Neural Syntactic Generative Models with Exact Marginalization

Jan Buys[1,2] **and Phil Blunsom**[1,3]
[1]Department of Computer Science, University of Oxford
[2]Paul G. Allen School of Computer Science & Engineering, University of Washington
[3]DeepMind
jbuys@cs.washington.edu, phil.blunsom@cs.ox.ac.uk

Abstract

We present neural syntactic generative models with exact marginalization that support both dependency parsing and language modeling. Exact marginalization is made tractable through dynamic programming over shift-reduce parsing and minimal RNN-based feature sets. Our algorithms complement previous approaches by supporting batched training and enabling online computation of next word probabilities. For supervised dependency parsing, our model achieves a state-of-the-art result among generative approaches. We also report empirical results on unsupervised syntactic models and their role in language modeling. We find that our model formulation of latent dependencies with exact marginalization do not lead to better intrinsic language modeling performance than vanilla RNNs, and that parsing accuracy is not correlated with language modeling perplexity in stack-based models.

1 Introduction

We investigate the feasibility of neural syntactic generative models with structured latent variables in which exact inference is tractable. Recent models have added structure to recurrent neural networks at the cost of giving up exact inference, or through using soft structure instead of latent variables (Dyer et al., 2016; Yogatama et al., 2016; Grefenstette et al., 2015). We propose generative models in which syntactic structure is modelled with a discrete stack which can be marginalized as a latent variable through dynamic programming. This enables us to investigate the trade-off between model expressivity and exact marginalization in probabilistic models based on recurrent neural networks (RNNs).

While Long Short-term Memory (Hochreiter and Schmidhuber, 1997) (LSTM) RNNs have driven strong improvements in intrinsic language modelling performance, they fail at capturing certain long-distance dependencies, such as those required for modelling subject-verb agreement (Linzen et al., 2016) or performing synthetic transduction tasks based on context-free grammars (Grefenstette et al., 2015). We propose generative models, based on transition-based dependency parsing (Nivre, 2008), a widely used framework for incremental syntactic parsing, that are able to capture desirable dependencies.

Our generative approach to dependency parsing encodes sentences with an RNN and estimate transition and next word probability distributions by conditioning on a small number of features represented by RNN encoder vectors. In contrast to previous syntactic language models such as RNNG (Dyer et al., 2016), marginal word probabilities can be computed both online and exactly. A GPU implementation which exploits parallelization enables unsupervised learning and fast training and decoding. The price of exact inference is that our models are less expressive than RNNG, as the recurrence is not syntax-dependent.

Our generative models are based on the arc-eager and arc-hybrid transition systems, with $O(n^3)$ dynamic programs based on Kuhlmann et al. (2011). Previous work on dynamic programming for transition-based parsing either required approximate inference due to a too high polynomial order run-time complexity (Huang and Sagae, 2010), or had too restrictive feature spaces to be used as accurate models (Kuhlmann et al., 2011; Cohen et al., 2011). Recent work showed that bidirectional RNNs enable accurate graph-based and transition-based dependency parsing using minimal feature spaces (Kiperwasser and Goldberg, 2016; Cross and Huang, 2016; Dozat and Manning, 2017). Shi et al. (2017) further showed that under this approach exact decoding

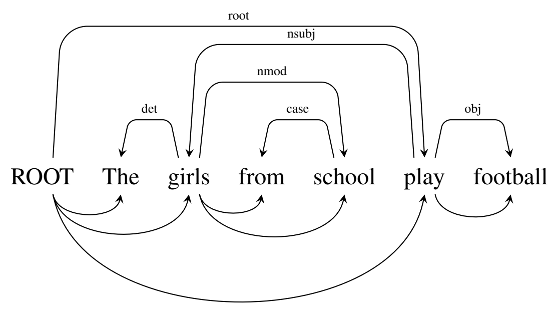

Figure 1: A dependency tree (arcs above words) together with dependencies captured by the generative model for word prediction (arcs below words).

Stack σ	Index $\beta - 1$	Prediction
ROOT	ROOT	sh(The)
ROOT, The	The	la(det)
ROOT	The	sh(girls)
ROOT, girls	girls	sh(from)
ROOT, girls, from	from	la(case)
ROOT, girls	from	sh(school)
ROOT, girls, school	school	ra(nmod)
ROOT, girls	school	la(nsubj)
ROOT	school	sh(play)
ROOT, play	play	sh(football)
ROOT, play, football	football	ra(obj)
ROOT, play	football	ra(root)
ROOT	football	re

Table 1: Arc-hybrid transition system derivation for the sentence "The girls from school play football." The transitions are shift (sh), reduce (re), left-arc (la) and right-arc (ra).

and globally-normalized discriminative training is tractable with dynamic programming.

While discriminative neural network-based models obtain state-of-the-art parsing accuracies (Dozat and Manning, 2017), generative models for structured prediction have a number of advantages: They do not suffer from label bias or explaining away effects (Yu et al., 2017), have lower sample complexity (Yogatama et al., 2017), are amenable to unsupervised learning and can model uncertainty and incorporate prior knowledge through latent variables.

As a supervised parser our model obtains state of the art performance in transition-based generative dependency parsing. While its intrinsic language modelling performance is worse than that of a well-tuned vanilla RNN, we see that the formulation of the generative model has a large impact on both the informedness of the syntactic structure and the parsing accuracy of the model. Furthermore, there is a discrepancy between the model structure most suitable for parsing and for language modeling. Our analysis shows that there exist informative syntactically-motivated dependencies which LSTMs are not capturing, even though our syntactic models are not able to predict them accurately enough during online processing to improve language modelling performance. Our implementation is available at `https://github.com/janmbuys/ndp-parser`.

2 Generative shift-reduce parsing

We start by defining a shift-reduce transition system which does not predict dependency arcs, but simply processes the words in a sentence left to right through shifting words onto a stack and reducing (popping) them from the stack. We define a generative model for this transition system and a dynamic program to perform inference over all possible shift-reduce transitions to process a given sentence. An example dependency tree is given in Figure 1, along with the dependencies our generative model is capturing when making word predictions. The arc-hybrid transition sequence for the example is given in Table 1.

Let sentence $\mathbf{w}_{0:n}$ be a sequence of words, where w_0 is always the designated root symbol `ROOT` and w_n the end-of-sentence symbol `EOS`. The state variables of the transition system are the stack σ, consisting of word indexes, and a current word index β, also referred to as the buffer. The first and second elements on the stack are referred to as σ_0 and σ_1, respectively. We use the notation $\sigma|j$ to indicate that j is on top of the stack. The initial state (σ, β) is $([0], 1)$ and the final state is $([], n)$. There are two transition actions, *shift* and *reduce*. Shift updates the transition state from (σ, j) to $(\sigma|j, j+1)$. Reduce changes the state from $(\sigma|i, j)$ to (σ, j).

2.1 Generative model

The generative model for this transition system is defined by a probability distribution over $\mathbf{w}$,

$$p(\mathbf{w}) = \sum_{\mathbf{t}} p(\mathbf{w}_{0:n}, \mathbf{t}_{0:2n}), \qquad (1)$$

where $\mathbf{t}_{0:2n}$ is a transition sequence that processes the sentence. Shift actions predict (assign probability to) the next word in the sentence. The end-of-sentence symbol is generated implicitly when ROOT is reduced from the stack.

The sentence is encoded left-to-right by an LSTM RNN taking the word embedding of the last predicted word as input at each time step, independent of $\mathbf{t}$. The RNN hidden states $\boldsymbol{h}_{0:n}$ represent each sentence position in its linear context. The probability of a shift action and the word that it predicts is

$$p_{\text{tr}}(\text{sh}|h_{\sigma_0}, h_{\beta-1})p_{\text{gen}}(w|h_{\sigma_0}, h_{\beta-1}).$$

Reduce is predicted with probability $p_{\text{tr}}(\text{re}|h_{\sigma_0}, h_{\beta-1}) = 1 - p_{\text{tr}}(\text{sh}|h_{\sigma_0}, h_{\beta-1})$.

The transition and word probability distributions are estimated by non-linear output layers that take the context-depending RNN representations of positions in the transition system as input,

$$p_{\text{tr}} = \text{sigmoid}(r^T \text{relu}(W_{ts}h_{\sigma_0} + W_{tb}h_{\beta-1})) \tag{2}$$

$$p_{\text{gen}} = \text{softmax}(R^T \tanh(W_{gs}h_{\sigma_0} + W_{tb}h_{\beta-1})), \tag{3}$$

where R and the W's are neural network parameter matrices and r is a parameter vector.

The model has two ways of representing context: The RNN encoding, which has a recency bias, and the stack, which can represent long range dependencies and has a syntactic distance bias. The choice of RNN states (corresponding to stack elements) to condition on is restricted by our goal of making the dynamic programming tractable.

We propose two formulations of the generative model: In the first, referred to as *stack-next*, shift generates the word pushed on the stack, which is currently at position β, as in the equations above. In the second formulation, referred to as *buffer-next*, shift generates the word at position $\beta+1$, i.e., the next word on the buffer. The first formulation has a more intuitive generative story as the generation of a word is conditioned on the top of the stack when it is generated (see Table 1), but the second formulation has the advantage that transition predictions are conditioned on the current word at position β, which is more informative for parsing predictions. Models are defined using *stack-next* unless stated otherwise.

2.2 Dynamic program

We now define a dynamic program for this model, based on the algorithms proposed by Kuhlmann et al. (2011) and their application to generative dependency parsing (Cohen et al., 2011).

The key to the dynamic program is the decomposition of the transition sequence into *push computations*. Each push computation is a sequence of transitions which results in a single node having been pushed to the stack. The simplest push computation is a single shift operation. Push computations can be composed recursively: combining two consecutive push computations followed by a reduce transition yields a new push operation. Therefore the derivation of a sentence under the transition system can be seen as a composition of push computations.

Items in the deduction system (Shieber et al., 1995) of the dynamic program have the form $[i, j]$, which has the interpretation that there exists a push computation between actions a_k and a_l such that $\beta = i$ at time step k and $\sigma_0 = i$ and $\beta = j$ at time step l. In the deduction system $[0, 1]$ is an axiom, $[0, n]$ is the goal and the deduction rules corresponding to the transitions are

$$\begin{aligned} [i, j-1] &\rightarrow [j-1, j] \qquad &\text{(shift)} \\ [i, k][k, j] &\rightarrow [i, j] \qquad &\text{(reduce).} \end{aligned}$$

The marginal probability distribution is computed by defining the inside score $I(i, j) = p(\mathbf{w}_{i:j-1})$ for every deduction system item. Computing the sentence probability corresponds to computing the inside score of the goal, $I(0, n) = p(\mathbf{w}_{0:n-1})$, followed by computing the final reduce probability.

Reduce probabilities are computed conditioned on positions k and j, which are accessible through the dynamic program deduction rule. However the shift probabilities cannot be computed at the *shift* rule for deducing $[j-1, j]$, as it does not have access there to the top of the stack. One solution is to extend the deduction system to a three-tuple that can track the value of an additional position, leading to a $O(n^4)$ dynamic program. Instead Shi et al. (2017) showed that the computation can be performed in the $O(n^3)$ algorithm by computing the shift probability of word k during the *reduce* deduction, as it was generated when i was on top of the stack. The inside algorithm is given in Algorithm 1.

To train the model without supervised transition sequences, we can optimize the negative log likelihood of $p(w_{0:n})$ directly with gradient-based optimization using automatic differentiation, which

Algorithm 1 Inside algorithm for the shift-reduce transition-based generative model.

```
1: $I(0,1) \leftarrow 1$
2: for $j = 2, \ldots, n$ do
3:     $I(j-1, j) \leftarrow 1$
4:     for $i = j-2, \ldots, 0$ do
5:         for $k = i+1, \ldots, j-1$ do
6:             $T(k) \leftarrow p_{\text{tr}}(\text{sh}|h_i, h_{k-1}) p_{\text{gen}}(w_k|h_i, h_{k-1})$
7:         end for
8:         $I(i,j) \leftarrow \sum_{k=i+1}^{j-1} I(i,k)I(k,j)p_{tr}(\text{re}|h_k, h_{j-1})T(k)$
9:     end for
10: end for
11: return $I(0,n) + p_{tr}(\text{re}|h_0, h_{n-1})$
```

is equivalent to computing the gradients with the outside algorithm (Eisner, 2016). For decoding we perform Viterbi search over the dynamic program by maximizing rather than summing over different split positions (values of k when reducing).

The *buffer-next* generative formulation, where shift generates the next word β, can also be computed with the dynamic program. Here w_1 is predicted at the initial state in $I(0,1)$, while the end-of-sentence token is generated explicitly when a shift action results in buffer being set to position n, regardless of the state of the stack.

3 Transition-based dependency parsing

The arc-eager (Nivre, 2008) and arc-hybrid (Kuhlmann et al., 2011) transition systems for projective dependency parsing use the same shift-reduce operations but predict left- and right-arcs at different time steps. We propose generative models for these transition systems based on the dynamic program for shift-reduce parsing proposed above, again following Kuhlmann et al. (2011). For supervised training we optimize the joint probability distribution $p(\mathbf{w}, \mathbf{t})$, where an oracle is used to derive transition sequence $\mathbf{t}$ from the training examples. In cases of spurious ambiguity arcs are added as soon as possible.

3.1 Arc-hybrid parser

The arc-hybrid transition system has three actions: Shift, left-arc and right-arc (see Table 2 for definitions). *Left-arc* and *right-arc* are both reduce actions, but they add arcs between different word pairs. Arc label predictions are conditioned on the same context as transition predictions. Right-arc adds a dependency of which σ_1 is the head, but the dynamic program does not allow conditioning on it when making transition decisions. However, we found that this does not actually decrease performance.

The dynamic program for the arc-hybrid parser has the same structure as the shift-reduce model. The marginal probability is independent of arc directionality, as it does not influence future decisions. Consequently unsupervised training based on this model cannot learn to predict arc directions. Exact decoding is performed with the Viterbi algorithm: At every item $[i, j]$ the highest scoring arc direction is recorded. After the most likely transition sequence is extracted, arc labels are predicted greedily.

3.2 Arc-eager parser

The arc-eager parser has four transitions, as defined in Table 2. *Shift* and *right-arc* are shift actions, while *left-arc* and *reduce* are reduce actions. However the two reduce actions, reduce and left-arc, are always mutually exclusive; the former is only valid if the stack top has already been assigned a head (through a previous right-arc) and the latter only if the stack top is not headed. To keep track of which actions are valid, the state configuration and the dynamic program are augmented to record whether elements on the stack are headed. As with arc-hybrid, we decompose the transition probability into deciding between shifting and reducing, and then predicting directionality. In this case, the shift decision decomposes into *shift* and *right-arc* transitions, where *shift* is implicitly deciding that the shifted word will be reduced through a left-arc. Consequently the only real difference between the arc-hybrid and arc-eager transition systems under dynamic programming is the information conditioned on when arc directionality is predicted.

A different deduction system is defined for arc-eager, although it follows the same structure as the shift-reduce one. Items have the form $[i^c, j]$, where c is a binary variable indicating whether node i is headed. The axiom and goal are $[0^0, n]$ and $[0^0, 1]$, respectively. The deduction rules are

$$\begin{aligned}
[i^c, j] &\rightarrow [j^0, j+1] && \text{(shift)}\\
[i^c, j] &\rightarrow [j^1, j+1] && \text{(right-arc)}\\
[i^c, k][k^0, j] &\rightarrow [i^c, j] && \text{(left-arc)}\\
[i^c, k][k^1, j] &\rightarrow [i^c, j] && \text{(reduce)}
\end{aligned}$$

The inside algorithm for arc-eager parsing is given in Algorithm 2. The algorithm is structured such that the inner loop computations (lines 8−22) can be vectorized, which is crucial for efficient

Action	State before	State after	Arc added	Probability
Shift	$(\sigma\|i, j)$	$(\sigma\|i\|j, j+1)$	-	$p_{tr}(\text{sh}\|h_i, h_{j-1})p_{gen}(w_j\|h_i, h_{j-1})$
Left-arc	$(\sigma\|i, j)$	(σ, j)	$j \to i$	$p_{tr}(\text{re}\|h_i, h_{j-1})p_{dir}(\text{la}\|h_i, h_{j-1})$
Right-arc	$(\sigma\|l\|i, j)$	$(\sigma\|l, j)$	$l \to i$	$p_{tr}(\text{re}\|h_i, h_{j-1})p_{dir}(\text{ra}\|h_i, h_{j-1})$
Shift	$(\sigma\|i^b, j)$	$(\sigma\|i^b\|j^0, j+1)$	-	$p_{tr}(\text{sh}\|h_i, h_{j-1})p_{dir}(\text{la}\|h_i, h_{j-1})p_{gen}(w_j\|h_i, h_{j-1})$
Right-arc	$(\sigma\|i^b, j)$	$(\sigma\|i^b\|j^1, j+1)$	$i \to j$	$p_{tr}(\text{sh}\|h_i, h_{j-1})p_{dir}(\text{ra}\|h_i, h_{j-1})p_{gen}(w_j\|h_i, h_{j-1})$
Left-arc	$(\sigma\|i^0, j)$	(σ, j)	$j \to i$	$p_{tr}(\text{re}\|h_i, h_{j-1})$
Reduce	$(\sigma\|i^1, j)$	(σ, j)	-	$p_{tr}(\text{re}\|h_i, h_{j-1})$

Table 2: The arc-hybrid (above) and arc-eager (below) transition systems. States represent (stack, current index).

Algorithm 2 Inside algorithm for arc-eager parser.

```
1:  for $j = 0, \ldots, n-1$ do
2:      $I(j^0, j{+}1) \leftarrow 1$
3:      $I(j^1, j{+}1) \leftarrow 1$
4:  end for
5:  for $gap = 2, \ldots, n$ do
6:      for $i = 0, \ldots, n - gap$ do
7:          $j = i + gap$
8:          for $c = 0, 1$ do
9:              for $k = i+1, \ldots, j-1$ do
10:                 if $j > 0$ then
11:                     $W(k) \leftarrow p_{\text{tr}}(\text{sh}|h_i, h_{k-1})$
12:                         $\cdot p_{\text{dir}}(\text{ra}|h_i, h_{k-1})p_{\text{gen}}(w_k|h_i, h_{k-1})$
13:                     $T(k) \leftarrow I(k^1, j)p_{\text{tr}}(\text{re}|h_k, h_{j-1})W(k)$
14:                 end if
15:                 if $j < n$ then
16:                     $V(k) \leftarrow p_{\text{tr}}(\text{sh}|h_i, h_{k-1})$
17:                         $\cdot p_{\text{dir}}(\text{la}|h_i, h_{k-1})p_{\text{gen}}(w_k|h_i, h_{k-1})$
18:                     $T(k) \leftarrow T(k) + I(k^0, j)p_{\text{tr}}(\text{re}|h_k, h_{j-1}))V(k)$
19:                 end if
20:             end for
21:             $I(i^c, j) \leftarrow \sum_{k=i+1}^{j-1} I(i^c, k)T(k)$
22:         end for
23:     end for
24: end for
25: return $I(0, n) + p_{tr}(\text{re}|h_0, h_{n-1})$
```

GPU implementation. At $\beta = n$, the dynamic program is restricted to allow only reduce transitions, requiring the remaining stack elements (apart from ROOT) to be headed. The Viterbi algorithm again follows the same structure as the inside algorithm: For every item $[i^c, j]$ the highest scoring splitting item k^b is recorded, where k is the splitting point and b indicates whether word k is headed or not, which corresponds to whether a reduce or left-arc is performed.

4 Experiments

We follow the standard setup for English dependency parsing, training on sections 2-21 of the Penn Treebank (PTB) Wall Street Journal corpus, using section 22 for development and section 23 for testing. Dependency trees follow the Stanford dependency (SD) representation (version 3.3.0) used in recent parsing research (Chen and Manning, 2014; Dyer et al., 2015). We also report some results using the older representation of Yamada and Matsumoto (2003) (YM). We follow Buys and Blunsom (2015b) and Dyer et al. (2016) in replacing training singletons and unknown words in the test set with unknown word class tokens based to their surface forms, following the rules implemented in the Berkeley parser.[1]

Our models are implemented in PyTorch, which constructs computation graphs dynamically.[2] During training, sentences are shuffled at each epoch, and minibatches are constructed of sentences of the same length. We base the hyperparameters of our models primarily on the language models of Zaremba et al. (2014). Models are based on two-layer LSTMs with embedding and hidden state size 650 with dropout of 0.5 on the RNN inputs and outputs. For all models weights are initialized randomly from the uniform distribution over $[-0.05, 0.05]$. Gradient norms are clipped to 5.0. The supervised parsers are trained with batch size 16, with an initial learning rate 1.0, which is decreased by a factor of 1.7 for every epoch after 6 initial epochs. The sequential LSTM baseline is trained with the same parameters, except that the learning rate decay is 1.4. The unsupervised models are trained with an initial learning rate 0.1, which is decreased by a factor of 2.0 for every epoch, with batch size 8.

We train and execute our models on a GPU, obtaining significant speed improvements over CPUs. For supervised training we also perform batch processing: After the sentences are encoded with an RNN, we extract the inputs to the transition, word and relation prediction models across the batch, and then perform the neural network computations in parallel. The supervised models' training speed is about 3 minutes per epoch.

[1]http://github.com/slavpetrov/berkeleyparser

[2]http://pytorch.org/

Model	Greedy	Exact
Arc-Hybrid uniRNN	84.12/81.54	84.21/81.61
Arc-Eager uniRNN	79.90/77.67	81.37/79.08
Arc-Hybrid biRNN	92.85/90.42	92.89/90.47
Arc-Eager biRNN	92.82/90.63	**92.90/90.68**
Arc-Hybrid Gen stack-next	56.98/52.22	82.77/78.01
Arc-Hybrid Gen buffer-next	85.25/82.83	**91.19/88.66**
Arc-Eager Gen buffer-next	80.79/78.56	87.34/84.84

Table 3: PTB development set parsing results (SD dependencies), reporting unlabelled and labelled attachment scores (UAS/LAS). Discriminative models (above the line) use either unidirectional or bidirectional RNNs.

Model	SD	YM
Buys and Blunsom (2015b)	90.10/87.74	90.16/88.83
Titov and Henderson (2007)	91.43/89.02	90.75/89.29
Arc-eager	88.20/85.91	87.61/86.36
Arc-hybrid	**91.01/88.54**	**90.71/88.68**

Table 4: PTB test set parsing results with supervised generative models, on the Stanford (SD) and Yamada and Matsumoto (2003) (YM) dependencies. The models from Buys and Blunsom (2015b) and Titov and Henderson (2007) were retrained to make results directly comparable.

4.1 Parsing

In order to benchmark parsing performance, we train discriminative baselines using the same feature space as the generative models. Unidirectional or bidirectional RNNs can be used; we see that the bidirectional encoder is crucial for accuracy (Table 3). The performance of our implementation is on par with than that of the arc-hybrid transition-based parser of Kiperwasser and Goldberg (2016), which obtains 93.2/91.2 UAS/LAS on the test set against 93.29/90.83 for our arc-hybrid model. State of the art parsing performance is 95.7%/94.1 UAS/LAS (Dozat and Manning, 2017).

Exact decoding is only marginally more accurate than greedy decoding, giving further evidence of the label bias problem. Andor et al. (2016) similarly showed that a locally normalised model without lookahead features cannot obtain good performance even with beam-search (81.35% UAS), while their globally normalised model can reach close to optimal performance without look-ahead. Shi et al. (2017) showed that globally normalised training improves the accuracy of these discriminative models.

Exact decoding is crucial to the performance of

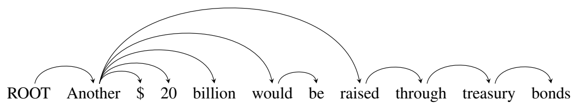

Figure 2: Sentence with dependencies induced by the unsupervised model.

the generative models (Table 3). They are much more accurate than the unidirectional discriminative models, which shows that the word prediction model benefits parsing accuracy. The arc-hybrid model is more accurate than arc-eager, as was the case for the unidirectional discriminative models. This can be explained by arc-eager making attachment decisions earlier in the transition sequence than arc-hybrid, which means that it has access to less context to condition these predictions on.

Our best generative model outperforms a previous incremental generative dependency parser based on feed-forward neural networks and approximate inference (Buys and Blunsom, 2015b) (Table 4). It is competitive with a previous RNN-based generative parser with a much more complex architecture than our model, including recurrent connections based on parsing decision (Titov and Henderson, 2007). Our exact decoding algorithm is also actually faster than the beam-search approaches for previous models, as it is implemented on GPU. Our arc-hybrid model parses 7.4 sentences per second, against 4 sentences per second for Buys and Blunsom (2015b) and approximately 1 sentence per second for Titov and Henderson (2007).

We also train the model as an unsupervised parser by directly optimizing the marginal sentence probability. The limitation of our approach is that our models cannot learn arc directionality without supervision, so we interpret shift as adding a (right-arc) dependency between top of the stack and the word being generated. In our experiments the model did not succeed in learning informative, non-trivial tree structures – in most cases it learns to attach words either to the immediate previous word or to the root. However, unsupervised dependency parsers usually require elaborate initialization schemes or biases to produce non-trivial trees (Klein and Manning, 2004; Spitkovsky et al., 2010; Bisk and Hockenmaier, 2015). An example dependency tree predicted by the unsupervised model is given in Figure 2.

Model	Perplexity
Interpolated Kneser-Ney 5-gram	170.09
Sequential LSTM (unbatched)	118.69
Sequential LSTM (batched)	**100.67**
Buys and Blunsom (2015b)	138.62
RNNG (Kuncoro et al., 2017)	**101.2**
Supervised (SD) shift-reduce buffer-next	111.53
Supervised (SD) shift-reduce stack-next	**107.61**
Unsupervised shift-reduce stack-next	125.20

Table 5: Language modelling perplexity results on the PTB parsing test set.

4.2 Language modelling

We apply our model to language modelling with both supervised and unsupervised training. The supervised models are trained as arc-hybrid parsers; the performance of arc-eager is almost identical as arc labels and directionality are not predicted. The unsupervised model is trained with only shift and reduce transitions as latent.

We evaluate language models with a sentence i.i.d. assumption. In contrast, the standard evaluation setup for RNN language models treats the entire corpus as single sequence. To evaluate the consequence of the sentence independence assumption, we trained a model on the most widely used PTB language modelling setup (Chelba and Jelinek, 2000; Mikolov et al., 2011), which uses a different training/testing split and preprocessing which limits the vocabulary to 10k. Our baseline LSTM obtains 92.71 test perplexity on this setup, against 78.4 of Zaremba et al. (2014), which uses the same hyperparameters without a sentence i.i.d. assumption. The syntactic neural language model of Emami and Jelinek (2005) obtained 131.3.

Results are reported in Table 5. Perplexity is obtained by exponentiating the negative log likelihood per token; end of sentence symbols are predicted but excluded from the token counts. As baselines without syntactic structure we use the interpolated Kneser-Ney n-gram model (Kneser and Ney, 1995) and vanilla LSTMs, trained with or without batching.

The LSTM baselines already outperform the syntactic feed-forward neural model of Buys and Blunsom (2015b). We see that there is a significant difference between training with or without mini-batching for the baseline; similarly our model's perplexities also improve when trained with batching. The batched baseline performs slightly better than Recurrent Neural Network Grammars (RNNG) (Dyer et al., 2016; Kuncoro et al., 2017), a constituency syntax-based RNN language model trained without batching.[3]

The results show that our syntactic language models perform slightly worse than the LSTM baseline. We experimented with different dependency representations on the development set, including SD, YM and Universal Dependencies (Nivre et al., 2016). We found little difference in language modelling performance between the dependency representations. Unsupervised training does not lead to better perplexity than the supervised models; however, due to much longer training times we did less hyperparameter tuning for the unsupervised model.

4.3 Analysis

We further analyze the probability distribution that our model is learning by calculating some perplexity-related quantities. We compare the perplexity of the marginal distribution $p(\mathbf{w})$ to the perplexity based only on the most likely transition sequence $\mathbf{a} = \text{argmax}\ p(\mathbf{w}, \mathbf{a})$, based on either the joint distribution $p(\mathbf{w}, \mathbf{a})$ or the conditional distribution $p(\mathbf{w}|\mathbf{a})$. Note that while the former is a bound on the marginal perplexity, the latter is not a true perplexity but simply helps us to quantify the contribution of the syntactic structure to reducing the uncertainty in the prediction.

The results (Table 6) show that the difference between the joint and marginal perplexities are relatively small for the supervised models, indicating that the distribution is very peaked around the most likely parse trees. However the conditional quantity shows that the syntactic structure encoded by the *stack-next* model is much more informative than that of the *buffer-next* model, although the only difference between them is the choice of elements to condition on when predicting the next word. Although the *stack-next* model has a better marginal perplexity, the disadvantage is that it has more uncertainty in the syntactic structure it is predicting (as can be seen by lower parsing accuracy) even though that structure is more informative.

The strength of RNNG over our approach is that it computes a compositional representation of

[3]Our experimental setup is the same as Dyer et al. (2016), except for a minor implementation difference in unknown word clustering; Dyer et al. (2016) reports 169.31 perplexity on the same IKN model.

Model	Marginal ppl	Argmax parse joint ppl	Argmax parse conditional ppl
RNNG (Dyer et al., 2016)	104.10	107.58	41.60
Supervised (SD) shift-reduce buffer-next	111.53	120.09	102.28
Supervised (SD) shift-reduce stack-next	107.61	119.20	71.27
Unsupervised shift-reduce stack-next	125.20	350.01	169.87

Table 6: Language modelling perplexity analysis on the PTB test set.

the stack and the partially constructed parse tree, while our model can only make use of the position on top of the stack and otherwise has to rely on the sequentially computed RNN representations. The disadvantage of RNNG is that inference can only be performed over entire sentences, as the proposal distribution for their importance sampling method is a discriminative parser. Exact inference allows our models to estimate next word probabilities from partially observed sequences.

5 Related work

5.1 Syntactic generative models

Chelba and Jelinek (2000) and Emami and Jelinek (2005) proposed incremental syntactic language models that predict binarized constituency trees with a shift-reduce model, parameterized by interpolated n-gram smoothing and feed-forward neural networks, respectively. Language modelling probabilities were approximated incrementally using beam-search. Rastrow et al. (2012) applied a transition-based dependency n-gram language model to speech recognition. These models obtained perplexity improvements primarily when interpolated with standard n-gram models, and were not employed as parsers.

Henderson (2004) proposed an incremental constituency parser based on recurrent neural networks that have additional connections to previous recurrent states based on the parser configuration at each time step. The generative version of this model was more accurate than the discriminative one. Titov and Henderson (2007) applied a similar approach to dependency parsing. Buys and Blunsom (2015a) and Buys and Blunsom (2015b) proposed generative syntactic models that are applied to both dependency parsing and language modelling, using Bayesian and feed-forward neural networks, respectively. Recurrent Neural Network Grammar (RNNG) (Dyer et al., 2016) is a generative transition-based constituency parser based on stack LSTMs (Dyer et al., 2015), that was also applied as a language model.

Recently, Shen et al. (2017) proposed an RNN-based language model that uses a soft gating mechanism to learn structure that can be interpreted as constituency trees, reporting strong language modelling performance. There has also been work on non-incremental syntactic language modelling: Mirowski and Vlachos (2015) proposed a dependency neural language model where each word is conditioned on its ancestors in the dependency tree, and showed that this model achieves strong performance on a sentence completion task.

5.2 Neural models with latent structure

There have been a number of recent proposals for neural abstract machines that augment RNNs with external memory, including stacks and other data structures that are operated on with differentiable operations to enable end-to-end learning. Neural Turing machines (Graves et al., 2014) have read-write memory that is updated at each timestep. Grefenstette et al. (2015) proposed a neural stack that is operated on with differentiable push and pop computations.

Another strand of recent work which our models are related to has proposed neural models with structured latent variables: Rastogi et al. (2016) incorporated neural context into weighted finite-state transducers with a bidirectional RNN, while Tran et al. (2016) proposed a neural hidden Markov model for Part-of-Speech (POS) induction. Yu et al. (2016) proposed a neural transduction model with polynomial-time inference where the alignment is a latent variable. Kim et al. (2017) proposed structured attention mechanisms that compute features by taking expectations over latent structure. They define a tree-structured model with a latent variable for head selection, along with projectivity constraints. The soft head selection learned by the model is used as features in an attention-based decoder.

Reinforcement learning has been proposed to learn compositional tree-based representations in the context of an end task (Andreas et al., 2016; Yogatama et al., 2016), but this approach has high variance and provide no guarantees of finding optimal trees.

6 Conclusion

We proposed a new framework for generative models of syntactic structure based on recurrent neural networks. We presented efficient algorithms for training these models with or without supervision, and to apply them to make online predictions for language modelling through exact marginalization. Results show that the model obtains state-of-the-art performance on supervised generative dependency parsing, but does not obtain better intrinsic language modelling performance than a standard RNN.

Acknowledgments

We thank members of the Oxford NLP group for discussions, Yejin Choi for valuable feedback, and the anonymous reviewers for their comments.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of ACL*. Association for Computational Linguistics, Berlin, Germany, pages 2442–2452. `http://www.aclweb.org/anthology/P16-1231`.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Learning to compose neural networks for question answering. In *Proceedings of NAACL*. pages 1545–1554. `http://www.aclweb.org/anthology/N16-1181`.

Yonatan Bisk and Julia Hockenmaier. 2015. Probing the linguistic strengths and limitations of unsupervised grammar induction. In *Proceedings of ACL*. Beijing,China, pages 1395–1404.

Jan Buys and Phil Blunsom. 2015a. A Bayesian model for generative transition-based dependency parsing. In *Proceedings of the 3rd International Conference on Dependency Linguistics (Depling)*.

Jan Buys and Phil Blunsom. 2015b. Generative incremental dependency parsing with neural networks. In *Proceedings of ACL-IJCNLP (short papers)*. pages 863–869.

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*.

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

James Cross and Liang Huang. 2016. Incremental parsing with minimal features using bi-directional LSTM. In *Proceedings of ACL*. page 32.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of ICLR*. `http://arxiv.org/abs/1611.01734`.

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*. pages 334–343. `http://www.aclweb.org/anthology/P15-1033`.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of NAACL*.

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP*. Association for Computational Linguistics, Austin, TX, pages 1–17. `http://aclweb.org/anthology/W16-5901`.

Ahmad Emami and Frederick Jelinek. 2005. A neural syntactic language model. *Machine Learning* 60(1-3):195–227. `https://doi.org/10.1007/s10994-005-0916-y`.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401* .

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*. pages 1828–1836.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of ACL*. Association for Computational Linguistics, page 95.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL*. pages 1077–1086. http://www.aclweb.org/anthology/P10-1110.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. 2017. Structured attention networks. In *Proceedings of ICLR*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics* 4:313–327.

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*. IEEE, volume 1, pages 181–184.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of ACL*. pages 673–682.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *Proceedings of EACL*. Association for Computational Linguistics, Valencia, Spain, pages 1249–1258. http://www.aclweb.org/anthology/E17-1117.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics* 4:521–535. https://www.transacl.org/ojs/index.php/tacl/article/view/972.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *ICASSP*. IEEE, pages 5528–5531.

Piotr Mirowski and Andreas Vlachos. 2015. Dependency recurrent neural language models for sentence completion. *CoRR* abs/1507.01193. http://arxiv.org/abs/1507.01193.

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

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. 2016. Universal dependencies v1: A multilingual treebank collection. In *Tenth International Conference on Language Resources and Evaluation (LREC 2016)*.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighting finite-state transductions with neural context. In *Proceedings of NAACL*.

Ariya Rastrow, Mark Dredze, and Sanjeev Khudanpur. 2012. Efficient structured language modeling for speech recognition. In *INTERSPEECH*. http://interspeech2012.org/accepted-abstract.html?id=1436.

Yikang Shen, Zhouhan Lin, Chin-Wei Huang, and Aaron C. Courville. 2017. Neural language modeling by jointly learning syntax and lexicon. *CoRR* abs/1711.02013. http://arxiv.org/abs/1711.02013.

Tianze Shi, Liang Huang, and Lillian Lee. 2017. Fast(er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Copenhagen, Denmark, pages 12–23. https://www.aclweb.org/anthology/D17-1002.

Stuart M Shieber, Yves Schabes, and Fernando CN Pereira. 1995. Principles and implementation of deductive parsing. *The Journal of logic programming* 24(1):3–36.

Valentin I Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2010. From baby steps to leapfrog: how less is more in unsupervised dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. pages 751–759.

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. http://www.aclweb.org/anthology/W/W07/W07-2218.

Ke M. Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. 2016. Unsupervised neural hidden markov models. In *Proceedings of the Workshop on Structured Prediction for NLP*. Association for Computational Linguistics, Austin, TX, pages 63–71. http://aclweb.org/anthology/W16-5907.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the International Conference on Parsing Technologies*. volume 3.

D. Yogatama, C. Dyer, W. Ling, and P. Blunsom. 2017. Generative and discriminative text classification with recurrent neural networks. *CoRR* abs/1703.01898. http://arxiv.org/abs/1703.01898.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. 2016. Learning to compose words into sentences with reinforcement learning. *CoRR* abs/1611.09100.

Lei Yu, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Tomas Kocisky. 2017. The neural noisy channel. *Proceedings of ICLR* .

Lei Yu, Jan Buys, and Phil Blunsom. 2016. Online segment to segment neural transduction. In *Proceedings of EMNLP*. pages 1307–1316.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *CoRR* abs/1409.2329. `http://arxiv.org/abs/1409.2329`.