**NLP SEE Evaluation - Group N3**

**Objective: To develop a fine grain sentiment analyzer using Stanford Sentiment Tree Bank Dataset**

**Input Files**:

We downloaded raw datasets from Stanford’s link as given in the question paper.

Overview of dataset:

1. Sentiment\_labels.txt : Contains sentence along with the index
2. Dictionary.txt : Contains index along with a decimal number

For 3-class classification:

|  |  |  |  |
| --- | --- | --- | --- |
| **Decimal Number** | **<0.4** | **0.4-0.6** | **>0.6** |
| **Class** | 1 | 2 | 3 |
| **Class label** | Negative | Neutral | Positive |

For 5-class classification:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Decimal Number** | **0.0-0.2** | **0.2-0.4** | **0.4-0.6** | **0.6-0.8** | **0.8-1.0** |
| **Class** | 1 | 2 | 3 | 4 | 5 |
| **Class label** | Very Negative | Negative | Neutral | Positive | Very Positive |

Raw dataset obtained (dataset\_three.csv and dataset\_five.csv):

1. Both the CSV files contained data in two columns.

2. The first column indicated the phrase

3. The second column indicated the class it belongs to.

4. dataset\_three.csv contained class labels 1,2,3 representing classes as mentioned above.

5. dataset\_five.csv contained class labels 1,2,3,4,5 representing classes as mentioned above.

**Preprocessing:**

1. The pickle files for training and testing were created, using the dataset downloaded, after cleaning the data.   
 2. Cleaning dataset involved removing hashes, dollar symbols and other random characters .

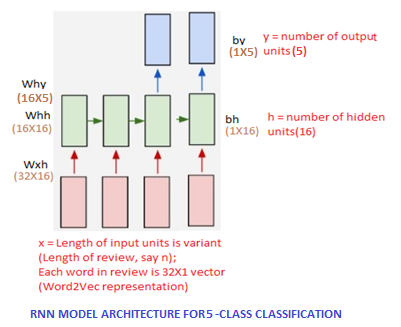
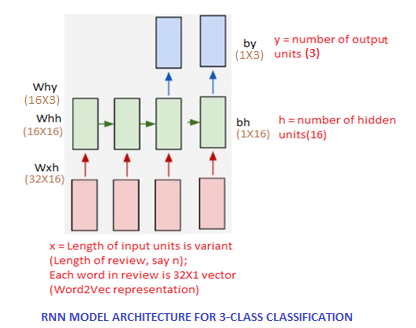
3. get\_data.py separates the data according to the respective class labels and dumps the data into .pkl files.

4. word2vec.py converts the words in each sentence (dumped in .pkl files previously) to vector of length 32 using gensim model.

5. Size of Train And Test set were 80% of total file size and 20 % of total file size, repectively.

**Design1 : Vanilla RNN**

Each Stanford Data is trained and tested sentence by sentence using vanilla RNN.   
In this case, we will have one RNN model. Basically, we would have trained weight matrix and bias vector between hidden and input, weight matrix between hidden and hidden, weight matrix and bias vector between hidden and output.   
The input given to RNN is a sentence (arbitrary length with words represented as vectors) and output to the RNN is a vector with size (1X3) which represents whether the classified result was ( Positive, Neutral, Negative) for 3-class classification



In the above figures,

1. black color text represents the parameters of the model used

2. brown color text represents the dimensions

3. red color text represents the model

Hyper parameters:

1) Number of units in each layer:

- Input: Length of sentence; variant;

- Word representation: 32X1 vector

- Hidden : 16 units

- Output : 2 units (First unit represents negative and second unit represents positive)

2) Learning rate: 1e-1

3) Epochs: We tried with 10, 25 and 1000

**Functions:**

Input : (32X1) vector whose size is number of sequence of words in a sentence

Hidden (Forward): tanh function with (Wxh.x + Whh.h(prev) + bh) as its parameter

Output: Softmax function with (Wyh.[h(forward)]+by) as its parameter. Contains probability distribution of two classes (positive/negative)

Procedure:

1. For training, we passed the sentences in shuffled order (class1 data, class2 data, class3 data, class4 data, class5 data). Each sentence has a set of words. Each word was converted to a 32 sized vector using gensim's Word2vec model and passed as input for rnn. The target was 0 for negative, 1 for neutral and 2 for positive class for a 3-class classifier or was 0 for very negative,1 for negative, 2 for neutral, 3 for positive class and 4 for very positive for a 5-class classifier.

2. For each such input, forward propagation and backward propagation was done as explained in the functions above. Here, we want to generate the output only after the entire review is read, so, if the sentence length is t, output is generated at (t+1)th time step. Also, during backward propagation, we copy the probabilities at last time step only as we do not require it after that, for the reason mentioned above.

3. After the weights were updated, this model was used to test data for new sequence of lists of sentences.

Building the model:

In the forward propagation,

1. h(forward) is evaluated from left to right

2. Output, o is evaluated (we evaluated it from left to right).

Error propagation in backward propagation:

1. Errors were propagated from output to hidden using first order differential function of softmax.

2. Errors were propagated from hidden to input from right to left, using first order differential function of tanh.

3. Weights were updated.

Process continues till loss gets minimized beyond a threshold

**Design 1.1 :Enhancement Using GRU on RNN**

We replaced the hidden layer generated from the previous layer with the GRU Units

We incorporated this using the following equations:

Variables

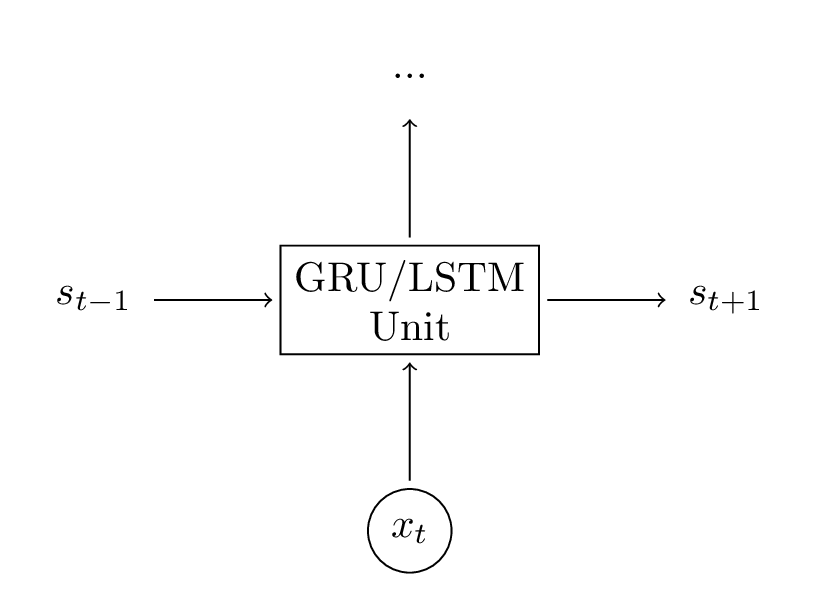
x t {\displaystyle x\_{t}}  x(t): input vector

 h t {\displaystyle h\_{t}} h(t): hidden vector

 z t {\displaystyle z\_{t}} z(t): update gate vector

 r t {\displaystyle r\_{t}} r(t): reset gate vector

 W {\displaystyle W} U, W, Ur , Uz, Wr and Wz areU {\displaystyle U} b {\displaystyle b} parameter matrices and bz, br are bias vectors.



**Procedure:**

**Model Parameters to be trained:**

Initially all the weight functions are random.

The model parameters are:

Wxh – Weight function btw input and hidden

Whh – Weight function between hidden and hidden

Wz - Weight matrix for update gate, between input and hidden

Uz - Weight matrix for update gate between hidden and output

Wr - Weight matrix for reset gate, between input and hidden

Ur - Weight matrix for reset gate, between hidden and output

W - Weight matrix to calculate new memory content, between input and hidden

U - Weight matrix calculate new memory content, between hidden and output

Why –Weight function between hidden and output

bh- hidden unit bias

by- output unit bias

These parameters are changed for every iteration and learning is done on the training data . Performed training for 10 epochs.

**Test Process:**

**Forward Propagation:**

Output is the probability distribution .We rank the probability distributions in the decreasing order.

The class with highest probability becomes the class label.

**Backward Propagation:**

This is used to find the errors in the RNN which are responsible for loss in the final output.

**Visualization:**

Stanford CoreNLP provides a set of natural language analysis tools.

It can give the base forms of words, their parts of speech, whether they are names of companies,

people, normalize dates, times, and numeric quantities, mark up the structure of sentences in

terms of phrases and word dependencies, indicate which noun phrases refer to the same entities,

indicate sentiment, extract particular or open-class relations between mentions.

The Stanford CoreNLP parser is used to get the parse tree structure of the sentence.

The sentence to be parsed is present in input.txt file inside the CoreNLP directory.

The sentence is parsed usind the command:

**java -cp "\*" -Xmx2g edu.stanford.nlp.pipeline.StanfordCoreNLP -annotators tokenize,ssplit,pos,lemma,ner,parse,dcoref -file input.txt**

This command tokenizes, gives pos tagging to the words, named entity recognition and parses the sentence to a tree format.

The results will be saved in **input.txt.xml** file in the same directory.

We are using et3 library to visualize the parse tree.

To parse the xml file and get the structure of the parse tree we are using minidom of xml.dom library.

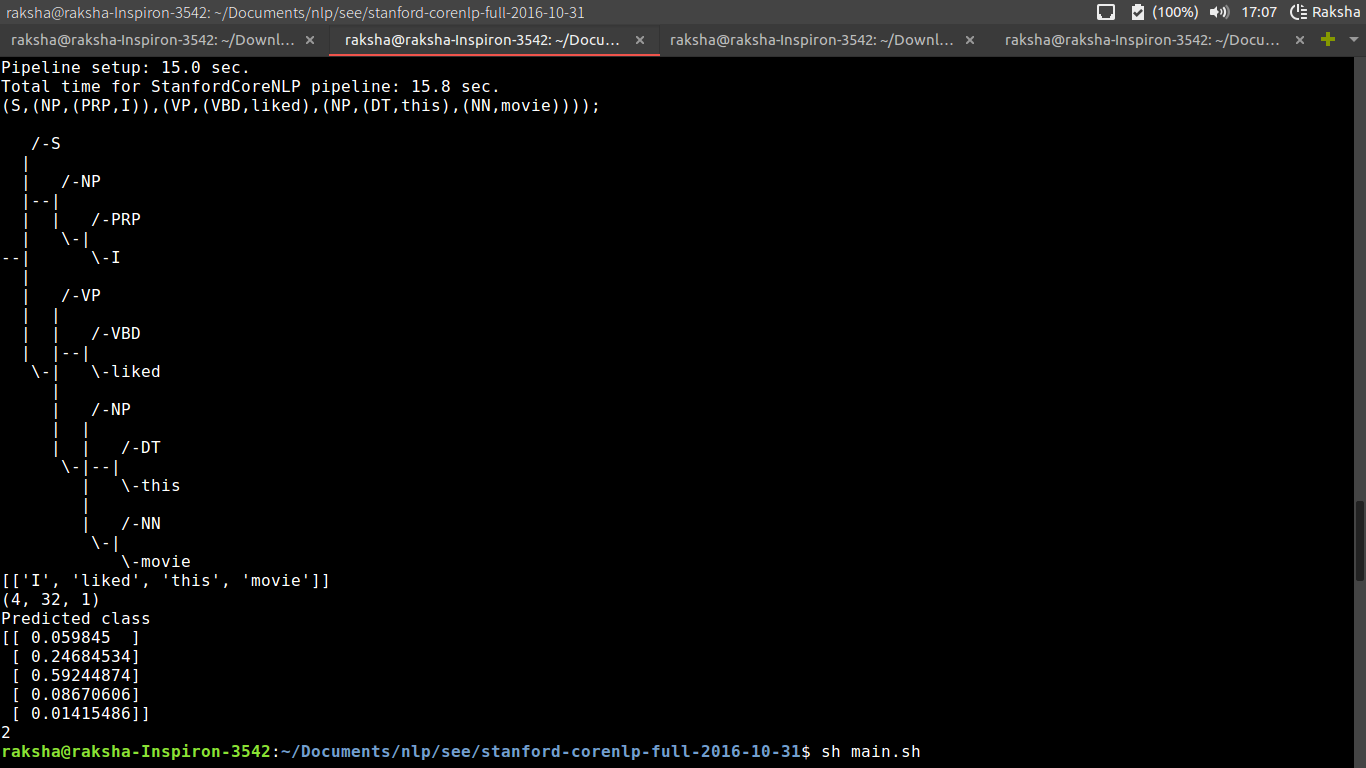
The tree structure is present in between the tags <parse> in the xml file.

**(ROOT (S (NP (NN today)) (VP (VBD was) (NP (DT the) (JJS best) (NN day)) (ADVP (RB ever)))))**

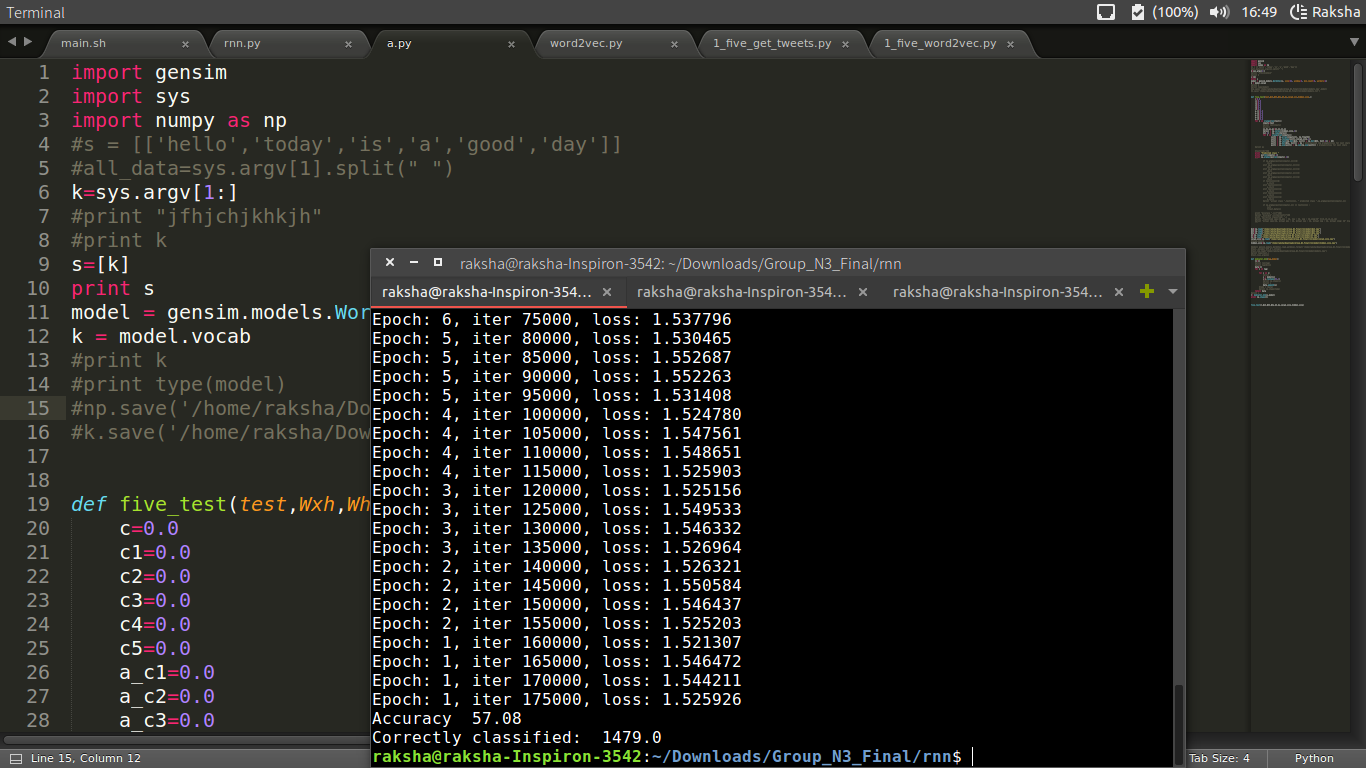
This tree structure is now processed to get the tree structure for the et3 library.

The Tree method of the library is called to visualise the tree.

**Fig: Stanford ParseTree for sentence**

****

**RNN Accuracy Results:**

****

**Convolutional Neural Networks**

**Input:**

We used the same dataset as RNN for CNN .

**CNN Architecture:**

We used three bigram filters, three trigram filters and three four-gram filters .

Window size for bigrams is 2, for trigrams is 3 and four-grams is 4.

The first layer is the convolution layer, followed by a max-pool layer and a softmax layer.

Parameters:

bigram\_filter1: ( vocab\_size\*2X1)

bigram\_filter2: ( vocab\_size\*2X1)

bigram\_filter3: ( vocab\_size\*2X1)

trigram\_filter1: ( vocab\_size\*3X1)

trigram\_filter2: ( vocab\_size\*3X1)

trigram\_filter3: ( vocab\_size\*3X1)

fourgram\_filter1: ( vocab\_size\*4X1)

fourgram\_filter2: ( vocab\_size\*4X1)

fourgram\_filter3: ( vocab\_size\*4X1)

Convolution Layer:

Bigram:

t=np.concatenate((inputs[i]),axis=0)

o = np.tanh(np.dot(t.T,filter))+bias

Trigram:

t=np.concatenate((inputs[i],inputs[i+1]),axis=0)

o = np.tanh(np.dot(t.T,filter))+bias

Fourgram:

t=np.concatenate((inputs[i],inputs[i+1], inputs[i+2]),axis=0)

o = np.tanh(np.dot(t.T,filter))+bias

Max-Pool Layer:

Return max{ois}

Softmax layer:

t = np.dot(np.asarray(input).T,Why)+by

k= np.exp(t) / np.sum(np.exp(t))