CS 388 Natural Language Processing
Homework 3: Active Learning for Neural Dependency Parsing

Lihang Liu
EID: ll32632

1 Introduction

Learning Neural Dependency parsing from treebank data is a time-consuming task which requires not only time but also linguistic experts to do labelling. In this project, we are going to explore the algorithm proposed by [Hwa(2000)]. The technique used in this paper is called Active Learning to help relief the burden of human works on labelling. In this report, we will present several variants of active learning and experiments on the WSJ dataset.

2 Background

In this section, we will introduce the basic concept of Active Learning and Neural Dependency Parsing.

2.1 Active Learning

Active learning is a concept in the machine learning domain used to reduce the number of the training examples in the cases that the label of training examples are costly to obtain. This may be motivated from the learning process of humans. When humans learn some knowledge, they will focus and take more time on difficult examples in order to have a better understanding. In the machine learning domain, we use labelled data to train the parameters of some certain models. When trained with more difficult examples, the model will have a larger update towards the optimal or sub-optimal points.

However, we still need to evaluate the examples in order to decide which examples to select for training. In the worse case, we may need to evaluate all the examples. The selection method will have a large influence on the final performance which we will discuss more in the following content.

2.2 Neural Dependency Parsing

Neural Dependency Parsing proposed by [Chen and Manning(2014)] used neural networks to decide the next transition given current context. This serves as the machine learning model of our experiments. And each sentence is an data example. Note that we can also use another parser as long as there are ways to evaluate the data examples.

2.3 Selection Functions

How to evaluate and select the training examples from the unlabelled sentences is most important in this project, which have directly influence on the final performance. Here we compared 4 different selection functions:

- **Random**. We use a random number generator to assign random scores to the sentences and then use the sentences with top scores as the training examples.

- **Sentence Length**. In Natural Language, the longer the sentence, the lower its probability is. Also, longer sentences may also have more complex parse trees. In this method, we select the sentences with the top longest length.
- **Raw Parse Probability**. For a sentence with length n, each word requires a shift operation to push it from the buffer onto the stack, and requires a reduction operation to reduce it with a constituent. The code provided by Stanford Dependency Parser doesn't give probability to each transition so a softmax layer is used to obtain probabilities. To calculate the raw parse probability, we take product of the probability of each transition. Also, to reduce the influence of sentence length, we normalize the product by taking a $2n$-th root. The intuition is that with lower the raw parse probability, the neural network of the parser can have larger updates.
- **Margin Parse Probability**. This method is very similar to the Raw Parse probability. They all utilize the final outputs of the neural network. The difference is instead of take the product of probabilities of all transitions, margin parse probability is calculated by taking the product of the margin between probabilities assigned to two top transitions at every step. The same as raw parse probability, we normalized it by taking a $2n$-th root. The intuition here is that for one step, if the difference between the probabilities of the top two transitions is small, then it means that the top two transitions are ambiguous for the neural network to pick. Such sentences are more informational to train on.

3 Implementation

In this section, we will discuss the details about how to implement active learning based on the tools provided by the Stanford Dependency Parser: https://nlp.stanford.edu/software/nndep.shtml.

3.1 Data Preparation

3.1.1 Data Format

The data format used by the Stanford Dependency Parser is CoNLL-X format. In the CoNLL-X format, annotations are encoded in plain text files. All the sentences are put in one or more text files and each sentences are separated by a blank line. One sentence consists of one or more tokens and each token is represented on one line, which has 10 fields. So it's easy to extract some information from the CoNLL-X file. For example, we can count how many lines the sentence has to get its sentence length.

3.1.2 Generate the Data Files

To generate the data files, we make use of the following commands to put several CoNLL-X files into one CoNLL-X file:

```
$ cat /projects/nlp/penn-dependencybank/wsj-conllx/00/* > wsj_00.
    conllx
$ cat /projects/nlp/penn-dependencybank/wsj-conllx/{01,02,03}/* >
    wsj_01_03.conllx
$ cat /projects/nlp/penn-dependencybank/wsj-conllx/20/* > wsj_20.
    conllx
```

3.2 Active Learning

To implement active learning, we should be able to maintain a set of train examples and an unlabeled training pool. Each time the model will evaluate all the examples in the unlabeled training pool and select some from the pool to add into the train examples and re-train the model. However, current implementation of Stanford Dependency Parser only takes the data file as input arguments for training, such limitation requires us to update the data files for each iteration.

To make our active learning more flexible and get around such limitation, we create a new train() function which takes the Java instances of train examples directly as input and free our mind of managing the data files. More concretely, we create two Java class:

- **DataEntity**: Each instance represents for one sentence.
- **Dataset**:
  - Maintains a list of DataEntities.
  - $ArrayList < DataEntity > \ loadData(String\ datapath)$ takes a data file path as input and load all the sentences as DataEntities.
  - $void\ saved2file(String\ savedpath)$ writes the current list of DataEntities into a CoNLL-X text file.

With the above two class, we implement our own train() function as:

$$void\, train(Dataset\ trainData, String\ modelPath, String\ embeddingPath)$$

Within the new train() function, it will write the examples in the trainData into a temporary CoNLL-X file and call the Stanford Dependency Parser to train it. Now we don't need to concern about managing the data files anymore.

To calculate Raw Parse Probability, we make use of the property of ”RawScore” and ”MarginScore” provided in the demo code. Note that this is non-normalized log probabilities, so we divide them by the length of sentence to normalize.

To get the data examples from the train pool with the highest uncertainty, we use the bubble sort algorithm to sort the data examples by the scores return from a given selection function and then choose the top ones.

4 Experiments

4.1 Dataset

We conduct our experiments on the WSJ dataset. We use the first 50 sentences from the section 00 as the initial train data, the sentences from the section 01-03 as the unlabeled train pool, and the sentences from the section 20 as test data.

4.2 Experimental Setup

For each iteration, we use the selection function to choose a set of data examples whose words count approximate 1500 from the unlabeled train pool to the train data. We set the ”maxIter” of the Stanford Dependency Parser to be 500, in order to make sure the training has converged.

4.3 Evaluation metrics

To evaluate the performance, we use LAS score as the metric.

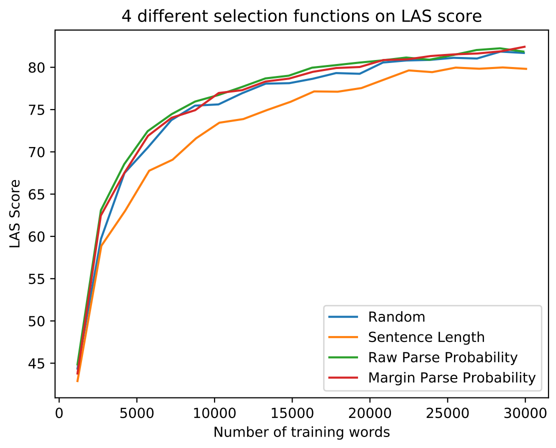

Figure 1: Comparison of 4 different selection functions on LAS score.

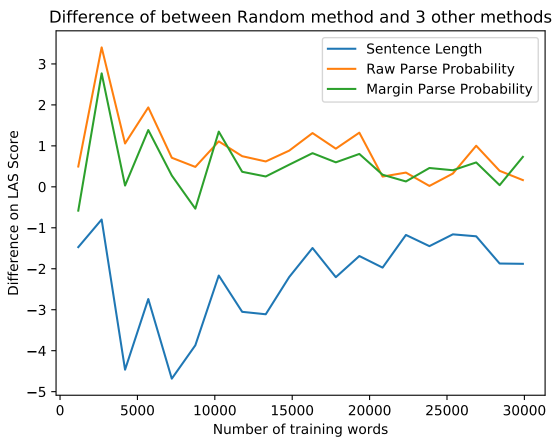

Figure 2: Comparison of the difference of between the Random method and 3 other methods.

4.4 Comparisons

We compared the 4 selections stated in Sec. 2.3: **Random**, **Sentence Length**, **Raw Parse Probability**, and **Margin Parse Probability**.

4.5 Results

We plot the changes of LAS score with the number of training words used in Fig. 1, as well the difference between the random method and 3 other selection methods in Fig. 2.

4.6 Discussions

1. Do the active learning methods perform better than random selection of training examples? Why?

   The active learning with Raw Parse Probability and Margin Parse Probability as the selection function performs better than random selection. Because the selected examples are with high uncertainty, so they provide a stronger supervised signal to the neural network. As a result, the parser learns better.

2. Does active learning help across the complete learning curve, or are there parts of the learning curve where it performs best? Why?

   From Fig.2, we can see the trend that for Raw Parse Probability and Margin Parse Probability, they both help more in the first several iterations and then gradually the difference on LAS score go down. Meanwhile, the difference between Random selection and the Sentence Length selection also becomes smaller. Because with more training examples, the average uncertainty of selected examples of different selection methods will become closer. In the worse case, all examples from the unlabeled train pool are selected for training then all selection method will have the same results. As a result, the difference on LAS score between random selection and other methods will become smaller as the number of training example grows.

3. How do the different methods for measuring uncertainty perform compared to each other? Try to explain any observed differences between methods.

   Raw Parse Probability seems slightly better than Margin Parse Probability. One explanation may be on extreme cases of very low probabilities. For Raw Parse Probability, if there are extremely low probability, then this should give a larger update on the model. However, for Margin Parse Probability, the extremely low probability will occur when the top two are very close, but these two may be both with high probability, which give less updates than that of Raw Parse Probability.

The Sentence Length method is even worse than random selection. Because although with longer sentence, you have lower uncertainty, but the normalized uncertainty may be low. While to have the same number of train words, it ends up with less number of sentences. So it performs worse.

4. How do your results compare to those presented by Hwa (2000)? Try to explain any differences.

   In Hwa's experiments, the sentence length method performs better than the random selection while in our experiment, the sentence length method is worse. This is because in Hwa's experiment, they choose a fix number of sentences for each iteration, so the sentence length method will actually provide more train words than random selection. However, in our case, the sentence selection method will provide less train sentences for training. Consider the difference in the number of train words, our results are difference from Hwa's results.

References

[Chen and Manning(2014)] Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750.

[Hwa(2000)] Rebecca Hwa. 2000. Sample selection for statistical grammar induction. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 45–52. Association for Computational Linguistics.