Natural Language Processing

Subhasish Basak (MDS201803)

Assignment 4 : Sentiment Analysis

April 19, 2020

1 Introduction

In this Assignment we discuss the implementation & design of an **ANN** for *Sentence wise fine grained* **Sentiment Analysis**. Given our implementation of the RNN in Assignment 3 which is used to predict the sentiment of a movie review, now we should also be able to detect if there is any bias in the labelled reviews. The main motivation of the task is discussed as below,

- Our implementation should be able to estimate the *Polarity* of each sentence present in the reviews.
- Using the individual polarities of the sentences, we compute a *Consolidated Polarity* which signifies the polarity of the review.
- We compare the *Consolidated Polarity* of each review with its *Labelled Polarity* to find if there is any bias in the review.

Moreover here we are concerned with *Fine-grained* sentiment analysis. Thus unlike Assignment 3 we are no more restricted to only 2 sentiment classes. The typical breakdown of fine-grained sentiment uses five discrete classes, as shown below.

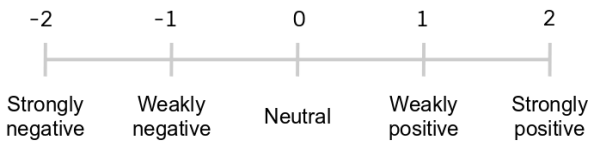

Figure 1: polarity classes

2 Data Preprocessing

We are working with the same **IMDB** movies review dataset. The labelled data is **not** fine grained, both the training and testing part contains 25000 reviews half of which are positive and half negative. We assume that we have a *list of sentiment words* as an aid to identify the sentiment of a sentence. We pre-process the data using the following steps (The first 3 steps are almost the same as Assignment 3):

- **Removing punctuation & symbols** : This step includes removing unecessary punctuation and symbols present in the reviews.Since we are intereted in the english words only we preprocess the data to remove them. Regular Expressions are used for the purpose. Also all the alphabets are converted into lower cases.
- **Removing the stopwords** : Words having too high or too low frequencies are of no use in our study. Most commonly occuring words like "the", "a" etc are present in all the reviews and contribute nothing in our analysis. Similarly there are some rare words also which have no discriminating power. We remove all such words.

- **Lemmatizing & Stemming the words** : Lemmatizing & Stemming are essential part of data preprocessing. This helps in reducing the vocabulary size.
- **Breaking into sentences** : Since we are interested in *Sentence wise* sentiment analysis, we need to split each review into sentences.
- **Labelling the sentences** : As mentioned earlier in our dataset each of the reviews are labelled either 0 or 1 as polarity. But for our purpose we need to label each sentence in the reviews into 5 polarity classes. Given the list of sentiment words we classify each sentence in the training reviews in the 5 polarity classes.
- **Resizing the Sentences** : In order to input the sentence vectors in a RNN we need all of them of the same size, i.e. this sequence length is same as number of time steps for the RNN. To deal with both short and long sentences, we will pad the small ones and truncate all the large reviews to a specific length. We denote this length by **sentence_length**. For sentences shorter than sentence_length, we will pad with 0s. For reviews longer than sentence_length we will truncate them to the first sentence_length words.
- **Tokenizing the words** : After the preprocessing till this step we tokenize each unique word in our vocabulary. All the sentences are then encoded with these tokens. So now we are left with list of sentences in which each word is represented by a perticular word index (token number).

3 The ANN Architecture

Here we start with a **Many to One** Recurrent Neural Network**(RNN)** framework with **Long Short Term Memory** (LSTM) layer. Our Neural network has the following configuration :

- Input Layer
  - **Number of nodes** : sentence_length
  - Corresponding to each time-step ($t = 1(1)sentence\_length$) there is the token corresponding the english word from the sentence.
- Embedding Layer
  - **Number of nodes** : vocab_size
  - This layer converts our word tokens (integers) into embedding equals to vocabulary size. Unlike previous assignment here we let the model learn its embedding by its own.
- LSTM Layer
  - **Number of nodes** : Accordingly as the embedding size and the hidden layer dimensions
  - This LSTM is the imporvement over the last implementation in Assignment 3. The LSTM layer solves the *Problem of Long-Term Dependencies* (viz. vanishing or exploding gradients).
- Hidden Layer
  - **Number of nodes** : Hyperparameter subject to tuning
  - This layer maps output of LSTM layer to a desired output size.
- Softmax Layer
  - Transforms the output values from the hidden layer into probabilities.
- Output Layer
  - **Number of nodes** : 1
  - Softmax output from the last timestep is considered as the final output of this network. This corresponds to the maximum probability.

Our aim is to classify each sentence to two the 5 polarity classes, i.e. given a *sentence_length* $\times$ 1 input vector our Neural Network should compute the probability vector (a 5 $\times$ 1 vector representing the probability of the review falling into the sentiment classes) and return the class corresponding the maximum probability.

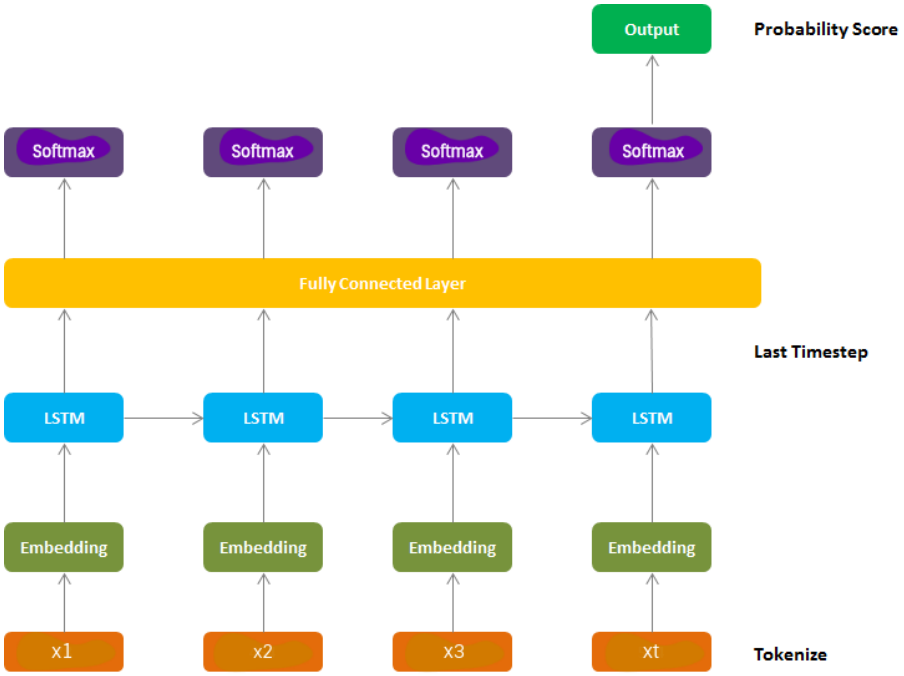

Figure 2: The ANN architecture

3.1 The LSTM Layer

The traditional RNN suffers from problems like vanishing or exploding gradients. These issues are the main motivation behind the LSTM model which introduces a new structure called a **memory cell** .

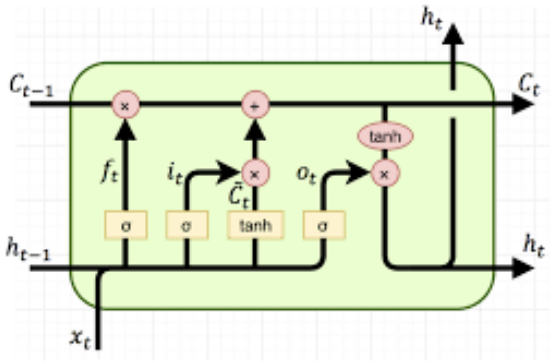

Figure 3: LSTM memory cell

A memory cell is composed of four main elements:

- **an input gate** : decides which values we'll update
- **a neuron with a self-recurrent connection** : this is the key to LSTM s is the cell state, the horizontal line running through the top of the diagram.
- **a forget gate** : decides what information we're going to throw away from the cell state

- **an output gate** : decides the state of the memory cell to have an effect on other neurons or prevent it.

Again the weights and biases of the LSTM layers are also learnt through the *Back Propagation Through Time* algorithms.

4 Input & Training the model

- **Input** : Our RNN takes input vectors of tokens, of size $sentence\_size \times 1$. Each element of the vector is a token corresponding to the English word in the sentence. As mentioned earlier the ANN takes review sentences as inputs and outputs the most probable polarity class.
- **Training** : We use the labelled sentences from the training reviews obtained during data pre-processing. Each sentence is assigned one of the 5 polarities. We train over all the sentences from the 2500 movie reviews.

5 Hyperparameters

In our model we have considered the following Hyperparameters.

- **Sentence_length** : This is the length to which each sentence is truncated at or padded to. An increase in the *Sentence_length* parameter will definitely incorporate more information about the sentence and on the other hand it will increase computation complexity as well.
- **Vocab_size** : This is the number of unique words in our vocabulary we are working with. Like previous Assignmnet we have restricted our *Vocab_size* to 10000 to reduce the time complexity. As we increase the *Vocab_size* time complexity increases but it also increases accuracy.
- **Hidden layer size** : The number of nodes in the hidden layer is the next crucial hyperparameter. Increasing number of hidden layer nodes increases the training accuracy to some point but after that it starts decreasing. For our previous Assignment we got the optimal hidden layer size as 20 .

6 Computation of consolidated polarity and bias

Using our **ANN** we compute the polarity of each of the sentences of all the 2500 test reviews. Let **R** be a perticular review which has **k** sentences viz. $s_1, s_2, \ldots s_k$. Also let their corresponding polarities be $p_1, p_2, \ldots p_k$, where each p_i, $i = 1(1)k$ belongs to any one of the discrete polarity classes, i.e. they take the values from the set $[-2, -1, 0, 1, 2]$. To compute the consolidated review corresponding **R**, we propose the following formula,

$$c = \frac{1}{k} \sum_{i=1}^{k} p_i \tag{1}$$

Thus the consolidated polarity is nothing simple weighted mean of the polarities of the sentences present in that review.

6.1 Detecting Bias

In this context, if there are more sentences with negative sentiments for a review and if the label was positive, then we say that there is a **bias** in the review.
For all the reviews we have the original label $\boldsymbol{l}$ (0 or 1 respectively for positive or negative). Now let **c** be its consolidated review as computed by equation (1) and using our RNN implementation. We classify a review **R** as biased if either,

- $l = 0$ and $c <= 0$
  - i.e. the True label is *positive* and the consolidated label is *negative*.

  OR
- $l = 1$ and $c > 0$
  - i.e. the True label is *negative* and the consolidated label is *positive*.