A Project Report

on

"Word Sense Disambiguation (WSD)"

Submitted to the

Savitribai Phule Pune University

In partial fulfillment for the award of the Degree of

Bachelor of Engineering

in

Information Technology

by

Ankit Bhawsar (407001 BE-1)

Anup Bandawar (407002 BE-1)

Omkar Aurangabadkar (407005 BE-1)

Pratik Zode (407062 BE-1)

Under the guidance of

Mr. C. D. Kokane

Sinhgad Institutes

Department of Information Technology

STES's Sinhgad College of Engineering

Vadgaon (Bk.), Off. Sinhgad Road,

Pune 411041.

Semester-VII, Final Year Engineering

2019-2020

Sinhgad Institutes

CERTIFICATE

This is to certify that the preliminary project report entitled

"Word Sense Disambiguation (WSD)"

being submitted by
Ankit Bhawsar (407001 BE 1)
Anup Bandawar (407002 BE 1)
Omkar Aurangabadkar (407005 BE 1)
Pratik Zode (407062 BE 1)

is a record of bonafide work carried out by them under the supervision and guidance of Mr. C. D. Kokane and it is approved for the partial fulfilment of the requirement of Savitribai Phule Pune University for the award of the Degree of Bachelor of Engineering (Information Technology)
This project report has not been earlier submitted to any other Institute or University for the award of any degree or diploma.
Date: 5/10/2019

Place: Pune

Mr. C. D. Kokane	Prof. G. R. Pathak
Project Guide	Head of the Department

Dr. S. D. Lokhande
Principal

Internal Examiner	External Examiner

SINHGAD COLLEG OF ENGINEERING * Pune-41 * IT DEPT.

ACKNOWLEDGEMENT

We are highly indebted to our guide Mr. C. D. Kokane for his guidance, constant supervision as well as for providing necessary information regarding the project and also for his support in completing the project report. We would also like to express my special gratitude and thanks to the Staff Members of department of Information Technology for giving us such attention and time.

This acknowledgement would be incomplete without expressing our thanks to Prof. G. R. Pathak, Head of the Department (Information Technology) for his support during the work.

We would like to extend our heartfelt gratitude to our Principal, Dr. S. D. Lokhande who provided a lot of valuable support, mostly from behind the veils of college bureaucracy.

We would also like to express our gratitude towards our parents and friends for their kind co-operation and encouragement which help us in completion of this report.

Ankit Bhawsar
Anup Bandawar
Omkar Aurangabadkar
Pratik Zode

Abstract

Word Sense Disambiguation (WSD) is the task of removing ambiguity in different senses of words. It is a core research field in computational linguistics dealing with the automatic assignment of senses to words occurring in a given context. Humans are inherently good at WSD and distinguish senses used in words through spoken language. Computers on the other hand have difficulties identifying correct senses of words. Various advancements have been made in the task of disambiguation using mainly four approaches: Knowledge-based, Supervised, Semi-Supervised, and Unsupervised. Better understanding of the human language will help computer's performance in various applications such as search engine optimization, information retrieval, information extraction, software assistants, and voice command interpretation. The objective of this work is to present a supervised neural network model using machine learning algorithms dedicated to the task of maximizing accuracy of sense detection. The input layer of the neural network will consist of nodes having binary values depending on the presence or absence of frequently occurring context words related to the ambiguous words. The output layer will consist of nodes equal to the number of senses the ambiguous word has. Training and testing of the model will be done using lexical resources such as SemCor or OMSTI. Accuracy will be calculated based on All- Word tasks from SemEval International Workshops

Contents

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Introduction to Word Sense Disambiguation (WSD)

Word-Sense Disambiguation (WSD) is a branch of Natural Language Processing (NLP) which specifies some open problems concerned with identifying the sense of a word is used in a respective sentence. Many words used in the English language have various different senses or meanings. WSD is concerned with the problem of selecting the correct meaning. The solution to this problem impacts improving relevance of search engines.

The human mind is very proficient at word-sense disambiguation. Simple context is all that is needed for humans to understand the correct sense or meaning of a word. Human language developed in a way that reflects (and also has helped to develop) the innate ability provided by the brain's neural networks. In computer science and the information technology that it enables, it has been a long-term challenge to develop the ability in computers to perform natural language processing and machine learning.

For example, consider a word bass in English which has two meanings: any of various North American lean-fleshed freshwater fishes and the other: denoting the member of a family of instruments that is the lowest in pitch. Word sense disambiguation replaces the ambiguous word by the proper one depending on the surrounding context.

1.2 Motivation behind project topic

In today's modern world, people are heavily invested in a computer's ability to solve various problems in their daily lives. From finding directions via GPS to calculating their tax returns, most people are reliant on computer devices in one way or another. For better user experience and improved interfacing between man and machine, there needs to be clear communication between them. One obstruction in the way is the problem of ambiguity in word senses. In an effort to reduce this problem and enhance intelligence of computers, we propose our system for Word Sense Disambiguation.

1.3 Aim and Objective(s) of the work

Aim

The aim of this project is to create a model which maximizes accuracy of Word Sense Disambiguation (WSD) using Neural Networks.

Objectives

1. To understand the problem concerning word sense ambiguity.
2. To study the previous research work done in the field of Word Sense Disambiguation.
3. To find and acquire necessary labelled datasets for the purpose of training and testing.
4. To design and train a WSD model based on machine learning and neural network algorithms, with the goal of maximizing accuracy.
5. To compute accuracy of the model using appropriate testing data, and to compare it with other WSD models already documented.

1.4 Introduction to Machine Learning

Machine Learning (ML) is defined as programming computers to optimize a performance criterion using example data or past experience. In our system, the performance criterion is the accuracy of the model on testing data. Supervised ML consists of 2 main parts: Training and Testing.
Training comprises of feeding labelled data into the model to gain experience.
Testing comprises of predicting outputs by trained model based on experience.

1.5 Machine Learning in WSD

As stated before, the human brain is masterful at distinguishing between various senses of a word based on their context. The best way to reproduce this capability within machines is to make the computer think like humans do, allow it to learn from experience and make predictions based on this experience. The way this is implemented is Machine Learning, specifically using a Neural Network.

1.6 Artificial Neural Networks

AÂ neural networkÂ is a network or circuit ofÂ neurons composed ofÂ artificial neuronsÂ or nodes. Artificial neural networksÂ (ANN)Â are computing systems that are inspired by theÂ biological neural networksÂ that constitute humanÂ brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.Â

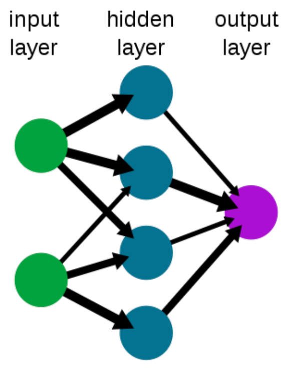


Figure 1.1: Simple Neural Network

Chapter 2

BACKGROUND OF WORD SENSE DISAMBIGUATION

2.1 Fundamentals

A rich variety of techniques have been studied, from knowledge-based methods that use the data encoded in lexical resources such as MRDs to supervised machine learning methods in which a classifier is trained for each word on a corpus of manually solved examples, to completely unsupervised methods that cluster occurrences of words, thereby inducing word senses.

2.1.1 Knowledge-based Approach

Knowledge based algorithms use various lexical resources such as Machine Readable Dictionaries (MRDs), WordNet to identify the correct sense of words.

These Algorithms are easy to implement and were the first to be developed while trying to solve the problem of WSD. A knowledge based system only needs access to commercial dictionary resources to start process of disambiguation.

Drawback of these algorithms is that their performance is limited on the speed of searching and retrieval of these resources. As the size of the resources increase, so does the latency and hence performance decreases.

Lesk Algorithm

The Lesk algorithm is the most influential dictionary-based method. It is based on the hypothesis that words used together in a sentence are related to each other and that the relation can be observed in the definitions of the words and their meanings. Two or more words are disambiguated by finding the pair of senses with the greatest word overlap in their dictionary definitions.

For example, when differentiating the words in "pine cone", the definitions of the appropriate senses both include the words evergreen and tree.

2.1.2 Supervised Approach

Supervised methods are based on the hypothesis that the context can provide enough indication on its own to disambiguate words (hence, common sense and reasoning are deemed unnecessary).

A learning set is prepared for the system to predict the actual meaning of an ambiguous word using a few sentences, having a specific meaning for that particular word. A system finds the actual sense of an ambiguous word for a particular context based on that defined learning set.

Supervised approach always gives superior performance than any other methods. However, these supervised methods are subject to a new knowledge acquisition holdup since they rely on considerable amounts of manually sense-tagged resources for training, which are arduous and expensive to create.

Naive Bayes Algorithm

Naive Bayes Method is a supervised approach. This method makes use of probabilistic approach which is one of the statistical methods, used to estimate probabilistic parameters. This probabilistic method usually expresses joint probability distribution or conditional probabilities in a given context and categories. Naive Bayes algorithm uses classifiers which are mainly based on Bayes theorems to calculate the conditional probability for each sense (say k) of a word for which the features are defined (x1, x2,. . . , xm). Let P(k) and P(xi/k) are the probabilistic parameters of the model and they can be projected from the training data, using relative frequency counts.[6]

$$\begin{aligned}\arg\max_{k} P(k \mid x_1,\ldots,x_m) &= \arg\max_{k} \frac{P(x_1,\ldots,x_m \mid k)P(k)}{P(x_1,\ldots,x_m)} \\ &= \arg\max_{k} P(k)\prod_{i=1}^{m} P(x_i \mid k).\end{aligned}$$

Figure 2.1: Naive Bayes Formula
[6]

Decision Tree Model

The Supervised decision tree method is based on the estimation based model. The sense tagged corpus is used as knowledge base on which the training is to be done. The yes-no form of rules are used to classify rules in this method. These rules are then used to recursively analyse training data. The main characteristic of this method is that all the internal nodes are used to represent the features while edges are representing feature values and all leaf nodes are used to represent the important senses. While testing, the ambiguous word with corresponding feature vector is traversed from root to leaf node. Then the reached leaf node sense is considered to be correct sense of the ambiguous word.

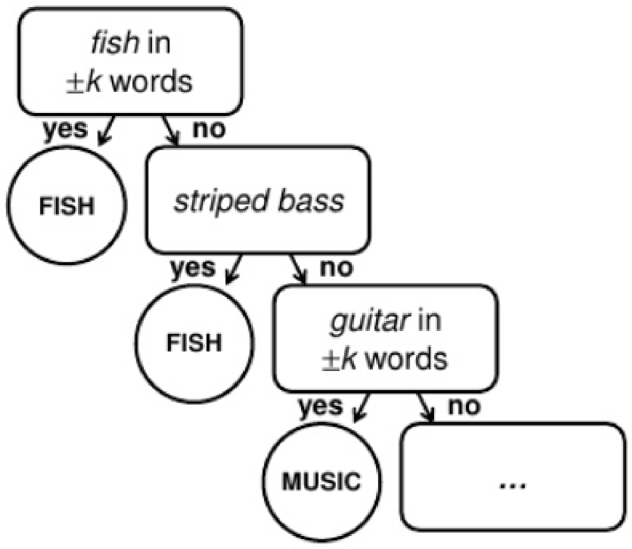


Figure 2.2: Decision Tree Example

2.1.3 Semi-supervised Approach

Many word sense disambiguation algorithms use semi-supervised learning which allows both labelled and unlabelled data because of the lack of training data. The bootstrapping method starts from a small amount of seed data for each word: either a small number of sure fire decision rules (e.g., 'play' in the context of 'bass' almost always states the musical instrument) or manually tagged training corpus. Using any of the supervised method, seeds are used to train an initial classifier. This classifier is then used on the untagged portion of the corpus to extract a larger training set, in which only the most assured classifications are included. This procedure repeats, each new classifier being trained on a successively larger training data, until the complete data is consumed, or until a given maximum number of iterations are reached.

Yarowsky Bootstrapping Method

One of the most effective uses of the bootstrapping approach in Natural Language Processing (NLP) is made by the Yarowsky in 1995. The Yarowsky method is incremental and one of simple iterative algorithm which does not requires large training sets and depends only on relatively small number of instances of each sense. As semi-supervised method uses labelled instances, these labelled instances are then used as raw information to train the classifier initially using other supervised methods. The trained initial classifiers are then used to extract a larger training set from the remaining untagged corpus. The trained sets which are obtained above the particular threshold are kept for future to train the other untrained sets of data for next iteration.[6]

Label Propagation Algorithm

This method works by representing labelled and unlabelled examples as vertices in a connected graph, then circulating the label information from any vertex to nearby vertices through weighted edges iteratively, finally inferring the labels of unlabelled examples after the propagation process converges. In this particular algorithm, label information of any vertex in a graph is propagated to nearby vertices through weighted edges until a global stable stage is achieved. Larger edge weights allow labels to travel through easier.

2.1.4 Unsupervised approach

Unsupervised learning methods are the greatest challenge for WSD researchers. The underlying assumption is that similar meaning words occur in similar contexts, and thus senses can be induced from text by clustering word occurrences using some measure of similarity of context, a task referred to as word sense discrimination or induction. To disambiguate a word they use some measure of similarity in context to get the correct sense.

Performance has been observed to be lower than for the other methods described above, but it is hoped that unsupervised learning will overcome the knowledge acquisition bottleneck because they are not dependent on manual effort.

Co-occurrence Graphs/ Hyperlex Algorithm

Where the previous techniques use vectors to represent the words, the algorithms in this domain make use of graphs. Every word in the input text becomes a vertex and syntactic relations become edges between respective vertices. The context units (e.g. paragraph, sentence) in which the target words occur, are used to create the graphs. The edge weights are inversely proportional to the frequency of co-occurrence of these target words.

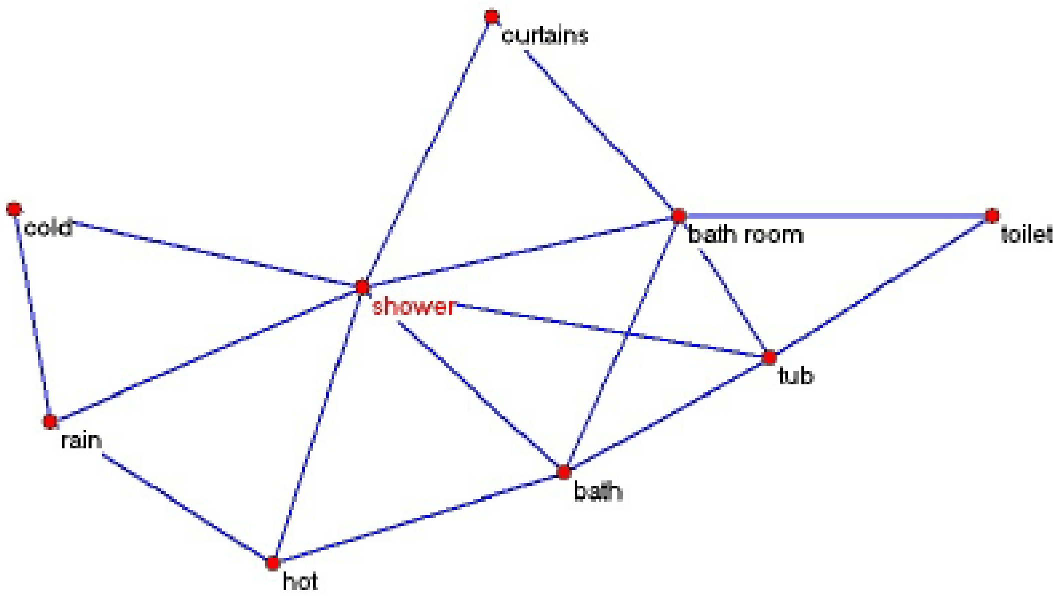


Figure 2.3: Co-occurrence Graph Example

WSD using Parallel Corpora

It was experimentally found out that, words in one particular language, which have ambiguous meanings, have distinct translations in some other language. This assumption is utilized by Ide (2002) in an algorithm for disambiguation. The algorithm was designed with the objective of obtaining large sense marked corpus automatically marked with high efficiency. For this purpose, the algorithm needs raw data set from more than one language (hence the name parallel corpora).

2.2 Literature Survey of Previous Research Work

2018 Myung Yun Kang, Tae Hong Min, Jae Sung Lee [1]

In this paper, they have attempted to extend the word vector space model to reflect a more fine-grained meaning in context vectors by incorporating embedded senses. They have used a large Korean sense-tagged corpus and built an embedded sense space with supervised learning and evaluated the effectiveness of the sense embedding for word sense disambiguation.

The results of their experiment with a Korean sense-tagged corpus showed that the proposed method, i.e., embedded sense space model, is more effective than the word space model. Embedded sense space model is not useful because sense context or disambiguated word context is not available in a normal query.

2017 Pratibha Rani, Vikram Pudi, Dipti Misra Sharma[2]

In this paper, the authors have presented a generic Word Sense Disambiguation (WSD) method using semi-supervised approach. They explain that current WSD systems use extensive domain resources and require advanced linguistic knowledge. Therefore, to improve these factors, they propose a system that extracts context based list from a small amount of seed data containing sense tagged and untagged training data. Their experiments in Hindi and Marathi language domains show that the system gives good performance without language specific information with exception of sense IDs present in the training set, with approximately 60-70% precision.

2016 Ignacio Iacobacci, Mohammad Taher Pilehvar, Roberto Navigli[3]

The main focus of this paper is on word embedding. i.e. is to collect the semantic information from the collection of the datasets. It is an example of knowledge-based approach. Word embedding is usually a collection of names for a set of language modelling and advanced learning techniques in the natural language processing.
In this the results are evaluated by using two methods:
1. Lexical Sample WSD Experiments.
2. All words WSD Experiments.
The main interests were on the training parameters of embedding and WSD features which were impacting on the WSD performance. The maximum accuracy observed during this experiment was 69.9%.

2015 Udaya Raj Dhungana, Subarna Shakya, Kabita Baral and Bharat Sharma[4]

In this paper, they used the knowledge based approach. They have used adapted Lesk algorithm to disambiguate the polysemy word in Nepali language. They grouped each sense of a polysemy word based on the verb, noun, adverb and adjective with which the sense of the polysemy word can be used in a sentence. The experiment is performed on 348 words (including the different senses of 59 polysemy words and context words) with the test data containing 201 Nepali sentences shows the accuracy of their system to be 88.05%.

2013 Alok Pal, Anupam Munshi and Diganta Saha[5]

The key focus of this paper is to speed-up the process of Word Sense Disambiguation by using filtering method which finds appropriate senses of the given ambiguous word through part-of-speech tagging.

The exact part-of-speech of the ambiguous word at that particular instance is obtained. In the next method, online dictionaries are referred such as WordNet etc. which are related to the part-of-speech to disambiguate the correct sense of that particular ambiguous word.

In the training data phase, brown corpus is used for part of speech tagging and WordNet as an online dictionary.

In this method to speed up the process of WSD, some of the relevant glosses (words) are filtered out and accuracy is increased.

2013 Lokesh Nandanwar, Kalyani Mamulkar[6]

This is a survey paper which tells about three approaches used in word sense disambiguation:
1. Supervised Approach.
2. Semi-Supervised Approach.
3. Unsupervised Approach.

These approaches are found to be very useful and successful in the field of word sense disambiguation. They are categorized based on the main source of knowledge which is used to differentiate senses and amount of annotated corpora required.

For the following approaches described above semi-supervised approach requires less amount of annotated corpora as compared to supervised approach. Here, annotated means adding some opinion to the text and corpora means data which is required. By observing and testing the approaches, supervised approach gives better performance as compared to the other approaches.

2009 Niladri Chatterjee and Rohit Misra[7]

In this paper, the team members have presented a trainable model for Word Sense Disambiguation (WSD). The model uses concepts of information theory to find appropriate sense of a word when the context of word is provided. Given training text, the model learns to classify each occurance of target word to correct sense.

The model presented uses the Principle of Maximum Entropy, considering the sense of target word as a random variable with various outcomes. The model then estimates the probability of each sense by measuring the 'bias' of surrounding words. The sense with highest probability is chosen as correct sense of the target word. The model has been proved to deliver accuracy as high as 85%.

2009 Bartosz Broda, Maciej Piasecki[8]

In this paper, word sense ambiguity is resolved by using semi-supervised approach and results have showed that the approach is very close in its precision to the supervised approach.

Drawbacks of using supervised and unsupervised approach is that in the supervised approach it requires laborious and costly manual preparation of the training data. On the other hand, unsupervised approach express significantly lower accuracy and the results are not satisfied for solving the problem.

The main task of this model is to reduce the human involvement, but assign the senses manually using lexical semantic resource known as WordNet.

Here, Lexicographer (LexCSD) is used is gather the corpus from the given keyword. This keyword is then split into clusters and some common keywords are found from the given word. It is analysed to search for the common characteristics or senses in each cluster. Evaluation is done later by cross-checking the MRDs.

1986 Michael Lesk[9]

The paper written by Michael Lesk in 1986 has been proved to be revolutionary work in Word Sense Disambiguation (WSD). In this paper, he presented his famous Lesk algorithm which has been the pivotal algorithm for knowledge based approach WSD. The Lesk algorithm uses various machine readable dictionaries (MRDs) to find correct senses of words. The algorithm searches for overlaps in various senses or signatures of a word. Senses having maximum overlap are chosen as the correct senses of the word. Lesk has concluded that the algorithm produces an accuracy of about 50-70% depending on the MRD used.

2.3 Semi-Supervised WSD with Neural Models[14]

Dayu Yuan, Julian Richardson, Ryan Doherty, Colin Evans, Eric Altendorf

In this paper, the research team have used Supervised and Semi-supervised approaches to Word Sense Disambiguation (WSD) using Neural Nets and label propagation method respectively. They compare and contrast the 2 different ways of obtaining word embeddings including using Word2Vec model and a recurrent neural network model using Long Short-Term Memory (LSTM) architecture.

This LSTM Recurrent Neural Network shows extremely high accuracy results when tested in SemEval International Workshop Tasks with training done on both SemCor[11] and OMSTI [12]

They also implement Semi-supervised WSD using label propagation method. This method works by representing labelled and unlabelled examples as vertices in a connected graph.The label information is then circulated from any vertex to nearby vertices through weighted edges iteratively, finally inferring the labels of unlabelled examples after the propagation process converges.

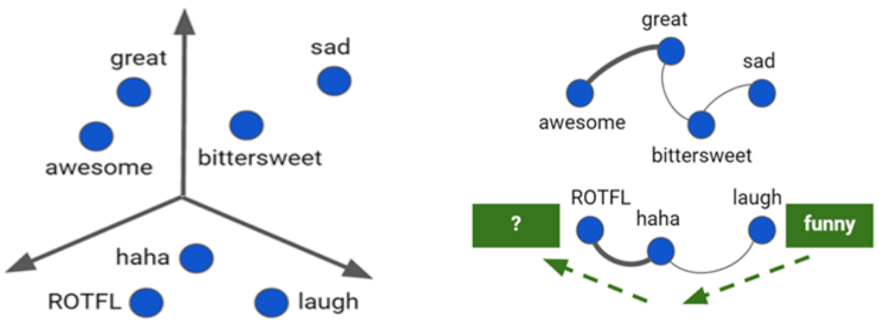


Figure 2.4: Label Propagation

We have chosen this paper as our base paper since its accuracy results are consistently higher than any previous research models. We plan to use the Long Short-Term Memory (LTSM) Architecture as the team have used to build Word Embeddings. But instead of testing data based on these word embeddings, we plan to use them to create feature vectors for our neural network. We hypothesize that using LTSM input vectors along with suitable cost reduction functions for our neural network will produce an increase in accuracy. Using principal component analysis (PCA), feature reduction can also be done to help visualize the data for better understanding.

2.4 Summary of Literature Survey

Table 2.1: Literature Survey Summary

Year	Author	Title of Paper	Method	Advantages	Limitations
2018	Myung Yun Kang, Tae Hong Min, Jae Sung Lee	Sense Space for Word Sense Disambiguation	Word Space Model	Sense Space Model more effective than Word space model	Not practical because sense context is not available in normal queries.
2017	Pratibha Rani, Vikram Pudi, Dipti M. Sharma	Semi-supervised Data-Driven Word Sense Disambiguation for Resource-poor Languages	Contextual similarity property based on hypothesis of Yarowsky (1993)	Generic Method is suitable for resource-poor languages and it can be used for various languages without requiring a large sense tagged corpus.	Provides comparatively low accuracy for English language as compared to other systems.

Continued on next page

Table 2.1 – continued from previous page

Year	Author	Title of Paper	Method	Advantages	Limitations
2016	Ignacio Iacobacci, Mohammad Taher Pilehvar, Roberto Navigli	Embedding for Word Sense Disambiguation: An Evaluation Study	Knowledge-based word embedding	Word Embedding can be used to improve state-of-the-art Supervised WSD.	Best performance is obtained when standard WSD features are augmented with the additional knowledge from Word2vec vectors.
2015	Udaya Raj Dhungana, Subarna Shakya, Kabita Baral and Bharat Sharma	Word Sense Disambiguation using WSD Specific WordNet of Polysemy Words	Adapted Lesk algorithm	Accuracy is increased by 3.484% in compared to the accuracy of previous systems of Nepali Language.	Requires large amount of training data.

Continued on next page

Table 2.1 – continued from previous page

Year	Author	Title of Paper	Method	Advantages	Limitations
2013	Alok Pal, Anupam Munshi and Diganta Saha	An Approach To Speed-up the Word Sense Disambiguation Procedure Through Sense Filtering	Filtering Method using unsupervised method	Correct sense of an ambiguous word is found using the Part-of-Speech Tagging before the disambiguation procedure.	Execution time might be increased
2013	Lokesh Nandanwar and Kalyani Mamulkar	Supervised, Semi supervised, Unsupervised WSD Approaches	Survey Paper which summarizes approaches regarding WSD	Provides useful information regarding basics of WSD	Does not go into details about any of the approaches it mentions
2009	Niladri Chatterjee and Rohit Misra	Word-Sense Disambiguation using Maximum Entropy Model	Principle of Maximum Entropy	High accuracy of around 85% through the language independent model.	Same concept applied to Supervised methods might reward higher accuracy.

Continued on next page

Table 2.1 – continued from previous page

Year	Author	Title of Paper	Method	Advantages	Limitations
2009	Bartosz Broda, Maciej Piasecki	Semi-Supervised Word Sense Disambiguation Based on Weakly Controlled Sense Induction	LexCSD algorithm ,Clustering algorithm, NB algorithm	Reduced human involvement	Limited to unsupervised methods which has a lower accuracy and produce results which are not satisfying for many applications
1986	Michael Lesk	Automatic Sense Disambiguation using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone	Supervised Method, Lesk Algorithm	Laid the foundation of Supervised Learning, providing 50-70% accuracy	Accuracy is limited to technology of lexical resources.

Chapter 3

DESIGN

3.1 Introduction

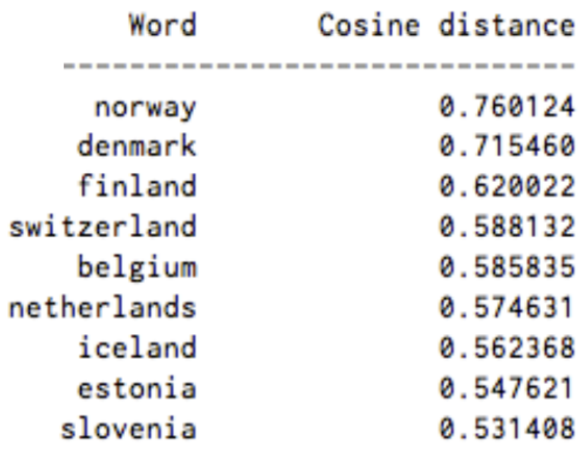
Various solutions to word sense ambiguity have been put forward. Most of these systems use one of the four approaches mentioned earlier. Out of these approaches, supervised approach to WSD has been proven to produce maximum accuracy. Therefore, in our proposed model we will be making use of supervised approach as well.

As mentioned earlier, Artificial Neural Networks mimic the functioning of a real human brain. Given the context words in which an ambiguous word occurs, the neural net should be able to successfully predict the correct sense of the ambiguous word. For creating a accurate neural net classifier we also require large amounts of labelled data as such a model falls under the supervised approach to WSD. We will be using SemCor[11] and OMSTI[12] labelled data sets for this purpose. Once trained, we can judge the accuracy of the Neural Net by using the test data set. The classifier should also be able to return the correct sense of an ambiguous word based on input given by user.

3.2 Word Embeddings

The input feature vector of an ambiguous word for the neural network will be created using its word embeddings. The vectors we use to represent words are called neural word embeddings. Word Embeddings are created using the words similar and most commonly used context words. They can be created using various methods such as Word2Vec, Recurrent RNN models such as LSTM, etc. Word Embeddings measure cosine similarity,i.e. no similarity is expressed as a 0, while total similarity is expressed as 1.

Example:
Figure 3.1 shows word embeddings for the word 'Sweden'. Since Norway and other Scandinavian countries are closely related to Sweden, their cosine values are closest to 1.



```
       Word      Cosine distance
----------------------------------
     norway             0.760124
    denmark             0.715460
    finland             0.620022
switzerland             0.588132
    belgium             0.585835
netherlands             0.574631
    iceland             0.562368
    estonia             0.547621
   slovenia             0.531408
```

Figure 3.1: Word Embeddings for the word 'Sweden'

3.3 Feature Vectors

The input feature vectors to the Neural Network will be created using both the input of dataset and the word embeddings of the ambiguous word. If embedding words are present in the input context then they will be represented in the feature vector with value '1', else with a '0'. After every word in input context is checked for its presence in word embeddings, the resultant feature vector will be passed to the Neural Net input layer. Therefore number of nodes in the input layer of Neural Net will be equal to the number of word embeddings we use.

Example:

Assume that the word embeddings for the word 'crown' are:

[gold jewels king teeth drill dentist]

and that input is the sentence:

"The dentist did a really good job putting the crown on my teeth"

Then the Input Feature vector will be:

[0 0 0 1 0 1]

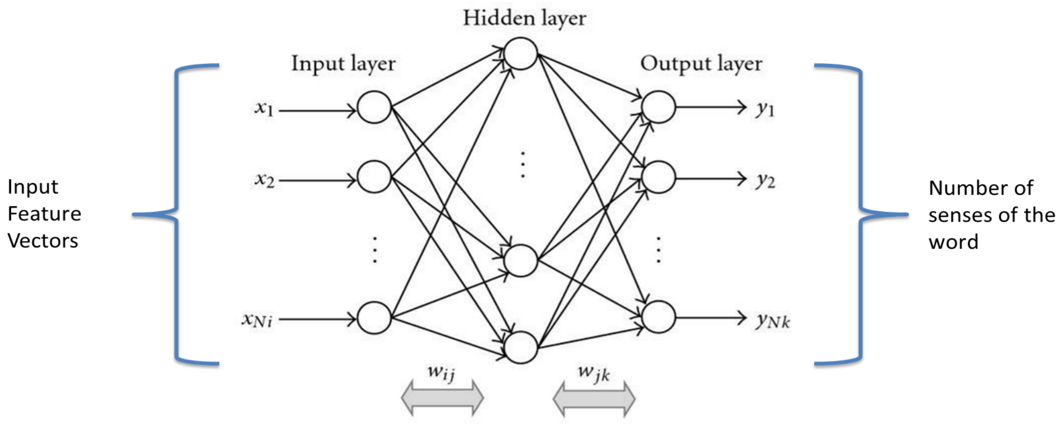


Figure 3.2: Neural Network for an Ambiguous Word

3.4 Feed Forward Neural Networks

The following figure shows an example of a forward propagation step in a Feed Forward Neural Network. In vectorised implementations, input sets are represented in columns of a matrix which are multiplied by a weight matrix theta that represents each layer of a neural network. The use of matrices reduces time and increases efficiency for calculations as it does not require loops in the program structure to multiply each element individually.

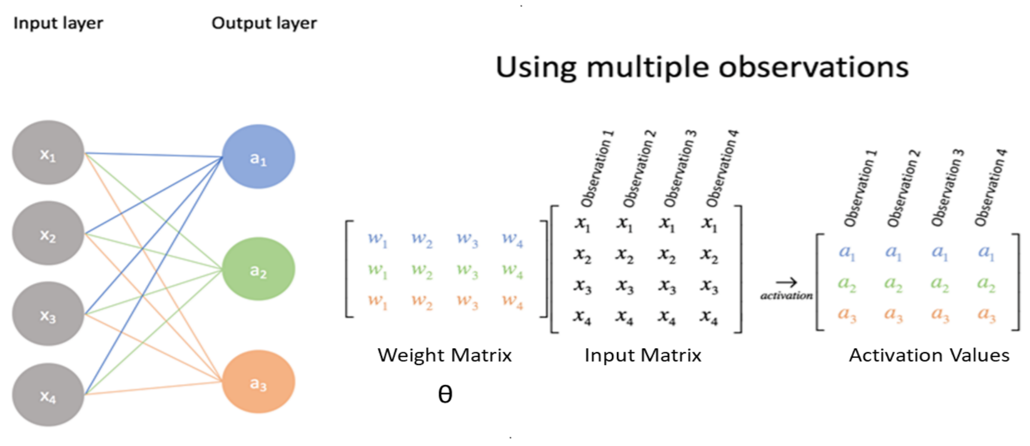


Figure 3.3: One step of forward propagation

3.5 Cost Calculations of Neural Network

Calculating costs is the one definitive way of understanding that our Neural Network is working correctly. After every iteration the cost of the neural network is calculated using the cost function given below. Displaying the value of cost every few hundred iterations can help us accurately gauge whether our neural net is actually learning or not.

Cost Function of a Neural Network is given by:

$$h_\Theta(x) \in \mathbb{R}^K \qquad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Where:

- m = number of features
- K = number of output layer nodes
- L = number of layers in network
- θ represents weight matrix

Figure 3.4: Cost Function

3.6 Output of Neural Network

The output layer of neural network contains nodes equal to the number of different senses for the ambiguous word, according to the WordNet[10] dictionary. The node for which the highest numerical value is calculated among the different output nodes will represent the predicted sense. If the third node of output layer has the highest value, then it means that the system has predicted the third sense, all senses being annotated by WordNet[10].

3.7 Datasets

List of approximately 1000 highly ambiguous words will be created from various dictionary sources such as dillfrog[13]. The words in this list have a minimum of 10 different senses and a maximum of 73. Each ambiguous word will have its own neural net and word embeddings.

Supervised Machine Learning requires several thousand datasets for effectively training and testing models. The more data that is made available to machine learning models, the more accuracy it can achieve. Neural Networks are no different. For the purpose of WSD using neural models, we will require massive amounts of labelled or tagged data which will help us to train and test our model and help it achieve high accuracy.

For such purposes, we will be using 2 different labelled data sources:

3.7.1 OMSTI

OMSTI (One Million Sense Tagged Instances)[12] and SemCor (Semantic Cortex)[11]. OMSTI is a dataset of a million examples of sense tagged sentences that was created for the purposes of supervised WSD.

3.7.2 SemCor

SemCor on the other hand has around a few hundred thousand instances but these datasets are manually tagged and hence more accurate.

All the resources mentioned above are annotated using WordNet[10] 3.0. It is a large lexical database of English words. WordNet is open source and is easily imported and executed in plenty of languages. It's primary use is in automatic text analysis and artificial intelligence applications.

3.8 Data Flow Diagrams(DFD)

3.8.1 Data Flow Diagram Level 0

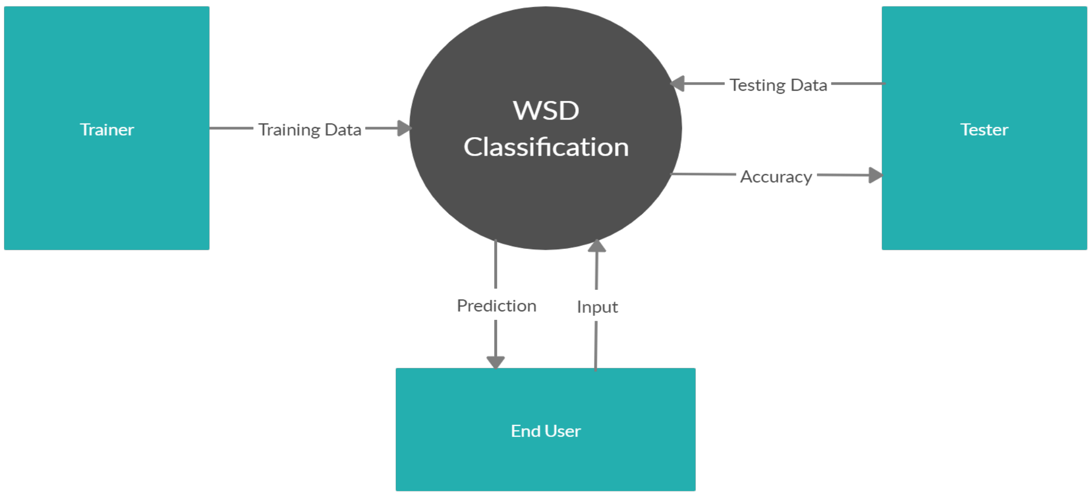


Figure 3.5: DFD Level 0

3.8.2 Data Flow Diagram Level 1

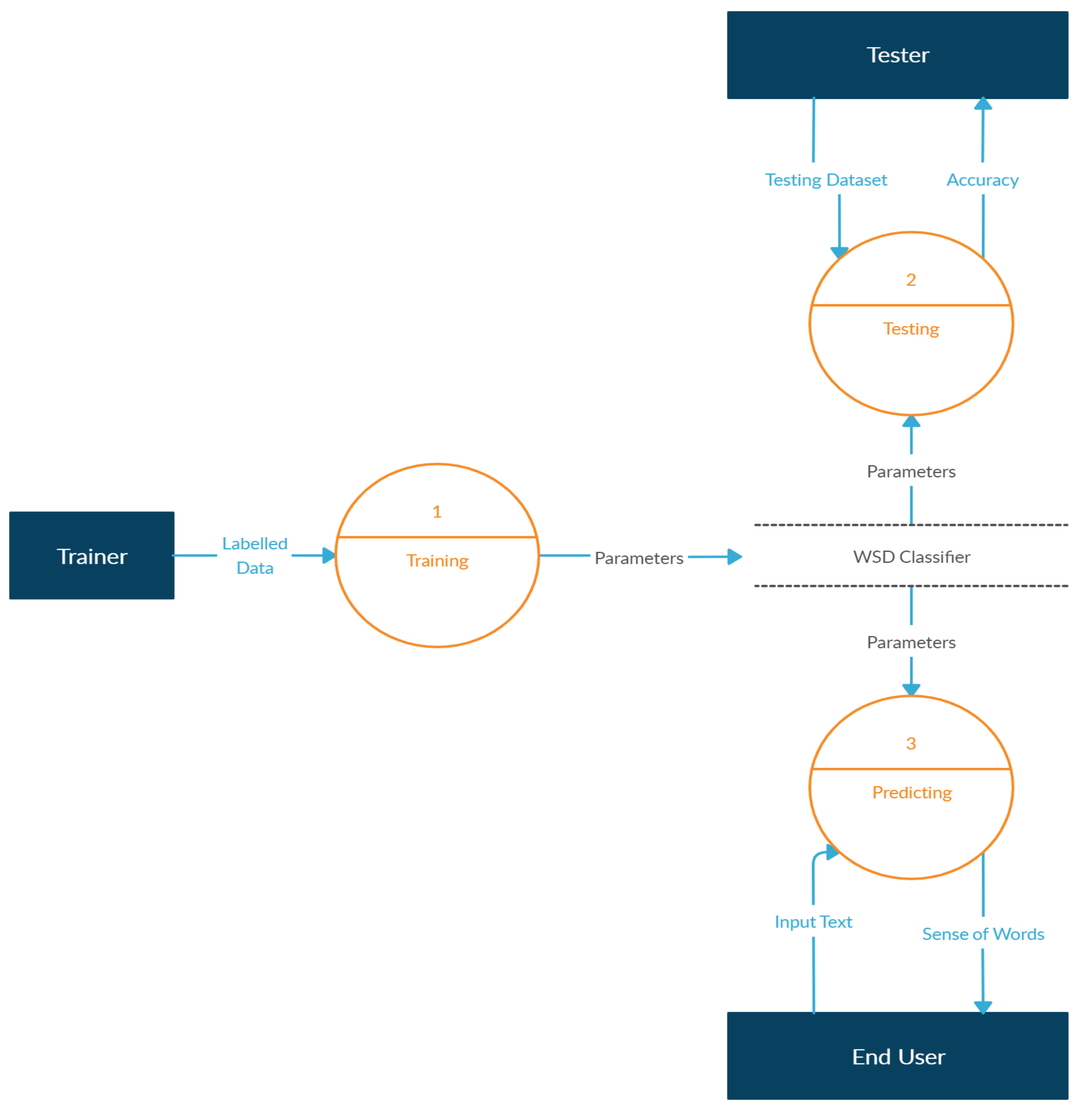


Figure 3.6: DFD Level 1

3.8.3 Data Flow Diagram Level 2

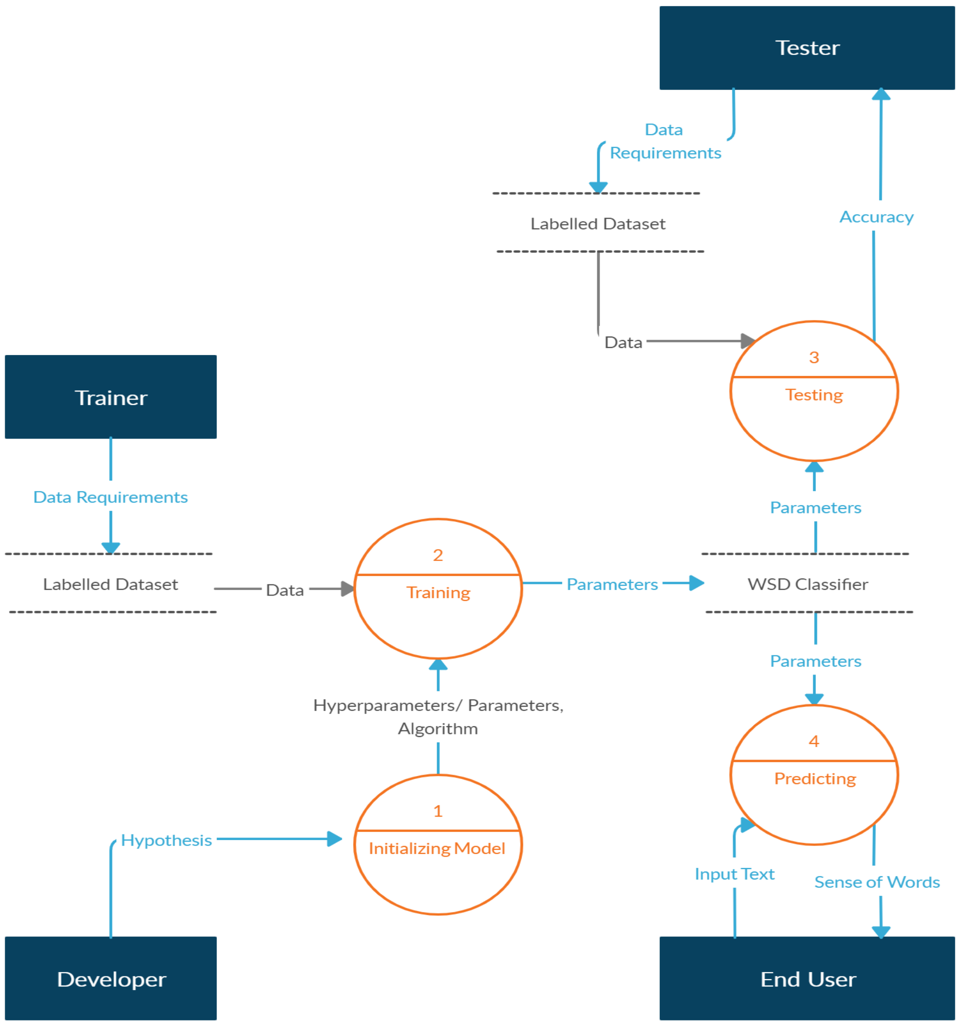


Figure 3.7: DFD level 2

3.9 UML Diagrams

3.9.1 Use Case Diagram

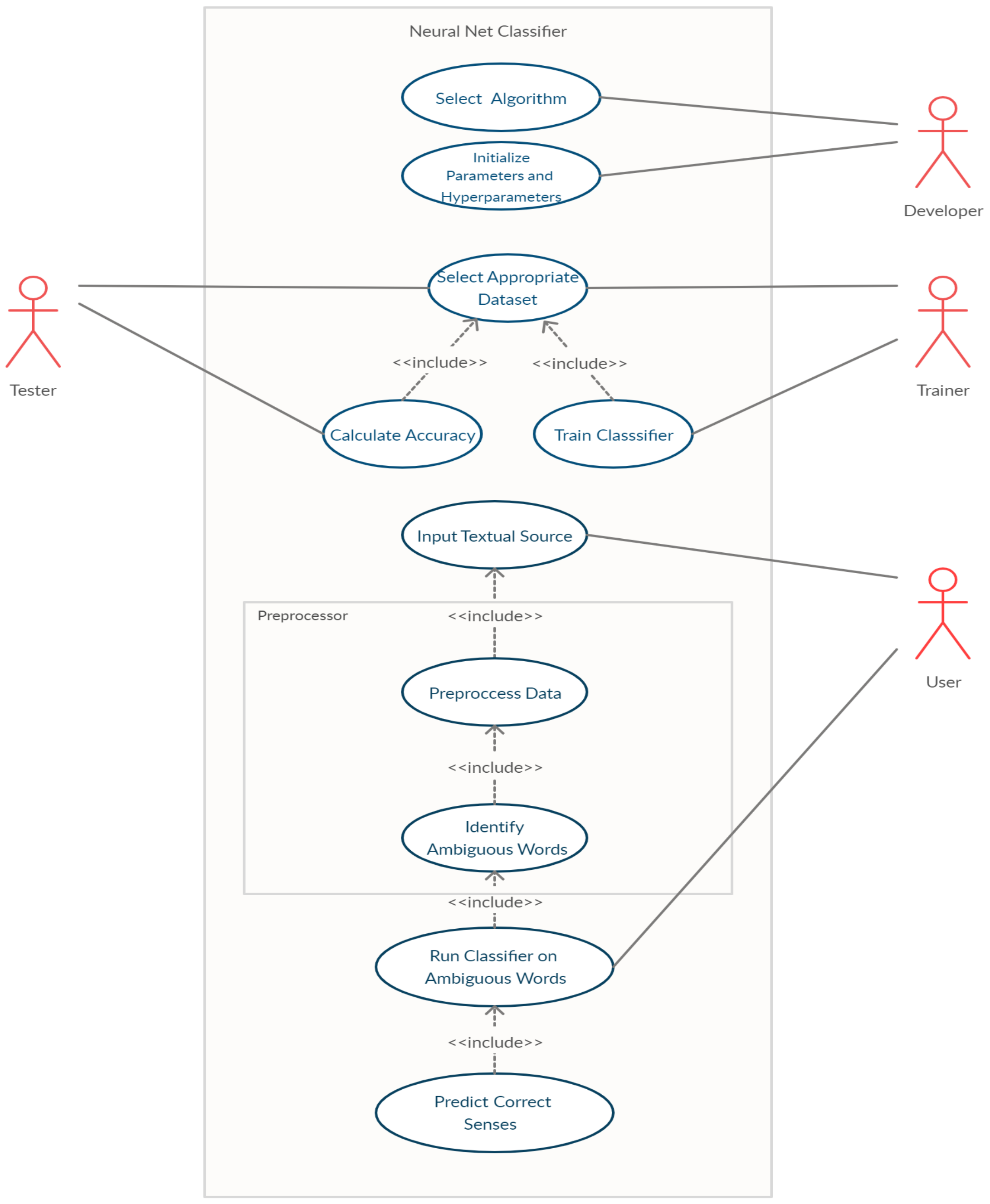
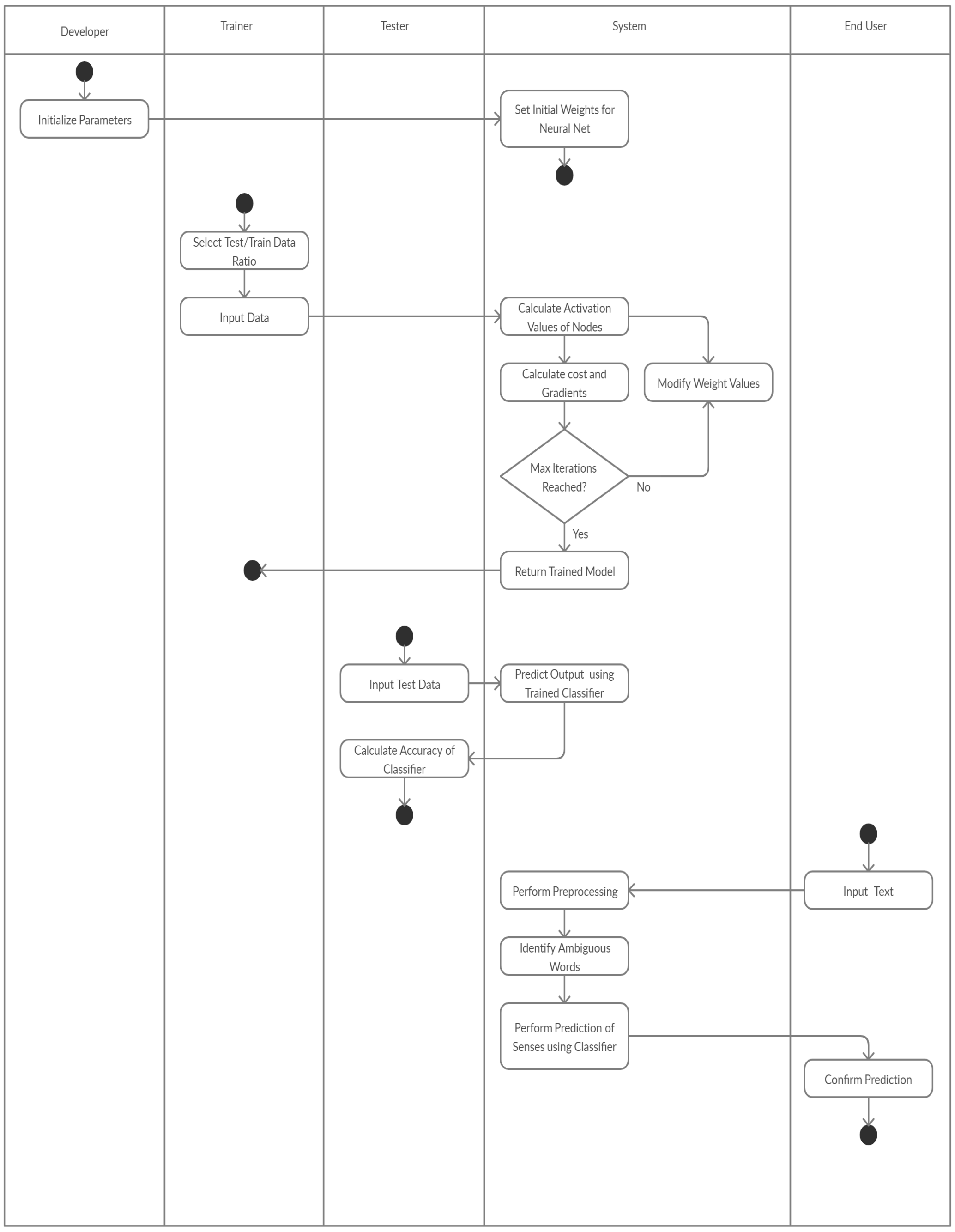


Figure 3.8: Use Case Diagram for WSD Neural Net Classifier

3.9.2 Activity Chart



Developer
Trainer
Tester
System
End User
Initialize Parameters
Set Initial Weights for Neural Net
Select Test/Train Data Ratio
Input Data
Calculate Activation Values of Nodes
Calculate cost and Gradients
Modify Weight Values
Max Iterations Reached?
No
Yes
Return Trained Model
Input Test Data
Predict Output using Trained Classifier
Calculate Accuracy of Classifier
Input Text
Perform Preprocessing
Identify Ambiguous Words
Perform Prediction of Senses using Classifier
Confirm Prediction

Figure 3.9: Activity Chart

3.9.3 State Transition Diagram

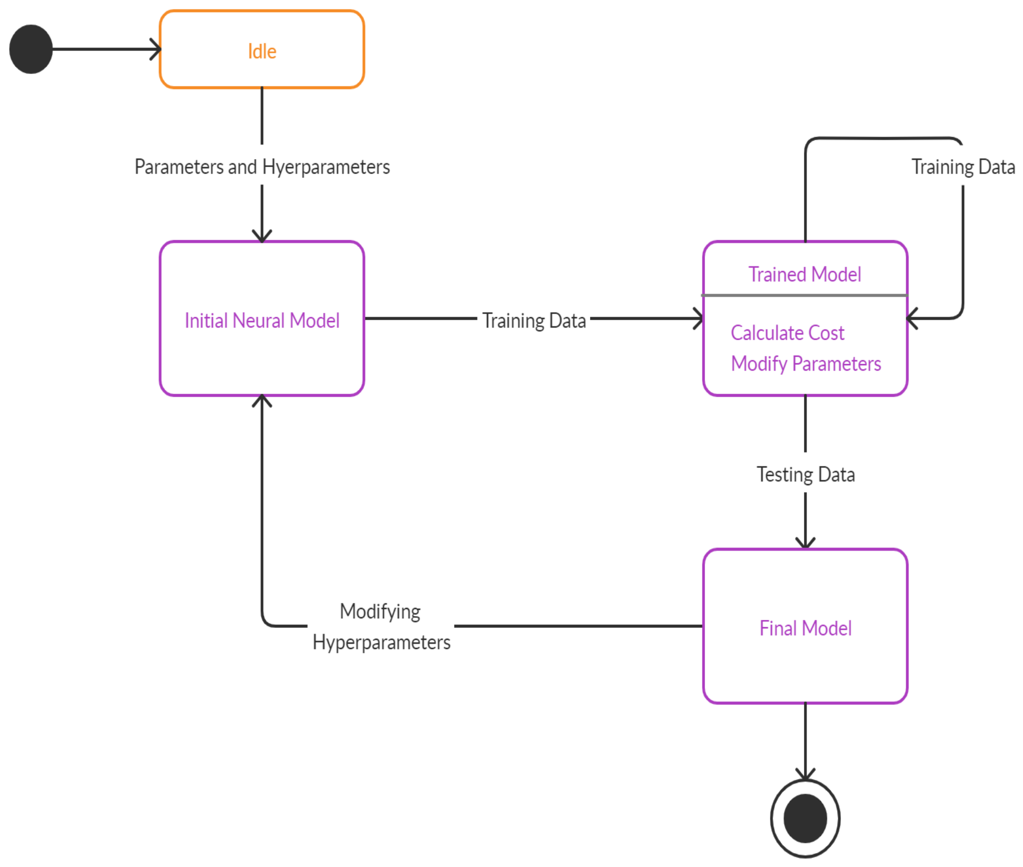
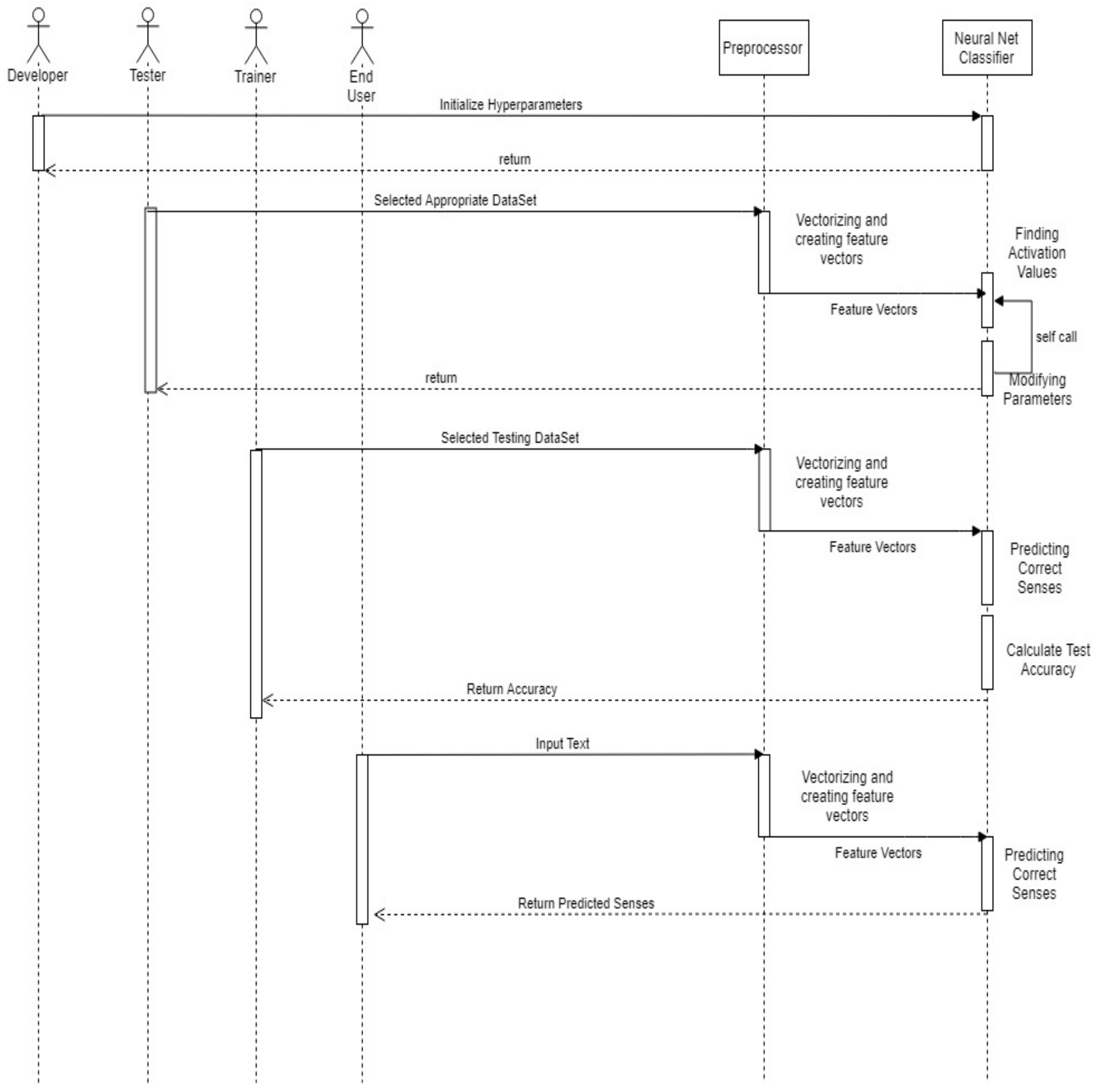


Figure 3.10: State Transition Diagram

3.9.4 Sequence Diagram



Developer
Tester
Trainer
End User
Preprocessor
Neural Net Classifier
Initialize Hyperparameters
return
Selected Appropriate DataSet
Vectorizing and creating feature vectors
Finding Activation Values
Feature Vectors
self call
return
Modifying Parameters
Selected Testing DataSet
Vectorizing and creating feature vectors
Feature Vectors
Predicting Correct Senses
Calculate Test Accuracy
Return Accuracy
Input Text
Vectorizing and creating feature vectors
Feature Vectors
Predicting Correct Senses
Return Predicted Senses

Figure 3.11: Sequence Diagram

3.9.5 Class Diagram

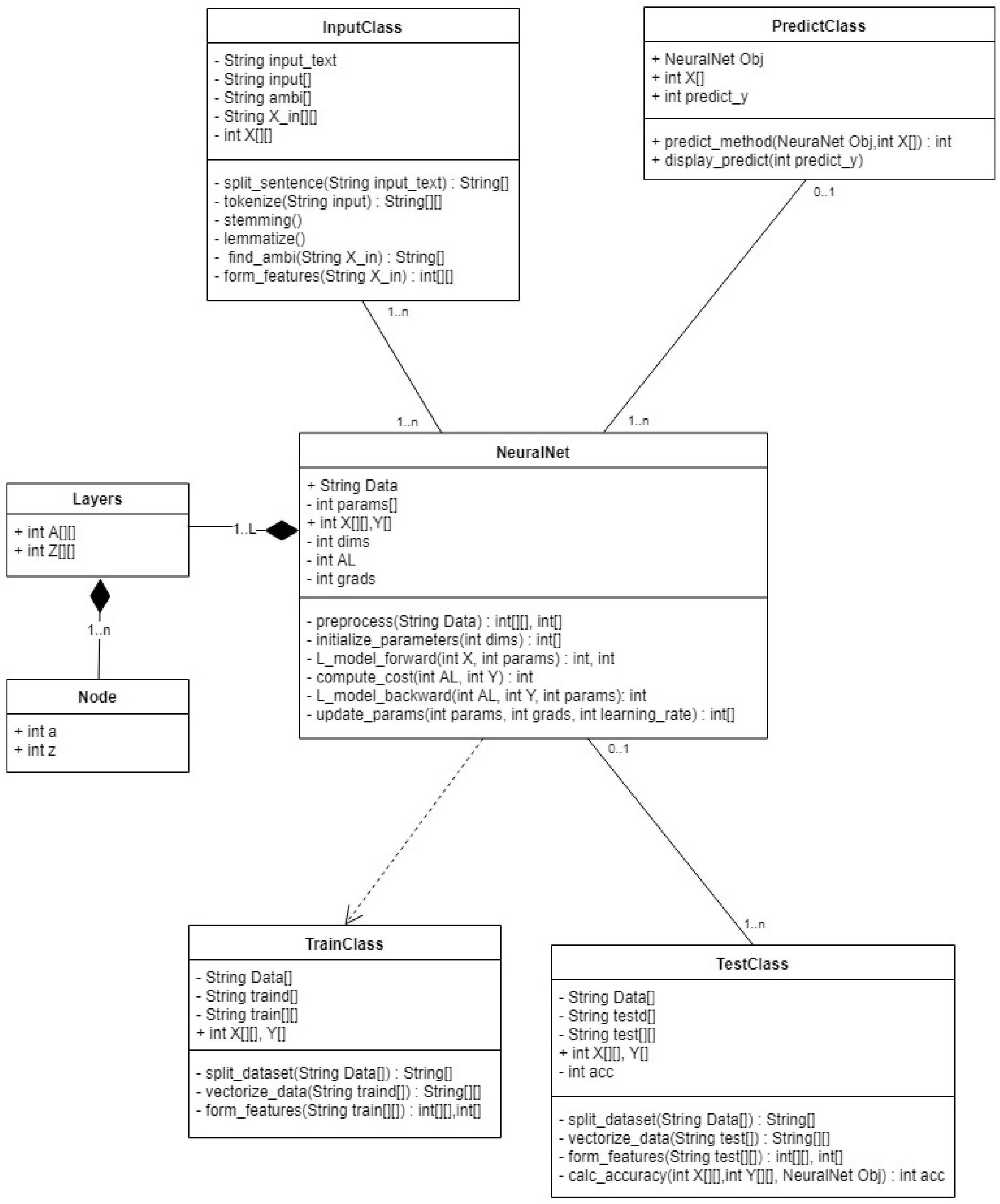

InputClass
- String input_text
- String input[]
- String ambi[]
- String X_in[][]
- int X[][]
- split_sentence(String input_text) : String[]
- tokenize(String input) : String[][]
- stemming()
- lemmatize()
- find_ambi(String X_in) : String[]
- form_features(String X_in) : int[][]
PredictClass
+ NeuralNet Obj
+ int X[]
+ int predict_y
+ predict_method(NeuraNet Obj,int X[]) : int
+ display_predict(int predict_y)
0..1
1..n
1..n
1..n
NeuralNet
+ String Data
- int params[]
+ int X[][],Y[]
- int dims
- int AL
- int grads
- preprocess(String Data) : int[][], int[]
- initialize_parameters(int dims) : int[]
- L_model_forward(int X, int params) : int, int
- compute_cost(int AL, int Y) : int
- L_model_backward(int AL, int Y, int params): int
- update_params(int params, int grads, int learning_rate) : int[]
Layers
+ int A[][]
+ int Z[][]
1..L
1..n
Node
+ int a
+ int z
0..1
1..n
TrainClass
- String Data[]
- String traind[]
- String train[][]
+ int X[][], Y[]
- split_dataset(String Data[]) : String[]
- vectorize_data(String traind[]) : String[][]
- form_features(String train[][]) : int[][],int[]
TestClass
- String Data[]
- String testd[]
- String test[][]
+ int X[][], Y[]
- int acc
- split_dataset(String Data[]) : String[]
- vectorize_data(String test[]) : String[][]
- form_features(String test[][]) : int[][], int[]
- calc_accuracy(int X[][],int Y[][], NeuralNet Obj) : int acc

Figure 3.12: Class Diagram

3.10 System Architecture

3.10.1 Classification Architecture

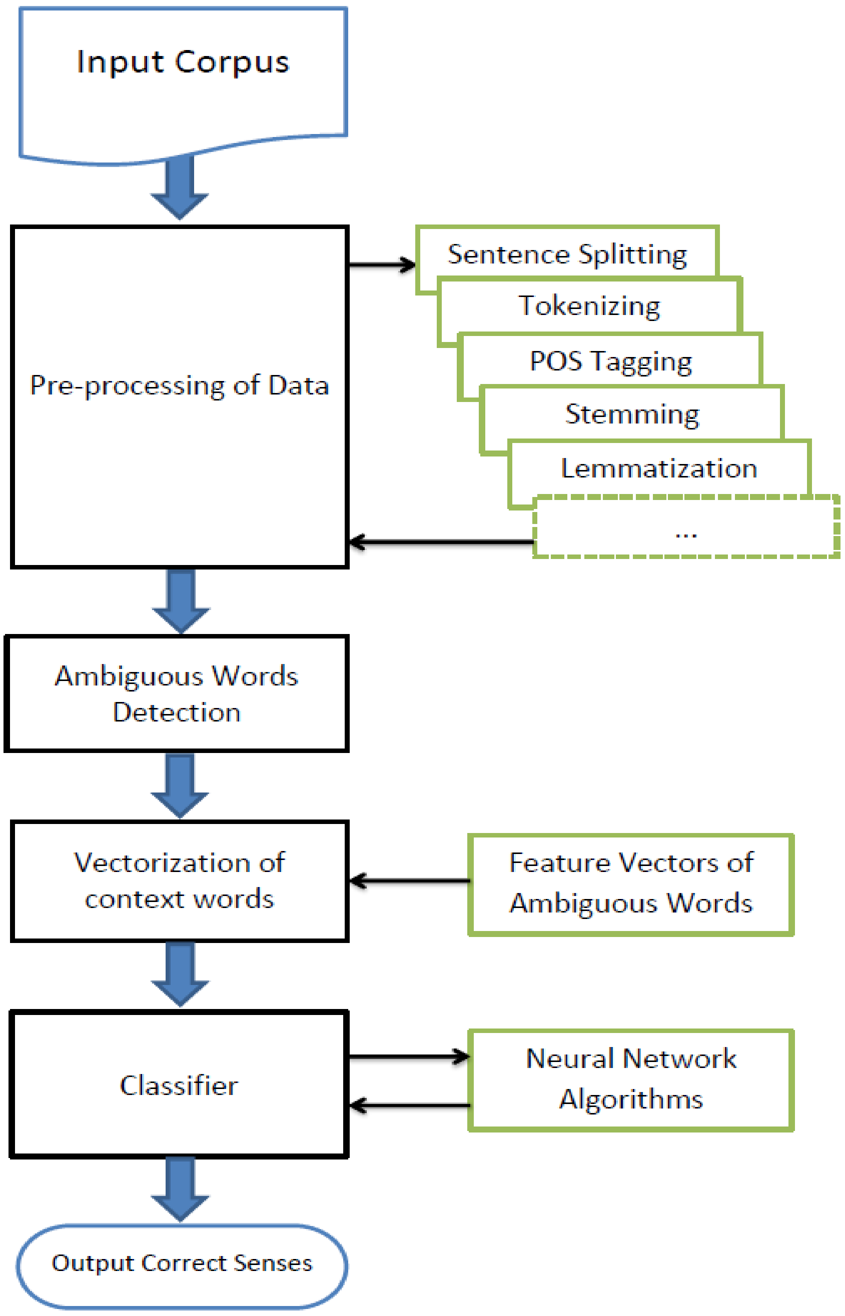
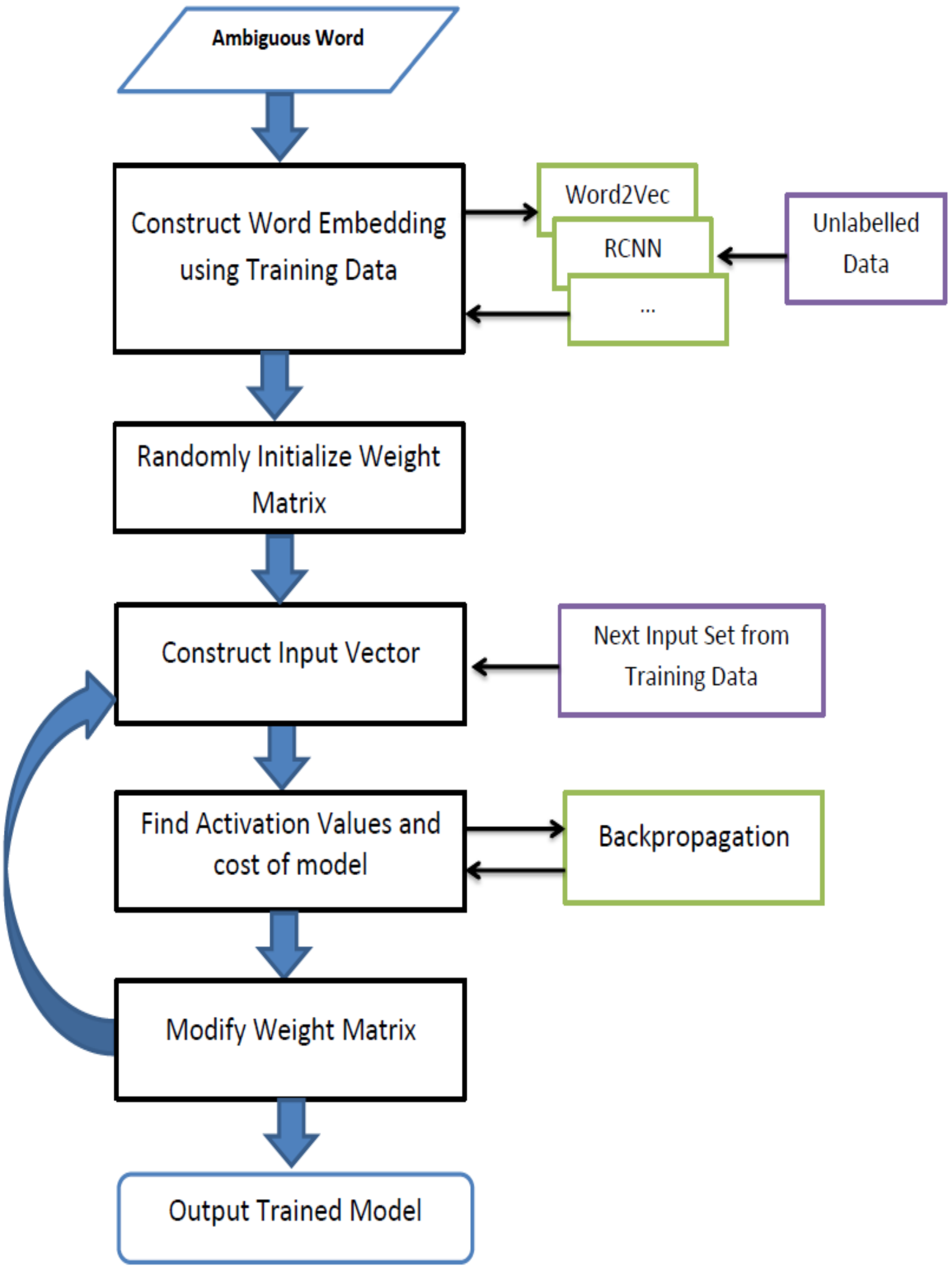


Figure 3.13: Classification Architecture

3.10.2 Training Architecture



Ambiguous Word
Construct Word Embedding using Training Data
Word2Vec
RCNN
...
Unlabelled Data
Randomly Initialize Weight Matrix
Construct Input Vector
Next Input Set from Training Data
Find Activation Values and cost of model
Backpropagation
Modify Weight Matrix
Output Trained Model

Figure 3.14: Training Architecture

Chapter 4

IMPLEMENTATION

4.1 Introduction

The platform chosen for the project execution was Python. Python is an interpreted, high-level, general-purpose programming language, which provides extensive library support for data mining, data processing as well as deep learning and artificial intelligence. The libraries necessary for our project were easily made available in Python using import packages options.

Implementation of the project was done using Anaconda Navigator, a graphical user interface providing various different source code editors for coding in Python 3.7. The GUI of the project was made using PyQt5, a Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in, as well as Qt Designer platform for drag-drop functionality.

4.2 Libraries used in Python

Various libraries had to be imported for successful implementation on our python platform:-

4.2.1 Numpy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The import of Numpy allows us to store various data in appropriate data structures so as to train classifiers on the data easily with hassle-free conversions.

4.2.2 Nltk

The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. NLTK gives us access to important datasets that are easily available in Python for manipulation and processing. S

4.2.3 WordNet

Part of NLTK corpii, WordNet is a lexical database for the English language, which was created by Princeton University. You can use WordNet alongside the NLTK module to find the meanings of words, synonyms, antonyms, and more. Let's cover some examples, which is a crucial part of our project. All the data in OMSTI and SemCor is tagged using sense numbers from WordNet. The Senses of words in WordNet do not change respective positions so if the output of the Neural Net predicts the second sense, the definition of the sense does not vary, which is what makes it a very helpful to our cause.

4.2.4 Tensorflow

Tensorflow is a free and open source library for dataflow and differential programming fora range of task. For this project we will be using tensorflow 2.1.0 with GPU support. The CUDA toolkit will also be used which would help accelerate the training process

4.2.5 Keras

Keras is an High level API built over tensorflow. We will be using keras with backend support as tensorflow. The CUDNN package is also installed which can help keras model run over GPU. Keras libraries will help create and fit neural models to the data. The model will then be written to file and can be read and predictions can be made based on this model in future.

4.3 Softwares and IDEs

Anaconda Navigator

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution includes data-science packages suitable for Windows, Linux, and macOS. Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

Spyder 4

Spyder is an open source cross-platform integrated development environment (IDE) for scientific programming in the Python language. Spyder integrates with a number of prominent packages in the scientific Python stack, including NumPy, SciPy, Matplotlib, pandas, IPython, SymPy and Cython, as well as other open source software.

Google Colab

Colaboratory (also known as Colab) is a free Jupyter notebook environment that runs in the cloud and stores its notebooks on Google Drive. Colab was originally an internal Google project; an attempt was made to open source all the code and work more directly upstream, leading to the development of the "Open in Colab" Google Chrome extension, but this eventually ended, and Colab development continued internally. As of October 2019, the Colaboratory UI only allows you to create notebooks with Python 2 and Python 3 kernels; however, if you have an existing notebook whose kernelspec is IR or Swift, that will work, since both R and Swift are installed in the container. It allows us to use the performance of top of the line GPUs at no extra cost.

QtDesigner

Qt Creator is a cross-platform C++, JavaScript and QML integrated development environment which simplifies GUI application development. It is part of the SDK for the Qt GUI application development framework and uses the Qt API, which encapsulates host OS GUI function calls. It includes a visual debugger and an integrated WYSIWYG GUI layout and forms designer. The Designer allows us create a GUI for the project.

4.4 Modules

4.4.1 OMSTIimport.py

This is the main module that is used for importing Dataset from xml files existing in local drives. The data to be used for training comprises 2 main files: Semcor.xml and OMSTI+SemCor.xml. The program gives user the option to select searching for sentences based on any one of the 2 corpii. The program finds all occurrences of given word in the datasets and create 2 separate data structures: One for storing such occurring sentences and one for storing the sense keys of the words in those sentences. Both structures are written to file for use in other models.

4.4.2 key2sens.py

This module of the project is used to read the sense keys stored to file in the previous module, and convert each individual sense keys to the appropriate synset according to wordnet.

4.4.3 makeembeddingfeauture.py

This module is used to create word embeddings for the ambiguous word given by the user. The program then uses these word embeddings to create feature vectors from the sentences extracted from the datasets in module 1. The result is feature vectors with size of number of sentences by the number of words in word embeddings of that word. The words in word embeddings are cross referenced with the words in sentence and in case of absence or presence a 0 or 1 is added to feature vectors.

4.4.4 kerasNeuralNet.py

This is the main module of the project, which creates a model using keras library imported in anaconda. The model is then fit to the feature vectors found in the previous module. The model is fit using a categorical cross entropy loss function that is perfect for our system. The training accuracy is returned and gives option to also test the model generated on unseen data. The testing accuracy can also be printed.

4.4.5 preprocessing.py

The preprocessing module is used to process the input given by user. The input text is first stemmed, lemmatized for verbs,adjectives, and nouns and then lastly the stop words are removed from the string input. Another task of this module is to create a list of ambiguous words that are found using the models created previously. If any word is ambiguous that is not in the list, it should first be modelled using kerasNeuralNet module.

4.4.6 userend.py

This module takes input from user and runs functions from the preprocessing module on it. The text is then ran through a feature vector creator to create vectors. The models of the ambiguous words are loaded from the file and keras library functions are used to predict values based on inputs vectors from user. The output prediction is a dictionary containing the ambiguous word, sense number and the definition of that sense.

4.4.7 WSDui.py

This the user end module of the project. It uses functions and libraries of PyQt5 to create a user centric graphical user interface for the project. The GUI consists of 2 pages. The first page takes input from user and passes it to preprocessing modules and displays its output. Once user clicks run button the second page is displayed, upon which the The prediction is done based on the input processed recently. The Result is then displayed in tabular format displaying the senses of all ambiguous words.

4.5 Important Source Code

4.5.1 Xpath Querying for Dataset Processing

```
with open("semcor+omsti.data.xml","r") as f:
file_content = f.read()
tree = etree.XML(file_content)
s_ids=tree.xpath('//sentence[instance/@lemma="'+aword+'"]/@id')

for sid in s_ids:
sentl = tree.xpath('//sentence[@id="'+str(sid)+'"]//@lemma')
datax.append(sentl)
```

```
for i in s_ids:
wx=tree.
xpath('//instance[@lemma="'+aword+'" and ../@id="'+i+'"]/@id')
print("s:"+str(i)+" ; wnsn:"+str(wx))
datay.append(wx)

linno=0

fil = open('semcor+omsti.gold.key.txt', "rt")

for f in fil:
linno=linno+1
for i in range(len(datay)):
if datay[i][0] in f:
s1=f.find(aword)
dataysensekeys.append(f[s1:-1])
fil.close()

with open(aword+"_data_x.txt", "wb") as fp:
pickle.dump(datax, fp)
with open(aword+"_data_sense_keys.txt", "wb") as fp:
pickle.dump(dataysensekeys, fp)
```

4.5.2 Extracting Synsets from Sense Keys

```
def synset_from_sense_key(sense_key):
lemma, ss_type, lex_num, lex_id, head_word, head_id =
re.match(sense_key_regex, sense_key).groups()
lemma=aword
for syn in wn.synsets(lemma):
for syn2 in syn.lemmas():
if syn2.key() in sense_key:
return lemma,syn
s=wn.synsets(lemma)[0]
return lemma,s
```

4.5.3 Creating Word Embeddings for Words

```
def createembeddingmodel(aword):

wembed=[]
syn=wn.synsets(aword)
for s in syn:
sent=s.definition()
sent1=s.examples()
filt=preprocess(str(sent))
filt1=preprocess(str(sent1))
#print(filt,"\n")
wembed=wembed+filt+filt1

with open(aword+"_embedding.txt", "wb") as fp:
pickle.dump(wembed, fp)

return wembed
```

4.5.4 Training Neural Net

```
with open(aword+"Xtr.txt", "rb") as fp:
Xtrain = pickle.load(fp)
with open(aword+"Ytr.txt", "rb") as fp:
Ytrain = pickle.load(fp)

Xtrain=Xtrain.transpose()
Ytrain=Ytrain.transpose()

model = Sequential()
model.add(Dense(150, activation='relu',input_dim=Xtrain.shape[1]))
model.add(Dense(Ytrain.shape[1], activation='sigmoid'))
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])

model.fit(Xtrain, Ytrain, epochs=100, batch_size=32)

with open(aword+"_model.txt", "wb") as fp:
```

```
pickle.dump(model,fp)
return model
```

4.5.5 Testing Neural Net

```
with open(aword+"_model.txt", "rb") as fp:
model=pickle.load(fp)
with open(aword+"Xte.txt", "rb") as fp:
Xtest = pickle.load(fp)
with open(+aword+"Yte.txt", "rb") as fp:
Ytest = pickle.load(fp)
Xtest=Xtest.transpose()
Ytest=Ytest.transpose()

accu=model.evaluate(Xtest,Ytest, batch_size=128)
acc=accu[1]*100
```

4.5.6 Preprocessing Code

```
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(example_sent)
filt = []

for w in word_tokens:
if w not in stop_words and w.isalpha():
filt.append(w)

#print("after tokenization:\n",word_tokens)
#print("after filtration:\n",filt)
filt=[f.lower() for f in filt]
filt=[lemmatizer.lemmatize(f,pos='n') for f in filt]
filt=[lemmatizer.lemmatize(f,pos='v') for f in filt]
filt=[lemmatizer.lemmatize(f,pos='a') for f in filt]

for i in range(len(filt)):
if filt[i].endswith("ily"):
filt[i]=filt[i][:-3]+"y"
```

Chapter 5

RESULTS

5.1 Implementation Status

The entire implementation was completed as per schedule with little to no delays. The most the project deviated from schedule was during the data processing stage. The extreme computing requirement of processing the 1.5 gigabyte dataset of OMSTI was proving extremely hard to work with. The computing environment available to us was simply not sufficient for efficiently finishing work in a timely manner.

The team then made a decision to transfer the data processing module to the Google Colabolatory where we could get the computing power of Google servers with 25 Gigabytes of RAM at our disposal. For this to work we first had to upload the said OMSTI dataset to Google Cloud for working on it.

After we got the module to run, the rest of the project got back on schedule and was completed with ease. As of now, the Neural Network training and testing framework is completed as well as the data processing unit for creation of data inputs to neural net.

5.2 Testing Strategy

The testing of the project, was done in a V model basis. Right after realizing the business requirements of the system, test cases for acceptance testing were being developed.

The modular structure of our core developement meant it allowed us to begin testing at a very early stage. Initial modules were unit tested using singular inputs from the OMSTI dataset. After several modules were developed, their integration with the overall system was tested using component testing through comprehensive testing data extracted from OMSTI and SemCor.

Upon developement of the modules, the system integration testing was started. Various erroneous inputs were given to the models as a way to test rhobustness of the system. Once system was sufficiently tested, the accuracy calculations were begun.

After user interface was designed and implemented, its testing was consequently begun. GUI error messages were added to handle incorrect inputs and testing purposes.

5.3 Accuracy Results

The Neural Network framework created was tested for accuracy calculated using the Keras library. The sense-tagged dataset obtained from OMSTI is divided into training and testing data based on a flexible ratio. The testing was done on a word by word basis, using the keras function, predict(). Giving the testing data as input to the function, it returns accuracy of the model.

For testing purposes, we considered the model of 76 different words that are generally considered to be ambiguous. The average training accuracy of these models came out to be **79.322%**. These results ranged from **70% to 88%** generally dependent on the data input size that was used to train the model. The testing accuracy was initially extremely low i.e. around 20%. But after regularization and adjusting hyperparameters, we were able to raise the testing accuracy upto **67%** on previously unseen data. Based on previous research work conducted in this field, we can say with sufficient evidence that our results are almost upto the mark of industry leading algorithms.

Chapter 6

CONCLUSION

1. Successfully created a Neural Network Framework for the creation of models which maximize accuracy.

2. Achieved accuracy results upto 80% competing with industry standards based on our models.

3. Our WSD Graphical User Interface also provides for end users to try out and test our models by themselves without any coding knowledge. Upon input from user, system can process data, run it through the appropriate model classifier and output the result in an user friendly verbose language rather than Synset Keys which are unknown to most regular casual users.

4. Our model provides a simple yet effective system for Word Sense Disambiguation (WSD) for use in bleeding edge technology.

5. The limited computing power on our PCs might provide a hinderence for local use of system. Models require high computing power to traverse and process dataset for extracting features.

6. For future work, we might consider creating models through different libraries and algorithms that might help in creating and fitting the data more accurately.

7. Future work also might include the use of developed model in end user applications such as search engine optimization, or creating a chatbot.

References

[1] Myung Yun Kang, Tae Hong Min, Jae Sung Lee.
"Sense Space for Word Sense Disambiguation".
2018 IEEE International Conference on Big Data and Smart Computing

[2] Pratibha Rani, Vikram Pudi, Dipti M. Sharma
"Semi-supervised Data-Driven Word Sense Disambiguation for Resource-poor Languages"
2017 14th International Conference on Natural Language Processing (ICON)

[3] Ignacio Iacobacci, Mohammad Taher Pilehvar, Roberto Navigli
"Embedding for Word Sense Disambiguation: An Evaluation Study"
2016 Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 897-907

[4] Udaya Raj Dhungana, Subarna Shakya, Kabita Baral and Bharat Sharma
"Word Sense Disambiguation using WSD Specific WordNet of Polysemy Words"
2015 Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing

[5] Alok Pal, Anupam Munshi and Diganta Saha
"An Approach To Speed-up the Word Sense Disambiguation Procedure Through Sense Filtering"
2013 International Journal of Instrumentation and Control Systems (IJICS) Vol.3, No.4, October 2013

[6] Lokesh Nandanwar and Kalyani Mamulkar
"Supervised, Semi-Supervised and Unsupervised WSD Approaches: An Overview"
2013 International Journal of Science and Research (IJSR)

[7] Niladri Chatterjee and Rohit Misra// Word-Sense Disambiguation using

Maximum Entropy Model// International Conference on Methods and Models in Computer Science, 2009

[8] Bartosz Broda and Maciej Piasecki.
"Semi-Supervised Word Sense Disambiguation Based on Weakly Controlled Sense Induction".
2009 Proceedings of the International Multiconference on Computer Science and Information Technology

[9] Michael Lesk// Automatic Sense Disambiguation using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone// Bell communications Research Morristown,NJ 07960

[10] George A. Miller
WordNet: A Lexical Database for English.
Communications of the ACM Vol. 38, No. 11: 39-41
Bell communications Research Morristown,NJ 07960
https://wordnet.princeton.edu/

[11] George A. Miller
SemCor: Semantically Annotated English Corpus
Princeton University

[12] Kaveh Taghipour and Hwee Tou Ng
One Million Sense-Tagged Instances for Word Sense Disambiguation and Induction
2015 Conference on Computational Language Learning

[13] https://muse.dillfrog.com/
Online Dictionary

[14] Dayu Yuan, Julian Richardson, Ryan Doherty, Colin Evans, Eric Altendorf
2016 Semi-Supervised Word Sense Disambiguation with Neural Models
Google, Mountain View CA, USA