Multi-Tagging for Transition-based Dependency Parsing

Alexander Whillas

School of Computer Science and Engineering
University of New South Wales, NSW, Australia
arbw870@cse.unsw.edu.au

Abstract. This project focuses on a modification of a greedy transition based dependency parser. Typically a Part-Of-Speech (POS) tagger models a probability distribution over all the possible tags for each word in the given sentence and chooses one as its best guess. This is then pass on to the parser which uses this information to build a parse tree. The current state of the art for POS tagging is about 97% word accuracy, which seems high but results in a around 56% sentence accuracy [Manning, 2011]. Small errors at the POS tagging phase can lead to large errors down the NLP pipeline and transition based parsers are particularity sensitive to these types of mistakes.
A maximum entropy Markov model was trained as a POS multi-tagger passing more than its 1-best guess to the parser which was thought could make a better decision when committing to a parse for the sentence. This has been shown [Curran et al., 2006] to give improved accuracy in other parsing approaches. We shown there is a correlation between tagging ambiguity and parsers accuracy and in fact the higher the average tags per word the higher the accuracy.

1 Introduction

In Natural Language Processing (NLP) two fundamental tasks are part-of-speech (POS) tagging, assigning lexical categories to words in a sentence such as Verb or Noun, and sentence parsing which is preforming a grammatical analysis on the sentence the output of which is typically a parse tree representing the structure of the words. Typically the POS taggers output is feed into the parser as input creating an NLP pipeline.

The parse trees that the parser produces depends on the grammar formalism that is being used to model the language. These can be dived into two main categories: phrase-structure grammars [Chomsky, 1957] which adhere to the constituency relation and dependency grammars [Tesnière, 1959] which are based on the dependency relation (without intermittent constituents in the tree).

Dependency parsing produces a connected, directed acyclic graph (DAG)[1] with nodes that are all the words in a sentence, typically with the addition of an artificial ROOT node, and with arcs that are the dependency relations. There are three types of parser methods for producing these trees: transition based methods, similar to finite state automata; graph based methods[2] which find a maximum spanning tree (MST); and grammar-based methods, which use predefined projective dependency grammar similar

[1] aka arborescence
[2] or 'ArcFactored models'

to context-free grammars and thus parsing becomes a constraint satisfaction problem. [Kübler et al., 2009]

2 Maximum Entropy multi-tagger

Maximum entropy Markov models (MEMM) are conditional probabilistic sequence models. Given an observation sequence they produces probability distributions over possible label sequences as output [3].

The conditional probabilities take the form $p(y|x)$ where y is a POS tag and x is a local context in which y occurs. p has the log-linear form:

$$p(y|x;v) = \frac{e^{\vec{v}\cdot\vec{f}(x,y)}}{\sum_{y'\in\mathcal{Y}} e^{\vec{v}\cdot\vec{f}(x,y')}}. \quad (1)$$

where $\vec{f}(x,y)$ is a vector of indicator feature functions and $\vec{v}$ is a vector of feature weights. The divisor is a normalisation term to ensure a proper probability distribution.

Feature functions are contextual predicates that identify the presence of selected features that are estimated to be significant factors in predicting the tag for the current word. They are typically boolean valued functions returning 1 if the feature is present and 0 otherwise. A typical feature function might look like:

$$f_k(x,y) = \begin{cases} 1 & \text{if } word(x) = \texttt{the} \wedge y = \texttt{DET} \\ 0 & \text{otherwise} \end{cases}$$

where the predicate $word(x) = \texttt{the} \wedge y = \texttt{DET}$ is true if the current word is "the" and the predicted POS tag is "DET". These feature functions are generated from a template from every word position in every sentence and typically number in the hundreds of thousands even for small training data sets. Feature templates very similar to those used by [Ratnaparkhi et al., 1996] were used and are listed in table 1 with the addition of up to four suffixes.

with $w_i = fox$ and the previous two tags are previous tags are RB and JJ, and next two are VBZ and RB. Usually when moving from left to right or right to left only the previous tags from the last two steps are available. In that case the features that require tags ahead of the direction the tagger is moving are just not present.

In some cases words which have a high enough frequency and little to no POS tag ambiguity e.g. "the" for example, tags are looked up before inference begins and thus their POS tags are always available. This was used to speed up the forward-backward algorithm only consider the looked up tags instead of all the possibilities.

Parameter estimation of the large number of feature weights, v above, can be done using an off the shelf gradient accent method. The LBFGS algorithm, a quasi Newtonian method, was chosen as it handles large gradient vectors and is fast as it approximates the Hessian matrix from previous steps instead of recalculating it every time.

[3] Even thought a MEMM is much slower to train and tag than a Perceptron it does have the advantage of producing a probability distribution for all tags at all points in a context and this allows us to multi tag

Feature	Template	Example	
Current word	w_i	& t_i	w_i=fox & NN
Previous word	w_{i-1}	& t_i	w_{i-1}=brown & NN
Word two back	w_{i-2}	& t_i	w_{i-2}=quick & NN
Next word	w_{i+1}	& t_i	w_{i+1}=jumped & NN
Word two ahead	w_{i+2}	& t_i	w_{i+2}=over & NN
Bigram features	w_{i-2}, w_{i-1}	& t_i	w_{i-2}=quick, w_{i-1}=brown & NN
	w_{i-1}, w_i	& t_i	w_{i-1}=brown, w_i=fox & NN
	w_i w_{i+1}	& t_i	w_i=fox $w_{i+1} = jumped$ & NN
	w_{i+1}, w_{i+2}	& t_i	w_{i+1}=jumped, w_{i+2}=over & NN
Bigram tag features	t_{i-1}	& t_i	t_{i-1}=JJ & NN
	t_{i-2}	& t_i	t_{i-2}=RB & NN
	t_{i+1}	& t_i	t_{i+1}=VBZ & NN
	t_{i+2}	& t_i	t_{i+2}=RB & NN
Tri-gram tag	t_{i-2}, t_{i-1}	& t_i	t_{i-2}=RB, t_{i-1}=JJ & NN
	t_{i-1}, t_{i+1}	& t_i	t_{i-1}=JJ, t_{i+1}=VBZ & NN
	t_{i+1}, t_{i+2}	& t_i	t_{i+1}=VBZ, t_{i+2}=RB & NN

Table 1: Feature templates. Examples using:
`The quick brown` **`fox`** `jumped over the lazy dog`

To avoid over-fitting the training data a regularisation prior term was added to the objective function.

Forward-backward algorithm [Rabiner, 1989] is used to generate the probability distributions over all the tags for each word in a sentence. It finds the most probable state for any given point in a sequence but not necessarily the most probable sequence of states[4]. It does this by combining the probabilities of all the state sequences leading up to a point in a sequence, the forward probabilities, and then all the sequences after the point starting from the end, the backward probabilities. The brut force time complexity of combining every state transition is $\mathcal{O}(n^k)$ where n is the length of the sequence and k is the number of states[5]. The forward-backward algorithm is a dynamic programming algorithm and manages it in $\mathcal{O}(nk^2)$.

3 Data

The Penn Tree Bank (PTB) corpus which has become the de facto standard data set for comparing POS tagging and parser performance. The problem with this data is that it is not freely available to the general public and the process of obtaining it via official channels is also not straight forward. This is a problem for new and underfunded

[4] See the Viterbi algorithm for most probable state sequence.
[5] or tags in this case which was 50

researchers. Fortunately a free alternative corpus has just been released. albeit a third the size, the development was done with this corpus.[6]

The corpus used was the English part of the **Universal Dependencies** version 1.1 [Agić et al., 2015] which was in turn build from the English Web Treebank [Silveira et al., 2014]. "The corpus comprises 254,830 words and 16,622 sentences, taken from various web media including weblogs, newsgroups, emails, reviews, and Yahoo! answers". Web corpus usually have loser and more diverse grammar than commercial publications such as the Wall Street Journal and so are perhaps more challenging for NLP tasks.

3.1 Handling unknown words

A dictionary of all the words seen in the training data is built during train. Those with a high enough frequency and which are not very ambiguous in terms of POS tag frequency are kept in a list along with the tag. This is then used for fast lookup.

Unknown words seen at testing time are then mapped to one of a dozen pseudo-words which follow the "word features" of [Bikel et al., 1999]. This is has the effect of closing the vocabulary which means that at test time every word would have been seen at least once.

Word Feature	Example Text	Intuition
twoDigitNum	90	Two-digit year
fourDigitNum	1990	Four digit year
containsDigitAndAlpha	A8956-67	Product code
containsDigitAndDash	09-96	Date
containsDigitAndSlash	11/9/89	Date
containsDigitAndComma	23,000.00	Monetary amount
containsDigitAndPeriod	1.00	Monetary amount, percentage
otherNum	456789	Other number
allCaps	BBN	Organisations
capPeriod	M.	Person name initial
firstWord	*first word of sentence*	No useful capitalisation information
initCap	Sally	Capitalised word
lowerCase	can	Uncapitalised word
other	,	Punctuation marks, all other words

Table 2: Pseudo-words classes that unseen words are mated to, in order of precedence as per [Bikel et al., 1999]

4 Multi-tagging

The multi-tagging approach taken here is heavily influenced by [Curran et al., 2006] in which they used a multi tagging approach in a POS-tagger and an intermediate super-

[6] Not having access to the PTB means that results are not comparable to previous work but this wasn't too important as the results only had to compare to themselves.

tagger for a CCG [7] parser and got a 1.6% word accuracy and almost 20% sentence accuracy improvement by adding an extra 1 tag in 20 words.

We adopted their tag selection process taking all tags for a word with in a factor γ of the most probable tag for each word.

$$\mathfrak{C}_i = c|P(C_i = c|S) > \gamma P(C_i = c_{max}|S) \tag{2}$$

where $\mathfrak{C}_i$ is the set of all tags assigned to the ith word, C_i is the random variable of the tag of the ith word, c_{max} is the highest probability of a tag of the ith word, and S is the sentence. This ensures that if there is very little ambiguity as to the tag for a given word, its c_{max} is much higher than all other candidates, then no extra tags will be added.

To calculate probability distribution across all tags for a word i, $P(C_i = c|S)$, the forward-backwards algorithm, typically used in Markov models, was used [Charniak et al., 1996].

Real-valued features were used as per [Curran et al., 2006] which gave them the best performance. The perceptron dependency parsers model uses features very similar to those used by the MEMM except instead for defining predicates over the lists of words and tags features are defined over the parse configurations i.e. the stack, buffer and arc set (see 'Dependency parsing' bellow for definition of these).

Thus the standard boolean features of the parser were extend to use real valued features e.g.

$$f_k(x, y) = \begin{cases} p(POS(x) = \texttt{NN}) & \text{if } y = \texttt{NP} \\ 0 & \text{otherwise} \end{cases}$$

5 Dependency parsing

A dependency parser can be thought of as an abstract machine[8] which takes as input a sequence of words and mapping them to a matching set of head indices (see Table 3). Typically an artificial ROOT word is added which takes the first (zero index).

indices:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
words:	ROOT	I	have	never	hated	a	man	enough	to	give	his	diamonds	back	.
heads:	-	4	4	4	0	6	4	4	9	4	11	9	9	4

Table 3: Example dependency parse from [Agić et al., 2015] CV set.

More formally we can define a dependency tree as a labelled directed tree $T = (V, A)$ where the vertices $V = \{w_1, w_2..., w_n\}$ are the words in the sentence and the arcs $A \subseteq V \times L \times V$ where L is a set of arch labels. The labels are left out for the unlabelled case $A \subseteq L \times V$ and the dependency tree can be uniquely defined by a set of directed arch of the form (h, d), h being the head vertex/word and d it's dependant.

[7] Combinatory categorial grammar

[8] A finite-state transducer

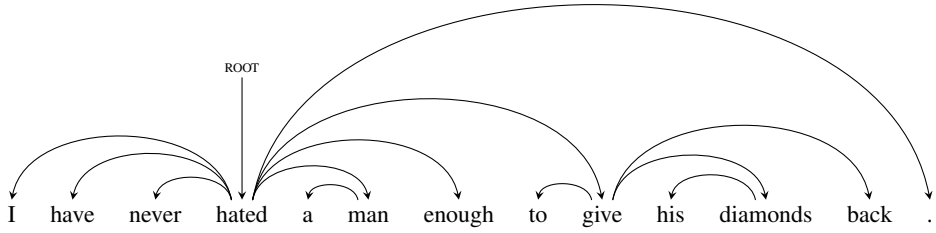

Fig. 1: A projective dependancy tree visualising the example from table 3.

Projectivity is a property of a dependency tree that put simply means that the arcs of the tree do not cross as in Figure1. We assume that all the dependency tree considered have this property and those that did not were filtered out of the training set. A non-projective tree is shown in Figure2. These made up about 4% of the corpus. Non-projectivity is more prevalent in languages like Czech but are not considered a problem in English.

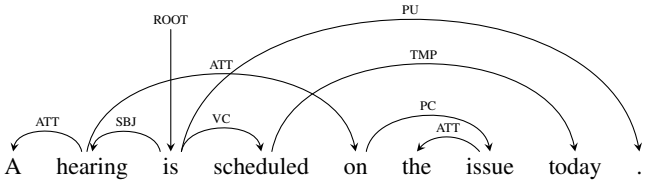

Fig. 2: A non-projective, labeled dependancy tree.

A transition-based dependency parser treats the parsing task as a search for an optimal transition sequence for a given sentence. The transition sequence is broken down into a series of decisions from a small set of options (see figure 3). At each step a classifier is trained using an oracle that, given a gold parse tree, determines optimal transition sequences and thus a correct choice can be know and trained against.

A transition system state is defined as having a configuration $c = (\sigma, \beta, A)$ where σ is a stack, β is a buffer and A is a set of dependency arcs. Initially the system has an empty stack and set of arcs and the buffer is is set to the word in the sentence to be parsed $\beta = V = w_1, w_2, ..., w_n, ROOT$. The final terminal state is a configuration in which the stack is empty and the buffer only contains $ROOT$[9].

An Arc-Eager [Goldberg and Nivre, 2013] transition system has four transitions:

There is a precondition on the $RIGHT$ and $LEFT$ transitions that $b \neq ROOT$ and also the stack σ must not be empty. $LEFT$ is also only legal if s does not have a parent in the existing arcs A and $REDUCE$ must have a parent in in A. This system

[9] and A contains the set of arcs of the parse tree

$$
\begin{aligned}
SHIFT[(\sigma, b|\beta, A)] &= (\sigma|b, \beta, A) \\
RIGHT[(\sigma|s, b|\beta, A)] &= (\sigma|s|b, \beta, A \cup (s, b)) \\
LEFT[(\sigma|s, b|\beta, A)] &= (\sigma, b|\beta, A \cup (b, s)) \\
REDUCE[(\sigma|s, \beta, A)] &= (\sigma, \beta, A)
\end{aligned}
$$

Fig. 3: Transitions in an unlabelled Arc-Eager transition parser model taken from [Goldberg and Nivre, 2013]

collects all of its left dependants first then its right dependants and is "eager" because it adds arcs as early as possible[10].

Dependancy relations can also include labels to indicate the nature of the relation, such as 'subject' or 'object'. The Arc-Eager system defined above can be extended to include the labels which simply requires a new set of $LEFT$ and $RIGHT$ transitions for each label. All the work here can be trivially extended to this case.

The averaged perceptron classifier was introduced to NLP in [Collins, 2002]. Its become the dominant classifier for transition based dependancy parsing because it is a simple design and yet very effective. It is also linear to train and predict with while maintaining an competitive accuracy. It works the same way the classical perceptron algorithm, updating the weights of features on bad predictions: incrementing those that lead to the right prediction and decrementing those that gave the wrong answer. The main difference is at the end of the training cycle the average of each weights history over all the training examples is taken.

A dynamic oracle , introduced by [Goldberg and Nivre, 2012], is used at training time to predict the optimal transition sequence given any previous transition history including those that deviate from the gold tree[11] parse. This is done non-deterministically because a set transitions is returned for a given parse state and gold tree. This allows a greedy parser to learn how to recover from mistakes and reduces the effects of error propagation and results in better parse accuracy.

6 Experiments

The data was split into three sets: a large training set, a cross validation set and a testing set from which the final accuracies are draw.

The tag dictionary, which recorded words and their corresponding tag frequencies was used as a baseline with unknown words using the highest scoring class they fell into. This gave a baseline tagging accuracy of 84%.

[10] unlike the Arc-Standard model which lacks the $REDUCE$ transition and builds its trees bottom up i.e. each word collects its dependence before attaching itself to its head word.

[11] "gold" standard i.e. the best, most reliable data

The MEMM model was trained on the training set and then its regularisation parameter was tuned using the cross validation set. A value of 0.66 as settled on see table 4.

regularisation	Word%	Sent.%
0.1	89.71	43.86
0.2	89.72	43.81
0.33	89.70	43.76
0.5	89.79	43.96
0.58	89.80	44.01
0.66	**89.80**	**44.01**
0.9	89.78	43.71
1.0	89.52	43.41

Table 4: Regularisation parameters tested on the UDP 1.1 cross-validation set.

The training data was multi-tagged using the MEMM model using 10-fold cross validation. The leave-one-out set that got multi-tagged and saved to use in the the parser training. The dynamic oracle of the transition parser requires tagged data that has the mistakes the tagger is likely to make at test time so it can learn to recover from transition histories that incorporate mistakes. This has been shown [Goldberg and Nivre, 2012] to give 1.5 - 3% improvement in accuracy.

Distributed processing was employed as the MEMM tagger took over 26 hours to run on 2000 sentences. Doing this ten times for the 10-fold cross validation was thus not feasible without distributed processing power. Training and tagging of the training set was done one 49 computers each with 4 cores[12]

6.1 Parsing

The final test set was made up of 96.34% projective trees. The parses were using the standard unlabelled attachment score (UAS) which simply counts the number of correct head dependencies returned by the parser. The sentence accuracy was also counted which requires the parser to get the dependencies right for a whole sentence.

This was using a MEMM POS tagger trained on the full training set and which achieved 89.94% word accuracy and 44.63% sentence accuracy on the testing set. The values for the parser accuracy here are well under state of the art and have not been tuned.

[12] This was possible thank to the *dispy* python module. `http://dispy.sourceforge.net/`

Ambiguity		Baseline		Multi tagging		Difference	
γ	tags/word	UAS	UAS Sent.	UAS	UAS Sent.	Word	Sent.
0.0	1.92	78.86	46.41	79.34	47.28	0.48	0.87
0.1	1.40	79.08	46.75	79.38	47.21	0.30	0.46
0.2	1.31	79.09	46.88	79.35	46.97	0.26	0.09
0.3	1.27	79.11	46.92	79.38	46.89	0.27	-0.03
0.4	1.24	79.13	46.94	79.31	46.81	0.18	-0.13
0.5	1.22	79.04	46.82	79.28	46.79	0.24	-0.03
0.6	1.21	79.06	46.81	79.28	46.81	0.22	0.00
0.7	1.19	79.07	46.78	79.26	46.74	0.19	-0.04
0.8	1.18	79.07	46.83	79.24	46.69	0.17	-0.14
0.9	1.17	79.07	46.73	79.21	46.66	0.14	-0.07
1.0	1.00	79.08	46.73	79.21	46.70	0.13	-0.03

Table 5: Unlabelled attachment Score (UAS) for word and sentence for both the baseline without multi-tags and with multi-tagging. The difference column is the difference between the two sets.

7 Conclusion

The results listed in Table 5 indicate that as the number of tags per word increases (i.e. as $gamma$ approaches zero the ambiguity increases) the UAS increases as well.

The MEMM tagger was also bellow the published 96% accuracy. Some of this is the result of the data used here being web data and only about a third the size of the Penn Wall Street Journal treebank. Due to the sparsity of the features in the MEMM model they tent to benefit from more data.

Another problem with the MEMM tagger used here is that its run time is much slower than the parser that it generates it's multiple tags for i.e. polynomial verse linear. Regardless the aim of the experiment was to see if multi tagging features have some benefit to a feature based dependency parser. Other, faster, methods of generating probability distributions could be explored in other work such as the SoftMax layer of [Weiss et al., 2015] for example.

The results were not as significant as those reported by [Curran et al., 2006] on which this work was inspired. There a probabilistic based parser, the CYK algorithm that uses a Probabilistic Context Free Grammar (PCFG), to generate a packed chart of all possible parses over the given tagged sentences[Steedman, 2000]. These are then ranked by a log-linear model similar to a MEMM. Given that the parser was using probabilities in its calculations the effect of injecting real-valued features would have been more prominent.

This small experiment shows that there is potential along this line of enquiry if multi-tagging can be done in a more efficient manor.

Acknowledgements

I would like to thank Alan Blair for supervising this project. Also Matthew Honnibal for the initial idea behind this line of enquiry and he use of this dependency parser code.

References

Agić et al., 2015. Agić, Ž., Aranzabe, M. J., Atutxa, A., Bosco, C., Choi, J., de Marneffe, M.-C., Dozat, T., Farkas, R., Foster, J., Ginter, F., Goenaga, I., Gojenola, K., Goldberg, Y., Hajič, J., Johannsen, A. T., Kanerva, J., Kuokkala, J., Laippala, V., Lenci, A., Lindén, K., Ljubešić, N., Lynn, T., Manning, C., Martínez, H. A., McDonald, R., Missilä, A., Montemagni, S., Nivre, J., Nurmi, H., Osenova, P., Petrov, S., Piitulainen, J., Plank, B., Prokopidis, P., Pyysalo, S., Seeker, W., Seraji, M., Silveira, N., Simi, M., Simov, K., Smith, A., Tsarfaty, R., Vincze, V., and Zeman, D. (2015). Universal dependencies 1.1.

Bikel et al., 1999. Bikel, D., Schwartz, R., and Weischedel, R. (1999). An algorithm that learns what's in a name. *Machine Learning*, 34(1-3):211–231.

Charniak et al., 1996. Charniak, E., Carroll, G., Adcock, J., Cassandra, A., Gotoh, Y., Katz, J., Littman, M., and McCann, J. (1996). Taggers for parsers. *Artificial Intelligence*, 85(1):45–57.

Chomsky, 1957. Chomsky, N. (1957). *Syntactic structures*. Walter de Gruyter.

Collins, 2002. Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics.

Curran et al., 2006. Curran, J. R., Clark, S., and Vadas, D. (2006). Multi-tagging for lexicalized-grammar parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 697–704. Association for Computational Linguistics.

Goldberg and Nivre, 2012. Goldberg, Y. and Nivre, J. (2012). A dynamic oracle for arc-eager dependency parsing. In *COLING*, pages 959–976.

Goldberg and Nivre, 2013. Goldberg, Y. and Nivre, J. (2013). Training deterministic parsers with non-deterministic oracles. *Transactions of the association for Computational Linguistics*, 1:403–414.

Kübler et al., 2009. Kübler, S., McDonald, R., and Nivre, J. (2009). Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127.

Manning, 2011. Manning, C. D. (2011). Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *Computational Linguistics and Intelligent Text Processing*, pages 171–189. Springer.

Rabiner, 1989. Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Ratnaparkhi et al., 1996. Ratnaparkhi, A. et al. (1996). A maximum entropy model for part-of-speech tagging. In *Proceedings of the conference on empirical methods in natural language processing*, volume 1, pages 133–142. Philadelphia, USA.

Silveira et al., 2014. Silveira, N., Dozat, T., de Marneffe, M.-C., Bowman, S., Connor, M., Bauer, J., and Manning, C. D. (2014). A gold standard dependency corpus for English. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*.

Steedman, 2000. Steedman, M. (2000). *The syntactic process*, volume 24. MIT Press.

Tesnière, 1959. Tesnière, L. (1959). *Eléments de syntaxe structurale*. Librairie C. Klincksieck.

Weiss et al., 2015. Weiss, D., Alberti, C., Collins, M., and Petrov, S. (2015). Structured training for neural network transition-based parsing. *arXiv preprint arXiv:1506.06158*.