STCS Assignment Phase-II

Performing Sentiment Analysis on word vectors

SUBMITTED BY - Vitthal Bhandari 2017A7PS0136P

SUBMITTED TO - Dr. Poonam Goyal (CSIS Dept., BITS Pilani)

Prepared in partial fulfilment of the course

SEL TOPICS FROM COMP SC (CS F441)
Ist Semester 2020-21

Table of Contents

1. Dataset Used

The dataset that has been used in this project is the Movies Review dataset provided beforehand along with the problem statement. It is a set of four files - '*Dictionary.txt', 'Senti_scores.txt'*, *'Train-Dev_test split.txt'* and *'Movie Review Dataset.txt'*. The last file contains movie reviews without any alteration. The first file contains a dictionary of phrases of varying lengths generated from these reviews, with each of them mapped to an index. The second file provides a mapping between the phrase index and corresponding score ranging from $[0,1]$. The third file provides the split for training, validation and test set.

2. Data Preprocessing

Once all the data from '*Dictionary.txt'* and *'Senti_scores.txt'* has been loaded, we begin with the task of making the data usable. First, the problem is converted to a multi-class classification problem by converting all the scores in the range of $[0,0.2]$, $(0.2,0.4]$, $(0.4,0.6]$, $(0.6,0.8]$ and $(0.8,1.0]$ to class labels 0, 1, 2, 3 and 4 respectively. Next, the phrases need to be processed so as to make them suitable for ingestion by the neural network. This is done by using the nltk library for data preprocessing. The various steps involved are:

a. Alphabetization

All characters in the corpus, except for the English alphabets (a-zA-Z), are removed by employing the regex functionality of python. Thus the corpus is left only with the English alphabets. The library used is *re*.

b. Tokenization

It is the process of splitting sentences into words. The entire corpus was split into words and then these individual words (called tokens) were further sent for preprocessing. The class used is *nltk.tokenize*.

c. Lower casing

The individual tokens are converted to lowercase so that only the characters a-z appear in the corpus. This makes it easy for the Neural Network to learn the model parameters.

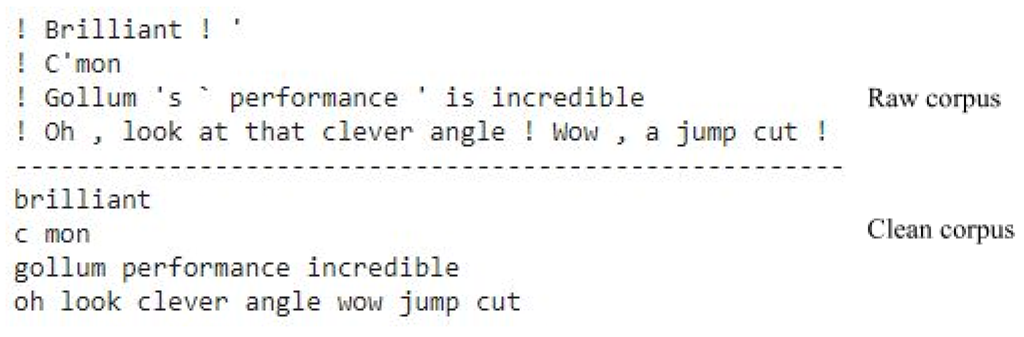
d. Stop words removal

Using a list of very commonly used stopwords (such as ‘a’, ‘an’, ‘the’ etc.) offered by nltk, the corpus is filtered to remove these words as they occur very frequently and do not contribute significantly to the sentiment. The class used is *nltk.corpus*.

e. Lemmatization

It is the process of reducing the token to its base form. For e.g. cars → car. This way most word variations of a particular word can be avoided. Thus the network has to learn only one representation for a particular word and its variations. The class used is *nltk.stem*.

After preprocessing, the phrases showed the following difference:



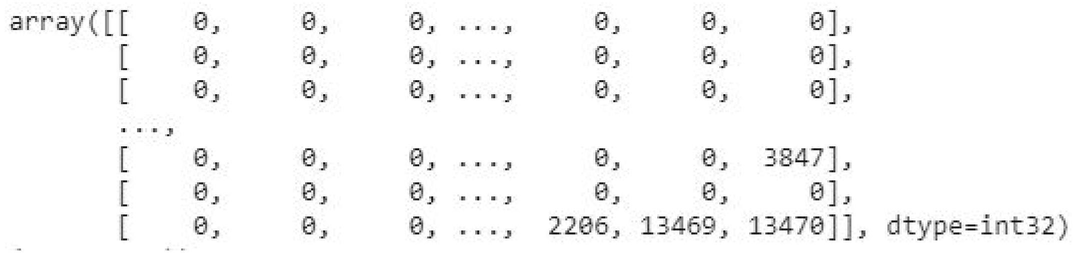
```
! Brilliant ! '
! C'mon
! Gollum 's ` performance ' is incredible
! Oh , look at that clever angle ! Wow , a jump cut !
------------------------------------------------------
brilliant
c mon
gollum performance incredible
oh look clever angle wow jump cut
```

3. Converting data to the final input

Once cleaned, the phrases need to be converted from string to integer or float so that the final input can be obtained. This is also a form of preprocessing wherein the corpus is prepared for final use by the neural network.

The library used is *Keras*. The class *keras.preprocessing.text.Tokenizer* allows vectorizing a text corpus, by turning each text into a sequence of integers (each integer being the index of a token in a dictionary). So the tokenizer creates a dictionary of unique words on the clean phrases and converts each phrase to a sequence of integers where each integer represents the corresponding word from the dictionary.

Then using the class *keras.preprocessing.sequence.pad_sequences*, a 2-D array of integers is created wherein each row represents a phrase sequence. The maximum length of each phrase sequence is truncated to 8 integers and phrases less than 8 integers in length are padded with an appropriate number of 0's in the beginning. The number 8 was chosen carefully since the average length of phrases in the dataset was 4.23 whereas the maximum phrase length was 20. Thus a middle value was chosen. The matrix padded_seq looks like:


```
array([[    0,     0,     0, ...,     0,     0,     0],
       [    0,     0,     0, ...,     0,     0,     0],
       [    0,     0,     0, ...,     0,     0,     0],
       ...,
       [    0,     0,     0, ...,     0,     0,  3847],
       [    0,     0,     0, ...,     0,     0,     0],
       [    0,     0,     0, ...,  2206, 13469, 13470]], dtype=int32)
```

Currently, each phrase sequence is having a size (8,). The preprocessing is still incomplete. We need to replace integer-based phrase representation to a vector-based phrase representation. For this, each of the 8 integers in a phrase sequence is converted to its corresponding vector. For the purpose of comparison, two sources have been used to obtain vector representations:

a. Word vectors trained in Phase I

b. GloVe vectors

In each case, the vectors are 50-dimensional vectors. GloVe vectors have been made publicly available for use on GloVe: Global Vectors for Word Representation. The padded sequence of phrases is converted to vectorial representation by the use of an embedding matrix. After replacing each integer with its vector, the current size of each phrase of the corpus is $(8, 50)$. However, the representation is still not complete.

Each input example is still a 2-Dimensional matrix. To convert it into a 1-D array, we flatten the matrix. That is, for each phrase matrix of size $(8, 50)$, we flatten it to lay the rows side-by-side. The resulting array is having a size $(400,)$. Thus after final preprocessing, each phrase is converted to an input array of size $(400,)$. This can now be fed to the Neural Network.

4. Skewed Dataset

Once the data was preprocessed, NN architecture was defined and data was split for training, validation and testing, it was observed that the dataset was skewed, i.e., there was a significant class imbalance among the dataset. One can observe that among a total of 2,39,232 phrases given in the dataset, the number of phrases in each class are:

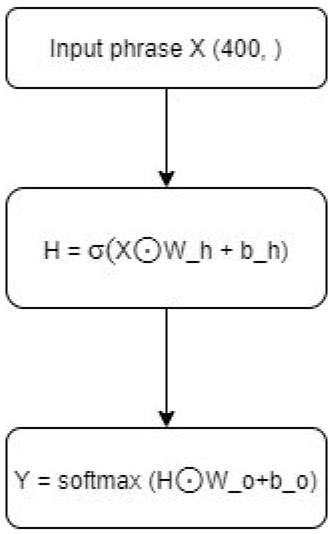
```
{0: 11352, 1: 43028, 2: 119449, 3: 50148, 4: 15255}
```

This means that almost 50% of the data belonged to class 2, ~18% to classes 1 and 3 each and only a total of 14% belonged to classes 0 and 4 combined. This huge imbalance would invariably cause the model to almost always predict class 2 for any input fed. This is a major issue while dealing with classification problems.

In response, while splitting the data into train, dev and test sets, care was taken so that almost an equal number of examples from each class populated the sets and that the model does not discriminate based on class imbalance. This is possible because only a subset of the total data was used for training, validation and testing. Exactly 12,772 phrases were used for training and an equal number of phrases were deployed for validation and testing. Once all the preprocessing has been done, the NN is ready to be used.

5. Model architecture

The model implemented in this phase is a fairly simple neural network with 1 hidden layer. Everything has been implemented from scratch using Python and NumPy only.



The final input having shape $(400,\ )$ is fed to the network. At first the network computes h -the weighted sum of the input layer neurons with the hidden matrix ($h = X \odot W_h + b_h$) . Next the network obtains the hidden layer neurons by applying sigmoid activation function to h ($H = \sigma(h)$). Further, the output layer first computes the weighted sum of the hidden layer neurons with the output matrix ($y = H \odot W_o + b_o$) and finally gives the output layer neurons as a softmax output of y ($Y = softmax\ (y)$). The softmax function assigns the highest probability to one of the 5 possible output classes.

The hidden matrix has shape $(400,\ 500)$ and bias in this layer has the shape $(500,\ )$. Similarly the output matrix has shape $(500,\ 5)$ and bias in this layer has the shape $(5,\ )$. Thus the total number of trainable parameters in this network is $(400 * 500\ +\ 500)\ +\ (500 * 5\ +\ 5) = 2,03,005$.

Cross-entropy loss has been used as the choice of loss function along with softmax activation. Training involves forward propagation with appropriate steps for backward propagation. The size of examples (i.e. number of phrases) used for training, validation and testing is 12,772 each. However, this size was chosen carefully as explained below.

6. Training on word vectors from Phase I

In Phase I, 50-Dimensional word vectors were trained from scratch using the CBOW model. The same vectors were used for sentiment analysis to check the accuracy of the training process. The following graph depicts the plot of training and validation loss v/s epoch for the training process:

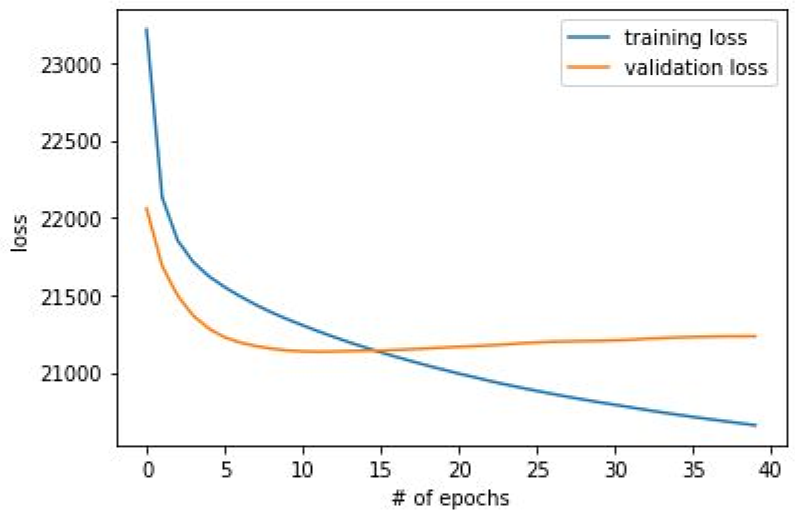


Figure 1: figure depicting the plot of training loss and validation loss v/s epoch for the case where word embeddings from Phase I were used. The model took 22 minutes for 40 epochs with a learning rate of 0.01

The observed accuracies were:

Training Accuracy	Validation Accuracy	Testing Accuracy
20.85 %	19.51 %	19.88 %

The accuracies are as good as random guessing (assuming there is a uniform 20% chance of correctly assigning one of five classes to a sample chosen at random). Thus the model does not perform well on pre-trained embeddings. There are many reasons for this, as this kind of result was expected. The primary reason is that the embeddings were not trained on a large corpus (due to hardware limitations). Furthermore, the corpus for sentiment analysis and word training differ widely. This caused a word mismatch between the trained vectors and the vocabulary of the current corpus as most of the new words did not exist in the trained vocabulary. Thus the embedding matrix for sentiment analysis was sparse and composed mostly of 0's.

The model accuracy can be significantly increased by using vectors pre-trained on a large corpus, preferably similar to the corpus being used for sentiment analysis. This is the case with GloVe vectors as we'll see in the next section.

Since the training and validation accuracy were too low to be increased by parameter tuning, regularization was not applied to the model. One can also conclude that the model was unable to *'learn'* appropriately from the corpus.

7. Training on GloVe vectors

Next, the model was trained using pre-trained 50-Dimensional GloVe word vectors. The following graph depicts the plot of training and validation loss v/s epoch for the training process:

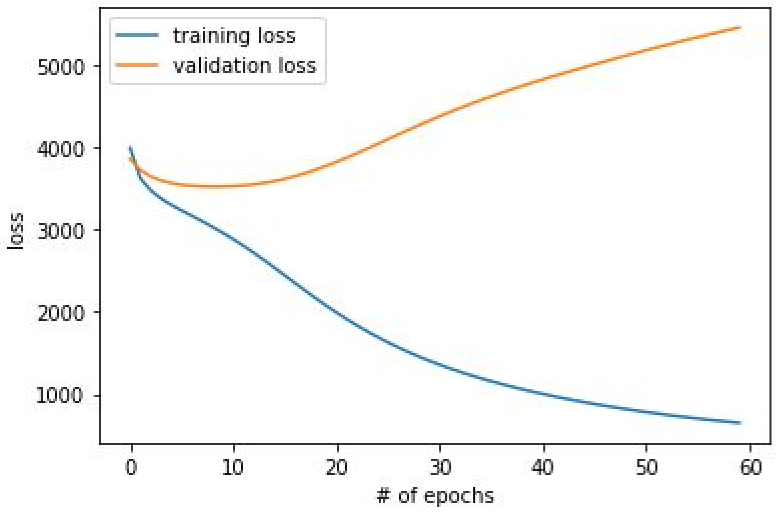


Figure 2: figure depicting the plot of training loss and validation loss v/s epoch for the case where GloVe word embeddings were used. The model took 24 minutes for 60 epochs with a learning rate of 0.007

The observed accuracies were:

Training Accuracy	Validation Accuracy	Testing Accuracy
88.47 %	42.67 %	41.41 %

This is a significant increase (approx 67% abs) over the accuracy obtained using word vectors trained during Phase I. This shows that GloVe vectors are more accurate and robust as compared to vectors trained in Phase I.

It can be clearly seen that the reduction in training loss is accompanied by an increase in validation loss, indicating that the model is overfitting. The problem of overfitting can be tackled by regularization which penalizes large weights for contributing more to the training set. It helps the model to generalize well on unseen data.

8. GloVe vectors with regularization

To overcome the problem of overfitting, regularization was applied on the model. Of the various options, L2 Regularization was used. L2 Regularization is a form of weight regularization that uses the L2 norm $\left(\|w\|^2\right)$ to penalize the weight parameters of the network, with values of the regularization hyperparameter λ often on a logarithmic scale between 0 and 0.1, such as 0.1, 0.001, 0.0001, etc.

In the following graph we depict the application of L2 regularization on our model with varying values of λ. All values were tested by running the model for 60 epochs with a learning rate of 0.006 and using the same initial values of hyperparameters by setting a seed for random initialization. The results may differ in a different environment since random initialization may produce different initial values requiring more/less epochs and/or a higher/lower learning rate.

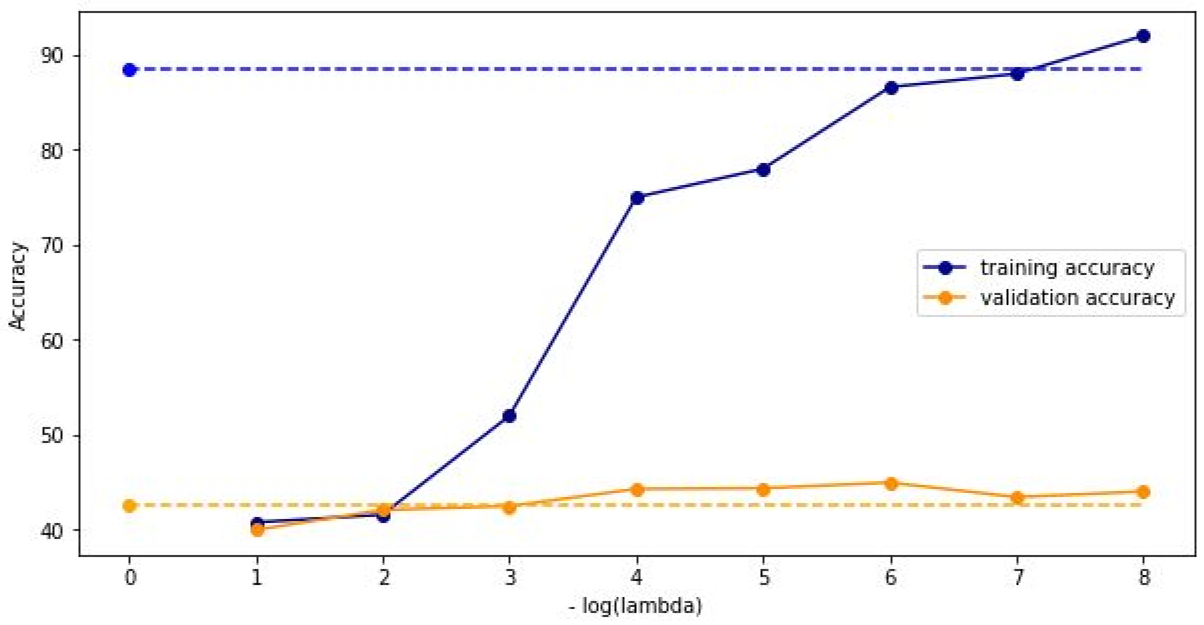


Figure 3: figure depicting the plot of training and validation accuracy v/s negative log of λ for the case where GloVe vectors were used. Dotted lines represent the accuracies for the non-regularized case.

The dotted lines in **Fig 3** represent the training and validation accuracies for the case where regularization was not used ($\lambda = 0$). Now, weight regularization penalizes high weight values by bringing them closer to other lower values. The effect of this is that training accuracy decreases (because weights have been tampered with) and validation accuracy increases (because overfitting due to high weights is resolved). The same is visible in **Fig 3**.

Once regularization is applied with $\lambda = 0.1$, training accuracy falls rapidly. With further changes in λ(by using values in a logarithmic fashion less than 0.1), training accuracy starts to increase gradually while the validation accuracy also increases but more subtly. At $\lambda = 1e - 8$, training accuracy reaches a value as high as 92% while validation accuracy is 44.04%.

a. Key takeaways from the graph

i. Regularization does have an expected effect on training accuracy. It decreases, resulting in more random weight values that would generalize well. The effect on validation accuracy, however, is minute and quite subtle.

ii. One can see from the graph that $\lambda = 1e-6$ is the **best performing value** on dev set with validation accuracy of 44.98% and training accuracy of 86.59%.

iii. Further values of λ would only end up with the model overfitting on the training set. Thus L2 regularization does not help the problem of overfitting to a large extent.

iv. Hence, a more robust form of regularization must be experimented with. For instance elastic net regularization or dropout might be more effective for the given dataset.

Thus, $\lambda = 1e-6$ is the choice of regularization parameter for the current problem and the corresponding accuracies are:

Training Accuracy	Validation Accuracy	Testing Accuracy
86.59 %	44.98 %	44.31 %

9. Limitations and Future scope

a. The word vectors trained in Phase I proved to contain insufficient information for sentiment analysis since the embeddings were trained on a small corpus due to the limitation of time and hardware resources. Thus if one wants to use their own vectors, they should be trained on a much larger corpus, preferably similar to the corpus being used for sentiment analysis.

b. Whether with or without regularization, the validation accuracy did not change much. A big reason could be that a shallow neural network was employed for the task on a subset of the larger dataset due to time and hardware constraints. Hence, a deeper neural network with different choices of activation functions at each layer might be more successful in generalizing rather than overfitting. Increasing the size of the training corpus along with better randomizing techniques could also increase the validation accuracy.

c. Other regularization techniques should be explored to observe changes in accuracies as they might prove to be more effective than weight decay (L2 Regularization). Due to limitation of time, all the options could not be implemented.

d. A major factor to avoid computational complexity during data preprocessing was truncating the size of each phrase. This size could be increased to the length of the longest phrase. Furthermore, longer GloVe vectors of 300 dimensions could be used for representing each word. Both these measures, however, require more computational resources.

e. Thus in future, with more time and better hardware resources at hand, the model accuracy can be increased via a plethora of options.

10. Conclusion

In this report we analysed the performance of word vectors on sentiment analysis. We began by preprocessing the raw data which involved a lot of steps including alphabetization, tokenization, lower casing, stop words removal and lemmatization. Next the data was converted into numeric form so that it could be used as input to the neural network.

The NN architecture consisted of a single hidden layer with a softmax output layer. First sentiment analysis was performed using the word vectors obtained from Phase I. It was observed that the accuracy of the model was very low. We concluded that the model did not perform well since the vectors did not capture enough information.

Then we trained the model on GloVe vectors and a prominent increase in accuracies was observed. This can be attributed to the precision of GloVe vectors. However, the model was overfitting and to address this issue, regularization was introduced.

We observed that application of L2 Regularization had a mixed impact on model performance. The validation accuracy did improve but only by a small margin whereas the test accuracy was impacted strongly. By varying the values of the regularization parameter we saw potential in improving model accuracy by applying a stricter, more robust form of regularization to the model.

We also came to the conclusion that model accuracy can only be increased to a certain extent by regularization. This implies that other methods also need to be implemented to achieve higher accuracies such as using a deeper network or a more complex architecture involving LSTMs or GRUs.