HW3: Dependency Parsing

Jayanth Vishwas
CSCI 662
USC ID: 3226534944
thalari@usc.edu

Abstract

A dependency parser analyzes the sentence structure by examining the relationships between words. The majority of dependency parsers classify based on millions of sparse features. However, these features are poorly generalizable and also reduce parsing speed. In this assignment, a neural dependency parser has been implemented which is similar to the work of Danqi Chen. In the work of ”A Fast and Accurate Dependency Parser using Neural Networks” by Danqi Chen, a neural network classifier on transition-based dependency parser is proposed. Which uses a small number of features and has been proven to be faster compared to the legacy parsers. In this assignment, a similar transition-based neural parser has been implemented based on the arc eager shift-reduce method. The dataset used is English dependency trees converted from Penn Tree Bank which is in CoNLL format. The neural classifier can classify with 93 and 94 percent accuracies on validation and training datasets respectively.

1 Introduction

Dependency Parsing is a traditional grammatical exercise in which a text is broken down into its constituent parts, as well as the function, relationship, and syntactic relation of each part of speech. In a sentence, every word except one has a parent word that represents the smallest syntactic unit the word is not the head of. A word may have zero or more words that regard it as their head. Every word in a phrase has a parent word that indicates the lowest syntactic unit of which the word is not the head. A word can have zero or more words that consider it to be their head. The word which has no head is the root of the sentence. Each parent-child relationship may be labeled with the role of the phrase in which the child is the head. There are 37 such label types in universal dependencies.

In the past, many feature-based discriminative dependency parsers have been achieved with considerable speed. However, these parsers suffer from the use of millions of feature weights, which are poorly estimated, and the use of many incomplete feature templates. In this assignment, dense features are considered over sparse indicator features which can reduce the sparsity and provide a good base for building the features. A neural network classifier is trained to make parsing decisions within a transition-based dependency parser. The network learns the arc eager shift-reduce transition to produce the transition along with the arc. The features include words, parts of speech tags, and dependency labels from the dense representation of words.

2 Arc-Eager Dependency Parser

The transition-based method begins with the definition of a transition system or state machine to map the set to its dependency graph. The analysis process is then reduced to derive a model that predicts the next transition, given the current state and the previous transition history. In this work, the greedy parsing has been experimented which uses a classifier to predict the correct transition based on features extracted from the configuration.

Given a set L of dependency labels, we define a dependency graph for a sentence x = w1 , w2,..wn as a labeled directed graph G = (Vx ,A), consisting of a set of nodes Vx = 0,1, . . . , n, where each node i corresponds to the linear position of a word wi in the sentence, plus an extra artificial root node 0, and a set of labeled arcs A Vx × L ×Vx , where each arc (i, l, j) represents a dependency with head wi , dependent wj , and label l. In order for a dependency graph to be well-formed, we usually require it to be a dependency tree, which is a directed spanning tree rooted at the node 0.

Arc-eager dependency parsing changes the ARC-RIGHT action so that right dependents can be attached before all of their dependents have been found. Rather than removing the modifier from both the buffer and stack, the ARC-RIGHT action pushes the modifier on to the stack, on top of the head. Because the stack can now contain elements that already have parents in the partial dependency graph, two additional changes are necessary: a) A precondition is required to ensure that the ARC-LEFT action cannot be applied when the top element on the stack already has a parent in A. b) A new REDUCE action is introduced, which can remove elements from the stack if they already have a parent in A.

Projectivity: A sentence is classified as projective when there is a cross section of arcs between two words. The arc-standard and arc-eager transition systems are guaranteed to produce projective dependency trees, because all arcs are between the word at the top of the stack and the left-most edge of the buffer.

Feature extraction: a) the first three words on the stack and the buffer (and their POS tags) b) the words, POS tags, and arc labels of the first and second leftmost and rightmost children of the first two words on the stack. c) the words, POS tags, and arc labels of leftmost child of the leftmost child and rightmost child of rightmost child of the first two words of the stack.

3 Neural Parser

3.1 Model

The number of nodes in the hidden layer is a hyper parameter which we can pick and choose. How many nodes and how many hidden layers to pick is something to be decided after tuning them for various values. A network having multiple hidden layers is what's normally called a deep neural network. However, it does not necessarily improve the model's performance by adding multiple layers. Some of the hyperparameters in a neural network are epoch, batch size, hidden layers, input size, learning rate. To improve the model learning we use activation functions which help the neurons to learn the complex pattern from the data. The activation function used is tanh.

The neural parser has two hidden layers and two outputs (one for classifying the transition and other for the arc). Each input is a d-dimensional vector. The words, pos-tags and arc labels will be transformed into embedding vectors before it goes to the hidden layer. The features are selected from the stack and buffer and are converted to embeddings. The feature sets are denoted as S^w, S^t, S^l respectively. The outputs of the hidden layer is mapped to tanh activation function.

h = tanh($W_1^w x^w$ + $W_1^t x^t$ + $W_1^l x^l$ + b1)

A softmax layer is finally added on the top of the hidden layer for modeling multi-class probabilities.

3.2 Embeddings

The embeddings for words, pos-tags and arc labels are obtained by the help of nn.embedding from pytorch. The vocabulary index of word, pos and arc labels are passed to the layer with predefined dimension for the size of the vector. The resulting output is the d-dimension vector for each of the item in the vocabulary of word, pos and arc label respectively.

3.3 Training

The parser generates the training samples and the final objective is to minimize the cross entropy loss.

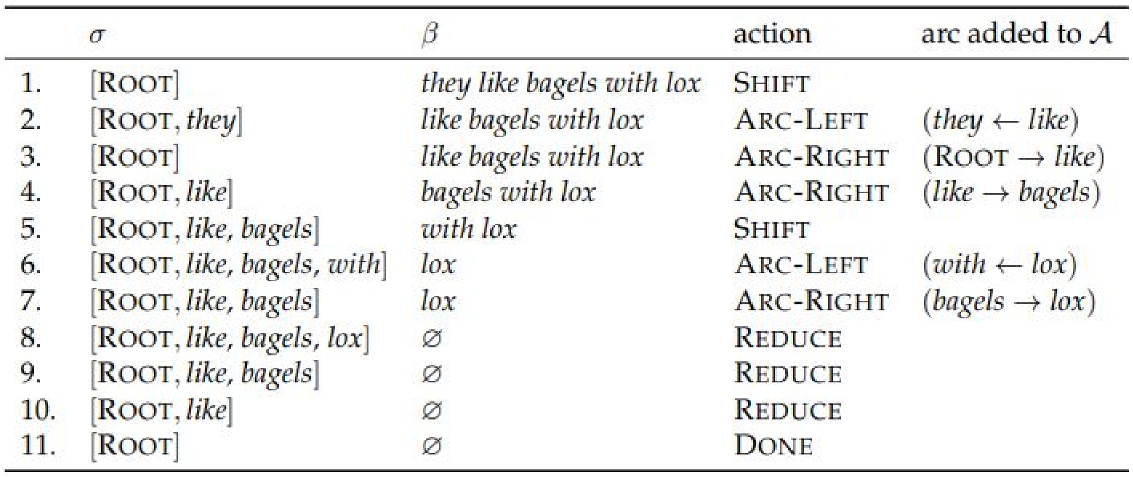
L(θ) $= -(\sum(log\mathrm{p}_{ti})$ + $\sum(log\mathrm{p}_{mj}))$

where $\sum(log\mathrm{p}_{ti})$ is the loss of classification of arc labels and $\sum(log\mathrm{p}_{mj})$ is the loss of classification of transition labels.

The size of embedding vector is 50 and the vectors are generated from scratch. The learning rate is set to 0.01 and min batch adam optimizer is used with adaptive learning rate. We set the values before training and observe the performance for each of the value. Grid search is used in this experiment to find the optimal hyper parameter for the machine learning model. Hyper parameter tuning is performed on learning rate, batch size, activation functions.

3.4 Parsing

The parsing is done in the final step to convert the sentence into dependency tree. At each step, we extract all the corresponding word, POS and label embeddings from the current configuration c, compute the hidden layer and pick the transition with the highest score using argmax. Intially the arc labels in the feature set are zero and the neural parser predicts the arc labels and the transition. Based on the transition, the parent ID and childrens are updated for the respective words. This in turn will be used to predict the next transition and the arc label. This sequential operation results in the dependency tree for each sentence. The parser reduces feature



	σ	β	action	arc added to $\mathcal{A}$
1.	[ROOT]	*they like bagels with lox*	SHIFT	
2.	[ROOT, *they*]	*like bagels with lox*	ARC-LEFT	(*they* ← *like*)
3.	[ROOT]	*like bagels with lox*	ARC-RIGHT	(ROOT → *like*)
4.	[ROOT, *like*]	*bagels with lox*	ARC-RIGHT	(*like* → *bagels*)
5.	[ROOT, *like, bagels*]	*with lox*	SHIFT	
6.	[ROOT, *like, bagels, with*]	*lox*	ARC-LEFT	(*with* ← *lox*)
7.	[ROOT, *like, bagels*]	*lox*	ARC-RIGHT	(*bagels* → *lox*)
8.	[ROOT, *like, bagels, lox*]	∅	REDUCE	
9.	[ROOT, *like, bagels*]	∅	REDUCE	
10.	[ROOT, *like*]	∅	REDUCE	
11.	[ROOT]	∅	DONE	

Figure 1: Arc-eager derivation of the unlabeled dependency parse for the input they like bagels with lox.

Dataset	**projective**	**non- projective (removed)**
Train	**39796**	36
Dev	**1699**	1

Table 1: Number of Projective trees

Parser	**UAS**	**LAS**
Arc-eager	**0.361**	0.341

Table 2: UAS and LAS scores

generation time by doing matrix addition and multiplication operations and by pre-compiling the top 10, 000 words for each position chosen from S_w. The pre-computation trick increases the speed of neural network parser.

4 Experiments and Results

The experiments are done on English Penn Treebank datasets. The number of non projective sentences are 36 for the train.orig.conll which is over 99 percent of the sentences are projective. The number of hidden layer units is 500 in the first layer and 300 in the second hidden layer and the learning rate used is 0.01 with batch size of 512. Activation function used is tanh. The neural classifier is able to classify with 93 ad 94 percent accuracies for validation and training dataset (Table3). And the parser is able to get a LAS (labeled attachment score) 0.346 and UAS (Unlabelled Attachment score) 0.361 (Table 2.

Model	**Train Acc.**	**Val Acc.**
Arc-eager neural classifier	**0.94**	0.93

Table 3: Accuracy

Model	**Train loss**	**Val loss**
Arc-eager neural classifier	**0.367**	0.40

Table 4: loss

5 Conclusion

In this work, a neural dependency parser is presented inspired by the work of Danqi Chen. Experiments show using dense vectors the dependency parser outperforms other greedy parsers. The words, pos-tags and arc labels are represented as dense vectors and the model predicts the arc label and transition with good accuracy and speed.

References

[1] A Fast and Accurate Dependency Parser using Neural Networks

[2] Eisenstein NLP

[3] Dependency Parsing Sandra Kübler, Ryan McDonald, and Joakim Nivre