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

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

4.1 $p(x_t \mid l_t, \mathbf{x}_{<t})$

The first term takes the form a language model, where the distribution over the next token

$$p(x_t \mid l_t, \mathbf{x}_{<t}) = f([\mathbf{h}_{l_t:t-1}, h_t])$$

where f is either a linear projection or additional residual blocks, applied to a concatenation of the output of modified LSTMN (LSTM Memory-Network) h_t and a convex combination of the previous LSTMN outputs (i.e. attention over $h_{l_t:t-1}$). f is referred to as the Predict Network.

4.1.1 LSTMN

The LSTMN, referred to as the Reading Network, is as follows: at each time step, the hidden and cell input are given by

$$\{\tilde{h}_t, \tilde{c}_t\} = \sum_{i=1}^{t-1} s_i^t \{h_i, c_i\},$$

a convex combination of the previous hidden and cell outputs. Then

$$h_t, c_t = \texttt{LSTM}(\tilde{h}_t, \tilde{c}_t).$$

The attention is computed in the same way as detailed in section 4.1.2. The input to the LSTMN module is either the previous token or the output of a previous LSTMN layer.

4.1.2 Attention

As all attention modules in the PRPN network use the same formulation of attention we will use y_t to refer to the input of the attention module, $\boldsymbol{g}_t(l_t)$ the gate coefficients which are a function of the RV l_t, $\boldsymbol{m}_t$ as the memories to attend over, and $\boldsymbol{s}_t$ as the attention coefficients. The RV l_t corresponds to the index of the token that satisfies condition (a) or (b). First the key is computed using a linear projection of the input $k_t = W_k y_t$. Then the intermediate scores are computed

$$\tilde{s}_t^i = \text{softmax}\left(\frac{m_t^i k_t^T}{\sqrt{\delta_k}}\right),$$

where δ_k is the dimension of the hidden state. Finally, the gate coefficients are normalized and used to prevent the attention from extending too far in the past.

$$s_t^i = \frac{g_t^i(l_t)}{\sum_j g_t^j(l_t)} \tilde{s}_t^i,$$

where the gate coefficients $\boldsymbol{g}_t(l_t)$ are detailed in section 4.1.3.

4.1.3 Gating Self-attention

Recall that the gates g_t^i also indirectly induce tree structure (due to the previously stated insight that modulating self-attention using a ranking induces a binary tree). At every timestep for token x_t, Shen et al. (2018) define a latent variable l_t which corresponds to the index of the token satisfying either condition (a) or (b).

We then would have

$$g_t^i(l_t) = \mathbf{1}(l_t \leq i < t)$$

where we allow the model at timestep t to only attend up to the token at leftmost limit l_t. We would then renormalize the attention accordingly. In order to calculate the gates $g_t^i(l_t)$ we must model $p(l_t \mid \mathbf{x}_{0:t})$

(Shen et al. (2018) use the posterior distribution in their notation). They propose to model

$$p(l_t = i \mid \mathbf{x}_{0:t-1}) = (1 - \alpha_i^t) \prod_{j=i+1}^{t-1} \alpha_j^t$$

using a stick-breaking process (GEM). The reason this is preferred over a simple categorical distribution over indices is because of the parameterization of the model. Although unlikely, a bimodal distribution is possible with a categorical distribution. I believe it would be possible to reparameterize the model and constrain the parameterization of a categorical distribution to achieve a similar effect.

Note that this itself is not a Dirichlet Process, as there is no base measure H_0. In a $DP(\alpha, H_0)$ we have $\pi \sim GEM(\alpha)$, where $b_i \sim Beta(1, \alpha)$ and $\pi_i = (1 - b_i)\prod_{j=0}^{i-1} b_j$. We then have $\theta_i \sim H_0$ and $G = \sum_i \pi_i \delta_{\theta_i} \sim DP(\alpha, H_0)$. Another indication that this is not a Dirichlet Process is the fact that the only way $p(l_t) = DP$ is if the base distribution is Dirichlet as well. This seems complicated and would definitely warrant further explanation.

Rather than resorting to approximate inference, they instead use the expected value of g_i^t

$$\mathbb{E}\left[g_i^t(l_t)\right] = F_{l_t}(l_t \leq i \mid \mathbf{x}_{0:t-1}) = \prod_{j=i+1}^{t-1} \alpha_j^t.$$

The $\alpha_j^t = \sigma(d_t - d_j)$, where σ is the hard sigmoid function. The $d_j \in \mathbb{R}_+$ are given by a CNN over the token embeddings.

The network used to compute the gates is called the Parsing Network. When computing $p(x_t \mid \ldots)$, we cannot use the posterior score d_t, so an estimate is used based on the previous k words, where k is the kernel width of the convolution. However, afterwards Shen et al. (2018) use the posterior score $d_i \mid x_{i-K}, \ldots, x_i$ for all d_i when $i < t$.

5 Training

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

6 Results

6.1 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

6.2 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.

6.3 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.

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

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

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 4.1.3.

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`.