PaLM: A Hybrid Parser and Language Model

Hao Peng[♠] **Roy Schwartz**[♠◇] **Noah A. Smith**[♠◇]
[♠]Paul G. Allen School of Computer Science & Engineering, University of Washington
[◇]Allen Institute for Artificial Intelligence
{hapeng,roysch,nasmith}@cs.washington.edu

Abstract

We present PaLM, a hybrid **pa**rser and neural **l**anguage **m**odel. Building on an RNN language model, PaLM adds an attention layer over *text spans* in the left context. An unsupervised constituency parser can be derived from its attention weights, using a greedy decoding algorithm. We evaluate PaLM on language modeling, and empirically show that it outperforms strong baselines. If syntactic annotations *are* available, the attention component can be trained in a supervised manner, providing syntactically-informed representations of the context, and further improving language modeling performance.

1 Introduction

Recent language models have shown very strong data-fitting performance (Jozefowicz et al., 2016; Merity et al., 2018). They offer useful products including, most notably, contextual embeddings (Peters et al., 2018; Radford et al., 2019), which benefit many NLP tasks such as text classification (Howard and Ruder, 2018) and dataset creation (Zellers et al., 2018).

Language models are typically trained on large amounts of raw text, and therefore do not explicitly encode any notion of structural information. Structures in the form of syntactic trees have been shown to benefit both classical NLP models (Gildea and Palmer, 2002; Punyakanok et al., 2008; Das et al., 2012, *inter alia*) and recent state-of-the-art neural models (Dyer et al., 2016; Swayamdipta et al., 2018; Peng et al., 2018b; Strubell et al., 2018, *inter alia*). In this paper we show that LMs can benefit from syntactically-inspired encoding of the context.

We introduce PaLM (**pa**rser and **l**anguage **m**odel; Fig. 1), a novel hybrid model combining an RNN language model with a constituency parser. The LM in PaLM attends over *spans of tokens*, implicitly learning which syntactic constituents are likely. A span-based parser is then derived from the attention information (Stern et al., 2017).

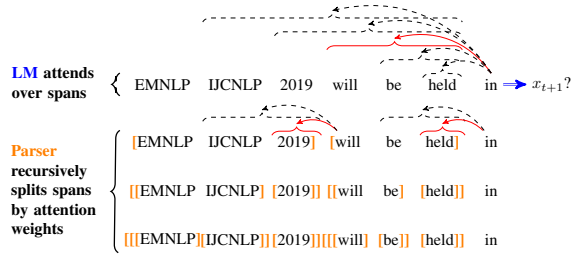

Figure 1: An illustration of PaLM. The LM (first line) predicts the next word (x_{t+1}, double blue arrow) by attending over previous spans ending in time $t-1$ (dashed lines). The parser (lines 2–4) splits the prefix into two spans (line 2) by the taking the top scoring attended span (red solid line) and the prefix leading to it. It then recursively splits the two sub-spans using the same procedure (line 3). Finally, spans of length two are trivially split into terminal nodes (line 4).

PaLM has several benefits. First, it is an intuitive and lightweight way of incorporating structural information (§2.1), requiring no marginal inference, which can be computationally expensive (Jelinek and Lafferty, 1991; Chelba and Jelinek, 1998; Roark, 2001; Dyer et al., 2016; Buys and Blunsom, 2018; Kim et al., 2019, *inter alia*). Second, the attention can be syntactically informed, in the sense that the attention component can optionally be supervised using syntactic annotations, either through pretraining or by joint training with the LM (§2.2). Last, PaLM can derive an unsupervised constituency parser (§2.2), whose parameters are estimated purely using the language modeling objective.

To demonstrate the empirical benefits of PaLM, we experiment with language modeling (§3). PaLM outperforms the AWD-LSTM model (Merity et al., 2018) on both the Penn Treebank

(PTB; Marcus et al., 1993) and WikiText-2 (Merity et al., 2017) datasets by small but consistent margins in the unsupervised setup. When the parser is trained jointly with the language model, we see additional perplexity reductions in both cases. Our implementation is available at https://github.com/Noahs-ARK/PaLM.

2 PaLM—Parser and Language Model

We describe PaLM in detail. At its core is an attention component, gathering the representations of preceding *spans* at each time step. Similar to self-attention, PaLM can be implemented on top of RNN encoders (Parikh et al., 2016), or as it is (Vaswani et al., 2017). Here we encode the tokens using a left-to-right RNN, denoted with vectors $\mathbf{h}_t$.[1]

Below we describe the span-attention component and the parsing algorithm. We use $[i, j], i \leq j$ to denote text span $x_i \dots x_j$, i.e., inclusive on both sides. When $i = j$, it consists of a single token.

2.1 Span Attention

We want the language model attention to gather context information aware of syntactic structures. A constituency parse can be seen as a collection of syntactic constituents, i.e., token spans. Therefore we attend over preceding spans.[2]

At step t, PaLM attends over the spans ending at $t-1$, up to a maximum length m, i.e., $\{[i, t-1]\}_{i=t-m}^{t-1}$.[3] Essentially, this can be seen as splitting the prefix span $[t-m, t-1]$ into two, and attending over the one on the right. Such a span attention mechanism is inspired by the top-down greedy span parser of Stern et al. (2017), which recursively divides phrases. In §2.2, we will use a similar algorithm to derive a constituency parser from the span attention weights.

Bidirectional span representation with rational RNNs. Meaningful span representations are crucial in span-based tasks (Lee et al., 2017; Peng et al., 2018c; Swayamdipta et al., 2018, *inter alia*). Typical design choices are based on start and end token vectors contextualized by bidirectional RNNs. However, a language model does not have access to future words, and hence running a backward RNN from right to left is less straightforward: one will have to start an RNN running at each token, which is computationally daunting (Kong et al., 2016). To compute span representations efficiently, we use *rational* RNNs (RRNNs; Peng et al., 2018a).

RRNNs are a family of RNN models, where the recurrent function can be computed with weighted finite-state automata (WFSAs). We use the unigram WFSA–inspired RRNN (Peng et al., 2018a), where the cell state update is

$$\mathbf{f}_t = \boldsymbol{\sigma}\left(\mathbf{W}_f \mathbf{h}_t\right), \tag{1a}$$

$$\mathbf{u}_t = \left(\mathbf{1} - \mathbf{f}_t\right) \odot \tanh\left(\mathbf{W}_u \mathbf{h}_t\right), \tag{1b}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{u}_t. \tag{1c}$$

$\mathbf{f}_t$ is a forget gate implemented with the elementwise sigmoid function $\boldsymbol{\sigma}$, and $\odot$ denotes elementwise multiplication. $\mathbf{W}_u$ and $\mathbf{W}_f$ are learned matrices. Bias terms are suppressed for clarity.[4]

Slightly overloading the notation, let $\overrightarrow{\mathbf{c}}_{i,j}$ denote the encoding of span $[i, j]$ by running a forward RRNN in Eq. 1, from left to right. It can be efficiently computed by subtracting $\overrightarrow{\mathbf{c}}_{i-1}$ from $\overrightarrow{\mathbf{c}}_j$, weighted by a product of forget gates:

$$\overrightarrow{\mathbf{c}}_{i,j} = \overrightarrow{\mathbf{c}}_j - \overrightarrow{\mathbf{c}}_{i-1} \bigodot_{k=i}^{j} \overrightarrow{\mathbf{f}}_k. \tag{2}$$

$\overrightarrow{\mathbf{f}}_k$ vectors are the forget gates. See §B for a detailed derivation.

Using this observation, we now derive an efficient algorithm to calculate the span representations based on bidirectional RRNNs. In the interest of space, Alg. 1 describes the forward span representations. It takes advantage of the distributivity property of rational RNNs (Peng et al., 2018a), and the number of RNN function calls is linear in the input length.[5] Although overall asymptotic time complexity is still quadratic, Alg. 1 only involves elementwise operations, which can be eas-

[1] We experiment with a strong LSTM implementation for language modeling (Merity et al., 2018), see §3.

[2] Standard token-based self-attention naturally relates to dependency structures through head selection (Strubell et al., 2018). In a left-to-right factored language model, dependencies are less natural if we want to allow a child to precede its parent.

[3] m is set to 20. This reduces the number of considered spans from $\mathcal{O}(n^2)$ to $\mathcal{O}(mn)$. Besides practical concerns, it makes less sense if a phrase goes beyond one single sentence (the average sentence length of WSJ training sentences is 21).

[4] Unlike other RNNs such as LSTM (Hochreiter and Schmidhuber, 1997) or GRU (Cho et al., 2014), RRNNs do not apply an affine transformation or a nonlinear dependency of $\mathbf{c}_t$ on $\mathbf{c}_{t-1}$.

[5] In contrast, the dynamic program of Kong et al. (2016) for segmental (span-scoring) RNNs requires a quadratic number of recurrent function calls, since they use LSTMs, where distributivity does not hold.

Algorithm 1 RRNN-based span representation.[6]

procedure SPANREPR($\{\overrightarrow{\mathbf{c}}_t, \overrightarrow{\mathbf{f}}_t\}$)
  ▷ Accumulate forward forget gates
  for $i = 1, \ldots, n$ **do**
    $\overrightarrow{\mathbf{f}}_{1,i} = \overrightarrow{\mathbf{f}}_{1,i-1} \odot \overrightarrow{\mathbf{f}}_i$
  end for
  for $j = 1, \ldots, n$ **do**
    for $i = 1, \ldots, j$ **do**
      $\overrightarrow{\mathbf{c}}_{i,j} = \overrightarrow{\mathbf{c}}_j - \overrightarrow{\mathbf{c}}_{i-1} \odot \overrightarrow{\mathbf{f}}_{1,j} / \overrightarrow{\mathbf{f}}_{1,i-1}$
    end for
  end for
  return $\overrightarrow{\mathbf{c}}_{i,j}$ vectors
end procedure

ily parallelized on modern GPUs. The backward one (right to left) is analogous.

Computing attention. As in standard attention, we use a normalized weighted sum of the span representations. Let $\mathbf{g}([i,j]) = [\overrightarrow{\mathbf{c}}_{i,j}; \overleftarrow{\mathbf{c}}_{i,j}]$ denote the representation of span $[i,j]$, which concatenates the forward and backward representations calculated using Alg. 1. The context vector $\mathbf{a}_t$ is

$$\mathbf{a}_{t+1} = \sum_{i=0}^{m-1} \omega_{t,i}\ \mathbf{g}([t-i,t]), \tag{3a}$$

$$\omega_{t,i} = \frac{\exp s_{t,i}}{\sum_{j=0}^{m-1} \exp s_{t,j}}. \tag{3b}$$

Here $s_{t,i}$ is implemented as an MLP, taking as input the concatenation of $\mathbf{h}_{t+1}$ and $\mathbf{g}([t-i,t])$ and outputs the attention score. The context vector is then concatenated with the hidden state $\bar{\mathbf{h}}_{t+1} = [\mathbf{h}_{t+1}; \mathbf{a}_{t+1}]$, and fed into onward computation.

In summary, given an input sequence, PaLM:

1. First uses a standard left-to-right RNN to calculate the hidden states $\mathbf{h}_t$.
2. Feed $\mathbf{h}_t$ vectors into a one-layer bidirectional rational RNN (Eq. 1), using Alg. 1 to compute the span representations.
3. Attends over spans (Eq. 3b) to predict the next word.

2.2 Attention-Based Constituency Parsing

We next describe the other facet or PaLM: the constituency parser. Our parsing algorithm is similar to the greedy top-down algorithm proposed by Stern et al. (2017). It recursively divides a span into two smaller ones, until a single-token span, i.e., a leaf, is reached. The order of the partition specifies the tree structure.[7] Formally, for a maximum span length m, at each time step $j+1$, we split the span $[j-m+1, j]$ into two smaller parts $[j-m+1, k_0]$ and $[k_0+1, j]$. The partitioning point is greedily selected, maximizing the attention scores of spans ending at j:[8]

$$k_0 = \operatorname*{argmax}_{k \in \{0,\ldots,m-1\}} s_{j,k}. \tag{4}$$

The span is directly returned as a leaf if it contains a single token. A full parse is derived by running the algorithm recursively, starting with the input as a single span (with a special end-of-sentence mark at the end). The runtime is $\mathcal{O}(n^2)$, with $n-1$ partitioning points. See Fig. 1 for an illustration.

Supervising the attention. Now that we are able to derive phrase structures from attention weights, we can further inform the attention if syntactic annotations are available, using oracle span selections. For each token, the gold selection is a m-dimensional binary vector, and then normalized to sum to one, denoted $\mathbf{y}_t$.[9] We add a cross-entropy loss (averaged across the training data) to the language modeling objective, with λ trading off between the two:

$$\mathcal{L} = \mathcal{L}_{\text{LM}} + \frac{\lambda}{N} \sum_{t=1}^{N} \mathcal{H}(\mathbf{y}_t, \boldsymbol{\omega}_t), \tag{5}$$

with $\boldsymbol{\omega}$ being the attention distribution at step t, and N the length of the training corpus. As we will see in §3, providing syntactically guided span attention improves language modeling performance.

Discussion. PaLM provides an intuitive way to inject structural inductive bias into the language model—by supervising the attention distribution. This setting can be seen as a very lightweight multitask learning, where no actual syntactic tree is predicted during language modeling training or evaluation. The attention weight predictor (i.e., the s scores in Eq. 3b) can be replaced with an off-the-shelf parser, or deterministically set (e.g., to simulate left/right-branching).

[6] / denotes elementwise division. Both elementwise product and division are implemented in log-space.

[7] It is only able to produce binarized unlabeled trees.

[8] Another natural choice is to maximize the sum of the scores of $[i, k_0]$ and $[k_0+1, j]$. The attention score of $[i, k_0]$ is computed at time step k_0, and hence does not know anything about the other span on the right. Therefore we consider only the score of the right span.

[9] Not necessarily one-hot: multiple spans can end at the same token.

3 Experiments

We evaluate PaLM on language modeling. We experiment with the Penn Treebank corpus (PTB) and WikiText-2 (WT2). We follow the preprocessing of Mikolov et al. (2010) for PTB and Merity et al. (2018) for WT2. More implementation details are described in Appendix A. We compare two configurations of PaLM:

- PaLM-U builds on top of AWD-LSTM (Merity et al., 2018), a state-of-the-art of LSTM implementation for language modeling. The span attention is included *before* the last layer.[10]
- PaLM-S is the same model as PaLM-U, but uses phrase syntax annotation to provide additional supervision to the attention component (§2.2).[11]

We compare against the AWD-LSTM baseline. On PTB, we also compare to two models using structural information in language modeling: parsing-reading-predict networks (PRPN; Shen et al., 2018a) predicts syntactic distance as structural features for language modeling; ordered-neuron LSTM (ON-LSTM; Shen et al., 2018b) posits a novel ordering on LSTM gates, simulating the covering of phrases at different levels in a constituency parse. On PTB we also compare to PaLM-RB, a baseline deterministically setting the attention scores (Eq. 3b) in decreasing order, such that the derived trees will be right-branching.[12]

Tables 1 and 2 summarize the language modeling results. On both datasets, the unsupervised configuration (PaLM-U) outperforms AWD-LSTM. On PTB, PaLM-U achieves similar performance to ON-LSTM and much better performance than PRPN. PaLM-S further reduces the perplexity by 1.6–3.4% (relative), showing that incorporating structural information with supervised span attention helps language modeling. Naively promoting right-branching attention (PaLM-RB) yields no improvement over the baseline.

Unsupervised constituency parsing. We evaluate the parser component of PaLM-U on WSJ-40. It uses the same data as in language modeling, but filters out sentences longer than 40 tokens after punctuation removal. The model is selected based on language modeling validation perplexity.

Model	# Params.	Dev.	Test
AWD-LSTM	24M	60.0	57.3
PRPN	-	-	62.0
ON-LSTM	25M	58.3	56.2
PaLM-U	24M	58.6	56.4
PaLM-RB	24M	60.1	57.5
PaLM-S	24M	**57.9**	**55.5**

Table 1: PTB language modeling perplexity (lower is better). Bold fonts indicate best performance.[13]

Model	# Params.	Dev.	Test
AWD-LSTM	33M	68.6	65.8
PaLM-U	36M	68.4	65.4
PaLM-S	36M	**65.5**	**63.2**

Table 2: WikiText-2 language modeling perplexity (lower is better). Bold fonts indicate best performance.

In addition to PRPN, we compare to DIORA (Drozdov et al., 2019), which uses an inside-outside dynamic program in an autoencoder. Table 3 shows the F_1 results. PaLM outperforms the right branching baseline, but is not as accurate as the other models.[14] This indicates that the type of syntactic trees learned by it, albeit useful to the LM component, do not correspond well to PTB-like syntactic trees.

Discussion. Despite its strong performance, the parsing algorithm used by Shen et al. (2018a) and Shen et al. (2018b) suffers from an incomplete support issue (Dyer et al., 2019). More precisely, it fails to produce "close-open-open," i.e.,) ((structures. As a result, the parser is intrinsically biased toward right-branching structures. PaLM, on the other hand, scores all the spans, and therefore can produce any binary tree spanning a given sentence: the algorithm recovers any given binary tree by letting $s_{j,j-i} = 1$ if the tree contains non-terminal $[i, j]$, and 0 otherwise.[15]

Is PaLM empirically biased toward any branching direction? In greedily selected trees, we measure the percentage of left-branching splits (dividing $[i, j]$ into $[i, j-1]$ and j) and right-branching

[10] Preliminary experiments show that including the span attention after the last layer yields similar empirical results, but is more sensitive to hyperparameters.

[11] We use the WSJ portion of PTB for parsing annotations.

[12] We set scores to $m, m-1, \ldots, 1$, before the softmax.

[13] Several recent works report better language modeling perplexity (Yang et al., 2019; Takase et al., 2018; Dai et al., 2019, *inter alia*). Their contribution is orthogonal to ours and not head-to-head comparable to the models in the table.

[14] Evaluation on WSJ-10, which contains sentences with 10 or less tokens, shows a similar trend.

[15] The maximum span length m is only forced in language modeling training and evaluation.

Model	Unlabeled F_1
Right Branching	40.7
†DIORA	60.6
‡PRPN	52.4
‡PaLM-U	42.0

Table 3: Unlabeled unsupervised parsing F_1 on WSJ-40. ‡ trains on the training split of WSJ, while † trains on AllNLI (Htut et al., 2018). The PRPN result is taken from Drozdov et al. (2019).

	% Left Splits		% Right Splits	
Random	39.3	±10.5	41.2	±8.8
PaLM-U	1.1		85.6	
Gold	6.5		52.7	

Table 4: Percentage of left and right splits. The first row shows the numbers averaging over 25 differently randomly initialized PaLM models, without training. ± indicates standard deviation.

splits (dividing $[i, j]$ into i and $[i+1, j]$).[16] Table 4 summarizes the results on WSJ-40 test set. The first row shows the results for randomly initialized models without training. We observe no significant trend of favoring one branching direction over the other. However, after training with the language modeling objective, PaLM-U shows a clear right-skewness more than it should: it produces much more right-branching structures than the gold annotation. This means that the span attention mechanism has learned to emphasize longer prefixes, rather than make strong Markov assumptions. More exploration of this effect is left to future work.

4 Conclusion

We present PaLM, a hybrid parser and language model. PaLM attends over the preceding text spans. From its attention weights phrase structures can be derived. The attention component can be separately trained to provide syntactically-informed context gathering. PaLM outperforms strong baselines on language modeling. Incorporating syntactic supervision during training leads to further language modeling improvements. Training our unsupervised model on large-scale corpora could result in both stronger language models and, potentially, stronger parsers. Our code is publicly available at https://github.com/Noahs-ARK/PaLM.

[16] We exclude trivial splits dividing a length-2 span into two tokens.

Acknowledgments

We thank members of the ARK at the University of Washington, and researchers at the Allen Institute for Artificial Intelligence for their helpful comments on an earlier version of this work, and the anonymous reviewers for their insightful feedback. This work was supported in part by NSF grant 1562364.

References

Jan Buys and Phil Blunsom. 2018. Neural syntactic generative models with exact marginalization. In *Proc. of NAACL*.

Ciprian Chelba and Frederick Jelinek. 1998. Exploiting syntactic structure for language modeling. In *Proc. of COLING*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. of EMNLP*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proc. of ACL*.

Dipanjan Das, André F. T. Martins, and Noah A. Smith. 2012. An exact dual decomposition algorithm for shallow semantic parsing with constraints. In *Proc. of *SEM*.

Andrew Drozdov, Pat Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive autoencoders. In *Proc. of NAACL*.

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

Chris Dyer, Gbor Melis, and Phil Blunsom. 2019. A critical analysis of biased parsers in unsupervised parsing. arXiv:1909.09428.

Daniel Gildea and Martha Palmer. 2002. The necessity of parsing for predicate argument recognition. In *Proc. of ACL*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. *Proc. of CVPR*.

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

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proc. of ACL*.

Phu Mon Htut, Kyunghyun Cho, and Samuel Bowman. 2018. Grammar induction with neural language models: An unusual replication. In *Proc. of EMNLP*.

Frederick Jelinek and John D. Lafferty. 1991. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–353.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. arXiv:1602.02410.

Yoon Kim, Alexander M. Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019. Unsupervised recurrent neural network grammars. In *Proc. of NAACL*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv:1412.6980.

Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Segmental recurrent neural networks. In *Proc. of ICLR*.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end neural coreference resolution. In *Proc. of EMNLP*.

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

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and optimizing LSTM language models. In *Proc. of ICLR*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *Proc. of ICLR*.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proc. of INTERSPEECH*.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proc. of EMNLP*.

Hao Peng, Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018a. Rational recurrences. In *Proc. of EMNLP*.

Hao Peng, Sam Thomson, and Noah A. Smith. 2018b. Backpropagating through structured argmax using a spigot. In *Proc. of ACL*.

Hao Peng, Sam Thomson, Swabha Swayamdipta, and Noah A. Smith. 2018c. Learning joint semantic parsers from disjoint data. In *Proc. of NAACL*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

B. T. Polyak and A. B. Juditsky. 1992. Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim.*, 30(4):838–855.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih. 2008. The importance of syntactic parsing and inference in semantic role labeling. *American Journal of Computational Linguistics*, 34(2):257–287.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. OpenAI Blog.

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

Yikang Shen, Zhouhan Lin, Chin wei Huang, and Aaron Courville. 2018a. Neural language modeling by jointly learning syntax and lexicon. In *Proc. of ICLR*.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron C. Courville. 2018b. Ordered neurons: Integrating tree structures into recurrent neural networks. In *Proc. of ICLR*.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proc. of ACL*.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proc. of EMNLP*.

Swabha Swayamdipta, Sam Thomson, Kenton Lee, Luke Zettlemoyer, Chris Dyer, and Noah A. Smith. 2018. Syntactic scaffolds for semantic structures. In *Proc. of EMNLP*.

Sho Takase, Jun Suzuki, and Masaaki Nagata. 2018. Direct output connection for a high-rank language model. In *Proc. of EMNLP*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. of NeurIPS*.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. 2019. Breaking the softmax bottleneck: A high-rank RNN language model. In *Proc. of ICLR*.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. SWAG: A large-scale adversarial dataset for grounded commonsense inference. In *Proc. of EMNLP*.

Appendices

A Implementation Details

Neural Network Architecture Our implementation is based on AWD-LSTM (Merity et al., 2018).[17] It uses a three-layer LSTM, with carefully designed regularization techniques. PaLM includes the span attention after the second layer. Preliminary results show that it yields similar results, but is less sensitive to hyperparameters, compared to adding it to the last layer.

The context is concatenated to the hidden state ($\bar{\mathbf{h}}_t = [\mathbf{h}_t; \mathbf{a}_t]$), and then fed to a $\tanh$-MLP controlled by a residual gate $\mathbf{g}_r$ (He et al., 2016), before fed onward into the next LSTM layer:

$$\hat{\mathbf{h}}_t = \mathbf{g}_r \odot \text{MLP}\left(\bar{\mathbf{h}}_t\right) + (\mathbf{1} - \mathbf{g}_r) \odot \mathbf{h}_t. \quad (6)$$

The rest of the architecture stays the same as AWD-LSTM. We refer the readers to Merity et al. (2018) for more details.

More details on PaLM-S. PaLM-S uses exactly the same architecture and hyperparameters as its unsupervised counterpart. We derive, from PTB training data, a m-dimensional 0-1 vector for each token. Each element specifies whether the corresponding span appears in the gold parse. Trivial spans (i.e., the ones over single tokens and full sentences) are ignored. The vector are normalized to sum to one, in order to facilitate the use of cross-entropy loss. λ in Eq. 5 is set to 0.01.

Hyperparameters. The regularization and hyperparameters largely follow Merity et al. (2018). We only differ from them by using smaller hidden size (and hence smaller dropout rate) to control for the amount of parameters in the PTB experiments, summarized in Table 5 For the WikiText-2 experiments, we use 200 rational RNN size and 400 dimensional context vectors. Other hyperparameters follow Merity et al. (2018). The max span length m is set to 20 for PTB experiments, and 10 for WikiText-2.

Merity et al. (2018) start by using SGD to train the model, and switch to averaged SGD (Polyak and Juditsky, 1992) after 5 nonimprovement-epochs. We instead use Adam (Kingma and Ba, 2014) with default PyTorch settings to train the model for 40 epochs, and then switch to ASGD, allowing for faster convergence.

[17] https://github.com/salesforce/awd-lstm-lm

Type	Values
Rational RNN size	200
Context Vector Size	400
LSTM Hidden Size	1020
Weight Dropout	0.45
Vertical Dropout	0.2

Table 5: The hyperparameters used in the PTB language modeling experiment.

B Span Representations

Below is the derivation for Eq. 2.

$$\begin{aligned}
\overrightarrow{\mathbf{c}}_{i,j} &= \overrightarrow{\mathbf{u}}_j + \sum_{k=i}^{j-1} \overrightarrow{\mathbf{u}}_k \bigodot_{\ell=k+1}^{j} \overrightarrow{\mathbf{f}}_\ell \\
&= \overrightarrow{\mathbf{u}}_j + \sum_{k=1}^{j-1} \overrightarrow{\mathbf{u}}_k \bigodot_{\ell=k+1}^{j} \overrightarrow{\mathbf{f}}_\ell - \sum_{k=1}^{i-1} \overrightarrow{\mathbf{u}}_k \bigodot_{\ell=k+1}^{j} \overrightarrow{\mathbf{f}}_\ell \\
&= \overrightarrow{\mathbf{c}}_j - \left(\overrightarrow{\mathbf{u}}_{i-1} + \sum_{k=1}^{i-2} \overrightarrow{\mathbf{u}}_k \bigodot_{\ell=k+1}^{i-1} \overrightarrow{\mathbf{f}}_\ell \right) \bigodot_{\ell=i}^{j} \overrightarrow{\mathbf{f}}_\ell \\
&= \overrightarrow{\mathbf{c}}_j - \overrightarrow{\mathbf{c}}_{i-1} \bigodot_{k=i}^{j} \overrightarrow{\mathbf{f}}_k
\end{aligned}$$