Attention is Crucial for Targeted Aspect-Based Sentiment Analysis

1 BACKGROUND

Sentiment analysis (SA) is an important task in natural language processing, it detects the overall polarity of an unstructured comment and understand people's emotions. However, people tend to express their opinions with different aspects, in which case SA may not uncover the fine-grained emotions. The task of aspect-based sentiment analysis (ABSA) aims to identify fine-grained polarity towards a specific aspect.

Both SA and ABSA are sentence-level tasks, but one comment may refer to more than one object, and sentence-level tasks cannot handle sentences with multiple targets. Therefore, Saeidi et al. [1] introduce the task of targeted aspect-based sentiment analysis (TABSA), which aims to identify fine-grained opinion polarity towards a specific aspect associated with a given target. Instead of the stack of comments, we can view the comments as the target-aspect pairs, and then analyze the sentiment.

Sun C, Huang L et al. have put forward a method to handle TABSA problem by fine-tuning BERT model. [2] They focus on solving the task of ABSA and TABSA by constructing an auxiliary sentence from the aspect and converting it to a sentence-pair classification task, such as question answering (QA) and natural language inference (NLI). It fine-tunes the pretrained BERT model and compares experimental results of single sentence classification and sentence-pair classification.

We investigate and compare three main algorithms, LSTM, Attention, and BERT, referring to the methodology in the paper[2] that constructing an auxiliary sentence and transforming TABSA into a sentence-pair classification task. In addition to implementing the algorithms, we also extend these basic algorithms to see if we can achieve better results. We extend LSTM to GRU and Target-Dependent LSTM, and extend BERT to a new algorithm, XLNET, which is also a Transformer based model but overcomes the limitations of BERT through its autoregressive formulation. BERT and XLNET both achieve new state-of-the-art results on TABSA task.

2 DATESET

We evaluate our method on the SentiHood (Saeidi et al., 2016)[1] dataset, and its theme is restaurant sentiment analysis. The dataset consists of 5,215 sentences, 3,862 of which contain a single target, and the remainder multiple targets. The dataset is divided into training set, validation set and test set, and the ratio is about 4:1:2.

As we can see in Table 1, each sentence contains a list of target-aspect pairs $\{(t, a)\}$, where that each t is denoted by location − 1, location − 2 in the sentence and a is belongs to a fixed aspect set $A = \{\text{general, price, transit, safety}\}$. For every target-aspect pair of a sentence, there is a sentiment polarity, which is Positive, None or Negative. So we can think of our TABSA as a sentence-pair classification task, what we need to do is to predict the sentiment polarity.

Table 1: Example of Dataset based on Single Sentence

Sentence_1	Sentence_2	Label
i always consider location - 2 to be great value for money - cheap rents in location - 1 , decent public transport and good local amenities	location - 1 - general	None
i always consider location - 2 to be great value for money - cheap rents in location - 1 , decent public transport and good local amenities	location - 1 - price	Positive
i always consider location - 2 to be great value for money - cheap rents in location - 1 , decent public transport and good local amenities	location - 1 - safety	None
i always consider location - 2 to be great value for money - cheap rents in location - 1 , decent public transport and good local amenities	location - 1 - transit	Positive

3 MODEL

3.1 Word Embedding

3.1.1 *Pretrained Word Embedding*

The first step for deep learning applications in practical problems such as Computer Vision or Natural Language Processing is to convert data into a form that neural networks are able to recognize. In our ABSA scenario, we have to convert words in sentence into a format such as vectors and feed them into models. We adopt GLoVe as pretrained word embedding.[3] GLoVe is trained on the basis of word co-occurrences in large corpus such as Wikipedia articles. Two versions of GLoVe are in the scope of our discussion: one is GLoVe.6B.300d, trained on 6 billion tokens with each embedding dimension = 300; another is GLoVe.840B.300d, trained on 840 billion tokens with each embedding dimension = 300.

3.1.2 *Word-Aspect Embedding*

It is noticeable that normal LSTM/GRU cannot handle different aspects, and suffers from the issue of aspect ignorance. We have to figure out a way to insert information of aspects. We can append aspect's word embedding vector behind word embedding vector of each word in sentence_1 to get word-aspect embedding. It is simple to achieve because each aspect is just one word. In this way, the model is able to distinguish different aspects for the same sentence to achieve the goal of aspect-specific sentiment prediction. The

length of word-aspect embedding vectors are twice as much as normal word embedding vectors.

3.2 Recurrent Neural Networks

3.2.1 GRU

The GRU model can be regarded as a baseline model for our problem, because it is almost the simplest but relatively effective model for sentiment analysis. We first create Embedding instance from pretrained weights. Then we add a layer of gated recurrent unit (GRU) RNN with 300 features in the hidden state. Finally, we apply a linear transformation and output polarities of 3 dimensions.

3.2.2 Target-Dependent LSTM

Target-Dependent LSTM(TD-LSTM) is a variant of normal LSTM.[7] We use two LSTM neural networks, a left one $LSTM_L$ and a right one $LSTM_R$, to model the preceding and following contexts respectively. The $LSTM_L$ ends with target string and the $LSTM_R$ starts with target string. The outputs of two LSTMs are concatenated and delivered to a Softmax function layer. It mainly solves the problem that LSTM usually use the same feature representations for different targets (since they share the same input chain).

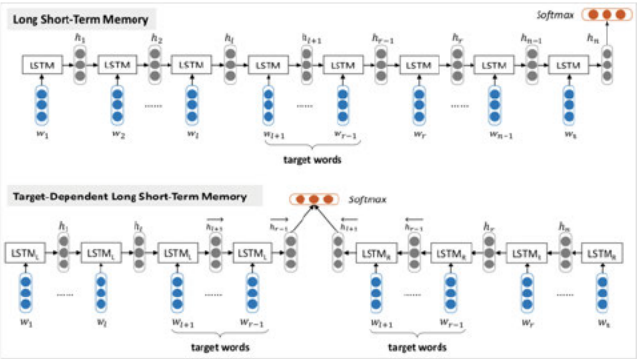


Figure 1: Target-Dependent LSTM Model[7]

3.3 Attention

3.3.1 Why We Need Attention?

Attention model is widely used in other NLP tasks such as machine translation. This is because the language property in different language. For example, we always put adverbial of time at the end of the sentence in English, while time is usually declared at the beginning in Chinese. If we adopt LSTM or GRU to parse the sentence, it is highly possible that the gradient has been vanished when the stream reaches adverbial of time and it loses the information of subjective and predicates in the beginning of the sentence in Chinese. Therefore, the idea of attention mechanism should also be practical in TABSA scenario, because the precise positioning of key information and object in different locations of sentence are essential for sentiment analysis. [4]

Attention Model is helpful to overcome the difficulty. It consists of Encoder and Decoder (Figure 2). Encoder part accepts original sentence (sentence_1), process and pass useful information

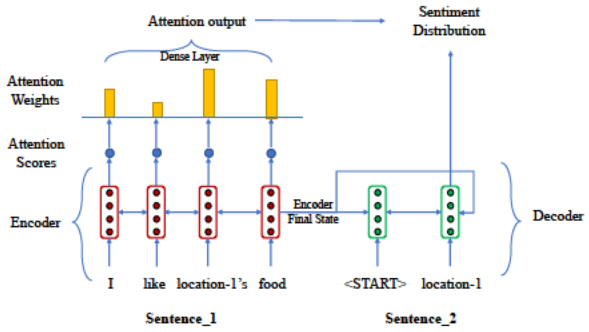


Figure 2: Encoder-Decoder Structure in Attention Model

into Decoder. Decoder part accepts auxiliary sentence (sentence_2), combine both the input sentence and initial state from Encoder to process and output a probability distribution of 3-label sentiment. Attention mechanism ensures all words in encoder visible to all words in decoder, and give a different weight for each location. In our scenario, attention mechanism is also helpful for us to locate our target, because 'location-i' appears both in Encoder input and Decoder input and they communicate by attention mechanism in the model. As is illustrated in Figure 3, the word "cool" is highly positive sentiment word in the sentence, and it has a dominant attention weight compared with other words.

3.3.2 Attention Model Formula

Given a sentence in the source language, we look up the word embeddings from GLoVe embeddings. We define m is the length of the source sentence and e is the embedding size. We feed these embeddings to the bidirectional Encoder, yielding hidden states and cell states for both the forwards ($\rightarrow$) and backwards ($\leftarrow$) LSTMs. The forwards and backwards versions are concatenated to give hidden states $\mathbf{h}_i^{\text{enc}}$ and cell states $\mathbf{c}_i^{\text{enc}}$:

$$\mathbf{h}_i^{\text{enc}} = \left[\overleftarrow{\mathbf{h}_i^{\text{enc}}}; \overrightarrow{\mathbf{h}_i^{\text{enc}}}\right] \text{ where } \mathbf{h}_i^{\text{enc}} \in \mathbb{R}^{2h\times 1}, \overleftarrow{\mathbf{h}_i^{\text{enc}}}, \overrightarrow{\mathbf{h}_i^{\text{enc}}} \in \mathbb{R}^{h\times 1},$$

$$\mathbf{c}_i^{\text{enc}} = \left[\overleftarrow{\mathbf{c}_i^{\text{enc}}}; \overrightarrow{\mathbf{c}_i^{\text{enc}}}\right] \text{ where } \mathbf{c}_i^{\text{enc}} \in \mathbb{R}^{2h\times 1}, \overleftarrow{\mathbf{c}_i^{\text{enc}}}, \overrightarrow{\mathbf{c}_i^{\text{enc}}} \in \mathbb{R}^{h\times 1},$$

where $1 \le i \le m$. In our experiment, $h = 128$.

After Encoder, the basic structure of Decoder is also bidirectional LSTM. To obtain the initial vector of forwards and backwards of hidden state and cell state, final hidden state and final cell state in the Encoder are put into dense layer respectively. Assume that k is the length of auxiliary sentence, then:

$$\overrightarrow{\mathbf{h}_0^{\text{dec}}} = \mathbf{W}_{h_f}\left[\overleftarrow{\mathbf{h}_1^{\text{enc}}}; \overrightarrow{\mathbf{h}_m^{\text{enc}}}\right] \text{ where } \mathbf{h}_0^{\text{dec}} \in \mathbb{R}^{h\times 1}, \mathbf{W}_h \in \mathbb{R}^{h\times 2h}$$

$$\overrightarrow{\mathbf{c}_0^{\text{dec}}} = \mathbf{W}_{c_f}\left[\overleftarrow{\mathbf{c}_1^{\text{enc}}}; \overrightarrow{\mathbf{c}_m^{\text{enc}}}\right] \text{ where } \mathbf{c}_0^{\text{dec}} \in \mathbb{R}^{h\times 1}, \mathbf{W}_c \in \mathbb{R}^{h\times 2h}$$

$$\overleftarrow{\mathbf{h}_{k+1}^{\text{dec}}} = \mathbf{W}_{h_b}\left[\overleftarrow{\mathbf{h}_1^{\text{enc}}}; \overrightarrow{\mathbf{h}_m^{\text{enc}}}\right] \text{ where } \mathbf{h}_0^{\text{dec}} \in \mathbb{R}^{h\times 1}, \mathbf{W}_h \in \mathbb{R}^{h\times 2h}$$

$$\overleftarrow{\mathbf{c}_{k+1}^{\text{dec}}} = \mathbf{W}_{c_b}\left[\overleftarrow{\mathbf{c}_1^{\text{enc}}}; \overrightarrow{\mathbf{c}_m^{\text{enc}}}\right] \text{ where } \mathbf{c}_0^{\text{dec}} \in \mathbb{R}^{h\times 1}, \mathbf{W}_c \in \mathbb{R}^{h\times 2h}$$

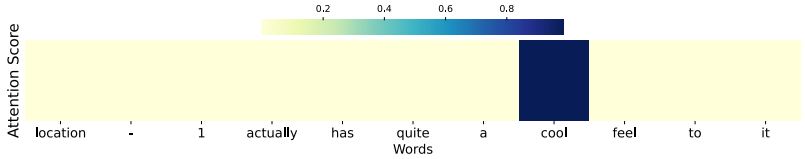


Figure 3: Example of Attention Scores in Sentence

As for the attention part, we then introduce a dense layer to build a bridge from each state in Encoder to Decoder:

$$\mathbf{e}_{t,i} = \left(\mathbf{h}_t^{\text{dec}}\right)^T \mathbf{W}_{\text{attProj}}\, \mathbf{h}_i^{\text{enc}} \quad \text{where } \mathbf{e}_t \in \mathbb{R}^{m\times 1}, \mathbf{W}_{\text{attProj}} \in \mathbb{R}^{h\times 2h}$$

$$\alpha_t = \text{Softmax}\left(\mathbf{e}_t\right) \quad \text{where } \alpha_t \in \mathbb{R}^{m\times 1}$$

$$\mathbf{a}_t = \sum_i^m \alpha_{t,i} \mathbf{h}_i^{\text{enc}} \quad \text{where } \mathbf{a}_t \in \mathbb{R}^{2h\times 1}$$

We now concatenate the attention output $\mathbf{a}_t$ with the decoder hidden state $\mathbf{h}_t^{\text{dec}}$.

$$\mathbf{h}_t^{\text{dec}} = \left[\overleftarrow{\mathbf{h}_t^{\text{enc}}}; \overrightarrow{\mathbf{h}_t^{\text{enc}}}\right] \in \mathbb{R}^{2h\times 1}$$

$$\mathbf{u}_t = \left[\mathbf{a}_t; \mathbf{h}_t^{\text{dec}}\right] \quad \text{where } \mathbf{u}_t \in \mathbb{R}^{4h\times 1}$$

where $1 \le t \le k$.

From then on, the meaning of auxiliary sentence and attention values of original sentence have been integrated. After adopting dense layer and Softmax, 3-label sentiment probability distribution is computed as output and we extract the argmax of the probability distribution to determine the sentiment of the sentence pair.

3.4 Transformer-based Models

3.4.1 BERT

BERT(Bidirectional Encoder Representation of Transformer), a giant model which is the encoder part of Transformer.[6] Transformer also consists of Encoder and Decoder, but the structure is more sophisticated than that of attention model as we have introduced before.[5] Both Encoder and Decoder are stacked with several layers containing self-attention sub-layers, which achieve words communication within a single sentence by special dense layers representing queries, keys and values. The advantage of Transformer over RNN is that it gets rid of recurrent units in which next status has to wait for previous status, so that Transformer supports more sufficient parallel computing. On the basis of this property, Google use unsupervised tasks, Masked Language Model and Next Sentence Prediction, to train a Transformer and extract its Encoder part as BERT pretrained model. In this project, we use BERT-base (layers number = 12, hidden size = 768) and BERT-large (layers number = 24, hidden size = 1024) to obtain sentence pair embedding, and extend it by a dense layer to compute sentiment distribution.

Apart from the ability to pretrain from huge corpus to summarize more general property of a language, BERT also has advantages in other aspects.

- Target/Aspect detection: In attention model, we manually introduce additional dense layers to build a bridge from Encoder to Decoder. We also introduce aspect embedding previously to solve the problem of aspect ignorance. However, in BERT pretrained model, self-attention layers in BERT Encoder layer have been furnished with the ability of words communication, so we only need to preserve the structure of sentence_2 (e.g. location – 2 - safety), concatenate sentence_1 and sentence_2 by '[SEP]' which Segment Layer and Position Layer of BERT Embedding Layer are able to recognize, and adopt all sentence pairs as input. It extremely facilitates our implementation.
- Unseen Words handling: When using GLoVe embedding as pretrained word embedding, all unseen words are represented by a general vector, such as the average of all word embedding vectors in the dictionary. Nevertheless, in BERT embedding layer, it supports sub-word embedding to decompose unseen words into several fine parts such as prefix, suffix, stems, etc., which are in the domain of BERT embedding dictionary. BERT embedding dictionary is a mapping utilized in Token Layer of BERT Embedding Layer in order to pick up the embedding vector for each sub-word. In this case, BERT is able to find a tricky way to represent unseen words and speculate their meaning as mush as possible.

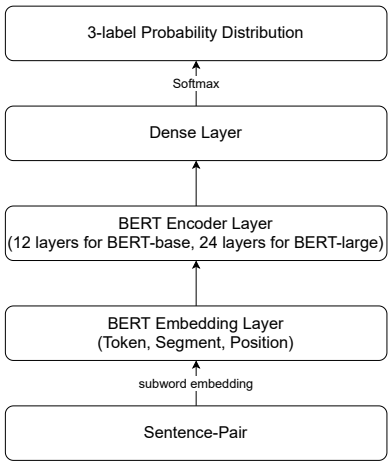


Figure 4: BERT Application in TABSA Problem

3.4.2 XLNet

XLNet is also based on transformer.[8] But unlike BERT as a auto-encoder language model(AE LM), XLNet is a auto-regressive language model(AR LM). An auto-regressive language model tries to use the context before word T_i to predict T_i, while an auto-encoder language model tries to use context around the word T_i to predict

T_i. Thus BERT can use Masked LM technique for training, and use context information before and after the masked word. Unlike traditional AR LM that can only use one direction context, XLNet uses Permutation Language Modeling and Two-Stream Self-Attention to gather both context information before and after the certain word (Figure 5).

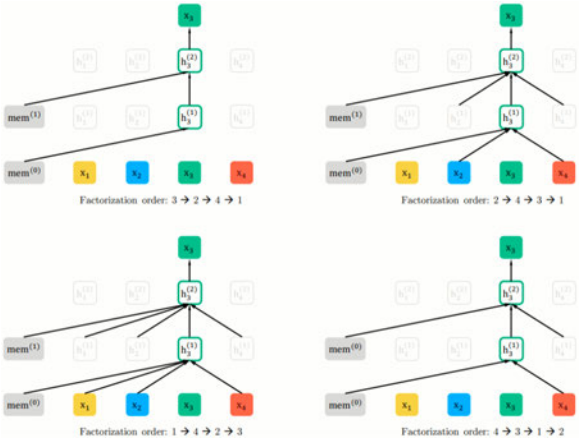


Figure 5: XLNet Permutation Language Modeling[8]

Another improvement from XLNet is that XLNet uses Segment Recurrence Mechanism, which enables the model to use hidden states from previous segments. This method makes the model capture long texts (longer than the segment length which is usually 512, and thus should be splitted into multiple segments) context information between different segment and allows XLNet to handle long text better than BERT.

XLNet also supports sentence pairs input, and the input is the same as BERTs' [CLS, A, SEP, B, SEP], where CLS and SEP are special tokens and A B are two segment. Specially, we use the implementation by Huggingface, and it uses special token string '<cls>' and '<sep>'. Therefore, XLNet shares all advantages of BERT and possesses its own strength.

4 EXPERIMENT

We have conducted sufficient numeric experiment on all models that we have introduced in previous sections. Due to different properties and structure complexity, the optimal choice of hyperparameters such as learning rate, batch size, etc., are also different among all models. We will also discuss model performance on some important hyperparameters by further experiment for more detailed exploration.

4.1 Training Strategy

Hyperparameters tuning in deep learning models is crucial, because the loss function for neural networks to optimize is usually a highly complicated non-convex function, and the quality of local minimum that optimization algorithm converges to depends on the choice of hyperparameters. After manually adjusting different groups of hyperparameters, we almost figure out the optimal learning rate, batch size and epoch for each model to reach a satisfying solution as efficient as possible, which have been listed in Table 2. The model selection strategy is that we select the model with lowest loss on development set from all epochs.

Importantly, we adapt weighted sampler to give different weight for different classes in order to handle data imbalance when training GRU, TD-LSTM and Attention-LSTM, because the ratio for 'None' label is more than 60%.

Table 2: Hyperparameters to Reach Best Results

Model	Learning Rate	Batch Size	Epoch
GRU	0.001	16	20
TD-LSTM	0.001	16	40
Attention-LSTM	0.001	2	50
BERT	10^{-5}	16	4
XLNet	10^{-5}	2	20

4.2 Best Results

The best result for each model by hyperparameters in Table 2 is listed in Table 3. We use accuracy and macro F-1 score to measure the performance of all models. Accuracy represents true prediction hit rate, while macro F-1 score summarize a more general prediction performance, which assesses the quality of problems with multiple classes, like the 3-label sentiment prediction in our TABSA problem, by taking false positive and false negative cases into account.

Table 3: Best Results of All Models

Model	Accuracy	Macro-F1
GRU	0.87	0.61
TD-LSTM	0.89	0.69
Attention-LSTM	0.90	0.75
BERT-Base	0.92	0.80
BERT-Large	**0.94**	0.85
XLNet	**0.94**	**0.86**

Among all the models, the basic GRU model performs worst. This is not surprising because the TABSA task needs to identify fine-grained opinion polarity towards a specific aspect, while the basic GRU model does not capture any target and aspect information. TD-LSTM obtains improvement over GRU when target signals are taken into consideration. It models the preceding and following contexts surrounding the target string, so that contexts in both directions could be used as feature representations for sentiment analysis.

Moreover, we observe that all models practicing attention mechanism (normal attention or self-attention), such as Attention-LSTM, BERT and XLNet, perform better than RNN-based models. It is essential to point out that normal attention mechanism adopts a more intuitive technique, and achieves a slightly better performance than TD-LSTM, which requires more computation. Therefore, attention mechanism should be not only helpful for long text processing, but also practical for key information positioning. It should be an economical option to incline to attention mechanism for problems related to target locating.

It is obvious that transformer-based models achieve the best performance on the TABSA problem. BERT-large takes advantage of BERT-based due to its twice larger structure, and XLNet performs slightly better than BERT-large on macro F-1.

4.3 More Observation on Batch Size

During implementation, the effect of batch size on model training arouses our interests. According to our existing computing capacity, we select TD-LSTM, Attention-LSTM and BERT as the our targets to conduct further experiment.

Here is an interesting result in TD-LSTM, when we changing the batch size, the accuracy of test set does not have huge difference, but F1-score has a huge decrease with batch size increase from 8 to 128, it achieve its best performance around 8 or 16. What's more, with a smaller batch size, it has a better prediction on negative and positive polarities. While, a larger batch size does good on neutral polarity prediction. In general, small batch sizes performs well, but it has limited information among pairs . So to obtain both advantages, it maybe a good idea to shuffle our dataset in some degrees and train model with small batch size.

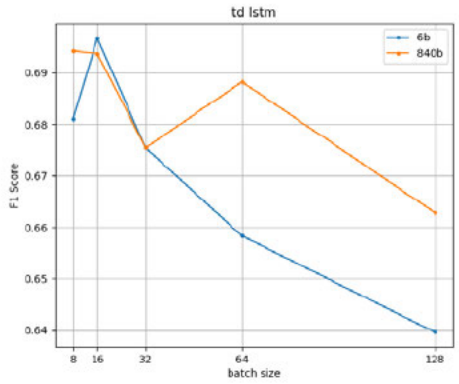
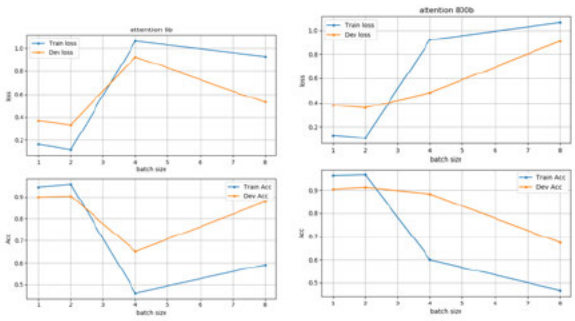


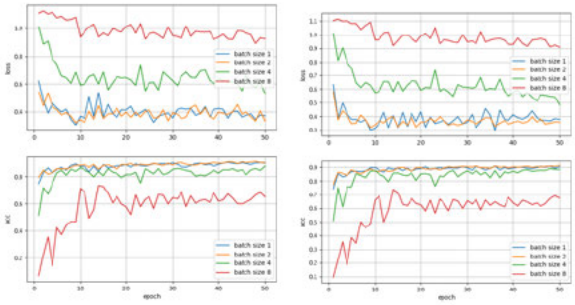
Figure 6: F-1 Scores of TD-LSTM

As for Attention-LSTM model, we use loss and accuracy on training set and development set to show the influence of batch size in Figure 7 and illustrate metrics progress along with epochs on development set in Figure 8. It is obvious that small batch sizes such as 1 and 2 achieve significantly better results than larger batch sizes 4 and 8. At the beginning we doubt that it suffers from underfitting when batch size is large, because on the same level of epoch number, larger batch size means less iterations of loss descent in optimization algorithm. However, we have executed additional experiment when batch size = 8 and epoch = 400, but the accuracy on development set still wanders around 70%, which is still worse than cases when batch size is smaller. It reveals that our implementation of Attention-LSTM prefers small batch size to achieve satisfying prediction effect.



(a) GLoVe.6B **(b) GLoVe.840B**

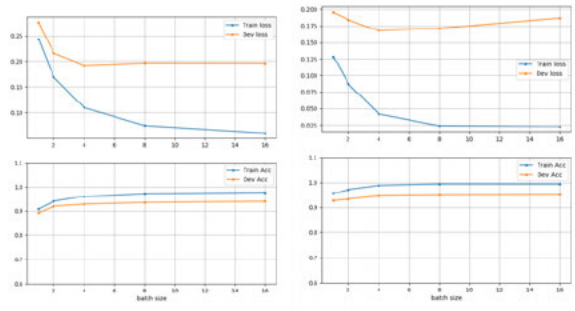
Figure 7: Results of Attention-LSTM



(a) GLoVe.6B **(b) GLoVe.840B**

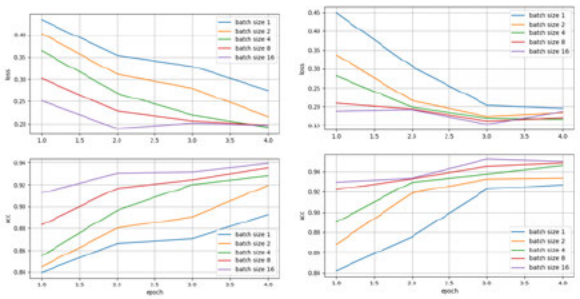
Figure 8: Training Process of Attention-LSTM

The last part is further experiment on BERT model. According to GitHub official repository of BERT by Google [9], it mentions that batch size recommendations for BERT is 16, 32, 64. In our experiment, we have attempted on smaller batch sizes 1, 2, 4, 8, 16. From Figure 9 and Figure 10, we witness that BERT truly prefers larger batch size to contain more general information when updating parameters.



(a) GLoVe.6B **(b) GLoVe.840B**

Figure 9: Results of BERT



(a) base (b) large

Figure 10: Training Process of BERT

All of further experiment above vividly points out that the choice of batch size should be highly depend on the conditions of different models. Though the results are intriguing, it still requires more efforts to explain the reasons in mathematics theory.

5 CONCLUSION

In this project, we transform a Targeted Aspect-Based Sentiment Analysis problem from a single sentence classification task to a sentence pair classification task. We tried three main types of algorithms and did a lot of extensions on them. We trained all models, compared the experimental results and attempt to analyze the relationship between batch size and model performance. We conclude that the pretrained BERT model and XLNet obtained the new state-of-the-art results, and attention mechanism should be practical for location-specific tasks such as TABSA problem. In the future, we will extend our practice to diverse datasets.

REFERENCES

[1] Marzieh Saeidi, Guillaume Bouchard, Maria Liakata, and Sebastian Riedel. 2016. Sentihood: targeted aspect based sentiment analysis dataset for urban neighborhoods. arXiv: 1610.03771.

[2] Sun C, Huang L, Qiu X. Utilizing BERT for aspect-based sentiment analysis via constructing auxiliary sentence[J]. arXiv preprint arXiv:1903.09588, 2019.

[3] Pennington J, Socher R, Manning C D. GLoVe: Global vectors for word representation[C]//Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014: 1532-1543

[4] Wang, Yequan, et al. "Attention-based LSTM for aspect-level sentiment classification." Proceedings of the 2016 conference on empirical methods in natural language processing. 2016.

[5] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. arXiv preprint arXiv:1706.03762, 2017.

[6] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[7] Tang, Duyu, et al. "Effective LSTMs for target-dependent sentiment classification." arXiv preprint arXiv:1512.01100 (2015).

[8] Yang, Zhilin, et al. "Xlnet: Generalized autoregressive pretraining for language understanding." arXiv preprint arXiv:1906.08237 (2019).

[9] https://github.com/google-research/bert