Jointly Extracting Event Triggers and Arguments by Dependency-Bridge RNN and Tensor-Based Argument Interaction

Abstract

Event extraction plays an important role in natural language processing (NLP) applications including question answering and information retrieval. Traditional event extraction relies heavily on lexical and syntactic features, which require intensive human engineering and may not generalize to different datasets. Deep neural networks, on the other hand, are able to automatically learn underlying features, but existing networks do not make full use of syntactic relations. In this paper, we propose a novel dependency bridge recurrent neural network (dbRNN) for event extraction. We build our model upon a recurrent neural network, but enhance it with dependency bridges, which carry syntactically related information when modeling each word. We illustrates that simultaneously applying tree structure and sequence structure in RNN brings much better performance than only uses sequential RNN. In addition, we use a tensor layer to simultaneously capture the various types of latent interaction between candidate arguments as well as identify/classify all arguments of an event. Experiments show that our approach achieves competitive results compared with previous work.

Introduction

Event extraction plays an important role in various NLP applications including question answering and information retrieval (Yang et al. 2003; Glavaš and Šnajder 2014). Figure 1 shows an example of the event extraction task, which aims to discover events (*die* and *attack*) with triggering words (*died* and *fired*) and their corresponding arguments (e.g., *Baghdad*, *cameraman*). Typically, event extraction can be divided into several subtasks: trigger identification, argument identification, and argument role classification, as defined by the ACE 2005 dataset, a benchmark for event extraction (Grishman, Westbrook, and Meyers 2005).

In early years, researchers have designed various features, e.g., lexical and contextual ones, for event extraction (Grishman, Westbrook, and Meyers 2005; Ji and Grishman 2008; Lu and Roth 2012). Such methods, however, require extensive human engineering, which also largely affects model performance.

Recently, neural networks are widely applied in NLP tasks including event extraction, as they can automatically extract underlying features. Chen et al. (2015) explore convolutional neural networks (CNNs) and Nguyen, Cho, and Grishman (2016) explore recurrent neural networks (RNNs) in the event extraction task. A potential limitation of the above methods is that they do not make use of syntactic features during the design of neural architectures. Nguyen, Cho, and Grishman (2016) use a binary vector to model dependency relations, but the method is still weak in terms of syntactic modeling. Another limitation in existing approaches is the lack of argument-argument interaction. The observation is that jointly modeling all argument candidates enables a more global view of the relationship among arguments, which is beneficial to argument identification and classification. In Figure 1, for example, when deciding the role of `Palestine Hotel` with the trigger `fire`, we find that `American tank` and `the Palestine Hotel` have the common dependency parent `fire`. Therefore, if `American tank` is an argument, then the probability of `the Palestine Hotel` also being an argument is increased.

In this paper, we propose a dependency bridge recurrent neural network (dbRNN) for event extraction. Our model is built upon a bidirectional recurrent neural network (RNN) with long short term memory (LSTM) units, but we enhance it with dependency bridges to connect syntactically related words. We then build a tensor layer on each pair of two candidate arguments, enabling intensive argument-level information interaction. During training, we adopt a max-margin criterion to jointly extract event triggers and arguments.

We evaluated dbRNN on the ACE 2005 dataset. Experimental results show that our model outperforms previous state-of-the-art approaches in terms of all subtasks. We also made efforts in visualizing tensor features to better understand our model.

Task Description

We describe the event extraction task by following the convention in Automatic Content Extraction (ACE) 2005[1]; Table 1 summarizes relevant terminologies. As said, there are three subtasks in event extraction:

- Trigger identification, which aims to identify the most important word that characterizes an event and to classify it

[1]https://catalog.ldc.upenn.edu/ldc2006t06

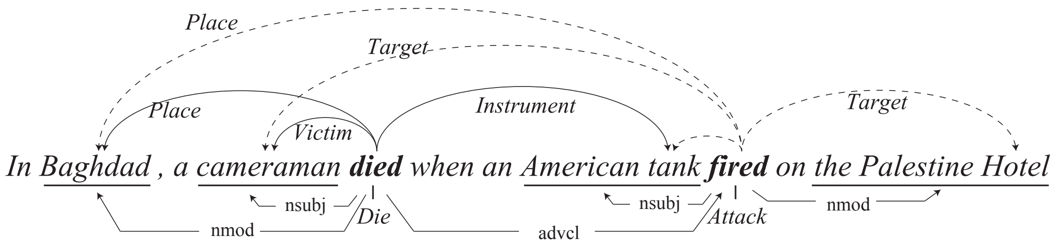

Place
Target
Place
Victim
Instrument
Target
In Baghdad , a cameraman died when an American tank fired on the Palestine Hotel
nsubj
Die
nmod
advcl
nsubj
Attack
nmod

Figure 1: An example of event mentions. There are two event mentions that share three arguments, namely the Die event mention triggered by "died", and the Attack event mention triggered by "fired". The arrows under the sentence is the sample of dependency parse edges.

Entity	an object or a set of objects in one of the semantic categories
Entity mention	a reference to an entity, usually a noun phrase (NP)
Event trigger	main word which most clearly expresses an event occurrence
Event arguments	the entity mentions that are involved in an event
Argument roles	the relation of arguments to the event where they participate (35 total possible roles defined by ACE)
Event mention	a phrase or sentence within which an event is described including trigger and arguments

Table 1: Glossary (Grishman, Westbrook, and Meyers 2005).

into 33 predefined, fine-grained categories. For evaluation purposes, a trigger is considered correct if both the identified word and its type match the groundtruth reference.

- Argument identification. Each argument of an event is an *entity mention*, which has already been tagged by the ACE corpus. This subtask aims to identify if an entity mention is an argument for a particular event. We say an argument is correctly identified if the argument itself and its event type are correct.
- Argument classification. This subtask aims to determine the *role* (e.g., "victim") of an identified argument. An argument is correctly classified if it is correctly identified and its role matches the reference.

Approach

In this section, we elaborate the proposed model. We start from the basics of bidirectional long short term memory RNNs. Then we introduce the notion of dependency bridges and the tensor interaction for argument-argument modeling. Finally, we present the max-margin criteria by which event triggers and arguments are jointly learned.

Long Short Term Memory

The recurrent neural network (RNN) is suitable for modeling sequential data as it keeps a set of hidden states, which evolve over discrete time steps according to the input. However, vanilla RNNs are difficult to train because of gradient exploding or vanishing problem during backpropagation over time. Long short term memory (LSTM) units are proposed to addresses this problem (Hochreiter and Schmidhuber 1997).

Formally, LSTM states at a time step t are a collection of vectors in $\mathbb{R}^d$: an input gate i_t, a forget gate f_t, an output gate o_t, a memory cell c_t, and a candidate memory cell state $\widetilde{c}_t$ and a hidden state h_t. The entries of the gating vectors i_t, f_t and o_t are in $[0, 1]$. We refer to d as the memory dimension of the LSTM. The LSTM update equations are

$$\begin{bmatrix} i_t \\ f_t \\ o_t \\ \widetilde{c}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \left(W_L \cdot \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b_L\right) \quad (1)$$
$$c_t = i_t \odot \widetilde{c}_t + f_t \odot c_{t-1}$$
$$h_t = o_t \odot \tanh(c_t)$$

where x_t is the input at the current time step, σ denotes the logistic sigmoid function, and $\odot$ denotes element-wise multiplication. $W_L \in \mathbb{R}^{4d\times 2d}$ and $b_L \in \mathbb{R}^{4d\times 1}$ are parameters. Intuitively, the forget gate controls the extent to which the previous memory cell is forgotten, the input gate controls how much information is input to each unit, and the output gate controls the exposure of the internal memory state. The hidden state vector in an LSTM unit is therefore a gated, partial view of the state of the unit's internal memory cell. Since the value of the gating variables vary for each vector element, the model can learn to represent information over multiple time scales.

Bidirectional LSTM

We enhance the above LSTM with bidirectionality, which is proposed in Schuster and Paliwal (1997). The hidden states of bidirectional RNN (BiLSTM) are computed both in forward and backward ways at each time step. We denote the outputs of the forward and backward LSTMs as $h_t^{\rightarrow}$ and $h_t^{\leftarrow}$, respectively.

$$h_t^{\rightarrow} = \mathrm{LSTM}^{\rightarrow}(h_{t-1}^{\rightarrow}, x_t)$$
$$h_t^{\leftarrow} = \mathrm{LSTM}^{\leftarrow}(h_{t+1}^{\leftarrow}, x_t) \quad (2)$$

Then the output at time t is $h_t = [h_t^{\rightarrow}, h_t^{\leftarrow}]$.

Dependency Bridges

In this part, we propose dependency bridges over Bi-LSTM for event extraction.

Figure 1 shows the dependency parse tree of a sentence. We see that there is an "advcl" edge between "died" and "fired" and a "nmod" edge between "fired" and "hotel." These kinds of dependency edges contain useful information

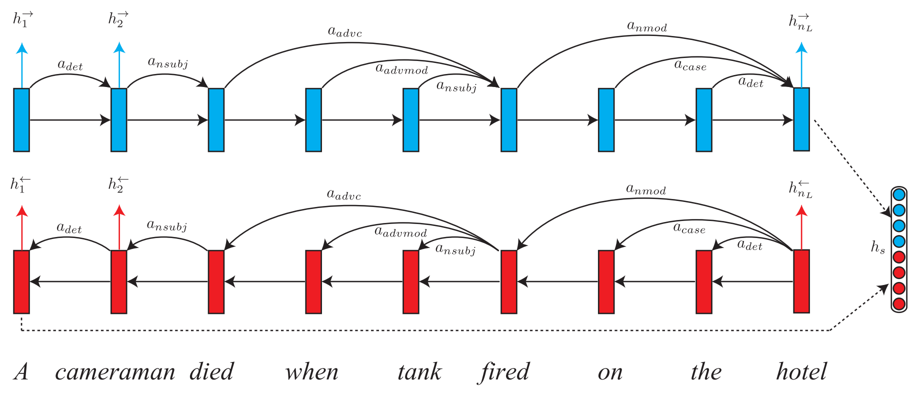


Figure 2: Dependency bridge on LSTM. Apart from the last LSTM cell, each cell also receives information from former syntactically related cells.

about temporal, consequence, conditional, purpose, etc., and are shown to be helpful for joint feature-based event extraction in (Li, Ji, and Huang 2013). Since traditional RNN cannot take advantage of dependency relations directly, we propose to add dependency edges to the BiLSTM architecture as a shortcut that brings syntactically related information directly to the current LSTM cell. We call this "dependency bridges."

The proposed dependency bridges are shown in Figure 2. We assign each type of dependency relations (e.g., `nsubj`, `dobj`) a weight; different directions of the same dependency relation also have different weights. For example, the forward direction of "nsubj" has weight a_{+nsubj}, the backward direction has weight a_{-nsubj}.

During LSTM information propagation, the dependency bridge information mainly affects the hidden layer of the LSTM cell. In the tth step, we denote the set of dependency bridges linked to the current cell as S_{in}. Each element in S_{in} is a pair (index, type) representing the source index of the dependency edge and the dependency type, respectively. Then the hidden layer is computed as

$$h_t = o_t \odot \tanh(c_t) + d_t \odot \Big(\frac{1}{|S_{in}|} \sum_{(i,p)\in S_{in}} a_p h_i\Big) \tag{3}$$

where d_t is a new gate we set to avoid the dependency information affect original information too much. d_t is computed the same way as other gates:

$$d_t = \sigma(W_d \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b_d) \tag{4}$$

where $W_d \in \mathbb{R}^{d\times 2d}$, $b_d \in \mathbb{R}^{d\times 1}$ are parameters.

The main difference between DB and previous tree-based methods, such as tree-LSTM (Tai, Socher, and Manning 2015), tree-CNN (Mou et al. 2015b; 2015a; 2016), recursive-NN (Socher et al. 2013; 2011), is that all of them are using tree method alone while we are trying to use tree method and sequential method together.

Trigger Classification

When extracting event triggers, we enumerate every noun, verb, and adjective in the sentence as candidate triggers. The encoded representation h_s of a sentence of length N_L is the concatenation of the final forward and backward outputs, $h_s = [h_{N_L}^{\rightarrow}, h_1^{\leftarrow}]$. Then we concatenate the corresponding BiLSTM's output of the candidate trigger h_{tri} and the sentence's encoded vector S together as the candidate trigger's feature $C = [h_{tri}, h_s]$. Then we feed the feature into a multilayer perceptron

$$\begin{aligned} H_C &= \tanh(W_c C + b_c) \\ O_T &= \text{softmax}(W_T H_C + b_T) \end{aligned} \tag{5}$$

where $W_c \in \mathbb{R}^{n_c \times 2n_{hu}}$, $b_c \in \mathbb{R}^{n_c}$, $W_T \in \mathbb{R}^{n_{ET}\times n_c}$, $b_T \in \mathbb{R}^{n_{ET}}$ are parameters for training. Here, n_c is the length of H_C, n_{hu} is the BiLSTM's hidden layer size, and n_{ET} is the number of event types (including the "NoEvent" type). Each entry of $O_T \in \mathbb{R}^{n_{ET}}$ represents the probability of an event type.

Argument Classification

When classifying the arguments, we need to consider the argument-argument interaction. The argument-argument interaction is very important in argument identification and classification. Intuitively, when considering all the candidate arguments together, the differences and commonality between them are much easier to be figured out, which is beneficial to argument identification and classification. The argument-argument interaction includes various types of argument relations, such as, whether two candidate arguments have the same dependency parent, whether two candidate arguments are semantically coherent, etc.

Sha et al. (2016) proposed to split the argument-argument interaction into two categories: positive and negative. In real case, these two interactions can only determine whether to identify the arguments. As for which role to take, we need much more complicated interactions like "whether these two candidate arguments have opposed roles". In this paper, we propose to represent the interactions by a vector.

When considering all candidate arguments simultaneously, as is shown in Figure 3, we collect all of their corresponding hidden layers (the BiLSTM's output) as a matrix $H \in \mathbb{R}^{n_{hu}\times n_A}$, where n_A is the number of candidate arguments.

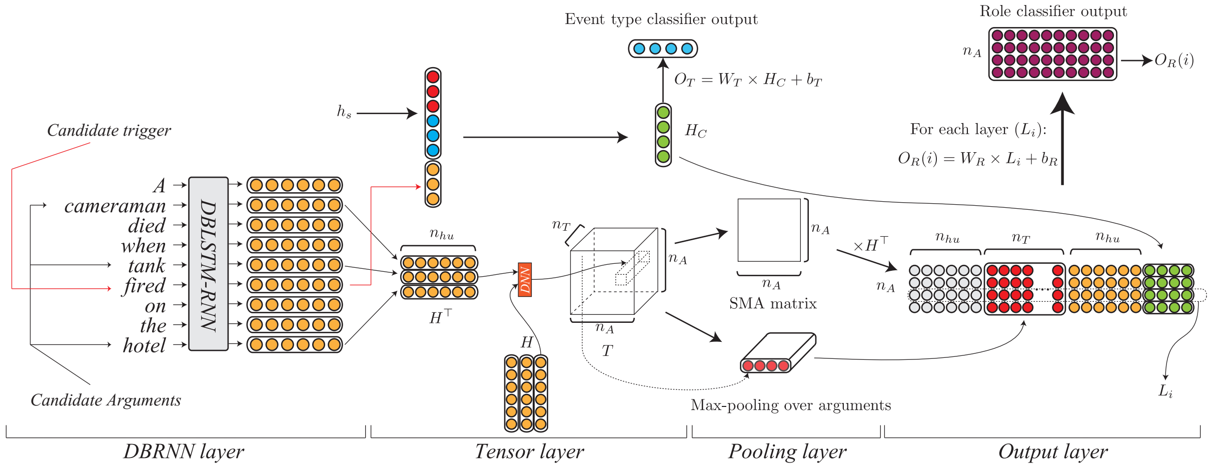


Figure 3: The architecture of dependency-bridge recurrent neural network.

For each pair of arguments, we use a feed-forward layer to predict the relationship between the two arguments. Each argument-argument relationship is represented by a vector of length n_T. Given the hidden layers $H = \{h_i | i \in [1, n_A]\}$, This process is stated as follows:

$$T_{ij} = \tanh(W_d[h_i, h_j] + b_d) \tag{6}$$

where $W_d \in \mathbb{R}^{n_T \times 2n_{hu}}$, $b_d \in \mathbb{R}^{n_T}$ are transformation matrix and bias vector.

Then we can obtain a 3D tensor T which directly models the interactions. Intuitively, each element of T_{ij} represents a kind of argument-argument relationship. Inspired by Collobert et al. (2011), we use max-pooling to capture the most useful interaction feature over arguments as shown in Equation 7.

$$F_{maxp}(i, k) = \max_{j=1}^{n_A} T(i, j, k) \tag{7}$$

Where $F_{maxp} \in \mathbb{R}^{n_A \times n_T}$ is the max-pooled interaction feature of the n_A candidate arguments.

To judge whether two candidate arguments tend to occur together, we also add the argument-argument self-matching attention (SMA) matrix into our architecture for capturing `pos` and `neg` interactions proposed by Sha et al. (2016). Different from them, the value in our SMA matrix ranges from 0 to 1 instead of only three values $(-1, 0, 1)$. We use a feed-forward layer to compute the SMA matrix A:

$$A_{ij} = \text{softmax}(\tanh(W_a T_{ij} + b_a)) \tag{8}$$

An element in SMA matrix represents the possibility of two candidate arguments occurring in one event. Then we multiply A and the candidate arguments' hidden layer matrix H so that the SMA matrix dynamically collects evidence from all attended candidate arguments in the sentence, given by

$$F_A = AH^\top, \quad F_A \in \mathbb{R}^{n_A \times n_{hu}} \tag{9}$$

Since the hidden layer matrix H contains lexical features and contextual features, then we concatenate the interaction features F_A and F_{maxp}, the contextual features H, and the trigger's feature H_C together

$$L = [F_A, F_{maxp}, H^T, H_C \otimes e_{n_A}] \tag{10}$$

where $H_C \otimes e_{n_A}$ means to copy H_C for n_A times, and $L \in \mathbb{R}^{n_A \times (n_T + 2n_{hu} + n_c)}$. In L, each candidate argument has a corresponding feature vector L_i with length $(n_T + 2n_{hu} + n_c)$, which contains lexical features, contextual features, interaction features, and trigger's features. Then L_i is fed into a classifier:

$$O_R(i) = \text{softmax}(W_R L_i + b_R) \tag{11}$$

where $O_R(i) \in \mathbb{R}^{1 \times n_R}$ is the final output of the i-th candidate argument and $O_R(i, j)$ is the score for argument role j.

Max-Margin Training for Joint Model

We use the Max-Margin criterion to train our model. Intuitively, the Max-Margin criterion provides an alternative to probabilistic, likelihood-based estimation methods by concentrating directly on the robustness of the decision boundary of a model (Taskar et al. 2005).

In order to simultaneously extract the event trigger and arguments, we use $Y(x_i)$ to denote the set of all possible event types (p_i) and role label sequences (y_i) for the candidate event[2] in a given sentence x_i.

We define a structured margin loss $\Delta(y_i, p_i, \hat{y}, \hat{p})$ based on a predicted role label sequence $\hat{y}$, a predicted event type $\hat{p}$, a given correct role label sequence y_i, and a given correct event type p_i:

$$\Delta(y_i, p_i, \hat{y}, \hat{p}) = \kappa \sum_{j=1}^{n_A} \mathbf{1}\{y_{ij} \neq \hat{y}_j\} + \mathbf{1}\{p_i \neq \hat{p}\} \tag{12}$$

where κ is a discount parameter. The loss is proportional to the number of candidate arguments with an incorrect role label or incorrect event type in the predicted event. For x_i in an input instance with a role label sequence y_i and an event type p_i, we define a sentence-level score by the sum and multiplication of network scores for joint extraction of

[2]The event triggered by the candidate trigger.

event triggers and arguments:

$$s(x_i, y_i, p_i; \theta) = O_T(p_i) + \mu \sum_{j=1}^{n} O_R(j, y_{ij}) \tag{13}$$

where μ is a discount parameter, and the parameters of our model are represented by θ. For a given training instance (x_i, y_i, p_i), the score of the correct event type p_i and role label sequence y_i will be larger up to a margin to other possible event type and role label sequences $(\hat{p}, \hat{y}) \in Y(x_i)$:

$$s(x_i, y_i, p_i; \theta) \leq s(x_i, \hat{y}, \hat{p}; \theta) + \Delta(y_i, p_i, \hat{y}, \hat{p}) \tag{14}$$

This leads to the regularized objective function for m training examples:

$$\begin{aligned} J(\theta) =& \frac{1}{m} \sum_{i=1}^{m} l_i(\theta) + \frac{\lambda}{2} \|\theta\|^2 \\ l_i(\theta) =& \max_{\hat{y} \in Y(x_i)} (s(x_i, \hat{y}, \hat{p}, \theta) + \Delta(y_i, p_i, \hat{y}, \hat{p}) \\ & - s(x_i, y_i, p_i, \theta)) \end{aligned} \tag{15}$$

By minimizing this objective, the score of the correct event type and role label sequence (p_i, y_i) is increased and score of the highest scoring incorrect event type and sequence $(\hat{p}, \hat{y})$ is decreased.

Due to the hinge loss, the objective function is not differentiable. We use subgradient method (Ratliff, Bagnell, and Zinkevich 2007) to compute a gradient-like direction. The subgradient of Equation 15 is:

$$\begin{aligned} \frac{\partial J}{\partial \theta} = \frac{1}{m} \sum_{i} \Big(& \frac{\partial s(x_i, \hat{y}_{max}, \hat{p}_{max}; \theta)}{\partial \theta} \\ & - \frac{\partial s(x_i, y_i, p_i; \theta)}{\partial \theta} \Big) + \lambda \theta \end{aligned} \tag{16}$$

where $(\hat{p}_{max}, \hat{y}_{max})$ is the event type and role label sequence with the highest score in Equation 15. To compute the network parameter θ, we use Adadelta (Zeiler 2012) with shuffled mini-batches to minimize the objective.

Experiment

Data

We evaluate our dbRNN model on the ACE 2005 dataset. To comply with previous work, we use a pre-defined split of the documents provided by Li, Ji, and Huang (2013), in which the newswire texts in ACE2005 dataset are divided into 529 training documents (14,840 sentences), 30 developing documents (863 sentences) and 40 testing documents (672 sentences).

Word embeddings are obtained using WORD2VEC[3] and trained by the default "text8" data with embedding size 100. The hyperparameters of our model are chosen as in Table 2 according to the development set.

When predicting the candidate arguments, we use Stanford constituency parser[4] to parse every sentence in the corpus, and then take the phrases attached to NP nodes as the candidate arguments. Also, we use Stanford dependency parser[5] to decide the dependency bridges.

[3]http://code.google.com/p/word2vec/

[4]https://nlp.stanford.edu/software/lex-parser.shtml

[5]https://nlp.stanford.edu/software/stanford-dependencies.shtml

Hyperparameters	
Word embedding size	$d = 100$
Score function discount	$\mu = 0.8$
Margin loss discount	$\kappa = 0.2$
Regularization	$\lambda = 10^{-4}$
Dropout fraction	0.5
Hidden unit number	$n_{hu} = 100$
Hidden unit number	$n_c = 100$
Interaction feature length	$n_T = 150$

Table 2: Hyperparameters of our model.

Method	Trigger id+cl	Argument id	Argument id+cl
JointBeam	64.2	38.0	35.0
RBPB	67.8	55.4	43.8
JointEventEntity	68.7	50.6	48.4
dbRNN	**69.6**	**57.2**	**50.1**

Table 4: Overall performance with predicted entities.

Overall Performance

We compare our performance with the following stare-of-the-art methods:

1. **Cross-Event** is proposed by Liao and Grishman (2010), which uses document level information to improve the performance of ACE event extraction;
2. **Cross-Entity** is proposed by Hong et al. (2011), which uses cross-entity inference for event extraction;
3. **JointBeam** is the method proposed by Li, Ji, and Huang (2013), which extracts events based on structure prediction. They manually designed several discrete features about arguments, which is certainly far from enough;
4. **DMCNN** is proposed by Chen et al. (2015), which is the first approach based on deep neural network.
5. **JointEventEntity** is proposed by (Yang and Mitchell 2016), which jointly extracting events and entities.
6. **RBPB** is proposed by Sha et al. (2016), which considers two kinds of argument relationships (positive and negative). Different from us, they only deal with two argument relationships while our tensor layer can deal with countless number of argument relationships.
7. **JRNN** is proposed by Nguyen, Cho, and Grishman (2016), which uses a bidirectional RNN and manually designed features to extract events. Difference from us, they only take dependency relations as input features instead of model them directly into architecture. In addition, they use the features of JointBeam for arguments.

Table 3 shows the overall performance with gold-standard entities (in ACE2005 dataset). Table 4 shows the performance with predicted candidate arguments (take NPs as candidate arguments). Notice that trigger / argument identification is integrated into trigger / argument classification, that is to say, if a candidate trigger / argument is assigned the

Method	Trigger Identification +Classification (%)			Argument Identification (%)			Argument Role (%)		
	P	R	F_1	P	R	F_1	P	R	F_1
Cross-Event	68.7	68.9	68.8	50.9	49.7	50.3	45.1	44.1	44.6
Cross-Entity	72.9	64.3	68.3	53.4	52.9	53.1	51.6	45.5	48.3
JointBeam	73.7	62.3	67.5	69.8	47.9	56.8	64.7	44.4	52.7
DMCNN	**75.6**	63.6	69.1	68.8	51.9	59.1	62.2	46.9	53.5
RBPB	70.3	67.5	68.9	63.2	59.4	61.2	54.1	53.5	53.8
JRNN	66.0	**73.0**	69.3	61.4	64.2	62.8	54.2	**56.7**	55.4
dbRNN	74.1	69.8	**71.9**	**71.3**	**64.5**	**67.7**	**66.2**	52.8	**58.7**

Table 3: Overall performance with gold-standard entities.

Method	Trigger id+cl	Argument id	Argument id+cl
Our model without DB	69.0	62.7	54.6
+ binary DB	71.2	63.9	56.8
+ typed DB (full)	71.9	64.4	57.2

Table 5: Comparison after adding dependency bridges (DB). The numbers are F_1 scores. We compare with two baselines: no dependency bridges considered and only binary dependency bridges.

"NoEvent" / "NoRole" label during classification, then it is not identified.

From the results in Table 3, we can see that the dbRNN model we proposed with automatically learned features achieves the best performance among all of the competing methods. dbRNN outperforms the feature-based model (RBPB) by 2.0% in trigger classification and 4.9% by argument classification, which is significant (Wilcoxon signed-rank test, $p < 0.05$). Our model can also outperform the deep learning method (JRNN) by 1.6% in trigger classification and 3.3% by argument classification ($p < 0.05$). This demonstrates the effectiveness of the proposed method. In Table 4, our result is also 0.5% and 1.2% higher than the JointEventEntity method in trigger classification and argument classification ($p < 0.05$). We found that dbRNN significantly outperforms previous methods in argument identification / classification. This demonstrate that our model generally achieves higher performance than previous work including human engineered features as well as existing NN models.

Effect of Dependency Bridges

The effect of dependency bridges is shown in Table 5. "DB(binary)" means all weights of the dependency relations are set to 1. We can see that after adding binary dependency bridge, the F_1 value of all the three evaluation metrics (trigger classification, argument identification, argument role classification) are significantly improved (Wilcoxon signed-rank test, $p < 0.05$). After we set all weights of the dependency relations trainable, in the "+ DB" row, the performances gain a further improvement, which illustrated the effectiveness of dependency bridges. The visualization of trained weights is shown in Figure 4, we can see

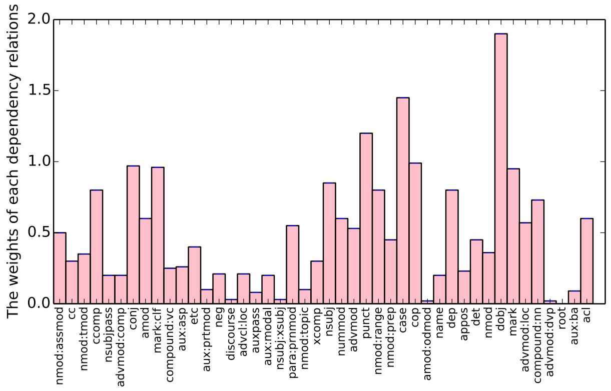


Figure 4: The visualization of trained weights of each dependency relations.

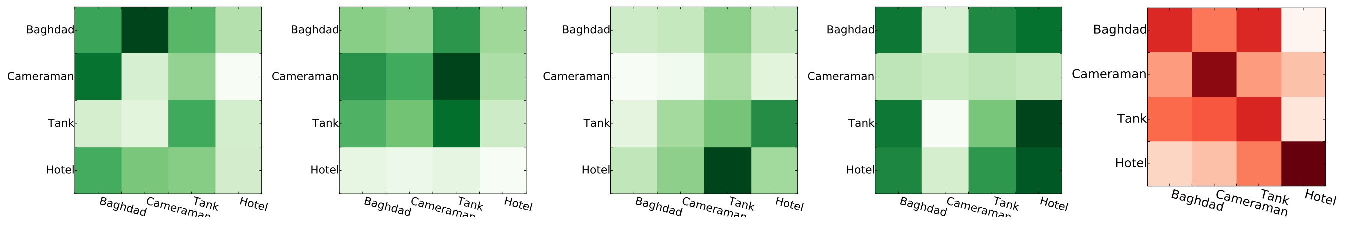
	Method	$1/1$	$1/N$	All
Argument Identification	dbRNN-SMA	59.5	67.0	64.1
	dbRNN-MP	59.7	64.8	62.0
	dbRNN-TL	59.6	55.8	58.2
	dbRNN	59.9	69.5	67.7
Argument Role Classification	dbRNN-SMA	54.6	56.5	56.0
	dbRNN-MP	54.7	55.8	55.2
	dbRNN-TL	54.9	52.3	53.1
	dbRNN	54.6	60.9	58.7

Table 6: Comparison between dbRNN and dbRNN without tensor layer. (The latter is denoted as "dbRNN-TL".) Also, "dbRNN-SMA" means to only cast SMA away from the whole model and "dbRNN-MP" means cast the max-pooling feature matrix away. Here, we report the argument performance since the tensor layer is only applied to argument extraction.

that dependency relationship type `dobj` receives the highest weight after training. According to grammar knowledge, `dobj` should be an informative relationship for event extraction task, and our model considers `dobj` as the most influential dependency type automatically.

Effect of Tensor Layer

Table 6 illustrates that the method based on tensor transformation layer (dbRNN) outperforms the method without tensor layer (dbRNN-TL). It shows that the interaction features extracted by tensor layer is extremely useful for argumen-



(a) Die1 (b) Die2 (c) Fire1 (d) Fire2 (e) SMA (die)

Figure 5: Visualization of several slices of the interaction feature in the tensor layer. The processed sentence is as Figure 1. Trigger of (a) and (b) is "die". Trigger of (c) and (d) is "fire". Darker green means stronger interaction. (e) is the visualization of SMA matrix when the trigger is "die". Easy to show, the candidate arguments `Baghdad` and `tank` tend to relate to each other.

t extraction. Specifically, dbRNN yields a 13.7% improvement for argument identification and 8.6% improvement for argument classification on the sentences with multiple entities ($1/N$) (Wilcoxon signed-rank test, $p < 0.05$). This improvement is much larger than in sentences with a single entity ($1/1$). Similar trend can also be observed in the two subparts of tensor later (SMA and MP). This demonstrates that the proposed dbRNN can effectively capture more valuable clues than the dbRNN-TL, especially when a sentence contains more than one entities.

Analysis of "Interaction Features"

We use the sentence in Figure 1 as an example to illustrate the captured interaction features. Since there are two triggers (`die` and `fire`) in the sentence, we select two slices which are relatively easier to be explained out of tensor T's n_T slices (T is the tensor-shaped interaction feature in Figure 3) based on the two triggers. Figures 5a and 5c are from the same position of the n_T slices in F, so does Figures 5b and Figure 5d.

We can see that Figures 5(a) and 5(c) capture whether two candidate arguments have the same dependency parent. When the trigger is `die`, we found that `Baghdad` and `cameraman` have stronger interactions. When the trigger is `fire`, we found that `tank` and `hotel` have stronger interactions. This is identical to the dependency structure in Figure 1. Likewise, Figures 5(b) and 5(d) capture whether two candidate arguments are semantically coherent. When the trigger is `die`, we found that `Baghdad`, `cameraman` and `tank` are semantically relevant and human tend to identify them as arguments, so the interaction between any of them are stronger. Although there are also some slices of T that cannot be directly explained, our tensor layer still have captured many useful interaction features according to the interaction feature visualization.

We also visualize the SMA matrix in Figure 5(e). It corresponds to the same example in Figure 1 when the trigger is `die`. We can see that `Baghdad` and `tank` tend to occur together. This fact illustrates that the SMA matrix successfully captures the attention between different candidate arguments.

Related Work

Event extraction is important in the knowledge mining field. Previous work can be classified to two kinds of methods: (1) traditional approaches using a set of elaborately designed features that are extracted by textual analysis and linguistic knowledge (2) deep neural network based approaches.

In traditional approaches, some focus on lexical features (Grishman, Westbrook, and Meyers 2005; Huang and Riloff 2012; Li, Bontcheva, and Cunningham 2005; Liu and Strzalkowski 2012), while others considered broader context when deciding the role fillers (Gu and Cercone 2006; Patwardhan and Riloff 2009). Also, there are systems taking the whole discourse features into consideration (Liao and Grishman 2010; Hong et al. 2011; Huang and Riloff 2011). Ji and Grishman (2008) even consider the topic-related documents, so they propose the cross-document method. Liao and Grishman (2010), Hong et al. (2011), Li, Ji, and Huang (2013), Lu and Roth (2012) and Yang and Mitchell (2016) use a series of global features (for example, the occurrence of one event type lead to the occurrence of another) to improve the role assignment and event classification performance. Sha et al. (2016) consider two of the argument relationships: positive and negative. Huang et al. (2016) use AMR as symbolic features and simultaneously extract events and event schemas.

Dynamic multi-pooling convolutional neural network (DMCNN) (Chen et al. 2015) is the first deep neural network-based approach. (Nguyen, Cho, and Grishman 2016) enhance bidirectional RNN with manually designed features to extract events. Based on these above methods, we use dependency relations as a new structure and learn argument-argument interaction feature.

Conclusion

This paper proposes a novel deep neural network architecture, called dependency-bridge recurrent neural network (dbRNN) for the task of event extraction. dbRNN introduces dependency bridges to RNN so that syntactical information can be explicitly considered in the model. Then a tensor layer is applied to capture various of argument-argument interactions when extracting events, which significantly improve the performance of argument identification and classification. In addition, we jointly extract event triggers and arguments by a max-margin criterion so that the two subtasks can benefit each other. The experiment results show the effectiveness of our proposed method.

References

Chen, Y.; Xu, L.; Liu, K.; Zeng, D.; and Zhao, J. 2015. Event extraction via dynamic multi-pooling convolutional neural networks. In *ACL*, 167–176.

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. 2011. Natural language processing (almost) from scratch. *JMLR* 12:2461–2505.

Glavaš, G., and Šnajder, J. 2014. Event graphs for information retrieval and multi-document summarization. *Expert systems with applications* 41(15):6904–6916.

Grishman, R.; Westbrook, D.; and Meyers, A. 2005. Nyu's english ACE 2005 system description. *ACE* 5.

Gu, Z., and Cercone, N. 2006. Segment-based hidden markov models for information extraction. In *ACL*, 481–488.

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

Hong, Y.; Zhang, J.; Ma, B.; Yao, J.; Zhou, G.; and Zhu, Q. 2011. Using cross-entity inference to improve event extraction. In *ACL*, 1127–1136.

Huang, R., and Riloff, E. 2011. Peeling back the layers: detecting event role fillers in secondary contexts. In *ACL*, 1137–1147.

Huang, R., and Riloff, E. 2012. Bootstrapped training of event extraction classifiers. In *EACL*, 286–295.

Huang, L.; Cassidy, T.; Feng, X.; Ji, H.; Voss, R. C.; Han, J.; and Sil, A. 2016. Liberal event extraction and event schema induction. In *ACL*, 258–268.

Ji, H., and Grishman, R. 2008. Refining event extraction through cross-document inference. In *ACL*, 254–262.

Li, Y.; Bontcheva, K.; and Cunningham, H. 2005. Using uneven margins svm and perceptron for information extraction. In *CoNLL*, 72–79.

Li, Q.; Ji, H.; and Huang, L. 2013. Joint event extraction via structured prediction with global features. In *ACL*, 73–82.

Liao, S., and Grishman, R. 2010. Using document level cross-event inference to improve event extraction. In *ACL*, 789–797.

Liu, T., and Strzalkowski, T. 2012. Bootstrapping events and relations from text. In *EACL*, 296–305.

Lu, W., and Roth, D. 2012. Automatic event extraction with structured preference modeling. In *ACL*, 835–844.

Mou, L.; Men, R.; Li, G.; Xu, Y.; Zhang, L.; Yan, R.; and Jin, Z. 2015a. Natural language inference by tree-based convolution and heuristic matching. *arXiv preprint arXiv:1512.08422*.

Mou, L.; Peng, H.; Li, G.; Xu, Y.; Zhang, L.; and Jin, Z. 2015b. Discriminative neural sentence modeling by tree-based convolution. *arXiv preprint arXiv:1504.01106*.

Mou, L.; Li, G.; Zhang, L.; Wang, T.; and Jin, Z. 2016. Convolutional neural networks over tree structures for programming language processing. In *AAAI*, 1287–1293.

Nguyen, H. T.; Cho, K.; and Grishman, R. 2016. Joint event extraction via recurrent neural networks. In *NAACL*, 300–309.

Patwardhan, S., and Riloff, E. 2009. A unified model of phrasal and sentential evidence for information extraction. In *EMNLP*, 151–160.

Ratliff, N. D.; Bagnell, J. A.; and Zinkevich, M. 2007. (online) subgradient methods for structured prediction. In *AISTATS*, 380–387.

Schuster, M., and Paliwal, K. K. 1997. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on* 45(11):2673–2681.

Sha, L.; Liu, J.; Lin, C.-Y.; Li, S.; Chang, B.; and Sui, Z. 2016. Rbpb: Regularization-based pattern balancing method for event extraction. In *ACL*, 1224–1234.

Socher, R.; Huang, E. H.; Pennington, J.; Ng, A. Y.; and Manning, C. D. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *NIPS*.

Socher, R.; Perelygin, A.; Wu, J. Y.; Chuang, J.; Manning, C. D.; Ng, A. Y.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. *ACL*.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

Taskar, B.; Chatalbashev, V.; Koller, D.; and Guestrin, C. 2005. Learning structured prediction models: A large margin approach. In *ICML*, 896–903. ACM.

Yang, B., and Mitchell, M. T. 2016. Joint extraction of events and entities within a document context. In *NAACL*, 289–299. Association for Computational Linguistics.

Yang, H.; Chua, T.-S.; Wang, S.; and Koh, C.-K. 2003. Structured use of external knowledge for event-based open domain question answering. In *SIGIR*, 33–40.

Zeiler, M. D. 2012. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.