Domain Adaptation for Relation Extraction with Domain Adversarial Neural Network

Lisheng Fu Thien Huu Nguyen[*] Bonan Min[†] Ralph Grishman
New York University, New York, NY, USA
{lisheng, grishman}@cs.nyu.edu
[*] University of Oregon, Eugene, OR, USA
thien@cs.uoregon.edu
[†] Raytheon BBN Technologies, Cambridge, MA, USA
bonan.min@raytheon.com

Abstract

Relations are expressed in many domains such as newswire, weblogs and phone conversations. Trained on a source domain, a relation extractor's performance degrades when applied to target domains other than the source. A common yet labor-intensive method for domain adaptation is to construct a target-domain-specific labeled dataset for adapting the extractor. In response, we present an unsupervised domain adaptation method which only requires labels from the source domain. Our method is a joint model consisting of a CNN-based relation classifier and a domain-adversarial classifier. The two components are optimized jointly to learn a domain-independent representation for prediction on the target domain. Our model outperforms the state-of-the-art on all three test domains of ACE 2005.

1 Introduction

Relation Extraction (RE) captures the semantic relation between two entities within a sentence, such as the Located relation between e1 and e2 in the sentence: "in the <e2>West Bank</e2>, a <e1>passenger </e1>was wounded when an Israeli bus came under fire." The same relation might be expressed differently across diverse documents, topics and genres. We often observed that a relation extractor's performance degrades when applied to a domain other than the domain it is trained on.

A simple method for domain adaptation (Blitzer et al., 2006; Daume, 2007; Jing and Zhai, 2007) is to construct a labeled dataset for the target domain, and then adjust a trained model with it. This is inefficient for relations - annotation is laborious to obtain, not to mention that relation mentions are sparse in the text. Take ACE 2004 as an example, Personal/Social relations appear only once on average per document. Such a method will not scale to the open-ended set of possible domains.

Among the features (Zhou et al., 2005) used for relation extraction, shortest dependency path can be applied cross-domain while argument-specific features (e.g., entity types, lexical forms) are likely to be more domain-specific. We hypothesize that it is possible to learn both domain-invariant and domain-specific representations with neural networks, and use the domain-invariant representation for many new domains.

In this paper, we propose to use a Domain Adversarial Neural Network (DANN) (Ganin and Lempitsky, 2015; Ajakan et al., 2014) to learn a domain-invariant representation for relations. Our contributions are twofold:

- We propose a novel domain adaptation approach for relation extraction that learns cross-domain features by itself and that requires no labels in targets.
- Experiments on the ACE domains show that our approach improves on the state-of-the-art across all domains.

In the rest of the paper, we will first briefly summarize related work, then describe the model (Section 3). We will present experimental results and conclusion at the end.

2 Related Work

There has been a lot of research on domain adaptation in natural language processing (Blitzer et al., 2006; Daume, 2007; Jing and Zhai, 2007; Glorot et al., 2011; Ajakan et al., 2014;

Ganin and Lempitsky, 2015). Most of the existing domain adaptation methods are based on discrete feature representations and linear classifiers. There is also recent work on domain adaptation for relation extraction including feature-based systems (Nguyen and Grishman, 2014; Nguyen et al., 2014) and kernel-based system (Plank and Moschitti, 2013). Nguyen and Grishman (2014) and Nguyen et al. (2014) both require a few labels in the target domain. Our proposed method can perform domain adaptation without target labels.

Some other methods also do not have such requirement. Plank and Moschitti (2013) designed the semantic syntactic tree kernel (SSTK) to learn cross-domain patterns. Nguyen et al. (2015b) constructed a case study comparing feature-based methods and kernel-based models. They presented some effective features and kernels (e.g. word embedding).We share the same intuition of finding those cross-domain features, but our work differs from such previous work in that they manually designed those features and kernels while we automatically learn cross-domain features from unlabeled target-domain examples with neural networks. To our best knowledge, this is the first work on neural networks for domain adaptation of relation extraction.

3 Model

We formulate the relation extraction task as a classification problem over all entity pairs (relation candidates) in a sentence. The overall structure of the model is shown in Figure 1. The model will first convert a relation candidate into a fixed-length matrix, then uses a single-layer Convolutional Neural Network (CNN) with dropout to learn its hidden representation *repr*. On top of *repr*, it uses two decoders: a fully-connected layer with dropout for predicting the relation type (Zeng et al., 2014) (Section 3.1), and another decoder with domain adversarial neural network(Ganin and Lempitsky, 2015) to predict its domain. The additional domain-adversarial decoder is used to enforce the feature layer to be domain-invariant (Section 3.2).

3.1 CNN-based Encoder-Decoder Model for Relations

Each sentence is truncated or padded to a fixed length (l_s) of tokens. Each token of the text is then represented as the concatenation of its word embedding, its position embedding and its entity type embedding. They form the input layer:

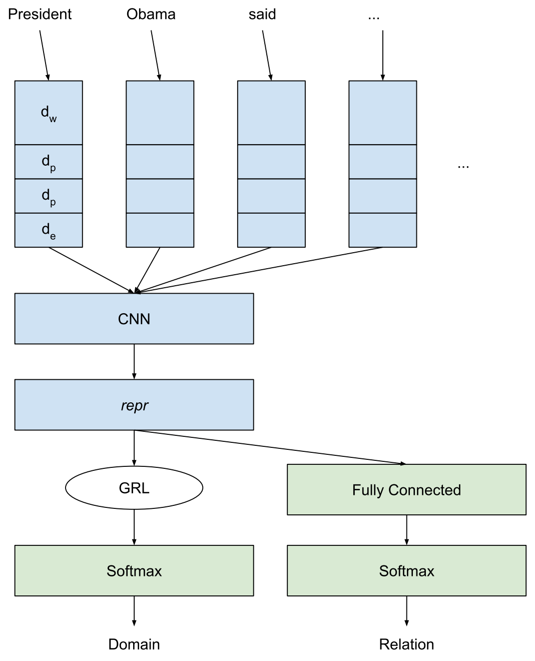

Figure 1: Model architecture

Word embedding: We use pre-trained word embedding from word2vec (Mikolov et al., 2013). The size of the embedding is $|V| \cdot d_w$, where $|V|$ is the vocabulary size, and d_w is the embedding dimension.

Position embedding: For each token, we look up its two position embeddings from the two position embedding tables (randomly initialized) with its relative distances to the two arguments, respectively. The final embedding is the concatenation of the two. The size of one embedding table is $(2 \cdot l_s - 1) \cdot d_p$, where l_s is the sentence length, and d_p is the embedding dimension.

Entity type embedding: For each token, we look up its entity-type embedding from the entity-type embedding table. Tokens outside the two entity spans will be randomly initialized to the same non-entity vector. Tokens within the two arguments will be converted to the vector of the argument's entity type. The size of the embedding table is $(|E| + 1) * d_e$, where $|E|$ is the number of entity types, and d_e is the embedding dimension.

Chunk embedding: Similar to entity type, we have chunk embedding according to each token's chunk type. The size of embedding table is $(|C| + 1) * d_c$, where $|C|$ is the number of chunk types, and d_c is the embedding dimension.

On dep path embedding: For each token, we have a vector to indicate whether the token is on the dependency path between the two entities. The vector size is d_d.

The input layer is a matrix with size $(d_w + 2 \cdot d_p + d_e + d_c + d_d) \cdot l_s$. A standard convolution layer with variable window sizes (feature maps) is applied on this, following by max-pooling and dropout. Each filter with the same window size has the same filter size. The output is the feature representation layer ($repr$) of size $d_f \cdot |W|$, where d_f is the filter size, and $|W|$ is the size of the set of window sizes. We add fully connected layers to this feature representation with softmax to predict the relation type. The model is similar to that in (Nguyen and Grishman, 2016), but with fewer features.

3.2 Domain Adversarial Neural Network

How does domain adaptation work without any labeled examples for the target domain? Following Ganin and Lempitsky (2015) and Ajakan et al. (2014), we use DANN to learn a representation that is more general across domains and eliminating source-only distinctive features that are easily learned with labeled source data.

It learns domain invariant features by jointly optimizing the underlying feature layer from the main learning task and the domain label predictor. In this case, the main learning task is the relation type prediction in Section 3.1. The domain label predictor is a binary classifier that discriminates whether the example is from source or target. The domain classifier consists of the gradient reversal layer (GRL) and a few fully connected layers. The GRL is defined as an identity function with reversed gradient for backpropagation. For input layer x:

$$GRL(x) = x, \frac{d}{dx}GRL(x) = -I$$

where I is the identity matrix.

We use a binary cross-entropy loss for the domain classifier:

$$L_{domain} = \sum_{i=0}^{N_s+N_t} \{d_i log(\hat{d_i}) + (1 - d_i) log(1 - \hat{d_i})\}$$

where $d_i \epsilon \{0, 1\}$ is the domain label $\{source, target\}$, and N_s, N_t stand for the number of examples in source and target.

The loss of the whole model is the linear combination of the task loss and the domain loss:

$$L = L_{relation} + \lambda \cdot L_{domain}$$

where λ is the adaptation weight, and $L_{relation}$ is the loss of the relation classifier.

During the training, half of the examples comes from the source and half of them comes from the target in a single batch. Only examples from source have relation labels, while both source and target examples have domain labels. As the result, the source part is used to calculate the relation loss $L_{relation}$. The whole batch is used to calculate the domain loss L_{domain}

We choose the feature representation layer ($repr$) from the relation model (Section 3.1) as the input to GRL. During the training, while the parameters of the relation and domain decoders are both optimized to *minimize* their errors, the parameters of $repr$ are optimized to *minimize* the loss of the relation decoder and to *maximize* (due to GRL) the loss of the domain classifier. The latter encourages domain-invariant features to emerge for domain adaptation.

In feature-based models, lexicon-level features are often domain-specific such as a person's name. e.g. word-level features that contain *Obama* and *US* can be indicators for employment relation. It is true in many news articles, but not in general. Instead of manually deciding whether to use the feature or not, we can use DANN to read the target domain text to make the decision depending on the domain.

4 Experiements

4.1 Dataset

We use the ACE 2005 dataset to evaluate domain adaptation by dividing its articles from its six genres into respective domains: broadcast conversation (*bc*), broadcast news (*bn*), telephone conversation (*cts*), newswire (*nw*), usenet (*un*) and weblogs (*wl*). Previous work (Gormley et al., 2015; Nguyen and Grishman, 2016) uses newswire (*bn & nw*) as the training set, half of *bc* as the development set, the other half of *bc*, *cts* and *wl* as the test sets. We use the same data split. Our model requires unlabeled target domain instances. To meet this requirement and avoid train-on-test, we also split *cts* and *wl* when adapting to them. For all three test domains, we use half of the dataset as the development set, and the other half as the test set (Table 1). We use the same training set and the same preprocessing. This results in 43,497 entity pairs for training. We also use the same label set which is expanded by creating two relation types for each asymmetric relation.

Split	bc†	wl	cts
train	nw & bn	nw & bn	nw & bn
dev	half of bc	half of wl	half of cts
test	half of bc	half of wl	half of cts

Table 1: Data split for the experiments. †This data split is the same as several previous work

4.2 Configuration and Hyperparameters

We use word embedding pre-trained on newswire with 300 dimensions from word2vec (Mikolov et al. 2013). We fix the word embeddings during the training because tuning did not show improvement. We follow Nguyen and Grishman (2016) to set the hyperparameters for CNN: the embedding sizes (Section 3.1) $d_e, d_p, d_d, d_c, d_d,$ = 50, the max sentence length l_s = 50, the set of filter window sizes W = 2, 3, 4, 5, the number of filters for each window size d_f = 150, and the dropout rate to be 0.5. We use one fully connected layer with 300 dimensions for the relation decoder before the softmax layer. We only use a softmax layer for domain decoder. The learning rate is 0.001. We halve the learning rate every two epoches. We use Adam as the optimization method. The adaptation weight is tuned to be 0.1 using the dev set. For all scores, we run experiments 10 times and take the average.

4.3 Evaluation

Method	bc	wl	cts	avg
Gormley 2015	61.90	N/A	N/A	N/A
Nguyen 2016	63.26	N/A	N/A	N/A
CNN	64.44	54.58	57.02	58.64
CNN + DANN	**65.16**	**55.55**	**57.19**	**59.30**

Table 2: Adaptation to the *bc* domain. F1 scores are reported on test sets with same splits. *p-value* < 0.01 for bc CNN vs. CNN+DANN.

Our baseline CNN model achieved comparable performance to the state-of-the-art relation extraction methods (Table 2). Compared to (Gormley et al., 2015; Nguyen and Grishman, 2016), our baseline model already obtained higher score on *bc*. They also reported higher scores by ensemble with other models (feature-based or multiple neural net models) which is not directly comparable to a single model. Essentially, our model can also serve as one of the base models in the ensemble.

We trained DANN to read the development set of *bc* to adapt to this domain. Although the gain seems to be small, the improvement is statistically significant (*p-value* < 0.01). We ran an instance-based sign test on the combination of the output of 10 experiments. We have 10 observations of each instance in the original dataset. we treat them as different examples when calculating the significance. While DANN improves *bc* significantly, we also want to find out how it works on other domains. In the original split used by previous work, *wl* and *cts* do not have dev and test split. We, therefore, created the data split by ourselves and compare the results to our own baseline model. We observe similar improvement on *wl*, but not on *cts*. By doing some feature engineering on the embedding layer, we found that the Chunk embedding and On dep path embedding improves the *cts* a lot. The model obtains 52.96 (without) and 57.02 (with) these embeddings. With DANN, it obtains 53.74 (+0.78) and 57.19 (+0.17). The effective hand-designed cross-domain features from the embedding layer could make the room for improvement smaller.

Given a group of documents, our approach is to let the DANN read more unlabeled documents from the same domain and train the relation decoder along with it. Then, we obtain a better model for this domain. This also means that we will have to train different models for different domains. Ideally, we would like to have a model that can work on all domains at the same time. To test this, we try to adapt to the three domains in the dataset at the same time. Under this setting, DANN reads unlabeled data from all three domains along with the supervised relation model. As the result (Table 3), the model tends to learn something in between. It performs better on *bc* and *wl*, but worse on *cts*. It is not very surprising since DANN will force the representation layer to be domain-invariant. To really lift the performance of all the domains with a single model, the model needs to capture some domain-specific representation as well. This would be hard to achieve without labels from the target domains, but still an interesting direction to investigate. Under the current situation, it would be better to train separate models that are adapted to each domain.

Method	bc	wl	cts	avg
CNN	64.33	54.58	57.02	58.64
+ DANN (all)	64.94	55.17	56.08	58.73
+ DANN (each)	**65.16**	**55.55**	**57.19**	**59.30**

Table 3: F1 scores on adaptation to all three domains at the same time and adaptation to each domain individually.

5 Conclusion

Our model successfully obtains improvement on all three test domains of relations at ACE 2005. It uses a domain adversarial neural network to learn cross-domain features. It does not require hand-crafted features for domain adaptation. It can be a useful tool for relation extraction since labeled data is always hard to acquire.

References

Hana Ajakan, Pascal Germain, Hugo Larochelle, Franois Laviolette, and Mario Marchand. 2014. Domain-adversarial neural networks. In *arXiv*.

John Blitzer, Ryan McDonald, and Fernando Pereira. 2006. Domain adaptation with structural correspondence learning. In *EMNLP*.

Hal Daume. 2007. Frustratingly easy domain adaptation. In *ACL*.

Yaroslav Ganin and Victor Lempitsky. 2015. Unsupervised domain adaptation by backpropagation. In *ICML*.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *ICML*.

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

Jiang Jing and ChengXiang Zhai. 2007. Instance weighting for domain adaptation in nlp. In *ACL*.

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

Minh Luan Nguyen, Ivor W. Tsang, Kian Ming Adam Chai, and Hai Leong Chieu. 2014. Robust domain adaptation for relation extraction via clustering consistency. In *ACL*.

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

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

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

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

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

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