Sentiment Classifier Empowered With LSTM Autoencoder

Term Project

Sayan Ghosh[1]

[1]Big Data Analytics 2019-21, Department of Computer Science, Ramakrishna Mission Vivekananda Educational and Research Institute, Belur Math, Howrah, India

Abstract

Sentiment analysis is a well researched problem in the literature. Starting its journey with Naive Bayes classifier it has passed through the era of recurrent neural networks and has reached the episode of transformers. Although the recent state-of-the-art models for sentiment analysis, namely BERT and its variants achieve significantly good performance in general, in this paper we try to exploit the power of RNNs in light of autoencoders. We integrate an LSTM autoencoder with the general RNN classifier and simultaneously optimize two objective functions to train the network. We have used a subset of Amazon Reviews Dataset for training and evaluation of our model.

1 Introduction

Sentiment analysis, formerly known as opinion mining, is an automated process of determining opinion of people on certain topic based on their statements. It helps companies to monitor their brand value, gain insights from customers' feedback and much more. To have a flavour of the popularity of the problem and its wide range of applications one may refer to [18] and [11].

In a supervised learning framework, sentiment analysis is mainly formulated as a classification problem, where we have a set of predefined classes like {positive, negative} or {happy, sad, anger, surprise, fear}, and the task is to predict which class does a particular document/sentence belong to. There are several ways to do supervised learning for this classification task. The most primitive one is the *Naive Bayes Classifier* [5] which follows a probabilistic approach. It computes the posterior probability of a class, based on the distribution of the words in a particular example (document/sentence). The model works with the

BOWs feature extraction which ignores the position of the word in the example. It uses Bayes Theorem to predict the probability that a given feature set belongs to a particular label. Based on the assumption that the features (words in the vocabulary) are independent, the formula for calculating posterior probability can be written as: $P(label|example) \propto \pi_{label} \prod_{i=1}^{K} P(word_i)$, where K is the vocabulary size and π_{label} corresponds to the prior probability of a particular label. There are several improvements to this model, such as *Multinomial Naive Bayes Model* [17], a hybrid model of Naive Bayes classifier & decision tree namely *NBtree* [7] etc. To alleviate the weakness of the independence assumption used in Naive Bayes, *Bayesian Network Model* [9] came into picture. It is a directed acyclic graph whose nodes represent random variables and edges represent conditional dependencies. This framework is able to capture the dependencies between words in a sentence, but due to excessive computational complexity it is not much used in practice.

On the other hand, with the success of deep learning, people tried to apply neural nets for this problem. The first most successful approach in this domain is *Recurrent Neural Networks* [12]. Here we feed the words of a document/sentence to the model sequentially and get some dense representations of those words using a word embedding layer or by using some pretrained word-embeddings like GLoVe or Word2Vec. These dense representations of words are passed through several layers of RNN modules, which spit out some form of encoded representations of the words and their relation with the previous part of the sentence at each time step. The last time step output is fed to a classifier having several fully connected layers and the classifier predicts the posterior probability distribution for a particular example. The exact calculations behind this mechanism can be found in [15]. But this method primarily had two main problems. To get the contextual information for a word it only looks at the preceding words in the sentence. This issue got resolved with the proposal of *Bidirectional RNN* [13] which is able to capture the contextual information from both sides of a word. The second problem is that for longer sequences, at the later time steps the basic RNN framework is not able to memorize information about the beginning of the sequence. This problem was, to a large extent, wiped out when people introduced $LSTM$[14] and GRU[2]. Recently many transformer based architectures like $ELMo$[10], $BERT$[3] and its variants has achieved state of the art performance in sentiment analysis (along with many other NLP related tasks).

Although transformer based architectures achieve significantly good performance in general, they are difficult to train because of their humongous model size. In this paper, we propose an end-to-end trainable model for sentiment analysis, in the heart of which lies the unidirectional RNN composed of LSTM cells. We introduce an LSTM autoencoder in this framework which ensures to get better encoded representations from the last LSTM layer and helps in achieving better classification performance. Detailed description of model architecture has been presented in section 3.

2 Description of Data

We have utilized Amazon Reviews Dataset * for our work. It is an open source dataset which contains reviews of customers about various Amazon products and their corresponding sentiment in fastText format†. Below are a few characteristics of this dataset:

1. Training dataset consists of 36 lakh examples out of which 2 lakhs have been used for training and 1 lakh for validation (Low computational power is the reason for using smaller number of examples).

2. Test dataset comprises of 4 lakh examples but only 2 lakhs have been used for testing purpose.

3. Most of the reviews are in English, but there a few reviews which are in Spanish, French etc. It would very difficult for any model to learn the characteristics and patterns of the non-English languages from inadequate examples. Moreover, if in the test data the model encounters a language that is not in the training data, it may behave abnormally. So the best idea is to translate the non-English reviews to English language for all the training, validation and test datasets. *googletrans*[4] python module has helped to achieve that.

4. This data, meant for binary sentiment classification, consists of two labels/classes, namely — __label__1 and __label__2. __label__1 corresponds to 1- and 2-star ratings and __label__2 corresponds to 4- and 5-star ratings.

Apart from translation, we followed the usual preprocessing techniques, like — lowercase conversion, removal of urls and special characters etc.

3 Methodology

In this section we discuss the formulation, model architecture, some specific hyperparameters and loss functions used for training the network.

3.1 Formulation

Although the main motto of our chosen problem is classification, for training purpose we formulated our problem as a combination of classification and reconstruction problem, where the task is not just to classify a given review but also to reconstruct the same. The only input to the model is a sequence of words. At inference time the task of reconstruction is ignored.

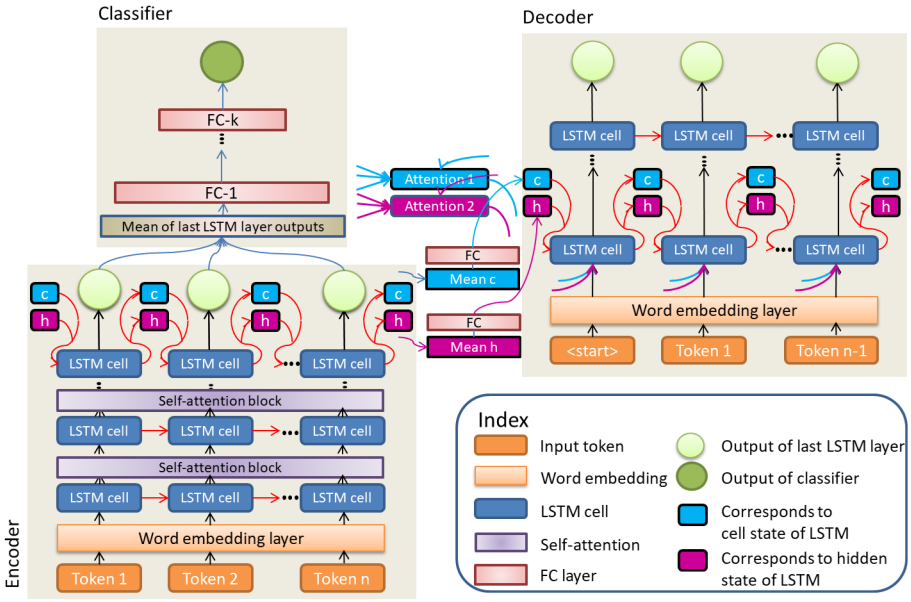


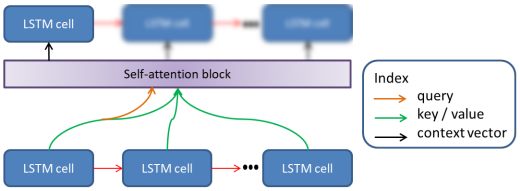
Figure 1: Model Architecture

3.2 Model Architecture

Our model architecture has three main components — encoder, classifier and decoder, which are discussed below. Figure 1 is a schematic of the model.

1. Encoder: It has the following key features:
   - It is an usual rnn framework with a trainable word embedding layer followed by multiple lstm layers — which takes in as input the word tokens and outputs a vector at each time step.
   - For encoder, we have kept the number of LSTM layers to be 5 and hidden size to be 50.
   - Inspired from BERT, we introduced self-attention block in between two LSTM layers to help the encoder learn better by focusing on important parts of the sequence. In this self-attention mechanism, the query is the output of the previous LSTM layer for a particular time step, and keys/values are the outputs of the previous LSTM layer for all time steps.

*Source of Dataset: Kaggle
†fastText format: __<label Y>__ <review>

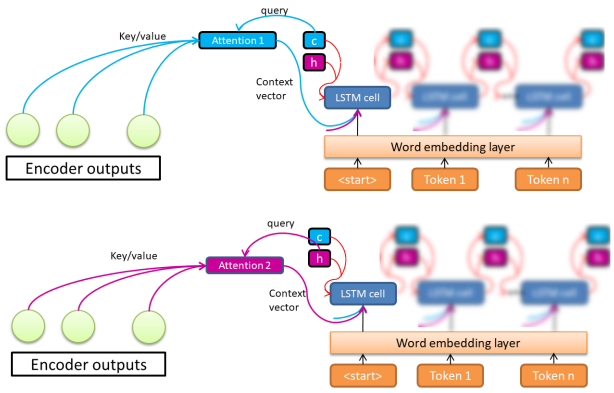


2. Classifier: It has the following key features:

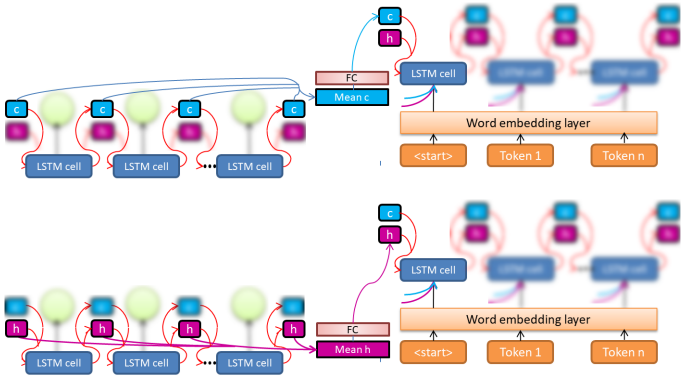
   - It is a feed-forward neural network with multiple fully connected layers — which takes in as input the output of last LSTM layer and outputs the probability of a particular example being in a particular class.
   - We have kept 3 fully connected layers in the classifier, the first two contains 100 and 25 units respectively. The number of units in the last layer is equal to the number of classes (2) and the output of the last layer is passed through softmax activation to get the posterior probability distribution for a particular example.
   - Usually the last time step output of encoder is taken as input to the classifier. But for longer sequences this may result in information loss. To account for this loss of information, average of all time-step output has been taken as input to classifier.

3. Decoder: It has the following key features:

   - It is an RNN framework with trainable word embedding layer followed by multiple LSTM layers, initiated with <start> token in the first time step.
   - For decoder, we have kept the number of LSTM layers to be 4 and hidden size to be 40.
   - The initial LSTM layer of decoder has three inputs — embedded token (output of word embedding layer), context vectors coming from two attention modules where cell state / hidden state of decoder LSTM cells acts as query and outputs of last LSTM layer of encoder act as keys/values. At each time step these attention mechanisms help the decoder to judiciously utilize the information encoded by the encoder to generate a particular word.



- In any RNN framework for generating sequence, usually the hidden states of the LSTM layers are initialized randomly or by zero vector. But here to make the full utilization of the information capsuled by the encoder, we connect the cell state and hidden state of the last LSTM layer of the encoder with that of the first LSTM layer of the decoder by means of a single fully connected layer.



3.3 Hyperparameters

We mention below a few hyperparameters that we think are important to build the aforesaid model.

i. Number of LSTM layers in encoder is greater than that of decoder because good reconstruction with weaker decoder requires the later layers of encoder to learn something really meaningful. [16]

ii. We used SELU activation [6] in classifier. This does away with the need of batch normalization, resulting in fewer parameters and less training time.

iii. AdamW optimizer [8] is used for training both the autoencoder and classifier. It is a variant of Adam, but is known to generalize well than Adam.

iv. We used a dropout of 20% throughout the network. It acts as regularizer and also stabilizes the training process.

3.4 Loss functions

In each epoch we train our model in two phases — in the first phase we train the autoencoder (i.e., the encoder and decoder together) and in the second phase we train the classifier along with the encoder. In the first phase, by trying to reconstruct the original sequence we are ensuring whatever is learnt by the encoder is meaningful. In the second phase, we are asking the model to learn how to classify properly. So in these two phases we use two different loss functions:

1. In phase I, as the reconstruction loss we use the average of the cross-entropy loss of reconstructed token and original token of all time steps. The padded tokens are masked out while calculating this average.

2. In phase II, we use a binary cross-entropy loss function to measure the classification error.

4 Result

Figure 2: Change of training and validation loss during training

We present the results based on the performance of our model on the test dataset. The evaluation matrices are accuracy, precision and recall. We trained our model for 10 epochs and the change in training and validation loss during this training process is shown in Figure 2.

With the trained model, the accuracy, precision and recall obtained on the test dataset are respectively 90.5215%, 91.1331% and 89.7154%.

5 Conclusion

The architecture we proposed in this paper is quite generic. People can use this backbone and add further complexities to the model by introducing additional LSTM layers, changing unidirectional LSTM to bidirectional LSTM etc. Advanced activation functions like SERLU or DELU can be tried in place of SELU. In our model we have used the basic *Bahadanau's attention* [1], which can be replaced with more sophisticated attention mechanisms to achieve better performance. Last but not the least, our intuition that when decoder is weaker than encoder it helps in gaining better encoded representations, is yet to be proved.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[2] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].

[3] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[4] SuHun Han. *googletrans 3.0.0*. 2020. URL: https://pypi.org/project/googletrans/.

[5] Pouria Kaviani and Sunita Dhotre. "Short Survey on Naive Bayes Algorithm". In: *International Journal of Advance Research in Computer Science and Management* 04 (Nov. 2017).

[6] Günter Klambauer et al. *Self-Normalizing Neural Networks*. 2017. arXiv: 1706.02515 [cs.LG].

[7] Ron Kohavi. "Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid". In: *Second International Conference on Knoledge Discovery and Data Mining*. 1996, pp. 202–207.

[8] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

[9] Jonathan Ortigosa-Hernández et al. "Approaching Sentiment Analysis by using semi-supervised learning of multi-dimensional classifiers". In: *Neurocomputing* 92 (2012). Data Mining Applications and Case Study, pp. 98–115. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2012.01.030. URL: https://www.sciencedirect.com/science/article/pii/S0925231212001282.

[10] Matthew E. Peters et al. *Deep contextualized word representations*. 2018. arXiv: 1802.05365 [cs.CL].

[11] Ines Roldos. *5 Real-World Sentiment Analysis Examples*. 2020. URL: https://monkeylearn.com/blog/sentiment-analysis-examples/.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362. ISBN: 026268053X.

[13] Mike Schuster and Kuldip K. Paliwal. "Bidirectional Recurrent Neural Networks". In: *IEEE Transactions on Signal Processing* (1997).

[14] Sepp, Jurgen Hochreiter, and Schmidhuber. "Long short-term memory". In: (1997). URL: https://www.bioinf.jku.at/publications/older/2604.pdf.

[15] Alex Sherstinsky. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306. ISSN: 0167-2789. DOI: https://doi.org/10.1016/j.physd.2019.132306. URL: https://www.sciencedirect.com/science/article/pii/S0167278919305974.

[16] *Should autoencoders really be symmetric?* 2020. URL: https://www.reddit.com/r/MachineLearning/comments/ef1xe8/d_should_autoencoders_really_be_symmetric/.

[17] Shriram. *Multinomial Naive Bayes Explained: Function, Advantages Disadvantages, Applications in 2021*. 2021. URL: https://www.upgrad.com/blog/multinomial-naive-bayes-explained/.

[18] Walaa et al. "Sentiment analysis algorithms and applications: A survey". In: *Ain Shams Engineering Journal* 5.4 (2014), pp. 1093–1113. ISSN: 2090-4479. DOI: https://doi.org/10.1016/j.asej.2014.04.011. URL: https://www.sciencedirect.com/science/article/pii/S2090447914000550.