Advances in Structured Prediction

John Langford
Microsoft Research
jl@hunch.net

Hal Daumé III
U Maryland
me@hal3.name

Slides and more: http://hunch.net/~l2s

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
```

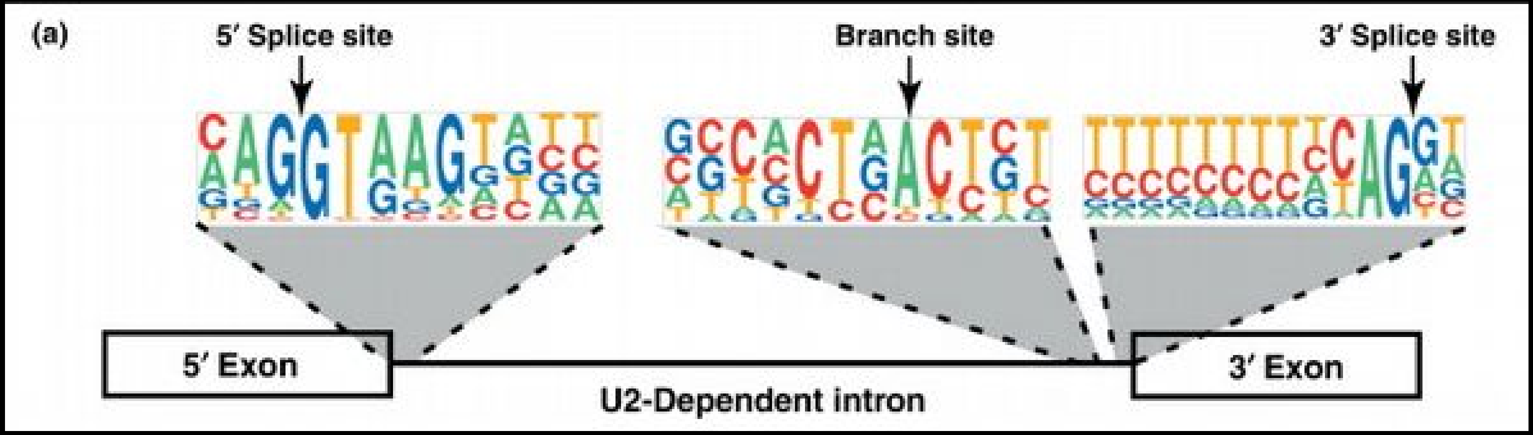
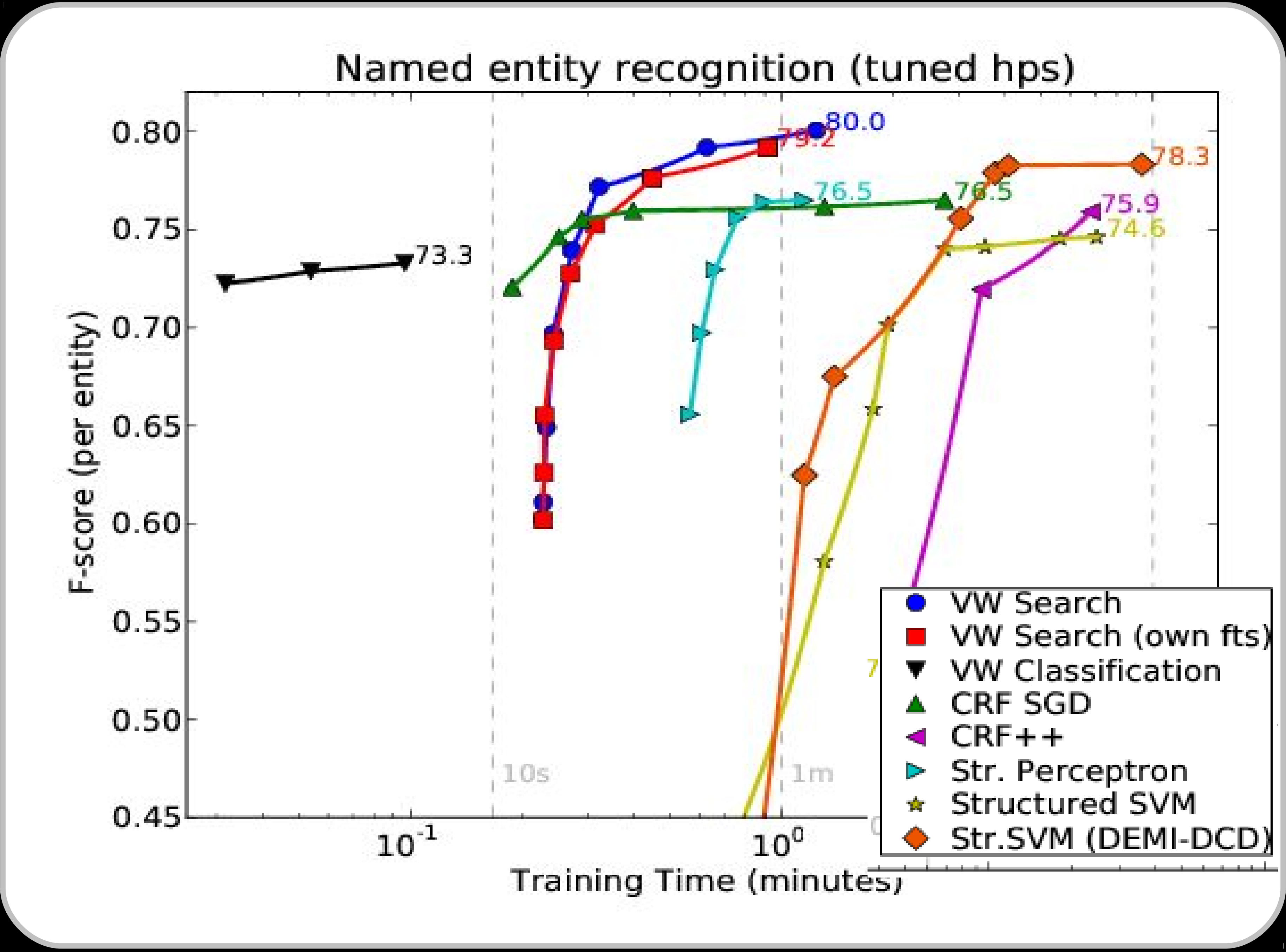
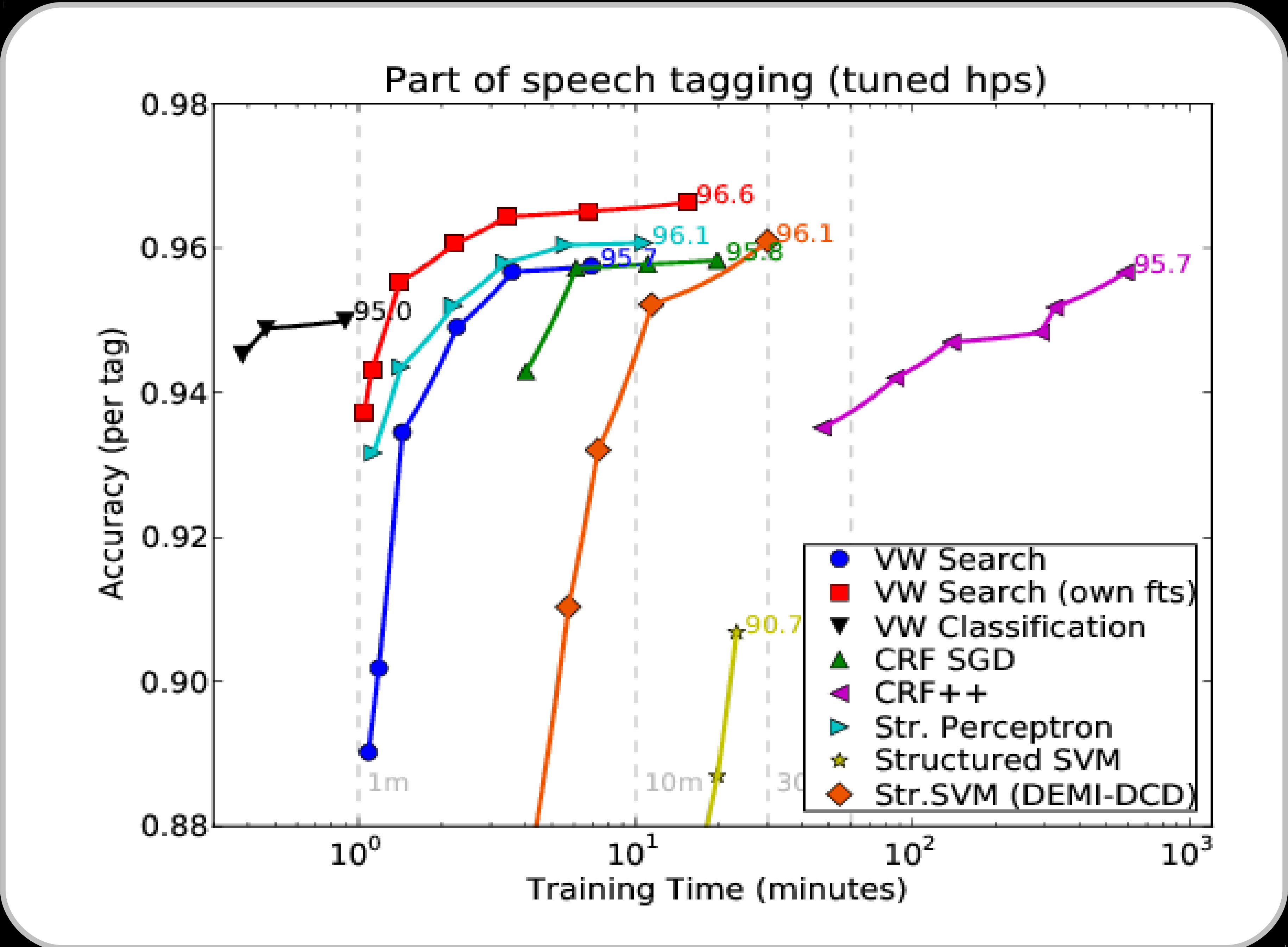


image credit: Richard Padgett

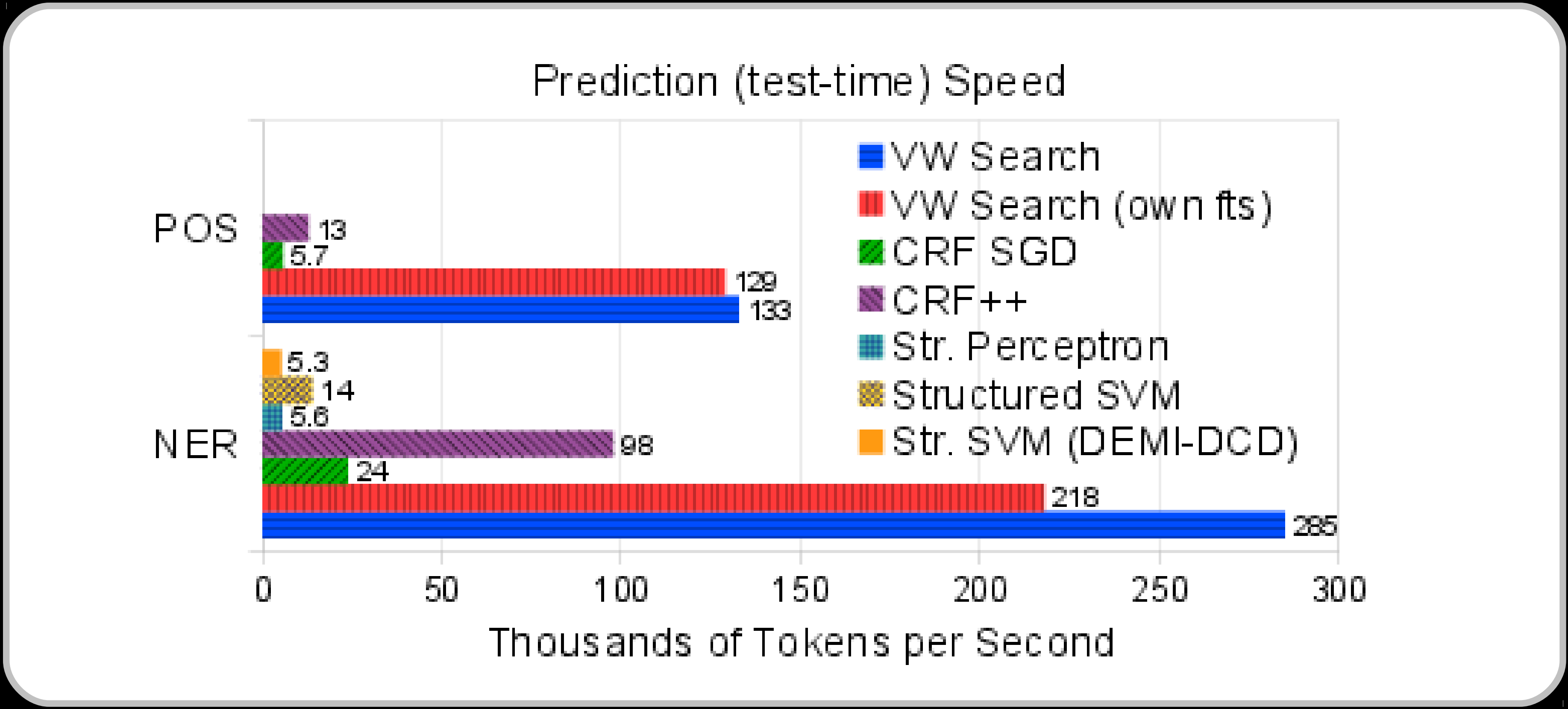
Training time versus test accuracy



Training time versus test accuracy



Test time speed



There's even a python interface

seq.py

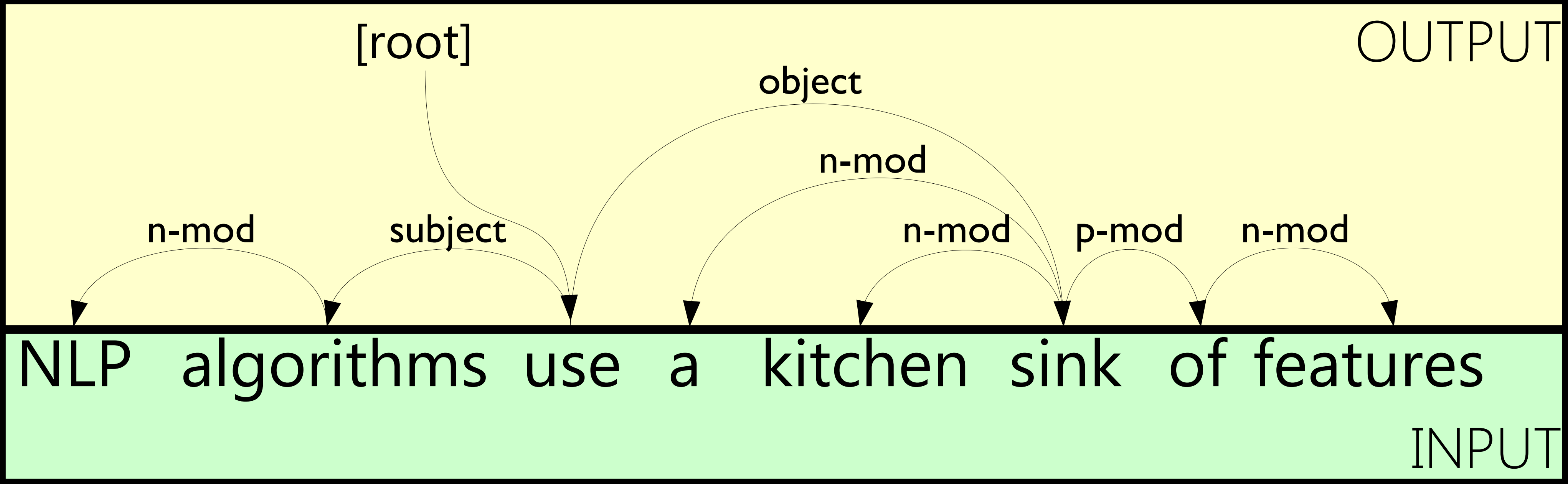
File Edit Options Buffers Tools Python YASnippet Development Help

```python
import pyvw

class SequenceLabeler(pyvw.SearchTask):
    def __init__(self, vw, sch, num_actions):
        pyvw.SearchTask.__init__(self, vw, sch, num_actions)
        sch.set_options( sch.AUTO_HAMMING_LOSS |
                         sch.AUTO_CONDITION_FEATURES )

    def _run(self, sentence):
        output = []
        for n in range(len(sentence)):
            pos,word = sentence[n]
            with self.example({'w': [word]}) as ex:
                pred = self.sch.predict(examples=ex,
                                        my_tag=n+1,
                                        oracle=pos,
                                        condition=(n,'p'))
                output.append(pred)
        return output

vw = pyvw.vw("--search 4 --quiet --search_task hook --ring_size 1024")
sequenceLabeler = vw.init_search_task(SequenceLabeler)
sequenceLabeler.learn(my_dataset)
```

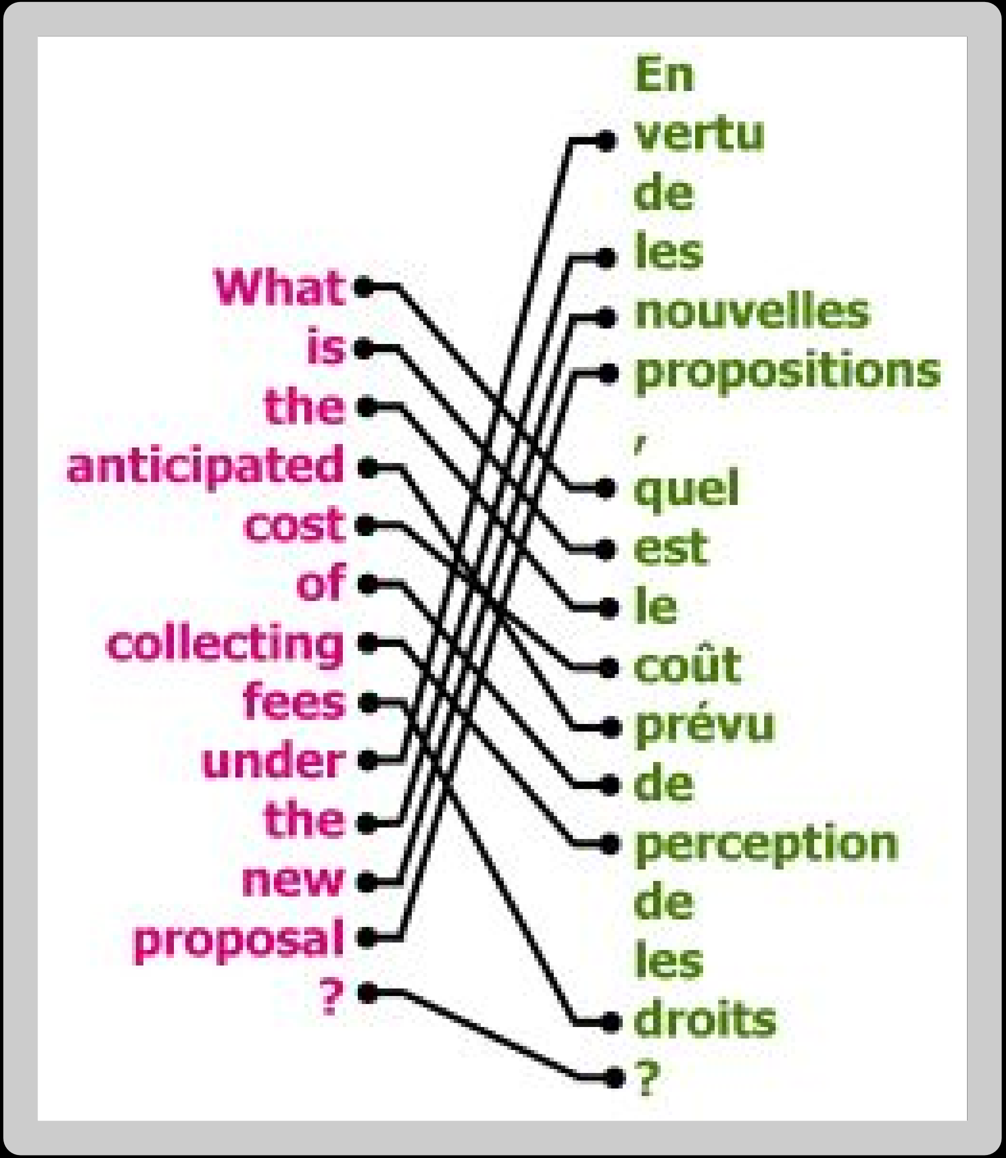
U:**- seq.py All (2,0) (Python +2 AC yas)

(D+Langford+Ross, NIPS 2014 reject – on arxiv)

Natural language parsing



(Bipartite) matching



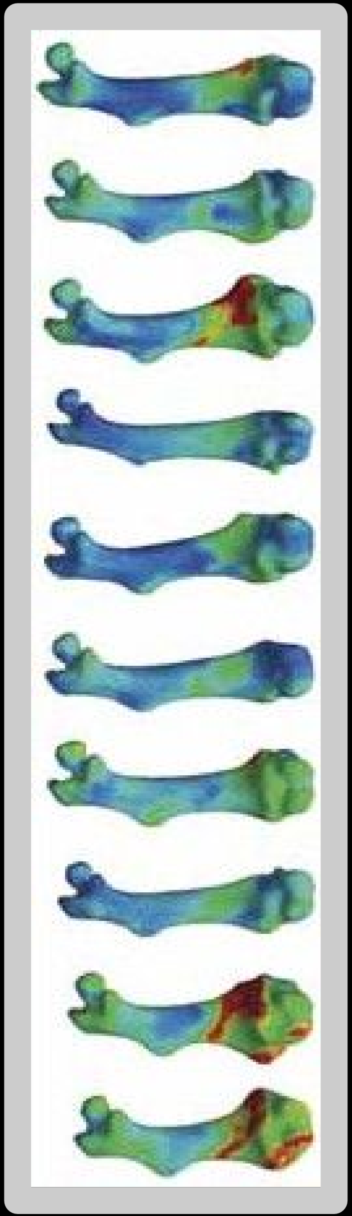


image credit: Ben Taskar; Liz Jurrus

Machine translation

Image segmentation

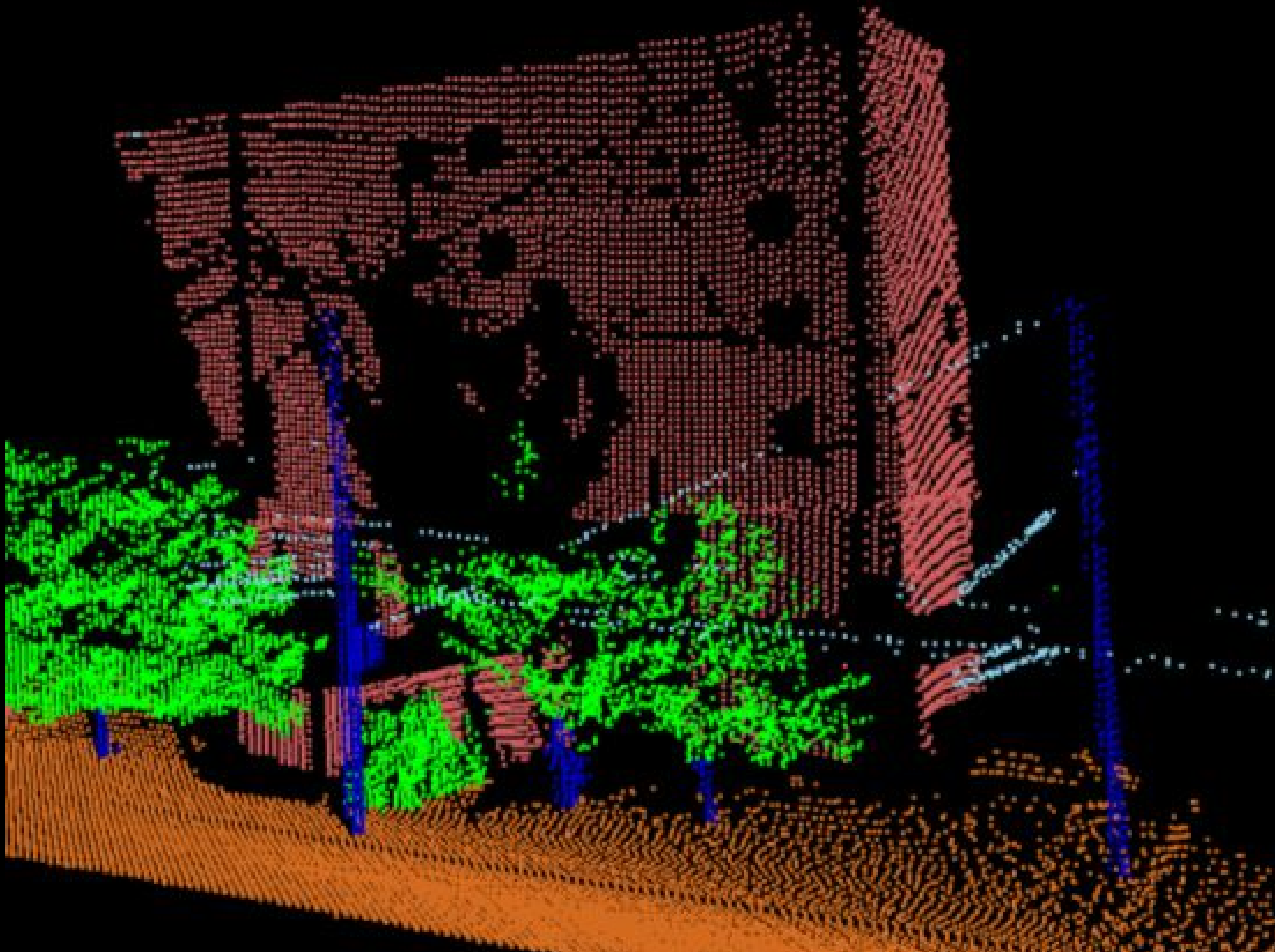
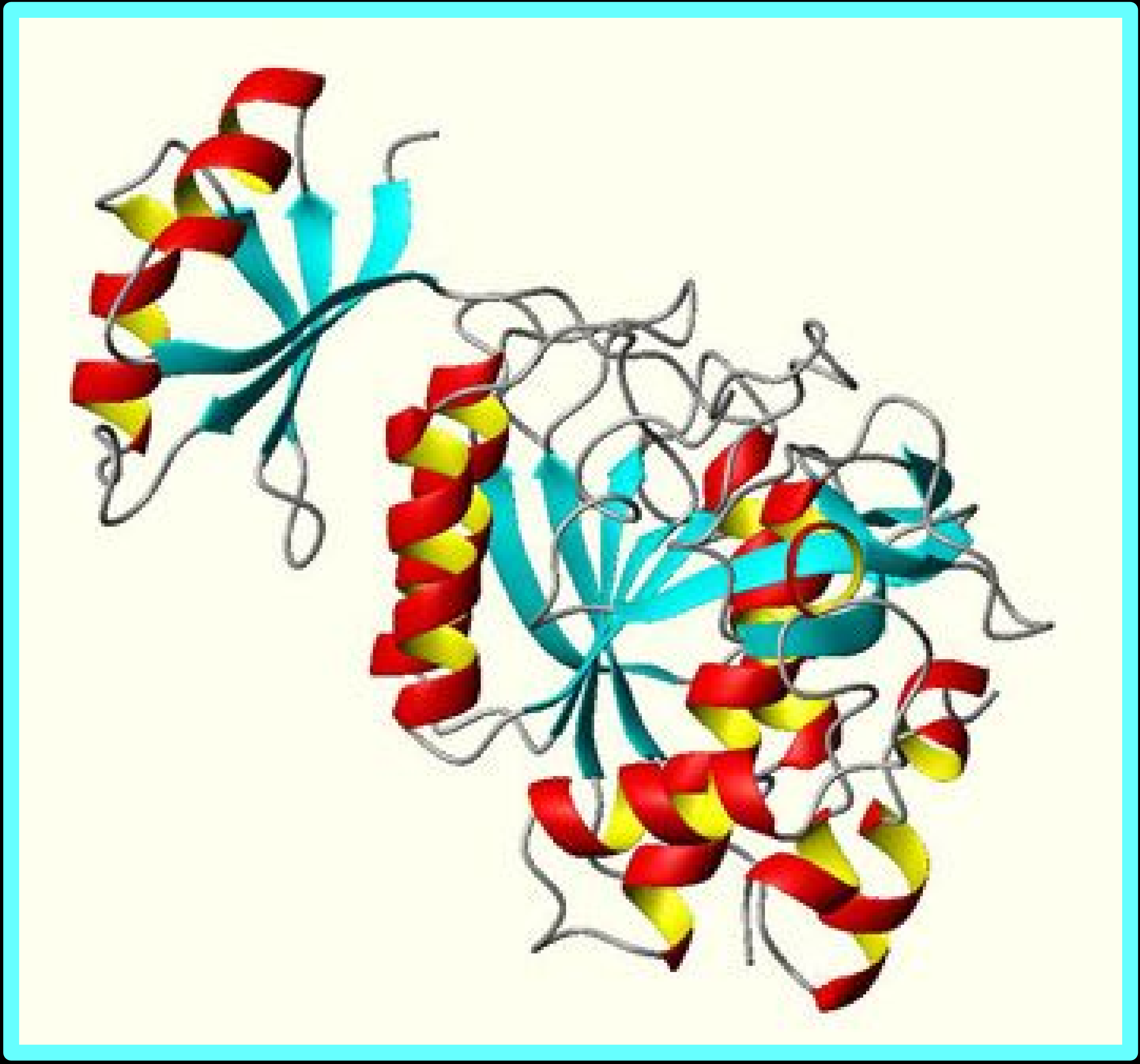


image credit: Daniel Muñoz

Protein secondary structure prediction



State of the art accuracy in....

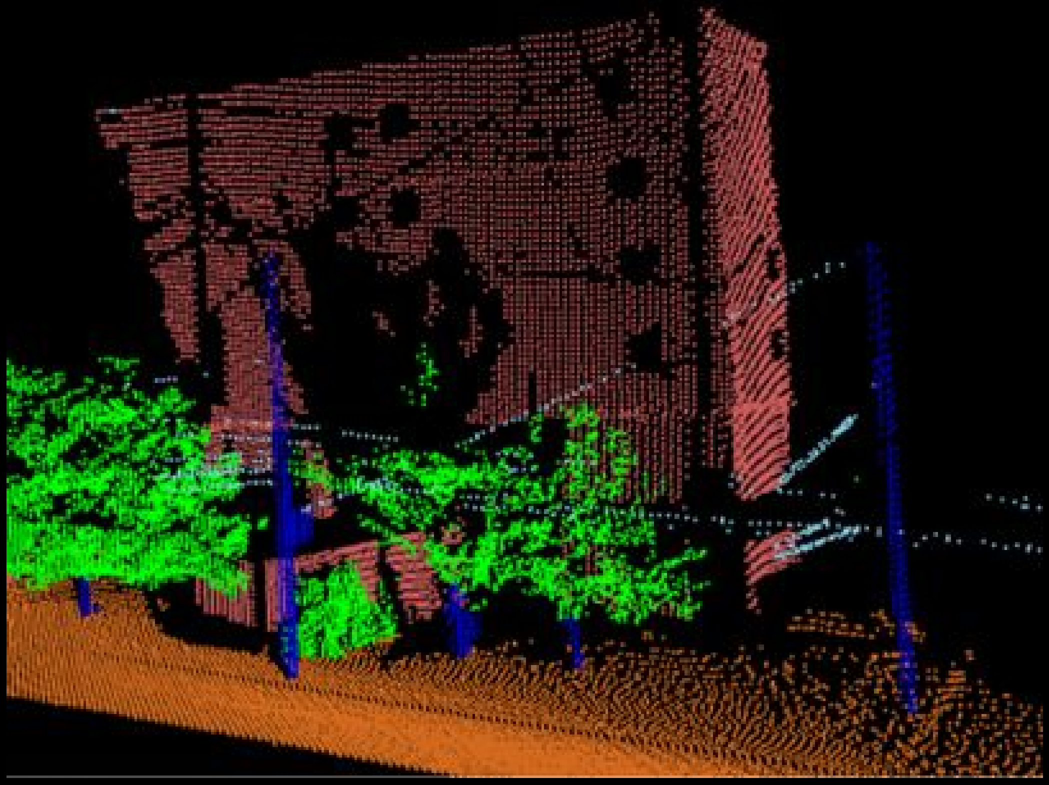
- Part of speech tagging (1 million words)

  - US: 6 lines of code 1 minute to train
  - CRFsgd: 1068 lines 30 minutes
  - CRF++: 777 lines hours

- Named entity recognition (200 thousand words)

  - US: 30 lines of code 10 seconds to train
  - CRFsgd: 1 minute
  - CRF++: 10 minutes
  - SVMstr: 876 lines 30 minutes (suboptimal accuracy)

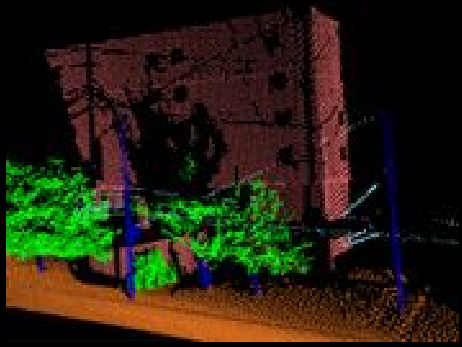
Standard solution methods



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

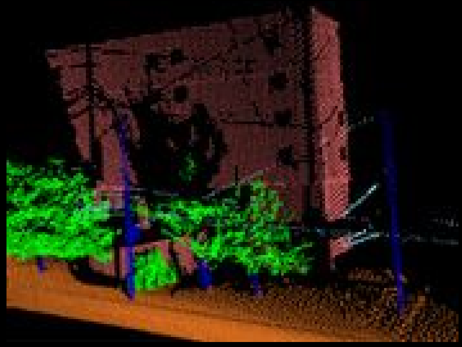
✓ Very simple to implement, often efficient

✗ 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

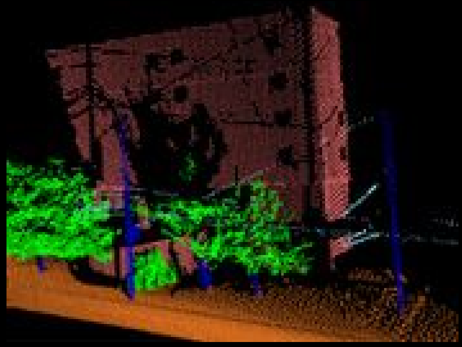
✓ May implicitly capture correlations

✗ 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 optimize:
  - 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

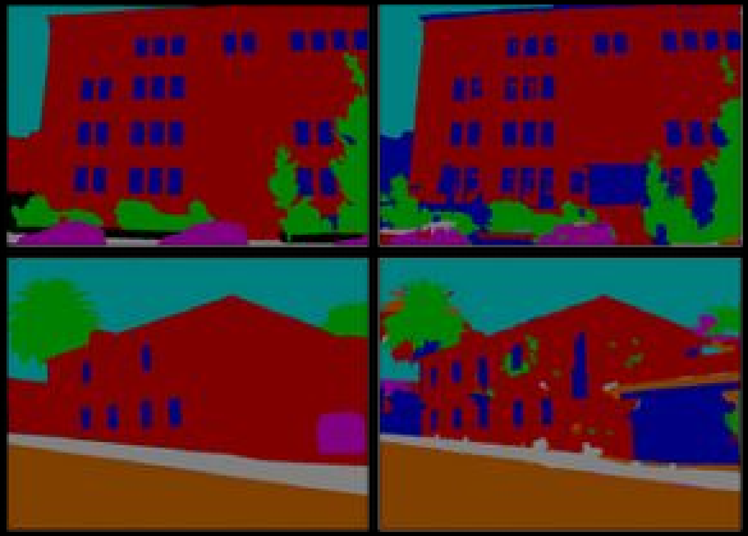
$$(x_n, y_n) \sim D$$

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

		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

image credit: M. Pawan Kumar & Wikipedia

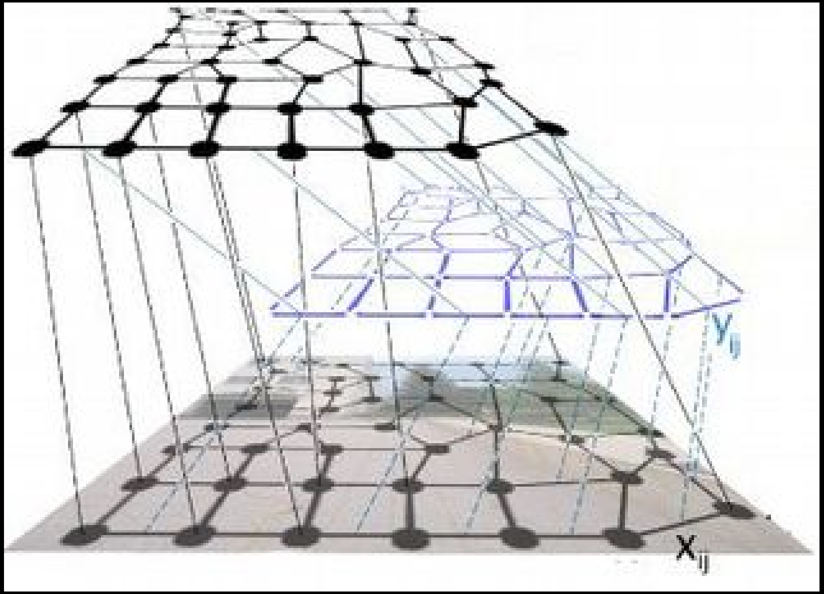
Decomposition of label

- Decomposition of $\mathbf{y}$ 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....

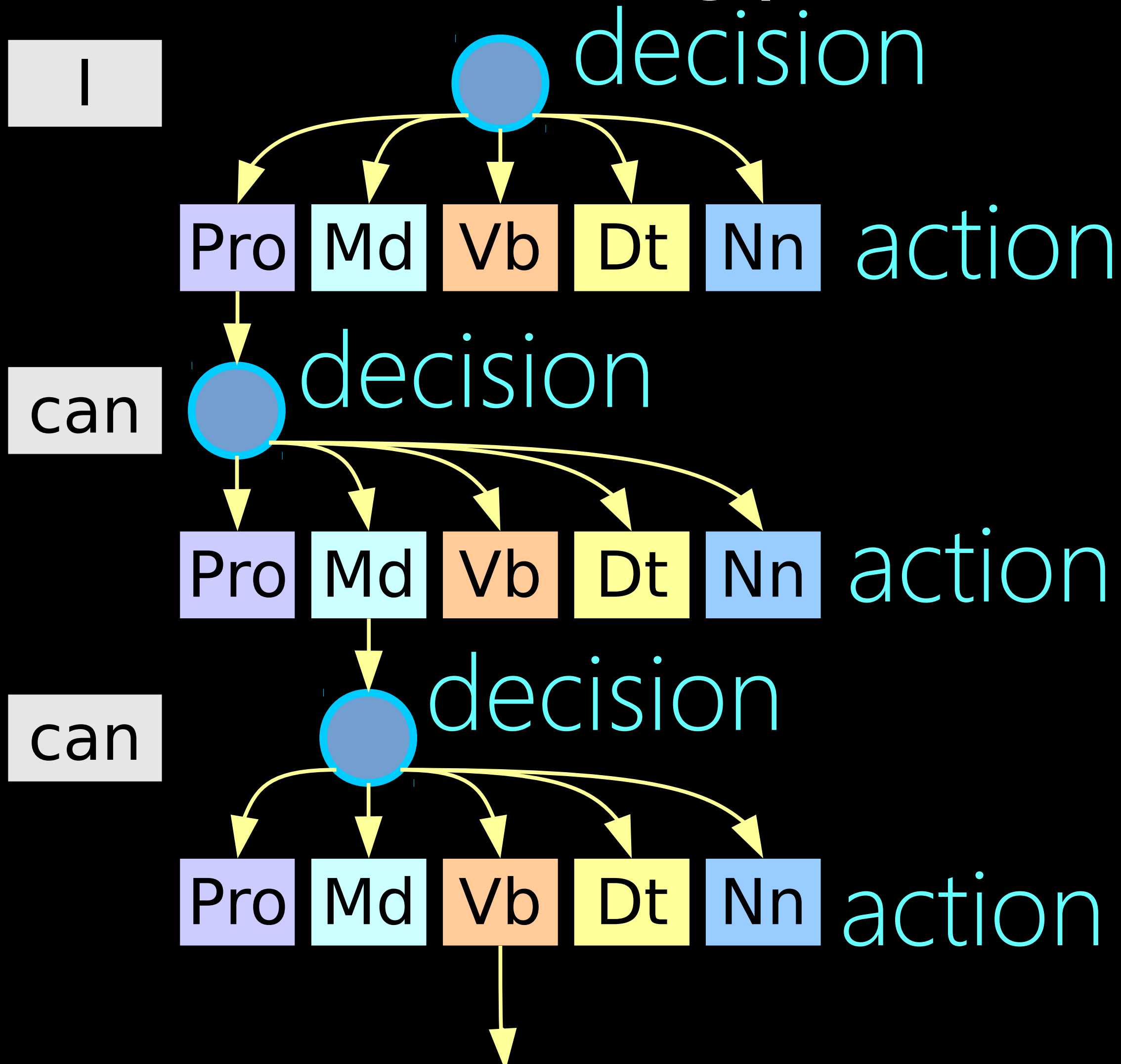


(we'll come back to this case later...)

image credit: Wikipedia & Ashutosh Saxena

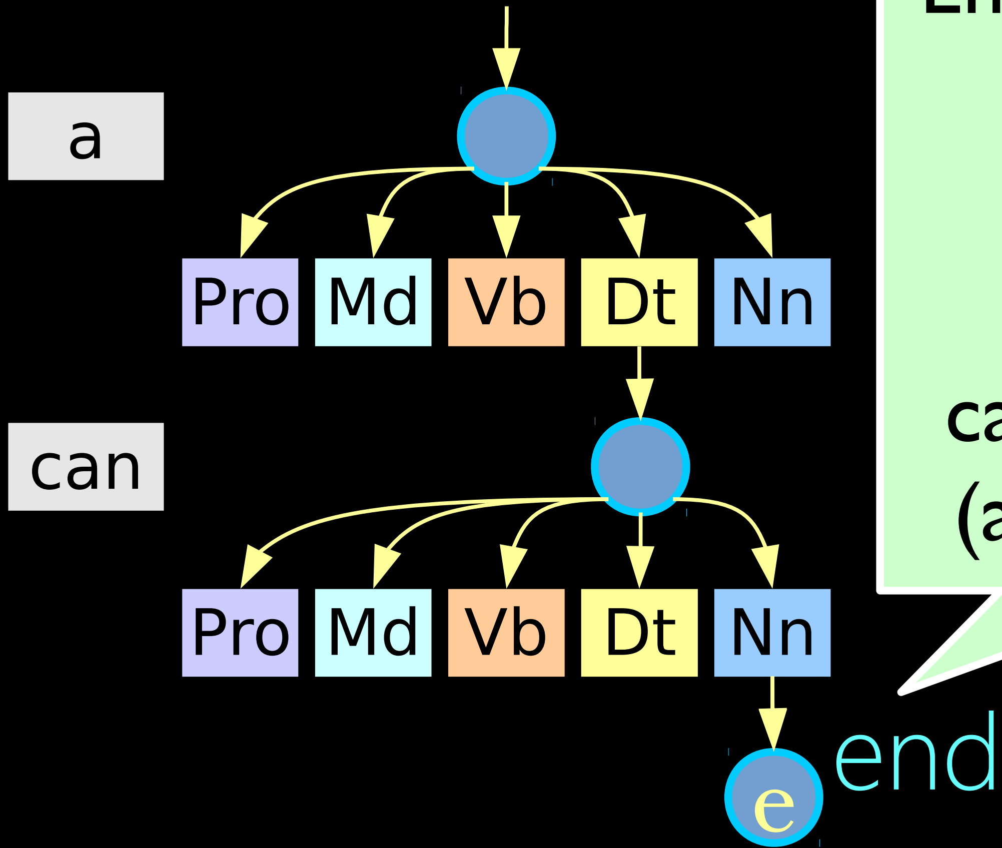
Search spaces

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



Search spaces

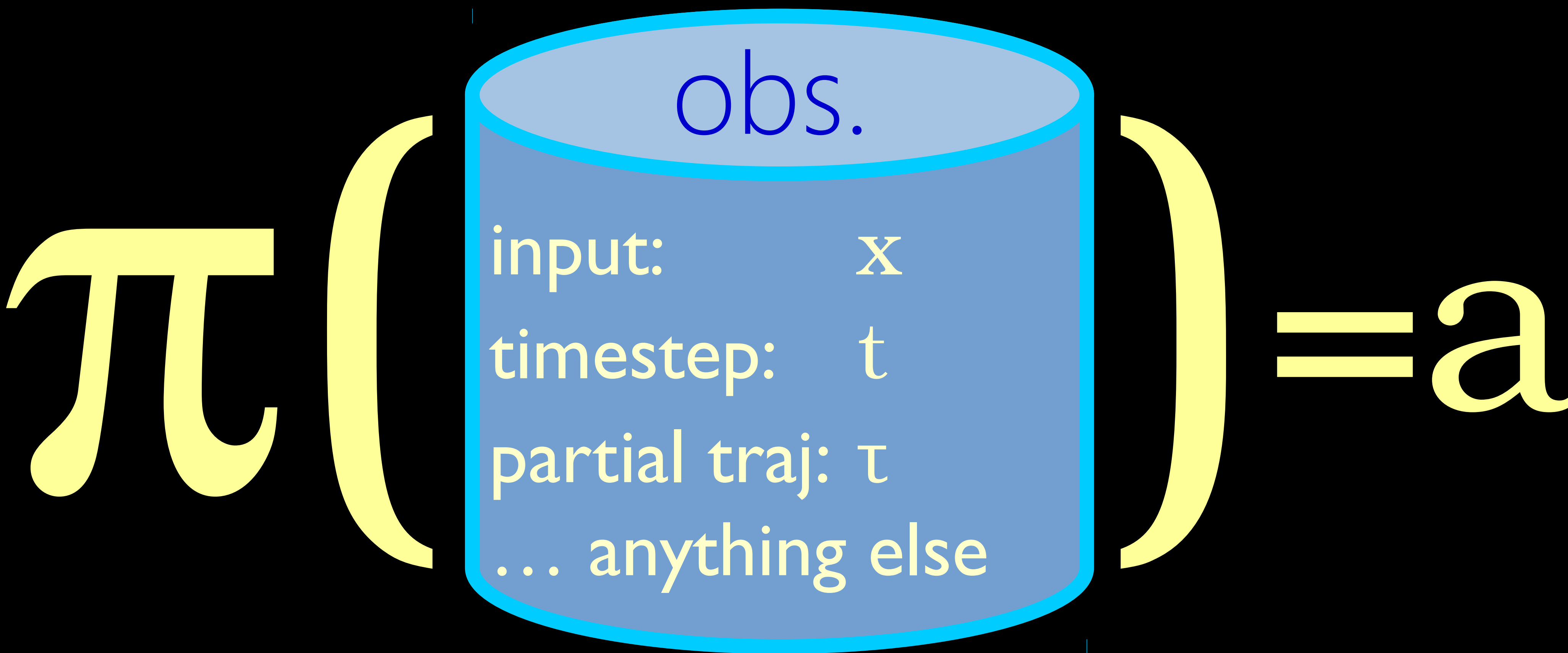
- When y decomposes in an ordered manner, a sequential decision making process emerges



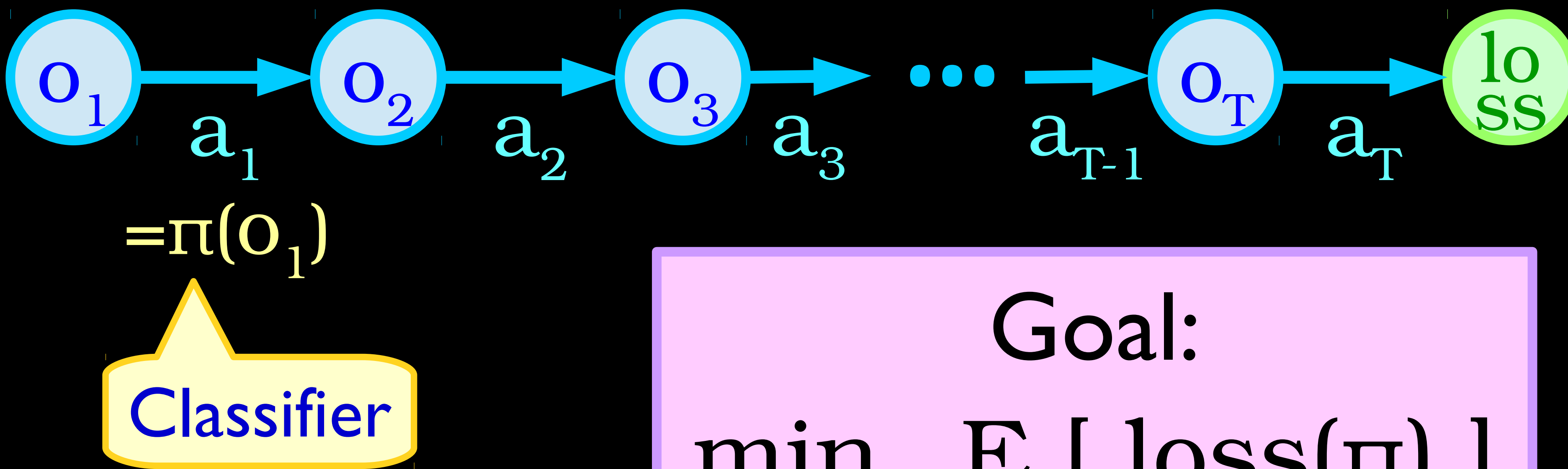
Encodes an output
$\hat{y} = \hat{y}(e)$
from which
$\mathrm{loss}(y, \hat{y})$
can be computed
(at training time)

Policies

- A policy maps observations to actions



Versus reinforcement learning



Goal:

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

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:

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

The *optimal action* is the corresponding a_t

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

Ingredients for learning to search

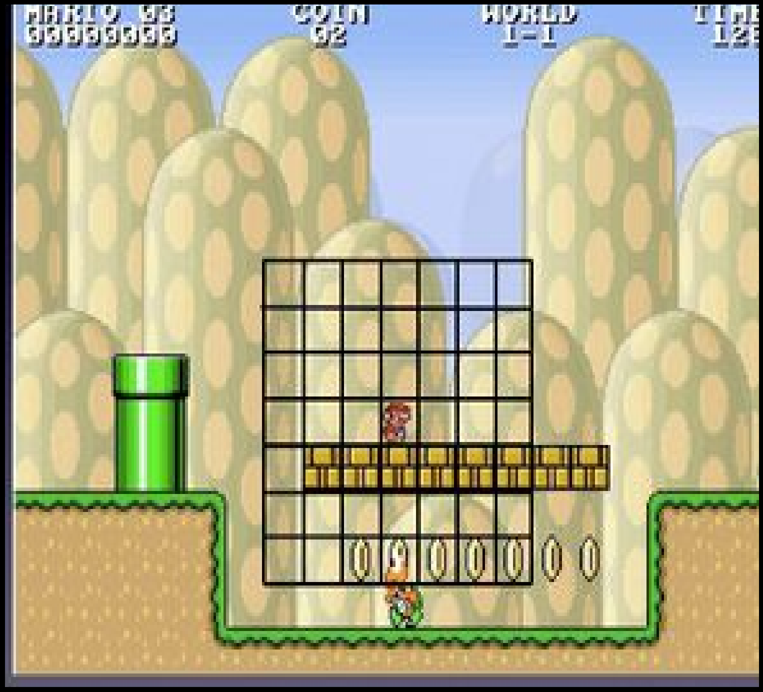
- Training data: $(x_n, y_n) \sim D$
- Output space: $Y(x)$
- Loss function: $\text{loss}(y, \hat{y})$

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

An analogy from playing Mario

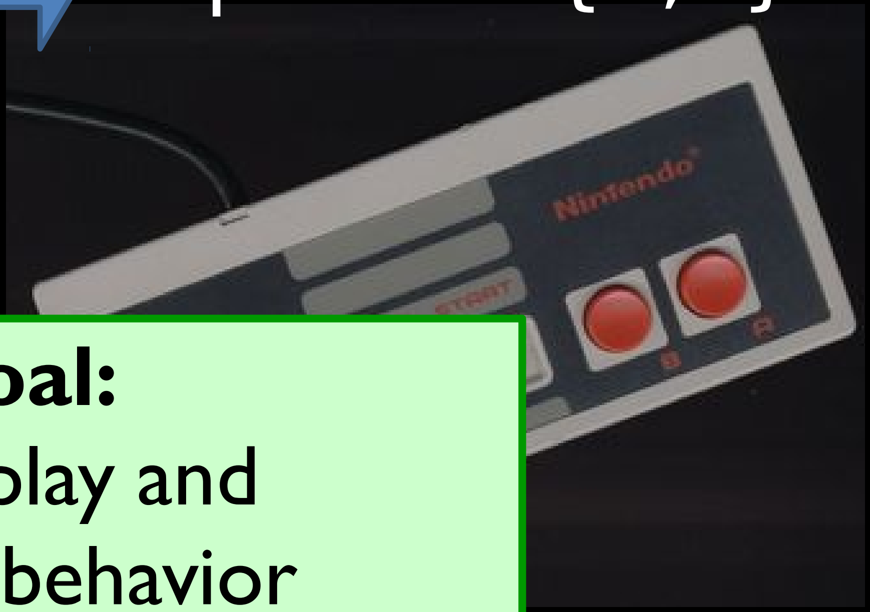
From Mario AI competition 2009

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

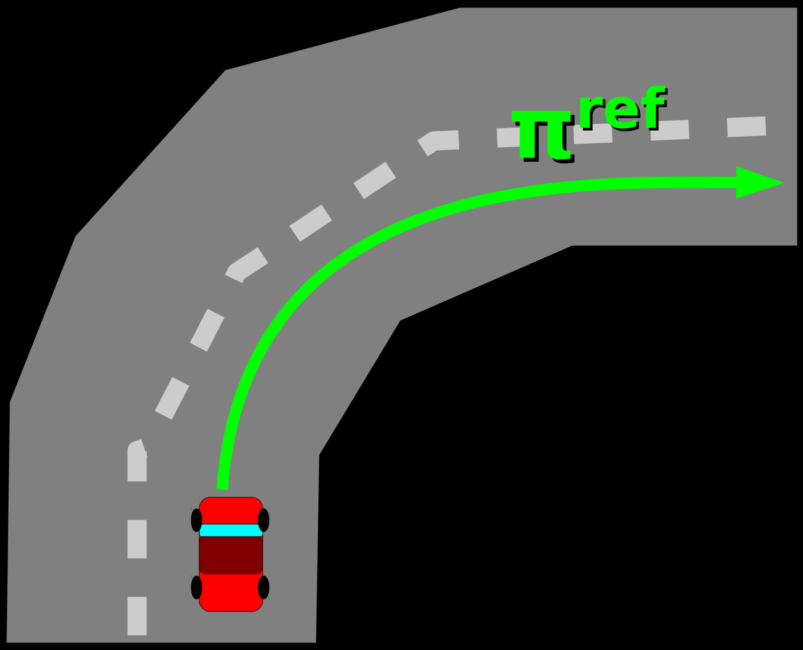
Training (expert)

Video credit: Stéphane Ross, Geoff Gordon and Drew Bagnell

Warm-up: Supervised learning

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

- **Let π play the game!**



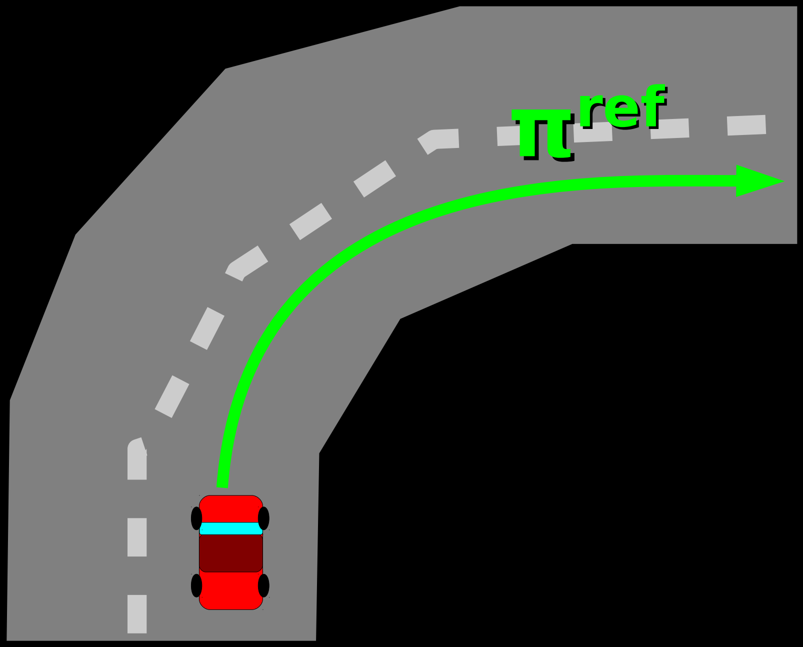
Test-time execution (sup. learning)

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

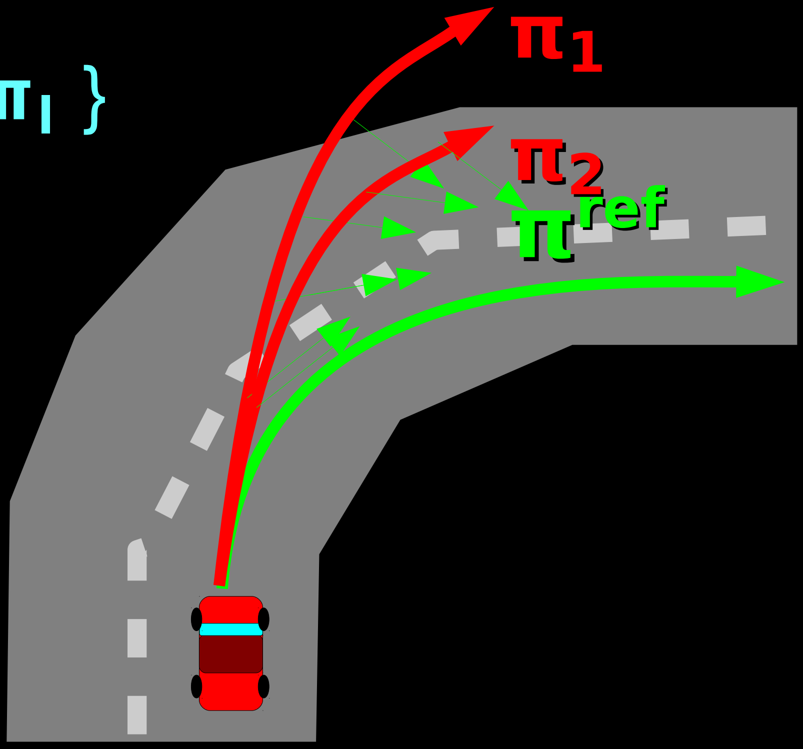
1. Collect trajectories from expert π^{ref}
2. Dataset $\mathbf{D_0}$ = { (o, π^{ref}(o,y)) | o ~ π^{ref} }
3. Train π_1 on $\mathbf{D_0}$
4. Collect new trajectories from π_1
   - But let the *expert* steer!
5. Dataset $\mathbf{D_1}$ = { (o, π^{ref}(o,y)) | o ~ π_1 }
6. Train π_2 on $\mathbf{D_0} \cup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

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

If $N = T \log T$,

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

for some n



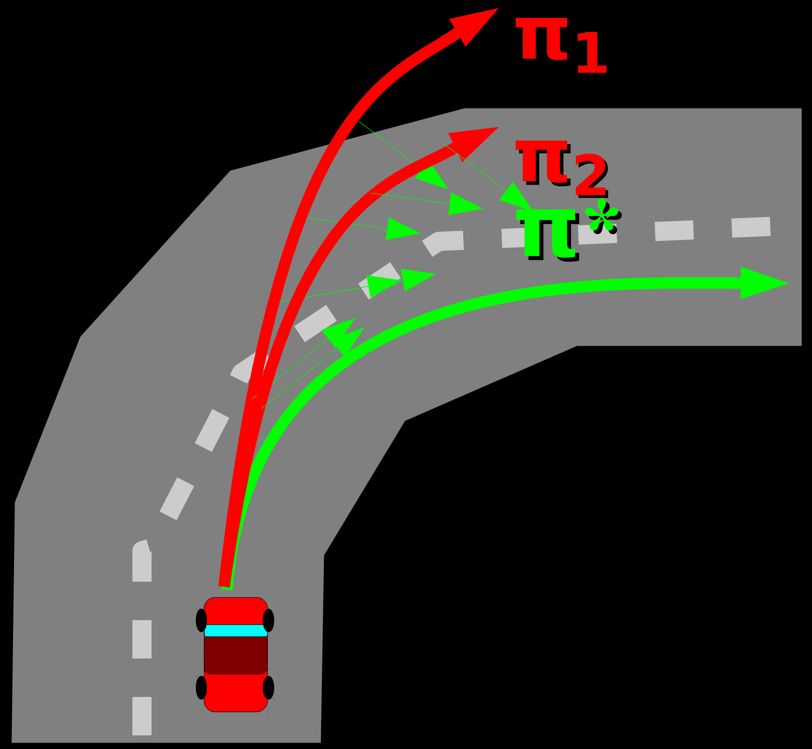
Test-time execution (DAgger)

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: $h : x \to [K]$

Multiclass classification

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

Cost-sensitive classification

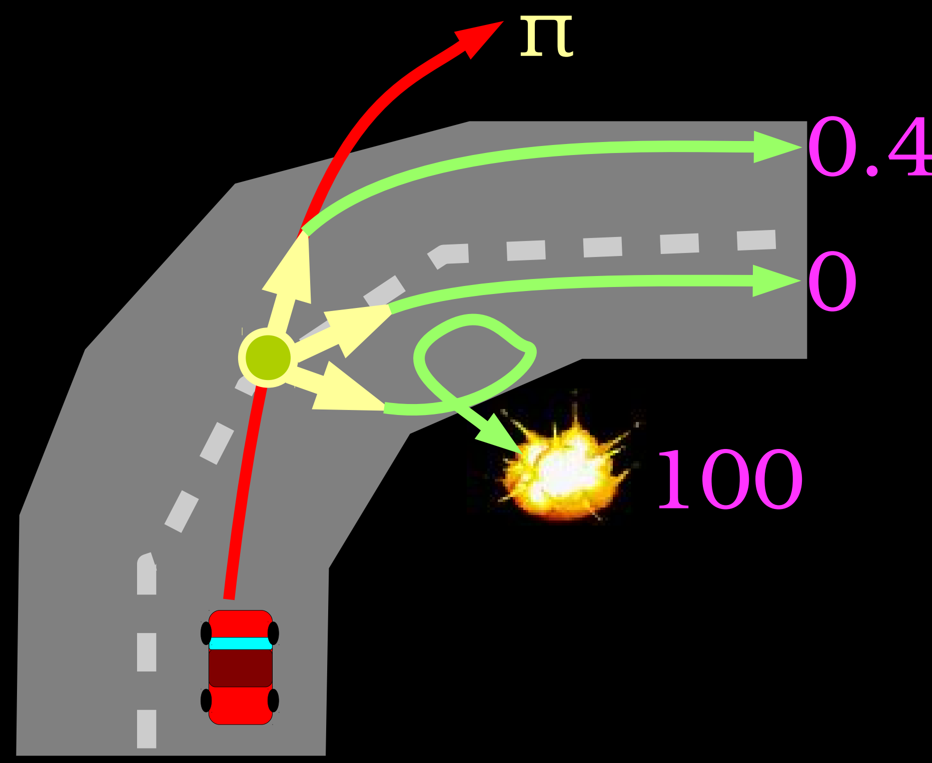
- Data: $(x,c) \in X \times [0,\infty)^K$
- Goal: $\min_h E_{(x,\vec{c})}\ [\ c_{h(x)}\ ]$

Learning to search: AggraVaTe

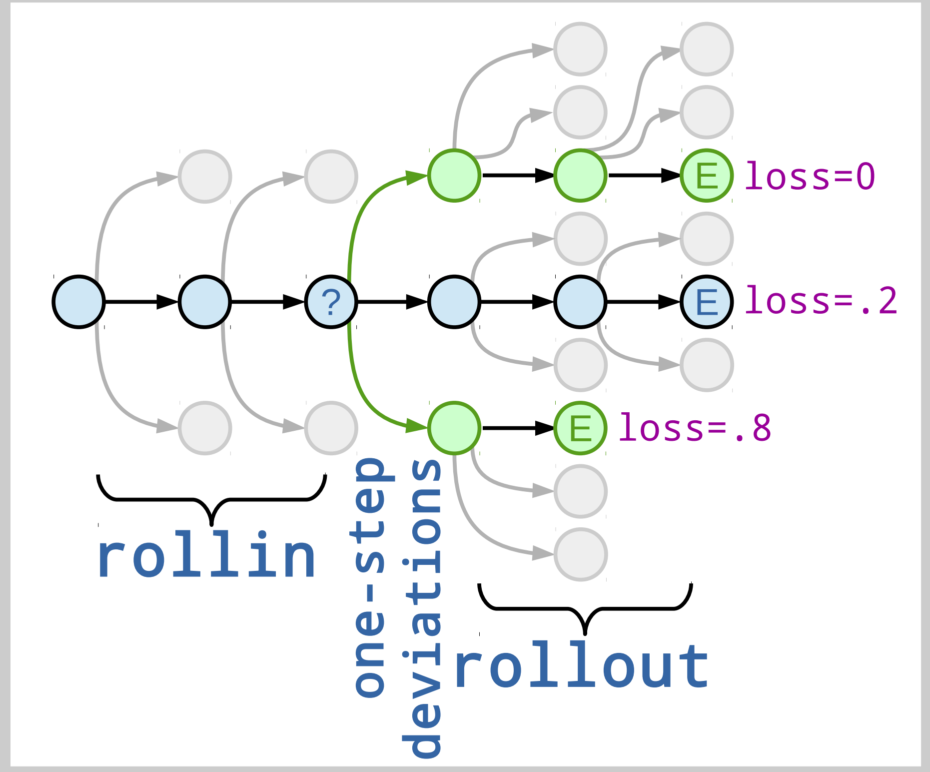
1. Let learned policy π drive for t timesteps to obs. o
2. For each possible action a:
    - Take action a, and let expert π^{ref} drive the rest
    - Record the overall loss, c_a
3. Update π based on example:

   $$(o, \langle c_1, c_2, \ldots, c_K \rangle)$$

4. Goto (1)



Learning to search: AggraVaTe

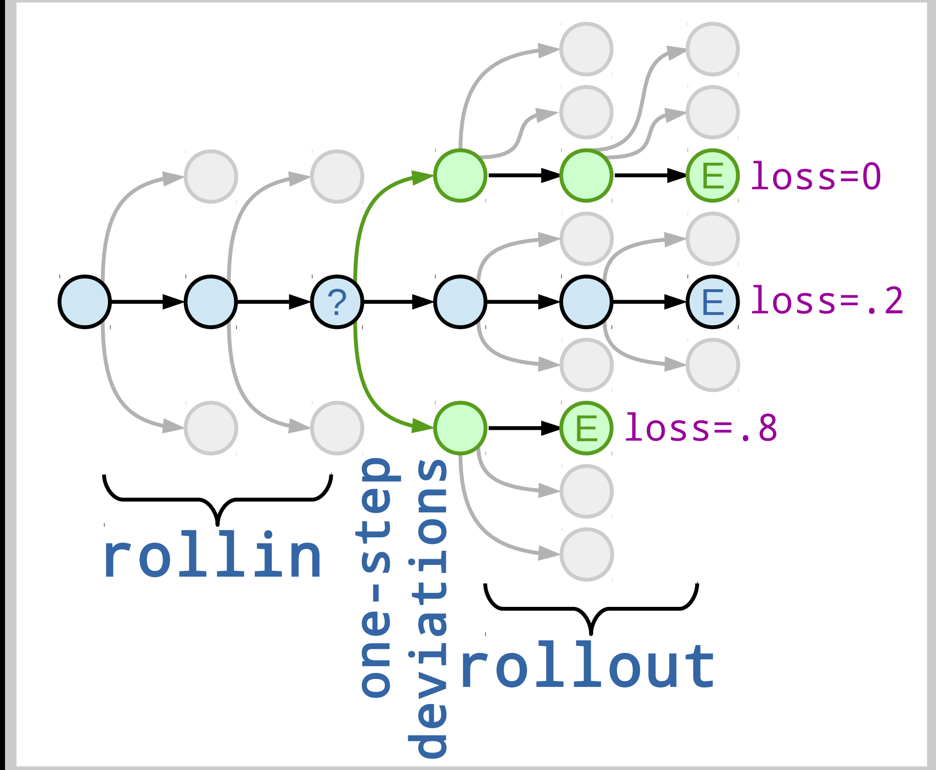


1. Generate an initial trajectory using the *current policy*
2. Foreach decision on that trajectory with obs. o:
   a) Foreach possible action a (one-step deviations)
      i. Take that action
      ii. Complete this trajectory using reference policy
      iii. Obtain a final loss, c_a

   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 reference policy
      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:

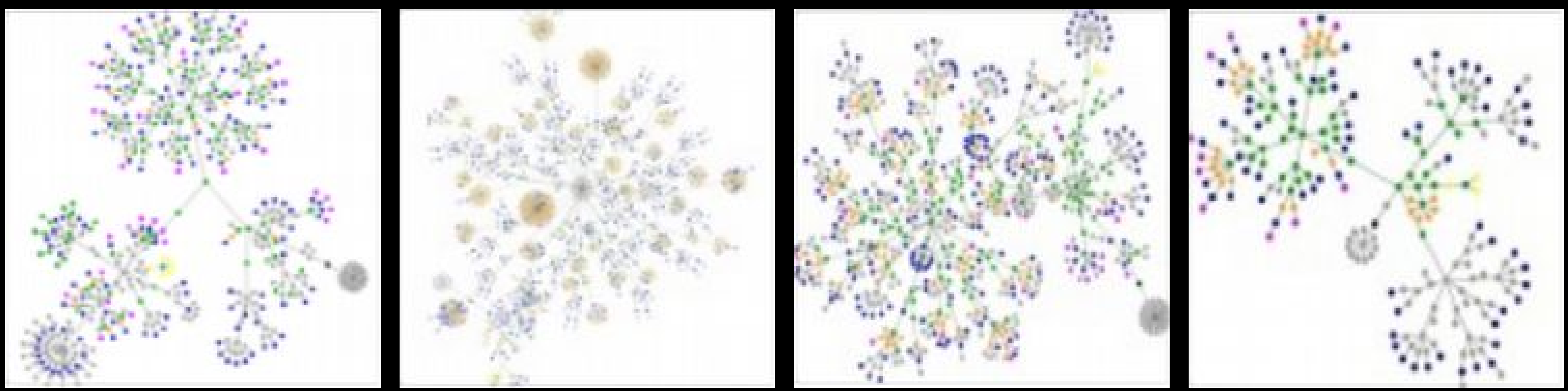
```
   x = the monster ate the sandwich
ŷ_Dt = Dt  Dt
ŷ_Nn = Dt  Nn
ŷ_Vb = Dt  Vb
```

- 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

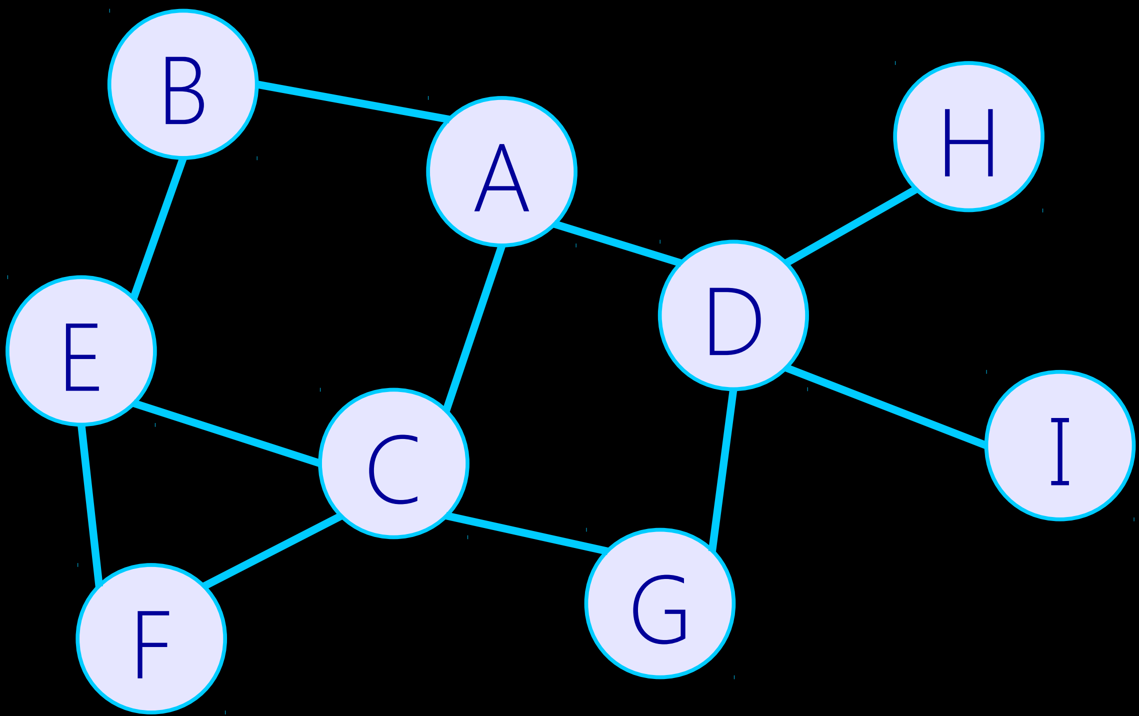


U Wisconsin U Washington U Texas Cornell

- Node features: text on web page
- Edge features: text in hyperlinks

Example II: Graph labeling

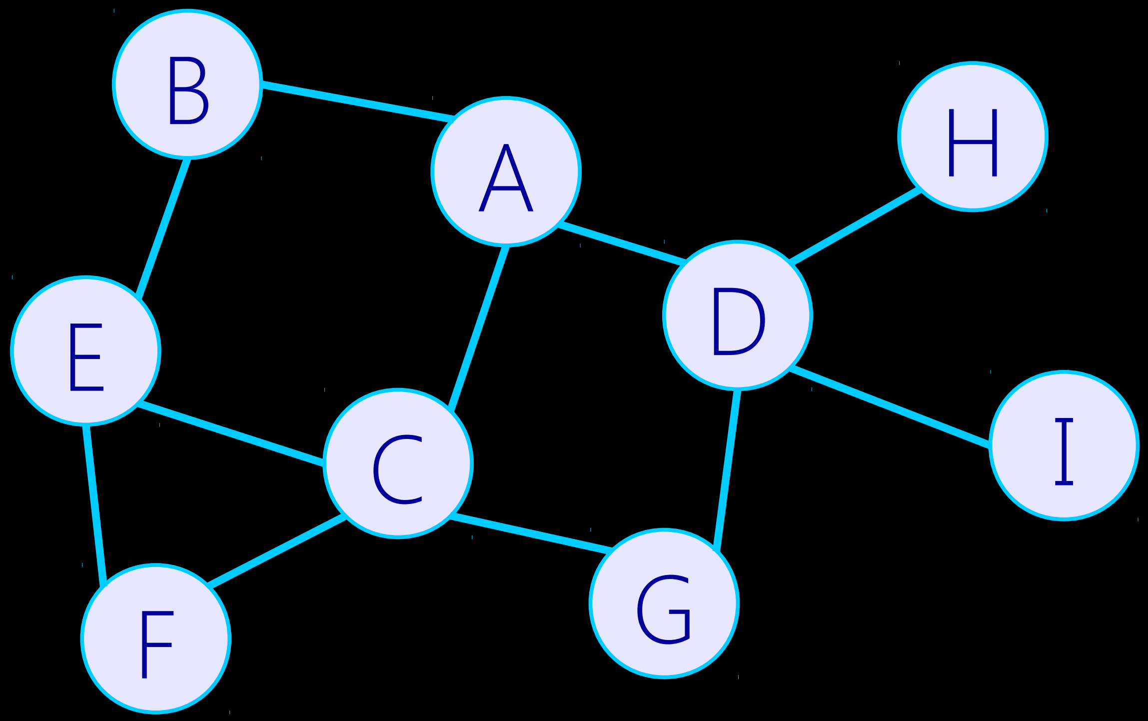
- How to linearize? Like belief propagation might!
- Pick a starting node (A), run BFS out
- Alternate outward and inward passes



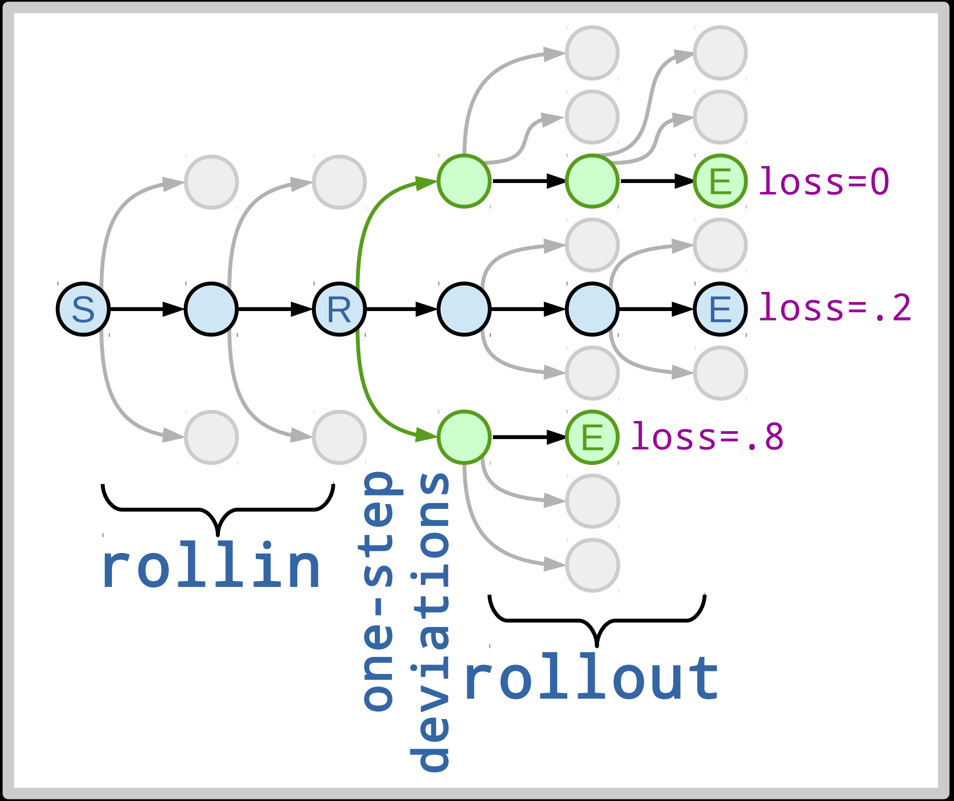
Linearization:
ABCDEFGHI
HGFEDCBA
BCDEFGHI
HGFEDCBA
. . .

Example II: Graph labeling

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



How to train?



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:

$$(\Phi(R), \langle c_a \rangle_{a \in A})$$

Choosing the rollin/rollo

- Three basic options:
  - The currently learned policy (“lear
  - The reference/expert policy (“ref
  - A stochastic mixture of these ix”)

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

Out / In	Ref	ix	Learn
Ref	Inconsistent One-step fai	nconsistent	Inconsistent
Learn	One-step fail	**Good**	Really hard

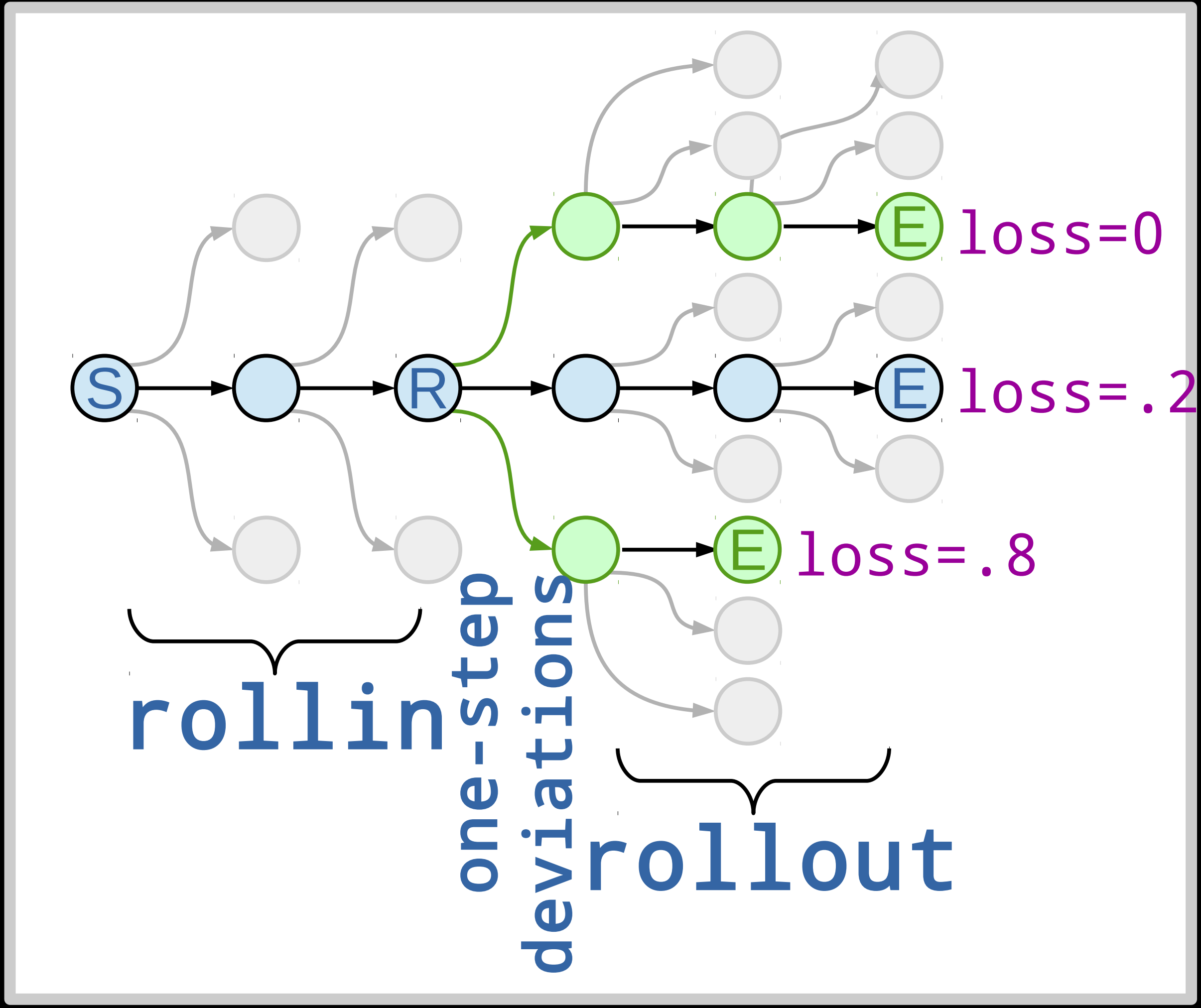
Sanity check: which of these is closest to DAgger?

From Mario back to POS tagging

```
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
```

- The oracle (reference) policy gives the true label for the corresponding word
- Sanity check: why/when is this optimal?

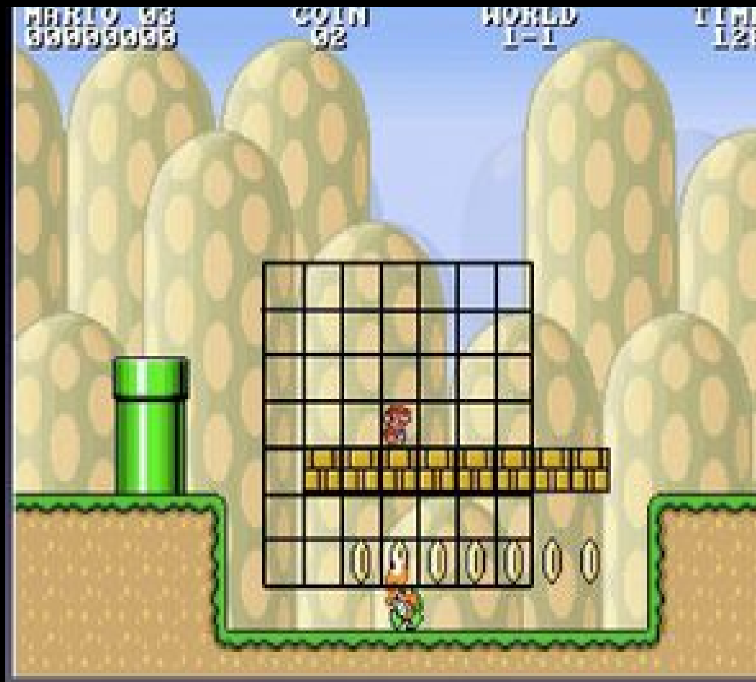
Optimal policies



- Given:
  - Training input *x*
  - State R
  - Loss function
- Return the action a that:
  - (If all future actions are taken optimally)
  - Minimizes the corresponding loss

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

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, TACL 2014