ECE 598: Interplay between Control and Machine Learning **Fall 2020**

Lecture 13

Policy-Based RL Methods

Lecturer: Bin Hu, Date:10/22/2020-10/29/2020

In the last lecture, we have talked about value-based RL methods including Q-learning, SARSA, fitted Q-iteration, and DQN (with experience replay). Those methods try to fit the optimal Q function to enforce the left and right sides of the optimal Bellman equation to be roughly equal to each other on data. Sometimes these methods have stability issues for continuous control problems. For example, the ε-greedy policy for certain Q_k may destabilize the system and the learning will fail. For continuous control tasks with stability concerns, we can use policy-based RL methods which incrementally update the policy without destabilizing the closed-loop dynamics. In this lecture, we will cover several popular policy-based RL methods.

There are many different types of policy-based RL methods, e.g. REINFORCE, Actor-Critic, natural policy gradient, TRPO, PPO, ACTKR, SAC, SVPG, etc. The basic idea is that we parameterize the policy and then perform gradient-based optimization on these policy parameters using data.

To illustrate the idea of policy gradient methods, let's look at the LQR problem first. Again, consider the following linear dynamical system

$$x_{t+1} = Ax_t + Bu_t \tag{13.1}$$

with the quadratic cost:

$$\mathcal{C} = \mathbb{E}_{x_0 \sim \mathcal{D}} \sum_{t=0}^{\infty} (x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t) \tag{13.2}$$

where Q and R are positive definite matrices. Let Σ_0 be the covariance matrix of x_0.

To apply policy gradient methods, we first need to parameterize the policy. For LQR, it is known that a state-feedback linear controller can achieve the optimal performance. Hence we can just confine the policy search to linear state-feedback policies, $u_t = -Kx_t$. Now the cost becomes a function of K. We have

$$\mathcal{C}(K) = \mathbb{E}_{x_0 \sim \mathcal{D}} \sum_{t=0}^{\infty} x_t^\mathsf{T} (Q + K^\mathsf{T} R K) x_t \tag{13.3}$$

In principle, we can just think the above function as an objective function and K as decision variables. This allows us to formulate the LQR problem as an optimization problem

$$\min_K \mathcal{C}(K)$$

We can then apply various gradient-based methods to find the optimal policy. For example, we can apply the gradient descent method, i.e.

$$K_{l+1} = K_l - \alpha \nabla \mathcal{C}(K_l)$$

We can also try Newton's method, Gauss-Newton method, trust region methods or other types of faster methods.

If we know (A, B, Q, R), then we can calculate both $\mathcal{C}(K)$ and $\nabla \mathcal{C}(K)$ in closed-form. Specifically, we know $\mathcal{C}(K)$ can be calculated as $\mathrm{trace}(P_K \Sigma_0)$ by solving the following Bellman equation (Lyapunov equation)

$$P_K = Q + K^\mathsf{T} R K + (A - BK)^\mathsf{T} P_K (A - BK) \tag{13.4}$$

A useful result from control theory states that the gradient $\nabla \mathcal{C}(K)$ can be calculated as

$$\nabla \mathcal{C}(K) = 2((R + B^\mathsf{T} P_K B)K - B^\mathsf{T} P_K A)\Sigma_K \tag{13.5}$$

where $\Sigma_K = \sum_{t=0}^{\infty} \mathbb{E}(x_t x_t^\mathsf{T}) = \sum_{t=0}^{\infty} (A - BK)^t \Sigma_0 ((A - BK)^\mathsf{T})^t$. There are several proofs for this result. We present one proof here. We can take the total derivative of both sides of the Bellman equation to get

$$dP_K = d(K^\mathsf{T} R K) + d\left((A - BK)^\mathsf{T} P_K (A - BK)\right)$$

By the chain rule, we have

$$\begin{aligned}
&dP_K \\
=&dK^\mathsf{T} R K + K^\mathsf{T} R dK + (A - BK)^\mathsf{T} dP_K (A - BK) - dK^\mathsf{T} B^\mathsf{T} P_K (A - BK) - (A - BK)^\mathsf{T} P_K B dK \\
=&dK^\mathsf{T}\left((R + B^\mathsf{T} P_K B)K - B^\mathsf{T} P_K A\right) + \left(K^\mathsf{T}(R + B^\mathsf{T} P_K B) - A^\mathsf{T} P_K B\right) dK + (A - BK)^\mathsf{T} dP_K (A - BK)
\end{aligned}$$

If we view dP_K as the variable, the above is a Bellman equation which can be solved as

$$dP_K = \sum_{t=0}^{\infty} ((A - BK)^\mathsf{T})^t (dK^\mathsf{T} E_K + E_K^\mathsf{T} dK)(A - BK)^t$$

where $E_K = (R + B^\mathsf{T} P_K B)K - B^\mathsf{T} P_K A$. By definition, we have $d\mathcal{C}(K) = \sum_{i,j} \frac{\partial \mathcal{C}}{\partial K_{ij}} dK_{ij} = \mathrm{trace}(\nabla \mathcal{C}(K) dK^\mathsf{T})$. Since $\mathcal{C}(K) = \mathrm{trace}(P_K \Sigma_0)$, it is also straightforward to show (you should verify this step by yourself)

$$d\mathcal{C}(K) = \mathrm{trace}(dP_K \Sigma_0) = \mathrm{trace}(2 E_K \Sigma_K dK^\mathsf{T})$$

Therefore, we have $\nabla \mathcal{C}(K) = 2 E_k \Sigma_K$. Based on the gradient formula, three algorithms can be used:

- Policy gradient: $K_{l+1} = K_l - \alpha \nabla \mathcal{C}(K_l)$

- Natural policy gradient: $K_{l+1} = K_l - \alpha \nabla \mathcal{C}(K_l) \Sigma_{K_l}^{-1} = K_l - 2\alpha((R + B^\mathsf{T} P_{K_l} B)K_l - B^\mathsf{T} P_{K_l} A)$
- Gauss-Newton: $K_{l+1} = K_l - 2\alpha(K_l - (R + B^\mathsf{T} P_{K_l} B)^{-1} B^\mathsf{T} P_{K_l} A)$. When $\alpha = \frac{1}{2}$, the Gauss-Newton method exactly becomes the policy iteration method.

If the model is unknown, then we need to estimate $\nabla \mathcal{C}(K_l)$ from data, and that is the basic idea of the policy-based RL. This is different from value-based RL methods where we parameterize the optimal Q function and then try to fit it. Here we parameterize the policy and try to iterate on the policy space to obtain a good policy.

We can also try to estimate Σ_K and then implement the natural policy gradient method. The Gauss-Newton method with $\alpha = \frac{1}{2}$ can also be implemented in a data-driven manner by using Least Square Policy Iteration (LSPI).

How to estimate $\nabla \mathcal{C}(K)$ from data? There is a famous result: policy gradient theorem. This theorem can be used to estimate the policy gradient directly. Notice that policy gradient theorem requires the policy to be stochastic so that we can explore the space. Denote the policy parameter as θ. Now we discuss various versions of the policy gradient theorem.

Warm-up: Policy Gradient Theorem V0. To make things simple, we first use a large N to approximate the infinite horizon. Therefore, we consider

$$\mathcal{C}(\theta) = \mathbb{E} \sum_{t=0}^{N} \gamma^t c(x_t, u_t) \tag{13.6}$$

Clearly the joint density of $(x_0, u_0, x_1, u_1, x_2, u_2, \ldots, x_N, u_N)$ depends on θ and we denote this density as f_θ. By definition, we have

$$\mathbb{E} c(x_t, u_t) = \int c(x_t, u_t) f_\theta(x_0, u_0, x_1, u_1, x_2, u_2, \ldots, x_N, u_N) dx_0 du_0 dx_1 du_1 \ldots dx_N du_N$$

Hence we can take gradient on both sides to show

$$\begin{aligned}
\nabla_\theta \mathbb{E} c(x_t, u_t) &= \int c(x_t, u_t) \nabla_\theta f_\theta(\cdot) dx_0 du_0 dx_1 du_1 \ldots dx_N du_N \\
&= \int c(x_t, u_t) \frac{\nabla_\theta f_\theta(\cdot)}{f_\theta(\cdot)} f_\theta(x_0, u_0, x_1, u_1, \ldots, x_N, u_N) dx_0 du_0 dx_1 du_1 \ldots dx_N du_N \\
&= \int c(x_t, u_t) \nabla_\theta \log f_\theta(\cdot) f_\theta(x_0, u_0, x_1, u_1, \ldots, x_N, u_N) dx_0 du_0 dx_1 du_1 \ldots dx_N du_N \\
&= \mathbb{E}\left[c(x_t, u_t) \nabla_\theta \log f_\theta(\cdot)\right]
\end{aligned}$$

In addition, we have

$$\begin{aligned}
&\log f_\theta(x_0, u_0, \ldots, x_N, u_N) \\
&= \log \pi_\theta(u_N | x_N) + \log p(x_N | x_{N-1}, u_{N-1}) + \log \pi_\theta(u_{N-1} | x_{N-1}) + \log p(x_{N-1} | x_{N-2}, u_{N-2}) + \ldots
\end{aligned}$$

After taking gradients, we have

$$\nabla_\theta \log f_\theta(x_0, u_0, \dots, x_N, u_N) = \sum_{t=0}^{N} \nabla_\theta \log \pi_\theta(u_t|x_t)$$

Putting things together, we have

$$\begin{aligned}\nabla \mathcal{C}(\theta) &= \sum_{t=0}^{N} \gamma^t \mathbb{E}\left[c(x_t, u_t) \sum_{k=0}^{N} \nabla_\theta \log \pi(u_k|x_k)\right] \\ &= \mathbb{E}\left[\big(\sum_{t=0}^{N} \gamma^t c(x_t, u_t)\big)\big(\sum_{k=0}^{N} \nabla_\theta \log \pi(u_k|x_k)\big)\right]\end{aligned}$$

The above is the most naive version of the policy gradient theorem. It states that we can estimate the policy gradient by sampling the cost sequence and calculating $\nabla_\theta \log \pi(u_t|x_t)$.

How to calculate $\nabla_\theta \log \pi_\theta(u_t|x_t)$? When applying the policy gradient theorem, a stochastic policy has to be used. The noise is injected into the gradient for exploration purpose. For example, consider a LQR problem where one can confine the policy search to linear policies, i.e. $u_t = -Kx_t$. However, to apply the policy gradient theorem, noise has to be injected into the policy, and hence one typically use a Gaussian policy, i.e. $u_t \sim \mathcal{N}(-Kx, \sigma I)$. Basically we just add zero-mean Gaussian noise to the control input $-Kx_t$. How can we calculate $\nabla_\theta \log \pi_\theta(u_t|x_t)$ for such a Gaussian policy? Suppose σ is fixed. First notice K is a matrix. Hence $\nabla_\theta \log \pi_\theta$ is also a matrix. The (i, j)-th entry of this matrix is just

$$\frac{\partial \log \pi_\theta}{\partial K_{ij}} = -\sigma^{-1}(u_t^{(i)} + \sum_{p=1}^{n_x} K_{ip} x_t^{(p)}) x_t^{(j)}$$

where the superscript (i) denotes the i-th entry of the vector. More compactly, we can write $\nabla_\theta \log \pi_\theta(u_t|x_t) = -(\sigma I)^{-1}(u_t + Kx_t)x_t^\mathsf{T}$.

Neural Network Policy. For complicated tasks, we typically use neural network to parameterize the policy. For example, we can use a two-layer neural network to parameterize the Gaussian policy, i.e. $u_t \sim \mathcal{N}(W^1\sigma(W^0 x_t), \sigma I)$ where σ is the elementwise activation. How to calculate $\nabla_\theta \log \pi_\theta(u_t|x_t)$ for such a policy? The derivative with respect to W^1 can be directly calculated as

$$\frac{\partial}{\partial W^1} \log \pi_\theta(u_t|x_t) = \sigma^{-1}(u_t - W^1 h(W^0 x_t))(h(W^0 x_t))^\mathsf{T}$$

The derivative with respect to W^0 requires a backpropagation step and can be calculated as

$$\frac{\partial}{\partial W^0} \log \pi_\theta(u_t|x_t) = \sigma^{-1}(W^1 \operatorname{diag}(h'(W^0 x_t)))^\mathsf{T}(u_t - W^1 h(W^0 x_t))x_t^\mathsf{T}$$

where $\text{diag}(h'(W^0 x_t))$ is a diagonal matrix whose (i,i)-th entry is equal to the i-th entry of the vector $h'(W^0 x_t)$. In general, back propagation can be used to efficiently calculate the term $\nabla_\theta \log \pi_\theta(u_t|x_t)$ when a neural network policy is used. Such operations are available in PyTorch or TensorFlow.

REINFORCE: Policy Gradient Theorem V1. The gradient estimator introduced above is noisy and has high variance. Now we discuss an improved version of the policy gradient theorem. Let us directly address the infinite horizon problem, i.e.

$$\mathcal{C}(\theta) = \mathbb{E} \sum_{t=0}^{\infty} \gamma^t c(x_t, u_t)$$

Notice the process is causal, i.e. (x_t, u_t) do not depend on the states/actions in the future. To calculate $\mathbb{E}c(x_t, u_t)$, we only need the joint density of $(x_0, u_0, x_1, u_1, \ldots, x_t, u_t)$. Hence we have

$$\mathbb{E}c(x_t, u_t) = \int c(x_t, u_t) f_\theta(x_0, u_0, x_1, u_1, x_2, u_2, \ldots, x_t, u_t) dx_0 du_0 dx_1 du_1 \ldots dx_t du_t$$

After applying similar tricks, we can show

$$\nabla_\theta \mathbb{E}c(x_t, u_t) = \mathbb{E}\left[c(x_t, u_t) \sum_{k=0}^{t} \nabla_\theta \log \pi_\theta(u_k|x_k) \right]$$

To see what happens when we sum all terms, we write out the first few term explicitly:

$$\begin{aligned}
\nabla_\theta \mathbb{E}c(x_0, u_0) &= \mathbb{E}\left[c(x_0, u_0) \nabla_\theta \log \pi_\theta(u_0|x_0)\right] \\
\gamma \nabla_\theta \mathbb{E}c(x_1, u_1) &= \mathbb{E}\left[\gamma c(x_1, u_1) \nabla_\theta \log \pi_\theta(u_0|x_0)\right] + \mathbb{E}\left[\gamma c(x_1, u_1) \nabla_\theta \log \pi_\theta(u_1|x_1)\right] \\
\gamma^2 \nabla_\theta \mathbb{E}c(x_2, u_2) &= \mathbb{E}\left[\gamma^2 c(x_2, u_2) \nabla_\theta \log \pi_\theta(u_0|x_0)\right] + \mathbb{E}\left[\gamma^2 c(x_2, u_2) \nabla_\theta \log \pi_\theta(u_1|x_1)\right] \\
&\quad + \mathbb{E}\left[\gamma^2 c(x_2, u_2) \nabla_\theta \log \pi_\theta(u_2|x_2)\right]
\end{aligned}$$

Based on the above pattern, we can get

$$\begin{aligned}
\nabla \mathcal{C}(\theta) &= \mathbb{E} \sum_{t=0}^{\infty} \left[\Big(\sum_{k=t}^{\infty} \gamma^k c(x_k, u_k) \Big) \nabla_\theta \log \pi_\theta(u_t|x_t) \right] \\
&= \mathbb{E} \sum_{t=0}^{\infty} \left[\gamma^t \Big(\sum_{k=t}^{\infty} \gamma^{k-t} c(x_k, u_k) \Big) \nabla_\theta \log \pi_\theta(u_t|x_t) \right]
\end{aligned}$$

which gives an improved version of the policy gradient theorem. The algorithm REINFORCE is actually based on the above gradient formula.

Baseline In the actual implementation of REINFORCE, a baseline is introduced based on the following key fact:

$$\begin{aligned}\mathbb{E}\left[b(x_t)\nabla_\theta \log \pi_\theta(u_t|x_t)\right] &= \int b(x_t)\nabla_\theta \pi_\theta(u_t|x_t)p(x_t)dx_t du_t\\ &= \nabla_\theta \int b(x_t)(\int \pi_\theta(u_t|x_t)du_t)p(x_t)dx_t\\ &= \nabla_\theta \int b(x_t)p(x_t)dx_t\\ &= 0\end{aligned}$$

Therefore, the following policy gradient formula also holds

$$\nabla \mathcal{C}(\theta) = \mathbb{E}\sum_{t=0}^{\infty}\left[\gamma^t\big(\sum_{k=t}^{\infty}\gamma^{k-t}c(x_k,u_k) - b(x_t)\big)\nabla_\theta \log \pi_\theta(u_t|x_t)\right]$$

As long as the baseline does not depend on u_t, the formula works. In REINFORCE, we just choose $b(x_t)$ as the average of $\sum_{k=t}^{\infty}\gamma^{k-t}c(x_k,u_k)$ in the previous rollouts. In this case, the baseline is a constant.

Actor-Critic: Policy Gradient Theorem V2. We can further twist the above gradient formula. Based on some basic property of conditional expectation, we have

$$\begin{aligned}\mathbb{E}\left[\big(\sum_{k=t}^{\infty}\gamma^{k-t}c(x_k,u_k)\big)\nabla_\theta \log \pi_\theta(u_t|x_t)\right] &= \mathbb{E}\left[\mathbb{E}\big[\sum_{k=t}^{\infty}\gamma^{k-t}c(x_k,u_k)\nabla_\theta \log \pi_\theta(u_t|x_t)\big|\, x_t,u_t\big]\right]\\ &= \mathbb{E}\left[\mathbb{E}\big[\sum_{k=t}^{\infty}\gamma^{k-t}c(x_k,u_k)\big|\, x_t,u_t\big]\nabla_\theta \log \pi_\theta(u_t|x_t)\right]\\ &= \mathbb{E}\left[Q^\pi(x_t,u_t)\nabla_\theta \log \pi_\theta(u_t|x_t)\right]\end{aligned}$$

Therefore, we can substitute the above formula to show

$$\nabla \mathcal{C}(\theta) = \mathbb{E}\sum_{t=0}^{\infty}\left[\gamma^t Q^\pi(x_t,u_t)\nabla_\theta \log \pi_\theta(u_t|x_t)\right]$$

In the AC algorithm, we use data to simultaneously estimate Q and $\nabla \mathcal{C}(\theta)$. We can also subtract the state value function as a baseline and get

$$\nabla \mathcal{C}(\theta) = \mathbb{E}\sum_{t=0}^{\infty}\left[\gamma^t (Q^\pi(x_t,u_t) - V^\pi(x_t))\nabla_\theta \log \pi_\theta(u_t|x_t)\right]$$

The function $(Q^\pi(x_t,u_t) - V^\pi(x_t))$ is called the advantage function $A^\pi(x_t,u_t)$. Another popular way to implement AC is to estimate the advantage function using TD residual $c(x_t,u_t) + \gamma V^\pi(x_{t+1}) - V^\pi(x_t)$ and hence the critic is used to estimate V^π whose dimension is much lower than Q^π. In general, the critic step relies on some TD learning technique to estimate the value function, and then the actor step makes use of such value estimations to update the policy gradient.

Summary of Policy Gradient Theorem. Here is the key take-away. Based on the policy gradient theorem, for a discount MDP, we can estimate the policy gradient as

$$\nabla \mathcal{C}(\theta) = \mathbb{E} \sum_{t=0}^{\infty} [\gamma^t \Psi_t \nabla_\theta \log \pi_\theta(u_t|x_t)] \tag{13.7}$$

Many options for Ψ_t are available. Popular options for Ψ_t include:

- Monte Carlo estimation: $\sum_{t'=t}^{\infty} \gamma^{t'-t} c_{t'}$
- Baselined versions of Monte Carlo estimation: $\sum_{t'=t}^{\infty} (\gamma^{t'-t} c_{t'} - b(x_t))$
- State-action value function: $Q^\pi(x_t, u_t)$
- Advantage function: $A^\pi(x_t, u_t)$
- TD residual: $c_t + \gamma V^\pi(x_{t+1}) - V^\pi(x_t)$
- Generalized advantage estimation

Zeroth-order optimization. When using the policy gradient theorem, we inject noise into the control actions for exploration purposes. Suppose now we want to directly learn the gradient of a deterministic policy. Then we can use evolution strategies (or zeroth-order optimization) which does not require stochastic policy. The exploration in zeroth-order optimization is done by perturbing the policy randomly. Specifically, consider the following estimation of the policy gradient:

$$\nabla \mathcal{C}(K) \approx \frac{\mathbb{E}_{\varepsilon \sim \mathcal{N}(0,\sigma^2 I)} \mathcal{C}(K+\varepsilon)\varepsilon}{\sigma^2}$$

To understand this update rule, we analyze a shifted variant of the above update:

$$g = \frac{\mathbb{E}_{\varepsilon \sim \mathcal{N}(0,\sigma^2 I)} (\mathcal{C}(K+\varepsilon) - \mathcal{C}(K))\varepsilon}{\sigma^2} = \frac{\mathbb{E}_{\varepsilon \sim \mathcal{N}(0,I)} (\mathcal{C}(K+\sigma\varepsilon) - \mathcal{C}(K))\varepsilon}{\sigma}$$

Roughly speaking, the above estimation shifts the original zeroth-order gradient estimate with a zero mean vector and should not change the mean of the gradient estimator. Due to the fact $\lim_{\sigma \to 0} \frac{\mathcal{C}(K+\sigma\varepsilon)-\mathcal{C}(K)}{\sigma} = (\nabla \mathcal{C}(K))^\mathsf{T} \varepsilon$, we can show:

$$\begin{aligned}
\mathbb{E}_{\varepsilon \sim \mathcal{N}(0,I)} \left(\lim_{\sigma \to 0} \frac{\mathcal{C}(K+\sigma\varepsilon) - \mathcal{C}(K)}{\sigma} \right) \varepsilon &= \mathbb{E}_{\varepsilon \sim \mathcal{N}(0,I)} (\varepsilon^\mathsf{T} \nabla \mathcal{C}(K))\varepsilon \\
&= \mathbb{E}_{\varepsilon \sim \mathcal{N}(0,I)} \varepsilon (\varepsilon^\mathsf{T} \nabla C(K)) \\
&= \mathbb{E}_{\varepsilon \sim \mathcal{N}(0,I)} (\varepsilon \varepsilon^\mathsf{T}) \nabla C(K) \\
&= \nabla C(K)
\end{aligned}$$

Therefore, the zeroth-order optimization can be viewed as a stochastic version of the finite difference method. Notice that zeroth-order optimization is general since it can address any cost function. The drawback is that this approach is too general and does not exploit the problem structure of MDPs. Hence the sample efficiency of zeroth-order optimization is not very good.

Natural Policy Gradient. The problem with the policy gradient method is that it may take too many steps. Another way to view the policy gradient method is that at every step k, we approximate the cost function as

$$\mathcal{C}(\theta) \approx \mathcal{C}(\theta_k) + \nabla \mathcal{C}(\theta_k)^\mathsf{T}(\theta - \theta_k)$$

We can only trust such an first-order Taylor approximation in some small region around θ_k. Therefore, it is more reasonable to optimize the above first-order approximation subject to a constraint enforcing θ to be not too far away from θ_k. For example, if we choose the constraint as $\|\theta - \theta_k\|^2 \leq \delta$. Suppose θ_{k+1} minimizes the above first-order approximation subject to such a constraint. Based on the Lagrange theorem, we know

$$\nabla \mathcal{C}(\theta_k) + 2\lambda(\theta_{k+1} - \theta_k) = 0$$

which leads to the update rule $\theta_{k+1} = \theta_k - \frac{1}{2\lambda}\nabla \mathcal{C}(\theta_k)$. This is exactly the policy gradient method. Notice the constraint $\|\theta - \theta_k\|^2$ may not be necessarily appropriate for policy optimization since it completely ignore the curvature information. This constraint uses the Euclidean distance to measure how far away θ is from θ_k. However, the distance in policy parameters is different from the distance in policy space. It is better to define a distance metric on the policy space. One such constraint can be defined as $(\theta - \theta_k)^\mathsf{T} F(\theta_k)(\theta - \theta_k) \leq \delta$ where $F(\theta_k)$ is the so-called fisher information matrix at θ_k. The fisher information matrix is defined as

$$F(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(u_k|x_k)(\nabla_\theta \log \pi_\theta(u_k|x_k))^\mathsf{T}\right]$$

The condition $(\theta - \theta_k)^\mathsf{T} F(\theta_k)(\theta - \theta_k) \leq \delta$ roughly enforces that the KL divergence of the distributions under θ and θ_k is small. Such a constraint on the distributions in policy space is more beneficial for policy optimization. Then by Lagrange theorem, we have

$$\nabla \mathcal{C}(\theta_k) + 2\lambda F(\theta_k)(\theta_{k+1} - \theta_k) = 0$$

which can be rewritten as $\theta_{k+1} = \theta_k - \frac{1}{2\lambda}F(\theta_k)^{-1}\nabla \mathcal{C}(\theta_k)$. This is exactly the update rule for the natural policy gradient method. Natural policy gradient is faster than the policy gradient method and also improves the stability at the same time. Under the fisher information constraint, it is less likely that the properties of $\pi_{\theta_{k+1}}$ and π_{θ_k} are fundamentally different. There are still two remaining issues: 1) How to compute $F(\theta_k)^{-1}\nabla \mathcal{C}(\theta_k)$ efficiently when θ is a high dimensional vector (this is the case for deep RL)? 2) How to choose the step size for the natural policy gradient? These issues are addressed in the developments of TRPO.

Trust Region Policy Optimization TRPO is a popular method for practical deep RL problems. It can be viewed as an improved version of the natural policy gradient method.

1. How to calculate $F(\theta_k)^{-1}\nabla \mathcal{C}(\theta_k)$? Instead of inverting the Fisher information matrix, TRPO uses a conjugate gradient (CG) method to directly calculate $F(\theta_k)^{-1}\nabla \mathcal{C}(\theta_k)$. Such method is very efficient and is less costly compared with the matrix inversion operation.

2. How to choose the stepsize? Another key feature of TRPO is that it uses the line search to determine the stepsize. Suppose we want to choose α_k and make the update $\theta_{k+1} = \theta_k - \alpha_k F(\theta_k)^{-1}\nabla\mathcal{C}(\theta_k)$. We can specify some $\alpha < 1$ and search j such that $\mathcal{C}(\theta_k - \alpha^j F(\theta_k)^{-1}\nabla\mathcal{C}(\theta_k)) < \mathcal{C}(\theta_k)$. Then we can just set $\alpha_k = \alpha^j$. However, to do such a line search, we need to evaluate $\mathcal{C}(\theta_k - \alpha^j F(\theta_k)^{-1}\nabla\mathcal{C}(\theta_k))$ for multiple j. We want to skip such costly policy evaluation tasks. What can we do? The core idea of TRPO is that it introduces a surrogate cost which approximates the true cost well in some "trust region" and can be easily evaluated to facilitate the line search step. Specifically, TRPO introduces the following cost under the trust region constraint $\mathbb{E}_{\pi_{\theta_k}} D_{KL}(\pi_\theta || \pi_{\theta_k}) \leq \delta$.

$$L_{\theta_k}(\theta) = \mathbb{E}_{\pi_{\theta_k}}\left[\frac{\pi_\theta(x|u)}{\pi_{\theta_k}(x|u)} A^{\pi_{\theta_k}}(x,u)\right]$$

   Such a cost can be efficiently evaluated using data generated from π_{θ_k} and hence one can perform line search using such a surrogate cost efficiently. Importantly, the gradient of the above surrogate cost at θ_k is the same as $\nabla\mathcal{C}(\theta_k)$. Hence, with such a surrogate cost function, everything is the same as before except that we use L_{θ_k} in the line search step.

More explanations on the surrogate cost in TRPO. Now we provide more explanations for this cost. We need to use the following the relative policy performance identity:

$$\mathcal{C}(\theta) - \mathcal{C}(\theta_k) = \mathbb{E}_{\pi_\theta} \sum_{t=0}^{\infty} \gamma^t A^{\pi_{\theta_k}}(x_t, u_t)$$

Notice $\min_\theta \mathcal{C}(\theta) = \min_\theta(\mathcal{C}(\theta) - \mathcal{C}(\theta_k)) = \min_\theta \mathbb{E}_{\pi_\theta} \sum_{t=0}^{\infty} \gamma^t A^{\pi_{\theta_k}}(x_t, u_t)$. Samples from π_θ is still needed. After some calculations, one can show

$$\begin{aligned}\mathcal{C}(\theta) - \mathcal{C}(\theta_k) &= \frac{1}{1-\gamma}\mathbb{E}_{x\sim d, u\sim\pi_\theta} A^{\pi_{\theta_k}}(x,u) \\ &= \frac{1}{1-\gamma}\mathbb{E}_{x\sim d, u\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(x|u)}{\pi_{\theta_k}(x|u)} A^{\pi_{\theta_k}}(x,u)\right]\end{aligned}$$

What if we just change d to the future state distribution over π_{θ_k}? This is going to give us the surrogate cost function we want. It turns out that we can make such an approximation as long as we have the trust region constraint. This leads to the implementation of TRPO where the line search is performed on L_{θ_k} to enforce both the policy improvement and the constraint feasibility.