Advances in Structured Prediction

John Langford
Microsoft Research
jl@hunch.net

Hal Daumé III
U Maryland
me@hal3.name

Examples of ~~structured~~ joint prediction

Sequence labeling

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

x = Yesterday I traveled to Lille
y = - PER - - LOC

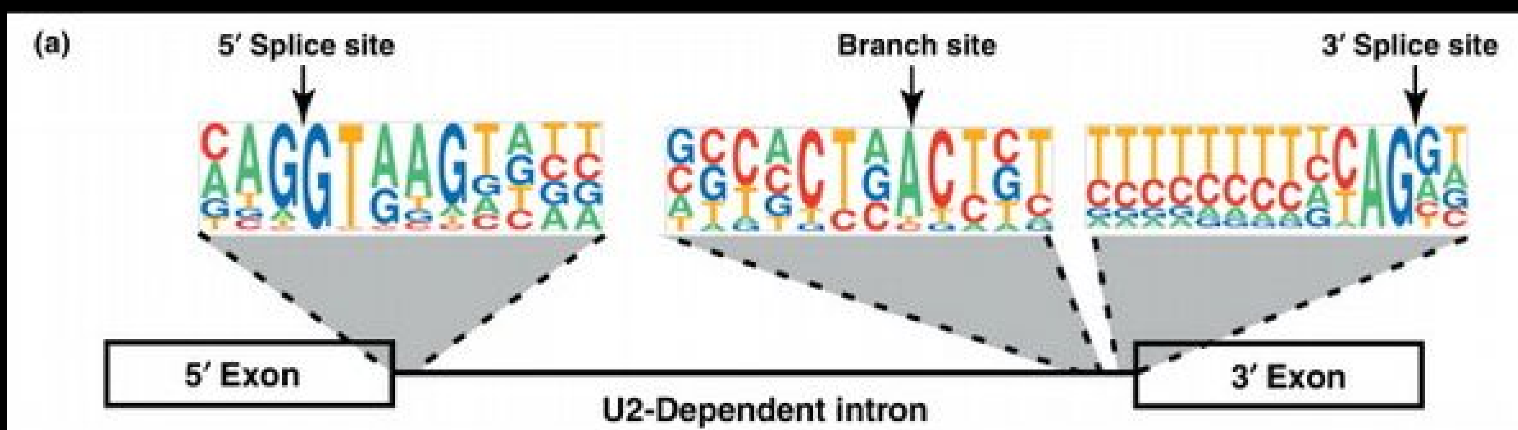
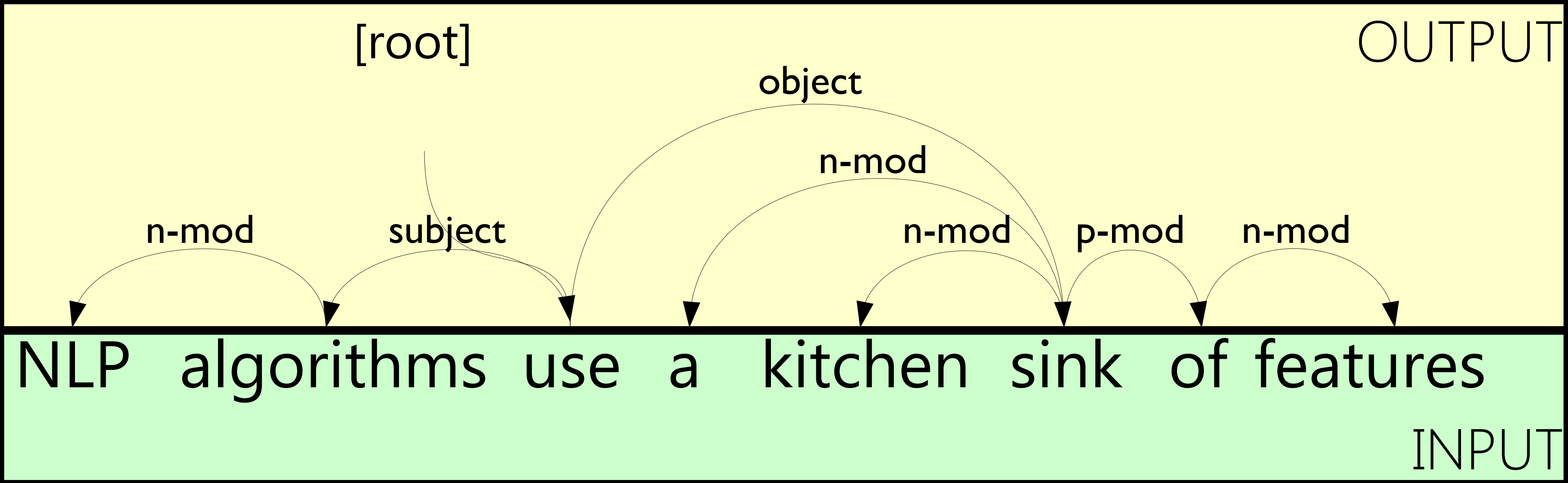


image credit: Richard Padgett

Natural language parsing



(Bipartite) matching

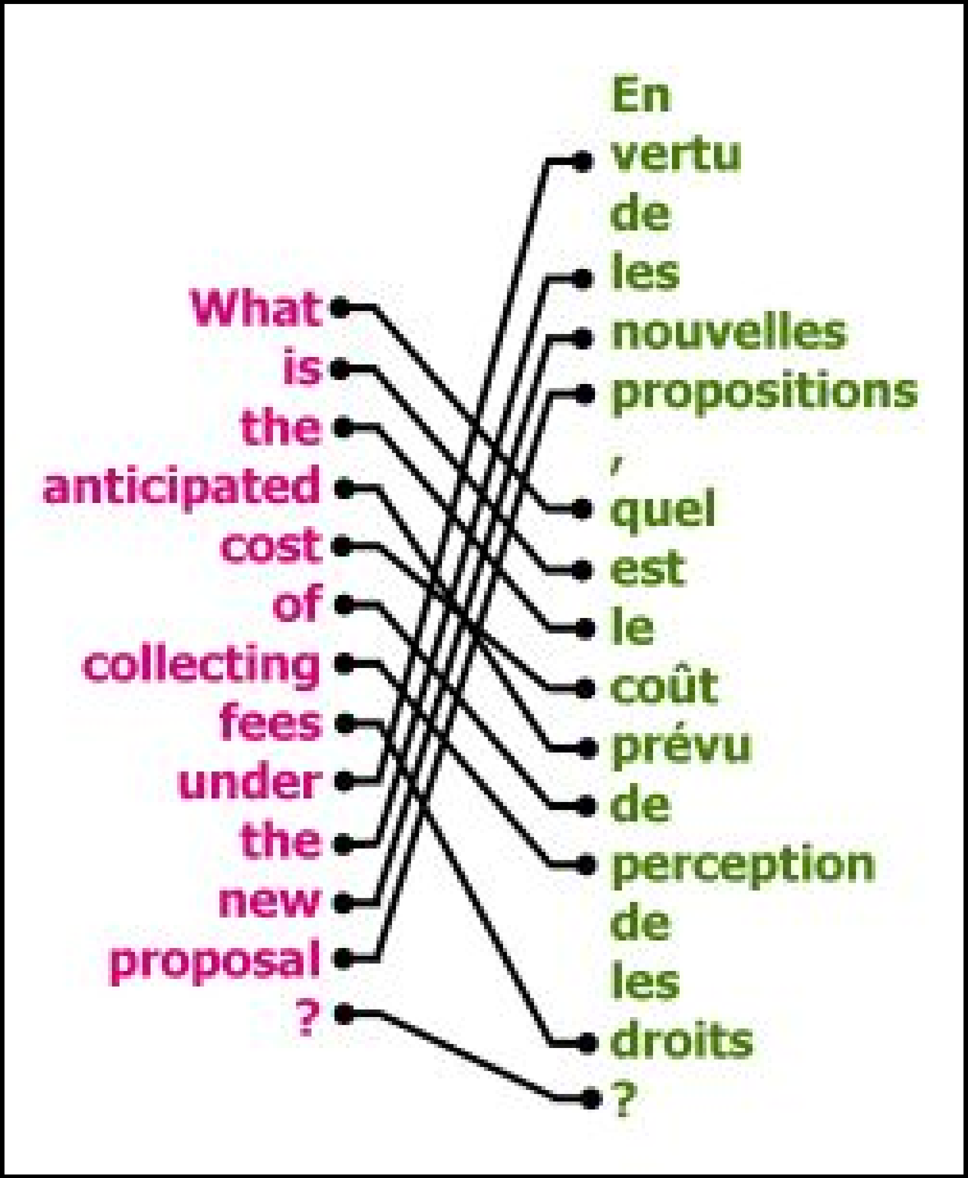


image credit: Ben Taskar

Machine translation

Image segmentation

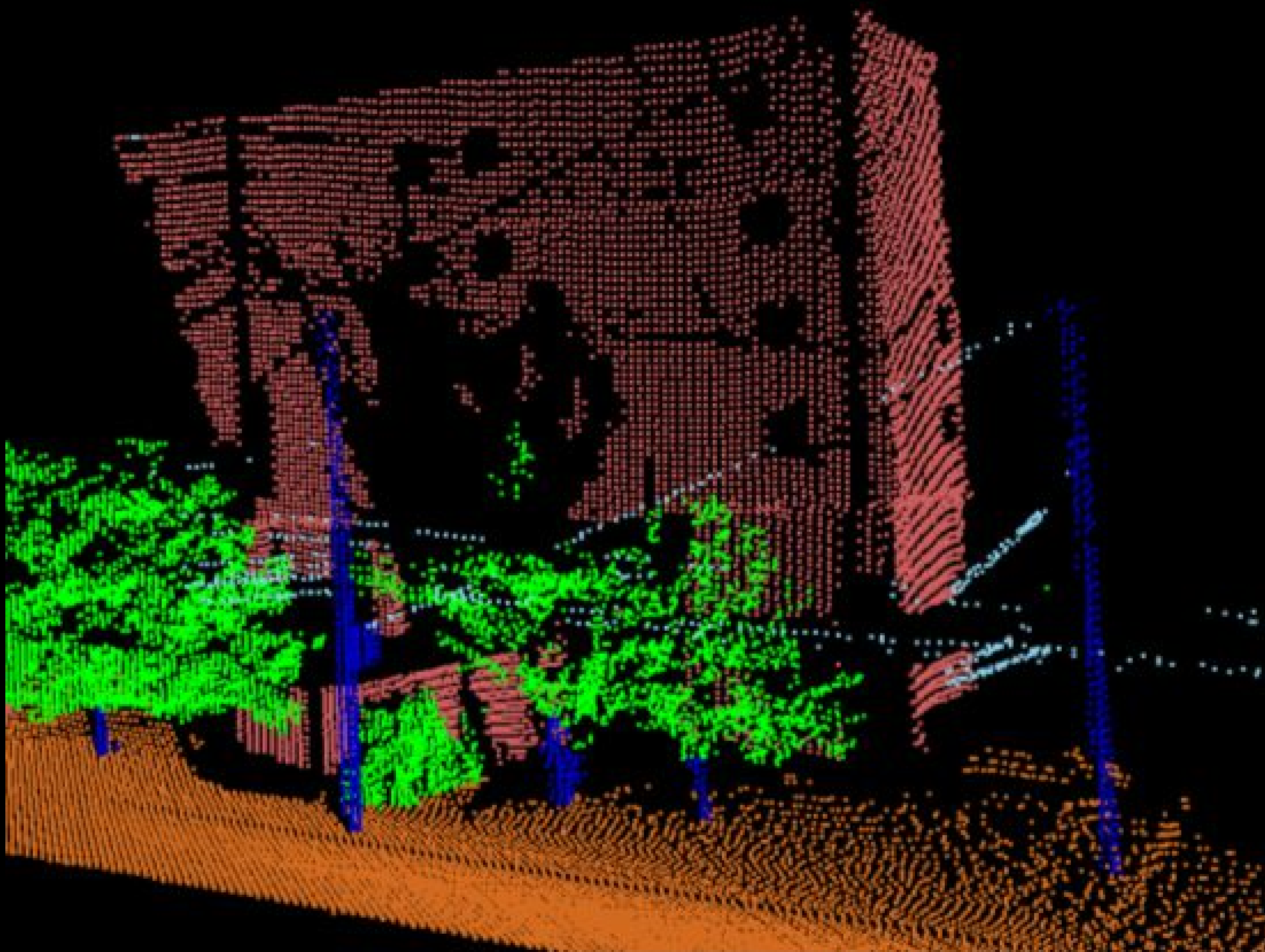
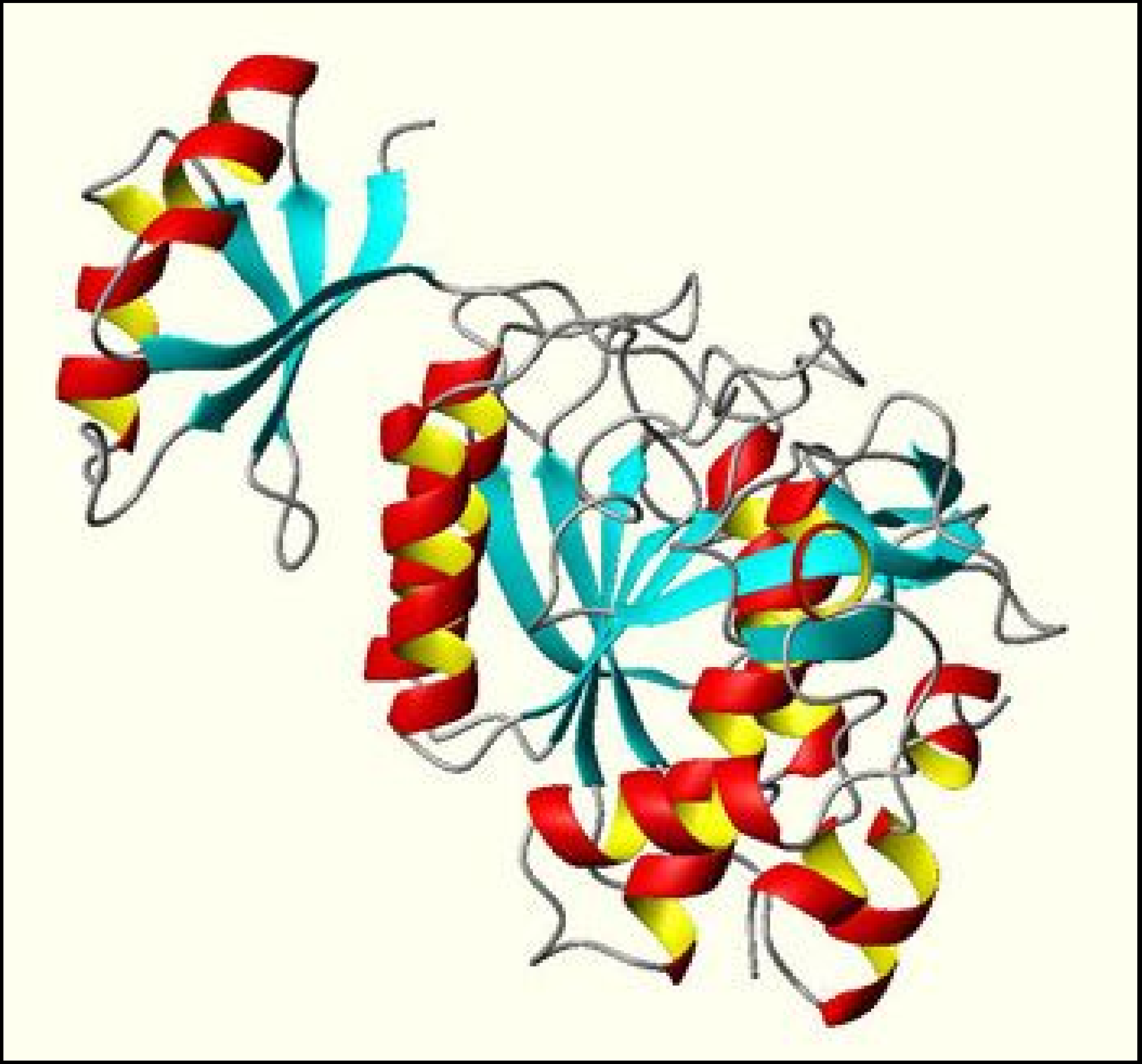
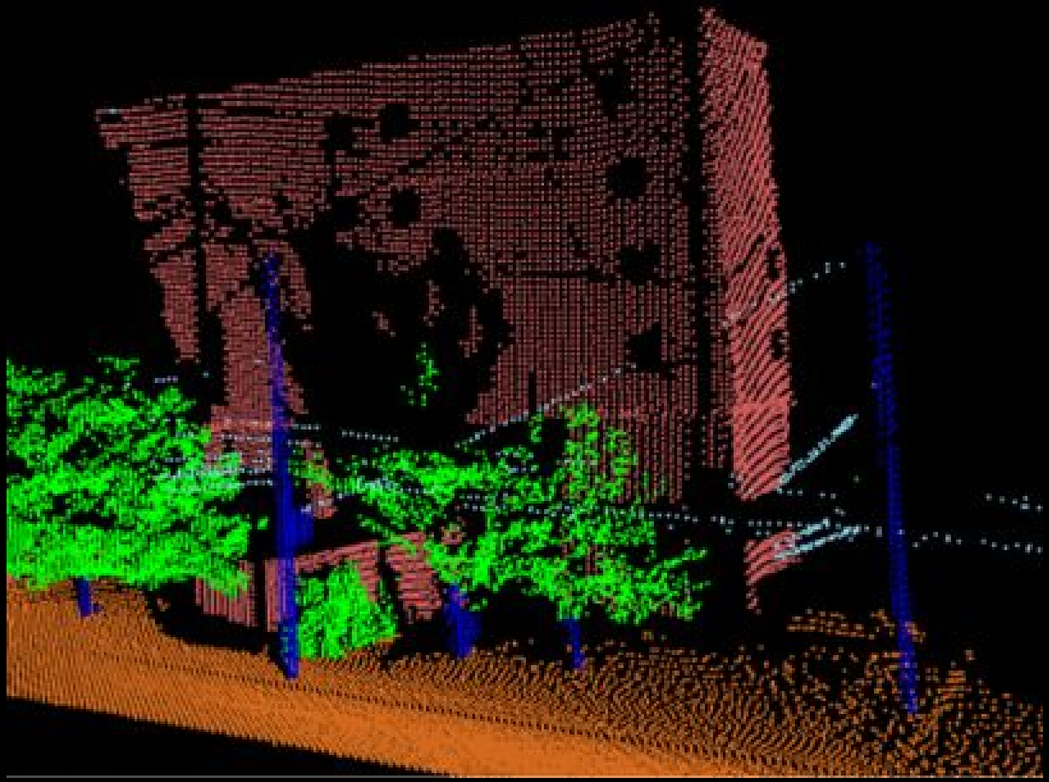


image credit: Daniel Muñoz

Protein secondary structure prediction



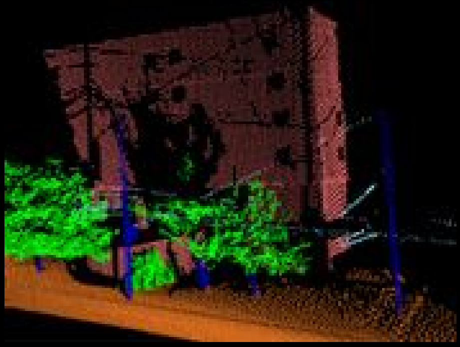
Standard solution methods



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

image credit: Daniel Muñoz

Predicting independently



- h : features of nearby voxels → class
- Ensure output is coherent at test time

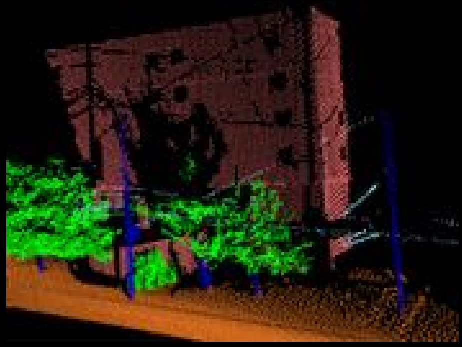
✓ Very simple to implement, often efficient

✗ Cannot capture correlations between predictions

✗ Cannot optimize a joint loss

image credit: Daniel Muñoz

Prediction with multitask bias



- h : features → (hidden representation)
  → yes/no
- Share (hidden representation) across all classes

✓ All advantages of predicting independently

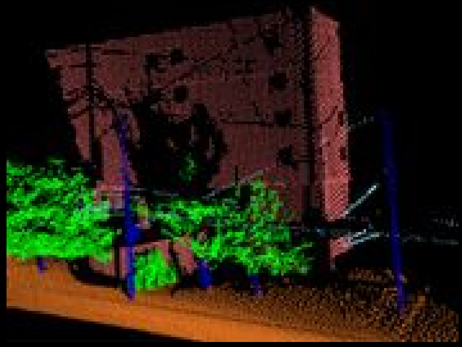
✓ May implicitly capture correlations

✗ Learning may be hard (… or not?)

✗ Still not optimizing a joint loss

image credit: Daniel Muñoz

Optimizing graphical models



- Encode output as a graphical model
- Learn parameters of that model to maximize:
  - p(true labels | input) *or*
  - cvx u.b. on loss(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

image credit: Daniel Muñoz

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: $(x,y) \sim D$

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

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

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

based on N samples

$$(x_n, y_n) \sim D$$

Challenges

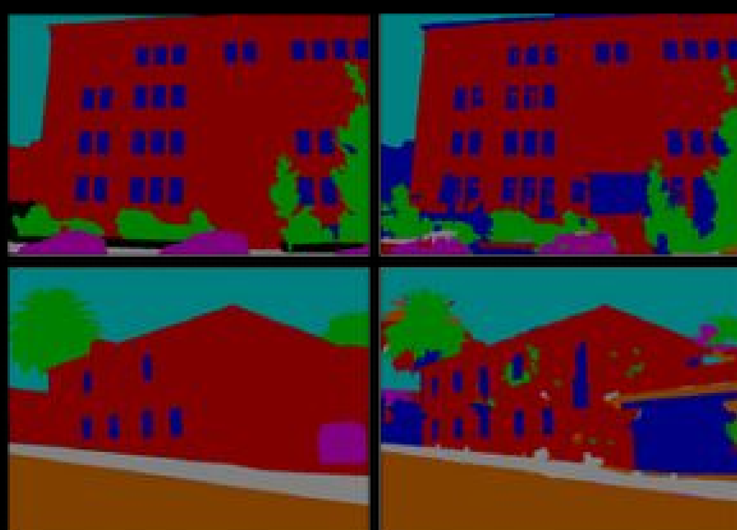
- Output space is too big to exhaustively search:
  - Typically exponential in size of input
  - *implies* y *must* decompose in some way

(often: x has many pieces to label)

- Loss function has combinatorial structure:

- Intersection over union

- Edit Distance



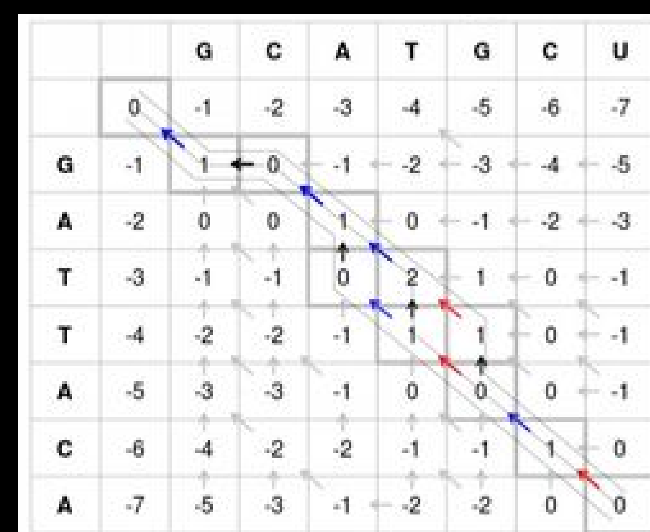
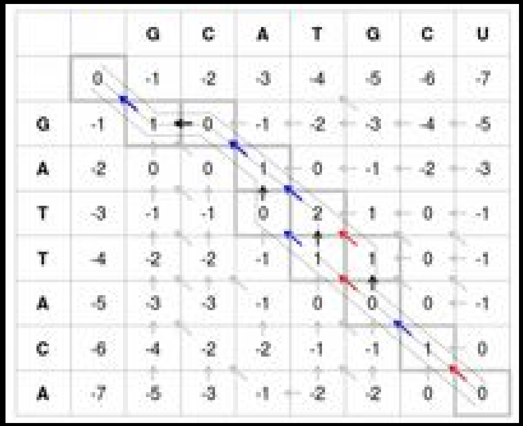


image credit: M. Pawan Kumar & Wikipedia

Decomposition of label

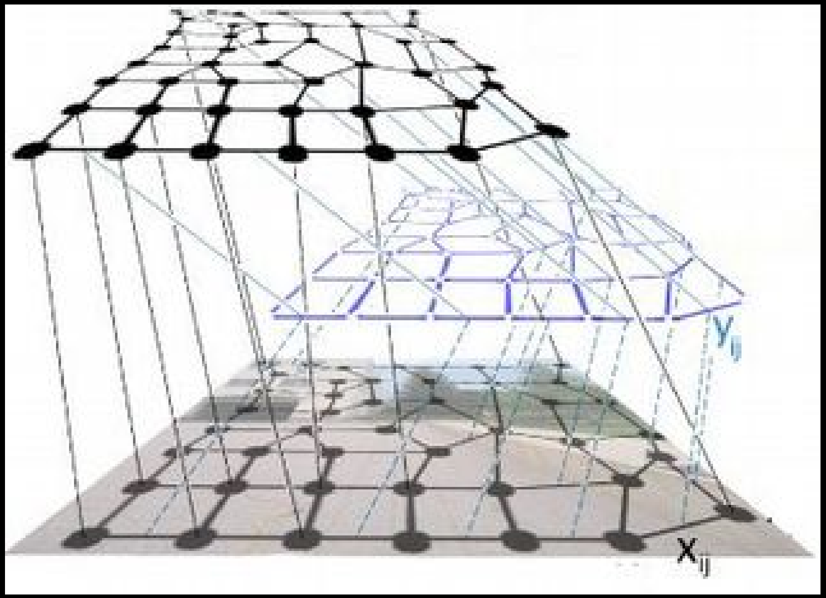
- Decomposition of **y** often implies an ordering

I	can	can	a	can
Pro	Md	Vb	Dt	Nn



		G	C	A	T	G	C	U
	0	-1	-2	-3	-4	-5	-6	-7
G	-1	1	0	-1	-2	-3	-4	-5
A	-2	0	0	1	0	-1	-2	-3
T	-3	-1	-1	0	2	1	0	-1
T	-4	-2	-2	-1	1	1	0	-1
A	-5	-3	-3	-1	0	0	0	-1
C	-6	-4	-2	-2	-1	-1	1	0
A	-7	-5	-3	-1	-2	-2	0	0

- But sometimes not so obvious....

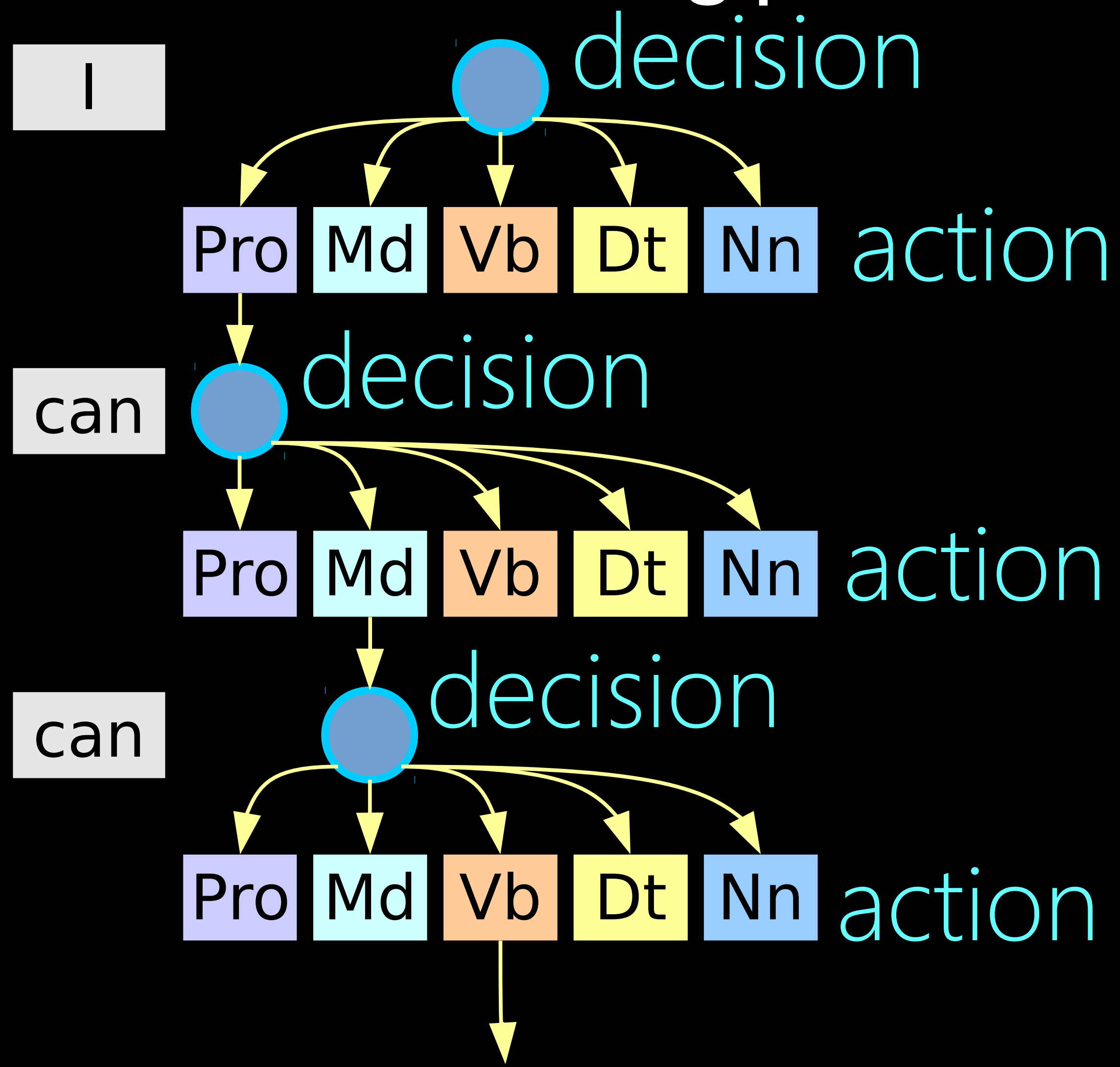


(we'll come back to this case later...)

image credit: Wikipedia & Ashutosh Saxena

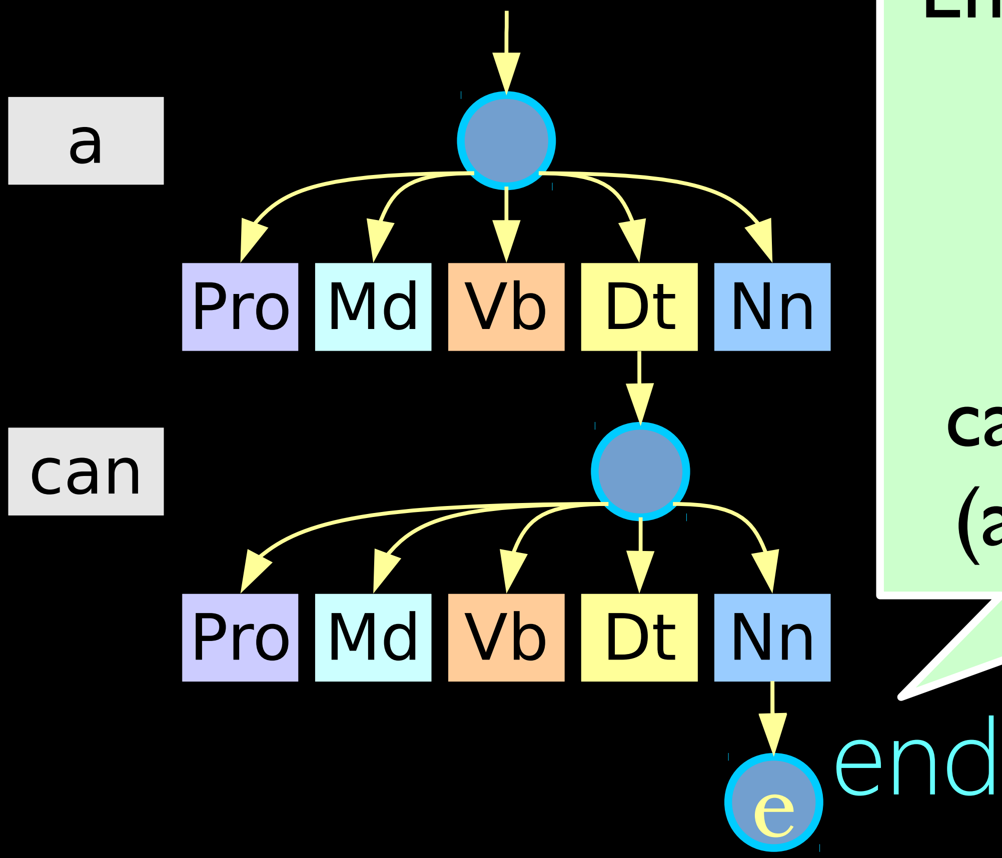
Search spaces

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



Search spaces

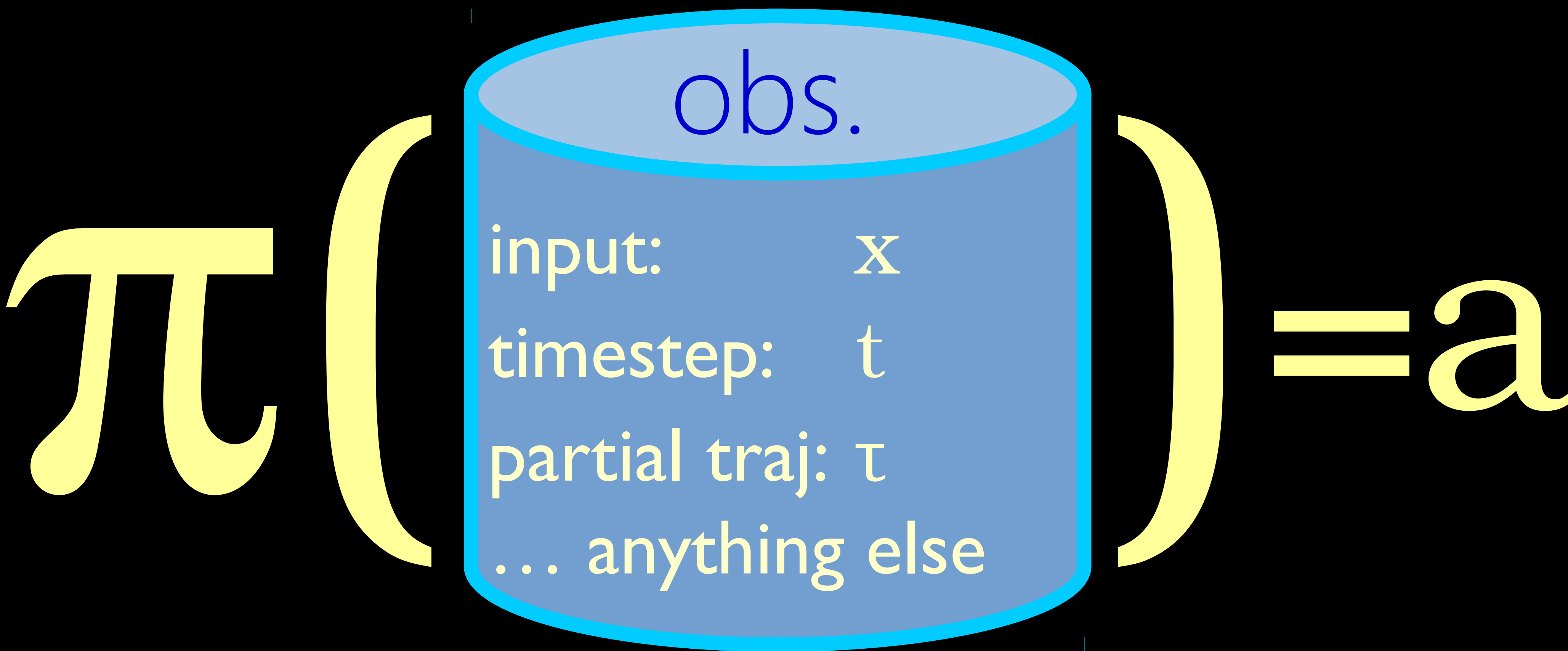
- When y decomposes in an ordered manner, a sequential decision making process emerges



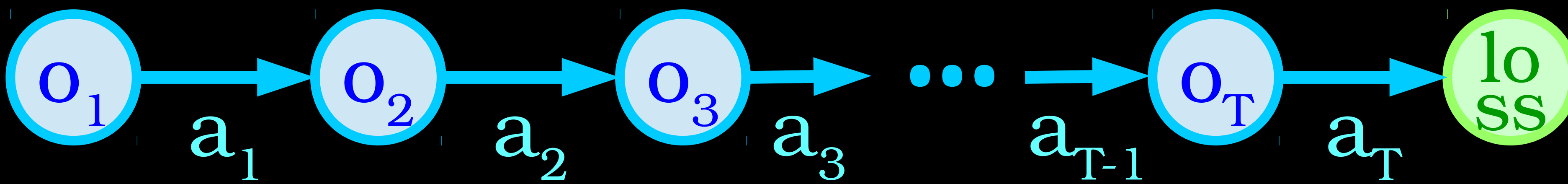
Encodes an output
$\hat{y} = \hat{y}(e)$
from which
$loss(y, \hat{y})$
can be computed
(at training time)

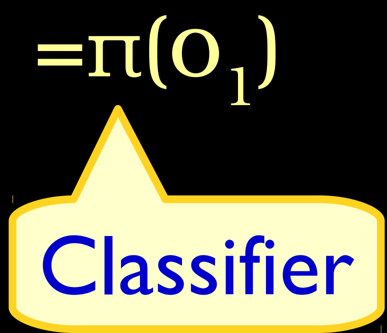
Policies

- A policy maps observations to actions



Versus reinforcement learning





Goal:

$$\min_\pi E\,[\,\text{loss}(\pi)\,]$$

In learning to search (L2S):

- *Labeled data* at training time
  $\Rightarrow$ can construct good/optimal policies
- Can "reset" and try the same example many times

Labeled data → Reference policy

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

The *minimum achievable loss* is:

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

The *optimal action* is the corresponding a_t

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

Ingredients for learning to search

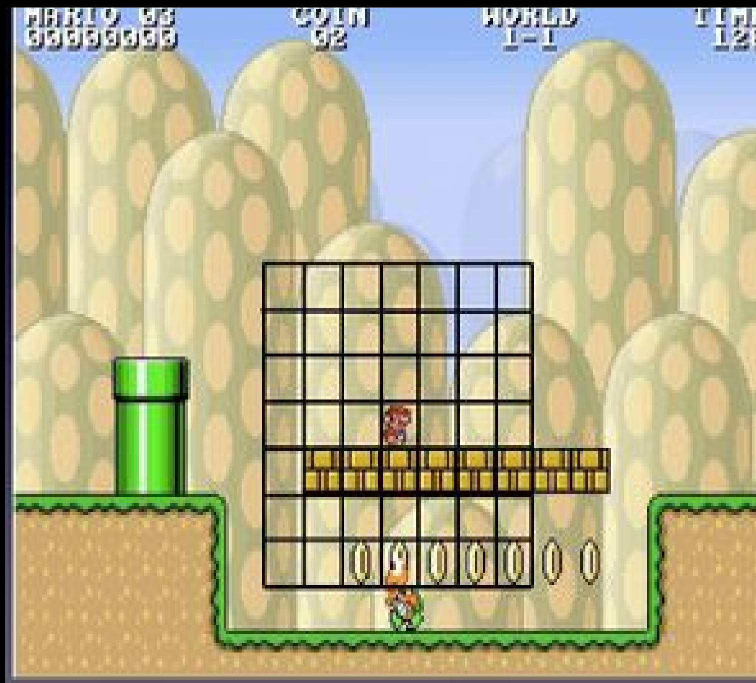
- Training data: $(x_n, y_n) \sim D$
- Output space: $Y(x)$
- Loss function: $\text{loss}(y, \hat{y})$

- Decomposition: $\{o\}, \{a\}, \ldots$
- Reference policy: $\pi^{ref}(o, y)$

An analogy from playing Mario

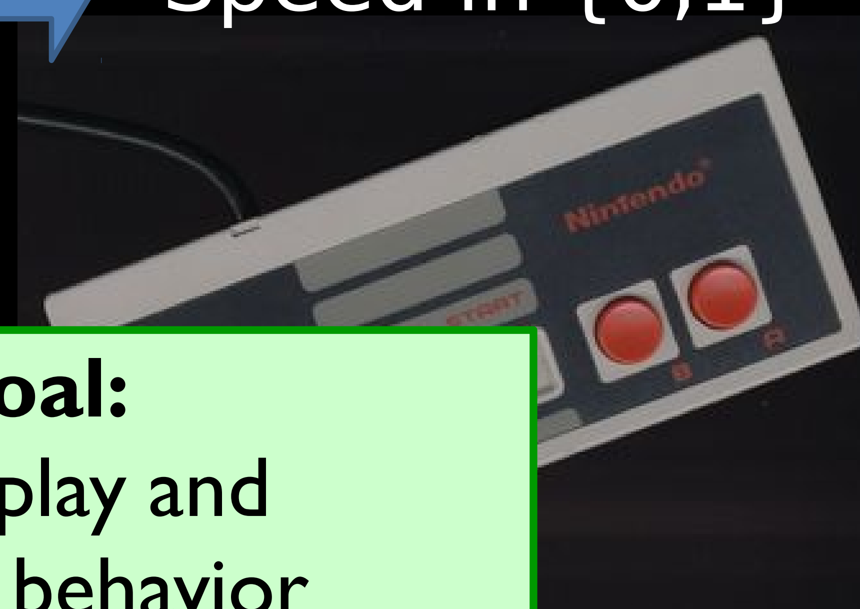
From Mario AI competition 2009

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

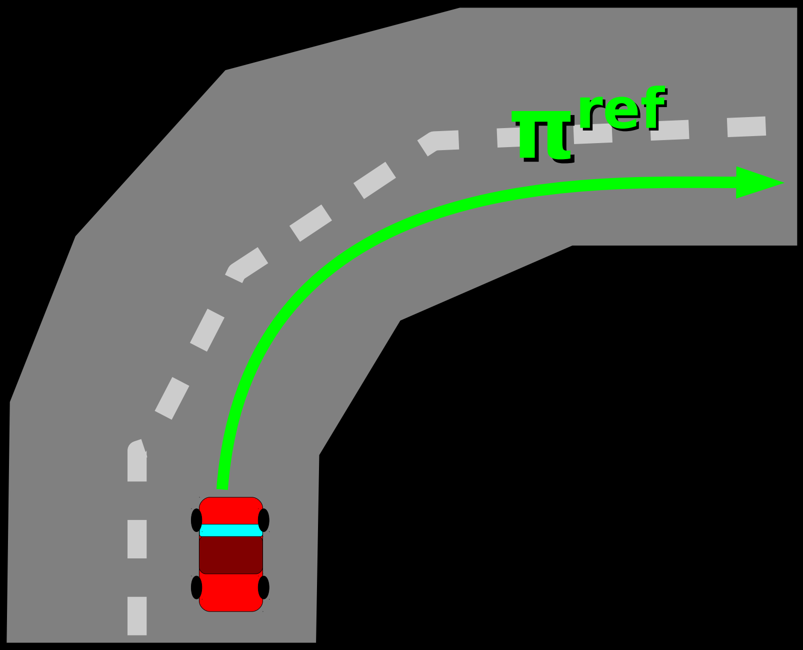
Training (expert)

Video credit: Stéphane Ross, Geoff Gordon and Drew Bagnell

Warm-up: Supervised learning

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

- **Let π play the game!**



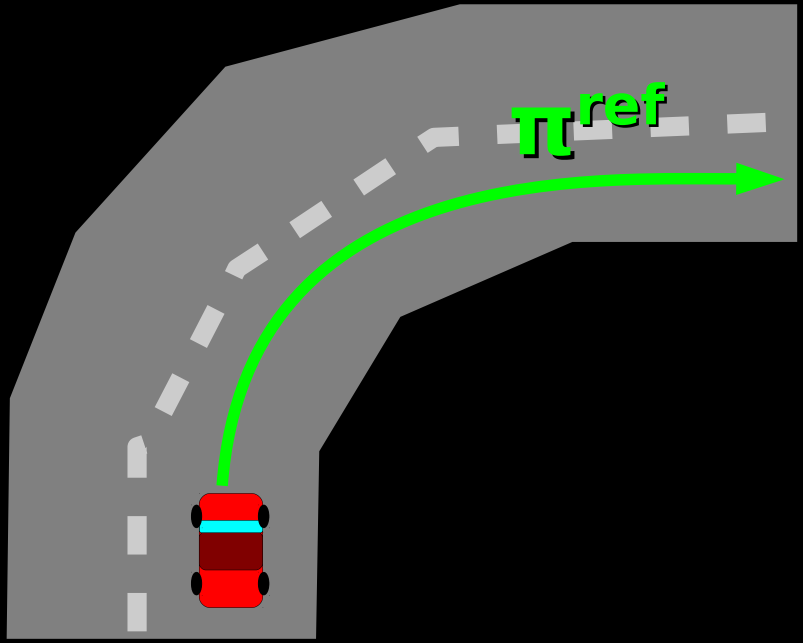
Test-time execution (sup. learning)

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!



Warm-up II: Imitation learning

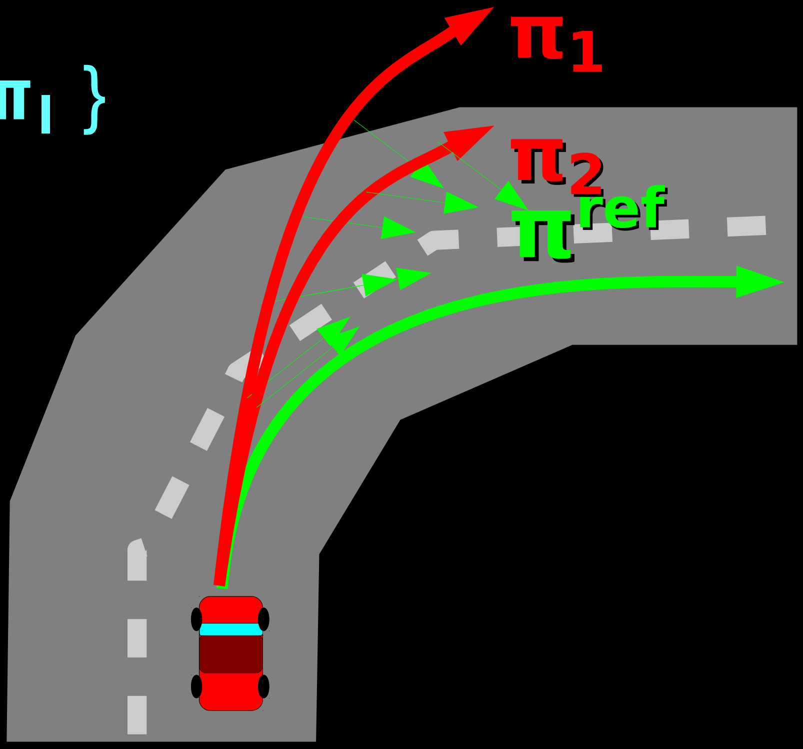
1. Collect trajectories from expert π^{ref}
2. Dataset $D_0 = \{ (o, \pi^{ref}(o,y)) \mid o \sim \pi^{ref} \}$
3. Train π_1 on D_0
4. Collect new trajectories from π_1
   - But let the *expert* steer!
5. Dataset $D_1 = \{ (o, \pi^{ref}(o,y)) \mid o \sim \pi_1 \}$
6. Train π_2 on $D_0 \cup D_1$

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

If $N = T \log T$,

$$L(\pi_n) < T\,\epsilon_N + O(1)$$

for some n



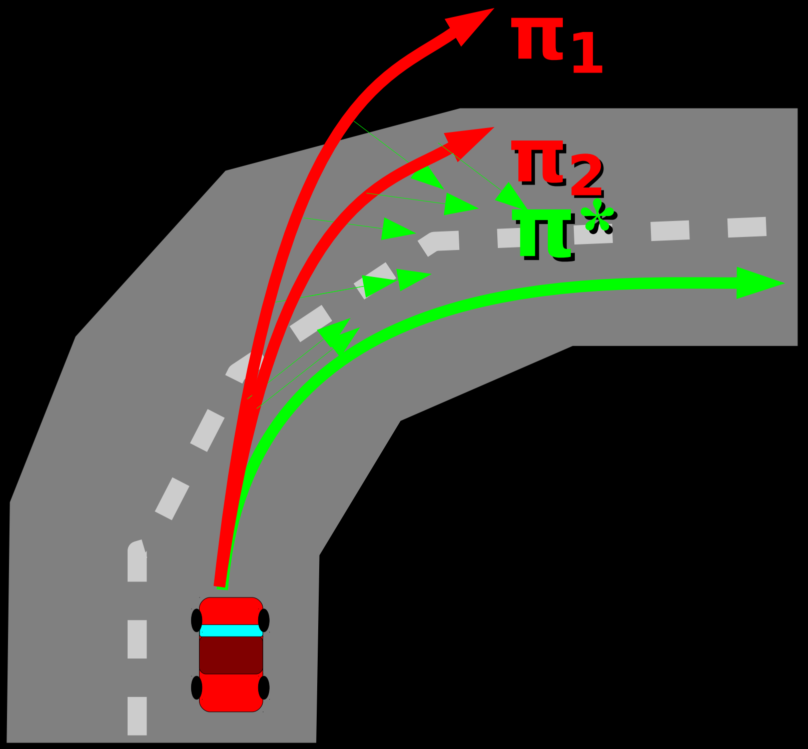
Test-time execution (DAgger)

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



Aside: cost-sensitive classification

Classifier: $h : x \rightarrow [K]$

Multiclass classification

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

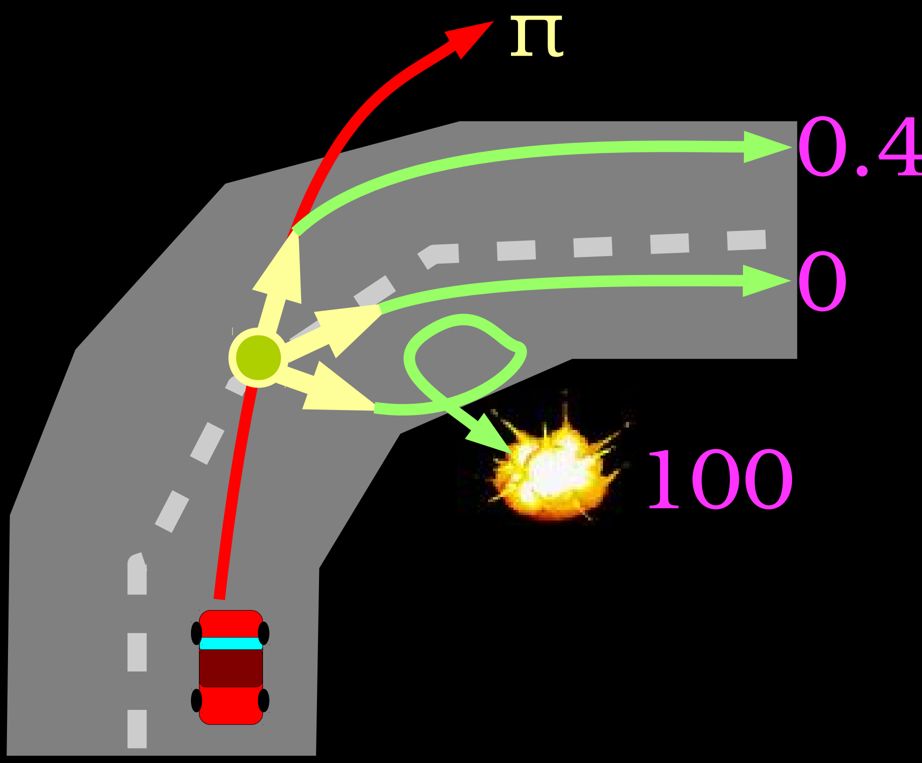
Cost-sensitive classification

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

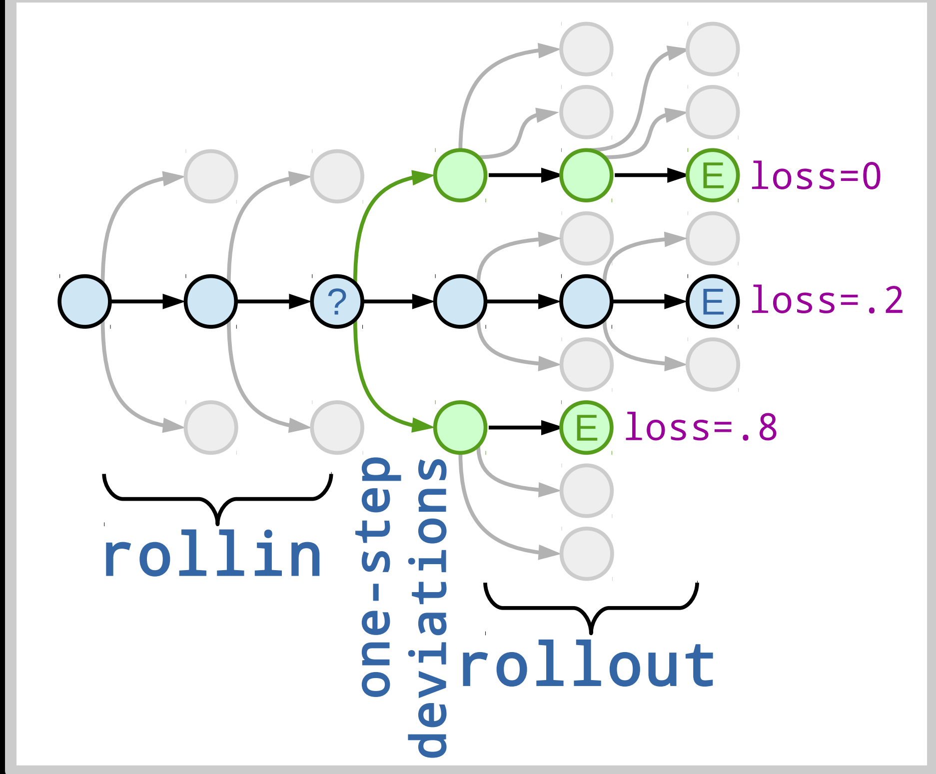
Learning to search: AggraVaTe

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)

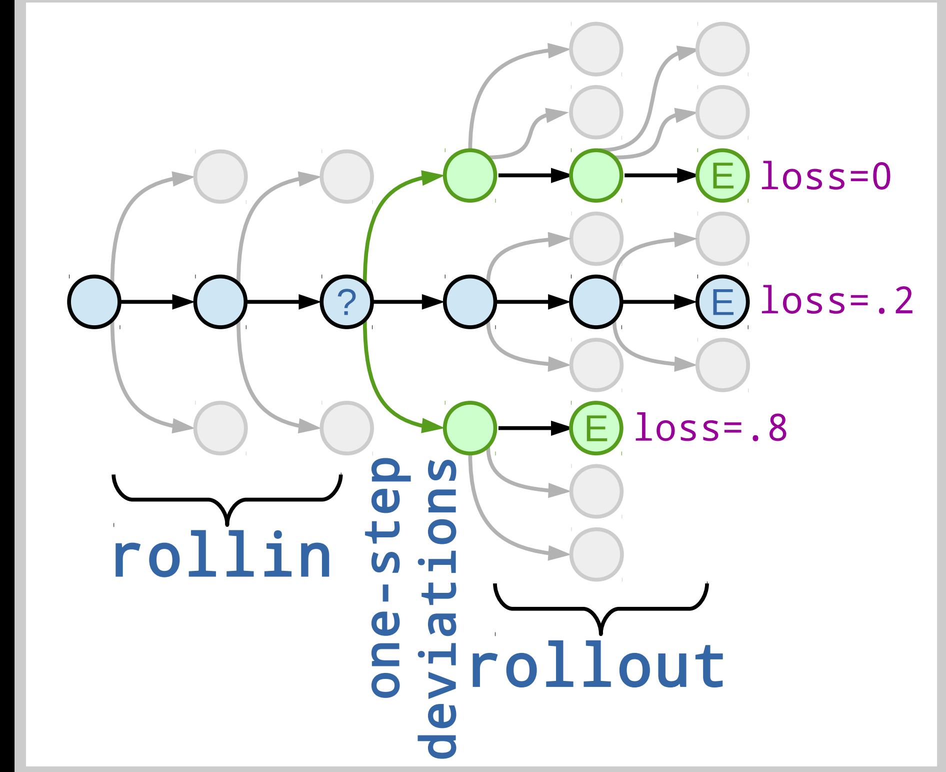


Learning to search: AggraVaTe



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: $(o, \vec{c})$

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 r
      iii. Obtain a final loss, c_a
   b) Generate a cost-sensitive classification example: $(\, o, \vec{c}\,)$

Often it's possible to analytically compute this loss *without* having to execute a roll-out!

Example I: Sequence labeling

- Receive input:

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

- Make a sequence of predictions:

```
x = the monster ate the sandwich
ŷ = Dt  Dt      Dt  Dt  Dt
```

- Pick a timestep and try all perturbations there:

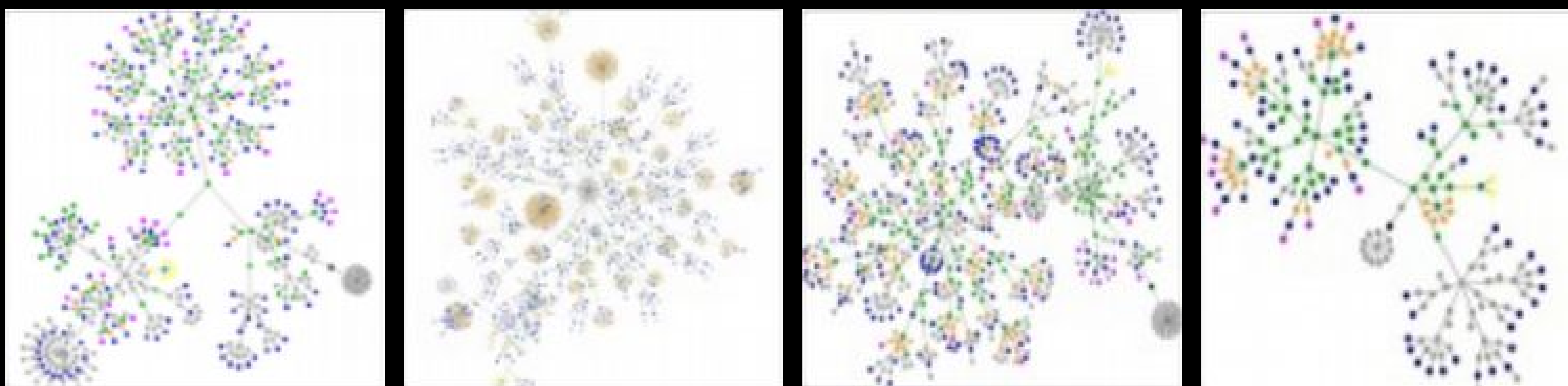
```
   x = the monster ate the sandwich
ŷ_Dt = Dt  Dt
ŷ_Nn = Dt  Nn
ŷ_Vb = Dt  Vb
```

- Compute losses and construct example:

```
( { w=monster, p=Dt, …},
  [1,0,1] )
```

Example II: Graph labeling

- Task: label nodes of a graph given node features (and possibly edge features)
- Example: WebKB webpage labeling

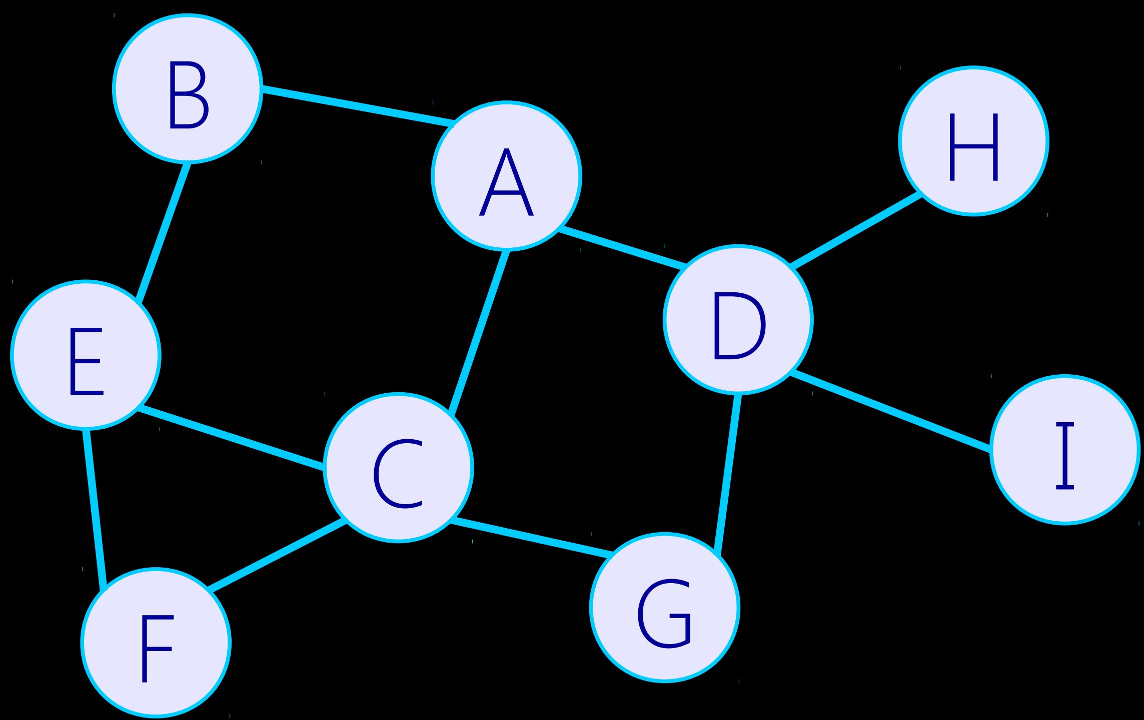


U Wisconsin U Washington U Texas Cornell

- 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

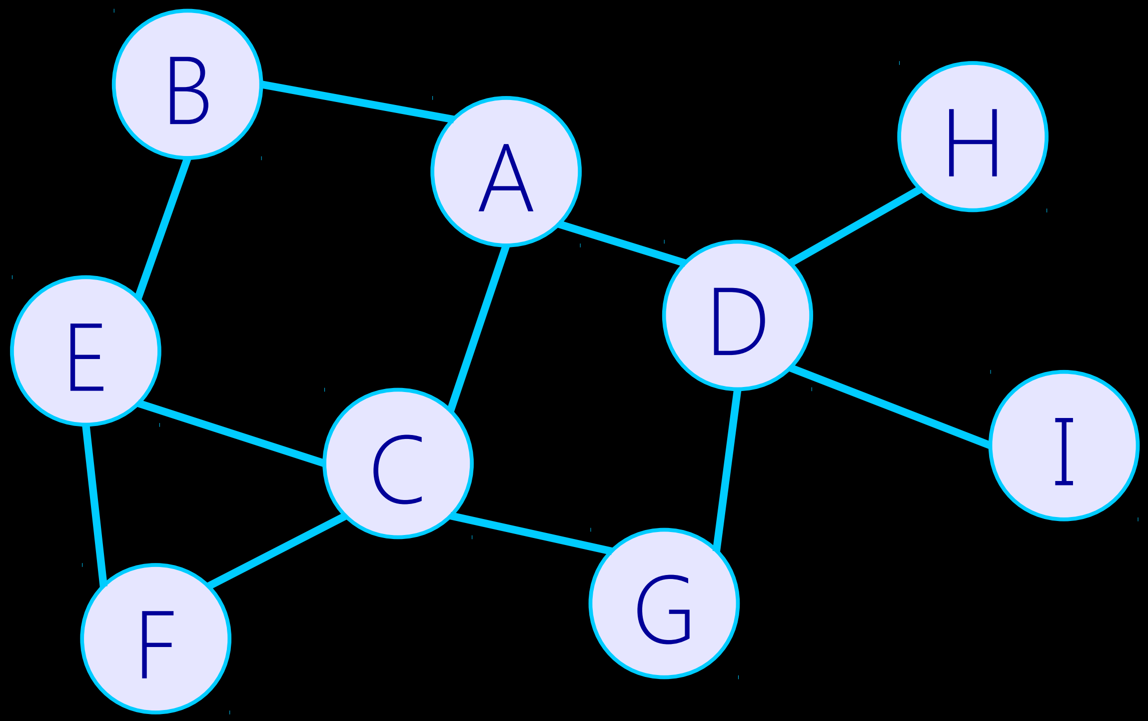


Linearization:

ABCDEFGHI
HGFEDCBA
BCDEFGHI
HGFEDCBA
. . .

Example II: Graph labeling

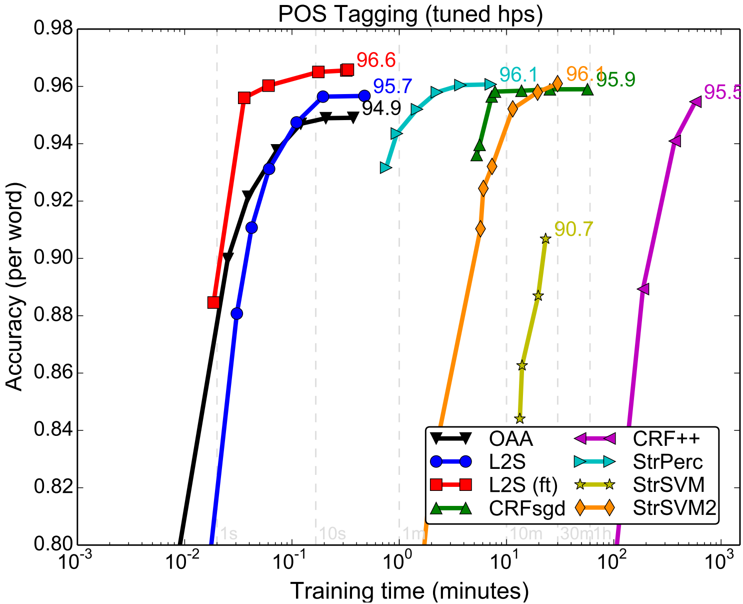
1. Pick a node (= timestep)
2. Construct example based on neighbors' labels
3. Perturb current node's label



Outline

1. Empirics
2. Analysis
3. Programming
4. Others and Issues

What part of speech are the words?



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

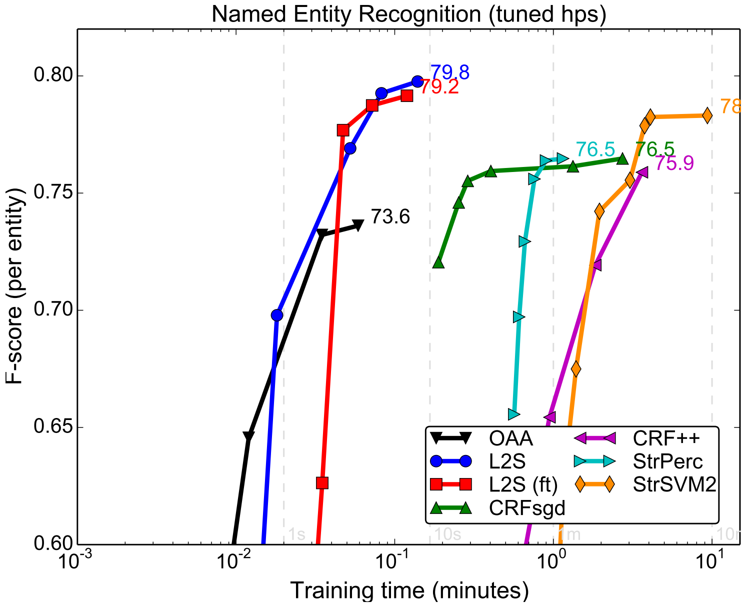
1 |w Despite
2 |w continuing
3 |w problems
1 |w in
4 |w its
5 |w newsprint
5 |w business

...

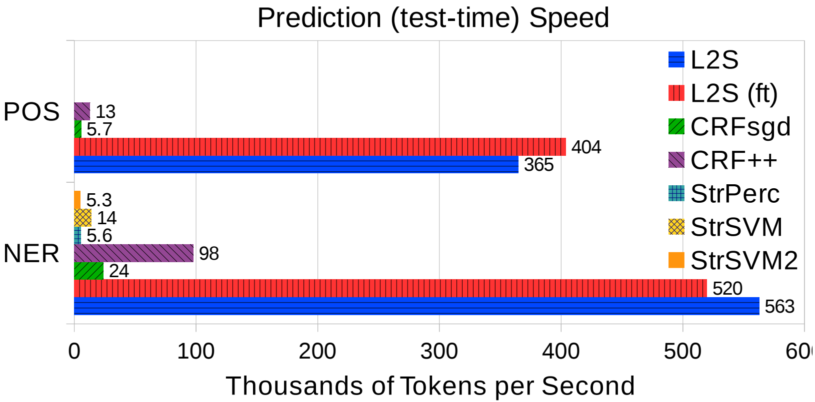
vw -b 24 -d wsj.train.vw -c –search_task sequence –search 45 –search_alpha 1e-8 –search_neighbor_features -1:w,1:w –affix -1w,+1w -f foo.reg

vw -t -i foo.reg wsj.test.vw

Is this word a name or not?



How fast in evaluation?



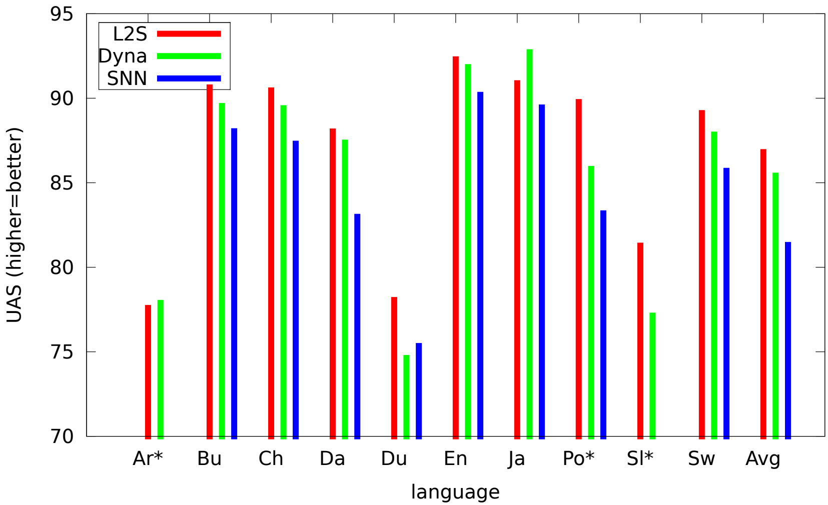
Entity Relation

Goal: find the Entities and then find their Relations

Method	Entity F1	Relation F1	Train Time
Structured SVM	88.00	50.04	300 seconds
L2S	92.51	52.03	13 seconds

L2S uses ~100 LOC.

Find dependency structure of sentences.



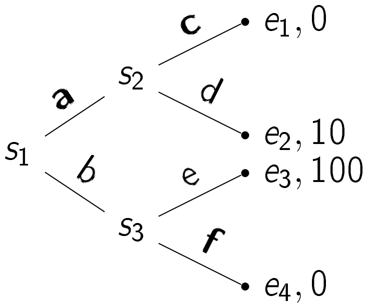
L2S uses ~300 LOC.

Outline

1. Empirics
2. Analysis
3. Programming
4. Others and Issues

Effect of Roll-in and Roll-out Policies

roll-out → ↓ roll-in	Reference	Half-n-half	Learned
Reference	Inconsistent		
Learned	.		



Effect of Roll-in and Roll-out Policies

roll-out → / ↓ roll-in	Reference	Half-n-half	Learned
Reference	Inconsistent		
Learned	.		

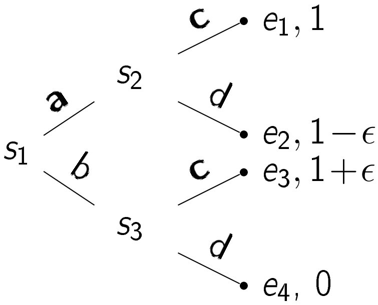
Theorem

Roll-in with ref:

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

Effect of Roll-in and Roll-out Policies

roll-out → ↓ roll-in	Reference	Half-n-half	Learned
Reference	Inconsistent		
Learned	Consistent No local opt		



Effect of Roll-in and Roll-out Policies

roll-out → ↓ roll-in	Reference	Half-n-half	Learned
Reference	Inconsistent		
Learned	Consistent No local opt		

Theorem

Roll-out with Ref:

0 *cost-sensitive regret* ⇒ 0 *joint regret*

(but not local optimality)

Effect of Roll-in and Roll-out Policies

roll-out $\rightarrow$ $\downarrow$ roll-in	Reference	Half-n-half	Learned
Reference	Inconsistent		
Learned	Consistent No local opt		Reinf. L.

Theorem

Ignore Ref:
$\Rightarrow$ *Equivalent to reinforcement learning.*

Effect of Roll-in and Roll-out Policies

roll-out → / ↓ roll-in	Reference	Half-n-half	Learned
Reference	Inconsistent		
Learned	Consistent No local opt	Consistent Local Opt	Reinf. L.

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

π^{ref} = reference policy
$\bar{\pi}$ = stochastic average learned policy
$J(\pi)$ = expected loss of π.

Theorem

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

AggreVaTe Regret Decomposition

π^{ref} = reference policy
$\bar{\pi}$ = stochastic average learned policy
$J(\pi)$ = expected loss of π.

Theorem

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

T = number of steps
$\hat{\pi}_n$ = nth learned policy
$D^t_{\hat{\pi}_n}$ = distribution over x at time t induced by $\hat{\pi}_n$
$Q^{\pi}(x, \pi')$ = loss of π' at x then π to finish

Proof

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

Proof

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

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

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

Proof

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

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

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

Proof

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

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

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

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

Proof

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

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

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

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

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

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

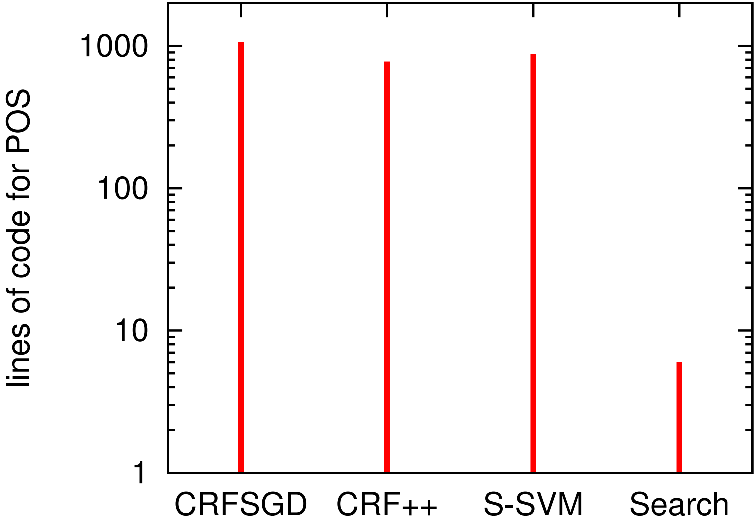
So $J(\bar{\pi}) - J(\pi^{\text{ref}})$

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

Outline

1. Empirics
2. Analysis
3. Programming
4. Others and Issues

Lines of Code



How?

Sequential_RUN(*examples*)

```
for i = 1 to len(examples) do
    prediction ← predict(examples[i], examples[i].label)
    loss(prediction ≠ examples[i].label)
end for
```

How?

Sequential_RUN(*examples*)

```
for i = 1 to len(examples) do
    prediction ← predict(examples[i], examples[i].label)
    loss(prediction ≠ examples[i].label)
end for
```

Decoder + loss + reference advice

RunParser(*sentence*)

stack $S \leftarrow$ {**Root**}
buffer $B \leftarrow$ [words in sentence]
arcs $A \leftarrow \emptyset$
while $B \neq \emptyset$ or $|S| > 1$ **do**
    ValidActs $\leftarrow$ GetValidActions(S, B)
    features $\leftarrow$ GetFeat(S, B, A)
    ref $\leftarrow$ GetGoldAction(S, B)
    action $\leftarrow$ **predict**(features, ref, ValidActs)
    $S, B, A \leftarrow$ Transition(S, B, A, action)
end while
loss($A[w] \neq A^*[w], \forall w \in$ sentence)
return output

Program/Search equivalence

Theorem: Every algorithm which:

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

defines a search space, and such an algorithm exists for every search space.

It even works in Python

```
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.

Inc. P.	Collins & Roark, ACL 2004.
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.

Inc. P. Collins & Roark, ACL 2004.

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

Searn Daume III et al., MLJ 2009.

Aggra Ross & Bagnell, `http://arxiv.org/pdf/1406.5979`

Learning to Search

When the reference policy is optimal

Searn Daume III et al., MLJ 2009.

Aggra Ross & Bagnell, `http://arxiv.org/pdf/1406.5979`

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, http://arxiv.org/pdf/1406.5979

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.

1st sol. Boyd, 1994.

Inverse Reinforcement Learning

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

propose Kalman, 1968.

1st sol. Boyd, 1994.

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.

1st sol. Boyd, 1994.

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

What's missing? Automatic Search order

Learning to search $\simeq$ 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.

What's missing?
Efficient Cost-Sensitive Learning

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.

How to optimize discrete joint loss?

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

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.

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.

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