SENTIMENT ANALYSIS WITH ATTENTIVE TREE-LSTM

— Dynamically Deciphering COVID-19 Tweets

Lun Li,[1] Shengjie Liu,[1]
[1]Stevens Institute of Technology
lli68@stevens.edu, sliu88@stevens.edu

ABSTRACT

In this project, we adopt tree-LSTM neural network for sentiment analysis. Attention mechanism is fused into the network architecture to enhance the prediction by identify/annotating the key factors – words and phrases. An enhanced version is proposed to partially penalize non-root scores in the parsing tree. The model is calibrated based on standard tweets and applied on COVID19 tweets through a periodically turning process. We also discussed training with unlabelled data, which is usually the case for arising new topics.

1 Introduction

Sentiment analysis is one of the fundamental topics in natural language processing, the applications has been seen in various areas, sentiment extraction from newsletter, sentiment identification from social media platform, sentiment analysis of reviews, e.t.c.. Though numerous researches have been devoted to this area, there are several outstanding challenges while conducting such analysis: 1) the language style appears to be highly domain dependent; 2) in practice, the data are oftentimes unlabelled; 3) the effectiveness of model is determined by its architecture and training methodology that is by no means crystal clear.

On the other hand, in terms language modeling, the prevalence of deep learning promotes the usage of neural networks, convolution neural network(CNN), recurrent neural network(RNN) has succeeded in many applications. Nevertheless, neither RNN nor CNN is not capable to fully capture the grammatical perspective of sentence. Indeed, linguists have shown the the syntactic properties of natural language inherits a nested structure, therefore the design of neural network should account for this recursive pattern. Recent advance proposes the tree-recursive neural network seems to be a good fit to interpret sentence. And they already had successful stories in language parsing, namely, dependency parsing and constituency parsing.

In this project, we present methodologies to address challenges in sentiment analysis with help of recursive neural network, in particular, we propose an enhanced version of attentive tree-LSTM structure to improve the performance of prediction. As an application, we deploy the model to cope with unlabelled tweets related to ongoing epidemic – COVID19, shedding light on real-time sentiment monitoring.

2 Literature Review

The earlier approaches for sentimental analysis was on a symbolic level (unigrams, bigrams, bag-of-words features), which suffer from curse of dimension, huge enumeration space, and poor generalization ability[2]. Miklov et al [5, 9] came up with the concept of distributed word representation, i.e., numericalization of word via deep learning model. Based on the representation, various type of neural network can be applied for different language tasks. The current state of art for predictive language task, such as semantic relatedness and sentiment analysis, is *Tree-LSTM*, which requires syntactic parsing and learning LSTM parameterization in conjunction with tree topology. Syntactic parsing is addressed mainly by Socher et al and Manning et al in their seminal papers [8][4], where the former leads to a constituency parser based on neural network and latter generates dependency parser via learning a transition-based task. The work by Zhu et al [10] and Tai et al[6] device similar techniques to have one-to-many dependencies of each LSTM and proved their advantages over baseline models.

In general, for language tasks, attention mechanism would potentially improve performance. This is usually done by assigning high probability on important words. Reader can refer to [3] for the principles and paper[1] presented by researchers in Google Brain. In tree-LSTM, there are also attempts made to enhance tree-LSTM by attentions, for research along this line has [7, 11].

3 Methodology

We start by having an overview of the methodology adopted(see *Figure 1*). Notice, as our intention is to deploy the sentiment classifier in real-time environment, it is critical to guarantee both quality and efficiency.

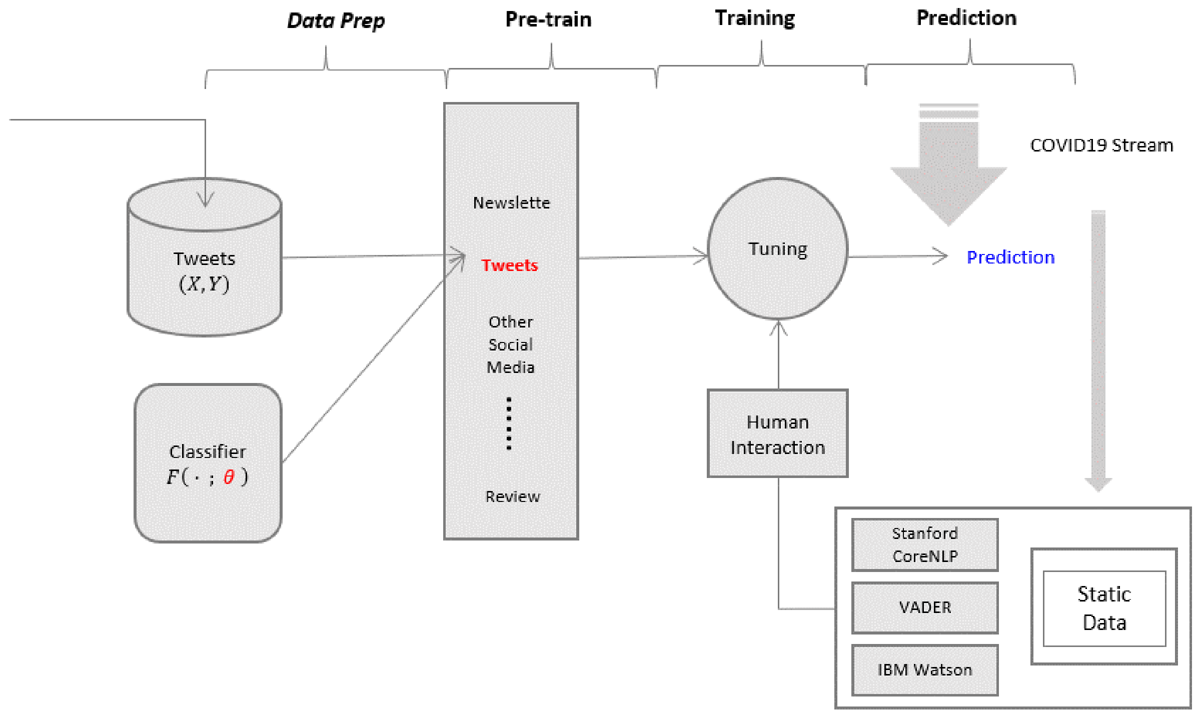


Figure 1: Workflow

- *Data/Model Preparation:* clean raw data from Kaggle database, e.g. tweets, corresponding sentiment, and gather a set of candidate classifiers parameterized by θ;
- *Pre-training:* pre-training the chosen model with tweets data collected; as each domain has its own unique language style, for example, the text in newsletter is quite different from social platform such as tweets, as the latter has more wild styles, it makes sense to warm up the model with similar text style at this stage. This is analogous to using different word embedding depends on the applications.
- *Calibration:* calibrate periodically the underlying model with static COVID19 data persists from streaming data. Notice these tweets are unlabelled, to be able to fine tune the model, standard NLP agents are hired to automate the assignment of label. If the majority vote results in a tie, human interaction is required.
- *Prediction:* predicate streaming COVID19 tweets sentiment with up-to-date model.

Observe the model parameters get updated in two steps above, the pre-training step as well as calibration step. While the former one situates the model in the proper context, fine-tuning calibrates to the COVID19 data of interests. In the rest of the section, we detailed each steps involved.

3.1 Data Cleansing

Text data cleansing is much involved than other format due to rich vocabulary and high variability. The common pre-processing includes tokenization, unification of cases, removal of punctuation and stopwords, stemming and lemmatization. Tweets language is quite non-standard, as there is almost no restrictions on the language style, i.e., typo, slang, abbreviations, all kinds of symbols. Therefore, extra cares need to be taken to pre-process tweets.

Web links occurs very frequently in tweets when people wants to direct others to the website. For sentiment analysis, this usually has limited contributions, thus simply remove them is not an unreasonable step. The URL does not come up with a single format, e.g.,

"www" "http://" ".com" ".edu" ".org"

The most efficient strategy is to use regular expression for removal.

Hashtag is another special feature of tweets, used to group people of common topics. The words following hashtag, however, is most time not a proper English word but a concatenated phrase. For instance, we could have "#stayathome", which really means "stay at home". To cope this, we first locate the hash tagged words, replace # by whitespace, then apply word segmentation[1] This ensures the hashtag gets properly interpreted.

In tweets, we also observe considerable occurrences of cursing words that people often write "*" or repeated "*" instead. Although word embeddings can assign a vector to punctuations, but the issue is its irregular form. It is clear distinguishing different replications of "*" by different vectors is undesirable. We unify all of them by a self-defined token, i.e., ⟨BADWORD⟩. To give an example:

My legs are so **** sore. And my feet hurt to walk.
⇒ My legs are so ⟨BADWORD⟩ sore. And my feet hurt to walk. What a day.

As casual as tweets can be, there are legitimate *typos* that people in this community has common understanding. Nevertheless, for machine learning task, those variants won't be recognized easily, in fact, they bring enormous confusions. The word *cool* in sentences below have exactly the same meaning, but they are shown in different form:

it's so cooooool, this a cool hairstyle, he is such a coooool guy.

When embedding these words, they will result in different vectors and initialized with completely different values. More importantly, some of these words are directly impacting sentiment of the sentence. To deal with redundant duplicates alphabet in the word, we apply a heuristic search, kill duplicates one at a time until finding a matching English word. Although one would be concerned about many candidate words popping up in this procedure, in practice, it works pretty well, as most of these words are quite simple word.

3.2 Base Model

Before introducing the recursive neural network, in particular, treeLSTM, we introduce our base model and its enhanced version. This helps for comparison and understanding its weakness.

Among various neural network structures, recursive neural network appears to be a standard choice for sequence modeling. In general, the model works as follows: given input sentence $X = \{w_1, ..., w_n\}$(w_i's are word embeddings), the θ-parameterized computational unit $F(\cdot)$ (could be GRU, LSTM, .e.t.c..) unrolls exactly n times [2],

$$\rightarrow \; F(w_1) \; \rightarrow \; F(w_2) \; \rightarrow \cdots \; \rightarrow \; F(w_n) \; \rightarrow \tag{1}$$

The explicit output of each F is the information gets passed along to the next unit, while there is also implicit outputs, depending on the type of F used, at each word, an output vector y_i is generated.

Our baseline model set up is a *Bi-Direction LSTM* with self-attention. The stacked bidirectional LSTM unit, at each stage, produce two hidden vectors $\hat{h_i}$ and $\bar{h_i}$, these are further used to produce attentions. Namely, we introduce a two-layer(with last year being dimension 1) neural network $G(\cdot)$ with certain activation, to normalize all single output to a probability distribution, softmax is applied, i.e.,

$$\bar{\alpha}_i = G([\hat{h}_i \; \bar{h}_i]), \;\; \alpha = \textit{softMax}(\bar{\alpha}), \;\; \alpha \in \mathbb{R}^n \tag{2}$$

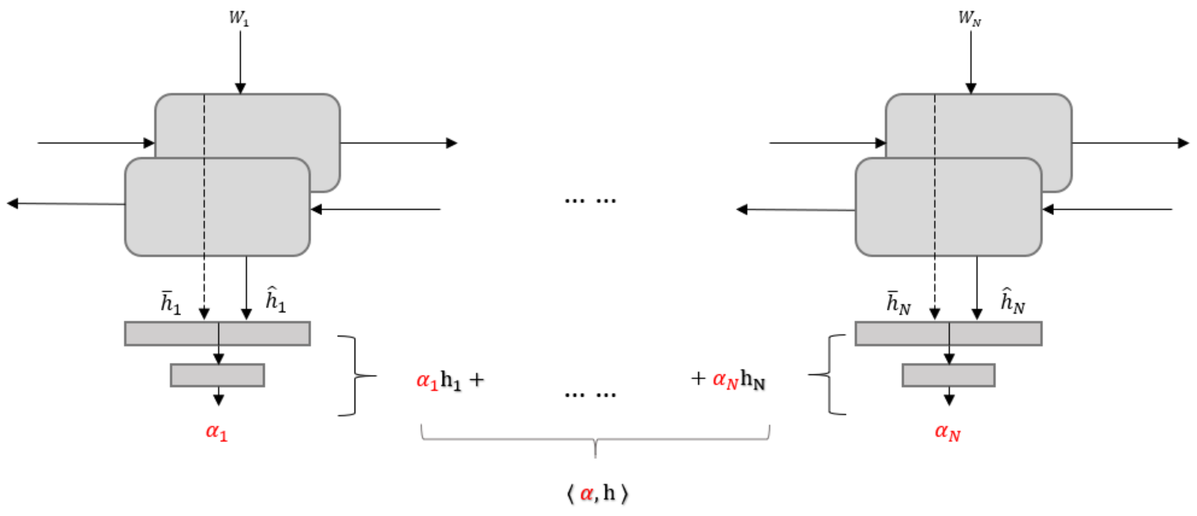
To synthesize these output vectors, we apply a weighted sum,

$$\bar{Y} = \sum_{I=1}^{n} \alpha_i [\hat{h}_i \; \bar{h}_i]^\top \tag{3}$$

Compared to standard recurrent chain only using the single output from the last unit for classification, the attention mechanism takes weight of each words into considerations. In return, in the prediction stage, it could potentially recognize the most relevant words determine the sentiment.

[1]Python has package wordsegement that is based on a trillion-word corpus.

[2]For simplicity, we put aside padding.



W_1
W_N
$\bar{h}_1$
$\hat{h}_1$
$\bar{h}_N$
$\hat{h}_N$
α_1
α_N
$\alpha_1 \mathbf{h_1} +$
$+ \alpha_N \mathbf{h_N}$
$\langle \alpha, \mathbf{h} \rangle$

Figure 2: Bi-Directional LSTM with Self-Attention

In some cases, the data comes in with selected text, i.e., the words extracted from original tex directly leading to the sentiment. This piece of information is useful to better train attention mechanism. Along with text X, the selected text $S \subseteq X$ is also part of input, if we can transform S to the same format as α, then there is a target to benchmark to. Remember, α is essentially a distribution over X, and

$$\sum_{I=1}^{n} \alpha_i = 1 \tag{4}$$

S can be turned into a distribution as follows:

$$\alpha^*(w) = \mathbf{1}_{w \in S} \frac{1}{|S|}, \textit{ where } |S| \textit{ is the length of selected text.} \tag{5}$$

Intuitively, it equally distribute probability mass on the selected words and left zeros to others. If we can minimize the distance between α and α^*, it will shape α much closer to the desirable distribution. Here the distance metric needs to be design with cautious, because for we do not want to punish too harsh on $\alpha^* = 0$,

$$\lambda = \begin{cases} \lambda_1, & \textit{if } \alpha \neq 0, \\ \lambda_2, & \textit{otherwise} \end{cases}, \quad \lambda_1 >> \lambda_2. \tag{6}$$

Putting all together, the loss function on a sample point $[X, S; Y]$ is

$$\mathcal{L} = H(\hat{Y}, Y) + \langle \lambda, \delta\alpha^\top \delta\alpha \rangle \tag{7}$$

with $\delta\alpha = \alpha - \alpha^*$ and $H(\cdot, \cdot)$ being *cross entropy loss*.

Remark: In practice, a threshold ϵ can be set up, and above regularization is only applied if $|S| < \epsilon$.

3.3 Tree-LSTM

Vanilla Tree-LSTM

Instead of using linear structure as in (1), a recursive architecture is more suitable to re-express sentence. In particular, tree recursive neural network can effectively capture the nested nature of sentence. Among those, we choose Tree-LSTM model that has two variants: Child sum Tree LSTM and N-ary Tree LSTM, for this project, we focus on the *Binary Tree LSTM* that works better with constituency parsers. *Figure 3* shows how to convert a sentence to constituency parsing tree and then parameterize the tree into a Binary Tree LSTM.

Traditional LSTM generates a new hidden and cell state from the pair of last time instance, i.e., $(h_{t-1}, c_{t-1}$, and current sequential input x_t[3]. In Binary Tree-LSTM, each unit at node j maintains a hidden state h_j and a cell state

[3]Abuse of notation, $x_t = w_t$, where w_t is word vector.

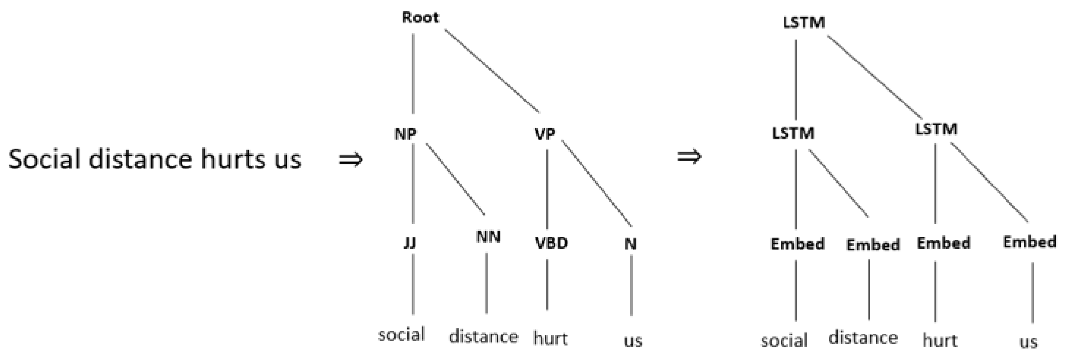


Figure 3: Text to Tree-LSTM

c_j. The unit j takes the input vector x_j and the hidden states of the child nodes: $h_{jl}, l = 1, 2$ as input, then update its hidden state h_j and c_j by

$$i_j = \sigma(W^{(i)}x_j + \sum_{l=1}^{2} U_l^{(i)} h_{jl} + b^{(i)})$$

$$f_{jk} = \sigma(W^{(f)}x_j + \sum_{l=1}^{2} U_{kl}^{(f)} h_{jl} + b^{(f)})$$

$$o_j = \sigma(W^{(o)}x_j + \sum_{l=1}^{2} U_l^{(o)} h_{jl} + b^{(o)})$$

$$u_j = \tanh(W^{(u)}x_j + \sum_{l=1}^{2} U_l^{(u)} h_{jl} + b^{(u)})$$

$$c_j = i_j \bigodot u_j + \sum_{l=1}^{2} f_{jl} \bigodot c_{jl}, \ h_j = o_j \cdot \tanh(c_j)$$

In order to implement batching during training, all trees in a single batch are pooled into a single graph with reordered node ID. Forward propagation and back propagation, i.e message passing, under such framework can be realized under the guidance of the traversal path (traversal order of each tree). In terms of loss function, the root note outputs the probability distribution over the sentiment of the tree, which is compared to the actual label (converted to 1-hot vector). Therefore, we can write:

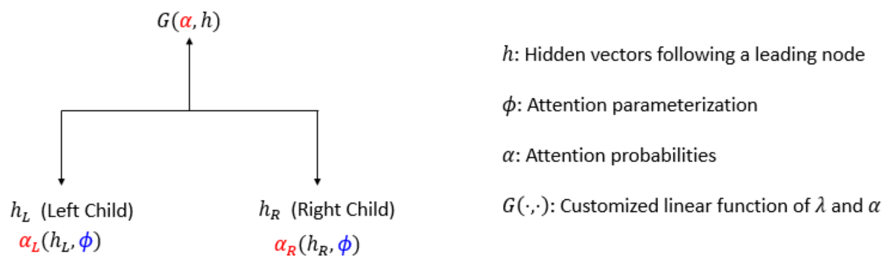
$$\mathcal{L} = H(\hat{y}_R, y_R) \tag{8}$$

where y_R stands for label and $\hat{y}_R$ corresponds to the distribution.

Attentive Tree-LSTM and Enhancement

According to forward propagation for Tree LSTM, it treats every word within a sub tree equally important, while, in reality, sentence are usually analyzed with emphasis on certain words. Similar to attentive LSTM in *Sect. 3.2*, it is natural to equip with self-attention mechanism to avoid uniformity(see *Figure 4*).

For this project, additive self-attention and multiplicative self-attention are both implemented. Borrowing the concept from classical information retrieval theory, the calculating attention involves three matrices: *key*, *query* and *value*, i.e., $(W^{(k)}, W^{(q)}, W^v)$. Denoting $[h_1, h_2]$ (concatenation of hidden states from two children) as M, for the additive



$G(\alpha, h)$
h_L (Left Child)
$\alpha_L(h_L, \phi)$
h_R (Right Child)
$\alpha_R(h_R, \phi)$
h: Hidden vectors following a leading node
ϕ: Attention parameterization
α: Attention probabilities
$G(\cdot,\cdot)$: Customized linear function of λ and α

Figure 4: Attentive Tree-LSTM Subtree

self-attention, the hidden states of parent node are updated as follows:

$$\begin{aligned}
&\text{key} = W^{(k)}M, \ \ \text{query} = W^{(q)}M, \ \ \text{value} = W^{(v)}M \\
&\text{align} = (\text{query})^T\text{key} \cdot \frac{1}{\sqrt{d}}, \ \ d \text{ is a normalizing constant} \\
&\alpha = \text{softmax}(\text{align}), \ \ \tilde{h} = \text{bmm}(\alpha, \text{value})
\end{aligned}$$

As for the multiplicative self-attention, the hidden states of parent node are updated via:

$$\begin{aligned}
&\text{key} = W^{(k)}M, \ \ \text{query} = W^{(q)}M \\
&\text{align} = (\text{query})^T\text{key} \cdot \frac{1}{\sqrt{d}}, \ \ d \text{ is a normalizing constant} \\
&\alpha = \text{softmax}(\text{align}), \ \hat{h} = \text{bmm}(\alpha, M), \ \tilde{h} = \tanh(W^{(v)}\hat{h} + b)
\end{aligned}$$

To strength the attention as we did in linear LSTM model, we seek for other benchmark in a data point. Fortunately, *Stanford CoreNLP* is trained not only with root sentiment but also phrase/word level sentiment. In other words, the transcribed constituency parsing tree has all nodes labelled with sentiments. We hope the introduction of multi-layer labelling could benefit the training and strength the attention. For this purpose, we re-define our loss function as:

$$\text{Loss} = H(\hat{y}_R, y_R) + \lambda H(\hat{y}_N, y_N) \tag{9}$$

Notice the first cross entropy loss gauges the error from root label, while the second one accounts for the rest of nodes. The λ adjuster is of the same spirit as in (7), it allows a fine control of the extent of punishment over these nodes. Since the main objective is to predict the root sentiment, we put a relative low λ so that the focus is not shifted.

4 Experiment Design

In this project, there are two kinds of datasets: Kaggle twitter data used to pre-train the classification models and COVID 19 streaming data for fine-tuning. The schema of Kaggle twitter data reads:

SCHEMA = [TextId, Text, Selected-Text, Sentiment]

where sentiment is taken from {neural, positive, negative}. The statistic summary is presented in *Table* **??**.

	Trainig	Testing
Count	27,479	3,535
Posititive	28%	28%
Neutral	40%	40%
Negative	32%	31%

Table 1: Kaggle Data Statistics Summary

Observe among three labels, tweets of neutral sentiment shows a higher percentage. We also notice the Kaggle data has an additional column *Selected-Text* that can be used for training with method (7). To set the threshold ϵ, we plot the histogram of ratio between length of selected-text over length of text:

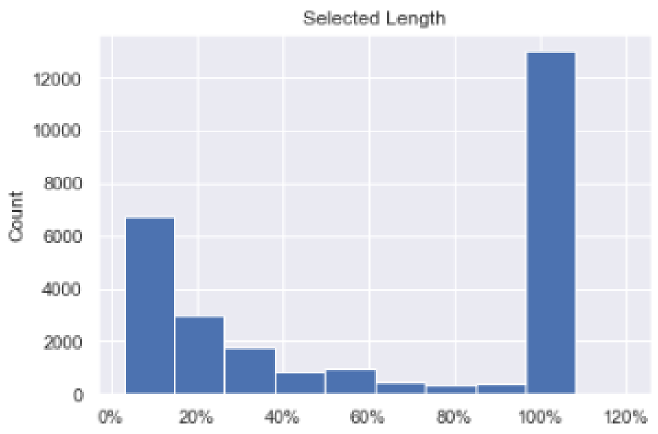


Figure 5: x-Axis is the ratio $= |\textit{Selected-Text}|/|\textit{Text}|$

As we can see, there is a spike around 100% and the rest is concentrated around $10\% \sim 40\%$. Obviously, the higher the overlap between text and selected-text, the less information it carries. We finally set the threshold at 40%, which mean only these tweets have the extra loss while learning.

The live COVID19 data is of streaming fashion, as mentioned before, we periodically sampling from the streaming for model re-calibration. For demonstration purpose, we collect a dataset from April 1st to April 20th, 2020. To narrow the scope, we filtered with pre-defined key words, such as, ['COVID19', 'Mask', 'lockdown', 'staysafestayhome', ...,]. In terms of sampling, we adopted stratified method, i.e., extracting data proportionally based on the amount of tweets within each time bin(see *Figure 6*)

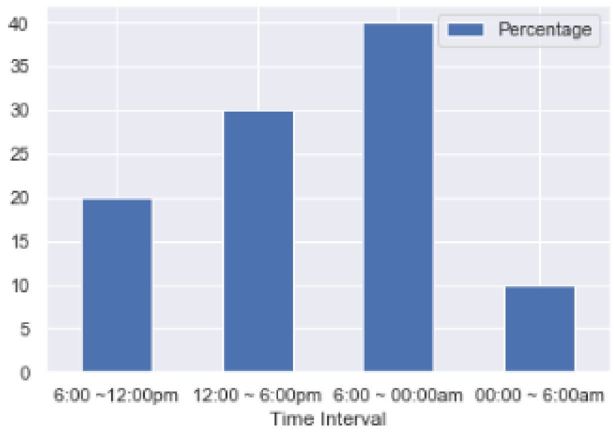


Figure 6: Stratified Sampling

As for streaming COVID19 tweets data, one critical step is to adding labels because it is a necessity for training phase. Human labeling is apparently very laborious and also subjective. To have efficient labelling process, we deployed three standard sentiment analysis agents:

- *Stanford CoreNLP:* built with recursive neural network, trained on treebank;
- *IBM Watson NLP Understanding API:* built with recurrent neural network, trained on tweets;
- *VADER:* rule -based sentiment classifier, trained on social media text;

For each tweets, we have three judgements from the agents, the final label is determined by the majority. In case tie occurs, human interaction is required. Usually, the tie happens very infrequently, which is also the observation from our dataset ($\sim 5\%$, and *Figure 7*).

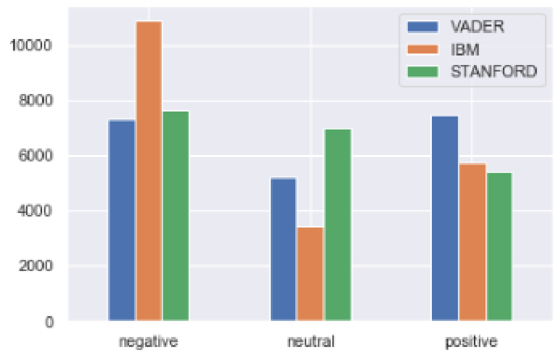


Figure 7: Three Agents Labeling

5 Result Analysis

5.1 Pre-training with Kaggle Data

With attentive Bi-Direction LSTM, we experimented with different size of hidden/cell layer and depth of the network. The test data results is summarized in table below(after 30 epochs, results is subject to truncation error):

Hidden Size	Num Layers	Loss (Att'n)	Accuracy (Att'n)	Loss(Att'n+)	Accuracy (Att'n+)
100	2	0.64	75.5%	0.58	76.3%
200	2	0.62	75.7%	0.55	77.1%
300	2	0.65	75.6%	0.57	76.4%
400	2	0.67	75.4%	0.57	76.4%
500	2	0.68	75.3%	0.56	76.3%
100	4	0.61	75.7%	0.55	77.2%
200	4	0.59	76.1%	0.52	78.2%
300	4	0.62	75.7%	0.53	78.0%
400	4	0.61	75.8 %	0.55	77.3%
500	4	0.63	75.6%	0.54	77.2%

Table 2: Bi-Direction LSTM with Attention(Att'n) & Enhanced(Att'n+)

The best performance is achieved when we have hidden size as 200 and 2-layer bidrection LSTM(highlighted blue). Observe in general, the enhanced version, i.e., Att'n+, score high accuracy across all configurations.

Similarly, we attempted set of configurations for Tree-LSTM, the variables are hidden/cells size of LSTM unit.

Hidden Size	Loss (Att'n)	Accuracy (Att'n)	Loss(Att'n+)	Accuracy (Att'n+)
100	0.50	78.7 %	0.48	79.5%
200	0.48	79.4%	0.46	80.0%
300	0.49	79.1%	0.47	79.7%
400	0.51	78.4%	0.47	79.6%
500	0.49	79.9%	0.49	79.3%

Table 3: Tree-LSTM with Attention(Att'n) & Enhanced(Att'n+)

The Tree-LSTM outperforms Bi-Direction LSTM in almost all cases. From a slightly different angle, if we compare the best Tree-LSTM and Bi-Direction LSTM on the class level, we found the neutral class is relatively harder class for both classifier to predict. In *Figure 8*, we examine the convergence on training and validation data set. The observation is:

1. The Tree-LSTM starts at a higher accuracy and convergence faster than Bi-Drection LSTM;

2. The Tree-LSTM has relatively stable trajectory, or, the Bi-Direction LSTM exhibits more fluctuations towards convergence.

	Bi-Direction LSTM(Att'n+)	TreeLSTM (Att'n+)
Negative	82.4%	84.1%
Neutral	69.3%	70.2%
Positive	81.7%	83.2%

Table 4: Precision Comparison on Class Level

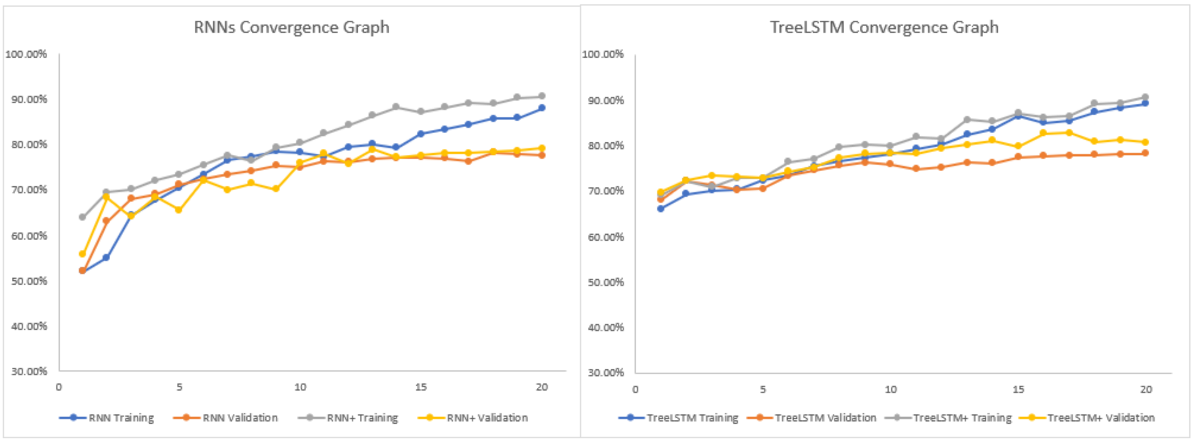


Figure 8: Convergence Graph: Tree-LSTM v.s. Bi-Direction LSTM

5.2 COVID19 Calibration

Using the best configuration of Tree-LSTM and Bi-Direction LSTM colluded from above, we re-trained the model with COVID19 tweets, respectively. We obtained consistent results:

	Att'n	Att'n+
Bi-Direction LSTM	79.2 %	80.1%
TreeLSTM	82.3%	83.4%

Table 5: Accuracy On COVID19 Test Data

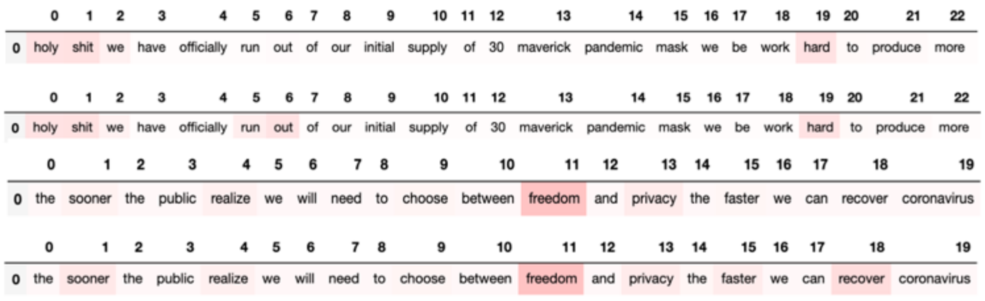
It seems attention mechanism, especially with Tree-Structure, improves the accuracy of sentiment prediction. Indeed, let us give two examples to demonstrate attention is indeed contributing:

Negative: *Holy shit we have official run out of our initial supply of 30 maverick pandemic mask we be work hard to produce more ...*

Positive: *the sooner the public realize we will need to choose between freedom and privacy the faster we can recover coronavirus*

6 Conclusion & Further Research

In this project, we studied the effectiveness of Tree-LSTM on sentiment analysis task. Compared to recurrent neural network, it has better accuracy and faster/stable convergence. Our main contribution is an innovation term in the loss function to boost learning as well as prediction. In addition, we discussed the potential to deploy the sentiment classifier in production environment. The key to ensure quality of prediction is to periodically fine-tuning the model against the labelled agreed by standard agents.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0	holy	shit	we	have	officially	run	out	of	our	initial	supply	of	30	maverick	pandemic	mask	we	be	work	hard	to	produce	more

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0	holy	shit	we	have	officially	run	out	of	our	initial	supply	of	30	maverick	pandemic	mask	we	be	work	hard	to	produce	more

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	the	sooner	the	public	realize	we	will	need	to	choose	between	freedom	and	privacy	the	faster	we	can	recover	coronavirus

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	the	sooner	the	public	realize	we	will	need	to	choose	between	freedom	and	privacy	the	faster	we	can	recover	coronavirus

As for future research, we can potentially work on the following aspects:

1. accommodating the selected text into Tree-LSTM learning. Similar to the enhanced version for Bi-Direction LSTM, a benchmark attention can be imposed in the leave level. This ideally should strengthen the attention across all nodes, as the forward/backward propagation traversal the whole tree.
2. the standard agents could be out-of-date while dealing with innovations in tweets. Instead, we can use unsupervised learning model, e.g., clustering method, to form multi-classes based on extracted feature from text. Combined with agents, it could have more reasonable labelings.

7 Contribution

Lun Li and Shengjie Liu both contributed to writing the reports and the code. Lun Li finished data cleaning, data labeling, created streaming data pipeline with applications, and built the baseline RNN with attention and an enhanced version. Shengjie Liu built the tree LSTM model, added self-attention mechanism and modified the loss function.

References

[1] N. Parmar A. Vaswani, N. Shazeer. Attention is all you need. In *31st Conference on Neural Information Processing System*, 2017.

[2] S. Vaithyanathan B. Pang, L. Lee. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86, 2002.

[3] Y. Bengio D. Bahdanau, K. Cho. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR*, 2015.

[4] C.D. Manning D. Chen. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference On Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, 2014.

[5] C.D. Manning J. Pennington, R. Socher. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

[6] C.D. Manning K.S. Tai. R. Socher. Improved semantic representations from tree-structured long short-term memory networks. *Computer Science*, 5(1):36, 2015.

[7] R.E. Mercer M. Ahmed, M.R. Samee. Improving tree-lstm with tree attention. In *13th IEEE International Conference on Semantic Computing*, 2019.

[8] C.D. Manning A. Ng R. Socher, J. Bauer. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computation Linguistics*, pages 445–465, 2013.

[9] K. Chen J. Dean T. Mikolov, G. Corrado. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations*, 2013.

[10] P. Guo X. Zhu. Long short-term memory over tree structure. *arXiv: 1503.04881*, 2015.

[11] Y. Pan Y. Zhou, C. Liu. Modelling sentence pairs with tree-structured attentive encoder. In *arXiv:1610.0826*, 2018.