Learning Word Vector Representation in Multi-Task Framework

A Project Report Submitted
in Partial Fulfillment of Requirements
for the Degree of

Bachelor of Technology

by
Vishal Anand
11010170

under the guidance of
Dr. Ashish Anand

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati 781039, Assam, India

Abstract

Medical textual data present us with huge volumes of useful raw data, which can be pre-processed with the use of proper learning techniques. The information that is generated using the techniques can be leveraged to extract knowledge and predict varied medical correlations and conditions. Some examples of the predictions can include deciding the course of a given person's condition in the context of the related data from the medical corpus, the correlation of varied diseases with each other, among others.

To generate these analytical data, we have utilized the corpus of textual medical data to train artificial neural networks(ANNs). The ANNs have been loaded with the words from the corpus in an unsupervised fashion to train itself. The method employed utilizes the concepts of backpropagation using hidden layers. With the word vector representation generated, it can be used in varied tasks such as Parts-of-Speech tagging, Chunking, Named Entity Recognition, Semantic Role Labelling in a corpus' data. In this part of the thesis, I have focussed on the window-based model to analyze and generate the word-vector representation so as to extend it to a multi-task framework once the vectors have been trained.

Acknowledgements

I take this opportunity to express a deep sense of gratitude towards my guide Dr. Ashish Anand, for providing excellent guidance, encouragement and inspiration throughout the project work. Without his invaluable guidance, this work would never have been a successful one. I would also like to thank Muneeb T H and Sunil Kumar Sahu for their valuable suggestions and helpful discussions.

Vishal Anand
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati-781039, Assam, India
April 26, 2015

Honor Code

I certify that I have properly cited any material taken from other sources and have obtained permission for any copyrighted material included in this report. I take full responsibility for any code submitted as part of this project and the contents of this report.

Vishal Anand

Certificate

It is certified that the B. Tech. project “Learning Word Vector Representation in Multi-Task Framework" has been done by the student: Vishal Anand under my supervision. This report has been submitted towards partial fulfillment of B. Tech. degree requirements.

Dr. Ashish Anand
Faculty Supervisor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati-781039, Assam, India

Contents

Chapter 1

Introduction

1.1 Medical Textual Data Learning

When presented with a corpus of textual medical data, we can apply learning techniques to deduce relations and then try to predict relationship of the entities(features defined) which includes the subject's future condition, varied diseases, medicines and effects among others.

1.1.1 Motivation

To solve this problem people have used varied robust techniques wherein the features have been carefully defined and have used clustering methods among others to get satisfying results. But in the recent past, the focus has shifted to the use of neural networks employing the window based unsupervised training approach which has shown phenomenal results and has broken the benchmarks of many of the tasks in question. This approach has been used by Collobert et al.(2011) [1] and Huang et al.(2012)[2].

The visualization of the motivation of using a neural network is intuitive and corresponds to the development of a human brain. Going by an example, the motivation would be lucid. Let us imagine a person having general interests in a genre of movies and in some genres of music. When this person is presented with a new music or a new movie, which has not been previously classified into the correct genre, the person would either like it, dislike it or remain lukewarm to it. If the correct classification was in the subset of the taste of the person, the chances are very high the new product would be appreciated by the person.

If we try to map this to a neural network, we can assume the taste of the person as a trained neural network, with the datapoints for the training as the past movies and music that the person was exposed to in the past.

The new movie/music can be thought of as the new data point which has to be analysed and converted into knowledge for useful applications in the medicinal field. Thus, the neural network develops a sense of logic for teh new data points and an analogy can be drawn to the human brain the way the neural network is trained.

In the works of Collobert et al., on experimentation with the window based approach of training of the neural network, the results were phenomenal and had broken all benchmarks in this domain. Thus, I have investigated in this domain and have tried to study this field.

1.2 Neural Networks

The artificial neural network (ANN) is a system of logic and data structures that emulates the functioning of the human brain. A usual implementation of neural network involves the use of a large number of parallel processors, each comprising of a separate set of knowledge and data-trained weights. Initially a neural network is "trained" and fed large amounts of data and rules about data relationships (for example, "The sun rises in the east"). After the initial training, the logic of the ANN can then respond to an external stimuli(may or may not be textual data), and can even initiate interaction with the outside world(which can be the user of the system as well).

1.2.1 Representation of Neural Networks

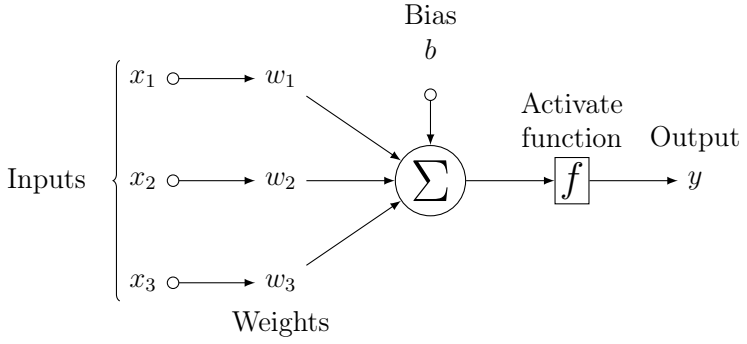


Figure 1.1: Neuron Representation

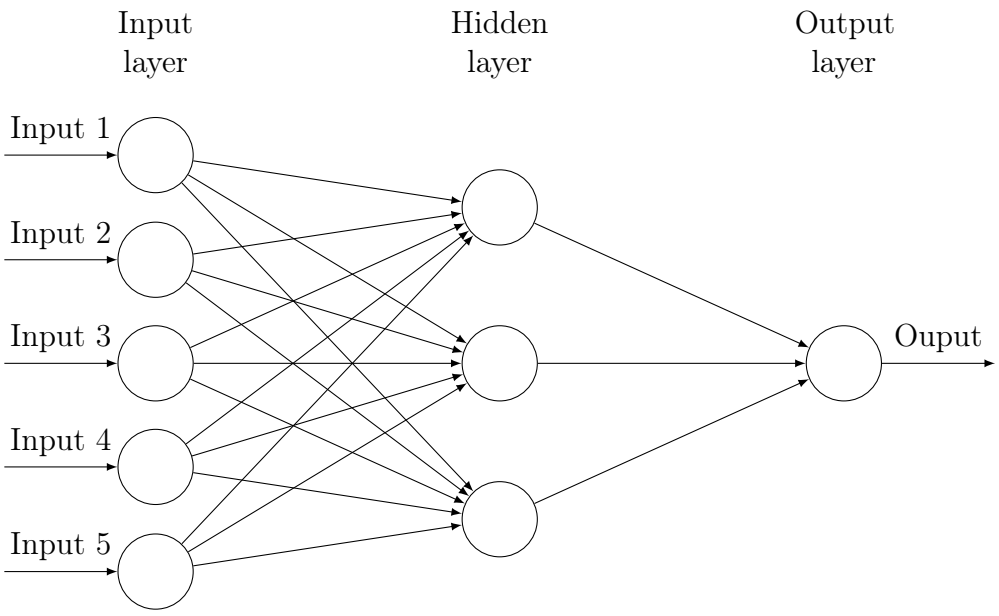


Figure 1.2: Neural Network with the hidden layers

1.2.2 Window-Based Neural Networks

In the window based neural networks approach, the only input we have is a corpus of data. From this we generate the positive and the negative samples for training of the neural networks by replacing the word vectors of a selected window. For the positive sampling, we generate the inputs by taking the word vector corresponding to the window-sized number of words from the corpus in an iterative order, of which the score produced by the neural network is evaluated. The centre-most word of the window is then replaced with an arbitrary word to create an instance of negative sample. The process of training for the specific central word of the window is continued for the random words in the corpus, till we get the condition $\text{Score}_{\text{correct}}$-$\text{Score}_{\text{random}}>1$. Post this, the next random word is taken from the corpus and the process is repeated, for all the windows from the training corpus.

This method has been used by Collobert et al. and has been shown to break the existing benchmarks and this model is very adaptive to change. Going by an example : we can visualize a problem that, if we are given a set of words with the centre-most word missing/corrupted and we have to identify the correct word. The training for the window based model would imply passing on the window to the neural network without the centre-most word(the centre-most word's input would correspond to zero). In this fash-

ion the score would be calculated; and thus when finally presented with a test data with incomplete points, we would pass this incomplete set of words and the score generated would correspond to the absent/incorrect word, thus identifying the word. The model can be easily extended to many other examples with slight changes to the existing framework. Hence I have taken to studying this approach in the thesis.

1.2.3 Forward Propagation

For a sentence from the corpus: "This is a sample sentence", and the window of size 5, each of the corresponding word vectors being represented by $\mathrm{x}_{\mathrm{word}}$, we have the following input to the neural network :

$$\mathbf{x} = [\mathbf{x_{this}}\ \mathbf{x_{is}}\ \mathbf{x_{a}}\ \mathbf{x_{sample}}\ \mathbf{x_{sentence}}]$$

The equations for the single hidden layer is as follows :

$$z = Wx + b \tag{1.1}$$
$$a = f(z) \tag{1.2}$$
$$s(x) = U^{\mathrm{T}}a \tag{1.3}$$
$$\Rightarrow s(x) = U^{\mathrm{T}}f(Wx + b) \tag{1.4}$$
$$J = max(0, 1 - s(x) + s(x_{\mathrm{e}})) \tag{1.5}$$

The equation 1.4 is called as the feed-forward process of the neutral network, and the equation 1.5 is the objective function that is minimized for each window. As this is a continuous function, we can use stochastic gradient descent for neural network training.

1.2.4 Backpropagation

Using this algorithm, we follow the inverse direction of the forward propagation and calculate the derivatives of the activation values at each level, which is then used to update the weights and vectors of the window in context. This approach has been used traditionally by Bryson et al.,1963 [3]; Werbos,1974[4] and Altman et al.,1994[5].

$$\frac{\partial S}{\partial U} = \frac{\partial}{\partial U}U^T a = a \tag{1.6}$$

$$\frac{\partial S}{\partial W} = \frac{\partial}{\partial W}U^T a = \frac{\partial}{\partial W}U^T f(z) = \frac{\partial}{\partial W}U^T f(Wx + b) \tag{1.7}$$

$$
\begin{aligned}
\frac{\partial}{\partial W_{ij}} U^T a &= \frac{\partial}{\partial W_{ij}} U_i a_i \\
&= U_i \frac{\partial}{\partial W_{ij}} a_i \\
&= U_i \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\
&= U_i \frac{\partial f(z_i)}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\
&= U_i f'(z_i) \frac{\partial z_i}{\partial W_{ij}} \\
&= U_i f'(z_i) \frac{\partial W_i x + b_i}{\partial W_{ij}} \\
&= U_i f'(z_i) \frac{\partial}{\partial W_{ij}} \sum_k W_{ik} x_k \\
&= \underbrace{U_i f'(z_i)}_{\delta_i} x_i \\
&= \delta_i x_i \\
\Rightarrow \frac{\partial}{\partial W_{ij}} U^T a &= \delta_i x_i
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial J}{\partial W_{ij}} &= U_i f'(z_i) x_j = \delta_i x_j \\
\Rightarrow \frac{\partial J}{\partial W} &= \delta x^T
\end{aligned}
$$

$$
\begin{aligned}
U_i \frac{\partial}{\partial b_i} a_i &= U_i f'(z_i) \frac{\delta W_i x_i + b_i}{\delta b_i} x_j \\
&= \delta_i \\
\Rightarrow U^T a &= \delta
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial S}{\partial x_j} &= \sum_i \frac{\delta S}{\delta a_i}\frac{\delta a_i}{\delta x_j} \\
&= \sum_i \frac{\delta U^T a}{\delta a_i}\frac{\delta a_i}{\delta x_j} \\
&= \sum_i U_i \frac{\delta f(W_i x + b)}{\delta x_j} \\
&= \sum_i U_i f'(W_i x + b)\frac{\delta W_i x}{\delta x_j} \\
&= \sum_i \delta_i W_{ij}
\end{aligned}
$$

1.3 Organization of the Report

The next section describes the way a section of the project (NER) has been dealt with and the ensuing chapter gives a timeline of the work that is to be extended from the current version to generate a multi-task framework for the neural network for textual medical data.

Chapter 2

Natural Language Processing

2.1 Tasks

In this section, a brief introduction of four of the standard NLP tasks on which the neural network is to be used for building the word vectors is presented: Part-Of-Speech tagging (POS), chunking (CHUNK), Named Entity Recognition (NER) and Semantic Role Labeling (SRL). For each of them, we consider a standard experimental setup and give an overview of state-of-the-art systems on this setup.

2.1.1 Part of Speech Tagging

Part of Speech tagging labels the words of a sentence into the corresponding literal mapping into their corresponding part of speech, viz. nouns, adjectives, verbs, adverbs, etc. In this fashion, the words of the corpus are segregated in lines with their syntactical role in the sentence. This involves a multitude of pre-processing such as segmentation, extraction of the stop-words, segregation of the more popular roots of the vocabulary for the tagging purposes to deal with the overheads involved in training for the relatively insignificant words. The use of the POS tagging with the afore-mentioned preprocessing tasks depends on the kind of learning and knowledge extraction required for a particular event. Among the best of the POS classifiers, these are based on classifiers using window-text based training, which then "use bidirectional decoding algorithms during inference" [1]. In general the approaches change the numbers to 'DGDD' named tag as the essence of the word remains the same for the POS tagging.

2.1.2 Chunking

In some literature, this is termed as shallow parsing. This task marks the parts of sentences with labels of noun-phrases and verb-phrases. Each word is assigned a single unique tag, generally encoded as a begin-chunk or inside-chunk tag. The benchmark driving systems use features composed of words, POS tags, among other features.

2.1.3 Named Entity Recognition

Named Entity Recognition(NER) tagging refers to the action of labelling and classification of words into pre-defined classes or categories, for example into Person/Non-Person, Location/Non-Location, Quantity/Non-qunatity among others. In this part of the thesis, I have worked in the NER tagging domain, which is described in the next chapter.

2.1.4 Semantic Role Labelling

The Semantic role labeling is sometimes referred to as shallow semantic parsing. The role of the Semantic Role Labelling(SRL) is to detect the semantic arguments associated with the predicate or verb of a sentence and their classification into their specific roles. Taking an example, we can try to label the words as the doer, the recipient, the object of the sentence and the action in relation to a sentence in question. This would relate to finding the proper meaning of a given sentence. The semantic representation resides at a higher-level of abstraction than a syntax tree. The reason for this is that even if the voice of the sentence changes, the syntactical tree would change, but the semantic role's tree would remain the same. In the wikipedia this has been brought to light with a very good example [6].

Chapter 3

NER Implementation

3.1 Introduction

In the current semester, I have taken a fragment of the multi-task framework and implemented the NER module of the word-vector training, which can then be generalized to the other tasks and then ultimately merged into a common representation of the multi-task framework. Based on the window based approach described in the section 1.2.2, at the hidden layer, I have used the hyperbolic tangent function, and at the output layer, the softmax function used for the two layered decision is sigmoid function.

3.1.1 Equations derivations

For the model of neural network I have trained, the window of the word-size used is 5, where each of the words has 50 sized word vectors. The size of the hidden layer is 50 nodes. After including the bias, the input to the neural network each node is 251((5*50) + 1) and we find the score in the forward direction using the equations from the following subsection. Once the scores are calculated, we find the backward propagation equations for each of the nodes. The NER tagging feature implemented is that for a PERSON/NON_PERSON. The PERSON tag has a score of 1 and the tag of NON-PERSON has a tag value of 0. The training set comprises of a textual data-set which has pre-tagged values for the feature and the same is used for the error function (viz. the difference between the output layer's score versus the actual value for the central word of a context window, described in the equations). The activation values at the hidden layer are fed into the hyperbolic tangent function (f(x)) and the activation value at the output layer is fed into the sigmoid function (g(x)) which returns the score value for the central word of the context window in the range [0,1].

In the following derivations, I have used a toy example of the window size to be comprised of three words, each of which has the word vector matrix of 3X1; and the number of size of the hidden layer to be comprised of three nodes. The output layer comprises of a single node. With the final results coming in the form of a set of matrix equations, the same would be generalized for any configuration of compatible neural network which has a single hidden layer.

Forward Propagation

$$
\begin{aligned}
&Let x \in \Re^3, A \in \Re^3 h_\theta \in \Re^1 \\
&\qquad Z_j^1 = w_{j1}x_1 + w_{j2}x_2 + w_{j3}x_3 + b_j^1 \\
&\qquad Z^1 = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + w_{13}x_3 \\ w_{21}x_1 + w_{22}x_2 + w_{23}x_3 \\ w_{31}x_1 + w_{32}x_2 + w_{33}x_3 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix} \\
&\qquad Z^1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix} \\
&\qquad = WX + b^1
\end{aligned}
$$

The first layer's calculations are done

$$
\begin{aligned}
a_j^2 &= f(Z_j^1) \\
A &= \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \\
&= \begin{bmatrix} f(Z_1) \\ f(Z_2) \\ f(Z_3) \end{bmatrix} \\
&= f(Z^1) \\
Z^2 &= u_{11}a_1 + u_{12}a_2 + u_{13}a_3 + b^2 \\
&= \begin{bmatrix} u_{11} & u_{12} & u_{13} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + b^2 \\
&= U^T A + b^2 \\
h_w(x) &= g(U^T A + b^2) \\
&= g(Z^2)
\end{aligned}
$$

Backward-Propagation

$$
\begin{aligned}
Error &= \frac{1}{2}(y - h_w(x))^2 \\
Update of u_{1i} & \\
\frac{\partial Error}{\partial u_{1i}} &= (y - h_w(x))\frac{\partial h_w(x)}{\partial u_{1i}} \\
&= (y - h_w(x))\frac{\partial g(Z^2)}{\partial u_{1i}} \\
&= (y - h_w(x))g'(Z^2)\frac{\partial Z^2}{\partial u_{1i}} \\
&= (y - h_w(x))g'(Z^2)a_i \\
Let \quad \delta_1^1 &= (y - h_w(x))g'(x^2) \\
\frac{\partial Error}{\partial u} &= A(y - h_w(x))g'(Z^2) \\
&= A\delta_1^1 \\
\frac{\partial Error}{\partial b^2} &= \delta_1^1 \\
\frac{\partial Error}{\partial b_j^1} &= \delta_j^2 \\
\delta_j^2 &= \delta_1^1 u_{1j} f'(a_j) \\
\frac{\delta Error}{\partial w_{ji}} &= \delta^2 X^T \\
\frac{\delta Error}{\partial x_i} &= w^T \delta^2
\end{aligned}
$$

Stochastic Gradient Descent

With the error derivatives calculated for the θ parameters, one can update the parameters using the equation :

$$\theta = \theta - \alpha\frac{\partial Error}{\partial \theta}$$

Here, the α's value is decided by the user, and is generally set to 0.001 for training of the neural network.This is the Stochastic gradient descent which is used for updating the values of the parameters of the neural network and the word vectors as well.

Chapter 4

Word Vector construction

4.1 Introduction

The Artificial Neural Network on which I have used the NER implemntation depended on the mapping of each of the words to a n-sized word vector. The vectors were obtained from the training of the entire Wikipedia text and was trained on the complete Wikipedia corpus and was obtained from Richard SocherâĂŹs implementation which had been run on parallel GPUs. The particular word-vector sample was sized at 50 each feature points. Since, the objective of the thesis is to learn word-vector representation in a multi-task framework using neural networks, the corpus I had to train the neural network had to be a medical textual corpus.

4.2 Data Collection

I obtained a set of Medical Publications (referred to as PubMed henceforth), and selected a 424 MB data-sample from it and it comprised of a total of 152,413 research papers. Writing python and bash scripts, I extracted these into PubMed Extract which comprised of just the abstracts of the papers, so as to train on the essence of the paper, rather than get the complete data. Once the abstract training is done, the motive would be to train it on the complete data, which however is expected to get similar results, since the abstracts are written in such a fashion to mirror the content of the paper. Hence, I wrote python and bash scripts to extract the abstracts of the PubMed papers. The data now amounted to a 250 MB file which comprised of all the abstracts stored in a single line.

4.3 Data issues and configuration

The primary objective to train the Neural Network using a manageable chunk of data has been done. However, the data set could not be visualized, and the text editors and related terminal-shell based tools would keep on crashing while seeing the text for checking the validity of the data-conversion.

A small tweak solved the issue, which involved saving the PubMed abstract in a structure wherein each line corresponds to each of the research paper's abstract. The reason why this tweak worked was that the entire line from the data file was being loaded into the RAM and since the entire file could not fit into the available space, the SWAP page frames were getting created, which was causing the operations to get slowed down terribly and was the main cause to crash the text operations.

Logically this seemed more relevant since, while training for the word vectors' the sliging-window vector would traverse through a set of words in the document fed into the neural network. So, when a particular abstract ends, the next abstract would be non-related to the previous abstract one, and hence they should be padded with START-OF-DOCUMENT and END-OF-DOCUMENT respectively so as to make the division across the different abstracts visible to the neural network.

4.4 Experimental Setup

I started with an in-house development of the neural network targeted towards to the Collobert's window-based neural network approach for word-vector training method.

The in-house development setup was initiated with a base which can enable easy expansion into various methods of training of word vectors, and even for activities such as usage of the vectors to evaluate incomplete sentences, to come up with the missing word in a given sentence.

4.4.1 Motivation for the setup

An example for the expansion of the in-house setup can be visualized in this fashion:

> We get a data-set which has to be tested for its correctness. We can initialize the neural network with the trained values from a related corpus done in accordance with a pre-defined size of window. Thus, with the trained word-vector mapping of the words and

weights assigned to the layer of the neural network, the centre-most word of the window is deleted and passed through the neural network. The network tries to assign a vector from its dictionary and evaluates the score of the window. If the score comes out to be true, the vector is reverse mapped with the words in the dictionary and compared with the given middle-word.

As an extension to this problem. One can try to predict the ensuing stages of a patient's health. One can try to find the future state of a patient based on some medication suggested by the professional. To get to this state, we can actually have no middle-word. In such an extension of the neural network, the training would be carried out in the context of the last word in the window, in place of the centre-most word. Then, with a given window, the following stage(or medication) of a given patient would be evaluated and can be used as an indication of the effectiveness of a given drug in the prior stage.

4.4.2 Input to the system

In order to make the neural network very much extensible and adaptive, I made the network independent of any set of rules. As in, the input of the system would just be a text file along which the entire network is to be trained. Thus, all the system cares about is the input which is then processed and made compatible with the training of the neural network.

The user has the option to specify the

- deep neural network size [=50]
- window-size [=5]
- word-vector size [=50]
- iteration-count upper limit [=5000]
- epsilon value [=0.001]
- input file [= *in.txt*]
- vocabulary log file [= *build_vocab.txt*]
- vocabulary log with frequencies [= *build_vocab_count.txt*]
- vector output [= *word_vector_output.txt*]

- upper case allowance [=True]
- verbosity [=True]
- arguments [=True]

Each of these options is optional, even if the user does not enter an input file, "in.txt" is used as a default file for training of the neural network. However, if the input file is specified by the user, the rest of the parameters are set to a default value by the system.

The deep neural network size refers to the number of nodes in the hidden layer of the Artificial Neural Network system.
The window size refers to the parameter of sliding window which is used while training for the Collobert's approach.
The word-vector size refers to the dimensionality of each of the word vector to be generated.
The iteration-count upper limit refers to the maximum number of iterations the system would try to train the network in case the weights and vectors do not converge by these pre-defined set of repetitions of back-propagations.
The epsilon-value refers to the coefficient of the back-propagation update value to be used while updating the weights and word-vectors.
The input file simply refers to the file that is to be utilized for the training of the neural network.
The vocabulary log file parameter refers to the log file's name to be generated while processing the input file.
The vocabulary log file with frequencies refers to the verbose log fileâĂŹs name to be generated while processing the input file, which would also comprise of the repetitions of the given word in the corpus file used for training of the network.
The vector output takes in the file in which the final set of word vectors is to be written into.
The upper case allowance takes in a boolean value which signifies if the corpus has to be treated as a cluster of upper and lower case characters, or only a single case set.
The verbosity refers to the verbose outputs for debugging and related purposes in order to get to speed with the working of the entire system.
The arguments when set to true outputs the other arguments entered by the user so as to double-check the sanctity and working of the training of the neural network.

4.4.3 Weight initialization

The neural network is firstly initialized with random weights based on the parameters fed into the system. Since it consists of two layers of weights, one at the interface between the input words(W1) and the hidden layer and finally the other one at the interface between the hidden layer and the output layer(W2).

$$
\begin{aligned}
input\_size &= args.word\_size \times args.window\_size \\
W\_1 &= np.random.randn(args.deep\_neural, inputsize + 1) \times .001 \\
W\_2 &= np.random.randn(1, args.deep\_neural + 1) \times .001
\end{aligned}
$$

Once the weights have been initialized I store it into a model variable, which is to be updated as the training proceeds in the later stages.

4.4.4 Word Vocabulary building

The input file on which the neural network is to be trained is read line by line and they are stripped into tokens.
If the user specifies that the upper case has to be left as it is, we ignore changing and modifying the cases, or else the complete set of words are treated as the same and the vocabulary is then stored in variables.

However, if we follow the Collobert's approach of sliding-window approach, we realize that each of the abstracts or continuous textsâĂŹ has to be padded so as to consider the beginning of each document as a positive sample while training the network. Hence, I used a padding using the word : "DGDD" at the beginning and at the end of each of the documents while reading in the lines from the input file.

Thus a file which has the string S, then the variable S is prepended and appended with ["DGDD"] those number of times such that the first word and the last word can be made as the center of the window when the sliding window encounters them.

The complete words are saved into a dictionary with their counts as well. In case a wordâĂŹs count is lesser than the threshold specified by the user, then the word is discarded by the system.

4.4.5 Word Vector Initialization

From the words' dictionary populated in the previous step, each of the words in the given threshold limit is initialized with a random set of vectors :

$vec\_init = [random.uniform(0, 2) for\_in range(0, args.word\_size)]$

For the dummy words added to the document, i.e. "DGDD", a unary vector is initialized with each of these.
$vec\_dummy = [1] \times args.word\_size$

For faster access of the words, the dictionary is saved in a sorted fashion so as not to traverse through the entire set of words in the dictionary if the threshold has been reached. This would bring down the computation really well, since the words with lesser frequency are too many, and hence the control of the program would know when to step out of the process of word vector initialization.

During this process, the word vectors are also written into the given file for the analysis by the user or for visualization purposes, as to know how the word vectors got updated and to study how well the neural network has been trained.

4.4.6 Word Vector Processing

In this section of the neural network training, the input file is read and each of the abstract is sent to the a parser, which converts the whole set to a sliding window format, which is what is actually required by the neural network. This process is then iterated over and over again till the complete input file has been handled and parsed. In case the words are encountered, which have to be left out of the system (i.e. ones which have lesser frequency than the threshold, they are replaced with the dummy variable as we are not worried about optimizing the vector corresponding to the left-out word).

Sliding Window

Once the a particular document's entire content is given to this segment, the whole set of sentences is checked for proper padding by the dummy words : "DGDD". And once basic checking of the input is done, the words are clustered into sliding windows. As an example, let us consider :

> "This is a sample sentence for neural network training using Collobert's approach"
>
> This sentence is converted to (considering 3 window-sized parameter): "DGDD This is a sample sentence for neural network training using Collobert's approach DGDD".

In case the parameter of window-size is specified as 5, the same sentence is converted to : "DGDD DGDD This is a sample sentence for neural network training using CollobertâĂŹs approach DGDD DGDD".

The sliding window parses the sentence to : ["DGDD DGDD This is a", "DGDD This is a sample", "This is a sample sentence", "is a sample sentence for", "a sample sentence for neural", "sample sentence for neural network", "sentence for neural network training", "for neural network training using", "neural network training using CollobertâĂŹs", "network training using Collobert's approach", "training using CollobertâĂŹs approach DGDD", "using Collobert's approach DGDD DGDD"]

[The above example considers the sliding-window size parameter to be of length 5]

Mapping words and first layer

Once the sliding window has been created, the words fed into the neural network is then mapped into their corresponding word-vectors initialized in the previous sub-section. With the mapping produced, the complete word vector of the window comes out to be "word-vector-size * window-size". However, we add a bias entry(valued at 1) in the vector of the sliding window, so as to account for the bias in the first layer of neural network. Thus the sliding window is passed onto the first layer where it is forward propagated and the output is stored as Z1 :

$$\begin{aligned} add\_row &= np.array([1]) \\ add\_row &= add\_row.reshape(1,1) \\ X &= np.vstack((add\_row, X)) \\ Z1 &= W1.dot(X) \end{aligned}$$

First layer and activation

After the first layer has been invoked, the output is then sent over to an activation function so as to streamline the functioning of the training(as has been decided by the programmer). Here A1 denotes the activated first layerâĂŹs output value. I was using a sigmoid function for the first layer initially. A detailed analysis is to be followed in a later section.

$$A1 = sigmoid(Z1)$$

Second and Final Layer

After getting the first layerâĂŹs activated value, the bias is again added for the second layer and is used for evaluation of the input for the second layer's activation function. In the activation function, I have used the identity function.

$$A1 = np.concatenate((np.array([[1]]), A1), 0)$$
$$Z2 = W2.dot(A1)$$

Negative Sample Generation

Once the sliding window is evaluated to some value, the middle word is replaced with a random word and is then used as a negative sample to form as a guidance for the neural network so as to drive its weight into the correct direction. This can be understood in this fashion : If the correct word is replaced by a random word, then the trained neural network should be able to point out to the person that the word introduced is incorrect, hence the system then evaluates the score for the incorrect word and tries to drive the weight of the network and the vectors of the words as well, such that the following property holds :

$$"Score(correct) >= Score(incorrect) + 1"$$

Backward Propagation

The replacement of the centre-most word is carried out in such a fashion that each of the admissible word in the vocabulary is substituted and then evaluated for the forward propagation and the corresponding error is then evaluated and is used for backward propagation :

$$\begin{aligned} margin &= 1 \\ dZ2 &= np.zeros\_like(Z2) \\ dZ2[0][0] &= error\_value \\ dW2 &= dZ2.dot(A1.T) \\ dA1 &= np.dot(W2.T, dZ2) \\ dA1 &= dA1[1:] \\ dZ1 &= dsigmoid(Z1) \times dA1 \end{aligned}$$

$$dW1 = dZ1.dot(X.T)$$
$$dx = np.dot(W1.T, dZ1)$$
$$dx = dx[1:]$$

Care is taken to remove the bias values while back-propagating to correctly evaluate the update variables. We finally only need the following values for updates :

- dW1
- dW2
- dx

Update Of The Vectors and Weights

The word vectors and the weights corresponding to the layers of the neural network are updated in the following fashion :

$$W1 = W1 + dW1$$
$$W2 = W2 + dW2$$
$$X = X + dx$$

The final statement is actually dealt with care, since it consist of various words that have to be updated in the complete sliding window.

Iteration Control

The entire document's contents are iterated over once for the sliding windows generated and the control goes back to the input file, when the next document's text is read and parsed again.

4.4.7 Input File Iteration

Once, the document that is parsed is sent over to the word-vector-processing unit, the input file's next line is read and parsed, which is then processed. Once the entire input file has been processed once, the total error value encountered is added up and compared with a threshold. If the threshold is crossed, then the entire input file is re-read and re-processed on the updated weights and corresponding updated word-vectors. This process goes on again

and again either till the threshold value is more than the error value encountered in the previous iteration, or till the number of iteration of the input file specified by the user has been exhausted.

Printing the vectors and weights

This section of the system logs all the weights and word-vectors that has been arrived at. This is exported into csv formats and text formats for ease of reuse in terms of scripting usage(text format) and also for excel viewing (csv format)

Chapter 5

Implementation and Visualization Issues

5.1 Introduction

While developing and analyzing the flexible Neural Network Framework for training the network and the word-vectors a multitude of errors cropped up unexpectedly and it was quite an effort to whisk away the problems and continue forward.

In the following sections, I would point out some of the major problems that had cropped up during the process of design, development and analysis of the working system.

5.2 Implementation Issues

While creating the raw framework for the neural network training, some of the major issues faced are :

5.2.1 Activation Function

While deciding upon the activation function in the neural network, it was not very clear how to go about fixating a particular function. A multitude of references pointed at the usage of the sigmoid function at the hidden layer for the training of the word-vectors, however it was not very much obvious as to what the input to the function would correspond to. If the value comes out to be >=10, the sigmoid function maps it to 1.000. It was observed that for the correct sliding window and for the incorrect sliding window as well, the input to the activation function would almost always correspond to >=10 and the

sigmoid would always map it to 1.000 for each of the rows. The final layer after processing would then result in the same values for both the correct and incorrect sequences. Thus, the system was insensitive to detecting the correctness (and error) of the sliding sequences. It may also be noted that the above actions can be acceptable during training of the neural network; however the training would not make any progress, since the system would always come up with the same output score for all the sequences. Thus, the back-propagation never actually worked.

To deal with this issue, I changed the way the weights were initialized. Firstly, the weights of the layers were getting randomly initialized from floating point values in the range of (1-10), this was changed to floating point values in the range of (0-1). It also did not yield any result.

The next thought of meddling with the initialization of the word-vectors came up, but it would be wrong to play around with the vector initialization, since after the optimization is done bny the system, the vectors can practically range to any real value. So, a limited starting point would not make sense.

Since most of the times, the score of the hidden layer produced a score of >=10 (though even a score of >=2 would have been unacceptable, since the sigmoid would produce 1.000 score after the activation function gets invoked for the sliding windows), I tried normalizing the rows of the first layer such that each row corresponds to a unit vector and thus chances would be better of having a smaller score at the input of the activation function. This also failed, and it was also observed that with a far lesser magnitude of the row vectors, the sigmoid function would still spit out unit outputs corresponding to each row (i.e. corresponding to each word).

This led to the decision of removing the sigmoid function as the activation function and replacing it with the identity function. Since here we were only concerned with the differentiation of the correct sequence with the incorrect sequence, the sigmoid function was not serving any purpose to this cause. It also brought out the realization that the sigmoid functions are generally used in cases which involves the multi-class classification of the output since they involve multi-layered outputs and the pre-tagged labels guides the training of the vectors. However in our case, we were producing the the negative samples by replacement and the system was unable to correctly deduce the samples. The system seemed to work fine. But there cropped up another issue.

5.2.2 Out of Bounds Error

Once the sigmoid function was removed, the error value calculation was done smoothly and the back-propagation seemed to work fine. However, the differentials that were being evaluation started getting inflated really really fast and would balloon up to 10^{22} and even 10^{100}, after which it would get stored as $+\infty$, or at times it would deteriorate into $-\infty$ (in case the direction would turn out to be negative). Installing some sort of step function seemed appropriate, but the previous episode with the sigmoid function was acting as a deterrent to this thought.

It was then decided to come up with a error scaling function, which would determine how fast the error is ballooning and would dampen the back-propagation. Ideally it should have driven the weights in the correct direction fast, but in this process, the optimization of the weights that is being achieved would be overshot and then the system would start oscillating in the form of a banana function optimization. Thus a steady drive towards the solution seemed the best way to go about it. In case the back-propagation values were above a certain threshold, the magnitude of the values were reduced to 10^3 and the back-propagation was carried. This ensured the whole system moved gracefully and would hopefully come to a solution fast, rather than oscillate (not to mention resolving the out of bounds error)

5.2.3 Time for Training

The approach taken in the training of the Neural Network originally involved the use of parallel GPUs and corpus being the complete set of Wikipedia data, took a very large computing timing. When the first run of the system was done (when the activation error was found), it had taken 90 hours with only 3 iterations of the complete input file, thus it averaged a total of 30 hours for each of the iterations of a portion of the PubMed extract. Another factor to be kept into consideration is that the server (having 32 GB RAM) on which the system was executed, had varied processes running alongside this job process and hence it had to wait a long while, thus the actual time would be a great deal shorter than the 90 hours recorded. Thus, it has to be converted to a parallel computing approach so as to get acceptable training time.

5.3 Visualization Issues

In order to compare the word-vectors generated by the Collobert's approach and by other approaches, I looked at various visualization techniques, but since it was a set of multi-dimensional vector, most of the forums and data

platforms suggested the use of tSNE visualization technique which has Principal Component Analysis(PCA) at the heart of itself.

However, when the data to be analyzed was looked at closely, it was observed that the vectors were generated for 5.45 lakh words and corresponding to 50 vectors per word, the file generated corresponded to 260 MB, and when it corresponded to 200 vectors per word, the file size amounted to 1.1 GB. When these were set as inputs to the tSNE platform, the PCA would crash while tryiong to invert the matrix in one of its routines.

Looking for alternatives, I looked at other implementations of the tSNE and looked at the excel format analysis of the same. In this, one had to simply save the vectors clustered with the words in a csv or excel sheet and the same would be taken up by the tSNE platform and a 2-dimensional graph would be generated. For some reason the data was not getting converted to the excel format. While trying Google Drives and related tools available openly, the problem could still not be solved. It was then evident that a majority of the tools do not allow for saving files >20 MB of data, hence the problem was being faced. Also, it was evident after trying a lot of work-around that the excel formats only save at most 65000 rows of data, and not more than that. Hence the entire point of visualizing the data using excel formats was defeated.

Matlab implementation of tSNE also failed(this would also crash at the PCS matrix inversion procedure). Python, Matlab, and Java implementations also failed. A possible reason could be the RAM of the system it was being run on (6 GB of RAM). Escalating the RAM of the system to 32 GB also did not help, and the system would throw Out of Memory Error.

Finally it was decided to analyze only a select few words (i.e. few thousand words only), such that the tSNE is able to reduce the dimensionality of the word-vectors and a visual estimate of the closeness of the words is obtained and visual evaluation of the word-vector producing technique is achieved.

This approach yielded graphs for comparison(across various techniques) as is mentioned in the next chapter.

Chapter 6

Visualization of Word Vectors

6.1 Introduction

I compared the word vectors produced using various techniques namely, Collobert's neural network in-house approach, word2vec approach and Glove approach. For the word-vectors, I had selected a setting of 50 vectors corresponding to each of the words and applied tSNE on these to get the following graphs. The vectors were trained from the PubMed corpus.

From the following sections, we can see that the three visualizations are widely varied. But to make an accurate estimate of what the vectors are really signifying, they have to be tested and mapped to a tagged data-set. The use of standard Natural Language Processing(NLP) unit testing such as Parts-of-Speech(POS) tagging, Named Entity Recognition(NER) tagging would give a accurate measure of how efficiently the word vectors have been evaluated.

Each of the diagrams correspond to dimensionality reduction from 50 dimensional vectors to 2-dimensions using the tSNE approach by Laurens van der Maaten's approach[7]

6.2 Word2Vec

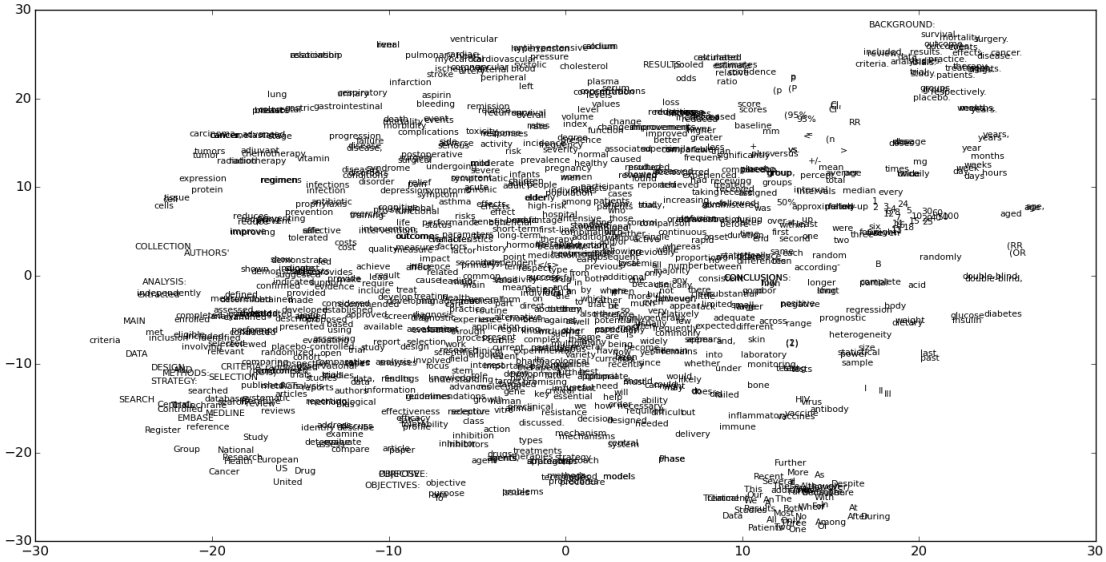


Figure 6.1: Word2vec's[8] tSNE visualization[7] of 1000 most frequent words from the PubMed corpus.

6.3 Glove

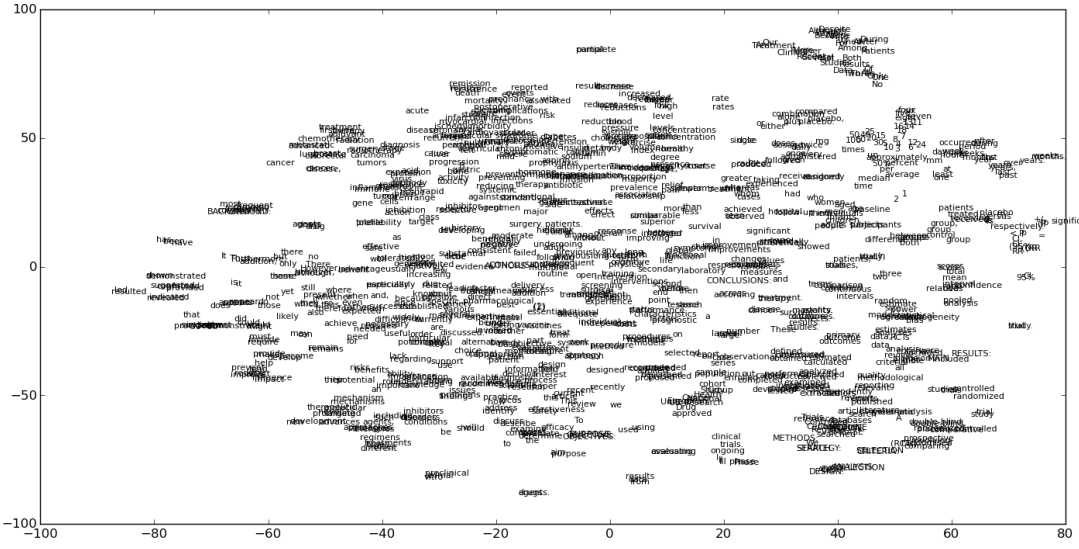


Figure 6.2: Glove's[9] tSNE visualization[7] of 1000 most frequent words from the PubMed corpus.

6.4 In-house approach for Sliding Window Neural Network

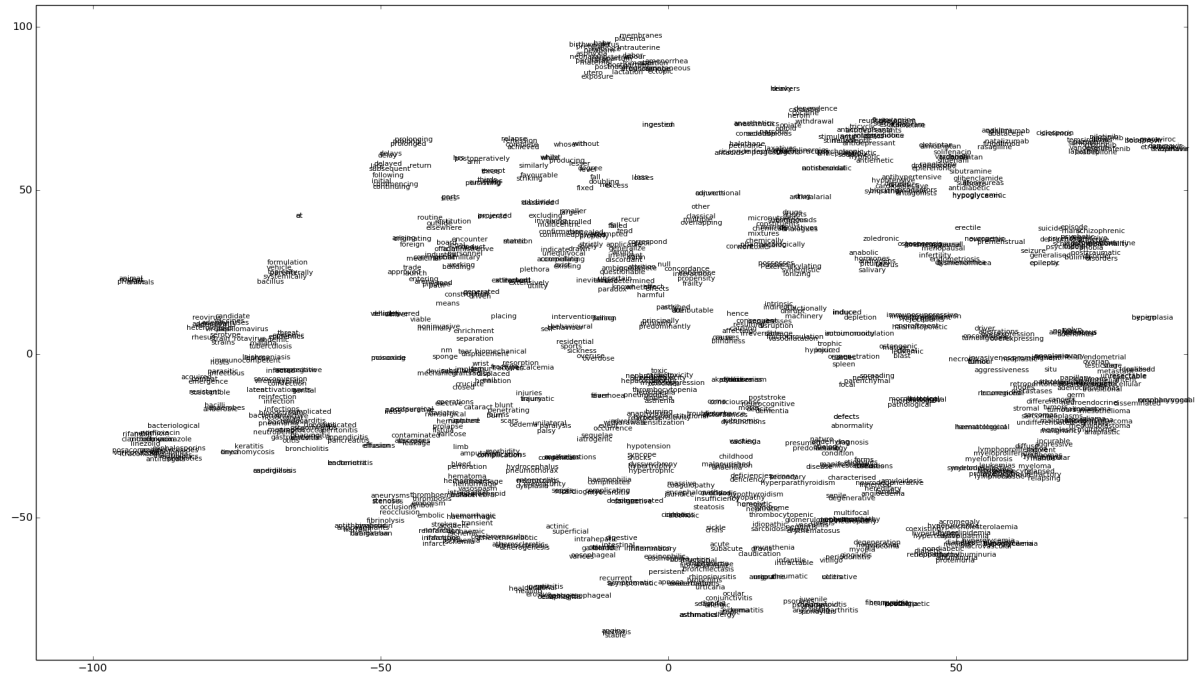


Figure 6.3: In-house[10] tSNE visualization[7] of 1000 most frequent words from the PubMed corpus

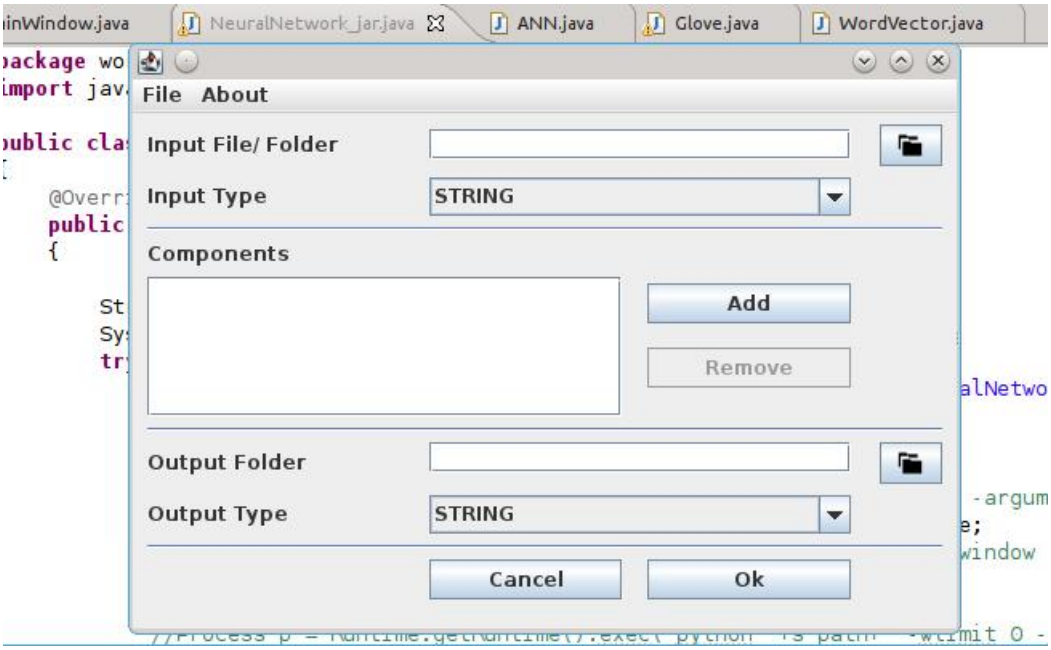
Chapter 7

Integration into Software Suite

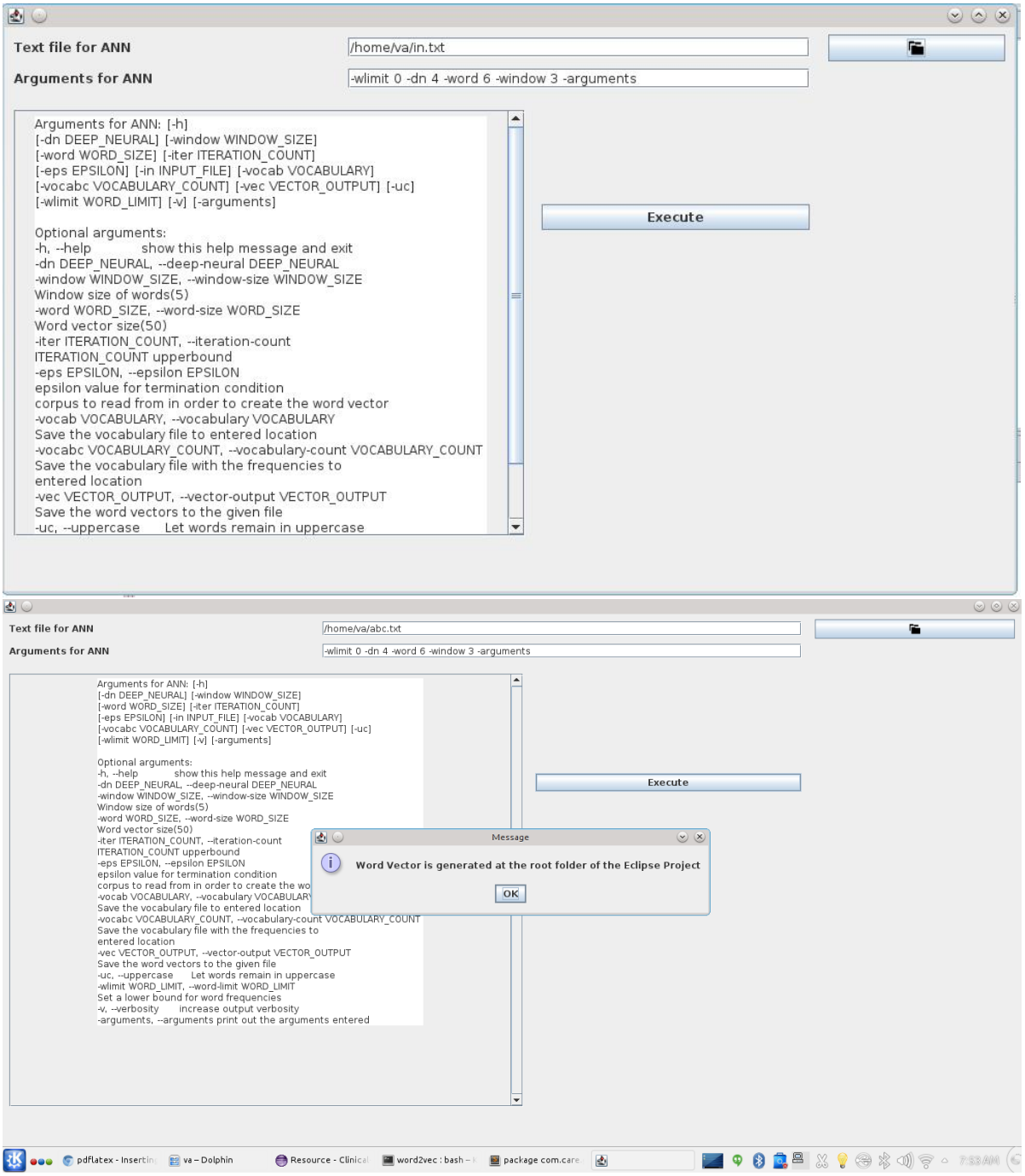
7.1 Introduction

The Neural Network framework was embedded into the Clinical Data Extraction Software Suite. Following this, the frameworks for Word2Vec and Glove's word-vector generation approaches were integrated as well.

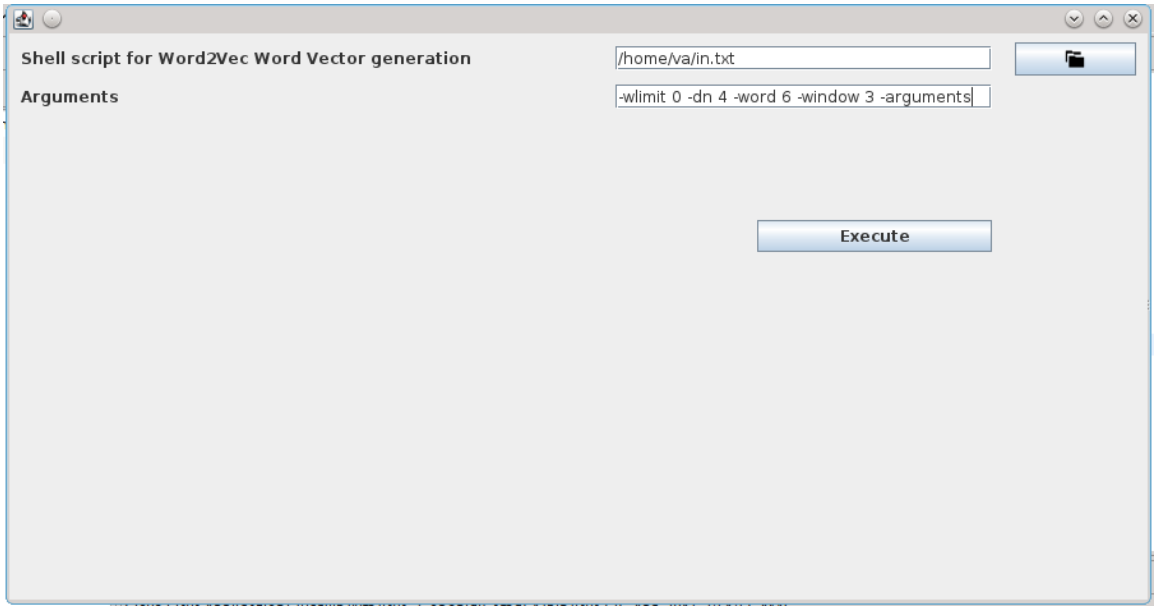
7.2 Root Window



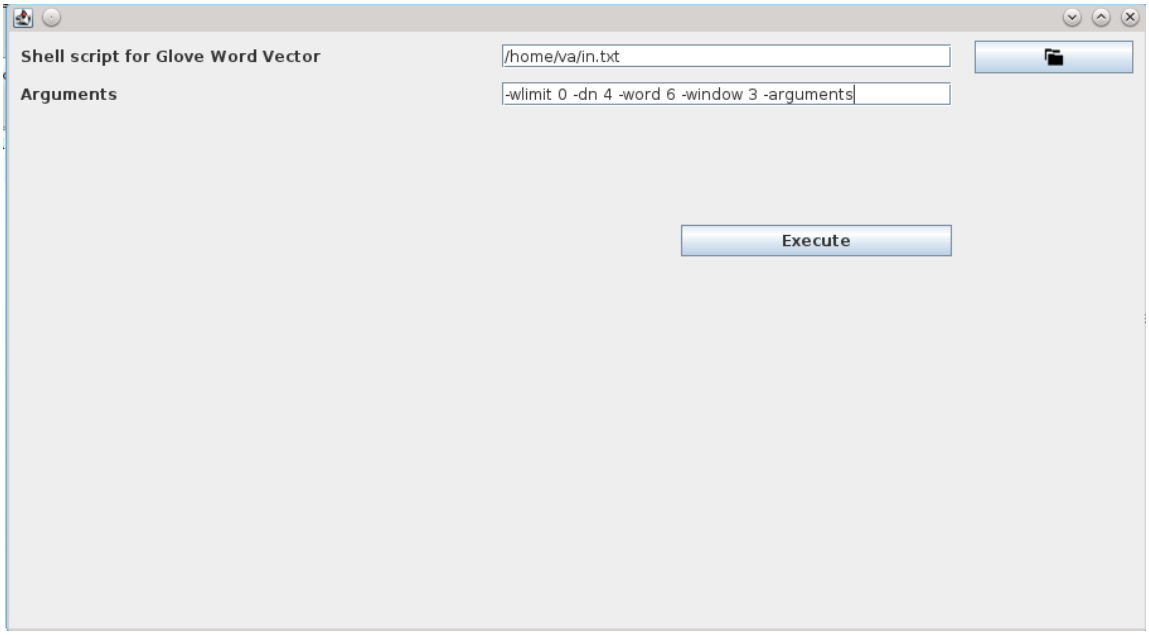
7.3 In-house Collobert's Neural Network Approach



7.4 Word2Vec

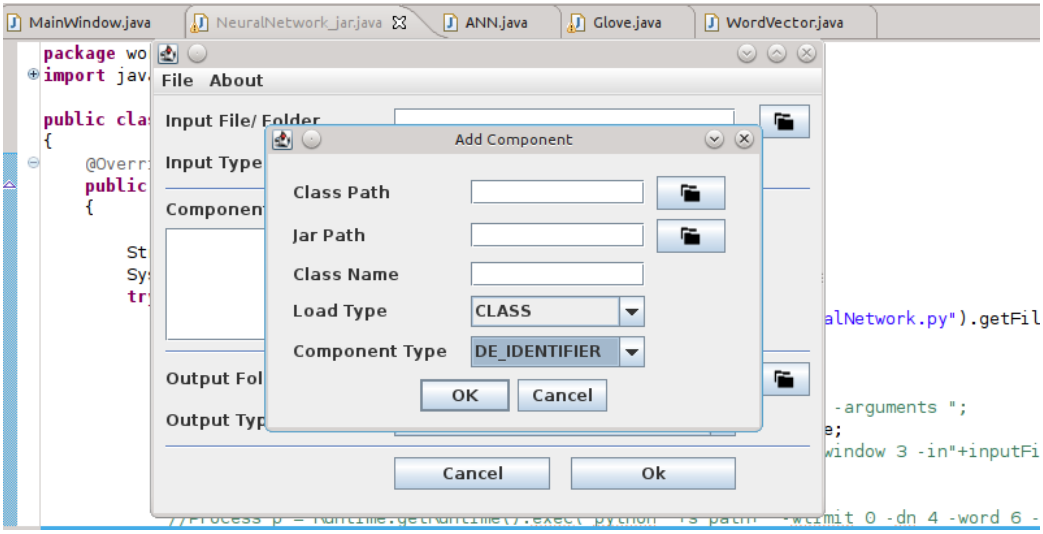


7.5 Glove Word Vector Approach



7.6 Other Features

The clusters of the already existing features in the suite(de-identifier, stopword-removal, stemmer, tokenizer, dictionary builder, pre-processor) are also shown below for reference :



7.7 Comments on the In-house Neural Network integration

It may be noted that the software suite has a cluster of features, but the Neural Network integrated into the system has no dependency with the existing framework, the primary reason being that the complete Neural Network can be completely configured in terms of the parameters alone and internally has all the features of tokenizing, dictionary building, pre-processing, stemming, stop-word-removal, etc. A set of other features can be easily added to the Neural Network system. While adding additional features into the Suite may be tedious, as it might involve taking and parsing input formats, however the in-house Neural Network only has to be fed with the raw input file and if at all the pre-processing has been done by another agent, the same can be supplied into the system as parameters, so as to avoid the same preprocessing. Still, since a majority of time is taken up by the iteration of the documents and updating of the weights and word-vectors, the preprocessing is insignificant in terms of overhead for the Neural Network.

If at all some pre-processing data has to be passed to the network, then

this should be in terms of the weights and mapping of the pre-evaluated word-vectors, since this involves consumption of really huge amount of CPU resources and time.

Adding on to it, the Neural Network can be integrated with other methods of evaluating the word-vectors, such as considering the global context as well apart from the local context of the sliding window, to name a few.

In terms of the way in which the integration of the code is carried out in the Suite, a very elegant approach that was employed was invoking the base code from a Java function, which was converted to a executable Jar file. In the software suite, when the corresponding button was clicked, the action event associated with it unzipped the jar file and after executing the command, removed the extra log files that get generated, which is generally used for the reasearch purposes and debugging, but is of little interest to the person using the system. However, the parameters to the Neural Network can be easily added to specify if the log files have to be left as it is, or only the word-vectors have to be stored into the system after execution of the programme.

Bibliography

[1] R. Collobert et al. Natural language processing (almost) from scratch. *J. mach. learn. res.*, 12:2493–2537, Nov. 2011. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1953048.2078186.

[2] E. H. Huang et al. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th annual meeting of the association for computational linguistics: long papers - volume 1*. In ACL '12. Association for Computational Linguistics, Stroudsburg, PA, USA, 2012, pp. 873–882. URL: http://dl.acm.org/citation.cfm?id=2390524.2390645.

[3] A. E. Bryson et al. Optimal programming problems with inequality constraints. *Aiaa journal*, 1(11):2544–2550, 1963.

[4] P. Werbos. Beyond regression: new tools for prediction and analysis in the behavioral sciences, 1974.

[5] E. I. Altman et al. Corporate distress diagnosis: comparisons using linear discriminant analysis and neural networks (the italian experience). *Journal of banking & finance*, 18(3):505–529, 1994.

[6] Wikipedia. Semantic role labeling — wikipedia, the free encyclopedia. 2014. URL: http://en.wikipedia.org/w/index.php?title=Semantic_role_labeling&oldid=632438139.

[7] L. v. d. Maaten et al. Visualizing high-dimensional data using t-sne, 2008.

[8] Google. Word2vec. 2013. URL: https://code.google.com/archive/p/word2vec/.

[9] J. Pennington et al. Glove: global vectors for word representation. In *Empirical methods in natural language processing (emnlp)*, 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[10] V. Anand. Colobert's approach on window based neural network. 2015. URL: http://github.com/vishalanand/Neural-Network/.