Information Retrieval with Dimensionality Reduction using Deep Belief Networks

Vincent Slot

Master Thesis

Rijksuniversiteit Groningen,
Department of Artificial Intelligence

April, 2016

Supervisors:
Dr. Marco Wiering
Artificial Intelligence, Rijksuniversiteit Groningen
Prof. Dr. Gertjan van Noord
Language Technology, Rijksuniversiteit Groningen

university of groningen

faculty of mathematics and natural sciences

Abstract

The most common method to achieve dimensionality reduction of data would be to perform matrix decompositions, as is the case in Latent Semantic Analysis (LSA) and Principal Component Analysis (PCA). However, dimensionality reduction can also be achieved by layered Restricted Boltzmann Machines (RBM), called Deep Belief Networks (DBN). A primary application of LSA is information retrieval (IR), in this context often referred to as Latent Semantic Indexing (LSI). Dimensionality reduction in IR is used to interpret data on an abstract level to represent the data as patterns in more informative, fewer dimensions. Through these patterns meaning can be attributed to words and documents.

This research investigates whether or not neural networks can compete with these traditional methods of dimensionality reduction in IR in the setting of a small database. Experiments are conducted with a search engine constructed for the purpose of comparison of the different algorithms. It compares vector space search, LSI search, DBN search and different variations of DBN search. These variations are based on so-called phantom document querying, where queries are being replaced by the entire text body of the top search results of a different search engine. For each search algorithm different parameter configurations are being researched and their optimal implementations are compared with each other.

The results show that the DBN shows slightly inferior results to the traditional methods in the application of deep learning in a relatively small database.

Contents

Chapter 1

Introduction

1.1 Relevance

In the last decade a new area of artificial intelligence has become popular: deep learning [24]. As opposed to 'traditional' or 'shallow' machine learning, deep learning has a different approach to representing information. Instead of classifying or clustering data based on the direct input, a deep learning algorithm will process information toward an abstract level. So not only the direct data are important, but also the context of the data, and any latent cross-connections that might be in the data. Deep learning algorithms represent the data as meaningful patterns which might not be directly visible to the naked eye.

A popular quote circulating on social media these days: *"A smart man knows that a tomato is a fruit. A wise man knows not to put it in a fruit salad."*. This captures the essence of deep learning in layman's terms. In this metaphor, it is easy for an algorithm to classify a tomato as fruit. It's an entirely different thing to deduce, based on knowledge of the world around us, that a tomato doesn't belong in a fruit salad.

Deep learning arguably is or could become an entirely new paradigm within the field of artificial intelligence: one step closer to the way the human brain processes the information from our natural sensors, which has proven to be incredibly hard, if not impossible, to formalize. It is a very obvious consequence then, that the big software corporations and leading AI-scientists take deep learning very seriously. One of the most popular deep learning architectures is the convolutional neural network [23]. However, this is a supervised learning algorithm. So the data needs to be labelled or annotated. When an application calls for unsupervised machine learning, deep belief networks (DBNs) [16] are a

common type of deep learning algorithm.

Another subject that concerns this research is the field of information retrieval (IR) [27]. This is a field that focuses on retrieving the most relevant information from an unordered database. This is becoming more and more relevant in the recent years, where the internet is taking a leap in the amount of information available to anyone who owns a computer and an internet connection. Big data is a term that's buzzing around the large internet corporations these days. Everything a user does on the web generates statistics: activity on social media, search behavior on the web, purchases in web stores, etcetera.

As the amount of data available to users and companies grows and grows, it becomes harder to find the right information. This is the reason search engines become more and more important. Searching for novel methods to find the most relevant search results in a database is therefore important. The innovation in this area should keep up with the increase in size of the data.

1.2 Research Question

In this thesis the use of deep learning with DBNs in the context of IR will be examined. IR is the branch of science that focuses on techniques to effectively find relevant data in databases. There are many ways to process and index data to make a search engine as effective as possible, among which even ways to make abstractions of the data. This leads us to the following research question:

Can deep neural networks improve search engines, when compared to the conventional methods in information retrieval?

The question will be answered based on experiments with software programmed for this research.

1.3 Outline

This thesis has the following structure.

- First the necessary background information on IR and DBNs will be provided in Chapter 2
- Then the different types of search engines that will be compared will be discussed in Chapter 3. Also the experimental setup will be discussed in this chapter.

- The results will be presented in Chapter 4
- The results will be discussed in Chapter 5
- Finally the conclusions and recommendations will be given in Chapter 6

Chapter 2

Context and Background

2.1 Historical Context: the Origins of Information Retrieval

To understand the significance of this research, it is vital to know the context in which it takes place. For that reason this section contains a brief history of Information Retrieval (IR). This section is mostly based on [34].

Over the years IR has become a mature field of science. The need to efficiently retrieve and rank data has grown along with the need to find our way through enormous masses of data that become more and more common nowadays. IR is the science of finding the desired information among documents or entries in a database. It is based around a query given by the end-user, which determines the subset of most relevant documents in the database. This subset of documents is called the search result and can be ranked according to relevance. The most common appliance of IR is one most people use on a daily basis: searching the entire Internet with the well-known search engines of *Google*, *Bing* or *Yahoo!*. However the full spectrum of applications is a lot broader than that. Any system using a database of semi-structured data is subject to IR. Some examples include video search (e.g. *YouTube*), web shops (e.g. *amazon.com*) and social media (e.g. *Twitter*).

Computer-based search applications have been around since the 1940s. Ever since computers started to appear there has been information to be searched through. Most of these databases were small enough to manually set up. However, with the increase of processor and memory capacities, the libraries of information grew, and search strategies became more and more advanced.

2.1.1 Development of Basic Information Retrieval Techniques

From the 1950s important IR techniques surfaced: indexing and ranked retrieval. This marked a huge step: from a predetermined hierarchy, often numerically labeled, to a list of keywords for each document [45]. This made it clear in a single glance if a document could possibly be relevant to a query or not. Nowadays this seems like a basic and trivial step, but obviously it was a huge improvement back then.

The next big step came quickly after: ranked retrieval. Instead of just returning whether or not a document fits a query, return how *good* it fits that particular query [26]. Some documents are more relevant to a certain query than others. The first basal techniques assumed that the more a term appeared in a document, the more relevant it would be to queries containing that term. This technique later became known as term-frequency weighting.

The 1960s gave birth to a number of relevant techniques that we still use today: vector-space models [43], relevance feedback [29] and document clustering. At this time search engines became interesting for bigger data, and thus for companies. However, commercial enterprises conservatively held on to the old, inferior techniques. Somehow current research on IR did not make it through into commercial applications.

The 1970s brought another major innovation to the field of IR: *tf-idf* [40, 33]. This relevance measure does not only consider the occurrence of a term in the document itself, but also considers how often it occurs in all the other documents. This way, a document is ranked also based on the context it is in. So in this case the terms in the query are also weighted: if that specific term appears in every document in the database, it is not going to yield relevant results. Hence it is unimportant. A term that is searched for and only appears in a few documents is very important by this measure.

2.1.2 Novel Approaches to Information Retrieval

Throughout the 80s and 90s two other important techniques, or rather, trends, appeared in the field of IR. The first one was to incorporate natural language processing in IR. This means that instead of just looking at the text as if it were an unordered set of words, these systems could consider things like the syntactical structure of data and could consider semantical interpretations like synonimity, lexical variation or named entities.

The second technique from this time period also used semantic information to organize the data: Latent Semantic Analysis (LSA) [11]. This method, consisting of various mathematical operations from the field of linear algebra, would reduce the bag-of-words interpretation to an abstract level, a semantic interpretation. If this analysis is used in the context of search engines, it is called Latent Semantic Indexing (LSI). This way search engines can utilize semantic knowledge about a query in ranking the search results. This method is the primary subject of this thesis.

Furthermore, during this period of time, information retrieval got a lot of interest from commercial parties. Research on the subject increased, as did the size of the unstructured data. During this period another idea dawned: there is no such thing as the Holy Grail of information retrieval. Each dataset is different in size, topics and lexical variety, so there will never be one ultimate information retrieval system.

2.1.3 The Emergence of Web Search Engines

The Internet quickly emerged in the course of the 1990s. It grew tremendously and thus the need for good information retrieval grew explosively. This development marked the birth of web search. According to the historical summary of [38] the actual first web search engine was called *Archie*, which searched through file names on an FTP-server. The index had to be built manually.

This first automatic indexing system emerged around 1993: the *World Wide Web Wanderer* from MIT. Its purpose was interestingly not to build a search index, but to do something that would be a sheer impossibility these days: measure the size of the Internet by indexing it all. This program is documented poorly, the author reported the findings online[1].

In 1994 a revolutionary search engine appears: *AltaVista*. In many ways this is a unique search engine, most notably because it accepts queries in natural language. Quickly many new web search engines pop-up, *Yahoo!*, *Lycos*, somewhat later *Google*, and more recently *Bing*. The latter examples are web search engines as we know them nowadays: smart complex systems that aim to match the user's query with the most relevant documents in their database. The sheer amount of research tech-giant Google performs[2] in the fields of IR, natural language processing and artificial intelligence suggests that their search

[1] Measuring the Growth of the Web, *http://www.mit.edu/people/mkgray/growth/*
[2] Research at Google, *http://research.google.com/pubs/papers.html*

results are scored by complex composite algorithms that together operate as one search engine.

2.1.4 PageRank

Up until this point, all the previously mentioned algorithms were based purely on content. However, there is another reason Google survived where many other search engines bit the dust. Google developed an algorithm called PageRank [28]. PageRank is an algorithm that views the Internet as a graph. Each webpage is represented by a node in this graph, and each link is represented by an edge. Via this structure the importance of certain web pages can be measured: getting a lot of links toward a page makes it more important. In this situation highly ranked pages add more importance than lowly ranked pages. For example, a certain page would benefit more from getting a link from the BBC than it would from getting a link from my personal web page.

Taking into account this information makes it easier to discriminate between important high-profile websites and someone's weblog where he or she writes about his or her hobbies. This would be a lot harder purely based on content. This algorithm is said to be one of the prime reasons that Google outgrew its peers AltaVista and Yahoo!. The significance of search results improves tremendously using PageRank.

2.2 Common Techniques in Information Retrieval

This section describes the most commonly used techniques in IR. It is mostly based on [27], unless referenced otherwise.

2.2.1 Search Engines

We all use search engines now and then, but what exactly happens under the hood when a search query is submitted? Basically a search engine consists of three parts: an index, a crawler (or spider, these terms are interchangeable) and an interface. The interface is of little technical interest: it is the (often graphical) environment in which the user submits his or her query and receives the list of search results. The index and the crawler constitute the technical side of the search engine.

In a giant database, it is unthinkable to try to match every word in the query to every word in every document. There is often too much irrelevant

text in the data for this to be useful. That is why documents need to be properly indexed. The index is basically a list of key-value pairs, where the key (the index) is the most relevant possible summary of the value (the document). So only the key is matched to the query, being able to handle requests a lot quicker, but also possibly better. Because not only is the index most often a lot smaller than its corresponding document, the developer of the search engine can exercise control on what should and what should not appear in the index. For example, unimportant words can be filtered out, while others are assigned extra importance.

The data in the index are gathered automatically by a program called a crawler. This program gets a list of seed URLs as its input, which it visits and pushes to the index. Then, it gathers the hyperlinks on those pages, and generates a new fetch list out of those links. This way the crawler 'crawls' through the Internet (which paints a picture why crawlers are also called 'spiders'), keeping anything it comes across.

2.2.2 Boolean Retrieval

The most basic form of a search engine is through Boolean Retrieval. Each document in the index is represented by an array of Boolean values. Each of these values represents a term that appears in the set of terms known to the search engine (the search engine dictionary). So each document simply is indexed by whether or not the term appears in it or not. To illustrate: an empty document will contain only zeros in its index, whereas a document that contains all the words in the dictionary (no matter how many times) will be indexed with only ones. When a one-word query is entered, all the documents containing that word will be returned. When a multi-word query is entered, a choice will have to be made which Boolean operator is going to be applied (*term1* OR *term2* vs. *term1* AND *term2*).

2.2.3 Scoring

One of the main problems with Boolean retrieval is that also the results are returned as a Boolean: either a document is a valid search result, or it is not. There is nothing in between. So the relevance of the search result is not reflected in any way; the results are unordered. Imagine performing a *Google*-search for "buy Mercedes", and having to go through the (at the time of typing) 121,000,000 results by hand searching for a relevant result, because they are unordered. That

would be a completely hopeless endeavor. This is where the concept of scoring comes in: the search engine calculates a score for each document about how well it matches the query. This should lead to a list of results where the most relevant ones are at the top, and the least relevant ones stay out of view: at the bottom. There is a wide range of different measures to calculate this score. The most commonly used ones will be discussed next.

2.2.4 Term Frequency

Term frequency weighting is the technique where the score of each document is defined as how often the words in the search query appear in the document. This means that it assumes that if a term is important to a document, it is probable that it will appear more often in it. Intuitively this makes sense: documents mentioning the word *car* ten times are probably related to cars. But one can imagine situations where this is not at all true. For example, if the word 'the' is the most frequent word in a document, is it the most important one? If a news article mentions a famous singer once, as opposed to a fan's web page where it appears 30 times, is the latter a 30 times more relevant search result? It seems term weighting is a step in the right direction, but we're not quite there yet.

2.2.5 Term Frequency - Inverse Document Frequency

So besides term frequency, another measure should be used, to prevent insignificant common words to gain the upper hand in determining the (order of the) search results. Because obviously not every term has the same importance, a measure was defined of how important a term is within a certain database. This measure is called Inverse Document Frequency. It is defined in the following formula:

$$idf_t = log\frac{N}{df_t}$$

The *idf* for term t is a logarithm of the total amount of documents in the database (N), divided by the amount of documents containing the term (df_t). Basically, the more rare a term is, the more important it becomes. The TF-IDF score for a term in a document can then be obtained by multiplying the *tf* by the *idf* score. This is a commonly used and good score for terms in a document, because it incorporates information about the context of a given document, instead of only counting within the document.

2.2.6 Vector Space Models

Now we can simply build a document-term matrix with corresponding TF-IDF scores. This matrix consists of document vectors $\vec{V}(d)$. One can use these vectors to calculate similarities between the vectors. One could, for example, consider using the vector difference between the two documents as a measure. However, this will lead to bad results when the length of the documents differ. Topical similar documents can have a very large vector difference if one is very short and the other very long. For this reason the common approach is to calculate the cosine similarity. This similarity of document d_1 and d_2 given their vector representations $\vec{V}(d_1)$ and $\vec{V}(d_2)$ is defined by the following formula:

$$sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}$$

In this formula the dot product of the vectors is corrected for their Euclidian length (which is the square root of the sum of all squared values). Another way to do the same thing would be to first calculate a length-normalized unit vector $\vec{v}(d) = \vec{V}(d)/|\vec{V}(d)|$. In that case a simple multiplication of two document vectors would suffice to find their cosine similarity. To apply this model to search engine technology, one should consider each query as a document. Then the score of each document d for a query q could be calculated by the dot product $\vec{v}(q) \cdot \vec{v}(d)$.

Inverted Indices

Since nowadays in the current state of hardware, storage is in many applications not really a bottleneck anymore, there is a trick to speed up the search process in a vector space model: inverted indexing. The potential downside being that one needs to store extra information.

In inverted indexing, there is an extra index: instead of only storing which document contains which term, we also store which term appears in which document. This way, we can check a query for the relevant documents, and skip expensive computations on documents which do not even contain one of the terms in the query.

Relevance Feedback and Query Expansion

Two more methods of improving search results are relevance feedback and query expansion. For relevance feedback, input from the user is needed. It is the

process of letting a user give feedback on which results are good and which are bad, and improving the set of results based on this feedback. More results similar to the documents which are marked as 'good' by the user will be returned, while documents similar to the ones marked 'bad' will be shunned. This way a search engine can be improved incrementally.

This type of feedback can also be implicit. In that case the user will not have to mark documents explicitly, but for example, clicking behavior is monitored. If some documents are often selected by many users in relation to a certain query, this could mean that these documents should be boosted for that specific query.

Query expansion is the technique of changing the query to improve search results. An example of this is stemming. In stemming each word in the overall dictionary is reduced to its stem. This way, for example, 'searched', 'searching' and 'search' will end up as the same entry in the feature set. This could also be applicable to query expansion: each word in the query could be reduced to its stem. This assumes that each conjugation of a word has the same, or at least similar, meaning.

An elaboration of the two techniques, relevance feedback and query expansion, is Phantom Document Masking, as explained in section 3.3.5. It combines any number of the most relevant search results into a new query. This way we use so-called blind relevance feedback (the top search results) and query expansion (replace the query by phantom documents).

2.3 Latent Semantic Analysis

Another way of approaching the data in IR is by using Latent Semantic Analysis [11]. The basic idea of this approach is to reduce the dimensionality of the data to get a more abstract interpretation. Such a high level interpretation of the data shows similarity to human interpretation of documents. Previously described methods have no further interpretation of the data than just the words that occur in the documents. The human interpretation of a given database would be very different. For example, we might group documents based on abstract concepts such as underlying topics. This is a very different similarity measure than counting word frequencies. In fact, for a human to group a document under a certain theme, the actual theme itself doesn't need to be included in the document. For example, a news article about the current presidential candidates in the US would be categorized by humans under 'politics', while the term 'politics' itself probably doesn't even occur in the document. Another article,

for example about the US's policy towards a foreign war (which would clearly also be labeled 'politics', even though it might not share many of the words from the previous document) should be associated with the same category. To achieve this, we need a semantic interpretation, and that is where LSA comes in.

2.3.1 Singular Value Decomposition

Singular Value Decomposition is a mathematical technique to decompose a matrix into the product of three other matrices: $M = U\Sigma V^*$. In this function, U and V are matrices with the so-called singular vectors. Σ is a matrix with only diagonal entries: the singular values. If M is reconstructed with a smaller number of singular values (the rest replaced by 0's), the result is a low-rank matrix approximation. The matrix retains its original dimensions, but is reconstructed from a lower dimensionality, a truncated version of itself. This way the values are stripped of noise, and provide a more generalized view of the data. Latent Semantic Analysis is the analysis of data on an implicit semantic level, which is achieved by reducing the dimensionality. It is also the basis for the approach of Latent Semantic Indexing.

2.3.2 Latent Semantic Indexing

Latent Semantic Indexing (LSI) is the application of the previously described method, LSA, in the context of information retrieval [2]. The gist of the method is to store the dimensionality-reduced data in the index of the search engine. In this section we discuss a list of possible advantages and disadvantages of LSI versus simple vector space search, according to [30] and [11].

2.3.3 Advantages

Latent Dimensions

The basis of LSI is that the lower dimensionality of the transformed data shows a truer representation of reality. The documents are a sample of empirical data, but won't contain perfect semantical information. This basis of true latent semantic information is obscured by the fact that each document contains a different set of words, and thus different dimensions. Because of this, the actual semantics can never be represented in such a noisy high-dimensional structure. The assumption is that LSI recovers this structure.

Synonimity

One of the most prominent advantages of LSI is that this search engine searches through (what are assumed to be) more informative features than the regular vector space model: concepts rather than terms. There are loads of terms in the index of a regular search engine, which more or less mean the same thing: they share a certain grade of synonimity. This synonimity is compressed into one abstract semantically relevant feature.

Polysemy

Another issue tackled with LSI is polysemy, multiple terms referring to one and the same semantical concept. If a query would contain multiple polysemous terms, the vector space model would be very noisy: a lot of documents which contain the terms are actually about something completely different! LSI tries to reduce this noise by averaging out polysemous terms. This is a disadvantage as well as an advantage. It reduces noise among polysemous terms, but it also drags them towards the average. So this makes polysemy work well if the meaning of the terms in the query are close to those terms' average meaning.

Interdependence

The vector space model assumes term independence. Each term is treated as a separate piece of information, in no way linked to related or similar information. This assumption never represents reality, as terms show very much dependence among documents. LSI does account for this high level of interdependency among words.

2.3.4 Disadvantages

Memory Efficiency

Intuitively one might think that this decreased dimensionality would lead to a reduced amount of memory needed to store the model. However, this is not true. Assuming that one would store only non-zero entries in a document-term matrix, this database is quite modest in size, since the matrices are very, very sparse. Using SVD to reduce these data to a lower dimensionality indeed reduces the size of the matrix. The problem is however that the density of the matrix also greatly increases: the probability of a zero value in the matrix is next to nothing. However, as previously discussed, in the current developments in

computer hardware, storage is seldom a problem. The problem lies mostly in computational efficiency.

Computational Efficiency

The most important way to increase the computational speed of the vector space model is by means of inverted indexing (see section 2.2.6). This is unfortunately not possible in LSI. Inverted indexing exploits the fact that the data consist of sparse matrices containing natural numbers. When the data are transformed using LSA, the index becomes a dense matrix of real numbers. This means that for every document a score has to be calculated.

2.4 Artificial Neural Networks

In this first section on neural networks we will give a short introduction to the traditional supervised neural network. In the next section we will discuss the type of neural network that is more relevant to understand for this research.

A traditional neural network is a machine learning algorithm with a wide variety of applications. It can solve non-linear classification problems, a type of problem very common in the real world. A neural network in AI (or rather: artificial neural network) is loosely inspired on the actual structure of neurons in the human brain.

2.4.1 Structure

The neural network consists of a number of neurons (sometimes referred to as 'nodes'). These neurons are topologically structured. Neurons have one or more weighted connections to other neurons. In this sense a neural network can be seen as a weighted graph. The network contains input neurons, output neurons and possibly any number of so-called 'hidden' neurons. These earn their name from the fact that the user generally only observes input and output, and everything in between is hidden (the so-called 'black box'). Each neuron has an activation level. Input neurons get their activation directly from the input data represented as a feature vector. Each input neuron represents and is fed by one feature of each data point. Each other neuron has an activation level based on the activation level of the neurons that have a weighted connection to it. The activation of the output neurons is the output of the system, which can be multi-dimensional, based on the number of output neurons.

2.4.2 Supervised Learning

In many applications neural networks are supervised learning algorithms. That is, they learn from a labeled dataset to build a model. This model is a mapping between inputs and their corresponding outputs. This means that there needs to be a training set to determine the weights between the neurons. This is a useful approach if the data contain labeled examples, and has a wide variety of applications, ranging from computer vision and robotics to simple classification tasks.

2.4.3 Applications

Neural networks come in lots of varieties [13]. There are supervised networks vs. unsupervised networks, feedforward networks vs. recurrent networks and so on. For this reason, applications of neural networks have been as broad as the use of artificial intelligence in general: ranging from medical applications (for example, predicting medical outcomes [46]) to economical applications (e.g. predicting bank failure [44]). Neural networks are a very popular solution for classification and prediction. Mainly because so many different types of neural networks exist, they are very flexible and form a fitting solution for a large variety of problems.

Some examples of applications in the area of language and information include text generation [42] and text classification [37]. Different incarnations of neural networks have already been used in the specific context of information retrieval, such as self-organizing maps [25] and Hopfield-networks [4].

2.5 Retricted Boltzmann Machines

2.5.1 Structure

In the context of this research, the most important type of neural network to examine is the Restricted Boltzmann Machine (RBM) [39]. The RBM differs from the previously described neural network in that it is an unsupervised probabilistic network. Instead of mapping an input to a desired output, this is an unsupervised algorithm: it learns what the inputs look like, but there is no specific output. The RBM is a generative model: once the inputs are learnt, it can generate data based on its learned interpretation of what the real data looks like. It can be viewed as an undirected graph where a node is only dependent

on the nodes it has a connection to. Such a network is called a Markov Random Field, and the 'neighbourhood' a node solely depends on is called the Markov Blanket of that node.

The Restricted Boltzmann Machine consists of two layers of nodes: one visible layer v and one hidden layer h. For the visible layer we use binary inputs. Each node is connected to all the other nodes in the other layer, but is connected to none of the nodes in the same layer. This is why this type of Boltzmann Machine is called 'Restricted'. Each of the edges in the graph represents a weight value. Besides that, every node has a unary bias. The reason for the restriction in the connectivity is that the training algorithm, explained later, is very inefficient for the fully connected Boltzmann Machine.

2.5.2 Training a Restricted Boltzmann Machine

This section is mainly based on [15].

The state of a Boltzmann Machine is represented by an energy function:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

Where a_i is the bias for visible neuron i, v_i is its binary state. Similarly, b_j and h_j represent respectively the bias and state of hidden neuron j. Finally w_{ij} is the weight between visible neuron i and hidden neuron j.

Based on this energy function the RBM assigns a probability to each possible combination of visible and invisible neurons:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}$$

Z is the partition function for normalization, the total energy summed over all visible/hidden neuron pairs. If we sum over all possible hidden vectors, we get the probability for a visible vector $\mathbf{v}$:

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$$

So to train the network we want to set the weights in such a manner that each training example that we put into the RBM has a high probability.

First let us look at the process of presenting an input vector $\mathbf{v}$ to the RBM. The probability that a hidden neuron j has the value 1 is given by the function:

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_i v_i w_{ij})$$

Where σ represents the sigmoid function. In the same way, if we have a distribution for the hidden units, we can symmetrically calculate a visible vector $\mathbf{v}$, due to the undirected nature of the graph.

$$p(v_j = 1|\mathbf{h}) = \sigma(a_i + \sum_j h_j w_{ij})$$

This back-projection then is called the reconstruction. The learning rule is based on this reconstruction:

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon})$$

In this function ϵ is the learning rate. This process can be repeated: feed the reconstruction back and let it recalculate the values of the hidden layer. This process of reconstructing the visible layer and recalculating the hidden layer can go on infinitely. This is called repeated Gibbs sampling. Ideally the reconstruction used for the formula above would be the reconstruction after infinite Gibbs samples. This is called maximum likelihood learning: it maximizes the likelihood that it produces vectors that are just like the training data. But since this is computationally very expensive, it suffices to do the Gibbs sampling just for one step. This is not the optimal training, but it suffices for most applications. This method of training the RBM is called contrastive divergence [16].

2.6 Deep Belief Networks

2.6.1 Deep Learning

Deep learning is a popular name for machine learning algorithms that search for high-level abstractions through stacked algorithms (in our case the most successful of the lot, neural networks) that step-by-step reduce the complexity of the data. In this way the data can be represented in a more essential set of features. In the specific case of information retrieval, this set of features would be considered 'semantical concepts': abstract notions of meaning, instead of plain words. It is clear that there is a parallel with the previously discussed

latent semantic analysis. Now we reach the actual heart of the matter: how do these forms of data-abstraction compare?

2.6.2 Structure

Deep neural networks (DNNs) are multiple neural networks, in this case Restricted Boltzmann Machines, linked together. This type of DNN is also called a *deep belief network* (DBN) [16]. The input enters the network in the first RBM. The output of the first RBM forms the input of the second, and so on until the actual output layer is reached.

These stacks of RBMs may vary in size: they can contain a few RBMs, but can also contain a large stack of RBMs. Furthermore, the amount of neurons in each layer may vary. Each layer may contain less than, equal to or more neurons than the previous. These structures are very flexible and can thus be applied in lots of different applications.

The nodes in an RBM layer can be viewed as dimensions. If we stack a number of RBMs and reduce the size of neurons towards a small number in the hidden layer of the final RBM, we achieve dimensionality reduction. So the DBNs can be viewed as a way to reduce dimensionality. Using this type of neural network structure, the deep belief network, to reduce the dimensionality of data has initially been used in [17]. The findings of that research were that in many different applications the DBNs outperformed traditional dimensionality reduction algorithms like Principal Component Analysis (elaborately explained in [19]).

The results presented in [17] were a direct motivation for this research. LSA shows many similarities to PCA, and thus it might be hypothesized that DBNs also might outperform LSA in the context of information retrieval.

2.6.3 Optimization

A DBN contains a lot of free parameters that have to be tuned. On the scale of the DBN itself there is of course the topological structure: how many layers are there, how many nodes per layer are there? On the scale of the RBM there are also parameters to tune, most importantly the learning rate (and optionally its decay as training progresses). Then the process of training can be varied in length to produce qualitatively different DBNs.

Optimization can be achieved through a grid search: automatically cross-test different settings for each of these parameters. This is a long process, but

eventually leads to the optimal DBN-configuration.

2.6.4 Applications

Deep neural networks have been around at least since 1980 [12]. But only when a way was found to effectively train these networks [16] and use them for dimensionality reduction [17] in 2006, the principle of deep learning gained momentum in the field of artificial intelligence. Since then, many applications have been found.

One of the main applications of deep neural networks in general is within the research area of image recognition and object detection (see [6, 5, 21, 7] among others). These studies use convolutional neural networks. Images, like text, also have the property of often being very high-dimensional, but the true meaning of the data is in abstractions of that complex, noisy data. The meaningful structures in this case are patterns or shapes that are relevant to recognize in the world, like traffic signs [5] or signs of a certain disease on medical scans [7]. We can draw an analogy with our textual 'search engine' data here: terms relate to pixels (sub-units), in the same way concepts relate to shapes (bigger structure). A very recent and notable effort of image analysis is [14], a very successful attempt by Microsoft to improve results on one of the biggest challenges and benchmarks in image analysis: the Large Scale Visual Recognition Challenge 2015 [32]. They produce the lowest error rate in object detection ever produced on this dataset using a 152-layer convolutional DNN.

Another interesting application of DNNs is speech recognition [10]. In this case the DNN would pre-process the data before they would be inserted in the Hidden Markov Model, which is the staple algorithm for speech recognition. Results seem to indicate that in this case the neural networks outperform the previous preprocessing methods (among which the most commonly used Gaussian Mixture Model).

Finally, even a little more closely related to this project, DNNs have been applied to natural language processing [8]. In this field, the aim is to represent natural language into a formal language, to be understood and manipulated by computers. An example of this is part-of-speech tagging: label words in a sentence as nouns, verbs etcetera. Natural language is (from a computational point of view) unnecessarily complex: a lot of words have little to no meaning. To filter out this 'noise', once again DNNs can be applied to reduce the dimensionality of the data.

Another application in the understanding of natural language was researched

[35]. The authors of this study compared DBNs to conventional methods of text classification. Common methods of text classification are Support Vector Machines [9] and Maximum Entropy classifiers [1], but DBNs could at least match their performance. As an extra boost for the system, the researchers added additional input data at one of the middle layers of their DNN. So where normally only abstract data would be found, they added fresh uncompressed training data. This improved the performance significantly. Using this method they were able to achieve higher performances than the 'conventional' classification tools they tested.

Chapter 3

Experiments

In this chapter the experimental procedures will be explained. We will discuss the data used, the queries used and the different search engines that are being compared.

3.1 Data

3.1.1 Dataset

For the experimental part of the research data from the 20 newsgroup[1] data is used. We use a subset[2] of the data to represent smaller datasets. In many, for example commercial, applications of IR this is a plausible amount of documents. Concrete examples of commercial databases of this size include web stores or news archives. This is data collected from Usenet newsgroups. These are discussion forums about a very broad variety of topics. The 20 newsgroup dataset, as the name suggests, contains data about 20 of these topics. These are the topics, more or less sorted by topical similarity:

```
comp.graphics
comp.os.ms-windows.misc
comp.sys.ibm.pc.hardware
comp.sys.mac.hardware
comp.windows.x
```

[1] Home Page for 20 Newsgroups Data Set, *http://qwone.com/ jason/20Newsgroups/*
[2] 20 Newsgroups, *http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html*

```
rec.autos
rec.motorcycles
rec.sport.baseball
rec.sport.hockey

sci.crypt
sci.electronics
sci.med
sci.space

misc.forsale

talk.politics.misc
talk.politics.guns
talk.politics.mideast

talk.religion.misc
alt.atheism
soc.religion.christian
```

As visible from this list, the dataset contains topics that are similar (`comp.sys.ibm.pc.hardware` and `comp.sys.mac.hardware`) and topics that are very diverse (`talk.politics.mideast` and `rec.motorcycles`). This makes it a very suitable dataset for this experiment: we want to be sure to test the different search engines on different levels of data abstraction (and thus different levels of topical similarity). The dataset was gathered for the study [22], but commonly used since then.

3.1.2 Preprocessing

Before any document is read from file, the system compiles a dictionary out of all the documents. This dictionary contains all the relevant terms in the database. That is, all the stemmed types, without any stop words. Words are split by spaces, and all the non-letter characters are removed. The stemming algorithm used is the Snowball implementation[3] of the Porter-stemmer. The stop words are filtered according to the Xpo6 stopword list[4]. Out of the resulting words, the

[3]Snowball, *http://snowball.tartarus.org/index.php*
[4]Xpo6 Stopword List, *http://xpo6.com/wp-content/uploads/2015/01/stop-word-list.csv*

2000 most common words are picked to fill the dictionary with. This leaves out irrelevant words, which adds noise to the data. Using only the most common words significantly improves efficiency during the training of the neural networks and during the performance of the singular value decomposition.

3.1.3 Representation

Each document is represented as a vector. Each element in the vector represents a term in the total dictionary that is extracted from the relevant dataset. We can distinguish between two types of data representation used.

1. **Numerical** Each document is represented as a vector containing numerical values. Each element of the vector represents the amount of times the respective term occurs in the document. This representation is used for the vector space search, the LSI search, the autoassociator and the phantom document-DBN search (explained later on).

2. **Binary** Each document is represented as a vector containing binary values. Each element of the vector represents whether or not the respective term occurs in the document. This representation is used for the DBN-search. The reason for this representation is that it can serve as input for the first layer of the first RBM of the DBN, which, in this research, takes binary data (see section 2.5.1).

3.2 Queries

The experimental queries will vary in specificity: the algorithms should return a lot of relevant data when we search for a very broad term (for example, `science`). However, a very specific query (for example `Honda ST1100`, a specific type of motorcycle) should yield documents about that topic only. Hence the query-answer set contains queries and the subset of relevant documents. These relevant documents can be documents in a certain newsgroup category (broad) or a manually grounded (annotated) selection of the documents (narrow). For a comprehensive overview of all the tested queries, see the appendix. These two types of tests will from now on be dubbed 'broad' queries and 'narrow' queries.

3.2.1 Broad Queries

Examples of broad queries are queries that are relevant to an entire newsgroup. For example, there is the newsgroup `rec.motorcycles`. Each newsgroup in our dataset contains 100 documents. When we query the system for *motorcycle* ideally we would want those 100 documents to be in the top 100 search results. Other examples include *atheism*, *hockey*, *space* and *graphics*, which all are representative for the documents in one newsgroup.

3.2.2 Narrow Queries

The narrow queries are only relevant to 2 - 12 documents. The only correct results associated with these queries are these documents. Search scores will be a lot lower, because it is a lot harder to get a top 2 correct than a sizeable portion of 'broad' top 100. Since both could be important in finding desired information, both types of tests will be considered.

3.2.3 Relevance

The relevance of the queries has been determined in two different ways. For the broad queries, we assume that each query tested represents an entire newsgroup category. For example, for the query *space* the relevant results are every document within the category `sci.space`. Each of the 20 newsgroups has such a query associated with it in the broad query tests: *cryptography* for category `sci.crypt`, *hockey* for category `rec.sport.hockey` and so on.

For the narrow experiments the queries were selected manually. To create a set of queries that would be relevant to only a small number of documents, the conversational nature of the dataset was used. The newsgroups are discussion forums where discussions take place in topics. So for each conversation topic, a few documents exist in the dataset. For example, there is a topic where motorcycle enthuasist discuss wheelies on a shaft driven motorcycle, under the topic of `Subject: Re: Shaft-drives and Wheelies`. Nowhere else in the dataset are people talking about wheelies, so the query *wheelie* is associated with the set of documents in this particular relevant discussion. The set of queries was created in such a way that for each query there are some, but not a lot of relevant documents. Another restraint was that each query had to be in the 2000 word dictionary used to represent the data.

3.3 Types of Search Engines

3.3.1 Vector Space Search

The vector space model, as explained in section 2.2.6, will serve as the baseline. It is the most direct approach of information retrieval we will consider. It applies no dimensionality reduction, just straightforward query-document cosine similarity.

3.3.2 LSI Search

When the data have been read and preprocessed LSA is performed. When the search engine receives a query from the user, or any test environment, it folds it into the SVD of the training data. Then the search score for each document is calculated using the cosine similarity between the lower-dimensional version of the query and the lower-dimensional version of each document

Efficiency

In LSI the reduced vectors contain real values. The problem with having real valued number in the index, is that inverted indexing is not possible. Inverted indexing profits from the sparseness of the normal term vectors of each document. The reduced vectors are not sparse, so inverted indexing cannot be used. This makes search in large databases considerably slower.

3.3.3 Deep Belief Network Search

Data Representation

As mentioned before, the DBN trains on the newsgroup dataset. Once this model is calculated (or loaded from file, to prevent having to go through the training phase every run) each document (in binary representation) in the set is given a forward pass through the RBMs in the DBN. Then the query is passed through the network. Now we have low-dimensional representations for both the query and all of the documents. The scores for each document are then calculated based on the cosine-similarity between the low-dimensional representations of the query and that of the respective document.

Efficiency

Using DBNs as search engines has a few implications regarding efficiency. To be as quick as possible in returning the search results, the deep vectors of all the documents in the dataset will have to be calculated beforehand. This prevents long calculation times. Each query needs all deep vectors, so it would be very impractical to do a forward pass on each document for each vector. Similar to the efficiency issue the LSI has, as discussed above, DBNs cannot use inverted indexing to improve efficiency because there are real valued numbers in the index. There is no sparseness to profit from.

Dropout Training

A training technique to increase the performance of any neural network is the dropout-method [18, 41]. A problem with training neural networks on small datasets is that some features will only occur in the presence of other features. For example, in a small dataset the word *car* might hypothetically appear only in the context of the word *convertible*. Because these words only appear together in a document, this will mean that they are useless by themselves. This is purely the product of a lack of enough data to represent semantic reality. On themselves the words also have meaning, but the network will not be trained on a document with just one of these two words. To account for this, the network can be trained where in each training sample a portion of the features is left out: the dropout. This way the network will also see the documents with just either *car* or *convertible* and learn that without each other they have meaning.

3.3.4 Autoassociator

Another way of reducing the dimensionality of data with a neural network is called the autoassociator [3]. This is not a restricted Boltzmann machine, but a more common type of neural network: the multi-layer perceptron [31]. In most situations, the multi-layer perceptron is a supervised machine learning method. This means that the data on which it is trained need to be labelled in advance. However, in the case of an autoassociator, we use the perceptron for dimensionality reduction. In this case, the input and output layer have the same size. The output represents a reconstruction of the input. The hidden layer contains less neurons than the input and output layers. This is where the data are represented in a low-dimensional space. The search score for a document

in this system is calculated using the similarity of the hidden representations of each document and the user query.

The autoassociators training depends on the error of the reconstructed data: therefore the data should be normalized. Otherwise the training vectors with higher summed inputs will produce larger errors. That will make these samples weight heavier in the training. That is not desirable. To solve this, we make sure every input vector is preprocessed to have a sum of 1.

3.3.5 Phantom Document Masking

Phantom document masking is an alternative condition that could be applied to any of the search engines. It is a form of query expansion.

In this search case the same engines are used as in the regular search experiments. However, there is an extra step. First we use the query with some (another or the same) search engine. The top result of that search is then returned, and used as input to the search. This way the hope is to substitute the query by the document that is its best representation in the dataset. The document is richer, less sparse with more (hopefully relevant) features than the original query. We call this document a phantom document.

Variations to this approach are to take more than just one phantom document. We could take any number of top results. The more documents included in the query, the more stable the representation of the query in the dataset should be: the query is less dependent of the results generated by an entirely different search engine. However, when we make the amount of phantom documents too high, we lose specificity. We lose the ability to search very specifically for one keyword. This is a trade-off between robustness and search precision that will be researched by comparing results of various sized sets of phantom documents.

3.4 Search Engine Parameters

The algorithms that are at the heart of the different search engines contain a lot of free parameters. That means that they require a lot of fine tuning. These parameters are very dependent on each other: changing one means that the optimal value of the other might very well shift. This means that many different configurations need to be tested. Test results will thus contain various parameter settings.

3.4.1 Parameters for LSI

The main parameter to be experimented with for the LSI-search engine is of course the number of dimensions after reduction. Literature states that the optimal dimensionality after reduction lies somewhere between 70 and 300 dimensions [30, 20]. This will be closely examined.

3.4.2 Parameters for the Deep Belief Network

Topology

A DBN has many parameters to optimize. Similar to LSI, one very important variable is of course the dimensionality. But for the neural network this is more complex because not just the final layer of neurons is relevant for the success of the network, but also the topology of the network. The amount of layers, and how many neurons there are in each layer could affect the quality of the dimensionality reduction.

Some interesting variations in this area could concern the amount of layers, but also the amounts of nodes per layer. First simple two-layer nets will be tested. They are the most straightforward networks. In these experiments a very broad range of different numbers of hidden neurons will be tested, ranging from about 50 to 1000. This is the DBN equivalent of the amount of LSI dimensions, since the amount of hidden neurons will essentially be the dimensionality of the data.

Then different amounts and sizes of hidden layers will be tested. When we look at the DBN used for text classification in [35], we see three hidden layers of 500 units. Although the applications are not the same: they use only a few class label units as output, whereas in this experiment we use an arbitrary number of abstract dimensions as smallest layer in the network. However, it could be worth testing this successful topology for our purposes.

Training Examples

The performance of the neural network will be very dependent on the amount of training examples it will see during training. Using the experiments we find out how often the network needs to see all the data before it can successfully function as the basis for a search engine.

After a certain amount of training the network converges. Up to this point the performance is supposed to increase. Beyond this point training starts to

have very small effects (or no effect at all) on the performance of the network. For each network topology and parameter configuration convergence might be reached after a different number of training iterations.

Epsilon

A parameter that's known to greatly influence the training of a neural network is its learning rate, commonly denoted as ϵ. This is considered the most important parameter for training an RBM [36]. Choosing a good value for ϵ is vital for the process of training a DBN. It has a great impact on the final performance of the network. Different values of ϵ will be tried out.

3.5 Measuring Quality

Common measures in IR are for example precision, recall and F-score. However, this is based on a search engine that returns a set of data as being relevant. In this returned set of documents the precision and recall can be calculated. However, in these experiments ranked results are returned. That means all the documents in the entire database will be returned, ordered by relevance score. For ranked results a different method is needed that does not rely on a binary relevance assessment.

To measure the quality of an instantiation of one of the search engines described in this chapter the mean average precision (MAP) is measured. This is a common measure for the quality of ranked retrieval. Each tested query yields an average precision. The MAP is the mean average precision over all the queries.

To calculate the average precision, each relevant document will be assigned a precision value. This value is calculated according to the precision of the subset of documents up to the rank of that document. When the precision is calculated for each relevant document, this value is averaged over all the relevant documents. The mean average precision is the mean of the calculated values over all the queries.

For example, we consider a query which has three relevant results. The three relevant documents are ranked as first, third and fifth search result. The precision for the first relevant document is 1 out of 1: $\frac{1}{1} = 1$. For the second relevant document (with rank 3) the precision is 2 out of 3: $\frac{2}{3}$. For the third relevant document (with rank 5) the precision is 3 out of 5 documents: $\frac{3}{5}$. The average precision of these results for this query is $(1 + \frac{2}{3} + \frac{3}{5})/3$. The mean of all

the average precisions of all the queries yields the mean average precision, our measure of quality.

Chapter 4

Results & Discussion

This chapter will contain the results of the experiments. For many experiments there is a broad and a narrow variant, which concerns the specificity of the test queries. This is done to ensure that the experiments test both the results for mean average precision about general topics (broad), as well as finding specific documents in the top search results (narrow).

4.1 Latent Semantic Indexing Dimensions

First, results of possible varying dimensionality reductions of the decomposition in LSI will be inspected. This determines the level of abstractness in the reduced representations of the data in the LSI search. According to literature the optimal value should lie between 70 and 300 dimensions [30, 20]. To assert the truth of this claim a wider variety of values will be tested. Results are shown in figure 4.1.

4.2 Phantom Documents

First we will examine the potential added value of the phantom document queries. We will consider this in a broad perspective as explained in the previous chapter, as well as in the narrow ones. Experiments with 1, 2 and 5 phantom documents were performed. The phantom documents were concatenated to form a mask that represents the query. These concatenated phantom documents were entered into the search engine instead of the query itself. The phantom documents were collected using a simple vector space search, under the assumption

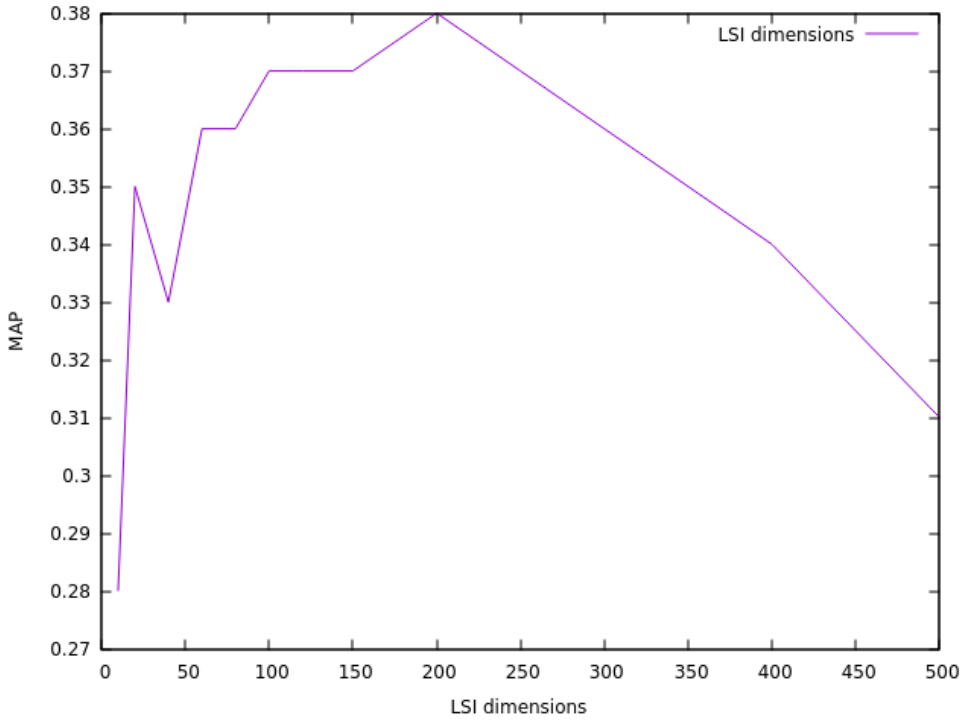

Figure 4.1: *The influence of the number of dimensions kept in the LSI on the percentage of correct answers in the broad experiment. The best results are obtained with 150 - 200 dimensions.*

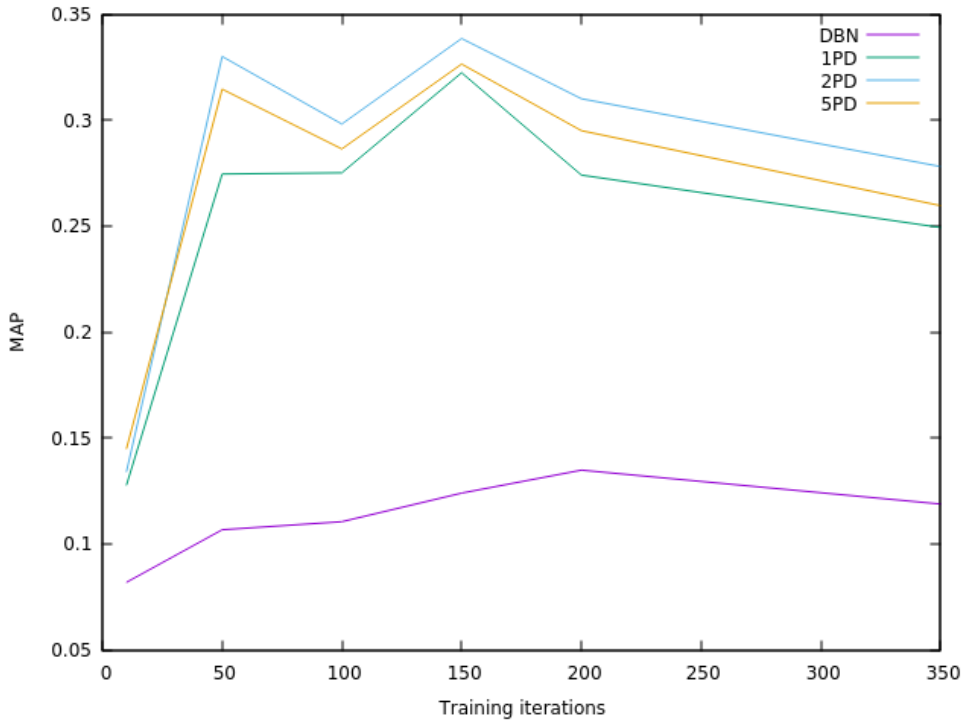

Figure 4.2: *Results of the broad query tests with the DBN search engine. This figure shows the influence of the amount of training iterations on the performance of the search engine. The best results are acquired around 150 iterations. DBN = deep belief network/restricted Boltzmann machine, 1PD = DBN with one phantom document, 2PD = DBN with two phantom documents, 5PD = DBN with five phantom documents.*

that the top results of vector space search would be a fair substitution for the query.

4.3 Training Iterations

4.3.1 Broad Queries

In this experiment we zoom in on the influence of the amount of training iterations. That is, how many times does the network see each document during the training phase. The most important question in this experiment is: up to which point does it make sense to increase the amount of iterations? The results are

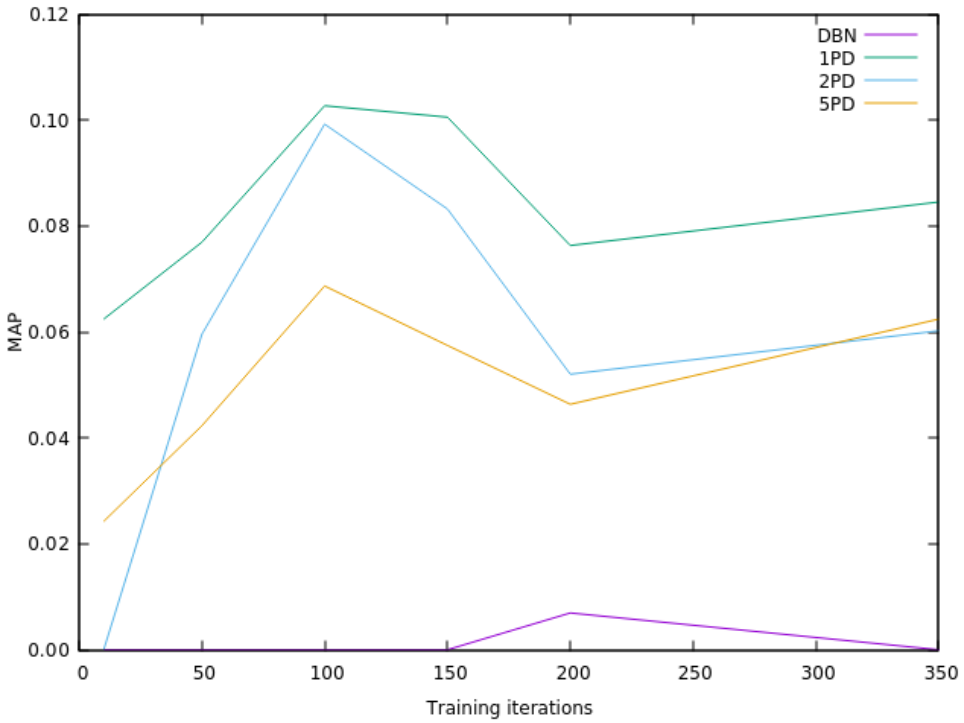

Figure 4.3: *Results of the narrow query tests with the DBN search engine. This figure shows the influence of the amount of training iterations on the performance of the search engine. The best results are acquired between 150 and 250 iterations, depending on the amount of phantom documents. DBN = deep belief network/restricted Boltzmann machine, 1PD = DBN with one phantom document, 2PD = DBN with two phantom documents, 5PD = DBN with five phantom documents.*

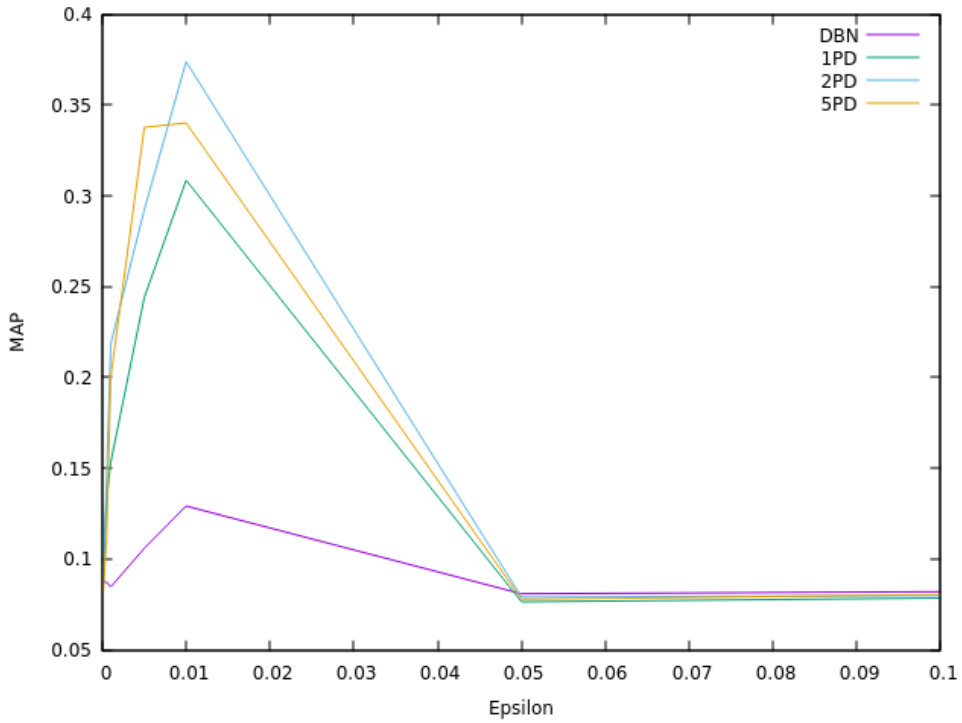

Figure 4.4: *Results of the tests with the DBN search engine. This figure shows the influence of the learining rate ϵ on the performance of the search results. We clearly see the best ϵ-value within the scope of the tests at 0.01. DBN = deep belief network/restricted Boltzmann machine, 1PD = DBN with one phantom document, 2PD = DBN with two phantom documents, 5PD = DBN with five phantom documents.*

shown in figure 4.2. This figure shows the mean average precision influenced by the amount of training iterations. One training iteration means that the network has been trained on all of the documents exactly one time. The figure also shows different amounts of phantom document policies. There are four lines in the figure: one for the results of the deep belief network without any phantom documents (DBN), one for the results using a single phantom document (1PD), one for the results with two phantom documents (2PD) and one for the results using five phantom documents (5PD)

4.3.2 Narrow Queries

To confirm that also in the narrow experiment the neural network reached convergence the influence of the amount of iterations was also tested, shown in figure 4.3.

4.4 Restricted Boltzmann Machine Parameters

The first thing to find out about the RBM is how the parameters should be configured. The main ones being the learning rate (ϵ) and the amount of training. The results of this experiment are displayed in figure 4.4. We see that the best tested value, under any tested phantom document-condition, is 0.01.

4.5 Deep Learning

Deep belief networks come in many different shapes and sizes: from shallow single RBMs to many layered stacks of RBMs. In this section we will discuss experiments that compare different topologies. Deep learning occurs when multiple RBMs are stacked. Here we investigate whether this extra abstraction of the data improves search performance.

4.5.1 Triple Layer DBN

The first contained two RBMs, with the following layer configuration: $\{2000, 500, 45\}$, meaning that there were 2000 input nodes, 500 hidden nodes for the first RBM and visible nodes for the second, and 45 hidden nodes for the second RBM. The

best result found here was 0.291. This result was found using a learning rate of 0.01 and a maximum number of training iterations of 150.

4.5.2 Quadruple Layer DBN

The second deep network tested had four layers: $\{2000, 500, 500, 45\}$. The best MAP found in this case was 0.1152. Unfortunately we see the results declining when more layers are added. The limited size of the dataset might account for this. Deep learning is aimed at high level abstract patterns that might not be visible in a small dataset like the 20 newsgroups data.

4.5.3 Dropout Training

The networks have also been trained using dropout training, as explained in section 3.3.3. In the experiments, the dropouts were 20% of the features, randomly selected. This amount of dropout has proven successful in previous studies [18]. However, no improvements or decreases in search engine performance were measured using this technique.

4.6 Comparison of Search Algorithms

In this experiment we compare the best results of the different search algorithms. The tested search algorithms are: vector space search (VSS), latent semantic indexing (LSI) and deep belief network (DBN). For each search engine, the table also shows the best result obtained using one or more phantom documents as query. Out of all the experiments the best parameter settings are collected and compared among each other. For each of the search engines, they show the highest results obtained during the process of testing different parameter configurations. The tables also show the relevant parameters. For LSI the relevant parameter is the number of dimensions kept in the SVD. For the deep belief networks, the relevant paramters are the amount of hidden units, the amount of training the network has had, and the learning rate ϵ. For each trial where phantom document querying was applied, the amount of phantom documents used is also shown. The best score of each algorithm is the highest mean average precision it achieves. Tables 4.1 and 4.2 show the results.

Search engine	Parameter settings	Best MAP score measured
VSS	-	0.326
PD-VSS	1 PD	0.245
LSI	200 dimensions	0.386
PD-LSI	1 PD, 200 dimensions	0.352
Autoassociator	100 dimensions	0.091
DBN	45 hid. un., 150 it., $\epsilon = 0.01$	0.138
PD-DBN	2 PD, 45 hid. un., 100 it., $\epsilon = 0.01$	0.374

Table 4.1: *Top results on the broad query experiment. For each type of search engine this table shows the highest obtained result on the broad query experiment. VSS = vector space search, LSI = latent semantic analysis, DBN = deep belief network/restricted Boltzmann machine, PD = phantom document, hid. un. = hidden units, it. = amount of training iterations.*

Search engine	Parameter settings	Best MAP score measured
VSS	-	0.327
PD-VSS	1 PD	0.112
LSI	200 dimensions	0.188
PD-LSI	1 PD, 200 dimensions	0.129
DBN	45 hid. un., 100 it., $\epsilon = 0.01$	0.006
PD-DBN	2PD, 45 hid. un., 100 it., $\epsilon = 0.01$	0.096

Table 4.2: *Top results on the narrow query experiment. For each type of search engine this table shows the highest obtained result on the narrow query experiment. VSS = vector space search, LSI = latent semantic analysis, DBN = deep belief network/restricted Boltzmann machine, PD = phantom document, hid. un. = hidden units, it. = amount of training iterations.*

Chapter 5

Discussion

In this chapter the results will be discussed. The different experiments will be handled in the same order as in the previous chapter.

5.1 LSI Dimensions

In this experiment the optimal number of LSI dimensions kept during the SVD was tested. The number of dimensions was varied, and then performance was tested in the broad experiment.

The claim that 70 to 300 LSI dimensions would yield the best results seems to be quite plausible (fig. 4.1). However, the higher scores are around 150 to 200, and not 70. Raising the number even higher yields slightly worse results. This is a result that is very dependent on the data [20]. Each different data set has its own optimal number of LSI dimensions. This result is thus only for reference in this research rather than that it is generalizable to a broader perspective. The optimal number of dimensions is dependent on the number of documents in the data and the distribution of features among the documents.

5.2 Autoassociator

Using an autoassociator as a means to reduce the dimensionality of data has not proven to be an effective method to improve search results.

5.3 Deep Belief Network Parameters

5.3.1 Learning Rate

The influence of the learning rate ϵ is shown in figure 4.4. The results are obvious: the best option among the tested values is 0.01. Any of the experiments conducted yielded the best scores with this value.

5.3.2 Training Iterations

Broad Queries

The influence of the amount of training is shown in figure 4.2 for the broad experiments. In this experiment we can see that the best duration of the training phase is about 150 iterations (i.e. the amount of times the network has seen all the training samples). Beyond that the quality of the search slowly declines. Possibly the results decline because the network is overfitting: it only trains on the specific combinations of features that are in the training set. The query itself might not be exactly like one of these trained documents, and an overfitted network might have trouble generalizing and matching the similarity of the query to the documents.

Narrow Queries

The results for this experiment are shown in figure 4.3. In these experiments we see that the value with the best results lies a bit higher. This might be explained due to the fact that there has to be a bit more specificity in the network: the global meaning of the documents is a lot less important, the desired documents are rather specific in their content.

5.4 Phantom Documents for the Deep Belief Network

The influence of the phantom documents on the quality of DBN searches is shown in the same figures: 4.2 and 4.3.

5.4.1 Broad Queries

We find that using phantom document masks does certainly improve the quality of the search results. In comparison to the bare queries inserted in the search DBN, the phantom documents yield much higher scores. One other clear finding is that we see that the amount of documents used to form a phantom query is not too important here. The different amounts of phantom documents used to expand the query with show similar results. Also note that, despite the absolute difference, the relative performance for the amount of phantom documents is similar for any number of training iterations. The training time is irrelevant for the added value of the use of phantom documents.

5.4.2 Narrow Queries

Without phantom documents we see that the neural network search yields very low MAPs. Also in figure 4.3 showing the results of the narrow experiments, we see the relative success of phantom documents. All the variants show better results than just the DBN as search engine. We also see the trend here that few training iterations yield low success rates, but somewhere between 200 and 300 iterations lies the value with the best highest mean average precision.

5.4.3 Analysis

The increase in performance when using phantom documents can be explained by sparsity. The documents all contain more words that a single worded query. So the network is not trained on data similar to the query. This leads to low similarity between the low dimensional representations of each document and the query. If all the similarities are low, faulty results are more likely to occur. The phantom document solves this: in this case the query contains many useful features (in its original representation, but also in the low dimensional representation) to match with the documents. The phantom documents are much more similar (or, in the case of a single phantom document, even identical) to the samples in the training data.

5.5 Comparison of Search Algorithms

All the best results that were found during the experiments are gathered in table 4.1 and table 4.2. Each different result was obtained using a specific parameter

configuration. This configuration is also included in the tables. As shown in previously discussed experiments, these parameter settings influence the results a lot.

5.5.1 Broad Queries

In table 4.1 we see the best results obtained with each search engine for the broad experiments. We see the scores of the phantom document DBN search engine achieve nearly the same results as the LSI search. These two outperform the other search engines slightly, but only by a verys small margin. The phantom document variant of the LSI search comes very close after.

5.5.2 Narrow Queries

In table 4.2 we see that none of the DBN search engines can compete with the conventional methods. The results are very low. Since all the results are relatively low, we can consider two explanations. Perhaps the queries were too specific. That means that they never found any relevant representation in the low dimensional feature space. In this case the noise in natural language would obfuscate these tiny bits of semantic information. Another explanation is that perhaps the documents selected in the process of annotation were not relevant enough. Either way, we could still consider these results because all the search engines used the same queries with the same relevant target documents. Within the scope of the comparison in this thesis the results all suffer from the same defect. However, the broad query experiments yielded much more insightful results than the narrow queries.

Chapter 6

Conclusion and Future Work

6.1 Different Types of Search Engines

It seems that DBNs do not meet their expectations on a data set of the size used in these experiments. Further research is needed to determine the effectiveness of dimensionality reduction by DBNs as a means of building a search index in the case of a considerably bigger database. Deep learning is generally considered most effective when the data set is very large. There are very many parameters that need to be trained in case of a very big DBN. That is the reason that a lot of data is needed to train the networks on. One conclusion we can draw from the results of these experiments is that 2.000 training examples is not enough to increase the quality of a search engine based on DBNs. The DBN search engine does not outperform any of the more basic search engines.

6.2 Analysis

To account for the difference in performance among the two compared algorithms LSI and RBM we need to look at the internal, low dimensional representation of a one word query. The internal representation of a query in LSI looks very different from the internal representation in the neural networks. These internal representations by the LSI and the RBM are respectively shown in figure 6.1 and figure 6.2. In the case of LSI, there are values very close to 0. If we take the example of the query *cryptography*, we see values taking on values between 0 and 1, with quite a lot of variation. The information is packaged very subtly in informative and sensitive features. When we look at the internal representation

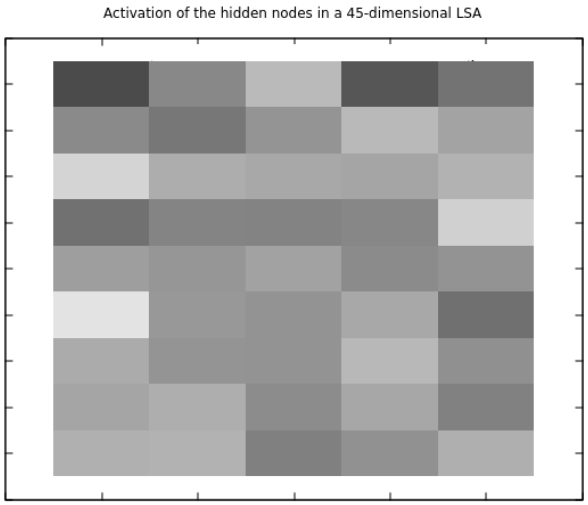

Figure 6.1: *A visualisation of an example of an internal vector representation in LSI. This is the query 'cryptography' folded in a 45-dimensional SVD.*

Figure 6.2: *A visualisation of an example of an internal representation in the RBM. This is the query 'cryptography' passed forward to the hidden layer of a 45-dimensional RBM.*

in the neural network we see values ranging from nearly 0 to nearly 1, since the vectors are made up of probabilities. The striking thing about these features is that many of them are actually nearly 0 or 1. This means that they are (nearly) being treated as the binary values that an RBM uses. The information density is much lower when we consider the documents in practically binary features. We see that the internal representation in the RBM is very black-or-white. The values are almost binary values. The values in the internal SVD representation are shades of gray, which contain a lot more information. This shows a weakness in using the RBMs to represent documents in the context of information retrieval.

6.3 Phantom Documents

When considering performance we see that the DBN search really benefits from phantom documents. However none of the other search engines benefit from this trick. The cause of this difference could lie in the fact that the DBN has binary representations under the hood. That means that small differences in the input might not make any differences further on in the RBM. So a one-word document (the query) has no relevant features active in the deeper layers. It has more resemblance to an empty document, than to any document in the dataset. This could be why the bare DBN scores very bad in either of these experiments. Changing the query to the contents of an actual document make it possible to do a proper assessment of the similarity between it and the other documents. This also explains why the other search engines don't benefit from it: both a vector space search and an LSI search can properly calculate the similarity between a one-word query and a document. In further research concerning the use of DBNs for indexing search engines using a bigger dataset, adding phantom documents would be something to examine.

6.4 Autoassociator

The autoassociator is a common 'shallow' way of reducing the dimensionality of data: there is no deep learning applied. The underlying idea for including this method in the experiments is that we use only a small dataset. Shallow methods are more direct in using the data, and could work better when the search database is of a relatively limited size. However, in the experiments, using the hidden layer of the autoassociator has not shown to be a useful addition

to the search engine. It scores inferior to the other dimensionality reduction methods, DBNs and LSI.

6.5 Deep Learning

On the topic of deep learning we can draw a clear conclusion: with a database this size, 2000 documents, we cannot plausibly apply deep learning. The results of the multi-layer DBNs show very poor results compared to the single layer variants. Successful application of deep learning requires very much training data. So for web search this could be an option, but for relatively small search engines, deep learning will not be of any added value.

6.6 Deep Learning in Natural Language Processing

One explanation for the lack of fruitful results in this thesis could be that deep learning might not be as suited for NLP related problems as it is for, for example, computer vision or genome data. Despite language being very rich in features (there are many different words in any language) it is a very different problem. Contrary to other applications, in language features can be already very descriptive of reality. If the word *motorcycle* appears in a document, that feature alone is already quite informative as to the contents of the document. In the case of computer vision, a single red pixel is not at all informative in relation to the object shown. In the case of genome data, a single base-pair is not informative in relation to the higher structure of the gene. This effect might account for the fact that more than one layer of neural networks (or, in the case of LSI, a single SVD) is enough to capture abstract concepts of language.

6.7 Future Work

The results of the DBN search engine do not promise to be an improvement over conventional methods of search. However, there are some questions that this research leaves unanswered, that could lead to future research. The first interesting follow up question would be how the DBN search would perform on a huge database. For sitesearch solutions the DBN does not show to be useful, but

for searching the internet, billions and billions of indexed pages, deep learning might prove to be a more powerful tool.

The second subject of recommended future research is that the autoassociator might prove useful still. It was not the focus of this research, and thus did not see a lot of experimental exploration. However, there are many parameters, such as the activation function and learning rate that could be varied to potentially be a success. The autoassociator is a shallow method for dimensionality reduction, and should not be bothered by the relatively small size of the dataset.

Appendix

A Broad Experiment Queries

Query	Accepted newsgroups
baseball	rec.sport.baseball
motorcycle	rec.motorcycles
electronics	sci.electronics
hockey	rec.sport.hockey
windows	comp.os.ms-windows.misc comp.windows.x
atheism	alt.atheism
graphics	comp.graphics
ibm	comp.sys.ibm.pc.hardware
auto	rec.autos
mac	comp.sys.mac.hardware
cryptography	sci.crypt
medicine	sci.med
space	sci.space
christianity	soc.religion.christian
guns	talk.politics.guns
religion	talk.religion.misc soc.religion.christian
politics	talk.politics.misc talk.politics.mideast talk.politics.guns
sports	rec.sport.baseball rec.sport.hockey
computer	comp.os.ms-windows.misc comp.windows.x comp.graphics comp.sys.ibm.pc.hardware comp.sys.mac.hardware
sale	misc.forsale

B Narrow Experiment Queries

Query	Accepted documents
evolution	"51139", "53061", "53433", "53610", "83599"
rendering	"37916", "38674", "38788", "38843", "38853"
alarm	"76441", "102914", "103667", "103723"
wheelie	"104387", "104639", "104752", "104217", "105220", "105249"
apple	"52155", "52190", "52214", "52248", "68012"
prediction	"102606", "102649", "104385", "105039", "53540", "59154", "20491", "53564"
password	"15413", "15482", "15868", "54010"
mask	"53862", "53961", "38375", "54010"
surgery	"59424", "59456"
placebo	"58951", "59076", "59579"
moon	"60821", "60950", "60976", "60995", "61066", "61087", "61118", "61253", "61272", "61318", "61484"
radar	"52794", "53872", "54041", "54248", "76117", "76795", "102733", "102852", "103141", "103497", "103680", "61253"

Bibliography

[1] A. L. Berger, V. J. D. Pietra, and S. A. D. Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71, 1996.

[2] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM review*, 37(4):573–595, 1995.

[3] M. Bianchini, P. Frasconi, and M. Gori. Learning in multilayered networks used as autoassociators. *Neural Networks, IEEE Transactions on*, 6(2):512–515, 1995.

[4] H. Chen. Machine learning for information retrieval: neural networks, symbolic learning, and genetic algorithms. *Journal of the American Society for Information Science*, 46(3):194, 1995.

[5] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

[6] D. Cireşan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[7] D. C. Cireşan, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI*, pages 411–418. Springer, 2013.

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

[9] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[10] G. E. Dahl, D. Yu, L. Deng, and A. Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

[11] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the Association for Information Science and Technology*, 41(6):391–407, 1990.

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

[13] K. Gurney. *An introduction to neural networks*. CRC press, 1997.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *ArXiv e-prints*, Dec. 2015.

[15] G. Hinton. A practical guide to training restricted Boltzmann machines. *Momentum*, 9(1):926, 2010.

[16] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

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

[18] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[19] I. Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[20] A. Kontostathis. Essential dimensions of latent semantic indexing (LSI). In *Hawaii International Conference on System Sciences*, page 73. IEEE, 2007.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[22] K. Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the 12th international conference on machine learning*, pages 331–339, 1995.

[23] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[24] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[25] X. Lin, D. Soergel, and G. Marchionini. A self-organizing semantic map for information retrieval. In *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 262–269, 1991.

[26] H. P. Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1(4):309–317, 1957.

[27] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[28] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 66, November 1999.

[29] J. J. Rocchio. Relevance feedback in information retrieval. *The SMART system-experiments in automatic document processing*, pages 313–323, 1971.

[30] B. Rosario. Latent semantic indexing: An overview. *Techn. rep. INFOSYS*, 240, 2000.

[31] F. Rosenblatt. *Perceptrons and the theory of brain mechanisms*. Spartan Books, 1962.

[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[33] G. Salton and C.-S. Yang. On the specification of term values in automatic indexing. *Journal of documentation*, 29(4):351–372, 1973.

[34] M. Sanderson and W. B. Croft. The history of information retrieval research. *Proceedings of the IEEE*, 100(Special Centennial Issue):1444–1451, 2012.

[35] R. Sarikaya, G. E. Hinton, and A. Deoras. Application of deep belief networks for natural language understanding. *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, 22(4):778–784, 2014.

[36] H. Schulz, A. Müller, and S. Behnke. Investigating convergence of restricted Boltzmann machine learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[37] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[38] T. Seymour, D. Frantsvog, and S. Kumar. History of search engines. *International Journal of Management & Information Systems (IJMIS)*, 15(4):47–58, 2011.

[39] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. *DTIC Document*, 1986.

[40] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[42] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[43] P. Switzer. Vector images in document retrieval. *Statistical association methods for mechanized documentation*, pages 163–171, 1965.

[44] K. Y. Tam and M. Y. Kiang. Managerial applications of neural networks: the case of bank failure predictions. *Management science*, 38(7):926–947, 1992.

[45] M. Taube, C. Gull, and I. S. Wachtel. Unit terms in coordinate indexing. *American documentation*, 3(4):213–218, 1952.

[46] J. V. Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology*, 49(11):1225–1231, 1996.