A Neural Transition-based Model for Argumentation Mining

Jianzhu Bao[1,2*], **Chuang Fan**[1,2*], **Jipeng Wu**[1,2], **Yixue Dang**[3]
Jiachen Du[1,2], **Ruifeng Xu**[1,4†]
[1]Harbin Institute of Technology (Shenzhen), China
[2]Joint Lab of China Merchants Securities and HITSZ
[3]China Merchants Securities Co., Ltd.
[4]Peng Cheng Laboratory, Shenzhen, China
{jianzhubao, fanchuanghit}@gmail.com
wujipeng@stu.hit.edu.cn, dangyixue@cmschina.com.cn
jacobvan199165@gmail.com, xuruifeng@hit.edu.cn

Abstract

The goal of argumentation mining is to automatically extract argumentation structures from argumentative texts. Most existing methods determine argumentative relations by exhaustively enumerating all possible pairs of argument components, which suffer from low efficiency and class imbalance. Moreover, due to the complex nature of argumentation, there is, so far, no universal method that can address both tree and non-tree structured argumentation. Towards these issues, we propose a neural transition-based model for argumentation mining, which incrementally builds an argumentation graph by generating a sequence of actions, avoiding inefficient enumeration operations. Furthermore, our model can handle both tree and non-tree structured argumentation without introducing any structural constraints. Experimental results show that our model achieves the best performance on two public datasets of different structures.

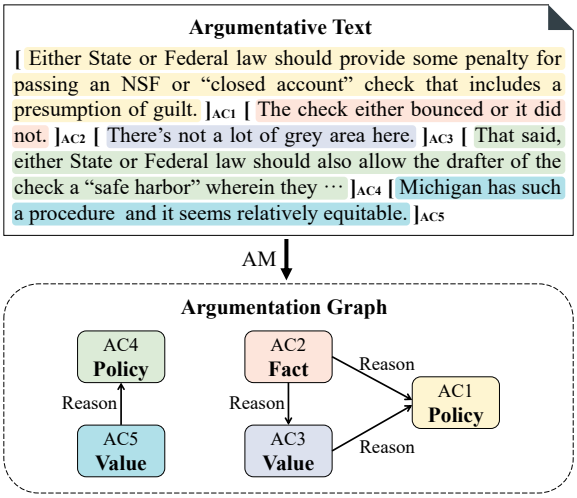


Figure 1: An example of argumentation mining from the CDCP dataset (Park and Cardie, 2018). *Policy*, *Fact*, and *Value* represent the types of ACs and *Reason* refers to the types of ARs. Note that, the CDCP dataset we use is preprocessed by Niculae et al. (2017).

1 Introduction

Argumentation mining (AM) aims to identify the argumentation structures in text, which has received widespread attention in recent years (Lawrence and Reed, 2019). It has been shown beneficial in a broad range of fields, such as information retrieval (Carstens and Toni, 2015; Stab et al., 2018), automated essay scoring (Wachsmuth et al., 2016; Ke et al., 2018), and legal decision support (Palau and Moens, 2009; Walker et al., 2018). Given a piece of paragraph-level argumentative text, an AM system first detects argument components (ACs), which are segments of text with argumentative meaning, and then extracts the argumentative relations (ARs) between ACs to obtain an argumentation graph, where the nodes and edges represent ACs and ARs, respectively. An example of AM is shown in Figure 1, where the text is segmented into five ACs, and there are four ARs. In this instance, the types of AC2 and AC3 are *Fact* (non-experiential objective proposition) and *Value* (proposition containing value judgments), respectively. In addition, there is an AR from AC2 to AC3, i.e., "*The check either bounced or it did not.*" is the reason of "*There's not a lot of grey area here.*", for the latter is a value judgment based on the fact of the former.

Generally, AM involves several subtasks, including 1) Argument component segmentation (ACS), which separates argumentative text from non-argumentative text; 2) Argument component type classification (ACTC), which determines the types of ACs (e.g., *Policy*, *Fact*, *Value*, etc.); 3) Argumentative relation identification (ARI), which identifies ARs between ACs; 4) Argumentative relation type classification (ARTC), which determines

*Equal Contribution
†Corresponding Author

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pages 6354–6364
August 1–6, 2021.

the types of ARs (e.g., *Reason* and *Evidence*). Most previous works assume that subtask 1) ACS has been completed, that is, ACs have been segmented, and focus on other subtasks (Potash et al., 2017; Kuribayashi et al., 2019; Chakrabarty et al., 2019). In this paper, we also make such an assumption, and perform ACTC and ARI on this basis.

Among all the subtasks of AM, ARI is the most challenging because it requires understanding complex semantic interactions between ACs. Most previous works exhaustively enumerate all possible pairs of ACs (i.e., all ACs are matched to each other by Cartesian products) to determine the ARs between them (Kuribayashi et al., 2019; Morio et al., 2020). However, these approaches are of low efficiency and can cause class imbalance, since the majority of AC pairs have no relation. Besides, due to different annotation schemes, there are mainly two kinds of structures of argumentation graphs, tree (Stab and Gurevych, 2014; Peldszus, 2014) and non-tree (Park and Cardie, 2018). Briefly, in tree structures, each AC has at most one outgoing AR, but there is no such restriction in non-tree structures (Figure 1). However, studies on these two kinds of structures are usually conducted separately. To date, there is no universal method that can address both tree and non-tree structured argumentation without any corpus-specific constraints.

Towards these issues, we present a neural transition-based model for AM, which can classify the types of ACs and identify ARs simultaneously. Our model predicts a sequence of actions to incrementally construct a directed argumentation graph, often with $O(n)$ parsing complexity. This allows our model to avoid inefficient enumeration operations and reduce the number of potential AC pairs that need evaluating, thus alleviating the class imbalance problem and achieving speedup. Also, our transition-based model does not introduce any corpus-specific structural constraints, and thus can handle both tree and non-tree structured argumentation, yielding promising generalization ability. Furthermore, we enhance our transition-based model with pre-trained BERT (Devlin et al., 2019), and use LSTM (Hochreiter and Schmidhuber, 1997) to represent the parser state of our model.

Extensive experiments on two public datasets with different structures show that our transition-based model outperforms previous methods, and achieves state-of-the-art results. Further analysis reveals that our model is of low parsing complexity and has a strong structure adaptive ability. To the best of our knowledge, we are the first to investigate transition-based methods for AM.

2 Related Work

In computational AM, there are mainly two types of approaches to model argumentation structures, that is, tree and non-tree.

2.1 Tree Structured AM

Most previous works assume that the argumentation graphs can be viewed as tree or forest structures, which makes the problem computationally easier because many tree-based structural constraints can be applied.

Under the theory of Van Eemeren et al. (2004), Palau and Moens (2009) modeled argumentation in the legal text as tree structures and used handcrafted context-free grammar to identify these structures. Presented by Stab and Gurevych (2014, 2017), the tree structured Persuasive Essay (PE) dataset has been utilized in a number of studies in AM. Following this dataset, Persing and Ng (2016) and Stab and Gurevych (2017) leveraged the Integer Linear Programming (ILP) framework to jointly predict ARs and AC types, in which several structural constraints are defined to ensure the tree structures. The arg-microtext (MT) dataset, created by Peldszus (2014), is another tree structured dataset. Studies on this dataset usually apply decoding mechanisms based on tree structures, such as Minimum Spanning Trees (MST) (Peldszus and Stede, 2015) and ILP (Afantenos et al., 2018).

Regarding neural network-based methods, Eger et al. (2017) studied AM as a dependency parsing and a sequence labeling problem with multiple neural networks. Potash et al. (2017) introduced the sequence-to-sequence based Pointer Networks (Vinyals et al., 2015) to AM, and used the output of encoder and decoder to identify AC types and the presence of ARs, respectively. Kuribayashi et al. (2019) proposed an argumentation structure parsing model based on span representation, which used ELMo (Peters et al., 2018) to obtain representations for ACs.

2.2 Non-tree Structured AM

Those studies described in Section 2.1 are all based upon the assumption that the argumentation forms tree structures. However, this assumption is somewhat idealistic since argumentation structures in

real-life scenarios may not be such well-formed. Hence, some studies have focused on non-tree structured AM, and these studies typically use the Consumer Debt Collection Practices (CDCP) (Park and Cardie, 2018) dataset. Regarding this dataset, Niculae et al. (2017) presented a structured learning approach based on factor graphs, which can also handle the tree structured PE dataset. However, the factor graph needs to be specifically designed according to the types of argumentation structures. Galassi et al. (2018) adopted residual networks for AM on the CDCP dataset. Recently, Morio et al. (2020) proposed a model devoted to non-tree structured AM, with a task-specific parameterization module to encode ACs and a biaffine attention module to capture ARs.

To the best of our knowledge, until now there is no universal method that can address both tree and non-tree structured argumentation without any corpus-specific design. Thus, in this work, we fill this gap by proposing a neural transition-based model that can identify both tree and non-tree argumentation structures without introducing any prior structural assumptions.

2.3 Transition-based Methods

Transition-based methods are commonly used in dependency parsing (Chen and Manning, 2014; Gómez-Rodríguez et al., 2018), and has also been successfully applied to other NLP tasks with promising performance, such as discourse parsing (Yu et al., 2018), information extraction (Zhang et al., 2019), word segmentation (Zhang et al., 2016) and mention recognition (Wang et al., 2018).

3 Task Definition

Following previous works (Potash et al., 2017; Kuribayashi et al., 2019), we assume subtask 1) ACS has been completed, i.e., the spans of ACs are given. Then, we aim at jointly classifying AC types (ACTC) and determining the presence of ARs (ARI). The reason why we do not jointly conduct AR type classification (ARTC) is that performing ARTC along with ACTC and ARI jointly will hurt the overall performance. More details on this issue will be discussed in Section 6.4.

Formally, we assume a piece of argumentation related paragraph $\mathcal{P} = (w_1, w_2, \ldots, w_m)$ consisting of m tokens and a set $X = (x_1, x_2, \ldots, x_n)$ consisting of n AC spans are given. Each AC span x_i is a tuple containing the beginning token index b_i and the ending token index e_i of this AC, i.e., $x_i = (b_i, e_i)$. The goal is to classify the types of ACs and identify the ARs, and finally obtain a directed argumentation graph with ACs and ARs representing nodes and edges, respectively.

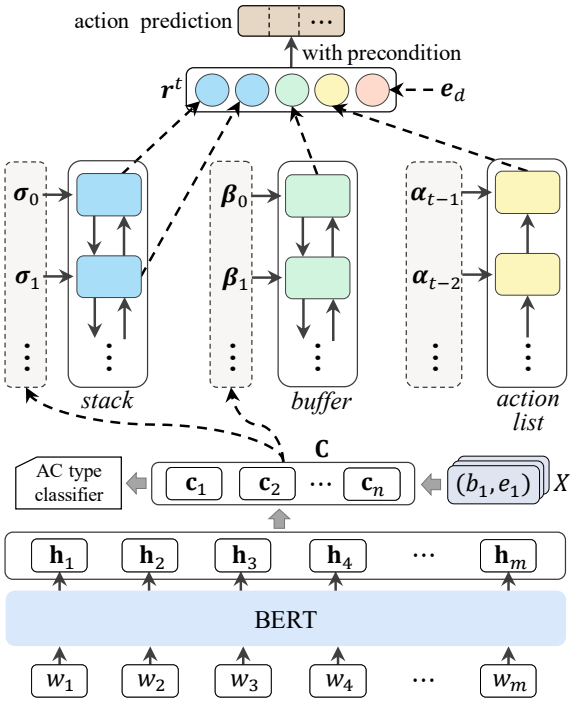


Figure 2: The architecture of our model. ARs are identified by the action prediction. The types of ACs are determined by the AC type classifier.

4 Our Approach

We present a neural transition-based model for AM, which can jointly learn ACTC and ARI. Our model generates a sequence of actions in terms of the parser state to incrementally build an argumentation graph. We utilize BERT and LSTM to represent our parser state, which contains a *stack* $\boldsymbol{\sigma}$ to store processed ACs, a *buffer* $\boldsymbol{\beta}$ to store unprocessed ACs, a *delay set* D to record ACs that need to be removed subsequently, and an *action list* $\boldsymbol{\alpha}$ to record historical actions. Then, the learning problem is framed as: given the parser state of current step t: $(\boldsymbol{\sigma}^t, \boldsymbol{\beta}^t, D^t, \boldsymbol{\alpha}^t)$, predict an action to determine the parser state of the next step, and simultaneously identify ARs according to the predicted action. Figure 2 shows the architecture of our model. In the following, we first introduce our transition system, then describe the parser state representation.

Action	Change of state	Precondition
SH	$(\boldsymbol{\sigma}_0\|\boldsymbol{\sigma}_1\|\boldsymbol{\sigma}, \boldsymbol{\beta}_0\|\boldsymbol{\beta}, D, R) \Rightarrow (\boldsymbol{\beta}_0\|\boldsymbol{\sigma}_0\|\boldsymbol{\sigma}_1\|\boldsymbol{\sigma}, \boldsymbol{\beta}, D, R)$	$\boldsymbol{\sigma}_1 \notin D \wedge \boldsymbol{\beta} \neq [\,]$
DE_d	$(\boldsymbol{\sigma}_0\|\boldsymbol{\sigma}_1\|\boldsymbol{\sigma}, \boldsymbol{\beta}_0\|\boldsymbol{\beta}, D, R) \Rightarrow (\boldsymbol{\sigma}_0\|\boldsymbol{\sigma}, \boldsymbol{\beta}_0\|\boldsymbol{\beta}, D - \{\boldsymbol{\sigma}_1\}, R)$	$\boldsymbol{\sigma}_1 \in D \wedge \boldsymbol{\beta} \neq [\,]$
DE	$(\boldsymbol{\sigma}_0\|\boldsymbol{\sigma}_1\|\boldsymbol{\sigma}, \|\boldsymbol{\beta}, D, R) \Rightarrow (\boldsymbol{\sigma}_0\|\boldsymbol{\sigma}, \|\boldsymbol{\beta}, D, R)$	$\boldsymbol{\beta} = [\,]$
RA	$(\boldsymbol{\sigma}_0\|\boldsymbol{\sigma}_1\|\boldsymbol{\sigma}, \|\boldsymbol{\beta}, D, R) \Rightarrow (\boldsymbol{\sigma}_1\|\boldsymbol{\sigma}, \|\boldsymbol{\beta}, D, R \cup \{\boldsymbol{\sigma}_0 \rightarrow \boldsymbol{\sigma}_1\})$	$\boldsymbol{\beta} = [\,]$
RA_d	$(\boldsymbol{\sigma}_0\|\boldsymbol{\sigma}_1\|\boldsymbol{\sigma}, \boldsymbol{\beta}_0\|\boldsymbol{\beta}, D, R) \Rightarrow (\boldsymbol{\beta}_0\|\boldsymbol{\sigma}_0\|\boldsymbol{\sigma}_1\|\boldsymbol{\sigma}, \boldsymbol{\beta}, D \cup \{\boldsymbol{\sigma}_0\}, R \cup \{\boldsymbol{\sigma}_0 \rightarrow \boldsymbol{\sigma}_1\})$	$\boldsymbol{\beta} \neq [\,]$
LA	$(\boldsymbol{\sigma}_0\|\boldsymbol{\sigma}_1\|\boldsymbol{\sigma}, \boldsymbol{\beta}_0\|\boldsymbol{\beta}, D, R) \Rightarrow (\boldsymbol{\sigma}_0\|\boldsymbol{\sigma}, \boldsymbol{\beta}_0\|\boldsymbol{\beta}, D, R \cup \{\boldsymbol{\sigma}_0 \leftarrow \boldsymbol{\sigma}_1\})$	

Table 1: Actions designed in our transition system. R denotes the set of ARs extracted so far. For simplicity, we omit the superscript t and use the subscript $i \in \{0, 1, ...\}$ to denote the element index in *stack* and *buffer*. For example, $\boldsymbol{\sigma}_0|\boldsymbol{\sigma}_1|\boldsymbol{\sigma}$ denotes the top two items in *stack*. An action can be selected only if its precondition is satisfied.

Stack $\boldsymbol{\sigma}$	Buffer $\boldsymbol{\beta}$	Delay-Set D	Action $\boldsymbol{\alpha}$	AR R
[ ]	[1,2,3,4,5]	$\varnothing$	SH	-
[1]	[2,3,4,5]	$\varnothing$	SH	-
[2,1]	[3,4,5]	$\varnothing$	RA_d	$2 \rightarrow 1$
[3,2,1]	[4,5]	$\{2\}$	LA	$3 \leftarrow 2$
[3,1]	[4,5]	$\{2\}$	RA_d	$3 \rightarrow 1$
[4,3,1]	[5]	$\{2,3\}$	DE_d	-
[4,1]	[5]	$\{2\}$	SH	-
[5,4,1]	[ ]	$\{2\}$	RA	$5 \rightarrow 4$
[4,1]	[ ]	$\{2\}$	DE	-
[4]	[ ]	$\{2\}$	-	-

Table 2: Transition sequence for the text in Figure 1. For simplicity, we use indices to denote ACs.

4.1 Transition System

Our transition system contains six types of actions. Different actions will change the state in different ways, which are also summarized in Table 1:

- SHIFT (SH): When $\boldsymbol{\beta}^t$ is not empty and $\boldsymbol{\sigma}_1$ is not in D^t, pop $\boldsymbol{\beta}_0$ from $\boldsymbol{\beta}^t$ and move it to the top of $\boldsymbol{\sigma}^t$.
- DELETE-DELAY (DE_d). When $\boldsymbol{\beta}^t$ is not empty and $\boldsymbol{\sigma}_1$ is in D^t, remove $\boldsymbol{\sigma}_1$ from $\boldsymbol{\sigma}^t$ and D^t, and keep $\boldsymbol{\beta}^t$ unchanged.
- DELETE (DE). When $\boldsymbol{\beta}^t$ is empty, remove $\boldsymbol{\sigma}_1$ from $\boldsymbol{\sigma}^t$ and keep $\boldsymbol{\beta}^t$ and D^t unchanged.
- RIGHT-ARC (RA). When $\boldsymbol{\beta}^t$ is empty, remove $\boldsymbol{\sigma}_0$ from $\boldsymbol{\sigma}^t$ and assign an AR from $\boldsymbol{\sigma}_0$ to $\boldsymbol{\sigma}_1$.
- RIGHT-ARC-DELAY (RA_d). When $\boldsymbol{\beta}^t$ is not empty, pop $\boldsymbol{\beta}_0$ from $\boldsymbol{\beta}^t$ and move it to the top of $\boldsymbol{\sigma}^t$. Then assign an AR from $\boldsymbol{\sigma}_0$ to $\boldsymbol{\sigma}_1$ and add $\boldsymbol{\sigma}_0$ into D^t for delayed deletion. This strategy can help extract more ARs related to $\boldsymbol{\sigma}_0$.
- LEFT-ARC (LA). Remove $\boldsymbol{\sigma}_1$ from $\boldsymbol{\sigma}^t$ and assign an AR from $\boldsymbol{\sigma}_1$ to $\boldsymbol{\sigma}_0$.

Table 2 illustrates the golden transition sequence of the text in Figure 1. This example text contains five ACs and four ARs. At the initial state, all ACs are in *buffer*. Then, a series of actions change the parser state according to Table 1, and extract ARs simultaneously. This procedure stops when meeting the terminal state, that is, *buffer* is empty and *stack* only contains one element.

4.2 State Representation

We employ BERT to obtain the representation of each AC and use LSTM to encode the long-term dependencies of *stack*, *buffer* and *action list*.

Representation of ACs. We feed the input paragraph $\mathcal{P} = (w_1, w_2, \ldots, w_m)$ into BERT to get the contextual representation matrix $\mathbf{H} \in \mathbb{R}^{m \times d_b}$, where d_b is the vector dimension of the last layer of BERT. In this way, paragraph $\mathcal{P}$ can be represented as $\mathbf{H} = (\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_m)$, where $\mathbf{h}_i$ is the contextual representation of the i-th token of $\mathcal{P}$.

Then, we use the AC spans set $X = (x_1, x_2, \ldots, x_n)$ to produce a contextual representation of each AC from $\mathbf{H}$ by mean pooling over the representations of words in each AC span. Specifically, for the i-th AC with span $x_i = (b_i, e_i)$, the contextual representation of this AC could be obtained by:

$$\mathbf{u}_i = \frac{1}{e_i - b_i + 1} \sum_{j=b_i}^{e_i} \mathbf{h}_j \qquad (1)$$

where $\mathbf{u}_i \in \mathbb{R}^{d_b}$. In addition, following previous works (Potash et al., 2017; Kuribayashi et al., 2019), we also combine some extra features with $\mathbf{u}_i$ to represent ACs, including the bag-of-words (BoW) vector, position and paragraph type embedding of each AC[1]. We denote these features of the i-th AC as $\boldsymbol{\phi}_i$. Then, the i-th AC is represented by

[1]Details of these features are described in Appendix A.

the concatenation of $\mathbf{u}_i$ and $\boldsymbol{\phi}_i$:

$$\mathbf{c}_i = [\mathbf{u}_i; \boldsymbol{\phi}_i] \tag{2}$$

Hence, the ACs in paragraph $\mathcal{P}$ can be represented as $\mathbf{C} = (\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n)$.

Representation of Parser State. Our transition-based model utilizes the parser state to predict a sequence of actions. At each step t, we denote our parser state as $(\boldsymbol{\sigma}^t, \boldsymbol{\beta}^t, D^t, \boldsymbol{\alpha}^t)$. $\boldsymbol{\sigma}^t$ and $\boldsymbol{\beta}^t$ are *stack* and *buffer*, which store the representations of processed and unprocessed ACs, respectively. D^t is the *delay set* that records ACs that need to be removed from *stack* subsequently. $\boldsymbol{\alpha}^t$ is the *action list* that stores the actions generated so far. At the beginning, all ACs are in the *buffer*, i.e., the initial parser state is $([\,], [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n], \varnothing, [\,])$. Then, a series of predicted actions will iteratively change the parser state.

Specifically, at step t, we have $\boldsymbol{\sigma}^t = (\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_1, \ldots)$, $\boldsymbol{\beta}^t = (\boldsymbol{\beta}_0, \boldsymbol{\beta}_1, \ldots)$, where $\boldsymbol{\sigma}_i$ and $\boldsymbol{\beta}_i$ indicate the representations of ACs in the *stack* and the *buffer* at the current state. In addition, we also have $\boldsymbol{\alpha}^t = (\ldots, \boldsymbol{\alpha}_{t-2}, \boldsymbol{\alpha}_{t-1})$ where $\boldsymbol{\alpha}_i$ denotes the distributed representation of the i-th action obtained by a looking-up table $\mathbf{E}_a$. In order to capture the context information in the *stack* $\boldsymbol{\sigma}^t$, we feed it into a bidirectional LSTM:

$$\begin{aligned} \mathbf{S}^t &= [\mathbf{s}_0, \mathbf{s}_1, \ldots] \\ &= \mathrm{BiLSTM_s}([\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_1, \ldots]) \end{aligned} \tag{3}$$

where $\mathbf{S}^t \in \mathbb{R}^{|\boldsymbol{\sigma}^t| \times 2d_l}$ is the output of LSTM from both directions, $|\boldsymbol{\sigma}^t|$ is the size of *stack*, and d_l is the hidden size of LSTM. Similarly, we can obtain the contextual representation of $\boldsymbol{\beta}^t$ by:

$$\begin{aligned} \mathbf{B}^t &= [\mathbf{b}_0, \mathbf{b}_1, \ldots] \\ &= \mathrm{BiLSTM_b}([\boldsymbol{\beta}_0, \boldsymbol{\beta}_1, \ldots]) \end{aligned} \tag{4}$$

where $\mathbf{B}^t \in \mathbb{R}^{|\boldsymbol{\beta}^t| \times 2d_l}$, $|\boldsymbol{\beta}^t|$ is the size of *buffer*. Besides, in order to incorporate the historical action information into our model, we apply a unidirectional LSTM to process the *action list*:

$$\begin{aligned} \mathbf{A}^t &= [\ldots, \mathbf{a}_{t-2}, \mathbf{a}_{t-1}] \\ &= \mathrm{LSTM_a}([\ldots, \boldsymbol{\alpha}_{t-2}, \boldsymbol{\alpha}_{t-1}]) \end{aligned} \tag{5}$$

where $\mathbf{A}^t \in \mathbb{R}^{|\boldsymbol{\alpha}^t| \times d_l}$, $|\boldsymbol{\alpha}^t|$ is the size of *action list*.

Furthermore, since the relative distance between the pair $(\boldsymbol{\sigma}_0, \boldsymbol{\sigma}_1)$ is a strong feature for determining their relations, we represent it as an embedding $\mathbf{e}_d$ through another looking-up table $\mathbf{E}_d$. Thus, the parser state representation $\mathbf{r}^t$ can be obtained by:

$$\mathbf{r}^t = [\mathbf{s}_0; \mathbf{s}_1; \mathbf{b}_0; \mathbf{a}_{t-1}; \mathbf{e}_d] \tag{6}$$

where $\mathbf{s}_0$ and $\mathbf{s}_1$ denote the first and second elements of $\mathbf{S}^t$, $\mathbf{b}_0$ is the first element of the $\mathbf{B}^t$, and $\mathbf{a}_{t-1}$ indicates the last action representation of $\mathbf{A}^t$.

4.3 Action Prediction

To predict the current action at step t, we first apply a multi-layer perceptron (MLP) with ReLU activation to squeeze the state representation $\mathbf{r}^t$ to a lower-dimensional vector $\mathbf{z}^t$, and then compute the action probability by a softmax output layer:

$$\mathbf{z}^t = \mathrm{MLP_a}(\mathbf{r}^t) \tag{7}$$

$$p(\alpha_t|\mathbf{z}^t) = \frac{exp(\mathbf{W}_\alpha^\top \mathbf{z}^t + \mathbf{b}_\alpha)}{\sum_{\alpha' \in \mathcal{A}(S)} exp(\mathbf{W}_{\alpha'}^\top \mathbf{z}^t + \mathbf{b}_{\alpha'})} \tag{8}$$

where $\mathbf{W}_\alpha$ denotes a learnable parameter matrix, $\mathbf{b}_\alpha$ is the bias term, α_t is the predicted action for step t. $\mathcal{A}(S)$ represents the set of valid candidate actions that may be taken according to the preconditions. For efficient decoding, we greedily take the candidate action with the highest probability. With the predicted action sequence, we could identify ARs according to Table 1. Note that, the univocal supervision over actions for one input paragraph is built based on the gold labels of ARs.

4.4 Training

We jointly train an AC type classifier over the AC representations: $p(y_i|\mathbf{C}) = softmax(\mathrm{MLP_c}(\mathbf{c}_i))$, where y_i is the predicted type for the i-th AC. Finally, combining this task with action prediction, the training objective of our model can be obtained:

$$\begin{aligned} \mathcal{J}(\boldsymbol{\theta}) = &\sum_t \log p(\alpha_t|\mathbf{z}^t) + \sum_i \log p(y_i|\mathbf{C}) \\ &+ \frac{\lambda}{2}||\boldsymbol{\theta}||^2 \end{aligned} \tag{9}$$

where λ is the coefficient of L_2-norm regularization, and $\boldsymbol{\theta}$ denotes all the parameters in this model.

5 Experimental Setup

5.1 Dataset

We conduct experiments on two datasets: Persuasive Essays (PE) (Stab and Gurevych, 2017) and Consumer Debt Collection Practices (CDCP) (Niculae et al., 2017; Park and Cardie, 2018).

The PE dataset contains 402 essays (1,833 paragraphs), in which 80 essays (369 paragraphs) are held out for testing. There are three types of ACs in this dataset: *Major-Claim*, *Claim*, and *Premise*. Also, each AC in PE dataset has at most one outgoing AR. That is, the argumentation graph of one paragraph can be either directed trees or forests. We extend each AC by including its argumentative marker in the same manner as Kuribayashi et al. (2019).

The CDCP dataset consists of 731 paragraphs, and 150 of them are reserved for testing. It provides five types of ACs (propositions): *Reference*, *Fact*, *Testimony*, *Value*, and *Policy*. Unlike PE dataset, each AC in CDCP dataset can have two or more outgoing ARs, thus forming non-tree structures.

5.2 Implementation Details

For PE dataset, we randomly choose 10% of the training set as the validation set, which is consistent with the work of Kuribayashi et al. (2019). For CDCP dataset, we randomly choose 15% of the training set for validation. Following Potash et al. (2017), for ACTC, we employ F_1 score for each AC type and their macro averaged score to measure the performance. Similarly, for ARI, we present F_1 scores for the presence/absence of links between ACs and their macro averaged score. All experiments are performed 5 times with different random seeds, and the scores are averaged.

We finetune uncased BERT$_{Base}$ [2] in our model. AdamW optimizer (Loshchilov and Hutter, 2019) is adopted for parameter optimization, and the initial learning rates for the BERT layer and other layers are set to 1e-5 and 1e-3, respectively. All LSTMs are 1 layer with the hidden size of 256, and the hidden size of MLP is 512. Besides, the dropout rate (Srivastava et al., 2014) is set to 0.5, and the batch size is set to 32. All parameters of our model are unfixed and can be learned during training. We train the model 50 epochs with early stopping strategy, and choose model parameters with the best performance (average of macro F_1 scores of ACTC and ARI) on the validation set. Our model is implemented in PyTorch (Paszke et al., 2019) on a NVIDIA Tesla V100 GPU.

5.3 Baselines

In order to evaluate our proposed **BERT-Trans** model, we compare it with several baselines.

[2] https://github.com/huggingface/transformers

For PE dataset, the following baselines are compared:

Joint-ILP (Stab and Gurevych, 2017) jointly optimizes AC types and ARs by Integer Linear Programming (ILP).

St-SVM-full is structured SVM with full factor graph, which performs best on PE dataset in the work of Niculae et al. (2017).

Joint-PN (Potash et al., 2017) applies Pointer Network with attention mechanism to AM, which can jointly address both ACTC and ARI.

Span-LSTM (Kuribayashi et al., 2019) employs LSTM-minus-based span representation with pretrained ELMo embedding for AM, which is the current state-of-the-art method on PE dataset.

For CDCP dataset, we compare our model with the following baselines:

Deep-Res-LG (Galassi et al., 2018) applies residual network model with link-guided training procedure, to perform ACTC and ARI.

St-SVM-strict is structured SVM with strict factor graph, which performs best on CDCP dataset in the work of (Niculae et al., 2017).

TSP-PLBA (Morio et al., 2020) uses task-specific parameterization to encode ACs and biaffine attention to capture ARs with ELMo based features, which is the current state-of-the-art method on CDCP dataset.

Furthermore, in order to show the effectiveness of our proposed transition system, we implemented two additional baselines:

Span-LSTM-Trans incorporates the span representation method used in Span-LSTM and our transition system on PE dataset. For a fair comparison, features and ELMo used to represent ACs are consistent with that of Span-LSTM.

ELMo-Trans replaces BERT in our proposed model with ELMo on CDCP dataset for a fair comparison with TSP-PLBA.

6 Results and Analysis

6.1 Main Results

The overall performance of our proposed model and the baselines are shown in Table 3 and Table 4. Our model achieves the best performance on both datasets. On PE dataset, our model outperforms the current sota model Span-LSTM by at least 1.1% and 1.4% in macro F_1 score over ACTC and ARI, respectively. On CDCP dataset, compared with TSP-PLBA, our model obtains at least 3.6% higher

Method	ACTC				ARI		
	Macro	MC	Claim	Premise	Macro	Rel	No-Rel
Joint-ILP	82.6	89.1	68.2	90.3	75.1	58.5	91.8
St-SVM-full	77.6	78.2	64.5	90.2	-	60.1	-
Joint-PN	84.9	89.4	73.2	92.1	76.7	60.8	92.5
Span-LSTM	87.3	-	-	-	81.1	-	-
Span-LSTM-Trans	87.5	**93.8**	76.4	92.2	82.0	69.8	94.1
BERT-Trans (Ours)	**88.4**	93.2	**78.8**	**93.1**	**82.5**	**70.6**	**94.3**

Table 3: Comparison results with baselines on PE dataset (%). The best scores are in bold.

Method	ACTC						ARI		
	Macro	Value	Policy	Testimony	Fact	Ref	Macro	Rel	No-Rel
Deep-Res-LG	65.3	72.2	74.4	72.9	40.3	66.7	-	29.3	-
St-SVM-strict	73.2	76.4	76.8	71.5	41.3	100.0	-	26.7	-
TSP-PLBA	78.9	-	-	-	-	-	-	34.0	-
ELMo-Trans	78.9	80.0	82.3	80.6	51.5	100.0	67.1	35.6	**98.6**
BERT-Trans (Ours)	**82.5**	**83.2**	**86.3**	**84.9**	**58.3**	100.0	**67.8**	**37.3**	98.3

Table 4: Comparison results with baselines on CDCP dataset (%). The best scores are in bold.

Method	ACTC		ARI	
	Macro	∇	Macro	∇
BERT-Trans (Ours)	**88.4**	-	**82.5**	-
w/o LSTM	87.9	-0.5	80.5	-2.0
w/o buffer	88.1	-0.3	80.7	-1.8
w/o action	87.8	-0.6	80.9	-1.6
w/o distance	88.1	-0.3	81.8	-0.7
w/o BoW	85.9	-2.5	80.6	-1.9

Table 5: The results of ablation experiments on PE dataset (%). The best scores are in bold.

macro F_1 score over ACTC, and achieves about 3.3% higher relation F_1 over ARI.

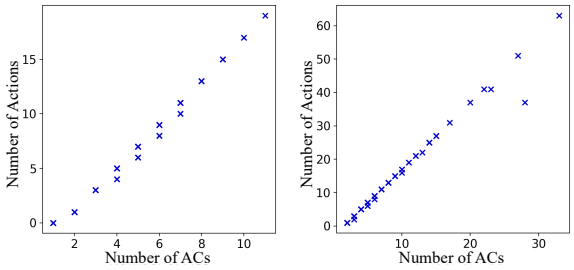
We also show the results where our BERT-based AC representation is replaced by the ELMo-based method, that is, Span-LSTM-Trans on PE dataset and ELMo-Trans on CDCP dataset. We found that, without employing pre-trained BERT, Span-LSTM-Trans and ELMo-Trans still outperform Span-LSTM and TSP-PLBA over ARI, respectively, which demonstrates the effectiveness of our proposed transition system. It can also be observed that our BERT-based AC representation method can further improve the model performance.

Some of the baselines improve overall performance by imposing structural constraints when predicting or decoding. For example, Joint-PN only predicts one outgoing AR for each AC to partially enforce the predicted argumentation graphs as tree structures. Similarly, to ensure tree structures, Span-LSTM applies MST algorithm based on the probabilities calculated by the model. However, these two methods can only deal with tree structured argumentation. The method proposed by Niculae et al. (2017), which is based on factor graph, can handle both tree and no-tree structured argumentative text (St-SVM-full and St-SVM-strict), but the factor graph need to be specifically designed for datasets of different structures. Differently, our proposed model can handle datasets of both tree and non-tree structures without introducing any corpus-specific structural constraints and also outperforms all the structured baselines.

6.2 Ablation Study

We conduct ablation experiments on the PE dataset to further investigate the impacts of each component in BERT-Trans. The results are shown in Table 5. It can be observed that applying LSTM to encode *buffer*, *stack*, and *action list* contributes about 2.0% macro F_1 score of ARI, showing the necessity of capturing non-local dependencies in parser state. Also, incorporating *buffer* into parser state can improve the macro F_1 score of ARI by about 1.8%, for *buffer* can provide crucial information about subsequent ACs to be processed. Besides, the macro F_1 score of ARI drops heavily without *action list* (-1.6%), indicating that the historical action information has a significant impact on predicting the next action. Without the distance information between the top two ACs of the *stack*, the macro F_1 score of ARI decreases by 0.7%. The model components described above mainly affect ARI by modifying the parsing procedure, but have little impact on ACTC. However, BoW feature has a significant influence on both two tasks, and removing it causes 2.5% and 1.9% decreases in macro F_1 score of ACTC and ARI, respectively.

6.3 Parsing Complexity



(a) PE dataset (b) CDCP dataset

Figure 3: The number of actions relative to the number of ACs of each paragraph.

Most previous models parse argumentation graphs by exhaustively enumerating all possible pairs of ACs, that is, all ACs are connected by Cartesian products, which lead to $O(n^2)$ parsing complexity. Differently, our transition-based model can incrementally parse an argumentation graph by predicting a sequence of actions, often with linear parsing complexity. Concretely, given a paragraph with n ACs, our system can parse it with $O(n)$ actions.

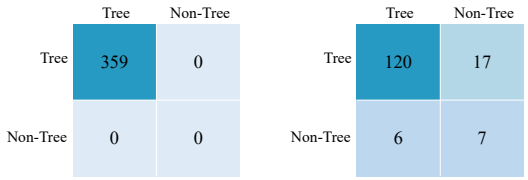
Parsing complexity of our transition system can be determined by the number of actions performed with respect to the number of ACs in a paragraph. Specifically, we measure the length of the action sequence predicted by our model for every paragraph from the test sets of PE dataset and CDCP dataset and depict the relation between them and the number of ACs. As shown in Figure 3, the number of predicted actions is linearly related to the number of ACs in both two datasets, proving that our system can construct an argumentation graph with $O(n)$ complexity. In addition, we also compared our model with the current state-of-the-art model on PE dataset, i.e., Span-LSTM, in terms of training time, and our model is around two times faster.

6.4 Joint Learning Analysis

Following Kuribayashi et al. (2019), we also try to add the task of AR type classification (ARTC) to our model for joint learning on PE dataset. However, as shown in Table 6, jointly learning ARTC together with ACTC and ARI degrades the overall performance, while learning ARTC separately actually yields better performance. Such an observation is consistent with the joint learning results

Method	Joint Tasks	Macro F_1		
		ACTC	ARI	ARTC
BERT-Trans (Ours)	ALL	86.8	81.8	78.4
	ACTC+ARI	**88.4**	**82.5**	-
	ARTC	-	-	**81.0**
Span-LSTM	ALL	85.7	80.7	79.0
	ACTC+ARI	87.3	81.1	-
	ARTC	-	-	79.6

Table 6: Joint learning results on PE dataset (%). ALL indicates joint learning of all three subtasks.



(a) PE dataset. (b) CDCP dataset.

Figure 4: Confusion matrices of structure type on test set. Vertical direction indicates predicted structure type, horizontal direction indicates gold structure type.

of Span-LSTM in Kuribayashi et al. (2019). The reason may be that the class labels are usually very unbalanced for ARTC (around 1:10 in PE dataset and 1:25 in CDCP dataset), such that the high uncertainty can seriously affect the overall learning. Thus, we mainly focus on joint learning of ACTC and ARI. We also argue that learning ARTC individually is better than jointly learning it with other subtasks. Besides, our model outperforms Span-LSTM over ACTC and ARI even when joint learning all three subtasks.

6.5 Structure Adaptive

To validate the structure adaptive ability of our model on both tree and non-tree structures, we analyze the structure type of the predicted argumentation graphs on the test set of both PE and CDCP datasets in Figure 4. It can be seen that for non-tree structured CDCP dataset, even though there are few non-tree structured paragraphs in the test set of CDCP (only 16%), our model is still able to identify 29.2% of them. This is an acceptable performance considering the poor results of ARI on the CDCP dataset due to the complex non-tree structures. For tree structured PE dataset, our model predicts all the paragraphs as tree structures, showing a strong structure adaptive ability. In contrast, most previous models like Joint-PN and Span-LSTM can only predict tree structures.

7 Conclusion

In this paper, we propose a neural transition-based model for argumentation mining, which can incrementally construct an argumentation graph by predicting a sequence of actions. Our proposed model can handle both tree and non-tree structures, and often with linear parsing complexity. The experimental results on two public datasets demonstrate the effectiveness of our model. One potential drawback of our model is the greedy decoding for action prediction. For future work, we plan to optimize the decoding process by using methods like beam search to further boost the performance.

Acknowledgments

This work was partially supported by National Natural Science Foundation of China (61632011, 61876053, 62006062), Guangdong Province Covid-19 Pandemic Control Research Funding (2020KZDZX1224), Shenzhen Foundational Research Funding (JCYJ20180507183527919, JCYJ20180507183608379), and the Joint Lab of China Merchants Securities and HITSZ.

References

Stergos D. Afantenos, Andreas Peldszus, and Manfred Stede. 2018. Comparing decoding mechanisms for parsing argumentative structures. *Argument Comput.*, 9(3):177–192.

Lucas Carstens and Francesca Toni. 2015. Towards relation based argumentation mining. In *Proceedings of the 2nd Workshop on Argumentation Mining, ArgMining@HLT-NAACL 2015, June 4, 2015, Denver, Colorado, USA*, pages 29–34. The Association for Computational Linguistics.

Tuhin Chakrabarty, Christopher Hidey, Smaranda Muresan, Kathy McKeown, and Alyssa Hwang. 2019. AMPERSAND: argument mining for persuasive online discussions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2933–2943. Association for Computational Linguistics.

Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 740–750. ACL.

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 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Steffen Eger, Johannes Daxenberger, and Iryna Gurevych. 2017. Neural end-to-end learning for computational argumentation mining. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 11–22. Association for Computational Linguistics.

Andrea Galassi, Marco Lippi, and Paolo Torroni. 2018. Argumentative link prediction using residual networks and multi-objective learning. In *Proceedings of the 5th Workshop on Argument Mining, ArgMining@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pages 1–10. Association for Computational Linguistics.

Carlos Gómez-Rodríguez, Tianze Shi, and Lillian Lee. 2018. Global transition-based non-projective dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 2664–2675. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Zixuan Ke, Winston Carlile, Nishant Gurrapadi, and Vincent Ng. 2018. Learning to give feedback: Modeling attributes affecting argument persuasiveness in student essays. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 4130–4136. ijcai.org.

Tatsuki Kuribayashi, Hiroki Ouchi, Naoya Inoue, Paul Reisert, Toshinori Miyoshi, Jun Suzuki, and Kentaro Inui. 2019. An empirical study of span representations in argumentation structure parsing. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4691–4698. Association for Computational Linguistics.

John Lawrence and Chris Reed. 2019. Argument mining: A survey. *Comput. Linguistics*, 45(4):765–818.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Gaku Morio, Hiroaki Ozaki, Terufumi Morishita, Yuta Koreeda, and Kohsuke Yanai. 2020. Towards better non-tree argument mining: Proposition-level biaffine parsing with task-specific parameterization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 3259–3266. Association for Computational Linguistics.

Vlad Niculae, Joonsuk Park, and Claire Cardie. 2017. Argument mining with structured svms and rnns. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 985–995. Association for Computational Linguistics.

Raquel Mochales Palau and Marie-Francine Moens. 2009. Argumentation mining: the detection, classification and structure of arguments in text. In *The 12th International Conference on Artificial Intelligence and Law, Proceedings of the Conference, June 8-12, 2009, Barcelona, Spain*, pages 98–107. ACM.

Joonsuk Park and Claire Cardie. 2018. A corpus of erulemaking user comments for measuring evaluability of arguments. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*. European Language Resources Association (ELRA).

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

Andreas Peldszus. 2014. Towards segment-based recognition of argumentation structure in short texts. In *Proceedings of the First Workshop on Argument Mining, hosted by the 52nd Annual Meeting of the Association for Computational Linguistics, ArgMining@ACL 2014, June 26, 2014, Baltimore, Maryland, USA*, pages 88–97. The Association for Computer Linguistics.

Andreas Peldszus and Manfred Stede. 2015. Joint prediction in mst-style discourse parsing for argumentation mining. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 938–948. The Association for Computational Linguistics.

Isaac Persing and Vincent Ng. 2016. End-to-end argumentation mining in student essays. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 1384–1394. The Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics.

Peter Potash, Alexey Romanov, and Anna Rumshisky. 2017. Here's my point: Joint pointer architecture for argument mining. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1364–1373. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

Christian Stab and Iryna Gurevych. 2014. Annotating argument components and relations in persuasive essays. In *COLING 2014, 25th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, August 23-29, 2014, Dublin, Ireland*, pages 1501–1510. ACL.

Christian Stab and Iryna Gurevych. 2017. Parsing argumentation structures in persuasive essays. *Comput. Linguistics*, 43(3):619–659.

Christian Stab, Tristan Miller, Benjamin Schiller, Pranav Rai, and Iryna Gurevych. 2018. Cross-topic argument mining from heterogeneous sources. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 3664–3674. Association for Computational Linguistics.

Frans Van Eemeren, Rob Grootendorst, and Frans H van Eemeren. 2004. *A systematic theory of argumentation: The pragma-dialectical approach*. Cambridge University Press.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2692–2700.

Henning Wachsmuth, Khalid Al Khatib, and Benno Stein. 2016. Using argument mining to assess the argumentation quality of essays. In *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan*, pages 1680–1691. ACL.

Vern R. Walker, Dina Foerster, Julia Monica Ponce, and Matthew Rosen. 2018. Evidence types, credibility factors, and patterns or soft rules for weighing conflicting evidence: Argument mining in the context of legal rules governing evidence assessment. In *Proceedings of the 5th Workshop on Argument Mining, ArgMining@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pages 68–78. Association for Computational Linguistics.

Bailin Wang, Wei Lu, Yu Wang, and Hongxia Jin. 2018. A neural transition-based model for nested mention recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1011–1017. Association for Computational Linguistics.

Nan Yu, Meishan Zhang, and Guohong Fu. 2018. Transition-based neural RST parsing with implicit syntax features. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 559–570. Association for Computational Linguistics.

Junchi Zhang, Yanxia Qin, Yue Zhang, Mengchi Liu, and Donghong Ji. 2019. Extracting entities and events as a single task using a transition-based neural model. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 5422–5428. ijcai.org.

Meishan Zhang, Yue Zhang, and Guohong Fu. 2016. Transition-based neural word segmentation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Appendices

A Extra Features

Following the work of Potash et al. (2017) and Kuribayashi et al. (2019), we further incorporate some extra features to represent ACs, including:

- the bag-of-words (BoW) vector: one-hot vector, which is later fed into a MLP layer.
- position embedding of each AC: The position of an AC in the paragraph, which is represented as an embedding vector through a looking-up table $\mathbf{E}_p$.
- paragraph type embedding of each AC: The type (intro, body, conclusion) of the paragraph in which the AC is present, which is also represented as an embedding vector through another looking-up table $\mathbf{E}_t$.