

Advances in Structured Prediction



John Langford
Microsoft Research
jl@hunch.net



Hal Daumé III
U Maryland
me@hal3.name

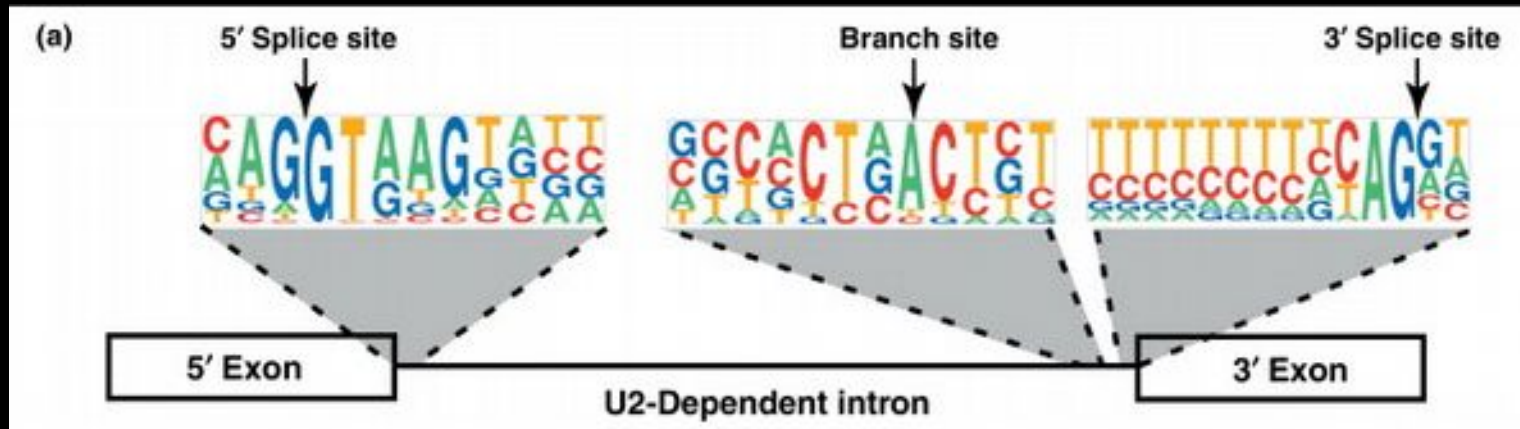
Slides and more: <http://hunch.net/~12s>

Examples of ~~structured joint~~ prediction

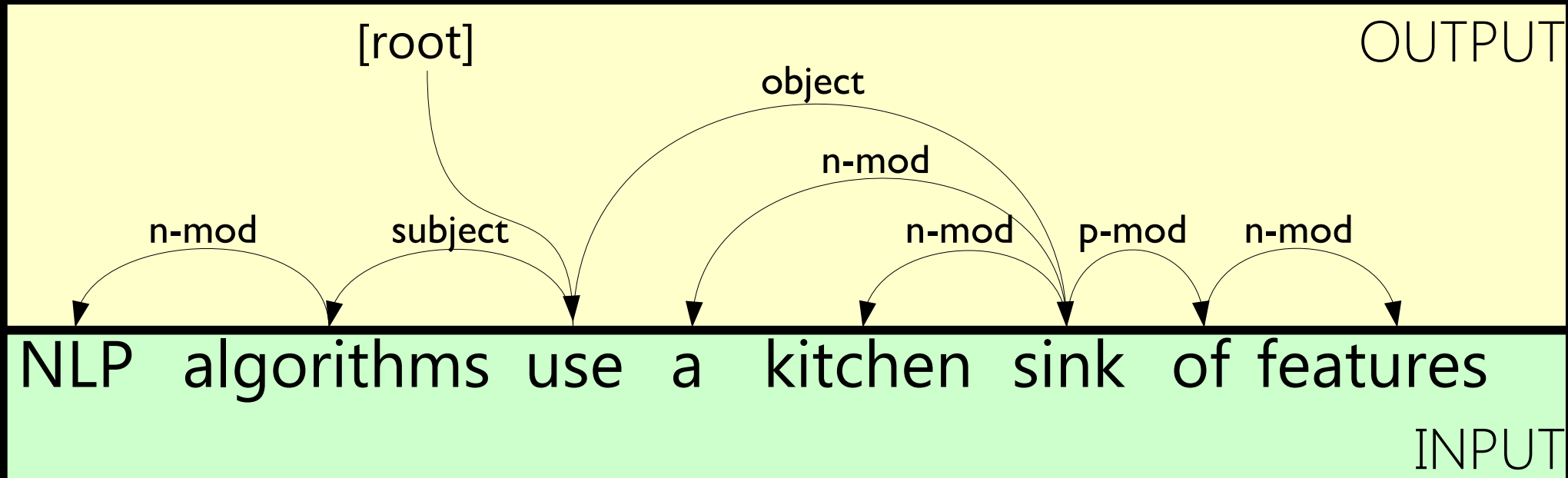
Sequence labeling

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

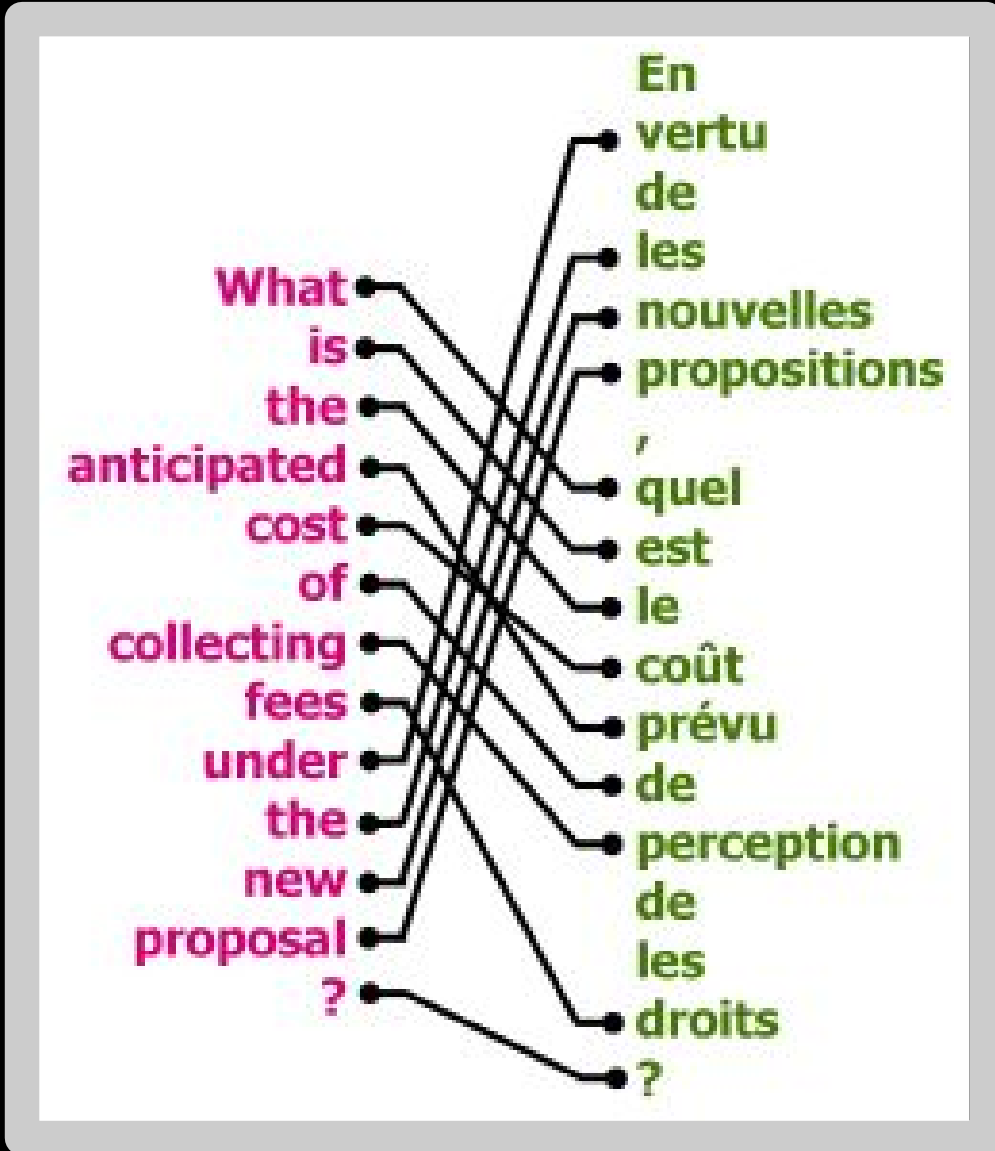
x = Yesterday I traveled to Lille
y = - PER - - LOC



Natural language parsing



(Bipartite) matching



Machine translation



The image shows a screenshot of the Google Translate web interface. At the top left is the Google logo, and to its right is the word "Translate" in a blue box. Below this, a light blue banner states "This text has been [automatically translated](#) from Arabic:". Underneath is a text area containing the English translation of an Arabic text. Below the text area is a section titled "Translate text" with a text area containing the original Arabic text. At the bottom, there is a dropdown menu showing "from Arabic to English BETA" and a "Translate" button.

Google Translate

This text has been [automatically translated](#) from Arabic:

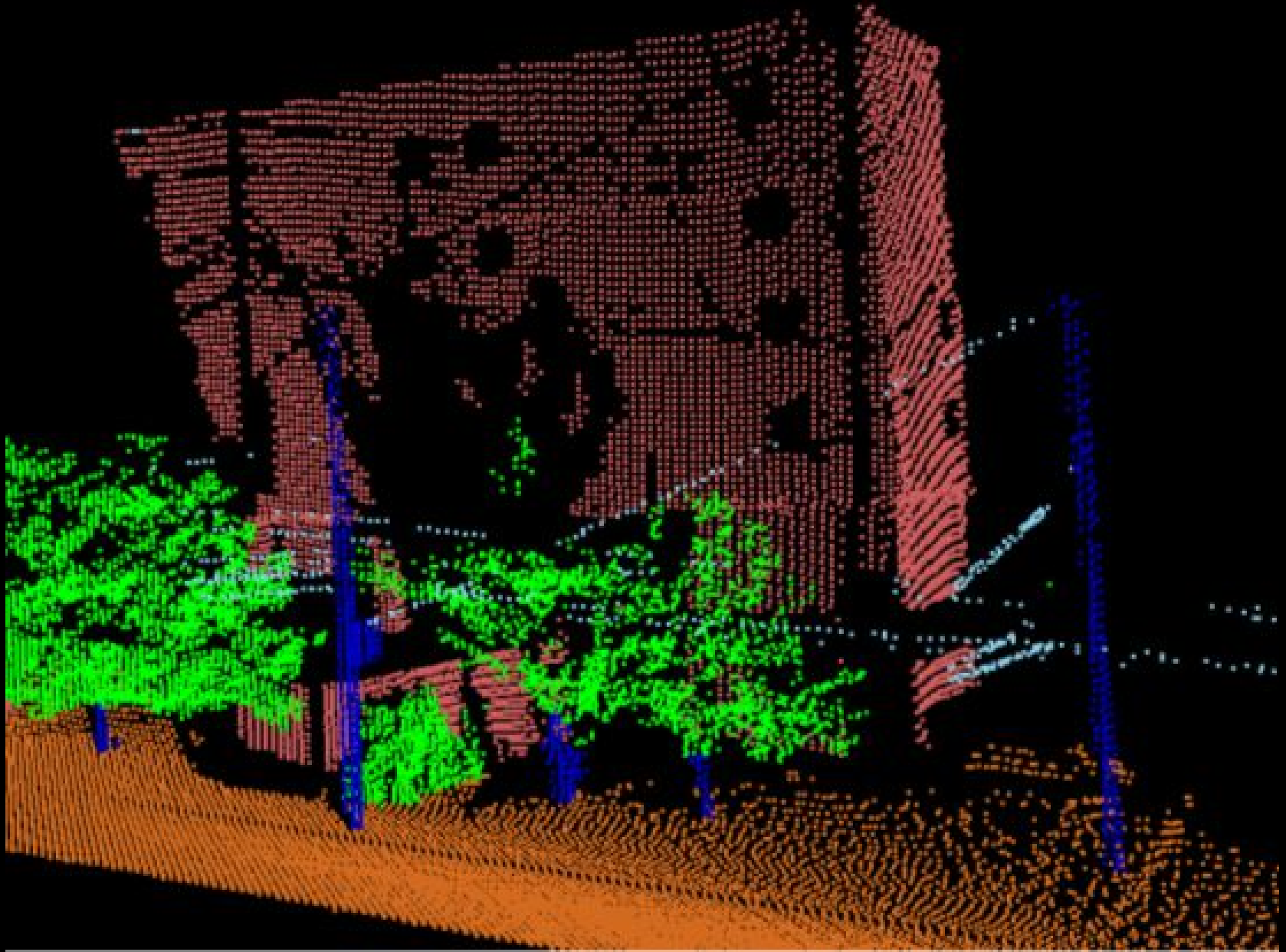
Moscow stressed tone against Iran on its nuclear program. He called Russian Foreign Minister Tehran to take concrete steps to restore confidence with the international community, to cooperate fully with the IAEA. Conversely Tehran expressed its willingness

Translate text

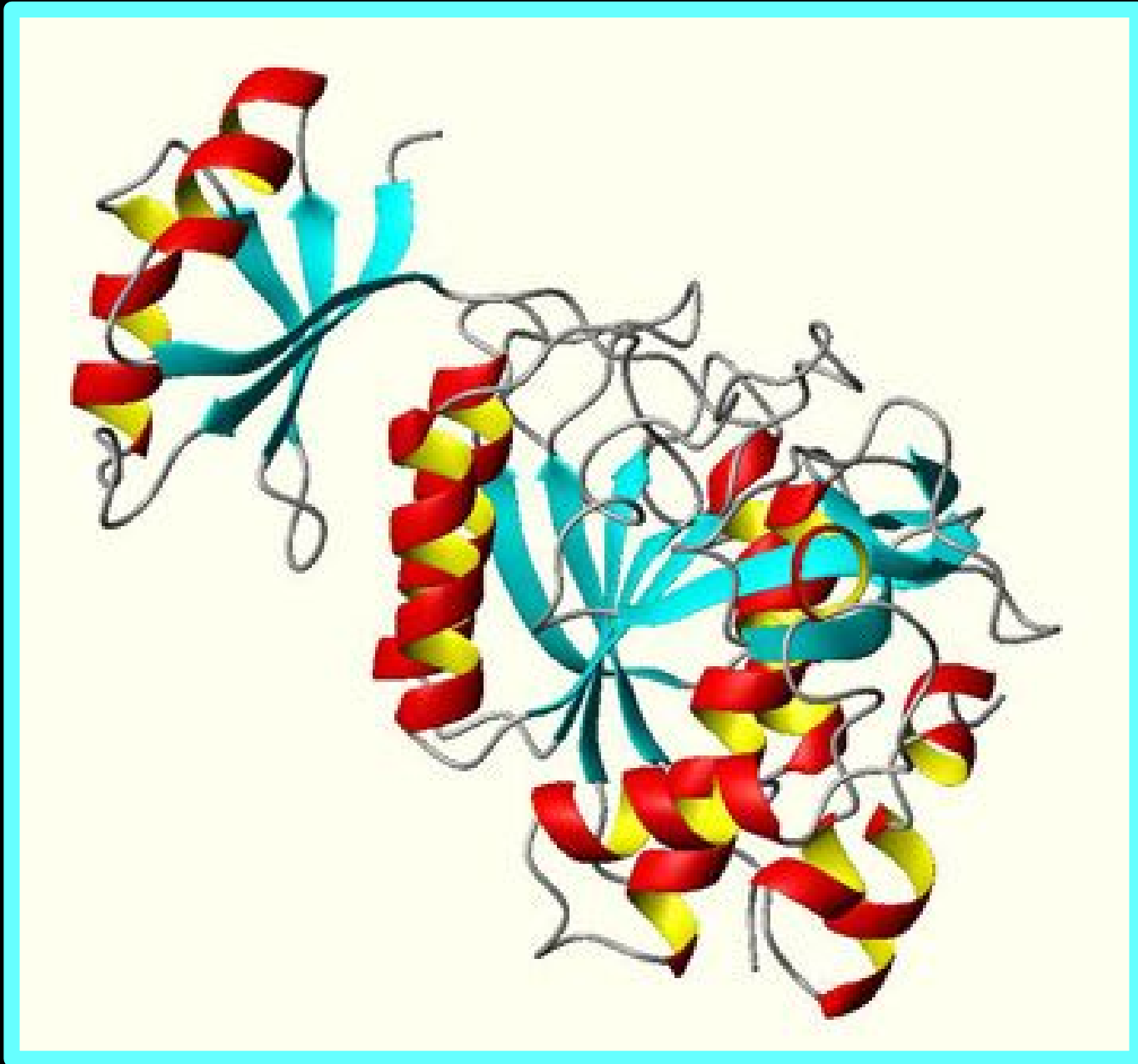
شدت موسكو لهجتها ضد إيران بشأن برنامجها النووي. ودعا وزير الخارجية الروسي طهران إلى اتخاذ خطوات ملموسة لاستعادة الثقة مع المجتمع الدولي والتعاون الكامل مع الوكالة الذرية. بالمقابل أبدت طهران استعدادها لاستئناف السماح بعمليات التفتيش المفاجئة بشرط إسقاط مجلس الأمن ملفها النووي.

from Arabic to English BETA Translate

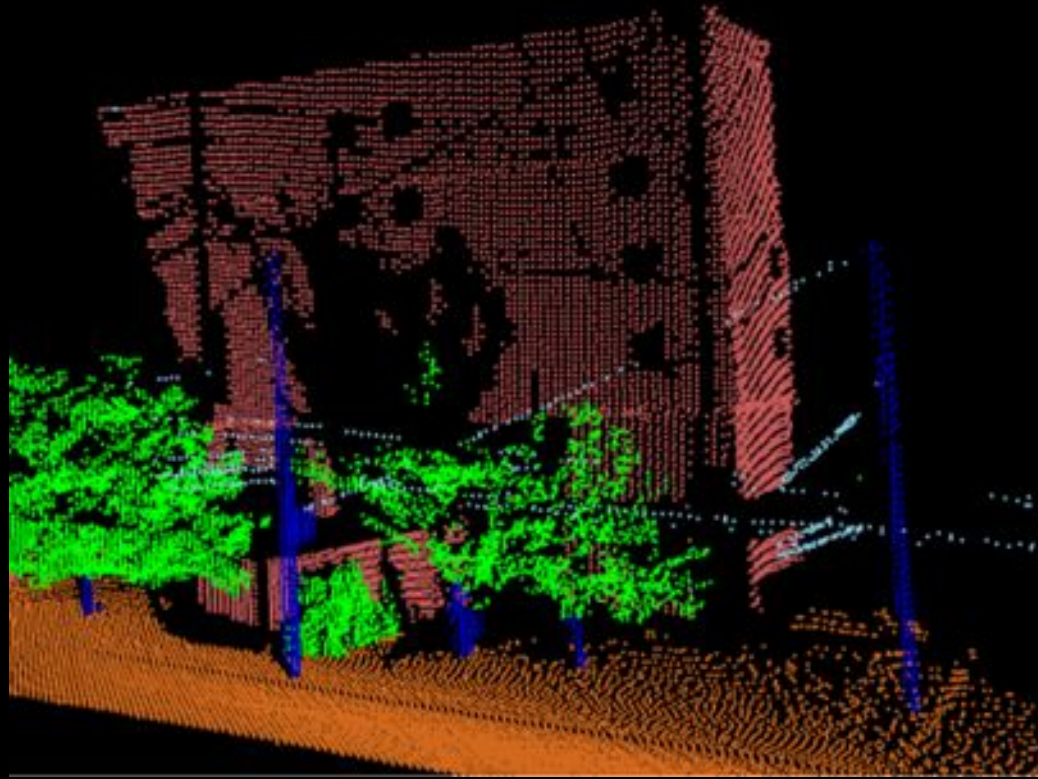
Image segmentation



Protein secondary structure prediction

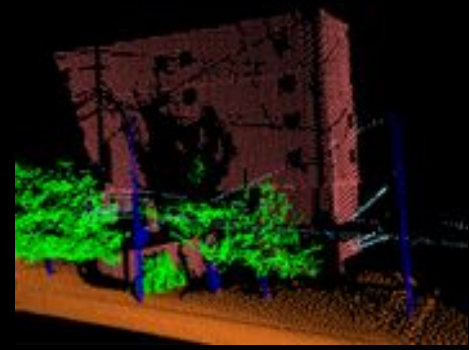


Standard solution methods



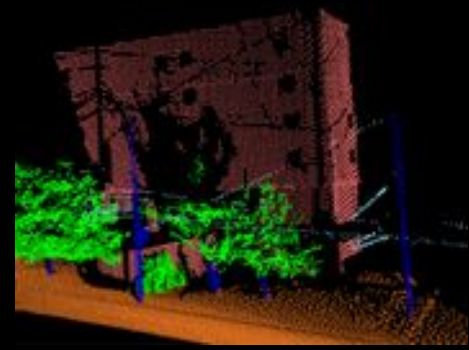
1. Each prediction is independent
2. Shared parameters via “multitask learning”
3. Assume tractable graphical model; optimize
4. Hand-crafted

Predicting independently



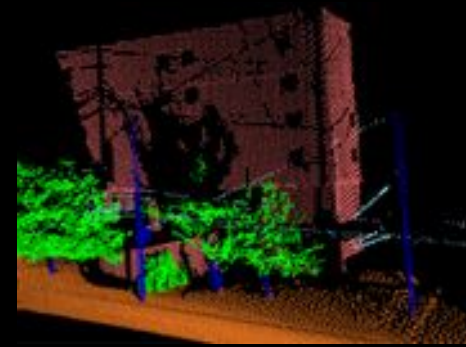
- h : features of nearby voxels \rightarrow class
- Ensure output is coherent at test time
- ✓ Very simple to implement, often efficient
- ✗ Cannot capture correlations between predictions
- ✗ Cannot optimize a joint loss

Prediction with multitask bias



- h : features \rightarrow (hidden representation)
 \rightarrow 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 optimize:
 - $p(\text{true labels} \mid \text{input})$ *or*
 - cvx u.b. on $\text{loss}(\text{true labels}, \text{predicted labels})$

- ✓ **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: $\mathbf{x} \in X$ 

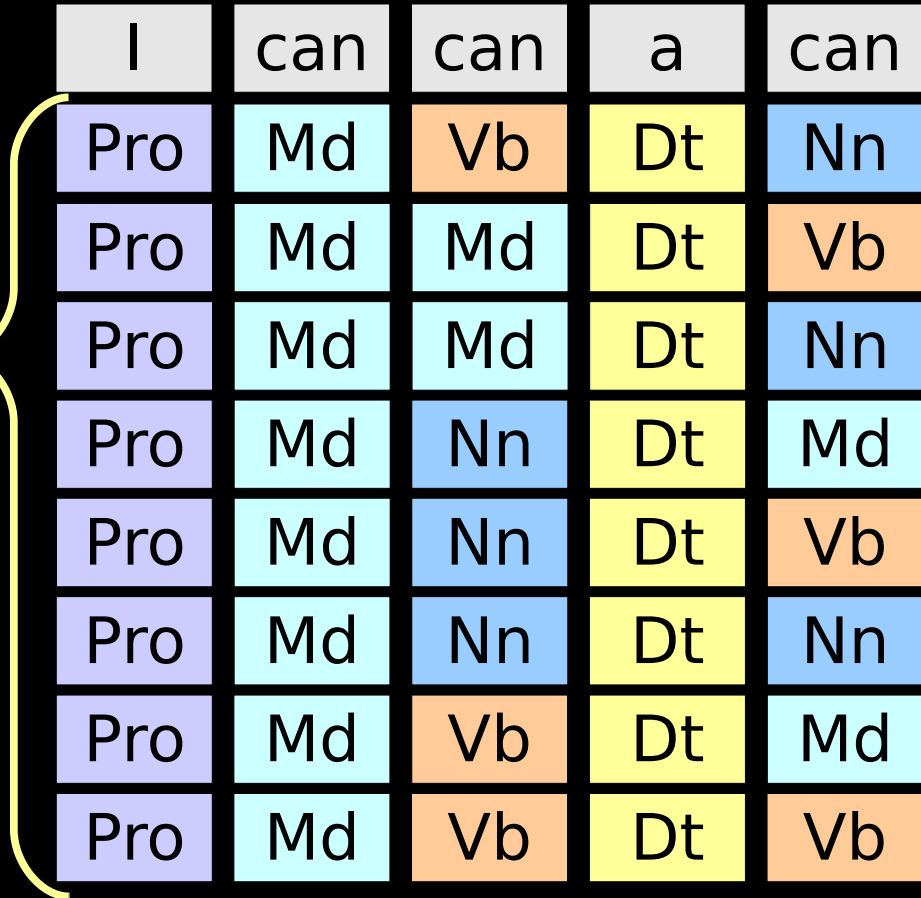
• Truth: $\mathbf{y} \in Y(\mathbf{x})$ 

• Outputs: $Y(\mathbf{x})$ 

• Predicted: $\hat{\mathbf{y}} \in Y(\mathbf{x})$

• Loss: $\text{loss}(\mathbf{y}, \hat{\mathbf{y}})$

• Data: $(\mathbf{x}, \mathbf{y}) \sim D$



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

Goal:

find $h \in H$
such that $h(\mathbf{x}) \in Y(\mathbf{x})$
minimizing

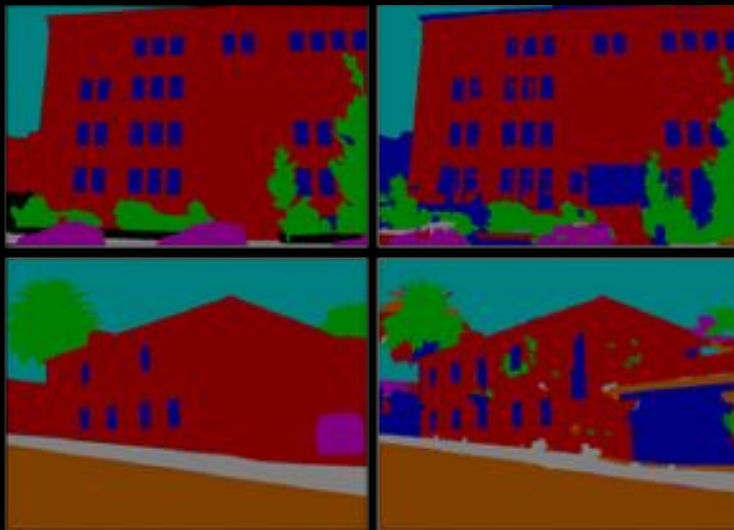
$$E_{(\mathbf{x}, y) \sim D} [\text{loss}(y, h(\mathbf{x}))]$$

based on N samples

$$(\mathbf{x}_n, y_n) \sim D$$

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

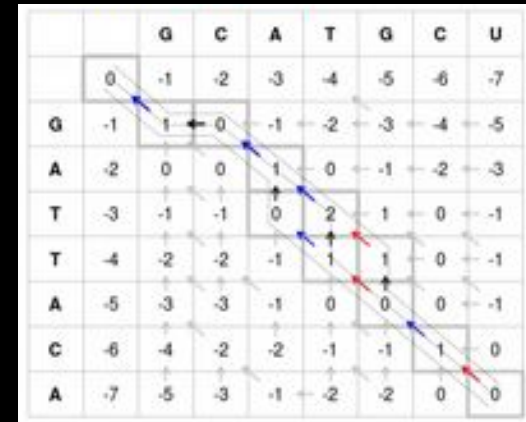


		G	C	A	T	G	C	U
	0	-1	-2	-3	-4	-5	-6	-7
G	-1	1	0	-1	-2	-3	-4	-5
A	-2	0	0	1	0	-1	-2	-3
T	-3	-1	-1	0	2	1	0	-1
T	-4	-2	-2	-1	1	1	0	-1
A	-5	-3	-3	-1	0	0	0	-1
C	-6	-4	-2	-2	-1	-1	1	0
A	-7	-5	-3	-1	-2	-2	0	0

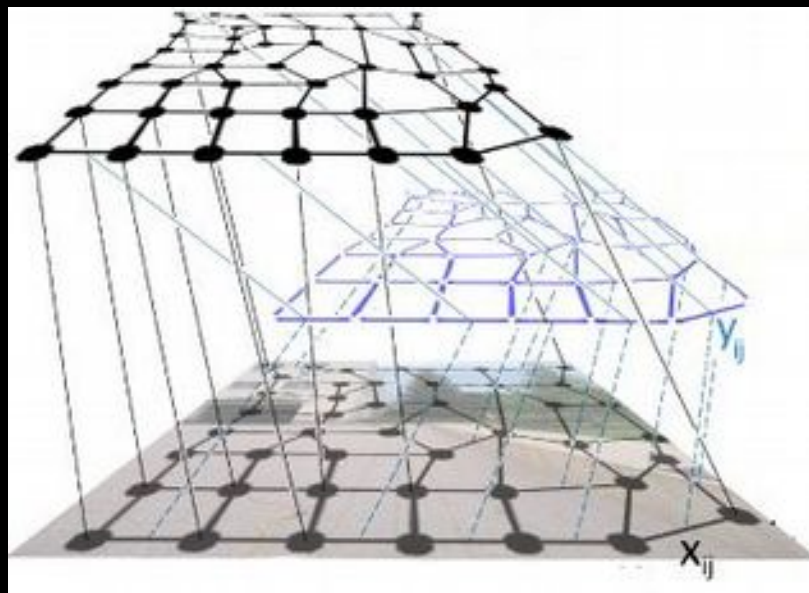
Decomposition of label

- Decomposition of y often implies an ordering

I	can	can	a	can
Pro	Md	Vb	Dt	Nn



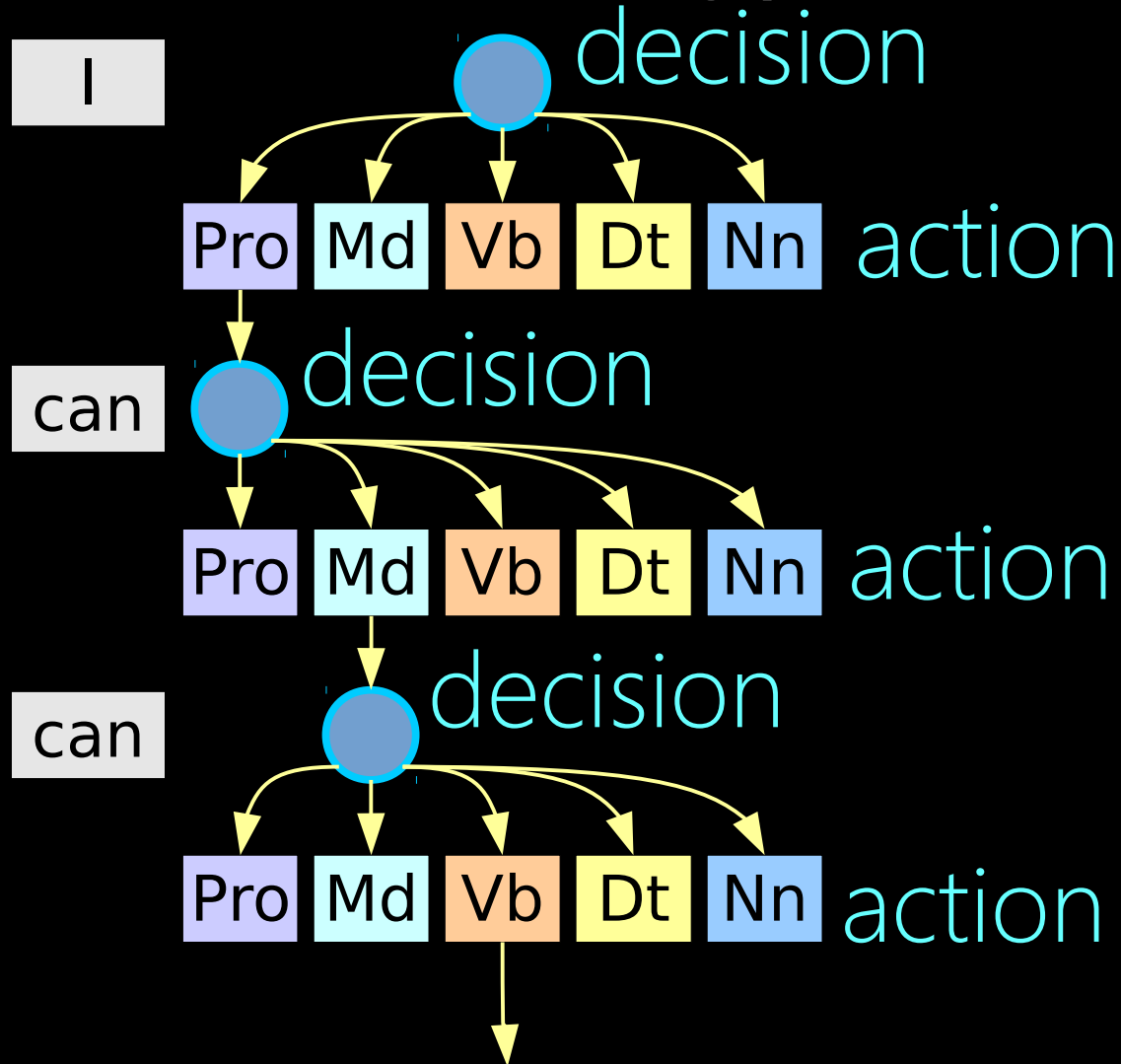
- But sometimes not so obvious....



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

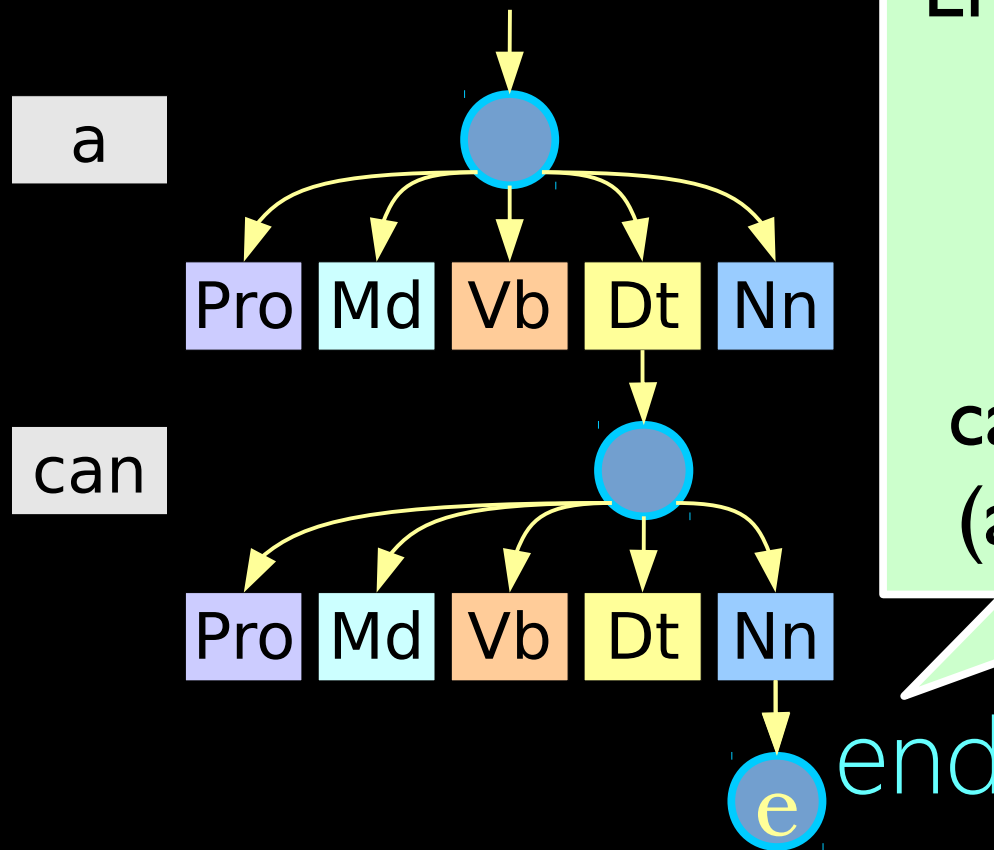
Search spaces

- 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

$$\hat{y} = \hat{y}(e)$$

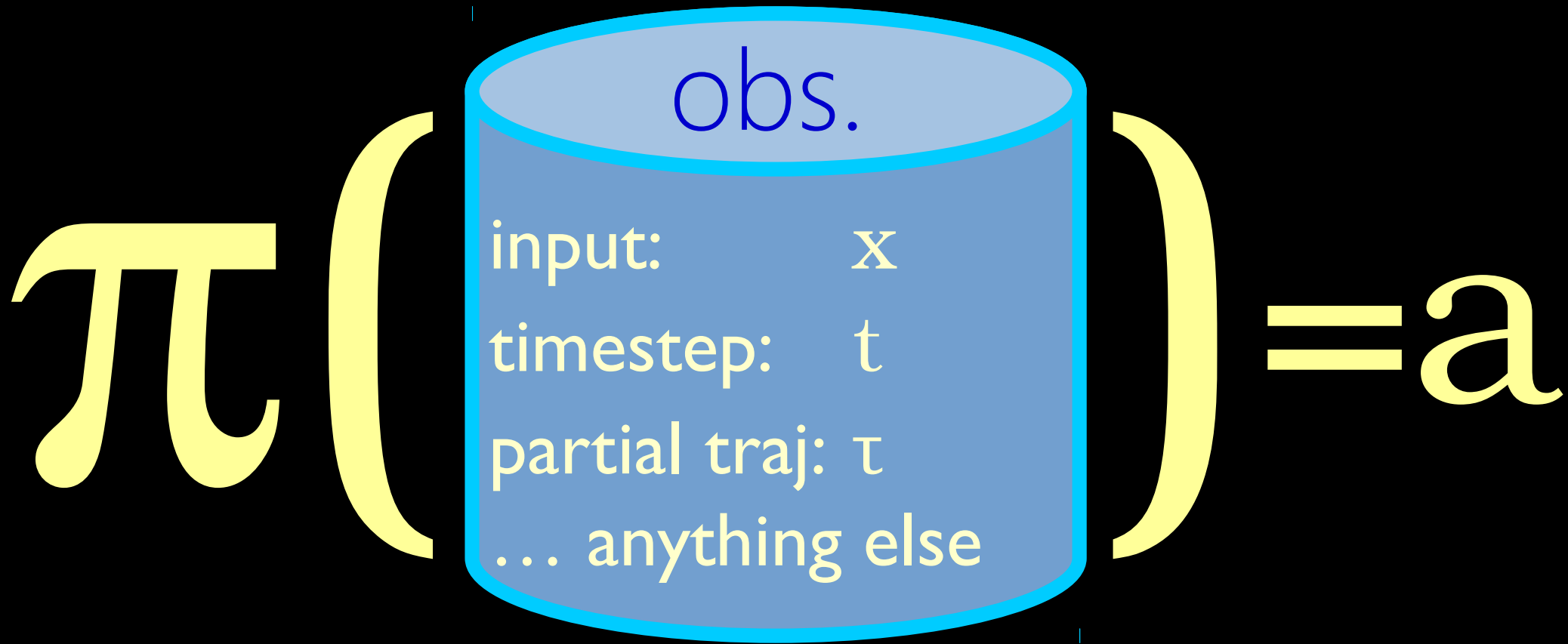
from which

$$\text{loss}(y, \hat{y})$$

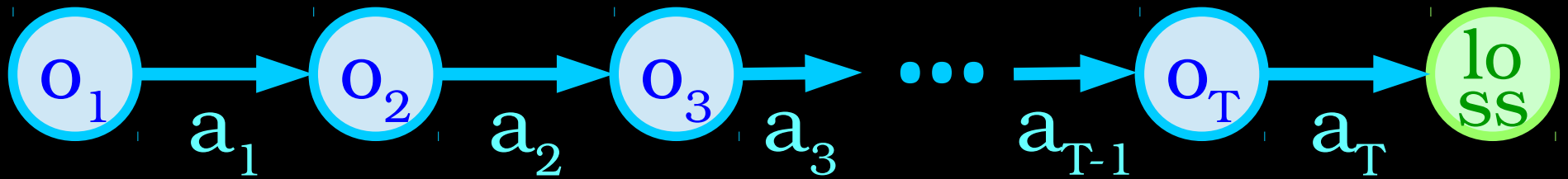
can be computed
(at training time)

Policies

- A policy maps observations to actions



Versus reinforcement learning



$=\pi(O_1)$

Classifier

Goal:

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

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 \rightarrow Reference policy

Given partial traj. $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_{t-1}$ and true label \mathbf{y}

The *minimum achievable loss* is:

$$\min_{(\mathbf{a}_t, \mathbf{a}_{t+1}, \dots)} \text{loss}(\mathbf{y}, \hat{\mathbf{y}}(\vec{\mathbf{a}}))$$

The *optimal action* is the corresponding \mathbf{a}_t

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

Ingredients for learning to search

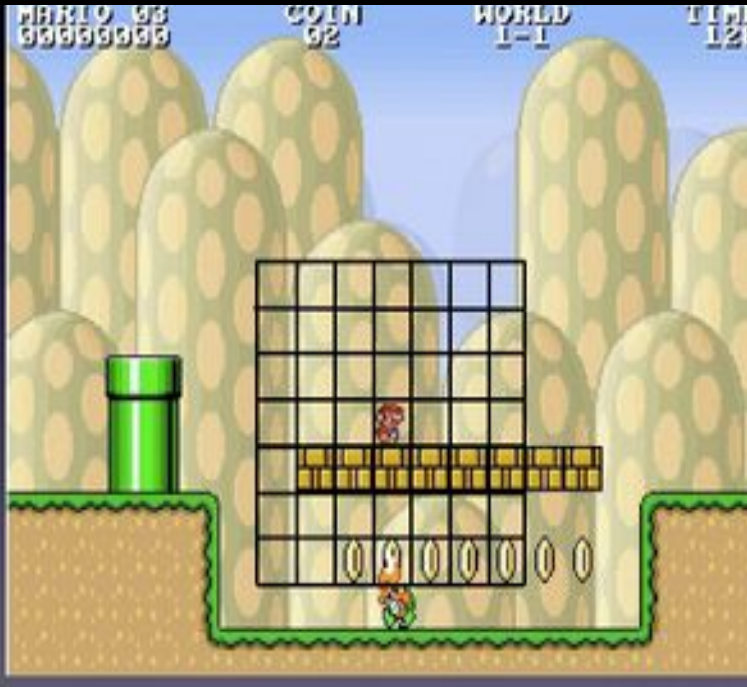
- Training data: $(\mathbf{x}_n, y_n) \sim \mathcal{D}$
- Output space: $Y(\mathbf{x})$
- Loss function: $\text{loss}(y, \hat{y})$

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

An analogy from playing Mario

From Mario AI competition 2009

Input:



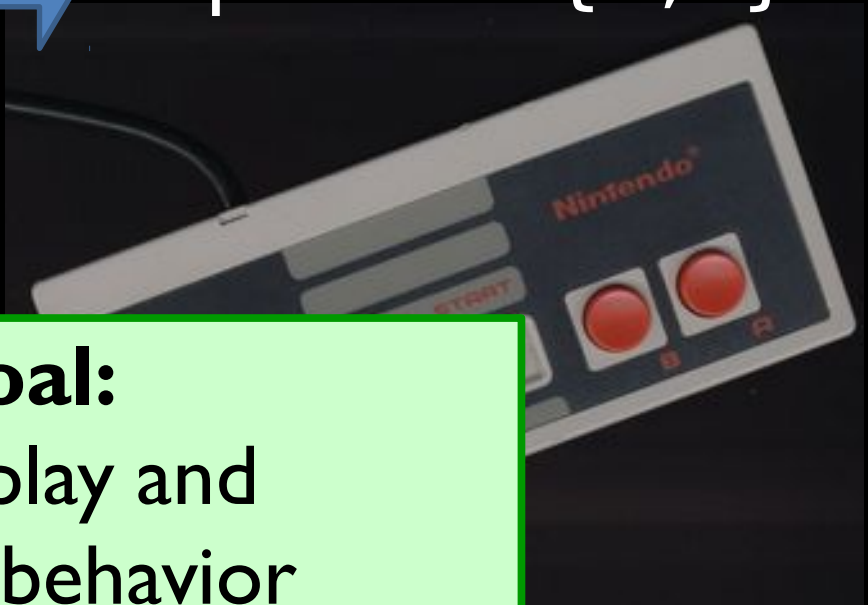
Output:

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

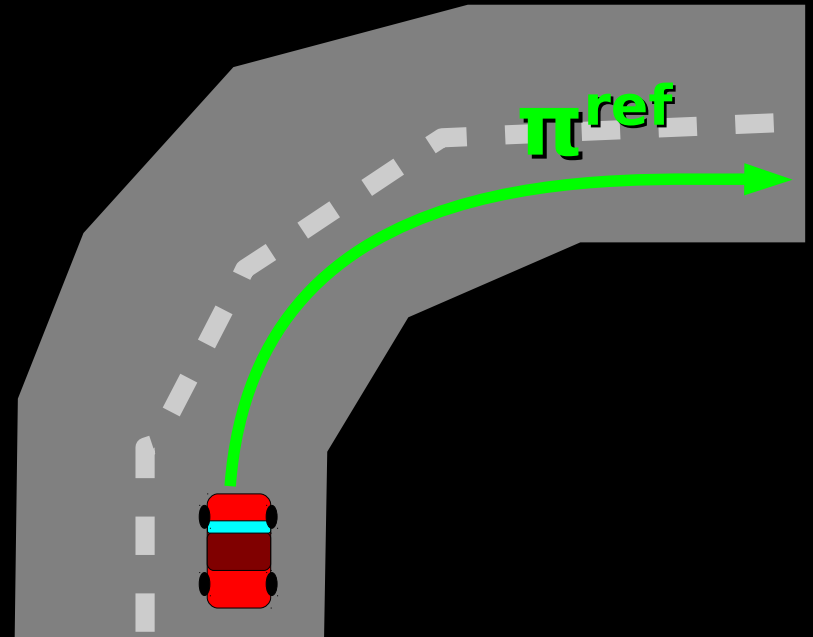


Training (expert)



Warm-up: Supervised learning

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



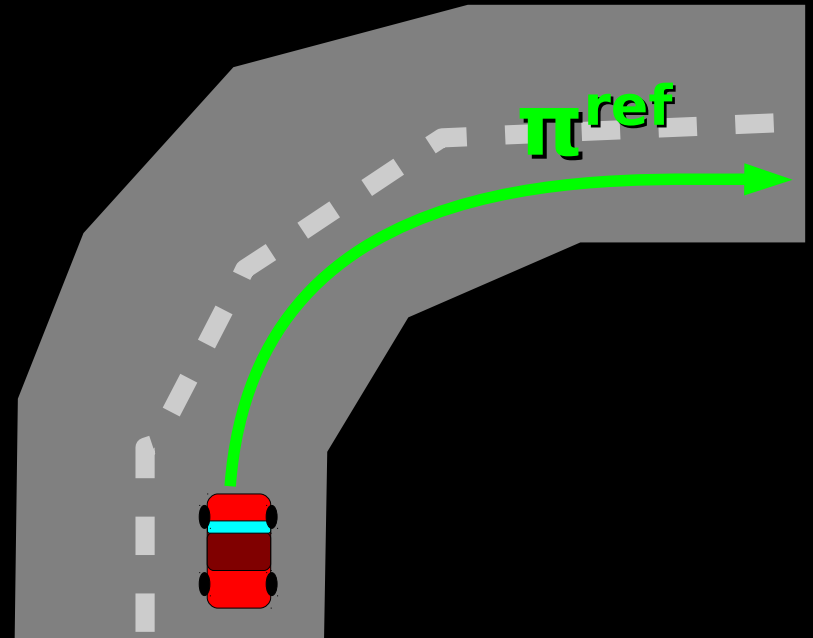
Test-time execution (sup. learning)



What's the (biggest) failure mode?

The expert never gets stuck next to pipes

⇒ Classifier doesn't learn to recover!



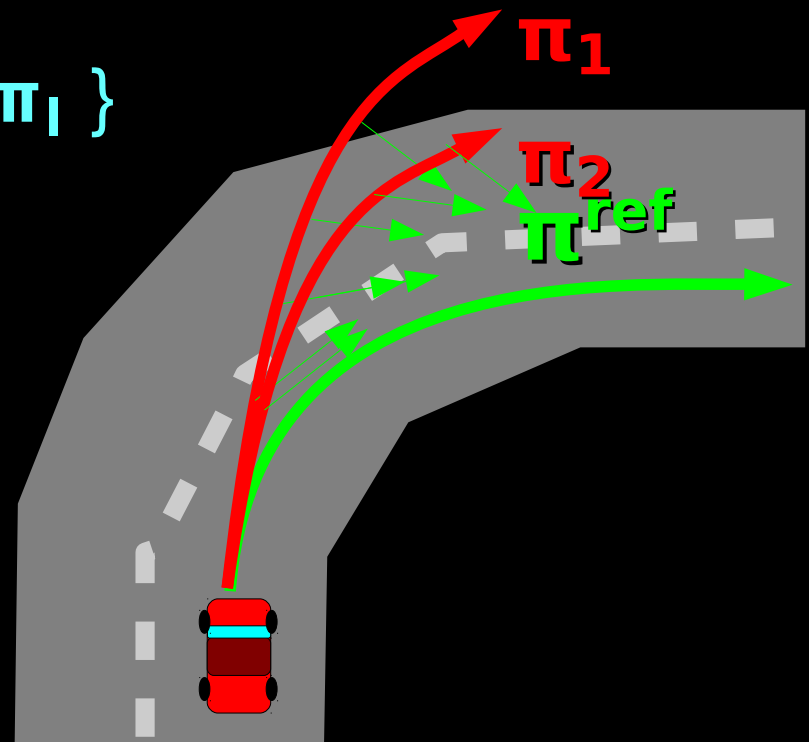
Warm-up II: Imitation learning

1. Collect trajectories from expert π^{ref}
2. Dataset $\mathbf{D}_0 = \{ (o, \pi^{\text{ref}}(o, y)) \mid o \sim \pi^{\text{ref}} \}$
3. Train π_1 on \mathbf{D}_0
4. Collect new trajectories from π_1
 - But let the *expert* steer!
5. Dataset $\mathbf{D}_1 = \{ (o, \pi^{\text{ref}}(o, y)) \mid o \sim \pi_1 \}$
6. Train π_2 on $\mathbf{D}_0 \cup \mathbf{D}_1$

• In general:

- $\mathbf{D}_n = \{ (o, \pi^{\text{ref}}(o, y)) \mid o \sim \pi_n \}$
- Train π_{n+1} on $\bigcup_{i \leq n} \mathbf{D}_i$

If $N = T \log T$,
 $\mathbf{L}(\pi_n) < T \epsilon_N + \mathbf{O}(1)$
for some n



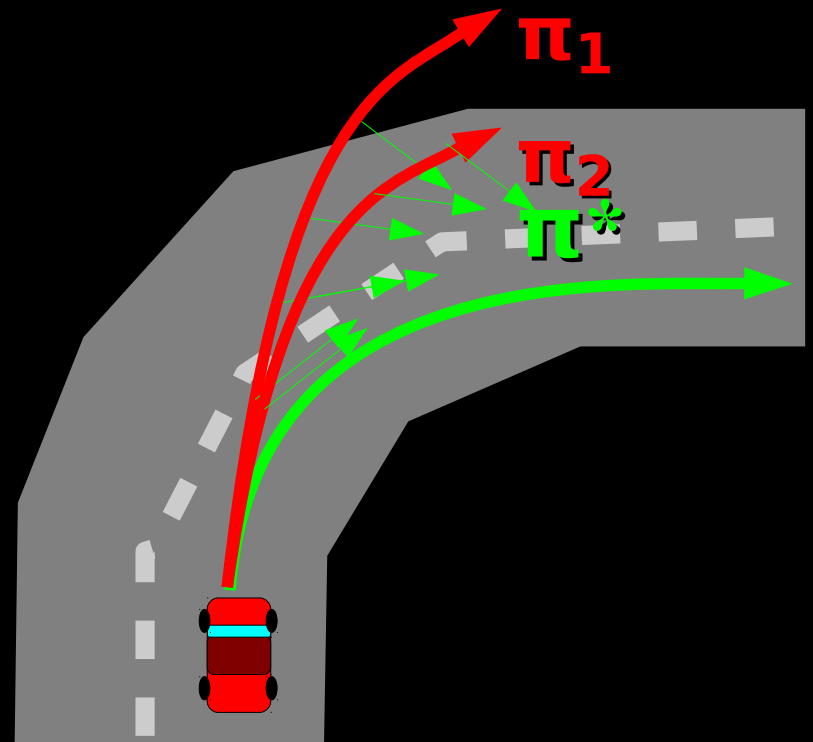
Test-time execution (Dagger)



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: $h : \mathbf{x} \rightarrow [\mathbf{K}]$

Multiclass classification

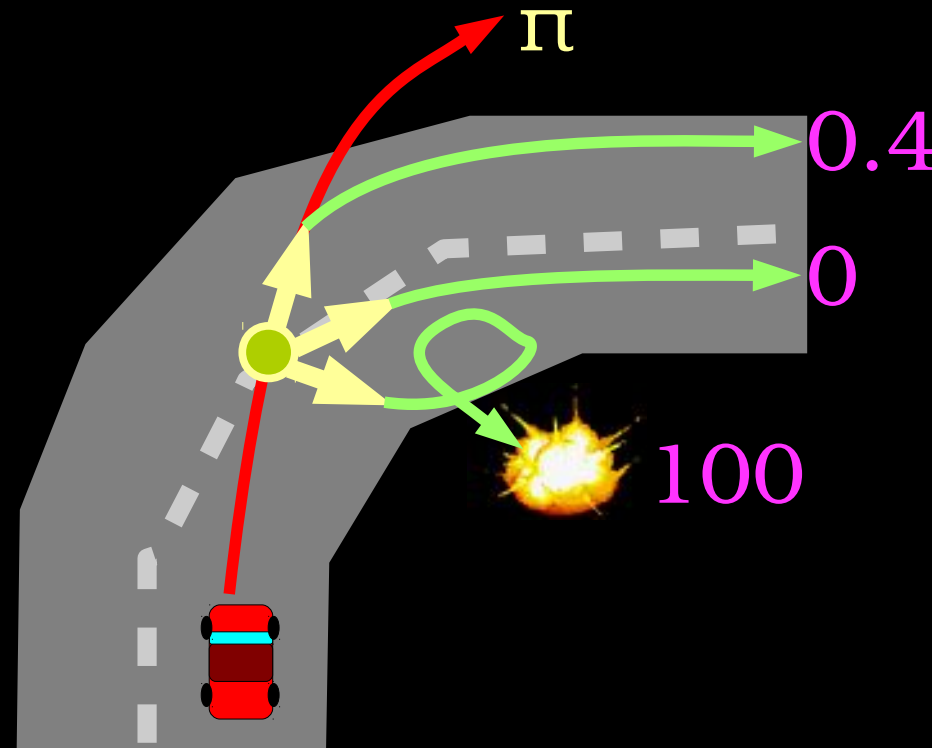
- Data: $(\mathbf{x}, y) \in X \times [\mathbf{K}]$
- Goal: $\min_h \Pr(h(\mathbf{x}) \neq y)$

Cost-sensitive classification

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

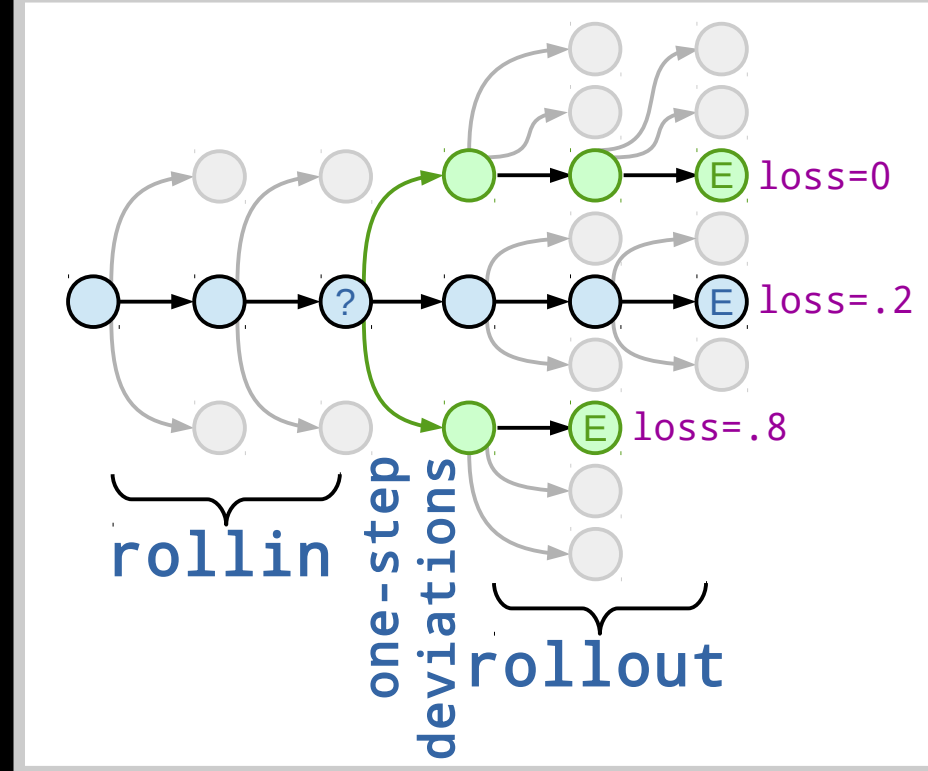
Learning to search: AggraVaTe

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



Learning to search: AggraVaTe

1. Generate **an initial trajectory** using the *current policy*



2. For each decision on that trajectory with obs. o :

a) For each possible action a (one-step deviations)

i. Take that action

ii. Complete **this trajectory** using reference policy

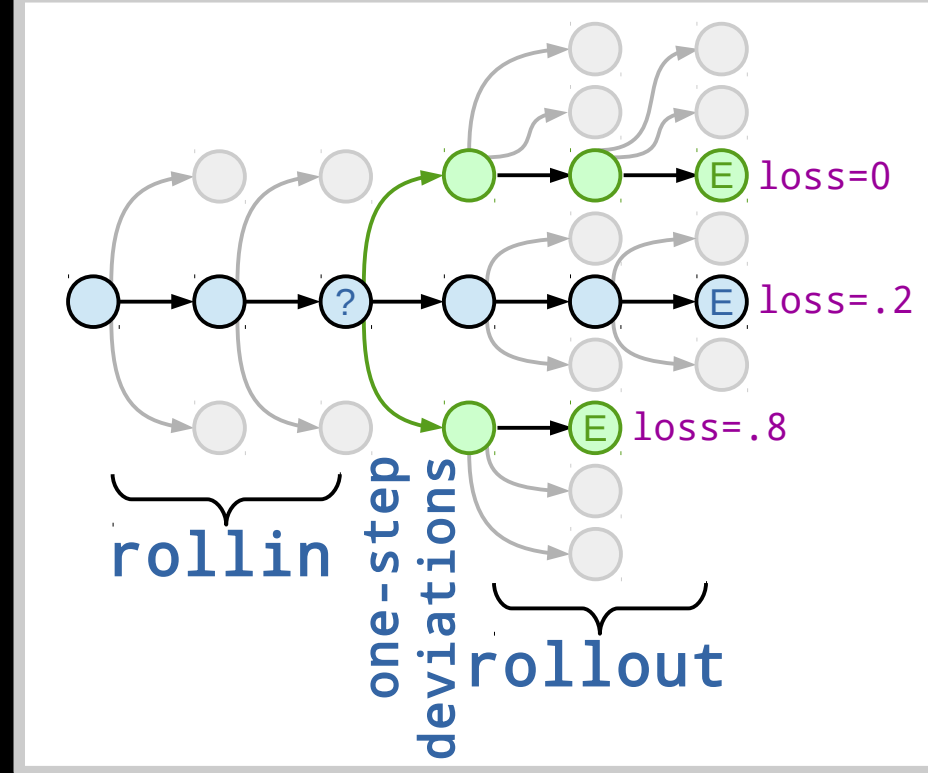
iii. Obtain a **final loss**, C_a

b) Generate a cost-sensitive classification example:

(o, \vec{c})

Learning to search: AggraVaTe

1. Generate **an initial trajectory** using the *current policy*



2. For each decision on that trajectory with obs. o :

a) For each possible action a (one-step deviations)

i. Take that action

Often it's possible to analytically

ii. Complete **this trajectory** using **reference policy** compute this loss *without*

iii. Obtain a **final loss**, C_a having to execute a roll-out!

b) Generate a cost-sensitive classification example:

(o, \vec{c})

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
 $\hat{y} =$ Dt Dt Dt Dt Dt

- Pick a timestep and try all perturbations there:

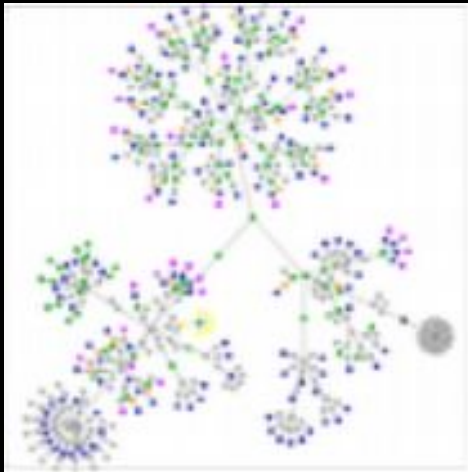
$x =$ the monster ate the sandwich
 $\hat{y}_{Dt} =$ Dt Dt
 $\hat{y}_{Nn} =$ Dt Nn
 $\hat{y}_{Vb} =$ Dt Vb

- Compute losses and construct example:

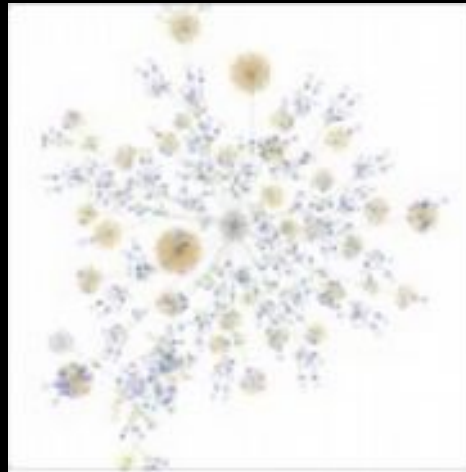
({ $w=$ monster, $p=Dt$, ... } ,
[1, 0, 1])

Example II: Graph labeling

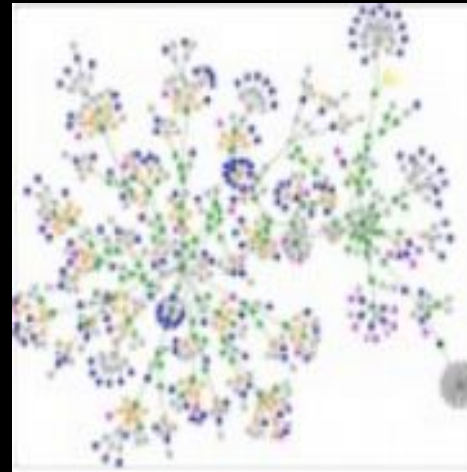
- **Task:** label nodes of a graph given node features (and possibly edge features)
- **Example:** WebKB webpage labeling



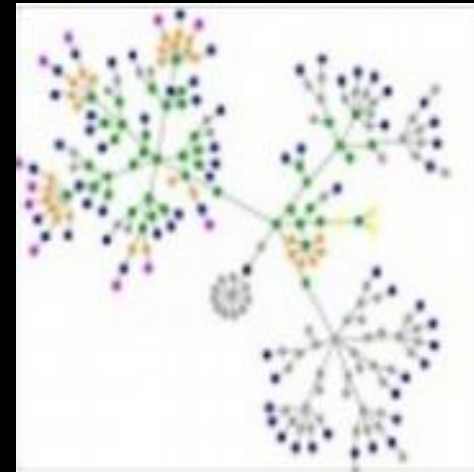
U Wisconsin



U Washington



U Texas

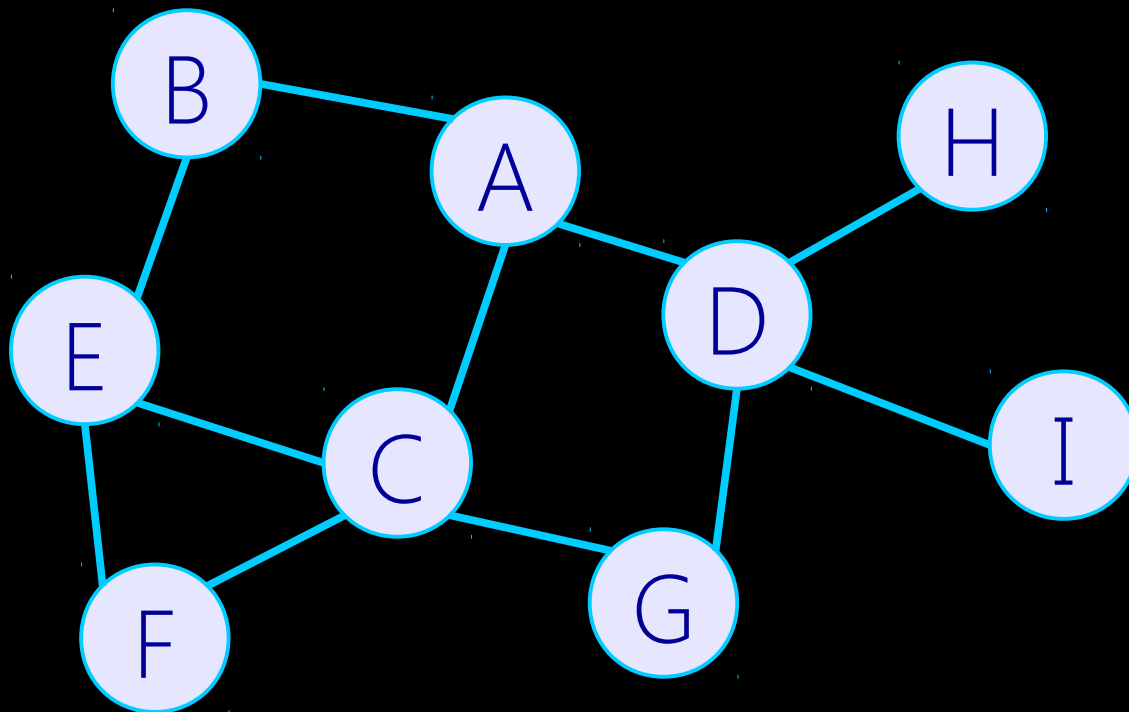


Cornell

- **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
HG FEDCBA
BCDEFGHI
HG FEDCBA
...

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

