Deep Belief Networks for Sentiment Analysis

by

Yong Jin

Master of Science in Statistics, Wuhan University of Technology in China, 2012
Bachelor of Science in Statistics, Wuhan University of Technology in China, 2009

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

In the Graduate Academic Unit of Faculty of Computer Science

Supervisor(s):	Huajie Zhang, PhD in Computer Science/Machine Learning
	Donglei Du, PhD in Business Administration/Operations Research
Examining Board:	Weiqiu Yu, PhD, Economics, Chair
	Weichang Du, PhD, Computer Science
	Wei Song, PhD, Computer Science
External Examiner:	Dan Wu, PhD, Computer Science/Machine Learning, University of Windsor

This dissertation is accepted by the

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

May, 2017 of submission to Graduate School

Abstract

Sentiment analysis is a highly popular issue both in academic and engineering fields. Nowadays there is an increasingly large amount of online opinion resources, so people are inclined to develop some systems that can automatically determine the polarities of opinions, especially in the decision-making process of a company. On the other hand, deep learning is a recently developed popular topic and has received much attention in machine learning area. Deep belief network (DBN) is one important deep learning model, which has proved powerful in many domains including natural language processing. However, there still exist some big challenges for DBNs in sentiment analysis because of the complexity to express opinions. Therefore, this study tries to improve DBNs in sentiment analysis area from the following three aspects: (1) The neuron models are investigated in DBNs for sentiment prediction. We perform various experiments and apply both total accuracy and F-measure to evaluate the performance, which proves that Gaussian neuron model with specific parameter setting has better effect. (2) In addition

to the traditional bag-of-words representation for each sentence, the word positional information is considered in the input. We propose a new word positional contribution form and another novel word-to-segment matrix representation for text to incorporate the positional information into DBNs for sentiment analysis. Finally, we evaluate the performance via the total accuracy. The results show that the word positional information of sentences helps to improve the performance of DBNs for sentiment classification. (3) We propose a new method to improve the DBN learning algorithm based on the unsupervised training phase of restricted Boltzmann machines (RBMs). That is, the RBM generates the hidden layer in an unsupervised fashion, and then we use this hidden layer as the output of a single-layer neural network, which is trained via the delta rule (DR). The new weights trained from DR are then transmitted into the whole network for initialization of back propagation (BP). This way keeps more correction signal for each layer in the BP algorithm compared to the ordinary DBN training. Our experimental results demonstrate that this updated learning method performs better than the ordinary learning for sentiment classification.

Dedication

I would like to dedicate this dissertation to my grandmother, from whom I have got unconditional love and caring since I was brought up by her. She always tries to offer me the best from the date when I was born. I can not study well and keep healthy without her excellent care and earnest instructions. Also I would not be who I am today without the love and support of my grandfather. He loves me very much and always encourages me to become a strong and successful man. Although he can not wait for me to graduate from this PhD program, his contributions to my life will accompany me forever.

Acknowledgements

First of all, I would like to express my sincere heartfelt thanks to my supervisors, Profs. Huajie Zhang and Donglei Du, for their invaluable advice, constant encouragement and precise modification, and I admire their broad and profound knowledge. They taught me so much in every aspect of science, from experimental design to data analysis, from critical thinking to scientific writing. Thanks again for Prof. Zhang's help and informative suggestions in my life in Canada.

Meanwhile, my thanks go to all the teachers who have taught me these years, who have helped me enrich and broaden my knowledge. Especially Dr. Guohua Yan at UNB Statistics, he helped me much when I came abroad to Canada at the first time in my life. I am really regret that we could not continue working together due to my personal career development reasons. My thanks also go to the other advisory committee members, Drs. Bruce Spencer, Weichang Du, Wei Song, Weiqiu Yu, and Dan Wu. Thanks for their reviews and valuable comments in my dissertation accomplishment process.

Finally, I would like to express my sincere thanks to my family members and relatives who have been pouring out their care, support and encouragement to me. My beloved wife, Xi Chen, accompanies me to come here for my PhD studies, and she always takes care of me even sometimes I am depressed. Although we are occasionally struggled with life in Canada, we have opened our eyes to the world and extended our international view, and also we lead a happy and fulfilling life during these years. Thanks to my father for his ongoing love and support, and his respect for my studying abroad. Thanks to my grandmother Xianyun Jiang and my aunt Yinfeng Jin, for their constant love and care. Last thanks to my parents-in-law, for their understanding, respect and help during my PhD studies.

Table of Contents

List of Tables

List of Figures

List of Symbols, Nomenclature or Abbreviations

Symbols:

$\sum$	\sum, aggregate summation
∂	\partial, partial derivative
log	\log, natural logarithmic function
Δ	\Delta, change of any changeable quantity
$\int$	\int, integral function

Abbreviations:

NLP – Natural Language Processing
SVM – Support Vector Machine
NB – Naive Bayes
LM – Language Model
POS – Part-of-Speech Tagging
NN – Neural Network
DBN – Deep Belief Network
CNN – Convolutional Neural Network
RNN – Recurrent Neural Network
LSTM – Long Short-Term Memory
BP – Back Propagation
RBM – Restricted Boltzmann Machine
CD – Contrastive Divergence
PCD – Persistent Contrastive Divergence
DR – Delta Rule

Chapter 1

Introduction

1.1 Background and Objectives

Sentiment analysis, or opinion mining, popular both in academic and engineering fields, is to identify the attitude of a speaker or a writer on a topic. Due to the increasingly large amount of online opinion resources, such as discussion forums, product review sites and social media websites, our social life has been profoundly impacted in every aspect. Hence, people are inclined to develop systems that can automatically determine the polarities of opinions. These systems can be useful in many ways. Consider marketing a new product, a company can track the popularity and discover the particular features of a product for which people like or dislike in order to judge and determine the success probability of the new product's launch. Another

example is that a review for a laptop could be generally positive but there is one negative exception about its weight. This will give customers a clearer picture on their purchasing decisions. Specifically, if a customer cares more about the weight of a laptop, she/he will probably not purchase it, otherwise the laptop will be a good choice. Thus, an accurate approach for automatic sentiment prediction regarding the overall sentiment or sentiment of one feature is valuable and necessary. This real-life issue attracts many researchers' interest for sentiment analysis [27, 70, 38].

Typically, some traditional methods, such as support vector machines (SVMs) and Bayesian network models [53], have been widely employed in sentiment analysis. However, considering that the semantic and structural factors influence text sentiments, the recently developed deep learning methods described in [4] can be applied in opinion mining. Deep learning is a popular topic and has received much attention in machine learning area during recent years. Deep belief network (DBN) is one important deep learning model, which improves the training of deep neural networks and has proved powerful in many domains including natural language processing. Figure 1.1 shows a deep neural network with five layers including visible layer, three hidden layers and output layer. It is typically used in supervised learning to make classification or prediction [4]. While the DBN model is a graphical model consisting of multiple layers of hidden variables, and is employed to abstract higher representations from raw inputs. Figure 1.2 depicts a basic DBN architecture with three hidden layers, which is normally used to pre-train useful feature

representations.

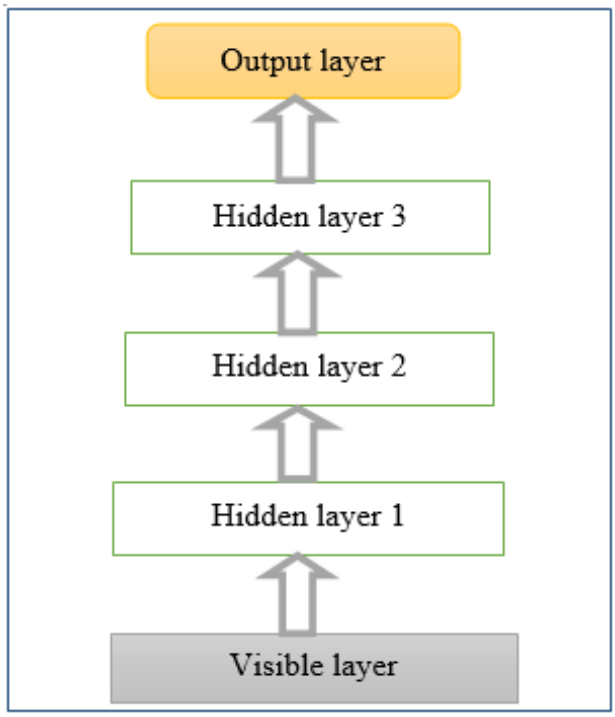

Figure 1.1: A graph example of multi-layer neural network.

A basic DBN has one restricted Boltzmann machine (RBM) at the top two adjacent hidden layers of the network, in which the units between two layers are bidirectional and there are no connections within the same layer [4] (detailed in Chapter 2). For example, the DBN in Figure 1.2 has one RBM between hidden layer 2 and hidden layer 3, while the connections of hidden layer 1 and hidden layer 2, visible layer and hidden layer 1, are directed (directed arcs indicate direct dependence, while bidirectional arcs mean mutual dependence).

Even though deep learning has proved effective in natural language process-

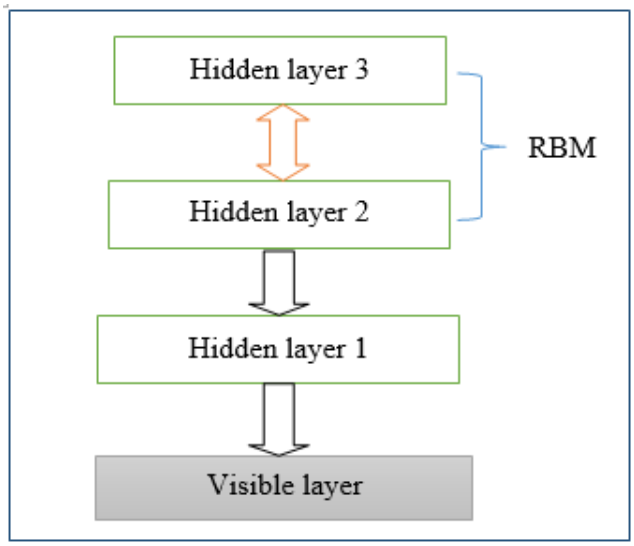

Figure 1.2: A DBN structure including three hidden layers.

ing (NLP) [10, 46, 67, 64], there still exist some big challenges for DBNs in sentiment analysis area because of the complexity to express opinions. Therefore, this research work tries to improve DBNs in sentiment analysis from the following three perspectives: (1) Considering the neuron model is an important internal component in DBNs, we want to explore the effect of neuron models in DBNs for sentiment prediction, and then work out a better neuron model through experiments. (2) In addition to the traditional bag-of-words representation for each sentence, the word positional information is a significant external component for DBNs in sentiment analysis. We are to explore how to incorporate the positional information to generate more useful input features based on the performance of DBNs in sentiment analysis. (3) We intend to improve the learning mechanism for DBNs. Through the investigation of ordinary two-phase DBN training, we are devoted to propose

a new algorithm to train DBNs on sentiment data sets.

1.2 Motivations

Based on the current work, some deep learning models such as convolutional neural networks [10], recursive neural networks [67], and recurrent neural networks [43] are very popular in NLP problems and also have achieved great success. However, DBNs have much to do for the sentiment analysis task in academia because there is not much work about DBNs for sentiment analysis. There are probably some obstacles for applying DBNs to sentiment analysis, because (1) each sentence has variable lengths and each word can not express its context very well towards the sentiment of a sentence; (2) the useful word vector (or word embedding) representation may not be directly applied in DBNs for NLP problems; and (3) DBNs may not express the structure of language in an accurate way. Given that DBNs perform promisingly well in the domains such as image and video, we believe that DBNs can also perform well in sentiment analysis, but need us to explore.

Hence, we are trying to investigate the performance of DBNs for sentiment analysis, and help this academic research direction to move forward step by step. Our work tries to start from the basic bag-of-words representation as the raw input, and then fit the word relationships in the hidden layers of DBNs. For the overall purpose of improving DBNs in sentiment analysis area,

we want to address some research questions from the following motivations corresponding to the mentioned obstacles.

Motivation One: The traditional bag-of-words vector provides a fixed length of inputs for each data set. On the other hand, the neuron model defines the form of the neuron's output, so it naturally affects the performance of a specific problem. Especially in sentiment analysis, a good neuron model should output a good value for the words that can capture some hidden relationships towards the sentiment of a sentence. But there is little work about the effect of neuron model in DBNs for sentiment analysis. So we argue that there would be a more effective neuron model in DBNs for sentiment prediction.

Motivation Two: Since word vectors may not be directly applied in DBNs, we try to find another effective representations for sentences. Based on the bag-of-words representation for each text, does the positional information (e.g. word order or word position) play an important role in DBNs for sentiment analysis? Are there any more useful representations to express positional information?

Motivation Three: The weights of DBNs can define the structure of a language text if the DBNs are trained properly. We think that a good structure for texts can be set up if a good training algorithm is found. Currently, DBNs are normally trained in a greedy layer-wise fashion including the RBM unsupervised training and the supervised BP training. However, regarding

sentiment analysis, can this DBN training mechanism be improved through a better learning algorithm?

1.3 Dissertation Structure

The rest of this dissertation is organized as follows. In Chapter 2, some literature review is introduced and some preliminary ideas are proposed correspondingly. In Chapter 3, the first idea - Gaussian neuron in DBNs for sentiment prediction, is discussed and implemented in experiments. In Chapter 4, the second idea - incorporating positional information into DBNs, is discussed with experimental results analysis as well. In Chapter 5, we propose a new DBN learning algorithm via introducing the delta rule (DR) for sentiment classification. In Chapter 6, we conclude the dissertation work and discuss some possible future work.

1.4 Contributions of the Dissertation

With the completion of this dissertation work, we make some contributions as follows:

1. We study the neuron models in DBNs for sentiment prediction, because to our knowledge there is no related work to investigate the neuron models in DBNs for sentiment analysis. So the Gaussian neuron model is introduced

into DBNs for sentiment prediction task through a variety of empirical studies on several sentiment text data sets. This neuron proves to work better than the baseline sigmoid neuron model with DBNs for sentiment analysis.

2. We propose a new word positional contribution form by scale normalization and another novel word-to-segment matrix representation for texts. They are both applied into DBNs for sentiment classification. Specifically, the two representations are incorporated into traditional bag-of-words representation for each sentence, and also demonstrate to provide positive evidence in DBNs for sentiment analysis. This result shows that there exists a more useful representation for texts in sentiment analysis area using DBNs.

3. We propose a new three-phase learning algorithm to improve the ordinary two-phase DBN training in sentiment analysis. This new algorithm is to insert another phase of the DR learning between the ordinary two phases. To our knowledge, the DR learning is originally incorporated into DBNs. Specifically, the inserted phase is to use the pre-trained RBMs to update the weights through single-layer networks, which helps to keep more correction signal in the back propagation (BP) process. Furthermore, the new algorithm has proved effective with experiments in sentiment classification task.

Chapter 2

Literature Review

2.1 Sentiment Analysis and Related Traditional Techniques

2.1.1 Sentiment Analysis

Sentiment analysis is targeted to mine people's sentiments or feelings towards entities, events and other specific topics. With the development of World Wide Web, people can express any views or ideas about almost anything in social media such as Facebook, Twitter, blogs, or internet forums. So there are plenty of online reviews and thoughts, which makes it difficult to manually analyze these opinions when we want to do product or topic review analysis. Hence, sentiment analysis has been widely studied in the NLP

community. B. Liu describes some popular topics in the sentiment analysis area [37], which includes but not limited to those below:

(1) Definition of sentiment analysis for any scientific problem. Given any scientific problem in sentiment analysis area, we need to define the problem and formalize it. Technically, the problem should be formalized to a real machine learning problem: task definition, inputs and outputs, applications.

(2) Sentiment classification. This area has been studied most in research, which treats sentiment analysis as a text classification problem. For example, it is to classify a review as *positive* or *negative* (or other sentiment labels). This is also our research area in this dissertation.

(3) Analysis for sentiment of comparative sentences. This task is not to give a *direct opinion* (e.g. *positive* or *negative*) for an object, but to give comparison opinions based on other similar objects. For example, the review "*The display quality of this laptop is amazing.*" expresses a direct opinion, while "*The display quality of this laptop is better than another.*" expresses a comparison opinion. It is obvious that identification of such kind of sentences is also valuable.

(4) Feature-based sentiment analysis. It is to determine opinions for the components in a sentence. The component also refers to attribute or feature of an object. An object can be a product, individual, organization, topic, etc. For instance, in a review regarding a laptop "*The display quality of this laptop is amazing, but the weight seems a little heavy.*", it indicates that

this object (laptop) has positive opinion on its feature "*display quality*" and negative opinion on the feature of "*weight*".

(5) Opinion search and retrieval. Currently, there are various opinions on the Web for many issues in our life. Hence, it is no surprise that people are more willing to search opinions for a specific problem. For example, given a topic keyword "*Canada Election*", we want to find positive and negative opinions about this topic from related opinion or poll sources. And then, these opinionated reviews are identified and ranked with different sentiment scores to see which candidate receives more support from the public.

In the book of *Sentiment Analysis and Opinion Mining* [38], B. Liu systematically describes and analyzes those tasks with more details. Meanwhile, in the survey of [53], B. Pang and L. Lee introduce sentiment analysis and its applications, including sentiment classification and extraction, summarization, and the related approaches. In the survey of [1], some machine learning algorithms are widely employed for sentiment analysis, such as naive Bayes (NB), maximum entropy (ME), and support vector machines (SVMs).

2.1.2 Related Traditional Techniques

Some traditional techniques are employed in sentiment analysis, including NB, ME, SVMs, language models (LMs), and other NLP methods. Andrew *et al* apply multi-variate Bernoulli and multinomial model into NB classifier

for text classification [42]. The results show that the multinomial usually performs relatively better with large vocabulary size, while the multi-variate Bernoulli only performs well with small vocabulary size.

B. Pang *et al* apply three machine learning methods: NB, ME, and SVMs to movie sentiment classification [54]. Their task considers to classify documents not by topic, but by overall sentiment, e.g., determining whether a review is positive or negative. Those results show that, better performance (much better performance for SVMs) is achieved by accounting only for feature presence, not feature frequency. In addition, B. Pang and L. Lee propose the minimum-cut framework to extract subjective portions and then examine the relation between subjectivity detection and polarity classification by using NB and SVMs polarity classifiers [52]. It shows that the subjectivity extraction is more effective input for NB classifier, compared to the originating document.

Especially, based on NB classifier, V. Raychev and P. Nakov introduce a novel language independent approach to the task of determining sentiment polarities of the author's opinion regarding a specific topic in natural language texts [58]. It intakes the positional information into NB classifier. In particular, it introduces a method to measure the positional importance, that is, instead of value one, the occurrences of the words at different positions contribute different values to their frequency values in the sentence. Then the updated frequency values are applied into NB model combined with subjec-

tivity sort or subjectivity filter. Inspired by this idea, we try to incorporate the positional information into DBNs for sentiment analysis. We not only improve the linear positional contribution mentioned above, but also propose a new word-to-segment matrix representation to represent each text to integrate the word positional information in a sentence for further sentiment investigation, which is described in detail in Chapter 4.

Language models (LMs) [56, 81], either probabilistic or non-probabilistic, have been widely accommodated in information retrieval (IR), NLP and sentiment analysis. An LM calculates a probability to a sequence of words (n-gram), but we should first suppose that the probability of a word only depends on its previous n-1 words. K. Liu *et al* propose a method to train an emoticon smoothed language model (ESLAM) based on the manually labeled data, and then use the noisy emoticon data for smoothing [40].

B. Liu *et al* raise a supervised method to extract object features [39]. For instance, in the sentence "*The display quality of this laptop is amazing.*", the object feature is "*display quality*", and the opinion on this feature is positive. Unsupervised methods are also proposed to identify object features and mine feature opinions. A set of techniques are presented in [26], such as Part-of-Speech Tagging (POS), Frequent Features Identification, Opinion Words Extraction, to provide a feature-based summary of lots of online customer reviews regarding a product. Another unsupervised information extraction system called *OPINE* is also proposed to build a model of important product

features from reviews [57]. It has proved to perform well in finding opinion phrases and their polarities.

Some lexicon-based approaches are applied to determine whether the opinions on specific features are positive, negative or neutral. X. Ding *et al* come up with a holistic lexicon-based approach to deal with context-dependent opinion words. This approach handles many special words, phrases and language structures that impact opinions based on their linguistic patterns [11]. In [26], the lexicon of WordNet is also used to identify the semantic orientations of adjectives, which are then applied to predict sentiment polarities of opinion sentences [57].

Some techniques are proposed to mine opinions from comparative sentences. A comparative sentence usually expresses an ordering relation between two or more sets of objects regarding some specific features. So some hidden opinions may also exist in comparative sentences. Based on two sequential rules of class sequential rules and label sequential rules, N. Jindal and B. Liu [28] perform two techniques of NB and conditional random fields (CRF) to identify comparative sentences and extract comparative relations. In [13], it is devoted to mining opinions from comparative sentences. It proposes a new measure called *one-side association* (OSA), to measure the association between the comparative word and the object feature. This measure is then accommodated to identify the preferred objects of a customer.

2.2 Deep Learning

2.2.1 Representation Learning

The success of machine learning algorithms generally relies on data representation, and it is mostly due to the fact that different representations can entangle and hide more or less explanatory information behind the data [3]. Deep learning methods aim at learning feature hierarchies with higher level features formed by the composition of lower level features. The tutorial in [34] explains why we need deep architectures [31]. It is because deep architectures can be representationally efficient with fewer computational units for same function, and might allow for feature hierarchy. Through automatically learning features at multiple levels of abstraction, a deep learning system can learn complex mapping functions directly from the input (raw data) to the output, without any human-crafted features.

Figure 2.1 depicts a general diagram for data processing. In this figure, a data modeling process is divided into five steps in order: raw data, pre-processing, feature extraction, feature selection, and model inference or application. Especially, the process from pre-processing to feature selection is called *representation learning* (or *feature learning*), which is the most critical for the model accuracy. However, representation learning process is often hand-crafted in practice before deep learning methods are effectively employed, such as NB, SVMs or decision trees. It is mostly time-consuming in

development cycle and also takes much computation time for testing. Hence, the rapidly developing deep learning methods have proved more efficient and more effective for representation learning [3].

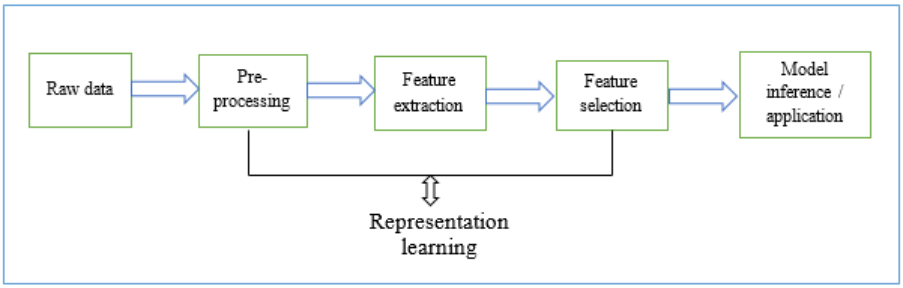

Figure 2.1: Diagram of data processing.

Deep learning actually means learning multi-level feature representations from huge amount of data, which makes deep learning a very popular method that is very different from other state-of-the-art algorithms. It is deep if it has more than one stage of non-linear feature transformation as the diagram in Figure 2.2 [30].

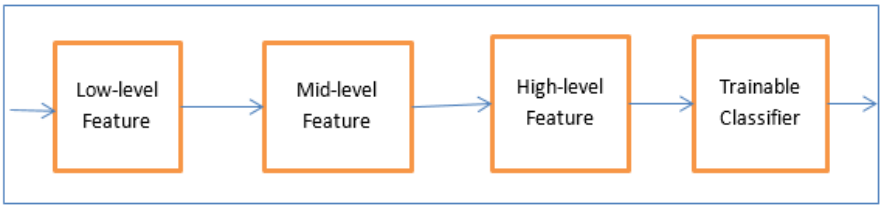

Figure 2.2: Diagram of learning hierarchical representations.

It can be seen from Figure 2.2 that hierarchical representations are increasing levels of feature abstraction, and each phase is a type of trainable feature transformation. Several examples are illustrated below [30]:

(1) For image recognition problem, we have the following representations transformation process: Pixel → edge → texton → motif → part → object. (Note: each raw input of image is denoted by pixels.)

(2) For natural language processing problem, the transformation process from raw input (character) to last selected features (story) is: Character → word → word group → clause → sentence→ story.

(3) For speech recognition problem, the feature representation transformation is: Sample → spectral band → sound → $\cdots$ → phone → phoneme → word.

That is the brief introduction for the feature representation hierarchy in deep learning. In practice, deep models use deep architectures to learn the different level feature representations.

2.2.2 Deep Belief Network

Deep belief network (DBN) [22] is composed of multiple layers of stochastic, hidden variables [4], and is structurally based on restricted Boltzmann machine (RBM) as depicted in Figure 1.2. This figure illustrates that each layer from bottom to up actually represents a relatively higher feature abstraction. Therefore, the training diagram from visible layer to the top hidden layer (Hidden layer 3 in Figure 1.2) is a representation learning process.

Compared to deep neural networks, DBNs have a phase of unsupervised pre-training [5], and have proved more effective with a set of experiments. In

detail, deep neural networks trained with the BP algorithm without unsupervised pre-training perform even worse than shallow networks. There are some problems with BP training algorithm [5, 34]: (1) The gradient for BP is progressively getting more dilute since the correction signal is minimal below the top few layers; (2) The BP algorithm for deep networks will become trapped in local minima especially because they initialize weights far from "good" regions (i.e., random initialization); and (3) BP is a supervised training method that normally only works for labeled data, but the data in reality is mostly unlabeled. These problems prevent BP from working very well for deep architectures. However, the DBNs with unsupervised training phase actually work relatively better. There is a phase of unsupervised learning – greedy layer-wise training for stacks of RBMs employed in DBNs [5], which allows to develop hierarchical abstractions from bottom to up and helps the BP phase of the network start with good parameters. So RBM is an important component in a DBN since it distinguishes a DBN from an ordinary deep neural network.

Theoretically, RBM is a variant of Boltzmann machine (BM). BM is a particular form of log-linear Markov Random Field (MRF), and there are undirected relationships between each unit of a BM no matter the unit is hidden or visible. However, the computation of the gradient to train BMs is very expensive, and training time is very long [4]. Hence, the connectivity of a general BM is restricted to make learning easier, with the restriction that their neurons must form a bipartite graph (restricted Boltzmann machine,

RBM): they have visible units, corresponding to features of their inputs, hidden units that are trained, and each connection in the RBM must connect a visible unit to a hidden unit, which is depicted in Figure 2.3. This restriction allows for more efficient training algorithms that are available for the general class of BMs, particularly the gradient-based contrastive divergence (CD) learning algorithm [7].

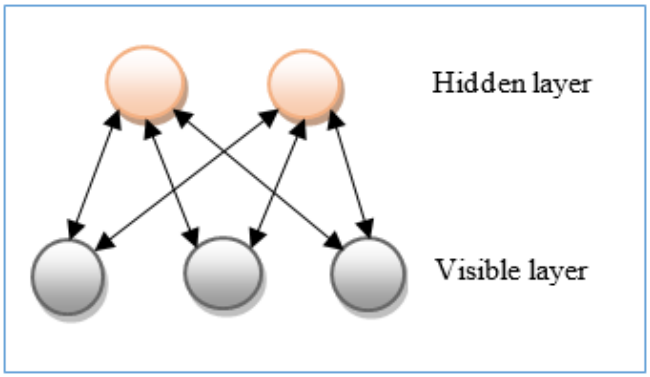

Figure 2.3: A graphical representation example for RBM.

One key point is that, the hidden units in RBMs are conditionally independent given the visible states. Specifically, referring to the text in [4], the energy function for the RBM configuration is given as Eq. 2.1.

$$Energy(x, h) = -b^T x - c^T h - h^T w x, \tag{2.1}$$

where x represents the vector of visible units, h is hidden units vector, h^T is the transposition of h, w represents the weights connecting hidden and visible units, and b, c are the bias terms of the visible and hidden layers

respectively.

Based on the energy function, the joint probability distribution over all the visible and hidden units $P(x, h)$, and the marginal distribution $P(x)$ over the visible units are defined in the following two formulas:

$$P(x, h) = \frac{1}{Z} e^{-Energy(x,h)}, \tag{2.2}$$

$$P(x) = \frac{1}{Z} \sum\nolimits_{h} e^{-Energy(x,h)}, \tag{2.3}$$

where Z is a partition function defined as the sum of $e^{-Energy(x,h)}$ over all possible visible and hidden units.

To train RBMs, the Gibbs sampling plays a significant role in calculating $P(h|x)$ and $P(x|h)$ [4]. Gibbs sampling is a Markov chain Monte Carlo (MCMC) technique to sample a sequence when direct sampling is hard to perform. Since it is hard to estimate the joint distribution $P(x, h)$, with Gibbs sampling in a RBM, the joint variables (x, h) can be sampled in the Gibbs chain in two sub-steps: (1) sample h given x, and (2) sample new x given h. The sampling algorithm in RBMs is described in Chapter 3. Without loss of generality, suppose binary units {0,1} are used, then the probabilistic functions for each hidden unit and visible unit are given by the following formulas Eq. 2.4 and Eq. 2.5.

$$P(h_j|x) = \varphi(b_j + w_{\cdot j} x), \tag{2.4}$$

$$P(x_i|h) = \varphi(c_i + w'_{i\cdot}h), \tag{2.5}$$

where φ is activation function (e.g. sigmoid function), w and w' are two different weight matrices. Also, $w_{\cdot j}$ denotes the vector of weights connected from each visible unit to the j^{th} hidden unit, while $w'_{i\cdot}$ represents the weight vector connecting each hidden unit to the i^{th} visible unit. The b_j and c_i are corresponding bias terms.

The RBM is to learn a marginal probability distribution over its input, by maximizing the log probability over a training set X (input matrix, each row of X denotes an example x), namely,

$$\arg\max_w \sum\nolimits_{x \in X} \log P(x),$$

where $P(x)$ represents marginal probability over all the visible features, and w is weight matrix connecting the units of the two layers [4]. The goal is the same as minimizing the loss function – negative log-likelihood function, which is similar as: $\arg\min_w[-\sum_{x \in X} \log P(x)]$.

Gibbs sampling is applied to generate samples to model the distribution $P(x)$ over visible features, and theoretically the chain of samples can converge after enough number of sampling steps [4]. However, an RBM is trained using Contrastive Divergence (CD) in practice [23, 80], which is an approximate maximum-likelihood learning approach. CD does not wait for the samples chain to converge. Normally samples are generated after only k steps of

Gibbs sampling, which is CD-k, and $k = 1$ has been shown to work surprisingly well [7]. The training of RBMs provides a better weight initialization for BP process, which distinguishes DBNs from ordinary neural networks where the initial weights are randomly generated. Normally, learning DBN is divided into two phases [22]: unsupervised RBM training and supervised neural network training (BP training). The first phase comes from the principle of greedy layer-wise unsupervised training that a DBN is trained with RBMs as the building blocks for each layer [22, 5], and each RBM is trained following the practical guide written by G. Hinton [21]. Specifically, every two adjacent layers in Figure 1.2 helps to form stacks of RBMs for unsupervised training. The second phase is actually the traditional BP algorithm, a common method to train artificial neural networks, introduced by D.E. Rumelhart *et al* [60].

2.2.3 Related Work for DBNs

1. Some work is to consider the influence of neuron models on the deep learning system. V. Nair and G. Hinton use noisy rectified linear units to approximate the stepped sigmoid units [47], but it has only proved better on face verification data. X. Glorot *et al* propose rectifying neuron: $max(0, x)$ to create sparse representations with true zeros [17], which is suitable for naturally sparse data, but is unfortunately non-differentiable at zero. Motivated by these work, assuming that different neuron models will play different ef-

fects for a specific application problem, the effect of neuron model in DBNs is investigated in this dissertation for sentiment prediction.

2. Some useful techniques are presented in order to avoid over-fitting in machine learning especially for neural networks training, such as early stopping, the L1 and L2 regularization for weight penalties [6, 79], and soft weight sharing [49]. Except for these general techniques, there are also some methods proposed to avoid over-fitting in deep neural networks. *Dropout* technique deals with over-fitting in large networks by randomly dropping units (together with their connections) during the network training process [68]. In [75], a generalization of DropOut, called *DropConnect*, is proposed to regularize neural networks with large fully-connected layers. Unlike randomly selecting a subset of activation units to be zeros within each layer for DropOut, DropConnect instead assigns a randomly selected subset of weights within the network to zeros. M. Cogswell *et al* [9] propose a new regularizer called *DeCov* to reduce over-fitting and obtain better generalization. This regularizer minimizes cross-covariance of hidden units in deep neural networks, trying to obtain diverse or non-redundant feature representations.

3. There are also some studies with the learning process of DBNs. P. Lin *et al* present a ME learning method with DBNs [36], which maximizes only the entropy of parameters in DBNs, allowing more effective generalization ability. In RBMs, instead of doing negative sampling from the model distribution to approximate the negative gradient [23], A. Yuille proposes a way to initialize

a Markov chain at a state when the previous model is ended [80], so this initialization is considered fairly close to the model distribution. R. Neal also uses this method with sigmoid belief networks to approximately sample from the posterior distribution of hidden layer units given the visible layer units [48]. Meanwhile, different from the standard CD, a new algorithm called *Persistent Contrastive Divergence* (PCD), is proposed by T. Tieleman to train RBM based on the above approach of A. Yuille, which is to draw samples from almost exactly the model distribution [72].

In view of this point, we also take advantage of the previous RBM training during the whole training process. However, we do not use it to approximate the model distribution, but use the previous trained RBM to generate hidden units that can form a single-layer network together with the visible layer of RBM. Considering the traditional BP learning for neural networks is to minimize the error between the actual output and target output in a back ward fashion, the correction signal of each layer will get dilute as the number of layers in the network increases. Hence, we apply the delta rule (DR) [69] learning into the single-layer network after RBM training, trying to not lose much correction signal in total. The weights trained from DR for each single-layer network are then used as the initialization for whole BP training.

4. Currently, there is not much work about the DBNs for NLP as well as sentiment analysis. Some applications of DBNs regarding various problems for natural language understanding are studied in [62]. The method is to learn

new features from unlabeled data through a multi-layer generative model. And then the newly learned features are adopted to train an ordinary feed-forward neural network. S. Zhou *et al* propose hybrid DBNs to address the semi-supervised sentiment classification problem [82]. P. Ruangkanokmas *et al* also develop a semi-supervised learning algorithm, called *Deep Belief Networks with Feature Selection* (DBNFS) for sentiment classification [59].

2.2.4 Other Deep Learning Models for NLP and Sentiment Analysis

Deep learning models have been widely used in image, audio, video, and NLP problems, and also have achieved great success. Deep learning indeed represents a set of machine learning algorithms with deep architectures. It not only includes the above introduced DBN, but also includes some other popular models, such as auto-encoder, convolutional neural network (CNN), recurrent neural network (RNN), and recursive neural network. They are introduced as below.

1. Auto-Encoder

The auto-encoder [4], also called auto-associator, aims at learning a more efficient representation from raw input, typically with the purpose of dimension deduction. Specifically, it is to encode the input $\mathbf{x}$ into some representation $\mathbf{c}(\mathbf{x})$, so that we can reconstruct $\mathbf{x}$ from $\mathbf{c}(\mathbf{x})$. In other words, the auto-

encoder is actually to encode the input itself. The reconstruction error for training an auto-encoder is generally described in Eq. 2.6.

$$Re\_err = -logP(\mathbf{x}|\mathbf{c}(\mathbf{x})). \tag{2.6}$$

A deep auto-encoder structure is stacked with multiple encoders, which is called *stacked auto-encoders* [24]. To discover more robust features with auto-encoder and avoid the auto-encoder to simply learn the identical features from input, denoising auto-encoder is proposed to force the learned representations more robust to small irrelevant changes in input pattern [74]. A hierarchical neural auto-encoder is proposed for natural language generation of coherent long texts in [35].

2. Convolutional neural network

A convolutional neural network (CNN) [4] is inspired by the animal visual system structure, which uses a convolution operation to mathematically describe the overlapping regions tiling the visual field [77]. CNNs have the characteristics of shared weights and translation invariance. A CNN structure usually consists of several layers introduced below [77]. (1) Convolutional layer: In a convolutional layer, it often assumes each neuron is connected to only a small subset of the input (local connectivity), so that the parameter sharing scheme is used to mathematically control the number of free parameters. (2) Pooling layer: Pooling is another important concept of CNNs following the

convolutional layer. It normally uses a way of non-linear down-sampling to reduce the number of parameters in the network to save computation time, which is also to control over-fitting. (3) Fully-connected layer: This layer is constructed after several convolutional layers and pooling layers to perform high-level inference through the ordinary neural networks trained with BP algorithm. (4) Output layer: Normally it is to use softmax function to predict the class labels.

Y. LeCun *et al* have designed convolutional neural networks (CNNs) trained with the error gradient, to obtain great performance for pattern recognition tasks [33, 32]. LeCun's CNN is normally named as LeNet-5, which is a typical 7-level convolutional network. An example LeNet-5 is depicted in Figure 2.4. Gradient-based learning algorithm is applied to train LeNet-5 for document recognition.

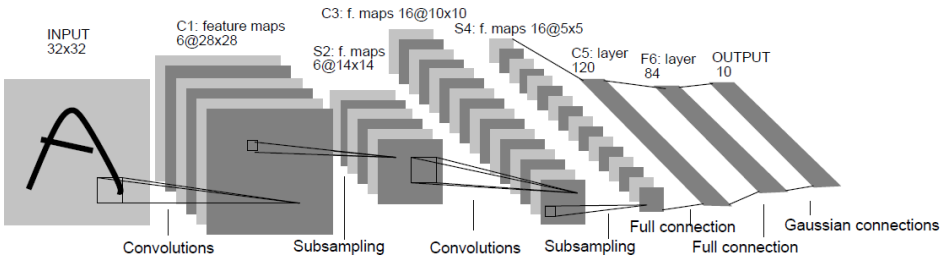

Figure 2.4: A CNN of LeNet-5 for digits recognition, each plane is a feature map ([33]).

Besides, R. Collobert and J. Weston describe a unified CNN structure for NLP tasks [10], such as Part-Of-Speech Tagging (POS), Chunking, Named Entity Recognition (NER), Semantic Role Labeling (SRL). For NLP prob-

lems, the input sentence is processed by several feature extraction layers. The BP algorithm is applied to automatically learn the features of deep layers within the network. Figure 2.5 depicts a general deep neural network architecture for NLP.

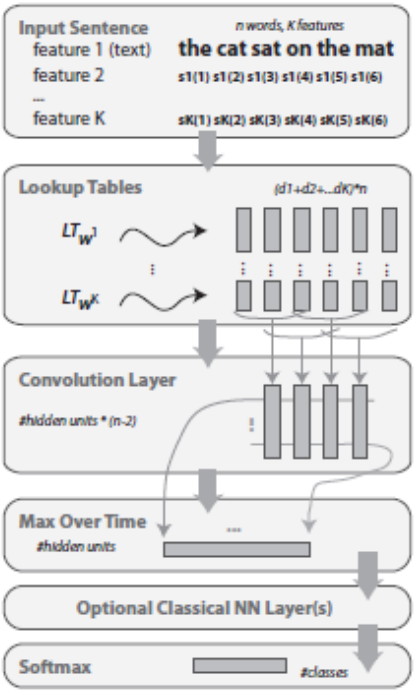

Figure 2.5: A general CNN structure for NLP ([10]).

3. Recurrent neural network

A recurrent neural network (RNN) denotes a type of artificial neural network in which the connections between units can form a directed cycle. T. Mikolov *et al* present a new recurrent neural network based language model (RNN LM) with applications to speech recognition [43, 45]. A simple RNN is shown in Figure 2.6, which is also called *Elman* network [12]. For this RNN, at time

t, it has an input layer $x(t)$, output layer $y(t)$, and hidden layer $c(t)$ (also called context layer). As depicted in the figure, the current context $c(t)$ is computed from the current input $x(t)$ and its previous context $c(t-1)$, which is generally introduced in Eq. 2.7.

$$c(t) = \varphi(Ux(t) + Wc(t-1)), \tag{2.7}$$

where t is time point, and φ is activation function, U and W is weight matrices for $x(t)$ and $c(t-1)$ respectively.

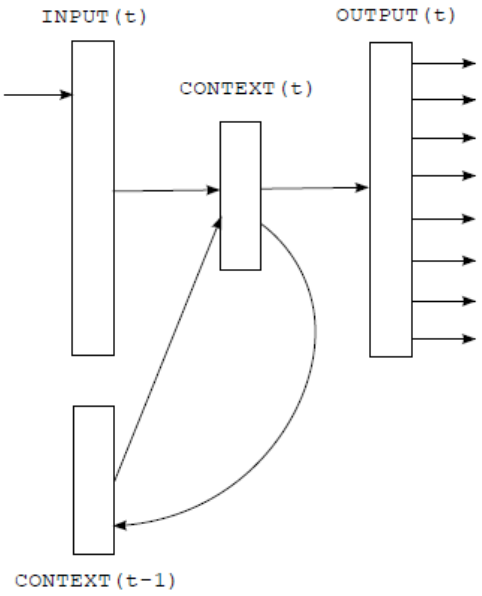

Figure 2.6: A simple recurrent neural network ([43]).

RNNs have proved effective because they can use the internal memory to deal with arbitrary sequences of inputs theoretically. Nowadays, many researchers employ a RNN model called *Long short-term memory* (LSTM) network de-

scribed in [25]. Unlike other RNNs, LSTM can avoid back-propagated errors to vanish or explode, which is usually improved by recurrent gates called forget gates [14]. Some other LSTM applications for NLP are introduced in [76, 41, 15, 73].

4. Recursive neural network

A recursive neural network is to apply the same set of weights recursively through a recursive structure [19]. A simple recursive neural network is depicted in Figure 2.7. In this figure, suppose that c_1 and c_2 are N-dimensional vectors to represent the nodes, their parent node p will be an N-dimensional vector as well calculated through Eq. 2.8 and the local score s is calculated via Eq. 2.9 [66].

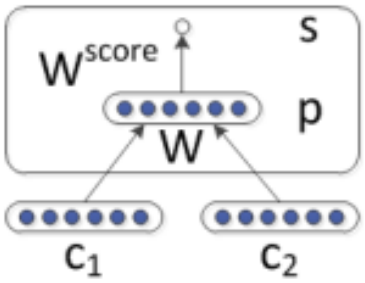

Figure 2.7: A simple recursive neural network that is replicated for each pair of possible input vectors ([66]).

$$p = \varphi(W[c_1; c_2] + b), \tag{2.8}$$

$$s = W^{score} p, \tag{2.9}$$

where W is an $N * 2N$ weight matrix, s represents the local score that is used to calculate the final score of the structure, W^{score} is an N-dimensional row vector for weights corresponding to the node p.

Various recursive models are applied in NLP studies. Based on the word-vector representations (or word embeddings) that assume each word can be represented by an N-dimensional vector [44], R. Socher *et al* incorporate word vectors to replace the traditional word count or presence features, and then they integrate word vectors into multiple recursive modes such as recursive auto-encoders [63], recursive matrix-vector model [65], and recursive deep network [67], which have obtained effective results but require a very large labeled tree bank for training. R. Socher also systematically describes the recursive deep learning for NLP studies in [64]. R. Paulus *et al* apply neural word vectors into global belief recursive neural networks for sentiment classification [55].

5. In addition to the above models, there are also some other related work. Some researchers study new deep learning structures with specific applications. D. Tang *et al* integrate sentiment-specific word embedding features into neural networks and develop a neural network with a hybrid loss function to the deep learning system [71]. X. Glorot *et al* adopt domain adaptation into sentiment classifiers in which a system is trained on labeled reviews of one source domain but would also be employed on another [16].

Some people consider the word order information into several different NLP

tasks. T. Pahikkala *et al* introduce a framework based on a word-position matrix representation of text for natural language disambiguation tasks [51]. Specifically, in disambiguation tasks, each input example is consisting of a word to be disambiguated and its surrounding context words, then a kernel function is applied to map the features. R. Johnson and T. Zhang propose an effective way of bag-of-words conversion into convolutional layer to exploit the word order of text for text categorization using convolutional neural networks (CNNs) [29]. In detail, the region representation called *seq-CNN* is to embed text regions into low dimensional vector space. Meanwhile, considering the model becomes too complex and the training is too expensive, they provide an alternative way *bow-CNN* to perform bag-of-words conversion to make region vectors. For example, suppose the vocabulary is {"*do*", "*I*", "*you*", "*love*", "*hate*"}, given a sentence "*I love you*", then two convolutional phrases "*I love*" and "*love you*" in the sentence can be represented by the converted bag-of-words vectors as $[0, 1, 0, 1, 0]^T$ and $[0, 0, 1, 1, 0]^T$. In this way, the word order in a local region "*I love*" or "*love you*" is lost while the order of the two phrases is indicated. Inspired by this concept of representation, we propose the word-to-segment matrix to represent a sentence, in which each sentence is divided into several segments (not convolutional). We do not consider the word order within the segment, but the segment order is represented using the word-to-segment matrix.

Chapter 3

Gaussian Neuron in Deep Belief Networks for Sentiment Prediction

3.1 Gaussian Neuron in DBN

As introduced previously, the neuron model is a significant component in a DBN, because the conditional probabilities of each hidden unit and visible unit, $P(h|x)$ and $P(x|h)$, are given by Eq. 2.4 and Eq. 2.5. It indicates that the activation function calculates the probabilities. Except the widely used sigmoid neuron function, some other neurons are also used in deep networks, such as the rectified linear units that have only proved efficient on

image-related tasks unfortunately [47]. But to our knowledge, there is no related study on the neuron models in DBNs for sentiment analysis. Hence, in this chapter, we investigate different neuron models in DBNs for sentiment analysis. Especially, we introduce the Gaussian neuron into DBNs for the sentiment prediction task.

3.1.1 Gaussian Neuron

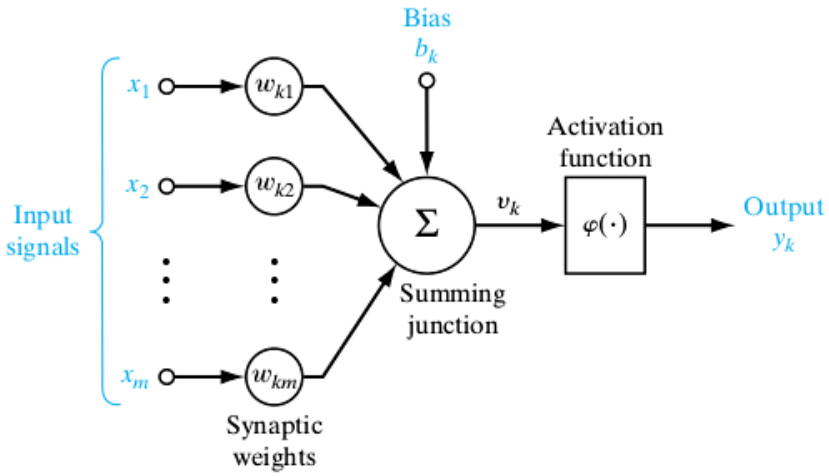

Figure 3.1: A single neuron diagram ([8]).

The neuron flow diagram works like Figure 3.1, which shows the working mechanism of a single neuron in a neural network. In detail, for the k^{th} neuron in a neural network, all the inputs from x_1 to x_m are fed into a summing function with corresponding weights of w_{k1} through w_{km} plus a bias term b_k. The summing result v_k in Eq. 3.1 goes through the activation function φ to obtain output y_k in Eq. 3.2.

$$v_k = b_k + x_1 w_{k1} + x_2 w_{k2} + ... + x_m w_{km}, \tag{3.1}$$

$$y_k = \varphi(v_k). \tag{3.2}$$

Generally, the sigmoid neuron function in Eq. 3.3 with respect to the independent variable t is widely used in neural networks.

$$\varphi(t) = sigm(t) = \frac{1}{1 + e^{-t}}. \tag{3.3}$$

The Eq. 3.3 indicates that the sigmoid value will be greater than 0.5 when t is greater than 0, which normally means the neuron is in active status. On the other hand, considering the activation function is actually to compute a probability, it naturally inspires us to come up with some other feasible probability functions. Especially in sentiment analysis, the neuron activation function needs further investigation, because the t value in an activation function comes from a linear combination of the previous layer's units, which indeed represents a sequence of words (visible layer), or a sequence of phrases (hidden layers). Hence, how this combination contributes to the sentiment of the next layer, probably has different effect curves (not only sigmoid, or tangent hyperbolic functions). Here, the Gaussian neuron function written in Eq. 3.4 is investigated for sentiment prediction.

$$\varphi(t) = Gaussian(t; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{t} e^{-(z-\mu)^2/2\sigma^2} dz, \tag{3.4}$$

where t is independent variable, and φ is actually the cumulative distribution function of Gaussian density with mean and variance (μ, σ^2). This neuron has two advantages: (1) it gives diversified activation function curves just by setting different values of (μ, σ^2); and (2), in the learning process, the derivatives for gradient descent in BP training can be easily computed just via its density function.

Therefore, the DBNs with Gaussian neuron and sigmoid neuron respectively will be implemented on several sentiment data sets including balanced and imbalanced for performance comparison.

3.1.2 Learning Approach

In the DBNs for sentiment prediction, there are some function options for the output layer, such as softmax, sigmoid and Gaussian cumulative function. However, the sigmoid function proves more effective in both neuron models in this research, which is written as Eq. 3.5.

$$Label^S = sigm(w^{out} \cdot S + b^{out}), \tag{3.5}$$

where w^{out} is a weight matrix connecting the output layer and its previous hidden layer, b^{out} is the corresponding bias vector, and S represents the

"Sentence" of the penultimate layer, and $Label^S$ is finally calculated as a C-vector (C is the number of classes) in which the largest value indicates its class.

Learning the DBN is divided into two phases: unsupervised RBM training and supervised neural network training. The first phase of RBM is trained via CD algorithm following the practical guide by G. Hinton [21], and the second one is actually adopting the common BP method to train artificial neural networks, introduced by D.E. Rumelhart *et al* [60]. Below is the description of the learning algorithm:

Input: a set X of training examples, and corresponding manually labeled class Y_X

Output: a DBN classifier

Note: x is an example in X which denotes a sentence; y_x, belonging to Y_X, is the class vector of x.

Phase I: Unsupervised RBM training

- Take a training example x, compute the conditional probabilities of hidden units using Eq. 2.4 and sample a hidden activation vector h from this distribution;
- Given h, compute the conditional probabilities of visible units through Eq. 2.5, and sample from this distribution to reconstruct the visible units x_{new};

- Given x_{new}, resample a hidden activation h_{new} by computing conditional probabilities through Eq. 2.4;
- For some learning rate α, update the weight matrix and bias term via Eq. 3.6 and Eq. 3.7:

$$w \leftarrow w + \alpha(xh^T - x_{new}h_{new}^T), \tag{3.6}$$

$$b \leftarrow b + \alpha(x - x_{new}). \tag{3.7}$$

Phase II: Supervised neural network training

- Obtain the weight matrix w and bias term b from the first phase as the initial weights in the traditional neural network;
- Implement forward propagation to compute the cost function (average sum-of-squares error) of the traditional neural network;
- For each layer, implement BP process to compute partial derivatives;
- Fine-tune the weight matrix w and bias term b using the partial derivatives as gradient descent until convergence.

A graph example to train DBNs is shown in Figure 3.2 corresponding to the above learning algorithm. In this figure, the training process clearly consists of two parts: the interior blue rectangle (P1: Phase I) and the external green circle (P2: Phase II). Especially when it refers to the derivative of Gaussian cumulative distribution function, the Gaussian density function

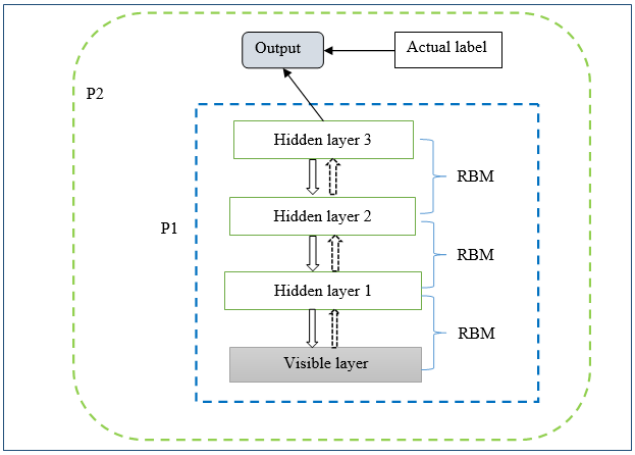

Figure 3.2: A graph example to train DBNs.

can be easily calculated. Hence, the Gaussian neuron in DBN has similar time complexity as sigmoid neuron, and therefore we need only pay attention to their classification performances.

3.1.3 Evaluation

In this chapter, two main evaluation metrics are employed to compare the performances: total accuracy (TA) and F-measure (F_1 score) [50]. Firstly, the predictive accuracy TA is typically used for evaluating the performance of machine learning algorithms. Besides, for the purpose to evaluate the performance regarding the imbalanced data, F_1 score is used to combine precision and recall into evaluation together. Suppose in the binary classification task, the four variables, tp, tn, fp, and fn, represent the true positives, true negatives, false positives, and false negatives respectively. Then the two measures

are calculated by Eq. 3.8 and Eq. 3.9.

$$TA = \frac{tp + tn}{tp + tn + fp + fn}, \tag{3.8}$$

$$F_1 = \frac{2tp}{2tp + fp + fn}. \tag{3.9}$$

3.2 Experiments and Results

To demonstrate the effect of neuron models in DBNs, we design experiments as below and achieve corresponding results through a range of implementations.

3.2.1 Data Collection and Pre-processing

This chapter focuses on Twitter sentiment prediction. The labeled twitter corpus we use is the Stanford Twitter Sentiment (STS) [18], which includes 1.6 million training examples labeled as positive and negative automatically using the emoticons of the tweets. In order to construct some independent data sets for more comparisons and also to speed up our training time, we randomly sample ten non-overlapped small data sets from this training set of STS.

Each raw data set needs to be pre-processed for training the model. Firstly, all characters are converted to lowercase since upper case and lower case have

no differences for sentiment polarities; Secondly, the URLs in the data set are removed, because they do not make much sense for the sentiment; Thirdly, we transform some acronyms and abbreviations to their completely expanded forms. For example, "*i've*" is replaced by "*i have*", "*can't*" or "*cannt*" is "*can not*", "*won't*" or "*wont*" is "*will not*", "*shouldn't*" or "*shouldnt*" is "*should not*", with details in Table 3.1. In this way some meaningful words especially the negation word "*not*" are kept as they are essential for sentiment. Finally, some punctuations, such as @, /, |, $, are deleted as well since they contribute little to the text sentiment.

After the above pre-processing, we need to obtain initial trainable data that can be directly used in DBNs. Since Twitter texts are all limited to 140 characters long, each word in one Twitter text occurs only once for most of the time, so the vocabulary of each data set is extracted as attributes, each word token is denoted by its presence or not, and then each sentence (training example) can be represented by a vector where the element is one if the word exists in it, otherwise zero (filtered by the *StringToWord* in Weka 3.6.12 [20]). Besides, the sentence's label is denoted by a vector with value one at corresponding position and zeros elsewhere. For example, here is an example with positive label in a binary classification task, then its label vector (output) can be defined as (1, 0) in which the first element denotes positive while the second element is negative.

The randomly sampled ten non-overlapped data sets, including four balanced

Table 3.1: Corresponding expanded forms for abbreviations.

abbrev.	expanded forms	abbrev.	expanded forms
i've	i have	won't/wont	will not
i'm	i am	wouldn't/wouldnt	would not
i've	i have	shouldn't/shouldnt	should not
i'll	i will	can't/cannt	can not
it's	it is	couldn't/couldnt	could not
let's	let us	isn't/isnt	is not
she's	she is	wasn't/wasnt	was not
he's	he is	aren't/arent	are not
she'll	she will	weren't/werent	were not
he'll	he will	don't/dont	do not
you've	you have	doesn't/doesnt	does not
there're	there are	didn't/didnt	did not
there's	there is	haven't/havent	have not

data sets and six imbalanced data sets, are described in Table 3.2. Here 'attributes' means the number of word tokens for each data set, IR is the imbalanced ratio which equals the ratio of majority over minority (balanced for $IR = 1.0$, otherwise imbalanced).

Table 3.2: Description of data sets used in experiments.

Data set	size	attributes	classes	positive	negative	IR
pos500neg500	1000	1015	2	500	500	1.00
pos500neg5000	5500	1060	2	500	5000	10.00
pos1000neg1000	2000	1107	2	1000	1000	1.00
pos3000neg3000	6000	1016	2	3000	3000	1.00
pos3000neg10000	13000	995	2	3000	10000	3.33
pos3000neg20000	23000	988	2	3000	20000	6.67
pos5000neg500	5500	1033	2	5000	500	10.00
pos5000neg5000	10000	990	2	5000	5000	1.00
pos10000neg3000	13000	988	2	10000	3000	3.33
pos20000neg3000	23000	978	2	20000	3000	6.67

3.2.2 Experimental Results

In our experiments, to demonstrate that the Gaussian neuron in DBN performs better in Twitter sentiment prediction, the widely used sigmoid neuron is implemented for baseline. So we perform two DBNs in this chapter, DBN-Gaussian and DBN-sigmoid. Different DBN architectures have been tested based on a variety of experiments, including the number of hidden layers (1, 2, 3) and hidden units (100, 200, 300, 400, 500), the functions for the output layer (Gaussian, sigmoid, softmax), and the settings of (μ, σ) with combinations of (μ = 0, 0.1, 0.2, 0.5, 0.8, 1, 2; σ = 0.1, 0.5, 1, 1.5, 2.0, 3, 4, 5). Finally, we manually set the system as: input (visible units) $\rightarrow$ 200 hidden units $\rightarrow$ 100 hidden units $\rightarrow$ output layer (class labels), and the parameter setting of ($\mu = 0.8, \sigma^2 = 3^2$) is selected for DBN-Gaussian. The implementations are performed on the platform of MATLAB 2014a in the PC of 64-bit OS, Intel Core i5-5200U, CPU 2.20GHz, and RAM 8.0GB (Same platform in the whole dissertation).

Each data set is performed using ten-fold cross validation (nine for training and one for testing) to obtain an average accuracy. Firstly, in RBM unsupervised training, the momentum is 0.1 and learning rate is 0.9. According to the reconstruction error plot in Figure 3.3, it has decreased to a stable level and then usually 50 epochs would be enough for Twitter sentiment prediction in DBNs, which would also help save computation time. Besides, for supervised BP training, the maximum number of epochs is 1000 with 10^{-6}

as the convergence control based on early stopping rule. Finally, a statistical t-test is accommodated to test the difference between the results of two approaches on each data set.

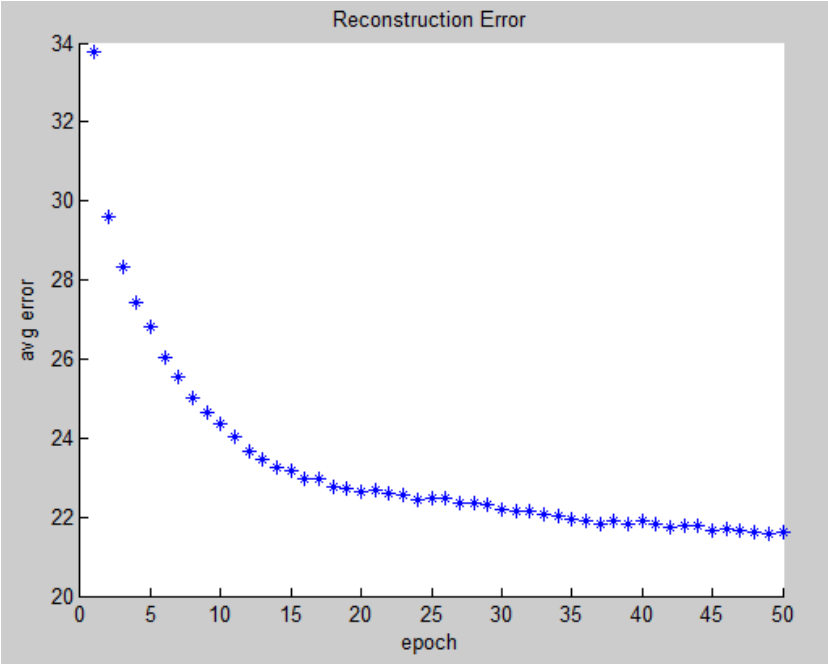

Figure 3.3: Reconstruction error for one time performance.

Table 3.3 displays total accuracy TA and F_1 score of the two neuron DBN models on each data set, and each cell of TA is represented by the mean and standard deviation of the ten-fold cross validation results, and F_1 score is the average value. Based on the comparisons of sigmoid and Gaussian neuron in DBN, the ordinary accuracy TA and F_1 score do make some differences among these data sets. Here we assume that the accuracy has won (lost) if it passes a two-tailed t-test within 95% confidence level. (Note: even if some F_1 numbers of DBN-Gaussian are not with *, but they are still higher than

those of DBN-sigmoid.)

Table 3.3: Experimental results on classification accuracy (%).

Data set	DBN-Gaussian		DBN-sigmoid	
	TA	F_1	TA	F_1
pos500neg500	69.10±4.10 *	0.670	67.20±4.00	0.668
pos500neg5000	89.60±1.42	0.253 *	90.96±1.15 *	0.101
pos1000neg1000	66.90±2.97	0.657	68.00±3.06	0.682 *
pos3000neg3000	71.32±1.84 *	0.719 *	70.05±2.04	0.705
pos3000neg10000	80.72±1.26 *	0.533 *	78.34±0.94	0.511
pos3000neg20000	88.02±0.76 *	0.419 *	86.49±1.24	0.395
pos5000neg500	88.20±2.08	0.936	90.87±0.98 *	0.951 *
pos5000neg5000	72.29±1.77	0.726	71.21±1.87	0.717
pos10000neg3000	80.68±0.81 *	0.879 *	78.46±1.34	0.863
pos20000neg3000	87.70±0.72 *	0.932 *	86.10±0.93	0.921
average	**79.45**±1.77	0.672	78.77±1.76	0.651

In Table 3.4, an explicit comparison between Gaussian DBN and sigmoid DBN is summarized, in which $w/l/t$ means the method wins in w data sets, loses in l data sets, and ties in t data sets. For example, 6/2/2 under TA column of all data sets means that the Gaussian DBN has 6 wins, 2 losses, and 2 ties on TA compared to sigmoid DBN. In all, the results are summarized briefly that, (1) Gaussian DBN has relatively higher TA and F_1 score than sigmoid DBN for all data sets; (2) Gaussian DBN performs better on TA but ties in F_1 score comparison for balanced data sets; and (3) Gaussian DBN works much better than sigmoid DBN with respect to both TA and F_1 score for imbalanced data sets.

Table 3.4: Summary of evaluation comparisons.

Model	all data sets (10)		balanced data sets (4)		imbalanced data sets (6)	
	TA	F_1 score	TA	F_1 score	TA	F_1 score
DBN-Gaussian	6/2/2	6/2/2	2/0/2	1/1/2	4/2/0	5/1/0
DBN-sigmoid	2/6/2	2/6/2	0/2/2	1/1/2	2/4/0	1/5/0

3.2.3 Analysis and Verification

Through the results introduced above, we try to analyze the difference between the two neuron models. Mathematically, the curves of the two neuron functions are plotted in Figure 3.4. Here we suppose that Gaussian neuron is the Gaussian cumulative function with mean 0.8 and variance 3^2.

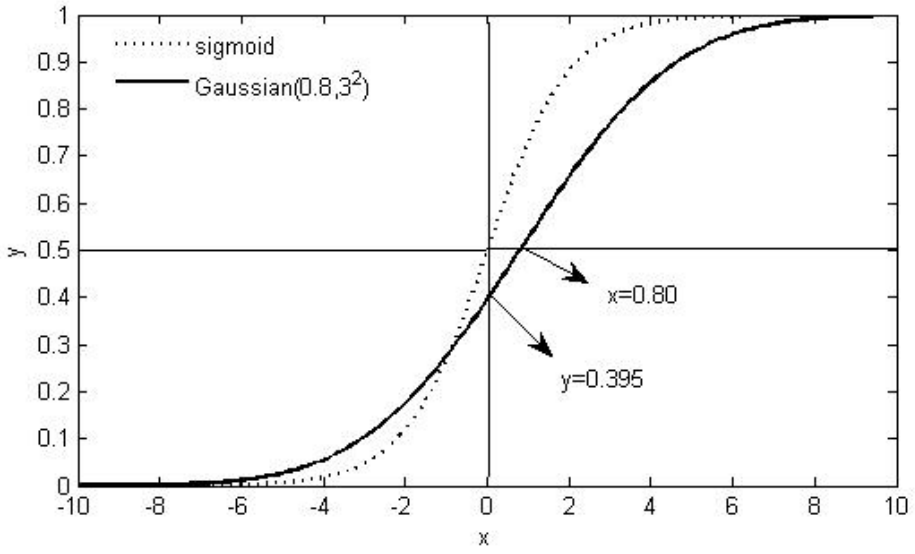

Figure 3.4: Curves for Gaussian and sigmoid functions.

Figure 3.4 shows the curves for sigmoid (dash line) and Gaussian cumulative distribution (solid line) functions. The Gaussian activation function indicates

the probability is greater than 0.5 (the neuron is active) when $x > 0.8$ (μ), not $x > 0$ as sigmoid function. When $x = 0$, the Gaussian function value is only 0.395 (the neuron is inactive). Hence, compared to sigmoid DBN, it gives some indications for Gaussian DBN regarding Twitter sentiment prediction: (1) the mean μ of Gaussian DBN should be greater than zero, but not too big; and (2), the slope (derivative) of the activation function should be relatively smoother, which means the standard deviation σ should be relatively larger. These two indications can provide us some useful guidance towards the activation function settings for sentiment prediction.

To further verify this analysis, we collect several other sentiment text data sets which are described with details in the following Chapter 4. These ten data sets are performed on the DBNs with the two introduced neuron models above, and the network is set as: input (visible units) $\rightarrow$ 400 hidden units $\rightarrow$ 100 hidden units $\rightarrow$ output layer (class labels). The other hyper-parameters are set the same as those in the previous experiments. Finally, the data is fed into DBN-Gaussian and DBN-sigmoid respectively. The total accuracy through five-fold cross validation is applied here for evaluation comparison to further verify our conclusion. Table 3.5 shows that the DBN-Gaussian still performs better. In detail, DBN-Gaussian has won in six data sets and only one loss, while DBN-sigmoid has only one win. Besides, the average total accuracy of DBN-Gaussian is 67.81%, also higher than 66.34% for DBN-sigmoid.

3.3 Chapter Summary

In this chapter, we propose an idea of Gaussian neuron by changing the neuron model that can flexibly adapt the DBN model into sentiment prediction, and then we accommodate both the total accuracy and F_1 score to evaluate the performance. Our experimental results reveal that there do exist some significant improvements on the performance of Gaussian neuron compared to the widely used sigmoid neuron in DBNs, especially on imbalanced data sets. The results further demonstrate our initial assumption that the neuron model in DBNs needs more investigation for specific applications.

Table 3.5: Experimental results for further verification (%).

Data set	DBN-sigmoid	DBN-Gaussian
STS-T	61.01	67.07 *
STS-G	83.50	83.45
HCR	64.91	66.00
HCR2	78.11 *	75.63
SST	53.40	55.50 *
SST2	70.00	71.23 *
FT	51.16	53.22 *
FT2	68.18	69.66 *
GT	57.69	60.07 *
GT2	75.39	76.24
average	66.34	**67.81**

Chapter 4

Incorporating Positional Information into Deep Belief Networks for Sentiment Classification

4.1 DBN Incorporating Positional Information

Each sentence is traditionally represented by a bag-of-words vector. But in this way it loses the word positional information (positional contribution

and word order), because it contributes much to the sentiment polarity of a sentence. This work is to explore an effective way to incorporate positional information of a sentence into DBNs for sentiment classification.

Firstly, based on the positional information that is used in the NB classifier [58], we propose a new word positional function by scale normalization and apply it into DBNs for sentiment classification. Secondly, we propose a novel word-to-segment matrix representation of text to improve the traditional bag-of-words representation in DBNs for sentiment classification. There are two related studies: (1) The word-position matrix representation is proposed for natural language disambiguation tasks [51]; and (2) An effective way of bag-of-words conversion to make region vectors is presented to exploit the word order of text for text categorization using CNNs [29]. The difference between our word-to-segment matrix representation and the above two studies is that, we propose that each text is divided into several segments, and then the word-to-segment matrix representation is supposed to represent the segment order information of a sentence, but the word order information within the segment is omitted to simplify the representation.

4.1.1 Positional Contribution

Typically, a word is a grammar component in a sentence, and it has neighboring words resulting in different combinations of words, which means each word has a positional contribution value in the sentence. Basically, a word

should be assigned a relatively higher weight as its position occurrence increases in the sentence. The reason is that when we read the first word, the sentence's sentiment polarity is not clear at all, but when we go to the last word after reading all previous words, the sentiment polarity becomes unambiguous.

The positional information has been introduced into the NB classifier [58] in the form of Eq. 4.1, in which a simple linear interpolation is used to measure the position-dependent information in a sentence. Suppose the vocabulary has N word attributes, W_j denotes the j^{th} word (attribute) in the vocabulary, and it has value of x_j that is equal to zero when word W_j does not exist in the sentence, otherwise it is calculated via Eq. 4.1.

$$x_j = q_0 + q \cdot \frac{p}{n}, q_0 \geq 0, q > 0, j = 1, 2, ..., N, \tag{4.1}$$

where q_0 represents some constant value from the starting position of the sentence, and q is the position fractional weight, p is the position occurrence of word W_j in the sentence, n is the length of the sentence.

In order to normalize the values from Eq. 4.1 within the same scale, we force the value x_j to fall in the range of $[0, 1]$ (scale normalization). So we give a special positional contribution form described in Eq. 4.2.

$$x_j = \theta + (1 - \theta) \cdot \frac{p}{n}, 0 \leq \theta \leq 1, \tag{4.2}$$

where p and n represent the same as those in Eq. 4.1, θ is the ratio between the word's presence and its positional contribution within the range of [0, 1]. When $\theta = 0$, the value only represents the positional contribution, while it means the traditional presence value if $\theta = 1$. Actually, the form of Eq. 4.2 is a special form of Eq. 4.1 just through the assumption of $q + q_0 = 1$. However, given that $0 < p \leq n$ is correct for each sentence because p represents the word's position in a sentence while n denotes the length of the sentence, this form offers the flexibility to adjust the ratio θ to assign the value into the interval of [0, 1] that is fed into the DBNs. Each sentence incorporating the positional contribution is represented by an N-dimensional vector.

4.1.2 Matrix Representation

To better represent a sentence with the vocabulary words and incorporate the word order information into the model, also inspired by the linear transformation of the word-position matrix representation in the word disambiguation task [51], we try to use a matrix to represent a sentence. Intuitively, word-to-word matrix representation is introduced here. Suppose that a sentence is represented by an $N*N$ matrix M with the following two definitions:

- $M_{ii} = 1$, if the word W_i exists in the sentence, otherwise is 0, for $i =$ 1, 2, ..., N;
- $M_{ij} = 1$,if the word W_i occurs before W_j with only one space (the two

words are neighbors in the sentence), otherwise is 0, for i, j = 1, 2, ..., N (i is not equal to j).

For instance, assume the vocabulary is {"*are*", "*cat*", "*mat*", "*on*", "*sat*", "*take*", "*that*", "*the*"} with size $N = 8$, and we retrieve a simple sentence as "*the cat sat on that mat*". So based on the above definitions, an example of word-to-word matrix representation M for this sentence is described in Table 4.1.

Table 4.1: An example of word-to-word matrix representation.

Vocab	*are*	*cat*	*mat*	*on*	*sat*	*take*	*that*	*the*
are								
cat		1			1			
mat			1					
on				1			1	
sat				1	1			
take								
that			1				1	
the		1						1

It is shown that the word-to-word matrix representation can exactly describe the word order in each sentence. However, because there is normally a large vocabulary size for each training data set, the word-to-word matrix representation will consequently result in an extremely large size of input for training. Specifically, if the vocabulary size is N, the word-to-word matrix will be N^2, then the training time will be squarely increased, which will also cost too much memory of the computer.

In order to decrease the dimension size, we come up with an idea: each

sentence is roughly divided into three segments in grammar order of the sentence. For example, the sentence "*the cat sat on that mat*" is divided into three segments as: segment 1 of {"*the cat*"}, segment 2 of {"*sat on*"}, and segment 3 of {"*that mat*"}. In this case, we do not consider the word order in a local region (within each segment), but we take into account the order information of the three segments in the sentence, which will not lose much information for short texts, and meanwhile the dimensional size is not too large.

Hence, we propose a simplified form of word-to-word matrix representation called word-to-segment matrix representation. But there is still some difference between word-to-word and word-to-segment: word-to-word is denoted by a 0-1 matrix, while word-to-segment would distinguish word's impact on different segments (detailed in the later matrix definitions). Based on the different impact relationship values, we propose two types of matrix representations. Considering the words in a sentence are normally organized in a grammatical order, and also to simplify the problem, the number of segments is not defined from the grammar view, but from the view that the number of words in each segment would be approximately similar. Specifically, let $nSeg$ denote the number of segments, n is the sentence length for a specific sentence. Then we define $nSeg$ as below:

- If $n < nSeg$, then each segment is at most one word in the sentence, for example, the sentence "*love it*" to be divided into three segments,

then the first segment is {"*love*"}, the second segment is {"*it*"}, while the third segment is empty;

- If $n \geq nSeg$, the length of each segment $segLen$ (except for the last segment) is defined as the floor integer of n over $nSeg$. In detail, for the segment k through 1 to $nSeg - 1$, the position of segment k in the sentence is from $1 + (k-1) * segLen$ to $k * segLen$; while for the segment $nSeg$, the position is from $k * segLen + 1$ to n. For example, a sentence "*it is the best*" to be divided into three segments in order is {"*it*"}, {"*is*"}, and {"*the best*"}.

On the other hand, there are two types of impact definitions introduced here. Let the word-to-segment matrix denoted by M (each column represents each word in the vocabulary and each row denotes each segment), its element value is defined in the following two types accordingly. For both two types' definitions, $M_{ij} = 1$ when the word W_j occurs in the segment i for i = 1,2,3 and j = 1,2, ..., N, which means this word has full impact on its own segment. The word's impact on different segments is described below.

To clearly illustrate the matrix representation, we still take the previous example illustrated in Table 4.1 for example. For the first type in Table 4.2, the words existing in the first segment(S1: "*the cat*") have no impact on the latter two segments, the words in the second segment (S2: "*sat on*") have impact value of a on the first segment, and the words in the third segment (S3: "*that mat*") have impact on both the second and first segments with

impact values a and b respectively. Given that the distance between the third segment and the first segment is larger than that between the third segment and the second segment, the impact values should satisfy the condition of $0 < b < a < 1$.

Table 4.2: Type one of word-to-segment matrix representation for one sentence (matrix1).

Segment	*are*	*cat*	*mat*	*on*	*sat*	*take*	*that*	*the*	words
S1		1	b	a	a		b	1	"*the cat*"
S2			a	1	1		a		"*sat on*"
S3			1				1		"*that mat*"

The second type is shown in Table 4.3. Another reasonable assumption is that a word in the segment not only has impact on its previous segment but also on its latter segment, but does not have impact on its remote segment (e.g., the first and the third segments). Furthermore, the three segments do not have the same impact on each other because they locate in different positions of the sentence. Specifically, the words in segment 1 (or segment 3) have impact on segment 2 with value c (or e), and the words in segment 2 have impact on both segment 1 and segment 3 with value d. The impact values have the restriction of $0 < c < d < e < 1$.

Table 4.3: Type two of word-to-segment matrix representation for one sentence (matrix2).

Segment	*are*	*cat*	*mat*	*on*	*sat*	*take*	*that*	*the*	words
S1		1		d	d			1	"*the cat*"
S2		c	e	1	1		e	c	"*sat on*"
S3			1	d	d		1		"*that mat*"

4.1.3 DBN Settings

Through the definitions of the above two types of word-to-segment representations for each sentence, the input into the DBN model is transformed via the average operation over each column of the matrix. That is, each sentence is represented by an N-dimensional vector and each element value equals to the average of corresponding column of the three segments, which not only keeps the same size of input features as the original bag-of-words representation, but also contains the prior segment order information of the sentence.

Therefore, compared to traditional bag-of-words representation, there are overall four kinds of inputs: baseline bag-of-words representation, positional representation, matrix1 representation and matrix2 representation. The inputs are then fed into our DBNs respectively for further experimental comparison.

The DBNs introduced in this chapter are similar in [22], which includes RBMs to learn hierarchical representations in an unsupervised way before the supervised BP training. The whole learning process (two phases) is similar as described in Chapter 3, and the neuron model is the Gaussian neuron (μ=0.8, $\sigma^2 = 3^2$) because it has proved better than the widely used sigmoid function. Specially, for each data set, the same size of input is fed into the DBNs, so the running time difference lies in the computation of different input transformations that will not cost much time compared to

DBN training. Therefore, here we focus on their classification performance on total accuracy.

4.2 Experiments and Results

In this section, we design a variety of experiments to verify the power of positional information of sentences in the DBNs. Then the experimental results are compared from different angles to analyze the effect and sensitivity of positional information for sentiment classification.

4.2.1 Data Collection and Pre-processing

This work mainly focuses on short text sentiment classification since the positional representation and word-to-segment matrix representation described above will lose some important effect for long text sentiment classification. Therefore, several short text data sets, e.g. Twitter messages limited to only 140 characters long, are selected here for implementations.

(1) STS-T: Stanford Twitter Sentiment (STS) Test Set [61], a manually annotated data set of STS with positive and negative labels.

(2) STS-G: a gold data set extracted from STS with positive and negative labels [61].

(3) SST: Sentiment Strength Twitter data set [61], including three senti-

ment labels (positive, negative and neutral). We will use the data set as two, one is tri-class data set and the other is binary class removing neutral tweets.

(4) HCR: Health Care Reform (HCR) Twitter data set [61], including three classes (positive, negative and neutral). Similar as (3), the data set is used a tri-class set and a binary class set.

(5) FT: Full Twitter data [2],including three classes (positive, negative and neutral). Similar as (3), the data set is used of tri-class data set and binary class data set.

(6) GT: Game Tweets regarding the video games, are real-time collected and labeled by us with three labels (positive, negative and neutral). Also, this data set is used for tri-class and binary data sets.

(7) HCR2, SST2, FT2, GT2: These four data sets are respectively derived from HCR, SST, FT and GT with neutral tweets removed for binary classification.

Each data set needs to be pre-processed to obtain the initial trainable data for training the model as described in Chapter 3. In order to decrease the dimension of the vocabulary size without losing much attribute information and speed up training process, each data set is loaded into Weka 3.6.12 [20], and then filtered using the unsupervised *StringToWord* filter to keep the most important words as the vocabulary set for our experiments. The data

set description is summarized in Table 4.4.

Table 4.4: Summary statistics of data sets used in the experiments.

Data set	size	N	avgL	maxL	C	class sizes	mini-batch
STS-T	495	2067	14.5	32	3	179/139/177	11
STS-G	2000	1172	16.4	33	2	632/1368	50
HCR	2300	1018	18.8	32	3	541/400/1359	46
HCR2	1900	1084	19.1	32	2	541/1359	38
SST	4000	985	16.6	37	3	1251/1800/949	50
SST2	2200	1030	17.6	37	2	1251/949	44
FT	5000	1066	14.0	34	3	1664/1664/1672	50
FT2	3250	1162	15.2	34	2	1625/1625	50
GT	12000	929	13.8	33	3	3983/4013/4004	50
GT2	8000	984	14.3	33	2	3984/4016	50

*size: number of examples; N: number of words extracted as the vocabulary; avgL: average text length; maxL: maximal text length; C: number of classes; class sizes: number of examples for positive/neutral/negative if $C = 3$, otherwise for positive/negative if $C = 2$; mini-batch: the size of mini-batch for each data set in the experiments.

4.2.2 Experimental Results and Analysis

In this chapter we implement the experiments based on the following four model variations:

(1) DBN-presence: DBN model with the basic bag-of-words representation (presence / non-presence) as input with $\theta = 1$ in Eq. 4.2.

(2) DBN-position: DBN model with the vocabulary incorporating the word's presence and positional contribution value as input with $0 \leq \theta < 1$ in Eq. 4.2.

(3) DBN-matrix1: DBN model with the average of first type word-to-

segment matrix representation as input. Each word attribute has value one if existing in its own segment of the sentence (full impact on its own segment), while it only has impact on its all preceding segments. That is, the words existing in the first segment have no impact on the other two segments, the words in the second segment have some impact (value a) on the first segment, and finally the words in the third segment have impact on both the second and first segments with decreasing impact values of a and b respectively in Table 4.2, with the restriction of $0 < b < a < 1$.

(4) DBN-matrix2: DBN model with the average of second type word-to-segment matrix representation as input. Each word attribute has value one if it exists in its own segment of the sentence, and also has impact on its nearest segment(s) including its preceding and posterior segments with the impact values of c, d, e with $0 < c < d < e < 1$ in Table 4.3.

In this chapter, we perform the above four models on ten data sets listed in Table 4.4. Through a variety of implementations similar as introduced in Chapter 3, the DBN structure is finally manually set to consist of two hidden layers. Specifically, it is: input (visible units) $\rightarrow$ 400 hidden units $\rightarrow$ 100 hidden units $\rightarrow$ output layer (class labels). The hidden unit is sigmoid function, and it has also proved more effective from the penultimate layer to output layer during our experiments. In addition, the NB classifier is also performed for reference comparison.

To speed up the experiments in this chapter, each data set is performed using five-fold cross validation (four for training and one for testing) to obtain an average accuracy. For some hyper-parameters in our system, firstly in RBM unsupervised training, the momentum is 0.1 and learning rate is 0.1, the number of epochs is set as 50. Secondly, for supervised BP training, the mini-batch sizes are listed in Table 4.4, the sparsity penalty parameter is 0.1, and the maximum number of epochs is 500 with 10^{-6} as the convergence control based on early stopping rule.

Here we firstly manually set the parameters for the classification results and give a comparison among different DBN model variations. Each accuracy value in the following tables is average total accuracy of cross validation. Then, we perform a range of experiments to analyze the effect of parameters with respect to θ, (a, b), and (c, d, e), investigating whether there exist some hidden patterns for these parameters in different data sets or whether they are robust to the classification performance.

4.2.2.1 Classification Results

Here we set the parameters for each data set listed in Table 4.5 for experimental results through a number of tests.

The classification results of different model variations on the ten data sets are shown in Table 4.6. Note that the NB classifier is performed based on the presence/non-presence word features of each data set. The accuracy val-

Table 4.5: Positional parameters of each data set used in the experiments.

Model	Position (θ)	Matrix1 (a, b)	Matrix2 (c, d, e)
STS-T	0.9	(0.2, 0.1)	(0.3, 0.4, 0.6)
STS-G	0.9	(0.4, 0.1)	(0.2, 0.4, 0.6)
HCR	0.7	(0.6, 0.5)	(0.3, 0.4, 0.8)
HCR2	0.9	(0.8, 0.1)	(0.2, 0.5, 0.8)
SST	0.7	(0.8, 0.1)	(0.1, 0.5, 0.6)
SST2	0.6	(0.4, 0.1)	(0.3, 0.5, 0.7)
FT	0.6	(0.4, 0.1)	(0.3, 0.5, 0.8)
FT2	0.5	(0.2, 0.1)	(0.2, 0.5, 0.8)
GT	0.5	(0.4, 0.1)	(0.3, 0.4, 0.7)
GT2	0.7	(0.2, 0.1)	(0.2, 0.4, 0.7)

ues of the three models (DBN-position, DBN-matrix1, DBN-matrix2) are respectively compared with DBN-presence using a two-tailed t-test. The results indicate all the four DBN models perform better than the NB model (average 63.74%). Also, in the four model variations, the DBN-presence has no significantly higher accuracy than the other three DBN models. Meanwhile, DBN-matrix2 has a relatively highest average accuracy (69.22%) for all the data sets, and DBN-presence performs worse than the other three DBN models.

In Table 4.7, an explicit comparison among the four model variations is summarized, where $w/l/t$ indicates the same meaning as Chapter 3 (The model wins in w data sets, loses in l data sets, and ties in t data sets). In Table 4.7, the three rows (DBN-position, DBN-matrix1, and DBN-matrix2) show their comparisons with DBN-presence respectively. Especially, because the DBN-presence only has ties or losses with the other three models, it is

Table 4.6: Experimental results on classification accuracy (%).

Model	NB	DBN-presence	DBN-position	DBN-matrix1	DBN-matrix2
STS-T	66.90	67.07	69.09 *	68.48 *	67.67
STS-G	79.70	83.45	84.55	84.10	83.75
HCR	63.10	66.00	66.65	66.87	68.17 *
HCR2	74.40	75.63	76.58	77.79 *	77.97 *
SST	51.70	55.50	56.15	56.55	56.85 *
SST2	66.50	71.23	73.41 *	73.64 *	73.77 *
FT	47.00	53.22	54.66 *	54.82 *	55.28 *
FT2	63.30	69.66	69.63	69.94	70.37
GT	54.40	60.07	61.11	61.28	61.75 *
GT2	70.40	76.24	76.51	76.96	76.65
average	63.74	67.81	68.83	69.04	**69.22**

*The numbers followed by the sign * indicate the accuracy passes the t-test at the significance level of 95% for each data set.

omitted in Table 4.7. To summarize more clearly, the models are compared in three aspects: all data sets (10), tri-class data sets (5) (Note: five data sets with three labels: STS-T, HCR, SST, FT, and GT), and binary data sets (5) (Note: five data sets with only two labels: STS-G, HCR2, SST2, FT2, and GT2.).

Table 4.7: Summary of classification accuracy comparisons.

Model	all data sets (10)	tri-class data sets (5)	binary data sets (5)
DBN-position	3/0/7	2/0/3	1/0/4
DBN-matrix1	4/0/6	2/0/3	2/0/3
DBN-matrix2	6/0/4	4/0/1	2/0/3

Table 4.6 demonstrates that DBN-matrix2 performs best on the average accuracy. Also, Table 4.7 shows that it performs best on all the ten data sets for one-by-one comparisons (six wins and four ties). Besides, the DBN-matrix1 and DBN-position also have four wins and three wins respectively,

while the DBN-presence has no winners. Consequently, it is obvious that the positional information of sentences exactly provides positive effect on the sentiment classification issues, and also the matrix representation is more effective than positional contribution form. On the other hand, the positional information seems more effective for tri-class data sets, because DBN-matrix2 has four wins and only one tie for five tri-class data sets. It may be due to the fact that the word position and the word order will account more exact information for the sentiment. So the more positional information is integrated, the better for multi-class sentiment classification.

To give a further overview of the effect of the positional information for sentiment classification. The DBNs with sigmoid neuron (discussed in Chapter 3) are also performed. The parameters are shown in Table 4.8 and the results are obtained in Table 4.9.

Table 4.8: Positional parameters used in DBNs with sigmoid.

Model	Position (θ)	Matrix1 (a, b)	Matrix2 (c, d, e)
STS-T	0.3	(0.6, 0.1)	(0.3, 0.4, 0.6)
STS-G	0.6	(0.8, 0.3)	(0.3, 0.4, 0.8)
HCR	0.9	(0.6, 0.5)	(0.2, 0.5, 0.7)
HCR2	0.6	(0.8, 0.3)	(0.2, 0.5, 0.8)
SST	0.8	(0.4, 0.3)	(0.2, 0.4, 0.7)
SST2	0.3	(0.8, 0.1)	(0.2, 0.5, 0.7)
FT	0.4	(0.6, 0.1)	(0.3, 0.5, 0.8)
FT2	0.9	(0.8, 0.3)	(0.2, 0.5, 0.8)
GT	0.7	(0.6, 0.5)	(0.2, 0.5, 0.8)
GT2	0.7	(0.8, 0.3)	(0.2, 0.4, 0.7)

Table 4.9 shows that all the four DBNs perform better than the NB classi-

fier except for STS-T data set. Besides, compared to DBN-presence, DBN-position, DBN-matrix1, and DBN-matrix2 have five wins, four wins, and seven wins respectively. The results also demonstrate that DBN-matrix2 performs best for DBNs with sigmoid neuron.

Table 4.9: Further results for DBNs with sigmoid on classification accuracy (%).

Model	NB	DBN-presence	DBN-position	DBN-matrix1	DBN-matrix2
STS-T	66.9	61.01	64.65 *	65.66 *	66.06 *
STS-G	79.7	83.50	84.85	82.60	83.25
HCR	63.1	64.91	66.40 *	65.61	66.33 *
HCR2	74.4	78.11	78.66	78.26	78.89
SST	51.7	53.40	54.45	54.80	55.30 *
SST2	66.5	70.00	71.59 *	71.64 *	72.77 *
FT	47.0	51.16	52.70 *	52.56 *	53.44 *
FT2	63.3	68.18	68.71	68.95	68.92
GT	54.4	57.69	59.93 *	60.73 *	60.71 *
GT2	70.4	75.39	75.83	75.46	76.94 *
average	63.74	66.34	67.78	67.61	**68.26**

In essence, the positional contribution form and word-to-segment matrix representation are different ways to describe the word positional information of a sentence. For positional contribution form, the contribution values are linearly augmented as the position increases; for matrix1 representation, the words have relatively larger weights in the latter segments, but they are the same in the same segments; while for matrix2 representation, the words in the middle segments have largest weights since they play impact on both its previous and latter segments. To summarize, the word positional representations improve the word presence features for short text sentiment classification.

4.2.2.2 Effect of Parameters

In our experiments, there are some important parameters which need to be set manually. Whether the results are sensitive to the parameters needs further investigation. Hence, we perform a variety of experiments in DBNs with Gaussian neuron to examine the effect of parameters in this part.

Firstly, we investigate the effect of position parameter θ in Eq. 4.2. Here the values are set from 0.0 through 1.0 with the interval of 0.1. Each data set is performed with five-fold cross validation for each parameter. The results are shown in Table 4.10.

Table 4.10: Classification accuracy of four data sets vs. position parameter.

θ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
STS-T	59.60	60.61	61.82	65.05	65.86	63.03	62.83	62.02	66.26	**69.09**	67.07
STS-G	81.25	81.20	81.95	83.95	83.30	84.00	84.95	84.50	84.10	**84.55**	83.45
HCR	65.70	66.22	65.17	64.57	65.43	65.61	64.83	**66.65**	65.91	65.09	66.00
HCR2	74.58	75.68	75.89	76.05	76.05	76.37	76.53	75.47	76.32	**76.58**	75.63
SST	53.73	54.75	55.25	55.63	54.95	55.88	54.38	**56.15**	55.80	54.68	55.50
SST2	71.18	71.68	71.86	71.18	72.23	71.77	**73.41**	72.05	71.14	72.18	71.23
FT	52.32	54.20	54.20	53.34	53.84	53.90	**54.66**	53.56	53.78	54.00	53.22
FT2	67.54	68.58	68.71	67.72	68.52	**69.63**	69.38	68.58	68.31	68.46	69.66
GT	59.18	59.94	59.35	60.68	61.00	**61.11**	60.94	61.05	61.03	60.40	60.07
GT2	73.96	74.51	75.50	75.09	75.38	76.19	76.33	**76.51**	76.36	75.54	76.24

*Bold numbers represent the highest accuracy for each row (except the presence with $\theta = 1.0$), similar with the later tables.

Table 4.10 shows the effect of position parameter θ on the average classification accuracy of the ten data sets. In this table, the last data column for each data set with position ratio at 1.0 represents the traditional bag-of-words representation (presence), and the bold number is the highest accuracy value in each row for θ values ranging from 0.0 through 0.9. It means that different ratio values between the word's presence and its positional contri-

bution have different contributions to the text's sentiment. Normally, if θ is relatively bigger, less positional information is integrated, while the word contributes more information if θ is smaller. It indicates that, the parameter value of θ needs a careful investigation for each data set. In Table 4.10, the θ value corresponding with the bold number for each row is selected for our final experiments comparison introduced previously (similar in Table 4.11 and Table 4.12).

Secondly, the matrix1 representation is investigated for the values of $a = 0.2$, 0.4, 0.6, 0.8 and $b = 0.1$, 0.3, 0.5, with the restriction of $0 < b < a < 1$. Table 4.11 shows the results of accuracy on each set of (a, b). For instance, the data set of STS-T has the highest average accuracy of 68.48% at (0.2, 0.1) among all the nine parameter settings for matrix1 representation. It gives a picture of the sensitivity for the values of (a, b) on the performance and provides evidence to choose good parameters.

Table 4.11: Classification accuracy of four data sets vs. matrix1 parameters.

(a, b)	(2,1)	(4,1)	(6,1)	(8,1)	(4,3)	(6,3)	(8,3)	(6,5)	(8,5)
STS-T	**68.48**	65.45	65.86	65.45	66.06	63.84	63.43	61.41	61.21
STS-G	83.95	**84.10**	83.30	83.10	83.00	83.40	82.85	82.65	80.40
HCR	66.65	66.43	65.35	66.70	66.04	65.87	64.91	**66.87**	66.70
HCR2	76.95	76.89	77.47	**77.79**	76.89	75.53	76.58	76.53	75.84
SST	56.50	56.23	55.65	**56.55**	55.40	56.00	55.15	56.20	55.13
SST2	71.59	**73.64**	72.41	72.05	72.50	72.36	71.95	71.59	71.91
FT	54.26	**54.82**	54.78	53.56	54.00	54.20	53.48	54.06	53.88
FT2	**69.94**	69.17	69.60	68.43	68.77	69.11	69.20	68.77	67.42
GT	61.13	**61.28**	60.62	60.59	59.93	61.06	60.28	61.11	60.20
GT2	**76.96**	76.50	76.70	75.43	75.66	75.46	75.71	75.86	75.68

*The values of (a, b) is a simplified form in order to format the table: (2,1) represents $a = 0.2$, and $b = 0.1$, similar to other numbers.

Finally, the matrix2 representation is performed with the parameter values of $c = 0.1, 0.2, 0.3$, $d = 0.4, 0.5$ and $e = 0.6, 0.7, 0.8$. The accuracy results of each set of (c, d, e) are given in Table 4.12. For example, the parameter setting $(c = 0.3, d = 0.5, e = 0.8)$ reaches the highest accuracy 55.28% for FT data set.

Table 4.12: Classification accuracy of four data sets vs. matrix2 parameters.

(c, d, e)	(1,4,6)	(1,4,7)	(1,4,8)	(1,5,6)	(1,5,7)	(1,5,8)	(2,4,6)	(2,4,7)	(2,4,8)
STS-T	65.86	65.25	65.25	60.81	66.67	63.84	62.63	63.84	66.26
STS-G	83.25	83.25	82.50	83.05	81.90	82.55	**83.75**	83.35	81.55
HCR	66.04	65.91	65.78	67.22	66.48	66.83	65.52	66.61	68.52
HCR2	77.26	75.79	76.95	77.32	76.63	74.79	75.53	76.58	76.63
SST	56.10	56.15	56.43	**56.85**	54.93	55.18	55.60	56.68	55.58
SST2	72.32	73.41	72.05	71.73	72.32	71.82	72.27	72.23	72.50
FT	54.18	54.96	54.78	54.14	54.86	54.20	54.42	54.78	55.00
FT2	69.88	69.05	69.23	68.55	69.38	69.51	69.66	69.11	67.82
GT	60.98	61.10	60.88	61.10	60.40	61.33	61.28	61.23	61.03
GT2	76.15	75.75	76.58	75.36	75.75	75.91	76.13	**76.65**	76.60
(c,d,e)	(2,5,6)	(2,5,7)	(2,5,8)	(3,4,6)	(3,4,7)	(3,4,8)	(3,5,6)	(3,5,7)	(3,5,8)
STS-T	67.07	64.44	62.63	**67.67**	64.44	63.84	66.06	65.45	62.83
STS-G	82.95	83.10	81.75	82.95	83.15	83.70	83.10	83.70	83.50
HCR	66.61	66.09	66.57	66.26	65.17	**68.17**	65.65	66.39	66.43
HCR2	76.79	77.16	**77.97**	77.21	77.16	77.47	76.74	77.37	75.95
SST	56.35	54.68	55.08	55.75	56.40	56.43	56.20	55.58	55.90
SST2	72.64	71.82	72.09	72.36	71.86	71.55	73.50	**73.77**	72.82
FT	54.08	54.20	54.32	53.98	54.06	52.80	54.54	54.74	**55.28**
FT2	69.29	67.32	**70.37**	69.48	69.48	69.14	69.14	68.62	68.37
GT	60.93	60.93	61.34	61.46	**61.75**	61.21	61.15	60.83	61.13
GT2	76.00	75.40	76.50	75.78	75.73	76.40	75.65	76.04	75.11

*The values of (c, d, e) is a simplified form in order to format the table: (1,4,6) represents $c = 0.1$, $d = 0.4$ and $e = 0.6$, the other numbers are similar.

The results of different parameter settings not only show how to choose the parameters, but also demonstrate these parameters play a significant role in DBN models, because the performance results are not very robust to those parameters. On the other hand, it is possible that some other better param-

eters are not included in our experiments since it is difficult to perform all experiments with exhaustive parameter searching. However, it is really necessary to point out that the positional information (positional contribution and word-to-segment) affects the sentiment classification positively if we set the right parameters.

4.3 Chapter Summary

In this chapter, we propose several ways to incorporate the positional information of texts into DBNs with four model variations for sentiment classification and perform a variety of experiments to verify the effect of positional information towards the sentence sentiments. By choosing the reasonable parameters, the experiments reveal that the word position and word order can improve the classification performance. The results indicate that the traditional bag-of-words representation can be improved by incorporating some positional information represented by the word positional contribution form and the word-to-segment matrix representation. Also, it can be seen that positional information works more effective for tri-class classification compared to binary classification. In other words, each word attribute in a sentence has different effect for sentence sentiment.

Chapter 5

Improving Deep Belief Networks via the Delta Rule for Sentiment Classification

5.1 DBN with the Delta Rule

In Chapter 3 and Chapter 4, the DBN is usually trained in a greedy layer-wise fashion of two phases including the RBM unsupervised training and the supervised BP training [22, 5]. This ordinary two-phase learning approach of DBNs has proved more effective than the learning with only the BP algorithm. On the one hand, for natural language understanding tasks, the DBN unsupervised training phase is applied to learn new features from unlabeled

data through a multi-layer generative model. And then the newly learned features are adopted to train other classifiers such as SVMs [62]. On the other hand, the PCD algorithm is a gradient approximation algorithm to use the pre-trained RBMs to model the joint distribution of DBNs [72]. These two points indicate that the pre-trained features from DBN unsupervised training are extensively useful.

Hence, in this chapter, we propose another way to take advantage of the pre-trained RBMs for sentiment analysis. To our knowledge, there is no work done in the literature incorporating the delta rule (DR) into DBNs. So we propose a new learning algorithm via the DR embedded within DBNs. That is, we apply the DR to learn the feed forward single-layer network of the two layers from the RBM, which further slightly updates the weights between the two layers of the single-layer network, and then these weights are used for initialization of the whole BP process. Furthermore, this algorithm is further verified with experiments in sentiment analysis.

5.1.1 Introducing the Delta Rule into DBNs

A DBN is normally trained as building several stacks of RBMs in a greedy layer-wise fashion [5]. Hence, it can also be viewed as several single-layer networks that are trained using the DR learning independently. Through the DR learning based on the RBM training, the BP algorithm for the whole network is supposed to learn more accurate hierarchical representations and

keep more correction signal, which can improve the learning of the whole network.

The DR is a learning rule using gradient descent to fine-tune the weights between the input units and output units in a single-layer neural network, in which the inputs are directly connected to the outputs through a series of weights. It is a special case of the general BP algorithm, and its weights are obtained by trying to minimize the error of actual output and target output in a single-layer neural network through gradient descent. The error for a neural network with k outputs can be measured as Eq. 5.1 [78].

$$Err = \sum_{j=1}^{k} \frac{1}{2}(t_j - y_j)^2, \tag{5.1}$$

where t_j is the target output, y_j is the actual output. It holds that, $y_j = \varphi(u_j)$, $u_j = \sum_i x_i w_{ji}$ (i indicates the i^{th} input unit), φ is an activation function of neuron j, u_j is weighted sum for the j^{th} neuron over all input units, x_i is the i^{th} input unit, and w_{ji} represents the weight connecting the i^{th} input unit and the j^{th} output unit.

In order to minimize the error in Eq. 5.1 to obtain the weight matrix w. The partial derivative of the error with respect to each weight element $\frac{\partial Err}{\partial w_{ji}}$ is calculated towards gradient direction. Finally the target equation of gradient is calculated in Eq. 5.2.

$$\Delta w_{ji} = \alpha(t_j - y_j)\varphi'(u_j)x_i, \tag{5.2}$$

where α is a small learning rate, φ' represents the derivative of activation function φ. The error is then minimized when we update the weights towards the negative direction of gradient.

Based on the introduction for DR, in a specified single-layer network extracted from a neural network with two or more hidden layers, we should define the input and actual output since each data set only provides the original input. Hence, the input unit x_i and actual output y_j in a single-layer network are described in detail as below.

A single-layer network structure is actually similar as an RBM structure. The difference is that the single-layer network is unidirectional, while the RBM is bidirectional. The RBM is to learn a marginal probability distribution over its input, by maximizing the log probability over a training set V (input matrix, each row of which denotes an example v), that is written as $\arg\max_w \sum_{v \in V} \log P(v)$, where $P(v)$ denotes marginal probability over all the visible features, and w is weight matrix connecting the units of the two layers [4]. It means that the other layer of RBM can be generated by the means of CD-k (normally $k = 1$) [23].

Figure 5.1 depicts a deep network including visible layer V, two hidden layers (H1, H2), and output layer O. In detail, it is trained as stacks of two RBMs, also can be viewed as two single-layer networks after each RBM is trained,

where DR is incorporated. Take RBM1 as an example, which includes V and H1 with bidirectional relationship, and CD-k learning approach is accommodated to train RBM1 in an unsupervised fashion. After that, H1 is obtained from RBM1 training, so the structure from V to H1 is treated as a feed forward single-layer neural network, which is trained using DR. It is similar with RBM2 in Figure. 5.1. Finally, the weights of V and H1, H1 and H2 that are trained from DR are kept for initial weights of the whole BP process.

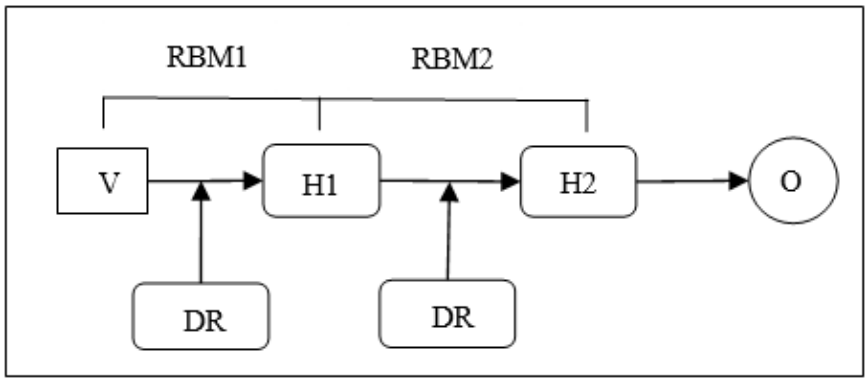

Figure 5.1: A deep network including two hidden layers.

The goal of the DR learning incorporated into DBN learning is to avoid the correction signal of each layer not to decrease as much as the ordinary DBN training. Thus, suppose that the hidden units for each hidden layer are generated from RBM training, we process these hidden units in order for DR learning in the single-layer network. The hidden units trained from RBM are originally a higher abstraction of the previous layer's units. While for the purpose to strengthen the feature abstraction, we make an extreme

treatment on these hidden units, which is to treat the largest element of each row as value one and the others as zeros. Hence, we obtain the actual output unit y_j for the single-layer network in Eq. 5.3, and its new input unit x_i in Eq. 5.4.

$$y_j = \begin{cases} 1 & \text{if } y_j = max(y); \\ 0 & \text{otherwise.} \end{cases}, \tag{5.3}$$

$$x_i = \begin{cases} v_i & \text{if it is the first RBM;} \\ \varphi(u_i) & \text{otherwise.} \end{cases}, \tag{5.4}$$

where v_i is the i^{th} visible unit, and u_i is weighted sum of the i^{th} neuron for its previous layer's hidden units. That is, when we perform the second (or next if exist) single-layer network, the input unit is computed via the activation function with respect to the weighted sum of the output units of the previous single-layer network.

5.1.2 Learning DBNs with the Delta Rule

The ordinary training process of DBNs is divided into two phases [22]: unsupervised RBM training and supervised neural network training. Our proposed learning method improves this ordinary training via introducing the DR learning, which is technically described as below (corresponding to that shown in Figure 5.1).

- Step 1: For the first RBM in the network structure, we train this RBM using CD-1, following the practical guide by G. Hinton [21];
- Step 2: Based on the weights generated from the first RBM, compute its hidden layer, and generate corresponding actual output layer and new input layer using Eq. 5.3 and Eq. 5.4;
- Step 3: Applying the DR learning approach to the single-layer network with the first layer of RBM as the input and generated output as the output layer;
- Step 4: For the second RBM (or next if exists), the new input layer is generated via the activation function with respect to the weights from Step 3 and its input units, then repeat from Step 1 to Step 3;
- Step 5: The weights trained from the DR are transmitted to the whole network that is fine-tuned using the traditional BP algorithm introduced by D.E. Rumelhart *et al* [60].

Compared to ordinary DBN training [21], this learning approach indeed has another phase: the DR learning for single-layer networks (Step 2 and Step 3 above). Hence, the running time difference lies in the training time of the DR learning. Because the DR training does not cost much time compared to the ordinary two-phase learning process, and meanwhile, this work is to propose a new way to avoid the correction signal to get dilute during whole BP training as the number of layers increases (deep networks), we do not focus on the

training time difference, but on their classification performances.

5.2 Experiments and Results

Based on the above discussion regarding how the DR is applied into ordinary DBN learning, we perform a variety of empirical studies for DBNs using the ordinary DBN learning approach and the new updated learning embedded with the DR on several sentiment text (Twitter) data sets to verify our idea. The ten data sets are the same as those used in Chapter 4, illustrated in Table 4.4.

Here, in order to investigate whether the DR works well on different network architectures, we implement the experiments using three types of deep networks on the ten data sets. That is, the three network structures are respectively manually set to consist of one, two and three hidden layers. Specifically, the network of one hidden layer is: input (visible units) $\rightarrow$ 200 hidden units $\rightarrow$ output layer (class labels); the network with two hidden layers is: input (visible units) $\rightarrow$ 400 hidden units $\rightarrow$ 100 hidden units $\rightarrow$ output layer; while the network including three hidden layers is: input $\rightarrow$ 600 hidden units $\rightarrow$ 200 hidden units $\rightarrow$ 50 hidden units $\rightarrow$ output layer. According to the discussion in Chapter 3, the hidden units are derived from Gaussian neuron function, and the sigmoid function is applied as the output function.

To speed up the experiments in this chapter, each data set is performed using five-fold cross validation (four for training and one for testing) to obtain an average accuracy. For some hyper-parameters in our experiments, we manually set them for each data set in Table 5.1 for experimental results through a variety of experiments. Especially, for the supervised BP training, the maximum number of epochs is 500 with 10^{-6} as the convergence control based on early stopping rule. The mini-batch sizes for each data set are respectively listed in Table 4.4.

As shown in Table 5.1, the ordinary DBN learning includes two parts of RBM and BP, while the DBN with DR learning (DBN-DR) includes all the three parts of RBM, DR, and BP. A small learning rate (0.001) of DR is set because it aims to slightly update the weights towards extreme labels generated from RBMs.

Table 5.1: Hyper-parameters set in the experiments.

Model parts	α	momentum	epochs	sparsity target
RBM	0.1	0.1	50	-
DR	0.001	-	100	-
BP	1.0	-	500	0.1

*RBM: Restricted Boltzmann machines, DR: Delta rule; BP: Back propagation.

The classification results of three deep network structures on the ten data sets are displayed in Table 5.2. The accuracy of DBN is compared with DBN-DR for the three networks independently using a two-tailed t-test. The results indicate DBN-DR performs better than DBN on the average accuracy for all the three networks. In detail, DBN-DR has a comparable average accuracy

of 68.04% with the DBN of 67.86% in the network with one hidden layer, and 68.03% compared to 67.81% for the network of two hidden layers. However, for the three hidden layers network, DBN has only 66.28% lower than DBN-DR of 68.06%.

Table 5.2: Experimental results on classification accuracy (%).

Dataset	One hidden layer		Two hidden layers		Three hidden layers	
	DBN	DBN-DR	DBN	DBN-DR	DBN	DBN-DR
STS-T	66.06	66.87	67.07	68.28 *	59.80	68.48 *
STS-G	83.55	83.40	83.45	83.90	84.00	83.90
HCR	67.30 *	65.61	66.00	64.96	63.96	64.35
HCR2	76.69 *	75.16	75.63	75.68	75.89	75.95
SST	55.70	55.90	55.50	55.45	51.98	55.63 *
SST2	71.73	72.64	71.23	73.00 *	70.32	72.32 *
FT	53.16	53.98	53.22	52.90	52.40	54.28 *
FT2	68.92	69.29	69.66	68.86	68.65	68.83
GT	59.80	61.23 *	60.07	61.23 *	58.95	60.87 *
GT2	75.71	76.28	76.24	76.06	76.86	75.98
average	67.86	68.04	67.81	68.03	66.28	**68.06**

Table 5.3: Summary of classification accuracy comparisons.

Model	One hidden layer	Two hidden layers	Three hidden layers
DBN	2/1/7	0/3/7	0/5/5
DBN-DR	1/2/7	3/0/7	5/0/5

Table 5.3 gives a summary comparison for these results. It shows that DBN-DR and DBN make little difference for the network with one hidden layer since DBN has two wins and DBN-DR has one win. However, what we focus is that DBN-DR has five wins and no loss in the network of three hidden layers, and it has three wins and no loss in the two hidden layers network. So

the summary in Table 5.3 indicates that DBN-DR works better than DBN for the networks with two and three hidden layers. Furthermore, based on these results, we compare the value gaps between the average accuracy of the two models for each network, the gap values for the networks with one, two, and three hidden layers are respectively 0.18 (= 68.04 - 67.86), 0.22 (= 68.03 - 67.81), and 1.78 (= 68.06 - 66.28). This shows that the gap value is increasing as the architecture goes deeper. It not only indicates that DBN-DR performs better than DBN for deep networks with three hidden layers, but also reveals that DBN-DR works better on the network of more hidden layers than that of fewer hidden layers.

The purpose that we propose the DR learning into DBNs is to keep the correctional signal of the BP process as much as possible so that the weight parameters are more accurate for the abstracted representations. Intuitively, if the network has more hidden layers, the DBN with DR learning will keep much more correction signal compared to the ordinary DBN learning. To summarize, the DR really helps to improve DBN learning for sentiment classification, and also it performs better as the number of hidden layers increases.

To further verify this conclusion, we also perform the above two DBNs with sigmoid neuron for comparisons. All the parameters are set the same as previously. The results are displayed in Table 5.4 and summarized in Table 5.5.

The results in Table 5.4 indicate DBN-DR performs better than DBN on

the average accuracy for all the three networks. Also, the gap values for the networks with one, two, and three hidden layers are respectively 0.40 (=67.38-66.98), 0.82 (=67.16-66.34), and 1.68 (=67.44-65.76). These values also increases when the architecture becomes deeper. On the other hand, Table 5.5 gives a summary comparison for these results, and shows that, DBN-DR and DBN make no difference for the network with one hidden layer. However, DBN-DR has five wins and no loss in the network of three hidden layers, and it has four wins and one loss in the two hidden layers network. These results help to further verify our conclusion introduced previously in this chapter.

Table 5.4: Further experimental results of DBNs with sigmoid on classification accuracy (%).

Dataset	One hidden layer		Two hidden layers		Three hidden layers	
	DBN	DBN-DR	DBN	DBN-DR	DBN	DBN-DR
STS-T	61.12	65.26 *	61.01	67.68 *	60.59	66.87 *
STS-G	83.85	83.55	83.50	83.05	82.75	83.10
HCR	66.39*	64.39	64.91	65.61	65.17	65.52
HCR2	77.14	76.37	78.11 *	75.84	77.16	76.32
SST	54.58	54.62	53.40	52.85	53.22	54.00
SST2	72.58	71.82	70.00	71.55 *	69.09	72.91 *
FT	53.00	53.52	51.16	51.30	50.04	52.08 *
FT2	68.22	69.12	68.18	69.57 *	67.11	69.66 *
GT	57.18	58.95	57.69	58.43 *	57.11	58.36 *
GT2	75.73	76.24	75.39	75.75	75.37	75.61
average	66.98	67.38	66.34	67.16	65.76	**67.44**

Table 5.5: Summary of classification accuracy comparisons for DBNs with sigmoid.

Model	One hidden layer	Two hidden layers	Three hidden layers
DBN	1/1/8	1/4/5	0/5/5
DBN-DR	1/1/8	4/1/5	5/0/5

5.3 Chapter Summary

In this chapter, in order to further update the weights of unsupervised training for deep networks, we improve the ordinary DBN learning algorithm by introducing the DR based on the RBM unsupervised learning. Through the experiments on ten sentiment text data sets, the results demonstrate that DBNs incorporating the DR have better effect on the classification performance. This new learning algorithm avoids the correction signal to get the same dilute as the whole BP process in DBNs. In addition, each single-layer network in a deep network has more correction signal during the training with the DR, so it is concluded that the deeper the network is, the more accurate this new learning algorithm will be.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This dissertation starts from a plethora of surveys regarding sentiment analysis and deep learning, and then identifies some problems regarding the DBNs for sentiment analysis from various aspects, such as neuron model, positional information and learning algorithm. These three aspects are presented with detailed explanations and also demonstrated via a set of experiments. Here, the corresponding conclusions are reviewed as below.

To begin with, through investigation on the structure of DBNs, a key internal component - neuron model acts as a significant role for DBNs. In particular, we assume that each problem (e.g. sentiment classification, or image classification) should have its own activation function since activation is actually

indicating at what point to activate the (hidden) neuron. So we propose Gaussian neuron in DBNs for Twitter sentiment prediction because it offers good flexibility to adjust neuron model curves within DBNs corresponding with specific applications. Also Our experimental results demonstrate that it works better than sigmoid neuron in DBNs for Twitter sentiment prediction. Meanwhile, a set of experimental results in Chapter 3 help to prove that Gaussian neuron works well in DBNs for text sentiment prediction, especially on imbalanced data sets.

In addition, an external component (independent of the network) for problem is studied. In detail, the input for NLP problems is normally represented by bag-of-words representations or word-vector matrices, while each word in the sentence (or document) has its own context environment. Positional information is presented in this dissertation to add onto the traditional bag-of-words representations. We not only propose a new positional contribution form (linear increasingly positional weight), but also come up with a new word-to-segment matrix representation (divisions of sentences to express the order among segments) to generate new representations. The classification performance and sensitivity analysis with DBNs incorporating positional information are discussed in Chapter 4, which demonstrate that there really exist some useful representations to express the texts' positional information and they can be efficient added on the traditional bag-of-words representations, especially for multi-class sentiment classification problems.

Lastly, after discussing the internal and external components of DBNs for sentiment analysis, our attention is paid to the learning mechanism of DBNs. Normally, DBNs are trained with the two-phase algorithm (unsupervised RBM training and supervised BP training). Based on the ordinary DBN learning, a new learning method is proposed to improve it with the introduction of the DR in Chapter 5. The DR learning is actually like another phase inserted between the two-phase learning algorithm. With the experiments in text sentiment classification task, the results indicate that the DR phase helps to keep more correctional signal compared to the same DBN structure, and also it proves that DBNs with the DR work better for deeper architectures.

6.2 Future Work

As summarized above, we have reached some statistically significant accomplishments. However, during the process, there are also some other work to do for future research directions.

Firstly, we give some suggestions regarding how to choose the activation function in DBNs for Twitter sentiment prediction, preferring the activation function with the positive mean value and relative large standard deviation value (compared to sigmoid function). However, in the Gaussian DBNs, the form of Gaussian neuron (mean and standard deviation) is manually set in

the experiments. So, in the future, it can be set to learn during the training which will best fit a specific application. Moreover, some more sophisticated neuron functions or mixture of cumulative functions could have better effect.

Secondly, the work for positional information incorporated into DBNs can be improved in the following ways: (1) For the positional representation, the linear positional transformations implemented in Chapter 4 seem a little simple, and some more sophisticated curve functions (e.g. logit function or symmetrical function) probably perform relatively better; (2) Try to extend the matrix representation into a row vector for each sentence (even though it will be an extremely large sparse matrix and cost much memory during the training, this is an approach), letting each element value in the matrix be a single feature into the model; (3) Only three segments are introduced in Chapter 4, more segments (e.g. four or more segments) for a sentence (or long text) may be more reasonable, or some other segmentation techniques (e.g. segmentation in syntax) will probably account more word order information into the model; (4) Some other different segment impact relationships can be tried as well. These will be some of our future research directions.

Finally, for the learning of DBNs via the DR can be further investigated from the following aspects: (1) Three types of networks are implemented in this paper, but some more types of networks can be performed to further verify the results; (2) To test if this learning algorithm is robust, some other classification problems like image or video classification (not only sentiment

classification) should also be a new area; and (3) The DR learning in a single-layer network is actually like a linear-transformable problem, so some other linear classification algorithms can be tried to test whether it works better, e.g. multi-class logistic regression classification.

Bibliography

[1] Gupta V. Kaur A., *A survey on sentiment analysis and opinion mining techniques*, J.Emerg.Technol.Web Intell.Journal of Emerging Technologies in Web Intelligence **5** (2013), no. 4, 367–371 (English), ID: 5509415296.

[2] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau, *Sentiment analysis of twitter data*, Proceedings of the workshop on languages in social media, Association for Computational Linguistics, 2011, pp. 30–38.

[3] Y. Bengio, A. Courville, and P. Vincent, *Representation learning: A review and new perspectives*, Pattern Analysis and Machine Intelligence, IEEE Transactions on **35** (2013), no. 8, 1798–1828, ID: 1.

[4] Yoshua Bengio, *Learning deep architectures for ai*, Foundations and trends in Machine Learning **2** (2009), no. 1, 1–127.

[5] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle,

Greedy layer-wise training of deep networks, Advances in neural information processing systems **19** (2007), 153.

[6] Christopher M Bishop, *The pattern recognition and machine learning*, (2007).

[7] Miguel A Carreira-Perpinan and Geoffrey Hinton, *On contrastive divergence learning.*, AISTATS, vol. 10, Citeseer, 2005, pp. 33–40.

[8] ABC: Always Be Coding, *The men who stare at codes @ONLINE*, March 2014.

[9] Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick, and Dhruv Batra, *Reducing overfitting in deep networks by decorrelating representations*, arXiv preprint arXiv:1511.06068 (2015).

[10] Ronan Collobert and Jason Weston, *A unified architecture for natural language processing: Deep neural networks with multitask learning*, Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 160–167.

[11] Xiaowen Ding, Bing Liu, and Philip S Yu, *A holistic lexicon-based approach to opinion mining*, Proceedings of the 2008 international conference on web search and data mining, ACM, 2008, pp. 231–240.

[12] Jeffrey L Elman, *Finding structure in time*, Cognitive science **14** (1990), no. 2, 179–211.

[13] Murthy Ganapathibhotla and Bing Liu, *Mining opinions in comparative sentences*, Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1, Association for Computational Linguistics, 2008, pp. 241–248.

[14] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber, *Learning precise timing with lstm recurrent networks*, Journal of machine learning research **3** (2002), no. Aug, 115–143.

[15] Shalini Ghosh, Oriol Vinyals, Brian Strope, Scott Roy, Tom Dean, and Larry Heck, *Contextual lstm (clstm) models for large scale nlp tasks*, arXiv preprint arXiv:1602.06291 (2016).

[16] Xavier Glorot, Antoine Bordes, and Yoshua Bengio, *Domain adaptation for large-scale sentiment classification: A deep learning approach*, Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 513–520.

[17] Bengio Y. 14th International Conference on Artificial Intelligence Glorot X., Bordes A. and AISTATS 2011 Statistics, *Deep sparse rectifier neural networks*, J.Mach.Learn.Res.Journal of Machine Learning Research **15** (2011), 315–323 (English), ID: 796818411.

[18] Alec Go, Richa Bhayani, and Lei Huang, *Twitter sentiment classification using distant supervision*, CS224N Project Report, Stanford **1** (2009), 12.

[19] Christoph Goller and Andreas Kuchler, *Learning task-dependent distributed representations by backpropagation through structure*, Neural Networks, 1996., IEEE International Conference on, vol. 1, IEEE, 1996, pp. 347–352.

[20] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten, *The weka data mining software: an update*, ACM SIGKDD explorations newsletter **11** (2009), no. 1, 10–18.

[21] Geoffrey Hinton, *A practical guide to training restricted boltzmann machines*, Momentum **9** (2010), no. 1, 926.

[22] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh, *A fast learning algorithm for deep belief nets*, Neural computation **18** (2006), no. 7, 1527–1554.

[23] Geoffrey E. Hinton, *Training products of experts by minimizing contrastive divergence*, Neural computation **14** (2002), no. 8, 1771–1800.

[24] Geoffrey E Hinton and Ruslan R Salakhutdinov, *Reducing the dimensionality of data with neural networks*, Science **313** (2006), no. 5786, 504–507.

[25] Sepp Hochreiter and Jürgen Schmidhuber, *Long short-term memory*, Neural computation **9** (1997), no. 8, 1735–1780.

[26] Minqing Hu and Bing Liu, *Mining and summarizing customer reviews*, Proceedings of the tenth ACM SIGKDD international conference on

Knowledge discovery and data mining, ACM, 2004, pp. 168–177.

[27] X. Hu, J. Tang, H. Gao, and H. Liu, *Unsupervised sentiment analysis with emotional signals*, WWW 2013 - Proceedings of the 22nd International Conference on World Wide Web, 2013, pp. 607–617.

[28] Nitin Jindal and Bing Liu, *Mining comparative sentences and relations*, 2006.

[29] R. Johnson and T. Zhang, *Effective use of word order for text categorization with convolutional neural networks*, NAACL HLT 2015 - 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 2015, pp. 103–112.

[30] Yann LeCun, *Deep learning tutorial*, Citeseer.

[31] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, *Deep learning*, Nature **521** (2015), no. 7553, 436–444.

[32] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation **1** (1989), no. 4, 541–551.

[33] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE **86** (1998), no. 11, 2278–2324.

[34] Honglak Lee, *Tutorial on deep learning and applications*, Citeseer.

[35] Jiwei Li, Minh-Thang Luong, and Dan Jurafsky, *A hierarchical neural autoencoder for paragraphs and documents*, arXiv preprint arXiv:1506.01057 (2015).

[36] Payton Lin, Szu-Wei Fu, Syu-Siang Wang, Ying-Hui Lai, and Yu Tsao, *Maximum entropy learning with deep belief networks*, Entropy **18** (2016), no. 7, 251.

[37] Bing Liu, *Sentiment analysis and subjectivity.*

[38] Bing Liu, *Sentiment analysis and opinion mining*, Synthesis lectures on human language technologies **5** (2012), no. 1, 1–167.

[39] Bing Liu, Minqing Hu, and Junsheng Cheng, *Opinion observer: analyzing and comparing opinions on the web*, Proceedings of the 14th international conference on World Wide Web, ACM, 2005, pp. 342–351.

[40] Kun-Lin Liu, Wu-Jun Li, and Minyi Guo, *Emoticon smoothed language models for twitter sentiment analysis.*, AAAI, 2012.

[41] Yang Liu, Chengjie Sun, Lei Lin, and Xiaolong Wang, *Learning natural language inference using bidirectional lstm model and inner-attention*, arXiv preprint arXiv:1605.09090 (2016).

[42] Andrew McCallum, Kamal Nigam, et al., *A comparison of event models for naive bayes text classification*, Citeseer.

[43] Tomas Mikolov, *Recurrent neural network based language model.*

[44] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, *Efficient estimation of word representations in vector space*, arXiv preprint arXiv:1301.3781 (2013).

[45] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur, *Extensions of recurrent neural network language model*, 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2011, pp. 5528–5531.

[46] Tom Mikolov, *Statistical language models based on neural networks*, Presentation at Google, Mountain View, 2nd April (2012).

[47] Vinod Nair and Geoffrey E. Hinton, *Rectified linear units improve restricted boltzmann machines*, Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010, pp. 807–814.

[48] Radford M. Neal, *Connectionist learning of belief networks*, Artificial Intelligence **56** (1992), no. 1, 71–113.

[49] Steven J Nowlan and Geoffrey E Hinton, *Simplifying neural networks by soft weight-sharing*, Neural computation **4** (1992), no. 4, 473–493.

[50] David L. Olson and Dursun Delen, *Advanced data mining techniques*, Springer Science and Business Media, 2008.

[51] T. Pahikkala, S. Pyysalo, J. Boberg, J. Jrvinen, and T. Salakoski, *Matrix*

representations, linear transformations, and kernels for disambiguation in natural language, Machine Learning **74** (2009), no. 2, 133–158.

[52] Bo Pang and Lillian Lee, *A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts*, Proceedings of the 42nd annual meeting on Association for Computational Linguistics, Association for Computational Linguistics, 2004, p. 271.

[53] Bo Pang and Lillian Lee, *Opinion mining and sentiment analysis*, Foundations and trends in information retrieval **2** (2008), no. 1-2, 1–135.

[54] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan, *Thumbs up?: sentiment classification using machine learning techniques*, Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10, Association for Computational Linguistics, 2002, pp. 79–86.

[55] R. Paulus, R. Socher, and C. D. Manning, *Global belief recursive neural networks*, Advances in Neural Information Processing Systems, vol. 4, 2014, pp. 2888–2896.

[56] Jay M. Ponte and W. Bruce Croft, *A language modeling approach to information retrieval*, Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, ACM, 1998, pp. 275–281.

[57] Ana-Maria Popescu and Orena Etzioni, *Extracting product features and*

opinions from reviews, Natural language processing and text mining, Springer, 2007, pp. 9–28.

[58] Veselin Raychev and Preslav Nakov, *Language-independent sentiment analysis using subjectivity and positional information.*, RANLP, 2009, pp. 360–364.

[59] Patrawut Ruangkanokmas, Tiranee Achalakul, and Khajonpong Akkarajitsakul, *Deep belief networks with feature selection for sentiment classification*, 7th International Conference on Intelligent Systems, Modelling and Simulation, 2016.

[60] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, *Learning representations by back-propagating errors*, Cognitive modeling **5** (1988).

[61] Hassan Saif, Miriam Fernandez, Yulan He, and Harith Alani, *Evaluation datasets for twitter sentiment analysis: a survey and a new dataset, the sts-gold*, (2013).

[62] R. Sarikaya, G. E. Hinton, and A. Deoras, *Application of deep belief networks for natural language understanding*, IEEE Transactions on Audio, Speech and Language Processing **22** (2014), no. 4, 778–784 (English).

[63] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, *Semi-supervised recursive autoencoders for predicting sentiment distri-*

butions, EMNLP 2011 - Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference, 2011, pp. 151–161.

[64] Richard Socher, Recursive Deep Learning for Natural Language Processing and Computer Vision (2014).

[65] Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng, *Semantic compositionality through recursive matrix-vector spaces*, Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Association for Computational Linguistics, 2012, pp. 1201–1211.

[66] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng, *Parsing natural scenes and natural language with recursive neural networks*, Proceedings of the 28th international conference on machine learning (ICML-11), 2011, pp. 129–136.

[67] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts, *Recursive deep models for semantic compositionality over a sentiment treebank*, Proceedings of the conference on empirical methods in natural language processing (EMNLP), vol. 1631, Citeseer, 2013, p. 1642.

[68] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting.*, Journal of Machine Learning Research **15**

(2014), no. 1, 1929–1958.

[69] Gregory O. Stone, *An analysis of the delta rule and the learning of statistical associations*, Parallel distributed processing: explorations in the microstructure of cognition, vol. 1, MIT Press, 1986, pp. 444–459.

[70] D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, and B. Qin, *Learning sentiment-specific word embedding for twitter sentiment classification*, 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference, vol. 1, 2014, pp. 1555–1565.

[71] Duyu Tang, Furu Wei, Bing Qin, Ting Liu, and Ming Zhou, *Coooolll: A deep learning system for twitter sentiment classification*, SemEval 2014 (2014), 208.

[72] Tijmen Tieleman, *Training restricted boltzmann machines using approximations to the likelihood gradient*, Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 1064–1071.

[73] Subhashini Venugopalan, Lisa Anne Hendricks, Raymond Mooney, and Kate Saenko, *Improving lstm-based video description with linguistic knowledge mined from text*, arXiv preprint arXiv:1604.01729 (2016).

[74] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol, *Extracting and composing robust features with denoising autoencoders*, Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 1096–1103.

[75] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus, *Regularization of neural networks using dropconnect*, Proceedings of the 30th International Conference on Machine Learning (ICML-13), 2013, pp. 1058–1066.

[76] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young, *Semantically conditioned lstm-based natural language generation for spoken dialogue systems*, arXiv preprint arXiv:1508.01745 (2015).

[77] Wikipedia, *Convolutional neural network — wikipedia, the free encyclopedia*, 2016, [Online; accessed 8-October-2016].

[78] ———, *Delta rule — wikipedia, the free encyclopedia*, 2016, [Online; accessed 11-September-2016].

[79] Ian H Witten and Eibe Frank, *Data mining: Practical machine learning tools and techniques*, Morgan Kaufmann, 2005.

[80] Alan L. Yuille, *The convergence of contrastive divergences*, Department of Statistics, UCLA (2006).

[81] Chengxiang Zhai and John Lafferty, *A study of smoothing methods for language models applied to information retrieval*, ACM Transactions on Information Systems (TOIS) **22** (2004), no. 2, 179–214.

[82] Shusen Zhou, Qingcai Chen, Xiaolong Wang, and Xiaoling Li, *Hybrid deep belief networks for semi-supervised sentiment classification*, COL-

ING, 2014, pp. 1341–1349.

Vita

Candidate's full name: Yong Jin
University attended (with dates and degrees obtained):
2012-2013 PhD candidate in Math & Stats, University of New Brunswick
(transferred into the PhD CS program effective Jan.1, 2013)
2013-2017 PhD candidate in Computer Science, University of New Brunswick
2009-2012 MS in Statistical Theory & Methods, Wuhan University of Technology
2005-2009 BS in Statistics, Wuhan University of Technology

Publications:

Yong Jin, Harry Zhang, Donglei Du, Improving Deep Belief Networks via Delta Rule for Sentiment Classification, *Proceedings on the 28th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2016.

Yunli Wang, Yong Jin, Xiaodan Zhu and Cyril Goutte, Extracting Discriminative Keyphrases with Learned Semantic Hierarchies, *Proceedings on the 26th International Conference on Computational Linguistics (COLING)*, 2016.

Yong Jin, Donglei Du, Harry Zhang, Gaussian Neuron in Deep Belief Networks for Sentiment Prediction, *Proceedings on the 29th Canadian Conference on Artificial Intelligence*, 2016.

Yong Jin, Harry Zhang, Donglei Du, Incorporating Positional Information into Deep Belief Networks for Sentiment Classification, *Proceedings on the*

17th Industrial Conference on Data Mining(ICDM 2017).

Yonghong Wu, Qiong Li, Yong Jin, European Option Pricing on Foreign Exchanges under Stochastic Interest Rates, *Journal of Wuhan University of Technology (Transportation Science & Engineering)*, 2010 (published in China, Chinese version).

Yong Jin, Zhongjun Wang, Study on Social Status of Chinese Enterprises' Scientific and Technological Workers, *FINANCE & ECONOMY*, 08(73-74), 2008 (published in China, Chinese version).

Conference Presentations:

Improving Deep Belief Networks via Delta Rule for Sentiment Classification, Presented at the 28th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), November 6-8, 2016, San Jose, California, USA.

Gaussian Neuron in Deep Belief Network for Sentiment Prediction, presented at the 29th Canadian Conference on Artificial Intelligence, May 31-June 3, 2016, Victoria, BC, Canada.