# Stuff I did in the Spring while not Replying to Email

(aka “advances in structured prediction”)

Hal Daumé III | University of Maryland | me@hal3.name | @haldaume3

# 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
```

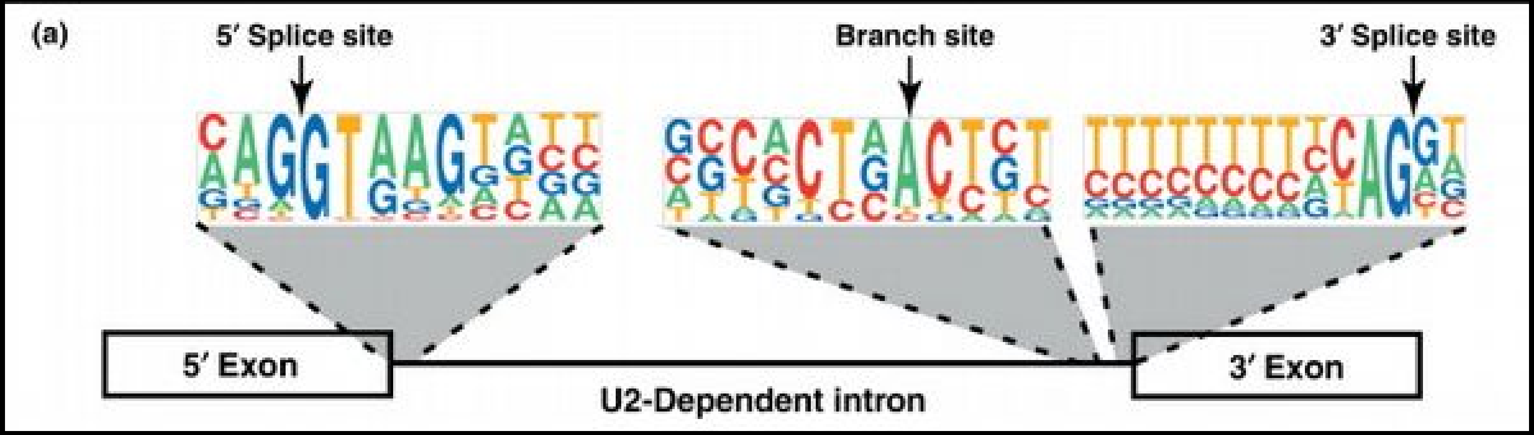
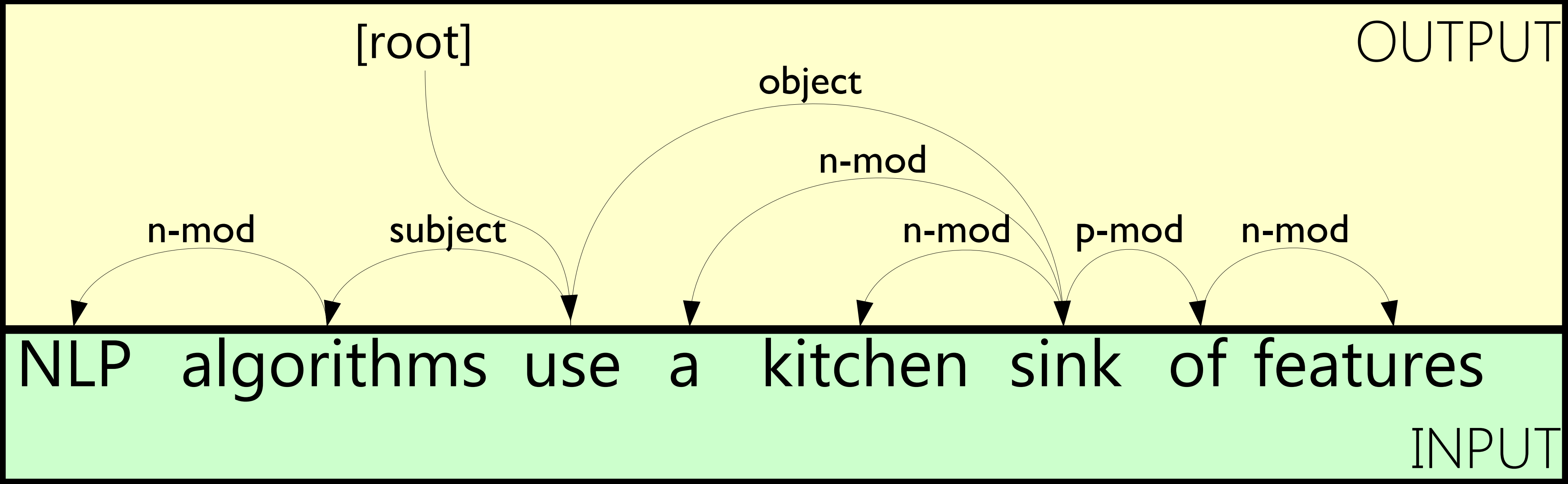
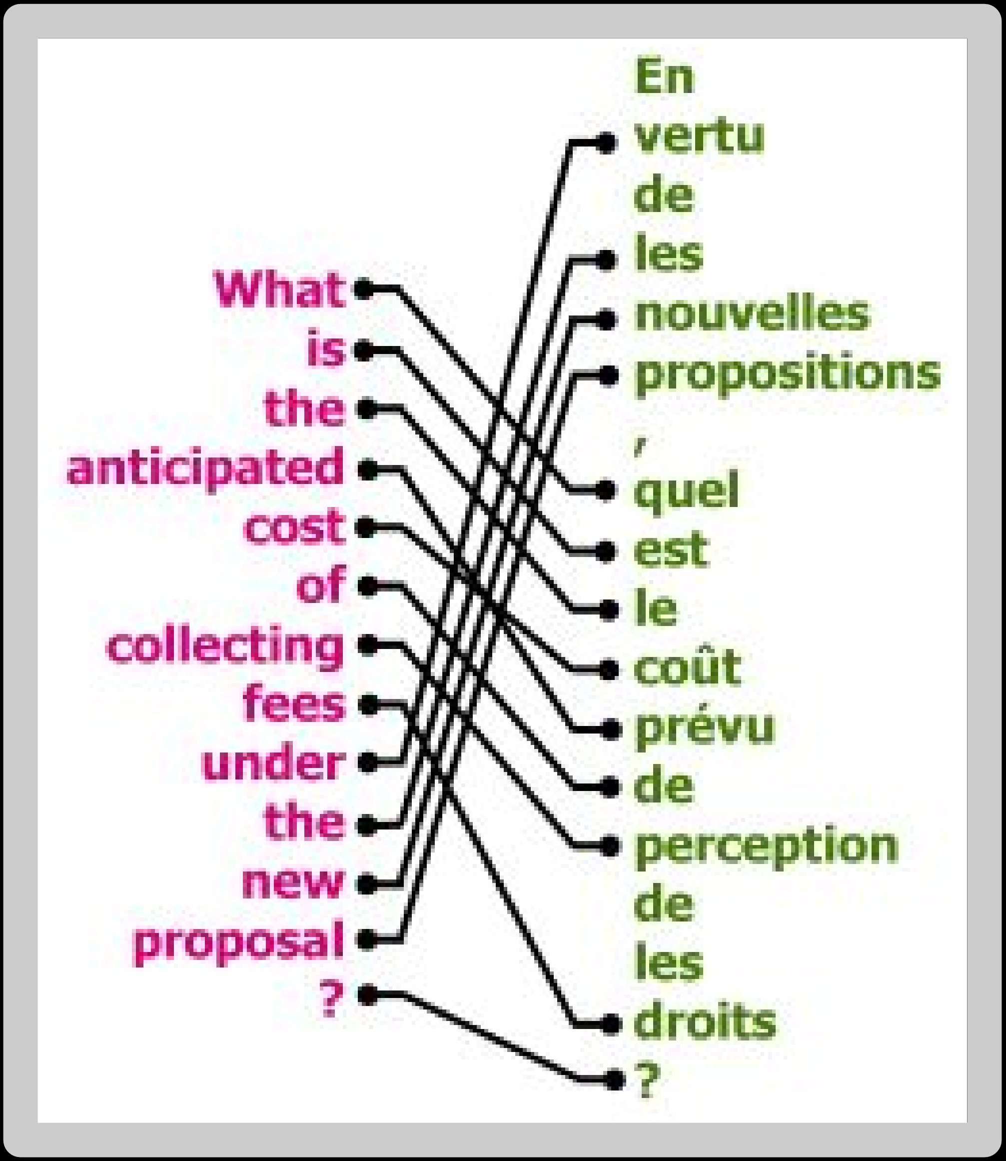



image credit: Richard Padgett

# Natural language parsing



# (Bipartite) matching



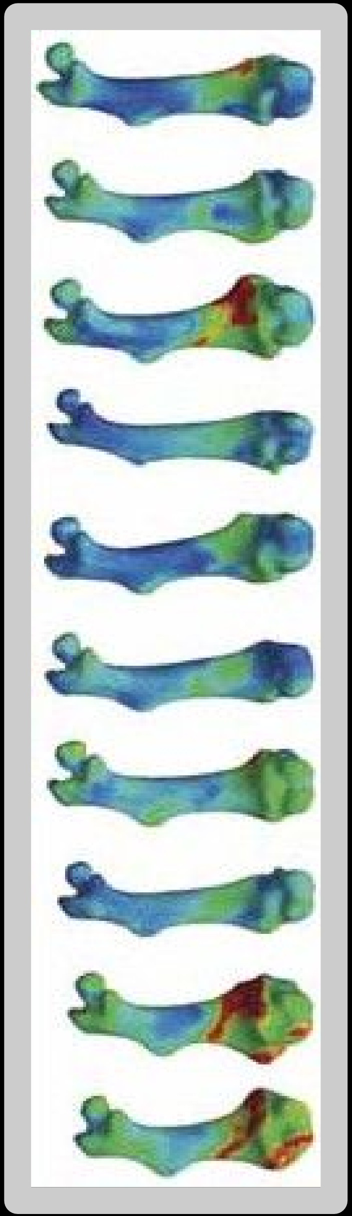



image credit: Ben Taskar; Liz Jurrus

# Machine translation

# Segmentation

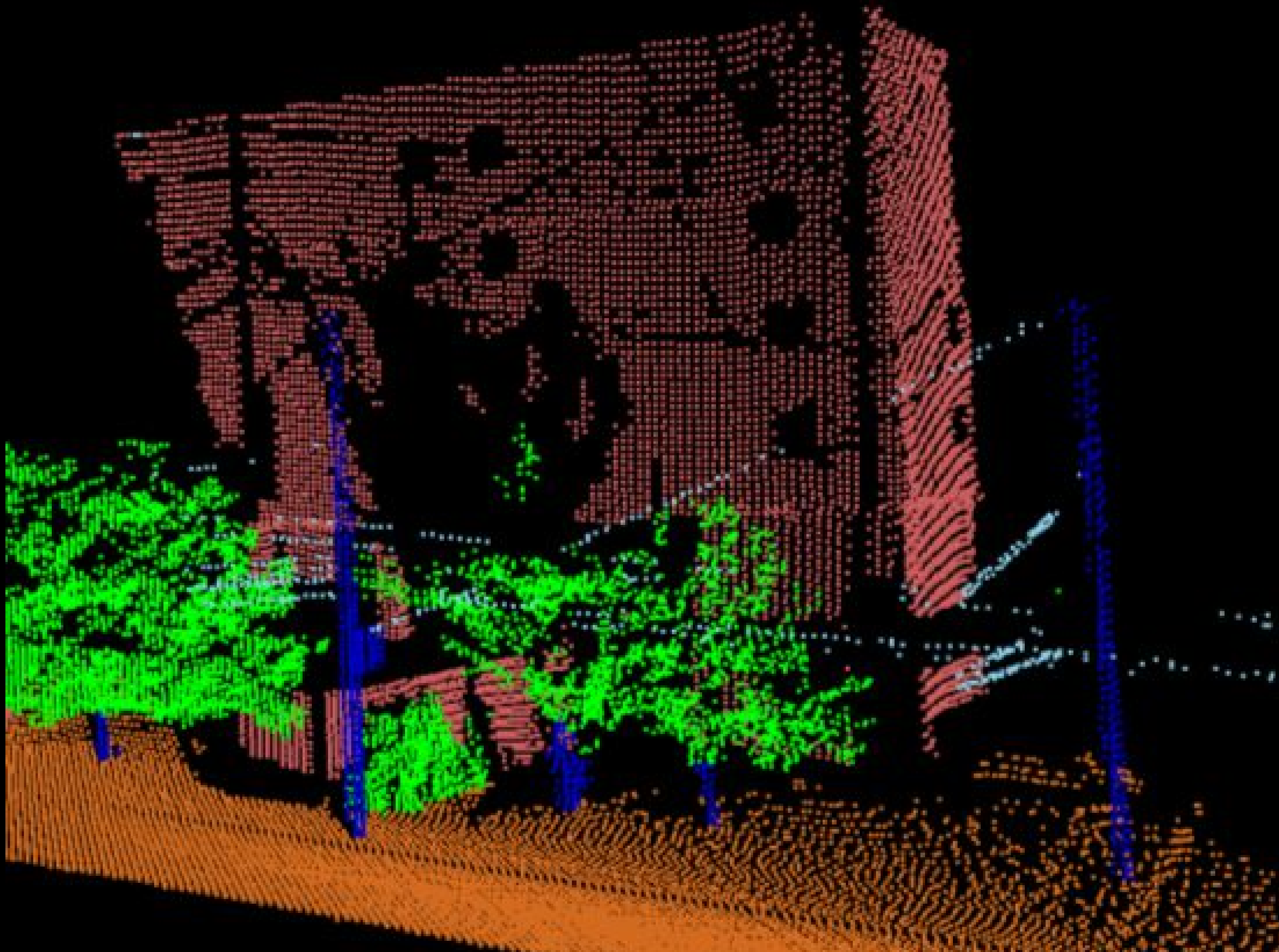
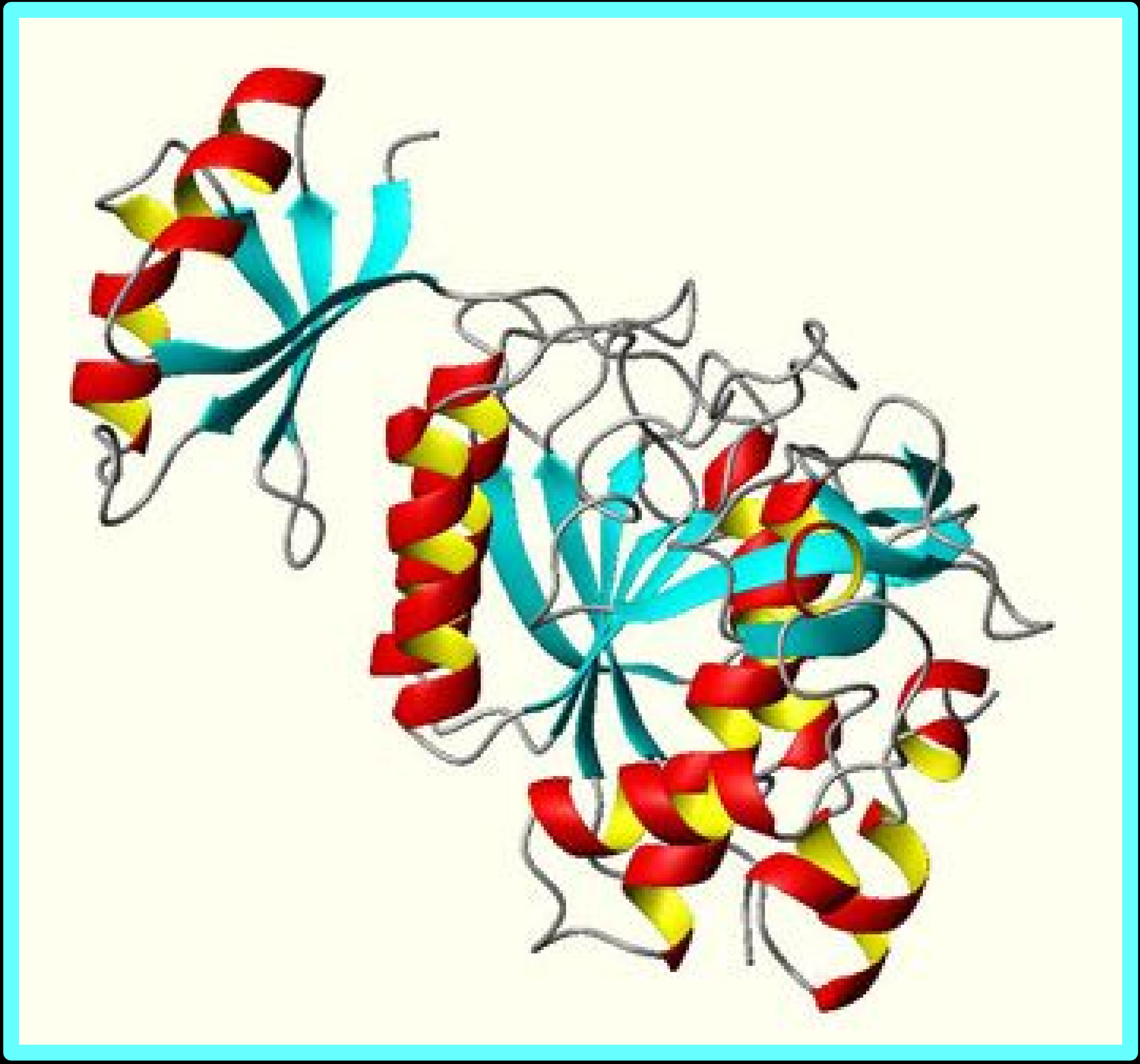


image credit: Daniel Muñoz

# Protein secondary structure prediction



# Outline

Isn't this kinda narrow?

- Background: learning to s
- Stuff I did in the Spring
  - Imperative DSL/library for learning to search
  - SOTA examples for tagging, parsing, relation extraction, etc.
  - Learning to search under bandit feedback
  - Hardness results for learning to search
  - Active learning for accelerating learning to search
- Stuff I'm trying to do now
  - Distant supervision
  - Mashups with recurrent neural networks

# My experience, 6 months in industry

- Standard adage: academia=freedom, industry=time
  - Number of responsibilities vs number of bosses
- Aspects I didn't anticipate
  - Breadth (academia) versus depth (industry)
  - Collaborating through students versus directly
  - Security through tenure versus security through $
- At the end of the day: who are your colleagues and what do you have to do to pay the piper?

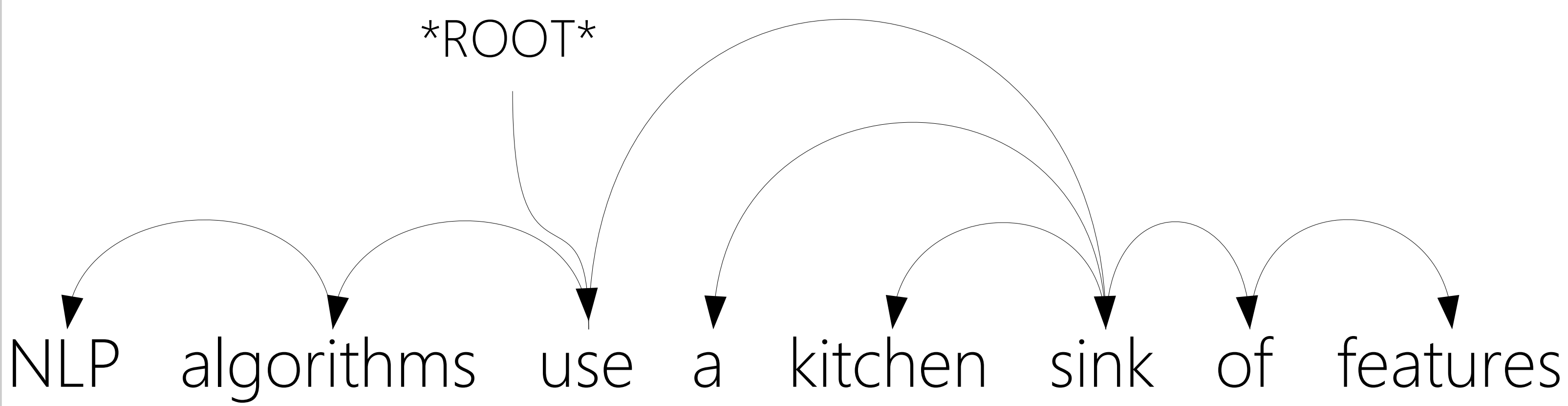
Major caveat: this is comparing a top ranked CS dept to top industry lab, in a time when there's tons of money in this area (more in industry)

# Joint prediction via learning to search

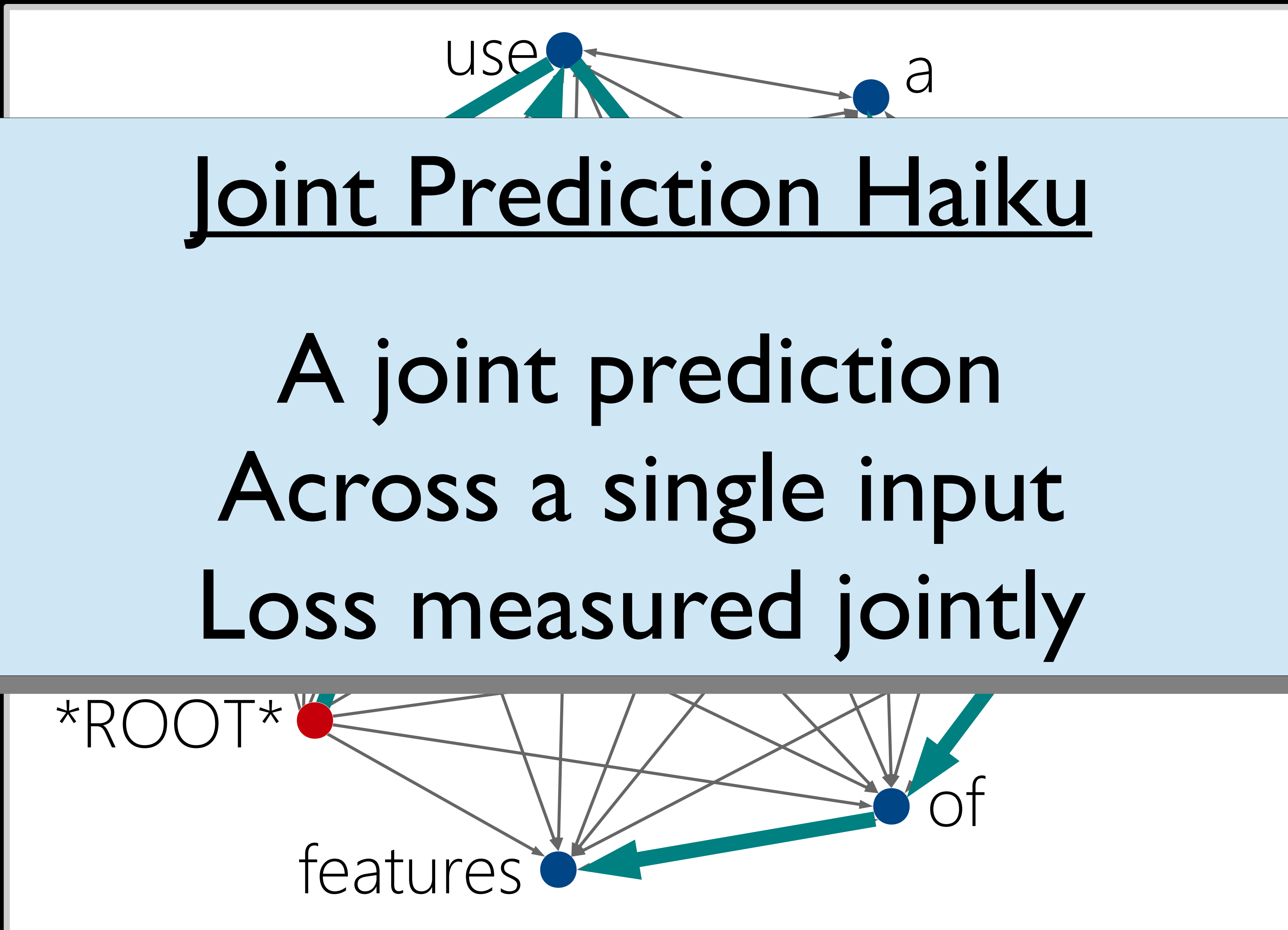
## Part of Speech Tagging

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| NN | NNS | VBP | DT | NN | NN | IN | NNS |
| NLP | algorithms | use | a | kitchen | sink | of | features |

## Dependency Parsing



# Joint prediction via learning to search




## Joint Prediction Haiku

A joint prediction
Across a single input
Loss measured jointly

# 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

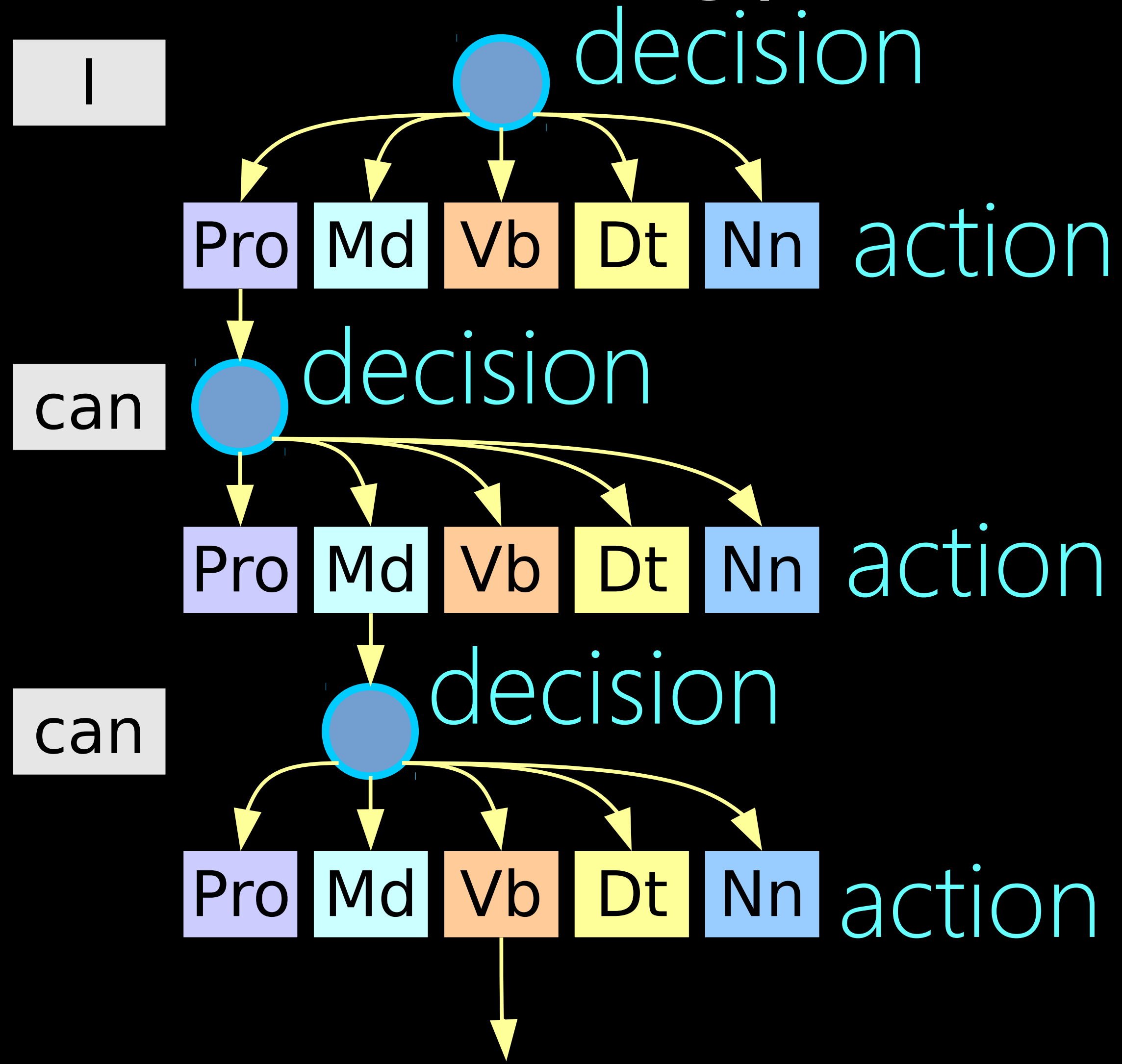
$$E_{(x,y)\sim D} [ \text{loss}(y, h(x)) ]$$

based on N samples

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

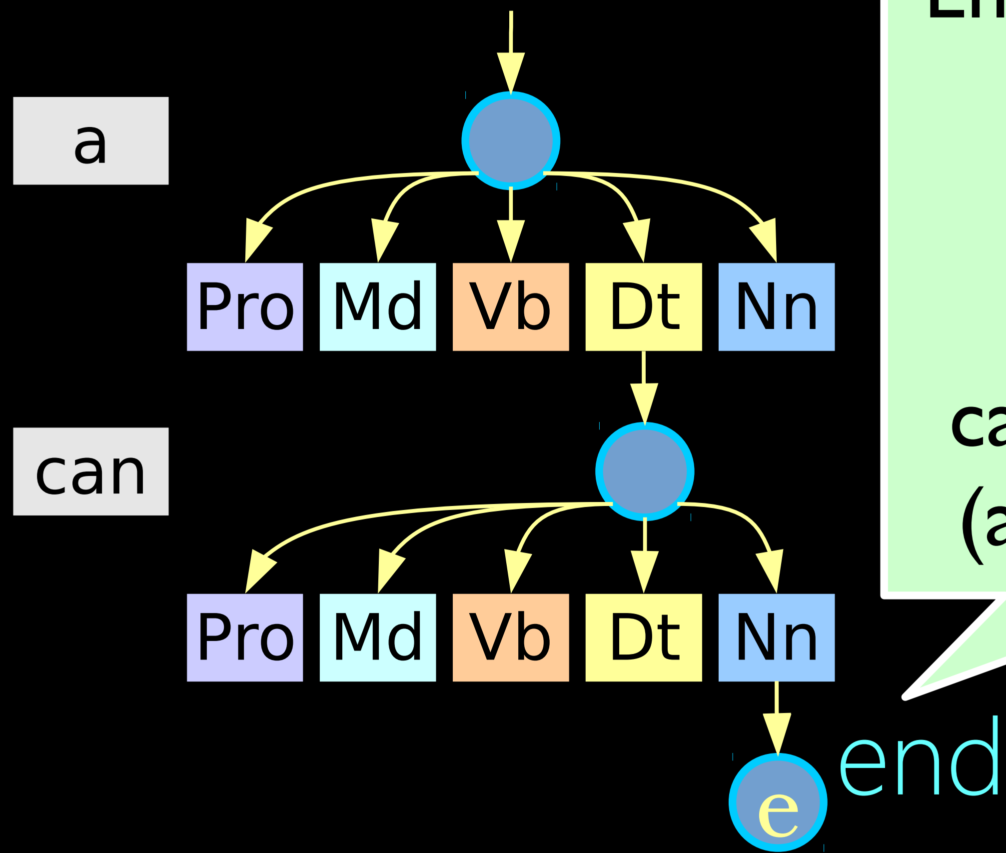
# Search spaces

- When $\mathbf{y}$ decomposes in an ordered manner, a sequential decision making process emerges



# Search spaces

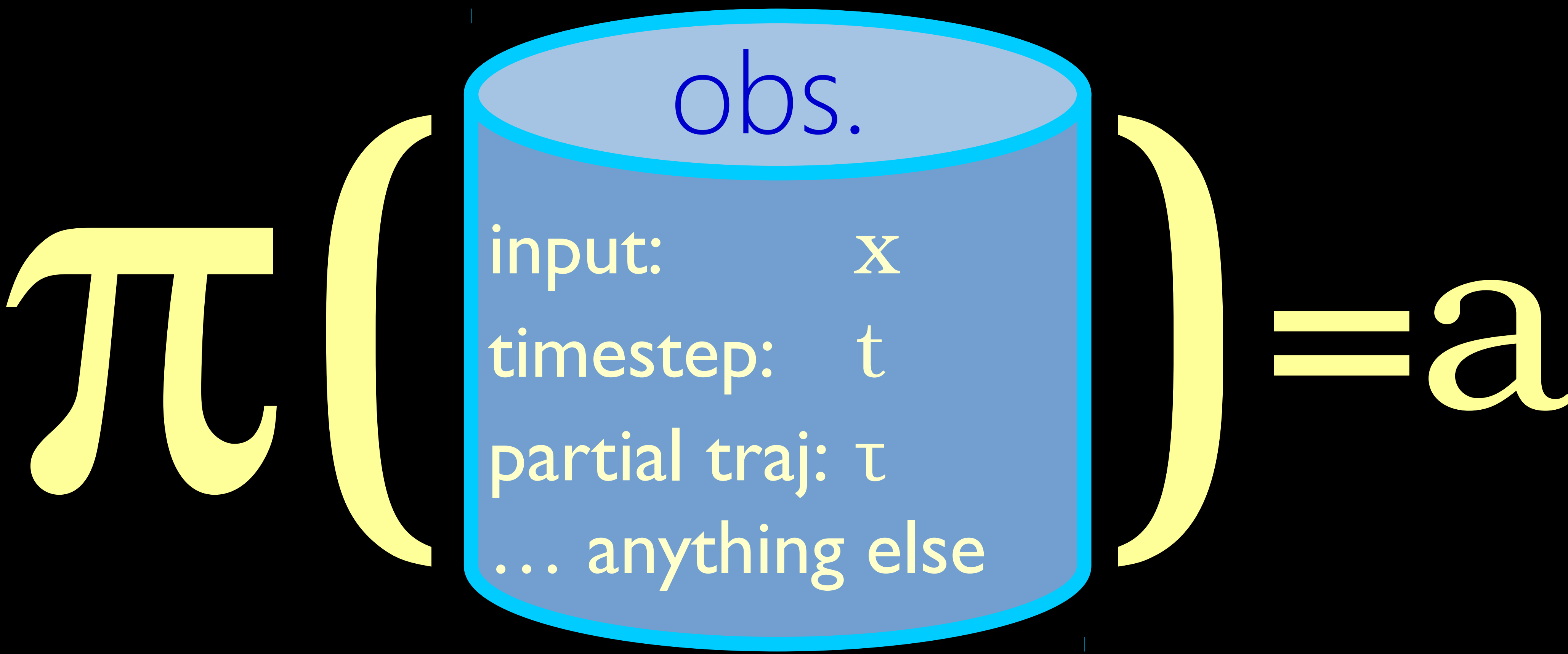
- When $y$ decomposes in an ordered manner, a sequential decision making process emerges




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

# Policies

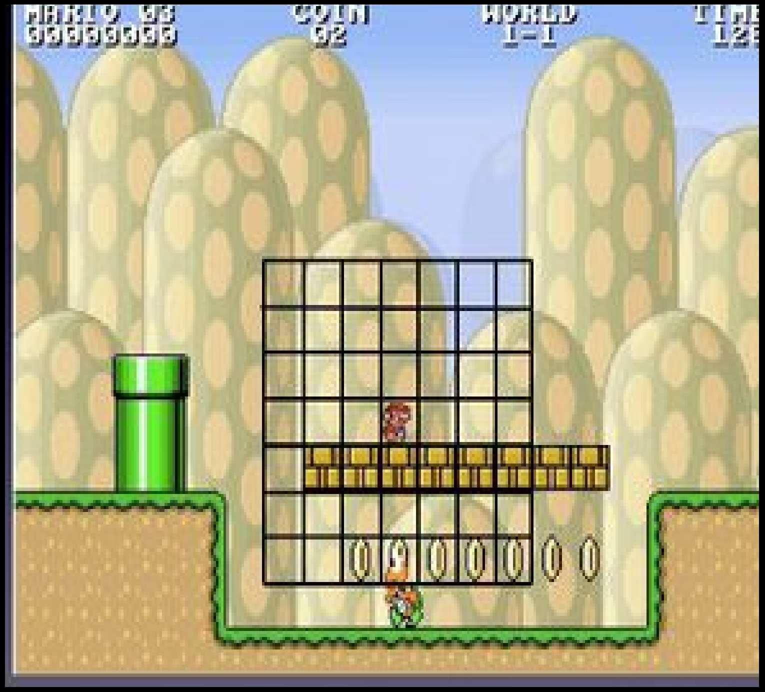
- A policy maps observations to actions



# An analogy from playing Mario

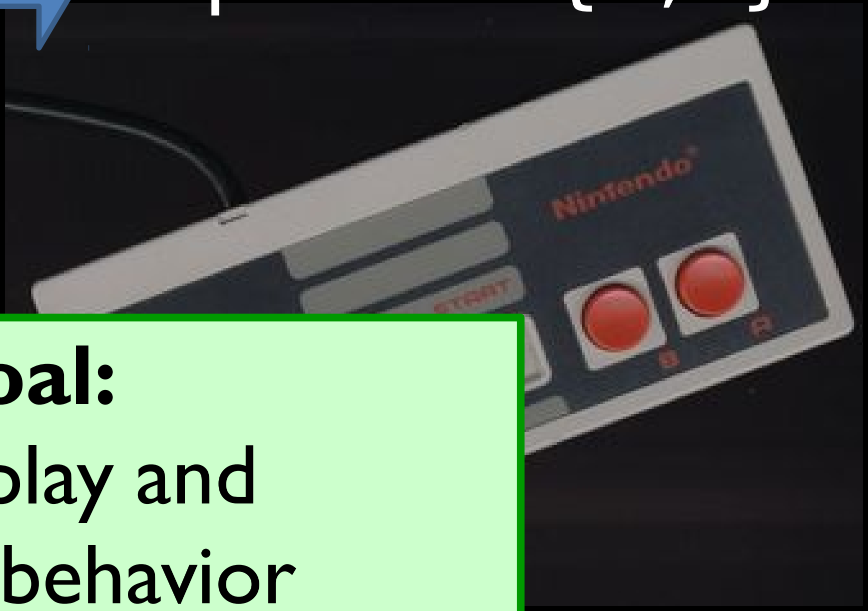
From Mario AI competition 2009

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

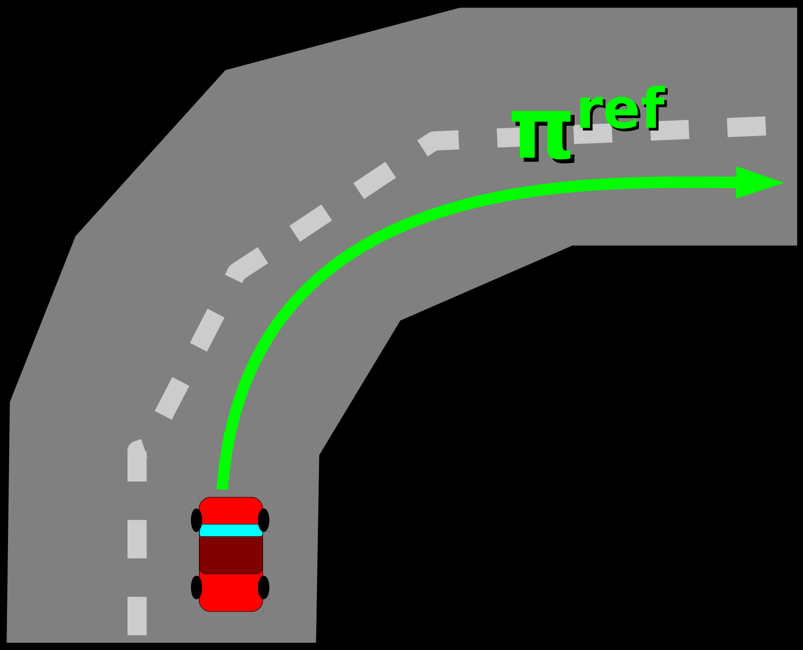
# Training (expert)


Video credit: Stéphane Ross, Geoff Gordon and Drew Bagnell

# Warm-up: Supervised learning

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

- **Let π play the game!**



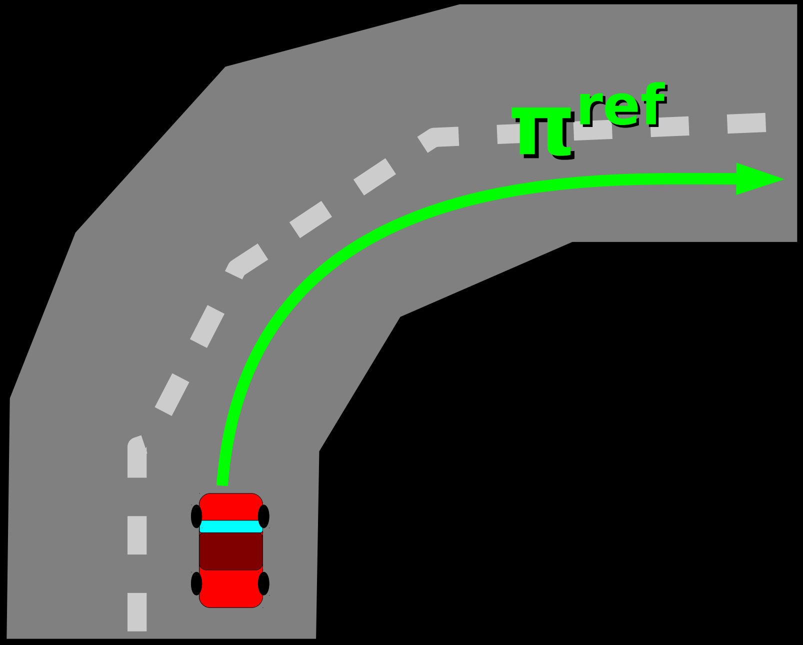
# Test-time execution (sup. learning)


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

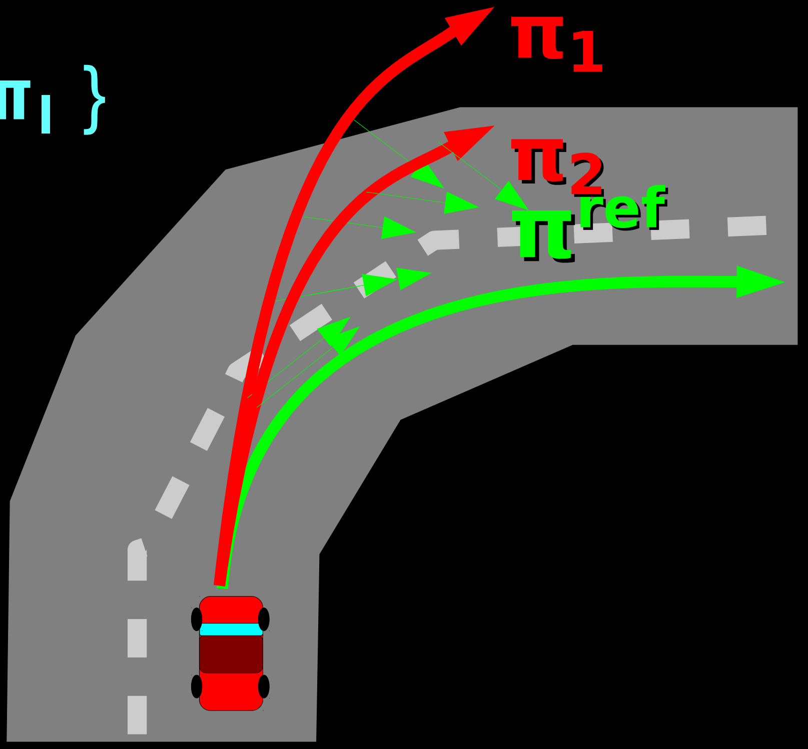
1. Collect trajectories from expert $\pi^{ref}$
2. Dataset $D_0 = \{ ( o, \pi^{ref}(o,y) ) \mid o \sim \pi^{ref} \}$
3. Train $\pi_1$ on $D_0$
4. Collect new trajectories from $\pi_1$
   - But let the *expert* steer!
5. Dataset $D_1 = \{ ( o, \pi^{ref}(o,y) ) \mid o \sim \pi_1 \}$
6. Train $\pi_2$ on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

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

If $N = T \log T$,

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

for some n



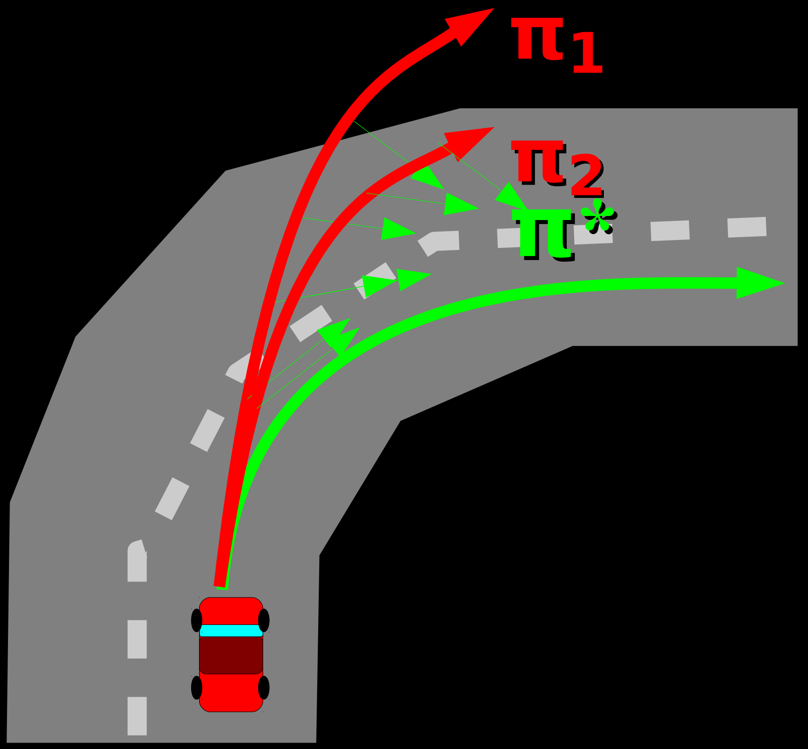
# Test-time execution (DAgger)


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

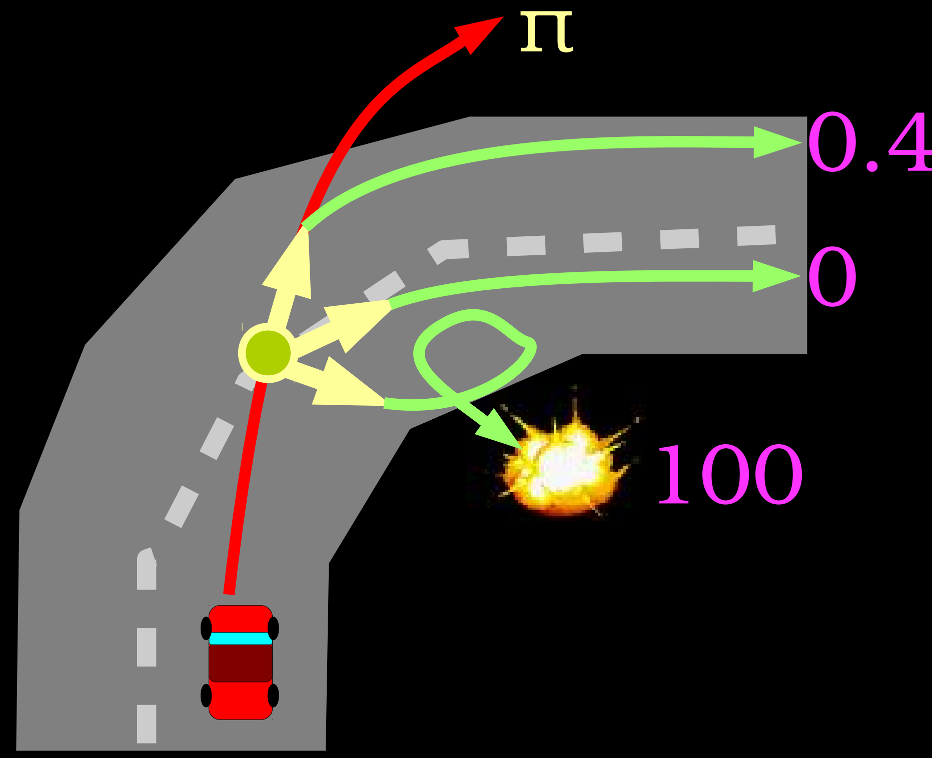


# Learning to search: AggraVaTe

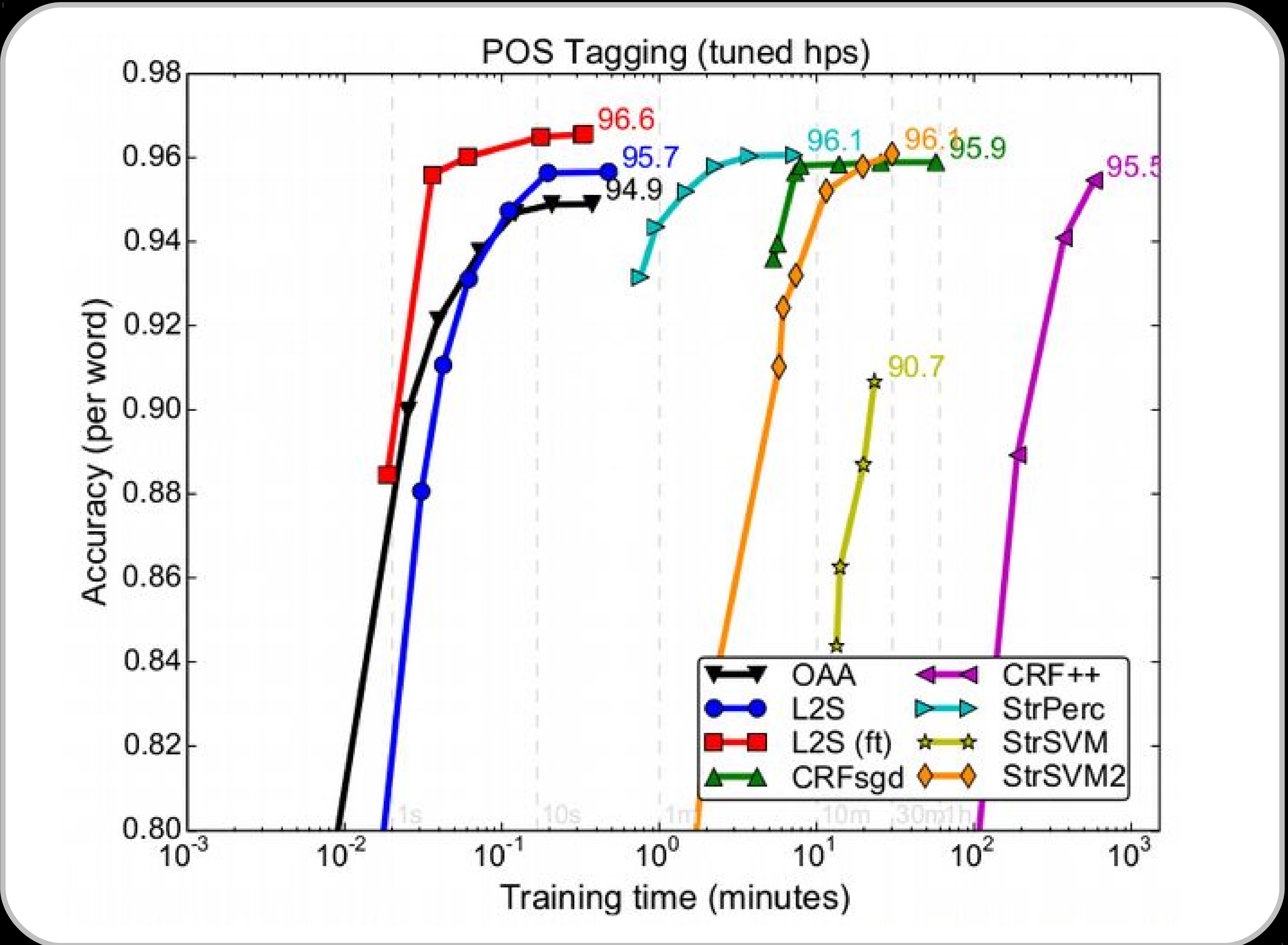
1. Let learned policy $\pi$ drive for $t$ timesteps to obs. $o$
2. For each possible action $a$:
   - Take action $a$, and let expert $\pi^{\text{ref}}$ drive the rest
   - Record the overall loss, $c_a$
3. Update $\pi$ based on example:

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

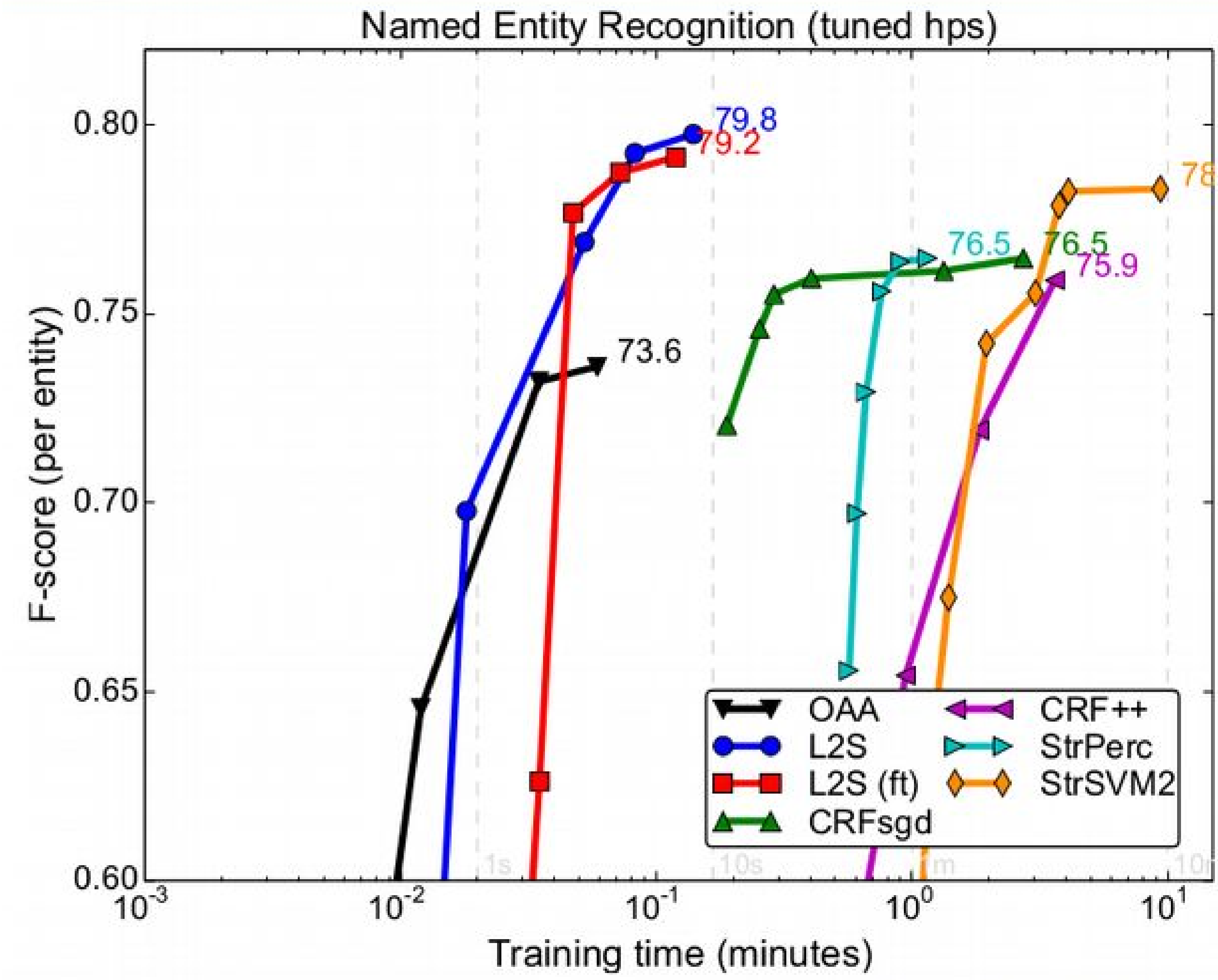
4. Goto (1)



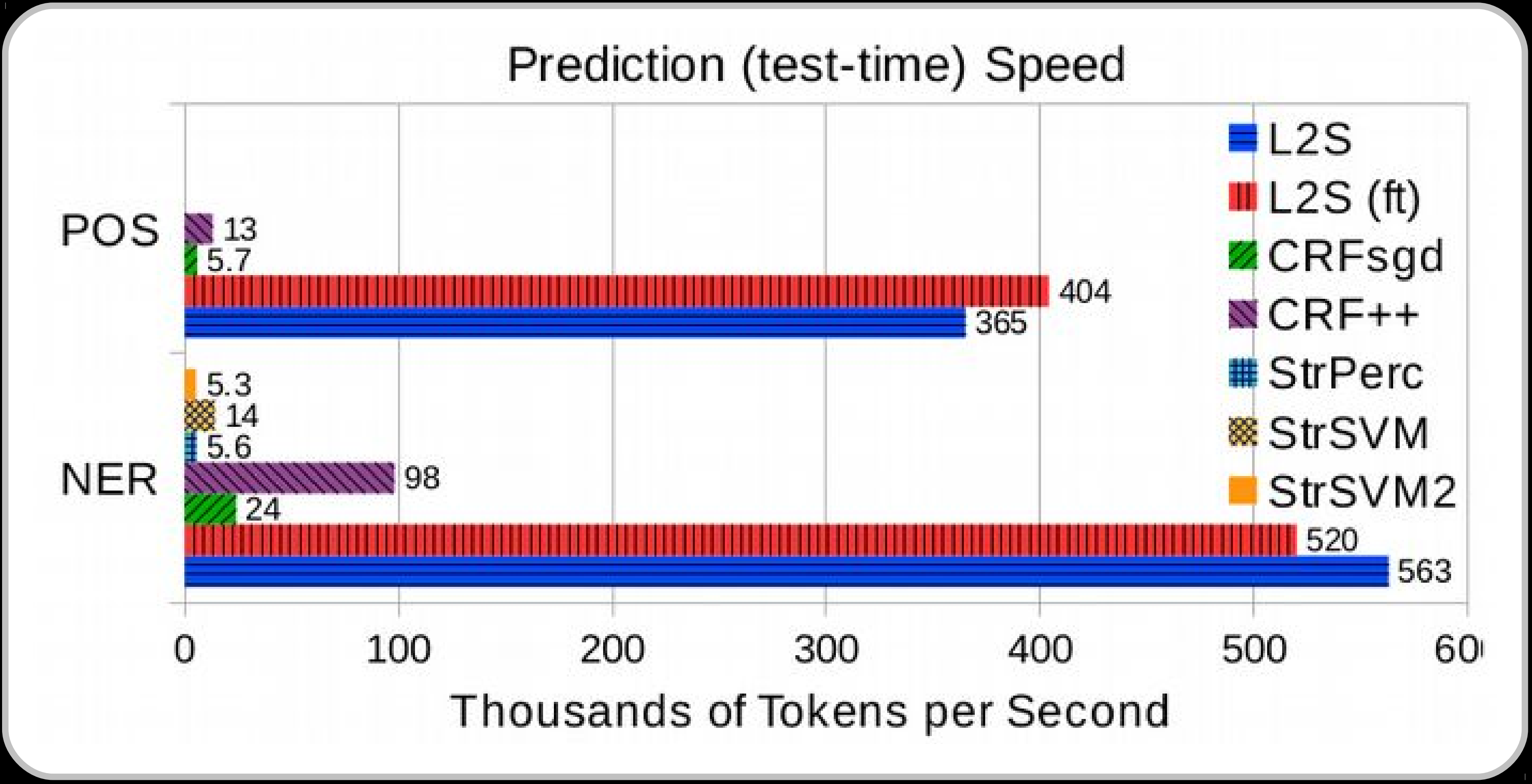
# Training time versus test accuracy



# Training time versus test accuracy



# Test time speed



# State of the art accuracy in....

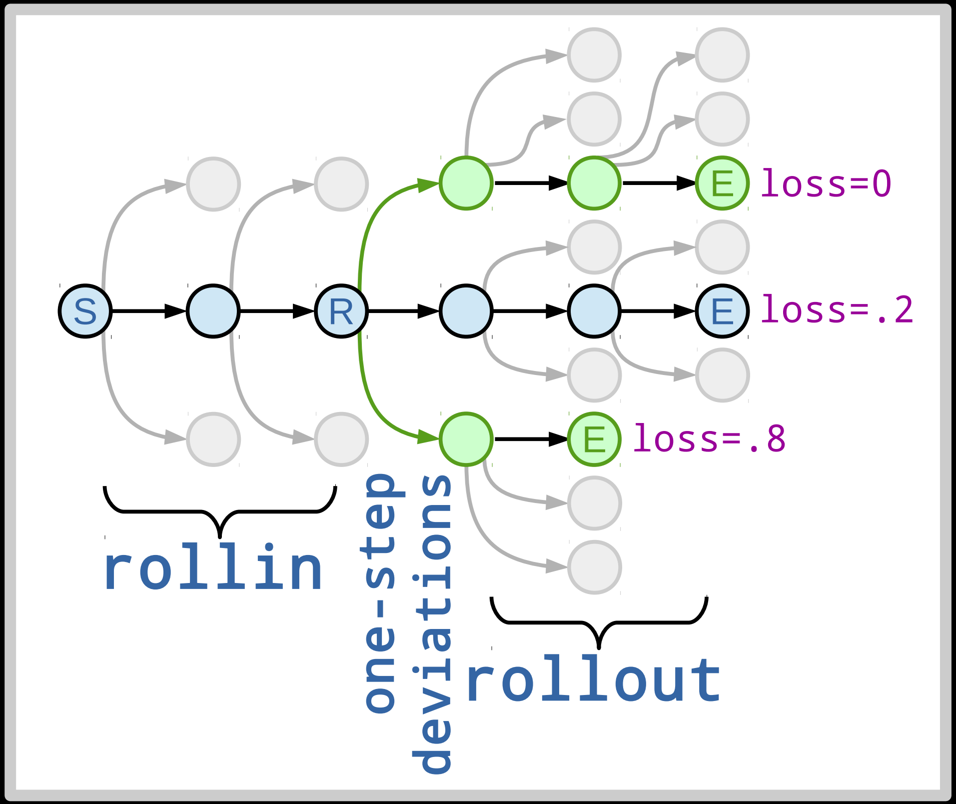
- Part of speech tagging (1 million words)
  - US: 6 lines of code 10 seconds to train
  - CRFsgd: 1068 lines 30 minutes
  - CRF++: 777 lines hours
- Named entity recognition (200 thousand words)
  - US: 30 lines of code 5 seconds to train
  - CRFsgd: 1 minute
  - CRF++: 10 minutes
  - SVMstr: 876 lines 30 minutes (suboptimal accuracy)

# The Magic

- You write some greedy “test-time” code
  - In your favorite imperative language (C++/Python)
  - It makes arbitrary calls to a `Predict` function
  - And you add some minor decoration

- We will automatically:
  - Perform learning
  - Generate non-determinstic (beam) search
  - Run faster than specialized learning software

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

# The magic in practice

```
run(vector<example> ec)
  for i = 0 .. ec.size
    y_true = get_example_label(ec[i])
    y_pred = Predict(ec[i], y_true)
  Loss( # of y_true != y_pred )
```

A "hint" about the correct decision only at training time

not hiding anything...

```
void run(search& sch, vector<example*> ec) {
  for (size_t i=0; i<ec.size(); i++) {
    uint32_t y_true = get_example_label(ec[i]);
    uint32_t y_pred = sch.predict(ec[i], y_true);

    sch.loss( y_true != y_pred );

    if (sch.output().good())
      sch.output() << y_pred << ' ';
  }
}
```

# The illusion of control

- Execute `run` O(T×A) times, modifying `Predict`
- For each time step myT = 1 .. T:

  For each possible action myA = 1 .. A:

  $$\text{define } \texttt{Predict}(\ldots) = \begin{cases} \text{myA} & \text{if } t = \text{myT} \\ \pi & \text{otherwise} \end{cases}$$

  run your code in full

  set $\text{cost}_a$ = result of `Loss`

  Make classification example on $x_{\text{myT}}$ with $\langle \text{cost}_a \rangle$

```
run(vector<example> ec)
  for i = 0 .. ec.size
    y_true = get_example_label(ec[i])
    y_pred = Predict(ec[i], y_true)
  Loss( # of y_true != y_pred )
```

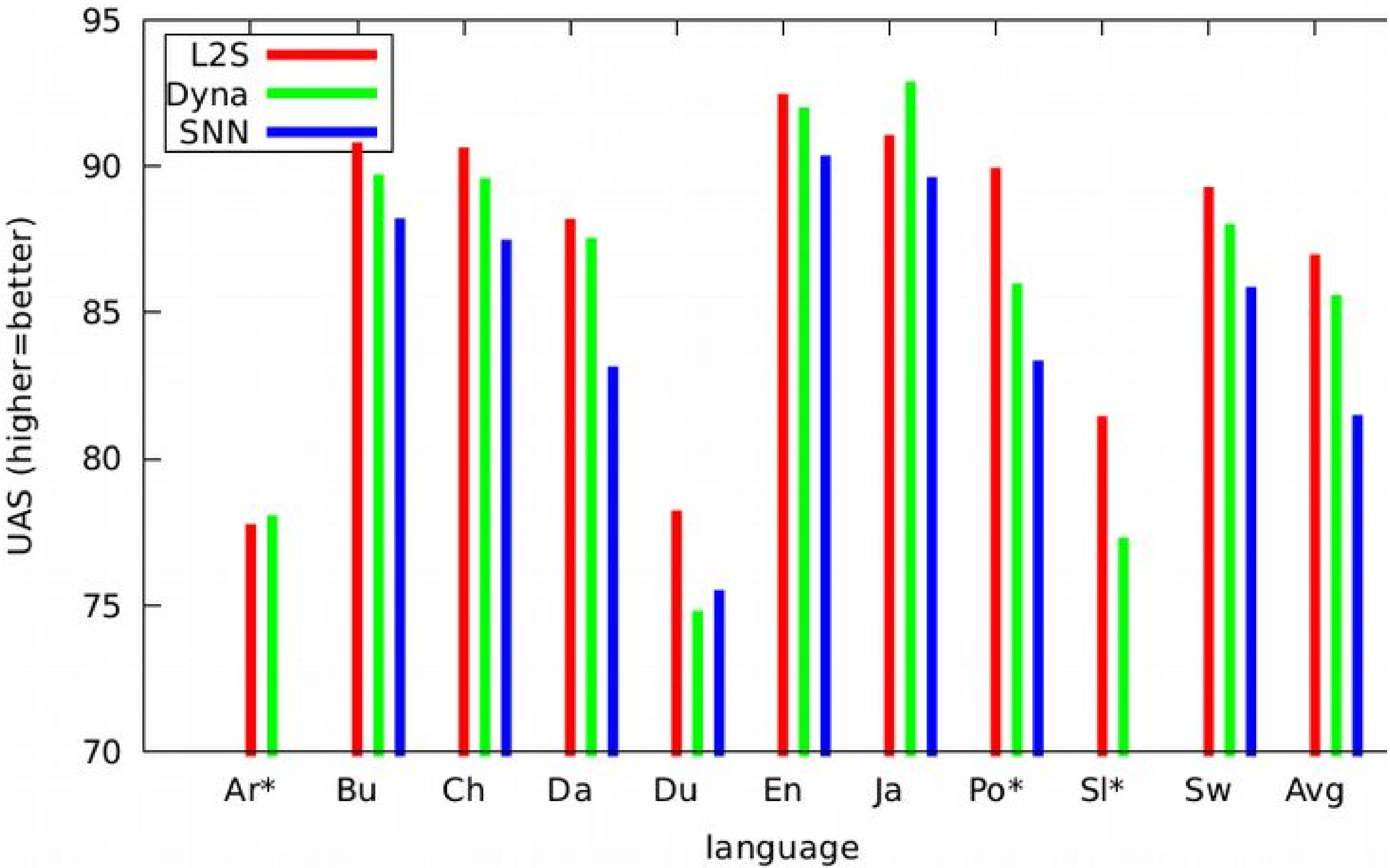
# Entity/relation identification

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.

# Dependency parsing

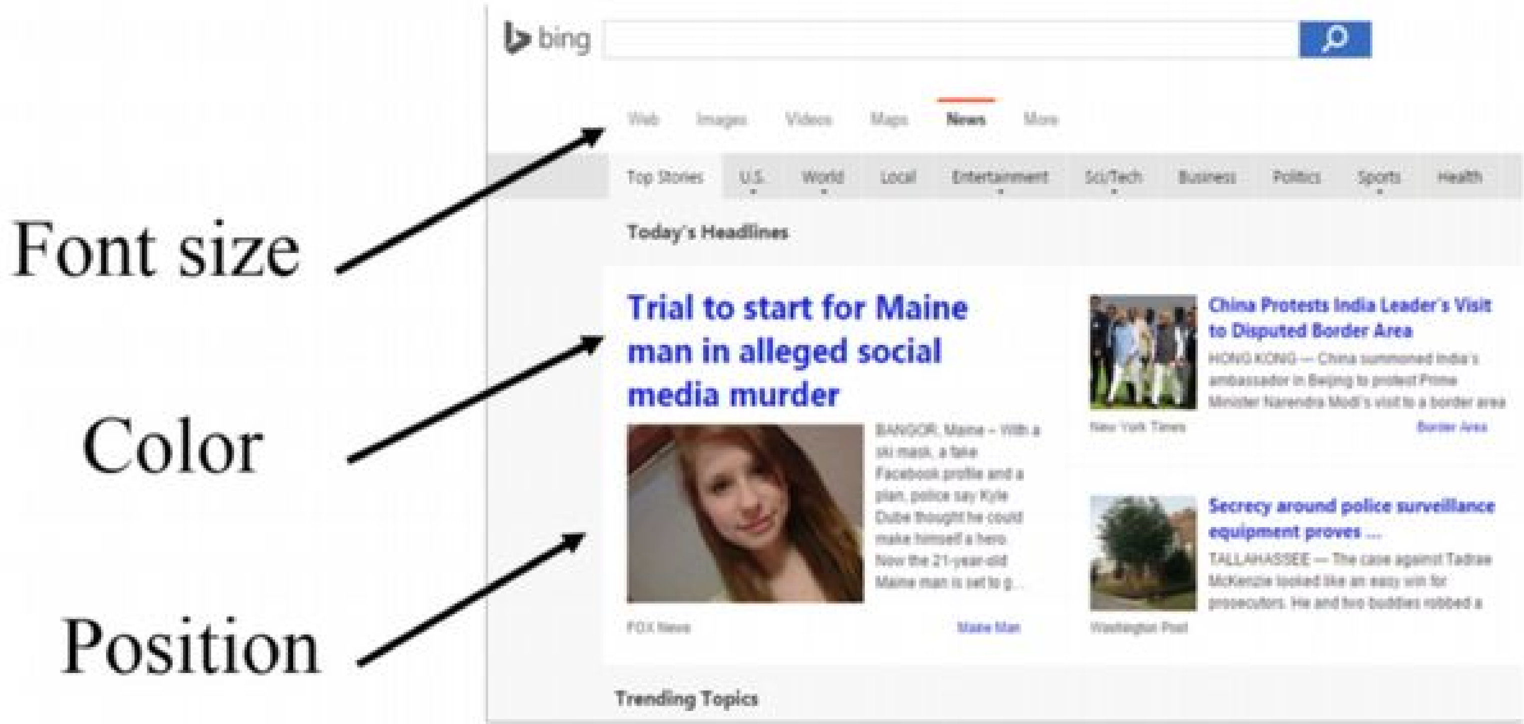



L2S uses ~300 LOC.

# Outline

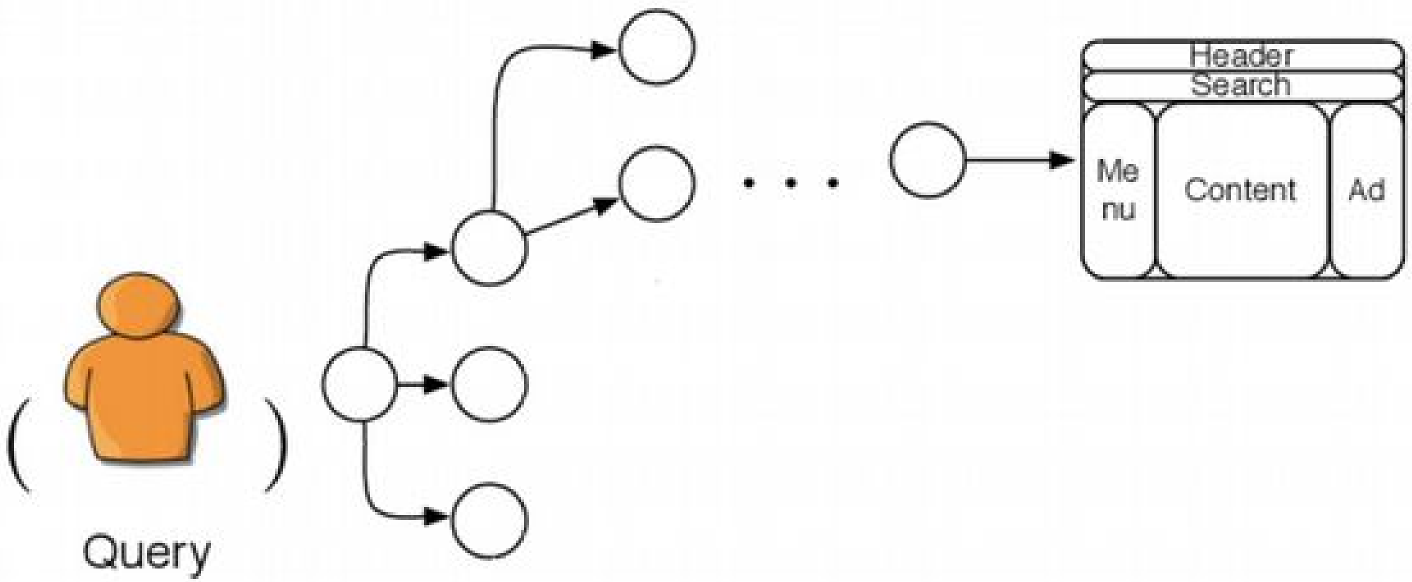
- Background: learning to search
- Stuff I did in the Spring
  - Imperative DSL/library for learning to search
  - SOTA examples for tagging, parsing, relation extraction, etc.
  - Learning to search under bandit feedback
  - Hardness results for learning to search
  - Active learning for accelerating learning to search
- Stuff I'm trying to do now
  - Distant supervision
  - Mashups with recurrent neural networks

# Structured Contextual Bandit

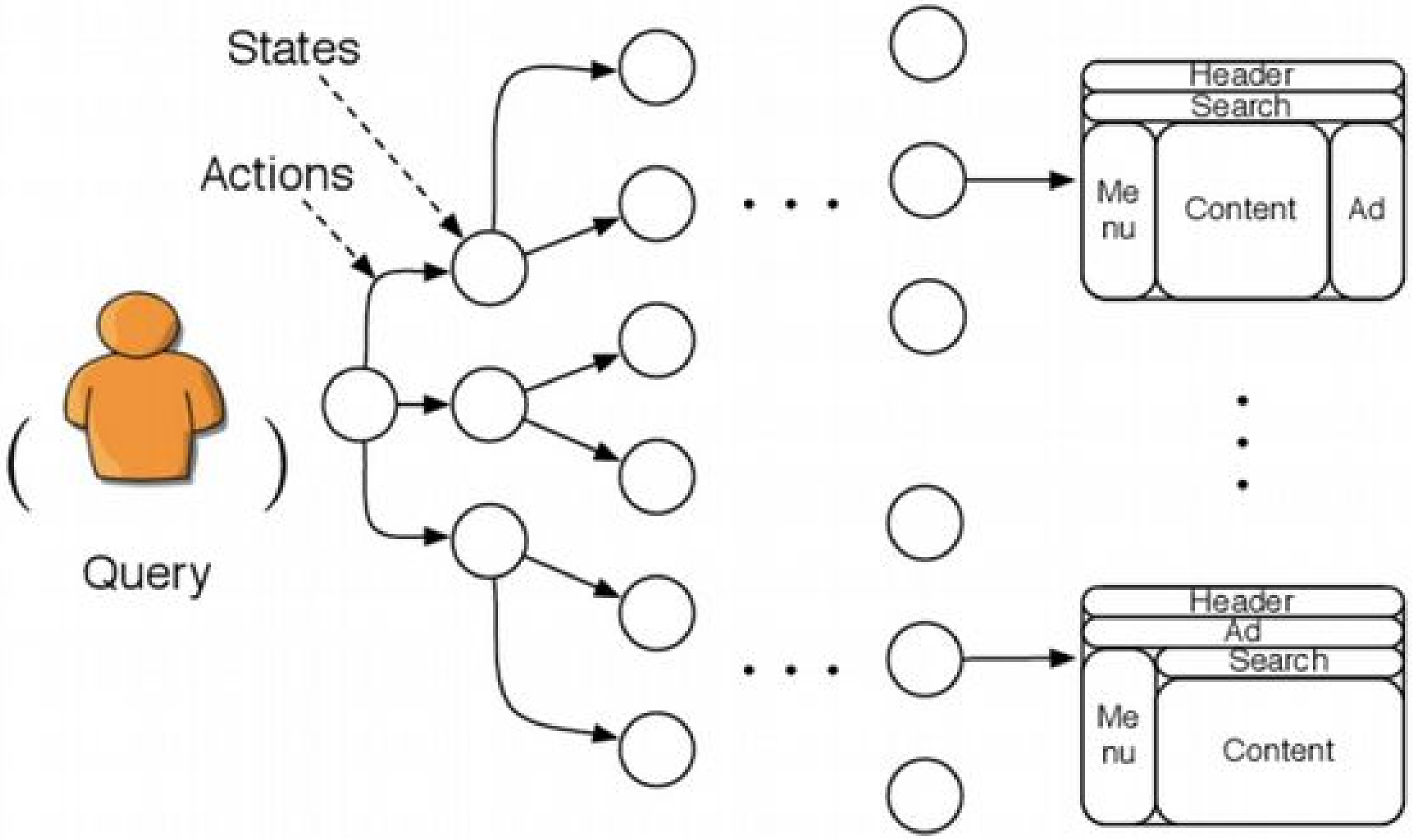



- Loss of a single structured label can be observed.

# A Search Problem

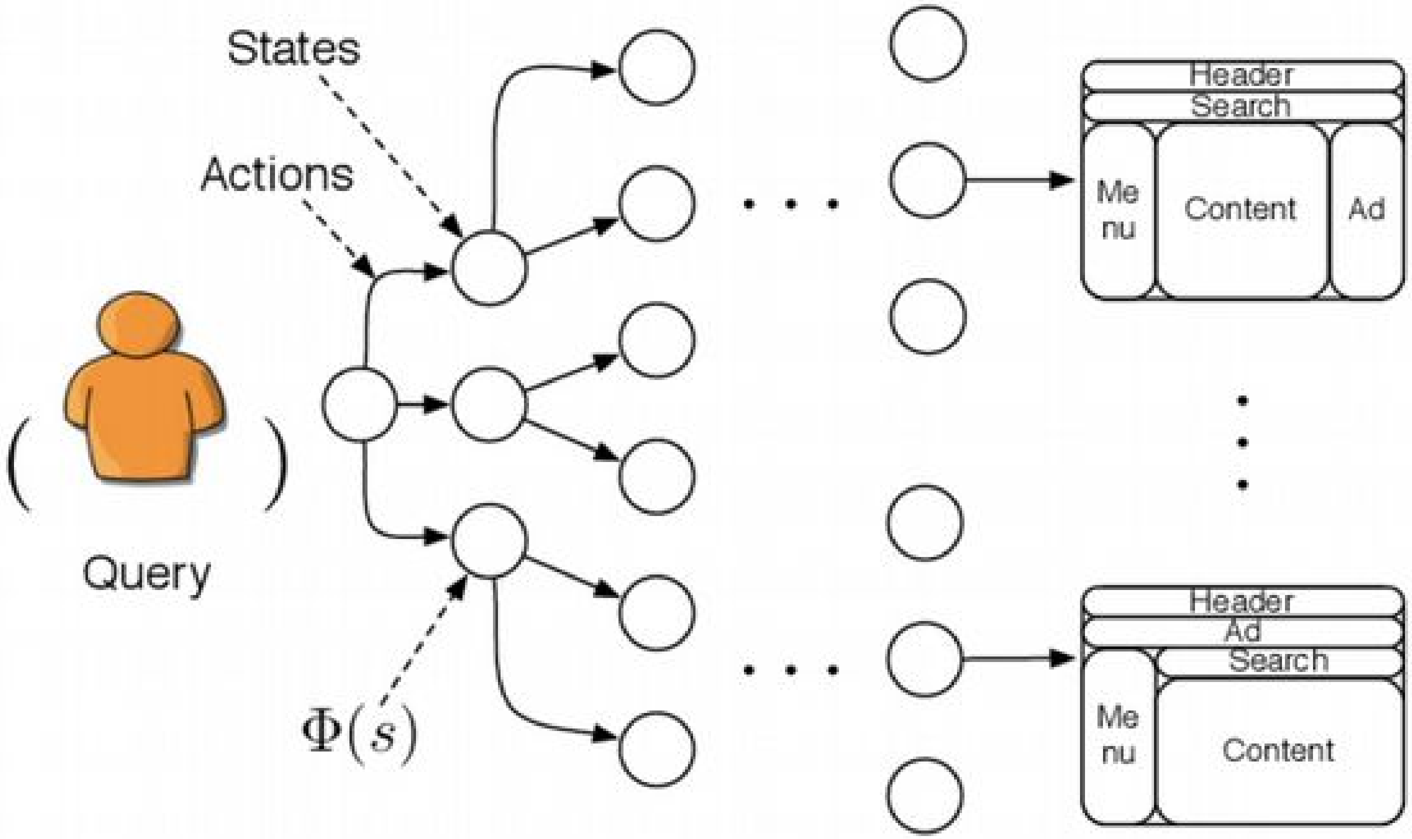


# A Search Problem




Convert SCB into search problem (search space + actions).

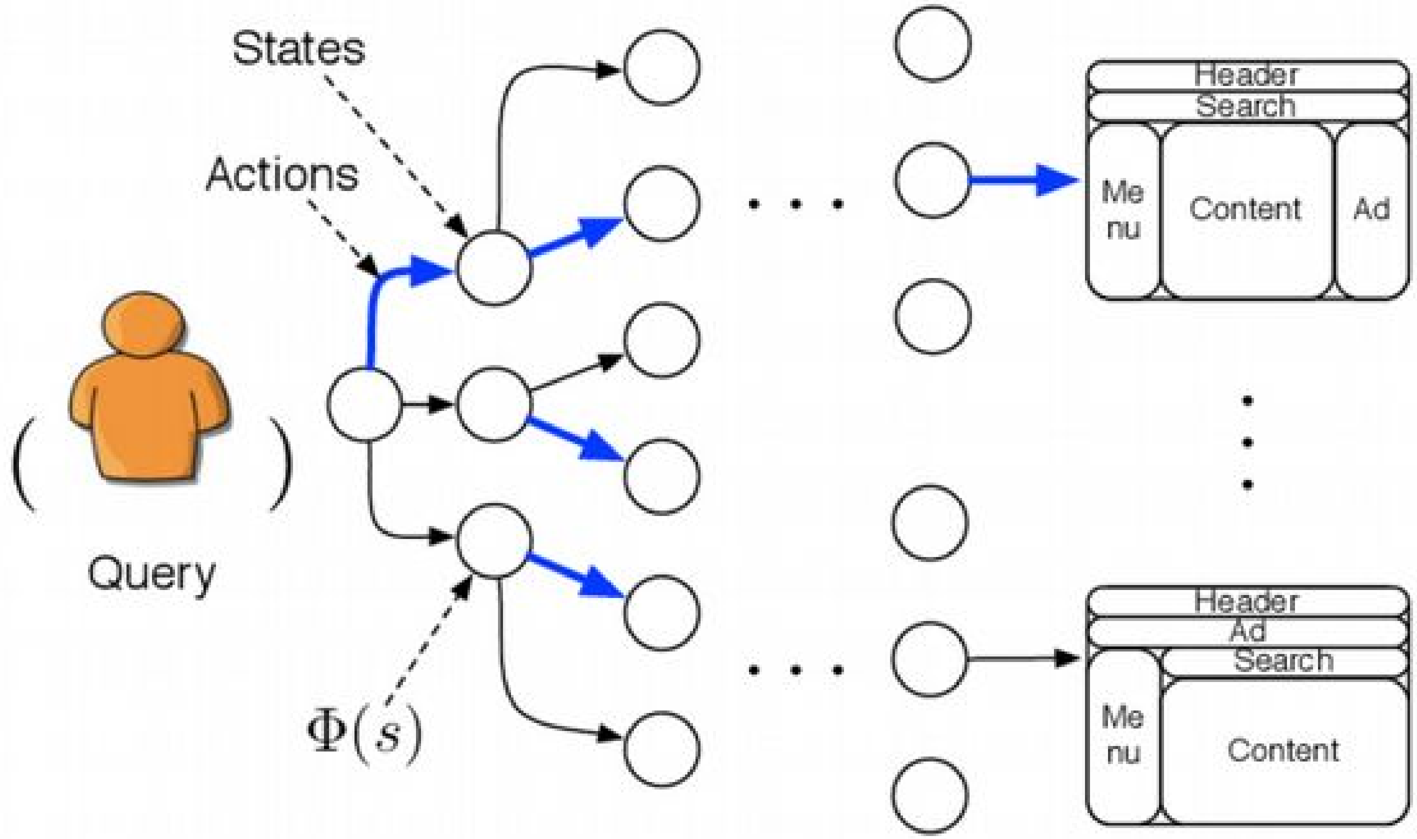
# A Search Problem




Define structured features over each state.

- Learn policy mapping features to actions.

# A Search Problem




Use status quo system as reference policy (Ref).

- Goal is to improve on Ref.

# Connection with Learning to Search

| | L2S | SCB |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |

# Connection with Learning to Search

| | L2S | SCB |
|---|---|---|
| Search Space | States, Actions, Features | |
| | | |
| | | |
| | | |
| | | |

# Connection with Learning to Search

| | L2S | SCB |
|---|---|---|
| Search Space | States, Actions, Features | |
| Feedback | Labels | Single Loss |
| | | |
| | | |
| | | |

# Connection with Learning to Search

| | L2S | SCB |
|---|---|---|
| Search Space | States, Actions, Features | |
| Feedback | Labels | Single Loss |
| Ref Quality | Optimal | Can be bad |
| | | |
| | | |

# Connection with Learning to Search

| | L2S | SCB |
|---|---|---|
| Search Space | States, Actions, Features | |
| Feedback | Labels | Single Loss |
| Ref Quality | Optimal | Can be bad |
| Ref Availability | Training only | Train and Test |
| | | |

# Connection with Learning to Search

| | L2S | SCB |
|---|---|---|
| Search Space | States, Actions, Features | |
| Feedback | Labels | Single Loss |
| Ref Quality | Optimal | Can be bad |
| Ref Availability | Training only | Train and Test |
| Goal | Imitate Ref | Improve upon Ref |

# Connection with Learning to Search

| | L2S | SCB |
|---|---|---|
| Search Space | States, Actions, Features | |
| Feedback | Labels | Single Loss |
| Ref Quality | Optimal | Can be bad |
| Ref Availability | Training only | Train and Test |
| Goal | Imitate Ref | Improve upon Ref |

Existing L2S algorithms give:

$$L(\pi) \leq L(\pi^{\text{ref}}) + o(1)$$

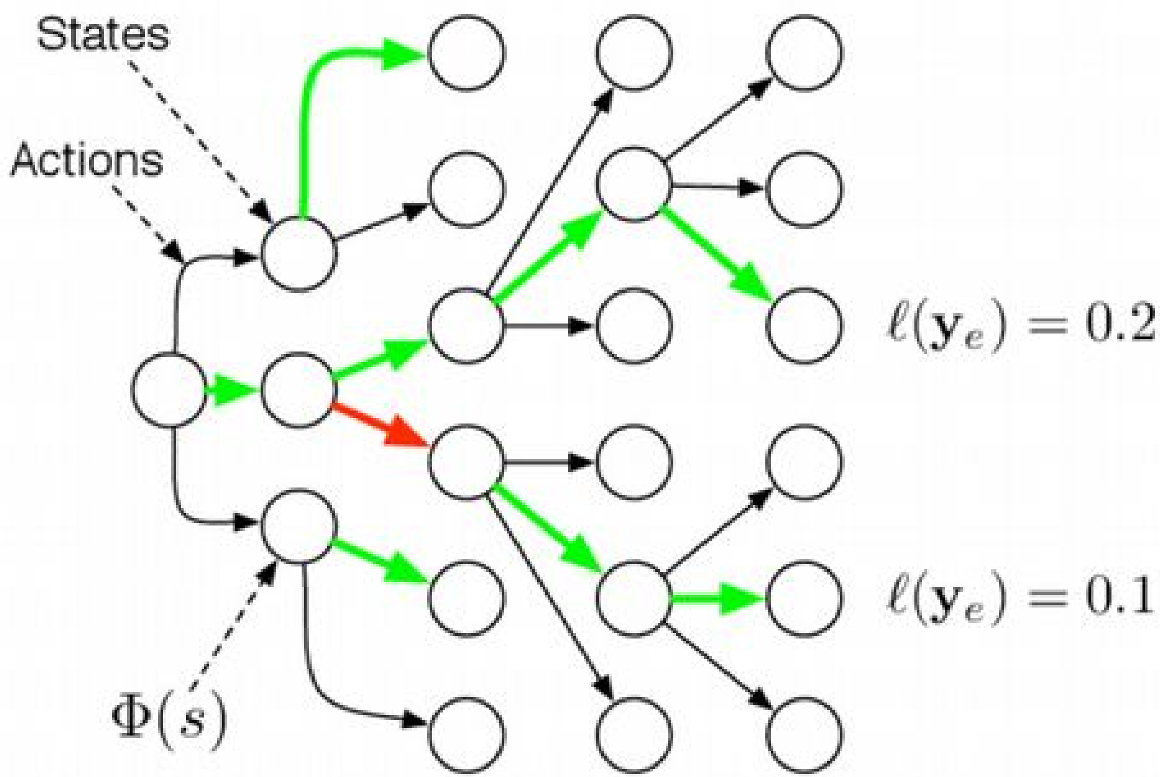
# Main Goal

Learning to Search with:

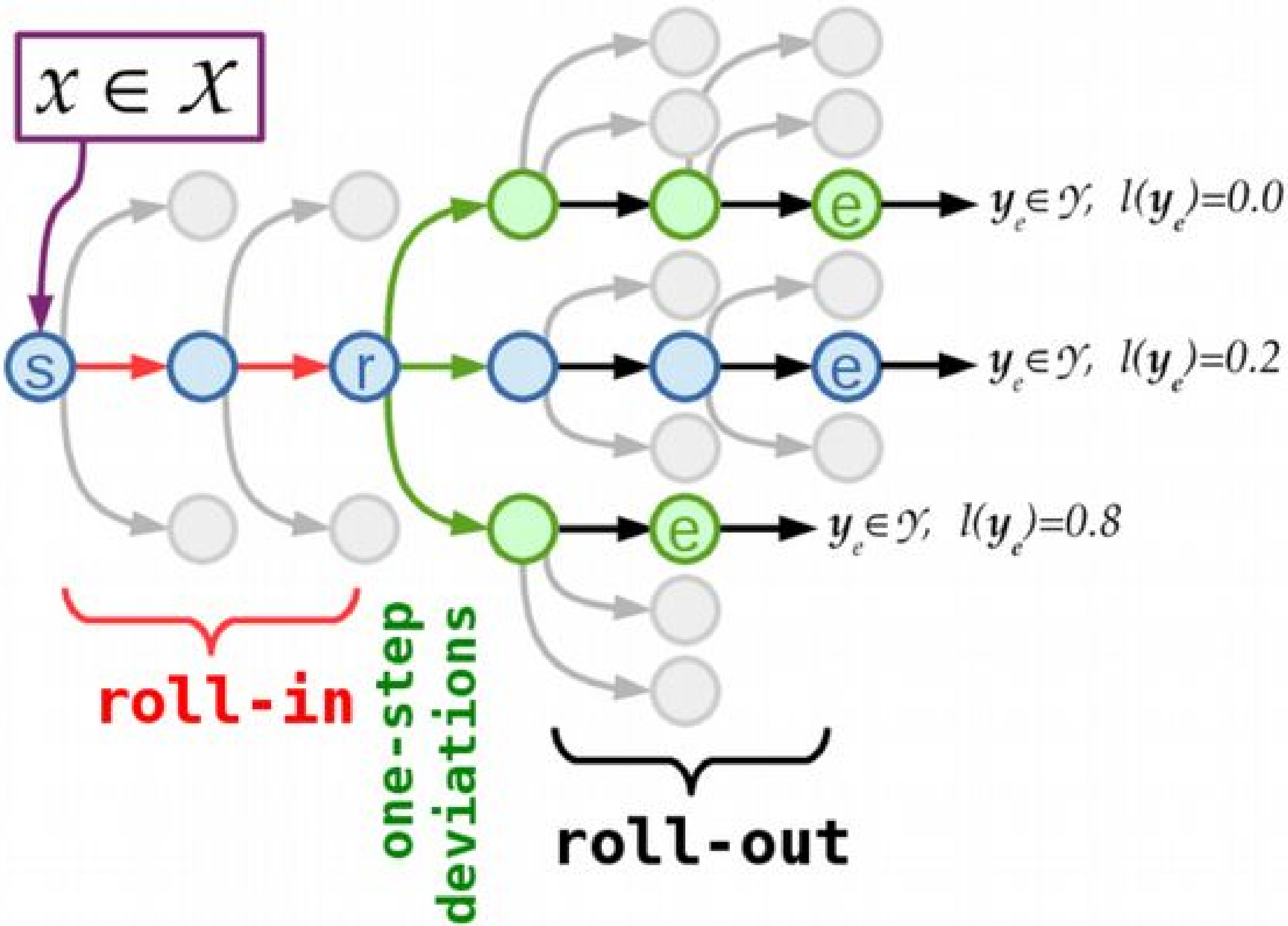
1. A suboptimal reference $\Rightarrow$ Learn policy that improves on Ref.
2. Partial feedback.

# DISEDERATA

- Compete with Ref.
  - Global optimality if Ref is optimal and realizable.
- Local optimality.
  - Compete with your own one-step deviations.



# Exploration in Search Space




- Roll-in choice affects what states we train on.
- Roll-out choice affects how we score actions.

# Effect of Roll-in and Roll-out Policies

| roll-out → <br> ↓ roll-in | Reference | Half/Half | Learned |
|---|---|---|---|
| Reference | Inconsistent | | |
| Learned | No local opt | Good | RL |

# Effect of Roll-in and Roll-out Policies

| roll-out $\rightarrow$ <br> $\downarrow$ roll-in | Reference | Half/Half | Learned |
|---|---|---|---|
| Reference | **Inconsistent** | | |
| Learned | No local opt | Good | RL |

**Theorem**

*Roll-in with Ref can generate a model with unbounded structured regret but zero cost-sensitive regret.*

- States trained on are not representative of those seen at prediction time.

# Effect of Roll-in and Roll-out Policies

| roll-out → <br> ↓ roll-in | Reference | Half/Half | Learned |
|---|---|---|---|
| Reference | Inconsistent | | |
| Learned | **No local opt** | Good | RL |

**Theorem**

*Roll-out with Ref may cause the learned policy to fail to converge to a local optimum if the reference policy is suboptimal.*

- Causes poor assessment of comparison policies.

# Effect of Roll-in and Roll-out Policies

| roll-out → <br> ↓ roll-in | Reference | Half/Half | Learned |
|---|---|---|---|
| Reference | Inconsistent | | |
| Learned | No local opt | Good | **RL** |

**Theorem**

*Roll-in and roll-out with the learned policy ignores Ref and is equivalent to reinforcement learning.*

# Effect of Roll-in and Roll-out Policies

| roll-out → <br> ↓ roll-in | Reference | Half/Half | Learned |
|---|---|---|---|
| Reference | | Inconsistent | |
| Learned | No local opt | **Good** | RL |

**Theorem**

*LOLS minimizes a combination of regret to Ref and regret to its own one-step deviations.*

# LOLS Regret Bound

### Theorem

*LOLS minimizes a combination of regret to Ref and regret to its own one-step deviations.*

$$\frac{1}{2}\underbrace{\Big(L(\hat{\pi}) - L(\pi^{\mathsf{ref}})\Big)}_{\text{Regret to Ref}} + \frac{1}{2}\underbrace{\Big(L(\hat{\pi}) - L(\pi^{\mathsf{dev}})\Big)}_{\text{Regret to devs}} \text{ is small.}$$

# LOLS Regret Bound

### Theorem

*LOLS minimizes a combination of regret to Ref and regret to its own one-step deviations.*

$$\underbrace{\frac{1}{2}\left(L(\hat{\pi}) - L(\pi^{\mathsf{ref}})\right)}_{\text{Regret to Ref}} + \underbrace{\frac{1}{2}\left(L(\hat{\pi}) - L(\pi^{\mathsf{dev}})\right)}_{\geq 0} \text{ is small.}$$

- Competes with Ref.

# LOLS Regret Bound

**Theorem**

*LOLS minimizes a combination of regret to Ref and regret to its own one-step deviations.*

$$\underbrace{\frac{1}{2}\left(L(\hat{\pi}) - L(\pi^{\mathsf{ref}})\right)}_{\geq 0} + \underbrace{\frac{1}{2}\left(L(\hat{\pi}) - L(\pi^{\mathsf{dev}})\right)}_{\text{Regret to devs}} \text{ is small.}$$

- Competes with Ref.
- Locally optimal when Ref is optimal (even if unrealizable).

# LOLS Regret Bound

**Theorem**

*LOLS minimizes a combination of regret to Ref and regret to its own one-step deviations.*

$$\underbrace{\frac{1}{2}\left(L(\hat{\pi}) - L(\pi^{\mathsf{ref}})\right)}_{\text{Regret to Ref}} + \underbrace{\frac{1}{2}\left(L(\hat{\pi}) - L(\pi^{\mathsf{dev}})\right)}_{\text{Regret to devs}} \text{ is small.}$$

- Competes with Ref.
- Locally optimal when Ref is optimal (even if unrealizable).
- If Ref suboptimal, either locally optimal or better than Ref.

# Local Optimality is Hard

Finding local optimum could be hard without further assumptions.

**Theorem**

*Can require $\Omega(2^T)$ cost-sensitive classification examples to reach local optimum.*

$T$ is the number of decisions per example.

# Dependency Parsing

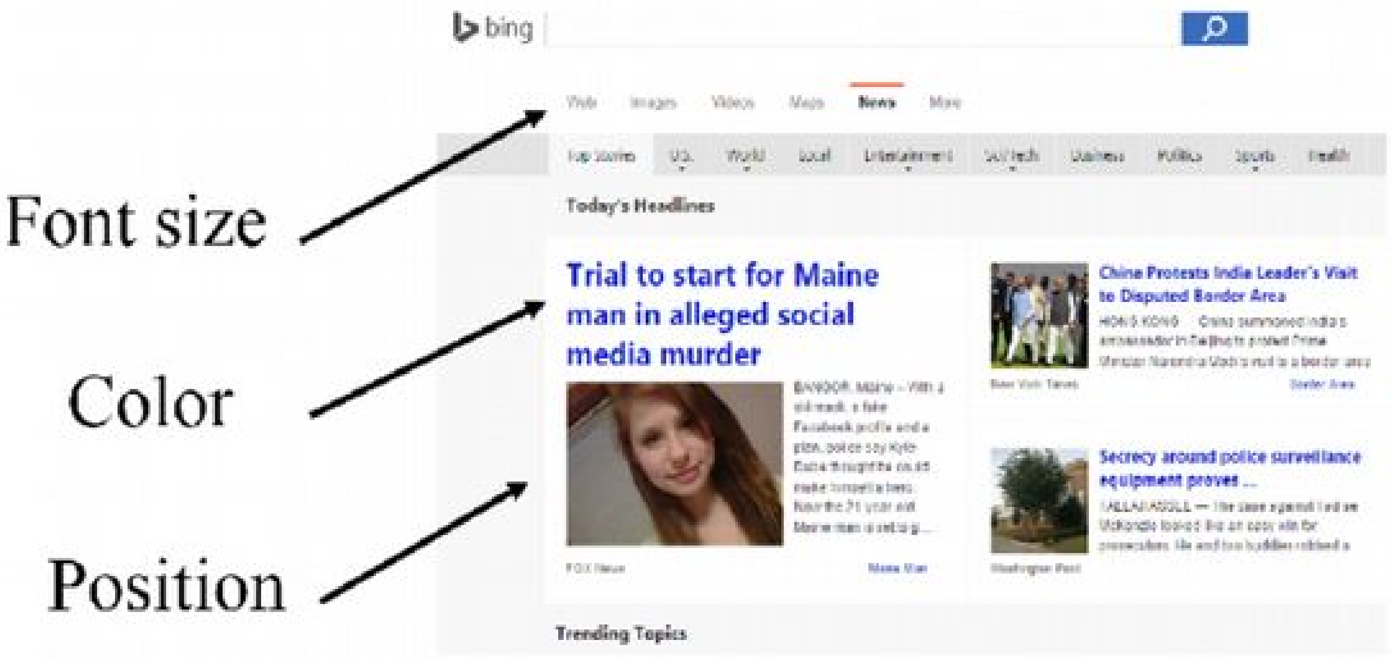
Find the dependency structure of words in a sentence.

| roll-out → <br> ↓ roll-in | **Reference** | **Mixture** | **Learned** |
|---|---|---|---|
| | | **Reference is optimal** | |
| **Reference** | 87.2 | 89.7 | 88.2 |
| **Learned** | **90.7** | 90.5 | 86.9 |
| | | **Reference is suboptimal** | |
| **Reference** | 83.3 | 87.2 | 81.6 |
| **Learned** | 87.1 | **90.2** | 86.8 |
| | | **Reference is bad** | |
| **Reference** | 68.7 | 65.4 | 66.7 |
| **Learned** | 75.8 | **89.4** | 87.5 |

LOLS always good, even with suboptimal Ref.

# Structured Contextual Bandit

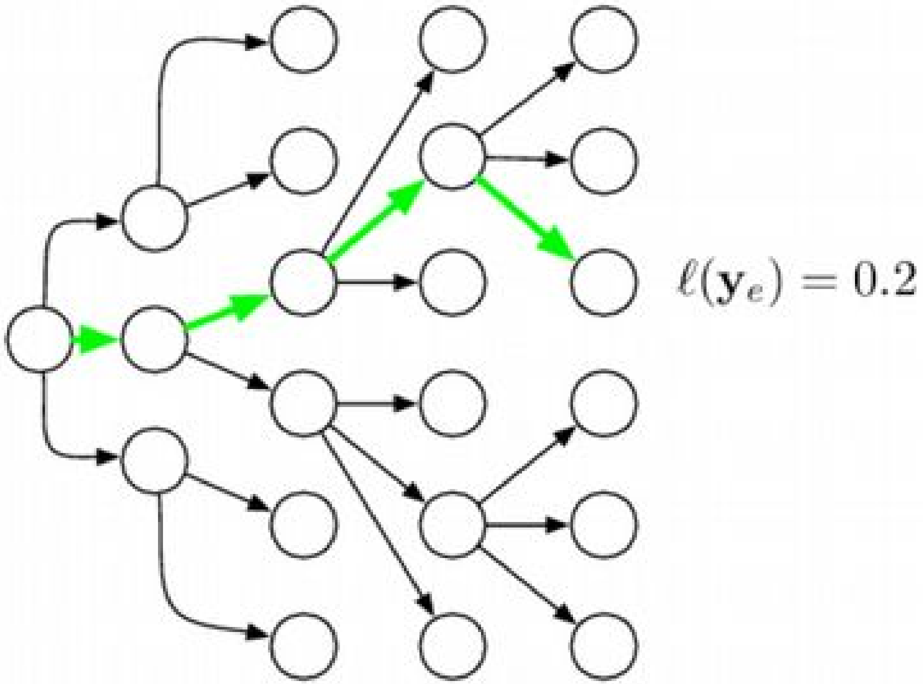
- Loss of a single structured label can be observed.
- Reference policy is not optimal under this setting.



# Structured Contextual Bandit

We adapt an $\epsilon$-greedy algorithm
Upon receiving an example:

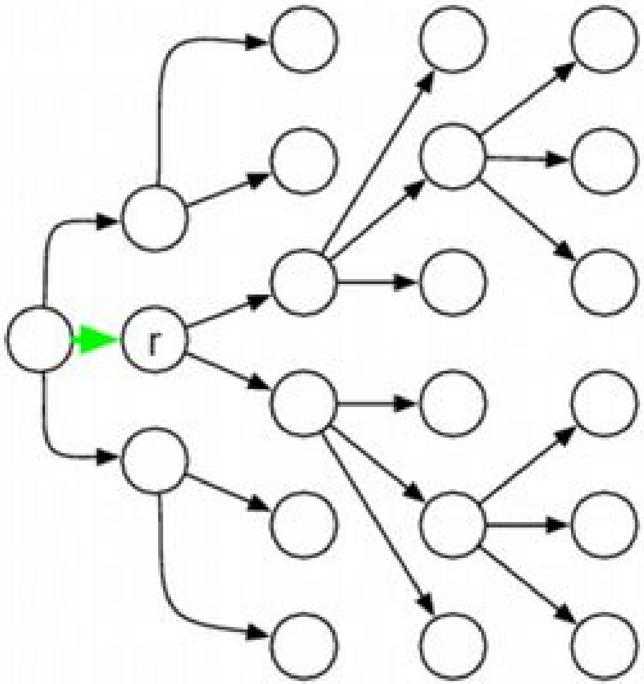
- w/ prob. $1 - \epsilon$: follow the current policy



# Structured Contextual Bandit

We adapt an $\epsilon$-greedy algorithm
Upon receiving an example:

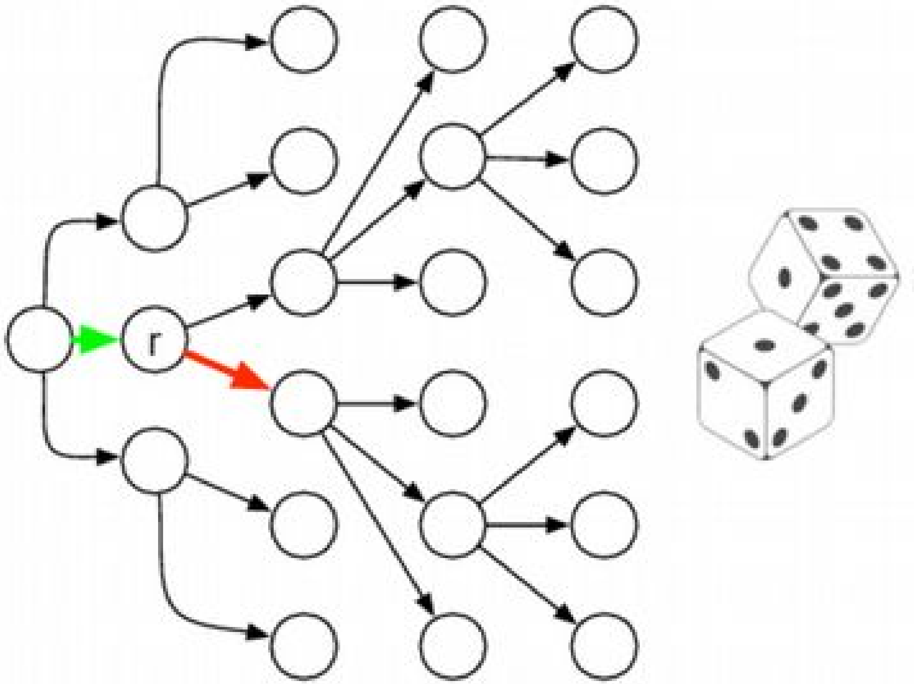
- w/ prob. $1 - \epsilon$: follow the current policy
- w/ prob. $\epsilon$: perform a randomized LOLS update.



# Structured Contextual Bandit

We adapt an $\epsilon$-greedy algorithm
Upon receiving an example:

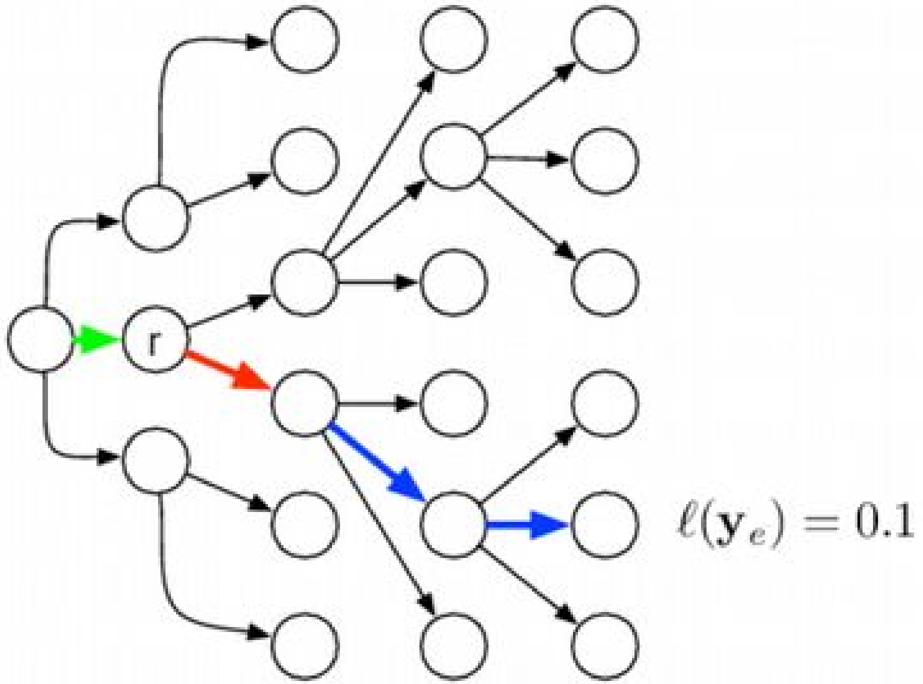
- w/ prob. $1 - \epsilon$: follow the current policy
- w/ prob. $\epsilon$: perform a randomized LOLS update.



# Structured Contextual Bandit

We adapt an $\epsilon$-greedy algorithm
Upon receiving an example:

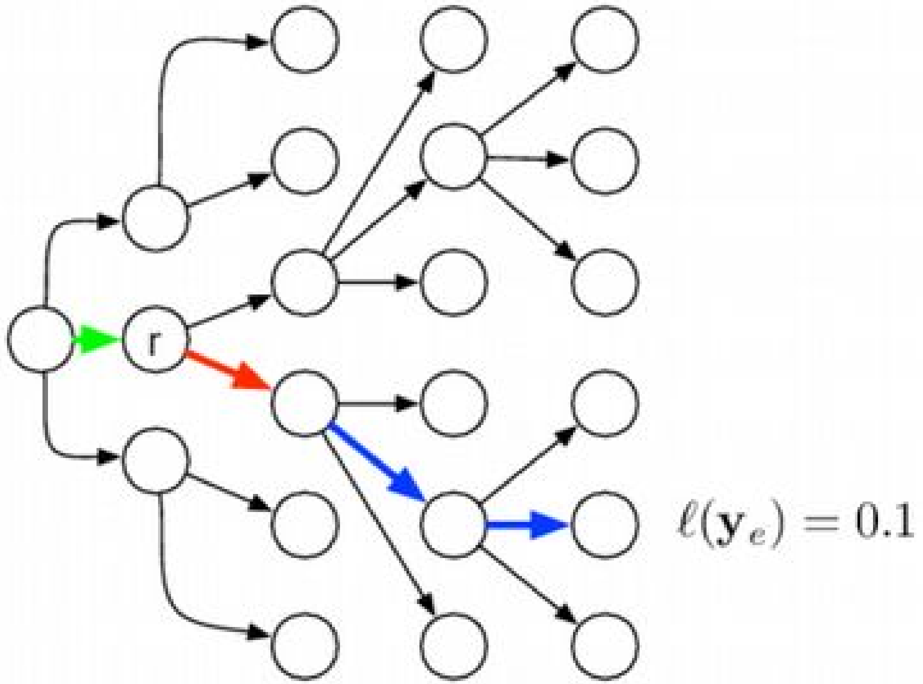
- w/ prob. $1 - \epsilon$: follow the current policy
- w/ prob. $\epsilon$: perform a randomized LOLS update.



# Structured Contextual Bandit

We adapt an $\epsilon$-greedy algorithm
Upon receiving an example:

- w/ prob. $1 - \epsilon$: follow the current policy
- w/ prob. $\epsilon$: perform a randomized LOLS update.




Regret against ref and deviations is still bounded.

# Outline

- Background: learning
- Stuff I did in the Spr
  - Imperative DSL/libra
  - SOTA examples for
  - Learning to search un
  - Hardness results for lear
  - Active learning for acceleratin
- Stuff I'm trying to do now
  - Distant supervision
  - Mashups with recurrent neural networks

Observation: rollouts at all time steps not equally useful

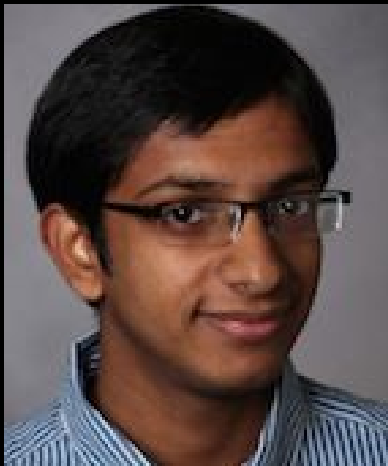
Solution: importance-weighted active learning selection of where to rollout vs skip

Hacky heuristic: 5* speedup, slightly increased accuracy

Training RNNs with LOLS yields drastic increases in performance on non-adversarial synthetic data

# Distant supervision

- Learning with a human in the loop
- Repeat forever:
  - Information need
  - Machine makes complex prediction
  - Human is happy or unhappy, provides extra feedback
  - Machine learns
  - **Human learns**
- How to handle the last step?



Alekh
Agarwal

Kai-Wei
Chang

Akshay
Krishnamurthy

John
Langford

Alina
Beygelzimmer

Paul
Mineiro

Stéphane
Ross

He
He

- Novel programming paradigm for integrating ML into software
- State of the art results on many tasks, very quickly, little code
- New problems, new algorithms
- Positive results (notion of local optimality, and regret guarantees)
- Negative results (hardness of exact local optimality)
- Lots of places to go from here...

Thanks! Questions?