# CS 388 HW3: Neural Dependency Parsing with "Unsupervised" Domain Adaptation

Mit Shah (UT EID: mks3226)
University of Texas at Austin
Austin, TX 78712
mkshah@utexas.edu

## Abstract

*This assignment is aimed at exploring the concept of domain adaptation / transfer learning, while using the neural parsers. Parsers learned are generally specific to training corpus and when tested on other data sets, their performance drops significantly. One of the approach to handle this is to learn on Source domain, where labels are available (Seed data) and then transfer it to target domain by using a few labeled or no labeled data (Self-training data) present for it. So, a series of experiments are performed here by varying the size of seed and self-training data and their effects on test accuracy of target domain is analyzed. Stanford Neural Dependency parser is used here for the experiments.*

## 1. Introduction

We will be looking at task of parsing the sentences. So, let's first have a look at what Statistical Dependency Parsers are and in particular, Neural Dependency parser. Then, we will look at domain adaptation.

### 1.1. Statistical Parsers

Statistical parsing uses a probabilistic model of syntax in order to assign probabilities to each parse tree. It is a principled approach to solve syntactic ambiguity. It allows supervised learning of parsers from tree-banks of parse trees provided by human linguists. It also allows unsupervised parsing from unannotated text, but accuracy of such parsers are very low. Its main applications are in Observation Likelihood, Most likely derivation and Maximum likelihood training.

### 1.2. Shift Reduce Parser

They deterministically build a parse incrementally bottom up and left to right, without backtracking. It maintains a buffer of input words and a stack of constructed constituents. Then, at each step one of the two actions - either "Shift" or "Reduce" are performed. Shift pushes the next word on the buffer on to the stack. Reduce replaces a set of top elements on the stack with a constituent composed of them. It was originally introduced to parse programming languages which are DCFLs.

Using for NLP requires heuristic to pick an action at each step. It can be easily adapted to dependency parsing by using reduce operators that introduce dependency arcs. Rather than Shift and Reduce actions, now actions can be seen as "LEFT-ARC", "RIGHT-ARC" and "Shift". Now there are various ways of choosing which action to perform at each step, one of which is known as Neural Dependency Parser, which is discussed in the following subsection.

#### 1.2.1 Neural Dependency Parser

Mainly it involves training a neural net to choose the best shift-reduce parser action to take at each step. It uses words, POS tags, arc labels extracted from the current stack, buffer and arcs as context. Inputs are embedded into a 50 dimensional set of input features instead of using one hot encoding and it uses Cubic Activation Function.

There are mainly two evaluation metrics which are used for Dependency Parsing, Unlabeled Attachment Score (UAS) and Labeled Attachment Score (LAS). Former calculates percentage of tokens for which system has predicted the current parent, while later requires the arc label to also be correct in addition to parent. Here, we will use LAS to compare different models.

### 1.3. Domain Adaptation

As mentioned in Abstract, parsers learned can be quite specific to genre of the training corpus.For example, a parser trained on WSJ will perform significantly less on Brown dataset, which contains a wide variety of genres. Generally, there is extensive amount of labeled data available for a "source" dataset, while we may not have that for other "target" dataset. Labeling the target dataset for super-

vised training is very expensive and labor intensive. So, our goal is to transfer the learning on "source" dataset to "target" dataset, while using very little or no labeled data from target. This task is called Domain Adaptation or Transfer Learning.

Here, we will be working on the task of Unsupervised Domain Adaptation. We will only have unlabeled training data for target domain. Techniques to do so are mentioned in the following section.

## 2. Method

There are various ways to deal with the task of Unsupervised Domain Adaptation. One of them is Self training, a form of Semi-Supervised Learning. Here, system is initially trained on labeled data from the source domain, which is called Seed data. Then model generated is used to automatically label some portion of the target domain, which is called the Self-Training data. Afterwards, Self-Training data and their labels are appended to seed data and model is retrained. Here, we will be using this approach to check the feasibility of self training for PCFGs.

Here, we will be replicating results in Reichart and Rappoport paper using Stanford neural dependency parser instead of the Collins' parser.

### 2.1. Preprocessing Dataset

For WSJ dataset, folders 02-10 were used as training set and folder 23 was used as testing set. So to generate a single train file, all the files in folders 02-10 were concatenated and same for the test file. Because training size has to be varied, different files containing first 1000, 2000, etc. sentences were generated from the train file.

For Brown dataset, there were multiple genres. All files belonging to one genre were concatenated initially. Afterwards, initial 90% of each genre file were extracted and merged into a single train file. Remaining 10% of each genre was merged into a single test file. Again, as training size has to be varied, different files containing first 1000, 2000, etc. sentences were generated from the train file.

This way, we had dataset ready in the form that can be directly given input to actual algorithm.

## 3. Experiments and Results

There were mainly four set of experiments performed. First two had WSJ as source domain and Brown as target domain while last two had other way around. In both of them seed data size and self-training data size were varied one by one to see their effects on the LAS score. Each time 200 iterations were used for both - training model from seed data and retraining that model with additional self-training data. We will look at each of them one by one.

### 3.1. WSJ as Source and Brown as target

Here, we will do 2 experiments. First we will vary the seed size of WSJ data set and next, we will vary the self training size of Brown data set.

#### 3.1.1 Varying Seed Size

Here we will train models for different size of seed data from WSJ. We will use first 1000, 2000, 3000, 4000, 5000, 7000, 10000, 12000 and 14000 sentences from train file of WSJ as seed data. Then full train file of Brown will be used as self training data and finally the model will be tested on test file of Brown.

Also, for the comparison purpose, we will use these different size of seed data to train a model and will directly test on test files of both - WSJ and Brown.

As we can see in 1, increasing size of seed data helps improve the performance for all the 3 curves in long term. Also, self-training on brown helps improve the performance compared to direct testing on Brown by around 1% throughout. Though performance of testing on WSJ itself is much higher than both of them, which shows that testing on different domain results in significant drop in performance.

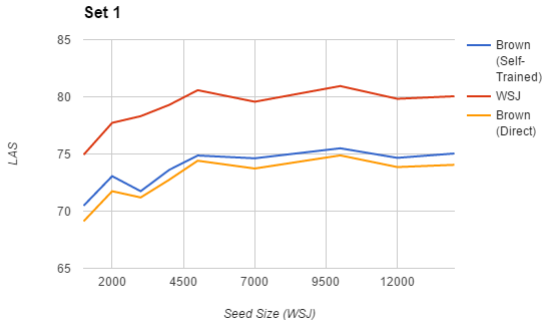


Figure 1. Set 1

#### 3.1.2 Varying Self-Training Size

Here we will keep the size of seed data fixed to 10000 sentences of WSJ and will use first 1000, 2000, 3000, 4000, 5000, 7000, 10000, 13000, 17000 and 21000 sentences from brown as self-training data. And the model will be tested on test file of Brown.

Form 2, we can see that increasing the size of self training data has actually adverse effect on the performance. With increasing self training data performance drops down from around 78% to 75.5%. It shows that a minimum amount of self training data is required. Additional self training data does not help, in fact reduces the performance.

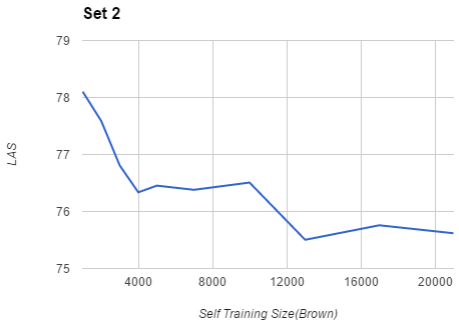


Figure 2. Set 2

### 3.2. Brown as Source and WSJ as target

Here, we will do 2 experiments. First we will vary the seed size of Brown data set and next, we will vary the self training size of WSJ data set.

#### 3.2.1 Varying Seed Size

Here we will train models for different size of seed data from Brown. We will use first 1000, 2000, 3000, 4000, 5000, 7000, 10000, 13000, 17000 and 21000 sentences from train file of Brown as seed data. Then full train file of WSJ will be used as self training data and finally the model will be tested on test file of WSJ.

Also, for the comparison purpose, we will use these different size of seed data to train a model and will directly test on test files of both - Brown and WSJ.

Same trends are observed as in first set. As we can see in 3, increasing size of seed data significantly improves the performance for all the 3 curves. Self-training on WSJ does help improve the performance by around 2% throughout. Again, performance of testing on Brown itself is much higher than both of them, which again confirms that the testing on different domain results in significant drop in performance.

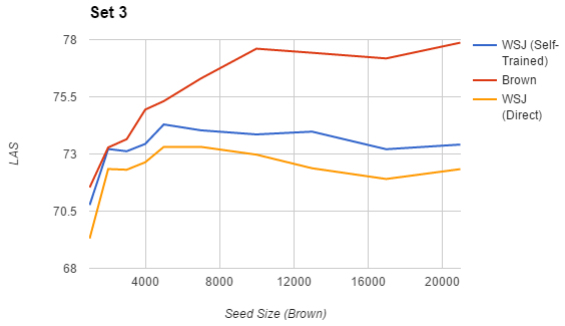


Figure 3. Set 3

#### 3.2.2 Varying Self-Training Size

Here we will keep the size of seed data fixed to 10000 sentences of Brown and will use first 1000, 2000, 3000, 4000, 5000, 7000, 10000, 12000 and 14000 sentences from WSJ as self-training data. And the model will be tested on test file of WSJ.

Here also, same trends are observed as in Set 2. As in 4, we can confirm that increasing the size of self training data has adverse effect on the performance. With increasing self training data of WSJ, performance drops down from around 76% to 74%.

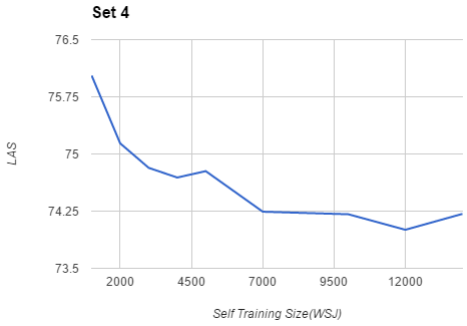


Figure 4. Set 4

## 4. Discussion

From the experiments in Set 1 and 3, we concluded that there was significant drop from in-domain testing to out-domain testing. When trained on WSJ, testing accuracy on WSJ was 80%, while that on Brown was 74%. Same way when, trained on Brown, testing accuracy on Brown was 78%, while that on WSJ was 72%.

Also, from 1 and 3, we analyzed that unsupervised domain adaptation did help improve the performance on out-domain testing. When self-trained and tested on Brown, it improved around 1% compared to direct testing on Brown. For WSJ, that increase was around 2%.

Also, we noticed that increasing seed size improves the relative performance by 2-4%, while increasing self-training size has negative effect on performance and reduces it around 2-3%.

When source and target domains were interchanged, overall trends remained same only. Like, domain-adaptation and increasing seed size did improve the performance; while increasing self training size reduced the performance. And there was significant drop from in-domain to out-domain testing. But if we run for more iterations, it maybe possible that training on Brown and direct testing on WSJ performs relatively better than other way around, that is training on WSJ and direct testing on Brown; as Brown has different genres and it can be more easily generalized.

If we compare our results with Reichart and Rappoport paper for the OI setting, we can see that it is coherent with their results. They also notice that with increase in manually annotated data (seed size here), there is a relative increase in performance. They also come to conclusion that unsupervised domain adaptation does help improve the performance compared to no self-training, which is given as baseline in their experiments.

## 5. Conclusion

In this assignment, problem of unsupervised domain adaptation was explored in context of neural dependency parsers. Self-Training approach was used to handle this task. Different sets of experiments were performed in order to analyze effects of changing seed size and self-training data. They were performed on WSJ and Brown data sets and Stanford Neural Dependency parser was used for the same.