CHAIN BASED RNN FOR RELATION CLASSIFICATION

Javid Ebrahimi

Towards Completion of Directed Research Project
Computer Science Department
University of Oregon
December 2014

Committee:
Dejing Dou, Chair
Daniel Lowd
Reza Rejaie

Abstract

Deep Learning is a new area of Machine Learning research, which mainly addresses the problem of time consuming, often incomplete feature engineering in machine learning. Recursive Neural Network (RNN) is a new deep learning architecture that has been highly successful in several Natural Language Processing tasks.

We propose a new approach for relation classification, using an RNN, based on the shortest path between two entities in the dependency graph. Most previous works on RNN are based on constituency-based parsing because phrasal nodes in a parse tree can capture compositionality in a sentence. Compared with constituency-based parse trees, dependency graphs can represent the relation more compactly. This is particularly important in sentences with distant entities, where the parse tree spans words that are not relevant to the relation. In such cases RNN cannot be trained effectively in a timely manner. On the other hand, dependency graphs lack phrasal nodes that complicates the application of RNN. In order to tackle this problem, we employ dependency constituent units called *chains*. Further, we devise two methods to incorporate chains into an RNN. The first model uses a fixed tree structure based on a heuristic, while the second one predicts the structure by means of a recursive autoencoder.

Chain based RNN provides a smaller network which performs considerably faster, and achieves better classification results. Experiments on SemEval 2010 relation classification task and SemEval 2013 drug drug interaction task demonstrate the effectiveness of our approach compared with the state-of-the-art models.

Introduction

Relation extraction is the task of finding relations between entities in text, which is useful for several tasks such as information extraction, summarization and question answering [31]. For instance, in the sentence: those "cancers" were caused by radiation "exposures", the two entities have *cause-effect* relation. As reported in detail in [21], the early approaches to the problem involve supervised methods where the models rely on lexical, syntactic and semantic features to classify relations between pairs of entities. While this approach provides good precision and recall, one has to re-train the model for other domains with different target relations and thus it is not scalable to web, where thousands of (previously-unseen) relations exist [1]. To address this problem, Open Information Extraction has been proposed which does not require supervision. In these systems [1, 17, 9], templates based on lexical, syntactic, POS and dependency features are extracted. While these patterns give good precision, they suffer from low recall [2]. This is because they fail to extract patterns which have not been pre-specified, and thereby are unable to generalize. Another difference between supervised and unsupervised methods is that the latter cannot detect *semantic* relations, in which the relation is stated somewhere in the text not necessarily where entities are mentioned. Self supervision is another approach to relation extraction. Here, the goal is to create a data set without manual labeling [32, 30, 31]. An example of this approach is Kylin [30] which aims to generate new and complete existing Wikipedia infoboxes. The relations are between attributes of the infobox and their corresponding values. For instance: (BornIn, *Nelson Mandela*, *South Afriaca*) is a relation tuple from the infobox of Mandela Wikipedia page. The assumptions is that any sentence that mentions both Nelson Mandela and South Africa in the article, is an instance of BornIn relation and can be used as a positive training example.

Deep Learning [14] is a new area of Machine Learning research which mainly addresses the problem of time consuming, often incomplete feature engineering in machine learning. Moving away from *one-hot* and *distributional representation* of text and introducing vector based representation of words, was the key element in success of deep learning [29]. Some deep learning applications in Human Language Technologies include [6] where convolutional neural network is used for sequence labeling tasks and [18] where recurrent neural network is used for speech recognition. Recursive Neural Network (RNN) has proven to be highly successful in capturing semantic

compositionality in text and has improved the results of several Natural Language Processing tasks. It has been used to predict parse trees [25], classify relations [23, 12, 16] and analyze sentiments in text [27, 8].

All previous applications of RNN to relation extraction are based on constituency-based parsers. These RNNs may span words that do not contribute to the relation. We investigate the incorporation of RNN with dependency parsing that can give a more compact representation of the relation. We do this by using the shortest path between two entities.

Our first contribution is to introduce a compositional account of dependency graphs that can match RNN's recursive nature, and can be applied to relation classification. Our second contribution is to improve the-state-of-the-art RNN results in terms of performance and efficiency. We propose two models: the first model uses a fixed structure in which we build a tree representation of the path with a heuristic. The idea is to start from entities and to recursively combine dependent-head pairs along the shortest path between them. The other model predicts the tree structure by a recursive autoencoder in pre-training, during which we minimize the reconstruction error of the predicted tree. Both of these approaches output a full binary tree that can later be used for the RNN training. Our heuristic based model outperforms the-state-of-the-art methods in classifying relations. Moreover, compared with the constituency-based RNN, our approach is much more efficient by saving up to 70% in running time.

In the next section, we cover the related works on RNN and relation extraction. Later, we will elaborate on our chain based RNN and how proposed models extract features from the chain. Next, we will discuss the details of the learning procedure and finally we will demonstrate our results and conclude the report.

Background and Related Work

At the core of deep learning techniques for NLP lies the vector based word representation, which maps words to an *n-dimensional* space. This approach enables us to have multiple dimensions of similarity and encode the syntactic and semantic features of the words in a compressed numerical format. Having word vectors as parameters, rather than fixed input, makes neural models flexible in finding different word embeddings for separate tasks [6]. Furthermore, through a composition model, one can map phrases and sen-

tences to the same *n-dimensional* space. These word vectors are stacked into a word embedding matrix $L \in \mathbb{R}^{n \times |V|}$ where $|V|$ is the size of the vocabulary. Each word is associated with a vocabulary index k into the embedding matrix which we retrieve the word's vector from.

Numerical operations on these vectors are based on Artificial Neural Networks (ANN). Inspired by the brain, ANNs have a neuron-like behavior as their primary computational unit. The behavior of a neuron is controlled by its input weights θ. Hence, the weights are where the information learned by the neuron is stored. More precisely a neuron uses the weighted sum of its inputs, and squeezes them into the interval [0, 1] using a nonlinearity function like sigmoid:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{1}$$

Feed Forward Neural Network

A feed forward neural network (FFNN) is a type of ANN where the neurons are structured in layers, and only connections to subsequent layers are allowed. Training is achieved by minimizing the network error (E). In order to

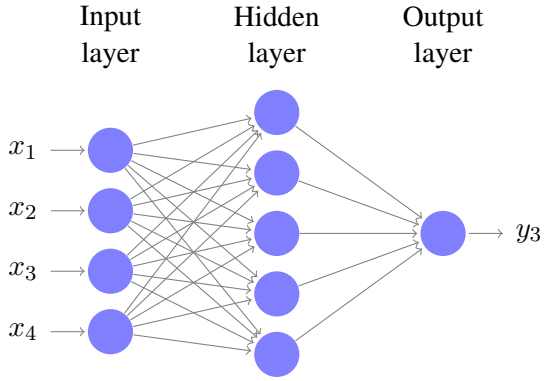

Figure 1: FFNN with four input neurons, one hidden layer, and 1 output neuron. This type of architecture is appropriate for binary classification of some data x $\in \mathbb{R}^4$

minimize this function, the gradient $\frac{\partial E}{\partial \theta}$ needs to be calculated, which can be done using backpropagation. The result of this processes is a set of weights that enables the network to do the desired input-output mapping, as defined

by the training data. An example of FFNN with four input neurons, one hidden layer, and 1 output neuron is shown in Figure 1.

RNN for Relation Classification

A Recursive Neural Network (RNN) is a type of FFNN that can process data through an arbitrary binary tree structure. This is achieved by tying weight across all nodes and restricting the output of each node to have the same dimensionality as its children. The input data is placed in the leaf nodes of the tree, and the structure of this tree is used recursively combine the sibling nodes to the root node. Figure 2 displays an RNN example where each node contains four neurons.

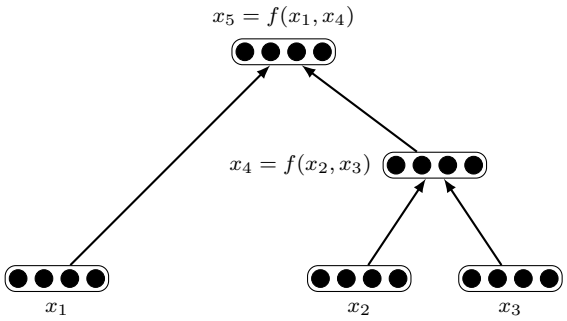

Figure 2: RNN example. Each node has a vector of 4 neurons. At each internal node, a nonlinearity function (tanh or sigmoid) is applied and a parent node in the same 4-dimensional space is created.

The central idea in using RNN for NLP tasks is the Principle of Compositionality which states that:

> the meaning of a complex expression is determined by the meanings of its constituent expressions and the rules used to combine them.

While the initial composition models focused on linear combination of constituents, RNNs provide a new setting to use nonlinear combination of constituents. Binarized parse tree is an elegant recursive structure that matches RNN and can contain nonlinear combination of nodes in its internal nodes.

As an example, in [25], each node in the parse tree is associated with a vector and at each internal node p, there exists a composition function that

takes its input from its children $c_1 \in \mathbb{R}^n$ and $c_2 \in \mathbb{R}^n$.

$$p = f(c_1, c_2) = tanh(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b) \tag{2}$$

The matrix $W \in \mathbb{R}^{n \times 2n}$ is the global composition parameter, b is the bias term and the output of the function $p \in \mathbb{R}^n$ is another vector in the same space of the inputs. Finally, the final vector at the root of the parse tree, computed recursively in a bottom up fashion, represents the *distributed representation* of the sentence.

In [23], Matrix-Vector Recursive Neural Network (MV-RNN) is proposed where instead of using only vectors for words, an additional matrix for each word is used to capture operator semantics in language. This model is tested on SemEval 2010 relation classification task with impressive results. They first find the path in the parse tree between the two entities and apply compositions bottom up. Then, they select the highest node of the path and classify the relationship based on that node's vector as features. Hashimoto et al. [12] follow the same design but introduce a different composition function. They make use of word-POS pairs and include weight matrices based on the phrase categories of the pair.

Socher et al. in [24] independently proposed a dependency based RNN that extracts features from a dependency graph which has major differences from our model.

Dependency Parsing for Information Extraction

Dependency parsing has been used in both supervised and unsupervised relation extraction. Of the former, kernels over dependency graphs are the most popular methods [7, 3]. For example in [3] the kernel function is based on the similarity of the nodes on the shortest path, between two entities in a dependency graph. They discuss intuitions on why using the shortest dependency path for relation extraction makes sense. For example, if entities are the arguments of the same predicate, then the shortest path will pass through the predicate. While the use of dependency parsing has been debated among researchers [1, 15] - especially among OpenIE community due to performance issues - there has been growing interest in using dependency parsing [31, 17, 9]. For instance, WOE [31] and OLLIE [17] both extract patterns of dependency path between candidate entities and use augmentation mechanisms to extract more coherent relations. This rising interest in

dependency parsing over constituency based parsing, is due to advances in dependency parsing [19, 10] with high speed and state-of-the-art accuracy.

Chain based RNN

The constituency-based parse trees distinguish between terminal and non-terminal nodes. The interior or phrasal nodes are labeled by non-terminal categories of the grammar, while the leaf nodes are labeled by terminal categories i.e. words. Figure 3 displays the constituency parse tree of a sentence. This structure matches RNN and can contain nonlinear combination of nodes in its phrasal nodes. While constituency-based parsing seems to be a reasonable choice for compositionality in general and specifically for sentiment analysis, it may not be the best choice for all NLP tasks. In particular, for relation classification, one may prefer to use a structure that encodes more information about the relations between the words in a sentence. To this end, we use dependency-based parsing that provide a one-to-one correspondence between nodes in a dependency graph (DG). In dependency-based

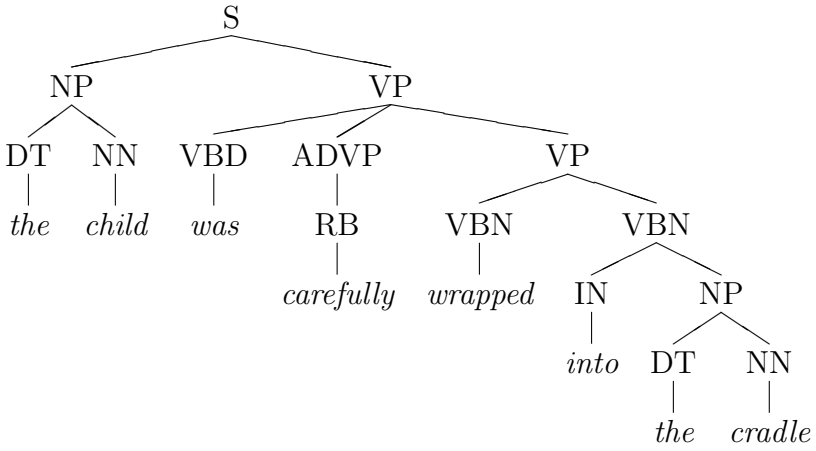

Figure 3: Phrase constituents of: *the child was carefully wrapped into the cradle.*

parsing, the syntactic head of each word in the sentence is found. This creates a DG where the nodes are words and edges are the binary relation between *head* and *dependent*. Dependency graphs are significantly different from constituency parse trees since they lack phrasal nodes. More precisely, the internal nodes where the nonlinear combinations take place, do not exist in DGs.

Thus we must modify the original RNN and present a dependency-based RNN for relation classification. In our experiments, we restrict ourselves to dependency trees where each dependent has only one head. We also use the example in Figure 4 for better illustration; in this example the arguments of the relation are *child* and *cradle*. It is evident that applying compositions

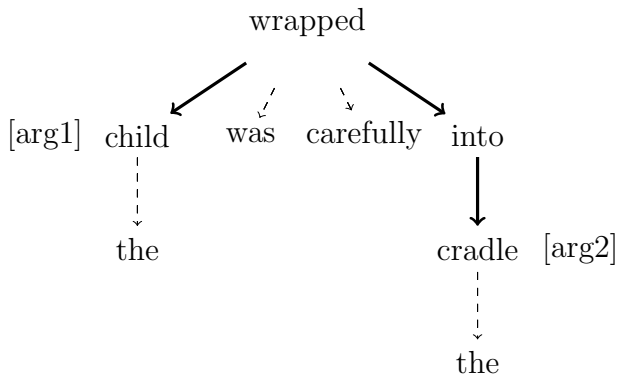

Figure 4: DG of : *the child was carefully wrapped into the cradle.*

on the complete DG is a formidable task and thus we apply compositions on the words on only the shortest path between entities (shown by thick lines). From a linguistics point of view, this type of composition is related to the concept of *chain* or *dependency constituent* unit in DGs [28].

> **Chain**: The words A ... B ... C ... (order irrelevant) form a chain iff A immediately dominates (is the parent of) B and C, or if A immediately dominates B and B immediately dominates C.

Based on this definition, *child wrapped, into cradle, wrapped into cradle, child wrapped into cradle* all qualify as chain while *child was* does not. With all words represented as vectors, we need to find a reduced dimensional representation of the chain in a fixed size. To this end, we transform this chain to a *full binary* tree in which the root of the tree represents the extracted features.

Fixed Tree Structure

We cannot use an off-the-shelf syntax parser to create a tree for the chain, because the chain may not necessarily be a coherent English statement, and

thus parsers fail. We build the tree deterministically using a heuristic. The idea is to start from argument(s) and recursively combine dependent-head pairs to the (common) ancestor i.e., each head is combined with the subtree below itself. In the simplest case: $a \rightarrow b$ results in $p = f(a, b)$. The subtlety of this approach lies in the treatment of words with two dependents. The reason is that the composition matrix depends on the arity and the shape of the tree.

If one of the arguments is an ancestor of the other argument, then every head on the chain has exactly one dependent. If the arguments have a common ancestor, then that particular node has two dependents. The common ancestor only takes part in one composition and the result will be combined with the other subtree's output vector. The order by which we select the dependents to be merged with the parent is based on the position of the dependent relative to the parent in the sentence. The words occurring before the parent precede the ones occurring after the word. Furthermore, the preceding words are prioritized based on their distance from the parent. For our example, *wrapped* is combined with *child* and not *into*. The final tree is depicted in Figure 5. One observation is that discarding the words that are not on the chain e.g., *carefully*, *was*, results in a more compact representation compared with constituency-based parsing which includes all the words between entities. We now prove that a tree built by this algorithm is a full binary tree.

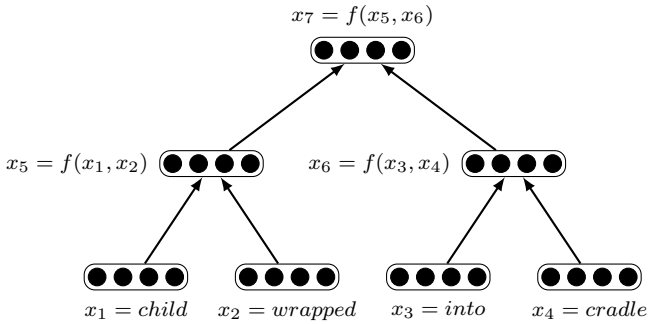

Figure 5: fixed tree example

Lemma: *There is at most one node with exactly two none leaf children in the tree.*

Proof. If one of the arguments is an ancestor of the other argument e.g., arg1 $\rightarrow$ ... $\rightarrow$ arg2, then obviously every head on the chain has exactly one

dependent. Combination of each head and its subtree's output vector results in a full binary node in the tree. If the arguments have a common ancestor p e.g., arg1 $\rightarrow$ ... p ... $\leftarrow$ arg2, then that particular node has two dependents. In this case, the parent is combined with either its left or right subtrees, and its result is combined with the output of the other child. No other head has this property otherwise p is not the common ancestor.

Theorem: *The algorithm produces a full binary tree.*

Proof. It is obvious that the leaves of the tree are the words of the chain. Based on the lemma, there exists one root and all the other internal nodes have exactly two children.

Predicted Tree Structure

Instead of using a deterministic approach to create the tree, we use Recursive Autoencoders (RAE) to find the best representation of the chain. We saw that each parent is the result of applying the composition function on its children as in Eq. 1. Based on this approach children can also be reconstructed from the parent similarly.

$$\begin{bmatrix} c_1^{'} \\ c_2^{'} \end{bmatrix} = tanh(W^{'} \begin{bmatrix} p \\ p \end{bmatrix} + b^{'}) \tag{3}$$

where $W^{'} \in \mathbb{R}^{2n \times 2n}$ and $b^{'} \in \mathbb{R}^{2n}$. Figure 6 depicts an example of an RAE where at each node the inputs are reconstructed. For each pair, we compute the Euclidean distance between the original input and its reconstruction. The aim is to minimize the reconstruction error of all vector pairs of nodes in a tree.

$$E_{rec}(\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}) = \frac{1}{2} \left| \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} - \begin{bmatrix} c_1^{'} \\ c_2^{'} \end{bmatrix} \right|^2 \tag{4}$$

We define $A(x)$ as the set of all possible trees that can be built from a chain. Further, $T(y)$ returns the set of all non-terminal nodes in a tree. Using the reconstruction error of Eq. 3, we can compute the reconstruction error of input x by Eq. 4.

$$E(x) = \arg \min_{y \in A(x)} \sum_{s \in T(y)} E_{rec}(\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}_s) \tag{5}$$

This model is similar to [26] with some modifications in implementation. They use a semi supervised method where the objective function is a weighted

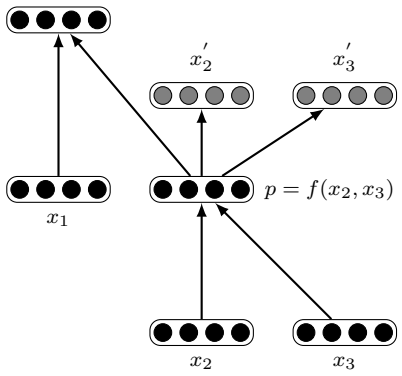

Figure 6: autoencoder example

sum of the supervised and unsupervised error. Our model is a pipeline where first, during pre-training, the unsupervised autoencoder predicts the structure of RNN by minimizing $E(x)$ and then during training the supervised objective function, cross entropy error, is minimized. We also found out that keeping word vectors fixed during backpropagation, results in better accuracy during training.

To predict the structure of the tree, we use a greedy algorithm [26]. It takes pairs of neighboring vectors, sets them as potential children of a chain and gives them as input to the autoencoder. For each word pair, the potential parent node and the resulting reconstruction error are saved. The pair with the lowest reconstruction error is chosen and then those words are replaced with the computed representation. We repeat the process until we have one representation on top of the tree. The predicted tree is then used for the supervised training. To optimize $E(x)$, L-BFGS is used and derivatives are computed by backpropagation through structure, the details of which is described in the next section.

Learning

The vector on top of the tree represents the *distributed representation* of the chain. This vector encodes features representing the relation between two arguments. To predict the label of the relation, a softmax classifier is added on top of the tree i.e., $y_i = softmax(P_n^T W^{lable})$ where $L \in \mathbb{R}^k$, k is the number of classes, and P_n is the final vector on top of the tree for sentence

n. The objective function is the sum of cross entropy error at all the nodes, for all the sentences in the training set.

$$E(\theta) = -\sum_{n}\sum_{k} t_n^k \log y_n^k + \frac{\lambda}{2} \parallel \theta \parallel^2 \tag{6}$$

The vectors for target, predicted labels, and regularization parameters are denoted by t_n, y_n and λ respectively. Model parameters θ include: W, W^{label} and W_O which denote the composition matrix, the softmax layer weights, and the word vectors respectively. We initialize the word vectors $W_O \in \mathbb{R}^n$ with pre-trained 50-dimensional words from [6] and initialize other parameters by a normal distribution with mean of 0 and standard deviation of 0.01.

Backpropagation through Structure

Derivatives are computed by backpropagation through structure [11] and L-BFGS [4] is used for optimization. The error term associated with each softmax unit can be computed by:

$$\delta^k = y^k - t^k \tag{7}$$

At each node, there is an error term associated with the softmax units as:

$$\delta^i = \sum_k \delta^k W_{ik}^{label} \otimes f'(x_i) \tag{8}$$

where $f'(x_i) = (1 - x_i^2)$ and $\otimes$ is the Hadamard product between the two vectors.
In an RNN, children c_i, receive error terms from parent i:

$$\delta^{c_i \leftarrow i} = \delta^i W_i \otimes f'(x_i) \tag{9}$$

where W_i is the i_{th} $n \times n$ block of the matrix W. In the case of leaves, $f'(x_i) = 1$. Finally, the total derivative for W is the sum of all derivatives at each node t in the tree.

$$\frac{\partial E}{\partial W} = \sum_t \delta^t \begin{bmatrix} x_t^1 \\ x_t^2 \end{bmatrix} \tag{10}$$

where x_t^i represents the i_{th} child.

For the case of autoencoder, delta at each node is the sum of error term from the parent δ^i and from the immediate reconstruction layer δ^i_{RAE}. The total delta at each node can be computed by:

$$\delta^{c_i} = (\delta^i W_i + \delta^i_{RAE}) \otimes f'(x_i) \tag{11}$$

Since we need to compute the derivatives based on both c and $c^{'}$, we have δ^i_{RAE} as: $[c - c^{'} + W^{'}_i(c^{'} - c) \otimes f'(c^{'})]$.

Experiments

In this section we discuss our results on two well used data sets for relation classification. To derive the dependency tree for each sentence, we use arc-eager Malt parser [10]. We set the hyper parameters through a validation set for the first data set and use them for the second data set too. Similar to the previous works, few other features are also included. These features comprise the depth of the tree, distance between entities, three context words, and the type of dependencies in our model. We found that using dependency types inside the composition function as in typed functions worsens the results. Furthermore, while collapsed dependency trees [5] have been preferred for relation extraction in general, our results with general dependencies were better. We also tested two layer RNNs with a hidden layer, but the results of the single layer neurons were not improved.

SemEval-2010 Task 8

This data set consists of 10017 sentences and nine types of relation between nominals [13]. To encourage generalization on noisier data sets in real-world applications, a tenth class "other" is also added to this set. In addition, taking the order of entities into account, the problem will become a 19-class classification. For example: The burst$[_{e1}]$ has been caused by water hammer pressure$[_{e2}]$, denotes a **Cause-Effect(e2,e1)** relation, while: Financial stress$[_{e1}]$ is one of the main causes of divorce$[_{e2}]$, denotes a **Cause-Effect(e1,e2)** relation. Table 1 compares the results of our heuristic based chain RNN (C-RNN) and the autoencoder based one (C-RNN-RAE) with other RNN models and the best system participating [20]. Evaluation of the systems was done by comparing the F-measure of their best runs.

As it can bee seen in Table 1, C-RNN achieves the best results. The baseline RNN, uses a global composition function and $\mathbb{R}^{50}$ vectors for each word. We also use the same number of model parameters. The advantage of our approach is that our models are computationally less expensive compared with other RNN models. MV-RNN uses an additional matrix $\mathbb{R}^{50\times50}$ for each word, resulting in 50 fold increase in the number of model parameters. POS-RNN uses POS word pairs e.g., cause-NN and cause-VB both represent cause with different POS tags. In addition, they use 13 phrase categories whose combinations adds 169 more weight matrices. This results in around 100% increase in model parameters compared to C-RNN.

The best system [20] used SVM with many sets of features. Adding POS tags, WordNet hypernyms and named entity tags (NER) of the two words, helps C-RNN improve their results. We also implemented SDT-RNN [24] which has similar complexity as our model but has significantly lower F-measure.

Method	Fmeasure	Feature sets
RNN	74.8	-
SDT-RNN	75.12	-
MV-RNN	79.1	-
POS-RNN	79.4	-
C-RNN-RAE	78.78	-
C-RNN	**79.68**	-
SVM	82.2	POS, WordNet, Levine classes, PropBank, FrameNet, TextRunner, paraphrases, Google *n*-grams, NormLex-Plus, morphological features, dependency parse features
RNN	77.6	POS, NER, WordNet
MV-RNN	82.4	POS, NER, WordNet
C-RNN	**82.66**	POS, NER, WordNet

Table 1: Results on SemEval 2010 relation classification task with the feature sets used. C-RNN outperforms all RNN based models. By including three extra features, it achieves the state-of-the-art performance.

SemEval-2013 Task 9.b

In this task, the aim is to extract interactions between drug mentions in text. DDI corpus [22] is a semantically annotated corpus of documents describing drug-drug interactions from the DrugBank database and MedLine abstracts on the subject of drug-drug interactions. The corpus consists of 1,017 texts

(784 DrugBank texts and 233 MedLine abstracts) that was manually annotated with a total of 5021 drug-drug interactions. These interactions include *mechanism, effect, advise* and *int*. This data set contains many long sentences with long distances between the entities. It makes the comparison between MV-RNN and C-RNN more interesting.

We used the MV-RNN code, publicly available, to compare with C-RNN on this data set. Since we lack pre-trained vector representation of 2861 terms in this dataset: mostly pharmacological e.g., *Cholestyramine*, *Metoclopramide*, we cannot effectively use C-RNN-RAE.

Relations with long distances between entities are harder to classify. This is illustrated in Figure 7 where MV-RNN and C-RNN are compared. Considering three bins for the distance between two entities, the figure shows what fraction of test instances are misclassified in each bin. Both classifiers make more errors when the distance between entities is longer than 10. The performance of the two classifiers for distances less than 5 is quite similar while C-RNN has the advantage in classifying more relations correctly when the distance increases.

Method	Precision	Recall	Fmeasure
MV-RNN	74.07	65.53	67.84
C-RNN	**75.31**	**66.19**	**68.64**

Table 2: Results on SemEval 2013 Drug Drug Interaction task

Table 2 shows that C-RNN performs better than MV-RNN in classifying relations. Interestingly, chain RNN results in 83% decrease in the number of neurons in average.

Running Time

Dependency graphs can represent the relation more compactly by utilizing only the words on the shortest path between entities. C-RNN uses a sixth of neural computations of MV-RNN. More precisely, there is an 83% decrease in the number of *tanh* evaluations. Consequently, as in Figure 8, C-RNN runs 3.21 and 1.95 times faster for SemEval 2010 and SemEval 2013 respectively.

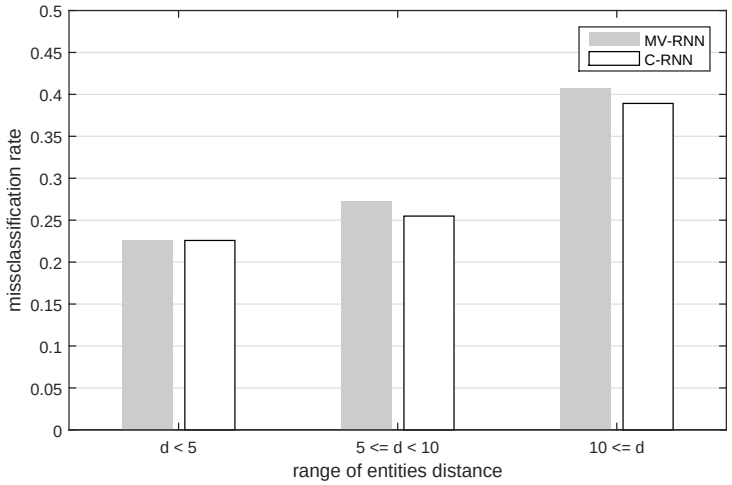

Figure 7: Misclassification based on entities distance in three bins. More erros occur when entities are separated by more than ten words. C-RNN performs better in long distances.

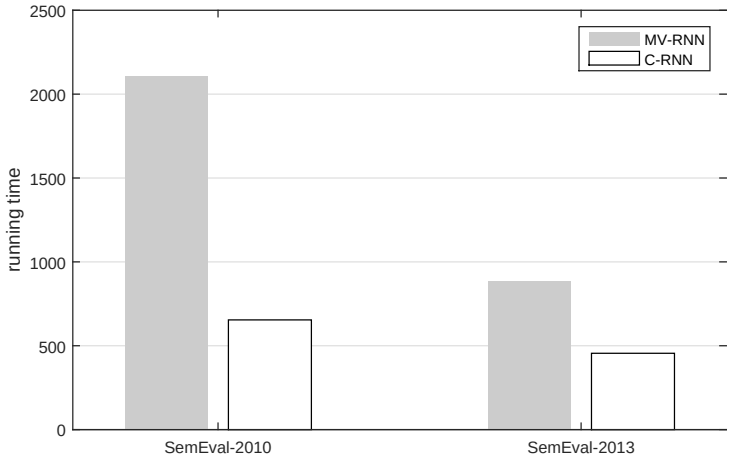

Figure 8: Running time measured by seconds. Experiments were run on a cluster node with 6 core 2.66GHz cpu.

Conclusion and Future Work

Recently, Recursive Neural Network (RNN) has found a wide appeal in Machine Learning community. This deep architecture has been applied in several NLP tasks including relation classification. Most previous works are limited to constituency-based parsing due to compositionality. We devise an RNN

architecture based on a compositional account of dependency graphs. To represent the relation more compactly, the proposed RNN model is based on the shortest path between entities in a dependency graph. The resulting shallow network is superior for supervised learning in terms of speed and accuracy. We improve the classification results of the competing approaches with up to 70% saving in running time.

The current method assumes the named entities to be known in advance. An extension to this work is to jointly extract entities and relations. Further, application of deep learning to self-supervised relation extraction is another direction that we will focus in the future.

Bibliography

[1] Michele Banko, Michael J Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the web. In *Proceedings of IJCAI*, pages 2670–2676, 2007.

[2] Michele Banko and Oren Etzioni. The tradeoffs between open and traditional relation extraction. In *Proceedings of ACL*, pages 28–36, 2008.

[3] Razvan Bunescu and Raymond J. Mooney. A Shortest Path Dependency Kernel for Relation Extraction . In *Proceedings of HLT/EMNLP*, pages 724–731, 2005.

[4] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[5] Marie catherine De Marneffe and Christopher D. Manning. Stanford typed dependencies manual, 2008.

[6] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of ICML*, pages 160–167, 2008.

[7] Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of ACL*, pages 423–429, 2004.

[8] Li Dong, Furu Wei, Ming Zhou, and Ke Xu. Adaptive multi-compositionality for recursive neural models with applications to sentiment analysis. In *Proceedings of AAAI*, pages 1537–1543, 2014.

[9] Pablo Gamallo, Marcos Garcia, and Santiago Fernandez-Lanza. Dependency-based open information extraction. In *Proceedings of the*

Joint Workshop on Unsupervised and Semi-Supervised Learning in NLP, ROBUS-UNSUP, pages 10–18, 2012.

[10] Yoav Goldberg and Joakim Nivre. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING*, pages 959–976, 2012.

[11] Christoph Goller and Andreas Kchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of ICNN*, pages 347–352, 1996.

[12] Kazuma Hashimoto, Makoto Miwa, Yoshimasa Tsuruoka, and Takashi Chikayama. Simple customization of recursive neural networks for semantic relation classification. In *Proceedings of EMNLP*, pages 1372–1376, 2013.

[13] 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 *Proceedings of SemEval*, pages 33–38, 2010.

[14] Geoffrey E. Hinton and Simon Osindero. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[15] Jing Jiang and Chengxiang Zhai. A systematic exploration of the feature space for relation extraction. In *Proceedings of NAACL*, pages 113–120, 2007.

[16] Daniel Khashabi. On the recursive neural networks for relation extraction and entity recognition. Technical report, UIUC, 2013.

[17] Mausam, Michael D Schmitz, Robert E. Bart, Stephen Soderland, and Oren Etzioni. Open Language Learning for Information Extraction . In *Proceedings of EMNLP*, pages 523–534, 2012.

[18] Tomas Mikolov, Martin Karafit, Lukas Burget, Jan Cernock, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proceedings of INTERSPEECH*, pages 1045–1048, 2010.

[19] Joakim Nivre, Johan Hall, and Jens Nilsson. Memory-based dependency parsing. In *Proceedings of CoNLL*, pages 49–56, 2004.

[20] Bryan Rink and Sanda Harabagiu. Utd: Classifying semantic relations by combining lexical and semantic resources. In *Proceedings of SemaEval*, pages 256–259, 2010.

[21] Sunita SaraWagi. Information Extraction. In *Foundations and Trends in Databases,Volume 1 Issue 3*, pages 261–377, 2008.

[22] Isabel Segura-Bedmar, Paloma Martínez, and María Herrero Zazo. Semeval-2013 task 9 : Extraction of drug-drug interactions from biomedical texts (DDIExtraction 2013). In *Proceedings of SemEval*, pages 341–350, 2013.

[23] Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. Semantic Compositionality Through Recursive Matrix-Vector Spaces. In *Proceedings of EMNLP*, pages 1201–1211, 2012.

[24] Richard Socher, Andrej Karpathy, Quoc V. Le, Christopher D. Manning, and Andrew Y. Ng. Grounded compositional semantics for finding and describing images with sentences. *TACL*, 2:207–218, 2014.

[25] Richard Socher, Christopher D. Manning, and Andrew Y. Ng. Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS*, pages 1–9, 2010.

[26] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Proceedings of EMNLP*, pages 151–161, 2011.

[27] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of EMNLP*, pages 1631–1642, 2013.

[28] Osborne Timothy. Beyond the constituent: a dependency grammar analysis of chains. *Folia Linguistica*, 39(3-4):251–297, 2005.

[29] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of ACL*, pages 384–394, 2010.

[30] Fei Wu, Raphael Hoffmann, and Daniel S. Weld. Information extraction from wikipedia: Moving down the long tail. In *Proceedings of KDD*, pages 731–739, 2008.

[31] Fei Wu and Daniel S. Weld. Open information extraction using wikipedia. In *Proceeding of ACL*, pages 118–127, 2010.

[32] Alexander Yates, Michele Banko, Matthew Broadhead, Michael J Cafarella, Oren Etzioni, and Stephen Soderland. TextRunner: Open Information Extraction on the Web. In *NAACL-HLT (Demonstrations)*, pages 25–26, 2007.