New York University 2016 System for KBP Event Nugget: A Deep Learning Approach

Thien Huu Nguyen, Adam Meyers and Ralph Grishman
Computer Science Department
New York University
New York, NY 10003 USA
{thien,meyers,grishman}@cs.nyu.edu

Abstract

This is the first time New York University (NYU) participates in the event nugget (EN) evaluation of the Text Analysis Conference (TAC). We developed EN systems for both subtasks of event nugget, i.e, EN Task 1: Event Nugget Detection and EN Task 2: Event Nugget Detection and Coreference. The systems are mainly based on our recent research on deep learning for event detection (Nguyen and Grishman, 2015a; Nguyen and Grishman, 2016a). Due to the limited time we could devote to system development this year, we only ran the systems on the English evaluation data. However, we expect that the adaptation of the current systems to new languages can be done quickly. The development experiments show that although our current systems do not rely on complicated feature engineering, they significantly outperform the reported systems last year for the EN subtasks on the 2015 evaluation data.

1 Overview

We follow a pipelined approach to build the EN systems this year. Essentially, the whole system involves three following components in order: (i) event detection and classification (called *typeEN*), (ii) event realis classification (called *realisEN*), and (iii) event coreference resolution (called *corefEN*). The input for *typeEN* is raw text while the other components (i.e, *realisEN* and *corefEN*) take the outputs of their previous components (i.e, *typeEN* and *realisEN* respectively) as the inputs. The output of *realisEN* is submitted to the EN Task 1 evaluation while the output of *corefEN* is used for the EN Task 2 evaluation.

We use neural networks as the main technique in all the three components (i.e, *typeEN*, *realisEN* and *corefEN*). This allows us to automatically learn features from data rather than doing manual feature engineering as the traditional methods for EN. The following sections will present each component in detail. An overview of the whole system is presented in Figure 1.

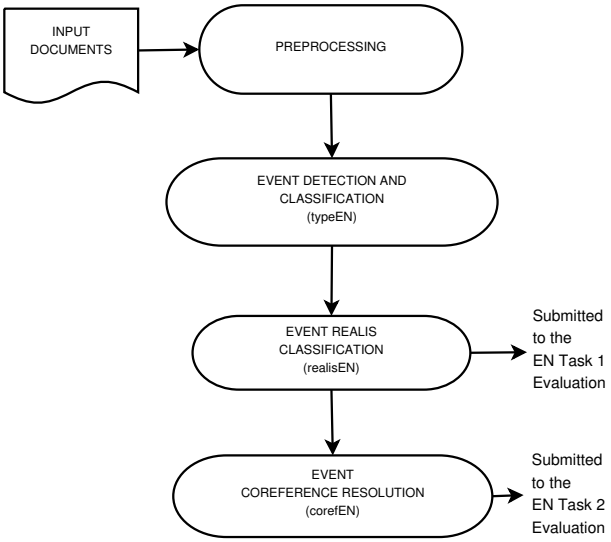

Figure 1: The NYU 2016 System for Event Nugget.

2 Event Detection and Classification: *typeEN*

The goal of this component is to detect the event mentions in the input documents and classify them into the predefined types and subtypes of interest.

In the current system, we predict the event subtypes directly and use these subtypes to map back to the types when we report the detected event mentions for submission.

In order to prepare the input documents for neural networks, our preprocessing steps include sentence detection and tokenization using the OpenNLP toolkit[1], and dependency parsing for the detected sentences using the CoreNLP toolkit[2] from Stanford University.

In this work, we assume event mentions to be only single tokens in text and do not consider event mentions spanning multiple tokens. Although this assumption affects our performance on the multiple-token event mentions, it enables the introduction of the position information into the neural network models that is very helpful to improve our overall performance.

Given the single-token view, we cast the event detection and classification problem as a classification problem for every token in the input documents[3]. Note that although the selected event subtypes for evaluation this year (with 19 subtypes) is only a subset of the last year's event subtype set (with 38 subtypes), we still predict the tokens for 39 classes in the systems (38 subtypes in the last year's evaluation plus one type for "*NONE*"). However, in the submission files, we do not include the detected event mentions whose subtypes are excluded in the evaluation this year. We expect that this inclusion of more subtypes for prediction would help the systems to identify event mentions and distinguish them better.

2.1 Encoding

Formally, for every token in an input sentence, we want to predict its event subtype ("*NONE*" if it is not an event mention). The current token along with its context in the sentence constitute an event mention candidate or an example in multi-class classification terms. In order to prepare the example for neural networks, we limit the context to a fixed window size by trimming longer sentences and padding shorter sentences with a special token when necessary. Let $2w + 1$ be the fixed window size, and $x = [x_{-w}, x_{-w+1}, \ldots, x_0, \ldots, x_{w-1}, x_w]$ be some event mention candidate where the current token is positioned in the middle of the window (token x_0). Before entering the neural networks, each token x_i is transformed into a real-valued vector $\mathbf{x}_i$ using the concatenation of the following vectors:

1. Word Embedding of x_i (pre-trained on some large corpus): to capture the hidden semantic and syntactic properties of the token (Mikolov et al., 2013).

2. Position Embedding of x_i: to embed the relative distance i of x_i to the current token x_0. In practice, we initialize this table randomly.

3. Dependency Vector of x_i: to encode the dependency features that are shown to be helpful in the previous research (Nguyen et al., 2016b). The dimensionality of this vector is the number of the possible relations between tokens in the dependency trees. The value at each dimension is set to 1 if there exists one edge of the corresponding relation connected to x_i in the dependency tree of x, and 0 otherwise.

After the transformation for each token, the original event mention x is transformed into a matrix $\mathbf{x} = [\mathbf{x}_{-w}, \mathbf{x}_{-w+1}, \ldots, \mathbf{x}_0, \ldots, \mathbf{x}_{w-1}, \mathbf{x}_w]$ of size $m_t \times (2w+1)$ (m_t is the dimensionality of the concatenated vectors of the tokens). The matrix representation $\mathbf{x}$ is then fed into a neural network model to learn a representation for event subtype classification.

2.2 Non-consecutive Convolutional Neural Networks

The main neural network architecture we employ in this work is the non-consecutive convolutional neural networks (NCNN) (Nguyen et al., 2016b). The architecture of NCNN is similar to that of the traditional convolutional neural networks (denoted by CNN) as both NCNN and CNN pass the input matrix representation $\mathbf{x}$ through a convolution layer, followed by a max pooling layer to induce a more abstract representation (automatic feature extraction) for the current token (Nguyen and

[1] https://opennlp.apache.org

[2] http://stanfordnlp.github.io/CoreNLP

[3] We do not perform classification at the sentence level in this work as the token level has been shown to be more effective for event detection with neural networks in (Nguyen and Grishman, 2016a).

Grishman, 2015a; Nguyen and Grishman, 2015b; Nguyen et al., 2016c). However, NCNN differs from the traditional CNN in that the convolution in NCNN is done over the arbitrary non-consecutive n-grams while the traditional CNN only convolutes over the consecutive n-grams in the sentences temporally. This helps NCNN to explicitly model the non-consecutive n-grams that are crucial to the prediction of event subtypes in some situations. For instance, consider the following sentence with the word "*leave*" from the ACE 2005 corpus:

*The mystery is that she took the job in the first place or didn't **leave** earlier.*

The correct event subtype for the word "*leave*" in this case is "*End-Position*". However, the traditional CNN models might not be able to detect "*leave*" as an event mention or incorrectly predict its subtype as "*Movement*". This is caused by their reliance on the consecutive local k-grams such as "*leave earlier*". Consequently, we need to resort to the non-consecutive pattern "*job leave*" to correctly determine the event type of "*leave*" in this case. NCNN has been shown to be significantly better than the traditional CNN for event detection in the ACE 2005 dataset (Nguyen and Grishman, 2015a; Nguyen and Grishman, 2016a). A more formal presentation of NCNN can be found in (Nguyen and Grishman, 2016a).

Let $\mathbf{f}_{\text{NCNN}}$ be the vector representation learnt by NCNN for $\mathbf{x}$. In order to enrich $\mathbf{f}_{\text{NCNN}}$, we concatenate it with the local vector $\mathbf{f}_{\text{local}}$. The resulting concatenation vector $[\mathbf{f}_{\text{NCNN}}, \mathbf{f}_{\text{local}}]$ is then used as input for another fully connected layer, followed by a softmax layer to compute the probability distribution over the possible event subtypes for the current token x_0. In this work, the local vector $\mathbf{f}_{\text{local}}$ is obtained by concatenating the word embeddings of the tokens in some fixed context window[4] of x_0. An overview of NCNN is demonstrated in Figure 2.

2.3 Recurrent Neural Networks

Besides NCNN, in this work, we also examine bidirectional recurrent neural networks for event nugget (BRNN). However, as BRNN is shown to be worse than NCNN for event detection in (Nguyen and Grishman, 2016a), we seek the application of BRNN by combining it with NCNN. We expect that BRNN and NCNN would capture different information for event nugget so their combination would help to improve the overall performance (Nguyen and Grishman, 2016d).

In particular, for BRNN, we consider the input matrix $\mathbf{x}$ as a sequence of vectors: $\mathbf{x} = (\mathbf{x}_{-w}, \mathbf{x}_{-w+1}, \ldots, \mathbf{x}_0, \ldots, \mathbf{x}_{w-1}, \mathbf{x}_w)$, indexed from $-w$ to w. At each step i, we compute the hidden vector α_i based on the current input vector $\mathbf{x}_i$ and the previous hidden vector α_{i-1}, using the non-linear transformation function[5] Φ: $\alpha_i = \Phi(\mathbf{x}_i, \alpha_{i-1})$. This recurrent computation is done over $\mathbf{x}$ to generate the hidden vector sequence $(\alpha_{-w}, \alpha_{-w+1}, \ldots, \alpha_0, \ldots, \alpha_{w-1}, \alpha_w)$, denoted by $\overrightarrow{\text{RNN}}(\mathbf{x}_{-w}, \mathbf{x}_{-w+1}, \ldots, \mathbf{x}_0, \ldots, \mathbf{x}_{w-1}, \mathbf{x}_w) = (\alpha_{-w}, \alpha_{-w+1}, \ldots, \alpha_0, \ldots, \alpha_{w-1}, \alpha_w)$. In addition, following (Nguyen et al., 2016b), we run a second recurrent neural network in the reverse direction from $\mathbf{x}_w$ from $\mathbf{x}_{-w}$ to generate second hidden sequence: $\overleftarrow{\text{RNN}}(\mathbf{x}_w, \mathbf{x}_{w-1}, \ldots, \mathbf{x}_0, \ldots, \mathbf{x}_{-w+1}, \mathbf{x}_{-w}) = (\alpha'_w, \alpha'_{w-1}, \ldots, \alpha_0, \ldots, \alpha_{-w+1}, \alpha_{-w})$. Afterward, the concatenation of the hidden vectors α_0 and α'_0 is used as the representation $\mathbf{f}_{\text{BRNN}} = [\alpha_0, \alpha'_0]$ of BRNN for the original entity mention x. Finally, in order to combine BRNN and NCNN, we concatenate the representation vectors $\mathbf{f}_{\text{BRNN}}$, $\mathbf{f}_{\text{NCNN}}$ and $\mathbf{f}_{\text{local}}$ to form the input for computing the subtype probability distribution as in Section 2.2. This combined model is called NCNN+BRNN for convenience.

3 Realis Classification: *realisEN*

We employ the similar encoding and network architectures (i.e, NCNN and NCNN+BRNN) as *typeEN* to classify entity mentions for realis. The only difference is that we are predicting three classes for realis (i.e, ACTUAL, GENERIC and OTHER) rather than 39 event subtype classes as in *typeEN*. Note that as our system is pipelined, we only perform realis prediction for the event mentions detected in typeEN.

Modality Features

In this work, beside the automatic learnt features

[4]The size of this window is set to 5 in our systems.

[5]We use the gated recurrent units (GRU) for Φ to avoid the vanishing gradient problem in this work.

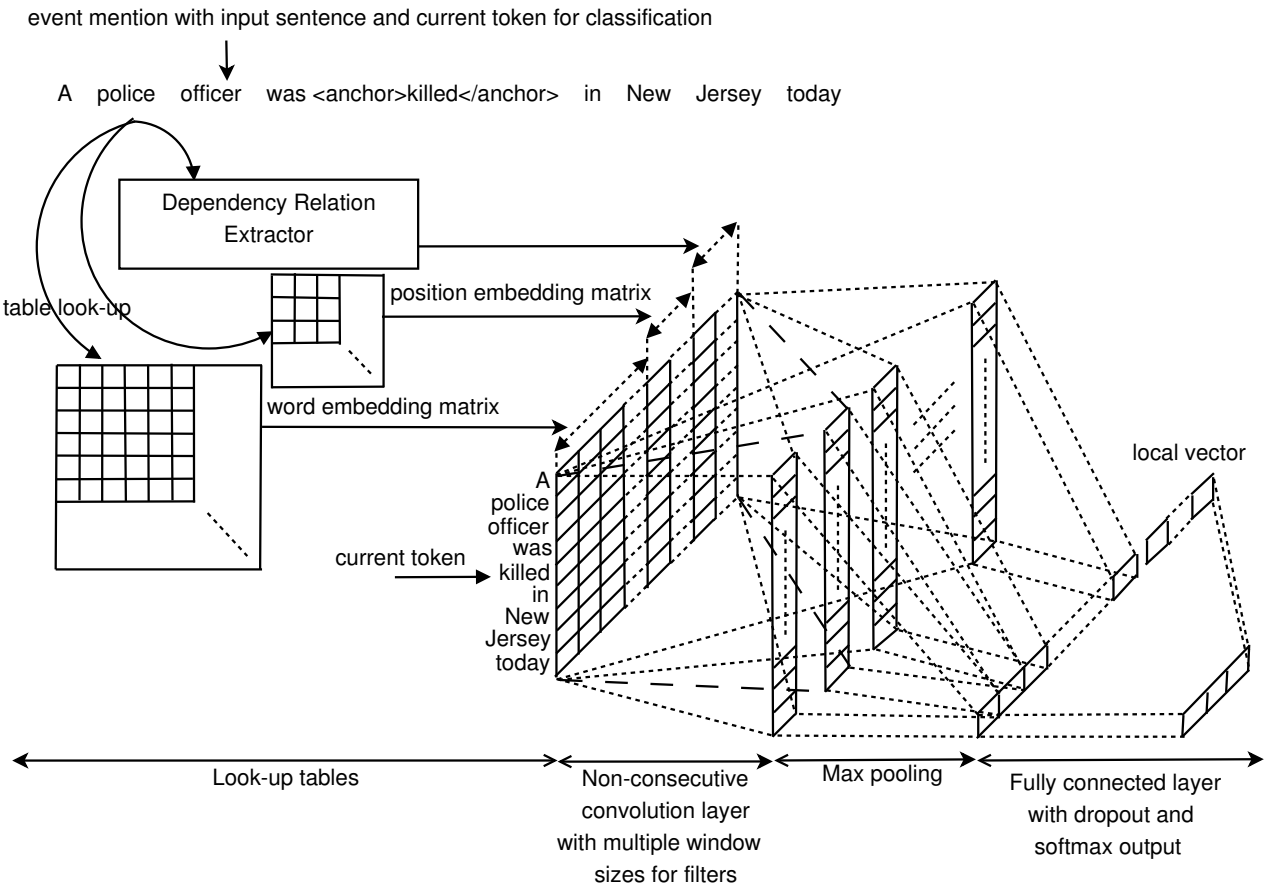

Figure 2: Non-consecutive Convolutional Neural Networks for Event Detection and Classification.

from neural network models for realis classification, we also examine some in-house binary features that are specifically designed to capture the modality of the event mentions. We augment the automatic features from the neural networks with such binary features in the same way we combine the representations of BRNN and NCNN.

Specifically, we use the following types of features extracted from the output of the GLARF semantic parser (Meyers et al., 2009; Meyers et al., 2011)[6], including:

- Features based on whether event tokens (or their arguments) are in the scope of particular scope operators, including: quantifiers (i.e, *"every", "some"* etc.), verbs licensing belief contexts (i.e, *"believe", "assume"* etc.), epistemic adverbs/adjectives (i.e, *"possible", "possibly", "maybe"* etc.), modals (i.e, *"may", "will"* etc.), infinitival *"to"*, temporal adverbs/adjectives (i.e, *"tomorrow", "future"* etc.), negation words (i.e, *"not", "no", "none", "never", "deny", "refuse"*, etc.) and others. In the following examples, the boldfaced event word is in the scope of the underlined word, indicating that the event is not ACTUAL.
  - *A military spokesman on Saturday denied troops had **crossed** into Iraq.*
  - *This upcoming **visit** to Russia will be my first **trip** aboard.*
  - *Terrorists normally **attack** during the winter.*
  - *Every terrorist gets **arrested** eventually.*
- Morphological features of particular words, projected on words within their scope. For example, present tense verbs with non-definite noun arguments are the most likely to be

[6] http://nlp.cs.nyu.edu/meyers/GLARF.html

generic, e.g., *Terrorists attack in March*.

- Features about attribution, indicating if an event is attributed to someone other than the author (e.g., *John said that they attacked*) or furthermore, expressing something about the realis features of the attribution event, e.g., *Mary may have said that they attacked*.

- Features based on manual rules that predict a more fine-grained set of realis-like distinctions (similar to the features used in ACE (LDC, 2005)). The last type of features are largely dependent on all of the former sets among other features. These are rules tying particular morphological tenses to particular features, e.g., past tense largely predicts ACTUAL (*They attacked in March*), and present tense in combination with indefinite noun arguments largely predicts GENERIC (*Terrorists attack in March*).

4 Event Coreference Resolution: *corefEN*

Regarding the event coreference resolution component *corefEN*, we cast it as a binary classification task for every event mention pair in a document (i.e, whether two event mentions in a document corefer or not). The coreference decisions for all the event mention pairs are then used to form an undirected graph whose connected components are reported as coreference chains for the document. Again, we only consider the pairs of event mentions that are detected in the previous components.

Our coreference resolution component is based on the assumption that two event mentions corefer if their subtypes and realises match, and their context information is similar. In this work, the matches of subtypes and realis are encoded via a one-hot feature vector while the context similarity is captured via the vector representations learnt from neural networks for event mentions. In particular, let $\mathbf{u}$ and $\mathbf{v}$ be the two event mentions of interest, and $\mathbf{f}_{\text{binary}}$ be the one-hot feature vector for the matches of subtypes and realis of $\mathbf{u}$ and $\mathbf{v}$. Note that we also include the number of sentences between the two event mentions in the document as one piece of information in $\mathbf{f}_{\text{binary}}$. Also, let $\mathbf{f}^u_{\text{NN}}$ and $\mathbf{f}^v_{\text{NN}}$ be the two vector representations learnt by some neural network (i.e, NCNN or NCNN+BRNN)[7], and $\mathbf{f}^u_{\text{local}}$ and $\mathbf{f}^v_{\text{local}}$ be the local vector for $\mathbf{u}$ and $\mathbf{v}$ respectively. Given such vectors, their concatenation, i.e, $[\mathbf{f}^u_{\text{NN}}, \mathbf{f}^v_{\text{NN}}, \mathbf{f}^u_{\text{local}}, \mathbf{f}^v_{\text{local}}, \mathbf{f}_{\text{binary}}]$, is then used as features to make the coreference decision for $\mathbf{u}$ and $\mathbf{v}$ in *corefEN*.

5 Parameters, Resources and Training

We use the same parameters for the neural network models in *typeEN*, *realisEN* and *corefEN*. Specifically, we employ: window sizes in the set $\{2, 3, 4, 5\}$ for the convolution operations (Nguyen and Grishman, 2015a; Nguyen and Grishman, 2016a), 300 filters for each convolution window size, and 300 hidden units for BRNN.

Regarding the embeddings, we use 50 and 300 dimensions for the position embeddings and word embeddings respectively. We pre-train the word embeddings from the English Gigaword corpus utilizing the *word2vec* toolkit[8]. Following the previous work, we employ the context window of 5, the subsampling of the frequent words set to $1e$-05 and 10 negative samples (Nguyen et al., 2016b; Nguyen et al., 2016e). Note that we modify the CBOW model in *word2vec* so it predicts the current words using the concatenation of the word embeddings of the surrounding words. We apply this modified CBOW model for training the word embeddings as in (Nguyen et al., 2016b; Nguyen et al., 2016e).

Finally, we train the neural network models using stochastic gradient descent with shuffled mini-batches (batch size = 50), dropout for regularization (rate = 0.5), back-propagation for gradients, and the AdaDelta update rule. We rescale the weights whose l_2-norms exceed a predefined threshold (set to 3 in this work)[9]. Most of the parameters and settings in this section are inherited from our previous studies (Nguyen and Grishman, 2015a; Nguyen and Grishman, 2015b). The other parameters are tuned via the development data.

[7] Note that the encoding of the event mentions in *corefEN* is similar to that of *typeEN* and *realisEN*.

[8] `https://code.google.com/p/word2vec/`

[9] There is one exception in the *corefEN* component where we do not apply dropout and the l_2 norm in the training.

6 Evaluation

NYU submitted three runs to the EN evaluation this year (called NYU1, NYU2 and NYU3). The runs are different in the type of the neural networks and features employed in the components *typeEN*, *realisEN* and *corefEN*. The configurations of the runs are presented in Table 1.

Runs	Components		
	typeEN	*realisEN*	*corefEN*
NYU1	NCNN	NCNN	NCNN
NYU2	NCNN	NCNN + Modality Features	NCNN
NYU3	NCNN+BRNN	NCNN+BRNN	NCNN+BRNN

Table 1: Models and features for different runs of NYU.

System	Plain	Type	Realis	Type & Realis	Coref score
NYU1	71.07	62.72	56.12	49.70	43.14
NYU2	71.16	62.65	57.41	50.43	43.40
NYU3	70.03	62.38	55.62	49.86	43.94

Table 2: Performance of NYU1, NYU2 and NYU3 on the development data.

System	Plain	Type	Realis	Type & Realis	Coref score
NYU1	53.84	44.37	42.68	35.24	27.07
NYU2	52.39	44.12	41.73	35.22	26.28
NYU3	54.07	44.38	41.19	33.60	26.94

Table 3: Performance of NYU1, NYU2 and NYU3 on the 2016 official evaluation data for English.

6.1 Evaluating the Proposed Systems

In order to train the proposed systems for official submission, we use the union of the three following corpora:

- The training data for the EN 2015 evaluation (called A)
- The DEFT Rich ERE English Training Annotation Dataset (called B)
- Haft of the evaluation data for EN 2015 evaluation (i.e, 102 out of 202 evaluation documents for 2015) (called C)

We utilize the remaining documents in the 2015 evaluation data for system development (i.e, 100 remaining documents). The performance of the three systems on the development data and the official evaluation data for English this year is presented in Table 2 and Table 3 respectively. These tables includes the scores for event detection (*Plain*), event classification (*Type*), realis classification (*Realis*), realis and type classification (*Type & Realis*), and coreference resolution (*coref*). All the scores are computed using the official scorer this year. Note that the scores in Table 2 are based on the 38 event subtypes in the 2015 evaluation while the scores in Table 3 correspond to the 19 evaluated event subtypes this year.

As we can see from the tables, the reduction from the 38 subtypes last year to the subset of 19 subtypes this year causes the significant performance drops over all the subtasks and systems. This demonstrates that the selected subtypes for this year are actually more challenging than the other subtypes last year and we should spend more research effort targeting these challenging subtypes. Regarding the comparison of NYU1, NYU2 and NYU3, it is very difficult to draw a concrete conclusion as their performance in the two tables shows some mixed assessments over different subtasks and settings (i.e, using 38 subtypes or 19 subtypes). For instance, NYU2 is better than NYU1 and NYU3 on the two tasks of realis (i.e, *Realis* and *Type & Realis*) in Table 2 for the development data, suggesting the benefits of the binary modality features for event realis classification. However, this is not true in Table 3 for the evaluation data this year as NYU2 turns out to be a little worse than NYU1 in this case. The only general trend we can observe is the small performance gap between the systems (i.e, NYU1, NYU2 and NYU3) in the same settings. This suggests that BRNN and the modality features do not introduce much new information for NCNN and we can just use NCNN for different subtasks of event nugget. Consequently, we will only focus on NYU1 in the following evaluation.

6.2 Comparing to the 2015 Systems

In order to compare our neural network systems with the systems participating in the event nugget evaluation last year, we train NYU1 on the the union of the corpora A and B and evaluate it on the 2015 official evaluation data. Table 4 presents the performance of

NYU1 and the three top systems in the evaluation last year (Mitamura et al., 2015).

System	Plain	Type	Realis	Type & Realis	Coref score
NYU1	**67.77**	**59.74**	**53.82**	**47.26**	**40.11**
Top 1 in 2015	60.77	57.18	40.35	38.06	39.12
Top 2 in 2015	62.13	57.41	47.85	43.73	37.23
Top 3 in 2015	64.56	57.45	45.21	39.67	32.36

Table 4: Performance comparison of NYU1 and the best systems in the 2015 event nugget evaluation.

It is very clear from the table that NYU1 is substantially better than the top systems last year over all the tasks, demonstrating the advantages of the NCNN model for the event nugget tasks.

References

LDC. 2005. Ace (automatic content extraction) english annotation guidelines for events. https://www.ldc.upenn.edu/sites/www.ldc.upenn.edu/files/english-events-guidelines-v5.4.3.pdf.

A. Meyers, M. Kosaka, H. Ji, N. Xue, M. Harper, A. Sun, W. Xu, and S. Liao. 2009. Transducing Logical Relations from Automatic and Manual Annotation. In *The Linguistic Annotation Workshop III, ACL 2009*.

A. Meyers, M. Kosaka, S. Liao, and N. Xue. 2011. Improving mt word alignment using aligned multi-stage parses. In *SSST-5*.

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

Teruko Mitamura, Zhengzhong Liu, and Eduard Hovy. 2015. Overview of tac kbp 2015 event nugget track. In *TAC*.

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

Thien Huu Nguyen and Ralph Grishman. 2015b. Relation extraction: Perspective from convolutional neural networks. In *Proceedings of the 1st NAACL Workshop on Vector Space Modeling for NLP (VSM)*.

Thien Huu Nguyen and Ralph Grishman. 2016a. Modeling skip-grams for event detection with convolutional neural networks. In *Proceedings of EMNLP*.

Thien Huu Nguyen and Ralph Grishman. 2016d. Combining neural networks and log-linear models to improve relation extraction. In *Proceedings of IJCAI Workshop on Deep Learning for Artificial Intelligence (DLAI)*.

Thien Huu Nguyen, Kyunghyun Cho, and Ralph Grishman. 2016b. Joint event extraction via recurrent neural networks. In *NAACL*.

Thien Huu Nguyen, Lisheng Fu, Kyunghyun Cho, and Ralph Grishman. 2016c. A two-stage approach for extending event detection to new types via neural networks. In *Proceedings of the 1st ACL Workshop on Representation Learning for NLP (RepL4NLP)*.

Thien Huu Nguyen, Avirup Sil, Georgiana Dinu, and Radu Florian. 2016e. Toward mention detection robustness with recurrent neural networks. In *Proceedings of IJCAI Workshop on Deep Learning for Artificial Intelligence (DLAI)*.