DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Deep Learning for Sentiment Analysis

Ankit Bahuguna

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Deep Learning for Sentiment Analysis

Deep Learning Methoden für Sentiment Analysis Probleme

Author:	Ankit Bahuguna
Supervisor:	Prof. Georg Groh
Advisor:	Prof. Georg Groh
Submission Date:	September 15, 2015

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, Germany, Ankit Bahuguna
September 15, 2015

Acknowledgments

I would like to wholeheartedly thank my supervisor and adviser **Mr. Georg Groh**, who advised my Master's Thesis at TUM and encouraged me the whole time. He played a pivotal role in me continuing the study in the area of my interest and help me chase the state of the art systems with utmost frivolity. I would also thank, **Prof. Hinrich Schütze**, with whom, I started this study during an interdisciplinary project at Ludwig Maximillian's University (München) in April 2014 . His supervision and guidance helped me to get started with the new area of Deep Learning and its application to problems in Natural Language Processing. Additionally, I would thank my previous guide, **Prof. Pushpak Bhattacharrya**,(Indian Institute of Technology, Bombay) who introduced the field of Natural Language Processing and helped kick-start my interest in the area and encouraged me continuously to aim higher and helped me open my mind to look at problems from various perspectives.

In the end, I would also like to thank my family, who gave me this opportunity to study at Technical University of Munich, Germany. Without them, nothing of this would be possible.

Abstract

Word vector representations or word embeddings, learned using deep neural network models prove extremely useful at capturing semantically similar words and phrases. Recent work in the area, utilize semantic word vector spaces and sentiment compositionality to classify sentiment better. One of the biggest constraints with these methods, is need of manually labeled sentiment for phrases in parse trees of sentences making its use challenging for languages with constrained resources.

Moreover, traditional methods do not efficiently leverage the use of semantic word vector spaces and word context in overall sentiment classification. In this report, we first present a broad overview of the problem, discuss traditional methods and then introduce Deep Learning based methods for Natural Language Processing which help in understanding the problem of Sentiment Analysis, better than traditional methods. Also presented is the analysis of concatenating various word vector representations which do not necessarily require phrase level labeling, comparisons of their results in various experimental configurations and showing improvement over traditional unsupervised baseline methods and state of the art results. In the same experiment, we also explore in-domain, out-of-domain and mixed-domain data towards training these word vectors and show how each one affects sentiment classification.

We also present promising future work, which may lead to better deep neural network based word representations for various NLP tasks.

Contents

1 Introduction

1.1 World of Opinions and Sentiment

"*What other people think*", has been a very important element in the overall decision making process. This is more relevant in the modern era with the explosion of web and internet based services where the people have a new platform where they can voice their opinions and discuss about day to day things. With the advent of this platform, the subjective information too has grown extremely rapidly over the years.

It is interesting to note that social opinion of users impact a large corpus of audience. A simple case example is, that if a product is negatively perceived in the social circles of the internet, it will not be taken positively by a new user who is interested in purchasing the product. Hence, /this also is a tool for organizations to understand how the product is performing in the public. This, has given rise to a new area of study known as social media monitoring and analysis.

In the modern day, organizations have made significant investments to process this vast amount of data and make sense of the important information contained in it, so that they can leverage their products and services and improve their overall user experience or increase chances of making more profit, by identifying key areas which are causing an indirect impact on the overall profit margin. In the modern world, the organizations which makes the best use of data is the clear winner and it also is an opportunity for the researchers working in the area to aggressively pursue new techniques and algorithms which beat the current state of the art systems, giving the organizations slight edge over their competitors, which can in turn bring millions of dollars in revenue.

1.2 Sentiment Analysis: An Introduction

It's well said that the fundamental difference, between a human and a computer is that, the computer doesn't perceive or express emotions. If someday, this barrier can be overcome, then it will be very hard to distinguish a human conversation from a machine conversation. Thus, the broad goal for the study of Sentiment Analysis, is making computers recognize and express emotions.

First, we present some of the common terms, which are regularly used in context of Sentiment Analysis, viz., **opinion**, **sentiment** and **subjectivity** in text. These can be defined in the following manner:

opinion : *A view or judgment formed about something, not necessarily based on fact or knowledge.*

sentiment : *A view or opinion that is held or expressed.*

subjectivity : *Refers to how someone's judgment is shaped by personal opinions and feelings instead of outside influences.*

The term opinion mining appears in a paper by Dave et al. [11]. Interestingly, an ideal opinion-mining tool would "process a set of search results for a given item, generating a list of product attributes (quality, features, etc.) and aggregating opinions about each of them (poor, mixed, good)."

The history of the phrase sentiment analysis parallels that of "opinion mining" in certain respects. The term "sentiment" used in reference to the automatic analysis of evaluative text and tracking of the predictive judgments therein appears in 2001 papers by Das and Chen [10] and Tong [41], due to these authors' interest in analyzing market sentiment. It subsequently occurred within 2002 papers by Turney [42] and Pang et al. [29], which were published in the proceedings of the annual meeting of the Association for Computational Linguistics (ACL) and the annual conference on Empirical Methods in Natural Language Processing (EMNLP). These papers really increased the popularity of the term "Sentiment Analysis" in the Natural Language Processing research community. A number of papers mentioning "sentiment analysis" focus on the specific application of classifying reviews as to their polarity (either positive or negative), a fact that appears to have caused some authors to suggest that the phrase refers specifically to this narrowly defined task. However, nowadays many use the term more broadly to mean the computational treatment of opinion, sentiment, and subjectivity in text. A famous and more comprehensive account on this subject was presented by Bo Pang and Lillian Lee[**pangleesentiment**] [29] which covered the various techniques and approaches that promised opinion oriented information seeking systems. This work accounted a full fledged review, which this current literature also follows in parts in the beginning two chapters.

If we look in a broad view, the terms sentiment analysis and opinion mining roughly imply the same field of study, which itself can be attributed as the sub field of subjectivity analysis. For the sake of simplicity, we will stick to "sentiment analysis" throughout this work.

1.3 Applications

There are a number of application areas of this study, although, in the introduction we have mentioned reviews related to websites and products but in general there are many other possibilities. It is because of all the possible applications, there are a good number of small startups to large organizations which have dedicated teams who look into this subject matter and have it as part of their mission statement.

1.3.1 A Review based Search Engine

A classic example can be a review oriented search engine. Consider a product search engine which also utilizes the reviews on the products made. And ranks the result based on the positive polarity of the products. Thus, the products which has maximum positive product reviews are ranked higher than the ones which receives lower reviews. This will help a client who is user of the search engine to have high confidence about the kind of products returned. Such a search engine will in a way give an assurance to the customer that the best reviewed product which are socially favorable and appreciated are shown first to the user, hence making it easy to make the final choice of eventually purchasing the product.

1.3.2 Websites hosting Reviews

There is a growing number of opinion hosting websites, for example, epinions.com; along with the booming e-commerce industry which showcase products and offers, there are millions of reviews generated every week. Thus, they are a primary playground for sentiment analysis. These can be a good benchmark for the companies to evaluate how the product is performing in the market. By performing a comprehensive analysis of the reviews which are hosted on these websites, the companies get a fair idea about the pros and cons about their product and also about the competitor, and a general sentiment about the public perception can help organizations make important decisions about how to go or not go about the future iterations of the projects. Some organizations, perform a real time analysis, so that if a feature is causing a universal cry, resources can be quickly put together into immediately fixing the issue and thus leveraging overall user experience, leading to a longer user loyalty retention.

1.3.3 Sub-System Technology

Sentiment analysis can also be used a sub-system leveraging the working of another intelligent computational system. One possibility is the augmentation to a *recommendation system*. Thus, a recommendation system will utilize a sentiment analysis system, which can then not recommend products which receive a lot of negative feedback.

Detection of overheated or agnostic language in emails or any other form of communication is another use of subjectivity detection and classification.

In *online systems and display advertisements*, where its required to display advertisements relevant to the content of the web-page and some may contain sensitive material, it always better is such sensitive websites can be recognized in advance. A more sophisticated system can display an advertisement when more positive statements in content are discovered and nix the same when negative statements begin to appear.

Also in *information extraction* which is heavily based on objectivity in text, can leverage subjectivity detection and discard any subjectivity content, thus improving overall quality of information extracted.

Question Answering systems can also be improved, as the opinion oriented questions can be handled differently. These questions differ in the subjective sense and a pre-analysis of the same can be useful to generate better answers. Similarly, *summarization* can also be made useful in the context of the multiple views points.

In the area of citation analysis, where potentially, one can detect whether the citation done by the researcher in his scientific work is a support or a rebuttal of the cited work.

1.3.4 Business and Government Intelligence

Sentiment Analysis is used effectively as a business intelligence tools. The reviews about the product can be indicative directly what is right or wrong about the product or it may be useful to mine the reviews from all the general domains like opinion websites,blogs, public forms and product pages and summarize each f the review entities corresponding to each product thereby giving a consensus of all the key points relevant in the eyes of the users with respect to the product. The data can also determine what are the current trends in the given business category and can be used to leverage the key insights which in-turn can benefit sales.

With a view point of government Intelligence, such a tool can be used to monitor hostile behavior in a communication. This form of government intelligence can help in also beefing up security if a known threat is determined in advance.

1.3.5 Inter-Domain Applications

Sentiment Analysis has not only been cherished by computer and data scientists but also in the political circles of the government. It has actually turned out to be a golden tool in political elections. Knowing what the people think about the candidates in a general election can allow a party to choose the right candidate and also work on its shortcomings in due time, to win the ever significant final elections.

Moreover, There are now e-portals which are facilitating *eRuleMaking*, where a new policy is first open to the public and then based on the various opinions of the general public, the decision to move forward and cement the proposed rules into a law or updating its various sections is taken into consideration. This has proved to be an effective tools to iron out any problems which may exist in a proposed law even before its presented to the democratic authority. Thus, saving both time and resources.

1.4 General Challenges

Often a simple question comes to mind, when discussing Sentiment Analysis: *How is it different from classic text mining and fact based analysis?*

Traditionally, text categorization involves categorizing text into categories. These categories can be topics which are either predefined or looked up as more data is processed. These categories can also be user or application specific. And, for a given task, we might be dealing with wither a two class or binary classification or a multi-class classification (say, among thousands of classes). In contrast, with sentiment analysis, we are usually focused with 3 classes: positive, negative or neutral (coarse grained) or, 5 classes: positive, slightly positive, negative, slightly negative and neutral (fine grained). Thus, the number of classification classes are fairly limited and it generalizes well to many information domains and users. moreover the topics or categories can sometimes be highly non-correlated whereas the sentiments classes are often opposite in nature (in case of binary: positive- negative classification.)

Also, there is fundamentally a difference when it comes to answering opinion oriented questions vs fact based questions. The traditional information extraction methods which work well for a number of facts based templates. An opinion oriented information extraction method too will be a generalized version of the traditional system, as they are focused on similar fields of an opinion expression (holder, type and strength) irrespective of the topic. Although, it may seem the propositions presented above may make the task of sentiment analysis relatively easy, but its far from the actual truth. We will learn next about the difficulties in developing sentiment analysis systems, as compared to traditional fact based text analysis systems.

1.4.1 Difficulty in Sentiment Classification?

First, we start with a simple scenario and a connected question: Given, we have an opinionated sentence in some given language, our task is to classify it into one of positive, negative or neutral classes. How hard is this overall task? Let's start with some examples:

1. It was a great restaurant.
2. It should have been a great restaurant.
3. The restaurant was great in that it will make all future meals seem more delicious.
4. Despite a pleasant experience I can't support the many reviews that it was a great restaurant.

First sentence is a positive sentence, implied by "great restaurant", The rest of the sentences are all negative. But there exists a varying degree of negativeness among them. The second sentence with the phrase "should have been" implies the desired result. Which suggests that the restaurant was expected to be better than it actually was. The third sentence is a classic case of saracasm. This is hardest to spot even for humans; On the first glance we might be mistaken to consider this as positive. Sentence 4, started with a positive vibe "pleasant experience", but the reviewer then retracts his support to all the positive reviews made about the restaurants by others, thus making his overall review negative.

These four sentences just provide a glimpse to how hard is to analyze sentiment. Moreover, apart from detecting polarity in subjective sentences, there is something, which is known as beyond polarity analysis, where the task is to detect more expressive features like anger, happiness, sadness and relaxation in text, these too are very hard to predict and are highly context driven. The context may be based on the expression in the sentence or say, in a conversation among two individuals, their relationships between them. For example, the sentence "Your're a Liar!", may mean either positive or negative given the two individuals and their relationships. Like, a candid conversation between a couple, one may take this in positive sense (not always!), but among politicians it certainly means seems negative.

Then there is also a problem of **entity level sentiment** and **domain specificity**. The sentiment of a reviewer of a product may be dependent on individual entities composing a product. Say, for example: A user is writing a review about a **digital camera**:

"I consider this is as a decent looking camera. The lens is of high quality and performs well in low light condition. Although, the flash is timely, it just seems to fade away the colors. There is also the issue with the battery which only takes about 100-150 photos on full charge, making periodic recharging necessary. Last, the price is the reason, I won't recommend this camera to anyone, because there are other better cameras for the same specs and lower price which are better buy in the category. "

In the above review, the user is highly positive about the looks and lens of the camera. Thus, the review begins on a positive note and then the user comments about the flash, battery and the price, eventually making the review a negative one. This example is essentially useful in understanding the complexity involved in extracting the overall sentiment of a long review. Such long reviews are common in the real world and thus, this task is harder than it seems. Often, in such cases, it's best to understand what entity or sentence contributes the most to overall sentiment. Although, this is easier said than done!

One may, also find that the specific qualities of the entities, which seem good for a certain product may not be good for another. This is where domain specificity is required. For example,

- "**unpredictable**" may be negative in a car review, but positive in a movie review [42]
- "**cheap**" may be positive in a travel/lodging review, but negative in a toys review

In this current work, we are more concerned about determining the orientation of an opinionated text. We assume that the sentences which we analyze are subjective in nature and express opinions. If, that were not the case, we wold have to first filter out opinion-centric sentences from the non - opinionated or objective sentences. This type of analysis is more specifically subjectivity analysis and goes beyond the scope of this work. In simple terms, sentiment classification is a sub-field of subjectivity analysis.

Next, we dig deeper into the problem and discuss some relevant methods which have been developed over the years to solve problems related to sentiment classification.

2 Sentiment Analysis

Till now we have studied about Sentiment Analysis with a general perspective. This chapter starts a more computational view of solving this problem. Before we begin discussing various methods, its important to understand the scope of the problem.

2.1 Sentiment Analysis Levels

The sentiment analysis problem is not a single view problem, rather its can be viewed as multi-level problem. They are discussed as follows:

- **Document Level:** Identifying if the *document* (product review, blog-post, forum post etc.) expresses opinions and if they do, classifying them as positive, negative or neutral. this takes into account the overall sentiment expressed by the opinion holder for the whole document.
- **Sentence Level:** Identifying whether a single *sentence* is positive, negative or neutral.
- **Attribute Level:** Extracting the *object attributes* (image quality, zoom size etc.) that are subject of an opinion and the opinion orientation.

  It is important to note here that as the object becomes more granular, the intensity or difficulty of sentiment analysis increases.

2.2 Document-Level Sentiment Analysis

We start with the traditional methods for document analysis sentiment analysis. It's important to keep in mind few assumptions which come along the way.

- The document is opinionated on a single object
- the opinions are from a single opinion holder.

In real world data, these assumptions don't usually hold. But, for the sake of simplicity, we start with them to make the understanding of the methods easy.

Also, more importantly, this analysis is slightly different from the topic based text classification. In topic classification, heavy emphasis is given to the extraction of the topics. Thus, the *topic words* become very important. Whereas, in sentiment classification, the opinion words are more important. For example: wonderful, fabulous and terrible.

So to start with, a naive sentiment classifier can be based on identification and processing of these opinion words. Opinionated words are also known as sentiment words, opinion lexicon, polarity words or opinion bearing words. They are categorized into two types. First, **Base types**, for example:
Positive: wonderful, elegant, amazing
Negative: horrible disgusting poor
Second, Comparative Types, for example: better, worse etc.

2.2.1 Counting Opinion Words

We start with a simple method. We try to count the opinion words in a given document. We are given an opinion word list which have both positive and negative words. And we assign an **orientation score** (-1,1) to all words in this list. Such that,

Positive opinion words (+1): great, amazing, love ...
Negative opinion words (-1): hate, horrible, etc ...
Strength of these words can also be defined [0,1]

Thus the orientation score for the document, is the sum of the orientation scores of all opinion words found. Lets, take an example review and count the opinion words in the same and calculate overall orientation score for that review. Example:
"My Canon Powershot takes **great** pictures! ... My friend had gotten one about a year ago and she **loves** it. So, after seeing her enthusiasm about it I decided to get one and I will never go back to any other camera. I absolutely **love** this camera. I believe that every person on Earth should own one of these. ... It is **amazing**! ... There is not one thing I **hate** about this product, which is strange because I am a very picky person! ..."
Positive Sentiment Words: great (1), love (2) and amazing (1); Total: 4
Negative Sentiment Words: hate (1); Total: 1

Overall Sentiment Orientation for Review = 4-1 = 3 or **Positive Review Overall**!
After looking at this analysis, lets ask ourselves a question. **Is simply counting the opinion words good enough?**

The answer to that is **NO**! Lets' see why. Just take into consideration the last sentence in the above review.
There is **not** one thing I **hate** about this product. $\rightarrow$ Not Negative!

This is positive sentence overall. Since, we only captured our negative sentiment word **"hate"**, we labeled this sentence as negative, which is not true. Because of the negation indicator "not", the sentiment is reversed as in **not ... hate** $\Rightarrow$ **like**.. Thus, an important lesson here is that we need to handle negation. A common pattern of analysis is:

"not ... **negative**" $\rightarrow$ **positive**
"never ... **negative**" $\rightarrow$ **positive**

Thus, the overall orientation score of the review is 4 +1 = 5 and not 3 as stated previously.

A word of caution: Negation must be handled very carefully! As, "not" in the pattern: "not only ... but also" does not change the orientation.

2.2.2 Rule Based Method

Another method to compute the overall sentiment of a document is through handcrafted rules which are provided by a trained linguist, based on the identification of linguistic phenomenon which determine sentiment for a given language. We first discuss some related terminology:

- **Pattern / Rule:** A sequence of tokens.
- **Token:** An abstraction of a word, represented using lemma, polarity tag or a part of speech tag. There are two special tokens. TOPIC: an attribute (ex. weight, size etc.) and GAP_DIGIT_DIGIT: How many words can be skipping between two tokens to allow more tolerant matching. Ex GAP_1_2
- **Polarity Tag:** positive, negative, neutral, NOT (negation)
- **POS Tag:** NN (Noun), VB (Verb), JJ (Adjective), RB (Adverb), IN (preposition) ...

Examples of some basic opinion rules, Label Sequential Pattern (LSP) Matching:

- Subject like | adore | want | work TOPIC $\rightarrow$ positive, e.g. – "I like the old camera"

- Subject is | are great | fantastic | simple | easy $\rightarrow$ positive, e.g. – "This camera is fantastic"
- TOPIC GAP_0_3 NOT work $\rightarrow$ negative, e.g. – "The new search still does not work"
- Please do NOT VB $\rightarrow$ negative, e.g. – "Please do not roll out this new search!"
- NOT GAP_0_3 want | think | believe | need | get $\rightarrow$ negative, e.g. – "I do not want large size pictures in the Gallery window."
- get | bring | give | put | change GAP_0_3 TOPIC GAP_0_3 back $\rightarrow$ positive – "Please put the old search and browse back!"

Although this method seems to work pretty well, but there is a fundamental limitation in putting it to use at large scale. Any rule based mining methods are computationally expensive. In this case, only a limited number of opinionated words are found and only a limited number of patterns can be created.

Note: There are methods, which are helpful in generation and classification of these sentiment words automatically, but we do not go deeper into the same. For the more curious, some methods which are used for this purpose are discussed in the works of Turney [42] which make use of PMI (Point-wise Mutual Information), Dictionary based methods (SentiWordnet) [13], Corpus based Methods (which rely on syntactic or co-occurrence patterns in large text corpora) etc.

TODO: Refer several related papers here!

2.2.3 Supervised Machine Learning Techniques

So, far we have seen simple count based and rule based methods. Although these methods are simple to comprehend and implement, they don't scale well for large corpora. So, to achieve an overall improvement, we make use of machine learning. What machine learning allows is to help find the patterns in known set of documents and then apply those patterns learned on new documents. For the case of supervise learning, we require training and testing datasets. And a set of features to represent documents.

In the case of sentiment analysis the final learning goal is to classify sentence into positive, negative or neutral. A very common way to try out this method is on product reviews dataset. In product reviews, we find that the users review by selecting starts [1,5] or thumbs up or down, where 5 or thumbs up means very good rating and 1 or thumbs down represents very low rating. This can be used his review as positive or

negative and use the same as training data. We can then perform sentiment analysis on those reviews which have not been specifically rated by the user.
An important step in the process is representing these reviews in such a way that its consistent and they are also understood by computers alike. This begins with something known as "Feature Extraction". Some general methods are discussed below:

- **Terms and Frequency**
  - Unigram and more general n-grams.
  - Word Position Information
  - Term Frequency - Inverse Document Frequency (TF-IDF) weighting.
- **Part of Speech Tags:** Adjectives are usually important indicators of subjectivity and opinions.
- **Syntactic Dependencies:** The syntactic structure of a sentence can be captured by representing the sentence as a **Syntax Parse Tree**. See Figure 2.1.

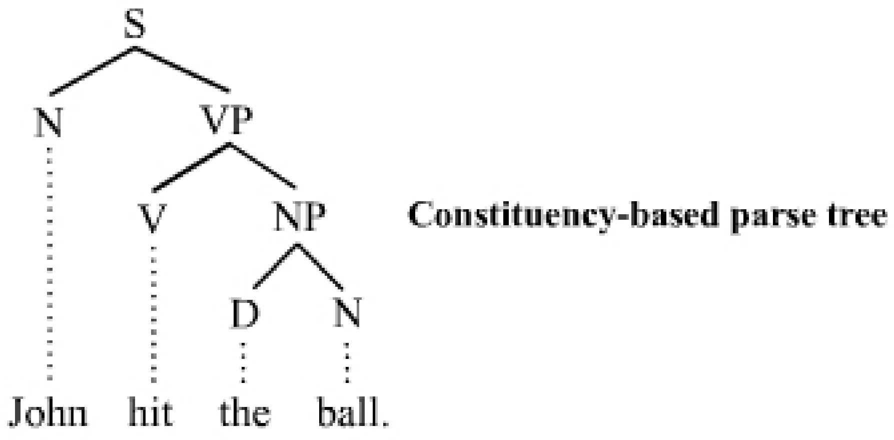


Figure 2.1: Parse Tree for sentence "John hit the ball."[1]

Once we have performed feature extraction, then the next step is to feed the feature input to the machine learning algorithm. There are a number of very successful and efficient supervised machine learning methods. They are described briefly as follows:

- **Naïve Bayes:** A simple probabilistic classifier based on applying the Bayes′ theorem with strong (naive) independence assumption.
- **Maximum Entropy (ME):** A probabilistic model that estimates the conditional distribution of the class label

- **Support Vector Machines (SVM)** [Pang et al, [29]] A representation of the examples as points in space in which support vectors are computed to provide a best division of points/examples into categories
- **Logistic Regression (LR) Model** [Pang & Lee,[30]] A LR model predicts the classes from a set of variables that may be continuous, discrete, or a mixture.

Since, the topics of this work are focused on exploring unsupervised word representations learned via deep learning and its application to the problem of sentiment analysis, we will not expand this discussion further.

2.2.4 Domain Dependency Problem and Domain Adaptation

When sentiment analysis is performed in real life, a common issue arises. A classifier trained using opinionated documents from domain A often performs poorly when tested on documents from domain B. There are genuine reasons for the same:

1. Reason 1: words used in different domains can be substantially different, e.g.
    - Cars vs. Movies
    - Cameras vs. Strollers
2. Reason 2: some words mean opposite in two domains, e.g.
    - "unpredictable" may be negative in a car review, but positive in a movie review [Turney, ACL 2002]
    - "cheap" may be positive in a travel/lodging review, but negative in a toys review

A common way to mitigate this issue is **Domain Adaptation**. It's actually a well studied problem. [Aue & Gamon [1]; Blitzer et al, [5]; Yang et al,[44]]. Some of the basic steps involved when perfroming domain adaptation is as follows:

1. Use labeled data from one domain and unlabeled data from both source the target domain and general opinion words as features
2. Choose a set of pivot features which occur frequently in both domains
3. Model correlations between the pivot features and all other features by training linear pivot predictors to predict occurrences of each pivot in the unlabeled data from both domains

Recently, Jochim and Schütze [20] showed improvement in citation polarity classification using product reviews, by making use of Domain Adaptation, their out-of domain data was Amazon's Product Review data-set.

2.3 Sentence-Level Sentiment Analysis

Document level sentiment analysis is too coarse for most applications,☺ or, ☹? Example:

"I bought a new X phone yesterday. The voice quality is **super** and I really **like** it. However, it is a little bit **heavy**. Plus, the key pad is **too soft** and it **doesn't feel comfortable**. I think the image quality is **good** enough but I am **not** sure about the battery life..."

- **Task:** Determine whether a sentence **s** is subjective or objective, and if **s** is subjective, determine whether its orientation is positive or negative
- **Assumptions:** The sentence is opinionated on a single object and the opinion is from a single opinion holder.

It is important to highlight that at sentence level, the syntax becomes of paramount importance. Hence we begin next with understanding syntactic patterns.

2.3.1 Syntactic Pattern Learning

In thier 2003 work, Riloff and Wiebe [34] presented a bootstrapping process that learns linguistically rich extraction patterns for subjective (opinionated) expressions. The proposed process can be explained briefly as follows:

1. Use high precision but low recall classifiers to automatically identify some subjective and objective sentences.
   - A subjective classifier: the sentence contains two or more strong subjective clues
   - An objective classifier: the sentence contains no strong subjective clues
   - Based on manually collected single words and n-grams, which are good subjective clues
2. Learn a set of patterns from subjective and objective sentences identified above
   - Syntactic templates are used to restrict the kinds of patterns to be discovered, e.g. <subject> active-verb $\Rightarrow$ the customer complained
3. The learned patterns are used to extract more subjective and objective sentences (the process can be repeated)

2.4 Attribute - Level Sentiment Analysis

We have seen from some of the reviews example presented earlier that a positive or a negative labeled document doesn't imply that the author likes or dislikes all attributes of the product. Thus, an interesting area of study is to analyze what attribute is most positively rated and which one is most negatively rated in a given product review. Also, in a general opinion corpus about a product, another interesting study is the percentage of positive to negative review for a given attribute. This is useful to determine, what exactly about the product do the people love or hate about the product. These attribute can be anything from product properties, the individual components or important topics etc.

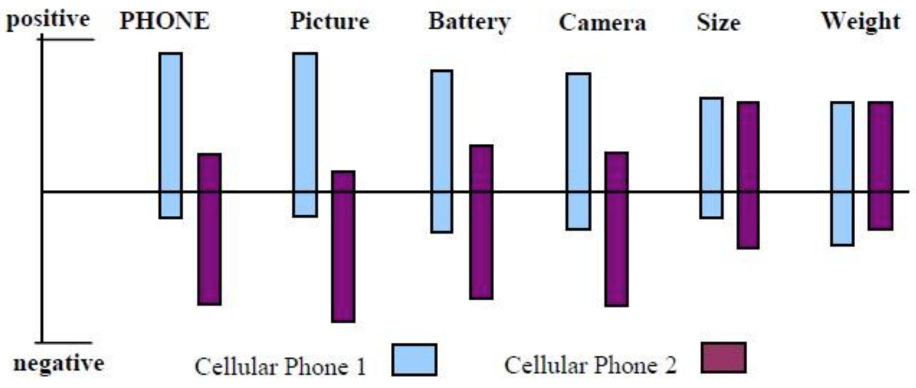


Figure 2.2: Attribute Level Sentiment Analysis Example

The Figure 2.2 shows an example of attribute level sentiment analysis. It displays the comparative analysis of two cellular phone and their individual attributes like positive or negative. Referring to the figure, one can clearly see that Cellphone 1 has overall positive sentiment, and it scores more positively for the attributes: picture, battery and camera. Where as, the size and weight of both the devices has received roughly similar reviews.

Again, there are many methods proposed which allows us to extract these attributes automatically. for the sake of focusing on this work, we will curtail the discussion here.

3 Deep Learning

Machine Learning in recent time runs in the heart of every artificially intelligent system, from image and text search on the web, to recommendation systems in popular e-commerce websites, to automatic tagging of your photos on social networking sites etc. Machine Learning has allowed us to create some very interesting applications which were not possible few decades ago. A prominent requirement in traditional machine learning systems is the use of hand labeled tagged data-set for a specific task, upon which the machine learning algorithm trains itself to explore and learn the inherent relevant patterns. We have already inferred from the discussion made in this work about few supervised techniques, which follow the same fundamental principles. All of these systems this huge effort upfront to label the data. this process known as feature engineering has a huge cost associated to it. This becomes a huge bottle neck if there is a variety in the underlying data. Specially, the problems in natural language processing, where the systems are built for various different languages, performing the feature engineering for each and every language and for each specific task can be really daunting. It's for this reason, there has been a proposed alternative to the current process, commonly known as "**Unsupervised Machine Learning**".

The idea behind this approach is to make the whole process of learning features from the data automatic for each specific task. Thus, the algorithms are just provided with the raw and unlabeled text (in case of NLP - text based ML systems) and their first task is to capture enough relevant features from the data which can be used to solve some more trivial tasks like POS Tagging, Machine Translation, Morphology Analysis, Sentiment Analysis etc. Again, there have been several approaches which have been introduced in this area of unsupervised learning and in this work, we will focus on "Deep Learning", which according to Deng et. al [12], can be defined as:

"Deep learning is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using model architectures, with complex structures or otherwise, composed of multiple non-linear transformations."

Before, we apply the deep learning algorithms and techniques to our problem of Sentiment Analysis, we will first learn about the origins of Deep Learning and its current impact and how it makes such a promising alternative to solve many complex problems in Artificial Intelligence, which is thus not only limited to text based systems, but also Speech and Image related tasks.

Conventional machine learning techniques in feature engineering, required the transformation of raw data into some form of suitable internal representation or feature vectors, which are able to learn a AI subsystem like classifier which can successfully classify patterns into relevant classes. More recently, a new form of learning has taken shape, Its known as **representation learning**. The concept of representational learning involves giving the sub-system an input of raw data directly and intrinsically learning the features automatically. Deep Learning methods are similarly a type of representation learning methods, which have multiple levels of representations. This is obtained by composing simple but non linear modules that transforms the input on the level and outputs the result to next level which uses that result to perform yet another transformation. Thus, the representations obtained at the highest level can successfully capture the more abstract view of the raw input which are more useful for classification and can also eliminate the non-essential part from the raw data which may be noise to the system.

Let's take an example of a deep learning system, which is responsible for recognizing animals in an image. The raw data are the pixels of the images to the system and they are fed to the first layer as input. The first layer captures the non-linearity in the pixels and recognizes the minimalist features like edges in the images. The next layer tries to then recognize small motifs, or similar patterns arising out of these edges. The subsequent layers tries to form the parts from these motifs and eventually the final layer will be able to successfully reconstruct the animal body or face, from the respective parts. The beauty about this procedure is that the internal layers are not handcrafted by humans but they are engineered using a standard learning procedure, which has its roots in traditional neural networks.

It is important to know at this point, that deep learning is a current buzzword but actually in its roots, they are a rebranding of neural networks. Hence, to understand these methods, we take a review of neural networks and try to understand how they work.

3.1 Foundations: Neural Networks

Neural networks are inspired by neuro-science, where the neural interconnection between the information carrying cells called neurons, connected by axons and dendrites. These vast interconnections, gives all the abilities to our brain to function in a unique way and allows us to think. Although, the area is more generally known as artificial neural networks and in practical terms, there are very few similarities, but inspiration is certainly rooted in biology.

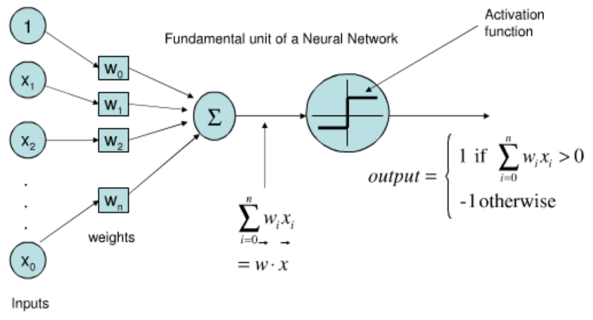

Fundamental unit of a Neural Network
Activation function
1
X1
X2
Xn
W0
W1
W2
Wn
weights
Inputs
Σ
$\sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$
$output = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

Figure 3.1: Neural Network - Single Layered Perceptron [1]

3.1.1 Background

Historically, the earliest work which laid the foundation of neural networks was done by McCulloch et al. [25], which created for the first time, a computational model based on mathematics and algorithms for neural networks. In 1958, Roseblaut F. [35] put forward a pattern recognition algorithm based on a two layer computer learning algorithm. In 1975, Paul Werbos, introduced the back propagation algorithm [43], which immediately help solved the problem of designing a circuit for the exclusive-or using just perceptron.

In Artificial Intelligence, a neural network is modeled either as a function $f : X \rightarrow Y$, or as probability distribution, or inherently defined in form of a learning rule. The simplest one layer neural network is the perceptron. It only has one input layer consisting of neuron inputs x_i and it has one output layer as shown in Figure 3.1. First a weighted sum of all the inputs is calculated and then this is later fed to an activation function (Ex. sigmoid, tanh etc.), so as to obtain the output values of these given set of input values within a given range. In the example, the weighted sum is marked as 1, if its positive and greater than 0 and -1 otherwise. Thus, a single layer perceptron works well as a binary classifier. For more complex classification with multiple inputs, one needs a more complex configuration. And also a learning mechanism which can help to send back the intermediate values and learn more accurate weight values. In such cases, hidden layers are introduced comprising purely of neurons, which take the inputs directly and forms an intermediate layer. There can be a number of intermediate layers, but the more the layers, more time it takes to train the machine learning model.

[1]See `http://i.stack.imgur.com/KUvpQ.png`

The work describes use of one such typical configuration *i.e.,* Feed Forward Neural Networks. Other configurations are recurrent neural network etc. The training is done by using back propagation which is used in tandem with an optimization method, ex. gradient descent.

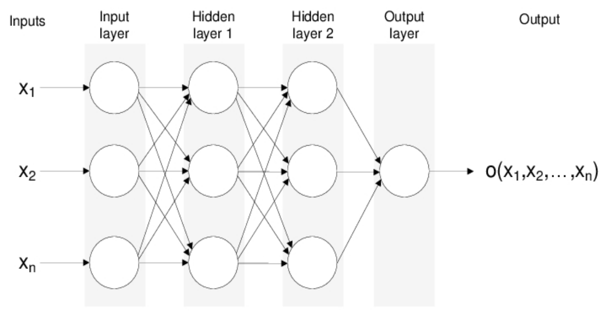


Figure 3.2: Neural Network - Multiple Layered Perceptron [2]

3.1.2 Feed Forward Neutral Networks and Back-Propagation

Feed forward neural networks [4] are a type of artificial neural network, where the neurons or computation units do not form a cycle. The network proceeds only in one direction. A perceptron is a single layered feed forward neural network, devoid of any hidden layers. The *delta rule* is used in this case for training the perceptron. It calculates the errors and suggests weight adjustment based on difference in output and pre-collected sample output data. This mimics sort of gradient descent algorithm. The output too is in form of a step function, either one value or the other. A multilayer perceptron helps to get more continuous values as output through the use of the activation functions like logistic function.

$$y = \frac{1}{1+e^x}$$

One such multi-layered perceptron is shown in Figure 3.2. As shown multiple hidden layers are involved between the input and the output layers. It uses back propagation learning algorithm for training. This is performed in two steps: propagation and updating weights. The propagation stage involves starting with a random set of weights and then propagating network forward to get the output activation values. Next we setup the backward propagation where using the sampled target values and the output activation values, we compute the deltas of all output and hidden neurons. This information is used in the next stage of weight update, which computes the weight

[2]See `http://www.intechopen.com/source/html/38302/media/ann.jpg`

gradient by multiplying the output delta with input activation. At last, we subtract a ratio of this gradient from the weights and hence learn new weights. The ratio is our learning rate which determines speed and quality of learning. This process is repeated till the network gives sufficient performance. The back propagation algorithm tries to find such set of weights which can minimize the error (in our case the square error E) using gradient descent. As the error needs to be minimized, this can be seen as an optimization problem. Thus, we aim to find first derivative of this error. The error is described as:

$$E = \frac{1}{2}(t - y)^2, \text{ where,}$$

E = Squared Error
t = Training Sample's target or expected output
y = Actual computed output of the output neuron.

3.1.3 Example: Solving Neural Network with Back Propagation

Consider the following neural network, Figure 3.3 with given inputs A and B, weights and sigmoid activation function: $y = \frac{1}{1+e^{-x}}$

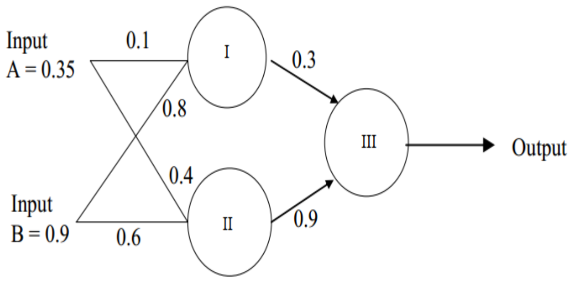


Figure 3.3: Neural Network - Example

Forward Pass: Calculating the weighted sum for each neuron.
Neuron I : 0.35 $\times$ 0.1 + 0.9 $\times$ 0.8 = 0.755
Output I [O(I)]: $y = \frac{1}{1+e^{-0.755}}$ = 0.68
Neuron II: 0.9 $\times$ 0.6 + 0.35 $\times$ 0.4 = 0.68
Output II [O(II)]: $y = \frac{1}{1+e^{-0.68}}$ = 0.6637
Neuron III: 0.3 $\times$ 0.68 + 0.9 $\times$ 0.6637 = 0.80133
Output III [O(III)]: $y = \frac{1}{1+e^{-0.80133}}$ = 0.69

Backward Pass: Original TARGET = 0.5

Output Error: δ
δ = (TARGET - O(III)) $\times$ (1-O(III)) $\times$ O(III)
= (0.5-0.69) $\times$ (1-0.69) $\times$ (0.69)
= -0.0406
New Weights for Output Layer *i.e.*, the weights connecting to Neuron III :
For computing the new weights,
$Input_I{}^+ = Output_I$
$Input_{II}{}^+ = Output_{II}$
$W_1{}^+ = W_1 + (\delta \times Input_I{}^+)$
= 0.3 +(-0.0406 $\times$0.68)
= 0.2723
$W_2{}^+ = W_2 + (\delta \times Input_{II}{}^+)$
= 0.9 + (-0.0406 $\times$0.6637)
= 0.87305

Errors for Hidden Layers:
$\delta_I = \delta \times W_1 \times (1 - O(I)) \times O(I)$
= -0.0406 $\times$ 0.272392 $\times$ (1-0.68) $\times$ 0.68
= -2.406×10^{-3}
$\delta_{II} = \delta \times W_2 \times (1 - O(II)) \times O(II)$
= -0.0406 $\times$ 0.87305 $\times$ (1- 0.6637) $\times$ 0.6637
= -7.916×10^{-3}

New Hidden Layer Weights:
(Input is A = 0.35 and B = 0.90)
$W_{new} = W_{old} + (\delta \times Input)$
For weights entering Neuron 1:
$W_3{}^+ = 0.1 + (-2.406 \times 10^{-3} \times 0.35)$
= 0.09916
$W_4{}^+ = 0.8 + (-2.406 \times 10^{-3} \times 0.9)$
= 0.7978
For weights entering Neuron 2:
$W_5{}^+ = 0.4 + (-7.916 \times 10^{-3} \times 0.35)$
= 0.3972
$W_6{}^+ = 0.6 + (-7.916 \times 10^{-3} \times 0.9)$
= 0.5928
Thus, after one forward pass and one backward propagation step we have the following neural network as shown in Figure 3.4.

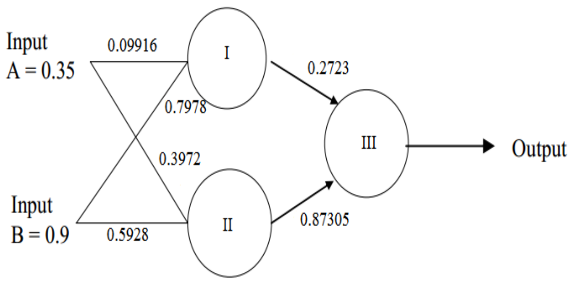


Figure 3.4: Neural Network - After one forward and one back propagation step.

Another Forward Pass
Neuron I: 0.35 $\times$ 0.09916 + 0.9 $\times$ 0.7978 = 0.752726
Output (I): 0.67977
Neuron II: 0.9 $\times$ 0.5928 + 0.35 $\times$ 0.3972 = 0.67254
Output (II): 0.6620
Neuron III: 0.2723 $\times$ 0.67977 + 0.87305 $\times$ 0.6620 = 0.76306
Output (III): 0.68201

Computing Overall Error:
$Target_{Original}$ = 0.50
$Target_{Pass1}$ = 0.69
Error (Pass 1) = $Target_{Original} - Target_{Pass1}$
= - 0.19
$Target_{Original}$ = 0.50
$Target_{Pass2}$ = 0.68201
Error (Pass 2) = $Target_{Original} - Target_{Pass2}$
= - 0.18201

So, we find: **Error (Pass 2) is less than Error (Pass 1)** Therefore, the error has reduced.

We repeat this process, till the point that, we have no reduction in overall error. At each iteration, the machine automatically adjusts it's internal parameters or weights, which cause the error to be reduced. At this point the process is stopped and the learned weights are chosen as our final weights. These weights can be used to a random input to predict the correct class or output.

3.2 From Neural Networks to Deep Neural Networks

In subsection 3.1.3, we saw a simple two unit, single hidden layer neural network and its working through back-propagation algorithm. We was the most important take away message is the most important outcome of this learning procedure is actually to **learn the correct weights** for the network. We saw that to properly compute weights, the algorithm computes a gradient vector that, for each weight, indicates by what amount the error would increase or decrease if the weight were increased by a tiny amount. The weight vector is then adjusted in the opposite direction to the gradient vector. This negative value of the gradient vector takes the objective function (averaged over all training values in a high dimensional plane of weight vectors) to its minimum value, where the output error is low on average.

The commonly used procedure to achieve this is **Stochastic Gradient Descent** (SGD). This consists of showing the input vector for a few examples, computing the outputs and the errors, computing the average gradient for those examples, and adjusting the weights accordingly. The process is repeated for many small sets of examples from the training set until the average of the objective function stops decreasing. It is called *stochastic* because each small set of examples gives a noisy estimate of the average gradient over all examples. [24] This is highly efficient technique as compared to other optimization techniques. Later, the system's performance is evaluated over a test set. this is a good generalization of the trained model since it helps us to test model on unseen data.

Most current systems based on machine learning use the two class classifier over hand crafted features. although, these are effective to an extent, but they do not fully understand the underlying complexity. For making the classifiers more powerful towards the given task, one can use the non-linear features as used in Kernel Methods [36], but even the features which arise with traditional Gaussian Kernels, do not generalize well far from the examples seen during training. The alternative is to hand craft features, but as we know its a tedious task. The better way as we know it is to learn features automatically. This is an advantage of Deep Learning.

A **deep learning architecture** can be visualized as a **multi-layered stack** of simple modules of learning which are responsible to compute **non-linear input-output mappings**. Each module aims to transform the input to increase both the selectivity and in-variance of the representation. The system can implement many complex and intricate functions if we increase the number of such non-linear layers (to a depth of 5 - 20) and model relevant parts of the input and also leave out irrelevant details, leading to a more concise and precise representation.

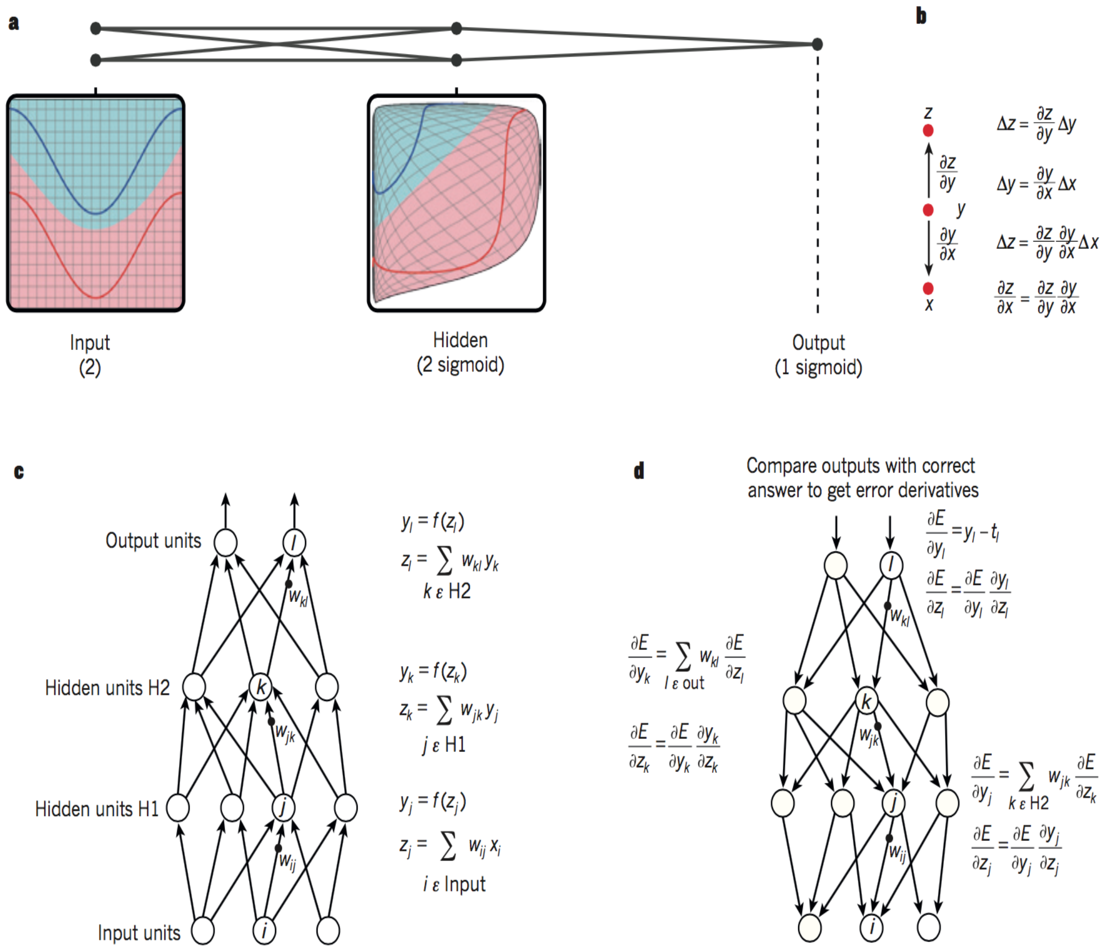

a
Input (2)
Hidden (2 sigmoid)
Output (1 sigmoid)
b
z
y
x
$\frac{\partial z}{\partial y}$
$\frac{\partial y}{\partial x}$
$\Delta z = \frac{\partial z}{\partial y} \Delta y$
$\Delta y = \frac{\partial y}{\partial x} \Delta x$
$\Delta z = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x$
$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$
c
Output units
Hidden units H2
Hidden units H1
Input units
$y_l = f(z_l)$
$z_l = \sum_{k \varepsilon H2} w_{kl} y_k$
$y_k = f(z_k)$
$z_k = \sum_{j \varepsilon H1} w_{jk} y_j$
$y_j = f(z_j)$
$z_j = \sum_{i \varepsilon \text{Input}} w_{ij} x_i$
d
Compare outputs with correct answer to get error derivatives
$\frac{\partial E}{\partial y_l} = y_l - t_l$
$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial z_l}$
$\frac{\partial E}{\partial y_k} = \sum_{l \varepsilon \text{out}} w_{kl} \frac{\partial E}{\partial z_l}$
$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k}$
$\frac{\partial E}{\partial y_j} = \sum_{k \varepsilon H2} w_{jk} \frac{\partial E}{\partial z_k}$
$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j}$

Figure 3.5: Multi-layered Neural Networks and Back-Propagation [3]

Many Deep Learning Applications use Feed forward Neural Network Architectures Figure 3.5., which learn to map fixed sized input to a fixed sized output. to go from one layer to the next, a set of units compute a weighted sum of their inputs from the previous layers and pass the result through a non linear function. In the past, traditional neural networks used the **sigmoid** or **hyperbolic tangent** non linear functions. The more recent deep learning architectures rely of **rectified linear units** (ReLU). They are otherwise known as half wave rectifiers f(z) = max(z,0). They are exceptionally good at learning at a much faster rate in multi-layered networks and allow training of a deep

[3]See `http://colah.github.io/`

supervised network without unsupervised pre-training. The hidden units in the hidden layer (layers other than the input and output layer) distort the input signal in a non linear way such that the categories become separable by the last layer.

Also, if we look at the history of neural networks, they started out brilliantly in solving a number of AI problems and showing good results, but by late 1990's they weren't as highly regarded. One of the main proponents against their adoption was the underlying use of Stochastic Gradient Descent. It was thought that while exploring the global minimum value for the objective function, the method might get to only the **local minima**. Thus the weight configuration changes were not considered highly optimized. What was found later in practice with large multi-layered networks working on high dimensions is that this problem has hardly any significant effect. In fact, the results using a local minima or a global minima are often quite comparable.

In 2006, the research group from Canada Institute for Advanced Research (CIFAR), introduced unsupervised learning methods that created layers of feature detectors using unlabeled input data. The objective in learning, each layer was to model the activities of layers below it. Using a reconstruction objective and pre-training several layers of complex feature detectors, the weights of neural network can be initialized to proper values. A final layer is later added to the system and the system itself is then fine tuned using back-propagation. This was a huge breakthrough as it worked remarkably well for recognizing hand written digits or for detecting pedestrians, even though the amount of hand labeled data was limited.

Later this pretraining approach, helped in speech recognition, partly with the adoption and ease of use of Graphical Processing Units (GPU), which allowed a 10x - 20x times of increase in overall training time. the procedure was able to beat the current state of the art with ease and by 2012, the method was fine tuned and adopted into mobile hand held devices running Android[4]. Interestingly, it was found out through numerous experiments that pre-training is useful for the case where we have a small data-set to begin with. Another class of deep feed forward neural network which have gained popularity are the **Convolutional Neural Networks** (CNN). They are easiest to train and generalize much better than the networks which had full connectivity between the layers. These have been remarkable in several image processing tasks and have been adopted widely by the computer vision community.

4 Deep Learning and NLP: Distributed Representations and Language Modelling

Most, of the current Deep Learning research in NLP is focused on representation learning. By that, we mean that we represent our word, phrase or a sentence as a series of numerical values or a vector in a given vector subspace, which somehow captures the representation. This is an extensive area of research and the roots are based on the concept of Language Modeling.

4.1 Language Modelling

A language model is a probabilistic model that assigns probabilities to any sequence of words. Such as:

$$p(w_1, \ldots, w_T)$$

Language modeling is the task of learning a language model that assigns high probabilities to well formed sentences. We take into consideration: nth order Markov assumption, where we assume that the i^{th} word was generated based only on the n-1 previous words. It plays a crucial role in speech recognition and machine translation systems.

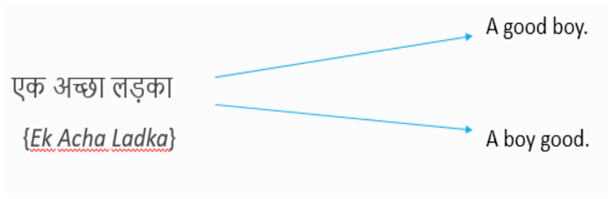


Figure 4.1: Translation of a Hindi language Sentence to English language

Consider the example in Figure 4.1. We can see that the Hindi language sentence can be translated in two different ways. First, *A good boy* and second, *A boy good*. What a good language model will tell us that the first one is the correct and more generally used form of the sentence. This is the power and importance of language modeling and is in the heart of all the current unsupervised natural language processing machine learning algorithmic tasks.

4.1.1 Language Modelling using Ngrams

An N-gram is defined as a sequence of n-words. For example:
unigrams (n=1): "is", "a", "sequence", etc.
bigrams (n=2): ["is", "a"], ["a", "sequence"], etc.
trigrams (n=3): ["is", "a", "sequence"], ["a", "sequence", "of"], etc.
The **n-gram models** estimate the conditional from n-grams counts.

$$P(w_t|w_{t-(n-1)}, ..., w_{t-1}) = \frac{count(w_{t-(n-1)}, ..., w_{t-1}, w_t)}{count(w_{t-(n-1)}, ..., w_{t-1})} \tag{4.1}$$

The counts are obtained from a training corpus (a data-set of word text)

We want the value of **n** to be large, for the model to be realistic. But, for large values of **n**, it is likely that a given n-gram will not have been observed in the training corpora. Smoothing helps but only partially!

4.1.2 Word Representations

In context of current problem, we use usupervised deep learning techniques, to learn a word representation C(w) which is a continuous vector and is both syntactically and semantically similar. More precisely, we learn a continuous representation of words and would like the distance $||C(w) - C(w')||$ to reflect meaningful similarity between words **w** and **w'**.

Chiefly, we explore the following word representations: Word2Vec, Polyglot, GloVe and their concatenated combinations along with Bag of Words representation and compare their results for the task of sentiment analysis.

A traditional view of the word representation is the **bag of words** or **one-hot vector** representation for the word. In this model, a text (such as a sentence or a document) is represented as the bag (multi-set) of its words, disregarding grammar and even word order but keeping multiplicity. Example: Suppose we have two documents D1 and D2

D1: *John likes to watch movies. Mary likes movies too.*

D2: *John also likes to watch football games.*

Vocabulary {Word : Index}

"John": 1, "likes": 2, "to": 3, "watch": 4, "movies": 5, "also": 6, "football": 7, "games": 8, "Mary": 9, "too": 10

There are 10 distinct words and using the indexes of the Vocabulary , each document is represented by a 10-entry vector:

[1, 2, 1, 1, 2, 0, 0, 0, 1, 1]
[1, 1, 1, 1, 0, 1, 1, 1, 0, 0]

There is a concern that in this representation, the more frequent words are weighted more than less frequent words. So, to normalize the weights across documents another representation called TF-IDF (Term Frequency - Inverse Document Frequency) is introduced.The main cause of concern with BOW and TF-IDF representation is they don't make the use of contextuality of words in the corpus. Thus, there is no way of distinguishing the order of the words correctly through this vector representation. A more recent view on this is taken through neural language modeling and it brings forward a new concept known as **Word Embeddings**. On the contrary to the one-hot representation, they build on a concept of **Distributed Representations**.

Originally, word embeddings were introduced by Bengio et al., [2] [3] a few years before the discussed 2006 deep learning renewal, at a time when neural networks were not that popular. Although, the idea of distributed representations for symbols is much more older, as proposed by Hinton et. al. [18]
A **word embedding** W : words $\Rightarrow R^n$ is a paramaterized function mapping words in some language to high-dimensional vectors (150 - 600 dimensions). For example, we might find:
W('parrot") = (0.3, -0.5, 0.7, ...)
W("carrot") = (0.0, 0.6, -0.1, ...)

Typically, the function is a lookup table, parameterized by a matrix, θ, with a row for each word: $W_\theta(w_n) = \theta_n$. W is first initialized randomly for each word. When we train the word vector matrix, it learns to have meaningful vectors in order to perform some task.

For example, we can train a neural network to predict whether a 5-GRAM (sequence of five words) is correct or 'valid'. We can start with 5-grams from Wikipedia corpus (Ex.. "plane is going to fly") and then corrupt half of them by switching a word with a random word (eg. "plane is drunk to fly"), thus, almost certainly making our corrupted 5-gram absolute nonsense.The neural network we train will feed each word in the 5-gram through W to get a vector representing it as intermediate output and feed those intermediate results into another layer say 'L' which tries to predict if the 5-gram is 'valid' or 'corrupt.' [6] We would obtain something like:

L(W("plane"), W("is"), W("going"), W("to"), W("fly"))=1

L(W("plane"), W("is"), W("drunk"), W("to"), W("fly"))=0

In order to predict these values accurately, the network needs to learn good parameters for both W and L. This is accomplished through back-propagation, which we have already covered earlier.

An interesting side effect which is observed, with these word vectors, is when trained on a very large corpus like Google News, the vectors for days like Tuesday and Wednesday fall close to one another in high dimensional space. It's not just limited to days, but also gender, where the same gender words fall in a certain cluster group and difference between the tow gender entities is captured across relations. For Example:

v("king") - v("queen") = v("man") - v("woman")

These are interesting relationships which are somehow giving the sense that they capture the semantic similarity in vector space. Thus, with such results, the entire field of Deep Learning and NLP looks really fascinating. Next, we will discuss some of the recent methods, which have contributed strongly to this discussion.

4.2 Word2Vec

Mikolov et. al. proposed two novel architectures Figure 4.3 [26] [27] for computing continuous vector representations of words from very large datasets. They are:

- Continous Bag of Words (cbow)
- Continous Skip Gram (skip)

Word2Vec focuses on distributed representations learned by neural networks Figure 4.2. The N-previous words are encoded using 1-of-V hot-vector coding. The words

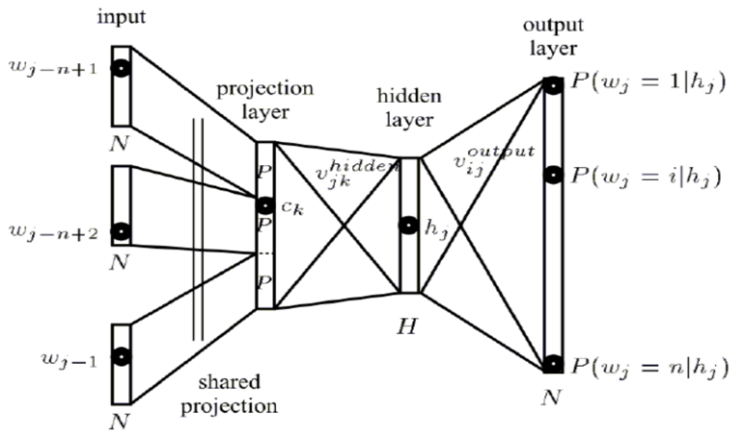


Figure 4.2: Word2Vec Neural Network Language Model

are then projected by a linear operation on the projection layer. Softmax function is used at the output layer to ensure that 0 < = p < = 1. All models are trained and weights are learned using stochastic gradient descent and back propagation. For all models, the training complexity is proportional to:

$$O = E \times T \times Q,$$

Where, E: # of Training epochs; T = # Words in Training Set; Q = Defined further for each model architecture.

CBOW: Predicts the current word based on the context.

- Similar to feed-forward neural network language model, where the non linear hidden layer is removed and projection layer is shared for all words (not just the projection matrix); thus all words are projected into the same position (their vectors are averaged).
- Best performance, by building the log linear classifier with four future and four history words as input, where training criteria is to correctly classify the current (middle) word.

SKIP: Tries to maximize classification of word based on another word in the same sentence.

- We use each current word as input to a log linear classifier with continuous projection layer and predicts words within a certain range before and after current word.

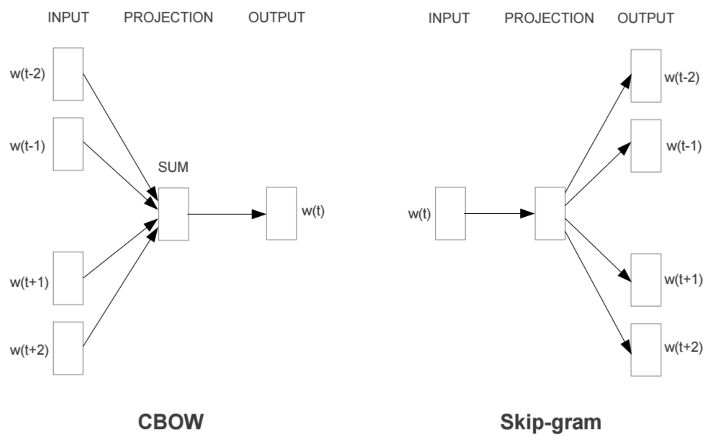


Figure 4.3: Word2Vec Model Architectures: CBOW and Skip-Gram

Example: The analogy "**king is to queen as man is to woman**" should be encoded in the vector space by the vector equation: king - queen = man - woman

Among the two, the **Skip-Gram model seems to perform better** with respect to CBOW model on the Word-analogy task (see example above) for both syntactic and semantic categories.

Mikolov et. al discuss the skip gram model separately in another work [27]. The training objective for the skip gram model is to find the word representations that are useful for predicting the surounding words in a sentence or a document. More formally, given a sequence of training words $w_1, w_2, w_3, ..., w_T$, the objective of the Skip-gram model is to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|w_t) \tag{4.2}$$

where **c** is the size of the training context (which can be a function of the center word wt). Larger **c** results in more training examples and thus can lead to a higher accuracy, at the expense of the training time. The basic Skip-gram formulation defines $p(w_{t+j}|w_t)$ using the softmax function:

$$p(w_O|w_I) = \frac{exp({v'_{wO}}^T v_{wI})}{\sum_{w=1}^{W} exp({v'_w}^T v_{wI})} \tag{4.3}$$

where v_w and v'_w are the "input" and "output" vector representations of w, and W is the number of words in the vocabulary.

Hierarchical Softmax [28] is computationally more efficient than the traditional softmax. It uses a binary tree representation of the output layer with the W words as its leaves and for each node explicitly represents the relative probabilities of its child nodes. These define a random walk that assigns probabilities to words. The key advantage here is instead of evaluating W output nodes in the neural network to obtain the probability distribution, only about log(W) nodes are required to be evaluated. More precisely, each word w can be reached by an appropriate path from the root of the tree. Let n(w, j) be the j-th node on the path from the root to w, and let L(w) be the length of this path, so n(w, 1) = root and n(w, L(w)) = w. In addition, for any inner node n, let ch(n) be an arbitrary fixed child of n and let [[x]] be 1 if x is true and -1 otherwise. Then the hierarchical softmax defines $p(w_O|w_I)$ as follows:

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma\big([n(w,j+1) = ch(n(w,j))].{v'_{n(w,j)}}^T v_{wI}\big) \tag{4.4}$$

Where, $\sigma(x) = 1/(1+exp(-x))$.

The work also introduces negative sampling. Its a simplified form of Noise Contrastive Estimation (NCE), which was introduced by Gutmann et al. [16]. This simplification can be done as long as the vector representations retain their quality. Negative Sampling is defined by the objective function:

$$log\sigma({v'_{w_O}}^T v_{w_I}) + \sum_{i=1}^{k} E_{w_i \sim P_n(w)}[log\sigma(-{v'_{w_i}}^T v_{w_I})] \tag{4.5}$$

which is used to replace every $logP(w_O|w_I)$ term in the skip-gram objective. Thus the task is to distinguish the target word w_O from draws from the noise distribution $P_n(w)$ using logistic regression, where there are k negative samples for each data sample. After experimental investigations, the unigram distribution $U(w)^{3/4}/Z$ outperformed other uniform and unigram distributions.

Moreover, to counter the imbalance between the rare and frequent words, a simple **subsampling approach** was used, where each word w_i in the training set is discarded with probability computed by the formula:

$$Pw_i = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{4.6}$$

where, $f(w_i)$ is the frequency of word w_i and **t** is the threshold which is 10^{-5}. The method, aggressively subsamples words whose frequency is greater than **t** while preserving the ranking of the frequencies.

4.3 GloVe

Pennington et al. [32] introduced GloVe or **Global Vectors for Word Representations**, a global bi-linear regression model that combines the advantages of the two major model families in the literature: global matrix factorization and local context window methods. The model efficiently leverages statistical information by training only on the nonzero elements in a word-word co-occurrence matrix, rather than on the entire sparse matrix or on individual context windows in a large corpus. This study was at large developed to understand why there exists linguistic similarity properties in word embeddings.

Let the matrix of word-word co-occurrence counts be denoted by X, whose entries X_{ij} tabulate the number of times word j occurs in the context of word i. Let $X_i = \sum_k X_{ik}$ be the number of times any word appears in the context of word i. Finally, let $P_{ij} = P(j|i) = X_{ij}/X_i$ be the probability that word j appear in the context of word i.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 4.4: Co-occurrence probabilities for target words ice and steam with selected context words f

Now considering two words i = ice and j = steam, The relationship of these words Figure 4.4 can be examined by studying the ratio of their co-occurrence probabilities with various probe words, k. For words k related to ice but not steam, say k = solid, we expect the ratio P_{ik}/P_{jk} will be large. Similarly, for words k related to steam but not ice, say k = gas, the ratio should be small. For words k like water or fashion, that are either related to both ice and steam, or to neither, the ratio should be close to one. As compared to raw probabilities, the ratio is better able to distinguish relevant words (solid and gas) from irrelevant words (water and fashion) and it is also better able to discriminate between the two relevant words.

As the ratio P_{ik}/P_{jk} depends on three words i, j and k, the most general model takes the form:

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \tag{4.7}$$

where, w are word vectors and $\tilde{w}$ are context word vectors. F can depend on a number of parameters and the RHS is inferred from the corpus. We wold like F to encode information present in the ration of probabilities, we can achieve this by applying vector differences to Equation 4.7.

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \tag{4.8}$$

In Equation 4.8 the parameters of F are all vectors whereas the RHS is scalar. Thus, we convert the LHS to scalar by taking a dot product. This also prevents F to mix up vector dimensions undesirably.

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \tag{4.9}$$

Note that, for word-word co-occurrence matrices, the distinction between a word and a context word is arbitrary and e can exchange the two roles. This can be performed on Equation 4.9 in two steps. First, we require that F be a homomorphism between the groups (R,+) and ($R_{>0}$, x), i.e.,

$$F(w_i - {w_j}^T, \tilde{w}_k) = \frac{F({w_i}^T \tilde{w}_k)}{F({w_j}^T \tilde{w}_k)} \tag{4.10}$$

which by Equation 4.9 is solved by:

$$F({w_i}^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i} \tag{4.11}$$

The solution to Equation 4.10 is F = exp,

$${w_i}^T \tilde{w}_k = log(P_{ik}) = log(X_{ik}) - log(X_i). \tag{4.12}$$

In Equation 4.12, the term $log(X_i)$ is independent of **k** so it can be absorbed into a bias b_i for w_i. Finally, adding an additional bias $\tilde{b}_k$ for $\tilde{w}_k$ restores the symmetry,

$${w_i}^T \tilde{w}_k + b_i + \tilde{b}_k = log(X_{ik}). \tag{4.13}$$

There are two problems with Equation 4.13, first, when argument is zero for logarithm, it diverges. This can be overcome by substitution of $log(X_{ik}) \rightarrow log(1 + X_{ik})$.

Another drawback is that the model weighs all occurrences equally even for those which happen rarely or never.

Finally we model Equation 4.13 as new weighted least squares regression model that addresses these problems. A weighting function $f(log(X_{ij})$ is introduced into the cost function giving us:

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i{}^T \tilde{w}_k + b_i + \tilde{b}_k - log(X_{ik})\right)^2. \tag{4.14}$$

where V is size of Vocabulary. The choice of function f was generalized as follows:

$$f(x) = \begin{cases} (x/x_{max})^{\alpha} & x < x_{max} \\ 1 & otherwise \end{cases} \tag{4.15}$$

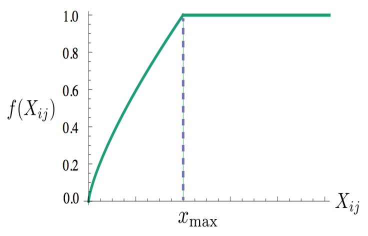


Figure 4.5: GloVe Weight Function f with α =3/4

The value for x_{max} was fixed at 100 and value of $\alpha = 3/4$ Figure 4.5 provided modest improvement over $\alpha = 1$ after experimental evaluation. Thus the, Equation 4.14 is the final GloVe model. Interestingly, The GloVe model beat the Word2Vec model on the standard analogy task on both the syntactic and semantic categories.

4.4 Polyglot

Rami Al-Rofou et. al, [33] trained word embeddings for more than 100 languages using their corresponding Wikipedias and evaluated them against a standard Part of Speech (POS) tagging task. To construct the objective function for the neural network, a given sequence of words $S = [w_{i-n} \cdots w_i \cdots w_{i+n}]$ observed in the corpus T, another corrupted sequence S′ is constructed by replacing the word in the middle w_i with a word w_j chosen at random. The neural network represents a function score that scores each phrase, the model is penalized through the hinge loss function J(T) as shown in

Equation 4.16

$$J(T) = \frac{1}{|T|} \sum_{i \in T} |1 - score(S') + score(S)|_{+} \tag{4.16}$$

First, each word is mapped through a vocabulary dictionary, size |V| to an index that is used to index a shared matrix C with the size |V| $\times$ M Figure 4.6. Here, M is the size of the vector representing the word.

These vectors are then concatenated into one vector called projection layer P with size (2n + 1) $\times$ M. The projection layer plays the role of an input to a hidden layer with size |H|, the activations A of which are calculated according to equation 3, where W_1, b_1 are the weights and bias of the hidden layer.

$$A = tanh(W_1 P + b_1) \tag{4.17}$$

To calculate the phrase score, a linear combination of the hidden layer activations A is computed using W_2 and b_2 .

$$score(P) = W_2 A + b_2 \tag{4.18}$$

Therefore, the five parameters that have to be learned are W_1 , b_1, W_2, b_2 and C.

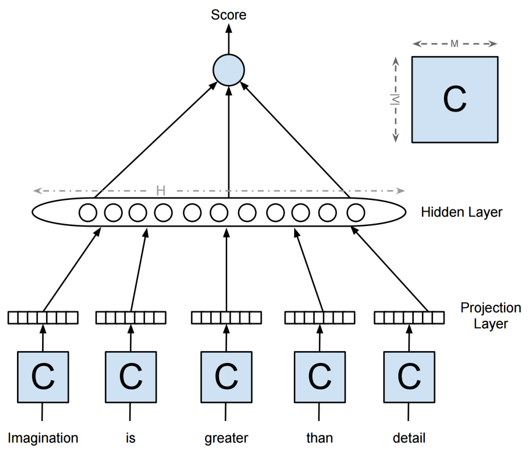


Figure 4.6: Polyglot Neural Network Architecture

4.5 Paragraph Vectors

Quoc Le and Thomas Mikolov [22] introduced Paragraph Vectors, an unsupervised algorithm that learns fixed length fixed representations from variable lengths texts like sentences, paragraphs and documents. The algorithm **represents an entire document as a dense vector**, which is trained to predict words in the document. The inspiration for paragraph vectors are the vectors, which are asked to predict the next word in the sequence. And even though these vectors are randomly initialized, they can contribute to the semantics and capture semantic similarity as an indirect side-effect of this prediction task. What makes paragraph vectors different is that they too contribute to prediction task, but the context is not words but paragraphs. They build upon the earlier work done by Mikolov et al. [26] [27] and use the same fundamental objective function and additionally now uses both words and paragraph vectors as context.

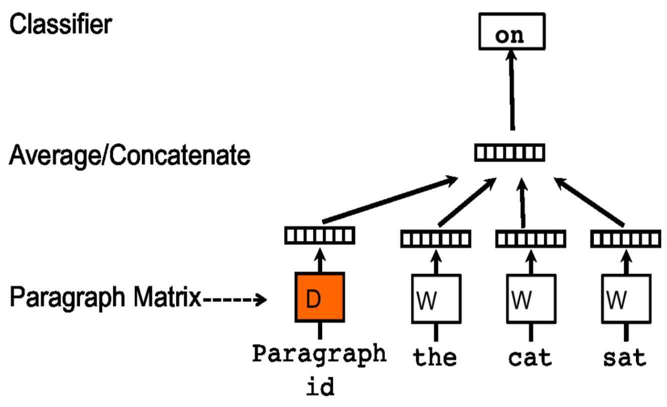


Figure 4.7: Framework for Learning Paragraph Vectors

The framework for learning paragraph vectors can be seen in Figure 4.7. every paragraph is mapped to a unique vector D along with the words which are mapped to a unique word vector W. The paragraph vectors and the word vectors are concatenated or averaged to predict the next word in the sentence. When learning paragraph vectors, both D and W are used. The paragraph token can be seen as just another word. This acts as a memory which preserves the topic of the paragraph. It's for this reason, paragraph vector model is also known as Distributed Memory Model of Paragraph Vectors (PV-DM). The context window is fixed and sampled using a sliding window across paragraphs. The paragraph vector is shared across all the contexts generate across same paragraph, whereas the word vectors are shared across all the paragraphs in the document. Both vectors are trained using stochastic gradient descent and the gradient is obtained using back-propagation. At prediction time, we perform an inference step which computes the paragraph vector for a new paragraph which is again obtained using stochastic gradient descent, but there the parameters for the model softmax and word vectors W are fixed.

4.6 Interesting properties of Word Vectors

Word Vectors have a very interesting property, when these word vectors are projected in high dimensional space, the semantically closer words are found to be similar to each other. This was highlighted first in Word2Vec and then later in GloVe. Few examples are shown below visualize them on a 2D plane by using PCA dimensionality reduction Figure 4.8 and t-SNE Figure 4.9.

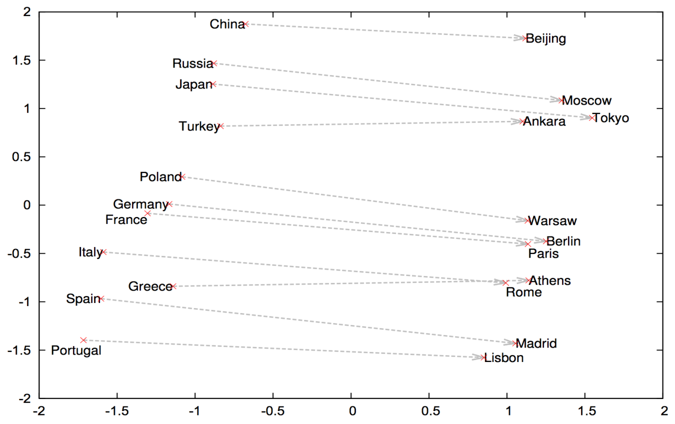


Figure 4.8: Word2Vec - Countries and Capitals Relationship [27]

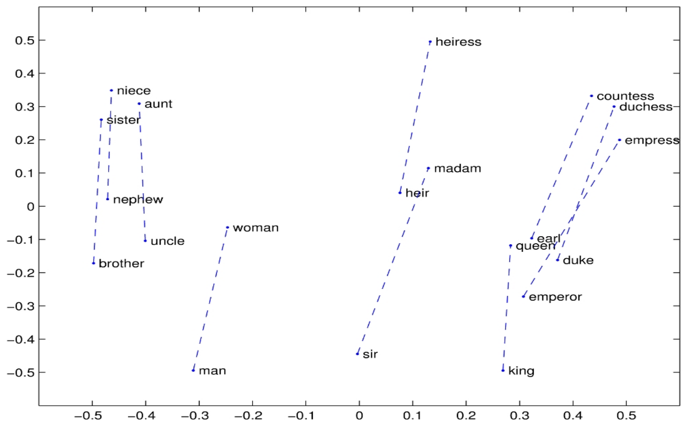


Figure 4.9: GloVe - Gender Relationship [32]

A direct result of these are the analogy relations based on difference vectors. For Example: *v("king") - v("queen") = v("man") - v("woman")*

5 Deep Learning: Modern Apporaches for Unsupervised Sentiment Analysis

We have just discussed some of the most recent work in the field of deep learning. In chapter 6 we make use of them in unique ways and target a specific variant of sentiment analysis problem. Before that, we present some of the novel work done in this area ordered year-wise, which use various deep learning algorithms and in some way describe how the area has graduated over the years.

5.1 RNTN : Recursive Neural Tensor Networks

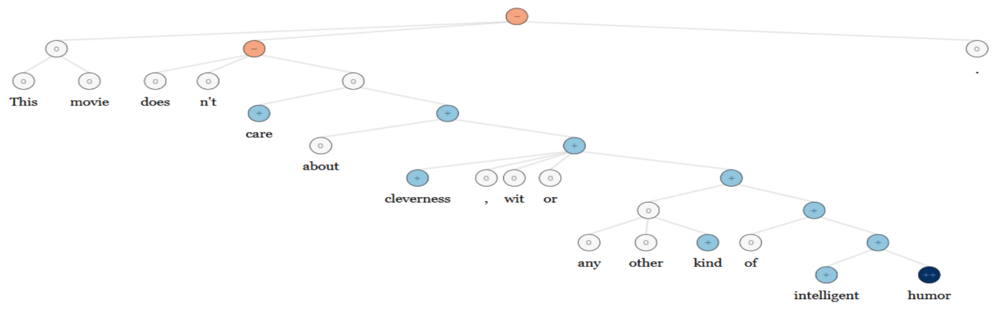


Figure 5.1: Example of Recursive Neural Tensor Network predicting sentiment at each node and capturing negation.

Socher et. al [39] in 2013, introduced Recursive Neural Tensor Networks (RNTN) Figure 5.1 and applied them to sentences and phrases of movie reviews and predicted sentiment across five broad categories in fine grained scheme (positive, slightly positive, neutral, negative and slightly negative) and across three categories in coarse grained scheme (positive, negative and neutral). The model was trained on a Sentiment Treebank which included fine grained sentiment labels for 215,154 phrases in the parse trees of the 11,855 sentences. The inspiration behind the work is understanding compositionality in sentences using deep learning and leveraging it to identify sentiment.

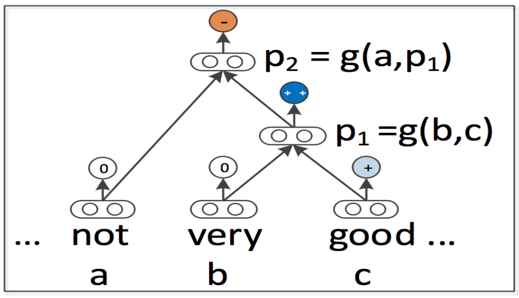


Figure 5.2: Recursive Neural Network models for sentiment: Computing parent vectors in a bottom up fashion using compositionality function **g**.

RNTN are based on recursive neural network models. They compute compositional vector representations for phrases of variable length and syntactic type. When n-gram is given as input to compositional models, it is parsed into a binary tree and each leaf node, corresponding to a word is represented as a vector. RNN then compute parent vectors in a bottom up fashion using the various composiitonality functions g. They are again given as input features to a classifier. Figure 5.2 shows a tri-gram example.

Each vector is represented as a d - dimensional vector. They are first initialized randomly from a uniform distribution U(-r, r), where r = 0.0001. They are then stacked in a word embedding matrix $L \in R_{dx|V|}$, where V is vocabulary size. L matrix is seen as a parameter which is trained jointly with the compositional models. For classification into five sentiment classes, the vectors can be fed to a softmax classifier. We compute the posterior probability over labels given the word vector via:

$$y_\alpha = softmax(W_s a) \tag{5.1}$$

Here, W_s is sentiment classification matrix. This is repeated in the trigram for b and c as well. Figure 5.2. There are other models which are used to determine hidden layers / units. They are simple RNN [15] [38] and Matrix Vector RNN (MV-RNN) [37], which have also been proposed but RNTN tries to answer the quesion, which was unanswered by the latter two models, *i.e., Can a single, more powerful composition function perform better and compose aggregate meaning from smaller constituents more accurately than many input specific ones?* And thus, the main idea behind RNTN is to use the same tensor based composition function, for all nodes.

Figure 5.3 shows a single layer neural network. An output of a tensor product h is

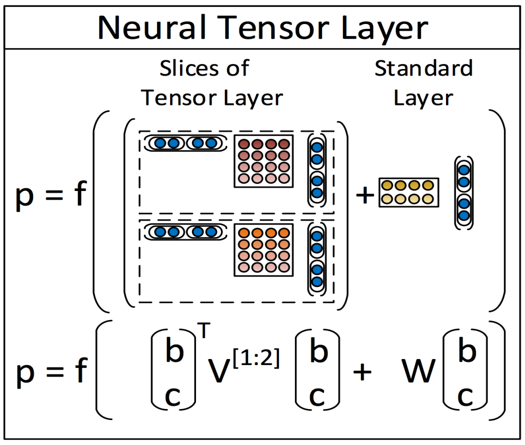


Figure 5.3: Single layer of the Recursive Neural Tensor Network. Each dashed box represents one of d-many slices and can capture a type of influence a child can have on its parent

defined as:

$$h = \begin{bmatrix} b \\ c \end{bmatrix}^T V_{[1:d]} \begin{bmatrix} b \\ c \end{bmatrix}; \tag{5.2}$$

where, $V_{[1:d]}$ tensor defines multiple bi-linear forms. Thus, to compute p_1, RNTN follows:

$$p_1 = f(\begin{bmatrix} b \\ c \end{bmatrix}^T V_{[1:d]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}) \tag{5.3}$$

and similarly, p_2 is defined as:

$$p_2 = f(\begin{bmatrix} a \\ p_1 \end{bmatrix}^T V_{[1:d]} \begin{bmatrix} a \\ p_1 \end{bmatrix} + W \begin{bmatrix} a \\ p_1 \end{bmatrix}) \tag{5.4}$$

Intuitively, we find that the RNTN slice tensor model, captures a specific type of composition.

5.2 CNN : Convolutional Neural Networks

Convolutional neural networks were originally designed for the application in Computer Vision by Le Cunn et. al in 1998 [23]. But, they have shown promising results in several NLP tasks. In this work, Yoon Kim in 2014 [21], uses pre-trained word vectors along with CNN for several sentiment classification tasks. It was reported that the system performed exceedingly well, using a simple model with little hyper-parameter tuning.

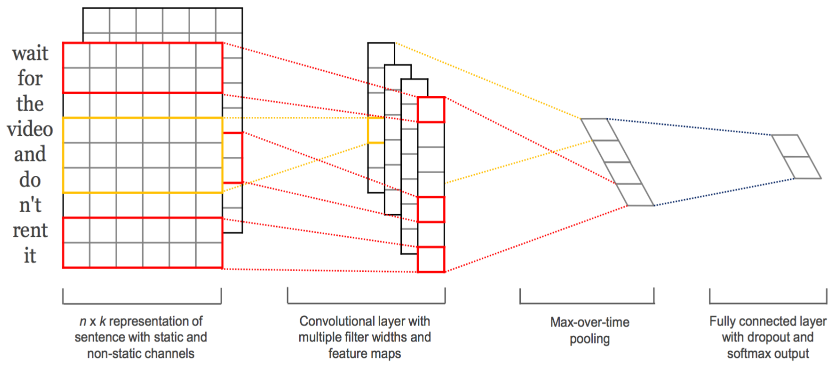

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 5.4: CNN Model architecture with two channels for an example sentence

We start with the model architecture Figure 5.4, x_i, a k dimensional vector for i-th word in the sentence. A sentence of length n (padding if less than n) is represented as:

$$x_{1:n} = x_1 \oplus x_2 \oplus \cdots \oplus x_n \tag{5.5}$$

$\oplus$ is the concatenation operator. In general, for a given concatenation of words: $x_{i:i+j}$, a convolutional operation involves application of a *filter* w over a window of h words producing a new feature c_i:

$$c_i = f(w.x_{i:i+h-1} + b) \tag{5.6}$$

where b is the bias term and f is a non linear function such as hyperbolic tangent. This filter is then applied to each possible window to produce a *feature map*.

$$c = [c_1, c_2, \cdots, c_{n-h+1}] \tag{5.7}$$

We then apply the maximum -over time pooling operation [8] to c and choose the maximum value, $\hat{c} = maxc$ as the feature corresponding to this particular filter. This helps to capture the most important feature per feature map. Overall, the model uses many filters to obtain multiple features. These features form the pen-ultimate layer and they are then fed to a full connected soft-max classifier layer , which outputs the probability distribution over sentence class labels.

Also for regularization, a dropout was implemented on the pen-ultimate layer with constraint on l-2 normalization of weight vectors. It helps prevent co-adaptation of hidden units. For the purpose, a proportion of the hidden units are dropped out or set to zero during forward back propagation step. Or given the penultimate layer of max-pooled features, with m filters: z = $[\hat{c}_1, \cdots, \hat{c}_m]$, instead of using

$$y = w.z + b \tag{5.8}$$

The output unit y in forward propagation dropout uses

$$y = w.(z \circ r) + b \tag{5.9}$$

Where, $\circ$ is the element-wise multiplication operator and r is the masking vector of Bernouli random variables with probability p of being 1. Gradients are forward propagated through the unmasked units. And during test time, the weights are scaled up by a factor of p, such that $\hat{w} = pw$ and $\hat{w}$ is used to score unseen sentences. Also, a re-scaling of w using l2 normalization is also performed.

5.3 LSTM : Long and Short Term Memory Networks

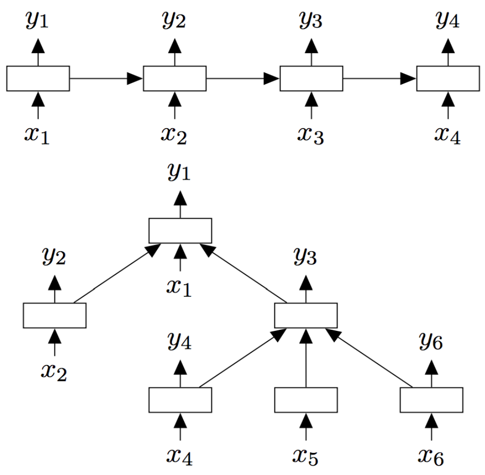


Figure 5.5: **Top:** Chain Structured LSTM; **Bottom:** Tree-Structured LSTM

LSTM's or Long and Short Term Memory network is a type of recurrent neural network architecture, which have shown extremely good results on sequence modeling tasks. They were first introduced by Sepp Hochreiter and Schmidhuber Jurgen in

1997 [19]. In 2015, Kai et. al [40] propose Tree-LSTM, a generalization of LSTMs to tree structured network topologies Figure 5.5.They are inspired to model the view, to combine the words into phrases as in natural language. Tree structured LSTM's are currently out-performing all the baselines for semantic relatedness task and sentiment classification task.

The problem found out, in the RNN application over long distance sequence modeling is that during training time, the gradient vector components can grow or decay exponentially over long sequences. This problem is commonly known as vanishing or exploding gradient. This makes it difficult for an RNN model to learn long distance co-relations in a sequence. LSTMs address this problem, by introducing a **memory cell**. The task of a memory cell is to preserve states over long periods of time.

Kai, et. al [40] provides the two extensions to the basic LSTM structure: the *Child-Sum Tree-LSTM* and *N-ary Tree-LSTM*. Both variants allow for richer network topologies, where each LSTM units can incorporate information from multiple child units.

Each Tree LSTM unit consists of input and output gates i_j and o_j a memory cell c_j and hidden cell h_j. The updates of the gating vector depends on several child units. Instead of a single forget fate, a Tree-LSTM unit consists of one forget gate f_{jk} per child unit k. This allows for selectivity of information from each child. Each Tree Structured LSTM takes a word vector x_j as input vector. The input word at each node depends on the tree structure used for the network. For a dependency tree based LSTM, each node in tree takes the head word as the input, whereas, for constituency tree, the leaf node take the corresponding word vector as input.

Given a tree, C(j) are set of children of node j. Thus, transition equations for **Child-Sum Tree LSTM** and $k \in C_j$ are:

$$\tilde{h}_j = \sum_{k \in C(j)} h_k \tag{5.10}$$

$$i_j = \sigma\big(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\big) \tag{5.11}$$

$$f_{jk} = \sigma\big(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\big) \tag{5.12}$$

$$o_j = \sigma\big(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\big) \tag{5.13}$$

$$u_j = \tanh\big(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\big) \tag{5.14}$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k \tag{5.15}$$

$$h_j = o_j \tanh c_j \tag{5.16}$$

Intuitively, the parameter matrix in above equations can be treated as encoding correlations between the component vectors of the Tree-LSTM unit, the input x_j , and the hidden states h_k of the unit's children. Child-sum LSTM's are well suited for trees whose branching factor is high, or whose children are un-ordered. Thus, it's a good choice for a dependency trees.

On the other hand N-ary Trees (not discussed in detail here), are good where the branching factor is known and at most N or the children are ordered.

The sentiment classification models using Tree LSTM models aim to predict labels $\hat{y}$ from a set of classes y for some subset of nodes in tree. For example: Phrase spanned by the node. At each node j, thelabel $\hat{y}_j$ given the input $\{x\}_j$ observed at nodes in the subtree rooted at j is predicted by a softmax classifier that takes one hidden state h_j at the node as the input:

$$\hat{p}_\theta(y|\{x\}_j) = softmax\big(W^{(s)}h_j + b^s\big) \tag{5.17}$$

$$\hat{y}_j = \arg\max_y \hat{p}_\theta(y|\{x\}_j) \tag{5.18}$$

The cost function is then the negative log-likelihood of the true class labels $y^{(k)}$ at each labeled node.

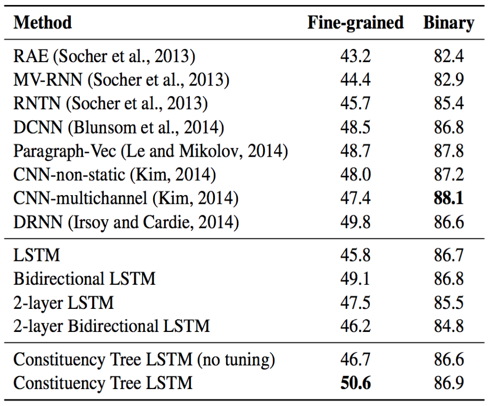
$$J(\theta) = -\frac{1}{m}\sum_{k=1}^{m} log\hat{p}_\theta\big(y^{(k)}|\{x\}^k\big) + \frac{\lambda}{2}\| \theta \|_2^2 \tag{5.19}$$

where, m is the number of labeled nodes in the training set and k indicates the k-th labeled node.

5.4 Comparative Evaluation

The current state of the art for fine grained or five class sentiment classification is with Tree based LSTM as discussed above. In the paper [40], the authors provide a comparative evaluation of all the methods which were state of the arts over a period of time, few of which have also been covered in current literature.

The data-set used for evaluation is the **Stanford Sentiment Treebank** [39] and the experiments are conducted on two types of categories, **Fine-grained** implies classification over five classes, positive, negative, slightly positive, slightly negative and neutral and binary or coarse grained implies positive and negative sentiment classes. As, we can see in the Figure 5.6 the current benchmark for the fine grained category is 50.6% obtained by the Constituency Tree LSTM with tuning. Whereas, the model proposed by Kim et. al [21] achieves the state of the art in overall binary classification with 88.1% accuracy over test data.



Method	**Fine-grained**	**Binary**
RAE (Socher et al., 2013)	43.2	82.4
MV-RNN (Socher et al., 2013)	44.4	82.9
RNTN (Socher et al., 2013)	45.7	85.4
DCNN (Blunsom et al., 2014)	48.5	86.8
Paragraph-Vec (Le and Mikolov, 2014)	48.7	87.8
CNN-non-static (Kim, 2014)	48.0	87.2
CNN-multichannel (Kim, 2014)	47.4	**88.1**
DRNN (Irsoy and Cardie, 2014)	49.8	86.6
LSTM	45.8	86.7
Bidirectional LSTM	49.1	86.8
2-layer LSTM	47.5	85.5
2-layer Bidirectional LSTM	46.2	84.8
Constituency Tree LSTM (no tuning)	46.7	86.6
Constituency Tree LSTM	**50.6**	86.9

Figure 5.6: Test set accuracies on Stanford Sentiment Tree-bank Dataset.

6 Concatenated Word Representations for Sentiment Classification

6.1 Problem

We aim to utilize four different word representations, **Bag of words** [45], **Word2vec** [26], **Polyglot** [33] and **Glove** [32]. The latter three make use of semantic word vector spaces and generate effective sentence or phrase level representations. We discuss various concatenated settings and strategies to extract sentence vectors from word vectors and compare results. We also explore use of in-domain, out of domain and mixed domain data to train word vectors. Our overall aim is to classify movie review phrases and sub-phrases, into five sentiment classes *viz,* positive, negative, neutral, slightly positive and slightly negative. The motivation of the work is a Kaggle contest[1].

6.2 Experiments - Word Vector Concatenation

For the experiments, we also used the traditional Bag of words (**B**) representation, along with the above mentioned representations: Word2Vec-skip (**W**), Polyglot (**P**) and Glove (**G**) and concatenated word vectors. We get following set configurations:

- **Single vectors**: [B]; [W]; [P]; [G].
- **Double vector concatenation**: [B|W]; [W|P]; [P|B]; [W|G]; [P|G]; [B|G]
- **Triple vector concatenation**: [B|W|P]; [W|P|G]; [P|B|G]; [B|W|G]
- **All vectors concatenation**: [B|W|P|G]

Our experimental setup consisted of a single 64-bit Intel Xeon based server with 24 virtualized (12 dedicated) cores via hyper-threading and 64 GB RAM, running Ubuntu Linux 12.04.1. We choose three different domains: **In-domain** data originally from Rotten Tomatoes and provided by Kaggle[2] consisting of 156,060 phrases for training with each having one of five sentiment labels: positive (0), slightly positive (1), neutral (2), slightly negative (3) and negative (4) and 66,292 test phrases; **Out-domain** data

[1]`https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews`
[2]`https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/data`

comprising of Google News text[3] (2.7 GB) for training word vectors.; **Mixed-Domain** data with domain adaptation comprising: Amazon Multi-domain sentiment dataset[4] [5] (Books and DVD reviews) + Google News (Out-Domain) + Kaggle Movie Reviews (In-Domain training reviews), total 3.6 GB in size.

To **train word vectors**, we lower-case the phrases and train using off the shelf trainers available for Word2Vec, Polyglot and Glove, with word vector dimensionality of 100, 64 and 50. For **extracting phrase vectors** from word vectors, we propose two strategies. *First*, substitute word counts in bag of words representation with the average of word vectors obtained via neural network models for a given word. *Second*, a phrase vector can also be computed by taking a centroid of all the word vectors within a given phrase. We also experimented with the effect of removal of commonly occurring stop words in English, since some of our training samples had only a stop word in them. For bag of words, we experimented with both count-based and tf-idf based representations and interestingly found that for our use-case, the count based representation worked better. Thus, we report results using the same. Once all 15 data-sets are generated for respective domains, the phrase vectors are normalized using L2 normalization available in scikit-learn's *preprocessing* module. Next, we use these phrase vectors as input to a *liblinear* [14] based SVM implementation provided by sci-kit [31].

It is interesting to point here that, initially, the test, development and training data sets were divided for computing the SVM hyper-parameters which best fit the data, using grid search cross validation. We achieved different results for the randomized test data folds (taken as a subset of train data) during cross validation. Hence, other measures like the F-score are not reported in the results. Rather, only the accuracy measures are reported, which have only been computed by submitting to Kaggle system thus exhibiting reproducibility of results. For the same reason, we have shared the code and the data as well. Also, some other experiments revealed the use of random forests with ensemble and neural network based classifier, can improve the results marginally, but on the contrary took a long time to train. The lib-linear version of SVM was eventually adopted for its computational performance improvement over other methods, which reduced the training time from an hour to few minutes. This was necessary because we were conducting a total of 75 experiments in this work.

Finally, sentiment labels over test set are predicted and written in pre-defined output format. To **evaluate the results**, we submit obtained predicted output for phrases in

[3]`http://www.statmt.org/wmt14/training-monolingual-news-crawl/news.2012.en.shuffled.gz`
[4]`http://www.cs.jhu.edu/~mdredze/datasets/sentiment/`

test set to Kaggle Server[5], which gives overall accuracy of prediction over test set. We have provided related code and data as supplementary packages[6].

6.3 Observations

S. No.	Representation	Indomain*	Outdomain*	Indomain	Indomain#	Mixdomain#
1	[B]	0.61175	0.60844	0.61175	**0.61175**	**0.60858**
2	[W]	0.60535	0.61140	0.51789	0.51789	0.51789
3	[P]	0.60516	0.61413	0.51789	0.51789	0.51789
4	[G]	0.59959	0.60448	0.51780	0.51762	0.51762
5	[B|W]	0.62155	0.62116	**0.61244**	**0.60988**	0.58802
6	[W|P]	0.61470	0.62201	0.51789	0.51789	0.51789
7	[P|B]	0.62172	0.62483	0.61178	0.60840	**0.60648**
8	[W|G]	0.61472	0.61905	0.51780	0.51780	0.51780
9	[P|G]	0.61569	0.62018	0.51780	0.51788	0.51788
10	[B|G]	0.62226	0.61697	0.61182	0.44686	0.45545
11	[B|W|P]	**0.62276**	**0.62498**	0.61240	0.60368	0.56937
12	[W|P|G]	0.61896	0.62237	0.51780	0.51789	0.51789
13	[P|B|G]	**0.62353**	**0.62386**	**0.61243**	0.52365	0.52717
14	[B|W|G]	0.62249	0.62143	0.61175	0.52472	0.56241
15	[B|W|P|G]	0.62169	0.62348	0.61235	0.53935	0.56044

Table 6.1: *Accuracy scores: Substituting word counts in BOW with the average of word vectors obtained via neural network models for a given word for in-domain and out-domain (*); Phrase vector computed from a centroid of all the word vectors within a given phrase; not removing (ø) stop words and after removing stop words (#) for in-domain and mixed-domain*

6.4 Result

Accuracy scores results, for first strategy (*), when BOW based word counts are substituted by average word vector scores and second strategy (ø, #), when word vector centroids are computed for each phrase, are tabulated as shown in Table 6.1. The two best results for each case are also highlighted. An interesting thing to note is, how many of our basic representations are exactly equal to the benchmark result[7] published on Kaggle which is: 0.51789. and our best result shows overall improvement of 11%. Additionally, to compare our results with state of the art results, we also trained an

[5]`https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/submit`

[6]`https://bit.ly/icon2015data`

[7]`https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/leaderboard`

RNTN model on in-domain data using Stanford's CoreNLP pipeline [9] to get an overall accuracy of 0.64518 on test data. This is fairly close to our best result of 0.62498, with RNTN recording a minor 2% additional accuracy. Although, it can be argued that the first strategy looses all the possible information gained from learned vectors, as we still employ a BOW model, but on close observation the result of 0.61244 [B|W, Indomain] clearly indicates our centroid based approach performs fairly well. The effect of removal of stop words between Indomain and Indomain$^{\#}$, where both are centroid based representations is pretty evident as the accuracy decreases in general. Domain adapted training is effective for the In- and out-domain* cases but not for others. In broad view, [P|B|G] performs the best. Overall training time, for (*) tasks is lesser than (ø, #) tasks.

6.5 Discussion and Future Work

In this work, we studied usage of unsupervised word embeddings and their concatenated representations learned using deep neural network based models to efficiently represent a phrase and its sub-phrases. We conclude, that we can substitute our vector concatenation approach with that of RNTN for learning word vector representations. Our approach is more effective as it doesn't require a hand labeled sentiment values at phrase and sub-phrase level to learn word and phrase vector representations. Our work is equally useful in a multi-lingual setting where such corpus is not available.

We also found that bag-of-word is the best when no combination of vector representation is considered, and combining additional embeddings only improves marginally and requires further investigation for conclusive proofs. One alternative that seems to conclusively beneficial is to keep stop words when combo of word embedding are considered, and that mix domain training may not be beneficial especially when more sophisticated embeddings (other than bag-of-words) are considered. Hence, this work can serve as a good benchmark, for the future design of several sentiment classifiers, which plan to make use of word embeddings.

Also, we aim to explore work done in area of paragraph vectors [22] by using concatenated vectors phrase-wise to form more efficient representations at paragraph and document levels. An interesting future work will be to see how these concatenated vectors capture larger reviews, with multiple sentences. Application of the concepts to other interesting NLP tasks can also be explored like information retrieval and text summarization.

NOTE: *The following work has been currently submitted for review at 12th International Conference on Natural Language Processing (ICON) 2015, Kerela, India.*

7 Moving from Text to Speech: Emotion Recognition

Moving forward in our aim in making computers understand emotions, In this chapter, we present a simple conventional model for emotion recognition in speech. We perform the task of emotion recognition directly from speaker's speech utterances using the well known Semaine datasets[1]. Semaine dataset doesn't have a predefined emotion labeling hence we use the valence and arousal values of the speaker's speech. Using the values, we develop a function, to extract emotion based on five classes. Primarily, the five emotion classes are Happy, Sad, Angry, Neutral and Relaxed.We use the traditional Gaussian Mixture Model - Universal Background Model (GMM-UBM) and the I-Vector (Total Variability Matrix) based models to perform our evaluations. These models were conventionally used for speaker detection and identification and we have modeled them to work for emotion detection.

7.1 Conventional Methods

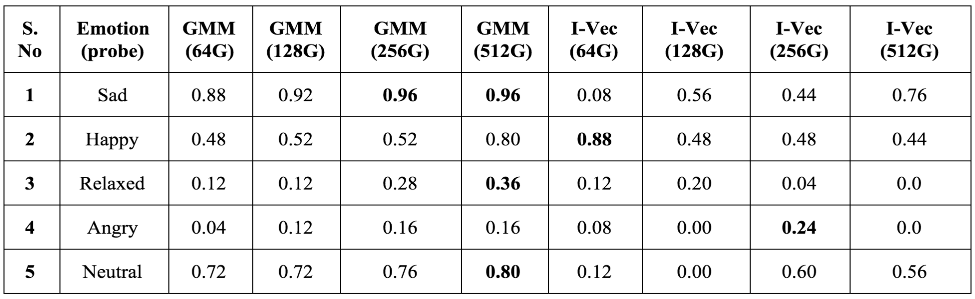
7.1.1 Preprocessing Semaine Data

We using the Semaine clean dataset and first mixing noise and convolved impulse response with all the available speakers speech utterances. Then we pre-process the noisy data in the following way:

1. We start with extraction of MFCC features of dimensionality = 39
2. Next, we extract emotions, based on five classes: Happy, Sad, Angry and Relaxed and Neutral using the Valence and Arousal values available for each speaker's mixed wav file in the *DV.txt (Valence) and *DA.txt. It is to be also noted that we did not try to remove the inherent operator's voice in the data. Thus, our system is also not 100
3. Next, we perform Voice Activity Detection based on linear SVM based VAD classifier which is 93

[1]`http://semaine-db.eu/pages/about/`

4. Next we normalize features, used l2 normalization available in Scikit Learn.
5. We train UBM using all TIMIT[2]. noise mixed and VAD-ed data, in gaussian dimensions 64G, 128G, 256G, 512G separately. We also created a Giant UBM (TIMIT UBM + non - emotion labeled speech FAU data) to get more coverage. We found that out of 18500+ speech utterances, only 4500+ were labeled with emotions, so it was best to use the non emotion labeled data. This still is valid as our UBM is still devoid of any emotion. It helped us to get a more bigger UBM which did not have any emotion specific data from Semaine dataset.
6. **Training and Testing:** Out of the 72 files (mix.wav from each speaker), only 60 of them were retrieved after successful mixing step. Thus, we applied the 80-20 split for train and testing to get 48 training wavs and 12 testing wavs. We also created an emotion wise dataset, where we extracted the MFCC based on the emotion values available to us per frame. Thus, we had this emotion wise dataset consisting of five emotion files in both Training and Testing. Also further, we created test splits based on folds, within the test data. Thus the framewise emotion specific data was divided into different folds: 25, 50, 100 and 250, thus varying the number of test sets and in the end comparing result for both GMM - UBM and I-Vector methods. **NOTE:** We don't perform any preprocessing (WCCN) for the I-Vector Task.



S. No	Emotion (probe)	GMM (64G)	GMM (128G)	GMM (256G)	GMM (512G)	I-Vec (64G)	I-Vec (128G)	I-Vec (256G)	I-Vec (512G)
1	Sad	0.88	0.92	**0.96**	**0.96**	0.08	0.56	0.44	0.76
2	Happy	0.48	0.52	0.52	0.80	**0.88**	0.48	0.48	0.44
3	Relaxed	0.12	0.12	0.28	**0.36**	0.12	0.20	0.04	0.0
4	Angry	0.04	0.12	0.16	0.16	0.08	0.00	**0.24**	0.0
5	Neutral	0.72	0.72	0.76	**0.80**	0.12	0.00	0.60	0.56

Figure 7.1: Accuracy: 25 Fold Test Data: GMM vs I - Vector

7.1.2 Experiments

As indicated, we have divided our test data into several folds and then run experiments based on trained GMM-UBM and I-Vector Models. The UBM of choice is Giant UBM

[2]https://catalog.ldc.upenn.edu/LDC93S1

(TIMIT UBM + FAU (non-emotion labeled)). The dimension of MFCC chosen was: 39. The number of test folds: 25, 50, 100 and 250. And number of Gaussian chosen was 64, 128, 256 and 512. We show the accuracy results for all the experiments in figures 7.1, 7.2, 7.3 and 7.4 and highlight the best score for each emotion probe.

7.1.3 Observations and Results: GMM vs I-Vectors

S. No	Emotion (probe)	GMM (64G)	GMM (128G)	GMM (256G)	GMM (512G)	I-Vec (64G)	I-Vec (128G)	I-Vec (256G)	I-Vec (512G)
1	**Sad**	**0.92**	**0.92**	**0.92**	**0.92**	0.18	0.54	0.44	0.78
2	**Happy**	0.44	0.50	0.60	0.78	**0.84**	0.50	0.48	0.42
3	**Relaxed**	0.06	0.12	0.20	**0.30**	0.08	0.22	0.04	0.06
4	**Angry**	0.04	0.14	0.14	0.16	0.06	0.0	**0.28**	0.0
5	**Neutral**	0.72	0.74	**0.76**	**0.76**	0.16	0.0	0.60	0.52

Figure 7.2: Accuracy: 50 Fold Test Data: GMM VS vs I - Vector

S. No	Emotion (probe)	GMM (64G)	GMM (128G)	GMM (256G)	GMM (512G)	I-Vec (64G)	I-Vec (128G)	I-Vec (256G)	I-Vec (512G)
1	**Sad**	**0.91**	**0.91**	**0.91**	**0.91**	0.24	0.53	0.45	0.78
2	**Happy**	0.44	0.56	0.59	0.72	**0.80**	0.54	0.45	0.44
3	**Relaxed**	0.07	0.14	0.23	**0.32**	0.08	0.17	0.04	0.05
4	**Angry**	0.05	0.14	0.16	0.16	0.07	0.03	**0.25**	0.02
5	**Neutral**	0.73	0.75	0.74	**0.76**	0.23	0.05	0.59	0.46

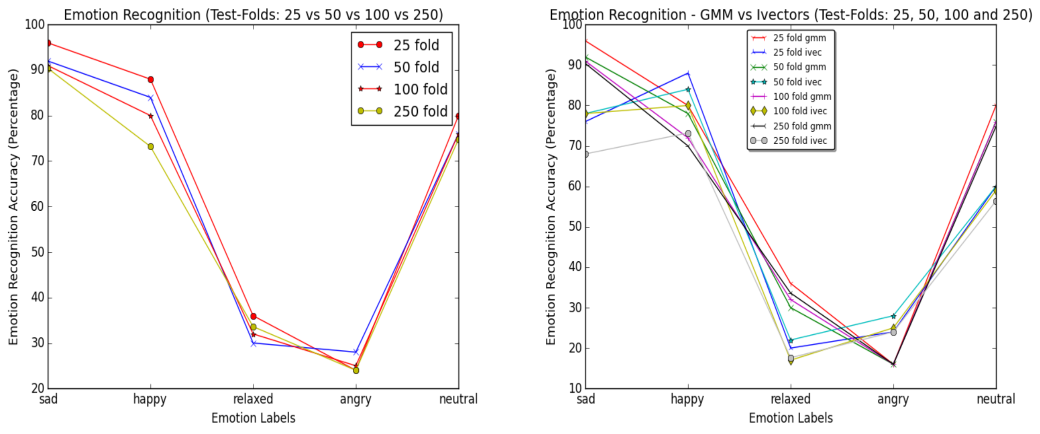
Figure 7.3: Accuracy: 100 Fold Test Data: GMM vs I - Vector

We clearly indicate in figures 7.1, 7.2, 7.3 and 7.4 that for the Semaine dataset, GMM method outperforms the I - Vector methods for all emotion categories except class “Angry”, which is around 28

Closely observing figure 7.5(a) which indicates the best result for each fold, we can see the effects of using the various test fold data on accuracy and conclude that: 25 folds is performing the best. Thus, increasing the number of folds has a rather negative effect. But, this is not true in context of GMM vs I-Vectors since the I-Vectors don’t perform well on low number of folds. We find best I-Vector results for higher number of folds, Figure 7.5(b). As per observations, 50 fold I-Vector performs the best among all I-Vectors roughly for all emotion classes.

S. No	Emotion (probe)	GMM (64G)	GMM (128G)	GMM (256G)	GMM (512G)	I-Vec (64G)	I-Vec (128G)	I-Vec (256G)	I-Vec (512G)
1	**Sad**	0.900	**0.904**	**0.904**	**0.904**	0.28	0.480	0.420	0.680
2	**Happy**	0.492	0.544	0.608	0.700	**0.732**	0.536	0.484	0.448
3	**Relaxed**	0.100	0.132	0.200	**0.336**	0.076	0.176	0.088	0.084
4	**Angry**	0.060	0.120	0.144	0.160	0.052	0.080	**0.24**	0.036
5	**Neutral**	0.704	0.724	0.732	**0.748**	0.244	0.052	0.564	0.352

Figure 7.4: Accuracy: 250 Fold Test Data: GMM vs I - Vector



(a) Best overall accuracy result fold-wise

(b) Best accuracy result for GMM-UBM vs I-Vector for each fold-wise.

Figure 7.5: Emotion wise plots for accuracy w.r.t. folds.

7.2 Deep Neural Networks for Emotion Recognition

Continuing our journey with application of Deep Learning to text for capturing sentiment, In this section we now move to apply the same idea to emotion recognition. We will only highlight one recent work, by Kun Han et. al, [17] who proposed to utilize Deep Neural Networks to extract high level features from raw data and show they are effective for speech emotion recognition. The method supposedly learns exceedingly well the low level features and demonstrates 20% accuracy over the current state of the art methods. The evaluation was performed upon the Interactive Emotional Dyadi Motion Capture (IEMOCAP) database [7].

8 Conclusion

Sentiment and Emotions is what makes us human; it separates us from artificial machines. As an AI researcher and a computer scientist, through this thesis, the task was to bridge this gap between humans and machines a little closer, by understanding in a natural way using algorithms and natural language processing the sentiments and emotions exhibited by humans in raw subjective data (text or speech).

At large, the primary focus was to learn in depth about the deep learning methods for sentiment classification of textual data. We started out with a brief intro to conventional methods for Sentiment Analysis and then we presented several deep learning approaches, which are current state of the art in both language modelling and sentiment classification. Some of the word representations, we discussed in dept were: Bag of Words, Word2Vec, GloVe, Polyglot and Paragraph Vectors. We also learned about the theory and principles of working behind a neural network and largely how they fit in the realms of deep learning and facilitate unsupervised learning and how they are able to automatically learn intricate features from raw un-labeled data.

A comprehensive study involving concatenation of word vectors and building sentence vectors from those concatenated representation and then applying the same to sentiment classification is also presented. Additionally, we tried to gravitate around from text to speech data and evaluated conventional GMM and UBM methods for emotion recognition in speakers speech data in noisy environments. To keep up with the main theme of this thesis, a deep neural network based method for emotion recognition was also mentioned briefly.

A good extension to this study is to naturally blend in the methods for Deep Learning with that of Language processing, as already done in the RNTN models and the Tree-LSTM models which try to capture the syntax of the sentence and also tries to capture long term relationship between phrases, and bring forward a unified learning algorithm which can capture scope of negation better. Also, one can also think about methods which involve capturing sarcasm in review based entities. This is indeed a very tough problem even for humans and developing algorithms for the same will be a challenge in itself.

List of Figures

List of Tables

References

[1] A. Aue and M. Gamon. *Customizing Sentiment Classifiers to New Domains: a Case Study*. In Proceedings of Recent Advances in Natural Language Processing (RANLP), 2005.

[2] Y. Bengio, R. Ducharme, and P. Vincent. *A Neural Probabilistic Language Model*. Advances in Neural Information Processing Systems, NIPS, 2001.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. *A Neural Probabilistic Language Model*. Journal of Machine Learning Research 3, 1137–1155, 2003.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2007.

[5] J. Blitzer, M. Dredze, and F. Pereira. *Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification*. Association of Computational Linguistics (ACL), 2007.

[6] L. Bottou. *From Machine Learning to Machine Reasoning*. arXiv:1102.1808, 2011.

[7] C. Busso, M. Bulut, C.-C. Lee, A. Kazemzadeh, E. Mower, S. Kim, J. N. Chang, S. Lee, and S. S. Narayanan. *IEMOCAP: Interactive emotional dyadic motion capture database*. Language resources and evaluation, vol. 42, no. 4, pp. 335–359, 2008.

[8] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuglu, and P. Kuksa. *Natural Language Processing (Almost) from Scratch*. Journal of Machine Learning Research 12:2493–2537., 2011.

[9] hristopher D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. *The Stanford CoreNLP Natural Language Processing Toolkit*. In Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60., 2014.

[10] S. Das and M. Chen. *Yahoo! for Amazon: Extracting market sentiment from stock message boards*. Proceedings of the Asia Pacific Finance Association Annual Conference (APFA), 2001.

[11] K. Dave, S. Lawrence, and D. M. Pennock. *Mining the peanut gallery: Opinion extraction and semantic classification of product reviews*. In Proceedings of WWW, pp. 519–528, 2003.

[12] L. Deng and Y. D. *Deep Learning: Methods and Applications*. Foundations and Trends in Signal Processing 7: 3–4, 2014.

[13] A. Esuli and F. Sebastiani. *SENTIWORDNET: A Publicly Available Lexical Resource for Opinion Mining*. In Proceedings of LREC, 2006.

[14] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. *LIBLINEAR: A Library for Large Linear Classification*. Journal of Machine Learning Research 9 (2008) 1871-1874, 2008.

[15] C. Goller and A. Kuchler. *Learning task dependent distributed representations by backpropagation through structure.* In Proceedings of the International Conference on Neural Networks (ICNN), 1996.

[16] M. U. Gutmann and A. Hyvarinen. *Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics.* The Journal of Machine Learning Research, 13:307–361, 2012.

[17] K. Han, D. Yu, and I. Tashev. *Speech Emotion Recognition Using Deep Neural Network and Extreme Learning Machine*. In Proceedings of INTERSPEECH, 2014.

[18] G. Hinton. *Learning Distributed Representations of Concepts*. Proceedings for 8th Annual Conference for Cognitive Science Society, Amherest, Mass., 1986.

[19] S. Hochreiter and S. Jurgen. *Long Short-Term Memory*. Neural Computation 9(8):1735–1780., 1997.

[20] C. Jochim and H. Schutze. *Improving Citation Polarity Classification with Product Reviews*. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pages 42-48, 2013.

[21] Y. Kim. *Convolutional Neural Networks for Sentence Classification*. In proceedings of EMNLP, 2014.

[22] Q. V. Le and T. Mikolov. *Distributed Representations of Sentences and Documents*. arXiv:1405.4053v2 [cs.CL], 2014.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[24] Y. LeCun, Y. Bengio, and G. Hinton. *Deep Learning*. Nature Publishing Group, a division of Macmillan Publishers Limited. SN. 0028-0836, Vol. 521, IS. 7553, SP. 436, EP. 444, 2015.

[25] W. McCulloch and W. Pitts. *A Logical Calculus of Ideas Immanent in Nervous Activity*. Bulletin of Mathematical Biophysics 5 (4): 115–133. doi:10.1007/BF02478259, 1943.

[26] T. Mikolov, K. Chen, G. Corrado, and J. Dean. *Efficient Estimation of Word Representations in Vector Space*. Proceedings of Workshop at ICLR, 2013.

[27] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. *Distributed Representations of Words and Phrases and their Compositionality*. In Proceedings of NIPS, 2013.

[28] F. Morin and Y. Bengio. *Hierarchical probabilistic neural network language model.* Proceedings of the International Workshop on artificial intelligence and statistics, pages 246-252, 2005.

[29] B. Pang, L. Lee, and S. Vaithyanathan. *Thumbs up? Sentiment classification using machine learning techniques*. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 79–86, 2002.

[30] B. Pang and L. Lee. *Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales.* In proceedings of ACL, 2005.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. *Scikit-learn: Machine Learning in Python*. JMLR 12, pp. 2825-2830, 2011., 2011.

[32] J. Pennington, R. Socher, and C. D. Manning. *GloVe or Global Vectors for Word Representations*. Proceedings of EMNLP, 2014.

[33] R. Al-Rfou, B. Perozzi, and S. Skiena. "Polyglot: Distributed Word Representations for Multilingual NLP." In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*. Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 183–192.

[34] E. Riloff and J. Wiebe. *Learning Extraction Patterns for Subjective Expressions*. In Proceedings of EMNLP, 2003.

[35] F. Rosenblatt. *The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain*. Psychological Review 65 (6): 386–408. doi:10.1037/h0042519. PMID 13602029, 1958.

[36] B. Scholkopf and A. Smola. *Learning with Kernels*. MIT Press, 2002.

[37] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. *Semantic compositionality through recursive matrixvector spaces.* In Proceedings of EMNLP, 2012.

[38] R. Socher, C. Lin, A. Y. Ng, and C. Manning. *Parsing Natural Scenes and Natural Language with Recursive Neural Networks.* In Proceedings of ICML, 2011.

[39] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, and C. Potts. *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*. Conference on Empirical Methods in Natural Language Processing (EMNLP), 2013.

[40] K. S. Tai, R. Socher, and C. D. Manning. *Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks*. arXiv:1503.00075v3 [cs.CL], 2015.

[41] R. M. Tong. *An operational system for detecting and tracking opinions in on-line discussion*. In Proceedings of the Workshop on Operational Text Classification (OTC), 2001.

[42] P. . Turney. *Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews*. In Proceedings of the Association for Computational Linguistics (ACL) pp. 417–424, 2002.

[43] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. b, 1975.

[44] H. Yang, L. Si, and J. Callan. *Knowledge Transfer and Opinion Detection in the TREC2006 Blog Track.* In Proceedings of TREC, 2006.

[45] H. Zellig. *Distributional Structure*. Word, 10(23): 146-162., 1954.