Chapter 5

Lemmatization Using Trie Data Structure and Hopfield Network

5.1 Introduction

Supervised WSD requires sense marked corpora. The sense comes from WordNet or Thesaurus. But both these knowledge sources store words in its root form. Words in the corpora are present in morphed form- both inflectional and derivational. In the lexical knowledge-bases like dictionary, Word Net, etc, the entries are usually root words with their morphological and semantic descriptions and therefore, when a morphed word is encountered in a raw text, itcannot be sense tagged unless and until its appropriate root word is determined through lemmatization.Thus, lemmatization is a basic need for any kind of semantic processing for languages.

A corpus will have words belonging to both inflectional and derivational morphology class. Inflection is the combination of a word stem with a grammatical morpheme , usually resulting in a word of the same class as the original stem , and usually fulfilling some syntactic function like agreement. For e.g. the word মানবতার (humanity's) has root or stem মানবতা. So মানবতার and মানবতা belongs to the same word class. But the root word মানবতা is only available in the WordNet. Therefore the basic function of a lemmatizer is to identify the root form of a word from its morphed form.

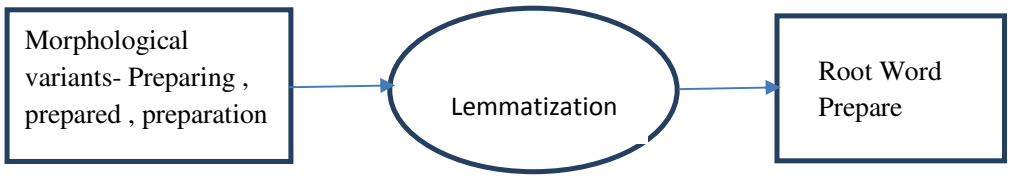

Figure 5.1 Lemmatization Process

5.2 Approaches for Lemmatization

Following are the approaches for lemmatization

- Edit Distance on dictionary algorithm which is combination of string matching and most frequent inflectional suffixes model.
- Morphological Analyzer which is based on "finite state automata".
- Affix lemmatizer which is combination of rule based and supervised training approach
- Fixed length truncation approach.
- Trie data structure which allow retrieving possible lemma of a given inflected or derivational form.

5.2.1 Levenshtein Distance

The Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e. Insertions, deletions or substitutions) required to change one word into the other[85].

Searching similar sequences of data is of great importance to many applications such as the gene similarity determination, speech recognition applications, database and/or Internet search engines, handwriting recognition, spell-checkers and other biology, genomics and text processing applications.

Therefore, algorithms that can efficiently manipulate sequences of data (in terms of time and/or space) are highly desirable, even with modest approximation guarantees.

The Levenshtein Distance of two strings A and B is the minimum number of character transformation required to convert string A to string B.

The following Equation 5.1used two find the Levenshtein distance between two strings a, b is given by:

$Liv_{(a,b)}(|a|,|b|)$ where:

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & if \min(i,j) = 0 \\ lev_{a,b}(i-1,j)+1 & \\ lev_{a,b}(i,,j-1)+1 & Otherwise \\ lev_{a,b}(i-1,j-1)+1_{a_i \neq b_i} & \end{cases} \quad \text{.....5.1}$$

The first element in the minimum corresponds to deletion (from a to b), the second to insertion and the third to match or mismatch, depending on whether the respective symbols are the same.

The edit distance algorithm is performed by using three most "primitive edit operation". By term primitive edit operation we refer to the substitution of one character to another, the deletion of a character and insertion of a character. So this algorithm can be performed by three basic operations like insertion, deletion and substitution. Some approached focused on suffix phenomena only. But this approach deals with both suffixes as well as prefixes. So it is known as affixation phenomena. Sometime it happens that suffixes added into the words are based on grammatical rules. For example in the case of word "going", this approach return headword "go". But for word "went", it returns discrete entry of lemma in dictionary. The idea is to find out all possible lemma for user's input word.

For each one of the target words, the similarity distance between the source and the target word is calculated and stored. When this process is completed, the algorithm returns a set of target words having the minimum edit distance from the source word.Sothe algorithm compare user input to the all available stored lemmas. The word that is at a minimum distance from the target word is retrieved.

The algorithm provides the option to select the value of the approximation that the system considers as desired similarity distance (e.g. if the user enters zero as the desired approximation, then only the target words with the minimum edit distance will be returned, whereas if he/she enters e.g. 2 as the desired

approximation, then the returned set will contain all the target words having a distance <=(minimum + 2) from the source word. This approach also distinguishes words like "entertained" and "entertainment". Its return entertain for the entertained word but not for entertainment, because entertainment is itself a noun and is different than entertained.

5.2.2 Morphological Analyzer Based Approach

A morphological analyzer is a program for analyzing the morphology of an input word, it detects morphemes of any text[86]. A Morphological Analyzer gives all possible analyses for a given word which is based on finite state technology, and it produces the morphological analysis of the word form as its output. This approach uses finite state automata and two level morphology to build a lexicon for a language with infinite vocabulary. Two-Level rules are declarative constraints that describe morphological alternations, such as the y->ie alternation in the plural of some English nouns (spy->spies). The aim of this approach is to convert two-level rules into deterministic, minimized finite-state transducers. It describes the format of two-level grammars, the rule formalism, and the user interface to the compiler.

It also explains how the compiler can assist the user in the development of a two-level grammar. A finite state transducer (FST) is a finite state machine with two tapes: an input tape and an output tape. This contrasts with an ordinary finite state automaton (or finite state acceptor), which has a single tape. Transducer means to translate a word from one state to another. Transducer is having two states, one is input tape and another is output tape.

Finite state transducer is 6-tuple (Q, Σ, Γ, I, F, δ) such that:

• Q is a finite set, the set of states;

• Σ is a finite set, called the input alphabet;

• Γ is a finite set, called the output alphabet;

• I is a subset of Q, the set of initial states;

• F is a subset of Q, the set of final states; and

$$\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times Q$$

(Where ε is the empty string) is the transition relation.

FSM give input actions and output depends on only state. State change from input tap to output tap based on this action performed.

5.2.3 Affix Lemmatizer

The most common approach for word normalization is to remove affix from a given word. Suffix or prefix are removed as per grammatical rulesof the language. To just remove suffix or prefix from word cannot give accurate head word or root word. To just use rule based approach cannot give accurate result so by combining rule based approach to some statistical approach like supervised training can give more accurate result.

Supervised training algorithm generates a data structure consisting of rules that a lemmatizer must traverse to arrive at a rule that is elected to fire[87]. After training, the data structure of rules is made permanent and can be consulted by a lemmatizer. The lemmatizer must elect and fire rules in the same way as the training algorithm, so that all words from the training set are lemmatized correctly. It may however fail to produce the correct lemmas for words that were not in the training set. For training words this approach uses prime and derived rules. Prime rule for training is the least specific rule that is needed to lemmatize whereas derived rules are more specific rule which can be created by adding or removing characters.

For example rule can be "watcha" which is derived from what are you, "yer" which is derived from you are rather than "your". This approach is more generalized than only suffix removal approach. The bulk of ‘normal’ training words must be bigger for this type of lemmatizer. This is because the algorithm generates immense numbers of candidate rules with only marginal differences in accuracy, requiring many examples to find the best candidate.

5.2.4 Fixed Length Truncation

In this approach, we simply truncate the words and use the first 5 and 7 characters of each word as its lemma. In this approach words with less than n characters are used as a lemma with no truncation. This approach is most appropriate for the languages like Turkish which has average length of 7.07 letters. This approach is used when time is of utmost priority. It is the simplest approach not dependent on any language or grammar. So it can be applied to any language.

5.2.5 Trie Based Approach

The trie data structure is one of the most important data storage mechanisms in programming. It's a natural way to represent essential utilities on a computer like the directory structure in a file system[58].

A trie, or prefix tree, is an ordered tree data structure that is used to store an associative array where the keys are usually strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree shows what key it is associated with. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string.

Many other objects can be stored in a tree data structure resulting in space and/or time efficiency. For example, when we have a huge number of dictionary (and/or non-dictionary) words or string that we want to store in memory we can use a tree structure to efficiently store the words instead of using a plain Array or Vector type that simply stores each word individually in memory.

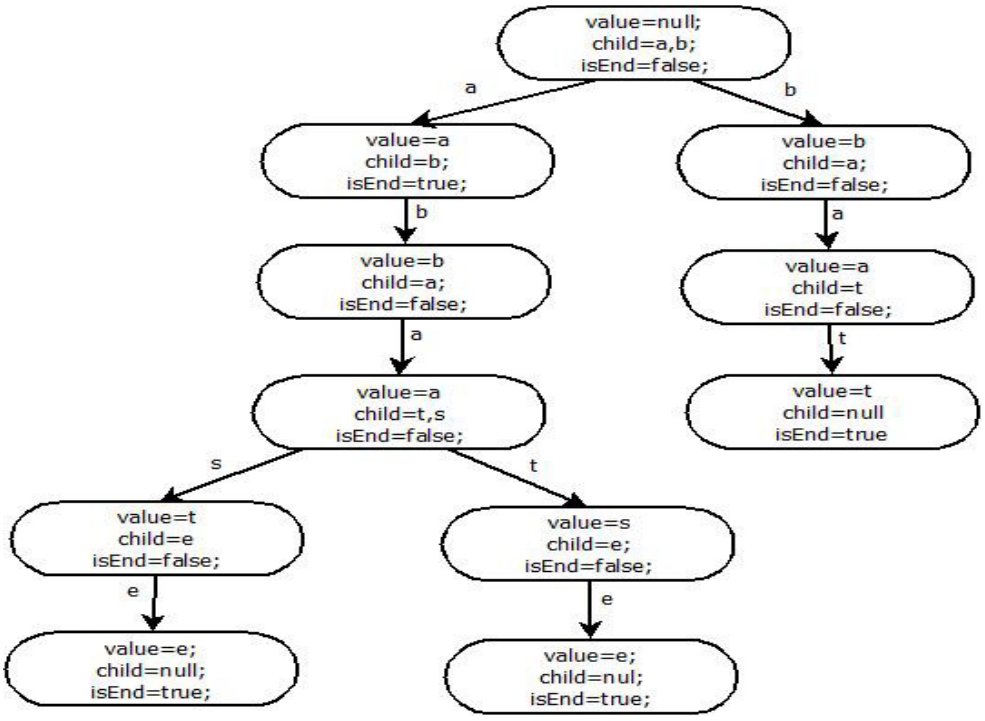

Figure 5.2Trie for a language consisting of words a , abase , abate and bat:

5.3 Basic Operations In Trie

Following are the basic operations in a trie:-

5.3.1 Searching In A Trie

To search for a key k in a trie T, we begin at the root which is a branch node. Let us suppose the key k is made up of characters k1 K2 k3….kn[58] The first character of the key K viz., k1 is extracted and the pChildren field corresponding to the letter k1 in the root branch node is spotted. If T->pChildren[k1- 'a'] is equal to NULL, then the search is unsuccessful, since no such key is found. If T->pChildren[k1–'a'] is not equal to NULL, then the pChildren field may either point to an information node or a branch node. If the information node holds K then the search is done. The key K has been successfully retrieved. Otherwise, it implies the presence of key(s) with a similar prefix. We extract the next character k2 of key K and move down the link field corresponding to k2 in the branch node encountered at level 2 and so on until the key is found in an information node or the search is unsuccessful.

The deeper the search, the more there are keys with similar but longer prefixes.

The search algorithm consists of the following steps:-

1. For each character in the string, see if there is a child node with that character as the content.
2. If that character does not exist, return false.
3. If that character exist, repeat step 1.
4. Do the above steps until the end of string is reached.
5. When end of string is reached and if the marker (NotLeaf) of the current Node is set to false, return true,else return false.

5.3.2 Insertion into a trie

The insertion algorithm consists of the following steps:-

1. Set current node to root node i.e. value=null
2. Set the current letter to the first letter in the word.
3. If the current node already has an existing reference to the current letter then set current node to that referenced node; else create a new node, set the letter to current letter, and set current node to this new node, set the value of isEnd to false.

4. Repeat step 3 until all letters in the current word has been processed. Set the value of
5. isEnd=true when the process end.

5.4 Algorithm For Lemmatization based on Trie Data Structure

The algorithm requires a file containing the list of the root words (lemma) of the language concerned.

At first, we need to create a trie structure using the dictionary root words[58]. Each node in the trie corresponds to a Unicode character of the language concerned and the nodes that end with the final character of any root word are marked as final nodes. The rest of the nodes are marked as non-final nodes. The key idea is that a trie is created out of the vocabulary (root words) of the language.

The lemmatizing process consists in navigating the trie, trying to find a match between the input word and an entry in the trie. To find the lemma of a surface word, the trie is navigated starting from the initial node in the trie and navigation ends when either the word is completely found in the trie or after some portion of the word there is no path present in the trie to navigate.

While navigating, some situations may occur, depending on which we take decision to determine the lemma. Those situations are described below.

CASE 1:

The surface word is a root word. In that case, the surface word itself is the lemma.

Example:

Stored word: abbreviate

Input: abbreviate

Matched String: Abbreviate

Output: Abbreviate

CASE 2:

The surface word is not a root word. In that case, the trie is navigated up to that node where the surface word completely ends or there is no path to navigate in the trie. We call this node as the end node. Here two different cases may occur.

CASE 2.1:

In the path from the initial node to the end node, if one or more than one root word is found i.e. if one or more final nodes are present in the path then we should pick that final node which is closest to the end node.

Example:

Stored words: a, an, and

Input: ands

Matched prefix: a, an, and;

Output: and

The word represented by the path from initial node to the picked final node is considered as the lemma.

CASE 2.2

If no root word is found in the path from the initial node to the end node.

Then find the final node in the trie which is closest to the end node.

Example:

Stored word: abbreviate

Input: abbreviating

Matched String: abbreviat

Output: abbreviate

The figures 5.3-5.5 show some of the outputs from the lemmatizer that we have developed based on the trie data structure.

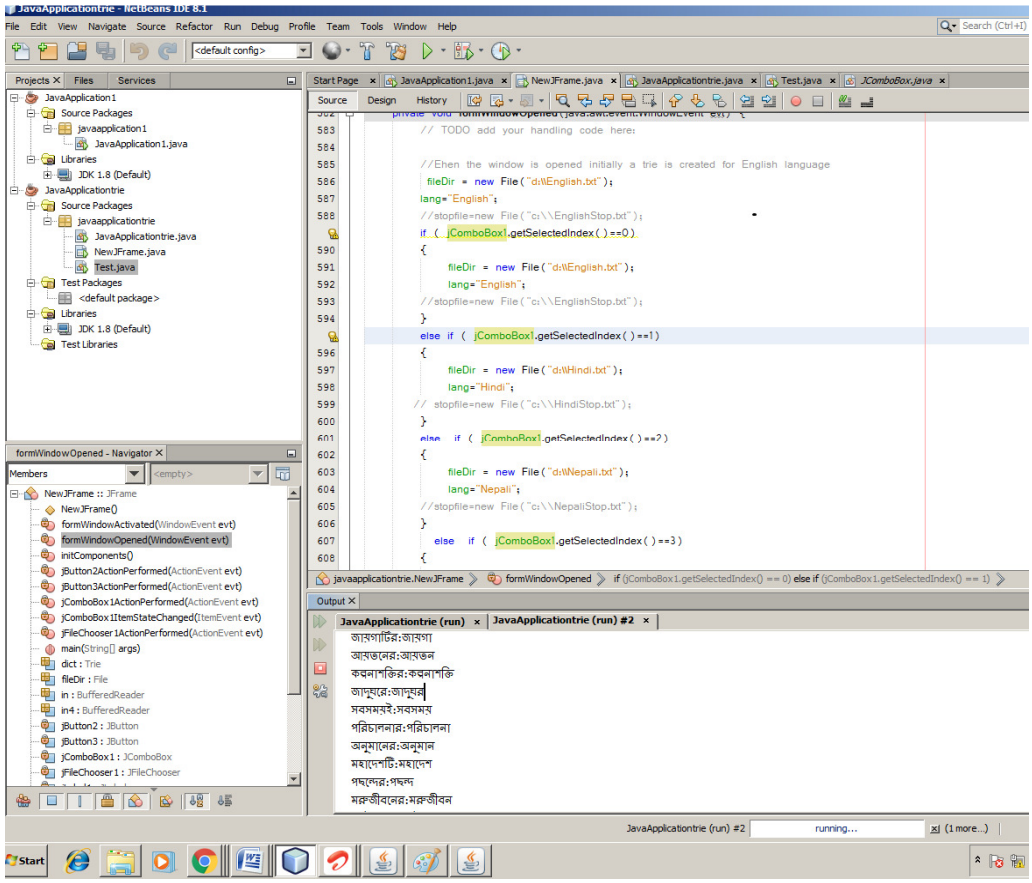

Figure 5.3 Lemmatizer showing the output for some Bengali inflected words

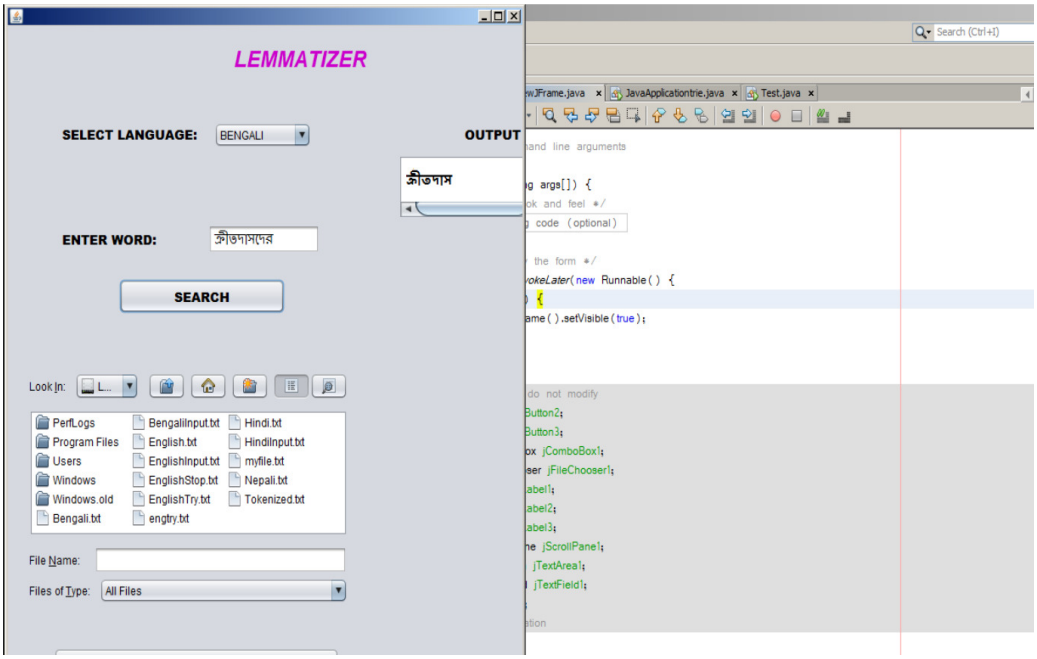

Figure 5.4Lemmatizer showing the output for Bengali inflected wordক্রীতদাসদের

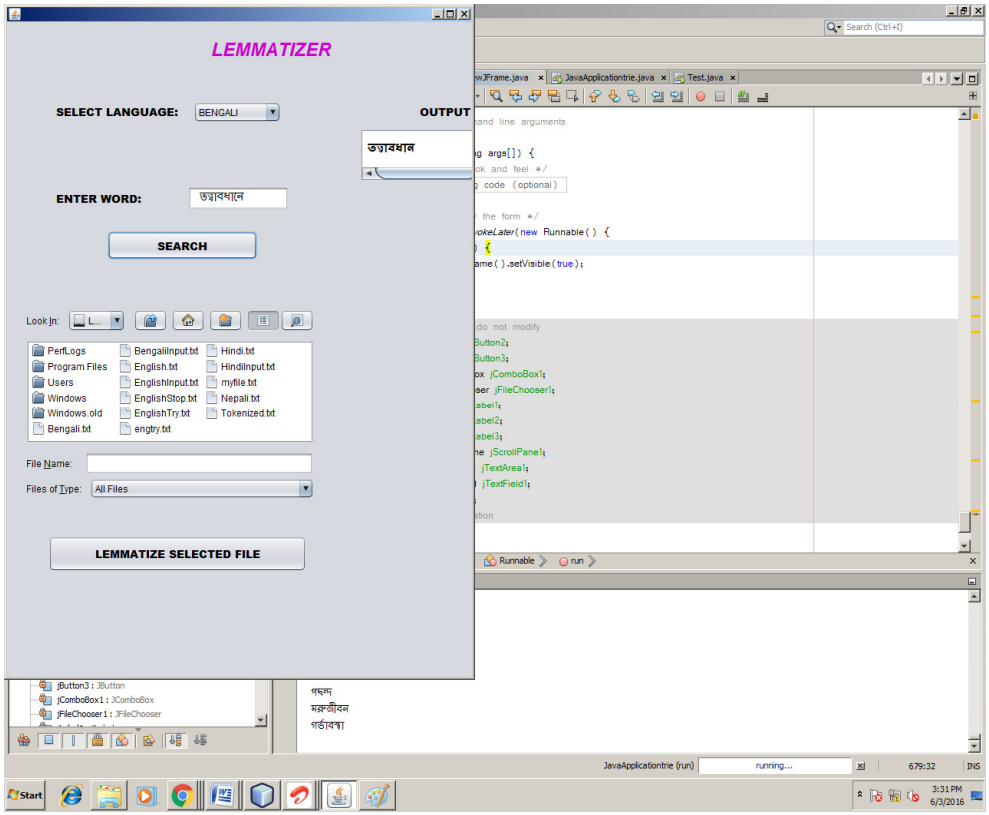

Figure5.5 :Lemmatizer showing the output for Bengali inflected word তত্ত্বাবধানে

5.5 Hopfield Network

Hopfield networks are constructed from artificial neurons[81] . Theseartificial neurons have N inputs. Each input ihas a weightw_i associated with it .They also have an output. The state of the output is maintained, untilthe neuron is updated. Updating a neuron is performed by the following rule:

$$s_i = \begin{cases} +1, & if \sum_j w_{ij} s_j > \theta_i \\ -1 \text{ or } 0, & otherwise \end{cases} \quad \text{.....(5.2)}$$

where

- w_{ij} is the weight of the connection between unit i and j
- s_j is the state of unit j
- θ_i is threshold of unit i.

A Hopfield network is a network of N such artificial neurons, which are fully connected. The connection weight from neuron j to neuron i is given by a number w_{ij} . The collection of all such numbers is represented by the weight matrix W, whose components are w_{ij}. In fact Hopfield network can be formally described as a complete undirected graph G= < V , F>where V is a set of McCulloch-Pitts neurons and f: $V^2 \rightarrow \mathbb{R}$ is a function that links pairs of nodes to a real value which is the weight between two units. Fig 5.6 shows a layout of Hopfield Network.

Given the weight matrix and the updating rule for neurons the dynamics of the network is defined if we tell in which order we update the neurons. There are two ways of updating neurons:

Asynchronous: One neuron is picked up, its weighted input sum is calculated and updating takes place immediately. This can be done in a fixed order, or neurons can be picked at random, which is called asynchronous random updating.

Synchronous: the weighted input sums of all neurons are calculated without updating the neurons. Then all neurons are set to their new value, according to the value of their weighted input sum.

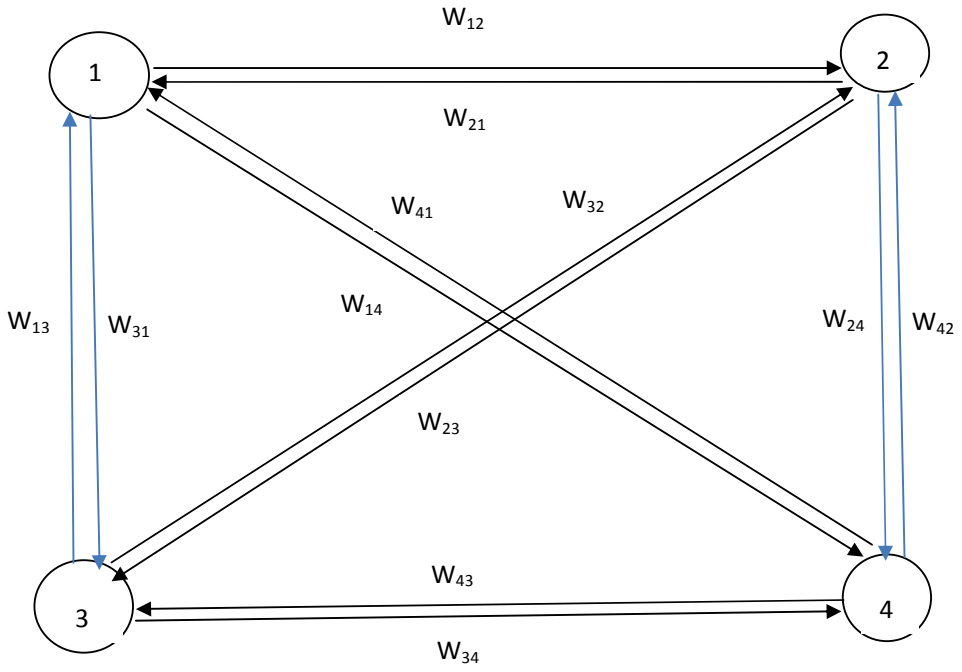

Figure 5.6: Layout of Hopfield Network

The connections in a Hopfield net typically have the following restrictions:

$w_{ii} = 0 \; \forall i$(no unit has a connection with itself)

$w_{ij} = w_{ji} \forall$i,j (connections are symmetric)

5.5.1 Attraction and repulsion between neurons

The weight between two units has a powerful impact upon the values of the neurons. Let us consider the connection weightw_{ij} between 2 neurons i and j. if w_{ij}> 0 updating rule implies the following:

If s_j =1 , the contribution of s_j toward weighted sum is positive and s_j pulls the value of s_i towards its value of 1.

If s_j =-1, the contribution of s_j toward weighted sum is negative and s_j pulls the value of s_i towards its value of -1.

So when the weight between two connections is positive neurons i and j will converge whereas when the weight between two connections is negative neurons i and j will diverge.

5.6 Energy of a Hopfield Network

Hopfield nets have a scalar value associated with each state of the network referred to as the "energy", E, of the network, where:

E= $-\sum \theta_i \, S_i +$

$\sum_{i=1}^{N} \sum_{j>1}^{N} W_{ij} * S_i * S_j$.....5.3

This value is called the energyand the definition ensures that when units are randomly chosen to update, the energy E will either lower in value or stay the same.

The change in energy is due to a change in the state of the neuron and is given by

$\Delta E = -\left[\sum_i S_j \, W_{ij} + S_i - \theta_i \Delta S_i \right]$.....5.4

If S_i is positive , it will change to -1/0 if

$S_i + \sum_i S_j \, W_{ij} < \theta_i$

From equation 5.4 it is seen that ΔE becomes negative and hence ΔE<0.

If S_i is negative/0, it will change to +1/1 if

$S_i + \sum_i S_j W_{ij} > \theta_i$

Again from equation 5.4 it is seen that ΔE becomes negative and hence ΔE<0.

So from both the cases we see that energy cannot increase for both positive and negative change in S_i and so the net must reach a stable equilibrium such that energy does not change with further iteration [81].

5.7 Chapter Summary

This chapter has discussed several lemmatization strategies with a special emphasis on trie data structure. Lemmatization is of high priority in language processing because words in corpora come in morphed forms and for sense tagging of the corpora, the root word has to be generated from the morphed forms as only root words are present in WordNet.

Hopfield network has also been briefly discussed in this chapter as we shall use this network , especially the energy function would be adapted for use as a scoring function to score the competing senses of a polysemous word thereby facilitating WSD.