Parsing II + Imitation Learning

CMSC 723 / LING 723 / INST 725

Hal Daumé III [he/him]

31 Oct 2019

(many slides c/o Marine Carpuat or Joakim Nivre or Ryan McDonald)

Announcements, logistics

- Project
  - **Reminder: P2 due on Tuesday**
  - Please go to ELMS and create a group for your group so that we can…
    - Will send our feedback
    - Send peer feedback
  - Assignment of teams to TAs posted, please meet with me and also with your TA before Thanksgiving break (sign-ups will be posted on ELMS)
- Homeworks 4 & 5 officially merged
  - To help with exam prep, this will be HW4-w, HW5-w, HW45-p (each worth 1/3 of grade)
- Grading
  - Midterm out – MIN: 54.0, MED: 86.0, MAX: 98.0, MEAN: 83.82, STD: 10.7
    Solution posted, exams distributed (log in to gradescope with university authentication)
  - Score adjustments: You must issue them on gradescope within one week of today

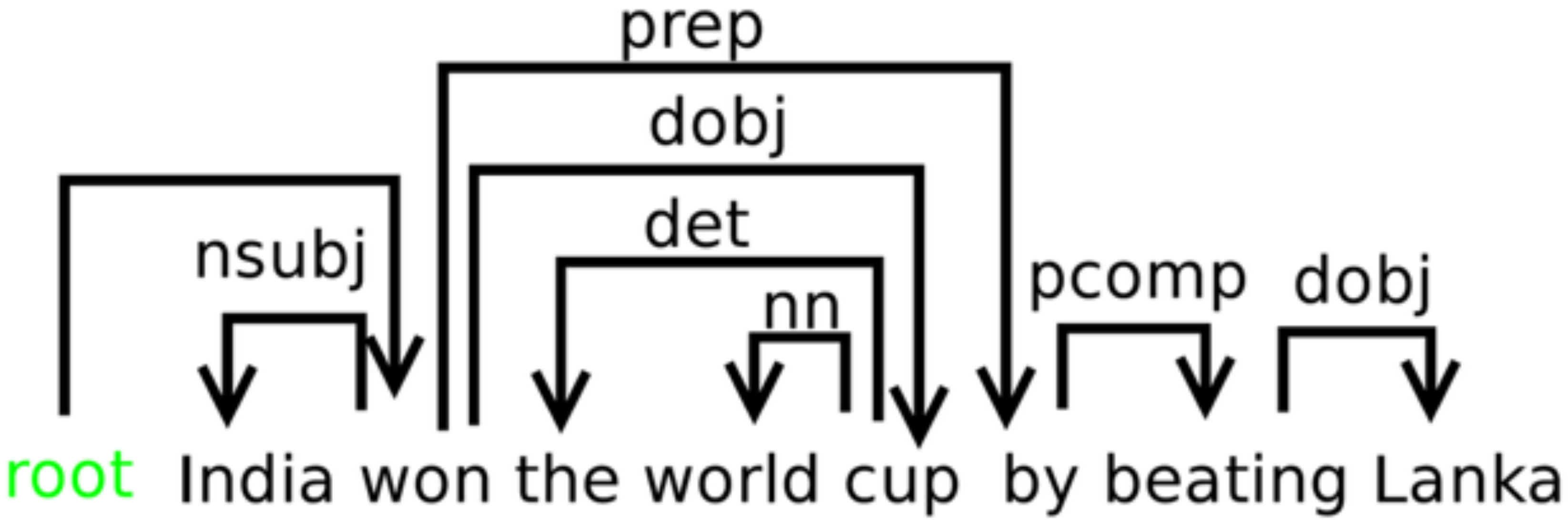
Last time

- Two views of syntactic structures
  - Context-Free Grammars
  - Dependency grammars
  - Can be used to capture various facts about the structure of language (but not all!)
- Treebanks as an important resource for NLP
- Transition-based dependency parsing
  - Shift-reduce parsing
  - Transition system
  - Learning/predicting parsing actions

Today

- Learning to correct mistakes
- More on features

Example Dependency Parse



Transition-based dependency parsing

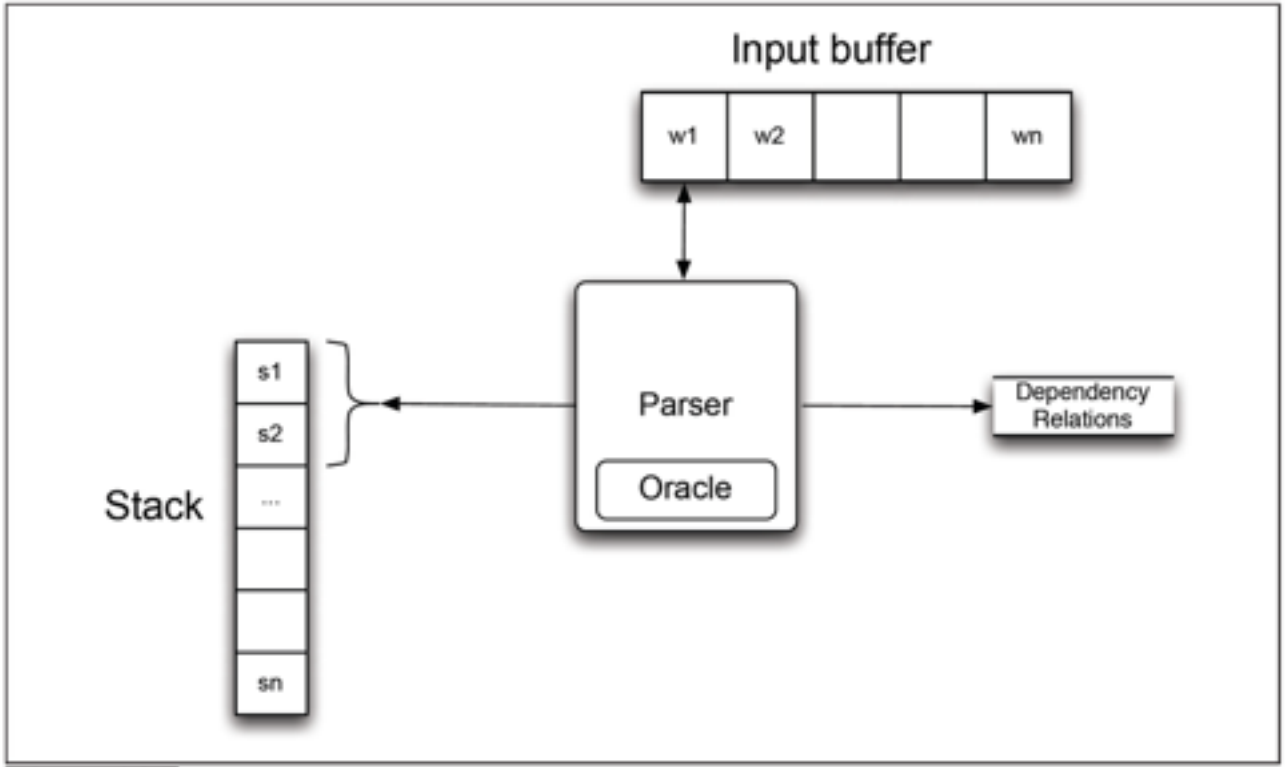


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

- Builds on shift-reduce parsing [Aho & Ullman, 1927]
- **Configuration**
  - **Stack**
  - **Input buffer** of words
  - Set of dependency relations
- Goal of parsing
  - find a final configuration where
  - all words accounted for
  - Relations form dependency tree

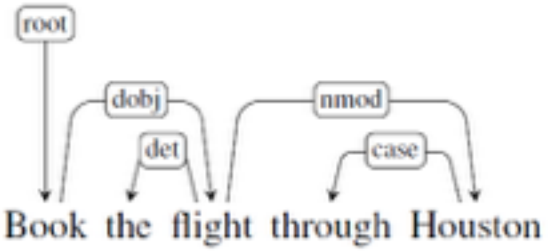
Where to we get an oracle?

- Multiclass classification problem
  - Input: current parsing state (e.g., current and previous configurations)
  - Output: one transition among all possible transitions
  - Q: size of output space?

- Supervised classifiers can be used
  - E.g., perceptron
- Open questions
  - What are good features for this task?
  - Where do we get training examples?

Generating Training Examples

- What we have in a treebank



- What we need to train an oracle
  - Pairs of configurations and

Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

Figure 14.8 Generating training items consisting of configuration/predicted action pairs by simulating a parse with a given reference parse.

Generating training examples

- Approach: simulate parsing to generate reference tree

- Given
  - A current config with stack S, dependency relations Rc
  - A reference parse (V,Rp)
- Do

LEFTARC(r): **if** $(S_1\ r\ S_2) \in R_p$
RIGHTARC(r): **if** $(S_2\ r\ S_1) \in R_p$ **and** $\forall r', w\ s.t.(S_1\ r'\ w) \in R_p$ **then** $(S_1\ r'\ w) \in R_c$
SHIFT: **otherwise**

Features example

Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

Figure 14.9 Standard feature templates for training transition-based dependency parsers. In the template specifications s_n refers to a location on the stack, b_n refers to a location in the word buffer, w refers to the wordform of the input, and t refers to the part of speech of the input.

Research highlight: Dependency parsing with stack-LSTMs

- From Dyer et al. 2015: http://www.aclweb.org/anthology/P15-1033

- Idea
  - Instead of hand-crafted feature
  - Predict next transition using recurrent neural networks to learn representation of stack, buffer, sequence of transitions

Research highlight:
Dependency parsing with stack-LSTMs

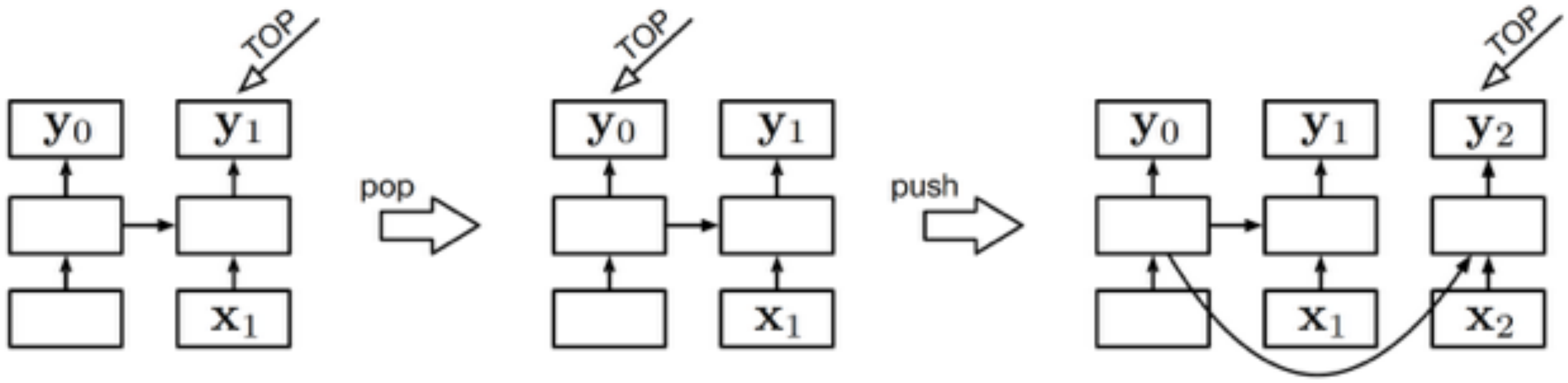


Figure 1: A stack LSTM extends a conventional left-to-right LSTM with the addition of a stack pointer (notated as TOP in the figure). This figure shows three configurations: a stack with a single element (left), the result of a pop operation to this (middle), and then the result of applying a push operation (right). The boxes in the lowest rows represent stack contents, which are the inputs to the LSTM, the upper rows are the outputs of the LSTM (in this paper, only the output pointed to by TOP is ever accessed), and the middle rows are the memory cells (the $\mathbf{c}_t$'s and $\mathbf{h}_t$'s) and gates. Arrows represent function applications (usually affine transformations followed by a nonlinearity), refer to §2.1 for specifics.

Research highlight: Dependency parsing with stack-LSTMs

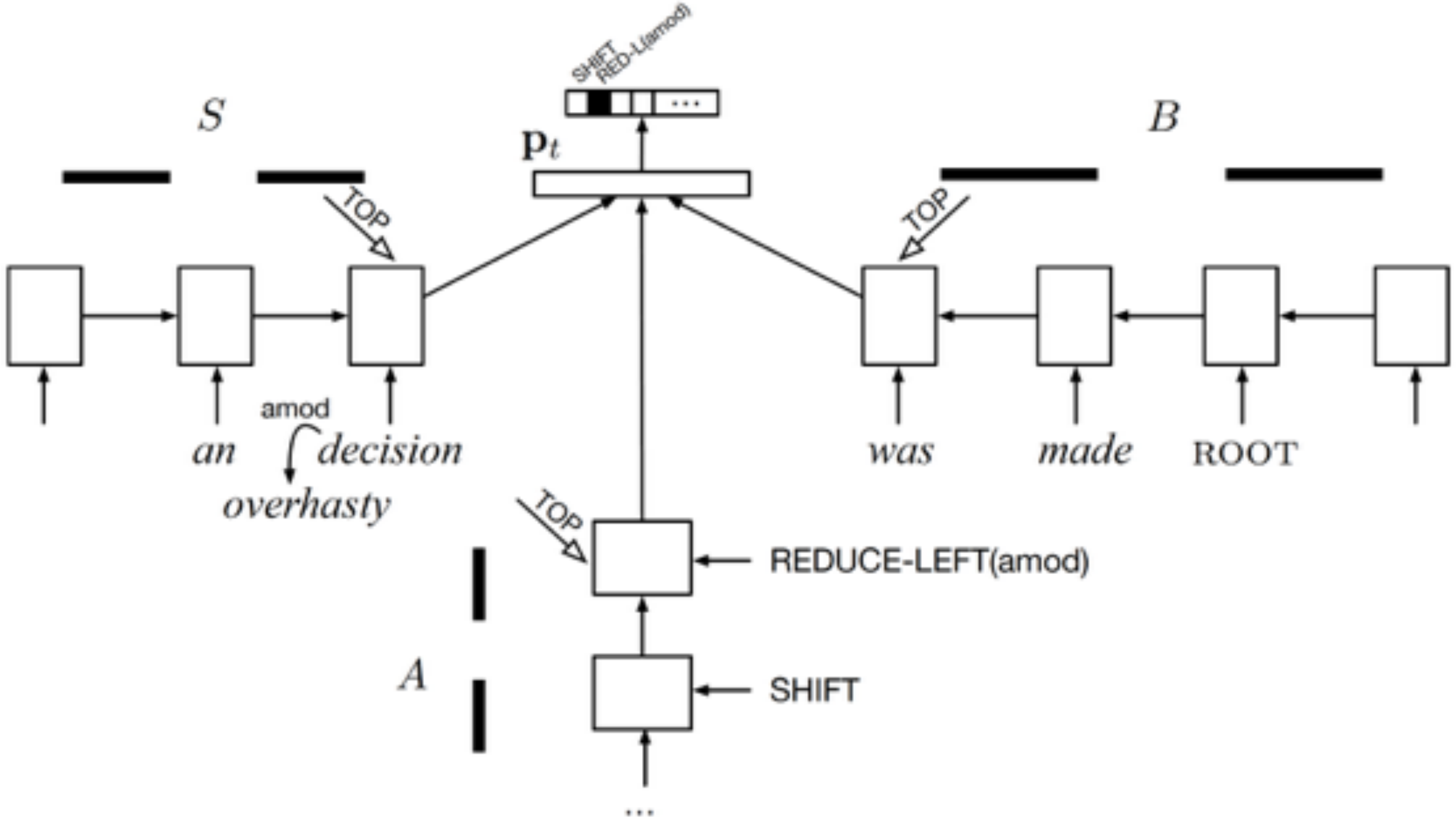


Figure 2: Parser state computation encountered while parsing the sentence "*an overhasty decision was made*." Here S designates the stack of partially constructed dependency subtrees and its LSTM encoding; B is the buffer of words remaining to be processed and its LSTM encoding; and A is the stack representing the history of actions taken by the parser. These are linearly transformed, passed through a ReLU nonlinearity to produce the parser state embedding $\mathbf{p}_t$. An affine transformation of this embedding is passed to a softmax layer to give a distribution over parsing decisions that can be taken.

Improving the oracle in transition-based dependency parsing

- Issues with oracle we've used so far
  - Based on configuration sequence that produces gold tree
  - What if there are multiple sequences for a single gold tree?
  - How can we recover if the parser deviates from gold sequence?

- Goldberg & Nivre [2012] propose an improved oracle

A Dynamic Oracle for Arc-Eager Dependency Parsing

Yoav Goldberg[1] *Joakim Nivre*[1,2]
(1) Google Inc.
(2) Uppsala University
yogo@google.com, joakim.nivre@lingfil.uu.se

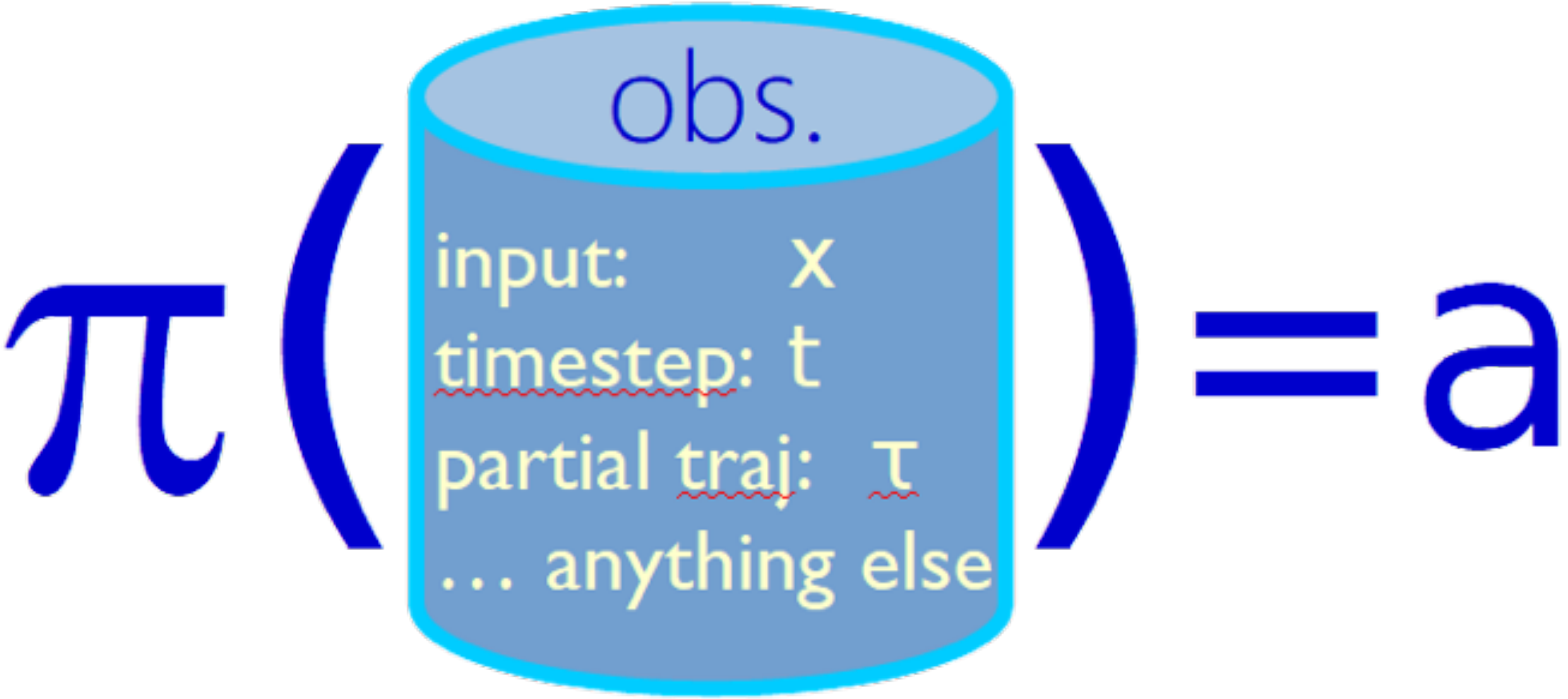
Imitation learning

= shortcutting exploration by
observing and imitating expert (teacher)

teacher provides demonstrations (or other labels)

Policies

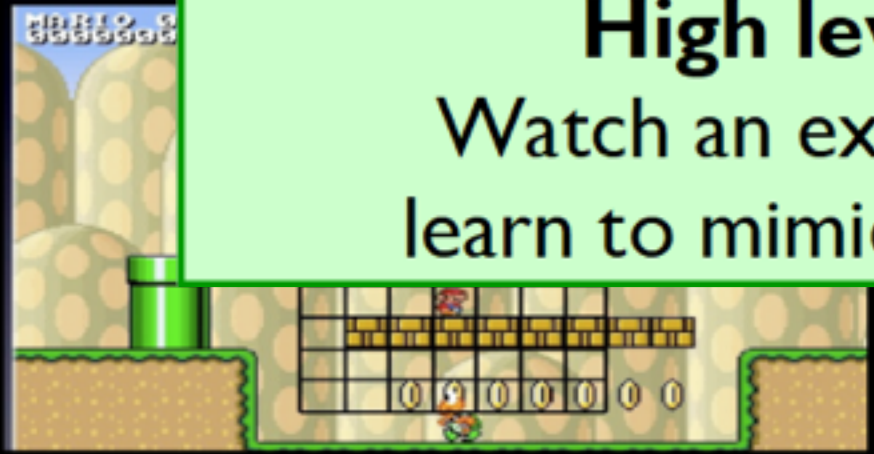
- A policy maps observations to actions



An analogy from playing Mario

From Mario AI competition 2009

Input:



Output:

Jump in {0,1}

,1}

}

0,1}

High level goal:
Watch an expert play and
learn to mimic her behavior

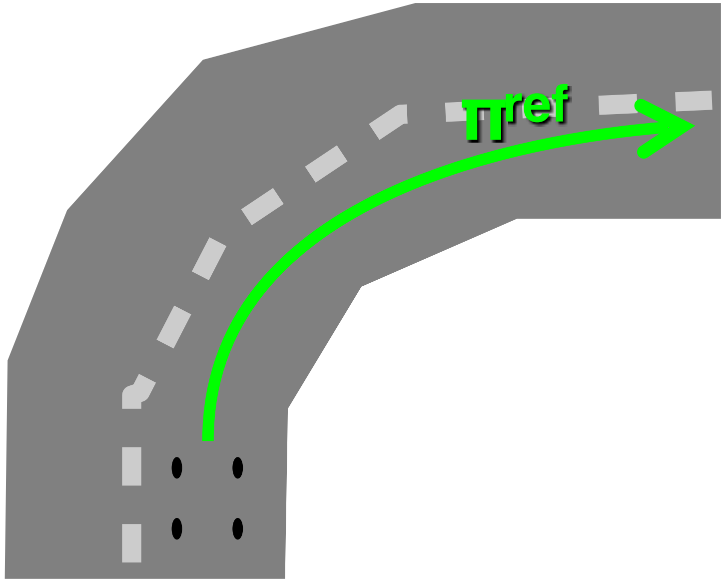
Expert


FPS: 24
Attempt: 1 of 1
AgentLinear
Selected Actions:
RIGHT
SPEED

Warm-up: Supervised learning

1. Collect trajectories from expert $\boldsymbol{\pi}^{\mathbf{ref}}$
2. Store as dataset $\mathbf{D} = \{ (o, \boldsymbol{\pi}^{\mathbf{ref}}(o,y)) \mid o \sim \boldsymbol{\pi}^{\mathbf{ref}} \}$
3. Train classifier $\boldsymbol{\pi}$ on $\mathbf{D}$

- **Let π play the game!**



Classifier


FPS: 24
Attempt: 1 of 1
AgentLinear
Selected Actions:
RIGHT
SPEED

What’s the biggest failure mode?

Learning from an expert: DAgger

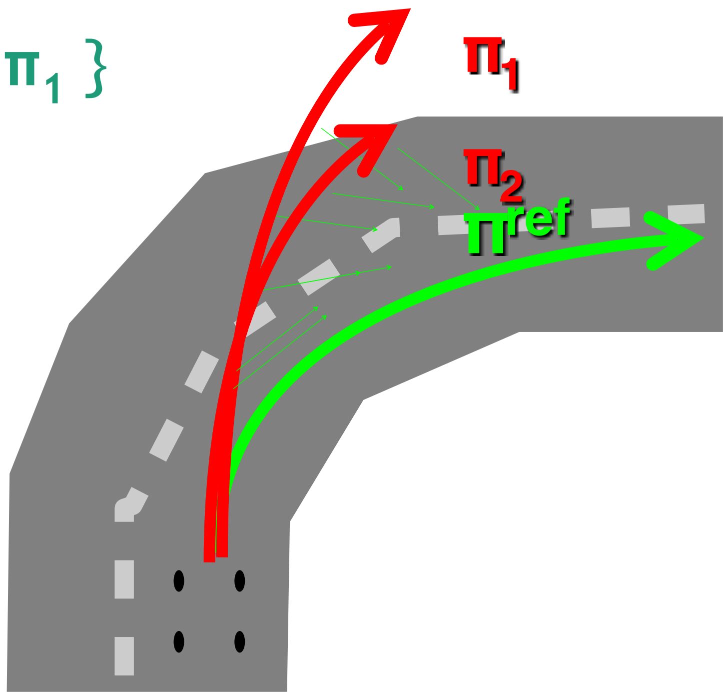
1. Collect trajectories from expert π^{ref}
2. Dataset $\mathbf{D_0}$ = { (o, π^{ref}(o,y)) | o ~ π^{ref} }
3. Train π_1 on $\mathbf{D_0}$
4. Collect new trajectories from π_1
   - But let the *expert* steer!
5. Dataset $\mathbf{D_1}$ = { (o, π^{ref}(o,y)) | o ~ π_1 }
6. Train π_2 on $\mathbf{D_0} \cup \mathbf{D_1}$

If N = T log T,

$$\mathbf{L(\pi_n) < T\ \boxed{?}_N + O(1)}$$

for some n

- In general:
- $\mathbf{D_n}$ = { (o, π^{ref}(o,y)) | o ~ π_n }
- Train π_{n+1} on $\cup_{i \le n}\ \mathbf{D_i}$



Interactive Expert

Labeled data → Reference policy

- Given partial trajectory $a_1,a_2,\ldots,a_{t-1}$ and true label y
- The *minimum achievable loss* is:

$$\min_{(a_t,a_{t+1},\ldots)} \mathrm{loss}(y, \hat{y}(\vec{a}))$$

- The *optimal action* is the corresponding a_t

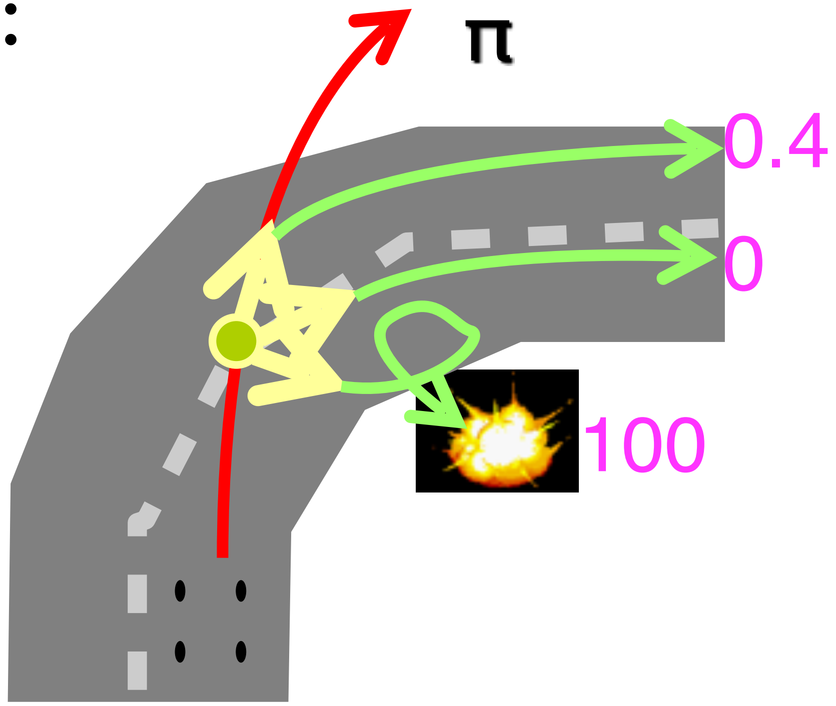
- The *optimal policy* is the policy that always selects the optimal action

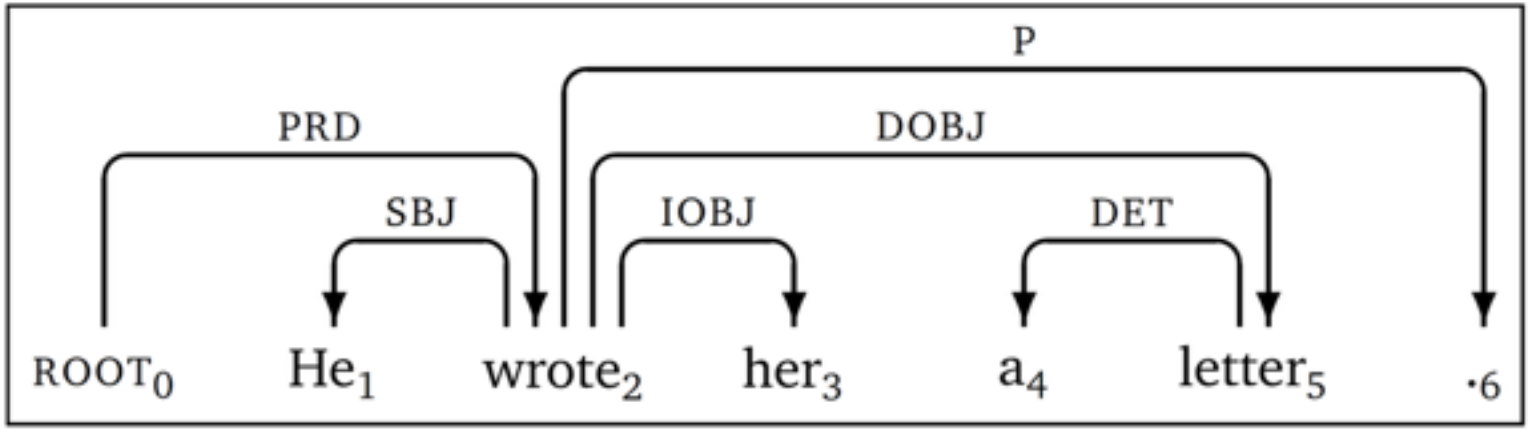
What does this mean for sequence labeling?

Learning to search: AggraVaTe

1. Let learned policy π drive for t timesteps to obs. o
2. For each possible action a:
   - Take action a, and let expert π^{ref} drive the rest
   - Record the overall loss, c_a
3. Update π based on example:

   $(o, \ \langle c_1, c_2, \ldots, c_K \rangle \ )$
4. Goto (1)





SH, LA_{SBJ}, RA_{PRD}, RA_{IOBJ}, SH, LA_{DET}, RE, RA_{DOBJ}, RE RA_{P}

SH, LA_{SBJ}, RA_{PRD}, RA_{IOBJ}, RE, SH, LA_{DET}, RA_{DOBJ}, RE RA_{P}

Exercise: which of these transition sequences produces the gold tree on the left?

Stack

Buffer

Dependency Arcs

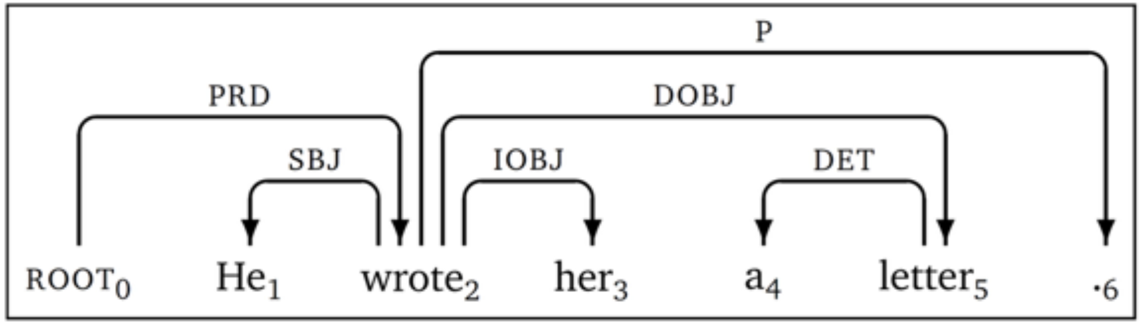
Arc from position j to position i, with dependency label l

Algorithm 1 Standard oracle for arc-eager dependency parsing

if $c = (\sigma|i, j|\beta, A)$ and $(j, l, i) \in A_{\text{gold}}$ **then**
    $t \leftarrow$ LEFT-ARC$_l$
else if $c = (\sigma|i, j|\beta, A)$ and $(i, l, j) \in A_{\text{gold}}$ **then**
    $t \leftarrow$ RIGHT-ARC$_l$
else if $c = (\sigma|i, j|\beta, A)$ and $\exists k[k < i \wedge \exists l[(k, l, j) \in A_{\text{gold}} \vee (j, l, k) \in A_{\text{gold}}]]$ **then**
    $t \leftarrow$ REDUCE
else
    $t \leftarrow$ SHIFT
return t

Algorithm 1 Standard oracle for arc-eager dependency parsing

if $c = (\sigma|i, j|\beta, A)$ and $(j, l, i) \in A_{\text{gold}}$ **then**
    $t \leftarrow$ LEFT-ARC$_l$
else if $c = (\sigma|i, j|\beta, A)$ and $(i, l, j) \in A_{\text{gold}}$ **then**
    $t \leftarrow$ RIGHT-ARC$_l$
else if $c = (\sigma|i, j|\beta, A)$ and $\exists k[k < i \wedge \exists l[(k, l, j) \in A_{\text{gold}} \vee (j, l, k) \in A_{\text{gold}}]]$ **then**
    $t \leftarrow$ REDUCE
else
    $t \leftarrow$ SHIFT
return t



SH, LA$_{\text{SBJ}}$, RA$_{\text{PRD}}$, RA$_{\text{IOBJ}}$, SH, LA$_{\text{DET}}$, RE, RA$_{\text{DOBJ}}$, RE RA$_{\text{P}}$
SH, LA$_{\text{SBJ}}$, RA$_{\text{PRD}}$, RA$_{\text{IOBJ}}$, RE, SH, LA$_{\text{DET}}$, RA$_{\text{DOBJ}}$, RE RA$_{\text{P}}$

Which of these transition sequences does the oracle algorithm produce?

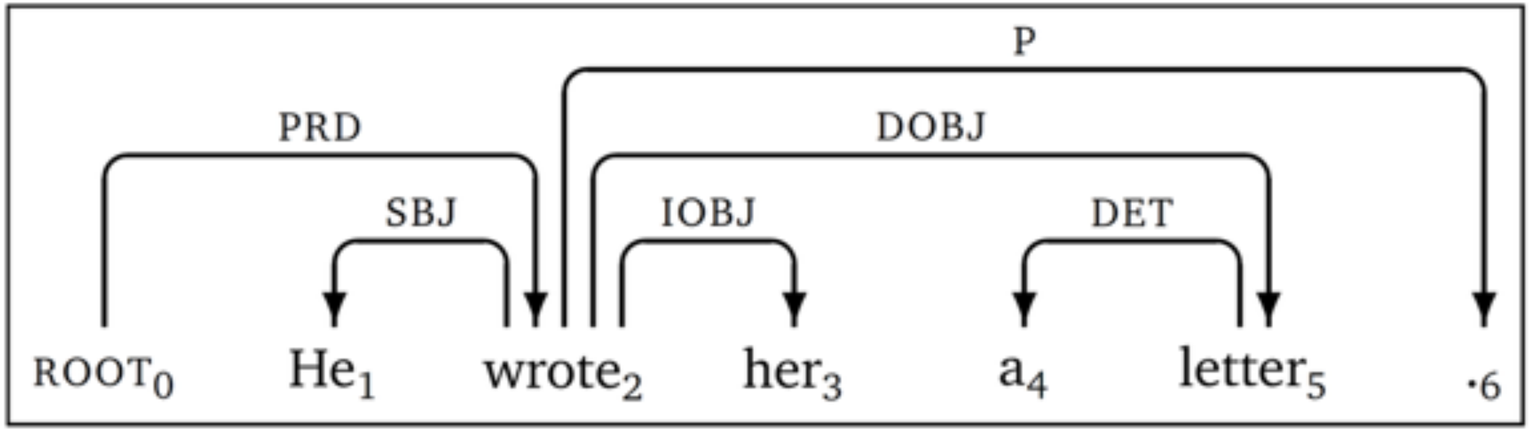
Improving the oracle in transition-based dependency parsing

- Issues with oracle we've used so far
  - Based on configuration sequence that produces gold tree
  - **What if there are multiple sequences for a single gold tree?**
  - How can we recover if the parser deviates from gold sequence?

- Goldberg & Nivre [2012] propose an improved oracle

A Dynamic Oracle for Arc-Eager Dependency Parsing

Yoav Goldberg[1] *Joakim Nivre*[1,2]
(1) Google Inc.
(2) Uppsala University
yogo@google.com, joakim.nivre@lingfil.uu.se

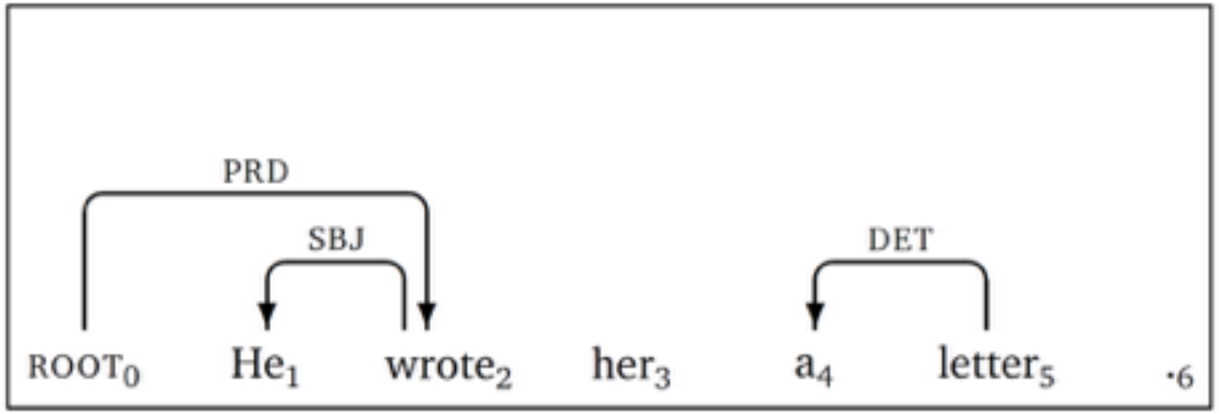


SH, LA_{SBJ}, RA_{PRD}, SHIFT

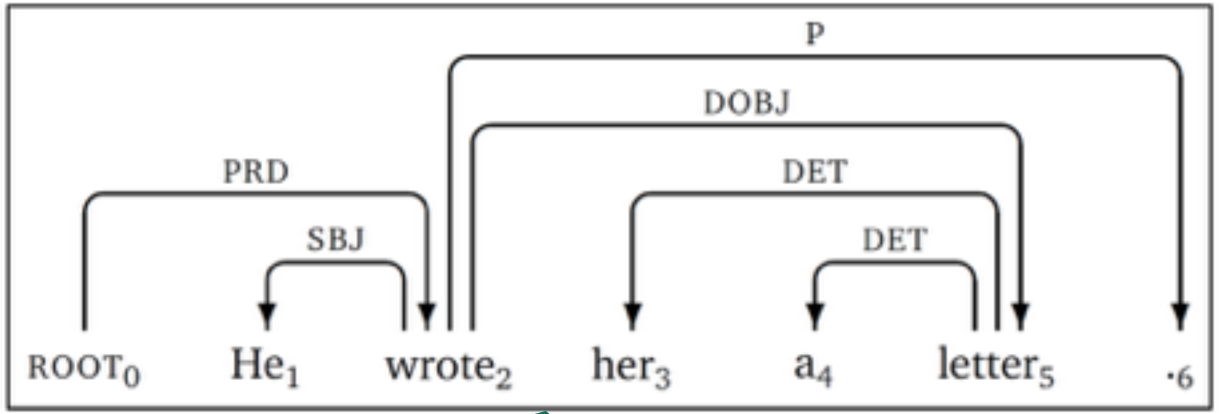
At test time, suppose the 4th transition predicted is SHIFT instead of RA_{IOBJ}
What happens if we apply the oracle next?

Measuring distance from gold tree

- **Labeled attachment loss:** number of arcs in gold tree that are not found in the predicted tree



Loss = 3



Loss = 1

Improving the oracle in transition-based dependency parsing

- Issues with oracle we've used so far
  - Based on configuration sequence that produces gold tree
  - What if there are multiple sequences for a single gold tree?
  - **How can we recover if the parser deviates from gold sequence?**

- Goldberg & Nivre [2012] propose an improved oracle

A Dynamic Oracle for Arc-Eager Dependency Parsing

Yoav Goldberg[1] *Joakim Nivre*[1,2]
(1) Google Inc.
(2) Uppsala University
yogo@google.com, joakim.nivre@lingfil.uu.se

Proposed solution:
2 key changes to training algorithm

Algorithm 3 Online training with a dynamic oracle

$\mathbf{w} \leftarrow 0$
for $I = 1 \rightarrow$ ITERATIONS **do**
    for sentence x with gold tree G_{gold} in corpus **do**
        $c \leftarrow c_s(x)$
        while c is not terminal **do**
            $t_p \leftarrow \arg\max_t \mathbf{w} \cdot \phi(c, t)$
            $\text{ZERO\_COST} \leftarrow \{t \mid o(t; c, G_{\text{gold}}) = \textbf{true}\}$
            $t_o \leftarrow \arg\max_{t \in \text{ZERO\_COST}} \mathbf{w} \cdot \phi(c, t)$
            if $t_p \notin \text{ZERO\_COST}$ **then**
                $\mathbf{w} \leftarrow \mathbf{w} + \phi(c, t_o) - \phi(c, t_p)$
            $t_n \leftarrow \text{CHOOSE\_NEXT}(I, t_p, \text{ZERO\_COST})$
            $c \leftarrow t_n(c)$
return $\mathbf{w}$

Any transition that can possibly lead to a correct tree is considered correct

Explore non-optimal transitions

Proposed solution:
2 key changes to training algorithm

Algorithm 3 Online training with a dynamic oracle

$\mathbf{w} \leftarrow 0$
for $I = 1 \rightarrow$ ITERATIONS **do**
    for sentence x with gold tree G_{gold} in corpus **do**
        $c \leftarrow c_s(x)$
        while c is not terminal **do**
            $t_p \leftarrow \arg\max_t \mathbf{w} \cdot \phi(c,t)$
            $\text{ZERO\_COST} \leftarrow \{t \mid o(t;c,G_{\text{gold}}) = \textbf{true}\}$
            $t_o \leftarrow \arg\max_{t \in \text{ZERO\_COST}} \mathbf{w} \cdot \phi(c,t)$
            if $t_p \notin \text{ZERO\_COST}$ **then**
                $\mathbf{w} \leftarrow \mathbf{w} + \phi(c,t_o) - \phi(c,t_p)$
            $t_n \leftarrow \text{CHOOSE\_NEXT}(I,t_p,\text{ZERO\_COST})$
            $c \leftarrow t_n(c)$
return w

function $\text{CHOOSE\_NEXT}_{\text{AMB}}(I,t,\text{ZERO\_COST})$
    if $t \in \text{ZERO\_COST}$ **then**
        return t
    else
        return $\text{RANDOM\_ELEMENT}(\text{ZERO\_COST})$

function $\text{CHOOSE\_NEXT}_{\text{EXP}}(I,t,\text{ZERO\_COST})$
    if $I > k$ and $\text{RAND}() > p$ **then**
        return t
    else
        return $\text{CHOOSE\_NEXT}_{\text{AMB}}(I,t,\text{ZERO\_COST})$

Defining the cost of a transition

- Loss difference between minimum loss trees achievable before and after transition

$$\mathscr{C}(t; c, G_{\text{gold}}) = \left[\min_{G:t(c) \rightsquigarrow G} \mathscr{L}(G, G_{\text{gold}}) \right] - \left[\min_{G:c \rightsquigarrow G} \mathscr{L}(G, G_{\text{gold}}) \right]$$

- Loss for trees nicely decomposes into losses for arcs
  - We can compute transition cost by counting gold arcs that are no longer reachable after transition

Today’s topics
Addressing compounding error

- Improving on gold parse oracle
  - Research highlight: [Goldberg & Nivre, 2012]
- Imitation learning for structured prediction
  - CIML ch 18

Imitation Learning
aka learning by demonstration

- Sequential decision making problem
  - At each point in time t
    - Receive input information x_t
    - Take action a_t
    - Suffer loss l_t
    - Move to next time step until time T
- Goal
  - learn a **policy** function $f(x_t) = y_t$
  - That minimizes expected **total** loss over all **trajectories** enabled by f

$$\boldsymbol{\tau} = \boldsymbol{x}_1 , \underbrace{a_1}_{=f(\boldsymbol{x}_1)} , \ell_1 , \boldsymbol{x}_2 , \underbrace{a_2}_{=f(\boldsymbol{x}_2)} , \ell_2 , \ldots , \boldsymbol{x}_T , \underbrace{a_T}_{=f(\boldsymbol{x}_T)} , \ell_T$$

Supervised Imitation Learning

Algorithm 43 $\textsc{SupervisedImitationTrain}(\mathcal{A}, \tau_1, \tau_2, \dots, \tau_N)$

$D \leftarrow \langle (x, a) \ : \ \forall n \ , \ \forall (x, a, \ell) \in \tau_n \rangle$ // collect all observation/action pairs
return $\mathcal{A}(D)$ // train multiclass classifier on D

Algorithm 44 $\textsc{SupervisedImitationTest}(f)$

---

for $t = 1 \dots T$ **do**
  $x_t \leftarrow$ current observation
  $a_t \leftarrow f(x_t)$ // ask policy to choose an action
  take action a_t
  $\ell_t \leftarrow$ observe instantaneous loss
end for
return $\sum_{t=1}^{T} \ell_t$ // return total loss

Supervised Imitation Learning

Algorithm 43 $\textsc{SupervisedImitationTrain}(\mathcal{A}, \tau_1, \tau_2, \dots, \tau_N)$

$D \leftarrow \langle (x, a) \ : \ \forall n \ , \ \forall (x, a, \ell) \in \tau_n \rangle$ // collect all observation/act pairs

return $\mathcal{A}(D)$ // t in multiclass er on D

Algorithm 44 SupervisedImit

for $t = 1 \dots T$ **do**

  $x_t \leftarrow$ current obse

  $a_t \leftarrow f(x_t)$

  take actio

  $\ell_t \leftarrow$ observe ins

end for

return $\sum_{t=1}^{T} \ell_t$ // return total loss

Problem with
supervised approach:
Compounding error

How can we train system to make better predictions off the expert path?

- We want a policy f that leads to good performance in configurations that f encounters

- A chicken and egg problem
    - Can be addressed by iterative approach

DAGGER: simple & effective imitation learning via Data AGGregation

Algorithm 45 DaggerTrain($\mathcal{A}$, MaxIter, N, expert)

$\langle \boldsymbol{\tau}_n^{(0)} \rangle_{n=1}^{N} \leftarrow$ run the expert N many times

$D_0 \leftarrow \langle (\boldsymbol{x}, a) \;:\; \forall n \,,\, \forall (\boldsymbol{x}, a, \ell) \in \boldsymbol{\tau}_n^{(0)} \rangle$ // collect all pairs (same as supervised)

$f_0 \leftarrow \mathcal{A}(D_0)$ // train initial policy (multiclass classifier) on D_0

for $i = 1 \ldots MaxIter$ **do**

  $\langle \boldsymbol{\tau}_n^{(i)} \rangle_{n=1}^{N} \leftarrow$ run policy f_{i-1} N-many times // trajectories by f_{i-1}

  $D_i \leftarrow \langle (\boldsymbol{x}, \text{expert}(\boldsymbol{x})) \;:\; \forall n \,,\, \forall (\boldsymbol{x}, a, \ell) \in \boldsymbol{\tau}_n^{(i)} \rangle$ // collect data set

  // observations $\boldsymbol{x}$ visited by f_{i-1}

  // but actions according to the expert

  $f_i \leftarrow \mathcal{A}\left(\bigcup_{j=0}^{i} D_j \right)$ // train policy f_i on union of all data so far

end for

return $\langle f_0, f_1, \ldots, f_{\text{MaxIter}} \rangle$ // return collection of all learned policies

Requires interaction with expert!

When is DAGGER used in practice?

- Interaction with expert is not always possible

- Classic use case
  - Expert = slow algorithm
  - Use DAGGER to learn a faster algorithm that imitates expert
  - Example: game playing where expert = brute-force search in simulation mode

- But also structured prediction

Sequence labeling via imitation learning

$$x = \text{“ monsters eat tasty bunnies “}$$

$$y = \quad \text{noun} \quad \text{verb} \quad \text{adj} \quad \text{noun}$$

- What is the “expert” here?
  - Given a loss function (e.g., Hamming loss)
  - Expert takes action that minimizes long-term loss

$$\text{expert}(\ell, y, \hat{y}) = \operatorname*{argmin}_a \text{best}(\ell, y, \hat{y} \circ a)$$

Output prefix at time t

Loss of best reachable output starting with prefix $\hat{y} \circ a$

- When expert can be computed exactly, it is called an **oracle**

- Key advantages
  - Can define features $\phi(x, \hat{y})$
  - No restriction to Markov features

Today's topics

- Improving on gold parse oracle
  - Research highlight: [Goldberg & Nivre, 2012]
- Imitation learning for structured prediction
  - CIML ch 18