Apprenticeship Learning via Inverse Reinforcement Learning

Pieter Abbeel and Andrew Y. Ng [ICML 2004]

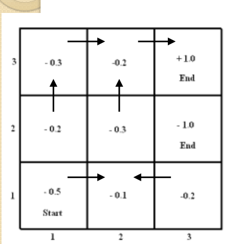
CS 6784
March 9, 2010
(Presented by Dane & Vasu)

Markov Decision Processes

- Used for modeling sequential decision problems.
- Set of states S
- Set of available actions A.
- Transition probabilities $P_{s,a}$
  - Give probabilities for arriving in a new state after performing action a while in state s.
- Reward functions $R(s)$
  - The 'value' of being in state s. Assume to be bounded in the absolute value by 1.

Example

Gridworld



- States given by grid cells
  - Additionally, specified start and end states
- At each cell, action is given by direction of movement
- Transition follows the specified action with 80% probability, else move to an adjacent cell randomly
- A transition to a given cell is accompanied by an immediate reward
- A policy maps each state to an action

Policies

- π gives a function from states to distributions over actions.
- The value of a policy is given by:

$$E_{s_0 \sim D}\left[V^{\pi}(s_0)\right] = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi\right]$$
$$= E\left[\sum_{t=0}^{\infty} \gamma^t w \cdot \phi(s_t) \mid \pi\right] = w \cdot \underbrace{E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi\right]}_{\mu(\pi)}$$

- D gives the distribution of starting states
- γ is a discount factor – earlier rewards are given more weight.

Computing Optimal Policies

- Reinforcement Learning
  - For instance, Q-learning
- Q: $S \times A \rightarrow \mathbb{R}$ is a function that gives the 'quality' of an action from a certain state
- The agent uses Q to explore the state space, and updates the function at each transition based on the experienced reward
- We know the reward function $R(s)$, but not the transition probabilities.

The Problem

- It is often difficult to specify the reward function, even if you are capable of making good decisions.
- E.g. You might be a perfectly good driver, but describing a reward function for good driving isn't so obvious.
- The solution: Apprenticeship learning.
  - Observe expert behavior, and assuming their actions to be optimal, derive the reward function.

Assumptions

- There is some feature vector over states
$$\phi : S \rightarrow [0,1]^k$$
- The unknown reward function $R^*(s)$ can be given by $w^{*T}\Phi(s)$ for some $w^* \in \mathbb{R}^k$, $\|w^*\|_1 \leq 1$

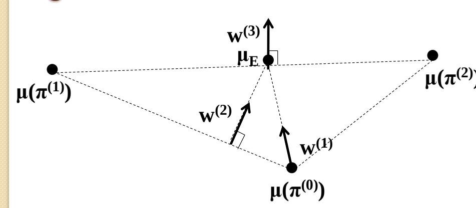
The Expert

- We have access to some expert policy π_E
- More accurately, we have examples of state sequences generated by said policy.
- We are also able to estimate the feature expectations μ_E
  - Given a set of m state sequences
$$\{s_0^{(i)}, s_1^{(i)}, \ldots\}_{i=1}^{m}$$
  - Calculate an estimate:
$$\hat{\mu}_E = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{\infty}\gamma^t\phi(s_t^{(i)})$$

Algorithm (max-margin)

- Given an MDP\R, a feature mapping Φ and the expert's feature expectations μ_E, find a policy whose performance is close to that of the expert's, on the unknown reward function $R^* = w^{*T}\Phi$.
- To accomplish this, we find a policy that induces feature expectations close to the expert policy.

Algorithm



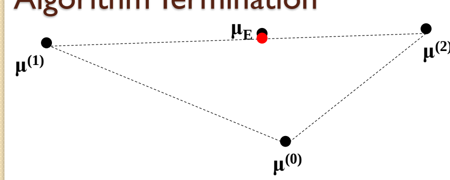
- Randomly pick some policy $\pi^{(0)}$, compute (or approx. via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$ and set $i = 1$
- Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{1,2,\ldots,(i-1)\}} w^T(\mu_E - \mu^{(j)})$ to get the argmax $w^{(i)}$
- If $t^{(i)} \leq \varepsilon$, then terminate.
- Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T\Phi$.
- Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
- Set $i = i + 1$, and go back to step 2.

Compare with Structural SVM

$$\arg\min_w \frac{1}{2}w.w + C\sum_i \xi_i \; s.t.$$
$$\forall i \forall j, w^T\Psi(x_i, y_i) \geq w^T\Psi(x_i, \hat{y}) + 1 - \frac{\xi_i}{\Delta(y_i, \hat{y})}$$

	SVM	**IRL (i^{th} iteration)**
Training examples:	$\{(x_1,y_1),\ldots,(x_n,y_n)\}$	$\{(\text{MDP}\backslash\text{R}, \mu_E)\}$
Constraints:	$\forall i \forall j, w^T\Psi(x_i, y_i) \geq w^T\Psi(x_i, \hat{y}) + 1 - \frac{\xi_i}{\Delta(y_i, \hat{y})}$	$\min_{j \in \{1,2,\ldots,(i-1)\}} w^T\mu_E \geq w^T\mu^{(j)} + \varepsilon$

Algorithm Termination

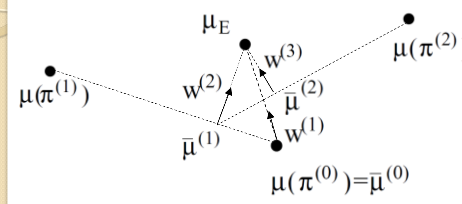


- Algorithm terminates with $t \leq \epsilon$. For any w (and in particular the expert's w_E) there is at least one $\pi^{(i)}$ whose performance under R is at least as good as the expert's performance minus ϵ
$$\forall w : \|w\|_2 \leq 1, \exists i : w^T\mu^{(i)} \geq w^T\mu_E - \varepsilon$$
- Ask the agent designer to manually test/examine the policies found by the algorithm, and pick one with acceptable performance.
- OR, solve: $\arg\min_\mu \|\mu_E - \mu\|_2 \; s.t. \; \mu = \sum_i \lambda_i\mu^{(i)}, \lambda_i \geq 0, \sum_i \lambda_i = 1$
- Note: the algorithm does not necessarily recover the underlying reward function correctly – it only (approximately) matches the feature expectations.

Projection Method

- Instead of keeping all prior feature expectations, just look at the most recent expectations, and an orthogonal projection of the expert expectations
  - Set $\bar{\mu}^{(i-1)} = \bar{\mu}^{(i-2)} + \frac{(\mu^{(i-1)}-\bar{\mu}^{(i-2)})^T(\mu_E-\bar{\mu}^{(i-2)})}{(\mu^{(i-1)}-\bar{\mu}^{(i-2)})^T(\mu^{(i-1)}-\bar{\mu}^{(i-2)})}(\mu^{(i-1)}-\bar{\mu}^{(i-2)})$ (This computes the orthogonal projection of μ_E onto the line through $\bar{\mu}^{(i-2)}$ and $\mu^{(i-1)}$.)
  - Set $w^{(i)} = \mu_E - \bar{\mu}^{(i-1)}$
- We no longer have to solve a QP, so no SVMs here.
  - In case you're just not an SVM kind of guy/gal.

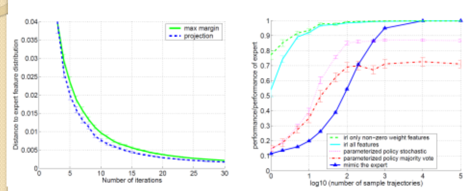
Another Nice Animation…



Terminate when the length of the projection falls below some threshold.

Experimental Results

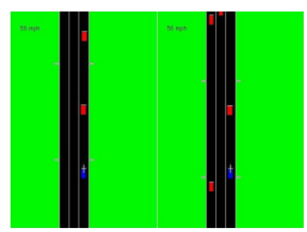
Gridworld:



- Projection method converges slightly faster than max-margin
- In general, IRL performs better than more naïve alternatives even with a small amount of training data

More Results

Driving task:



(Please turn off your cell phones during the movie)

Conclusions

- Assumed access to demonstrations by an expert maximizing a reward function linear in known features
  - (How reasonable is this? Quite, for rich feature spaces.)
- Algorithm based on inverse reinforcement learning
  - terminates in a small number of iterations
  - guarantees policy with performance comparable to or better than expert on the expert's unknown reward function (but without recovering the reward function!)
- Open problems:
  - non-linear reward functions
  - automatic feature construction and feature selection