Recall: Eisner’s Algorithm

- Left and right children are built independently, heads are edges of spans
- **Complete item**: all children are attached, head is at the “tall end”
- **Incomplete item**: arc from “tall end” to “short end”, may still expect children

Recall: MST Algorithm

- **Eisner**: search over the space of projective trees, \(O(n^3)\)
- **MST**: find maximum directed spanning tree — finds nonprojective trees as well as projective trees \(O(n^2)\)
- MST restricted to features on single dependencies, Eisner can be generalized to incorporate higher-order features (grandparents, siblings, etc.) at a time complexity cost, or with beaming
Recall: Transition-Based Parsing

- Start: stack contains [ROOT], buffer contains [I ate some spaghetti bolognese]
- Arc-standard system: three operations
  - Shift: top of buffer -> top of stack
  - Left-Arc: $[\sigma \, w_{-2}, w_{-1}] \rightarrow [\sigma \, w_{-1}], \quad w_{-2}$ is now a child of $w_{-1}$
  - Right-Arc: $[\sigma \, w_{-2}, w_{-1}] \rightarrow [\sigma \, w_{-2}], \quad w_{-1}$ is now a child of $w_{-2}$
- End: stack contains [ROOT], buffer is empty []
- Must take $2n$ steps for $n$ words ($n$ shifts, $n$ LA/RA)

This Lecture

- Global Decoding
- Early updating
- Connections to reinforcement learning, dynamic oracles
- State-of-the-art dependency parsers, related tasks

Greedy Training: Static States

- Greedy: each box forms a training example $(s, a^*)$
Global Decoding

- Greedy parser: trained to make the right decision (S, LA, RA) from any gold state we might come to

- What we optimizing when we decode each sentence?
  - Nothing...we’re executing:
    \[ a_{\text{best}} \leftarrow \arg\max_a w^T f(s, a) \]
    \[ s \leftarrow a_{\text{best}}(s) \]

- Why might this be bad?

Global Decoding: A Cartoon

- Correct: Right-arc, Shift, Right-arc, Right-arc

- Both wrong! Also both probably low scoring!

- Correct, high scoring option

Global Decoding

- Lookahead can help us avoid getting stuck in bad spots

- Global model: maximize sum of scores over all decisions

- Similar to how Viterbi works: we maintain uncertainty over the current state so that if another one looks more optimal going forward, we can use that one
Global Shift-Reduce Parsing

- Greedy: repeatedly execute
  \[ a_{\text{best}} \leftarrow \text{argmax}_s w^T f(s, a) \]
  \[ s \leftarrow a_{\text{best}}(s) \]
- Can we do search exactly?
  - How many states are there?
  - No! Use beam search

- Global:
  \[ \text{argmax}_{s, a} f(s, a) = \sum_{i=1}^{2n} w^T f(s_i, a_i) \]
  \[ s_{i+1} = a_i(s_i) \]

Beam search gave us the lookahead to make the right decision

Training Global Parsers

- Can compute approximate maxes with beam search
  \[ \text{argmax}_{s, a} f(s, a) = \sum_{i=1}^{2n} w^T f(s_i, a_i) \]

- Structured SVM: do loss-augmented decode, gradient = gold feats - guess feats

- Structured perceptron: normal decode, gradient = gold feats - guess feats

What happens if we set beam size = 1?

Global Training

For each epoch
For each sentence
  For i=1...2*len(sentence)  # 2n transitions in arc-standard
  \[ \text{beam}[i] = \text{compute_successors}(\text{beam}[i-1]) \]
  \[ \text{prediction} = \text{beam}[2*\text{len(sentence)}, 0] \]  # argmax = top of last beam
  # Feats are cumulative over the whole sentence
  apply_gradient_update(feats(gold) - feats(prediction))
Global Training

» In global, we keep going if we screw up!

Start state — Gold end state

» Learn negative weights for features in these states — greedy training would never see these states

Global vs. Greedy

» In global, we keep going if we screw up!

Start state — Pred end state — Gold end state

» Greedy: 2n local training examples

» Global: one global example

Early Updating

Collins and Roark (2004)
Early Updating

- ROOT gave him dinner
- [ROOT gave dinner] []
- +0.9
- [ROOT gave] [dinner]
- -1.2
- [ROOT gave] [him]
- -2.0
- [ROOT gave dinner] []
- -3.0

- Gold has fallen off beam!
- Update: gold feats - pred feats

- Wrong state — we already messed up!
- Ideally we don’t want to penalize this decision (update away from it) — instead just penalize the decision that was obviously wrong

Early Updating

- Solution: make an update as soon as the gold parse falls off the beam
- gold feats - guess feats computed up to this point

Training with Early Updating

For each epoch
For each sentence
For i=1...2*len(sentence)  # 2n transitions in arc-standard
beam[i] = compute_successors(beam[i-1])
If beam[i] does not contain gold:
# Feats are cumulative up until this point
apply_gradient_update(feats(gold[0:i]) - feats(beam[i,0]))
break
# Gold survived to the end but may still not be one-best
apply_gradient_update(feats(gold) - feats(beam[2*len(sentence),0]))
Connections to Reinforcement Learning

- Part of the benefit is we see states we wouldn’t have seen during greedy decoding
- (Still true even with early updating due to beam search)

Better Greedy Algorithm

For each epoch:
  For each sentence:
    Parse the sentence with the current weights
    For each state $s$ in the parse:
      Determine what the right action $a^*$ was
      How do we determine this?
      Train on this example (update towards $f(s, a^*)$, away from $f(s, a_{pred})$)

Dynamic Oracles

- When you make some bad decisions, how do you dig yourself out?
- $\text{best\_possible\_tree}(s)$: computes the optimal decision sequence from state $s$ to the end resulting the lowest overall loss
- Implemented by a bunch of logic that looks at the tree: “if we put a right-arc from $a \rightarrow b$, we can’t give $b$ any more children, so lose a point for every unbound child, also lose a point if $a$ isn’t $b$’s head…”
- Score of decision $a$ in state $s$ leading to $s'$:
  $\text{loss}(a) = \text{loss}(\text{best\_possible\_tree}(s')) - \text{loss}(\text{best\_possible\_tree}(s))$
- $a^* = \arg\min_a \text{loss}(a)$

Goldberg and Nivre (2012)
Connections to Reinforcement Learning

- Markov Decision Process: states $s$, actions $a$, transitions $T$, rewards $r$, discount factor $\gamma$
- $T$ is deterministic for us, $\gamma = 1$ (no discount)
- Maximize sum of rewards over the parse
- One reward system: $r = 1$ if action is what dynamic oracle says, 0 otherwise
- Using the “better greedy algorithm” corresponds to on-policy learning here
- But dynamic oracles are hard to build :(  

Searn

- What if we just had a loss function $\ell(y, y^*)$ that scored whole predictions? I.e., all reward comes at the end
- Searn: framework for turning structured problems into classification problems
- Take the current policy (= weights), generate states $s$ by running that policy on a given example
- Evaluate action $a$ in state $s$ by taking $a$, then following your current policy to completion and computing the loss (= best_possible_loss is approximated by current policy)
- DAGGER algorithm from RL literature  

Motivation

State $s$, evaluate actions $a$ ...by computing losses here

Global Models vs. RL

- Structured prediction problems aren’t really “RL” in that the environment dynamics are understood
- RL techniques are usually not the right thing to do unless you loss function and state space are really complicated
- Otherwise, best to use dynamic oracles or global models
- These issues arise far beyond parsing! Coreference, machine translation, dialogue systems, ...
State-of-the-art Parsers

- 2005: MSTParser got solid performance (~91 UAS)
- 2010: Koo's 3rd-order parser was SOTA for graph-based (~93 UAS)
- 2012: Maltparser was SOTA was for transition-based (~90 UAS), similar to what you’ll build
- 2014: Chen and Manning got 92 UAS with transition-based neural model

Current state-of-the-art, released by Google publicly
- 94.61 UAS on the Penn Treebank using a global transition-based system with early updating
- Additional data harvested via “tri-training”
- Feedforward neural nets looking at words and POS associated with
  - Words at the top of the stack
  - Those words’ children
  - Words in the buffer
- Feature set pioneered by Chen and Manning (2014), Google fine-tuned it

Chen and Manning (2014)

Parsey McParseFace

Andor et al. (2016)
Stack LSTMs

- Use LSTMs over stack, buffer, past action sequence. Trained greedily
- Slightly less good than Parsey

Semantic Role Labeling

- Another kind of tree-structured annotation, like a subset of dependency
- Verb roles from Propbank (Palmer et al., 2005), nominal predicates too

Gold

\[
\begin{array}{c}
\text{ARG1} \\
\text{V} \\
\text{ARG2} \\
\text{ARG3}
\end{array}
\]

\[
\text{Housing starts are expected to quicken a bit from August's pace}
\]

quicken:
- Arg0-PAG: causer of speed-up
- Arg1-PPT: thing becoming faster (vrole: 45.4-patient)
- Arg2-EXT: EXT
- Arg3-DIR: old speed
- Arg4-PRD: new speed

Figure from He et al. (2017)

Abstract Meaning Representation

- Graph-structured annotation
- Superset of SRL: full sentence analyses, contains coreference and multi-word expressions as well
- F1 scores in the 60s: hard!
- So comprehensive that it's hard to predict, but still doesn't handle tense or some other things...

The boy wants to go

Takeaways

- Global training is an alternative to greedy training
- Use beam search for inference combined with early updating for best results
- Dynamic oracles + following the predicted path in the state space looks like reinforcement learning
Survey

- Pace of last lecture + this lecture: [too slow] [just right] [too fast]
- Pace of class overall: [too slow] [just right] [too fast]
- Write one thing you like about the class
- Write one thing you don’t like about the class