Domain Adaptation for Syntactic and Semantic Dependency Parsing Using Deep Belief Networks

Haitong Yang, Tao Zhuang and Chengqing Zong
National Laboratory of Pattern Recognition
Institute of Automation, Chinese Academy of Sciences, Beijing, 100190, China
{htyang, tao.zhuang, cqzong}@nlpr.ia.ac.cn

Abstract

In current systems for syntactic and semantic dependency parsing, people usually define a very high-dimensional feature space to achieve good performance. But these systems often suffer severe performance drops on out-of-domain test data due to the diversity of features of different domains. This paper focuses on how to relieve this domain adaptation problem with the help of unlabeled target domain data. We propose a deep learning method to adapt both syntactic and semantic parsers. With additional unlabeled target domain data, our method can learn a latent feature representation (LFR) that is beneficial to both domains. Experiments on English data in the CoNLL 2009 shared task show that our method largely reduced the performance drop on out-of-domain test data. Moreover, we get a Macro F1 score that is 2.32 points higher than the best system in the CoNLL 2009 shared task in out-of-domain tests.

1 Introduction

Both syntactic and semantic dependency parsing are the standard tasks in the NLP community. The state-of-the-art model performs well if the test data comes from the domain of the training data. But if the test data comes from a different domain, the performance drops severely. The results of the shared tasks of CoNLL 2008 and 2009 (Surdeanu et al., 2008; Hajič et al., 2009) also substantiates the argument. To relieve the domain adaptation, in this paper, we propose a deep learning method for both syntactic and semantic parsers. We focus on the situation that, besides source domain training data and target domain test data, we also have some unlabeled target domain data.

Many syntactic and semantic parsers are developed using a supervised learning paradigm, where each data sample is represented as a vector of features, usually a high-dimensional feature. The performance degradation on target domain test data is mainly caused by the diversity of features of different domains, i.e., many features in target domain test data are never seen in source domain training data.

Previous work have shown that using word clusters to replace the sparse lexicalized features (Koo et al., 2008; Turian et al., 2010), helps relieve the performance degradation on the target domain. But for syntactic and semantic parsing, people also use a lot of syntactic features, i.e., features extracted from syntactic trees. For example, the relation path between a predicate and an argument is a syntactic feature used in semantic dependency parsing (Johansson and Nugues, 2008). Figure 1 shows an example of this relation path feature. Obviously, syntactic features like this are also very sparse and usually specific to each domain. The method of clustering fails in generalizing these kinds of features. Our method, however, is very different from clustering specific features and substituting these features using their clusters. Instead, we attack the domain adaption problem by learning a latent feature representation (LFR) for different domains, which is similar to Titov (2011). Formally, we propose a Deep Belief Network (DBN) model to represent a data sample using a vector of latent features. This latent feature vector is inferred by our DBN model

Transactions of the Association for Computational Linguistics, vol. 3, pp. 271–282, 2015. Action Editor: Hal Daumé III.
Submission batch: 3/2015; Published 5/2015.

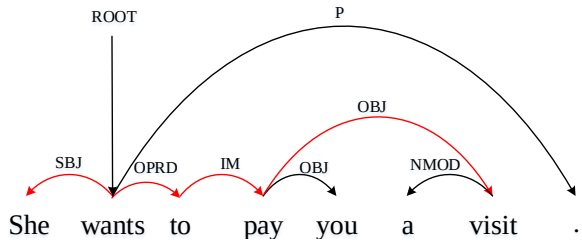

Figure 1: A path feature example. The red edges are the path between *She* and *visit* and thus the relation path feature between them is SBJ↑OPRD↓IM↓OBJ↓

based on the data sample's original feature vector. Our DBN model is trained unsupervisedly on original feature vectors of data in both domains: training data from the source domain, and unlabeled data from the target domain. So our DBN model can produce a common feature representation for data from both domains. A common feature representation can make two domains more similar and thus is very helpful for domain adaptation (Blitzer, 2006). Discriminative models using our latent features adapt better to the target domain than models using original features.

Discriminative models in syntactic and semantic parsers usually use millions of features. Applying a typical DBN to learn a sensible LFR on that many original features is computationally too expensive and impractical (Raina et al., 2009). Therefore, we constrain the DBN by splitting the original features into groups. In this way, we largely reduce the computational cost and make LFR learning practical. We carried out experiments on the English data of the CoNLL 2009 shared task. We use a basic pipelined system and compare the effectiveness of the two feature representations: original feature representation and our LFR. Using the original features, the performance drop on out-of-domain test data is 10.58 points in Macro F1 score. In contrast, using the LFR, the performance drop is only 4.97 points. And we have achieved a Macro F1 score of 80.83% on the out-of-domain test data. As far as we know, this is the best result on this data set to date.

2 Related Work

Dependency parsing and semantic role labeling are two standard tasks in the NLP community. There have been many works on the two tasks (McDonald et al., 2005; Gildea and Jurafsky, 2002; Yang and Zong, 2014; Zhuang and Zong, 2010a; Zhuang and Zong, 2010b, etc). Among them, researches on domain adaptation for dependency parsing and SRL are directly related to our work. Dredze et al., (2007) show that domain adaptation is hard for dependency parsing based on results in the CoNLL 2007 shared task (Nivre et al., 2007). Chen et al., (2008) adapted a syntactic dependency parser by learning reliable information on shorter dependencies in unlabeled target domain data. But they do not consider the task of semantic dependency parsing. Huang et al., (2010) used an HMM-based latent variable language model to adapt a SRL system. Their method is tailored for a chunking-based SRL system and can hardly be applied to our dependency based task. Weston et al., (2008) used deep neural networks to improve an SRL system. But their tests are on in-domain data.

On methodology, the work in Glorot et al., (2011) and Titov (2011) is closely related to ours. They also focus on learning LFRs for domain adaptation. However, their work deals with domain adaptation for sentiment classification, which uses much fewer features and training samples. So they do not need to worry about computational cost as much as we do. Titov (2011) used a graphical model that has only one layer of hidden variables. On contrast, we need to use a model with two layers of hidden variables and split the first hidden layer to reduce computational cost. The model of Titov (2011) also embodies a specific classifier. But our model is independent of the classifier to be used. Glorot et al., (2011) used a model called Stacked Denoising Auto-Encoders, which also contains multiple hidden layers. However, they do not exploit the hierarchical structure of their model to reduce computational cost. By splitting, our model contains much less parameters than theirs. In fact, the models in Glorot et al., (2011) and Titov (2011) cannot be applied to our task simply because of the high computational cost.

3 Our DBN Model for LFR

In discriminative models, each data sample is represented as a vector of features. Our DBN model maps this original feature vector to a vector of latent

features. And we use this latent feature vector to represent the sample, i.e., we replace the whole original feature vector by the latent feature vector. In this section, we introduce how our DBN model represent a data sample as a vector of latent features. Before introducing our DBN model, we first review a simpler model called Restricted Boltzman Machines (RBM) (Hinton et al., 2006). When training a DBN model, RBM is used as a basic unit in a DBN.

3.1 Restricted Boltzmann Machines

An RBM is an undirected graphical model with a layer of visible variables $\mathbf{v} = (v_1, ..., v_m)$, and a layer of hidden variables $\mathbf{h} = (h_1, ..., h_n)$. These variables are binary. Figure 2 shows a graphical representation of an RBM.

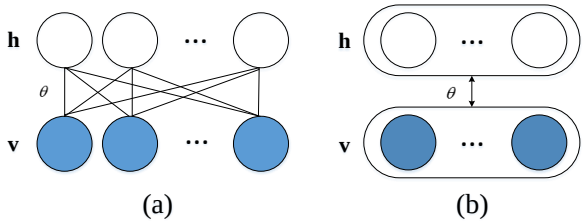

Figure 2: Graphical representations of an RBM: (a) represents an RBM. (b) is a more compact representation

The parameters of an RBM are $\theta = (W, a, b)$ where $W = (W_{ij})_{m \times n}$ is a matrix with W_{ij} being the weight for the edge between v_i and h_j, and $a = (a_1, ..., a_m)$, $b = (b_1, ..., b_n)$ are bias vectors for v and h respectively. The probabilistic model of an RBM is:

$$p(v, h|\theta) = \frac{1}{Z(\theta)} \exp(-E(v, h)) \qquad (1)$$

where

$$E(v, h) = -\sum_{i=1}^{m} a_i v_i - \sum_{j=1}^{n} b_j h_j - \sum_{i=1}^{m} \sum_{j=1}^{n} v_i w_{ij} h_j$$

$$Z(\theta) = \sum_{v,h} \exp(-E(v, h))$$

Because the connections in an RBM are only between visible and hidden variables, the conditional distribution over a hidden or a visible variable is quite simple:

$$p(h_j = 1|v) = \sigma(b_j + \sum_{i=1}^{m} v_i w_{ij}) \qquad (2)$$

$$p(v_i = 1|h) = \sigma(a_i + \sum_{j=1}^{n} h_i w_{ij}) \qquad (3)$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the logistic sigmoid function.

An RBM can be efficiently trained on a sequence of visible vectors using the *Contrastive Divergence* method (Hinton, 2002).

3.2 The Problem of Large Scale

In our syntactic and semantic parsing task, all features are binary. So each data sample (an shift action in syntactic parsing or an argument candidate in semantic parsing) is represented as a binary feature vector. By treating a sample's feature vector as visible variable vector in an RBM, and taking hidden variables as latent features, we could get the LFR of this sample using the RBM. However, for our syntactic and semantic parsing tasks, training such an RBM is computationally impractical due to the following considerations. Let m, n denote respectively the number of visible and hidden variables in the RBM. Then there are $O(mn)$ parameters in this RBM. If we train the RBM on d samples, then the time complexity for Contrastive Divergence training is $O(mnd)$. For syntactic or semantic parsing, there are over 1 million unique binary features, and millions of training samples. That means both m and d are in an order of 10^6. With m and n of that order, n should not be chosen too small to get a sensible LFR (Hinton, 2010). Our experience indicates that n should be at least in an order of 10^3. Now we see why the $O(mnd)$ complexity is formidable for our task.

3.3 Our DBN Model

A DBN is a probabilistic generative model that is composed of multiple layers of stochastic, latent variables (Hinton et al., 2006). The motivation of using a DBN is two-fold. First, previous research has shown that a deep network can capture high-level correlations between visible variables better than an RBM (Bengio, 2009). Second, as shown in the preceding subsection, the large scale of our task poses

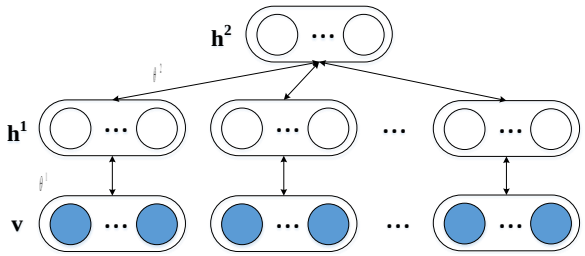

Figure 3: Our DBN model. The blue nodes stand for the visible variables (v) and the blank node stands for the hidden variables (h^1 and h^2). The symbols are also used in the figures of the following subsectins.

a great challenge for learning an LFR. By manipulating the hierarchical structure of a DBN, we can significantly reduce the number of parameters in the DBN model. This largely reduces the computational cost for training the DBN. Without this technique, it is impractical to learn a DBN model with that many parameters on large training sets.

As shown in Fig.3, our DBN model contains 2 layers of hidden variables: h^1, h^2, and a visible vector v. The visible vector corresponds to a sample's original feature vector. The second-layer hidden variable vector h^2 are used as the LFR of this sample.

Suppose there are m, n_1, n_2 variables in v, h^1, h^2 respectively. To reduce the number of parameters in the DBN, we split its first layer ($h^1 - v$) into k groups, as we will explain in the following subsection. We confine the connections in this layer to variables within the same group. So there are only mn_1/k parameters in the first layer. Without splitting, the number of parameters would be mn_1. Therefore, learning that many parameters requires too much computation. By splitting, we reduce the number of parameters by a factor of k. If we choose k big enough, learning is feasible.

The second layer ($h^2 - h^1$) is fully connected, so that the variables in the second layer can capture the relations between variables in different groups in the first layer. There are $n_1 n_2$ parameters in the second layers. Because n_1 and n_2 are relatively small, learning the parameters in the second layer is also feasible.

In summary, by splitting the first layer into groups, we have largely reduced the number of parameters in our DBN model. This makes learning our DBN model practical for our task. In our task, visible variables corresponds to original binary features and the second layer hidden variables are used as the LFR of these original features. One deficiency of splitting is that the relationships between original features in different groups can not be captured by hidden variables in the first layer. However, this deficiency is compensated by using the second layer to capture relationships between all variables in the first layer. In this way, the second layer still captures the relationships between all original features indirectly.

3.3.1 Splitting Features into Groups

When we split the first layer into k groups, every group, except the last one, contains $\lfloor m/k \rfloor$ visible variables and $\lfloor n_1/k \rfloor$ hidden variables. The last group contains the remaining visible and hidden variables. But how to split the visible variables, i.e., the original features, into these groups? Of course there are many ways to split the original features. But it is difficult to find a good principle to split. So we tried two splitting strategies in this paper. The first strategy is very simple. We arrange all features as the order they appeared in the training data . Suppose each group contains r original features. We just put the first r unique features of training data into the first group, the following r unique features into the second group, and so on.

The second strategy is more sophisticated. All features can be divided into three categories: the common features, the source-specific features and the target-specific features. Its main idea is to make each group contain the three categories of features evenly, which we think makes the distribution of features close to the 'true' distribution over domains. Let F_s and F_t denote the sets of features that appeared on source and target domain data respectively. We collect F_s and F_t from our training data. The features in F_s and F_t are are ordered the same as the order they appeared in training data. And let $F_{s\cap t} = F_s \cap F_t$ (the common features), $F_{s\setminus t} = F_s \setminus F_t$ (the source-specific features), $F_{t\setminus s} = F_t \setminus F_s$ (the target-specific features). So, to evenly distribute features in $F_{s\cap t}$, $F_{s\setminus t}$ and $F_{t\setminus s}$ to each group, each group should consist of $|F_{s\cap t}|/k$, $|F_{s\setminus t}|/k$ and $|F_{t\setminus s}|/k$ features from $F_{s\cap t}$, $F_{s\setminus t}$ and $F_{t\setminus s}$ respec-

tively. Therefore, we put the first $|F_{s\cap t}|/k$ features from $F_{s\cap t}$, the first $|F_{s\setminus t}|/k$ features from $F_{s\setminus t}$ and the first $|F_{t\setminus s}|/k$ features from $F_{t\setminus s}$ into the first group. Similarly, we put the second $|F_{s\cap t}|/k$ features from $F_{s\cap t}$, the second $|F_{s\setminus t}|/k$ features from $F_{s\setminus t}$ and the second $|F_{t\setminus s}|/k$ features from $F_{t\setminus s}$ into the second group. The intuition of this strategy is to let features in $F_{s\cap t}$ act as pivot features that link features in $F_{s\setminus t}$ and $F_{t\setminus s}$ in each group. In this way, the first hidden layer might capture better relationships between features from source and target domains.

3.3.2 LFR of a Sample

Given a sample represented as a vector of original features, our DBN model will represent it as a vector of latent features. The sample's original feature vector corresponds to the visible vector v in our DBN model in Figure 3. Our DBN model uses the second-layer hidden variable vector h^2 to represent this sample. Therefore, we must infer the value of hidden variables in the second-layer given the visible vector. This inference can be done using the methods in Hinton et al., (2006). Given the visible vector, the values of the hidden variables in every layer can be efficiently inferred in a single, bottom-up pass.

3.4 Training Our DBN Model

Inference in a DBN is simple and fast. Nonetheless, training a DBN is more complicated. A DBN can be trained in two stages: greedy layer-wise pretraining and fine tuning (Hinton et al., 2006).

3.4.1 Greedy Layer-wise Pretraining

In this stage, the DBN is treated as a stack of RBMs as shown in Figure 4.

The second layer is treated as a single RBM. The first layer is treated as k parallel RBMs with each group being one RBM. These k RBMs are parallel because their visible variable vectors constitute a partition of the original feature vector. In this stage, we train these constituent RBMs in a bottom-up layer-wise manner.

To learn parameters in the first layer, we only need to learn the parameters of each RBM in the first layer. With the original feature vector v given, these k RBMs can be trained using the *Contrastive Divergence* method (Hinton, 2002). After the first layer is trained, we will fix the parameters in the first layer and start to train the second layer.

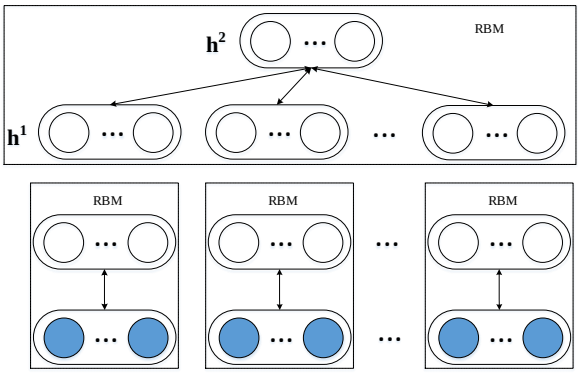

Figure 4: Stack of RBMs in pretraining.

For the RBM of the second layer, its visible variables are the hidden variables in the first layer. Given an original feature vector v, we first infer the activation probabilities for the hidden variables in the first layer using equation (2). And we use these activation probabilities as values for visible variables in the second layer RBM. Then we train the second layer RBM using contrastive divergence algorithm. Note that the activation probabilities are not binary values. But this is only a trick for training because using probabilities generally produces better models (Hinton et al., 2006). This trick does not change our assumption that each variable is binary.

3.4.2 Fine Tuning

The greedy layer-wise pretraining initializes the parameters of our DBN to sensible values. But these values are not optimal and the parameters need to be fine tuned. For fine tuning, we unroll the DBN to form an autoencoder as in Hinton and Salakhutdinov (2006), which is shown in Figure 5.

In this autoencoder, the stochastic activities of binary hidden variables are replaced by its activation probabilities. So the autoencoder is in essence a feed-forward neural network. We tune the parameters of our DBN model on this autoencoder using backpropagation algorithm.

4 Domain Adaptation with Our DBN Model

In this section, we introduce how to use our DBN model to adapt a basic syntactic and semantic de-

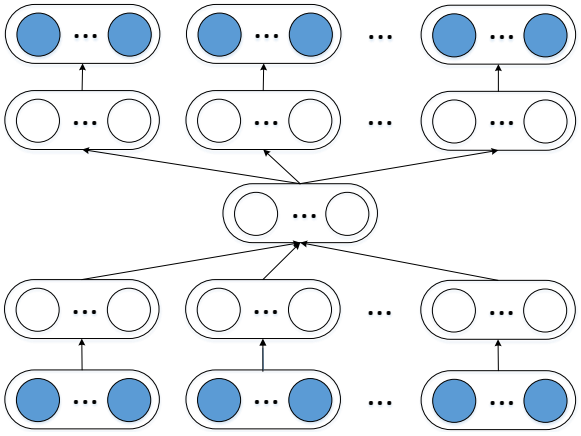

Figure 5: Unrolling the DBN.

pendency parsing system to target domain.

4.1 The Basic Pipelined System

We build a typical pipelined system, which first analyze syntactic dependencies, and then analyze semantic dependencies. This basic system only serves as a platform for experimenting with different feature representations. So we just briefly introduce our basic system in this subsection.

4.1.1 Syntactic Dependency Parsing

For syntactic dependency parsing, we use a deterministic shift-reduce method as in Nivre et al., (2006). It has four basic actions: left-arc, right-arc, shift, and reduce. A classifier is used to determine an action at each step. To decide the label for each dependency link, we extend the left/right-arc actions to their corresponding multi-label actions, leading to 31 left-arc and 66 right-arc actions. Altogether a 99-class problem is yielded for parsing action classification. We add arcs to the dependency graph in an arc eager manner as in Hall et al., (2007). We also projectivize the non-projective sequences in training data using the transformation from Nivre and Nilsson (2005). A maximum entropy classifier is used to make decisions at each step. The features utilized are the same as those in Zhao et al., (2008).

4.1.2 Semantic Dependency Parsing

Our semantic dependency parser is similar to the one in Che et al., (2009). We first train a predicate sense classifier on training data, using the same features as in Che et al., (2009). Again, a maximum entropy classifier is employed. Given a predicate, we need to decide its semantic dependency relation with each word in the sentence. To reduce the number of argument candidates, we adopt the pruning strategy in Zhao et al., (2009), which is adapted from the strategy in Xue and Palmer (2004). In the semantic role classification stage, we use a maximum entropy classifier to predict the probabilities of a candidate to be each semantic role. We train two different classifiers for verb and noun predicates using the same features as in Che et al., (2009). We use a simple method for post processing. If there are duplicate arguments for ARG0∼ARG5, we preserve the one with the highest classification probability and remove its duplicates.

4.2 Adapting the Basic System to Target Domain

In our basic pipeline system, both the syntactic and semantic dependency parsers are built using discriminative models. We train a syntactic parsing model and a semantic parsing model using the original feature representation. We will refer to this syntactic parsing model as *OriSynModel*, and the semantic parsing model as *OriSemModel*. However, these two models do not adapt well to the target domain. So we use the LFR of our DBN model to train new syntactic and semantic parsing models. We will refer to the new syntactic parsing model as *LatSynModel*, and the new semantic parsing model as *LatSemModel*. Details of using our DBN model are as follows.

4.2.1 Adapting the Syntactic Parser

The input data for training our DBN model are the original feature vectors on training and unlabeled data. Therefore, to train our DBN model, we first need to extract the original features for syntactic parsing on these data. Features on training data can be directly extracted using golden-standard annotations. On unlabeled data, however, some features cannot be directly extracted. This is because our syntactic parser uses history-based features which depend on previous actions taken when parsing a sentence. Therefore, features on unlabeled data can only be extracted after the data are parsed. To solve this problem, we first parse the unlabeled data using the already trained *OriSynModel*. In this way, we

can obtain the features on the unlabeled data. Because of the poor performance of the *OriSynModel* on the target domain, the extracted features on unlabeled data contains some noise. However, experiments show that our DBN model can still learn a good LFR despite the noise in the extracted features. Using the LFR, we can train the syntactic parsing model *LatSynModel*. Then by applying the LFR on test and unlabeled data, we can parse the data using *LatSynModel*. Experiments in later sections show that the *LatSynModel* adapts much better to the target domain than the *OriSynModel*.

4.2.2 Adapting the Semantic Parser

The situation here is similar to the adaptation of the syntactic parser. Features on training data can be directly extracted. To extract features on unlabeled data, we need to have syntactic dependency trees on this data. So we use our *LatSynModel* to parse the unlabeled data first. And we automatically identify predicates on unlabeled data using a classifier as in Che et al., (2008). Then we extract the original features for semantic parsing on unlabeled data. By feeding original features extracted on these data to our DBN model, we learn the LFR for semantic dependency parsing. Using the LFR, we can train the semantic parsing model *LatSemModel*.

5 Experiments

5.1 Experiment Setup

5.1.1 Experiment Data

We use the English data in the CoNLL 2009 shared task for experiments. The training data and in-domain test data are from the WSJ corpus, whereas the out-of-domain test data is from the Brown corpus. We also use unlabeled data consisting of the following sections of the Brown corpus: K, L, M, N, P. The test data are excerpts from fictions. The unlabeled data are also excerpts from fictions or stories, which are similar to the test data. Although the unlabeled data is actually annotated in Release 3 of the Penn Treebank, we do not use any information contained in the annotation, only using the raw texts. The training, test and unlabeled data contains 39279, 425, and 16407 sentences respectively.

5.1.2 Settings of Our DBN Model

For the syntactic parsing task, there are 748,598 original features in total. We use 7,486 hidden variables in the first layer and 3,743 hidden variables in the second layer. For semantic parsing, there are 1,074,786 original features. We use 10,748 hidden variables in the first layer and 5,374 hidden variables in the second layer.

In our DBN models, we need to determine the number of groups k. Because larger k means less computational cost, k should not be set too small. We empirically set k as follows: according to our experience, each group should contain about 5000 original features. We have about 10^6 original features in our tasks. So we estimate $k \approx 10^6/5000 = 200$. And we set k to be 200 in the DBN models for both syntactic and semantic parsing. As for splitting strategy, we use the more sophisticated one in subsection 3.3.1 because it should generate better results than the simple one.

5.1.3 Details of DBN Training

In greedy pretraining of the DBN, the contrastive divergence algorithm is configured as follows: the training data is divided to mini-batches, each containing 100 samples. The weights are updated with a learning rate of 0.3, momentum of 0.9, weight decay of 0.0001. Each layer is trained for 30 passes (epochs) over the entire training data.

In fine-tuning, the backpropagation algorithm is configured as follows: The training data is divided to mini-batches, each containing 50 samples. The weights are updated with a learning rate of 0.1, momentum of 0.9, weight decay of 0.0001. The fine-tuning is repeated for 50 epochs over the entire training data.

We use the fast computing technique in Raina et al., (2009) to learn the LFRs. Moreover, in greedy pretraining, we train RBMs in the first layer in parallel.

5.2 Results and Discussion

We use the official evaluation measures of the CoNLL 2009 shared task, which consist of three different scores: (i) syntactic dependencies are scored using the labeled attachment score, (ii) semantic dependencies are evaluated using a labeled F1 score, and (iii) the overall task is scored with a macro av-

Test data	System	LAS	Sem F1	Macro F1
WSJ	*Ori*	87.63	84.82	86.24
	Lat	87.30	84.25	85.80
Brown	*Ori*	79.72	71.57	75.67
	Lat	**82.84**	**78.75**	**80.83**

Table 1: The results of our basic and adapted systems

erage of the two previous scores. The three scores above are represented by LAS, Sem F1, and Macro F1 respectively in this paper.

5.2.1 Comparison with Un-adapted System

Our basic system uses the *OriSynModel* for syntactic parsing, and the *OriSemModel* for semantic parsing. Our adapted system uses the *LatSynModel* for syntactic parsing, and the *LatSemModel* for semantic parsing. The results of these two systems are shown in Table 1, in which our basic and adapted systems are denoted as *Ori* and *Lat* respectively.

From the results in Table 1, we can see that *Lat* performs slightly worse than *Ori* on in-domain WSJ test data. But on the out-of-domain Brown test data, *Lat* performs much better than *Ori*, with 5 points improvement in Macro F1 score. This shows the effectiveness of our method for domain adaptation tasks.

5.2.2 Different Splitting Configurations

As described in subsection 5.1.2, we have empirically set the number of groups k to be 200 and chosen the more sophisticated splitting strategy. In this subsection, we experiment with different splitting configurations to see their effects.

Under each splitting configuration, we learn the LFRs using our the DBN models. Using the LFRs, we test the our adapted systems on both in-domain and out-of-domain data. Therefore we get many test results, each corresponding to a splitting configuration. The in-domain and out-of-domain test results are reported in Table 2 and Table 3 respectively. In these two tables, 's1' and 's2' represents the simple and the more sophisticated splitting strategies in subsection 3.3.1 respectively. 'k' represents the number of groups in our DBN models. For both syntactic and semantic parsing, we use the same k in their DBN models. The 'Time' column reports the training time of our DBN models for both syntactic and semantic parsing. The unit of the 'Time' column is the hour. Please note that we only need to train our DBN models once. And we report the training time in Table 2. For easy viewing, we repeat those training times in Table 3. But this does not mean we need to train new DBN models for out-of-domain test.

Str	k	Time(h)	LAS	Sem F1	Macro F1
s1	100	392	**85.95**	**82.42**	**84.19**
	200	261	85.76	82.14	83.95
	300	218	85.48	81.68	83.58
	400	196	84.80	80.24	82.52
s2	100	392	**86.22**	**83.03**	**84.63**
	200	261	86.10	82.89	84.50
	300	218	85.72	82.24	83.98
	400	196	84.96	81.13	83.05

Table 2: Results of different splitting configurations on in-domain WSJ development data

Str	k	Time(h)	LAS	Sem F1	Macro F1
s1	100	392	**82.81**	**78.77**	**80.82**
	200	261	82.73	78.49	80.63
	300	218	82.44	77.90	80.37
	400	196	81.83	76.72	79.31
s2	100	392	**82.95**	**79.03**	**81.03**
	200	261	82.84	78.75	80.83
	300	218	82.63	78.34	80.50
	400	196	81.97	76.98	79.51

Table 3: Results of different splitting configurations on out-of-domain Brown test data

From Tables 2 and 3 we get the following observations:

First, although the more sophisticated splitting strategy 's2' generate slightly better result than the simple strategy 's1', the difference is not significant. This means that the hierarchical structure of our DBN model can robustly capture the relationships between features. Even with the simple splitting strategy 's1', we still get quite good results.

Second, the 'Time' column in Table 2 shows that different splitting strategies with the same k value has the same training time. This is reasonable because training time only depends on the number of parameters in our DBN model. And different splitting strategies do not affect the number of parameters in our DBN model.

Third, the number of groups k affects both the training time and the final results. When k increases, the training time reduces but the results degrade. As k gets larger, the time reduction gets less obvious, but the degradation of results gets more obvious. When $k = 100$, 200, 300, there is not much difference between the results. This shows that the results of our DBN model is not sensitive to the values of k within a range of 100 around our initial estimation 200. But when k is further away from our estimation, e.g. $k = 400$, the results get significantly worse.

Please note that the results in Tables 2 and 3 are not used to tune the parameter k or to choose a splitting strategy in our DBN model. As mentioned in subsection 5.1.2, we have chosen $k = 200$ and the more sophisticated splitting strategy beforehand. In this paper, we always use the results with $k = 200$ and the 's2' strategy as our main results, even though the results with $k = 100$ are better.

5.3 The Size of Unlabeled Target Domain Data

An interesting question for our method is how much unlabeled target domain data should be used. To empirically answer this question, we learn several LFRs by gradually adding more unlabeled data to train our DBN model. We compared the performance of these LFRs as shown in Figure 6.

Figure 6: Macro F1 scores on test data with respect to the size of unlabeled target domain data used in DBN training. The horizontal axis is the number of sentences in unlabeled target domain data and the coordinate axis is the Macro F1 Score.

From Figure 6, we can see that by adding more unlabeled target domain data, our system adapts better to the target domain with only small degradation of result on source domain. However, with more unlabeled data used, the improvement on target domain result gradually gets smaller.

5.4 Comparison with other methods

In this subsection, we compare our method with several systems. These are described below.

Daume07. Daumé III (2007) proposed a simple and effective adaptation method by augmenting feature vector. Its main idea is to augment the feature vector. They took each feature in the original problem and made three versions of it: a general version, a source-specific version and a target-specific version. Thus, the augmented source data contains only general and source-specific versions; the augmented target data contains general and target-specific versions. In the baseline system, we adopt the same technique for dependency and semantic parsing.

Chen. The participation system of Zhao et al., (2009), reached the best result in the out-of-domain test of the CoNLL 2009 shared task.

In Daumé III and and Marcu (2006), they presented and discussed several 'obvious' ways to attack the domain adaptation problem without developing new algorithms. Following their idea, we construct similar systems.

OnlySrc. The system is trained on only the data of the source domain (News).

OnlyTgt. The system is trained on only the data of the target domain (Fiction).

All. The system is trained on all data of the source domain and the target domain.

It is worth noting that training the systems of **Daume07**, **OnlyTgt** and **All** need the labeled data of the target domain. We utilize **OnlySrc** to parse the unlabeled data of the target domain to generate the labeled data.

ALl comparison results are shown in Table 4, in which the 'Diff' column is the difference of scores on in-domain and out-of-domain test data.

First, we compare *OnlySrc*, *OnlyTgt* and *All*. We can see that *OnlyTgt* performs very poor both in the source domain and in the target domain. It is not hard to understand that *OnlyTgt* performs poor in the source domain because of the adaptation problem. *OnlyTgt* also performs poor in the target domain. We think the main reason is that *OnlyTgt* is trained on the auto parsed data in which there are

Score	System	WSJ	Brown	Diff
LAS	OnlySrc	87.63	79.72	7.91
	OnlyTgt	73.25	78.30	5.05
	All	87.41	80.54	6.87
	Daume07	87.47	80.46	7.01
	Chen	**89.19**	82.38	6.81
	Ours	87.30	**82.84**	**4.46**
Sem F1	OnlySrc	84.82	71.57	13.25
	OnlyTgt	73.74	70.34	**3.40**
	All	84.68	72.75	11.93
	Daume07	84.52	72.90	11.62
	Chen	**86.15**	74.58	11.57
	Ours	84.25	**78.75**	5.50
Macro F1	OnlySrc	86.24	75.67	10.57
	OnlyTgt	73.50	74.32	**0.82**
	All	86.04	76.65	9.40
	Daume07	86.00	76.68	9.32
	Chen	**87.69**	78.51	9.18
	Ours	85.80	**80.83**	4.97

Table 4: Comparison with other methods.

many parsing errors. But we note that *All* performs better than both *OnlySrc* and *OnlyTgt* on the target domain test, although its training data contains some auto parsed data. Therefore, the data of the target domain, labeled or unlabeled, are potential in alleviating the adaptation problem of different domains. But *All* just puts the auto parsed data of the target domain into the training set. Thus, its improvement on the test data of the target domain is limited. In fact, how to use the data of the target domain, especially the unlabeled data, in the adaptation problem is still an open and hot topic in NLP and machine learning.

Second, we compare *Daume07*, *All* and our method. In *Daume07*, they reported improvement on the target domain test. But one point to note is that the target domain data used in their experiments is labeled while in our case there is only unlabeled data. We can see *Daume07* have comparable performance with *All* in which there is not any adaptation strategy besides adding more data of the target domain. We think the main reason is that there are many parsing errors in the data of the target domain. But our method performs much better than *Daume07* and *All* even though some faulty data are also utilized in our system. This suggests that our method successfully learns new robust representations for different domains, even when there are some noisy data.

Third, we compare *Chen* with our method. *Chen* reached the best result in the out-of-domain test of the CoNLL 2009 shared task. The results in Table 4 show that Chen's system performs better than ours on in-domain test data, especially on LAS score. Chen's system uses a sophisticated graph-based syntactic dependency parser. Graph-based parsers use substantially more features, e.g. more than 1.3×10^7 features are used in McDonald et al., (2005). Learning an LFR for that many features would take months of time using our DBN model. So at present we only use a transition-based parser. The better performance of Chen's system mainly comes from their sophisticated syntactic parsing method.

To reduce the sparsity of features, Chen's system uses word cluster features as in Koo et al., (2008). On out-of-domain tests, however, our system still performs much better than Chen's, especially on semantic parsing. To our knowledge, on out-of-domain tests on this data set, our system has obtained the best performance to date. More importantly, the performance difference between indomain and out-of-domain tests is much smaller in our system. This shows that our system adapts much better to the target domain.

6 Conclusions

In this paper, we propose a DBN model to learn LFRs for syntactic and semantic parsers. These LFRs are common representations of original features in both source and target domains. Syntactic and semantic parsers using the LFRs adapt to target domain much better than the same parsers using original feature representation. Our model provides a unified method that adapts both syntactic and semantic dependency parsers to a new domain. In the future, we hope to further scale up our method to adapt parsing models using substantially more features, such as graph-based syntactic dependency parsing models. We will also search for better splitting strategies for our DBN model. Finally, although our experiments are conducted on syntactic and semantic parsing, it is expected that the proposed ap-

proach can be applied to the domain adaptation of other tasks with little adaptation efforts.

Acknowledgements

The research work has been partially funded by the Natural Science Foundation of China under Grant No.61333018 and supported by the West Light Foundation of Chinese Academy of Sciences under Grant No.LHXZ201301. We thank the three anonymous reviewers and the Action Editor for their helpful comments and suggestions.

References

Yoshua Bengio. 2009. Learning Deep Architectures for AI. In *Foundations and Trends in Machine Learning*, 2(1):1-127.

John Blitzer, Ryan McDonald and Fernando Pereira. 2006. Domain Adaptation with sturctural correspondance learning. In *Proceedings of ACL-2006*.

Wanxiang Che, Zhenghua Li, Yuxuan Hu, Yongqiang Li, Bing Qin, Ting Liu and Sheng Li. 2008. A Cascaded Syntactic and Semantic Dependency Parsing System. In *Proceedings of CoNLL-2008 shared task*.

Wanxiang Che, Zhenghua Li, Yongqiang Li, Yuhang Guo, Bing Qin and Ting Liu. 2009. Multilingual Dependency-based Syntactic and Semantic Parsing. In *Proceedings of CoNLL-2009 shared task*.

Wenliang Chen, YouzhengWu and Hitoshi Isahara. 2008. Learning reliable information for dependency parsing adaptation. In *Proceedings of COLING-2008*.

Hal Daumé III. 2007. Frustratingly Easy Domain Adaptation. In *Proceedings of ACL-2007*.

Hal Daumé III and Daniel Marcu. 2006. Domain Adaptation for Statistical Classifer. In *Journal of Artificial Intelligence Research*, 26(2006), 101-126.

Mark Dredze, John Blitzer, Partha P. Talukdar, Kuzman Ganchev, Joao Graca and Fernando Pereira. 2007. Frustratingly Hard Domain Adaptation for Dependency Parsing. In *Proceedings of EMNLP-CoNLL-2007*.

Xavier Glorot, Antoine Bordes and Yoshua Bengio. 2011. Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach. In *Proceedings of International Conference on Machine Learning (ICML) 2011*.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic labeling for semantic roles. In *Computational Linguistics*, 28(3): 245-288.

I. Goodfellow, Q. Le, A. Saxe and A. Ng. 2009. Measuring invariances in deep networks. In *Proceedings of Advances in Neural Information Processing Systems(NIPS)2011*.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue and Yi Zhang. 2009. The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages. In *Proceedings of CoNLL-2009*.

J. Hall, J. Nilsson, J. Nivre, G. Eryiğit, B. Megyesi, M. Nilsson, and M. Saers. 2007. Single Malt or Blended? A Study in Multilingual Parser Optimization. In *Proceedings of EMNLP-CoNLL-2007*.

Geoffrey Hinton. 2010. A Practical Guide to Training Restricted Boltzmann Machines. In *Technical report 2010-003, Machine Learning Group, University of Toronto*.

Geoffrey Hinton. 2002. Training products of experts by minimizing constrastive divergence. In *Neural Computation*, 14(8): 1711-1800.

Geoffrey Hinton, Simon Osindero and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. In *Neural Computation*, 18(7): 1527-1554.

Geoffrey Hinton and R. Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. In *Science*, 313(5786), 504-507.

Richard Johansson and Pierre Nugues. 2008. Dependency-based semantic role labeling of PropBank. In *Proceedings of EMNLP-2008*.

Terry Koo, Xavier Carreras and Michael Collins. 2008. Simple Semi-supervised Dependency Parsing. In *Proceedings of ACL-HLT-2008*.

Lluís Màrquez, Xavier Carreras, Kenneth C.Litkowski and Suzanne Stevenson. 2008. Semantic Role Labeling: An Introduction to the Special Issue. In *Computational Linguistics*, 34(2):145-159.

Ryan McDonald, Fernando Pereira, Jan Hajˇc, and Kiril Ribarov. 2005. Non-projective dependency parsing using spanning tree algortihms. In *Proceedings of NAACL-HLT-2005*.

J. Nivre, J. Hall, S. Kübler, R. Mcdonald, J. Nilsson, S. Riedel, and D. Yuret. 2007. The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of CoNLL-2007*.

J. Nivre, J. Hall, J. Nilsson, G. Eryiğit and S. Marinov. 2006. Labeled Pseudo-Projective Dependency Parsing with Support Vector Machines. In *Proceedings of CoNLL-2006*.

J. Nivre, and J. Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of ACL-2005*.

Rajat Raina, Anand Madhavan, and Andrew Y. Ng. 2009. Large-scale Deep Unsupervised Learning using Graphics Processors. In *Proceedings of the 26th*

Annual International Conference on Machine Learning(ICML), pages 152-164.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez and Joakim Nivre. 2008. The CoNLL-2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies. In *Proceedings of CoNLL-2008*.

Ivan Titov. 2011. Domain Adaptation by Constraining Inter-Domain Variability of Latent Feature Representation. In *Proceedings of ACL-2011*.

Joseph Turian, Lev Ratinov and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of ACL-2010*.

J. Weston, F. Rattle, and R. Collobert. 2008. Deep Learning via Semi-Supervised Embedding. In *Proceedings of International Conference on Machine Learning(ICML)*.

Nianwen Xue and Martha Palmer. 2004. Calibrating features for semantic role labeling. In *Proceedings of EMNLP-2004*.

Haitong Yang and Chengqing Zong. 2014. Multi-Predicate Semantic Role Labeling. In *Proceedings of EMNLP-2014*.

Hai Zhao, Wenliang Chen, Chunyu Kit, Guodong Zhou. 2009. Multilingual Dependency Learning: Exploiting Rich Features for Tagging Syntactic and Semantic Dependencies. In *Proceedings of CoNLL-2009 shared task*.

Hai Zhao and Chunyu Kit. 2008. Parsing Syntactic and Semantic Dependencies with Two Single-Stage Maximum Entropy Models. In *Proceedings of CoNLL-2008*.

Tao Zhuang and Chengqing Zong. 2010a. A Minimum Error Weighting Combination Strategy for Chinese Semantic Role Labeling. In *Proceedings of COLING 2010*.

Tao Zhuang and Chengqing Zong. 2010b. Joint Inference for Bilingual Semantic Role Labeling. In *Proceedings of EMNLP 2010*.