Improving Slot Filling Performance with Attentive Neural Networks on Dependency Structures

Lifu Huang[1*], **Avirup Sil**[2], **Heng Ji**[1], **Radu Florian**[2]
[1] Rensselaer Polytechnic Institute, [2] IBM T.J. Watson Research Center
[1]{huangl7,jih}@rpi.edu, [2]{avi,raduf}@us.ibm.com

Abstract

Slot Filling (SF) aims to extract the values of certain types of attributes (or slots, such as person:cities_of_residence) for a given entity from a large collection of source documents. In this paper we propose an effective DNN architecture for SF with the following new strategies: (1). Take a regularized dependency graph instead of a raw sentence as input to DNN, to compress the wide contexts between query and candidate filler; (2). Incorporate two attention mechanisms: local attention learned from query and candidate filler, and global attention learned from external knowledge bases, to guide the model to better select indicative contexts to determine slot type. Experiments show that this framework outperforms state-of-the-art on both relation extraction (16% absolute F-score gain) and slot filling validation for each individual system (up to 8.5% absolute F-score gain).

1 Introduction

The goal of Slot Filling (SF) is to extract pre-defined types of attributes or slots (e.g., *per:cities_of_residence*) for a given query entity from a large collection of documents. The slot filler (attribute value) can be an entity, time expression or value (e.g., *per:charges*). The TAC-KBP slot filling task (Ji et al., 2011a; Surdeanu and Ji, 2014) defined 41 slot types, including 25 types for person and 16 types for organization.

One critical component of slot filling is relation extraction, namely to classify the relation between a pair of query entity and candidate slot filler into one of the 41 types or none. Most previous studies have treated SF in the same way as within-sentence relation extraction tasks in ACE [1] or SemEval (Hendrickx et al., 2009). They created training data based on crowd-sourcing or distant supervision, and then trained a multi-class classifier or multiple binary classifiers for each slot type based on a set of hand-crafted features.

Although Deep Neural Networks (DNN) such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) have achieved state-of-the-art results on within-sentence relation extraction (Zeng et al., 2014; Liu et al., 2015; Santos et al., 2015; Nguyen and Grishman, 2015; Yang et al., 2016; Wang et al., 2016), there are limited studies on SF using DNN. Adel and Schütze (2015) and Adel et al. (2016) exploited DNN for SF but did not achieve comparable results as traditional methods. In this paper we aim to answer the following questions: What is the difference between SF and ACE/SemEval relation extraction task? How can we make DNN work for SF?

We argue that SF is different and more challenging than traditional relation extraction. First, a query and its candidate filler are usually separated by much wider contexts than the entity pairs in traditional relation extraction. As Figure 1 shows, in ACE data, for 70% of relations, two mentions are embedded in each other or separated by at most one word. In contrast, in SF, more than 46% of ⟨*query, filler*⟩ entity pairs are separated by at least 7 words. For example, in the following sentence:

E1. "**Arcandor**$_{query}$ owns a 52-percent stake in Europe's second biggest tourism group Thomas Cook, the Karstadt chain of department stores and iconic shops such as the **KaDeWe**$_{filler}$ in what used to be the commercial heart of West Berlin.",

---
[1]http://www.itl.nist.gov/iad/mig/tests/ace/

Here, **Arcandor** and **KaDeWe** are far separated and it's difficult to determine the slot type as *org:subsidiaries* based on the raw wide contexts.
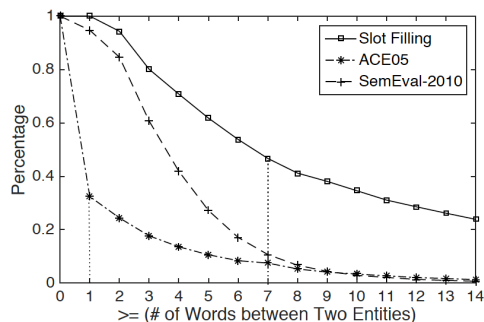

Figure 1: Comparison of the Percentage by the # of Words between two entity mentions in ACE05 and SemEval-2010 Task 8 relations, and between query and slot filler in KBP2013 Slot Filling.

In addition, compared with relations defined in ACE (18 types) and SemEval (9 types), slot types are more fine-grained and heavily rely on indicative contextual words for disambiguation. Yu et al. (2015) and Yu and Ji (2016) demonstrate that many slot types can be specified by contextual *trigger* words. Here, a *trigger* is defined as the word which is related to both the query and candidate filler, and can indicate the type of the target slot. Considering E1 again, *owns* is a trigger word between **Arcandor** and **KaDeWe**, which can indicate the slot type as *org:subsidiaries*. Most previous work manually constructed trigger lists for each slot type. However, for some slot types, the triggers can be implicit and ambiguous.

To address the above challenges, we propose the following new solutions:

- To compress wide contexts, we model the connection of query and candidate filler using dependency structures, and feed dependency graph to DNN. To our knowledge, we are the first to directly take dependency graphs as input to CNN.

- Motivated by the definition of *trigger*, we design two attention mechanisms: a local attention and a global attention using large external knowledge bases (KBs), to better capture implicit clues that indicate slot types.

2 Architecture Overview

Figure 2 illustrates the pipeline of a SF system. Given a query and a source corpus, the system retrieves related documents, identifies candidate fillers (including entities, time, values, and titles), extracts the relation between query and each candidate filler occurring in the same sentence, and finally determines the filler for each slot. Relation extraction plays a vital role in such a SF pipeline. In this work, we focus on relation extraction component and design a neural architecture.

Given a query, a candidate filler, and a sentence, we first construct a regularized dependency graph and take all ⟨*governor, dependent*⟩ word pairs as input to Convolutional Neural Networks (CNN).

Moreover, We design two attention mechanisms: **(1) Local Attention**, which utilizes the concatenation of *Query* and *Candidate Filler* vectors to measure the relatedness of each input bigram (we set filter width as 2) to the specific query and filler. **(2) Global attention**: We use prelearned slot type representations to measure the relatedness of each input bigram with each slot type via a transformation matrix. These two attention mechanisms will guide the pooling step to select the information which is related to query and filler and can indicate slot type.

3 Regularized Dependency Graph based CNN

3.1 Regularized Dependency Graph

Dependency parsing based features, especially the shortest dependency path between two entities, have been proved to be effective to extract the most important information for identifying the relation between two entities (Bunescu and Mooney, 2005; Zhao and Grishman, 2005; GuoDong et al., 2005; Jiang and Zhai, 2007). Several recent studies also explored transforming a dependency path into a sequence and applied Neural Networks to the sequence for relation classification (Liu et al., 2015; Cai et al., 2016; Xu et al., 2015).

However, for SF, the shortest dependency path between query and candidate filler is not always sufficient to infer the slot type due to two reasons. First, the most indicative words may not be included in the path. For example, in the following sentence:

E2. Survivors include two *sons* and *daughters*-in-law, **Troy**$_{filler}$ and Phyllis Perry, **Kenny**$_{query}$ and Donna Perry, all of Bluff City.

the shortest dependency path between **Kenny** and **Troy** is: "Troy ←^{conj} Perry ←^{conj} Kenny", which
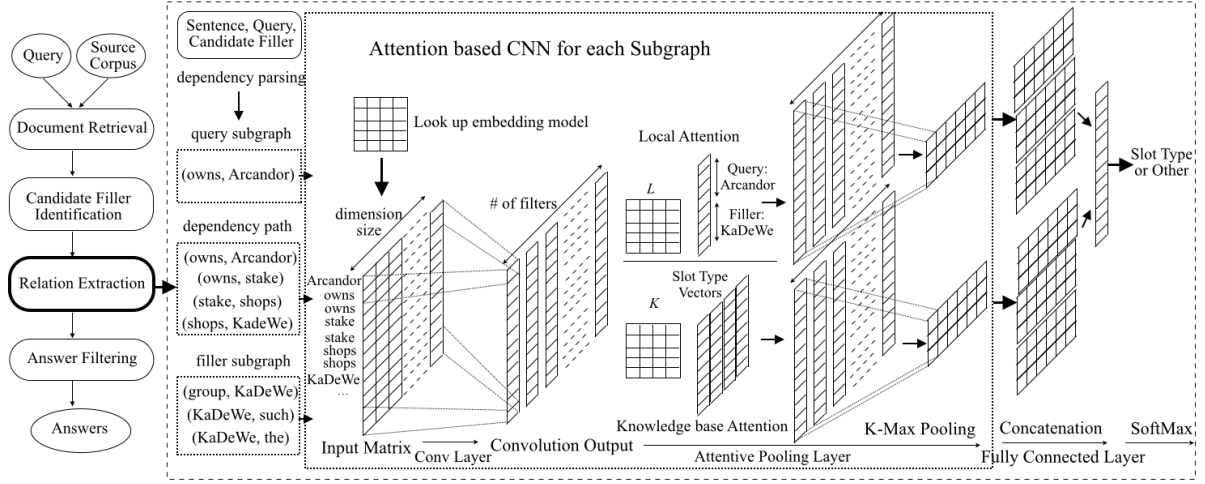
Figure 2: Overview of the Architecture.

does not include the most indicative words: *sons* and *daughters* for their *per:siblings* relation. In addition, the relation between query and candidate filler is also highly related to their entity types, especially for disambiguating slot types such as *per:country_of_birth per:state_of_birth* and *per:city_of_birth*. Entity types can be inferred by enriching query and filler related contexts. For example, in the following sentence:

E3. **Merkel**$_{query}$ died in the southern German city of **Passau**$_{filler}$ in 1967.

we can determine the slot type as *city* related by incorporating rich contexts (e.g., "*city*").

To tackle these problems, we propose to regularize the dependency graph, incorporating the shortest dependency path between query and candidate filler, as well as their rich contextual words.

Given a sentence s including a query q and candidate filler f, we first apply the Stanford Dependency Parser (Manning et al., 2014) to generate all dependent word pairs: ⟨*governor, dependent*⟩, then discover the shortest dependency path between query and candidate filler based on Breadth-First-Search (BFS) algorithm. The regularized dependency graph includes words on the shortest dependency path, as well as words which can be connected to query and filler within n hops. In our experiments, we set $n = 1$. Figure 3 shows the dependency parsing output for E1 mentioned in Section 1, and the regularized dependency graph with the bold circled nodes. We can see that, the most indicative trigger *owns* can be found in both the shortest dependency path of **Arcandor**
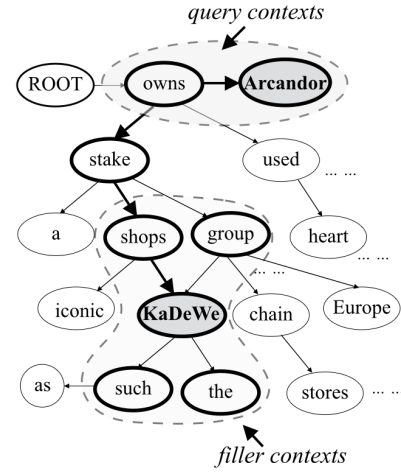

Figure 3: Regularized Dependency Graph for Query *Arcandor* and Filler *KaDeWe* in E1.

and **KaDeWe**, and the context words of **Arcandor**. In addition, the context words, such as *shops*, can also infer the type of candidate filler **KaDeWe** as an Organization.

3.2 Graph based CNN

Previous work (Adel et al., 2016) split an input sentence into three parts based on the positions of the query and candidate filler and generate a feature vector for each part using a shared CNN. To compress the wide contexts, instead of taking the raw sentence directly as input, we split the regularized dependency graph into three parts: query related subgraph, candidate filler related subgraph, and the dependency path between query and filler. Each subgraph will be taken as input to a CNN, as illustrated in Figure 2. We now describe the details

2573

of each part as follows.

Input layer: Each subgraph or path G in the regularized dependency graph is represented as a set of dependent word pairs $G = \{\langle g_1, d_1 \rangle, \langle g_2, d_2 \rangle, ... \langle g_n, d_n \rangle\}$. Here, g_i, d_i denote the *governor* and *dependent* respectively. Each word is represented as a d-dimensional pre-trained vector. For the word which does not exist in the pre-trained embedding model, we assign a random vector for it. Each word pair $\langle g_i, d_i \rangle$ is converted to a $\mathbb{R}^{2 \times d}$ matrix. We concatenate the matrices of all word pairs and get the input matrix $M \in \mathbb{R}^{2n \times d}$.

Convolution layer: For each subgraph, $M \in \mathbb{R}^{2n \times d}$ is the input of the convolution layer, which is a list of linear layers with parameters shared by filtering windows with various size. We set the stride as 2 to obtain all word pairs from the input matrix M. For each word pair $p_{in} = \langle v_{g_i}, v_{d_i} \rangle$, we compute the output vector p_{out} of a convolution layer as:

$$p_{out} = tanh(W \cdot p_{in} + b)$$

where p_{in} is the concatenation of vectors for the words v_{g_i} and v_{d_i}, W denotes the convolution weights, and b is the bias. In our work all three convolution layers share the same W and b.

K-Max Pooling Layer: we follow Adel et al. (2016) and use K-max pooling to select K values for each convolution layer. Later we will incorporate attention mechanisms into K-max pooling.

Fully Connected Layer: After getting the high-level features based on the (attentive) pooling layer for each input subgraph, we flatten and concatenate these three outputs as input to a fully connected layer. This layer connects each input to every single neuron it contains, and learns non-linear combinations based on the whole input.

Output Layer: It takes the output of the fully connected layer as input to a softmax regression function to predict the type. We use negative log-likelihood as loss function to train the parameters.

4 Attention Strategies for SF

4.1 Local Attention

The basic idea of attention mechanism is to assign a weight to each position of a lower layer when computing the representations for an upper layer, so that the model can be attentive to specific regions (Bahdanau et al., 2014). In SF, the indicative words are the most meaningful information that the model should pay attention to. Wang et al. (2016) applied attention from the entities directly to determine the most influential parts in the input sentence. Following the same intuition, we apply the attention from the query and candidate filler to the convolution output instead of the input, to avoid information vanishing during convolution process (Yin et al., 2016).
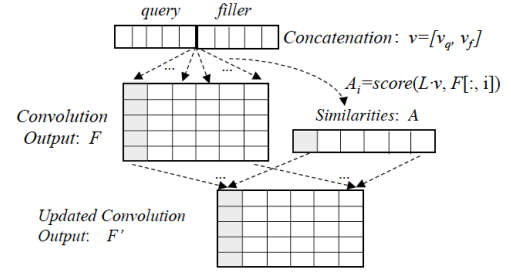

Figure 4: Local Attention.

Figure 4 illustrates our approach to incorporate local attention. We first concatenate the vector of query q and candidate filler f using pre-trained embeddings $v = [v_q, v_f], \in \mathbb{R}^{2d}$. For q or f that includes multiple words, we average the vectors of all individual words. For each convolution output F, which is a feature map $\in \mathbb{R}^{\widehat{K} \times N}$, where N is the number of word pairs from the input, and $\widehat{K}$ is the number of filters, we define the attention similarity matrix $A \in R^{N \times 1}$ as:

$$A_i = cosine(L \cdot v, F[:, i])$$

where $L \in \mathbb{R}^{\widehat{K} \times 2d}$ is the transformation matrix between the concatenated vector v and convolution output. $F[:, i]$ denotes the vector of column i in F. Then we use the attention matrix A to update each column of the feature map F, and generate an updated attention feature map F' as follows:

$$F'[:, i] = F[:, i] \cdot A[i]$$

4.2 Global Attention

Considering E1 in Section 1 again, the most discriminating word *owns* is not only related to the query and filler, but more specific to the type *org:subsidiaries*. Local attention aims to identify the query and filler related contexts. In order to detect type-indicative parts, we design global attention, using pre-learned slot type representations.

Wang et al. (2016) explored relation type attention with automatically learned type vectors from training data. However, in most cases, the training data is not balanced and some relation types cannot be assigned high-quality vectors with limited data. Thus, we designed two methods to generate pre-learned slot type representations.

First, we compose pre-trained lexical word embeddings of each slot type name to directly generate type representations. For example, for the type *per:date_of_birth*, we average the vectors of all single tokens (*person*, *birth*, *date*) within the type name as its representation.

Another new method is to take advantage of the large size of facts from external knowledge base (KB) to represent slot types. We use DBPedia as the target KB and manually map KB relations to slot types. For example, *per:alternate_names* can be mapped to *alternativeNames*, *birthName* and *nickName* in DBPedia. Thus for each slot type, we collect many triples: ⟨*query, slot, filler*⟩ and use TransE (Bordes et al., 2013), which models slot types as translations operating on the embeddings of query and filler, to derive a representation for each slot type. Compared with the first lexical based slot type representation induction approach, TransE jointly learns entity and relation representations and can better capture the correlation and differentiation among various slot types. Later, we will show the impact of these two types of slot type representations in Section 5.2.

Next we use the pre-learned slot type representations to guide the pooling process. Formally, let $R \in \mathbb{R}^{d \times r}$ be the matrix of all slot type vectors, where d is the vector dimension size and r is the number of slot types. Let $F \in \mathbb{R}^{\widehat{K} \times N}$ be a convolution output, which is the same as Section 4.1. We define the attention weight matrix S as:

$$S_{i,j} = cosine(F[:,i], W \cdot R[:,j])$$

where $W \in \mathbb{R}^{\widehat{K} \times d}$ is the transformation matrix for pre-learned slot type representations and convolution output. Given the weight matrix S, we generate the attention feature map F'' as follows:

$$F''[:,i] = F[:,i] \cdot \max_{j}\{S[i,j]\}$$

where $S[i,:]$ denotes the similarity scores between column i in F with all slot type vectors, and $\max\{S[i,:]\}$ is the max value among all similarity scores for column i in F.

Convolution Output: F   *Slot Type Vectors: R*

$S_{ij}=cosine\,(F[:,i],\,W*R)$

max

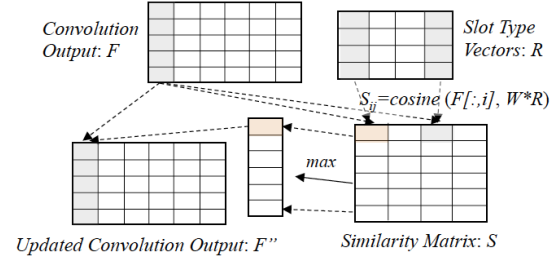*Updated Convolution Output: F''*   *Similarity Matrix: S*

Figure 5: Global Attention.

We apply local attention to each convolution output of each subgraph, then take the concatenation of three flattened attentive pooling outputs to a fully connected layer and generate a robust feature representation. Similarly, another feature representation is generated based on global attention. We concatenate these two features to the softmax layer to get the predicted types.

5 Experiments

5.1 Data

For model training, Angeli et al. (2014) created some high-quality clean annotations for SF based on crowd-sourcing[2]. In addition, Adel et al. (2016) automatically created a larger size of noisy training data based on distant supervision, including about 1,725,891 positive training instances for 41 slot types. We manually assessed the correctness of candidate filler identification and their slot type annotation, and extracted a subset of their noisy annotations and combined it with the clean annotations. Ultimately, we obtain 23,993 positive and 3,000 negative training instances for all slot types.

We evaluate our approach in two settings: (1) relation extraction for all slot types, given the boundaries of query and candidate fillers. We use a script[3] to generate a test set (4892 instances) from KBP 2012/2013 slot filling evaluation data sets with manual assessment. (2) apply our approach to re-classify and validate the results of slot filling systems. We use the data from the KBP 2013 Slot Filling Validation (SFV) shared task, which consists of merged responses returned by 52 runs from 18 teams submitted to the Slot Filling task.

We used the May-2014 English Wikipedia dump to learn word embeddings based on the Continuous Skip-gram model (Mikolov et al., 2013).

---

[2]http://nlp.stanford.edu/software/mimlre-2014-07-17-data.tar.gz

[3]http://cistern.cis.lmu.de.

Table 1 shows the hyper-parameters that we use to train embeddings and our model.

Parameter	Parameter Name	Value
d	Word Embedding Size	50
λ	Initial Learning Rate	0.1
$\widehat{K}$	# of Filters in Convolution Layer	500
h	Hidden Unit Size in Fully Connected Layer	1000
k^p	Max Pooling Size	3

Table 1: Hyper-parameters.

5.2 Relation Extraction

We compare with several existing state-of-the-art slot filling and relation extraction methods on slot filling data sets. Besides, we also design several variants to demonstrate the effectiveness of each component in our approach. Table 2 presents the detailed approaches and the features used by these methods.

We report scores with Macro F_1 and Micro F_1. Macro F_1 is computed from the average precision and recall of all types while Micro F_1 is computed from the overall precision and recall, which is more useful when the size of each category varies. Table 3 shows the comparison results on relation extraction.

We can see that by incorporating the shortest dependency path or regularized dependency graph into neural networks, the model can achieve more than 13% micro F-score gain over the previously widely adopted methods by state-of-the-art systems for SemEval relation classification. It confirms our claim that SF is a different and more challenging task than traditional relation classification and also demonstrates the effectiveness of dependency knowledge for SF.

In addition, by incorporating local or global attention mechanism into the GraphCNN, the performance can be further improved, which proves the effectiveness of these two attention mechanisms. Our method finally achieves absolute 16% F-score gain by incorporating the regularized dependency graph and two attention mechanisms.

To better quantify the contribution of different attention mechanisms on each slot type, we further compared the performances on each single slot type. Table 4 shows the gain/loss percentage of the Micro F1 by adding local attention or global attention into GraphCNN for each slot type. We can see that both attentions yield improvement for most slot types.

5.3 Slot Filling Validation

In TAC-KBP 2013 Slot Filling Validation (SFV) (Ji et al., 2011b) task, there are 100 queries. We first retrieve the sentences from the source corpus (about 2,099,319 documents) and identify the query and candidate filler using the offsets generated by each response, then apply our approach to re-predict the slot type. Figure 6 shows the F-scores based on our approach and the original system. For a system which has multiple runs, we select one for comparison. We can see that our approach consistently improves the performance of almost all SF systems in an absolute gain range of [-0.18%, 8.48%]. With analysis of each system run, we find that our approach can provide more gains to the SF systems which have lower precision.
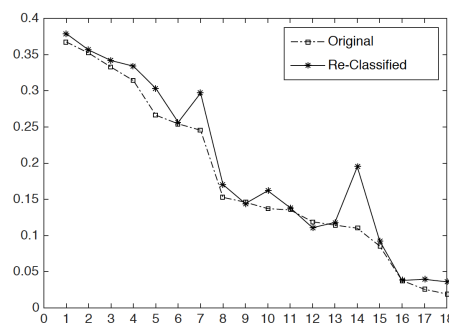

Figure 6: Comparison on Individual System

Previous studies (Tamang and Ji, 2011; Rodriguez et al., 2015; Zhi et al., 2015; Viswanathan et al., 2015; Rajani and Mooney, 2016a; Yu et al., 2014a; Rajani and Mooney, 2016b) for SFV trained supervised classifiers based on features such as confidence score of each response and system credibility. For comparison, we developed a new SFV approach: a new SVM classifier based on a set of features (docId, filler string, original predicted slot type and confidence score, new predicted slot type confidence score based on our neural architecture) for each response to take advantage of the redundant information from various system runs. Table 5 compares our SFV performance against previous reported scores on judging each response as true or false. We can see that our approach advances state-of-the-art methods.

5.4 Detailed Analysis

Significance Test: Table 3 shows the results of multiple variants of our approach. To demonstrate the difference between the results of these

	Method	Description	Features
Previous Methods	FCM (Yu et al., 2014b)	A factor-based compositional embedding model by deriving sentence-level and substructure representations	word embedding, dependency parse, WordNet, name tagging
	CR-CNN (Santos et al., 2015)	Applying a pairwise ranking loss function over CNNs	word embedding, word position embedding
	Context-CNN (Adel et al., 2016)	Splitting each sentence into three parts based on query and filler positions, and apply a CNNs to each part	word embedding
Our Methods	DepCNN	Applying CNNs to the shortest dependency path between query and filler	word embedding, dependency parse
	GraphCNN	DepCNN + applying CNNs to both query and filler related contextual graphs	word embedding, dependency parse
	GraphCNN+L	incorporating query and filler information as local attention into the GraphCNN	word embedding, dependency parse
	GraphCNN+G^1	incorporating slot type representations learned from type names as global attention into the GraphCNN	word embedding, dependency parse
	GraphCNN+G^2	incorporating slot type representations learned from external KB as global attention into the GraphCNN	word embedding, dependency parse, knowledge base
	GraphCNN+L+G^2	incorporating both local and KB based global attentions into the GraphCNN	word embedding, dependency parse, knowledge base

Table 2: Approach Descriptions for Multi-Class Relation Classification

	Method	Micro F1	Macro F1
Previous Methods	FCM	41.13	12.68
	CR-CNN	41.61	-
	ContextCNN	41.31	29.01
Variants of Our Methods	DepCNN	54.91	36.63
	GraphCNN	55.63	36.74
	GraphCNN+L	56.29	37.12
	GraphCNN+G^1	56.18	36.87
	GraphCNN+G^2	56.81	38.15
Our Method	GraphCNN+L+G^2	**57.39**	**38.26**

Table 3: Relation Extraction Component Performance on **Slot Filling Data Set** (%).

approaches are not random, we randomly sample 10 subsets (each contains 500 instances) from the testing dataset, and conduct paired t-test between each of these two approaches over these 10 data sets to check whether the average difference in their performances is significantly different or not. Table 6 shows the two-tailed P values. The differences are all considered to be statistically significant while all p-values are less than 0.05.

Impact of Training Data Size: We examine the impact of the size of training data on the performance for each slot type. Table 4 shows the distribution of training data and the F-score of each single type. We can see that, for some slot types, such as *per:date_of_birth* and *per:age*, the entity types of their candidate fillers are easy to learn and differentiate from other slot types, and their indicative words are usually explicit, thus our approach can get high f-score with limited training data (less than 507 instances). In contrast, for some slots, such as *org:location_of_headquarters*, their clues

are implicit and the entity types of candidate fillers are difficult to be inferred. Although the size of training data is larger (more than 1,433 instances), the f-score remains quite low. One possible solution is to incorporate fine-grained entity types from existing tools into the neural architecture.

Impact of Wide Context Distribution: We further compared the performance and distribution of instances with wide contexts across all slot types. A context is considered as wide if the query and candidate filler are separated with more than 7 words. The last column of Table 4 shows the performance by incorporating regularized dependency graph (ContextCNN *v.s.* GraphCNN). We can see that, for most slot types with wide contexts, such as *per:states_of_residence* and *per:employee_of*, the f-scores are improved significantly while for some slots such as *per:date_of_birth*, the f-scores decrease because most date phrases do not exist in our pre-trained embedding model.

Error Analysis: Both of the relation extraction and SFV results showed that, more than 58% classification errors are spurious. Besides, we also observed many misclassifications that are caused by conflicting clues. There may be several indicative words within the contexts, but only one slot type is labeled, especially between *per:location_of_death* and *per:location_of_residence*. For example, in the following sentence:

E4. ***Billy Mays***$_{query}$, *a beloved and parodied pitchman who became a pop-culture figure through his commercials for cleaning prod-*

| Slot Type | Impact of Attention (%) | | Training Data Distribution (%) | F1 (%) | Wide Context Distribution (%) | Impact of Dependency Graph (%) |
	Local	Global-KB				
state_of_death	9.8	-0.4	0.9	41.8	66.7	44.2
date_of_birth	7.3	121.3	**1.3**	**84.1**	**20.0**	**-81.9**
age	4.1	-5.3	**1.3**	**98.5**	15.9	28.5
per:alternate_names	-2.0	21.2	1.5	36.6	41.5	62.0
origin	-0.9	7.8	1.7	61.5	29.3	137.3
country_of_birth	16.7	12.0	1.9	61.5	55.6	162.5
city_of_death	1.1	3.3	1.9	61.3	70.3	24.4
state_of_headq.	9.7	-5.1	**3.1**	**51.7**	54.8	95.7
cities_of_residence	4.5	5.7	3.5	57.3	77.0	40.5
states_of_residence	-4.3	2.3	3.8	50.5	**45.9**	**175.8**
country_of_headq.	5.6	-0.8	**5.3**	**41.5**	54.4	146.3
city_of_headq.	1.6	-6.9	**6.7**	**30.3**	54.9	39.3
employee_of	14.9	4.9	7.3	65.9	**54.9**	**132.5**
countries_of_residence	37.7	8.6	7.4	47.4	47.2	134.9

Table 4: Comparison Analysis for Each Slot Type.

Methods	Precision (%)	Recall (%)	F-score (%)
Random	28.64	50.48	36.54
Voting	42.16	70.18	52.68
Linguistic Indicators	50.24	70.69	58.73
SVM	56.59	48.72	52.36
MTM (Yu et al., 2014a)	53.94	72.11	61.72
Our Approach	**70.46**	64.07	**67.11**

Table 5: Overall Performance for SFV: all the Baseline Systems are from Yu et al. (2014a).

Method 1	Method 2	P Value
DepCNN	GraphCNN	0.0165
GraphCNN	GraphCNN+L	0.0007
GraphCNN	GraphCNN+G^1	0.0160
GraphCNN	GraphCNN+G^2	<0.0001
GraphCNN+L	GraphCNN+L+G^2	0.0009
GraphCNN+G^2	GraphCNN+L+G^2	0.0010

Table 6: Statistical Significance Test.

*ucts like Orange Glo, OxiClean and Kaboom, died Sunday at his home in **Tampa**$_{filler}$, Fla.,*

the correct slot type is *per:city_of_death* while our approach mistakenly labeled it as *per:city_of_residence* with clue words like *home*. In addition, as we mentioned before, slot typing heavily relies on the fine-grained entity type of candidate filler, especially for the *location* (including *city, state, country*) related slot types. When the context is not specified enough, we can only rely on the pre-trained embeddings of candidate fillers, which may not be as informative as we hope. Such cases will benefit from introducing additional gazetteers such as *Geonames* [4].

---

[4]http://www.geonames.org/

6 Related Work

One major challenge of SF is the lack of labeled data to generalize a wide range of features and patterns, especially for slot types that are in the long-tail of the quite skewed distribution of slot fills (Ji et al., 2011a). Previous work has mostly focused on compensating the data needs by constructing patterns (Sun et al., 2011; Roth et al., 2014b), automatic annotation by distant supervision (Surdeanu et al., 2011; Roth et al., 2014a; Adel et al., 2016), and constructing trigger lists for unsupervised dependency graph mining (Yu and Ji, 2016; Yu et al., 2016). Some work (Rodriguez et al., 2015; Zhi et al., 2015; Viswanathan et al., 2015; Hong et al., 2015; Rajani and Mooney, 2016a; Yu et al., 2014a; Rajani and Mooney, 2016b; Ma et al., 2015) also attempted to validate slot types by combining results from multiple systems.

Our work is also related to dependency path based relation extraction. The effectiveness of dependency features for relation classification has been reported in some previous work (Bunescu and Mooney, 2005; Zhao and Grishman, 2005; GuoDong et al., 2005; Jiang and Zhai, 2007; Neville and Jensen, 2003; Ebrahimi and Dou, 2015; Xu et al., 2015; Ji et al., 2014). Liu et al. (2015), Cai et al. (2016) and Xu et al. (2015) applied CNN, bidirectional recurrent CNN and LSTM to CONLL relation extraction and demonstrated that the most important information has been included within the shortest paths between entities. Considering that the indicative words may not be included by the shortest dependency path between query and candidate filler, we enrich it to a regularized dependency graph by adding more contexts.

7 Conclusions and Future Work

In this work, we discussed the unique challenges of slot filling compared with tradition relation extraction tasks. We designed a regularized dependency graph based neural architecture for slot filling. By incorporating local and global attention mechanisms, this approach can better capture indicative contexts. Experiments on relation extraction and Slot Filling Validation data sets demonstrate the effectiveness of our neural architecture. In the future, we will combine additional rules, patterns, and constraints with DNN techniques to further improve slot filling.

Acknowledgments

References

Heike Adel, Benjamin Roth, and Hinrich Schütze. 2016. Comparing convolutional neural networks to traditional models for slot filling. In *Proc. NAACL2016*.

Heike Adel and Hinrich Schütze. 2015. Cis at tac cold start 2015: Neural networks and coreference resolution for slot filling. In *Proc. TAC2015*.

Gabor Angeli, Julie Tibshirani, Jean Wu, and Christopher D Manning. 2014. Combining distant and partial supervision for relation extraction. In *Proc. EMNLP2014*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Proc. NIPS2013*.

Razvan C Bunescu and Raymond J Mooney. 2005. A shortest path dependency kernel for relation extraction. In *Proc. EMNLP2005*.

Rui Cai, Xiaodong Zhang, and Houfeng Wang. 2016. Bidirectional recurrent convolutional neural network for relation classification. In *Proc. ACL2016*.

Javid Ebrahimi and Dejing Dou. 2015. Chain based rnn for relation classification. In *Proc. NAACL2015*.

Zhou GuoDong, Su Jian, Zhang Jie, and Zhang Min. 2005. Exploring various knowledge in relation extraction. In *Proc. ACL2005*.

Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. 2009. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions*.

Yu Hong, Di Lu, Dian Yu, Xiaoman Pan, Xiaobin Wang, Yadong Chen, Lifu Huang, and Heng Ji. 2015. Rpi blender tac-kbp2015 system description. In *Proc. TAC2015*.

Heng Ji, Taylor Cassidy, Qi Li, and Suzanne Tamang. 2014. Tackling representation, annotation and classification challenges for temporal knowledge base population. *Knowledge and Information Systems*.

Heng Ji, Ralph Grishman, and Hoa Trang Dang. 2011a. An overview of the tac2011 knowledge base population track. In *Proc. Text Analysis Conference (TAC2011)*.

Heng Ji, Ralph Grishman, and Hoa Trang Dang. 2011b. Overview of the tac2011 knowledge base population track.

Jing Jiang and ChengXiang Zhai. 2007. A systematic exploration of the feature space for relation extraction. In *HLT-NAACL*, pages 113–120.

Yang Liu, Furu Wei, Sujian Li, Heng Ji, and Ming Zhou. 2015. A dependency-based neural network for relation classification. In *Proc. ACL2015*.

Fenglong Ma, Yaliang Li, Qi Li, Minghui Qiu, Jing Gao, Shi Zhi, Lu Su, Bo Zhao, Heng Ji, and Jiawei Han. 2015. Faitcrowd: Fine grained truth discovery for crowdsourced data aggregation. In *Proc. SIGKDD*.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proc. ACL2014*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proc. NIPS2013*.

Jennifer Neville and David Jensen. 2003. Collective classification with relational dependency networks. In *Proceedings of the Second International Workshop on Multi-Relational Data Mining*.

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

Nazneen Fatema Rajani and Raymond J Mooney. 2016a. Combining supervised and unsupervised ensembles for knowledge base population. In *Proc. EMNLP2016*.

Nazneen Fatema Rajani and Raymond J Mooney. 2016b. Supervised and unsupervised ensembling for knowledge base population. *arXiv preprint arXiv:1604.04802*.

Miguel Rodriguez, Sean Goldberg, and Daisy Zhe Wang. 2015. University of florida dsr lab system for kbp slot filler validation 2015. In *Proc. TAC2015*.

Benjamin Roth, Tassilo Barth, Michael Wiegand, Mittul Singh, and Dietrich Klakow. 2014a. Effective slot filling based on shallow distant supervision methods. In *Proc. TAC KBP 2014*.

Benjamin Roth, Emma Strubell, John Sullivan, Lakshmi Vikraman, Kate Silverstein, and Andrew McCallum. 2014b. Universal schema for slot-filling, cold-start kbp and event argument extraction: Umassiesl at tac kbp 2014. In *TAC KBP*.

Cicero Nogueira dos Santos, Bing Xiang, and Bowen Zhou. 2015. Classifying relations by ranking with convolutional neural networks. In *Proc. ACL2015*.

Ang Sun, Ralph Grishman, Wei Xu, and Bonan Min. 2011. Nyu 2011 system for kbp slot filling. In *Proc. TAC2011*.

Mihai Surdeanu, Sonal Gupta, John Bauer, David McClosky, Angel X Chang, Valentin I Spitkovsky, and Christopher D Manning. 2011. Stanfords distantly supervised slot-filling system. In *Proc. TAC2011*.

Mihai Surdeanu and Heng Ji. 2014. Overview of the english slot filling track at the tac2014 knowledge base population evaluation. In *Proc. TAC2014*.

Suzanne Tamang and Heng Ji. 2011. Adding smarter systems instead of human annotators: re-ranking for system combination. In *Proceedings of the 1st international workshop on Search and mining entity-relationship data*, pages 3–8. ACM.

Vidhoon Viswanathan, Nazneen Fatema Rajani, Yinon Bentor, and Raymond Mooney. 2015. Stacked ensembles of information extractors for knowledge-base population. In *Proc. ACL2015*.

Linlin Wang, Zhu Cao, Gerard de Melo, and Zhiyuan Liu. 2016. Relation classification via multi-level attention cnns. In *Proc. ACL2016*.

Yan Xu, Lili Mou, Ge Li, Yunchuan Chen, Hao Peng, and Zhi Jin. 2015. Classifying relations via long short term memory networks along shortest dependency paths. In *Proc. EMNLP2015*.

Yunlun Yang, Yunhai Tong, Shulei Ma, and Zhi-Hong Deng. 2016. A position encoding convolutional neural network based on dependency tree for relation classification. In *Proc. EMNLP2016*.

Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. 2016. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. In *Proc. TACL2016*.

Dian Yu, Hongzhao Huang, Taylor Cassidy, Heng Ji, Chi Wang, Shi Zhi, Jiawei Han, and Malik Voss, Clare R anfd Magdon-Ismail. 2014a. The wisdom of minority: Unsupervised slot filling validation based on multi-dimensional truth-finding. In *Proc. COLING2014*.

Dian Yu and Heng Ji. 2016. Unsupervised person slot filling based on graph mining. In *Proc. ACL2016*.

Dian Yu, Heng Ji, Sujian Li, and Chin-Yew Lin. 2015. Why read if you can scan? trigger scoping strategy for biographical fact extraction. In *Proc. NAACL-HLT 2015*.

Dian Yu, Xiaoman Pan, Boliang Zhang, Lifu Huang, Di Lu, Spencer Whitehead, and Heng Ji. 2016. Rpi blender tac-kbp2016 system description.

Mo Yu, Matthew Gormley, and Mark Dredze. 2014b. Factor-based compositional embedding models. In *NIPS Workshop on Learning Semantics*.

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. 2014. Relation classification via convolutional deep neural network. In *Proc. COLING2014*.

Shubin Zhao and Ralph Grishman. 2005. Extracting relations with integrated information using kernel methods. In *Proc. ACL2005*.

Shi Zhi, Bo Zhao, Wenzhu Tong, Jing Gao, Dian Yu, Heng Ji, and Jiawei Han. 2015. Modeling truth existence in truth discovery. In *Proc. SIGKDD2015*.