Univeristy College London

Department of Computer Science

Deep learning for multi-class document level sentiment analysis

Academic supervisor: Dr. Daniel Hulme
Industry supervisor: Dr. David Corney

Author:
Miljan Martic

September 2015

Acknowledgments

Firstly, I would like to thank my academic supervisor, Dr. Daniel Hulme, for his continuous support during the project. The ideas and inspiration that came from our talks and his excellent industry insight helped greatly in shaping the final outcome of this project.

Secondly, many thanks to my internal supervisor, Dr. David Corney from Signal, as well as Dr. Miguel Martinez-Alvarez, whose inputs and support proved invaluable in every stage of the project.

Furthermore, my great appreciation goes to Signal and everyone who works there, as they always did their best to make me feel welcome and as part of the team. Moreover, Signal was kind to provide me with a huge amount of high quality news data which I used for analyzing a number of real-time events and evaluating my system.

My thanks also go to everyone involved with Chainer Python project for building an amazing neural networks library and prompt replies on the forum that helped me in tracking down tricky bugs.

To my parents

Abstract

The field of sentiment analysis (SA) has generated a large amount of interest in recent years, from both academia and industry. Recently, big leaps were made in this area, thanks to the advances in neural networks (NNs) and deep learning. In this thesis we explore these ideas, and present two novel deep learning architectures, the Adaptive Switching Convolutional NN (AS-CNN) and Augmented Recursive NN (ARecNN), that aim to tackle the problem of document level binary and fine-grained SA.

Working with the Stanford Sentiment Treebank (SSTb) dataset, the main contributions of this thesis are two novel state-of-the-art models with excellent generalization capabilities and multi-domain applicability. The presented models are able to deal with a vocabulary of more than one million words, while maintaining an impressive accuracy, which is a considerable improvement when compared to current state-of-the-art approaches. Moreover, we report major improvements over conventional baseline methods, of 4% in binary case, and 3.5% in fine-grained case. Furthermore, we also outperform a number of state-of-the-art neural network models, while maintaining the distinctive cross-domain generalization advantage. Finally, we also present a number of innovative techniques for dealing with NN training on large corpora, as well as performing searches in their hyperparameter space.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we take a quick look at the problem being discussed in this dissertation, the reasons and motivations behind it, as well as usefulness and applications of current solutions. We also define the dissertation aim and objectives which have helped guide and define the research and outcomes of this dissertation. Finally, a quick overview and further directions of this report are given.

1.1 Sentiment analysis problem

With the proliferation of Web 2.0 in the last two decades, and the ever increasing amount of data available online, the ability to understand emotional backing underpinning various news, blogs, social media postings, product reviews and other variety of textual content, has become ever more important. All of the aforementioned content can be classified as having either **positive, negative** or **neutral sentimen**t on the **document level**, with possible different levels of granularity of those classes e.g. for positive content we could have very positive or slightly positive. Moreover, sometimes it is also important to understand at who or what is the sentiment being directed, thus discerning sentiment with respect to different *aspects* (entities) mentioned in some content.

The problem of *Sentiment Analysis (SA)* is that of understanding syntactic and semantic features in the content that define belonging to one of the described classes. There are multiple ways of determining this, and the most successful techniques for a number of years have been machine learning approaches [80].

1.2 Commercial setting and applications

SA has been the topic of commercial interest for a number of years [80]. The views expressed through a variety of sources can provide invaluable insight about user opinions on product launches [57], as well as marketing campaigns and brand positioning [39, 100]. Moreover, SA can help governments and companies understand and predict things like consumer behavior [108], outcomes of political debates [102], and even stock market movements [14].

At a London AI start-up Signal[1], where this research was done, the users of Signal's market intelligence platform were constantly interested in using sentiment mining software like the one developed in this project, for tasks such as following their brand perception, getting aggregated user feedback on their product campaigns and understanding sentiment time series for stories they cared about. Therefore, Signal is planning to expand functionality of their platform by adding sentiment analysis in the near future, to allow informing users about sentiment spikes on topics they are following in real time, as well as provide users with automated short summaries on the reasons behind these sentiment movements. In the words of Signal's Head of Research, Dr. Miguel Martinez-Alvarez:

> "Our customers have realized the great potential sentiment analysis can have on their business, be that for tracking their brand position and recent campaigns or understanding the movement of financial markets, the ability to see and predict mass population sentiment in real time is providing them with a competitive edge. "

1.3 MSc dissertation objectives

In the past few years, the interest in neural networks and deep learning research has grown at an unprecedented scale. Many areas of machine learning research have benefitted from this, as a big number of classical benchmarks, in areas such as natural language processing and machine vision, have been vastly outperformed by deeper architectures trained on modern GPU powered hardware.

SA has also been affected by this deep learning renaissance, where convolutional neural networks (CNNs) and recursive neural networks (RecNNs) have

[1] http://signal.uk.com/

achieved new state-of-the-art results on a number of relevant benchmarks. Hence, this, along with a number of more intricate arguments discussed in chapter 2.2.3, is the main reason behind the aim and objectives of this thesis.

The aim of this masters dissertation is to:

"Objectively analyze and implement current state-of-the-art deep learning approaches to SA and propose potential improvements of current techniques."

The objectives of this masters dissertation are:

- Review current state-of-the-art deep learning approaches for binary and multi-class document level SA.
- Implement and evaluate a novel deep convolutional neural network architecture for SA.
- Implement and evaluate a novel deep recursive neural network architecture for SA.
- Examine the proposed approaches, evaluate on multiple real-world datasets and propose possible future research directions.

1.4 Chapter summary

In this chapter we looked at what constitutes a SA problem, the various use cases for such analysis, as well as future industry applications of this research. The rest of the dissertation is structured as follows. Chapter 2 will give a current theoretical overview of the field, starting with classical approaches and moving onto more recent deep learning approaches. In Chapter 3 we introduce two novel NN architectures, and describe our design process and the architecture of the implemented models. Chapter 4 looks at different evaluation metrics and tests the implemented models on various datasets. Finally, Chapter 5 outlines some future research directions and reflects upon the achievements and work performed in this dissertation.

Chapter 2

Literature review

This chapter aims to give an overview of the approaches to sentiment analysis problem, by first looking at the classical approaches that have been widely used and applied, and then focusing on newer deep learning ideas, and their applications to the problem of SA. As stated in [86], the approaches to SA are split between machine learning approaches and lexicon-based approaches. Moreover, machine learning techniques applied cover both supervised and unsupervised learning approaches, whereas lexicon-based techniques can be dictionary or corpus based. In this report the focus will be solely on machine learning approaches as they represent the current state-of-the-art, while noting that lexicon-based approaches do show promise when combined with standard machine learning techniques.

As Liu discusses in [81] the mains tasks in sentiment analysis are document-level sentiment classification, aspect-based SA and sentiment summarization as well as supporting tasks of subjectivity detection, analysis of comparative opinions and opinion spam detection. The deep learning models we describe in Chapter 3 deal with the problem of document-level sentiment classification, hence that remains the focus of this literature review as well, with occasional mentions of applicability of outlined models to other SA tasks.

The chapter starts by discussing classical supervised learning approaches, then going over semi-supervised and unsupervised approaches. Following that, a separate section discusses both supervised and unsupervised neural networks and deep learning models, as well as their specific application in SA. Finally, we describe most common ways of encoding textual content into both discrete and continuous vector representations.

2.1 Classical approaches

2.1.1 Probabilistic classifiers

Some of the commonly applied models in SA fall in this group, including Naive Bayes, Maximum Entropy and Expectation Maximization, along with their hybrid combinations (EM-NB etc.).

2.1.1.1 Naïve Bayes (NB)

One of the most often used classifiers in the past couple of decades [77], Naïve Bayes has been applied to the sentiment analysis problem in many different variants. Based on Bayes' theorem, in it's simplest form, it aims to estimate posterior probability of a given sample belonging to each class, and then applying Maximum a Posteriori (MAP) rule to give the most probable classification. The model is naive due to the assumption that all the features are independent as well. Thus, there is no learning involved in this algorithm, but rather the data in itself is the model.

Several common improvements are used to make NB more applicable to real problems. Firstly, given that the probability of a sample belonging to a class is the product of probabilities of each of it's features belonging to a class, due to independence statement, the whole product may become zero if the feature was never seen before. Thus Laplacian or some other kind of smoothing is usually used to account for this anomaly. Secondly, NB originally was designed with Gaussian distribution of features in mind, but given that sometimes this is not the case, Bernoulli and Multinomial distributions are commonly used now as well [77].

In sentiment analysis, Pang and Lee [109] did one of the first studies in SA using NB with POS tags, unigram and bigram features, which provided a solid benchmark that proved not so easy to beat in the following years. NB was also successfully applied in the case of aspect-based multilingual SA by Boiy and Moens [12]. Reyes et al. [114] showed the possibility of applying NB model to detecting irony in customer reviews. NB was also applied to determining market intelligence [143] and predicting market movements [14]. Kang et al. [63] presented a modified NB algorithm that aims to reduce the disparity between positive and negative classification accuracy of online reviews. In cross-domain sentiment analysis, Tan et al. [131] propose a modified NB model using expectation maximization (EM), which outperformed base-

line transfer learning models such as Naïve Bayes Transfer Classifier (NTBC). Finally, Bravo-Marquez et al. [15] have also shown a viable use of NB classifiers for large scale sentiment mining tasks.

2.1.1.2 Logistic Regression (LR)

This model comes from the family of generalized linear methods, and can be seen as probabilistic, due to the fact that is uses logit transformation to output the probability of a given instance belonging to a class. It is a binary classifier, but it can also be used in multi-class case by employing one-vs-all approach. Given a previously unseen instance x, we calculate the linear combination of our data with an intercept

$$\hat{x} = w_0 + w_1 x_1 + \cdots + w_n x_n,$$

and then plug the result into logistic function to obtain the probability

$$p = \frac{1}{1 + e^{\hat{x}}} = \frac{1}{1 + e^{(w_0 + w_1 x_1 + \cdots + w_n x_n)}}.$$

In order to learn the weights, standard gradient based algorithms can be used, such as gradient descent, and for the loss function, mean squared error (MSE) or log-loss are used.

In [13] Bollegala et al. showed application of logistic regression for state-of-the-art binary document level cross-domain SA, by using a sentiment sensitive thesaurus. In finer grained SA, with five classes, Thelwall et al. [132] presented excellent results using logistic regression on short social media texts.

2.1.1.3 Maximum Entropy classifier (MaxEnt)

Also known as multinomial logistic regression, this model has been shown to outperform a Naïve Bayes classifier in some cases, as it makes no assumptions about feature independence [9, 101]. In MaxEnt model features form constraints of the model, each with their corresponding weight $\lambda_{i,y}$, and the aim is to estimate the conditional probability of a sample $\mathbf{x}$ belonging to class y under constraints:

$$P(y \mid \mathbf{x}) = \frac{1}{\mathbf{Z}(\mathbf{x})} \exp\left\{\sum_i \lambda_{i,y} \, f_i \, (\mathbf{x}, y)\right\},$$

such that $\mathbf{Z}(\mathbf{x})$ is the normalization factor, and $f_i \, (\mathbf{x}, y)$ is a feature function

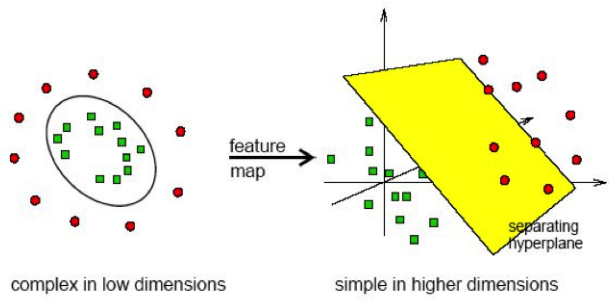


Figure 2.1: Kernel trick [21]

$$f_i(\mathbf{x}, y) = \begin{cases} 1, & \text{if } \mathbf{x} = \mathbf{x_i} \; and \; y = y_i \\ 0, & \text{otherwise} \end{cases}.$$

Using numerical optimization methods such as iterative scaling algorithm, we can obtain optimal weight parameters which maximize the entropy of the model.

Pang et al. [109] studied the performance of MaxEnt classifier on document level binary SA. On the other hand, Boiy and Moens [12] examined the applications of MaxEnt to aspect-based SA. Finally, MaxEnt models have also been shown to perform well in ensemble based approaches as shown in [139, 141].

2.1.2 Kernel classifiers

Kernel methods, and more specifically, support vector machines (SVM), are by far the most used method for the sentiment analysis problem. The SVM classifier projects the data into a higher dimensional space using a feature map, where an optimal separating hyperplane is found, which maximizes the margin between the two given classes. This is shown in Figure 2.1. In a multi-class case, one-vs-all approach is usually applied.

The power of SVM also lies in the fact that it only requires a number of support vectors to construct the maximum margin, as shown in Figure 2.2.

As Pang et al. have shown [109], SVM outperforms Naïve Bayes and MaxEnt models using n-gram and TF-IDF based features. Moreover, Saleh et al. [119] have done similar study while experimenting with different weighting schemes such as binary occurrence and term occurrence, thus obtaining state-of-the-art 85.37% accuracy on the Pang corpus. Another study by Mullen and Collier [98] showed that, using a hybrid SVM model, with a mix of n-gram, real valued and topic information features, one can obtain excellent accuracy

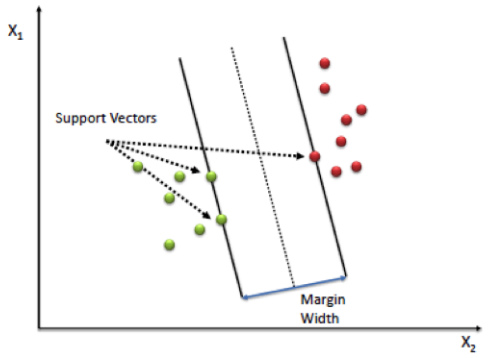


Figure 2.2: Maximum margin and support vectors [120]

on a variety of topics. An interesting study on determining the sentiment of conditional sentences such as "*if your Nokia phone is not good, buy this great Samsung phone*" was conducted by Narayanan et al. [99], by using SVM with a Gaussian kernel.

There have been a number of studies where SVM models have been compared to artificial neural networks (ANN), due to both of them being considered state-of-the-art approaches. Moras et al. [97] performed a study comparing ANN performance to SVM for the task of document level sentiment classification, and concluded that ANN's "*produce superior or at least comparable results to SVM's*". Furthermore, Ghiassi et al., [39] in their 2013 study, compared the effectiveness of a dynamical ANN architecture with an SVM classifier, using "*Twitter-specific lexicon with brand-specific terms for brand-related tweets*". They concluded that the neural network architecture outperforms SVM approaches due to better recall, which is very important in use cases such as online brand management. Thus, one of the reasons this report focuses on ANN models and deep learning is because of the potential of ANNs demonstrated in the aforementioned research.

2.1.3 Tree based classifiers

This category includes Decision Tree (DT) classifiers as well as Random Forest (RF) models.

2.1.3.1 Decision Trees

This classification technique creates a series of questions based on the attributes (features) of the data. This splits the data into subgroups, forming

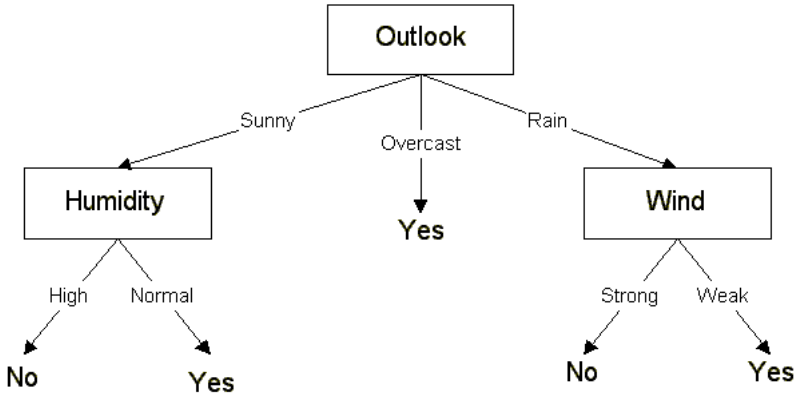


Figure 2.3: Binary decision tree example for deciding to go out [138]

a tree, where decisions lead over internal nodes (subgroups) to the leaf nodes, which represent class labels [17]. Given a data set, many different decision trees can be constructed, thus the aim is to construct the most informative DT, in a reasonable amount of time. Some of the most popular greedy DT algorithms include ID3, C4.5, CART and MARS. There are different ways of forming the tree, such as doing a binary split on the attribute, or a multi-way split. This depends on the value of such an attribute, and these values can be discretized or grouped as necessary. Moreover, we need to have a way of determining how well does a condition test (split) perform. The way this is done is to measure the degree of "impurity" before and after the split, that is in parent and child nodes. The higher the difference between these two set, the better the split. Some of the measures used for this include Entropy, Gini impurity and Variance reduction.

A recent study by Geva and Zahavi [38], about forecasting stock market movements, presented a viable use of decision trees built using genetic algorithm, in predicting market sentiment from news. In the paper of Wang et al. [139] DTs were used as part of the ensemble methods such as bagging and boosting, yet it was shown that they were outperformed by classifiers such as SVM and NB. In [82] Liu et al. developed a two phase algorithm to determine the helpfulness of an online review. They used bootstrap aggregation with fast decision tree classifier which outperformed multi-layered perceptron (MLP), simple linear regression and support vector regression (SVR). Reyes and Rosso [114] discussed the use of DT classifier for the purpose of detecting irony in customer reviews, and concluded that, for this problem, DT's give optimal performance on multiple datasets, when compared to SVM and NB classifiers. Finally, Ortigosa et al. [105] experimented with using DTs for

sentiment classification and sentiment change detection in Spanish Facebook comments. They used the C4.5 algorithm, while obtaining 83.13% accuracy which was just a bit short of 83.17% and 83.27% obtained by NB and SVM classifiers respectively.

2.1.3.2 Random Forests

Due to the fact that DTs tend to overfit if they are grown too deep, random forests present a way of reducing the variance of DT classifiers, by incurring a slight increase in bias cost [31]. This is done by constructing multiple DTs on different parts of the training and feature set and averaging them. Firstly, the selection of the training data subsample is done by bootstrap aggregation (bagging) method, which samples randomly from the dataset with replacement and creates a DT for each selected set. Secondly, after the training set is selected, RF classifier performs feature bagging, where a random subset of features is chosen. The optimal combination of training set size and number of trees is usually determined using cross-validation, whereas the feature bagging size is usually $\sqrt{k}$, where k is the total number of features [16]. Finally, given a previously unseen instance, in classification case, the class label is obtained by majority voting.

Coussement and den Poel [23] studied emotional indicators of customer churn behavior using multiple model, and showed that a RF classifier significantly outperforms Logistic regression and SVM models in this use case. Ghose and Ipeirotis [40] use RF model to successfully predict the subjectivity of online product reviews and their effect on product sales. Similarly, random forest models were also shown to be state-of-the-art in detecting spam reviews and comments on social network, outperforming various SVM approaches [22].

2.1.4 Semi-supervised and unsupervised classifiers

With the advent of big data in recent years, and the ever increasing amount of easily available unstructured information on the internet, some researchers looked at ways of extracting sentiment information and training classifiers on large corpora using techniques such as weak and distant supervision, semi-supervised learning and unsupervised learning. This is especially present in social media, where it has been very easy to obtain vast amounts of data that has an underlying sentiment, but where it is very expensive to obtain labels for such data.

Some of the early attempts were described by Pat and Paroubek [106] where they used distant supervision, in the form of emoticons encountered in tweets, to label a large corpus for further supervised classifier training. Furthermore, Hu et al. [58] looked at "*emotional signals*" at post level (emoticons, product ratings, restaurant stars,) and word level (abbreviations, sentiment lexicons), which are then used as regularizers to matrix tri-factorization based unsupervised SA framework.

Xianghua et al. [142] looked at multi-aspect unsupervised SA of Chinese online social reviews, using Latent Dirchlet Allocation (LDA) model for global topic discovery, after which they modeled the local topics and extracted associated sentiment using a sliding context window over the reviews. A notable contribution of this paper was in helping produce multi-aspect fine-grained sentiment labels in unsupervised setting, which are very useful in some commercial settings such as electronic hardware reviews.

Another popular way of doing unsupervised learning for SA is called corpus-based method [83, 136, 147], which uses seed words annotated with sentiment, and then learns the relation between the given corpus and seeds. These methods have been shown to perform well for cross-domain SA problems. Moreover, for multilingual SA problem, Zagibalov and Caroll [144] also achieved excellent results using corpus-based methods.

2.2 Neural networks and deep learning approaches

Neural networks (NNs) are amongst the most popular models applied both in supervised and unsupervised fashion, for a variety of tasks. In their most basic form, NNs consist of a single input layer, one or more hidden layers and an output layer. Each layer consists of neurons (units), and layers have a number of connections S s.t. $S \subset N \times N$, where N is the total number of units in the network. Every connection has a weight associated with it, and these weights are learned, using some (usually gradient based) optimization method. Each unit in a layer represents an activation function, such as identity, sigmoid, tanh etc., which transforms the weighted input from previous layers. The information travels through the network in a feedforward manner (this is slightly different for cyclic NNs, which can also take input from previous epochs), and the last layer outputs the predictions. Based on those predictions, we calculate our loss using an appropriate loss functions and backpropagate the gradient

update to all the weights. This process represents a single epoch, and NNs are usually trained through a number of epochs, until the loss improvements stop varying greatly [1].

In general, NN models can be generally seen as belonging to either *feed-forward (acyclic)* or *recurrent (cyclic)* type of NNs, and models from both groups are discussed in the rest of this chapter. In the following sections we first take a look at history of neural networks, from their inception, to present day, covering the most important milestones and their various applications. Next, a short discussion on what constitutes deep learning and why is it good is followed by more detailed look at most popular deep learning approaches in sentiment analysis.

2.2.1 Brief history of neural networks

The field of deep learning has enjoyed huge success in the last decade. With the advent of cheap GPU powered computation at the beginning of new millennia, many benchmarks in fields of NLP, machine vision, speech recognition and machine translation, among others, have been beaten by deep neural networks. But, neural networks (NNs) have been around for a long time. In their simplest form, where a network consist of one neuron which outputs a linear combination of its inputs, we have linear regression. Thus, these ideas can be traced as far as back to Gauss and 1800s [36].

Modern neural networks have first been introduced in 1950s, by Rosenblatt and his famous Perceptron model [116] (see Figure 2.4), followed by Multilayered Perceptron (MLP) models (e.g. Figure 2.6) in 1970s [60, 61], where many perceptrons were stacked together into layers to create a feedforward neural network, and trained using Group Method of Data Handling (GMDH).

Inbetween 1960s and 1980s work on the now famous *backpropagation (BP)* algorithm has been going on. Based on the principles of dynamic programming and reverse automatic differentiation [47], it allows for efficient gradient update dissemination throughout the network. In 1986 Rumelhart, Hinton and Williams [117] demonstrated an efficient use on BP algorithm in training a multilayered neural network.

On the other hand, during the same time, CNNs were introduced by Fukishima for the first time under the name *Neocognitron* [32, 33, 35]. Very similar to modern CNNs, *Neocognitron* used convolution (filters) and downsampling layers (max-pooling, averaging), which are described in more detail in section

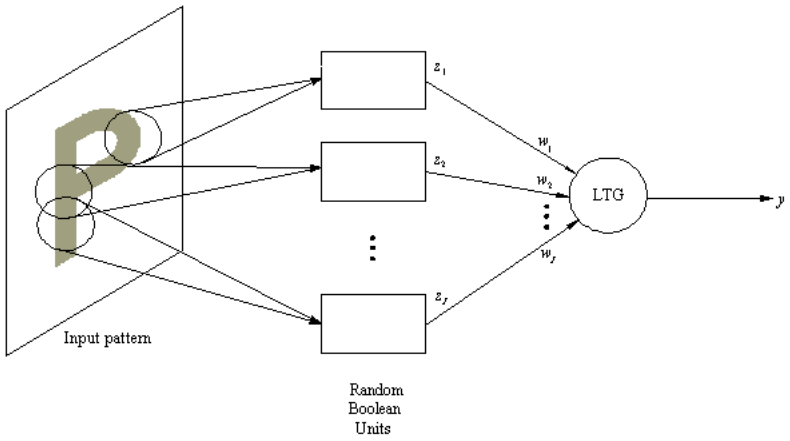


Figure 2.4: Rosenblatt's perceptron model [48]

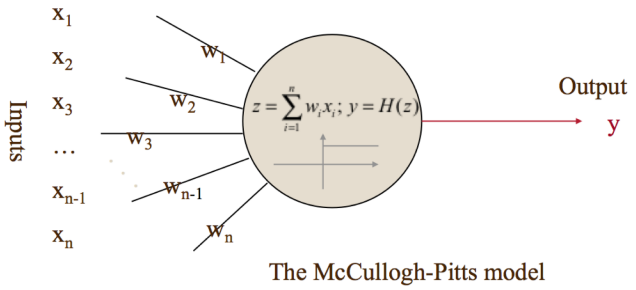


Figure 2.5: Modern neural unit model [145]

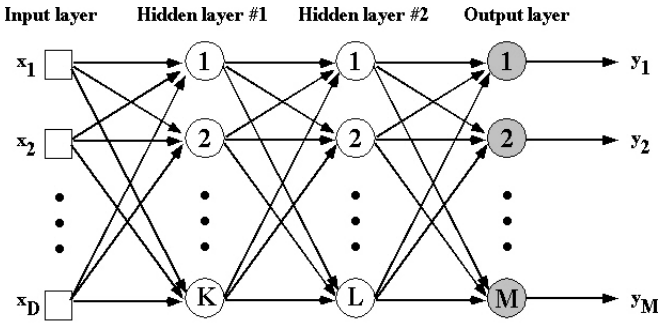


Figure 2.6: Classic multiclass MLP model with 2 hidden layers [115]

2.2.4. Furthermore, another avenue of research was presented by RNNs, as a way of dealing with time delayed events and temporal processing. A number of different architectures were developed, such as Time-Delay Neural Network (TDNN) [72], Hopfield Network [54] and Nonlinear AutoRegressive with eXogenous inputs (NARX) [78], among others, but their performance and generalization were not as good as expected, due to very slow training on hardware available at that time [121]. In the two last decades, with the rapid decline in GPU computation costs, and powered by new biologically inspired ideas [104] such as Long Short-Term Memory (LSTM) [53], RNNs have seen a comeback, especially in the area of NLP, where they have proven to be excellent in modeling languages described by finite state automata [10,137,146], context free languages [3,135], and more recently speech recognition [44,118]. In recent years, LSTM RNNs and max pooling CNNs results have been so ubiquitous that "most competition-winning or benchmark record-setting Deep Learners actually use one of the two [aforementioned] supervised techniques" [121]. This is proven by numerous competitions won by these systems such as ImageNet [70], PASCAL object detection [41], language identification [43], among others. A deeper look at these two types is given in chapters 2.2.4 and 2.2.5, where their application to SA problem is also explored.

On the other hand, a number of stochastic neural networks have also been popularized since the 1980s. In 1985 Hinton and Sejnowski introduced a Boltzmann Machine (BM) , [2], a stochastic recurrent network, with powerful theoretical modeling capabilities, but impractical training requirements. An improvements which alleviated the problem of training, was presented in the form of Restricted Boltzmann Machines (RBMs) [125], which restricted the recurrent connections between hidden and visible units (see Figure 2.7). These networks can then be trained using gradient based contrastive divergence [19]. Finally, RBMs can be stacked into a Deep Belief Network (DBN), so that hidden layer of each part represents a visible layer of the next part. Because of this, the network can be trained greedily, layer by layer, in an unsupervised fashion [51].

2.2.2 Deep versus shallow learning

Over the years there have been many discussions on what constitutes deep learning, and where does *shallow learning* end, and *deep learning* begin. Some authors specify that deep learners are characterized as models consisting of

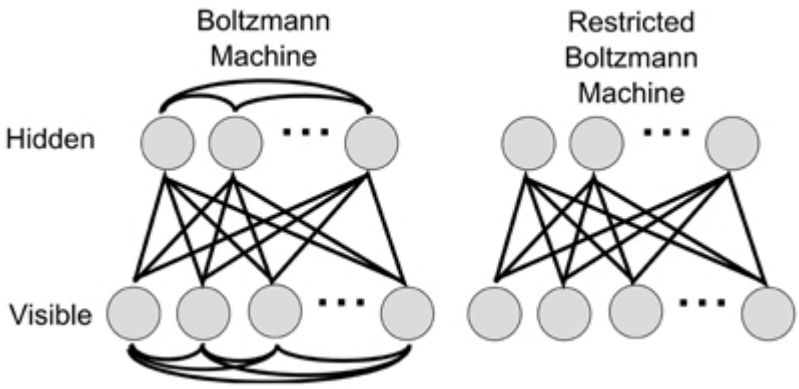


Figure 2.7: BM and RBM comparison [103]

multiple layers of non linear transformations [4]. On the other hand, just by looking at a number of layers, it can be hard to conclude the complexity of learning and credit assignment [94], especially in NNs with temporal elements such as recurrent neural nets, where updates backpropagate through time as well. An alternative way of looking at this issue was proposed by Schmidhuber [121], by observing the *Maximum Assignment Path (MAP)*, which is "the depth of how far backwards credit assignment can move down the causal chain to find a modifiable weight". On the other hand, MAP depth can sometimes be hard to calculate, thus in this report, a deep model is considered to be a hierarchical model with several layers of non-linear transformations, as described in [27].

2.2.3 Why is deep learning good?

The research into neural networks and deep learning is at the forefront of machine learning at the moment, but during the years it has had its ups and downs. Starting from the promise of Perceptron solving the AI problem in 1950s, over inability to effectively differentiate learning function across the network in 1960s, and failures of many applied MLP models in 1970s, the interests and funding in NN research has disappeared many times. During this years many researchers found comfort in the now famous Universal Approximation Theorem [24]. Initially discussed by Kolmogorov in 1956 [69], and later applied to neural networks with sigmoid activation by Cybenko [26] and generalized by Hornik [55, 56], it showed that a NN with a single hidden layer and enough hidden units can approximate any multivariate continuous function with arbitrary accuracy. However, although impressive at first, this result is diminished by the fact that the network has an exponential complexity dependent on the number of variables of the original function, meaning the number of units in

the hidden layer grows exponentially as the error goes to zero [73, 133]. Thus the representational power of this model, though theoretically unlimited, is practically limited. On the other hand, in deep learning, using hierarchical layers, we can represent functions that are computed in several steps of computation. For example, in machine vision, lower layers of the network learn to recognize low level representations in an image (edges and simple shapes), and as the layers progress, the objects being represented in them also grow in complexity, until the final layer resembles a representation of an object being detected.

Another important factor is discussed by Bengio in [5], and it states that the efficiency of deep learning lies in the ability to share low-level representations between different learning tasks and classes. As the *manifold hypothesis* states, high-level concepts lie close to the low-dimensional manifolds, thus having the ability to extract those low-level representations allows for having only small changes in the upper layers, between semantically similar examples, without having to do any updates in lower layers.

Moreover, recently, some new results were presented exploring the effectiveness of deep learning by relating it to one of the fundamental techniques in theoretical physics, the renormalization group. By proving that unsupervised deep learning implements the Kadanoff Real Space Variational Renormalization Group, Mehta and Schwab have shown that "deep learning algorithms may be employing a generalized RG-like scheme to learn relevant features from data" [87].

Now, one may wonder if the same results would apply to bayesian architecture of neural networks, such as Restricted Boltzmann Machines (RBMs). Initial proofs by Le Roux and Bengio [73, 74], as well as subsequent research by Montufar [96], have shown in that, indeed, RBMs and Deep Belief Networks (DBMs) can be seen as universal approximators, where they are able to "approximate any distribution over binary vectors to arbitrary accuracy, even when the width of each layer is limited to the dimensionality of the data" [130].

2.2.4 Convolutional neural networks (CNNs)

CNN is a type of feedforward NN, consisting of hidden *convolution* and *downsampling* layers stacked in an iterative manner, followed by a fully connected output layer, as shown in Figure 2.9. In essence, these networks represent a variation of MLP networks [76] and are biologically inspired by human

visual neurons [85]. They are trained in the same way as classical NNs, by learning weights using backpropagation and a loss function, with major differences being in convolution and downsampling layers.

2.2.4.1 Convolution layer

CNNs were initially created with machine vision in mind, and thus they have an implicit assumption that inputs are 3D matrices of images with *width*, *height* and *depth*, generally corresponding to width, height and RGB value of an image. On the other hand, the third dimension can be seen as the number of *feature maps* the network learns, thus CNNs are also able to operate on matrix encoded text.

The input layer is followed by a convolution layer, where each unit is connected only to a small subset of nodes in the previous layer, instead of each unit being connected to all the units from the previous layer. They operate in a standard way, by computing the dot product of inputs with their respective weights and applying a potential non-linear transformation. By focusing only of parts of the data matrix, we allow for localized feature detection, and by stacking these layers the network can learn features of ever increasing complexity [33]. Practically, convolution is done by having a filter of dimensions $w \times h$, which then moves over the data matrix. We also define the depth parameter d, which defines how many neurons are connected to the same area (in essence, the number of feature maps). With respect to these parameters, we then have *wide* and *narrow* types of convolution, which differs in deciding if connections of convolution nodes $\{c_1 \dots c_c\}$ connected to downsampling nodes $\{s_1 \dots s_c\}$ are zero-padded or not. If they are, we have wide convolution as seen on the right side of Figure 2.8, and if they are not, we have narrow convolution, as shown on the left side in Figure 2.8. Thus, we also have a parameter p, which specifies how long is our zero-padding. Finally, we need to select the size of *stride* s, which defines by how much does our filter move each time, i.e. if stride is one, then we will have big spatial overlapping between convolution neurons, as the filter moves one step each time.

2.2.4.2 Downsampling layer

This layer is used to reduce the increased dimensionality produced by the preceding convolution layer and to reduce overfitting. Again, units in this layer are connected to a subset of units from the previous layer, same as in

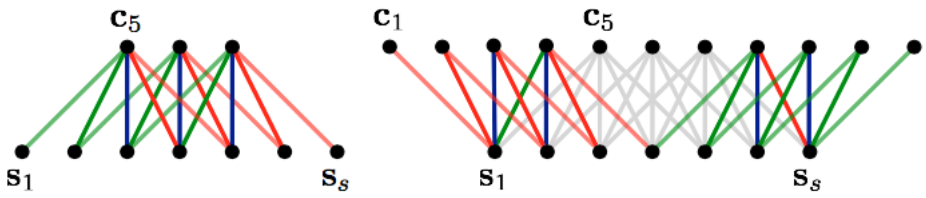


Figure 2.8: Narrow and wide type of convolution [62]

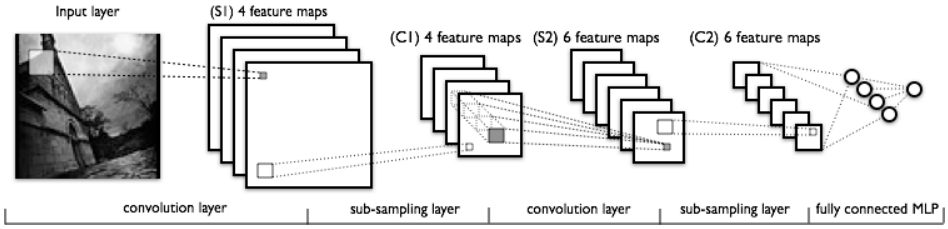


Figure 2.9: Example of CNN architecture - LeNet [75]

convolution layer. The downsampling can be performed in multiple ways, and one of the most popular ones is *max-pooling*. This technique selects the most active unit (the one with maximum output) out of all the input signals into a given downsampling layer unit [4]. An improved version of this is known as *k-max-pooling*, which selected k values instead of just one [62]. Another way of doing downsampling in images is *spatial averaging*, as demonstrated by Fukushima [34] or *average pooling* used by Kalchbrenner [62]. Again, for standard max-pooling, in this layer we have to specify parameters w, h, p and s, whereas the number of features maps d is the same as in the previous layer.

2.2.4.3 Applications in sentiment analysis

The excellent modeling performance of CNNs without any a priori knowledge of syntactic and semantic structures has been demonstrated by Zhang and LeCun [148]. They run two slightly different CNN architectures on a variety of NLP tasks, including SA, where they managed a 97.57% accuracy in binary sentiment classification, and 69.24% accuracy in five class classification on the Amazon reviews dataset.

In [29] the authors propose a deep CNN for SA on short text, such as tweets, by adding another hidden layer for character based features, similarly to [148], on top of word and sentence level embedding layers. Their model obtains excellent performance of 86.4% on Stanford Twitter Sentiment (STS) corpus.

Kalchbrenner et al. [62] presented a multilayered CNN with k-max-pooling named Dynamic CNN (DCNN) for sentence modeling, and applied it to a number of tasks, including SA problem, with great success, while outperforming many state-of-the-art RNN based models with 86.8% accuracy on Stanford Sentiment Treebank (SSTb) dataset.

Most recently, Kim [66] reported state-of-the-art results on multiple tasks, one of which was SA on SSTb dataset, with impressive 88.1% accuracy in binary classification.

2.2.5 Recurrent neural networks (RNNs)

The idea behind RNNs is to allow propagating context information through multiple time steps. RNN is a cyclic network model, characterized by one or more feedback connections, usually in the hidden layer. An example is given in Figure 2.10. Here, at each time step, the input to the hidden layer h_t with a recurrent connection is the output from the previous layer at time step t (in this case input layer), x_t , as well as output of that layer at a previous time-step h_{t-1}. The layer then performs a dot product of these inputs with the corresponding weight matrix and applies a potential non-linear transformation to obtain the output y_t, as in a standard NN.

The way RNNs are trained is using the a modified backpropagation technique called Backpropagation Through Time (BPTT) [140]. In BPTT, the network is *unfolded*, meaning the recurrent layer is replicated up to a certain depth d, which is previously specified, and then the training proceeds using the standard backpropagation algorithm, while providing inputs in a sequential fashion, as the order of training instances now carries meaning. Following an error update that is propagated through the unfolded network, the recurrent weights are averaged, due to the fact that they are tied together. BPTT still has the standard issue of vanishing/exploding gradient that is common with standard backpropagation algorithm, especially due to unfolding, which may require propagation of error much further than in other NN architectures. Furthermore, with gradient updated diminishing over multiple steps, the time needed to train the network also increases dramatically. A way of dealing with this problem in RNNs is by using Long Short-Term Memory (LSTM) units, which are not affected by the problem of exploding/vanishing gradient. Furthermore, BPTT also has a problem of being susceptible to local minima optimization [25].

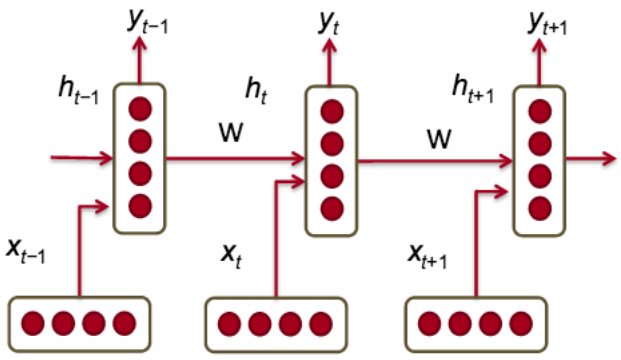


Figure 2.10: RNN unfolded through 3 time steps [95]

2.2.5.1 Bi-directional RNN (BRNN)

BRNN is a type of RNN commonly used for tasks such as part of speech (POS) tagging [65], as well as state-of-the-art phoneme prediction and handwriting recognition [45, 46]. First introduced in [123], a BRNN consists of two hidden layers, one for left-to-right propagation and one for right-to-left propagation. The output y_t is generated by combining the output of both RNN hidden layers. This can be seen useful in cases where, for example, we want to predict a POS tag for a given word, and we want to know the words that preceded it as well as follow it. A deep BRNN can be constructed by stacking multiple BRNN hidden layers together, as seen in figure 2.11. The difference here is that hidden layers starting from $h^{(2)}$, at time step t, will take as input the output from previous layer, which consists of two outputs because it is a bi-directional layer, as well as input from the previous time step $t-1$.

Some of the problems associated with BRNNs are related to the fact that they need a fixed beginning and ending, due to the bi-directional nature of data processing, and are thus unable to operate in a continuous or online setting [79].

2.2.5.2 LSTM

LSTM is a type of complex activation unit, which is used to create different RNN architectures, introduced by Hochreiter [53]. These networks can consist either entirely of LSTM units, or have a mix of standard units and LSTM units. This idea behind using this type of unit is to aid RNNs in capturing aforementioned long term dependencies. Even though RNNs are theoretically able to do that, because of vanishing gradient, it is very hard to train them to

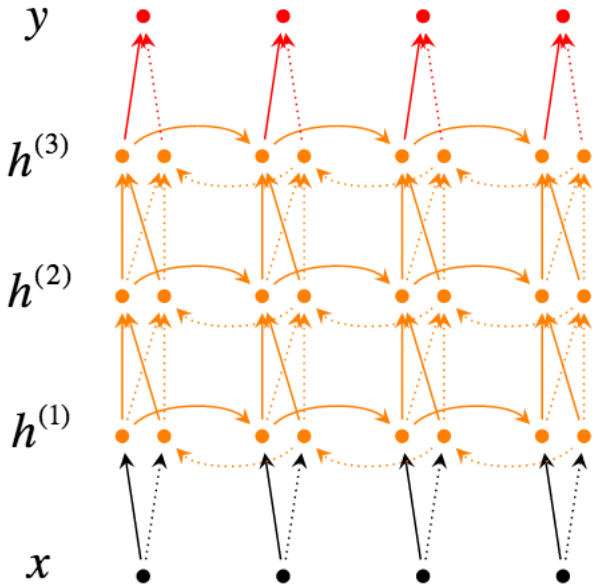


Figure 2.11: A Deep BRNN with three layers [95]

do so. An LSTM unit can remember a value for an arbitrary length of time in a persistent way. As shown in Figure 2.12, each LSTM unit consists of input section, three boolean gates and a memory cell.

Memory cell is a linear activation node at the center of LSTM where the information is being stored. It has a recurrent connection to itself with weight 1, which ensures the gradient can be backpropagated through time without being susceptible to vanishing gradient problem [79].

Input section is a learned sigmoidal unit, used to create a new memory consisting of current input x_t as well as previous hidden state h_{t-1}.

Input gate is a learned sigmoidal unit, which controls whether the new memory created in the input section is allowed to be stored. The way it does this is by calculating the relevance of the current input by using current input x_t , previous hidden state h_{t-1} and the current content of the memory cell.

Forget gate is a learned sigmoidal unit, used to erase the current information stored in the memory cell. Its inputs are also current input x_t , previous hidden state h_{t-1} and the current content of the memory cell. This gate was not part of the original LSTM model, but was subsequently proposed by Gers et al. [37]. By using forget gate, LSTM is able to operate on problems where training sequence start and end are not clearly defined, i.e. continual problems.

Output gate is a learned sigmoidal unit, which determines if the contents of the memory cell, which go through non-linear transformation (denoted as

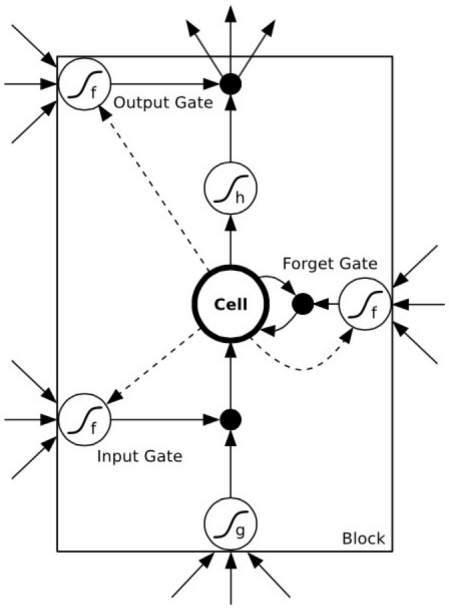


Figure 2.12: LSTM unit [113]

sigmoid h in Figure 2.12) should be output as input y_t to the next layer as well as for the next time step h_{t-1}. The activation of output gate also depends on current input x_t , previous hidden state h_{t-1} and the current content of the memory cell.

2.2.5.3 Applications in sentiment analysis

RNNs have been used for a number of different tasks in SA. One of the most important recent usages was described by Socher et al. [128] where they introduced the SSTb dataset. By using a recursive NN (RecNN), a generalization of a RNN, which operates on tree structures, they significantly improved over current state-of-the-art results in SA. They used several RNN based models for compositional feature representation such as standard RecNN, Matrix-Vector RNN (MV-RNN) and Recursive Neural Tensor Network (RNTN) and obtained 45.7% and 85.4% for fine-grained and binary classification, respectively.

Paulus, Socher and Manning [111] proposed an approach named Global Belief-Recursive Neural Network (GB-RecNN) as state-of-the-art in granular sentiment analysis. Using a RecNN, they perform a standard feedforward pass, as well as feedbackward pass from the tree root, to obtain the final result by softmax combination of the two passes. This model is very similar to a bi-directional RNN used in [59], and using it the authors were able to capture the relevant contextual sentiment.

In [71] Lakkaraju et al. looked at problems of Separate Aspect Sentiment Model (SAS) where aspects and their sentiment are predicted independently, and Joint Multi-Aspect Sentiment Model (JMAS) where aspect-sentiment pairs are modeled jointly. They used the RNN proposed by Socher et al. [128], while achieving state-of-the-art results on several multi-aspect SA benchmarks.

2.2.6 Other deep network architectures

In [127], the authors propose a semi-supervised approach based on recursive autoencoders for predicting sentiment distributions. Their model is able to learn vector space representation for multi-word phrases by exploring the recursive nature of sentences.

Zhou et al. [149] present a RBM based network named Active Deep Network (ADN) for semi-supervised sentiment classification. By using unsupervised pre-training, active learning for dataset optimization and supervised learning for parameter tuning, the authors obtain results that outperform classical approaches on a multitude of dataset. The results presented in this paper were consequently improved on in [42], by using rectifying units instead of other more common non-linearities.

2.2.7 Techniques for improving the training of NNs

Here we describe a number of practical techniques that were shown to improve the speed of NN training as well as improve the prediction accuracy and generalization of trained models.

2.2.7.1 Greedy layer-wise pretraining

Initially introduced by Hinton et al. [50] for training stochastic NNs like RBMs, greedy layer-wise pretraining works by training each layer of the network separately, in an unsupervised fashion, where each layer takes as input the representation learned by the previous layer. Following the unsupervised pretraining, the whole network can be fined-tuned using supervised methods. Thsi approach was extended by Bengio et al. [7] using Autoencoders, where we have two functions, the encoding function f, which takes the input x and maps it into a new space $y = f(x)$, and the decoder function, which maps back from space y to original input space $\hat{x} = g(y)$ (see Figure 2.13). By

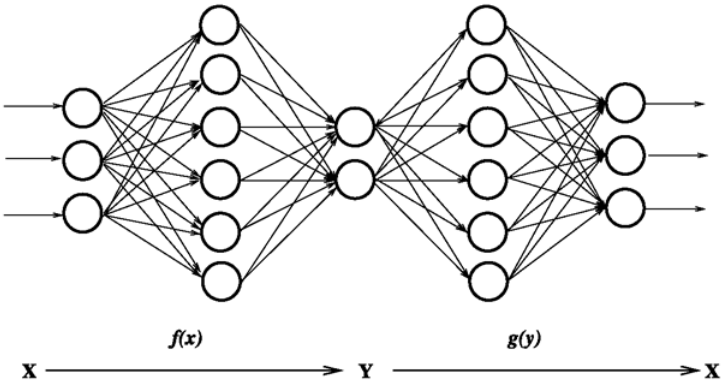


Figure 2.13: Autoencoder Neural Network [122]

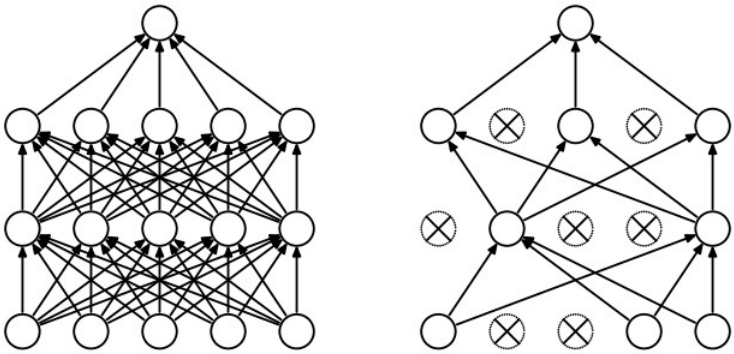


Figure 2.14: Dropout regularization [64]

minimizing the reconstruction error, we are learning to reconstruct the most likely input configurations, and in this way pre-initializing the weight matrix to most likely values.

2.2.7.2 Dropout

In addition to classic L_1 and L_2 regularization, dropout is another way of preventing overfitting during NN training. This approach, introduced by Hinton et al. [52], works by dropping units in hidden layers with some probability p, along with their connections, during the training phase (see Figure 2.14). By forcing the network to learn multiple independent representations of the same data, it prevents the network from co-adapting too much, and is comparable to the way ensemble methods increase the performance of multiple stacked models trained with similar sets of parameters [129].

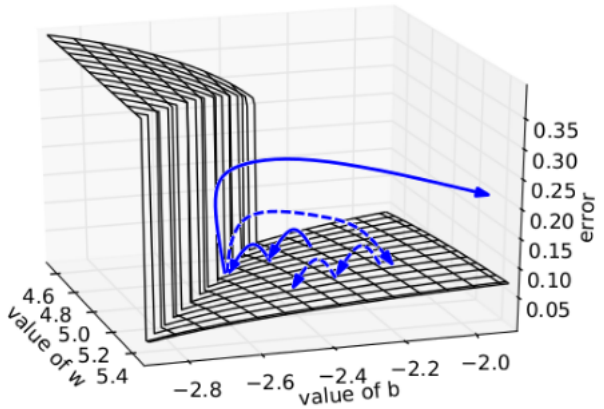


Figure 2.15: Gradient explosion clipping [110]

2.2.7.3 Exploding/Vanishing gradient countermeasures

As mentioned before, the problem of vanishing/exploding gradient is very common in NNs that backpropagate the gradient update for a long number of steps. If the value of the gradient grows too quickly, as it backpropagates through layers, it will cause a memory overflow, which can then be corrected. If, on the other hand, the gradient starts dropping too quickly, the updates to faraway layers can become extremely small to non-existent, thus seriously diminishing the quality of a trained model, while at the same time being very hard to detect [8].

To deal with the problem of exploding gradient, Mikolov [110] suggests using a method named gradient clipping. It is a simple method which sets the gradient back to a small number after it goes over a specific threshold using the following update rule

$$\hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|}\hat{g}.$$

The effects of gradient clipping are shown in Figure 2.15. Here a small RNN with a single hidden unit is trained though a number of gradient descend steps. The solid blue line shows the outcome without gradient clipping, i.e. when the optimization model hits the high error wall, gradient is pushed to a far away location on the surface. With gradient clipping, the error gradient is moved more closely to the original landscape, as shown in the dashed blue line.

On the other hand, for dealing with vanishing gradient, in RNNs we can use the previously mentioned LSTM units, or Rectified Linear Units (ReLUs)

instead of classical sigmoid activation function. Given that the derivative of a ReLU unit is 0 or 1, the error gradient is able to flow through units whose derivative is 1, without abating in the process.

2.2.7.4 Parameter tying

This optimization comes in a few different forms, depending on the architecture of the network, but the end goal is the same:

- reduce the number of parameters to be learned in a given layer,
- reduce the number of samples required for training,
- and reduce the time required for NN training.

For example, in CNNs the weights of a single channel can be tied together, as that would mean being able to detect a specific shape over the whole input space on that channel. Coupled with a big number of channels >100, and max-pooling layer, this enables CNNs to effectively select most active shapes at each localization of input space.

Another example, which works very well in the case of Autoencoders, is to make the decoder weight matrix equal to the transpose of the encoder weight matrix [5].

2.3 NLP feature design and language models

Here we review different ways of encoding textual input data to be used in training of a neural network. We start by looking at simple n-gram encodings, and then move on to neural language models, based on unsupervised NNs.

2.3.1 N-gram language model

An n-gram is a continuous sequence of n items, which could be phonemes, syllables, words, letters etc. In our case we are interested in modeling textual data, thus we will be using words as a unit of n-gram processing. A simplest n-gram sequence is a unigram sequence, where each word represents a single n-gram. Also common are bigram and trigram sequences, of two and three words grouped together. For example, bigrams for the sentence "*the wooden door*" are ("*the*", "*wooden*") and ("*wooden*", "*door*").

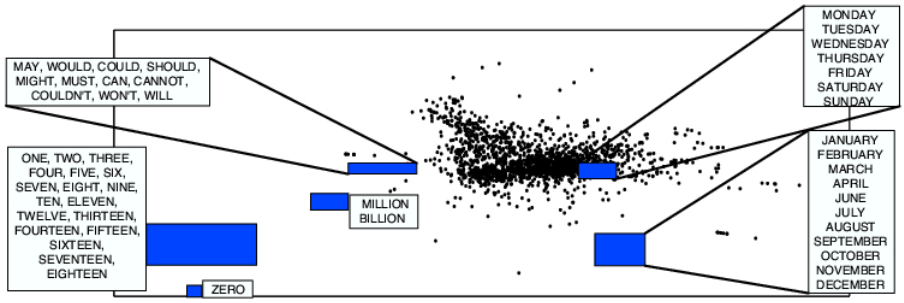


Figure 2.16: Continuous representation of words [11]

One way of using n-grams to represents the training samples in a textual corpus is by using binary vectors indicating presence or absence of specific n-grams. Given the curse of dimensionality, especially when n is high, generally a number of k top n-grams is selected [20]. For example, in a unigram case, with setting k to 1000, each training sample would be represented as a 1000-dimensional binary vector indicating presence or absence of 1000 most frequent words in the given corpus.

Although simple, these models toss away a lot of information, such as word orderings and long range dependencies, as well as detecting only most frequent words due to dimensionality being too great otherwise.

2.3.2 Neural language model

Neural language models have been around for a long time, but until recently it was not possible to train them efficiently on big amounts of data. Even so, they have been shown to outperform n-gram models trained on same amounts of data [6, 90, 124]. The main idea behind these models is to learn representations of words as continuous vectors, in order to reduce the impact of the curse of dimensionality. Thus, each word would then correspond to a point in n-dimensional feature space, and the aim is for semantically and syntactically similar words to be close to each other in this space. This property, in turn, allows NNs trained on these feature vectors to easily compact similar representations into a learned function, and achieve good modeling properties [11, 49].

The advantage of using distributed representations for training a NN model is in being able to easily generalize well on unseen sequences. This is because NNs map nearby inputs to nearby outputs, and thus the unseen feature vectors of similar words are mapped to similar predictions. Moreover, in this way, even

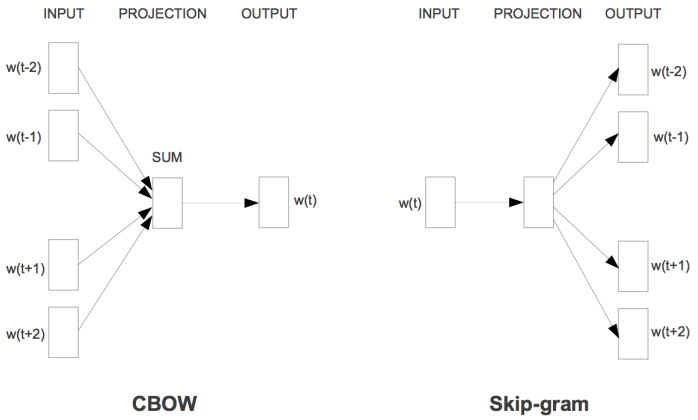


Figure 2.17: CBOW and Skip-gram architectures [89]

models with a small number of parameters are able to fit a very large training set, due to many different combinations of feature values being possible.

A number of different neural language models have been proposed over the years [6, 88, 91, 124], but here we will focus on most recent development by Mikolov [89], recently incorporated in the word2vec tool[1], as well as those discussed by Pennington, Socher and Manning in [112].

2.3.2.1 Continuous Bag of Words model (CBOW)

This model was introduced in [89] as a proposed improvement on the complexity of an earlier similar model named Feedforward Neural Net Language Model (NNLM) [6]. CBOW model consists of an input, projection and output layer. In the projection layer, the word vectors from the input layer are projected using a projection matrix and then averaged in the projection space. The output layer gives a probability over the corpus vocabulary of most likely word given the input sequence. In this model authors use four future and four history words to predict the word in the middle of this sequence. Given that the model does not take into account the order of words in the input sequence it is analogous to the classical bag of words approach. Moreover, CBOW model has demonstrated state-of-the-art results on syntactic part of Semantic-Syntactic Word Relationship test set. outperforming all other neural language model in this category.

[1]https://code.google.com/p/word2vec/

2.3.2.2 Continuous Skip-gram model

This model is a log-linear classifier containing the same layer architecture as the CBOW model, but with an aim of predicting the context given a current word. More precisely, it takes as an input a single word, and aims to predict the neighbouring words in a certain window. The reason it is named skip-gram is due to its ability to predict neighbouring words that are not necessarily preceding or following in sequence, e.g. fifth and second word preceding an input word and two words that immediately follow it. Finally, the results have also shown that skip-gram model outperforms any previous model, including CBOW, on the semantic part of Semantic-Syntactic Word Relationship test set [89].

2.3.2.3 Global Vectors for Word Representation (GloVe)

GloVe [112] is a log-bilinear model using the property of ratios of word-word co-occurence probabilities to encode information. Namely, when constructing a word-word co-occurence matrix, the ratios of those probabilities are able to cancel out the noise for both elements in the ratio and encode meaning. Given that, the probability of the ratio, if much greater or less than one, captures the relevant relationships in the co-occurence matrix.

The way this is implemented is by optimizing a weighted least-squares objective. The goal is to learn word vectors whose dot product equals the logarithm of the words' probability of co-occurrence. Thus, this can be viewed as optimization of vector differences in the word vector space.

GloVe has also been shown to outperform both skip-gram and CBOW architecture in word analogy task [89], word similarity tasks [84] and on the CoNLL-2003 benchmark dataset [134].

2.4 Chapter summary

In this chapter we reviewed classical approaches to sentiment mining as well as more recent advancements based on neural networks and deep learning.

In classical approaches section, we discussed theoretical background of some of the core algorithms used in the past 20 years, as well as their variations, and application to a number of tasks in sentiment prediction, including document-level, aspect-based, fine-grained and multilingual SA.

We then gave a brief overview and history of neural networks and relevant research milestone in their rich 50 year history. Furthermore, we discussed what constitutes deep learning and some of the core concepts associated with their advancements, before moving on to most popular deep learning models such as CNNs and RNNs. Moreover, we also looked at some practical implications and advice on training and designing such systems.

Finally, the chapter finishes with a review of different ways of encoding data used in training of SA models, such as n-grams, as well as more recent neural language models.

Chapter 3

Model design and experiments

This chapter starts by looking at the dataset used in the experiments before moving on to feature design process and data embedding. Following that, we take a look at a number of baseline measures and models. We then present two novel deep architectures, a CNN and a RecNN, trained on GPGPU AWS instances, with state-of-the-art results in a number of SA tasks, and improved generalization properties when compared to current state-of-the-art NN models.

3.1 Dataset

The main dataset used to train models in this dissertation is the Stanford Sentiment Treebank (SSTb) corpus. This corpus, created from the original Pang and Lee *Rotten Tomatoes movie review corpus* [107], was introduced by Socher et al. [128] in 2013. It consists of 11,855 sentences with full parse trees, created using Stanford parser [68], with about half of the sentences being negative and half of them being positive. From the given parse trees, 215,154 unique phrases were extracted, and each one of the phrases was labelled separately by three independent judges using a slider, after which the scores were averaged, to obtain the final phrase sentiment in the range of 1-25.

As Socher et al. note [128], "most annotators moved the slider to one of the five positions: negative, somewhat negative, neutral, positive or somewhat positive", thus "even a 5-class classification into these categories captures the main variability of the labels". Given this, the authors defined *fine-grained* sentiment labelling as a five class classification problem, with the aforementioned labels, in the range 1-5. In our experiments with fine-grained sentiment

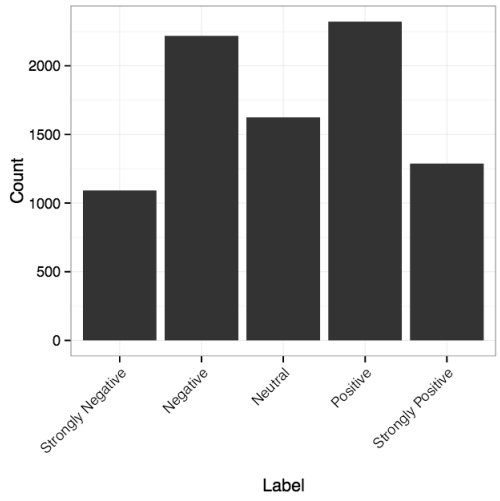


Figure 3.1: Distribution of labels in the fine-grained SSTb dataset

classification we also adhere to this convention, by mapping the original 1-25 annotation to a new 1-5 annotation. Furthermore, to be able to compare our models against other state-of-the-art models, we also create a dataset with *binary* labels, consisting only of negative and positive phrases, with two labels, 1 and 2, respectively.

For training a CNN, we used 156,060 parsed phrases for the training set, as we wanted our network to learn sentiment on different levels of granularity, but for validation and testing, we decided to use only full sentences. The reason behind this decision, which is also in line with experiments performed in [62], is that the final CNN model will be used to determine sentiment of longer textual content consisting of tens of sentences, thus the sentence level granularity is considered sufficient for this use case.

In the case of RecNN model, we used the original parse trees presented in [128] given in a 8545:1100:2210 split for train, development and test sets.

3.2 Features design and data embedding

The data from the SSTb corpus was encoded in a number of different ways in order to find the most optimal encoding that captures the highest amount of information about sentiment. We experimented with n-gram based features, as well as skip-gram and GloVe continuous word embeddings. The tests show that neural language models significantly outperform traditional n-gram based approaches.

3.2.1 N-gram features

In order to enable creation of n-gram feature vectors, the dataset had to be preprocessed in order to detect all n-grams and their frequencies. Thus, each of the sentences in the dataset had undergone through the process of:

- tokenization,
- punctuation removal,
- stopword removal,
- and word stemming,

before n-grams of size one, two and three were created (depending on the forwarded parameter), and their overall frequency in the corpus calculated. Following that, based on this frequency, a number of top k n-grams were selected to be used for presence detection in a binary encoded feature vector. Each of the selected n-grams was tied with a unique id in order to enable usage of sparse encoding matrices.

Following this step, each newly seen sentence would be preprocessed in the aforementioned way, and a sparsely encoded binary vector of size k would be returned for that instance.

3.2.2 Continuous word vector embeddings

Creation of continuous word embedding feature matrix for each sentence consisted of a number of steps. Firstly, the maximal length of sentence (in tokens) l had to determined, as that would define the width of each sentence feature matrix. Size was determined based on lengths of sentences in a corpus of 1000 recent online news documents obtained from Signal, as well as lengths of sentences encountered in the SSTb dataset. The optimal size was determined to be 43, which encompasses all of the positive and negative sentences in the SSTb dataset, as well as 95.61% of sentences found in data obtained from Signal, as shown in Figure 3.2.

Following this, each sentence was tokenized along with punctuation being removed. Each word token was then queried with one of the neural language models, which were trained in an unsupervised way on a very large text corpora, in order to obtain a word vector of dimensionality n. The neural models

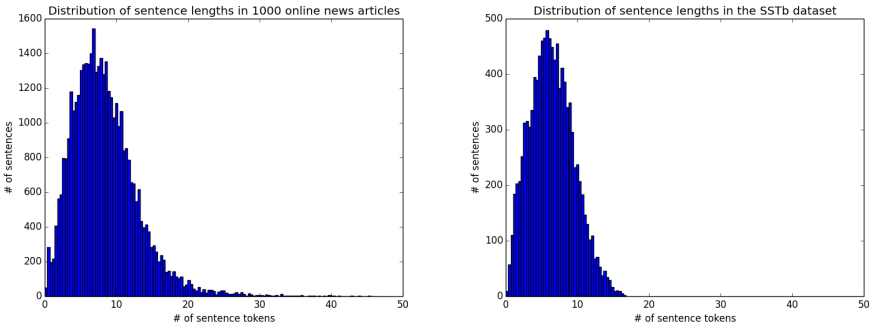


Figure 3.2: Distributions of sentence lengths in Signal corpus and the SSTb dataset

and their training is described in more detail in chapters 3.2.2.1 and 3.2.2.2. Thus, the outcome for each sentence was a feature matrix of dimensions $n \times l$.

This representation was suitable for CNN models, as they traditionally operate on images, and require data to be represented as a matrix. For benchmark models, the word vectors stacked in the sentence feature matrix were combined in different ways such as averaging and summing (see chapter 3.3), in order to obtain a one dimensional representation vector. Continuous word embedding were also used in training RecNN models, by encoding each of the leaf nodes in a binary tree as a continuous word vector.

3.2.2.1 Skip-gram embeddings

The skip-gram model used follows the architecture described in chapter 2.3.2.2. It was trained on the Google News dataset containing around 100 billion words, and returns 300-dimensional vectors for three million words and phrases. The phrases were created using a data-driven approach described by Mikolov in [92].

3.2.2.2 GloVe embeddings

The GloVe model is based on the architecture covered in chapter 3.2.2.2. A number of models with varying dimensions and trained on different corpuses were tested. The models are:

1. 100-dimensional model trained on Wikipedia 2014 dump and Gigaword 5 corpus (10 million news documents),

2. 200-dimensional model trained on Wikipedia 2014 dump and Gigaword 5 corpus,

3. 300-dimensional model trained on Common Crawl corpus of 840 billion tokens.

The experiments conducted with these models have shown that it is almost always better to use the longest possible vectors, as they capture more information, although in some instances, smaller vectors could be beneficial, if the entire dataset cannot fit into the working GPU memory. This comes as a consequence of having to switch parts of the dataset in and out of the working GPU memory, as the whole set is too big to fit in at the same time. Besides slowing down the model training, if too many of these swaps are made, and too often, it can destabilize the parameter optimization procedure, and cause the objective to diverge. This is discussed in more detail in chapter 3.4.3.2.

3.3 Baseline model design

We trained a number of models described in the classical approaches part of the literature review (chapter 2.1), as a baseline comparison for the NN models. Models were trained using n-gram based feature vectors, as well as continuous word vectors wherever possible. A number of hyperparameter settings were tested for each model, and the result of the best tuned model is reported in Table 4.1 for binary classification, and Table 4.2 for fine-grained (5-class) classification.

3.3.1 Dealing with continuous feature matrices

As mentioned previously, for continuous word vectors, each sentence was encoded as an $n \times l$ feature matrix, where n represented the dimensionality of the word vectors, and l is the maximum length of a sentence. In order to obtain a one dimensional vector, three different ways of combining vectors were evaluated, namely *summing*, *averaging* and *appending*. In summing, all word vectors in a sentence are summed together, giving another word vector as an output. This simple arithmetic operation was shown to be able to partially capture the meaning expressed with multiple phrases in a sentence [93]. In averaging, the procedure is similar, but the word vectors are averaged together. Finally, appending constitutes of reshaping the feature matrix into a

one dimensional vector, such that word vectors follow each other in the same order as seen in that sentence.

Experimental results have shown that summing consistently outperformed averaging, for all the models involved in baseline tests. On the other hand, appending showed slight improvements in metrics, most likely due to keeping the word ordering information, but those improvements came at a big computational cost, as feature vectors were much larger. Thus, it was concluded that the summing method allows for the best combination of performance and computational complexity.

3.4 Deep NN architectures for SA

Here we present two NN architectures for binary and fine-grained document-level SA problem. We start by looking at specific technologies used in developing them, as well as requirements for training these networks, before moving on to architecture description for both types of presented models. We examine the performance of these networks and results on different datasets in Chapter 4.

3.4.1 Chainer NN framework

Chainer[1] is a novel flexible NN framework, supporting CUDA and multi-GPU computation, characterized by *define-by-run* scheme compared a more traditional *define-and-run* paradigm, which was used to implement both CNN and RecNN architectures.

In define-and-run scheme (Caffe, Torch7, Theano), the network is first parametrized and fixed, and then trained on mini batches. Given that the network is static, i.e. it is defined prior to any forward/backward computation takes place, all the logic of the network must be embedded in the network architecture as data. Moreover, networks definitions in these frameworks are hard to debug, due to error happening in forward computation, which is far apart from where the bug occurred.

On the other hand, in Chainer, the network architecture is defined during program run using forward computation. By storing the history of all the previous computation instead of logic, Chainer gives us flexibility to easily combine NN elements, as well as much simpler ways of dealing with loops

[1] http://chainer.org/

and conditionals, when compared to more traditional deep NN frameworks. Furthermore, because of the define-by-run paradigm, debugging is also very easily performed in Chainer. Debugging flags can be set anywhere in the code, including the network definition and backward runs.

3.4.2 Amazon Web Services (AWS) NN training

Due to NNs having millions of parameters which needs to be learned, as well as tens of hyperparameters that also need to be optimized by some kind of search, training these models on standard CPU units would simply be too slow. Thus, this process had to be moved onto highly parallelizable General Purpose Graphical Processing Units (GPGPUs) which are able to provide speed ups of more than 100 times, when compared to traditional CPU bound computation.

In order to to this, a g2.2xlarge instance was provisioned on Amazon Web Services (AWS). The instance contains a single Grid K520 NVIDIA GPU with 1,536 CUDA cores and 4 GB of video memory. Once provisioned, the instance had to be set up with a distribution of Ubuntu, as well as support for CUDA computation and cudNN library for GPGPU speed-ups. AWS credits used for computational time were obtained through Amazon education grant, as well as an award for competing in the EpicCode CodeSprint competition during June/July on HackerRank[2].

On a side note, as a byproduct of this thesis, we wrote a white paper on setting up a GPGPU server on AWS from scratch, along with installing software required to run CUDA computation and setting up all the environmental variables. This report should appear shortly on Signal's newly setup research website, and any interested readers are also welcome to request a copy.

3.4.3 Adaptive Switching CNN (AS-CNN)

We now present a novel CNN architecture for multi-class document level SA problem, followed by a discussion of special techniques used and problems faced.

Disclaimer: part of this network was developed in collaboration with Matteo Hessel, who had a requirement for mid-layer representation that this network offers, for his MSc project. More specifically, we worked together on setting up the basic layer architecture in Chainer, provisioning the AWS in-

[2]https://www.hackerrank.com/

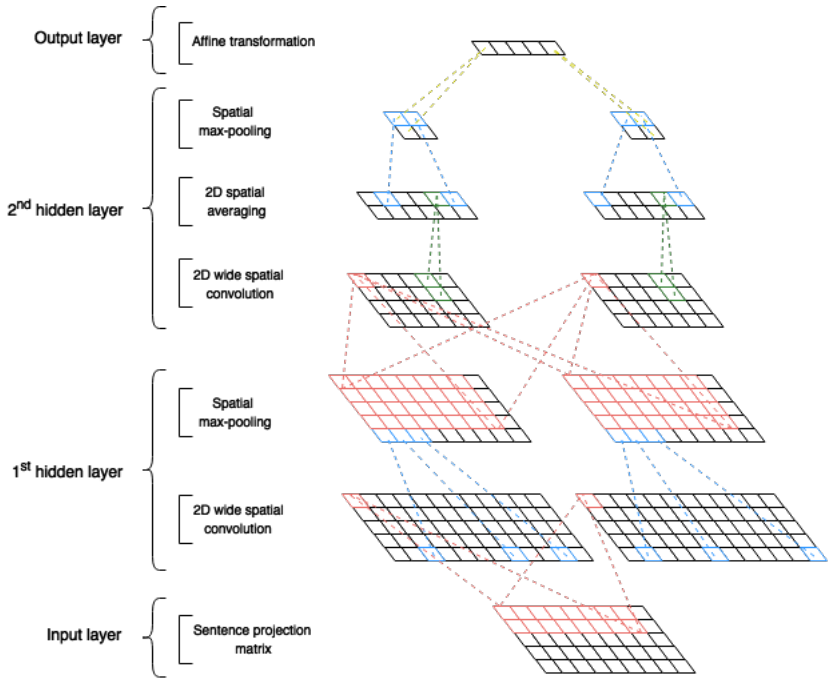


Figure 3.3: AS-CNN architecture

stance, as well as designing the hyperparameter optimization algorithm for the final network.

The implemented model consisting of an input layer, two hidden layers and an output layer. It was trained using dynamic switching of datasets during optimization phase, and tuned with an adaptive look-ahead hyperparameter search, thus appropriately named Adaptive Switching CNN. A detailed architecture of the network is shown in Figure 3.3.

We make use of two dimensional wide convolution, meaning the filter is a two dimensional shape, moving, as specified with stride, over the padded input matrix. Furthermore, we also include spatial max-pooling, which differs from normal max-pooling. Instead of taking a single maximum over a given dimension, spatial max-pooling, like convolution, uses filter and stride, and takes a maximum within the given filter. Hence, it is similar to k-max-pooling introduced in [62]. Finally, spatial averaging layer, in addition to spatial max-pooling, is used to reduce the dimensionality of the problem in the penultimate layer. This is achieved by taking the average value of two adjacent matrix dimensions and moving with a vertical stride of two.

The architecture is as follows. The input layer is represented by the sentence embedding matrix. This is followed by the first hidden layer, consisting of two dimensional wide convolution followed by a non-linearity, and spatial

max-pooling. This structure, with different hyperparameters, is repeated in the second hidden layer, with the addition of spatial averaging, inbetween convolution and pooling. While training, we also apply dropout after the non-linearity in the second layer. The output layer is an affine transformation, which feeds into softmax non-linearity in order to obtain a distribution over sentiment classes for a given sentence. The non-linearity used after both convolution layers is a Rectified Linear Unit (ReLU).

There are two different sets of hyperparameters we used, one for binary classification model, and one for fine-grained classification model. Both sets are optimal, and they were obtained using adaptive look-ahead hyperparameter optimization approach, described in more detail in section 3.4.3.3.

In binary model, the convolution filter sizes are (2, 8) and (4, 8), with a unit stride of (1, 1) in both cases. For spatial max-pooling, we use filters of size (1, 9) and (1, 7), with stride (1, 5) and (1, 4), respectively. For feature maps, in the first layer we used 27 maps, and in the second layer we used 29 feature maps.

For fine-grained model, the convolution filter sizes are (5, 7) and (6, 7), with a unit stride of (1, 1) in both cases. For spatial max-pooling, we use filters of size (1, 8) and (1, 7), with stride (1, 4) and (1, 3), respectively. For feature maps, we use 15 maps in the first layer, and 21 maps in the second layer.

We can also calculate the size of the representation matrix following each transformation. Initial sentence embedding matrix is of size $n \times l$. The size of the matrix following a convolution layer is given by

$$l = (l + 2p_l - f_l)/s_l,$$

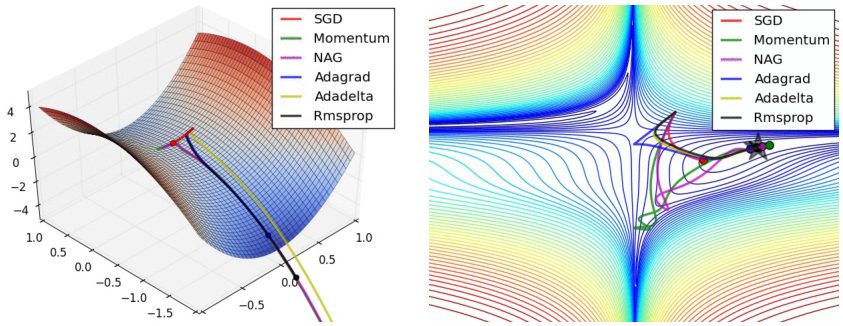
$$w = (w + 2p_w - f_w)/s_w,$$

where p is the padding, f is the filter and s represents the stride. Next, one dimensional spatial pooling layer matrix transformation is described by

$$w = \left\lfloor 1 + \frac{w - f_w}{s_w} \right\rfloor \text{ for cover-all mode, or}$$

$$w = \frac{w}{s_w} \text{ when cover-all mode is off,}$$

where cover-all mode means that all spatial locations are pooled into some output values. The dimension l is not changed in this transformation. Furthermore, the spatial pooling halves the matrix along the feature vector dimension, i.e.



(a) SGD on saddle point

(b) Adagrad almost overshooting on Beale's function

Figure 3.4: Fallacies of different optimization algorithms

$$l = \frac{l}{2}.$$

Finally, we end up with the affine transformation layer matrix that is of shape $c \times (\#ch * n * l)$, where c is the number of classes, and $\#ch$ is the number of feature maps (channels).

3.4.3.1 Optimization procedure

For AS-CNN optimization we tried a number of different algorithms, with some of them being shown to posses quicker convergence properties, while others being more successful in finding better minima.

The loss function which was optimized is cross-entropy over multiple non-linear hidden layers, hence presenting us with a highly non-linear objective. One of the things we tried during training was to dynamically switch between two optimizers every n epochs. This approach has been applied successfully before, in a number of optimization algorithms, such as Brent's minimization algorithm [18]. The idea behind this, is that a combination of two different optimizers would be able to overcome the fallacies of individual algorithms, such as instability of Adagrad [30] in some cases (e.g. Beale's function) or getting stuck in a saddle point for Stochastic Gradient Descent (SGD), as shown in Figure 3.4.

After a number of experiments, we found that, in this case, this approach was not applicable, and that any improvements would be due to stochasticity of the process itself. Nevertheless, this presents itself as an interesting idea that could warrant further research, as such an "ensemble" of optimizers could potentially provide significant improvements in case of some highly non-linear

objectives.

The final optimization procedure was performed using Adam optimization algorithm [67], a gradient-based optimizer, based on adaptive estimates of lower-order moments. It has been shown empirically that Adam is very well suited for problems with large amount of data and parameters. This especially holds true for deep CNN model, where it was shown that Adam outperforms Adagrad in convergence rate, as well as SGD and Nestorov's Accelerated Gradient in finding a better minima. These results were also in line with our experiments.

3.4.3.2 GPGPU memory switching

In cases where the network is trained on a dataset that is too large to fit into the working GPU memory, dynamic dataset switching needs to be applied during optimization procedure. We developed a procedure to apply this swapping automatically depending on currently available system resources. At the beginning of the training, the networks determines the size of the dataset and compares it with the available GPU memory. If necessary, the dataset is then split into multiple parts, which are then swapped in and out of working GPU memory in round-robin fashion.

The user has to define two parameters for this to work. First, the number of epochs each subset will be used is something that needs to be determined empirically, but our experiments show that around 5 to 10 epochs is a good number, taking into the account the fact that the training procedure for AS-CNN tends to start overfitting after epoch 25. The second parameter that needs to be defined is if there are to be any overlaps in data between subsets that are created. We found that having anywhere from 20% - 30% of overlapping data helps training, causing it to converge faster, and sometimes even avoiding divergence.

3.4.3.3 Adaptive look-ahead hyperparameter optimization

Once the architecture of the network is determined, the process of hyperparameter search needs to be performed. Standard NNs have tens of hyperparameters, such as learning rate, number of training epochs, length and width of convolutions etc., each of them taking a number of different values. In the case of AS-CNN, the number of variable hyperparameters was 16, and those included, among others, lengths and widths of convolution filters, pooling,

padding and strife, in multiple layers. A number of hyperparameters, such as number of epochs and batch size were optimized manually, and the optimal number of epochs was determined as a point of divergence between train and validation loss (see Figure 3.6), as suggested in [5].

All the other hyperparameters were optimized using an approach combining random search and meta-learning. The approach is as follows:

The hyperparameter search begins with a number of seed runs, which are used to determine the accuracy of the system on the development set, given a set of hyperparameters specified by the user. Following that, the user sets the number of optimization runs (usually from 50-100), and while running the system keeps track of the best hyperparameter set so far. That best set is then used as a vector of means of a constrained multivariate Gaussian distribution, along with a predetermined variance vector. The system also keeps a regressor model as a meta-model (in our case a Random Forest regressor), which is trained on a dataset of all previously used hyperparameter sets until that point, using their reported accuracies on the development set as labels. In the look-ahead step, a number of sampling runs k, from the aforementioned distribution, is performed, and for each of thus obtained hyperparameter sets, an accuracy prediction is obtained from the regressor model. The set with the best predicted accuracy out of these k sets is returned as a new candidate hyperparameter set, for the network to be trained on for a small number of epochs (usually 5).

As shown in Figure 3.5, this approach has given us notable improvements in reported accuracies, on both binary and fine-grained classification, of up to 8% in some cases. Moreover, we found out that some non-conventional combinations of parameters are able to outperform those more commonly used in CNNs for NLP [28,62]. For example, general practice for convolution in NLP CNNs is to use a filter that convolves only across a single vector dimension of the sentence. In our tests, we found that, on higher hidden layers, convolving over multiple vector dimensions, with a small stride, yields better results (as shown in Figure 3.3). This result is most likely due to higher level features depending on multiple factors that constitute the knowledge encoded in words vectors.

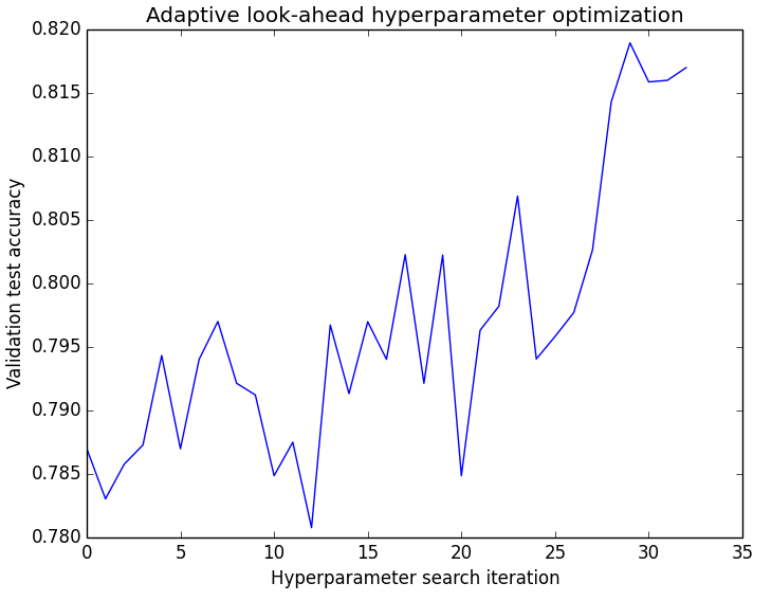


Figure 3.5: Hyperparameter search results

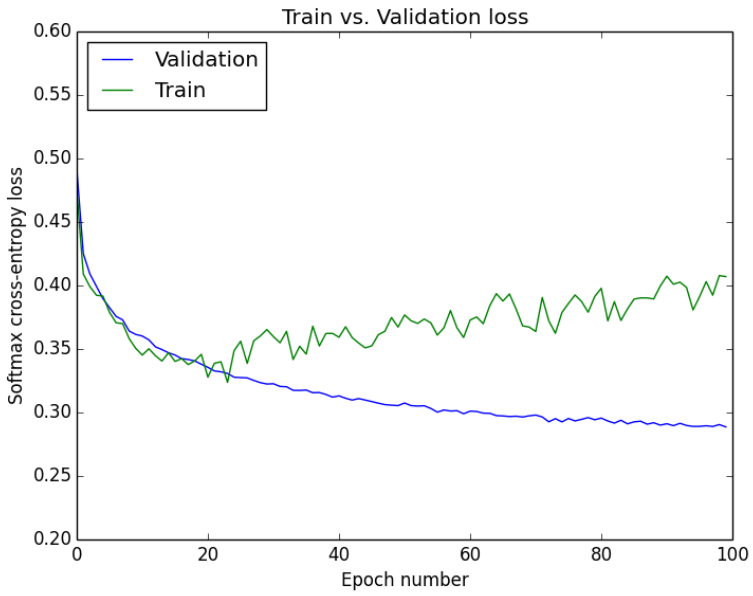


Figure 3.6: Train vs. validation loss for AS-CNN

3.4.4 Augmented Recursive Neural Network (ARecNN)

In this chapter we present another novel NN architecture for SA problem, this time from the RNN family of networks.

Classical recursive NNs for SA take a different approach in dealing with combining word vector representations. While in CNNs sentences and their continuous space representations are viewed as matrices of images, where we perform operations such as convolution and pooling, in RecNNs we operate by creating recursive semantic compositionality over sentence parse trees. Firstly, all sentences in an input text need to be parsed into binary trees, as capturing word order in this way can aid us in identifying complex negation patterns that are usually built from multiple phrases. Secondly, each leaf of the binary tree is then represented using continuous word representation model. Finally, as shown in Figure 3.7, the nodes are combined in a bottom-up approach, using a compositionality function f, where outputs of this composition on lower layers feed into higher level nodes as features. This can be expressed as,

$$p = f\,(W^{(1)} \begin{bmatrix} h_{left} \\ h_{right} \end{bmatrix} + b^{(1)}),$$

$$\hat{y} = softmax(W^{(s)}\, p + b^{(s)}),$$

where p is the representation combining left and right children nodes, f is the non-linear transformation ($tanh$ in our case), and $\hat{y}$ is the output prediction created by the softmax non-linearity. During training our aim is to learn weight matrices $W^{(1)}$, $W^{(s)}$ and their biases $b^{(1)}$, $b^{(s)}$.

3.4.4.1 Augmentation layer

We present a novel modification of the classical RecNN approach, by introducing a feature augmentation step, during which we create additional compositionality based features, as an input for the parent node. As shown in Figure 3.8, the child nodes are combined in an intermediate hidden layer, in order to form a more complex representation. Mathematically, this can be written as

$$h = f\,(W^{(1)} \begin{bmatrix} h_{left} \\ h_{right} \end{bmatrix} + b^{(1)}),$$

$$p = f\,(W^{(2)} \begin{bmatrix} h_{left} \\ h \\ h_{right} \end{bmatrix} + b^{(2)}),$$

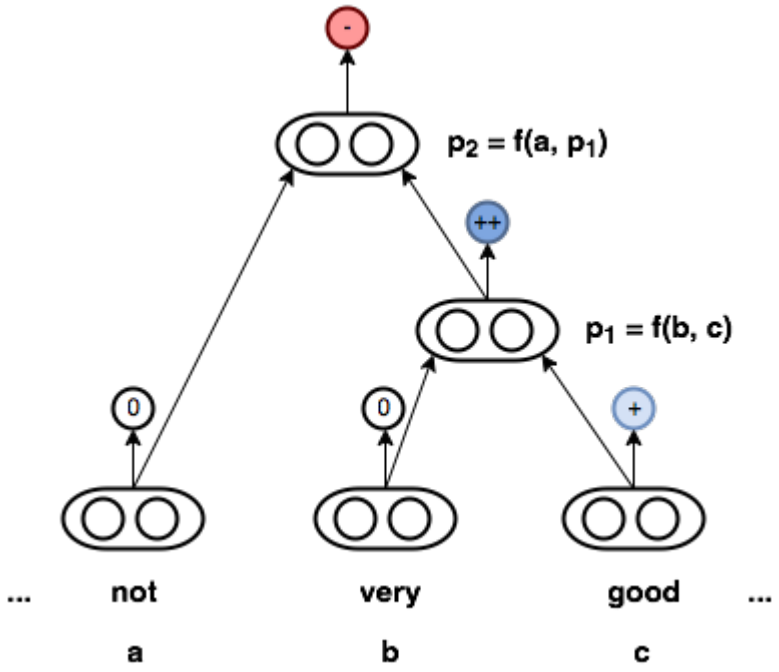


Figure 3.7: Classic RecNN bottom-up compositionality

$$\hat{y} = softmax(W^{(s)}\, p + b^{(s)}).$$

Doing this allows us to create a more complex decision boundary, and increase the representational capability of RecNN model.

3.4.4.2 Input layer representation

Moreover, another novel feature of ARecNN model, which is also applied in the AS-CNN architecture, is that instead of using an embedding matrix in the network input layer (as commonly seen in literature [62,126,127]), to learn our own word vector representations, we have opted for usage of a separately trained neural language model. Consequently, this allows us to query the neural model against a much larger corpus of words, and thus obtain meaningful continuous word representations. On the other hand, most current state-of-the-art models are unable to deal with any new words, and the ones that do, they perform mapping of all unseen words into a single representation for an unknown word.

While learning word embeddings for a given dataset can greatly increase reported metrics, this goes against one of the main strengths of NNs, which is being able to accommodate and map a large number of input combinations with a small number of parameters (as discussed in chapter 2.3.2), and is

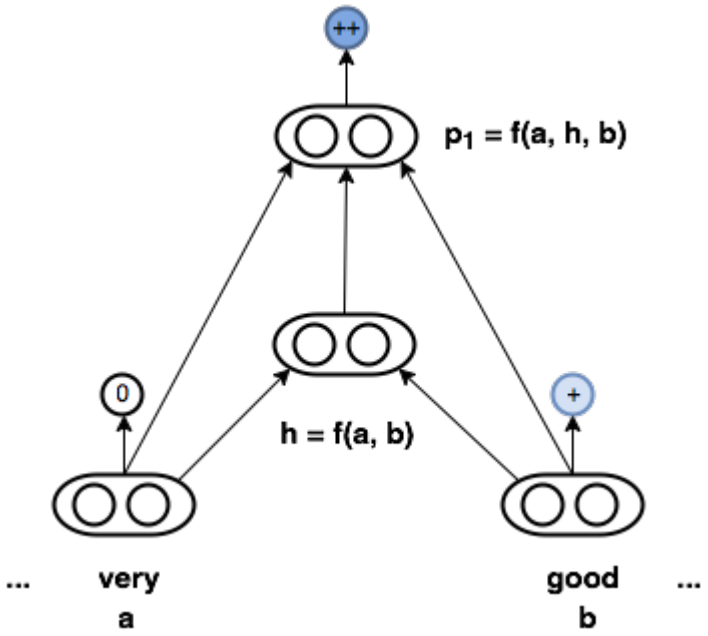


Figure 3.8: Augmented RecNN compositionality approach

basically a form of overfitting to the dataset. On the other hand, with this new approach, we are able to increase the generalization properties of this network, to a number of different domains, as in many cases the vocabulary differs from than the one used in the original training set.

3.4.4.3 Optimization procedure and hyperparameter selection

The network is trained through 400 epochs, taking approximately four days to complete on a single GPU unit. Given the high time cost of training, it is recommended to run the optimization in parallel, on multiple GPU units, which Chainer supports very well. Moreover, due to this cost, we were unable to run adaptive look-ahead optimization for hyperparameters, thus we were limited to trying a small number of values by hand.

We trained the model in the same way as we did for AS-CNN model, by minimizing the cross-entropy between predicted distribution and actual one, for each node. We used Adagrad optimization, with batch size 25, and learning rate 10^{-1}, as it has shown quicker convergence properties in this case, when compared to Adam and Momentum SGD. We also applied weight decay of 10^{-4} with great success, in order to minimize the possibility of overfitting.

For continuous word embeddings at leaf nodes, we decided to take the output of the previously described 300 dimensional GloVe neural language

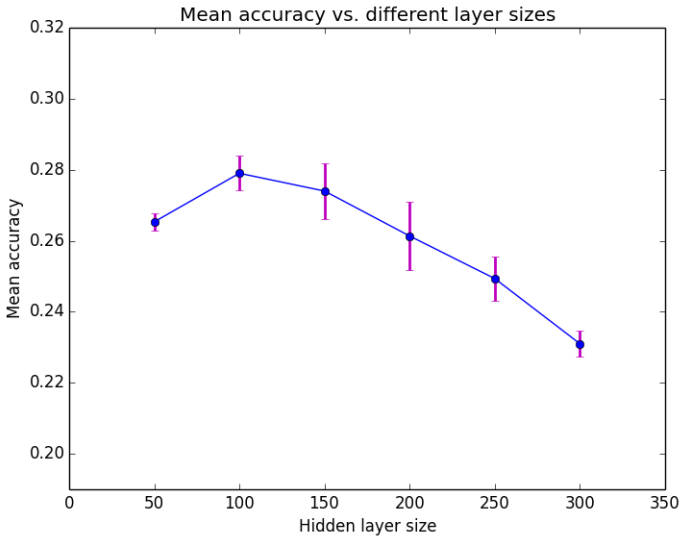


Figure 3.9: Mean accuracy with standard deviation on development set, after 3 runs and 30 epochs for each layer size

model. Furthermore, we experimented with different number of units in the hidden augmentation layer. We found the optimal size of the hidden layer to be $h \in \mathbb{R}^{1\times 100}$, from which it follows that the size of the input matrix for p is $\left[h_{left};\, h;\, h_{right}\right] \in \mathbb{R}^{1\times 700}$.

3.5 Chapter summary

In this chapter we first looked at the main dataset used to train CNN and RecNN models, and variations in its encoding and representation. We then gave an overview of feature design process, including n-gram based features as well as continuous word embeddings. Following that, we presented the modeling process of baseline models, to be used as a comparison to proposed NN architectures. We conclude the chapter by presenting two novel state-of-the-art deep learning architecture with improved representation and generalization properties, along with the challenges we encountered during modeling, training and optimization phases.

Chapter 4

Results and evaluation

In this chapter we evaluate the performance of AS-CNN and ARecNN architectures in two different settings, namely, the original SSTb dataset, as well as a dataset of tweets collected during London tube strike in August 2015. We also compared their performance to benchmark models, and other state-of-the-art SA NNs.

4.1 SSTb dataset

We first report the metrics for each of the baseline models, along with their confusion matrices, in order to have comparable results for our NN models. We then examine examine the AS-CNN and ARecNN model performance.

4.1.1 Baseline model results

The results of tuned models are shown in Table 4.1 and Table 4.2, for binary and fine-grained models, respectively.

For n-gram features, in both binary and fine-grained case, scores for logistic regression and Naïve Bayes classifiers were calculated using 10,000 most frequent unigrams, whereas for SVM and random forest classifiers, 2,000 most frequent unigrams were used. The reason behind choosing these numbers was that after 10,000 features we noticed the beginning of model overfitting. On the other hand, 2,000 features for the SVM model were selected due to high computational cost of training an SVM classifiers, given that in the fine-grained case one-vs-all approach is applied. A number of tests were performed with bigram and trigram features, but those did not give any considerable improvements on the reported results.

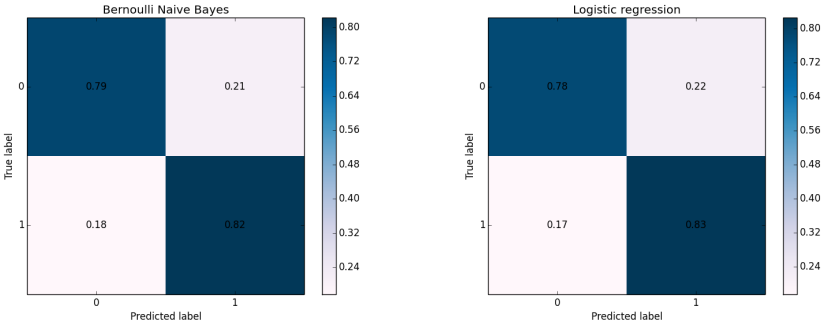


Figure 4.1: Binary benchmark model normalized confusion matrices

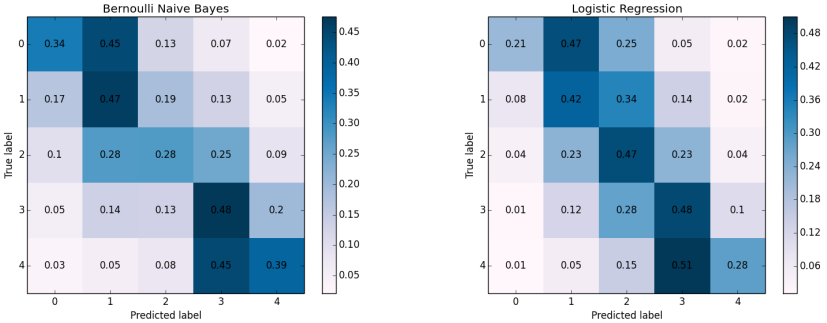


Figure 4.2: Fine-grained benchmark model normalized confusion matrices

Continuous features could not be applied to Naïve Bayes models due to discrete feature requirement, but for other three types of models, 300 dimensional summed vectors were used. We experimented with both Skip-gram and GloVe embeddings, and found that GloVe vectors consistently provided better results, ranging from 0.5% to 1% accuracy improvements. It should also be noted that continuous word vector models outperformed n-gram based models in every setting that was tested, and in some cases significantly so, thus corroborating findings discussed in 2.3.2.

In Figure 4.1 and Figure 4.2, we can see confusion matrices for the two best performing benchmark models. It is interesting to observe different distributions of errors in fine-grained case for both models. In the case of Bernoulli NB, the classifier performs really well in creating the negative/positive separation boundary, however it fails in modeling the sentiment gradation, i. e. different strengths of sentiment in both positive and negative cases. Hence, we hypothesize that a more complex decision boundary, such as those produced by NNs, could improve the performance of the classifier. We thus perform the same set experiments in the NN part of this chapter. Moreover, the results seen in the logistic regression confusion matrix point to a similar problem,

	Feature type							
	N-grams				**Continuous vectors**			
Model	*Acc.*	*Prec.*	*Rec.*	*F1*	*Acc.*	*Prec.*	*Rec.*	*F1*
Multinomial Naïve Bayes	**79.51**	80.67	79.51	80.09	/	/	/	/
Gaussian Naïve Bayes	**71.72**	72.07	71.72	71.89	/	/	/	/
Bernoulli Naïve Bayes	**80.18**	80.57	80.18	80.37	/	/	/	/
Logistic Regression	80.01	80.07	80.01	80.04	**80.58**	80.57	80.58	80.57
Gaussian SVM	64.88	65.32	64.88	65.09	**77.94**	77.93	77.94	77.93
Random Forest	70.70	70.71	70.70	70.70	**75.68**	79.75	75.68	77.66

Table 4.1: Baseline model scores for binary classification

with a more pronounced misclassification of the strongest sentiment in both positive and negative case. Again, this reinforces our aforementioned belief that a more complex classification function is imperative.

4.1.2 AS-CNN results

As shown in Table 4.3, the AS-CNN model achieves state-of-the-art accuracy when compared both to benchmark and NN models[1], in binary and fine-grained case. It is only beaten by a few models trained with a word embedding layer, while at the same time outperforming a good number of models that also posses that layer, such as MAX-TDNN, RecNN and MV-RNN.

Again, as discussed in chapter 3.4.4.2, it is very important to underline, that at a small cost in reported accuracy, the AS-CNN architecture allows for much greater generalization and cross-domain application of a single trained model. This comes as a consequence of using a separately trained neural language model, against which input text is queried, thus allowing for meaningful word vector representation for a much larger corpus of words (over one million). This is explored further with our experiments on the twitter dataset (Chapter 4.2).

Next, we also take a look at AS-CNN confusion matrix for the fine-grained case. As seen in Figure 4.3, there is a clearly pronounced diagonal with reduced

[1]Model accuracies obtained from [62, 128].

	Feature type							
	N-grams				**Continuous vectors**			
Model	*Acc.*	*Prec.*	*Rec.*	*F1*	*Acc.*	*Prec.*	*Rec.*	*F1*
Multinomial Naïve Bayes	**38.65**	45.73	38.65	41.89	/	/	/	/
Gaussian Naïve Bayes	**28.68**	32.15	28.68	30.31	/	/	/	/
Bernoulli Naïve Bayes	**40.70**	41.17	40.70	40.93	/	/	/	/
Logistic Regression	40.40	43.08	40.40	41.70	**41.07**	44.05	41.07	42.50
Gaussian SVM	27.58	29.12	27.58	28.33	**37.50**	33.48	37.50	32.26
Random Forest	30.85	34.70	30.85	32.66	**36.11**	45.34	36.11	40.20

Table 4.2: Baseline model scores for fine-grained classification

	Accuracy (%)	
Model	Binary	Fine-grained
MAX-TDNN	77.1	37.4
NBOW	80.5	42.4
RecNN	82.4	43.2
MV-RNN	82.9	44.4
AS-CNN	**84.5**	**43.6**
RNTN	85.4	45.7
DCNN	86.8	48.5

Table 4.3: AS-CNN binary and fine-grained accuracy

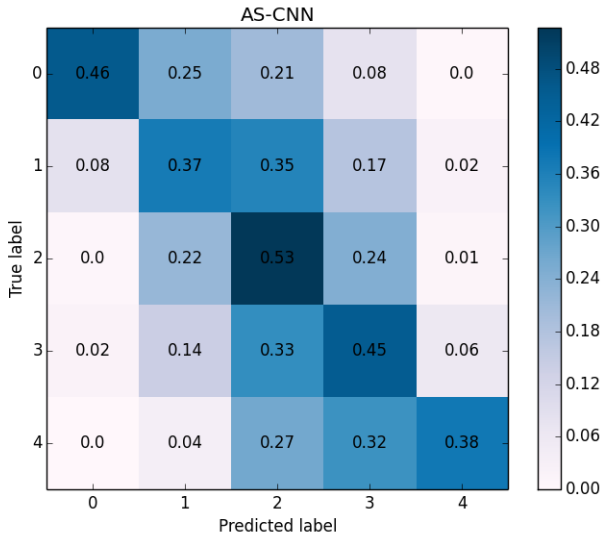


Figure 4.3: Fine-grained AS-CNN model normalized confusion matrix

misclassification rate between negative sentiment type gradations, but slightly increased misclassification rate for the most positive type of sentiment, when compared to Bernoulli NB model. However, the misclassification pattern in the case of AS-CNN is more naturally distributed, when compared to benchmark models, i.e. errors appear in all the neighbouring classes of a target class, rather than a single one, as is the case in both benchmark models. Hence, these findings corroborate the previously stated hypothesis about model complexity.

Ensemble results

Due to low time cost of training CNN models, we have experimented with a number of ensembles of AS-CNN models. Surprisingly, the improvements in reported accuracies were not as much as initially expected. As shown in Table 4.4, all ensembles obtained slightly improved accuracy when compared to a single model (third and fifth columns), but only up to half a percent.

This is most likely due to a phenomenon observed during training of these models. Specifically, if a number of feature maps used for first and second convolution layer is large (>30), the effects of other hyperparameters, such as filter size and stride, are diminished. This effect is due to convolution layer being exposed to a large number of features and their combinations, thus lowering the feature variance between different AS-CNN models. Hence, many different sets of hyperparameters are mapped to models with similar outputs and accuracies. Finally, this observation is not necessarily bad, as it gives

	Binary		Fine-grained	
# of models	Accuracy (%)	△Accuracy (%)	Accuracy (%)	△Accuracy (%)
3	84.7	+0.2	43.7	+0.1
5	84.6	+0.3	43.8	+0.2
7	85.1	+0.5	43.9	+0.3
10	85.0	+0.5	43.9	+0.3

Table 4.4: AS-CNN ensemble results

	Accuracy (%)	
Model	Binary	Fine-grained
MAX-TDNN	77.1	37.4
NBOW	80.5	42.4
RecNN	82.4	43.2
MV-RNN	82.9	44.4
ARecNN	**83.2**	**44.5**
RNTN	85.4	45.7
DCNN	86.8	48.5

Table 4.5: ARecNN binary and fine-grained accuracy

single models greater predictive capability, but it explains the unexpectedly minor improvements obtained by ensembles of such models.

4.1.3 ARecNN results

Here we compare the performance of our ARecNN model to a range of state-of-the-art models, as well as the original RecNN model. In Table 4.5, we can see the results of ARecNN compared to other state-of-the-art NN models. ARecNN significantly outperforms all the benchmarks models discussed in chapter 4.1.1, and scores higher in fine-grained accuracy and slightly lower in binary, when compared to our AS-CNN model. More importantly, ARecNN is able to outperform both the classic RecNN model, as well as its matrix-vector variation (MV-RNN), while at the same time not using the word embedding layer, which is not the case with RecNN and MV-RNN. Again, our model is outperformed by a couple of other RNN and CNN models using the aforementioned embedding. On the other hand, as also discussed in previous chapter, we believe that greatly increased transferability and cross-domain applicability of this model more than makes up for the differences in reported metrics.

In Figure 4.4, we can see node and root level accuracies as a function of number of epochs.

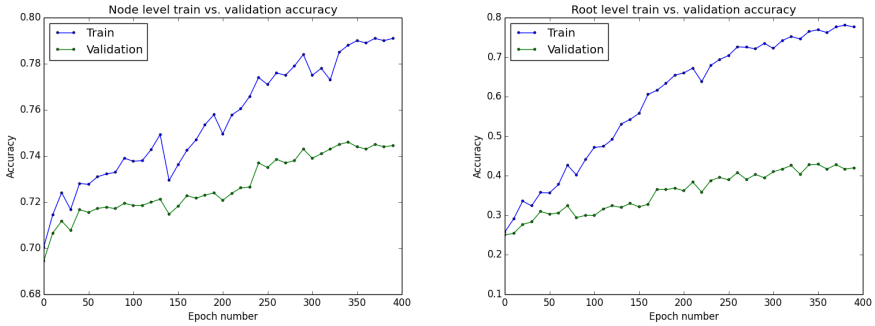


Figure 4.4: ARecNN node and root level training accuracy

It is very interesting to observe the normalized confusion matrix for ARecNN model (Figure 4.5). On the root level diagonal, we can see a very strong accuracy (around 50%) for all the classes except neutral. This stands in sharp contrast to the AS-CNN model, who achieves the highest accuracy on the neutral class. Given this, it is highly likely that an ensemble of AS-CNN and ARecNN models would give exceptional results, as they seem to be an excellent complement to each other.

The reason behind this can be seen in the recursive nature of the algorithm. The tree structure allows for creation of sentiment phrases from bottom-up, that then combine into more complex linguistic structures. This kind of structure allows for dealing with difficult examples containing things such as contrastive conjunction (X but Y structure), high-level negation and double negation. On the other hand, a large number of neutral phrases in the tree can easily be impacted by a slightly positive or negative part, thus swaying the prediction to one side. Hence, we can see that 86% of all the errors for the neutral class indeed do come from the neighbouring classes.

We explore these ideas further by hand, by delving deeper into the operation of the network, and visualizing its outputs. In Figure 4.6a, we can see an example of correctly classified contrastive conjunction, which other bag of word classifiers misclassify. Moreover, in Figure 4.6b, we have an example of pseudo high-level negation. What this means is that the negation is actually localized, and has little effect on the rest of the sentence, which again would not be visible in models with a non-recursive approach. Finally, we show an example of real high-level negation in Figure 4.6c, which was misclassified by the classic RecNN model.

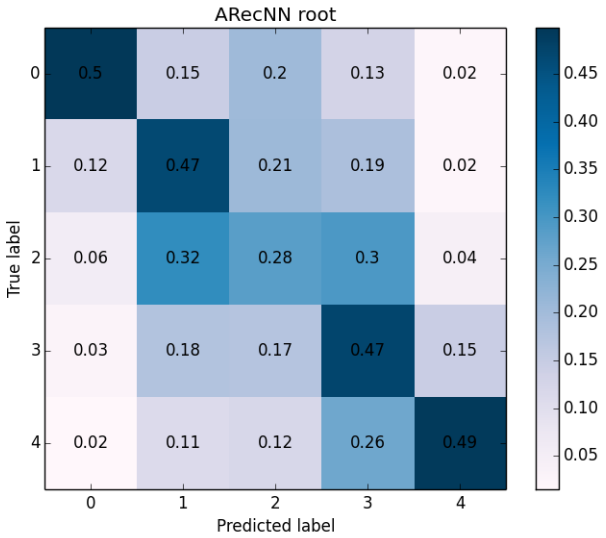


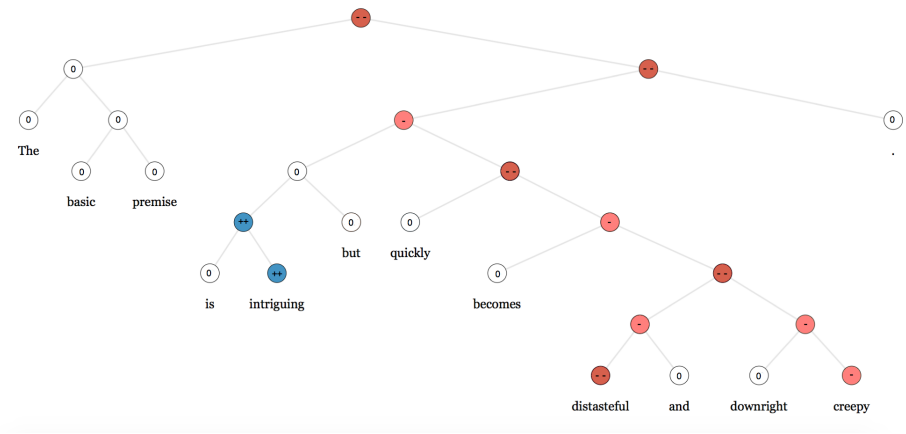
Figure 4.5: Fine-grained root level ARecNN model normalized confusion matrix

4.2 Twitter data

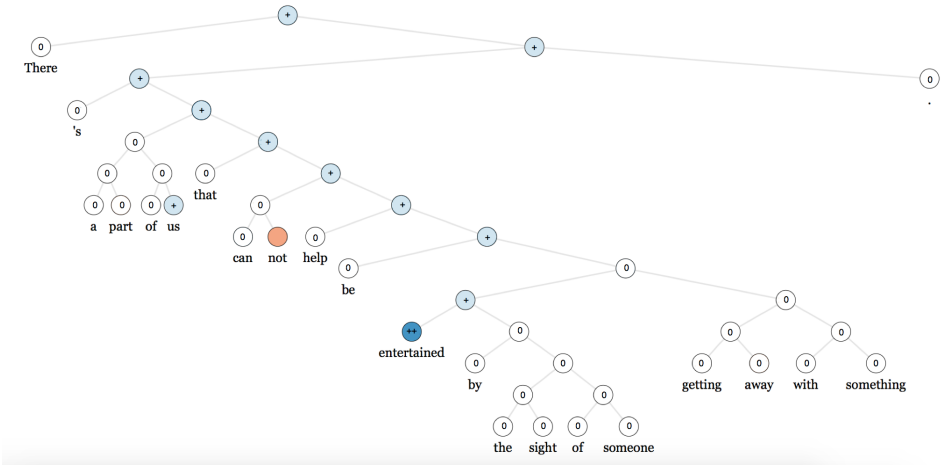
As a way of testing cross-domain applicability, we applied the models we developed on Twitter data, that we collected during August London Tube strike, in the week of August 5th. The corpus consists of 16,204 tweets, spread over 24 hours, starting from Wednesday 5th, at 16:00, 2 hours before the start of the strike, and ending on Thursday 6th, at 15:00.

We applied the AS-CNN and ARecNN in order to analyze the distribution of sentiment in the tweets for each hour of this period. A number of interesting observations were made. Namely, as seen in Figure 4.7, the sentiment was lowest at points where the strike began and people were stuck in their afternoon commute. Moreover, there is another spike in negative sentiment around 9-10 PM where people were most likely trying to get home after their evening activities. No data was collected during the night, hence a flat line on the graph. Finally, we can see another major spike at around 8:00 AM on Thursday, when the morning commuters were unable to get to their jobs.

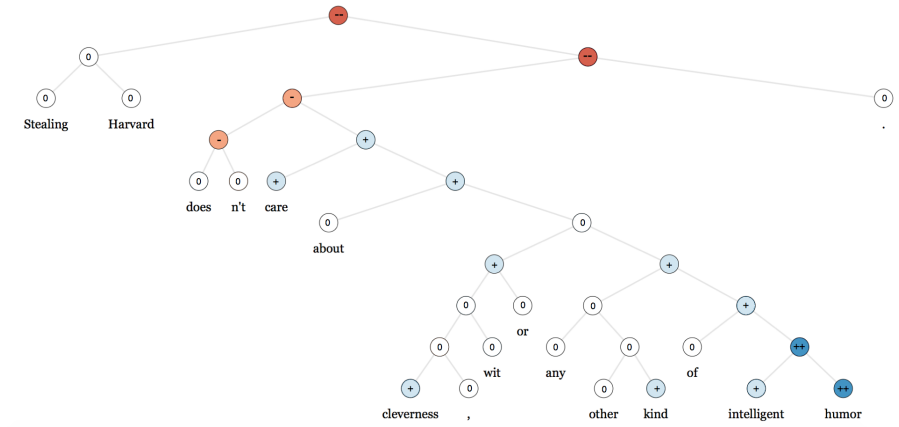
This analysis clearly shows the usefulness of sentiment mining models, as well as the ability of AS-CNN and ARecNN models, to be applied not just to a narrow domain of movie reviews, but to a large domain of textual content including highly unstructured, short and informal speech found on Twitter.



(a) Contrastive conjunction classification example



(b) Pseudo high-level negation classification example



(c) Real high-level negation classification example

Figure 4.6: Example outputs of ARecNN fine-grained model

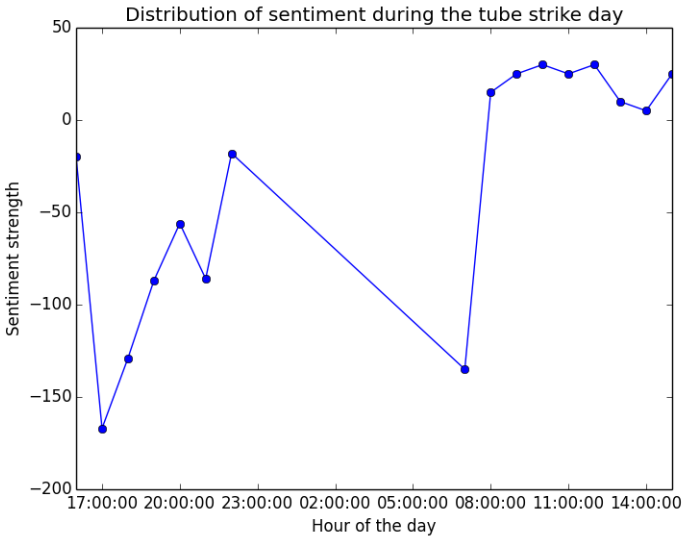


Figure 4.7: Tube strike sentiment analysis

4.3 Chapter summary

In this chapter we analyzed the performance of two novel models, namely AS-CNN and ARecNN, mainly on the SSTb movie reviews dataset, for both binary and fine-grained classification.

Firstly, we performed a detailed error analysis of benchmark results, obtained using a number of different features, encodings, and hyperparameters.

Secondly, we made a comparison of AS-CNN and ARecNN models to these benchmark models, as well as state-of-the-art NN models for SA, and laid out our reasoning with regards to reported accuracies.

Thirdly, we did a comprehensive analysis of errors using confusion matrices, and used this to point out strengths and weaknesses in all presented models.

Moreover, we looked deeper into functioning of the ARecNN model, and found interesting examples that the network managed to correctly classify, that were considered difficult by more simple approaches.

Furthermore, we examined the possibility of using ensemble methods on AS-CNN models, and reasoned about somewhat surprising results obtained.

Finally, the chapter concludes with tests for both models in a different setting and a completely unrelated domain, using data collected from Twitter, about the London Tube strike in August 2015.

Chapter 5

Conclusion and future work

In this thesis we have presented and implemented two novel deep neural network (NN) architectures for binary and fine-grained document level sentiment analysis, namely Adaptive Switching Convolutional Neural Network (AS-CNN) and Augmented Recursive Neural Network (ARecNN). Both models were trained on the Stanford Sentiment Treebank (SSTb) dataset, with reported results outperforming conventional baseline methods by 4% in binary case, and 3.5% in fine-grained case. Moreover, we also outperform a number of state-of-the-art NN models, such as RecNN and MV-RNN.

The contributions of this research are multiple.

Firstly, one of the main strengths of both models is strong generalization and multi-domain applicability due to usage of a separately trained neural language models in the network input layer. This layer allows both models to obtain meaningful word vector representation for more than one million words, which is a far greater number than that obtained by using a word embedding matrix in the input layer. We have shown this to be the case by applying our models to a dataset of tweets, consisting of short, highly informal and unstructured text, while obtaining meaningful analysis results.

Secondly, for AS-CNN model, we have defined and examined two novel techniques for training and optimizing such networks. The first one is GPU memory switching, which allows us to accommodate large datasets while training our network on a small number of GPU units. By using this switching technique, we can experiment with larger hyperparameters or word vectors, and be sure that we are able to get the most out of out data. The second technique is adaptive look-ahead hyperparameter optimization, which performs an improved Gaussian random search in the hyperparameter space, by fitting a

regression model using all the previous runs, thus narrowing down the search space as it progresses. Again, this technique can quickly find the optimal hyperparameter neighborhood, is easy to implement and is excellent for models where exhaustive search can not be performed.

Thirdly, for ARecNN model, we introduced a novel way of increasing the complexity of the decision boundary, by adding the additional augmentation step. Our experiments have shown that this improves the representational power of the model and its accuracy, when compared to the standard RecNN model.

In conclusion, we have presented two novel deep NN architectures along with ways of training and optimizing them, that obtain excellent accuracy scores, while retaining strong generalization capabilities and multi-domain applicability.

Finally, we would like to point out a number of promising avenues that could warrant further research.

As the use of embedding matrix in the input layer is a common feature of state-of-the-art models, we propose a hybrid approach, that keeps the strong generalization capabilities of our models, while making them more adaptable in the multi-domain setting. More specifically, at the moment, word vectors are trained in an unsupervised way, but allowing those vectors to change while training a model for a specific domain would allow for greater accuracy on that domain, while still keeping the ability to query the neural model in the input layer against a corpus of more than one million words. In addition to that, it would also be very interesting to further explore generalization capacity of presented models in a number of different domains, such as news or product reviews.

For training the AS-CNN models, some form of regularization, such as l_2 along with dropout in multiple layers could reduce the overfitting that tends to start happening around epoch 25, and allow models to be trained for longer, thus increasing their accuracy.

As mentioned before in Chapter 4, we also believe that an ensemble consisting of multiple AS-CNN and ARecNN models would give excellent results, as they both have complementary strengths and weaknesses.

In a more applied sense, we are also planning to start a long term integration project with Signal Media, as they are looking to add sentiment analysis capabilities to their existing NLP pipeline.

As a final note, the code of this project is available on GitHub (hyperlink).

Bibliography

[1] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data*. AMLBook, 2012.

[2] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines*. *Cognitive science*, 9(1):147–169, 1985.

[3] Robert Andrews, Joachim Diederich, and Alan B Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based systems*, 8(6):373–389, 1995.

[4] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[5] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[7] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[8] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

[9] Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71, 1996.

[10] Alan D Blair and Jordan B Pollack. Analysis of dynamical recognizers. *Neural Computation*, 9(5):1127–1142, 1997.

[11] John Blitzer, Kilian Q Weinberger, Lawrence K Saul, and Fernando CN Pereira. Hierarchical distributed representations for statistical language modeling. *Advances in Neural Information Processing Systems*, 17:185–192, 2005.

[12] Erik Boiy and Marie-Francine Moens. A machine learning approach to sentiment analysis in multilingual web texts. *Information retrieval*, 12(5):526–558, 2009.

[13] Danushka Bollegala, David Weir, and John Carroll. Cross-domain sentiment classification using a sentiment sensitive thesaurus. *Knowledge and Data Engineering, IEEE Transactions on*, 25(8):1719–1731, 2013.

[14] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1–8, 2011.

[15] Felipe Bravo-Marquez, Marcelo Mendoza, and Barbara Poblete. Meta-level sentiment models for big social data analysis. *Knowledge-Based Systems*, 69:86–99, 2014.

[16] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[17] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[18] Richard P Brent. *Algorithms for minimization without derivatives*. Courier Corporation, 2013.

[19] Miguel A Carreira-Perpinan and Geoffrey E Hinton. On contrastive divergence learning. In *Proceedings of the tenth international workshop on artificial intelligence and statistics*, pages 33–40. Citeseer, 2005.

[20] William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. *Ann Arbor MI*, 48113(2):161–175, 1994.

[21] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[22] Helen Costa, Luiz HC Merschmann, Fabrício Barth, and Fabrício Benevenuto. Pollution, bad-mouthing, and local marketing: The underground of location-based social networks. *Information Sciences*, 279:123–137, 2014.

[23] Kristof Coussement and Dirk Van den Poel. Improving customer attrition prediction by integrating emotions from client/company interaction emails and evaluating multiple classifiers. *Expert Systems with Applications*, 36(3):6127–6134, 2009.

[24] Balázs Csanád Csáji. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24, 2001.

[25] Manuel P Cuéllar, Miguel Delgado, and MC Pegalajar. An application of non-linear programming to train recurrent neural networks in time series prediction problems. In *Enterprise Information Systems VII*, pages 95–102. Springer, 2006.

[26] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[27] Li Deng and Dong Yu. Deep learning: methods and applications. *Foundations and Trends in Signal Processing*, 7(3–4):197–387, 2014.

[28] Misha Denil, Alban Demiraj, Nal Kalchbrenner, Phil Blunsom, and Nando de Freitas. Modelling, visualising and summarising documents with a single convolutional neural network. *arXiv preprint arXiv:1406.3830*, 2014.

[29] Cıcero Nogueira dos Santos and Maıra Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING), Dublin, Ireland*, 2014.

[30] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[31] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[32] Kazuki Fukushima, Sei Miyake, and Takao Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *Systems, Man and Cybernetics, IEEE Transactions on*, (5):826–834, 1983.

[33] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[34] Kunihiko Fukushima. Increasing robustness against background noise: Visual pattern recognition by a neocognitron. *Neural networks*, 24(7):767–778, 2011.

[35] Kunihiko Fukushima and Sei Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern recognition*, 15(6):455–469, 1982.

[36] Carl Friedrich Gauss. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium auctore Carolo Friderico Gauss*. sumtibus Frid. Perthes et IH Besser, 1809.

[37] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[38] Tomer Geva and Jacob Zahavi. Empirical evaluation of an automated intraday stock recommendation system incorporating both market data and textual news. *Decision support systems*, 57:212–223, 2014.

[39] M Ghiassi, J Skinner, and D Zimbra. Twitter brand sentiment analysis: A hybrid system using n-gram analysis and dynamic artificial neural network. *Expert Systems with applications*, 40(16):6266–6282, 2013.

[40] Anindya Ghose and Panagiotis G Ipeirotis. Estimating the helpfulness and economic impact of product reviews: Mining text and reviewer characteristics. *Knowledge and Data Engineering, IEEE Transactions on*, 23(10):1498–1512, 2011.

[41] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[42] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[43] Javier Gonzalez-Dominguez, Ignacio Lopez-Moreno, Hasim Sak, Joaquin Gonzalez-Rodriguez, and Pedro J Moreno. Automatic language identification using long short-term memory recurrent neural networks. In *Proc. Interspeech*, 2014.

[44] Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.

[45] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):855–868, 2009.

[46] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

[47] Andreas Griewank. Who invented the reverse mode of differentiation? *Optimization Stories, Documenta Matematica, Extra Volume ISMP (2012)*, pages 389–400, 2012.

[48] Mohamad H Hassoun. *Fundamentals of artificial neural networks*. MIT press, 1995.

[49] Geoffrey E Hinton. Distributed representations. 1984.

[50] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[51] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[52] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

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

[54] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[55] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[56] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[57] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.

[58] Xia Hu, Jiliang Tang, Huiji Gao, and Huan Liu. Unsupervised sentiment analysis with emotional signals. In *Proceedings of the 22nd international conference on World Wide Web*, pages 607–618. International World Wide Web Conferences Steering Committee, 2013.

[59] Ozan Irsoy and Claire Cardie. Bidirectional recursive neural networks for token-level labeling with structure. *arXiv preprint arXiv:1312.0493*, 2013.

[60] AG Ivakhnenko. The group method of data handling-a rival of the method of stochastic approximation. *Soviet Automatic Control*, 13(3):43–55, 1968.

[61] AG Ivakhnenko. Polynomial theory of complex systems. *Systems, Man and Cybernetics, IEEE Transactions on*, (4):364–378, 1971.

[62] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[63] Hanhoon Kang, Seong Joon Yoo, and Dongil Han. Senti-lexicon and improved naïve bayes algorithms for sentiment analysis of restaurant reviews. *Expert Systems with Applications*, 39(5):6000–6010, 2012.

[64] Andrej Karpathy. CS231n: Convolutional Neural Networks for Visual Recognition - Module 1, Part 2. `https://cs231n.github.io/neural-networks-2/`, 2015. [Online; accessed 2015-08-27].

[65] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *arXiv preprint arXiv:1412.2306*, 2014.

[66] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[67] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[68] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

[69] AN Kolmogoro. On the representation of continuous functions of several variables as superpositions of functions of smaller number of variables. In *Soviet. Math. Dokl*, volume 108, pages 179–182, 1956.

[70] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[71] Himabindu Lakkaraju, Richard Socher, and Chris Manning. Aspect specific sentiment analysis using hierarchical deep learning.

[72] Kevin J Lang, Alex H Waibel, and Geoffrey E Hinton. A time-delay neural network architecture for isolated word recognition. *Neural networks*, 3(1):23–43, 1990.

[73] Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural computation*, 20(6):1631–1649, 2008.

[74] Nicolas Le Roux and Yoshua Bengio. Deep belief networks are compact universal approximators. *Neural computation*, 22(8):2192–2207, 2010.

[75] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[76] Yann LeCun et al. Lenet-5, convolutional neural networks. *Internet: http://yann. lecun. com/exdb/lenet*, 2013.

[77] David D Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *Machine learning: ECML-98*, pages 4–15. Springer, 1998.

[78] Tsungnam Lin, Bil G Horne, Peter Tiňo, and C Lee Giles. Learning long-term dependencies in narx recurrent neural networks. *Neural Networks, IEEE Transactions on*, 7(6):1329–1338, 1996.

[79] Zachary C Lipton. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[80] Bing Liu. Sentiment analysis and subjectivity. *Handbook of natural language processing*, 2:627–666, 2010.

[81] Bing Liu. Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167, 2012.

[82] Ying Liu, Jian Jin, Ping Ji, Jenny A Harding, and Richard YK Fung. Identifying helpful online reviews: a product designer's perspective. *Computer-Aided Design*, 45(2):180–194, 2013.

[83] Yue Lu, Malu Castellanos, Umeshwar Dayal, and ChengXiang Zhai. Automatic construction of a context-aware sentiment lexicon: an optimization approach. In *Proceedings of the 20th international conference on World wide web*, pages 347–356. ACM, 2011.

[84] Minh-Thang Luong, Richard Socher, and Christopher D Manning. Better word representations with recursive neural networks for morphology. *CoNLL-2013*, 104, 2013.

[85] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, and Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5):555–559, 2003.

[86] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 5(4):1093–1113, 2014.

[87] Pankaj Mehta and David J Schwab. An exact mapping between the variational renormalization group and deep learning. *arXiv preprint arXiv:1410.3831*, 2014.

[88] Tomas Mikolov. *Language Modeling for Speech Recognition in Czech*. PhD thesis, Masters thesis, Brno, FIT BUT, 2007.

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

[90] Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukas Burget, and Jan Cernockỳ. Empirical evaluation and combination of advanced language modeling techniques. In *INTERSPEECH*, number s 1, pages 605–608, 2011.

[91] Tomáš Mikolov, Jiří Kopeckỳ, Lukáš Burget, Ondřej Glembek, and Jan Honza Černockỳ. Neural network based language models for highly inflective languages. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 4725–4728. IEEE, 2009.

[92] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[93] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013.

[94] Marvin Minsky. Steps toward artificial intelligence. *Computers and thought*, 406:450, 1963.

[95] Milad Mohammadi, Rohit Mundra, and Richard Socher. CS224D: Deep learning for NLP - Lecture Notes: Part IV. `http://cs224d.stanford.`

edu/lecture_notes/LectureNotes4.pdf, 2015. [Online; accessed 2015-08-27].

[96] Guido Montufar and Nihat Ay. Refinements of universal approximation results for deep belief networks and restricted boltzmann machines. *Neural Computation*, 23(5):1306–1319, 2011.

[97] Rodrigo Moraes, Joao Francisco Valiati, and Wilson P Gavião Neto. Document-level sentiment classification: An empirical comparison between svm and ann. *Expert Systems with Applications*, 40(2):621–633, 2013.

[98] Tony Mullen and Nigel Collier. Sentiment analysis using support vector machines with diverse information sources. In *EMNLP*, volume 4, pages 412–418, 2004.

[99] Ramanathan Narayanan, Bing Liu, and Alok Choudhary. Sentiment analysis of conditional sentences. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 180–189. Association for Computational Linguistics, 2009.

[100] Tetsuya Nasukawa and Jeonghee Yi. Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 70–77. ACM, 2003.

[101] Kamal Nigam, John Lafferty, and Andrew McCallum. Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, volume 1, pages 61–67, 1999.

[102] Brendan O'Connor, Ramnath Balasubramanyan, Bryan R Routledge, and Noah A Smith. From tweets to polls: Linking text sentiment to public opinion time series. *ICWSM*, 11(122-129):1–2, 2010.

[103] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in neuroscience*, 7, 2013.

[104] Randall C O'Reilly and Michael J Frank. Making working memory work: a computational model of learning in the prefrontal cortex and basal ganglia. *Neural computation*, 18(2):283–328, 2006.

[105] Alvaro Ortigosa, José M Martín, and Rosa M Carro. Sentiment analysis in facebook and its application to e-learning. *Computers in Human Behavior*, 31:527–541, 2014.

[106] Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *LREC*, volume 10, pages 1320–1326, 2010.

[107] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 115–124. Association for Computational Linguistics, 2005.

[108] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135, 2008.

[109] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 79–86, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[110] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2012.

[111] Romain Paulus, Richard Socher, and Christopher D Manning. Global belief recursive neural networks. In *Advances in Neural Information Processing Systems*, pages 2888–2896, 2014.

[112] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543, 2014.

[113] Sudeep Raja. Indian text recognition using bidirectional long-short term memory neural networks. `http://cse.iitkgp.ac.in/~psraja/projects.html`, 2014. [Online; accessed 2015-08-27].

[114] Antonio Reyes and Paolo Rosso. Making objective decisions from subjective data: Detecting irony in customer reviews. *Decision Support Systems*, 53(4):754–760, 2012.

[115] Paul J Roebber, Sara L Bruening, David M Schultz, and John V Cortinas Jr. Improving snowfall forecasting by diagnosing snow density. *Weather and Forecasting*, 18(2):264–287, 2003.

[116] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[117] DE Rumelhart. David e. rumelhart, geoffrey e. hinton, and ronald j. williams. *Nature*, 323:533–536, 1986.

[118] Hasim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proceedings of the Annual Conference of International Speech Communication Association (INTERSPEECH)*, 2014.

[119] M Rushdi Saleh, Maria Teresa Martín-Valdivia, Arturo Montejo-Ráez, and LA Ureña-López. Experiments with svm to classify opinions in different domains. *Expert Systems with Applications*, 38(12):14799–14804, 2011.

[120] Saed Sayad. Support vector machine - classification (svm). `http://www.saedsayad.com/support_vector_machine.htm`. [Online; accessed 2015-08-27].

[121] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[122] Matthias Scholz, Martin Fraunholz, and Joachim Selbig. Nonlinear principal component analysis: neural network models and applications. In *Principal manifolds for data visualization and dimension reduction*, pages 44–67. Springer, 2008.

[123] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681, 1997.

[124] Holger Schwenk. Continuous space language models. *Computer Speech & Language*, 21(3):492–518, 2007.

[125] Paul Smolensky. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. chapter information processing in dynamical systems: foundations of harmony theory. *MIT Press, Cambridge, MA, USA*, 15:18, 1986.

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

[127] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 151–161. Association for Computational Linguistics, 2011.

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

[129] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[130] Ilya Sutskever and Geoffrey E Hinton. Deep, narrow sigmoid belief networks are universal approximators. *Neural Computation*, 20(11):2629–2636, 2008.

[131] Songbo Tan, Xueqi Cheng, Yuefen Wang, and Hongbo Xu. Adapting naive bayes to domain adaptation for sentiment analysis. In *Advances in Information Retrieval*, pages 337–349. Springer, 2009.

[132] Mike Thelwall, Kevan Buckley, Georgios Paltoglou, Di Cai, and Arvid Kappas. Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology*, 61(12):2544–2558, 2010.

[133] Domonkos Tikk, László T Kóczy, and Tamás D Gedeon. A survey on universal approximation and its limits in soft computing techniques. *International Journal of Approximate Reasoning*, 33(2):185–202, 2003.

[134] Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics, 2003.

[135] B Tonkes and JH Wiles. Learning a context free task with a recurrent neural network. In *Fourth Biennnial Australasian Cognitive Science Conference*, volume 1, pages 1–7. The University of Newcastle, 1999.

[136] Peter D Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 417–424. Association for Computational Linguistics, 2002.

[137] A Vahed and Christian W Omlin. A machine learning method for extracting symbolic knowledge from recurrent neural networks. *Neural computation*, 16(1):59–71, 2004.

[138] Jose M. Vidal. Decision tree learning. `http://jmvidal.cse.sc.edu/talks/decisiontrees/allslides.html`, 2009. [Online; accessed 2015-08-27].

[139] Gang Wang, Jianshan Sun, Jian Ma, Kaiquan Xu, and Jibao Gu. Sentiment classification: The contribution of ensemble learning. *Decision support systems*, 57:77–93, 2014.

[140] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[141] Rui Xia, Chengqing Zong, and Shoushan Li. Ensemble of feature sets and classification algorithms for sentiment classification. *Information Sciences*, 181(6):1138–1152, 2011.

[142] Fu Xianghua, Liu Guo, Guo Yanyan, and Wang Zhiqiang. Multi-aspect sentiment analysis for chinese online social reviews based on topic modeling and hownet lexicon. *Knowledge-Based Systems*, 37:186–195, 2013.

[143] Yang Yu, Wenjing Duan, and Qing Cao. The impact of social and conventional media on firm equity value: A sentiment analysis approach. *Decision Support Systems*, 55(4):919–926, 2013.

[144] Taras Zagibalov and John Carroll. Automatic seed word selection for unsupervised sentiment classification of chinese text. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 1073–1080. Association for Computational Linguistics, 2008.

[145] JM Zarada. Artificial neural networks, 1992.

[146] Zheng Zeng, Rodney M Goodman, and Padhraic Smyth. Discrete recurrent neural networks for grammatical inference. *Neural Networks, IEEE Transactions on*, 5(2):320–330, 1994.

[147] Lei Zhang and Bing Liu. Identifying noun product features that imply opinions. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 575–580. Association for Computational Linguistics, 2011.

[148] Xiang Zhang and Yann LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

[149] Shusen Zhou, Qingcai Chen, and Xiaolong Wang. Active deep networks for semi-supervised sentiment classification. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 1515–1523. Association for Computational Linguistics, 2010.