TraceNet: Tracing and Locating the Key Elements in Sentiment Analysis

Anonymous ACL submission

Abstract

We study sentiment analysis task where the outcomes are mainly contributed by a few key elements of the inputs. Motivated by the two-streams hypothesis, we explore processing input items and their weights separately by developing a neural architecture, named TraceNet, to address this type of task. It not only learns discriminative representations for the target task via its encoders, but also traces key elements at the same time via its locators. In TraceNet, both encoders and locators are organized in a layer-wise manner, and a smoothness regularization is employed between adjacent encoder-locator combinations. Moreover, a sparsity constraint is enforced on locators for tracing purposes and items are proactively masked according to the item weights output by locators. A major advantage of TraceNet is that the outcomes are easier to understand, since the most responsible parts of inputs are identified. Also, under the guidance of locators, it is more robust to attacks due to its focus on key elements and the proactive masking training strategy. Experimental results show its effectiveness for sentiment classification. Moreover, we provide several case studies to demonstrate its robustness and interpretability.

1 Introduction

As we all know, in sentiment analysis (SA) task (Chen and Qian, 2019; Johnson and Zhang, 2015; Zhang et al., 2018), its overall sentiment always depends to a large extent on a few key elements of the inputs. For example. Given a short movie review "*deflated ending aside, there's much to recommend the film*" obtained from the SST-5 dataset (detailel in later Section), the three words *deflated*, *much*, and *recommend* have larger impacts on the overall sentiment polarity of the review.

For this type of task, a lesson from attention mechanism (Bahdanau et al., 2015; Vaswani et al., 2017; Velickovic et al., 2018) is worthy of learning, where a weighted sum over all input items is computed. Despite its effectiveness, this strategy remains simple and could not fully reveal nor exploit the unique input structure, *i.e.,* the existence of a few key elements. To be specific, the input structure is **implicitly** modeled, it is unclear whether the structure could enhance the model performance in terms of both prediction effectiveness and, better yet, other promising properties such as evaluation and robustness. Moreover, the importance weights of both attention models are **dense**, as a result of which the key elements are not directly revealed.

To alleviate the above issues and answer the questions, we take one step towards explicitly and separately modeling the input structure. Explicitly means that we explicitly associate each input item with a weight and update the weight during the training. Separately means that the input items and item weights are processed separately. Our work is motivated by the two-streams hypothesis (Goodale et al., 1992), which argues that the neural processing of vision and hearing follows two distinct streams. The ventral stream (a.k.a. "what pathway") is involved with the object and visual identification and recognition, while the dorsal stream (or, "where pathway") is involved with processing the spatial location relative to the viewer and with speech repetition. Such what-and-where decomposition has already shown its usefulness in computer vision (Jacobs et al., 1991; Simonyan and Zisserman, 2014; Wang and Liu, 2018; Zhang et al., 2021) and natural language processing (Zhang and Goldwasser, 2019) tasks. We assume that the input structure, i.e., input items and items importance, can be processed by different pathways and then be mutually reinforced. To implement this, we explore a neural architecture TraceNet, what distinguishes TraceNet from previous ones is that it not only learns discriminative representations, but also traces the key input elements at the same time.

Central to TraceNet are a set of Encoder-Locator Combinations (ELCs) such that encoders and loca-

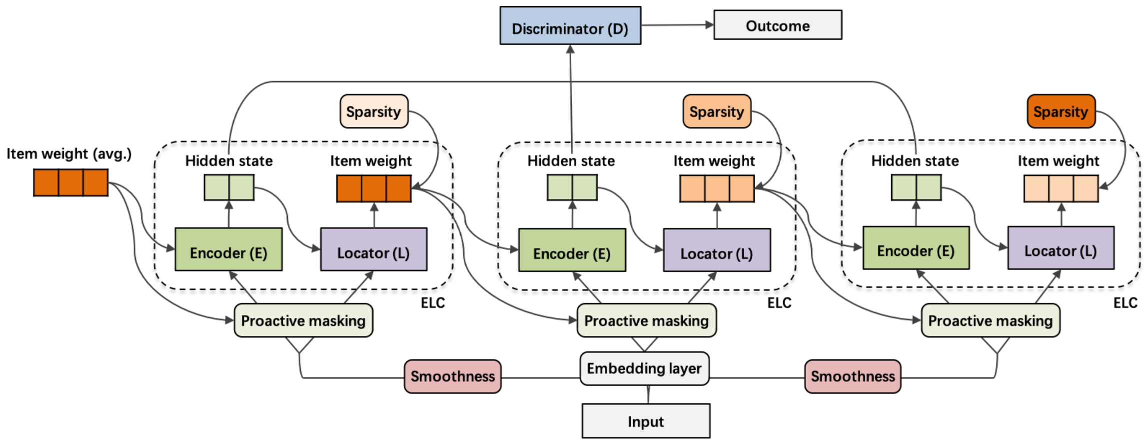


Figure 1: General architecture of TraceNet (hidden size and the number of input items are 2 and 3).

tors are responsible for the "what and where pathways" respectively. TraceNet adopts a layer-wise architecture to organize ELCs, which enables encoders and locators to collaborate for mutual reinforcement between the two sub-tasks, *i.e.,* representation learning and structure revealing. More specifically, locators utilize the hidden states of encoders to estimate item weights more accurately, and encoders are in turn guided by the item weights of locators to obtain more discriminative hidden states. Also, there is a smoothness regularization between the input item embeddings of adjacent ELCs. This is to prevent the hidden states from changing significantly and ensure the stabilization of learning across layers. For the purpose of tracing, TraceNet further enforces sparsity constraints with increasing strength on locators. As a result, locators are taught to identify a small subset of key elements eventually. In addition, TraceNet employs a proactive masking strategy, *i.e.,* proactively masking key elements as indicated by item weights during training. The strategy prevents TraceNet from simply learning feature co-adaption and assists it to resist attacks on key elements.

We exploit TraceNet for SA for evaluation. Experimental results on both sentence- and document-level sentiment classification demonstrate the effectiveness of TraceNet. Notably, despite the large-scale training corpus and many engineering efforts for the state-of-the-art pre-trained language models, TraceNet built upon XLNet and RoBERTa could further increase the classification accuracy over the two. Then, we provide a case study by considering a total of eight types of attacks, and show that TraceNet is more robust to attacks than XLNet, especially on hard attacks such as changing word orders and dropping information. Moreover, our qualitative analysis verifies that the revealed item weights make the outcomes of TraceNet easier to understand. Finally, we conduct several experiments to analyse the parameters sensitivity, e.g., masking probability, number of stacked ELCs and hidden state aggregation in each ELCs.

2 Related Work

Word embedding methods. GloVe (Pennington et al., 2014) performs on aggregating global word-word co-occurrence statistics from a corpus, it is an unsupervised learning algorithm for obtaining vector representations for words and is publicly available. Deep learning models, *e.g.,* convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have already demonstrated their superiority for the task (Cho et al., 2014; Choi et al., 2018; Kim, 2014). Distinct from exploiting the spatial and temporal patterns in texts as done by CNNs and RNNs, TraceNet tackles the problem by considering the special input structure such that the outcome is mainly contributed by a few key elements. Recently, large-scale pre-trained language models (Devlin et al., 2019; Liu et al., 2019; Yang et al., 2019) have further led to significant performance gains on a broad range of NLP tasks. TraceNet is capable of integrating any such effort through its embedding layer, and its contribution is to further enhance model performance by tracing key input elements. While we have also observed a growing trend in aspect-level sentiment analysis (Chen and Qian, 2019; Tang et al., 2019), in this work, we only consider the problem at sentence-level and document-level.

Two-stream hypothesis. (Zhang and Goldwasser, 2019) also borrows the notation from the two-stream hypothesis, where the segmentation tagging task is considered as a "where"-task (i.e., the location of entities), and the sentiment recognition as the "what"-task. The difference between TraceNet and (Zhang and Goldwasser, 2019) is that we separately treat the input items and item weights as "what" and "where", while the latter considers segmentation tagging and sentiment classification and "where" and "what". Since there are very different settings and evaluation datasets are adopted, we do not include it as our baseline.

3 Proposed Model

3.1 General Architecture

As mentioned earlier, we consider SA task whose input can be represented as a set of items, and the corresponding outcome is mainly contributed by a few key items. The proposed model is illustrated in Fig. 1. TraceNet first transforms the item-based input into continuous vector representation in its **embedding layer**. The core of TraceNet is a set of **encoder-locator combinations** (ELCs) organized layer-by-layer, as shown in the vertical-middle part of Fig. 1. Each ELC behaves as a basic functional unit of TraceNet, which jointly learns task-specific representation and reveals input structure. There is a **smoothness regularization** between the input item embeddings of adjacent ELCs. This is to prevent the hidden states from changing significantly and ensure the stabilization of learning across layers. TraceNet further places a **sparsity constraint** on the vector to derive sparse item weights. More specifically, it increases the strength of sparsity constraints on locators layer-by-layer, as shown by the varying colors of the sparsity components in Fig. 1. Since it is generally more challenging to identify key elements at the very beginning, the weaker sparsity constraint allows locators to select more key items for better error tolerance. Then the **proactive masking** strategy masks some input items (*i.e.,* setting the corresponding embeddings to zero) during training to boost model performance. As we describe the masking process as "proactive", it differs from traditional random masking like in BERT (Devlin et al., 2019) in the way that the probability of each item to be masked is given by its item weight. At the top of TraceNet is a **discriminator** D built to derive the corresponding outcome of every given input with respect to the task.

3.2 Input & Embedding Layer

For sentiment analysis, the input can be unified as a sequence of words $S = [w_1, w_2, \dots, w_n]$. The embedding layer could be any pre-trained language models among which BERT (Devlin et al., 2019), XLNet (Yang et al., 2019), and RoBERTa (Liu et al., 2019) are the most effective and popular. As such, each word $w_i \in S$ is transformed into a continuous vector representation $\boldsymbol{x}_i \in \mathbb{R}^{d'}$, d' represent the dimension of embeddings. By stacking these word vectors, we also have the corresponding word embedding matrix $\mathbf{X} \in \mathbb{R}^{n \times d'}$.

3.3 ELC & Sparsity Constraint

For the k-th ELC ($k < 1$), given the masked $\mathbf{C}_{k-1}$ and $\boldsymbol{l}_{k-1}$, the encoder essentially derives the hidden state $\boldsymbol{h}_k \in \mathbb{R}^d$ by summing over rows/words in $\mathbf{C}_{k-1}$ such that those more important are given higher weights. d is the dimension of vector representations. To achieve this, it first computes a query vector $\boldsymbol{q}_k = \boldsymbol{l}_{k-1}^{\intercal}\mathbf{C}_{k-1}$, which encodes key items in the current ELC based on the (sparse) item weights in $\boldsymbol{l}_{k-1}$. Thus, the query vector $\boldsymbol{q}_k$ could determine which words the encoder should pay more attention to. The hidden state $\boldsymbol{h}_k$ is then outputted by an attention layer, given $\boldsymbol{q}_k$ as query and rows in $\mathbf{C}_{k-1}$ as keys/values. Formally, the unnormalized attention weights are given by:

$$a(\boldsymbol{q}_k, \boldsymbol{c}_i^{k-1}) = \boldsymbol{v}_k^{\intercal} \tanh(\mathbf{W}_k^{att,q}\boldsymbol{q}_k + \mathbf{W}_k^{att,c}\boldsymbol{c}_i^{k-1} + \boldsymbol{b}_k^{att}), \tag{1}$$

where $\boldsymbol{c}_i^{k-1}$ is the i-th row of $\mathbf{C}_{k-1}$. Again, $\mathbf{W}_k^{att,q} \in \mathbb{R}^{d\times d}$, $\mathbf{W}_k^{att,c} \in \mathbb{R}^{d\times d}$, $\boldsymbol{v}_k \in \mathbb{R}^d$ and $\boldsymbol{b}_k^{att} \in \mathbb{R}^d$ are learnable parameters in the k-th ELC. Finally, hidden state $\boldsymbol{h}_k$ is computed by:

$$\boldsymbol{h}_k = \sum_i \frac{\exp(a(\boldsymbol{q}_k, \boldsymbol{c}_i^{k-1}))}{\sum_j \exp(a(\boldsymbol{q}_k, \boldsymbol{c}_j^{k-1}))}\boldsymbol{c}_i^{k-1}. \tag{2}$$

As for the locator to update item weights, it first obtains the *dense* item weight vector $\boldsymbol{l}'_k = \mathbf{C}_k\boldsymbol{h}_k \in \mathbb{R}^n$ based on the masked $\mathbf{C}_k$ and new hidden state $\boldsymbol{h}_k$. We adopt the sparsemax activation (Martins and Astudillo, 2016) to provide sparsity for $\boldsymbol{l}'_k$. More specifically, sparsemax($\boldsymbol{l}'_k$) returns the euclidean projection of $\boldsymbol{l}'_k$ on the probability simplex of the n-dimensional space. By this definition, the sparsity strength of sparsemax is not controllable. On the other hand, the activation of sparsemax depends ultimately on the absolute difference between the values in $\boldsymbol{l}'_k$. Intuitively, the lower the absolute difference is, the less sparse the activation is. We thus turn to linearly scaling $\boldsymbol{l}'_k$

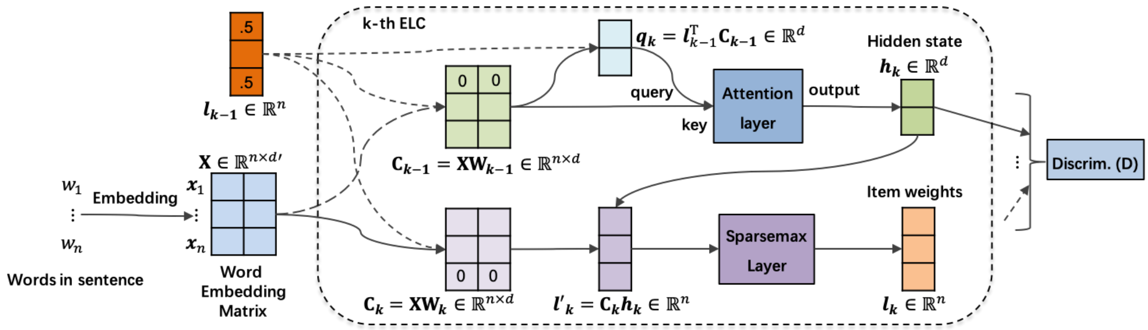


Figure 2: Implementation of a single ELC of TraceNet ($n = 3$, $d' = d = 2$ and we omit the bias vectors for computing $\mathbf{C}_{k-1}$ and $\mathbf{C}_k$).

before computing sparsemax:

$$\boldsymbol{l}_k = \mathsf{sparsemax}(\sigma(-\sum_{j=k}^{L-1} w_j^2 + w_L^2) \cdot \boldsymbol{l}'_k), \quad (3)$$

where L is the number of layers in TraceNet, $\sigma(x) = 1/(1+\exp(-x)) \in (0,1)$ is the sigmoid function, and $w_j \in \mathbb{R}$ $(1 \le j \le L)$ are learnable parameters. As can be easily verified, the linearly scaling weights increase with the increment of k, resulting in the increasing strength of sparsity.

3.4 Smoothness Regularization

After performing the proper transformation, the word embedding matrix $\mathbf{X}$ is fed into encoders and locators repetitively for further learning. To obtain layer-wise smoothness, we adopt the adjacent weight tying approach (Madotto et al., 2018; Sukhbaatar et al., 2015). Recall that each ELC requires two distinct transformed word embedding matrices that are used by the inside encoder and locator, respectively. The main idea of adjacent weight tying is to let every two adjacent ELCs share one transformed word embedding matrix. Formally, the k-th ELC ($k > 1$) only requires a newly-transformed matrix $\mathbf{C}_k = \mathbf{X}\mathbf{W}_k + \boldsymbol{b}_k \in \mathbb{R}^{n\times d}$ (the solid arrow from $\mathbf{X}$ to $\mathbf{C}_k$ in Fig. 2) and re-uses $\mathbf{C}_{k-1} = \mathbf{X}\mathbf{W}_{k-1} + \boldsymbol{b}_{k-1} \in \mathbb{R}^{n\times d}$ from the previous ELC (the dashed arrow from $\mathbf{X}$ to $\mathbf{C}_{k-1}$ in Fig. 2). Here $\mathbf{W}_k \in \mathbb{R}^{d'\times d}$ and $\boldsymbol{b}_k \in \mathbb{R}^d$ are learnable parameters in the k-th ELC. As for the first ELC, two transformed word embedding matrices are still required.

3.5 Proactive Masking

Before the core computation in the k-th ELC, $\mathbf{C}_{k-1}$ and $\mathbf{C}_k$ are further pre-processed by masking with a fixed probability. Take $\mathbf{C}_{k-1}$ as an example. With a pre-defined probability P_{msk}, $\mathbf{C}_{k-1}$ will be masked. We perform independent Bernoulli experiments for each row of $\mathbf{C}_{k-1}$ and the success rate of each experiment is equal to the corresponding item weight in $\boldsymbol{l}_{k-1} \in \mathbb{R}^n$ ($\boldsymbol{l}_{k-1}$ is an input to the k-th ELC). Afterward, all rows that pass the Bernoulli experiments will be replaced with zero. Note that this step is only turned on during training. Figure 2 also illustrates an example of proactive masking. Assume vector $\boldsymbol{l}_{k-1} = [0.5, 0, 0.5]^\intercal$ and $P_{msk} = 1$. Thus, both $\mathbf{C}_{k-1}$ and $\mathbf{C}_k$ are to be masked. For $\mathbf{C}_{k-1}$, it turns out only the first row passes the experiment, resulting in the first row being replaced with zero. Similarly, the last row of $\mathbf{C}_k$ passes the experiment and we show the masked $\mathbf{C}_k$ in Fig. 2.

3.6 Discriminator

We simply adopt a single layer feedforward neural network given the mean of all hidden states to build the discriminator:

$$D([w_1, w_2, \ldots, w_n]) = \mathsf{softmax}((\frac{1}{k}\sum_k \boldsymbol{h}_k)\mathbf{W}^{dis} + \boldsymbol{b}^{dis}). \quad (4)$$

Here, $D([w_1, w_2, \ldots, w_n])$ is the predictive sentiment class of the input. Assuming the number of classes being C, we have learnable parameters $\mathbf{W}^{dis} \in \mathbb{R}^{d\times C}$ and $\boldsymbol{b}^{dis} \in \mathbb{R}^C$.

4 Experiments

4.1 Experimental Setting

Datasets. We chose two datasets (SST-5 and YELP-5) to evaluate our TraceNet.

- SST-5 (Stanford Sentiment Treebank) (Socher et al., 2013) is a sentence-level sentiment classification with five sentiment classes (*i.e.*, very negative, negative, neutral, positive, very positive). We adopted the provided data split, resulting in 8,544, 1,101, and 2,210 sentences in the training, validation,

SST-5	CNN-rand	39.46	LSTM	45.04	BERT	51.99	TraceNet^{-}-X	54.86
	CNN-static	44.32	BiLSTM	45.18	XLNet	55.20	TraceNet-X	55.55
	CNN-nostat	44.62	GT-LSTM	40.70	RoBERTa	56.49	TraceNet^{-}-R	56.59
	CNN-mulch	43.54	TraceNet-G	**46.33**			TraceNet-R	**57.34**
YELP-5	CNN-rand	56.38	LSTM	57.14	BERT	63.42	TraceNet^{-}-X	66.89
	CNN-static	56.30	BiLSTM	55.32	XLNet	66.75	TraceNet-X	67.23
	CNN-nostat	57.24	GT-LSTM	53.38	RoBERTa	67.66	TraceNet^{-}-R	66.92
	CNN-mulch	57.14	TraceNet-G	**58.68**			TraceNet-R	**67.70**

Table 1: Overall accuracy (%) of sentiment classification.

and test sets, respectively. The average length of sentences is 18 words.

- YELP-5 is a document-level review corpus released in the Yelp Dataset Challenge 2015. It has five sentiment classes and the full dataset contains approximately 700,000 documents with an average length of 155 tokens. Due to GPU resource limitation, we only tested on a random 5% sample of the data, resulting in 32,500, 2,500, and 2,500 documents for training, validation, and test, respectively.

Metric. We adopted the classification accuracy (ACC) to evaluate performance, which is the fraction of accurately classified test instances over all test instances.

Baselines. We compared TraceNet with three types of baselines and one simplified variant.

- CNN-rand, CNN-static, CNN-nostat, and CNN-mulch are originally proposed in (Kim, 2014). They only differ in word vectors.

- LSTM, BiLSTM, and GT-LSTM are RNN-based baselines. We followed the implementation in (Cho et al., 2014) for Long Short-Term Memory (LSTM) and bidirectional LSTM (BiLSTM). Gumble Tree LSTM (Choi et al., 2018) (GT-LSTM) is a tree-structured LSTM which further composes task-specific tree structures.

- BERT (Devlin et al., 2019), XLNet (Yang et al., 2019), and RoBERTa (Liu et al., 2019) are the state-of-the-art pre-trained language models. TraceNet-G, TraceNet-X, TraceNet-R represent that the output of GloVe, XLNet and RoBERTa are treated as the input of TraceNet, respectively.

Implementation details. We used the official implementation of all baselines provided by authors. Pre-trained word vectors for CNN and RNN baselines were obtained from GloVe (Pennington et al., 2014). We started with the hyper-parameters recommended in the original papers and finetuned them on the validation set. Since BERT, XLNet, and RoBERTa were sensitive to batch size, learning rate, and maximum length of words on the small SST-5 data, we performed a grid search over $\{16, 32, 64\}$, $\{2e{-}5, 3e{-}5, 5e{-}5\}$, and $\{64, 128, 256\}$ for the three parameters, respectively. Please refer to the supplementary material for the concrete parameters. Code will be publicly available when the paper is accepted.

4.2 Main Results

In the first set of tests, we evaluate the overall performance of all approaches for sentiment classification. All tests were repeated five times. The average results are reported in Table 1, where the letters after TraceNet and TraceNet^{-} indicate the different embedding methods, *i.e.,* GloVe (G), XLNet (X), and RoBERTa (R).

We first compare TraceNet-G with other CNN and LSTM baselines. Except for CNN-rand, these approaches all exploit GloVe for initializing word embeddings and, therefore, can ensure a fair comparison. According to our tests, CNN and LSTM are generally comparable in terms of sentiment classification. By explicitly revealing the input structure, TraceNet-G obtains more promising results, which outperforms all approaches on the sentence-level SST-5 data. On the document-level YELP-5 dataset, we find that LSTMs are better than CNNs and TraceNet-G is the best among its counterparts.

The recent large-scale pre-trained language models significantly increase ACC compared with the aforementioned approaches. We also observe a consistent trend in their performance, such that RoBERTa is the best, followed by XLNet and BERT. Built upon these efforts, TraceNet is able to further enhance the performance. Notably, it refines the results of XLNet on both datasets. Finally, by comparing TraceNet with TraceNet^{-}, we find

Attack	(a) XLNet	(b) TraceNet⁻-X	(c) TraceNet-X	(c)-(a)	(c)-(b)
None	55.20	54.86	**55.55**	0.35	0.69
Replacement (cosine)	52.01	51.83	**52.82***	**0.81**	0.99
Replacement (SWN)	51.11	51.46	**52.34****	**1.23**	0.88
Insertion	47.69	**48.30**	48.13	**0.44**	-0.17
Shuffle	41.69	43.61	**43.95****	**2.25**	0.33
Deletion	41.89	43.19	**43.73****	**1.85**	0.54
Reversing	41.67	42.99	**43.39**	**1.72**	0.40
Replacement (random)	37.94	**39.28**	39.06*	**1.12**	-0.22
Concatenation	36.56	35.93	**38.96**	**2.40**	3.03

*/**: significantly outperform XLNet at the 0.05/0.01 level, t-test

Table 2: Accuracy (%) of sentiment classification under attacks on SST-5.

that the proactive masking strategy consistently has a positive impact. All the above results verify the effectiveness of TraceNet.

4.3 Analysis Under Attacks

In the second set of tests, we evaluate the robustness of TraceNet under attacks. Here we only experiment on SST-5 as the sentiment polarities of sentences are easier to be influenced given its shorter average length. We also only consider XLNet as the embedding method for TraceNet since RoBERTa (named from Robustly optimized BERT approach) has been augmented with a lot of robust designs including training the model longer, with bigger batches over more data, training on longer sequences, etc.[1]

We consider eight types of attacks. More specifically, **Reversing** and **Concatenation** are deterministic attacks such that the former reverses the word orders and the latter concatenates all words in a sentence into one (it will be sliced by XLNet later). The rest are stochastic attacks. The manipulation of **Shuffle** is clear by its name. For **Insertion**, **Deletion**, and **Replacement (random)**, we correspondingly modify one-third of words in a sentence and the new words (if needed) are uniformly sampled following the negative sampling method in word2vec (Mikolov et al., 2013). Finally, for (a) **Replacement (cosine)** and (b) **Replacement (SWN)**, we replace one-third of words in a sentence with (a) their closest terms evaluated by cosine similarity between GloVe vectors and (b) alternative terms within the same sentiment groups in SentiWordNet (Baccianella et al., 2010). We trained models on the original training data and computed ACC on the attacked test data. The results are reported in Table 2 where the numbers for stochastic attacks are the average results of ten independent runs on different attacked test sets.

The results are arranged in the ascending order of the strength of attacks, as evaluated by the ACC of TraceNet. **Replacement (cosine)** and **Replacement (SWN)** are weaker than the other attacks since the semantics or sentiment polarities of terms are not substantially changed. The following is **Insertion** which only introduces noises. Changing word orders (**Shuffle** and **Reversing**) and dropping information (**Deletion**) almost tie in terms of attack strength. Finally, the hardest attacks are **Replacement (random)** and **Concatenation** which both remove original information and introduce noises. Note that the above conclusions should be taken under our attack setting.

Under all attacks, TraceNet is consistently better than XLNet, further verifying the effectiveness of explicitly revealing the input structure. More importantly, the absolute improvement of TraceNet over XLNet is higher than on original data (*i.e.,* 0.35%), which indicates that TraceNet is generally more robust than XLNet under attacks. Since the ACC decreases under attacks, the relative improvement is indeed more prominent. Notably, TraceNet is good at dealing with harder attacks such as changing word orders and dropping information.

Finally, comparing TraceNet with TraceNet⁻, we can conclude that proactive masking boosts model performance in general under attacks. It is especially effective for **Concatenation** which will drop much information after re-slicing by XLNet. However, proactive masking could also lead to negative impacts under **Insertion** and **Replacement (random)** since it is not optimized for dealing with inserted noises.

4.4 Qualitative Analysis of Item Weights

We present a qualitative study on item weights estimated in different ELC layer, shown in Fig. 3. The two displayed movie reviews are retrieved from

[1] As such, we admit that TraceNet does not exhibit obviously better robustness compared with RoBERTa.

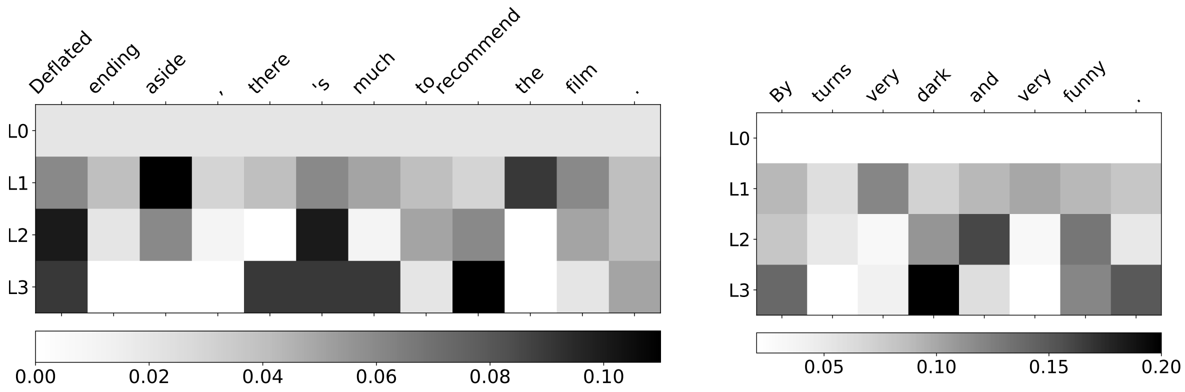
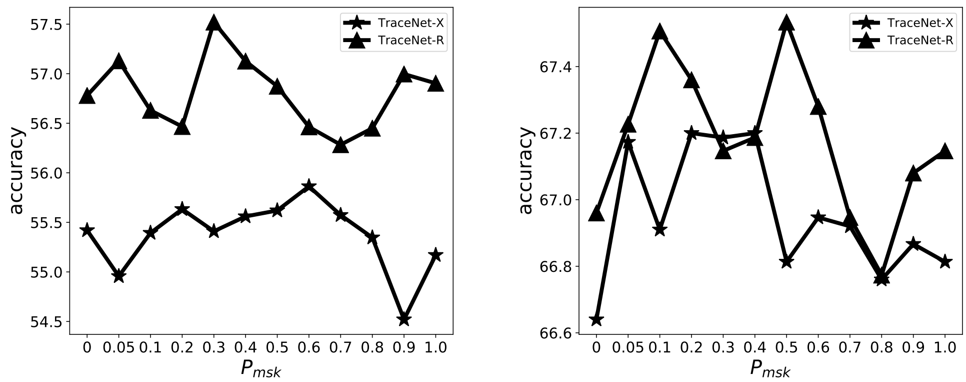


Figure 3: Illustration of item weights identified by TraceNet



(a) SST-5 (b) YELP-5

Figure 4: Impacts of masking probability P_{msk}.

the training set of SST-5, and their ground-truth sentiment labels are *positive* and *very-positive*, respectively. After training, TraceNet could produce accurate labels for both. In the left case, the key elements identified are *deflated*, *there's much*, and *recommend*, which make sense for the prediction result. Also note that it remains difficult to find sentiment words at the beginning. However, the multi-layer architecture enables TraceNet to eventually refine key elements, *e.g., deflated* is identified at the second layer and *recommend* is emphasized finally. Similarly, TraceNet successfully finds the two key words *dark* and *funny* for the right example after learning layer-by-layer. To conclude, these item weights generally make the outcomes of TraceNet easier to understand.

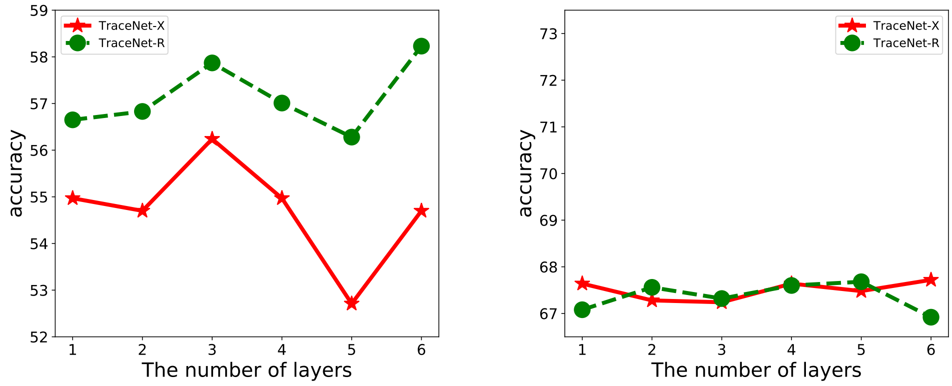
4.5 Analysis on Parameter Sensitivity

4.5.1 Impacts of masking probability P_{msk}

To evaluate the impacts of P_{msk}, we varied P_{msk} from 0 to 1 and computed the classification accuracy of both TraceNet-X and TraceNet-R. We omitted TraceNet-G since its effectiveness is not comparable to TraceNet-X and TraceNet-R. Each P_{msk} was tested 3 times with different seed, and the averaged value is reported in Fig. 4. It turns out that TraceNet is quite sensitive to parameter P_{msk}, possibly due to the randomness in choosing sentences to mask and choosing masked key items. However, compared with turning off proactive masking (*i.e.,* $P_{msk} = 0$), our training strategy remains effective within a certain range of P_{msk}, *e.g.,* $[0.3, 0.5]$ on SST-5 and $[0.05, 0.4]$ on YELP-5.

4.5.2 Impacts of the number L of layers (*i.e.,* ELCs)

To evaluate the impacts of L, we varied L from 1 to 6 and computed the ACC of both TraceNet-X and TraceNet-R on the two datasets. Note that the discriminator combines all the hidden states to derive the final classification results. The results are reported in Fig. 5.



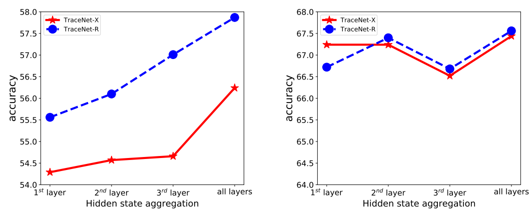
(a) SST-5 (b) YELP-5

Figure 5: Impacts of the number L of layers (*i.e.,* ELCs) on SST-5.

On the YELP-5 data, using more layers is generally more effective, while the impacts of L are quite gentle. On the other hand, the impacts of L are more complex on the SST-5 data. When $L \leq 3$, the ACC of TraceNet increases with the increment of L in general, indicating that TraceNet benefits from its multi-layer organization which enables to learn the input structure for multiple times. Further increase L will lead to the decrease of ACC due to over-fitting. Overall, $L = 3$ is a good choice for TraceNet, and this conclusion holds for the two variants of TraceNet.

4.5.3 Impacts of hidden state aggregation

To evaluate the impacts of hidden state aggregation, we computed the ACC of both TraceNet-X and TraceNet-R using single hidden states and all the three hidden states on the two datasets. The results are reported in Fig. 6.



(a) SST-5 (b) YELP-5

Figure 6: Impacts of hidden state aggregation.

For the case of using single hidden states, the best ACC is obtained by the third and second hidden states on the SST-5 and YELP-5 data, respectively. This is because of their different characteristics of short and long text, *i.e.,* the input structure of short sentences is harder to reveal given the limited information than long documents. Moreover, combining hidden states from all layers is consistently better than using single hidden states alone. We guess that combining hidden states enables the discriminator to directly supervise each layer in terms of revealing the input structure, which enhances the effectiveness.

5 Conclusion

In this paper, we proposed TraceNet to tackle sentiment analysis task such that the outcome is mainly contributed by a few key elements of the input. The idea behind TraceNet, which originates from the two-streams hypothesis, is to learn discriminative representations and reveal input structure simultaneously. To do this, TraceNet stacks several encoders and locators layer-by-layer, with increasing-strength sparsity constraints on locators for tracing key elements. Smoothness regularization is enforced on adjacent encoder-locator layer to ensure the stabilization of learning across layers. In addition, a proactive masking strategy is further incorporated into TraceNet for robustness. We applied TraceNet for sentence- and document-level sentiment analysis. The experiments demonstrated the effectiveness of TraceNet. Moreover, considering a total of eight types of attacks, we verified the better robustness of TraceNet in general. Finally, our qualitative analysis of item weights showed the advantage of TraceNet in terms interpretability.

References

Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. 2010. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *Proceedings of the International Conference on Language Resources and Evaluation, LREC*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR*.

Zhuang Chen and Tieyun Qian. 2019. Transfer capsule network for aspect level sentiment classification. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL*, pages 547–556.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1724–1734.

Jihun Choi, Kang Min Yoo, and Sang-goo Lee. 2018. Learning to compose task-specific tree structures. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18)*, pages 5094–5101.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pages 4171–4186.

Melvyn A Goodale, A David Milner, et al. 1992. Separate visual pathways for perception and action. *Trends Neurosci.*, 15(1):20–5.

Robert A Jacobs, Michael I Jordan, and Andrew G Barto. 1991. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive science*, 15(2):219–250.

Rie Johnson and Tong Zhang. 2015. Effective use of word order for text categorization with convolutional neural networks. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 103–112.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1746–1751.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Andrea Madotto, Chien-Sheng Wu, and Pascale Fung. 2018. Mem2seq: Effectively incorporating knowledge bases into end-to-end task-oriented dialog systems. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL*, pages 1468–1478.

André F. T. Martins and Ramón Fernández Astudillo. 2016. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, volume 48, pages 1614–1623.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1532–1543.

Karen Simonyan and Andrew Zisserman. 2014. Two-stream convolutional networks for action recognition in videos. *arXiv preprint arXiv:1406.2199*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1631–1642.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

Jialong Tang, Ziyao Lu, Jinsong Su, Yubin Ge, Linfeng Song, Le Sun, and Jiebo Luo. 2019. Progressive self-supervised attention learning for aspect-level sentiment analysis. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL*, pages 557–566.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *6th International Conference on Learning Representations, ICLR*.

Jiangliu Wang and Yunhui Liu. 2018. Kinematics features for 3d action recognition using two-stream cnn. In *2018 13th World Congress on Intelligent Control and Automation (WCICA)*, pages 1731–1736.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32*, pages 5754–5764.

Lei Zhang, Shuai Wang, and Bing Liu. 2018. Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1253.

Ning Zhang, Jingen Liu, Ke Wang, Dan Zeng, and Tao Mei. 2021. Robust visual object tracking with two-stream residual convolutional networks. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4123–4130. IEEE.

Xiao Zhang and Dan Goldwasser. 2019. Sentiment tagging with partial labels using modular architectures. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL*, pages 579–590.

A Example Appendix

Appendix A: Experimental Details

We first present more experimental details for reproduce purpose.

Public SST-5[2] and YELP-5[3] datasets are choosed to evaluate our TraceNet architecture. We adopted a third-party implementation[4] for CNN-rand, CNN-static, CNN-nostat, CNN-mulch, LSTM, and BiLSTM. The source code of GT-LSTM had been released[5] by its authors. We implemented BERT, XLNet, RoBERTa and TraceNet based on Hugging Face library[6]. All hyper-parameters of these approaches are summarized in Table 1. Finally, when initializing word embedding with pretrained vectors, glove.840B.300d[7] is adopted. Words not in the pretrained vectors vocabulary are initialized randomly. We have attached the code and data in the supplementary material.

All our tests were performed on Tesla V100 GPUs with 32GB memory. Model selection was performed according to the performance on the validation set such that the CNN- and LSTM-based baselines were trained for a maximum of 20 epochs and the rest approaches for a maximum of 10 epochs.

[2] https://nlp.stanford.edu/sentiment/
[3] http://goo.gl/JyCnZq
[4] https://github.com/andyweizhao/capsule_text_classification
[5] https://github.com/jihunchoi/unsupervised-treelstm
[6] https://github.com/huggingface/transformers
[7] https://nlp.stanford.edu/projects/glove/

Algorithm	SST-5	YELP-5
CNN-rand CNN-static CNN-nostat CNN-mulch	kernel size: {2,3,4,5} filter number (per kernel size): 300 L_2 weight: 0.01 batch size: 50 learning rate: 0.001 sequence length: 49	kernel size: {2,3,4,5} filter number (per kernel size): 300 L_2 weight: 0.01 batch size: 50 learning rate: 0.001 sequence length: 256
LSTM BiLSTM	hidden state size: 100 L_2 weight: 0.01 batch size: 50 learning rate: 0.001 sequence length: 49 dropout: 0.5	hidden state size: 100 and 50, respectively L_2 weight: 0.01 batch size: 50 learning rate: 0.001 sequence length: 256 dropout: 0.5
GT-LSTM	hidden state size: 300 batch size: 64 learning rate: 1.0, halved every two epochs sequence length: 49 dropout: 0.5	hidden state size: 300 batch size: 16 learning rate: 1.0, halved every two epochs sequence length: 256 dropout: 0.5.
BERT XLNet RoBERTa	hidden state size: 768 model type: base-cased, base and base, resp. weight decay: 0.1, 0.1, and 0.0, resp. Adam epsilon: 1e-8, 1e-8, and 1e-6, resp. batch size: 32, 16, and 16, resp. learning rate: 5e-5, 2e-5, and 2e-5, resp. sequence length: 128, 64, and 128, resp. dropout 0.1	hidden state size: 768 model type: base-cased, base and base, resp. weight decay: 0.1, 0.1 and 0.0, resp. Adam epsilon: 1e-8, 1e-8, and 1e-6, resp. batch size: 64 learning rate: 5e-5, 2e-5, and 2e-5, resp. sequence length: 256 dropout: 0.1
TraceNet-G TraceNet-X TraceNet-R	hidden state size: 50, 128, and 512, resp. weight decay: 0.2, 0.1, and 0.0, respectively Adam epsilon: 1e-8, 1e-8, and 1e-6, resp. batch size: 64, 16, and 16, respectively learning rate: 1e-3, 2e-5, and 2e-5, resp. sequence length: 49, 64, and 128, respectively dropout: 0.2, 0.3, and 0.1, respectively P_{msk}: 0.05, 0.2 and 0.3, respectively number L of layers: 3	hidden state size: 500, 512, and 768, resp. weight decay: 0.2, 0.1, and 0.1, respectively Adam epsilon: 1e-8 batch size: 64 learning rate: 1e-3, 2e-5, and 2e-5, resp. sequence length: 256 dropout: 0.2, 0.1, and 0.1, respectively. P_{msk}: 0.05, 0.05 and 1.0, respectively number L of layers: 3

Table 3: Hyper-parameters setting. (49 refers to the maximum sentence length of SST-5.)