Deep Dive in Scalable Sentiment Analysis with Modern Solution

Jimmy Gan
Graduate Stuent in M.S. Applied Data Science, University of Southern California, Los Angeles, California 90089, USA
(Dated: April 30, 2021)

Throughout the year numerous methods have been developed to do scalable text mining. Especially after social media started to grow more and more popularity after mid 2000s, large number of data on text have been stored. And they could only be put into commercial use after categorizing and sorting so that the user data can be utilized when making marketing or advertising decisions. The following sections will take a deep dive on the workflow with machine learning methods when it comes to sentiment analysis and the reasons behind each procedure as well as how to evaluate them.

Keywords: Machine Learning, Natural Language Processing, Neural Network

I. INTRODUCTION

Modern solution on sentiment analysis is combining supervised and unsupervised machine learning method such as rigid algorithmic process such as tokenization, part of speech tagging, and word2vec and utilization on different types of neural network such as Recurrent Neural Network or Transformers. In the following sections we are going to explore their effectiveness both theoretically and practically.

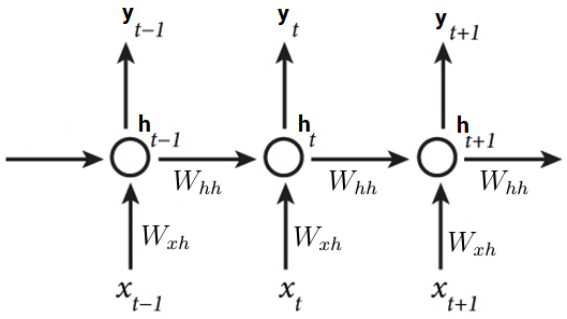

FIG. 1. Recurrent Neural Network architecture [3]

II. HYBRID MACHINE LEARNING METHODS ON SENTIMENT ANALYSIS

Hybrid machine learning was introduced and have been grown into more and more advanced since the beginning of natural language processing. Starting from word embedding, word2Vec, to mixing up with Recurrent Neural Network to train on sequences to utilizing Long Short-Term Memory Neural Network to effectively train on longer sentence.

A. Tokenization and POS Tagging

Basically Tokenizaion is a powerful tool that encode each valid word with a number [1]. And the tokenizaer API provided by TensorFlow Keras is powerful enough to recognize marks in sentence strings such as the exclamation mark, or coma. This helps the words to transform into numbers that machine can understand.

Part of Speech Tagging is aimed to categorize each words into different categories such as adjective, adverb or preposition. We can often enhanced categorizing words using Hidden Markov Model (HMM). Part of Speech tagging is basically having the word's tag probablity to be dependent on the previous word's tag, so that, in a sense, the algorithm can label the words more accurately because of the context of previous words. And this can be achieved in a algorithmic sense using backtracking to select the best word tag pair, in condition of a word that can be tagged with multiple tags. For example, the word "lie" can be labeled as None and Verb. With proper context, the word can be correctly labeled, whose mechanism can be utilized when constructing basic model for sentiment analysis.

Hidden Markov Model is another statistical model that helps the program to come up with the most likely decision based on previous trained decisions and what is presented in the real time data. It heavily relies on conditional probabilities. And it is often used in Part of Speech Tagging. It has a start probability with cumulative probability of 1 after summing up the whole sequence [5]. Like the figure shown below, it also has a transition probabilities in blue arrows. With the information from the adjacent state, one can navigate the probability of the occurences of adjacent word. This is particularly useful on carrying out reasonable guess on type or tag of the current word when HMM being applied in Part of Speech Tagging.

B. Unsupervised Machine Learning for Sentiment Analysis

A simple neural network with Embedding layer and another global average pooling layer can also be able to put out decent accuracy predicting sentiment of a sentence. After a fix length tokenization, and padding on the each sentence to make up for the data loss, it could

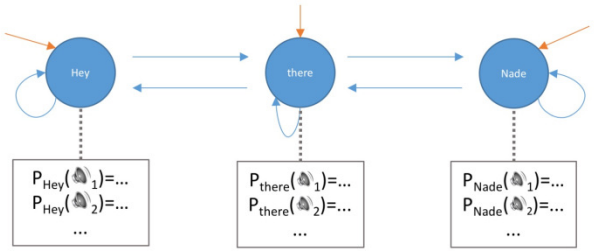

FIG. 2. Hidden Markov Model architecture [5]

then be put into the neural network process where the sequence will be evaluated and each word will be assigned a weight, to train [2]. Then after the model was trained, it then can put in this trained value of the word, and add them up to determine which end of the sequence of words are closer to, 1 for positive sentiment and 0 for negative sentiment. And this type of training is the basic neural network approach to determine the scale of sentiment that a sentence is expressing. However, the words can be in different orders of the same sentence, its sentiment evaluation would be the same in this case. The embedding layer is not training based on sequence of words, but the combinations of word's values from the tokenizaions. And adjusting those values according to training process.

Another approach would be via Recurrent Neural Network. According to Figure 1, the architecture of the Recurrent Neural Network indicates that the output is dependent on the previous computations, which means the sequence would be captured when it goes through the neural network. This architecture is similar to the algorithm used in the previous section on enhancing labeling unseen words for the Part of Speech Tagging function. However, in practice the RNNs are limited to looking back only a few steps instead of the entire previous sequence using some sort of feedback. [3]

Therefore Long Short-Term Memory networks (LSTM) are often utilized to do sequencing dependent neural network training, such as sentiment analysis, because it oftentimes learn the important feature of a long sequence, so next time, with the similar context, it could predict the sentiment more accurately. From the figure, the one main difference is that, there is a cell state that records the whole sequence, and on each LSTM network, there is a hidden state recording the previous input keeping the relevant information [3]. And the output is basically the output of the cell state to the next neural net. In practice, this approach can increase the accuracy on sentence that are not in the same order even with the same words, because ordering of words can oftentimes express sarcasm, a negative sentiment, hence optimize the model's performance.

RNN is performing worse than LSTM when it comes to longer sequence because due to inherent architecture that it takes weight depended on the previous cell's output, and it does not keep track of the long-term dependencies as the RNN's gradient starts to vanish further down to the beginning of the sequence in the case of a long sentence. On the other hand, LSTM utilizes cell state and a few extra gates to determine what information to put or remove from the cell state, which goes through the entire sequence, to keep track of the long term dependencies. LSTM is expected to have a better performance.

C. Comparison on Performance and Selecting Baseline

From the Problem Set 5 and previous lecture, we can establish the fact that single layer embedding is not going to perform as good as Recurrent Neural Net because word embedding does not record the sequence but only tokenize the words into vectors. And then the sentiment prediction is based on the sum of those values of embedded words. Therefore, Recurrent Neural Net is superior than basic embedding because it also process the sentence sequentially. However, due to the issue of vanishing gradient, which indicates that for longer sentence, the model can only effectively learn the last few words instead of the words in the beginning of the sentence, since back propagating gradient would be near 0 at that point, resulting in small to no learn at all, the model performs badly on long sentence while it misses out the information of the start of the sentence when making prediction. Therefore, the selected baseline would be based on LSTM layer. With the extra cell state that preserves most information of the sentence, and three inherent neural net that ultimately decide on that particular word what information in the cell state, which is the current sequence so far, should be updated, deleted or filtered then output to the next word [6], LSTM held its ground when it comes to longer sentence sentiment analysis, compared to RNN. However, due to nature of LSTM's architecture, it certainly is slower than RNN.

From the above comparisons, a model based on LSTM would be the baseline to use to compare with state of the art architecture like BERT.

D. Bidirectional Encoder Representations From Transformers

Currently the state of the art model for natural language processing is BERT, which stands for Bidirectional Encoder Representations From Transformers. The concept behind the transformers is basically to parallelize sequential data. And the nature of BERT architecture also helps the model to gives more attention to certain feature, which is certain types of words in this case.

Transformer encoder is accepting each words at the same time and generating the embedded vectors simultaneously while RNN or LSTM has to do it word by word. BERT first uses positional encoding to gather context information, in a result of better performance than one with only embedding. From the Figure 6, we can see

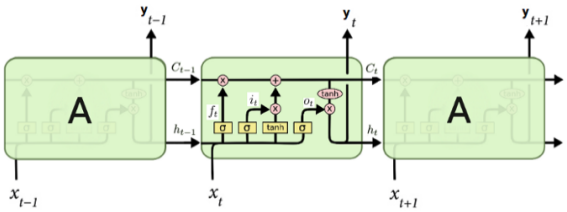

FIG. 3. Architecture of LSTMs [3]

$$Z = softmax\left(\frac{Q.K^T}{\sqrt{Dimension\ of\ vector\ Q, K\ or\ V}}\right).V$$

FIG. 4. Formula for Attention Vector of each word [7]

that the output with the positional information is then pass through an attention layer and feed forward layer. The attention layer is to capture which part of the input should be the primary focus. Sometimes, the verb or the adjative would be the primary focus compared to other words in a sentence. To prevent the attention on itself, it would then proceed to get multiple attention vectors of each word and compute the weighted average of those attention vectors to then be the finalized attention vector for each word. Inside the attention layer, we would have V, Q, K vectors that describe the attention for each word. And for multi-head attention layer, it simply means stacking the single headed attention layer. Since the next stage only takes one attention vector for each word, each attention vector Z is calculated using the formula in Figure 5 and then concatenate to one vector for that word. After the attention layer, feed forward layer wold perform reshaping the dimensions of results with hidden layer to prepare it for the next stage. From the paper [7], the input would be combined with output from the decoder both passed into another attention layer to evaluate the result. In the example of language translation, the other language would pass in each word in the decoder through the masked attention layer then trained with the complete English input from the encoder in another multi-head attention layer just to be fed forward to then proceed with linear and softmax operation to evaluate the designated language word's accuracy.

After each layer, not limited to attention layer, but also Feed Forward layer as well, normalization is applied, such as batch normalization to make the loss smoother. To summarize, transformer is largely applied in sequence to vector, sequence to sequence and vector to sequence transforming. And its benefit of learning with different attention to context is the reason why, in theory, more superior than LSTM when it comes to sentiment analysis.

E. Streamlining the Process of Sentiment Analysis

The data I am using is Sentiment 140 [9], a Baseline model is constructed based on the lecture [8], which is consisted of one embedding layer of 2500 frequent words, then two LSTM layers followed by one linear layer and one softmax layer. I used a custom method to clean the data getting rid of unrecognizable ascii characters and twitter handles, as well as hashtag and links and initialized a tokenizer of maximum length of 100 words and embed of 2500 words. And one particular attribute of the data is that there are no neutral cases in the training data while there are several neutral cases in the testing set. Therefore, I set a threshold on determine which cases should be categorize into neutral. However, when comparing performances between baseline and the transformers' neutral cases would be omitted.

After splitting the training data into 9 to 1 for the ratio of training to validation data, I used a small sample of data to test out different parameters such as number of words for the embedding, epochs, batch size and output dimension for the embedding layer. Since the data is balanced with 800,000 positive entries and 800,000 negative entries, even when I picked a small sample data to run, it is still pretty representative on what the outcome would look like. The baseline is overfitting on the training data even after I expanded the dataset and also turn down the learning rate so the validation accuracy won't stay around the same area throughout each epoch after the third or fourth epoch. The baseline has a performance of 72 to 74.37 percent accuracy for positive and negative cases when neutral cases cut off was at 0.6.

Then with the help of the Colab Project by Venelin Valkov [9], I was able to run the Sentiment140 data set on pre-trained base cased BERT model. Gathering the results from these two models, and evaluate the result of the confusion matrix, we can then calculate the accuracy of around 82 percent while only training on 36000 data entries out of 1.6 million due to limited hardware.

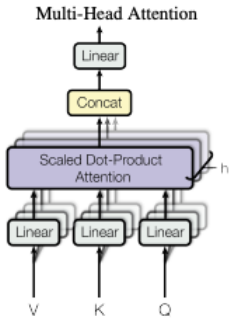

FIG. 5. Multi-Head attention layer in BERT [7]

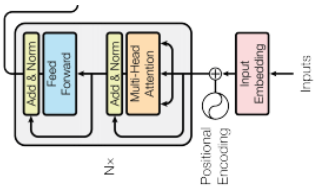

FIG. 6. BERT structure 1 [7]

F. Model Evaluation and Tuning

I have built two baseline because I was running two scripts on two different device since one of them is quite slow. And they both have the same content. But one of them has one more metrics that ignore neutral cases. And its accuracy is 292 over 359 positive/negative test cases from final-base-2. And the baseline can actually give out reasonable scores for neutral statement, which is showcase in final-base-2.

For the transformers model, I basically got rid of all the neutral test cases, and trained on data of 36000 entries of relatively balanced data from Sentiment140 and a validation set of 4000 entries. With such limited data, my validation accuracy and loss did not improve greatly. And I need to put more data entries in the model to train for better results. However, due to the limited usage on GPU on Colab, it would take far too long to train 1.6 million data entries when it almost took me 1 hour to train about 40000 entries with Colab's GPU. The accuracy is still impressive, which is 295 over 359 positive/negative test cases, around 82.17 percent while baseline is 81.13 percent.

From the above performance, we can see the gap between LSTM and the transformer. Even when the training/validation data set is more than 10 times smaller than baseline training/validation data set, the transformer can still achieve slightly better performance. Another factor may also be contribute to this is that, in the baseline, the custom data cleaning method does omit punctuation while the BERT data still has it because punctuation sometimes can be very useful to understand the context. Other than that, the attention on contextual learning to give the appropriate weight of the same word depending on different situation really shines when it comes to natural language processing because human language like English uses the same word for different expression so often in daily life.

G. Presentation

After saving the .bin file of with the highest validation accuracy, I initialize a FastAPI project. Then I put the model file in it and create a post method. Then I created the classifier model class as well as another class for encoding the input text, loading the pre-trained model to the classifier and making predictions. After adding the post method to make prediction on the incoming text, I deployed the FastAPI onto an EC2 instance in a Docker container instance. Shortly after, I built a simple user interface on React.js that can communicate with the FastAPI hosted on EC2 instance and deployed the frontend onto AWS Amplify. Now the application can be accessed via `https://main.d9lrjxdyfzsjr.amplifyapp.com`

III. CONCLUSIONS

Overall, based on my previous work on Problem Set 5, I take a deep dive in the state of the art tools for natural language processing. To build upon the foundation on this paper, I will take a look on the transformer model that I wrote, and maybe do some optimization to improve the training time. At the same time, I will use dedicated

EC2 instance to train models to save more time instead of using CPU to train for the majority of this project. Also, to develop a better understanding, I will choose another data set next time to have neutral cases in the training data or multi-labels scenarios for the project instead of binary sentiment analysis. Overall I demonstrate why BERT is superior than classical Recurrent Neural Net model even with disadvantage on the amount of data entries it can access when training and how it solidifies its position as one of the most powerful tools when it comes to natural language processing due to the encoder and decoder that better preserve the features of original data on the process of sequence to vector transforming.

Reference

The following references will be formatted later

[1]Barba, Paul. "Machine Learning (ML) for Natural Language Processing (NLP)." Lexalytics, 2 Oct. 2020, www.lexalytics.com/lexablog/machine-learning-natural-language-processing.

[2]"TensorFlow Course 3." Google, Google Colaboratory, colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow.

[3]Oppermann, Artem. "Sentiment Analysis with Deep Learning." Medium, Towards Data Science, 11 Aug. 2020, towardsdatascience.com/sentiment-analysis-with-deep-learning-62d4d0166ef6.

[4]"Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow", 2nd Edition by Aurélien Géron. Check also the Appendix B. Machine Learning

[5]Sritanyaratana, Nade. "Natural Language Processing (NLP) Fundamentals: Hidden Markov Models (HMMs)." Nade's Notes, 7 June 2017, nadesnotes.wordpress.com/2016/04/20/natural-language-processing-nlp-fundamentals-hidden-markov-models-hmms/.

[6]Olah, Christopher. "Understanding LSTM Networks." Understanding LSTM Networks – Colah's Blog, colah.github.io/posts/2015-08-Understanding-LSTMs/.

[7]Vaswani, Ashish, et al. "Attention Is All You Need." ArXiv.org, Cornell University, 6 Dec. 2017, arxiv.org/abs/1706.03762.

[8]Abram, Marcin. Lecture 21. DSCI 552, University of Southern California, 1 Apr. 2021

[9]"Sentiment140 - A Twitter Sentiment Analysis Tool." Sentiment140, Stanford University, help.sentiment140.com/for-students/.

[10]Valkov, Venelin. "Sentiment Analysis with BERT and Transformers by Hugging Face Using PyTorch and Python." Curiously, 20 Apr. 2020, curiously.com/posts/sentiment-analysis-with-bert-and-hugging-face-using-pytorch-and-python/.

[11]Valkov, Venelin. "Deploy BERT for Sentiment Analysis as REST API Using PyTorch, Transformers by Hugging Face and FastAPI." Curiously, 1 May 2020, curiously.com/posts/deploy-bert-for-sentiment-analysis-as-rest-api-using-pytorch-transformers-by-hugging-face-and-fastapi/.

DATA AVAILABILITY

Data is available at https://drive.google.com/drive/folders/1TE9wC_De7p4-AEOSwEQrCAIERQQ3BtVq?usp=sharing

CODE AVAILABILITY

Code is available at https://github.com/jimmygan0829/DS552-Final-NLP special thanks to Venelin Valkov for the sample code[10]