inner-attention. *arXiv preprint arXiv:1605.09090*.

[75] Maillard, J., Clark, S., and Yogatama, D. (2019). Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *Natural Language Engineering*, 25(4):433–449.

[76] Marelli, M., Bentivogli, L., Baroni, M., Bernardi, R., Menini, S., and Zamparelli, R. (2014). Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 1–8, Dublin, Ireland.

[77] McCann, B., Bradbury, J., Xiong, C., and Socher, R. (2017). Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6294–6305.

[78] Mikolov, T., Grave, É., Bojanowski, P., Puhrsch, C., and Joulin, A. (2018). Advances in pre-training distributed word representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

[79] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[80] Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

[81] Mitra, B., Diaz, F., and Craswell, N. (2017). Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1291–1299.

[82] Mou, L., Men, R., Li, G., Xu, Y., Zhang, L., Yan, R., and Jin, Z. (2016). Natural language inference by tree-based convolution and heuristic matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 130–136.

[83] Nie, Y. and Bansal, M. (2017). Shortcut-stacked sentence encoders for multi-domain inference. In *Proceedings of the 2nd Workshop on Evaluating Vector Space Representations for NLP*, pages 41–45.

[84] Pang, B. and Lee, L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271, Barcelona, Spain.

[85] Pang, B. and Lee, L. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 115–124, Ann Arbor, Michigan.

[86] Pang, L., Lan, Y., Guo, J., Xu, J., Wan, S., and Cheng, X. (2016). Text matching as image recognition. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2793–2799.

[87] Parikh, A., Täckström, O., Das, D., and Uszkoreit, J. (2016). A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255.

[88] Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2014). How to construct deep recurrent neural networks: Proceedings of the second international conference on learning representations (iclr 2014). In *2nd International Conference on Learning Representations, ICLR 2014*.

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

[90] Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, 46(1-2):77–105.

[91] Qian, Q., Tian, B., Huang, M., Liu, Y., Zhu, X., and Zhu, X. (2015). Learning tag embeddings and tag-specific composition functions in recursive neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1365–1374.

[92] Qiu, X. and Huang, X. (2015). Convolutional neural tensor network architecture for community-based question answering. In *Twenty-Fourth international joint conference on artificial intelligence*.

[93] Radford, A., Jozefowicz, R., and Sutskever, I. (2017). Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*.

[94] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

[95] Robertson, S. and Zaragoza, H. (2009). The probabilistic relevance framework: Bm25 and beyond. *Information Retrieval*, 3(4):333–389.

[96] Rocktäschel, T., Grefenstette, E., Hermann, K. M., and Blunsom, T. K. P. (2015). Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*.

[97] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

[98] Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242.

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

[100] Sha, L., Chang, B., Sui, Z., and Li, S. (2016). Reading and thinking: Re-read lstm unit for textual entailment recognition. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2870–2879.

[101] Shen, G., Deng, Z.-h., Huang, T., and Chen, X. (2019). Learning to compose over tree structures via pos tags for sentence representation. *Expert Systems with Applications*, page 112917.

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

[103] Shen, T., Zhou, T., Long, G., Jiang, J., Wang, S., and Zhang, C. (2018b). Reinforced self-attention network: a hybrid of hard and soft attention for sequence modeling. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4345–4352.

[104] Shen, T., Zhou, T., Long, G., Jiang, J., and Zhang, C. (2018c). Bi-directional block self-attention for fast and memory-efficient sequence modeling. *arXiv preprint arXiv:1804.00857*.

[105] Shen, T., Zhou, T., Long, G., Jiang, J., and Zhang, C. (2018d). Fast directional self-attention mechanism. *arXiv preprint arXiv:1805.00912*.

[106] Shen, Y., Chen, J., and Huang, X. (2016). Bidirectional long short-term memory with gated relevance network for paraphrase identification. In *Natural Language Understanding and Intelligent Applications*, pages 39–50. Springer.

[107] Shen, Y., He, X., Gao, J., Deng, L., and Mesnil, G. (2014). A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*, pages 101–110.

[108] Socher, R., Bauer, J., Manning, C. D., and Ng, A. Y. (2013a). Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465, Sofia, Bulgaria.

[109] Socher, R., Huval, B., Manning, C. D., and Ng, A. Y. (2012). Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1201–1211. Association for Computational Linguistics.

[110] Socher, R., Lin, C. C., Manning, C., and Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning*, pages 129–136, Bellevue, Washington, USA.

[111] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013b). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, Seattle, Washington, USA.

[112] Speer, R., Chin, J., and Havasi, C. (2017). Conceptnet 5.5: an open multilingual graph of general knowledge. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 4444–4451.

[113] Stahlberg, F. (2020). Neural machine translation: A review. *Journal of Artificial Intelligence Research*, 69:343–418.

[114] Suchanek, F. M., Kasneci, G., and Weikum, G. (2008). Yago: A large ontology from wikipedia and wordnet. *Journal of Web Semantics*, 6(3):203–217.

[115] Surdeanu, M., Ciaramita, M., and Zaragoza, H. (2011). Learning to rank answers to non-factoid questions from web collections. *Computational linguistics*, 37(2):351–383.

[116] Sutskever, I., Tenenbaum, J. B., and Salakhutdinov, R. R. (2009). Modelling relational data using bayesian clustered tensor factorization. In *Advances in neural information processing systems*, pages 1821–1828.

[117] Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China.

[118] Tan, C., Wei, F., Wang, W., Lv, W., and Zhou, M. (2018). Multiway attention networks for modeling sentence pairs. In *IJCAI*, pages 4411–4417.

[119] Tan, M., Dos Santos, C., Xiang, B., and Zhou, B. (2016). Improved representation learning for question answer matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 464–473.

[120] Tang, D., Qin, B., and Liu, T. (2015). Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422–1432.

[121] Tay, Y., Bahri, D., Metzler, D., Juan, D.-C., Zhao, Z., and Zheng, C. (2020a). Synthesizer: Rethinking self-attention in transformer models. *arXiv preprint arXiv:2005.00743*.

[122] Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. (2020b). Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*.

[123] Tay, Y., Luu, A. T., and Hui, S. C. (2018a). Compare, compress and propagate: Enhancing neural architectures with alignment factorization for natural language inference. In *EMNLP*.

[124] Tay, Y., Luu, A. T., and Hui, S. C. (2018b). Hermitian co-attention networks for text matching in asymmetrical domains. In *IJCAI*.

[125] Tay, Y., Luu, A. T., and Hui, S. C. (2018c). Recurrently controlled recurrent networks. In *Advances in Neural Information Processing Systems*, pages 4731–4743.

[126] Tay, Y., Phan, M. C., Tuan, L. A., and Hui, S. C. (2017). Learning to rank question answer pairs with holographic dual lstm architecture. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, pages 695–704.

[127] Teng, Z. and Zhang, Y. (2016). Bidirectional tree-structured lstm with head lexicalization. *arXiv preprint arXiv:1611.06788*.

[128] Teng, Z. and Zhang, Y. (2017). Head-lexicalized bidirectional tree lstms. *Transactions of the Association for Computational Linguistics*, 5:163–177.

[129] Tomar, G. S., Duque, T., Täckström, O., Uszkoreit, J., and Das, D. (2017). Neural paraphrase identification of questions with noisy pretraining. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 142–147.

[130] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

[131] Wan, S., Lan, Y., Guo, J., Xu, J., Pang, L., and Cheng, X. (2016a). A deep architecture for semantic matching with multiple positional sentence representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2835–2841.

[132] Wan, S., Lan, Y., Xu, J., Guo, J., Pang, L., and Cheng, X. (2016b). Match-srnn: modeling the recursive matching structure with spatial rnn. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 2922–2928.

[133] Wang, B., Liu, K., and Zhao, J. (2016a). Inner attention based recurrent neural networks for answer selection. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1288–1297.

[134] Wang, B., Liu, W., Lin, Z., Hu, X., Wei, J., and Liu, C. (2018). Text clustering algorithm based on deep representation learning. *The Journal of Engineering*, 2018(16):1407–1414.

[135] Wang, S. and Jiang, J. (2016a). A compare-aggregate model for matching text sequences. *arXiv preprint arXiv:1611.01747*.

[136] Wang, S. and Jiang, J. (2016b). Learning natural language inference with lstm. In *NAACL HLT 2016: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 17, pages 1442–1451.

[137] Wang, W., Yang, N., Wei, F., Chang, B., and Zhou, M. (2017a). Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 189–198.

[138] Wang, Y., Li, S., Yang, J., Sun, X., and Wang, H. (2017b). Tag-enhanced tree-structured neural networks for implicit discourse relation classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 496–505.

[139] Wang, Z., Hamza, W., and Florian, R. (2017c). Bilateral multi-perspective matching for natural language sentences. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 4144–4150, Melbourne, Australia.

[140] Wang, Z., Mi, H., and Ittycheriah, A. (2016b). Semi-supervised clustering for short text via deep representation learning. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 31–39.

[141] Wang, Z., Mi, H., and Ittycheriah, A. (2016c). Sentence similarity learning by lexical decomposition and composition. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics*, pages 1340–1349.

[142] Williams, A., Drozdov*, A., and Bowman, S. R. (2018). Do latent tree learning models identify meaningful structure in sentences? *Transactions of the Association for Computational Linguistics*, 6:253–267.

[143] Yang, B., Wong, D. F., Xiao, T., Chao, L. S., and Zhu, J. (2017). Towards bidirectional hierarchical representations for attention-based neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1432–1441, Copenhagen, Denmark.

[144] Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W., Salakhutdinov, R., and Manning, C. D. (2018). Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

[145] Yin, W. and H.Schütze (2015). Convolutional neural network for paraphrase identification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 901–911.

[146] Yin, W. and Schütze, H. (2015). Multigrancnn: An architecture for general matching of text chunks on multiple levels of granularity. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 63–73.

[147] Yin, W., Schütze, H., Xiang, B., and Zhou, B. (2016). Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association for Computational Linguistics*, 4:259–272.

[148] Yogatama, D., Blunsom, P., Dyer, C., Grefenstette, E., and Ling, W. (2017). Learning to compose words into sentences with reinforcement learning. In *5th International Conference on Learning Representations (ICLR 2017)*. International Conference on Learning Representations.

[149] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

[150] Zhang, B. and Sennrich, R. (2019). A lightweight recurrent network for sequence modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1538–1548.

[151] Zhang, B., Xiong, D., Su, J., Lin, Q., and Zhang, H. (2018a). Simplifying neural machine translation with addition-subtraction twin-gated recurrent networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4273–4283.

[152] Zhang, Q., Yang, Y., Chen, C., He, L., and Yu, Z. (2019a). Knowledge adaptive neural network for natural language inference. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

[153] Zhang, S., Yao, L., Sun, A., and Tay, Y. (2019b). Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38.

[154] Zhang, Y., Liu, Q., and Song, L. (2018b). Sentence-state lstm for text representation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 317–327.

[155] Zhang, Y. and Zhang, Y. (2019). Tree communication models for sentiment analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3518–3527.

[156] Zhang, Y., Zhang, Z., Miao, D., and Wang, J. (2019c). Three-way enhanced convolutional neural networks for sentence-level sentiment classification. *Information Sciences*, 477:55–64.

[157] Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H., and Xu, B. (2016a). Text classification improved by integrating bidirectional lstm with two-dimensional max pooling. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3485–3495.

[158] Zhou, Q., Wang, X., and Dong, X. (2018). Differentiated attentive representation learning for sentence classification. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4630–4636, Stockholm, Sweden.

[159] Zhou, Y., Liu, C., and Pan, Y. (2016b). Modelling sentence pairs with tree-structured attentive encoder. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2912–2922.

[160] Zhu, X., Sobihani, P., and Guo, H. (2015). Long short-term memory over recursive structures. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1604–1612, Lille, France.

[161] Zuo, Y., Zeng, J., Gong, M., and Jiao, L. (2016). Tag-aware recommender systems based on deep neural networks. *Neurocomputing*, 204:51–60.