End-to-End Argument Mining as Biaffine Dependency Parsing

Yuxiao Ye
University of Cambridge
Cambridge, UK
yy477@cl.cam.ac.uk

Simone Teufel
University of Cambridge
Cambridge, UK
sht25@cl.cam.ac.uk

Abstract

Non-neural approaches to argument mining (AM) are often pipelined and require heavy feature-engineering. In this paper, we propose a neural end-to-end approach to AM which is based on dependency parsing, in contrast to the current state-of-the-art which relies on relation extraction. Our biaffine AM dependency parser significantly outperforms the state-of-the-art, performing at $F_1 = 73.5\%$ for component identification and $F_1 = 46.4\%$ for relation identification. One of the advantages of treating AM as biaffine dependency parsing is the simple neural architecture that results. The idea of treating AM as dependency parsing is not new, but has previously been abandoned as it was lagging far behind the state-of-the-art. In a thorough analysis, we investigate the factors that contribute to the success of our model: the biaffine model itself, our representation for the dependency structure of arguments, different encoders in the biaffine model, and syntactic information additionally fed to the model. Our work demonstrates that dependency parsing for AM, an overlooked idea from the past, deserves more attention in the future.

1 Introduction

People often hold different opinions about the same thing. One very common way to express one's opinions is to construct an argument (Mercier and Sperber, 2011; Van Eemeren and Henkemans, 2016). To help people efficiently understand different opinions and their reasoning embedded in arguments, it is necessary to develop systems that can automatically analyse structures of arguments. An emerging research field called Argumentation Mining or Argument Mining (AM) (Peldszus and Stede, 2013; Green et al., 2014) addresses this problem.

To analyse the structure of arguments, AM typically proposes four subtasks: 1) *component segmentation*, *i.e.*, cutting a raw sequence into text spans that are either argumentative or non-argumentative segments (only argumentative segments are called argument components); 2) *component classification*, *i.e.*, labelling each argument components with a tag in a pre-defined scheme, (*e.g.*, "PREMISE" or "CLAIM"); 3) *relation detection*, *i.e.*, deciding if two argument components are directly related; and 4) *relation classification*, *i.e.*, categorizing a detected relation into a class in a pre-defined scheme, (*e.g.*, "ATTACK" or "SUPPORT") (Persing and Ng, 2016; Eger et al., 2017; Habernal and Gurevych, 2017; Stab and Gurevych, 2017; Lawrence and Reed, 2020). The first two subtasks are often referred together as *component identification*, and the last two as *relation identification*. An example of the argument structure is illustrated in Figure 1. We see that the "rain" component acts as the premise, supporting the claim of "beautiful".

Some of the AM approaches that try to solve all four subtasks use a pipelined architecture. Independent models are first trained for each subtask; the final results are then achieved by using model ensemble methods such as Integer Linear Programming (Persing and Ng, 2016; Stab and Gurevych, 2017). Like many other pipelined approaches, they often suffer from error propagation. Moreover, many of those models are rule-based or feature-based (Persing and Ng, 2016; Stab and Gurevych, 2017), which require extensive manual efforts and are not flexible or robust in cross-domain scenarios.

As a result, neural end-to-end approaches are desirable for AM. However, it is difficult to jointly model all AM subtasks within one single neural network, because component segmentation and

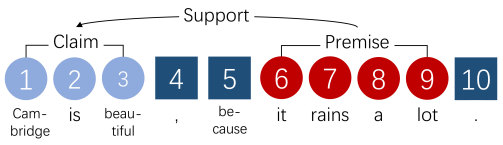

Figure 1: An example argument structure for "*Cambridge is beautiful, because it rains a lot.*"

Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, pages 669–678
April 19 - 23, 2021. ©2021 Association for Computational Linguistics

classification are usually approached at the token-level, but relation detection and classification at the segment-level. Therefore, many researchers try to solve only some of them with neural approaches. For example, Morio and Fujita (2019) focus only on component segmentation and classification; Niculae et al. (2017) and Morio et al. (2020) both ignore the component segmentation task, feeding manually segmented text as input into their models.

As far as we know, Eger et al. (2017) present the first study on neural end-to-end AM that addresses all subtasks in one model. They make a comprehensive comparison of methods, among which a model originally proposed for extracting entities and relations (LSTM-ER) achieves the best performance on a popular AM benchmark (Stab and Gurevych, 2017), far outperforming one that uses a dependency parsing (DP) approach.

In this paper, we propose another neural end-to-end approach for the full AM task, called Biaffine Dependency Parsing for Argument Mining (BiPAM). In our approach, AM is formalised as a DP problem, which we model with a modified biaffine neural network (Dozat and Manning, 2018), using our own dependency representation for arguments. Our representation, like the one in Eger et al. (2017), also unifies all AM subtasks under a token-level framework, so that they can be modelled with one single neural network. The biaffine parser in our approach mainly consists of a neural encoder to extract contextualized features of each token in a sequence, and a biaffine classifier that decides which relation holds between any two tokens, and returns a directed acyclic graph (DAG) expressing this information.

Compared with the DP approach in Eger et al. (2017), our model performs at a much higher performance rate. We argue that this is mainly due to the fact that our biaffine model is more powerful in modelling AM-style dependency structures, and also due to other factors such as our dependency representation, which seems closer aligned with linguistic intuitions. Eger et al. (2017) may have discredited the DP approach altogether due to its unsatisfactory results in their setting, but our work reaffirms its potential for AM, given a more powerful parser and a better dependency representation.

Experiments show that our approach also outperforms the best-performing approach (LSTM-ER) in Eger et al. (2017) and thus achieves a new state-of-the-art. Compared with LSTM-ER, our biaffine parser is conceptually and structurally simpler, and our approach is more general because the output of our biaffine parser is a DAG instead of a tree; this is required by several argument schemes which go beyond trees (Park and Cardie, 2018; Lawrence and Reed, 2020).

Our main contributions are as follows:

- We are the first to apply the biaffine parser to AM in an end-to-end approach addressing all four subtasks in one model. Our proposed biaffine argument parser is applicable to non-tree as well as tree-based argument schemes.
- As well as being a theoretically more clean-cut model, our proposed model also significantly outperforms the state-of-the-art approach by 3.3% in F_1 for component identification and 1.3% for relation identification.
- We also present a novel representation for dependency structures of arguments, which is empirically more efficient than Eger et al. (2017).

2 Related Work

Our work is closely related to existing approaches framing AM as DP and to research on neural end-to-end AM.

The essential aim of AM is to analyse the structure of arguments. Most argument schemes (Toulmin, 2003; Peldszus and Stede, 2013; Habernal and Gurevych, 2017; Visser et al., 2019) represent argument structures as trees or DAGs. Similar structures also exist in syntactic and semantic parsing, and can be efficiently analysed with existing dependency parsers (Dozat and Manning, 2017, 2018; Qi et al., 2018). However, unlike syntactic or semantic parsing, dependency structures in AM operate at the segment-level, not the token-level. To apply existing DP techniques to AM, one can either ignore the component segmentation task, only working on already segmented text, or one can convert segment-level dependencies to token-level dependencies.

Morio et al. (2020) take the first approach. In their work, the input is manually segmented, with argumentative and non-argumentative segments already given. They use task-specific BiLSTMs to encode those segments, and a biaffine dependency parser (Dozat and Manning, 2018) with minor alternation to classify argument components and their

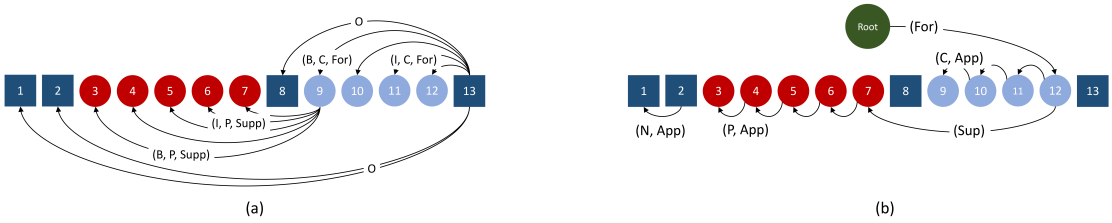

Figure 2: (a) Eger et al. (2017)'s representation, and (b) our representation, for "*Just because it killed much marine life, tourism has threatened nature.*"

relations. They report satisfactory results for component classification and relation identification on the Cornell eRulemaking Corpus (CDCP) (Niculae et al., 2017; Park and Cardie, 2018).

In contrast, Eger et al. (2017) take the second approach, taking raw text as input so that the component segmentation task is addressed. They design a dependency representation for argument structures, in which dependencies are represented at the token-level, with all edges pointing from parent to child[1]. They have evaluated feature-based parsers (McDonald et al., 2005; Bohnet and Nivre, 2012) and neural parsers (Dyer et al., 2015; Kiperwasser and Goldberg, 2016), both of which show unsatisfactory performance. As a result, they discarded DP as inferior.

Our work is similar to Morio et al. (2020) in that we also use a modified biaffine dependency parser to model dependencies in argument structures, but similar to Eger et al. (2017) in that we model AM as a full task starting from tokens, not segments.

Besides DP, Eger et al. (2017) also study other approaches to neural end-to-end AM, including sequence tagging (Ma and Hovy, 2016), multi-task tagging (Søgaard and Goldberg, 2016), and relation extraction (Miwa and Bansal, 2016). Among them, the LSTM-ER model for relation extraction performs the best. However, it requires an additional syntactic parser to produce syntactic dependency trees for the model input. Moreover, the tree-structured LSTM module used to encode those syntactic dependency trees is both conceptually and structurally complicated. In contrast, our proposed approach does not require such syntactic information to break the state-of-the-art (although we will show that it can be enhanced if this information is provided), and the architecture of our biaffine dependency parser is both simpler and more general than that of the LSTM-ER.

[1]Note the difference in arrow conventions between AM and standard DP. Figure 2 shows dependencies pointing from head to dependent, as is common in DP. In contrast, in AM, relations are shown in the opposite direction (*i.e.*, pointing from parent to child), as shown in Figure 2(a). In the rest of this paper, A→B means A pointing to B in terms of argument schemes, while A⇐B means B pointing to A in terms of dependency representations.

3 Proposed Approach

We propose to formalise AM as DP, using a modified biaffine dependency parser, which outputs dependency graphs. To make this work, we first design a dependency representation for arguments which unifies all subtasks of AM under a token-level framework.

3.1 Dependency Representation for Arguments

Our dependency representation for the structure of arguments contains information about segment boundaries, and about types of relations that could potentially hold between segments, as illustrated in Figure 2(b). The relevant properties of our representation are as follows:

- Information on segment boundaries and segment labels is shown as within-segment labelled edges. Within a segment (*e.g.*, $token[3, ..., 7]$), whether argumentative or non-argumentative, each token, except the last one, is parented by its succeeding token. Labels of within-segment edges are in the form of "($segment\_label \in [component\_label, \mathrm{N}]$, APP)", in which $component\_label = \{\mathrm{MC}, \mathrm{C}, \mathrm{P...}\}$. "APP" here means "append", and "N" means "non-argumentative segment".
- Information on relations and relation labels is shown as inter-segment labelled edges. If a relation exists between two components, it is expressed as a labelled edge between the last token of the parent, pointing to the last token of the child (*e.g.*, the "SUPPORT" edge from $token[12]$ to $token[7]$). The last token in a non-argumentative segment does not have any parent node.

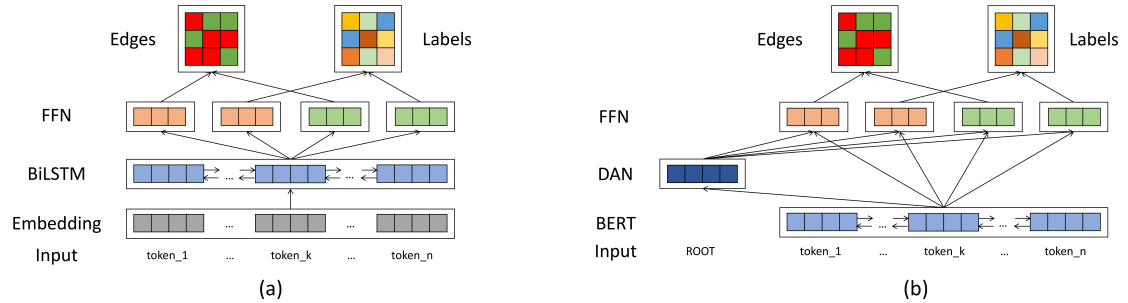

Figure 3: (a) the biaffine parser in Dozat and Manning (2018), and (b) our modified biaffine parser.

- A pseudo-token $ROOT$ is added to the beginning of each argument. This $ROOT$ token can be used to represent the topic or the gist of the entire argument, making it possible to model the relationship between components and the topic or the gist. $ROOT$ is always acting as a parent to the highest-level component(s).
- Each token is allowed to have zero, one, or more parents, resulting in a dependency graph.

In contrast, Eger et al. (2017) use a representation where each token in a segment is parented by the first token in its parent. But this contradicts general linguistic intuitions, whereby tokens within a segment should be closely related to their surrounding tokens, as is the case in our representation. As a result, we might expect that our representation is better able to model within-segment and long-distance inter-segment dependencies. There are also differences between how we and Eger et al. conceptualize the root node of the argument. They use the terminating token in an argument as root, with all non-argumentative tokens parented by it; in their case this is always a punctuation mark. We think this conceptualisation of the root node is troublesome for two reasons. First, there is no reason why all non-argumentative segments should be related to the terminating token. Second, for arbitrary annotation schemes, the terminating token could be a part of a component. When this happens, all non-argumentative tokens will be parented by the last component in an argument, which goes against linguistic intuitions. In contrast, our representation uses the pseudo-node $ROOT$ to avoid this, which is outside all textual segments, and non-argumentative segments never have any inter-segment relations in our representation. In addition, Eger et al. (2017) use the BIO scheme to encode segment boundaries, which results in a larger prediction space than ours, because we can encode the same information topologically[2], making our representation potentially computationally more efficient.

3.2 Modified Biaffine Parser

To our knowledge, the state-of-the-art dependency parser is the one in Dozat and Manning (2018). It consists of a BiLSTM to encoder input text, and a deep biaffine attention module to score each possible head-dependent pair, as shown in Figure 3(a).

The structure of our biaffine parser, as shown in Figure 3(b), closely follows that of Dozat and Manning (2018). We have replaced the Embedding layer and the BiLSTM layer with a pre-trained BERT encoder (Devlin et al., 2019), so that the parser can benefit from rich unsupervised data. Given a text sequence $S = s_1 s_2 ... s_n$, its encoded representation $r_{\_S} \in \mathbb{R}^{n \times d_{enc}}$ is calculated as in Eq. (1), where BERT means the BERT encoder.

$$r_{\_S} = \text{BERT}(s_1 s_2 ... s_n) \quad (1)$$

$$r_{\_ROOT} = \text{FFN}(\text{mean}(r_{\_S}), \text{axis}=0) \quad (2)$$

$$R = [r_{\_S}; r_{\_ROOT}], \text{axis}=0 \quad (3)$$

$$H^{\text{edge\_parent}} = \text{FFN}^{\text{edge\_parent}}(R) \quad (4)$$

$$H^{\text{label\_parent}} = \text{FFN}^{\text{label\_parent}}(R) \quad (5)$$

$$H^{\text{edge\_child}} = \text{FFN}^{\text{edge\_child}}(R) \quad (6)$$

$$H^{\text{label\_child}} = \text{FFN}^{\text{label\_child}}(R) \quad (7)$$

$$\text{Biaff}(x, y) = x^{\top} \text{U} y + \text{W}(x \oplus y) + b \quad (8)$$

$$sc^{\text{edge}} = \text{Biaff}^{\text{edge}}(H^{\text{edge\_parent}}, H^{\text{edge\_child}}) \quad (9)$$

$$sc^{\text{label}} = \text{Biaff}^{\text{label}}(H^{\text{label\_parent}}, H^{\text{label\_child}}) \quad (10)$$

$$y'^{\text{edge}}_{i,j} = \{sc^{\text{edge}}_{i,j} \geq 0\} \quad (11)$$

$$y'^{\text{label}}_{i,j} = argmax\ sc^{\text{label}}_{i,j} \quad (12)$$

$$\mathscr{L} = (1 - \lambda)\mathscr{L}^{\text{edge}} + \lambda \mathscr{L}^{\text{label}}, \lambda \in (0, 1) \quad (13)$$

Eq. (2) shows how $r_{\_ROOT} \in \mathbb{R}^{1 \times d_{enc}}$, the representation of $ROOT$, is calculated using a

[2]Segment boundaries in our system can be recognised by parenthood or by label – when a label does not contain "APP", a segment boundary has occurred.

deep averaging network (DAN) (Iyyer et al., 2015). $ROOT$ does not go through the BERT encoder like other tokens, because it is considered to represent the gist of the whole argument for the dataset in our experiments. FFN means a feedforward network.

In Eq. (3), $R \in \mathbb{R}^{(n+1)\times d_{enc}}$, the $ROOT$-inclusive representation of S, is derived as the concatenation of $r_{\text{-}ROOT}$ and $r_{\text{-}S}$.

In Eq. (4, 5, 6, 7), four representations are created for R respectively, including two parent-wise representations and two child-wise representations for edge and label prediction, as in Dozat and Manning (2018).

Eq. (8) shows the biaffine classifier in Dozat and Manning (2018), with U, W and b being trainable variables. Using this classifier, edges and their labels are predicted respectively as in Eq. (9, 10, 11, 12). Like Dozat and Manning (2018), the edge-classifier is trained with sigmoid cross-entropy, and the label-classifier with softmax cross-entropy. The total loss is calculated as in Eq. (13). During training, losses are back-propagated to the label-classifier only through edges in the gold standard.

Dropout (Srivastava et al., 2014) is applied for each layer in the proposed model.

4 Experiments

We conduct experiments on a benchmark for AM to evaluate the performance of our model.

4.1 Dataset

The dataset we use in our experiments is a benchmark constructed by Stab and Gurevych (2017), which is also used in Eger et al. (2017). This dataset consists of 402 persuasive essays randomly selected from an online forum (322 for training and 80 for testing). Statistics for this dataset are shown in Table 1.

The argument scheme adopted for this dataset

		All	Per Essay
Size	Sentence	7,116	18
	Token	147,271	366
	Paragraph	1,833	5
Components	MajorClaim	751	2
	Claim	1,506	4
	Premise	3,832	10
	Total	6,089	15
Relation	For	2,345	6
	Against	496	1
	Support	3,613	9
	Attack	219	1

Table 1: Statistics of the entire Stab and Gurevych (2017) dataset (test+train).

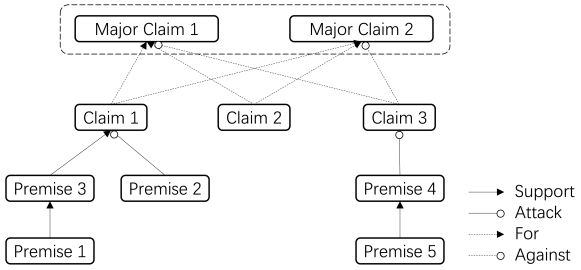

Figure 4: The argument scheme adopted in the Stab and Gurevych (2017) dataset.

is illustrated in Figure 4. In this dataset, there are four kinds of components. A *major claim* (MajorClaim) is the overall stance of the author, which can be regarded as the overall gist of the whole essay. A *claim* (Claim) is a statement that is either *for* (For) or *against* (Against) one or more major claims. A *premise* (Premise) is the lowest-level component that either *supports* (Support) or *attacks* (Attack) a claim or another premise.

Some special characteristics of this dataset are as follows:

- Each essay contains one or more major claims. All major claims within an essay are considered to be equivalent in meaning, and are treated as an equivalence class. In the case of multiple major claims, the structure of the argument is no longer a tree (it is reconstructed as a tree by regarding multiple major claims as one single node in Eger et al. (2017)).
- A claim is allowed to have no supporting or attacking premise.
- Premise→Claim and Premise→Premise pairs are always within-paragraph. Only Claim→MajorClaim pairs are allowed to cut across paragraphs.

Since most relations are within-paragraph for this dataset, our proposed model operates at the paragraph-level. According to our dependency representation, we need a pseudo-token $ROOT$ for each paragraph. If there is one or more major claims in a paragraph, $ROOT$ takes them all as children. Otherwise, $ROOT$ takes all claims in that paragraph as children.

4.2 Competing Models

We choose two models in Eger et al. (2017) for comparison: 1) LSTM-Parser, the best-performing DP model, and 2) LSTM-ER, the overall best-performing model. We use our reimplementation of these models and observe results close to those

reported in Eger et al. (2017). Both models are trained at the paragraph-level, with the default hyperparameter configuration provided in the source code. The same pre-trained GloVe embeddings (Pennington et al., 2014) are used as in Eger et al. (2017)[3]. We use the Stanford syntactic dependency parser (Chen and Manning, 2014) to produce syntactic trees required by the LSTM-ER model.

4.3 Training

We create a development set by choosing 30 essays randomly from the training set, which we use for tuning the interpolation factor λ ($\lambda = 0.05$) in Eq. (13) and dropout rate ($dropout = 0.1$), as well as for the early stopping mechanism. We choose the pre-trained BERT model by OpenAI from Devlin et al. (2019) as the encoder. We abandoned another pre-trained encoder GPT2 (Radford et al., 2019) because it performed slightly below BERT. The hidden size of FFNs and biaffine classifiers is set to 600, in line with Dozat and Manning (2018). We use the Adam optimizer (Kingma and Ba, 2014) ($\beta_1 = 0.95, \beta_1 = 0.98, \epsilon = 1e^{-9}$), and adopt the strategy in Vaswani et al. (2017) for the learning rate, with warm up steps set to 1000.

Each model in our experiments, except LSTM-ER and LSTM-Parser (and their related models), which we run only once[4], is trained ten times with different random initialisations. Results are reported in terms of average performance with standard deviations.

4.4 Final Graph Generation

First, several post-processing techniques are applied to make the parser output compatible with the AM dataset (similar techniques are also applied in Eger et al. (2017)), in the following order.

1. The parser sometimes recognises a segment as (mainly) belonging to one label, but interrupts it with a small number of non-fitting labels. We assume the label intended is the majority label and therefore assign it to the segment, if at least 3/5 of the edges share that same majority label in a sequence of consecutive tokens which are linked together.
2. For the remaining inter-segment edges, only valid edges are kept, namely Claim⇐MajorClaim, Premise⇐Claim, and Premise⇐Premise.
3. If two remaining segments are linked by multiple edges, all edges except the one with the highest probability are deleted.
4. If a remaining segment has multiple inter-segment edges, only Claim⇐MajorClaim edges are kept (as only claims can have multiple parents, which have to be major claims).

[3]This pre-trained GloVe embeddings are used for all LSTM encoders in our experiments.

[4]This is due to excessively long training time.

Component	Model_a	Model_b
g_1	TP	FN
g_2	FN	FP
...	...	...
g_m	TP	TP
p_1	FP	TN
p_2	FP	FP
...	...	...
p_n	TN	FP

Table 2: Result table for paired permutation tests.

Since our model operates at the paragraph-level rather than the essay-level, we need to combine the sub-graph produced with each paragraph. In the Stab and Gurevych (2017) dataset, only Claim→MajorClaim pairs can connect all sub-graphs in an entire essay. As a result, for each Claim→$ROOT$ or Claim→MajorClaim pair, we first eliminate it, and then redirect that claim to all major claims, keeping the original edge label. By doing so, a final graph for the argument structure of the entire essay can be generated.

4.5 Evaluation

Similar to most work in AM, we adopt the evaluation metric in Persing and Ng (2016) to evaluate the performance of AM models. The F_1 scores for component and relation identification are calculated as $F = \frac{2\text{TP}}{2\text{TP}+\text{FP}+\text{FN}}$, in which TP is a true positive, FP false positive, and FN false negative.

We also design a method to apply paired Monte Carlo permutation tests (Dwass, 1957; Nichols and Holmes, 2002) to the results of two AM models. In our situation, results are measured by F_1 scores based on components, but a paired permutation test cannot be straightforwardly applied, because in general the number of components returned by the models is not the same. To address this, a result table is first constructed, as illustrated in Table 2. The set of items consists of the union of : 1) all gold standard components $\{g_1, g_2, ..., g_m\}$, and 2) all predicted components $\{p_1, p_2, ..., p_m\}$ by both models (those that are not duplicates, *i.e.*already contained in the gold standard). The treatment of

Model	Component	Relation
LSTM-Parser	58.4 (58.9)	34.9 (35.6)
LSTM-ER	70.2 (70.8)	45.1 (45.5)
BiPAM	72.9 ± 0.7	45.9 ± 0.5

Table 3: F_1 scores for competing models and our BiPAM model, with published results for competing models (the best instead of average results) in brackets.

Model	Component	Relation
LSTM-Eger (LSTM-Parser)	58.4	34.9
LSTM-ours	67.9	36.0
Biaff-Eger	65.7 ± 0.5	38.4 ± 0.5
Biaff-ours (BiPAM)	72.9 ± 0.7	45.9 ± 0.5

Table 4: F_1 scores for models with different combinations of parsers and representations.

gold standard components is standard for such tests (*i.e.*, TP, FN, FP are possible labels), but the predicted components are added by us, and can only be labelled as TN or FP. As these components are not in the gold standard, each time a model proposes such a component, it will be punished by receiving an FP. A predicted component p_i counts as a true negative (TN) for Model$_a$ if p_i has been predicted by Model$_b$ but not Model$_a$. Permutations are generated by swapping labels in the second and third columns for combinations of randomly selected rows. The two-tailed p-value is then calculated as in Eq. (14, 15), in which F_a means the real F_1 score for Model$_a$, and $F_a^{'i}$ means the F_1 score for Model$_a$ in the i-th permutation. N is the number of sampled permutations. When a model is trained with multiple initialisations, the one performing closest to the average is used for permutation tests.

$$diff_i = \begin{cases} 1 & , |F_a - F_b| < |F_a^{'i} - F_b^{'i}| \\ 0 & , |F_a - F_b| \geq |F_a^{'i} - F_b^{'i}| \end{cases} \quad (14)$$

$$p\text{−value} = \frac{1 + \sum_{i=1}^{N} diff_i}{1 + N} \quad (15)$$

5 Results and Discussion

The overall results of our experiments are shown in Table 3. For both component and relation identification, our model BiPAM significantly ($p < 0.01$) outperforms the state-of-the-art model LSTM-ER (72.9% vs. 70.2% and 45.9% vs. 45.1%), and also significantly outperforms LSTM-Parser by a large margin.

We can see from these results that the DP approach for AM is able to achieve the best results currently known on this dataset. This is interesting, as the current state-of-the-art, LSTM-ER, uses a far more complex neural architecture than our BiPAM.

The equivalent model to our BiPAM in Eger et al. (2017), LSTM-Parser, is outperformed by BiPAM by a large margin. We believe this is due to two factors: a biaffine parser is far more suitable for AM than an LSTM parser, and our dependency representation is superior to Eger et al. (2017)'s. We will now investigate the influence of these two factors. As we have already shown that our version of this approach beats the state-of-the-art, we will consider only those models which treat AM as DP.

5.1 Ablation Study

To understand why our proposed model performs better than LSTM-Parser in Eger et al. (2017), we conduct ablation experiments to assess the influence of our modified biaffine dependency parser and our representation for the dependency structure of arguments.

Besides LSTM-Parser and BiPAM (renamed as LSTM-Eger and Biaff-ours in Table 4), we also experiment with two other combinations of parsers and representations: 1) LSTM-ours, replacing Eger et al. (2017)'s representation in LSTM-Parser with ours; and 2) Biaff-Eger, replacing our representation in BiPAM with Eger et al. (2017)'s. Results of these cross-comparisons are shown in Table 4.

As far as parsers are concerned, our modified biaffine parser with either representation (Biaff-Eger and Biaff-ours) performs significantly ($p < 0.01$) better than its counterpart (LSTM-Eger and LSTM-ours). This suggests that our modified biaffine parser is better at capturing dependencies than the LSTM parser in Eger et al. (2017).

As for representations, either parser with our representation (LSTM-ours and Biaff-ours) significantly ($p < 0.01$) outperforms its counterpart (LSTM-Eger and Biaff-Eger). This demonstrates that our representation is superior.

The example text in Figure 5 is a paragraph in the test data that illustrates this fact. Our BiPAM model correctly predicts all components in it, as well as all the relations. In contrast, in the prediction of Biaff-Eger, the parenthetical phrase "such as Beijing Place[sic], Shanghai Artist Museum," in the relatively long component Premise2, is incorrectly recognized as non-argumentative. Biaff-Eger also fails to predict the relation between Premise3 and Claim, which is probably because these two components are located far from each other at opposite ends of the paragraph, making it difficult to detect the relations between and the first token in

Meanwhile, [historic buildings can be a source of maintenance fees]$_{\text{Claim}}$. [There are many historic buildings, such as Beijing Place, Shanghai Artist Museum, they are all able to support themselves financially by charging tens of thousands of tourists]$_{\text{Premise1}\rightarrow\text{Claim}}$. [Our governments will be happy with those efficient consequences, and a majority of cities also can imitate this economical cycle] $_{\text{Premise2}\rightarrow\text{Claim}}$.	Meanwhile, [historic buildings can be a source of maintenance fees]$_{\text{Claim}}$. [There are many historic buildings]$_{\text{Premise1}\rightarrow\text{Claim}}$, such as Beijing Place, Shanghai Artist Museum, [they are all able to support themselves financially by charging tens of thousands of tourists]$_{\text{Premise2}\rightarrow\text{Claim}}$. [Our governments will be happy with those efficient consequences, and a majority of cities also can imitate this economical cycle] $_{\text{Premise3}}$.
Gold standard & BiPAM	Biaff-Eger

Figure 5: Argument structures of an example paragraph predicted by BiPAM and by Biaff-Eger.

Model	Component	Relation
BiPAM	72.9 ± 0.7	45.9 ± 0.5
BiPAM-LSTM	71.4 ± 0.7	43.1 ± 0.7

Table 5: F_1 scores for BiPAM and BiPAM-LSTM.

Claim and tokens in Premise3.

We also want to understand the influence of the pre-trained BERT encoder in the biaffine parser. When we built BiPAM, we modified the original biaffine parser (Dozat and Manning, 2018) by replacing its Embedding layer and BiLSTM layer, with a pre-trained BERT encoder. It is possible that our performance improvements are at least partially due to this. We therefore test against a biaffine model without this replacement[5], named BiPAM-LSTM in Table 5. Results show that replacing the pre-trained BERT encoder with the original Embedding layer and BiLSTM layer significantly ($p < 0.01$) lowers the performance. This is easy to understand since the pre-trained BERT encoder has benefited from an tremendous amount of additional unsupervised data[6].

5.2 Enhancement with Syntactic Information

We believe that a part of the success of LSTM-ER in Eger et al. (2017) might be credited to the fact that the model is also given syntactic information in its input, in addition to the raw text. We borrow this idea to test whether BiPAM might get a performance boost in a similar way.

To do so, we use the Stanford syntactic dependency parser to produce syntactic trees. The pseudo-token $ROOT$ is headed by the real root node in the syntactic tree, with an edge labelled as "ROOT". For each input token, we record its syntactic information in a triple as [head, location of head, label of in-coming edge]. The syntactic information triples are encoded as follows: 1) head tokens are encoded by an FFN with pre-trained GloVe embeddings; 2) head locations are encoded using the positional encoding method proposed in Vaswani et al. (2017); and 3) edge labels are passed to an FFN using one-hot encoding. For each token, the encoded representation of its syntactic information triple is concatenated to its encoded representation by BERT, and the entire representation is then fed into the following layers in BiPAM.

Model	Component	Relation
BiPAM	72.9 ± 0.7	45.9 ± 0.5
BiPAM-syn	73.5 ± 0.7	46.4 ± 0.6

Table 6: F_1 scores for BiPAM and BiPAM enhanced with syntactic information.

Results in Table 6 show that introducing syntactic information to the input can significantly ($p < 0.01$) boost the performance of the original BiPAM, outperforming LSTM-ER by 3.3% for component identification and 1.3% for relation identification. These results represent our best and final system.

6 Conclusion

In this paper, we solve argument mining (AM) as a neural end-to-end dependency parsing (DP) problem. As a non-pipelined approach, our solution is free from error propagation. We have demonstrated that the biaffine dependency parser from Dozat and Manning (2018) can act as a powerful AM parser, under the right circumstances. We use BERT as the encoder, feed syntactic information into the model in addition to raw text, and propose a novel, efficient and linguistically plausible representation for the dependency structure of arguments. Our model significantly outperforms current state-of-the-art in neural end-to-end AM, performing at $F_1 = 73.5\%$ for component identification and 46.4% for relation identification. Our research suggests that DP, which has been prematurely abandoned in the past, is actually a very promising approach for AM. Through an ablation study, we find that both the modified biaffine parser and the dependency representation contribute to the performance improvement of BiPAM.

Acknowledgments

Yuxiao Ye is supported by a Ph.D. Studentship funded by Toshiba Research Europe Limited.

[5]With this model, there is one change we cannot avoid, and this is that we have to keep the DAN layer for $ROOT$, which does not go through the encoder.

[6]One could also argue that the right way to address the in-domain vs. out-of-domain problem is to train the BERT encoder from scratch using our in-domain training data. We do not perform these experiments, because due to the small data size, it is almost certain that BERT would be over-fitted.

References

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1455–1465.

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)*, pages 740–750.

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, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Timothy Dozat and Christopher D Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of the International Conference on Learning Representations*.

Timothy Dozat and Christopher D Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 484–490.

Meyer Dwass. 1957. Modified randomization tests for nonparametric hypotheses. *The Annals of Mathematical Statistics*, pages 181–187.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343.

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 (Volume 1: Long Papers)*, pages 11–22.

Nancy Green, Kevin D Ashley, Diane Litman, Chris Reed, and Vern Walker. 2014. Proceedings of the first workshop on argumentation mining. In *Proceedings of the First Workshop on Argumentation Mining*.

Ivan Habernal and Iryna Gurevych. 2017. Argumentation mining in user-generated web discourse. *Computational Linguistics*, 43(1):125–179.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*, pages 1681–1691.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

John Lawrence and Chris Reed. 2020. Argument mining: A survey. *Computational Linguistics*, 45(4):765–818.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of human language technology conference and conference on empirical methods in natural language processing*, pages 523–530.

Hugo Mercier and Dan Sperber. 2011. Why do humans reason? arguments for an argumentative theory.

Makoto Miwa and Mohit Bansal. 2016. End-to-end relation extraction using lstms on sequences and tree structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1105–1116.

Gaku Morio and Katsuhide Fujita. 2019. Syntactic graph convolution in multi-task learning for identifying and classifying the argument component. In *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*, pages 271–278. IEEE.

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*, pages 3259–3266.

Thomas E Nichols and Andrew P Holmes. 2002. Nonparametric permutation tests for functional neuroimaging: a primer with examples. *Human brain mapping*, 15(1):1–25.

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 (Volume 1: Long Papers)*, pages 985–995.

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)*.

Andreas Peldszus and Manfred Stede. 2013. From argument diagrams to argumentation mining in texts: A survey. *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, 7(1):1–31.

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.

Isaac Persing and Vincent Ng. 2016. End-to-end argumentation mining in student essays. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1384–1394.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D Manning. 2018. Universal dependency parsing from scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Anders Søgaard and Yoav Goldberg. 2016. Deep multitask learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 231–235.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

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

Stephen E Toulmin. 2003. *The uses of argument*. Cambridge university press.

Frans H Van Eemeren and A Francisca Sn Henkemans. 2016. *Argumentation: Analysis and evaluation*. Taylor & Francis.

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.

Jacky Visser, John Lawrence, Jean Wagemans, and Chris Reed. 2019. An annotated corpus of argument schemes in us election debates. In *Proceedings of the 9th Conference of the International Society for the Study of Argumentation (ISSA), 3-6 July 2018*, pages 1101–1111.