Latent Constituents via Self-Attention

September 6, 2018

1 Introduction

2 Problem

We would like to learn a generative model over source sentences $\mathbf{x} = \{x_0, x_1, \ldots\}$, using a distribution over latent trees $\mathbf{z}$. We are primarily interested in the posterior distribution over trees given a sentence, $p(\mathbf{z} \mid \mathbf{x})$. The performance of the generative model is secondary since we would like to analyze the output of the unsupervised model for any linguistic regularities, and use the model as a testbed for linguistic hypotheses (of which I have none at the moment).

(Shen et al., 2018) Yin et al. (2018)

3 PRPN (Shen et al., 2018)

The Parsing-Reading-Predict Network (Shen et al., 2018) is an approach to unsupervised tree induction that uses a soft approximation to a latent variable to a degree of success.

4 The Language Model

Let $\mathbf{x}$ be all tokens and $\mathbf{l}$ all latent variables with each $l_t \in \{0, \ldots, t-1\}$ a categorical RV over all previous indices. In addition to modeling the tokens, Shen et al. (2018) introduce a latent variable at every timestep l_t. They decompose the joint distribution

$$p(\mathbf{x}, \mathbf{l}) = \prod_t p(x_{t+1}, l_t \mid \mathbf{x}_{\leq t}, \mathbf{l}_{<t}) = \prod_t p(x_{t+1} \mid l_t, \mathbf{x}_{\leq t}, \mathbf{l}_{<t}) p(l_t \mid \mathbf{x}_{\leq t}). \tag{1}$$

4.1 $p(x_{t+1} \mid l_t, \mathbf{x}_{\leq t}, \mathbf{l}_{<t})$

The first term takes the form of a self-attentive language model, where the distribution over the next token

$$p(x_{t+1} \mid l_t, \mathbf{x}_{\leq t}, \mathbf{l}_{<t}) = \text{softmax}(f(\boldsymbol{m}_{<t}, l_t))$$

where f is either a linear projection or additional residual blocks, composed with the output of an LSTMN (LSTM Memory-Network), as well as the latent variable l_t. The LSTMN attends over the previous $\boldsymbol{m}_{<t}$ to produce m_t, the current hidden state. This is referred to as the Predict Network.

The attention mechanism is defined similarly throughout their network. The recurrent input to the current timestep is a convex combination of all previous outputs of the current layer. Let $\boldsymbol{m}_{<t}$ be the concatenated output of all previous timesteps, then the recurrent input would be

$$c = \sum_{i=0}^{t-1} s_i(l_t) m_i$$

and the output of the layer at the current timestep would be $m_t = LSTM(c, x_t)$ where s_i is the attention coefficient for that index at timestep t and x_t is the input.

	0	1	2	3	4
$\tilde{s}$	0.4	0.1	0.3	0.2	
$g \mid l_4 = 2$	0	1	1	1	-
$s(l_4 = 2)$	0	0.17	0.5	0.33	-

Figure 1: An example computation of the attention gates with the LVM. Let $t = 4$.

	0	1	2	3	4
$\tilde{s}$	0.4	0.1	0.3	0.2	
d	0.7	0.6	0.3	0.4	0.5
α	-	0.45	0.6	0.55	-
$p(l_4)$	0.45*0.6*0.55=0.1485	(1-0.45)*0.6*0.55=0.1815	(1-0.6)*0.55=0.22	1-0.55=0.45	
F_{l_4}	0.1485	0.1815+0.1485=0.33=0.6*0.55	0.22+0.1815+0.1485=0.55	1	
$s(\mathbb{E}[l_4])$	0.13	0.07	0.36	0.43	-

Figure 2: An example computation of the attention gates. Let $\alpha_i = \frac{d_t - d_i + 1}{2}$ and $t = 4$.

The attention coefficients are a renormalized weighted sum of the dot-product attention we are used to. Let $\tilde{\boldsymbol{s}}_t \in \Delta^{t-2}$, the $t-2$ simplex, be the output of a softmax over the dot-product of memories $\boldsymbol{m}_{<t}$ and query k_t which is a linear function of the input x_t. We then define

$$s_i(l_t) = \frac{g_i(l_t)\tilde{s}_i}{\sum_j g_j(l_t)\tilde{s}_j},$$

where $g_i(l_t) = \mathbf{1}(l_t \leq i)$. We have a set of these **per timestep**. Note that although we define $p(l_t)$ in the following section, we will end up using $\mathbb{E}_{l_t}[g_i(l_t)]$ as a soft approximation to the latent variable model.

[Work through figure 1]

4.2 $p(l_t \mid \mathbf{x}_{\leq t})$

We define

$$p(l_t = k \mid \mathbf{x}_{\leq t}) = \begin{cases} 1 - \alpha_{t-1}, & k = t-1 \\ \prod_{j=1}^{t-1} \alpha_j, & k = 0 \\ (1-\alpha_k)\prod_{j=k+1}^{t-1} \alpha_j, & \text{otherwise} \end{cases}$$

Where the α_k are drawn from 'a Beta distribution'. Thus $l_t \sim \text{Cat}(\theta)$ where the parameters θ are determined through a stick-breaking process. This ends up looking more like a generalized geometric distribution, probability of success or failure at each time step is flexible and there is a maximum number of trials, which hints at an order-statistic interpretation which we will touch upon if there is time.

Note that this cannot be called a Dirichlet Process, as a $DP(\alpha, H_0)$ is defined by $b_i \sim \text{Beta}(1, \alpha), \pi_i = (1-b_i)\prod_{j=0}^{i-1} b_i, \boldsymbol{\pi} \sim GEM(\alpha)$ and with $\theta_i \sim H_0$ we have $G = \sum_i \pi_i \delta_{\theta_i} \sim DP(\alpha, H_0)$. This would only make sense in this context if the base distribution H_0 were also Dirichlet, but this would likely warrant further discussion.

Rather than proceed further in defining the generative model, Shen et al. (2018) instead claim that instead of using $g_i(l_t)$ they use its expectation (taken wrt l_t). We have

$$\mathbb{E}_{l_t}[g_i(l_t)] = \prod_{j=i+1}^{t-1} \alpha_j,$$

and we then use

$$\mathbb{E}_{l_t}[s_i(l_t)] = \frac{\mathbb{E}_{l_t}[g_i(l_t)]\,\tilde{s}_i}{\sum_j \mathbb{E}_{l_t}[g_j(l_t)]\,\tilde{s}_j}$$

as the attention coefficients instead of $s_i(l_t)$. They then parameterize the α_k of timestep t deterministically as follows:

$$\alpha_k = \frac{\text{hardtanh}(2(d_t - d_k)/\tau + 1) + 1}{2},$$

where τ is some scaling parameter. The d_k are obtained from a CNN over the tokens $\mathbf{x}_{k-L:k}$, where L is the width of the convolution kernel (this is set to 5).

[Work through Figure 2]

5 Training

With the soft model that uses the expectation of $s_i(l_t)$ training is very simple. We simply maximize the likelihood of the data directly since the model is fully neural.

6 Extracting a Parse Tree

Use greedy approximation, go over on board?

7 Results

7.1 Language Modeling

The language modeling results are decent, but it's unclear whether the gating mechanism offers any improvement. They perform ablation studies, but they do not seem to be replicable so it is possible the perplexity improvements come mainly from embedding-tying and partly from self-attention.

Model	PPL
RNN-LDA + KN-5 + cache (**?**)	92.0
LSTM (**?**)	78.4
Variational LSTM (**?**)	78.9
CharCNN (**?**)	78.9
Pointer Sentinel-LSTM (**?**)	70.9
LSTM + continuous cache pointer (**?**)	72.1
Variational LSTM (tied) + augmented loss (**?**)	68.5
Variational RHN (tied) (**?**)	65.4
NAS Cell (tied) (**?**)	62.4
4-layer skip connection LSTM (tied) (**?**)	**58.3**
PRPN	61.98

Table 1: PPL on the Penn Treebank test set

7.2 Parsing

The parsing results, on the other hand, are quite good. On WSJ10, the model beats the right-branching baseline but not the CCM model. On the full WSJ, the model gets an averaged sentence F1 of 38.8, which beats random, balanced, and right branching baselines (the highest F1 reported is 21.3). Why should we care about this? There is no reason apriori that limiting self-attention should work at all for learning to rank, as a fully left-branching structure would allow the self-attention to function as normal.

7.3 Ranking Models

Before discussing the Shen et al. (2018) model, we first introduce some notation and two possible ranking models, where both assign each item a latent score. Let $\pi(i)$ denote the rank given to item i, and $\pi^{-1}(j)$ denote the item at rank j.

Model	UF_1
LBRANCH	28.7
RANDOM	34.7
DEP-PCFG (**?**)	48.2
RBRANCH	61.7
CCM (**?**)	71.9
DMV+CCM (**?**)	77.6
UML-DOP (**?**)	**82.9**
PRPN	66.4
UPPER BOUND	88.1

Table 2: Parsing Performance on the WSJ10 dataset

1. Plackett-Luce / Thurstone: There is a hidden score θ_i assigned to every item i. The jth ranking item is chosen from a categorical distribution parameterized by the output of a softmax over the remaining items' scores. Here we have $\mathbf{z} = \{\pi^{-1}(0), \pi^{-1}(1), \ldots, \pi^{-1}(T)\}$. The probability of a given permutation is
$$p(\mathbf{z}) = \prod_{i=0}^{T} \frac{\exp(\theta_{z_i})}{\sum_{k=i}^{T} \exp(\theta_{z_k})} = \prod_{i=0}^{T} \frac{\exp(\theta_{\pi^{-1}(i)})}{\sum_{k=i}^{T} \exp(\theta_{\pi^{-1}(k)})}$$
2. Pairwise Comparison Model (Bradley-Terry): The same as PL, but samples pairwise comparisons from a bernoulli distribution using the item's scores. The probability that item i is ranked higher than item k is $\frac{\theta_{z_i}}{\theta_{z_i}+\theta_{z_k}}$. Similary, we have the probability of a permutation is
$$p(\mathbf{z}) = \prod_{i=0}^{T} \prod_{k=i+1}^{T} \frac{\theta_{z_i}}{\theta_{z_i} + \theta_{z_k}}$$

One could use the Bradley-Terry (pairwise) model of ranking for training a language model, but then switch to the Plackett-Luce model at test time for outputting parse trees. This would allow us to learn a scoring model which can then be shared between the pairwise and global ranking models. In the pairwise model, we could model
$$l_t = \max\left\{i : \alpha_t^i = 1, 0 \le i < t\right\}, \alpha_t^i \sim \text{Bern}\left(\frac{z_t}{z_t + z_i}\right).$$

However, Shen et al. (2018) decide to instead model as a RV the index l_t (at every timestep) instead of the pairwise comparisons. We will get into to the details of their choices in section **??**.

The paper is inspired by the following hypothesis: given a binary tree, a token at index t only requires information up to index l_t that satisfies either of the following conditions:

(a) the token at l_t is the leftmost sibling of the token at t

(b) or, if the token at t is a leftmost child, l_t points to its parent's left sibling's leftmost child.

Although the hypothesis itself is not tested in the implementation and serves only as inspiration, the model does see empirical success in the task of language modeling. The model is realized through the following insight: given a ranking of tokens $\mathbf{x} = \{x_0, \ldots, x_T\}$, recursively splitting $\mathbf{x}$ using the following procedure induces a binary tree where the first token to the left of token x_t that has higher rank, denoted x_{l_t}, also satisfies condition (a) or (b): given token i with the next highest rank, create a subtree $(x_{<i}, (x_i, x_{>i}))$ and recursively perform this procedure until only terminal nodes remain.

References

Yikang Shen, Zhouhan Lin, Chin wei Huang, and Aaron Courville. Neural language modeling by jointly learning syntax and lexicon. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rkgOLb-0W`.

Pengcheng Yin, Chunting Zhou, Junxian He, and Graham Neubig. StructVAE: Tree-structured latent variable models for semi-supervised semantic parsing. In *The 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, Melbourne, Australia, July 2018. URL `https://arxiv.org/abs/1806.07832v1`.