IEOR 265 - Lecture 11
Approximate Policy Iteration

1 Approximate Policy Iteration

In this lecture, we will introduce approximations to the Policy Iteration (PI) algorithm. We start by recapping the policy evaluation and policy improvement steps which constitutes the PI algorithm:

$$J_{\mu^{(t)}}(i) = \sum_{j=1}^{n} p_{ij}(\mu^{(t)}(i))\big(g(i,\mu^{(t)}(i),j) + \alpha J_{\mu^{(t)}}(j)\big),\ \forall i \in \{1,...,n\} \tag{1}$$

$$\mu^{(t+1)}(i) \in \arg\min_{u\in U(i)} \left\{ \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_{\mu^{(t)}}(j)\big) \right\},\ \forall i \in \{1,...,n\} \tag{2}$$

Approximate PI can be framed in terms of approximating either, or both, steps of the PI algorithm. Such algorithms are called **Actor-Critic Algorithms**, where it is composed of two steps:

1. **Critic Step**: This replaces the policy evaluation step; given a current policy $\mu^{(t)}$ we use an approximation architecture to perform the policy evaluation, namely to compute the cost-to-go function $\tilde{J}_{\mu^{(t)}} \approx J_{\mu^{(t)}}$.
2. **Actor Step**: This replaces the policy improvement step; given the approximate cost-to-go function $\tilde{J}_{\mu^{(t)}}$, we solve a lookahead minimization to generate an improved policy $\mu^{(t+1)}$. Such minimization can also be approximated by using an architecture, such as Deep Neural Networks (DNN), to generate $\tilde{\mu}^{(t+1)} \approx \mu^{(t+1)}$.

There are many different ways in which either the critic and the actor steps can be implemented. On this lecture we will study each step separately and then we will combine them both into the Actor-Critic Algorithm. Again, we stress that there are many different ideas in the literature we our goal is to provide a solid basis in order to understand and evaluate those ideas.

2 Critic-only Algorithms

We will start with critic-only algorithms, where we use approximation architectures, such as a DNN, to perform the critic-step and we compute the actor step (policy improvement) exactly. Suppose we have at hand some base policy $\mu^{(t)}$.

We will approximate the cost-to-go function associated with this policy by an architecture with parameters θ. To that end, we write:

$$\tilde{J}_{\mu^{(t)}}(i,\theta) \approx J_{\mu^{(t)}}(i) \tag{3}$$

where the function on the left-hand side is our approximate function. We also note that, differently from Value Iteration (VI), our cost-to-go function approximation is associated with the policy $\mu^{(t)}$. Indeed, in PI, the cost-to-go functions will all always be associated with some (stationary) policy $\mu^{(t)}$.

As always, suppose that we can generate sample initial state/cost-to-go pairs:

$$(i_0^1,\beta^1),(i_0^2,\beta^2),...,(i_0^S,\beta^S) \tag{4}$$

where S is the number of samples. These cost-to-go values can be generated by applying the policy $\mu^{(t)}$, for M time periods and then adding another terminal cost approximation at the end. We illustrate this process in the figure below:

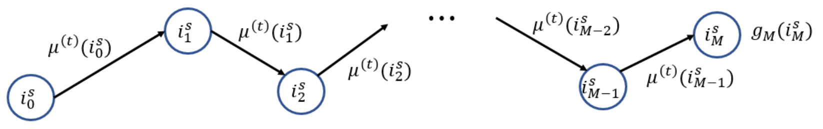

Figure 1: Schematic representation of the simulated trajectory generated bu sequentially applying the policy $\mu^{(t)}$

Then each cost-to-go "label" β^s is given by:

$$\beta^s = \sum_{k=0}^{M-1} \alpha^k g(i_k^s, \mu^{(t)}(i_k^s), i_{k+1}^s) + \alpha^M \hat{J}_M(i_M^s) \tag{5}$$

At this moment, it is not really important how the terminal cost approximation $\hat{J}_M(i_M^s)$ is generated. It could be simply equal to zero, or it could be given by some other algorithm, such as VI. In addition we note that for each initial sampled state i_0^s, we simulate an entire rollout trajectory for M stages. We also highlight that $\mu^{(t)}$ plays the role of the base policy in the Rollout Algorithm. Then the **critic** step boils down to the usual regression(training) problem:

$$\theta^{(t)} = \arg\min_{\theta} \left\{ \sum_{s=1}^{S} \left(\tilde{J}_{\mu^{(t)}}(i_0^s,\theta) - \beta^s\right)^2 \right\} \tag{6}$$

which can then be solved by some gradient-type method, such as Stochastic Gradient Descent (SGD), like before. After solving the regression problem to (local) optimality, as the problem is typically not convex, we apply the actor-step:

$$\mu^{(t+1)}(i) \in \arg\min_{u\in U(i)} \left\{ \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde{J}_{\mu^{(t)}}(j,\theta^{(t)})\big) \right\}, \forall i \in \{1,...,n\} \tag{7}$$

which is exactly the same as the policy improvement step in exact PI. Then the critic-only algorithm iterates: (1) use $\mu^{(t+1)}$ to simulate more trajectories;

(2) update $\theta^{(t)}$ to $\theta^{(t+1)}$; (3) use the new approximate function to generate a new policy, and so forth.

A central problem of this algorithm is that the samples that it generates across the iterations via simulation are always "linked" to some policy. In the literature, when this happens we say that the algorithm is *on-policy*: samples generated while using the policy $\mu^{(t)}$ cannot be treated to be the same as sampled generated under $\mu^{(t)}$. More formally, the transitions that we observe while generating the samples follow a probability distribution defined by $p_{ij}(\mu^{(t)}(i))$, which depends explicitly on the policy $\mu^{(t)}$ being used. This if often a "problem", because it limits online applications, since simulation is often very costly and the critic-algorithm requires simulation at every iteration. This can be a huge burden and represent a serious obstacle for application where training needs to be performed online.

In addition, if the rollout horizon M is too long, there will be a lot of variance in the simulated costs. On the other hand, if the rollout horizon M is too short, there will be a lot of bias since the terminal cost approximation will be used "too soon". This bias-tradeoff is ever-present in approximate PI and can be represented schematically in the following figure: We will defer

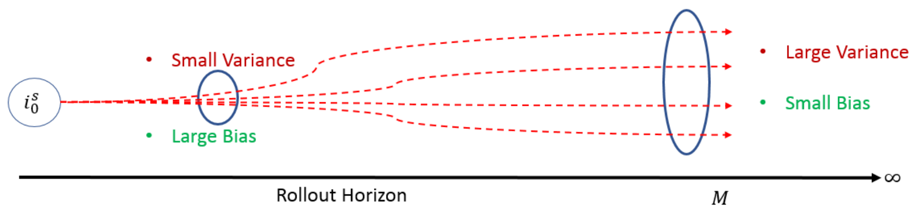

Figure 2: Schematic representation of the bias-variance trade-off faced by the critic-only algorithm: longer trajectories have less bias as costs are computed accumulated for longer, but have longer variance; while shorter horizon have low variance but hight bias.

the discussion on how to overcome this to the next lecture. Here it is enough to just state this tradeoff between needs to be addressed before presenting a practically useful algorithm. Lastly, we can also make the critic-only algorithm *model free*, that is, without the need to know the transition probabilities, by using Q-factors. As always with Q-factors, given a state-control pair (i, u) we generate the approximation:

$$\tilde{Q}_{\mu^{(t)}}(i, u, \theta) \approx Q_{\mu^{(t)}}(i, u) \tag{8}$$

then, the critic step becomes, after obtaining samples $\{i_0^s, u^s, \beta^s\}$:

$$\theta^{(t)} = \arg\min_{\theta} \left\{ \sum_{s=1}^{S} \left(\tilde{Q}_{\mu^{(t)}}(i_0^s, u^s, \theta) - \beta^s\right)^2 \right\} \tag{9}$$

with the Q-factor "labels", defined as:

$$\beta^s = g(i_0^s, u^s, i_{k+1}^s) + \sum_{k=1}^{M-1} \alpha^k g(i_k^s, \mu^{(t)}(i_k^s), i_{k+1}^s) + \alpha^M \hat{J}(i_M^s) \tag{10}$$

the policy improvement step becomes:

$$\mu^{(t+1)}(i) \in \arg\min_{u \in U(i)} \left\{ \tilde{Q}_{\mu^{(t)}}(i, u, \theta^{(t)}) \right\}, \forall i \in \{1, ..., n\} \tag{11}$$

where we note that the transition probabilities are not needed anymore to perform the above minimization.

3 Actor-only Algorithm: The Policy Gradient

Now we move to study Actor-only algorithms, where the main goal is to introduce approximations to the policy improvement step, or more generally, on how to obtain better and better policies. We will introduce approximations in the *policy space*, that is, we use a DNN with parameters θ, to generate:

$$\tilde{\mu}(i, \theta) \approx \mu(i) \tag{12}$$

where we note that now the approximation architecture provides a mapping from states to actions, instead of states to cost-to-go values. We note a particular case where the policy approximation is given via cost-to-go approximation. This can be defined as:

$$\tilde{\mu}(i, \theta) \in \arg\min_{u \in U(i)} \left\{ \sum_{j=1}^{n} p_{ij}(u) \big(g(i, u, j) + \alpha \tilde{J}(j, \theta) \big) \right\}, \forall i \in \{1, ..., n\} \tag{13}$$

In the equation above, the approximated policy is given as the "Argmin" of a 1-step lookahead problem. We have to be careful though, because such policy can be non-differentiable w.r.t. θ and this may lead to complications. We will refrain from going in this direction, by analyzing a method that directly obtains

$$\tilde{\mu}(i, \theta) \approx \mu(i)$$

by optimizing a certain criterion (instead of being given as a solution of a lookahead problem). This is the popular **Policy Gradient** method. Ideally what we want is to solve the following optimization problem:

$$\min_{\theta} \mathbb{E}_{p_0} \left[J_{\tilde{\mu}(\theta)}(i_0) \right] \tag{14}$$

In words: "we want to fit a DNN that provides a policy that minimizes the expected cost-to-go when we follow such policy, where the expectation is taken w.r.t. the initial state distribution". Notice upon solving such problem, we are not guaranteed to obtain the optimal stationary policy for the infinite-horizon DP. All we are doing is minimizing the cost-to-go function for the stationary policy generated by the DNN.

If we assume that $J_{\tilde{\mu}(\theta)}(\cdot)$ is differentiable w.r.t. θ and if we knew the initial state i_0, we could then write the usual gradient step:

$$\theta^{(t+1)} = \theta^{(t)} - \gamma^{(t)} \nabla J_{\tilde{\mu}(\theta^{(t)})}(i_0), \quad \forall t \geq 0 \tag{15}$$

the problems are of course the fact that the gradient $\nabla J_{\tilde{\mu}(\theta^{(t)})}(i_0)$ may not be given explicitly and we may not know the initial state distribution p_0. We will leverage the power of sampling and simulation to solve both issues. We will perform a "trick", where we will expand the feasible space of policies, by considering randomized policies, that is policies that provide a distribution of the available set of controls $U(i)$ given the state i (it is easy to see that deterministic policies are a special case of randomized policies where all the probability mass is placed on a single control). Namely we write:

$$\tilde{\mu}(i,\theta) = u, \quad w.p. \quad p(u|i,\theta) \qquad \forall u \in U(i) \tag{16}$$

then the optimization problem becomes:

$$\min_{\theta} \mathbb{E}_{p_{(z|\theta)}} \big[\sum_{k=0}^{\infty} \alpha^k g(i_k, u_k) \big] \tag{17}$$

where $p_{(z|\theta)}$ is the conditional distribution of $z = (i_0, u_0, i_1, u_1, ...)$ given θ. Note that for simplicity in the presentation we assumed that the stage costs only depend on the starting stage and control. Lastly, observe that we can unpack $p_{(z|\theta)}$ by applying the Markov Property:

$$p(z|\theta) = p(i_0, u_0, i_1, u_1..., |\theta) = p(i_0) \prod_{k=0}^{\infty} p_{i_k, i_{k+1}}(u_k) p(u_k|i_k, \theta) \tag{18}$$

Now we define $F(z)$ as:

$$F(z) = \sum_{k=0}^{\infty} \alpha^k g(i_k, u_k) \tag{19}$$

Then the optimization problem becomes:

$$\min_{\theta} \mathbb{E}_{p_{z|\theta}} \big[F(z) \big] \tag{20}$$

Now we proceed to take the gradient of the above objective function and we write the following series of equalities:

$$\begin{aligned}
&\nabla_\theta \big(\mathbb{E}_{p(z|\theta)} \big[F(z) \big] \big) = \nabla_\theta \big(\sum_{z \in Z} p(z|\theta) F(z) \big) = \sum_{z \in Z} \nabla_\theta \big(p(z|\theta) \big) F(z) = \\
&\sum_{z \in Z} p(z|\theta) \frac{\nabla_\theta (p(z|\theta))}{p(z|\theta)} F(z) = \sum_{z \in Z} p(z|\theta) \nabla_\theta \big(\ln(p(z|\theta)) \big) F(z) = \\
&\mathbb{E}_{p(z|\theta)} \Big[\nabla_\theta \big(\ln(p(z|\theta)) \big) F(z) \Big]
\end{aligned} \tag{21}$$

where from the third to fourth equality we applied the "log-trick" $\nabla \ln(p) = \frac{\nabla p}{p}$. Now we use Eq(18) to expand the gradient inside the expectation:

$$\begin{aligned}
&\nabla_\theta \big(\ln(p(z|\theta)) \big) = \nabla_\theta \big(\ln(p(i_0) \prod_{k=0}^{\infty} p_{i_k, i_{k+1}}(u_k) p(u_k|i_k, \theta)) \big) = \\
&\nabla_\theta \Bigg(\ln(p(i_0)) + \sum_{k=0}^{\infty} \ln(p_{i_k, i_{k+1}}(u_k)) + \sum_{k=0}^{\infty} \ln(p(u_k|i_k, \theta)) \Bigg) = \\
&\sum_{k=0}^{\infty} \nabla_\theta \big(\ln(p(u_k|i_k, \theta)) \big)
\end{aligned} \tag{22}$$

where in the last equality we used the fact that the first two terms do not depend on the θ (hence their gradient w.r.t. θ is zero). Now substituting it back in the gradient expression gives us:

$$\nabla_\theta\big(\mathbb{E}_{p(z|\theta)}\big[F(z)\big]\big) = \mathbb{E}_{p(z|\theta)}\Big[\nabla_\theta\big(\ln(p(z|\theta))\big)F(z)\Big] = \tag{23}$$

$$\mathbb{E}_{p(z|\theta)}\Bigg[\Big(\sum_{k=0}^{\infty}\nabla_\theta\big(\ln(p(u_k|i_k,\theta))\big)\Big)\Big(\sum_{k=0}^{\infty}\alpha^k g(i_k,u_k)\Big)\Bigg] \tag{24}$$

Now, we remove the infinite summations, by truncating the trajectories at some stage M and using a terminal cost approximation $\hat{J}_M$:

$$\mathbb{E}_{p(z|\theta)}\Bigg[\Big(\sum_{k=0}^{M-1}\nabla_\theta\big(\ln(p(u_k|i_k,\theta))\big)\Big)\Big(\sum_{k=0}^{M-1}\alpha^k g(i_k,u_k)+\alpha^M\hat{J}_M(i_m)\Big)\Bigg] \tag{25}$$

Now, as always, we use Sample Average Approximation (SAA) to replace the above expectation using simulation, and we obtain:

$$\nabla_\theta\big(\mathbb{E}_{p(z|\theta)}\big[F(z)\big]\big) \approx$$
$$\frac{1}{S}\sum_{s=1}^{S}\Big(\sum_{k=0}^{M-1}\nabla_\theta\big(\ln(p(u_k^s|i_k^s,\theta))\big)\Big)\Big(\sum_{k=0}^{M-1}\alpha^k g(i_k^s,u_k^s)+\alpha^M\hat{J}_M(i_m^s)\Big) \tag{26}$$

where we use uniform weight $1/S$ for every scenario, where S is the number of simulated trajectories. At last, we provide the algorithm known as the REINFORCE algoritmh (or simply the Polict Gradient) as follows:

Algorithm 1 REINFORCE Algorithm (Policy Gradient)

```
Input: Initial DNN parameters $\theta^{(0)}$ and randomized policy $\tilde{\mu}(\theta^{(0)})$.
1: for $t = 0, ..., T$ do (obtaining new samples)
2:     Collect $S$ sample trajectories $z^s = (i_0^s, u_0^s, ..., i_M^s)$ using the policy $\tilde{\mu}(\theta^{(t)})$
3:     Compute the policy gradient:
           $\nabla_\theta\big(\mathbb{E}_{p(z|\theta^{(t)})}\big[F(z)\big]\big)$
           $\approx \frac{1}{S}\sum_{s=1}^{S}\Big(\sum_{k=0}^{M-1}\nabla_\theta\big(\ln(p(u_k^s|i_k^s,\theta^{(t)}))\big)\Big)\Big(\sum_{k=0}^{M-1}\alpha^k g(i_k^s,u_k^s)+\alpha^M\hat{J}_M(i_m^s)\Big)$
4:     Perform the gradient step:
           $\theta^{(t+1)} = \theta^{(t)} - \gamma^{(t)}\nabla_\theta\big(\mathbb{E}_{p(z|\theta^{(t)})}\big[F(z)\big]\big)$
5: end for
Output: The last DNN configuration $\theta^{(T)}$. A suboptimal policy $\tilde{\mu}(\theta^{(T)})$
```

3.1 Example: Gaussian Policies

As an example of randomized policy, Gaussian Networks are often used in the policy gradient framework, when we have continuous actions (an analogous discrete-action version can be implemented using “soft-max” type policies with

Logistitc functions; we leave that as an extra reading). For the Gaussian case, we have:

$$\tilde{\mu}(i, \theta) \sim \mathcal{N}(m_\theta(i), \Sigma)$$

and the DNN architecture maps the state i of the MDP the mean of the Gaussian $m_\theta(i)$. And the policy simply samples an action from the Gaussian parametrized as Eq(27). In this case we can write:

$$\ln(p(u_k|i_k, \theta)) = -\frac{1}{2}(m_\theta(i_k) - u_k)^\top \Sigma (m_\theta(i_k) - u_k) + \text{constant} \tag{27}$$

and now taking the gradient w.r.t. θ yields the nice expression:

$$\nabla_\theta\big(\ln(p(u_k|i_k, \theta))\big) = -\frac{1}{2}\Sigma^{-1}(m(i_k) - u_k)\nabla_\theta(m_\theta(i_k)) \tag{28}$$

where $\nabla_\theta(m_\theta(i_k))$ is obtained via Backpropagation. The computation of the policy gradient becomes very efficient when use Gaussian Networks, which is a very attractive feature for online training.

3.2 Policy Gradient and weighted likelihood

We finish this lecture by highlighting the interesting connection between the Policy Gradient, Eq(26) and Maximum Likelihood Estimation (MLE). Let us restate the policy gradient:

$$\nabla_\theta\big(\mathbb{E}_{p(z|\theta)}\big[F(z)\big]\big) \approx \\ \frac{1}{S}\sum_{s=1}^{S}\Bigg(\sum_{k=0}^{M-1}\nabla_\theta\big(\ln(p(u_k|i_k^s, \theta))\big)\Bigg)\Bigg(\sum_{k=0}^{M-1}\alpha^k g(i_k^s, u_k^s) + \alpha^M \hat{J}_M(i_m^s)\Bigg) \tag{29}$$

We note that each term in the outer summation is composed of a product of two terms. Let's also recall our MDP model

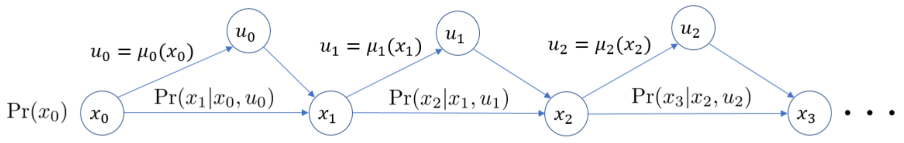

Figure 3: Schematic representation of the MDP

Let's ignore the costs, and focus only on simulating sample trajectories of the above MDP. Then for a randomized policy $\tilde{\mu}(\cdot, \theta)$ we can ask the following question: Find the parameter θ that makes the sampled sequences most likely to occur. That is exactly the MLE problem (which we studied before for the HMM problem), and it is given by:

$$\max_\theta\Big\{\prod_{s=1}^{S} p(z^s|\theta)\Big\} = \max_\theta\Big\{\prod_{s=1}^{S} p(i_0^s)\prod_{k=0}^{\infty} p_{i_k^s, i_{k+1}^s}(u_k^s)p(u_k^s|i_k^s, \theta)\Big\} \tag{30}$$

Now, taking the log and then the gradient we obtain the following expression:

$$\frac{1}{S}\sum_{s=1}^{S}\Bigg(\sum_{k=0}^{M-1}\nabla_\theta\big(\ln(p(u_k^s|i_k^s,\theta))\big)\Bigg) \tag{31}$$

This is exactly equal to the first component of the Policy Gradient. Again we state the policy gradient but now we highlight each component:

$$\nabla_\theta\big(\mathbb{E}_{p(z|\theta)}[F(z)]\big) \approx \frac{1}{S}\sum_{s=1}^{S}\underbrace{\Bigg(\sum_{k=0}^{M-1}\nabla_\theta\big(\ln(p(u_k|i_k^s,\theta))\big)\Bigg)}_{\boxed{\text{Maximum-Likelihood}}}\underbrace{\Bigg(\sum_{k=0}^{M-1}\alpha^k g(i_k^s,u_k^s)+\alpha^M\hat{J}_M(i_m^s)\Bigg)}_{\boxed{\text{Costs ``weights''}}}$$

So we can view the policy gradient as placing a "weight" on each sampled trajectory equal to the total cost associated with that trajectory. So we "tilt" our search for the configuration θ that makes states with higher costs **less** likely to occur (remember we go on the direction of the **negative gradient** in policy gradient!).

This answers the question of what the policy gradient is optimizing: It tries to solve a weighted MLE problem, where the weights are given by the costs associated with the sampled trajectories.