CS5740 Assignment 5 Github Group: technologica

Zexi Liu
zl558@cornell.edu

Yating Zhan
yz2237@cornell.edu

Chaoping Deng
cd446@cornell.edu

1 Introduction

In this assignment, we built a dependency parser using transition-based dependency parsing, the specific method we used is MaltParser (Nivre et al., 2007) with labeled arcs. Also, we trained a simple neural network as classifier to make decision of which action to perform at each given configuration (Chen and Manning, 2014). With careful hyper-parameters tuning, finally we achieve labeled attachment score 87.95% and unlabeled attachment score 89.35% on test set.

Section 2 introduces the MaltParser and our neural network structure. Section 3 talks about details of pre-processing especially how to generate the configurations we fed into our neural network, hyper-parameters tuning and experiments implementations. Section 4 shows experiment results and analysis. Section 5 focuses on error analysis and conclusions from experiments.

2 Approaches

2.1 Transition-based Dependency Parsing

We used transition-based dependency parsing to generate dependency parse tree, more specifically, we used MaltParser. In detail, the parser has four components.

- A stack σ, which starts with the ROOT symbol. The top of stack is at the most right side.
- A buffer β, which starts with the input sentence. The buffer is also a stack in the sense of data structure, whose top is at the most left side.
- A set of dependency arcs with label A, which starts off empty. The arcs are labeled and directional.
- A set of actions. There are three types of actions, LEFT-ARC(LABEL), RIGHT-ARC(LABEL) and SHIFT.

We called a configuration C = (S, B, A), which consists of a stack S, a buffer B, and a set of labeled dependency arcs A. A configuration c is terminal if the buffer is empty and the stack contains the single node ROOT, and the parse tree is given by the set of arcs of the terminal configuration. It worth mentioned that MaltParser is an arc-standard system. The three types of labeled actions are:

- LEFT-ARC(LABEL) Pop an element w_i from stack, pop and element w_j from buffer, create an left arc from w_j to w_i with label L, and push w_j back to buffer.
- RIGHT-ARC(LABEL) Pop an element w_i from stack, pop and element w_j from buffer, create an right arc from w_i to w_j with label L, and push w_i back to buffer.
- SHIFT Pop w_i from buffer and push it back to stack.

Assuming there are N labels, in total we have 2 * N + 1 possible actions.

To derive the dependency parse tree, the parser tries to predict a transition sequence from an initial configuration to some terminal configuration. The actions in each transition step is chosen greedily, that is, using a classifier to predict the current transition action based on features extracted from the current configuration.

Figure 1 is an example from (Chen and Manning, 2014) illustrating how MaltParser works by showing every action for each transition step and the configuration on each step.

2.2 Neural Network as Classifier

From raw CoNLL data, we first reconstructed the configurations sequence using MaltParser we introduced in previous part, the detail of this reconstruction will be shown in part 3.1 and part 3.2. Given the configurations, we trained a simple forward neural network as classifier to decide which action should be performed.

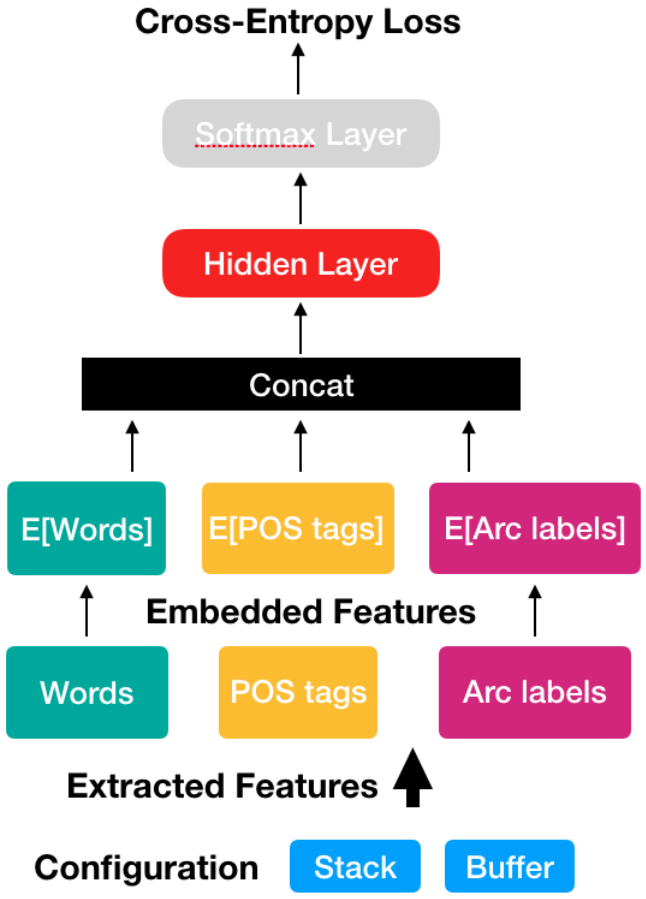
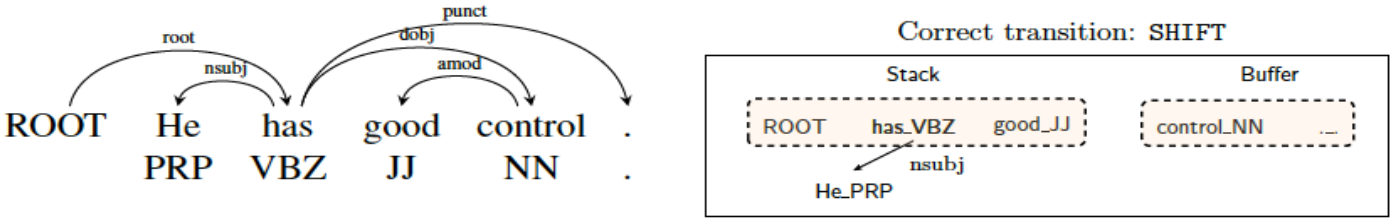


Figure 2: Neural network architecture

The input to the neural network is some word token features extracted from the configuration. The detail of feature selection will be shown in part 3.3. And then we embedded the token features as vectors and concatenate them together and fed the long vector to next layer. In this way, we trained our embedding of tokens in the end-to-end neural network.

Our neural network has only one fully connected hidden layer. And we applied softmax on output layer to generate probability over all the possible actions. The loss function is cross-entropy. The architecture is shown in Figure 2.



Transition	Stack	Buffer	A
	[ROOT]	[He has good control .]	∅
SHIFT	[ROOT He]	[has good control .]	
SHIFT	[ROOT He has]	[good control .]	
LEFT-ARC(nsubj)	[ROOT has]	[good control .]	A∪ nsubj(has,He)
SHIFT	[ROOT has good]	[control .]	
SHIFT	[ROOT has good control]	[.]	
LEFT-ARC(amod)	[ROOT has control]	[.]	A∪amod(control,good)
RIGHT-ARC(dobj)	[ROOT has]	[.]	A∪ dobj(has,control)
...	...	...	...
RIGHT-ARC(root)	[ROOT]	[]	A∪ root(ROOT,has)

Figure 1: An example of transition-based dependency parsing. Above left: a desired dependency tree, above right: an intermediate configuration, bottom: a transition sequence of the arc-standard system.

3 Experimental Setup

3.1 Data Overview

We trained our model using information from the train.conll data file which contains 600000 sentences and tune our model using the dev.conll dataset, which contains 200000 sentences. We then did an one-time run with the test.conll data and computed the UAS and LAS score. The train,development and test datasets all follow the convention of a standard CoNLL file.The data samples we trained on and predicted follows a format derived from the Conll data files,which will be illustrated in the following sections. In the end we transformed our format back to CoNLL format to generate the prediction CoNLL files.

3.2 Pre-processing

We transformed data contained in the CoNLL files into vectors that could be fed into a feed-forword neural network.We followed two steps:First,we created vocabularies for word features,POS features, dependency label features and actions. We only generated vocabularies for words in sentences which are projective. Second, we translated all CoNLL data files into data files containing feature vectors based on dictionaries we generated. We labeled words appearing less than 2 times as "UNK" to accommodate unknown words handling in dev and test dataset. We also added "ROOT" to indicate the root token of POS tags or words and 'NULL' to represent feature positions where no valid feature value could be computed.

3.3 Feature Generation

From ConLL data file,we parsed each data sentence into configurations containing a stack and a buffer. We then extracted a set of features and feed such feature vector into our neural network based on the method described in (Chen and Manning, 2014). We grouped such features into 3 different sources: words, POS tags and arc labels and set feature token length to: 20, 20 and 12 respectively. This provided us with an one-hot vector of length 52 as input to the neural network. Specifically, we designed the feature vector as the following according to (Weiss, 2015):

- s_i: the ith element on the stack
- b_i: the i'th element on the buffer
- $lc1(si)$, $lc2(si)$: the first and second left-child of word si
- $rc1(si)$, $rc2(si)$: the first and second right-children of si
- i= 1,2,3,4

Word feature:
word(s1),word(s2),word(s3),word(s4),
word(b1),word(b2),word(b3),word(b4),word(lc1(s1)),
word(lc1(s2)),word(lc2(s1)),word(lc2(s2)),
word(rc1(s1)),word(rc1(s2)),word(rc2(s1)),word(rc2(s2)),
word(lc1(lc1(s1))), word(lc1(lc1(s2))),word(rc1(rc1(s1))),
word(rc1(rc1(s2)))

POS and dependency label feature:
tokens from configuration vectors with paddings of "null" tokens.

3.4 Hyper parameters

We used greedy search to find the best hyper parameters set for the model. There are seven main hyper parameters that we are tuning in the experiment: word embedding dimension, part of speech (POS) tags embedding dimension, dependency embedding dimension, hidden layer dimension, activation function, dropout rate, and optimizer.

We conducted the greedy search of the sub-optimal set of hyper parameters by starting with finding the best word embedding dimension with the POS embedding dimension of 64, dependency embedding dimension of 64, hidden layer dimension of 100, batch size 100, dropout rate of 0.2, x^3 as the activation function and, SimpleSGDTrainer as the optimizer. After we found the best word embedding dimension with the above set of hyper parameters, we used it as the word embedding for all the following experiments. We continue on to find the best POS embedding dimension and use it to find other parameters. We continue by until we find went through all the hyper parameter and pick the parameters that perform best when all others hold the same. We use Labeled Attachment Score to determine the best parameters and the study is conducted on a randomly selected small

training set of 10000 sentences. We used a small training set because it help to speed up the hyper parameter tuning process.

3.5 Stop Criteria

We stop the training by identifying saturation for Labeled Attachment Score. Specifically, for every epoch, if the Labeled Attachment Score doesn't improve for more than 0.01, we stop the training process.

4 Results and Analysis

4.1 Hyper Parameters

- Word Embedding Dimension:

	64	128	256	512
LAS	75.96%	75.6%	75.75%	75.7%
UAS	78.82%	78.36%	78.45%	78.52%

Table 1: Word embedding dimension results

As shown in Table 1, both Labeled Attachment Score and Unlabeled Attachment Score achieve the highest when the word embedding dimension is 64.

- Part of Speech Tags Embedding Dimension:

	32	64	128	256
LAS	75.48%	75.96%	75.44%	75.23%
UAS	78.37%	78.82%	78.26%	78.09%

Table 2: POS embedding dimension results plot

As shown in Table 2, both Labeled Attachment Score and Unlabeled Attachment Score achieve the highest when the POS tags embedding dimension is 64.

- Dependency embedding dimension:

	32	64	128	256
LAS	75.96%	76.34%	75.52%	75.72%
UAS	78.82%	79.25%	78.35%	78.63%

Table 3: Dependency embedding dimension results plot

As shown in Table 3, both Labeled Attachment Score and Unlabeled Attachment Score achieve the highest when the dependency embedding dimension is 64.

- Hidden layer dimension:

	50	100	200	500
LAS	73.17%	75.64%	76.79%	76.82%
UAS	76.63%	78.36%	79.41%	79.41%

Table 4: Hidden Layer dimension results plot

As shown in Table 4, both Labeled Attachment Score and Unlabeled Attachment Score achieve the highest when the hidden layer dimension is 200.

- Activation function:

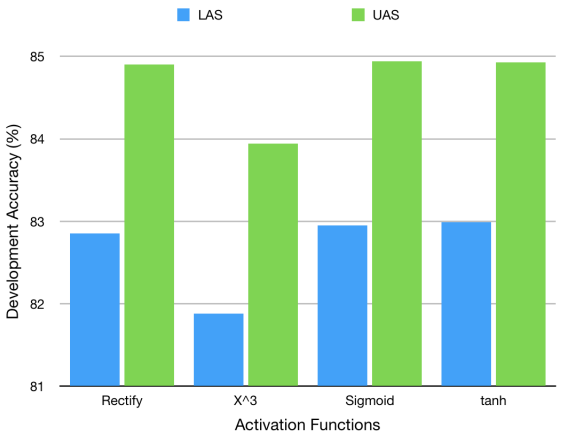


Figure 3: Activation Functions results plot

As shown in Figure 3, Labeled Attachment Score achieves the highest when using tanh as the activation function while the Unlabeled Attachment Score achieves the highest when using sigmoid as the activation function. We chose tanh as the activation function for the following experiments.

- Dropout Rate:

	0.0	0.2	0.5
LAS	82.99%	81.04%	77.61%
UAS	84.93%	83.08%	77.78%

Table 5: Dropout rates experiment results plot

As shown in Table 5, both Labeled Attachment Score and Unlabeled Attachment Score achieve the highest when we don't use any dropouts.

- Optimizer:

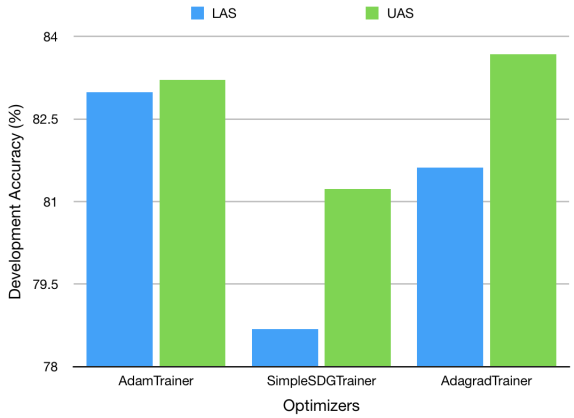


Figure 4: Optimizers experiment results plot

As shown in Figure 5, Labeled Attachment Score achieves the highest when using AdamTrainer as the optimizer while the Unlabeled Attachment Score achieves the highest when using AdagradTrainer. We used AdamTrainer for our final model.

4.2 Final Model Performance

We used the following hyper parameters for our final model on the whole training data-set: word embedding dimension: 64, POS embedding dimension: 32, dependency embedding dimension: 32, hidden layer dimension: 100, batch size: 100, activation function: rectify, dropout: 0.0, trainer: AdamTrainer. And the test labeled attachment score is 87.95% and the test unlabeled attachment score is 89.35%.

5 Error Analysis and Conclusion

5.1 Error Analysis

We went through some error examples and categorize them as following.

1. Wrong verb to object connection: Example: When St. Martin of the Broadway League says, I dont think theres ever been a more diverse grouping of shows as now, shes not kidding. Error: grouping should be the object of think but our model thinks its the object of is.

2. Wrong dependency relationship: Example: To go beyond that, you either need to pay on your own -LRB- either out-of-pocket or with supplemental insurance -RRB-. Error: 2nd either is used to explain -LRB- but our model think its used like the first either to explain the word after it. Error: -RRB- has dependency relationship with both out-of-pocket and supplemental insurance, but the more direct dependency relationship is with supplemental insurance.

3. Preposition attachment error: Example: Vikings Coach Brad Childress described Favres play as a gutty, gutty performance. Error: as is used for play but our modes thinks its used for described, this might due to a high frequency of the phrase describe as.

   Example: During that time Mrs Dark travelled to France on several occasions, unaware of her status as a convicted criminal. Error: as should be used for status, but our model attached it to unaware.

   Example: Dr Thompson began researching stripes after coming across Ancient Greek columns at the 550BC Temple of Hera in Paestrum, while on holiday. Error: after should be attached to the action researching but our model thinks that its attached to began.

4. Wrong ROOT: Exampel: That is made up; they are looking to generate opinion. Error: the ROOT should be looking but our model thinks the ROOT is made. This might due to a punctuation error in the sentence.

5. Ambiguous answer: Example: Raising the China threat is an oft-used DPP electoral tactic. Error: the ROOT is tactic in dev.conll while our model thinks that the ROOT is Raising. But the gold label in dev.conll is questionable in this case.

   Example: More than four times. Error: the than is attached to times in the gold label, and our model thinks its attached to four. Since four times is a phrase, attaching to either one word should be correct.

To analyze the labeled attachment accuracies, we extracted the first 1 million words in the dev dataset(conll file), given that we predict the attachment arcs correctly, the graph shows dependency labels with lowest predicted f1 score.Moreover,we are interested to see what each of the labels are mostly classified to:

Dependency Label	Most Frequent Misclassification
Csubj	ROOT
Dep	ROOT, ccomp, nsubj
Iobj	dobj
Parataxis	ROOT, dep

Table 6: Most misclassified dependency labels

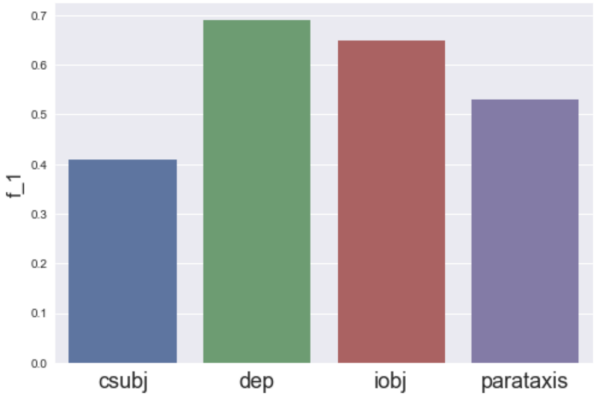


Figure 5: F1 score of predicted dependency labels

csubj stands for a clausal syntactic subject of a clause, which is complex in nature due to its complicated structure.Similarly ,parataxis is also associated with the placing of clauses or phrases one after another.These two labels are easily mixed with ROOT due to its grammatical structure. Dep indicates the case when the system is unable to determine a more precise dependency relation between two words. The machine would not be able to learn this well, since it contains various strange grammatical structures.

IOBJ, on the other hand, is easily mixed with dobj. This is not surprising since iobj stands for indirect object and dobj stands for direct object. For the sentence, his manager gave him a raise, it would produce two relationship iobj(gave him) and dobj(gave raise).

5.2 Conclusion

From our experiment we found that the feed-forward neural network is a powerful tool for stand-arc dependency parsing. It requires careful feature generation as well as parameter tuning to achieve decent performance. A more sophisticated feature structure as well as deeper neural networks are likely to further improve the model performance.

References

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.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering* 13(2):95–135.

Chris Alberti Michael Collins Slav Petrov David Weiss. 2015. Structured training for neural network transition-based parsing. In *In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*. volume 1, pages 323–333.