Here's My Point: Joint Pointer Architecture for Argument Mining

Peter Potash, Alexey Romanov, Anna Rumshisky
Department of Computer Science
University of Massachusetts Lowell
{ppotash,aromanov,arum}@cs.uml.edu

Abstract

In order to determine argument structure in text, one must understand how individual components of the overall argument are linked. This work presents the first neural network-based approach to link extraction in argument mining. Specifically, we propose a novel architecture that applies Pointer Network sequence-to-sequence attention modeling to structural prediction in discourse parsing tasks. We then develop a joint model that extends this architecture to simultaneously address the link extraction task and the classification of argument components. The proposed joint model achieves state-of-the-art results on two separate evaluation corpora, showing far superior performance than the previously proposed corpus-specific and heavily feature-engineered models. Furthermore, our results demonstrate that jointly optimizing for both tasks is crucial for high performance.

1 Introduction

An important goal in argument mining is to understand the structure in argumentative text (Persing and Ng, 2016; Peldszus and Stede, 2015; Stab and Gurevych, 2016; Nguyen and Litman, 2016). One fundamental assumption when working with argumentative text is the presence of Arguments Components (ACs). The types of ACs are generally characterized as a *claim* or a *premise* (Govier, 2013), with premises acting as support (or possibly attack) units for claims (though some corpora have further AC types, such as *major claim* (Stab and Gurevych, 2016, 2014b)).

The task of processing argument structure encapsulates four distinct subtasks (our work focuses on subtasks 2 and 3): (1) Given a sequence of tokens that represents an entire argumentative text, determine the token subsequences that constitute non-intersecting ACs; (2) Given an AC, determine the type of AC (*claim*, *premise*, etc.); (3) Given a set/list of ACs, determine which ACs have directed links that encapsulate overall argument structure; (4) Given two linked ACs, determine whether the link is a supporting or attacking relation. This can be labeled as a 'micro' approach to argument mining (Stab and Gurevych, 2016). In contrast, there have been a number of efforts to identify argument structure at a higher level (Boltuzic and Šnajder, 2014; Ghosh et al., 2014; Cabrio and Villata, 2012), as well as slightly re-ordering the pipeline with respect to AC types (Rinott et al., 2015)).

There are two key assumptions our work makes going forward. First, we assume subtask 1 has been completed, i.e. ACs have already been identified. Second, we follow previous work that assumes a tree structure for the linking of ACs (Palau and Moens, 2009; Cohen, 1987; Peldszus and Stede, 2015; Stab and Gurevych, 2016). Specifically, a given AC can only have a single outgoing link, but can have numerous incoming links. Furthermore, there is a 'head' component that has no outgoing link (the top of the tree). Depending on the corpus (see Section 4), an argument structure can be either a single tree or a forest, consisting of multiple trees. Figure 1 shows an example that we will use throughout the paper to concretely explain how our approach works. First, the left side of the figure presents the raw text of a paragraph in a persuasive essay (Stab and Gurevych, 2016), with the ACs contained in square brackets. Squiggly vs straight underlining differentiates between claims and premises, respectively. The ACs have been annotated as to how they are linked, and the right side of the figure reflects this structure. The argument

Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 1364–1373
Copenhagen, Denmark, September 7–11, 2017.

First, [cloning will be beneficial for many people who are in need of organ transplants]$_{AC1}$. In addition, [it shortens the healing process]$_{AC2}$. Usually, [it is very rare to find an appropriate organ donor]$_{AC3}$ and [by using cloning in order to raise required organs the waiting time can be shortened tremendously]$_{AC4}$.

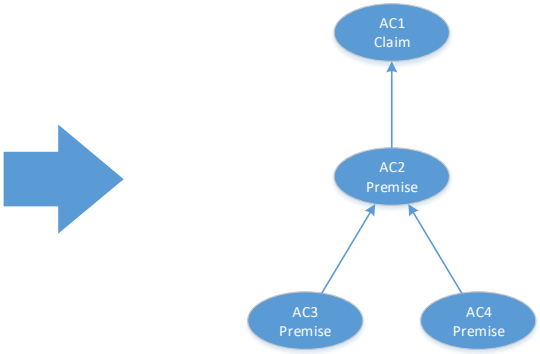

Figure 1: An example of argument structure with four ACs. The left side shows raw text that has been annotated for the presence of ACs. Squiggly or straight underlining means an AC is a *claim* or *premise*, respectively. The ACs in the text have also been annotated for links to other ACs, which is shown in the right figure. ACs 3 and 4 are *premises* that link to another *premise*, AC2. Finally, AC2 links to a *claim*, AC1. AC1 therefore acts as the central argumentative component.

structure with four ACs forms a tree, where AC2 has two incoming links, and AC1 acts as the head, with no outgoing links. We also specify the *type* of AC, with the head AC marked as a *claim* and the remaining ACs marked as *premises*. Lastly, we note that the order of argument components can be a strong indicator of how components should relate. Linking to the first argument component can provide a competitive baseline heuristic (Peldszus and Stede, 2015; Stab and Gurevych, 2016).

Given the above considerations, we propose that sequence-to-sequence attention modeling, in the spirit of a Pointer Network (PN) (Vinyals et al., 2015b), can be effective for predicting argument structure. To the best of our knowledge, a clean, elegant implementation of a PN-based model has yet to be introduced for discourse parsing tasks. A PN is a sequence-to-sequence model (Sutskever et al., 2014) that outputs a distribution over the encoding indices at each decoding timestep. More generally, it is a recurrent model with attention (Bahdanau et al., 2014), and we claim that as such, it is promising for link extraction because it inherently possesses three important characteristics: (1) it is able to model the sequential nature of ACs, (2) it constrains ACs to have a single outgoing link, thus partly enforcing the tree structure, and (3) the hidden representations learned by the model can be used for jointly predicting multiple subtasks. Furthermore, we believe the sequence-to-sequence aspect of the model provides two distinct benefits: (1) it allows for two separate representations of a single AC (one for the source and one for the target of the link), and (2) the decoder network could learn to predict correct sequences of linked indices, which is a second recurrence over ACs. Note that we also test the sequence-to-sequence architecture against a simplified model that only uses hidden states from an encoding network to make predictions (see Section 5).

The main technical contribution of our work is a joint model that simultaneously predicts links between ACs and determines their *type*. Our joint model uses the hidden representation of ACs produced during the encoding step (see Section 3.4). While PNs were originally proposed to allow a variable length decoding sequence, our model differs in that it decodes for the same number of timesteps as there are inputs. This is a key insight that allows for a sequence-to-sequence model to be used for structural prediction. Aside from the partial assumption of tree structure in the argumentative text, our models do not make any additional assumptions about the AC types or connectivity, unlike the work of Peldszus (2014). Lastly, in respect to the broad task of parsing, our model is flexible because it can easily handle non-projective, multi-root dependencies. We evaluate our models on the corpora of Stab and Gurevych (2016) and Peldszus (2014), and compare our results with the results of the aforementioned authors. Our results show that (1) joint modeling is imperative for competitive performance on the link extraction task, (2) the presence of the second recurrence improves performance over a non-sequence-to-sequence model, and (3) the joint model can outperform models with heavy feature-engineering and corpus-specific constraints.

2 Related Work

Palau and Moens (2009) is an early work in argument mining, using a hand-crafted Context-Free Grammar to determine the structure of ACs in a corpus of legal texts. Lawrence et al. (2014) leverage a topic modeling-based AC similarity to uncover tree-structured arguments in philosophical texts. Recent work offers data-driven approaches to the task of predicting links between ACs. Stab and Gurevych (2014b) approach the task as a binary classification problem. The authors train an SVM with various semantic and structural features. Peldszus and Stede have also used classification models for predicting the presence of links (2015). The first neural network-based model for argumentation mining was proposed by Laha and Raykar (2016), who use two recurrent networks in end-to-end fashion to classify AC types.

Various authors have also proposed to jointly model link extraction with other subtasks from the argument mining pipeline, using either an Integer Linear Programming (ILP) framework (Persing and Ng, 2016; Stab and Gurevych, 2016) or directly feeding previous subtask predictions into a tree-based parser. The former joint approaches are evaluated on an annotated corpus of persuasive essays (Stab and Gurevych, 2014a, 2016), and the latter on a corpus of microtexts (Peldszus, 2014). The ILP framework is effective in enforcing a tree structure between ACs when predictions are made from otherwise naive base classifiers.

Recurrent neural networks have previously been proposed to model tree/graph structures in a linear manner. Vinyals et al. (2015c) use a sequence-to-sequence model for the task of syntactic parsing. Bowman et al. (2015) experiment on an artificial entailment dataset that is specifically engineered to capture recursive logic (Bowman et al., 2014). Standard recurrent neural networks can take in complete sentence sequences and perform competitively with a recursive neural network. Multi-task learning for sequence-to-sequence has also been proposed (Luong et al., 2015), though none of the models used a PN for prediction.

In the field of discourse parsing, the work of Li et al. (2016) is the only work, to our knowledge, that incorporates attention into the network architecture. However, the attention is only used in the process of creating representations of the text itself. Attention is *not* used to predict the overall discourse structure. In fact, the model still relies on a binary classifier to determine if textual components should have a link. Arguably the most similar approach to ours is in the field of dependency parsing (Cheng et al., 2016). The authors propose a model that performs 'queries' between word representations in order to determine a distribution over potential headwords.

3 Proposed Approach

In this section, we describe our approach to using a sequence-to-sequence model with attention for argument mining, specifically, identifying AC types and extracting the links between them. We begin by giving a brief overview of these models.

3.1 Pointer Network

A PN is a sequence-to-sequence model (Sutskever et al., 2014) with attention (Bahdanau et al., 2014) that was proposed to handle decoding sequences over the encoding inputs, and can be extended to arbitrary sets (Vinyals et al., 2015a). The original motivation for a pointer network was to allow networks to learn solutions to algorithmic problems, such as the traveling salesperson and convex hull problems, where the solution is a sequence over input points. The PN model is trained on input/output sequence pairs (E, D), where E is the source and D is the target (our choice of E,D is meant to represent the encoding, decoding steps of the sequence-to-sequence model). Given model parameters Θ, we apply the chain rule to determine the probability of a single training example:

$$p(D|E;\Theta) = \prod_{i=1}^{m(E)} p(D_i|D_1, ..., D_{i-1}, E; \Theta) \quad (1)$$

where the function m signifies that the number of decoding timesteps is a function of each individual training example. We will discuss shortly why we need to modify the original definition of m for our application. By taking the log-likelihood of Equation 1, we arrive at the optimization objective:

$$\Theta^* = \arg\max_{\Theta} \sum_{E,D} \log p(D|E;\Theta) \quad (2)$$

which is the sum over all training example pairs.

The PN uses Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) for sequential modeling, which produces a hidden layer h at each encoding/decoding timestep. In practice, the PN has two separate LSTMs, one for

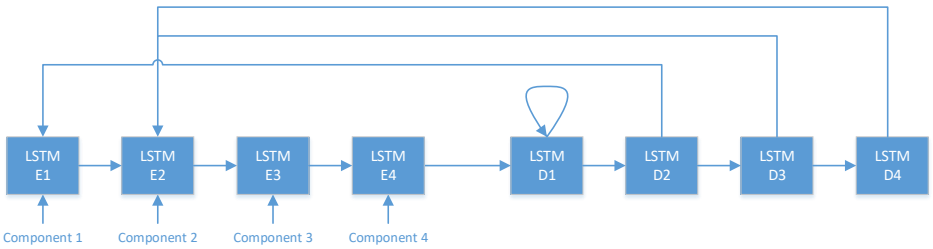

Figure 2: Applying a Pointer Network to the example paragraph in Figure 1 with LSTMs unrolled over time. Note that D1 points to itself to denote that it has not outgoing link and is therefore the head of a tree.

encoding and one for decoding. Thus, we refer to encoding hidden layers as e, and decoding hidden layers as d.

The PN uses a form of content-based attention (Bahdanau et al., 2014) to allow the model to produce a distribution over input elements. This can also be thought of as a distribution over input indices, wherein a decoding step 'points' to the input. Formally, given encoding hidden states $(e_1, ..., e_n)$, the model calculates $p(D_i|D_1, ..., D_{i-1}, E)$ as follows:

$$u^i_j = v^T \tanh(W_1 e_j + W_2 d_i) \quad (3)$$

$$p(D_i|D_1, ..., D_{j-1}, E) = softmax(u^i) \quad (4)$$

where matrices W_1, W_2 and vector v are parameters of the model (along with the LSTM parameters used for encoding and decoding). In Equation 3, prior to taking the dot product with v, the resulting transformation can be thought of as creating a joint hidden representation of inputs i and j. Vector u^i in equation 4 is of length n, and index j corresponds to input element j. Therefore, by taking the softmax of u^i, we are able to create a distribution over the input.

3.2 Link Extraction as Sequence Modeling

A given piece of text has a set of ACs, which occur in a specific order in the text: $(C_1, ..., C_n)$. Therefore, at encoding timestep i, the model is fed a representation of C_i. Since the representation is large and sparse (see Section 3.3 for details on how we represent ACs), we add a fully-connected layer before the LSTM input. Given a representation R_i for AC C_i, the LSTM input A_i is calculated as:

$$A_i = \sigma(W_{rep} R_i + b_{rep}) \quad (5)$$

where W_{rep}, b_{rep} in turn become model parameters, and σ is the sigmoid function[1]. Similarly, the decoding network applies a fully-connected layer with sigmoid activation to its inputs, see Figure 3. At encoding step i, the encoding LSTM produces hidden layer e_i, which can be thought of as a hidden representation of AC C_i.

In order to make sequence-to-sequence modeling applicable to the problem of link extraction, we explicitly set the number of decoding timesteps to be equal to the number of input components. Using notation from Equation 1, the decoding sequence length for an encoding sequence E is simply $m(E) = |\{C_1, ..., C_n\}|$, which is trivially equal to n. By constructing the decoding sequence in this manner, we can associate decoding timestep i with AC C_i.

From Equation 4, decoding timestep i will output a distribution over input indices. The result of this distribution will indicate to which AC component C_i links. Recall there is a possibility that an AC has no outgoing link, such as if it's the root of the tree. In this case, we state that if AC C_i does not have an outgoing link, decoding step D_i will output index i. Conversely, if D_i outputs index j, such that j is not equal to i, this implies that C_i has an outgoing link to C_j. For the argument structure in Figure 1, the corresponding decoding sequence is $(1, 1, 2, 2)$. The topology of this decoding sequence is illustrated in Figure 2. Observe how C_1 points to itself since it has no outgoing link.

Finally, we note that we have a Bidirectional LSTM (Graves and Schmidhuber, 2005) as the encoder, unlike the model proposed by Vinyals et al. (2015b). Thus, e_i is the concatenation of forward and backward hidden states $\overrightarrow{e}_i$ and $\overleftarrow{e}_{n-i+1}$, produced by two separate LSTMs. The decoder remains a standard forward LSTM.

3.3 Representing Argument Components

At each timestep of the encoder, the network takes in a representation of an AC. Each AC is itself

[1]We also experimented with relu and elu activations, but found sigmoid to yield the best performance.

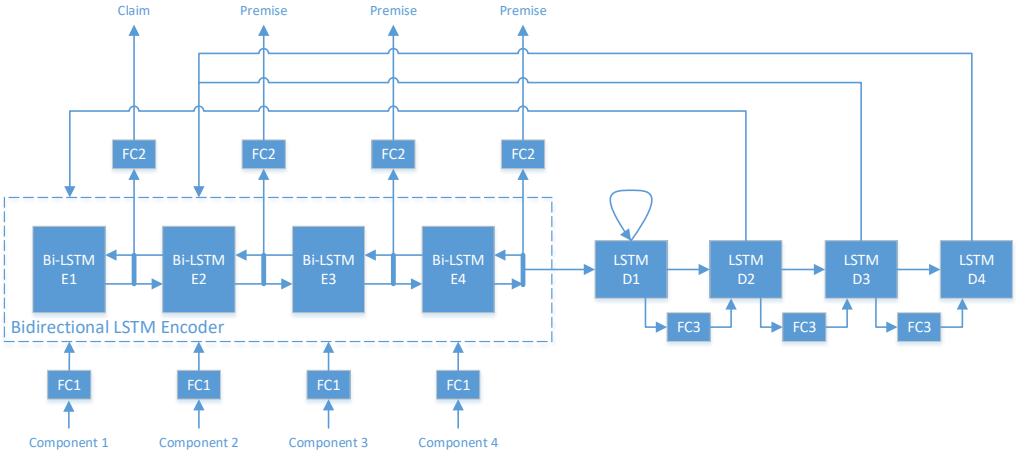

Figure 3: Architecture of the joint model applied to the example in Figure 1. Note that D1 points to itself to denote that it has not outgoing link and is therefore the head of a tree.

a sequence of tokens, similar to the Question-Answering dataset from Weston et al. (2015). We follow the work of Stab and Gurevych (2016) and focus on three different types of features to represent our ACs: (1) Bag-of-Words of the AC; (2) Embedding representation based on GloVe embeddings (Pennington et al., 2014), which uses average, max, and min pooling across the token embeddings; (3) Structural features: Whether or not the AC is the first AC in a paragraph, and whether the AC is in an opening, body, or closing paragraph. See Section 6 for an ablation study of the proposed features.

3.4 Joint Neural Model

Up to this point, we focused on the task of extracting links between ACs. However, recent work has shown that joint models that simultaneously try to complete multiple aspects of the subtask pipeline outperform models that focus on a single subtask (Persing and Ng, 2016; Stab and Gurevych, 2014b; Peldszus and Stede, 2015). Therefore, we will modify the single-task architecture so that it would allow us to perform AC classification (Kwon et al., 2007; Rooney et al., 2012) together with link prediction. Knowledge of an individual subtask's predictions can aid in other subtasks. For example, *claims* do not have an outgoing link, so knowing the type of AC can aid in the link prediction task. This can be seen as a way of regularizing the hidden representations from the encoding component (Che et al., 2015).

At each timestep, predicting AC type is a straightforward classification task: given AC C_i, we need to predict whether it is a *claim*, *premise*, or possibly *major claim*. More generally, this is another sequence modeling problem: given input sequence E, we want to predict a sequence of argument types T. For encoding timestep i, the model creates hidden representation e_i. This can be thought of as a representation of AC C_i. Therefore, our joint model will simply pass this representation through a fully-connected layer as follows:

$$z_i = W_{cls} e_i + b_{cls} \tag{6}$$

where W_{cls}, b_{cls} become elements of the model parameters, Θ. The dimensionality of W_{cls}, b_{cls} is determined by the number of classes. Lastly, we use softmax to form a distribution over the possible classes.

Consequently, the probability of predicting the component type at timestep i is defined as:

$$p(T_i|E_i;\Theta) = softmax(z_i) \tag{7}$$

Finally, combining this new prediction task with Equation 2, we arrive at the new training objective:

$$\begin{aligned}\Theta^* = \arg\max_{\Theta} \alpha \sum_{E,D} \log p(D|E;\Theta) \\ +(1-\alpha) \sum_{E} \log p(T|E;\Theta)\end{aligned} \tag{8}$$

which simply sums the costs of the individual prediction tasks, and the second summation is the cost for the new task of predicting AC type. $\alpha \in [0, 1]$ is a hyperparameter that specifies how we weight the two prediction tasks in our cost function. The architecture of the joint model, applied to our ongoing example, is illustrated in Figure 3.

4 Experimental Design

As we have mentioned, our work assumes that ACs have already been identified. The order of ACs corresponds directly to the order in which the ACs appear in the text. We test the effectiveness of our proposed model on a dataset of persuasive essays (PEC) (Stab and Gurevych, 2016), as well as a dataset of microtexts (MTC) (Peldszus, 2014). The feature space for the PEC has roughly 3,000 dimensions, and the MTC feature space has between 2,500 and 3,000 dimensions, depending on the data split. The PEC contains a total of 402 essays, with a frozen set of 80 essays held out for testing. There are three AC types in this corpus: *major claim*, *claim*, and *premise*. In this corpus, individual structures can be either trees or forests. Also, in this corpus, each essay has multiple paragraphs, and argument structure is only uncovered within a given paragraph. The MTC contains 112 short texts. Unlike the PEC, each text in this corpus is itself a complete example, as well as a single tree. Since the dataset is small, the authors have created 10 sets of 5-fold cross-validation, reporting the the average across all splits for final model evaluation. This corpus contains only two types of ACs: *claim* and *premise*. Note that link prediction is directed, i.e., predicting a link between the pair $C_i, C_j (i \neq j)$ is different than C_j, C_i.

We implement our models in TensorFlow (Abadi et al., 2015). We use the following parameters: hidden input dimension size 512, hidden layer size 256 for the bidirectional LSTMs, hidden layer size 512 for the LSTM decoder, α equal to 0.5, and dropout (Srivastava et al., 2014) of 0.9. We believe the need for such high dropout is due to the small amounts of training data (Zarrella and Marsh, 2016), particularly in the MTC. All models are trained with Adam optimizer (Kingma and Ba, 2014) with a batch size of 16. For a given training set, we randomly select 10% to become the validation set. Training occurs for 4,000 epochs. Once training is completed, we select the model with the highest validation accuracy (on the link prediction task) and evaluate it on the held-out test set. At test time, we take a greedy approach and select the index of the probability distribution (whether link or type prediction) with the highest value.

5 Results

The results of our experiments are presented in Tables 1 and 2. For each corpus, we present f1 scores for the AC type classification experiment, with a macro-averaged score of the individual class f1 scores. We also present the f1 scores for predicting the presence/absence of links between ACs, as well as the associated macro-average between these two values.

We implement and compare four types of neural models: 1) The previously described joint model from Section 3.4 (called Joint Model in the tables); 2) The same as 1), but without the fully-connected input layers (called Joint Model No FC Input in the table); 3) The same as 1), but the model only predicts the link task, and is therefore not optimized for type prediction (called Single-Task Model in the table); 4) A non-sequence-to-sequence model that uses the hidden layers produced by the BLSTM encoder with the same type of attention as the joint model (called Joint Model No Seq2Seq in the table). That is, d_i in Equation 3 is replaced by e_i.

In both corpora we compare against the following previously proposed models: Base Classifier (Stab and Gurevych, 2016) is a feature-rich, task-specific (AC type or link extraction) SVM classifier. Neither of these classifiers enforce structural or global constraints. Conversely, the ILP Joint Model (Stab and Gurevych, 2016) provides constraints by sharing prediction information between the base classifiers. For example, the model attempts to enforce a tree structure among ACs within a given paragraph, as well as using incoming link predictions to better predict the type class *claim*. For the MTC only, we also have the following comparative models: Simple (Peldszus and Stede, 2015) is a feature-rich logistic regression classifier. Best EG (Peldszus and Stede, 2015) creates an Evidence Graph (EG) from the predictions of a set of base classifiers. The EG models the potential argument structure, and offers a global optimization objective that the base classifiers attempt to optimize by adjusting their individual weights. Lastly, MP+p (Peldszus and Stede, 2015) combines predictions from base classifiers with a Minimum Spanning Tree Parser (MSTParser).

6 Discussion

First, we point out that the joint model achieves state-of-the-art on 10 of the 13 metrics in Tables 1 and 2, including the highest results in all metrics on the PEC, as well as link prediction on the MTC. The performance on the MTC is very en-

	Type prediction				Link prediction		
Model	Macro f1	MC f1	Cl f1	Pr f1	Macro f1	Link f1	No Link f1
Base Classifier	.794	.891	.611	.879	.717	.508	.917
ILP Joint Model	.826	.891	.682	.903	.751	.585	.918
Single-Task Model	-	-	-	-	.709	.511	.906
Joint Model No Seq2Seq	.810	.830	.688	.912	.754	.589	.919
Joint Model No FC Input	.791	.826	.642	.906	.708	.514	.901
Joint Model	**.849**	**.894**	**.732**	**.921**	**.767**	**.608**	**.925**

Table 1: Results on the Persuasive Essay corpus. All models we tested are joint models, except for the Single-Task Model model, which only predicts links. All model have a fully-connected input layer, except for the row titled 'Joint Model No FC Input'. See Section 5 for a full description of the models.

	Type prediction			Link prediction		
Model	Macro f1	Cl f1	Pr f1	Macro f1	Link f1	No Link f1
Simple	.817	-	-	.663	.478	.848
Best EG	**.869**	-	-	.693	.502	.884
MP+p	.831	-	-	.720	.546	.894
Base Classifier	.830	.712	.937	.650	.446	.841
ILP Joint Model	.857	.770	.943	.683	.486	.881
Joint Model	.813	.692	.934	**.740**	**.577**	**.903**

Table 2: Results on the Microtext corpus.

couraging for several reasons. First, the fact that the model can perform so well with only a hundred training examples is rather remarkable. Second, although we motivate the use of an attention model due to the fact that it partially enforces a tree structure, other models we compare against explicitly contain further constraints (for example, only premises can have outgoing links). Moreover, the MP+p model directly enforces the single tree constraint unique to the microtext corpus (the PEC allows forests). Even though the joint model does not have the tree constraint directly encoded, it able to learn the structure effectively from the training examples so that it can outperform the Mp+p model for link prediction. As for the other neural models, the joint model with no seq2seq performs competitively with the ILP joint model on the PEC, but trails the performance of the joint model. We believe this is because the joint model is able to create two different representations for each AC, one each in the encoding/decoding state, which benefits performance in the two tasks. We also believe that the joint model benefits from a second recurrence over the ACs, modeling the tree/forest structure in a linear manner. Conversely, the joint model with no seq2seq must encode information relating to type as well as link prediction in a single hidden representation. On one hand, the joint model no seq2seq outperforms the ILP model on link prediction, yet it is not able to match the ILP joint model's performance on type prediction, primarily due to the poor performance on predicting the *major claim* class. Another interesting outcome is the importance of the fully-connected layer before the LSTM input. This extra layer seems to be crucial for improving performance on this task. The results dictate that even a simple fully-connected layer with sigmoid activation can provide a useful dimensionality reduction step. Finally, and arguably most importantly, the single-task model, only optimized for link prediction, suffers a large drop in performance, conveying that the dual optimization of the joint model is vital for high performance in the link prediction task. We believe this is because the joint optimization creates more expressive representations of the ACs, which capture the natural relation between AC type and AC linking.

Table 3 shows the results of an ablation study for AC feature representation. Regarding link prediction, BOW features are clearly the most important, as their absence results in the highest drop in performance. Conversely, the presence of structural features provides the smallest boost in performance, as the model is still able to record state-

	Type prediction				Link prediction		
Model	Macro f1	MC f1	Cl f1	Pr f1	Macro f1	Link f1	No Link f1
No structural	.808	.824	.694	.907	.760	.598	.922
No BOW	.796	.833	.652	.902	.728	.543	.912
No Embeddings	.827	.874	.695	.911	.750	.581	.918
Only Avg Emb*	.832	.873	.717	.917	.751	.583	.918
Only Max Emb*	.843	.874	**.732**	**.923**	.766	**.608**	.924
Only Min Emb*	.838	.878	.719	.918	.763	.602	.924
All features	**.849**	**.894**	**.732**	.921	**.767**	**.608**	**.925**

Table 3: Feature ablation study. * indicates that both BOW and Structural are present, as well as the stated embedding type.

	Type prediction				Link prediction		
Bin	Macro f1	MC f1	Cl f1	Pr f1	Macro f1	Link f1	No Link f1
$1 \leq len < 4$	.863	.902	.798	.889	.918	.866	.969
$4 \leq len < 8$	.680	.444	.675	.920	.749	.586	.912
$8 \leq len < 12$	.862*	.000*	.762	.961	.742	.542	.941

Table 4: Results of binning test data by length of AC sequence. * indicates that this bin does not contain any *major claim* labels, and this average only applies to *claim* and *premise* classes. However, we do not disable the model from predicting this class: the model was able to avoid predicting this class on its own.

of-the-art results compared to the ILP Joint Model. This shows that the Joint Model is able to capture structural cues through sequence modeling and semantics. When considering type prediction, both BOW and structural features are important, and it is the embedding features that provide the least benefit. The ablation results also provide an interesting insight into the effectiveness of different pooling strategies for using individual token embeddings to create a multi-word embedding. The popular method of averaging embeddings (which is used by Stab and Gurevych (2016) in their system) is in fact the worst method, although its performance is still competitive with the previous state-of-the-art. Conversely, max pooling results are on par with the joint model results in Table 1.

Table 4 shows results on the PEC test set with the test examples binned by sequence length. First, it is not surprising to see that the model performs best when the sequences are the shortest (for link prediction; type prediction actually sees the worst performance in the middle bin). As the sequence length increases, the accuracy on link prediction drops. This is possibly due to the fact that as the length increases, a given AC has more possibilities as to which other AC it can link to, making the task more difficult. Conversely, there is actually a rise in no link prediction accuracy from the second to third row. This is likely due to the fact that since the model predicts at most one outgoing link, it indirectly predicts no link for the remaining ACs in the sequence. Since the chance probability is low for having a link between a given AC in a long sequence, the no link performance is actually better in longer sequences. The results of the length-based binning could also potentially give insight into the poor performance on the type prediction task in the MTC. Since the arguments in the MTC average 5 ACs, they would be in the second bin (row 2) of Table 4. The *claim* and *premise* f1 scores for this bin are similar to those from the same system's performance on the MTC.

7 Conclusion

In this paper we have proposed how to use a joint sequence-to-sequence model with attention (Vinyals et al., 2015b) to both extract links between ACs and classify AC type. We evaluate our models on two corpora: a corpus of persuasive essays (Stab and Gurevych, 2016), and a corpus of microtexts (Peldszus, 2014). The Joint Model records state-of-the-art results on the persuasive essay corpus, as well as achieving state-of-the-art results for link prediction on the microtext corpus. The results show that jointly modeling the two prediction tasks is critical for high performance. Future work can attempt to learn the AC representations themselves, such as in Kumar et al. (2015).

Lastly, future work can integrate subtasks 1 and 4 into the model. The representations produced by Equation 3 could potentially be used to predict link type, i.e. supporting or attacking (the fourth subtask in the pipeline). In addition, a segmenting technique, such as the one proposed by Weston et al. (2014), can accomplish subtask 1.

Acknowledgments

This work was supported in part by the U.S. Army Research Office under Grant No. W911NF-16-1-0174.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Filip Boltuzic and Jan Šnajder. 2014. Back up your stance: Recognizing arguments in online discussions. In *Proceedings of the First Workshop on Argumentation Mining*, pages 49–58. Citeseer.

Samuel R Bowman, Christopher D Manning, and Christopher Potts. 2015. Tree-structured composition in neural networks without tree-structured architectures. *arXiv preprint arXiv:1506.04834*.

Samuel R Bowman, Christopher Potts, and Christopher D Manning. 2014. Recursive neural networks can learn logical semantics. *arXiv preprint arXiv:1406.1827*.

Elena Cabrio and Serena Villata. 2012. Natural language arguments: A combined approach. In *ECAI*, volume 242, pages 205–210.

Zhengping Che, David Kale, Wenzhe Li, Mohammad Taha Bahadori, and Yan Liu. 2015. Deep computational phenotyping. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 507–516. ACM.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. 2016. Bi-directional attention with agreement for dependency parsing. *arXiv preprint arXiv:1608.02076*.

Robin Cohen. 1987. Analyzing the structure of argumentative discourse. *Computational linguistics*, 13(1-2):11–24.

Debanjan Ghosh, Smaranda Muresan, Nina Wacholder, Mark Aakhus, and Matthew Mitsui. 2014. Analyzing argumentative discourse units in online interactions. In *Proceedings of the First Workshop on Argumentation Mining*, pages 39–48.

Trudy Govier. 2013. *A practical study of argument*. Cengage Learning.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610.

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

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

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. 2015. Ask me anything: Dynamic memory networks for natural language processing. *arXiv preprint arXiv:1506.07285*.

Namhee Kwon, Liang Zhou, Eduard Hovy, and Stuart W Shulman. 2007. Identifying and classifying subjective claims. In *Proceedings of the 8th annual international conference on Digital government research: bridging disciplines & domains*, pages 76–81. Digital Government Society of North America.

Anirban Laha and Vikas Raykar. 2016. An empirical evaluation of various deep learning architectures for bi-sequence classification tasks. In *COLING*.

John Lawrence, Chris Reed, Colin Allen, Simon McAlister, Andrew Ravenscroft, and David Bourget. 2014. Mining arguments from 19th century philosophical texts using topic based modelling. In *Proceedings of the First Workshop on Argumentation Mining*, pages 79–87. Citeseer.

Qi Li, Tianshi Li, and Baobao Chang. 2016. Discourse parsing with attention-based hierarchical neural networks. In *EMNLP*.

Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2015. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*.

Huy V Nguyen and Diane J Litman. 2016. Context-aware argumentative relation mining.

Raquel Mochales Palau and Marie-Francine Moens. 2009. Argumentation mining: the detection, classification and structure of arguments in text. In *Proceedings of the 12th international conference on artificial intelligence and law*, pages 98–107. ACM.

Andreas Peldszus. 2014. Towards segment-based recognition of argumentation structure in short texts. *ACL 2014*, page 88.

Andreas Peldszus and Manfred Stede. 2015. Joint prediction in mst-style discourse parsing for argumentation mining. In *Proc. of the Conference on Empirical Methods in Natural Language Processing*, pages 938–948.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–43.

Isaac Persing and Vincent Ng. 2016. End-to-end argumentation mining in student essays. In *Proceedings of NAACL-HLT*, pages 1384–1394.

Ruty Rinott, Lena Dankin, Carlos Alzate Perez, Mitesh M Khapra, Ehud Aharoni, and Noam Slonim. 2015. Show me your evidence-an automatic method for context dependent evidence detection. In *EMNLP*, pages 440–450.

Niall Rooney, Hui Wang, and Fiona Browne. 2012. Applying kernel methods to argumentation mining. In *FLAIRS Conference*.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Christian Stab and Iryna Gurevych. 2014a. Annotating argument components and relations in persuasive essays. In *COLING*, pages 1501–1510.

Christian Stab and Iryna Gurevych. 2014b. Identifying argumentative discourse structures in persuasive essays. In *EMNLP*, pages 46–56.

Christian Stab and Iryna Gurevych. 2016. Parsing argumentation structures in persuasive essays. *arXiv preprint arXiv:1604.07370*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2015a. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015b. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015c. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. 2015. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory networks. *arXiv preprint arXiv:1410.3916*.

Guido Zarrella and Amy Marsh. 2016. Mitre at semeval-2016 task 6: Transfer learning for stance detection. *arXiv preprint arXiv:1606.03784*.