Tackling Error Propagation through Reinforcement Learning: A Case of Greedy Dependency Parsing

Minh Le and Antske Fokkens

---

**Transition-based Dependency Parsing**

Starting with a sentence and an empty parse, a parser adds one arc at a time until a complete parse is built.

Example: arc-standard parser

```
<br> waves hit stocks themselves on the Big Board
```

Error propagation: an error leads to an unusual configuration which makes further errors more likely.

---

**Reinforcement Learning**

During training: Train on both totally correct and partially correct examples.

- Expected reward = \[\text{sum(score*prob)}\]
- Follows gradient w.r.t. parameters: \[\partial\text{(Expected reward)}/\partial(\theta)\]

During testing: greedy as usual (fast)

Sampling:
- REINFORCE: one sample at a time, unbiased estimate
- Oracle: control setting, uses only the optimal sequence of actions for each sentence
- Random: \[k\] examples per sentence
- Memory: random + remember samples with high scores

---

**Experimental Design**

- Corpora:
  - English: Penn Treebank
  - German: SPMRL

- Transition systems:
  - Arc-standard
  - Arc-eager
  - Swap-standard

- Questions:
  - Oracle vs. Supervised: is reward maximization better than negative log likelihood?
  - Sampling: which method performs the best?
  - RL vs. Supervised: does RL improve performance?
  - Error propagation: does RL reduce error propagation?

---

**Conclusions**

- Reinforcement learning improves performance of a greedy dependency parser
- It does so by reducing error propagation

Other observations:
- Reward maximization is better than negative log likelihood, even with the same number of training examples
- More samples is better (higher accuracy, lower variance)
- Training: slower; testing: equal running time

Future work:
- Reinforcement learning for other natural language processing tasks
  - Efficiency: faster training time

Source code: [bitbucket.org/cltl/redep-java](http://bitbucket.org/cltl/redep-java)