Imitation Learning

Richard Zhu, Andrew Kang

April 26, 2016

Table of Contents

Introduction

Where we've been

The journey so far:

- Online learning
  - Follow-the-Leader, Perceptron
  - Multi-armed Bandits (UCB1, LinUCB)
- Reinforcement learning
  - MCTS + deep NN $\rightarrow$ AlphaGo
  - Deep Q-learning $\rightarrow$ DeepMind Atari
  - Apprenticeship $\rightarrow$ Autonomous helicopter flight

Trending towards generality (MAB a special case of MDP in reinforcement learning) and more efficient exploration (esp. with experts).

Takeaways

Simple iterative procedures can yield great performance.

In particular, utilizing expert demonstrations to learn a policy can do very well, if done in a clever way.

Where we'll go

The **sequential prediction** problem:

- States not drawn i.i.d. from true distribution!
- Future states depend on previous states / actions made.
- Breaks supervised learning (even expert demos fail).

Examples:

- Legged locomotion [Ratliff 2006]
- Outdoor navigation [Silver 2008]
- Car driving [Pomerleau 1989]
- Helicopter flight [Abbeel 2007] (last Thursday's talk!)

Where we'll go

Where we'll go

Automation of these processes are challenging control problems

- high-dimensionality
- non-linearity of dynamics
- stochasticity

Reduce to sequence prediction problems, solved using imitation learning.

We'll examine the current state-of-the-art in imitation learning: DAGGER, a meta-algorithm which learns a *stationary* & *deterministic* policy which is guaranteed to perform "well" (exact meaning TBA) under its induced distribution of states.

Preliminaries

High-level view of imitation learning

Imitation learning approaches the sequential prediction problem by seeking clever ways of using expert demonstrations (training dataset) to learn a policy that makes predictions (actions) that approximate expert actions.

In general, we interpret problems in imitation learning as follows:

- A **learner** makes **actions** (or predictions).
- These actions influence the current environment **state**.
- The environment returns **reward** to learner.

This sounds pretty similar to apprenticeship learning! What's *is* the difference?

Imitation v. apprenticeship learning

Typical approaches in reinforcement/apprenticeship learning seek to balance explore/exploitation tradeoff.

Exploration is hard, computationally expensive!

- MCTS
- UCB1, LinUCB
- MDP - Q-learning, policy gradient, apprenticeship

Imitation v. apprenticeship learning

Apprenticeship learning (autonomous helicopter):

- Apprenticeship trades exploration completely for exploitative/greedy policy - difficult to generalize.
- Hand-tweaking required to accomplish difficult maneuvers.
- Recovering from errors/unexplored states in state space impossible.

What if we're in new situations e.g. self-driving car? A more stable and more reliable framework is desired.

Notation

- Policies: $\pi \in \Pi$, Π denotes class of policies considered by learner
- Task horizon: T
- Distribution of states at time t given that π was executed from time 1 to $t-1$: d_π^t
- Average distribution of states if we follow π for T steps: $d_\pi = \frac{1}{T}\sum_{t=1}^{T} d_\pi^t$
- Expected immediate task cost of performing action a in state s: $C(s, a)$
- Expected immediate policy cost of π in s (assume C bounded in $[0, 1]$): $C_\pi(s) = \mathbb{E}_{a\sim\pi(s)}[C(s, a)]$

Notation

- Total cost (cost-to-go) $J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s \sim d_\pi^t}[C_\pi(s)] = T\mathbb{E}_{s \sim d_\pi}[C_\pi(s)]$
  - May not necessarily know true costs $C(s, a)$. Instead, observe expert demonstrations π^* and seek to bound $J(\pi)$ with surrogate loss based on how well π mimics, or "imitates" the expert policy π^*.
  - We do this because π^* is not necessarily in our policy class Π under consideration.
- Observed surrogate loss $\ell(s, \pi)$
  - Possibilities: expected 0-1 loss of π w.r.t. π^* in state s, squared/hinge loss of π w.r.t. π^* in s.
  - Should become more clear as we proceed.

Example situation: learning to drive from demonstrations

Input:

Camera Image

Policy

Output:

Steering in [-1,1]

Hard left turn

Hard right turn

Learning goal

Objective: Find a policy $\hat{\pi}$ which minimizes the observed surrogate loss ℓ under its induced distribution of states.

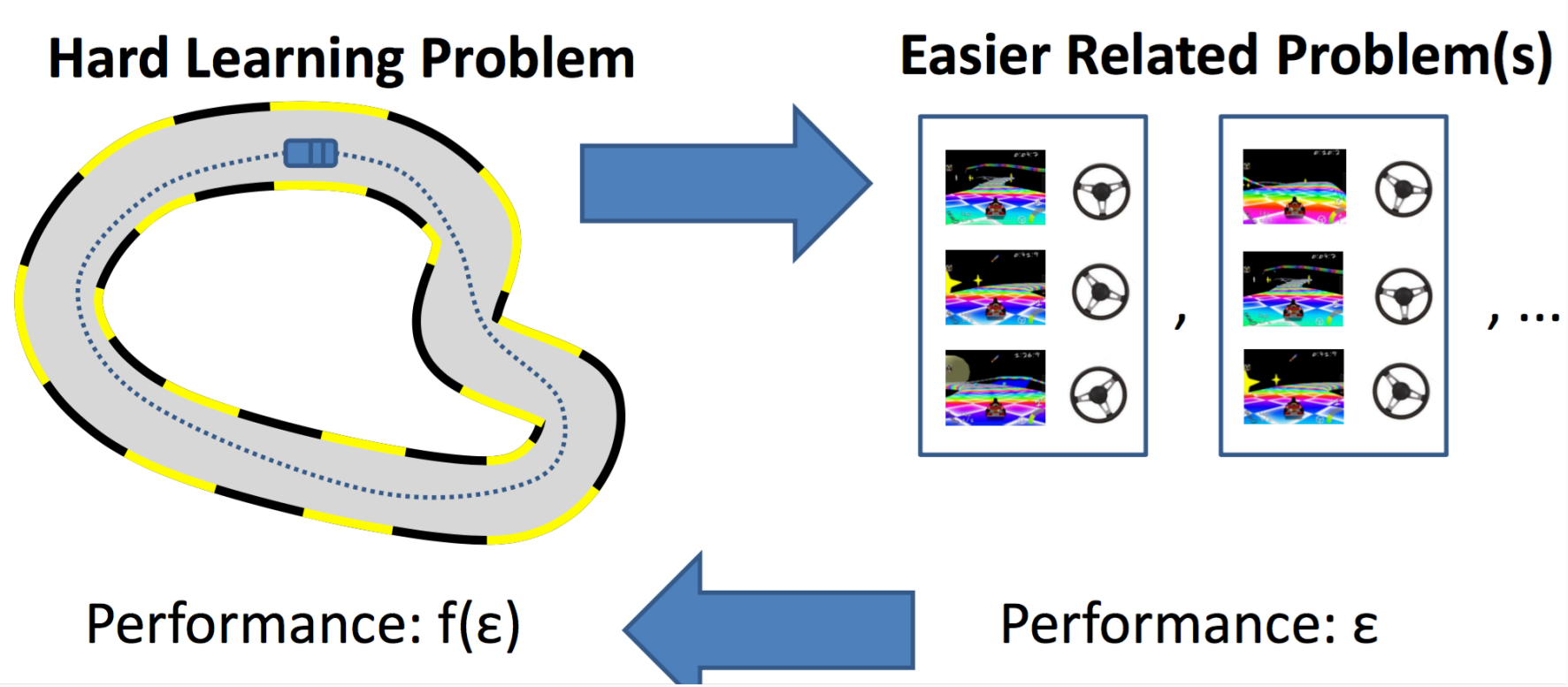
$$\hat{\pi} = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi} [\ell(s, \pi)]$$

In the case of apprenticeship learning for the helicopter problem, the dynamics and progression from state to state (by necessity) were also part of the learning process, i.e. explicitly modeled. In order to maintain generality, we assume unknown and complex system dynamics.

Ultimately, this is a non-i.i.d. supervised learning problem (d_π depends on π). Optimization is difficult, since the influence of policy on future states makes the objective non-convex (even if $\ell(s, \cdot)$ is convex in π for all s).

Reduction-based approach/analysis

For generality, employ reductions. Abstract away the learner algorithm, allows leveraging of any supervised learning algorithm!



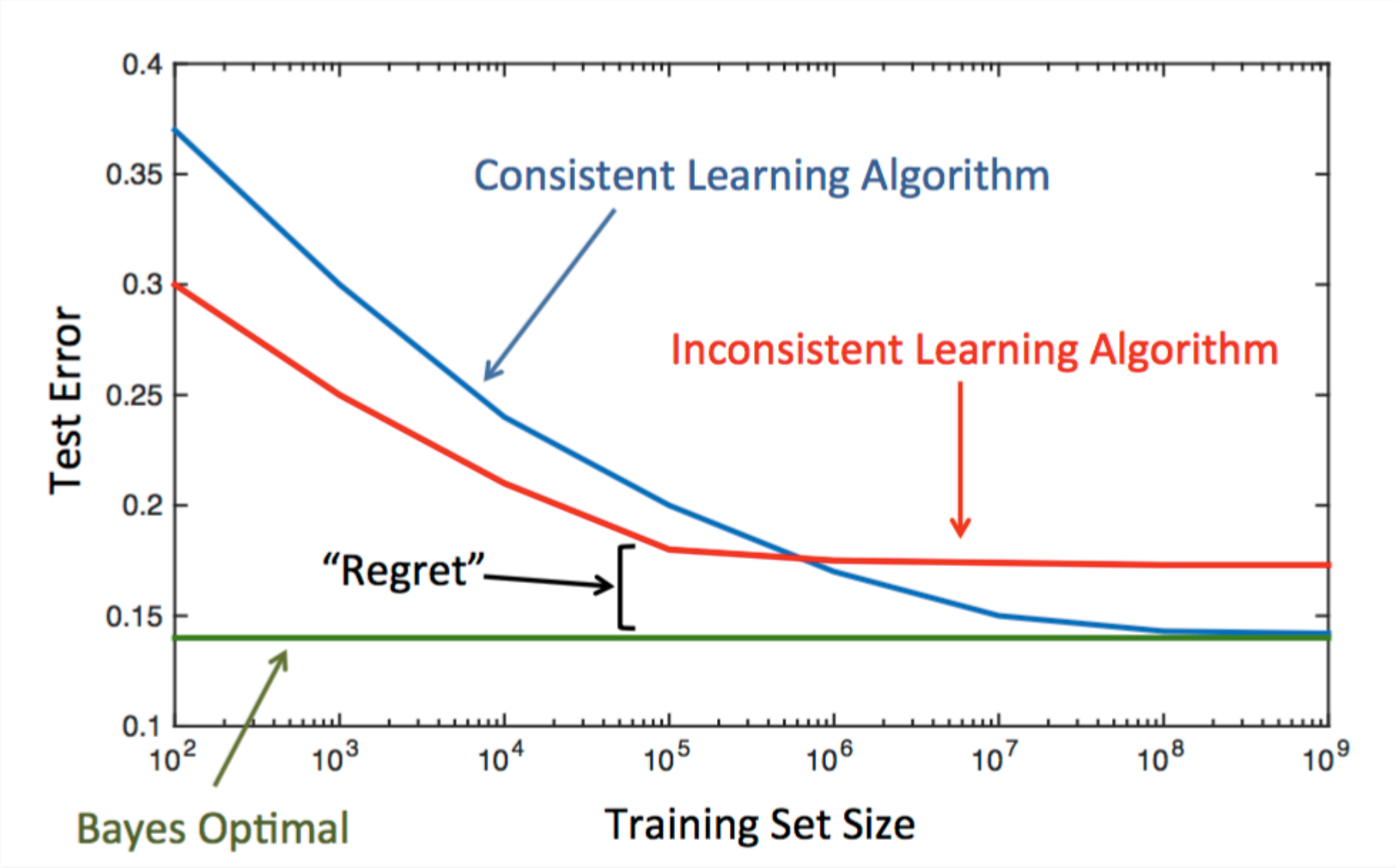
Previous approaches

We begin by reviewing some previous approaches, along with their performance guarantees. Overview of algorithms (spoilers!):

- Standard supervised learning
  - $O(T^2\epsilon)$ regret, terribly impractical
- Forward Training [Ross and Bagnell 2010]
  - $O(uT\epsilon)$ (near-linear) regret
  - Impractical - maintains T non-stationary policies
- Stochastic Mixing Iterative Learning (SMILe) [Ross and Bagnell 2010]
  - $O(uT\epsilon)$ (near-linear) regret
  - Impractical - adds a new policy to mixture at each iteration; policy is stochastic and thus unstable
- Dataset Aggregation (DAgger) [Ross et al. 2011]
  - Stationary (mostly) deterministic policy (stable finite mixture of policies)
  - No regret (in suitable conditions)

These bounds all assume that the base learner has ϵ error rate.

Consistent / no-regret algorithms



Supervised Learning

The naive approach at solving the learning objective is simply to train a policy π over the distribution of states encountered by the expert, d_{π^*}.

Reducing the "hard" imitation learning problem to many decoupled supervised learning problems, we may employ any standard supervised learning algorithm:

$$\hat{\pi}_{sup} = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim d_{\pi^*}} [\ell(s, \pi)]$$

Note that our actual objective is to find $\hat{\pi}$:

$$\hat{\pi} = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim d_{\pi}} [\ell(s, \pi)]$$

A small difference, right? Obviously minor...

Supervised Learning Performance

Assuming ℓ is the 0-1 loss, we get the following performance guarantee:

Theorem

Let $\mathbb{E}_{s \sim d^*_\pi}[\ell(s, \pi)] = \epsilon$, *then* $J(\pi) \leq J(\pi^*) + T^2\epsilon$.

Details of proof may be found in references, similar approach demonstrated later.

Intuition: as soon as $\hat{\pi}$ makes a mistake ($T\epsilon$ factor), it might end up in new states not visited by expert policy π^*, always incurring maximal 0-1 loss of 1 at each step from then on (T factor).

Prefer approaches with performance closer to $O(T\epsilon)$ or "near"-linear $O(uT\epsilon)$.

What is u? Max difference between cost of worst action and optimal action.

Supervised Learning Summary

Loss grows superlinearly in time horizon: $O(T^2\epsilon)$.

Intuition: naive supervised learning fails to capture the change in states, and is unable to generalize to unseen situations. If the algorithm makes an error, it does not know how to get back to states seen by the expert.

State distribution mismatch between training distribution and testing distribution. Errors cascade!

Forward Training

The **forward training** algorithm (Ross and Bagnell 2010) trains a non-stationary policy (one policy π_t at each timestep t) iteratively over T iterations.

At each iteration t, π_t is trained to mimic π^* on the distribution of states induced by the previously trained policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$.

Rectifies fundamental errors in naive supervised learning approach!

Forward Training Intuition

In iteration i (up to T iterations), correct π_i^i in the final policy $\pi^i = \{\pi_1^i, \ldots, \pi_T^i\}$.

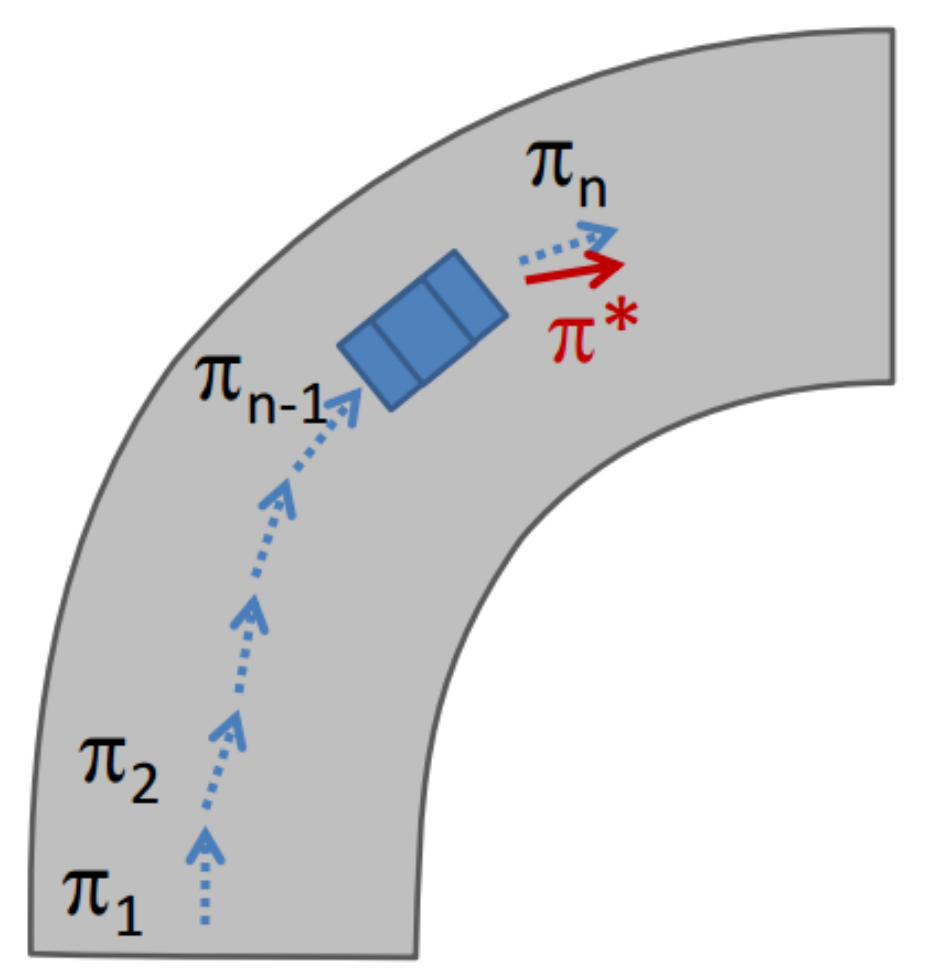
- Generate trajectories by following $\pi^{i-1} = \{\pi_1^{i-1}, \ldots, \pi_T^{i-1}\}$.
- $\mathcal{D} = \{(s_i, \pi^*(s_i)\}$ - states + actions taken by expert at step i
- Train a learner π_i^i on objective

$$\pi_i^i = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim \mathcal{D}}(e_\pi(s))$$

- Keep all other policies constant.

Iterative training on state distribution after each round, has good performance!

Forward Training Algorithm



Forward Training Algorithm

Initialize $\pi_1^0, \ldots, \pi_T^0$ to query and execute π^*.
for $i = 1$ **to** T **do**
  Sample T-step trajectories by following π^{i-1}.
  Get dataset $\mathcal{D} = \{(s_i, \pi^*(s_i))\}$ of states, actions taken by expert at step i.
  Train classifier $\pi_i^i = \text{argmin}_{\pi \in \Pi} \mathbb{E}_{s \sim \mathcal{D}}(e_\pi(s))$.
  $\pi_j^i = \pi_j^{i-1}$ for all $j \neq i$
end for
Return $\pi_1^T, \ldots, \pi_T^T$

Forward Training Performance

Let $Q_t^{\pi'}(s, \pi)$ denote the cost of following π in initial state s and then following π' at timestep t. Further, assume $\ell(s, \pi)$ is 0-1 loss (or upper bound on 0-1 loss). With task cost function $C \in [0, 1]$, we are guaranteed performance by the following theorem:

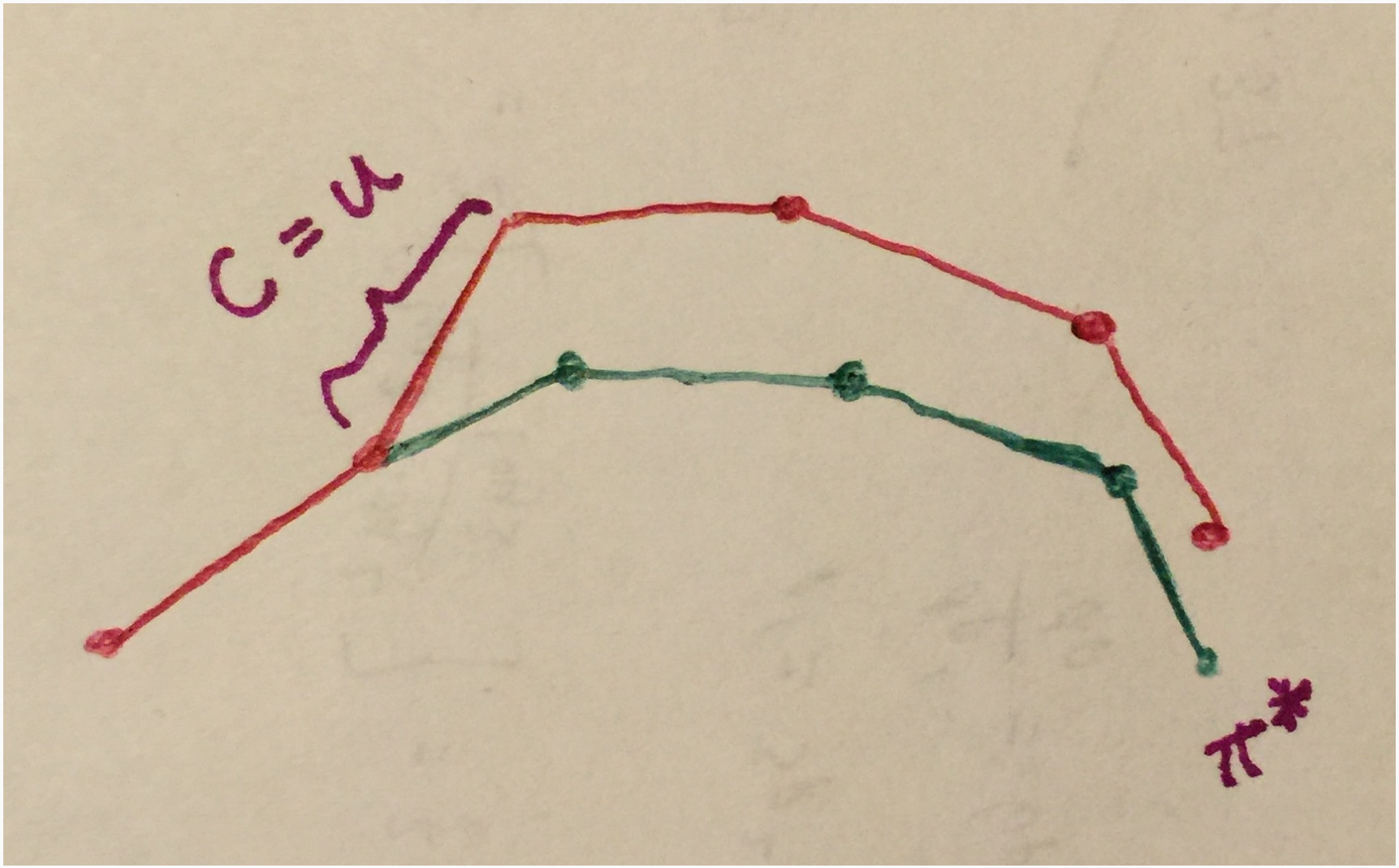
Theorem

Let π be such that $\mathbb{E}_{s \sim d_\pi}[\ell(s, \pi)] = \epsilon$ and $Q^{\pi^*}_{T-t+1}(s, a) - Q^{\pi^*}_{T-t+1}(s, \pi^*) \leq u$ for all actions a and $t \in \{1, 2, \ldots, T\}$ with $d_\pi^t(s) > 0$. Then

$$J(\pi) \leq J(\pi^*) + uT\epsilon$$

Near-linear regret! $O(uT\epsilon)$. What is u?

Forward Training Performance



Forward Training Performance

Proof. Let $\pi_{1:t}$ be the policy which executes π in the first t-steps, then executes π^*. We then have

$$
\begin{aligned}
J(\pi) &= J(\pi^*) + \sum_{t=0}^{T-1} [J(\pi_{1:T-t}) - J(\pi_{1:T-t-1})] \quad \text{telescoping sum} \\
&= J(\pi^*) + \sum_{t=1}^{T} \mathbb{E}_{s \sim d_\pi^t} [Q_{T-t+1}^{\pi^*}(s, \pi) - Q_{T-t+1}^{\pi^*}(s, \pi^*)] \quad \text{definition} \\
&\leq J(\pi^*) + u \sum_{t=1}^{T} \mathbb{E}_{s \sim d_\pi^t} [\ell(s, \pi)] \quad \ell \text{ upper bounds 0-1 loss} \\
&= J(\pi^*) + uT\epsilon
\end{aligned}
$$

Forward Training Performance

Worst case, $u \sim O(T)$ and forward algorithm provides no improvement over naive supervised learning.

Best case (often), $u \sim O(1)$ or u sublinear in T.

In particular, if π^* (expert) can recover from mistakes made by π, then $u \sim O(1)$. What does recover mean? Within few steps, $d_\pi^t \to d_{\pi^*}^{t+\Delta t}$.

Forward Training Summary

Pros:

- Recovery from mistakes.
- Trained on the distribution of states actually encountered.

Cons:

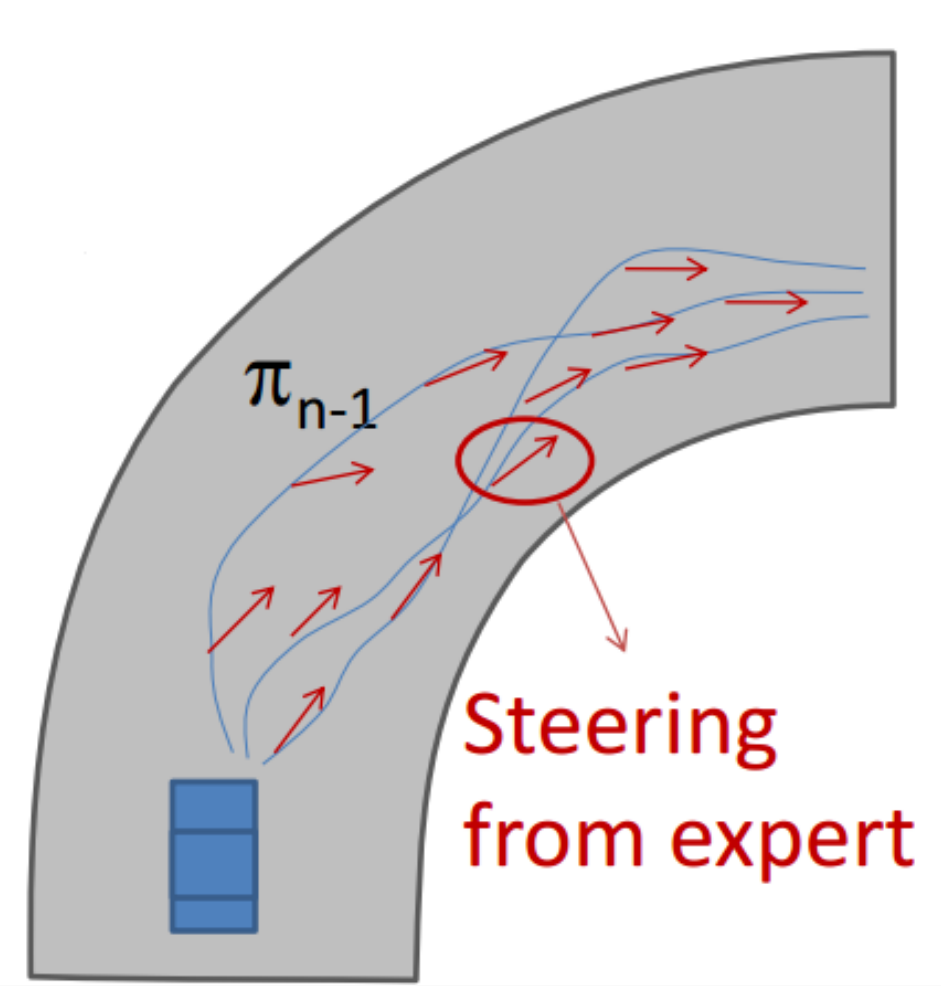
- Requires T iterations (what if undefined?) For large T, lots of iterations and policies to maintain. Cannot stop before all T completed.
- Non-stationary (not time-invariant) policy.
- Impractical in real-world.

SMILe

Stochastic Mixing Iterative Learning (SMILe), also proposed by Ross and Bagnell (2010), succeeds where Forward Training fails.

It can be applied when T is large or undefined by adopting an approach similar to SEARN (Daumé III et al., 2009) where a stochastic stationary policy is trained over several iterations.

SMILe Algorithm



SMILe Intuition

High-level understanding: “geometric” stochastic mixing of policies trained.

- Start with a policy π_0 which mimics exactly the expert’s actions.
- At each iteration n, train policy $\hat{\pi}_n$ to mimic the expert under the trajectories induced by π_{n-1}.
- Add trained policy to previous mix of policies with a geometric discount factor $\alpha(1-\alpha)^{n-1}$.

Result: π_n then is a mix of n policies, with the probability of using the expert’s action as $(1-\alpha)^n$.

Selecting $\alpha = O(\frac{1}{T^2})$ and $N = O(T^2 \log T)$ guarantees near-linear regret in T, ϵ for some classes of problems.

DAgger

DAgger

DAgger (Dataset Aggregation): iterative algorithm that trains a deterministic policy that achieves good performance guarantees under its induced distribution of states.

Unlike the previous algorithms, DAggertrains a single non-stochastic policy and remembers experience of the expert and previous policies by aggregating data.

Overview

High level overview:

- Uses expert policy to gather dataset of trajectories D and train a policy that best mimics the expert on those trajectories.
- Iteratively uses previous policy to collect more trajectories and add them to D.
- Next policy is the policy that best mimics the expert on whole D.

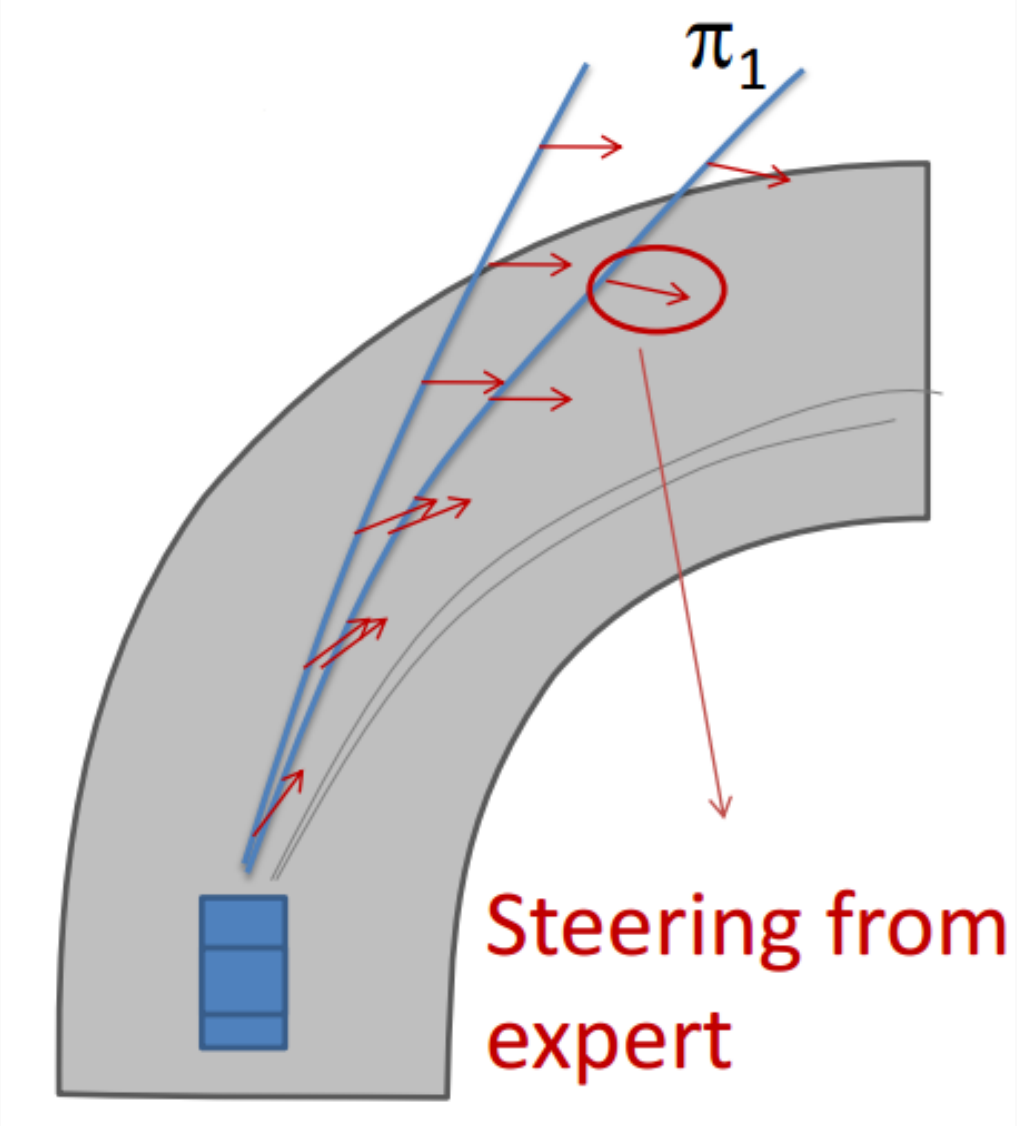
Overview

In other words, at each iteration n: collect dataset with $\hat{\pi}_n$ and then train $\hat{\pi}_{n+1}$ with dataset aggregate.

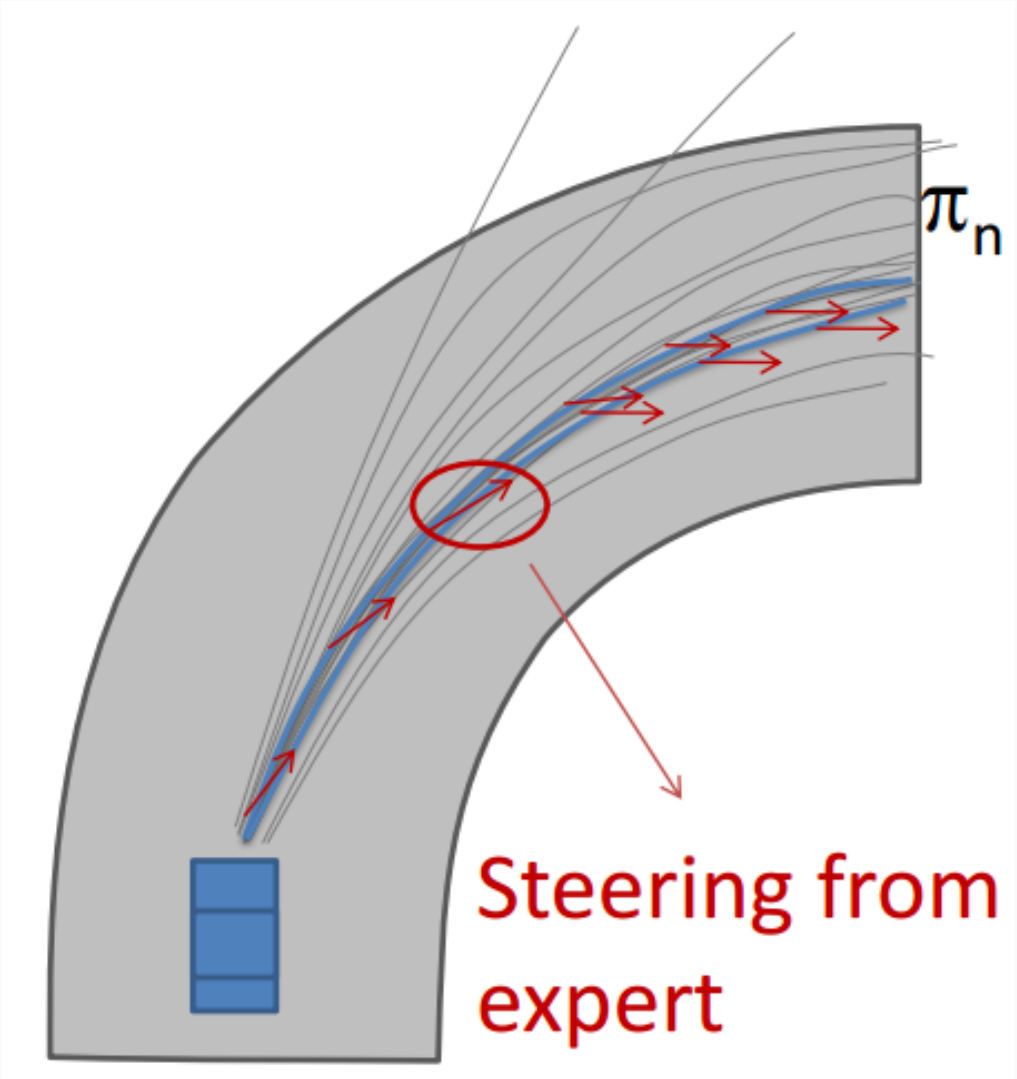
Intuition: build a set of inputs that the final policy is likely to encounter based on previous experience.

Can be interpreted as a Follow-The-Leader algorithm, since it chooses the best next policy in hindsight.

Example



Example



Overview

Optional: at iteration i, use $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$, i.e. stochastic sampling.

Imagine flipping a coin, weighted with probability β_i, to choose whether to use π^* (expert) or $\hat{\pi}_i$ (learned) to determine the next action.

Intuition: first few policies may make many more mistakes as they are trained on smaller datasets and visit states that are irrelevant as the policy improves.

Stochastic Sampling

Typically:

1. $\beta_1 = 1 \rightarrow \hat{\pi}_1 = \pi^*$, i.e. no initial policy needs to be specified
2. $\beta_i = p^{i-1}$, i.e. exponential decay

Only requirement for guarantees is that $\bar{\beta}_N = \frac{1}{N} \sum_{i=1}^{N} \beta_i \rightarrow 0$ as $N \rightarrow \infty$.

Algorithm

Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
  Sample T-step trajectories using π_i.
  Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by π_i and actions given by expert.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
  Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.

Guarantees

Notation

Notation:

- Sequence of policies: $\pi_{1:N} = \pi_1, \ldots, \pi_N$
- True loss of best policy in hindight:
  $\epsilon_N = \min_{\pi \in \Pi} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{s \sim d_{\pi_i}}[\ell(s, \pi)]$
- Regret: γ_N s.t.:

$$\frac{1}{N} \sum_{i=1}^{N} \ell_i(\pi_i) - \min_{\pi \in \Pi} \frac{1}{N} \sum_{i=1}^{N} \ell_i(\pi) \leq \gamma_N$$

In particular, the algorithm is no regret if $\lim_{N \to \infty} \gamma_N = 0$.

Guarantees

Assumptions:

- ℓ is strongly convex and bounded over Π
- $\beta_i \leq (1-\alpha)^{i-1}\ \forall i$ for some constant α independent of T

The following holds in the infinite sample case:

Theorem

For DAGGER, if N is $\tilde{O}(T)$ there exists a policy $\hat{\pi} \in \hat{\pi}_{1:N}$ s.t. $\mathbb{E}_{s \sim d_{\hat{\pi}}}[\ell(s, \hat{\pi})] \leq \epsilon_N + O(1/T)$.

Guarantees

Note:

- Must hold for best policy in hindsight
- If ℓ upper bounds C, then $J(\hat{\pi}) \leq T\epsilon_N + O(1)$

Then similarly to theorem from Forward Training:

Theorem

For DAGGER, if N is $\tilde{O}(uT)$ there exists a policy $\hat{\pi} \in \hat{\pi}_{1:N}$ s.t. $J(\hat{\pi}) \leq J(\pi^*) + uT\epsilon_N + O(1)$.

No Regret Properties

- Relies solely on no regret property of underlying Follow-The-Leader algorithm on strong convex losses which picks the sequence of policies $\hat{\pi}_{1:N}$.
- By reduction, the following results hold for *any* other no regret online learning algorithm!
- No regret algorithms can be useful for finding a policy which has good performance guarantees under its own distribution of states if we choose loss functions to be loss under distribution of states of current policy $\ell_i(\pi) = \mathbb{E}_{s \sim d_{\pi_i}}[\ell(s, \pi)]$.

Lemma

Total variation distance: We first bound the difference between distributions encountered by $\hat{\pi}_i$ and π_i:

Lemma

$\|d_{\pi_i} - d_{\hat{\pi}_i}\|_1 \leq 2T\beta_i$

Lemma Proof

Proof: If d is the distribution of states over T steps where π_i picks π^* at least once, then $d_{\hat{\pi}_i} = (1-\beta_i)^T d_{\hat{\pi}_i} + (1-(1-\beta_i)^T)d$:

$$
\begin{aligned}
&||d_{\pi_i} - d_{\hat{\pi}_i}||_1 \\
&= (1-(1-\beta_i)^T)||d - d_{\hat{\pi}_i}||_1 \\
&\leq 2(1-(1-\beta_i)^T) \\
&\leq 2T\beta_i
\end{aligned}
$$

since $(1-\beta)^T \geq 1-\beta T$ for any $\beta \in [0,1]$.

(Only better than trivial bound $\|d_{\pi_i} - d_{\hat{\pi}_i}\|_1 \leq 2$ for $\beta_i \leq \frac{1}{T}$!)

Theorem

Assumptions:

- β_i is non-increasing
- n_β is the largest $n \leq N$ such that $\beta_n > \frac{1}{T}$
- $\ell_{\max}$ is an upper bound on loss, i.e. $\ell_i(s, \hat{\pi}_i) \leq \ell_{\max}$

We have the following:

Theorem

For DAGGER, there exists a policy $\hat{\pi} \in \hat{\pi}_{1:N}$ s.t.
$\mathbb{E}_{s \sim d_{\hat{\pi}}}[\ell(s, \hat{\pi})] \leq \epsilon_N + \gamma_N + \frac{2\ell_{\max}}{N}[n_\beta + T\sum_{i=n_\beta+1}^{N} \beta_i]$*, for γ_N the average regret of $\hat{\pi}_{1:N}$.*

Theorem Proof

By lemma: $\mathbb{E}_{s\sim d_{\hat{\pi}_i}}(\ell_i(s,\hat{\pi}_i)) \leq \mathbb{E}_{s\sim d_{\hat{\pi}_i}}(\ell_i(s,\hat{\pi}_i)) + 2\ell_{\max}\min(1, T\beta_i)$.

Thus:

$$
\begin{aligned}
&\min_{\hat{\pi}\in\hat{\pi}_{1:N}} \mathbb{E}_{s\sim d_{\hat{\pi}}}[\ell(s,\hat{\pi})] \\
&\leq \tfrac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{s\sim d_{\hat{\pi}_i}}(\ell(s,\hat{\pi}_i)) \\
&\leq \tfrac{1}{N}\sum_{i=1}^{N} [\mathbb{E}_{s\sim d_{\pi_i}}(\ell(s,\hat{\pi}_i)) + 2\ell_{\max}\min(1, T\beta_i)] \\
&\leq \gamma_N + \tfrac{2\ell_{\max}}{N}[n_\beta + T\sum_{i=n_\beta+1}^{N}\beta_i] + \min_{\pi\in\Pi}\tfrac{1}{N}\sum_{i=1}^{N}\ell_i(\pi) \\
&= \gamma_N + \epsilon_N + \tfrac{2\ell_{\max}}{N}[n_\beta + T\sum_{i=n_\beta+1}^{N}\beta_i]
\end{aligned}
$$

Generalization

Theorem

Finite sample case: The finite case is a sort of generalization, as we cannot make infinite samples in practice. Assume we sample some finite number of trajectories m at each iteration. Then using Azuma-Hoeffding's inequality:

Theorem

For DAGGER, if N is $O(T^2 \log(1/\delta))$ and m is $O(1)$ then with probability at least $1 - \delta$ there exists a policy $\hat{\pi} \in \hat{\pi}_{1:N}$ s.t.
$\mathbb{E}_{s \sim d_{\hat{\pi}}}[\ell(s, \phi)] \leq \hat{\epsilon}_N + O(1/T)$.

Theorem

More refined analysis assuming strongly convex loss:

Theorem

For DAGGER, if N is $O(u^2T^2log(1/\delta))$ and m is $O(1)$ then with probability at least $1-\delta$ there exists a policy $\hat{\pi} \in \hat{\pi}_{1:N}$ s.t.
$J(\hat{\pi}) \leq J(\pi^*) + uT\hat{\epsilon}_N + O(1)$.

Theorem

At each iteration i with dataset D_i:

- Online learner guarantees
  $\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{s\sim D_i}[l(s,\pi_i)] - \min_{\pi\in\Pi}\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{s\sim D_i}[l(s,\pi_i)] \leq \gamma_N$
- $\hat{\epsilon}_N = \min_{\pi\in\Pi}\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{s\sim D_i}[l(s,\pi_i)]$

We obtain:

Theorem

For DAGGER, with probability at least $1-\delta$, there exists a policy $\hat{\pi} \in \hat{\pi}_{1:N}$ s.t. $\mathbb{E}_{s\sim d_{\hat{\pi}}}[l(s,\hat{\pi})] \leq \hat{\epsilon}_N + \gamma_N + \frac{2\ell_{\max}}{N}[n_\beta + T\sum_{i=n_\beta+1}^{N}\beta_i] + \ell_{\max}\sqrt{\frac{2\log(1/\delta)}{mN}}$, for γ_N the average regret of $\hat{\pi}_{1:N}$.

Theorem Proof

Proof:

- Y_{ij} is the difference between expected per step loss of $\hat{\pi}_i$ under d_{π_i} and average per step loss $\hat{\pi}_i$ under j^{th} sample trajectory with π_i at iteration i
- Y_{ij} are all zero mean and bounded in $[-\ell_{\max}, \ell_{\max}]$ and form a martinagle
- By Azuma-Hoeffding's inequality
  $\frac{1}{mN}\sum_{i=1}^{N}\sum_{j=1}^{m} Y_{ij} \leq \ell_{\max}\sqrt{\frac{2\log(1/\delta)}{mN}}$

Theorem Proof

$$
\begin{aligned}
&\min_{\hat{\pi} \in \hat{\pi}_{1:N}} \mathbb{E}_{s \sim d_{\hat{\pi}}}[\ell(s, \hat{\pi})] \\
&\leq \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{s \sim d_{\hat{\pi}_i}}[\ell(s, \hat{\pi}_i)] \\
&\leq \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{s \sim d_{\pi_i}}[\ell(s, \hat{\pi}_i)] + \frac{2\ell_{\max}}{N}[n_\beta + T \sum_{i=n_\beta+1}^{N} \beta_i] \\
&= \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{s \sim D_i}[\ell(s, \hat{\pi}_i)] + \frac{1}{mN} \sum_{i=1}^{N} \sum_{j=1}^{m} Y_{ij} \\
&\quad + \frac{2\ell_{\max}}{N}[n_\beta + T \sum_{i=n_\beta+1}^{N} \beta_i] \\
&\leq \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{s \sim D_i}[\ell(s, \hat{\pi}_i)] + \ell_{\max} \sqrt{\frac{2 \log(1/\delta)}{mN}} \\
&\quad + \frac{2\ell_{\max}}{N}[n_\beta + T \sum_{i=n_\beta+1}^{N} \beta_i] \\
&\leq \hat{\epsilon}_N + \gamma_N + \ell_{\max} \sqrt{\frac{2 \log(1/\delta)}{mN}} + \frac{2\ell_{\max}}{N}[n_\beta + T \sum_{i=n_\beta+1}^{N} \beta_i]
\end{aligned}
$$

Performance

Problems

To test the algorithm, two imitation learning problems and one benchmark sequence labeling problem were used:

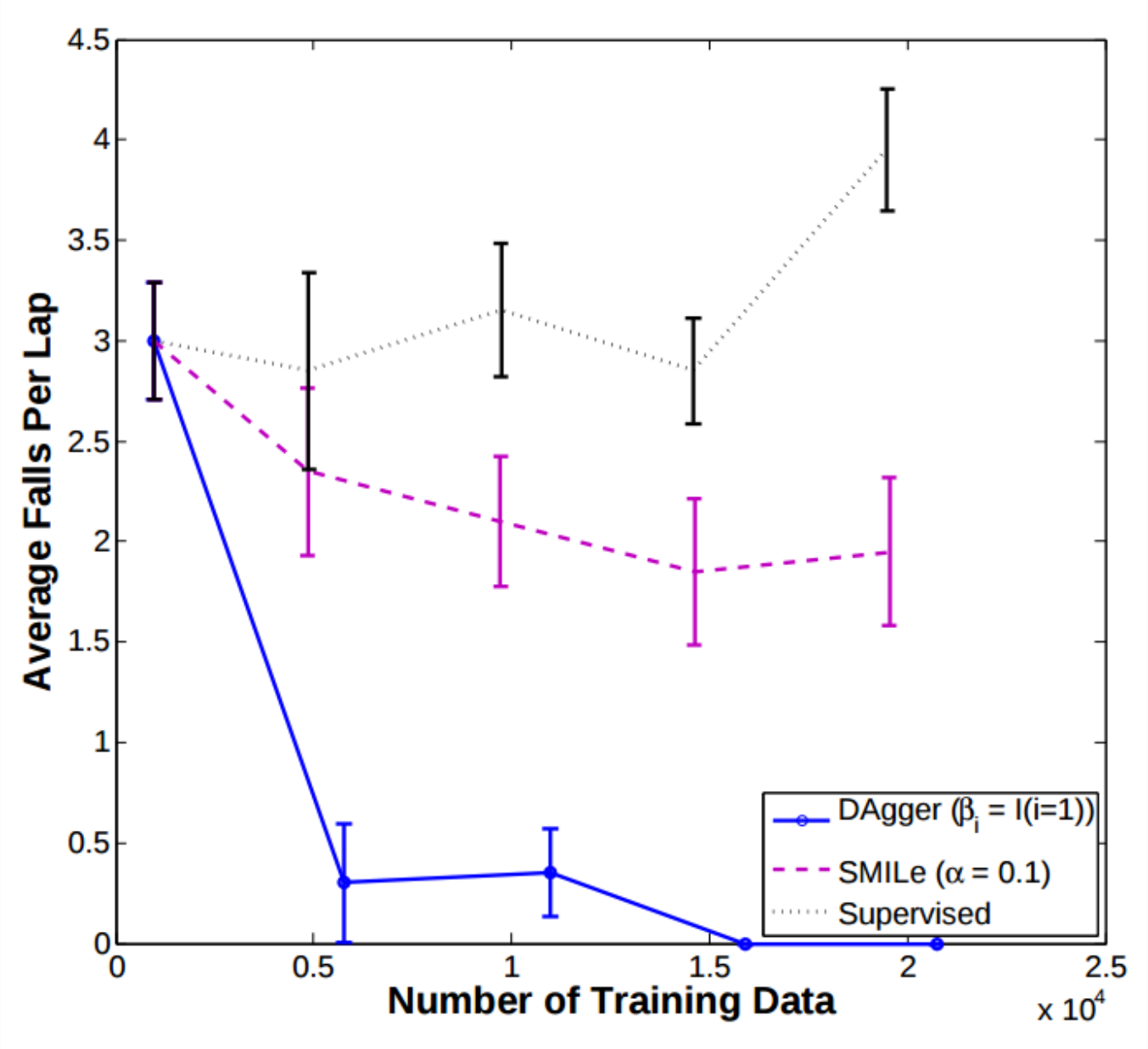
- Super Tux Kart
- Super Mario Bros.
- Handwriting Recognition

Experiment 1: Super Tux Kart

Super Tux Kart: 3D racing game similar to Mario Kart

- Input: Current game image
- Output: Analog joystick value in [-1, 1]
- Expert: Human feedback for each observed game image
- Base learner: Linear controller minimizing ridge regression
- Setting: “Star Track”: track floats in space (kart can fall off)
- Performance: Average number of falls per lap

Experiment 1: Super Tux Kart



Experiment 1: Super Tux Kart

Takeaway:

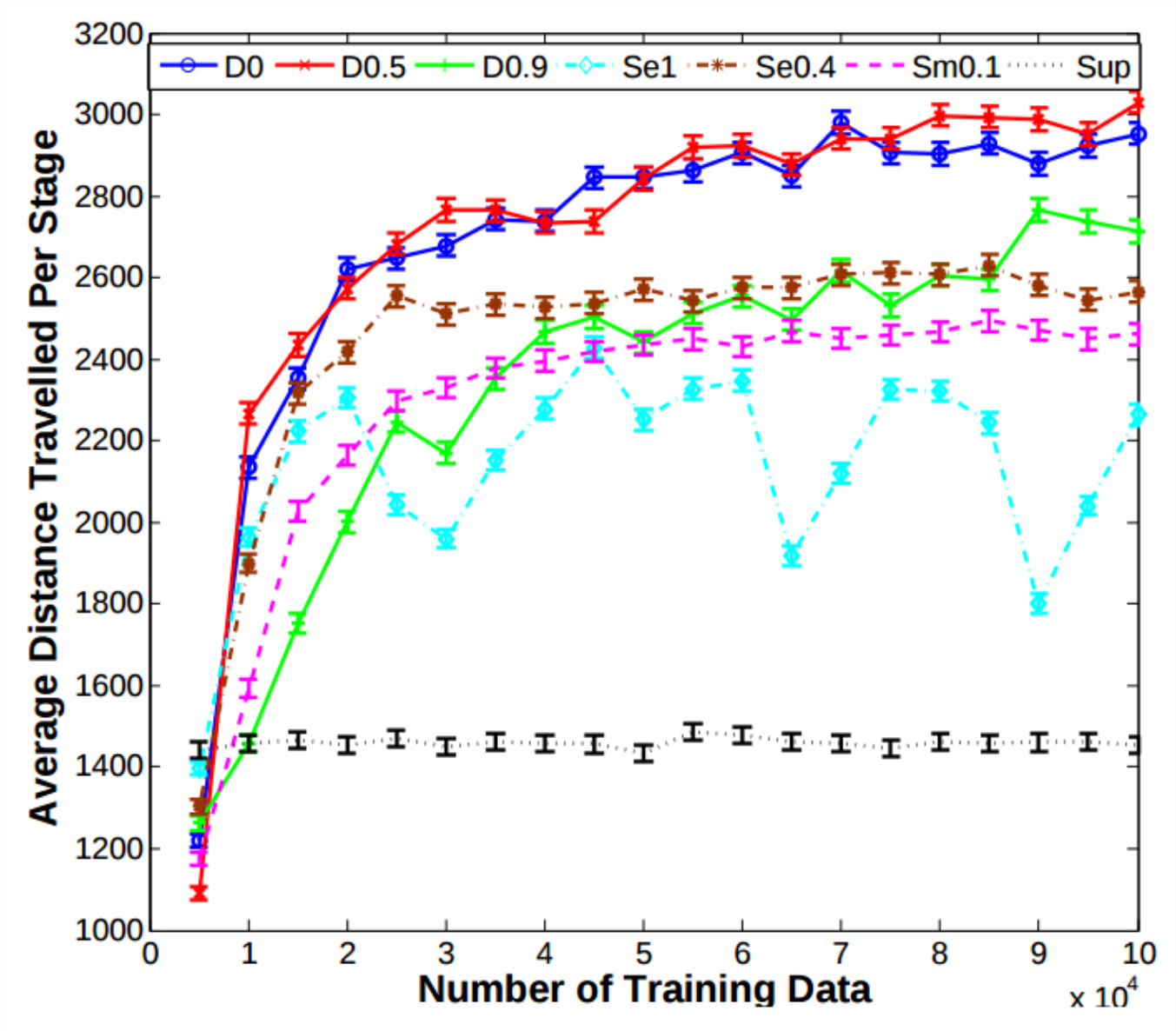
- Improvements as more data is collected
- Only a few iterations required, i.e. quadratic training time not a big problem
- Qualitatively better

Experiment 2: Super Mario Bros.

Super Mario Bros: Simulator from Mario Bros. AI competition which can randomly generate stages of varying difficulty

- Input: Current game image
- Output: Subset of {left, right, jump, speed}
- Expert: Planning algorithm with full access to internal state
- Base learner: 4 independent linear SVM
- Setting: Stage with difficulty 1 and time limit 60 seconds
- Performance: Average distance travelled

Experiment 2: Super Mario Bros.



Experiment 2: Super Mario Bros.

Takeaway:

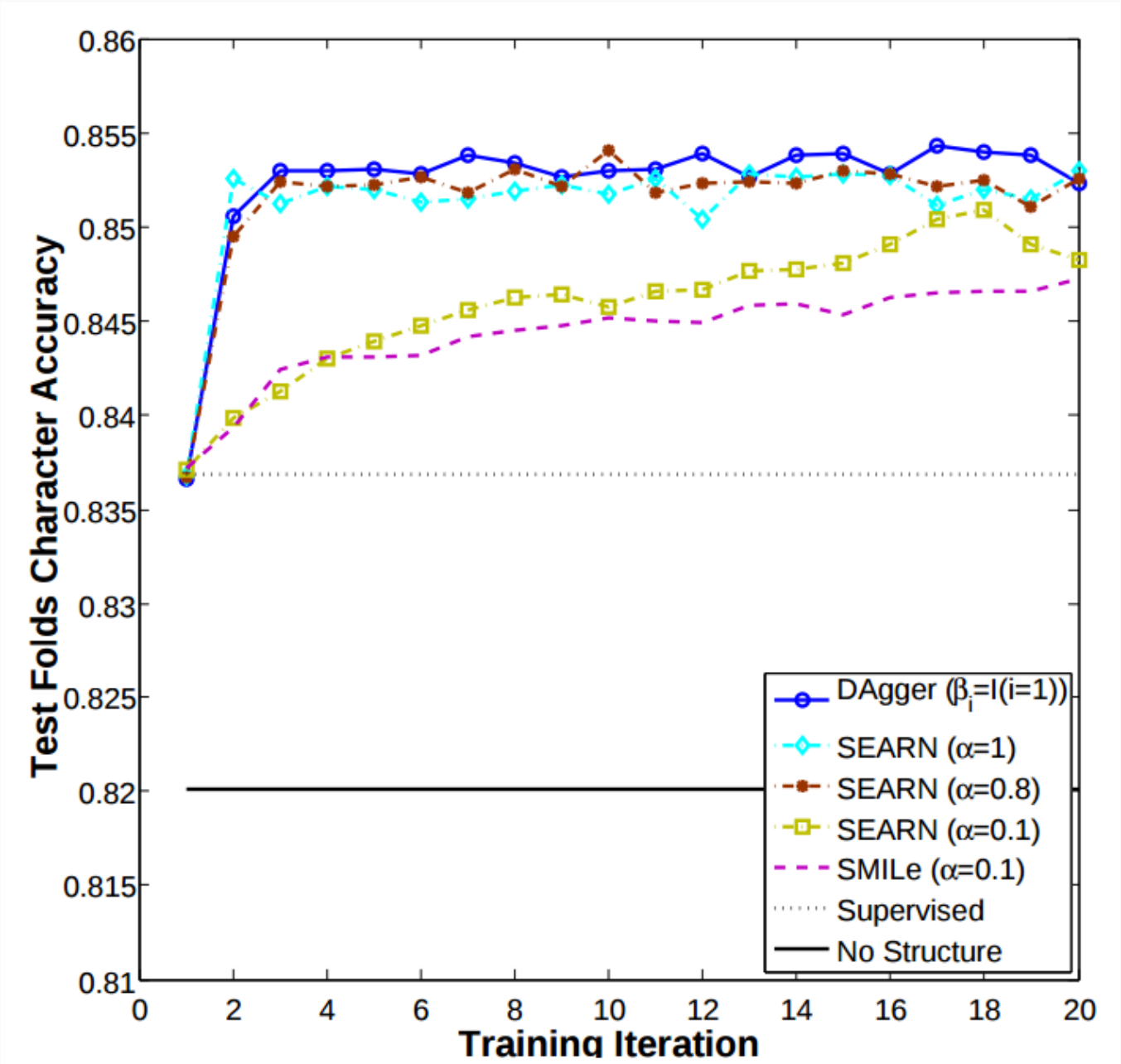
- Discrete actions
- Different stages
- Policy sampling
- Expert easily accessible (rare!)

Experiment 3: Handwriting Recognition

Handwriting Recognition: Parsing handwriting into words

- Input: Handwritten words, partitioned in 10 folds
- Output: Word predictions
- Expert: Tagged data
- Base learner: Linear SVM
- Setting: General handwriting dataset
- Performance: Character accuracy

Experiment 3: Handwriting Recognition



Experiment 3: Handwriting Recognition

Takeaway:

- DAGGER works for simple problems too

Video

Video party time! *cue video*

References I

S. Ross and J. A. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

S. Ross. Comparison of imitation learning approaches on Super Tux Kart, 2010a. URL http://www. youtube.com/watch?v=V00npNnWzSU.

S. Ross. Comparison of imitation learning approaches on Super Mario Bros, 2010b. URL http://www.youtube.com/watch?v=anOI0xZ3kGM.

S. Ross, G. J. Gordon, J. A. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. arXiv preprint. arXiv:1011:0686v3, 2011.

Questions?