CONCEPT EXTRACTION FROM MEDICAL DISCHARGE SUMMARIES USING DEEP LEARNING

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

BACHELOR OF TECHNOLOGY

in

Mathematics and Computing

by

Anurag Pratik
(Roll No. 10012310)

to the

DEPARTMENT OF MATHEMATICS
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, INDIA

April 2014

CERTIFICATE

This is to certify that the work contained in this project report entitled "**Concept Extraction from Medical Discharge Summaries using Deep Learning**" submitted by **Anurag Pratik** (**Roll No.: 10012310**) to Indian Institute of Technology Guwahati towards partial requirement of **Bachelor of Technology** in Mathematics and Computing has been carried out by him under my supervision and that it has not been submitted elsewhere for the award of any degree.

Guwahati - 781 039 Dr. Ashish Anand Dr. Arabin Kumar Dey

April 2014 Project Supervisors

ABSTRACT

The main aim of the project was to apply techniques of Deep Learning to a slight variant of popular problem discussed in Natural Language Processing. The main problem statement was given and worked upon by many participants in the 2010 i2b2/VA (Informatics for Integrating Biology and the Bedside) challenge on concepts, assertions, and relations in clinical text. The details of the challenge have been illustrated in the introduction chapter later.

In the previous semester, as part of the project work in MA498, we explored various techniques that could be used to solve the problem of concept extraction. Concept extraction involves parsing clinical text in the form of medical discharge summaries and identifying and extracting the text corresponding to different categories: *patient medical problems, tests, treatments* and *none*. Solving this problem is similar to building a NER (Named Entity Recognition) tagger that takes as input unlabelled text and outputs the appropriate tag (or class) for each word (token) in the text. An elementary NER tagger for classifying text into two classes was then built using a Neural Network.

The bulk of the work this semester was implementation of our NER tagger to the clinical data to actually get results. It involved parsing raw data, learning appropriate word vectors, checking the validity of the learnt word vectors, extending the binary NER tagger implemented last semester to a multi-class tagger and then eventually combining the the parts together. All this was accomplished successfully and the final result obtained was a F1-score of 0.79 under inexact evaluation. Elaborate details follow in the ensuing chapters.

Contents

List of Figures **v**

List of Tables **vi**

1 Introduction **1**

1.1 The i2b2/VA Challenge 2010 1

1.2 Evaluation metrics 3

1.2.1 Micro-average and Macro-average 3

1.2.2 Exact and Inexact Matching 5

1.3 State-of-the-art 6

1.4 Dealing with Concept Extraction 6

1.5 Structure of the paper 7

2 Neural Networks **8**

2.1 The Structure of the Neural Network 8

2.2 The Backpropogation Algorithm 9

2.2.1 Intuition 10

2.2.2 Learning parameters 10

3 Named Entity Recognition using a Neural Network **11**

3.1 Model Overview 11

3.2 The Feedforward Function 12
3.3 Backpropogation 13
3.4 Results 14

4 Going Forward **16**
4.1 Conclusions from last time 16
4.2 Refining the Corpus 17
4.3 Learning Word Vectors 18
4.3.1 Huang Et Al[6] 18
4.3.2 Word2vec[8] 21
4.4 Validating the word vectors obtained 23
4.5 Concept Extraction 24

5 Conclusion **27**

A Autoencoders **28**
A.1 Structure of the Autoencoder 29
A.2 Hand written digit recognition 30
A.3 Results 31

Bibliography **33**

List of Figures

1.1 Sample clinical text excerpts with its annotated concepts, assertions and relations. Taken from the following i2b2 conference paper [1] 2

2.1 A simple Neural Network with 3 input layer nodes, 3 hidden layer nodes and 1 output node. [3] 9

4.1 An overview of the nueral network model. The model makes use of both local and global context to compute a score that should be large for the actual next word (bank in the example), compared to the score for other words. When word meaning is still ambiguous given local context, information in global context can help disambiguation. [6] 20

4.2 The work flow, from top to bottom, of using the two ways, nameley Huang Et Al[6] and Word2vec[8] of learning word vectors. 22

4.3 Model architectures used. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. [8] 22

4.4 The work flow of the concept extraction system, denoting the input files, the algorithm and the output files from left to right. 26

A.1 A simple Autoencoder with equal number of input and output nodes and 1 hidden layer 30
A.2 Edge-like features learnt by the autoencoder as obtained by me 31
A.3 Penstroke-like features learnt by the autoencoder for the MNIST dataset as obtained by me 32

List of Tables

3.1 Tuning the Neural Network. The best set of parameters are used later for the multi-class NER tagger. 15

4.1 Comparison of results obtained by random initialization and update of given word vectors vs using the given word vectors without updating them. We see that the neural network performs better when the word vectors are updated. 17

4.2 Intersection of clinical data with the pubmed corpus. The idea was to use the learnt word vectors of the words in the intersection and initialize the rest randomly for the NER tagger to learn by itself 23

4.3 F1-scores obtained using the two methods discussed 24

4.4 Concept extraction results using inexact matching and micro-averaging. The F1-training score is the F1 score obtained on the training data and the F1-test score is the F1 score obtained on the test data. 26

Chapter 1

Introduction

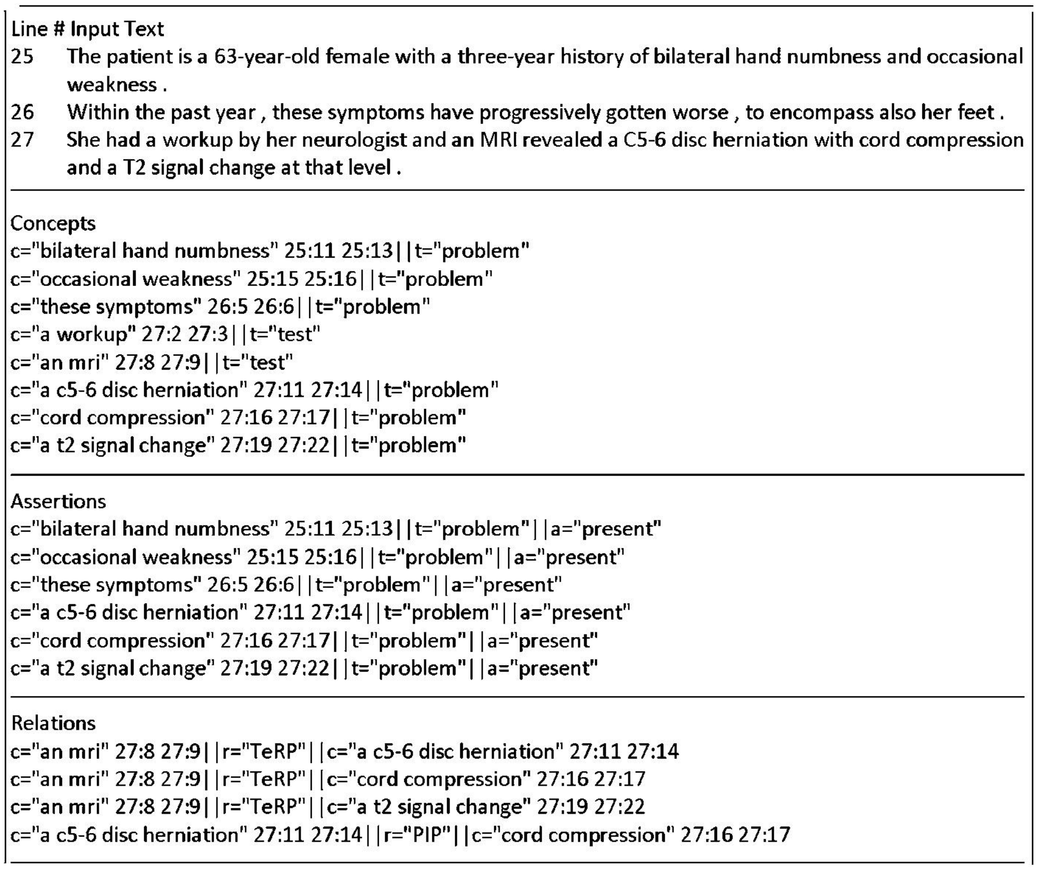
This chapter introduces the details of the problem statement, especially the part about concept extraction as mentioned in the abstract, and the necessary definitions of the machine learning techniques used. It also defines the parameters used for evaluation of machine learning systems. A section also talks about the techniques used so far for solving the problem of concept extraction and their performances.

1.1 The i2b2/VA Challenge 2010

i2b2 is an NIH-funded National Center for Biomedical Computing based at Partners HealthCare System, Boston, Massachusetts.

The 2010 i2b2/VA Workshop on Natural Language Processing Challenges for Clinical Records[1] presented three tasks: *a concept extraction task* focused on the extraction of medical concepts from patient reports; *an assertion classification task* focused on assigning assertion types for medical problem concepts; and *a relation classification task* focused on assigning relation types that hold between medical problems, tests, and treatments. i2b2 and the VA provided an annotated reference standard corpus for the three tasks.

Concept extraction was designed as an information extraction task. Given unannotated text of patient reports, systems had to identify and extract the text corresponding to patient medical problems, treatments, and tests, as shown in Figure 1.1. The most effective concept extraction systems in the i2b2 challenge used conditional random fields (CRFs). It achieved an exact F measure of 0.852 [1] and was significantly different from the rest of the concept extraction systems.



```
Line # Input Text
25    The patient is a 63-year-old female with a three-year history of bilateral hand numbness and occasional
      weakness .
26    Within the past year , these symptoms have progressively gotten worse , to encompass also her feet .
27    She had a workup by her neurologist and an MRI revealed a C5-6 disc herniation with cord compression
      and a T2 signal change at that level .

Concepts
c="bilateral hand numbness" 25:11 25:13||t="problem"
c="occasional weakness" 25:15 25:16||t="problem"
c="these symptoms" 26:5 26:6||t="problem"
c="a workup" 27:2 27:3||t="test"
c="an mri" 27:8 27:9||t="test"
c="a c5-6 disc herniation" 27:11 27:14||t="problem"
c="cord compression" 27:16 27:17||t="problem"
c="a t2 signal change" 27:19 27:22||t="problem"

Assertions
c="bilateral hand numbness" 25:11 25:13||t="problem"||a="present"
c="occasional weakness" 25:15 25:16||t="problem"||a="present"
c="these symptoms" 26:5 26:6||t="problem"||a="present"
c="a c5-6 disc herniation" 27:11 27:14||t="problem"||a="present"
c="cord compression" 27:16 27:17||t="problem"||a="present"
c="a t2 signal change" 27:19 27:22||t="problem"||a="present"

Relations
c="an mri" 27:8 27:9||r="TeRP"||c="a c5-6 disc herniation" 27:11 27:14
c="an mri" 27:8 27:9||r="TeRP"||c="cord compression" 27:16 27:17
c="an mri" 27:8 27:9||r="TeRP"||c="a t2 signal change" 27:19 27:22
c="a c5-6 disc herniation" 27:11 27:14||r="PIP"||c="cord compression" 27:16 27:17
```

Figure 1.1: Sample clinical text excerpts with its annotated concepts, assertions and relations. Taken from the following i2b2 conference paper [1]

1.2 Evaluation metrics

We will evaluate our eventual results using precision, recall, and the F1 measure. These metrics rely on true positives (TP), false positives (FP), and false negatives (FN). In layman's words, precision is the fraction of retrieved documents that were relevant to the search made, where as recall is the percent of all relevant documents that is returned by the search.

Definition 1.2.1.

$$Precision(P) = \frac{TruePositives}{TruePositives + FalsePositives} \tag{1.1}$$

$$Recall(R) = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{1.2}$$

$$F_1 = \frac{2 * P * R}{P + R} \tag{1.3}$$

Remark 1.2.2. True positives are the items correctly labeled and which also belonging to the positive class. False negatives are items which were not labeled as belonging to the positive class but should have been. False positives are items incorrectly labeled as belonging to the class.

For the case of multi-class classification, usual either micro or macro measures are used.

1.2.1 Micro-average and Macro-average

In micro-average method, the true positives, false negatives and false positives are all summed together and then used in equations 1.1, 1.2 and 1.3.

For example, for a set of input data (SET1), let the following values be obtained:

$$TruePositives(TP1) = 10 \tag{1.4}$$

$$FalsePositives(FP1) = 4 \tag{1.5}$$

$$FalseNegatives(FN1) = 2 \tag{1.6}$$

Using equation 1.1, 1.2 and 1.3

$$Precision(P1) = TP1/(TP1 + FP1) = 10/(10 + 4) = 0.71 \tag{1.7}$$

$$Recall(R1) = TP1/(TP1 + FN1) = 10/(10 + 2) = 0.83 \tag{1.8}$$

For another set of input data (SET2), let the following values be obtained:

$$TruePositives(TP2) = 20 \tag{1.9}$$

$$FalsePositives(FP2) = 9 \tag{1.10}$$

$$FalseNegatives(FN2) = 7 \tag{1.11}$$

Using equation 1.1, 1.2 and 1.3

$$Precision(P2) = TP2/(TP2 + FP2) = 20/(20 + 9) = 0.69 \tag{1.12}$$

$$Recall(R2) = TP2/(TP2 + FN2) = 20/(20 + 7) = 0.74 \tag{1.13}$$

Using micro-average techniques, all the true positives, false positives and false negatives will first be summed up and then the precision, recall and F1 scores computed.

$$Precision_{micro} = \frac{TP1 + TP2}{TP1 + TP2 + FP1 + FP2} = \frac{20 + 10}{20 + 10 + 4 + 9} = \frac{30}{43} = 0.69 \tag{1.14}$$

$$Recall_{micro} = \frac{TP1 + TP2}{TP1 + TP2 + FN1 + FN2} = \frac{20 + 10}{20 + 10 + 2 + 7} = \frac{30}{39} = 0.76 \tag{1.15}$$

$$F1_{micro} = \frac{2 * Precision_{micro} * Recall_{micro}}{Precision_{micro} + Recall_{micro}} = \frac{2 * 0.69 * 0.76}{0.69 + 0.76} = 0.72 \tag{1.16}$$

Using macro-average technique, individual precision, recall and F1 scores are computed and there average is taken.

$$Precision_{macro} = \frac{P1 + P2}{2} = \frac{0.71 + 0.69}{2} = 0.70 \tag{1.17}$$

$$Recall_{macro} = \frac{R1 + R2}{2} = \frac{0.83 + 0.74}{2} = 0.79 \tag{1.18}$$

$$F1_{macro} = \frac{2 * Precision_{macro} * Recall_{macro}}{Precision_{macro} + Recall_{macro}} = \frac{2 * 0.70 * 0.79}{0.70 + 0.79} = 0.74 \tag{1.19}$$

For the purposes of the evaluation done in this project, the micro-averaging technique has been used.

1.2.2 Exact and Inexact Matching

The systems developed in the i2b2 challenge[1] were tested for both exact and inexact matching. For a concept extraction system, exact and inexact matching are defined as follows:

Let the phrase "coronary artery bypass graft" be a treatment. When passed into the concept extraction system, if all four words are correctly labelled as a treatment, an exact matching occurs. Otherwise if some word does not get labelled as a treatment, an inexact matching is said to have occurred.

For the purposes of the evaluation done in this project, inexact matching has been used.

1.3 State-of-the-art

In general, all concept extraction systems performed better in inexact evaluation than exact evaluation[1]. The concept extraction systems benefited the most from textual features. Consequently, the best system in this task used a very high dimensional feature space with millions of textual features.

The most challenging examples for the concept extraction systems were abbreviations, for example, CXR for chest x-ray, and descriptive concept phrases, for example, subtle decreased flow signal within the sylvian branches. Our idea to use neural networks and deep learning was to be able to do better in making such predictions as our neural network would take into account the context of adjacent words before making a prediction.

1.4 Dealing with Concept Extraction

Though we have annotated text as training data for learning and implementing our concept extractor, we explored the possibility of training our eventual system on unannotated clinical text. As a lot of recent work on deep neural networks has shown, the unsupervised approach to learning has often fared far better than the supervised approach and also saves precious human resource that go into manually preparing features and training data for supervised learning methods.

Autoencoders have been popular means of learning features from data which are then used for a variety of classification and regression problems.

The idea of working with autoencoders was to enable me to eventually able to implement it if needed for learning features for the NER tagger that we wanted to build.

After finishing work with the autoencoder, we went on to implement a simple NER tagger for two classes which learnt using a supervised method. The codes for both the autoencoder and the NER tagger were implemented in Matlab and Java respectively.

1.5 Structure of the paper

Chapter 2 deals with Neural Networks, which are the underlying building blocks of our algorithms, and explains basic terminologies. References to detailed explainations are provided where needed. Chapter 3 explains the NER system implemented in the odd semester. Chapter 4 covers the work done this semester where word vectors were learned for the vocabulary, the binary NER tagger implemented last semester was extended to a multi-class tagger and the two combined together. The appendix covers some other ideas that we explored during the duration of the project.

Chapter 2

Neural Networks

Neural networks are a way of defining a complex, non-linear form of hypotheses $h_{W,b}(x)$, with parameters W, b that we can fit to our data.

In this chapter we introduce some basic terminology and the workings of Neural Networks, something that we made use of in all the parts of the project. A detailed introduction to Neural Networks can be found at various places on the internet[2]. Neural Networks in themselves are a vast subject and this chapter only covers the bare essentials of how neural networks are structured and more importantly, trained.

2.1 The Structure of the Neural Network

A neural network is an arrangement of layers of nodes, each called a neuron. The output of a neuron in one layer acts as input to another neuron in the next layer. Here is a small neural network:

Layer L1 is called the input layer. L2 is the hidden layer and L3 the output layer. There could be any number of hidden layers in a neural network and any number of nodes in each of the layers.

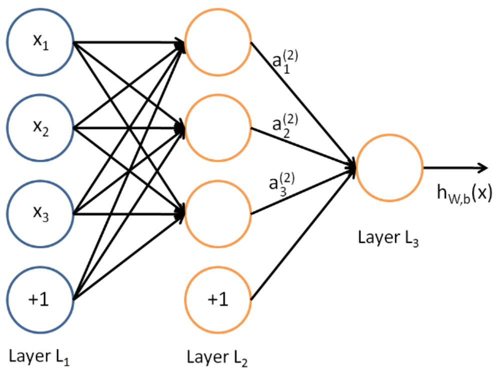


Figure 2.1: A simple Neural Network with 3 input layer nodes, 3 hidden layer nodes and 1 output node. [3]

Definition 2.1.1. The activation function of a node defines the output of that node given an input or set of inputs.

Remark 2.1.2. Usually the sigmoid or the tanh functions are used as activation functions.

2.2 The Backpropogation Algorithm

We train our Neural Network using a fixed training set $(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})$ of m training examples using batch gradient descent. In detail, for a single training example (x, y), we define the cost function with respect to that single example to be the squared error or in some cases the negative of the log-likelihood of our parameters or the binary cross-entropy error. But the main objective remains to compare the output of the neural network y, with the expected output of the neural network $y^{(i)}$ and then try and minimize this error.

2.2.1 Intuition

The intuition behind the backpropagation algorithm is as follows. Given a training example $(x^{(i)}, y^{(i)})$, we first run a forward pass to compute all the activations throughout the network, including the output value of the hypothesis (the hypothesis is just the prediction that our neural network makes). Once we have the prediction, we compute the relative error. Then, for each node in the hidden layer, we would like to compute an error term that measures how much that node was responsible for the error.

2.2.2 Learning parameters

After implementing the neural network cost function and gradient computation, the next step is use an algorithm like SGD (Stochastic gradient descent)[13] to learn a good set parameters.

The idea of SGD, or other similar algorithms, is really simple, instead of minimizing the full cost function J(θ) over the entire dataset, we simply take derivatives over only a single data sample (a single window), and then update all the parameters by taking one single step in the right direction. After sufficient number of iterations, the parameters are learnt.

The detailed gradient computation is very extensive and can be found in various of the references[2]. Also, the entire derivation has been done in the following chapter on NER (Named Entity Recognition) using Neural Networks. The purpose of this chapter was to introduce the intuition and ideas behind the formulation and training of neural networks.

Chapter 3

Named Entity Recognition using a Neural Network

Named Entity Recognition is the ability of a system to parse a piece of text and be able to classify each word as belonging to a certain class. Since the problem of concept extraction is an NER problem, I wanted to be able to implement a basic NER system on which the more elaborate concept extraction system could be built. We were able to implement a Neural Network for NER for 2 classes. The problem statement was from Stanford, who also provided the test and training data and some starter code in Java.

3.1 Model Overview

The detailed problem statement can be found at `http://nlp.stanford.edu/~socherr/pa4_ner.pdf`[5]

Stanford provided a vocabulary of 100232 words along with a 50*1 word vector representation for each of those words. We need these word vectors as input to the neural network can only be in the form of numbers. Stanford's word vectors were learnt using an unsupervised method, something we will need to do for our medical data set as well once we implement a NER for it. A context window of size C was chosen and these C words at a time were fed to the Neural Network. The idea behind choosing a context window was to be able to take into consideration the context of the words while making a prediction. For each context window, the prediction was made for the word at the center.

There was one hidden layer of dimension H, and an output layer consisting of just one node. The hidden layer was used as features for a logistic regression classifier which will return the probability that the center word belongs to either of the two classes. The two classes are PERSON or O (which basically stands for NOT A PERSON).

3.2 The Feedforward Function

$$h_\Theta(x^{(i)}) = g(U^T f(W \begin{bmatrix} x_{i-1} \\ x_i \\ x_{i+1} \end{bmatrix} + b^{(1)}) + b^{(2)})$$

Here W is $H * Cn$ matrix, where H is dimensionality of hidden layer, C is the window size and n is the size of the word vector representation (50 in our case).

U is $H * 1$ matrix comprising of weigths from the hidden layer to the solitary node of the output layer.

$b^{(1)}$ is a $H * 1$ matrix comprising of the bias units to each node in the hidden layer.

$b^{(2)}$ is a scalar and is the bias to the one node in the output layer.

f is the tanh function which is used as activation function for the hidden layer.

g is the sigmoid function used to predict the probability of a word belonging to the PERSON class or not at the output layer.

3.3 Backpropogation

Implementing the neural network involved deriving all the gradients of the cost function to be able to train the neural network using backpropogation. The derivation is stated in this section. The essence of the derivations remain same for all neural networks.

The cost function for our neural network, which is negative of the log-likelihood of our parameters is as follows :

$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}log(h_\Theta(x^{(i)})) - (1-y^{(i)})log(1-h_\Theta(x^{(i)}))] + \frac{C}{2m}[\sum_{j=1}^{nC}\sum_{k=1}^{H}W_{k,j}^2 + \sum_{k=1}^{H}U_k^2]$

We need to find the following gradients : $\frac{\partial J}{\partial U}, \frac{\partial J}{\partial W}, \frac{\partial J}{\partial b}$

For sake of convenience, we define the following terms :

$z_j = \sum_{i=1}^{Cn} W_{ij}x_i + b_j^{(1)}$

$a_j = f(z_j)$

$X = \sum_{j=1}^{H} a_jU_j + b^{(2)}$

$h_\Theta = g(X)$

Differentiating the cost function, we get the following expressions for the gradients :

$\frac{\partial J}{\partial W_{ij}} = x_i(h_\Theta - y^i)\sum_{j=1}^{H} U_j(1-a_j^2) + \frac{C}{m}W_{ij}$

$\frac{\partial J}{\partial U_j} = (h_\Theta - y^i)a_j + \frac{C}{m}U_j$

$\frac{\partial J}{\partial b_j^{(1)}} = (h_\Theta - y^i)\sum_{j=1}^{H} U_j(1-a_j^2)$

$\frac{\partial J}{\partial b^{(2)}} = (h_\Theta - y^i)$

These expressions were then converted to matrix notation which helped in implementing them using the SimpleMatrix Java library. The results from trying different parameter values are elucidated in the next section.

3.4 Results

The exercise[5] mentioned that if trained properly, an F1 score of at least 61% should be obtained. The following tables show the corresponding F1 scores for different values of the parameters. The parameter tuned were :

1. iter : Number of iterations through the trainig set to train the neural network
2. Cw : Windowsize
3. epsW : Random initialization of matrix W in interval $[-epsW, epsW]$
4. epsU : Random initialization of matrix U in interval $[-epsU, epsU]$
5. H : dimensionality of hidden layer
6. α : learning rate
7. C : regularization constant

iter	**Cw**	**epsW**	**epsU**	α	**C**	**F1**
1	**5**	**0.13**	**0.24**	**0.001**	**2**	**0.63**
1	**5**	**0.13**	**0.24**	**0.001**	**4**	**0.62**
1	**5**	**0.13**	**0.24**	**0.001**	**0**	**0.61**
1	**5**	**0.06**	**0.06**	**0.001**	**0**	**0.57**
5	**5**	**0.06**	**0.06**	**0.001**	**0**	**0.65**
5	**5**	**0.13**	**0.24**	**0.001**	**0**	**0.68**
5	**5**	**0.13**	**0.24**	**0.001**	**2**	**0.69**
1	**5**	**0.13**	**0.24**	**0.002**	**2**	**0.61**
1	**5**	**0.13**	**0.24**	**0.0005**	**2**	**0.59**
1	**3**	**0.13**	**0.24**	**0.001**	**2**	**0.43**
50	**5**	**0.13**	**0.24**	**0.001**	**2**	**0.72**

Table 3.1: Tuning the Neural Network. The best set of parameters are used later for the multi-class NER tagger.

Chapter 4

Going Forward

This chapter intends to act as a bridge between the work done last semester and the work done this semester by recalling the conclusions we made last time and elaborating on how each of those issues have been tackled this time. Appropriate discussions of concepts and research papers have been incorporated.

4.1 Conclusions from last time

We enumerate here the conclusions we drew at the end of the work done last semester.

1. Learning word vector representations for the vocabulary : One possible idea in this regard was to randomly initialize word vectors and then update them as well when we train our neural network using backpropogation.

2. Implementing the NER tagger for multiple classes for our clinical data. This would complete the concept extraction task.

4.2 Refining the Corpus

As mentioned in the conclusions last semester, we tried different ways to learn efficient word vectors. Initially for the NER tagger built last semester, learnt word vectors had been provided to us. We extended that work this semester by randomly initializing the word vectors and then training our NER tagger by also updating the word vectors, something we had not done previously. The following results were obtained.

iter	**F1-train**	**F1-test**	**Updating Word Vectors**
1	**0.59**	**0.61**	**No**
1	**0.61**	**0.6**	**Yes**
5	**0.68**	**0.69**	**No**
5	**0.76**	**0.71**	**Yes**

Table 4.1: Comparison of results obtained by random initialization and update of given word vectors vs using the given word vectors without updating them. We see that the neural network performs better when the word vectors are updated.

We got better F1 scores for this model as before implying that the neural network model was going to learn good enough word vectors on its own if trained for sufficiently a long duration.

This theory was also validated by a paper by Huang Et Al, titled Improving Word Representations via Global Context and Multiple Word Prototypes,[6] where a neural network model was used to learn word vectors for the corpus taking into account local as well as global context. The authors also had made available their training code which we used to train our corpus.

But before we used the code from Huang Et Al,[6] we needed to parse our raw corpus and convert it into a form that the code could work with. Python and C++ were used to write code, analyse the properties of the corpus and obtain the desired files. The next section presents a step by step guide as to how the raw corpus was converted to a suitable form, and then how its validity was tested.

4.3 Learning Word Vectors

The corpus, which we will henceforth call the pubmed corpus[10], that we used was a collection of 1,28,958 medical documents and abstracts. It had a total of 4,10,47,872 words and a vocabulary size of 5,14,060.

4.3.1 Huang Et Al[6]

Overview

The paper by Huang Et Al[6] introduces a neural network architecture that learns word vector representations in an unsupervised manner by taking into account both local and global context. The idea behind using global context was to be able to learn better word representations. For example, in a technology-related document, Apple would probably refer to the American multinational whereas in a food-related article, apple would probably refer to the fruit. Taking into account the global context was shown to give better results than existing models in the paper.

Given a word sequence s and document d in which the sequence occurs, the goal was to discriminate the correct last word in s from other random words. A scoring function $g(.,.)$ was defined and scores $g(s,d)$ and $g(s^{\mathrm{w}},d)$ were calculated where s^{w} is the sequence s with the last word replaced by word w. $g(s,d)$ had to be higher than $g(s^{\mathrm{w}},d)$ by at least one.

The scoring function was the sum of the local and global scores, score_l and score_g. The neural network architectures have been represented in Figure

4.1. The local score was calculated using a neural network with one hidden layer and scores as follows:

$$a_1 = f(W_1([x_1; x_2; ...; x_m] + b_1))$$
$$score_l = W_2 a_1 + b_2$$

Here $[x_1; x_2; ...; x_m]$ is the concatenation of m word embeddings of the sequence s, f is an element-wise activation function, $a_1 \in R^{h*1}$ is the activation of the hidden layer with h nodes, $W_1 \in R^{h*(mn)}$ and $W_2 \in R^{1*h}$ are the respective weights of the first and second layers of the neural network with biases b_1 and b_2.

For the global score, the document is listed as a sequence of word embeddings and the weighted average is computed of all the words.

$$c = \frac{\sum_{i=1}^{k} w(t_i) d_i}{\sum_{i=1}^{k} w(t_i)}$$
$$\text{where } w(t_i) = log \frac{N}{d : t_i \in d}$$

N being the total number of documents, and the denominator being all those documents in which t_i occurs. Another two layer neural network similar to the one used to calculate the local score is used to calculate the global score as follows:

$$a_1^{(g)} = f(W_1^{(g)}([c; x_m] + b_1^{(g)}))$$
$$score_g = W_2^{(g)} a_1^{(g)} + b_2^{(g)}$$

Here $[c; x_m]$ is the concatenation of the weighted average document vector and the vector of the last word of the sequence s, f is an element-wise activation function, $a_1^{(g)} \in R^{h(g)*1}$ is the activation of the hidden layer with $h^{(g)}$ nodes, $W_1^{(g)} \in R^{h(g)*(2n)}$ and $W_2^{(g)} \in R^{1*h(g)}$ are the respective weights of the first and second layers of the neural network with biases $b_1^{(g)}$ and $b_2^{(g)}$.

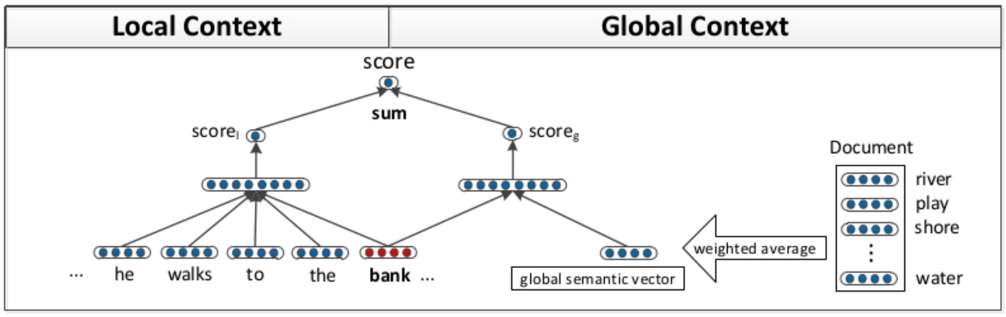


Figure 4.1: An overview of the nueral network model. The model makes use of both local and global context to compute a score that should be large for the actual next word (bank in the example), compared to the score for other words. When word meaning is still ambiguous given local context, information in global context can help disambiguation. [6]

Training

We needed to obtain 3 files from this corpus to be able to train it using the code provide by Huang Et Al[6].

The training code needed 3 input files.

1. A vocab.txt file. As the name suggests, this file was to consist of all the unique words that make up the pubmed corpus. However it had to be in the following format : <word> <word_id> <word frequency in the corpus>. Also, three tokens other than the words in the vocabulary were going to be used namely:

   (a) <s> : a pad for the beginning of a document, for training word windows for words at the beginning of a document.

   (b) </s> : a pad for the end of a document, for training word windows for words at the end of a document.

   (c) eeeoddd : this denotes the end of a document in the corpus files

2. A corpus file. Each word in the pubmed corpus was to be replaced by its corresponding positive word_id with just one id on each line. Since the corpus could be very large, we had to break down the corpus into 50 files each consisting of roughly 8,18,270 words respectively.

3. A document frequency file. The first line of this file was to be the number of documents in the corpus. Every i^{th} line starting from the second line was to have the number of documents in which the word with word_id i-1 appeared.

A python script, using NLTK (a popular natural language toolkit), was written to parse the raw corpus and obtain these 3 files in the specified format. The code took 20 minutes to run. Once we had these files, they were fed into the training code provided by Huang Et Al[6] to obtain word vectors. Tuning the parameters of the training algorithm is a crucial step in any machine learning algorithm and we are tried to find the most optimal set of parameters.

4.3.2 Word2vec[8]

Overview

This was another approach we tried to learn word vectors. The two architectures proposed in this paper were neural network models. The first, Continuous Bag-of-Words model, was similar to the Huang Et Al paper, where the representation of a word was learnt taking into account the context of the words before and after it. The second, Continuous Skip-gram model, makes use of the current word to make predictions about the words before and after it[8].

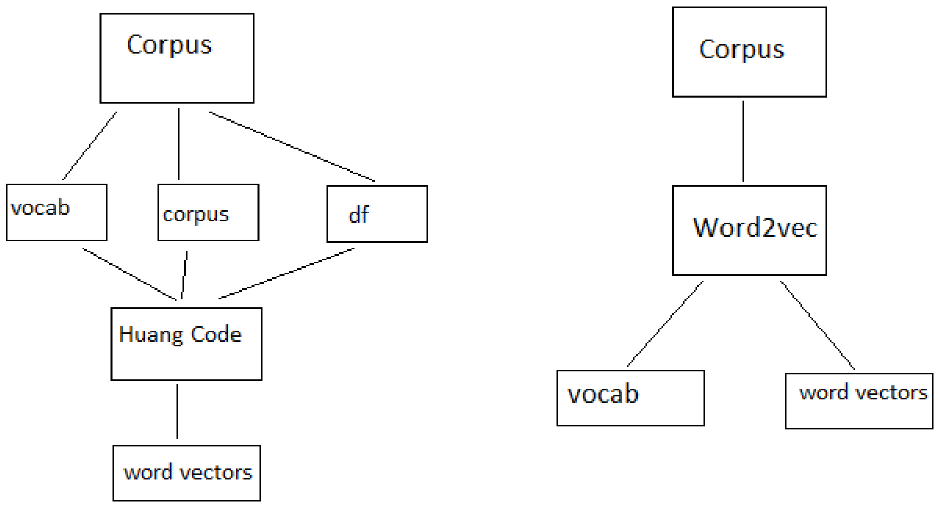


Figure 4.2: The work flow, from top to bottom, of using the two ways, nameley Huang Et Al[6] and Word2vec[8] of learning word vectors.

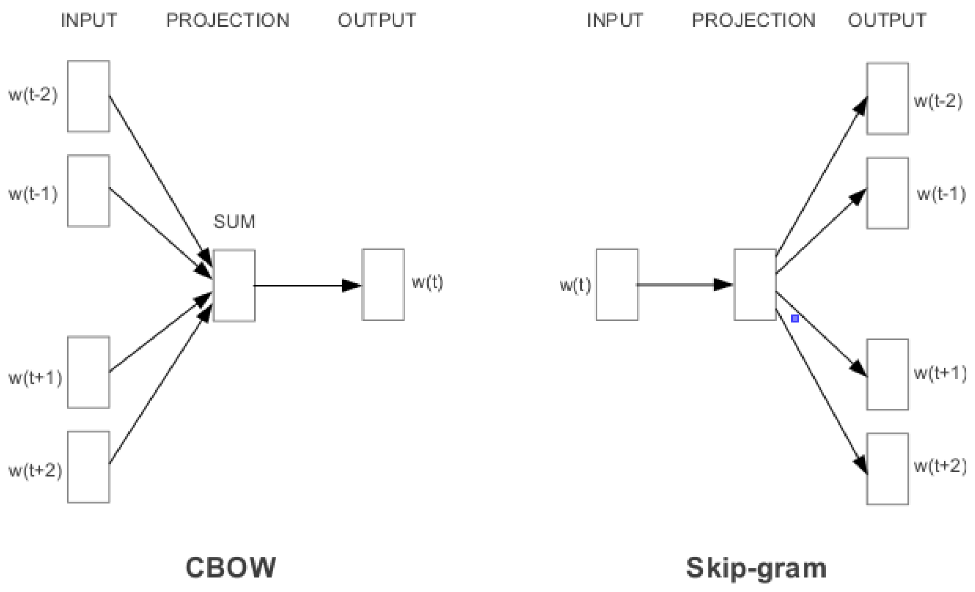


Figure 4.3: Model architectures used. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. [8]

Training

A single-machine multi-threaded C++ code for computing the word vectors using models used in the paper was made available by the authors[9]. This tool, available freely under Apache License 2.0, provides an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing vector representations of words. It takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words.

4.4 Validating the word vectors obtained

Once the word vectors were obtained using the two methods mentioned above, what remained was checking how good a set of word vectors we had learnt. For this purpose, four categories of words from Unified Medical Language System (UMLS)[11] databases were used. These four types of words were *problems, tests, treatments* and *none.* Table 4.2 shows the percentages of the different category words that were found in the pubmed corpus.

Category	In Clinical Data	Found in Pubmed Corpus	Not Found	Percentage Found
test	**1368**	**1311**	**57**	**95.8**
problem	**10299**	**9552**	**747**	**92.7**
treatment	**17768**	**16663**	**1105**	**93.7**
none	**21458**	**20141**	**1317**	**93.8**

Table 4.2: Intersection of clinical data with the pubmed corpus. The idea was to use the learnt word vectors of the words in the intersection and initialize the rest randomly for the NER tagger to learn by itself

For all these words found in the pubmed corpus, 10 nearest neighbours were calculated using cosine similarity.[7] The cosine scores of all nearest neighbours except the word itself belonging to different categories were summed up and the word was assigned the category with the largest cosine sum.

The following results were obtained for Huang Et Al[6] and Word2vec[8] respectively.

	F1 score (Huang)	**F1 score (Word2vec)**
Overall	**0.4**	**0.6**
none	**0.48**	**0.66**
problem	**0.13**	**0.41**
test	**0.002**	**0.13**
treatment	**0.44**	**0.61**

Table 4.3: F1-scores obtained using the two methods discussed

Since the word2vec tool was found to give better word representations, we went ahead with using it to learn word vector representations for our concept extraction system.

4.5 Concept Extraction

Since by this time, we have learnt and used ways to learn suitable word vectors, we were finally ready to piece together the two major parts of the project. The learnt word vectors will now be used with the NER tagger implemented in the previous semester to finally build a system that takes as input discharge summaries and outputs the various concepts and their categories.

As described in the previous sections, we had learnt decent word reprsentations for the vocabulary size of 5,14,060 in the pubmed corpus. The vocabulary from this corpus, being a collection of medical documents and abstracts, was bound to have an intersection with our clinical discharge summary corpus. The clinical data that we had was found to have a vocabulary of size 15479[12]. Of these, 11196 (72.33%) were found in the pubmed corpus. This meant that we could use the learnt word vectors of these 11196 words and initialize the others randomly for our NER tagger. The random initialization had been tested to work well and thus this should give decent results.

The NER tagger implemented last semester worked on 4 input files.

1. vocab.txt : This file comprised of all the word in the vocabulary.
2. wordVectors.txt : The ith line in this file corresponds to the vector representation of the word at the ith line in the vocab.txt file.
3. train : This was the training file, where each line consisted of a word and its correspoding tag.
4. dev : This was the test file, in a format similar to that of train. This file would be used to test the parameters that the neural network had learnt.

These files were obtained from the provided clinical data corpus. The clinical data corpus provided had 170 medical discharge summaries along with the annotations of the 3 classes, namely problem, test and treatment. Every word not in any of these classes was classified as none.

Another important part was extending the binary class tagger implemented last semester to a multi-class tagger. The output layer of the neural network implemented last semester had just one node. It was extended to 4 nodes (one each for the classes none, test, treatment and problem). The derivations were updated accordingly and the following results were obtained.

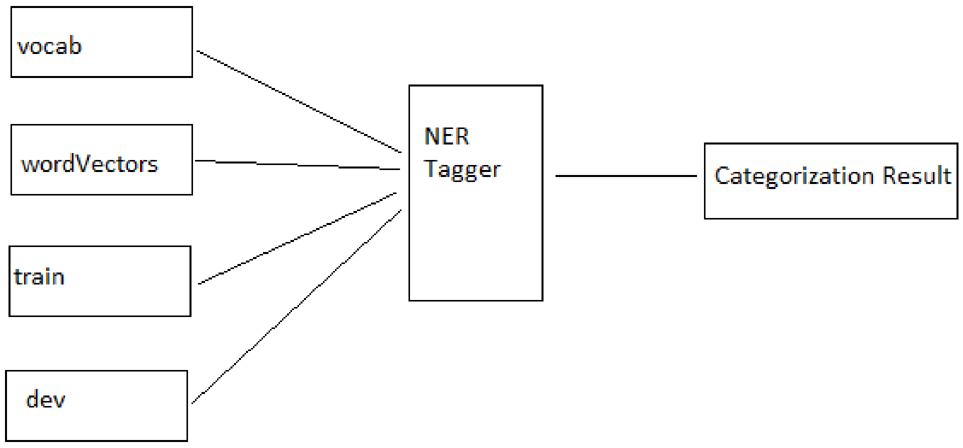


Figure 4.4: The work flow of the concept extraction system, denoting the input files, the algorithm and the output files from left to right.

iter	**Cw**	**epsW**	**epsU**	α	**C**	**F1-training**	**F1-test**	**Updating Word Vectors**
1	**5**	**0.13**	**0.24**	**0.001**	**3**	**0.91**	**0.78**	**No**
1	**5**	**0.13**	**0.24**	**0.001**	**3**	**0.96**	**0.78**	**Yes**
5	**5**	**0.13**	**0.24**	**0.001**	**3**	**0.92**	**0.79**	**No**
5	**5**	**0.13**	**0.24**	**0.001**	**3**	**0.98**	**0.77**	**Yes**
10	**5**	**0.13**	**0.24**	**0.001**	**3**	**0.94**	**0.78**	**No**
10	**5**	**0.13**	**0.24**	**0.001**	**3**	**0.99**	**0.77**	**Yes**

Table 4.4: Concept extraction results using inexact matching and micro-averaging. The F1-training score is the F1 score obtained on the training data and the F1-test score is the F1 score obtained on the test data.

Chapter 5

Conclusion

We set out to try and solve the problem of concept extraction using Deep Learning. We broke down the problem into different parts. The two major parts were learning word vector representations for the vocabulary of the corpus, and implementing a NER tagger to finally classify words into one of four categories. Since we are dealing with very large files and computationally intense algorithms, training was a little time consuming. The final results of the concept extraction have been obtained only after training on 88647 words. Training on a larger corpus is expected to give better results.

Appendix A

Autoencoders

Supervised learning is one of the most powerful tools of AI, and has led to automatic zip code recognition, speech recognition, self-driving cars, and a continually improving understanding of the human genome. Despite its significant successes, supervised learning today is still severely limited. Specifically, most applications of it still require that we manually specify the input features x given to the algorithm. Once a good feature representation is given, a supervised learning algorithm can do well. But in such domains as computer vision, audio processing, and natural language processing, there are now hundreds or perhaps thousands of researchers who have spent years of their lives slowly and laboriously hand-engineering vision, audio or text features.

While much of this feature-engineering work is extremely clever, one has to wonder if we can do better. Certainly this labor-intensive hand-engineering approach does not scale well to new problems; further, ideally we would like to have algorithms that can automatically learn even better feature representations than the hand-engineered ones. We worked with the sparse autoencoder learning algorithm, which is one approach to automatically learn features from unlabelled data. In some domains, such as computer vision, this approach is not by itself competitive with the best hand-engineered features, but the features it can learn do turn out to be useful for a range of problems (including ones in audio, text, etc). We solved a problem of recognizing hand-written digits using our sparse autoencoder. We build from the previous chapter which discusses the workings of a Neural Network to describe our sparse autoencoder.

A.1 Structure of the Autoencoder

An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs. I.e., it uses $y^{(i)} = x^{(i)}$. Since it is bottlenecked on the number of nodes in the hidden layers, the hidden layers thus learn important features about the input data.

The autoencoder tries to learn an approximation to the identity function. The identity function seems a particularly trivial function to be trying to learn; but by placing constraints on the network, such as by limiting the number of hidden units, we can discover interesting structure about the data.

Even when the number of hidden units is large (perhaps even greater than the number of input pixels), we can still discover interesting structure, by imposing other constraints on the network. In particular, if we impose a sparsity constraint on the hidden units, then the autoencoder will still discover interesting structure in the data, even if the number of hidden units is large. Informally, we will think of a neuron as being active (or as firing) if its output value is close to 1, or as being inactive if its output value is close to 0. We would like to constrain the neurons to be inactive most of the time.

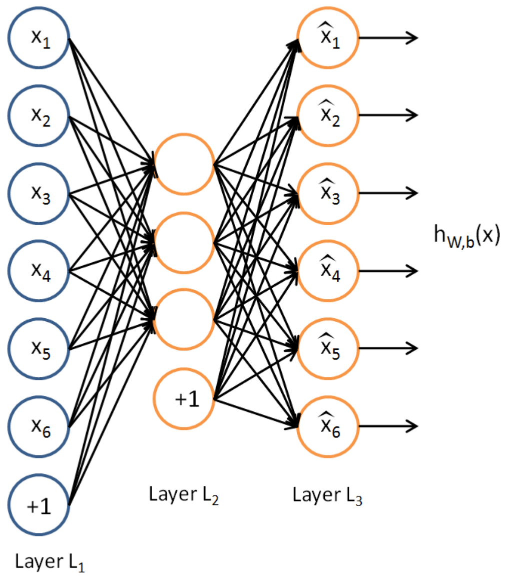


Figure A.1: A simple Autoencoder with equal number of input and output nodes and 1 hidden layer

A.2 Hand written digit recognition

The purpose of this exercise was to implement an autoencoder that could learn edge like features which could then be used to predict samples of hand written digits. This was another one of many exercises Stanford's ML group provides for free. The autoencoder was trained on patches of images. Each patch was treated as a matrix comprising of the pixel intesity values. The starter code to sample these patches was provided. We went onto implement the backpropogation learning algorithm and then visualize the results obtained as shown in the next section. The optimization algorithm used for training was L-BFGS (Limited memory BFGS) [4].

A.3 Results

After training the autoencoder, decent results were obtained, including the edge-like features shown.

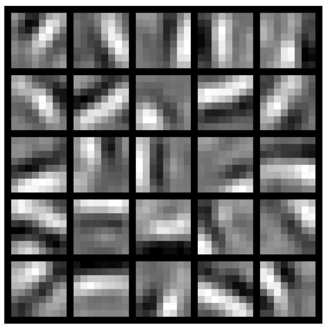


Figure A.2: Edge-like features learnt by the autoencoder as obtained by me

Once the edge-like features were learnt, the autoencoder was run on 28*28 patches from the MNIST data set to obtain the following pen stroke like features.

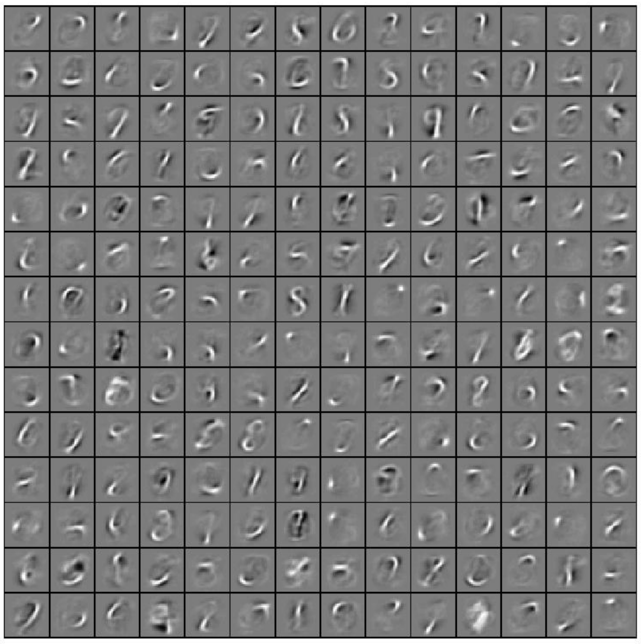


Figure A.3: Penstroke-like features learnt by the autoencoder for the MNIST dataset as obtained by me

Bibliography

[1] Ozlem Uzuner, Brett R South, Shuying Shen, Scott L DuVall. (2010), “2010 i2b2/VA challenge on concepts, assertions, and relations in clinical text”.

[2] Learning from Data, a Caltech course. http://work.caltech.edu/telecourse.html

[3] Andrew Ng, “Sparse Autoencoders”, www.stanford.edu/class/cs294a/sparseAutoencoder.pdf

[4] Limited memory BFGS, http://en.wikipedia.org/wiki/Limited-memory_BFGS

[5] Stanford Problem statement for Named Entity Recognition for 2 classes, http://nlp.stanford.edu/~socherr/pa4_ner.pdf

[6] Eric H. Huang and Richard Socher and Christopher D. Manning and Andrew Y. Ng, Improving Word Representations via Global Context and Multiple Word Prototypes, ”Annual Meeting of the Association for Computational Linguistics (ACL)”, 2012

[7] Enhancing clinical concept extraction with distributional semantics Siddhartha Jonnalagadda, Trevor Cohen, Stephen Wu, Graciela Gonzalez

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013.

[9] Word2vec. A tool to compute continuous distributed representations of words. `https://code.google.com/p/word2vec/`

[10] PubMed is a free search engine accessing primarily the MEDLINE database of references and abstracts on life sciences and biomedical topics. `http://www.ncbi.nlm.nih.gov/pubmed`

[11] United Medical Language System `http://www.nlm.nih.gov/research/umls/`

[12] i2b2 Challenge Data Sets `https://www.i2b2.org/NLP/DataSets/Main.php`

[13] Stochastic Gradient Descent `http://en.wikipedia.org/wiki/Stochastic_gradient_descent`