PhraseRNN: Phrase Recursive Neural Network for Aspect-based Sentiment Analysis

Thien Hai Nguyen Kiyoaki Shirai
School of Information Science
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan
{nhthien, kshirai}@jaist.ac.jp

Abstract

This paper presents a new method to identify sentiment of an aspect of an entity. It is an extension of RNN (Recursive Neural Network) that takes both dependency and constituent trees of a sentence into account. Results of an experiment show that our method significantly outperforms previous methods.

1 Introduction

Aspect-based sentiment analysis (ABSA) has been found to play a significant role in many applications such as opinion mining on product or restaurant reviews. It is a task to determine an attitude, opinion and emotions of people toward aspects in a sentence. For example, given a sentence “Except the design, the phone is bad for me”, the system should classify positive and negative as the sentiments for the aspects ‘design’ and ‘phone’, respectively.

The simple approach is to calculate a sentiment score of a given aspect as the weighted sum of opinion scores, which are defined by a sentiment lexicon, of all words in the sentence (Liu and Zhang, 2012; Pang and Lee, 2008). This method is further improved by identifying the aspect-opinion relations using tree kernel method (Nguyen and Shirai, 2015a).

Other researches have attempted to use unsupervised topic modeling methods. To identify the sentiment category of the aspect, topic models which can simultaneously exploit aspect and sentiment have been proposed, such as TSLDA (Nguyen and Shirai, 2015b), ASUM (Jo and Oh, 2011), JST (Lin and He, 2009) and FACTS model (Lakkaraju et al., 2011).

Recursive Neural Network (RNN) is a kind of deep neural network. Using distributed representations of words (aka word embedding) (Bengio et al., 2003; Hinton, 1986), RNN merges word representations to represent phrases or sentences. It is one of the best methods to predict sentiment labels for the phrases (Socher et al., 2011; Socher et al., 2012; Socher et al., 2013). AdaRNN (Adaptive Recursive Neural Network) is an extension of RNN for Twitter sentiment classification (Dong et al., 2014a; Dong et al., 2014b).

This paper proposes a new method PhraseRNN for ABSA. It is an extended model of RNN and AdaRNN, which are briefly introduced in Section 2. The basic idea is to make the representation of the target aspect richer by using syntactic information from both the dependency and constituent trees of the sentence.

2 Recursive Neural Networks for ABSA

In RNN and AdaRNN, given a sentence containing a target aspect, “binary dependency tree” is built from a dependency tree of the sentence. Intuitively, it represents syntactic relations associated with the aspect. Each word (leaf) or phrase (internal node) in the binary dependency tree is represented as a d-dimensional vector. From bottom to top, the representations of a parent node \( v \) is calculated by combining of left and right child vector representations \( (v_l, v_r) \) using a global function \( g \) in RNN:

\[
g(v_l, v_r) = W \begin{bmatrix} v_l \\ v_r \end{bmatrix} + b
\]

(1)

where \( W \in \mathbb{R}^{d \times 2d} \) is the composition matrix and \( b \in \mathbb{R}^d \) is the bias vector. Then \( v = f(g(v_l, v_r)) \) where \( f \) is a nonlinear function such as \( \text{tanh} \).

Instead of using only a global function \( g \), AdaRNN employed \( n \) compositional functions \( G = \{g_1, \ldots, g_n\} \) and selected them depending on the linguistic tags and combined vectors as follows:

\[
v = f\left( \sum_{i=1}^n P(g_i|v_l, v_r, e)g_i(v_l, v_r) \right)
\]

(2)
where \( P(g_i|v_l,v_r,e) \) is the probability of function \( g_i \) given the child vectors \( v_l, v_r \) and external feature vector \( e \). The probabilities are estimated as Equation (3).

\[
    \begin{align*}
    P(g_1|v_l,v_r,e) & \cdots \\ 
    P(g_n|v_l,v_r,e) & \end{align*} = \text{softmax} \left( \beta R \begin{bmatrix} v_l \\ v_r \\ e \end{bmatrix} \right)
\]

where \( \beta \in \mathbb{R} \) is a hyper-parameter, and \( R \in \mathbb{R}^{n \times (2d+1)} \) is the parameter matrix.

The vector of the root node of the binary dependency tree is regarded as a representation of the target aspect. It is fed to a logistic regression to predict the sentiment category of the aspect.

### 3 PhraseRNN: Phrase Recursive Neural Network

In this model, a representation of an aspect will be obtained from a “target dependent binary phrase dependency tree” constructed by combining the constituent and dependency trees. In addition, instead of using a list of global functions \( G \) as in AdaRNN, two kinds of composition functions \( G \) in inner-phrase and \( H \) in outer-phrase are used.

#### 3.1 Building Hierarchical Structure

First, the basic phrases (noun phrases, verb phrases, preposition phrases and so on) are extracted from the constituent tree of the sentence. For example, a list of phrases \( P = \{\text{PP[Except the design]}, \text{NP[the phone]}, \text{VP[is bad for me]}\} \) is extracted from the constituent tree in Figure 1.

Given a dependency tree and a list of phrases, a phrase dependency tree is created by Algorithm 1. The input is a dependency tree \( T = (V,E) \) consisting of a set of vertices \( V = \{v_1, \cdots, v|V|\} \) and a set of relation edges \( E = \{(v_j,v_i,v_j)\} \) between two vertices, and a list of phrases \( P = \{p_1, \cdots, p_K\} \) extracted from the constituent tree. The output is a phrase dependency tree \( pT = (pV,pE) \) where \( pV = \{T_1, \cdots, T_K\} \) \( (T_i = (V_i,E_i) \) is a subtree) and \( pE = \{(r_{ji},T_i,T_j)\} \) (a set of relations between two subtrees). With the dependency tree and the phrase list in Figure 2(a), the algorithm will output a phrase dependency tree in Figure 2(b).

#### Algorithm 1: Convert to Phrase Dependency Tree

**Input:** dependency tree \( T = (V,E) \), phrase list \( P = \{p_1, \cdots, p_K\} \)

**Output:** phrase dependency tree:

\[
    pT = (pV,pE)
\]

1. For each phrase \( p_i \in P \) do
   2. \( V_i \leftarrow \{v_j|v_j|p_i\} \)
3. End

4. For each edge \( (r_{nm},v_m,v_n) \in E \) do
   5. \( v_m \in p_k, v_n \in p_l \)
   6. If \( k = l \) then
      7. \( E_k \leftarrow E_k \cup \{(r_{nm},v_m,v_n)\} \)
   8. Else
      9. \( pE \leftarrow pE \cup \{(r_{nm},T_k,T_l)\} \)
10. End
11. End

The phrase dependency tree is transformed into a target dependent binary phrase dependency tree \( bpT \) by Algorithm 2. The input of the algorithm is a phrase dependency tree \( pT = (pV,pE) \) and a target word \( v_1 \) (the aspect word we want to predict the sentiment category). The output is the binary tree \( bpT \). Note that the leaves of the binary tree \( bpT \) are binary subtrees \( bT_1, \cdots, bT_K \) which are the binary versions of subtrees \( T_1, \cdots, T_K \). On the other hand, the leaves of binary subtree \( bT_i \) are the words in phrase \( p_i \). \( bpT \) and \( bT_i \) are obtained by convert function defined as Algorithm 3. It can convert an arbitrary tree to a binary tree.  

#### 3.2 Constructing the Aspect Representation

Each node in the binary tree is represented as a d-dimensional vector. In this research, we use the pre-trained Google News dataset 2 by word2vec algorithms (Mikolov et al., 2013). Each word is

\[1\] Note that convert function returns a tree represented by nested brackets such as [PP[NP[VP]]].

\[2\] https://code.google.com/p/word2vec/
Algorithm 2: Convert to Target Dependent Binary Phrase Dependency Tree

Input: phrase dependency tree: \( pT = (pV, pE) \), target \( v_t \)

Output: target dependent binary phrase dependency tree: \( bpT \)

1. for \( T_i = (V_i, E_i) \in pV \) do
   2. if \( v_t \in V_i \) then
      3. \( h \leftarrow v_t \)
   4. else
      5. \( h \leftarrow \) vertex having no head in \( E_i \)
   6. end
   7. \( bT_i \leftarrow \text{convert}(E_i, h) \)
8. end
9. \( T_{v_t} \leftarrow T_i \) that contains \( v_t \)
10. \( bpT \leftarrow \text{convert}(pE, T_{v_t}) \)
11. Replace all \( T_i \) in \( bpT \) with \( bT_i \)

Algorithm 3: Convert to a Binary Tree

Function convert\( (E, v_t) \):

1. \( v \leftarrow v_t \)
2. for \( v_i \rightarrow v_t, v_t \rightarrow v_i \) in \( E \) do
   3. if \( v_t \rightarrow v_i \) then
      4. \( E' \leftarrow E \setminus \{v_t \rightarrow v_i\} \)
      5. \( w \leftarrow \text{convert}(E, v_i, v) \)
   6. else
      7. \( E' \leftarrow E \setminus \{v_i \rightarrow v_t\} \)
      8. \( w \leftarrow [v, \text{convert}(E, v_t)] \)
   9. end
10. \( v \leftarrow w \)
11. return \( v \)

Figure 2: Hierarchical Structures in PhraseRNN: (a) Dependency Tree, (b) Phrase Dependency Tree and (c) Target Dependent Binary Phrase Dependency Tree

Figure 3: Another Target Dependent Binary Phrase Dependency Tree (Target Aspect ‘phone’)

PhraseRNN uses two kinds of composition function \( G = \{g_1, \cdots, g_n\} \) for inner-phrase and \( H = \{h_1, \cdots, h_m\} \) for outer-phrase. \( n \) and \( m \) are the number of functions in \( G \) and \( H \), respectively.

The vector of the parent node \( v_{in} \) in the binary subtree \( bT_i \), where \( v_l \) and \( v_r \) are the vectors of the left and right children, is computed as:

\[
v_{in} = f \left( \sum_{i=1}^{n} P(g_i|v_l, v_r, e_{in})g_i(v_l, v_r) \right) (4)
\]

where \( e_{in} \) is the external feature vector. \( P(g_i|v_l, v_r, e_{in}) \) is the probability of function \( g_i \) given the child vectors \( v_l, v_r \) and \( e_{in} \). It is...
defined as Equation (5).

\[
P(g_1|v_l,v_r,e_{in})
\cdots
P(g_n|v_l,v_r,e_{in})
\begin{bmatrix}
\end{bmatrix}
= \cdots \text{two lists of global functions. One for inner-phrase, the other for outer-phrase: } G = \{g_1, \cdots , g_n\} \text{ and } H = \{h_1, \cdots , h_m\}
\]  

where $\beta \in \mathbb{R}$ is a hyper-parameter, and $R \in \mathbb{R}^{n \times (2d+|e_{in}|)}$ is the parameter matrix.

In the target dependent binary phrase dependency tree $bpT$, the vector of the parent node $v_{out}$, where the vectors of the left and right children are $v_l$ and $v_r$, is computed as:

\[
v_{out} = f \left( \sum_{i=1}^{m} P(h_i|v_l,v_r,e_{out})h_i(v_l,v_r) \right)
\]  

\[
P(h_i|v_l,v_r,e_{out}) = \text{softmax} \left( \beta S \begin{bmatrix} v_l \\ v_r \end{bmatrix} \right)
\]  

where $S \in \mathbb{R}^{m \times (2d+|e_{out}|)}$ is the parameter matrix.

The external features $e_i$ ($e_{in}$ and $e_{out}$) of the node $v_l$ consists of three types of features: $Label_l$, $Label_r$, and $DepType_i$. $Label_l$ and $Label_r$ are the labels of the left and right child nodes, respectively. If node $v_l$ is a leaf word, $Label_l$ is the POS of the word $v_l$. Otherwise, it is the non-terminal symbol of the lowest common parent of descendants of $v_l$ in the constituent tree. For example, the $Label$ of the node combined from ‘the’ and ‘design’ in Figure 2(c) is ‘NP’ which is the lowest common parent of these two words in the constituent tree in Figure 1. $DepType_i$ is the dependency relation for node $v_l$. If the left and right children of $v_l$ are leaf nodes, it is the direct relation in the dependency tree between them. Otherwise, $DepType_i$ is the relation between head words of the left and right nodes. For instance, in Figure 2(c), let $a$ be the parent of ‘is’ and ‘bad’, $b$ is the parent of ‘for’ and ‘me’, $c$ is the parent of $a$ and $b$. $DepType$ of $a$ and $b$ are ‘COP’ and ‘POBJ’ that are direct relations between two child nodes in the dependency tree in Figure 2(a). While, $DepType$ of $c$ is ‘PREP’ that is the dependency relation between two head words ‘bad’ and ‘for’. $e_i$ is a binary vector where the weight of the vector represents the presence of each feature.

We suppose a batch training data consisting of $B$ instances \{$(x^{(1)}, t^{(1)}), \cdots , (x^{(B)}, t^{(B)})$\}, where $x^{(b)}$ and $t^{(b)}$ are the aspect and its sentiment category of $b$-th instance. Let $y^{(b)}$ be the predicted sentiment category for aspect $x^{(b)}$ by PhraseRNN. The goal is to minimize the loss function which is the sum of the mean of negative log likelihood and L2 regularization penalty in a batch training set as in Equation (8).

\[
L = -\frac{1}{B} \sum_{b=1}^{B} \log (P(y^{(b)} = t^{(b)}|x^{(b)}, \theta)) + \lambda \sum_{\theta \in \theta} \parallel \theta \parallel^2
\]  

where $\lambda$ is a constant controlling the degree of penalty, $\theta$ is all the parameters in the model.

Stochastic gradient descent is used to optimize the loss function. Backpropagation is employed to propagate the errors from the top node to the leaf nodes. The derivatives of parameters are used to update the parameters.

4 Evaluation

We use the restaurant reviews dataset in SemEval2014 Task 4 consisting of over 3000 English sentences. For each aspect, “positive”, “negative” or “neutral” is annotated as its polarity. Dataset is divided into three parts: 70% training, 10% development and 20% test.

We compare the following methods:

ASA w/o RE: It defines a sentiment score of a given aspect as the weighted sum of opinion scores of all words in the sentence, where the weight is defined by the distance from the aspect (Liu and Zhang, 2012; Pang and Lee, 2008).

ASA with RE: It improves “ASA w/o RE” by firstly identifying the aspect-opinion relations using tree kernel, then integrating them to the sentiment calculation (Nguyen and Shirai, 2015a).

RNN: It uses only one global function $g_1$ over the binary dependency tree.

AdaRNN: It uses multi-composition functions $G = \{g_1, \cdots , g_n\}$ over a binary dependency tree (Dong et al., 2014a).

PhraseRNN-1: our PhraseRNN with only one global function: $G = H = g_1$

PhraseRNN-2: our PhraseRNN with two global functions. One for inner-phrase, the other for outer-phrase: $G = g_1$ and $H = h_1$

PhraseRNN-3: our PhraseRNN with multiple global functions: $G = H = \{g_1, \cdots , g_n\}$

PhraseRNN-4: our PhraseRNN with two lists of global functions. One for inner-phrase, the other for outer-phrase: $G = \{g_1, \cdots , g_n\}$ and $H = \{h_1, \cdots , h_m\}$
Stanford CoreNLP (Manning et al., 2014) is used to parse the sentence and obtain constituent and dependency trees. For RNN, AdaRNN and PhraseRNN, the optimal parameters, which minimize the error in the development set, are used for the sentiment classification of the test set. We set $\beta = 1$ for AdaRNN and PhraseRNN since it is reported that $\beta = 1$ is the best parameter (Dong et al., 2014a). The optimized number of composition functions $n$ and $m = \frac{n}{2}$ are selected by grid search with $n = \{2, 4, 6, 8, 10\}$ on the development set. $\lambda = 0.0001$ is employed. Accuracy ($A$), Precision ($P$), Recall ($R$) and F-measure ($F$) are used as evaluation metrics.

Table 1 shows the results of the methods. Differences of PhraseRNN and RNN are verified by statistical significance tests. We use the paired randomization test because it does not require additional assumption about distribution of outputs (Smucker et al., 2007). The results indicate that four variations of our PhraseRNN outperform “ASA w/o RE”, “ASA with RE”, RNN and AdaRNN methods from 5.35% to 19.44% accuracy and 8% to 16.48% F-measure. Among four variations, PhraseRNN-2 and PhraseRNN-3 achieved the best performance. By using different global functions in the inner and outer phrases, PhraseRNN-2 improves PhraseRNN-1 by 2.54% F-measure while keeping the comparable accuracy. Using multi-composition functions is also effective since PhraseRNN-3 was better than PhraseRNN-1 by 1.55% accuracy. PhraseRNN-4 improved PhraseRNN-3 by 6.38% precision while keeping comparable in other metrics.

Since our PhraseRNN-1 and PhraseRNN-3 outperform RNN and AdaRNN (the models relying on the binary dependency tree) respectively, we can conclude that our target dependent binary phrase dependency tree is much effective than binary dependency tree for ABSA.

In the data used in (Dong et al., 2014a), one sentence contains only one aspect. On the other hand, two or more aspects can be appeared in one sentence in SemEval 2014 data. It is common in the real text. To examine in which cases our method is better than the others, we conduct an additional experiment by dividing the test set into three disjoint subsets. The first subset (S1) contains sentences having only one aspect. The second subset (S2) and third subset (S3) have two or more aspects in each sentence. All aspects in a sentence in S2 have the same sentiment category, while different sentiment categories in S3. The number of aspects in S1, S2 and S3 are 200, 323 and 187, respectively.

Table 2 shows the number of aspects where their sentiments are correctly identified by the methods in the subsets S1, S2 and S3. The accuracies in S1, S2 and S3 are 200, 323 and 187, respectively.

5 Conclusion

We proposed PhraseRNN to identify the sentiment of the aspect in the sentence. Propagating the semantics through the binary dependency tree in RNN and AdaRNN could not be enough to represent the sentiment of the aspect. A new hierarchical structure was constructed by integrating the dependency relations and phrases. The results indicated that our PhraseRNN outperformed “ASA w/o RE”, “ASA with RE”, RNN and AdaRNN.
References


