The Coadaptation Problem when Learning How and What to Compose

Anonymous ACL submission

Abstract

This paper discusses a potential problem with tree-sequence models that induce syntax, in that there exists a failure mode when naively treating the relationship between composition and parsing. The first section presents a tree-sequence model that induces syntax. The second section proposes a strategy that attempts to prevent coadaptation between composition and parsing. The third section covers a method for randomly sampling binary trees in a transition-based setting, a simple and useful technique for transition-based parsing models.

1 Introduction

The Stacked-augmented Parser-Interpreter Neural Network (SPINN) from Bowman et al. (2016) is a hybrid tree-sequence model that uses Recursive Neural Networks (Socher et al., 2011) with TreeLSTM composition units (Tai et al., 2015) to leverage syntactic information for semantic classification tasks. Recent work (Yogatama et al., 2016) has modified SPINN to induce a projective binary structure over its input sentence using signal from the classification task as a reward for a parsing component optimized with the REINFORCE algorithm (Williams, 1992). The tree structures in Yogatama et al. (2016) are observed to be almost trivial in nature, completely left or right leaning, rarely recovering linguistically inspired structure. This trivial behavior resembles similar behavior to a failure mode of the parser when composition becomes too strongly tied (coadapts) to a type of parse tree structure.

If the coadaptation problem is a true problem, it happens when the composition is exposed to many similar looking parse trees consecutively during training time. During evaluation time, any unseen parse tree would provide significant noise to composition. In turn, the semantic classification task becomes unnecessarily difficult. Similarly, switching parsing strategies abruptly during training can confuse composition, inhibiting downstream task performance until composition adjusts to the new strategy.

In lieu of a perfect parsing strategy during train time (or simply having labeled data), one potential method to prevent composition from coadapting to any parsing strategy is to observe as many trees for a given set of data as possible. This can be done by uniformly sampling over valid parse trees for each input. It might be helpful to still greedily parse any input to benefit semantic classification. When doing so it might help to inhibit the learning of the parser so that is more influenced by successfully randomly sampled structure rather than the biased greedy and potentially trivial structure.

Using SPINN with REINFORCE is descibed in section 2. The schedule used to randomly sample trees and modify the magnitude of gradient updates to the parser is called Soft Wake Sleep and is described in section 3. An algorithm for uniformly sampling binary trees in a transition-based parser is covered in section 4.

2 SPINN + REINFORCE

We use a version of the thin stack algorithm from Bowman et al. (2016) akin to what is done in Yogatama et al. (2016). [1]

Our initial experiments with SPINN in this setting unveiled two problematic circumstances:

1. The model commonly follows parsing behavior that returns strictly left or right branching

[1] Code published on github. Left out for anonymous submission.

trees (similar behavior was seen in Yogatama et al. (2016)).

2 When the parser in SPINN becomes fixated in its parsing strategy for a significant number of examples, the composition function coadapts with the parsing behavior.

For these reasons, it's highly desirable to prevent coadaptation because an unprepared compositional function propogates noise throughout the model, making classification difficult and exploring non-trivial tree structures too costly.

In the following section we detail a training strategy that attempts to prevent coadaptation and ease the exploration of substructures.

3 Soft Wake Sleep

To interpolate 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 (as seen in Figure 1). We apply a similar temperature to scale the loss of the transition network.

When T is large (> 1), the distribution for sampling binary trees is most random. Simultaneously, ρ will be high and the parser will be most effected by gradient updates. When T is small (< 1), the parser will sample binary trees from a more greedy-like distribution and the parser will be less effected by updates. This periodic behavior encourages the composition function to act reasonably over many different tree structures, and discourages the parser from becoming fixated in a trivial parsing strategy.

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 $T \to \infty$.

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

Algorithm 1 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
```

Algorithm 2 SPINN

```
function SPINN(sentences, Ts)
    bufs := {[0⃗] + s.reverse() | s ∈ sentences}
    stacks := {[0⃗,0⃗] | 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)
```

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
```

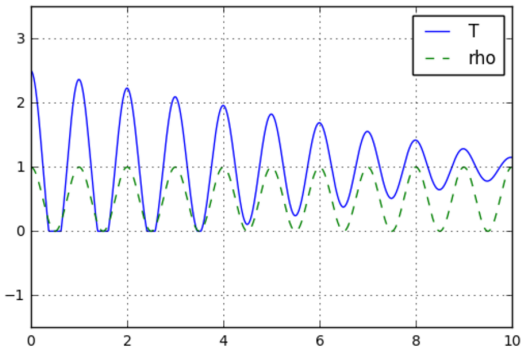

Figure 1: The temperature T used to interpolate between greedy and random distributions when sampling transitions follows a strictly positive sinusoidal wave with annealed magnitude. The scale ρ on the REINFORCE loss follows a similar schedule but is not annealed.

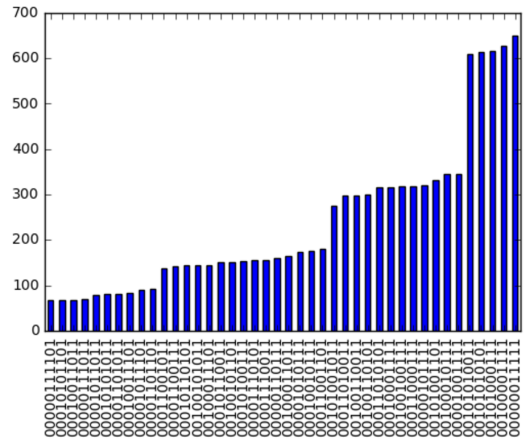

Figure 2: Sampling uniformly 10k times results in frequencies that resemble a step function. Trees are represented as 0s (Shift) and 1s (Reduce).

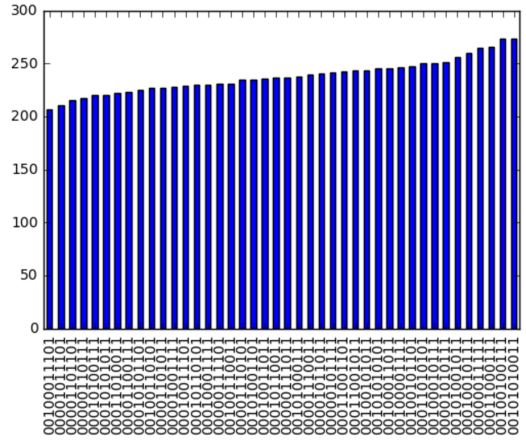

Figure 3: Sampling from the Catalan Pyramid results in a smoother distribution of frequencies.

4 Catalan Pyramid

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, and results in a smoother plot of frequency counts (Figure 3). 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 unusable 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 the following subsection and call the related distribution the Catalan Pyramid (Table 4) 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.

4.1 Algorithm

For a sequence with N tokens, N*2 - 1 transitions, and N - 1 reduce transitions, can build a so-called "Catalan Pyramid" (Table 4) 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.

	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	

Figure 4: Catalan Pyramid. Represents probability of predicting shift at timestep t given there have already been r reduces.

4.2 Linear Interpolation of Probabilities

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.

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

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

Acknowledgments

Left out for anonymous submission.

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*.

Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. 2011. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*. pages 129–136.

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* .

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* .