**Reinforcement Learning** **Spring Semester, 2018/9**

# Lecture 13: June 12, 2019

*Lecturer: Yishay Mansour* *Scribe: ym*

Today lecture will have two different topics. The first is a different model for large state pace MDP, called *generative model*. The second is *Inverse Reinforcement Learning* (IRL) where we are given the optimal policy and try to build the reward function which is consistent with it.

## 13.1 Generative Model

A generative model is a different way to model a large state space MDP. Rather then assuming a succinct representation, we are given a "black box", called generator, that can simulate the model. The generator gets an inputs a state $s \in S$ and an action $a \in A$ and outputs a next state $s' \in S$ distributed according to $p(\cdot|s,a)$ and a reward $r$ distributed according to $R(s,a)$. We can also extend the model to describe a POMDP. For a POMDP, the inputs are a state $s \in S$ and an action $a \in A$ and outputs a next state $s' \in S$ distributed according to $p(\cdot|s,a)$, a reward $r$ distributed according to $R(s,a)$, and an observation $o$ distributed according to $O(\cdot|s',a)$.

### 13.1.1 Policy Evaluation

Recall that for policy evaluation, we are given a policy $\pi$ and would like to estimate its return from the start state, i.e., $V^{\pi}(s_0)$.

We will consider a discounted return, and hence we have an effective horizon $H = (1/(1-\gamma))\log(R_{max}/\epsilon)$ which guarantees that the expected return in the first $H$ steps is within $\epsilon$ of the true return. For this reason we can consider the *truncated discounted return* which is $E[\sum_{t=0}^{H} r_t]$.

Using the generative model we can generate a random trajectory of $\pi$ of length $H$. We start by applying the generative model on $(s_0, \pi(s_0))$ and observing $(r_0, s_1)$. At time $t$ we input the generative model $(s_t, \pi(s_t))$ observe $(r_t, s_{t+1})$, and terminate at time $H$. We have generated a trajectory $(s_0, \pi(s_0), r_0, s_1, \ldots, s_H, r_H, s_{H+1})$. The return of the trajectory is $v = \sum_{t=0}^{H} r_t$, and clearly this is an unbiased estimate of the true truncated discounted return, i.e., $E[v] = E[\sum_{t=1}^{H} r_t]$.

We can use the generative model in a similar way for a POMDP. The main issue here is that the policy gets as inputs only the observations and not the states. More specifically, we start by applying the generative model on $(s_0, \pi(s_0))$ observing $(r_0, o_1, s_1)$ and set the history to be $h_1 = o_1$. At time $t$ we input the generative model $(s_t, \pi(h_t))$ observe $(r_t, o_{t+1}, s_{t+1})$, and set $h_{t+1} = (h_t, o_{t+1})$. We terminate at time $H$. The return of the trajectory is $v = \sum_{t=0}^{H} r_t$, and clearly this is an unbiased estimate of the true truncated discounted return, i.e., $E[v] = E[\sum_{t=1}^{H} r_t]$. Note that the policy gets only the observations $o_t$ and not the states $s_t$.

We can use multiple estimates $v_i$ to get with high probability an accurate estimate. Namely, set $m = (V_{max}^2/\epsilon^2)\log(1/\delta)$, where $V_{max} = R_{max}/(1-\gamma)$. We run $m$ trajectories and for the $i$-th trajectory we have an estimate $v_i$. Our final estimate is $\hat{V}^{\pi}(s_0) = (1/m)\sum_{i=1}^{m} v_i$. This implies the following theorem:

**Theorem 13.1.** *With probability $1-\delta$, for $m = O((V_{max}^2/\epsilon^2)\log(1/\delta))$, we have that,*

$$|\hat{V}^{\pi}(s_0) - V^{\pi}(s_0)| \leq \epsilon$$

### 13.1.2 Trajectory tree

Our goal is to build a single data structure, that latter, given *any* policy $\pi$ we will be able to estimate $V^{\pi}(s_0)$. We will first build such a data structure for a single unbiased estimate, and then we will extend it for multiple estimates.

The data structure will be called *trajectory tree*, which will be a tree of depth $H$. The nodes will be labeled by states, and the edges by actions (and rewards). the process of creating a trajectory tree would be stochastic using the generative model.

The trajectory tree will be a complete directed tree of degree $A$. For each inner-node, the out edges are labeled by action $a \in A$ (each action labels one edge). Initially, we label the root by $s_0$. We use the generative model on input $(s_0, a)$, for each $a \in A$, and get $(s_{1,a}, r_{1,a})$. We label the $a$-th child by $s_{1,a}$ and the edge by $r_{1,a}$. We continue the process, where given a node labeled by $s_t$, we use the generative model on input $(s_t, a)$, for each $a \in A$, and get $(s_{t+1,a}, r_{t+1,a})$. We label the $a$-th child of $s_t$ by $s_{t+1,a}$ and the edge by $r_{t+1,a}$.

Given a policy $\pi$ we can generate an unbiased trajectory, by simply following the policy $\pi$ in the tree. Namely, we start at the root, $s_0$, continue to the child $\pi(s_0) \in A$, call it $s_1$, then continue to child $\pi(s_1)$, etc. The estimate for the return of $\pi$ is the sum of the rewards on the path it follows, i..e, $T(\pi) = \sum_{t=0}^{H} r_t$. Clearly, when we average over the trajectory trees $T$ we have an unbiased estimate (since we are generating the trajectories with the right probabilities). This implies that $E_T[T(\pi)] = V^{\pi}(s_0)$. Note that if we use the same tree $T$ for multiple polices $\pi_i$, we

get an unbiased estimate for each policy $\pi_i$, but the estimates of different policies $\pi_i$ and $\pi_j$ are correlated (due to using the same tree $T$).

We can extend the trajectory tree to multiple trajectory trees, and average the estimated returns. However, we will prefer to extend a single tree, and let it encode multiple trees. More specifically, given a parameter $m$, we create a complete tree of depth $H$ where the degree of each inner node is $m|A|$, where we have $m$ edges for each action $a \in A$. Otherwise we create the trajectory tree the same way as before.

### 13.1.3 Policy Evaluation: Finite hypothesis class

Assume we have a finite hypothesis class $\Pi = \{\pi : S \to A\}$. Our goal is to estimate for each $\pi \in \Pi$ its return and select the policy with the highest estimated return.

For this task we build a multi-trajectory tree with the parameter $m = (V_{max}^2/\epsilon^2)\log(|\Pi|/\delta)$. for each $\pi \in \Pi$ we have $m$ estimates $v_i$ which we average $\hat{V}^\pi(s_0) = (1/m)\sum_{i=1}^m v_i$. We are guaranteed that for each $\pi \in \Pi$ we have that with probability $1 - \delta/|\Pi|$ that $|\hat{V}^\pi(s_0) - V^\pi(s_0)| \le \epsilon$. Using a union bound over $\pi \in \Pi$ we have:

**Theorem 13.2.** *With probability $1 - \delta$, for $m = O((V_{max}^2/\epsilon^2)\log(|\Pi|/\delta))$, we have that*

$$|V^{\hat{\pi}^*}(s_0) - V^{\pi^*}(s_0)| \le \epsilon$$

*where $\hat{\pi}^* = \arg\max_{\pi\in\Pi} \hat{V}^\pi(s_0)$ and $\pi^* = \arg\max_{\pi\in\Pi} V^\pi(s_0)$.*

Note that the above theorem does not depend on the size of the state space $|S|$ or the complexity of the optimal policy.

### 13.1.4 Gradient based algorithm

Assume that we have a parameterized policy class $\Pi$ where the policies are smooth in the parameter. Namely we have policies $\pi(\cdot;\theta)$ and the policy distribution is smooth in $\theta$. Our goal is to compute the gradient of the expected reward, and allow to improve the policy.

Fix a trajectory tree $T$. We want to compute $\nabla_\theta T(\pi(\cdot;\theta))$, both efficiently and unbiased estimate.

For each node $i \in T$ we have a depth $d_i$ and history $h_i$ leading to it. The history $h_i$ has a reward $v_i^1$. Using policy $\pi(\cdot;\theta)$ we induce a distribution over the leaves of $T$, starting at node $i$, and let $v_{i,a}^2$ be the expected return starting at node $i$ and performing action $a$.

Given the policy $\pi(\cdot;\theta)$ we have a probability $P[i]$ of the trajectory reaching node $i$ when it follows policy $\pi(\cdot;\theta)$, and a distribution over actions in node $i$ which is $p_{i,a} = \pi(a|i;\theta)$.

The expected value of the policy is

$$E[T(\pi(\cdot;\theta))] = \sum_{t=0}^{H} \gamma^t E[r_t|T, \pi(\cdot;\theta)] \propto \sum_{i \in T} \sum_{a \in A} P[i](v_i^1 + \gamma^{d_i} E[v_{i,a}^2]\pi(a|i,\theta)$$

(Since we have that $\sum_{i \in T} P[i] = H$ the expressions are only proportional.)

Using the chain rule we have,

$$\frac{\partial}{\partial \theta} T(\pi(\cdot;\theta)) = \sum_{i \in T, a \in A} \frac{\partial T}{\partial p_{i,a}} \frac{\partial p_{i,a}}{\partial \theta}$$

This implies that

$$\frac{\partial}{\partial \theta} T \propto \sum_{i \in T, a \in A} P[i] \gamma^{d_i} E[v_{i,a}^2] \frac{\partial}{\partial \theta} p_{i,a}$$

we are still left with the question of how do we get an unbiased estimate.

One solution is to sample $i$ with probability proportional to $P[i]\gamma^{d_i}$, and then sample, for each $a \in A$, a continuation of a trajectory with return $V_{i,a}^2$ (and expected return $v_{i,a}^2$). We can then return $V_{i,a}^2 \nabla_\theta p_{i,a}$, and our estimate would be $\sum_{a \in A} \pi(a|i,\theta) V_{i,a}^2 \nabla_\theta p_{i,a}$.

We can implement this sampling as follows. We select a depth $d$ from a geometric distribution with parameter $1 - \gamma$ (i.e., $\Pr[d = \ell] \propto \gamma^\ell$). We follow policy $\pi(\cdot;\theta)$ to depth $d$, and define the node we reached as node $i$. For each action $a \in A$ we follow the trajectory from $i$ which starts with action $a$, and uses policy $\pi(\cdot;\theta)$ to select the future actions. Let $V_{i,a}^2$ the return of this trajectory (from node $i$ starting with action $a$). We return

$$\sum_{a \in A} \pi(a|i,\theta) V_{i,a}^2 \nabla_\theta p_{i,a}$$

We described an efficient way to compute the gradient in an unbiased way. This implies that if we follow the gradients we will reach a local maximum of the expected return.

For the complexity analysis, note that we do not need to compute the entire trajectory tree in advance, we can do it in a lazy way. We add a path for each gradient we compute. This implies that for each gradient estimation we have $O(H|A|)$ call to the generative model.

## 13.2 Near Optimal Policy

We would like to define a near optimal policy given a generative model. The idea is that we define an algorithm, that given a state $s$, runs and queries the generative model as a black box, and return an action $a$. In this way the algorithm defines implicitly a policy. Our goal is that the running time of the algorithm would be efficient. Our algorithm would run in time exponential in $H$, the effective horizon, but independent of the state space size $|S|$.

Our main goal is to establish the following theorem:

**Theorem 13.3.** *The exists an algorithm that, given access to a generative model, defines a policy $\pi$ such that for any state $s$ we have*

$$V^*(s) - V^\pi(s) \leq \epsilon$$

*and the running time of the algorithm is exponential in $H$, the effective discounted horizon, but independent of the state space size.*

We start by looking at a restricted case. Assume that for any state $s \in S$ and action $a \in A$ the support of the distribution of the next state $p(\cdot|s,a)$ is bounded by $d$. (We will latter get rid of this assumption.)

Fix a state $s$ (the input to the algorithm). Using the generative model, recover all the states that can be reached from $s$ with at most $H$ steps. (Assume that we can guarantee to recover all those states, actually, we should have a small error probability, for low probability states.)

Let $\Delta(s,H)$ be the set of states reachable from $s$ with at most $H$ steps. Given our assumption on the support we have $|\Delta(s,H)| \leq (d|A|)^H$. We restrict the MDP to the set $\Delta(s,H)$ and compute the optimal $Q$-function on this MDP, denoted by $Q^{*,H}(s,a)$. Let $a'$ be the action that maximizes the $Q^{*,H}(s,\cdot)$. We have that

$$Q^*(s,a) - \epsilon \leq Q^{*,H}(s,a) \leq Q^*(s,a)$$

where the lower bound is due to the horizon definition.

We would like to show that if in each step we select an action that nearly maximizes the $Q^*$-function, then the policy is near optimal.

**Lemma 13.1.** *If for every state $s$, with probability $1-\delta$, we have*

$$Q^*(s,a^*) - Q^*(s,\pi(s)) \leq \epsilon$$

*where $a^*$ is the optimal action in state $s$. Then, for every state $s$ we have*

$$V^*(s) - V^\pi(s) \leq \frac{\epsilon + 2\delta V_{max}}{1-\gamma}$$

*Proof.*

$$\begin{aligned} E[Q^*(s,\pi(s))] &\geq (1-\delta)(Q^*(s,a^*)-\epsilon) - \delta V_{max} \\ &\geq Q^*(s,a^*) - \epsilon - 2\delta V_{max} \end{aligned}$$

Let $\beta = \epsilon + 2\delta V_{max}$. The following claim completes the proof.

**Claim 13.1.** *If for every state $s$ we have*

$$Q^*(s,a^*) - Q^*(s,\pi(s)) \leq \beta$$

*Then*

$$V^*(s) - V^{\pi}(s) \leq \frac{\beta}{1-\gamma}$$

*Proof.* Let $\pi_i$ be the policy that runs $\pi$ in the first $i$ steps, and then switches to $\pi^*$. For $\pi_1$ we have $V^*(s) - V^{\pi_1}(s) \leq \beta$. For $\pi_i$, since the first $i$ steps are identical, we have $V^{\pi_i}(s) - V^{\pi_{i+1}}(s) \leq \gamma^i \beta$. By summing the inequalities we have that

$$V^*(s) - V^{\pi_H}(s) \leq \sum_{t=0}^{H} \gamma^t \beta \leq \frac{\beta}{1-\gamma}$$

□

□

we now would like to address the general case. For example consider the case that the next state distribution is uniform over $S$. We clearly cannot hope to approximate this model well, simply too large.

Our main insight will be to use the trajectory trees, to approximate the neighborhood around a state $s$, which will be the root. The main issue will be to study the errors in the process.

The basic idea is given a state $s$, we sample each action $a \in A$ for $m$ times. Let $s'_{a,i}$ be the next state in the $i$-th time we execute action $a$ from state $s$. We now would like to compute recursively the return from each $s'_{a,i}$.

The main conceptual issue is whether we are making progress. We wanted to estimate $V^*(s)$, and we have now to estimate $V^*(s'_{a,i})$. It seems that instead to estimate the optimal value from one state we need to approximate it from $m|A|$ state. The main point is that due to discounting we are making progress. We can allow now a larger error rate!

First, we consider the error due to the sampling of $m$ state.

**Lemma 13.2.** *Let* $\hat{Q}^*(s,a) = r(s,a) + (\gamma/m)\sum_{i=1}^m V^*(s'_{a,i})$, *where* $s'_{a,i} \sim p(\cdot|s,a)$. *With probability* $1 - e^{-\epsilon^2 m/V_{max}^2}$ *we have* $|Q^*(s,a) - \hat{Q}^*(s,a)| \leq \epsilon$.

The proof follows from an application of the Chernoff concentration bounds.

**Near optimal algorithm**

`EstimateQ(n,s,a)`:
IF $n = 0$ THEN return 0
ELSE
Sample $m$ times action $a$ in state $s$ (using the generative model).
Let $s'_{a,i}$ be the next next state in the $i$-th sample.
Return $(1/m)\sum_{i=1}^m$ `EstimateV(n-1,s,a)`

`EstimateV(n,s)`: return $\max_a$ `EstimateQ(n,s,a)`.

`main(s)`: return `EstimateV(n,s)`

For the algorithm, we have two sources of errors. The first is due to the finite sample $m$, and is addressed in Lemma 13.2. The second, and more challenging, is the recursive calls. We will use the discounting to show that the error decrease.

We start with a few notations.

$$Q^n(s,a) = \texttt{EstimateQ}(n,s,a)$$
$$V^n(s) = \texttt{EstimateV}(n,s)$$

The computation is simply

$$Q^n(s,a) = r(s,a) + (\gamma/m)\sum_{i=1}^m V^{n-1}(s'_{a,i})$$
$$V^{n-1}(s) = \max_a Q^{n-1}(s,a)$$

We define the error bounds as

$$|Q^*(s,a) - Q^n(s,a)| \leq \alpha_n$$
$$|V^*(s) - V^n(s)| \leq \alpha_n$$

where the first inequality implies the second.

Trivially, we have $\alpha_0 = V_{max}$. For the error bound we have

$$\begin{aligned}|Q^*(s,a) - Q^n(s,a)| =& \gamma |E[V^*(s')] - (1/m)\sum_{i=1}^{m} V^{n-1}(s'_{a,i})| \\ \leq& \gamma |E[V^*(s')] - (1/m)\sum_{i=1}^{m} V^*(s'_{a,i})| \\ &+ \gamma |(1/m)\sum_{i=1}^{m} V^*(s'_{a,i}) - (1/m)\sum_{i=1}^{m} V^{n-1}(s'_{a,i})|\end{aligned}$$

This implies that $\alpha_n \leq \gamma(\epsilon + \alpha_{n-1})$, which implies that

$$\alpha_H \leq \sum_{i=0}^{H} \gamma^i \epsilon + \gamma^H V_{max} \leq \frac{\epsilon}{1-\gamma} + \gamma^H V_{max}$$

To complete the algorithm, we set the parameter $\epsilon'$, by setting $\epsilon = \epsilon'(1-\gamma)$ and $H = (1/(1-\gamma)) \log(V_{max}/\epsilon')$. This implies that $\alpha_H \leq 2\epsilon'$ and have $V^*(s) - V^\pi(s) \leq 2\epsilon'$.

**Lower bound**

The exponential dependence on the horizon $H$ is inherent. To see this, consider an MDP which is a complete $|A|$-degree tree of depth $H$, where for each node each outgoing edge is associated with a different $a \in A$. All the rewards are zero, except one leaf where the reward is one, which is selected at random.

Any algorithm would have to check, on average, at least half the leaves before it finds the unique leaf.

**Theorem 13.4.** *Any algorithm that uses a generative model need to make $\Omega(|A|^H)$ queries.*

## 13.3 Inverse Reinforcement Learning

The classical reinforcement problem has as inputs the dynamics and the rewards and the output is the optimal policy. In the inverse reinforcement learning problem we are given access to the optimal policy and the dynamics, and would like to learn a reward function which induces the given optimal policy.

More specifically, we are given as inputs the states $S$, actions $A$, dynamics $p(\cdot|s,a)$, and access to the optimal policy $\pi^*$, either explicitly or implicitly, by observing trajectories of $\pi^*$.

We have a few related problems:

1. Behavioral cloning: given trajectories learn the optimal policy.
2. Inverse Reinforcement Learning (IRL): which recovers a reward function $R$ for which $\pi^*$ is optimal.
3. Apprentice learning via IRL: Learn a good policy given the rewards $R$ of IRL.

### 13.3.1 Behavioral Cloning

Formulated as a classical supervised classification problem. Given trajectories generated from $\pi^*$, we extract pairs $(s_t, a_t)$, where $\pi^*(s_t) = a_t$. We do not need to know the dynamics or the rewards. We define a classification problem, where the training examples are $T = \{(s_t, a_t)\}$.

We can now learn a classifier for $T$, and use any classifier, DNN, SVM, decision tree, and more. We learn the optimal policy as a classification problem, and if we have low error we can hope for performance near that of the optimal policy.

There are a few problems with the approach. After we make errors we might reach completely new states, which we never encountered before (since the optimal policy never reaches them). Also, the different errors might have very different effects, but we treat them all equally. Other issues include generalization, which would depend on the test and train distributions (of states) being identical, but small differences in the policies might lead to significant differences in the distributions.

### 13.3.2 IRL: small state space

Assume we can use a look-up table to represent the policies. We are given the set of states $S$, actions $A$, $p(\cdot|s,a)$ and the optimal policy $\pi^*$. (All are given explicitly, as look-up tables.) The output of the algorithm is a reward function $r(s,a)$ for which $\pi^*$ is optimal.

We first characterize all the rewards $R$ for which $\pi^*$ is optimal. Bellman optimality requires that for every $s \in S$ and $a \in A$ we have,

$$Q^*(s, \pi^*(s)) \geq Q^*(s,a)$$

First we will try to get a more explicit characterization.

For simplicity of the notation let $a_1 = \pi^*(s)$ for every $s \in S$. (This is simply renaming the actions.) Also, we associate the rewards with states (rater then state-action pairs). Define matrices $P^a$, for $a \in A$, as follows: $P^a[i,j] = p(s_j|s_i, a)$, which is the transition probabilities using action $a$. Since $a_1$ is the optimal action, we define $P^* = P^{a_1}$.

For the optimal value function $V^* \in \mathbb{R}^{|S|}$ has

$$V^* = R + \gamma P^* V^*$$

This implies that $V^* = (I - \gamma P^*)^{-1} R$. (The inverse exists, since the eigenvalues of $P^*$ are in the unit circle, and the eigenvalues of $I - \gamma P^*$ are all non-zero.)

The optimal state-action function is,

$$Q^* = R + \gamma P^a V^* = R + \gamma P^a (I - \gamma P^*)^{-1} R$$

From the Bellman optimality we have $Q^*(s, \pi^*(s)) - Q^*(s, a) \geq 0$ which implies,

$$(P^* - P^a)(I - \gamma P^*)^{-1} R \geq 0$$

This establishes the following claim:

**Claim 13.2.** *Reward function $R$ is optimal iff $(P^* - P^a)(I - \gamma P^*)^{-1} R \geq 0$.*

Although we have an exact characterization, we have a few challenges.

1. Trivial solutions: We can simply set $R = 0$ and any policy is optimal!
2. What happens if we observe only trajectories of $\pi^*$ and not the explicit policy and dynamics.
3. Large state space.

To overcome the ambiguity of the reward function we can set an objective that will force a unique outcome. One solution is a *max margin*, specifically,

$$\max_{s \in S} \left\{ Q^*(s, a^*) - \max_{a \neq a^*} Q^*(s, a) \right\}$$

The resulting linear program is,

$$\begin{aligned} & \max \lambda \\ \forall a^* \neq a \quad & (P^* - P^a)(I - \gamma P^*)^{-1} R \geq \lambda \\ & -R_{max} \leq R \leq R_{max} \end{aligned}$$

### 13.3.3 IRL: Linear function approximation

In the case of a large state space we would use function approximation. We consider a simple case that the rewards are linear in the state representation. For each state $s \in S$ we have $\phi(s) \in \mathbb{R}^n$. There is a reward weights $w \in \mathbb{R}^n$. The reward in state $s$ is $w^\top \phi(s)$.

Consider the value function of the policy $\pi$.

$$\begin{aligned} V^*(s_0) =& E[\sum_{t=0}^{\infty} \gamma^t r_t | \pi] \\ =& E[\sum_{t=0}^{\infty} \gamma^t w^\top \phi(s_t) | \pi] \\ =& w^\top \sum_{t=0}^{\infty} \gamma^t E[\phi(s_t)|\pi] \\ =& w^\top \mu(\pi) \end{aligned}$$

where $\mu(\pi) = \sum_{t=0}^{\infty} \gamma^t E[\phi(s_t)|\pi]$. Note that $\mu(\pi)$ depends on the dynamics (steady state distribution) that policy $\pi$ induces.

For IRL we like to recover a weight vector $w$ such that for any policy $\pi$ we have,

$$w^\top \mu(\pi^*) \geq w^\top \mu(\pi)$$

First, we can approximate $\mu(\pi)$ from sample of $m$ trajectories,

$$\mu(\pi) \approx \hat{\mu}(\pi) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{H} \gamma^t \phi(s_t)$$

Second, observe that if we have $\|\mu(\pi) - \hat{\mu}(\pi)\|_2 \leq \epsilon$ then for any $w$ we have

$$|w^\top \mu(\pi) - w^\top \hat{\mu}(\pi)| \leq \epsilon \|w\|_2$$

which implies that if we find a policy $\pi'$ such that $\mu(\pi') \approx \mu(\pi^*)$, then for any weights $w$ the policy $\pi'$ will give a similar expected return as $\pi^*$.

As before, we have a problem that $w = 0$ is a solution. We will overcome this by defining a max margin linear program.

$$\begin{aligned} & \max \lambda \\ \forall \pi \quad & w^\top \mu(\pi^*) \geq w^\top \mu(\pi) + \lambda \\ & \|w\|_2^2 \leq 1 \end{aligned}$$

A clear weakness is that we have an exponential size linear program. We will overcome this by an incremental constraint generation. We maintain a set of policies $\Pi$, where initially $\Pi = \emptyset$. Initially we select some $\pi^0$ and let $\mu^0 = \mu(\pi^0)$. At iteration $i$ we solve

$$\max_{w} \min_{\pi_j \in \Pi} w^\top (\mu^* - \mu^j)$$

If the value is at most $\epsilon$ we terminate. Otherwise let $w^i$ be the maximizer. We solve for the optimal policy $\pi^i$ given weights $w^i$ and add $\pi^i$ to $\Pi$. At termination, we compute a $\mu = \sum_{i=1}^{k} a_i \mu^i$ which minimizes $\|\mu^* - \mu\|_2$, where $a_i \geq 0$ and $\sum_i a_i = 1$.

The correctness of the constraint generation follows from the following lemma.

**Lemma 13.3.** *At termination* $\|\mu^* - \mu\|_2 \leq \epsilon$

We can implement $\mu$ by selecting policy $\pi^i \in \Pi$ with probability $a_i$ and running it to completion.

The following lemma guarantees the convergence.

**Lemma 13.4.** *The number of iterations is bounded by*

$$O(\frac{n}{(1-\gamma)^2\epsilon^2} \log \frac{n}{(1-\gamma)\epsilon})$$

## 13.4 Bibliography Remarks

The trajectory trees for approximate planning and evaluation is taken from [2, 3].

The inverse reinforcement learning follows [1, 4]

# Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, 2004.

[2] Michael J. Kearns, Yishay Mansour, and Andrew Y. Ng. Approximate planning in large pomdps via reusable trajectories. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 1001–1007, 1999.

[3] Michael J. Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.

[4] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 663–670, 2000.