NAIST-IS-DD1461006

Doctoral Dissertation

Recurrent Neural Networks for Natural Language and Biological Sequence

Masashi Tsubaki

March 16, 2017

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of SCIENCE

Masashi Tsubaki

Thesis Committee:

Professor Yuji Matsumoto	(Supervisor)
Professor Kazushi Ikeda	(Co-supervisor)
Associate Professor Masashi Shimbo	(Co-supervisor)
Assistant Professor Hiroyuki Shindo	(Co-supervisor)
Assistant Professor Hiroshi Noji	(Co-supervisor)

Recurrent Neural Networks for Natural Language and Biological Sequence*

Masashi Tsubaki

Abstract

Recently, in various research areas such as computer vision, natural language processing (NLP), and bioinformatics, machine learning is one of the most important techniques. My research goal in this thesis is to develop machine learning methods that can (i) capture the properties of data, in particular sequential data such as natural languages and proteins, and (ii) solve the higher level problems in NLP and bioinformatics.

Very recently, deep neural networks have achieved excellent performance in solving difficult problems such as speech recognition and machine translation. While various architectures of deep neural networks have been proposed for solving various problems, in this thesis I use recurrent neural networks (RNNs). The RNN is a well-suited neural network for the problems whose inputs are arbitrary length sequences such as natural languages and proteins.

In this thesis, I focus on the problems in NLP and bioinformatics, in particular the problems of *semantic composition* and *protein structure prediction*. In addition, I solve these with various RNN-based architectures specified for each problem. I first use RNNs with long short-term memory (LSTM) units, which can find long-term dependencies in a sequence and store the information for a long period of time. I show that LSTMs provide effective and general sequential representations for both natural languages and proteins. Then, on top of the LSTM, I combine other machine learning techniques such as kernel method, convolutional neural network, and attention mechanism to capture each property in natural languages and proteins. In my experiments, I demonstrate the quantitative and qualitative effectiveness of my methods compared to various existing methods.

*Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD1461006, March 16, 2017.

Keywords:

Machine Learning, Deep Learning, Recurrent Neural Networks, Natural Language Processing, Protein Structure Prediction

Acknowledgments

松本裕治教授には，研究全般についてご指導ご鞭撻いただき，本当にありがとうございました．自由な環境で研究に専念できたことを，今後の研究者人生における大きな糧として，日々精進していきたいと思います．

池田和司教授には，私が修士 1 年の時の研究室でお世話になり，その後研究室を移動した後にも中間審査・公聴会・最終審査に至るまで助言をいただきました．深く感謝致します．

新保仁准教授には，勉強会での進捗報告や論文紹介の際に多くの助言をいただきました．特に，論文執筆の指導において非常にお世話になり，良い文章を書くことの難しさと大切さを教えていただきました．本当にありがとうございました．

進藤裕之助教授と能地宏助教授には，短い間でしたがお世話になりました．特に，能地宏助教授とは 1 年という本当に短い間でしたが，日本酒を通じて様々なお話ができたことが楽しい思い出として残っています．

Kevin Duh 先生には，現在は Johns Hopkins 大学に移られましたが，修士 2 年から博士 1 年にかけて大変お世話になりました．特に，私の初めての国際会議でのプレゼンテーションの際に，ご指導いただきありがとうございました．

加えて，秘書の北川祐子さんには，国内や海外への学会出張の際や，その他研究室生活全般について大変お世話になりました．そして友人，先輩，後輩の皆さんとは，研究室での雑談，食事会，そして飲み会など，研究室の内外で非常に楽しい時間を過ごすことが出来ました．とても感謝しています．

最後に，ずっと私を支えてくれた家族に心から感謝します．

Contents

Acknowledgments **iii**

1 Introduction **1**
1.1 Overview 1
1.2 Contributions and Outline of This Thesis 5

2 Preliminaries **9**
2.1 Feedforward Neural Networks 9
2.2 Backpropagation 11
2.3 Recurrent Neural Networks 14
2.4 Long Short-Term Memory 16
2.5 Optimization 17

3 Semantic Textual Similarity **19**
3.1 Introduction 19
3.2 Background 21
3.2.1 Word Representations 21
3.2.2 Compositional Vector Semantics 24
3.2.3 Distance Metric and Similarity Learning 25
3.2.4 Neural Networks for Paired Data 25
3.3 Method 26
3.3.1 Sentence Vector with LSTM 27
3.3.2 Non-linear Similarity Learning with Kernel Functions 28
3.3.3 Deep Multiple Kernel Learning 29
3.4 Related Work 30
3.5 Experiments and Results 31
3.5.1 Dataset 31
3.5.2 Implementation 32

3.5.3 Compared Methods . . . 33
3.5.4 Main Result . . . 34
3.5.5 Analysis . . . 35
3.6 Conclusion . . . 37

4 Protein Fold Recognition **39**
4.1 Introduction . . . 39
4.2 Related Work . . . 42
4.3 Method . . . 44
4.3.1 Evolutionary and Physicochemical Features . . . 44
4.3.2 Word Representation Learning in Proteins . . . 45
4.3.3 Protein Fold Recognition with LSTMs . . . 45
4.3.4 Training . . . 48
4.4 Experiments and Results . . . 48
4.4.1 Dataset . . . 48
4.4.2 Implementation . . . 49
4.4.3 Main Result . . . 49
4.4.4 Analysis . . . 52
4.5 Conclusion . . . 55

5 Residue-Residue Contact Prediction **57**
5.1 Introduction . . . 57
5.2 Background and Related Work . . . 60
5.2.1 Task and Evaluation . . . 60
5.2.2 Physical Constraints, Indirect Interactions, and Folding Process . . . 62
5.3 Method . . . 63
5.3.1 Features . . . 63
5.3.2 Baseline . . . 64
5.3.3 Multi-layer Stacked Bi-directional LSTM . . . 65
5.3.4 Modeling Indirect Interactions with Convolution . . . 67
5.3.5 Modeling Folding Process with Attention . . . 68
5.4 Experiments and Results . . . 70
5.4.1 Dataset . . . 70
5.4.2 Implementation . . . 71
5.4.3 Main Result . . . 72

5.4.4 Analysis 73
5.5 Conclusion 74

6 Conclusions **75**

Bibliography **77**

List of Figures

1.1 Vector spaces for natural language sentences and protein sequences . . 2
1.2 Overview of the protein residue-residue contact prediction 4

2.1 Architectures of RNN and LSTM 17

3.1 Non-linear similarly learning for semantic compositionality 20
3.2 Skip-gram (SG) and continuous bag-of-words (CBOW) of word2vec . 23
3.3 Deep multiple kernel learning 29
3.4 Some examples in the STS dataset. 31
3.5 Data padding and batch processing in RNN 32
3.6 Pearson correlations with various word representations 35
3.7 Learning curve with various word representations 36
3.8 Learning curve with various kernel functions 37

4.1 Protein fold recognition 40
4.2 Two LSTMs for protein fold recognition 42
4.3 Protein vectors with word representation learning 46
4.4 Learning curves of time vs. accuracy in protein fold recognition . . . 51
4.5 Learning curves of time vs. accuracy by various window sizes in protein fold recognition 52
4.6 Learning curves of time vs. accuracy by various batch sizes in protein fold recognition 53
4.7 Learning curves of time vs. accuracy by various sequence lengths in protein fold recognition 54
4.8 The effect of word representations and the re-training in protein fold recognition 55

5.1 An input and output of the residue-residue contact prediction task . . . 58
5.2 The definition of a protein residue-residue contact map 60
5.3 The baseline model for protein residue-residue contact prediction . . . 64

5.4 The multi-layer stacked bi-directional LSTM 66
5.5 The convolutional architectures for residue-residue contact prediction . 67
5.6 The attention mdoel for residue-residue contact prediction 69
5.7 The effects of various components in my LSTMs. 73

List of Tables

3.1 Correlation scores of semantic textual similarity task 33
3.2 Correlation scores of competitive models of semantic textual similarity task 34

4.1 Accuracy of protein fold recognition 50

5.1 Accuracy of residue-residue contact prediction 71

Chapter 1

Introduction

1.1 Overview

Recently, in various research areas such as computer vision, natural language processing (NLP), and bioinformatics, machine learning is one of the most important techniques. For example, machine learning allows computers to automatically recognize the objects in an image (i.e., visual object recognition) [39], classify whether the sentence is positive or negative (i.e., sentiment analysis) [69], and predict the 3D structure of protein from its amino acid sequence [17] using the large data on the web and maintained databases. My research goal in this thesis is to develop machine learning methods that can (i) capture the properties of data, in particular sequential data such as natural languages and proteins, and (ii) solve the higher level problems in NLP and bioinformatics.

Very recently, deep neural networks have achieved excellent performance in solving difficult problems such as visual object recognition [39], speech recognition [31], and machine translation [71]. While various deep architectures have been proposed for solving various problems, in this thesis I use recurrent neural networks (RNNs). The RNN is a well-suited neural network for the problems whose inputs are arbitrary length sequences such as natural languages and proteins. The importance of developing machine learning methods, which can map arbitrary length sequences to fixed-dimensional vector representations, is exemplified by a large number of tasks in NLP and bioinformatics (Figure 1.1).

In this thesis, I focus on the problems in NLP and bioinformatics, in particular, the problems of *semantic composition* and *protein structure prediction*. In addition, I solve these using various RNN-based architectures specified for each problem. In each

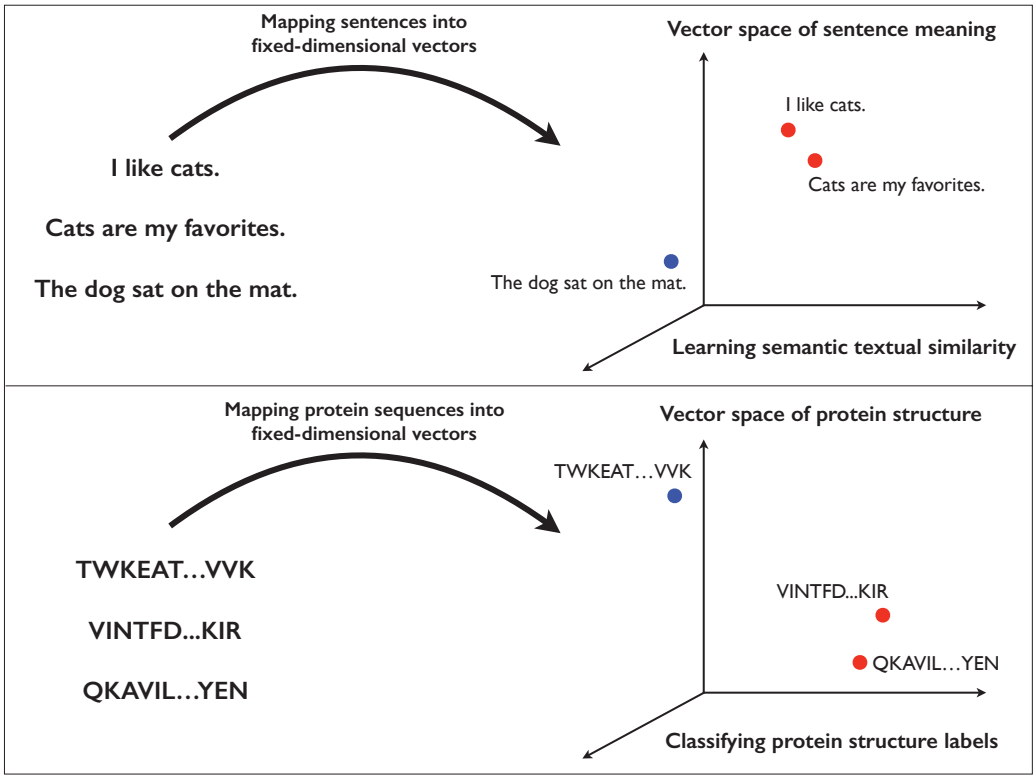

Figure 1.1: Natural languages and protein sequences have a lot in common as a sequential data. The problems in NLP and bioinformatics can be solved with similar machine learning techniques (chapter 3 and chapter 4).

problem, my concern is as follows:

1. **Semantic Composition in NLP:** The notion of semantic similarity between text data (e.g., words, sentences, and documents) plays an important role in NLP applications such as information retrieval, classification, and extraction. Recently, semantic vector space of words have become popular [15, 45, 52]. While such vector representations are sufficient to compute the semantic similarity between words, it is not trivial to capture sentence meanings composed of individual words. This is called the problem of *semantic composition* [25, 46]. Very recently, while RNN-based architectures are widely used for representing a sentence with a fixed-dimensional vector considering the word order [73], there are still some problems in semantic composition. In particular, I am concerned with the following question: "How can I represent the meaning of sentences, which cover richer meaning variations than that of words, in a vector space?" Presum-

ably, a sentence space must be of higher dimensionality than the word space, since sentences contain more information than words. For example, a sentence "Newton was inspired to formulate gravitation by watching the fall of an apple from a tree." should have a more complex representation than words "apple", "gravitation", "formulate", and "by" in the sentence.

2. **Protein Structure Classification in Bioinformatics:** Nature describes biological information as sequences of nucleotides (DNA) or of amino acids (proteins), in what is called the language of life [63]. In particular, a protein is an arbitrary length sequence of 20 kinds of amino acids such as A (alanine) and E (glutamic acid). Each protein has a unique 3D structure determined by the types and order of amino acids in the sequence. Because the protein 3D structure determines the biochemical functions, the structure information plays a key role in drug discovery and design. Based on this observation, I first tackle the problem of *protein fold recognition*. This is a classification task for protein structure labels, which are pre-defined in protein databases such as SCOP [48] and CATH [49]. This structure classification problem for proteins is used as an intermediate step for predicting the 3D structures. Generally, the process of protein fold recognition is divided into two steps: (1) extracting features and (2) learning classifiers. In existing methods, evolutionary features and support vector machine (SVM)-based classifiers are mainly used [57, 43]. However, such machine learning methods require a fixed-length feature vector as an input, whereas protein sequences have arbitrary lengths. This problem remains a fundamental challenge in protein fold recognition. In this problem, unlike natural language sentences, proteins have a univariate direction from beginning to end and it is needed to be considered both directions along the sequences. In addition, natural language sentences are usually not so long compared to protein sequences, and this might allows RNN-based architectures to work well. In this thesis, I examine whether RNN-based architectures can also work well on very long protein sequences (e.g., over 300 amino acids). Furthermore, while evolutionary information [1] is widely used as an input feature in bioinformatics, I explore whether word representation learning methods in NLP [45, 52] can also work well on biological sequences. Thus, inspiration for my protein fold recognition method comes from combining ideas from the fields of deep learning and NLP.

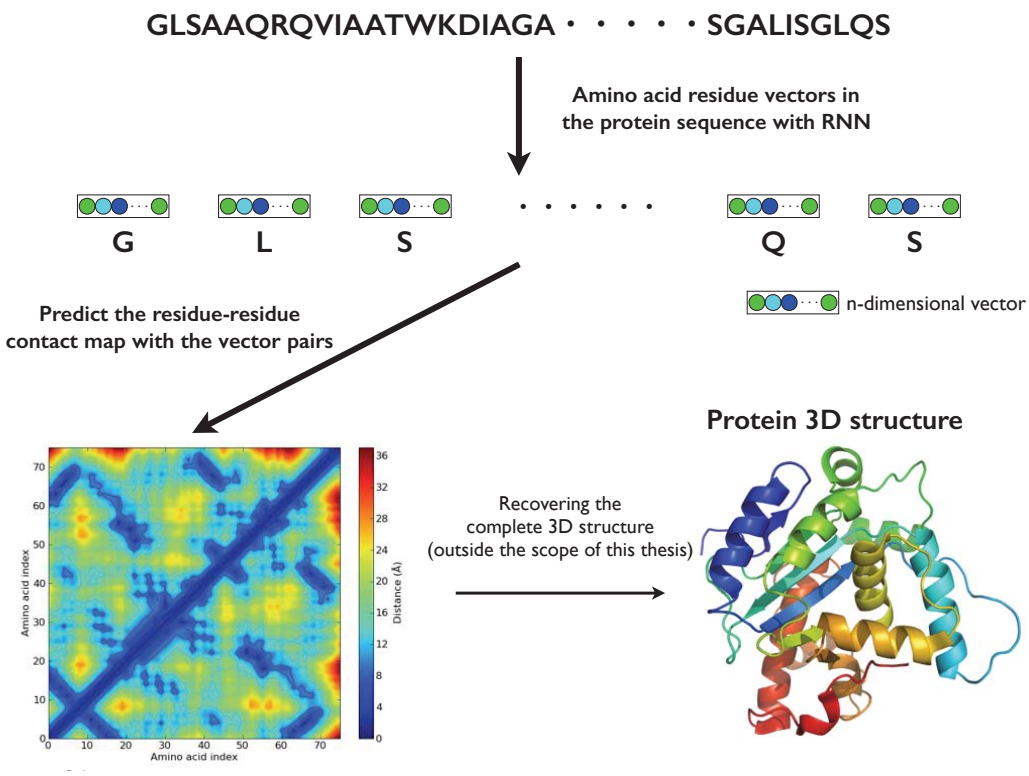

Figure 1.2: The above contact map is provided from `https://web.stanford.edu/class/cs279/lectures/lecture2.pdf`, and the above figure of protein 3D structure is provided from `https://nanohub.org/resources/contactmaps`.

Protein 3D Structure Prediction in Bioinformatics: For describing protein 3D structures more detail than fold labels, protein data bank (PDB) files provide information about 3D Cartesian coordinates of all atoms in proteins. Then we generally say that two amino acid residues in a protein sequence are in *contact* if the Euclidean distance between the residues is less than 8Å. The contact information is useful to understand the amino acid interactions, protein folding structures, biophysical and biochemical properties of proteins. In bioinformatics, it remains challenging to predict residue-residue contacts in a protein from its amino acid sequence. This problem is called *residue-residue contact prediction* (Figure 1.2) and assessed in the critical assessment of protein structure prediction (CASP), which is a worldwide experiment taking place every two years since 1994. However, this is a very difficult problem because it can always occur that two residues far away from each other in a 1D sequence get close to each other in a 3D structure due to the protein folding process. For modeling residue-residue contact prediction with machine learning, there are three main

issues as follows: (1) Residue-residue contacts are spatially correlated and not randomly distributed in native protein structures derived from the effect of evolutionary and physical information over the protein sequence. (2) In protein 3D structures, there are noises derived from the effect of indirect interaction among residues. The indirect interaction effects where direct interaction between A–B and B–C can result in observed correlations between A–C, even though no direct interaction exists between A and C. (3) Proteins do not assume a 3D conformation instantaneously, but rather through a folding process that gradually refines the 3D structure. However, most methods (1) independently consider the pair of residue features and predict the contact element-by-element, (2) indirectly model the indirect interaction (e.g., model with sparsity constraints in contact data), and (3) attempt to learn contact probabilities at the local level in a single step.

The machine learning methods in this thesis address the above problems. For the all problems in natural language sentences and protein sequences, I first use RNNs with long short-term memory (LSTM) units [33], which can find long-term dependencies in a sequence and store the information for a long period of time. I show that LSTMs provide effective and general sequential representations for both natural languages and proteins. The specific solutions for the problems and the results are described in the following section as contributions and outline of this thesis.

1.2 Contributions and Outline of This Thesis

Chapter 2: Preliminaries: In this chapter, I provide fundamental knowledge about deep neural networks.

Chapter 3: Semantic Textual Similarity: In this chapter, I propose a new method of non-linear similarity learning for semantic compositionality. In this method, both parameters in LSTM and word vector representations are learned through the semantic similarity learning for compositional sentences. In particular, the similarity learning is efficiently done in a high-dimensional space derived form various kernel functions. My motivation of the kernelization in semantic compositionality is to obtain a good high-dimensional space for representing richer meanings of sentences than words while maintaining computational tractability. On the task of predicting the semantic relatedness of two sentences in SemEval 2014 [44], the proposed method outperforms linear

baselines, feature engineering approaches, and recent other deep learning models.

Chapter 4: Protein Fold Recognition: In this chapter, I show that a simple application of LSTM can learn to map a protein sequence of arbitrary length into a fixed-dimensional vector representation, capture the long-term dependencies of amino acids in the sequence, and then correctly classify the protein fold. In particular, I propose the use of two LSTM architectures: concatenated and stacked bi-directional LSTMs for considering both directions along the protein sequences. On a benchmark dataset of EDD [20], compared to existing methods such as using 544 physicochemical attributes and the state-of-the-art method with SVM, my LSTMs achieved higher accuracy despite the use of a minimal set of features and little tuning of hyper-parameters in the neural network. In addition, I provide analyses of the differences between the concatenated and stacked LSTM architectures, the impact of hyper-parameters, and the effect of protein length. In particular, I show that the stacked architecture achieves excellent performance and the LSTM does not suffer on very long sequences of proteins (e.g., over 300 amino acids).

Chapter 5: Residue-Residue Contact Prediction: In this chapter, for residue-residue contact prediction, I design neural networks to capture the protein properties such as indirect interactions and folding process. First, I combine the stacked bi-directional LSTM (used in the chapter 4) and convolutional neural network (CNN) on top of the hidden vectors obtained by the LSTM. LSTM allows the obtained hidden vectors to consider evolutionary and physical information such as α-helix and β-strand of all residues over the protein sequence. As a result, the model can automatically capture the all features between residues even if I simply use the residue pair as an input for the contact classifier. In addition, I apply an CNN to the pairs of hidden vectors, directly consider the effect of indirect interactions between residues, and then learn the combined architecture of LSTM and CNN with end-to-end fashion. Furthermore, I attempt to model the folding process of proteins in the training process of neural networks using attention mechanism [2, 60, 58]. The attention mechanism can find some important features in the sequence for the target feature by using weights, which are called attentions. I assume that the training process of attentions, in particular sparse attentions, allows the neural network to simulate the protein folding process. On the CASP dataset, the proposed methods outperformed all existing methods based on the homology modeling and *ab initio* model with various machine learning techniques. In

particular, the stacked bi-directional LSTM with sparse attention mechanism achieved the highest performance in my methods.

The next chapter will give the necessary background of deep neural networks.

Chapter 2

Preliminaries

This chapter provides the necessary background on neural networks that will make this thesis self-contained.

Notation: Throughout this thesis, vectors are written with lowercase boldface letters (e.g., $\mathbf{v} \in \mathbb{R}^n$), matrices are written with uppercase boldface letters (e.g., $\mathbf{M} \in \mathbb{R}^{m \times n}$), and scalars and discrete symbols such as words and sequence length are written with non-bold letters (e.g., w and L). Note that $\mathbf{x}_i$ is the i-th vector in the sequence of vectors $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_L$ and x_i is the i-th element of the vector $\mathbf{x}$.

2.1 Feedforward Neural Networks

In this section, I will give a basic introduction of feedforward neural networks. Let the input vector be $\mathbf{x} \in \mathbb{R}^n$ and the weight vector be $\mathbf{w} \in \mathbb{R}^n$, the output scalar $y \in \mathbb{R}$ is computed as follows:

$$y = f(\mathbf{w}^\top \mathbf{x} + b) = f\left(\sum_{i=1}^{n} w_i x_i + b\right),$$

where $b \in \mathbb{R}$ is the bias unit and f is the non-linear activation function. One of the widely used non-linear activation function is the sigmoid function as follows:

$$f(x) = \text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}.$$

This sigmoid function maps any real number to the $[0, 1]$ interval. This can be interpreted as the probability for the neural unit parameterized by the weight vector $\mathbf{w}$ and

the bias unit b. As another activation function, the hyperbolic tangent is also widely used and its function is as follows:

$$f(x) = \tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}.$$

Despite the loss of a probabilistic interpretation such as sigmoid, $\tanh$ function is often preferred in practice due to better empirical performance (possibly due to having both negative and positive values inside the recursion unlike the sigmoid function). In addition, rectified linear unit (ReLU) function

$$f(x) = \text{ReLU}(x) = \max(0, x)$$

is also widely used in recent successful deep neural networks. ReLU does not suffer from the vanishing gradient problem [62, 5].

A neural network can be stacked using single neurons vertically (i.e., on top of each other) and is then followed by a final output layer. I will describe the computation with stacked multiplication between the weight matrix and input vector, and the application of non-linear activation function f as follows:

$$\begin{aligned}\mathbf{h} &= \mathbf{W}\mathbf{x} + \mathbf{b},\\ \mathbf{y} &= f(\mathbf{h}),\end{aligned}$$

where $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^m$ is the bias vector, and the activation function f is applied element-wise as follows:

$$f(\mathbf{h}) = f([h_1, h_2, \cdots, h_m]) = [f(h_1), f(h_2), \cdots, f(h_m)].$$

The output vector of a neural network can be seen as a transformation of the input vector that captures various interactions of the elements with the weight matrix. Then I stack more layers on top of the layer as follows:

$$\begin{aligned}\mathbf{h}^{(1)} &= f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})\\ \mathbf{h}^{(2)} &= f(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})\\ &\cdots\\ \mathbf{h}^{(\ell)} &= f(\mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}),\\ \mathbf{y} &= \mathbf{W}_y\mathbf{h}^{(\ell)} + \mathbf{b}_y,\end{aligned}$$

where ℓ is the ℓ-th layer of the deep neural network. The last layer is given to an output vector $\mathbf{y} \in \mathbb{R}^k$ on which an error function is minimized. Standard examples are a linear

output layer with a least squares error, or a softmax classifier (i.e., logistic regression) and training with the cross-entropy error. For a regression problem, the loss function with least squares error is as follows:

$$\mathcal{L}(\mathbf{y}, \mathbf{t}) = \frac{1}{2}||\mathbf{t} - \mathbf{y}||^2,$$

where $\mathbf{t} \in \mathbb{R}^k$ is the target vector or scalar (when $k = 1$) to be minimized the distance. For a classification problem, a softmax classifier is widely used as follows:

$$\mathcal{L}(\mathbf{y}, t) = \frac{\exp(y_t)}{\sum_{i=1}^{k} \exp(y_t)},$$

where $t \in \{1, \cdots, k\}$ is the index label of ouput classes. Given a set of the training samples, the final objective $\mathcal{L}_{total}$ is the sum of the above loss for each data plus an L2-regularization term as follows:

$$\mathcal{L}_{total} = \sum_{i=1}^{N} \mathcal{L}(\mathbf{y}_i, t_i) + \frac{\lambda}{2}||\Theta||_2^2,$$

where Θ is the set of all weight matrices and bias vectors to learn in the neural network, N is the total number of training samples, and λ is an L2 regularization hyper-parameter.

2.2 Backpropagation

The backpropagation algorithm [59] is based on the inverse direction of the computation in a feedforward neural network, which is to obtain all the gradients of learning parameters in the network. To illustrate this, I will give the backpropagation using a concrete example, network network-based language model in NLP: neural language model (NLM) [13, 14]. This is a good example to describe the training procedure of neural network, in particular word representation learning idea.

Specifically, I first represent a word sequence as a vector $\mathbf{x}$ as follows:

$$\mathbf{x} = [\mathbf{w}_1; \mathbf{w}_2; \cdots; \mathbf{w}_L],$$

where $\mathbf{w}_i \in \mathbb{R}^d$ is the vector representation (the initial value is random) assigned for i-th word in the sequence, L is the window size to be considered as a hyper-parameter, and $\mathbf{x} \in \mathbb{R}^{dL}$ is the concatenation of all the word vectors. Following, I use $\mathbf{x}$ as the

input vector for the neural network. In the NLM, the score $z \in \mathbb{R}$ for the above word sequence is computed with a neural network computation as follows:

$$\begin{aligned}\mathbf{y} &= \mathbf{W}\mathbf{x} + \mathbf{b}, \\ \mathbf{a} &= f(\mathbf{y}), \\ z &= \mathbf{u}^\top \mathbf{a},\end{aligned}$$

where $\mathbf{W} \in \mathbb{R}^{h \times dL}$ is the weight matrix to learn (h is the size of hidden layer), $\mathbf{b} \in \mathbb{R}^h$ is the bias vector to learn, and $\mathbf{u} \in \mathbb{R}^h$ is the weight vector to learn for computing the score z with dot product. Then I also create a corrupted sequence $\mathbf{x}' = [\mathbf{w}_1; \mathbf{w}_2; \cdots; \mathbf{w}'_L]$, where the word of $\mathbf{w}'_L$ is chosen randomly from the vocabulary. The score z' is computed for this implicit negative sequence with the same neural network. Finally, we get the cost function to be minimized in NLM for the all training sequence samples in a corpus as follows:

$$\mathcal{L}(\Theta) = \sum_{i=1}^{N} \max(0, 1 - z_i + z'_i), \tag{2.1}$$

where N is the number of training samples and $\Theta = (\mathbf{u}, \mathbf{W}, \mathbf{b}, \mathbf{x}_i)$. This cost function aims the correct sequence score to be higher than corrupted sequence score. Note that the set of training parameters Θ contains $\mathbf{x}_i$ (i.e., input word vectors) in addition to the standard learning parameters in the neural network such as weight matrix $\mathbf{W}$. This is called *word representation learning* in a neural network architecture. Following, in practice I describe the backpropagation to compute the gradients of the above four parameters $\mathbf{u}$, $\mathbf{W}$, $\mathbf{b}$, and $\mathbf{x}$.

For the condition $1 - z + z' > 0$ in Eq. 2.1, I describe the derivative for the score z. The first gradient, which is for the weight vector $\mathbf{u}$, is simply computed as follows:

$$\frac{\partial z}{\partial \mathbf{u}} = \frac{\partial \mathbf{u}^\top \mathbf{a}}{\partial \mathbf{u}} = \mathbf{a}.$$

Next, the gradients for weight matrix $\mathbf{W}$, which is main learning parameter in the

network, can be obtained with *chain rule* and computed as follows:

$$\begin{aligned}
\frac{\partial z}{\partial W_{ij}} &= \frac{\partial u_i a_i}{\partial W_{ij}} \\
&= u_i \frac{\partial a_i}{\partial W_{ij}} \\
&= u_i \frac{\partial a_i}{\partial y_i} \frac{\partial y_i}{\partial W_{ij}} \\
&= u_i \frac{\partial f(y_i)}{\partial y_i} \frac{\partial y_i}{\partial W_{ij}} \\
&= u_i f'(y_i) \frac{\partial (\mathbf{w}_i^\top \mathbf{x} + b_i)}{\partial W_{ij}} \\
&= u_i f'(y_i) x_j = \delta_i x_j,
\end{aligned}$$

where $\mathbf{w}_i$ is the i-th column vector of $\mathbf{W}$. The above

$$u_i f'(y_i) = u_i f'(\mathbf{w}_i^\top \mathbf{x} + b_i) = \delta_i$$

is called a *delta message* or *error signal*, and in summary I describe the above derivative as follows:

$$\frac{\partial z}{\partial W_{ij}} = \delta_i x_j.$$

Note that since all combinations of i and j are needed to be considered, we use the outer product of the vectors and represent the gradient of $\mathbf{W}$ as follows:

$$\frac{\partial z}{\partial \mathbf{W}} = \boldsymbol{\delta} \mathbf{x}^\top.$$

In addition, the derivative of the bias vector $\mathbf{b}$ is computed in a similar way and represented as follows:

$$\frac{\partial z}{\partial \mathbf{b}} = \boldsymbol{\delta}.$$

So far, I describe the backpropagation as the use of derivatives and chain rule for the network parameters, i.e., $\mathbf{u}$, $\mathbf{W}$, and $\mathbf{b}$. In addition, the final goal is to train the word representations, i.e., input vector $\mathbf{x}$ for the neural network.

For the training of word representations as input vectors, I can also describe the derivatives with the chain rule for the j-th element of the input vector $\mathbf{x}$ as follows:

$$\begin{aligned}
\frac{\partial z}{\partial x_j} &= \sum_{i=1}^{h} \frac{\partial z}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{h} u_i \frac{\partial f(\mathbf{w}_i^\top \mathbf{x} + b_i)}{\partial x_j} \\
&= \sum_{i=1}^{h} u_i f'(\mathbf{w}_i^\top \mathbf{x} + b_i) \frac{\partial \mathbf{w}_i^\top \mathbf{x}}{\partial x_j} \\
&= \sum_{i=1}^{h} \delta_i \frac{\partial \mathbf{w}_i^\top \mathbf{x}}{\partial x_j} \\
&= \sum_{i=1}^{h} \delta_i W_{ij} \\
&= \boldsymbol{\delta}^\top \mathbf{w}_j.
\end{aligned}$$

In summary, I describe the derivative as follows:

$$\frac{\partial z}{\partial \mathbf{x}} = \mathbf{W}^\top \boldsymbol{\delta}.$$

Minimizing the objective function of Eq. 2.1 allows us to obtain low-dimensional word vector representations in the training process of neural network, which can capture syntactic and semantic similarities between words.

2.3 Recurrent Neural Networks

In this section, I describe a recurrent neural network (RNN), which the central object of this thesis. RNNs have a recursive function on a hidden layer $\mathbf{h}_t \in \mathbb{R}^n$, where t is the time step. At each t, $\mathbf{h}_t$ is computed from two vectors: the input vector $\mathbf{x}_t \in \mathbb{R}^n$ and its previous hidden layer $\mathbf{h}_{t-1}$ (when the time step $t = 1$, $\mathbf{h}_0 = \mathbf{0}$). For these vectors, RNNs use an affine transformation with non-linear functions f such as $\tanh$

and output $\mathbf{y}_t \in \mathbb{R}^m$ as follows:

$$\begin{aligned}\mathbf{u}_t &= \mathbf{W}_h[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_h, \\ \mathbf{h}_t &= f(\mathbf{u}_t), \\ \mathbf{y}_t &= \mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y,\end{aligned}$$

where $\mathbf{W}_h \in \mathbb{R}^{n \times 2n}$ and $\mathbf{W}_y \in \mathbb{R}^{m \times n}$ are weight matrices to learn, $\mathbf{b}_h \in \mathbb{R}^m$ and $\mathbf{b}_y \in \mathbb{R}^m$ are bias vectors to learn, and m is the output dimensionality. Since RNN has a recursive architecture, it is easy to follow the computation using pseudo code as follows.

1: $\mathbf{h}_0 = \mathbf{0}$
2: **for** t **from** 1 **to** T **do**
3: $\quad \mathbf{u}_t = \mathbf{W}_h[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_h$
4: $\quad \mathbf{h}_t = f(\mathbf{u}_t)$
5: $\quad \mathbf{y}_t = \mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y$
6: **end for**

Thus an RNN can be recognized as a deep feedforward neural network that has a layer for each time step, its weight parameters to learn are shared across time, and the depth is equivalent to the sequence length. In addition, the loss function of the RNN is a sum of per-timestep losses as follows:

$$\mathcal{L}_{total} = \sum_{t=1}^{T} \mathcal{L}(\mathbf{y}_t, \mathbf{t}_t),$$

where $\mathcal{L}(\mathbf{y}_t, \mathbf{t}_t)$ is the loss function between $\mathbf{y}_t$ and $\mathbf{t}_t$ such as mean squared error and softmax cross-entropy as described in the previous section.

Indeed, RNNs are generally difficult to train due to the problems with long-range temporal dependencies [62, 5]. For example, in learning of RNNs for long sequences, a small change to an iterative process can grow after many iterations and then the derivative of the loss function at one time can be large. On the other hand, RNNs also suffer from the vanishing gradient problem, which the derivative can be vanished. Thus, the loss function of RNNs is very sensitive and the vanishing and exploding gradient problems make it difficult to train RNNs on sequences with long-range temporal dependencies.

2.4 Long Short-Term Memory

To address the vanishing and the exploding gradient problems, Hochreiter and Schmidhuber, 1997 [33] proposed an RNN with long short-term memory (LSTM) units. LSTMs have memory cells in a hidden layer in which information can be stored for a long period of time. LSTMs can find long-term dependencies in a sequence and store the information thanks to the memory cells.

Specifically, the memory cells consist of three types of gates that control the flow of information into and out of these cells: an input gate $\mathbf{i}_t \in \mathbb{R}^n$, a forget gate $\mathbf{f}_t \in \mathbb{R}^n$, and an output gate $\mathbf{o}_t \in \mathbb{R}^n$. Given an input vector $\mathbf{x}_t \in \mathbb{R}^n$, the previous cell layer $\mathbf{c}_{t-1} \in \mathbb{R}^n$, and a hidden layer $\mathbf{h}_{t-1} \in \mathbb{R}^n$ (when the time step $t = 1$, $\mathbf{h}_0 = \mathbf{0}$ and $\mathbf{c}_0 = \mathbf{0}$), an LSTM computes the next $\mathbf{c}_t$ and $\mathbf{h}_t$ as follows:

$$\mathbf{u}_t = [\mathbf{x}_t; \mathbf{h}_{t-1}], \tag{2.2}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{u}_t + \mathbf{b}_i), \tag{2.3}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{u}_t + \mathbf{b}_f), \tag{2.4}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{u}_t + \mathbf{b}_o), \tag{2.5}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_c \mathbf{u}_t + \mathbf{b}_c), \tag{2.6}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{2.7}$$

where $\mathbf{W}_t, \mathbf{W}_f, \mathbf{W}_o, \mathbf{W}_c \in \mathbb{R}^{n \times 2n}$ are weight matrices to learn, $\mathbf{b}_t, \mathbf{b}_f, \mathbf{b}_o, \mathbf{b}_c \in \mathbb{R}^n$ are bias vectors to learn, σ is the element-wise sigmoid function, $\tanh$ is the element-wise hyperbolic tangent function, and $\odot$ is the element-wise multiplication of two vectors. These parameters can be trained using a standard backpropagation technique. In this paper, we represent a series of computation in Equations (2.2)–(2.7) by

$$\langle \mathbf{h}_t, \mathbf{c}_t \rangle = \text{lstm}(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}). \tag{2.8}$$

The above parameters in an LSTM can be trained using a standard backpropagation technique. Figure 2.1 shows the RNN and LSTM architectures. LSTM uses memory cells located in a hidden layer of RNNs and can store information for a long period of time. The gates allow the cells to store and access information over long periods of time. When the input gate $\mathbf{i}_t$ is closed, the new input information is not affect the previous cell state $\mathbf{c}_{t-1}$. Forget gate $\mathbf{f}_t$ removes the historical information stored in the cells. When the output gate $\mathbf{o}_t$ is open, the network can access stored information in the cells.

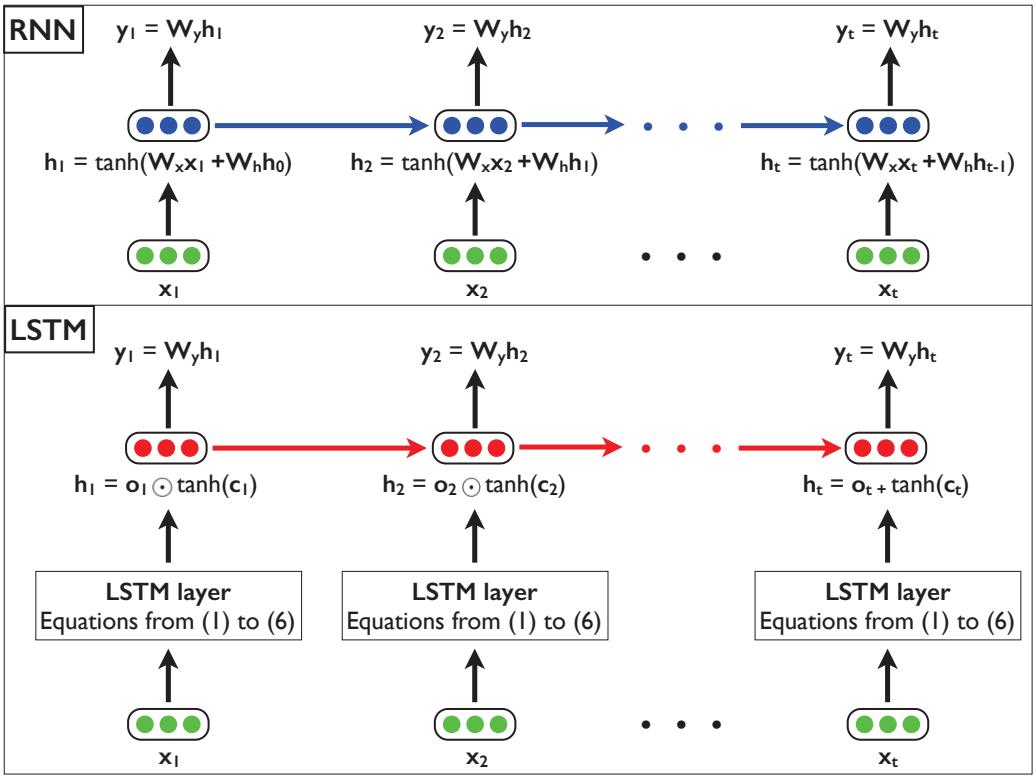

Figure 2.1: Architectures of RNN and LSTM.

2.5 Optimization

Generally, deep neural networks are trained with stochastic gradient descent-based optimization methods. In particular, Kingma and Ba [38] introduced ADAM, which is an algorithm for first-order gradient descent-based optimization, based on adaptive estimates of lower-order moments. ADAM is easy to implement, computationally efficient, required little memory, and is well suited for problems that are large in terms of data and parameters. In addition, the hyper-parameters in ADAM have intuitive interpretations and typically require little tuning. The empirical results in KIgma and Ba's paper [38] show that ADAM works well in practice compared to other stochastic optimization methods.

More precisely, let the gradient vector be $\mathbf{g}_t \in \mathbb{R}^n$, where n is the number of parameters to train and t is the time step, each momentum $\mathbf{m}_t \in \mathbb{R}^n$ and $\mathbf{v}_t \in \mathbb{R}^n$ (when the

time step $t = 0$, $\mathbf{m}_0 = \mathbf{0}$ and $\mathbf{v}_0 = \mathbf{0}$) is computed as follows:

$$\begin{aligned}
\mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t, \\
\mathbf{v}_t &= \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2, \\
\hat{\mathbf{m}}_t &= \frac{\mathbf{m}_t}{1 - \beta_1^t}, \\
\hat{\mathbf{v}}_t &= \frac{\mathbf{v}_t}{1 - \beta_2^t},
\end{aligned}$$

and then the learning parameter $\mathbf{x}_t \in \mathbb{R}^n$ (i.e., vectorized parameters of all weight matrices in the neural network) is updated as follows:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\alpha}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \hat{\mathbf{m}}_t,$$

where $\sqrt{\hat{\mathbf{v}}_t}$ denotes the element-wise square root. Note that the learning rate $\alpha = 0.001$, first momentum coefficient $\beta_1 = 0.9$, second momentum coefficient $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$, which are the recommended configurations in [38].

Chapter 3

Semantic Textual Similarity

3.1 Introduction

The notion of semantic similarity between text data (e.g., words, phrases, sentences, and documents) plays an important role in NLP applications such as information retrieval, classification, and extraction. Although the simplest similarity measurement is based on word matchings (i.e., counting the same words in two documents) rather than word meanings this suffers from lack of generalization.

Recently, semantic vector space of words has become popular [78, 14, 52]. While such word vector representations are sufficient to compute the semantic similarity between words, it is not trivial to capture the meaning of phrases and sentences composed of individual words. To overcome the weakness, modeling and learning semantic compositionality have received a lot of attention [46]. The goal of this research is to formulate how word vectors and operations are learned and modeled to properly represent phrasal and sentential semantics. However, there are still some problems in this compositional vector semantics. In particular, in this thesis I am concerned with the following question: "How can we represent the meaning of sentences, which cover richer meaning variations than that of words, in a vector space?"

To answer this question, I propose a new method of *non-linear similarity learning* for semantic compositionality. My approach is to capture the sentence meanings in a vector space different from that of words. Presumably, this space must be of higher dimensionality than the word space, since a sentence contains far more information than words. For example, a sentence "Newton was inspired to formulate gravitation by watching the fall of an apple from a tree." should have a more complex (therefore, higher-dimensional) representation than words "apple", "gravitation", "formu-

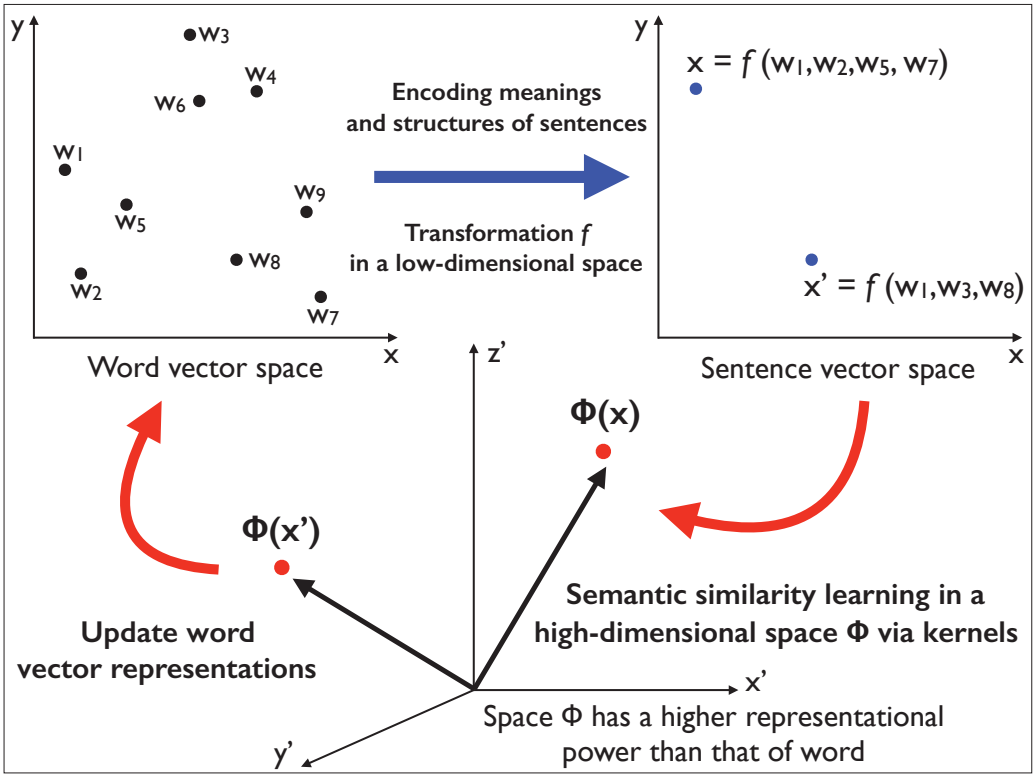

Figure 3.1: Non-linear similarly learning for semantic compositionality with kernel functions, which can implicitly map to a high-dimensional semantic space.

late", and "by".

The proposed method is inspired by the previous work on distance metric and similarity learning [3], and in particular leverages the non-linear capacity of kernel functions [36]. My goal is to obtain word representations and compositional functions for sentences through the similarity learning of sentential meaning in the kernel-induced high-dimensional space. Figure 3.1 shows the geometric intuition of my model architecture. I can learn and obtain the parameters (i.e., word representations and parameters in LSTM as a compositional function) inexpensively by using non-linear kernel functions without explicit computation of sentence vectors in the high-dimensional space. Note that my approach differs from that of recent deep learning models such as [67] and [73]. These use intricate neural network architectures to model and learn sentence representations only in a low-dimensional space, whereas my method integrates both low- and high-dimensional space.

My contributions are two-fold:

1. To the best of my knowledge, my method is the first work that addresses compositionality by combining similarity learning in a kernel-induced high-dimensional space, and structure embedding in a low-dimensional space.

2. My method is simple and effective. It outperforms linear baselines and feature engineering approaches, and achieves competitive results with recent deep learning models on the task of predicting the semantic relatedness of two sentences.

3.2 Background

In NLP, learning methods of low-dimensional vector representations for words have a long history [15]. One of the recent successful methods is based on neural network language models [14]. In these models, all word representations are initialized with random vectors and trained with a large corpus. The learned word representations have been used as feature vectors in various NLP tasks [46, 76, 69].

3.2.1 Word Representations

Latent Semantic Analysis (LSA)

Let $\mathbf{M}$ be a matrix of co-occurrences between words and documents. The element of $\mathbf{M}$ is $\#(w, c)$, which is the number of times that each word w appears in each document c, and the dimensionality of $\mathbf{M}$ is $W \times C$ where W is the number of words and C is the number of documents. Each row of matrix $\mathbf{M}$ can be regarded as a vector representation of a word, and each column a representation of a document. Since W and C are huge and the number of words in a document and the number of documents a word appears are both limited, these vectors are sparse and high-dimensional. Although such vectors work well in some cases, there are advantages for efficiency to use dense and low-dimensional representations. Such vectors can be obtained with a dimensionality reduction technique such as latent semantic analysis (LSA) [15].

LSA uses singular value decomposition (SVD) to factorize the word-document co-occurrence matrix $\mathbf{M}$ into the product of three matrices as follows:

$$\mathbf{M} \approx \mathbf{U}_d \mathbf{\Sigma}_d \mathbf{V}_d^\top,$$

where $d \ll \min(W, C)$ is the reduced dimensionality of data, $\mathbf{U}_d$ ($W \times d$) and $\mathbf{V}_d$ ($C \times d$) are left and right singular orthonormal matrices, and $\mathbf{\Sigma}$ ($d \times d$) is a diagonal matrix of singular values in decreasing order. From (1), I obtain word representation matrix $\mathbf{W} = \mathbf{U}_d\mathbf{\Sigma}_d$ ($W \times d$) and $\mathbf{C} = \mathbf{V}_d$, where I view the rows of $\mathbf{W}$ as word vectors and the rows of $\mathbf{C}$ as document vectors. Since d is much smaller than W and C, they are typically dense low-dimensional vectors.

Skip-Gram (SG) and Continuous Bag-Of-Words (CBOW) of Word2vec

The training objective of the skip-gram (SG) model of word2vec [45] is to learn d-dimensional word representations that are useful for predicting surrounding words of a target word in a sentence. Specifically, given a sequence of training words $w_1, w_2, \cdots, w_L$, the objective function J to maximize is the average log probability as follows:

$$J = \frac{1}{L}\sum_{t=1}^{L}\sum_{-\ell \le i \le \ell} \log p(c_{t+i}|w_t), \tag{3.1}$$

where w_t is the center word, ℓ is the window size of the context which is the number of words to consider around of w_t, and c_i is the i-th context word in the window. In the SG model, the probability in equation (3.1) is defined as follows:

$$p(c_{t+i}|w_t) = \frac{\exp({\mathbf{c}_{t+i}}^\top \mathbf{w}_t)}{\sum_{j=1}^{V} \exp({\mathbf{c}_j}^\top \mathbf{w}_t)}, \tag{3.2}$$

where V is the number of words in the vocabulary, $\mathbf{w} \in \mathbb{R}^d$ is the d-dimensional vector representation of word w, and $\mathbf{c} \in \mathbb{R}^d$ is the d-dimensional vector representation of context word c. Note that $\mathbf{w}_i \in \mathbb{R}^d$ is the i-th row vector of $\mathbf{W} \in \mathbb{R}^{V \times d}$ and $\mathbf{c}_i \in \mathbb{R}^d$ is the i-th column vector of $\mathbf{C} \in \mathbb{R}^{d \times C}$, where C is the number of context words.

However, the above function is impractical because its computational cost is proportional to V. To reduce the cost, the word2vec toolkit implements negative sampling. Negative sampling approximates the probability in equation (3.2) by maximizing the inner product of vectors for correct example pairs (w, c) that occur in the corpus, and minimizing it for negative example pairs (w, c') that do not occur in the corpus. With negative sampling, the approximate probability of equation (3.2) is computed as follows:

$$p(c_{t+i}|w_t) \approx \sigma(\mathbf{c}_{t+i}^\top \mathbf{w}_t) \prod_{k=1}^{K} \sigma(-{\mathbf{c}'}_k^\top \mathbf{w}_t),$$

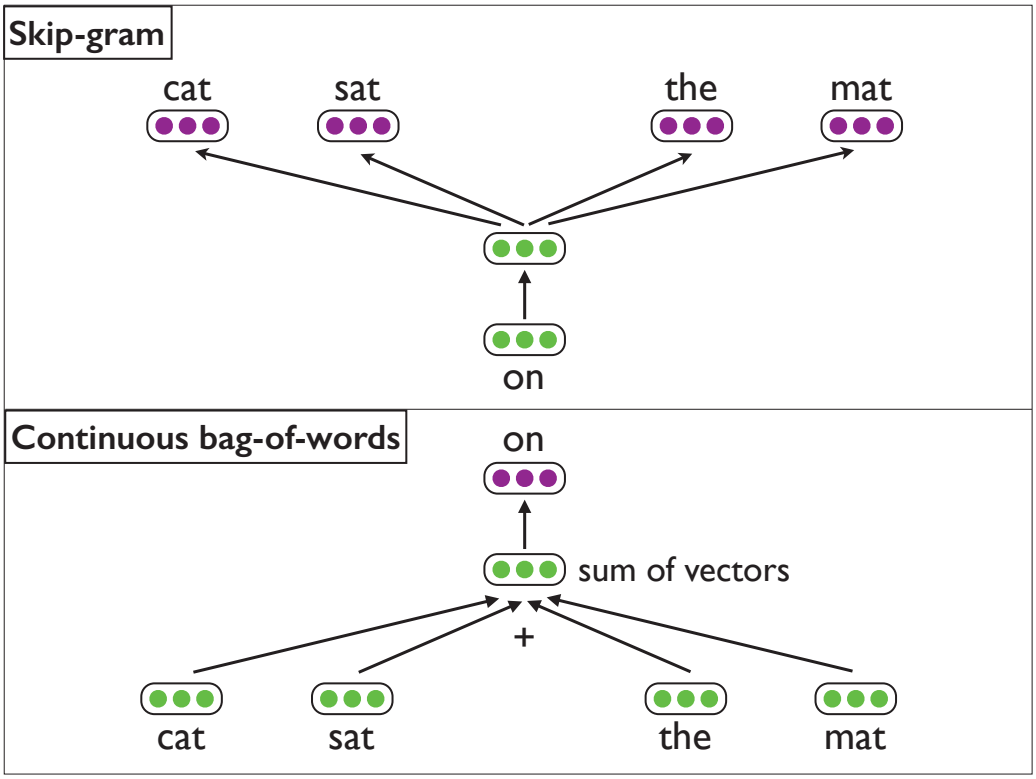

Figure 3.2: Models of skip-gram (SG) and continuous bag-of-words (CBOW) with a sequence of training words "cat sat on the mat" as a example. The SG model predicts surrounding words (i.e., "cat", "sat", "the", "mat") given the current word (i.e., "on"). In contrast, the CBOW model predicts the center word based on the sum of the surrounding word vectors.

where σ is the sigmoid function, $\mathbf{c}'$ is the word vector of negative example, and K is the number of negative sampling words.

In summary, the SG model predicts surrounding words given a center word. In contrast, continuous bag-of-words (CBOW) model, which is another implementation in word2vec, predicts the center word based on the sum of the surrounding word vectors. Thus, CBOW is the opposite of the skip-gram. Figure 3.2 illustrates these two approaches.

Global Vectors (GloVe)

Global vectors (GloVe) [52] relies on a global log-bilinear regression model that combines and leverages the advantages of two ideas: (i) a local context window such

as used in word2vec and (ii) a global co-occurrence statistics in a corpus.

Let w_i be the i-th word, $\mathbf{w}_i \in \mathbb{R}^d$ be the d-dimensional vector for the word w_i, c_j be the j-th context, and $\mathbf{c}_j \in \mathbb{R}^d$ be the d-dimensional vector of context c_j. The objective function J to minimize is defined as follows:

$$J = \sum_{i,j=1}^{V} f(\#(w_i, c_j))(\mathbf{w}_i^\top \mathbf{c}_j + b_i + b_j - \log(\#(w_i, c_j))^2, \tag{3.3}$$

where V is the number of words in the vocabulary, $\#(w, c)$ is the number of times that word w occurs in context c, b_i and b_j are word/context-specific bias terms that are also learning parameters in addition to $\mathbf{w}$ and $\mathbf{c}$, and $f(x)$ is a weighting function. In [52], the authors use the weighting function

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & x < x_{\max} \\ 1 & \text{otherwise,} \end{cases}$$

where $x_{\max} = 100$ in all experiments and report that $\alpha = 3/4$ gives a modest improvement over $\alpha = 1$. From Equation (3.3), I see that GloVe is fit to minimize the weighted least square loss giving more weight to frequent (w, c) with this weighting function $f(x)$.

We can also explicitly define GloVe as a factorization of a log-count matrix shifted by the bias matrix as follows [15]:

$$\mathbf{M} \approx \mathbf{W}\mathbf{C}^\top + \mathbf{B},$$

where $\mathbf{M}$ is the log-count matrix of which the (i, j)-element $\mathbf{M}_{i,j} = \log(\#(w_i, c_j))$, $\mathbf{W}$ and $\mathbf{C}^\top$ are factorized matrices consisting of word vectors and context vectors respectively, and $\mathbf{B}$ is the matrix of bias terms of which the (i, j)-element $\mathbf{B}_{i,j} = b_i + b_j$.

3.2.2 Compositional Vector Semantics

Despite their usefulness of word representations, the vectors do not capture semantic compositionality. As a result, modeling and learning compositional semantics in the word vector space have emerged as another important line of research [46]. For phrases and sentences, many different models have been explored [69, 77].

In particular, recursive neural networks (RNNs) are used to represent a sentence vector. RNNs use the following composition function to compute a phrase vector $\mathbf{p}$

from two vectors $\mathbf{w}_i \in \mathbb{R}^d$ and $\mathbf{w}_j \in \mathbb{R}^d$ of words w_i and w_j:

$$\mathbf{p} = f\left(\mathbf{W}\begin{bmatrix}\mathbf{w}_i \\ \mathbf{w}_j\end{bmatrix} + \mathbf{b}\right),$$

where $\mathbf{W} \in \mathbb{R}^{d \times 2d}$ is the weight matrix to learn, $\mathbf{b} \in \mathbb{R}^d$ is the bias vector to learn, and f is a non-linear function such as sigmoid. To obtain the sentence vector representation, I run a parser on the sentence, and apply RNNs recursively at each node of the parse tree, using phrase vectors in place of $\mathbf{w}_i$, or $\mathbf{w}_j$ whenever necessary.

3.2.3 Distance Metric and Similarity Learning

The notion of the metric plays an important role in machine learning [3]. There exists a line of research on learning a measure of distance or similarity between data that is suitable for problems at hand.

For example, the distance metric learning [82] is to optimize the Mahalanobis distance: $D_{\mathbf{M}}(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^\top \mathbf{M}(\mathbf{x} - \mathbf{x}')$, and the similarity learning [56] is to optimize the inner product: $K_M(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{M}\mathbf{x}'$, where $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{x}' \in \mathbb{R}^d$ are feature vectors, and $\mathbf{M} \in \mathbb{R}^{n \times n}$ is a transformation matrix to learn. Here, by decomposing positive semi-definite matrix $\mathbf{M} = \mathbf{W}^\top \mathbf{W}$, I can reformulate the above equations as follows:

$$D_{\mathbf{M}}(\mathbf{x}, \mathbf{x}') = ||\mathbf{W}\mathbf{x} - \mathbf{W}\mathbf{x}'||^2,$$
$$K_{\mathbf{M}}(\mathbf{x}, \mathbf{x}') = (\mathbf{W}\mathbf{x})^\top (\mathbf{W}\mathbf{x}').$$

From this perspective, distance metric and similarity learning are equivalent to learning the linear projection $\mathbf{W}$, which maps $\mathbf{x}$ and $\mathbf{x}'$ into new representations. Furthermore, these learning techniques can be extended with kernel methods [36]. The theory describes that the kernel function K implicitly maps original input data set $\mathcal{X}$ to a high-dimensional (possibly infinite) reproducing kernel Hilbert space (RHKS) $\mathcal{H}$ through $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{H}$ and computes the inner product therein; i.e., $K(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}'),\ \mathbf{x}, \mathbf{x}' \in \mathcal{X}$. The motivation of kernelization is to obtain a good high-dimensional space for solving problems while maintaining computational tractability.

3.2.4 Neural Networks for Paired Data

Recently, there has been growing interest in applying neural networks to various paired data. For example, [27] construct a neural network for machine translation with

parallel training data. This model represents a pair of source and target phrases in a common low-dimensional space, and their translation score is computed and optimized using the cosine similarity between the pair of vectors. [37] also construct a neural network for multi-modal representations by integrating linguistic vectors with visual concept vectors. These methods allow us to learn richer meaning representations by optimizing the Euclidean distance and cosine similarity for paired data.

Note that, from the distance metric and similarity learning perspective as described in the above, these methods are equivalent to learning a non-linear transformation function $||f(\mathbf{x}) - f(\mathbf{x}')||^2$ and $f(\mathbf{x})^\top f(\mathbf{x}')$, where f is a non-linear function, and $\mathbf{x}$ and $\mathbf{x}'$ are feature vectors of paired data. Typically, these functions are optimized with neural networks and this architecture is called deep metric or similarity learning [81].

3.3 Method

In this section, I introduce a method of non-linear similarity learning for semantic compositionality and discuss the motivation behind it. Consider a training dataset that contains N pairs of tuples $\{((S_i, S_i'), t_i)\}_{i=1}^N$, where (S_i, S_i') is a sentence pair and $t_i \in [0.0, 1.0]$ is the normalized similarity score of (S_i, S_i'). The goal of this task is to predict the similarity scores for sentence pairs. My aim is to design a model, which jointly learns low-dimensional word representations, compositional functions based on neural networks, and the semantic similarity of the compositional sentence representations in a high-dimensional space via kernels (Figure 3.1). My approach is motivated by the following hypothesis:

> For similarity computations that require composition (e.g., sentence similarity), it is necessary to map the compositional representation to a space with higher representational power than the original low-dimensional space of the components (e.g., words).

Under this hypothesis, the goal of compositional semantics is to model how high-dimensional sentence similarity is computed from low-dimensional word representations. This goal differs from that of the current neural network-based models, which attempt to embed and measure the similarity of sentences in the same low-dimensional space of words. My motivation for kernelization in compositionality is to obtain a good high-dimensional space for representing richer meanings of sentences while maintaining computational tractability.

3.3.1 Sentence Vector with LSTM

Given a sentence $S = w_1, w_2, \cdots, w_L$, where w_i is the i-th word and L is the sentence length, I transform the all words w to word vector representations $\mathbf{x} \in \mathbb{R}^d$ such as GloVe described in the previous section. I represent the obtained word vector sequence as

$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots, \mathbf{x}_{L-1}, \mathbf{x}_L,$$

where $\mathbf{x}_i$ is the i-th word vector in the sentence S. This is a input vector sequence of my models.

Average model (BOWs): I first compute a vector representation $\mathbf{s} \in \mathbb{R}^d$ of a sentence S with the simplest model, i.e., average of the word vectors in the sentence as follows:

$$\mathbf{s} = \frac{1}{L} \sum_{i=1}^{L} \mathbf{x}_i.$$

This is a bag-of-words (BOWs) approach in which the sequence and tree structure of the sentence is not considered. While this is the simplest BOWs model, it is known that it can achieve reasonable performance in some tasks [67].

LSTM: The proposed LSTM reads the above word vector sequence and compose a sentence representation from the hidden vectors of words. Given a word vector sequence $\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_L$ as an input, I apply the LSTM function given by Equation (2.8) to obtain a hidden vector sequence $\mathbf{h}_1, \mathbf{h}_2, \cdots \mathbf{h}_L$, where $\mathbf{h}_i \in \mathbb{R}^d$ is the i-th hidden vector of $\mathbf{x}_i$. The computation of the LSTM can be stated as follows:

$$\langle \mathbf{h}_i, \mathbf{c}_i \rangle = \text{lstm}(\mathbf{x}_i, \mathbf{h}_{i-1}, \mathbf{c}_{i-1}).$$

With the obtained hidden vector sequence $\mathbf{h}_1, \mathbf{h}_2, \cdots \mathbf{h}_L$, I compute an output vector, i.e., sentence vector $\mathbf{s} \in \mathbb{R}^d$ as follows:

$$\mathbf{s} = \frac{1}{L} \sum_{i=1}^{L} \mathbf{h}_i.$$

Note that, while I can consider the use of last hidden vector as a sentence representation, the performance was lower than this average model with LSTM in my experiments.

3.3.2 Non-linear Similarity Learning with Kernel Functions

In this section, I present a non-linear similarity learning for sentence representations with kernel functions. In the following, the normalized inner product, i.e., cosine similarity, is employed as the most basic kernel function $\mathcal{K}$, which is computed between the sentence vectors $\mathbf{s}$ and $\mathbf{s}'$ obtained by using average model or LSTM describe in the previous section. I represent the cosine kernel function as follows:

$$\mathcal{K}_{cos}(\mathbf{s}, \mathbf{s}') = \frac{\mathbf{s}^\top \mathbf{s}'}{||\mathbf{s}||\ ||\mathbf{s}'||}.$$

Note that all kernel functions $\mathcal{K}$, including the non-linear ones I introduce below, are normalized for simplicity and preventing the kernel value from growing out of control during learning process. When $\mathcal{K}$ is a non-linear kernel, the normalized kernel $\tilde{\mathcal{K}}$ is represented as follows:

$$\tilde{\mathcal{K}}(\mathbf{s}, \mathbf{s}') = \frac{\mathcal{K}(\mathbf{s}, \mathbf{s}')}{\sqrt{\mathcal{K}(\mathbf{s}, \mathbf{s})}\ \sqrt{\mathcal{K}(\mathbf{s}', \mathbf{s}')}}.$$

In addition, two non-linear kernels are employed in this section, normalized polynomial kernel $\mathcal{K}_{poly}$ and Gaussian kernel $\mathcal{K}_{rbf}$[1], defined as follows:

$$\tilde{\mathcal{K}}_{poly}(\mathbf{s}, \mathbf{s}') = \left(\frac{c + \mathcal{K}_{cos}(\mathbf{s}, \mathbf{s}')}{c + 1} \right)^p,$$

$$\tilde{\mathcal{K}}_{rbf}(\mathbf{s}, \mathbf{s}') = \exp\left(\frac{\mathcal{K}_{cos}(\mathbf{s}, \mathbf{s}') - 1}{\sigma^2} \right).$$

Now, the training objective is to minimize $L(\Theta)$ for training dataset $\{((S_i, S_i'), t_i)\}_{i=1}^N$ as follows:

$$\mathcal{L}(\Theta) = \frac{1}{2} \sum_{i=1}^{N} \left\{ t_i - \tilde{\mathcal{K}}(\mathbf{s}_i, \mathbf{s}_i') \right\}^2 + \frac{\lambda}{2} ||\Theta||_2^2,$$

where Θ is the set of all parameters: the weight matrices and bias vectors in LSTM, all word vector representations, and parameters in kernel functions such as c in the polynomial and σ in Gaussian kernel. Thus, I can obtain the non-linear similarity between two sentences, and the computation and learning are done inexpensively through a kernel function in the implicit high-dimensional space.

[1]Note that Gaussian kernel based on Euclidean distance is $\mathcal{K}_{rbf}(\mathbf{s}, \mathbf{s}') = \exp\left(-||\mathbf{s} - \mathbf{s}'||^2 / 2\sigma^2\right)$. Using $||\mathbf{s} - \mathbf{s}'||^2 = \mathbf{s}^\top \mathbf{s} + \mathbf{s}'^\top \mathbf{s}' - 2\mathbf{s}^\top \mathbf{s}'$ and substituting $\mathcal{K}_{cos}(\mathbf{s}, \mathbf{s}')$ in place of the inner product, I obtain $||\mathbf{s} - \mathbf{s}'||^2 = 2 - 2\mathcal{K}_{cos}(\mathbf{s}, \mathbf{s}')$.

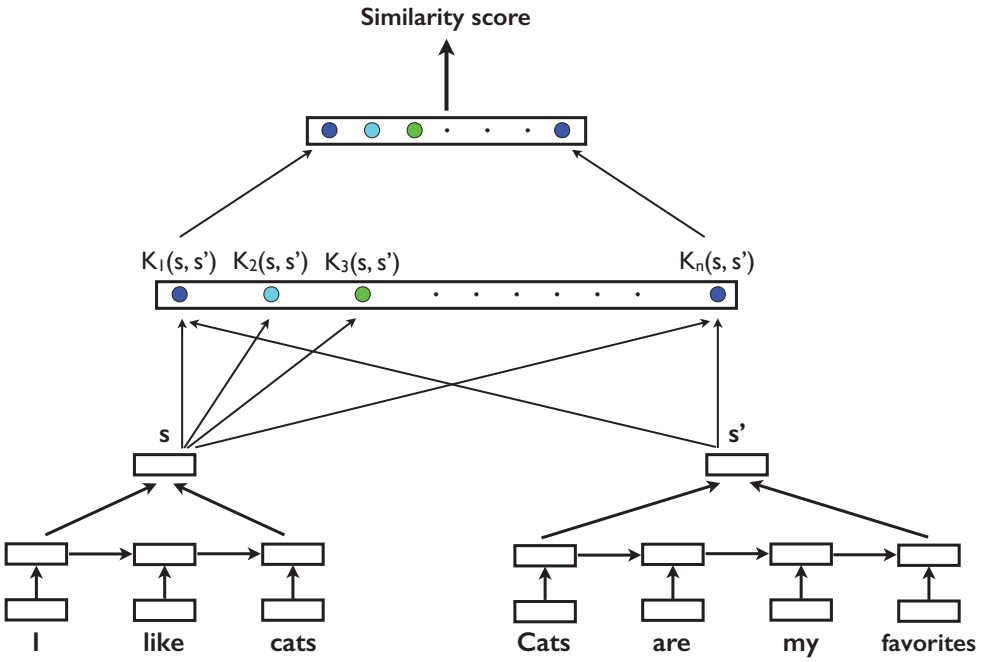

Figure 3.3: Deep multiple kernel learning between two sentences.

3.3.3 Deep Multiple Kernel Learning

Since various similarity values with various kernel functions (i.e., multiple kernels [86, 29, 70]) can be considered, I consider the various kernel values as a vector. I call this *multiple kernel vector* and this is represented as follows:

$$\mathbf{k} = [\mathcal{K}_1(\mathbf{s}, \mathbf{s}'), \mathcal{K}_2(\mathbf{s}, \mathbf{s}'), \cdots, \mathcal{K}_n(\mathbf{s}, \mathbf{s}')],$$

where $\mathcal{K}_i$ is the i-th kernel function and n is the number of kernel functions to be considered. Then I can use the multiple kernel vector $\mathbf{k} \in \mathbb{R}^n$ as a input vector of a deep neural network for the similarity learning as follows:

$$\begin{aligned}
\mathbf{h}^{(1)} &= f(\mathbf{W}^{(1)}\mathbf{k} + \mathbf{b}^{(1)}) \\
\mathbf{h}^{(2)} &= f(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \\
&\cdots \\
y &= f(\mathbf{W}^{(\ell)}\mathbf{h}^{(\ell)} + \mathbf{b}^{(\ell)}),
\end{aligned}$$

where f is the non-linear activation function such as ReLU. Note that the size of final weight matrix is $\mathbf{W}^{(\ell)} \in \mathbb{R}^{1\times d}$ and the objective function is as follows:

$$\mathcal{L}(\Theta) = \sum_{i=1}^{N} \frac{1}{2}\{t_i - y_i\}^2 + \frac{\lambda}{2}||\Theta||_2^2.$$

Figure 3.3 shows the deep multiple kernel learning for semantic similarity between sentences.

3.4 Related Work

Successful approaches for the semantic relatedness task in SemEval 2014 (describe in next section as a evaluation task of my methods) combine several kinds of features widely used in NLP tasks such as surface form overlap, lexical distance, and WordNet. Some of the submitted systems show that purely compositional models reach performance above Pearson correlation r about 0.70 and these scores are lower than the one of the best purely non-compositional system which reaches r over 0.80 [44]. Note that all top systems such as [85, 6, 34, 40] are heavily feature engineered and use external resources.

The most common way to build a sentence representation from words is to simply average the word vectors. While this bag-of-words (BOWs) model can yield reasonable performance in some tasks, it cannot distinguish the word order (i.e., sequence) and structure of the sentence. Unlike the BOWs model, recursive neural networks [68] combine word vectors in constituency trees of sentences, which have potentially many hidden vectors. While the sentence vectors work well on many tasks, they must also take into account the syntactic structure of sentence in detail [66]. Unlike the model with constituency trees, dependency tree-based models [67] naturally focus on the action and agents in a sentence. They are better able to abstract from the details of word order and syntactic expression using dependency relations. On the other hand, in LSTM models [33], while the standard composes its hidden state from the input at the current time step and the hidden state of the unit in the previous time step, the constituency and dependency tree-LSTM [73] composes its state from an input vector and the hidden states of arbitrarily many child units.

These deep learning and my approach have several important similarities and differences in terms of non-linearity and high-dimensionality for modeling and learning compositional semantics in a vector space. Previous models based on only neural

Sentence 1	Sentence 2	Sim
A man is making music with a flute.	A man is playing flute.	4.8
Many people are skating in an ice park.	An ice skating rink placed outdoors is full of people.	4.5
A man is playing a guitar.	A man is opening a box with a knife.	1.2

Figure 3.4: Some examples, the sentence pair and similarity score that is determined by annotators, in the STS dataset.

networks use a composition function and apply these recursively inside a parse tree to compute a sentence vector. Note that this leads us to represent sentence vector in a low-dimensional space (e.g., the same dimensionality as words). In contrast, my method is not subjected to dimensional restraints, which come from modeling with only neural networks, and can flexibly represent and properly learn the sentence meaning with kernels.

3.5 Experiments and Results

3.5.1 Dataset

I evaluate the proposed methods in the SemEval 2014 semantic relatedness task, which uses the sentences involving compositional knowledge (SICK) dataset[2] [44]. This task is to predict the relatedness (or similarity) of two sentences as judged by human annotators on a continuous scale from 1.0 (indicating that the two sentences are completely dissimilar) to 5.0 (indicating that the two sentences are very similar). Figure 3.4 shows some examples in STS dataset. The dataset consists of 9927 sentence pairs in a 4500/500/4927 train/development/test split. My proposed methods are evaluated by computing the Pearson's r correlation, Spearman's ρ correlation, and mean squared error (MSE) between the gold similarity scores and scores predicted with the models.

[2] http://alt.qcri.org/semeval2014/task1/

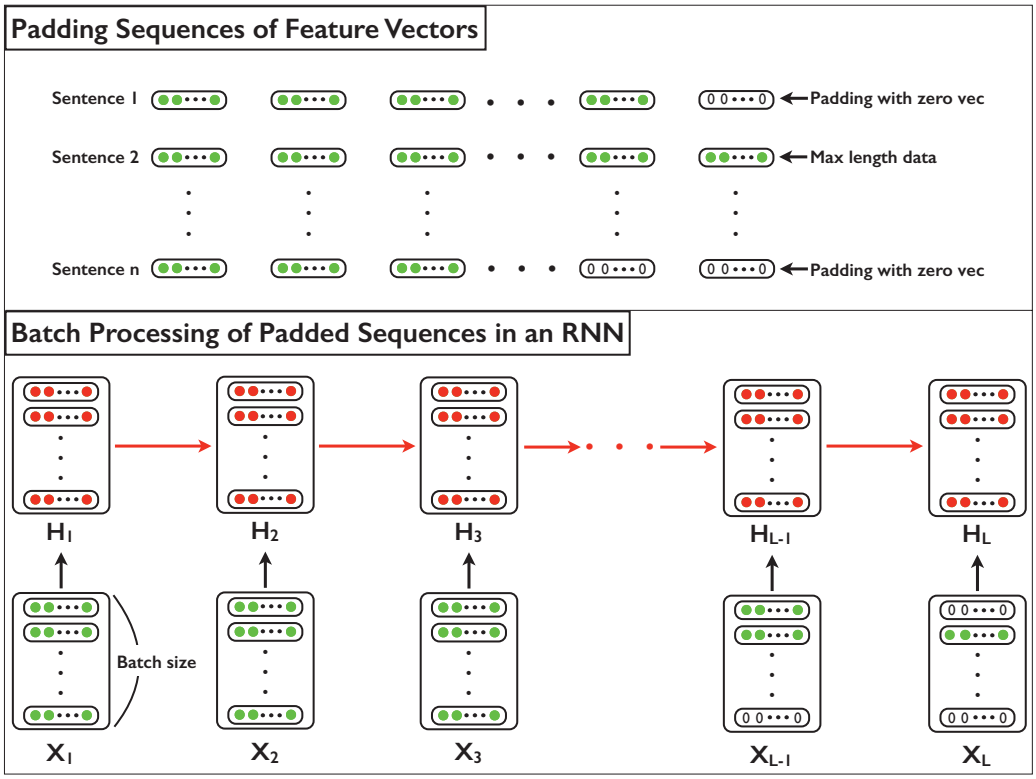

Figure 3.5: Padding sequences of word vectors considering the max length data in a dataset and batch processing of the padded sequences in an RNN architecture. This is one of the most important techniques for efficient implementation of deep neural networks. In this figure, $\mathbf{X}_i$ is the matrix that consists of the i-th input vectors (i.e., i-th word vectors of sentences in a batch), and $\mathbf{H}_i$ is the matrix that consists of the i-th hidden vectors computed with an RNN.

3.5.2 Implementation

I used 50 or 300 dimensional word representations with several different initializations (i.e., pre-trained) as follows: (1) random initialization within $(-0.1, 0.1)$, (2) LSA [78], (3) NLM[3], (4) word2vec[4], and (5) GloVe[5]. In addition, I initialized the parameters in kernels as $c = 1.0$ in polynomial and $\sigma = 1.0$ in RBF. For the multiple kearnel vector, I create the set of kernels as follows: (1) Gaussian kernels with 13 different widths $\{2^{-6}, 2^{-5}, 2^{-4}, \cdots, 2^{6}\}$ and (2) polynomial kernels of degree 1 to 7.

[3] http://ronan.collobert.com/senna/
[4] https://code.google.com/archive/p/word2vec/
[5] http://nlp.stanford.edu/projects/glove/

Method	r	ρ	MSE
Cosine (BOWs)	0.7588	0.7391	0.4830
Polynomial kernel (BOWs)	0.8198	0.7710	0.3105
Gaussian kernel (BOWs)	0.8108	0.7699	0.3269
Cosine (LSTM)	0.7616	0.7723	0.4209
Polynomial kernel (LSTM)	0.8415	0.7800	0.3009
Gaussian kernel (LSTM)	0.8333	0.7801	0.2950
Deep multiple kernels (LSTM)	**0.8501**	**0.8011**	**0.2786**

Table 3.1: Correlation scores with various sentence representations and kernel functions.

Thus, the dimensionality of the multiple kearnel vector is 20. The number of layer on top of the multiple kearnel vector is 3-layers, which the dimensionality of the all layer is the same as input layer.

I implemented my LSTMs using Chainer[6] [75] and the optimization method was ADAM [38], which is one of the most effective stochastic gradient descent (SGD)-based algorithms for training deep neural networks. I set ADAM with a first momentum coefficient of 0.9 and a second momentum coefficient of 0.999, which were the recommended configurations in [38]. The LSTM training details are as follows:

- Batch size: 8, 16, 32, and 64.
- L2-regularization strength λ: 1e-6, 1e-7, 1e-8, and 1e-9.

These hyper-parameters were tuned on the development set. Note that my LSTM implementation is based on a batch processing for padded sequential data. This technique is important for efficient implementation when using an RNN architecture (Figure 3.5).

3.5.3 Compared Methods

I compare my methods against the top systems for the SemEval 2014 semantic relatedness task: ECNU, the meaning factory, UNAL-NLP, and Illinois-LH [85, 6, 34, 40],

[6] `http://chainer.org/`

Method	r	ρ	MSE
Illinois-LH_run1 [40]	0.7993	0.7538	0.3692
UNAL-NLP_run1 [34]	0.8043	0.7458	0.3593
Meaning_Factory_run1 [6]	0.8268	0.7722	0.3224
ECNU_run1 [85]	0.8280	0.7689	0.3250
DT-RNN [67]	0.7923	0.7319	0.3822
SDT-RNN [67]	0.7900	0.7304	0.3848
Constituency Tree LSTM [73]	**0.8582**	**0.7966**	**0.2734**
Dependency Tree LSTM [73]	**0.8676**	**0.8083**	**0.2532**
Deep multiple kernels (LSTM)	**0.8501**	**0.8011**	**0.2786**

Table 3.2: Comparison to competitive performers. Results are grouped as follows: SemEval 2014 submissions, RNN models, sequential and structural LSTM and my best model of deep multiple kernels. My best result outperforms the four of the top systems submitted to SemEval and close to results of tree-LSTM moddels.

which use a various types of handicrafted features. I also compare with models based on recursive neural networks and tree LSTM [67, 73], which use deep learning for computing and learning sentence representations in only a low-dimensional space. These are described in the previous section.

3.5.4 Main Result

Correlateion and MSE Scores: Table 3.1 shows r, ρ, and MSE for different sentence representations and kernel functions. I found that both Pearson's r and Spearman's ρ are higher with two non-linear (polynomial and Gaussian) kernels than the linear (cosine) kernel by a large margin. In particular, the model with LSTM and polynomial kernel achieved the best result among single kernel models. In addition, the deep multiple kernel model achieved the best performance in my proposed methods. These results suggest that the similarity learning in a kernel-induced high-dimensional space is effective, and the kernel space allows us to obtain new word vectors and compositional functions which are suitable for sentence representations.

While the LSTM achieved higher performance than BOWs model, the results of

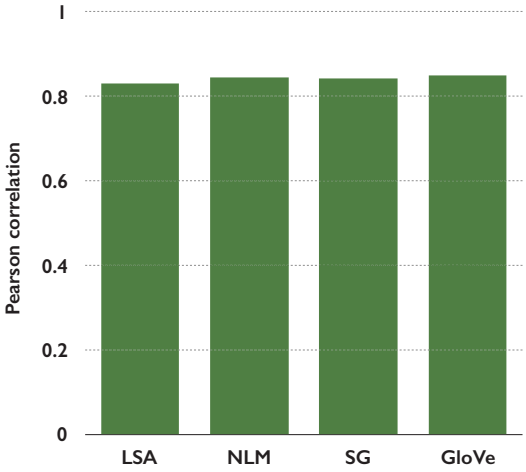

Figure 3.6: Pearson correlations with various word representations. The dimensionality of all word representations is 50 the exception of word2vec of SG (300 dim).

BOWs model with non-linear kernels are also high even though the model ignores the sequence structures. This suggests that the high-dimensional kernel space is sufficient to capture sentence meanings without modeling the structures in details. In addition, the result implies that the co-occurrence information of words in a sentence is important to capture the meaning.

Comparison to Competitive Performers: Table 3.2 shows the comparison to competitive performers. The result can be summarized as follow. First, my model outperformed four of the top systems submitted to SICK in SemEval 2014. Note that these four are heavily feature engineered systems, whereas our approach is mainly dependent on learning of word representations, and does not require a large number of features and external resources. Second, my correlation scores and MSE are lower than DT-RNN and SDT-RNN [67] models. Third, in terms of the best correlation score, my model is the competitive with tree-LSTM models. These tree-LSTM models also compute a sentence representation in a low-dimensional space in the RNNs fashion. This result also shows the effectiveness of my model.

3.5.5 Analysis

Initialization of Word Representations: I further analyze the influence of using

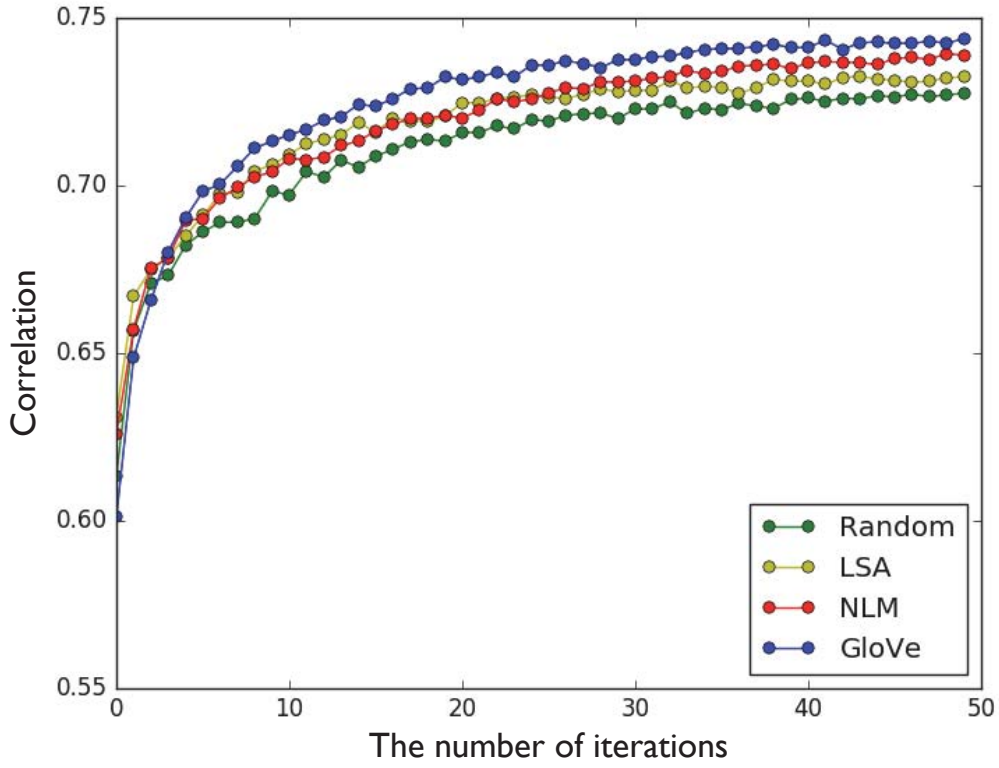

Figure 3.7: Learning curve with various word representations. The dimensionality of all word representations is 50.

pre-trained word representations for initialization. Figures 3.6 and 3.7 show that all word representations achieved similar results. In addition, I also obtain, after a sufficient number of iterations, high performance with random initialization. This suggests that the representation learning with kernels is still able to achieve comparable accuracy without the help of pre-training, and I need to re-train representations for solving problems.

The Effect of Kernel Functions: In Table 3.8, although two non-linear kernels outperforms cosine, the correlation score of Gaussian kernel is lower than cosine early in the training process. However, the Gaussian kernel eventually achieves a higher correlation score than cosine by increasing the training time. On the other hand, polynomial kernel keeps the learning curve stable from early in the training process. This suggests that the Gaussian kernel is sensitive to the hyper-parameter σ and require the careful training of σ.

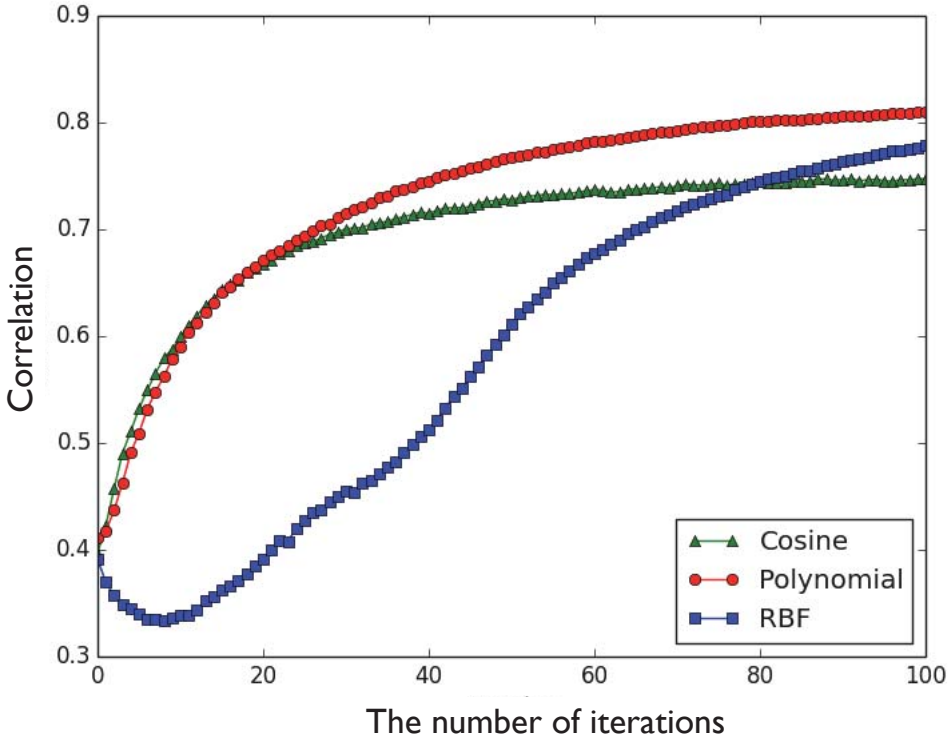

Figure 3.8: Learning curve with various kernel functions.

3.6 Conclusion

In this chapter, I have proposed a new method of non-linear similarity learning for semantic compositionality. Instead of relying on only neural network-based operating in a low-dimensional space, I train kernel functions that allows us to measure the semantic similarity in a high-dimensional space. In the task of predicting the relatedness of two sentences, my method have outperformed linear baselines, feature engineering approaches, and achieved competitive results with recent deep learning models.

Chapter 4

Protein Fold Recognition

4.1 Introduction

Predicting the 3D structure of a protein from its amino acid sequence using computational methods is an important challenge in bioinformatics. However, it is difficult to directly predict the 3D structure from the sequence. Therefore, classifying a protein into one of the *folds*, which are pre-defined structural labels in protein databases such as SCOPe[1] and CATH[2], is used as an intermediate step for predicting the 3D structure. This classification task is called *protein fold recognition* (Figure 4.1). Although a protein fold can be accurately predicted when the homologous templates are identified, it remains a challenging problem when the similarities between the sequence and templates are low. Even if two proteins have low sequence homology, they can have the same fold.

The process of protein fold recognition is mainly divided into two steps: (1) extracting features and (2) learning classifiers. The features are generally extracted from syntactical, physicochemical, and evolutionary information of proteins. Syntactical features are based on the amino acid occurrence [72], the pairwise frequency of amino acids separated by one or adjacent residues [28], and the combination of these [83]. Physicochemical features are based on attributes such as hydrophobicity, secondary structure, polarity, polarizability, and van der Waals volume [21]. In addition to these, various attributes have been used such as solvent accessibility [84], a strategy to select them has been explored [65], and their impact has been examined [16]. Evolutionary features are based on sequence profiles, which are obtained by running PSI-

[1] `http://scop.berkeley.edu/`
[2] `http://www.cathdb.info/`

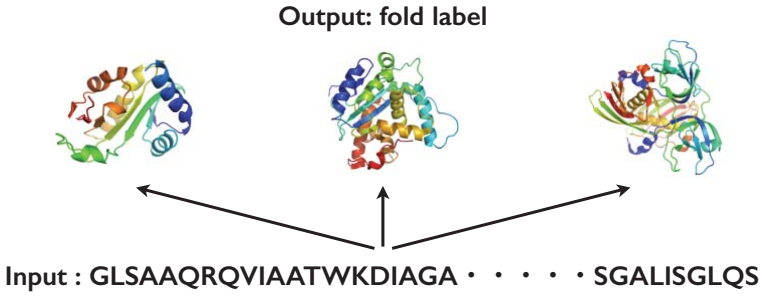

Figure 4.1: An input and output of the protein fold recognition task. The above protein figures are provided from `https://nanohub.org/resources/contactmaps`.

BLAST [1]. The sequence profile is the probability score of amino acid occurring at a specific position in the sequence. This is also called position specific scoring matrix (PSSM) and computed with multiple sequence alignment for all proteins in the database. As a result, each amino acid feature is represented with a 20-dimensional vector, in which the elements are the position specific scores of amino acid occurring probability. Among these three features, evolutionary features have shown to be useful for protein fold recognition and their use can improve its accuracy by a large margin [64, 50, 42, 57, 43]. For the classifiers, several types have been used, such as hidden Markov models (HMMs) [8], support vector machines (SVMs) [20], and neural networks (NNs) [10]. Among these classifiers, SVM-based methods have been widely used and shown promising results [57, 43]. However, one major issue is that SVMs and feed-forward NNs require a fixed-length feature vector as an input, whereas protein sequences have arbitrary lengths. This problem remains a fundamental and difficult challenge in protein fold recognition using machine learning methods.

Recently, deep neural networks have achieved excellent performance in solving difficult problems such as visual object recognition [39] and speech recognition [31]. In particular, for sequential problems in NLP such as machine translation [71], RNNs with LSTM units [33] have re-emerged as a popular architecture. This is mostly owing to the capability of this architecture in learning to map a sequence of arbitrary length into a fixed-dimensional vector representation and capturing long-term dependencies in the sequence. Indeed, LSTMs have also been applied to problems in bioinformatics such as homology detection [32].

In this thesis, I show that the LSTM architecture can (1) learn to map a protein sequence of arbitrary length into a fixed-dimensional vector representation, (2) capture the long-term dependencies of amino acids in the sequence, and then (3) correctly

classify the protein fold. The idea is to use an LSTM to read a sequence of feature vectors of amino acids and output a fixed length vector to be used as an input to a protein fold classifier. In more detail, the whole of my method is as follows:

1. For each amino acid in the protein sequence, I assign a feature vector that consists of a sequence profile, secondary structure, and solvent accessibility by running PSI-BLAST [1], SSpro [54], and ACCpro [53] (chapter 4.3.1). As another feature, I attempt to use the vector representations of amino acids obtained with word representation learning methods in NLP [45, 52] (chapter 4.3.2).

2. To the feature vector sequence, I apply an RNN with LSTM units (chapter 4.3.3). The LSTM (1) reads the feature vectors one by one, (2) computes the hidden vectors, and (3) computes the sum of hidden vectors. This fixed-length vector is then input to a softmax classifier (i.e., logistic regression) of protein folds.

3. However, since proteins have a univariate direction from beginning to end, instead of vanilla LSTMs I use bi-directional LSTM architectures that can consider both directions along the protein sequence. In particular, I propose the use of two architectures: concatenated [30] and stacked [51] bi-directional LSTMs. Fig 4.2 shows the two bi-directional LSTM architectures for protein fold recognition.

My contributions are three-fold:

1. To the best of my knowledge, this is the first work that attempts to use LSTMs for protein fold recognition with biological feature sequence as an input.

2. On the benchmark dataset called EDD [20], the proposed methods outperformed existing methods with various features and machine learning techniques. In particular, compared to existing methods such as using a large number of (over 500) physicochemical attributes in addition to syntactical and evolutionary features [57] and the state-of-the-art method with SVM [43], my stacked bi-directional LSTM achieved higher performance despite the use of a minimal set of features and little tuning of hyper-parameters in the neural network.

3. I provided analyses of the differences between the concatenated and stacked architectures, the impact of hyper-parameters, protein length, and word representation learning. In particular, I showed that the stacked architecture achieved excellent performance compared to the concatenated architecture. In addition, the LSTMs did not suffer on very long protein sequences. LSTM is known to

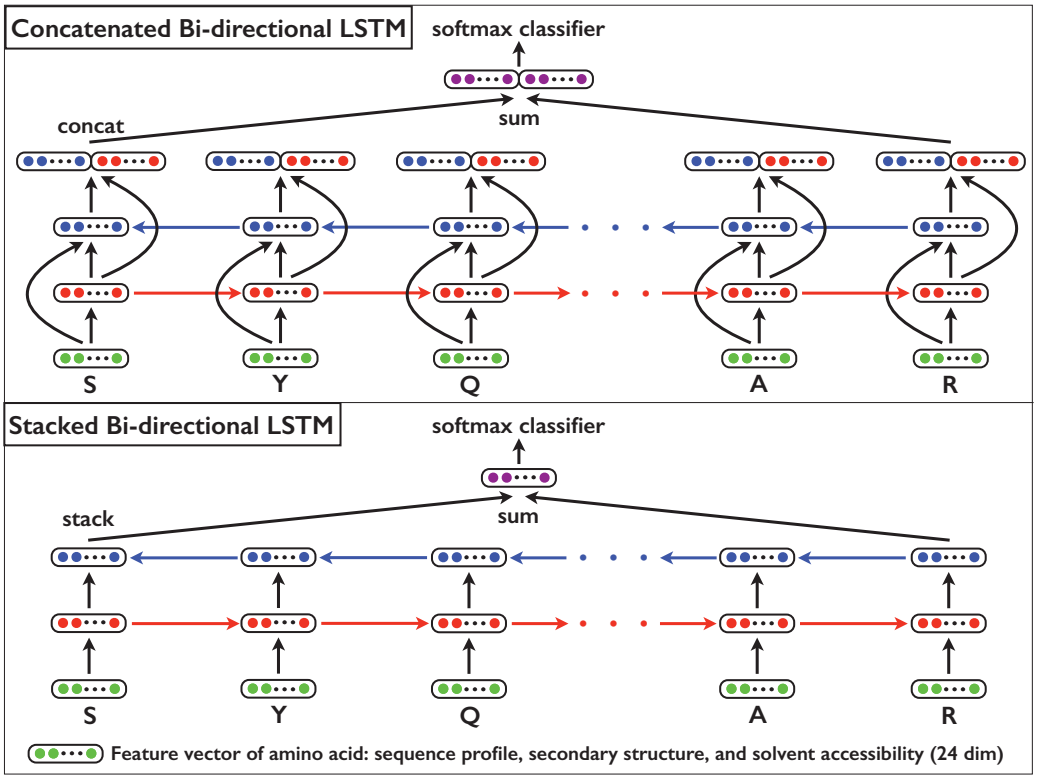

Figure 4.2: The concatenated and stacked bi-directional LSTM architectures for protein fold recognition. Each amino acid feature vector in a protein sequence consists of a sequence profile, secondary structure, and solvent accessibility. In this paper, we propose the use of two architectures: concatenated and stacked bi-directional long short-term memory. The concatenated architecture [30] computes two hidden layers, one for each of direction, and then concatenates the hidden layers. On the other hand, the stacked architecture assumes a hidden layer as an input of the next layer [51].

work well on sentences and other natural language sequences, but protein sequences are typically much longer. Despite this difference, our results show that LSTMs work quite well on very long protein sequences.

4.2 Related Work

As described in chapter 4.1, features for protein fold recognition combine syntactical, physicochemical, and evolutionary information of amino acids. Dubchak et al. [21] suggested the combination of syntactical and physicochemical features that con-

sist of amino acid composition (AAC) as syntactical information (20 features) and the following five physicochemical attributes: hydrophobicity (H), predicted secondary structure based on the normalized frequency of α-helix (X), polarity (P), polarizability (Z), and van der Waals volume (V). The set of these five attributes is represented as "HXPZV," which consists of 105 features, and is widely used as a basic physicochemical feature set.

On the other hand, Taguchi and Gromiha [72] proposed features that are based on the amino acid occurrence as a syntactical feature and argued that only syntactical features should be considered because physicochemical features have no important information for protein fold recognition. In addition, Ghanty and Pal [28] introduced features that are based on the pairwise frequency of amino acids separated by one residue (called PF1) and pairwise frequency of adjacent residues (called PF2). Furthermore, Yang et al. [83] used PF, the concatenated feature of PF1 and PF2. Because PF1 and PF2 consist of 400 (20×20) features, PF consists of 800 features. While these syntactical features have been shown to be useful, other physicochemical attributes have also been proposed. A strategy to select 30 attributes has been explored [65] and the impact of 55 attributes has been examined [16].

Recently, evolutionary information based on a sequence profile (or called PSSM describe in the previous section) obtained by running PSI-BLAST [1] has been shown to be useful for protein fold recognition. The simple use of the sequence profile can improve its accuracy by a large margin compared to the above syntactical and physicochemical features. For example, Sharma et al. [64] used mono-gram and bi-gram features extracted from the sequence profile. In addition, Paliwal et al. [50] used tri-gram features and combined them with syntactical features. Furthermore, Lyons et al. [42] used an alignment method using dynamic programming (DP) to extract evolutionary features. Very recently, Raicar et al. [57] proposed a large feature set consisting of 544 physicochemical attributes in addition to existing syntactical and evolutionary information, and Lyons et al. [43] applied HMM-HMM alignment of protein sequences to extract the profile HMM (PHMM) matrix and computed the distance between respective PHMM matrices using DP (PHMM-DP). These two methods achieved over 90% accuracy in the benchmark dataset called EDD [20]. In these methods, SVM-based classifiers were mainly used.

4.3 Method

In this section, I first describe the evolutionary and physicochemical features of amino acids in a protein sequence, which are used as inputs for protein fold recognition system (chapter 4.3.1). Second, as another feature, I describe the definition of words in protein sequences, creation of a large protein corpus for word representation learning, and application of word2vec [45] and GloVe [52] to this corpus (chapter 4.3.2). Third, I present the use of two LSTMs for protein fold recognition with concatenated and stacked bi-directional architectures (chapter 4.3.3), followed by the explanation of the training and re-training procedure of the LSTMs and pre-trained word representations (chapter 4.3.4).

4.3.1 Evolutionary and Physicochemical Features

In this thesis, I simply use a minimal set of amino acid features that are commonly used in protein structure prediction tasks such as [22, 17]. This feature set is also used in the next chapter.

Each amino acid feature vector in a protein sequence includes three kinds of information as follows:

1. PSI-BLAST sequence profile (20 real values: an amino acid is represented with a 20-dimensional vector, in which the elements are the position specific scores of amino acid occurring probability as described in chapter 4.1).

2. Secondary structures (three binary values: α-helix, β-strand, or coil).

3. Solvent accessibilities (two binary values: buried or exposed).

The sequence profile is obtained by running PSI-BLAST [1] with an E-value cutoff equal to 0.001 and for four iterations against NCBI's non-redundant version of the protein sequence database NR filtered at 90% sequence identity. The secondary structure and solvent accessibility are predicted by running SSpro [54] and ACCpro [53] from the SCRATCH suite [12]. In this thesis, I represent the feature vector of the i-th amino acid in a protein sequence as $\mathbf{x}_i \in \mathbb{R}^d$ (in the above feature, $d = 24$).

4.3.2 Word Representation Learning in Proteins

As another feature, I introduce word representations in protein sequences. In order to apply word representation learning methods such as SG, CBOW, and GloVe described in the previous chapter to protein sequences, I define a word of proteins as an n-gram of amino acid and split the sequences into overlapping of the n-gram amino acids. Since there are 20 types of amino acids that make up proteins, the total number of possible n-grams is 20^n. In this thesis, to keep the vocabulary size tractable and to avoid low-frequency words in learning representations, I set a moderate small n-gram length of $n = 3$. For example, I can split a sequence of protein into an overlapping 3-gram amino acid sequence as follows: $GLSAA \cdots LQS \rightarrow$ *"-GL"*, *"GLS"*, *"LSA"*, $\cdots$, *"LQS"*, *"QS-"*. With this word definition and splitting, I create a large protein corpus from a large protein database. Then, to this corpus, I simply apply SG, CBOW, and GloVe (Figure 4.3). Note that the above word definition and splitting for protein sequences are common approaches in bioinformatics and similar to previous work such as [19, 26, 50].

Why word representations? In NLP, word representations can capture the meaning and its similarity using the co-occurrence information of words in a corpus [15]. Similarly, it has been pointed out that the co-occurrence information of amino acids in proteins is also important for capturing protein structures [72, 28, 83]. In addition, in most existing methods for protein fold recognition, amino acid features typically include evolutionary and physicochemical information such as hydrophobicity, polarity, and van der Waals volume. While these features are widely used, the feature vectors themselves are fixed during learning process [18, 65, 16]. In contrast, above word representations can be re-trained during learning process with deep neural networks described in the following subsection.

4.3.3 Protein Fold Recognition with LSTMs

Input Vector Sequence: Given a protein sequence $P = a_1, a_2, \cdots, a_L$, where a_i is the i-th amino acid and L is the protein length, I transform the all amino acids to

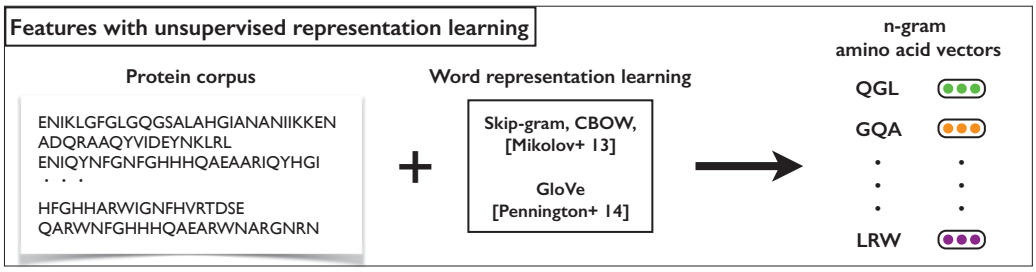

Figure 4.3: The application of word representation learning methods for the protein corpus.

feature vectors[3]. Let the obtained feature vector sequence be

$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots, \mathbf{x}_{L-1}, \mathbf{x}_L.$$

Alternatively, I can also consider a vector sequence whose elements consist of concatenated feature vectors. For example, the vector sequence composed of the concatenation of three contiguous feature vectors is as follows:

$$[\mathbf{0}; \mathbf{x}_1; \mathbf{x}_2], [\mathbf{x}_1; \mathbf{x}_2; \mathbf{x}_3], \cdots, [\mathbf{x}_{L-2}; \mathbf{x}_{L-1}; \mathbf{x}_L], [\mathbf{x}_{L-1}; \mathbf{x}_L; \mathbf{0}], \tag{4.1}$$

where $\mathbf{0}$ is the d-dimensional zero vector for padding and $[\mathbf{x}_i; \mathbf{x}_j; \mathbf{x}_k] \in \mathbb{R}^{3d}$ represents the concatenation of $\mathbf{x}_i$, $\mathbf{x}_j$, and $\mathbf{x}_k$. This allows us to consider surrounding features (e.g., $\mathbf{x}_1$ and $\mathbf{x}_3$) when processing the target feature (e.g., $\mathbf{x}_2$). In this thesis, I represent the feature vector as $\mathbf{x}_{i:i+w-1} = [\mathbf{x}_i; \mathbf{x}_{i+1}; \cdots; \mathbf{x}_{i+w-1}] \in \mathbb{R}^{wd}$, where w is the window size (in the above example, $w = 3$). Such a vector sequence is used as an input sequence for an RNN as described in the following.

LSTMs: The proposed LSTMs read the above feature vector sequence of amino acids (e.g., $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_L$) and learn the parameters (e.g., $\mathbf{W}_i$ and $\mathbf{W}_f$) to classify the protein folds. However, although protein sequences have a univariate direction from beginning to end in contrast to natural language sentences, vanilla LSTMs use only the past sequence and ignore the future sequence. To overcome this limitation, I use bi-directional architectures that can consider both directions along the protein sequence. The original bi-directional LSTM proposed by [30] computes two hidden vectors, one for each direction, and then concatenates the hidden vectors (Figure 4.2). In addition to this concatenated architecture, I attempt to improve the representational power of a

[3]Note that when using word representations, I represent $P = w_1, w_2, \cdots, w_L$ as a word sequence (i.e., an overlapping split sequence with n-gram amino acids), where w_i is the i-th word.

bi-directional LSTM with stacked architectures (Figure 4.2), which assumes a hidden vector as an input of the next layer [62, 23, 51]. In later experiments, I compare the performance of the two architectures.

Concatenated Bi-directional LSTM: Given a feature vector sequence of amino acids $\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_L$ as an input, I apply two LSTM functions given by Equation (2.8) to obtain a hidden vector sequence $\mathbf{h}_1, \mathbf{h}_2, \cdots \mathbf{h}_L$, where $\mathbf{h}_i \in \mathbb{R}^d$ is the hidden vector of $\mathbf{x}_i$. Let $lstm_f$ be a forward LSTM function and $lstm_b$ be a backward LSTM function that are given by Equation (2.8). Note that I distinguish these two functions by subscripts, f and b, because they do not share learning parameters (e.g., $\mathbf{W}_i$ and $\mathbf{W}_f$). Then, the forward and backward computations of the concatenated bi-directional LSTM can be stated as follows:

$$\begin{aligned}
\langle \overrightarrow{\mathbf{c}_i}, \overrightarrow{\mathbf{h}_i} \rangle &= lstm_f(\mathbf{x}_i, \overrightarrow{\mathbf{h}_{i-1}}, \overrightarrow{\mathbf{c}_{i-1}}), \\
\langle \overleftarrow{\mathbf{c}_i}, \overleftarrow{\mathbf{h}_i} \rangle &= lstm_b(\mathbf{x}_i, \overleftarrow{\mathbf{h}_{i+1}}, \overleftarrow{\mathbf{c}_{i+1}}), \\
\mathbf{h}_i &= [\overrightarrow{\mathbf{h}_i}; \overleftarrow{\mathbf{h}_i}],
\end{aligned}$$

where $\mathbf{c}_i$ is the i-th cell layer and $[\overrightarrow{\mathbf{h}_i}; \overleftarrow{\mathbf{h}_i}] \in \mathbb{R}^{2d}$ is the concatenation of $\overrightarrow{\mathbf{h}_i}$ and $\overleftarrow{\mathbf{h}_i}$. Thus, the dimensionality of the obtained hidden vector is $2d$. With the hidden vector sequence $\mathbf{h}_1, \mathbf{h}_2, \cdots \mathbf{h}_L$, I compute an output vector $\mathbf{y} \in \mathbb{R}^k$ as follows:

$$\mathbf{y} = \mathbf{W}_y \sum_{i=1}^{L} \mathbf{h}_i + \mathbf{b}_y, \tag{4.2}$$

where $\mathbf{W}_y \in \mathbb{R}^{k \times 2d}$ is the weight matrix to learn, $\mathbf{b}_y \in \mathbb{R}^k$ is the bias vector to learn, and k is the dimensionality of the output vector $\mathbf{y}$ (i.e., the number of protein folds to classify). Figure 4.2 shows the architecture of the concatenated bi-directional LSTM.

Stacked Bi-directional LSTM: The stacked bi-directional LSTM, which assumes a hidden vector as an input of the next layer, can be stated as follows:

$$\begin{aligned}
\langle \overrightarrow{\mathbf{c}_i}, \overrightarrow{\mathbf{h}_i} \rangle &= lstm_f(\mathbf{x}_i, \overrightarrow{\mathbf{h}_{i-1}}, \overrightarrow{\mathbf{c}_{i-1}}), \\
\langle \mathbf{c}_i, \mathbf{h}_i \rangle &= lstm_b(\overrightarrow{\mathbf{h}_i}, \overleftarrow{\mathbf{h}_{i+1}}, \overleftarrow{\mathbf{c}_{i+1}}).
\end{aligned}$$

Thus, the dimensionality of the obtained hidden vector with stacked architecture is n, which is different from the concatenated architecture. With the hidden vector sequence $\mathbf{h}_1, \mathbf{h}_2, \cdots \mathbf{h}_L$, I compute the output vector $\mathbf{y}$ with Equation (4.2), where $\mathbf{W}_y \in \mathbb{R}^{k \times n}$. Figure 4.2 shows the architecture of the stacked bi-directional LSTM.

4.3.4 Training

Finally, a softmax layer is added on the top of the output vector $\mathbf{y}$ for modeling multi-class (i.e., k-class) probabilities as follows:

$$p_t = \frac{\exp(y_t)}{\sum_{i=1}^{k} \exp(y_i)},$$

where $t = \{1, \cdots, k\}$ is the index of protein fold label and y_i is the i-th element (i.e., scalar) in the output vector $\mathbf{y}$. Given a training dataset $\{(P_i, t_i)\}_{i=1}^{N}$, where P is the protein sequence, t is the index of correct protein fold, and N is the number of training samples, our training objective is to minimize the loss function $\mathcal{L}$, given as the cross-entropy loss plus an L2-regularization term as follows:

$$\mathcal{L}(\Theta) = -\sum_{i=1}^{N} \log p_{t_i} + \frac{\lambda}{2}||\Theta||_2^2,$$

where Θ is the set of all weight matrices and bias vectors to learn in LSTMs described in chapter 2.2, $\mathbf{W}_y$, and $\mathbf{W}_b$ described in this section. p_{t_i} is the probability of t_i and λ is an L2 regularization hyper-parameter. I use a standard backpropagation technique to train Θ. Note that, when using word representations, I include the word representations in Θ and re-train them as learning parameters.

4.4 Experiments and Results

4.4.1 Dataset

I used the EDD dataset [20], which is a benchmark for protein fold recognition. This dataset consists of 3418 proteins, of which any two distinct proteins have less than 40% of sequence identity. The number of protein folds is 27 and it includes labels such as DNA/RNA-binding, immunoglobulin-like, TIM beta/alpha-barrel, and ferredoxin-like, which are based on the major structural classes (i.e., α, β, α/β, and $\alpha+\beta$) in the SCOPe database[4]. I split the EDD dataset 8:1:1 and used these as train/development/test sets.

[4] `http://scop.berkeley.edu/`

4.4.2 Implementation

LSTMs: I implemented my LSTMs using Chainer[5] [75], and the optimization method was ADAM [38], which is one of the most effective stochastic gradient descent (SGD)-based algorithms for training deep neural networks. I set ADAM with a first momentum coefficient of 0.9 and a second momentum coefficient of 0.999, which were the recommended configurations by [38]. I found that the LSTMs do not need careful tuning of hyper-parameters and are easy to train in practice. The training details are as follows:

- Window size w (described in chapter 4.3.3): 1, 3, 5, and 7.
- Batch size: 64, 128, 256, and 512.
- L2-regularization strength λ: 1e-6, 1e-7, 1e-8, and 1e-9.

In my experiments, I compared the results with various window sizes and batch sizes, and examined the effect of these hyper-parameters for protein fold recognition. In particular, it is expected that a larger window size improves the accuracy because it allows more surrounding features to be considered, but the increased input dimensionality slows down training. In contrast, it is expected that a larger batch size improves the training speed, but the accuracy is lower.

Word Representations: I set the word vector dimensionality $d = 100$, window size $\ell = 10$, the number of iterations over a corpus is 10 in CBOW and 100 in SG and GloVe, the number of negative sampling words $K = 5$ (in SG and CBOW), and other hyper-parameters are set to default values in the toolkits of word2vec[6] and GloVe[7]. These word representations are used to initialize our LSTMs but not fixed, they are later re-trained during learning process of LSTMs. I discuss the impact of pre-training and re-training of word representations.

4.4.3 Main Result

Accuracy: Table 4.1 shows the accuracies of existing methods and my methods in the EDD dataset. I observed that concatenated bi-directional LSTM achieved competitive accuracy (92.3%) and stacked bi-directional LSTM outperformed all existing

[5] `http://chainer.org/`
[6] `https://code.google.com/archive/p/word2vec/`
[7] `http://nlp.stanford.edu/projects/glove/`

Existing methods	**Accuracy (%)**
Syntactical features	
Taguchi and Gromiha [72] (Occurrence)	46.9
Ghanty and Pal [28] (PF1)	50.8
Ghanty and Pal [28] (PF2)	49.9
Yang et al. [83] (PF1 and PF2)	55.6
Physicochemical features	
Dubchak et al. [21] (HXPZV)	40.9
Sharma et al. [65] (15 attributes)	51.3
Sharma et al. [65] (30 attributes)	54.6
Dehzangi et al. [16] (55 attributes)	56.7
Evolutionary features	
Sharma et al. [64] (Mono-gram)	76.9
Sharma et al. [64] (Bi-gram)	84.5
Paliwal et al. [50] (Tri-gram)	86.2
Lyons et al. [42] (Alignment method)	90.2
Raicar et al. [57] (+ 544 physicochemical attributes)	90.2
Lyons et al. [43] (PHMM-DP)	92.9
My methods	**Accuracy (%)**
Concatenated bi-directional LSTM	92.3
Stacked bi-directional LSTM	**93.2**

Table 4.1: Accuracies of existing methods and my methods in the EDD dataset.

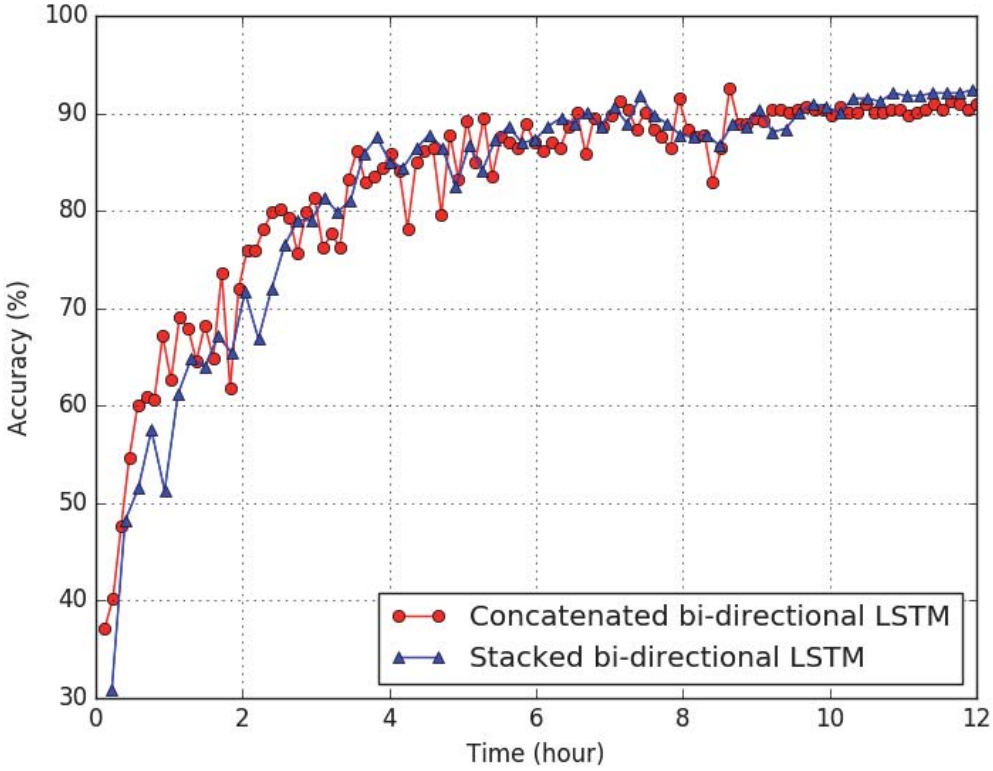

Figure 4.4: Learning curves of time (hour) vs. accuracy (%) for the best performing concatenated and stacked LSTM architectures.

methods (93.2%). The settings of the best performing architectures are as follows: Concat: window size, 5; batch size, 128; and L2-regularization λ, 1e-8; Stack: window size, 7; batch size, 128; and L2-regularization λ, 1e-8. To the best of my knowledge, this is the first result to show that a neural network-based method outperforms existing SVM-based methods for protein fold recognition.

Learning Curves: Time vs. Accuracy: Figure 4.4 shows the learning curves of time (hour) vs. accuracy (%) for the best performing concatenated and stacked LSTM architectures. Note that, in my all experiments, I measured training time with a single thread of the Intel Xeon CPU E5-2699 2.30GHz machine using only one core as a baseline evaluation. Training speed can be significantly improved using all cores of the machine and graphics processing unit (GPU) in practice.

In Figure 4.4, I observe that both LSTMs take about 7 hours to achieve over 90% accuracy. In addition, neither LSTM keeps the learning curves stable early in the training process. However, both eventually achieve high accuracies by increasing the

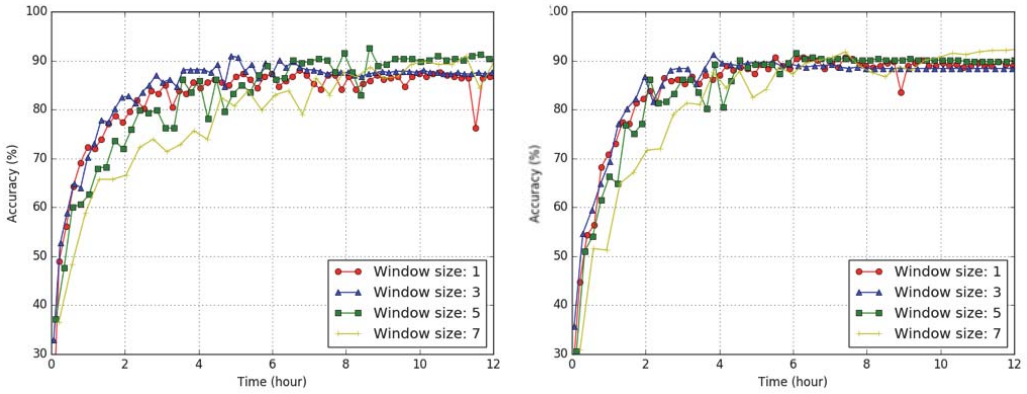

Figure 4.5: Learning curves of time (hour) vs. accuracy (%) by various window sizes. The graph on the left shows the result of concatenated bi-directional LSTM, and that on the right shows the result of stacked bi-directional LSTM. In these figures, the batch size is 128.

training time. Subsequently, the learning curves remain stable later in the training process. These unstable learning curves may be observed due to the great variability of sequence lengths among proteins in the training dataset (discussed in chapter 4.4.4).

Given that it is generally more difficult to achieve high performance in neural networks without careful tuning of the hyper-parameters compared to existing SVM-based methods, I examine the effect of various hyper-parameters of the LSTMs in the following subsection.

4.4.4 Analysis

The Effect of Window Size: Figure 4.5 shows the learning curves of time (hour) vs. accuracy (%) by various window sizes. In Figure 4.5, I observe that the learning speed is slower when the window size is larger (particularly when the window size is 7) because of the high dimensionality of the input vector for an LSTM. However, both LSTMs achieve high accuracies after training time is increased. In the stacked bi-directional LSTM, the influence of the window sizes (1, 3, and 5) vanishes as the training time is increased, and they eventually achieves the same level of accuracy. This suggests that the stacked LSTM is robust for the window size. In contrast, the concatenated LSTM is a little sensitive to the window size.

Incorporating surrounding features did not lead to marked improvement in accuracy; the best accuracy was achieved with the stacked architecture when the window size was 7, but smaller window sizes also achieved competitive accuracies (over 90%). On the

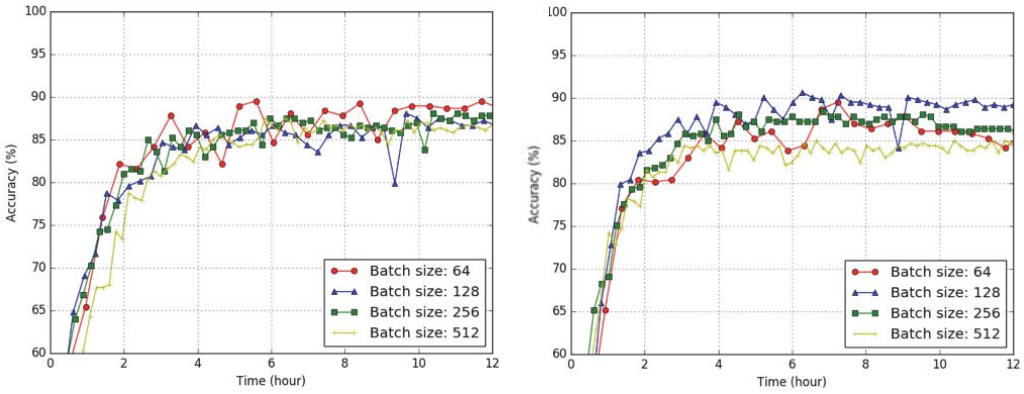

Figure 4.6: Learning curves of time (hour) vs. accuracy (%) by various batch sizes. The graph on the left shows the result of concatenated bi-directional LSTM, and that on the right shows the result of stacked bi-directional LSTM. In these figures, the window size is 1.

basis of this observation, I found that the stacked architecture that reads each feature vector of amino acid one by one in a protein sequence can be learned fast, robustly facilitating high performance.

The Effect of Batch Size: Figure 4.6 shows the learning curves of time (hour) vs. accuracy (%) by various batch sizes. The batch processing and batch size are important for efficient implementation of RNNs.

In Figure 4.6, I observe that the batch size has an insignificant effect on the performance in the concatenated LSTM, and all the accuracies with various batch sizes nearly equal. This suggests that the concatenated LSTM is robust for different batch sizes. However, in stacked LSTM, batch size 128 achieves the best accuracy, and smaller or larger batch sizes degrade the performance. In particular, neither LSTM can keep the learning curve stable when the batch size is small (e.g., 64).

The Effect of Protein Length: One of the most important concerns with LSTMs for proteins is that it may not be able to work well on long sequences. In particular, while LSTMs perform well in NLP, sentences are usually much shorter than protein sequences, which often comprises over 300 amino acids. However, I were surprised to discover that my LSTMs worked well on such long protein sequences, which is shown quantitatively in Figure 4.7 using the learning curves of time (hour) vs. accuracy (%) for various sequence lengths.

In Figure 4.7, I observe that concatenated and stacked LSTMs rapidly learn short

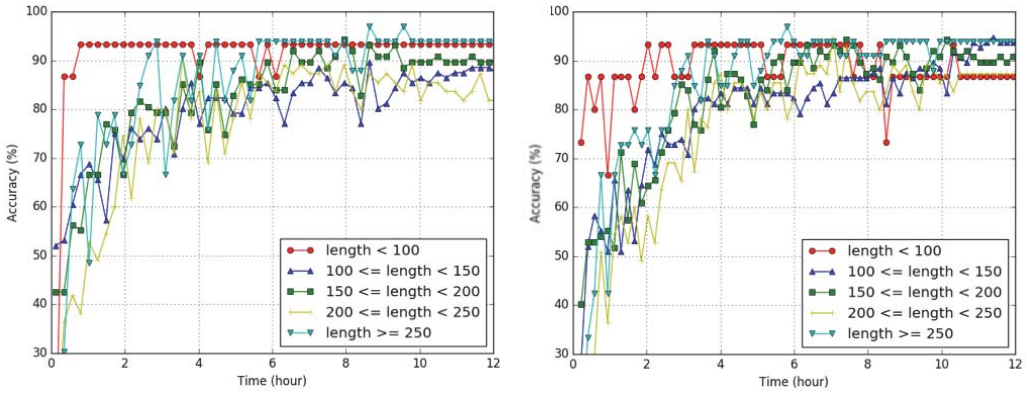

Figure 4.7: Learning curves of time (hour) vs. accuracy (%) by various sequence lengths. The graph on the left shows the result of concatenated bi-directional LSTM, and that on the right shows the result of stacked bi-directional LSTM. In these figures, the window size is 1 and the batch size is 128.

sequences and achieve over 90% accuracy early in the training process. In addition, both LSTMs slowly learn long sequences and achieve nearly 90% accuracy later in the training process. However, the stacked LSTM shows degradation for short sequences by increasing training time. In contrast, concatenated LSTM keeps the accuracy for short sequences stable. These results suggest that the representational power of the stacked architecture is higher and makes it easier to overfit for short sequences. To address the limitation with regard to sequence length, curriculum learning strategies considering data lengths might be needed to be applied [4]. This is another aspect of LSTMs for proteins to be solved in future studies.

The Effect of Word Representations: In Figure 4.8, I found that the performance does not depend on initialization of word representations such as SG, CBOW, and GloVe. The reason is that these representations are similar in terms of having the co-occurrence information of amino acids in proteins.

In Figure 4.8, I also found that re-training of word representations does not always give an improvement. While stacked bi-directional LSTM shows the improved accuracy with the help of re-training representations, any substantial improvement was not observed when concatenated bi-directional LSTM is used. Interestingly, Figure 4.8 shows that even if I use random initialized vectors for word representations, I achieve about 40 % accuracy when the representations are retrained with the stacked bi-directional LSTM. This suggests that stacked architecture can achieve comparable accuracy to existing methods that use basic five physicochemical features without the

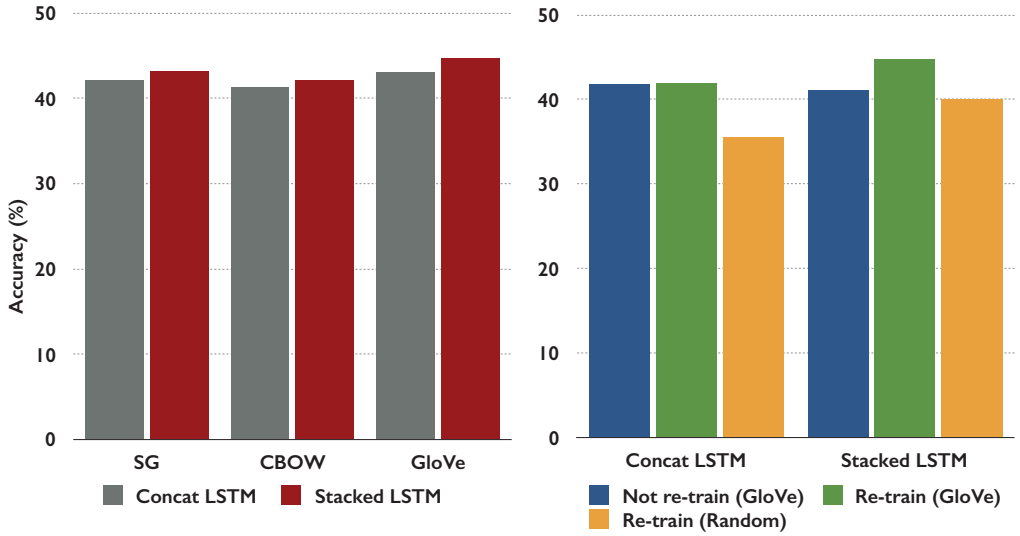

Figure 4.8: The effect of word representations and the re-training.

help of pre-trained representations. This also suggests that obtained features, with learning of long-term dependencies of amino acids in the protein sequence, can cover the information of basic physicochemical attributes.

However, my all experiments in this chapter showed that evolutionary information is eventually most effective feature for protein fold recognition.

4.5 Conclusion

In this chapter, I have shown LSTMs for protein fold recognition. Then I have proposed the use of two LSTM architectures: concatenated and stacked bi-directional LSTMs. On a benchmark dataset, my LSTMs have achieved high accuracy despite the use of a minimal set of features and limited tuning of hyper-parameters. In addition, throughout the analyses, I have found that the stacked architecture achieved excellent performance and LSTM was not adversely affected by very long protein sequences. Directions for future research include the following:

1. Exploration of further potential synergy of various other features between syntactical, physicochemical, and evolutionary information.

2. Extension to deeper and more effective stacked architectures.

3. Experiments on other bioinformatics tasks with my LSTMs.

Chapter 5

Residue-Residue Contact Prediction

5.1 Introduction

As described in the previous chapter, each protein has a unique 3D structure determined by the types and order of amino acid residues in the sequence. For describing such 3D structures in more detail than fold labels, protein data bank (PDB)[1] files provide information about 3D Cartesian coordinates of all atoms in proteins. The coordinate information is useful to understand the amino acid interactions, folding structures, biophysical and biochemical properties in proteins. However, since the protein 3D structure is invariant under translations and rotations, it is necessary to consider the structural representations that do not depend on Cartesian coordinates in PDB files.

As one of the representations for describing protein 3D structures, we generally say that two amino acid residues in a protein sequence are in *contact* if the Euclidean distance between the residues is less than 8Å. Then we can describe the residue-residue contact information as a binary $L \times L$ matrix, where L is the protein length. In this matrix, an element with value 1 indicates the corresponding two residues are in contact, otherwise they are not in contact and the value is 0. This symmetrical matrix is called *contact map* (Figure 1.2 in chapter 1), each element of which shows whether the two residues in a 1D sequence are close enough in a 3D space to interact or not.

In bioinformatics, protein 3D structure prediction from its amino acid sequence is assessed in the critical assessment of protein structure prediction (CASP)[2], which is a worldwide experiment taking place every two years since 1994. In particular, it remains challenging to predict residue-residue contacts in a protein from its amino

[1] `http://www.rcsb.org/pdb/`

[2] `http://www.predictioncenter.org/`.

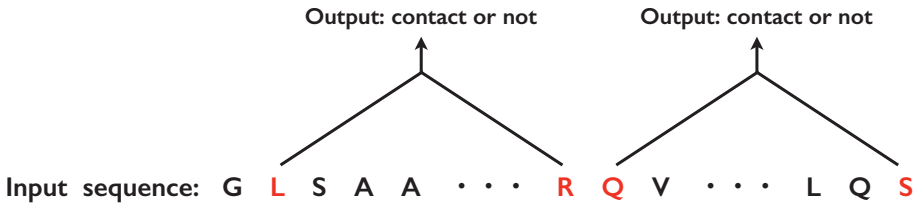

Figure 5.1: An input and output of the residue-residue contact prediction task. The input is a pair of amino acid residues in a protein sequence (e.g., "L" and "R" or "Q" and "S" in the above example) and the output is a label of contact or not.

acid sequence. This problem is called *residue-residue contact prediction* (Figure 5.1). Although the residue-residue contact prediction may be considered as a sub-problem in the protein 3D structure prediction problem, it is known that it is essentially equivalent to the 3D structure prediction because 3D structure can be recovered from sufficiently large number of true contacts [55, 61, 79].

However, residue-residue contact prediction is a very difficult problem because it can always occur that two residues far away from each other in a 1D sequence get close in a 3D structure due to the *protein folding process*. Such contacts between sequentially distant residues are called *long-range contacts*. The long-range contact information imposes strong spatial and physical constraints on the protein, which is important for understanding the folding process and recovering the 3D structure completely. However, despite significant efforts in CASP experiments, the progress has remained slow [24] and even the best methods have shown very low accuracies about 20% for long-range contacts [9]. In order to recover good 3D structures, simulations suggest that the accuracy for long-range contacts ought to be increased to about 35%.

To date, successful methods for residue-residue contact prediction are based on homology modeling [47, 35]. In homology modeling, the methods use known sequences and structures in databases, called *templates*, that are similar to unknown target protein. Since the methods require a large number (at least several hundreds) of templates for the target protein, there is a fundamental problem as follows: When the sequence similarities between the target sequence and sequences in databases are weak or good templates do not exist in databases, the prediction does not work well in the first place. This suggests that there is still room to consider the prediction from scratch (i.e., without templates), which is called *ab initio* modeling [7] in contrast to homology modeling.

Reliable methods based on *ab initio* modeling use some form of machine learning

such as support vector machines (SVMs) [11] and neural networks (NNs) [74]. For modeling residue-residue contact prediction with machine learning, there are three main issues [80, 47, 17] to be considered as follows:

1. Residue-residue contacts are spatially correlated and not randomly distributed in protein structures derived from the effect of evolutionary and physical information over the long protein sequence.

2. In protein 3D structures, there are effects derived from indirect interactions among residues. The indirect interaction where direct coupling of residues between A–B and B–C can result in observed correlations between A–C, even though no direct interaction exists between A and C.

3. Proteins do not assume a 3D conformation instantaneously, but rather through a folding process that gradually refines the structure.

While there are above issues, most methods (1) consider the pair of residue features independently and predict the contact element-by-element, (2) simply model the indirect interactions with sparsity constraints in contact data, and (3) attempt to learn contact probabilities at the local level in a single step.

To address these issues, in this thesis I model the above three properties in protein structures using deep neural networks. In particular, I combine the stacked bidirectional LSTM (described in chapter 4) and convolutional architecture on top of the hidden vectors obtained by the LSTM. Using the LSTM, I first obtain the hidden vectors that encode evolutionary and physical features such as α-helix and β-strand of amino acid residues over the protein sequence. As a result, I can automatically capture the all features between residues even I simply use the residue pair as an input for the contact classifier. In addition, I apply a convolutional neural network (CNN) to the pairs of hidden vectors, model the effect of indirect interactions in the forward computation of neural network, and then learn the combined architecture of LSTM and CNN in an end-to-end fashion. Furthermore, I apply the attention mechanism to LSTM, [58], which is widely used in NLP tasks such as machine translation.

My contributions are three-fold:

1. To the best of my knowledge, this is the first work that attempts to use LSTMs for protein residue-residue contact prediction and combine various architectures for capturing physical constraints, indirect interactions, and folding process.

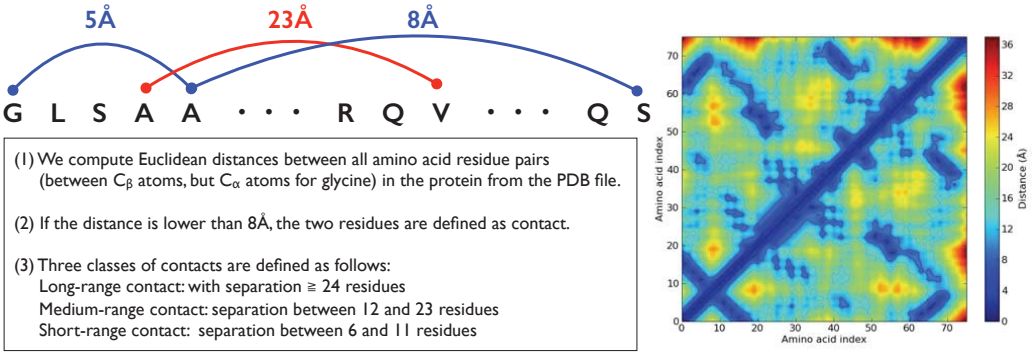

Figure 5.2: The definition of a protein residue-residue contact map.

2. On the CASP dataset, the proposed methods outperformed existing methods based on the homology modeling and *ab initio* model with various machine learning techniques. In particular, the stacked bi-directional LSTM with sparse attention mechanism for capturing protein folding process achieved the highest performance in my methods.

3. I showed that LSTMs also can also work well on very long protein sequences in residue-residue contact prediction. In addition, by comparing my various methods, I found that the simplest use of LSTM is enough to achieve high performance in residue-residue contact prediction.

5.2 Background and Related Work

In this section, I describe the task and evaluation of the protein residue-residue contact prediction (chapter 5.2.1). Then, I describe the protein properties to be considered in the prediction model with machine learning (chapter 5.2.2).

5.2.1 Task and Evaluation

To create a protein residue-residue contact map, I first compute Euclidean distances between all amino acid residue pairs from the PDB file of the protein. More precisely, the distance is computed with C_β atoms in amino acids (C_α atoms in the case of glycine). Next, if the distance is lower than 8Å, the pair of the two residues is defined as contact. In addition, three classes of contacts are defined as follows:

1. Long-range contacts with separation $\geqq 24$ residues.

2. Medium-range contacts with separation between 12 and 23 residues.

3. Short-range contacts with separation between 6 and 11 residues.

Finally, with these contact definitions, I create a binary $L \times L$ matrix, where L is the protein length, and this is the residue-residue contact map 5.2. While short-range contacts are easy to predict, long-range contacts are difficult to predict and important information for understanding protein 3D structures. The goal of this task is to correctly predict the long-range contacts.

Since there are few contacts in a sequence (in particular the number of long-range contacts is very few), the accuracy of contact prediction is evaluated with the precision as follows:

$$\text{Acc} = \text{TP}/(\text{TP} + \text{FP}),$$

where TP and FP are the true positive and false positive predicted contacts. If the above equation is $0/0$ and contact pairs exist in the sequence, the Acc is 0 %. On the other hand, if the above equation is $0/0$ and contact pairs do not exist in the sequence, the Acc is 100 %. Note that this Acc is computed for the sets of top $L/5$, $L/10$ and 5 predicted pairs, where L is the protein length. For example, the top $L/5$ predicted pairs mean the top $L/5$ data, which have high contact probability scores computed with trained predictor such as softmax classifier. The most widely used measure is Acc for top $L/5$ pairs of long-range.

It appears that $L/5$ is too small because the total number of residue pairs is $L(L-1)/2$. For example, while the total number of residue pairs is $4950 = (100 \times 99)/2$ when the protein length $L = 100$, only $100/5 = 20$ residue pairs are evaluated. However, there are some reasons to rely on this measure as follows:

- For example, since even a few correct long-range contact information can be useful to improve *ab initio* protein structure prediction, we do not need to know all correct contacts. It has been shown that even very sparse true contact information can also help to generate correct modeling at the fold level [55, 61, 79].

- In addition, since contact predictors are usually quite inaccurate, (e.g., the best accuracy at CASP rarely exceeds 20% for long-range contacts), it is difficult to distinguish a good predictor from a bad one if we consider all residue pairs.

- Furthermore, it is known that the number of contacts in a protein is linear in its sequence length.

5.2.2 Physical Constraints, Indirect Interactions, and Folding Process

In chapter 4, I showed that the LSTM with biological features achieved high performance in the protein fold recognition. However, protein residue-residue contact prediction is more difficult and complex task than fold recognition because the output is more detailed 3D structure information about proteins. For residue-residue contact prediction, previous work such as [35, 80, 41] mainly consider three properties of proteins and model these with machine learning as follows:

(1) **Physical constraints:** One of the physical constraints is that the number of contacts in a protein is only linear in its sequence length. This also means that the number of contacts in a protein is small, i.e., a residue-residue contact map must be sparse. Based on this observation, physical constraints are modeled through machine learning methods such as sparse inverse covariance estimation [47, 35]. However, the sparsity constraint-based methods ignore many concrete constraints derived from secondary structures such as α-helix and β-strand over the protein sequence. On the other hand, considering of the concrete constraints such as parallel, anti-parallel, and bridge patterns in secondary structures leads to more heuristic processing [80]. To overcome these weaknesses, I need a model to automatically capture the all physical information over the arbitrary length of protein sequence .

(2) **Indirect interactions:** Indirect interactions between residues in a protein is caused by the two direct contact pairs that share a common residue, which results in the so-called *transitive dependencies*. For example, each residue pair, A-B and B-C, is contact which induces the residue pair A–C is also contact. Such indirect interaction among residues is widely observed, which restricts the performance of contact prediction. Indeed, sometimes the interaction of the indirect residue pair is stronger even than the direct residue pair, which is called *transitive noise*. This transitive noise can also be removed through sparse inverse covariance estimation [35] as described in the above. However, indirect interaction effect can be modeled more explicitly with machine learning rather than modeled implicitly with the sparsity constraint in a contact

map.

(3) **Protein Folding:** As described in section 5.1, while residue-residue contacts are spatially correlated, most of the methods attempt to predict the contact probability considering only the target residue pair and ignoring all other residues. In addition, while proteins do not assume a 3D conformation instantaneously but rather through a folding process that gradually refines the structure, most of the methods attempt to learn the contact probability at the local level in a single step. This suggests that machine learning models for residue-residue contact prediction ought to simulate the protein folding process in the training process.

In this thesis, while I also focus on the above protein properties as the same as previous work mainly consider [35, 80, 41], I model these using deep neural architectures described in the following section.

5.3 Method

5.3.1 Features

In protein residue-residue contact prediction, I use a minimal set of amino acid residue features, which are used in protein fold recognition described in chapter 4.3.1. More precisely, each residue feature vector in a protein sequence includes three kinds of information as follows:

1. PSI-BLAST sequence profile (20-dimensional real value vector).
2. Secondary structures (three binary values: α-helix, β-strand, or coil).
3. Solvent accessibilities (two binary values: buried or exposed).

The sequence profile is obtained by running PSI-BLAST [1] with an E-value cutoff equal to 0.001 and for four iterations against NCBI's non-redundant version of the protein sequence database NR filtered at 90% sequence identity. The secondary structure and solvent accessibility are predicted by running SSpro [54] and ACCpro [53] from the SCRATCH suite [12]. In this chapter, I represent the feature vector of the i-th residue in a protein sequence as $\mathbf{x}_i \in \mathbb{R}^d$ (in the above feature, $d = 24$), which is the same representation used in the previous chapter. Then, given a protein sequence

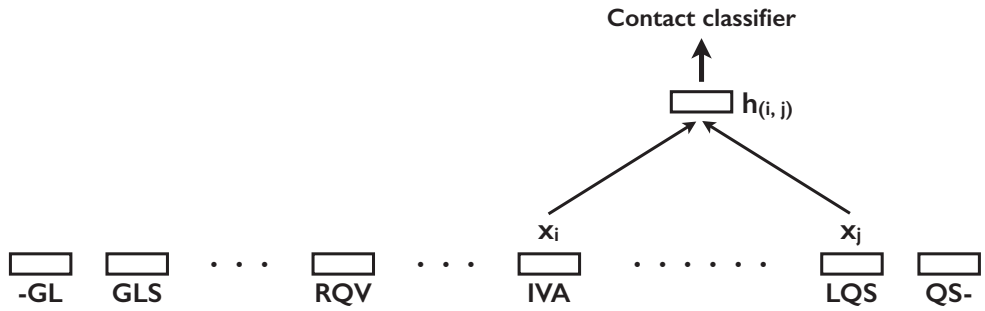

Figure 5.3: The baseline model for protein residue-residue contact prediction.

P, I transform the all residues to feature vectors and concatenate them based on the window size as described in Equation (4.1). In the following section, I use a feature vector sequence $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_L$ as an input sequence for an RNN.

5.3.2 Baseline

In this subsection, I describe a baseline method for protein residue-residue contact prediction with a neural network-based architecture. This baseline method uses the pair of residue feature vectors as an input for the neural network and predict the contact with a softmax classifier (Figure 5.3).

Forward Computation: More precisely, for the pair of $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{x}_j \in \mathbb{R}^d$, which are the i-th and j-th residue feature vectors in a protein sequence, I obtain the hidden vector $\mathbf{h}_{(i,j)} \in \mathbb{R}^d$ using a neural network as follows:

$$\mathbf{h}_{(i,j)} = f\left(\mathbf{W}_h \begin{bmatrix} \mathbf{x}_i \\ \mathbf{x}_j \\ |i-j|/L \end{bmatrix} + \mathbf{b}_h\right),$$

where $\mathbf{W}_h \in \mathbb{R}^{d \times (2d+1)}$ is the weight matrix to learn, $\mathbf{b}_h \in \mathbb{R}^d$ is the bias vector to learn, and f is the non-linear activation function such as ReLU. Note that I consider the *position feature* of i and j as a scalar value $|i-j|/L$. Then I obtain the output vector $\mathbf{y}_{(i,j)} \in \mathbb{R}^2$ (i.e., binary classification) as follows:

$$\mathbf{y}_{(i,j)} = \mathbf{W}_y \mathbf{h}_{(i,j)} + \mathbf{b}_y,$$

where $\mathbf{W}_y \in \mathbb{R}^{2 \times d}$ is the weight matrix to learn and $\mathbf{b}_y \in \mathbb{R}^2$ is the bias vector to learn. Finally, a softmax layer is added on top of the output vector $\mathbf{y}_{(i,j)}$ for modeling contact

probabilities as follows:

$$p_{t_{(i,j)}} = \frac{\exp(y_{t_{(i,j)}})}{\sum_{k=1}^{2} \exp(y_k)},$$

where $t_{(i,j)} \in \{1, 2\}$ is the label of contact or not between i-th and j-th residues, and $p_{t_{(i,j)}}$ is the probability of $t_{(i,j)}$.

Training: Given a set of the all residue pairs in a protein sequence, the training objective is to minimize the loss function $\mathcal{L}$, given as the cross-entropy loss as follows:

$$\mathcal{L}(\Theta) = \sum_{i=1}^{L-1} \sum_{j=i+1}^{L} \log p_{t_{(i,j)}},$$

where L is the protein length and Θ is the set of the all weight matrices and bias vectors to learn in LSTMs and $\mathbf{W}_h$, $\mathbf{W}_y$, $\mathbf{b}_h$, and $\mathbf{b}_y$ described in this subsection. Note that while the above equation considers the all residue pairs in a protein sequence, in practice I consider the residue pairs with separation $\geqq 6$ and randomly select 20% of the negative (i.e., not contact) examples while keeping all the positive (i.e., contact) examples because the contact data is unbalanced. In addition, the above loss is computed for a protein sequence and I represent the loss for the n-th protein in the training dataset as $\mathcal{L}_n(\Theta)$. In the end, the final training objective is to minimize the loss function $\mathcal{L}_{total}$, given as the sum of cross-entropy losses for all proteins in the training dataset plus an L2-regularization term as follows:

$$\mathcal{L}_{total}(\Theta) = \sum_{n=1}^{N} \mathcal{L}_n(\Theta) + \frac{\lambda}{2}||\Theta||_2^2,$$

where N is the total number of protein sequences in the training dataset and λ is an L2 regularization hyper-parameter. Then I use a standard backpropagation technique to train Θ.

5.3.3 Multi-layer Stacked Bi-directional LSTM

In this subsection, I apply a stacked bi-directional LSTM (described in the previous chapter) to residue-residue contact prediction and then extend the LSTM to multi-layer architecture (Figure 5.4).

More precisely, given a feature vector sequence of residues $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_L]$ as an input, I apply a stacked bi-directional LSTM function to obtain a hidden vector

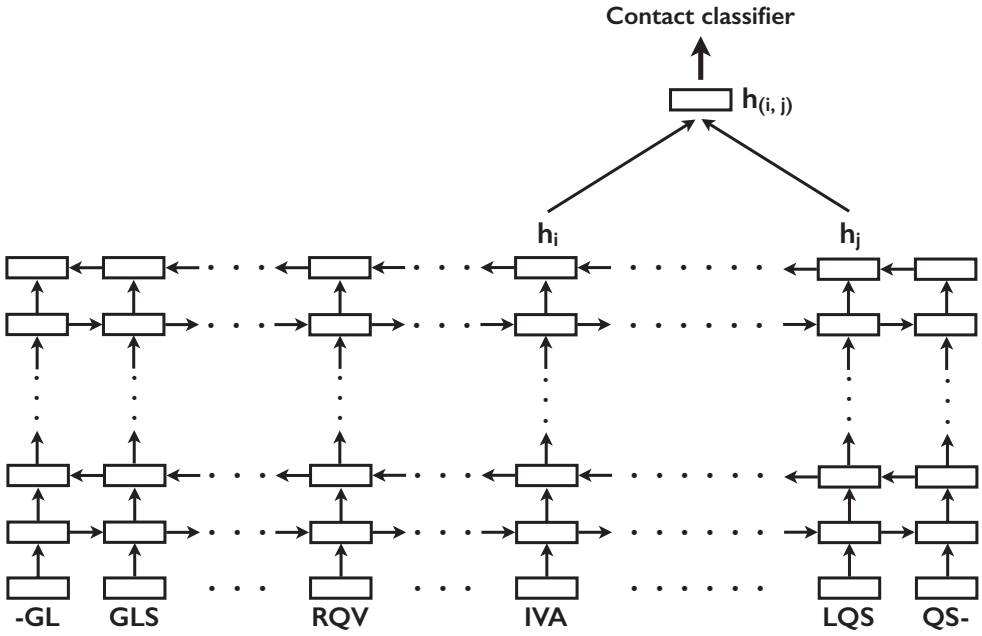

Figure 5.4: The multi-layer stacked bi-directional LSTM.

sequence $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_L]$, where $\mathbf{h}_i \in \mathbb{R}^d$ is the hidden vector of $\mathbf{x}_i$. This stacked bi-directional LSTM function is represented as follows:

$$\mathbf{H} = \text{stack-bilstm}(\mathbf{X}). \tag{5.1}$$

Then I use the obtained hidden vector sequence $\mathbf{H}$ as an input sequence of the next layer and represent the ℓ-th hidden vector sequence as $\mathbf{H}^{(\ell)}$. I call this architecture *multi-layer stacked bi-directional LSTM* and the stacked computation can be stated with Equation (5.1) as follows:

$$\begin{aligned}
&\mathbf{H}^{(1)} = \text{stack-bilstm}(\mathbf{X}),\\
&\mathbf{H}^{(2)} = \text{stack-bilstm}(\mathbf{H}^{(1)}),\\
&\cdots\\
&\mathbf{H}^{(\ell)} = \text{stack-bilstm}(\mathbf{H}^{(\ell-1)}).
\end{aligned}$$

With the obtained ℓ-th hidden vector sequence $\mathbf{H}^{(\ell)} = [\mathbf{h}_1^{(\ell)}, \mathbf{h}_2^{(\ell)}, \cdots, \mathbf{h}_L^{(\ell)}]$, the final

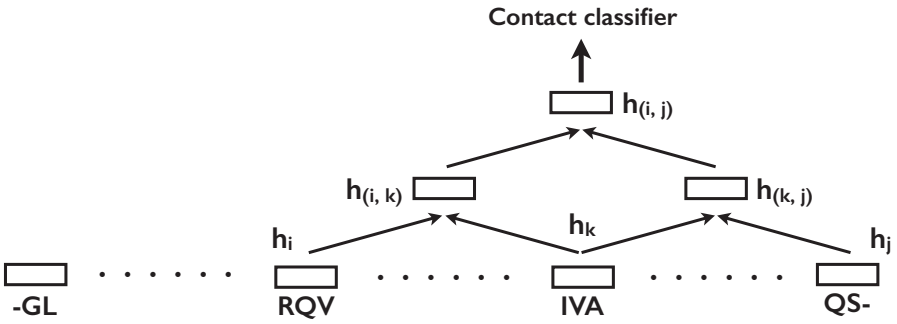

Figure 5.5: The convolutional architectures for capturing indirect interactions between residues.

hidden vector $\mathbf{h}^{(\ell)}_{(i,j)} \in \mathbb{R}^d$ for the pair of i-th and j-th residues is computed as follows:

$$\mathbf{h}^{(\ell)}_{(i,j)} = f\left(\mathbf{W}_h \begin{bmatrix} \mathbf{h}^{(\ell)}_i \\ \mathbf{h}^{(\ell)}_j \\ |i-j|/L \end{bmatrix} + \mathbf{b}_h\right).$$

Thus, the processing by using the LSTMs allows all residue pairs to automatically capture the evolutionary and physical information over the protein sequence even if the residue pairs are independently extracted and inputed for the contact classifier such as baseline.

5.3.4 Modeling Indirect Interactions with Convolution

In this subsection, I attempt to directly model the indirect interactions among residues described in chapter 5.2.2 using convolutional architectures (Figure 5.5) on top of the LSTM.

Given i-th and j-th hidden vectors of residues, which the distance of the pair is long-range (i.e., $|i-j| > 24$), I use the $\mathbf{h}_k$, where $k = \lceil (i+j)/2 \rceil$, for computing the hidden

vector of $\mathbf{h}_i$ and $\mathbf{h}_j$ as follows:

$$\mathbf{h}_{(i,k)} = f\left(\mathbf{W}_h \begin{bmatrix} \mathbf{h}_i \\ \mathbf{h}_k \\ \lfloor |i-k|/L \rfloor \end{bmatrix} + \mathbf{b}_h\right),$$
$$\mathbf{h}_{(k,j)} = f\left(\mathbf{W}_h \begin{bmatrix} \mathbf{h}_k \\ \mathbf{h}_j \\ \lfloor |k-j|/L \rfloor \end{bmatrix} + \mathbf{b}_h\right),$$

and then I obtain the hidden vector $\mathbf{h}_{(i,j)}$ as follows:

$$\mathbf{h}_{(i,j)} = f\left(\mathbf{W}_h \begin{bmatrix} \mathbf{h}_{(i,k)} \\ \mathbf{h}_{(k,j)} \\ \lfloor |i-j|/L \rfloor \end{bmatrix} + \mathbf{b}_h\right).$$

This allows us to directly and naturally model the indirect interactions between the residues in the neural network. In addition, the above architecture is easy to extend to capture more indirect interactions (i.e., two-step and three-step indirect interactions). In my experiments, the condition, i.e., the range and number of indirect interactions, is as follows: for $48 \leq |i-j| < 96$: one-step; for $96 \leq |i-j| < 144$: two-step; and for $|i-j| \geq 144$: three-step.

5.3.5 Modeling Folding Process with Attention

In this subsection, I attempt to model the folding process of proteins in the training process of neural networks. The idea is based on *attention* mechanism [58] that can capture some important features in the sequence for the target feature by using weights, which are called attentions. The attentions are computed and trained with neural networks. For the residue-residue contacts, I consider the use of attentions and extend these to sparse because the sparsity constraint is an important property in contacts. In addition, I also assume that the training process of sparse attentions allows the neural network to simulate the protein folding process.

Given a hidden vector $\mathbf{h}_i \in \mathbb{R}^d$, I obtain $\alpha_{(i,k)} \in \mathbb{R}$, which is the attention (i.e., weight) for another k-th hidden vector $\mathbf{h}_k \in \mathbb{R}^d$ in the sequence, by using a bilinear function as follows:

$$\alpha_{(i,k)} = \mathbf{h}_i^\top \mathbf{M} \mathbf{h}_k,$$

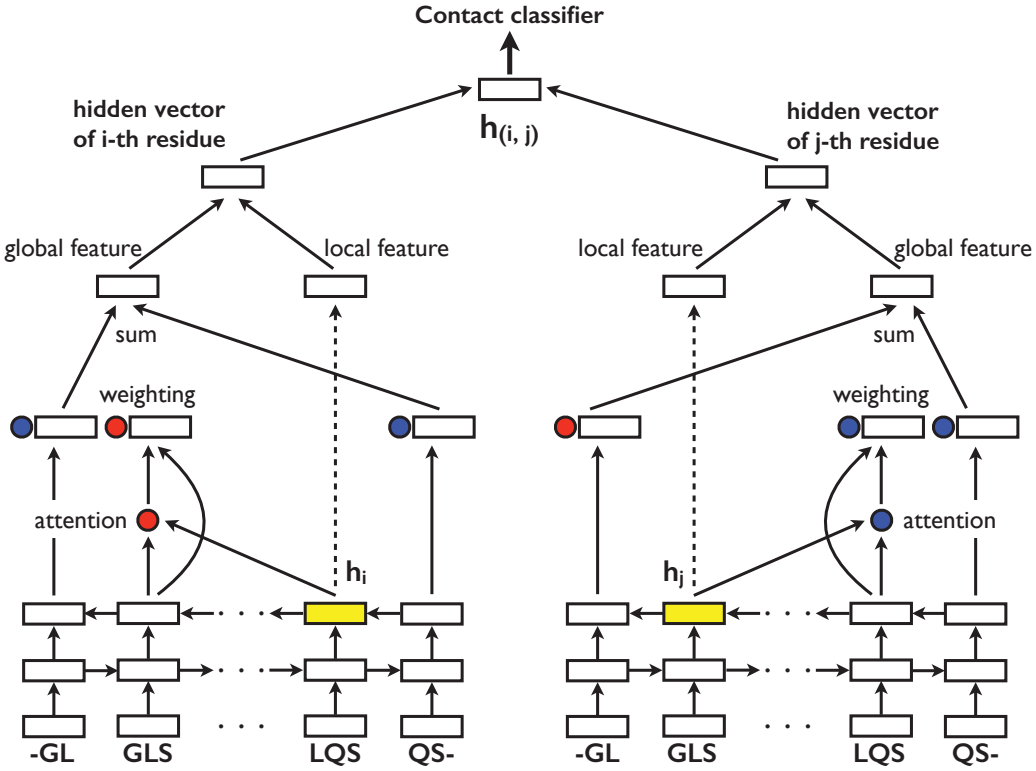

Figure 5.6: The attention mdoel for residue-residue contact prediction.

where $\mathbf{M} \in \mathbb{R}^{d \times d}$ is the weight matrix to learn. Then I compute the all attentions over the sequence and obtain the *attention vector* $\boldsymbol{\alpha}_i \in \mathbb{R}^L$ as follows:

$$\boldsymbol{\alpha}_i = [\alpha_{(i,1)}, \alpha_{(i,2)}, \cdots, \alpha_{(i,L)}].$$

The element $\alpha_{(i,k)}$ in the attention vector $\boldsymbol{\alpha}_i$ represents the effect or interaction between the hidden vectors of i-th and k-th residues. To the attention vector $\boldsymbol{\alpha}_i$, I apply the softmax function parameterized with a temperature as follows:

$$\tilde{\boldsymbol{\alpha}}_i = \text{softmax}\left(\frac{\boldsymbol{\alpha}_i}{T}\right),$$

where $T \in \mathbb{R}$ is the temperature. T is a hyper-parameter and the initial value is very high (e.g., 10^6). Then I gradually decrease the temperature according to the training process and this allows the model to simulate the protein folding process (described in the next paragraph). With the attention vector $\tilde{\boldsymbol{\alpha}}_i$, I compute the weighted sum of the all hidden vectors in the sequence and obtain the vector $\mathbf{r}_i \in \mathbb{R}^d$ as follows:

$$\mathbf{r}_i = \sum_{m=1}^{L} \tilde{\alpha}_{(i,m)} \mathbf{h}_m.$$

Then, I compute a new hidden vector $\mathbf{s}_i \in \mathbb{R}^d$ from $\mathbf{h}_i$ and $\mathbf{r}_i$ as follows:

$$\mathbf{s}_i = f\left(\mathbf{W}_s \begin{bmatrix} \mathbf{r}_i \\ \mathbf{h}_i \end{bmatrix} + \mathbf{b}_s\right).$$

In the same way, for another hidden vector $\mathbf{h}_j$, I compute the attention vector $\tilde{\boldsymbol{\alpha}}_j$ and obtain $\mathbf{s}_j$. Finally, the hidden vector $\mathbf{h}_{(i,j)}$ is computed as follows:

$$\mathbf{h}_{(i,j)} = f\left(\mathbf{W}_h \begin{bmatrix} \mathbf{s}_i \\ \mathbf{s}_j \\ |i-j|/L \end{bmatrix} + \mathbf{b}_h\right).$$

Note that $\mathbf{r}_i$ and $\mathbf{r}_j$ are zero vectors early in the training process because the protein length is usually long (e.g., several hundreds), and all elements in such several hundred-dimensional attention vector are very small positive values due to the softmax function. However, according to the training process, the attention vector can be sparse due to the decreasing of the temperature T and gradually select more important residues for the target residue $\mathbf{x}_i$ or $\mathbf{x}_j$. This gradual weighting with attentions and the sparsity, which have strong interactions for only the specific residues according to the training process, allow the neural network to simulate the protein folding process. In addition, sparsity of the attention is also important for residue-residue contacts [35]. Indeed, while the idea is similar to the [41], I efficiently consider the all residue features over the sequence, model the residue-residue interactions as attention weights, and learn the parameters with deep neural networks (Figure 5.6).

5.4 Experiments and Results

5.4.1 Dataset

For training, I use the DNcon dataset [22]. This dataset consists of 1492 proteins in protein data bank of which any two proteins have less than 30% of sequence identity. I split the DNcon dataset 9:1 and used these as train/development sets. As described in chapter 5.3.2, since the contact data is unbalanced, I randomly select 20% of the negative (i.e., not contact) examples while keeping all the positive (i.e., contact) examples. This setting is the same as previous work such as [22, 41]. For test, I used the CASP10

Compared Methods	**Accuracy (Long-range L/5)**
SVMcon (Cheng and Baldi, 2007)	23.3
NNcon (Tegge et al., 2009)	18.8
Evfold (Morcos et al., 2011)	22.5
PSICOV (Jones et al., 2012)	22.5
DNcon (Eickholt et al., 2012)	29.1
CMAPro (Lena et al., 2012)	31.3
CoinCDA (Ma et al., 2015)	35.1
PhyCAMP (Wang and Xu, 2013)	37.3
My Methods	**Accuracy (Long-range L/5)**
Baseline	23.4
Stack Bi-LSTM	35.6
Stack Bi-LSTM + CNN	36.9
Stack Bi-LSTM + Attention	**37.5**

Table 5.1: Long-range accuracies of compared methods and my methods in the CASP10 dataset.

dataset[3]. This dataset consists of 124 proteins and all the training proteins in DNcon are selected before CASP10 (started in May 2012).

Note that while the number of protein sequences in the training and development dataset is 1426, the number of samples, i.e., the total number of amino acid residue pairs is 5111528. The size of this training dataset is enough to train deep neural networks.

5.4.2 Implementation

I implemented my LSTMs using Chainer[4] [75] and the optimization method was ADAM [38], which is one of the most effective stochastic gradient descent (SGD)-based algorithms for training deep neural networks. I set ADAM with a first momentum coefficient of 0.9 and a second momentum coefficient of 0.999, which were the

[3] `http://predictioncenter.org/casp10/index.cgi`
[4] `http://chainer.org/`

recommended configurations in that study. I found that the LSTMs do not need careful tuning of hyper-parameters and are easy to train in practice. The training details are as follows:

- Window size w: 9 (the same as compared method such as [22, 41]).
- Batch size: the number of residue pairs in each protein.
- L2-regularization strength λ: 1e-6, 1e-7, 1e-8, and 1e-9.

Thus, while I tuned only L2-regularization strength λ as a hyper-parameter with the development set, my LSTMs showed excellent performance as described in the following section.

5.4.3 Main Result

Accuracy: Table 5.1 shows the accuracies of compared methods and my methods in the CASP10 dataset. I observed that the stacked bi-directional LSTM with attention mechanism outperformed all existing methods. To the best of my knowledge, this is the first result to show that deep neural network-based method, in particular the LSTM architecture outperformed existing methods in protein residue-residue contact prediction task. Surprisingly, while the accuracy of my baseline method (chapter 5.2.2) is lower than 25 % and NNcon are also similar accuracies), the simple use of the LSTM can achieve the accuracy over 35 %. The improvement derived from the LSTM is over 10 %.

Comparison with Other Methods: In Table 5.1, I observed that the simple LSTM outperformed all other deep learning models such as DNcon (Eickholt et al., 2012) [22] and CMAPro (Lena et al., 2012) [41]. Interestingly, while the DNcon dataset consist of only low sequence identity (less than 30%) proteins, my LSTMs can achieve high accuracies and outperformed all existing homology modeling-based methods such as Evfold (Morcos et al., 2011) [47] and PSICOV (Jones et al., 2012) [35]. On the other hand, PhyCAMP (Wang and Xu, 2013) [80], which consider the concrete physical constraints such as parallel, anti-parallel, and bridge patterns in secondary structures over the protein sequence, achieved high performance and the accuracy is 37.3 %. In the future, deep neural networks with more biological knowledge can be considered for the improvement.

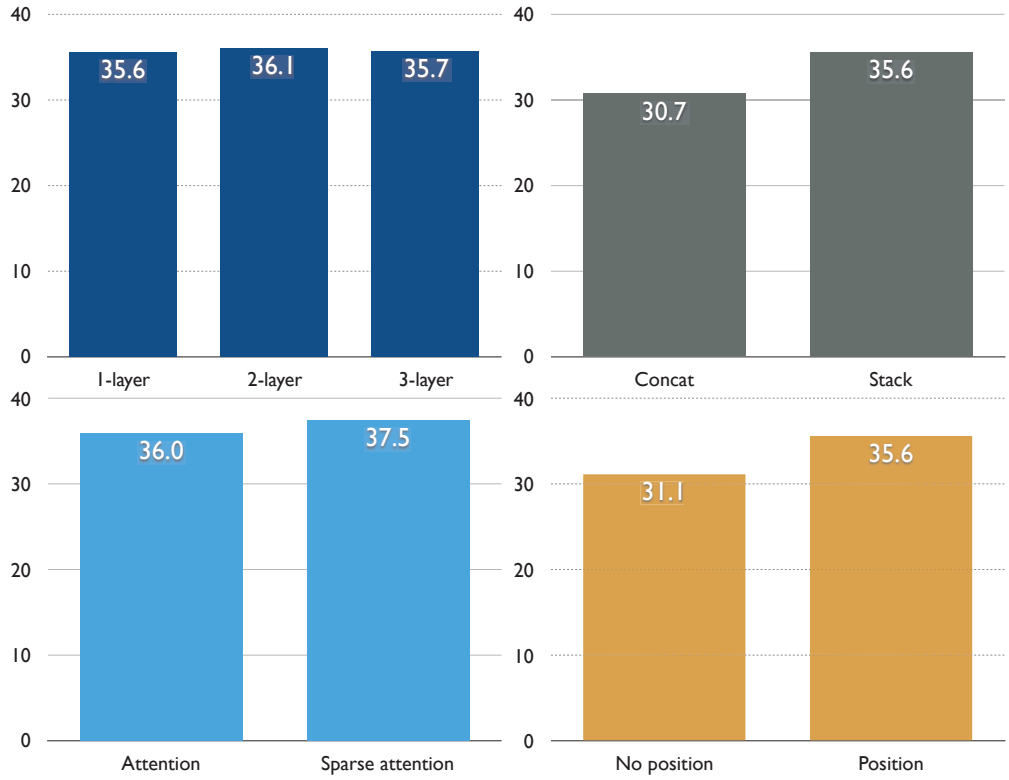

Figure 5.7: The effects of various components in my LSTMs.

5.4.4 Analysis

The Effect of Multi-layers in LSTMs: Figure 5.7 shows the accuracies (long-range L/5) of the various architectures. This suggests that the accuracy improvement is small even if much of the multi-layer stacked bi-directional architectures are considered and the number of learning parameters in LSTMs becomes large. On the basis of this observation, I found that the 1-layer stacked architecture can be learned fast, robustly facilitating high performance in residue-residue contact prediction.

The Effect of Stacked Architecture: While the effect of multi-layers in LSTM is small for the accuracy improvement, Figure 5.7 shows that the accuracy of stacked architecture is higher than concatenated architecture by a large margin. This difference and effect of the stacked architecture is larger than the result in fold recognition described in chapter 4.4. On the basis of this observation, I found that the stacked architecture is more effective for residue-residue contact prediction rather than fold recognition.

The Effect of Position Features: Interestingly, the position feature gives an improvement by a large margin as shown in Figure 5.7 even if the feature is a simple scalar value such as $|i-j|/L$. If the position feature is not considered, the accuracy degrades about 5 %. This suggests that it is important for the long-range residue-residue to explicitly take into account the position in the sequence and input the feature for the contact classifier.

The Effect Sparse Attention: In Figure 5.7, while the sparse attention achieved the best accuracy in my proposed methods, the effect of sparsity in attention is relatively small (improvement is 1.5 %) compared with the effect of stacked architecture and position feature.

Throughout these analysis, I found that the effective components in the prediction model of protein residue-residue contact is (1) LSTM, in particular the stacked architecture (not multi-layers) and (2) explicit position feature for the classifier.

5.5 Conclusion

In this chapter, I have modeled the various properties in protein structures with deep neural networks. In particular, I have combined the LSTM and CNN, and then learned the combined architecture with end-to-end fashion. In addition, I have applied the sparse attention mechanism to the LSTM. To the best of my knowledge, this is the first work that attempts to use LSTMs for protein residue-residue contact prediction and combine various architectures for capturing physical constraints, indirect interactions, and folding process.

Chapter 6

Conclusions

In this thesis, I have focused on the problems in NLP and bioinformatics, in particular the problems of semantic composition and protein structure prediction. In addition, I have solved the problems with various RNN-based neural network architectures specified for each problem. I first used RNNs with long short-term memory (LSTM) units, which can find long-term dependencies in a sequence and store the information for a long period of time. Through my experiments, I have shown that LSTMs provide effective and general sequential representations for both natural languages and proteins. Then, on top of the LSTM, I have combined other machine learning techniques such as the kernel method, convolutional neural network, and attention mechanism to capture each property in natural languages and proteins.

In chapter 3, I have proposed a new method of non-linear similarity learning for semantic compositionality. Instead of relying on only neural network-based operating in a low-dimensional space, I train kernel functions that allows us to measure the semantic similarity in a high-dimensional space. In the task of predicting the relatedness of two sentences, my method have outperformed linear baselines, feature engineering approaches, and achieved competitive results with recent deep learning models.

In chapter 4, I have shown LSTMs for protein fold recognition. Then I have proposed the use of two LSTM architectures: concatenated and stacked bi-directional LSTMs. On a benchmark dataset, my LSTMs have achieved high accuracy despite the use of a minimal set of features and limited tuning of hyper-parameters. In addition, throughout the analyses, I have found that the stacked architecture achieved excellent performance and LSTM was not adversely affected by very long protein sequences.

In chapter 5, I have modeled the various properties in protein structures with deep neural networks. In particular, I have combined the LSTM and CNN, and then learned

the combined architecture with end-to-end fashion. In addition, I have applied the sparse attention mechanism to the LSTM. To the best of my knowledge, this is the first work that attempts to use LSTMs for protein residue-residue contact prediction and combine various architectures for capturing physical constraints, indirect interactions, and folding process.

Bibliography

[1] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

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

[3] A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. *CoRR*, abs/1306.6709, 2013.

[4] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *International conference on machine learning*, pp. 41–48, 2009.

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

[6] J. Bjerva, J. Bos, R. van der Goot, and M. Nissim. The meaning factory: Formal semantics for recognizing textual entailment and determining semantic similarity. In *SemEval*, 2014.

[7] R. Bonneau and D. Baker. Ab initio protein structure prediction: progress and prospects. *Annual review of biophysics and biomolecular structure*, 30(1):173–189, 2001.

[8] D. Bouchaffra and J. Tan. Protein fold recognition using a structural hidden markov model. In *International Conference on Pattern Recognition (ICPR)*, 2006.

[9] L. Burger and E. Van Nimwegen. Disentangling direct from indirect co-evolution of residues in protein alignments. *PLoS Comput Biol*, 6(1):e1000633, 2010.

[10] Y. Chen, X. Zhang, M. Q. Yang, and J. Y. Yang. Ensemble of probabilistic neural networks for protein fold recognition. In *International Symposium on BioInformatics and BioEngineering*, 2007.

[11] J. Cheng and P. Baldi. Improved residue contact prediction using support vector machines and a large feature set. *BMC bioinformatics*, 8(1):1, 2007.

[12] J. Cheng, A. Z. Randall, M. J. Sweredoski, and P. Baldi. Scratch: a protein structure and structural feature prediction server. *Nucleic acids research*, 33(suppl 2):W72–W76, 2005.

[13] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the Conference on International Conference on Machine Learning (ICML)*, 2008.

[14] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research (JMLR)*, 2011.

[15] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *JASIS*, 1990.

[16] A. Dehzangi, A. Sharma, J. Lyons, K. K. Paliwal, and A. Sattar. A mixture of physicochemical and evolutionary–based feature extraction approaches for protein fold recognition. *International journal of data mining and bioinformatics*, 11(1):115–138, 2014.

[17] P. Di Lena, K. Nagata, and P. Baldi. Deep architectures for protein contact map prediction. *Bioinformatics*, 28(19):2449–2457, 2012.

[18] C. H. Ding and I. Dubchak. Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics*, 17(4):349–358, 2001.

[19] Q.-w. Dong, X.-l. Wang, and L. Lin. Application of latent semantic analysis to protein remote homology detection. *Bioinformatics*, 22(3):285–290, 2006.

[20] Q. Dong, S. Zhou, and J. Guan. A new taxonomy-based protein fold recognition approach based on autocross-covariance transformation. *Bioinformatics*, 25(20):2655–2662, 2009.

[21] I. Dubchak, I. B. Muchnik, and S.-H. Kim. Protein folding class predictor for scop: approach based on global descriptors. In *Ismb*, 1997.

[22] J. Eickholt and J. Cheng. Predicting protein residue–residue contacts using deep networks and boosting. *Bioinformatics*, 28(23):3066–3072, 2012.

[23] S. El Hihi and Y. Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems (NIPS)*, pp. 493–499, 1995.

[24] I. Ezkurdia, O. Grana, J. M. Izarzugaza, and M. L. Tress. Assessment of domain boundary predictions and the prediction of intramolecular contacts in casp8. *Proteins: Structure, Function, and Bioinformatics*, 77(S9):196–209, 2009.

[25] G. Frege. Über sinn und bedeutung. In *Zeitschfrift für Philosophie und philosophische Kritik, 100*, 1892.

[26] M. Ganapathiraju, D. Weisser, R. Rosenfeld, J. Carbonell, R. Reddy, and J. Klein-Seetharaman. Comparative n-gram analysis of whole-genome protein sequences. In *International Conference on Human Language Technology Research*, 2002.

[27] J. Gao, X. He, W.-t. Yih, and L. Deng. Learning continuous phrase representations for translation modeling. In *Proceedings of the Conference on Association for Computational Linguistics (ACL)*, 2014.

[28] P. Ghanty and N. R. Pal. Prediction of protein folds: extraction of new features, dimensionality reduction, and fusion of heterogeneous classifiers. *NanoBioscience*, 8(1):100–110, 2009.

[29] M. Gönen and E. Alpaydın. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12(Jul):2211–2268, 2011.

[30] A. Graves, N. Jaitly, and A.-R. Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU)*, 2013.

[31] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[32] S. Hochreiter, M. Heusel, and K. Obermayer. Fast model-based protein homology detection without alignment. *Bioinformatics*, 23(14):1728–1736, 2007.

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

[34] S. Jimenez, G. Duenas, J. Baquero, A. Gelbukh, A. J. D. Bátiz, and A. Mendizábal. Unal-nlp: Combining soft cardinality features for semantic textual similarity, relatedness and entailment. In *SemEval*, 2014.

[35] D. T. Jones, D. W. Buchan, D. Cozzetto, and M. Pontil. Psicov: precise structural contact prediction using sparse inverse covariance estimation on large multiple sequence alignments. *Bioinformatics*, 28(2):184–190, 2012.

[36] D. Kedem, S. Tyree, F. Sha, G. R. Lanckriet, and K. Q. Weinberger. Non-linear metric learning. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS)*, 2012.

[37] D. Kiela and L. Bottou. Learning image embeddings using convolutional neural networks for improved multi-modal semantics. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing (EMNLP)*, 2014.

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

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NIPS)*, 2012.

[40] A. Lai and J. Hockenmaier. Illinois-lh: A denotational and distributional approach to semantics. In *SemEval*, 2014.

[41] P. D. Lena, K. Nagata, and P. F. Baldi. Deep spatio-temporal architectures and learning for protein structure prediction. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 512–520, 2012.

[42] J. Lyons, N. Biswas, A. Sharma, A. Dehzangi, and K. K. Paliwal. Protein fold recognition by alignment of amino acid residues using kernelized dynamic time warping. *Journal of theoretical biology*, 354:137–145, 2014.

[43] J. Lyons, K. K. Paliwal, A. Dehzangi, R. Heffernan, T. Tsunoda, and A. Sharma. Protein fold recognition using hmm–hmm alignment and dynamic programming. *Journal of theoretical biology*, 393:67–74, 2016.

[44] M. Marelli, L. Bentivogli, M. Baroni, R. Bernardi, S. Menini, and R. Zamparelli. Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *SemEval*, 2014.

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

[46] J. Mitchell and M. Lapata. Composition in distributional models of semantics. *Cognitive Science*, 34(8):1388–1439, 2010.

[47] F. Morcos, A. Pagnani, B. Lunt, A. Bertolino, D. S. Marks, C. Sander, R. Zecchina, J. N. Onuchic, T. Hwa, and M. Weigt. Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proceedings of the National Academy of Sciences*, 108(49):E1293–E1301, 2011.

[48] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. Scop: a structural classification of proteins database for the investigation of sequences and structures. *Journal of molecular biology*, 247(4):536–540, 1995.

[49] C. A. Orengo, A. Michie, S. Jones, D. T. Jones, M. Swindells, and J. M. Thornton. Cath–a hierarchic classification of protein domain structures. *Structure*, 5(8):1093–1109, 1997.

[50] K. K. Paliwal, A. Sharma, J. Lyons, and A. Dehzangi. A tri-gram based feature extraction technique using linear probabilities of position specific scoring matrix for protein fold recognition. *IEEE transactions on nanobioscience*, 13(1):44–50, 2014.

[51] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

[52] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing (EMNLP)*, 2014.

[53] G. Pollastri, P. Baldi, P. Fariselli, and R. Casadio. Prediction of coordination number and relative solvent accessibility in proteins. *Proteins: Structure, Function, and Bioinformatics*, 47(2):142–153, 2002.

[54] G. Pollastri, D. Przybylski, B. Rost, and P. Baldi. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics*, 47(2):228–235, 2002.

[55] M. Porto, U. Bastolla, H. E. Roman, and M. Vendruscolo. Reconstruction of protein structures from a vectorial representation. *Physical review letters*, 92(21):218101, 2004.

[56] A. M. Qamar, E. Gaussier, J.-P. Chevallet, and J. H. Lim. Similarity learning for nearest neighbor classification. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, 2008.

[57] G. Raicar, H. Saini, A. Dehzangi, S. Lal, and A. Sharma. Improving protein fold recognition and structural class prediction accuracies using physicochemical properties of amino acids. *Journal of theoretical biology*, 402:117–128, 2016.

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

[59] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[60] A. M. Rush, S. Chopra, and J. Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

[61] R. Sathyapriya, J. M. Duarte, H. Stehr, I. Filippis, and M. Lappe. Defining an essence of structure determining residue contacts in proteins. *PLoS Comput Biol*, 5(12):e1000584, 2009.

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

[63] D. B. Searls. The language of genes. *Nature*, 420(6912):211–217, 2002.

[64] A. Sharma, J. Lyons, A. Dehzangi, and K. K. Paliwal. A feature extraction technique using bi-gram probabilities of position specific scoring matrix for protein fold recognition. *Journal of theoretical biology*, 320:41–46, 2013.

[65] A. Sharma, K. K. Paliwal, A. Dehzangi, J. Lyons, S. Imoto, and S. Miyano. A strategy to select suitable physicochemical attributes of amino acids for protein fold recognition. *BMC bioinformatics*, 14(1):1, 2013.

[66] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In *EMNLP-CoNLL*, 2012.

[67] R. Socher, Q. V. Le, C. D. Manning, and A. Y. Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics (TACL)*, 2014.

[68] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2011.

[69] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing (EMNLP)*, 2013.

[70] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7(Jul):1531–1565, 2006.

[71] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems (NIPS)*, 2014.

[72] Y. Taguchi and M. M. Gromiha. Application of amino acid occurrence for discriminating different folding types of globular proteins. *BMC bioinformatics*, 8(1):1, 2007.

[73] K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[74] A. N. Tegge, Z. Wang, J. Eickholt, and J. Cheng. Nncon: improved protein contact map prediction using 2d-recursive neural networks. *Nucleic acids research*, 37(suppl 2):W515–W518, 2009.

[75] S. Tokui, K. Oono, S. Hido, and J. Clayton. Chainer: a next-generation open source framework for deep learning. In *Workshop on Machine Learning Systems (LearningSys) in Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.

[76] M. Tsubaki, K. Duh, M. Shimbo, and Y. Matsumoto. Modeling and learning semantic co-compositionality through prototype projections and neural networks. In *Empirical Methods on Natural Language Processing (EMNLP)*, 2013.

[77] M. Tsubaki, K. Duh, M. Shimbo, and Y. Matsumoto. Modeling and learning semantic co-compositionality through prototype projections and neural networks. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing (EMNLP)*, 2013.

[78] P. D. Turney. Domain and function: A dual-space model of semantic relations and compositions. *Journal of Artificial Intelligence Research (JAIR)*, 2012.

[79] M. Vassura, L. Margara, P. Di Lena, F. Medri, P. Fariselli, and R. Casadio. Ft-comar: fault tolerant three-dimensional structure reconstruction from protein contact maps. *Bioinformatics*, 24(10):1313–1315, 2008.

[80] Z. Wang and J. Xu. Predicting protein contact map using evolutionary and physical constraints by integer programming. *Bioinformatics*, 29(13):i266–i273, 2013.

[81] P. Wu, S. C. Hoi, H. Xia, P. Zhao, D. Wang, and C. Miao. Online multimodal deep similarity learning with application to image retrieval. In *Proceedings of the ACM international conference on Multimedia*, 2013.

[82] E. P. Xing, M. I. Jordan, S. Russell, and A. Y. Ng. Distance metric learning with application to clustering with side-information. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS)*, 2002.

[83] T. Yang, V. Kecman, L. Cao, C. Zhang, and J. Z. Huang. Margin-based ensemble classifier for protein fold recognition. *Expert Systems with Applications*, 38(10):12348–12355, 2011.

[84] H. Zhang, T. Zhang, J. Gao, J. Ruan, S. Shen, and L. Kurgan. Determination of protein folding kinetic types using sequence and predicted secondary structure and solvent accessibility. *Amino acids*, 42(1):271–283, 2012.

[85] J. Zhao, T. T. Zhu, and M. Lan. Ecnu: One stone two birds: Ensemble of heterogenous measures for semantic relatedness and textual entailment. In *SemEval*, 2014.

[86] J. Zhuang, I. W. Tsang, and S. C. Hoi. Two-layer multiple kernel learning. In *AISTATS*, pp. 909–917, 2011.

List of Publications

Journal Papers (refereed)

- Masashi Tsubaki, Masashi Shimbo, and Yuji Matsumoto, “Protein Fold Recognition with Representation Learning and Long Short-Term Memory”, IPSJ Transactions on Bioinformatics, Vol.10, pp.2-8, January 2017.
- 椿 真史, 新保 仁, 松本 裕治, “意味構成のための非線形類似度学習”, 人工知能学会論文誌, Vol.31, No.2, p.O-FA2_1-10, June 2016.

International Conferences (refereed)

- Masashi Tsubaki, Kevin Duh, Masashi Shimbo, and Yuji Matsumoto, “Non-Linear Similarity Learning for Compositionality”, In Proceeding of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16), pp. 2828-2834, Phoenix, Arizona, USA, February 14, 2016.
- Masashi Tsubaki, Kevin Duh, Masashi Shimbo, and Yuji Matsumoto, “Modeling and Learning Semantic Co-Compositionality through Prototype Projections and Neural Networks” In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp.130-140, October 2013.

Domestic Conferences (non-refereed)

- 椿真史, 新保仁, 松本裕治, “タンパク質構造予測のための表現学習”, 情報処理学会 第 44 回バイオ情報学研究発表会, Vol.2015-BIO-44, No.2, pp.1-6, December 2015.
- 椿真史, Kevin Duh, 新保仁, 松本裕治, “Deep Kernel を用いた高次元空間への階層的な写像とその最適化” 人工知能学会全国大会 (JSAI2015) 論文集, 3O1-13in, pp.1-4, June 1, 2015.
- 椿真史, Kevin Duh, 新保仁, 松本裕治, “意味と構造の統一的なカーネル埋め込みによる非線形類似度学習” 人工知能学会全国大会 (JSAI2015) 論文集, 4K1-2, pp.1-4, June 2, 2015.

- 椿真史, Kevin Duh, 新保仁, 松本裕治, “一般二項定理による多項式カーネルの拡張と学習” 言語処理学会第21回年次大会発表論文集, 京都大学, pp.676-679, March 18 2015.
- 椿真史, Duh Kevin, 新保仁, 松本裕治, “意味と構造の構成演算と類似度学習における非線形性” 情報処理学会 第220回自然言語処理研究会, Vol.2015-NL-220, No.10, pp.1-, January 2015.
- 椿 真史, Kevin Duh, 新保 仁, 松本 裕治, “生成語彙論における共構成のモデル化と意味の合成性を内在する単語ベクトルの教師なし学習” 情報処理学会研究報告 第213回自然言語処理研究会, Vol.2013-NL-213, No.3, pp.1-9, September 2013.