An Unsupervised Approach to Relatedness Analysis of Legal Language

by

Ying Wang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2018

Abstract

Learning distributed representations of sentences and analyzing semantic similarity between sentences is one of the essential works in the field of Natural Language Processing. In the domain of legal language, the future of Artificial Intelligence-related legal-tech applications is very promising. This thesis comprises a very detailed investigation of distributional representations of words and sentences, and the related machine learning and deep learning techniques. Then, we proposed an innovative approach, Word2Sent, for measuring the degree of similarity between sentences. The proposed model is completely in an unsupervised manner. Thus, it can be well applied with unlabeled data. An enhancement of the other unsupervised sentence embeddings model, SIF-model, is made by this thesis. Demonstrated by multiple experiments, our proposed model can effectively work with long legal sentences on several textual similarity tasks.

Acknowledgements

I would first like to thank my supervisor Professor Liang-Liang Xie of the Department of Electrical and Computer Engineering at the University of Waterloo. The door to Prof. Xie's office was always open whenever I ran into a trouble spot on my research. Thanks to his meticulous teaching, leading me to the right direction, and all the support during my master's study. Thanks for the precious guidance and advices from Prof. Xie, which will have an immeasurable impact on my future life.

I would like to thank Professors Alfred Yu and Pin-Han Ho of the Department of Electrical and Computer Engineering at the University of Waterloo for their valuable remarks and suggestions on this work. I would like to thank the Data Science and NLP experts, Sarah Wang and Wei Sun, at Beagle.ai. Without their supports and assistance, this work could not have been successfully conducted. I would also like to acknowledge Zirui Wu at the University of Waterloo for his solid verifications on the experiments and practical comments on this work.

I would also like to thank my best friend, Chang Cui, at the University of Waterloo. Whenever I felt distressed, she always brought brightness and delight to my life. Whenever I felt cheerful, she was always the first audience to share the joyfulness. I appreciated every minute we spend together and the measureless togetherness we created together.

Last but not least, I would like to thank my parents, Yimin Cai and Zhiyi Wang, for their unlimited support and understanding. Without their meticulous care and sincere encouragement, I will not be able to reach the current stage of life.

Thanks to all of you. You make my days at the University of Waterloo unforgettable.

Dedication

This thesis is wholeheartedly dedicated to my beloved parents.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Artificial Intelligence (AI) is a comprehensive term that defines a collection of technologies intended to perform tasks that once demanded human intelligence, intercommunication, and decision-making. Among the more notable utilization of AI technology are speech and image recognition, autonomous cars, computer-aided medical diagnosis, algorithmic trading, market analysis, and robotics. In the circumstances of legal service, there are dozens of applications using AI technology to support services such as legal data research, contract review and management, intelligent interfaces, and mining litigation data for strategic insights. One major branch included in the broad AI-driven applications is the semantic understanding of the legal document. Many further, more advanced and complex services are based on the proper and reasonable interpretation of the given natural language. For example, legal Question & Answer (Q&A) systems require to correctly interpret the query information a client has typed into a dialogue box, and the designed algorithm provides the appropriate solution that is tailored to the client's needs to guide the client in finishing primary legal documents. Contract review systems analyze the contract and extract valuable information, then feedback the client with practical assistance such as synonyms recommendation or sentence rationality detection. Therefore, the fundamental concern is to let the machine interpret a word, a sentence or a document in a human-like manner.

An AI-based legal technology in general applies Machine Learning and Natural Language Processing (NLP) to clean, tag, and structure raw litigation data so it can be efficiently searched and reveal new perspicacity. Machine Learning is disposed to search through vast amounts of data to recognize patterns and proper operation based on those patterns. It uses algorithms that can "learn" from data in an iterative manner and make predictions by constructing a model from inputs [1]. NLP provides a considerable ability to read natural language, i.e., normal text that we all use. It helps to process human language as it is written or spoken, without requiring customers to modify the syntax to adapt to system requirements. For example, the input of an NLP-based system could be a contract, and the output is the interpreted knowledge and the key phrases of the document, or the part that different from the standard and regular clauses according to the user's expectation. Alternatively, it could be applied to understand a legal query from the customer and explore through enormous legal datasets to feedback any related information not only regarding keywords appearing in the context but also the semantic concepts matching with the question.

No AI application maintains the inimitably human ability to integrate varied skill sets and make conscious decisions the way lawyers do, nor can they articulate why a particular clarification or judgment is the correct one. However, these applications indeed involve processing and handling enormous amounts of data quite quickly and assist with the automation of specific low-level repeatable tasks. Legal professionals and their valuable specialized knowledges will not be replaced by these applications, but AI will definitely and profoundly make alternations to the conventional legal services delivery manner. Lawyers can leverage AI-based legal tools that efficiently work with massive datasets to distinguish drafting flaws, predict likely legislative consequences, and enhance the quality of investigation. Law firms can reduce the financial plan on the business such as reviewing contracts, assessing risks, and polishing writing with the assistance of AI-powered legal tools. Some commercial enterprises also benefit from the tidal wave of AI. Based on the fact that more and more legal professionals are playing the key role in the corporation, such as business partners, strategy planners, or business advice counselors, these roles can be effectively substituted for the AI-based legal tools. Such adoption of technology can significantly help the business to make more analytical plans and decision, and enhance the chance to achieve innovation.

Human communication is frustratingly uncertain at all times, not only in the field of legal, such as colloquialisms, abbreviations, and misspellings. All these inconsistencies make the machine analyze the natural language tedious. However, in the domain of legal documents, more impact factors need to be considered regarding the intrinsic features of the legal language. One of the most significant characteristics of the legal language is that those clauses in a contract have to be objective, to be enforceable. So, to be written objectively could sort of narrows down the number of ways that a sentence can be constructed from an English perspective. Because of the strictness of structure, legal language is composed of relatively long sentences. Also, because of the normative nature of usage, the vocabulary and the commonly used phrases are relatively fixed and standardized.

With the rapid development of Machine Learning and Deep Learning in recent decades, the research status of semantic analysis also progresses immeasurably. The breakthrough in the distributed representation of words solidifies the basis of semantic analysis. Many different unsupervised training methods, which generates word embeddings from unstructured data, make the upcoming high-level semantic analysis models achieve the state-of-art performance. As for the higher-level perspective, it is critical to find out an efficient way to represent the semantic meaning of the longer pieces of text, such as phrases, sentences, or documents, to achieve a more reasonable interpretation. A bunch of wellperformed supervised learning approaches has been developed to solve a particular or narrow range of tasks, given the sizeable qualified training data, which requires high labor costs. The typical representatives with the excellent performance of these deep supervised models include Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short-Term Memory Network (LSTM), and some other variants of language models. On the other hand, some statistics-inspired approaches, such as Latent Semantic Indexing (LSI) and Matrix Factorization, can learn from the more massive unlabeled datasets, which gains apparent advantages in the unsupervised tasks. However, some models perform better in short sentences, and when the sentences become longer, the accuracy decreases quickly. Thus, such models are not the best candidates in the legal field. Some models rely heavily on a significant volume of labeled data. The model that wins people's favor is a model that produces excellent outcomes through unsupervised learning. Therefore, it is essential to pick a sentence embedding model that is specifically applicable to legal language, or a better but more difficult solution is to find a language modeling method that is suitable for all types of documents.

In this thesis, we review some of the sentence embedding methods in both supervised and unsupervised manner, especially analyze the rationale behind each model. In addition to the fundamental and relevant knowledge and techniques that would be utilized in the involved models, the distributed representation learning of word embeddings, which is almost the foundation of every sentence or higher-level semantic models, is explained in detail. The main contribution of this paper can be summarized as follows: On the one hand, in a particular circumstance of the legal domain, a novel evaluation method for sentence similarity analysis is proposed, with the experimental evidence on different datasets from the different domains for its validation and outstanding performance. On the other hand, a slight modification is applied to a baseline sentence embedding model named SIF [2], which is studied in detail, and several experiments would show that the performance is improved. Lastly, the conclusion and some ideas about the future work improvement shall be discussed.

Chapter 2
Background

2.1 Introduction to Representation Learning

2.1.1 The Overview of Representation Learning

In many areas of our lives, an iron rule is to ensure the quality of information processing results. The difficulty of information processing depends on how we represent the information. Therefore, the representation of data has become an essential part of the Machine Learning. At the very beginning of Representation Learning, people used their prior knowledge of the data to carry out some features. However, the cost of this approach is a high amount of workforce and cannot extract and organize complex information from the data if the original data is massive and complicated.

Artificial Intelligence is the final goal pursued by many Machine Learning algorithms, and the fundamental requirement of Artificial Intelligence is to understand the world around us fully. With the goal of expanding the scope and the practicality of Machine Learning algorithms, it is extremely desirable to make learning algorithms less dependent on the feature engineering. It is believed that a better representation of data and identifying the hidden explanatory factors from a complex data is the cornerstone of achieving a high-performance algorithm. In the supervised learning tasks, a good representation of the data is the one that can make it easier to build a classifier or predictor.

To understand Representation Learning better, we assume that the underlying explanatory factors cause the data to form a particular distribution. Many of these factors are the generalization of some salient factors by similar learning rules. For example, in a figurative statement, there may be several light sources intertwined with each other to form the whole light perception of the entire image. The light, shadows and the relative position of objects combine to build a more complex composition of the image. To clarify the existence of every boundary of objects, we hope to decompose these constituents and make a clear distinction between the blended light and shadow [3]. After investigating various methods, people ultimately adopted the way of leveraging data itself by learning the representations of the underlying factors of the data, so as to separate the various "light sources in the image." In many tasks, there are vast quantities of natural data available introducing the common sense for disentangling the underlying factors of variation.

From another point of view, the purpose for Representation Learning is to learn invariant features of the observed data, which can find out the features insensitive to variation in the data that are less informative to the tasks. However, the most intractable problem is that we have no prior knowledge to infer which features are concerned with the tasks. With the consideration of this problem, the most reliable method is to disentangle as many factors as possible, discarding those factors with little information of the data [3]. For instance, the directions with the least information in variation are abandoned in dimensionality reduction.

One of the essential concepts related to the application of Representation Learning is Transfer Learning, which indicates the ability for a representation to exploit the sharing features or common statistical characteristics between different learning tasks [4]. In other words, with Transfer Learning, the representation learned from one task can be generalized with improvement in another task. A good representation is the one that is capable of disentangling the underlying explanatory factors of variation, and these factors may be relevant to the factors of some other different tasks. The objective of Transfer Learning is sharing the statistical strength of each task to explain either inputs distribution or output distribution, so that helps model generalization [4].

2.1.2 Distributed Representation

In 1986, Geoffrey Hinton introduced the concept of distributed representation, refers to a many-to-many relationship between two types of representation [5]. Concretely, if we use neurons to represent concepts in a distributed representation manner, each concept is represented by several neurons, while each neuron participates in the representation of multiple concepts. Distributed representation is one of the most essential methods for Representation Learning. Many learning models, such as the traditional feed-forward neural network with multiple hidden units or probabilistic models with multiple latent variables, all take advantage of the concept of distributed representation. As we have discussed that a good representation can disentangle the underlying factors to explain the distribution of the input data, the distributed representation qualifies with that requirement. If distributed representation uses k different values to describe n different features hidden in the data, then k^n different concepts are created in the representation space, with each representing one underlying explanatory factor of variation.

Comparing with one of the typical non-distributed representation methods, the so-called symbolic representation, where n different concepts are represented by n different symbols in the input space, the advantage of distributed representation becomes more magnificent. For example, in many word embeddings tasks, a common representation is the one-hot vectors, using a binary vector with n bits (keep the assumption that n different symbols are given in input) to represent each concept. Although it is straightforward for representation and interpretation, the vectors would become extremely sparse when the number of distinct symbols in the input data increases (imagine a vector with 10,000 dimensions but only one entry equals to 1 and others are all 0). Thus, it is inefficient to capture the underlying factors for explaining the input data. Instead, distributed representation uses the whole vector with not that large dimension to represent a symbol of input space. Every element of the vector denotes one underlying factor that influences the location of that input example in the high-dimensional concept space. The relative distance between two vectors represents the degree of their similarity. Thus, in most of the real-world tasks with incredibly massive data need to be handled, the application of distributed representation would become one of the optimal choices.

2.1.3 Deep Representation

Human defines objects with a clear hierarchy. Some high-level and abstract concepts are composed of multiple microscopic and figurative concepts. This assumption is where the concept of deep representation based. Although challenges for deep representation exists, such as computational inefficiency, there indeed some unique advantages carried along with it.

One distinct advantage is the reuse of features [4], which builds a hierarchical architecture of features and takes full advantage of distributed representations. For example, given a graph, the depth of the graph indicates the length of the longest path from the source node to the destination node. With the depth increases, the ways to reuse different portions of paths increases exponentially. If changing the mathematical operation of each node, such as product, weighted sum, or logic gates, the depth of the graph changes by a constant factor [4]. It proves deep representation structure could be exponentially efficient than “shallow” representations, regarding these families of functions [6]. Furthermore, if the representation of these types of functions learns via fewer parameters, then fewer training data is required, which also improves the computational efficiency and the statistical strength of the algorithm.

The other considerable benefit is that within the representation hierarchy, the deep structure helps to learn higher-level features since the abstract higher-level concept is often composed of numerous concrete lower-level concepts. For example, an object detection tasks with a CNN, the first several layers of the model may perform as the simple objects detector such as edges. Then by composing these detected single objects in the later layers of the model, more and more complex features could be learned such as shape and color.

2.1.4 Autoencoder

Among various categories of Representation Learning models, autoencoder is a feature learning technique that learns a direct encoding to map from the given input to the distributed representation space. By definition, autoencoder is a neural network aimed to replicate the input to the output, consists of an encoder and a decoder. As shown in Figure 1, the encoder function $h = f(x)$ outputs the encoding to the hidden layer which can be perceived as the representation of the input, while the decoder function $r = g(h)$ takes the output of the encoder as input and generates a reconstruction. The architecture is illustrated as below.

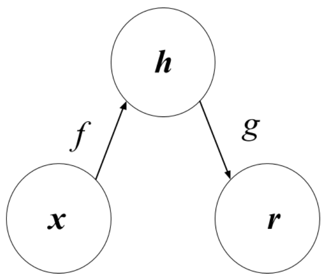


Figure 1 The structure of autoencoder, where $\boldsymbol{x}$ is the input and $\boldsymbol{r}$ is the output. The encoder is represented by $\boldsymbol{f}$, while the decoder is $\boldsymbol{g}$.

Briefly, the encoder f maps the input x to the internal representation h, and the decoder g maps h to reconstructed output r. During the training process, the set of parameters θ of encoder and decoder are updated simultaneously through minimizing the reconstruction error, the difference between the output and the original input.

$$\mathcal{J}_{AE}(\theta) = \sum_t L(x^t, g_\theta(f_\theta(x^t))) \tag{1}$$

Just as the training method for the traditional neural network, the optimization process is carried out by stochastic gradient descent or some other optimization algorithms, and both encoder and decoder are applied with non-linear transformation:

$$f_\theta(x) = s_f(b + Wx) \tag{2}$$

$$g_\theta(h) = s_g(d + W'h) \tag{3}$$

, where s_f and s_g are some types of non-linear function, such as $sigmoid$ or $tanh$ function. Thus, the learning parameters consists of W, W', b and d, where b and d are called bias vector of encoder and decoder, and W, W' are weight matrices.

With the objective described above, often, autoencoder prone to learn to identity function from the input to output, which indeed perfectly copy the data. But that is not what we want. A good generalization of the autoencoder is to achieve low reconstruction error of the validation and test examples, not just training examples. Thus, instead of learning to copy the input to output without any error, the goal of autoencoder is to copy in a not-so-accurate way to force the encoder to learn the useful properties of data and capture the underlying factors. Various types of autoencoder were proposed in the past several years, including regularized autoencoder, denoising autoencoder, and contrastive autoencoder.

Autoencoder has a broad application in the field of dimensionality reduction. Since low-dimension representation results in low memory requirement and high computational efficiency, and data with similar meaning can be located closer in the representation space with dimension reduction techniques, it would be a huge convenience and improvement for many downstream tasks. For example, in information retrieval, autoencoder could help in mapping the high dimensional input data into the internal representation space which has relatively lower dimension, and this low dimension space searches significantly efficient.

2.2 Distributed Representation for Word Embeddings

In NLP, using distributional representation to describe the meanings of words is the most crucial step before any downstream tasks. At the very beginning, people used a taxonomy like WordNet that has hypernyms (is-a) relationships and some synonym sets to represent the meaning of a word. This method has too many limitations, such as requiring too much human labor to create and maintain, hard to update, hard to compute word similarity accurately. Thus, it was shortly replaced by some other approaches. Distributional representation is based on the underlying idea that “a word is characterized by the company it keeps” claimed by Firth [7] and the distributional hypothesis by Harris [8] in 1954, saying that words have similar meanings if used in similar contexts. Distributional word vectors or so-called word embeddings indicate the “most” essential information of the context around the particular word, and these vectors with the characteristics of the neighbors of a word make the similarity measurement among words possible. Conceptually, word embeddings is a general term used in language modeling and feature learning in NLP by mapping the high-dimensional space in which a word is located in a vocabulary to a low-dimensional continuous vector space consists of dense of real numbers. There are generally two categories of approaches to generate distributed representation for words. One is applying dimensionality reduction after constructing word co-occurrence matrix, the other is learning representation via a shallow neural network.

Word embeddings are indispensable in many NLP tasks. A good word embeddings model can properly capture the semantic and syntactic relations of words in the corpus. The typical example is word vectors could simulate the semantic relationship of words regarding human's interpretation by simple addition and subtraction operations, such as

$$vector(\text{“}king\text{”}) - vector(\text{“}queen\text{”}) = vector(\text{“}man\text{”}) - vector(\text{“}woman\text{”})$$

Thus, word embeddings can provide practical assistance for the subsequent downstream tasks such as sentiment analysis, topic classification, and sentence similarity.

2.2.1 Traditional Count-based Model

In the early 20th century, some distributional similarity-based data representation algorithms were introduced, such as Principal Component Analysis (PCA, proposed by K. Pearson in 1901 [9]) and Linear Discriminant Analysis (LDA, proposed by R. Fisher in 1936). Both are designed to learn the low-dimensional representation of data with a linear projection. As mentioned earlier, if the data is represented in a high-dimensional way such as one-hot vectors, there would be several issues related to statistical efficiency, memory storage as well as the curse of dimensionality [10].

PCA is one of the data compression methods which effectively reduces the dimension of one space into another subspace with lower dimensionality by a linear projection and reserves the salient properties of data as much as possible [9] [11]. In this particular subspace, the variance of the data of all directions is maximal. Thus, this subspace is the “best” linear space [12]. The original input data x is mapped into a new space of representation z, whose elements are statistically independent (no linear correlation) with each other. Conversely, new data in the representation space can be reconstructed to the original data with minimal error. Mathematically, there are two ways of implementing PCA, Singular Value Decomposition (SVD) and Eigen-Decomposition.

Specifically, we denote the input data as the matrix X with $m \times n$-dimension, and we assume that the input data has zero mean in all directions, $E[x] = 0$.

The unbiased sample covariance matrix of X is given by:

$$Var[x] = \frac{1}{m-1} \times X^T X \tag{4}$$

The goal of PCA is to find a linear transformation $z = x^T W$ so that $Var[z]$ is diagonal.

Through singular value decomposition, matrix X can be factorized into the product of three matrices:

$$X = UDV^T \tag{5}$$

where the columns of U and V consist of the left and right singular vectors, respectively, are orthonormal, and the matrix D is diagonal with positive real entries indicating the singular values of X. The principal components of matrix X are given by the right singular vectors V. Then, we can express the variance of X as:

$$\begin{aligned} Var[x] &= \frac{1}{m-1} \times X^T X \\ &= \frac{1}{m-1} \times (UDV^T)^T (UDV^T) \\ &= \frac{1}{m-1} \times VD^2V^T \end{aligned} \tag{6}$$

$U^T U = I$ since the matrix U is defined to be orthonormal. If we take the linear transformation $z = x^T W$, we can ensure that the covariance of matrix z is diagonal as required:

$$\begin{aligned} Var[z] &= \frac{1}{m-1} \times Z^T Z \\ &= \frac{1}{m-1} V^T X^T X V \\ &= \frac{1}{m-1} V^T V D^2 V^T V \\ &= \frac{1}{m-1} D^2 \end{aligned} \tag{7}$$

, $V^T V = I$ since the matrix V is defined to be orthonormal.

Therefore, when projecting the data x through a linear transformation W to z, we obtain representations as results with a diagonal covariance matrix as given by D^2 , which indicates that the elements of z are statistical independent with each other. The dimension of vector z represents some linear combinations of dimensions of the input data x, and these dimensions are ordered in the way consistent with their variance to the data. Recall that the objective of Distributed Representation Learning is to disentangle the underlying explanatory factors hidden behind the data, and PCA exactly provides a method to disentangle the data by rotating the input space via W to maximize the variance of the new representation space.

However, every coin has two sides, the most obvious shortcomings of PCA can be concluded as follows: Firstly, when the scale of the input data is huge (say millions of words or documents), PCA takes all data into account for the decomposition, which is pretty infeasible. Secondly, this method is not adjustable once the input dataset gets updated, such as adding a new word to the corpus. PCA has to re-compute the entire co-occurrence matrix and re-perform every step. Thirdly, PCA only works well under certain assumptions. For example,

- It assumes that the data is linear correlated so that PCA can find orthogonal projections of the data containing the highest variance. If the input data is not linearly correlated, for example, PCA fails to work under input like $y = t \times \sin(t)$.
- It assumes that the principal components are orthogonal, which is a restriction to find projections with the highest variance. For example, in Figure 2, when the non-orthogonal principal components are required to represent data, PCA would be failed not like the other methods such as Independent Component Analysis (ICA) [13].

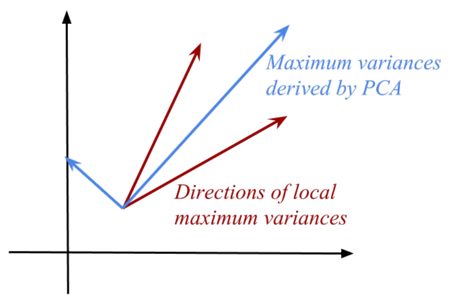

Maximum variances derived by PCA
Directions of local maximum variances

Figure 2 The blue color vectors are principal components. However, the actual maximum variance directions are red color vectors.

- It assumes that the principal components with low variance are not essential to the representation, such as the removable noise. In fact, the properties of data are not always such ideal. Sometimes, these low-variance components count some significant features of data.

Even though limitations of PCA exist, it still acts as a prestigious dimensionality reduction technique which gives great inspiration to many variants such as ICA [13] and Random Indexing [14], for addressing those issues brought by standard PCA.

A popular distributional representation approach, Latent Semantic Analysis (LSA), uses PCA to decompose large matrices that capture statistical information for the data [15]. LSA is broadly applied for relationship analysis between documents and the terms they contain, by representing documents and terms with the concepts. Recall the distributional hypothesis, words with similar meaning will also not be far from each other, LSA produces a sparse "term-document" matrix containing word counts per document, with rows representing unique words and columns representing documents. The matrix describes the occurrences of terms in documents. Each element in this sparse matrix has a weight indicating the relative importance of the specific word in the corresponding document. A typical weighting scheme is called Term Frequency-Inverse Document Frequency (TF-IDF) [16], where the weight of an element is proportional to the number of appearing times of the word in the document. After applying PCA to the matrix to reduce the dimension, the resulting low-dimensional matrix can be applied to many applications for semantic analysis. The similarity of any two rows is determined by a measurement such as Cosine Distance, or Euclidean Distance between two vectors.

However, shortcomings indeed exist with LSA. For example, not every dimension of the resulting low-dimensional matrix can give the reasonable interpretation by human; it fails with polysemy or synonymy existing since the words and concept space is in one-to-one relationship; the term-document matrix is obtained from the Bag-of-Words (BOW) model, where the order of words in the observed text is neglected [17].

A similar method to LSA, one named Hyperspace Analogue to Language (HAL) [18], uses a kind of "term-term" matrix instead of "term-document" matrix of LSA to represent the word co-occurrence knowledge, where rows and columns represent unique words in the corpus. Each entry of the matrix represents the number of times that the word given by the row appears in the context of another word given by the column. One observation of the application of this model is the comparatively weak performance in word analogy tasks since it can hardly handle the contribution from some highly frequent words such as “the”, “and”. When computing the degree of similarity, those common fixed phrases would have a tremendous bias on the result rather than the meaning of words itself. Pointwise Mutual Information (PMI) [19] are designed to address this issue brought by HAL. It indicates as the log ratio of the joint probability, and the product of the marginal probabilities of two words, which can be expressed in the mathematical formula:

$$PMI(w,c) = \log\left(\frac{P(w,c)}{P(w)P(c)}\right) \approx \log\left(\frac{number\ of(w,c) \times |D|}{number\ of\ (w) \times number\ of\ (c)}\right) \tag{8}$$

, where w belongs to V_w and c belongs to V_c, representing the collection of words in the vocabulary and their contexts, respectively; D is the collection of observed word-context pairs.

Although PMI-based methods effectively control the influence of the most frequent words to results, the common downside is that it fails to deal with the case of rare context. For example, for the given word w, if the context word c is so infrequent that it only occurs once with word w, then it will result in a relatively high PMI score since the term $P(c)$ in the denominator is a very small value.

2.2.2 Neural Network-based Model

The neural network-based approaches directly learn distributed representations for words that help to predict the local context of words within a fixed-sized window. In 2003, Bengio et al. proposed an architecture for estimating Neural Network Language Model (NNLM) [20]. This model associate words in the vocabulary with corresponding distributed word feature vectors and expresses the joint probability function of the given word sequences. The model learns the word feature vectors and the parameters of the probability function simultaneously. The underlying architecture is a shallow feed-forward neural network consists of a linear projection layer and a non-linear hidden layer. The input data is one-hot vectors, the dimensionality of each vector is V, where V is the number of words in the vocabulary. Since only N words are fed as input to the model at one time, there is not too much computation before the projection layer. After “embedding” the words into the projection layer, the matrix becomes denser, so it needs more complex computation. Over the past several years, many variants on this model have been proposed, such as the Recurrent Neural Network Language Model (RNNLM), which makes use of the “short-term” memory of the recurrent network to fit with the more complicated structure of input data [21]. Although differences exist among these models, the core idea is to build a reliable language model that would give a comprehensive and logical sentence a high conditional probability. When calculating the conditional probability, the fixed-size window n is chosen as the range of the contextual words to make it tractable to model the sentence, since the length of a sentence can be changeable. In the case of $n = 1$ (unigram), it bases on the assumption that the word occurrences are utterly independent of each other. However, this assumption seems not make so much sense, since the surrounding contexts of a word are highly related in human's natural language. Thus, it makes more sense to set $n = 2$ (bigram) or $n = 3$ (trigram), or some larger values and to learn from the conditional probability.

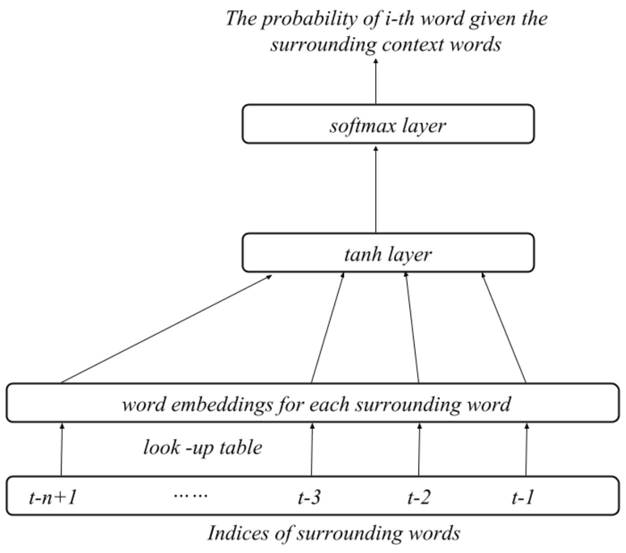
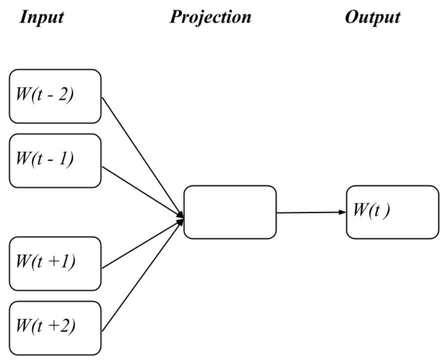

The probability of i-th word given the surrounding context words
softmax layer
tanh layer
word embeddings for each surrounding word
look -up table
t-n+1
……
t-3
t-2
t-1
Indices of surrounding words

Figure 3 Neural architecture.

2.2.2.1 Word-level Word Embeddings

A significant breakthrough in the development of word embeddings occurred in the year of 2013. Mikolov et al. proposed two models using contextual information to effectively learn high-quality word representation from massive amounts of unstructured text data. One is called Continuous Bag-of-Words (CBOW) model, the other is Continuous Skip-gram model (Skip-gram), and both are the shallow neural networks [22] [23]. They notably optimize the computational complexity when comparing with the traditional feed-forward neural network without dense matrix multiplications. CBOW calculates the conditional probability of a target word given by the context words surrounding it within a window of size k. While the Skip-gram model performs in the opposite direction, by predicting the surrounding context words within a window of size k given the central target word.



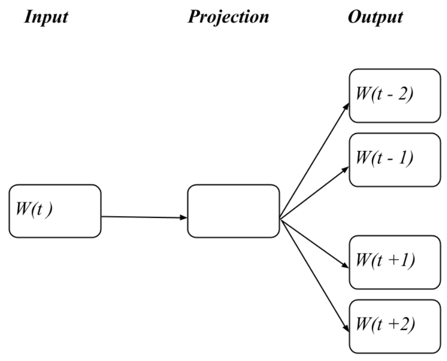


Figure 4 The architecture of CBOW model.

Figure 5 The architecture of Skip-gram model.

As shown in Figure 4, the CBOW model is simply a fully connected neural network with only one hidden layer. Each word has two representation matrices $\mathcal{W} \in \mathbb{R}^{N\times V}$ and $\mathcal{W}' \in \mathbb{R}^{V\times N}$, representing the input word matrix and the output word matrix, respectively, where n is a variable size indicating the size of the embedding space, V is the number of words in the vocabulary. In the input matrix $\mathcal{W}$, the vector v_i in the i^{th} column represents the n-dimensional input word embeddings of the word w_i. Similarly, the vector u_j in the j^{th} row of the output matrix $\mathcal{W}'$ represents the n-dimensional output word embeddings of the word w_i.

The input layer takes the one-hot vectors

$$x^{c-m}, \dots, x^{c-1}, x^{c+1}, \dots, x^{c+m}$$

as the input context words with window size m. Then, the one-hot vectors are projected to the embedded vectors layer, via the input matrix $\mathcal{W} \in \mathbb{R}^{N\times V}$, and the embedded word vectors for the context are obtained as:

$$v_{c-m} = \mathcal{W}x^{c-m}, \dots, v_{c+m} = \mathcal{W}x^{c+m} \tag{9}$$

Then calculating the average vector over these contextual word embeddings to get

$$\hat{v} = \frac{v_{c-m} + v_{c-m+1} + \cdots + v_{c+m}}{2m} \tag{10}$$

as the n-dimension hidden layer. The hidden layer is connected with the output layer by the output matrix $\mathcal{W}' \in \mathbb{R}^{V\times N}$, therefore to obtain a score vector:

$$z = \mathcal{W}'\hat{v} \tag{11}$$

A Softmax function is added on the top of the output layer, transforming the individual scores on each category to the probability distribution over all words in the vocabulary,

$$\hat{y} = softmax(z) \tag{12}$$

to match the true probabilities y, which is one-hot vector of the actual word. The goal of the model is to learn the input matrix and output matrix of the words in the vocabulary, with a loss function based on Cross-Entropy between the predicted label $\hat{y}$ and the ground truth label y, which is

$$H(\hat{y}, y) = -y_j\, log(\hat{y}_j) \tag{13}$$

Therefore, the overall objective function is formulated as:

$$\begin{aligned} minimize\, J &= -\log P(w_c | w_{c-m}, \dots, w_{c-1}, w_{c+1}, \dots, w_{c+m}) \\ &= -logP \\ &= -\log\left(\frac{\exp(u_c^T \hat{v})}{\sum_{j=1}^{V} \exp(u_j^T \hat{v})}\right) \\ &= -u_c^T \hat{v} + log \sum_{j=1}^{V} \exp(u_j^T \hat{v}) \end{aligned} \tag{14}$$

Then using stochastic gradient descent and backpropagation [24] to update all relevant word vectors u_c and v_j.

For the Skip-gram model, illustrated in Figure 5, there is no big difference from the CBOW model regarding the model structure and optimization, except that the input word is the central word, the output layer predicts the surrounding words. For example, the objective of the Skip-gram model is to minimize the average negative log probability:

$$\begin{aligned} minimize\, J &= -logP(w_{c-m}, \dots, w_{c-1}, w_{c+1}, \dots, w_{c+m} | w_c) \\ &= -log \prod_{j=0, j\neq m}^{2m} P(w_{c-m+j} | w_c) \\ &= -log \prod_{j=0, j\neq m}^{2m} P(u_{c-m+j} | v_c) \end{aligned} \tag{15}$$

$$= -log \prod_{j=0, j\neq m}^{2m} \frac{\exp\left(u_{c-m+j}^{T} v_c\right)}{\sum_{k=1}^{V} \exp\left(u_k^T v_c\right)}$$

$$= - \sum_{j=0, j\neq m}^{2m} u_{c-m+j}^{T}\, v_c + 2mlog \sum_{k=1}^{V} \exp\left(u_k^T v_c\right)$$

, where m is the size of the context window. It shows that larger contextual window size, which takes more words into account in every prediction, could achieve higher accuracy, and takes much more training time, however.

However, performing the Softmax in the loss function of the Skip-gram model is computationally expensive, since the Softmax function sums over all words in the vocabulary, which is intractable when the vocabulary size is enormous. Same year, Mikolov et al. proposed some novel techniques specially to speed up the training process [23]. One is using a Hierarchical Softmax (H-Softmax) to approximate the full Softmax and reduce the computation perplexity. Concretely, using a binary tree with the W words as its leaf nodes to reduce the evaluation perplexity from W to $log_2(W)$ to obtain the probability of a word. The second alternative is called Negative Sampling (NEG) [23], inspired by Noise Contrastive Estimation (NCE) [25]. Considering logistic regression classifier for a binary classification between target words and samples draw from noise, based on the ratio of probabilities of samples under the model and the noise distribution, which makes the training time independent of the vocabulary size [25]. Moreover, authors in [23] also claimed that Subsampling of frequent words can effectively deal with the imbalance between the rare and frequent words, for example, the most frequent words like "the", "a" may occur much more times than some relatively rare but more informative words. They defined that the probability of a word to be discarded during training time is proportional to its frequency in the training dataset by

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{16}$$

, where $f(w_i)$is the frequency of the word w_i and t is the threshold. This technique is also proven to be effective to boost the training efficiency and accuracy [23]. Another result found by their work is the property of compositionality of words, by which the word embeddings can be composed by the simple summation to generate a meaningful phrase embedding. Furthermore, the original words can be replaced by newly generated phrase embedding as a single token in the training dataset [23].

2.2.2.2 Character-level Word Embeddings

We have introduced Word2Vec, a model which learns a set of vectors to words by learning parameters of a shallow neural network. It is good at capturing the syntactic and semantic information among a long sequence of words. However, one of the severe challenges in NLP is the issue of Out-of-Vocabulary (OOV), which frequently occurs in morphologically rich languages, such as French, Spanish, and Finnish. Concretely, there are often dozens of different forms of deformation in a noun, and the same circumstance for verbs. Thus, some forms of words may rarely occur in the corpus but lots of useful information is involved. Word embeddings models like Word2Vec ignore the internal structure of words since the minimal unit for representation is a unigram. This issue can be effectively addressed by introducing the character-level information of words.

In past years, several methods were proposed to take the advantage of the morphological information for word embeddings, such as adding knowledge-based morphological features to word embeddings [26], obtaining word vector by different composition of morphemes [27] [28] [29], jointly learning method for morphologically Chinese characters [30], etc. Another type of approaches makes use of the high capacity architectures such as the deep neural network to learn various patterns of the combinations of the character sequences to words. For example, composing morphs into word embeddings via recursive models [31], using Bidirectional LSTM on characters to learn different types of word embeddings [32], and applying character-level word embeddings to machine translation tasks with RNN-based models [33] [34]. CNN also plays an important role in character-level word representation, related work such as character-level language modeling [35] and text classification [36], and the combination structure of CNNs and RNNs [37]. In 2016, Wieting et al. proposed CHARAGRAM, a character-based Representation Learning method training with paraphrase pairs, which embeds a character sequence by adding the vectors of its character n-gram followed by an elementwise nonlinearity [38].

In 2017, Bojanowski et al. from Facebook AI Research proposed a character n-gram representation model, fastText, which significantly enhances the performance of word embeddings in morphologically-rich languages by using character-level information and reduces training time on the large corpus. The model is an extension of the continuous Skip-gram model on word embedding, considering the representations for every n-gram of a word and representing the word as the summation of the composed n-gram vectors. For each word w, first we add special boundary symbols to the front and the end of the word, then create a bag of character n-gram of w including w itself, where n is a variable with a pre-defined range. Concretely, given a dictionary of n-gram of size G. Denote the set of character n-gram appearing in the given word w as $G_w \subset \{1, \ldots, G\}$, denote each n-gram g as z_g. Simply applied with the idea of compositionality among subwords, the word embeddings for word w can be represented by the sum of all the pairwise inner product between its n-gram vectors and the context vector, and the representations are shared across words. The scoring function is:

$$s(w, c) = \sum_{g \in \mathcal{G}_w} {z_g}^T v_c \tag{17}$$

The model uses a different objective function from the Skip-gram model:

$$\sum_{t=1}^{T} [\sum_{c \in C_t} l\big(s(w_t, w_c)\big) + \sum_{n \in N_{t,c}} l(-s(w_t, n))] \tag{18}$$

, where T is the length of a sequence of words, C_t is the window size of the target word, w_t is the target word vector from the corpus. With Negative Sampling, a set of data from noise which is denoted as $N_{t,c}$. Function $l(\cdot)$ represents the logistic loss function, function $s(\cdot)$ is precisely the scoring function above. This model is proven to outperform many word-level embeddings and other types of morphological representations with the issues as OOV. From experiments, the best size of n-gram is between 3 and 6 characters by the fact that n-gram of this length would cover a wide range of information well. However, with specific languages and specific tasks, the optimal choice of length ranges needs to be tuned appropriately.

2.2.3 Summary & Discussion

In this section, we reviewed the development of the distributed representation of word embeddings. From the count-based co-occurrence matrix approaches with one-hot representation to the low-dimensional distributional vectors with dimensionality reduction techniques, and finally arrives at the prediction-based methods, by learning the low-dimensional word vectors with a neural network. We introduced word embeddings models, and deeply analyzed their implementation process and the underlying mathematical theory. Through the series of iterations, the quality of the word embeddings models has made remarkable progress, which reflects in some downstream tasks such as word similarity analysis that relies heavily on word embeddings. We introduced in detail about the typical dimensionality reduction techniques, PCA, and discussed some other popular approaches such as HAL and PMI. These methods take the efficient usage of statistics so that they achieve the fast training process, but fail to scale with the variable-sized corpus. However, for those neural network-based models such as NNLM, CBOW, Skip-gram, and fastText, they can successfully generalize the model itself to corpus with any size and generate the representational vectors capturing more complex patterns and properties. Thus, it improves the performance of many downstream tasks. The disadvantage of this type of models is the inadequate and inefficient usage of the statistical information.

Moreover, with the neural network-based models, the performance would be boosted if the size of the corpus increases. Some hyperparameters in the model also influence the performance to some extent, such as the dimension of embedding vector and the number of negative samples. Too low dimension vectors would cause high bias and are not capable of capturing all useful properties in the corpus, while too high dimension may cause high variance problem since the model may capture some unexpected noise and cause negative influence for generalization. A comprehensive study of these two major methods for word embeddings by Levy et al. [39] shows that these hyperparameters of the models should be tuned explicitly according to different tasks, which would contribute expressively to the improvement of performance much more than using a better algorithm or a larger training corpus. They also show that the hyperparameters implemented in prediction-based models are transferable to be adapted and applied to the count-based methods. Practically, Mikolov et al. conducted meticulous experiments on different word embeddings models with various corpus and summed up many empirical rules that can be referred during the training process, such as de-duplicating sentences in large corpus, building the phrases in the pre-processing step, adding the position-dependent weights and sub-word features to the CBOW model [40]. All these tricks might be useful in boosting the accuracy of the word embeddings models.

Chapter 3
Deep Learning Techniques

3.1 Overview

Deep Learning is a class of techniques included in Machine Learning research community [41]. It consists of several successive multiple layers of non-linear operation nodes, and each layer takes the output from the previous layer as the input. Deep Learning algorithms are usually designed in a hierarchy structure, learning representations and extracting features of the input data with multiple levels of abstraction. In many tasks, such as speech recognition, object detection, language modeling, and transfer learning, a variety of state-of-the-art Deep Learning models has achieved the significant performance [42].

Early in 1965, Ivakhnenko et al. proposed the first supervised and deep feedforward multilayer perceptron working learning algorithm [43]. Six years later, an architecture of deep network with eight layers trained by a group method of data handling algorithm was described in another paper of Ivakhnenko [44]. In 1989, Yann LeCun et al. [45] applied the standard backpropagation algorithm to a deep neural network for a handwritten ZIP codes recognition system. The research on neural network gained popularity and reached the peak in the early 1990s. However, this boom of neural networks was quite short-lived. At that time, due to various difficulties, many attempts at training deep supervised neural networks did not feedback with positive results. Thus, people thought that Deep Learning is not an efficient way to solve the problem. Neural networks were soon replaced by some other classic Machine Learning techniques and became not so that popular until the modern Deep Learning renaissance that began in 2006 [3]. Some research groups began to concentrate on stacking unsupervised Representation Learning algorithms to gain deeper representation with the complicated structure of data [46] [47] [48], and many later. One turning point is the publications by Hinton et al., they proposed deep belief nets, showing a multilayered feedforward neural network can be well pre-trained one layer at a time, introduced the concepts of Restricted Boltzmann Machine, fine-tuned the model via supervised backpropagation [46] [49]. Since then, Deep Learning attracted attention not only in the academic field, but also many industry-leading technology companies like Google, Microsoft, and Facebook. For example, the speech recognition system based on Deep Learning techniques named Microsoft Audio Video Indexing Service (MAVIS) was released in 2012 [50]. The same year, Dahl et al. claimed that they obtained the relative improvement on the speech recognition benchmark of Bing mobile business search dataset [51]. The research groups of Hinton and Bengio focused on the classification problem for MNIST digit numbers and successfully broken the performance record maintained by traditional Machine Learning methods [46] [47]. Ciresan et al. proposed the convolutional architecture-based model with 0.27% error achieved on MNIST dataset which was the state-of-the-art method at that time [52]. The magnificent breakthrough in object recognition of natural images was achieved on the ImageNet dataset by Krizhevsky et al. in 2012 [53]. By using the CNN structure, the performance of their model has achieved unprecedented improvement over other methods. Their successful attempt impressively promoted the development of object recognition and other related tasks and brought great inspirations to the research community. Another primary field of applications of Deep Learning is NLP. Hinton et al. first introduced the idea of distributed representations for symbolic data in 1986, [54]. Later in 2003, Bengio et al. claimed a neural probabilistic language model based on the architecture of feedforward neural network [55], which laid the groundwork for the enhancement of word embeddings later on. In 2011, the neural net language model developed by Mikolov et al. [56] with adding recurrence to the hidden layers beats the state-of-the-art smoothed n-gram models in the aspect of perplexity, as well as error rate in speech recognition. Soon after, many models were proposed with the goal of solving word representation problem and some higher-level semantic understanding tasks. Models based on various types of Deep Learning techniques such as CNN, RNN, and recursive networks were broadly studied and practiced.

In recent years, many study cases shown that Deep Learning performs promisingly on various intractable tasks. Deep Learning algorithms are good at discovering the intricate structure of complex and high-dimensional data. In addition to the field of image recognition and speech recognition, Deep Learning also has a significant impact on drug discovery, genetic learning, and disease detection over many Machine Learning methods. More surprisingly, in some very complicated tasks in the domain of NLP such as topic classification, sentiment analysis, question answering and machine translation, Deep Learning-based methods are also able to produce promising results [42].

In this chapter, we will first introduce the history and current status of Deep Learning techniques comprehensively. Next, we will present different types of deep neural network structures closely related to the topic of this thesis, namely, CNN, RNN, as well as the various variations. For each model, we will analyze the overall structure, the connection, calculation between layers, the intuition, and implication behind the architecture.

3.2 Convolutional Neural Network

Convolutional architecture is one of the essential factors that make Deep Learning successfully in many areas [45]. CNN is a specialized class of deep and feedforward neural networks having a known, grid-like topology [42] [3]. Data with different forms such as one-dimension sequential data, a two-dimension grid of pixels of image data, three-dimension video, or volumetric images can be inputs to CNN. As the name implies, one of the primary features of CNN is the application of a linear mathematical transformation between convolutional layers, known as "convolution." The four core ideas behind CNN that make use of the nature of natural signals are local connections, shared weights, sub-sampling, and applying multiple layers [42] [57].

3.2.1 Architecture Overview

From a high-level perspective, a CNN consists of the input layer, the output layer, and multiple hidden layers. Concretely, the hidden layers typically contain Convolutional Layers (CONV), Pooling Layers (POOL), and Fully Connected Layers (FC). These layers are stacked in some specific manners to form a full convolutional architecture. For example, the standard CNN is merely a set of layers in sequence: [INPUT – CONV- RELU – POOL - FC], and each layer takes an input 3D volume of data and transforms it to an output 3D volume through a differentiable function [58]. In more detail, INPUT is in the form of multidimensional array of data, while units in CONV layer form feature maps, obtained by the convolution operation between the local patches (receptive field) in the feature maps of the previous layer, and a multidimensional array of parameters (kernel or filter) that are adapted by the learning algorithm. Then the results of convolution are passed through a non-linearity function such as Rectified Linear Unit (ReLU) for an elementwise activation operation. The output from ReLU is down sampled along the spatial dimensions in the POOL layer. Finally, the FC layer calculates scores corresponding to each class, and neurons in a fully connected layer are connected to all activation units in the previous layer via matrix multiplication followed by a bias.

3.2.2 Convolutional Layer

The most notable difference between CNN and the standard NN is that the layer-to-layer operation of the former (at least one of the layers) are achieved through convolution, while the latter is matrix multiplication. Convolution helps to enhance the performance of the model through three different aspects: Sparse Interactions, Parameter Sharing, and Equivariant Representations [3]. Sparse Interactions describe the characteristics of convolutional layers. When met with a very large volume of input data, the size of the filter should be much smaller than the size of the input data, which allows the model to have fewer parameters and times of operations. Sparse Interactions reduce the memory requirements and boost the computational efficiency. Parameter Sharing scheme in the convolutional network is applied by using one same filter for each receptive field in the layer. Although it has no influence on the runtime of forward propagation, it dramatically decreases the storage requirement. Therefore, combining these two factors makes convolution considerably more efficient. The property named Equivariance to Translation, is caused by parameter sharing. When shifting or changing the input by some value, the same amounts of shifts or changes will appear in the output. Equivariance is quite important when the small but crucial feature information of the input data is useful for many other input locations.

3.2.3 Pooling Layer

While the convolutional layer is to detect and extract the effective features, the purpose of the pooling layer is to yield the approximate characteristics of the output of convolution layer by transforming several statistical numbers in a specific range into one via a pooling function. There are various types of pooling function being used in different CNN models, such as the max pooling, average pooling, and neighboring pooling [42]. The effect of pooling layer is to make the representation Invariance to Translation. When shifting the input by a small amount, the most parts of the output stay unchanged. This property is advantageous when we care about the presence and functionality of some feature extractors, not its exact location. It enables the function learned by each convolutional layer to be invariant to small changes, which significantly improve statistical efficiency [3]. The other advantages of the pooling layer are reducing the input size, thus makes the model be statistical efficient and requires less storage for parameters. The pooling layer can deal with the variable-size input by varying the hyperparameters, so that the size of the output keeps consistent regardless of the input size [3].

3.2.4 Fully Connected Layer

The output of the convolutional layers represents the high-level features of the data, and it can be flattened and connected to the output layer. Adding a fully connected layer is an inexpensive way for learning nonlinear combinations of these features. Every neuron in a fully connected layer is connected to every neuron in the other layer, behaving in the same way as the tradition neural networks.

3.2.5 Discussion & Summary

We have introduced the structure of the CNN, the details of each type of layers and their connectivity. In practical applications, there are many other issues, such as how to avoid underfitting and overfitting, the choice of loss function and optimization methods. However, these issues are involved in the framework of tradition neural networks, so we will not introduce in detail in this section.

The CNN was initially designed for the application of image classification and recognition tasks. As early as the 1990s, Yann LeCun developed LeNet, which was used to identify zip codes and digits, and it became the first successful attempt on convolutional network in the field of image recognition [45]. However, due to the limited computing power of the machine at that time, the convolutional network seemed to fade out of sight. Until 2012, Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton proposed AlexNet and achieved outstanding results in the ImageNet challenge competition [59]. AlexNet was similar to LeNet regarding the overall architecture, but it was deeper (increased number of layers) and bigger (increased size of filters). From then on, the convolutional network became famous again in the field of computer vision and began to be broadly used in commercial systems. Over the next few years, various novel convolutional network-based models emerged in an endless stream with improved performance in many scenarios. While pursuing high accuracy, researchers are continuously exploring the approaches of optimizing model size and reducing computation expense. In 2014, the concept of Inception Model was introduced by GoogleNet, which significantly reduced the total number of parameters, from 60M of AlexNet to 4M [60]. In the same year, the research group of VGGNet claimed that the depth of the network is a critical factor related to quality [61]. Kaiming He et al. solves the problem, that the accuracy becomes saturated or degraded severely as the depth of the network becomes deeper, by Skip Connection of the Residual Network [62]. CNN has been proved not only suitable for processing data with a precise grid topology, such as two-dimension image topology, but also has the auspicious performance for one-dimension word sequence data, which will be introduced with examples in the next few chapters.

3.3 Sequential Model

3.3.1 Overview

RNN is a class of neural networks for recognizing patterns in sequences of data. The connections between units form a directed graph along the sequence. Just as CNN is a neural network that specializes in processing two-dimension data such as image, RNN is dedicated to processing data in the form of a one-dimension sequence such as text.

In the previous section, we mentioned the idea of Parameter Sharing, which makes the model flexible, allowing the model to be adapted to handle different forms of data (including variable-length data for text information) and to generalize across them [3]. Conversely, if units in different time steps have separate parameters, it is hard for the model to be generalized to a sequence of unseen length during the training process. The Parameter Sharing mechanism is especially important for dealing with semantic understanding issues. For example, a particular piece of information may appear in multiple locations in the long sequence data. A good model is required to discover this particular information regardless of its location. A feed-forward neural network can hardly achieve this because a fully connected neural network uses separate parameters for each input features. For a specific sentence, the model needs to learn all rules of the language according to different locations to extract corresponding features. In other words, a feed-forward network takes no care about time steps, it considers the current input training example only and forgets about the previous ones. However, RNN makes use of the idea of Parameter Sharing in time series through a very deep structure of computational graph to overcome this issue.

3.3.2 Recurrent Neural Network

One of the most significant differences between RNN and the feed-forward neural network is that the former has a "memory" which captures information about what has been calculated so far by a feedback loop structure connected to the past decisions, while the latter treats all inputs independently with each other. The term Recurrent means that an RNN applies the same rule for every unit of a sequence, and each member of the output is a function of previous members of the output. For every computation unit of an RNN, there are two sources of inputs, one is from the present input, the other is from the recent past information. Therefore, the output of every time steps is affected by the combination of these two factors.

The purpose of adding this memory function to the network is inspired by the fact that the information of the sequential data is not independent in time, but the sequence itself contains much useful information. That kind of sequential information is represented by the corresponding hidden states of RNN and is fed into the next recurrent unit, along with the new example of the next time step. As there is a saying that, "The past that does not pass away." The correlations called "Long-Term

Dependency" among these separate moments are probably the underlying factors that influence each other.

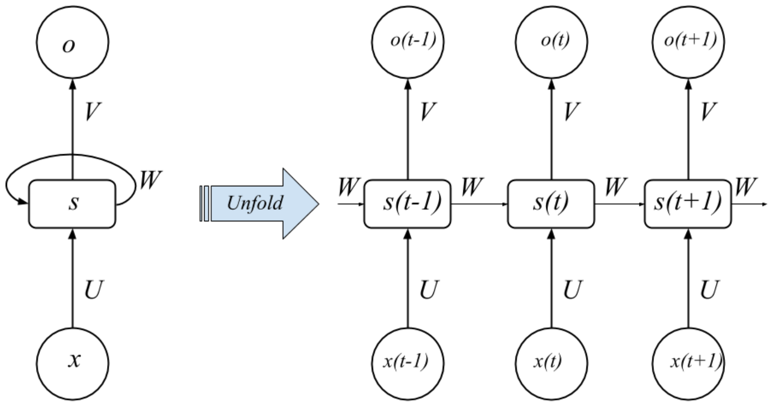


Figure 6 The basic architecture of an RNN in two forms.

Two forms of an RNN are shown in Figure 6, the left one is in the loop form, and the right one is unrolled into a full network. x_t is the input at time step t. For example, x_t could be a word vector corresponding to the t-th word in the given sentence. s_t is the hidden state at time step t, which represents the "Memory" holding by the network so far, not only contains information from the previously hidden state, but also all those that preceded s_{t-1} for as long as memory can persist. s_t is obtained from the previous hidden state s_{t-1} and the present input x_t. In the mathematical formula,

$$s_t = f(Ux_t + Ws_{t-1}) \tag{19}$$

, where U and W are the matrices of weights, which can be viewed as filters that determine how much attention is paid to the present input and the past hidden state, and they will be updated through backpropagation [24] until the error no longer increasing. The weighted summation of the input and the past information is then squashed by the function $f(\cdot)$, which can be nonlinearity functions such as $ReLU$ or $tanh$, making gradients workable for backpropagation. Usually, we initialize the hidden state $s_{t-1}$of the first unit to all zeroes. $o_t$is the output at time step t. A Softmax function is added on the top of the output to produce a probability distribution across all labels. In the formula,

$$o_t = softmax(Vs_t) \tag{20}$$

and o_t is calculated based on the memory at time step t.

Tradition neural networks uses different parameters without Parameter Sharing scheme at each layer. On the contrary, RNN shares the same parameters U, W, V across all time steps, while the only change is the input at each time step. With the idea of Parameter Sharing, the total number of parameters is significantly reduced, thus alleviating the need for memory and computational power.

RNN proves to perform well on many tasks in NLP, while various drawbacks existing. Gradient Vanishing and Exploding is among the most severe obstacles to improve the performance of RNN [63] [64]. RNN explores the connections among the present unit and the units of many time steps before, and sometimes it fails since the way of information passing through is by several multiplication with the weight matrix. When the data sequence is longer, the gradients during backpropagation are prone to be infinitely close to 0 (Gradient Vanishing) or infinitely close to 1 (Gradient Exploding). Gradient Exploding can be handled via truncation or some squashing functions on the exploded gradients [65]. However, the vanishing gradients are too small for the network to learn.

3.3.3 Long Short-Term Memory Network

The RNN-based structure called LSTM was proposed by Hochreiter and Schmidhuber in 1997 as the solution to Gradient Vanishing problem [66] and was improved in 2000 by Felix Gers et al. [67]. An RNN composed of LSTM units is called an LSTM network, which is initially designed for solving the problem of Long-Term Dependency. The overall structure of LSTM is similar to RNN, but the structure and connectivity among units is entirely different. A LSTM unit consists of three gates and one cell. The cell is similar to the hidden state of RNN, called Memory Cell, which is responsible for remembering the information in the past time steps. Three gates, namely, Input Gate, Output Gate, and Forget Gate, can be viewed as a neuron of the traditional neural network, for computing an activation of a weighted sum. In other words, these gates regulate how much information could pass through the current unit. Unambiguously, the forget gate decides what information will be forgotten from the cell state. The input gate decides which values from the previous hidden states and new input data should be updated. The "$tanh$" layer next to the input gate creates a vector of new candidate which will be added to the current state, u_t. Then the old cell state c_{t-1}can be updated into the new cell state c_t with the regulation of the input gate and the forget gate. Finally, the output is obtained by the current cell state with regulated by the output gate.

Mathematically, at time step t, we denote input gate as $\boldsymbol{i}_t$, forget gate as $\boldsymbol{f}_t$, output gate as $\boldsymbol{o}_t$, memory cell as $\boldsymbol{c}_t$, hidden state as $\boldsymbol{h}_t$, input data as $\boldsymbol{x}_t$, $\odot$ denotes element-wise multiplication. The range of values for these gate variables is limited to $[0, 1]$. Then the overall transition equations of a LSTM network are as follows:

$$\boldsymbol{i}_t = \boldsymbol{\sigma}(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{U}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)}) \tag{21}$$

$$\boldsymbol{f}_t = \boldsymbol{\sigma}(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{U}^{(f)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(f)}) \tag{22}$$

$$\boldsymbol{o}_t = \boldsymbol{\sigma}(\boldsymbol{W}^{(o)}\boldsymbol{x}_t + \boldsymbol{U}^{(\boldsymbol{o})}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(o)}) \tag{23}$$

$$\boldsymbol{u}_t = \boldsymbol{\sigma}\left(\boldsymbol{W}^{(u)}\boldsymbol{x}_t + \boldsymbol{U}^{(u)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(u)}\right) \tag{24}$$

$$\boldsymbol{c}_t = \boldsymbol{i}_t \odot \boldsymbol{u}_t + \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} \tag{25}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh{(\boldsymbol{c}_t)} \tag{26}$$

$W, U, and\ b$ are weight matrices and bias vector parameters which are learned during the training process, where the superscripts i, f, o, u refer to the corresponding types of gates such as input, forget output, and the intermediate candidate value.

3.3.4 Gated Recurrent Unit (GRU)

The architecture of LSTM network is quite complicated, and the functionality of each gate and the state of the internal components does not seem so clear. Some voices questioned there might exists better architectures [68]. Gated Recurrent Unit is one of the variants of LSTM network, proposed by Cho et al. in 2014 [69]. It makes a slight modification to the standard LSTM, combining the forget and input gates into a single update gate. Thus, the cell state and hidden state in LSTM are merged as one. Gated Recurrent Unit can be viewed as a simpler version of LSTM network, which has fewer parameters. Specifically, GRU only has two types of gates. One is the reset gate (denote as r), the other is the update gate (denote as z). The reset gate decides the way of combining the input at the current time step with the past information, while the update gate regulates how much previous memory needed to be kept. The extreme case comes when all reset gates become 1 and all update gates be all 0, which is the same as the standard RNN.

In the following equations, at time step t, we define x_t as the input, r_t as the reset gate, z_t as the update gate, h_t as the hidden state. W and b is the weight matrices and bias vector, which is quite similar to LSTM network as we discussed above.

$$\boldsymbol{r}^t = \sigma(\boldsymbol{W}_r\boldsymbol{x}^t + \boldsymbol{U}_r\boldsymbol{h}^{t-1}) \tag{27}$$

$$\boldsymbol{z}^t = \sigma(\boldsymbol{W}_z\boldsymbol{x}^t + \boldsymbol{U}_z\boldsymbol{h}^{t-1}) \tag{28}$$

$$\widetilde{\boldsymbol{h}}^t = tanh(\boldsymbol{W}\boldsymbol{x}^t + \boldsymbol{U}(\boldsymbol{r}^t \odot \boldsymbol{h}^{t-1})) \tag{29}$$

$$\boldsymbol{h}^t = (1 - \boldsymbol{z}^t) \odot \boldsymbol{h}^{t-1} + \boldsymbol{z}^t \odot \widetilde{\boldsymbol{h}}^t \tag{30}$$

According to [68], it is hard to say which one is better between GRU and LSTM. Depending on specific tasks and datasets, both architectures produce relatively good performance. Besides, different hyperparameters seem to be one of the critical factors having influences on the quality of the model. GRU is the winner regarding the number of parameters. Thus, it can be generalized easier and requires less memory storage. However, LSTM may show somewhat stronger power when the training data is sufficiently enough.

3.3.5 Bidirectional RNN

All recurrent networks we described are organized in a "casual" structure, meaning that each time step t only captures the information from the past time steps and the current new input. The restriction is that the future input information cannot be reached from the current state. In many applications of NLP, people hope that the output at every time step t is not only determined by the recent past information but also determined by the whole input sequence. For example, in the task of speech recognition, because of co-articulation, the correct understanding of the present phoneme is based on several nearby phonemes. It may even be influenced by words in the future or past since the semantic dependencies among nearby words [3].

Bidirectional RNN was introduced in 1997 by Schuster and Paliwal to increase the amount of the input information available to the network [70]. The high-level structure of bidirectional RNN consists of two standard RNNs. One of them moves forward from the beginning of the input sequence to the end (positive time direction), the other moves backward through time (negative time direction). At time step t, we denote x_t as the input, h_t with the forward arrow as the forward hidden state of the sub-RNN that moves forward through time, h_t with the backward arrow as the backward hidden state of the sub-RNN. Thus, the information from past and future can be used to compute the output y_t. $W, V, and\ U$ are weight matrices, and c is the bias. $f(\cdot)$ represents one of the nonlinear functions such as $tanh$ or $ReLU$.

Bidirectional RNN has been applied to many applications such as handwriting recognition [71], speech recognition [72], and bioinformatics [73]. It improves the performance significantly compared to the standard RNN.

3.3.6 Other variants of RNN

There are some other notable variants of recurrent network such as Depth Gated RNNs [74], Grid LSTM [75], which seems to perform promising as well. In general, the computation in most RNN-based models is made up of three parts, respectively, from the input to the hidden state, from the previously hidden state to the next hidden state, and from the hidden state to the output. Different weight matrices are associated with these transformations, and each of the computation is quite shallow thus it can be viewed as just a single layer network. However, Graves et al. claimed that introducing depth in each of these operations can benefit a lot in performance [76], which is the basic concept of Deep Recurrent Network. Specifically, a standard RNN can go more in-depth with many methods, such as using deeper computation in those three computational parts or breaking the hidden states into groups in a hierarchical way [77].

3.3.7 Recursive Neural Networks

Recursive Neural Network was proposed by Pollack et al. in 1990 [78], and it has been successfully applied in NLP by Socher et al. [79]. Recursive Neural Network can be understood as a generalization of RNN with a different structure of the computational graph, which is organized in a deep tree-like structure, rather than the chain-like structure. It is a special type of deep neural network, which predicts over variable-size input with the same set of weights applied to the input recursively. Recursive Neural Network works well on learning sequence and tree structures. Especially, it is good at learning a phrase or sentence representation based on word embedding since the representation of a phrase or sentence is determined by both the meaning of its words and the rules that combine them, and Recursive Neural Network can jointly learn compositional vector representation as well as parse trees. Specifically, the input of Recursive Neural Network is two candidate children's representations. While the output consists of two parts, one is the merged representation of two children nodes, and the other is the score indicating how likely the new node would be or whether it is a correct merging decision. Children nodes are combined into a parent node by a weight matrix which is shared across the whole network, and a non-linearity function such as $tanh$ is applied to the parent node. Mathematically,

$$p_{i,j} = \tanh\left(W\left[c_i; c_j\right] + b\right) \quad (31)$$

$$S_{i,j} = U^T p_{i,j} \quad (32)$$

, where the representation c_i and c_j of children node which has the same dimension as the parent node $p_{\mathrm{i,j}}$. Assume the dimension of the representation vectors is n, then W is the weight matrix with the dimension of $n \times 2n$ and b is the bias. The score is computed based on the parent node and the weight matrix U with dimension $1 \times n$.

The objective of the learning process is to increase the scores of good segment pairs. During each iteration, scores of all pairs of neighboring nodes (say they are all in the set C) are being computed by the same weight matrices, and then the pair with the highest score will be selected to be the new node. Then this pair is removed from set C, as well as pairs contain either one of them. After that, the corresponding row and column in the weight matrices that indicate the changes is updated. The process will not stop until only one parent node is left in the set C, which represents the overall meaning of the input sequence. Then, the tree can be recovered by unfolding every merged decision down to the original segments which are leaf nodes of the tree. Moreover, the final score for prediction is merely the summation of all sub-scores of the decisions:

$$s(RNN(\theta, x_i, y')) = \sum_{d \in N(y')} s_d \quad (33)$$

where θ is all the parameters for computing a score s, x_i is the input sequence, y' is the prediction, $N(y')$ is the set of non-terminal nodes.

Compared with the RNN, one distinct advantage of recursive networks is that for the input sequence of length l, the number of nonlinear computing operations can be drastically reduced from l to the $O(log\ l)$ due to the tree-like architecture, which relieves the problem of long-term dependency.

Chapter 4
Sentence Embedding Models

4.1 Overview to Semantic Compositionality

In Chapter 2, we discussed several word embeddings models, such as count-based co-occurrence matrix, prediction-based Word2Vec, and fastText. The feasibility and effectiveness of these methods have been confirmed on many downstream tasks. For the word embeddings problem, research focuses on the interpretability of the distributed representation of a token or a set of tokens. However, in this chapter, the concentration will be shifted to composability. In other words, when multiple individual words are grouped in a specific manner, such as words consist of a phrase, phrases consist of a sentence, sentences consist of a document, whether the distributed representation of the resulting larger unit is still interpretable or not?

What related tightly to the concept of composability is the concept of compositionality. In 1995, Partee et al. [80] claimed the concept of compositionality, the meaning of a whole is a function of the meaning of the parts. Concretely, compositionality of language is not only the summation of the meanings of constituents but also the ability to combine these constituents [81]. Fodor et al. [81] and Holyoak et al. [82] claimed that the meaning of the combination of symbols is not merely the concatenation of the meaning of these symbols, while each symbol preserves its identities. Pinker [83] described the discrete combinatorial systems and the blending systems of language and argued that the blending-based composition would make better generality than specificity. For the former, the meanings of some complex structures within the unlimited number of distinct combinations go beyond those individual elements. While the latter, the meaning of the composition stands in between of those individual elements and some properties are lost in blending.

Composability indicates the ability of compositionality, which is a crucial factor in measuring the quality of the distributed representation model. Representations for word sequences is the focus of the study for the semantic composability. Recent studies for creating representations for sentences can be divided into the following two categories [12], one is semantic aspect approach, the other is representational aspect approach. The semantic aspect approach focuses on how and why symbols are composed together and what is the resulting distributional representation for these symbols. The representational aspect is biased towards the analysis, decoding, and deconstruction of the overall structures of the composed distributional representation. Concretely, the compositional model can be achieved through simple addition operations, complex deep neural network structures, or autoencoders. A distributional representation of sentences can be obtained through all these approaches, some are in a supervised learning manner, some are in an unsupervised manner. For these methods as mentioned earlier, we will conduct detailed analysis and evaluation of some typical models of each category.

4.2 BOW-based Model

4.2.1 Paragraph Vector

There are two different types of Paragraph Vector, one is called Distributed Memory Model of Paragraph Vectors (PV-DM), the other is Distributed Bag of Words version of Paragraph Vector (PV-DBOW). It was proposed by Mikolov et al. in 2014 and inspired by the CBOW model and the Skip-gram model in Word2Vec [84]. Since these two models for word embeddings can well capture the semantic features of each word in the process of achieving the final prediction, the same idea can be applied to learning representations for paragraphs. For the PV-DM, each word is mapped to a unique word vector, and each paragraph is mapped to a unique paragraph vector. The prediction for the next word in the sentence is based on both context word vectors and the paragraph vector, in a manner of averaging or concatenating vectors. The paragraph vector plays the role of a memory of the whole context that has been considered so far. The same paragraph vector is shared by all context within the same paragraph during the learning process, while the paragraph vectors for different paragraphs are utterly distinct to each other. For each word vector, it is shared across the whole training dataset, no matter from which paragraphs it comes. The learning processing is achieved via stochastic gradient descent and backpropagation [24]. The learned representations can be directly used as the features of the corresponding paragraph for the downstream tasks. The PV-DBOW model works in a way that only slightly different from the PV-DM model, similar to the difference between the Skip-gram model and the CBOW model. Instead of relying on the word vectors of the context words, the PV-DBOW model only uses the paragraph vector as the input and forces itself to predict words randomly sampled from the paragraph, then applies as the classification over them.

Collecting and labeling data is the most time-consuming and labor-intensive work. One remarkable advantage of the Paragraph Vector is the unsupervised learning process. Thus, it works well for tasks with large unlabeled data. The Paragraph Vector starts from the BOW model but does not stop there. One of the most significant problems of BOW-based models is the insensitivity of the order, which fails to preserve so much useful information of the paragraph. The Paragraph Vector indeed considers the order of words in a given context window. Moreover, since the Paragraph Vector is in an almost same training rule as Word2Vec, it successfully maintained the persuasive ability of Word2Vec to capture semantic information. However, the shortcoming of the Paragraph Vector is at prediction time. The paragraph vectors for new paragraphs need to be computed and applied with gradient descending, which is not time-saving, although the parameters for the rest of the model is fixed.

4.2.2 FastSent

FastSent is a sentence-level log-linear bag-of-words model for semantic distributed representation, proposed by Hill et al. in 2016 [85]. It is an unsupervised method, thus only requires massive unlabeled dataset for training and predicts next sentences of the given sentence with distributed representations. Concretely, for a word w in the text, the model learns two types of embeddings, one is source vector u_w, the other is target vector v_w. The vector representation s_i for sentence S_i can be represented as the summation of all source vectors of the words consists of S_i:

$$s_i = \sum_{w \in s_i} u_w \tag{34}$$

Given a sentence tuple (S_{i-1}, S_i, S_{i+1}), the loss function of this tuple is:

$$\sum_{w \in S_{i-1} \cup S_{i+1}} softmax(s_i, v_w) \tag{35}$$

They also experimented with a variant model, which is

$$\sum_{w \in S_{i-1} \cup S_i \cup S_{i+1}} softmax(s_i, v_w) \tag{36}$$

The difference is that words in the target sentence are also considered when predicting the sentence embedding of the target sentence. At testing time, the model proceeds the encoding process and finally produces the distributional representations for the new sentence S:

$$s = \sum_{w \in S} u_w \tag{37}$$

The log-linear models generally have a reliable performance in unsupervised learning tasks. It is quite surprising that the role of words order seems unclear after the experiments. Since they found that both of the types of models, the models which do not care about the word order (like BOW model) and the models which are very sensitive to word orders (like the RNN-based model), have almost the same scores for both supervised and unsupervised evaluation tasks. The authors threw a guess that the robust model used for generating distributed representations of sentences can disambiguate most sentences in the corpus, even some order-critical sentences can be captured regarding the inner conceptual semantics. However, the reason might be the testing data currently used for evaluation does not reflect the traits of the word order. This presumable reason also reflects in some of the models, such as Siamese Continuous Bag of Words model and Smooth Inverse Frequency model.

4.2.3 Siamese CBOW

Inspired by the authors of FastSent, who claimed that the order of words have little effect on the performance of the sentence vector representation, Kenter et al. developed the Siamese CBOW model in 2016 [86]. Before their work, one of the simplest ways to generate sentence embedding is to compute the average word embeddings of all words in the sentence, and this has proven to be a quite robust baseline in multiple tasks [87] [88]. The word embeddings being used mostly are the pre-trained word embeddings from the Word2Vec or GloVe, which are not explicitly learned for representing sentences in such way. Siamese CBOW provides an approach to optimize the word embedding's objective function to obtain sentence embedding with word embeddings being averaged. It proves to be better suited for this type of tasks than Word2Vec model does.

Mathematically, define a probability $\mathrm{p}(\mathrm{s_i}, \mathrm{s_j})$ indicates how likely it is for the given pair of sentences $(\mathrm{s_i}, s_j)$ to be adjacent to each other in the training corpus. It is computed via a Softmax function, where the denominator should iterate through all existing sentences in the corpus theoretically and computing the cosine similarity between the two sentences:

$$\mathrm{p}_\theta(s_i, s_j) = \frac{\exp(\cos(s_i^\theta, s_j^\theta))}{\sum_{s_k \in S} \exp(\cos(s_i^\theta, s_k^\theta))} \tag{38}$$

, where s_x^θ denotes the embedding of sentence s_x and θ represents the parameters of the model. However, when the size of the corpus is enormous, it is intractable to compute the part of the denominator to get the probability $\mathrm{p}_\theta(s_i, s_j)$. Thus, the objective function is then modified as:

$$\mathrm{p}_\theta(s_i, s_j) = \frac{\exp(\cos(s_i^\theta, s_j^\theta))}{\sum_{s_k \in \{S^+ \cup S^-\}} \exp(\cos(s_i^\theta, s_k^\theta))} \tag{39}$$

, where S^+ represents sentences that located next to the current sentence in the corpus and S^- represents n sentences that are randomly chosen from those which are not next to the current sentence. Finally, the overall objective of the model is to minimize the negative log function based on the categorical Cross-Entropy:

$$\text{minimize L} = -\sum_{s_j \in \{S^+ \cup S^-\}} p(s_i, s_j) \times \log\left(p_\theta(s_i, s_j)\right) \tag{40}$$

, where $p(s_i, s_j)$ is the ground truth probability defined as:

$$p(s_i, s_j) = \begin{cases} \frac{1}{|S^+|}, & \text{if } \mathrm{s_j} \in S^+ \\ 0, & \text{if } \mathrm{s_j} \in S^- \end{cases} \tag{41}$$

Every word in the input sentence and the sampled sentences is mapped to a projection layer which selects the corresponding word embeddings in word embeddings matrix W. These selected word embeddings in one sentence are being averaged to obtain a sentence embedding with the same dimension as the word embeddings. The cosine layer is used to compute the cosine similarity between the input sentence and the other sentences. A Softmax function is added on the top of the cosine layer to produce the probability distribution for final prediction. The model is optimized by stochastic gradient descent and parameters are updated via backpropagation [24].

It can be readily observed that many factors might affect the performance of the model, such as the number of iterations, the number of negative examples, and the number of dimensions. As the training time increases, the performance of the model does not change obviously. Two negative examples every time seems an excellent choice for stable performance, and the 200 or 300 dimensionalities of word embeddings are preferred across tasks.

Siamese CBOW predicts a sentence given its neighboring sentences and generates sentence embeddings by averaging the embeddings of its components. However, the most significant difference between Siamese CBOW and CBOW is that the former directly compares the distance between two sentences vectors while the latter compares partial information of a sentence with a word embedding, which is also one of the differences between Siamese CBOW and FastSent [85]. Apart from that, the number of parameters of Siamese CBOW is about half of FastSent, and the modification to the original Softmax function dramatically helps the model to be much more efficient than FastSent.

4.2.4 CHARAGRAM

CHARAGRAM is an approach for learning character-based distributed representations of words or word sequences, proposed by Wieting et al. in 2016 [89]. CHARAGRAM model encodes a word or a word sequence as a vector containing counts of character n-grams. Then the vector is projected into a low-dimensional space via a single nonlinear transformation, which is the resulting representations of character n-grams. Finally, the sequence embeddings indicating both word embeddings and sentence embeddings can be obtained through a summation operation over all the character n-grams of that sequence.

Concretely, character is the smallest unit of the sequence of texts in CHARAGRAM. Given a word sequence, denote it as $x =< x_1, x_2, \dots, x_m >$, and special characters such as spaces and start-of-sequence, end-of-sequence are included. Denote the subsequence of characters from position i to position j as $x_i^j =< x_i, x_{i+1}, \dots, x_j >$, and $x_i^i = x_i$. To embed a character sequence, the model computes the summation of the vectors of all character n-grams of the sequence, then applied an elementwise nonlinearity function to the result:

$$g_{CHAR}(x) = h(\boldsymbol{b} + \sum_{i=1}^{m+1} \sum_{j=1+i-k}^{i} \mathbb{I}[x_j^i \in V] W^{x_j^i}) \qquad (42)$$

, where h is one type of nonlinear functions, $\boldsymbol{b} \in \mathbb{R}^d$ is a bias term, and k is the maximum length of any character n-gram. $\mathbb{I}[p]$ represents the indicator function, which returns 1 if p is true and returns 0 if p is false. V represents the vocabulary of character n-grams in the model and $W^{x_j^i}$ is the vector representation for character n-grams x_j^i with the same dimension as the bias term $\boldsymbol{b}$.

Before training, the vocabulary of character n-grams V needs to be constructed. The number of n-grams and the order of n-grams can affect the performance of the model. It shows that for different tasks, the influence of the number of n-grams varies, especially for semantic similarity tasks it is better to include more n-grams in the vocabulary. In the experiments of how the length of n-grams influence the performance, the authors used all character bigram, trigram, and 4-grams appearing in the training corpus at least C times, where C is a hyperparameter varying in $\{1,2\}$. The CHARAGRAM model has a significant advantage regarding the number of parameters compared with character-based CNN and character-based LSTM model with the same training examples since it has nearly ten times parameters fewer than those two models. Besides, it is confirmed again with CHARAGRAM model that the OOV problem can be effectively addressed when utilizing the character-level models. Compared with the other semantic representation model, PARAGRAM-PHRASE, which is trained with large volumes of paraphrase pairs [87], CHARAGRAM outperforms the model, especially with more unknown words. Since the character-level model can embed any character sequences, and the behavior is quite robust with different length of sentences. Another superiority of CHARAGRAM is that it considers word order while PARAGRAM-PHRASE averages merely the word embeddings in the sequence, and one interesting phenomenon is CHARAGRAM can handle the meaning of negation very well. The advantages of this feature are highlighted in finding etymological links of words.

4.2.5 Sent2Vec

Proposed by Pagliardini et al. in 2017 [90], Sent2Vec is another CBOW-based sentence embedding model with as lower model complexity as the averaging word embeddings approach [87]. Sent2Vec embeds sentences via learning word vectors, n-gram embeddings of the sentence, and the semantic composition of these components. The model can be viewed as an extension from the CBOW model to a larger sentence context with a modified unsupervised objective function, which significantly helps the model to learn efficiently on the large dataset in an unsupervised manner.

Same as the CBOW model, for each word w in the vocabulary, there is a source embedding $\boldsymbol{v}_w \in \mathbb{R}^h$ and a target embedding $\boldsymbol{u}_w \in \mathbb{R}^h$ to learn, with the particular form for the objective function:

$$\min_{\boldsymbol{U},\boldsymbol{V}} \sum_{S \in \mathcal{C}} f_S(\boldsymbol{U}\boldsymbol{V}\boldsymbol{\ell}_S) \tag{43}$$

, where $\boldsymbol{U} \in \mathbb{R}^{k\times h}, \boldsymbol{V} \in \mathbb{R}^{h\times|\mathcal{V}|}$ are the source and target embedding matrices respectively and $\mathcal{V}$ represents the vocabulary. The final learned word embeddings are stored in the columns of $\boldsymbol{V}$. $\boldsymbol{\ell}_S \in \mathbb{R}^{|\mathcal{V}|}$ is the indicator vector that encodes the words occurring in the sentence as a set of bag-of-word binary vectors. The input sentence is mapped into a numerical loss via each loss function within one window $f_S : \mathbb{R}^k \to \mathbb{R}$ and S is the context window size that iterates over the training corpus $\mathcal{C}$. Sent2Vec embeds a sentence S by computing the average source word embeddings as well as the n-grams embeddings appearing in the sentence:

$$\boldsymbol{v}_S := \frac{1}{|R(S)|} \sum_{w \in R(S)} \boldsymbol{v}_w \tag{44}$$

, where $R(S)$ represents a set of each word and all n-grams appearing in the sentence S.

Sent2Vec also uses the speeding up techniques proposed along with the Word2Vec model, negative sampling, and subsampling, to improve the training efficiency. The overall objective function of Sent2Vec model becomes as:

$$\min_{\boldsymbol{U},\boldsymbol{V}} \sum_{S\in\mathcal{C}} \sum_{w_t\in S} \left(q_p(w_t)\ell\left(\boldsymbol{u}_{w_t}^T \boldsymbol{v}_{(S\backslash\{w_t\})}\right) + |N_{w_t}| \sum_{w'\in\mathcal{V}} q_n(w')\ell\left(-\boldsymbol{u}_{w'}^T \boldsymbol{v}_{(S\backslash\{w_t\})}\right) \right) \tag{45}$$

, where $\ell(x) = \log(1 + e^{-x})$ is a binary logistic loss function. Each negative word w' are sampled from the training corpus (denote all negative samples as N_{w_t}) with the probability $q_n(w')$ consistent with the overall frequency of that word $f_{w'}$:

$$q_n(w') := \frac{\sqrt{f_{w'}}}{\sum_{w_i\in\mathcal{V}} \sqrt{f_{w_i}}} \tag{46}$$

$q_p(w_t)$ is the probability of a word w_t can be sampled through subsampling process, defined in the same way as in the Word2Vec model:

$$q_p(w_t) := \min\{1, \sqrt{\frac{t}{f_{w_t}}} + \frac{t}{f_{w_t}}\} \tag{47}$$

, where t is the subsampling hyper-parameter that can be tuned during training process. The learning process is done with backpropagation [24], and the parameters get updated via stochastic gradient descent with a linearly learning rate decay. Moreover, the authors applied dropout mechanism to the n-grams set $R(S)$ excluding all unigrams $U(S)$ of each sentence and L1 regularization to the word embeddings to prevent overfitting. Compared with the CBOW model, the vocabulary set of Sent2Vec is considerably enlarged by these n-grams of sentences, while the computational complexity keeps the same.

4.3 Deep structure-based Model

4.3.1 Skip-Thought Vectors

Proposed by Kiros et al. [91] in 2015, Skip-Thought vector is an unsupervised approach for learning a generic and distributed representation of sentences. Inspired by the idea of the Skip-gram model, this method modifies the objective function to generalize to the sentence level. Instead of using a word to predict the other words within a fixed-length context window, Skip-Thought vector predicts the two sentences surrounding the central sentence. The model is a style of sequence-to-sequence manner with a structure of encoder-decoder, which is very similar to the RNN Encoder-Decoder structure in neural machine translation task [92]. The encoder of Skip-Thought vector is an RNN network with GRU cells, while the decoder is also an RNN network with a condition on the output of the encoder. Thus, both encoder and decoder are sensitive to the order of the word in the sentences. The authors described that there is a unidirectional or bidirectional RNN for the encoder, while the decoder uses two single layers RNN to predict the previous and the next sentence in the document. The learning process are done via backpropagation [24] and Adam optimization. As a result, the model would produce the word embeddings for all words existing in the vocabulary as a side product. Thus, it is also feasible to use the pre-trained word vectors such as Word2Vec for the training. Sentence embeddings are generated at the intermediate stage between the encoder and the decoder.

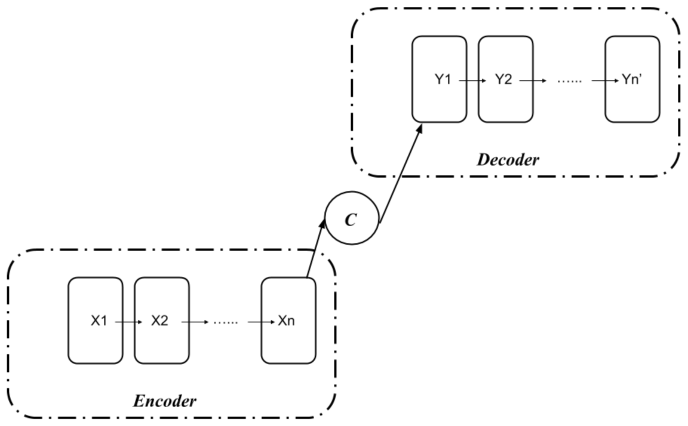


Figure 7 The RNN encoder-decoder architecture for neural machine translation **[69]**.

Concretely, for a given sentence tuple (s_{i-1}, s_i, s_{i+1}), denote the t-th word for sentence s_i as w_i^t, denote the word embeddings of this word as x_i^t. For the sentence s_i, assume the length is N, and it consists of a sequence of words

$$w_i^1, w_i^2, \dots, w_i^{N-1}, w_i^N$$

The GRU-like encoder produces a hidden state $\boldsymbol{h}_i^t$ at every time step t which represents the overall meaning of the sequence in the past, $w_i^1, w_i^2, \dots, w_i^{t-1}$. Thus $\boldsymbol{h}_i^N$ represents the information of the full sentence s_i. The encoding process follows in this way:

$$\boldsymbol{r}^t = \sigma(\boldsymbol{W}_r\boldsymbol{x}^t + \boldsymbol{U}_r\boldsymbol{h}^{t-1}) \tag{48}$$

$$\boldsymbol{z}^t = \sigma(\boldsymbol{W}_z\boldsymbol{x}^t + \boldsymbol{U}_z\boldsymbol{h}^{t-1}) \tag{49}$$

$$\widetilde{\boldsymbol{h}}^t = tanh(\boldsymbol{W}\boldsymbol{x}^t + \boldsymbol{U}(\boldsymbol{r}^t \odot \boldsymbol{h}^{t-1})) \tag{50}$$

$$\boldsymbol{h}^t = (1 - \boldsymbol{z}^t) \odot \boldsymbol{h}^{t-1} + \boldsymbol{z}^t \odot \widetilde{\boldsymbol{h}}^t \tag{51}$$

, where $\widetilde{\boldsymbol{h}}^t$ is the candidate hidden state at time t, $\boldsymbol{r}^t$ is the reset gate, and $\boldsymbol{z}^t$ is the update gate, $\odot$ denotes an element-wise product. The decoder of Skip-Thought vector works similarly except for one extra condition to the GRU, which comes from the encoder of the corresponding sentence s_i. Specifically, the decoder follows a GRU network as:

$$\boldsymbol{r}^t = \sigma\left(\boldsymbol{W}_r^d \boldsymbol{x}^{t-1} + \boldsymbol{U}_r^d \boldsymbol{h}^{t-1} + \boldsymbol{C}_r \boldsymbol{h}_i\right) \tag{52}$$

$$\boldsymbol{z}^t = \sigma\left(\boldsymbol{W}_z^d \boldsymbol{x}^{t-1} + \boldsymbol{U}_z^d \boldsymbol{h}^{t-1} + \boldsymbol{C}_z \boldsymbol{h}_i\right) \tag{53}$$

$$\widetilde{\boldsymbol{h}}^t = tanh\left(\boldsymbol{W}^d \boldsymbol{x}^{t-1} + \boldsymbol{U}^d (\boldsymbol{r}^t \odot \boldsymbol{h}^{t-1}) + \boldsymbol{C} \boldsymbol{h}_i\right) \tag{54}$$

$$\boldsymbol{h}_{i+1}^t = (1 - \boldsymbol{z}^t) \odot \boldsymbol{h}^{t-1} + \boldsymbol{z}^t \odot \widetilde{\boldsymbol{h}}^t \tag{55}$$

, where $\boldsymbol{C}_r$, $\boldsymbol{C}_z$, and $\boldsymbol{C}$ are the bias matrices introduced to the reset gate, update gate and the hidden state, respectively. For two decoders, the bias matrices are not shared between them. As we can see from the equations above, at time step t, we can calculate the hidden state of the sentence s_{i+1} given the hidden state of the sentence s_i. We can also obtain the hidden state of the sentence s_{i-1} in the same way. The objective function of the given tuple (s_{i-1}, s_i, s_{i+1}) is the sum of the log-probabilities for the next sentence and the previous sentence given the encoder of the current sentence and the total objective function over the whole dataset is the summation of all these tuples:

$$\sum_t logP\left(w_{i+1}^t \middle| w_{i+1}^{<t}, \boldsymbol{h}_i\right) + \sum_t logP\left(w_{i-1}^t \middle| w_{i-1}^{<t}, \boldsymbol{h}_i\right) \tag{56}$$

, where the probability of a word w_{i+1}^t given the previous $t - 1$ words and the encoding result $\boldsymbol{h}_i$ is proportional to the cosine similarity of the word vector of w_{i+1}^t and the hidden state representation of the sentence s_{i-1}:

$$P\left(w_{i+1}^t \middle| w_{i+1}^{<t}, \boldsymbol{h}_i\right) \propto \exp\left(\boldsymbol{v}_{w_{i+1}^t} \boldsymbol{h}_{i+1}^t\right) \tag{57}$$

4.3.2 Tree-LSTM

LSTM network [66] [67] has been proven to be useful to keep the valuable information of the long sequential context, and many novel approaches inspired by LSTM network have been proposed in the past few years for language modeling and semantic composition [93] [94]. Although the standard LSTM network can capture the Long-Term Dependencies and representational power, it still neglects some crucial features and information carried by the language itself, such as the syntactic interpretations of sentence structure. Tree-LSTM structure is one of the various types of LSTM-based sequential models, which generalizes from the standard LSTM architecture and introduces tree-structured network topologies, proposed by Tai et al. in 2015 [95]. Recall that in the standard LSTM, the hidden state of the current time step is determined by the hidden state of the previous time step and the current input. Tree-LSTM treats the gating vectors and the memory cell of one-time step dependently with the current input word embeddings and the states of arbitrarily many child units based on the s of dependency parsing. This main discrepancy between the standard LSTM and Tree-LSTM makes it possible for Tree-LSTM to capture relatively important information regarding the syntactic structure of sentences.

Concretely, given a dependency parsing tree, let $C(j)$ denotes the set of children of node j. For a word w_j in a sentence, denote the word embeddings with dimension h of that word as $x_j \in \mathbb{R}^h$ and denote the input gate, the output gate, the memory cell, the hidden state corresponding to the word w_j as i_j, o_j, c_j and h_j, respectively. As for the forget gate, every unit j in Tree-LSTM architecture can have k forget gates as many as the number of children belongs to the unit j, each forgets gate can be represented as f_{jk}. The equations illustrating Tree-LSTM network are the following:

$$\widetilde{h_j} = \sum_{k \in C(j)} h_k \tag{58}$$

$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \widetilde{h_j} + b^{(i)}\right) \tag{59}$$

$$f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right), k \in C(j) \tag{60}$$

$$o_j = \sigma(W^{(o)} x_j + U^{(o)} \widetilde{h_j} + b^{(o)}) \tag{61}$$

$$u_j = \tanh\,(W^{(u)} x_j + U^{(u)} \widetilde{h_j} + b^{(u)}) \tag{62}$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k \tag{63}$$

$$h_j = o_j \odot \tanh\,(c_j) \tag{64}$$

, where W, U, b are the weight matrices and bias vectors of each type of gates that can be learned through the learning process. As we can observe from these equations, each child of unit j has its own forget gate, and the memory cell c_j is related to every child unit, which forms the tree-structured topology. In their experiments, the model uses pre-trained word embeddings from GloVe vectors [96], uses AdaGrad [97] for the learning process, and applies L2 regularization to prevent overfitting. In the tasks of semantic relatedness analysis and sentiment classification, Tree-LSTM architecture shows its effectiveness when the length of sentences varies a lot compared with the standard LSTM and Bidirectional LSTM. It is probably because of the impact of the syntactic information. However, compared with those unsupervised models, the much more promising results of Tree-LSTM model are based on the high-quality labeled dataset, which can be obtained costly and cumbersomely. Besides, the other limitation of Tree-LSTM is that the tree-structured topology is derived from dependency parsing of sentences. Therefore, this model is only applicable to languages that can perform dependency parsing.

4.3.3 DCNN

Dynamic CNN (DCNN) is a CNN-based sentence modeling approach developed by Kalchbrenner et al. in 2014 [97]. From a high-level view, DCNN consists of several consecutive layers of a convolutional layer followed by a max pooling layer, which is quite similar to the classical Time-Delay Neural Network (TDNN) [98] [99] and the sentence model proposed by Collobert et al. [100]. The convolutional layer applies the operation named One-Dimensional Convolution to the input sentences $\boldsymbol{s} \in \mathbb{R}^{s}$ with a vector (filter of the convolution) $\boldsymbol{m} \in \mathbb{R}^{m}$. The idea is mainly operating the dot product of the vector $\boldsymbol{m}$ with each m-gram of the sentence $\boldsymbol{s}$ to yield a vector $\boldsymbol{c}$, where

$$\boldsymbol{c}_j = \boldsymbol{m}^T \boldsymbol{s}_{j-m+1:j} \tag{65}$$

There are usually two types of convolution operation, Narrow Convolution and Wide Convolution, which determines different ranges of the index j. As the name shows, Wide Convolution takes the words at the margins of the sentence into account when operating the convolution. In the other word, Wide Convolution uses all weights in the filter vector to convolve with every part of the sentence, thus resulting in the broader dimension of $\boldsymbol{c}$, while Narrow Convolution produces a resulting sequence $\boldsymbol{c}$ with narrow dimension.

Before diving into DCNN, we firstly discuss the other model called Max-TDNN proposed by Collobert et al. [100] briefly. The Max-TDNN is developed upon the classical TDNN architecture, where a sequence of inputs $\boldsymbol{s} \in \mathbb{R}^{d \times s}$ and a weight matrix $\boldsymbol{m} \in \mathbb{R}^{d \times m}$ were convolved in the convolutional layer, and multiple such convolutional layers were stacked by taking the resulting matrix $\boldsymbol{c}$ as the input to the next layer. Each column (denoted as $\boldsymbol{w}_i \in \mathbb{R}^{d}$) in $\boldsymbol{s}$ represents the d-dimensional word vector of the word at the position i of the sentence. To make the model adapted to variable-length input sentences, a Max Pooling operation was introduced after the convolutional layer, where only the maximum of each row in matrix $\boldsymbol{c}$ is preserved and thus yielding a fixed-size vector $\boldsymbol{c}_{max} \in \mathbb{R}^d$:

$$\boldsymbol{c}_{max} = \begin{bmatrix} \max(\boldsymbol{c}_1, :) \\ \vdots \\ \max(\boldsymbol{c}_d, :) \end{bmatrix} \tag{66}$$

Then the vector $\boldsymbol{c}_{max}$ is used as the input to the fully connected layer for classification.

DCNN improves the Max Pooling layer in Max-TDNN model to the so-called Dynamic k-Max Pooling operation, where k is a function of the length of the sentence and the depth of the whole network:

$$k_l = \max\left(k_{top}, \left\lceil \frac{L-l}{L} s \right\rceil\right) \tag{67}$$

, where l is the number of the current convolutional layer to which the pooling is applied, and L is the total number of convolutional layers in the network; k_{top} is the fixed pooling parameter for the topmost convolutional layer. The intuition behind this modification is that adjusting the number of features that being extracted by each layer according to the overall process over the input sentence. The overall architecture of DCNN is alternating the wide convolutional layer with the Dynamic k-Max Pooling Layer, except that after the topmost convolutional layer applied with the original max pooling layer to ensure the input to the fully connected layer become independent of the length of the input sentence.

The intuition behind the CNN-based sentence modeling approach is quite straightforward. The convolutional layers apply one-dimensional filters (with size m) across each sentence in the input sentences matrix, and the features of every m-gram of the sentence can be extracted independently of their position. The max pooling layer in the traditional CNN for object recognition [45] is a type of non-linear subsampling function returning the maximum of the set of given values, which makes the output to be invariant to some little changes of input. Dynamic k-Max Pooling not only extracts k most active features of the input sequence but also preserves the order and relative positions of the features that have been occurred in the layer before.

Compared with some other structures such as RNN-based or BOW-based models, CNN-based models have some promising advantages over them. Concretely, since CNN is a position-sensitive architecture, DCNN is good at capturing the order of the words and the relative position of the most relevant n-grams in the input sentence, at the same time, preserving the invariance to the absolute positions due to the Dynamic k-Max Pooling scheme. However, the most obvious shortcoming of the BOW-based models is order-insensitivity. Although RNN can remember the order of the word sequence, it was found that a strong bias can be introduced by the latest words in the input [101] and the same issue appears in the Recursive Neural Network [102].

On the other side, the particular property of CNN helps the model build up an internal feature graph over the input. In DCNN, nodes in the lower layer are connected with nodes in the higher layer via a convolution operation, and nodes without being selected by the pooling layer are thrown out of the graph, which results in a tree-structured topology after the final pooling layer. This characteristic is coincidentally the same as CNN for object recognition, where a weighted, connected and directed acyclic graph is also induced over the input image. BOW-based models are too shallow to form such kind of graph. Traditional RNN-based models result in linear-chain architecture. Recursive Neural Network-based models have to rely on an external parsing tree. DCNN forms a hierarchical architecture making features ordered. This resulting tree is similar to a parser tree derived from the dependency parser but is more than that, the resulting graph also includes information, indicating the short or long-range semantic relations between words which might or might not have relations according to syntax, captured by DCNN. Thus, DCNN can be applied with context or some languages which are not able to be parsed.

4.3.4 Character-Aware

Character-Aware is a sentence modeling approach based on a mixed structure of a Character-level CNN (CharCNN), a Highway Network [103] and a Multi-layer LSTM network. At time step t, the input of the model is a word represented in character-level embeddings instead of word embeddings as in many other language models, while the output of the model is still at word-level. The character embeddings of the word are concatenated as a matrix then convolved with multiple filters of different widths. The resulting matrix produced by the convolutional layer is then applied to a Max-Over-Time Pooling operation, which can make the dimension of the embedding of that word become fixed. After that, this fixed-size representation vector becomes the input of the Highway Network. The next component of the model is the Multi-layer LSTM network, which takes the output of the Highway Network as its input, as well as the hidden state from the previous time step. Finally, the Softmax function is added on the top of each hidden state of the LSTM network to obtain the probability distribution to predict the next word.

Concretely, in the part of CharCNN, given a word k consists of a sequence of characters $[c_1, c_2, \ldots, c_l]$, the character-level representational matrix of the word k is denoted as $\boldsymbol{C}^k \in \mathbb{R}^{d \times l}$, where d is the dimensionality of the character embeddings and l is the length of the word k. There is a narrow convolution between $\boldsymbol{C}^k$ and the filter $\boldsymbol{H} \in \mathbb{R}^{d \times w}$ of width w followed by a nonlinearity function yielding a feature map $\boldsymbol{f}^k \in \mathbb{R}^{l-w+1}$. Different width of the filters corresponds to the features to be extracted from the same size of character n-grams. The i-th element of $\boldsymbol{f}^k$ computed as follows:

$$\boldsymbol{f}^k[i] = \tanh\left(\langle \boldsymbol{C}^k[*, i: i+w-1], \boldsymbol{H} \rangle + b\right) \tag{68}$$

, where $\langle \boldsymbol{M}, \boldsymbol{N} \rangle = Tr(\boldsymbol{M}\boldsymbol{N}^T)$ is the component-wise inner product of two matrices and b is a bias. The Max-Over-Time operation produces the feature y^k of word k representing the most relevant feature for the given filter, and the dimension of the feature vector $\boldsymbol{y}^k$ is consistent with the number of filters with different width:

$$y^k = \max_i \boldsymbol{f}^k[i] \tag{69}$$

After CharCNN, the Highway Network takes the feature vector $\boldsymbol{y}^k$ as the input and outputs the vector $\boldsymbol{z}$ with the same dimension as the $\boldsymbol{y}^k$:

$$\boldsymbol{z} = \boldsymbol{t} \odot g(\boldsymbol{W}_H \boldsymbol{y} + \boldsymbol{b}_H) + (1 - \boldsymbol{t}) \odot \boldsymbol{y} \tag{70}$$

$$\boldsymbol{t} = \sigma(\boldsymbol{W}_T \boldsymbol{y} + \boldsymbol{b}_T) \tag{71}$$

, where g is a nonlinear transformation, $\boldsymbol{W}_H, \boldsymbol{W}_T$ are the square weight matrices, $\boldsymbol{b}_H, \boldsymbol{b}_T$ are the bias vectors, and $\boldsymbol{t}$ is the transform gate while $(1 - \boldsymbol{t})$ is the Carry Gate [103].

The last component of the model is the Multi-layer LSTM, where it takes the output of the Highway Network as its input, performs several steps which are the same as the standard LSTM as we have introduced before and outputs the hidden states to a Softmax function, which generates:

$$\Pr(w_{t+1} = j | w_{1:t}) = \frac{\exp\left(\boldsymbol{h}_t \cdot \boldsymbol{p}^j + q^j\right)}{\sum_{j' \in \mathcal{V}} \exp\left(\boldsymbol{h}_t \cdot \boldsymbol{p}^{j'} + q^{j'}\right)} \tag{72}$$

, where $\boldsymbol{p}^j$ is the output embedding and q^j is a bias. The objective of the model is to minimize a negative log-likelihood of the training sequence, and the model is trained by truncated backpropagation through time [104] [105]:

$$J = -\sum_{t=1}^{T} \log \Pr\left(w_t | w_{1:t-1}\right) \tag{73}$$

, where T is the length of word sequence in the training corpus.

Unlike the models which are merely built on a single type of deep neural networks, the Character-Aware model is built on the combination of three types of neural work. CharCNN produces the encodings of words in a simple level, focusing on the morphological similarities. The Highway Network further improves the encoding in semantic level. CharCNN seems to be well-suited to work with the Highway Network since the most active features extracted by each filter can be better combined with the Highway Network. The Character-Aware maintains the most distinct advantages of the character-level representation models, capturing the morphological information of words and dealing with the OOV issue. Moreover, Character-Aware effectively controls the number of parameters in the model since it does not require the word embeddings as inputs.

4.4 Auto-Encoder

4.4.1 Recursive Autoencoder

As we have already introduced in Chapter 3, the tradition Semi-Supervised, Recursive Autoencoder (RAE) learns vector representations for the input phrases or sentences given the prior knowledge of sentences such as a parse tree. In the paper published by Socher et al. in 2011, they described Greedy Unsupervised Recursive Autoencoder with no requirement on the prior tree of the sentence, but instead, building a tree by a greedy approximation [106]. The idea is to calculate the reconstruction error of all possible trees of the given sentence and choose the one with the minimal error as the desired tree structure:

$$RAE_\theta(x) = \underset{y \in A(x)}{\arg\min} \sum_{s \in T(y)} E_{rec}([c_1; c_2]_s) \tag{74}$$

, where $A(x)$ is the set of all possible trees and $T(y)$ represents all triplets for one tree. Given an input sentence consists of m words, the autoencoder firstly takes the first pair of adjacent nodes $(x_1; x_2)$ as the candidate child nodes, thus $(c_1; c_2) = (x_1; x_2)$ and produces the embedding of the parent node $p_{(1,2)}$. Assume that the model starts from the very left side of the sentence, then the model moves one position to the right, takes $(c_1; c_2) = (x_2; x_3)$ as the current input and again produce parent embedding $p_{(2,3)}$. This process will be repeated until the full sentence is covered. Then the pair with the minimal reconstruction error will be selected, and the corresponding parent node will replace its children in the further process. There is a difference when computing the reconstruction error from the traditional Recursive Autoencoder, where they take a weighted summation scheme due to different length the of the child nodes may have different degrees of influence to the parent node representation:

$$E_{rec}([c_1; c_2]; \theta) = \frac{n_1}{n_1 + n_2} \left|\left|c_1 - c_1'\right|\right|^2 + \frac{n_2}{n_1 + n_2} \left|\left|c_2 - c_2'\right|\right|^2 \tag{75}$$

, where n_1, n_2 are the number of words of the current candidate child nodes. To avoid the situation that the magnitude of the hidden layer becomes very small (since the model always attempts to make the reconstruction error lower), the parent node is obtained after normalization:

$$p = \frac{f\left(W^{(1)}[c_1; c_2] + b^{(1)}\right)}{\left|\left|f(W^{(1)}[c_1; c_2] + b^{(1)})\right|\right|} \tag{76}$$

, where $W^{(1)} \in \mathbb{R}^{n \times 2n}$ is the weight matrix, $b^{(1)}$ is a bias vector, $[c_1; c_2]$ is the concatenation of the embeddings of two children, and $f(\cdot)$ is the nonlinearity function such as $tanh$. The reconstructed representations of the original inputs are the same as traditional Recursive Autoencoder:

$$[c_1'; c_2'] = W^{(2)}p + b^{(2)} \tag{77}$$

, where $W^{(2)}$ is the weight matrix and $b^{(2)}$ is a bias vector. Thus, the proposed Recursive Autoencoder can build a tree of the given sentence, and each node of the tree has a vector, as a vector, which can be viewed as the distributed representation of the corresponding phrase.

Furthermore, a Semi-Supervised Recursive Autoencoder is developed on this completely unsupervised model, with a simple Softmax function added on the top of each parent node to do a classification task:

$$d(p; \theta) = softmax\,(W^{label}p) \tag{78}$$

So, the loss function is based on Cross-Entropy error:

$$E_{cE}(p, t; \theta) = -\sum_{k=1}^{K} t_k \log d_k(p; \theta) \tag{79}$$

, where t_k is the k-th element of the multinomial target label distribution t and d_k is the outputs of the Softmax function. The error of each node can be represented as the weighted sum of reconstruction error and Cross-Entropy error:

$$E([c_1; c_2]_s, p_s, t, \theta) = \alpha E_{rec}([c_1; c_2]_s; \theta) + (1-\alpha) E_{cE}(p_S, t; \theta) \tag{80}$$

, where α is a hyperparameter controls the weight applied to these two parts. Moreover, for each possible tree, the error is to sum over the error of every node of the tree constructed by the greedy scheme:

$$E(x, t; \theta) = \sum_{s \in T(RAE_\theta(x))} E([c_1; c_2]_s, p_s, t, \theta) \tag{81}$$

The overall loss function of this semi-supervised model is:

$$J = \frac{1}{N} \sum_{(x,t)} E(x, t; \theta) + \frac{\lambda}{2} ||\theta||^2 \tag{82}$$

, where (x, t) is the pair of sentence and label in the given corpus. Finally, the Recursive Autoencoder ends up producing a hierarchical structure of the input sentence, and learning distributed representations of words, phrases and the full sentence from the tree. Even though the resulting structures are not very interpretable according to syntax, the model manages to capture as many useful features from semantic aspect as possible.

4.4.2 SDAEs

Sequential Denoising Autoencoders (SDAEs) is proposed by Hill et al. in 2016 [85]. SDAEs was developed with the goal of making use of plenty of data that are not in the strict order or artificial language generated from symbolic knowledge. Denoising Autoencoders (DAEs) take high-dimensional data as inputs, adding some noises to the input data and learning parameters of encoder and decoder via the objective of recovering the original data from the corrupted data. Finally, the encoder of DAEs captures the useful features describing the input data and represents the given data as a fixed-size vector.

Concretely, to corrupt the input data, the authors applied a well-designed noise function to the sentence S, which can be seen as $N(S|p_o, p_x)$, p_o is the probability of each word w in sentence S to be dropped off, and p_x represents the probability of each non-overlapping bi-gram $w_i w_{i+1}$ in sentence S to be swapped. Then the corrupted input data is fed into the model, which is an LSTM-based encoder-decoder architecture. The objective of the model is to reconstruct the original input S from the corrupted input $N(S|p_o, p_x)$ and minimize the reconstruction error. Finally, word sequences with an arbitrary order in the training corpus can be encoded in the style of distributed representations.

4.5 Weighted Averaging Model

4.5.1 SIF-Model

As we have observed so far, the distributed representation of sentences can be derived from different types of architectures of the models. Some approaches are based on high-capacity deep neural networks, such as RNN and CNN [91] [94] [95] [97]. Some rely on shallow neural network and the idea of bag-of-words [84] [85] [86] [89] [90], and others take advantage of autoencoder to achieve an unsupervised Representation Learning [85] [106]. Depending on specific tasks and domains, good performance can be achieved through these models with some pre-processing, post-processing techniques, and fine-tuning of the hyperparameters. In 2016, Wieting et al. proposed a sentence embedding model trained on paraphrastic pairs and showed that the best sentence embeddings is obtained by merely averaging word embeddings of each word in the sentence [87].

Inspired by their results, in 2017, Arora et al. [2] claimed an improved sentence embedding method based on the weighted averaging summation of word embeddings in the given sentence, followed by a post-processing technique of subtracting the first principal component of the given set of sentences. The proposed weighting scheme is called Smooth Inverse Frequency (SIF), saying that the weight of each word is associated with the frequency of that word in the entire corpus. The theoretical justification of this reweighting scheme is based on the generative model for sentences proposed by Arora et al. [107] in 2016. The generative model is used to generate the next word in a sentence with the restrictions bounded by the discourse vector $\boldsymbol{c}_s$ of the current sentence, where the discourse vector represents things are being talked about:

$$\Pr\{w \text{ emitted in sentence } s|\boldsymbol{c}_s\} = \alpha p(w) + (1-\alpha)\frac{\exp\left(\widetilde{\boldsymbol{c}_s}^T \boldsymbol{v}_w\right)}{Z_{\widetilde{\boldsymbol{c}_s}}}, \tag{83}$$

where $Z_{\widetilde{\boldsymbol{c}_s}} \coloneqq \sum_{w\in\mathcal{V}} \exp(\widetilde{\boldsymbol{c}_s}^T \boldsymbol{v}_w)$ is the normalizing constant, and $\widetilde{\boldsymbol{c}_s} \coloneqq \beta\boldsymbol{c}_0 + (1-\beta)\boldsymbol{c}_s$, α, β are scalars hyperparameters and $\boldsymbol{c}_0 \perp \boldsymbol{c}_s$. $\boldsymbol{c}_0$ is the common discourse vector, which represents as a correction term for the most frequent discourse often related to syntax. From the above equation, we observe that the probability of one word is generated in the sentence can be boosted as long as one of the following factors is satisfied: when the word has a high frequency, which is associated with $p(w)$; when the word is correlated with the context that the sentence is talking about, represented by $\boldsymbol{c}_s$; when the word matches the common discourse vector $\boldsymbol{c}_0$ according to syntax.

The probability of the sentence is generated under the discourse vector can be computed as the product of the probability of each word that generated by the discourse vector $\boldsymbol{c}_s$. With the assumption that all words are roughly uniformly distributed, they treat $Z_{\widetilde{\boldsymbol{c}_s}}$ for different $\widetilde{\boldsymbol{c}_s}$ as the same one as Z. Thus, the probability of generating the sentence s for the given discourse vector $\boldsymbol{c}_s$ is:

$$Pr[s|\boldsymbol{c}_s] = \prod_{w\in s} Pr(w|\boldsymbol{c}_s) = \prod_{w\in s} \left[\alpha p(w) + (1-\alpha)\frac{\exp\left(\widetilde{\boldsymbol{c}_s}^T \boldsymbol{v}_w\right)}{Z}\right]. \tag{84}$$

So, they further applied a log likelihood to each term as:

$$f_w(\widetilde{\boldsymbol{c}_s}) = \log\,[\alpha p(w) + (1-\alpha)\frac{\exp\left(\widetilde{\boldsymbol{c}_s}^T \boldsymbol{v}_w\right)}{Z}] \tag{85}$$

After taking Taylor expansion, which approximates the term as:

$$f_w(\widetilde{\boldsymbol{c}_s}) \approx f_w(0) + \nabla f_w(0)^T \widetilde{\boldsymbol{c}_s} = \text{constant} + \frac{\frac{1-\alpha}{\alpha Z}}{p(w) + \frac{1-\alpha}{\alpha Z}} \left(\widetilde{\boldsymbol{c}_s}^T \boldsymbol{v}_w\right) \tag{86}$$

Thus, the Maximum Likelihood Estimator of $\widetilde{\boldsymbol{c}_s}$ can be approximated as:

$$\arg\max \sum_{w\in s} f_w(\widetilde{\boldsymbol{c}_s}) \propto \sum_{w\in s} \frac{a}{p(w)+a} \boldsymbol{v}_w \tag{87}$$

, where $a = \frac{1-\alpha}{\alpha Z}$. Thus, the Maximum Likelihood Estimation is approximately the weighted summation of the embeddings of all words in the sentence, where the weight of each word is represented as $\frac{a}{p(w)+a}$. The weight of the word would be smaller if the frequency of that word is higher, and vice versa, where the influences of some frequent words are weighted down. The final sentence embeddings can be produced via subtracting $\boldsymbol{c}_0$ which is estimated by the first principal component of $\widetilde{\boldsymbol{c}_s}$ given a set of sentences, where observed that $\boldsymbol{c}_0$ is roughly corresponding to the syntactic information or the common words, such as "just", "but", "there", so on and so forth.

With this reweighting scheme and the post-processing step of subtracting the first principal component of the given sentences, each sentence can be represented as a fixed-size vector with the same dimension as the word embeddings being used. The word embeddings can be derived from the pre-trained word embeddings such as GloVe [96], Word2Vec [22] [23].

Generally, the SIF-method does not rely on deep neural structures or some other kinds of the neural network, while only requires the pre-trained word embeddings. It is considerably straightforward to apply to various domains with the corresponding word embeddings library. Hence, the speed of this model is also fast. Same as most of the BOW-based models, the most noticeable drawback of this method is the ignorance of the words order. Thus, for some sentences whose semantic meaning is heavily dependent on the words order, this model fails to work properly.

Chapter 5
Main Works

5.1 Sentence Similarity Measurement: Word2Sent

In this chapter, we described our proposed model, Word2Sent-V1. The model is designed to measure the degree of relatedness of two sentences, which takes two sentences with arbitrary length as input and outputs a sentence similarity score. The sentence similarity score is a numerical value ranging from -1 to 1 indicating the degree of similarity between the sentences. Specifically, if the two sentences have a relatively high degree of semantic similarity, then the sentence similarity score would be closer to 1; otherwise, the score would be closer to -1. To represent each word of the input sentences, the model uses pre-trained word embeddings generated by different approaches (i.e., Word2Vec, GloVe, or fastText) trained on distinct unlabeled dataset. Particularly, all words are in the form of 300-dimension word vectors from one specific word embeddings library.

Given two sentences with the length of m and n, respectively. After mapping words into the corresponding word vectors, the input becomes two sequences of word vectors with the length of m and n. We then bundle every word of the first sentence with every word of the second sentence as a pair, so, there are $(m \times n)$ pairs in total. The word similarity score for the two words in every pair can be computed by the Cosine Distance between two corresponding word vectors, which ranges from -1 to 1. A higher word similarity score indicates the degree of similarity between two words is also higher, and vice versa. After computing the word similarity scores for all pairs, we sort all pairs according to their word similarity scores in descending order. For example, if a pair contains two same words, meaning the word similarity score of this pair would be 1, then this pair would be the first among all other pairs after sorting. Now the pair with the highest word similarity score is selected as the first candidate pair, and other pairs which contains either one of the words in this selected pair would be removed from the rest of the sorted list. This step finds merely the most similar words regarding semantics in two sentences. The selected word pair would be kept as one of the candidate pairs for computing the sentences similarity score and would not be considered in the further selection. Next, we repeat the previous steps: Select the pair with the highest word similarity score from all remaining pairs and remove it, then remove all other pairs containing either one of the words of the selected pair. Until there is no left pair in the original sorted pairs list, which indicates that we have selected min (m, n) pairs as the candidate pairs for computing the final sentence similarity score. Finally, we directly compute the average of the word similarity scores of those candidate pairs as the sentence similarity score.

Algorithm 1: Word2Sent-V1, sentence similarity measurement without weighting scheme

```
INPUT: Word embeddings $\{v_w: w \in \mathcal{V}\}$, two sentences $s_1$(with length $m$) and $s_2$ (with length $n$), an empty set $\mathcal{P}$, and an empty set $\mathcal{C}$.
OUTPUT: A sentence similarity score $sim\_unwt$ for the input two sentences.
1.  Bundle every word of $s_1$ with every word of $s_2$ as a pair and put all pairs $\{(w_{1i}, w_{2j}): i = 0,1,\ldots,m-1, j = 0,1,\ldots,n-1\}$ into $\mathcal{P}$.
2.  for each pair $p_x$ in $\mathcal{P}$ do:
        Unweighted word similarity score: $W_{un\_p_x} \leftarrow cosine\_distance(v_{w_{1i}}, v_{w_{2j}})$
    end for
3.  while $\mathcal{P}$ is not empty do:
        Move the pair $p_h: (w_{1m}, w_{2n})$ into set $\mathcal{C}$, where $W_{un\_p_h}$ is the maximal among all other pairs in $\mathcal{P}$; Delete pairs containing either $w_{1m}$ or $w_{2n}$ from $\mathcal{P}$.
    end while
4.  Compute the average $W_{un\_avg}$ of all pairs in set $\mathcal{C}$, and output $sim\_unwt \leftarrow W_{unwt\_avg}$.
```

Example 1:

Sentence 1: "Children in red shirts are playing with leaves."

Sentence 2: "Three kids are sitting in the leaves."

The length of the first sentence is 8, while the second is 7. Thus, the number of pairs of the whole set is $8 \times 7 = 56$. After sorting these 56 pairs by their word similarity scores, 3 pairs, *(leaves, leaves), (are, are), (in, in),* would have the word similarity score as 1 since the word vectors within those pairs are exactly the same.

After picking the first most similar word pair *(leaves, leaves)* as the candidate pair, all other pairs containing the word *"leaves"* would be removed from the sorted list. The same rules would be applied to the next several steps. Finally, we would obtain the following candidate pairs for computing the final sentence similarity score between the input sentences:

(leaves, leaves), (are, are), (in, in), (children, kids), (with, three), (playing, sitting), (red, the).

The word similarity scores for these pairs are:

[1.0, 1.0, 1.0, 0.814095, 0.545506, 0.463181, 0.352678]

The sentence similarity score for the two input sentences merely takes the average of those word pair similarity scores, which is 0.739352.

Example 2:

Sentence 1: "A boy is standing outside the water."

Sentence 2: "The boy is wading through the blue ocean."

After sorting and selecting, the resulting candidate pairs are:

(the, the), (is, is), (boy, boy), (water, ocean), (outside, through), (a, blue), (standing, wading)

The word similarity scores are:

[1.0, 1.0, 1.0, 0.600632, 0.519733, 0.321847, 0.300111]

Thus, the sentence similarity score for the input sentences is 0.677475.

The time complexity of this algorithm is $O(m \times n)$.

Weighting Scheme Inspired by the experiments and results done by Arora et al. [2], we introduced the same weighting scheme as the SIF-model in the second version of our proposed model, Word2Sent-V2. The idea is to assign a weight to each word, and the weight of each word is associated with the frequency of that word in the whole corpus. The weight of the word is computed by $\frac{a}{f(w)+a}$, where $f(w)$ represents the (estimated) frequency of the word w and a is a parameter. Arora et al. [2] demonstrated that the weighting parameter $a = 10^{-3}$ is the optimal setting in most cases. Thus, we applied the same configuration of the weighting parameter a in all our experiments.

For a specific word, if the frequency is higher, then its weight would be lower, and vice versa. This weighting scheme weakens the influences of some frequent words, such as "and", "a", and "the". The SIF-model obtained sentence embeddings by computing the weighted average of word embeddings of words from the sentence. The overall performance for the SIF-model is boosted significantly by introducing this weighing scheme on various semantic similarity tasks [2]. In the domain of legal language, the length of sentences is prone to be longer and the meaning of a sentence is carried by key words which are relatively not so frequent in the entire corpus. Instinctively, this experimental conclusion could be adapted to many scenarios in human languages. Thus, it is reasonable and necessary to pay less attention to very frequent words such as stop words, because those words usually contribute less meanings to build the overall meanings of the sentence.

We applied such a weighting scheme to Word2Sent-V2. Specifically, after computing the Cosine Distance between the two word embeddings of all pairs, each word similarity score is multiplied by the average weight of the weights of two words to obtain the weighted word similarity score. Then all pairs are sorted by the weighted word similarity scores in descending order, and it comes with the selection of the highest weighted word similarity score. However, as the selection is based on the weighted word similarity scores, the model would also record the unweighted word similarity score of each candidate pair. The rest steps of Word2Sent-V2 is the same as the original version. Finally, the sentence similarity score is computed by taking the average of the unweighted word similarity scores of all candidate pairs.

To illustrate the difference between Word2Sent-V1 and Word2Sent-V2, we use the same input sentences as *Example 1* and *2*.

Example 3:

Sentence 1: "Children in red shirts are playing with leaves."

Sentence 2: "Three kids are sitting in the leaves."

After calculating the weighted word similarity score for each pair and sorting all pairs by their weighted word similarity score, we can obtain the following candidate pairs:

(leaves, leaves), (children, kids), (playing, sitting), (are, are), (red, the), (shirts, three), (with, in).

The unweighted word similarity scores are:

[1.0, 0.814095, 0.463181, 1.0, 0.352678, 0.190496, 0.461608],

and the overall sentence similarity score is 0.611723.

Example 4:

Sentence 1: "A boy is standing outside the water."

Sentence 2: "The boy is wading through the blue ocean."

The candidate pairs are obtained by the weighted word similarity scores:

(boy, boy), (water, ocean), (outside, through), (standing, wading), (is, is), (a, blue), (the, the)

The unweighted word similarity scores are:

[1.0, 0.600632, 0.519733, 0.320120, 1.0, 0.158368, 1.0],

and the overall score is 0.656979.

As shown by *Example 3* and *4*, after introducing this weighting scheme, some relatively frequent but meaningless words move backward or directly be filtered out from the candidate pairs. Meanwhile, some meaningful word pairs with lower frequency have chance to move forward so as to be selected as the candidate pairs and contribute to the sentence similarity.

```
Algorithm 2: Word2Sent-V2, sentence similarity measurement with weighting scheme
INPUT: Word embeddings $\{v_w : w \in \mathcal{V}\}$, two sentences $s_1$(with length $m$) and $s_2$ (with length $n$), an empty set $\mathcal{P}$, an empty set $\mathcal{C}$, parameter $a$, estimated frequency $\{f(w) : w \in \mathcal{V}\}$ of each word.
OUTPUT: A sentence similarity score $sim\_wt$ for the input two sentences.
1.  Bundle every word of $s_1$ with every word of $s_2$ as a pair and put all pairs $\{(w_{1i}, w_{2j}) : i = 0,1,\dots,m-1, j = 0,1,\dots,n-1\}$ into $\mathcal{P}$.
2.  for each pair $p_x$ in $\mathcal{P}$ do:
        Weighted word similarity score :
            $W_{wt\_p_x} \leftarrow mean\left(\frac{a}{a+f(w_{1i})}, \frac{a}{a+f(w_{2j})}\right) \times cosine\_distance(v_{w_{1i}}, v_{w_{2j}})$
        Unweighted word similarity score:
            $W_{un\_p_x} \leftarrow cosine\_distance(v_{w_{1i}}, v_{w_{2j}})$
    end for
3.  while $\mathcal{P}$ is not empty do:
        Move the pair $p_h : (w_{1m}, w_{2n})$ into set $\mathcal{C}$, where $W_{wt\_p_h}$ is the maximal among all other pairs in $\mathcal{P}$; Delete pairs containing either $w_{1m}$ or $w_{2n}$ from $\mathcal{P}$.
    end while
4.  Compute the average $W_{un\_avg}$ of unweighted word similarity scores of all pairs in set $\mathcal{C}$, and $sim\_wt \leftarrow W_{un\_avg}$.
```

To demonstrate the influence of introducing the weighting scheme, we evaluated the proposed two versions on the test datasets with different word embeddings libraries. The details for experiment settings and results analysis can be found in Chapter 6.

5.2 Insight into SIF-model

The other contribution of this thesis focuses on the SIF-model proposed by Arora et al. [2]. On the one hand, we improved the overall performance of the SIF-model generally on all tasks by introducing a parameter σ. On the other hand, to adapt the SIF-model to the legal scenario and obtain better prediction results, we applied and experimented the post-processing technique to the model.

Recall that in the SIF-model, a discourse vector $\boldsymbol{c}_s$ of the current sentence represents the topics that are being talked about. When calculating the probability that a word w would be generated given the current discourse vector, an assumption is made as $\tilde{\boldsymbol{c}_s} := \beta \boldsymbol{c}_0 + (1-\beta)\boldsymbol{c}_s$, where $\boldsymbol{c}_0$ is the common discourse vector representing a correction term for the most frequent discourse related to syntax, such as "just", "but", "there" and so on. Besides, $\boldsymbol{c}_0$ is estimated by the first principal component of the given set of sentences, β are scalars hyperparameters, and $\boldsymbol{c}_0 \perp \boldsymbol{c}_s$. In the last, $\boldsymbol{c}_0$ is subtracted from the sentence embedding to generate the final sentence embedding. They observed that $\boldsymbol{c}_0$ is roughly corresponding to the syntactic information or the common words.

As demonstrated in the original paper [2], the post-processing step, subtracting the first principal component of a set of sentences ($\boldsymbol{c}_0$), is a very crucial step to boost the performance. Moreover, we found that it is also important for the post-processing step to be consistent with the theoretical assumption made when computing the probability of the generation of the specific word given the current discourse vector. Therefore, we introduced a parameter $\sigma \in (0,1)$ to the post-processing step, where $\boldsymbol{c}_s = \tilde{\boldsymbol{c}_s} - \sigma * \boldsymbol{c}_0$, to control the ratio of $\boldsymbol{c}_0$ that is subtracted from $\boldsymbol{c}_s$. The parameter σ is designed as the reflection of β in the assumption $\tilde{\boldsymbol{c}_s} \coloneqq \beta \boldsymbol{c}_0 + (1-\beta)\boldsymbol{c}_s$. Through the experiments in Chapter 6, we found that when $\sigma = 0.8$, the overall performance of the SIF-model could be boosted slightly (about 1%) for all textual similarity tasks evaluated in the original paper.

```
Algorithm 3: Improved SIF-model
INPUT: Word embeddings $\{v_w: w \in \mathcal{V}\}$, a set of sentences $S$, parameter $a$, estimated frequency $\{f(w): w \in \mathcal{V}\}$ of each word.
OUTPUT: Sentence embeddings $\{v_s: s \in S\}$
1.  for all sentences $s$ in $S$ do:
        $v_s \leftarrow \frac{1}{|s|}\sum_{w \in s} \frac{a}{a + f(w)} v_w$
    end for
2.  Compute the first principal component $u$ of $\{v_s: s \in S\}$
3.  for all sentence $s$ in $S$ do:
        $v_s \leftarrow v_s - \sigma \times uu^T v_s$
    end for
```

Moreover, the sentence similarity scores generated by the SIF-model range from -1 to 1, while most of the real-world scenario tasks require the scores in different scales. Take the task with the ground-truth label indicating the degree of sentence similarity ranging from 1 to 5 as an example, the first and simplest option is applying a linear transformation on top of SIF-model to map the prediction score from $[-1,1]$ to $[1,5]$. From the experiments, we found that it is not effective to boost the overall performance regarding Pearson Correlation Coefficient (Pearson's r) [108] or Mean Square Error (MSE). Secondly, we used a small labeled dataset (LawSents) with scores ranged from 1 to 5 to train an SVM score predictor in a supervised manner and applied it on top of the SIF-model as the non-linear transformation. Experiments in Chapter 6 demonstrated that the SVM-applied model could deliver better performance than the original SIF-model. The Pearson's r is boosted about 1~2%. The same idea can also be applied to our proposed model.

As the last part of the main work in this thesis, we explored the influence of different methods for measuring the degree of similarity between two word vectors. The SIF-model calculates Cosine Distance between two fixed-dimension vectors as the word similarity score, which compares the difference between two vectors in direction. Some models apply Euclidean Distance as the metric for the degree of similarity, which considers the absolute distance between two vectors. Instead of merely using Cosine Distance, we experimented several different settings for the evaluation. For example, only using Euclidean Distance, or using the different combinations of Cosine Distance and Euclidean Distance. However, the evaluation results show that simply using Cosine Distance as a measurement metric achieves the best performance, and the performance decreases rapidly as taking more Euclidean Distance into considerations in the combined measurement experiments. The Cosine Distance is better at catching the semantic similarity, the direction the word vectors indicating its meaning. Thus, for words with similar meanings, their embeddings would be similar. The similarity metrics applied in word embeddings training methods (Word2Vec, GloVe, and fastText) is Cosine Distance, and the SIF-model is based on the word embeddings library trained by one of these models to represent each word. Therefore, the metric being used in the model should keep the consistency as Cosine Distance.

Chapter 6
Evaluation

In this chapter, we first evaluate our proposed model with two different versions on two datasets, using distinctive hyper-parameters, and compared the performance with the baseline SIF-model. Then, we interpret and analyze the evaluation results regarding the characteristics of these models, word embeddings, and datasets.

6.1 Word Embeddings

The two versions of our proposed model highly rely on the word embeddings, since the degree of similarity between two sentences is determined by the degree of similarity between each candidate pairs. Several factors influence the properties of word embeddings such as different training methods, the domain, size, and the quality of the training data. Thus, these factors would further influence the experiment results. In our experiments, three different word embeddings libraries are utilized, namely, GloVe-Common Crawl (2.2M vocab, cased, 300d vectors), fastText-Common Crawl (2M vocab, cased, 300d vectors), and Word2Vec-LegalDocs. The dimension of all word vectors is 300.

GloVe and fastText are the pre-trained word embeddings libraries. Word2Vec-LegalDocs is trained on a corpus with 500K various types of unlabeled legal contracts by Beagle. Before training, we first removed various types of noises from all massive unstructured plain text files. Then the Skip-gram model is applied to proceed with the training process.

6.2 Datasets

We test our models on two textual similarity datasets, namely, SICK2014 [109] and LawSents (https://github.com/yvettewang/Word2Sent/tree/master/eval_data). The purpose is to predict the degree of similarity between every sentence pair in the dataset. The metric is the Pearson's r between the predicted similarity score and the gold-standard human judgments. LawSents consist of 200 pairs of long legal sentences. The average length of sentences in LawSents is around 27 words, while for SICK2014 is approximately 10 words per sentence. Sentences in LawSents cover the domain of legal language, while SICK2014 dataset involves compositional knowledge with diverse topics.

6.3 Experiments

The objective of the experiments is to evaluate the two versions of our proposed model and compare the performance with the baseline SIF-model. We experimented every model with three word embeddings libraries on two datasets. Specifically, we use "name of word embeddings" + "name of models" to represent each combination of the model and the word embeddings library. For instance, "fastText+SIF" indicates SIF-model with fastText word embeddings library and "LegalDocs + V2" indicates our proposed Word2Sent-V2 model with LegalDocs word embeddings library. All experiments are completely in an unsupervised manner.

Models	*LawSents*	*SICK2014*
fastText+SIF	51.81%	**71.00%**
fastText+V1	74.62%	64.72%
fastText+V2	**76.53%**	66.61%
GloVe+SIF	42.32%	**70.90%**
GloVe+V1	73.73%	65.92%
GloVe+V2	**79.00%**	68.62%
LegalDocs+SIF	56.74%	**57.59%**
LegalDocs+V1	73.80%	56.99%
LegalDocs+V2	**80.66%**	54.79%

Table 1 Experimental results ($Pearson's\ r \times 100$) on textual similarity tasks. The highest score in each row is in bold. "GloVe+V1" stands for applying our Word2Sent-V1 to the GloVe word vectors; "LegalDocs+V1" is for Word2Vec-LegalDocs word vectors. See the main text for the description of the methods.

6.4 Results & Analysis

The results can be found in Table 1. There are three splitting datasets in the SICK2014 dataset, train, dev, and test. For clarity, we only report the average result for the SICK2014 task.

For LawSents dataset, the advantages of our proposed model are very significant, as both versions of the proposed model beat the baseline SIF-model by 20% to 25%. Especially, the LegalDocs word embeddings with Word2Sent-V2 model achieves the highest Pearson's r at 80.66%, which strongly demonstrates the influence of the word embeddings library to the overall performance of the model. The LegalDocs word embeddings library is trained on massive legal documents, contrast with fastText and GloVe trained on the universal context. The relatively frequent words in legal language have a narrower range of meanings compared with them in the universal context. The LegalDocs word embeddings library is trained to capture these types of narrower and more specific meanings. Therefore, we may confidently presume that it is comparatively optimal to apply the domain-specific word embeddings to test datasets in the same domain.

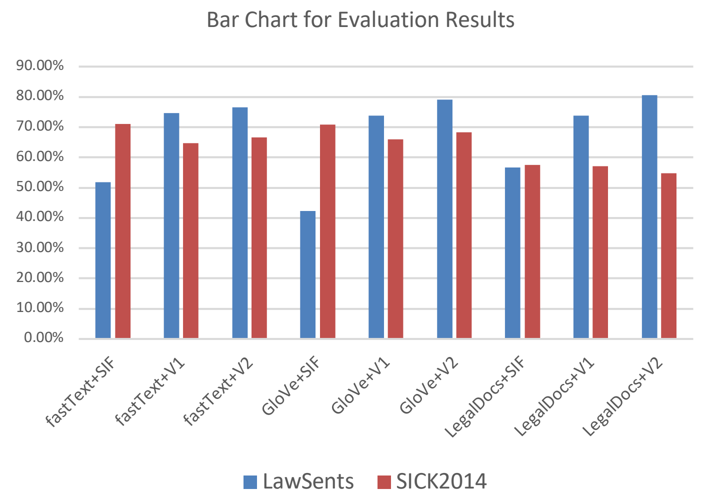


Figure 8 Evaluation results in bar chart for all proposed models and the baseline SIF-model on LawSents and SICK2014 dataset.

In the evaluation of SICK2014 dataset, the advantages of our models are not as significant as in LawSents. As we can observe from Table 1, the Pearson's r achieved by the SIF-model on either GloVe word embeddings or fastText word embeddings is around 71%. While the best performance achieved by our models is 69%, which is slightly lower than the SIF-model. It is quite hard to conjecture the specific factors causing this phenomenon, however, we may presume that our proposed model is much more suitable for longer sentences. As the length of the sentence increases, more information would be contained in the sentence. Therefore, when comparing the degree of similarity between two longer sentences, more complicated and mixed information should be considered. The SIF-model generates a sentence embedding for each sentence with 300-Dimension, which is the same as the dimension of word embeddings. In such a process of generating sentence embeddings, plenty of valuable information of the sentences would be lost due to this simple weighted average operation, and the longer the sentence, the more information will be missed. However, our proposed model measures the degree of similarity between two sentences by considering the semantic similarity of all candidate word pairs. The number of these candidate pairs are dynamically changed due to different lengths of the given sentences. If both sentences are longer, then more candidate pairs would be selected to influence and decide the final similarity score. If the length of two sentences varies a lot, more words of the longer sentence cannot be considered as the candidate pairs. But in terms of human intuition and the characteristics of legal language, if one sentence is very long while the other is quite short, it is less possible for these sentences to be similar regarding semantics. Therefore, the advantage of our proposed model becomes more magnificent when the input sentences become longer or vary a lot in length, while the performance of the other weighted average SIF-model decays severely in the same circumstance.

Then, we analyzed the difference of the overall performance of the models according to different word embeddings libraries. On SICK dataset, we can observe that any of the models with fastText or GloVe word embeddings achieves almost the same Pearson's r. One of the reasons is that fastText and GloVe word embeddings are trained with the almost the same context covering diverse topics. And the Common Crawl dataset is relatively cleaner than the various types of legal contracts for the training usage of LegalDocs. The performance decreases severely when any models plus LegalDocs with SICK dataset. One possible reason is that the LegalDocs is trained on massive legal documents, and the overall performance of the models are highly determined by the word embeddings library, in terms of quality, size, and domains. The LegalDocs word embeddings library is trained to capture the semantic and syntactic relationship within the domain of legal language. Thus, this very domain-specific word embeddings library contains relatively narrow information regarding the semantics of words, and some words appearing in the testing dataset even failed to be mapped to the word vector through the library. However, the SICK2014 dataset essentially covers a variety of topics. Therefore, merely considering semantic information features extracted by LegalDocs word embeddings cannot fulfill the actual semantic analysis in other domains.

Particularly, we looked at LegalDocs+V2 on LawSents dataset and obtained more insights through examples with relatively larger MSE between the ground-truth label and predicted label. For example, in the pair:

The requirement to protect Confidential Information disclosed under this Agreement shall survive termination of this Agreement.

This undertaking shall be governed by the laws of New South Wales and shall terminate upon cessation of obligations under the Confidentiality Deed in accordance with clause 6 (Term) of the Confidentiality Deed.

The ground-truth label indicating the degree of similarity between the above two sentences is one, since in terms of legal professional's interpretation, the two sentences have no semantic relation at all. However, all models find several candidate word pairs containing exactly the same words. Thus, the predicted score is much higher than it should be. Moreover, from the aspect of LegalDocs word embeddings library, the word similarity score of some pairs is relatively divergent from legal professional's judgement. For example, the pair (*cessation, termination*) has a very low word similarity score while they are actually synonyms to each other. This phenomenon is definitely affected by the quality of the word embeddings library. For example, in the training dataset of the word embeddings, some common-used phrases are not correctly separated, some words in the vocabulary set are misspelled, as well as lots of capital errors occur. These noises greatly diminish the ability of the word embeddings model to capture the proper semantic relationship between words. The overall performance of the model is decreased severely as expected. Therefore, we anticipated that if the quality of the word embeddings is enhanced, then the overall performance of the model would have a noticeable growth.

In Table 2, we refer to the evaluation results obtained by [90] for different unsupervised models testing on SICK2014 dataset and compare with our evaluation results. The Skip-gram and C-BOW represent the sentence embedding obtained by averaging of word embeddings generated by those two models. Unigram TF-IDF represents the weighted averaging model using TF-IDF frequencies as the weighting scheme. As we can observed from Table 2, FastSent model achieves the best performance on Pearson's r as 72%, followed by the SIF-model with fastText word embeddings, Sent2Vec, and our proposed model Word2Sent-V2 with GloVe word embeddings, which achieves 69% Pearson's r.

Models	*SICK2014*
SAE	31%
SAE + embs.	49%
SDAE	46%
SDAE + embs.	46%
ParagraphVec DBOW	46%
ParagraphVec DM	40%
Skip-gram	69%

C-BOW	69%
Unigram TF-IDF	58%
Sent2Vec uni. + bi.	70%
SkipThought	60%
FastSent	**72%**
FastSent+AE	65%
fastText+SIF	71%
GloVe+V2	69%

Table 2 Comparison of the performance of different unsupervised models on $Pearson's\ r \times 100$ (results collected from [90]). An underline indicates the best performance for the SICK2014 dataset. The top performances are shown in bold.

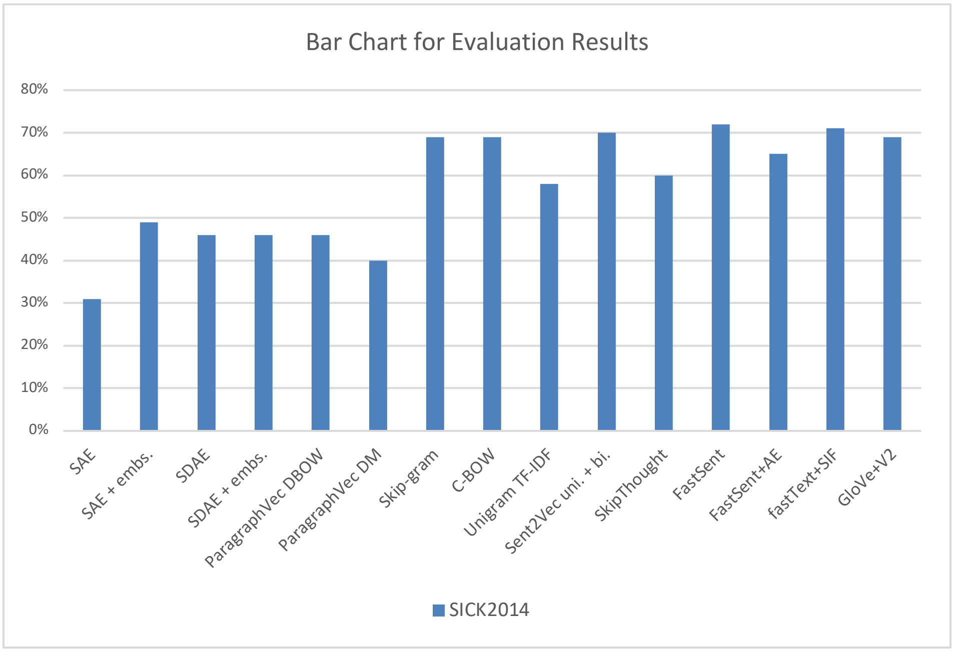


Figure 9 Evaluation results in bar chart for different unsupervised models on Pearson's r on SICK2014 dataset.

Next, for the SIF-model, we experimented different values of the parameter σ, from 0.5 to 1. σ scales the first principal component that is being removed from every sentence embedding, where $\sigma = 1$ means the entire first principal component is removed, $\sigma = 0.5$ means half of the first principal component is removed. We evaluated the overall performance of the improved SIF-model on the SICK2014 test dataset by only adjusting the value of parameter σ. The evaluation results can be found in Figure 10. As the value of σ increases, the Pearson's r also increases steadily. Until $\sigma = 0.8$, the Pearson's r achieves at the peak value, 75.01%. The Pearson's r then decreases significantly as the value of σ continues increasing.

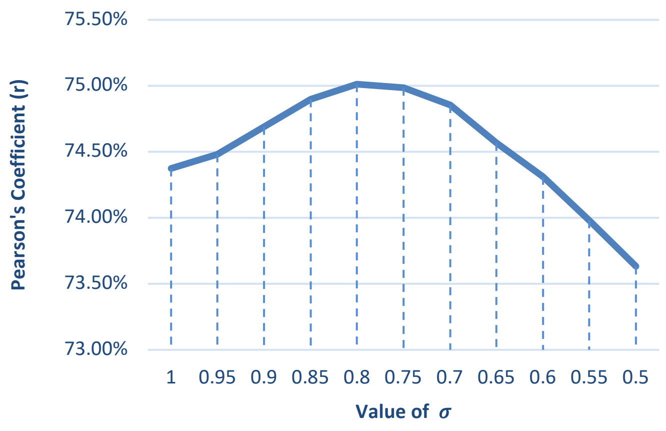


Figure 10 Tested on SICK2014 dataset, the Pearson's r becomes higher as the value of σ increases.

Figure 11 shows the influence of the SVM non-linear score predictor. As described in Chapter 5, we trained an SVM score predictor with LawSents dataset in a supervised manner and applied it on top of the SIF-model. We tested this combined model on the other dataset, SICK2014 test dataset, it demonstrated that the strength of this slight change is quite significant since the overall performance is boosted by the SVM predictor about 1.5%.

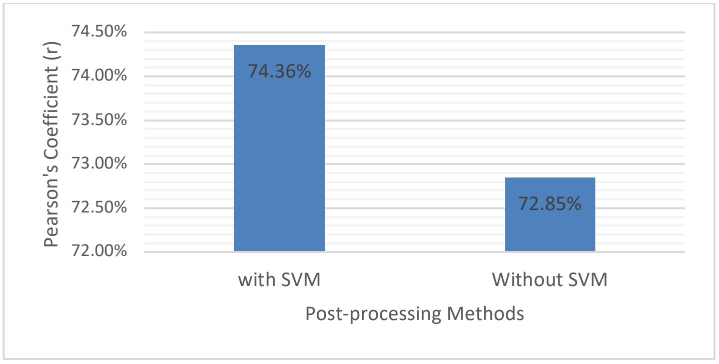


Figure 11 Tested on SICK2014 test dataset, with σ=1, the model with an SVM predictor beats the one without the SVM by about 1.5%.

In Chapter 5, we attempted to apply different methods for measuring the degree of similarity between two words in a pair. In the experiments (see Figure 12), we displayed different results by applying different ratios of linear combination between Cosine Distance and Euclidean Distance as the metric for the degree of similarity. From the results, we demonstrated that the overall performance of the model continues decaying as more Euclidean Distance being considered as the metric. Thus, we simply applied Cosine Distance as the metric for the degree of similarity between two words in all experiments.

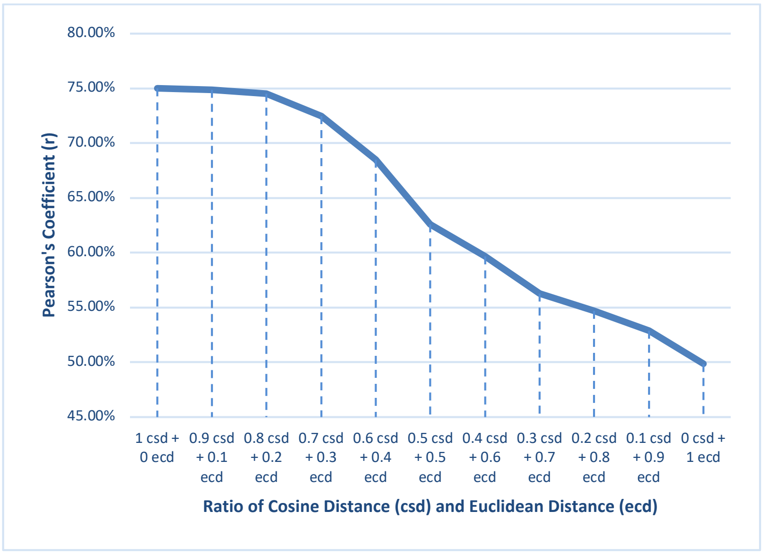


Figure 12 Evaluation result related to different types of similarity metrics.

Chapter 7
Conclusion and Future Works

In this thesis, we studied about the development of representation learning and inspected the concept of distributional hypothesis. Then, based on the conception of semantic compositionality, we reviewed a variety of sentence embeddings models for distributed representation of higher context levels. As the main contribution, on the one hand, we introduced a novel model named Word2Sent for legal sentence similarity analysis in a simple but entirely unsupervised manner. The idea behind the proposed model stands on both distributional hypothesis and semantic compositionality. On the other hand, we also improved the overall performance of the SIF-model by controlling the ratio of the common discourse vector being removed. Then, we evaluated the proposed model as well as the baseline SIF-model with different word embeddings on several datasets in diverse domains. From the evaluation results, we observed that the proposed weighted model with LegalDocs word embeddings achieves the best performance for the sentence similarity analysis of legal language. Lastly, we found that applying the fine-tuned SVM score predictor on top of the model can effectively enhance the final performance. To improve the performance of the proposed model, attention could be paid to the following aspects as the future work:

Improving the quality of the word embeddings library: As digging into the training data of the LegalDocs word embeddings library, we found that several issues related to the cleanness of the data may reduce the quality of LegalDocs. For example, some common-used phrases are not perfectly separated, some words in the vocabulary set are misspelled, as well as lots of capital errors occur. Therefore, the overall performance of the model is decreased severely. Removing those noises in the training data is quite time-consuming and intricate. However, it is indeed one of the most critical factors to boost the quality of the word embeddings library.

Improving the weighting scheme: By observing the evaluation results in Chapter 6, we concluded that the weighting scheme contributes a lot for the enhancement of the overall performance. With the current weighting scheme, the weight of each word is only relevant to its frequency. Words with higher frequency have comparatively lower weights, while the rare words are assigned with higher weights. However, this weighting scheme might not be the best. Instinctively, the distribution of words for different datasets can be varied a lot. Thus, designing the corpus-specific weighing scheme might be helpful to heighten the performance of the model. For example, the word "not" appears very frequently; thus, the weight of this word is relatively low among all words in the vocabulary set. However, the word "not" often expresses a negative of a statement, which plays a turning point role in a sentence. When comes with a pair of almost identical sentences, where one expresses positively, while the other expresses negatively with a "not", a more rational weighting scheme should assign "not" a relatively larger weight to distinguish the utterly contradictory meaning of two sentences.

Improving the model: Our model concentrates on the degree of semantic similarity of every unigram word pair, then computes the overall sentence similarity score for the given sentences. However, human language is frustratingly ambivalent and complicated in some circumstances. For example, polysemy, homonymy, and collocations are significant language phenomenon in every language. Take the phrase "Toronto Blue Jays" as an example, when a single word "Jays" appears in a sentence, it might be understood as the particular strain of birds. However, if the preceding word of "Jays" is "Blue", then the phrase "Blue Jays" refers to the particular baseball association. Due to the characteristic of semantic compositionality, the meaning of a sentence is composed of the meanings of its components, which might be the single words or longer phrases. Thus, the other potential aspect of improving the overall performance is to enhance the model itself, by means of considering the similarity of phrases in the entire sentence. It might be helpful to deliver a more comprehensive analysis between two sentences, especially from the perspective of semantics.

Reference

[1] Understanding the New Language of Legal Technology - Law http://www.lawtechnologytoday.org/2018/01/the-new-language-of-legal-technology/

[2] Arora, S., Li, Y., Liang, Y., Ma, T. (2017). A Simple but Tough-to-Beat Baseline for Sentence Embeddings, International Conference on Learning Representations.

[3] Ian Goodfellow and Yoshua Bengio and Aaron Courville (2016) “Deep Learning”, http://www.deeplearningbook.org, MIT press

[4] Bengio, Y., Courville, A., and Vincent, P. (2013d). Representation learning: A review and new perspectives. IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI), 35(8), 1798–1828.

[5] Hinton, G. E. (1986). Learning distributed representations of concepts. In Proc. 8th Conf. Cog. Sc. Society, pages 1–12

[6] Bengio, Y. and LeCun, Y. (2007). Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, Large Scale Kernel Machines. MIT Press.

[7] Firth, J.R. (1957). "A synopsis of linguistic theory 1930-1955". Studies in Linguistic Analysis. Oxford: Philological Society: 1–32. Reprinted in F.R. Palmer, ed. (1968). Selected Papers of J.R. Firth 1952-1959. London: Longman.

[8] Harris, Z. (1954). "Distributional structure". Word. 10 (23): 146–162.

[9] Karl Pearson. 1901. Principal components analysis. The London, Edinburgh and Dublin Philosophical Magazine and Journal 6, 2 (1901), 566.

[10] Jerome H Friedman. 1997. On bias, variance, 0/1loss, and the curse-of-dimensionality. Data mining and knowledge discovery 1, 1 (1997), 55-77.

[11] Ivan Markovsky. 2012. Low Rank Approximation: Algorithms, Implementation, Applications. (January 2012). http://eprints.soton.ac.uk/273101/

[12] Ferrone, Lorenzo & Zanzotto, Fabio Massimo. (2017). Symbolic, Distributed and Distributional Representations for Natural Language Processing in the Era of Deep Learning: a Survey.

[13] Lee, T. (1998). Independent Component Analysis. Independent Component Analysis, 27-66. doi:10.1007/978-1-4757-2851-4_2

[14] Sahlgren, Magnus. (2004). An Introduction to Random Indexing. Language. 1-9.

[15] Dumais, S. T. (n.d.). LSA and Information Retrieval. Handbook of Latent Semantic Analysis. doi:10.4324/9780203936399.ch16

[16]] G. Salton 1989 Automatic text processing: The transformation, analysis, and retrieval of information by computer. Choice Reviews Online, 27(01). doi:10.5860/choice.27-0351

[17] Hu, X., Cai, Z., Wiemer-Hastings, P., Graesser, A. C., & Mcnamara, D. S. (n.d.). Strengths, Limitations, and Extensions of LSA. Handbook of Latent Semantic Analysis. doi:10.4324/9780203936399.ch20

[18] Lund, Kevin & Burgess, Curt. (1996). Producing high-dimensional semantic space from lexical co-occurence. Behavior Research Methods Instruments & Computers. 28. 203-208. 10.3758/BF03204766.

[19] Church, K. W., & Hanks, P. (1989). Word association norms, mutual information, and lexicography. Proceedings of the 27th Annual Meeting on Association for Computational Linguistics -. doi:10.3115/981623.981633

[20] Bengio, Y & Ducharme, Réjean & Vincent, Pascal. (2000). A Neural Probabilistic Language Model. Journal of Machine Learning Research. 3. 932-938. 10.1162/153244303322533223.

[21] Mikolov, Tomas & Karafiát, Martin & Burget, Lukas & Cernocký, Jan & Khudanpur, Sanjeev. (2010). Recurrent neural network-based language model. Proceedings of the 11th Annual Conference of the International Speech Communication Association, INTERSPEECH 2010. 2. 1045-1048.

[22] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. JMLR, 3, 1137–1155.

[23] Mikolov, Tomas & Chen, Kai & Corrado, G.s & Dean, Jeffrey. (2013). Efficient Estimation of Word Representations in Vector Space. Proceedings of Workshop at ICLR. 2013.

[24] Mikolov, Tomas & Sutskever, Ilya & Chen, Kai & Corrado, G.s & Dean, Jeffrey. (2013). Distributed Representations of Words and Phrases and their Compositionality. Advances in Neural Information Processing Systems. 26.

[25] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-536. doi:10.1038/323533a0

[26] Mnih, A & Kavukcuoglu, K. (2013). Learning word embeddings efficiently with noise-contrastive estimation. Proceedings of Neural Information Processing Systems-NIPS. 2265-2273.

[27] Alexandrescu, A., & Kirchhoff, K. (2006). Factored neural language models. Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers on XX - NAACL 06. doi:10.3115/1614049.1614050

[28] Lazaridou, Angeliki & Marelli, Marco & Zamparelli, Roberto & Baroni, Marco. (2013). Compositional-ly derived representations of morphologically complex words in distributional semantics. ACL 2013 - 51st Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference.

[29] A. Botha, Jan & Blunsom, Phil. (2014). Compositional Morphology for Word Representations and Language Modelling. 31st International Conference on Machine Learning, ICML 2014. 5.

[30] Cui, Qing & Gao, Bin & Bian, Jiang & Qiu, Siyu & Liu, Tie-Yan. (2014). Learning Effective Word Embedding using Morphological Word Similarity.

[31] Xinxiong Chen, Lei Xu, Zhiyuan Liu, Maosong Sun, and Huanbo Luan. 2015. Joint learning of character and word embeddings. In Proc. IJCAI.

[32] Thang, Luong & Socher, Richard & Manning, Christoper. (2013). Better Word Representations with Recursive Neural Networks for Morphology. Proceedings of the Seventeenth Conference on Computational Natural Language Learning. 104-113.

[33] Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., . . . Luis, T. (2015). Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. doi:10.18653/v1/d15-1176

[34] Ling, Wang & Trancoso, Isabel & Dyer, Chris & W Black, Alan. (2015). Character-based Neural Machine Translation.

[35] Luong, M., & Manning, C. D. (2016). Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). doi:10.18653/v1/p16-1100

[36] Kim, Yoon & Jernite, Yacine & Sontag, David & M. Rush, Alexander. (2015). Character-Aware Neural Language Models.

[37] Zhang, Xiang & Zhao, Junbo & Lecun, Yann. (2015). Character-level Convolutional Networks for Text Classification. In Advances in Neural Information Processing Systems 28.

[38] Jozefowicz, Rafal & Vinyals, Oriol & Schuster, Mike & Shazeer, Noam & Wu, Yonghui. (2016). Exploring the Limits of Language Modeling.

[39] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. CHARAGRAM: Embedding words and sentences via character n-grams. In Proc. EMNLP.

[40] Levy, O & Goldberg, Yoav & Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. Transactions of the Association for Computational Linguistics. 3. 211-225.

[41] Mikolov, Tomas & Grave, Edouard & Bojanowski, Piotr & Puhrsch, Christian & Joulin, Armand. (2017). Advances in Pre-Training Distributed Word Representations.

[42] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural Networks, 61, 85-117. doi:10.1016/j.neunet.2014.09.003

[43] Bengio, Yoshua; LeCun, Yann; Hinton, Geoffrey (2015). "Deep Learning". Nature. 521 (7553): 436–444. doi:10.1038/nature14539. PMID 26017442.]

[44] Ivakhnenko, A. G. (1973). Cybernetic Predicting Devices. CCM Information Corporation.

[45] Ivakhnenko, Alexey (1971). "Polynomial theory of complex systems". IEEE Transactions on Systems, Man and Cybernetics. 1 (4): 364–378. doi:10.1109/TSMC.1971.4308320.

[46] Lecun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation, 1(4), 541-551. doi:10.1162/neco.1989.1.4.541

[47] Hinton, G. E.; Osindero, S.; Teh, Y. W. (2006). "A Fast Learning Algorithm for Deep Belief Nets" (PDF). Neural Computation. 18 (7): 1527–1554.

[48] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In NIPS'2006.

[49] Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. In NIPS'2006.

[50] Hinton, G. E. (2007). Learning multiple layers of representation. Trends in Cognitive Sciences, 11(10), 428-434. doi:10.1016/j.tics.2007.09.004

[51] Seide, F., Li, G., and Yu, D. (2011a). Conversational speech transcription using context-dependent deep neural networks. In Interspeech 2011, pages 437–440.

[52] Dahl, G. E., Yu, D., Deng, L., and Acero, A. (2012). Context dependent pre-trained deep neural networks for large vocabulary speech recognition. IEEE Transactions on Audio, Speech, and Language Processing, 20(1), 33–42.

[53] Ciresan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. Technical report, arXiv:1202.2745.

[54] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In NIPS'2012.

[55] Hinton, G. E. (1986). Learning distributed representations of concepts. In Proc. 8th Conf. Cog. Sc. Society, pages 1–12.

[56] Mikolov, T., Deoras, A., Kombrink, S., Burget, L., and Cernocky, J. (2011). Empirical evaluation and combination of advanced language modeling techniques. In INTERSPEECH'2011.

[57] Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. doi:10.1109/5.726791

[58] Stanford University CS231n: Convolutional Neural Networks for Visual Recognition by Andrej Karpathy

[59] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84-90. doi:10.1145/3065386

[60] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2015.7298594

[61] Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[62] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385.

[63] Hochreiter, S. (1998). The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 06(02), 107-116. doi:10.1142/s0218488598000094

[64] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 157-166. doi:10.1109/72.279181

[65] Pascanu, R., Mikolov, T., and Bengio, Y.. On the difficulty of training recurrent neural networks. arXiv preprint arXiv:1211.5063, 2012.

[66] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780. doi:10.1162/neco.1997.9.8.1735

[67] Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to Forget: Continual Prediction with LSTM. Neural Computation, 12(10), 2451-2471. doi:10.1162/089976600300015015

[68] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An Empirical Exploration of Recurrent Network Architectures. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15). 2342–2350.

[69] Cho, K., Merrienboer, B. V., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). doi:10.3115/v1/d14-1179

[70] Schuster, M., & Paliwal, K. (1997). Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing,45(11), 2673-2681. doi:10.1109/78.650093

[71] Liwicki, Marcus, et al. "A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks." Proc. 9th Int. Conf. on Document Analysis and Recognition. Vol. 1. 2007.

[72] Graves, A., Santiago F., and Jürgen S. "Bidirectional LSTM networks for improved phoneme classification and recognition." Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005. Springer Berlin Heidelberg, 2005. 799-804.

[73] Baldi, Pierre, et al. "Exploiting the past and the future in protein secondary structure prediction." Bioinformatics 15.11 (1999): 937-946.

[74] Yao, Kaisheng, Cohn, Trevor, Vylomova, Katerina, Duh, Kevin & Dyer, Chris. (2015). Depth-Gated Recurrent Neural Networks.

[75] Kalchbrenner, N., Danihelka, I., and Graves, A. (2015). Grid long short-term memory. arXiv preprint arXiv:1507.01526

[76] Graves, A., Mohamed, A., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In ICASSP'2013, pages 6645–6649.

[77] Pascanu, R., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014a). How to construct deep recurrent neural networks. In ICLR'2014

[78] Pollack, J. B. (1990). Recursive distributed representations. Artificial Intelligence,46(1),77–105

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

[80] "Lexical Semantics and Compositionality", in Invitation to Cognitive Science, 2nd edition. Daniel Osherson, general editor; in Part I: Language, Lila Gleitman and Mark Liberman, eds. MIT Press, Cambridge (1995), pp. 311-360.

[81] Fodor, J. A., & Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. Cognition, 28(1-2), 3-71. doi:10.1016/0010-0277(88)90031-5

[82] Holyoak, Keith & Hummel, John. (2012). The proper treatment of symbols in a connectionist architecture.

[83] Miller, N., Oldham, G., & Pinker, S. (1994). The Language Instinct: How the Mind Creates Language. The Antioch Review, 52(3), 534. doi:10.2307/4613021

[84] V. Le, Quoc & Mikolov, Tomas. (2014). Distributed Representations of Sentences and Documents. 31st International Conference on Machine Learning, ICML 2014. 4.

[85] Hill, F., Cho, K., & Korhonen, A. (2016). Learning Distributed Representations of Sentences from Unlabelled Data. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. doi:10.18653/v1/n16-1162

[86] Kenter, T., Borisov, A., & Rijke, M. D. (2016). Siamese CBOW: Optimizing Word Embeddings for Sentence Representations. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). doi:10.18653/v1/p16-1089

[87] Wieting, John & Bansal, Mohit & Gimpel, Kevin & Livescu, K. (2015). Towards Universal Paraphrastic Sentence Embeddings.

[88] Gershman, S., & Tenenbaum, J.B. (2015). Phrase similarity in humans and machines. CogSci.

[89] Wieting, J., Bansal, M., Gimpel, K., & Livescu, K. (2016). Charagram: Embedding Words and Sentences via Character n-grams. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. doi:10.18653/v1/d16-1157

[90] Pagliardini, M., Gupta, P., & Jaggi, M. (2018). Unsupervised Learning of Sentence Embeddings Using Compositional n-Gram Features. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). doi:10.18653/v1/n18-1049

[91] Kiros, Ryan & Zhu, Yukun & R. Salakhutdinov, Ruslan & S. Zemel, Richard & Torralba, Antonio & Urtasun, Raquel & Fidler, Sanja. (2015). Skip-Thought Vectors. Advances in Neural Information Processing Systems. 28.

[92] Cho, K., Merrienboer, B. V., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). doi:10.3115/v1/d14-1179

[93] Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., . . . Luis, T. (2015). Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. doi:10.18653/v1/d15-1176

[94] Liu, P., Qiu, X., Chen, X., Wu, S., & Huang, X. (2015). Multi-Timescale Long Short-Term Memory Neural Network for Modelling Sentences and Documents. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. doi:10.18653/v1/d15-1280

[95] Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. Proceedings of the 53rd Annual Meeting of

the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). doi:10.3115/v1/p15-1150

[96] Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). doi:10.3115/v1/d14-1162

[97] Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). doi:10.3115/v1/p14-1062

[98] Hinton, G. E. (1989). Connectionist learning procedures. Artificial Intelligence, 40(1-3), 185-234. doi:10.1016/0004-3702(89)90049-0

[99] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. J. (1990). Phoneme Recognition Using Time-Delay Neural Networks. Readings in Speech Recognition, 393-404. doi:10.1016/b978-0-08-051584-7.50037-1

[100] Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing. Proceedings of the 25th International Conference on Machine Learning - ICML 08. doi:10.1145/1390156.1390177

[101] Mikolov, T., Kombrink, S., Burget, L., Cernocky, J., & Khudanpur, S. (2011). Extensions of recurrent neural network language model. 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). doi:10.1109/icassp.2011.5947611

[102] Socher, R., Karpathy, A., Le, Q.V., Manning, C.D., & Ng, A.Y. (2014). Grounded Compositional Semantics for Finding and Describing Images with Sentences. TACL, 2, 207-218.

[103] Srivastava, R.K., Greff, K., & Schmidhuber, J. (2015). Training Very Deep Networks. NIPS.

[104] Werbos, P. (1990). Backpropagation through time: What it does and how to do it. Proceedings of the IEEE, 78(10), 1550-1560. doi:10.1109/5.58337

[105] Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. CoRR, abs/1308.0850.

[106] Socher, R., Pennington, J., Huang, E.H., Ng, A.Y., & Manning, C.D. (2011). Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. EMNLP.

[107] Arora, S., Li, Y., Liang, Y., Ma, T., & Risteski, A. (2016). A Latent Variable Model Approach to PMI-based Word Embeddings. TACL, 4, 385-399.

[108] Pearson, Karl. Note on regression and inheritance in the case of two parents. Proceedings of the Royal Society of London, 58: 240–242, 1895.

[109] Marelli, Marco, Menini, Stefano, Baroni, Marco, Bentivogli, Luisa, Bernardi, Raffaella, and Zamparelli, Roberto. A sick cure for the evaluation of compositional distributional semantic models. In LREC, pp. 216–223, 2014.