Combining Neural Networks and Log-linear Models to Improve Relation Extraction

Thien Huu Nguyen
Computer Science Department
New York University
New York, NY 10003 USA
thien@cs.nyu.edu

Ralph Grishman
Computer Science Department
New York University
New York, NY 10003 USA
grishman@cs.nyu.edu

Abstract

The last decade has witnessed the success of traditional feature-based methods for exploiting the discrete structures such as words or lexical patterns to extract relations from text. Recently, convolutional and recurrent neural networks have been shown to capture effective hidden structures within sentences via continuous representations, thereby significantly advancing the performance of relation extraction. The advantage of convolutional neural networks is their capacity to generalize the consecutive k-grams in the sentences while recurrent neural networks are effective to encode long range sentence context. This paper proposes to combine the traditional feature-based method, the convolutional and recurrent neural networks to simultaneously benefit from their advantages. Our systematic evaluation of different network architectures and combination methods demonstrates the effectiveness of this approach and results in the state-of-the-art performance on the ACE 2005 and SemEval datasets.

1 Introduction

We study the relation extraction (RE) problem, one of the important problem of information extraction and natural language processing (NLP). Given two entity mentions in a sentence (relation mentions), we need to identify the semantic relationship (if any) between the two entity mentions. One example is the recognition of the *Located* relation between "*He*" and "*Texas*" in the sentence "*He lives in Texas*".

The two methods dominating RE research in the last decade are the feature-based method [Boschee *et al.*, 2005; Zhou *et al.*, 2005; Jiang and Zhai, 2007; Chan and Roth, 2010; Sun *et al.*, 2011] and the kernel-based method [Zelenko *et al.*, 2003; Culotta and Sorensen, 2004; Bunescu and Mooney, 2005; Plank and Moschitti, 2013]. This research extensively studied the leverage of linguistic analysis and knowledge resources to construct the feature representations, involving the combination of *discrete* properties such as lexicon, syntax, and gazetteers. Although these approaches are able to exploit the symbolic (discrete) structures within relation mentions, they also suffer from the difficulty to generalize over the unseen words [Gormley *et al.*, 2015], motivating some very recent work on employing the *continuous* representations of words (word embeddings) to do RE. The most popular method involves neural networks (NNs) that effectively learn hidden and continuous structures of relation mentions from such word embeddings, thus achieving the top performance for RE [Zeng *et al.*, 2014; dos Santos *et al.*, 2015; Xu *et al.*, 2015].

The NN research for relation extraction and classification has centered around two main network architectures: convolutional neural networks (CNNs) [dos Santos *et al.*, 2015; Zeng *et al.*, 2015] and recursive/recurrent neural networks [Socher *et al.*, 2012; Xu *et al.*, 2015]. The distinction between convolutional neural networks and recurrent neural networks (RNNs) for RE is that the former aim to generalize the short and consecutive context (i.e, the k-grams) of the relation mentions [Nguyen and Grishman, 2015a; Lei *et al.*, 2015] while the latter adaptively accumulate the context information in the whole sentence via the memory units, thereby encoding the long and possibly non-consecutive patterns for RE [Hochreiter and Schmidhuber, 1997]. Consequently, the traditional feature-based method (i.e, the *log-linear* or MaxEnt model with hand-crafted and discrete features), the CNNs and the RNNs tend to focus on different angles for RE. Guided by this intuition, in this work, we propose to combine the three models to further improve the performance of RE.

While the architecture design of CNNs for RE is quite established due to the extensive studies in the last couple of years, the application of RNNs to RE is only very recent and the optimal designs of RNNs for RE are still a subject of ongoing research. In this work, we first perform a systematic exploration of various network architectures to seek the best RNN model for RE. In the next step, we extensively study different methods to assemble the log-linear model, CNNs and RNNs for RE, leading to the combined models that yield the state-of-the-art performance on the ACE 2005 and SemEval datasets. To the best of our knowledge, this is the first work to systematically examine the RNN architectures as well as combine them with CNNs and the traditional feature-based approach for RE.

2 Models

Relation mentions consist of sentences marked with two entity mentions of interest. In this paper, we examine two different representations for the sentences in RE: (i) the standard representation, called SEQ that takes all the words in the sen-

tences into account and (ii) the dependency representation, called DEP that only considers the words along the dependency paths between the two entity mention heads of the sentences. In the following, unless indicated specifically, all the statements about the sentences hold for both representations SEQ and DEP.

Throughout this paper, for convenience, we assume that the input sentences of the relation mentions have the same fixed length n. This can be achieved by setting n to the length of the longest input sentences and padding the shorter sentences with a special token. Let $W = w_1 w_2 \dots w_n$ be the input sentence of some relation mention, where w_i is the i-th word in the sentence. Also, let w_{i_1} and w_{i_2} be the two heads of the two entity mentions of interest. In order to prepare the relation mention for neural networks, we first transform each word w_i into a real-valued vector x_i using the concatenation of the following seven vectors, motivated by the previous research on feature analysis for RE [Zhou *et al.*, 2005; Sun *et al.*, 2011; Zeng *et al.*, 2014; Gormley *et al.*, 2015]:

- The real-valued word embedding vector e_i of w_i, obtained by looking up the word embedding table E.

- The real-valued distance embedding vectors d_{i_1}, d_{i_2} to encode the relative distances $i - i_1$ and $i - i_2$ of w_i to the two entity heads of interest w_{i_1} and w_{i_2}: $d_{i_1} = D[i - i_1]$, $d_{i_2} = D[i - i_2]$ where D is the distance embedding table (initialized randomly). The objective is to inform the networks the positions of the two entity mentions for relation prediction.

- The real-valued embedding vectors for entity types t_i and chunks q_i to embed the entity type and chunking information for w_i. These vectors are generated by looking up the entity type and chunk embedding tables (also initialized randomly) (i.e, T and Q respectively) for the entity type ent_i and chunking label $chunk_i$ of w_i: $t_i = T[ent_i]$, $q_i = Q[chunk_i]$.

- The binary vector p_i with one dimension to indicate whether the word w_i is on the dependency path between w_{i_1} and w_{i_2} or not.

- The binary vector g_i whose dimensions correspond to the possible relations between words in the dependency trees. The value at a dimension of g_i is only set to 1 if there exists one edge of the corresponding relation connected to w_i in the dependency tree.

The transformation from the word w_i to the vector $x_i = [e_i, d_{i_1}, d_{i_2}, t_i, q_i, p_i, g_i]$ essentially converts the relation mention with the input sentence W into a real-valued matrix $X = [x_1, x_2, \dots, x_n]$, to be used by the neural networks presented below.

2.1 The Separate Models

We describe two typical NN architectures for RE underlying the combined models in this work.

The Convolutional Neural Networks

In CNNs [Kalchbrenner *et al.*, 2014; Nguyen and Grishman, 2015a], given a window size of k, we have a set of c_k feature maps (filters). Each feature map $\mathbf{f}$ is a weight matrix $\mathbf{f} = [\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_k]$ where $\mathbf{f}_i$ is a vector to be learnt during training as the model parameters. The core of CNNs is the application of the convolutional operator on the input matrix X and the filter matrix $\mathbf{f}$ to produce a score sequence (also called the hidden vector) $\mathbf{s^f} = [s_1^\mathbf{f}, s_2^\mathbf{f}, \dots, s_{n-k+1}^\mathbf{f}]$, interpreted as a more abstract representation of the input matrix X:

$$s_i^\mathbf{f} = g(\sum_{j=0}^{k-1} \mathbf{f}_{j+1} x_{j+i} + b)$$

where b is a bias term and g is the $\tanh$ function.

In the next step, we further abstract the scores in $\mathbf{s^f}$ by aggregating it via the $\max$ function to obtain the max-pooling score $\mathbf{s}_{max}^\mathbf{f}$. We then repeat this process for all the c_k feature maps with different window sizes k to generate a vector of the max-pooling scores. In the final step, we pass this vector into some standard multilayer neural network, followed by a softmax layer to produce the probabilistic distribution $p_\mathrm{C}(y|X)$ over the possible relation classes y in the prediction task.

The Recurrent Neural Networks

In RNNs, we consider the input matrix $X = [x_1, x_2, \dots, x_n]$ as a sequence of column vectors indexed from 1 to n. At each step i, we compute the hidden vector h_i from the current input vector x_i and the previous hidden vector h_{i-1} using the non-linear transformation function Φ: $h_i = \Phi(x_i, h_{i-1})$.

This recurrent computation can be done via three different directional mechanisms: (i) the forward mechanism that recurs from 1 to n and generate the forward hidden vector sequence: $R(x_1, x_2, \dots, x_n) = h_1, h_2, \dots, h_n$, (ii) the backward mechanism that runs RNNs from n to 1 and results in the backward hidden vector sequence $R(x_n, x_{n-1}, \dots, x_1) = h'_n, h'_{n-1}, \dots, h'_1$[1], and (iii) the bidirectional mechanism that performs RNNs in both directions to produce the forward and backward hidden vector sequences, and then concatenate them at each position to generate the new hidden vector sequence $h_1^b, h_2^b, \dots, h_n^b$: $h_i^b = [h_i, h'_i]$.

Given the hidden vector sequence $h_1, h_2, \dots, h_n$ obtained from one of the three mechanisms above, we study the following two strategies to generate the representation vector v^R for the initial relation mention. Note that this representation vector can be again fed into some standard multilayer neural network with a softmax layer in the end, resulting in the distribution $p_\mathrm{R}(y|X)$ for the RNN models:

- The *HEAD* strategy: In this strategy, v^R is the concatenation of the hidden vectors at the positions of the two entity mention heads of interest: $v^R = [h_{i_1}, h_{i_2}]$. This is motivated by the importance of the two mention heads in RE [Sun *et al.*, 2011; Nguyen and Grishman, 2014].

- The *MAX* strategy: This strategy is similar to our max-pooling mechanism in CNNs. In particular, v_R is obtained by taking the maximum along each dimension of the hidden vectors $h_1, h_2, \dots, h_n$. The idea is to further abstract the hidden vectors by retaining only the most important feature in each dimension.

Regarding the non-linear function, the simplest form of Φ in the literature considers it as a one-layer feed-forward neural network, called FF: $h_i = FF(x_i, h_{i-1}) = \phi(Ux_i + Vh_{i-1})$ where ϕ is the *sigmoid* function, U and V are weight matrices. Unfortunately, the application of FF 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]. In this work, we use a variant

[1]The initial hidden vectors are set to the zero vector.

of the LSTM, called the *Gated Recurrent Units* by Cho et al. [2014], (GRU). GRU is shown to be simpler than LSTM in terms of computation but still achieves comparable performance [Józefowicz *et al.*, 2015].

2.2 The Combined Models

We first present three different methods to assemble CNNs and RNNs: ensembling, stacking and voting, to be investigated in this work. The combination of the neural networks with the log-linear model would be discussed in the next section.

Ensembling

In this method, we first run some CNN and RNN in Section 2.1 over the input matrix X to gather the corresponding distributions $p_C(y|X)$ and $p_R(y|X)$. We then combine the CNN and RNN by multiplying their distributions (element-wise): $p_{\text{ensemble}}(y|X) = \frac{1}{Z} p_{\text{C}}(y|X) p_{\text{R}}(y|X)$ (Z is a normalization constant).

Stacking

The overall architecture of the stacking method is to use one of the two network architectures (i.e, CNNs and RNNs) to generalize the hidden vectors of the other architecture. The expectation is that we can learn more effective features for RE via such a deeper architecture by alternating between the local and global representations provided by CNNs and RNNs.

We examine two variants for this method. The first variant, called *RNN-CNN*, applies the CNN model in Section 2.1 on the hidden vector sequence generated by some RNN in Section 2.1 to perform RE. The second variant, called *CNN-RNN*, on the other hand, utilize the CNN model to acquire the hidden vector sequence, that is, in turn, fed as the input into some RNN for RE. For the second variant, as the length of the hidden vector $\mathbf{s^f} = [s_1^{\mathbf{f}}, s_2^{\mathbf{f}}, \ldots, s_{n-k+1}^{\mathbf{f}}]$ in the CNN model depends on the specified window size k for the feature map $\mathbf{f}$, we need to pad the input matrix X with $\lfloor \frac{k}{2} \rfloor$ zero column vectors on both sides to ensure the same fixed length n for all the hidden vectors: $\mathbf{s^f} = [s_1^{\mathbf{f}}, s_2^{\mathbf{f}}, \ldots, s_n^{\mathbf{f}}]$. Besides, we need to re-arrange the scores in the hidden vectors from different feature maps of the CNN so they are grouped according to the positions in the sentence, thus being compatible with the input requirement of RNNs.

Voting

Instead of integrating CNNs and RNNs at the model level as the two previous methods, the voting method makes decisions for a relation mention X by voting the individual decisions of the different models. While there are several voting schemes in the literature, for this work, we employ the simplest scheme of majority voting. If there is more than one relation class receiving the highest number of votes, the relation class returned by a model and having the highest probability would be chosen.

2.3 The Hybrid Models

In order to further improve the RE performance of models above, we investigate the integration of these neural network models with the traditional log-linear model that relies on various linguistic features from the past research on RE [Zhou *et al.*, 2005; Sun *et al.*, 2011; Gormley *et al.*, 2015]. Specifically, in such integration models (called *the hybrid models*), the relation class distribution is obtained from the element-wise multiplication between the distributions of the neural network models and the log-linear model. Let us take the ensembling model in Section 2.2 as an example. The corresponding hybrid model in this case would be: $p_{\text{hybrid}}(y|X) = \frac{1}{Z} p_{\text{C}}(y|X) p_{\text{R}}(y|X) p_{\text{login}}(y|X)$, assuming $p_{\text{login}}(y|X)$ is the distribution of the log-linear model and Z is the normalization constant. The parameters of the log-linear model are learnt jointly with the parameters of the neural networks.

Hypothesis: Let S be the set of relation mentions correctly predicted by some neural network model in some dataset (the coverage set). The introduction of the log-linear model into this neural network model essentially changes the coverage set of the network, resulting in the new coverage set S' that might or might not subsume the original set S. In this work, we hypothesize that although S and S' overlap, there are still some relation mentions that only belong to either set. Consequently, we propose to implement a majority voting system (called the *hybrid-voting system*) on the outputs of the network and its corresponding hybrid model to enhance both models.

Note that the voting models in Section 2.2 involve the voting on two models (i.e, CNN and RNN). In order to integrate the log-linear model into such voting models, we first augment the separate CNN and RNN models with the log-linear model before we perform the voting procedure on the resulting models. Finally, the corresponding *hybrid-voting* systems would involve the voting on four models (CNN, hybrid CNN, RNN and hybrid RNN).

2.4 Training

We train the models by minimizing the negative log-likelihood function using the stochastic gradient descent algorithm with shuffled mini-batches and the AdaDelta update rule [Zeiler, 2012]. The gradients are computed via back-propagation while regularization is executed by a dropout on the hidden vectors before the the multilayer neural networks [Hinton *et al.*, 2012]. During training, besides the weight matrices, we also optimize the embedding tables E, D, T, Q to achieve the optimal state. Finally, we rescale the weights whose l_2-norms exceed a hyperparameter [Nguyen and Grishman, 2015a].

3 Experiments

3.1 Resources and Parameters

For all the experiments below, we utilize the pre-trained word embeddings `word2vec` with 300 dimensions from Mikolov et al. [2013] to initialize the word embedding table E. The parameters for CNNs and training the networks are inherited from the previous studies, i.e, the window size set for feature maps = $\{2, 3, 4, 5\}$, 150 feature maps for each window size, 50 dimensions for all the embedding tables (except the word embedding table E), the dropout rate = 0.5, the mini-batch size = 50, the hyperparameter for the l_2 norms = 3 [Nguyen and Grishman, 2015a]. Regarding RNNs, we employ 300 units in the hidden layers.

3.2 Dataset

We evaluate our models on two datasets: the ACE 2005 dataset for relation extraction and the SemEval-2010 Task 8

dataset [Hendrickx *et al.*, 2010] for relation classification.

The ACE 2005 corpus comes with 6 different domains: broadcast conversation (bc), broadcast news (bn), telephone conversation (cts), newswire (nw), usenet (un) and weblogs (wl). Following the common practice of domain adaptation research on this dataset [Plank and Moschitti, 2013; Nguyen and Grishman, 2014; Nguyen *et al.*, 2015c; Gormley *et al.*, 2015], we use news (the union of bn and nw) as the training data, half of bc as the development set and the remainder (cts, wl and the other half of bc) as the test data. Note that we are using the data prepared by Gormley et. al [2015], thus utilizing the same data split on bc as well as the same data processing and NLP toolkits. The total number of relations in the training set is 43,497[2]. We employ the BIO annotation scheme to capture the chunking information for words in the sentences and only mark the entity types of the two entity mention heads (obtained from human annotation) for this dataset.

The SemEval dataset concerns the relation classification task that aims to determine the relation type (or no relation) between two entities in sentences. In order to make it compatible with the previous research [Socher *et al.*, 2012; Gormley *et al.*, 2015], for this dataset, besides the word embeddings and the distance embeddings, we apply the name tagging, part of speech tagging and WordNet features (inherited from Socher et al. [2012] and encoded by the real-valued vectors for each word). The other settings are also adopted from the past studies [Socher *et al.*, 2012; Xu *et al.*, 2015].

3.3 RNN Architectures

This section evaluates the performance of various RNN architectures for RE on the ACE 2005 development set. In particular, we compare different design combinations of the following four factors: (i) sentence representations (i.e, SEQ or DEP), (ii) transformation functions Φ (i.e, FF or GRU), (iii) the strategies to employ the hidden vector sequence for RE (i.e, HEAD or MAX), and (iv) the directions to run RNNs (i.e, forward ($\rightarrow$), backward ($\leftarrow$) or bidirectional ($\rightleftharpoons$)). Table 1 presents the results.

Systems			DEP	SEQ
FF	HEAD	$\rightleftharpoons$	60.78	63.22
		$\rightarrow$	55.55	60.05
		$\leftarrow$	57.69	58.54
	MAX	$\rightleftharpoons$	50.00	51.22
		$\rightarrow$	52.08	53.96
		$\leftarrow$	45.07	33.50
GRU	HEAD	$\rightleftharpoons$	63.32	63.23
		$\rightarrow$	63.69	62.77
		$\leftarrow$	61.57	62.55
	MAX	$\rightleftharpoons$	60.96	**64.24**
		$\rightarrow$	61.97	**64.59**
		$\leftarrow$	61.56	**64.30**

Table 1: Performance (F1 scores) of RNNs on the dev set

The main conclusions include:

[2]It was an error in Gormley et al. [2015] that reported 43,518 total relations in the training set. The authors acknowledged this error.

(i) Assuming the same choices for the other three corresponding factors, GRU is more effective than FF, SEQ is better than DEP most of the time, and HEAD outperforms MAX (except in the case where SEQ and GRU are applied) for RE with RNNs. Note that the outperformance of SEQ over DEP can be partly explained by the domination of the relation mentions with short distances between the two entity mentions in the ACE 2005 dataset.

(ii) Regarding the direction mechanisms, the bidirectional mechanism achieves the best performance for the HEAD strategy while the forward direction is the best mechanism for the MAX strategy. This can be partly explained by the lack of past or future context information in the HEAD strategy when we follow the backward or forward direction respectively.

The best performance corresponds to the application of the SEQ representation, the GRU function and the MAX strategy that would be used in all the RNN models below. We call such RNN models with the forward, backward and bidirectional mechanism **FORWARD**, **BACKWARD** and **BIDIRECT** respectively. We also apply the SEQ representation for the CNN model (called CNN) in the following experiments for consistency.

3.4 Evaluating the Combined Models

Model	P	R	F1
BIDIRECT	69.16	59.97	64.24
FORWARD	69.33	60.45	64.59
BACKWARD	65.60	63.05	64.30
CNN	68.35	59.16	63.42
Ensembling			
CNN-BIDIRECT	71.22	54.13	61.51
CNN-FORWARD	66.19	59.64	62.75
CNN-BACKWARD	65.09	60.13	62.51
Stacking			
CNN-BIDIRECT	66.55	59.97	63.09
CNN-FORWARD	69.46	63.05	**66.10**
CNN-BACKWARD	72.58	58.35	64.69
BIDIRECT-CNN	65.63	61.59	63.55
FORWARD-CNN	73.13	58.67	65.11
BACKWARD-CNN	67.60	58.51	62.73
Voting			
CNN-BIDIRECT	71.08	60.94	**65.62**
CNN-FORWARD	70.38	59.32	64.38
CNN-BACKWARD	69.78	61.75	**65.52**

Table 2: Performance of the Combination Methods

We evaluate the combination methods for CNNs and RNNs presented in Section 2.2. In particular, for each method, we examine three models that are combined from one of the three RNN models FORWARD, BACKWARD, BIDIRECT and the CNN model. For instance, in the stacking method, the three combined models corresponding to the *RNN-CNN* variant are FORWARD-CNN, BACKWARD-CNN, BIDIRECT-CNN while the three combined models corresponding to the *CNN-RNN* variant are CNN-FORWARD, CNN-BACKWARD, CNN-BIDIRECT. The notations for the other methods are self-explained. The model performance on the development set is given in Table 2 that also includes the performance of the separate models (i.e, CNN, FORWARD, BACKWARD, BIDIRECT) for convenient comparison.

The first observation is that the ensembling method is not an effective way to combine CNNs and RNNs as its perfor-

Model	Neural Networks			Hybrid Models			Hybrid-Voting Models		
	P	R	F1	P	R	F1	P	R	F1
CNN	68.35	59.16	63.42	66.44	64.51	65.46	69.07	63.70	66.27
BIDIRECT	69.16	59.97	64.24	68.04	59.00	63.19	71.13	60.29	65.26
FORWARD	69.33	60.45	64.59	66.11	63.86	64.96	72.69	61.26	66.49
BACKWARD	65.60	63.05	64.30	66.03	62.07	63.99	71.56	63.21	67.13
Combined Models									
VOTE-BIDIRECT	71.08	60.94	65.62	69.24	62.40	65.64	71.30	62.40	**66.55**
STACK-FORWARD	69.46	63.05	66.10	65.93	68.07	66.99	69.32	66.29	**67.77**
VOTE-BACKWARD	69.78	61.75	65.52	67.30	63.05	65.10	70.79	64.02	**67.23**

Table 3: Performance of the Hybrid Models on the ACE 2005 Development Set

System	bc			cts			wl			
	P	R	F1	P	R	F1	P	R	F1	Ave
The State-of-the-art Systems										
FCM	66.56	57.86	61.90	65.62	44.35	52.93	57.80	44.62	50.36	55.06
Hybrid FCM	74.39	55.35	63.48	74.53	45.01	56.12	65.63	47.59	55.17	58.26
Separate Systems										
Log-Linear	68.44	50.07	57.83	73.62	41.57	53.14	60.40	47.31	53.06	54.68
CNN	65.62	61.06	63.26	65.92	48.12	55.63	54.14	53.68	53.91	57.60
BIDIRECT	65.23	61.06	63.07	66.15	49.26	56.47	55.91	51.56	53.65	57.73
FORWARD	63.64	59.39	61.44	60.12	50.57	54.93	55.54	54.67	55.10	57.16
BACKWARD	60.44	61.2	60.82	58.20	54.01	56.03	51.03	52.55	51.78	56.21
Hybrid-Voting Systems										
VOTE-BIDIRECT	70.40	63.84	**66.96**†	66.74	49.92	57.12†	59.24	54.96	57.02†	60.37
STACK-FORWARD	65.75	66.48	66.11†	63.58	51.72	57.04†	56.35	57.22	56.78†	59.98
VOTE-BACKWARD	69.57	63.28	66.28†	65.91	52.21	**58.26**†	58.81	55.81	**57.27**†	**60.60**

Table 4: Comparison to the State of the art on the ACE 2005 Dataset. The cells marked with †designates the models that are significantly better than the other neural network models ($\rho < 0.05$) on the corresponding domains.

mance is worse than the separate models. Second, regarding the stacking method, the best way to combine CNNs and RNNs in this framework is to assemble the CNN model and the FORWARD model. In fact, the combination of the CNN and FORWARD models helps to improve the performance of the separate models in both variants of this method (referring to the models CNN-FORWARD and FORWARD-CNN). Finally, the voting method is also helpful as it outperforms the separate models with the CNN-BIDIRECT and CNN-BACKWARD combinations.

For the following experiments, we would only focus on the three best combined models in this section, i.e, the CNN-FORWARD model in the stacking method (called **STACK-FORWARD**) and the CNN-BIDIRECT and CNN-BACKWARD models in the voting methods (called **VOTE-BIDIRECT** and **VOTE-BACKWARD** respectively).

3.5 Evaluating the Hybrid Models

This section investigates the hybrid and *hybrid-voting* models (Section 2.3) to see if they can further improve the performance of the neural network models. In particular, we evaluate the separate models: CNN, BIDIRECT, FORWARD, BACKWARD, and the combined models: STACK-FORWARD, VOTE-BIDIRECT and VOTE-BACKWARD when they are augmented with the traditional log-linear model (the hybrid models). Besides, in order to verify the hypothesis in Section 2.3, we also test the corresponding *hybrid-voting* models. The experimental results are shown in Table 3. There are three main conclusions:

(i) For all the models in columns "*Neural Networks*", "*Hybrid Models*" and "*Hybrid-Voting Models*", we see that the combined models outperform their corresponding separate models (only except the hybrid model of VOTE-BACKWARD), thereby further confirming the benefits of the combined models.

(ii) Comparing columns "*Neural Networks*" and "*Hybrid Models*", we find that the traditional log-linear model significantly helps the CNN model. The effects on the other models are not clear.

(iii) More interestingly, for all the neural networks being examined (either separate or combined), the corresponding *hybrid-voting* systems substantially improve both the neural network models as well as the corresponding hybrid models, testifying to the hypothesis about the *hybrid-voting* approach in Section 2.3. Note that the simpler voting systems on three models: the log-linear model, the CNN model and some RNN model (i.e, either BIDIRECT, FORWARD or BACKWARD) produce worse performance than the *hybrid-voting* methods (the respective performance is 66.13%, 65.27%, and 65.96%).

3.6 Comparing to the State of the art

The state-of-the-art system on the ACE 2005 for the unseen domains has been the feature-rich compositional embedding model (FCM) and the hybrid FCM model from Gormley et al. [2015]. In this section, we compare the proposed *hybrid-voting* systems with these state-of-the-art systems on the test domains `bc`, `cts`, `wl`. Table 4 reports the results. For completeness, we also include the performance of the log-linear model and the separate models CNN, BIDIRECT, FORWARD, BACKWARD, serving as the other baselines for this work.

Classifier	F
SVM [Hendrickx *et al.*, 2010]	82.2
RNN [Socher *et al.*, 2012]	77.6
MVRNN [Socher *et al.*, 2012]	82.4
CNN [Zeng *et al.*, 2014]	82.7
CR-CNN [dos Santos *et al.*, 2015]	**84.1**†
FCM [Gormley *et al.*, 2015]	83.0
Hybrid FCM [Gormley *et al.*, 2015]	83.4
DepNN [Liu *et al.*, 2015]	83.6
SDP-LSTM [Xu *et al.*, 2015]	83.7
Systems in this work	
CNN	83.5
BIDIRECT	81.8
FORWARD	81.9
BACKWARD	82.4
VOTE-BIDIRECT	**84.1**
STACK-FORWARD	83.4
VOTE-BACKWARD	**84.1**

Table 5: Performance of Relation Classification Systems. The "†" refers to special treatment of the `Other` class.

From the table, we see that although the separate neural networks outperform the FCM model across domains, they are still worse than the hybrid FCM model due to the introduction of the log-linear model into FCM. However, when the networks are combined and integrated with the log-linear model, they (the *hybrid-voting* systems) become significantly better than the FCM models across all domains (up to 2% improvement on the average absolute F score), *yielding the state-of-the-art performance for the unseen domains in this dataset*.

3.7 Relation Classification Experiments

We further evaluate the proposed systems for the relation classification task on the SemEval dataset. Table 5 presents the performance of the seprate models, the proposed systems as well as the other representative systems on this task. The most important observation is that the *hybrid-voting* systems VOTE-BIDIRECT and VOTE-BACKWARD achieve the state-of-the-art performance for this dataset, further highlighting their benefit for relation classification. The *hybrid-voting* STACK-FORWARD system performs less effectively in this case, possibly due to the small size of the SemEval dataset that is not sufficient to training such a deep model.

3.8 Analysis

In order to better understand why the combination of CNNs and RNNs outperforms the individual networks, we evaluate the performance breakdown per relation for the CNN and BIDIRECT models. The results on the development set of the ACE 2005 dataset are provided in Table 6.

One of the main insights is that although CNN and BIDIRECT have comparable overall performance, their recalls on individual relations are very divergent. In particular, BIDIRECT has much better recall for the PHYS relation while the recalls of CNN are significantly better for the ART, ORG-AFF and GEN-AFF relations. A closer investigation reveals two facts: (i) the PHYS relation mentions that are only correctly predicted by BIDIRECT involve long distances between two entity mentions, such as the PHYS relation between "*Some*" (a person entity) and "*desert*" (a location entity) in the following sentence: "*Some of the 40,000 British troops are kicking up a lot of dust in the Iraqi desert making sure that nothing is left behind them that could hurt them.*", and (ii) the ART, ORG-AFF, GEN-AFF relation mentions only correctly predicted by CNN contain patterns between the two entity mentions that are short but meaningful enough to decide the relation classes, such as "*The Iraqi unit in possession of those guns*" (the ART relation between "*unit*" and "*guns*"), or "*the al Qaeda chief operations officer*" (the ORG-AFF relation between "*al Qaeda*" and "*officer*"). The failure of CNN on the PHYS relation mentions with long distances originates from its mechanism to model short and consecutive k-grams (up to length 5 in our case), causing difficulty in capturing long and/or non-consecutive patterns. BIDIRECT, on the other hand, fails to predict the short (but expressive enough) patterns for ART, ORG-AFF, GEN-AFF because it involves the hidden vectors that only model the context words outside the short patterns, potentially introducing unnecessary and noisy information into the max-pooling scores for prediction. Eventually, the combination of RNNs and CNNs helps to compensate for the drawbacks of each model.

Relation Class	CNN			BIDIRECT		
	P	R	F1	P	R	F1
PHYS	66.7	34.7	45.7	57.4	50.9	54.0
PART-WHOLE	68.6	67.8	68.2	74.4	70.1	72.2
ART	64.2	51.2	57.0	68.6	41.7	51.9
ORG-AFF	70.2	83.0	76.0	79.3	76.1	77.7
PER-SOC	71.1	59.3	64.6	69.6	59.3	64.0
GEN-AFF	65.9	55.1	60.0	59.0	46.9	52.3
all	68.4	59.2	63.4	69.2	60.0	64.2

Table 6: The Performance Breakdown per Relation for CNN and BIDIRECT on the development set.

4 Related Work

For relation extraction/classification, most work on neural networks has focused on the relation classification task. In particular, Socher et al. [2012] study the recursive NNs that recur over the tree structures while Xu et al. [2015] investigate recurrent NNs. Regarding CNNs, Zeng et al. [2014] examine CNNs via the sequential representation of sentences, dos Santos et al. [2015] explore a ranking loss function with data cleaning while Zeng et al. [2015] propose dynamic pooling and multi-instance learning. For RE, Yu et al. [2015] and Gormley et al. [2015] work on the feature-rich compositional embedding models. Finally, the only work that combines NN architectures is due to Liu et al. [2015] but it only focuses on the stacking of the recursive NNs and CNNs for relation classification.

5 Conclusion

We investigate different methods to combine CNNs, RNNs as well as the hybrid models to integrate the log-linear model into the NNs. The experimental results demonstrate that the stacking and majority voting between CNNs, RNNs and their corresponding hybrid models are the best combination methods. We achieve the state-of-the-art performance for both relation extraction and relation classification. In the future, we plan to further evaluate the proposed methods on the other tasks such as event extraction and slot filling in the Knowledge Base Population (KBP) evaluation.

References

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

[Boschee *et al.*, 2005] Elizabeth Boschee, Ralph Weischedel, , and Alex Zamanian. Automatic information extraction. In *Proceedings of the International Conference on Intelligence Analysis*, 2005.

[Bunescu and Mooney, 2005] Razvan Bunescu and Raymond Mooney. A shortest path dependency kernel for relation extraction. In *HLT-EMNLP*, 2005.

[Chan and Roth, 2010] Yee S. Chan and Dan Roth. Exploiting background knowledge for relation extraction. In *COLING*, 2010.

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

[Culotta and Sorensen, 2004] Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *ACL*, 2004.

[dos Santos *et al.*, 2015] Cicero dos Santos, Bing Xiang, and Bowen Zhou. Classifying relations by ranking with convolutional neural networks. In *ACL-IJCNLP*, 2015.

[Gormley *et al.*, 2015] Matthew R. Gormley, Mo Yu, and Mark Dredze. Improved relation extraction with feature-rich compositional embedding models. In *EMNLP*, 2015.

[Hendrickx *et al.*, 2010] Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *SemEval*, 2010.

[Hinton *et al.*, 2012] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. In *CoRR, abs/1207.0580.*, 2012.

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

[Jiang and Zhai, 2007] Jing Jiang and ChengXiang Zhai. A systematic exploration of the feature space for relation extraction. In *NAACL-HLT*, 2007.

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

[Kalchbrenner *et al.*, 2014] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *ACL*, 2014.

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

[Lei *et al.*, 2015] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Molding cnns for text: non-linear, non-consecutive convolutions. In *EMNLP*, 2015.

[Liu *et al.*, 2015] Yang Liu, Furu Wei, Sujian Li, Heng Ji, Ming Zhou, and Houfeng WANG. A dependency-based neural network for relation classification. In *ACL-IJCNLP*, 2015.

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

[Nguyen and Grishman, 2014] Thien Huu Nguyen and Ralph Grishman. Employing word representations and regularization for domain adaptation of relation extraction. In *ACL*, 2014.

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

[Nguyen *et al.*, 2015c] Thien Huu Nguyen, Barbara Plank, and Ralph Grishman. Semantic representations for domain adaptation: A case study on the tree kernel-based method for relation extraction. In *ACL-IJCNLP*, 2015c.

[Plank and Moschitti, 2013] Barbara Plank and Alessandro Moschitti. Embedding semantic similarity in tree kernels for domain adaptation of relation extraction. In *ACL*, 2013.

[Socher *et al.*, 2012] Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In *EMNLP*, 2012.

[Sun *et al.*, 2011] Ang Sun, Ralph Grishman, and Satoshi Sekine. Semi-supervised relation extraction with large-scale word clustering. In *ACL*, 2011.

[Xu *et al.*, 2015] Yan Xu, Lili Mou, Ge Li, Yunchuan Chen, Hao Peng, and Zhi Jin. Classifying relations via long short term memory networks along shortest dependency paths. In *EMNLP*, 2015.

[Yu *et al.*, 2015] Mo Yu, Matthew R. Gormley, and Mark Dredze. Combining word embeddings and feature embeddings for fine-grained relation extraction. In *NAACL*, 2015.

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

[Zelenko *et al.*, 2003] Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. Exploring various knowledge in relation extraction. In *Journal of Machine Learning Research*, 2003.

[Zeng *et al.*, 2014] Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. Relation classification via convolutional deep neural network. In *COLING*, 2014.

[Zeng *et al.*, 2015] Daojian Zeng, Kang Liu, Yubo Chen, and Jun Zhao. Distant supervision for relation extraction via piecewise convolutional neural networks. In *EMNLP*, 2015.

[Zhou *et al.*, 2005] GuoDong Zhou, Jian Su, Jie Zhang, and Min Zhang. Exploring various knowledge in relation extraction. In *ACL*, 2005.