COL778: Principles of Autonomous Systems
Semester II, 2023-24

Imitation Learning

Rohan Paul

Outline

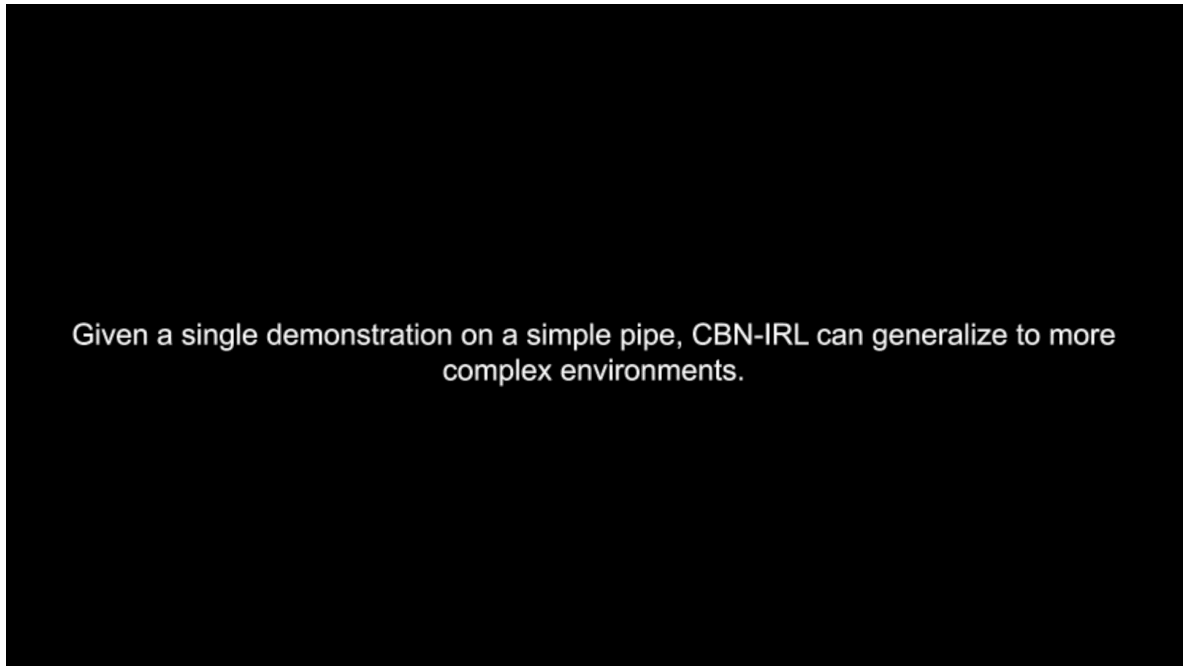
- Last Class
  - An introduction to Reinforcement Learning
- This Class
  - Imitation Learning
- Reference Material
  - Please follow the notes as the primary reference on this topic.

Acknowledgements

These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Nicholas Roy, Wolfram Burgard, Dieter Fox, Sebastian Thrun, Siddharth Srinivasa, Dan Klein, Pieter Abbeel, Max Likhachev and others.

Supervised Learning of Behaviors

- We learn many skills by observing an expert.
- Try to “imitate” the expert. Take the same action that the expert has taken.
- Next, we look at learning from demonstration.
- In practice, imitation is often combined with exploration to learn well.
- Imitation is really useful when?
  - The state space is too large for exploration-based RL to work.
  - Exploration is very expensive.



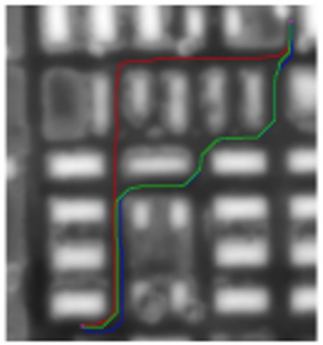
Imitation Learning Successes

Simulated highway driving

- Abbeel and Ng, ICML 2004
- Syed and Schapire, NIPS 2007
- Majumdar et al., RSS 2017

Aerial imagery-based navigation

- Ratliff, Bagnell, and Zinkevich, ICML 2006

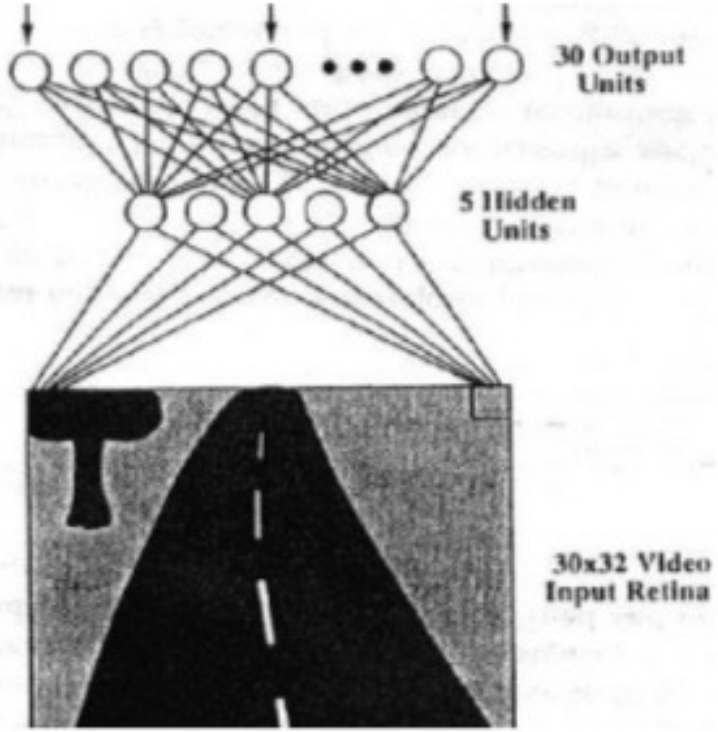


Parking lot navigation

- Abbeel, Dolgov, Ng, and Thrun, IROS 2008

Original (Deep) Imitation Learning System

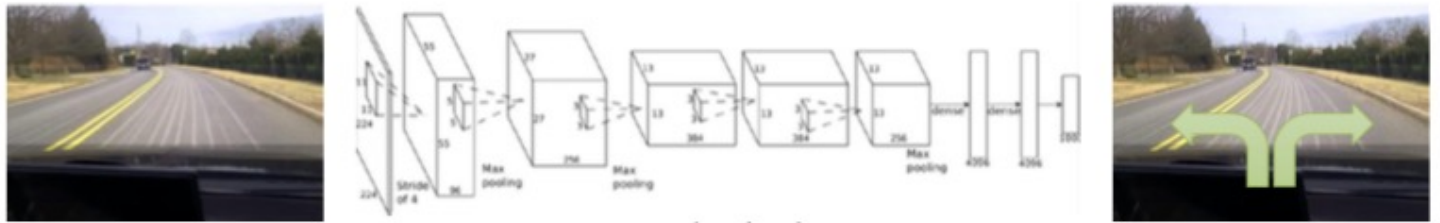
ALVINN: **A**utonomous **L**and **V**ehicle **I**n a **N**eural **N**etwork
1989



Example: ALVINN System

Imitation Learning Setup

Policy: a mapping from observations to actions



- Assume actions in the expert trajectories are i.i.d. (independent and identically distributed)
- Train a function to map observations/states to actions at each time step of the trajectory

Learning from Demonstration

- Expert provides a set of demonstration trajectories: sequences of states and actions
- Imitation learning is useful when is easier for the expert to demonstrate the desired behavior rather than:
  - come up with a reward that would generate such behavior,
  - coding up the desired policy directly

Notation

State: s (sometimes x) (**state may only be partially observed)

Action: a (sometimes y)

Policy: $\pi\theta$ (sometimes h)

- Policy maps states to actions: $\pi\theta(s) \rightarrow a$
- ...or distributions over actions: $\pi\theta(s) \rightarrow P(a)$

State Dynamics: P(s'|s,a)

- Typically not known to policy.
- Essentially the simulator/environment

Notation

Rollout: sequentially execute $\pi(s_0)$ on an initial state

- Produce trajectory $\boldsymbol{\tau}=(s_0,a_0,s_1,a_1,\ldots)$

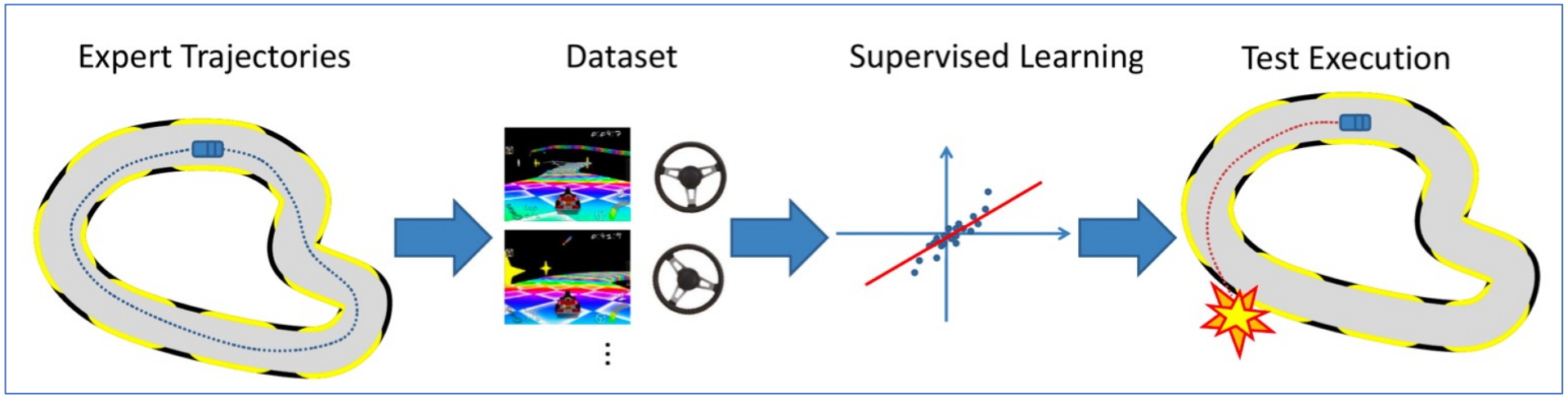
$P(\boldsymbol{\tau}|\pi)$: distribution of trajectories induced by a policy

1. Sample s_0 from P_0 (distribution over initial states), initialize t = 1.
2. Sample action a_i from $\pi(s_{t-1})$
3. Sample next state s_t from applying a_t to s_{t-1} (requires access to environment)
4. Repeat from Step 2 with t=t+1

$P(s|\pi)$: distribution of states induced by a policy

- Let $P_t(s|\pi)$ denote distribution over t-th state
- $P(s|\pi) = (1/T)\sum_t P_t(s|\pi)$

Behavior Cloning



- Simply mimic the teacher's actions.
- Reduction to supervised learning
  - Given the state action pairs from the demonstrator, learn to predict the same action as the expert
- Minimise the 1-step deviation
- Treat as a supervised learning problem.

Behavior Cloning

What we want

General Imitation Learning:

$$\underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{s\sim P(s|\pi_\theta)}\left[\mathcal{L}\left(\pi^*(s),\pi_\theta(s)\right)\right]$$

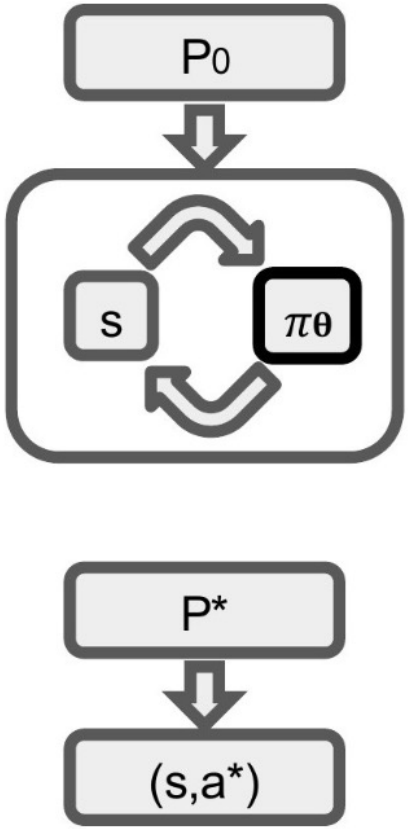
- State distribution $P(s|\pi_\theta)$ depends on rollout determined by current policy π_θ

What we have

Behavior Cloning:

$$\underset{\theta}{\operatorname{argmin}}\ \underbrace{\mathbb{E}_{(s^*,a^*)\sim P^*}\left[\mathcal{L}\left(a^*,\pi_\theta(s^*)\right)\right]}_{=\sum_{i=1}^{N}\mathcal{L}\left(a_i^*,\pi_\theta(s_i^*)\right)}$$

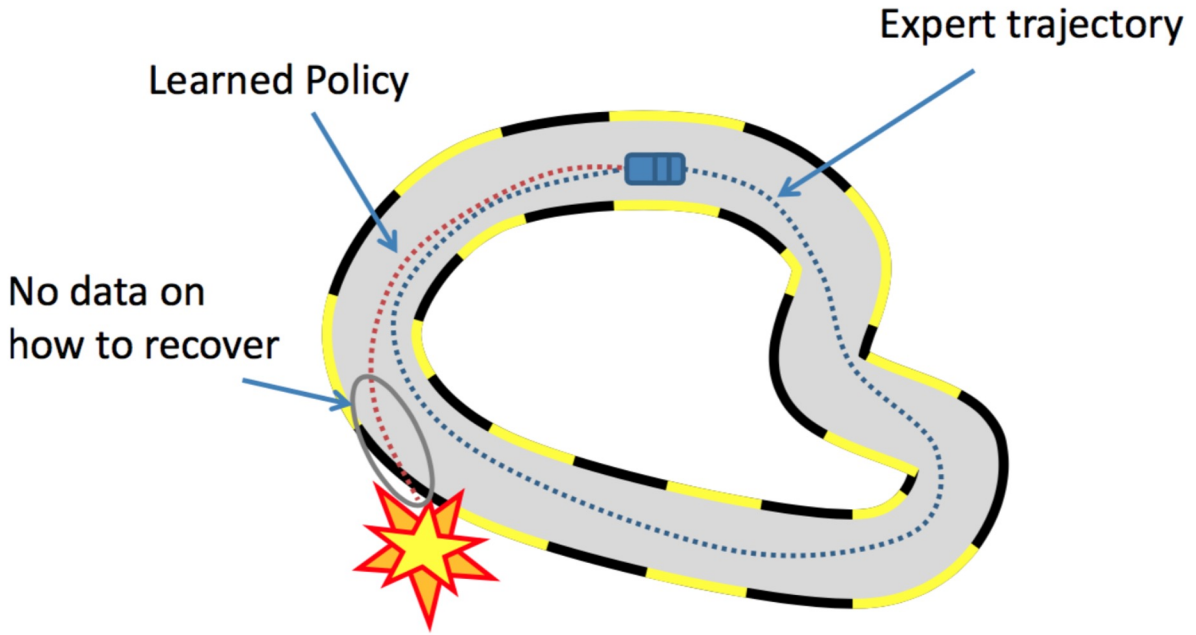
- State distribution P^* provided by expert
- Reduces to supervised learning problem



Problem: Catastrophic Failures

- Samples are from the teacher’s distribution.
- What the agent experiences are states from rollouts of its own policy.
- Catastrophic failures

Why catastrophic failures occur?



Data distribution mismatch!

In supervised learning, $(x, y) \sim D$ during train **and** test. In MDPs:

- Train: $s_t \sim D_{\pi^*}$
- Test: $s_t \sim D_{\pi_\theta}$

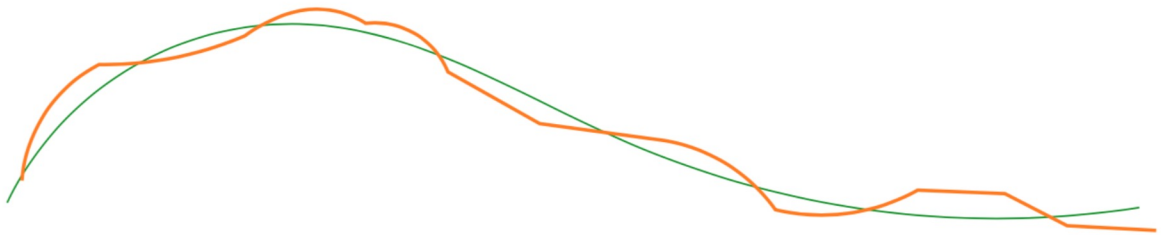
Distributional Shift Problem

Supervised learning approach assumes that the training and the test distributions are the same.

Expected number of total errors grows as T

Now apply this supervised learning setup to the MDP or RL context.

Supervised learning assumes iid. (s, a) pairs and ignores temporal structure
Independent in time errors:



Error at time t with probability ϵ
$\mathbb{E}[\text{Total errors}] \leq \epsilon T$

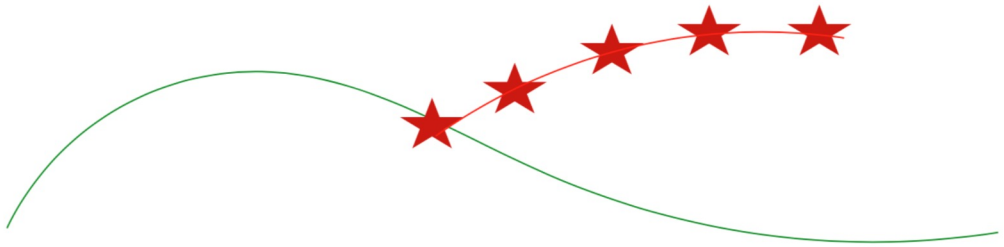
Distributional Shift Problem

Key: Sequence of states experienced in an MDP depends on the actions taken while executing a policy.

The actions determine the distribution of states the agent will be encountering. The samples are not IID as in supervised learning.

An error can lead the agent to parts of the state space where it hasn't been trained on. Hence, it will make more errors.

At any time step will make T more errors then T-1 more errors, then T-2 more errors. The errors compound a lot. Hence, the error overall grows as T^2.



Error at time t with probability ϵ

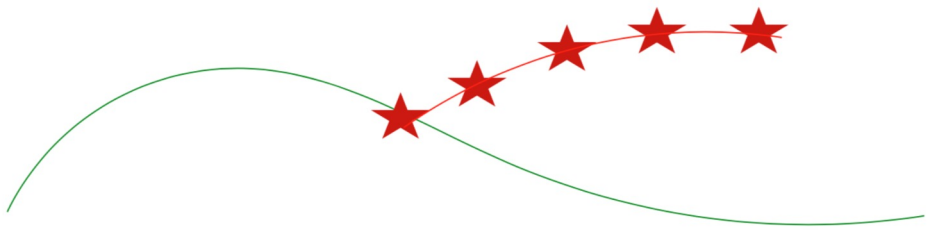
$\mathbb{E}[\text{Total errors}] \leq \epsilon(T + (T-1) + (T-2)\ldots + 1) \propto \epsilon T^2$

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, Ross et al. 2011

Key insight: errors compound a lot when the iid assumption is violated.

Distributional Shift Problem

Catastrophic accumulation of errors



Error at time t with probability ϵ

$\mathbb{E}[\text{Total errors}] \leq \epsilon(T + (T-1) + (T-2)\ldots + 1) \propto \epsilon T^2$

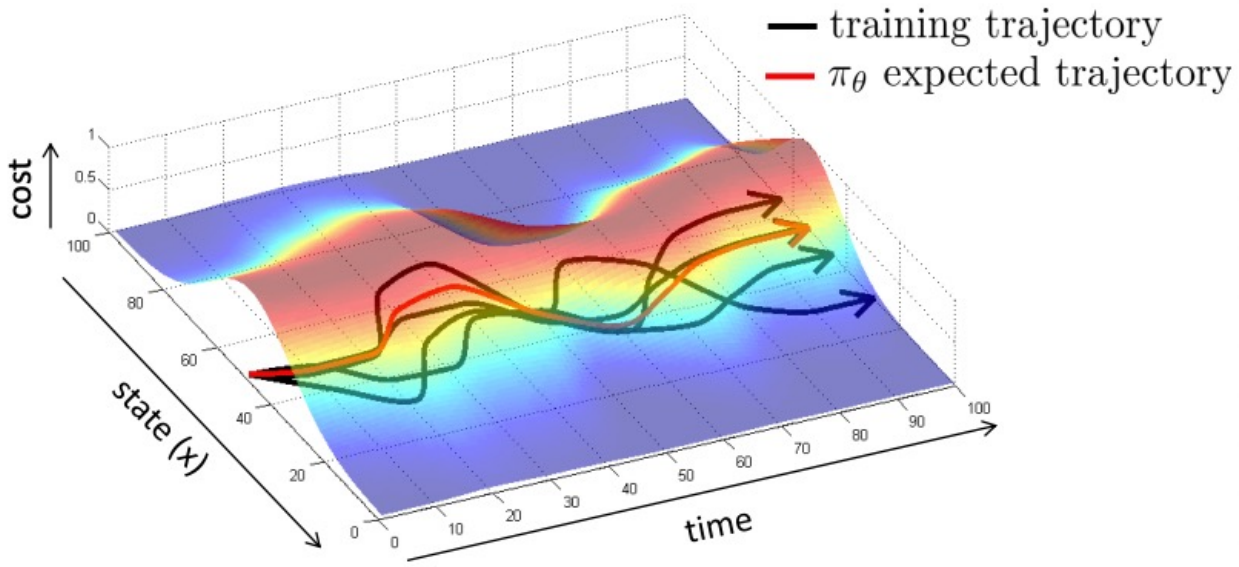
Analogy is with tight-rope walking! Recovery is hard.

The core challenge in imitation learning is how to learn from state-action labels from the teacher while accounting for the sequential nature of the rollout. This is also called structured prediction problem in machine learning.

How to address the problem?

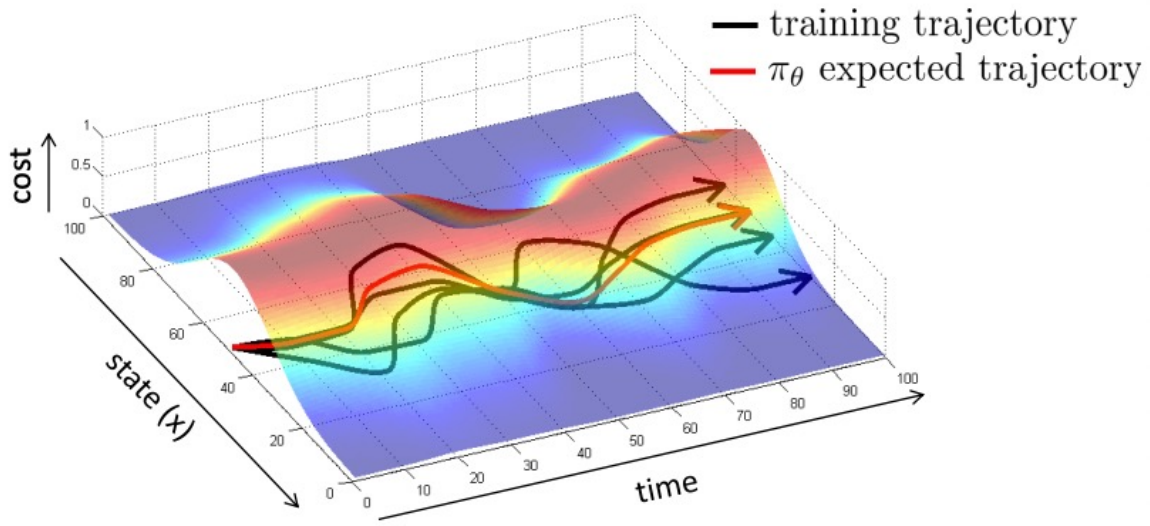
- Core Problem: The distribution of states provided by teacher's policy (where there are action labels) does not cover all the states that may be seen by the agent when its own policy is rolled out.

- Data Augmentation: Add examples (state-action pairs) in expert demonstration trajectories to cover the states/ observations points where the agent will land when trying out its own policy.



Need for Data Augmentation

- In essence, increase the availability of data (hence called augmentation). How?
  - As an expert to label more data (on the fly). Interactively query the expert when a new state is encountered.
  - Increase the size observation space itself (so that the expert covers more state space).



Data Augmentation I: Data set Aggregation (DAGGER)

- Core Idea
  - Get the distribution of states encountered by the learned policy close to the distribution of states for which we have labels.
- An interactive expert.
  - Assume access to an interactive expert.
  - Get more labels of the expert action in the new states along the policy computed by BC.
  - Can provide “action” or any “state” we query. Interactively get the action labels for the encountered state.
- Keep adding the data.
  - Essentially train from all past mistakes.
  - Perform policy blending.
- The policy improves with the increasing data (guarantees exist).

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.

https://www.ri.cmu.edu/pub_files/2011/4/Ross-AISTATS11-NoRegret.pdf

Policy Aggregation

- Blend the policy used to collect the data between the expert and the trained policy in each iteration.
- With a certain likelihood (beta) decide to pick the action given by the expert vs. decide to act as per the last trained policy.
- Initially, beta is one and then decays as the trained policy improves.
- In essence, encouraging exploration with the acquired policy w.r.t. the expert policy.
- This approach is called SMILe (Ross and Bragnell 2010)

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.

https://www.ri.cmu.edu/pub_files/2011/4/Ross-AISTATS11-NoRegret.pdf

Autonomous Flight Using DAGGER

Learning Monocular Reactive UAV Control in Cluttered Natural Environments

Stéphane Ross*, Narek Melik-Barkhudarov*, Kumar Shaurya Shankar*,
Andreas Wendel†, Debadeepta Dey*, J. Andrew Bagnell* and Martial Hebert*
*The Robotics Institute
Carnegie Mellon University, Pittsburgh, PA, USA
Email: {sross1, nmelikba, kumarsha, debadeep, dbagnell, hebert}@andrew.cmu.edu
†Institute for Computer Graphics and Vision
Graz University of Technology, Austria
Email: wendel@icg.tugraz.at

Fig. 1. We present a novel method for high-speed, autonomous MAV flight through dense forest areas. The system is based on purely visual input and imitates human reactive control.

https://www.youtube.com/watch?v=hNsP6-K3Hn4&t=2s

Technical Details

B. The DAgger Algorithm

DAgger trains a policy that mimics the expert's behavior through multiple iterations of training. Initially, the expert demonstrates the task and a first policy π_1 is learned on this data (by solving a classification or regression problem). Then, at iteration n, the learner's current policy π_{n-1} is executed to collect more data about the expert's behavior. In our particular scenario, the drone executes its own controls based on π_{n-1}, and as the drone is flying, the pilot provides the correct actions to perform in the environments the drone visits, via a joystick. This allows the learner to collect data in new situations which the current policy might visit, but which were not previously observed under the expert demonstrations, and learn the proper recovery behavior when these are encountered. The next policy π_n is obtained by training a policy on all the training data collected over all iterations (from iteration 1 to n). This is iterated for some number of iterations N and the best policy found at mimicing the expert under its induced distribution of environments is returned. See [9] for details.

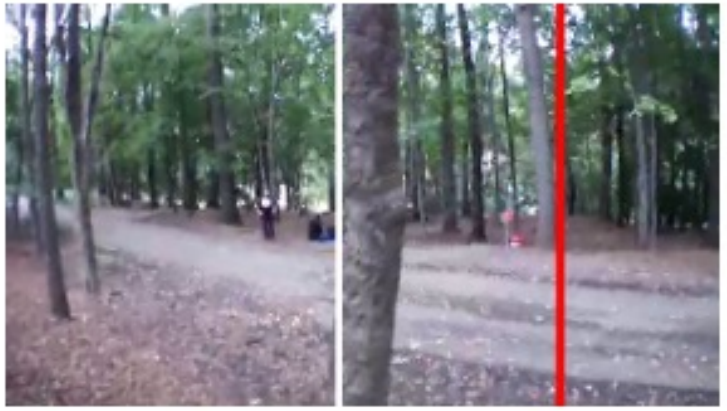
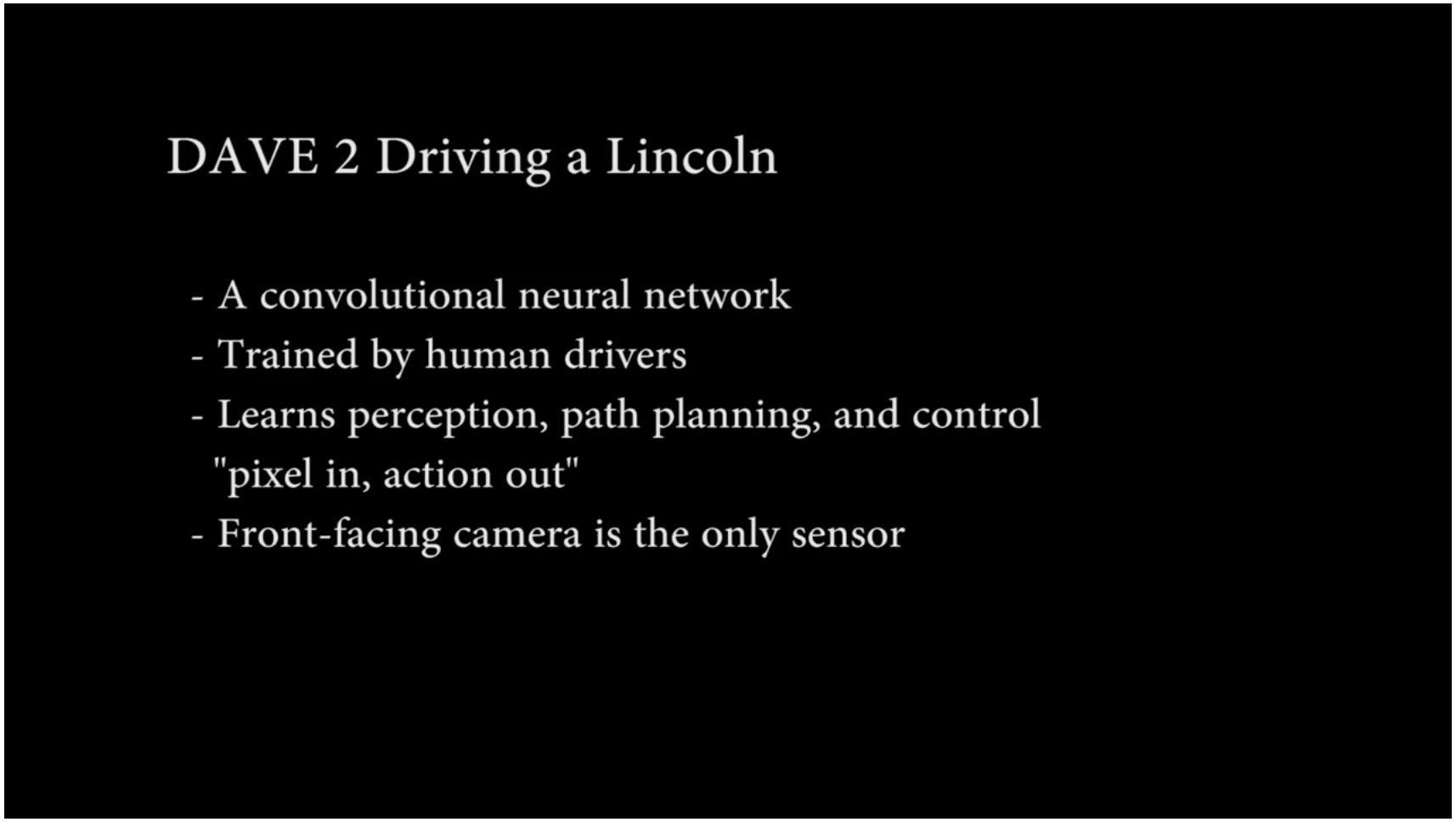


Fig. 2. One frame from MAV camera stream. The white line indicates the current yaw commanded by the current DAgger policy π_{n-1} while the red line indicates the expert commanded yaw. In this frame DAgger is wrongly heading for the tree in the middle while the expert is providing the correct yaw command to go to the right instead. These expert controls are recorded for training later iterations but not executed in the current run.

Paper: https://arxiv.org/pdf/1211.1690.pdf

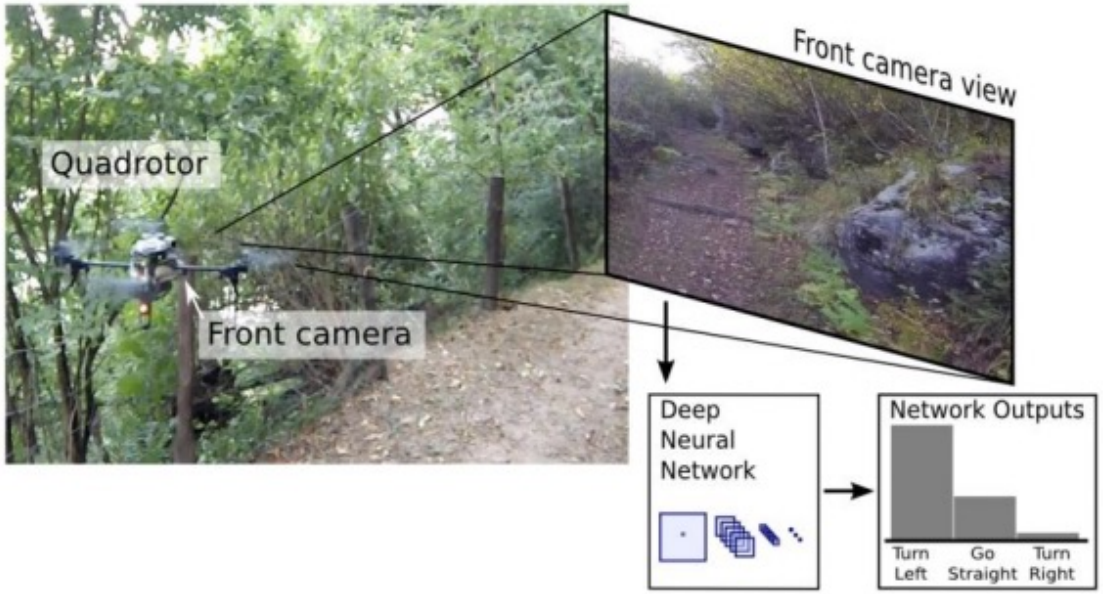
Example: Dave 2 End-to-end Driving System



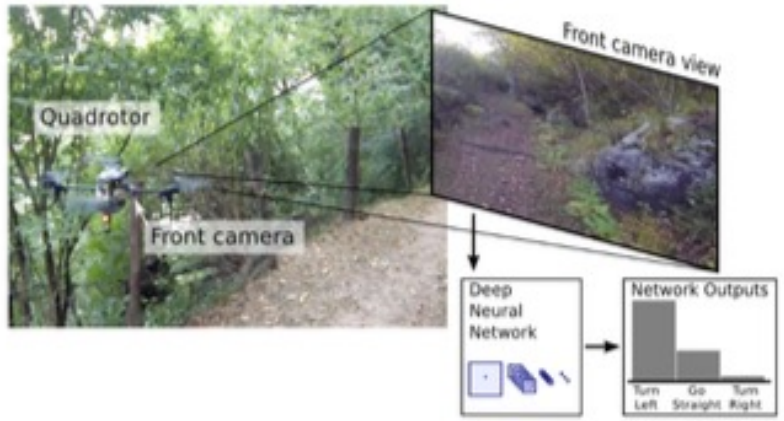
Data Augmentation II: Increased Observation Size

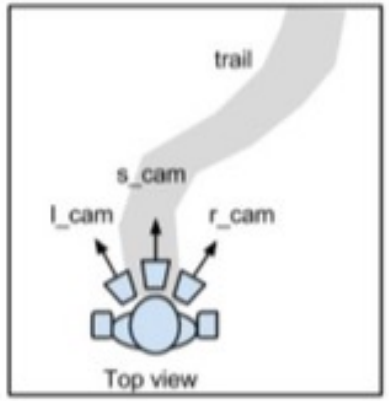
A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots

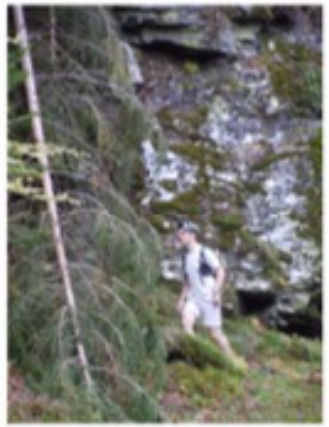
Alessandro Giusti[1], Jérôme Guzzi[1], Dan C. Cireşan[1], Fang-Lin He[1], Juan P. Rodríguez[1]
Flavio Fontana[2], Matthias Faessler[2], Christian Forster[2]
Jürgen Schmidhuber[1], Gianni Di Caro[1], Davide Scaramuzza[2], Luca M. Gambardella[1]



Data Augmentation II: Increased Observation Siz







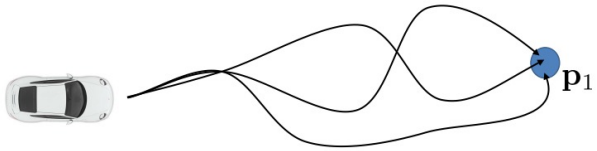
A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots. *Giusti et al. 2015*

Imitation Learning for Multiple Goals/Tasks

Often, we care about learning policies that achieve many related goals.

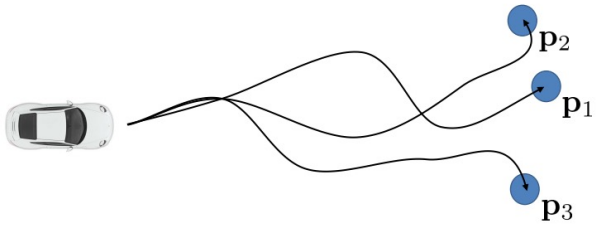
Examples

- Pushing to diverse locations.
- Pouring to different bottles
- Driving to different destinations.



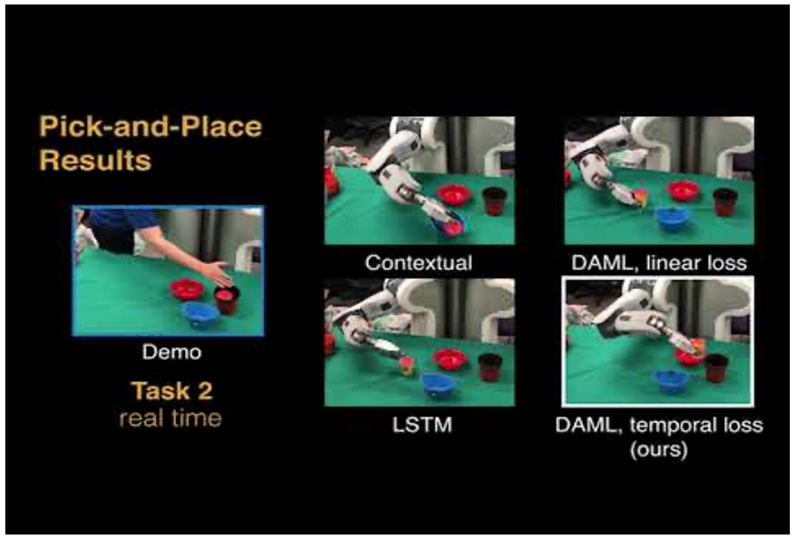
$\pi_\theta(\mathbf{a}|\mathbf{s})$

policy for reaching $\mathbf{p}_1$



$\pi_\theta(\mathbf{a}|\mathbf{s}, \mathbf{p})$

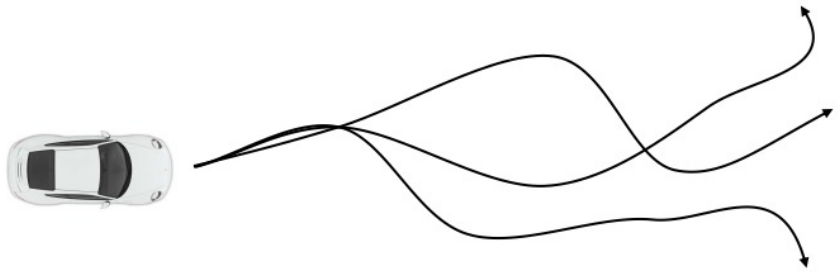
policy for reaching *any* $\mathbf{p}$



Imitation Learning for Multiple Tasks (Goals)

Shared structure in learning

- For example: push/moving object A to (10,10,10) and to (10,12,10)
- The two policies should have many things in common
- Training such policies jointly may be beneficial



training time:

demo 1: $\{\mathbf{s}_1, \mathbf{a}_t, \ldots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, \mathbf{s}_T\}$ ⟵ successful demo for reaching $\mathbf{s}_T$

demo 2: $\{\mathbf{s}_1, \mathbf{a}_t, \ldots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, \mathbf{s}_T\}$ learn $\pi_\theta(\mathbf{a}|\mathbf{s}, \mathbf{g})$ ⟵ goal state

demo 3: $\{\mathbf{s}_1, \mathbf{a}_t, \ldots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, \mathbf{s}_T\}$

for each demo $\{\mathbf{s}_1^i, \mathbf{a}_1^i, \ldots, \mathbf{s}_{T-1}^i, \mathbf{a}_{T-1}^i, \mathbf{s}_T^i\}$

maximize $\log \pi_\theta(\mathbf{a}_t^i | \mathbf{s}_t^i, \mathbf{g} = \mathbf{s}_T^i)$

Imitation Learning for Multiple Tasks (Goals)

- Need to communicate the goal during learning of the policy.
- MDPs in the multi-goal setting
  - Policy and the rewards are conditioned on the goal.
  - The experience tuples contain the goal.
- Learning is now conditioned on the goal. Alternatively, we assign a goal that the demonstration was trying to attain.

$$V(s;\theta) \rightarrow V(s,g;\theta)$$

$$\pi(s;\theta) \rightarrow \pi(s,g;\theta)\ s,g \in \mathcal{S}$$

$$(s,a,r,s') \rightarrow (s,g,a,r,s')$$

Universal Value Function Approximators, Schaul et al.

Multi-goal MDP

Formalism

We define a discrete-time finite-horizon discounted Markov decision process (MDP) by a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma, H)$, where $\mathcal{S}$ is a state set, $\mathcal{A}$ is an action set, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ is a transition probability distribution, $\gamma \in [0, 1]$ is a discount factor, and H is the horizon. Our objective is to find a stochastic policy π_θ that maximizes the expected discounted reward within the MDP, $\eta(\pi_\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T} \gamma^t r(s_t, a_t, s_{t+1})]$. We denote by $\tau = (s_0, a_0, ...,)$ an entire state-action trajectory, where $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi_\theta(\cdot|s_t)$, and $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$. In the goal-conditioned setting that we use here, the policy and the reward are also conditioned on a "goal" $g \in \mathcal{S}$. The reward is $r(s_t, a_t, s_{t+1}, g) = \mathbb{1}\left[s_{t+1} == g\right]$, and hence the return is the γ^h, where h is the number of time-steps to the goal. Given that the transition probability is not affected by the goal, g can be "relabeled" in hindsight, so a transition $(s_t, a_t, s_{t+1}, g, r = 0)$ can be treated as $(s_t, a_t, s_{t+1}, g' = s_{t+1}, r = 1)$. Finally, we also assume access to D trajectories $\left\{(s_0^j, a_0^j, s_1^j, ...)\right\}_{j=0}^{D} \sim \tau_{\text{expert}}$ that were collected by an expert attempting to reach the goals $\{g_j\}_{j=0}^{D}$ sampled uniformly among the feasible goals. Those trajectories must be approximately geodesics, meaning that the actions are taken such that the goal is reached as fast as possible.

Goal-conditioned Imitation Learning: https://arxiv.org/pdf/1906.05838.pdf

Goal-Conditioned Behavior Cloning

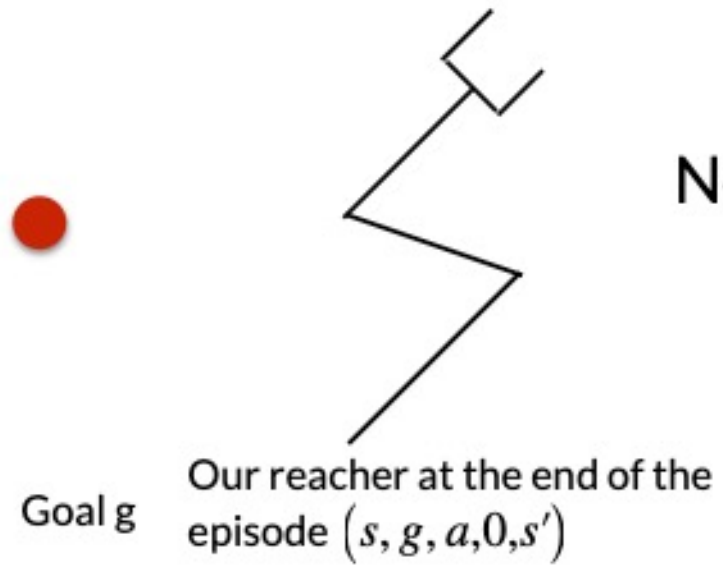
Assumes access to a set of trajectories $\mathcal{T} = \{o_1^j, a_1^j, o_2^j, a_2^j, o_3^j, a_3^j, \ldots, o_T^j, a_T^j, g^j, j = 1 \ldots T\}$. Trains a policy by minimizing a standard supervised learning objective:

$$\mathcal{L}_{BC}(\theta, \mathcal{T}) = \mathbb{E}_{(s_t^j, a_t^j, g^j) \sim \mathcal{T}} \left[\| a_t^j - \pi_\theta(s_t^j, g^j) \|_2^2 \right]$$

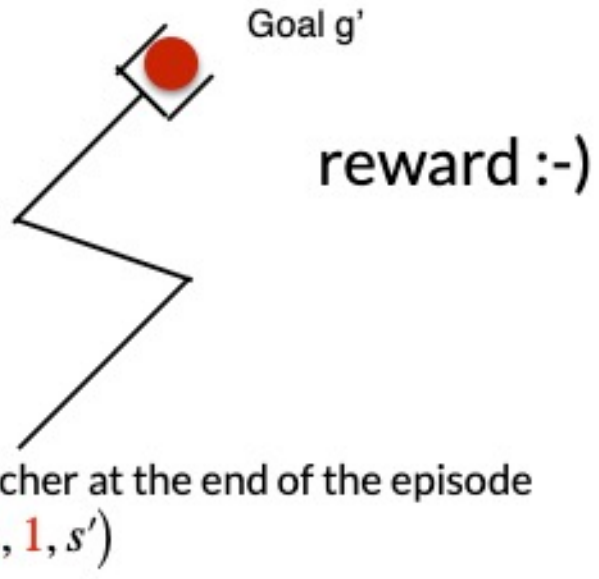
- Note: Data coverage problem is still there. The problem is limited expert data.
- How to perform “data augmentation” under the multi-goal setting.
- There are interesting ways to do this!

Goal Relabeling

Idea: use failed executions under one goal g, as successful executions under an alternative goal g' (which is where we ended at the end of the episode).



No reward :-(

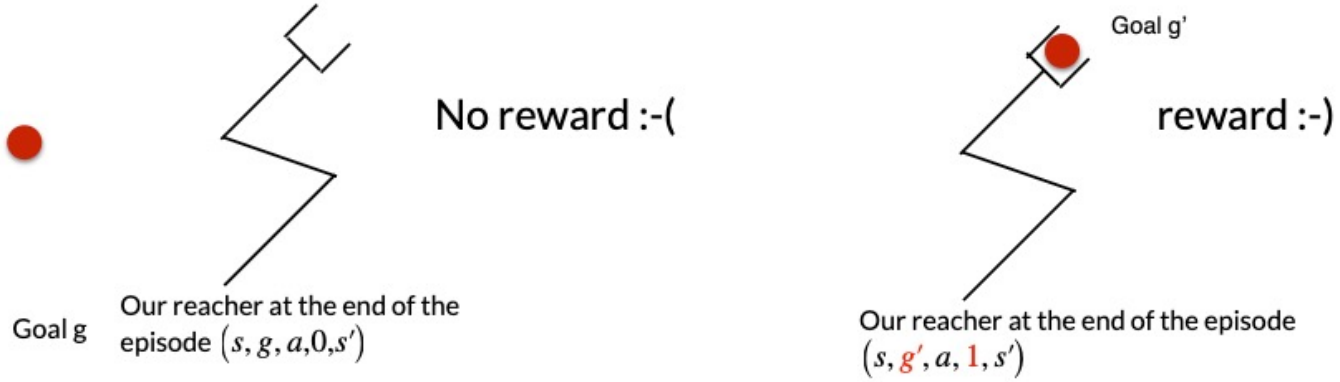


reward :-)

Assigning the goal as the state the agent lands up in and changing reward.

Goal Relabeling

The reward obtained for the transition that landed in the goal is relabelled to a reward of 1 (from zero)



Relabelling augments expert demonstrations.

If $(s_t^j, a_t^j, s_{t+1}^j, g^j)$ is in a demonstration trajectory, we also add $(s_t^j, a_t^j, s_{t+1}^j, g' = s_{t+k}^j)$

Transition tuples are to be allocated a goal. These can be the original goals (picking of a specific block) or the relabelled state where the robot hand reached.

Hindsight Experience Replay (HER)

Hindsight Experience Replay

Marcin Andrychowicz*, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong,
Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel†, Wojciech Zaremba†
OpenAI

Idea: use failed executions under one goal g, as successful executions under an alternative goal g' (which is where we ended at the end of the episode).

Key idea:

- Set of “failed” demonstration for a goal provide data for ”another” goal. The data gets augmented! It is a general technique applicable widely in RL

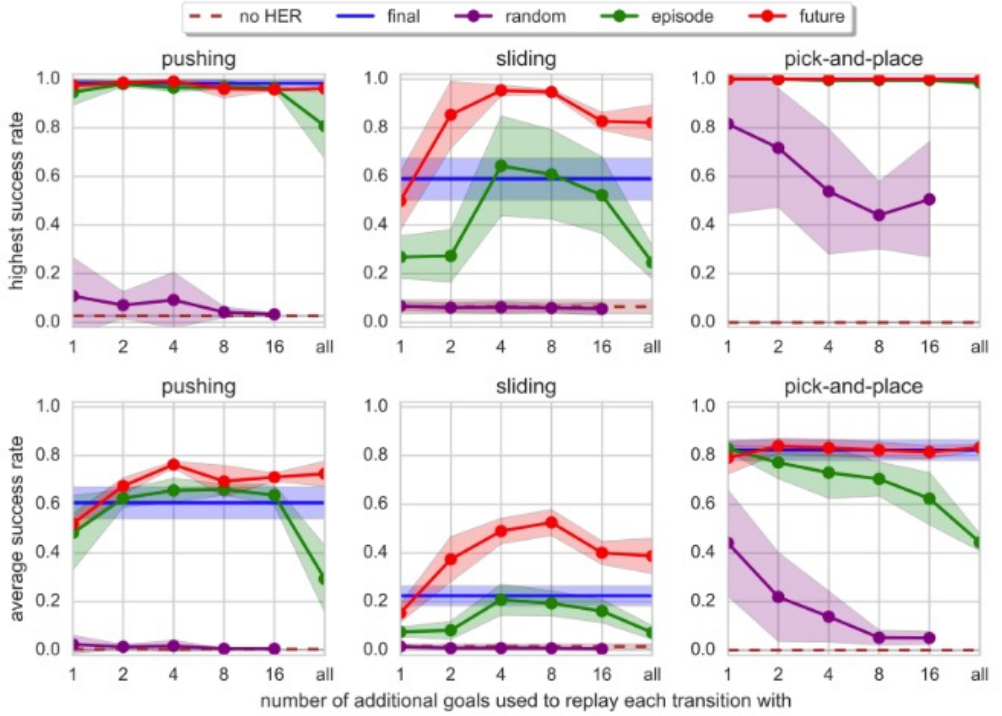
Selecting States as Additional Goals for Relabeling

- Final states of a rollout. For each terminal state, use the last state reached in the episode as a goal
- States encountered in future. For each state, select a random number (say 4) states observed after the state in the same episode as a goal.
- Each transition is then replayed with additional goals.
- Empirically, above 8 (the number of additional goals used to replay each transition with), the performance degrades. The relabelled data are “way more” than real data, performance degrades.

Actual goals for picking

Figure 7: The pick-and-place policy deployed on the physical robot.

Impact of increasing the additional goals used for relabeling.



Hindsight Experience Replay (HER)

- RL with goal relabeling.
- The core idea is one of goal relabeling.
- Select certain states as ”goals” use roll outs that reach these goals for policy training.
- The specifics of the algorithm will be clearer when more RL techniques are covered later.

Algorithm 1 Hindsight Experience Replay (HER)

Given:

- an off-policy RL algorithm $\mathbb{A}$, ▷ e.g. DQN, DDPG, NAF, SDQN
- a strategy $\mathbb{S}$ for sampling goals for replay, ▷ e.g. $\mathbb{S}(s_0, \ldots, s_T) = m(s_T)$
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$. ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$

Initialize $\mathbb{A}$ ▷ e.g. initialize neural networks
Initialize replay buffer R
for episode = 1, M **do**
  Sample a goal g and an initial state s_0.
  for $t = 0, T-1$ **do**
    Sample an action a_t using the behavioral policy from $\mathbb{A}$:
      $a_t \leftarrow \pi_b(s_t||g)$ ▷ $||$ denotes concatenation
    Execute the action a_t and observe a new state s_{t+1}
  end for
  for $t = 0, T-1$ **do**
    $r_t := r(s_t, a_t, g)$
    Store the transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ in R ▷ standard experience replay
    Sample a set of additional goals for replay $G := \mathbb{S}$(**current episode**)
    for $g' \in G$ **do**
      $r' := r(s_t, a_t, g')$
      Store the transition $(s_t||g', a_t, r', s_{t+1}||g')$ in R ▷ HER
    end for
  end for
  for $t = 1, N$ **do**
    Sample a minibatch B from the replay buffer R
    Perform one step of optimization using $\mathbb{A}$ and minibatch B
  end for
end for

The reward here is $\|s_t - g\|$

G : the states of the current episode

Usually as additional goal we pick the goal that this episode achieved, and the reward becomes non zero