Avoiding Coadaptation for Composition when Exploring Tree Structures for Sequence Classification*

Anonymous ACL submission

Abstract

If it is true that some sequences have a tree-like optimal substructure that can be exploited for sequence labeling tasks, then this substructure (or a similar one) should be discoverable using Reinforcement Learning techniques.

In this work, we present a task called LISTOPS which involves predicting the output of nested list operations. Since, SPINN performs much better than a comparable RNN, we determine that knowing that knowing the substructure of LISTOPS makes this a more feasible task. Nonetheless, when training SPINN's parsing component with the REINFORCE algorithm, it does not succeed to converge to a parsing strategy with a non-neglibible difference from a trivial parsing strategy. This is similar to what has been seen in related work, and we find this to be due to the following reasons:

1. There is a relationship between composition and parsing in SPINN. Composition is highly dependent on parsing.

2. Naively sampling random binary trees using transition based parsing does not uniformly sample over binary trees.

3. There is a lot of variance in this task, exacerbated by the label bias problem encountered in transition based parsing.

We address the first two issues by:

1. Using a soft wake sleep to interpolate between training the parser and the composition function, limiting the coadaptation between the two.

2. Sampling from a "Catalan Pyramid", ensuring a uniform distribution over binary trees.

Although do not treat the 3rd issue.

*Thanks for the fish (obligatory *Hitchhiker's Guide* Reference)!

1 Introduction

TODO. Related work: SPINN + REINFORCE (Yogatama et al., 2016); Inducing Parse Trees (Dyer et al., 2016)

2 SPINN + REINFORCE

We use a slightly modified thin stack algorithm from Bowman et al. (2016) (pseudocode provided in Appendix A.3). It's simple to add a method that trains the parser with the REINFORCE algorithm (Williams, 1992). REINFORCE approximates the gradient for non-differentiable functions, and due to its high variance, it's necessary to subtract a baseline from the reward. This difference (commonly called the advantage) has much less variance and can be used to make more consistent gradient steps. In our case, the reward is provided by the classification accuracy and we run our model in the inference setting to baseline the reward (Rennie et al., 2016).

To illustrate why REINFORCE may be problematic in the context of SPINN, consider a parser that returns strictly left branching trees (which turns out to be similar as encoding a sequence in the forward direction with a Recurrent Neural Network – RNN). If you train a composition function with this fixated parser for a significant number of examples, the composition function will coadapt to this forward sequential structure. At this point, changing the behavior of the parser to return strictly right branching parse trees (which is similar to encoding a sequence in the backward direction) will introduce examples to the composition function that it has not seen previously, lead-

ing to noisy compositional output, which will propogate noise to any downstream components of the model.

For these reasons, it's highly desirable to prevent coadaptation for two reasons:

1. An unprepared compositional function propogates noise throughout the model, making classification difficult and exploring non-trivial tree structures too costly.

2. Models that attempt to explore will eventually follow a degenerate training regime and fail at their semantic task.

In the following section we detail a training strategy that prevents coadaptation and improves the exploration of substructures, addressing these two problems.

3 Soft Wake Sleep

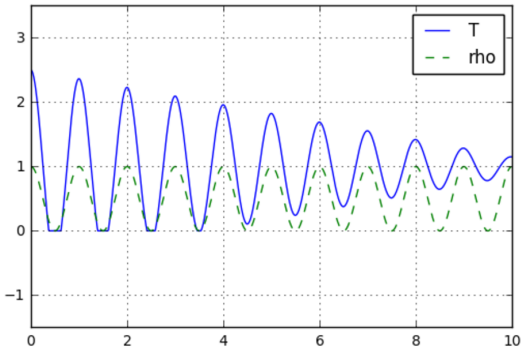

Figure 1: The temperature T used to interpolate between greedy and random distributions when sampling transitions follows a strictly positive sinusoidal wave with annealed magnitude. The scale ρ on the REINFORCE loss follows a similar schedule but is not annealed.

By interpolating between training the parser and the composition function, we apply temperature to the softmax layer of the transition network defined by an annealed sinusoidal schedule. We apply a similar temperature to scale the loss of the transition network. The combination of these two actions searches randomly over all relevant binary trees, teaching the parser to replicate only useful trees in the future and the composition function to exhibit reasonable behavior over many different recursive trees.

Adding temperature to the softmax is done by dividing the input of the softmax by a variable T. The output of the softmax resembles a uniform distribution when $T \to \infty$.

$$\sigma(\frac{\vec{x}}{T}) = \frac{e^{\frac{x}{T}}}{\Sigma_{\hat{x} \in \vec{x}} e^{\frac{\hat{x}}{T}}} \quad (1)$$

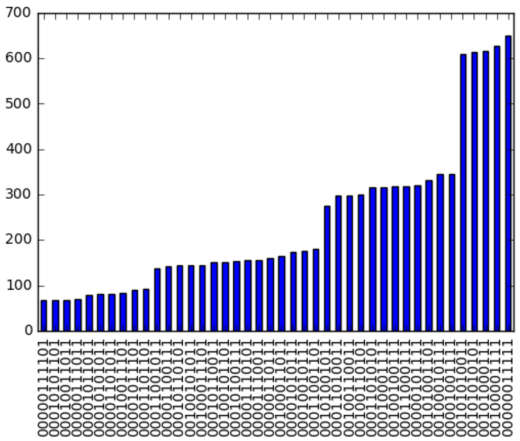

Figure 2: Frequency counts of randomly generated binary trees with N=6 leaves by using the a uniform distribution over Shift/Reduce transitions and the distribution from the Catalan Pyramid for Shift/Reduce transitions in sequence over 10k samples.

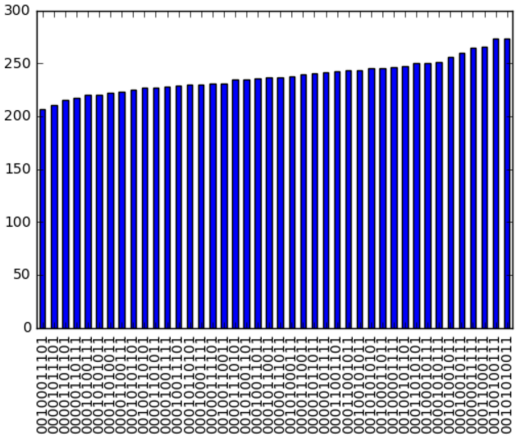

Figure 3: Frequency counts of randomly generated binary trees with N=6 leaves by using the a uniform distribution over Shift/Reduce transitions and the distribution from the Catalan Pyramid for Shift/Reduce transitions in sequence over 10k samples.

Iteratively uniformly sampling transitions does not lead to uniforming sampling over binary trees in Shift Reduce Transition Based Parsing. Rather, the distribution of trees resembles a step function

as seen in Figure 2, which shows the frequency counts of binary trees over iterative random samples. To uniformly sample over binary trees, it's necessary to sample from a distribution representing the number of valid paths containing either Shift or Reduce as the next transition. A naive algorithm can come up with this distribution in exponential time by enumerating all possible trees of a given length, and filtering to relevant trees, but this approach is unusuable for trees with even a moderate number leaves. Fortunately, this distribution seems to have a closed form solution which can be recovered in linear time for any transition in any binary tree, and amortized constant time. We present the recursive algorithm in Figure 3 and call the related distribution the Catalan Pyramid as each row contains fractions over the Catalan Numbers (which represent the count of binary trees with N leaves). A quick search did not turn up previous strategies for sampling uniformly over binary trees in transition based parsing.

The distribution from the Catalan Pyramid is not uniformly random at each time step, so the formula for temperature must be adjusted to take the proper distribution into account. Since temperature leads to a uniform distribution at its limit, it is a linear interpolation of the distribution based purely on the logits ($T = 1$) and the uniform distribution. We can recover the linear parameter directly to perform a similar interpolation between any distribution, including the one from the Catalan Pyramid.

$$\sigma(\frac{\vec{x}}{T}) = i \cdot \sigma(\vec{x}) + (1 - i) \cdot \sigma(0) \quad (2)$$

$$i = \frac{\sigma(\frac{\vec{x}}{T}) - \sigma(0)}{\sigma(\vec{x})} \quad (3)$$

4 Experiments

TODO

5 Discussion

We succeed in consistently finding interesting parse trees, but are unable to find a benefit when it comes to predicting sequence lables nor recover a substructure that matches our expectations. We suspect that treating the 3rd issue could be key to future success with SPINN and REINFORCE.

Acknowledgments

TODO

References

Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. 2016. A fast unified model for parsing and sentence understanding. *arXiv preprint arXiv:1603.06021* .

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

Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. 2016. Self-critical sequence training for image captioning. *arXiv preprint arXiv:1612.00563* .

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. 2016. Learning to compose words into sentences with reinforcement learning. *arXiv preprint arXiv:1611.09100* .

	t=0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
r=0	1	1	$\frac{297}{429}$	$\frac{165}{297}$	$\frac{75}{165}$	$\frac{27}{75}$	$\frac{7}{27}$	$\frac{1}{7}$							
1				1	$\frac{90}{132}$		$\frac{20}{48}$	$\frac{6}{20}$	$\frac{1}{6}$						
2						1	$\frac{28}{42}$	$\frac{14}{28}$	$\frac{5}{14}$	$\frac{1}{5}$					
3								1	$\frac{9}{14}$	$\frac{4}{9}$	$\frac{1}{4}$				
4										1	$\frac{3}{5}$	$\frac{1}{3}$			
5												1	$\frac{1}{2}$		
6														1	

Table 1: Catalan Pyramid. Represents probability of predicting shift at timestep t given there have already been r reduces.

A Algorithms

A.1 Supervised SPINN

Algorithm 1 SPINN

function SPINN(sentences, Ts)
    bufs := {[$\vec{0}$] + s.reverse() | s ∈ sentences}
    stacks := {[$\vec{0}$,$\vec{0}$] | s ∈ sentences}
    Ts′ := transitions.T
    for ts ∈ transitions **do**
        ts′ := BATCHSTEP(bufs, stacks, ts)
        Ts′.$push$(ts′)
    return stacks[-1], Ts′
function BATCHSTEP(bufs, stacks, ts)
    inp := [bufs[-1], stacks[-1], stacks[-2]]
    ts′ := $f^P(f^T$(inp))
    ts′ := VALIDATE(ts, ts′)
    R, S := DISTRIBUTE(ts′, bufs, stacks)
    REDUCE(R); SHIFT(S)
    return ts′
function DISTRIBUTE(ts, bufs, stacks)
    R, S := [], []
    for t, S, Q ∈ ts, bufs, stacks **do**
        if t = REDUCE **then**
            R.$push$(Q.pop(), S, Q)
        else if t = SHIFT **then**
            right, left := S.pop(), S.pop()
            S.$push$([left, right], S, Q)
    return R, S
function REDUCE(R)
    [lefts, rights], bufs, stacks := R.T
    inp = [lefts, rights]
    outp = split(f^C(inp))
    for o ∈ outp, S ∈ stacks **do**
        S.$push$(o)
function SHIFT(S)
    for top, S, Q ∈ S **do**
        S.$push$(top)

A.2 Supervised Model

Algorithm 2 Supervised Model

function MODEL(sentences, transitions, y)
    Ts := transitions.T
    outp, Ts′ := SPINN(sentences, Ts)
    y′ := f^y(outp)
    Loss$_y$:= NLL(y, y′)
    Loss$_T$:= NLL(Ts, Ts′)
    return Loss$_y$, Loss$_T$

A.3 Reinforce Model

Algorithm 3 Reinforce Model

function MODEL(sentences, transitions, y)
    Ts := transitions.T
    outp, Ts′ := SPINN(sentences, Ts)
    y′ := f^y(outp)
    Loss$_y$:= NLL(y, y′)
    Loss$_{RL}$:= REINFORCE(Ts′, Loss$_y$)
    return Loss$_y$, Loss$_{RL}$

B Catalan Pyramid

For a sequence with N tokens, N*2 - 1 transitions, and N - 1 reduce transitions, can build a so-called "Catalan Pyramid" (Table 1) using these 6 rules:

1. $row_{i,0,0} = 1$
2. $row_{i,0,1} = i + 2$
3. $row_{i,n_i-1,1} = Catalan(i + 2)$
4. $row_{i,n_i-1,0} = row_{i,n_i-1,1} - row_{i-1,n_i-2,1}$
5. $row_{i,j,0} = row_{i,j-1,1}$
6. $row_{i,j,1} = row_{i,j,0} + row_{i-1,j,1}$

These rules will build the non-zero fractions in each row of the table (0=numerator, 1=denominator), then padding the fractions as show with zeros will give you the probability of SHIFTing given that you are at timestep t and have already REDUCEd r times.