Language Modeling with Shared Grammar

Yuyu Zhang
College of Computing
Georgia Institute of Technology
yuyu@gatech.edu

Le Song
College of Computing
Georgia Institute of Technology
lsong@cc.gatech.edu

Abstract

Sequential recurrent neural networks have achieved superior performance on language modeling, but overlook the structure information in natural language. Recent works on structure-aware models have shown promising results on language modeling. However, how to incorporate structure knowledge on corpus without syntactic annotations remains an open problem. In this work, we propose neural variational language model (NVLM), which enables the sharing of grammar knowledge among different corpora. Experimental results demonstrate the effectiveness of our framework on two popular benchmark datasets. With the help of shared grammar, our language model converges significantly faster to a lower perplexity on new training corpus.

1 Introduction

Language modeling has been a long-standing fundamental task in natural language processing. In recent years, sequential recurrent neural networks (RNNs) based language models have made astonishing progress, which achieve remarkable results on various benchmark datasets (Mikolov et al., 2010a; Jozefowicz et al., 2016; Melis et al., 2017; Elbayad et al., 2018; Gong et al., 2018; Takase et al., 2018; Dai et al., 2019). Despite the huge success, the structure information in natural language is largely overlooked due to the structural limit of sequential RNN-based language models.

Recently, researchers have explored to explicitly exploit the latent structures in natural language, such as recurrent neural network grammars (RNNGs; Dyer et al., 2016; Kuncoro et al., 2017) and parsing-reading-predict networks (PRPNs; Shen et al., 2017). These structure-aware models have shown promising results on language modeling, demonstrating that the latent nested structure in language indeed helps improve sequential language models. Models like RNNG exploit treebank data with syntactic annotations to learn grammar, which is then used to improve language model performance by a significant margin. This is definitely intriguing, but we have to pay the cost: accurate syntactic annotation is very costly, and treebank data such as the Penn Treebank (Marcus et al., 1993) is typically small-scale and not open to the public for free.

On new corpus which has no syntactic annotations, how to improve language modeling with grammar knowledge? This is an important and challenging open problem. As a motivating example, we conduct a simple experiment by training a RNN language model on one corpus and testing it on another, and report the results in Table 1. The RNN language model performs terribly when training and testing on different datasets, which is reasonable since the data distribution may vary dramatically on different corpora. Training from scratch on every new corpus is obviously not good enough: 1) it is computationally expensive and not data-efficient; 2) the size of target corpus may be too small to train a decent RNN-based language model; 3) the common grammar is not leveraged. Some recent works on transfer learning have made attempts on language model adaptation (Yoon et al., 2017; Ma et al., 2017; Chen et al., 2015), however, none of them explicitly exploits the common grammar knowledge shared between corpora.

To bridge the gap of language modeling on different corpora, we believe that grammar is the key since all corpora are in the same language and should share the same grammar. Motivated by that, we propose neural variational language model (NVLM). Specifically, our framework consists of two probabilistic components: a constituency parser and a joint generative model of sentence and parse tree. When treebank data is

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 4442–4453
Florence, Italy, July 28 - August 2, 2019.

	Train on PTB	Train on OBWB
Test on PTB	112.3	**419.9**
Test on OBWB	**242.2**	139.3

Table 1: Test perplexity of RNN language model, which performs terribly when training and testing on different datasets.

available, we can separately train both components. On new corpus without tree annotations, we fix the pre-trained parser and train the generative model either from scratch or with warm-up. The pre-trained parser is armed with grammar knowledge, thus it boosts up our language model to land on new corpus. Our proposed framework also supports end-to-end joint training of the two components, so that we can fine-tune the language model. Experimental results show that our proposed framework is effective in all leaning schemes, which achieves good performance on two popular benchmark datasets. With the help of shared grammar, our language model converges significantly faster to a lower perplexity on new corpus.

Our contributions in this paper are summarized as follows:

- *Grammar-sharing framework*: We propose a framework for grammar-sharing language modeling, which incorporates the common grammar knowledge into language modeling. With the shared grammar, our framework helps language model efficiently transfer to new corpus with better performance and using shorter time.
- *End-to-end learning*: Our framework can be end-to-end trained without syntactic annotations. To tackle the technical challenges in end-to-end learning, we use variational methods that exploit policy gradient algorithm for joint training.
- *Efficient software package*: We provide a highly efficient implementation of our work on GPUs. Our parser is capable of parsing one million sentences per hour on a single GPU. See Appendix D for details.

2 Model

In this section, we first provide an overview of the proposed framework, then briefly introduce how components work together, and finally present the probabilistic formulation of each component.

2.1 Framework

As shown in Figure 1, neural variational language model (NVLM) consists of two probabilistic components: 1) a constituency parser $P_{\theta_1}(y|x)$ which models the conditional probability of the parse tree y (a syntax tree without terminal tokens) given the input sentence x (a sequence of terminal tokens); 2) a joint generative model $P_{\theta_2}(x, y)$ which models the joint probability of the sentence and the parse tree.

Constituency Parsing. Our parser can work independently, which takes as input a sentence x and parses x according to

$$\operatorname*{argmax}_{y' \in \mathcal{Y}(x)} P_{\theta_1}(y'|x), \tag{1}$$

where $\mathcal{Y}(x)$ denotes the collection of all possible parses of x. Our parser can also cooperate with the joint generative model as

$$\operatorname*{argmax}_{y' \sim P_{\theta_1}(y|x)} P_{\theta_2}(x, y'), \tag{2}$$

where the parsing candidates y' sampled from $P_{\theta_1}(y|x)$ are fed into the generative model $P_{\theta_2}(x, y)$ to be reranked.

Language Modeling. Statistical language models are typically formulated as

$$\begin{aligned} P(x) &= P(x_1, x_2, \ldots, x_{L_x}) \\ &= \prod_{t=1}^{L_x} P(x_t|x_{<t}), \end{aligned} \tag{3}$$

where x_t denotes the t-th token in the sentence x, the length of x is denoted as L_x, and $x_{<t}$ indicates all tokens before x_t. To evaluate NVLM as a language model, we need to marginalize the joint probability as $P(x) = \sum_{y' \in \mathcal{Y}(x)} P(x, y')$. This is extremely hard to compute due to the exponentially large space of $\mathcal{Y}(x)$. We use importance sampling technique to overcome this computational intractability, which is detailed in Section 4.

With treebank data such as Penn Treebank (Marcus et al., 1993), we have pairs of (x, y) to train the two components respectively, and get high-quality language model with the parser providing grammar knowledge. However, due to the expensive cost of accurate parsing annotation,

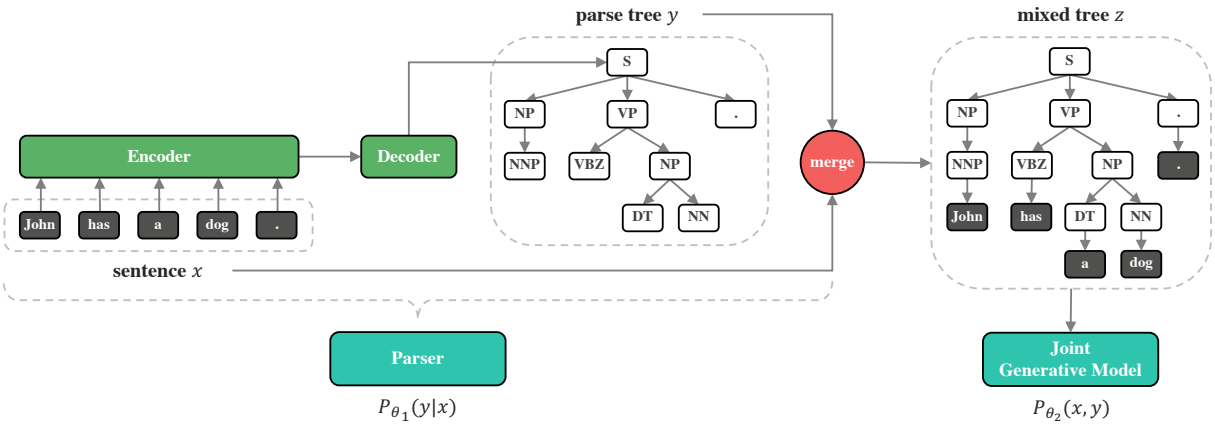


Figure 1: Overall framework of neural variational language model (NVLM). It consists of two probabilistic components: a constituency parser $P_{\theta_1}(y|x)$ and a joint generative model $P_{\theta_2}(x, y)$. The parser takes as input a sentence x and predicts the corresponding parse tree y. Specifically, we use an encoder-decoder structure to parameterize the parser. The joint generative model defines a joint distribution on parse trees (y) and sentences (x). When treebank data is available, we can learn the parameters θ_1 and θ_2 for each component respectively. To train language model on new corpus, we fix the pre-trained θ_1 and only update θ_2. Our framework can also be end-to-end jointly trained to fine-tune the language model, where θ_1 and θ_2 are co-updated together.

treebank data is typically scarce. For new corpus without parsing annotations, our proposed framework can still leverage the parser to train high-quality language model adapted to the new corpus. Also, we can co-train the two components together to fine-tune the language model on new corpus.

In the rest of this section, we present our parameterization of the two probabilistic components $P_{\theta_1}(y|x)$ and $P_{\theta_2}(x, y)$. To avoid notational clutter, we use standard RNN as the basic building block in the rest of this section.[1]

2.2 Constituency Parser

To parameterize the constituency parser $P_{\theta_1}(y|x)$, it is natural to first encode the input sentence x into an embedding vector, then pass the vector to a decoder to generate the parse tree y. There are quite a few choices for both encoder and decoder, among which recurrent neural network (RNN) and convolutional neural network (CNN) are the most popular ones, since they are powerful to capture the structural patterns in natural language (Sutskever et al., 2014; Zhang et al., 2015). Vinyals et al. (2015) have found that the RNN-powered sequence-to-sequence (Seq2Seq) architecture with attention mechanism achieves state-of-the-art parsing performance. This architecture is conceptually simple yet powerful and of large model capacity.

In this paper, we adapt the sequence-to-sequence architecture for NVLM. We linearize a parse tree as a bracket representation (a sequence of nodes and brackets ordered by a pre-order traversal of the tree), which is a one-to-one mapping of the tree structure. For example, the parse tree shown in Figure 1 can be linearized as `(S (NP NNP ) (VP VBZ (NP DT NN ) ) . )`. Interestingly, the parser is now similar to a neural translation model (Bahdanau et al., 2014), which translates a sentence into a linearized parse tree. Next we show how the parser computes $P_{\theta_1}(y|x)$ in detail.

Formally, the input sentence x is fed into the encoder, and is encoded as a sequence of hidden states $H = \{h_1, h_2, \ldots, h_{L_x}\}$, where L_x is the length of x, and $h_i = \text{RNN}_{\text{enc}}(x_i, h_{i-1})$ where x_i is first embedded into a vector (word embedding) and then fed into the recurrent unit, and h_0 is a learnable vector for the special start-of-sentence token `<SOS>`. The decoder uses a separate RNN to calculate the hidden states $s_j = \text{RNN}_{\text{dec}}([y_{j-1}; c_j], s_{j-1})$, where y_0 is set as `<SOS>`, s_0 is set as h_{L_x} (the last hidden state of the encoder), and y_{j-1} is the decoder's previous output token sampled from the categorical distribution of the decoder's softmax layer (or specified in teacher-forcing training mode), embedded and then concatenated with the context vector c_j

[1] The proposed neural variational language model is independent of any specific implementation of the recurrent unit such as LSTM (Hochreiter and Schmidhuber, 1997), GRU (Cho et al., 2014) and SRU (Lei and Zhang, 2017), and can be directly applied to their deep and bi-directional variants.

to serve as the RNN input. The context c_j is calculated as $c_j = \sum_{i=1}^{L_x} \alpha_{i,j} h_i$, where $\alpha_{i,j} = \frac{\exp\left(h_i^\top s_{j-1}\right)}{\sum_{i'=1}^{L_x} \exp\left(h_{i'}^\top s_{j-1}\right)}$. This is a simplified version of the conventional attention mechanism (Bahdanau et al., 2014) used in the Seq2Seq parser (Vinyals et al., 2015). Our parser is designed to be lighter and faster so that it can efficiently work together with the NVLM joint generative model.

Finally, the likelihood $P_{\theta_1}(y|x)$ is computed as

$$P_{\theta_1}(y|x) = \prod_{t=1}^{L_y} P(y_t|x, y_{<t}) = \prod_{t=1}^{L_y} \left[\text{softmax}\Big(f\big([s_t; c_t]\big)\Big)\right]_{y_t}, \quad (4)$$

where $f(\cdot)$ refers to a fully-connected layer with $\tanh$ activation, and the subscript y_t is to select its probability in the categorical distribution. L_y denotes the length of y, which is determined by the decoder itself. Once the decoder emits the special end-of-sentence token <EOS>, the decoding phase is terminated.

The trainable parameters θ_1 in the parser component $P_{\theta_1}(y|x)$ include the weights (and biases) in RNN_{enc} and RNN_{dec}, and all word embeddings. We use separate weights and word embeddings for the encoder and the decoder.

2.3 Joint Generative Model

Similar to the parser, the joint generative model $P_{\theta_2}(x, y)$ can also be parameterized in various ways. For example, Choe and Charniak (2016) uses a LSTM language model trained on the parser output (with terminal words), which is then used to rerank an existing parser and achieves state-of-the-art parsing performance. Inspired by that, we parameterize the joint generative model as

$$P_{\theta_2}(x, y) = P(z) = \prod_{t=1}^{L_z} P(z_t|z_{<t}), \quad (5)$$

where z is the mixed parse tree of x and y, which is then mapped to a sequential representation following a pre-order traversal. Figure 1 illustrates how a sentence x and its parse tree y can be merged into a mixed tree. We use another RNN_{gen} to compute the likelihood $P(z_t|z_{<t})$, and finally get $P_{\theta_2}(x, y)$ using Eq. (5).

Algorithm 1: Sentence word attaching

input : sentence x; parser output tokens y
output: mixed tree tokens z

```
if y is not balanced then
    y ← BalanceTree(y)
z ← Ø
j ← 1
L_x ← Length of x
L_y ← Length of y
for i ← 1, ..., L_y do
    if y_i ∈ LeafNode(y) ∧ j <= L_x then
        z ← z ∪ {"(" + y_i}
        z ← z ∪ {x_j}
        z ← z ∪ {")" + y_i}
        j ← j + 1
    else
        z ← z ∪ {y_i}
```

The trainable parameters θ_2 in the joint generative model $P_{\theta_2}(x, y)$ include the weights (and biases) in RNN_{gen} and all word embeddings.

In Choe and Charniak (2016), the parser is fixed and well-trained before training the generative model. Unlike that, our parser can be jointly trained with the generative model, where the parser may not be fully trained yet. Therefore, the generated parse tree can be malformed, which mismatches the sentence with incorrect number of leaves. An even worse case is when the parser's output is not balanced and not able to form a legitimate tree. To handle these cases, we propose a sentence word attaching algorithm, which guarantees to generate a well-formed mixed tree. We describe our algorithm for mixed tree generation in Algorithm 1. This algorithm takes as input a sentence and its parse tree, and generates a mixed tree by attaching the sentence words to the leaf nodes of the parse tree. To handle unbalanced parse tree, which is rare case but happens due to the nature of sequential parser, we simply add brackets to either the head or tail to make the parse tree balanced.

3 Learning

In this section, we describe our algorithms for learning the model parameters in the constituency parser $P_{\theta_1}(y|x)$ and the joint generative model $P_{\theta_2}(x, y)$.

3.1 Learning Schemes

NVLM can be trained in three different schemes: 1) *fully supervised learning*, where sentences (x) and their corresponding parse trees (y) are available; 2) *distant-supervised learning*, where we have a pre-trained parser and a new corpus without parsing annotations; 3) *semi-supervised learning*, where we have no parsing annotations available.

Let $\mathcal{D}_{XY} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ denote the annotated training data, where each sentence is paired with a parse tree. Let $\mathcal{D}_X = \{x^{(i)}\}_{i=1}^{m}$ denote the unannotated training data, where only sentences are available. Next, we show how to train NVLM under each setting respectively.

Supervised: In the fully supervised setting, we use $\mathcal{D}_{XY}$ to separately train the parser and the generative model, by maximizing their respective data log likelihood

$$\mathcal{J}_{\theta_1}(\mathcal{D}_{XY}) = \frac{1}{n}\sum_{i=1}^{n} \log P_{\theta_1}(y^{(i)}|x^{(i)}), \quad (6)$$

$$\mathcal{J}_{\theta_2}(\mathcal{D}_{XY}) = \frac{1}{n}\sum_{i=1}^{n} \log P_{\theta_2}(x^{(i)}, y^{(i)}), \quad (7)$$

where $P_{\theta_1}(\cdot)$ and $P_{\theta_2}(\cdot)$ are defined in Eq. (4) and Eq. (5). We obtain the gradients $\nabla_{\theta_1}\mathcal{J}_1$ and $\nabla_{\theta_2}\mathcal{J}_2$ by chain rule, and iteratively update θ_1 and θ_2 with standard optimizers.

Distant-supervised: In distant-supervised learning, we have pre-trained the parser on corpus $\mathcal{D}_{XY}$, and fix the parser to train the joint generative model on new corpus $\mathcal{D}_X$. The generative model can be either trained from scratch on $\mathcal{D}_X$ or warmed-up on $\mathcal{D}_{XY}$. This setting is of practical importance, since we often need a language model on new corpus without annotations, and the parser pre-trained on treebank data can help since it encodes common grammar knowledge of the language.

Under this setting, the pre-trained parser generates parse trees using Eq. (4) for unannotated sentences, and form (x, y) pairs to train the joint generative model through $\nabla_{\theta_2}\mathcal{J}_2$. The parser's parameters θ_1 remain fixed.

Semi-supervised: NVLM can be end-to-end trained with only unannotated data $\mathcal{D}_X$. This is extremely hard if we train everything from scratch. However, it is very useful to fine-tune the language model on new corpus. Unlike distant-supervised learning, we now train the parser and the joint generative model together, and co-update the parameters θ_1 and θ_2. Here we also maximize the data log likelihood

$$\mathcal{J}(\mathcal{D}_X) = \frac{1}{m}\sum_{i=1}^{m} \log \left(\sum_{y \in \mathcal{Y}(x^{(i)})} P_{\theta_2}(x^{(i)}, y) \right). \quad (8)$$

Unfortunately, the derivative of $\mathcal{J}(\mathcal{D}_X)$ is computationally intractable due to the large space of $\mathcal{Y}$. To tackle this challenge, we use variational methods to maximize the lower bound of $\mathcal{J}(\mathcal{D}_X)$, and exploit policy gradient algorithm to update the parser's parameters. Details are described in Section 3.2 and Section 3.3. Our algorithm for semi-supervised learning is summarized in Algorithm 2, where we assume mini-batch size as 1 to avoid notational clutter.

Algorithm 2: Semi-supervised learning

input : annotated training data $\mathcal{D}_{XY}$;
unannotated training data $\mathcal{D}_X$;
optimizer $\mathcal{G}(\cdot)$

Initialize θ_1 with $\mathcal{D}_{XY}$ using Eq. (6)
Initialize θ_2 with $\mathcal{D}_{XY}$ using Eq. (7)
while *model not converged* **do**
  Sample a sentence $x^{(i)}$ from $\mathcal{D}_X$
  Sample a parse tree $y^{(i)}$ from $P_{\theta_1}(y|x^{(i)})$
  Standardize the signal $A(x^{(i)}, y^{(i)})$
  Update the baseline function with $b(x^{(i)})$
  $\theta_1 \leftarrow \theta_1 + \mathcal{G}\big(\nabla_{\theta_1}\tilde{\mathcal{J}}(\mathcal{D}_X)\big)$ using Eq. (12)
  $\theta_2 \leftarrow \theta_2 + \mathcal{G}\big(\nabla_{\theta_2}\tilde{\mathcal{J}}(\mathcal{D}_X)\big)$ using Eq. (10)

3.2 Variational EM

As described above, to overcome the computational intractability of maximizing $\mathcal{J}(\mathcal{D}_X)$ directly, we use variational expectation-maximization (EM) algorithm to maximize the evidence lower bound (ELBO):

$$\tilde{\mathcal{J}}(\mathcal{D}_X) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}_{P_{\theta_1}(y|x^{(i)})}\Big[\log P_{\theta_2}(x^{(i)}, y) - \log P_{\theta_1}(y|x^{(i)}) \Big], \quad (9)$$

where we use our parser as the variational posterior $P_{\theta_1}(y|x)$. For readability, from now on we assume $m = 1$ and omit summing over training samples. With the Monte Carlo method, we ob-

tain the unbiased gradient

$$\nabla_{\theta_2}\tilde{\mathcal{J}}(\mathcal{D}_X) = \mathbb{E}_{P_{\theta_1}(y|x)}\Big[\nabla_{\theta_2} P_{\theta_2}(x,y)\Big]. \quad (10)$$

3.3 Policy Gradient

To get the gradient of $\tilde{\mathcal{J}}(\mathcal{D}_X)$ w.r.t. the parser parameters θ_1, we need more work since y is sampled from a series of categorical distributions. Here we use policy gradient algorithm (Williams, 1992) to get an unbiased estimator of the gradient

$$\nabla_{\theta_1}\tilde{\mathcal{J}}(\mathcal{D}_X) = \mathbb{E}_{P_{\theta_1}(y|x)}\Big[\nabla_{\theta_1} P_{\theta_1}(y|x) A(x,y)\Big], \quad (11)$$

where $A(x,y) = \log P_{\theta_2}(x,y) - \log P_{\theta_1}(y|x)$ is used as the learning signal. Due to the limit of space, we provide detailed derivation of Eq. (11) in Appendix E. In order to stabilize the learning process, we use standard variance reduction techniques to reduce the variance of gradient (Greensmith et al., 2004; Mnih and Gregor, 2014; Zhang et al., 2017). Specifically, we first standardize the signal (rescaling it to zero mean and unit variance) and then subtract a baseline function $b(x)$. Then we use a separate GRU as the baseline function and fit the centered signal by minimizing the mean square loss. Finally, the gradient can be approximated as

$$\nabla_{\theta_1}\tilde{\mathcal{J}}(\mathcal{D}_X) \approx \mathbb{E}_{P_{\theta_1}(y|x)}\Bigg[\nabla_{\theta_1} P_{\theta_1}(y|x) \left(\frac{A(x,y)-\tilde{\mu}}{\tilde{\sigma}} - b(x)\right)\Bigg], \quad (12)$$

where $\tilde{\mu}$ is the sample mean and $\tilde{\sigma}$ is the sample standard deviation, which estimate the mean and standard deviation of the learning signal $A(x,y)$.

4 Inference

With the two components in NVLM, a parser $P_{\theta_1}(y|x)$ and a joint generative model $P_{\theta_2}(x,y)$, we can do three types of inference:

- Parsing, where we sample the parser with greedy decoding to generate the parse tree for input sentence;
- Evaluating $P_{\theta_2}(x,y)$, which is obtained from the joint generative model, and can be used to help rerank parsing candidates;
- Estimating $P(x) = \sum_{y' \in \mathcal{Y}(x)} P(x,y')$ and evaluating the model perplexity, which is intractable due to the exponentially large space of $\mathcal{Y}(x)$. Similar to Dyer et al. (2016), we use importance sampling technique to estimate $P(x)$.

Specifically, we use our parser $P_{\theta_1}(y|x)$ as the proposal distribution. The estimator of $P(x)$ is derived as

$$\begin{aligned} P(x) &= \sum_{y' \in \mathcal{Y}(x)} P_{\theta_1}(y'|x)\frac{P_{\theta_2}(x,y')}{P_{\theta_1}(y'|x)} \\ &= \mathbb{E}_{P_{\theta_1}(y'|x)}\frac{P_{\theta_2}(x,y')}{P_{\theta_1}(y'|x)}. \end{aligned} \quad (13)$$

5 Experiments

5.1 Settings

Datasets. We conduct experiments on two popular datasets for language modeling: Penn Treebank (PTB; Marcus et al., 1993) and One Billion Word Benchmark (OBWB; Chelba et al., 2013).[2] The PTB dataset has parsing annotations, while OBWB dataset has no annotations. For the PTB dataset, we adopt the standard train / validation / test split. We build the vocabulary based on PTB, using one unknown token for singleton words. For the OBWB dataset, we use the original train/test split, and subsample each to have similar size of PTB (50K sentences for training, and 2.5K sentences for test). The subsampling is for the efficiency of evaluating perplexity using importance sampling, which requires to sample multiple (we use 100) parse trees for each sentence. The training of our framework is actually much more scalable than the perplexity evaluation, and not restricted to the size of downsampled dataset. Note that our data preprocessing scheme follows what is standard in parsing instead of language modeling, since parsing typically requires more information (such as capital letters) and larger vocabulary. The vocabulary size for text is 26,620, and we have a separate vocabulary of size 74 for nonterminal nodes of parsing, such as `(NP` and `NNP`. Refer to Appendix A for more data preprocessing details.

Tasks. We work on three different tasks: 1) *supervised learning*: we separately train both the parser

[2]The treebank data is publicly available through the Linguistic Data Consortium (LDC): Penn Treebank (LDC99T42). The original OBWB dataset is downloaded from http://www.statmt.org/lm-benchmark/.

and the joint generative model on PTB; 2) *distant-supervised learning*: we pre-train the parser on PTB, and then fix the parser to train the joint generative model on OBWB either from scratch or with PTB warmed-up model; 3) *semi-supervised learning*: we jointly train the parser and the generative model together on OBWB to fine-tune the language model.

Evaluation. We mainly focus on language modeling, and use per-word perplexity to evaluate our framework and competitor models. As for the parser, we compare with state-of-the-art parsers in terms of training and testing speed.

Baselines. On the PTB dataset, we compare our language model with the following baselines: 1) Kneser-Ney 5-gram language model; 2) LSTM language model; 3) GRU language model implemented by ourselves; 4) recent state-of-the-art language models that also incorporate grammar to improve language modeling, including RNNG, SO-RNNG and GA-RNNG (Dyer et al., 2016; Kuncoro et al., 2017). On OBWB dataset, since there is no parsing annotations available, we compare with GRU language model as a strong baseline.

Optimization. All our models are trained on a single NVIDIA GTX 1080 GPU. For all NVLM models, we use Adam optimizer (Kingma and Ba, 2014) for the parser, and standard SGD for the joint generative model. Gradients are clipped at 0.25. See Appendix B for more details.

5.2 Supervised Learning

We first experiment on PTB dataset in the supervised learning setting. We separately train our parser and joint generative model on PTB training dataset, and then evaluate our language model on PTB test dataset. Table 2 lists the performance of our framework and competitor models. GRU-256 LM is our implemented language model using 2-layer GRU with hidden size 256, which is also used in other experiments. Parsing annotations are used by RNNG, SO-RNNG, GA-RNNG and NVLM. These grammar-aware models achieve significantly better performance compared to state-of-the-art sequential RNN-based language models, showing that grammar indeed helps language modeling. NVLM substantially improves over the current state of the art, by 10% reduction on test perplexity.

With respect to our parser, instead of pursuing state-of-the-art parsing performance, it is designed to be light and fast to efficiently work together with the NVLM joint generative model. Our parser achieves 90.7 F_1 accuracy on PTB test dataset, which is comparable to state-of-the-art parsers. Due to the page limit, we report the detailed parsing performance in Appendix C.

Model	Perplexity
KN-5-gram (Kneser and Ney, 1995)	169.3
LSTM-128 LM (Zaremba et al., 2014)	113.4
GRU-256 LM	112.3
RNNG (Dyer et al., 2016)	102.4
SO-RNNG (Kuncoro et al., 2017)	101.2
GA-RNNG (Kuncoro et al., 2017)	100.9
NVLM	**91.6**

Table 2: Test perplexity on PTB §23. KN-5-gram refers to Kneser-Ney 5-gram LM. Note that, since parsing typically requires more information (e.g., capital letters), we follow the standard data preprocessing of syntax-aware language modeling as in Dyer et al. (2016), thus the vocabulary size (∼27K) is much larger than the capped vocabulary size (10K) in standard language modeling setting. Therefore, all the perplexity results reported in this paper are not directly comparable to that achieved by syntax-agnostic language models with a much smaller vocabulary, such as the perplexity 57.3 reported in Merity et al. (2017) and 54.5 reported in Dai et al. (2019). This also applies to the perplexity results on the OBWB dataset.

5.3 Distant-supervised Learning

We then experiment on learning language model on new corpus without tree annotations. This is to verify whether the learned parser can help language model softly land on new corpus. We use the subsampled OBWB dataset for model training and evaluation. GRU-256 LM is used as a strong baseline. We have two different settings for both GRU LM and our framework: 1) *from-scratch*: For GRU LM, we randomly initialize it before training on OBWB. For NVLM, we train the parser on PTB and fix it, and randomly initialize the joint generative model; 2) *warmed-up*: For GRU LM, we pre-train GRU LM on PTB before training it on the OBWB dataset. For NVLM, we train the parser on PTB and fix it, and pre-train the joint generative model on PTB as warm-up. In both cases, NVLM uses its parser (trained on PTB) to generate parse trees for the OBWB dataset, and train the joint generative model with these silver-

standard parse trees.

Figure 2(a) shows the test perplexity curves along with number of training epochs. In both from-scratch and warmed-up settings, NVLM performs significantly better than GRU LM by 22.7 and 7.8 points in perplexity reduction. The warmed-up NVLM converges fast and achieves the lowest perplexity. Even when trained from scratch, NVLM achieves better performance than the warmed-up GRU LM, though it takes longer to converge. Note that the warm-up of GRU LM is directly training $P(x)$ with more data, while the warm-up of NVLM is only for $P(x, y)$. This explains why GRU LM seems to benefit more from warm-up at beginning, and why NVLM from scratch takes longer to converge.

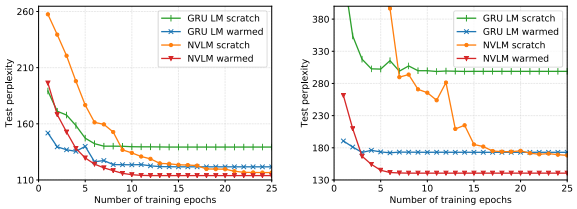
Unlike the supervised learning setting, NVLM can now be trained on new corpus without parsing annotations, and still leverages the common grammar knowledge. To further study the adaptation speed of NVLM on new corpus, we train NVLM with variant proportion of training data in both from-scratch and warmed-up settings. We also train GRU LM as a strong baseline. Results are reported in Table 3.

As shown in Table 3, with smaller amount of data, NVLM outperforms GRU LM even more significantly. We find that with only 20% of training data, the warmed-up NVLM achieves test perplexity 140.6, which is comparable to GRU LM trained with full data from scratch (139.3). This demonstrates that our framework is data-efficient, and can quickly adapt to new corpus without parsing annotations. Moreover, we notice that even without looking at the new corpus (0% training data), the warmed-up NVLM achieves reasonable perplexity (151.8) which is substantially lower than the warmed-up GRU LM (242.2). This agrees with our conjecture that grammar knowledge is sharable among different corpora. Figure 2(b) plots the test perplexity curves using 20% OBWB training data. The warmed-up NVLM quickly converges, and achieves much lower perplexity compared to the warmed-up GRU LM.

5.4 Semi-supervised Learning

In distant-supervised setting, the parser is fixed when training NVLM on new corpus. We can actually continue training the parser together with the generative model, so that the language model can be fine-tuned in end-to-end fashion. This is essentially semi-supervised learning: the parser has to be updated without parsing annotations, and the generative model will be updated together. Due to the exponentially large space of parse trees, such joint training is computationally intractable. To tackle the challenge, we use variational EM (Section 3.2) and exploit policy gradient (Section 3.3) algorithm to co-update both components of NVLM.

Technically, when the parser is fixed, the model is also maximizing the lower bound of data log likelihood. This empirically works well, as shown in the distant-supervised setting. With joint training, the model is essentially trying to find a better posterior on the new corpus and maximize a tighter lower bound. Therefore, the data log likelihood in Eq. (8) can be better optimized. In semi-supervised setting, we use full OBWB training data (subsampled). As reported in Table 4, NVLM achieves the lowest perplexity (110.2) with joint training. We also evaluate the parser on PTB after joint training. It couldn't get improved since it has



(a) 100% training data (b) 20% training data

Figure 2: Test perplexity curves on the subsampled OBWB dataset. Models are trained respectively on 100% and 20% training data. Models randomly initialized are marked as "scratch", while models pre-trained on the PTB dataset are marked as "warmed".

Training Data	GRU LM		NVLM	
	scratch	warmed	scratch	warmed
0%	>1000	242.2	>1000	151.8
20%	298.8	173.1	168.2	140.6
40%	210.3	147.3	143.3	138.5
60%	177.6	136.5	135.8	133.5
80%	152.5	130.5	128.6	125.6
100%	139.3	121.7	**116.6**	**113.9**

Table 3: Test perplexity on the subsampled OBWB dataset. Models are trained on variant proportion of training data. Models randomly initialized are marked as "scratch", while models pre-trained on the PTB dataset are marked as "warmed".

Model	Perplexity
GRU LM – from scratch	139.3
GRU LM – warmed-up on PTB	121.7
NVLM – from scratch	116.6
NVLM – warmed-up on PTB	113.9
NVLM – fine-tuned by joint training	**110.2**

Table 4: Test perplexity on the subsampled OBWB dataset. All models are trained with 100% training data.

been fine-tuned on new corpus, and we have no annotations to evaluate the parser on new corpus.

6 Related Work and Discussion

Due to the remarkable success of RNN-based language models (Mikolov et al., 2010b,a; Jozefowicz et al., 2016; Yang et al., 2017; Merity et al., 2017; Elbayad et al., 2018; Gong et al., 2018; Takase et al., 2018; Dai et al., 2019), not too much attention has been paid to incorporate syntactic knowledge into language model. Although RNN-based models achieve impressive results on language modeling and other NLP tasks such as machine translation (Cho et al., 2014; Bahdanau et al., 2014; Xia et al., 2016) and parsing (Vinyals et al., 2015; Dyer et al., 2015; Choe and Charniak, 2016), it is far from perfect since it overlooks the language structure and simply generates sentence from left to right. The words in natural language are largely organized in latent nested structures rather than simple sequential order (Chomsky, 2002).

Our work is related to syntactic language models, which has a long history. Traditional syntactic language models jointly generate syntactic structure with words using either bottom-up (Jelinek and Lafferty, 1991; Emami and Jelinek, 2004, 2005; Henderson, 2004), or top-down strategy (Charniak, 2000; Roark, 2001). Recently, some studies show the benefits of incorporating language structure into RNN-based language model, such as RNNG (Dyer et al., 2016; Kuncoro et al., 2017). Different from our work, these models mainly focus on parsing instead of language modeling, and cannot be trained without parsing annotations. Works on programming code generation (Rabinovich et al., 2017; Yin and Neubig, 2017) demonstrate that grammar is the key of effective code generation. Compared to natural language, programming code is more regulated and typically has well-defined grammar, thus it is more challenging to exploit the grammar knowledge in natural language.

Our work is also related to transfer learning of deep learning models (Bengio, 2012). There are some recent studies on neural language model adaptation (Yoon et al., 2017; Ma et al., 2017; Chen et al., 2015). However, none of them exploits grammar knowledge. There exists other lines of work on a broad field of general text generation (not for language modeling), such as GAN-based methods (Guo et al., 2017; Li et al., 2017) and VAE-based ones (Hu et al., 2017). It is a promising direction to incorporate syntactic knowledge into these generative models. Our work is also inspired by works on syntactical structured RNNs, such as tree LSTM (Tai et al., 2015), hierarchical RNNs (Chung et al., 2016) and doubly RNNs (Alvarez-Melis and Jaakkola, 2016).

Language models are widely used in a broad range of applications. We believe that a high-quality language model can benefit many downstream tasks, such as machine translation, dialogue systems, and speech recognition. We consider to explore whether our framework can be seamlessly used in those applications, and leave it as future work.

7 Conclusion

In this work, we aim to improve language modeling with shared grammar. Our framework contains two probabilistic components: a constituency parser and a joint generative model. The parser encodes grammar knowledge in natural language, which helps language model quickly land on new corpus. We also propose algorithms for jointly training the two components to fine-tune the language model on new corpus without parsing annotations. Experiments demonstrate that our method improves language modeling on new corpus in terms of both convergence speed and perplexity.

Acknowledgements

This project was supported in part by NSF IIS-1218749,NIH BIGDATA 1R01GM108341, NSF CAREER IIS-1350983, NSF IIS-1639792 EAGER, NSF IIS-1841351 EA-GER, NSF CNS-1704701, ONR N00014-15-1-2340, IntelISTC, NVIDIA, Google and Amazon AWS.

References

David Alvarez-Melis and Tommi S Jaakkola. 2016. Tree-structured decoding with doubly-recurrent neural networks.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Yoshua Bengio. 2012. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 132–139. Association for Computational Linguistics.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 173–180. Association for Computational Linguistics.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

Xie Chen, Tian Tan, Xunying Liu, Pierre Lanchantin, Moquan Wan, Mark JF Gales, and Philip C Woodland. 2015. Recurrent neural network language model adaptation for multi-genre broadcast speech recognition. In *Sixteenth Annual Conference of the International Speech Communication Association*.

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. *arXiv preprint arXiv:1406.1078*.

Do Kook Choe and Eugene Charniak. 2016. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336.

Noam Chomsky. 2002. *Syntactic structures*. Walter de Gruyter.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2016. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*.

Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. 2015. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. In *Proceedings of NAACL-HLT*, pages 199–209.

Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2018. Token-level and sequence-level loss smoothing for RNN language models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2094–2103, Melbourne, Australia. Association for Computational Linguistics.

Ahmad Emami and Frederick Jelinek. 2004. Exact training of a neural syntactic language model. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 1, pages I–245. IEEE.

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

Chengyue Gong, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2018. Frage: frequency-agnostic word representation. In *Advances in Neural Information Processing Systems*, pages 1334–1345.

Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. 2004. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530.

Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. 2017. Long text generation via adversarial training with leaked information. *arXiv preprint arXiv:1709.08624*.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 95. Association for Computational Linguistics.

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

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. 2017. Toward controlled generation of text. In *International Conference on Machine Learning*, pages 1587–1596.

Frederick Jelinek and John D Lafferty. 1991. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–323.

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

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

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

Dan Klein and Christopher D Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting of the association for computational linguistics*.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE.

Lingpeng Kong and Noah A Smith. 2014. An empirical comparison of parsing methods for stanford dependencies. *arXiv preprint arXiv:1404.4314*.

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 the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258, Valencia, Spain. Association for Computational Linguistics.

Tao Lei and Yu Zhang. 2017. Training rnns as fast as cnns. *arXiv preprint arXiv:1709.02755*.

Jiwei Li, Will Monroe, Tianlin Shi, Alan Ritter, and Dan Jurafsky. 2017. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*.

Min Ma, Michael Nirschl, Fadi Biadsy, and Shankar Kumar. 2017. Approaches for neural-network language model adaptation. *Proc. Interspeech 2017*, pages 259–263.

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.

Gábor Melis, Chris Dyer, and Phil Blunsom. 2017. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010a. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010b. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3.

Andriy Mnih and Karol Gregor. 2014. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. *arXiv preprint arXiv:1704.07535*.

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. 2017. Neural language modeling by jointly learning syntax and lexicon. *arXiv preprint arXiv:1711.02013*.

Richard Socher, John Bauer, Christopher D Manning, et al. 2013. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 455–465.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Jun Suzuki, Sho Takase, Hidetaka Kamigaito, Makoto Morishita, and Masaaki Nagata. 2018. An empirical study of building a strong baseline for constituency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 612–618, Melbourne, Australia. Association for Computational Linguistics.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

Sho Takase, Jun Suzuki, and Masaaki Nagata. 2018. Direct output connection for a high-rank language model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4599–4609, Brussels, Belgium. Association for Computational Linguistics.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer.

Yingce Xia, Di He, Tao Qin, Liwei Wang, Nenghai Yu, Tie-Yan Liu, and Wei-Ying Ma. 2016. Dual learning for machine translation. *arXiv preprint arXiv:1611.00179*.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. 2017. Breaking the softmax bottleneck: a high-rank rnn language model. *arXiv preprint arXiv:1711.03953*.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*.

Seunghyun Yoon, Hyeongu Yun, Yuna Kim, Gyu-tae Park, and Kyomin Jung. 2017. Efficient transfer learning schemes for personalized language modeling using recurrent neural network. *arXiv preprint arXiv:1701.03578*.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J Smola, and Le Song. 2017. Variational reasoning for question answering with knowledge graph. *arXiv preprint arXiv:1709.04071*.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 434–443.