Inverse Reinforcement Learning

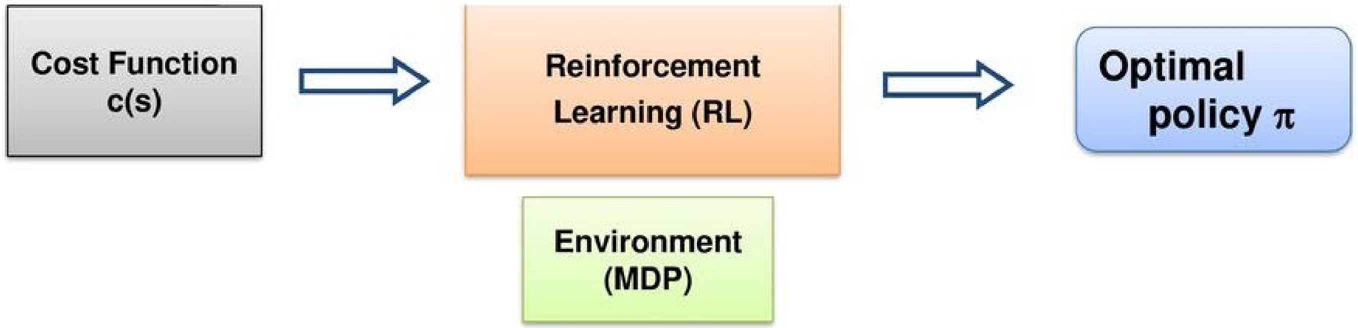
Introduction, theory and applications

Inverse Reinforcement Learning

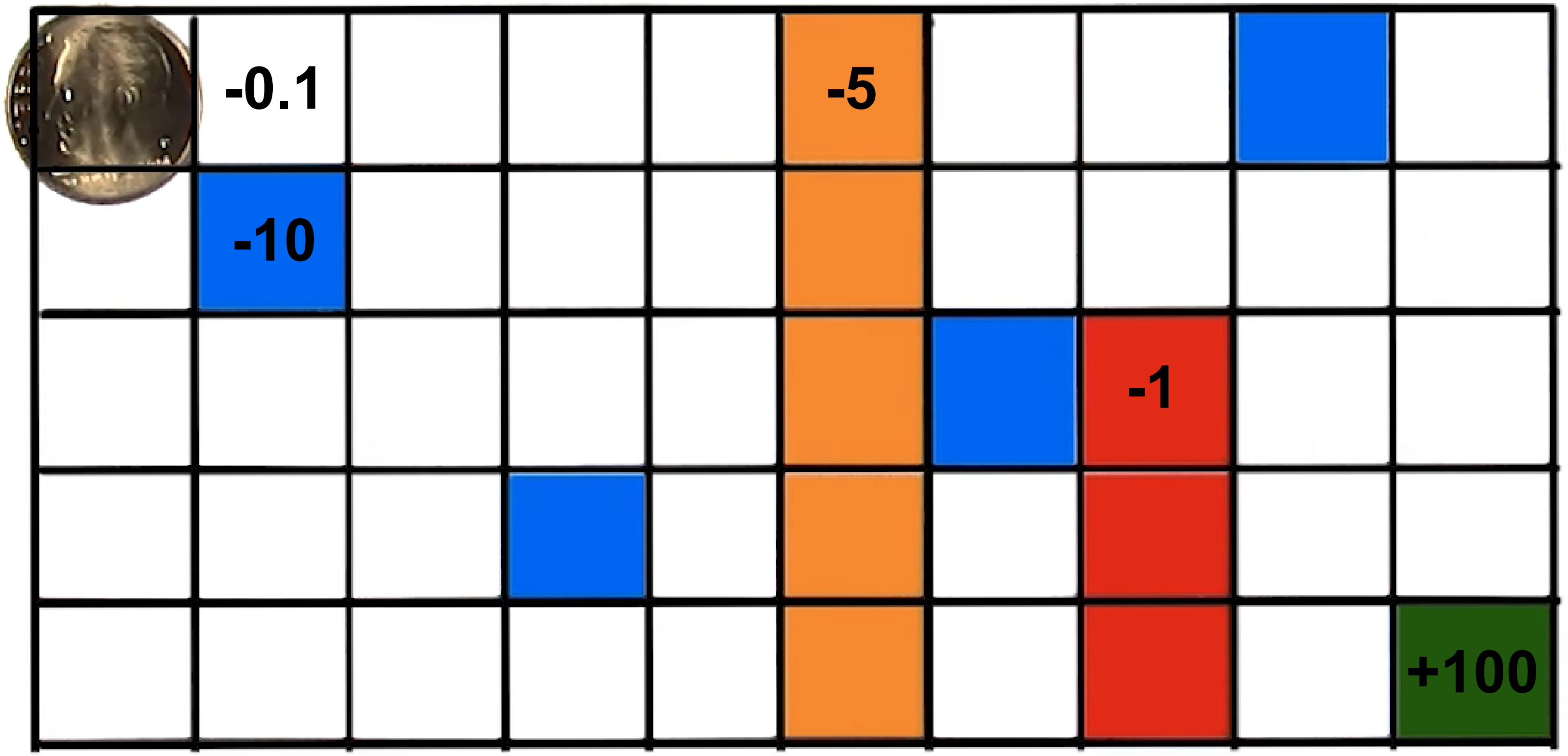
- Goal
- Problems of the formulation
- Approaches
  - Linear programming / SVM
  - Bayesian Learning
  - Maximum Entropy approach
  - Adversarial Learning
- Applications

Inverse Reinforcement Learning

- Idea
  - RL consists in Learning policy from reward function

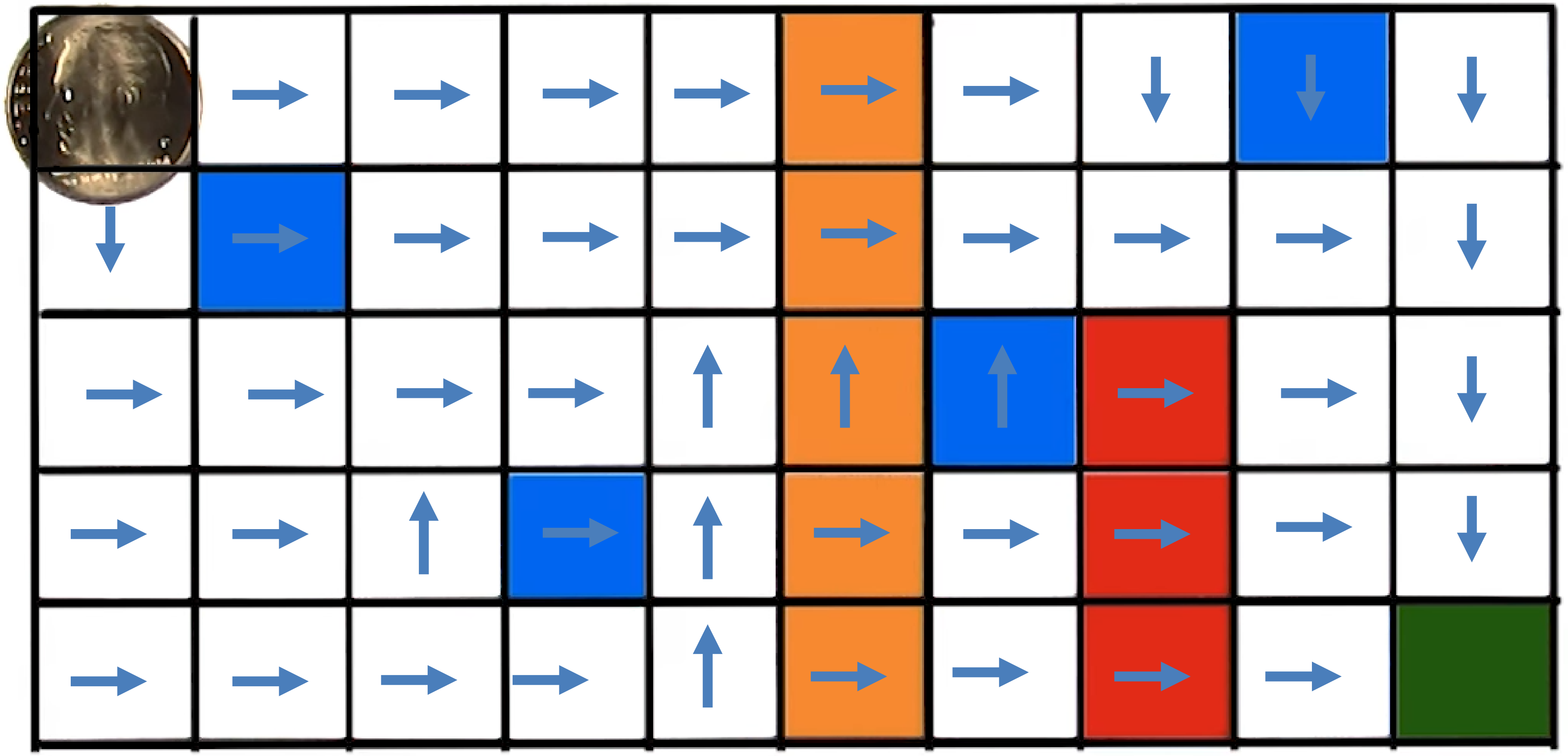


RL summary



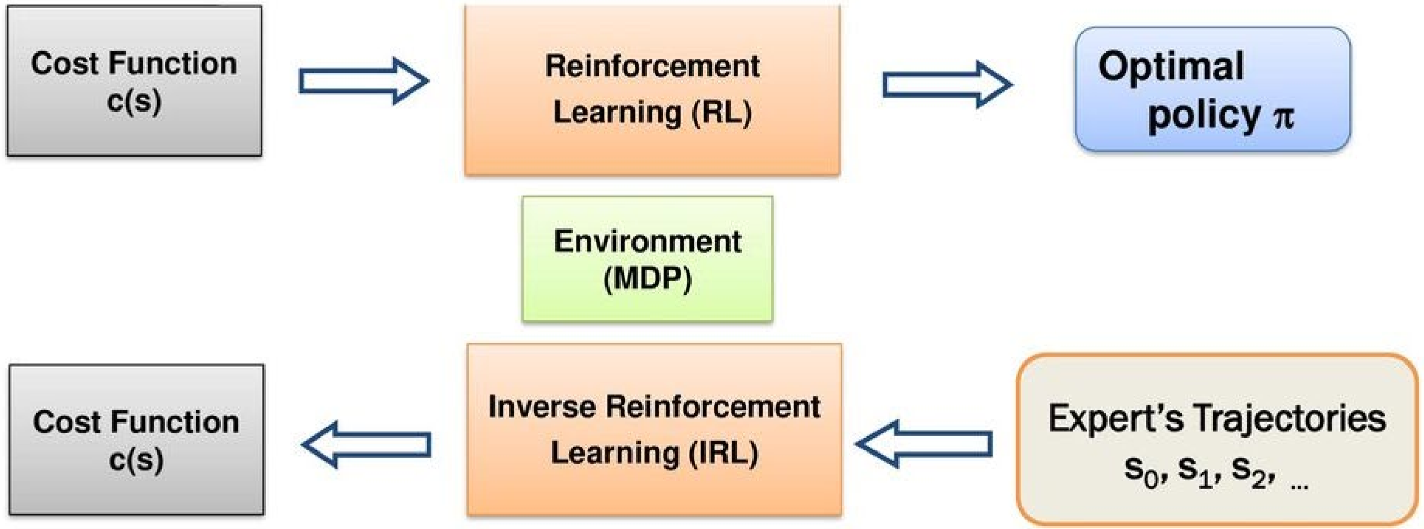
MDP:
<S, A, T, R>

RL summary

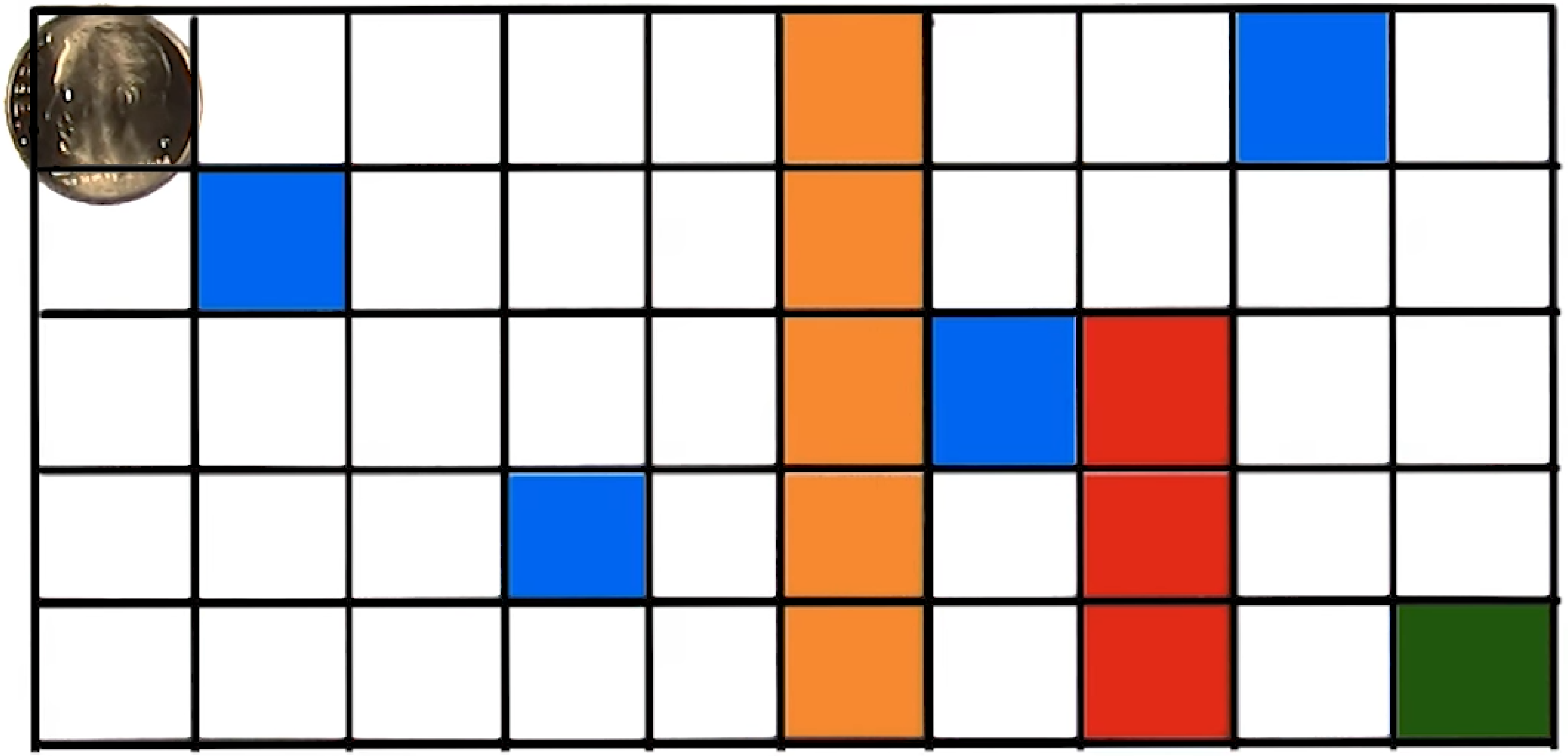


Inverse Reinforcement Learning

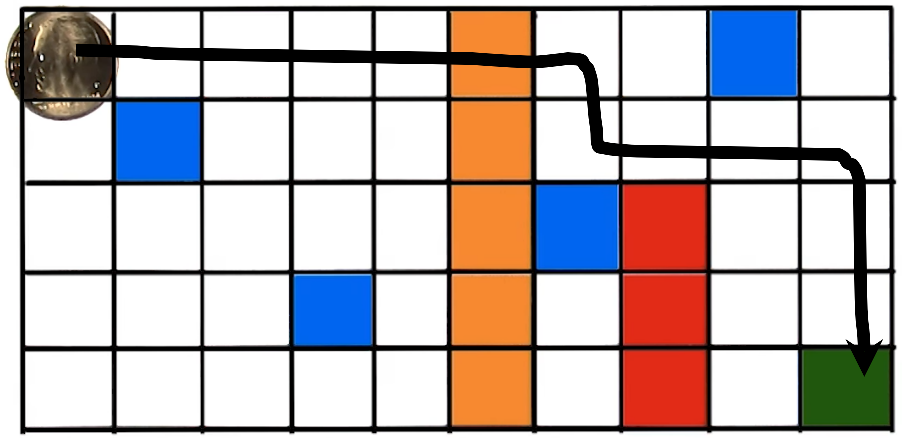
- Idea
  - RL consists in Learning policy from reward function
  - *Inverse* RL consists in learning the reward function from the policy
  - … or at least, examples of the policy



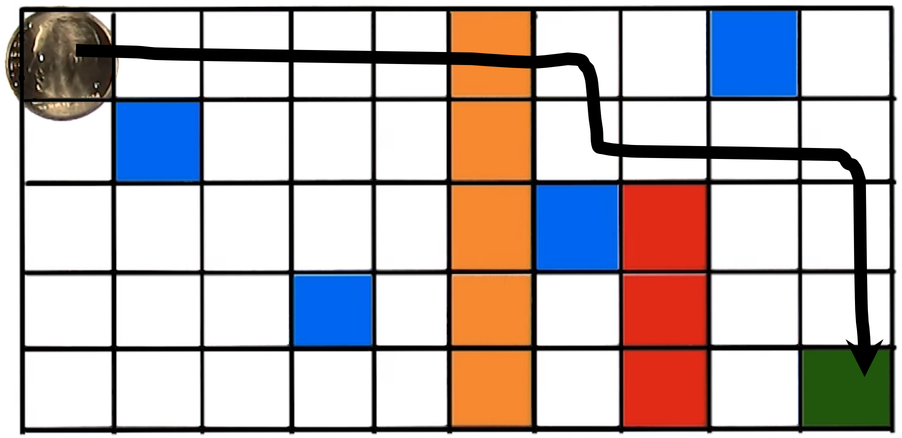
IRL example

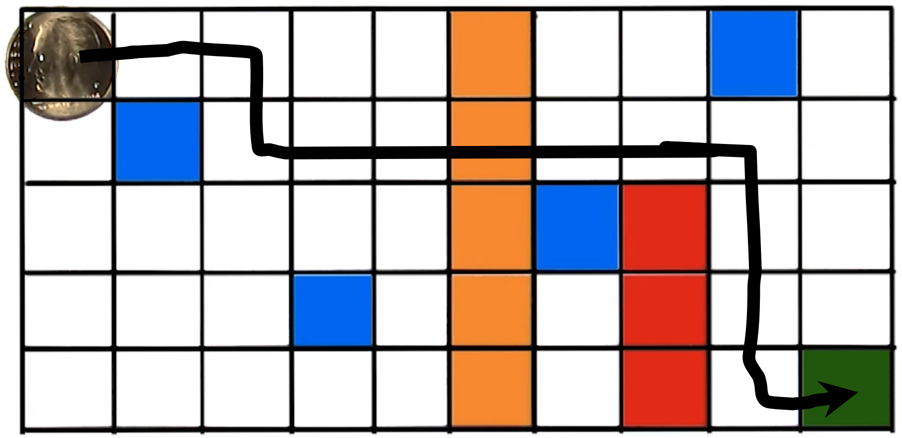


IRL example

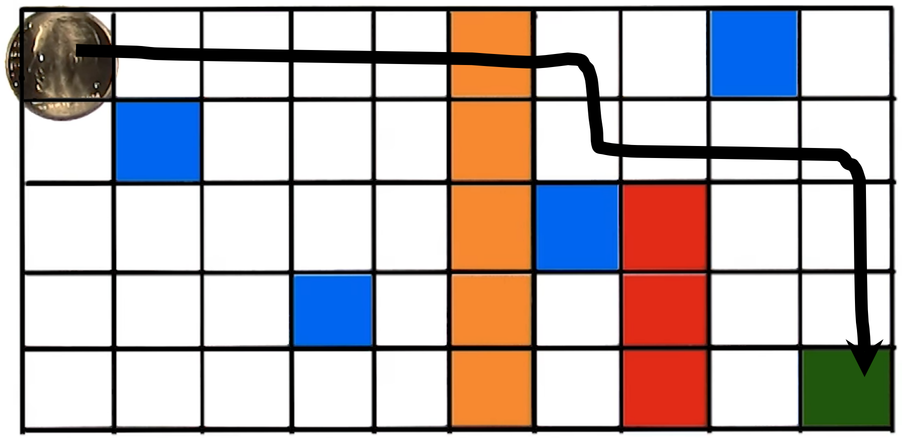


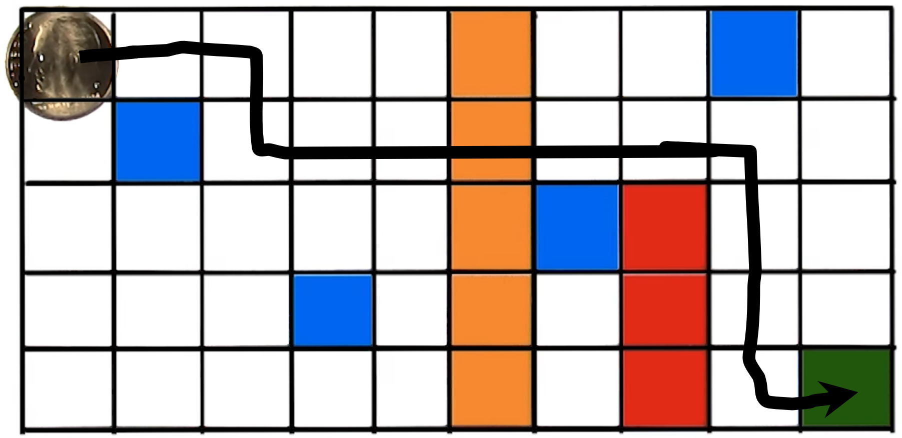
IRL example

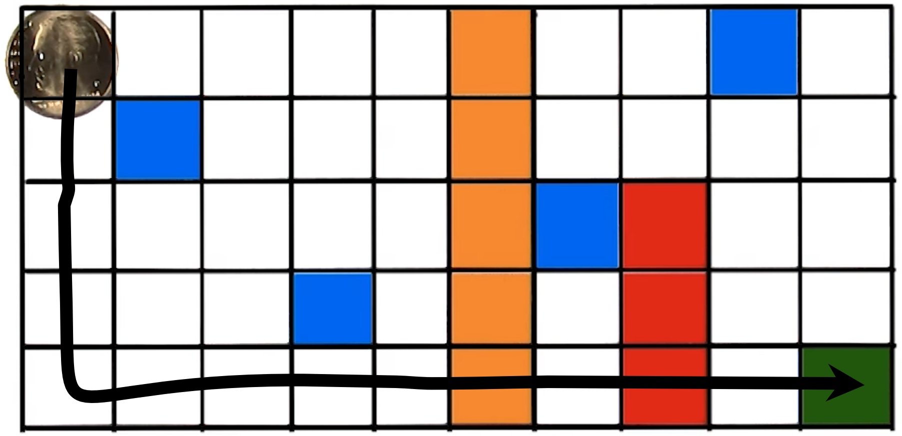




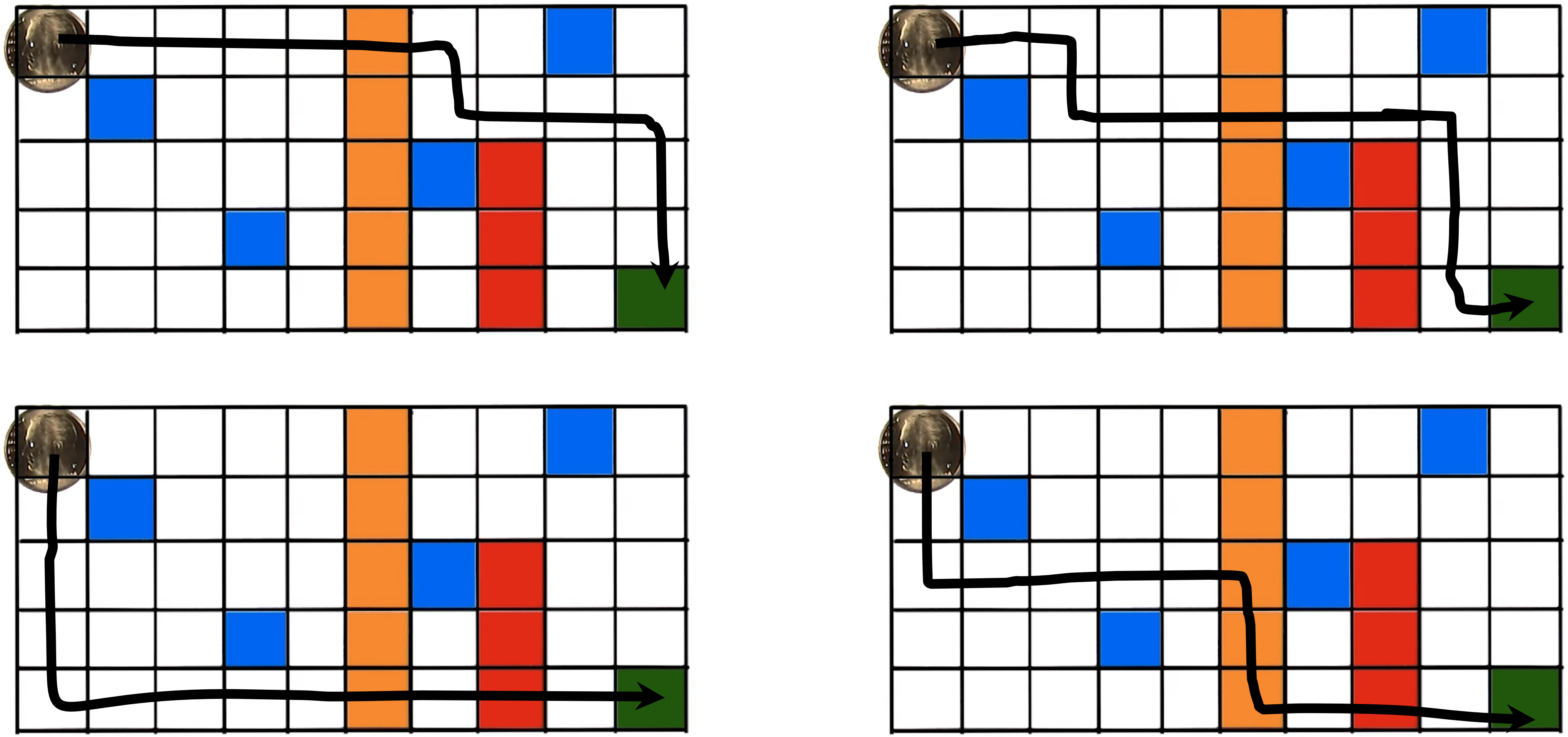
IRL example







IRL example



Uses of IRL

- Learn from examples (imitation learning)
  - Behavior Cloning
    - Drift, not robust, do not generalize well to unseen data
    - Dataset Aggregation: DAGGER (Roos et. Alt. 11)

Uses of IRL

- Learn from examples (imitation learning)
  - Behavior Cloning
    - Drift, not robust, do not generalize well to unseen data
    - Dataset Aggregation: DAGGER (Roos et. Alt. 11)

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.

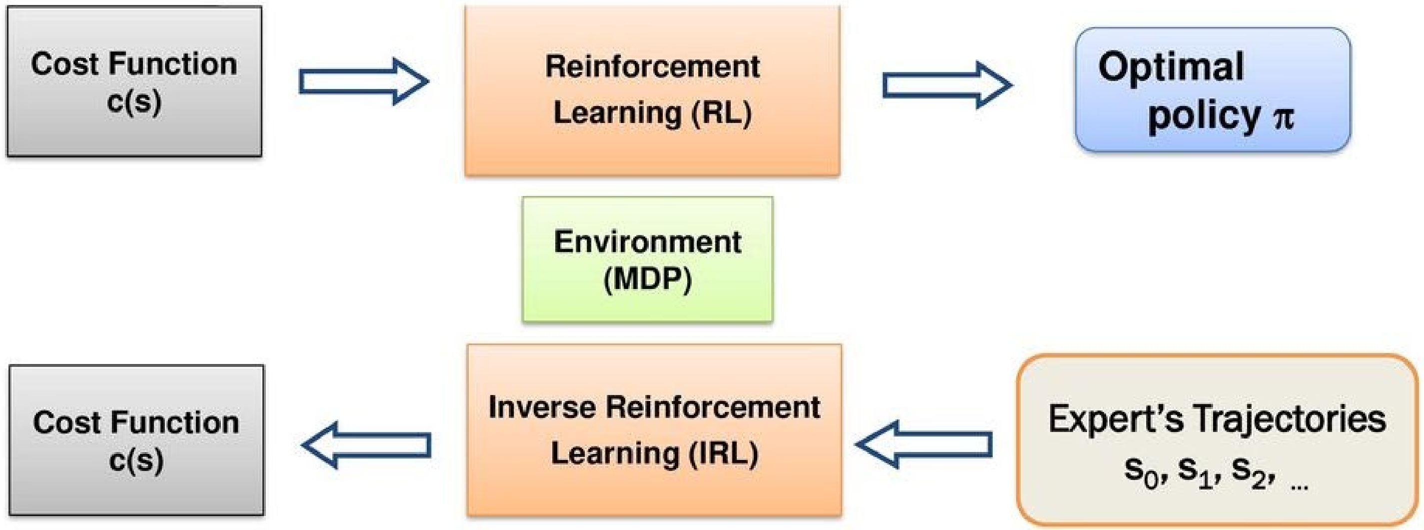
Algorithm 3.1: DAGGER Algorithm.

Uses of IRL: (1) Learn from examples

- Learn from examples (imitation learning)
  - Behavior Cloning
    - Drift, not robust, do not generalize well to unseen data
    - Dataset Aggregation: DAGGER (Roos et. Alt. 11)
  - IRL
    - Learn policy from derived R function from examples [helicopter acrobatics]

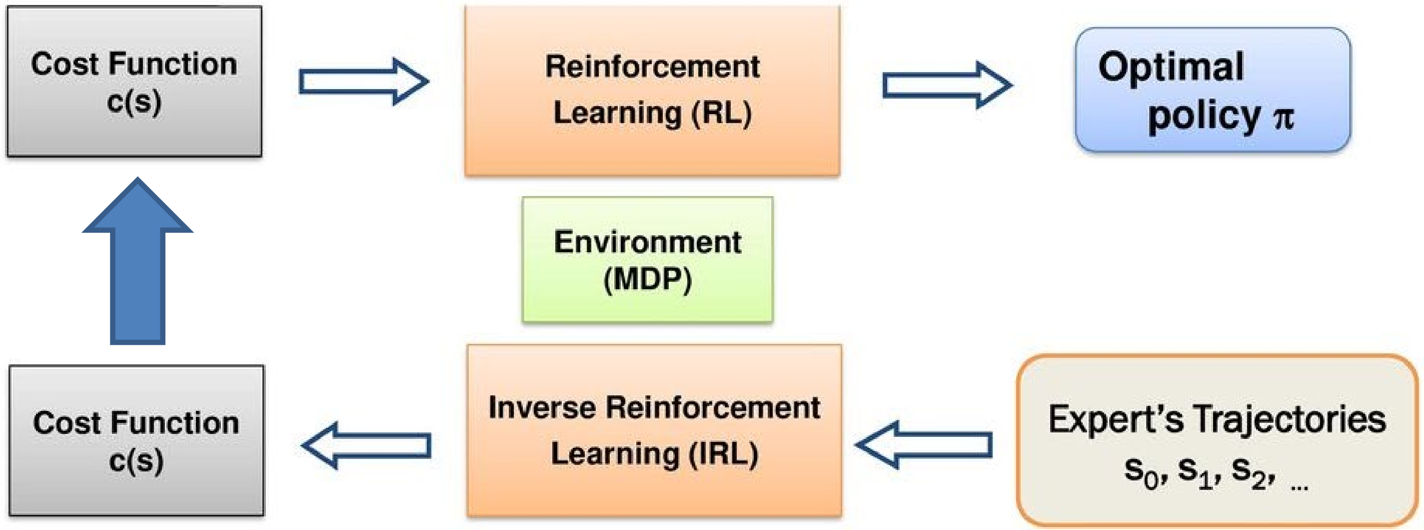
Uses of IRL: (1) Learn from examples

- Learn from examples (imitation learning)
  - Behavior Cloning
    - Drift, not robust, do not generalize well to unseen data
    - Dataset Aggregation: DAGGER (Roos et. Alt. 11)
  - IRL
    - Learn policy from derived R function from examples [helicopter acrobacies]



Uses of IRL: (1) Learn from examples

- Learn from examples (imitation learning)
  - Behavior Cloning
    - Drift, not robust, do not generalize well to unseen data
    - Dataset Aggregation: DAGGER (Roos et. Alt. 11)
  - IRL
    - Learn policy from derived R function from examples [helicopter acrobacies]



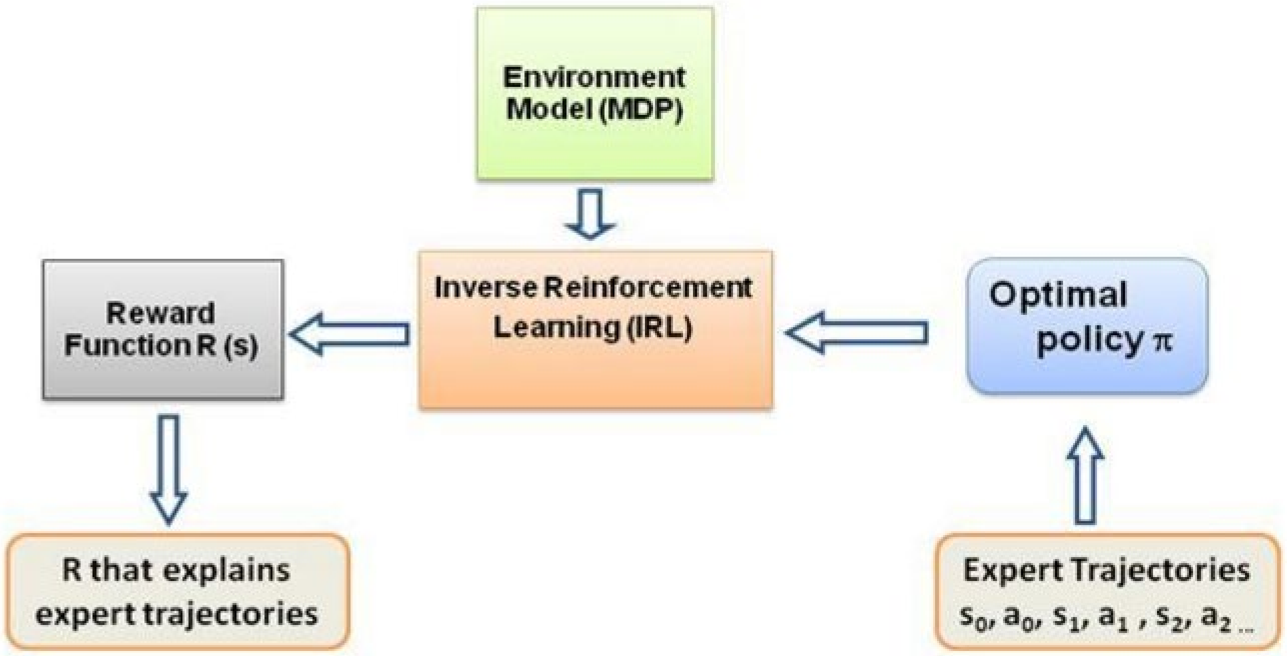
Uses of IRL: (2) Learn a reward fuction

- We assume that reward function is easy to design but
  - Difficult to know: Drive behavior (subgoals to balance)
  - Sometimes tricky and/or surprising: Cobra example
  - [See how agents cheat in AI]

- So, use IRL to learn a reward function when it is difficult to define

Uses of IRL: (3) Predict intents of agents

- Useful to model rational behavior and to deduce intents of agents (predict behavior)



Predicting behavior

Activity forecasting
(Kitani 2012)

Fig. 1. Given a single pedestrian detection, our proposed approach forecasts plausible paths and destinations from noisy vision-input

state = location

action = movement direction

environment = visual scene

Predicting wide receiver trajectories in American football
(Lee 2016)

Advantages

- Generalize better than BC (R acts as a *regularizer*)
- Reward function introduces rationality to BC
- Reward function is a brief and better description of behavior.
- It should be easier to learn a policy from a reward function than from another policy
- Applications to:
  - Animal behavior
  - Multi-agent framework
  - …

Inverse Reinforcement Learning

Formulation and problems

Formalization

- Given MDP (except R), and π
  - π used to generate examples of the policy (sometimes with examples we have enough)
- But problem not well defined mathematically: many different possible reward functions under which observed behavior is optimal
  - Constant reward for all states explains any policy (degenerate solution)
    - [Because only + examples... But this is common in RL]
  - Multiple of the reward function (R and 5R) explain same behavior
  - Shaping
- Some solutions:
  - Regularization (entropy)
  - Sparse solutions
  - Some kind of normalization

More practical problems

- How to evaluate a learned reward function?
  - We can only compare indirectly with optimal trajectories from it
  - … but it is very costly for very large problems since it requires to solve a RL problem at each iteration of the algorithm
- Moreover… Source of examples?
  - Should the policy be optimal to generate examples?
  - Humans are not following always same behavior to solve one problem (multimodal solutions)
  - One human not always consistent
  - Do they incorporate probability of successful trajectories?

Approaches

- Some solutions to IRL
  - Linear programming approximation
  - Quadratic programming
  - Bayesian approach
  - Probabilistic and Maximum Entropy methods
  - GANs

Inverse Reinforcement Learning

Approaches to solve IRL

Approaches

- Some solutions to IRL
  - **Linear programming approximation**
  - Quadratic programming
  - Bayesian approach
  - Probabilistic and Maximum Entropy methods
  - GANs

Linear programming

LP approach (Ng & Russel, 2000)

First algorithm for simple cases:

- From definition of optimality of the policy:

$$\forall s \in \mathcal{S}, \forall a \in \mathcal{A} : \sum_{s'} T(s'|s,\pi(s))V^{\pi}(s') \geq \sum_{s'} T(s'|s,a)V^{\pi}(s')$$

$$\Leftrightarrow \forall \mathbf{T}^{i} \in \mathbf{T}^{\neg\pi} : \mathbf{T}^{\pi}\mathbf{V}^{\pi} \succeq \mathbf{T}^{i}\mathbf{V}^{\pi}$$

Linear programming

(Ng & Russel, 2000)

- We know, from Bellman eqs. in RL, we can solve V using LP

$$\begin{aligned} \mathbf{V}^\pi &= \mathbf{R} + \gamma \mathbf{T}^\pi \mathbf{V}^\pi \\ \Leftrightarrow \mathbf{V}^\pi - \gamma \mathbf{T}^\pi \mathbf{V}^\pi &= \mathbf{R} \\ \Leftrightarrow (\mathbf{I} - \gamma \mathbf{T}^\pi)\mathbf{V}^\pi &= \mathbf{R} \\ \Leftrightarrow \mathbf{V}^\pi &= (\mathbf{I} - \gamma \mathbf{T}^\pi)^{-1}\mathbf{R} \end{aligned}$$

- Replacing in constraint:

$$\forall \mathbf{T}^i \in \mathbf{T}^{\neg\pi} : \mathbf{T}^\pi \mathbf{V}^\pi \succeq \mathbf{T}^i \mathbf{V}^\pi \Leftrightarrow \forall \mathbf{T}^i \in \mathbf{T}^{\neg\pi} : \mathbf{T}^\pi (\mathbf{I} - \gamma \mathbf{T}^\pi)^{-1}\mathbf{R} \succeq \mathbf{T}^i (\mathbf{I} - \gamma \mathbf{T}^\pi)^{-1}\mathbf{R}$$

$$\Leftrightarrow \forall \mathbf{T}^i \in \mathbf{T}^{\neg\pi} : \mathbf{T}^\pi (\mathbf{I} - \gamma \mathbf{T}^\pi)^{-1}\mathbf{R} - \mathbf{T}^i (\mathbf{I} - \gamma \mathbf{T}^\pi)^{-1}\mathbf{R} \succeq 0 \Leftrightarrow \forall \mathbf{T}^i \in \mathbf{T}^{\neg\pi} : (\mathbf{T}^\pi - \mathbf{T}^i)(\mathbf{I} - \gamma \mathbf{T}^\pi)^{-1}\mathbf{R} \succeq 0$$

Linear programming

(Ng & Russel, 2000)

- Problems:
  - IRL is ill-posed: **Any** R function that fulfill these constraints is valid (in particular R=0)
  - We typically only observe expert traces rather than the entire expert policy
  - Assumes the expert is optimal
  - How to find R? Assumes we can enumerate all policies

Linear programming

(Ng & Russel, 2000)

- Heuristics to get rid of degenerate solutions
- **First approach**: Maximize dif. between best and second best action at the same time

$$\text{maximize}: \sum_{s \in \mathcal{S}} \left(Q^{\pi}(s, \pi(s)) - \max_{a \in \mathcal{A} \setminus \pi(s)} Q^{\pi}(s, a) \right)$$

1. Reward functions with small rewards are more natural and should be preferred:
   a. Regularization:

   $$-\lambda ||\hat{\mathbf{R}}||_1$$

   b. Hard limit on max R

   $$\forall s \in \mathcal{S} : |\hat{\mathcal{R}}(s)| \leq R_{\max}$$

Linear programming

(Ng & Russel, 2000)

Second algorithm: Large state spaces -> need for function approximation

- Not for all states but in **sample of states**... and to generalize to other states
- ... assume **R can be expressed as a linear comb**. of feature vector (α)

$$\hat{\mathcal{R}}_\theta(s) = \hat{\alpha}_1 \cdot \phi_1(s) + \ldots + \hat{\alpha}_d \cdot \phi_d(s)$$

Linear programming

(Ng & Russel, 2000)

- So, use LP to solve this problem:

$$\text{maximize} \sum_{s \in \mathcal{S}_0} \min_{a \in \mathcal{A} \setminus \pi(s)} p(\mathbb{E}_{s' \sim T(s'|s,\pi(s))}[V^\pi(s')] - \mathbb{E}_{s' \sim T(s'|s,a)}[V^\pi(s')])$$

$$\text{such that } |\alpha_i| \leq 1, i = 1, \ldots d$$

  - Basically the first condition in previous alg. for regularization
  - p is function to penalize constraint violation when negative argument (x2)

Linear programming

(Ng & Russel, 2000)

Third algorithm: No policy available always (extract info from trajectories)

- Compute values of *m* trajectories

$$\text{maximize} \sum_{i=1}^{m} p(\hat{V}(\zeta_{\pi^*}) - \hat{V}(\zeta_{\pi^i}))$$

$$\text{such that } |\alpha_i| \leq 1, i = 1, \ldots, d$$

- where

$$\hat{\mathcal{R}}_\theta(s) = \hat{\alpha}_1 \cdot \phi_1(s) + \ldots + \hat{\alpha}_d \cdot \phi_d(s)$$

$$\hat{V}_i(\zeta) = \sum_{s_t \in \zeta} \gamma^t \mathcal{R}_i(s_t) = \sum_{s_t \in \zeta} \gamma^t \alpha_i \phi_i(s_t) = \alpha_i \sum_{s_t \in \zeta} \gamma^t \phi_i(s_t) = \alpha_i \hat{\mu}_i(\zeta)$$

$$\hat{V}(\zeta) = \hat{V}_1(\zeta) + \ldots + \hat{V}_d(\zeta) = \hat{\alpha}_1 \hat{\mu}_1(\zeta) + \ldots + \hat{\alpha}_d \hat{\mu}_d(\zeta)$$

Linear programming (Ng & Russel, 2000)

Third algorithm: No policy available always (extract info from trajectories)

- Compute values of *m* trajectories

$$\text{maximize} \sum_{i=1}^{m} p(\hat{V}(\zeta_{\pi^*}) - \hat{V}(\zeta_{\pi^i}))$$

$$\text{such that } |\alpha_i| \leq 1, i = 1, \ldots, d$$

- where

$$\hat{\mathcal{R}}_\theta(s) = \hat{\alpha}_1 \cdot \phi_1(s) + \ldots + \hat{\alpha}_d \cdot \phi_d(s)$$

Feature counts

$$\hat{V}_i(\zeta) = \sum_{s_t \in \zeta} \gamma^t \mathcal{R}_i(s_t) = \sum_{s_t \in \zeta} \gamma^t \alpha_i \phi_i(s_t) = \alpha_i \boxed{\sum_{s_t \in \zeta} \gamma^t \phi_i(s_t)} = \alpha_i \boxed{\hat{\mu}_i(\zeta)}$$

$$\hat{V}(\zeta) = \hat{V}_1(\zeta) + \ldots + \hat{V}_d(\zeta) = \hat{\alpha}_1 \hat{\mu}_1(\zeta) + \ldots + \hat{\alpha}_d \hat{\mu}_d(\zeta)$$

Linear programming

(Ng & Russel, 2000)

Third algorithm: No policy available always (extract info from trajectories)

- Compute values of trajectories

$$\text{maximize} \sum_{i=1}^{m} p(\hat{V}(\zeta_{\pi^*}) - \hat{V}(\zeta_{\pi^i}))$$

$$\text{such that } |\alpha_i| \leq 1, i = 1, \ldots, d$$

- Find R (α) with values for expert trajectories better than values for trajectories from any another policy.
  - Start with random π, find R following above equation using LP, learn π for that R and repeat (π becomes competitive) [include in comparison also older π]

Linear programming

(Ng & Russel, 2000)

Third algorithm: No policy available always (extract info from trajectories)

- Compute values of trajectories

$$\text{maximize} \sum_{i=1}^{m} p(\hat{V}(\zeta_{\pi^*}) - \hat{V}(\zeta_{\pi^i}))$$

$$\text{such that } |\alpha_i| \leq 1, i = 1, \ldots, d$$

- Find R (α) with values for expert trajectories better than values for trajectories from any another policy.
  - Start with random π, find R following above equation using LP, learn π for that R and repeat (π becomes competitive) [include in comparison also older π]
- You get both: policy and reward function.

Approaches

- Some solutions to IRL
  - Linear programming approximation
  - **Quadratic programming**
  - Bayesian approach
  - Probabilistic and Maximum Entropy methods
  - GANs

Apprenticeship Learning via IRL (SVM)

- Assume again R as linear comb of feature vector

$$\hat{\mathcal{R}}_\theta(s) = \hat{\alpha}_1 \cdot \phi_1(s) + \ldots + \hat{\alpha}_d \cdot \phi_d(s)$$

- **Feature counts** (features should appear in trajectories of learnt policy like in *D* trajectories)

$$V^\pi = E\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t) \,|\pi\right] = E\left[\sum_{t=0}^{\infty} \alpha\, \gamma^t \phi(s_t) \,|\pi\right] = \alpha\, E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \,|\pi\right] = \alpha\, \mu(\pi) \in \mathbb{R}^k$$

- Solve using **SVM** instead of LP (Abbeel & Ng 04)

Apprenticeship Learning via IRL (SVM)

- Optimize:

$$\min_w \|w\|_2^2 + C\xi$$

$$\text{s.t. } w^\top \mu(\pi^*) \geq w^\top \mu(\pi) + m(\pi^*, \pi) - \xi$$

- Where:
  - m is difference between policies (f.i. Hamming distance)
  - w are parameters of linear parametrization of reward function
  - π is current policy (or set of previous policies)
  - Slacks are to allow errors like in SVMs
- Start with random policy. Iterate: Compute features counts, find optimal w (so R), learn policy. Iterate until small enough changes

Approaches

- Some solutions to IRL
  - Linear programming approximation
  - Quadratic programming
  - **Bayesian approach**
  - Probabilistic and Maximum Entropy methods
  - GANs

Bayesian IRL (BIRL)

- BIRL (Ramachandran & Amir 07)
  - Probabilistic def of *P(a|s,R)* and of set (*D*) of trajectories *P(D|R)*.

$$\Pr(a|s,\hat{\mathcal{R}}) = \frac{\exp\left[\alpha \cdot \hat{Q}^*(s,a)\right]}{Z} \qquad \Pr(\mathcal{D}|\hat{\mathcal{R}}) = \frac{\exp\left[\alpha \cdot \left(\sum_{\zeta\in\mathcal{D}} \sum_{(s,a)\in\zeta} \hat{Q}^*(s,a)\right)\right]}{Z}$$

  - Use Bayes to find *P(R|D)*

$$\Pr(\hat{\mathcal{R}}|\mathcal{D}) = \frac{\Pr(\mathcal{D}|\hat{\mathcal{R}}) \cdot \Pr(\hat{\mathcal{R}})}{\Pr(\mathcal{D})}$$

  - Need *P(D)* but intractable ->use of MCMC
  - Apply MAP (Maximum a posteriori) to find R
- Robust BIRL
  - Faces the problem of suboptimality of some actions

Approaches

- Some solutions to IRL
  - Linear programming approximation
  - Quadratic programming
  - Bayesian approach
  - **Probabilistic and Maximum Entropy methods**
  - GANs

Entropy methods

- Maximizing log-likelihood of trajectories while satisfying the constraint of feature expectation matching

$$\max_{P} \quad -\sum_{\zeta} P(\zeta) \log P(\zeta)$$

Traj. generated by π_θ

$$\text{s.t.} \sum_{\zeta} P(\zeta)\mu(\zeta) = \mu(\pi^*)$$

As random as possible while matching features

- Solution under maximum entropy criteria is with form:

$$\Pr(\zeta|\theta) = \frac{e^{\mathcal{R}(\zeta)}}{\sum_{\tau} e^{\mathcal{R}(\tau)}} \propto e^{\mathcal{R}(\zeta)}$$

- Does **not assume optimality** of expert trajectories

Entropy methods:

- Let's maximize the Log-likelihood of trajectories

$$
\begin{aligned}
\theta^* &= \mathrm{argmax}_\theta \, \mathcal{L}(\theta) \\
&= \mathrm{argmax}_\theta \log \prod_{\zeta_d \in \mathcal{D}} P\left(\zeta_d | \theta, T\right) \\
&= \mathrm{argmax}_\theta \sum_{\zeta_d \in \mathcal{D}} \log P\left(\zeta_d | \theta, T\right) \\
&\approx \mathrm{argmax}_\theta \sum_{\zeta_d \in \mathcal{D}} \log \left(\frac{e^{\mathcal{R}_\theta(\zeta_d)}}{Z(\theta, T)} \prod_{s_t, a_t, s_{t+1} \in \zeta_d} P_T\left(s_{t+1} | s_t, a_t\right) \right)
\end{aligned}
$$

Entropy methods:

$$= \operatorname{argmax}_\theta \sum_{\zeta_d \in \mathcal{D}} \log \left(\frac{e^{\mathcal{R}_\theta(\zeta_d)}}{Z(\theta, T)} \prod_{s_t, a_t, s_{t+1} \in \zeta_d} P_T\left(s_{t+1} | s_t, a_t\right) \right)$$

$$= \arg\max_\theta \sum_{\zeta_d \in \mathcal{D}} \left(\mathcal{R}_\theta\left(\zeta_d\right) - \log(Z(\theta, T))\right) + \sum_{s_t, a_t, s_{t+1} \in \zeta_d} \log\left(P_T\left(s_{t+1} | s_t, a_t\right)\right)$$

$$= \arg\max_\theta \frac{1}{N} \sum_{\zeta_d \in \mathcal{D}} \mathcal{R}_\theta\left(\zeta_d\right) - \log(Z(\theta, T))$$

$$= \arg\max_\theta \frac{1}{N} \sum_{\zeta_d \in \mathcal{D}} \mathcal{R}_\theta\left(\zeta_d\right) - \log\left(\sum_\zeta e^{\mathcal{R}_\theta(\zeta)} \prod_{s_t, a_t, s_{t+1} \in \zeta} P_T\left(s_{t+1} | s_t, a_t\right) \right)$$

$$= \arg\max_\theta \frac{1}{N} \sum_{\zeta_d \in \mathcal{D}} \mathcal{R}_\theta\left(\zeta_d\right) - \log\left(\sum_\zeta e^{\mathcal{R}_\theta(\zeta)} \right) - \sum_{s_t, a_t, s_{t+1} \in \zeta} \log\left(P_T\left(s_{t+1} | s_t, a_t\right)\right)$$

$$= \arg\max_\theta \frac{1}{N} \sum_{\zeta_d \in \mathcal{D}} \mathcal{R}_\theta\left(\zeta_d\right) - \log\left(\sum_\zeta e^{\mathcal{R}_\theta(\zeta)} \right)$$

Entropy methods idea

$$= \arg\max_{\theta} \frac{1}{|D|} \sum_{\zeta_d \in \mathcal{D}} \mathcal{R}_\theta\left(\zeta_d\right) - \log\left(\sum_{\zeta} e^{\mathcal{R}_\theta(\zeta)}\right)$$

Entropy methods idea

$$= \arg\max_{\theta} \frac{1}{|D|} \sum_{\zeta_d \in \mathcal{D}} \mathcal{R}_{\theta}(\zeta_d) - \log\left(\sum_{\zeta} e^{\mathcal{R}_{\theta}(\zeta)}\right)$$

Av. reward of
expert traject.

Entropy methods idea

$$= \arg\max_{\theta} \frac{1}{|D|} \sum_{\zeta_d \in \mathcal{D}} \mathcal{R}_\theta\left(\zeta_d\right) - \log\left(\sum_{\zeta} e^{\mathcal{R}_\theta(\zeta)}\right)$$

Av. reward of expert traject.

Soft-max reward

Entropy methods idea

$$= \arg\max_{\theta} \frac{1}{|D|} \sum_{\zeta_d \in \mathcal{D}} \mathcal{R}_{\theta}(\zeta_d) - \log\left(\sum_{\zeta} e^{\mathcal{R}_{\theta}(\zeta)}\right)$$

Av. reward of expert traject.

Soft-max reward

- maximizing the difference of expert trajectory rewards and the reward of best possible trajectory, ensuring that expert demonstrations will achieve near-optimal reward when the objective is maximized
- **But how to estimate the second term?**

Entropy methods: First way [MAXENT (Ziebartet et al. 2008)]

- Let's apply gradients:

$$\begin{aligned}
\nabla_\theta \mathcal{L} &= \frac{1}{|\mathcal{D}|}\sum_{\zeta_d \in \mathcal{D}} \left(\frac{\partial \mathcal{R}_\theta\left(\zeta_d\right)}{\partial \theta}\right) - \frac{1}{\sum_\zeta e^{\mathcal{R}_\theta(\zeta)}} \sum_\zeta e_\theta^{\mathcal{R}}(\zeta) \frac{\partial \mathcal{R}_\theta(\zeta)}{\partial \theta} \\
&= \frac{1}{|\mathcal{D}|}\sum_{\zeta_d \in \mathcal{D}} \left(\frac{\partial \mathcal{R}_\theta\left(\zeta_d\right)}{\partial \theta}\right) - \sum_\zeta \frac{e_\theta^{\mathcal{R}}(\zeta)}{\sum_\zeta e^{\mathcal{R}_\theta(\zeta)}} \frac{\partial \mathcal{R}_\theta(\zeta)}{\partial \theta} \\
&= \frac{1}{|\mathcal{D}|}\sum_{\zeta_d \in \mathcal{D}} \left(\frac{\partial \mathcal{R}_\theta\left(\zeta_d\right)}{\partial \theta}\right) - \sum_\zeta P(\zeta|\theta, T) \frac{\partial \mathcal{R}_\theta(\zeta)}{\partial \theta} \\
&= \frac{1}{|\mathcal{D}|}\sum_{\zeta_d \in \mathcal{D}} \left(\frac{\partial \mathcal{R}_\theta\left(\zeta_d\right)}{\partial \theta}\right) - \sum_s P(s|\theta, T) \frac{\partial r_\theta(s)}{\partial \theta}
\end{aligned}$$

Entropy methods: First way [MAXENT (Ziebartet et al. 2008)]

- Let's apply gradients:

$$\begin{aligned}\nabla_\theta \mathcal{L} &= \frac{1}{|\mathcal{D}|}\sum_{\zeta_d \in \mathcal{D}}\left(\frac{\partial \mathcal{R}_\theta(\zeta_d)}{\partial \theta}\right) - \frac{1}{\sum_\zeta e^{\mathcal{R}_\theta(\zeta)}}\sum_\zeta e_\theta^{\mathcal{R}}(\zeta)\frac{\partial \mathcal{R}_\theta(\zeta)}{\partial \theta} \\ &= \frac{1}{|\mathcal{D}|}\sum_{\zeta_d \in \mathcal{D}}\left(\frac{\partial \mathcal{R}_\theta(\zeta_d)}{\partial \theta}\right) - \sum_\zeta \frac{e_\theta^{\mathcal{R}}(\zeta)}{\sum_\zeta e^{\mathcal{R}_\theta(\zeta)}}\frac{\partial \mathcal{R}_\theta(\zeta)}{\partial \theta} \\ &= \frac{1}{|\mathcal{D}|}\sum_{\zeta_d \in \mathcal{D}}\left(\frac{\partial \mathcal{R}_\theta(\zeta_d)}{\partial \theta}\right) - \sum_\zeta P(\zeta|\theta, T)\frac{\partial \mathcal{R}_\theta(\zeta)}{\partial \theta} \\ &= \frac{1}{|\mathcal{D}|}\sum_{\zeta_d \in \mathcal{D}}\left(\frac{\partial \mathcal{R}_\theta(\zeta_d)}{\partial \theta}\right) - \sum_s P(s|\theta, T)\frac{\partial r_\theta(s)}{\partial \theta}\end{aligned}$$

Visitation prob. of states under Reward (and so policy). DP computation possible

Entropy methods: First way [MAXENT (Ziebartet et al. 2008)]

- Let's apply gradients:

$$\nabla_\theta \mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{\zeta_d \in \mathcal{D}} \left(\frac{\partial \mathcal{R}_\theta(\zeta_d)}{\partial \theta} \right) - \frac{1}{\sum_\zeta e^{\mathcal{R}_\theta(\zeta)}} \sum_\zeta e_\theta^{\mathcal{R}}(\zeta) \frac{\partial \mathcal{R}_\theta(\zeta)}{\partial \theta}$$

$$= \frac{1}{|\mathcal{D}|} \sum_{\zeta_d \in \mathcal{D}} \left(\frac{\partial \mathcal{R}_\theta(\zeta_d)}{\partial \theta} \right) - \sum_\zeta \frac{e_\theta^{\mathcal{R}}(\zeta)}{\sum_\zeta e^{\mathcal{R}_\theta(\zeta)}} \frac{\partial \mathcal{R}_\theta(\zeta)}{\partial \theta}$$

$$= \frac{1}{|\mathcal{D}|} \sum_{\zeta_d \in \mathcal{D}} \left(\frac{\partial \mathcal{R}_\theta(\zeta_d)}{\partial \theta} \right) - \sum_\zeta P(\zeta|\theta, T) \frac{\partial \mathcal{R}_\theta(\zeta)}{\partial \theta}$$

$$= \frac{1}{|\mathcal{D}|} \sum_{\zeta_d \in \mathcal{D}} \left(\frac{\partial \mathcal{R}_\theta(\zeta_d)}{\partial \theta} \right) - \sum_s P(s|\theta, T) \frac{\partial r_\theta(s)}{\partial \theta}$$

No need for linear assumption of reward (NN's) (Wulfmeier et al. 2016)

Visitation prob. of states under
Reward (and so policy).
DP computation possible

Entropy methods: First way [MAXENT (Ziebartet et al. 2008)]

- Algorithm

1. Initialize ψ, gather demonstrations $\mathcal{D}$
2. Solve for optimal policy $\pi(\mathbf{a}|\mathbf{s})$ w.r.t. reward r_ψ
3. Solve for state visitation frequencies $p(\mathbf{s}|\psi)$
4. Compute gradient $\nabla_\psi \mathcal{L} = -\frac{1}{|\mathcal{D}|} \sum_{\tau_d \in \mathcal{D}} \frac{dr_\psi}{d\psi}(\tau_d) - \sum_s p(s|\psi) \frac{dr_\psi}{d\psi}(s)$
5. Update ψ with one gradient step using $\nabla_\psi \mathcal{L}$

Entropy methods: Second way [GCL (Finn et al. 2016)]

- In general, visit counts cannot be computed in large stat spaces
- *Guided Cost Learning*: Use **importance sampling** to estimate Z

$$Z = \sum_{\zeta} \exp[\mathcal{R}(\zeta)] = \sum_{\zeta} \frac{\exp[\mathcal{R}(\zeta)]}{q(\zeta)} q(\zeta) = E_q\left[\frac{\exp[\mathcal{R}(\zeta)]}{q(\zeta)}\right]$$

Based on a sample set $\mathcal{D}_q$ of trajectories, the estimate then is

$$Z \simeq \frac{1}{|\mathcal{D}_q|} \sum_{\zeta} \frac{\exp[\mathcal{R}(\zeta)]}{q(\zeta)}$$

Entropy methods: Second way [GCL (Finn et al. 2016)]

- The proposal distribution *q* **should samples trajectories with high reward**, since these trajectories have the highest impact on the partition function (so optimized policy on current reward function).
- So now:

$$\log \mathcal{L}(\theta) = \left(\frac{1}{|\mathcal{D}|} \sum_{\zeta_e \in \mathcal{D}} \hat{\mathcal{R}}_\theta \left(\zeta_e \right) \right) - \log \left[\frac{1}{|\mathcal{D}_q|} \sum_{\zeta_q \in \mathcal{D}_q} \frac{\exp \left[\hat{\mathcal{R}}_\theta \left(\zeta_q \right) \right]}{q \left(\zeta_q \right)} \right]$$

- No need to solve the whole MDP (just approximation), so more efficient

GANs and GCL

(Finn et al. 2016b)

- It can be proved that GCL is equivalent to a GAN:

IRL	GAN
trajectory ζ	sample x
sample distribution q based on π_q	generator G
reward function $\mathcal{R}_\theta$	discriminator D
expert demonstrations $\mathcal{D}$	training set from true data distribution p

- Optimal point for discriminator:

$$D_\theta(\zeta) = \frac{\frac{1}{Z}\exp\left[\hat{\mathcal{R}}_\theta(\zeta)\right]}{\frac{1}{Z}\exp\left[\hat{\mathcal{R}}_\theta(\zeta)\right] + q(\zeta)}$$

Adversarial IRL

(Fu et al. 2017)

- Some problems with GCL:
  - You need whole trajectories
  - Entangled reward with actions (don't allow transfer learning)
- AIRL proposes:

$$D_\theta(s,a) = \frac{\exp\left[A^{\mathrm{soft}}(s,a)\right]}{\exp\left[A^{\mathrm{soft}}(s,a)\right] + \pi_q(a|s)}$$
$$= \frac{\exp\left[\mathcal{R}_\theta(s) + \gamma \cdot V_\psi^{\mathrm{soft}}(s') - V_\psi^{\mathrm{soft}}(s)\right]}{\exp\left[\mathcal{R}_\theta(s) + \gamma \cdot V_\psi^{\mathrm{soft}}(s') - V_\psi^{\mathrm{soft}}(s)\right] + \pi_q(a|s)}$$

- AIRL [example]()

Imitation Learning

- Closely related but different. They do not return the reward function, only policy
- Can also be learnt from adversarial networks (same idea)
  - GAIL
  - InfoGAIL
  - GAifO

- See DeepMimic video presentation for IL

MAP of methods