Imitation Learning

Tutorial in the SMILES Summer School
21 October 2020

Part 1

Kamil Ciosek

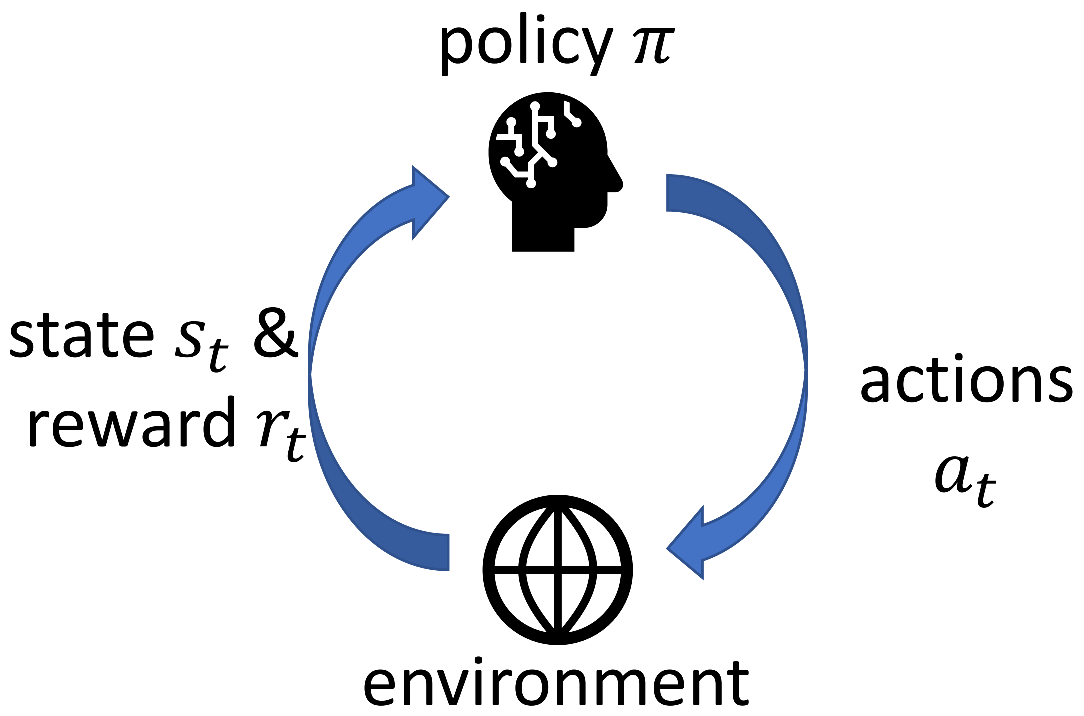
Imitation Learning Tutorial - Logistics

- Session 1: (40 minute talk + 5-10 minutes for questions).
- Session 2: (40 minute talk + 5-10 minutes for questions).

Questions about imitation learning will be prioritized, but please feel free to ask about RL or RL at Microsoft!

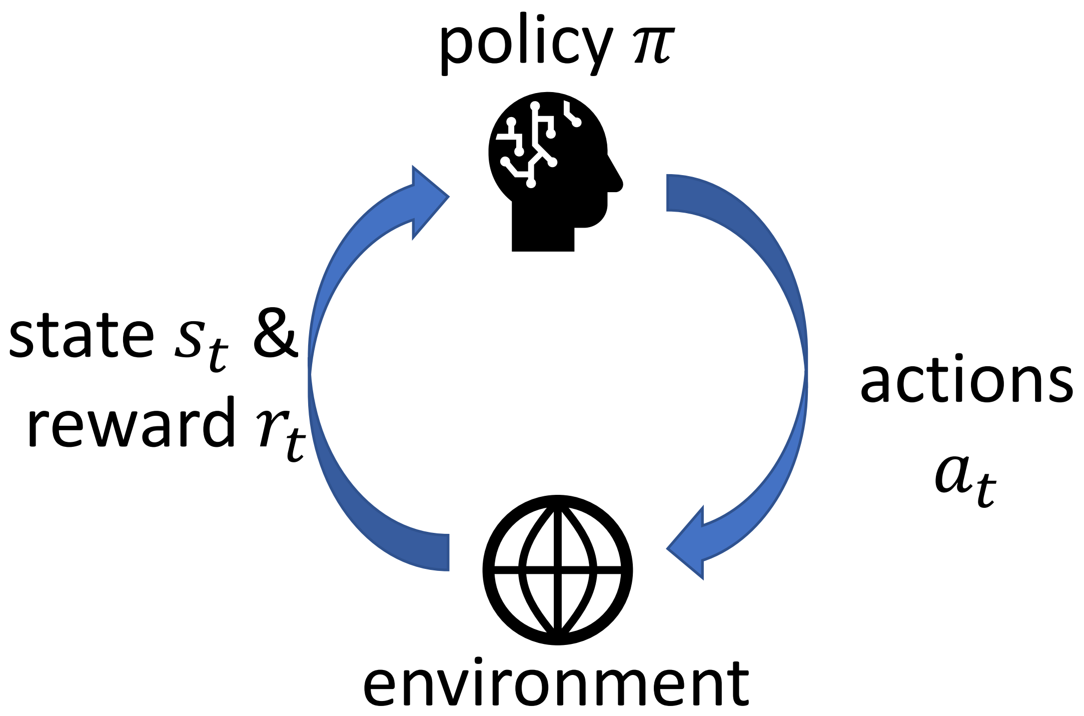
So you want to play Minecraft?

Reinforcement Learning



Train a game bot!

Reinforcement Learning - Formalization



Markov Decision Process
(average-reward version)

$$M = (S, A, T, R, p_0)$$

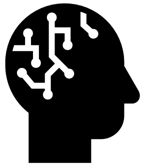
- States $s_t \in S$
- Actions $a_t \in A$
- Transitions $T(s'|\, s, a)$
- Reward $r \in R$, bounded

Given $\pi\colon\ S \to \Delta(A)$, we generate trajectories:

$$\tau = (s_0, a_0, s_1, a_1, \dots, s_H, a_H)$$

Policies and Returns

policy π



$$\pi\colon\ S \to \Delta(A)$$

$$J_\theta = \lim_{H\to\infty} E_\tau\left[\frac{1}{H}\sum_{t=1}^{H} r_t\right]$$

$$\pi^\star = \arg\max_\theta J_\theta$$

Markov Decision Process
(average-reward version)

$$M = \ (S, A, T, R, p_0)$$

- States $s_t \in S$
- Actions $a_t \in A$
- Transitions $T(s' |\ s, a)$
- Reward
  $R(s,a) \in [0,1], r_t = R(s_t, a_t)$

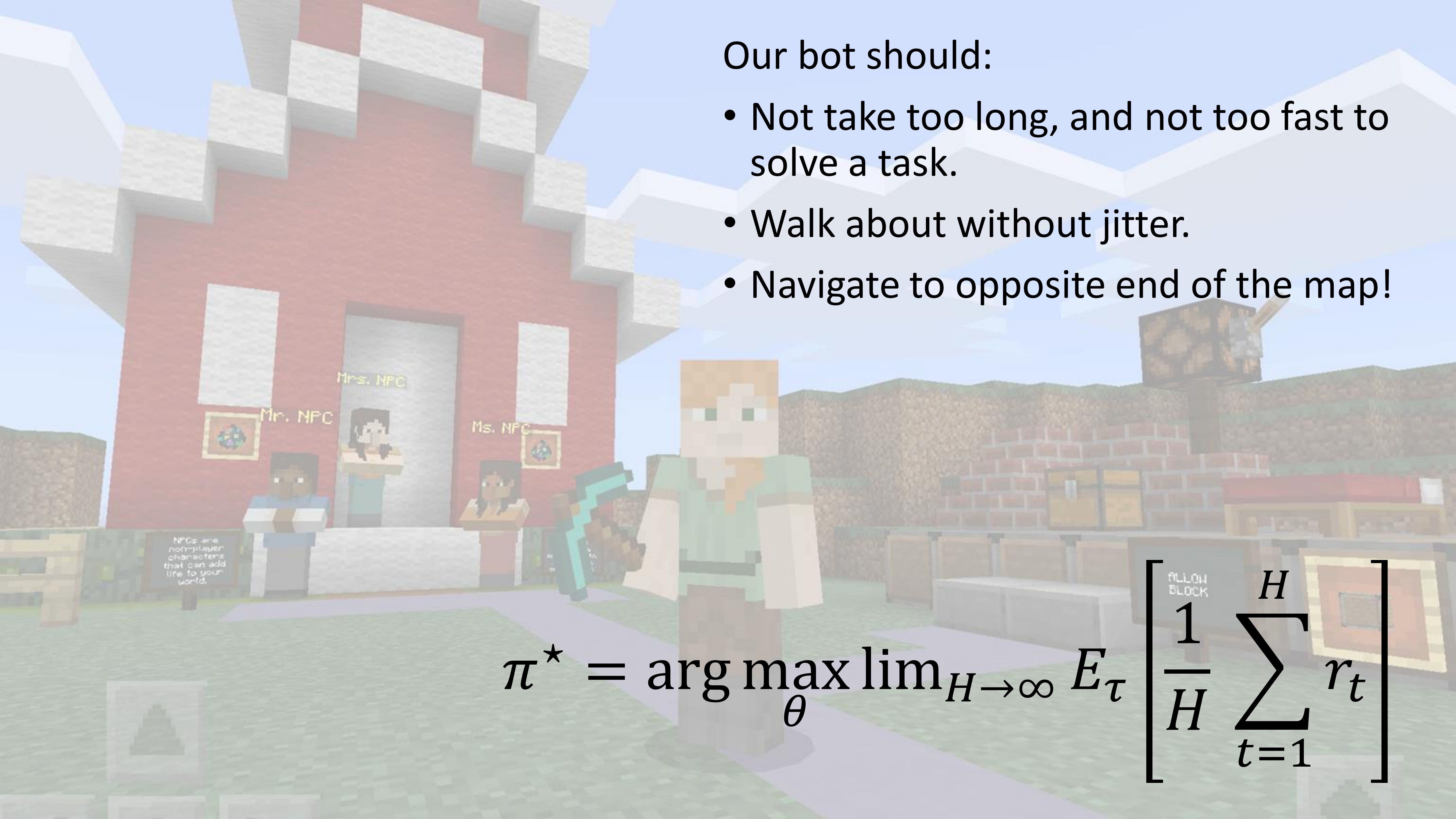
Given $\pi\colon\ S \to \Delta(A)$, we generate trajectories:

$$\tau = (s_0, a_0, s_1, a_1, \dots, s_H, a_H)$$

Our bot should:

- Not take too long, and not too fast to solve a task.
- Walk about without jitter.
- Navigate to opposite end of the map!

$$\pi^{\star} = \arg\max_{\theta} \lim_{H\to\infty} E_{\tau}\left[\frac{1}{H}\sum_{t=1}^{H} r_t\right]$$



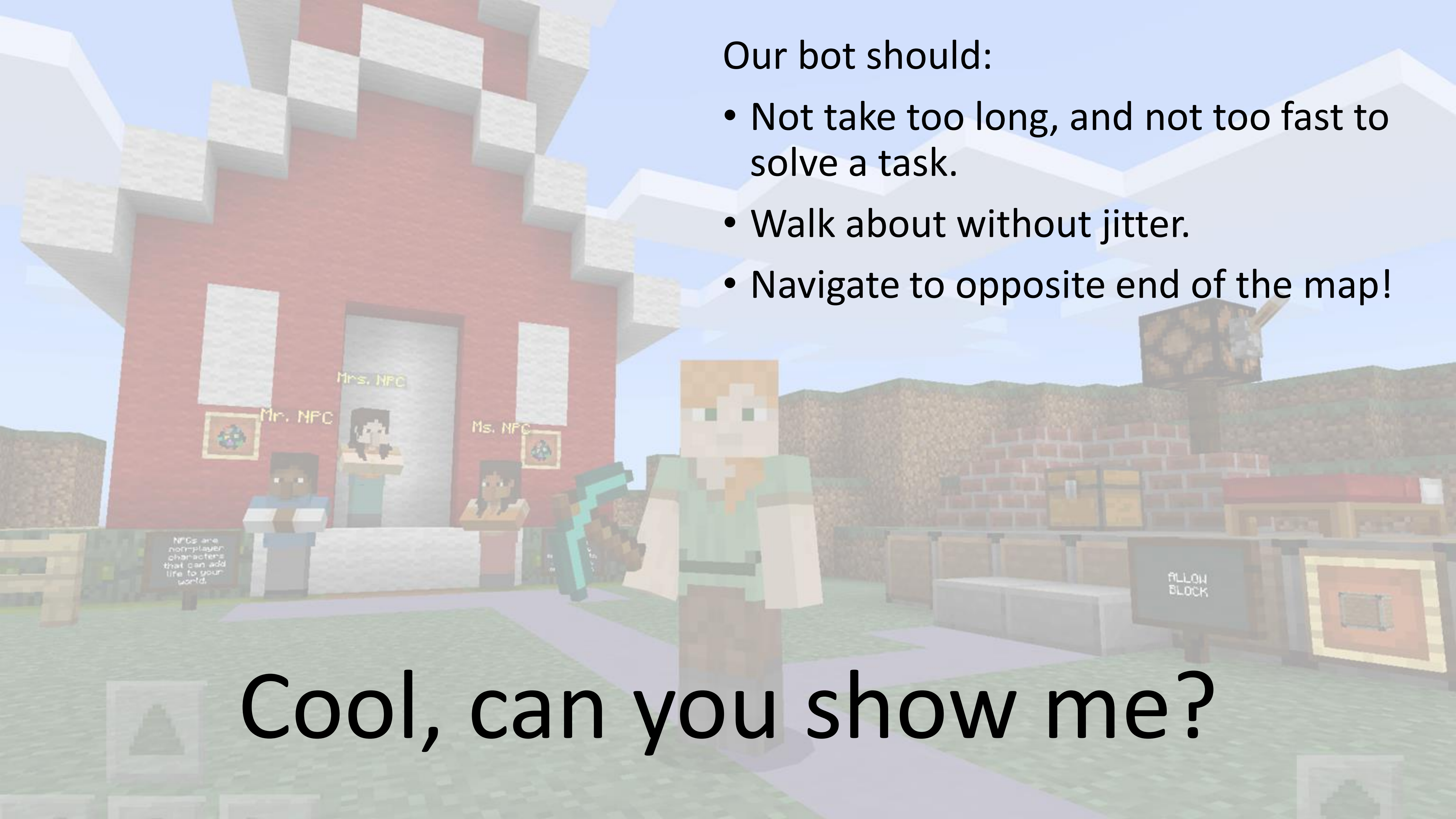
Standard Reinforcement Learning is Hard!

- Specifying these rewards is challenging. It is not always easy to describe what we want.
- Even if we get rewards right, RL methods sometimes fail to discover interesting states – exploration is still hard in many practical settings.

Solution: Imitation Learning

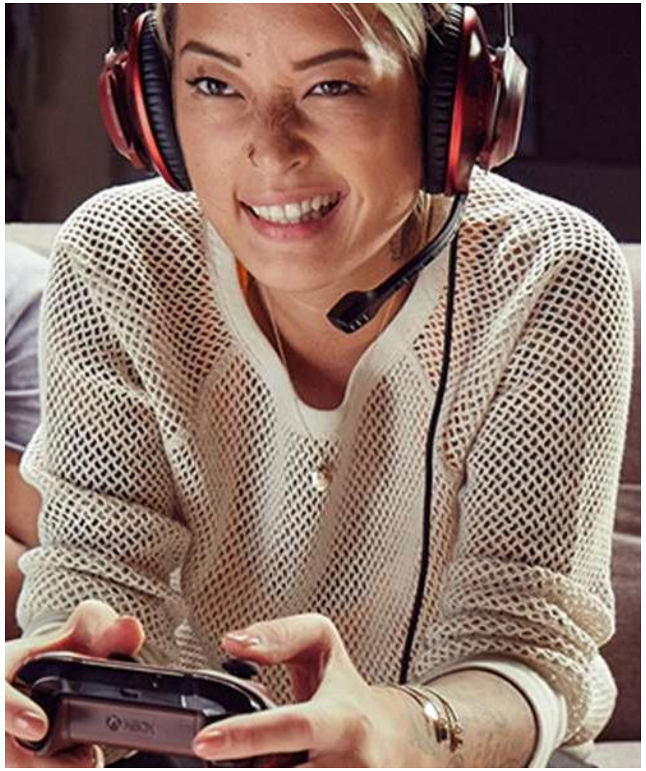
Our bot should:

- Not take too long, and not too fast to solve a task.
- Walk about without jitter.
- Navigate to opposite end of the map!



Cool, can you show me?

What is imitation learning?

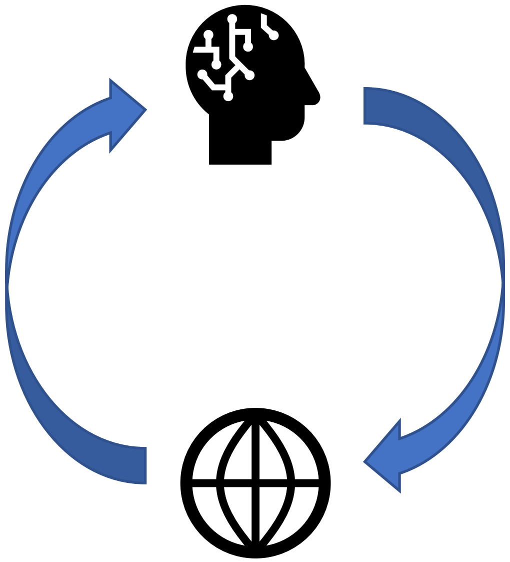


expert

data

policy $\pi\colon S \rightarrow \Delta(A)$

some IL methods use feedback loop as well



state

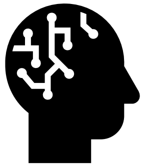
actions

environment

Reward isn't used anywhere!

Observability

policy π



π: $S \rightarrow \Delta(A)$

Markov Decision Process
(average-reward version)

$$M = (S, A, T, R, p_0)$$

In this talk, we assume:

- We have a **Markov** state
- Expert and imitation learning observe the **same** states.
- Expert and imitation learning algorithm have **same** action space.

Btw. all of these assumptions can be relaxed!

Tutorial Plan

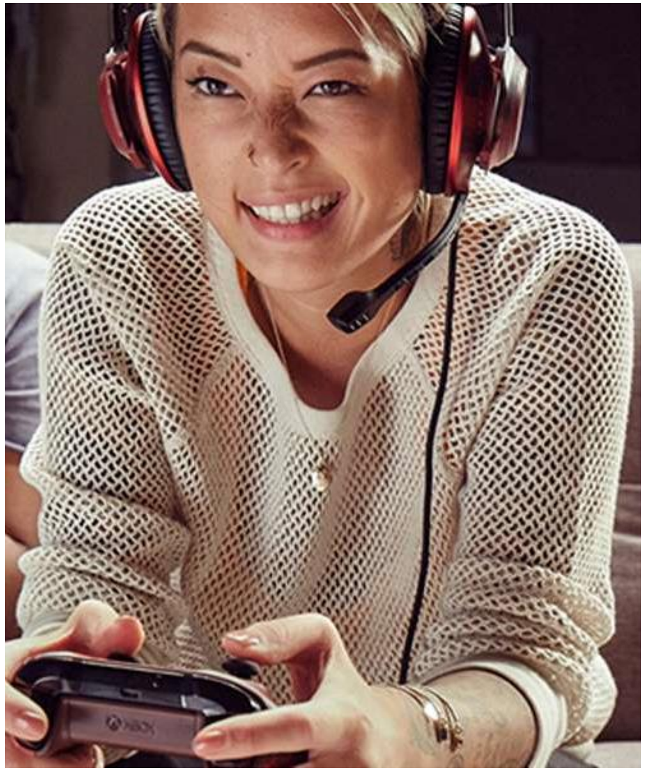
Behavioral Cloning → Inverse Reinforcement Learning → Adversarial Imitation Learning → Summary & Next Steps

Goal: you understand the basics and know where to look for more.

Behavioral Cloning

A simple Imitation Learning baseline which often works well.

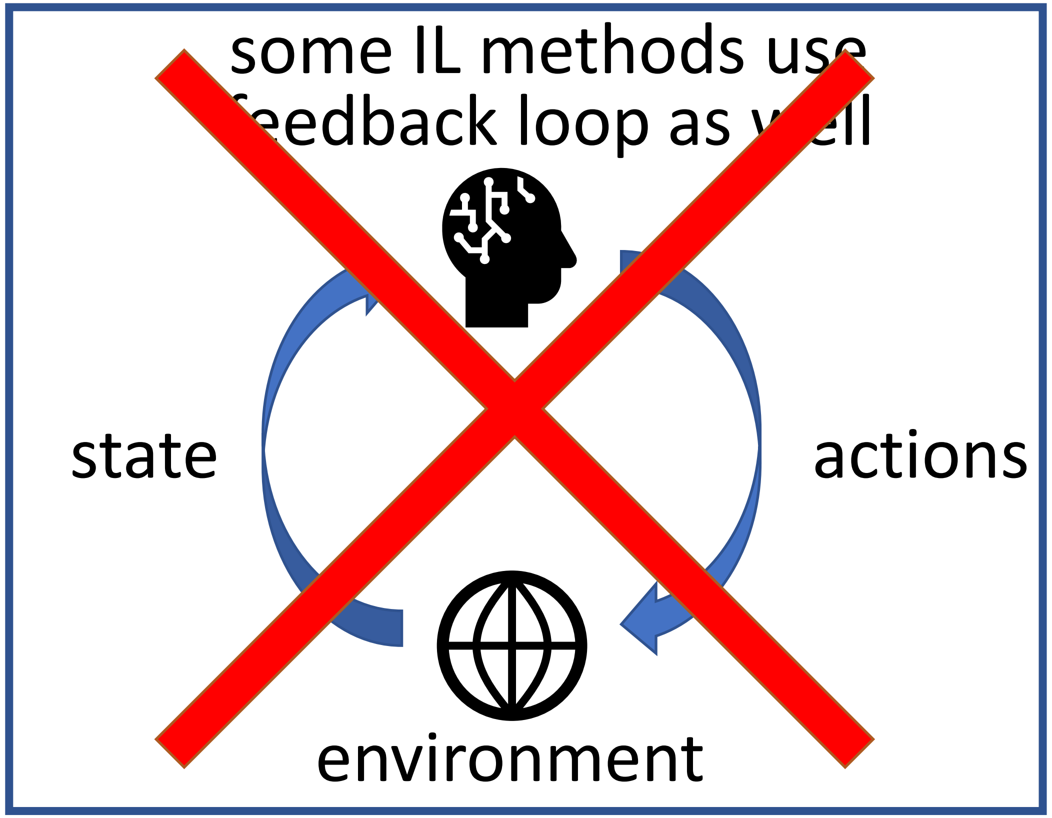
What is imitation learning?



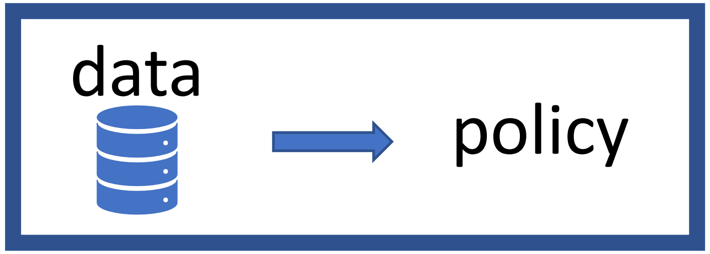
expert

data

policy π: $S \rightarrow \Delta(A)$



Behavioral Cloning



Do the simplest thing possible:
supervised learning

Maximum Likelihood

Given $\pi_\theta\colon\ S \to \Delta(A)$, we generate a trajectory:
$\tau = (s_0, a_0, s_1, a_1, \ldots, s_H, a_H)$. Denote by $P_\theta(\tau)$ the density of τ.

$$\boxed{\theta^\star = \mathrm{argmax}_\theta \log P_\theta(\tau)}$$

Maximum Likelihood estimation was introduced by R. Fisher in 1912, and is still used very often in Machine Learning.

Behavioral Cloning

$$
\begin{aligned}
&\operatorname{argmax}_\theta\ \log P_\theta(\tau) \\
&= \operatorname{argmax}_\theta \log p_0(s_0) + \sum_t \log \pi_\theta(a_t \mid s_t) + \ \log T(s_{t+1}|a_t, s_t) \\
&= \operatorname{argmax}_\theta \sum_t \log \pi_\theta(a_t \mid s_t)
\end{aligned}
$$

We can also do this with multiple trajectories (sum the likelihoods).

Implementation

Maximum likelihood is exactly what machine learning frameworks are optimized to do!

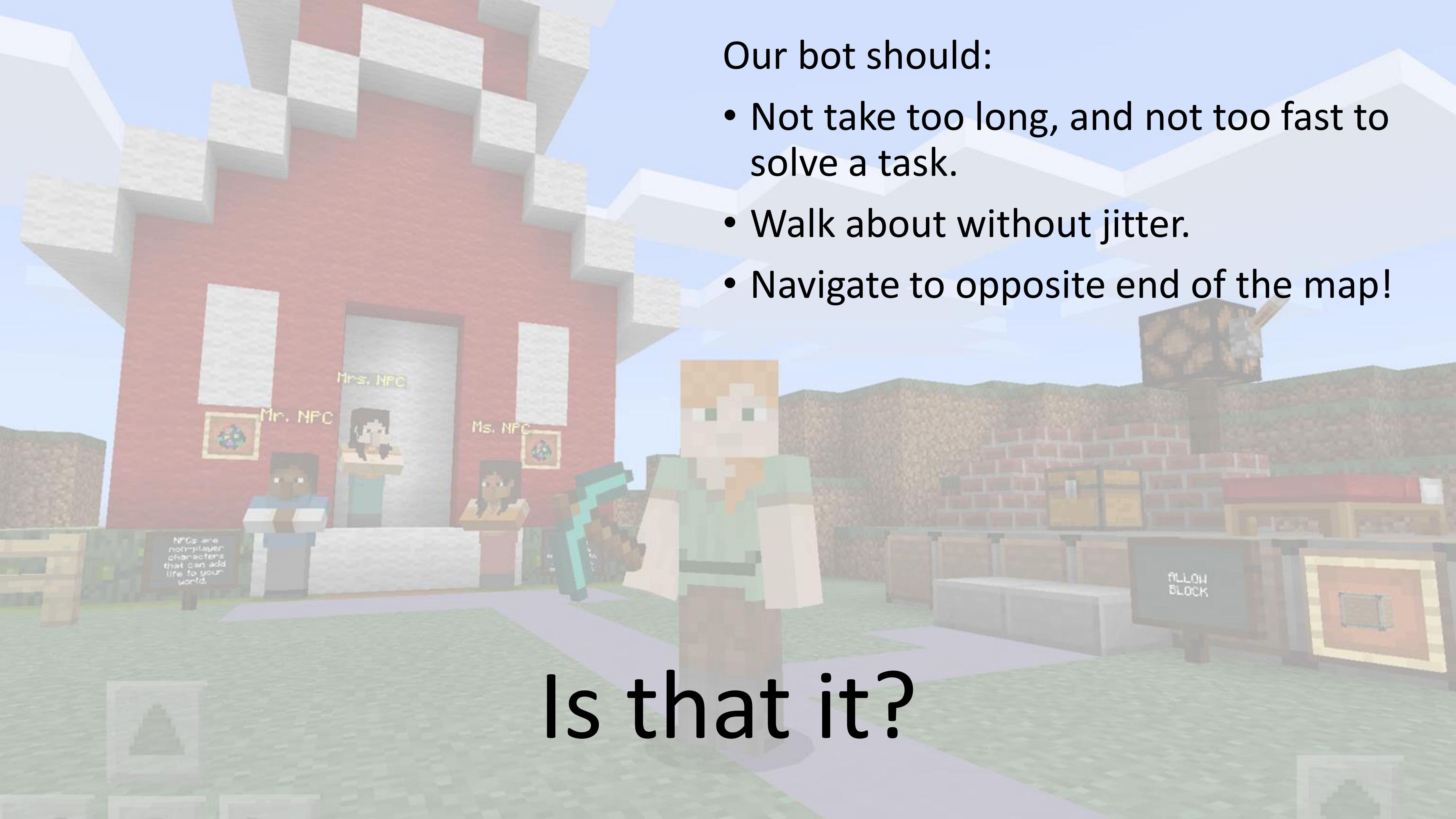
Consider the case of $s_t \in R^5, a_t \in \{0,1\}$.

Torch implementation:

```
# inputs: state_batch, action_batch
softmax_layer = nn.LogSoftmax(dim=1)
policy_layer = nn.Linear(5, 2)
policy_outputs = softmax_layer ( policy_layer (state_batch) )
loss = nn.NLLLoss()
loss(policy_outputs, action_batch)
loss.backward()
```

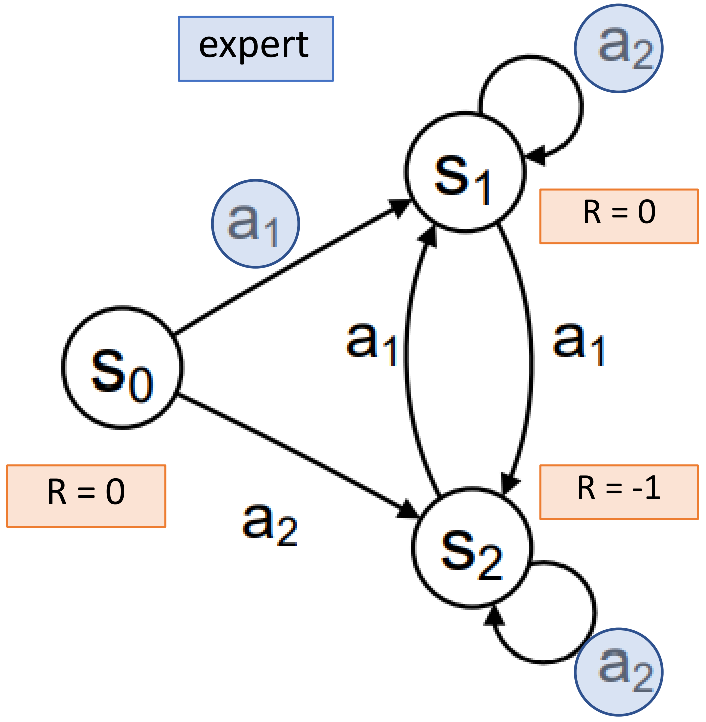
Our bot should:

- Not take too long, and not too fast to solve a task.
- Walk about without jitter.
- Navigate to opposite end of the map!



Is that it?

What could possibly go wrong?



Source: Ross & Bagnell:
Efficient Reductions for Imitation Learning

- In practice, the estimated policy will have an error.
- Consider a policy π_θ.

$$\pi_\theta(a_1|\, s_0) = 1 - T\epsilon$$
$$\pi_\theta(a_2|\, s_0) = T\epsilon$$
$$\pi_\theta(a_2|\, s_1) = 1$$
$$\pi_\theta(a_2|\, s_2) = 1$$

- Along trajectory, we have misclassification error ϵ.
- Expert return is 0.
- Return of π_θ is $(T\epsilon)(-(T-1)) = -T^2\epsilon - T\epsilon$.

What could possibly go wrong?

$$E[J_{\hat{\pi}}] = -T^2\epsilon - T\epsilon$$
$$E[J_E] = 0$$

Per-step return:

$$\frac{1}{T}(E[J_E] - E[J_{\hat{\pi}}]) = \epsilon + T\,\epsilon$$

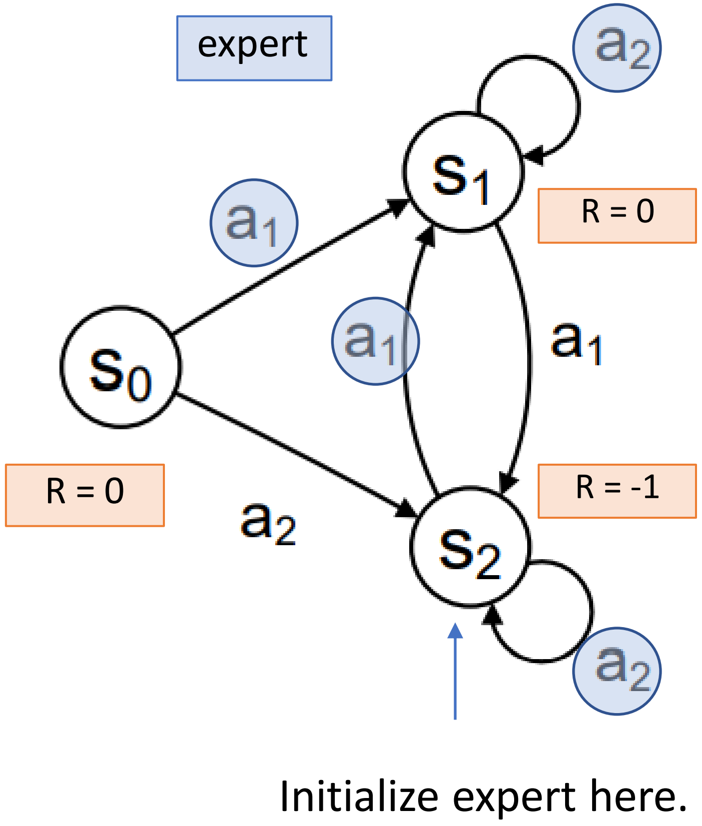
Long-term consequences
small mistakes in policy mean large difference in return.

We should be careful applying supervised learning theory to IL.

Behavioral Cloning: don’t give up on it.

- While the negative example from the slide before holds, in many practical MDPs:
  - It is possible to recover form mistakes.
  - You have data on how to recover.

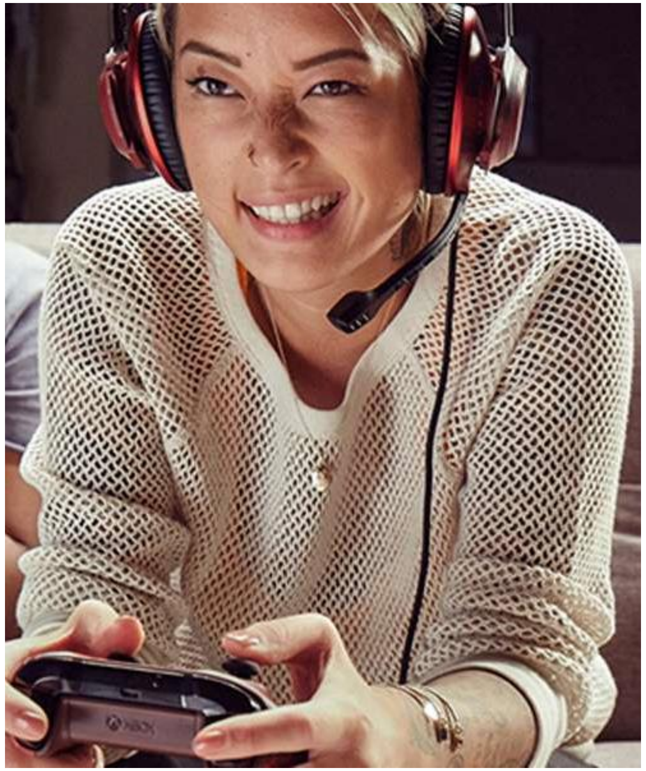
- With enough data, BC can be made to work.



Inverse Reinforcement Learning

Imitation Learning with intrinsic (made up) rewards

Imitation Learning



expert

Reward isn't used anywhere!

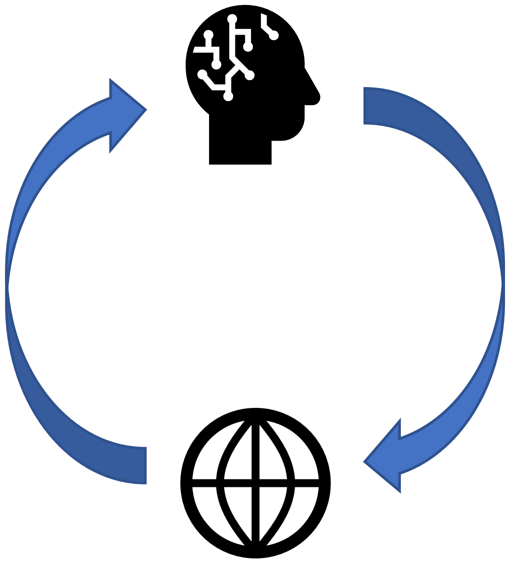
Let's come up with a reward

data

policy π: $S \rightarrow \Delta(A)$

some IL methods use feedback loop as well

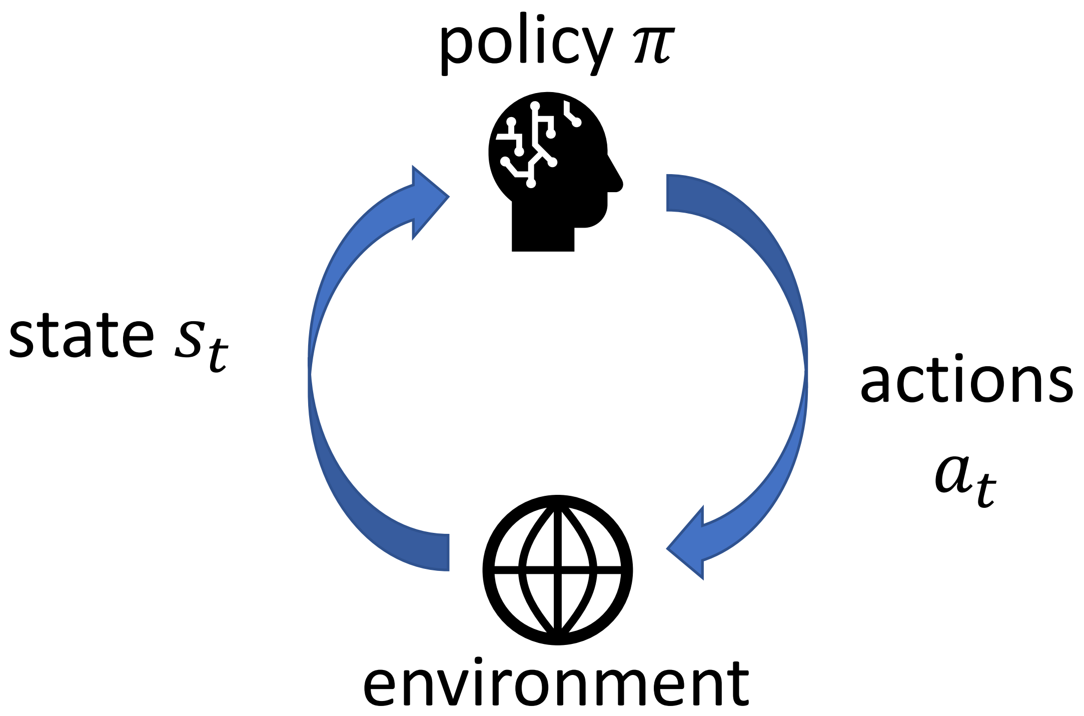
state



actions

environment

How about reconstructing expert reward?



- Assume expert is maximizing some reward r_t.
- Can we recover it from knowing how often the expert visited each state, formalized as $\rho(s, a)$?

No, clearly we can't recover the scaling.

Actually, the problem is worse

Say there is only one state s.

$$\rho(s, a_1) = 0$$
$$\rho(s, a_2) = 1$$
$$\rho(s, a_3) = 0$$
$$\rho(s, a_4) = 0$$

All we can deduce is:

$$R(s, a_1) \leq R(s, a_2)$$
$$R(s, a_3) \leq R(s, a_2)$$
$$R(s, a_4) \leq R(s, a_2)$$

- Assume expert is maximizing some reward r_t.
- Can we recover it from knowing how often the expert visited each state, formalized as $\rho(s, a)$?

It's not just scaling we can't recover.

OK, so we can’t reconstruct expert reward.

Instead, make a reward leading to behavior that matches expert.

Reward Structure in Apprenticeship Learning

Assumption: true (expert) reward function has linear structure

$$\boxed{R(s) = w^\top \phi(s) \text{ with } \|w\|_2 \leq 1}$$

$$V(s) = \mathrm{E}_{\tau\sim\pi}\left[\sum_t w^\top \phi(s_t)\right] = w^\top \mathrm{E}_{\tau\sim\pi}\left[\sum_t \phi(s_t)\right]$$

Linearity is in the features!

Each feature $\phi(s)$ can be a nonlinear function of state.

Apprenticeship Learning Algorithm

Make intrinsic rewards so that
the expert does well and the existing polices do poorly.

Interleave training policy
and constructing reward.

Call RL solver maximizing these intrinsic rewards.
The resulting policy will be closer to the expert.

Apprenticeship Learning Algorithm

Make intrinsic rewards:

$$\mathrm{w}_t^\star = \mathrm{argmax}_{\mathrm{w}:\,\|w\|_2\le 1} \min_j w^\top \left(\mathrm{E}_{\tau\sim\pi_E}\left[\sum_t \phi(s_t)\right] - \mathrm{E}_{\tau\sim\pi_j}\left[\sum_t \phi(s_t)\right]\right)$$

Feature Matching.

Interleave training policy and constructing reward.

Call RL solver with rewards, obtain π_t.

$$\mathrm{R}_t(\mathrm{s}) = {\mathrm{w}_t^\star}^\top \phi(s)$$

Matching Features

$$|J_E - J_S| = \left| \mathrm{E}_{\tau\sim\pi_E}\left[\sum_t w^\top \phi(s_t)\right] - \mathrm{E}_{\tau\sim\pi}\left[\sum_t w^\top \phi(s_t)\right] \right|$$

Cauchy-Schwartz

$$\leq \|w\|_2 \left\| \mathrm{E}_{\tau\sim\pi_E}\left[\sum_t \phi(s_t)\right] - \mathrm{E}_{\tau\sim\pi}\left[\sum_t \phi(s_t)\right] \right\|_2$$

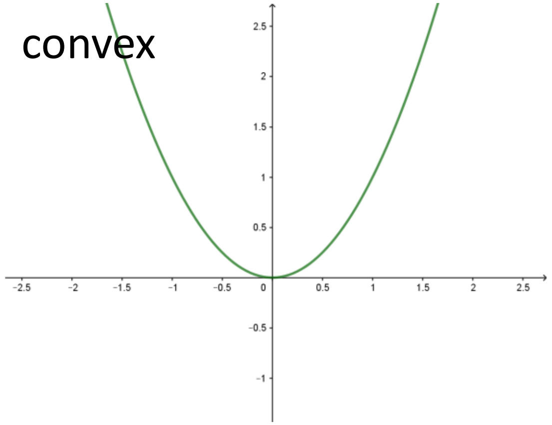
Assumption on $\|w\|_2$

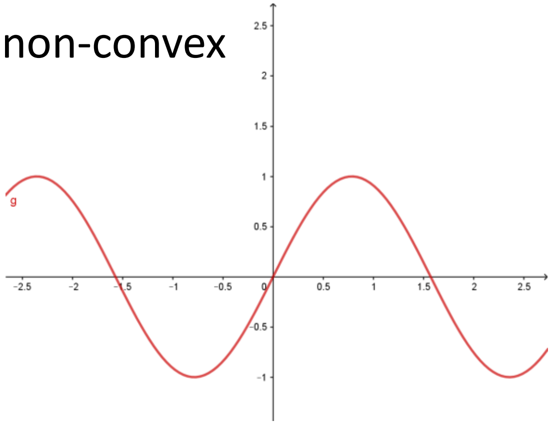
$$\leq \underbrace{\left\| \mathrm{E}_{\tau\sim\pi_E}\left[\sum_t \phi(s_t)\right] - \mathrm{E}_{\tau\sim\pi}\left[\sum_t \phi(s_t)\right] \right\|_2}$$

At the end of the iteration we have a π that makes this term small!

Apprenticeship Learning: Summary

- Algorithm can be implemented only using convex optimization





- If you already have good linear features, this is the thing to try.
- Precursor of more modern imitation learning algorithms, which are based on deep learning.

Question Time!

Imitation Learning

Tutorial in the SMILES Summer School
21 October 2020

Part 2

Kamil Ciosek

Adversarial Imitation Learning

Imitation Learning by Minimizing Divergences.

Another view at imitation learning

Consider an MDP.

- Expert produces some distribution over state-action pairs $\rho_E(s,a)$
- Our imitation learning policy $\hat{\pi}_\theta$ also produces some distribution over state-action pairs $\rho_{\hat{\pi}_\theta}(s,a)$.
- Let's tune the parameters θ of $\hat{\pi}$ to make the divergence between these two distributions small.

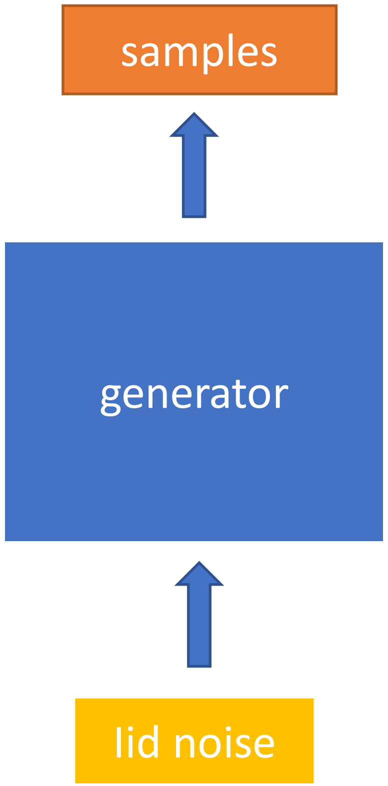
$$\boxed{\operatorname{argmin}_\theta D(\rho_E, \rho_{\hat{\pi}_\theta})}$$

Hmm, haven't I seen it somewhere?

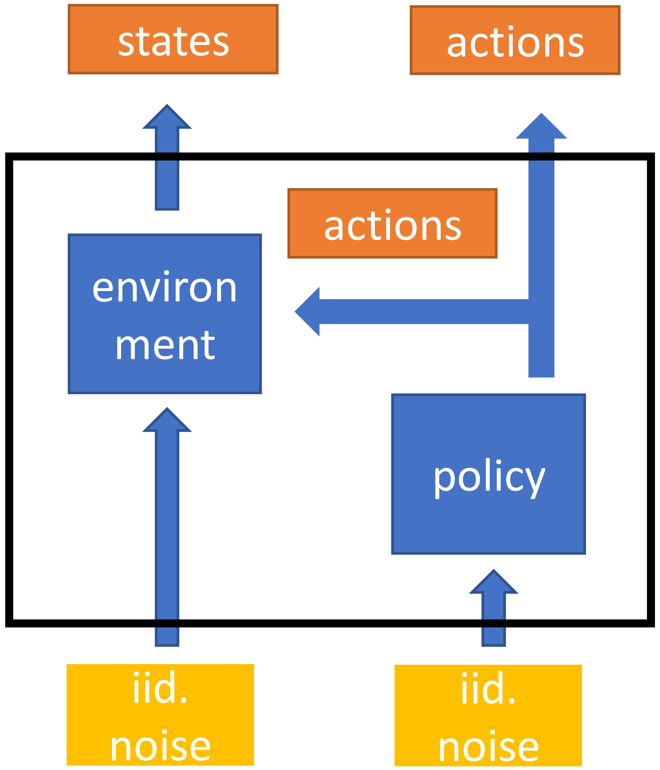
Is adversarial imitation learning a GAN?

Generators in regular GAN vs in Adversarial IL

Generator in a regular GAN



Generator in Adversarial IL



samples
aren’t iid.

Discriminator

Same in a regular GAN and in adversarial IL

Math Recap: JS-Divergence

How do we compare probability distributions ρ_1, ρ_2?

$$D_{KL}(\rho_1, \rho_2) = \sum_z \rho_1(s,a) \log \frac{\rho_1(s,a)}{\rho_2(s,a)}$$

$$\boxed{D_{JS}(\rho_1, \rho_2) = \frac{1}{2}\left(D_{KL}\left(\rho_1, \frac{\rho_1 + \rho_2}{2}\right) + D_{KL}\left(\rho_2, \frac{\rho_1 + \rho_2}{2}\right)\right)}$$

Rewriting JS-Divergence as optimization

How do we compare probability distributions ρ_1, ρ_2?

$$D_{JS}(\rho_1, \rho_2) = \frac{1}{2}\left(D_{KL}\left(\rho_1, \frac{\rho_1 + \rho_2}{2}\right) + D_{KL}\left(\rho_2, \frac{\rho_1 + \rho_2}{2}\right)\right)$$

is equivalent to

$$D_{JS}(\rho_S, \rho_E) \;=\; \min_{\mathrm{w}} \mathrm{E}_{(s,a)\sim\rho_S}\left[\log(\mathrm{D_w}(\mathrm{s},\mathrm{a}))\right] + \mathrm{E}_{(s,a)\sim\rho_E}\left[\log(1 - \mathrm{D_w}(\mathrm{s},\mathrm{a}))\right] + \mathrm{const}$$

Source: I. Goodfellow, et al.: *Generative Adversarial Networks*

Training GAN discriminators

$$D_{JS}(\rho_S, \rho_E) = \min_{\mathrm{w}} \mathrm{E}_{(s,a)\sim\rho_S} \underbrace{[\log(\mathrm{D_w}(\mathrm{s},\mathrm{a}))]}_{\text{log probability of simulator (“fake”) data}} + \mathrm{E}_{(s,a)\sim\rho_E} \underbrace{[\log(1 - \mathrm{D_w}(\mathrm{s},\mathrm{a}))]}_{\text{log probability of expert (“real”) data}} + \text{const}$$

Find a discriminator weights $w^{\star}$ given the generator.

Make sure state-action pairs used for training discriminator are independent(ish)

Training the policy (generator)

$$D_{JS}(\rho_1, \rho_2) \; = \; \mathrm{E}_{(s,a)\,\sim\rho_S}\,[\log(\mathrm{D_{w^\star}}(\mathrm{s,a}))] + \underbrace{\mathrm{E}_{(s,a)\,\sim\rho_E}\,[\log(1 \; - \; \mathrm{D_{w^\star}}(\mathrm{s,a}))] + \text{const}}_{\text{Terms that don't depend on the policy.}}$$

Given $w^\star$, improve the generator

Fake (aka intrinsic) reward

$$D_{JS}(\rho_1, \rho_2) = \underbrace{\mathrm{E}_{(s,a)\sim\rho_S}\left[\log(\mathrm{D}_{\mathrm{w}^\star}(\mathrm{s},\mathrm{a}))\right]}_{\text{Average-reward RL formulation.}} + \text{const'}$$

Compare with $J_\theta = \lim_{H\to\infty} E_\tau\left[\frac{1}{H}\sum_{t=1}^{H} r_t\right] = \mathrm{E}_{(s,a)\sim\rho_S}\left[R(\mathrm{s},\mathrm{a})\right]$

Just do RL with reward R(s,a) = $\log(\mathrm{D}_{\mathrm{w}^\star}(\mathrm{s},\mathrm{a}))$!

Adversarial Imitation Learning Algorithm

Train discriminator by minimizing

$$D_{JS}(\rho_S, \rho_E) = \min_{\mathrm{w}} \mathrm{E}_{(s,a)\sim\rho_S}[\log(\mathrm{D_w}(\mathrm{s,a}))] + \mathrm{E}_{(s,a)\sim\rho_E}[\log(1 - \mathrm{D_w}(\mathrm{s,a}))] + \text{const}$$

Interleave training policy and discriminator.

Train generator by calling RL solver with rewards:

$$\mathrm{R(s,a)} = \log(\mathrm{D_{w^\star}(s,a)}).$$

Adversarial Imitation Learning Algorithm

Train discriminator by minimizing

$$D_{JS}(\rho_S, \rho_E) = \min_{\mathrm{w}} \mathrm{E}_{(s,a)\sim\rho_S}[\log(\mathrm{D_w}(\mathrm{s,a}))] + \mathrm{E}_{(s,a)\sim\rho_E}[\log(1 - \mathrm{D_w}(\mathrm{s,a}))] + \text{const}$$

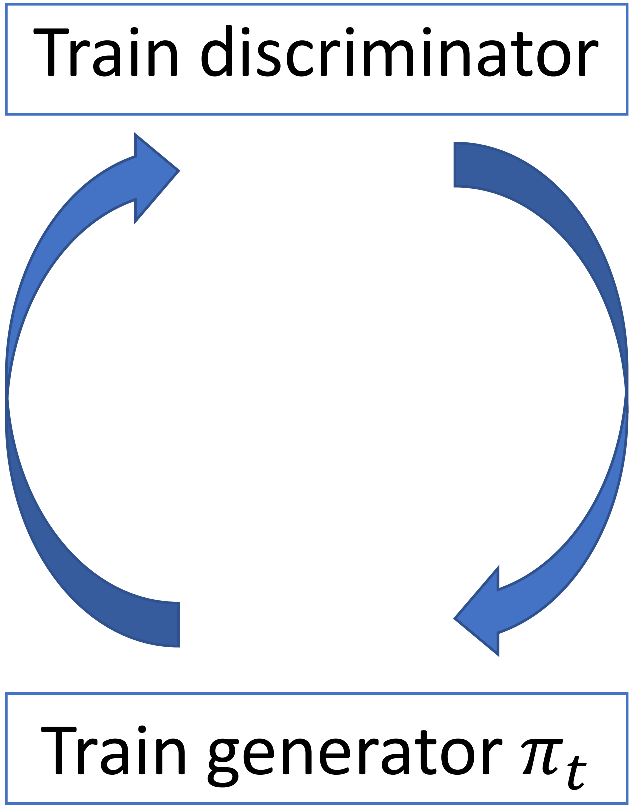
Interleave training policy and discriminator.

Does this converge?

Train generator by calling RL solver with rewards:

$$\mathrm{R(s,a)} = \log(\mathrm{D_{w^\star}(s,a)}).$$

Adversarial Imitation Learning Algorithm



- In general, for any distribution over the expert data:
  - Train for $T + T'$ timesteps.
  - Use a mixture of π_ts for t $= T, T+1, T+2, T+T'$.
  - We can optionally distill the mixture into a single policy.
- Under assumptions on expert, we can just take the policy from last iteration. This is very often the case in practice.

Why call it adversarial

- We change the policy to **maximize** total expected reward.
- We change the discriminator (reward) to compute JS divergence (**minimization**).

Interleaving max & min is known as a minmax problem (or game)

Not same as having an adversary in the environment trying to prevent agent from winning.

Link to Inverse Reinforcement Learning

The idea of getting a reward from expert data isn't specific to GAIL!

Train generator by calling RL solver with rewards:

$$\mathrm{R}(\mathrm{s},\mathrm{a}) = \log(\mathrm{D}_{\mathrm{w}^\star}(\mathrm{s},\mathrm{a})).$$

Imitation Learning From States Only

When we introduced GAIL, we starts out with a divergence $D_{JS}(\rho_1, \rho_2)$ between $\rho_1(s, a)$ and $\rho_2(s, a)$.

Instead, minimize divergence between pairs of state-successor state $\rho_1(s, s')$ and $\rho_2(s, s')$.

Algorithm is (almost) the same!

Like learning to drive without seeing when driver moves steering wheel & presses down pedals!

Isn’t GAIL overkill?

- Generative Adversarial Imitation Learning is hard to implement:
  - GANs are hard to stabilize
  - Actor-Critic methods (the most common way to make an RL algorithm) are also hard !
  - GAIL is a combination of two hard problems!
- Do we really need GAIL for imitation?

It is often not necessary.

Digression: Other Divergences

- We started with $D_{JS}(\rho_1, \rho_2)$, as in the original GAN paper.
- Any divergence where $D(\rho_1, \rho_2) = 0$ implies $\rho_1 = \rho_2$ could be used.
- Choice of divergence is somewhat arbitrary. The main problem is if we can optimize it well.
- For imitation learning, there isn't a broad study about which divergence works best.

Comparison of divergences in GANs:

f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization: S. Nowozin, B. Cseke, R. Tomioka

Digression: BC vs divergence minimization

$$D_{KL}(\rho_1, \rho_2) = \sum_z \rho_1(z) \log \frac{\rho_1(z)}{\rho_2(z)} = \sum_z \rho_1(z) \log \rho_1(z) - \sum_z \rho_1(z) \log \rho_2(z)$$

$$\theta^\star = \mathrm{argmax}_\theta \, \mathrm{E}_{\tau \sim P_E}[\log P_\theta(\tau)]$$
$$= \mathrm{argmin}_\theta \, \mathrm{KL}(P_E, P_\theta)$$

Maximum Likelihood minimizes KL divergence

Digression: BC vs divergence minimization

Is BC an adversarial IL algorithm?

No! P_E , P_θ are distributions over trajectories.

$$\theta^\star = \mathrm{argmax}_\theta \, \mathrm{E}_{\tau \sim P_E}[\log P_\theta(\tau)]$$
$$= \mathrm{argmin}_\theta \, \mathrm{KL}(P_E , P_\theta)$$

Maximum Likelihood minimizes KL divergence

Digression: BC vs divergence minimization

- BC minimizes KL between $P_E(\tau)$ and $P_\theta\ (\tau)$.
- BC **does not** minimize KL divergence between $P_E(s, a)$ and $P_\theta(s, a)$.
- **BC is not the same** as adversarial IL with KL instead of JS divergence – if we did that, we would get an imitation learning with a KL-GAN.

While BC minimizes a divergence,
it is not an adversarial IL algorithm
in the sense GAIL is.

Summary & Next Steps

We compare IL algorithms and give a short outline of other IL research.

Comparison of Algorithms

	Ease of Implementation	Weak Points
Behavioral Cloning	very easy (supervised learning)	needs **lots** of data
Apprenticeship Learning	easy (call convex optimizer)	needs reward features
Generative Adversarial Imitation Learning	hard (GAN + actor-critic)	complexity of GAN training

Practical Tip

BC often works well (with enough data).
If you are implementing IL, try BC first.
Worst case is you just use it to initialize your policy.

Other problems in imitation learning

A taster of other research on imitation learning:

- If we can ask the expert for more data, what data do we ask for?
- Can we do better than the expert under additional assumptions?
- Can we do imitation learning is systems with partial observability?
- Can we do imitation learning where several agents have to communicate to solve a goal.
- Can we still do IL if we test in an MDP different (but still similar) than the one expert used?

Reading Recommendations on IL

“Modern classics” on IL:

- *Efficient Reductions for Imitation Learning*: S. Ross, J. A. Bagnell
- *Apprenticeship learning via inverse reinforcement learning*: P. Abbeel, A. Y. Ng.
- *Generative Adversarial Imitation Learning:* J. Ho, S. Ermon

How to learn this in practice?

- Start with a small problem.
  - Pick a task with a short horizon (<10).
  - Get expert data by training an RL algorithm.
- Implement Behavioral Cloning.
  - See how BC performance changes as you vary the amount of expert data.
- Implement apprenticeship learning with features learned by BC.
- Implement Generative Adversarial Imitation Learning:
  - First, make sure your code can compute a divergence between two fixed distributions.
  - Second, hook it to an RL algorithm you know works.

Before we finish – Project Paidia

- Project Paidia is making computer games even more fun!
- Thanks to all my colleagues in at Microsoft Research.
- See link below for more information on how we apply RL/IL at Microsoft!

Dave Bignell
Research SDE II

Kamil Ciosek
Senior Researcher

Sam Devlin
Senior Researcher

Raluca Georgescu
Research SDE II

Katja Hofmann
Principal Researcher

Mikhail Jacob
Researcher

Oliver Kilian
Software Engineer

Robert Loftin
Researcher

Adrian O'Grady
Principal Research Engineer

Jaroslaw Rzepecki
Senior Research Engineer

https://www.microsoft.com/en-us/research/project/project-paidia/

Microsoft

Question Time!