A Bayesian Model for Generative Transition-based Dependency Parsing

Jan Buys[1] and **Phil Blunsom**[1,2]
[1]Department of Computer Science, University of Oxford [2]Google DeepMind
{jan.buys,phil.blunsom}@cs.ox.ac.uk

Abstract

We propose a simple, scalable, fully generative model for transition-based dependency parsing with high accuracy. The model, parameterized by Hierarchical Pitman-Yor Processes, overcomes the limitations of previous generative models by allowing fast and accurate inference. We propose an efficient decoding algorithm based on particle filtering that can adapt the beam size to the uncertainty in the model while jointly predicting POS tags and parse trees. The UAS of the parser is on par with that of a greedy discriminative baseline. As a language model, it obtains better perplexity than a n-gram model by performing semi-supervised learning over a large unlabelled corpus. We show that the model is able to generate locally and syntactically coherent sentences, opening the door to further applications in language generation.

1 Introduction

Transition-based dependency parsing algorithms that perform greedy local inference have proven to be very successful at fast and accurate discriminative parsing (Nivre, 2008; Zhang and Nivre, 2011; Chen and Manning, 2014). Beam-search decoding further improves performance (Zhang and Clark, 2008; Huang and Sagae, 2010; Choi and McCallum, 2013), but increases decoding time. Graph-based parsers (McDonald et al., 2005; Koo and Collins, 2010; Lei et al., 2014) perform global inference and although they are more accurate in some cases, inference tends to be slower.

In this paper we aim to transfer the advantages of transition-based parsing to generative dependency parsing. While generative models have been used widely and successfully for constituency parsing (Collins, 1997; Petrov et al., 2006), their use in dependency parsing has been limited. Generative models offer a principled approach to semi- and unsupervised learning, and can also be applied to natural language generation tasks.

Dependency grammar induction models (Klein and Manning, 2004; Blunsom and Cohn, 2010) are generative, but not expressive enough for high-accuracy parsing. A previous generative transition-based dependency parser (Titov and Henderson, 2007) obtains competitive accuracies, but training and decoding is computationally very expensive. Syntactic language models have also been shown to improve performance in speech recognition and machine translation (Chelba and Jelinek, 2000; Charniak et al., 2003). However, the main limitation of most existing generative syntactic models is their inefficiency.

We propose a generative model for transition-based parsing (§2). The model, parameterized by Hierarchical Pitman-Yor Processes (HPYPs) (Teh, 2006), learns a distribution over derivations of parser transitions, words and POS tags (§3).

To enable efficient inference, we propose a novel algorithm for linear-time decoding in a generative transition-based parser (§4). The algorithm is based on particle filtering (Doucet et al., 2001), a method for sequential Monte Carlo sampling. This method enables the beam-size during decoding to depend on the uncertainty of the model.

Experimental results (§5) show that the model obtains 88.5% UAS on the standard WSJ parsing task, compared to 88.9% for a greedy discriminative model with similar features. The model can accurately parse up to 200 sentences per second. Although this performance is below state-of-the-art discriminative models, it exceeds existing generative dependency parsing models in either accuracy, speed or both.

As a language model, the transition-based parser offers an inexpensive way to incorporate

Proceedings of the Third International Conference on Dependency Linguistics (Depling 2015), pages 58–67,
Uppsala, Sweden, August 24–26 2015.

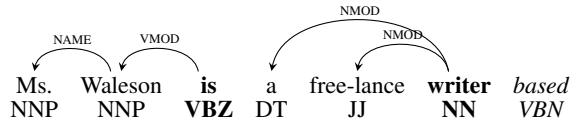

Figure 1: A partially-derived dependency tree for the sentence *Ms. Waleson is a free-lance writer based in New York.* The next word to be predicted by the generative model is *based*. Words in bold are on the stack.

syntactic structure into incremental word prediction. With supervised training the model's perplexity is comparable to that of n-gram models, although generated examples shows greater syntactic coherence. With semi-supervised learning over a large unannotated corpus its perplexity is considerably better than that of a n-gram model.

2 Generative Transition-based Parsing

Our parsing model is based on transition-based projective dependency parsing with the arc-standard parsing strategy (Nivre and Scholz, 2004). Parsing is restricted to (labelled) projective trees. An arc $(i, l, j) \in A$ encodes a dependency between two words, where i is the head node, j the dependent and l is the dependency type of j. In our generative model a word can be represented by its lexical (word) type and/or its POS tag. We add a root node to the beginning of the sentence (although it could also be added at the end of the sentence), such that the head word of the sentence is the dependent of the root node.

A parser configuration (σ, β, A) for sentence $\mathbf{s}$ consists of a stack σ of indices in $\mathbf{s}$, an index β to the next word to be generated, and a set of arcs A. The stack elements are referred to as $\sigma_1, \ldots, \sigma_{|\sigma|}$, where σ_1 is the top element. For any node a, $lc_1(a)$ refers to the leftmost child of a in A, and $rc_1(a)$ to its rightmost child.

The initial configuration is $([], 0, \emptyset)$. A terminal configuration is reached when $\beta > |\mathbf{s}|$, and σ consists only of the root. A sentence is generated left-to-right by performing a sequence of transitions. As a generative model it assigns probabilities to sentences and dependency trees: A word w (including its POS tag) is generated when it is shifted on to the stack, similar to the generative models proposed by Titov and Henderson (2007) and Cohen et al. (2011), and the joint tagging and parsing model of Bohnet and Nivre (2012).

The types of transitions in this model are shift (sh), left-arc (la) and right-arc (ra):

$$\text{sh}_w\colon (\sigma, i, A) \vdash (\sigma|i, i+1, A)$$

$$\text{la}_l\colon (\sigma|i|j, \beta, A) \vdash (\sigma|j, \beta, A \cup \{(j, l, i)\})$$

$$\text{ra}_l\colon (\sigma|i|j, \beta, A) \vdash (\sigma|i, \beta, A \cup \{(i, l, j)\})$$

Left-arc and right-arc (reduce) transitions add an arc between the top two words on the stack, and also generate an arc label l. The parsing strategy adds arcs bottom-up. No arc that would make the root node the dependent of another node may be added. To illustrate the generative process, the configuration of a partially generated dependency tree is given in Figure 1.

In general parses may have multiple derivations. In transition-based parsing it is common to define an oracle $o(c, G)$ that maps the current configuration c and the gold parse G to the next transition that should be performed. In our probabilistic model we are interested in performing inference over all latent structure, including spurious derivations. Therefore we propose a non-deterministic oracle which allows us to find all derivations of G. In contrast to dynamic oracles (Goldberg and Nivre, 2013), we are only interested in derivations of the correct parse tree, so the oracle can assume that given c there exists a derivation for G.

First, to enforce the bottom-up property our oracle has to ensure that an arc (i, j) in G may only be added once j has been attached to all its children – we refer to these arcs as *valid*. Most deterministic oracles add valid arcs greedily. Second, we note that if there exists a valid arc between σ_2 and σ_1 and the oracle decides to shift, the same pair will only occur on the top of the stack again after a right dependent has been attached to σ_1. Therefore right arcs have to be added greedily if they are valid, while adding a valid left arc may be delayed if σ_1 has unattached right dependents in G.

3 Probabilistic Generative Model

Our model defines a joint probability distribution over a parsed sentence with POS tags $\mathbf{t}_{1:n}$, words $\mathbf{w}_{1:n}$ and a transition sequence $\mathbf{a}_{1:2n}$ as

$$p(\mathbf{t}_{1:n}, \mathbf{w}_{1:n}, \mathbf{a}_{1:2n}) = \prod_{i=1}^{n} \Big(p(t_i|\mathbf{h}^t_{m_i}) p(w_i|t_i, \mathbf{h}^w_{m_i}) \prod_{j=m_i+1}^{m_{i+1}} p(a_j|\mathbf{h}^a_j) \Big),$$

where m_i is the number of transitions that have been performed when (t_i, w_i) is generated and $\mathbf{h}^t$, $\mathbf{h}^w$ and $\mathbf{h}^a$ are sequences representing the conditioning contexts for the tag, word and transition distributions, respectively.

In the generative process a shift transition is followed by a sequence of 0 or more reduce transitions. This is repeated until all the words have been generated and a terminal configuration of the parser has been reached. We shall also consider unlexicalised models, based only on POS tags.

3.1 Hierarchical Pitman-Yor processes

The probability distributions for predicting words, tags and transitions are drawn from hierarchical Pitmar-Yor Process (HPYP) priors. HPYP models were originally proposed for n-gram language modelling (Teh, 2006), and have been applied to various NLP tasks. A version of approximate inference in the HPYP model recovers interpolated Kneser-Ney smoothing (Kneser and Ney, 1995), one of the best preforming n-gram language models. The Pitman-Yor Process (PYP) is a generalization of the Dirichlet process which defines a distribution over distributions over a probability space X, with discount parameter $0 \leq d < 1$, strength parameter $\theta > -d$ and base distribution B. PYP priors encode the power-law distribution found in the distribution of words.

Sampling from the posterior is characterized by the Chinese Restaurant Process analogy, where each variable in a sequence is represented by a customer entering a restaurant and sitting at one of an infinite number of tables. Let c_k be the number of customers sitting at table k and K the number of occupied tables. The customer chooses to sit at a table according to the probability

$$P(z_i = k|\mathbf{z}_{1:i-1}) = \begin{cases} \frac{c_k - d}{i-1+\theta} & 1 \leq k \leq K \\ \frac{Kd+\theta}{i-1+\theta} & k = K+1, \end{cases}$$

where z_i is the index of the table chosen by the ith customer and $\mathbf{z}_{1:i-1}$ is the seating arrangement of the previous $i-1$ customers.

All customers at a table share the same dish, corresponding to the value assigned to the variables they represent. When a customer sits at an empty table, a dish is assigned to the table by drawing from the base distribution of the PYP.

For HPYPs, the PYP base distribution can itself be drawn from a PYP. The restaurant analogy is extended to the Chinese Restaurant Franchise, where the base distribution of a PYP corresponds to another restaurant. So when a customer sits at a new table, the dish is chosen by letting a new customer enter the base distribution restaurant. All dishes can be traced back to a uniform base distribution at the top of the hierarchy.

Inference over seating arrangements in the model is performed with Gibbs sampling, based on routines to add or remove a customer from a restaurant. In our implementation we use the efficient data structures proposed by Blunsom et al. (2009). In addition to sampling the seating arrangement, the discount and strength parameters are also sampled, using slice sampling.

In our model $T_{\mathbf{h}^t}, W_{\mathbf{h}^w}$ and $A_{\mathbf{h}^a}$ are HPYPs for the tag, word and transition distributions, respectively. The PYPs for the transition prediction distribution, with conditioning context sequence $\mathbf{h}^a_{1:L}$, are defined hierarchically as

$$\begin{aligned} A_{\mathbf{h}^a_{1:L}} &\sim \text{PYP}(d^A_L, \theta^A_L, A_{\mathbf{h}^a_{1:L-1}}) \\ A_{\mathbf{h}^a_{1:L-1}} &\sim \text{PYP}(d^A_{L-1}, \theta^A_{L-1}, A_{\mathbf{h}^a_{1:L-2}}) \\ \cdots &\quad \cdots \\ A_{\emptyset} &\sim \text{PYP}(d^A_0, \theta^A_0, \text{Uniform}), \end{aligned}$$

where d^A_k and θ^A_k are the discount and strength discount parameters for PYPs with conditioning context length k. Each back-off level drops one context element. The distribution given the empty context backs off to the uniform distribution over all predictions. The word and tag distributions are defined by similarly-structured HPYPs.

The prior specifies an ordering of the symbols in the context from most informative to least informative to the distributions being estimated. The choice and ordering of this context is crucial in the formulation of our model. The contexts that we use are given in Table 1.

4 Decoding

In the standard approach to beam search for transition-based parsing (Zhang and Clark, 2008), the beam stores partial derivations with the same number of transitions performed, and the lowest-scoring ones are removed when the size of the beam exceeds a set threshold. However, in our model we cannot compare derivations with the same number of transitions but which differ in the number of words shifted. One solution is to keep n separate beams, each containing only derivations with i words shifted, but this approach leads to

	Context elements
a_i	$\sigma_1.t, \sigma_2.t, rc_1(\sigma_1).t, lc_1(\sigma_1).t, \sigma_3.t,$ $rc_1(\sigma_2).t, \sigma_1.w, \sigma_2.w$
t_j	$\sigma_1.t, \sigma_2.t, rc_1(\sigma_1).t, lc_1(\sigma_1).t, \sigma_3.t,$ $rc_1(\sigma_2).t, \sigma_1.w, \sigma_2.w$
w_j	$\beta.t, \sigma_1.t, rc_1(\sigma_1).t, lc_1(\sigma_1).t, \sigma_1.w, \sigma_2.w$

Table 1: HPYP prediction contexts for the transition, tag and word distributions. The context elements are ordered from most important to least important; the last elements in the lists are dropped first in the back-off structure. The POS tag of node s is referred to as $s.t$ and the word type as $s.w$.

$O(n^2)$ decoding complexity. Another option is to prune the beam every time after the next word is shifted in all derivations – however the number of reduce transitions that can be performed between shifts is bounded by the stack size, so decoding complexity remains quadratic.

We propose a novel linear-time decoding algorithm inspired by particle filtering (see Algorithm 1). Instead of specifying a fixed limit on the size of the beam, the beam size is controlled by setting the number of particles K. Every partial derivation d_j in the beam is associated with k_j particles, such that $\sum_j k_j = K$. Each pass through the beam advances each d_j until the next word is shifted.

At each step, to predict the next transition for d_j, k_j is divided proportionally between taking a shift or reduce transition, according to $p(a|d_j.\mathbf{h}^a)$. If a non-zero number of particles are assigned to reduce, the highest scoring left-arc and right-arc transitions are chosen deterministically, and derivations that execute them are added to the beam. In practice we found that adding only the highest scoring reduce transition (left-arc or right-arc) gives very similar performance. The shift transition is performed on the current derivation, and the derivation weight is also updated with the word generation probability.

A POS tag is also generated along with a shift transition. Up to three candidate tags are assigned (more do not improve performance) and corresponding derivations are added to the beam, with particles distributed relative to the tag probability (in Algorithm 1 only one tag is predicted).

A pass is complete once the derivations in the beam, including those added by reduce transitions during the pass, have been iterated through. Then a selection step is performed to determine which

Input: Sentence $\mathbf{w}_{1:n}$, K particles.
Output: Parse tree of $\arg\max_{d \text{ in beam}} d.\theta$.
Initialize the beam with parser configuration d with weight $d.\theta = 1$ and $d.k = K$ particles;
for $i \leftarrow 1$ **to** N **do**
  Search step;
  foreach *derivation* d *in beam* **do**
    nShift = round$(d.k \cdot p(\text{sh}|d.\mathbf{h}^a))$;
    nReduce = $d.k$ − nShift;
    if *nReduce* > 0 **then**
      $a = \arg\max_{a \neq \text{sh}} p(a|d.\mathbf{h}^a)$;
      beam.append($dd \leftarrow d$);
      $dd.k \leftarrow$ nReduce;
      $dd.\theta \leftarrow dd.\theta \cdot p(a|d.\mathbf{h}^a)$;
      dd.execute(a);
    end
    $d.k \leftarrow$ nShift;
    if *nShift* > 0 **then**
      $d.\theta \leftarrow d.\theta \cdot p(\text{sh}|d.\mathbf{h}^a) \cdot \max_{t_i} p(t_i|d.\mathbf{h}^t) p(w_i|d.\mathbf{h}^w)$;
      d.execute(sh);
    end
  end
  Selection step;
  foreach *derivation* d *in beam* **do**
    $d.\theta' \leftarrow \frac{d.k \cdot d.\theta}{\sum_{d'} d'.k \cdot d'.\theta}$;
  end
  foreach *derivation* d *in beam* **do**
    $d.k = \lfloor d.\theta' \cdot K \rfloor$;
    if $d.k = 0$ **then**
      beam.remove(d);
    end
  end
end

Algorithm 1: Beam search decoder for arc-standard generative dependency parsing.

derivations are kept. The number of particles for each derivation are reallocated based on the normalised weights of the derivations, each weighted by its current number of particles. Derivations to which zero particles are assigned are eliminated. The selection step allows the size of the beam to depend on the uncertainty of the model during decoding. The selectional branching method proposed by Choi and McCallum (2013) for discriminative beam-search parsing has a similar goal.

After the last word in the sentence has been shifted, reduce transitions are performed on each derivation until it reaches a terminal configuration. The parse tree corresponding to the highest scoring final derivation is returned.

The main differences between our algorithm and particle filtering are that we divide particles proportionally instead of sampling with replacement, and in the selection step we base the redistribution on the derivation weight instead of the importance weight (the word generation probability). Our method can be interpreted as maximizing

by sampling from a peaked version of the distribution over derivations.

5 Experiments

5.1 Parsing Setup

We evaluate our model as a parser on the standard English Penn Treebank (Marcus et al., 1993) setup, training on WSJ sections 02-21, developing on section 22, and testing on section 23. We use the head-finding rules of Yamada and Matsumoto (2003) (YM)[1] for constituency-to-dependency conversion, to enable comparison with previous results. We also evaluate on the Stanford dependency representation (De Marneffe and Manning, 2008) (SD)[2].

Words that occur only once in the training data are treated as unknown words. We classify unknown words according to capitalization, numbers, punctuation and common suffixes into classes similar to those used in the implementation of generative constituency parsers such as the Stanford parser (Klein and Manning, 2003).

As a discriminative baseline we use MaltParser (Nivre et al., 2006), a discriminative, greedy transition-based parser, performing arc-standard parsing with LibLinear as classifier. Although the accuracy of this model is not state-of-the-art, it does enable us to compare our model against an optimised discriminative model with a feature-set based on the same elements as we include in our conditioning contexts.

Our HPYP dependency parser (HPYP-DP) is trained with 20 iterations of Gibbs sampling, resampling the hyper-parameters after every iteration, except when performing inference over latent structure, in which case they are only resampled every 5 iterations. Training with a deterministic oracle takes 28 seconds per iteration (excluding resampling hyper-parameters), while a non-deterministic oracle (sampling with 100 particles) takes 458 seconds.

5.2 Modelling Choices

We consider several modelling choices in the construction of our generative dependency parsing model. Development set parsing results are given in Table 2. We report unlabelled attachment score (UAS) and labelled attachment score (LAS), excluding punctuation.

Model	UAS	LAS
MaltParser Unlex	85.23	82.80
MaltParser Lex	89.17	87.81
Unlexicalised	85.64	82.93
Lexicalised, unlex context	87.95	85.04
Lexicalised, tagger POS	87.84	85.54
Lexicalised, predict POS	**89.09**	**86.78**
Lexicalised, gold POS	89.30	87.28

Table 2: HPYP parsing accuracies on the YM development set, for various lexicalised and unlexicalised setups.

Context elements	UAS	LAS
$\sigma_1.t, \sigma_2.t$	73.25	70.14
$+rc_1(\sigma_1).t$	80.21	76.64
$+lc_1(\sigma_1).t$	85.18	82.03
$+\sigma_3.t$	87.23	84.26
$+rc_1(\sigma_2).t$	87.95	85.04
$+\sigma_1.w$	88.53	86.11
$+\sigma_2.w$	88.93	86.57

Table 3: Effect of including elements in the model conditioning contexts. Results are given on the YM development set.

HPYP priors

The first modelling choice is the selection and ordering of elements in the conditioning contexts of the HPYP priors. Table 3 shows how the development set accuracy increases as more elements are added to the conditioning context. The first two words on the stack are the most important, but insufficient – second-order dependencies and further elements on the stack should also be included in the contexts. The challenge is that the back-off structure of each HPYP specifies an ordering of the elements based on their importance in the prediction. We are therefore much more restricted than classifiers with large, sparse feature-sets which are commonly used in transition-based parsers. Due to sparsity, the word types are the first elements to be dropped in the back-off structure, and elements such as third-order dependencies, which have been shown to improve parsing performance, cannot be included successfully in our model.

Sampling over parsing derivations during training further improves performance by 0.16% to

[1] http://stp.lingfil.uu.se/ nivre/research/Penn2Malt.html

[2] Converted with version 3.4.1 of the Stanford parser, available at http:/nlp.stanford.edu/software/lex-parser.shtml.

89.09 UAS. Adding the root symbol at the end of the sentence rather than at the front gives very similar parsing performance. When unknown words are not clustered according to surface features, performance drops to 88.60 UAS.

POS tags and lexicalisation

It is standard practice in transition-based parsing to obtain POS tags with a stand-alone tagger before parsing. However, as we have a generative model, we can use the model to assign POS tags in decoding, while predicting the transition sequence. We compare predicting tags against using gold standard POS tags and tags obtain using the Stanford POS tagger[3] (Toutanova et al., 2003). Even though the predicted tags are slightly less accurate than the Stanford tags on the development set (95.6%), jointly predicting tags and decoding increases the UAS by 1.1%. The jointly predicted tags are a better fit to the generative model, which can be seen by an improvement in the likelihood of the test data. Bohnet and Nivre (2012) found that joint prediction increases both POS and parsing accuracy. However, their model rescored a k-best list of tags obtained with an preprocessing tagger, while our model does not use the external tagger at all during joint prediction.

We train lexicalised and unlexicalised versions of our model. Unlexicalised parsing gives us a strong baseline (85.6 UAS) over which to consider our model's ability to predict and condition on words. Unlexicalised parsing is also considered to be robust for applications such as cross-lingual parsing (McDonald et al., 2011). Additionally, we consider a version of the model that don't include lexical elements in the conditioning context. This model performs only 1% UAS lower than the best lexicalised model, although it makes much stronger independence assumptions. The main benefit of lexicalised conditioning contexts are to make incremental decoding easier.

Speed vs accuracy trade-offs

We consider a number of trade-offs between speed and accuracy in the model. We compare using different numbers of particles during decoding, as well as jointly predicting POS tags against using pre-obtained tags (Table 4).

[3]We use the efficient "left 3 words" model, trained on the same data as the parsing model, excluding distributional features. Tagging accuracy is 95.9% on the development set and 96.5% on the test set.

Particles	Sent/sec	UAS
5000	18	89.04
1000	27	88.93
100	54	87.99
10	104	85.27
1000	108	87.59
100	198	87.46
10	333	85.86

Table 4: Speed and accuracy for different configurations of the decoding algorithm. Above the line, POS tags are predicted by the model, below pre-tagged POS are used.

Model	UAS	LAS
Eisner (1996)	80.7	-
Wallach et al. (2008)	85.7	-
Titov and Henderson (2007)	89.36	87.65
HPYP-DP	**88.47**	**86.13**
MaltParser	88.88	87.41
Zhang and Nivre (2011)	92.9	91.8
Choi and McCallum (2013)	92.96	91.93

Table 5: Parsing accuracies on the YM test set. compared against previous published results. Titov and Henderson (2007) was retrained to enable direct comparison.

The optimal number of particles is found to be 1000 - more particles only increase accuracy by about 0.1 UAS. Although jointly predicting tags is more accurate, using pre-obtained tags provides a better trade-off between speed and accuracy – 87.59 against 85.27 UAS at around 100 sentences per second. In comparison, the MaltParser parses around 500 sentences per second.

We also compare our particle filter-based algorithm against a more standard beam-search algorithm that prunes the beam to a fixed size after each word is shifted. This algorithm is much slower than the particle-based algorithm – to get similar accuracy it parses only 3 sentences per second (against 27) when predicting tags jointly, and 29 (against 108) when using pre-obtained tags.

5.3 Parsing Results

Test set results comparing our model against existing discriminative and generative dependency parsers are given in Table 5. Our HPYP model performs much better than Eisner's generative model as well as the Bayesian version of that model proposed by Wallach et al. (2008) (the result for Eis-

ner's model is given as reported by Wallach et al. (2008) on the WSJ). The accuracy of our model is only 0.8 UAS below the generative model of Titov and Henderson (2007), despite that model being much more powerful. The Titov and Henderson model takes 3 days to train, and its decoding speed is around 1 sentence per second.

The UAS of our model is very close to that of the MaltParser. However, we do note that our model's performance is relatively worse on LAS than on UAS. An explanation for this is that as we do not include labels in the conditioning contexts, the predicted labels are independent of words that have not yet been generated.

We also test the model on the Stanford dependencies, which have a larger label set. Our model obtains 87.9/83.2 against the MaltParser's 88.9/86.2 UAS/LAS.

Despite these promising results, our model's performance still lags behind recent discriminative parsers (Zhang and Nivre, 2011; Choi and McCallum, 2013) with beam-search and richer feature sets than can be incorporated in our model. In terms of speed, Zhang and Nivre (2011) parse 29 sentences per second, against the 110 sentences per second of Choi and McCallum (2013). Recently proposed neural networks for dependency parsers have further improved performance (Dyer et al., 2015; Weiss et al., 2015), reaching up to 94.0% UAS with Stanford dependencies.

We argue that the main weakness of the HPYP parser is sparsity in the large conditioning contexts composed of tags and words. The POS tags in the parser configuration context already give a very strong signal for predicting the next transition. As a result it is challenging to construct PYP reduction lists that also include word types without making the back-off contexts too sparse.

The other limitation is that our decoding algorithm, although efficient, still prunes the search space aggressively, while not being able to take advantage of look-ahead features as discriminative models can. Interestingly, we note that a discriminative parser cannot reach high performance without look-ahead features.

5.4 Language Modelling

Next we evaluate our model as a language model. First we use the standard WSJ language modelling setup, training on sections $00 - 20$, developing on $21 - 22$ and testing on $23 - 24$. Punctuation is removed, numbers and symbols are mapped to a single symbol and the vocabulary is limited to $10,000$ words. Second we consider a semi-supervised setup where we train the model, in addition to the WSJ, on a subset of 1 million sentences (24.1 million words) from the WMT English monolingual training data[4]. This model is evaluated on `newstest2012`.

When training our models for language modelling, we first perform standard supervised training, as for parsing (although we don't predict labels). This is followed by a second training stage, where we train the model only on words, regarding the tags and parse trees as latent structure. In this unsupervised stage we train the model with particle Gibbs sampling (Andrieu et al., 2010), using a particle filter to sample parse trees. When only training on the WSJ, we perform this step on the same data, now allowing the model to learn parses that are not necessarily consistent with the annotated parse trees.

For semi-supervised training, unsupervised learning is performed on the large unannotated corpus. However, here we find the highest scoring parse trees, rather than sampling. Only the word prediction distribution is updated, not the tag and transition distributions.

Language modelling perplexity results are given in Table 6. We note that the perplexities reported are upper bounds on the true perplexity of the model, as it is intractable to sum over all possible parses of a sentence to compute the marginal probability of the words. As an approximation we sum over the final beam after decoding.

The results show that on the WSJ the model performs slightly better than a HPYP n-gram model. One disadvantage of evaluating on this dataset is that due to removing punctuation and restricting the vocabulary, the model parsing accuracy drops to 84.6 UAS. Also note that in contrast to many other evaluations, we do not interpolate with a n-gram model – this will improve perplexity further.

On the big dataset we see a larger improvement over the n-gram model. This is a promising result, as it shows that our model can successfully generalize to larger vocabularies and unannotated datasets.

[4] Available at http://www.statmt.org/wmt14/translation-task.html.

Model	Perplexity
HPYP 5-gram	147.22
Chelba and Jelinek (2000)	146.1
Emami and Jelinek (2005)	131.3
HPYP-DP	**145.54**
HPYP 5-gram	178.13
HPYP-DP	**163.96**

Table 6: Language modelling test results. Above, training and testing on WSJ. Below, training semi-supervised and testing on WMT.

5.5 Generation

To support our claim that our generative model is a good model for sentences, we generate some examples. The samples given here were obtained by generating 1000 samples, and choosing the 10 highest scoring ones with length greater or equal to 10. The models are trained on the standard WSJ training set (including punctuation).

The examples are given in Table 7. The quality of the sentences generated by the dependency model is superior to that of the n-gram model, despite the models have similar test set perplexities. The sentences generated by the dependency model tend to have more global syntactic structure (for examples having verbs where expected), while retaining the local coherence of n-gram models. The dependency model was also able to generate balanced quotation marks.

6 Related work

One of the earliest graph-based dependency parsing models (Eisner, 1996) is generative, estimating the probability of dependents given their head and previously generated siblings. To counter sparsity in the conditioning context of the distributions, backoff and smoothing are performed. Wallach et al. (2008) proposed a Bayesian HPYP parameterisation of this model.

Other generative models for dependency trees have been proposed mostly in the context of unsupervised parsing. The first successful model was the dependency model with valence (DMV) (Klein and Manning, 2004). Several extensions have been proposed for this model, for example using structural annaeling (Smith and Eisner, 2006), Viterbi EM training (Spitkovsky et al., 2010) or richer contexts (Blunsom and Cohn, 2010). However, these models are not powerful enough for either accurate parsing or language modelling with rich contexts (they are usually restricted to first-order dependencies and valency).

Although any generative parsing model can be applied to language modelling by marginalising out the possible parses of a sentence, in practice the success of such models has been limited. Lexicalised PCFGs applied to language modelling (Roark, 2001; Charniak, 2001) show improvements over n-gram models, but decoding is prohibitively expensive for practical integration in language generation applications.

Chelba and Jelinek (2000) as well as Emami and Jelinek (2005) proposed incremental syntactic language models with some similarities to our model. Those models predict binarized constituency trees with a transition-based model, and are parameterized by deleted interpolation and neural networks, respectively. Rastrow et al. (2012) applies a transition-based dependency language model to speech recognition, using hierarchical interpolation and relative entropy pruning. However, the model perplexity only improves over an n-gram model when interpolated with one.

Titov and Henderson (2007) introduced a generative latent variable model for transition-based parsing. The model is based on an incremental sigmoid belief networks, using the arc-eager parsing strategy. Exact inference is intractable, so neural networks and variational mean field methods are proposed to perform approximate inference. However, this is much slower and therefore less scalable than our model.

A generative transition-based parsing model for non-projective parsing is proposed in (Cohen et al., 2011), along with a dynamic program for inference. The parser is similar to ours, but the dynamic program restricts the conditioning context to the top 2 or 3 words on the stack. No experimental results are included.

Le and Zuidema (2014) proposed a recursive neural network generative model over dependency trees. However, their model can only score trees, not perform parsing, and its perplexity (236.58 on the PTB development set) is worse than model's, despite using neural networks to combat sparsity.

Finally, incremental parsing with particle filtering has been proposed previously (Levy et al., 2009) to model human online sentence processing.

sales rose NUM to NUM million from $ NUM .
estimated volume was about $ NUM a share , .
meanwhile , annual sales rose to NUM % from $ NUM .
mr. bush 's profit climbed NUM % , to $ NUM from $ NUM million million , or NUM cents a share .
treasury securities inc. is a unit of great issues .
" he is looking out their shareholders , " says .
while he has done well , she was out .
that 's increased in the second quarter 's new conventional wisdom .
mci communications said net dropped NUM % for an investor .
association motorola inc. , offering of $ NUM and NUM cents a share .
otherwise , actual profit is compared with the 300-day estimate .
the companies are followed by at least three analysts , and had a minimum five-cent change in actual earnings per share .
bonds : shearson lehman hutton treasury index NUM , up
posted yields on NUM year mortgage commitments for delivery within NUM days .
in composite trading on the new york mercantile exchange .
the company , which has NUM million shares outstanding .
the NUM results included a one-time gain of $ NUM million .
however , operating profit fell NUM % to $ NUM billion from $ NUM billion .
merrill lynch ready assets trust : NUM % NUM days ; NUM % NUM to NUM days ; NUM % NUM to NUM days .
in new york stock exchange composite trading , one trader .

Table 7: Sentences generated, above by the generative dependency model, below by a n-gram model. In both cases, 1000 samples were generated, and the most likely sentences of length 10 or more are given.

7 Conclusion

We presented a generative dependency parsing model that, unlike previous models, retains most of the speed and accuracy of discriminative parsers. Our models can accurately estimate probabilities conditioned on long context sequences. The model is scalable to large training and test sets, and even though it defines a full probability distribution over sentences and parses, decoding speed is efficient. Additionally, the generative model gives strong performance as a language model. For future work we believe that this model can be applied successfully to natural language generation tasks such as machine translation.

References

Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. 2010. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342.

Phil Blunsom and Trevor Cohn. 2010. Unsupervised induction of tree substitution grammars for dependency parsing. In *EMNLP*, pages 1204–1213.

Phil Blunsom, Trevor Cohn, Sharon Goldwater, and Mark Johnson. 2009. A note on the implementation of hierarchical Dirichlet processes. In *ACL/IJCNLP (Short Papers)*, pages 337–340.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *EMNLP-CoNLL*, pages 1455–1465.

Eugene Charniak, Kevin Knight, and Kenji Yamada. 2003. Syntax-based language models for statistical machine translation. In *Proceedings of MT Summit IX*, pages 40–46.

Eugene Charniak. 2001. Immediate-head parsing for language models. In *Proceedings of ACL*, pages 124–131.

Ciprian Chelba and Frederick Jelinek. 2000. Structured language modeling. *Computer Speech & Language*, 14(4):283–332.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*.

Jinho D. Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *ACL*.

Shay B. Cohen, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Exact inference for generative probabilistic non-projective dependency parsing. In *EMNLP*, pages 1234–1245.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *ACL*, pages 16–23.

Marie-Catherine De Marneffe and Christopher D Manning. 2008. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8.

Arnaud Doucet, Nando De Freitas, and Neil Gordon. 2001. *Sequential Monte Carlo methods in practice*. Springer.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of ACL 2015*.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *COLING*, pages 340–345.

Ahmad Emami and Frederick Jelinek. 2005. A neural syntactic language model. *Machine Learning*, 60(1-3):195–227.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *TACL*, 1:403–414.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *ACL*, pages 1077–1086.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *ACL*, pages 423–430.

Dan Klein and Christopher D Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *ACL*, pages 478–586.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *ICASSP*, volume 1, pages 181–184. IEEE.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *ACL*, pages 1–11.

Phong Le and Willem Zuidema. 2014. The inside-outside recursive neural network model for dependency parsing. In *EMNLP*, pages 729–739.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of ACL (Volume 1: Long Papers)*, pages 1381–1391.

Roger P Levy, Florencia Reali, and Thomas L Griffiths. 2009. Modeling the effects of memory on human online sentence processing with particle filters. In *Advances in neural information processing systems*, pages 937–944.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Ryan T. McDonald, Koby Crammer, and Fernando C. N. Pereira. 2005. Online large-margin training of dependency parsers. In *ACL*.

Ryan McDonald, Slav Petrov, and Keith Hall. 2011. Multi-source transfer of delexicalized dependency parsers. In *EMNLP*, pages 62–72. Association for Computational Linguistics.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *COLING*.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, volume 6, pages 2216–2219.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *COLING-ACL*, pages 433–440.

Ariya Rastrow, Mark Dredze, and Sanjeev Khudanpur. 2012. Efficient structured language modeling for speech recognition. In *INTERSPEECH*.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational linguistics*, 27(2):249–276.

Noah A. Smith and Jason Eisner. 2006. Annealing structural bias in multilingual weighted grammar induction. In *Proceedings of COLING-ACL*, pages 569–576.

Valentin I. Spitkovsky, Hiyan Alshawi, Daniel Jurafsky, and Christopher D. Manning. 2010. Viterbi training improves unsupervised dependency parsing. In *CoNLL*, pages 9–17.

Yee Whye Teh. 2006. A hierarchical Bayesian language model based on Pitman-Yor processes. In *ACL*.

Ivan Titov and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 144–155.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL*, pages 173–180.

Hanna M Wallach, Charles Sutton, and Andrew McCallum. 2008. Bayesian modeling of dependency trees using hierarchical Pitman-Yor priors. In *ICML Workshop on Prior Knowledge for Text and Language Processing*.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of ACL 2015*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *EMNLP*, pages 562–571.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *ACL-HLT short papers-Volume 2*, pages 188–193.