NTNU
Norwegian University of
Science and Technology

# Deep Learning with emphasis on extracting information from text data

**Tobias Liland Bjormyr**

Master of Science in Physics and Mathematics

| | |
|---|---|
| Submission date: | April 2016 |
| Supervisor: | Håvard Rue, MATH |
| Co-supervisor: | Thiago Martins, Yahoo! |

Norwegian University of Science and Technology
Department of Mathematical Sciences

# Preface

This thesis concludes my master's degree in Applied Physics and Mathematics with a specialization in Industrial Mathematics at the Norwegian University of Science and Technology (NTNU). The thesis was written at the Department of Mathematical Sciences under the supervision of Professor Håvard Rue and co-supervisor Thaigo Martins, data scientist at Yahoo!.

I would like to thank Håvard and Thiago for their guidance and interesting input. And I would especially like to thank Thiago for taking an interest in my thesis, this collaboration has been very rewarding and inspiring.

Tobias Bjormyr
Trondheim, April 3, 2016

**Abstract**

## Abstract

In this thesis the Natural Language Processing (NLP) problems of predicting the negative or positive sentiment of a movie review (sentiment analysis) and Automated Essay Grading (AES) were analyzed. The data set used for the movie review part is from the IMDB database and the essays were published by the Hewlett foundation. Features were retrieved by using both conventional methods, such as Bag of Words, and newer methods, such as word vectors. These features were used to train both conventional statistical methods and more computational demanding Deep Learning models. The results shows that the conventional methods still perform quite well relative to the new "hot" methods on the problems tested in this thesis. However, a significant increase in available data observations might change this.

## Sammendrag (Abstract in Norwegian)

## Sammendrag (Abstract in Norwegian)

I denne oppgaven blir "Naturlig Språk Prosessering" (Natural Language Processing (NLP)) problemene å predikere om en filmanmeldelse er negative eller positive (sentiment analyse) og automatisk stilretting analysert. Datasettet brukt for filmanmeldelse delen er fra IMDB sin database og stilene brukt har blitt publisert av Hewlett stiftelsen. Dataen ble representert både med tradisjonelle metoder, som "Bag of Words", og nyere metoder, som ord vektorer. Disse datarepresentasjonene ble brukt til å trene både vanlige statistiske metoder og mer beregningskrevende "Deep Learning" metoder. Resultatene viser at tradisjonelle metoder fortsatt presterer ganske bra relativt til de nye populære metodene testet i denne oppgaven. Denne oppførelsene kan imidlertid endres om datastørrelsen økes signifikant.

# Contents

# 1 Introduction

Natural Language Programming (NLP) is a challenging machine learning subject faced by statisticians and data scientists. Language can express emotion that is obscured by sarcasm, plays of word, ambiguity, etc. which can be not just misleading to a human not familiar with the context, but especially to statistical models which aren't trained on problem/domain specific features. Adding information from a domain expert would make the matter much easier, but this is not always feasible due to the size of data and features available. As part of the "Big Data" era the interest in Deep learning models, especially deep neural networks inspired by the architecture of the human brain, have been rekindled and with enough computing power these models have been shown to perform well on many challenging problems including NLP problems, even without some level of domain knowledge.

This thesis focuses on extracting information from text data. The focus is on how to best represent the data and how to model it with both common statistical models and deep learning models. The data sets analyzed in this thesis are the relatively large scale IMDB dataset, analyzed through sentiment analysis, and the relatively complex Hewlett Foundations Automated Essay Scoring dataset.

The thesis is split into chapters. Chapter 2 introduces the datasets and framework used in the text analysis. Chapter 3 introduces Natural Language Processing (NLP) theory. Chapter 4 introduces Deep Learning models relevant for text analysis. Chapter 5 presents the results obtained from the analysis of the text datasets. And Chapter 6 contains the conclusion of the work and mention possibilities of further work on the subject.

# 2 Dataset and framework used

In this chapter the framework, used to train some of the text representations and Deep Learning models, and the datasets, that are used to motivate theory and are experimented on in this thesis, are introduced.

## 2.1 Framework

The python library Keras[1] is the framework used for modelling some of the deep learning models used in this thesis. Keras is meant to be a minimalistic library with a focus on fast experimentation. It can be run on top of either TensorFlow or Theano, both enables running computations on GPU's. In this thesis the combination of Keras and Theano[2] is used. The python package Gensim[3] is used for handling the word vector models used.

Keras, Theano and Gensim are currently only available as beta versions. The used versions are the developer versions 0.2.0 of Keras and 0.8.0 of Theano and the general beta version 0.12.1 of Gensim (these developer versions are updated from day to day without changing version number as they are bleeding edge). This implies algorithms could be changed and that the toolbox currently available is not written in stone.

## 2.2 IMDB - Bag of Words Meets Bags of Popcorn

A data set of IMDB movie reviews specifically selected for sentiment analysis was collected in association with the publication Maas et al. (2011). The data set consists of a labeled data set of 50,000 IMDB movie reviews. The sentiment is binary, meaning the IMDB rating below 5 results in a sentiment score of 0, and ratings above 6 has a sentiment score of 1. This means that reviews with ratings 5 and 6 are not part of this dataset, this is to make the sentiments clearly separated in their respective part of the "negative/positive"-scale. No movie has more than 30 reviews. The labeled dataset is split 50/50 into a training set and test set, each set contains 25,000 movie reviews. The training and test set does not contain any of the same movies. In addition there is also another 50,000 (unlabeled) IMDB reviews provided without any rating labels, which can be used for training relevant word vectors for example.

[1]Keras: http://keras.io
[2]http://deeplearning.net/software/theano/
[3]https://radimrehurek.com/gensim/

The dataset[4] can be found at Kaggle[5] and is described in greater detail there. Kaggle is a platform where companies and researchers can post their data in form of predictive modelling and analytics competitions. Statisticians and data scientists from all over the world participate in these competitions. This crowd-sourcing approach motivates varied problem solving approaches.

## 2.3 Hewlett Foundation: Automated Essay Scoring - Dataset

The Hewlett Foundation[6] released a dataset of essays for the Automated Essay Scoring competition[7] at Kaggle. The dataset contains eight essay sets, each essay set contains around 1,800 essays (except the 8th one which contains around 700 essays). These essays sets length range from an average length of 150 to 350 words (except the 8th one which has an average length of 650 words). All of these essays were hand graded by two different graders. And each of these eight essay sets has their own unique characteristics. This variability was intended to test the limits of participants algorithms scoring capabilities. The grade levels of the essays range from grade 7 to 10.

All the essay sets are graded over one domain, except for essay set 2 which is graded over two domains ("Writing Applications" and "Language Conventions"). The final domain score is a function of the two different graders. This function varies in the different essay sets. This variation in scoring is ignored and only the final domain score is used for training and observing the validity of the models used in this thesis. The range of the domain scores varies in the different essay sets.

A more detailed description of the dataset can be found at Kaggle[8].

[4]IMDB Dataset: https://www.kaggle.com/c/word2vec-nlp-tutorial/data
[5]Kaggle: https://www.kaggle.com/
[6]Hewlett Foundation: http://www.hewlett.org/
[7]Automated Essay Scoring - Dataset: https://www.kaggle.com/c/asap-aes
[8]https://www.kaggle.com/c/asap-aes/data

# 3 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a field of computer science concerned with interactions between computers and human natural languages. The main concern is enabling computers to retrieve meaning from human languages. This can be achieved by modelling the text data using handmade rules or features, or by letting an algorithm construct abstract features which model the data well. The latter approach is the main concern of the deep learning community and this thesis.

## 3.1 Sentiment analysis

Sentiment analysis is a common NLP task which aims to identify polarity of a text document. This is usually done on consumer reviews for market analysis, for example. The simplest case where one only discriminate between positive and negative sentiment can be modelled as a binary classification problem. The IMDB datasets classification task is an example of this.

## 3.2 Input types

The input types used to represent the text data in this thesis are non-sequential input, defined in Section 3.2.2, and sequential input, defined in Section 3.2.3. Transforming the text data into these representations requires some pre-processing of the text.

### 3.2.1 Text pre-processing

The text can be pre-processed by creating a dictionary (or using a predefined one) which contains a unique identification key for each word in the **text corpus**[1] (unknown words and words or characters deemed insignificant are usually given a predefined "dump"-key, usually 0). Usually different grammatical versions of a word is viewed as different words. Non-word objects are usually removed from the document if they have no specific significance for the objective domain. Different capitalized variations of words may appear (e.g. New York, NEW YORK), but this problem can be avoided by ignoring the difference between upper and lower case.

It is convenient to numerate a word with the value according to how frequent it occurs in the corpus or by it's importance as calculated in a TF-IDF representation (this representation will be introduced in section 3.2.2.2). Meaning that the 4th most

[1]A **text corpus** is a set of texts, for example a collection of text documents.

frequent or important word is encoded with the key "4". Often the most frequent and infrequent words are removed as they are deemed insignificant (i.e. they do not contain any information of enough significance and may contribute to making the model too complex, resulting in overfitting).

### 3.2.2 Non-sequential input

Non-sequential input is the simplest input variant which ignore the placement of words and as such ignore any spatial correlation between words (i.e. the significance of the words placement according to each other). Among these input types Bag of Words and TF-IDF (term frequency–inverse document frequency) are quite popular and will be introduced here.

#### 3.2.2.1 Bag of Words

Bag of Words is a simple approach which uses a dictionary $G$ to create a counting vector $v$ of the length $|G|$, where each element $v_i$ refers to the number of occurrences of the word $G_i$ in a document. This approach is often used for document classification.

**An example of a Bag of Words representation:**
Here we view the whole text corpus as the two "documents" listed below.

1. Ron eats potatoes. His sister stole a potato.
2. Ron ate a potato. His sister still stole a potato.

The corresponding dictionary becomes (here in the order as observed):
[ "Ron", "eats", "potatoes", "His", "sister", "stole", "a", "potato", "ate", "still" ]
, which contains 10 distinct words. And using the indexes as given in the dictionary both documents can be represented as vectors of length 10. These vectors becomes:

1. $[1,1,1,1,1,1,1,1,0,0]$
2. $[1,0,0,1,1,1,2,2,1,1]$

#### 3.2.2.2 TF-IDF

TF-IDF (often written as tf-idf) stands for "term frequency–inverse document frequency", which is a more advanced non-sequential representation form than Bag of Words. This statistic is intended to imply how important a word is to a document based on the text corpus. The approximated importance of a word increases proportionally to the number of times the word appears in the document and decreases based on how often it appears in the rest of the text corpus. Variations of TF-IDF are often used in search engines as a tool for ranking the relevance of a document based on a search query. The TF-IDF weight is composed of the two terms Term frequency (TF) and the Inverse Document Frequency (IDF).

**Term Frequency** (TF) measures how frequent a term appears in a given document. Because of the variability of document lengths it is possible that a term appears more often in a longer document than a shorter more relevant document. This is incorporated by normalizing the number of appearances of a word by the total number of words in the document. The Term Frequency is defined as

$$TF(t,d) = \frac{N(t,d)}{\sum_t N(t,d)},$$

where $N(t,d)$ is the number of times the word with index $t$, in the dictionary, appears in document $d$. $TF(t,d)$ is then the frequency that the word with index $t$, in the dictionary, appears in document $d$.

**Inverse Document Frequency** (IDF) measures how important a term is. This is done by penalizing frequent terms (e.g. "is", "of", "that", etc.) and scaling up the importance of infrequent terms. One variant of the IDF is defined as

$$IDF(t) = \log\left(\frac{|D|}{\sum_{d\in D}(N(t,d)! = 0)}\right) = \log\left(\frac{|D|}{|D| - [\sum_{d\in D}(N(t,d) == 0)]}\right)$$

where $D$ is the set of documents and $|D|$ gives the total number of documents.

The simplest TF-IDF method is defined as

$$\text{TF-IDF}(t,d) = TF(t,d) \cdot IDF(t). \tag{3.1}$$

A common modification of this is

$$\text{TF-IDF}(t,d) = TF(t,d) \cdot (1 + IDF(t)), \tag{3.2}$$

which effectively gives a word that occurs in every document the TF-IDF$(t,d)$ value equal to $TF(t,d)$ instead of zero. This modification is used for the TF-IDF modelling in this thesis.

**An example of a TF-IDF representation:**
The text corpus is the same as in the Bag of Words example. Here we view the whole text corpus as the two "documents" listed below.

1. Ron eats potatoes. His sister stole a potato.
2. Ron ate a potato. His sister still stole a potato.

The corresponding dictionary becomes (here in the order as observed):
[ "Ron", "eats", "potatoes", "His", "sister", "stole", "a", "potato", "ate", "still" ]
, which contains 10 distinct words. Using the indexes as given in the dictionary both documents can be represented as vectors of length 10. There are two documents so $|D| = 2$, and the first sentence contains 8 words and the second 10 words. The $TF(t,1)$ and $TF(t,2)$ vectors are then equal to the corresponding Bag of words vectors divided by the number of words in each sentence:

1. $\frac{1}{8} \times [1,1,1,1,1,1,1,1,0,0]$
2. $\frac{1}{10} \times [1,0,0,1,1,1,2,2,1,1]$

The how many documents each word appears in is given by the vector

$$[2,1,1,2,2,2,2,2,1,1,1], \tag{3.3}$$

and since $\log(1) = 0$ and $\log(2) \approx 0.69$ the $IDF$ vector becomes

$$0.69 \times [0,1,1,0,0,0,0,0,1,1]. \tag{3.4}$$

Using the TF-IDF modification TF-IDF$(t,d) = TF(t,d) \cdot (1 + IDF(t))$ we get

1. $\frac{1}{8} \times [1,1.69,1.69,1,1,1,1,1,0,0]$
2. $\frac{1}{10} \times [1,0,0,1,1,1,2,2,1.69,1.69]$

### 3.2.3 Sequential input

As the previous text representations methods does not retain word order a more advanced representation method is desirable. One approach is to represent the text data as a sequence, this retains the word order and results in insight of word placement correlation. This is done by creating a vector of smaller or equal length to the document and transforming the document into a sequence of numbers where each word is enumerated in accordance to a dictionary. Usually it is desired that all the vectors are of the same length. This is done by choosing a desired length and cutting the vectors that are too long (and removing the redundant parts, an example is removing the end of the documents that are too long) and padding 0's to the vectors that are too short so that they all are of the same length.

**A word sequence example:**
Assume the text corpus contain the two documents:

1. Leonardo was amazing in Inception.
2. Leonardo needs an Oscar.

Using the dictionary:

- Leonardo: 1
- was: 2
- amazing: 3
- in: 4
- Inception: 5
- needs: 6
- an: 7
- Oscar: 8

And creating word sequences of length 5 this results in the word sequences:

1. $[1,2,3,4,5]$
2. $[1,6,7,8,0]$

Utilizing this representation one can represent each word as a vector (popularly called a word vector) and not blindly assume that the worth of a word in a sentence is most effectively represented by a scalar value.

### 3.2.4 Word vectors

Representing each word as a vector will incorporate more complex information into the representation, which can be used to retrieve even more information from the interactions of each word and their placement relative to each other. One can either train one's own representations of each word or use pre-trained representations such as GloVe [2] and Googles word2vec [3]. The GloVe representation is introduced by Pennington, Socher, and Manning (2014). The word2vec representation is based on the papers Mikolov et al. (2013b), Mikolov et al. (2013a) and Mikolov, Yih, and Zweig (2013) which introduce, evaluate and improve the Continuous Bag-of-Words Model (CBOW) and the Continuous Skip-gram Model (popularly called the Skip-gram model).

The *TF-IDF* approach give us some idea of a word's relative importance in a given corpus, it however does not give any insight into the words semantic meaning. Word vectors have been shown to capture syntactic and semantic linguistic regularities well (Mikolov, Yih, and Zweig 2013). These word vectors are quite useful as features in NLP problems.

**A popular example showing that word vectors are able to capture semantic similarities between words:**
Assume that the words {"king", "queen", "man", "woman"} are represented by the vectors $v_{\text{king}}, v_{\text{queen}}, v_{\text{man}}, v_{\text{woman}}$. Then the relationship

$$v_{\text{king}} - v_{\text{man}} + v_{\text{woman}} \approx v_{\text{queen}} \tag{3.5}$$

holds (this behaviour is the result of a huge training set such as the Google News dataset of about 100 billion words). Another example is the relationship:

$$v_{\text{Paris}} - v_{\text{France}} + v_{\text{Italy}} \approx v_{\text{Rome}} \tag{3.6}$$

*Note: These results were reported by* Mikolov *et al. (2013b).*

Asymptotically it is intuitive that one would get better theoretical results if one train domain specific word vectors instead of using pre-defined ones. Of course in many circumstances one would need lot of training data and training time to get the domain specific word vectors to outperform the pre-trained vectors (which has been trained on extremely large datasets).

[2] Pre-trained GloVe word vectors: `http://nlp.stanford.edu/projects/glove/`
[3] Pre-trained word2vec word vectors: `https://code.google.com/p/word2vec/`

**A word sequence example with word vectors:**
Assume one has a weight matrix $W$ and the following sequence vectors:

1. $[1,2,3]$
2. $[2,1,0]$

Assume $W$ is given as:

$$W^T = \begin{array}{c} \begin{array}{cccc} \{0\} & \{1\} & \{2\} & \{3\} \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 2 \\ 0 & 1 & 2 & 2 \end{pmatrix} \end{array} \tag{3.7}$$

Each number in the sequence vectors refers to a row in the weight matrix $W$ which is the corresponding word vector. The initial row with id 0 models all unknown words (usually a zero vector) and the 3 other rows contains the known word representations. Using this weight matrix the word sequences can be modelled as matrices:

1. $\begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 2 \end{bmatrix}$
2. $\begin{bmatrix} 1 & 1 & 0 \\ 2 & 1 & 0 \end{bmatrix}$

Here each word in the sequence is replaced by the corresponding word vector, as a column (eg. the first word in a sequence is represented by column 1 and the $n$'th word in a sequence is represented by column $n$).

The weight matrix is kept for efficient representations as it would take too much memory to at all times represent each occurrence of a word as a vector when the vectors become large.

Methods for computing word vectors are described in section 4.6.

### 3.2.5 More advanced non-sequential input

More advanced non-sequential input representations like document vectors and word clusters use the word vector representations to compute their values. These representations are introduced here.

#### 3.2.5.1 Document Vectors - More advanced non-sequential input

Le and Mikolov (2014) introduced the concept of computing the average of the word vectors representing a sentence or document and use it as the input to a classification/regression model. This averaged vector is dubbed a Document vector and can be used to represent documents in a more semantically rich way than standard BoW and TF-IDF representations and is much cheaper to use for training than sequential input. Iyyer et al. (2015) used Document vectors as the input neural network models and efficiently trained a model with good results.

#### 3.2.5.2 Word clusters - Bag of Clusters

In word clusters the most similar words are paired together in clusters. Meaning that instead of counting words the Bag of Clusters representation counts occurrences of each cluster. Word clusters can easily be trained by using Support vector clustering (SVC) (a clustering method using support vector machines introduced by Ben-Hur et al. (2002)) to create word clusters (dubbed Bag of Clusters). Bekkerman et al. (2003) report good results from using word clusters for classification.

## 3.3 Methods to extend the vocabulary

Ways of extending the vocabulary, such as N-grams and Skip-grams, are introduced in this section.

### 3.3.1 N-grams

N-grams can be useful to represent phrases with unique meaning as single items. An example would be "Air Canada", as its meaning cannot easily be combined from the meanings of the separate items "Canada" and "Air". *n*-gram's are continuous sequences of *n* items from a given sequence (usually from text or speech) which can be used to extract such phrases.

**A n-gram example:**
Assume the sentence:

"The Shawshank Redemption is quite excellent."

The 1-gram and 2-gram representations of the sentence:

- 1-grams (uni-grams):
  { The, Shawshank, Redemption, is, quite, excellent }
- 2-grams (bi-grams):
  { The Shawshank, Shawshank Redemption, Redemption is, is quite, quite excellent }

#### 3.3.1.1 N-grams and tf-idf

N-grams are especially nice to use to boost a tf-idf representation as it can weigh important phrases with high values and unimportant phrases with low values. Any redundant phrases can be removed by only keeping the values over a certain threshold or the vocabulary with the *k*-highest values.

### 3.3.2 Skip-gram

Skip-gram modelling is a generalization of *n*-grams which handles data sparsity better than classic *n*-grams as shown by Guthrie et al. (2006) (i.e. in cases where the vocabulary is too small it can be extended with skip-grams to improve performance of the models used). Skip-grams gives the ability to skip words (items) in a sequence. A *k*-skip-*n*-gram contains all the sub-sequences of *n* words where each word is a distance *k* or less from the previous one.

**An skip-gram example:**
Assume the sentence:

"The potatoes are the real victims here!"

The resulting 1-skip-2-gram representation:

{ The potatoes, The are, potatoes are, potatoes the, are the, are real, the real, the victims, real victims, real here, victims here }

# 4 Deep Learning for Text Analysis

In this chapter Deep Learning models that are relevant to the Natural Language Processing problems dealt with in this thesis are introduced. Neural networks are some interesting Deep Learning models, as Hornik, Stinchcombe, and White (1989) established that a Multilayer perceptron (MLP), a simple Neural Network, is able to approximate any real valued function. The MLP and the other Deep Learning models introduced in this section can be seen as models extracting features from the available data, which are used in the final classification/regression layer of the model.

This Chapter introduces the Single Neuron Model, which is the basis of the Neural Network family; the Multilayer Perceptron (MLP) model; Back-propagation, the method used for training neural networks; Dropout, a regularization method specialized for neural networks; Word vector models, used to train word vector representations, such as GloVe, the Continuous Bag of Words (CBoW) model and the Skip-gram model; the Convolutional Neural Network model; the Recurrent Neural Network (RNN) family, with mentions of the LSTM and GRU.

## 4.1 The Single Neuron Model

Neural networks consists of multiple neurons in multiple layers, each neuron is modelled in the same way, usually with the same parameters in each layer. Each neuron is activated to a certain degree based on the input given. A single neuron model is illustrated in Figure 4.1. The neuron activation function is defined as

$$y = f(x) = \sigma\left(\sum_{i}^{K} w_i x_i\right) = \sigma\left(\mathbf{w}^T \mathbf{x}\right), \tag{4.1}$$

where $\mathbf{x}$ is the input vector and $\mathbf{w}$ is the weight vector. $f$ is usually a non-linear activation function that maps the vector $\mathbf{x}$ to the scalar output $y$. Often a bias is added by setting $x_0 = 1$ (where the corresponding $w_0$ acts as the bias).

## 4.2 Activation functions

An activation function is said to be activated if its output is non-zero. It is also said to have a strong activation if the output is relatively high and have a weak activation if its output is relatively small. An activation function is desired to be non-linear, continuously differentiable and monotonic, it is further desired that

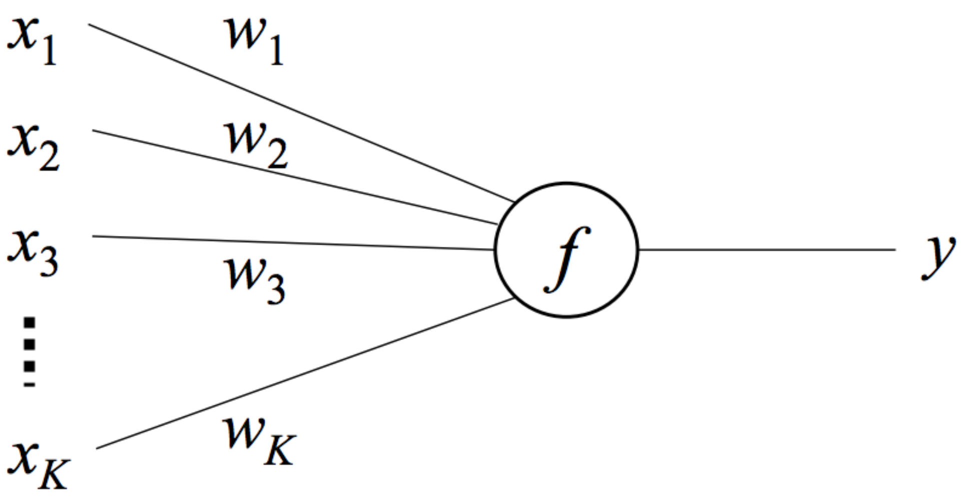


Figure 4.1: This figure shows an illustration of a single neuron model. (Source: http://alexminnaar.com/tag/deep-learning.html)

the function $f(x) \approx x$ when $x$ approaches 0. The activation functions are desired to be non-linear as this is a feature needed for the neural network to be an universal approximator (Chen and Chen 1995). Continuously differentiable activation functions are necessary for gradient-based optimization methods. Monotone activation functions guarantees a convex error surface of a single-layer model (Wu 2009). And if $f(x) \approx x$ when $x$ approaches 0 the networks can train more efficiently (if this is not satisfied weights must be initialized with care (Sussillo 2014)). There are many activation functions to choose from. The sigmoid, hyperbolic tangent and the ReLU activation functions are introduced and defined in this section. Some classification and regression functions used in the final layer of neural networks are also introduced.

### 4.2.1 Sigmoidal functions

The most common form of activation functions are the sigmoidal functions which are monotonically increasing functions that asymptotically approaches some value as the input approaches $\pm\infty$. The most common sigmoidal functions are the standard logistic function (usually referred to as the sigmoid function) and the hyperbolic tangent. The logistic sigmoid, motivated somewhat by the biological neurons, is defined as

$$f(x) = \frac{1}{1+e^{-x}} \in [0,1). \tag{4.2}$$

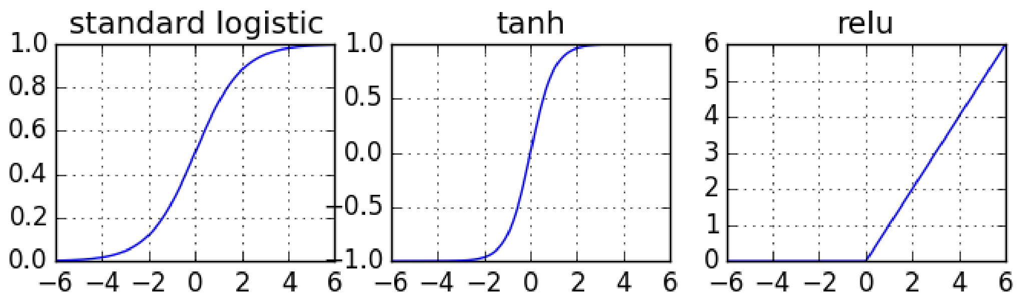


Figure 4.2: Plots of the standard logistic function $f(x) = 1/(1+e^{-x})$, the hyperbolic tangent $f(x) = \tanh(x)$ and the rectifier function (ReLU) $f(x) = \max(0, x)$.

The hyperbolic tangent which approximates the logistic sigmoids behaviour and normalize the data between -1 and 1 is defined as

$$f(x) = \tanh(x) \in (-1, 1). \tag{4.3}$$

Both sigmoidal functions are plotted in the figure 4.2.

The hyperbolic tanget is better for training models efficiently with back-propagation (this is an optimization method used for neural networks and which will be introduced in section 4.4). LeCun et al. (2012) motivates that sigmoidal functions that are symmetric around the origin are preferred because they on average produce outputs close to zero which results in a faster convergence. Both sigmoidal functions however face **the vanishing gradient problem**.

The vanishing gradient problem comes from how neural networks are trained with back-propagation. The error signal computed in an $n$-layer model consists of $n$ gradients in the range of $(-1, 1)$[1] multiplied together, this results in a small (vanishing) error signal and in turn results in slow training of the model.

### 4.2.2 Rectifier (ReLU)

Another activation function of interest is the rectifier function, also known as the ramp function, which as of 2015 is the most popular activation function for deep neural network according to LeCun, Bengio, and Hinton (2015). A unit using the rectifier activation function is called a **rectifier linear unit**, **ReLU**. The activation function is often referenced as ReLU in deep learning applications/programming. The rectifier function is defined as

$$f(x) = \max(0, x). \tag{4.4}$$

[1] Hyperbolic tangent: $(-1, 1)$; Sigmoid: $[0, 1)$

The rectifier function is plotted in figure 4.2. Glorot, Bordes, and Bengio (2011) argues that it is more biologically plausible than the logistic sigmoid and that it is more efficient to train than the hyperbolic tangent. Glorot, Bordes, and Bengio (2011) also shows that the rectifier function is "remarkably" adapted to sentiment analysis in text-based tasks. Further motivations for the Rectifier functions is that it results in sparse activations (on an average only about 50 % of the ReLU units are activated) and that it isn't afflicted by the vanishing gradient. It is however not differentiable close to 0, a way around this is the smooth approximation of the rectifier function called the **softplus** function. The softplus function is defined as

$$f(x) = ln(1 + e^x). \tag{4.5}$$

The softplus function however doesn't induce the sparsity that the ReLU function does. Other variants of the Rectifier have been tailored for various specific deep learning tasks, as for example the Leaky ReLU used in Maas, Hannun, and Ng (2013) and the Parametric ReLU used in He et al. (2015).

### 4.2.3 Final layer

For the final layer in the neural networks probabilistic classification functions, such as the softmax function, or regression functions, such as the linear activation function, are preferred depending on if it is a classification or regression problem. The softmax and linear activation functions are described in this section, along with a mention about how support vector machines might surpass their performance.

#### 4.2.3.1 Softmax

When the classification problem isn't binary but contains multiple class the softmax function is usually used, it is however also viable for the binary case. This is a generalization of the logistic function and is just another name for a multinomial classification model when one assumes that there exists no hierarchy among the classes. The softmax function is nice as it gives an approximation of the probability that a class is the correct one. The simplest approach is to simply choose the class with the highest probability, and ignore the rest. But since it is a probabilistic function it can also be used for a generative model. The softmax scores (probabilities) are computed by the normalizing function

$$\sigma_j = P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^T\mathbf{w}_j}}{\sum_k^K e^{\mathbf{x}^T\mathbf{w}_k}}, \tag{4.6}$$

$j \in \{0, 1, ..., K\}$ and $K$ is the number of classes. $P(y = j|\mathbf{x})$ is the probability of class $j$ being the correct class of the $K$ classes given the observations $\mathbf{x}$, and the weights $\mathbf{w}_j$.

#### 4.2.3.2 Linear

The linear activation layer is just a simple linear regression model trained on the incoming features. Given input features $x_i \in X$ and weights $w_i \in W$ where $i \in 1, \ldots, n$ the output of the linear layer is given as

$$y = f(x) = X^T W = \sum_{i}^{n} w_i x_i \tag{4.7}$$

#### 4.2.3.3 SVM

However Tang (2013) demonstrate a small but consistent advantage in classification problems of replacing the final softmax layer with a linear support vector machine (SVM). These findings could also imply that the SVM extension for regression, the Support Vector Regression (SVR) model (Smola and Vapnik (1997)), could outperform the final linear activation layer in regression problems. This behavior could also be motivated by the results in the experiment chapter (Chapter 5), where SVM classification outperforms Logistic Regression and SVR regression outperforms Linear Regression on representations of the datasets.

The current version of the Keras framework used to train the Deep Learning models in this thesis does not support SVM and SVR activation layers in the current version. So these activation functions were not tested.

### 4.2.4 Weight initialization

At initialization it is desirable that the weights are close to the center of the possible values of it's domain, so that the activation function operates in the domain where it is approximately linear and the gradients are close to their potential maximums. Glorot and Bengio (2010) recommends drawing the initial weights from the uniform distribution. The width of the uniform distribution sampled from depends on the activation function and the variable $n$, which is the sum of $n_{in}$, the number of input values given to the hidden layer that the weights belong to, and $n_{out}$, the number of hidden units in the hidden layer. This normalizing of the uniform distribution is meant to fulfill the objective of maintaining activation variances and back-propagated gradient variances.

For the standard logistic function draw from:

$$U[-4 \times \frac{\sqrt{6}}{\sqrt{n}}, 4 \times \frac{\sqrt{6}}{\sqrt{n}}] \tag{4.8}$$

For the hyperbolic tangent draw from:

$$U[-6 \times \frac{\sqrt{6}}{\sqrt{n}}, 6 \times \frac{\sqrt{6}}{\sqrt{n}}] \tag{4.9}$$

The scaling values 4 and 6 corresponds to the width of the area that the standard logistic and hyperbolic tangents haven't yet reached their maximum values, as $|x|$ values greater than 4 (logistic function) and 6 (hyperbolic tangent) results in $f(x)$ reaching it's min or max value. For the rectifier function, ReLU, He et al. (2015) recommends sampling the weights from the normal distribution $N(0,\sigma^2)$, with variance 0.01. If the layer is very large (wide) using a variance of 0.001 may induce better performance/results.

## 4.3 Multilayer Perceptron (MLP)

The Multilayer perceptron (MLP) model is a basic neural network that consists of multiple neurons. The MLP model with 1 hidden layer is illustrated in Figure 4.3 and consists of an input layer, a hidden layer and an output layer. The input layer consists of the input vector $\mathbf{x} = \{x_1,...,x_K\}$, with $K$ input variables. The hidden layer consists of the hidden vector $\mathbf{h} = \{h_1,...,h_N\}$, with $N$ neurons (where each neuron behave just as the single neuron model). And the output layer consists of the output vector $\mathbf{y} = \{y_1,...,y_M\}$, with $M$ neurons. Every element in the input layer is connected to every element in the hidden layer, where element $w_{ki}$ of weight matrix $W$ $(K \times N)$ indicates the weight associated with input element $k$ and hidden element $i$. The same connection structure is also present between the hidden layer and the output layer (i.e. that every element in the hidden layer is connected to every element in the output layer), and here element $w'_{ij}$ of weight matrix $W'$ $(N \times M)$ indicates the weight associated with hidden element $i$ and output element $j$. The output of hidden element $h_i$ is given by the equation

$$h_i = f(u_i) = f\left(\sum_{k=1}^{K} w_{ki} x_k\right) \; \forall i \in \{1,2,...,N\},$$

where $u_i$ is the input of the activation function of hidden element $h_i$. And the output values $y_j$ of the output layer $y$ are given by the equation

$$y_j = g(v_j) = g\left(\sum_{i=1}^{N} w'_{ij} h_i\right) \; \forall j \in \{1,2,...,M\},$$

where $v_j$ is the input sent to some activation function $g$ that computes the value for $y_j$. The weights $W$, $W'$ are trained by using Stochastic Gradient Descent as it is a computationally efficient alternative to standard optimization methods. The full optimization method is called back-propagation.

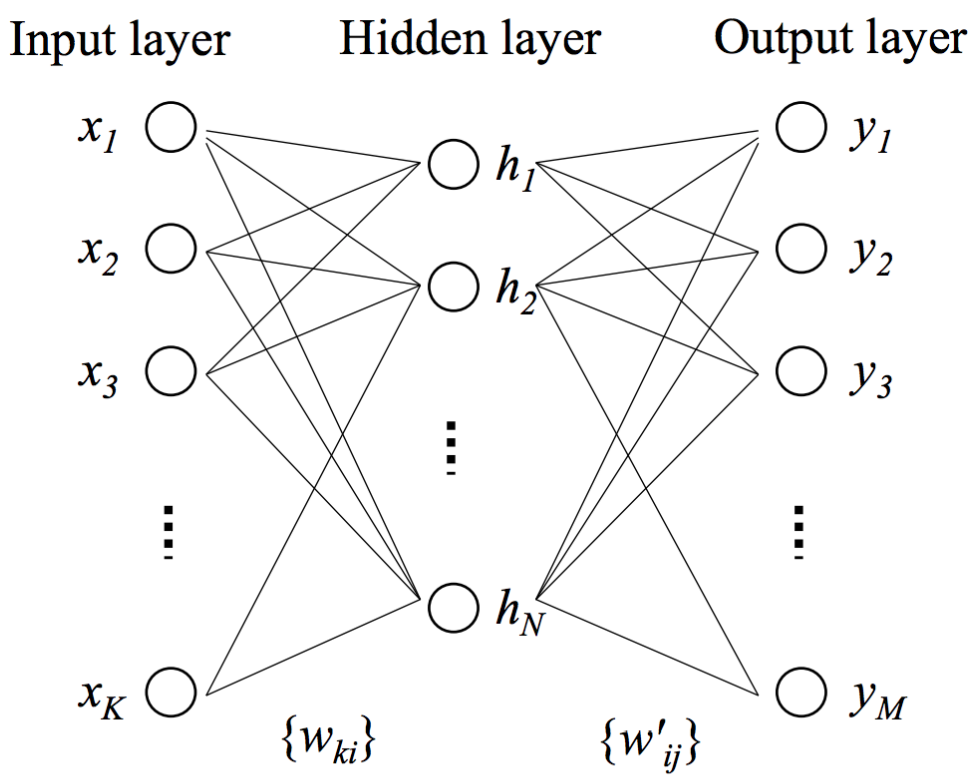


Figure 4.3: The MLP model with one hidden layer. (Source: http://alexminnaar.com/tag/deep-learning.html)

## 4.4 Back-Propagation

Back-propagation is the method used to train neural networks. Back-propagation is an abbreviation of "backward propagation errors", i.e. the final classification/value error gets propagated backwards in the network in order to update the weights.

**An example of using back-propagation on the MLP model with one hidden layer:** We want to train the MLP model by updating the weights $W$ and $W'$. We first find the gradient of the chosen loss function $E$ with respect to $W'$. Using the chain rule we get

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial v_j}\frac{\partial v_j}{\partial w'_{ij}},$$

where

$$\frac{\partial v_j}{\partial w'_{ij}} = h_i.$$

$\frac{\partial E}{\partial w'_{ij}}$ can be rewritten to

$$\frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j}$$

which makes it easier to compute the gradient when the loss and activation functions are known. Thus the gradient can be written as

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j} h_i \ \forall w'_{ij} \in W' \tag{4.10}$$

Next we find the gradient of the loss function $E$ with respect to $W$. Using the chain rule we get

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i}\frac{\partial h_i}{\partial u_i}\frac{\partial u_i}{\partial w_{ki}},$$

where

$$\frac{\partial u_i}{\partial w_{ki}} = x_k$$

and

$$\frac{\partial E}{\partial h_i} = \sum_j^M \left( \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j}\frac{\partial v_j}{\partial h_i} \right) = \sum_j^M \left( \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j} w'_{ij} \right)$$

where $\frac{\partial E}{\partial y_j}$ and $\frac{\partial y_j}{\partial v_j}$ have already been computed for the weights $w'_{ij}$. Thus the gradient can be written as

$$\frac{\partial E}{\partial w_{ki}} = \sum_j^M \left( \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j} w'_{ij} \right) \frac{\partial h_i}{\partial u_i} x_k \ \forall w_{ki} \in W \tag{4.11}$$

Using the gradients defined in equations 4.10 and 4.11 the update algorithm of each set of weights respectively is given by the SGD-algorithm (this is the simplest approach with a momentum parameter of 0):

$$w'_{ij} \leftarrow \quad w'_{ij} - \eta \frac{\partial E}{\partial w'_{ij}} = \quad w'_{ij} - \eta \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j} h_i \qquad \forall w'_{ij} \in W' \tag{4.12}$$

$$w_{ki} \leftarrow \quad w_{ki} - \eta \frac{\partial E}{\partial w_{ki}} = \quad w_{ki} - \eta \sum_j^M \left( \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j} w'_{ij} \right) \frac{\partial h_i}{\partial u_i} x_k \qquad \forall w_{ki} \in W \tag{4.13}$$

## 4.5 Regularization

Regularization is a method commonly used to prevent overfitting on the training data, which occurs when the model describes noise instead of the underlying relationship of the data. The desired effect of a model is a good generalization for all observed and unobserved data in the domain it predicts in. There exists many regularization approaches such as adding regularization terms to the loss function, ensemble methods and early stopping. Regularization terms such as L1 (Lasso) and L2 (Ridge Regression) adds a constraint function to the weights, this adds extra terms to optimize and gets quite costly as a neural network usually has many weights, thus this isn't always feasible or preferred. Ensemble methods trains a lot of models and averages the output and this is not efficient for deep neural networks as it is quite costly to train a neural network and it would be extremely costly to train enough models for a good ensemble. Early stopping depends on a validation set (which is not part of the training set) and stops training when accuracy (or another evaluation metric) on the validation set stops improving. Early stopping is more of an intuitive method and isn't very theoretically backed. Another regularization method is needed as common ensemble methods and regularization terms are to costly and inefficient when used for deep neural networks (further early stopping is poorly theoretically motivated and more than a little luck based).

### 4.5.1 Dropout

Dropout is a regularization method introduced by Srivastava et al. (2014) and tailored specifically for deep neural networks. It effectively approximate model combination, prevents overfitting and approximates exponentially many neural nets efficiently. The idea is to prevent the model from being too specialized on the training data (i.e. overfit) by at random removing (hidden and input) units from the model temporarily. The units are present with probability $p$ as illustrated in Figure 4.4. $p = 0.5$ seems to be close to the optimal value for a wide range of networks and tasks. The input nodes seems to have an optimal $p$ closer to 1 (a typical value is 0.8) (Srivastava et al. 2014).

Since it is not feasible to average the prediction of many thinned models at test time an approximate averaging method is used. The final prediction model is one neural net where the weights are scaled by the dropout probability $p$, as shown in Figure 4.5. This ensurers that the output at test time is the same as the expected output at training time. Dropout leads to significantly lower generalization error compared to other regularization methods on a wide variety of task including object classification, digit recognition, speech recognition, document classification and analysis of computational biology data (Srivastava et al. 2014).

A drawback of droput is that it increases training time ($p = 0.5$ results in a approximately 2-3 times longer training time than the basis model). This is mainly because the parameter updates becomes very noisy. However this stochasticity is likely the factor that prevents overfitting. This results in a trade-off between overfitting and training time (i.e. with more training time, one can use higher dropout and suffer less overfitting).

There exists many modifications of dropout both general and for specific tasks, such as DropConnect, DropPart, Standout and Maxout. **Dropconnect** proposed by Wan et al. (2013) is meant to be a generalization of dropout, however it only achieves better results through much more expensive training. And Smirnov, Timoshenko, and Andrianov (2014) show empirically that Dropout works better than DropConnect on the ImageNet[2] dataset, this is not a proof but more of an indication that Dropconnect might not outperform dropout even if the extra computational cost is within bounds. **DropPart** proposed by Tomczak (2013) is a further generalization of DropConnect. **Standout** proposed by Ba and Frey (2013) is meant to be a more adaptive dropout method. **Maxout** proposed by Goodfellow et al. (2013) is a deep learning model designed to exploit how optimization works with dropout.

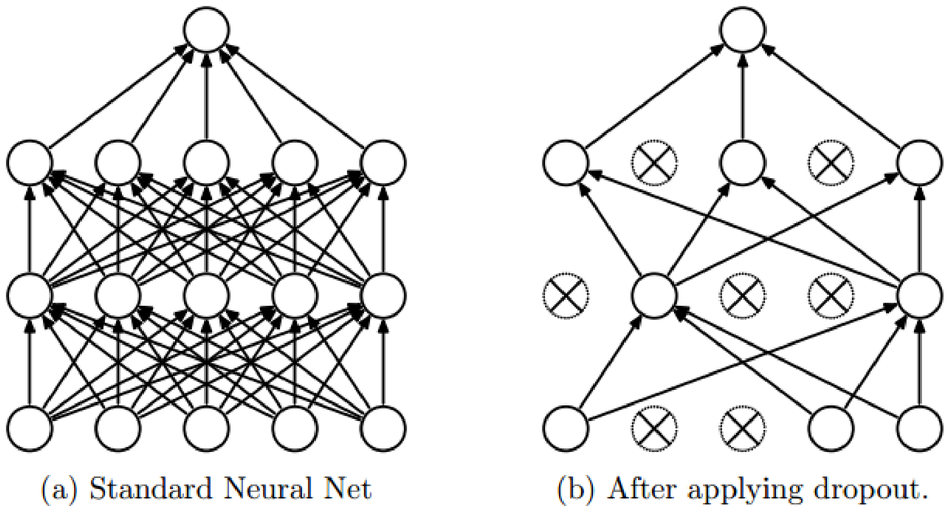


Figure 4.4: The Dropout Neural Network model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned network produced by applying dropout to the network on the left. Crossed units have been dropped. (Source: Srivastava et al. (2014))

## 4.6 Word vector models

This section explains how the word vectors introduced in section 3.2.4 are trained. Creating vector representations of words and phrases that retain semantic meanings requires appropriate models. The most popular models for training word vectors are context-counting and context-predicting model types.

[2] http://image-net.org/

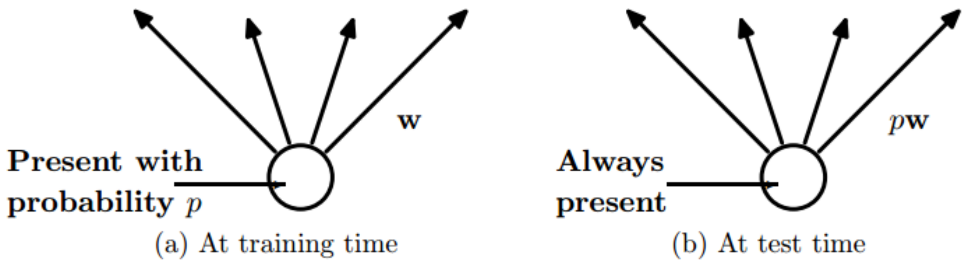


Figure 4.5: **Left:** At training time, the unit is present with probability $p$ and is connected to units in the next layer with weights **w**. **Right:** At test time, the unit is always present and the weights are scaled by a factor of $p$. The output at test time is then the same as the expected output at training time. (Source: Srivastava et al. (2014))

### 4.6.1 Count-based models - GloVe

Count-based models are trained by doing dimensionality reduction on a co-occurrence counts matrix. The co-occurrence count matrix $C$ (words × context) counts the co-occurrence of words and context. Context could for example be retrieved from document tags, for example a sports article or a deep learning paper. Since this matrix is extremely large one factorizes this matrix into (word × features) matrix $U$ and (context × features) matrix $V$. The relation can be represented as

$$C = UV^T. \tag{4.14}$$

These matrices $U$ and $V$ are trained by minimizing the "reconstruction loss", while trying to use low dimensional representations. The aim is to be able to explain most of the variance of the data given these matrices. The (word × features) matrix $U$ is used to represent the words, each row represents a word (or a class of words if one combines certain similar words like "is" and "are").

GloVe (Global Vectors for Word Representation) is a new and popular Count-based model out of Stanford. The model was introduced by Pennington, Socher, and Manning (2014) and is an unsupervised algorithm for training word vectors. Pre-trained GloVe vectors are available at the GloVe project site[3].

### 4.6.2 Predictive models - word2vec

Predictive models are trained by minimizing a loss function. The Skip-Gram model and the Continuous Bag-of-Words (CBOW) model are two popular predictive models from Google. These models are introduced and evaluated by Mikolov et al. (2013b),

[3] http://nlp.stanford.edu/projects/glove/

Mikolov et al. (2013a) and Mikolov, Yih, and Zweig (2013). These models are usually referred to as **word2vec** and pre-trained word vectors are available at the word2vec project site[4]. A more comprehensive and detailed literature is presented by Rong (2014). Both of these models are modelled by a simple neural network with one hidden layer and they are trained using back-propagation.

The Continuous Bag of Words (CBoW) model aims to predict a word given the context it is surrounded by. While the Skip-gram model tries to predict the context given the target word, it also creates more training cases by creating skip-grams of the word context (as shown in section 3.3.2). The skip-grams may create context examples of words that are far away from each other which together can give significant context information. A simple example of both models is illustrated in Figure 4.6, where $w(t)$ represents the target word and $w(t-2), w(t-1), w(t+1), w(t+2)$ represents the context of the word. This could represent the sentence:

*"Nobles dislike potato eating peasants."*

Which could be split up into the word set { "Nobles", "dislike", "potato", "eating", "peasants" } . Thus if the window size is 4 and the context is given as { "Nobles", "dislike", "eating", "peasants" } the target word is in this instance "potato".

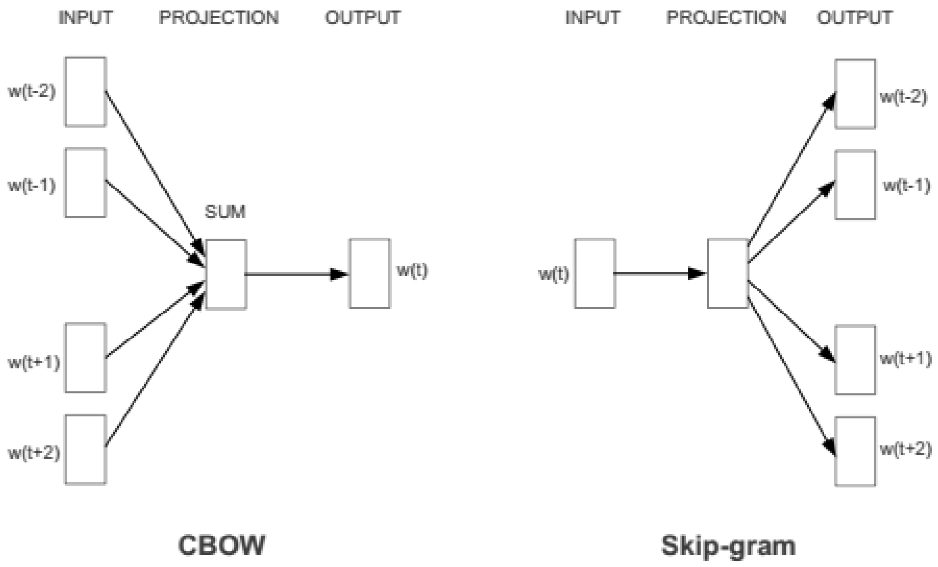


Figure 4.6: A simple example for comparison of the Continuous bag of words model and Skip-gram model. $w(t)$ represents the target word and $w(t-2), w(t-1), w(t+1), w(t+2)$ represents the word context. (Source: Mikolov et al. (2013b))

[4] https://code.google.com/archive/p/word2vec/

According to Mikolov[5] the Skip-gram model works well with small amounts of training data and represents even rare words and phrases well, while the CBoW model is much faster to train and has a slightly better accuracy for frequent words.

#### 4.6.2.1 Short on how the models are trained

This section will give a short description on how the Skip-gram and CBoW models are trained. These models are as mentioned trained with back-propagation (described in section 4.4).

Initial definitions:

- $C$ is the context size.
- $V$ is the size of the vocabulary used.
- $N$ is the number of features in the word vector representations.
- $J = \{1, 2, ... V\}$
- $I = \{1, 2, ..., C\}$
- Hot-encoded vectors: zero-vectors with the value 1 in the cell $j \in J$ which corresponds to the word in the vocabulary it represents.
- Weight matrix $W(V \times N)$
- Weight matrix $W'(N \times V)$

How the Continuous Bag of Word (CBoW) model is trained is illustrated in Figure 4.7. As mentioned, given the word context $x_1, x_2, ... x_C$ the model tries to predict which word the context surrounds. These vectors $x_i$ $(i \in I)$ are hot-encoded. The model can be simplified by summing the context vectors into a context vector $X$ $(X = \sum_i^C x_i)$. This context vector is connected to a weight matrix $W$ and through it the hidden layer $h$. The hidden layer is further connected to the weight matrix $W'$ and through it the output layer. The output layer computes a vector $y(1 \times V)$ and the highest value $y_j$ $(j \in J)$ corresponds to the word in the vocabulary the model predict that the context $X$ surrounds.

How the Skip-gram model is trained is illustrated in Figure 4.8. As mentioned, the only difference from the CBoW model is that the Skip-gram model tries to predict the context $y_1, y_2, ... y_C$ surrounding the word $x$. The model can be simplified by summing the context vectors into a context vector $Y$ $(Y = \sum_i^C Y_i)$. The hot-encoded

[5]Mikolov's post in a Google Groups thread:
https://groups.google.com/forum/#!msg/word2vec-toolkit/NLvYXU99cAM/E5ld8LcDxlAJ

vector $x$ is connected to a weight matrix $W$ and hidden layer $h$. The hidden layer is further connected to the weight matrix $W'$ and the output layer. The output layer computes a vector $y$, where the cells with the $C$ strongest activations refers to which words the model predicts that surrounds the word represented by $x$.

For both the models (after they have been trained) the rows in either $W$ or $(W')^T$ can be used to represent the words in the vocabulary as word vectors, it is usually $W$ that is used. Meaning that row $j \in J$ in $W$ represents the word vector for word $j$ in the vocabulary.

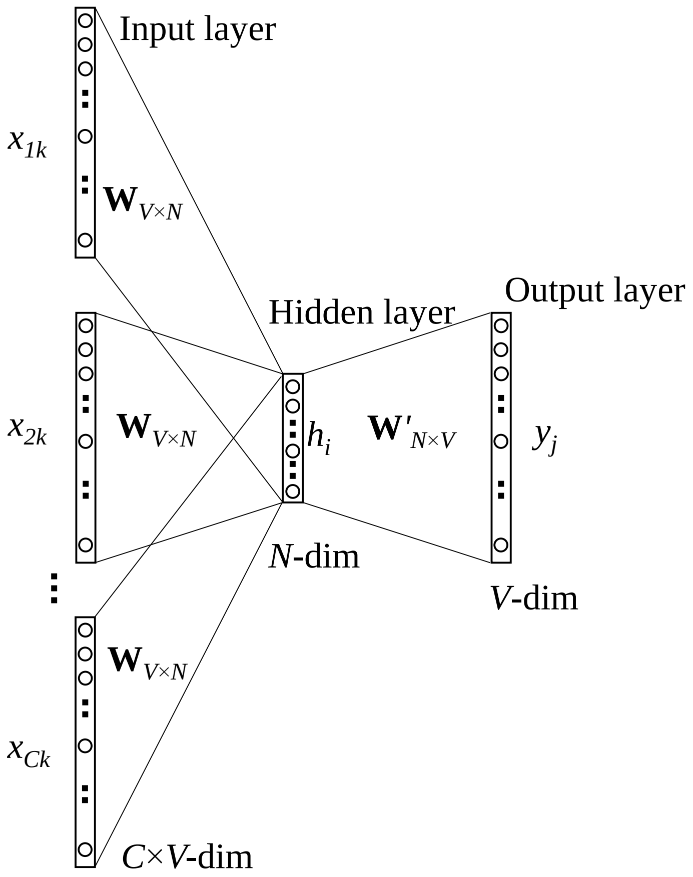


Figure 4.7: The Continuous bag-of-word model. (Source: Rong (2014))

## 4.6.3 Count-based models vs. Predictive models

Baroni, Dinu, and Kruszewski (2014) report that training word vectors using predictive (context-predicting) models outperform or perform as well as when using count-based (context-counting) models. While Pennington, Socher, and Manning

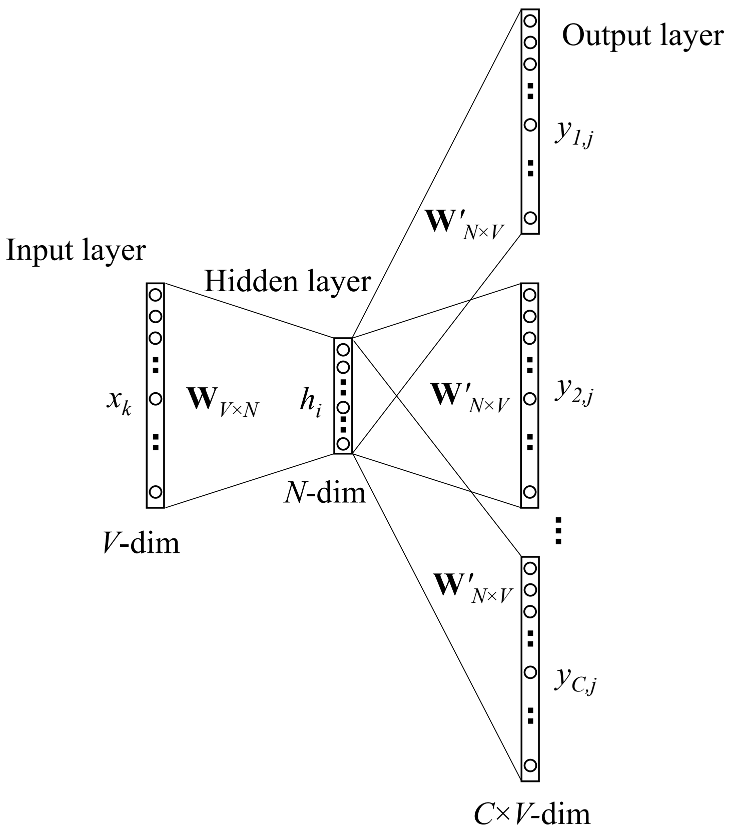


Figure 4.8: The skip-gram model. (Source: Rong (2014))

(2014) and Lebret and Collobert (2015) motivates that count-based models are more efficient to train, more easily parallelized and able to infer unseen words and phrases, which is an advantage over predictive models (which has to train representations of any new words if one wants to gain any relevant information from the word).

## 4.7 Convolutional Neural Network

The Convolutional Neural Network (CNN) is a neural network model inspired by how living creatures process natural image data. It is based on the work on the cat's visual cortex by Hubel and Wiesel (1968) which found that there exists cells that acts as local filters which search the natural images for patterns. These cells are ideal for exploiting the strong local correlation which is present in natural images. There exist many CNN models for image processing inspired by the visual cortex, a few of those are Fukushima (1980), Serre et al. (2007), LeCun et al. (1998) (LeNet-

5) and Krizhevsky, Sutskever, and Hinton (2012). Krizhevsky, Sutskever, and Hinton (2012) achieved "state-of-the-art" performance on the ImageNet[6] dataset using a CNN model, which further supports that the CNN model is well-suited for processing images that have a 2D structure with strong spatial correlation.

It has in recent years been motivated that the CNN is a viable model for NLP tasks, as there exists a 1D structure with strong local spatial correlation in natural languages. The effectiveness of CNN model on NLP tasks in comparison to "state-of-the-art" methods has been demonstrated by Johnson and Zhang (2014), Kim (2014), Kalchbrenner, Grefenstette, and Blunsom (2014), Shen et al. (2014) and Gao et al. (2015), implying that it is able to exploit the correlated 1D structure of text quite well. Johnson and Zhang (2014) used a CNN model with hot-encoded input to train domain specific word vectors in the same model which did the main task of classifying documents. Kim (2014) used the publicly available **word2vec** word vectors to encode the input of their CNN variants. Kalchbrenner, Grefenstette, and Blunsom (2014) introduced the Dynamic CNN which use dynamic $k$-max pooling. Further Shen et al. (2014) presents the Convolutional Latent Semantic Model (CLSM) and Gao et al. (2015) presents the Deep Semantic Similarity Model (DSSM), both models learn semantic representations of sentences for Information Retrieval (these models were trained to recommend documents to users based on what they are currently reading or have been reading).

### 4.7.1 The model

The CNN model consists of an input layer, a convolution layer, a pooling layer, fully connected nodes and a final prediction layer. It can consist of multiple convolution and pooling layers, but these are usually used in order (i.e. a convolution layer is always preceded by an input layer or pooling layer, while a pooling layer is always preceded by a convolution layer). Deeper CNN models with multiple convolution and pooling layers are usually reserved for extremely large datasets. In the text case the input is given as word sequences (defined in Section 3.2.3) where each word is encoded as a word vector. The word vectors can be static or be trained with the rest of the model. An example of a CNN model with text input is illustrated in Figure 4.9. The CNN model is like the other neural networks trained using back-propagation.

#### 4.7.1.1 Convolution

Each convolution node got one unique filter, which searches the input for a unique pattern. The convolution nodes computes a vector (a matrix in the image case) of the length equal to possible placements of the filter, each cell refers to how strongly the pattern was observed in a the unique location connected to that cell. A simple

[6] http://image-net.org/

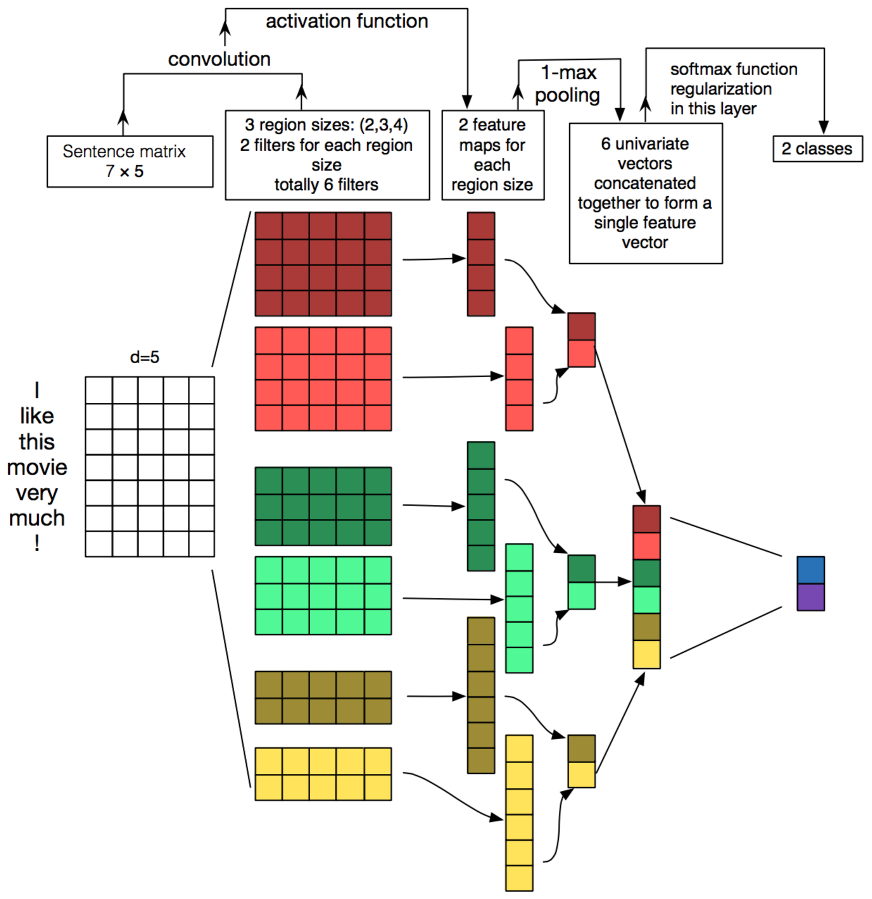

activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Figure 4.9: A graphical depiction of a CNN model using 1-max-pooling with text input. (Source: Zhang and Wallace (2015))

convolution on an image matrix is illustrated in figure 4.10, in the example the matrix represents a black and white image, where black is represented by 0 and white by 1. The $3 \times 3$ yellow window is a filter which slides over the image. In this example the filter multiplies its values element-wise with the part of the image it covers, and sends that value to the "Convolved Feature", which is the matrix it will send to the convolution node it is connected to. The step-wise convolution update given the filters iteration over the image is shown in figure 4.11. In the common text case the height of the filters is set to be the same length as the number of features in the word vectors. While the filters width, called the window size, which decides how many words fits in the filter is a hyper-parameter chosen through tuning (eg. line-search).

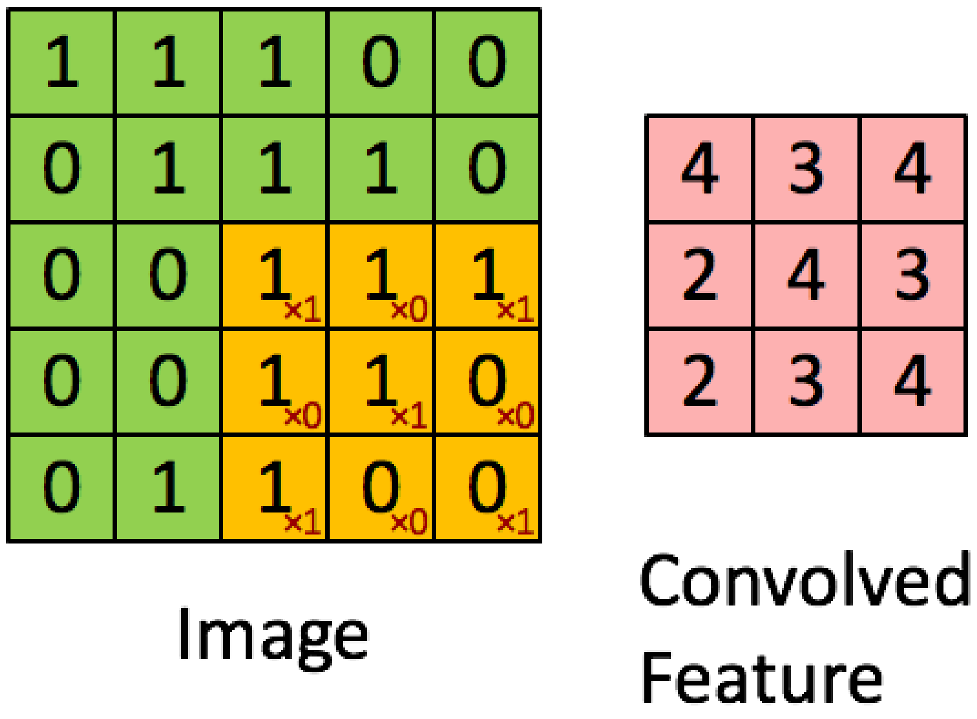
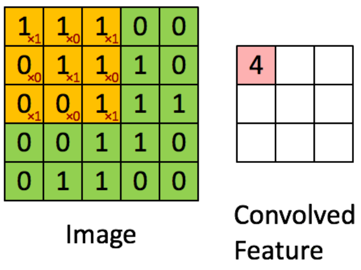


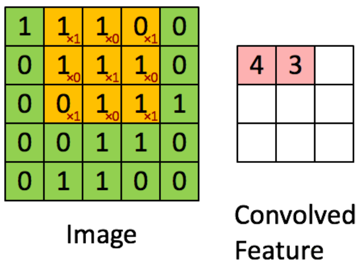
Figure 4.10: **Simple 2D convolutional layer example:** The matrix represents a black and white image, where black is represented by 0 and white by 1. The $3 \times 3$ yellow window is a filter which slides over the image. In this example the filter multiplies its values element-wise with the part of the image it covers, and sends that sum to the "Convolved Feature", which is the matrix it will send to the convolution node it is connected to. The step-wise convolution update given the filters iteration over the image is shown in Figure 4.11. (Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution#Convolutions)

#### 4.7.1.2 Pooling

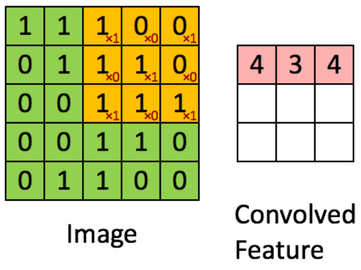
Each convolution node is connected to a unique pooling node. These pooling nodes looks for the most significant information retrieved by the convolution nodes by following different pooling schemes. The simplest pooling scheme is 1-max-pooling, which simply means that the pooling node retrieves the highest value from the filter. Pooling reduces dimensionality while retaining the most important information, this makes the model more computationally effective compared to just using convolution layers and fully connected layers. A max-pooling example is shown in Figure 4.12, where each colored area refers to a separate convolution node. The input matrix consists of 4 convolution nodes which are processed by 4 pooling nodes using



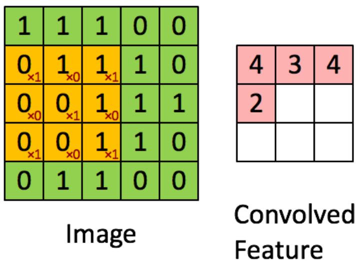
(a) Step 1



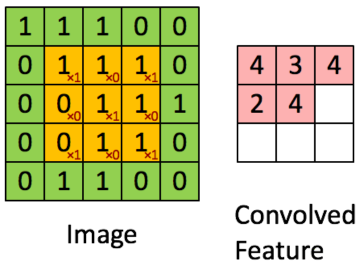
(b) Step 2



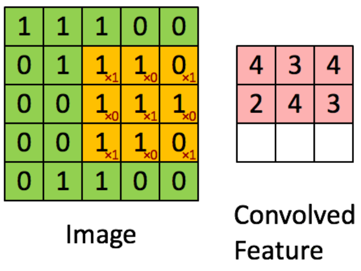
(c) Step 3



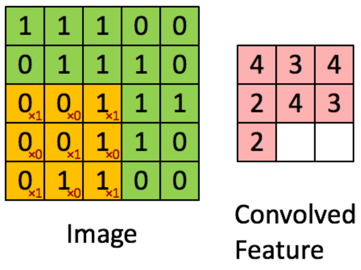
(d) Step 4



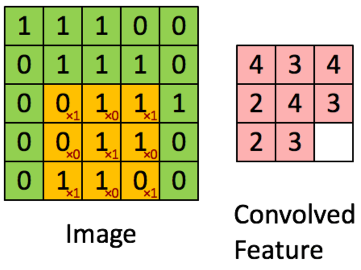
(e) Step 5



(f) Step 6



(g) Step 7



(h) Step 8

Figure 4.11: The stepwise convolution update given the filters iteration over the image is shown in this figure, see image 4.10. (Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution#Convolutions)

max-pooling.

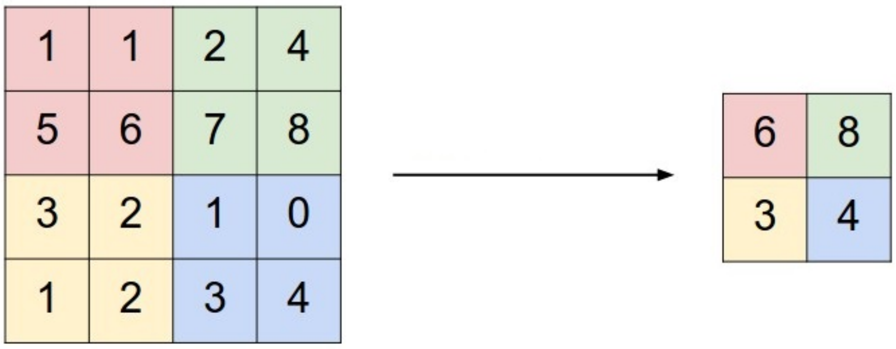


Figure 4.12: A pooling example where each colored area refers to a separate convolution node. The input matrix consists of 4 convolution nodes which are processed by 4 pooling nodes using max-pooling. (Source: `http://cs231n.github.io/convolutional-networks/#pool`)

## 4.8 Recurrent Neural Network

The Recurrent Neural Network (RNN) is a neural network which isn't an acyclic graph like the previous neural networks. The RNN uses directed cycles to model temporal behaviour. Mikolov et al. (2010), Mikolov et al. (2011), Yao et al. (2013) and Mesnil et al. (2014) have gotten good results using RNN-LM (Recurrent Neural Network Language Models) models on NLP tasks. Chung et al. (2014) motivates that the LSTM and GRU variants of the RNN perform best.

### 4.8.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is an old and popular RNN variant introduced by Hochreiter and Schmidhuber (1997). LSTM doesn't have the vanishing gradient problem and have shown good results on NLP tasks[7].

[7]Gers and Schmidhuber 2001.

### 4.8.2 Gated Recurrent Unit

The Gated Recurrent Unit (GRU) is a new RNN variant introduced by Cho et al. (2014). The GRU automatically learns the grammatical structure of a sentence.

# 5 Experiments

This chapter presents the results obtained from the analysis of the complete IMDB dataset and the Hewlett Foundations Automated Essay Scoring (AES) dataset. The IMDB dataset is big relative to the essay dataset (labelled training data 25.000 documents, test data 25.000 documents and unlabelled training data 50.000 documents), but not very complex (as reviews are either positive or negative). While the essay dataset is quite small in comparison (8 essay sets of around 1800 essays each), but very complex due to its nature. These datasets are both ideal for testing text analysis models.

Note that there are restrictions concerning memory and available disk space with respect to personal hardware, university hardware and the python packages used for modelling the data. A Tesla C2050[1] GPU is used for parallel training the neural networks to keep the training time within practical bounds. The disk space is a concern on the server with the GPU, which makes the use of domain word vectors with more than 300 features infeasible. This is a pity since word vectors of size 5000 have been shown to give good results on the IMDB dataset[2,3].

## 5.1 General information about the experiments

This section contains general information about the experiments such as information about the features used, models used, training time and general observations.

### 5.1.1 Features

The features used in the different experiments, as well as their abbreviations, are listed and described here.

- Word vectors used (defined in Section 3.2.4):
  - w2v: Googles pre-trained word vectors[4] with 300 features trained on part of the Google News dataset containing about 100 billion words. The full w2v vocabulary contains about 3 million words and phrases.

[1]http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_C2050_C2070_jul10_lores.pdf
[2]https://github.com/smartinsightsfromdata/kaggle-word2vec-movie-reviews
[3]https://github.com/smartinsightsfromdata/kaggle-sentiment-popcorn
[4]https://code.google.com/archive/p/word2vec/

  - domain: Domain word vectors with 300 or 5000 features trained on the IMDB labelled training, unlabelled training and test data.
  - Abbreviations: w2v (w2v-300), domain-300 and domain-5000 denotes the different word vector representations.
  - Note: The word vectors gave very good performance if the dataset was big enough for them to be effective.

- Bag of Words (defined in Section 3.2.2.1)
  - Ex: BoW500 is a Bag of Words representation containing the top 500 most frequent words.
  - Ex: BoW-max is a Bag of Words representation containing all the words in the vocabulary.
  - Note: The Bag of Words representation gave a good baseline and performed very well compared to the other features even if the dataset was relatively big. And in the small case of the AES dataset it was the most important feature.

- tf-idf (defined in Section 3.2.2.2)
  - Ex: tfidf(1,2) is a tf-idf representations of the vocabulary extended by 2-grams (i.e. from range 1 to 2, bi-grams) (defined in section 3.3.1).
  - Ex: tfidf500(1,3) is a tf-idf representation of the vocabulary extended by 3-grams containing the 500 highest weights.
  - Note: The feature was the most expressive feature of the non-sequential representations of the data in the IMDB case. And in the AES case it gave a significant boost to the Bag of Words representation, but did not perform exceptionally well alone.

- Bag of Clusters: Word clusters where words that are closer to each other according to the word vectors (w2v or domain) are in the same cluster (defined in Section 3.2.5.2).
  - Ex: BoC(domain) is a cluster representation using the domain vectors (with 500 features/clusters).

- Document vectors (d2v): the average of all the word vectors in a document (defined in section 3.2.5.1).
  - Ex: d2v(w2v) is a document vector created using the w2v word vectors with 300 features.
  - Note: A d2v representation without the word vector type defined uses w2v word vectors.

- Negative and Positive words:
  - The Count of Positive words and the Count of Negative words:
    - Ex: BoW500NP is a BoW500 representation combined with the count of positive words and the count of negative words.
  - Positive and Negative word ratio in the document:
    - Ex: BoW500NP-ratio is a BoW500 representation combined with the negative-positive word ratio.
- Number of words in the document
  - Ex: BoW300-nbWords is a BoW300 representation combined with the number of words in each document.
- Number of sentences in the document
  - Ex: BoW300-nbSent is an BoW300 representation combined with the number of sentences in each document.
- Average sentence length in a document.
  - Ex: BoW300-avSent is an BoW300 representation combined with the average sentence length in each document.
- Bag of Word Classes: A vector containing the number or ratio that each word class appears (e.g. the number of nouns, verbs, etc...)
  - Ex: BoW500-WC is a BoW500 representation combined with the count that each word class appears.
  - Ex: BoW500-WR is a BoW500 representation combined with the ratio that each word class appears.
- Number of stop words in the document. Stop words are high-frequency words such as "to" and "from".
  - Ex: BoW300-Stop is a BoW300 representation combined with a the number of stop words in each document.

The importance of each feature is described in detail in the experiment sections they are used in (IMDB section 5.2.5 and AES section 5.3.2). An interesting note is that the BoW and tf-idf representations are only able to gain information from the training data, while the models trained on word vectors are also able to gain information from the vocabulary of the test data, this makes them more flexible to new observations.

The list of words used to classify a word as positive or negative is available on the net[5]. The list was developed as part of the papers Hu and Liu (2004) and Liu, Hu, and Cheng (2005). The word classes were extracted by using the NLTK (Natural Language Toolkit[6] which is available as a python package. The NLTK package was also used to determine the list of stop words.

### 5.1.2 Models used

The models used in the experiments, as well as their abbreviations, are listed here:

- Random Forest Classifier
  - Ex: RF100 is a random forest classifier with 100 trees.
- Naive-Bayes Classifier
  - Short: NB
- Logistic Regression
  - Short: LR or LogReg
- Linear Regression
  - Short: LinR or LinReg
- Linear Support Vector Machine Classifier
  - Short: SVM
  - Note: This model performed best of the common statistical methods tested on the classification problem (IMDB).
- Support Vector Regression
  - Short: SVR
  - Note: This model performed best of the common statistical methods tested on the regression problem (AES).
- Multilayer Perceptron (MLP)
  - Ex: MLP-2-2 is a MLP with 1 hidden layer consisting of 2 nodes and a final softmax layer with 2 nodes.

[5]http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html
[6]http://www.nltk.org/

Table 5.1: Models and their approximated training time

| Models | Training time CPU | Training time GPU |
|---|---|---|
| RF100, NB, LogR, LinR, SVM, SVR | 1-5 min | - |
| MLP, RF1000 | 5-60 min | 1-15 min |
| CNN | 1-5 hour | 20-30 min |

  - Ex: MLP-512-32-2 is a MLP with one hidden layer of 512 nodes connected to another hidden layer of 32 nodes, which is connected to a softmax layer of 2 nodes.
  - Ex: MLP-2-1 is a MLP with 1 hidden layer with 2 nodes and a final linear regression layer computing one value (1 final node).

- Convolutional Neural Network (CNN)
  - Ex: CNN-300f-8d is a CNN with 300 filters and a window size of 8 cells/words.

#### 5.1.2.1 MLP

The MLP's were trained with dropout $p_0 = 0.8$ in the initial feature layer and $p = 0.5$ in any additional layers (except for the final layer). The final layer was a softmax layer for the IMDB dataset and a linear regression layer for the AES dataset. Using SVM and SVR (currently unavailable for the Keras framework used) for the final layers could increase performance.

#### 5.1.2.2 CNN

The CNN was trained with Relu activations and without any dropout as it seemed to interfere with the Relu activations. As with the MLP's final layer; the final layer was a softmax layer for the IMDB dataset and a linear regression layer for the AES dataset. Using SVM and SVR (currently unavailable for the Keras framework used) for the final layers could increase performance.

### 5.1.3 Training time

Approximated training times of the models used in the experiment section is shown in Table 5.1.

### 5.1.4 Observations

This section contains a few general observations done through the experiments.

#### 5.1.4.1 CNN vs. dropout

An interesting observation is that the rectifier performed better without dropout in the CNN case. This effect could be due to the fact that the rectifier and the dropout method both impose sparseness and combined they could introduce too much sparseness in the model, leading to increased model bias.

#### 5.1.4.2 Removing Stop words

Removing stop words is a common practice when pre-processing a text dataset, but removing them from the documents gave poorer results for both datasets. For the IMDB dataset this is probably because some stop words, such as "against" and "into", can contain semantic meaning (i.e. positive/negative meaning). In the essay set case it might be because they are needed to observe good sentence structures.

#### 5.1.4.3 Vocabulary size

The performance of models trained on Bag of Words and tf-idf representations seems to be a convex function of the vocabulary size. This function does not appear to be strictly increasing, meaning that too big a vocabulary could hurt the models by causing overfitting.

## 5.2 IMDB Full Data set

The results obtained from the analysis of the full IMDB dataset, described earlier in Section 2.2, will be presented here. The dataset was released as part of a competition where the "prize" was learning. The goal of the competition was to get the best predictions on the test data given the Area Under the Receiver Operating Characteristic curve (AUROC) evaluation metric (defined in 5.2.2). The scores of the competition can be found at the competitions leaderboard[7].

The AUROC metric, to be defined in Section 5.2.2, is in these experiments computed by submitting the predictions generated by the models to the Kaggle competition page[8] (note that the competition is over so any new submissions will not be recorded, but the corresponding AUROC value will still be computed for anyone wanting to evaluate their models). The accuracy is computed by rounding the predictions to binary values and submitting them (since the AUROC value of binary predictions is equal to the accuracy).

[7] https://www.kaggle.com/c/word2vec-nlp-tutorial/leaderboard

[8] https://www.kaggle.com/c/word2vec-nlp-tutorial/submissions/attach

### 5.2.1 Classification task

The task of predicting a review's positive or negative sentiment is a standard binary classification problem, however it is the computed probabilities that are most interesting when computing the AUROC value. In some cases a model with the higher AUROC value might not have the highest accuracy, which could be something noteworthy, but the values are usually highly correlated.

### 5.2.2 Evaluation Metric - Area Under the Receiver Operating Characteristic curve (AUROC)

The submissions at Kaggle are evaluated using the Area Under the Receiver Operating Characteristic curve (AUROC) metric, also referred to as the Area Under the Curve (AUC), although the latter name is a little ambiguous. This evaluation statistic is nice as it supports not just binary class predictions but also probability predictions. Probability predictions actually always get a higher or the same score as the rounded binary predictions.

The ROC curve (which the AUROC is the sum of the area under) plots the False positive rate ($FPR$) against the True positive rate ($TPR$), which both are defined as

$$TPR = \frac{TP}{TP + FN}$$
$$FPR = \frac{FP}{FP + TN}$$

where $TP$, $FP$, $TN$, $FN$ are the number of True positives, False positives, True negatives and False negatives. The $TPR$ is plotted against $FPR$ at many different thresholds (for example 0.00, 0.01, 0.02, ..., 1.00) which decides when a prediction is assumed to be true (e.g. at a threshold of 0.90 a prediction is assumed to be true if the computed probability is equal or higher than 90% and any prediction with a lower probability is assumed to be untrue). A random prediction will result in an AUROC value close to 0.5, which is usually used as a threshold. And if all the predictions are wrong the AUROC value is 0 and if they are all correct the value is 1. In the case of binary predictions the AUROC value is the same as the accuracy.

### 5.2.3 Validation set

The neural networks are validated during training by a valiation set which is created by splitting the training data into a 80-20 training/validation-set

### 5.2.4 Vocabulary

After pre-processing the training dataset contains a vocabulary of 74436 words. While the training and test sets combined contains of a vocabulary of 112540 words.

### 5.2.5 Features

The features used in the IMDB dataset experiments are listed here:

- Word vectors: w2v, domain-300, domain-5000
  - Feature importance: In the IMDB experiments the word vectors were the most important features as they were used as word encodings for the CNN model which gave the best performance of all the non-ensemble methods tested.
    - Note: For the CNN the w2v word vectors were used, although there were indications that the domain word vectors also would perform on the same level, if not significantly better in the case of the domain-5000 word vectors.
- Bag of Words (BoW)
  - Feature importance: The Bag of Word representations gave a good baseline for the experiments and performed relatively well compared to the other representations in the IMDB experiments.
- tf-idf
  - Feature importance: The tf-idf representation was the most important feature for the non-sequential models in the IMDB experiments.
- Bag of Clusters (BoC)
  - Feature importance: With the tested number of clusters (500) the representation did not give very good performance and was deemed unimportant. Increasing the cluster size might however increase the performance/importance. It could also have a potential for boosting other feature representations.
- Document vectors (d2v)
  - Feature importance: The document vectors tested did not give very good results alone, it could have a potential for boosting other representations. Increasing the feature size of the d2v representations might also improve the performance/importance of the feature.

- Count of Positive words and count of Negative words (NP) and Negative-positive word ratio (NP-ratio)
  - Feature importance: The NP and NP-ratio features gave little gain and were deemed insignificant.

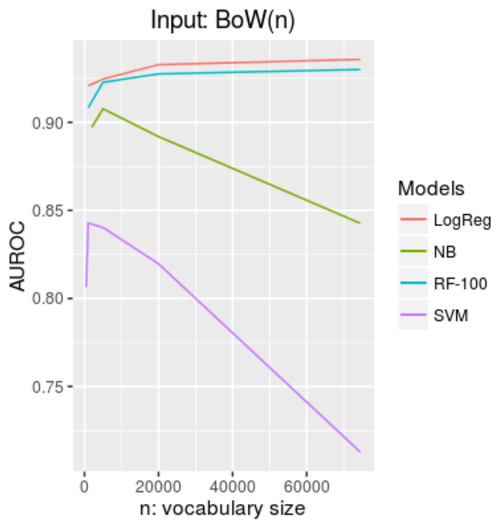
The positive and negative gain from using the document vectors (d2v) and NP/NP-ratio features to boost the other representations seemed to vary stochasticly around (or if not slightly below) 0, implying that they only introduced bias to the modelling and had no real benefit.

### 5.2.6 BoW

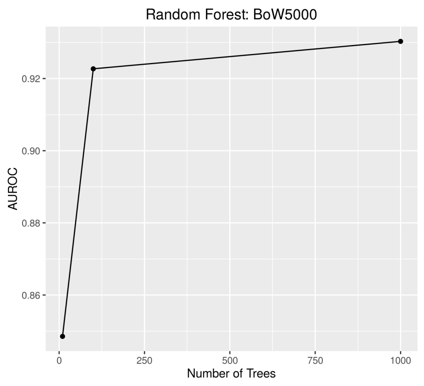
The performance of the models trained with the BoW representation are show in Figure 5.1. The Logistic Regression model had the best performance on the BoW representation of the IMDB data as shown in figure 5.1a. The Random Forest model performed very close to the Logistic Regression with 100 trees, but increasing the trees to a size of 1000 was too memory costly so it was only tested on a vocabulary size of 5000 (at which it did not outperform the Logistic Regression). Results from testing with 10, 100 and 1000 trees can be seen in figure 5.1b (the RF trains extremely inefficiently as every tree refers to a separate model). The Naive-Bayes performed moderately while the SVM performed the worst on the BoW representation. The Logistic Regressions performance boosted by the NP and/or NP-ratio features is shown in figure 5.1c, which shows that the extra features boost the model a little bit. And the best performance of all the classifiers and the boosted Logistic Regression model is shown in figure 5.1d.

### 5.2.7 tf-idf

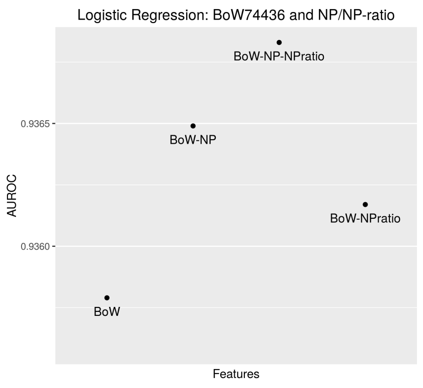
The models performance on the tf-idf representation of the IMDB data is visualized in figure 5.2. The SVM outperformed the Logistic Regression as can be seen in Figure 5.2a, which shows the models trained on different tf-idf representations based on different N-grams of size *g*. The best results seemed to be with N-grams of range 1 to 2. And it seemed uniform that all the models perform best when the full vocabulary was used and no words were removed. Neither the SVM nor the Logistic Regression did well when combining the NP and NP-ratio features to the tf-idf representations, the performance of the boosted Logistic Regression is shown in figure 5.2b (the SVM didn't do any better). The Naive-Bayes and Random Forest didn't perform significantly well on the tf-idf representation compared to the Logistic Regression and the SVM, and their performance are only shown in figure 5.2c which shows the best performance of each classifier.



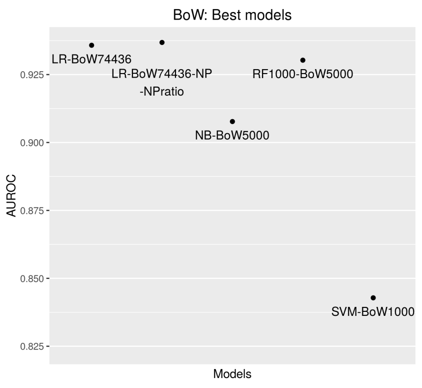
(a) The performance of the models plotted against the vocabulary size of the BoW representation.



(b) The performance of the RF plotted against the number of trees used in the ensemble.

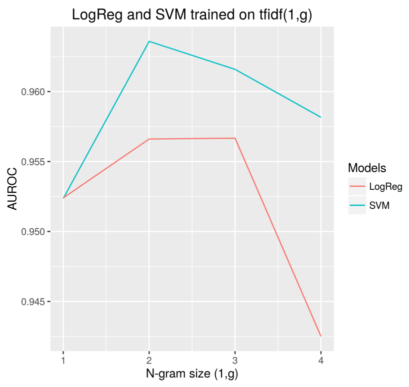


(c) The Logistic Regression trained on BoW combined with the count of Negative and Positive words and the ratio.

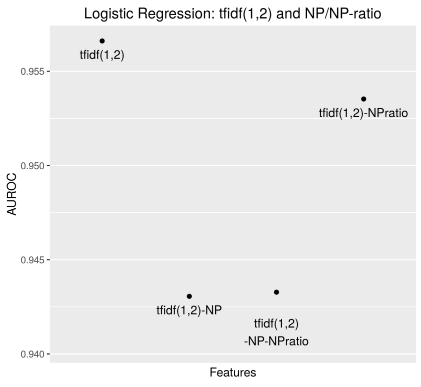


(d) The best performance of the different classifiers on the BoW representation and the boosted LogReg.

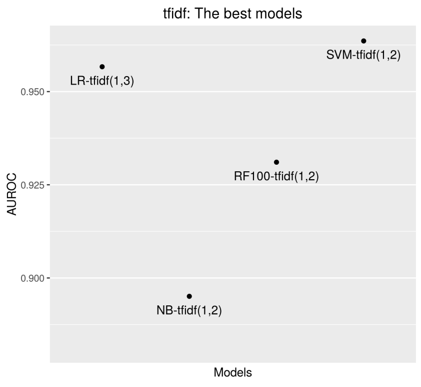
Figure 5.1: The plots showing the performance of the NB, RF, SVM and LogReg Classifiers on the BoW representation of the IMDB dataset.



(a) The performance of the Logistic Regression and SVM models plotted against the N-gram size used to create features for the tf-idf representation.



(b) The Logistic Regression boosted with the NP and NP-ratio features.



(c) The best performance of each classifier tested on the tf-idf representation.

Figure 5.2: The plots showing the performance of the NB, RF, SVM and LogReg Classifiers on the tf-idf representation of the IMDB dataset.

### 5.2.8 w2v vs.domain specific

In this experiment section the performance of Googles pre-trained w2v word vectors and domain specific word vectors are compared. The word vectors used in these experiments are listed here:

- w2v-300
- domain-300
- domain-5000

Googles w2v word vectors contains 300 features and were pre-trained on part of the Google News dataset (containing about 100 billion words). The full w2v vocabulary contains about 3 million words and phrases. While the domain-300 and domain-5000 word vectors were trained on the reviews of the IMDB dataset. The vectors were trained using both the available labelled and unlabelled (training and test) reviews. These domain specific word vectors were trained using the skip-gram model (as introduced in section 4.6.2) in the Word2vec function, which is a part of the **gensim** python package. The number of features were set to be 300 (the same number that Google's pre-trained word vectors have). Words occurring less than 40 times were ignored and a context size of 10 was used for training. Another set of word vectors with 5000 features (with the other hyper-parameters held constant) were also trained, but the size of the created data matrix was too big for training in reasonable time with respect to neural networks. These domain word vectors were created with a vocabulary of the 16490 most frequent words, to make testing easy. The w2v vocabulary was also reduced to the same vocabulary as the domain vectors in this section so that the word vectors performance could be compared. These word vectors were tested on BoC, d2v and sequential representations of the IMDB dataset. The results of analysis of Googles pre-trained w2v word vectors and domain specific trained word vectors are shown in figure 5.3.

The Bag of Clusters (BoC) representation was created by finding the 500 most representative clusters based on the word vectors and the document vector (d2v) representation were created by computing the mean over all the word vectors in each document. Figure 5.3a and 5.3b shows the performance of the Logistic Regression Classifier on the Bag of Cluster representations and the Document vector (d2v) representations of the IMDB data based on w2v and domain vectors. The results show that the domain vectors with 5000 features outperforms both the w2v vectors and the domain-300 vectors. The BoC and d2v results are poor, but they imply that the word vectors indeed have benefit and also that BoC and d2v representations with more features could give interesting results. BoC and d2v representations could also have potential to boost other representations. This analysis was constrained by hardware capabilities. Training clusters of a greater size than 500 demanded memory usage which exceeded the available in memory and was thus not done.

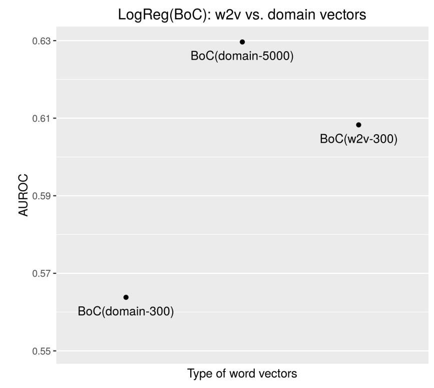
A CNN model with 300 filters (convolution nodes) and a window size of 8 was trained on the w2v-300 and domain-300 word representations. Figure 5.3c shows that the CNN (experiment specifics defined in Section 5.2.10) performed best on the domain-300 vectors. The domain vectors with 5000 features couldn't be used as input to the CNN as this would be too demanding on both memory and available disk space connected to the GPU.

The w2v (w2v-300) and domain-300 vectors didn't give significantly different results, this implies that the w2v vectors are good generalizations which works well globally (atleast in the case of the IMDB dataset). As a result the w2v vectors will be used with the full vocabulary on the rest of the experiments.

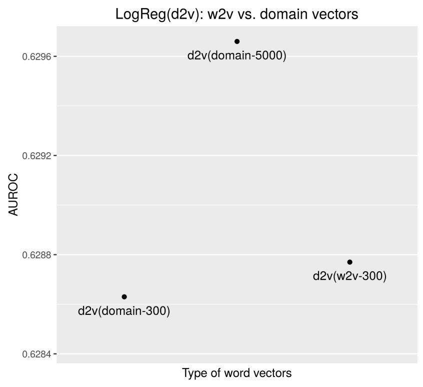
### 5.2.9 MLP

In this section MLP models trained on representations of the IMDB dataset are experimented on. The MLP's are trained with an initial dropout of 0.8 for the input layer and a dropout of 0.5 for any additional layers (except for the final classification layer). The Relu activations were used in all the layers, since it outperformed the hyperbolic tangent in the IMDB MLP experiments. The Relu was not used in the final layer, here the softmax was used to calculate the probability distribution of the two classes $\{0,1\}$. The experiments are illustrated in Figure 5.4 and Figure 5.5. In the figures the MLP's performance is plotted against the number of nodes $n$ of a specific layer, for readability the binary logarithm of $n$, $k = \log_2(n)$, is used to scale the the $x$-axis. Further $k = -1$ is defined to mean $n = 0$ (i.e. that the hidden layer is removed from the model).

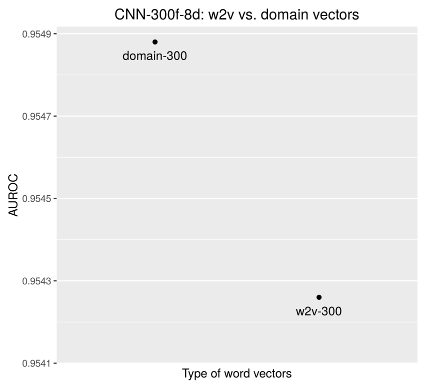
The MLP experiments on BoW representations of the IMDB dataset are illustrated in figure 5.4. Figure 5.4a shows the MLP-2 model plotted against the BoW of varying vocabulary sizes. It shows that a vocabulary of the 5000 most frequent words is optimal for a MLP-2 model. Figure 5.4b shows the MLP-$n$-2 model plotted against varying number of nodes at different BoW vocabulary sizes. It shows that the best model is a MLP-$n$-2 with $n = 2^{-1} = 0$ (which in this case as mentioned above is defined as 0) trained on the BoW5000 representation. Meaning the MLP-2-BoW5000 is the best MLP model on the BoW representation. Figure 5.4c shows the MLP-2 being boosted by the NP/NP-ratio features, it shows that the BoW5000 is slightly boosted by only adding the NP-ratio features. This is notable, since the selling point of the MLP and other Deep Learning models is that one shouldn't have to do feature selection and just feed all the available data into the model (as long as one has enough computing power). The MLP is supposed to do the feature selection for you, this performance could be due to the data set still being relatively "small" compared to the data sizes which Deep Learning models are motivated for. Further the MLP(BoW) became too complex when using hidden layers and had problems doing good generalizations, this could also be due to the lack of data/observations.



(a) The performance of the Logistic Regression on Bag of Clusters representations with 500 cluster/features based on w2v and domain vectors.



(b) The performance of Logistic Regression trained on d2v representations based on w2v and domain vectors.



(c) The performance a CNN model with 300 filters and a window size of 8 cells/words trained on sequential data represented by domain and w2v vectors.

Figure 5.3: The plots shows the performance of Googles w2v word vectors vs. domain specific trained word vectors in different input representations.

The best generalizations were given by MLP's without any hidden layers as the results show.

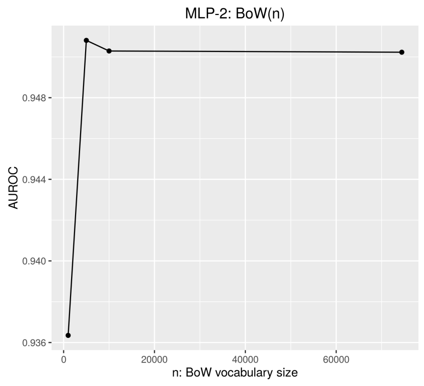
The MLP experiments on tf-idf representations of the IMDB dataset are illustrated in figure 5.5. The MLP-$n$-2-tf-idf($C$)(1,2) was trained against varying number of nodes $n$ at different vocabulary sizes $C$, these experiments are illustrated in Figure 5.5a. The optimal size of the vocabulary was found to be around the 60.000 highest weighted words and the optimal number of hidden nodes $n$ converges to 2 as the vocabulary size increases (i.e. the MLP-2-2-tfidf-60k(1,2) model gave the best performance). The MLP-2-2-tf-idf60k(1,$g$) model was trained at varying N-gram sizes $g$ , the results are illustrated in Figure 5.5b. The optimal N-gram size was found to be $g = 2$. Boosting the model MLP-2-2-tfidf(1,2)-60k with a BoW5000 representation and/or NP/NP-ratio features was attempted, but it did not improve the performance, these findings are illustrated in Figure 5.5c.

The best MLP trained on the tf-idf representation outperformed the MLP trained on the Bag of Words representation. This best MLP model outperformed the non-neural net models on the non-sequential data which is illustrated in Figure 5.7b.

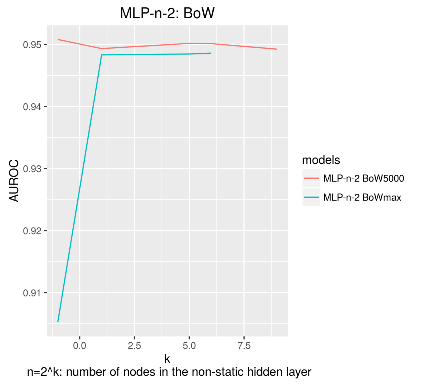
### 5.2.10 CNN

In this experiment section CNN models were trained on a sequential representation of the IMDB dataset. The CNN's were modelled with a convolutional layer connected to a max-pooling layer which was connected to a final softmax layer. The model was trained on sequential data represented by w2v word vectors. The CNN contained a number of filters $f$ which looks for 2D-patterns in the documents which is represented as a matrix (words × word-features). The 2D window searched through was of size $d \times 300$ (where 300 is the number of features each word is represented by and not to be confused with the number of filters). For the CNN model relu activations were used as it outperformed hyperbolic tangent activations and no dropout was used since it degraded the performance. The CNN was trained with a varying numbers of filters $f$ and varying windows sizes $d$, these experiments are illustrated in Figure 5.6. The optimal numbers of filters $f$ was found to be 300, while the optimal window size $d$ was found to be 8. These values were as high as the memory restrictions allowed, thus better results could be found with more filters and a wider window size.

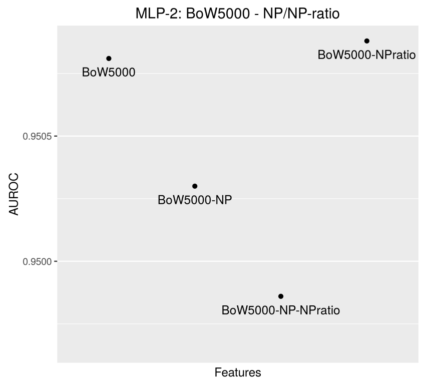
The CNN gives good results compared to the models trained on non-sequential representations of the data, as can be seen in Figure 5.7b which compares the models.



(a) The MLP-2 model is plotted against the BoW with varying vocabulary size.

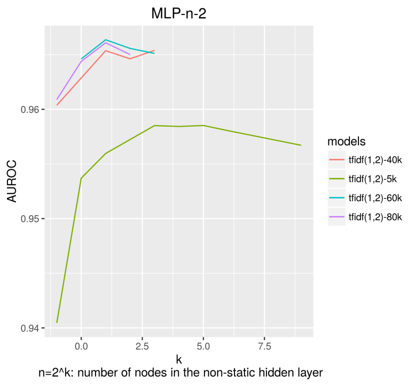


(b) The MLP-n-2-BoW model with varying number of hidden nodes $n$ is plotted with BoW vocabulary sizes 5000 and 74436(max).

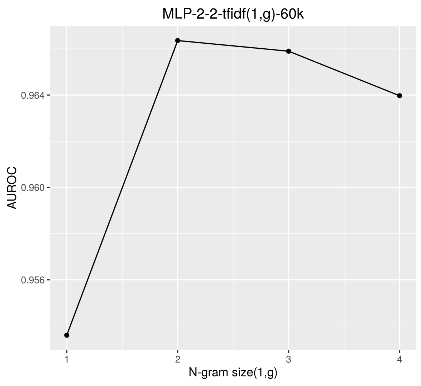


(c) The MLP-2-BoW5000 boosted with NP and/or NP-ratio features.

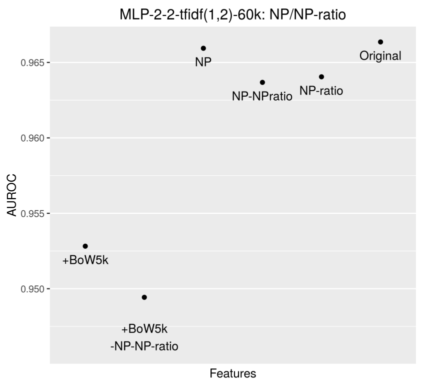
Figure 5.4: The performance of the MLP on the BoW representation of the IMDB dataset is illustrated in this figure.



(a) This figure illustrates the performance of MLP-n-2 trained on specific vocabulary sizes of tf-idf(1,2) plotted against the number of hidden nodes $n$.



(b) This figure plots the MLP-2-2-tfidf$(1,g)$-60k against the N-gram size $g$.



(c) This figure shows the best MLP model trained on the tf-idf representation boosted with BoW5000, NP and/or NP-ratio features.

Figure 5.5: An illustration of the performance of the MLP on the tf-idf representation of the IMDB dataset.

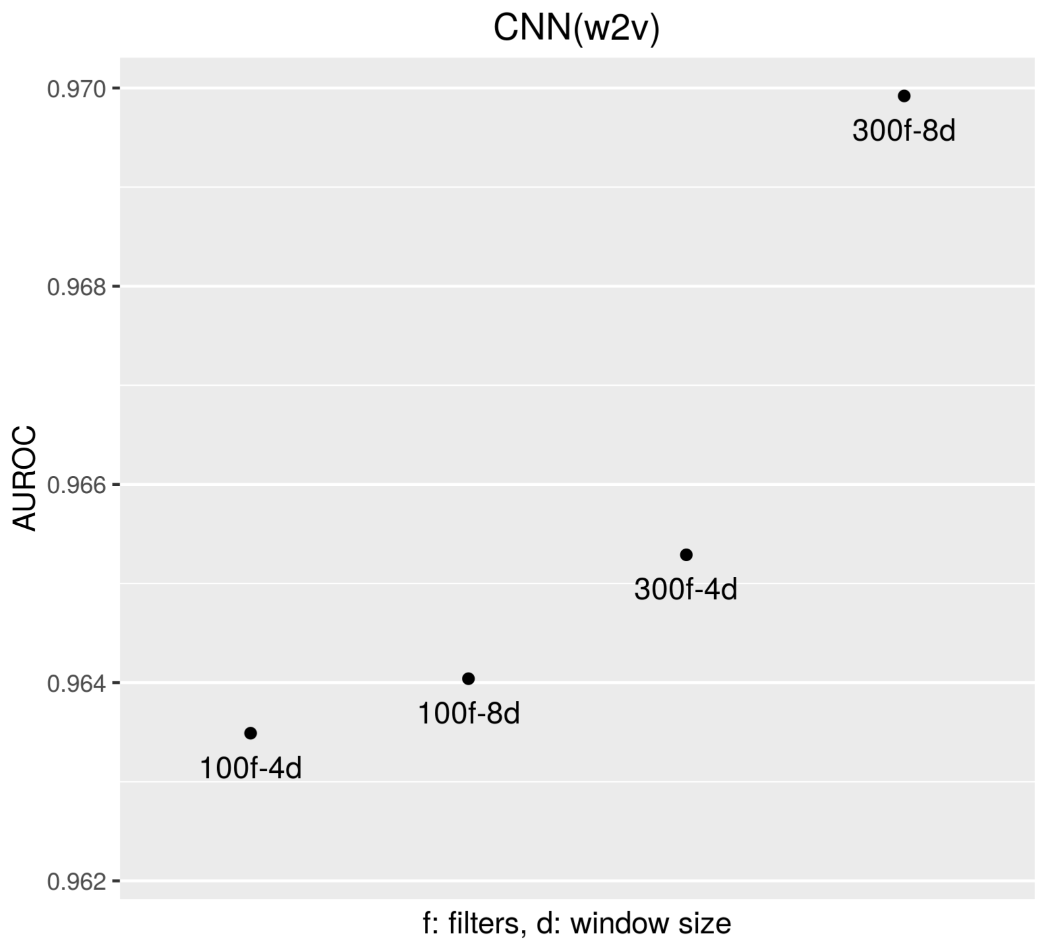


Figure 5.6: This figure illustrates the performance of the CNN model with varying number of filters $f$ and window size $d$ trained on sequential data represented by w2v word vectors.

### 5.2.11 Ensembles

In this experiment section some ensembles were trained by uniformly weighting the best performance of various models. The LR, SVM and MLP models used were the ones that gave the best AUROC performance by themselves. The AUROC and accuracy scores of the ensembles are listed in table 5.2 along with some notable single models. And the AUROC performance of the all the ensembles (except for the LR-SVM-ensemble) is also illustrated in Figure 5.7a, the best performance of the CNN model is also added as a reference point (since it was the best of the non-ensemble methods tested). Two ensembles, the GRU-ensemble and WA-ensemble, that are available on the net and which gave very good results are also added to the illustration as reference points. The GRU-ensemble generated the results which garnered the 3rd best result in the competition. Both these ensembles used the NB-SVM (Naive Bayes Support Vector Machine) classification model (Wang and Manning (2012)). Many of the ensembles did significantly better than many of the single models and some outperformed them all, which is part of the motivation for ensembles, the fact that they should be able to outperform good complex models.

#### 5.2.11.1 WA-ensemble

The WA-ensemble is a weighted-average ensemble training logistic functions on BoW, d2v and NB-SVM (Wang and Manning (2012)). The code for the WA-ensemble is available on the net[9].

#### 5.2.11.2 GRU-ensemble

The GRU-ensemble is an ensemble of a NB-SVM, a MLP trained on d2v representations and a GRU trained on sequential data represented by word vectors. This ensemble is based on Mesnil et al. (2014) and garnered the 3rd best result in the competition. The code for the ensemble is available on the net [10].

### 5.2.12 Result Summary

This section summarizes the results of the experiments on the IMDB dataset. Table 5.2 contains the best AUROC results gained in the experiments, the accuracy of these results is also included. The performance of the best models is also illustrated in figure 5.7b. Of the common statistical methods used the Logistic Regression model did best on the Bag of Words representation, it also outperformed the best MLP performance on this representation. While of the common statistical methods SVM did best on the tf-idf representation of the data, it was however outperformed by the MLP model on the same representation. While on the sequential data the CNN performed quite well and outperformed the other non-ensemble methods trained on the non-sequential data representations. While ensembles of the previously trained models were able to outperform the CNN.

LR-SVM-MLP-CNN-ensemble and SVM-MLP-CNN-ensemble did significantly worse on the AUROC score than the CNN, but it is noteworthy to mention that they both significantly outperformed the CNN on the accuracy score. While the MLP-CNN-ensemble, LR-CNN-ensemble and the LR-MLP-CNN significantly outperformed the CNN on both the AUROC and accuracy score.

[9] `https://github.com/smartinsightsfromdata/kaggle-word2vec-movie-reviews`
[10] `https://github.com/smartinsightsfromdata/kaggle-sentiment-popcorn`

Table 5.2: The accuracy and AUROC results of some noteworthy models on the full IMDB dataset.

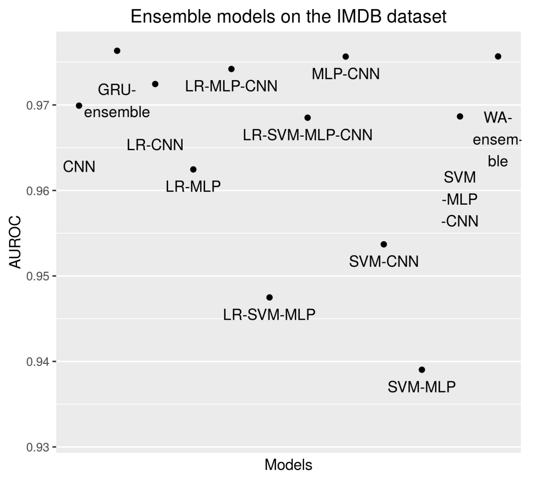
| Model | Input | Accuracy | AUC |
|---|---|---|---|
| LR-SVM | - | 0.77580 | 0.89195 |
| LR | BoW-74436-NP-NPratio | 0.86756 | 0.93683 |
| SVM-MLP | - | 0.86416 | 0.93903 |
| LR-SVM-MLP | - | 0.87692 | 0.94749 |
| MLP-2 | BoW5000NPratio | 0.88492 | 0.95088 |
| SVM-CNN | - | 0.89392 | 0.95370 |
| LR | tfidf(1,3;0) | 0.87372 | 0.95667 |
| LR-MLP | - | 0.89856 | 0.96246 |
| SVM | tfidf(1,2;0) | 0.90008 | 0.96359 |
| MLP-2-2 | tfidf60k(1,2) | 0.90320 | 0.96636 |
| LR-SVM-MLP-CNN | - | 0.91336 | 0.96851 |
| SVM-MLP-CNN | - | 0.91648 | 0.96866 |
| CNN | w2v (300f-8d) | 0.90812 | 0.96992 |
| LR-MLP-CNN | - | 0.91880 | 0.97421 |
| LR-CNN | - | 0.91620 | 0.97245 |
| MLP-CNN | - | 0.92108 | 0.97566 |
| WA-ensemble | - | - | 0.97568 |
| GRU-ensemble | - | 0.92384 | 0.97634 |

The MLP-CNN-ensemble was the model which was closest to of the WA- and GRU-ensembles performance, which performed very well on the IMDB contest. It performed about the same as the WA-ensemble, but there is still a minor gap between its performance and the GRU-ensembles performance. It is however interesting to see that the MLP-CNN model is so close.

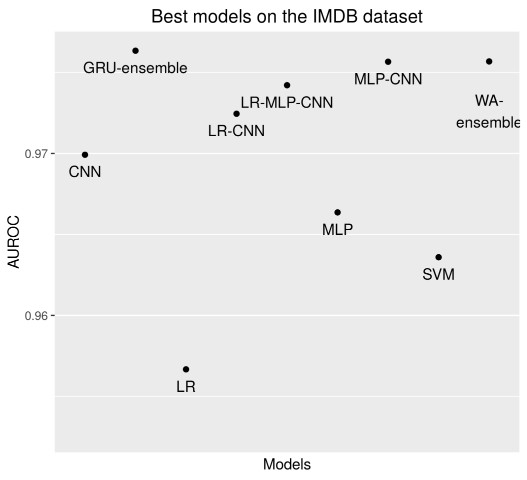
## 5.3 Automated Essay Scoring Task

The results obtained from the analysis of the Automated Essay Scoring dataset, described earlier in Section 2.3, will be presented here. The dataset was released as part of a competition with cash prizes at Kaggle. The goal of the prediction was to get the best predictions on a test dataset given the Quadratic Weighted Kappa (QWK) evaluation metric (defined in section 5.3.1). The scores of the competition can be found at the competitions leaderboard[11] (scores below a benchmark score around 0.5 seems to have been set to 0). None of the winning teams have published their models (atleast not in direct link to Kaggle).

[11] https://www.kaggle.com/c/asap-aes/leaderboard



(a) This figure illustrates the performance of various ensembles trained on the IMDB dataset.



(b) This figure illustrates the performance of the models that were among the best.

Figure 5.7: This figure illustrates the performance of various ensembles and the best models in general on the IMDB dataset.

Since the Kaggle competition is finished and is not open to post-competition submissions (unlike the IMDB competition) the models performance are evaluated on a subset of the training data used as a validation set.

All essay sets combined contains a vocabulary of 39056 words. The essay sets are divided into two types of essays, on type asks the writer to narrate an essay given a short problem statement and the other type asks the writer to read a medium size (1-4 pages) text source and asks the write to give a response given a problem

Table 5.3: Table showing different specifics of the essay sets.

| | Essay type | Number of essays | Average essay length | Vocabulary size | Score Range |
|---|---|---|---|---|---|
| Essay set 1 | Narrative | 1783 essays | 350 words | 15905 | $[2,12]$ |
| Essay set 2 | Narrative | 1800 essays | 350 words | 14412 | $[1,6]$, $[1,4]$ |
| Essay set 3 | Source Dependent | 1726 essays | 150 words | 6434 | $[0,3]$ |
| Essay set 4 | Source Dependent | 1772 essays | 150 words | 4899 | $[0,3]$ |
| Essay set 5 | Source Dependent | 1805 essays | 150 words | 4894 | $[0,4]$ |
| Essay set 6 | Source Dependent | 1800 essays | 150 words | 5343 | $[0,4]$ |
| Essay set 7 | Narrative | 1569 essays | 250 words | 10303 | $[0,30]$ |
| Essay set 8 | Narrative | 723 essays | 650 words | 12123 | [0.60] |

statement. Specifics about each dataset is shown in table 5.3.

### 5.3.1 Evaluation metric - Quadratic Weighted Kappa error metric

The predictions are evaluated using the Quadratic Weighted Kappa (QWK) error metric which measures the agreement between the predicted grades and the true grades. The metric's max value is 1, which implies complete agreement between the predicted grades and the true grades. If there is no agreement except for what is expected by chance the value is close to 0, and if there is even less agreement the value goes below 0.

In the Kaggle competition the average kappa value was computed over all the essay sets, this will also be done here.

#### 5.3.1.1 The method

Given a set of essays with $N$ possible grades and a two graders, Grader A and Grader B. Each essay grade $E$ can be seen as a pair of both graders response $e$, $(e_a, e_b)$.

The Quadratic Weighted Kappa is computed using a weight matrix $w$ and two histogram matrices $O$ and $E$.

The histogram matrix $O$ is a $N \times N$ matrix where $O_{i,j}$ corresponds to the number of essays that received a grading $i$ by Grader A and a grading $j$ by Grader B.

The weight matrix $w$ is a $N \times N$ weight matrix where each element $j, i \in [1, N]$ is defined by the equation

$$w_{i,j} = \frac{(i-j)^2}{(N-1)^2} \tag{5.1}$$

The histogram vector $E$ is a $N \times N$ matrix of the expected gradings assuming there is no correlation between the graders. This computed as the outer product of $E_a$ and $E_b$:

$$E = E_a E_b^T, \tag{5.2}$$

where $E_a$ and $E_b$ are histogram vectors of length $N$. Where cell $g \in N$ in vector $E_h$ ($h \in \{a, b\}$) reflects the number of essays the corresponding grader has given grade $g$. $E$ is normalized so that it has the same sum as $O$.

Given these matrices the quadratic weighted kappa is defined as

$$\kappa = 1 - \frac{\sum_{i,j} w_{i,j} O_{i,j}}{\sum_{i,j} w_{i,j} E_{i,j}} \tag{5.3}$$

To get the average of all the QWK of all the essay sets the fisher transformation defined here is used:

$$z = \frac{1}{2} \ln \frac{1+\kappa}{1-\kappa} \tag{5.4}$$

This transformation approaches infinity as the $\kappa$ value approaches 1, so the $\kappa$ values are capped at 0.999. The mean of the $z$ value of each essay set is computed (essay set 2 contains two scores that each are weighted by a weight of 0.5).

The final reverse transformation to get the quadratic kappa value is given as

$$\kappa = \frac{e^{2z} - 1}{e^{2z} + 1} \tag{5.5}$$

### 5.3.2 Features

The features used in the Automated Essay Scoring experiments are listed here:

- Bag of Words (BoW)
  - Feature importance: In the AES experiments it gave the most impact on the performance and was deemed the most important feature tested.
- tf-idf
  - Feature importance: In the AES experiments the representation alone didn't give significantly good performance, but it did significantly boost the Bag of Words representation, which gives it some significant importance.
- Document vectors (d2v)
  - Feature importance: The document vectors tested did not give very good results alone. The d2v(w2v-300) representation was tested for boosting models, it was deemed unimportant, an increase in features might however improve the performance.
- nbSent: Number of sentences in a document

  - Feature importance: This feature seemed to boost other representations well and was deemed important.

- Bag of WordClasses: The number and/or ratio that each word class appears (e.g. the number of nouns, verbs, etc...)
  - WC: Word class count
  - WR: Word class ratio
  - Feature importance: The WC and WR features gave good boosting results and were deemed important. But using both at the same time gave poorer results and the WR feature were preferred as it gave the best performance.

- Stop: The number of stop words in a document
  - Feature importance: This feature was deemed unimportant.

- avSent: Average sentence length
  - Feature importance: This feature was deemed unimportant.

- nbWords: Number of words in a document
  - Feature importance: This feature was deemed unimportant.

The positive and negative gain from using the document vectors (d2v), number of words, average sentence length and the number of stop words features to boost the other representations seemed to vary stochasticly around 0, implying that they only introduced bias to the modelling of the data and had no real benefit.

### 5.3.3 Training and validation set

Since the test data labels (grades) are not available the training data has been split into a training set and a validation set with a 70/30 ratio. Due to the relative small size of the training data this validation set might not be representative of the test data, but a larger validation set would decrease the training data too much (especially in the case of essay set 8 with only 723 essays). This decision was done to enable model performance evaluation as best as possible.

### 5.3.4 Linear Regression

This experiment section the Linear Regression model was trained on representations of the AES dataset. Linear Regression with both Lasso (l1) and Ridge Regression (l2) was used (i.e. Elastic Net). The Linear Regression experiments on the essay data set are illustrated in Figure 5.8. Figure 5.8a plots the Linear Regression model against

BoW representations with varying vocabulary size. It shows that a vocabulary of 200 is optimal. Figure 5.8b shows the Linear Regression model with a BoW vocabulary of 200 boosted by the various other features, it also shows the "best" model found through line search.

Figure 5.8c shows the Linear Regression model trained on various N-grams of the tf-idf(1,$g$) representation with varying vocabulary sizes $n$. It shows that a tf-idf representation of size 300 consisting of N-grams of size $g = 2$ (i.e. 2-gram/bi-grams) is optimal. Figure 5.8d shows the Linear Regression model trained on tf-idf(1,2)-300 boosted by the various extra features. The "best" model found through line search is also shown.

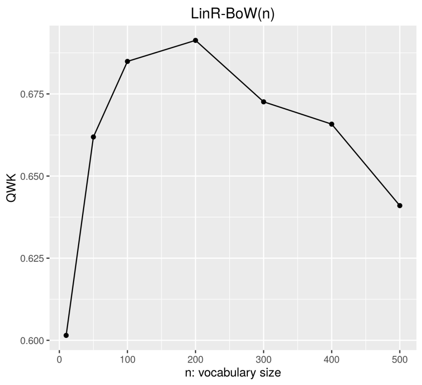
Figure 5.8e shows the best Linear Regression models based on BoW and tf-idf representations and a better combination of the representations found through line search.

### 5.3.5 SVR

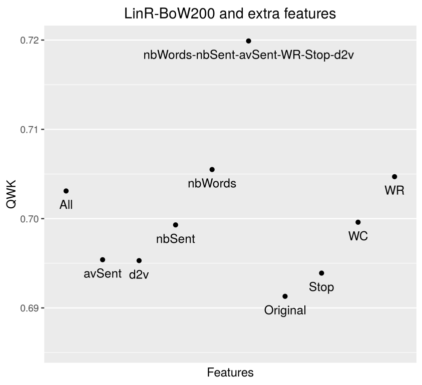
In this experiment section the SVR model trained on representations of the AES dataset. The SVR experiments on the essay dataset are illustrated in Figure 5.9. Figure 5.9a shows the performance of the SVR model plotted against the vocabulary size of the BoW representation. It shows that the optimal BoW vocabulary size is 1000. Figure 5.9b shows the performance of the SVR model plotted against varying vocabulary sizes of the tf-idf representations with varying N-gram sizes. It shows that the a vocabulary size of 50 and N-gram of size 1 is optimal for the tf-idf representations. The model trained on the tf-idf representations alone gave poor results compared to the other models trained on the essay dataset. Boosting the BoW representation with tf-idf subsets were tested, this increased performance slightly. The best boosted model was found through line search and the performance compared to a the best model trained on both BoW and the extra features are illustrated in Figure 5.9c.

### 5.3.6 MLP

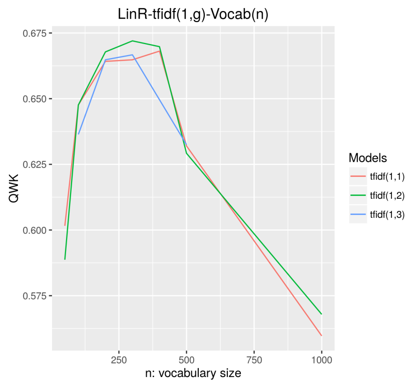
In this experiment section the MLP model is trained on representations of the AES dataset. The MLP's were trained with the Hyperbolic Tangent activation function (as it outperformed the ReLU activation function) and the final layer is a linear regression layer. A dropout parameter of 0.8 was used for the initial input layer and a parameter of 0.5 was used for the additional layers, except the final regression layer. There seems to be too many features and too few observations making it hard for the MLP to converge and generalize well. The performance of the MLP on the essay data is illustrated in Figure 5.10.



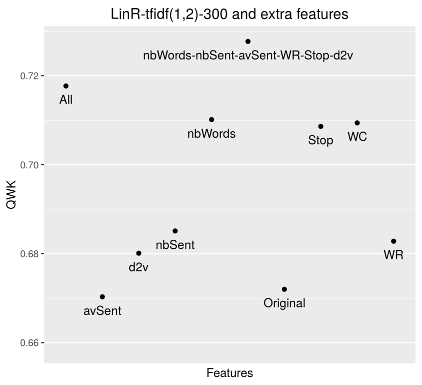
(a) The performance of the Linear Regression model plotted against the vocabulary size of the BoW representation.



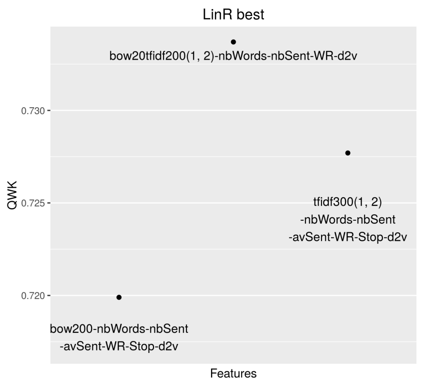
(b) The performance of the LinR-BoW200 model boosted by the various extra features.



(c) The performance of the Linear Regression model plotted against tf-idf$(1, g)$ with various N-gram sizes $g$ and varying vocabulary size $n$.

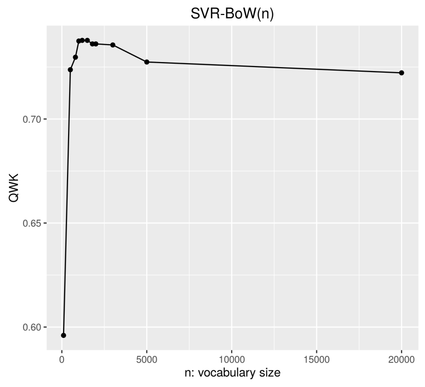


(d) The performance of the LinR-tfidf(1,2)-300 model boosted by the various extra features.

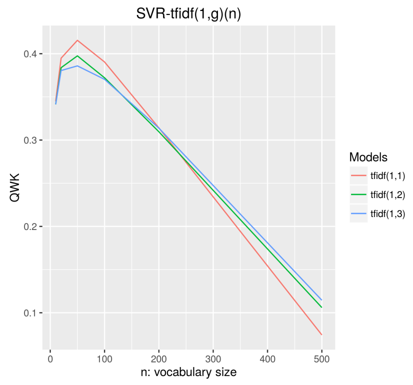


(e) The best Linear Regression models trained on BoW or tf-idf representations and a better model trained on a combination of BoW and tf-idf.

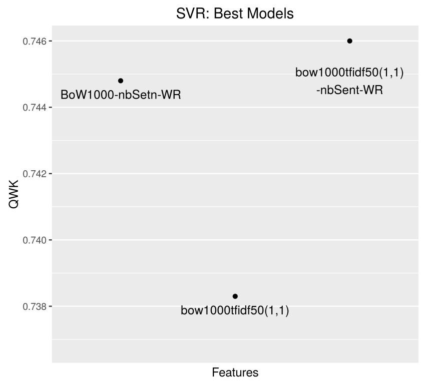
Figure 5.8: This Figure illustrates the Linear Regression models performance on the essay dataset.



(a) The performance of the SVR model plotted against the vocabulary size of the BoW representation.



(b) The performance of the SVR model plotted against the vocabulary size $n$ of the tfidf$(1,g)$ representation with different N-gram sizes $g$.



(c) The performance of boosted SVR models trained on both tf-idf, BoW and the extra features available.

Figure 5.9: This figure illustrates the performance of the SVR model on the essay dataset.

At first all the features are fed into the MLP for itself to decide which are important or not (which is the strong argument that the MLP should be able to do). These experiments with different number of layers and nodes are illustrated in Figure 5.10a. In this figure the MLP's performance is plotted against the number of nodes $n$ of a specific layer, for readability the binary logarithm of $n$, $k = \log_2(n)$ is used to scale the the $x$-axis. Further $k = -1$ is defined to mean $n = 0$ (i.e. that the hidden layer is removed from the model). The MLP-$n$-1 performed the best and the optimal number of nodes seem to be met at $n = 256$ ($k = 8$) ($n > 512$ was not tested due to memory limitations). The non-convex performance of the MLP-$n$-1 model could be due to the small data size, impact the MLP's ability to generalize well. It does however look like one could approximate a convex function which seems to increase atleast until 256 or 512 nodes $n$.

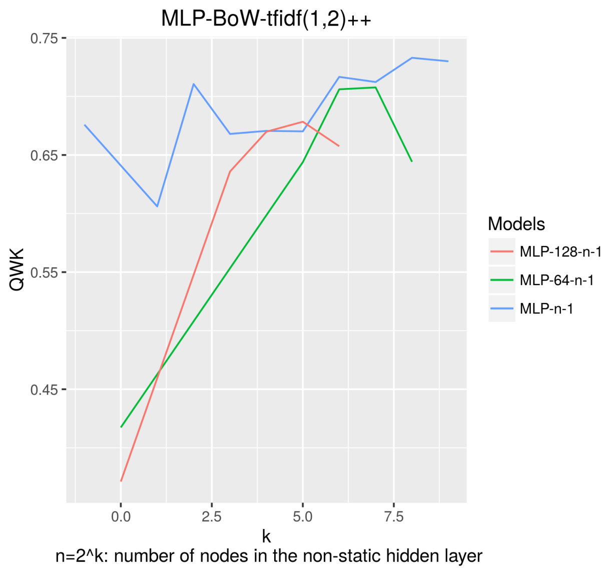
However since using both the complete BoW and tf-idf representations at the same time can seem a little bit redundant and since the number of observations are so low, which can hinder the MLP's performance when faced with too many redundant features it was also tested separately on tf-idf and BoW representations. The MLP trained on the BoW representation outperformed both the one trained on the tf-idf representation and the one trained on both representations. The best performance on both sets found through line search are illustrated in Figure 5.10b. It could be assumed that through line search there is the potential to find an optimal size of the tf-idf representation which could boost the MLP trained on the BoW representation, as was the case for both the Linear Regression and SVR models. This was however not tested.

### 5.3.7 CNN

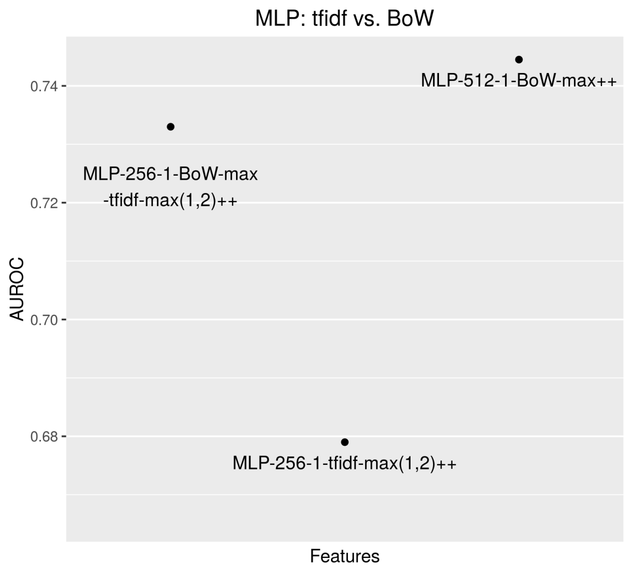
This section briefly discuss CNN modelling on the AES dataset. CNN models trained on the essay dataset generalized poorly. This poor performance could be due to the relatively small amount of observations available relative to the data sizes most Deep Learning models are motivated for.

### 5.3.8 Result Summary

This section summarizes the results of the experiments on the IMDB dataset. Table 5.4 lists the best results of each of the models trained on the essay text data, this data is also illustrated in Figure 5.11. The best MLP model did not outperform both the common statistical methods used, this is could be due to the relatively small size of essays in each set. The Linear Regression model performed slightly worse than the MLP model, and this performance (of the Linear Regression model) was only obtained after carefully changing the features through line search. The SVM performed a little bit better than the MLP, so small it is up to chance which performs the best. And this was after doing line search for the SVM with all the available



(a) The plots showing the performance of the MLP model with various layers and node combinations trained on all features available, including both the BoW and the tf-idf representations.



(b) This plot shows the MLP's best performance on the BoW, the tfidf(1,2) and the BoW-tfidf(1,2) representations. In this experiments all the extra features (nbWords-nbSent-avSent-WC-WR-Stop-d2v) were added to the feature set trained on.

Figure 5.10: This figure illustrates the performance of the MLP on the essay data.

Table 5.4: The QWK results of best models on the essay dataset.

| Model | Input | QWK |
|---|---|---|
| mlp256 | tfidf-max(1, 2)-nbWords-nbSent-avSent-WC-WR-Stop-d2v | 0.6790 |
| mlp256 | bow-max-tfidf-max(1, 2)-nbWords-nbSent-avSent-WC-WR-Stop-d2v | 0.7330 |
| LinR | bow20tfidf200(1, 2)-nbWords-nbSent-WR-d2v | 0.7337 |
| mlp512 | bowNone-nbWords-nbSent-avSent-WC-WR-Stop-d2v | 0.7445 |
| SVR | bow1000tfidf50(1, 1)-nbSent-WR | 0.7460 |

features. But the MLP also had to be tinkered for this performance as it would not perform on all the available features. Of course all these results are very close and since the validation set is so small it can be kind of random which models performs the best based on both initializations and the subset from available training data used for validation.

#### 5.3.8.1 Specific essays

This section summarizes the performance of the models on each of the essay sets. The performance on each essay set is listed in Table 5.5 and illustrated in Figure 5.12, for the MLP, Linear Regression and SVR models that gave the best performance. The different specifics are as previously mentioned displayed in Table 5.3. As the results shows the hardest sets to model were set 2 (both domains), set 3 and set 8. They penalized the final QWK score quite significantly. While being one of the longest essays and having a sizable vocabulary, essay set 2 could be hard to model as it needs to model two separate scores ("Writing Applications" and "Language Conventions"). Another interesting note about essay set 2 is that it is the only essay set in which the final score (determined by the first grader) isn't corrected if the second grader finds it to deviate too much. This could make the modelling much harder as personal traits of the grader could influence the final scores significantly. This could make the scores of an essay deemed to be on a specific level to not be ideally normal distributed around that levels appropriate score, possibly with a skewed or very wide distribution. Essay set 3 is not significantly different from the other essay sets, other than having a bit 20% bigger vocabulary size than the other essay sets of similar length. The difficulty with essay set 8 i probably due to the few (732) essays available and the fact that they had an average length of 650 words, which is more than double the average length of most of the other essay sets.

As the results of the different models on each essay set shows an ensemble model or a method of using specific models for each essay set could boost the performance, a combination of these two methods could also give a further boost to the performance. This would however be more interesting to analyze with more data available to test on.

Table 5.5: The QWK results of best models on the essay dataset with the score of each separate essay set included. The best result on each essay set is highlighted.

| Model | Set 1 | Set 2-1 | Set 2-2 | Set 3 | Set 4 | Set 5 | Set 6 | Set 7 | Set 8 | Final QWK |
|---|---|---|---|---|---|---|---|---|---|---|
| MLP-tfidf | 0.731 | 0.692 | 0.605 | 0.498 | 0.755 | 0.7356 | 0.717 | 0.713 | 0.568 | 0.6790 |
| MLP-BoW-tfidf | 0.752 | 0.679 | 0.623 | 0.575 | 0.782 | 0.804 | 0.788 | 0.775 | 0.671 | 0.7330 |
| LinR | 0.779 | 0.628 | 0.526 | 0.577 | 0.794 | 0.820 | **0.796** | 0.782 | 0.615 | 0.7337 |
| MLP-BoW | **0.834** | **0.695** | 0.589 | 0.614 | 0.800 | 0.776 | 0.729 | **0.800** | **0.688** | 0.7445 |
| SVR | 0.779 | 0.662 | **0.628** | **0.624** | **0.817** | **0.828** | 0.794 | 0.774 | 0.621 | **0.7460** |

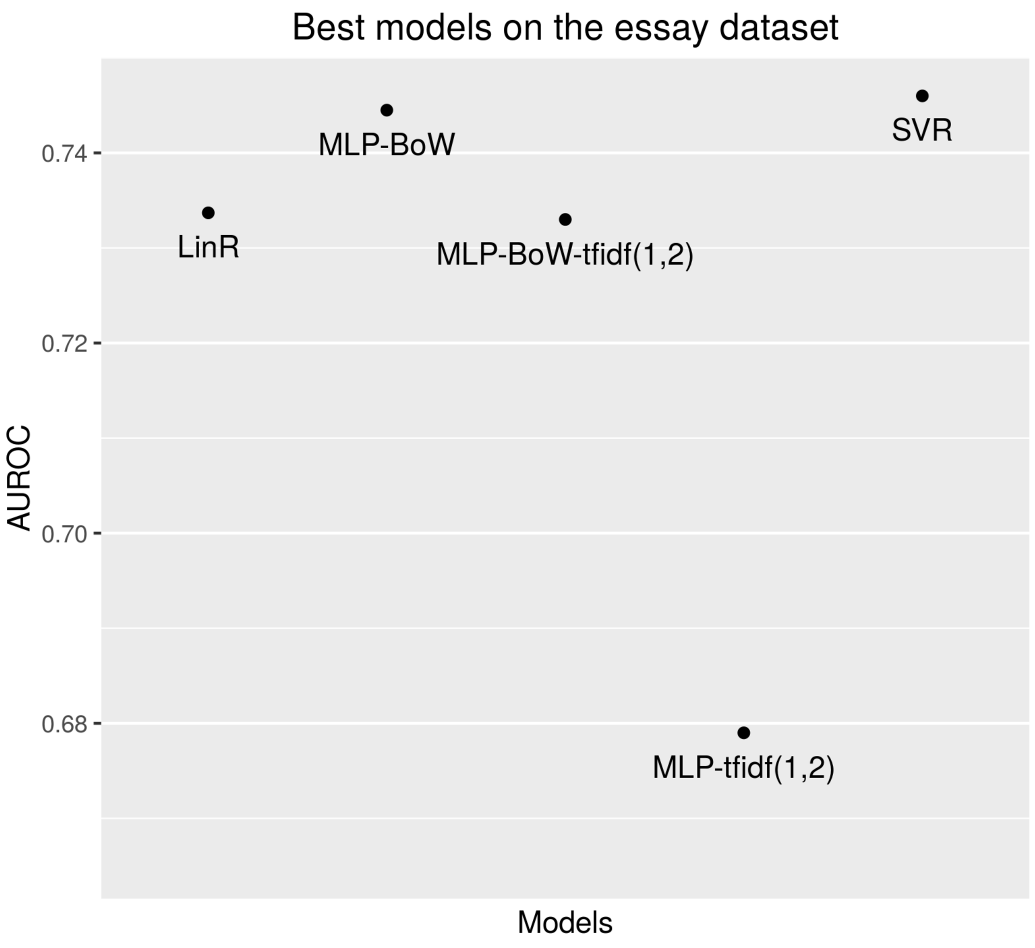


Figure 5.11: This figure illustrates the performance of the best models on the essay dataset.

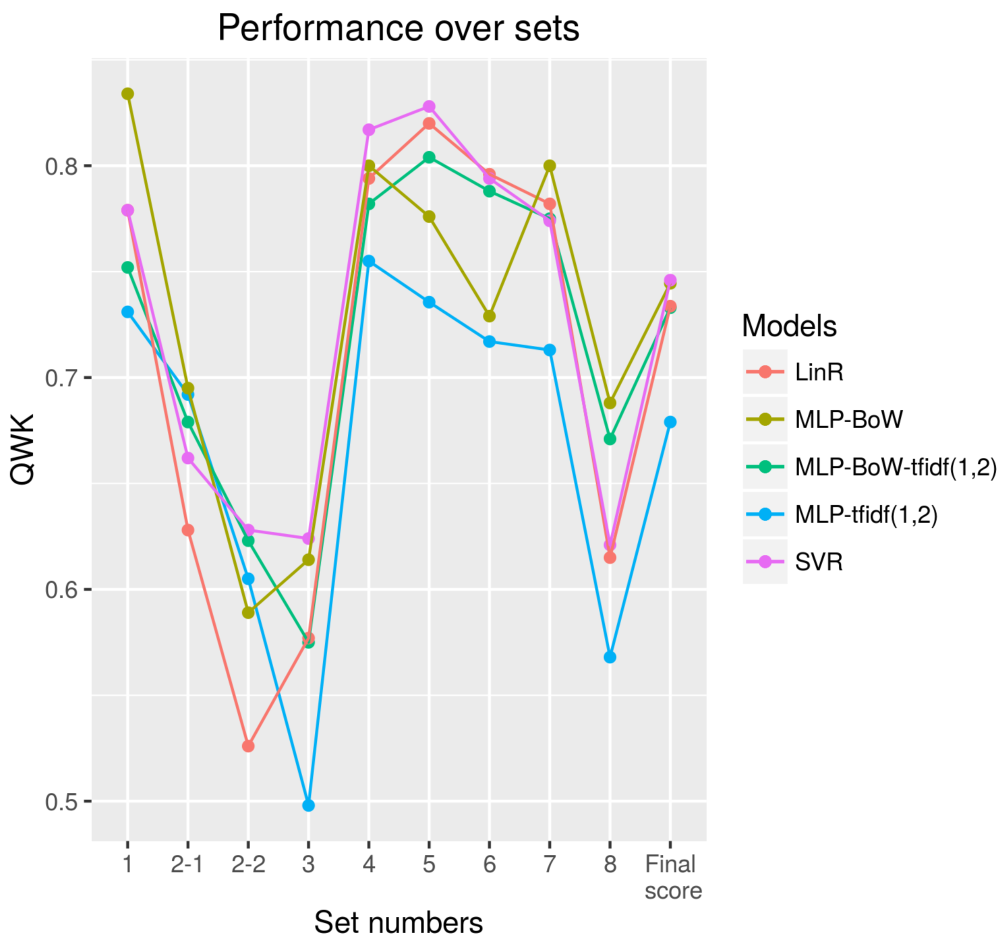


Figure 5.12: This figure illustrates the best models performance on each essay set.

# 6 Conclusion

The results of the analysis in this thesis shows that while Deep Learning models can outperform or perform close to regular classification and regression methods, the common classification and regression methods can still perform quite well on complex text data. Common classification and regression methods also take very little time to train while Deep Learning methods can take significantly longer to train even with powerful CPU/GPU's.

With enough computing power Deep Learning models are feasible for the data sizes tested on in this thesis if one wants a to increase the performance even by just some small percent. This boost in performance is assumed to scale with the data size. This implies that Deep Learning models can be extremely rewarding for big companies such a Yahoo! and Google which have access to much bigger data sets and much more computing power than used in this thesis. For these companies features and models that only improve the performance by a small percentage are still valuable even with a potential significant increase in computing power needed.

## 6.1 Further work

There are still many relevant interesting things left to test, some of them are mentioned here. Further interesting things to test include GloVe vectors, more testing on ensembles of both models and input, generate and use domain vectors with a significant greater number of features than 300. Testing on domain vectors of smaller sizes would also be interesting, to analyze the information gained through scaling the vector/feature size. Testing the performance of common statistical methods vs. Deep Learning models on different subsets sizes of the data sets would also be interesting to see how the performance of the models scale. Testing recurrent neural networks (RNN), such as LSTM and GRU, is also of high interest. And the next step in form of data sets to test on would be to test models (especially deep learning models) on data sets that are both large scale and complex.

With respect to the Automated Essay Scoring (AES) problem extra features expressing the complexity of words (where each word is given a complexity level by a complexity-dictionary) and features counting spelling errors could improve the performance of the models (spelling errors can be found using the PyEnchant[1] Python library). The Stanford Parser[2] could also be interesting to try out, it is a statistical tool that works out the grammatical structure of sentences. Testing with

[1]http://pythonhosted.org/pyenchant/

[2]http://nlp.stanford.edu/software/lex-parser.shtml

domain word vectors on the AES dataset could also show interesting results, as these domain vectors could be more expressive than the IMDB word vectors, as the AES problem is a more complex text problem than the IMDB problem (on which the global w2v vectors generalized relatively similarly to the domain vectors).

# Bibliography

Ba, J. and B. Frey (2013). “Adaptive dropout for training deep neural networks”. In: *Advances in Neural Information Processing Systems*, pp. 3084–3092 (cit. on p. 24).

Baroni, M., G. Dinu, and G. Kruszewski (2014). “Don’t count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Vol. 1, pp. 238–247 (cit. on p. 28).

Bekkerman, R., R. El-Yaniv, N. Tishby, and Y. Winter (2003). “Distributional word clusters vs. words for text categorization”. In: *The Journal of Machine Learning Research* 3, pp. 1183–1208 (cit. on p. 12).

Ben-Hur, A., D. Horn, H. T. Siegelmann, and V. Vapnik (2002). “Support vector clustering”. In: *The Journal of Machine Learning Research* 2, pp. 125–137 (cit. on p. 12).

Chen, T. and H. Chen (1995). “Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems”. In: *Neural Networks, IEEE Transactions on* 6.4, pp. 911–917 (cit. on p. 16).

Cho, K., B. van Merriënboer, D. Bahdanau, and Y. Bengio (2014). “On the properties of neural machine translation: Encoder-decoder approaches”. In: *arXiv preprint arXiv:1409.1259* (cit. on p. 35).

Chung, J., C. Gulcehre, K. Cho, and Y. Bengio (2014). “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (cit. on p. 34).

Fukushima, K. (1980). “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological cybernetics* 36.4, pp. 193–202 (cit. on p. 29).

Gao, J., L. Deng, M. Gamon, X. He, and P. Pantel (2015). *Modeling interestingness with deep neural networks*. US Patent 20,150,363,688 (cit. on p. 30).

Gers, F. A. and J. Schmidhuber (2001). “LSTM recurrent networks learn simple context-free and context-sensitive languages”. In: *Neural Networks, IEEE Transactions on* 12.6, pp. 1333–1340 (cit. on p. 34).

Glorot, X. and Y. Bengio (2010). “Understanding the difficulty of training deep feedforward neural networks”. In: *International conference on artificial intelligence and statistics*, pp. 249–256 (cit. on p. 19).

Glorot, X., A. Bordes, and Y. Bengio (2011). “Deep sparse rectifier neural networks”.

In: *International Conference on Artificial Intelligence and Statistics*, pp. 315–323 (cit. on p. 18).

Goodfellow, I. J., D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio (2013). "Maxout networks". In: *arXiv preprint arXiv:1302.4389* (cit. on p. 24).

Guthrie, D., B. Allison, W. Liu, L. Guthrie, and Y. Wilks (2006). "A closer look at skip-gram modelling". In: *Proceedings of the 5th international Conference on Language Resources and Evaluation (LREC-2006)*, pp. 1–4 (cit. on p. 13).

He, K., X. Zhang, S. Ren, and J. Sun (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *arXiv preprint arXiv:1502.01852* (cit. on pp. 18, 20).

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780 (cit. on p. 34).

Hornik, K., M. Stinchcombe, and H. White (1989). "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5, pp. 359–366 (cit. on p. 15).

Hu, M. and B. Liu (2004). "Mining and summarizing customer reviews". In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 168–177 (cit. on p. 40).

Hubel, D. H. and T. N. Wiesel (1968). "Receptive fields and functional architecture of monkey striate cortex". In: *The Journal of physiology* 195.1, pp. 215–243 (cit. on p. 29).

Iyyer, M., V. Manjunatha, J. Boyd-Graber, and H. D. III (2015). "Deep Unordered Composition Rivals Syntactic Methods for Text Classification". In: (cit. on p. 12).

Johnson, R. and T. Zhang (2014). "Effective use of word order for text categorization with convolutional neural networks". In: *arXiv preprint arXiv:1412.1058* (cit. on p. 30).

Kalchbrenner, N., E. Grefenstette, and P. Blunsom (2014). "A convolutional neural network for modelling sentences". In: *arXiv preprint arXiv:1404.2188* (cit. on p. 30).

Kim, Y. (2014). "Convolutional Neural Networks for Sentence Classification". In: *CoRR* abs/1408.5882 (cit. on p. 30).

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105 (cit. on p. 30).

Le, Q. V. and T. Mikolov (2014). "Distributed representations of sentences and documents". In: *arXiv preprint arXiv:1405.4053* (cit. on p. 11).

LeCun, Y. A., L. Bottou, G. B. Orr, and K.-R. Müller (2012). "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, pp. 9–48 (cit. on p. 17).

LeCun, Y., Y. Bengio, and G. Hinton (2015). "Deep learning".
In: *Nature* 521.7553, pp. 436–444 (cit. on p. 17).

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998).
"Gradient-based learning applied to document recognition".
In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (cit. on p. 29).

Lebret, R. and R. Collobert (2015).
"Rehabilitation of Count-based Models for Word Vector Representations".
In: *Computational Linguistics and Intelligent Text Processing*. Springer,
pp. 417–429 (cit. on p. 29).

Liu, B., M. Hu, and J. Cheng (2005).
"Opinion observer: analyzing and comparing opinions on the web".
In: *Proceedings of the 14th international conference on World Wide Web*. ACM,
pp. 342–351 (cit. on p. 40).

Maas, A. L., A. Y. Hannun, and A. Y. Ng (2013).
"Rectifier nonlinearities improve neural network acoustic models".
In: *Proc. ICML*. Vol. 30, p. 1 (cit. on p. 18).

Maas, A. L., R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts (2011).
"Learning Word Vectors for Sentiment Analysis".
In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150.
doi: 978-1-932432-87-9 (cit. on p. 3).

Mesnil, G., T. Mikolov, M. Ranzato, and Y. Bengio (2014).
"Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews". In: *arXiv preprint arXiv:1412.5335* (cit. on pp. 34, 55).

Mikolov, T., W.-t. Yih, and G. Zweig (2013).
"Linguistic Regularities in Continuous Space Word Representations."
In: *HLT-NAACL*, pp. 746–751 (cit. on pp. 10, 26).

Mikolov, T., M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur (2010).
"Recurrent neural network based language model." In: *INTERSPEECH*. Vol. 2,
p. 3 (cit. on p. 34).

Mikolov, T., S. Kombrink, L. Burget, J. H. Černockỳ, and S. Khudanpur (2011).
"Extensions of recurrent neural network language model". In: *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE,
pp. 5528–5531 (cit. on p. 34).

Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013a).
"Distributed representations of words and phrases and their compositionality".
In: *Advances in neural information processing systems*, pp. 3111–3119
(cit. on pp. 10, 26).

Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013b).
"Efficient estimation of word representations in vector space".
In: *arXiv preprint arXiv:1301.3781* (cit. on pp. 10, 25, 26).

Pennington, J., R. Socher, and C. D. Manning (2014).
"GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014*

*Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pp. 1532–1543 (cit. on pp. 10, 25, 28).

Rong, X. (2014). "word2vec Parameter Learning Explained". In: *arXiv preprint arXiv:1411.2738* (cit. on pp. 26, 28, 29).

Serre, T., L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio (2007). "Robust object recognition with cortex-like mechanisms". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29.3, pp. 411–426 (cit. on p. 29).

Shen, Y., X. He, J. Gao, L. Deng, and G. Mesnil (2014). "A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval". In: *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. CIKM '14. Shanghai, China: ACM, pp. 101–110. isbn: 978-1-4503-2598-1. doi: 10.1145/2661829.2661935 (cit. on p. 30).

Smirnov, E. A., D. M. Timoshenko, and S. N. Andrianov (2014). "Comparison of regularization methods for imagenet classification with deep convolutional neural networks". In: *AASRI Procedia* 6, pp. 89–94 (cit. on p. 24).

Smola, A. and V. Vapnik (1997). "Support vector regression machines". In: *Advances in neural information processing systems* 9, pp. 155–161 (cit. on p. 19).

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958 (cit. on pp. 23–25).

Sussillo, D. (2014). "RANDOM WALKS: TRAINING VERY DEEP NONLIN-EAR FEED-FORWARD NETWORKS WITH SMART INI". In: *arXiv preprint arXiv:1412.6558* (cit. on p. 16).

Tang, Y. (2013). "Deep learning using linear support vector machines". In: *arXiv preprint arXiv:1306.0239* (cit. on p. 19).

Tomczak, J. M. (2013). "Prediction of breast cancer recurrence using Classification Restricted Boltzmann Machine with Dropping". In: *CoRR* abs/1308.6324 (cit. on p. 24).

Wan, L., M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus (2013). "Regularization of neural networks using dropconnect". In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1058–1066 (cit. on p. 24).

Wang, S. and C. D. Manning (2012). "Baselines and bigrams: Simple, good sentiment and topic classification". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*. Association for Computational Linguistics, pp. 90–94 (cit. on p. 55).

Wu, H. (2009). "Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions". In: *Information Sciences* 179.19, pp. 3432–3441 (cit. on p. 16).

Yao, K., G. Zweig, M.-Y. Hwang, Y. Shi, and D. Yu (2013). "Recurrent neural networks for language understanding." In: *INTERSPEECH,* pp. 2524–2528 (cit. on p. 34).

Zhang, Y. and B. Wallace (2015). "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification". In: *arXiv preprint arXiv:1510.03820* (cit. on p. 31).