Explicitly Modeling Syntax in Language Model improves Generalization

Yikang Shen
Mila/Université de Montréal
and Microsoft Research
Montréal, Canada
yi-kang.shen@umontreal.ca

Shawn Tan
Mila/Université de Montréal
Montréal, Canada
tanjings@mila.quebec

Alessandro Sordoni
Microsoft Research
Montréal, Canada

Siva Reddy
Mila/McGill University
Montréal, Canada

Aaron Courville
Mila/Université de Montréal
Montréal, Canada

Abstract

Syntax is fundamental to our thinking about language. Although neural networks are very successful in many tasks, they do not explicitly model syntactic structure. Failing to capture the structure of inputs could lead to generalization problems and over-parametrization. In the present work, we propose a new syntax-aware language model: Syntactic Ordered Memory (SOM). The model explicitly models the structure with a one-step look-ahead parser and maintains the conditional probability setting of the standard language model. Experiments show that SOM can achieve strong results in language modeling and syntactic generalization tests, while using fewer parameters then other models.

1 Introduction

Natural language is the result of the interplay between several linguistic phenomena such as syntax, semantics and pragmatics. Current language models confound these phenomena, making it hard to assess their linguistic ability. Several recent works systematically study the linguistic abilities of modern language models, especially syntax (Linzen et al., 2016; Marvin and Linzen, 2018; Gulordava et al., 2018). Their main findings are that 1) most language models are good at capturing frequent syntactic structures but do not generalize well to those in the long tail and 2) although some excel at having low perplexity scores, this is less due to their syntactic ability but more to their capturing collocations (frequently co-occurring words). More recently, Hu et al. (2020) show that RNNs are bad at syntactic generalization, whereas models that have an explicit notion of syntax fare well but at the cost of language modeling (higher perplexity). On the other hand, transformer-based models achieve strong performance when trained with large datasets, but are worse than random when trained on a small dataset.

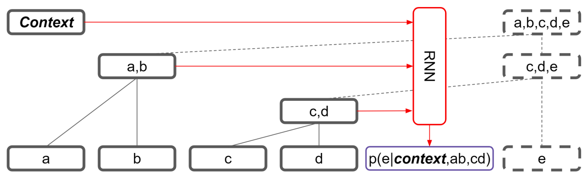


Figure 1: The mechanism of SOM. "*Context*" is a distributed representation of previous sentences. It could also represents the source sentence in a sequence to sequence task. It incrementally build subtrees given the input sentences. A RNN will takes the context representation and the representations of subtrees in the current sentence to predict next token.

In this work, we propose a new model, Syntactic Ordered Memory (SOM), that models explicit syntax and can also achieve strong language modeling and syntactic generalization performance on both small and large corpora. Inspired by psycholinguistic evidence (Marslen-Wilson, 1973; Altmann and Steedman, 1988; Tanenhaus et al., 1995) that humans language processing is incremental, SOM incrementally models syntactic structure, then use the structure as computation graph to predict the next token. In other words, the parser makes a syntactic decision that influences the language model predictions at each time step. The parse is represented as a distributed Ordered Memory (Shen et al., 2019), and the language model predicts a word conditioned on this ordered memory. Fig.1 shows the mechanism of SOM.

We also investigate if disentangling syntax from a language model improves generalization. Our motivation comes from the observation that syntax is autonomous from other linguistic phenomena, i.e., one can form syntactically valid sentences even if they are meaningless, e.g., the now infamous sentence from Chomsky (1956): *Colorless green ideas sleep furiously*. We make a language model competent at syntax through supervision from annotated data.

We compare our model with existing proposals of integrating syntax into language models. RNNGs (Dyer et al., 2016), Ordered Neurons (Shen et al., 2018) and Ordered Memory (Shen et al., 2019), in particular are closely related. RNNGs are generative models of language which define a joint distribution on syntactic structures and sequence of words. Ordered Neurons attempt to model the hierarchical structure of language by defining an ordering to the hidden states and the gates that impose that structure. Ordered Memory functions like a stacked based incremental parser with a one-step look-ahead. Consequently, it maintains a latent tree structure for the sequence. Based on the Ordered Memory, we propose the SOM which models a joint distribution like RNNG. But there are several important differences from previous models:

- Instead of using teacher forcing to train the incremental syntactic parser, we borrow techniques from imitation learning with our Adaptive Structural Loss. The new loss function generates structure labels on the fly, conditioning on the gold tree and faults made by the model earlier. This makes the model more robust at test time, given that it is directly trained to cope with its own parsing decisions.

- We introduce a dedicated prediction network to predict the next token. While the Ordered Memory use a one-step look-ahead parser to predict the structure for current input token, the prediction network is equipped with a zero-step look-ahead parser to predict the structure for next token.

- We provide a way to implement the autonomy of syntax in SOM, which uses a disentangled representation to model syntactic structure.

In terms of performance, our model achieves strong performance on language modeling, syntactic generalization, and incremental parsing tasks. We also find consistent evidence that our disentangled model achieve better generalization in the out-of-domain generalization.

2 Related Work

Evaluating Syntactic Generalization Recent tests have been developed that attempt to probe the linguistic abilities of language models, using some known rules of English grammar, and generating test cases (grammatical and ungrammatical) for language models. A good language model should then assign a grammatical sentence a higher probability. Gulordava et al. (2018) explores the extent to which RNNs are able to model grammar, independent of the semantics of the sentence. Marvin and Linzen (2018) evaluate language models on their ability to score sentences with and without the proper subject-verb agreements over a variety of different settings.

Hu et al. (2020) expands on these ideas, and propose a suite of syntactic generalization tests for language models over a series of different sized datasets. They find that while GPT-2 performs well, their performance is highly dependent on the scale of the language modeling training dataset, while other models remain more robust. In this paper, we use this test suite for the evaluation.

Incremental Parsing Given that an incremental parser processes a sentence from the start to the end, there are naturally some limitations. Hassan et al. (2009) show why either a beam or delay is necessary if performing incremental parsing with monotonic extensions: They experiment with a parser based on Combinatory Categorial Grammar (Steedman, 2000). Their parser achieves an accuracy of 86% when using look-ahead and performing greedy parsing (i.e. it does not use a beam). This accuracy drops significantly to 56% without look-ahead because the parser often commits to a structure incompatible with the continuation of a sentence.

The use of neural networks in incremental parsing is also not new. Costa et al. (2003) proposes using neural networks trained on gold trees for deciding a correct partial parse. Additionally, Nivre (2004) investigated the limits of incremental parsing as applied to dependency parses.

Autonomy of Syntax Applications of autonomous syntax to machine learning has been gaining some traction. Bailly and Gábor (2020) proposes a formal definition of autonomous syntax rooted in statistical independence. Russin et al. (2019) proposes a model with two "streams": one for the lexical semantics, and one for syntax, demonstrating that this separation performs well on the SCAN task (Lake and Baroni, 2017). Chen et al. (2019) uses a multi-task setting to disentangle syntax and semantics, using aligned corpora of

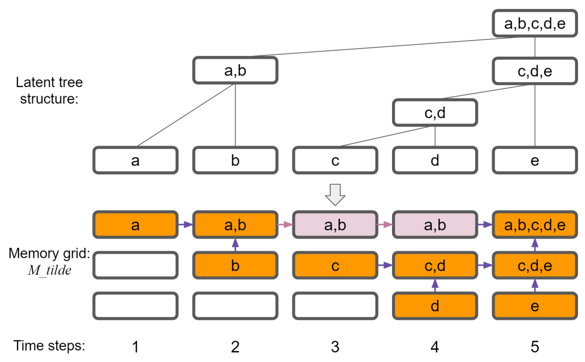


Figure 2: The grid view of a tree structure. Blue arrows represent composing children into parent. Gray arrows represent copying from previous time step. Orange slots are memories generated at the current time step. Gray slots are memories copied from previous time step.

paraphrases to separate two latent variables. They demonstrate that they perform well on both syntactic and semantic similarity benchmarks.

3 Model Architecture

3.1 Ordered Memory

The Ordered Memory (OM, Shen et al. 2019) is a recurrent neural network that explicitly models recursive structure through memory writing and erasing operations. OM maps the latent syntax into a $T \times N$ memory grid $\tilde{M}$, where T is the length of input sequence and N is the maximum number of memory slots. Figure 2 gives an intuition of what the grid contains. Empty blocks in the figure represent memory slots that can be discarded during inference. Ideally, the memory network should generate the t-th column of the grid $\tilde{M}_t$ at time step t. But generating $\tilde{M}_t$ requires that the model have accurate knowledge about the latent tree structure. For most NLP tasks, we do not have this information readily available. Instead, the model should learn to parse the input sequence to get the latent structure. Our goal in this paper is to build on OM, creating a language model that simultaneously builds the parse tree as it generates the text sequence.

As a recurrent model, OM performs one-step look-ahead incremental parsing through maintaining three states:

- Memory M_t: a matrix of dimension $N \times D$, where each occupied slot is a distributed representation for a node spanning an subsequence in $x_1, .., x_{t-1}$ conditioned on x_t, i.e. M_t represents one-step look-ahead parser stack. They represented by gray blocks in Figure 3b; if the model is making correct parsing decisions, then $M_t = \tilde{M}_{t-1}$.
- Candidate memory $\hat{M}_t$: a matrix of dimension $N \times D$ contains representations for all possible new nodes at time step t. At next time step $t+1$, the model will decide whether or not write these candidates into memory M_{t+1} conditioned on x_{t+1}. They are represented by orange blocks in Figure 3b.
- Memory mask $\overrightarrow{\pi}_t$: a binary of dimension N, where each element indicate whether the respective slot in $\hat{M}_t$ occupied by a candidate; for example, if $\overrightarrow{\pi}_t = (0,1,1)$, occupied slots are $\hat{M}_t^{\geq 2}$. At next time step, the model can only choose candidate from masked slots to write into the memory M_{t+1}.

At each time step, the model takes $[M_{t-1}, \hat{M}_{t-1}, \overrightarrow{\pi}_{t-1}]$ and word embedding x_t as inputs, returning the outputs $[M_t, \hat{M}_t, \overrightarrow{\pi}_t]$.

To generate the new memory M_t, we combine M_{t-1} and $\hat{M}_{t-1}$ to match $\tilde{M}_{t-1}$. The model uses x_t as its query to attend on previous candidates $\hat{M}_{t-1}$. The attention distribution is p_t, which models the split point of gray blocks and orange blocks in Figure 2. Suppose p_t is a one-hot distribution and $p_t^i = 1$. The candidates $\hat{M}_{t-1}^{\leq i}$ are written into the respective memory slot $M_t^{\leq i}$, while $M_{t-1}^{>i}$ are copied to $M_t^{>i}$:

$$M_t^{\leq i} = \hat{M}_{t-1}^{\leq i}, \quad M_t^{>i} = M_{t-1}^{>i} \tag{1}$$

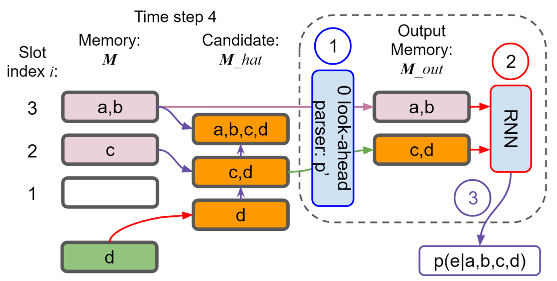
We will refer to the process of generating M_t as a 1 look-ahead parser, since the model is using the current input x_t as extra information to build the partial parse for time step $t-1$. To generate new candidates $\hat{M}_t$, the input embedding x_t is written into $\hat{M}_t^{i-1}$, and $\hat{M}_t^{\geq i}$ are computed recurrently with eq.3:

$$\hat{M}_t^{<i-1} = \emptyset, \quad \hat{M}_t^{i-1} = x_t \tag{2}$$

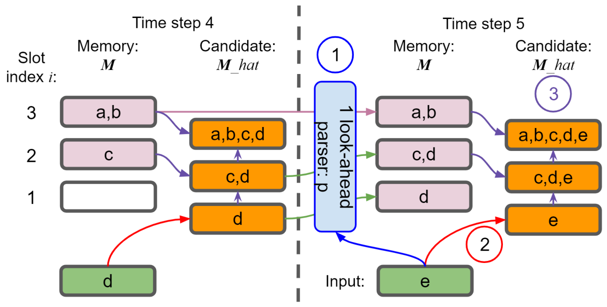
$$\hat{M}_t^{j} = \mathrm{cell}(M_t^j, \hat{M}_t^{j-1}), \forall j \geq i \tag{3}$$

where cell() is the composition function that takes children's representations as input and output parent's representation. The non-empty slots in candidate memory are then $\hat{M}_t^{\geq i-1}$, and they can be masked by:

$$\overrightarrow{\pi}_t^{<i-1} = 0, \quad \overrightarrow{\pi}_t^{\geq i} = 1 \tag{4}$$



(a) Predicting the next token at time step 4. ① The zero-step look-ahead parser combines M_t and $\hat{M}_t$ at time step t. ② The recurrent network token the combined memory M_t^{out} as input and output a hidden state $h_t = f(w_{\leq t})$. ③ h_t is then fed into an linear layer to compute $p(x_{t+1}|x_{\leq t})$.



(b) The transition from time step 4 to 5. ① The one-step look-ahead parser combines $\hat{M}_{t-1}$ and M_{t-1} considering on the current input x_t, in this example, the split point of $\hat{M}_{t-1}$ and M_{t-1} is $i = 2$. ② Current input x_t is written into the lower slot of new candidate memory $\hat{M}_t^{i-1}$. ③ The rest of new candidate memories $\hat{M}_t^{\geq i}$ are generated with bottom-up recurrent composition.

Figure 3: The prediction network (left) and recurrent transition (right) of SOM. The prediction network use a zero-step look-ahead parser to predict the location of the next phrase and acts as a prior on the syntactic structure. In the recurrent transition, a one-step look-ahead parser revises the syntax once e is actually observed and can be seen as a posterior over the syntax given the current word.

In other words, $\overrightarrow{\pi}_t^i = \sum_{j \leq i+1} p_t^j$, and $\overrightarrow{\pi}_t^i$ is monotonically increasing. More details of the OM can be found in Shen et al. (2019).

Augmenting OM, our proposal adds two extra components: the prediction network and the adaptive structural loss.

3.2 Prediction Network

At time step t, the prediction network takes $[M_t, \hat{M}_t, \overrightarrow{\pi}_t]$ as inputs, and outputs a probability distribution of next token $p(w_{t+1}|w_{\leq t})$. In order to predict the next token, we need to have a temporary estimate of the local structure. Thus, we need to approximate p_{t+1} with a zero-step look-ahead prediction p'_t:

$$\alpha_t^i = \frac{\mathbf{w}_2^{Att}\,\text{ReLU}\big(\mathbf{W}_1^{Att}\hat{M}_t^i + b_1\big) + b_2}{\sqrt{N}} \quad (5)$$

$$p'_t = \texttt{masked\_softmax}(\alpha_t, \text{mask} = \overrightarrow{\pi}_t) \quad (6)$$

where $\mathbf{W}_1^{Att}$ is $N \times N$ weight matrix, $\mathbf{w}_2^{Att}$ is a N dimension weight vector, and α_t^i is a scalar. Then we sample slot index i from the distribution p'_t. i is the zero-step look-ahead parsing decision, which means that the next phrase will be a sibling of node $\hat{M}_t^i$. Thus, we need to predict the next token considering on $\hat{M}_t^i$ and its previous contexts. So we feed memory slots $[M_t^N, M_t^{N-1}, ..., M_t^{i+1}, \hat{M}_t^i]$ into a recurrent neural network:

$$h_t = \texttt{RNN}\left(M_t^N, M_t^{N-1}, ..., M_t^{i+1}, \hat{M}_t^i\right) \quad (7)$$

where h_t is the final hidden state of the RNN. As shown in Figure 3a, the input sequence are representations of non-overlapping subtrees spanning from x_1 to x_t. h_t can therefore be seen as a distributed representation of the sequence $w_{\leq t}$. In the RNN, we use the same architecture as the cell function in OM to model the recurrent transition function:

$$\begin{bmatrix} f_j \\ i_j \\ c_j \\ u_j \end{bmatrix} = \mathbf{W}_2^{Cell}\texttt{ReLU}\left(\mathbf{W}_1^{Cell}\begin{bmatrix} h_j \\ M_j \end{bmatrix} + b_1\right) + b_2 \quad (8)$$

$$h_j = \texttt{LN}(\sigma(f_j) \odot h_j + \sigma(i_j) \odot M_j + \sigma(c_j) \odot u_j) \quad (9)$$

where σ is the sigmoid function, LN is layer normalization function. After obtaining h_t, we can compute the distribution for next token and the language model loss:

$$p(w_{t+1}|w_{\leq t}) = \texttt{softmax}(\mathbf{W}_{emb}h_t + b) \quad (10)$$

$$L_{\text{LM}} = -\sum_t \log(p(w_{t+1}|w_{\leq t})) \quad (11)$$

3.3 Adaptive Structural Loss

We also include an adaptive structure loss that provides supervision signal for p_t and p'_t. One way to do this is to train the parser with teacher forcing, feeding the gold tree to the model, and having the model predict future decisions. However, teacher forcing makes the language model overfit on the gold tree, resulting in bad perplexity scores (Table 2). Instead, we allow the model to recover from

```
Data: $\theta_1, ..., \theta_T, \Gamma$
Result: $\xi_1, ..., \xi_T$
initialize $\xi_1 = N$;
for $i \leftarrow 2$ to $T$ do
    $j = \texttt{first\_sibling}_\Gamma(i)$;
    $\mu_i = \max(\theta_{j+1}, ..., \theta_{i-1})$;
    $\xi_i = \max(\xi_j - 1, \mu_i)$;
end
```

Algorithm 1: The structure label generation algorithm, where Γ is the ground-truth tree and θ_i is the structural decisions made by our model. The algorithm produces a parse close to the original given the errors already made, and that new gold parse is converted into grid decisions. Given Γ, the function $\texttt{first\_sibling}_\Gamma(i)$ returns the index of the first token in the smallest clause that contains w_i, and where w_i is not the first token. Ideally, w_i should be written into the slot $(\xi_j - 1)$. For example, in Figure 2, c is written into the slot 2, then d, e should be written into the slot 1. However, the model could make a wrong decision between w_j and w_i. If the model has merged information from w_j into a higher slot μ_i, x_i should be written into slot μ_i as well.

its mistakes. To do this, we build the structure label for each time step based on the gold tree and previous decisions made by the model. During training, we sample the model's decision from p_t:

$$\theta_t = \texttt{Multinomial}(p_t) \tag{12}$$

and we make greedy decisions during evaluation:

$$\theta_t = \texttt{argmax}(p_t) \tag{13}$$

The same operations are applied to ${p'}_t$ as well.

We use the Algorithm.1 to convert the gold tree Γ into labels ξ_t for p_t. Since the zero-step look-ahead distribution ${p'}_t$ should match the one-step look-ahead distribution p_{t+1} at next time step $t+1$, we use ξ_{t+1} as label for ${p'}_t$. The structure loss is the negative log-likelihood:

$$L_S = -\sum_t \left(\log(p_t(\xi_t|w_{\leq t})) + \log({p'}_t(\xi_{t+1}|w_{\leq t}))\right)$$

For our model, the depth of Γ has a linear relation to the computational complexity and GPU memory consumption. To maximize the model's

Type	Max	Median	Mean
Constituency	29	7	7.7
Dependency	16	4	4.2

Table 1: Statics of tree depth for Penn Treebank. Dependency tree are converted from constituency tree with Stanford Corenlp toolkit.

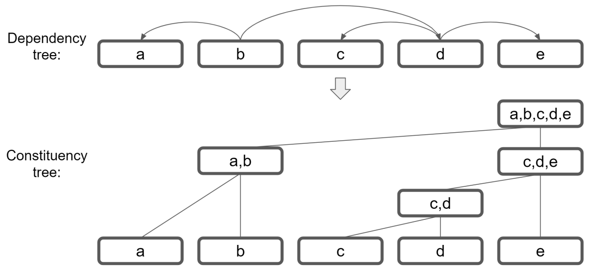


Figure 4: The universal dependency tree is converted into a constituency tree Γ through merging the head and its children into one single constituent. Since the grid view only works with binary trees, we binarize n-ary nodes with a left branching bias.

efficiently, the gold tree Γ is constructed from *universal dependency trees*.[1] There are two reasons we chose universal dependency trees instead of constituency trees: 1) In Table 1, the dependency trees are on average shallower than constituency trees; this means faster computation time and less memory consumption for our model. 2) Universal dependency trees can be applied to many more languages than Penn Treebank-style constituency grammar. Additionally, Penn Treebank-style trees can be easily converted to universal dependency trees. As shown in Figure 4, we convert the universal dependency tree into Γ by merging the head and its children into one single constituent.

3.4 Disentangling Semantic and Syntactic representations

Given the architecture of our model, we can easily disentangle the language model information flow and parsing information flow. Figure 5 illustrates the disentangled information and gradient flows in our model. The language model depends on both semantic and syntactic inputs, and it can backpropagate into both, because we need to predict both semantic and syntactic features of the next token. However, while the structure also depends on both inputs, it can only backpropagate into the syntactic input. This is because we want the parsing component to function independently, but still leverages the semantic information to deal with syntactic ambiguity.

[1] https://universaldependencies.org/

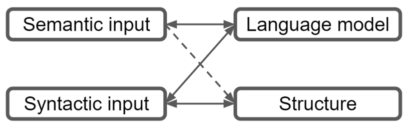


Figure 5: The schema of disentangling syntax from the language model. Solid lines represent dependency during inference, and gradients flow back during backpropagation. The dashed line represents the dependency during inference, but detached so that the gradients do not flow back during backpropagation.

It is possible that existing model architectures could implicitly learn to split these representations, even without the explicit disentanglement that we proposed here. Yet, Table 2 shows that entangled model can actually achieve stronger in-domain performance, thanks to the liberty to allocate capacity to the two different functionalities based on the training set. However, this disentanglement provides us a way to achieve autonomous syntax. In practice, this means that we could potentially achieve better generalization on out-of-domain data, as shown in Table 5.

To do so, we propose splitting word embeddings, memory slots, and intermediate hidden states into two segments: semantic segment and syntactic segment. We then replace linear layers in our cell functions with the following function:

$$\begin{bmatrix} y_{sem} \\ y_{syn} \end{bmatrix} = \begin{bmatrix} \mathbf{W}_{sem2sem} & \mathbf{W}_{syn2sem} \\ 0 & \mathbf{W}_{syn2syn} \end{bmatrix} \begin{bmatrix} x_{sem} \\ x_{syn} \end{bmatrix}$$

where y_{sem} and x_{sem} are the semantic segment which is optimized to minimize language modeling loss, y_{syn} and x_{syn} are the syntactic segment which is optimized to minimize both parsing and language modeling loss. This architecture results in the solid lines in Figure 5. Additionally, layer normalization functions are replaced with two separate functions for the two segments respectively. Meanwhile, p_t still depends on both semantic and syntactic segment, but the structural loss does not backpropagate into the semantic segment:

$$p_t = f(x_{t,sem}, x_{t,syn}, \hat{M}_{t,sem}, \hat{M}_{t,syn}) \tag{14}$$

$$\frac{\partial p_t}{\partial x_{t,sem}} = 0, \quad \frac{\partial p_t}{\partial \hat{M}_{t,sem}} = 0 \tag{15}$$

and the same for p'_t:

$$p'_t = f(\hat{M}_{t,sem}, \hat{M}_{t,syn}) \tag{16}$$

$$\frac{\partial p'_t}{\partial \hat{M}_{t,sem}} = 0 \tag{17}$$

This gradient detachment is represented by the dash line in Figure 5. In the experiment section, the disentangled models are denoted as dSOM, and entangled models are denoted as eSOM. For dSOM, the dimension of semantic and syntactic segments for memory slots are denoted as D_{sem} and D_{syn} respectively.

4 Experiments

We present the results of SOM on language modeling, syntactic generalization, and incremental parsing. Details of hyperparameters and experiment settings can be found in Appendix A.

4.1 Language Modeling

Penn Treebank has one million words of 1989 Wall Street Journal corpus annotated with constituency trees. Since SOM primarily focuses on sentence-level structure and language modeling, we use the same preprocessing schema as RNNG[2] (Dyer et al., 2016). Sentences are modeled separately, punctuation is retained, and singleton words are replaced with the Berkeley parser's mapping rules[3], resulting in 23,815-word types. Orthographic case distinction is preserved, and numbers (beyond singletons) are not normalized.

BLLIP is a large Penn Treebank-style parsed corpus of approximately 24 million sentences. We train and evaluate SOM on three splits of BLLIP: BLLIP-XS (40k sentences, 1M tokens), BLLIP-SM (200K sentences, 5M tokens), and BLLIP-MD (600K sentences, 14M tokens). They are obtained by randomly sampling sections from BLLIP 1987-89 Corpus Release 1. All models are tested on a shared held-out tested set.

Following the settings provided in (Hu et al., 2020), datasets are preprocessed into two different versions. The first setting is similar to the PTB dataset. Singleton words are mapped to UNK classes that preserve fine-grained information, such as orthographic case distinctions and morphological suffixes (e.g. `UNK-ed`, `UNK-ly`). The second setting use subword-level vocabulary extracted from the GPT-2 pretrained model rather than the BLLIP training corpora.

Out-of-domain Test set contains testsets from other English universal dependencies treebanks. It

[2] 2-21 for training, 24 for validation, 23 for evaluation.
[3] http://github.com/slavpetrov/berkeleyparser

Model	# parameters	ppl	p acc	UF1	p' acc
eSOM	17.7M	**77.68**	**0.927**	**87.96**	**0.870**
dSOM	16.3M	78.93	0.922	86.81	0.864
eSOM − Language model loss	17.7M	–	0.925	86.26	0.863
dSOM − Adaptive structure loss + Structural teacher forcing loss	16.3M	129.27	0.913	86.58	0.849
dSOM − Prediction network	13.0M	83.63	0.923	87.09	–
dSOM − Gold tree labels + Left-branching tree labels	16.3M	82.01	–	–	–
dSOM − Predicted tree + Gold tree	16.3M	60.87	0.947	–	0.884

Table 2: Ablation tests on the PTB dataset. "p acc" and "p' acc" are the prediction accuracies of the one-step look-ahead and zero-step look-ahead parsers respectively. "UF1" is the parsing performance with respect to the converted constituency tree Γ. "− Prediction network" means that the model uses the last candidate memory slot $\hat{M}_t^N$ to predict the next token, instead of using the h_t from the prediction network. "− Predicted tree + Gold tree" replace model's parsing decisions with ground truth decisions; the results can be considered as the performance upper bound of SOM.

Model	PTB
Without annotations	
RNNLM	93.2
PRPN(Shen et al., 2017)	96.7
URNNG(Kim et al., 2019)	90.6
With annotations	
RNNG(Dyer et al., 2016)	88.7
RNNG $\rightarrow$ URNNG(Kim et al., 2019)	85.9
dSOM	78.93
eSOM	**77.68**

Table 3: Perplexities on Penn Treebank datasets. "With annotation" means that the model uses the gold tree as supervision signal during training. Baseline results are from Kim et al. (2019)

Model	XS	SM	MD
n-gram	240.21	157.60	106.09
RNNG	122.46	86.72	69.57
LSTM	98.19	65.52	59.05
ON-LSTM	71.76	54.00	56.37
GPT-2	529.90*	183.10*	37.04*
dSOM	**69.28**	51.60	32.17*
eSOM	70.41	**51.47**	**31.95***

Table 4: Perplexities on BLLIP datasets achieved by different models. Perplexity scores across training dataset sizes are not strictly comparable for models that use word-level vocabulary. * results are using GPT-2's subword vocabulary.

contains corpora of different genres, including academic, email, blog, fiction, legal, news, etc. We use these datasets to test the generalization ability of models that are trained on PTB.

Experiment Results of language modeling are given in Table 3 and Table 4. SOM consistently outperforms both the annotated model and non-annotated models. While GPT-2 seems to fail to learn on smaller datasets, SOM still outperforms GPT-2 on the BLLIP-MD dataset with far fewer parameters (34.8M vs 124.4M), and achieves comparable results with the GPT-2 that is trained on a 3 times larger dataset BLLIP-LG (Hu et al., 2020).

The ablation test results are shown in Table 2. The biggest performance drop comes from replacing the adaptive structure loss with teacher forcing loss. We believe that it's because the language model part overfits on the gold tree, and suffers from the exposure bias during the evaluation. Another big performance drop happens after removing the prediction network. This suggests that predicting the attaching nodes of the next phrase with the zero-step look-ahead parsers helps to predict the next token. Replacing the gold tree labels with trivial left-branching tree labels also hurts the perplexity. This prove that syntactic structure helps language modeling. Among the proposed models, the eSOM has the best performance on the in-domain PTB test set. But Table 5 shows that the dSOM slightly outperforms eSOM in perplexity on out-of-domain test sets. This result is coherent with our intuition that the autonomous syntax component helps generalization, because syntax is shared across domains.

4.2 Syntactic Generalization

Syntactic Generalization (SG) test suites evaluate the syntactic knowledge of neural language models. Hu et al. (2020) proposed a set of 34 test suites to evaluation six different aspects of syntax: 1) agreement, 2) licensing, 3) garden-path effects, 4) gross syntactic expectation, 5) center embedding, 6) long-distance dependencies.

Following the settings in Hu et al. (2020), we evaluate our language models trained on BLLIP datasets. Language models are presented with a group of sentences with minor differences. To succeed in the test, the model needs to assign higher conditional probabilities to designated phrases in the sentence that are more grammatical.

Dataset	GUM		EWT		ParTUT		LinES		Pronouns		PUD		SG
Metric	**ppl**	**UF1**	**ppl**	**UF1**	**ppl**	**UF1**	**ppl**	**UF1**	**ppl**	**UF1**	**ppl**	**UF1**	**acc**
eSOM	351.5	**67.0**	**400.7**	**73.1**	282.6	**76.5**	253.3	**67.2**	**552.1**	87.1	281.0	**75.6**	**0.614**
dSOM	**350.3**	66.4	403.0	72.3	**269.8**	74.3	**252.1**	66.6	565.8	**87.3**	**280.3**	75.1	0.581
LB	375.5	–	436.5	–	300.9	–	267.5	–	620.4	–	300.7	–	0.513

Table 5: Out of domain test results. Models are trained on PTB. The test sets are obtained from English universal dependencies treebank. "LB" stands for left-branching tree labels. Thanks to the structure information, our models generalize much better then the left-branching baseline.

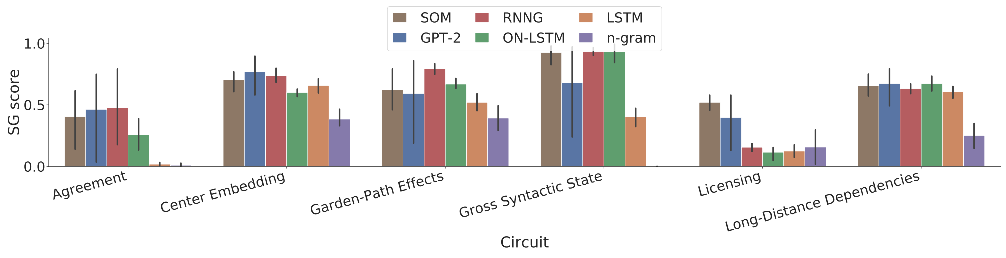


Figure 6: Evaluation results on all models, split across test suite circuits.

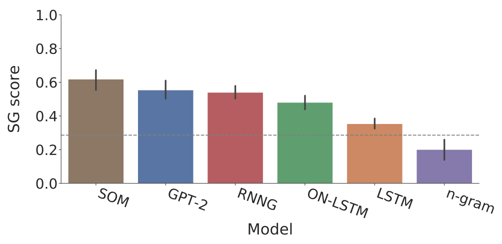


Figure 7: Average SG accuracy by model class.

Figure 7 shows the average accuracy over all model on the complete set of SG test suites. SOM achieves the best average accuracy, outperforms models with hierarchical structure bias (RNNG, ON-LSTM), and transformer-based model (GPT-2). However, according to Figure 8a in Appendix B.2, GPT-2 trained on BLLIP-LG and BLLIP-MD still outperform SOM. This could due to that the number of parameters in SOM is largely falling behind GPT-2.

Figure 6 provides fine-grained results on six SG classes. SOM achieves strong performance on licensing, gross syntactic state, center embedding, and long-distance embeddings. These classes require the model to keep track of syntactic features across large syntactic chunks (e.g., relative or subordination clauses). SOM can effectively keep this long-term information in higher-level memory slots, and revisit the information after the clause in the middle is ended. More detailed results can be found in Appendix B.2.

4.3 Incremental Parsing

To evaluate SOM's performance on incremental parsing, we compare the parsing results given by SOM with binary constituency trees Γ converted from universal dependency trees.[4] The parsing results are given in Table 2. We first observe that using teacher forcing loss results in the worst parsing performance. This suggests that our adaptive structure loss indeed help the model to learn a better parser. We observe that the eSOM provides the best parsing performance, that's because eSOM could allocate more capacity to the parsing component. After removing the language model loss, the UF1 drops 1.7 points. This shows that language model loss helps the model to learn better representations for syntax. We also trained and evaluated our models on the original PTB constituency trees. Results can be found in Appendix B.1.

5 Conclusion

In this work, we propose a new language model with an integrated incremental parser. It models the joint distribution of syntactic structure and sequence words. We find that by using the Adaptive Structure Loss and explicitly modeling the syntax, we can achieve strong performance on language modelling and syntactic generalization. We also see the model generalizes well to out-of-distribution test data. Our results suggest that syntax is an important factor if we seek to generalise to new scenarios or be robust to domain shifts.

[4]UF1 scores are computed by EVALB `https://nlp.cs.nyu.edu/evalb/`

References

G Altmann and M Steedman. 1988. Interaction with context during human sentence processing. *Cognition*, 30(3):191–238.

Raphaël Bailly and Kata Gábor. 2020. Emergence of syntax needs minimal supervision. *arXiv preprint arXiv:2005.01119*.

Mingda Chen, Qingming Tang, Sam Wiseman, and Kevin Gimpel. 2019. A multi-task approach for disentangling syntax and semantics in sentence representations. *arXiv preprint arXiv:1904.01173*.

Noam Chomsky. 1956. *Syntactic structures*. Mouton Publishers.

Fabrizio Costa, Paolo Frasconi, Vincenzo Lombardo, and Giovanni Soda. 2003. Towards incremental parsing of natural language using recursive neural networks. *Applied Intelligence*, 19(1-2):9–25.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776*.

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. 2018. Colorless green recurrent networks dream hierarchically. In *Proceedings of NAACL-HLT*, pages 1195–1205.

Hany Hassan, Khalil Sima'an, and Andy Way. 2009. Lexicalized semi-incremental dependency parsing. In *Proceedings of the International Conference RANLP-2009*, pages 128–134.

Jennifer Hu, Jon Gauthier, Peng Qian, Ethan Wilcox, and Roger P Levy. 2020. A systematic assessment of syntactic generalization in neural language models. *arXiv preprint arXiv:2005.03692*.

Yoon Kim, Alexander M Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019. Unsupervised recurrent neural network grammars. *arXiv preprint arXiv:1904.03746*.

Brenden M Lake and Marco Baroni. 2017. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. *arXiv preprint arXiv:1711.00350*.

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.

William Marslen-Wilson. 1973. Linguistic structure and speech shadowing at very short latencies. *Nature*, 244(5417):522–523.

Rebecca Marvin and Tal Linzen. 2018. Targeted syntactic evaluation of language models. *arXiv preprint arXiv:1808.09031*.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the workshop on incremental parsing: Bringing engineering and cognition together*, pages 50–57.

Jake Russin, Jason Jo, Randall C O'Reilly, and Yoshua Bengio. 2019. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *arXiv preprint arXiv:1904.09708*.

Yikang Shen, Zhouhan Lin, Chin-Wei Huang, and Aaron Courville. 2017. Neural language modeling by jointly learning syntax and lexicon. *arXiv preprint arXiv:1711.02013*.

Yikang Shen, Shawn Tan, Arian Hosseini, Zhouhan Lin, Alessandro Sordoni, and Aaron C Courville. 2019. Ordered memory. In *Advances in Neural Information Processing Systems*, pages 5038–5049.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. 2018. Ordered neurons: Integrating tree structures into recurrent neural networks. *arXiv preprint arXiv:1810.09536*.

Mark Steedman. 2000. *The syntactic process*, volume 24. MIT press Cambridge, MA.

Michael K Tanenhaus, Michael J Spivey-Knowlton, Kathleen M Eberhard, and Julie C Sedivy. 1995. Integration of visual and linguistic information in spoken language comprehension. *Science*, 268(5217):1632–1634.

A Hyperparameters

Dropout is applied before all linear layers in our model. They all share the same dropout rate, except the dropout before language model output layer has a different rate. We also applied embedding dropout which randomly set some embedding vectors to 0. Hyperparameters are chosen based on the perplexity on validation set.

Dataset	D_{sem}	D_{syn}	#slots	embedding dropout	dropout	output dropout
PTB	300	100	15	0.1	0.3	0.5
BLLIP-XS	300	100	15	0.1	0.3	0.5
BLLIP-SM	400	100	15	0.1	0.2	0.2
BLLIP-MD-BPE	400	100	15	0	0.1	0.1

Table 6: Hyperparameters. The hidden size of eSOM models are always the sum of D_{sem} and D_{syn}

Model	**XS**	**SM**	**MD**
RNNG	22.8M	48.4M	81.1M
LSTM	13.4M	30.5M	52.2M
ONLSTM+AWD	30.8M	44.2M	61.2M
GPT-2	124.4M	124.4M	124.4M
dSOM	16.4M	39.5M	34.8M
eSOM	17.8M	41.4M	37.9M

Table 7: Parameter counts for different models

B More Experiment Results

B.1 Incremental Constituency Parsing

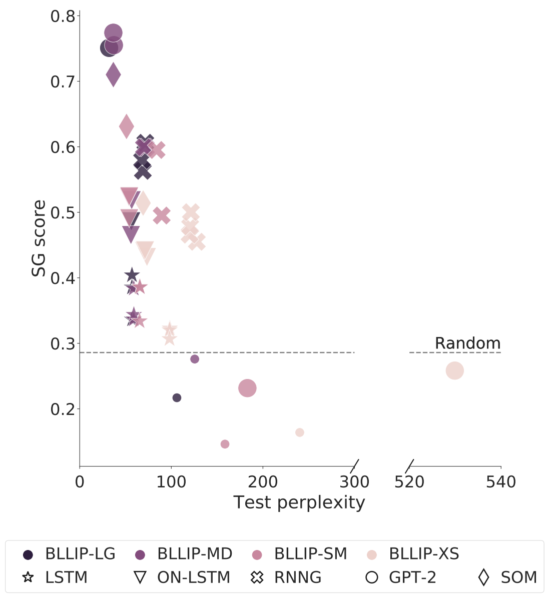
We also trained and evaluated our models on the original Penn Treebank constituency trees. Such that, we can compare our model's parsing performance with previous models. As for perplexity, constituency tree based models achieve marginally results as dependency tree based counterparts. But constituency tree based models require 2× GPU time and memory to train and evaluate.

Model	**UF1**
Not Incrmental	
URNNG*	40.7
RNNG	68.1
RNNG → URNNG	67.7
Incrmental	
PRPN*	41.2
ONLSTM*	47.7
Shift-reduce baseline	56.82
Baseline + LM + ASLoss	58.04
dSOM	66.83
eSOM	67.27
Oracle Binary Trees	82.5

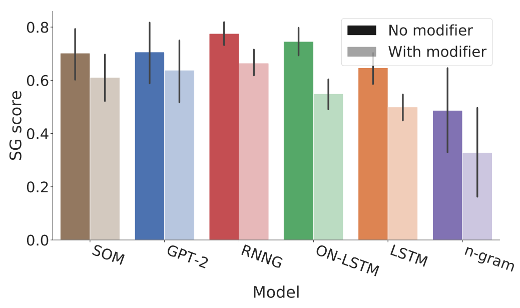
Table 8: Parsing results on PTB. "*" means that the model is doing unsupervised grammar induction. Baseline results are provided in Kim et al. (2019).

As shown in Table.8, our incremental parsers achieve similar performance as non-incremental parsing. Following the settings in Kim et al. (2019), we calculate unlabeled F1 using evalb, which ignores punctuation and discards trivial spans (width-one and sentence spans). Since we compare UF1 against the original, nonbinarized trees (per convention), UF1 scores is upper bounded by the oracle binary trees score.

B.2 Syntactic Generalization results



(a) Relationship between SG socre and perplexity on the held-out BLLIP test set.



(b) SG score on the pairs of test suites with and without intervening modifiers: Center Embedding, Cleft, MVRR, NPZ-Ambiguous, and NPZ-Object.

Figure 8

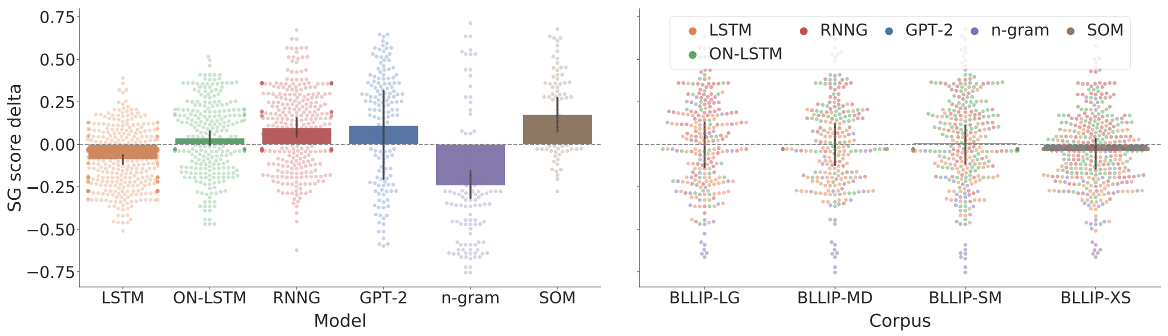
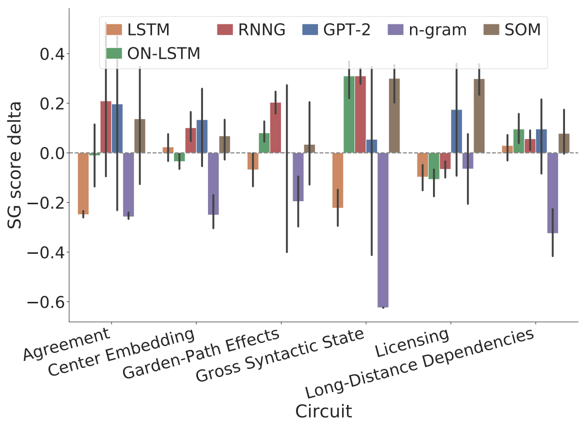


Figure 9: Left: Model class has a trong effect on SG scores. Right: Data scale has little effect on SG scores



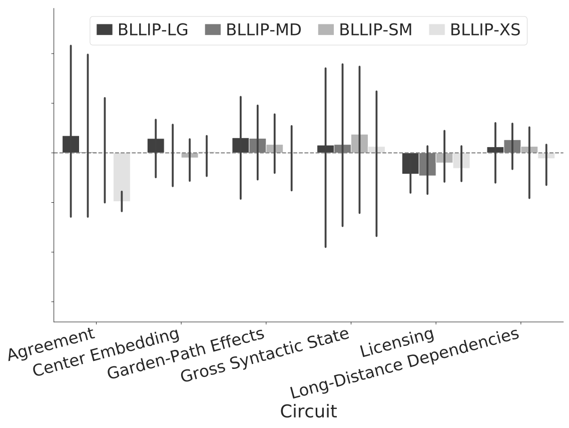


Figure 10: Left: differences in model class induce significant differences in SG scores for several circuits. Right: differences in training data size do not reliably account for most of circuits.