Advances in Structured Prediction

John Langford
Microsoft Research
jl@hunch.net

Hal Daumé III
U Maryland
me@hal3.name
Examples of structured prediction
Sequence labeling

x = the monster ate the sandwich
y = Dt Nn Vb Dt Nn

x = Yesterday I traveled to Lille
y = - PER - - LOC
NLP algorithms use a kitchen sink of features
(Bipartite) matching

- What is the anticipated cost of collecting fees under the new proposal?
- En vertu de les nouvelles propositions, quel est le coût prévu de la perception des droits?
Machine translation

Moscow stressed tone against Iran on its nuclear program. He called Russian Foreign Minister Tehran to take concrete steps to restore confidence with the international community, to cooperate fully with the IAEA. Conversely Tehran expressed its willingness.
Image segmentation
Protein secondary structure prediction
Standard solution methods

1. Each prediction is independent
2. Shared parameters via “multitask learning”
3. Assume tractable graphical model; optimize
4. Hand-crafted
Predicting independently

- $h : \text{features of nearby voxels} \rightarrow \text{class}$
- Ensure output is coherent at test time

- ✔ Very simple to implement, often efficient
- ✗ Cannot capture correlations between predictions
- ✗ Cannot optimize a joint loss
Prediction with multitask bias

- $h: \text{features} \rightarrow \text{(hidden representation)} \rightarrow \text{yes/no}$
- Share (hidden representation) across all classes

- All advantages of predicting independently
- May implicitly capture correlations

- Learning may be hard (… or not?)
- Still not optimizing a joint loss
Optimizing graphical models

- Encode output as a graphical model
- Learn parameters of that model to maximize:
  - $p(\text{true labels} \mid \text{input})$
  - $\text{cvx u.b. on loss(\text{true labels, predicted labels})}$

- Guaranteed consistent outputs
- Can capture correlations explicitly

- Assumed independence assumptions may not hold
- Computationally intractable with too many “edges” or non-decomposable loss function
Back to the original problem...

- How to optimize a discrete, joint loss?

- **Input:** $x \in X$
- **Truth:** $y \in Y(x)$
- **Outputs:** $Y(x)$
- **Predicted:** $\hat{y} \in Y(x)$
- **Loss:** $\text{loss}(y, \hat{y})$
- **Data:** $(x,y) \sim D$
Back to the original problem...

- How to optimize a discrete, joint loss?

- Input: \( x \in X \)
- Truth: \( y \in Y(x) \)
- Outputs: \( Y(x) \)
- Predicted: \( \hat{y} \in Y(x) \)
- Loss: \( \text{loss}(y, \hat{y}) \)
- Data: \( (x, y) \sim D \)

Goal:

find \( h \in H \)
such that \( h(x) \in Y(x) \)
minimizing

\[
E_{(x,y) \sim D} \left[ \text{loss}(y, h(x)) \right]
\]

based on \( N \) samples

\[
(x_n, y_n) \sim D
\]
Challenges

• Output space is too big to exhaustively search:
  • Typically exponential in size of input
  • implies \( y \) must decompose in some way
    (often: \( x \) has many pieces to label)

• Loss function has combinatorial structure:
  - Intersection over union
  - Edit Distance
Decomposition of label

- Decomposition of $y$ often implies an ordering

```
I can can a can
Pro Md Vb Dt Nn
```

- But sometimes not so obvious....

(we'll come back to this case later....)
Search spaces

- When $y$ decomposes in an ordered manner, a sequential decision making process emerges.
Search spaces

- When $y$ decomposes in an ordered manner, a sequential decision making process emerges.

Encodes an output $\hat{y} = \hat{y}(e)$ from which $\text{loss}(y, \hat{y})$ can be computed (at training time).
Policies

- A policy maps observations to actions

$$\pi(\text{obs.}, x, t, \tau, \ldots \text{anything else}) = a$$
Versus reinforcement learning

In learning to search (L2S):

- *Labeled data* at training time
  - can construct good/optimal policies
- Can “reset” and try the same example many times

Goal:

\[
\min_{\pi} \mathbb{E} [ \text{loss}(\pi) ]
\]
Labeled data → Reference policy

Given partial traj. $a_1, a_2, \ldots, a_{t-1}$ and true label $y$

The minimum achievable loss is:

$$\min \text{ loss}(y, \hat{y}(\tilde{a}))$$

$(a_t, a_{t+1}, \ldots)$

The optimal action is the corresponding $a_t$

The optimal policy is the policy that always selects the optimal action
Ingredients for learning to search

- Training data: \((x_n, y_n) \sim D\)
- Output space: \(Y(x)\)
- Loss function: \(loss(y, \hat{y})\)
- Decomposition: \(\{o\}, \{a\}, \ldots\)
- Reference policy: \(\pi^\text{ref}(o, y)\)
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
Training (expert)

Sample Expert Trajectories

Video credit: Stéphane Ross, Geoff Gordon and Drew Bagnell
Warm-up: Supervised learning

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

- Let $\pi$ play the game!
Test-time execution (sup. learning)

Supervised Approach after 100K Training Samples
What's the (biggest) failure mode?

The expert never gets stuck next to pipes

$\implies$ Classifier doesn't learn to recover!
Warm-up II: Imitation learning

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

● In general:
   ● $D_n = \{ (o, \pi^{ref}(o,y)) | o \sim \pi_n \}$
   ● Train $\pi_{n+1}$ on $\bigcup_{i \leq n} D_i$

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

Classifier: \( h : x \rightarrow [K] \)

Multiclass classification

- Data: \((x,y) \in X \times [K]\)
- Goal: \(\min_h \Pr(h(x) \neq y)\)

Cost-sensitive classification

- Data: \((x,c) \in X \times [0,\infty)^K\)
- Goal: \(\min_h E_{(x,c)} [ c_{h(x)} ]\)
Learning to search: AggraVaTe

1. Let learned policy $\pi$ drive for $t$ timesteps to obs. $o$

2. For each possible action $a$:
   - Take action $a$, and let expert $\pi^{\text{ref}}$ drive the rest
   - Record the overall loss, $c_a$

3. Update $\pi$ based on example:
   $$(o, \langle c_1, c_2, \ldots, c_K \rangle)$$

4. Goto (1)
Learning to search: AggraVaTe

1. Generate an initial trajectory using the current policy

2. Foreach decision on that trajectory with obs. o:
   a) Foreach possible action $a$ (one-step deviations)
      i. Take that action
      ii. Complete this trajectory using reference policy
      iii. Obtain a final loss, $c_a$
   b) Generate a cost-sensitive classification example: $(o, \bar{c})$
Learning to search: AggraVaTe

1. Generate an initial trajectory using the current policy

2. Foreach decision on that trajectory with obs. \( o \):
   a) Foreach possible action \( a \) (one-step deviations)
      i. Take that action
      ii. Complete this trajectory using reference policy
      iii. Obtain a final loss, \( c_a \)
   b) Generate a cost-sensitive classification example:
      \( (o, \bar{c}) \)

Often it's possible to analytically compute this loss without having to execute a roll-out!
Example I: Sequence labeling

- Receive input:
  \[ x = \text{the monster ate the sandwich} \]
  \[ y = \text{Dt Nn Vb Dt Nn} \]

- Make a sequence of predictions:
  \[ x = \text{the monster ate the sandwich} \]
  \[ \hat{y} = \text{Dt Dt Dt Dt Dt Dt Dt} \]

- Pick a timestep and try all perturbations there:
  \[ x = \text{the monster ate the sandwich} \]
  \[ \hat{y}_{\text{Dt}} = \text{Dt Dt} \]
  \[ \hat{y}_{\text{Nn}} = \text{Dt Nn} \]
  \[ \hat{y}_{\text{Vb}} = \text{Dt Vb} \]

- Compute losses and construct example:
  \[ ( \{ w=\text{monster}, p=\text{Dt}, \ldots \} , [1,0,1] ) \]
Example II: Graph labeling

- **Task:** label nodes of a graph given node features (and possibly edge features)
- **Example:** WebKB webpage labeling
  - **Node features:** text on web page
  - **Edge features:** text in hyperlinks
Example II: Graph labeling

- How to linearize? *Like belief propagation might!*
- Pick a starting node (A), run BFS out
- Alternate outward and inward passes

Linearization:
- ABCDEFGHI
- HGFEDCBA
- BCDEFGHI
- HGFEDCBA
- ...
Example II: Graph labeling

1. Pick a node (= timestep)
2. Construct example based on neighbors' labels
3. Perturb current node's label
Outline

1. Empirics
2. Analysis
3. Programming
4. Others and Issues
What part of speech are the words?

POS Tagging (tuned hps)

Accuracy (per word)

Training time (minutes)

1s 10s 1m 10m 30m 1h

POS Tagging (tuned hps)

OAA
L2S
L2S (ft)
CRFsgd
CRF++
StrPerc
StrSVM
StrSVM2
A demonstration

1 | w Despite
2 | w continuing
3 | w problems
1 | w in
4 | w its
5 | w newsprint
5 | w business

...
A demonstration

<table>
<thead>
<tr>
<th></th>
<th>w Despite</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>w continuing</td>
</tr>
<tr>
<td>3</td>
<td>w problems</td>
</tr>
<tr>
<td>1</td>
<td>w in</td>
</tr>
<tr>
<td>4</td>
<td>w its</td>
</tr>
<tr>
<td>5</td>
<td>w newsprint</td>
</tr>
<tr>
<td>5</td>
<td>w business</td>
</tr>
</tbody>
</table>

... 

```
vw -b 24 -d wsj.train.vw -c --search_task sequence --search 45 --search_alpha 1e-8 --search_neighbor_features -1:w,1:w --affix -1w,+1w -f foo.reg
vw -t -i foo.reg wsj.test.vw```


Is this word a name or not?

Named Entity Recognition (tuned hps)

- OAA
- L2S
- L2S (ft)
- CRFsgd
- CRF++
- StrPerc
- StrSVM2

F-score (per entity)
- 73.6
- 79.8
- 79.2
- 76.5
- 75.9
- 76.5
- 78.3

Training time (minutes)
- 0.60
- 0.65
- 0.70
- 0.75
- 0.80

Training time (minutes) vs. F-score (per entity)
How fast in evaluation?

**Prediction (test-time) Speed**

- **POSS**
  - L2S: 13
  - L2S (ft): 5.7
  - CRFsgd: 5.3
  - CRF++: 5.6
  - StrPerc: 14
  - StrSVM: 24
  - StrSVM2: 98

- **NER**
  - L2S: 365
  - L2S (ft): 404
  - CRFsgd: 13
  - CRF++: 5.7
  - StrPerc: 14
  - StrSVM: 24
  - StrSVM2: 98

**Thousands of Tokens per Second**
**Goal:** find the Entities and then find their Relations

<table>
<thead>
<tr>
<th>Method</th>
<th>Entity F1</th>
<th>Relation F1</th>
<th>Train Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured SVM</td>
<td>88.00</td>
<td>50.04</td>
<td>300 seconds</td>
</tr>
<tr>
<td>L2S</td>
<td>92.51</td>
<td>52.03</td>
<td>13 seconds</td>
</tr>
</tbody>
</table>

L2S uses ~100 LOC.
Find dependency structure of sentences.

L2S uses ~300 LOC.
Outline

1. Empirics
2. Analysis
3. Programming
4. Others and Issues
### Effect of Roll-in and Roll-out Policies

<table>
<thead>
<tr>
<th>Roll-out →</th>
<th>Reference</th>
<th>Half-n-half</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ Roll-in</td>
<td></td>
<td>Inconsistent</td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learned</td>
<td>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**

- States: $s_1, s_2, s_3, s_4$
- Actions: $a, b, c, d, e, f$
- Transitions:
  - $s_1 \xrightarrow{a} s_2$
  - $s_2 \xrightarrow{b} s_3$
  - $s_2 \xrightarrow{c} e_1, 0$
  - $s_3 \xrightarrow{e} e_3, 100$
  - $s_3 \xrightarrow{f} e_4, 0$
  - $s_2 \xrightarrow{d} e_2, 10$
### Effect of Roll-in and Roll-out Policies

<table>
<thead>
<tr>
<th>roll-out $\rightarrow$</th>
<th>Reference</th>
<th>Half-n-half</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\downarrow$ roll-in</td>
<td>Reference</td>
<td>Inconsistent</td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learned</td>
<td>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Theorem**

Roll-in with ref:

$0$ cost-sensitive regret $\Rightarrow$ unbounded joint regret
Effect of Roll-in and Roll-out Policies

<table>
<thead>
<tr>
<th>roll-out →</th>
<th>Reference</th>
<th>Half-n-half</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ roll-in</td>
<td>Inconsistent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Consistent</td>
<td>No local opt</td>
<td></td>
</tr>
<tr>
<td>Learned</td>
<td>Consistent</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
    s_1 & \quad a \quad s_2 \\
    & \quad b \quad s_3 \\
    c & \quad e_1, 1 \\
    d & \quad e_2, 1-\epsilon \\
    c & \quad e_3, 1+\epsilon \\
    d & \quad e_4, 0
\end{align*}
\]
## Effect of Roll-in and Roll-out Policies

<table>
<thead>
<tr>
<th>roll-out $\rightarrow$</th>
<th>Reference</th>
<th>Half-n-half</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\downarrow$ roll-in</td>
<td></td>
<td>Inconsistent</td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learned</td>
<td>Consistent</td>
<td>No local opt</td>
<td></td>
</tr>
</tbody>
</table>

### Theorem

*Roll-out with Ref:*

0 cost-sensitive regret $\Rightarrow$ 0 joint regret

*(but not local optimality)*
## Effect of Roll-in and Roll-out Policies

<table>
<thead>
<tr>
<th>roll-out →</th>
<th>Reference</th>
<th>Half-n-half</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ roll-in</td>
<td>Reference</td>
<td>Inconsistent</td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Consistent</td>
<td>No local opt</td>
<td>Reinf. L.</td>
</tr>
<tr>
<td>Learned</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Theorem

*Ignore Ref:*

⇒ *Equivalent to reinforcement learning.*
### Effect of Roll-in and Roll-out Policies

<table>
<thead>
<tr>
<th>Roll-out →</th>
<th>Reference</th>
<th>Half-n-half</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ Roll-in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td>Inconsistent</td>
<td></td>
</tr>
<tr>
<td>Learned</td>
<td>Consistent</td>
<td>Consistent</td>
<td>Reinf. L.</td>
</tr>
<tr>
<td></td>
<td>No local opt</td>
<td>Local Opt</td>
<td></td>
</tr>
</tbody>
</table>

**Theorem**

*Roll-out with \( p = 0.5 \) Ref and \( p = 0.5 \) Learned: 0 cost-sensitive regret \( \Rightarrow \) 0 joint regret + locally optimal*

See LOLS paper, Wednesday 11:20 Van Gogh
AggreVaTe Regret Decomposition

$\pi^{\text{ref}}$ = reference policy

$\bar{\pi}$ = stochastic average learned policy

$J(\pi) = \text{expected loss of } \pi$.

**Theorem**

\[
J(\bar{\pi}) - J(\pi^{\text{ref}}) \leq \sum_{t=1}^{T} E_{D_t^{\hat{\pi}_n}}\left[ Q_{\pi^{\text{ref}}}(x, \hat{\pi}_n) - Q_{\pi^{\text{ref}}}(x, \pi^{\text{ref}}) \right]
\]

$T$ = number of steps

$\hat{\pi}_n$ = $n$th learned policy

$D_t^{\hat{\pi}_n}$ = distribution over $x$ at time $t$ induced by $\hat{\pi}_n$
\[ \pi^{\text{ref}} = \text{reference policy} \]
\[ \bar{\pi} = \text{stochastic average learned policy} \]
\[ J(\pi) = \text{expected loss of } \pi. \]

**Theorem**

\[ J(\bar{\pi}) - J(\pi^{\text{ref}}) \leq T \mathbb{E}_{n,t} \mathbb{E}_{x \sim D^t_{\hat{\pi}_n}} \left[ Q^{\pi^{\text{ref}}}(x, \hat{\pi}_n) - Q^{\pi^{\text{ref}}}(x, \pi^{\text{ref}}) \right] \]

\[ T = \text{number of steps} \]
\[ \hat{\pi}_n = n\text{th learned policy} \]
\[ D^t_{\hat{\pi}_n} = \text{distribution over } x \text{ at time } t \text{ induced by } \hat{\pi}_n \]
\[ Q^{\pi}(x, \pi') = \text{loss of } \pi' \text{ at } x \text{ then } \pi \text{ to finish} \]
Proof

For all $\pi$ let $\pi^t$ play $\pi$ for rounds $1...t$ then play $\pi^{ref}$ for rounds $t + 1...T$. So $\pi^T = \pi$ and $\pi^0 = \pi^{ref}$
Proof

For all $\pi$ let $\pi^t$ play $\pi$ for rounds $1\ldots t$ then play $\pi^\text{ref}$ for rounds $t + 1\ldots T$. So $\pi^T = \pi$ and $\pi^0 = \pi^\text{ref}$

$$J(\pi) - J(\pi^\text{ref}) = \sum_{t=1}^{T} J(\pi^t) - J(\pi^{t-1}) \text{ (Telescoping sum)}$$
Proof

For all $\pi$ let $\pi^t$ play $\pi$ for rounds 1...$t$ then play $\pi^\text{ref}$ for rounds $t+1$...$T$. So $\pi^T = \pi$ and $\pi^0 = \pi^\text{ref}$

$J(\pi) - J(\pi^\text{ref})$

$= \sum_{t=1}^{T} J(\pi^t) - J(\pi^{t-1})$ (Telescoping sum)

$= \sum_{t=1}^{T} \mathbb{E}_{x \sim D^t_\pi} \left[ Q^\text{ref}_{\pi}(x, \pi) - Q^\text{ref}_{\pi}(x, \pi^\text{ref}) \right]$

since for all $\pi, t$, $J(\pi) = \mathbb{E}_{x \sim D^t_\pi} Q^\pi(x, \pi)$
Proof

For all $\pi$ let $\pi^t$ play $\pi$ for rounds $1...t$ then play $\pi^{ref}$ for rounds $t + 1...T$. So $\pi^T = \pi$ and $\pi^0 = \pi^{ref}$

$$J(\pi) - J(\pi^{ref})$$

$$= \sum_{t=1}^{T} J(\pi^t) - J(\pi^{t-1}) \text{ (Telescoping sum)}$$

$$= \sum_{t=1}^{T} \mathbb{E}_{x \sim D_{\pi}^t} \left[ Q^{\pi^{ref}}(x, \pi) - Q^{\pi^{ref}}(x, \pi^{ref}) \right]$$

since for all $\pi$, $t$, $J(\pi) = \mathbb{E}_{x \sim D_{\pi}^t} Q^{\pi}(x, \pi)$

$$= T \mathbb{E}_{t} \mathbb{E}_{x \sim D_{\pi}^t} \left[ Q^{\pi^{ref}}(x, \pi) - Q^{\pi^{ref}}(x, \pi^{ref}) \right]$$
Proof

For all $\pi$ let $\pi^t$ play $\pi$ for rounds $1...t$ then play $\pi^{ref}$ for rounds $t + 1...T$. So $\pi^T = \pi$ and $\pi^0 = \pi^{ref}$

$$J(\pi) - J(\pi^{ref})$$

$$= \sum_{t=1}^{T} J(\pi^t) - J(\pi^{t-1}) \quad \text{(Telescoping sum)}$$

$$= \sum_{t=1}^{T} \mathbb{E}_{x \sim D^t_\pi} \left[ Q^{\pi^{ref}}(x, \pi) - Q^{\pi^{ref}}(x, \pi^{ref}) \right]$$

since for all $\pi, t$, $J(\pi) = \mathbb{E}_{x \sim D^t_\pi} Q^\pi(x, \pi)$

$$= T \mathbb{E}_t \mathbb{E}_{x \sim D^t_\pi} \left[ Q^{\pi^{ref}}(x, \pi) - Q^{\pi^{ref}}(x, \pi^{ref}) \right]$$

So $J(\bar{\pi}) - J(\pi^{ref})$

$$= T \mathbb{E}_{t,n} \mathbb{E}_{x \sim D^t_{\bar{\pi}n}} \left[ Q^{\pi^{ref}}(x, \hat{\pi}_n) - Q^{\pi^{ref}}(x, \pi^{ref}) \right]$$
Outline

1 Empirics
2 Analysis
3 Programming
4 Others and Issues
Lines of Code

- CRFSGD
- CRF++
- S-SVM
- Search

End of document.
How?

Sequential_RUN(examples)

1: for $i = 1$ to len(examples) do
2:   prediction ← predict(examples[$i$], examples[$i$].label)
3:   loss(prediction ≠ examples[$i$].label)
4:   end for
Sequential_RUN(examples)

1: for $i = 1$ to len(examples) do
2:   prediction ← predict(examples[$i$], examples[$i$].label)
3:   loss(prediction \neq examples[$i$].label)
4: end for

Decoder + loss + reference advice
RunParser($sentence$)

1: $stack \ S \leftarrow \{ \text{Root} \} \$
2: $buffer \ B \leftarrow \left[ \text{words in sentence} \right]$
3: $arcs \ A \leftarrow \emptyset$
4: \textbf{while} $B \neq \emptyset$ or $|S| \geq 1$ do
5: \hspace{1em} $ValidActs \leftarrow \text{GetValidActions}(S, B)$
6: \hspace{1em} $features \leftarrow \text{GetFeat}(S, B, A)$
7: \hspace{1em} $ref \leftarrow \text{GetGoldAction}(S, B)$
8: \hspace{1em} $action \leftarrow \text{predict}(features, ref, ValidActs)$
9: \hspace{1em} $S, B, A \leftarrow \text{Transition}(S, B, A, action)$
10: \textbf{end while}
11: $loss(A[w] \neq A^*[w], \forall w \in \text{sentence})$
12: \textbf{return} output
Theorem: Every algorithm which:

1. Always terminates.
2. Takes as input relevant feature information $X$.
3. Make $0+$ calls to $\text{predict}$.
4. Reports $\text{loss}$ on termination.

defines a search space, and such an algorithm exists for every search space.
def _run(self, sentence):
    output = []
    for n in range(len(sentence)):
        pos, word = sentence[n]
        with self.vw.example('w': [word],
                           'p': [prev_word]) as ex:
            pred = self.sch.predict(examples=ex,
                                     my_tag=n+1, oracle=pos,
                                     condition=[(n,'p'), (n-1, 'q')])
            output.append(pred)
    return output
Bugs you cannot have

1. Never train/test mismatch.
Bugs you cannot have

1. Never train/test mismatch.
2. Never unexplained slow.
Bugs you cannot have

1. Never train/test mismatch.
2. Never unexplained slow.
3. Never fail to compensate for cascading failure.
Outline

1. Empirics
2. Analysis
3. Programming
4. Others and Issues
   1. Families of algorithms.
   2. What’s missing from learning to search?
Imitation Learning

Use perceptron-like update when learned deviates from gold standard.

LaSo Daume III & Marcu, ICML 2005.
Local Liang et al, ACL 2006.
Beam P. Xu et al., JMLR 2009.
Inexact Huang et al, NAACL 2012.
Imitation Learning

Use perceptron-like update when learned deviates from gold standard.

LaSo Daume III & Marcu, ICML 2005.
Local Liang et al, ACL 2006.
Beam P. Xu et al., JMLR 2009.
Inexact Huang et al, NAACL 2012.

Train a classifier to mimic an expert’s behavior

DAgger Ross et al., AIStats 2011.
Dyna O Goldberg et al., TACL 2014.
Learning to Search

When the reference policy is optimal

**Searc**h Daume III et al., MLJ 2009.

**Aggra** Ross & Bagnell,

Learning to Search

When the reference policy is optimal

**Sear**n  Daume III et al., MLJ 2009.

**Aggra**  Ross & Bagnell,


When it’s not

**LOLS**  Chang et al., ICML 2015.
Learning to Search

When the reference policy is optimal

Searn  Daume III et al., MLJ 2009.
Aggra  Ross & Bagnell,

When it’s not

LOLS  Chang et al., ICML 2015.

Code in Vowpal Wabbit http://hunch.net/~vw
Inverse Reinforcement Learning

Given observed expert behavior, infer the underlying reward function the expert seems to be optimizing.
Inverse Reinforcement Learning

Given observed expert behavior, infer the underlying reward function the expert seems to be optimizing

propose Kalman, 1968.

Inverse Reinforcement Learning

Given observed expert behavior, infer the underlying reward function the expert seems to be optimizing.

- propose Kalman, 1968.

from sample trajectories only
Ng & Russell, ICML 2000
Inverse Reinforcement Learning

Given observed expert behavior, infer the underlying reward function the expert seems to be optimizing.

Propose Kalman, 1968.


From sample trajectories only
Ng & Russell, ICML 2000

For apprenticeship learning

Apprent. Abbeel & Ng, ICML 2004

Maxmar. Ratliff et al., NIPS 2005

MaxEnt Ziebart et al., AAAI 2008
Learning to search \( \sim \) dependency + search order. Graphical models “work” given dependencies only.
What’s missing? The reference policy

A good reference policy is often nonobvious... yet critical to performance.
When choosing 1-of-$k$ things, $O(k)$ time is not exciting for machine translation.
What's missing? GPU fun

Vision often requires a GPU. Can that be done?
How to optimize discrete joint loss?
How to optimize discrete joint loss?

1. Programming complexity.
Programming complexity. Most complex problems addressed independently—too complex to do otherwise.
How to optimize discrete joint loss?

2. Prediction accuracy. It had better work well.
How to optimize discrete joint loss?


2. Prediction accuracy. It had better work well.

3. Train speed. Debug/development productivity + maximum data input.
How to optimize discrete joint loss?

1. **Programming complexity.** Most complex problems addressed independently—too complex to do otherwise.

2. **Prediction accuracy.** It had better work well.

3. **Train speed.** Debug/development productivity + maximum data input.

4. **Test speed.** Application efficiency