Lecture #5: Classifier-based Structured Prediction

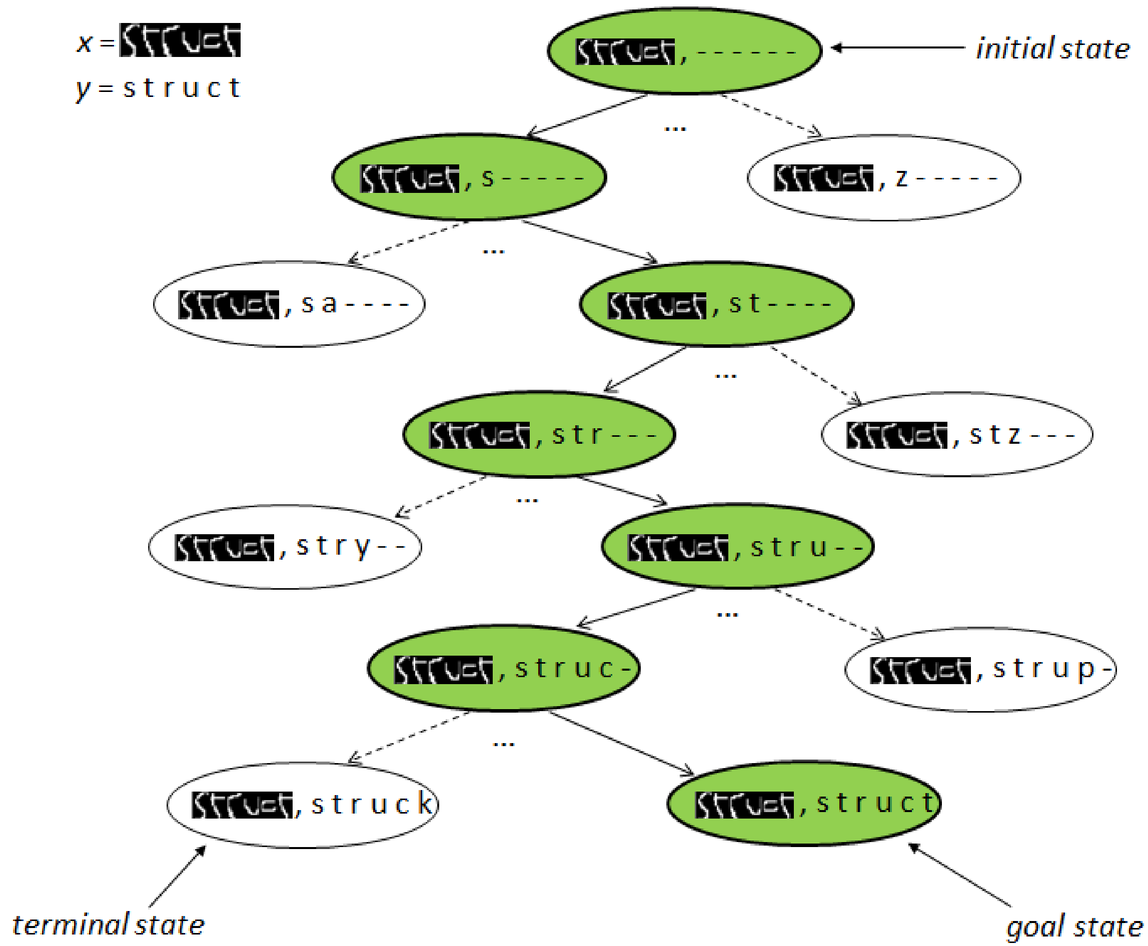
Acknowledgements: Imitation learning pictures are from Drew Bagnell (CMU)

Classifier-based Structured Prediction

- **Special case of LaSO Framework**
  - LaSO instantiated with greedy search
- **Reduction to Classifier Learning**
  - Learning for structured prediction ⇔ learning a multi-class classifier
  - Good classification performance ⇔ good structured prediction performance
- **Direct connection to imitation learning**
  - Training data (expert) provides the demonstration
  - Learner tries to imitate each decision performed by the expert

Imitation Learning: LaSO (Greedy)

- Reduction to classifier learning
  - 26 classes
- Algorithms
  - Recurrent
  - SEARN
  - Dagger
  - AggreVate
  - LOLS



Aside: Imitation Learning

- Imitation Learning ⬄ Learning from demonstration

Expert Demonstrations → Machine Learning Algorithm → Policy

Aside: Imitation Learning

- Imitation Learning ⇔ Learning from demonstration
- **Example:** Learning to drive from demonstrations

Input

Camera Image

Policy

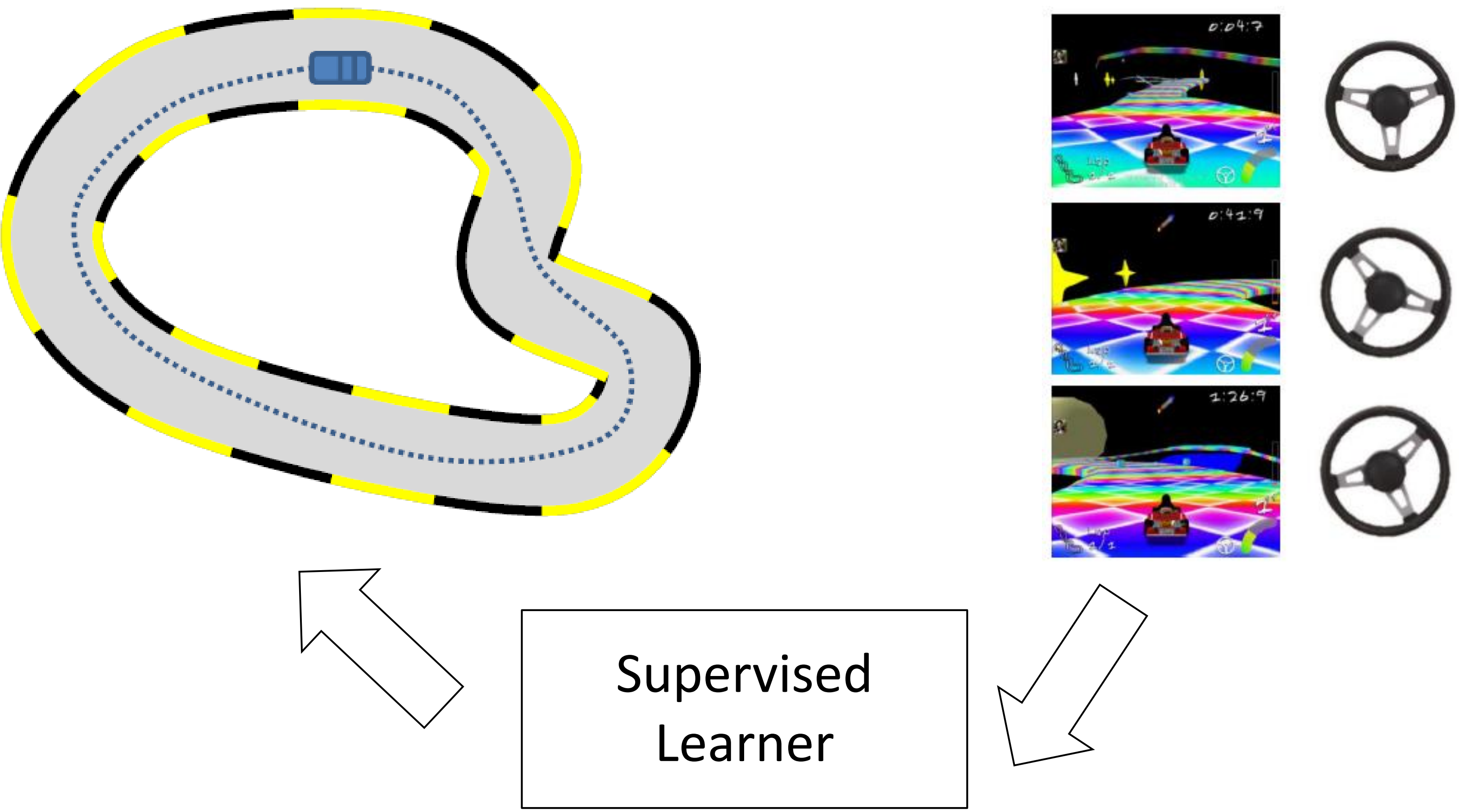
Output

Steering in [-1,1]

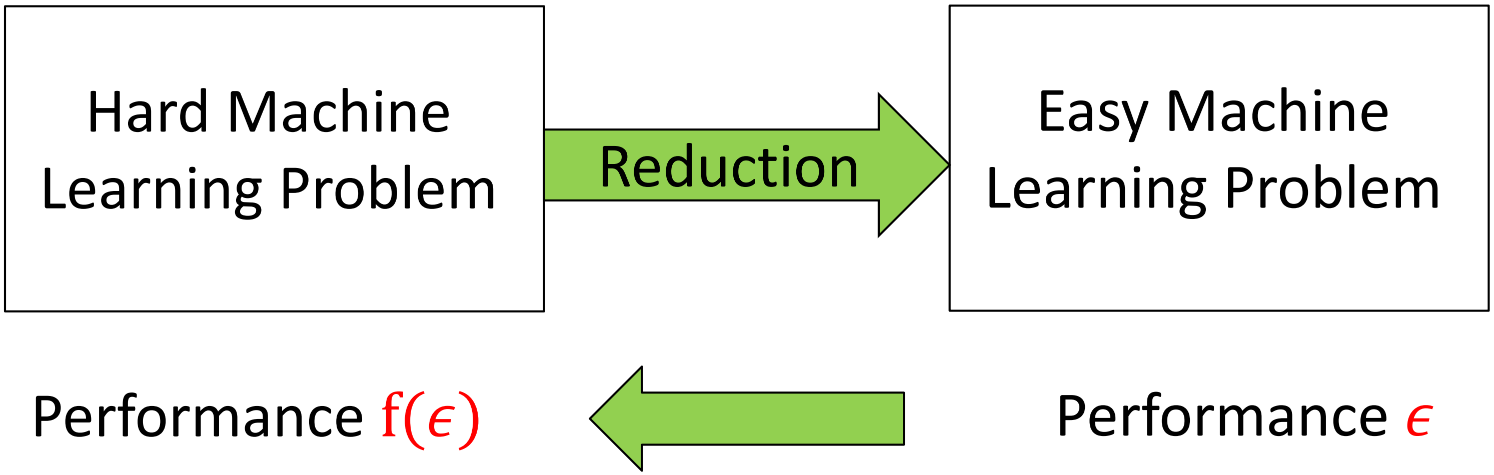
Aside: Imitation Learning

- Supervised Learning Approach

Expert Demonstrations **Training dataset**



Aside: Reductions in Machine Learning



- Reduce complex problem to simpler problem(s)
- A better algorithm for simpler problem means a better algorithm for complex problem
- Composability, modularity, ease-of-implementation

Aside: Reductions in Machine Learning

- **Some Examples:**
  - Multi-class classification to binary classification
  - Cost-sensitive classification to binary classification
  - Reinforcement Learning to classifier learning
  - Planning to classifier learning
  - Imitation learning to supervised learning
  - Structured prediction to classifier learning
  - …

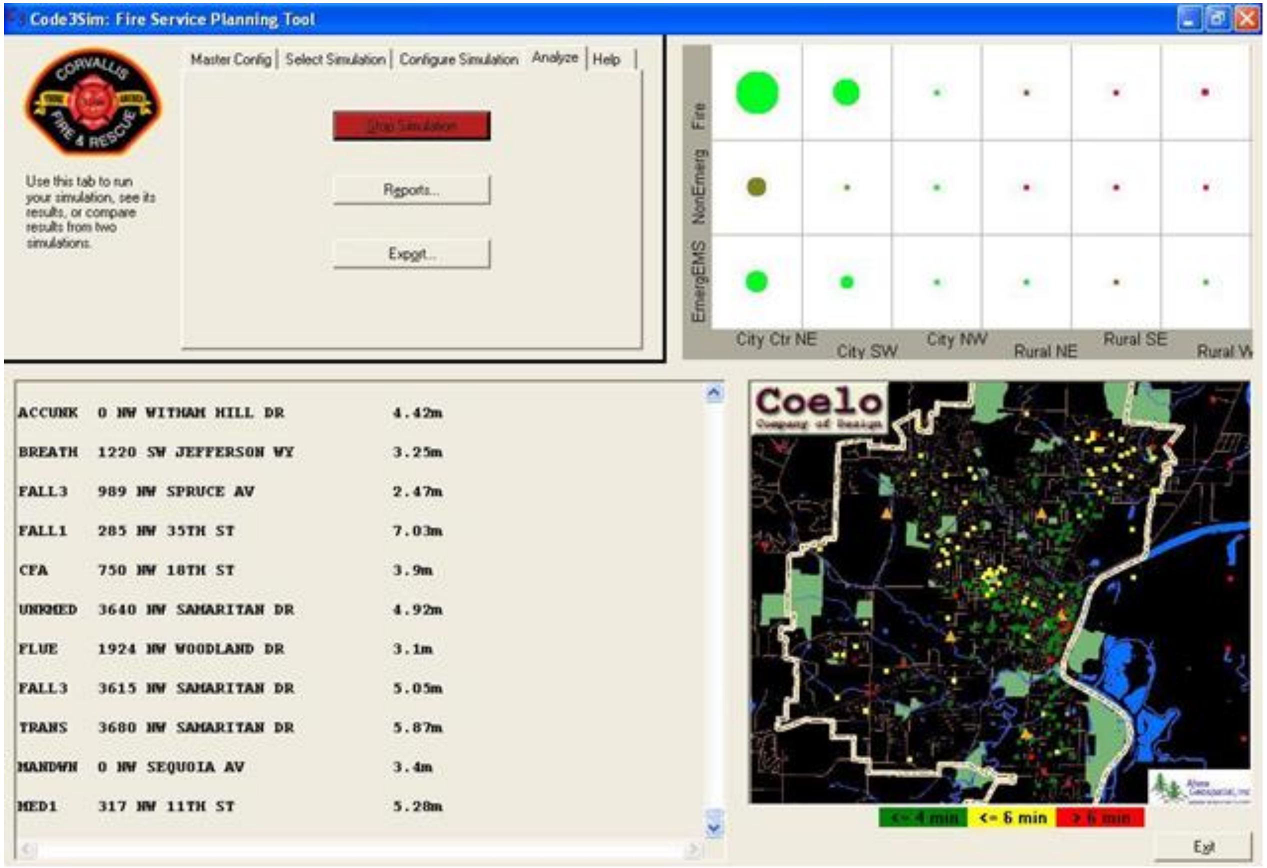
Imitation Learning vs. Reinforcement Learning

- Imitation learning (IL) is a exponentially better framework than RL
  - Assumes the availability of a good oracle or expert to drive the learning process
- At a very high-level, the difference is similar to supervised learning vs. exploratory learning
- Near-optimal RL is intractable for large state spaces
- When it is possible to learn a good approximation of the expert, the amount of data and time required to learn a expert policy is polynomial (quadratic or less) in time horizon (no. of decision steps)

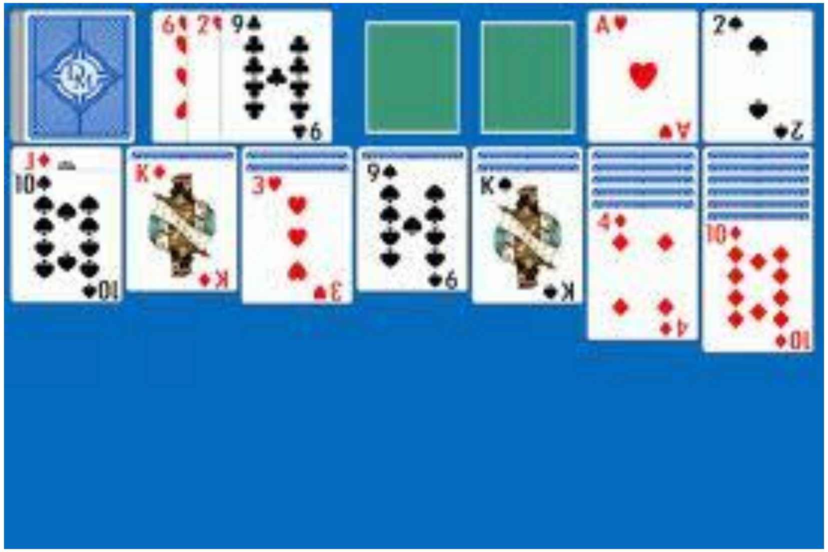
Reinforcement Learning: Introduction and Fundamental Concepts

Automated Planning Under Uncertainty

Optimizing Fire & Rescue Response Policies



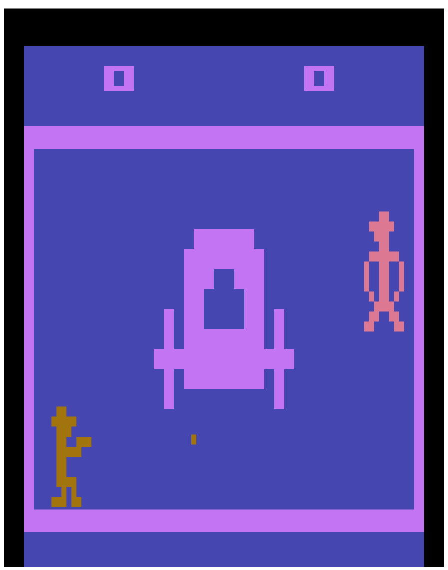
Automated Planning Under Uncertainty

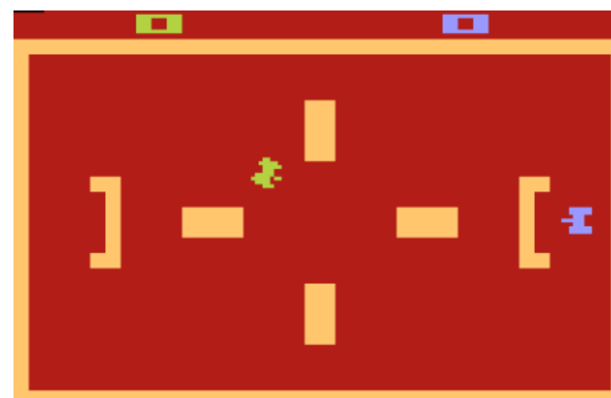


Klondike Solitaire

Real-Time Strategy Games

AI for General Atari 2600 Games

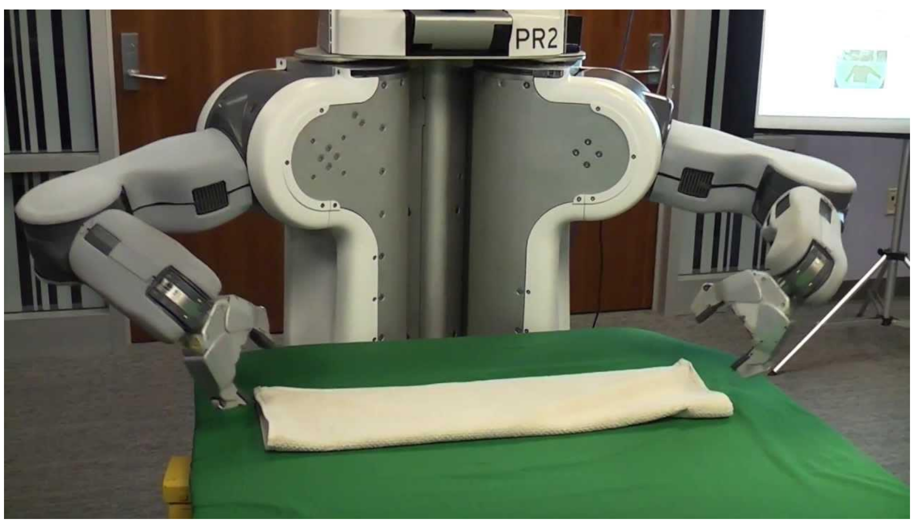




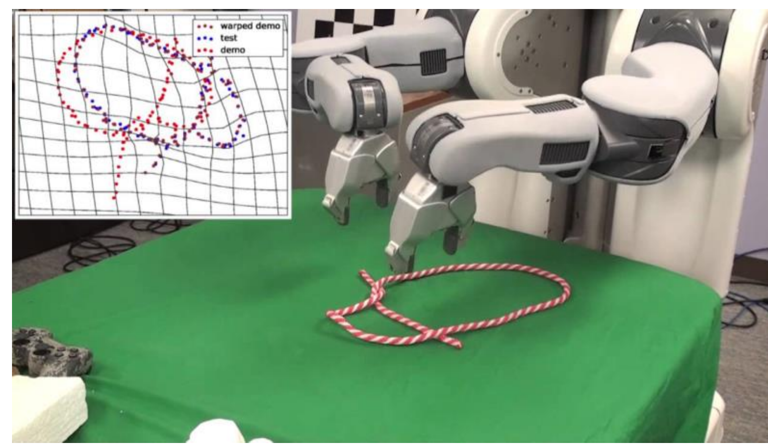
Robotics Control

Helicopter Control

Legged Robot Control



Laundry



Knot Tying

Smart Grids

Some AI Planning Problems

- Health Care
  - Personalized treatment planning
  - Hospital Logistics/Scheduling
- Transportation
  - Autonomous Vehicles
  - Supply Chain Logistics
  - Air traffic control
- Assistive Technologies
  - Dialog Management
  - Automated assistants for elderly/disabled
  - Household robots
  - Personal planner

Common Elements

- We have a controllable system that can change state over time (in some predictable way)
  - The state describes essential information about system (the visible card information in Solitaire)
- We have an objective that specifies which states, or state sequences, are more/less preferred
- Can (partially) control the system state transitions by taking actions
- **Problem:** At each moment must select an action to optimize the overall objective
  - Produce most preferred state sequences

Reinforcement Learning (1)

- **Problem:** Learning to Act (take decisions) by interacting with a system (world) to maximize the cumulative reward
- World is modeled as a **Markov Decision Process (MDP)**
  - Finite states, finite actions, stochastic transition function, and bounded real-valued reward function
- **Assumptions**
  - First-order Markovian dynamics
  - State-dependent reward
  - Stationary dynamics
  - Full observability
- Solution: policies (“plans” for MDPs)

Reinforcement Learning (2)

- **Non-stationary policy**
  - π:S x T → A; π(s,t) tells us what action to take at state s when there are t stages-to-go
  - Need when we are given a finite planning horizon H
- **Stationary policy**
  - π:S → A; π(s) is action to do at state s (regardless of time)
  - Need when we want to continue taking actions indefinitely
- **Value of a policy** π at state **s**
  - Depends on immediate reward, but also what you achieve subsequently by following that policy

$$V_{\pi}^{k}(s) = E\,[\sum_{t=0}^{k} R^{t} \mid \pi, s\,]$$

$$= E\,[\sum_{t=0}^{k} R(s^{t}) \mid a^{t} = \pi(s^{t}, k-t), s^{0} = s\,]$$

RL Algorithms: Big Picture

Planning with known model (MDP)

- **Policy evaluation**:
  - Given an MDP and a (non)stationary policy π
  - Compute finite-horizon value function $V_{\pi}^{k}(s)$ for any k
- **Policy optimization**:
  - Given an MDP and a horizon H
  - Compute the optimal finite-horizon policy
  - Equivalent to computing optimal value function (value iteration)

Planning with unknown model (MDP)

- **Policy evaluation**:
  - Given a stationary policy π, compute the value of policy
  - Passive RL: direct estimation, ADP, TD methods
- **Policy optimization:**
  - Compute the optimal policy
  - Active RL –- ADP, TD, and Q learning

Finite-Horizon: Policy Evaluation

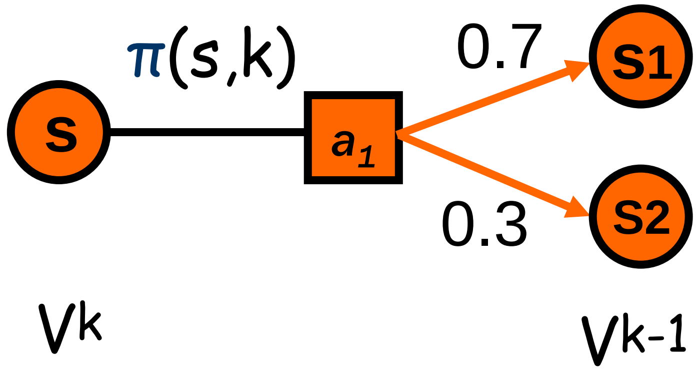
- Can use dynamic programming to compute $V_\pi^k(s)$
  - Markov property is critical for this

(k=0) $$V_\pi^0(s) = R(s), \quad \forall s$$

(k>0) $$V_\pi^k(s) = R(s) + \underbrace{\sum_{s'} T(s, \pi(s,k), s') \cdot V_\pi^{k-1}(s')}_{\text{expected future payoff with } k\text{-1 stages to go}}, \quad \forall s$$

immediate reward

expected future payoff with *k*-1 stages to go



Finite Horizon: Policy Optimization

- Markov property allows exploitation of DP principle for optimal policy construction
  - no need to enumerate $|A|^{Hn}$ possible policies

- Value Iteration

Bellman backup

$$V^0(s) = R(s), \quad \forall s$$

$$\boxed{V^k(s) = R(s) + \max_a \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')}$$

$$\pi^*(s,k) = \arg\max_a \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')$$

V^k is optimal k-stage-to-go value function

Π*(s,k) is optimal k-stage-to-go policy

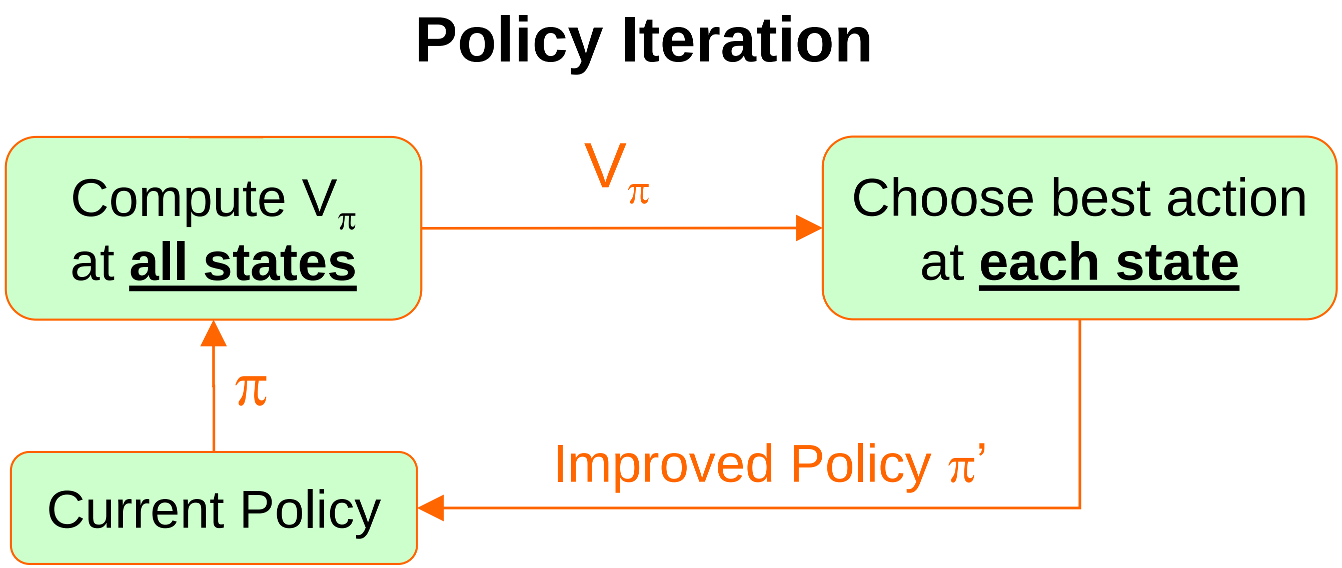
Passive RL: Policy Evaluation w/ unknown MDP

- Monte-Carlo Direct Estimation (model free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints
  - Converges slowly
- Adaptive Dynamic Programming (model based)
  - Harder to implement
  - Each update is a full policy evaluation (expensive)
  - Fully exploits Bellman constraints
  - Fast convergence (in terms of updates)
- Temporal Difference Learning (model free)
  - Update speed and implementation similiar to direct estimation
  - Partially exploits Bellman constraints---adjusts state to ‘agree’ with observed successor
    - Not ***all*** possible successors as in ADP
  - Convergence in between direct estimation and ADP

Active RL: Policy Optimization w/ unknown MDP

- **Exploration vs. Exploitation trade-off**
  - **Exploitation**: To try to get reward. We exploit our current knowledge to get a payoff.
  - **Exploration**: Get more information about the world. How do we know if there is not a pot of gold around the corner.
- **Basic intuition behind most approaches**
  - Explore more when knowledge is weak. Exploit more as we gain knowledge.
- **Exploration policy**
  - We want a policy that is greedy in the limit of infinite exploration (GLIE)
- **ADP-based (model based) RL**
  - Solve for optimal policy given the current model. Take action according to exploration policy. Update model based on new observation. Repeat.
- **TD-based (model based) RL**
  - Start with initial value function. Take action according to exploration policy. Update model based on new observation. Perform TD update to get new value function. Repeat.
- **Q-Learning (model free) RL**
  - Start with initial Q values. Take action according to exploration policy. Perform TD update to get new Q values. Repeat.

Approximate Policy Iteration for Large MDPs

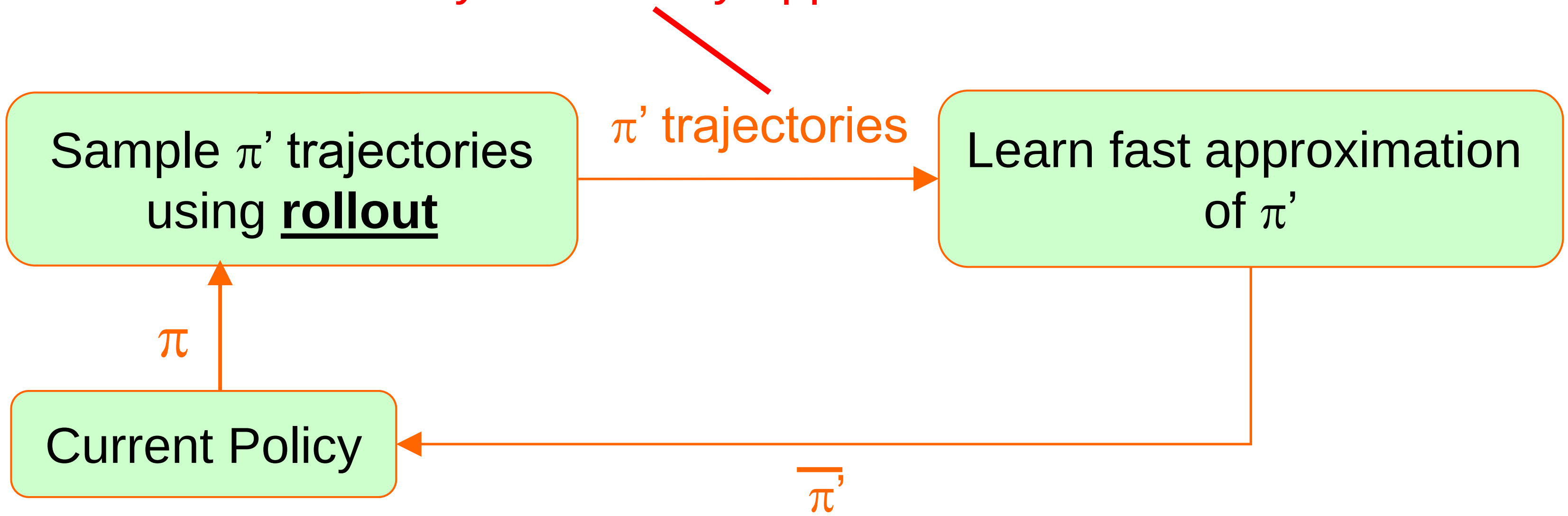


Approximate policy iteration:

- Only computes values and improved action at some states.
- Uses those to infer a fast, compact policy over all states.

Approximate Policy Iteration

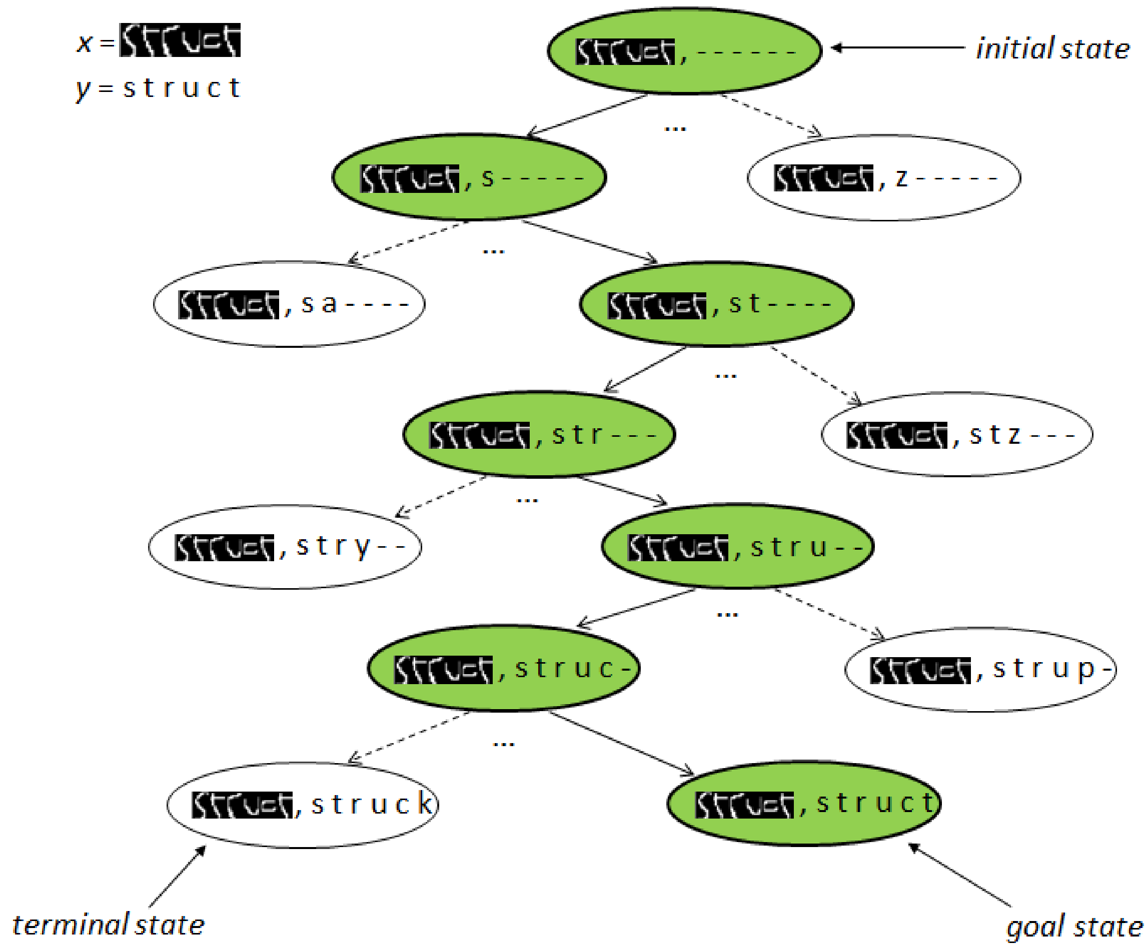
technically rollout only approximates π'.



1. Generate trajectories of rollout policy
   (starting state of each trajectory is drawn from initial state distribution I)
2. "Learn a fast approximation" of rollout policy
3. Loop to step 1 using the learned policy as the base policy

Back to Structured Prediction

- Reduction to classifier learning
  - 26 classes
- Algorithms
  - Recurrent
  - SEARN
  - Dagger
  - AggreVaTe
  - LOLS

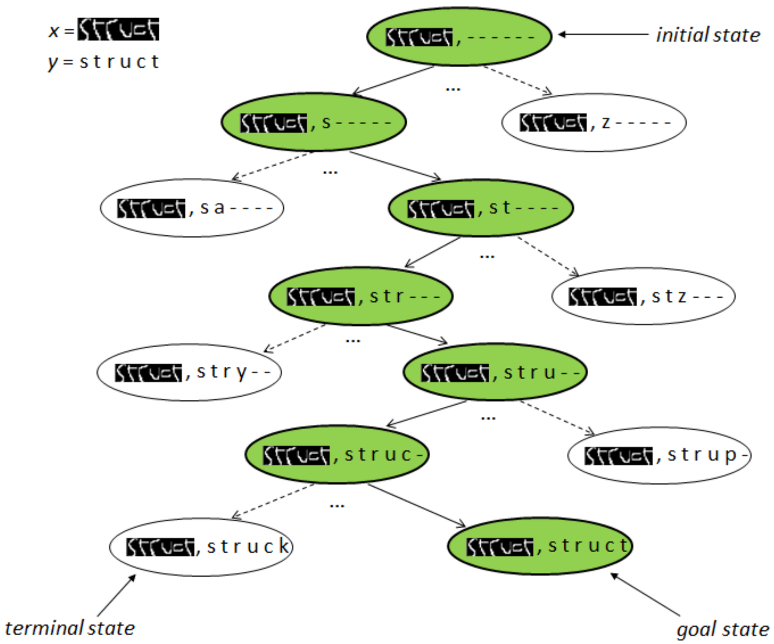


Imitation Learning Approach

- **Expert demonstrations**
  - each training example (input-output pair) can be seen as a "expert" demonstration for sequential decision-making
- **Collect classification examples**
  - Generate a multi-class classification example for each of the decisions
  - Input: f(n), features of the state n
  - Output: y_n, the correct decision at state n
- **Classifier Learning**
  - Learn a classifier from all the classification examples

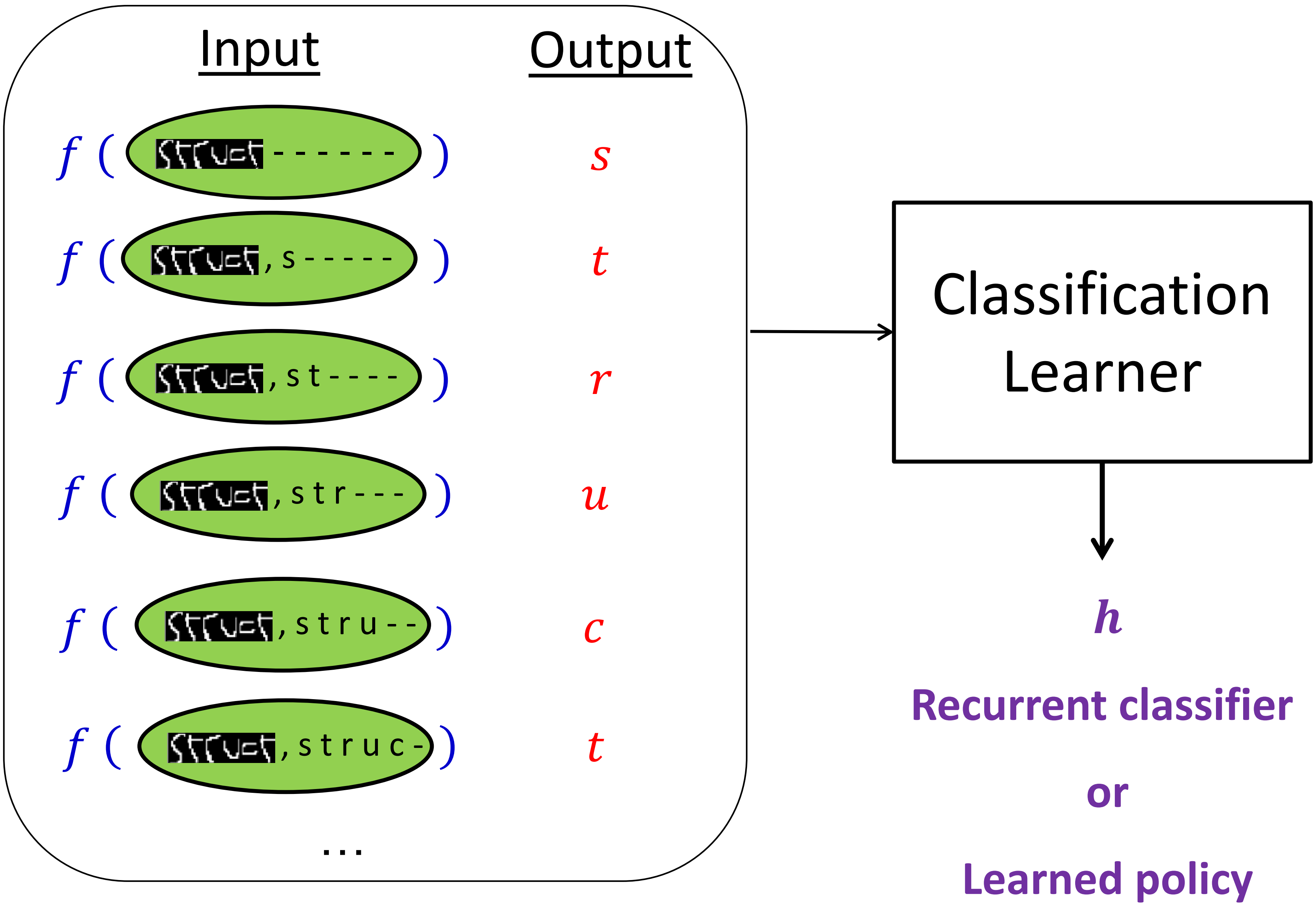
Exact Imitation: Classification examples

- For each training example

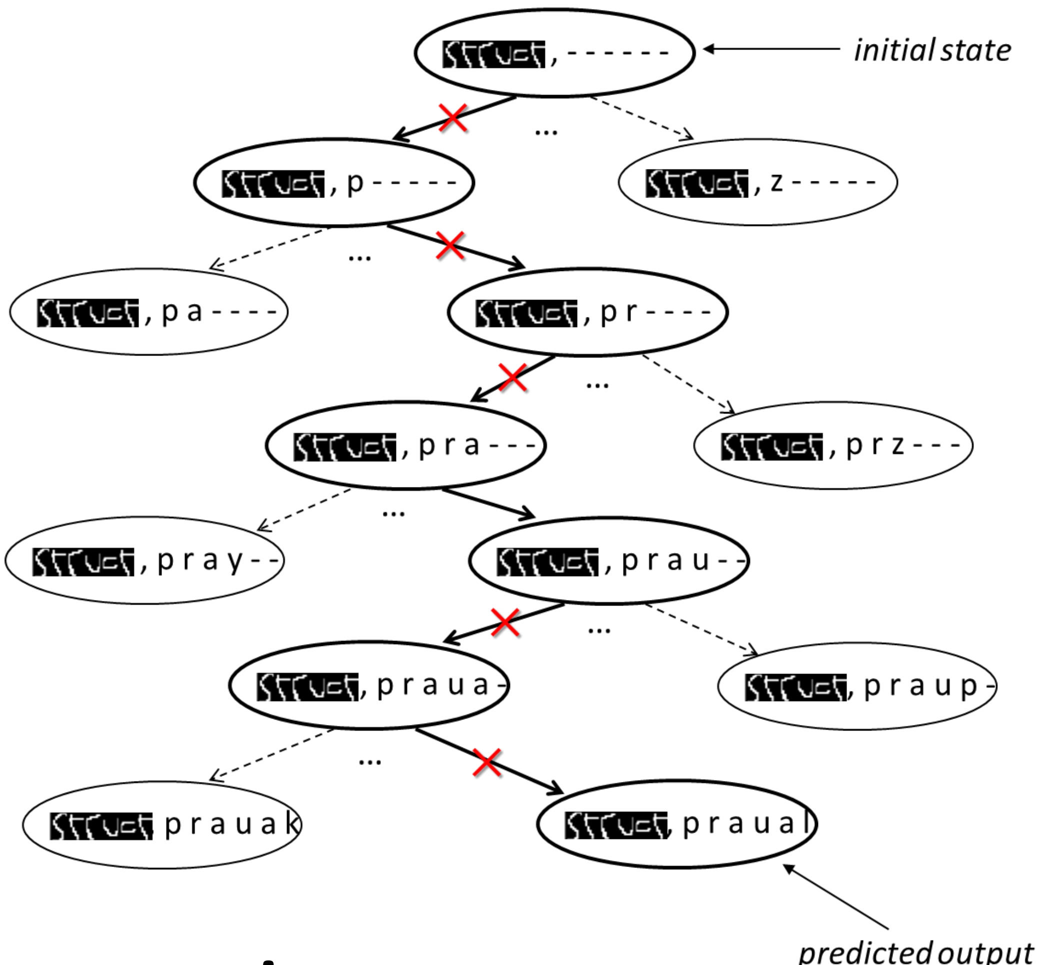


Input	Output
$f($ - - - - - - $)$	s
$f($, s - - - - - $)$	t
$f($, s t - - - - $)$	r
$f($, s t r - - - $)$	u
$f($, s t r u - - $)$	c
$f($, s t r u c - $)$	t

Exact Imitation: Classifier Learning



Learned Recurrent Classifier: Illustration



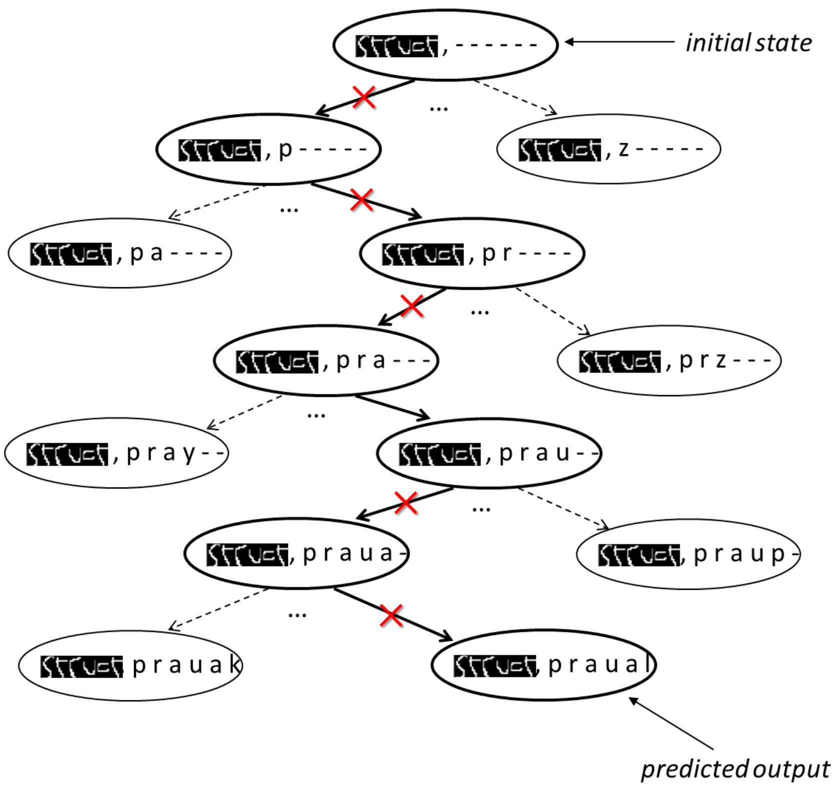
- **Error propagation:**
  - errors in early decisions propagate to down-stream decisions

Recurrent Error

- Can lead to poor global performance
- Early mistakes propagate to downstream decisions: $\mathrm{f}(\epsilon) = O(\epsilon T^2)$, where ϵ is the probability of error at each decision and T is the number of decision steps
- Mismatch between training (IID) and testing (non-IID) distribution
- Is there a way to address this issue?

Addressing Error Propagation

- **<u>Rough Idea:</u>** Iteratively observe current policy and augment training data to better represent important states
- Several variations on this idea [Fern et al., 2006], [Daume et al., 2009], [Xu & Fern 2010], [Ross & Bagnell 2010], [Ross et al. 2011, 2014], [Chang et al., 2015]



- Generate trajectories using current policy (or some variant)
- Collect additional classification examples using optimal policy (via ground-truth output)

Solution #1: Forward Training

- **Non-stationary decision function**
  - One classifier h_i for each decision step i
  - Inspired by Stacking algorithms
- **Key idea:**
  - Sequentially learn classifier h_{i+1} based on the distribution induced by h_i
  - Mistakes grow linearly (instead of quadratically)
- **Learning Algorithm:**
  - Learn h_1 over all the training examples
  - Learn h_2 over all the training examples conditioned on the predictions of h_1
  - So on …

Drawbacks of Forward Training

- Non-stationary decision function
- Learning and Inference doesn't scale if the no. of decision steps (T) is very large
  - for example, driving a car

- Can we address these problems??

Solution #2: SEARN

- Inspired by Conservative Policy Iteration (CPI) algorithm for Reinforcement Learning
- **Key Idea:**
  - Start by imitating the expert
  - Slowly move away from the expert as iterations progress to induce the IID distribution of the learner
- **Stochastic decision function**
  - A sequence of classifiers h_i along with their multinomial distribution
  - To make each decision, toss a coin, and pick one of the classifier to make the decision

Solution #2: SEARN

- Initialize the current policy to optimal policy
- Repeat until convergence
  - For every training example (x, y)
    - Compute the path traversed by the current policy
    - Generate a multi-class example whose classes are possible decisions and whose losses are based on the current policy
  - Learn a new multi-class classifier based on the generated examples (new policy)
  - find an interpolation constant β that can improve the performance on development data
  - Set the current policy to β times new policy plus $1 - \beta$ times the old policy
- Return the current policy without the optimal policy

SEARN: Inner Details

- **Inside each iteration, for one training example $(\boldsymbol{x}, \boldsymbol{y})$**

  - Compute the path traversed by the current policy

Terminal node

s_i = state with partial output

a_i = labeling action

$\hat{y}$ = predicted output

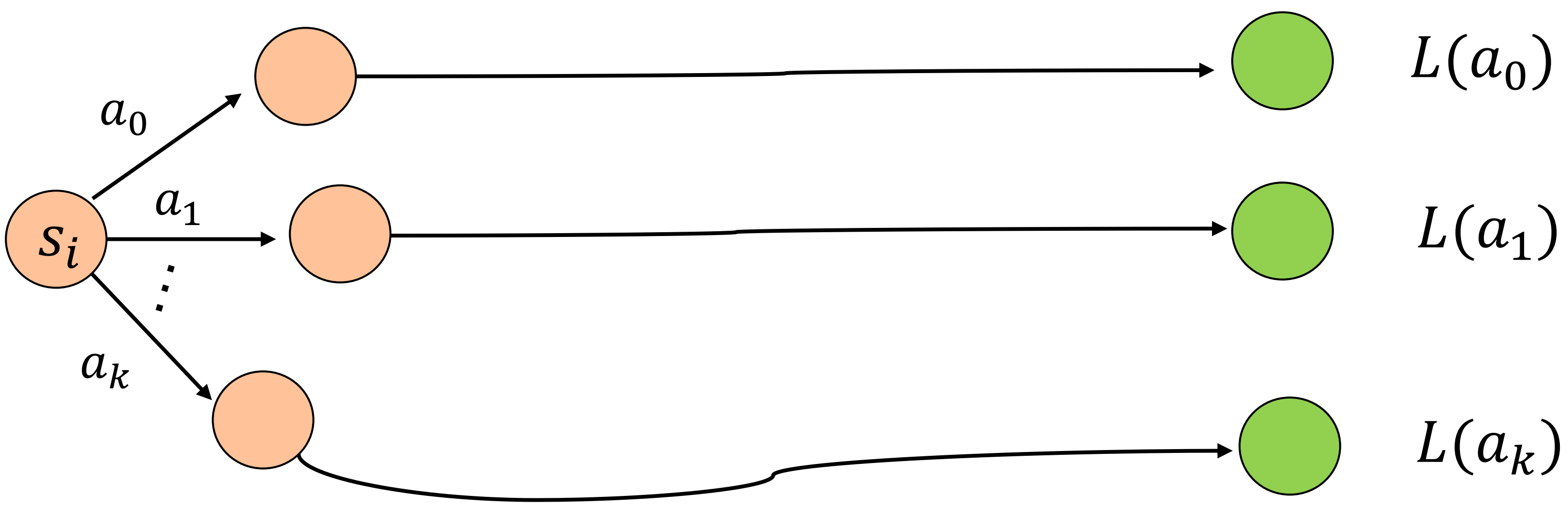
$L(x, \hat{y}, y) \geq 0$ is the loss

SEARN: Inner Details

- **Inside each iteration, for one training example $(\boldsymbol{x}, \boldsymbol{y})$**

> Generate a multi-class example whose classes are possible decisions and whose losses are based on the current policy (for a state s_i on the policy trajectory)

Monte-Carlo estimates: average loss over multiple runs



SEARN: Inner Details

- **Inside each iteration, for one training example** $(\boldsymbol{x}, \boldsymbol{y})$
  - Learn a new multi-class classifier based on the aggregate set of generated examples (new policy)
  - Say $h_i = \boldsymbol{Learn}(\mathrm{D})$

SEARN: Inner Details

- **Inside each iteration, for one training example $(\boldsymbol{x}, \boldsymbol{y})$**
  - Set the current policy to β times new policy plus $1-\beta$ times the old policy
  - $\pi_{i+1} = \beta * h_i + (1-\beta) * \pi_i$
- **Illustration:**
  - $\pi_0 = \pi^*$ (Initialize with optimal policy – expert)
  - $\pi_1 = \beta * h_1 + (1-\beta) * \pi^*$
  - $$\begin{aligned}\pi_2 &= \beta * h_2 + (1-\beta) * \pi_1 \\ &= \beta * h_2 + (1-\beta)\beta * h_1 + (1-\beta)^2 * \pi^*\end{aligned}$$

If β is small (say 0.1), the weight on the expert is gradually decreasing as iterations progress

SEARN: The final policy

- At the end of T iterations, the policy is
  - $\pi_T = \sum_{i=1}^{T} w_i * h_i + w_0 * \pi^*$
- **Remove the optimal policy (expert)** and re-normalize the weights of T classifiers
  - $\pi_{final} = \sum_{i=1}^{T} w_i' * h_i$
- **Making Predictions:**
  - At each decision step, toss a coin and pick one of the T classifiers according to the multinomial distribution to make the decision

Drawbacks of SEARN

- Stochastic decision function
  - We want to avoid stochastic behavior!
- Computing the losses of the current policy at each decision step in each iteration is very expensive
  - Optimal approximation avoids this problem, but the resulting algorithm may not be effective – no free lunch!

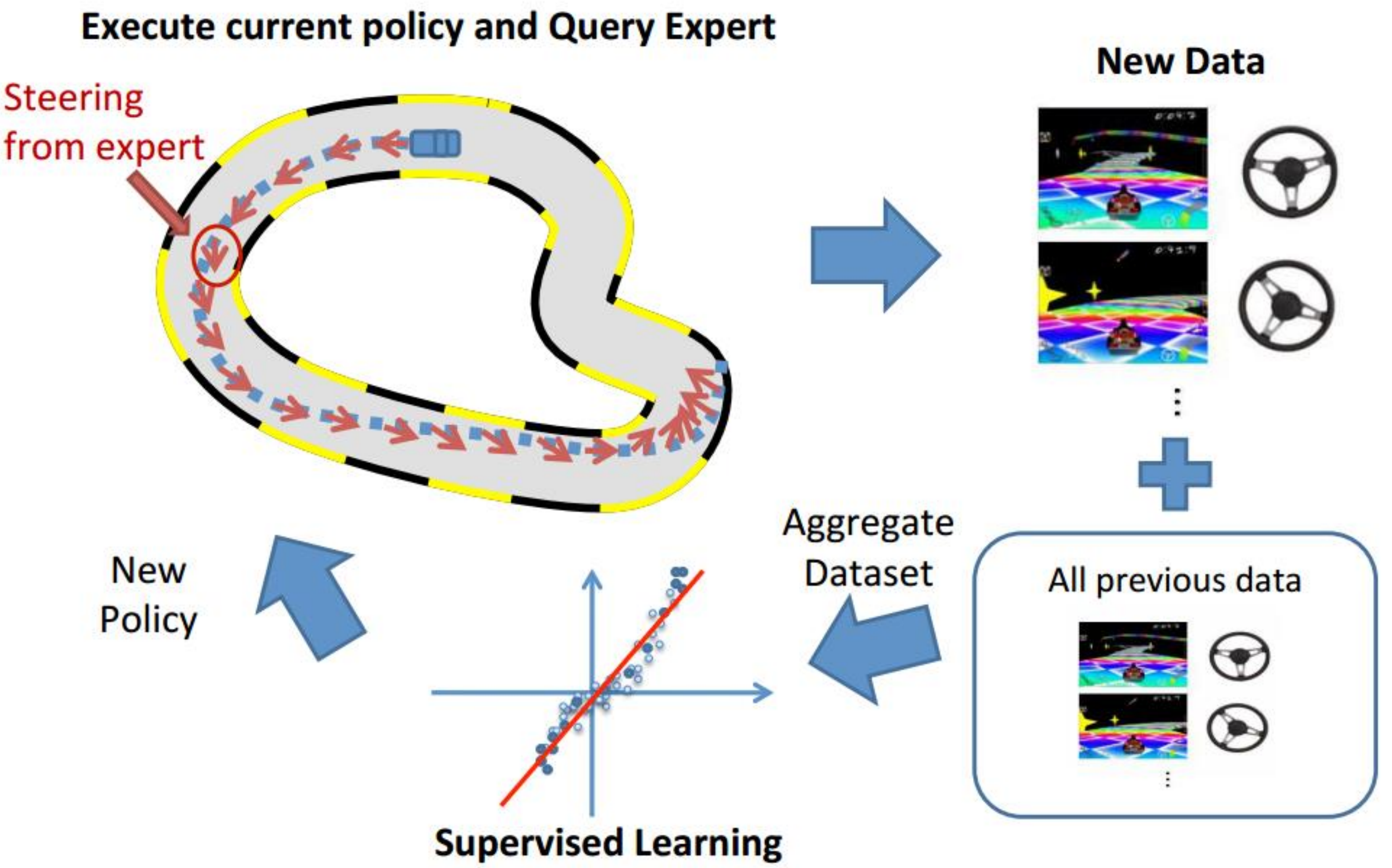
Recap of Last lecture

- SEARN (Search and Learn)
  - Inspired by Conservative Policy Iteration(CPI) algorithm
  - **Key Idea:**
  - Start by imitating the expert
  - Slowly move away from the expert as iterations progress to induce the IID distribution of the learner
  - Reduction to cost-sensitive classification
- Drawbacks
  - Stochastic decision function – not desirable
  - Computing costs via rollouts is expensive

Solution #3: DAgger

- **Key Idea:**
  - Iterative algorithm
  - Aggregate data over several iterations
  - Learn a classifier from the aggregate set of classification examples in each iteration
- **Connections:**
  - Can be seen as a class of online learning algorithms: Follow-The-Leader
  - Strong theoretical guarantees if we use a no-regret classifier learner

DAgger: Illustration



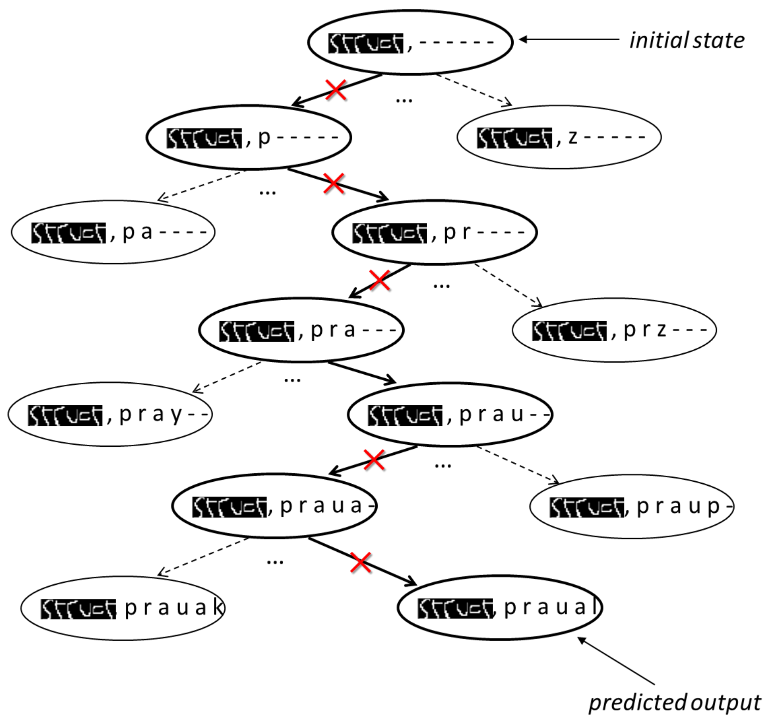
DAgger: Dataset Aggregation

- Collect classification examples from expert trajectories (exact imitation): D_0
- Train a classifier on this data: $h_1 = \boldsymbol{Learn}(D_0)$
- Collect new classification based on the mistakes made by h_1:D_1
- Aggregate data sets: $D = D_0 \cup D_1$
- Train a classifier on this data: $h_2 = \boldsymbol{Learn}(D)$
- …
- Pick the best classifier based on validation set

DAgger: Inner Details

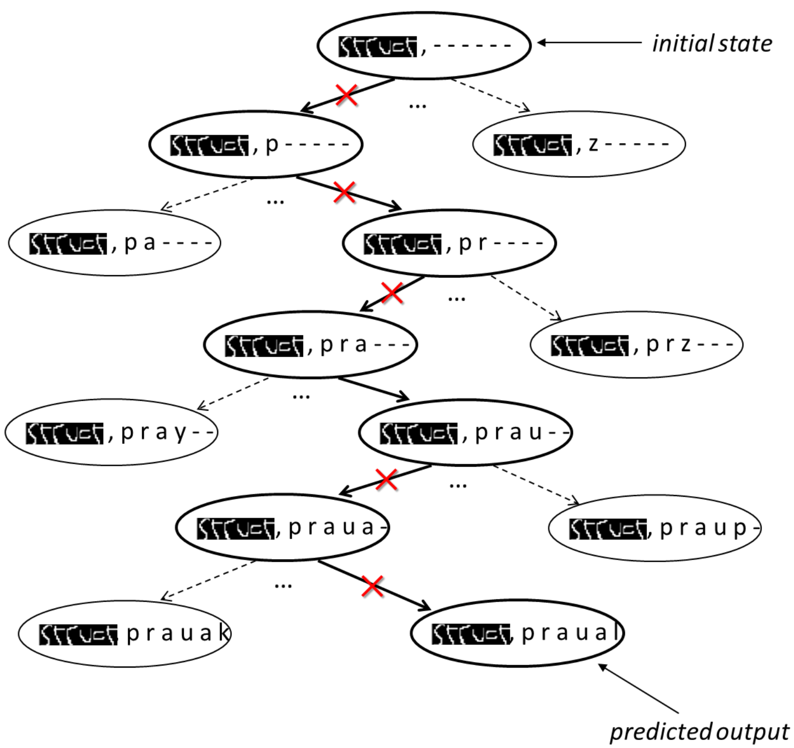
- **Inside each iteration, for one training example $(\boldsymbol{x}, \boldsymbol{y})$**

  - Compute the path traversed by the current policy: β times h_i (new classifier) + $1 - \beta$ times h^*(expert or Oracle classifier)



DAgger: Inner Details

- **Inside each iteration, for one training example (x, y)**
  - Generate additional examples to correct the errors



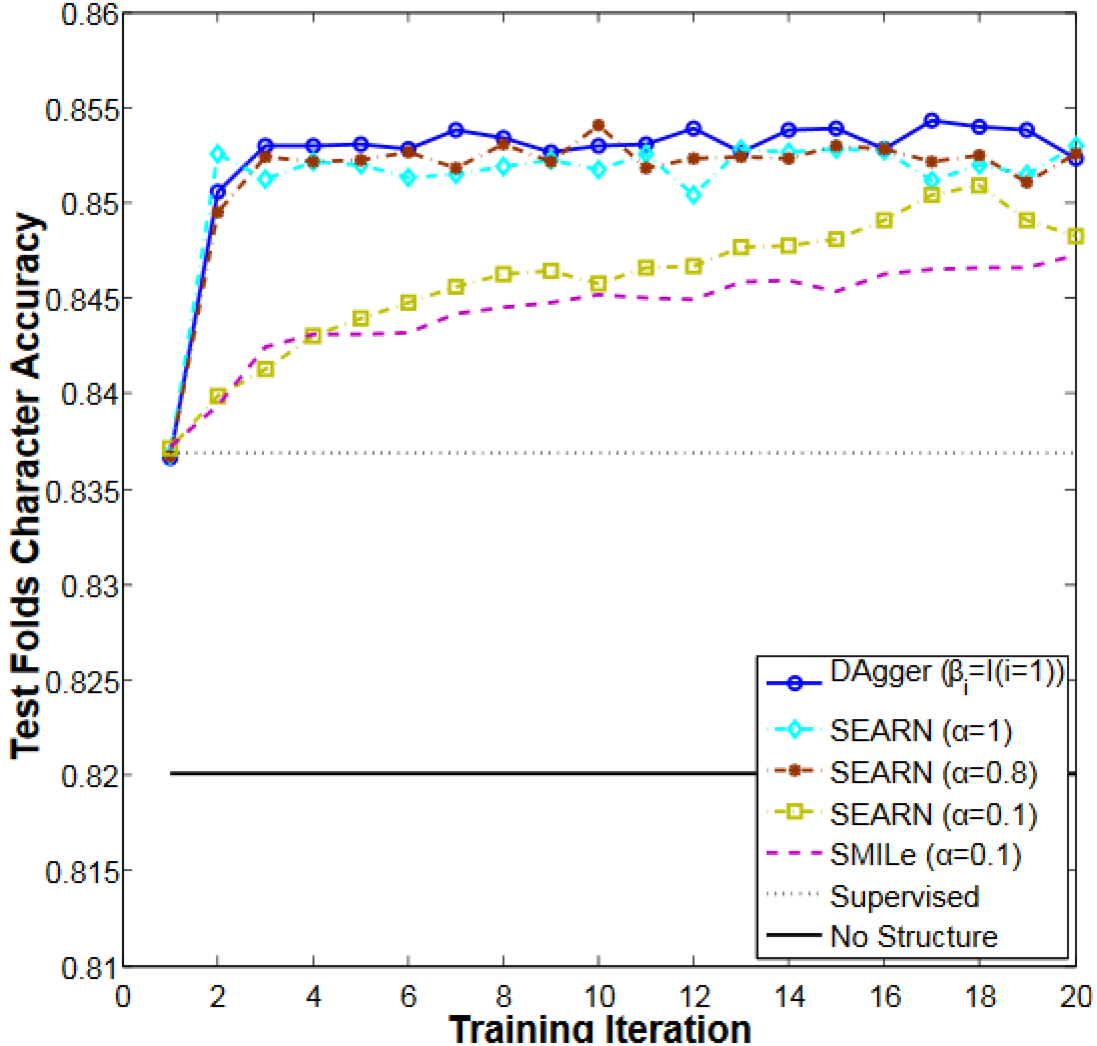
- Generate one classification example for each mistake on the greedy trajectory

DAgger: Inner Details

- At the end of each iteration i
  - Learn a classifier from the aggregate set of classification examples: $D = D_0 \cup D_1 \cup \cdots \cup D_i$
  - $h_{i+1} = LEARN(D)$

- At the end of all the iterations
  - Pick the best classifier among $h_1, h_2, \ldots, h_T$ based on the validation set
  - We get a deterministic classifier!

DAgger for Handwriting Recognition



- Source: [Ross et al., 2011]

DAgger: Discussion

- **Drawback**
  - Asks too many queries to the expert or oracle classifier
  - Very expensive for some tasks (e.g., training robot controllers)

- **Some ways to overcome this issue**
  - Active learning: ask queries selectively based on their usefulness in improving the performance of learner
  - Kshitij Judah, Alan Fern, Thomas G. Dietterich, Prasad Tadepalli: Active Imitation learning: formal and practical reductions to I.I.D. learning. Journal of Machine Learning Research 15(1): 3925-3963 (2014)

AggreVaTe (Ross and Bagnell, 2014)

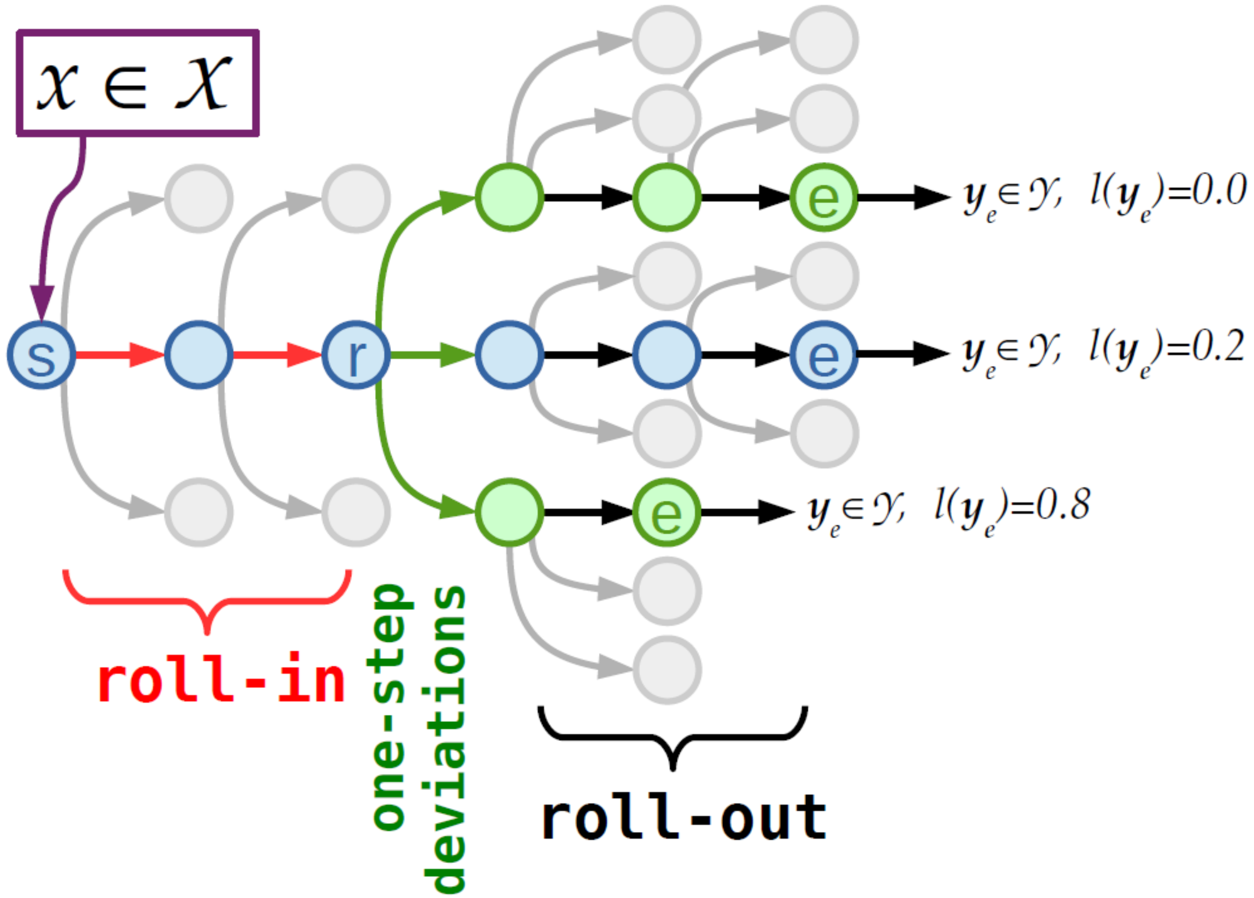
- ArXiv paper: rejected from NIPS 2014, currently under review at ICML 2016
  - http://arxiv.org/pdf/1406.5979v1.pdf
- **AggreVaTe:** Aggregate Values to Imitate
- **Key Idea:**
  - Cost-sensitive classification examples inside DAgger
  - Costs are generated by performing rollout with the expert (or reference) policy – similar to optimal approximation in SEARN
- NRPI (No Regret Policy Iteration) for Reinforcement Learning – adaptation of AggreVate algorithm

LOLS: Locally Optimal Learning to Search (Chang et al., ICML 2015)

- Imitation learning assumes a "very good" expert (reference or oracle) policy
- All the guarantees of the learned policy (predictor) is w.r.t the performance of the reference policy
- What if the reference policy is bad?
  - Can we learn a policy that improves over the reference policy?
  - Yes, the authors' provide LOLS algorithm

Acknowledgements: pictures related to LOLS from Chang et al., ICML 2015 talk

Roll-in vs. Roll-out Policies



- Roll-in choice dictates what states we train on
- Roll-out choice dictates how we score actions (costs)

Effect of Roll-in and Roll-out Policies

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

- **Roll-in with reference**
  - states trained on are not representative of those seen at prediction time – exact imitation training

Effect of Roll-in and Roll-out Policies

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

- **Roll-out with reference**
  - causes learned policy to fail to converge to a local optima if the reference policy is suboptimal
  - inaccurate comparison of policies

Effect of Roll-in and Roll-out Policies

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

- **Roll-in and Roll-out with learned policy**
  - Ignores reference policy and is equivalent to reinforcement learning

Effect of Roll-in and Roll-out Policies

roll-out $\rightarrow$ $\downarrow$ roll-in	Reference	Half/Half	Learned
Reference	Inconsistent		
Learned	No local opt	Good	RL

- **LOLS**
  - roll-in with learned and roll-out with a mixture of learned and reference policies
  - if reference is optimal, locally optimal
  - if reference is sub-optimal, either locally optimal or better than reference

Summary: Classifier-based SP

- Structured prediction as sequential decision-making task
  - training data is the oracle or expert demonstration
- In this view, structured prediction, imitation learning, and reinforcement learning are related
- Imitation learning algorithms: exact imitation, Forward Training, SEARN, DAgger, AggreVaTe, NRPI, LOLS
- Reductions to simple (cost-sensitive) classification – allows use to leverage existing algorithms and software

References

- Forward Training; Stochastic Mixing and Learning
  - http://www.cs.cmu.edu/~sross1/publications/Ross-AIStats10-paper.pdf
- SEARN
  - http://hunch.net/~jl/projects/reductions/searn/searn.pdf
- DAgger
  - http://www.cs.cmu.edu/~sross1/publications/Ross-AIStats11-NoRegret.pdf