Joint Event Extraction via Recurrent Neural Networks

Thien Huu Nguyen, Kyunghyun Cho and Ralph Grishman
Computer Science Department, New York University, New York, NY 10003, USA
thien@cs.nyu.edu,kyunghyun.cho@nyu.edu,grishman@cs.nyu.edu

Abstract

Event extraction is a particularly challenging problem in information extraction. The state-of-the-art models for this problem have either applied convolutional neural networks in a pipelined framework (Chen et al., 2015) or followed the joint architecture via structured prediction with rich local and global features (Li et al., 2013). The former is able to learn hidden feature representations automatically from data based on the continuous and generalized representations of words. The latter, on the other hand, is capable of mitigating the error propagation problem of the pipelined approach and exploiting the inter-dependencies between event triggers and argument roles via discrete structures. In this work, we propose to do event extraction in a joint framework with bidirectional recurrent neural networks, thereby benefiting from the advantages of the two models as well as addressing issues inherent in the existing approaches. We systematically investigate different memory features for the joint model and demonstrate that the proposed model achieves the state-of-the-art performance on the ACE 2005 dataset.

1 Introduction

We address the problem of event extraction (EE): identifying event triggers of specified types and their arguments in text. Triggers are often single verbs or normalizations that evoke some events of interest while arguments are the entities participating into such events. This is an important and challenging task of information extraction in natural language processing (NLP), as the same event might be present in various expressions, and an expression might expresses different events in different contexts.

There are two main approaches to EE: (i) the joint approach that predicts event triggers and arguments for sentences *simultaneously* as a structured prediction problem, and (ii) the pipelined approach that first performs trigger prediction and then identifies arguments in *separate* stages.

The most successful joint system for EE (Li et al., 2013) is based on the structured perceptron algorithm with a large set of local and global features[1]. These features are designed to capture the *discrete* structures that are intuitively helpful for EE using the NLP toolkits (e.g., part of speech tags, dependency and constituent tags). The advantages of such a joint system are twofold: (i) mitigating the error propagation from the upstream component (trigger identification) to the downstream classifier (argument identification), and (ii) benefiting from the the inter-dependencies among event triggers and argument roles via global features. For example, consider the following sentence (taken from Li et al. (2013)) in the ACE 2005 dataset:

In Baghdad, a cameraman **died** *when an American tank* **fired** *on the Palestine hotel.*

In this sentence, **died** and **fired** are the event triggers for the events of types *Die* and *Attack*, respectively. In the pipelined approach, it is often simple for the argument classifiers to realize that *camera-*

[1]Local features encapsulate the characteristics for the individual tasks (i.e, trigger and argument role labeling) while global features target the dependencies between triggers and arguments and are only available in the joint approach.

Proceedings of NAACL-HLT 2016, pages 300–309,
San Diego, California, June 12-17, 2016.

man is the *Target* argument of the *Die* event due to the proximity between *cameraman* and *died* in the sentence. However, as *cameraman* is far away from *fired*, the argument classifiers in the pipelined approach might fail to recognize *cameraman* as the *Target* argument for the event *Attack* with their local features. The joint approach can overcome this issue by relying on the global features to encode the fact that a *Victim* argument for the *Die* event is often the *Target* argument for the *Attack* event in the same sentence.

Despite the advantages presented above, the joint system by Li et al. (2013) suffers from the lack of generalization over the unseen words/features and the inability to extract the underlying structures for EE (due to its *discrete* representation from the handcrafted feature set) (Nguyen and Grishman, 2015b; Chen et al., 2015).

The most successful pipelined system for EE to date (Chen et al., 2015) addresses these drawbacks of the joint system by Li et al. (2013) via dynamic multi-pooling convolutional neural networks (DMCNN). In this system, words are represented by the *continuous* representations (Bengio et al., 2003; Turian et al., 2010; Mikolov et al., 2013a) and features are automatically learnt from data by the DMCNN, thereby alleviating the unseen word/feature problem and extracting more effective features for the given dataset. However, as the system by Chen et al. (2015) is pipelined, it still suffers from the inherent limitations of error propagation and failure to exploit the inter-dependencies between event triggers and argument roles (Li et al., 2013). Finally, we notice that the discrete features, shown to be helpful in the previous studies for EE (Li et al., 2013), are not considered in Chen et al. (2015).

Guided by these characteristics of the EE systems by Li et al. (2013) and Chen et al. (2015), in this work, we propose to solve the EE problem with the joint approach via recurrent neural networks (RNNs) (Hochreiter and Schmidhuber, 1997; Cho et al., 2014) augmented with the discrete features, thus inheriting all the benefits from both systems as well as overcoming their inherent issues. To the best of our knowledge, this is the first work to employ neural networks to do joint EE.

Our model involves two RNNs that run over the sentences in both forward and reverse directions to learn a richer representation for the sentences. This representation is then utilized to predict event triggers and argument roles jointly. In order to capture the inter-dependencies between triggers and argument roles, we introduce memory vectors/matrices to store the prediction information during the course of labeling the sentences.

We systematically explore various memory vector/matrices as well as different methods to learn word representations for the joint model. The experimental results show that our system achieves the state-of-the-art performance on the widely used ACE 2005 dataset.

2 Event Extraction Task

We focus on the EE task of the Automatic Context Extraction (ACE) evaluation[2]. ACE defines an event as something that happens or leads to some change of state. We employ the following terminology:

- **Event mention**: a phrase or sentence in which an event occurs, including one trigger and an arbitrary number of arguments.
- **Event trigger**: the main word that most clearly expresses an event occurrence.
- **Event argument**: an entity mention, temporal expression or value (e.g. *Job-Title*) that servers as a participant or attribute with a specific role in an event mention.

ACE annotates 8 types and 33 subtypes (e.g., *Attack*, *Die*, *Start-Position*) for event mentions that also correspond to the types and subtypes of the event triggers. Each event subtype has its own set of roles to be filled by the event arguments. For instance, the roles for the *Die* event include *Place*, *Victim* and *Time*. The total number of roles for all the event subtypes is 36.

Given an English text document, an event extraction system needs to recognize event triggers with specific subtypes and their corresponding arguments with the roles for each sentence. Following the previous work (Liao and Grishman, 2011; Li et al., 2013; Chen et al., 2015), we assume that the argument candidates (i.e, the entity mentions, temporal expressions and values) are provided (by the ACE annotation) to the event extraction systems.

[2] http://projects.ldc.upenn.edu/ace

3 Model

We formalize the EE task as follow. Let $W = w_1 w_2 \dots w_n$ be a sentence where n is the sentence length and w_i is the i-th token. Also, let $E = e_1, e_2, \dots, e_k$ be the entity mentions[3] in this sentence (k is the number of the entity mentions and can be zero). Each entity mention comes with the offsets of the head and the entity type. We further assume that $i_1, i_2, \dots, i_k$ be the indexes of the last words of the mention heads for $e_1, e_2, \dots, e_k$, respectively.

In EE, for every token w_i in the sentence, we need to predict the event subtype (if any) for it. If w_i is a trigger word for some event of interest, we then need to predict the roles (if any) that each entity mention e_j plays in such event.

The joint model for event extraction in this work consists of two phases: (i) the *encoding phase* that applies recurrent neural networks to induce a more abstract representation of the sentence, and (ii) the *prediction phase* that uses the new representation to perform event trigger and argument role identification simultaneously for W. Figure 1 shows an overview of the model.

3.1 Encoding

3.1.1 Sentence Encoding

In the encoding phase, we first transform each token w_i into a real-valued vector x_i using the concatenation of the following three vectors:

1. The word embedding vector of w_i: This is obtained by looking up a pre-trained word embedding table D (Collobert and Weston, 2008; Turian et al., 2010; Mikolov et al., 2013a).

2. The real-valued embedding vector for the entity type of w_i: This vector is motivated from the prior work (Nguyen and Grishman, 2015b) and generated by looking up the entity type embedding table (initialized randomly) for the entity type of w_i. Note that we also employ the BIO annotation schema to assign entity type labels to each token in the sentences using the heads of the entity mentions as do Nguyen and Grishman (2015b).

3. The binary vector whose dimensions correspond to the possible relations between words in the dependency trees. The value at each dimension of this vector is set to 1 only if there exists one edge of the corresponding relation connected to w_i in the dependency tree of W. This vector represents the dependency features that are shown to be helpful in the previous research (Li et al., 2013).

Note that we do not use the relative position features, unlike the prior work on neural networks for EE (Nguyen and Grishman, 2015b; Chen et al., 2015). The reason is we predict the whole sentences for triggers and argument roles jointly, thus having no fixed positions for anchoring in the sentences.

The transformation from the token w_i to the vector x_i essentially converts the input sentence W into a sequence of real-valued vectors $X = (x_1, x_2, \dots, x_n)$, to be used by recurrent neural networks to learn a more effective representation.

3.1.2 Recurrent Neural Networks

Consider the input sequence $X = (x_1, x_2, \dots, x_n)$. At each step i, we compute the hidden vector α_i based on the current input vector x_i and the previous hidden vector α_{i-1}, using the non-linear transformation function Φ: $\alpha_i = \Phi(x_i, \alpha_{i-1})$. This recurrent computation is done over X to generate the hidden vector sequence $(\alpha_1, \alpha_2, \dots, \alpha_n)$, denoted by $\overrightarrow{\text{RNN}}(x_1, x_2, \dots, x_n) = (\alpha_1, \alpha_2, \dots, \alpha_n)$.

An important characteristics of the recurrent mechanism is that it adaptively accumulates the context information from position 1 to i into the hidden vector α_i, making α_i a rich representation. However, α_i is not sufficient for the event trigger and argument predictions at position i as such predictions might need to rely on the context information in the future (i.e, from position i to n). In order to address this issue, we run a second RNN in the reverse direction from X_n to X_1 to generate the second hidden vector sequence: $\overleftarrow{\text{RNN}}(x_n, x_{n-1}, \dots, x_1) = (\alpha'_n, \alpha'_{n-1}, \dots, \alpha'_1)$ in which α'_i summarizes the context information from position n to i. Eventually, we obtain the new representation $(h_1, h_2, \dots, h_n)$ for X by concatenating the hidden vectors in $(\alpha_1, \alpha_2, \dots, \alpha_n)$ and $(\alpha'_n, \alpha'_{n-1}, \dots, \alpha'_1)$: $h_i = [\alpha_i, \alpha'_i]$. Note that h_i essentially encapsulates the context information over the whole sentence (from 1 to n) with a greater focus on position i.

Regarding the non-linear function, the simplest

[3] From now on, when mentioning entity mentions, we always refer to the ACE entity mentions, times and values.

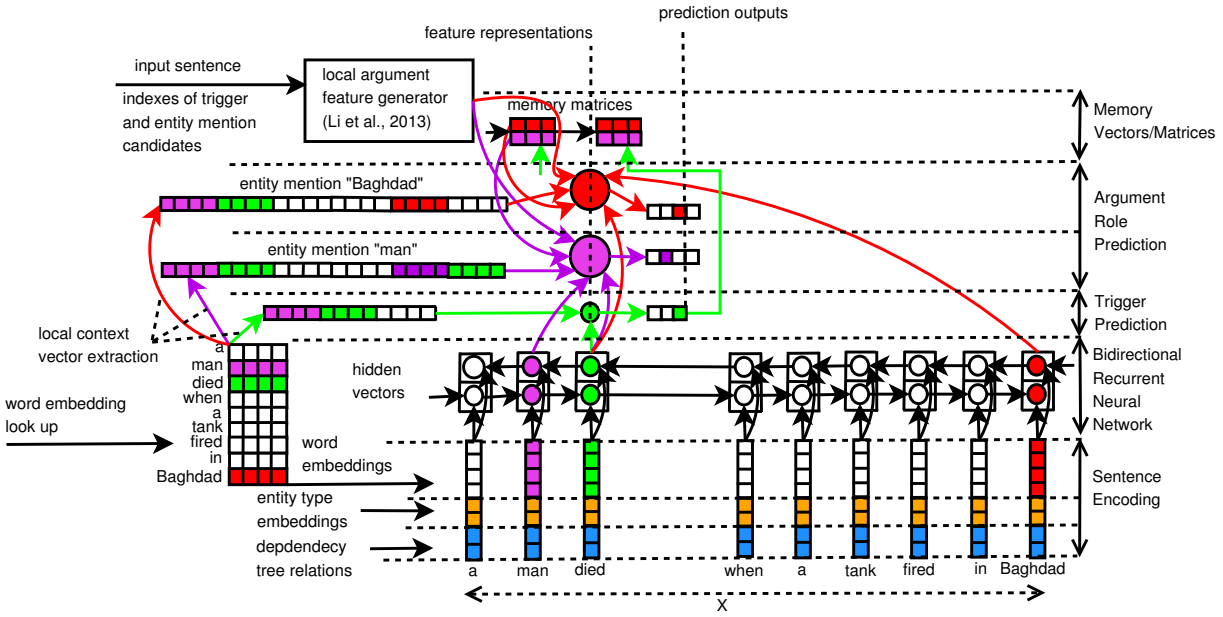

Figure 1: The joint EE model for the input sentence "*a man died when a tank fired in Baghdad*" with local context window $d = 1$. We only demonstrate the memory matrices $G_i^{\text{arg/trg}}$ in this figure. Green corresponds to the trigger candidate "*died*" at the current step while violet and red are for the entity mentions "*man*" and "*Baghdad*" respectively.

form of Φ in the literature considers it as a one-layer feed-forward neural network. Unfortunately, this function is prone to the "*vanishing gradient*" problem (Bengio et al., 1994), making it challenging to train RNNs properly. This problem can be alleviated by long-short term memory units (LSTM) (Hochreiter and Schmidhuber, 1997; Gers, 2001). In this work, we use a variant of LSTM; called the *Gated Recurrent Units* (GRU) from Cho et al. (2014). GRU has been shown to achieve comparable performance (Chung et al., 2014; Józefowicz et al., 2015).

3.2 Prediction

In order to jointly predict triggers and argument roles for W, we maintain a binary memory vector G_i^{trg} for triggers, and binary memory matrices G_i^{arg} and $G_i^{\text{arg/trg}}$ for arguments (at each time i). These vector/matrices are set to zeros initially ($i = 0$) and updated during the prediction process for W.

Given the bidirectional representation $h_1, h_2, \ldots, h_n$ in the encoding phase and the initialized memory vector/matrices, the joint prediction procedure loops over n tokens in the sentence (from 1 to n). At each time step i, we perform the following three stages in order:

(i) *trigger prediction* for w_i.

(ii) *argument role prediction* for all the entity mentions $e_1, e_2, \ldots, e_k$ with respect to the current token w_i.

(iii) *compute* G_i^{trg}, G_i^{arg} and $G_i^{\text{arg/trg}}$ for the current step using the previous memory vector/matrices G_{i-1}^{trg}, G_{i-1}^{arg} and $G_{i-1}^{\text{arg/trg}}$, and the prediction output in the earlier stages.

The output of this process would be the predicted trigger subtype t_i for w_i, the predicted argument roles $a_{i1}, a_{i2}, \ldots, a_{ik}$ and the memory vector/matrices G_i^{trg}, G_i^{arg} and $G_i^{\text{arg/trg}}$ for the current step. Note that t_i should be the event subtype if w_i is a trigger word for some event of interest, or "*Other*" in the other cases. a_{ij}, in constrast, should be the argument role of the entity mention e_j with respect to w_i if w_i is a trigger word and e_j is an argument of the corresponding event, otherwise a_{ij} is set to "*Other*" ($j = 1$ to k).

3.2.1 Trigger Prediction

In the trigger prediction stage for the current token w_i, we first compute the feature representation vector R_i^{trg} for w_i using the concatenation of the following three vectors:

- h_i: the hidden vector to encapsulate the global context of the input sentence.

- L_i^{trg}: the local context vector for w_i. L_i^{trg} is generated by concatenating the vectors of the words in a context window d of w_i: $L_i^{\text{trg}} = [D[w_{i-d}], \ldots, D[w_i], \ldots, D[w_{i+d}]]$.
- G_{i-1}^{trg}: the memory vector from the previous step.

The representation vector $R_i^{\text{trg}} = [h_i, L_i^{\text{trg}}, G_{i-1}^{\text{trg}}]$ is then fed into a feed-forward neural network F^{trg} with a softmax layer in the end to compute the probability distribution $P_{i;t}^{\text{trg}}$ over the possible trigger subtypes: $P_{i;t}^{\text{trg}} = P_i^{\text{trg}}(l = t) = F_t^{\text{trg}}(R_i^{\text{trg}})$ where t is a trigger subtype. Finally, we compute the predicted type t_i for w_i by: $t_i = \text{argmax}_t(P_{i;t}^{\text{trg}})$.

3.2.2 Argument Role Prediction

In the argument role prediction stage, we first check if the predicted trigger subtype t_i in the previous stage is "*Other*" or not. If yes, we can simply set a_{ij} to "*Other*" for all $j = 1$ to k and go to the next stage immediately. Otherwise, we loop over the entity mentions $e_1, e_2, \ldots, e_k$. For each entity mention e_j with the head index of i_j, we predict the argument role a_{ij} with respect to the trigger word w_i using the following procedure.

First, we generate the feature representation vector R_{ij}^{arg} for e_j and w_i by concatenating the following vectors:

- h_i and h_{i_j}: the hidden vectors to capture the global context of the input sentence for w_i and e_j, respectively.
- L_{ij}^{arg}: the local context vector for w_i and e_j. L_{ij}^{arg} is the concatenation of the vectors of the words in the context windows of size d for w_i and w_{i_j}: $L_{ij}^{\text{arg}} = [D[w_{i-d}], \ldots, D[w_i], \ldots, D[w_{i+d}], D[w_{i_j-d}], \ldots, D[w_{i_j}], \ldots, D[w_{i_j+d}]]$.
- B_{ij}: the hidden vector for the binary feature vector V_{ij}. V_{ij} is based on the local argument features between the tokens i and i_j from (Li et al., 2013). B_{ij} is then computed by feeding V_{ij} into a feed-forward neural network F^{binary} for further abstraction: $B_{ij} = F^{\text{binary}}(V_{ij})$.
- $G_{i-1}^{\text{arg}}[j]$ and $G_{i-1}^{\text{arg/trg}}[j]$: the memory vectors for e_j that are extracted out of the memory matrices G_{i-1}^{arg} and $G_{i-1}^{\text{arg/trg}}$ from the previous step.

In the next step, we again use a feed-forward neural network F^{arg} with a softmax layer in the end to transform $R_{ij}^{\text{arg}} = [h_i, h_{i_j}, L_{ij}^{\text{arg}}, B_{ij}, G_{i-1}^{\text{arg}}[j], G_{i-1}^{\text{arg/trg}}[j]]$ into the probability distribution $P_{ij;a}^{\text{trg}}$ over the possible argument roles: $P_{ij;a}^{\text{arg}} = P_{ij}^{\text{arg}}(l = a) = F_a^{\text{arg}}(R_{ij}^{\text{arg}})$ where a is an argument role. Eventually, the predicted argument role for w_i and e_j is $a_{ij} = \text{argmax}_a(P_{ij;a}^{\text{arg}})$.

Note that the binary vector V_{ij} enriches the feature representation R_{ij}^{arg} for argument labeling with the discrete structures discovered in the prior work on feature analysis for EE (Li et al., 2013). These features include the shortest dependency paths, the entity types, subtypes, etc.

3.2.3 The Memory Vector/Matrices

An important characteristics of EE is the existence of the dependencies between trigger labels and argument roles within the same sentences (Li et al., 2013). In this work, we encode these dependencies into the memory vectors/matrices G_i^{trg}, G_i^{arg} and $G_i^{\text{arg/trg}}$ ($i = 0$ to n) and use them as features in the trigger and argument prediction explicitly (as shown in the representation vectors R_i^{trg} and R_{ij}^{arg} above). We classify the dependencies into the following three categories:

1. The **dependencies among trigger subtypes**: are captured by the memory vectors G_i^{trg} ($G_i^{\text{trg}} \in \{0,1\}^{n_T}$ for $i = 0, \ldots, n$, and n_T is the number of the possible trigger subtypes). At time i, G_i^{trg} indicates which event subtypes have been recognized before i. We obtain G_i^{trg} from G_{i-1}^{trg} and the trigger prediction output t_i at time i: $G_i^{\text{trg}}[t] = 1$ if $t = t_i$ and $G_{i-1}^{\text{trg}}[t]$ otherwise.

A motivation for such dependencies is that if a *Die* event appears somewhere in the sentences, the possibility for the later occurrence of an *Attack* event would be likely.

2. The **dependencies among argument roles**: are encoded by the memory matrix G_i^{arg} ($G_i^{\text{arg}} \in \{0,1\}^{k \times n_A}$ for $i = 0, \ldots, n$, and n_A is the number of the possible argument roles). At time i, G_i^{arg} summarizes the argument roles that the entity mentions has played with some event in the past. In particular, $G_i^{\text{arg}}[j][a] = 1$ if and only if e_j has the role of a with some event before time i. G_i^{arg} is computed from G_{i-1}^{arg}, and the prediction outputs t_i and $a_{i1}, \ldots, a_{ik}$ at time i: $G_i^{\text{arg}}[j][a] = 1$ if $t_i \neq$ "*Other*" and $a = a_{ij}$, and $G_{i-1}^{\text{arg}}[j][a]$ otherwise (for $j = 1$ to k).

3. The **dependencies between argument roles and trigger subtypes**: are encoded by the memory matrix $G_i^{\text{arg/trg}}$ ($G_i^{\text{arg/trg}} \in \{0,1\}^{k \times n_T}$ for $i = 0$ to n). At time i, $G_i^{\text{arg/trg}}$ specifies which entity mentions have been identified as arguments for which event subtypes before. In particular, $G_i^{\text{arg/trg}}[j][t] = 1$ if and only if e_j has been detected as an argument for some event of subtype t before i. $G_i^{\text{arg/trg}}$ is computed from $G_{i-1}^{\text{arg/trg}}$ and the trigger prediction output t_i at time i: $G_i^{\text{arg/trg}}[j][t] = 1$ if $t_i \neq$ "*Other*" and $t = t_i$, and $G_{i-1}^{\text{arg/trg}}[j][t]$ otherwise (for all $j = 1$ to k).

3.3 Training

Denote the given trigger subtypes and argument roles for W in the training time as $T = t_1^*, t_2^*, \ldots, t_n^*$ and $A = (a_{ij}^*)_{i=1,n}^{j=1,k}$. We train the network by minimizing the joint negative log-likelihood function C for triggers and argument roles:

$$\begin{aligned} C(T, A, X, E) &= -\log P(T, A|X, E) \\ &= -\log P(T|X, E) - \log P(A|T, X, E) \\ &= -\sum_{i=1}^{n} \log P_{i;t_i^*}^{\text{trg}} \\ &\quad -\sum_{i=1}^{n} I(t_i \neq \text{"}\textit{Other}\text{"}) \sum_{j=1}^{k} \log P_{ij;a_{ij}^*}^{\text{arg}} \end{aligned}$$

where I is the indicator function.

We apply the stochastic gradient descent algorithm with mini-batches and the AdaDelta update rule (Zeiler, 2012). The gradients are computed using back-propagation. During training, besides the weight matrices, we also optimize the word and entity type embedding tables to achieve the optimal states. Finally, we rescale the weights whose Frobenius norms exceed a hyperparameter (Kim, 2014; Nguyen and Grishman, 2015a).

4 Word Representation

Following the prior work (Nguyen and Grishman, 2015b; Chen et al., 2015), we pre-train word embeddings from a large corpus and employ them to initialize the word embedding table. One of the models to train word embeddings have been proposed in Mikolov et al. (2013a; 2013b) that introduce two log-linear models, i.e the continuous bag-of-words model (CBOW) and the continuous skip-gram model (SKIP-GRAM). The CBOW model attempts to predict the current word based on *the average of the context word vectors* while the SKIP-GRAM model aims to predict the surrounding words in a sentence given the current word. In this work, besides the CBOW and SKIP-GRAM models, we examine a concatenation-based variant of CBOW (C-CBOW) to train word embeddings and compare the three models to understand their effectiveness for EE. The objective of C-CBOW is to predict the target word using *the concatenation of the vectors of the words surrounding it*.

5 Experiments

5.1 Resources, Parameters and Dataset

For all the experiments below, in the encoding phase, we use 50 dimensions for the entity type embeddings, 300 dimensions for the word embeddings and 300 units in the hidden layers for the RNNs.

Regarding the prediction phase, we employ the context window of 2 for the local features, and the feed-forward neural networks with one hidden layer for F^{trg}, F^{arg} and F^{binary} (the size of the hidden layers are 600, 600 and 300 respectively).

Finally, for training, we use the mini-batch size = 50 and the parameter for the Frobenius norms = 3.

These parameter values are either inherited from the prior research (Nguyen and Grishman, 2015b; Chen et al., 2015) or selected according to the validation data.

We pre-train the word embeddings from the English Gigaword corpus utilizing the word2vec toolkit[4] (modified to add the C-CBOW model). Following Baroni et al. (2014), we employ the context window of 5, the subsampling of the frequent words set to $1e$-05 and 10 negative samples.

We evaluate the model with the ACE 2005 corpus. For the purpose of comparison, we use the same data split as the previous work (Ji and Grishman, 2008; Liao and Grishman, 2010; Li et al., 2013; Nguyen and Grishman, 2015b; Chen et al., 2015). This data split includes 40 newswire articles (672 sentences) for the test set, 30 other documents (836 sentences) for the development set and 529 remaining documents (14,849 sentences) for the training set. Also,

[4] https://code.google.com/p/word2vec/

we follow the criteria of the previous work (Ji and Grishman, 2008; Liao and Grishman, 2010; Li et al., 2013; Chen et al., 2015) to judge the correctness of the predicted event mentions.

5.2 Memory Vector/Matrices

This section evaluates the effectiveness of the memory vector and matrices presented in Section 3.2.3. In particular, we test the joint model on different cases where the memory vector for triggers G^{trg} and the memory matrices for arguments $G^{arg/trg}$ and G^{arg} are included or excluded from the model. As there are 4 different ways to combine $G^{arg/trg}$ and G^{arg} for argument labeling and two options to to employ G^{trg} or not for trigger labeling, we have 8 systems for comparison in total. Table 1 reports the identification and classification performance (F1 scores) for triggers and argument roles on the development set. Note that we are using the word embeddings trained with the C-CBOW technique in this section.

System		*No*	$G^{arg/trg}$	G^{arg}	$G^{arg/trg}$+G^{arg}
No	Trigger	67.9	**68.0**	64.6	64.2
	Argument	55.6	**58.1**	55.2	53.1
G^{trg}	Trigger	63.8	61.0	61.3	66.8
	Argument	55.2	56.6	54.7	53.6

Table 1: Performance of the Memory Vector/Matrices on the development set. *No* means not using the memory vector/matrices.

We observe that the memory vector G^{trg} is not helpful for the joint model as it worsens both trigger and argument role performance (considering the same choice of the memory matrices $G^{arg/trg}$ and G^{arg} (i.e, the same row in the table) and except in the row with $G^{arg/trg} + G^{arg}$).

The clearest trend is that $G^{arg/trg}$ is very effective in improving the performance of argument labeling. This is true in both the inclusion and exclusion of G^{trg}. G^{arg} and its combination with $G^{arg/trg}$, on the other hand, have negative effect on this task. Finally, $G^{arg/trg}$ and G^{arg} do not contribute much to the trigger labeling performance in general (except in the case where G^t, $G^{arg/trg}$ and G^{arg} are all applied).

These observations suggest that the dependencies *among trigger subtypes* and *among argument roles* are not strong enough to be helpful for the joint model in this dataset. This is in contrast to the dependencies *between argument roles and trigger subtypes* that improve the joint model significantly.

The best system corresponds to the application of the memory matrix $G^{arg/trg}$ and will be used in all the experiments below.

5.3 Word Embedding Evaluation

We investigate different techniques to obtain the pre-trained word embeddings for initialization in the joint model of EE. Table 2 presents the performance (for both triggers and argument roles) on the development set when the CBOW, SKIP-GRAM and C-CBOW techniques are utilized to obtain word embeddings from the same corpus. We also report the performance of the joint model when it is initialized with the Word2Vec word embeddings from Mikolov et al. (2013a; 2013b) (trained with the Skip-gram model on Google News) (WORD2VEC). Finally, for comparison, the performance of the random word embeddings (RANDOM) is also included. All of these word embeddings are updated during the training of the model.

Word Embeddings	Trigger	Argument
RANDOM	59.9	50.1
SKIP-GRAM	66.7	57.1
CBOW	66.5	53.8
WORD2VEC	66.9	56.4
C-CBOW	**68.0**	**58.1**

Table 2: Performance of the Word Embedding Techniques.

The first observation from the table is that RANDOM is not good enough to initialize the word embeddings for joint EE and we need to borrow some pre-trained word embeddings for this purpose. Second, SKIP-GRAM, WORD2VEC and CBOW have comparable performance on trigger labeling while the argument labeling performance of SKIP-GRAM and WORD2VEC is much better than that of CBOW for the joint EE model. Third and most importantly, among the compared word embeddings, it is clear that C-CBOW significantly outperforms all the others. We believe that the better performance of C-CBOW stems from its concatenation of the multiple context word vectors, thus providing more information to learn better word embeddings than SKIP-GRAM and WORD2VEC. In addition, the concate-

Model	Trigger Identification (%)			Trigger Identification + Classification (%)			Argument Identification (%)			Argument Role (%)		
	P	R	F	P	R	F	P	R	F	P	R	F
Li's basline	76.2	60.5	67.4	74.5	59.1	65.9	74.1	37.4	49.7	65.4	33.1	43.9
Liao's cross-event†	N/A			68.7	68.9	68.8	50.9	49.7	50.3	45.1	44.1	44.6
Hong's cross-entity†	N/A			72.9	64.3	68.3	53.4	52.9	53.1	51.6	45.5	48.3
Li's structure	76.9	65.0	70.4	73.7	62.3	67.5	69.8	47.9	56.8	64.7	44.4	52.7
DMCNN	80.4	67.7	**73.5**	75.6	63.6	69.1	68.8	51.9	59.1	62.2	46.9	53.5
JRNN	68.5	75.7	71.9	66.0	73.0	**69.3**	61.4	64.2	**62.8**	54.2	56.7	**55.4**

Table 3: Overall Performance on the Blind Test Data. "†" designates the systems that employ the evidences beyond sentence level.

nation mechanism essentially helps to assign different weights to different context words, thereby being more flexible than CBOW that applies a single weight for all the context words.

From now on, for consistency, C-CBOW would be utilized in all the following experiments.

5.4 Comparison to the State of the art

The state-of-the-art systems for EE on the ACE 2005 dataset have been the pipelined system with dynamic multi-pooling convolutional neural networks by Chen et al. (2015) (**DMCNN**) and the joint system with structured prediction and various discrete local and global features by Li et al. (2013) (**Li's structure**).

Note that the pipelined system in Chen et al. (2015) is also the best-reported system based on neural networks for EE. Table 3 compares these state-of-the-art systems with the joint RNN-based model in this work (denoted by **JRNN**). For completeness, we also report the performance of the following representative systems:

1) **Li's baseline**: This is the pipelined system with local features by Li et al. (2013).

2) **Liao's cross event**: is the system by Liao and Grishman (2010) with the document-level information.

3) **Hong's cross-entity** (Hong et al., 2011): This system exploits the cross-entity inference, and is also the best-reported pipelined system with discrete features in the literature.

From the table, we see that JRNN achieves the best F1 scores (for both trigger and argument labeling) among all of the compared models. This is significant with the argument role labeling performance (an improvement of 1.9% over the best-reported model DMCNN in Chen et al. (2015)), and demonstrates the benefit of the joint model with RNNs and memory features in this work. In addition, as JRNN significantly outperforms the joint model with discrete features in Li et al. (2013) (an improvement of 1.8% and 2.7% for trigger and argument role labeling respectively), we can confirm the effectiveness of RNNs to learn effective feature representations for EE.

5.5 Sentences with Multiple Events

In order to further prove the effectiveness of JRNN, especially for those sentences with multiple events, we divide the test data into two parts according to the number of events in the sentences (i.e, single event and multiple events) and evaluate the performance separately, following Chen et al. (2015). Table 4 shows the performance (F1 scores) of JRNN, DMCNN and two other baseline systems, named **Embeddings+T** and **CNN** in Chen et al. (2015). Embeddings+T uses word embeddings and the traditional sentence-level features in (Li et al., 2013) while CNN is similar to DMCNN, except that it applies the standard pooling mechanism instead of the dynamic multi-pooling method (Chen et al., 2015).

The most important observation from the table is that JRNN significantly outperforms all the other methods with large margins when the input sentences contain more than one events (i.e, the row labeled with **1/N** in the table). In particular, JRNN is 13.9% better than DMCNN on trigger labeling while the corresponding improvement for argument role labeling is 6.5%, thereby further suggesting the benefit of JRNN with the memory features. Regard-

Stage	Model	1/1	1/N	all
Trigger	Embedding+T	68.1	25.5	59.8
	CNN	72.5	43.1	66.3
	DMCNN	74.3	50.9	69.1
	JRNN	**75.6**	**64.8**	**69.3**
Argument	Embedding+T	37.4	15.5	32.6
	CNN	51.6	36.6	48.9
	DMCNN	**54.6**	48.7	53.5
	JRNN	50.0	**55.2**	**55.4**

Table 4: System Performance on Single Event Sentences (**1/1**) and Multiple Event Sentences (**1/N**).

ing the performance on the single event sentences, JRNN is still the best system on trigger labeling although it is worse than DMCNN on argument role labeling. This can be partly explained by the fact that DMCNN includes the position embedding features for arguments and the memory matrix $G^{\text{arg/trg}}$ in JRNN is not functioning in this single event case.

6 Related Work

Early research on event extraction has primarily focused on local sentence-level representations in a pipelined architecture (Grishman et al., 2005; Ahn, 2006). After that, higher level features has been investigated to improve the performance (Ji and Grishman, 2008; Gupta and Ji, 2009; Patwardhan and Riloff, 2009; Liao and Grishman, 2010; Liao and Grishman, 2011; Hong et al., 2011; McClosky et al., 2011; Huang and Riloff, 2012; Li et al., 2013). Besides, some recent research has proposed joint models for EE, including the methods based on Markov Logic Networks (Riedel et al., 2009; Poon and Vanderwende, 2010; Venugopal et al., 2014), structured perceptron (Li et al., 2013; Li et al., 2014b), and dual decomposition (Riedel et al. (2009; 2011a; 2011b)).

The application of neural networks to EE is very recent. In particular, Nguyen and Grishman (2015b) study domain adaptation and event detection via CNNs while Chen et al. (2015) apply dynamic multi-pooling CNNs for EE in a pipelined framework. However, none of these work utilizes RNNs to perform joint EE as we do in this work.

7 Conclusion

We present a joint model to do EE based on bidirectional RNN to overcome the limitation of the previous models for this task. We introduce the memory matrix that can effectively capture the dependencies between argument roles and trigger subtypes. We demonstrate that the concatenation-based variant of the CBOW word embeddings is very helpful for the joint model. The proposed joint model is empirically shown to be effective on the sentences with multiple events as well as yields the state-of-the-art performance on the ACE 2005 dataset. In the future, we plan to apply this joint model on the event argument extraction task of the KBP evaluation as well as extend it to other joint tasks such as mention detection together with relation extraction etc.

References

David Ahn. 2006. The stages of event extraction. In *Proceedings of the Workshop on Annotating and Reasoning about Time and Events*.

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. 2014. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL*.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. In *Journal of Machine Learning Research 3*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. In *Journal of Machine Learning Research 3*.

Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. 2015. Event extraction via dynamic multi-pooling convolutional neural networks. In *ACL-IJCNLP*.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *EMNLP*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *arXiv preprint arXiv:1412.3555*.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML*.

Felix Gers. 2001. Long short-term memory in recurrent neural networks. In *PhD Thesis*.

Ralph Grishman, David Westbrook, and Adam Meyers. 2005. Nyus english ace 2005 system description. In *ACE 2005 Evaluation Workshop*.

Prashant Gupta and Heng Ji. 2009. Predicting unknown time arguments based on cross-event propagation. In *ACL-IJCNLP*.

Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. In *Neural Computation*.

Yu Hong, Jianfeng Zhang, Bin Ma, Jianmin Yao, Guodong Zhou, and Qiaoming Zhu. 2011. Using cross-entity inference to improve event extraction. In *ACL*.

Ruihong Huang and Ellen Riloff. 2012. Modeling textual cohesion for event extraction. In *AAAI*.

Heng Ji and Ralph Grishman. 2008. Refining event extraction through cross-document inference. In *ACL*.

Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *ICML*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP*.

Qi Li, Heng Ji, and Liang Huang. 2013. Joint event extraction via structured prediction with global features. In *ACL*.

Qi Li, Heng Ji, Yu Hong, and Sujian Li. 2014b. Constructing information networks using one single model. In *EMNLP*.

Shasha Liao and Ralph Grishman. 2010. Using document level cross-event inference to improve event extraction. In *ACL*.

Shasha Liao and Ralph Grishman. 2011. Acquiring topic features to improve event extraction: in pre-selected and balanced collections. In *RANLP*.

David McClosky, Mihai Surdeanu, and Christopher Manning. 2011. Event extraction as dependency parsing. In *BioNLP Shared Task Workshop*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In *ICLR*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *NIPS*.

Thien Huu Nguyen and Ralph Grishman. 2015a. Relation extraction: Perspective from convolutional neural networks. In *The NAACL Workshop on Vector Space Modeling for NLP (VSM)*.

Thien Huu Nguyen and Ralph Grishman. 2015b. Event detection and domain adaptation with convolutional neural networks. In *ACL-IJCNLP*.

Siddharth Patwardhan and Ellen Riloff. 2009. A unified model of phrasal and sentential evidence for information extraction. In *EMNLP*.

Hoifung Poon and Lucy Vanderwende. 2010. Joint inference for knowledge extraction from biomedical literature. In *NAACL-HLT*.

Sebastian Riedel and Andrew McCallum. 2011a. Fast and robust joint models for biomedical event extraction. In *EMNLP*.

Sebastian Riedel and Andrew McCallum. 2011b. Robust biomedical event extraction with dual decomposition and minimal domain adaptation. In *BioNLP Shared Task 2011 Workshop*.

Sebastian Riedel, Hong-Woo Chun, Toshihisa Takagi, and Jun'ichi Tsujii. 2009. A markov logic approach to bio-molecular event extraction. In *BioNLP 2009 Workshop*.

Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *ACL*.

Deepak Venugopal, Chen Chen, Vibhav Gogate, and Vincent Ng. 2014. Relieving the computational bottleneck: Joint inference for event extraction with high-dimensional features. In *EMNLP*.

Matthew D. Zeiler. 2012. Adadelta: An adaptive learning rate method. In *CoRR, abs/1212.5701*.