Lecture 12: Policy Gradient Methods

Oliver Wallscheid
Preface (1)

Shift from (indirect) value-based approaches

\[ \hat{q}(x, u, w) \approx q(x, u) \quad (12.1) \]

to (direct) policy-based solutions:

\[ \pi(u|x) = P[U = u|X = x] \approx \pi(u|x, \theta). \quad (12.2) \]

- Above, \( \theta \in \mathbb{R}^d \) is the policy parameter vector.
- Note, that \( u \) might contain multiple continuous quantities.

Goal of today's lecture

- Introduce an algorithm class based on a parameterizable policy \( \pi(\theta) \).
- Extend the action space to continuous actions.
- Combine the policy-based and value-based approach.
Fig. 12.1: Main categories of reinforcement learning algorithms (source: D. Silver, Reinforcement learning, 2016. CC BY-NC 4.0)
1. Policy Approximation and its Advantages
2. Monte Carlo Policy Gradient
3. Actor-Critic Methods (Episodic Tasks)
4. Actor-Critic Methods (Continuing Tasks)
5. Deterministic Gradient Policy
Motivating Example (1): Short-Corridor Problem

- Gridworld style problem with two actions: left (l), right (r)
- Second-left state’s action execution is reversed
- Feature representation: $\tilde{x}(x, u = r) = [1 \ 0]^T$, $\tilde{x}(x, u = l) = [0 \ 1]^T$
- $\varepsilon$-greedy value-based policy performs actions with $1 - \varepsilon/2$ probability
- A policy-based approach can search for the optimal probability split

Fig. 12.2: Short-corridor problem with $\varepsilon = 0.1$ (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018, CC BY-NC-ND 2.0)
Motivating Example (2): Strategic Gaming

Task: Two-player game of extended rock-paper-scissors
▶ A deterministic policy (i.e., value-based with given feature representation) can be easily exploited by the opponent.
▶ Conversely, a uniform random policy would be unpredictable.

Fig. 12.3: Rock paper scissors lizard Spock game mechanics
(source: www.wikipedia.org, by Director Doc CC BY-SA 4.0)
Example Policy Function: Discrete Action Space

Assumption:
- Action space is discrete and compact.

A typical policy function is:
- Soft-max in action preferences

\[
\pi(u|\mathbf{x}, \theta) = \frac{e^{h(\mathbf{x}, u, \theta)}}{\sum_i e^{h(\mathbf{x}, i, \theta)}}
\]  

(12.3)

with \( h(\mathbf{x}, u, \theta) : \mathcal{X} \times \mathcal{U} \times \mathbb{R}^d \rightarrow \mathbb{R} \) being the numerical preference per state-action pair.

- Denominator of (12.3) sums up action probabilities to one per state.
- Is designed as a stochastic policy but can approach deterministic behavior in the limit.
- The preference is parametrized via a function approximator, e.g., linear in features

\[
h(\mathbf{x}, u, \theta) = \theta^T \tilde{x}(\mathbf{x}, u).
\]  

(12.4)
Example Policy Function: Continuous Action Space (1)

Assumption:

- Action space is continuous and there is only one scalar action $u \in \mathbb{R}$.

A typical policy function is:

- Gaussian probability density

$\pi(u|x, \theta) = \frac{1}{\sigma(x, \theta) \sqrt{2\pi}} \exp\left(-\frac{(u - \mu(x, \theta))^2}{2\sigma(x, \theta)^2}\right)$  

(12.5)

with mean $\mu(x, \theta) : \mathcal{X} \times \mathbb{R}^d \rightarrow \mathbb{R}$ and standard deviation $\sigma(x, \theta) : \mathcal{X} \times \mathbb{R}^d \rightarrow \mathbb{R}$ given by parametric function approximation.

Variants regarding function $\mu$ and $\sigma$:

1. Both share a mutual parameter set $\theta$ (e.g., artificial neural network with multiple outputs).

2. Both are parametrized independently $\theta = [\theta_{\mu} \theta_{\sigma}]^T$ (e.g., by two linear regression functions).

3. Only $\mu(x, \theta)$ is parametrized while $\sigma$ is scheduled externally.
Example Policy Function: Continuous Action Space (2)

- Output of the functions $\mu_k = (x_k, \theta_k)$ and $\sigma_k = (x_k, \theta_k)$ can change in every time step.
- Depending on $\sigma$ exploration is an inherent part of the (stochastic) policy.

![Graph of univariate Gaussian probability density functions](Image)

**Fig. 12.4:** Exemplary univariate Gaussian probability density functions (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018, CC BY-NC-ND 2.0)
Assumption:

- Action space is continuous and there are multiple actions $u \in \mathbb{R}^m$.

A typical policy function is:

- **Multivariate Gaussian probability density**

$$
\pi(u|x, \theta) = \frac{1}{\sqrt{(2\pi)^m \det(\Sigma)}} \exp \left( -\frac{1}{2} (u - \mu)^T \Sigma^{-1} (u - \mu) \right)
$$

(12.6)

with mean $\mu(x, \theta) : \mathcal{X} \times \mathbb{R}^d \rightarrow \mathbb{R}^m$ and covariance matrix $\Sigma(x, \theta) : \mathcal{X} \times \mathbb{R}^d \rightarrow \mathbb{R}^{m \times m}$ given by parametric function approximation.

- Same parametrization variants apply to $\mu$ and $\Sigma$ as in the scalar action case.

- In addition, $\Sigma$ can be considered a diagonal matrix and clipped to reduce complexity as well as ensure nonsingularity.
Below we find an example for

\[ \mu = [-0.4 \ 0.3]^T \quad \text{and} \quad \Sigma = \begin{bmatrix} 0.04 & 0 \\ 0 & 0.02 \end{bmatrix}. \]

*Fig. 12.5: Exemplary bivariate Gaussian probability density function*
Policy Objective Function

- Goal: find optimal $\theta^*$ given the policy $\pi(u|x, \theta)$.
- Problem: which measure of optimality should we use?

Possible optimality metrics:

- **Start state value** (in episodic tasks):

  $$J(\theta) = v_{\pi\theta}(x_0) = \mathbb{E}[v|X = x_0, \theta] \quad (12.7)$$

- **Average reward** (in continuing tasks):

  $$J(\theta) = \bar{r}_{\pi\theta} = \int_X \mu_{\pi}(x) \int_U \pi(u|x, \theta) \int_{X,R} p(x', r|x, u)r \quad (12.8)$$

- Above, $\mu_{\pi}(x)$ is again the steady-state distribution

  $$\mu_{\pi}(x) = \lim_{k \to \infty} \mathbb{P}[X_k = x|U_{0:k-1} \sim \pi]$$.
In essence, policy-based RL is an optimization problem. Depending on the policy function and task, finding $\theta^*$ might be a non-linear, multidimensional and non-stationary problem. Hence, we might consider global optimization techniques\(^1\) like Simple heuristics: random search, grid search,... Meta-heuristics: evolutionary algorithms, particle swarm,... Surrogate-model-based optimization: Bayes opt,... Gradient-based techniques with multi-start initialization.

We will focus on gradient-based methods (policy gradient).

Hence, we will assume that the gradient

$$\nabla_\theta J(\theta) = \left[ \frac{\partial J}{\partial \theta_1} \cdots \frac{\partial J}{\partial \theta_d} \right]^T$$

required for gradient ascent optimization always exists:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta).$$

True gradient $\nabla_\theta J(\theta)$ is usually approximated, e.g., by stochastic gradient descent (SGD) or derived variants.

Fig. 12.6: Exemplary optimization paths for (stochastic) gradient ascent (derivative work of www.wikipedia.org, CC0 1.0)
Theorem 12.1: Policy Gradient

Given a metric \( J(\theta) \) for the undiscounted episodic (12.7) or continuing tasks (12.8) and a parameterizable policy \( \pi(u|x, \theta) \) the policy gradient is

\[
\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi} \left[ q_{\pi}(x, u) \frac{\nabla_{\theta} \pi(u|x, \theta)}{\pi(u|x, \theta)} \right].
\]  

(12.9)

- Having samples \( \langle x_i, u_i \rangle \), an estimate of \( q_{\pi} \) and the policy function \( \pi(\theta) \) available we receive an analytical solution for the policy gradient!

- Using identity \( \nabla \ln a = \frac{\nabla a}{a} \) we can re-write to

\[
\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi} \left[ q_{\pi}(x, u) \nabla_{\theta} \ln \pi(u|x, \theta) \right] \]  

(12.10)

with \( \nabla_{\theta} \ln \pi(u|x, \theta) \) also called the score function.

Intuitive Interpretation of Policy Parameter Update

- Inserting the policy gradient theorem into gradient ascent approach:
  \[ \theta \leftarrow \theta + \alpha \mathbb{E}_\pi \left[ q_\pi(x, u) \frac{\nabla_\theta \pi(u|x, \theta)}{\pi(u|x, \theta)} \right] \].

- Move in the direction that favor actions that yield an increased value.
- Scale the update step size inversely to the action probability to compensate that some actions are selected more frequently.

Also note:
- The policy gradient is not depending on the state distribution!
- Hence, we do not need any knowledge of the environment and receive a model-free RL approach!
Simple Score Function Examples

Soft-max policy with linear function approximation:

\[ \pi(u|x, \theta) = \frac{e^{\theta^T \tilde{x}(x,u)}}{\sum_i e^{\theta^T \tilde{x}(x,i)}} \]

\[ \Leftrightarrow \nabla_\theta \ln \pi(u|x, \theta) = \nabla_\theta \left( \theta^T \tilde{x}(x, u) - \ln \left( \sum_i e^{\theta^T \tilde{x}(x,i)} \right) \right) \]

\[ = \tilde{x}(x, u) - \mathbb{E}_\pi [\tilde{x}(x, \cdot)] \]

Univariate Gaussian policy with linear function approximation and given \( \sigma \):

\[ \pi(u|x, \theta) = \frac{1}{\sigma \sqrt{2\pi}} \exp \left( - \frac{(u - \theta^T \tilde{x}(x, u))^2}{2\sigma^2} \right) \]

\[ \Leftrightarrow \nabla_\theta \ln \pi(u|x, \theta) = \nabla_\theta \left( \ln \left( \frac{1}{\sigma \sqrt{2\pi}} \right) - \frac{(u - \theta^T \tilde{x}(x, u))^2}{2\sigma^2} \right) \]

\[ = \frac{(u - \theta^T \tilde{x}(x, u))}{\sigma^2} \tilde{x}(x, u) \]
Pro and Cons: Policy vs. Value-Based Approaches

Pro value-based solutions (e.g., Q-learning):
▶ Estimated value is an intuitive performance metric.
▶ Considered sample-efficient (cf. replay buffer or bootstrapping).

Pro policy-based solutions (e.g., using policy gradient):
▶ Policy gradient theorem provides strong convergence properties.
▶ Seamless integration of stochastic and dynamic policies.
▶ Straightforward applicable to large/continuous action spaces. In contrast, value-based approaches would require explicit optimization

\[ u^* = \arg \max_u q(x, u, w). \]

Mutual hassle:
▶ Gradient-based optimization with (non-linear) function approximation is likely to deliver only suboptimal and local policy optima.
Basic Concept

Initial situation:

- Score function $\nabla_\theta \ln \pi(u|x, \theta)$ can be calculated analytically using suitable policy and chain rule (e.g., by algorithmic differentiation).
- Open question: how to retrieve $q_\pi(x, u)$ in
  \[
  \nabla_\theta J(\theta) = \mathbb{E}_\pi [q_\pi(x, u) \nabla_\theta \ln \pi(u|x, \theta)] \]

Monte Carlo policy gradient:

- Use sampled episodic return $g_k$ to approximate $q_\pi(x, u)$:
  \[
  q_\pi(x, u) \approx g_k \\
  \theta_{k+1} = \theta_k + \alpha \gamma^k g_k \nabla_\theta \ln \pi(u_k|x_k, \theta_k).
  \]
- The discounting of the policy gradient is introduced as an extension to Theo. 12.1 (which assumed an undiscounted episodic task).
- Also known as *REINFORCE* approach.
Algorithmic Implementation: Monte Carlo Policy Gradient

- Usual technical convergence requirements regarding $\alpha$ apply.
- Use sampled return as unbiased estimate of $q$.
- Recall previous MC-based methods: high variance, slow learning.

```
input: a differentiable policy function $\pi(u|x, \theta)$
parameter: step size $\alpha \in \{\mathbb{R} | 0 < \alpha < 1\}$
init: parameter vector $\theta \in \mathbb{R}^d$ arbitrarily
for $j = 1, 2, \ldots, \text{episodes}$ do
  generate an episode following $\pi(\cdot|\cdot, \theta)$: $x_0, u_0, r_1, \ldots, x_T$
  for $k = 0, 1, \ldots, T - 1 \text{ time steps}$ do
    $g \leftarrow \sum_{i=k+1}^{T} \gamma^{i-k-1} r_i$;
    $\theta \leftarrow \theta + \alpha \gamma^k g \nabla_\theta \ln \pi(u_k|x_k, \theta)$;
```

**Algo. 12.1:** Monte Carlo policy gradient (output: parameter vector $\theta^*$ for $\pi^*(u|x, \theta^*)$)
REINFORCE Example: Short-Corridor Problem

Algorithm parameter: step size $\alpha > 0$

$G_0$
Total reward on episode averaged over 100 runs

$\nu_*(s_0)$

$\alpha = 2^{-13}$

$\alpha = 2^{-14}$

$\alpha = 2^{-12}$

Fig. 12.7: Comparison of Monte Carlo policy gradient approach on short-corridor problem from Fig. 12.2 for different learning rates (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018, CC BY-NC-ND 2.0)
Motivation: add a comparison term to the policy gradient to reduce variance while not affecting its expectation.

Introduce the baseline $b(x)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi [(q_\pi(x, u) - b(x)) \nabla_\theta \ln \pi(u|x, \theta)].$$  \hspace{1cm} (12.11)

Since $b(x)$ is only depending on the state but not on the actions/policy we did not change the policy gradient in expectation:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi [q_\pi(x, u) \nabla_\theta \ln \pi(u|x, \theta)] - \mathbb{E}_\pi \left[ b(x) \nabla_\theta \ln \pi(u|x, \theta) \right] = 0$$

Consequently, the Monte Carlo policy parameter update yields:

$$\theta_{k+1} = \theta_k + \alpha \gamma^k \left( g_k - b(x_k) \right) \nabla_\theta \ln \pi(u_k|x_k, \theta_k).$$
Advantage Function

- Intuitive choice of the baseline is the state value \( b(x) = v_\pi(x) \).
- The resulting policy gradient becomes

\[
\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[ (q_\pi(x, u) - v_\pi(x)) \nabla_\theta \ln \pi(u|x, \theta) \right]. \tag{12.12}
\]

- Here, the difference between action and state value is the advantage function

\[
a_\pi(x, u) = q_\pi(x, u) - v_\pi(x). \tag{12.13}
\]

- Interpretation: value difference taking (arbitrary) action \( u \) and thereafter following policy \( \pi \) compared to the state value following same policy (i.e., choosing \( u \sim \pi \)) given the state.
- Hence, we might rewrite to:

\[
\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[ a_\pi(x, u) \nabla_\theta \ln \pi(u|x, \theta) \right]. \tag{12.14}
\]
Implementation requires approximation $b(x) \approx \hat{v}(x, w)$.

Hence, we are learning two parameter sets $\theta$ and $w$.

Keep using sampled return as action-value estimate: $q_\pi(x, u) \approx g_k$.

**Algo. 12.2: Monte Carlo policy gradient with baseline**

| input: a differentiable policy function $\pi(u|x, \theta)$ |
| input: a differentiable state-value function $\hat{v}(x, w)$ |
| parameter: step sizes $\{\alpha_w, \alpha_\theta\} \in \mathbb{R}|0 < \alpha < 1|$ |
| init: parameter vectors $w \in \mathbb{R}^\zeta$ and $\theta \in \mathbb{R}^d$ arbitrarily |

for $j = 1, 2, \ldots$, episodes do

generate an episode following $\pi(\cdot|\cdot, \theta)$: $x_0, u_0, r_1, \ldots, x_T$;

for $k = 0, 1, \ldots, T - 1$ time steps do

$g \leftarrow \sum_{i=k+1}^{T} \gamma^{i-k-1} r_i$;

$\delta \leftarrow g - \hat{v}(x_k, w)$;

$w \leftarrow w + \alpha_w \delta \nabla_w \hat{v}(x_k, w)$;

$\theta \leftarrow \theta + \alpha_\theta \gamma^k \delta \nabla_\theta \ln \pi(u_k|x_k, \theta)$;

Algo. 12.2: Monte Carlo policy gradient with baseline (output: parameter vector $\theta^*$ for $\pi^*(u|x, \theta^*)$) and $w^*$ for $\hat{v}^*(x, w^*)$)
REINFORCE Comparison w/o Baseline

Because REINFORCE is a Monte Carlo method for learning the policy parameter, \( \theta \), it seems natural to also use a Monte Carlo method to learn the state-value weights, \( w \).

A complete pseudocode algorithm for REINFORCE with baseline using such a learned state-value function as the baseline is given in the box below.

REINFORCE with Baseline (episodic), for estimating \( \pi \)

**Input:** a differentiable policy parameterization \( \pi(\cdot|\cdot, \theta) \)

**Input:** a differentiable state-value function parameterization \( \hat{v}(s, w) \)

**Algorithm parameters:** step sizes \( \alpha_\theta > 0, \alpha_w > 0 \)

1. Initialize policy parameter \( \theta \) and state-value weights \( w \) (e.g., to 0)
2. Loop forever (for each episode):
   - Generate an episode \( S_0, A_0, R_1, ..., S_T, A_T, R_T \) following \( \pi(\cdot|\cdot, \theta) \)
   - Loop for each step of the episode \( t = 0, 1, ..., T \):
     - \( G_P^t = R_t - \hat{v}(S_t, w) \)
     - \( w \leftarrow w + \alpha_w \hat{v}(S_t, w) \)
     - \( \theta \leftarrow \theta + \alpha_\theta \ln \pi(A_t|S_t, \theta) \)

This algorithm has two step sizes, denoted \( \alpha_\theta \) and \( \alpha_w \) (where \( \alpha_\theta \) is the \( \frac{1}{E} \) in (13.11)). Choosing the step size for values (here \( \alpha_w \)) is relatively easy; in the linear case we have rules of thumb for setting it, such as \( \alpha_w = \frac{1}{E} \hat{v}(S_t, w) \)\( k^2 \mu_\pi \) (see Section 9.6). It is much less clear how to set the step size for the policy parameters, \( \alpha_\theta \), whose best value depends on the range of variation of the rewards and on the policy parameterization.

Fig. 12.8: Comparison of Monte Carlo policy gradient on short-corridor problem from Fig. 12.2 where both algorithms’ learning rates have been tuned (source: R. Sutton and G. Barto, Reinforcement learning: an introduction, 2018, CC BY-NC-ND 2.0)
1 Policy Approximation and its Advantages
2 Monte Carlo Policy Gradient
3 Actor-Critic Methods (Episodic Tasks)
4 Actor-Critic Methods (Continuing Tasks)
5 Deterministic Gradient Policy
General Actor-Critic Idea

Conclusion of Monte Carlo policy gradient with baseline:

- Will learn an unbiased policy gradient.
- As the other MC-based methods: learns slowly due to high variance.
- Updates only available after full episodes.

Alternative: use an additional function approximator, the so-called critic, to estimate $q_\pi$ (i.e., approximate policy gradient):

\[
\begin{align*}
    v(x) &\approx \hat{v}(x, w_v), \\
    q(x, u) &\approx \hat{q}(x, u, w_q), \\
    a(x, u) &\approx \hat{q}(x, u, w_q) - \hat{v}(x, w_v).
\end{align*}
\]

- Realization: any prediction tool discussed so far (TD(0), LSTD,...).
- Potential: we can use online step-by-step updates to estimate $\hat{q}$.
- Disadvantage: we would train two value estimates by $w_v$ and $w_q$. 
The TD error is
\[ \delta_\pi = r + \gamma v_\pi(x') - v_\pi(x). \] (12.15)

In expectation the TD error is equivalent to the advantage function
\[ \mathbb{E}_{\pi} [\delta_\pi | x, u] = \mathbb{E}_{\pi} [r + \gamma v_\pi(x') | x, u] - v_\pi(x) \]
\[ = q_\pi(x, u) - v_\pi(x) \]
\[ = a_\pi(x, u). \] (12.16)

Hence, the TD error can be used to calculate the policy gradient:
\[ \nabla_\theta J(\theta) = \mathbb{E}_{\pi} [\delta_\pi \nabla_\theta \ln \pi(u | x, \theta)]. \] (12.17)

This results in requiring only one function parameter set:
\[ \delta_\pi \approx r + \gamma \hat{v}_\pi(x', w) - \hat{v}_\pi(x, w). \] (12.18)
Actor-Critic Structure

- Critic (policy evaluation) and actor (policy improvement) can be considered another form of generalized policy iteration (GPI).
- Online and on-policy algorithm for discrete and continuous action spaces with built-in exploration by stochastic policy functions.

Fig. 12.9: Simplified flow diagram of actor-critic-based RL
Analog to MC-based policy gradient optional discounting on the gradient updates is introduced.

**input:** a differentiable policy function $\pi(u|x, \theta)$

**input:** a differentiable state-value function $\hat{v}(x, w)$

**parameter:** step sizes $\{\alpha_w, \alpha_\theta\} \in \{\mathbb{R}|0 < \alpha < 1\}$

**init:** parameter vectors $w \in \mathbb{R}^\zeta$ and $\theta \in \mathbb{R}^d$ arbitrarily

**for** $j = 1, 2, \ldots$, episodes **do**

- initialize $x_0$;

  **for** $k = 0, 1, \ldots, T - 1$ time steps **do**

    - apply $u_k \sim \pi(\cdot|x_k, \theta)$ and observe $x_{k+1}$ and $r_{k+1}$;
    - $\delta \leftarrow r_{k+1} + \gamma \hat{v}(x_{k+1}, w) - \hat{v}(x_k, w)$;
    - $w \leftarrow w + \alpha_w \delta \nabla_w \hat{v}(x_k, w)$;
    - $\theta \leftarrow \theta + \alpha_\theta \gamma^k \delta \nabla_\theta \ln \pi(u_k|x_k, \theta)$;

**Algo. 12.3:** Actor-critic for episodic tasks using TD(0) targets (output: parameter vector $\theta^*$ for $\pi^*(u|x, \theta^*)$) and $w^*$ for $\hat{v}^*(x, w^*)$)
Actor-Critic Generalization

- Using the TD(0) error as the target to train the critic is convenient.
- However, the usual alternatives can be applied to train $\hat{v}(x, w)$.
- $n$-step bootstrapping:

$$
v(x_k) \approx r_{k+1} + \gamma r_{k+2} + \cdots + \gamma^{n-1} r_{k+n} + \gamma^n \hat{v}_{k+n-1}(x_{k+n}, w).
$$

- $\lambda$-return (forward view):

$$
v(x_k) \approx (1 - \lambda) \sum_{n=1}^{T-k-1} \lambda^{(n-1)} g_{k:k+n} + \lambda^{T-k-1} g_k.
$$

- TD($\lambda$) using eligibility traces (backward view):

$$
z_k = \gamma \lambda z_{k-1} + \nabla_w \hat{v}(x_k, w_k),
$$
$$
\delta_k = r_{k+1} + \gamma \hat{v}(x_{k+1}, w_k) - \hat{v}(x_k, w_k).
$$
Algo. Implementation: Actor-Critic with TD($\lambda$) Targets

**input:** a differentiable policy function $\pi(u|x, \theta)$
**input:** a differentiable state-value function $\hat{v}(x, w)$
**parameter:** $\{\alpha_w, \alpha_\theta\} \in \{\mathbb{R}|0 < \alpha < 1\}$, $\{\lambda_w, \lambda_\theta\} \in \{\mathbb{R}|0 \leq \lambda \leq 1\}$
**init:** parameter vectors $w \in \mathbb{R}^\zeta$ and $\theta \in \mathbb{R}^d$ arbitrarily

for $j = 1, 2, \ldots$, episodes do
    initialize $x_0$, $z_w = 0$, $z_\theta = 0$;
    for $k = 0, 1, \ldots, T - 1$ time steps do
        apply $u_k \sim \pi(\cdot|x_k, \theta)$ and observe $x_{k+1}$ and $r_{k+1}$;
        $\delta \leftarrow r_{k+1} + \gamma \hat{v}(x_{k+1}, w) - \hat{v}(x_k, w)$;
        $z_w \leftarrow \gamma \lambda_w z_w + \nabla_w \hat{v}(x_k, w)$;
        $z_\theta \leftarrow \gamma \lambda_\theta z_\theta + \gamma^k \nabla_\theta \ln \pi(u_k|x_k, \theta)$;
        $w \leftarrow w + \alpha_w \delta z_w$;
        $\theta \leftarrow \theta + \alpha_\theta \delta z_\theta$;

Algo. 12.4: Actor-critic for episodic tasks using TD($\lambda$) targets (output: parameter vector $\theta^*$ for $\pi^*(u|x, \theta^*)$) and $w^*$ for $\hat{v}^*(x, w^*)$)
Most default prediction techniques will add a bias to $\hat{q}(w) \approx q$.

A biased policy gradient may not find the right solution.

However, following the below theorem we can prevent any bias.

**Theorem 12.2: Compatible Function Approximation**

If the value function approximator is compatible to the policy

$$\hat{q}(x, u, w) = (\nabla_{\theta} \ln \pi_{\theta}(x, u))^{T} w$$

(12.19)

and the value function parameters $w$ minimize the mean-squared error

$$w^* = \arg \max_{w} \mathbb{E} [q_\pi(x, u) - \hat{q}(x, u, w)]^2$$

(12.20)

then the policy gradient using $\hat{q}(w)$ is exact.
Compatible Function Approximation (2)

Interpretation:

- The policy gradient $\nabla_\theta \ln \pi_\theta(x, u)$ delivers input features for a linear mapping of the value function approximation:

$$\hat{q}(x, u, w) = (\nabla_\theta \ln \pi_\theta(x, u))^T w.$$ 

- The unknown parameters $w$ are the solution to a linear regression problem estimating $q_\pi(x, u)$:

$$w^* = \arg \max_w \mathbb{E} [q_\pi(x, u) - \hat{q}(x, u, w)]^2.$$ 

- The latter condition may be solved using a batch LSTD approach or relaxed in favor of policy evaluation algorithms using TD learning.
<table>
<thead>
<tr>
<th></th>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Policy Approximation and its Advantages</td>
</tr>
<tr>
<td>2</td>
<td>Monte Carlo Policy Gradient</td>
</tr>
<tr>
<td>3</td>
<td>Actor-Critic Methods (Episodic Tasks)</td>
</tr>
<tr>
<td>4</td>
<td>Actor-Critic Methods (Continuing Tasks)</td>
</tr>
<tr>
<td>5</td>
<td>Deterministic Gradient Policy</td>
</tr>
</tbody>
</table>
Introducing Average Rewards for Continuing Tasks

For continuing tasks the differential reward is used as the performance

\[ J(\theta) = \bar{r}_\pi = \lim_{h \to \infty} \frac{1}{h} \sum_{k=1}^{h} \mathbb{E} \left[ R_k \mid X_0, U_{0:k-1} \sim \pi \right] , \]

(12.21)

\[ = \lim_{k \to \infty} \mathbb{E} \left[ R_k \mid X_0, U_{0:k-1} \sim \pi \right] . \]

Consequently, the value definitions are adapted using the differential return:

\[ G_k = R_{k+1} - \bar{r}_\pi + R_{k+2} - \bar{r}_\pi + R_{k+3} - \bar{r}_\pi + \ldots , \]

\[ v_\pi(x) = \mathbb{E}_\pi \left[ G_k \mid X_k = x \right] , \]

(12.22)

\[ q_\pi(x, u) = \mathbb{E}_\pi \left[ G_k \mid X_k = x, U_k = u \right] . \]

With these modifications the policy gradient theorem (12.9) still holds. Although we do not have to consider discounting anymore.
The episodic actor-critic approach can be directly transferred to continuing tasks.

The critic is estimating the differential state value from (12.22):

\[ v_\pi(x) \approx \hat{v}(x, w). \] (12.23)

The target as the basis for estimating the state value is again flexible: TD(0), TD(\lambda), ... .

The differential TD error is also approximated straightforwardly:

\[ \delta_\pi \approx r - \hat{r} + \hat{v}(x', w) - \hat{v}(x, w). \] (12.24)

The policy parameter update is then:

\[ \theta_{k+1} = \theta_k + \alpha \delta_k \nabla \theta \ln \pi(u_k|x_k, \theta_k). \] (12.25)
Implementation: Actor-Critic with Diff. TD(0) Targets

**input:** a differentiable policy function $\pi(u|x, \theta)$

**input:** a differentiable state-value function $\hat{v}(x, w)$

**parameter:** step sizes $\{\alpha_w, \alpha_\theta, \beta\} \in \{\mathbb{R}|0 < \alpha, \beta < 1\}$

**init:** parameter vectors $w \in \mathbb{R}^\zeta$ and $\theta \in \mathbb{R}^d$ arbitrarily

**init:** avg. return estimate $\hat{r} \in \mathbb{R}$, starting state $x_0$

**for** $k = 0, 1, \ldots$ **time steps** **do**

- apply $u_k \sim \pi(\cdot|x_k, \theta)$ and observe $x_{k+1}$ and $r_{k+1}$;
- $\delta \leftarrow r_{k+1} - \hat{r} + \hat{v}(x_{k+1}, w) - \hat{v}(x_k, w)$;
- $\hat{r} \leftarrow \hat{r} + \beta \delta$;
- $w \leftarrow w + \alpha_w \delta \nabla_w \hat{v}(x_k, w)$;
- $\theta \leftarrow \theta + \alpha_\theta \delta \nabla_\theta \ln \pi(u_k|x_k, \theta)$;

**Algo. 12.5:** Actor-critic for continuing tasks using diff. TD(0) targets

(output: parameter vector $\theta^*$ for $\pi^*(u|x, \theta^*)$) and $w^*$ for $\hat{v}^*(x, w^*)$)
Implementation: Actor-Critic with Diff. TD(\(\lambda\)) Targets

\begin{verbatim}
input: a differentiable policy function \(\pi(u| x, \theta)\)
input: a differentiable state-value function \(\hat{v}(x, w)\)
parameter: step sizes \(\{\alpha_w, \alpha_\theta, \beta\} \in \{\mathbb{R} | 0 < \alpha, \beta < 1\}\)
parameter: trace decay rates \(\{\lambda_w, \lambda_\theta\} \in \{\mathbb{R} | 0 \leq \lambda \leq 1\}\)
init: parameter vectors \(w \in \mathbb{R}^\zeta\) and \(\theta \in \mathbb{R}^d\) arbitrarily
init: avg. return estimate \(\hat{r} \in \mathbb{R}\), starting state \(x_0, z_w = 0, z_\theta = 0\)
for \(k = 0, 1, \ldots\) time steps do
    apply \(u_k \sim \pi(\cdot|x_k, \theta)\) and observe \(x_{k+1}\) and \(r_{k+1}\);
    \(\delta \leftarrow r_{k+1} - \hat{r} + \hat{v}(x_{k+1}, w) - \hat{v}(x_k, w)\);
    \(\hat{r} \leftarrow \hat{r} + \beta \delta\);
    \(z_w \leftarrow \lambda_w z_w + \nabla_w \hat{v}(x_k, w)\);
    \(z_{\theta} \leftarrow \lambda_d z_\theta + \nabla_\theta \ln \pi(u_k|x_k, \theta)\);
    \(w \leftarrow w + \alpha_w \delta z_w\);
    \(\theta \leftarrow \theta + \alpha_\theta \delta z_\theta\);
\end{verbatim}

Algo. 12.6: Actor-critic for continuing tasks using differential TD(\(\lambda\)) targets with eligibility traces (output: parameter vector \(\theta^*\) for \(\pi^*(u|x, \theta^*)\) and \(w^*\) for \(\hat{v}^*(x, w^*)\))
Table of Contents

1. Policy Approximation and its Advantages
2. Monte Carlo Policy Gradient
3. Actor-Critic Methods (Episodic Tasks)
4. Actor-Critic Methods (Continuing Tasks)
5. Deterministic Gradient Policy
Background and Motivation

Recap on policy gradient so far:

- The previously discussed policy functions and the policy gradient theorem were assuming stochastic policies.
- The resulting on-policy algorithms may not provide top-class learning performance:
  - Non-guided exploration with step-by-step updates and
  - Greedy actions only in the limit (i.e., infeasible long learning).

The alternative:

- Apply a deterministic policy with separate exploration.
- Enable off-policy learning (with experience replay as a possible extension).
- Hence, we will focus on a deterministic policy function

\[
\pi(x, \theta) = \mu(x, \theta). \quad (12.26)
\]
Theorem 12.3: Deterministic Policy Gradient

Given a metric $J(\theta)$ for the undiscounted episodic (12.7) or continuing tasks (12.8) and a parameterizable policy $\mu(x, \theta)$ the deterministic policy gradient is

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\mu} \left[ \nabla_{\theta} \mu(x, \theta) \nabla_u q(x, u) | u = \mu(x) \right].$$  \hspace{1cm} (12.27)

- Again, $q$ needs to be approximated using samples, e.g., implementing a critic via TD learning.
- Bias problem applies as in the stochastic case and can be compensated using compatible function approximation w.r.t. $\hat{q}$.
- It turns out that (12.27) is also (approximately) valid in the off-policy case, i.e., if the sample distribution is obtained from a behavior policy.
- Proof can be found in D. Silver et al., *Deterministic Policy Gradient Algorithms*, International Conference on Machine Learning, 2014.
Exploration with a Deterministic Policy

- If the DPG approach is applied on-policy there is no inherent exploration.

- How to learn something?
  - The environment itself is sufficiently noisy (random impacts, measurement noise).
  - Or we have to add noise to the actions, i.e., making the approach off-policy.
  - Hence, utilizing a behavior policy is also possible.

- That additional action noise could be:
  - Simple Gaussian noise or
  - a shaped noise process like a discrete-time Ornstein-Uhlenbeck (OU) process

\[
\nu_{k+1} = \lambda \nu_k + \sigma \epsilon_k
\]

where \( \nu_k \) is the OU noise output, \( 0 < \lambda < 1 \) is a smoothing factor and \( \sigma \) is the variance scaling a standard Gaussian sequence (no mean) \( \epsilon_k \).
Algo. Implementation: Deterministic Actor-Critic (1)

**input:** a differentiable deterministic policy function \( \mu(x, \theta) \)

**input:** a differentiable action-value function \( \hat{q}(x, u, w) \)

**parameter:** step sizes \( \{\alpha_w, \alpha_\theta\} \in \mathbb{R} | 0 < \alpha < 1 \)

**init:** parameter vectors \( w \in \mathbb{R}^\zeta \) and \( \theta \in \mathbb{R}^d \) arbitrarily

**for** \( j = 1, 2, \ldots, \) episodes **do**

  initialize \( x_0 \);

  **for** \( k = 0, 1, \ldots, T - 1 \) time steps **do**

    \( u_k \leftarrow \) apply from \( \mu(x_k, \theta) \) w/wo noise or from behavior policy;

    observe \( x_{k+1} \) and \( r_{k+1} \);

    choose \( u' \) from \( \mu(x_{k+1}, \theta) \);

    \( \delta \leftarrow r_{k+1} + \gamma \hat{q}(x_{k+1}, u', w) - \hat{q}(x_k, u_k, w) \);

    \( w \leftarrow w + \alpha_w \delta \nabla_w \hat{q}(x_k, u_k, w) \);

    \( \theta \leftarrow \theta + \alpha_\theta \gamma^k \nabla_\theta \mu(x_k, \theta) \nabla_u \hat{q}(x_k, u_k, w) \big|_{u=\mu(x)} \);

**Algo. 12.7:** Deterministic actor-critic for episodic tasks using Sarsa(0) targets applicable on- and off-policy (output: parameter vector \( \theta^* \) for \( \mu^*(x, \theta^*) \)) and \( w^* \) for \( \hat{q}^*(x, u, w^*) \))
**Algorithm Implementation: Deterministic Actor-Critic (2)**

**Input:** a differentiable deterministic policy function $\mu(x, \theta)$

**Input:** a differentiable action-value function $\hat{q}(x, u, w)$

**Parameter:** step sizes $\{\alpha_w, \alpha_\theta, \beta\} \in \{\mathbb{R}|0 < \alpha, \beta < 1\}$

**Init:** parameter vectors $w \in \mathbb{R}^{\zeta}$ and $\theta \in \mathbb{R}^{d}$ arbitrarily

**Init:** avg. return estimate $\hat{r} \in \mathbb{R}$, starting state $x_0$

For $k = 0, 1, \ldots$ time steps do

- $u_k \leftarrow$ apply from $\mu(x_k, \theta)$ w/wo noise or from behavior policy;
- observe $x_{k+1}$ and $r_{k+1}$;
- choose $u'$ from $\mu(x_{k+1}, \theta)$;
- $\delta \leftarrow r_{k+1} - \hat{r} + \hat{q}(x_{k+1}, u', w) - \hat{q}(x_k, u_k, w)$;
- $\hat{r} \leftarrow \hat{r} + \beta \delta$;
- $w \leftarrow w + \alpha_w \delta \nabla w \hat{q}(x_k, u_k, w)$;
- $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \mu(x_k, \theta) \nabla_u \hat{q}(x_k, u_k, w)|_{u=\mu(x)}$;

**Algo. 12.8:** Deterministic actor-critic for continuing tasks using Sarsa(0) targets applicable on- and off-policy (output: parameter vector $\theta^*$ for $\mu^*(x, \theta^*)$) and $w^*$ for $\hat{q}^*(x, u, w^*)$)
Exemplary Comparison to Stochastic Policy Gradient

- DPG-based approach uses compatible function approximation, i.e., suitable linear $\hat{q}$ estimation. A fixed Gaussian behavior policy is applied for exploration.
- SAC uses a Gaussian policy with linear function approximation.

Fig. 12.10: Comparison of stochastic on-policy actor-critic (SAC), stochastic off-policy actor-critic (OffPAC), and deterministic off-policy actor-critic (COPDAC) on continuous-action reinforcement learning (source: D. Silver et al., *Deterministic Policy Gradient Algorithms*, International Conference on Machine Learning, 2014)
Policy-based methods are a new class within the RL toolbox.
- Instead of learning a policy indirectly from a value the policy is directly parametrized.
- The policy function allows discrete and continuous actions with inherent stochastic exploration.

Solving the underlying optimization task is complex. However, the policy gradient theorem provides a suitable theoretical baseline for gradient-based optimization.

Anyhow, to calculate policy gradients we require a value estimate.
- Monte Carlo prediction is straightforward, but comes with high variance and slow learning.
- Adding a state-dependent baseline comparison does not change the policy gradient in expectation but enables decreasing the variance.

Extending this idea naturally leads to integrating a critic network, i.e., an additional function approximation to estimate the value.

The critic can be fed by the usual targets (TD(0), TD(\(\lambda\)),...).
Thanks for your attention and have a nice week!