Joint prediction via imitation learning

Part of Speech Tagging

NLP algorithms use a kitchen sink of features

Dependency Parsing

*ROOT*

NLP algorithms use a kitchen sink of features
Joint prediction via imitation learning

Joint Prediction Haiku

A joint prediction
Across a single input
Loss measured jointly

*ROOT*

features

use

a

of
An analogy from playing Mario

From Mario AI competition 2009

Input:

Output:
Jump in \{0,1\}
Right in \{0,1\}
Left in \{0,1\}
Speed in \{0,1\}

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

1. Collect trajectories from expert $\pi^*$
   • Trajectory = sequence of state/action pairs over time
   • States are represented as feature vectors
     – Incorporates current “observations” …
     – … and any past decisions

2. Store as dataset $D = \{ (s, \pi^*(s)) | s \sim \pi^* \}$

3. Train classifier $\pi$ on $D$
   • Let $\pi$ play the game!
Training (expert)

Sample Expert Trajectories

Video credit: Stéphane Ross, Geoff Gordon and Drew Bagnell
Test-time execution (classifier)

Video credit: Stéphane Ross, Geoff Gordon and Drew Bagnell
What's the (biggest) failure mode?

- The expert never gets stuck next to pipes
- => Classifier doesn't learn to recover!
Imitation learning: DAgger

1. Collect trajectories from expert $\pi^*$
2. Dataset $D_0 = \{ (s, \pi^*(s)) \mid s \sim \pi^* \}$
3. Train $\pi_1$ on $D_0$
4. Collect new trajectories from $\pi_1$
   - But let the expert steer!
5. Dataset $D_1 = \{ (s, \pi^*(s)) \mid s \sim \pi_1 \}$
6. Train $\pi_2$ on $D_0 \cup D_1$

● In general:
  ● $D_n = \{ (s, \pi^*(s)) \mid s \sim \pi_n \}$
  ● Train $\pi_n$ on $\bigcup_{i<n} D_i$

If $N = T \log T$, $L(\pi_n) < T \varepsilon_N + O(1)$ for some $n$
Test-time execution (DAgger)

Video credit: Stéphane Ross, Geoff Gordon and Drew Bagnell
What's the biggest failure mode?

- Classifier only sees “right” versus “not-right”
  - No notion of “better” or “worse”
  - No “partial credit”
  - Must have a single “target” answer
Joint prediction via learning to search

Part of Speech Tagging

NLP algorithms use a kitchen sink of features

Dependency Parsing

*ROOT*

NLP algorithms use a kitchen sink of features
Learning to search

1. Generate an initial trajectory using a rollin policy

2. Foreach state $R$ on that trajectory:
   a) Foreach possible action $a$ (one-step deviations)
      i. Take that action
      ii. Complete this trajectory using a rollout policy
      iii. Obtain a final loss
   b) Generate a cost-sensitive classification example:
      $\left( \Phi(R), \langle c_a \rangle_{a \in A} \right)$
Choosing the rollin/rollout policies

- Three basic options:
  - The currently learned policy ("learn")
  - The reference/expert policy ("ref")
  - A stochastic mixture of these ("mix")

<table>
<thead>
<tr>
<th></th>
<th>Out</th>
<th>Ref</th>
<th>Mix</th>
<th>Learn</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ref</td>
<td>Inconsistent One-step fail</td>
<td>Inconsistent</td>
<td>Inconsistent</td>
<td></td>
</tr>
<tr>
<td>Learn</td>
<td>One-step fail</td>
<td>Good</td>
<td>Really hard</td>
<td></td>
</tr>
</tbody>
</table>

Note: if the reference policy is *optimal* then: In=Learn & Out=Ref is also a good choice.

Sanity check: which of these is closest to DAgger?
The oracle (reference) policy gives the true label for the corresponding word.

Sanity check: why/when is this optimal?

```python
def _run(self, sentence):
    out = []
    for n in range(len(sentence)):
        pos, word = sentence[n]
        ex = example({'w': word})
        pred = predict(ex, pos)
        out.append(pred)
    loss( # of pred != pos )
    return out
```
Optimal policies

- Given:
  - Training input $x$
  - State $R$
  - Loss function

- Return the action $a$ that:
  - (If all future actions are taken optimally)
  - Minimizes the corresponding loss
Optimal policies for harder problems

- Consider word-based machine translation
- You want to write
- But what does the optimal policy do?

F: Marie programme l'ordinateur
E: Mary programs the computer

State R: Mary ______
State R': The computer ______
State R'': Aardvarks ______

\[
\begin{align*}
F, \text{ref} &= \text{input} \\
E &= [ <s> ] \\
i &= 1 \\
cov &= {} \\
\text{while } |cov| \neq |F|:\n  a &= \text{predict}(cov, ???) \\
  e &= \text{predict}(F_a, ???) \\
  cov[a] &= \text{true} \\
  E\text{.push}(e) \\
i &= i + 1 \\
\text{loss}(1 - \text{BLEU}(E, \text{ref})) \\
\text{return } E
\end{align*}
\]
How can you do this for Mario?

Input:

Output:

Jump in \{0,1\}
Right in \{0,1\}
Left in \{0,1\}
Speed in \{0,1\}

Reference policy is constructed on-the-fly:
At each state, execute a depth-4 BFS
At each of the 64k leaves, evaluate
Choose initial action that leads to local optimum
Key concepts and commentary

- Rollin / rollout / one-step deviations
- Reference policy / optimal policy
- Joint loss

Tips:
- Defining a good reference can be tricky:
  - If optimal, do: in=learn, out=ref|none
  - If suboptimal, do: in=learn, out=mix
- Can only learn to avoid compounding errors given the right features
Coming up next....

- Instantiating these ideas in vw
- During the break, please:
  
git clone git@github.com:JohnLangford/vowpal_wabbit.git
make
make python
cd python
python test.py
python test_search.py

- And ask us any questions you might have!
- When we return, we'll build some predictors!
A short reading list

- **DAgger (imitation learning from oracle):**
  A reduction of imitation learning and structured prediction to no-regret online learning
  Ross, Gordon & Bagnell, *AIStats 2011*

- **AggreVaTe (roughly “DAgger with rollouts”)**
  Reinforcement and imitation learning via interactive no-regret learning
  Ross & Bagnell, *arXiv:1406.5979*

- **LOLS (analysis of rollin/rollout, lower bounds, suboptimal reference)**
  Learning to search better than your teacher
  Chang, Krishnamurthy, Agarwal, Daumé III & Langford, *ICML 2015*

- **Imperative learning to search (programming framework, sequence labeling results)**
  Efficient programmable learning to search
  Chang, Daumé III, Langford & Ross, *arXiv:1406.1837*

- **State of the art dependency parsing in ~300 lines of code**
  Learning to search for dependencies
  Chang, He, Daumé III & Langford, *arXiv:1503.05615*

- **Efficiently computing an optimal policy for shift-reduce dependency parsing**
  A tabular method for dynamic oracles in transition-based parsing
  Goldberg, Sartorio & Satta, *TAACL 2014*