CEA LIST Participation to the TAC 2017 Event Nugget Track

Dorian Kodelja, Romaric Besançon, Olivier Ferret, Hervé Le Borgne, E. Boroş

CEA LIST Participation to the TAC 2017 Event Nugget Track

D. Kodelja **R. Besançon** **O. Ferret** **H. Le Borgne** **E. Boros**
CEA, LIST, Vision and Content Engineering Laboratory, Gif-sur-Yvette, F-91191 France.
{dorian.kodelja,romaric.besancon,olivier.ferret,
herve.le-borgne}@cea.fr

Abstract

This is the first participation of CEA LIST to the Event Nugget track. We only considered the English dataset. The purpose of our submissions was to investigate the interest of considering the detection and the classification of event mentions as two separate tasks rather than a joint task. Our baseline, which yields our best results, is inspired from the TAC 2016 deep learning approach of (Nguyen et al., 2016b).

1 Overview

The goal of the Event Nugget (EN) task is to extract event mentions (*i.e.* words most clearly showing that an event takes place) from documents. Moreover, those event mentions are assigned a type and a subtype according to a pre-defined taxonomy. This taxonomy, the DEFT Rich ERE (Song et al., 2015), defines 38 subtypes among 9 types. Since the subtypes of Rich ERE are mentioned as "types" in the track guidelines and scorer, we will respectively refer to the Rich ERE type and subtype level annotation as “TYPE" and “type". Along these TYPEs and types, we need to predict a realis type, labeling the event as Actual, Generic or Other.

We treat these two tasks independently but in a similar manner, leading in both cases to a component based on neural networks as the core technique to benefit from their ability to automatically learn relevant features. We consider these tasks for every word of a target document. Since our two components are close in their architecture, we will focus on the event type classification first, then detail the specificity of the realis part.

2 Event Detection and Classification

The aim of this component is to find every event mention in an input document. To do this, each target document is first tokenized and split into sentences. We then apply syntactic parsing to these sentences to extract both constituents and dependencies. All this pre-processing is performed by the Stanford CoreNLP tool (Manning et al., 2014). Inspired from (Nguyen et al., 2016b), we only consider single token mentions. Our system treats the event detection task as a classification task for every token in the document. The choice of this single-token view was made considering that, since multi-token triggers represent only 1.9% and 3% of the triggers in the test set of TAC 2015 and TAC 2016 respectively, its impact on global performances was expected as limited. Moreover, it allows the introduction of a positional feature that greatly improves the performance of the neural network systems. Since we predict a class for every token of the document, which includes tokens that are not triggers, we introduce an "OTHER" class denoting that the current token is not a trigger of any type.

2.1 Encoding

We consider every token in every sentence as a trigger candidate that we represent by a fixed-size context window centered on this token. We trim longer sentences and use a special padding token if the context window falls out of the sentence boundaries. Let t_0 be the trigger candidate and w the size of the window. $\mathbf{t} = [t_{-w}, t_{-w+1}, \ldots, t_0, \ldots, t_{w-1}, t_w]$ denotes the context window centered on t_0. We then turn this vector of token indexes into a matrix $\mathbf{X} = [\mathbf{x_{-w}}, \mathbf{x_{-w+1}}, \ldots, \mathbf{x_o}, \ldots, \mathbf{x_{w-1}}, \mathbf{x_w}]$ by re-

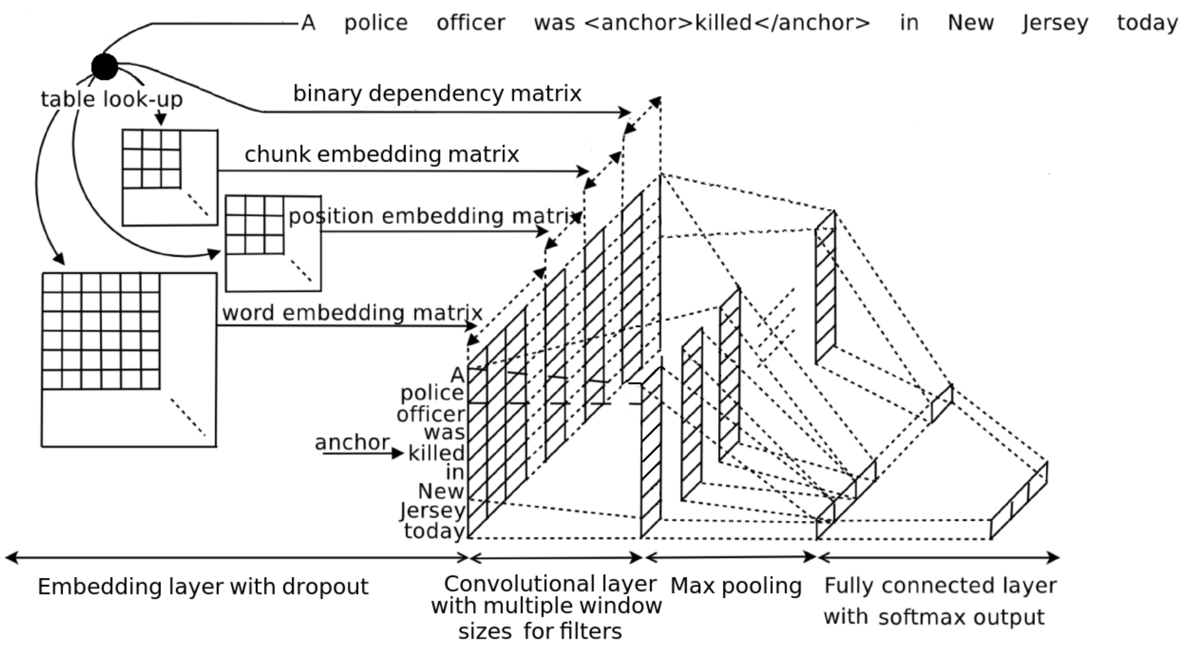
A police officer was <anchor>killed</anchor> in New Jersey today
table look-up
binary dependency matrix
chunk embedding matrix
position embedding matrix
word embedding matrix
A
police
officer
was
anchor
killed
in
New
Jersey
today
Embedding layer with dropout
Convolutional layer with multiple window sizes for filters
Max pooling
Fully connected layer with softmax output

Figure 1: Illustration of our convolutional neural network (CNN) model, adapted from (Nguyen et al., 2016b)

placing every token index t_i by its real-valued representation $\mathbf{x_i} = [e_i, d_i, g_i, q_i]$ combining the following representations:

Word embedding $\mathbf{e_i}$ This embedding corresponds to the word t_i and can be pre-trained on a large corpus to provide semantic and syntactic information about the word (Mikolov et al., 2013).

Distance embedding $\mathbf{d_i}$ This vector encodes the relative distance i between the current word t_i and the trigger candidate t_0.

Dependency vector $\mathbf{g_i}$ The size of this binary vector corresponds to the number of different basic dependencies provided by the Stanford CoreNLP tool. We set the value to 1 for each dimension if there exists a dependency of this kind between t_i and t_0.

Chunk embedding $\mathbf{q_i}$ This vector embeds the chunking information of the current token, with BIO annotations.

The input matrix $\mathbf{X}$ is then fed into a convolutional neural network (CNN) model that learns to map this representation to an event type (or the absence of event).

2.2 Token-based Convolutional Neural Network

We use a classical convolutional neural network, illustrated by Figure 1, which passes the input matrix $\mathbf{X}$ through a dropout layer then a convolutional layer and a maxpooling layer. This way, an abstract representation of the trigger candidate automatically is learned by the network. This representation is finally fed to a dense layer with one output per class and a softmax applied to it. This allows us to compute the probability distribution of the event classes for the trigger candidate.

3 Training schemes

Event type Since we predict the trigger type for every token in the document and because the vast majority of tokens are not triggers, we observe that more than 90% of the tokens are assigned the “OTHER" label. We suspect that this imbalance might hinder the training. In (Marino, 2016), the author introduces a taxonomic curriculum transfer scheme. In a classification problem where there exists different granularities of labels, taxonomic curriculum learning consists in first training a neural network on the coarse-grained classification task

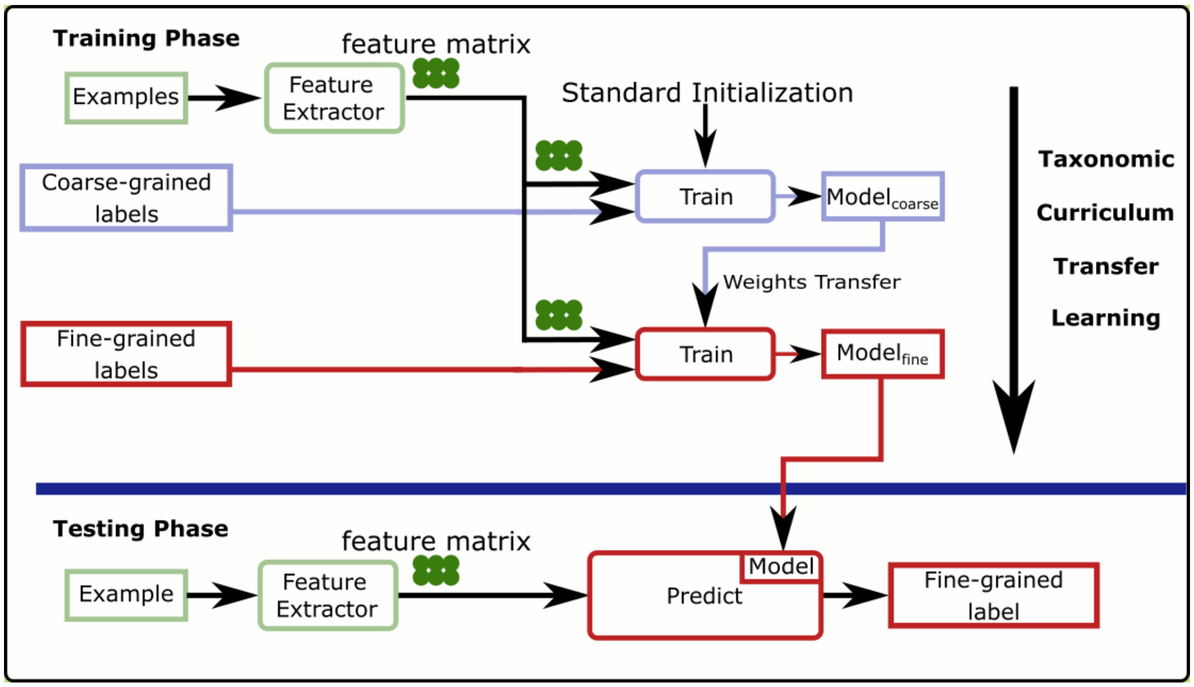

Figure 2: Illustration of the taxonomic curriculum transfer learning scheme. In our case, the coarse-grained labels are either the "isTrigger" binary labels or the 9 TYPE labels and the fine-grained ones are the final 39 type labels

to learn common low-level features and then using this network parameters to initialize a new network, trained on more fine-grained labels, as shown in Figure 2. This process can be repeated multiple times along the taxonomy. The author shows that this type of training scheme can be useful when there are fewer fine-grained examples than coarse-grained examples. In a similar manner, we want to study if heavily unbalanced fine-grained labels (type) can benefit from more balanced coarse-grained labels.

To study the influence of this imbalance, we have considered three alternative training schemes:

CNN$_{fromScratch}$ Our baseline consists in directly learning the 39 fine-grained classes (38 types + the OTHER class) of the final task (run *lvic_event1*).

CNN$_{fromTYPE}$ We first train a CNN$_{TYPE}$ to predict 10 coarse-grained classes (9 TYPE + the OTHER class), then use it to initialize the weights of a new CNN$_{fromTYPE}$ which is retrained on the 39 classes (run *lvic_event2*).

CNN$_{fromDetect}$ We infer a binary label "isTrigger" for every token, set to True if the token is a trigger of any type and to False otherwise. We then train a binary classifier CNN$_{detect}$ to predict this coarse-grained label and finally use it to initialize the weights of a new CNN$_{fromDetect}$ which is retrained on the 39 fine-grained type labels (run *lvic_event3*).

The Rich ERE annotations include 38 event subtypes but the evaluation restricts to a subset of 18 subtypes. We learn to predict 39 classes (the 38 subtypes in Rich ERE plus the “OTHER" class) but we only retain predictions for the 18 target subtypes for the evaluation.

Realis classification For the realis task, we use the same architecture and the same training scheme as for CNN$_{fromScratch}$, this time using the realis labels. For the evaluation, we predicted the realis label for every token in the dataset but only returned the realis label when we predicted a trigger during the previous step.

4 Parameters and Resources

We adopt the same parameters for all networks. More precisely, we use 150 filters for each window size in {2,3,4,5} with a $\tanh$ non linearity for the convolution step. We use the 300 dimension embeddings pre-trained on Google News provided with *word2vec* and optimize them during training. The POS embeddings have 50 dimensions. Our neural networks are trained using stochastic gradient descent, the batch size is set to 50 and we use the Adagrad optimizer with gradient clipping (the threshold is set to 3). We determine the number of epochs for training through early stopping during development, averaged across 10 runs. For the final model, we include the development set to our training set and train it with the same number of epochs.

As mentioned before, this architecture is inspired by the 2016 best system (Nguyen et al., 2016b). The main differences are:

- we remove the dropout on the dense layer and add dropout on the input layer, with a dropout rate $\rho = 0.8$;
- we use classical convolution instead of non-consecutive convolution;
- we initialize our word embeddings with Google News CBOW embeddings instead of the concatenated CBOW;
- we use Adagrad instead of Adadelta.

We did not adopt the non-consecutive convolution because the traditional convolution layer is less computationally intensive and yielded better results on last year evaluation dataset as shown in Table 3. Concerning the other differences, we also report the results of some preliminary experiments on development data in the next section.

5 Evaluation

We submitted three runs to the Event Nugget evaluation, based on the same architecture but differing according to the training scheme used, as detailed in Section 3. *lvic_event1* is our baseline. *lvic_event2* and *lvic_event3* refer to the networks whose weights were initialized from training respectively on event type and trigger detection. The realis prediction is the same for all runs.

5.1 Datasets

We relied on four datasets for the development and the test of our systems:

- dataset A: the union of DEFT_RICH_ERE_R2_V2 (LDC2015E68), DEFT_RICH_ERE_V2 (LDC2015E29) and TAC 2015 training data (from LDC2017E02), containing events from the 38 Rich ERE subtypes;
- dataset B: TAC 2015 evaluation data (from LDC2017E02) containing events from the 38 Rich ERE subtypes;
- dataset C: TAC 2016 evaluation data (from LDC2017E02) containing events from the same 19 subtypes evaluated this year;
- dataset D: TAC 2017 evaluation data containing events from 19 subtypes.

All scores reported in this section are micro-f1 scores generated by the official TAC 2017 scorer, unless stated otherwise.

5.2 Evaluation of our systems on the TAC 2017 dataset

Table 1 presents our official results on the 2017 test set while training on all other corpora. The best results are obtained with our baseline system, which tends to invalidate our hypothesis that a preliminary training with more balanced coarse-grained classes could help the system in identifying common generic features. However, we can note that our baseline clearly outperforms the median score for each task and is close to the best scores, especially for the realis and the realis+type task. The two other reported scores are from the first and third best models on the final task. Both these systems use ensembling methods, which are known to increase performance in almost any situation. On the other hand, our system ranks second while only using a single model.

Table 2 is a breakdown of our performance in event subtype classification, providing the intermediate *TYPE* score and the performance of CNN$_{TYPE}$ and CNN$_{detect}$, the networks trained to initialize *lvic_event2* and *lvic_event3* respectively. The pre-training step seems to have a negative impact on the performance of the final system: the performance of *lvic_event1* for *plain* and *TYPE* is higher than the performance of the network specifically trained for these tasks. Similarly, CNN$_{TYPE}$ performs better than CNN$_{detect}$ on the detection (plain) task. This could indicate that no generic feature can properly discriminate triggers from non triggers among the tokens. In fact, there might be as

Run	plain	**t**(ype)	**r**(ealis)	**t+r**
lvic_event1	59.95	50.14	47.48	39.28
lvic_event2	59.57	49.66	47.21	38.96
lvic_event3	58.32	49.22	46.23	38.59
1st ranked model	59.16	48.6	48.33	39.73
3rd ranked model	67.27	56.19	47.42	39.24
Median	56.92	47.95	41.69	33.77

Table 1: Results of our three runs on D (TAC 2017 evaluation data) trained on A+B+C. "median" report the median scores for all participants on each subtask independently. The two other scores reported are the best and third systems, ranked on the final **t+r** score

Run	plain	TYPE	type
lvic_event1	59.95	57.29	50.14
lvic_event2	59.57	56.79	49.66
lvic_event3	58.32	55.64	49.22
CNN$_{TYPE}$	59.27	55.47	–
CNN$_{detect}$	54.09	–	–

Table 2: Breakdown of the results of our various systems on the TAC 2017 test set

much difference among the mentions of the different event subtypes than between these mentions and the mentions of the negative class. The same observation can be done about the differences inside and between TYPE. Globally, it means that the *plain* detection task and the *TYPE* classification task are both complementary to the *type* final task and should be treated as joint tasks.

Event Argument (EA) is also a possible joint task likely to provide complementary information. First, we assume that detecting and classifying arguments would probably lead to more discriminative features for the Event Nugget task. Furthermore, the Rich ERE annotation scheme can assign multiple labels to the same mention while our system only assigns one label to every token. For example, in the sentence below, “purchases" is the trigger of three pairs of event types: for each year underlined in the example, there are both a Transfer-Ownership and a Transfer-Money event that differ only by a “thing" argument played by “firearm".

> *About 19.6 million background checks were carried out for firearm **purchases** in 2012, a 19-percent rise from 2011 and more than in any year since 1998, according to the FBI.*

In order to fully extract triggers in this sentence, we would need to identify that all the mentioned years play a role in the events. Performing the EN and EA tasks jointly is necessary in such a case and would allow us to generate multiple event triggers of the same type based on the different years. However, we would still miss the co-occurring event types. To solve this problem, we would also need to introduce multi-label prediction to make the neural network able to learn co-occurrences between event types. The most common co-occurring pairs of event types are Transfer-Ownership/Transfer-Money, Conflict-Attack/Life-Injure and Conflict-Attack/Life-Die.

Thus, dealing with multi-label prediction for the event type and taking into account the arguments of the event could lead to better performances on this dataset. In fact, naively duplicating every mention about a Life-Die event with a Conflict-Attack event mention leads to a 0.18 improvement of the final "type + realis" score.

5.3 Comparison to the 2016 systems

In order to compare our system to the TAC 2016 best systems, we trained our best system (*lvic_event1*) on corpora A and B only and tested it on the TAC 2016 evaluation dataset (C). We report the performance of last year's 3 best systems (Mitamura et al., 2016) and our system in Table 3. It is interesting to note that our neural network model outperforms the best TAC 2016 systems while it is a simpler version of the top 1 system, which made use of more sophisticated embeddings than ours, that is to say Concatenated Continuous Bag-Of-Word (CCBOW) and Non Consecutive convolutional neural network (NCNN).

The difference in the dropout place and the choice about pre-trained embeddings are based on preliminary experiments presented here. The systems were trained on the corpus A and we report the f1-mesure of our system on the corpus B without the official scorer. Note that the experiments about embeddings

Run	plain	**t**(ype)	**r**(ealis)	**t+r**
lvic_event1	57.15	47.36	45.99	37.89
Top 1 in 2016	53.84	44.37	42.68	35.24
Top 2 in 2016	54.59	46.99	39.78	33.58
Top 3 in 2016	49.39	44.47	36.96	33.1

Table 3: Results of our three runs on C (TAC 2016 evaluation data), with training on A+B

embedding type	type
GoogleNews	**58.8**
Structured-SkipGram	58.4
Cwindow	55.3
CBOW	55.9
fastText	56.9

Table 4: Study of the influence of different pre-trained embeddings, evaluated on corpus B and trained on corpus A

are anterior to the dropout experiment, which explains the non-matching scores.

Embeddings Concerning the embeddings, we compare the 300 dimension CBOW embeddings pre-trained on GoogleNews provided with *word2vec*[1] to the 300 dimension fastText (Bojanowski et al., 2017) embeddings pre-trained on Wikipedia[2]. We also trained on the Gigaword corpus 400 dimension embeddings for the CBOW model and 400 dimension embeddings according to the two models described in (Ling et al., 2015): Structured-Skipgram and Cwindow. All these embeddings were built with the best parameters found by (Baroni et al., 2014; Levy et al., 2015). The Cwindow embeddings, as a concatenated version of the CBOW model trained on the Gigaword corpus, seem theoretically comparable to the embeddings used in (Nguyen et al., 2016b) and introduced in (Nguyen et al., 2016a). However, it is unclear whether the two implementations are similar or not. We present in Table 4 the performance reached by our network with these different embeddings as initialization. The embeddings were adapted during training. It is clear from Table 4 that the GoogleNews version gives the best performance while (Nguyen et al., 2016a) relied on embeddings built from the Gigaword corpus.

[1]https://code.google.com/archive/p/word2vec/

[2]https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md

dropout input	dropout last layer	type
False	False	57.43
False	True	*56.79*
True	False	**60.98**
True	True	57.36

Table 5: Study of the influence of different dropout configurations, evaluated on corpus B and trained on corpus A

Dropout configuration Because of the high variation of results between runs, we investigated possible ways to stabilize the network and found out that adding a dropout to the input layer and removing it from the last layer not only reduced the variance but also reached higher performance, as shown by Table 5.

5.4 Comparison between 2016 and 2017 evaluation set

We can also see by comparing the results in Table 3 to those in Table 1 that for each subtask, our system performs better on the 2017 test set than on the 2016 development set. The 2016 results shown in Table 3 are produced by our development systems while the performance on the 2017 dataset (Table 1) are produced by our final systems. Since the final systems are trained on more data, by including our

Run	plain	**t**(ype)	**r**(ealis)	**t+r**
lvic_event1	59.24	48.86	47.12	38.39
lvic_event2	58.47	48.20	46.40	37.90
lvic_event3	56.40	47.01	44.82	36.90

Table 6: Results of our three runs on D (TAC 2017 evaluation data) with TAC 2016 evaluation data (C) not added to the training dataset

Corpus	plain	**t**(ype)	**r**(ealis)	**t+r**
A+B	57.15	47.36	45.99	37.89
A+B+D	57.01	47.70	45.77	38.11

Table 7: Influence of the addition of D to the training set while testing on C (TAC 2016 evaluation data)

previous validation set in the training set, this could partly explain the performance gap. In order to validate this hypothesis, we present in Table 6 the results obtained on the 2017 test set without including the 2016 evaluation data in the training set. While we see that the performance is indeed decreased (which confirms the impact of the added training data), the results are still better than the results from the 2016 test set. This suggests that the 2017 test set may be a bit easier than the 2016 test set.

To further explore the difference between the two datasets and confirm this hypothesis, Table 7 provides the reciprocal results, *i.e.* training *lvic_event1* on dataset A, B and D and evaluating on C. Similarly to the runs we submitted, we first train a network on A+B and use D as a development set to determine the number of epochs to train the final system. Table 7 confirms that adding training data has a small impact on results compared to the difficulty of the test dataset.

Finally, we investigate whether the transfer from the coarse-grained task to the fine-grained one is easier in this case, where the training and test data have the same number of classes. To answer the question, we provide in Table 8 the breakdown of our different networks trained on A and evaluated on B, with both datasets annotated for the 38 Rich ERE types. Once again, we reach the same global conclusion as previously even if in that case, CNN_{TYPE} performs similarly to *lvic_event1* and even slightly better for TYPE.

6 Conclusion

In this article, we presented an overview of the CEA LIST system for the KBP 2017 Event Nugget task. Although our system is a simpler version of the convolutional neural network that ranked first last year, our best model still ranks second on the plain, realis and the type+realis subtasks and third on the type subtask. Our preliminary experiments show that a careful tuning of our convolutional neural network can lead to significant improvements, especially concerning the application of dropout. We trained and evaluated our model exclusively for English but it can be easily adapted to other languages since it mainly relies on word embeddings, the influence of syntactic parsing being small.

Run	plain	TYPE	type
lvic_event1	68.86	66.72	60.98
lvic_event2	68.41	66.41	60.81
lvic_event3	66.63	64.84	60.32
CNN_{TYPE}	68.86	66.93	–
CNN_{detect}	63.67	–	–

Table 8: Analysis of our various training schemes on TAC 2015 test (corpus B) set while training on corpus A

References

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. 2014. Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 238–247, Baltimore, Maryland, June. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving Distributional Similarity with Lessons Learned from Word Embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Two/Too Simple Adaptations of Word2Vec for Syntax Problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1299–1304, Denver, Colorado, May–June. Association for Computational Linguistics.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland, June. Association for Computational Linguistics.

Joseph Marino. 2016. Taxonomic Curriculum Learning for Fine-Grained Object Recognition. Preprint at: `https://joelouismarino.github.io/research/taxonomic_curriculum_learning.pdf`.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Advances in Neural Information Processing Systems*.

Teruko Mitamura, Zhengzhong Liu, and Eduard Hovy. 2016. Overview of TAC-KBP 2016 Event Nugget Track. In *Text Analysis Conference*.

Thien Huu Nguyen, Kyunghyun Cho, and Ralph Grishman. 2016a. Joint Event Extraction via Recurrent Neural Networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 300–309, San Diego, California, June. Association for Computational Linguistics.

Thien Huu Nguyen, Ralph Grishman, and Adam Meyers. 2016b. New York University 2016 System for KBP Event Nugget: A Deep Learning Approach. In *Text Analysis Conference*.

Zhiyi Song, Ann Bies, Stephanie Strassel, Tom Riese, Justin Mott, Joe Ellis, Jonathan Wright, Seth Kulick, Neville Ryant, and Xiaoyi Ma. 2015. From Light to Rich ERE: Annotation of Entities, Relations, and Events. In *Proceedings of the 3rd Workshop on EVENTS at the NAACL-HLT*, pages 89–98.