IMPROVING INFORMATION RETRIEVAL USING A MORPHOLOGICAL NEURAL NETWORK MODEL

By

CHRISTIAN ROBERSON

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2008

To my parents and my girlfriend Ellanna

ACKNOWLEDGMENTS

First I thank my supervisory committee chair Douglas D. Dankel II. His patience and support made this research possible. I also thank Rory De Simone for letting me teach for her all these years. Her encouragement and support led me to continue beyond the undergraduate level. I also thank my parents, whose amazingly reliable safety net allowed me to focus solely on school and avoid many headaches. In addition I thank Charles and Jeff for putting up with all my complaining and keeping me on task. Lastly, and most of all I thank my girlfriend Ellanna. Her love, support, and patience have not wavered in this lengthy process. She has undoubtedly been the single most integral component of my success, and without her I never would have made it to graduation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure page

LIST OF ABBREVIATIONS

ANN	Artificial neural network
DNF	Disjunctive normal form
IDF	Inverse document frequency
IR	Information retrieval
LSI	Latent semantic indexing
MLP	Multi-layer perceptron
MNN	Morphological neural network
MNNIR	Morphological neural network information retrieval
RSV	Retrieval status value
SLMP	Single-layer morphological perceptron
SLP	Single-layer perceptron
SOM	Self-organizing map
SVD	Singular value decomposition
TF	Term frequency
TREC	Text retrieval and evaluation conference

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

IMPROVING INFORMATION RETRIEVAL USING A MORPHOLOGICAL NEURAL NETWORK MODEL

By

Christian Roberson

May 2008

Chair: Douglas D. Dankel II
Major: Computer Engineering

We investigated the use of a morphological neural network to improve the performance of information retrieval systems. A morphological neural network is a neural network based on lattice algebra that is capable of solving decision boundary problems. The morphological neural network structure is one that theoretically can be easily applied to information retrieval. Morphological neural networks compare favorably to other proven models for information retrieval both in terms of speed and precision.

CHAPTER 1
INTRODUCTION

Information retrieval (IR) is the science of searching for information in documents including web pages, digital books, images, databases, or any other kind of a growing number of sources of information. To search a document collection, users submit a query, typically a list of keywords, to an IR system and receive a list of documents that may be relevant, typically those documents containing the keywords. However, obtaining accurate and effective search results not always an easy task.

1.1 Growth of the IR Systems

Many information retrieval systems use well-known algorithms and techniques (Faloutsos 1985); however, these algorithms were developed for relatively small and coherent collections such as newspaper articles or book catalogs in a (physical) library. A document collection like the Internet, on the other hand, is massive with much less coherence, changes more rapidly, and is spread over geographically distributed computers. This requires new techniques, or extensions to existing methods, to deal with gathering information, making index structures scalable and efficiently updateable, and improving the ability of search engines to discriminate useful from useless information.

There is no question that collections like the Internet are huge and challenging for IR systems. While several studies estimating the size of the Internet (Bar-Yossef et al. 2000, Lawrence and Giles 1999, Lawrence and Giles 1998) have reported slightly different numbers, most agree that over a billion pages were available in 1999. Given that the average size of a web page is 5K to 10K bytes, just the textual data contained on these pages consisted of tens of terabytes. The growth rate of this data is even more dramatic. According to Lawrence and Giles (1999), the size of the Internet has doubled in less than two years, and this growth rate is

projected to continue or increase. Google's index reportedly contains more than 8 billion pages and only includes a fraction of the total information available.

Aside from these newly created pages, the existing pages are continuously updated (Pitkow and Pirolli 1987). For example, in a study of over half a million pages over 4 months, it was found that approximately 23% of pages changed daily. In the .com domain 40% of the pages changed daily, and the half-life of pages was approximately 10 days (in 10 days half of the pages were gone, i.e., their URLs were no longer valid).

In addition to its size and rapid change, the interlinked nature of the Internet sets it apart from other data collections. Several studies have examined how the Internet's links are structured and how that structure can be modeled (Broder et al. 2000, Huberman and Adamic 1999). Broder's study suggests that the link structure of the Internet is broken into four categories (Broder et al. 2000): ~28% of the pages constitute a strongly connected core, ~22% are pages that can be reached from the core but not vice versa, ~22% reach the core, but cannot be reached from it, and the remaining pages can neither reach the core nor be reached from the core.

Because of the rapid growth and constantly-changing state of the Internet, several difficult questions have evolved regarding the problem of effective web search:

- What is a "document" that can be retrieved and deemed relevant for users? A document could be a single web page, a group of related pages, a web site, a domain, or some other logical view. Depending on how "documents" are designated, different retrieval techniques may be required. For example, if we searched at the web site level, relevance must be determined over the site as a whole rather than on a page per page basis.
- How do we maintain freshness and coverage of our index? The larger the Internet becomes, the harder it is to maintain a fresh and accurate index by constantly re-visiting and re-indexing pages. Would other alternative search architectures help maintain freshness and coverage? Should web sites be forced to wait until a search engine sees their changes for those changes to be updated in the index?
- What other information can we exploit to improve search results? Is there a way to employ user information or query context to improve our search? Are there inherently different

types of queries for which we can use specialized search types? Can the growing efforts in the Semantic Web provide context and better search?

Clearly, the current level of achievement in the field will not be sufficient for future needs. The next generation of web search tools must provide significantly better capabilities to learn, anticipate, and assist with information gathering, analysis, and management. To successfully meet these needs, new assumptions and methodologies are required to replace the search systems of today.

1.2 Anatomy of an IR System

Before we describe our research goals, it is useful to understand how an information retrieval system is typically configured (Figure 1-1). Every engine relies on a crawler module to provide the raw data for its operation. Crawlers are small programs that browse the document collection on the IR system's behalf, similarly to how a human user follows hyperlinks to reach different pages. These programs are given a starting set of documents whose pages they retrieve from the document collection. The links extracted from these documents are given to the crawler control module, which determines what links to visit next, feeding these links back to the crawlers. The crawlers also pass the retrieved pages into a page repository and continue visiting the collection until local resources, such as storage, are exhausted.

This basic algorithm has many variations that give IR systems different levels of coverage or topic bias. For example, crawlers in one system might be biased to visit as many sites as possible, ignoring deeply buried pages within each site. The crawlers in other systems might specialize on sites within a specific domain, such as medical records. The crawler control module is responsible for directing the crawling operation.

Once the IR system has completed at least one entire crawling cycle, the crawler control module is provided the indexes that were created during these earlier crawl(s). The crawler

control module may, for example, use data obtained from a previous crawl to decide which links the crawlers should explore and which links they should ignore.

The indexer module extracts all the words from each page and records the document where each word occurred. The result is a very large “lookup table” providing all of the links from the given words to documents. The table is of course limited to the documents that were covered in the crawling process. As mentioned earlier, text indexing of large collections like the Internet pose special difficulties, due to their size and rapid rate of change. In addition to these quantitative challenges, some document collections call for some special, less common, indexes that are created by the collection analysis module (i.e., a structure index, which reflects the links between documents).

During a crawling and indexing run, most information retrieval systems store the documents retrieved by the IR system. The page repository in Figure 1-1 represents this—possibly temporary—collection. IR systems sometimes maintain a cache of the documents they have visited beyond the time required to build the index. This cache allows them to provide result pages very quickly, in addition to providing the basic search facilities. Some systems, such as the Internet Archive, have aimed to maintain a very large number of documents for permanent archival purposes. Storage at such a scale again requires special consideration.

The query engine module is responsible for receiving and filling search requests from users. The engine relies heavily on the indexes and, sometimes, on the page repository. Due to the very large size of most collections and the fact that users typically only enter one or two keywords, result sets are usually very large. Hence, the ranking module has the task of sorting the results such that items near the top of the results list are most likely closer to what the user wants. The query module is of special interest because traditional information retrieval (IR)

techniques run into selectivity problems when applied without modification to searching collections like the Internet: Most traditional techniques rely on measuring the similarity of query texts with texts in a collection's documents. Since the typical search involves a tiny query over the vast collection of information, this prevents a similarity-based approach from filtering out a sufficient number of irrelevant documents from the search results. When deployed in conjunction with additional IR algorithms (i.e. stemming and relevance feedback), these IR systems can significantly improve retrieval precision in Internet search scenarios.

Clearly attempting to focus on all aspects of and techniques for IR would not prove to be particularly useful. By limiting the type of search system and features of search, we attempt to define a problem that can be addressed thoroughly and effectively.

1.3 Neural Networks

An Artificial Neural Network (ANN) is an information processing paradigm inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. An ANN is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. As a result, they learn like humans – from example.

One of the more common learning techniques used by neural networks is the back-propagation algorithm. With back-propagation, the input data are repeatedly presented to the neural network. With each presentation, the output of the neural network is compared to the desired output and an error is computed. This error is then fed back (i.e., back-propagated) to the neural network and used to adjust the weights such that the error decreases with each iteration

allowing the neural model to get closer and closer to producing the desired output. This process is known as "training."

Another type of neural network, known as a Morphological Neural Network (MNN), differs from most other neural networks in the way computation occurs at the nodes. Rather than using a multiply-accumulate node with a threshold function, the MNN uses lattice algebra operations (Ritter and Beavers 1999). One particular application for MNNs is to solve decision boundary problems in multi-dimensional space, an application we have integrated into our research.

The neural network paradigm is especially useful to examine because the tasks involved in an information retrieval system (namely, indexing, classification, thesaurus construction, and search) are characterized by a lack of well-defined rules and algorithms to be used for their general solution. In general, they can be regarded as input/output classification tasks. Document classification, for instance, is based on the idea that similar documents should be relevant for the same query. Indexing is the act of describing or identifying a document in terms of its subject content. Thesauri and semantic networks are built by clustering similar terms into common classes and linking these classes with appropriate relations. The search process itself links a user query with the relevant documents of the collection. The common factor is a classification process that associates documents and terms of the collection in different ways.

1.4 Our Research

We show that morphological neural networks have a variety of valuable uses in the information retrieval field. First, by creating a query engine that transforms user queries into an MNN capable of filtering document vectors in term space we can gain an increase in performance over vector-based methods. We explore alternate weighing schemes for an MNN

IR system and the effects of irrelevant terms on system performance. We also show that MNNs can be used to create an efficient and effective architecture for document classification.

The organization of this dissertation is as follows. Chapter 2 surveys the current research efforts on Information Retrieval and Latent Semantic Indexing. Chapter 3 surveys the current research topics on neural networks, morphological networks, and the use of these networks in IR. Chapter 4 introduces the proposed morphological neural network query engine. Chapter 5 describes the testing environments used in our research and presents the data collected. The dissertation concludes with Chapter 6, which provides our conclusions and future research directions to be considered.

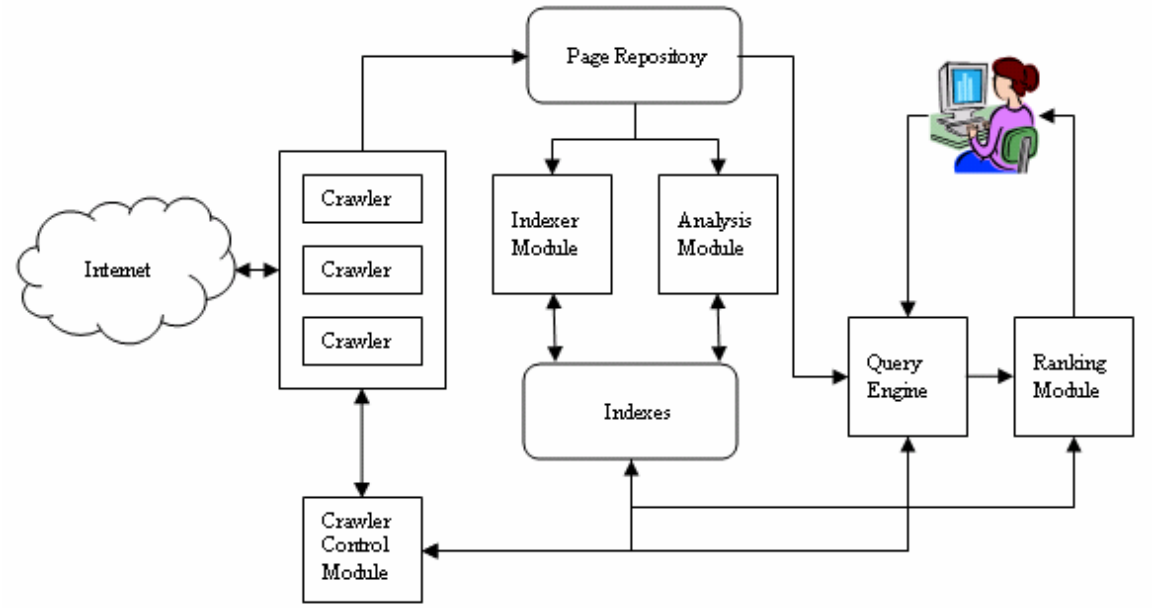

Figure 1-1. Typical IR system architecture

CHAPTER 2
INFORMATION RETRIEVAL MODELS

Most traditional information retrieval systems use index terms to locate and retrieve documents. An index term can be defined as a keyword or group of keywords having some meaning. In a more general sense, an index term is any word which appears in the document collection. Retrieval based on index terms is a fairly simple task but it raises some questions about the information retrieval task as a whole. For instance, retrieval using index terms uses the fundamental notion that the document semantics and user needs can be expressed though a set of index terms. This is problematic because of the loss of semantics and user input when we convert the text into a set of index terms. This process also maps the task of document matching to the very imprecise space of document terms. Typically, a document set retrieved for a keyword query contains many irrelevant documents. Another common problem is that most users are not well-trained in the formation of effective search queries.

One of the biggest problems with modern information retrieval systems is predicting which documents in the collection are considered relevant and which are not. Most IR systems use a ranking algorithm to order the retrieved documents, so documents at the top of the list are typically considered more relevant than ones further down. An effective ranking algorithm is usually a core component of any good information retrieval system.

Ranking algorithms operate on the concept of document relevance. Different notions of how to determine document relevance have given rise to a variety of information retrieval systems. By examining some of the established IR models, we can see how different notions of relevance are implemented. This chapter provides a survey of various IR models and discusses how these models relate to our research. By looking at the different flavors of IR we also provide a conceptual basis for the information retrieval system described later in this dissertation.

2.1 Boolean Model

The Boolean Model for information retrieval is a simple model derived from Set Theory and Boolean algebra. Because this model uses sets, most users find it easy to understand, and the semantics used by this model to write queries are very precise. Its inherent simplicity led to the Boolean model being widely adopted by the early generations of IR systems.

The Boolean model represents the index terms as either present or absent in a document. A query is composed of index terms connected by *and*s, *or*s, and *not*s, forming a Boolean expression that can be represented in disjunctive normal form (DNF). The query in Figure 2-1 requests documents dealing with both the terms "gator" and "basketball," but explicitly excluding any documents with the term "reptile." The model then predicts each document in the collection as either relevant or non-relevant. Because of the binary nature of each term, there is no notion of a partial match or a partially relevant document. The formal definition of the model is described below.

For the Boolean Model, the index term weight variables are all binary, i.e. $w_{i,j} \in \{0,1\}$. A query q is a conventional Boolean expression. Let $\vec{q}_{dnf}$ be the disjunctive normal form for the query q. Further, let $\vec{q}_{cc}$ be any of the conjunctive components of $\vec{q}_{dnf}$, and let g_i be a function that returns the weight associated with any index term k_i in any given vector. The similarity of a document d_j to the query q is defined as

$$sim(d_j, q) = \begin{cases} 1 & if \quad \exists \vec{q}_{cc} \mid (\vec{q}_{cc} \in \vec{q}_{dnf}) \wedge (\forall_{k_i}, g_i(\vec{d}_j) = g_i(\vec{q}_{cc})) \\ 0 & otherwise \end{cases}. \qquad (2\text{-}1)$$

If $sim(d_j, q) = 1$ then the Boolean Model predicts that the document d_j is relevant to the query q (in reality it may not be relevant); otherwise, the prediction is that the document is not relevant.

This model does have some major drawbacks which limit its usefulness. A document's relevance is determined by simply placing the document in either a relevant or not relevant group. There is no notion of a partially relevant document or any way to rank the relevant documents, which ultimately hinders the performance of the system. Because exact matching tends to lead to either a miniscule or massive result set, other models that include such concepts as index term weighting are more common in today's IR systems. Also while it is easy to write a query for the Boolean Model, it is very difficult to translate the information request of the user into the Boolean expression that forms the query. Because the Boolean expression query loses much of the semantics of the original user request, the accuracy of the relevant document set leaves something to be desired. Despite the drawbacks, the Boolean Model still provides a good basis for the information retrieval field.

2.1.1 Fuzzy Set Model

Using keywords to model documents and queries provides information that is only partly related to the semantics of the actual documents and queries, which can lead to poor retrieval performance. By modeling each query as a fuzzy set and each document as having some degree of membership in the set, the information retrieval process can be approached from another perspective. Various models of fuzzy set information retrieval systems have been proposed over the years, but here we discuss their common principles.

The fundamental premise of fuzzy set theory is representing items without a well-defined boundary. A membership function must be defined for each item in the system while also

allowing for partial membership of our classes. In the information retrieval context, a thesaurus for words in the document collection can be used to assign connections or weights between keywords to expand the user query and improve the performance of the system. By allowing partial inclusion it provides a ranking scheme in the information retrieval model.

The thesaurus can be constructed by defining a keyword connection matrix (Ogawa et al., 1991) whose rows and columns are associated to index terms in the document collection. In this matrix $\vec{c}$, a normalized correlation factor $c_{i,l}$ between two factors k_i and k_l can be defined by

$$c_{i,l} = \frac{n_{i,l}}{n_i + n_l - n_{i,l}} \tag{2-2}$$

where n_i is the number of documents which contain the term k_i, n_l is the number of documents which contain the term k_l, and $n_{i,l}$ is the number of documents that contain both terms. Such a correlation metric is quite common and has been used extensively with clustering algorithms.

The keyword correlation matrix $\vec{c}$ defines a fuzzy set associated to each index term k_i. In this fuzzy set, a document d_j has a degree of membership $\mu_{i,j}$ computed as

$$\mu_{i,j} = 1 - \prod_{k_l \in d_j} (1 - c_{i,l}) \tag{2-3}$$

which computes an algebraic sum over all the terms in d_j. A document d_j belongs to the fuzzy set associated to the term k_i if its own terms are related to k_i. Whenever there is at least one index term k_l of d_j which is strongly related to the index k_i ($c_{i,l} \cong 1$), then $\mu_{i,j} \cong 1$ and the index k_i is a good fuzzy index for the document d_j. In the case when all index terms of d_j are only loosely related to k_i, the index k_i is not a good fuzzy index for d_j ($\mu_{i,j} \cong 0$). The adoption

of an algebraic sum over all the terms in the document d_j allows a smooth transition for the values of the $\mu_{i,j}$ factor.

Fuzzy sets can provide much improved results in the IR field. They are superior to the classic Boolean Model because they allow for a less rigid interpretation of the user queries and provide the ability to order results from those deemed most relevant to the least relevant. Ultimately, though, there has not been much serious research into using fuzzy set models for retrieval in a large scale environment, so it is difficult to know how such a model would perform against other proven models.

2.1.2 Extended Boolean Model

Because of the inability to rank and order document results for a given query, the Boolean Model is no longer at the center of the modern IR world. Most new systems adopt some other form such as the Vector Model (see Section 2.3) or an extension to the classical Boolean Model which has functionality to allow for partial matching and term weighting.

The Extended Boolean Model is designed on the premise that a document including some of the query terms should be considered more relevant than a document containing none of the query terms. By formulating weights for all the term-document pairs in the collection, we can calculate the Euclidean distance from the ideal relevance and provide a degree of similarity ranking for each document in the collection.

Consider a two-dimensional case where we plot the queries and documents as shown in Figure 2-2 (Salton et al., 1983). A document d_j is positioned in this space through weights $w_{x,j}$ and $w_{y,j}$ associated with the pairs (k_x, d_j) and (k_y, d_j), where k_x and k_y are the Boolean terms in the query. We can assume the weights are normalized td-idf factors between 0 and 1 as

$$w_{x,j} = f_{x,j} \times \frac{idf_x}{m_i idf_i}$$

where $f_{x,j}$ is the normalized frequency of term k_x in document d_j and idf_i is the inverse document frequency for a generic term k_i. By looking at Figure 2-2 we can see that for a disjunctive query ($q_{or} = k_x \vee k_y$) taking the distance from (0,0) provides a measure of similarity to q_{or}. Likewise by taking the compliment of the distance from (1,1) for a conjunctive query ($q_{and} = k_x \wedge k_y$) provides a measure of similarity for q_{and}. If the distances are normalized we have a measure of similarity for both queries which can be defined as

$$sim(q_{or}, d) = \sqrt{\frac{x^2 + y^2}{2}} \text{ and} \qquad (2\text{-}4)$$

$$sim(q_{and}, d) = 1 - \sqrt{\frac{(1-x)^2 + (1-y)^2}{2}}\ .$$

If all the weights are Boolean ($w_{x,j} \in \{0,1\}$), a document is always positioned in one of the four corners of the space and the values for similarity are restricted to 0, $\sqrt{\frac{1}{2}}$, $1 - \sqrt{\frac{1}{2}}$, and 1. Given that the number of index terms in a document collection is defined as t, this model can be extended to consider distances in a t-dimensional space.

While this model provides an interesting solution to the problems found in conventional Boolean Model systems, there is insufficient research into the application of such a system to a document collection of some significance. Because of its relative obscurity, other categories of IR models are considered to be superior.

2.2 Probabilistic Model

The Probabilistic Model of information retrieval attempts to describe the IR problem within a probabilistic framework (Robertson and Sparck Jones, 1976). Fundamentally, for any user query there is a set of documents that contains only all the relevant documents (i.e., the ideal set of documents). If we had a description of this ideal set we could easily retrieve the documents relevant to the query. The goal is to be able to describe the properties associated with this ideal set. The challenge is that the exact properties for the set are not known, but that they are derived in some way from the semantics of the index terms. By guessing at these properties, we can form an initial probabilistic description and retrieve a result set of documents. Through an iterative process (perhaps with user feedback) we attempt to improve the description and ultimately obtain the ideal result set for the provided query. The model is described more formally below.

Given a query q and a document d_j in the collection, the probabilistic model attempts to estimate the probability document d_j is considered relevant. We are given binary variables for all the index weights (0,1), so we know a word is either included in a document or not. It is assumed that the probability of relevance is derived solely from the query and document collection and that there exists an ideal set of documents R that maximizes the probability of relevance as results to the query. By definition, documents in R are considered relevant to the query and documents not in R are considered non-relevant. For each query, a degree of similarity is assigned to each document d_j in the collection. The degree is determined as the ratio of $P(d_j$ relevant to $q) / P(d_j$ non-relevant to $q)$ or the odds d_j is relevant to q. By using this degree of similarity we can provide a ranking which should help minimize error in our selections. The degree of similarity is calculated as

$$sim(d_j, q) = \frac{P(\vec{d}_j \mid R) \times P(R)}{P(\vec{d}_j \mid \overline{R}) \times P(\overline{R})} \quad (2\text{-}5)$$

where $P(\vec{d}_j \mid R)$ is the probability of randomly selecting the document d_j from the set R of relevant documents, and $P(R)$ is the probability that a document randomly selected from the entire collection is relevant. $P(\vec{d}_j \mid \overline{R})$ and $P(\overline{R})$ have similar meanings, but appy to the set $\overline{R}$ of non-relevant documents.

Overall, being able to rank documents in order of their probability of being relevant is one of the biggest advantages of the Probabilistic Model. However, there are some disadvantages as well. To find the ideal set of documents the system must guess at an initial set of relevant and non-relevant documents, which is not an easy task. Because this model represents index terms solely as present or not-present, we cannot use any knowledge of the frequency of index terms in our rankings, which could hinder retrieval effectiveness. Also, user feedback is needed in many cases to refine the relevant document collection, which may not be seen as desirable by users.

2.2.1 Inference Network Model

The inference network model (Turtle and Croft, 1990, 1991) is based on a Bayesian network and associates random variables with the index terms, the documents, and the queries. A random variable associated with each document d_j represents the event of observing that document. The observation of d_j asserts a belief upon the random variables associated with its index terms. Thus by observing a document we cause an increased belief in the variables associated with the index terms.

Index terms and document variables are represented by nodes k_i in the network. Edges are directed from the document nodes to the term nodes, which indicate that observation of a documents yields improved belief on its term nodes. Queries are also represented by nodes q_i in

the network. The belief in the query node is a function of the beliefs in the nodes associated with the query terms, which is depicted by edges from the index term nodes to the query nodes. The ranking of a document d_j with respect to a given query q is a measure of how much evidential support the observation of d_j provides to the query q. The user information need is represented by a node I in the network. Another more general type of belief network model is described below.

2.2.2 Belief Network Model

The belief network model is different from the inference network model because it adopts a clearly defined sample space (Ribeiro-Neto and Muntz, 1996). This model yields a somewhat different network topology that provides a separation between the document and query portions of the network.

This model introduces what is referred to as the probability space. Here, each index term is viewed as an elementary concept and the set of all the index terms is referred to as the concept space. A concept is a subset of this space and could represent a document in the collection or perhaps a user query. This association of concepts with subsets is useful because it allows the logical notions of conjunction, disjunction, negation, and implication onto intersection, union, complementation, and inclusion.

In the belief network model the user query q is modeled as a network node associated with a binary random variable. This variable is set to 1 whenever q completely covers the concept space. This way we can assess the probability as the degree of belief in the coverage of the concept space by the query. Documents are networks nodes that also work in a similar fashion. They are modeled as the degree of belief in the coverage of the concept space by the document. Contrary to the inference network model, document nodes are referenced by the index term

nodes that compose the document. The document ranking scheme is based on the degree of coverage provided to a document by a query.

Overall the belief network model provides an interesting new idea for modeling the IR problem. However, because of the relative infancy of the concept, there has not been enough testing to show this model can consistently beat the vector model for information retrieval.

2.3 Vector Model

The Vector Model realizes that the use of binary weights is too limiting and proposes a framework in which partial matching is possible. This is accomplished by assigning non-binary weights to index terms in queries and in documents (Salton and Lesk, 1968). These term weights are ultimately used to calculate a degree of similarity between the documents in the collection and the query. By sorting the documents in decreasing order of this degree of similarity, the vector model takes into consideration documents which only match a subset of the query terms. The effect of this ordering is that the document answer set is generally more precise than the set retrieved by some of the other IR models.

For the Vector Model the weight $w_{i,j}$ associated with the term-document pair (k_i, d_j) is positive and non-binary. The index terms in the query are also weighted. Let $w_{i,q}$ be the weight associated with the pair (k_i, q) where $w_{i,q} \geq 0$. Then the query vector $\vec{q}$ is defined as $\vec{q} = (w_{1,q}, w_{2,q}, \ldots, w_{t,q})$ where t is the total number of index terms in the system. The vector for a document d_j is represented by $\vec{d}_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$.

Therefore, a document d_j and a query q are represented as t-dimensional vectors. The Vector Model proposes to evaluate the degree of similarity of the document d_j with respect to

the query q as a correlation between the vectors d_j and q. This correlation can be quantified by the cosine of the angle between these two vectors, which is defined as

$$sim(d_j, q) = \frac{\vec{d}_j \bullet \vec{q}}{\left|\vec{d}_j\right| \times \left|\vec{q}\right|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{t} w_{i,q}^2}} \qquad (2\text{-}6)$$

where $\left|\vec{d}_j\right|$ and $\left|\vec{q}\right|$ are the norms of the document and query vectors.

Since $w_{i,j} \geq 0$ and $w_{i,q} \geq 0$, the degree of similarity varies from 0 to 1. Rather than deciding if a document is relevant or not, the vector model simply ranks all the documents based on their degree of similarity. Partially matching documents can also be returned and a threshold can be set to eliminate matches with a very low similarity. To compute the rankings for a query over a document collection, we must first specify how the index weights are obtained.

Given a collection C of objects and a vague description of the set A, the goal of the simple clustering algorithm is to partition the collection C of objects into two sets: one which is composed of objects related to the set A and a second which is composed of objects not related to the set A. By vague description we mean that we do not have complete information for deciding precisely which objects are in set A. More sophisticated clustering algorithms might attempt to separate the objects of a collection into various clusters according to their properties. We only discuss the simplest clustering algorithm (relevant or non-relevant) to illustrate how the model works.

In a typical clustering problem, there are two primary issues that must be resolved. First, we must determine what features best describe the objects in set A. Second, we need to determine what features best distinguish the objects in set A from the remaining objects in collection C. The first set of features provides for quantification of intra-clustering similarity,

while the second set of features provides for quantification of inter-cluster dissimilarity. The most successful clustering algorithms attempt to balance these two sets.

In the vector model, intra-clustering similarity is quantified by measuring the raw frequency of a term k_i inside a document d_j. Such term frequency is usually referred to as the tf factor and provides one measure of how well that term describes the document contents. Inter-cluster dissimilarity is quantified by measuring the inverse of the frequency of a term k_i among the documents in the collection. This factor is most commonly referred to as the inverse document frequency (idf). The motivation for the idf factor is that terms appearing in many documents are not very useful for distinguishing a relevant document from a non-relevant one.

Let N be the total number of documents in the system and n_i the number of documents in which the index term k_i appears. Let $freq_{i,j}$ be the number of times term k_i is mentioned in the text of document d_j. The normalized frequency $f_{i,j}$ is given as

$$f_{i,j} = \frac{freq_{i,j}}{mfreq_{l,j}} \tag{2-7}$$

where $mfreq_{l,j}$ is the maximum frequency computed over all the terms that are mentioned in the text of the document d_j. If the term k_i does not appear in document d_j then $f_{i,j} = 0$. Let idf_i be the inverse document frequency for k_i be given as

$$idf_i = \log \frac{N}{n_i}. \tag{2-8}$$

The best known term-weighting schemes use weights which are given as

$$w_{i,j} = f_{i,j} \times \log \frac{N}{n_i} \tag{2-9}$$

or by a variation of this formula (Salton and Buckley, 1988). These term-weighting strategies are called $tf - idf$ schemes. A formula for query term weights is given as

$$w_{i,q} = \left(0.5 + \frac{0.5\, freq_{i,q}}{m_l\, freq_{l,q}} \right) \times \log \frac{N}{n_i} \qquad (2\text{-}10)$$

where $freq_{i,q}$ is the raw frequency of the term k_i in the text of the query q.

The main advantages of the vector model are that the term-weighting scheme improves retrieval performance, the partial matching strategy allows retrieval of documents that approximate the query conditions, and the cosine ranking formula sorts the documents according to their degree of similarity to the query. One argued disadvantage for the vector model is that the index terms are assumed to be mutually independent. However, in practice because of the locality of many term dependencies, the overall performance of the system can be hurt by considering them with respect to the entire document collection.

Despite its simplicity, the vector model is a resilient ranking strategy with general collections. It yields ranked answer sets that are difficult to improve upon without query expansion or relevance feedback within the framework of the vector model. A large variety of alternative ranking methods have been compared to the vector model, but the consensus seems to be that the vector model is either superior or almost as good. It is also simple to implement and very fast. For these reasons the vector model remains a popular model for information retrieval.

2.4 Latent Semantic Indexing Model

As discussed earlier, summarizing the contents of documents and queries through a set of index terms can lead to poor retrieval performance. First, many unrelated documents may be included in the answer set. Second, relevant documents which are not indexed by any of the

query keywords are not retrieved. The main reason for these two problems is the inherent vagueness associated with a retrieval process based on keyword sets.

The ideas in the query text are more related to the concepts described in it rather than to the index terms used in its description. The process of matching documents to a given query could be based on concept matching rather than index matching. This would allow retrieval of documents even when they are not indexed by query terms. For instance a document could be retrieved because it shares concepts with another document that is relevant to the given query. Latent semantic indexing is an approach which addresses these information retrieval issues.

The idea behind the latent semantic indexing model is to map each document and query vector into a lower dimensional space which is associated with concepts (Furnas et al., 1988). This is accomplished by mapping the index term vectors into the lower dimensional space. Retrieval in the reduced concept space is generally more effective than retrieval in the index term space. Let us discuss the basic terminology of this model.

Let t be the number of index terms in the collection and let N be the total number of documents. Define $\overrightarrow{M} = \left(M_{ij}\right)$ as a term-document association matrix with t rows and N columns. Each element M_{ij} is assigned a weight $w_{i,j}$ associated with the term-document pair $\left(k_i, d_j\right)$. This $w_{i,j}$ weight could be generated using the $tf - idf$ weighting scheme used in the classic vector model.

Latent semantic indexing decomposes the association matrix $\overrightarrow{M}$ into three components using singular value decomposition as given by

$$\overrightarrow{M} = \overrightarrow{K}\overrightarrow{S}\overrightarrow{D}^{t}. \qquad (2\text{-}11)$$

The matrix $\vec{K}$ is the matrix of eigenvectors derived from the term-to-term correlation matrix given by $\vec{M}\vec{M}^t$. The matrix $\vec{D}^t$ is the matrix of eigenvectors derived from the transpose of the document-to-document matrix given by $\vec{M}^t\vec{M}$. The matrix $\vec{S}$ is an $r \times r$ diagonal matrix of singular values where $r = \min(t, N)$ is the rank of $\vec{M}$.

Consider that only the s largest singular values of $\vec{S}$ are kept along with their corresponding columns in $\vec{K}$ and $\vec{D}^t$. The resultant $\vec{M}_s$ matrix is the matrix of rank s that is closest to the original matrix $\vec{M}$ in terms of least squares. This matrix is given as

$$\vec{M}_s = \vec{K}_s \vec{S}_s \vec{D}_s^t \qquad (2\text{-}12)$$

where s, $s < r$, is the dimensionality of the reduced concept space. The selection of a value for s attempts to balance two opposing effects. First, s should be large enough to allow for fitting of all the structure from the real data. Second, s should be small enough to allow the filtering out of the non-relevant representational details.

The relationship between any two documents in the reduced space of dimensionality s can be obtained from the $\vec{M}_s^t\vec{M}_s$ matrix given by

$$\vec{M}_s^t\vec{M}_S = \left(\vec{D}_s\vec{S}_s\right)\left(\vec{D}_s\vec{S}_s\right)^t. \qquad (2\text{-}13)$$

In this matrix the (i, j) element quantifies the relationship between documents d_i and d_j.

To rank documents with regard to a given user query, we simply model the query as a pseudo-document in the original $\vec{M}$ term-document matrix. We can assume the query is modeled as the document with number 0, then the first row in the matrix $\vec{M}_s^t\vec{M}_s$ provides the ranks of all documents with respect to this query.

Since the matrices used in the latent semantic indexing model are of rank s where $s \ll t$ and $s \ll N$, they form an efficient ranking scheme for the documents in the collection. They also provide for the elimination of noise (present in index term-based representations) and the removal of redundancy.

The latent semantic indexing model introduces an interesting conceptualization of the information retrieval model problem based on the theory of singular value decomposition. In our research we incorporate the features of the latent semantic indexing model into our proposed model for information retrieval.

2.5 Techniques for Information Retrieval

Most users are not well-versed in the structure or contents of a document collection. Without such knowledge it is difficult to form queries for accurate information retrieval. Many users spend a significant amount of time reformulating their queries to gain access to more relevant documents from the collection. This section examines various techniques for improving query formulation and techniques for document processing that can help to improve retrieval effectiveness for the users.

2.5.1 Query Expansion and Relevance Feedback

Relevance feedback is the most popular query reformulation strategy. To gain feedback, a user is provided a list of retrieved documents and asked to select those considered relevant. By examining the documents marked relevant, important terms and expressions found to be common in the document set can be used to refine the query and hopefully guide the new iteration of the query closer to the relevant documents in the collection.

Relevance feedback is a popular method for reformulating queries for several reasons. First, it abstracts away the details of the reformulation process by requiring the user simply to select relevant documents from a list. Second, it breaks the process of searching into several

smaller iterations, which tends to make the system more user-friendly. Finally, it provides a controlled process to help emphasize the relevant terms in the document collection which can help retrieval performance (Robertson and Sparck Jones, 1976). Since our research is focused on the vector model for information retrieval, we limit our discussion of relevance feedback techniques to the vector model.

The application of the relevance feedback model to the vector model assumes that the term-weight vectors of the relevant documents have similarities between them. It also assumes that the non-relevant document term-weight vectors are not similar to the relevant term-weight vectors. Ideally, the query should then be reformulated so it is closer to the term-weight vector space of the relevant documents.

For a given query, q, there are several sets of documents used in the process of term reweighting for relevance feedback. Let D_r represent the set of relevant documents as determined by the user from among the set of retrieved documents. Let D_n be the set of non-relevant documents from the set of retrieved documents. Let C_r be the set of relevant documents from the entire document collection. Also for these sets, let $|D_r|$, $|D_n|$, and $|C_r|$ be the number of documents in the defined sets. In our query optimizations we use α, β, and γ as tuning constants.

Let us consider the situation where the complete set of relevant documents, C_r, to a query q is known in advance. It can be shown that the optimal query vector for separating the relevant documents from the non-relevant ones is given as

$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\forall \vec{d}_j \in C_r} \vec{d}_j - \frac{1}{N - |C_r|} \sum_{\forall \vec{d}_j \notin C_r} \vec{d}_j . \qquad (2\text{-}14)$$

The problem with this equation is that documents comprising C_r are not known beforehand and are the ones we are attempting to find. One way to avoid this problem is to create an initial query and incrementally change the query vector as needed. This change occurs by restricting the computation to the documents known to be relevant based the user's input. The modified query is given by

$$\vec{q}_m = \alpha \vec{q} + \frac{\beta}{|D_r|} \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \frac{\gamma}{|D_n|} \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j \,. \qquad (2\text{-}15)$$

Notice how D_r and D_n are both sets of documents determined by the user's judgment. In most traditional version of this equation, α is fixed to be 1 (Rochio, 1971).

This query modification is adapted from the optimal query $\vec{q}_{opt}$ in which the terms of the original query are added. We include those terms because the original query could contain other important information. While there are several other techniques for query modification based on relevance feedback, they have been omitted from this discussion because they tend to yield comparable performance.

The above relevance feedback technique is simple and works well. Because the modified term weights are calculated directly from the retrieved documents the system is easily implemented. Based on observation of several experiments, the performance of the relevance feedback has been proven.

2.5.2 Stopword Removal

Stopwords are words that occur frequently throughout the document collection. Stopwords are generally considered to be poor discriminators and not useful in separating relevant documents from non-relevant documents. A few example of common words which are usually treated as stopwords are articles, prepositions, and conjunctions such as "a," "the," "in," "of,"

"the," "and," "or," etc. Some IR systems include extensive stopword lists that cover several hundred words (Frakes and Baeza-Yates, 1992).

Stopword removal has often been shown to improve retrieval performance beyond systems that simply weight common terms as insignificant. By reducing the overall size of a document collection's vocabulary, stopword removal also can lead to reduced space usage for storing the document index and improved retrieval time for system interaction with the retrieval index. Because of the perceived benefits, most production IR systems perform some level of stopword removal.

Stopword removal does have its disadvantages. There are plenty of cases where the stopwords are critical to user query. For example for the query "to be or not to be" the only word left after stopword removal is "be" since the remaining words are considered stopwords in most IR systems. To avoid this problem some IR systems opt to use full text indexing instead.

2.5.3 Stemming

A query for a document collection may sometimes use one form of a word while the relevant document uses a different form. Plurals and past tense are variations of word forms that prevent a match between a query and a relevant document. One approach to solve this problem is to only use word stems in the index.

A stem is the portion of a word that remains after the removal of its prefixes and suffixes (if any). For example, the word run is considered the stem for the words running, runner, runs, etc. Stems are considered useful in information retrieval because they map various words down to an underlying concept. Stemming also reduces the number of distinct index terms for the document collection.

There are three major types of stemming strategies: table lookup, n-grams, and affix removal (Frakes, 1992). Table lookup is simply pulling the stem for a word from a table of

stems. The major drawback to this method is the table is usually very large. N-gram stemming is based on identification of digrams and trigrams and is usually considered more of a clustering technique than a stemming technique. Affix removal relies on stripping off known prefixes or suffixes from words in the document collection. The most popular variant is the Porter algorithm, which is a five-phase process that uses a suffix list to assist in removal of unwanted suffixes (Porter, 1980).

While stemming seems to be a sensible approach to improve information retrieval performance, some experimental results indicate that stemming can lead to a loss of performance as well. Because of a lack of consensus, some information retrieval systems do not perform any kind of stemming.

2.6 Information Retrieval Evaluation

Time and space are the two most common metrics for evaluating computer systems but for information retrieval there are other metrics that may be of interest as well. Since user queries are inherently vague, the set of retrieved documents is not an exact answer and must be ranked according to their relevance. The ability to rank and retrieve the correct documents is of crucial importance, so a metric to evaluate how precise the answer set is for a given query is needed. This section discusses various metrics for retrieval performance evaluation.

2.6.1 Recall and Precision

Consider a query q and its set of relevant documents R. Let $|R|$ be the number of documents in this set. Assume a given retrieval strategy processes the query q and returns an answer set of documents A. Let $|A|$ be the number of documents in this set. Also, let $|Ra|$ be the number of documents in the intersection of the sets R and A. Figure 2-6 shows these sets in

the document collection. Recall is defined as the fraction of the relevant documents that has been retrieved, which is

$$recall = \frac{|Ra|}{|R|} \tag{2-16}$$

and precision is defined as the fraction of the retrieved documents that is relevant, which is

$$precision = \frac{|Ra|}{|A|}. \tag{2-17}$$

Recall and precision assume that all the documents in the answer set A have been seen. However, the user is generally not presented with the entire answer set at one time (most search engines only present between the next 10 to 20 relevant documents on each page). The documents in A are first sorted by a degree of relevance and presented in subsets, starting with the most relevant. In this situation, recall and precision can vary as the user examines the answer set A.

Precision and recall have been used extensively to evaluate retrieval performance. However, there are some problems with these two measures (Raghaven et al., 1989). First, the proper estimation of maximum recall for a query requires detailed knowledge of all the documents in a collection. With large document collections such information is unavailable, which implies recall cannot be estimated correctly. Second, recall and precision measure the effectiveness over a set of static queries. In many modern systems some degree of dynamic interactivity is used to arrive at the answer set, which means other types of measures could be more appropriate. Finally, recall and precision are well-defined when there is a linear ranking of the retrieved documents, but with weakly-ordered result sets they can be inadequate (Tague-Sutcliffe, 1992). We now examine some alternative measures for retrieval performance.

2.6.2 Alternative Measures

In some cases a single measure that combines recall and precision may provide a better evaluation of a retrieval system. One such measure, the harmonic mean F of recall and precision (Shaw et al., 1997), is defined as

$$F(j) = \frac{2}{\frac{1}{r(j)} + \frac{1}{P(j)}} \tag{2-18}$$

where $r(j)$ is the recall for the $j-th$ document in the ranking, $P(j)$ is the precision for the $j-th$ document in the ranking, and $F(j)$ is the harmonic mean of $r(j)$ and $P(j)$. The function F assumes values in the interval $[0,1]$ where 0 identifies no relevant documents have been retrieved and 1 all ranked documents are relevant. F only assumes a high value when both recall and precision are high. Determining the maximum value for F can be seen as an attempt to find the best possible compromise between recall and precision.

Another measure that combines recall and precision is called the E evaluation measure (van Rijsbergen, 1979). In this measure the users are able to specify whether they are more interested in recall or in precision. The E measure is defined as

$$E(j) = 1 - \frac{1+b^2}{\frac{b^2}{r(j)} + \frac{1}{P(j)}} \tag{2-19}$$

where $r(j)$ is the recall for the $j-th$ document in the ranking, $P(j)$ is the precision for the $j-th$ document in the ranking, $E(j)$ is the E evaluation measure relative to $r(j)$ and $P(j)$, and b is a user-specified parameter which reflects the importance of recall and precision. Values of b greater than 1 indicate that the user is more interested in precision while values of b less than 1 indicate the user is more interested in recall.

2.6.3 User-Oriented Measures

Recall and precision are both based on the assumption that the set of relevant documents for a query is always the same, independent of the user. However, different users may have different opinions of which documents are relevant. To address this problem some user-oriented measures have been invented (Korfhage, 1997).

Consider a document collection with a query q and a retrieval strategy to evaluate. Let R be the set of relevant documents for q and let A be the answer set retrieved. Let U be the subset of R which is known to the user. Let $|U|$ be the number of documents in the set U. The intersection of sets A and U yields the documents known to the user to be relevant that were retrieved, which we will call Rk. Let $|Rk|$ be the number of documents in this set. Also let $|Ru|$ be the number of relevant documents previously unknown to the user that were retrieved. Figure 2-7 provides an illustration of these sets. The coverage ratio is defined as the fraction of the documents known to the user to be relevant that has been retrieved, which is given by

$$cvrg = \frac{|Rk|}{|U|}. \tag{2-20}$$

The novelty ratio is defined as the fraction of the relevant documents retrieved that were known to the user, which is given by

$$novelty = \frac{|Ru|}{|Ru| + |Rk|}. \tag{2-21}$$

A high coverage ratio indicates that the system is finding most of the relevant documents the user expected to see. A high novelty ratio indicates that the system is revealing many new relevant documents to the user that were previously unknown.

2.6.4 Text Retrieval Conference Measures

Precision at 20 documents retrieved (P@20) counts the number of relevant documents in the top 20 documents in the ranked list returned for a topic. The measure closely correlates with user satisfaction in tasks such as web searching and is extremely easy to interpret. However, the measure is not a powerful discriminator among retrieval methods (the only thing that matters is a relevant document entering or leaving the top 20) and averages poorly (the constant cut-off of 20 represents very different recall levels for different topics). Because of these problems, P@20 has a much larger margin of error associated with it than mean average precision (MAP) (Buckley, 2000).

MAP is the mean of the precision scores obtained after each relevant document is retrieved, using zero as the precision for relevant documents that are not retrieved. Geometrically, it is equivalent to the area underneath an uninterpolated recall-precision graph. MAP is based on much more information than P@20 and is, therefore, a more powerful and more stable measure (Buckley, 2000). Its main drawback is that it is not easily interpreted. A MAP score of 0.4 can arise in a variety of ways, for example, whereas a P@20 score of 0.4 can only mean that eight of the top 20 documents retrieved are relevant.

In addition to these other measures, a preference-based measure is used to be robust in the face of incomplete relevance information rather than to exploit a different kind of judgment. The idea is to measure the effectiveness of a system on the basis of judged documents only. Since the scores for MAP and P@20 are completely determined by the ranks of the relevant documents in the result set, these measures make no distinction in pooled collections between documents that are explicitly judged as nonrelevant and documents that are assumed to be nonrelevant because they are unjudged. In contrast, the preference measure is a function of the number of times judged nonrelevant documents are retrieved before relevant documents. The measure is called

"bpref" because it uses binary relevance judgments to define the preference relation (any relevant document is preferred over any nonrelevant document for a given topic). For a topic with R relevant documents where r is a relevant document and n is a member of the first R judged nonrelevant documents as retrieved by the system,

$$bpref = \frac{1}{R}\sum_{r} 1 - \frac{\left| n \text{ ranked higher than } r \right|}{R}. \qquad (2\text{-}22)$$

The properties of bpref make it the preferred measure to use when comparing systems over test collections with incomplete relevance judgments (Buckley, 2004). Bpref is also more resilient to change than other measures when used on dynamic collections, though the limits of dynamic change that bpref can tolerate remain to be studied. A third type of test collection that bpref can be useful with is an embedded collection environment, where a test collection with known judgments is embedded in a much larger collection of similar documents with no judgments. Finally, bpref's property that relevant documents' scores are independent of the rank of other relevant documents make it more amenable to analysis than MAP.

2.7 Summary

In this chapter we provided a high-level survey of different information retrieval systems. We also discussed some of the techniques for improving retrieval performance. Finally, we concluded with a survey of some of the metrics for evaluating retrieval performance. Our proposed information retrieval model is an extension of the Vector model discussed in Section 2.3.

The next chapter provides a survey of neural network systems and their uses in information retrieval.

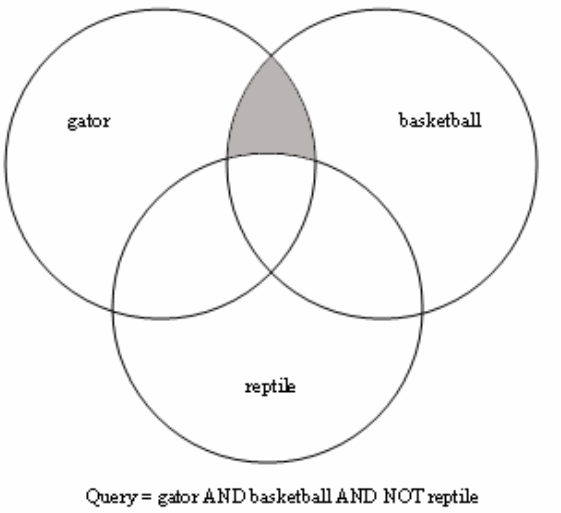

Figure 2-1. Example of a Boolean query

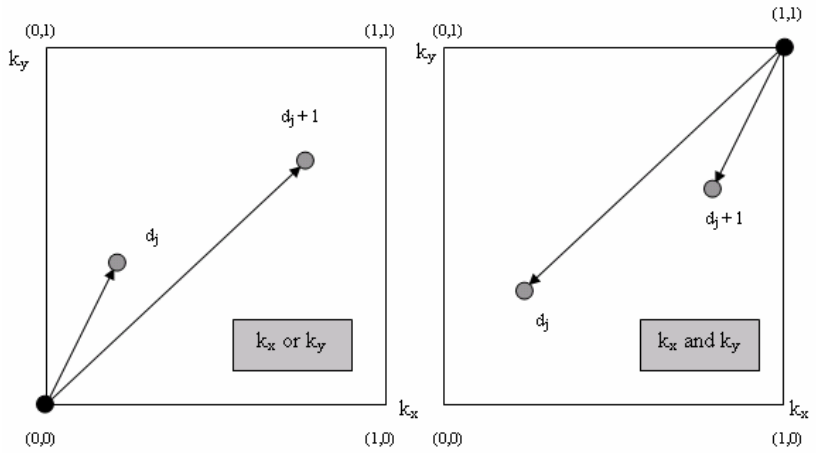

Figure 2-2. Sample Extended Boolean Queries in Two Dimensions

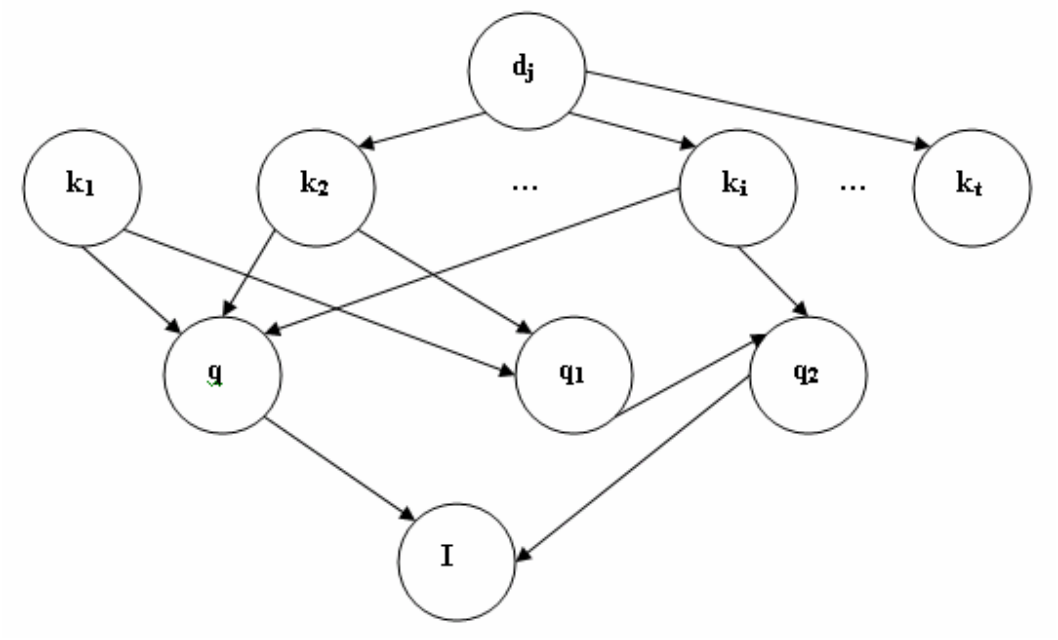

Figure 2-3. Sample inference network

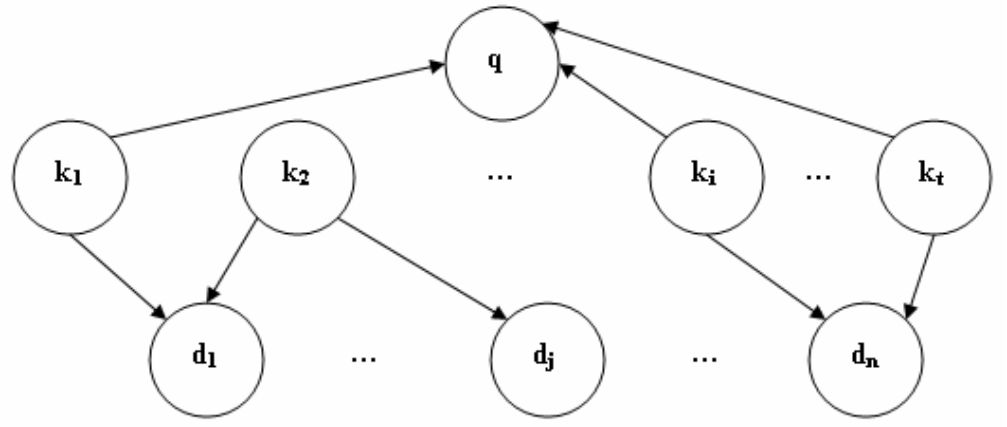

Figure 2-4. Sample belief network

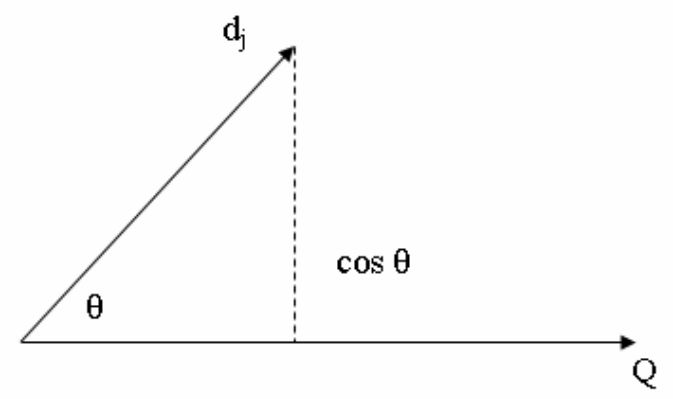

Figure 2-5. Cosine angle calculation

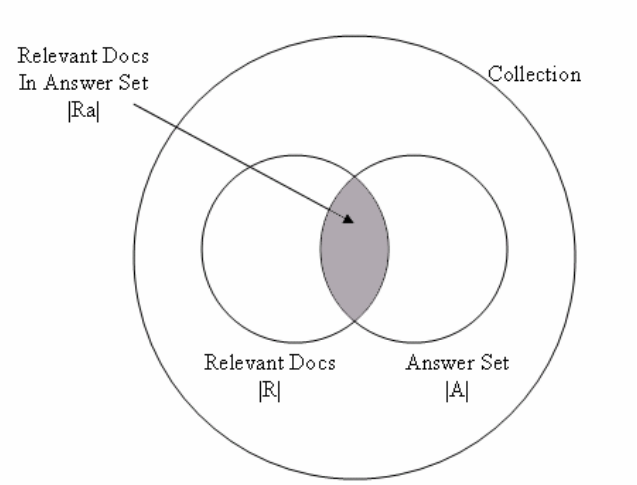

Figure 2-6. Precision and recall for a given query

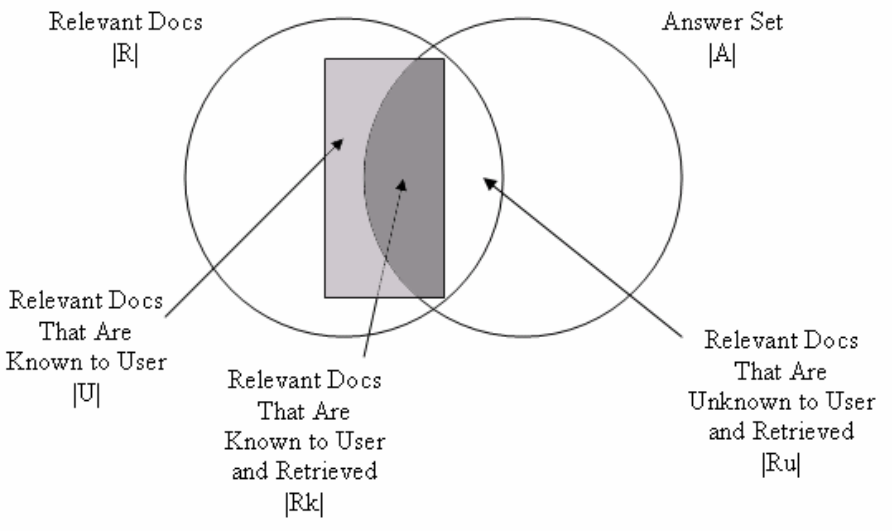

Figure 2-7. Coverage and novelty ratios for a given query

CHAPTER 3
NEURAL NETWORKS

Neural networks are systems whose architecture is modeled after the brain. They typically consist of many hundreds of simple processing units that are wired together in a complex communication network. Each unit or node is a simplified model of a real neuron that fires (sends a new signal) if it receives a sufficiently strong input signal from the other nodes to which it is connected. The strength of these connections may be varied for the network to perform different tasks corresponding to different patterns of node firing activity. This structure is very different from the traditional computing paradigm.

This chapter introduces artificial neural networks (ANNs) and morphological neural networks (MNNs) as the ANN's lattice algebra-based counterpart. First, an overview of biological neural network is presented. Then, we provide a summary of artificial neural networks and their implementation. Next, we look at several existing information retrieval models and how they incorporate ANNs. Finally, we present a brief overview of lattice algebra and examine the characteristics, computational capabilities, and main applications of single-layer morphological neural networks with dendritic structures.

3.1 Biological Neural Networks

Before discussing artificial neural network systems, we provide a brief overview of biological neural networks. This gives some basic insight into the modeling and function of our artificial neural networks as well as provides an explanation of some of the various terminology associated with neural networks.

Neurons are nerve cells responsible for receiving, processing, and transmitting information. Neurons come in several different varieties, but all neurons have a cell body (soma)

and multiple neurites that connect to the cell body. Most neurons feature a large nucleus with a single nucleolus.

There are two major types of neurites. Dendrites are neurites that conduct inputs towards the neuron, and the axon is a single neurite that transmits signals away from the neuron. Dendrites are profusely branching extensions of the cell body that established a large number of connections (synapses) with other neurons. The number of branches in the tree-like dendritic structure can range anywhere from several hundred to several hundred thousand. Axons are connected to the soma by the axon hillock. The axon also leads to the synapses connected to the dendrites of other neurons in the system.

Synapses are the regions where the axon of one neuron comes into close proximity with the dendrites of another neuron. It is in this region that the interneuronal communication occurs. Recent studies show that more computation may occur in the dendrites then originally thought (Segev, 1998). Synapses have one of two roles: excitatory and inhibitory. The excitatory synapses stimulate the target cell to receive impulses, while the inhibitory synapses try to keep the target neuron from receiving impulses. The postsynaptic neurons can also be either excitatory or inhibitory and also try to receive or ignore inputs accordingly.

By modeling a network after these biological networks we hope to be able to gain some of the functionality of the brain. The next section discusses some of the traditional types of artificial neural networks.

3.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are an attempt to model the functionality of the brain. The network is compromised of many small and simple processing units that are highly interconnected. Each unit is designed to simulate one neuron and mimic neuronal behaviors. While biological science has many different kinds of neurons, in the computer science realm

most variations of the neural network are based on one type of simplified abstraction of the neuron. Below, we discuss some of the properties of neural networks and some of the advantages for using them.

While different ANN models have been introduced since the creation of the neural network, most of these models have the same major characteristics. By design, most ANNs are inherently parallel. Typically, there are many small processing elements (neurons) running at the same time. In most models the neurons only have access to local information and transmit outputs based solely on the input connections to that specific neuron in the current state. Finally, ANNs have the potential to "learn." As their inputs vary over time, the neuron's output values tend to settle on a stable value.

ANNs present advantages that make them a viable alternative for certain classes of problems in information processing. ANNs are easy to create, and implementing a network to solve a specific problem can be done quickly. Most networks are fault tolerant and will still perform reasonably well with the loss of some of the processing elements or several errors in the inputs. Some ANNs are capable of learning and self-organization. Clearly, the ANN model is a powerful tool for solving certain types of problems, but it suffers from some limitations as well.

One major limitation is that while the ANN model is designed to run in parallel, most models are actually simulated on sequential systems. This can lead to a drop in performance. Second, they are also not very effective at solving sequential problems. For larger problems, the size of the network can lead to large increases in computational needs, so the network may lose performance without sufficient resources. Third, the ANN model is not capable of understanding the results it produces and cannot explain any learned features from the training

sets. Finally, neural networks can be very difficult to design, which can lead to poorly implemented solutions to the problem at hand and less than acceptable results.

There are many different areas where artificial neural networks have found acceptance. One of the most common applications of the ANN model is in classification and pattern recognition problems. Here, the ANN is given a training set of inputs designed to illustrate the differences between several classes of objects or types of patterns. After training, the network attempts to "learn" the characteristics and classify any new patterns or objects provided as input. Another use of the ANN model is for clustering problems. In a clustering problem, the network tries to group objects together into sets based on the features of the objects without any training data. This is an example of unsupervised training. Finally, ANNs are commonly used to solve prediction problems. Examples of these problems include weather prediction, stock market behavior forecasting, and inventory control analysis.

Because of the wide variety of applications for ANNs, many different classical models have emerged to solve these problems. While each ANN model has its own set of advantages and disadvantages, we limit our discussions to the perceptron model of the classical artificial neural network. This model is of particular importance because of the close relationship to the morphological perceptron model, which forms the basis of our information retrieval query engine.

3.2.1 Single-Layer Perceptron

The perceptron model of the neuron is a variation on the classical linear neuron used to build classifier networks. The perceptron creates a continuous output representing the firing of the neuron cell. A typical perceptron update function is described by

$y(t+1) = f\left(\sum_i w_i x_i(t) - \theta\right)$, where f is sigmoidal.

The single layer perceptron (SLP) typically updates its weights using the Delta Rule (Rosenblatt, 1958). This rule increases the neuron weight if it does not fire when it should and decreases the weight if it fires when it should not. Let j be a neuron with an output of x_j, and i be a neuron with an output of y_i. If w_{ij} is the weight of the synapse from j to i, the weight is updated by $\Delta w_{ij} = \eta\left(o_i - y_i\right)x_j$ where o_i is the desired output at neuron i and η is the learning rate of the system. The perceptron convergence theorem proves that the Delta Rule settles on a set of weights that can solve a linear classification problem (Arbib, 1998).

3.2.2 Multilayer Perceptron

The multi-layer perceptron (MLP) is a feed-forward neural network that contains multiple layers of neurons. Figure 3-3 shows a typical MLP, although for clarity some of the connections between neurons are omitted. Each layer feeds their result state forward to the next layer. There is one layer of input neurons, one layer of output neurons, and some number of trainable neurons typically called the hidden layer(s).

The MLP network is trained using what is known as the error backpropagation method (Rumelhart et al., 1986). This method takes the gradient of the error calculation and passes it back to the previous layer of neurons. The activation function for each neuron is a sigmoidal function f which is then passed to the next layer. The sigmoid f is used for its continuity which ensures the convergence of the training method.

We define the error E as $E = \sum_k \left(o_k - y_k\right)^2$ where k ranges over the output neurons. The neuron weights w_{ij} are updated using the gradient descent rule, which is defined as

$$\Delta w_{ij} = -\frac{\partial E}{\partial w_{ij}} = 2\sum_k \left(o_k - y_k\right)\frac{\partial y_k}{\partial w_{ij}}. \qquad (3\text{-}1)$$

The weights are updated starting at the output layer and proceeding backwards layer by layer. The weight change Δw_{ij} is proportional to $\delta_i y_j$, where the error δ is defined as $\delta_i = (o_i - y_i) f_i'$ for output neurons and $\delta_i = \left(\sum_k \delta_k w_{ki}\right) f_i'$ for hidden layer neurons.

The MLP has gained in popularity due to its ability to solve a variety of classification problems. One drawback of MLPs is the possibility of having open separation surfaces in the pattern space. Networks with less hidden neurons than input neurons create such open surfaces (Gori et al., 1998) and are unable to model patterns distributed in clusters because non-clustered patterns tend to be misclassified. Thus, the MLP is not always a good choice for some types of pattern recognition.

By increasing the number of hidden neurons the separation surfaces may be closed, but showing they are closed is an NP-hard problem (Gori et al., 1998). Another drawback of increasing the hidden neurons is that there may be an exponential increase in the training time for the network. Another type of model based on lattice algebra, discussed below, does not have the same limitation in pattern classification.

3.3 Neural Networks in Information Retrieval

One important class of problem where ANNs can provide a viable solution is information retrieval. In most information retrieval systems document vectors are compared to query vectors to determine a ranking of the documents. Because of their pattern matching capabilities, this has led to a variety of alternative models for IR systems. In this section we survey various models of neural network IR systems.

3.3.1 Three-Layered Neural Network Model

In this model a neural network is composed of three layers: one for the query terms, one for the document terms, and one for the documents in the collection. See Figure 3-4. Note that

the topology of this network is similar to the belief and inference networks described in Chapter 2. Here, the signals are initiated from the query nodes and propagated to the document nodes. Through a spreading activation, the documents may return a signal to the term nodes and then activate more documents. Eventually this process reaches equilibrium. This allows the system to match documents that may not match any of the original query terms, similar to using a thesaurus (Wilkinson, 1991).

The initial query term weights are set and fixed at 1. At each document node in the system, all of the incoming signals are summed. The activation level associated with each document d_j is given by

$$\sum_{i=1}^{t} \overline{w}_{i,q} \overline{w}_{i,j} = \frac{\sum_{i=1}^{t} w_{i,q} w_{i,j}}{\sqrt{\sum_{i=1}^{t} w_{i,q}^2 \times \sum_{i=1}^{t} w_{i,j}^2}}, \qquad (3\text{-}2)$$

which is similar to the ranking system used by the classic vector model for information retrieval. After completing one round of propagation, the spreading activation is continued until the signals stop. To improve results, a minimum threshold is employed to prevent weak signals from spreading to other document term nodes. This system can activate term nodes that may not have been in the original document and might lead to a more complete list of relevant documents.

3.3.2 Probabalistic IR Network Model

In this model (Kwok, 1989), a three-layer neural network is used to model a probabilistic IR system. The architecture is similar to the model described in Section 3.3.1, with the input layer Q representing the query terms, the hidden layer T represents the document terms, and the output layer D represents the documents in the collection. There are no connections between the neurons in each layer.

The initial connection strength between each query neuron a and index term neuron k, which represents the probability this query will use term k, is set as $w_{ak} = q_{ak} / L_a$ where q_{ak} is the term frequency and L_a is the length of the query. Similarly, the initial connection strength between each document neuron i and index term neuron k, which represents the probability this document will use term k, is set as $w_{ki} = q_{jk} / L_i$ where q_{jk} is the term frequency and L_i is the length of the document.

Once the initial strengths have been calculated, the strengths are reinforced using a spreading activation. The change in the term weight for an iteration is defined as $\Delta k_i = \beta\left(a_k - r_{ki}\right)$ where β is the learning rate of the network and r_{ki} is the original probability of the term's relevance before the learning process. When training is complete the probabilistic neural network is capable of performing information retrieval comparable to traditional probabilistic IR.

3.3.3 Competition-based Connectionist Model

In a competition-based connectionist model (Syu, 1994), a layer of nodes in the neural network represents the documents and another layer of nodes represents the index terms extracted from the document set, with weighted links connecting documents to the related index terms. A thesaurus, used to provide synonyms, is represented by another layer of nodes with weighted links connecting the index terms and the related synonyms. Each query causes activation of various synonym nodes, which then activates the appropriate term nodes in the network.

This model attempts to draw semantic information by incorporating a merged thesaurus by combining Roget's thesaurus with WordNet 1.4. WordNet organizes various words into synonym sets. These sets are connected based on synonymy, antonymy, merononyms, and

hyponymy. By activating a given synonym set all of the terms in the network associated with the set are included in the query as well. While this helps improve performance for most queries, one drawback to the use of WordNet is that terms are not broken down into parts of speech, which can fail to eliminate some unnecessary terms in the query processing.

To process a user query, both the present index terms and the synonymous terms are activated in the neural network. Let the activated set of neural nodes be τ^+ and the set of documents be D. A winner-take-all competition algorithm is then used to iteratively update τ^+ and then D. The rule for updating the index nodes is given by $\tau_j(t) = 1 - \prod_{d_i \in causes(r_j)} \left(1 - r_{ij} d_i(t)\right)$ and is run until the computation reaches equilibrium.

By capturing semantic associations between the documents and index terms in a neural network, this model is able to provide a faster and more effective information retrieval solution.

3.3.4 Self-organizing Semantic Map Model

The semantic map model (Lin et al., 1991) describes the relationship between words in a document collection using a self-organizing (Kohonen) feature map. A Kohonen feature map is a major unsupervised learning method for neural networks that models a set of features geometrically in two or three dimensions.

The feature map algorithm takes an input vector of n dimensions (features) and maps these inputs onto a two-dimensional grid of nodes. Each node in the grid is assigned an n-dimensional vector of initially small values. To train the network, a random input vector is selected and the grid of nodes is examined to find the closet node to the input. This (winning) node is adjusted to be closer to the input vector, and, in addition, other nearby nodes are adjusted with some diminishing return to also be closer to the input vector provided. Once the training converges, the network contains an ordered map of nodes reflecting the spatial relationships between the

features in the system, and the importance of each feature can seen by the amount of space it occupies on the grid (Ritter and Kohonen, 1989).

To demonstrate this technique, the model was given a database of 140 documents on Artificial Intelligence (the LISA database). After indexing the documents, 25 words were selected to use as indexing words in the feature map. The documents were used as inputs to the system and the grid contained 140 nodes with the 25-dimensional feature vector. Once the system has converged, the unit vectors for each of the 25 terms are compared to all the nodes and each node is assigned the "winning" word associated with it. If multiple words are in contention, then those regions are merged and both terms used (e.g., "natural" and "language" would become "natural language").

By providing a semantic map of terms that define a query, users have the ability to see the relationship between documents in the collection. Queries may be able to function with a lower recall and precision rate and still provide the user with a sufficiently good set of results. By reducing the number of dimensions in the result set, the result space is more manageable than the document space. The features for the result space can be either user-defined or automatically generated.

3.3.5 Conclusions

Using neural networks for information retrieval offers some benefits beyond some of the more traditional IR models. By allowing for the retrieval of documents not initially related to the query terms, recall and precision can be improved. Several different models have been created to demonstrate the feasibility of this idea, but one common design specification for most of them is that they model the term-document matrix into a neural network. Later, we develop a model that uses a neural network to rank documents in a vector-based IR model. But first we examine

lattice algebra and a specialized type of neural network based off it that is a critical component of our model.

3.4 Lattice Algebra

Lattice algebra is a variation of algebra that is based on the theory of lattices. The history of this theory is broken into three stages (Birkhoff, 1964). Lattice theory was originally discussed in Boole's "Mathematical Analysis of Logic" in 1847 and was followed by the work of various other mathematicians in both the 1930s and 1950s. Later, a nonlinear matrix calculus commonly referred to as minimax algebra was developed to formulate equivalent matrix algebra concepts for several linear ones.

Lattice algebra provides a computational framework for a new type of artificial neural network known as a morphological neural network. The use of morphological neural models based on lattice algebra is a focus of our research, and we examine the MNN in more detail in the next section. MNNs are intrinsically nonlinear due to the fact that lattice-induced operations are nonlinear.

In the context of linear algebra, matrix operations such as pointwise matrix addition and matrix multiplication are defined in terms of scalar operations of addition and multiplication (Ritter and Urcid, 2003). These operations are defined within a ring such as the ring of real numbers $(\mathbb{R}, +, \times)$. Each matrix operation from linear algebra has a corresponding matrix operation in lattice algebra. Lattice-based matrix operations are defined within the bounded lattice-ordered group $(\mathbb{R}_{\pm\infty}, \vee, \wedge, +, +')$, where $+$ denotes the extended real addition $a + (-\infty) = (-\infty) + a = -\infty \forall a \in \mathbb{R}_{-\infty}$ and $+'$ denotes the extended real addition $a +' \infty = \infty +' a = \infty \forall a \in \mathbb{R}_{\infty}$. Here, we define matrix operations based on lattice algebra.

First, we define point-wise minimum and maximum operations, which are based on the point-wise matrix addition from linear algebra. For two $m \times n$ matrices, let the point-wise maximum $A \vee B$ of matrices A and B be defined as $c_{ij} = a_{ij} \vee b_{ij}$, and the point-wise minimum $A \wedge B$ of matrices A and B be defined as $c_{ij} = a_{ij} \wedge b_{ij}$. Also, for a matrix A of size $m \times p$ and a matrix B of size $p \times n$, let the dilation operation (addition of every element in a to every element in b) be defined as $\vee_{k=1}^{p} \left(a_{ik} + b_{kj} \right)$, and the erosion operation (dilation of a with $'b$) be defined as $\wedge_{k=1}^{p} \left(a_{ik} + 'b_{kj} \right)$. These operations are closely tied to the dilation and erosion operations for an image and structuring element (Serra, 1982). For these matrices we can also define the max product matrix C where each point $c_{ij} = \vee_{k=1}^{p} \left(a_{ik} + b_{kj} \right)$ and the min product matrix D where each point $d_{ij} = \wedge_{k=1}^{p} \left(a_{ik} + 'b_{kj} \right)$.

The concepts from lattice algebra provide a framework for the definition of the morphological neural network. The next section provides a basis for understanding MNNs in detail.

3.5 Morphological Neural Networks

Morphological neural networks are the lattice algebra-based version of classical artificial neural networks. Just like the classical model, MNNs can be single-layer or multilayer and are capable of solving various types of classification problems. Because the MNN is based on a different foundation, the underlying properties and capabilities of these networks are different as well. In this section, we explore the design and capabilities of this new type of neural network in more detail. We focus on the single-layer version of the morphological perceptron (SLMP).

3.5.1 Computational Model of the SLMP

Single-layer morphological perceptrons with dendritic structures are feed-forward networks consisting of one layer of processing elements, with each element being a neuron equipped with dendrites and performing morphological operations based on lattice algebra. Because it is a single-layer network, the processing layer is also the output layer. As with most other neural network models, the input layer is not considered a layer because its sole purpose is to feed in the values from the input vector.

A typical SLMP consists of n input neurons and m output neurons denoted by N_i and M_j, where $i = 1,\ldots,n$ and $j = 1,\ldots,m$ (Ritter and Sussner, 1996). The axons of the input neurons split into branches whose terminal fibers connect to the dendrites of the output neurons. As with biological neurons but unlike the classical perceptron, an output neuron may have multiple synaptic contacts from the same input neuron. The value of input neuron N_i propagates through its axonal tree to the terminal branches that contact the synaptic sites of the output neuron's dendrites. The weight of the axonal terminal fiber of neuron N_i terminating on the k^{th} dendrite of an output neuron M_j is denoted by w_{ijk}^{ℓ}, where $\ell \in \{0,1\}$ distinguishes between excitatory $(\ell = 1)$ and inhibitory $(\ell = 0)$ input to the dendrite. The k^{th} dendrite of the neuron M_j responds to the total input received from the input neurons and will either accept or inhibit the input based on its own response factor. This basic model is represented in Figure 3-5.

The computation that occurs at each dendrite is based on morphological operations using lattice algebra (Ritter et al., 2003). Let D_{jk} denote the k^{th} dendrite of neuron M_j. The computation performed at D_{jk} is given by

$$\tau_k^j(x) = p_{jk} \bigwedge_{i \in I_{jk}} \bigwedge_{\ell \in L_{ijk}} (-1)^{1-\ell} \left(x_i + w_{ijk}^{\ell} \right) \tag{3-3}$$

where $x = (x_1, x_2, \ldots, x_i, \ldots, x_n)' \in \mathbb{R}^n$ denotes the input vector supplied to the input neurons, $I_{jk} \subseteq \{1, 2, \ldots, i, \ldots, n\}$ represents the set of all input neurons N_i that have at least one terminal fiber on dendrite D_{jk}, $L_{ijk} \subseteq \{0,1\}$ represents the set of terminal fibers of N_i on D_{jk}, $w_{ijk}^{\ell} \in \mathbb{R}_{\pm\infty}$ denotes the synaptic weight of the connected fibers, and $p_{jk} \in \{-1,1\}$ denotes the excitatory $(p_{jk} = 1)$ or inhibitory $(p_{jk} = -1)$ response of D_{jk} to the received input.

An input neuron N_i can have between zero and two connections to D_{jk}. If there is no connection then $L_{ijk} = \varnothing$, an excitatory connection is represented by a 1, and an inhibitory connection is represented by a 0. Both I_{jk} and L_{ijk} cannot be simultaneously empty, because if no neurons have any connections to a particular dendrite it has no impact of the output of the system. The term $(-1)^{1-\ell}$ represents the input response of the fiber ℓ connecting N_i to D_{jk}. Since it is possible for two connections between an input neuron and a particular dendrite to exist, one could be an excitatory while the other could be an inhibitory input.

The set I_{jk}, which provides the indices of input neurons connected to D_{jk} can also be either partial or complete. In the partial case, the system can be converted to have complete equivalency by substituting special connections for the missing fibers. By using placeholder weights of $+\infty$ for missing excitatory connections and $-\infty$ for missing inhibitory connections all of the input neurons can be connected to all of the output neuron dendrites. If the network is fully connected, the formula dendrite output can be changed to

$$\tau_k^j(x) = p_{jk} \bigwedge_{i=1}^{n} \left[\left(x_i + w_{ijk}^1 \right) \wedge - \left(x_i + w_{ijk}^0 \right) \right]. \tag{3-4}$$

When all the connections from an input neuron to a dendrite are present, the region in pattern space represented by the dendrite is a closed hyperbox. This characteristic proves useful for our query architecture.

The net input value $\tau_k^j(x)$, computed at dendrite D_{jk}, is then passed to the soma of M_j and the state of M_j is computed as a function of the input received from all the dendrites. Let $\tau^j(x)$ represent the total input received from all the dendrites, which is given by

$$\tau^j(x) = p_k \bigwedge_{k=1}^{K_j} \tau_k^j(x) \tag{3-5}$$

where the neuron response factor $p_j = \pm 1$. The output of the network is then calculated by applying an activation function f such that $y_j(x) = f\left[\tau^j(x)\right]$. The most common activation function is a hard-limiter. With this understanding of the workings of the SLMP, we can now examine the computational capabilities of this model.

3.5.2 Computational Capabilities of the SLMP

Both the classical single-layer perceptron and the single-layer morphological perceptron consist of one layer of output neurons connected to a layer of input neurons through their axonal fibers. However, unlike the classical SLP model, the SLMP model has a dendritic structure based on computations performed by lattice algebra. The computational capabilities of this model varies greatly from the classical perceptron as well.

The SLMP model can recognize patterns of a single category and can discriminate them from the rest of the pattern space. If $X \subset \mathbb{R}^n$ is compact and $\varepsilon > 0$, then there exists an SLMP that assigns every point of X to class C and every point $x \in \mathbb{R}^n$ to class $\overline{C}$ whenever $d(x, X) > \varepsilon$ (Ritter et al., 2003). Let $d(x, X)$ represent the distance of the point $x \in \mathbb{R}^n$ to the

set X . Regardless of the number of points, connectivity, or convex nature of the set, X can be approximated to any degree of accuracy $\varepsilon > 0$ by an SLMP with one output neuron.

For any SLMP trained to recognize the set X , let $y = 1$ for all points $x \in X$ and $y = 0$ for all points x outside the banded region that surrounds the set X . The output is unknown for points within this banded region, but the thickness of this region ε may be made arbitrarily thin. Therefore, any decision problem involving a finite set of points can be solved by constructing and training an SLMP to recognize it. This can also be generalized to include multiple sets, as seen in Figure 3-6.

3.5.3 SLMP Training Algorithms

Although there are several different training models for the SLMP, because of their dendritic properties all these models have some similar characteristics. These algorithms are also different from the training algorithms for the classical perceptron because of variations in the underlying neural network design.

Unlike the traditional training algorithms, the structure of the SLMP to be trained is not set beforehand. The perceptron grows any needed dendrites during the training process. This helps improve training efficiency because there is no need to correct the network after the training phase is over. The SLMP can incorporate the existence of a new training pattern without needing re-training on the existing training pattern set. Training takes one cycle so each training pattern is visited only once. This can reduce training time compared to training a classical network. Because of the compact nature of any training set, we are also guaranteed that an SLMP exists for any such set.

The primary distinction in the various training algorithms is the partitioning strategy used to determine the regions associated with each class in the pattern space. Some methods begin

with a small region around each pattern and grow the regions into a union of the space representing each class. Another approach is to encompass all of the patterns in one region, then remove any conflicting regions to correctly categorize each pattern. Because the final results for both approaches are essentially the same, we discuss in more detail only one variation of the region elimination algorithm (Ritter and Urcid, 2003) in this section.

The elimination-based algorithms begins by creating a large region that includes all the training patterns belonging to the excitatory region in class C_1. Any patterns that are part of inhibitory region in class C_0 inside the original region are then removed by creating hyperboxes around these patterns and excluding them from that region.

In this algorithm, the responses of the dendrites are always set the same way. The first inclusive region is recognized by the excitatory dendrite D_1. All of the removed regions are recognized by inhibitory dendrites, one per region. The response at the output neuron is also excitatory because its computation is the intersection of the regions recognized. The twelve step algorithm is described in more detail below.

Let $T = \left\{ x^\varsigma, d^\varsigma : \varsigma = 1,\ldots,m \right\}$ denote the training set where $x^\varsigma \in \mathbb{R}^n$ and $d^\varsigma \in \{0,1\}$ such that $d^\varsigma = 1$ if and only if $x^\varsigma \in C_1$ and $d^\varsigma = 0$ if and only if $x^\varsigma \in C_0$.

Step 1: Initialize parameters and the auxiliary index set. Set the weights of the hyperbox containing the entirety of C_1. Let $k = 1$, $P = \{1,\ldots,m\}$, $I = \{1,\ldots,n\}$, and $L(i) = \{0,1\}$ for all $i \in I$ and compute $w_{ik}^1 = -\underset{d^\varsigma = 1}{\wedge} x_i^\varsigma$ and $w_{ik}^0 = -\underset{d^\varsigma = 1}{\vee} x_i^\varsigma$ for $i = 1\ldots n$.

Step 2: Let $P_k = (-1)^{sgm(k-1)}$ and $r_{ik}^\ell = (-1)^{1-\ell}$ for all $i \in I, \ell \in L(i)$. Compute the response of the current dendrite $\tau_k\left(x^\varsigma\right) = P_k \underset{i \in I}{\wedge} \underset{\ell \in L}{\wedge} r_{ik}^\ell \left(x_i^\varsigma + w_{ik}^\ell \right)$ for all $\varsigma = 1,\ldots,m$.

Step 3: Compute the total response of the output neuron N $\tau\left(x^{\varsigma}\right)=\bigwedge_{j=1}^{k}\tau_j\left(x^{\varsigma}\right)$ for all $\varsigma=1,\ldots,m$.

Step 4: Test the network to see if with the k generated dendrites for all $\varsigma\in P$ that $f\left[\tau\left(x^{\varsigma}\right)\right]=d^{\varsigma}$. If this is the case then the training is complete and the network is formed. If not then additional dendrites must be grown.

Step 5: Add a new dendrite to N. Set $D=\{C_1\}$ and $I=I'=X=E=H=\varnothing$. Let $k=k+1$ as well.

Step 6: Select a misclassified pattern x^{γ} from C_0 such that $d^{\gamma}=0$ and $f\left[\tau\left(x^{\gamma}\right)\right]\neq d^{\gamma}$.

Step 7: Calculate the minimum Chebychev distance from the pattern x^{γ} to ALL patterns in D. Let $\mu=\bigwedge_{\varsigma\neq\gamma}\bigvee_{i=1}^{n}\left|x_{\varsigma}^{\gamma}-x_i^{\varsigma}\right|$ where $x^{\varsigma}\in D$.

Step 8: Keep the indices and coordinates of patterns in set D that are on the border of the hyperbox centered at x^{γ}. Let $I'=\left\{i:\left|x_i^{\gamma}-x_i^{\varsigma}\right|=\mu, x^{\varsigma}\in D,\varsigma\neq\gamma\right\}$ and $X=\left\{\left(i,x_i^{\varsigma}\right):\left|x_i^{\gamma}-x_i^{\varsigma}\right|=\mu, x^{\varsigma}\in D,\varsigma\neq\gamma\right\}$.

Step 9: Assign weights and input values for new axonal fibers in the current dendrite that provide correct classification for the misclassified pattern. For all $\left(i,x_i^{\varsigma}\right)\in X$ if $x_i^{\varsigma}<x_i^{\gamma}$ then set $w_{ik}^{1}=-x_i^{\varsigma}$ and $E=\{1\}$ or if $x_i^{\varsigma}>x_i^{\gamma}$ then set $w_{ik}^{0}=-x_i^{\varsigma}$ and $H=\{0\}$.

Step 10: Update the index sets I and L (only those input fibers and neurons needed to classify x^{γ} correctly). Set $I=I\cup I'$ and $L=E\cup H$.

Step 11: Keep the C_1 exemplars $\left\{x^\varsigma : \varsigma \neq \gamma\right\}$ that do not belong to the new region. Let $D' = \left\{x^\varsigma \in D : \forall i \in I, -w_{ik}^1 < x_i^\varsigma \, and \, / \, or \, - w_{ik}^0 > x_i^\varsigma\right\}$.

Step 12: Check to see if more axonal fibers need to be wired to the current dendrite. If $D' = \varnothing$ then proceed to Step 2. Otherwise, set $D = D'$ and return to Step 7.

3.6 Conclusions

In summary, SLMPs have several main advantages when compared to classical models such as multi-layer perceptrons. These include the ability to learn arbitrary sets within any desired degree of accuracy, drawing of closed separation surfaces in feature space, high speed convergence, and 100% correct recognition guaranteed for the training set. All of these are obtained without hidden layers, but with dendrites, which mimic more closely the biological model and are currently believed by some researchers to be the main processing elements of the neurons (Eccles, 1977). An SLMP is, therefore, a good candidate especially when one of the aspects mentioned above (such as speed) is a concern. Consequently, we use morphological perceptrons in our information retrieval system, which is described in the next chapter.

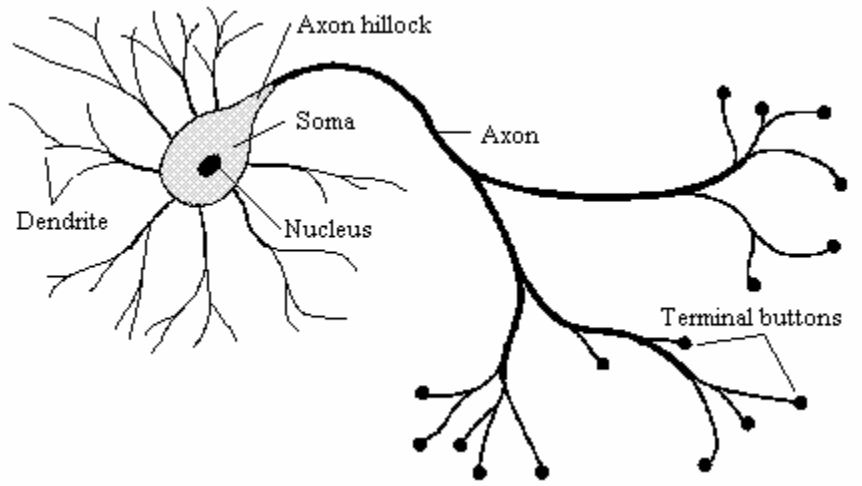

Figure 3-1. Biological neuron

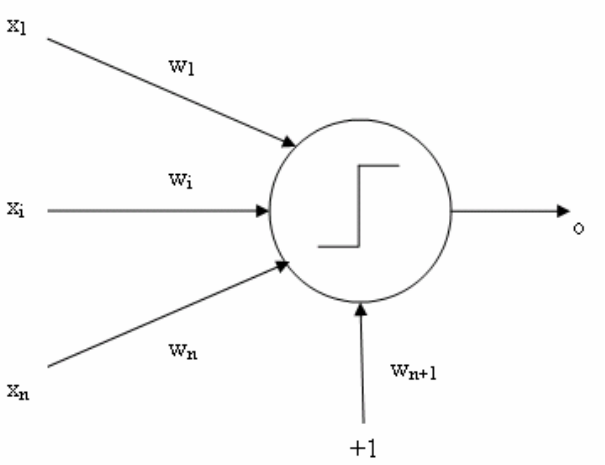

Figure 3-2. Typical perceptron model neuron

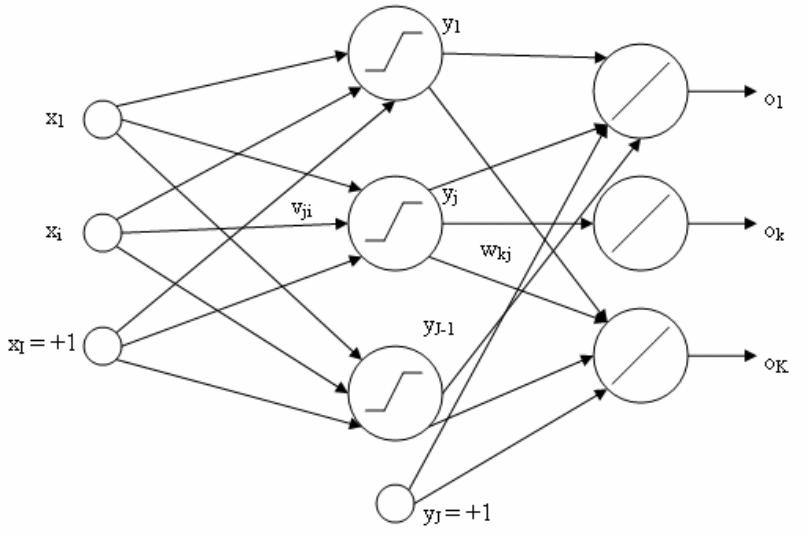

Figure 3-3. Multi-layer perceptron network

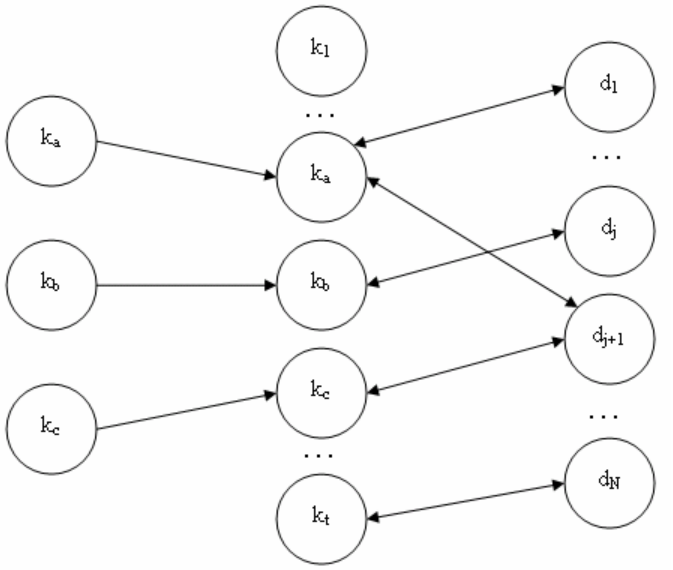

Figure 3-4. Three-layer neural network for information retrieval

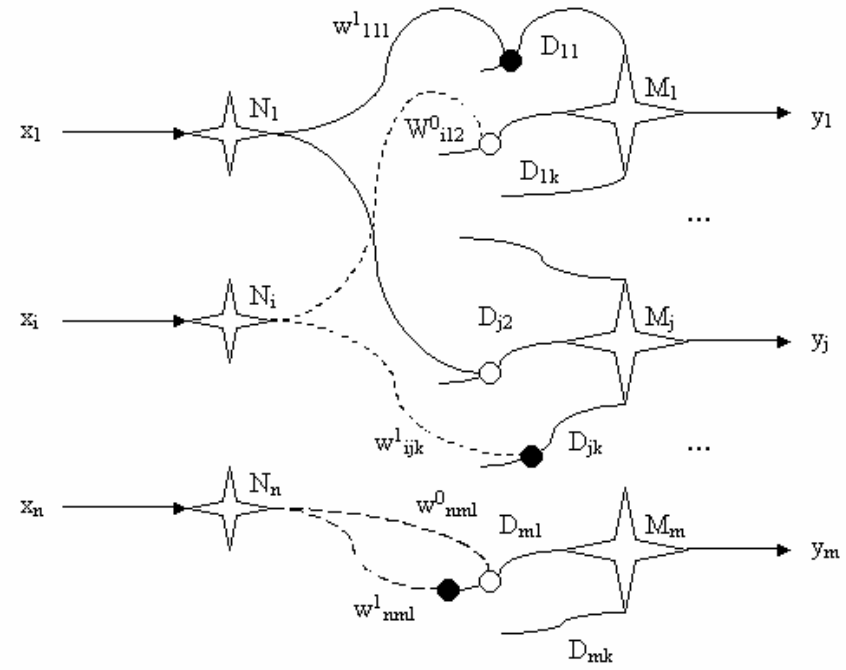

Figure 3-5. Single-layer morphological perceptron representation

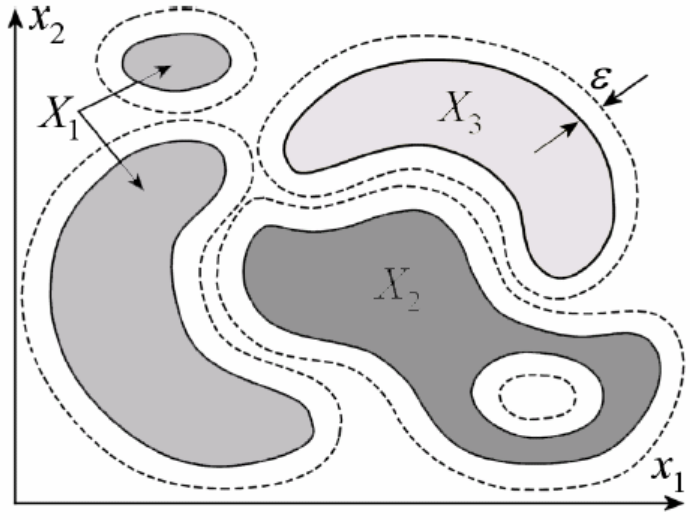

Figure 3-6. Compact sets and their border regions

CHAPTER 4
AN MNN INFORMATION RETRIEVAL MODEL

Here, we discuss a preliminary implementation of a Morphological Neural Network Information Retrieval (MNNIR) Model. This model is built on top of the Lucene IR system, an open source IR engine maintained by the Apache Software Foundation. First, we examine the components that compose the Lucene engine and then we review the MNN query engine.

4.1 The Lucene IR Engine

Lucene is a high performance, scalable, IR engine. Its indexing and searching capabilities can be built into any application, and this library is provided under the Apache Software License.

Lucene has a simple but powerful core API that provides full-text indexing and searching. It can index and search any data that can be converted to a textual format, including plain text, HTML, PDF, Microsoft Word documents, and other formats. Lucene is designed to be used as an application layer underneath search applications or can be modified to include new search technologies (Figure 4-1). The following is a more detailed look at the individual components of the engine and how the system works.

4.2 Lucene Indexing

To index with Lucene data must first be converted into a stream of plain-text tokens. Documents such as PDF, DOC, XML, or HTML must pass though a parser to extract the appropriate text tokens and convert them into Document and Field objects. Lucene's core distribution contains several common types of document parsers to assist with this task.

Once the parser has prepared the data for indexing, each Document object is run through the Analyzer to increase the document's indexing suitability. During this phase various optional operations are performed on the data, including conversion of tokens to lowercase, stopword

removal, token categorization, and stemming. This step is extensible and additional analysis techniques can be added as desired.

After the input has been analyzed, it is ready to be added to the index. Lucene stores the index in an inverted index data structure. In addition during indexing the system can apply a boosting factor to any document or field processed. By default all documents indexed have a boost factor of 1.0. By providing a boost factor to a document, all of the fields contained in the document receive an increase (or decrease for factors smaller than 1.0) in importance relative to the other documents. If a field boost factor is applied to a specific field, only that field is boosted. These boost factors are used within the scoring process.

4.3 Lucene Query Engine

Searching a Lucene index is a very fast and efficient process. In Lucene the IndexSearcher is the central class used to search the index for relevant documents. In the simplest case a search consists of a single term, which is a value matched with its field name. The term is used to build a Query object, which is then passed to IndexSearcher. The IndexSearcher's search method returns a Hits object which encapsulates access to the underlying documents returned from the search. Full documents are not immediately returned and are only fetched on demand to improve performance.

To handle more complex queries, Lucene uses of a QueryParser capable of parsing sophisticated query expressions. The QueryParser requires an analyzer and a query string to parse. Because terms were passed through an analyzer during the indexing phase, query terms should also be handed by an analyzer to ensure they will successfully match the index terms. The resulting Query object is then passed to the IndexSearcher for scoring.

Lucene's default scoring system takes into account several different factors when computing a similarity rating. Table 4-1 describes the different factors in the scoring formula.

Table 4-1. Lucene scoring formula factors

Factor	Description
tf(t in d)	Term frequency factor for the term (t) in the document d.
idf(t)	Inverse document frequency of the term.
boost(t.field in d)	Field boost, as set during indexing.
lengthNorm(t.field in d)	Normalization value of a field, given the number of terms within the field.
coord(q, d)	Coordination factor, based on the number of query terms the document contains.
queryNorm(q)	Normalization value for a query, given the sum of the squared weights of each of the query terms.

The calculated score is a raw score. Scores returned from the Hits object may not be the raw score. If the top-scoring document scores greater than 1.0, all scores are normalized from that score so they are guaranteed to be 1.0 or less.

Boost factors are built into the equation to let you affect a query or field's influence on the score. Document or field boosts are introduced explicitly in the equation using the boost(t.field in d) value set at indexing time. The default value of all boosts is 1.0. It is also possible to boost a particular Query object, allowing a specific query clause to have a greater impact on the overall score for a document.

Most of these scoring formula factors are controlled through an implementation of the Similarity class. One important feature of the scoring system is that all of these components can be extended to create custom versions. In this research, we extended several of these classes and created a custom scoring system for the MNN query engine.

4.4 Okapi BM25 Scoring

In addition to using the standard Lucene scoring formula, the MNNIR model implements a variation of the BM25 (Robertson, 1994) scoring formula. The BM25 weighting scheme, often called Okapi weighting after the system in which it was first implemented, was developed as a way of building a probabilistic model sensitive to term frequency and document length. BM stands for best match and twenty-five is simply the version number of the formula. This section does not describe the full theory behind the model, but presents the version used in the MNNIR model.

The simplest score or Retrieval Status Value (RSV) for a document d is just the idf weighting of the terms present,

$$RSV_d = \sum_{t \in q} \log idf_t \tag{4-1}$$

where q is the set of terms in the query. This formula is improved by factoring in the frequency of each term and the document length, as shown by

$$RSV_d = \sum_{t \in q} \left(\log idf_t\right) \frac{\left(k_1 + 1\right) tf_{td}}{k_1 \left(\left(1 - b\right) + b \times \left(\frac{L_d}{L_{ave}} \right) \right) + tf_{td}} . \tag{4-2}$$

Here tf_{td} is the frequency of term t in document d, and L_d and L_{ave} are the length of document d and the average document length of the whole collection. The variable k_1 (typically $k_1 < 2$) is a positive tuning parameter that calibrates the document term frequency scaling. A k_1 value of 0 corresponds to a binary model (i.e. no term frequency) and a large value corresponds to using raw term frequencies. The b variable is another tuning parameter $\left(0 \le b \le 1\right)$ that determines the scaling by document length. A value of 1 fully scales the term weight by document length, while a 0 value does no length normalization. If the query is long, a similar weighting can also be used

for the query terms. This is appropriate if the queries are paragraphs long, but is unnecessary for short queries.

The BM25 weighting formulas have been used quite widely and quite successfully across a range of document collections and search tasks. In TREC tracks these formulas have performed very well and have been adopted by many different participants. Because of their value for large document collections, one variation was included in the MNNIR scoring formulas.

4.5 MNN Query Engine

As discussed earlier in this chapter, the query engine is responsible for converting the user query into an appropriate form and performing the comparisons against the document collection to locate relevant documents. In the traditional Vector Model (see Section 2.3), the query is converted into a pseudo-document vector and the cosine angle formula is used to determine the relationship between each document and the query. A different approach is used in the MNNIR model.

First, the query is converted into a pseudo-document vector and weights are assigned to each of the query terms. The vector contains every term from the term space for the document collection and each term's weight is specified as

$$w_{i,q} = \left(0.5 + \frac{0.5\, freq_{i,q}}{m_l\, freq_{l,q}} \right) \times \log \frac{N}{n_i}, \qquad (4\text{-}3)$$

which is a variation of the Salton-Buckley query weights (Salton, 1988). The query vector is used to dynamically construct a morphological neural network to rank the documents in the collection.

The query network is a single-layer morphological perceptron with several positive dendrites designed to input a document vector and return a measure of relevance for the query

being examined. For each non-zero term in the query vector, an excitatory connection is made to the dendrite, and the connection weight is determined by the term weight of the query vector. There are no inhibitory connections to the dendrite because no positive weight for the inputs to the network exist that should not be included in the excitatory region of the network. Because the strength of the dendrite response is important, the neuron's activation function is a linear function rather than the standard hard-limiter function. The query network can be seen in Figure 4-2.

Let j represent the j^{th} excitatory connection to the dendrite in the network. For each j let $w_j^1 = -w_{j,q}$ be the connection weight to the dendrite, which is negative to provide a minimum threshold for the terms in the network. The dendrite output for any document vector x in the system is defined as

$$\tau(x) = \bigvee_{j=1}^{n} \left(x_j + w_j^1 \right). \tag{4-4}$$

The linear activation function returns an output value of $\tau(x)$ for all values greater than 0 and an output value of 0 otherwise. The larger the value of $\tau(x)$, the larger the perceived relevance of the document.

To integrate the MNN query engine into the Lucene system several modifications to the classic scoring package were needed. We created a new type of query designed to model the MNN-based query and also created a new Scorer class to encapsulate the query-time network creation and execution. The MNNQuery class makes use of the QueryParser to generate the appropriate TermQuery clauses used to score the documents. Currently, the engine still uses the DefaultSimilarity class for calculating some of the raw scoring components but can be easily extended to provide a custom weighing scheme for the MNN query engine.

At query time, the network is constructed from the provided query and each document in the collection is run through the network. Once the relevance scores for the collection have been obtained, the system ranks the documents in decreasing order of relevance and returns the results to the user. Because of the speed of the morphological neural network, the IR system can quickly and efficiently determine the relevance of the documents and filter out any unwanted parts of the collection.

4.5 Conclusions

Overall the Lucene system provided a simple yet flexible architecture that easily supported the addition of the Morphological Neural Network Query Engine. In addition Lucene's support for the TREC document collection indexing and evaluation made it the ideal choice for the development of our system. The Query Engine component designed is simple yet robust. In the next chapter we discuss the TREC Terabyte Task, and our evaluation of the MNNIR system on the GOV2 document collection.

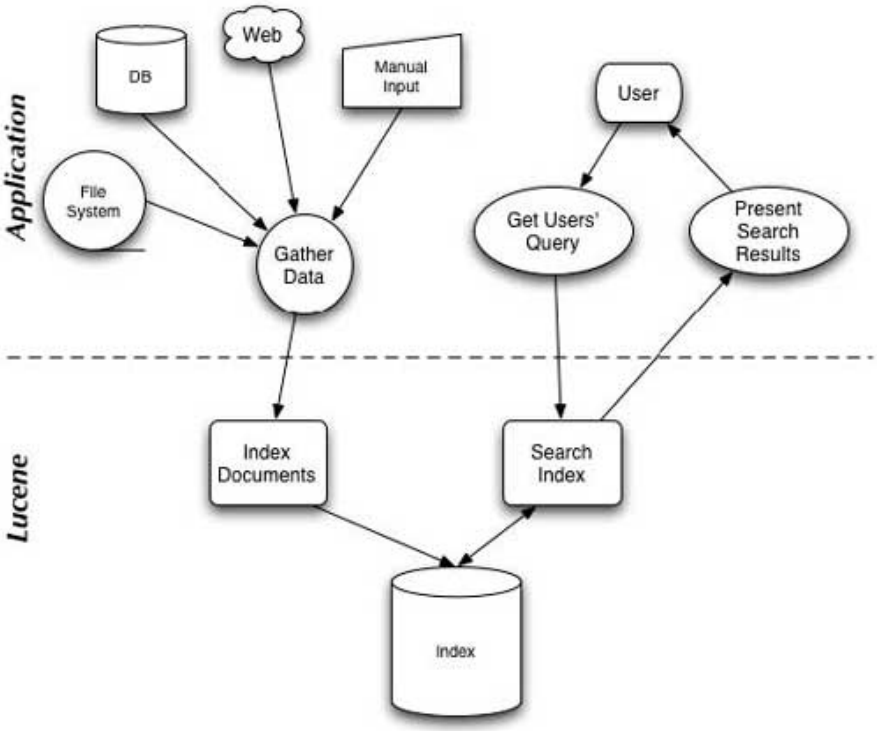

Figure 4-1. Lucene Engine

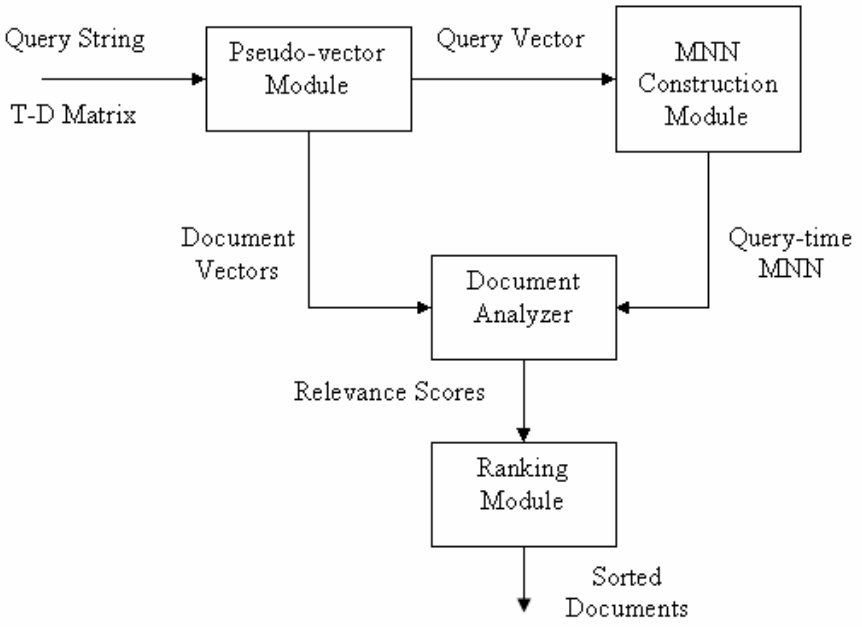

Figure 4-2. Morphological Neural Network Query Engine

CHAPTER 5
PERFORMANCE EVALUATION

To test our system we needed a document collection of significant size. We decided to use the Text Retrieval and Extraction Conference's (TREC) GOV2 collection from the conference's Terabyte Track. This collection has the size needed to demonstrate our IR engine in a real-world setting, but includes relevance markings from the past three years and results from some of the top research groups in the field for comparison.

5.1 Document Collection

The GOV2 collection is a collection of Web data crawled from Web sites in the .gov domain during early 2004. This collection contains a large portion of the crawlable pages in gov and is 426GB in size, containing 25 million documents. Document formats in the collection include text, HTML, PDF, Word, and Postscript documents. Any documents over 256 kB in size have been truncated to 256 kB. The average document size is just under 18 kB. We chose to use this collection because it is the collection used in the TREC Terabyte track, so we can easily test our data against results from other well-established IR research groups. The next section describes in more detail the Terabyte Track.

5.2 Testing Environment

The primary goal of the Terabyte Track is to develop an evaluation methodology for terabyte-scale document collections. In addition, the track examines efficiency and scalability issues, which can be studied more easily in the context of a larger collection. This track uses the GOV2 collection. While this collection is less than a full terabyte in size, it is considerably larger than the collections used in previous TREC tracks. In future years, the collection will be expanded to include data from other sources.

For evaluation of our system we use the adhoc retrieval task from the TREC Terabyte Track. The adhoc retrieval task investigates the performance of systems that search a static set of documents using previously-unseen topics. For each topic, participants create a query and generate a ranked list of documents. For the 2006 task, NIST created and assessed 50 new topics. An example is provided in Figure 5-1. In addition, the 99 topics created in 2004 and 2005 (topics 701-800) were re-used for automatic runs.

As is the case for most TREC adhoc tasks, a topic describes the underlying information need in three forms: a query title, a description, and a narrative. The title field contains a keyword query, similar to a query that might be entered into a Web search engine. For the example query the title field is "Big Dig pork" which is a reference to excessive spending on Boston's Central Artery project. The description field provides a longer statement of the topic requirements, in the form of a complete sentence or question. The sample description asks why the Boston Central Artery project is often characterized as a pork project. The narrative, which may be a full paragraph in length, supplements the other two fields and provides additional information required to specify the nature of a relevant document. For the sample the narrative field focuses on relevant documents that describe the Bid Dig as a waste of taxpayer money and also mentions documents that focus on individual contractors or cost-reports are not considered relevant.

For the adhoc task, an experimental run consisted of the top 10,000 retrieved documents for each topic. To generate a run, participants could create queries automatically or manually from the topics. For a run to be considered automatic it must be created from the topics without any human intervention. All other runs are manual. Manual runs used only the 50 new topics; automatic runs used all 149 topics from 2004–2006.

For most experimental runs, participants could use any or all of the topic fields when creating queries from the topic statements. However, a group submitting any automatic run was required to submit at least one automatic run that used only the title field of the topic statement. The remaining automatic runs could include information from all of the fields in the TREC query. Manual runs were encouraged, since these runs often add relevant documents to the evaluation pool that are not found by automatic systems using current technology. For the manual runs judged at TREC 2006 a noticeable increase in precision was observed over the automatic runs from the same participant. For our research we focused on automatic runs only and did not consider the manual runs in more detail.

Runs were pooled by NIST for judging. The details of the pooling process in 2006 differ substantially from previous years, and from previous TREC adhoc tasks. Assessors used a three-way scale of not relevant, relevant, and highly relevant. A document is considered relevant if any part of the document contains information that the assessor would include in a report on the topic. It is not sufficient for a document to contain a link that appears to point to a relevant Web page, the document itself must contain the relevant information.

It was left to the individual assessors to determine their own criteria for distinguishing between relevant and highly relevant documents. For the purpose of computing effectiveness measures that require binary relevance judgments, the relevant and highly relevant documents are combined into a single “relevant” set.

In addition to the top 10,000 documents for each run, details were collected about the hardware and software configuration used to generate them, including performance measurements such as total query processing time. For total query processing time, groups were asked to report the time required to return the top 20 documents, not the time to return the top

10,000. It was acceptable to execute a system twice for each run, once to generate the top 10,000 documents and once to measure the execution time for the top 20 documents, provided that the top 20 documents were the same in both cases.

5.3 Other TREC Systems

A variety of organizations attended the TREC 2006 conference. To better understand the other systems our MNNIR engine is being compared against, a brief summary of techniques used for the 2006 Terabyte Track are discussed here.

The Arctic Region Supercomputing Center examined the GOV2 corpus and designed a system to divide the documents according to web domain and for each topic using a Cray XD1 and the Amberfish IR engine (Fallen, 2006). The results from each domain were merged into single ranked list. The mean average precision scores of the results from two different merge algorithms applied to the domain-divided GOV2 collection and a randomized domain-divided collection are compared with a 2-way analysis of variance.

The University of Amsterdam investigated the feasibility of running terabyte scale information retrieval tasks on top of a relational database engine. The MonetDB/X100 database system is designed for high performance on data-intensive workloads like the GOV2 corpus (Héman et al., 2006). Their system used pre-calculated BM25 scores for the document collection and a two-pass querying approach to prune documents that were unlikely to be classified as relevant to improve efficiency. Another University of Amsterdam team experimented with home-grown extensions to the Lucene search engine to provide language-modeling features.

Dublin City University's experiments on the Terabyte track concentrated on the evaluation of a sorted inverted index. The aim of this approach is to sort the postings within each posting list in such a way that allows only a limited number of postings to be processed from each list, while at the same time minimizing the loss of effectiveness in terms of query precision

(Ferguson, 2006). To improve search efficiency the system pre-calculates BM25 scores for term-document pairs in the corpus.

Ecole Nationale Superieure des Mines de Saint Etienne examined fuzzy term proximity method applied to the Terabyte Task (Mercier, 2006). Fuzzy proximity's main feature is based on the idea that the closer the query terms are in a document, the more relevant this document is. Term proximity is controlled with an influence function, which, given a query term and a document, associates with each position in the text a value dependant of the distance of the nearest occurrence of this query term. To model proximity, this function is decreased with distance.

The IBM Haifa Research Lab evaluated the corpus using a document at-a-time (DAAT) approach (Carmel, 2006). The algorithm iterates in parallel over query terms and identifies candidate documents using a preliminary evaluation taking into account only partial information on term occurrences and no query independent factors. Once a candidate document is identified, it is fully evaluated, and its exact score is computed. If the result of this fast and rough evaluation is above a certain threshold, varied dynamically during the execution, then a full evaluation is performed and the exact score is computed.

The Max Planck Institute retrieves the best-scoring (so-called top-k) answers to a multi-keyword query using an aggregation of per-keyword scores over pre-computed index lists, one for each keyword in the query, which are sorted in descending order of per-keyword scores (Bast, 2006). This threshold algorithm approach aggregates scores on the fly and computes a lower bound for the total score of the current rank-k result document and an upper bound for the total scores of all other candidate documents. It is often able to terminate the index scans long

before it reaches the bottom of the index lists, namely, when the lower bound for the rank-k result, the threshold, is at least as high as the upper bound for all other candidates.

Northeastern University introduced the Hedge algorithm for metasearch, which effectively combines the ranked lists of documents returned by multiple retrieval systems in response to a given query and learns which documents are likely to be relevant from a sequence of on-line relevance judgments (Aslam, 2006). It has been demonstrated that the Hedge algorithm is an effective technique for metasearch, often significantly exceeding the performance of standard metasearch and IR techniques over small TREC collections.

The University of Melbourne used an impact-based ranking for their query engine (Anh, 2006). Impact ranking specifies a method to map each term-document pair to an integer impact value representing the importance of that term in the document. Document and query term impacts normally correlate with the term frequencies in documents or queries, but might or might not also depend on some collection-wide statistics such as collection frequency. The mapping from each term-document pair to the corresponding impact is done locally within each document during the indexing phase. In contrast to conventional information retrieval systems, where raw term statistics are stored in indexes, and most of the similarity calculation is carried out during query processing, impact based ranking allows computation of all of the document term impacts while the index is being constructed. The document term impacts are then stored in the index, reducing the computational burden during query processing. Normally, the indexes are impact-sorted.

The University of Waterloo focused on two techniques for TREC 2006: static index pruning and result re-ranking (Büttcher, 2006). Static index pruning is a method to limit the postings in each inverted list to those that have the greatest impact on a document's score when

encountered during query processing. By applying this technique to an inverted index, it is possible to dramatically decrease its size, which then leads to faster query processing at the cost of decreased retrieval effectiveness. By pruning more or less aggressively, the right point in this efficiency vs. effectiveness trade-off can be chosen. Re-ranking based on relevance models is similar to traditional query expansion by means of pseudo-relevance feedback, but does not require a predefined limit for the number of expansion terms, because it replaces the original query by an entire language model, without any bounds on the number of terms it contains.

The University of Massachusetts at Amherst continued their research with the Indri search engine (Strohman, 2005). The retrieval model implemented in Indri is an enhanced model that combines the language modeling and inference network approaches to information retrieval. The resulting model allows structured queries similar to those used in INQUERY to be evaluated using language modeling estimates within the network, rather than tf.idf estimates. In addition Peking University also conducted some experiments using the Indri search engine. RMIT examined the performance of a difference language-model engine called Zettair. The University of Glasgow continued experiments in Divergence from Randomness using their Terrier search engine.

5.4 Results

During evaluation of the MNNIR system we tested three different weighing schemes on the GOV2 document collection: the Lucene TF-IDF formula, Okapi BM25, and pivoted normalization. Each formula was modified slightly to function with the Lucene raw frequency values and placed into an appropriate subclass of the Lucene Scorer class. To stay consistent with the other TREC results available our system made use of only the title field of each query. For the query in Figure 5-1 the terms big, dig, and pork were extracted and passed to the query engine. Table 5-1 shows the results using the standard single-feature MNN query engine

measured using our three standard metrics (bpref, p@20, MAP) as well as two alternative metrics (p@10 and R-prec). For each of the metrics used, higher values indicate a better result. The Okapi BM25 scoring formula outperformed the other two scoring models for most topics in the collection. In addition, because of the higher overall stability of BM25, further tests using the TF-IDF variation or pivoted normalization were not run and BM25 was used exclusively.

Table 5-1. Metrics for weighting with the GOV2 corpus

Weighting Scheme	bpref	p@20	MAP	p@10	R-prec
TF-IDF	0.2532	0.2601	0.1525	0.2597	0.2074
Pivoted Normalization	0.2845	0.2430	0.1662	0.4160	0.2093
Okapi BM25	0.2928	0.2780	0.1737	0.4280	0.2174

The retrieval baseline is based on a standard document-ordered frequency index, without any positional information. Documents are ranked by Okapi BM25 as discussed above and terms are stemmed using Porter's stemmer algorithm. The retrieval effectiveness of this method for the ad-hoc retrieval varying BM25's document length normalization parameter b and document term frequency parameter k_1 is shown in Table 5-2. The precision of the table is represented by the P@20 metric, with higher values representing a higher precision. It is interesting that the default value $(b = 0.75)$ does not produce the best results. For the ad-hoc retrieval tasks search results are best if $(b \approx 0.5)$ and degrade as b is increased or decreased. In addition when considering the term frequency parameter, search results are best if $(k_1 \approx 1.25)$ and degrade as k_1 is increased or decreased. What this means is that on average a document that is relevant to one of the ad-hoc topics tends to be substantially larger than a typical document in the collection. The optimal tuning parameter values were used to test the various features of the MNN query engine.

Table 5-2. Tuning results for BM25 parameters

	b = 0.4	b = 0.5	b = 0.6	b = 0.7	b = 0.75	b = 0.8
k_1 = 1.0	0.3645	0.3723	0.3684	0.3572	0.3504	0.3226
k_1 = 1.25	0.3719	0.3771	0.3734	0.3697	0.3660	0.3294
k_1 = 1.5	0.3346	0.3394	0.3360	0.3327	0.3294	0.2965
k_1 = 1.75	0.3012	0.3054	0.3024	0.2994	0.2965	0.2668
k_1 = 2.0	0.2718	0.2749	0.2722	0.2695	0.2668	0.2401

Our research also examined the effect of several feature vectors in processing queries on the GOV2 document collection as shown in Table 5-3. In this and future tables the p@10 and R-prec metrics are not listed because the official TREC results only use bpref, p@20, and MAP so we were unable to directly compare our results with other participants. For all three metrics higher values represent better results. We used the Lucene TF-IDF model as our baseline for comparison. Using (Equation 4-4) the first dendrite model tested used a single positive dendrite weighted by the terms in the query. This gained almost a 15% improvement over the baseline model in terms of bpref measurements and almost a 40% improvement in overall speed. We also examined two feature dendrites using two-term and three-term combinations of weights which yielded additional improvements over the single term model with only a slight drop in performance speed. It is interesting to note that there is only a slight increase in effectiveness when the network is increased from two-term groupings to three-term groupings. In addition by blending these models into a single network we see an overall improvement of just under 37% in bpref scores over the baseline while still running over 34% faster than the baseline model.

Table 5-3. The MNN Features on the GOV2 corpus

Features	bpref	p@20	MAP	CPUs	Time(s)
Lucene baseline	0.2532	0.2601	0.1525	1	664
Single terms	0.2928	0.2780	0.1737	1	397
Term pairs	0.3205	0.3660	0.1959	1	413
Term triples	0.3328	0.3950	0.2033	1	425
Blended terms	0.3466	0.4002	0.2273	1	438

Table 5-4 provides a summary of the results obtained on the title-only automatic runs from the 2006 TREC conference sorted by run time. Only the best run from each group is shown. Data from the MNNIR system has been added for comparison to the original runs from the conference. The first two columns of each table identify the group and run. The next three columns provide the values of three standard effective measures for each run: bpref, precision at 20 documents (p@20), and mean average precision (MAP). The last two columns list the number of CPUs used to generate the run and the total query processing time.

The top-scoring automatic runs were generated using various retrieval methods, including Okapi BM25 and language modeling approaches. Many of these runs took features such as phrases, term proximity, inference networks, and matches in the title field into account during ranking. The University of Massachusetts at Amherst (and Peking University by using the same engine) were able to perform extremely well by building inference networks for their queries. In addition another strong entry, Ecole Nationale des Mines de Saint Etienne, showed improvements based on use of a fuzzy term proximity. Of particular note is the prevalence of pseudo-relevance feedback, which substantially improved performance for most groups. Both the University of Waterloo and IBM Haifa made extensive use of this feedback, both scoring in the top five positions in all three metrics tested. On the other hand, none of the top-eight runs used anchor text, and only one (Coveo Enterprise Search) used link analysis techniques.

Table 5-4. Time-sorted results from the TREC 2006 Terabyte Track Adhoc task

Group	bpref	p@20	MAP	CPUs	Time(s)
umelbourne.ngoc-anh	0.3682	0.5130	0.3039	1	25.25
max-planck	0.2849	0.4270	0.1805	2	38
polytechnicu.suel	0.3073	0.3920	0.2274	1	60
lowlands-team	0.3361	0.4780	0.2770	1	60.3
sabir.buckley	0.2434	0.3250	0.1361	1	77
coveo.soucy	0.3886	0.5440	0.3296	5	135
MNNIR	0.3466	0.4002	0.2273	1	438
rmit.scholer	0.3726	0.4800	0.3056	2	466.5
dublincityu.gurrin	0.3509	0.5090	0.2695	1	495
tsinghuau.zhang	0.3432	0.4600	0.2858	4	560
uwaterloo-clarke	0.4251	0.5570	0.3392	1	964
uamsterdam.ilps	0.3528	0.4850	0.2958	2	2394
umilano.vigna	0.3774	0.4510	0.2882	4	3000
ibm.carmel	0.4002	0.5670	0.3506	1	3375
Average	0.3502	0.4612	0.2759	N/A	24374.88
hummingbird	0.4172	0.5820	0.3452	1	36000
umass.allen	0.4229	0.5410	0.3687	1	38737
pekingu.yan	0.4193	0.5150	0.3737	4	56160
ecole-de-mines	0.3942	0.5170	0.3120	2	68344
northeasternu	0.2568	0.3460	0.1771	4	110000
ualaska	0.1463	0.0550	0.0541	108	120000
uglasgow.ounis	0.3995	0.5400	0.3456	1	N/A

When compared to the TREC 2006 participants several conclusions can be drawn about the MNNIR model. First the MNNIR model is significantly faster than most of the other participants in Terabyte track (Table 5-4), including a very large difference in speed when compared to the strongest systems. This is especially impressive considering that the MNNIR model runs without using any pre-calculated information other than the raw frequencies recorded at index time. While there are several systems that post a faster time than MNNIR, all of those systems rely on a collection of pre-calculated and/or pre-sorted values. In terms of effectiveness (Tables 5-5 through 5-8), the MNNIR model comes in slightly under the average bpref score of 0.3503, under the average p@20 score of 0.4612, and under the average MAP score of 0.2759. While these are not the ideal results, they are promising. In each of the three tables (shown in gray) our results come within one standard deviation of the average. In addition the MNNIR

model does not apply any advanced techniques (query expansion, term proximity calculations, pseudo-relevance feedback) during the scoring of documents for queries on the collection. Adding some or all of these other techniques would most likely result in further improvement to the system's performance with only minimal reduction in the overall speed.

Table 5-5. Bpref-sorted results from the TREC 2006 Terabyte Track Adhoc task

Group	bpref	p@20	MAP	CPUs	Time(s)
uwaterloo-clarke	0.4251	0.5570	0.3392	1	964
umass.allen	0.4229	0.5410	0.3687	1	38737
pekingu.yan	0.4193	0.5150	0.3737	4	56160
hummingbird	0.4172	0.5820	0.3452	1	36000
ibm.carmel	0.4002	0.5670	0.3506	1	3375
uglasgow.ounis	0.3995	0.5400	0.3456	1	N/A
ecole-de-mines	0.3942	0.5170	0.3120	2	68344
coveo.soucy	0.3886	0.5440	0.3296	5	135
umilano.vigna	0.3774	0.4510	0.2882	4	3000
rmit.scholer	0.3726	0.4800	0.3056	2	466.5
umelbourne.ngoc-anh	0.3682	0.5130	0.3039	1	25.25
uamsterdam.ilps	0.3528	0.4850	0.2958	2	2394
dublincityu.gurrin	0.3509	0.5090	0.2695	1	495
Average	0.3502	0.4612	0.2759	N/A	24374.88
MNNIR	0.3466	0.4002	0.2273	1	438
tsinghuau.zhang	0.3432	0.4600	0.2858	4	560
lowlands-team	0.3361	0.4780	0.2770	1	60.3
polytechnicu.suel	0.3073	0.3920	0.2274	1	60
max-planck	0.2849	0.4270	0.1805	2	38
northeasternu	0.2568	0.3460	0.1771	4	110000
sabir.buckley	0.2434	0.3250	0.1361	1	77
ualaska	0.1463	0.0550	0.0541	108	120000

As evident from our results, the MNNIR system is clearly capable of handling a sizable dataset such as GOV2 while still remaining competitive with other top research groups in the field.

Table 5-6. P@20-stored results from the TREC 2006 Terabyte Track Adhoc task

Group	bpref	p@20	MAP	CPUs	Time(s)
hummingbird	0.4172	0.5820	0.3452	1	36000
ibm.carmel	0.4002	0.5670	0.3506	1	3375
uwaterloo-clarke	0.4251	0.5570	0.3392	1	964
coveo.soucy	0.3886	0.5440	0.3296	5	135
umass.allen	0.4229	0.5410	0.3687	1	38737
uglasgow.ounis	0.3995	0.5400	0.3456	1	N/A
ecole-de-mines	0.3942	0.5170	0.3120	2	68344
pekingu.yan	0.4193	0.5150	0.3737	4	56160
umelbourne.ngoc-anh	0.3682	0.5130	0.3039	1	25.25
dublincityu.gurrin	0.3509	0.5090	0.2695	1	495
uamsterdam.ilps	0.3528	0.4850	0.2958	2	2394
rmit.scholer	0.3726	0.4800	0.3056	2	466.5
lowlands-team	0.3361	0.4780	0.2770	1	60.3
Average	0.3502	0.4612	0.2759	N/A	24374.88
tsinghuau.zhang	0.3432	0.4600	0.2858	4	560
umilano.vigna	0.3774	0.4510	0.2882	4	3000
max-planck	0.2849	0.4270	0.1805	2	38
MNNIR	0.3466	0.4002	0.2273	1	438
polytechnicu.suel	0.3073	0.3920	0.2274	1	60
northeasternu	0.2568	0.3460	0.1771	4	110000
sabir.buckley	0.2434	0.3250	0.1361	1	77
ualaska	0.1463	0.0550	0.0541	108	120000

Table 5-7. MAP-sorted results from the TREC 2006 Terabyte Track Adhoc task

Group	bpref	p@20	MAP	CPUs	Time(s)
pekingu.yan	0.4193	0.5150	0.3737	4	56160
umass.allen	0.4229	0.5410	0.3687	1	38737
ibm.carmel	0.4002	0.5670	0.3506	1	3375
uglasgow.ounis	0.3995	0.5400	0.3456	1	N/A
hummingbird	0.4172	0.5820	0.3452	1	36000
uwaterloo-clarke	0.4251	0.5570	0.3392	1	964
coveo.soucy	0.3886	0.5440	0.3296	5	135
ecole-de-mines	0.3942	0.5170	0.3120	2	68344
rmit.scholer	0.3726	0.4800	0.3056	2	466.5
umelbourne.ngoc-anh	0.3682	0.5130	0.3039	1	25.25
uamsterdam.ilps	0.3528	0.4850	0.2958	2	2394
umilano.vigna	0.3774	0.4510	0.2882	4	3000
tsinghuau.zhang	0.3432	0.4600	0.2858	4	560
lowlands-team	0.3361	0.4780	0.2770	1	60.3
Average	0.3502	0.4612	0.2759	N/A	24374.88
dublincityu.gurrin	0.3509	0.5090	0.2695	1	495
polytechnicu.suel	0.3073	0.3920	0.2274	1	60
MNNIR	0.3466	0.4002	0.2273	1	438
max-planck	0.2849	0.4270	0.1805	2	38
northeasternu	0.2568	0.3460	0.1771	4	110000
sabir.buckley	0.2434	0.3250	0.1361	1	77
ualaska	0.1463	0.0550	0.0541	108	120000

```
<top>
<num> Number: 835
<title> Big Dig pork
<desc> Description:
Why is Boston’s Central Artery project, also known as "The Big
Dig", characterized as "pork"?
<narr> Narrative:
Relevant documents discuss the Big Dig project, Boston’s Central
Artery Highway project, as being a big rip-off to American
taxpayers or refer to the project as "pork". Not relevant are
documents which report fraudulent acts by individual
contractors. Also not relevant are reports of cost-overruns on
their own.
</top>
```

Figure 5-1. Sample TREC Query

CHAPTER 6
CONCLUSIONS

In this research, we have shown significant progress towards developing an information retrieval model using morphological neural networks. We created a morphological neural network information retrieval (MNNIR) model showing meaningful performance gain in terms of speed while providing similar effectiveness in the query evaluation process. We summarize our contributions in Section 6.1 and discuss directions for future work in Section 6.2.

6.1 Contributions

Our objective was to design an intelligent information retrieval model using the morphological neural network as an alternative to the traditional retrieval techniques. The main contributions of the dissertation to information retrieval research are as follows:

Morphological Neural Network Query Engine: By replacing the more traditional vector-based querying model with a system using a dynamic query-time morphological network, we were able to improve the overall retrieval speed of the information retrieval engine while maintaining a high level of accuracy in the retrieved documents. The morphological neural network lends itself to this process because of the simplicity of its calculations and it can be easily constructed at query-time without any need for a traditional artificial neural network training period.

Analysis of features for Morphological Neural Network querying: In our baseline system we established a single feature used to construct the morphological neural network used for query evaluation. Upon further analysis we determined that more features were likely to increase system performance. By analyzing several basic features and testing them against the TREC data, we discovered which features improved retrieval accuracy which inhibited system performance. We found that a significant increase in performance was obtained by including

features to take advantage to larger term combinations. In addition, we found that using the Okapi BM25 scoring formula in conjunction with the term features gained addition ground over the conventional TF-IDF formula. By applying this knowledge to the query engine, we observed further increases in the system's ability to retrieve relevant documents. When compared to the top 20 systems from the TREC 2006 conference our results were right under the average for all three metrics tested, and in all three cases our results were within one standard deviation.

6.2 Future Work

While we have laid the groundwork for using morphological neural networks for information retrieval, there still remain several areas of this field that still must be examined.

One task that still needs to be completed is the exploration of the merits of using the MNNIR system in conjunction with query expansion. We observed that several of the top performing groups at TREC 2006 made use of some variation of query expansion in their systems. Both the University of Amsterdam and the Max Planck Institute used multi-pass systems to refine their results using the initial result set. In addition the University of Waterloo performed results re-ranking using pseudo-relevance feedback. It is possible that using some type of thesaurus (such as WordNet) or other term-to-term relations from the document collection to expand the query used by the Query Engine will yield additional improvements in the precision of the MNNIR system.

In addition to query expansion we want to examine the potential benefits offered by latent semantic indexing techniques. One potential issue with this approach is the matrix factorization step associated with a large dataset can be computationally prohibitive. Incremental singular-value decomposition approaches allow for more efficient computation while maintaining almost all of the accuracy of traditional systems (Ye et al., 2004). By applying an incremental singular-

value decomposition technique which can be applied to the index we hope to further increase the performance of the MNNIR system.

Another extension to this work would be to explore advanced noise reduction techniques of morphological neural networks to improve the IR engine. The current system uses no such techniques. Some possible techniques include hyperbox obstruction, hyperbox expansion, and hyperbox rotation (Iancu, 2005). It may be possible to gain some improvement in precision by applying some or all of these techniques during the construction of the morphological neural network.

APPENDIX
SAMPLE TREC EVALUATION

Table A-1. Sample TREC evaluation data

```
_____________________________________________
Number of queries  = 50
Retrieved          = 48831
Relevant           = 2279
Relevant retrieved = 1814

_____________________________________________
Average Precision: 0.2977
R Precision      : 0.3271

_____________________________________________
Precision at    1: 0.5600
Precision at    2: 0.5500
Precision at    3: 0.5400
Precision at    4: 0.5150
Precision at    5: 0.5000
Precision at   10: 0.4540
Precision at   15: 0.4200
Precision at   20: 0.3920
Precision at   30: 0.3280
Precision at   50: 0.2692
Precision at  100: 0.1964
Precision at  200: 0.1329
Precision at  500: 0.0658
Precision at 1000: 0.0363

_____________________________________________
Precision at   0%: 0.7325
Precision at  10%: 0.6022
Precision at  20%: 0.5180
Precision at  30%: 0.4212
Precision at  40%: 0.3616
Precision at  50%: 0.3026
Precision at  60%: 0.2292
Precision at  70%: 0.1696
Precision at  80%: 0.1151
Precision at  90%: 0.0582
Precision at 100%: 0.0151

_____________________________________________
Average Precision: 0.2977
```

LIST OF REFERENCES

Anh V., Webber W., Moffat A. (2006). Melbourne University at the 2006 Terabyte Track. *Proceedings of the 15th Annual Text Retrieval Conference*, Annapolis, MD.

Arbib, M. (1998). *The Handbook of Brain Theory and Neural Networks*. MIT Press, Cambridge, MA.

Aslam J., Pavlu V., Rei C. (2006). The Hedge Algorithm for Metasearch at TREC 2006. *Proceedings of the 15th Annual Text Retrieval Conference*.

Bar-Yossef Z., Berg A., Chien S., Fakcharoenphol J., Weitz D. (2000). Approximating Aggregate Queries about Web Pages via Random Walks. *Proceedings of the 26th International Conference on Very Large Data Bases*, p. 535-544.

Bast H., Majumdar D., Schenkel R., Theobald M., Weikum G. (2006). IO-Top-k at TREC 2006: Terabyte Track. *Proceedings of the 15th Annual Text Retrieval Conference*.

Birkhoff, G. (1964). *Lattice Theory*. American Mathematical Society, v. XXV.

Broder A., Kumar R., Maghoul F., Ragavan P., Rajagopalan S., Stata R., Tomkins A., Wiener J. (2000). Graph Structure in the Web: Experiments and Models. *In Proceedings of the Ninth International Conference on The World Wide Web*, p. 309-320.

Buckley C., Voorhees M. (2000). Evaluating evaluation measure stability. *In Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, p. 33–40.

Buckley C., Voorhees M. (2004). Retrieval evaluation with incomplete information. *In Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, p. 25-32.

Büttcher S., Clarke C., Yeung P. (2006). Index Pruning and Result Reranking: Effects on Ad-Hoc Retrieval and Named Page Finding. *Proceedings of the 15th Annual Text Retrieval Conference*.

Carmel D., Amitay E. (2006). Juru at TREC 2006: TAAT versus DAAT in the Terabyte Track. *Proceedings of the 15th Annual Text Retrieval Conference*.

Deerwester S., Dumais S., Furnas G., Landauer T., Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, v. 41 n. 6, p. 391-407.

Eccles, J. (1977). *The Understanding of the Brain*. McGraw-Hill, New York, NY.

Fallen T., Newby G. (2006). Partitioning the Gov2 Corpus by Internet Domain Name: A Result-set Merging Experiment. *Proceedings of the 15th Annual Text Retrieval Conference*.

Faloutsos C. (1985). Access methods for text. *ACM Computing Surveys*, v. 17 n. 1, p. 49-74.

Ferguson P., Smeaton A., Wilkins P. (2006). Dublin City University at the TREC 2006 Terabyte Track. *Proceedings of the 15th Annual Text Retrieval Conference*.

Frakes W. B., Baeza-Yates, R. (1992). *Information Retrieval: Data Structures & Algorithms*. Prentice Hall, Englewood Cliffs, NJ.

Furnas G., Deerwester S., Dumais S., Landauer T., Harshman R., Streeter L., Lochbaum K. (1988). Information Retrieval Using a Singular Value Decomposition Model of Latent Semantic Structure. *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, p. 465-480.

Gori M., Scarselli F. (1998). Are Multilayer Perceptrons Adequate for Pattern Recognition and Verification? *IEEE Trans. On Pattern Analysis and Machine Intelligence*, v. 20, n. 11, p. 1121-1132.

Héman S., Zukowski M., de Vries A., Boncz P. (2006). MonetDB/X100 at the 2006 TREC TeraByte Track. *Proceedings of the 15th Annual Text Retrieval Conference*.

Huberman B., Adamic L. (1999). Growth Dynamics of the World Wide Web. *Nature*, v. 401, n. 6749, p. 131.

Iancu L. (2005). Lattice Algebra Approach to Neural Computation. *Ph.D. Dissertation*, University of Florida.

Kohonen, T. (1989). *Self-organization and Associative Memory, 3rd Ed*. Springer-Verlag. Berlin, Germany.

Korfhage, R. (1997). *Information Storage and Retrieval*. John Wiley & Sons, New York, NY.

Kwok, K. L. (1989). A Neural Network for Probabalistic Information Retrieval. *Proceedings of the 12th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, p. 21-30.

Lawrence S., Giles, C. (1998). Searching the World Wide Web. *Science*, v. 280, n. 5360, p. 98-100.

Lawrence S., Giles C. (1999). Accessibility of Information on the Web. *Nature*, v. 400, n. 6740, p. 107-109.

Lin X., Soergel D., Marchionini G. (1991). A Self-organizing Semantic Map for Information Retrieval. *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, p. 262-269.

Mercier A., Beigbeder M. (2006). Fuzzy term proximity with boolean queries at 2006 TREC Terabyte task. *Proceedings of the 15th Annual Text Retrieval Conference*.

Ogawa Y., Morita T., Kobayashi K. (1991). A Fuzzy Document Retrieval System Using the Keyword Connection Matrix and a Learning Method. *Fuzzy Sets and Systems*, v. 39, p. 163-179.

Ounis I., Amati G., Plachouras V., He B., Macdonald C., Johnson D. (2005). Terrier Information Retrieval Platform. *Proceedings of the 27th Annual European Conference on Information Retrieval (ECIR 2005)*. p. 156-163.

Pitkow J., Pirolli P. (1987). Life, Death, and Lawfulness on the Electronic Frontier. *Proceedings of the SIGCHI conference on Human factors in computing systems*, p. 383-390.

Porter M. (1980). An Algorithm for Suffix Stripping, *Program*, v. 14, n. 3, p.130-137.

Raghaven V. V., Jung G. S., Bollman P. (1989). A Critical Investigation of Recall and Precision as Measures of Retrieval System Performance. *ACM Transactions on Office and Information Systems*, v. 7, n. 3, p. 205-229.

Ribeiro-Neto B., Muntz R. (1996). A Belief Network Model for IR. *Proceedings Of the 19th Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, p. 253-260.

Ritter G. X., Sussner P. (1996). An Introduction to Morphological Neural Networks. . *Proceedings of the 13th International Conference on Pattern Recognition*, p. 709-717.

Ritter G. X., Beavers T. (1999). Morphological Perceptrons. *IEEE Trans. on Neural Networks*, v. 1, p. 605-610.

Ritter G. X., Urcid G. (2003). Lattice algebra approach to single neuron computation. *IEEE Trans. on Neural Networks*, v. 14, n. 2, p. 282-295.

Ritter G. X., Iancu L., Urcid G. (2003). Morphological Perceptrons with Dendritic Structure. *Proceedings of the 12th IEEE International Conference on Fuzzy Systems*, p. 1296-1301.

Ritter H., Kohonen, T. (1989). Self-organizing Semantic Maps. *Biological Cybernetics*, v. 61, n. 4, p. 241-254.

Robertson S., Walker S., Jones S., Hancock-Beaulieu M., Gatford M. (1994). Okapi at TREC-3. *Proceedings of the 3rd Annual Text Retrieval Conference*.

Robertson S., Sparck Jones K. (1976). Relevance Weighting of Search Terms. *Journal of the American Society for Information Sciences*, v. 27, n. 3, p. 129-146.

Rocchio J. J. (1971). Relevance Feedback in Information Retrieval. *The SMART Retrieval System – Experiments in Automatic Document Processing*. Prentice Hall, New York, NY.

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, v. 65, n. 6, p. 386-408.

Rumelhart D., Hinton G., Williams R. (1986). Learning Internal Representations by Error Propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. v. 1, MIT Press, Cambridge, MA, p. 318-362.

Salton G., Buckley C. (1988). Term-weighting Approaches in Automatic Retrieval. *Information Processing & Management*, v. 24, n. 5, p. 513-523.

Salton G., Fox E., Wu H. (1983). Extended Boolean Information Retrieval. *Communications of the ACM*, v. 26, n. 11, p. 1022-1036.

Salton G., Lesk M. E. (1968). Computer Evaluation of Indexing and Text Processing. *Journal of the ACM*, v. 15, n. 1, p. 8-36.

Segev I. (1998). Dendritic Processing. *The Handbook of Brain Theory and Neural Networks*. MIT Press, Cambridge, MA, p. 282-289.

Serra J. (1982). *Image Analysis and Mathematical Morphology*. Academic Press, London, England.

Shaw Jr. W. M., Burgin R., Howell P. (1997). Performance Standards and Evaluations in IR Test Collections: Cluster-based Retrieval Models. *Information Processing & Management*, v. 33, n. 1, p. 15-36.

Strohman, T., Metzler, D., Turtle, H., and Croft, W.B. (2005). Indri: A language-model based search engine for complex queries. *CIIR Technical Report*.

Syu I., Lang S. D. (1994). Heuristic Information Retrieval: A Competition-Based Connectionist Model for Information Retrieval Using a Merged Thesaurus. *Proceedings of the Third International Conference on Information and Knowledge Management*, p. 248-265.

Tague-Sutcliffe J. (1992). Measuring the Informativeness of a Retrieval Process. *Proceedings of the 15th Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, p. 23-36.

Turtle H., Croft W. B. (1991). Evaluation of an Inference Network-Based Retrieval Model. *ACM Transactions on Information Systems*, v. 9, n. 3, p. 187-222.

Turtle H., Croft W. B. (1990). Inference Networks for Document Retrieval. *Proceedings Of the 13th Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, p. 1-20.

van Rijsbergen C. J. (1979). *Information Retrieval*. Butterworths, London, England.

Wilkinson R., Hingston P. (1991). Using the Cosine Measure in a Neural Network for Document Retrieval. *In Proceedings of the 14th Annual International SIGIR Conference on Research and Development in Information Retrieval*, p. 202-210.

Ye J., Li Q., Xiong H., Park H., Janardan R., Kumar V. (2004). IDR/QR: an incremental dimension reduction algorithm via QR decomposition. *In Proceedings of the 10th Annual International ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, p. 364-373.

BIOGRAPHICAL SKETCH

Christian Roberson received his Bachelor of Science in computer engineering from the University of Florida in the spring 2001. From fall 2001 through summer 2007, he was a graduate assistant in the department of Computer and Information Science and Engineering at the University of Florida. He completed his Master of Engineering in computer engineering in spring 2003. In fall 2007 he joined the faculty of the Computer Science and Technology department at Plymouth State University in New Hampshire. His research interests include information retrieval, artificial intelligence, and games. He also enjoys movies, reading, basketball, and digital photography.