Department of Electronic Engineering

FINAL YEAR PROJECT REPORT

BEngIE-2006/07-WMT-01

NLP-based Artificially Intelligent Chat-bot

Student Name: Peswani, Sundeep Baldev
Student ID:
Supervisor: Dr. Tsang, Peter W. M.
Assessor: Dr. Wong, K. W.

Bachelor of Engineering (Honours) in Information Engineering
Student Final Year Project Declaration

I have read the student handbook and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I declare that the work submitted for the final year project does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Student Handbook.

Project Title: NLP-based Artificially Intelligent Chat-bot

Student Name: Peswani Sundeep

Student ID:

Signature
Date: November 30, 2006
No part of this report may be reproduced, stored in a retrieval system, or transcribed in any form or by any means – electronic, mechanical, photocopying, recording or otherwise – without the prior written permission of City University of Hong Kong.
Table of Contents

1 Introduction...........................................................................................................................................1
2 Objectives............................................................................................................................................4
3 Theoretical Background.......................................................................................................................5
  3.1 Natural Language Processing........................................................................................................5
    3.1.1 Word sense disambiguation.....................................................................................................6
    3.1.2 Syntactic disambiguation.......................................................................................................6
    3.1.3 Irregular input detection........................................................................................................7
  3.2 ELIZA..............................................................................................................................................7
    3.2.1 Architecture.............................................................................................................................7
      3.2.1.1 The script.........................................................................................................................8
      3.2.1.2 Key words........................................................................................................................9
      3.2.1.3 Decomposition Rules.....................................................................................................10
      3.2.1.4 Reassembly Rules..........................................................................................................10
      3.2.1.5 Synonyms.......................................................................................................................11
    3.2.2 Problems................................................................................................................................11
      3.2.2.1 Key word identification.................................................................................................12
      3.2.2.2 Choosing the appropriate transformation......................................................................12
      3.2.2.3 Generation of undirected responses...............................................................................13
  3.3 Recursive Distributed Representation............................................................................................14
    3.3.1 Background.............................................................................................................................14
      3.3.1.1 Connectionism...............................................................................................................14
      3.3.1.2 Neural Networks............................................................................................................15
        3.3.1.2.1 Architecture............................................................................................................15
        3.3.1.2.2 Back-propagation Neural Networks.......................................................................18
    3.3.2 RAAM....................................................................................................................................19
      3.3.2.1 Compression....................................................................................................................20
      3.3.2.2 Reconstruction.................................................................................................................20
      3.3.2.3 Architecture....................................................................................................................21
    3.3.3 Parse Trees................................................................................................................................23
    3.3.4 Dictionary................................................................................................................................24
      3.3.4.1 Syntactical Category Identifier......................................................................................25
      3.3.4.2 Semantic Identifier.........................................................................................................26
    3.3.5 Architecture................................................................................................................................27
  3.4 Topic Modeling.................................................................................................................................28
    3.4.1 Mathematics............................................................................................................................29
      3.4.1.1 Statistical Relationship..................................................................................................30
        3.4.1.1.1 Documents..............................................................................................................30
        3.4.1.1.2 Topics.....................................................................................................................30
        3.4.1.1.3 Words.....................................................................................................................31
        3.4.1.1.4 Topic model distribution.........................................................................................31
      3.4.1.2 Gibbs' Sampling..............................................................................................................33
    3.4.2 Architecture...............................................................................................................................34
      3.4.2.1 Sparse Matrices..............................................................................................................34
Index of Figures

Figure 1: ELIZA Script Architecture................................................................. 9
Figure 2: Neural Network Architecture.......................................................... 16
Figure 3: Neuronal Calculations................................................................. 17
Figure 4: Back-propagation Learning............................................................ 19
Figure 5: Compressor and Reconstructor.................................................. 20
Figure 6: RAAM Architecture........................................................................ 22
Figure 7: Two-layer RAAM Architecture.................................................. 23
Figure 8: Parse Tree Types............................................................................. 23
Figure 9: Sample Parse Tree.......................................................................... 24
Figure 10: Codeword...................................................................................... 25
Figure 11: Recursive distributed representation initialization.................. 27
Figure 12: First representations....................................................................... 27
Figure 13: Pictorial depiction of Topic Modeling......................................... 32
Figure 14: Functional Flowchart of the System.......................................... 40
Figure 15: Functional Design of ELIZA...................................................... 41
Figure 16: Functional Design of the RAAM............................................... 42
Figure 17: Data-Mining Entity-Relationship Diagram.............................. 44
Figure 18: Topic-Modeling Entity-Relationship Diagram ......................... 45
Figure 19: Structural Design of ELIZA......................................................... 46
Figure 20: Structure of the Recursive Distributed Representation............. 47
Figure 21: Data Mining Structure............................................................... 48

Index of Tables

Table 1: Keyword-Rank Relationship.......................................................... 12
Table 2: Sample syntactical identifiers....................................................... 25
Abstract

Tools such as search engines and personal agents lack personality, evidenced by their mechanical acceptance of input and production of output, and practicality, demonstrated by their in-take of a single type of input, like keywords. To assuage these issues, techniques developed in Natural Language Processing (NLP) are used to produce a model of a system which combines a normal chatter-bot with a more intelligent document categorization and retrieval system, thereby creating a new digital assistant system.

The NLP techniques used in this project include topic modeling, recursive distributed representation, which is a form of connectionist modeling, and Weizenbaum's ELIZA. The first of the three is used to categorize a corpus of documents, while the other techniques are used to interact with the user.

Reliance on keywords was found to be unavoidable, but modeling the corpus to topics rather than a spatial distribution, such as that of frequency, proved to be successful at retrieving the relevant documents, to a certain extent. The topic modeling mechanism appeared to be highly influenced by the number of words in each document. The connectionist modeling was worse than ELIZA at recognizing trained phrases, but was better at dealing with unknown words.


1 Introduction

Natural Language Processing (NLP), a field of cognitive science derived from artificial intelligence and linguistics, deals with the generation and comprehension of natural human languages by computer, or other artificial, agents. Techniques and technologies developed in this discipline are used to solve some problems found in modern personal desktop assistants.

Personal digital assistants perform a myriad of services, including but not limited to, the organization and storage of data as well as the intelligent retrieval of subsets of information. These assistants range from the more personable Bonzi Buddy to the more useful Google Desktop Search. However, most of these assistants have two main flaws: syntactic indifference and word sense confusion. The goal of this project is, therefore, to assuage these problems, by building a proof-of-concept system which can deliver better results. By using NLP, as well as data-mining, techniques, a chatter bot can be built which imitates the power of traditional search engines, especially desktop search engines, and the user-friendliness and responsiveness of chatter bots like ELIZA or Artificial Linguistic Internet Computer Entity (or A.L.I.C.E), creating a truer digital personal assistant.

A proof-of-concept system was built according to the aforementioned criteria. It succeeded in some aspects, but failed in others. Nevertheless, it is believed that this represents an important step in improving current information retrieval and personal assistant systems.

The first of the two flaws, syntactic indifference, means that these software agents usually interpret input from the user by tokenizing, or segmenting, it into a series, or array, of words. They break down
the syntactical structure to such an extent that it is marginalized, if not lost completely. In some situations, this is preferable, but syntax is usually conducive to the better comprehension of the input. To solve this problem, two NLP techniques are employed: ELIZA and Recursive Distributed Representation.

Extensive testing showed that ELIZA was better at comprehending user input, when the input adhered to the trained rule set. It, therefore, did not demonstrate much flexibility in dealing with more generalized forms of human input. Recursive Distributed Representation, on the other hand, performed slightly worse than ELIZA when it encountered input similar to the training rules, but was better at coping with words or phrases it had not trained with. Therefore, Recursive Distributed Representation proved to be more flexible in handling human input, and it is believed that further development and training will provide even better results.

The second issue with modern personal desktop assistants is that of word sense disambiguation. This means that these assistants fail to semantically differentiate words, if they are homographs (or words that are spelled the same.) To solve this problem, topic modeling is employed to try and place each word into a topic. This hopefully allows each word to gain some semantic value.

However, it was discovered that simple frequency distributions provided more relevant results than topic modeling. This was likely due to the influence of larger documents on smaller ones, skewing the model. Documents which had more words were more likely to be more relevant or topical for all words, because of the probabilistic nature of the distribution used by topic modeling. It is believed that normalization of word frequencies in documents, or the separate processing of these documents, should solve this problem.
Natural Language Processing has grown tremendously since its early days, and while some of its developments perform better than others, many of its developments have not been used by the general public. This has depressed any improvements in the capabilities of supposedly-intelligent artificial agents and information retrieval systems, such as search engines. This project has demonstrated that these developments have legitimate practical uses and can be used beneficially in current systems.
2 Objectives

This project intends to solve some of the problems found in current information retrieval and personal assistant systems. Two of the most prevalent issues are:

1. Syntactic Indifference

   Syntactic indifference is the failure of these systems to recognize and involve syntactic structure in the comprehension and processing of user input. It is believed that by minimizing the impact of grammatical structure, the intended meaning of the input is skewed, if not lost.

2. Word Sense Confusion

   Word sense confusion involves the inability of these systems to distinguish between words if they are spelled the same. This is because these systems disregard context as a valuable part of processing words, unlike humans. Thus, words are thought to be the same if they are spelled the same.

The main goal, then, is to solve these two problems. This is done with the use of NLP techniques. The solutions used in this project are ELIZA, Recursive Distributed Representation and, Topic Modeling.

By using these techniques, it is possible to build a system which is able to parse, comprehend and answer a variety of user queries intelligently and which is able to analyze a corpus of documents and model them in such a way as to provide the most relevant results as required.
3 Theoretical Background

Before proceeding to the solution, the theoretical concepts behind the solutions are first examined. We will start by studying the field of Natural Language Processing (NLP), and proceed by examining the techniques used in this project, namely, ELIZA, Recursive Distributed Representation and Topic Modeling.

3.1 Natural Language Processing

Natural Language Processing (NLP) is a field derived from artificial intelligence and linguistics and deals with the generation and comprehension of natural human languages by computer, or other artificial, agents.

Given this definition of NLP, it is easy to see potential uses for its technologies. Its usefulness in processing natural language constructs means that it can form the basis of flexible, ambiguous or complex human-computer interaction, such as machine translation or information extraction, rather than more restricted or simpler forms of interaction, such as clicking or entering values into fields.

There are numerous problems or issues in dealing with general language. However, we will discuss the specific issues encountered in this project. These are:

- Word sense disambiguation
- Syntactic disambiguation
- Irregular input detection
We will analyze each of these in further detail.

### 3.1.1 Word sense disambiguation

Word sense disambiguation is the problem caused by words having more than one meaning. Because of this, each instance, or token, of any word may have a different meaning from other instances. In effect, this one-to-many relationship between words and meanings becomes many one-to-one relationships between word instances and meanings. It is therefore difficult to use a deterministic solution to this problem. In order to select the appropriate meaning for each word instance, this project employs a statistical method.

### 3.1.2 Syntactic disambiguation

Natural languages have ambiguous grammar structures. This is more notable in English than in more rigid languages like Latin. For this reason, when processing natural languages, one must be wary not only of different grammar structures, but also of different uses for each grammar structure. This leads us to a similar problem to that identified for words in Section 3.1.1. As with word sense disambiguation, context can be used to solve this problem. Additionally, semantics may be useful in solving this problem.

As this project is focused mainly on providing a more robust input interface for users, syntactic disambiguation was not a priority. Syntactic processes used, therefore, focused on the identification and comprehension of semantics rather than syntax.
3.1.3 Irregular input detection

Natural language is notorious for its lack of structure, unlike machine or programming languages. This can be compounded by errors in input, be it typographical or grammatical. These errors need to be detected, and if possible, fixed before other forms of natural language processing are applied.

As is common with many popular chatter bots, this project dealt with this issue by asking for further clarification rather than attempting to use time-consuming and imperfect algorithms.

3.2 ELIZA

ELIZA is a very famous piece of software invented in 1966 by Joseph Weizenbaum to imitate a psychotherapist. It poses questions to human subjects based on their input and it can do this without sophisticated natural language processing techniques. Instead, it scans input for general keywords and transforms it into new questions, based on a set of rules as defined in its “script”. Nevertheless, it is possible for a sufficiently advanced form of ELIZA to give the impression that sophisticated forms of natural language processing are being used and that the computer can actually understand the input it receives.

In order to better understand ELIZA, we will first analyze the system ELIZA uses. We will then go over some of the problems ELIZA faces, and how it deals with them.

3.2.1 Architecture

At the heart of any implementation of ELIZA lies the script. This script defines key words, decomposition and reassembly rules for each of these key words as well as pre- and post-processing synonyms for further extensibility.
We will examine each of these components in further detail.

3.2.1.1 The script

Conceptually, ELIZA stores its script as a tree, as depicted in Fig. 1. Each keyword begets a number of decomposition rules. These rules define how input which matches that keyword should be decomposed for further processing. In turn, each of these decomposition rules progenerates a number of reassembly rules, which define how a decomposed input should be reconstructed into a response. We shall examine each of these elements below.
3.2.1.2 Key words

Key words are simply words used to identify the intention of a sentence. The basic idea behind this is that any sentence has a specific intention. That intention necessitates the use of certain words. For example, the expression of desire necessitates the use of words like “want” or “need.” This relationship goes both ways. Therefore, the use of certain words, like “want” or “need,” indicate that the user desires something. Of course, this is not foolproof, so it is not effective in all cases. Nevertheless, its
simplicity and use of minimal resources makes this idea very attractive.

The word-intention relationship allows the author or owner of ELIZA to devise a set of suitable responses for the appropriate intention, if a keyword is found. Again, falling back to our previous example, the author or owner of ELIZA may choose to deal with all expressions of desire in a certain way. We shall examine such procedures in the next few sections. It is important to note however that this word-intention relationship forms the crux of all ELIZA procedures.

3.2.1.3 Decomposition Rules

Each key word in an ELIZA script is accompanied by a set of decomposition rules. These rules are used to break an inputted string into several components. These components can be further analyzed, or as is the norm, used in the reassembly process for further questioning.

If a user indicates an expression of desire, with the use of the word “want,” for example, a possible decomposition rule would be “I want (1).” (1) is the indicator which denotes a string of text. This text, in this example, details what the user wants. ELIZA can then proceed to ask the user why he wants (1). This is done in the reassembly process.

3.2.1.4 Reassembly Rules

For each decomposition rule, there exist a number of rules which define how the various components from a user’s input should be re-used. These rules are called reassembly rules.

Using our example from Section 3.2.1.3, we know that the user wants (1). We may then create a reassembly rule which asks the user for further details. For example, a reassembly rule may be: “Why
do you want (1)?” Thus, the decomposition rule works in conjunction with the reassembly rule to break
down and identify interesting or useful parts of a user’s input and to use this information in formulating
a response.

3.2.1.5 Synonyms

Another important aspect of modern ELIZA systems is the use of synonyms. The use of synonyms
extends ELIZA’s capabilities and allows it to deal with a variety of input with a simplified keyword
and rule set. This is because, each time a synonym is encountered, it is replaced with a common word.
Thus, all verbs denoting desire (“need”, “desire”, “request”) are replaced with a single verb, such as
“want.”

There are two forms of synonyms: pre-process synonyms and post-process synonyms. Pre-process
synonyms are replaced before any additional processing (such as key word identification) is done. Post-
process synonyms are used after the reassembly rule has been applied.

3.2.2 Problems

There are five main problems for ELIZA as identified by Weizenbaum¹, and they are:

1. the identification of key words;

2. the identification of context;

3. the choice of an appropriate transformation;

¹ Weizenbaum J. (1966). *ELIZA – A Computer Program for the Study of Natural Language Communication Between Man
and Machine*, Communications of the ACM Vol 9, Number 1
4. the generation of a response in the absence of any known key words; and,

5. the provision of an editing capacity.

I shall examine some of these issues and how they can be handled.

3.2.2.1 Key word identification

As explored in Section 3.2.1.2, ELIZA uses a simple scanning mechanism to select a key word from user input. Naturally, because of the simplified selection scheme, there may be more than one key word in each sentence. In order to deal with this situation, each key word in the script is given a rank or preference. Thus, if more than one key word is found in each sentence, the key word with the highest rank is selected for further processing.

Table 1 shows an extract of an ELIZA script. If the user inputs the following sentence: “Please tell me your name”, ELIZA will select “name” as the key word to use in further processing, as it has the highest rank.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>0</td>
</tr>
<tr>
<td>please</td>
<td>1</td>
</tr>
<tr>
<td>help</td>
<td>2</td>
</tr>
<tr>
<td>tell</td>
<td>3</td>
</tr>
<tr>
<td>name</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 1: Keyword-Rank Relationship

3.2.2.2 Choosing the appropriate transformation

Upon selection of an appropriate key word, a set of rules, which decompose and reassemble the user's input, are applied. However, given that each key word may have multiple rules, how is a rule selected?
ELIZA simply selects the first decomposition rule that matches the input. Given the input, “Please tell me your name,” as in Section 3.2.2.1, and given the following decomposition rules:

(i) (1) my name;

(ii) (1) tell (2) my name; and,

(iii)(1) tell (2) your name,

where (1) and (2) refer to any string of text, ELIZA will fail to match rules (i) and (ii) with the input. This leaves rule (iii). Thus, this is the rule used. Naturally, this simple matching mechanism will fail to work in a variety of instances.

### 3.2.2.3 Generation of undirected responses

As examined earlier, key words are used to determine the response to any input. We've examined how ELIZA behaves when it comes across multiple key words. But what if no key word is found?

ELIZA simply falls back on a default response. This usually requests the user for more information. This can be done with something as simple as “Please go on.”
3.3 Recursive Distributed Representation

Recursive distributed representation is a form of connectionist modeling. It decomposes input into a tree structure, and then uses a modified form of a back-propagation neural network to process this tree. The resultant representation is a unique compression of the input. Interestingly, the one-to-one nature of this relationship gives it the ability to reconstruct the original input, i.e., the tree can be traversed bidirectionally.

We will examine the roots of recursive distributed representation, which lie in Connectionism, and then proceed to analyze how recursive distributed representation works and its various components, including the Recursive Auto-Associative Memory (RAAM), which is the modified form of a back-propagation neural network used by this system, the parse tree and the dictionary.

3.3.1 Background

3.3.1.1 Connectionism

Connectionism models mental processes as the result of processing by networks of simpler functional units. With connectionism, any intelligence is the result of connections between these various units. The better, or more appropriate to the goal, these connections are, the more likely the entire network will return the expected output. Increasing intelligence or learning, therefore, is simply a matter of adjusting these connections formulaically towards the intended goal.

The most common connectionist model is the neural network. We will examine the basic neural network, and its most common variation, the back-propagation (or back-prop) neural network, as a theoretical preparation for the RAAM.
### 3.3.1.2 Neural Networks

An artificial neural network attempts to produce intelligence in much the same way a brain does. As such, it is modeled on the brain itself.

#### 3.3.1.2.1 Architecture

There are two main elements of any neural network. Singular units, or “neurons,” are rudimentary black boxes and process input. Weighted links connect these units and form the network.

There are three different layers to an artificial neural network: the input layer; the hidden layer; and, the output layer. These layers are depicted in Fig. 2.

The first, or the input, layer accepts input from outside the network, and delivers its output upwards through the network. The hidden layer forms the body of the network. There can be, and usually are, multiple hidden layers. These layers are used to increase the processing capability of the network. The final layer, the output layer, simply delivers the output, after processing.
Neurons accept input from either other neurons or from outside the network. In the first case, this input is multiplied by the weight of its link to other neurons. The summation of this input is then inputted into an activation function, usually the sigmoid function, to get the activation level of that neuron. The activation level can be thought of as the state of the neuron (on or off). Of course, there may be many degrees of activation (dim, bright, etc.) The activation level, $y$, becomes the output of the neuron, which is then passed on to other neurons via weighted links. This is demonstrated in Fig. 3.
Conceptually, links deliver output from one neuron to another neuron as input. Each link has a weight which marks its influence; a highly-weighted link has more influence than a lowly-weighted link. As inputs to the network cannot be adjusted to deliver the right output, weights must be adjusted in order to do so. Weights are adjusted algorithmically, and each type of neural network has its own basic formula for adjusting weights, i.e., calculating the difference in weights between each epoch.
3.3.1.2.2 Back-propagation Neural Networks

A back-propagation neural network is a form of artificial neural network, first defined by Werbos in 1974. It uses a form of supervised learning, and usually is entirely feed-forward, which means it’s free of any connection loops. It has the same elements as that of any artificial neural network. The only difference lies in the method used to adjust the weights, i.e., the learning or training algorithm. Therefore, in this section, we will only examine the learning process.

As a back-prop neural network uses supervised learning, it has a target, or expected, output. After any input is processed by the neural network, the output from this network is compared to the expected output. Any difference is called an error, or more technically, the error signal of the output layer neuron. This error is used to train the network.

Since this calculated error only pertains to the output level neurons, what of those in the hidden layer and the input layer? The idea of a back-propagation neural network is to propagate the error signal backwards through the network. Conceptually, the error is fed downwards. Each neuron then calculates its own error using the weighted links and the error propagated from upper levels. Summation of all error signals is required if more than one error signal is received.

We won’t examine the precise algorithms used, because they are out of the scope of the project itself. Suffice it to say that comprehension of the procedure is more important. To this end, Fig. 4 demonstrates this weight training process, for a few neurons. $\delta_c$ is equivalent to $\delta$. $\delta_c$ begets $\delta_A$, which, in turn, begets $\delta_1$. 
3.3.2 RAAM

Recursive auto-associative memory (RAAM) is an alternative form of the back-propagation neural network. There are two main differences: it processes inputs recursively; and, the input is the same as expected output. These two differences are required in order to successfully handle the parse tree as examined in Section 3.3.3.

There are also two distinct needs of any RAAM: compression and reconstruction. The RAAM must have the ability to compress a set of elements into an internalized representation, as shown in Fig. 5, and it must also have the ability to reconstruct those two elements from any internalized representation. These two needs result in the two aforementioned differences respectively.
3.3.2.1 Compression

Compression involves the representation of multiple input elements with one element. This element is, in RAAM, decided by the network itself. That is to say, this part of the learning process is unsupervised.

Compression allows larger sentences to be split into a smaller series of representations, *ad infinitum*. This is crucial if a parse tree, which recursive distributed representation needs, is to be used. Each of these subsequent representations is of the same size and same type as that of the input. This allows these new representations to be used together with the old, as required by the tree-parsing method. This, also, causes the first difference between back-prop neural nets and the RAAM: that of the re-use of output as input.

3.3.2.2 Reconstruction

Reconstruction is the process by which internalized representations can be reformed, or broken down, into their original constituents. This is the anti-thesis to the compression procedure. This process is that which permits the bi-directionality of the recursive distributed representation system. Without it, all
internal representations, calculated by the compression process, would be useless, if not actually meaningless.

This does present a problem however. If any internal representation can be broken down into its constituents, then wouldn’t the output of this process be the same as the input of the compression process? Actually, yes, this is in fact the theory which drives the RAAM. If input is meant to be the same as the output, we can use the input to the compressor as the expected output of the reconstructor, and use it to train this network.

### 3.3.2.3 Architecture

Now that we have examined the two main components of the recursive auto-associative memory, let us examine how these two are combined into something resembling a neural network.

We know that the output from the compression module is an internalized representation of its inputs. This output is used by the reconstruction module to re-generate the input. Then, the solution is obvious: we must join these two modules such that the output from the compressor is fed into the reconstructor. This gives us a network similar to that of Fig. 6.
Using this model, the internalized representation is stored at the hidden level. This must be extracted for use in further processing of the parse tree. This model is also symmetric about the hidden layer, i.e., the output is the same as the input. Essentially, the neural network must be able to output whatever its input is. This three-layer model may, then, seem redundant, as a simple two-layer model which has a one-to-one relationship between input and output, as shown in Fig. 7, would suffice. However, this two-layer model would not give us any internal representation for further processing, although it would be more accurate than this three-layer model.
3.3.3 Parse Trees

The second main component of the Recursive Distributed Representation is that of the parse tree. This is used to break down any sentence or phrase into a tree form. The sentence can be broken down syntactically or sequentially, as shown in Fig. 8.

![Parse Tree Types](image)

This deconstruction of the sentence into a tree allows us to adopt the RAAM in small steps. Each input set (a collection of leaf nodes) is processed in order to get an internalized representation, which is
placed into the tree, as a parent node for that set. This process continues upwards, until the entire sentence is compressed into a single representation.

Using Fig. 9 as an example, we can see that first A and B will be processed by the RAAM, resulting in an internal representation R1. C and D can be processed to get R2, and finally R1 and R2 are processed in order to get R3, which represents the tree as a whole.

Since one of the features of the RAAM is that any internal representation can theoretically be reconstructed, this tree, upon completion of the training process, is bi-directional, i.e., given R3, we should be able to recover the original sentence in its entirety.

### 3.3.4 Dictionary

The final piece of the recursive distributed representation is the dictionary. This simply maintains a one-to-one relationship between words and codewords. These codewords, usually in bits, are used in the RAAM itself, as binary math would be easier to handle.

However, the construction of the codewords is not random, or simply sequentialized. Each codeword
has two parts: a syntactical category identifier and a semantic identifier, as depicted in Fig. 10. Each \( b \) represents a single binary digit (bit.) We shall examine these two identifiers.

A special codeword, however, is included with each dictionary, and that is the empty codeword. When a word is encountered which does not have a matching entry in the dictionary, the empty codeword is used instead. This codeword is all 0s, in stark contrast to other codewords.

### 3.3.4.1 Syntactical Category Identifier

This identifier forms the first part of the codeword. This identifier simply identifies the grammatical type of the word, such as nouns, verbs, adjectives, etc.

This identifier is encoded in the 1-in-N format, which means that only 1 bit out of N is enabled (set to 1); the rest are 0. N, intuitively, is the number of syntactical categories which exist. Table 2 is a potential sample set of syntactical category identifiers.
1-in-N bit encoding is used so that there is no clashes. If a sequential format was used, such that the first category was encoded as 001, the second as 010, the third as 011, etc., a summation of a word from the first category and a word from the second category would result in a representation of a word from the third category. This should not happen, as grammar or syntax has no combinatorial properties, unlike mathematics.

### 3.3.4.2 Semantic Identifier

This identifier forms the second part of the codeword. It identifies the meaning, the stem, or the root of each word. As many words are derived from root word, this identifier is used to identify them as the same. Thus, in combination with the syntactical category identifier, each root may have different forms, or different conjugations.

Again, 1-in-N bit encoding is applied to reduce clashes or misappropriations, as semantics have no combinatorial properties either. However, in this case, N is the maximum number of words of each syntactical category. If, for example, the first category had 5 words, the second category had 8 and the third category had 2, N would be 8. This is done so that all codewords can be accommodated, but they all have the same length.

<table>
<thead>
<tr>
<th>Category</th>
<th>Syntactical Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noun</td>
<td>0001</td>
</tr>
<tr>
<td>Verb</td>
<td>0010</td>
</tr>
<tr>
<td>Adjective</td>
<td>0100</td>
</tr>
<tr>
<td>Adverb</td>
<td>1000</td>
</tr>
</tbody>
</table>

*Table 2: Sample syntactical identifiers*
3.3.5 Architecture

Now that we understand RAAM, the dictionary and the parse tree, we must analyze how recursive distributed representation works.

Given a sentence, each word is first replaced using the dictionary. It is then deconstructed into a parse tree, such that each codeword is a leaf node. This is shown in Fig. 11.

![Figure 11: Recursive distributed representation initialization](image)

Sentence: This is a sentence

<table>
<thead>
<tr>
<th>Word</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>this</td>
<td>A</td>
</tr>
<tr>
<td>is</td>
<td>B</td>
</tr>
<tr>
<td>a</td>
<td>C</td>
</tr>
<tr>
<td>sentence</td>
<td>D</td>
</tr>
</tbody>
</table>

Then, a set of these nodes, generally n in number for n-ary trees (e.g., two for binary trees), are passed to the RAAM. This returns an internalized representation from its hidden layer, and performs weight corrections. The rest of the sentence is handled in this way, until we have a new layer of internalized representations, as shown in Fig. 12.

![Figure 12: First representations](image)
These internalized representations are then used as input, much as the leaf nodes were. Eventually, we will have a single representation. This represents the entire sentence structure, and we can move on to the next sentence.

If we are using an n-ary tree, and the number of nodes is not a multiple of n, the special empty codeword, which is all 0s, can be used. As this codeword is all 0s which means it has no enabled bits, it, theoretically, has no substantial effect on the calculation.

### 3.4 Topic Modeling

Topic modeling, unlike ELIZA and RAAM, is a generative statistical language model. What this means is that it uses a probabilistic procedure by which documents can be generated, given topics and words. Topic modeling seeks to create a relationship between words and documents beyond that of simple frequency or other spatial distributions. To do this, it uses a third element which is called a topic. Each topic is a probability distribution over the unique words in the corpus, from which we can generate word counts for documents. Each topic, therefore, is defined by its relationships with these elements.

We must understand that topic modeling is a generative model, i.e., it was designed to generate things rather than break them down. A document, then, can be generated from words by first choosing, for the document, a distribution over all topics, and then selecting words from each topic according to the distribution. If, for document \( d \), we choose a distribution such that topic A has a weight of 0.4, topic B a weight of 0.5 and topic C a weight of 0.1, we will then select words from these topics according to this distribution. Thus, 40% of words in that document would belong to topic A, 50% to topic B and 10% to topic C. The selection of words from these topics adheres to each topic's own distribution over
words. If word $w$ has a 10% chance of belonging to topic A, 4% (10% of 40%) of document $d$ will be $w$.

However, since we already know the relationships between words and documents, we need to infer other relationships from these. Thus, standard statistical inference rules can be applied.

The benefit of this probabilistic representation is that each topic, after the topic modeling is processed, picks out a cluster of (probabilistically-)related terms or words, which have the highest probability of belonging to that topic. This gives a topic a certain meaning, derived from this cluster of words. One can think of it as akin to elements in a photograph: the photo itself has no intrinsic meaning; it's meaning is derived from the most powerful elements of the photograph.

We can see that the topic element simply splits the word-document relationship into two, similar in concept to the chain-rule for differentiation. This allows for easier processing and also gives us two unique, useful relationships, that of word-topic and that of topic-document. Words can describe topics, and topics can be used to describe documents.

We will fix examine the relationship between words, topics and documents in topic modeling, followed by the examination of how this is achieved. We will finally examine how topic modeling can be used as a search facility.

### 3.4.1 Mathematics

As topic modeling is a statistical model, mathematics plays an important role. Thus, before we understand the architecture of this model, we must have some background understanding of the
3.4.1.1 Statistical Relationship

In order to examine the statistical relationship between the different elements in topic modeling, namely topics, words and documents, we must first examine what these elements mean in this paradigm. We can then examine the topic model distribution itself.

3.4.1.1.1 Documents

A document, in the topic modeling paradigm, can be represented as a mixture of $T$ topics, where $T$ is the number of topics allocated. Each document has a different relationship, or probability coefficient, for these topics. However, the summation of all probability coefficients for a document must equal to 1, as is common with other statistical models. Mathematically speaking,

$$\sum_{t=1}^{T} p(t|d) = 1.$$ 

3.4.1.1.2 Topics

Topics are simply collections of probabilistic relationships between itself and the other two elements. It's an external addition to the common word-document relationship in order to make this relationship more malleable.

These topics are not pre-determined, which is to say, that words or documents are not pre-assigned to certain topics. They are merely statistical categories formed with the use of statistical procedures. They do however, in forming relationships with the other two elements, retain some meaning. This meaning is defined by its relationships with the other two elements, i.e., a topic is given a definition based on the
highest probabilistic relationships with words or documents.

If, for example, a certain topic had a probabilistically high relationship with words like “engineer”, “artificial”, “intelligence” and “project”, and documents such as this report and its reference materials, it may be defined as describing this project! However, the topic model itself assigns no intrinsic meaning to these topics.

### 3.4.1.1.3 Words

A word is a basic building block of the topic model. It is used to define a topic and help generate a document.

### 3.4.1.1.4 Topic model distribution

The topic model distribution links topics, words and documents together with a simple equation:

\[
p(w|d) = \sum_{t=1}^{T} p(w|t)p(t|d)
\]

For those versed in statistics, the relationship is clear. However, let us examine each element closely. The first element:

\[
p(w|d)
\]

defines the probability relationship between words and documents. This can be read as “the probability that word \( w \) belongs to document \( d \).” The next element is:
\[ p(w|t) , \]

which can be read as “the probability that word \( w \) belongs to topic \( t \).” The final element is:

\[ p(t|d) . \]

This can be read as “the probability that topic \( t \) belongs to document \( d \).” Thus, the probability that word \( w \) exists or belongs to document \( d \) (or the probability of word \( w \) given document \( d \)) is equivalent to the multiplication of the probability of that word belong to a certain topic multiplied by the probability that that topic belongs to that document, summed over all topics. This relationship is clearer if matrices are used instead of probability distributions, and this is depicted pictorially in Fig. 13.

\[
\begin{array}{c|c}
\text{documents} & \p(\text{w|d}) \\
\hline
\text{words} & \p(\text{w|t}) \\
\hline
\text{topics} & \p(\text{t|d}) \\
\hline
\text{documents} & \\
\end{array}
\]

\text{Figure 13: Pictorial depiction of Topic Modeling}

However, with further inspection, we are able to tell the probability that word \( w \) belongs to a document \( d \). This is given to us when we data-mine the documents. But what of the other two probability distributions? How are they generated?

They are generated using Gibbs' sampling, which we shall discuss further in the next section.
3.4.1.2 Gibbs' Sampling

Gibbs' sampling is a general method for statistical inference. It is an algorithm which is able to generate a sequence of samples from a joint probability distribution of multiple random variables. It is an example of a Markov Chain Monte Carlo (MCMC) algorithm. It is applicable when the joint distribution is unknown, but the conditions behind that distribution are known.

In order to explain the Gibbs' Sampler process, I will quote B. Walsh's excellent explanation:

To introduce the Gibbs sampler, consider a bivariate random variable \((x,y)\), and suppose we wish to compute one or both marginals, \(p(x)\) and \(p(y)\). The idea behind the sampler is that it is far easier to consider a sequence of conditional distributions, \(p(x|y)\) and \(p(y|x)\), than it is to obtain the marginal by integration of the joint density \(p(x,y)\). e.g., \(p(x) = \int p(x,y)dy\). The sampler starts with some initial value \(y_0\) for \(y\) and obtains \(x_0\) by generating a random variable from the conditional distribution \(p(x|y = y_0)\). The sampler then uses \(x_0\) to generate a new value of \(y_1\), drawing from the conditional distribution based on the value \(x_0\), \(p(y|x = x_0)\). The sampler proceeds as follows

\[
\begin{align*}
x_i &= p(x|y = y_{i-1}) \\
y_i &= p(y|x = x_i)
\end{align*}
\]

Suffice it to say that I do not wish to enter into a lengthy explanation of the mathematics behind this sampling method. I would refer the reader, however, to Walsh's paper on the subject.

---


3.4.2 Architecture

Now that we understand the mathematical principles behind topic modeling, let us analyze how it can be used as a searching mechanism.

We must remind ourselves that topic model is a generative model, and that we seek to reverse its process, if we are to use it as a searching mechanism. Rather than finding the probability that word $w$ belongs to document $d$, or $p(w|d)$, we seek instead to find the probability that document $d$ contains word $w$, or $p(d|w)$. Then, we select the document which has the highest probability of containing word $w$.

We can do this by slightly modifying the process. First we must see how probability distributions can be stored as sparse matrices, and then we will see how a search can be made.

3.4.2.1 Sparse Matrices

After we produce a word-topic probability distribution, $p(w|t)$, and a topic-probability distribution, $p(t|d)$, we can store these distributions as a sparse matrix. This is important, as it eliminates all redundant relationships, i.e., relationships which are non-existent or have a value of 0.

With computer programming languages, sparse matrices are best stored as hashes, rather than arrays. A hash stores data in a key-value relationship. With the word-topic probability distribution, for example, both $w$ and $t$ serve as keys, and $p(w|t)$ serves as the value. Thus if word $w_0$ had a probabilistic relationship with topic $t_0$ of 0.01, we, in Perl, would store this as:

$$word_topic_distribution[w_0][t_0] = 0.01.$$
In a database, we can simply use a tri-columnar table to store this relationship. Again, using the word-topic probability distribution as an example, the above relationship between $w_0$ and $t_0$ would be stored as:

<table>
<thead>
<tr>
<th>word_id</th>
<th>topic_id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_0$</td>
<td>$t_0$</td>
<td>0.01</td>
</tr>
</tbody>
</table>

### 3.4.2.2 Search

Given the sparse matrix storage, we can search for the most relevant document for a word using simple links. Given a word $w_k$, we can search for the topic $t_k$ which has the highest probabilistic relationship. Then, we can select the most probabilistic document $d_k$ for $t_k$. This document, then, is the most topically-relevant document for $w_k$. 


4 Methodologies

Any research topic, software or otherwise, requires certain methodologies to be used, and adhered to, in order to succeed. As the project encompasses different technical fields and requires specific handling for each subsection, the project was divided into two different sections: (1) data-mining and data-retrieval; and, (2) user interaction.

Thus the project was divided into the following key stages:

(i) Background research and investigation;

(ii) Structural design;

(iii) Development of core features; and,

(iv) Usability and applicability testing.

Each of these stages is further elaborated on below.

4.1 Background research and investigation

This stage was expected to be the one of the most time-consuming phases, as it involved compiling, analyzing and understanding materials about three different subjects: neural networks, statistical topic modeling, tensor product representations of syntactic structures and data-mining.

Research will include all forms of media, tools, programming libraries, application programming interfaces, external samples and efforts, scientific papers as well as textual content.
4.2 **Structural design**

After research, a modular approach was taken to design the structure of the project, so that while different parts of the software will be able to communicate with each other, they will not affect any operational facility, but their own.

4.3 **Development of core features**

This stage occupied the majority of the allocated project time. Development was methodologically identical for both sub-sections of the project. It involved the use of modules or supplemental frameworks, small, specifically targeted classes and programming blocks as well as regular unit testing.

Rather than lengthy, unmaintainable segments of code, the idea was to simplify algorithms into sub-algorithms until they were sufficiently atomic. The final project would involve the cohesion of these into a larger whole.

We will examine why Perl is used as the programming language, and why MySQL is used as the database.

4.3.1 **Perl**

Perl is a programming language invented by Larry Wall in 1987. It has a long history and a tremendous support system. This support system includes the development of many useful software libraries and modules, some of which have been used in this project.

Perl can be used on multiple platforms and is free and open-source, but none of these issues played greatly in the decision to use Perl.
Perl, compared to Java, has a smaller footprint, but is slower. This did result in some delays in processing, but the memory benefits were the trade-off. Furthermore, Perl is not a strongly-typed language, nor is it a strictly OOP language. This is important as most of the elements of this project did not necessitate the use of object-oriented programming paradigms.

Unlike C++, Perl is great at handling text and storage systems such as files and databases. As these three elements were crucial to the development of the project, it was imperative that the programming language had good support for this. Perl has good support for sparse matrices, with the use of hashes. This was essential for the implementation of topic modeling. C++ requires external libraries for these to work, and in many cases, they pale in comparison to Perl's feature-set.

Finally, there is the TIMTOWTDI belief Perl imbues in its development. TIMTOWTDI stands for “There is more than one way to do it.” This translates into a flexible developmental methodology.

4.3.2 MySQL

MySQL is, like Perl, free and open-source. It is supported on multiple platforms. However, the decision to use MySQL over other competing products such as Oracle, PostgreSQL or MSSQL, lies in its efficiency, excellent support and seamless developmental experience with Perl.

4.4 Usability and applicability testing

As each sub-section was complete enough to require specific testing, it was tested with a scaled-down, yet representative, dataset in order to ensure that the development worked as designed.
5 System Specifications

Now that we have a theoretical and methodological background to this project, we need to examine how the system was developed. To this end, we need to examine the design of the system, and then its implementation.

5.1 Design

Before we analyze the structural design of the system, we need to examine its functions.

5.1.1 Functional Design

From the functional outline of the system in Fig 14, we can see that there are two major sections of the system: the interface and the search engine. We can analyze each of these functions separately.
5.1.1.1 Interface

Functionally speaking, the interface seeks to get user input, and select a keyword, which it then passes on to the search engine. It must respond in the negative if no document is found. It must also be able to deal with erroneous input. There are two sub-systems used in the Interface: ELIZA and RAAM.

5.1.1.1.1 ELIZA

Quite simply, the functional aspect for the model is to use the same process as ELIZA to identify a keyword and select a rule. However, rather than return a response constructed from the reassembly
process, the ELIZA process is halted at the decomposition phase, and the search term is selected from the rule itself. Usually, the rule is akin to the following: “I am looking for (1)”, where (1) denotes the search term. This search term is then passed off to the search engine, which returns file paths for the most relevant documents.

This entire process is depicted in Fig. 15.

![Functional Design of ELIZA](image)

*Figure 15: Functional Design of ELIZA*

### 5.1.1.1.2 RAAM

Functionally speaking, the RAAM process is quite different from its intended usage.
When an input is received from the user, it is translated into codewords, using the dictionary, and then broken down into a parse tree. Then the input is fed to the RAAM. The resultant output should closely be similar to one of the trained phrases. The closest trained phrase is selected and this phrase is compared to the user's input to detect the search term, which is then passed off to the search engine.

Therefore rather than understanding the sentence, the RAAM is merely used to link a user's input to one of the trained phrases. In essence, the RAAM replaces the keyword scanning mechanism of ELIZA.

This entire process is depicted in Fig. 16.

![Functional Design of the RAAM](image-url)
5.1.1.2 Search Engine

The search engine, upon the reception of a keyword, queries the topic model as explained in Section 3.4.2.2. It returns the path of the most relevant document. In addition to the topic model, a search engine based on frequency relationship between words and documents was included as a control experiment.

However, there is another functional part of the search engine, which is the data-miner. This scans each document in the corpus, tracks word frequencies for each word in the document, and stores this information in the database.

5.1.2 Structural Design

Now that we understand what the specific sub-systems were required to do, let us examine the structural aspect of the design process.

5.1.2.1 Database

We will examine the entity-relationship diagrams for the data-mining and the topic-modeling tables in the database. The RAAM uses only one table, while ELIZA uses none, so they are ignored.
### 5.1.2.1.1 Data-Mining

![Data-Mining Entity-Relationship Diagram]

*Figure 17: Data-Mining Entity-Relationship Diagram*
5.1.2.1.2 Topic Modeling

There are two sub-systems to the Interface system: ELIZA and RAAM. Let us analyze each of these in more detail.

5.1.2.2 Interface

There are two sub-systems to the Interface system: ELIZA and RAAM. Let us analyze each of these in more detail.

5.1.2.2.1 ELIZA

Structurally, ELIZA is quite simple. It processes input into output using the script. This is shown in Fig. 19.
Figure 19: Structural Design of ELIZA
5.1.2.2 Recursive Distributed Representation

Structurally, the Recursive Distributed Representation is also quite simple. It processes input by breaking it down with a parse tree, using a dictionary, and then uses the RAAM, to generate the output, which is a trained phrase. This is depicted in Fig. 20.

![Diagram of Recursive Distributed Representation]

```
Figure 20: Structure of the Recursive Distributed Representation
```

5.1.2.3 Search Engine

The topical search engine uses three sparse matrices: word-topic matrix; topic-document matrix; and, topic-frequency matrix. The first is used to select the most relevant topic for a keyword, which is then passed on to the second to select the most relevant document for that topic. The final matrix is used mainly to normalize the other matrices.
The frequency-based search engine selects the most relevant document for a keyword based on how many times that word appears in that document.

5.1.2.4 Data Mining

A corpus of several documents is passed to a stock document miner, which is an external module. For each word in each document, a frequency is calculated. This is a sparse matrix and stored in the database upon completion.

![Data Mining Structure](image)

*Figure 21: Data Mining Structure*

5.2 Implementation

5.2.1 Database

The database used is MySQL. More importantly, we will examine each of the tables used in the database. This is crucial to understanding the data structures used by the rest of the system.
5.2.1.1 Data-Mining

The relationships between these tables were examined in Section 5.1.2.1.1.

**fits_documents**
Holds full document information to prevent double parsing

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc_id</td>
<td>integer</td>
<td>An identifier for each document</td>
</tr>
<tr>
<td>full_path</td>
<td>longtext</td>
<td>The full path for each document</td>
</tr>
<tr>
<td>last_modified_date</td>
<td>datetime</td>
<td>The last modified date for that document</td>
</tr>
</tbody>
</table>

**keyword_data**
Holds word frequency counts

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>word_id</td>
<td>integer</td>
<td>An identifier for each word</td>
</tr>
<tr>
<td>doc_id</td>
<td>integer</td>
<td>An identifier for each document</td>
</tr>
<tr>
<td>count</td>
<td>integer</td>
<td>The frequency count for that word</td>
</tr>
</tbody>
</table>

**keyword_docid**
Holds partial document information for FullTextSearch usage

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>Relative path for each document</td>
</tr>
<tr>
<td>id</td>
<td>integer</td>
<td>An identifier for each document</td>
</tr>
</tbody>
</table>

**keyword_fts**
FullTextSearch settings

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>param</td>
<td>string</td>
<td>Settings parameter</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>Settings value</td>
</tr>
</tbody>
</table>
**keyword_fts_words**
Holds word identification information

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>string</td>
<td>The word in full</td>
</tr>
<tr>
<td>id</td>
<td>integer</td>
<td>An identifier for each word</td>
</tr>
</tbody>
</table>

**sl_en_stoplist**
Holds a black-list for trivial words which are not to be mined

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>string</td>
<td>The word in full</td>
</tr>
</tbody>
</table>

### 5.2.1.2 RAAM

**raam_settings**
RAAM settings

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>Settings key</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>Settings value</td>
</tr>
</tbody>
</table>

**raam_weights**
Weights for all the links in the RAAM

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>set</td>
<td>Type of weight link (compressor or reconstructor)</td>
</tr>
<tr>
<td>input_id</td>
<td>integer</td>
<td>The id of the input neuron for the link</td>
</tr>
<tr>
<td>output_id</td>
<td>integer</td>
<td>The id of the output neuron for the link</td>
</tr>
<tr>
<td>value</td>
<td>double</td>
<td>The weight of the link</td>
</tr>
</tbody>
</table>
5.2.1.3 Topic Model

The relationship between these tables was examined in Section 5.1.2.1.2.

**topicmodel_document_assignments**
Holds topic-document frequency counts

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc_id</td>
<td>integer</td>
<td>An identifier for the document</td>
</tr>
<tr>
<td>topic_id</td>
<td>integer</td>
<td>An identifier for the topic</td>
</tr>
<tr>
<td>frequency</td>
<td>integer</td>
<td>The frequency count for that relationship</td>
</tr>
</tbody>
</table>

**topicmodel_document_probabilities**
Holds topic-document probabilities

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc_id</td>
<td>integer</td>
<td>An identifier for the document</td>
</tr>
<tr>
<td>topic_id</td>
<td>integer</td>
<td>An identifier for the topic</td>
</tr>
<tr>
<td>probability</td>
<td>integer</td>
<td>The probability for that relationship</td>
</tr>
</tbody>
</table>

**topicmodel_word_assignments**
Holds word-topic frequency counts

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>word_id</td>
<td>integer</td>
<td>An identifier for the word</td>
</tr>
<tr>
<td>topic_id</td>
<td>integer</td>
<td>An identifier for the topic</td>
</tr>
<tr>
<td>frequency</td>
<td>integer</td>
<td>The frequency count for that relationship</td>
</tr>
</tbody>
</table>
**topicmodel_word_probabilities**

Holds word-topic probabilities

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>word_id</td>
<td>integer</td>
<td>An identifier for the word</td>
</tr>
<tr>
<td>topic_id</td>
<td>integer</td>
<td>An identifier for the topic</td>
</tr>
<tr>
<td>probability</td>
<td>integer</td>
<td>The probability for that relationship</td>
</tr>
</tbody>
</table>

**topicmodel_topic_assignments**

Holds topic frequency counts

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic_id</td>
<td>integer</td>
<td>An identifier for the topic</td>
</tr>
<tr>
<td>frequency</td>
<td>integer</td>
<td>The frequency count for that topic</td>
</tr>
</tbody>
</table>

**topicmodel_token_assignments**

Holds word token-topic frequency counts

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>token_id</td>
<td>integer</td>
<td>An identifier for the word token</td>
</tr>
<tr>
<td>topic_id</td>
<td>integer</td>
<td>An identifier for the topic</td>
</tr>
<tr>
<td>count</td>
<td>integer</td>
<td>The frequency count for that word token</td>
</tr>
</tbody>
</table>

**topicmodel_settings**

Settings for the topic model

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>Settings key</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>Settings value</td>
</tr>
</tbody>
</table>

**5.2.2 Interface**

We must examine how interface elements such as ELIZA and RAAM were implemented.
5.2.2.1 **ELIZA**

For ELIZA, the Chatbot::Eliza module, version 1.04, created by John Nolan, was used. I adapted the default script for my own purposes, and this script is attached as Appendix I.

5.2.2.2 **Recursive Distributed Representation**

The Recursive Distributed Representation module was implemented almost entirely by me. It includes a back-prop neural network, complete with the changes needed in order to make it suitable for the RAAM. The parse tree uses a binary tree module, Tree::Binary, version 0.07, written by Stevan Little.

The dictionary used for this module is attached as Appendix II, and the original training phrases is attached at Appendix III.

5.2.3 **Search Engine**

The search engine comprises a frequency search and a topic model search. Both were implemented by me. Each of these uses simple SQL statements in order to return the most relevant document.

5.2.4 **Data Miner**

The data-miner comprises a document miner and a topic model. The first is implemented using DBIx::FullTextSearch version 0.73, designed by T.J. Mather and Jan Pazdziora. The document miner also includes an HTML parsing module, called HTML::TreeBuilder, version 3.23, developed by Pete Krawczyk. It requires the use of Sean Burke's HTML::Format, version 2.04.

---

5 Little, S. *Tree-Binary-0.07*. Infinity Interactive Inc, 2005. Available at: http://search.cpan.org/~stevan/Tree-Binary-0.07/
6 Mather, T.J. *DBIx-FullTextSearch-0.73*. Maxmind, 2003. Available at: http://search.cpan.org/~tjmather/DBIx-FullTextSearch-0.73/
The other element is the topic model which include the Gibbs' Sampler and was written entirely by me, following some of the specifications by Steyvers' and Griffiths' own topic modeling software.
6 Results

Now that we've examined the system, we must examine how the models fared.

6.1 ELIZA

ELIZA was used as a control experiment for comparison with Recursive Distributed Representation. It was fairly successful at recognizing search terms from trained phrases, but it couldn't deal with words or phrases it hadn't learned. This was intended as ELIZA was used to mimic the power of current systems, and to compare with the results from the Recursive Distributed Representation.

6.2 Recursive Distributed Representation

Recursive Distributed Representation provided different results than ELIZA. It was able to deal with untrained words or phrases, but performed poorly at identifying search terms.

For example, if the user inputs “I am looking for X,” ELIZA was able to identify X as the search term. Recursive Distributed Representation, however, seemed to identify “looking” as the search term. So, in such simple situations, it failed to work.

However, when the user inputs “I am seeking X,” ELIZA failed to identify the phrase at all, as it wasn't part of the ELIZA script. Recursive Distributed Representation identified X as the search term, even though “seeking” did not exist in the dictionary.

6.3 Topic Modeling

Topic Modeling failed to provide relevant results. For example, while searching for scientific terms, it
returned documents dealing with linguistics. Frequency search, however, returned the scientific documents. This was found to be related to the size of the documents: the larger the documents, the more they affected topics in their bias.

Normalization is a remedy for this problem, which is actually suggested by the authors of the model themselves.
7 Conclusion

The production of a digital assistant, with the use of NLP and other artificial intelligence techniques, requires an understanding of broader issues, including data structures, data-mining, data-retention and linguistics, amongst others.

With specific regard to this project, simplification of the aforementioned requisite knowledge has allowed me to pursue the development of a simulation of a digital assistant without being constrained by an extensive research period.

The results proved to be average. The Recursive Distributed Representation was able to handle words it had not learnt, but performed mediocly at selecting search terms. ELIZA was better at selecting search terms, but only for phrases it had learnt. This was expected, as ELIZA can only operate with rigid input. The topic model wasn't able to provide relevant results every time, but this was found to be related to the influence of larger documents on smaller documents. Normalization or other methods may fix this issue.

Generally speaking, this project is designed to break the barrier between the theoretical and the practically useful spheres of artificial intelligence. By bringing an AI-enabled application to the user, it is hoped that the field of artificial intelligence will further itself, at least in the long-term, with the user, rather than the ivory towers of academia, in mind. Overall, these NLP techniques have proven to have some promise towards this end. More specific improvements to each of these models will result in better performance.
8 References


Appendices
Appendix I: ELIZA Script

initial: How do you do? Please tell me what you are looking for.
final: Goodbye.
quit: bye
quit: goodbye
quit: done
quit: exit
quit: quit
pre: don't
dont: don't
cant: can't
pre: won't
wont: won't
recollect remember
pre: recall remember
dreamt dreamed
dream: dream
may: maybe
maybe: perhaps
certain: certainly
yes
pre: machine
computer
pre: machines
computer
pre: computers
computer
post: am are
your: your
my: my
yours: yours
mine: mine
were: were
was: was
me: me
you
you're: you are
i'm: i am
myself: myself
yourself: yourself
i you
you: me
post: my
your
me you
i you are
same: alike
identical: alike
equivalent: alike
believe: feel: think
wish
family: mother
mom: father
dad: sister
brother
wife
children
child
desire: want
need
sad: unhappy
depressed
sick
happy:
elated
better
cannot: can't
everyone
nobody
noone
be: am
is are
was
key: xnone
-1
decomp: *
reasm: I'm not sure I understand you fully.
reasm: Please go on.
reasmb: That is interesting. Please continue.
reasmb: Tell me more about that.
reasmb: Does talking about this bother you?
key: sorry 0
decomp: *
reasmb: Please don't apologise.
reasmb: Apologies are not necessary.
reasmb: I've told you that apologies are not required.
reasmb: It did not bother me. Please continue.
key: apologise 0
decomp: *
reasmb: goto sorry
key: xfremd 0
decomp: *
reasmb: I speak only English.
key: hello 0
decomp: *
reasmb: How do you do. Please state your problem.
reasmb: Hi. What seems to be your problem?
key: yes 0
decomp: *
reasmb: Please go on.
reasmb: Please tell me some more about this.
reasmb: Why don't you tell me a little more about this.
reasmb: I see.
reasmb: I understand.
key: no 0
decomp: * no one *
reasmb: Are you sure, no one (2) ?
reasmb: Surely someone (2) .
reasmb: Can you think of anyone at all ?
reasmb: Are you thinking of a very special person ?
reasmb: Who, may I ask ?
reasmb: You have a particular person in mind, don't you ?
reasmb: Who do you think you are talking about ?
decomp: *
reasmb: Are you saying no just to be negative?
reasmb: Does this make you feel unhappy?
reasmb: Why not ?
reasmb: Why 'no' ?
key: fuck 10
decomp: *
reasmb: goto xswear
key: fucker 10
decomp: *
reasmb: goto xswear
key: shit 10
decomp: *
reasmb: goto xswear
key: damn 10
decomp: *
reasmb: goto xswear
key: shut 10
decomp: * shut up *
reasmb: goto xswear
key: xswear 10
decomp: *
reasmb: Does it make you feel strong to use that kind of language?
reasmb: Are you venting your feelings now?
reasmb: Are you angry?
reasmb: Does this topic make you feel angry?
reasmb: Is something making you feel angry?
reasmb: Does using that kind of language make you feel better?
key: find 7
decomp: I can't find *
reasmb: Please wait while I search for (1)
reasmb: I am looking for "(1)" now. Please hold.
decomp: I am trying to find *
reasmb: Please wait while I search for (1)
reasmb: I am looking for "(1)" now. Please hold.
key: need 7
decomp: I need *
reasmb: Please wait while I search for (1)
reasmb: I am looking for "(1)" now. Please hold.
key: search 7
decomp: I can't search *
reasmb: Please wait while I search for (1)
reasmb: I am looking for "(1)" now. Please hold.
decomp: I am trying to search for *
reasmb: Please wait while I search for (1)
reasmb: I am looking for "(1)" now. Please hold.
key: finding 7
decomp: I am finding *
reasmb: Please wait while I search for (1)
reasmb: I am looking for "(1)" now. Please hold.
key: searching 7
decomp: I am searching for *
reasmb: Please wait while I search for (1)
reasmb: I am looking for "(1)" now. Please hold.
key: looking 7
decomp: I am looking for *
reasmb: Please wait while I search for (1)
reasmb: I am looking for "(1)" now. Please hold.
key: where 7
decomp: Where is *
reasmb: Please wait while I search for (1)
reasmb: I am looking for "(1)" now. Please hold.
decomp: * where * is
reasmb: Please wait while I search for (2)
Appendix II: RAAM Dictionary

# Five categories: Empty; Noun; Verb; Gerund; Prepositions
# Syntax:
# cccc iiii 

# Empty variable for unknown words
# keyword (n.) is here to make it similar to the empty value
0 0,0,0,0,0,0,0,0,0,0,0,0
keyword 0,0,0,0,0,0,0,0,0,0,0,1

# Noun
i 0,0,0,1,0,0,0,0,0,0,0,1

# Verb
am 0,0,1,0,0,0,0,0,0,0,0,1
can't 0,0,1,0,0,0,0,0,0,0,1,0
find 0,0,1,0,0,0,0,0,0,1,0,0
is 0,0,1,0,0,0,0,1,0,0,0,0
look 0,0,1,0,0,0,0,1,0,0,0,0
need 0,0,1,0,0,1,0,0,0,0,0,0
search 0,0,1,0,0,1,0,0,0,0,0,0
try 0,0,1,0,1,0,0,0,0,0,0,0

# Gerund
finding 0,1,0,0,0,0,0,0,0,1,0,0
looking 0,1,0,0,0,0,0,1,0,0,0,0
searching 0,1,0,0,0,0,1,0,0,0,0,0
trying 0,1,0,0,0,1,0,0,0,0,0,0

# Prepositions
for 1,0,0,0,0,0,0,0,0,0,0,1
to 1,0,0,0,0,0,0,0,0,1,0,0
where 1,0,0,0,0,0,0,0,1,0,0,0
Appendix III: RAAM Training Phrases

I can't find keyword
I am trying to find keyword
I need keyword
I can't search keyword
I am finding keyword
I am searching for keyword
I am looking for keyword
Where is keyword