Lecture 13: Further Contemporary RL Algorithms

Oliver Wallscheid
Table of Contents

1. Deep Deterministic Policy Gradient (DDPG)
2. Twin Delayed Deep Deterministic Policy Gradient (TD3)
3. Trust Region Policy Optimization (TRPO)
4. Proximal Policy Optimization (PPO)
The upcoming deep deterministic policy gradient (DDPG) algorithm was very much inspired by the successes of DQNs (cf. Algo. 10.8 and landmark paper by Mnih et al.) on discrete action spaces.

However, DQNs are not directly applicable to (quasi-)continuous action spaces.

Recall the incremental $Q$-learning equation using function approximation

$$ w \leftarrow w + \alpha \left[ r + \gamma \max_u \hat{q}(x', u, w) - \hat{q}(x, u, w) \right] \nabla_w \hat{q}(x, u, w). $$

For every policy inference and updating step we need to find $\max_u \hat{q}(x', u, w)$.

If $u \in \mathcal{U} \subset \mathbb{Z}$ (i.e., using integer-encoded actions) is a sufficiently small discrete set, that is straightforward by an exhaustive search.

In contrast, if $u \in \mathcal{U} \subset \mathbb{R}^m$ is a (quasi-)continuous variable solving $\max_u \hat{q}(x', u, w)$ requires an own optimization routine which is computationally expensive if we use nonlinear function approximation.
The Deterministic Policy Trick

When using a greedy, deterministic policy $\pi(x, \theta) = \mu(x, \theta)$ we can utilize it to approximate

$$\max_u \hat{q}(x', u, w) \approx \hat{q}(x', \mu(x', \theta), w). \tag{13.1}$$

Hence, we can obtain explicit $Q$-learning targets for continuous actions when using a deterministic policy.

For improving the policy we reuse the deterministic policy gradient theorem in an off-policy fashion

$$\nabla_\theta J(\theta) = \mathbb{E}_b \left[ \nabla_\theta \mu(X, \theta) \nabla_u q(X, U) \bigg| U = \mu(X, \theta) \right] \tag{13.2}$$

given a behavior policy $b(u|x)$. 

Oliver Wallscheid

RL Lecture 13

4
Hence, we can consider the DDPG approach as a combination of DQN + DPG rendering it an actor-critic off-policy approach for continuous state and action spaces.

Similarly to DQN we will introduce several 'tweaks' to stabilize and improve the DDPG learning process.

**Tweak #1: experience replay buffer**

- We store \( \langle x, u, r, x' \rangle \) in \( D \) after each transition step.
- The replay buffer \( D \) is of limited capacity, i.e., it discards the oldest data sample when updating once it is full (ring memory).
- This allows us to improve the \( Q \)-learning critic minimizing the mean-squared Bellman error (MSBE):

\[
\mathcal{L}(w) = \left[ (r + \gamma q(x', \mu(x', \theta), w)) - q(x, u, w) \right]^2_D.
\]  

(13.3)
Tweak #2: target networks

- Similar to DQN we introduce a (delayed) target network to estimate the $Q$-learning target

$$r + \gamma q(x', \mu(x', \theta), w)$$

since it depends on the same parameters $w$ which we want to update.

- Hence, the target network’s purpose is to mimic the generation of i.i.d. data as the ground truth to minimize (13.3).

- Since the policy parameters $\theta$ are also part of the target calculation it turns out that an additional policy target network is also beneficial to stabilize the $Q$-learning.

- In contrast to the classical DQN implementation, the original DDPG algorithm does not perform periodically hard target network updates but continuous ones using a low-pass filter characteristic

$$w^- \leftarrow (1 - \tau)w^- + \tau w, \quad \theta^- \leftarrow (1 - \tau)\theta^- + \tau \theta \quad (13.4)$$

with $\tau$ representing the equivalent filter constant (hyperparameter).
Tweak #3: mini-batch sampling

- Given a sufficiently filled memory $\mathcal{D}$ and the target networks parametrized by $w^-$ and $\theta^-$ we draw uniformly distributed mini-batch samples $\mathcal{D}_b$ from $\mathcal{D}$.
- The actual $Q$-learning is then based on the loss

$$\mathcal{L}(w) = \left[ (r + \gamma q(x', \mu(x', \theta^-), w^-)) - q(x, u, w) \right]_{\mathcal{D}_b}^2.$$  

(13.5)

Tweak #4: batch normalization

- Minimizing (13.5) is a supervised learning step within the DDPG.
- The original DDPG paper by Lillicrap et al. back in 2015/16 suggested to use batch normalization, i.e., re-centering and re-scaling the inputs of each layer in an ANN.
- This idea of batch normalization was presented at that time shortly before by Ioffe and Szegedy (cf. original paper).
- Today’s perspective: stick to the current state-of-the-art supervised ML algorithms for top-class $Q$-learning stability and speed (which are normally well-covered in popular supervised ML toolboxes).
Tweak #5: exploration

Since our policy is deterministic we require an exploratory behavior policy.

Similar to DPG the standard approach is to add noise to the greedy actions, e.g., again from an Ornstein-Uhlenbeck (OU) process

\[ u_k \sim b(u|x_k) = \mu(x_k, \theta_k) + \nu_k, \quad \nu_k = \lambda \nu_{k-1} + \sigma \epsilon_{k-1}. \]

One might also add a schedule for \(\lambda\) and \(\sigma\) along the training procedure, e.g., starting with significant noise levels (increased exploration) while reducing it over time (focusing exploitation)\(^1\).

However, many other behavior policies are possible, e.g., using model or expert-based guidance.

\(^1\)Please note that this 'lambda' is not related to TD(\(\lambda\)), Sarsa(\(\lambda\)), etc. Here, it is representing the stiffness of the OU noise process.
Visual Summary of DDPG Working Principle

Fig. 13.1: DDPG structure from a bird’s-eye perspective (derivative work of Fig. 1.1 and wikipedia.org, CC0 1.0)
Algo. Implementation: DDPG

input: diff. det. policy fct. $\mu(x, \theta)$ and action-value fct. $\hat{q}(x, u, w)$

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

init: weights $w = w^- \in \mathbb{R}^\zeta$ and $\theta = \theta^- \in \mathbb{R}^d$ arbitrarily, memory $D$

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 $r_{k+1}$;

store tuple $\langle x_k, u_k, r_{k+1}, x_{k+1} \rangle$ in $D$;

sample mini-batch $D_b$ from $D$ (after initial memory warmup);

for $i = 1, \ldots, b$ samples do calculate $Q$-targets

if $x_{i+1}$ is terminal then $y_i = r_{i+1}$;

else $y_i = r_{i+1} + \gamma \hat{q}(x_{i+1}, \mu(x_{i+1}, \theta^-), w^-)$;

fit $w$ on loss $L(w) = [y - \hat{q}(x, u, w)]^2_{D_b}$ with step size $\alpha_w$;

$\theta \leftarrow \theta + \alpha_\theta [\nabla_\theta \mu(x, \theta) \nabla_u \hat{q}(x, u, w)|_{u=\mu_\theta(x)}]_{D_b}$;

Update target net. $w^- \leftarrow (1 - \tau)w^- + \tau w$, $\theta^- \leftarrow (1 - \tau)\theta^- + \tau \theta$;

Algo. 13.1: Deep deterministic policy gradient (output: parameter vectors $\theta^*$ for $\mu^*(x, \theta^*)$) and $w^*$ for $\hat{q}^*(x, u, w^*)$)
Table of Contents

1. Deep Deterministic Policy Gradient (DDPG)

2. Twin Delayed Deep Deterministic Policy Gradient (TD3)

3. Trust Region Policy Optimization (TRPO)

4. Proximal Policy Optimization (PPO)
For $Q$-learning in the tabular case we have already discussed the maximization bias (cf. Fig. 5.19) issue.

Recap: Due to the greedy policy targets, $\hat{q}$ was overestimated when calculated using sampled values of stochastic MDPs.

Additional problem when applying function approximation: the estimator itself introduces additional variance during the learning process which represents another source of the maximization bias problem.

This issue is already known in the DQN context (cf. Algo. 10.8). Similar to the tabular case, double DQN introduces a second $Q$-network counteracting the overestimation issue (cf. paper by van Hasselt et al.).

However, we did not address this possible problem in an actor-critic context using function approximation (e.g., DDPG).
Overestimation Bias in Actor-Critic Approaches (1)

- It turns out that the overestimation bias is also an issue for actor-critic methods as shown next \(^1\).
- Consider an actor-critic policy with the current policy parameters \(\theta\).
- Let \(\tilde{\theta}\) define the parameters from the actor update induced by the maximization of the approximate critic \(\hat{q}_w(x, u)\).
- Let \(\theta^*\) be the parameters from the hypothetical actor update w.r.t. the true underlying value function \(q^\pi(x, u)\).
- Then, we perform the policy update

\[
\tilde{\theta} = \theta + \frac{\alpha}{Z_1} \mathbb{E}_\pi \left[ \nabla_{\theta} \pi_\theta(X) \nabla_u \hat{q}_w(X, U) \middle| U = \pi_\theta(X) \right],
\]

\[
\theta^* = \theta + \frac{\alpha}{Z_2} \mathbb{E}_\pi \left[ \nabla_{\theta} \pi_\theta(X) \nabla_u q^\pi(X, U) \middle| U = \pi_\theta(X) \right],
\]

(13.6)

where \(Z_1\) and \(Z_2\) normalize the gradient such that \(Z^{-1}||\mathbb{E} [\cdot]|| = 1\).


Oliver Wallscheid

RL Lecture 13

13
Overestimation Bias in Actor-Critic Approaches (2)

▶ Lets denote $\tilde{\pi}$ and $\pi^*$ as the policies with updated parameters $\tilde{\theta}$ and $\theta^*$ respectively.

▶ As the gradient direction is a local maximizer, there exists $\epsilon_1$ sufficiently small such that if $\alpha \leq \epsilon_1$ then the approximate value of $\tilde{\pi}$ will be bounded below by the approximate value of $\pi^*$:

$$E[\hat{q}_w(X, \tilde{\pi}(X))] \geq E[\hat{q}_w(X, \pi^*(X))]. \quad (13.7)$$

▶ Conversely, there exists $\epsilon_2$ sufficiently small such that if $\alpha \leq \epsilon_2$ then the true value of $\tilde{\pi}$ will be bounded above by the true value of $\pi^*$:

$$E[q^\pi(X, \pi^*(X))] \geq E[q^\pi(X, \tilde{\pi}(X))]. \quad (13.8)$$

▶ In other words: if the approximate and true critics differ from each other, the according policy gradient updates cannot lead to better policy updates of the respective other framework.
If the expected, estimated action value will be at least as large as the true action value w.r.t. $\theta^*$

$$
\mathbb{E}[\hat{q}_w(X, \pi^*(X))] \geq \mathbb{E}[q^\pi(X, \pi^*(X))], \quad (13.9)
$$

then (13.7) and (13.8) imply

$$
\mathbb{E}[\hat{q}_w(X, \tilde{\pi}(X))] \geq \mathbb{E}[q^\pi(X, \tilde{\pi}(X))], \quad (13.10)
$$

with a sufficiently small $\alpha < \min\{\epsilon_1, \epsilon_2\}$.

Hence, the maximization bias is also present in actor-critic updates.

It can add up over several estimation updates and, therefore, may lead to suboptimal policy updates.

A proof for unnormalized gradients can be also found in S. Fujimoto et al., *Addressing Function Approximation Error in Actor-Critic Methods*, 2018.
Fig. 13.2: Comparison of true and estimated values averaged over 10000 states in two robotic examples from OpenAI Gym. Estimated values originate from the approximate DDPG critic while the true values are based on the average discounted return over 1000 episodes following the current policy, starting from states sampled from the replay buffer (source: S. Fujimoto et al., *Addressing Function Approximation Error in Actor-Critic Methods*, 2018.)
Using function approximation, the Bellman equation is never exactly satisfied leaving room for some amount of residual TD-error $\tilde{\delta}(x, u)$:

$$\hat{q}_w(x, u) = r + \gamma \mathbb{E}_\pi \left[ \hat{q}_w(X', U') | X' = x', U' = u' \right] - \tilde{\delta}(x, u).$$

(13.11)

Although this error might be considered small per update step, it may accumulate over future steps if biased:

$$\hat{q}_w(x, u) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k \left( R_k - \tilde{\delta}_k(X, U) \right) \right] | X = x, U = u.$$

(13.12)

Observation: the variance of $\hat{q}$ will be proportional to the variance of future reward and residual TD-errors.

If $\gamma$ is large, the estimation variance might increase significantly.

Mini-batch sampling will contribute to this variance issue.
In order to reduce both the maximization bias and the learning variance, TD3 introduces mainly three measures on top of the DDPG algorithm. Hence, TD3 is a direct successor of DDPG.

Measure #1: clipped double $Q$-learning for actor-critic

- Following double $Q$-learning, a pair of critics $\{\hat{q}_{w_1}, \hat{q}_{w_2}\}$ is introduced.
- In contrast, the clipped target (with target networks $\{w_1^-, w_2^-\}$)

$$y = r + \gamma \min_{i=1,2} \hat{q}_{w_i^-}(x', u')$$  \hspace{1cm} (13.13)

provides an upper-bound on the estimated action value.

- May introduce some underestimation, which is considered less critical than overestimation, since the value of underestimated actions will not be explicitly propagated through the policy update.

- The $\min$ operator will also (indirectly) favor actions leading to values with estimation errors of lower variance.
Measure #2: target policy smoothing regularization

- Background: deterministic policies $\mu$ tend to overfit to narrow peaks in the action-value estimate.
- Counteraction: fit the action value of a small area around the target action (i.e., smoothing $\hat{q}$ in the action space):

$$y = r + \gamma \hat{q}_{w-}(x', \mu_{\theta-}(x') + \epsilon). \quad (13.14)$$

- Here, $\epsilon \sim \text{clip} \left( \mathcal{N}(0, \Sigma), -c, c \right)$ is a mean-free, Gaussian noise with covariance $\Sigma$, which is clipped at $\pm c$ while $\theta^{-}$ are the policy target network parameters.
- To satisfy possible action constraints (denoted by upper and lower box constraints $\{u, \bar{u}\}$), we add an additional clipping:

$$u' = \text{clip} \left( \mu_{\theta-}(x') + \epsilon, u, \bar{u} \right). \quad (13.15)$$

- This modified action is then used for the target calculation (13.13).
Measure #3: delayed policy updates

- Similar to DDPG, TD3 uses policy target networks $\theta^-$ and (two) critic target networks $\{w_1^-, w_2^-\}$ in order to provide (rather) fixed $Q$-learning targets trying to stabilize the learning of $\hat{q}$.
- The target networks are also continuously updated using
  \[ w_i^- \leftarrow (1 - \tau)w_i^- + \tau w_i, \quad \theta^- \leftarrow (1 - \tau)\theta^- + \tau \theta. \]

- However, each policy update will inherently change the (true) $Q$-learning target directly adding variance to the learning process (cf. Fig. 13.3 on next slide).
- Therefore, it is argued that a policy update should not follow after each $Q$-learning update such that the critic can adapt properly to the previous policy update.
- The original TD3 implementation suggests a policy update every second $Q$-learning update, however, we can consider this update rate a hyperparameter.
TD3 Extensions and Modifications (4)

Fig. 13.3: Average estimated action value of a randomly selected state on Hopper-v1 environment from OpenAI Gym (source: S. Fujimoto et al., *Addressing Function Approximation Error in Actor-Critic Methods*, 2018.)
**input:** diff. det. policy fct. $\mu(x, \theta)$ and action-value fct. $\hat{q}(x, u, w)$

**parameter:** step sizes and filter constant $\{\alpha_w, \alpha_{\theta}, \tau\} \in \{\mathbb{R} | 0 < \alpha, \tau < 1\}$, policy update rate $k_w \in \{\mathbb{N} | 1 \leq k_w\}$, target noise $\Sigma \in \mathbb{R}^{m \times m}$ and $c \in \mathbb{R}^m$

**init:** weights $\{w_1 = w_1^-, w_2 = w_2^-\} \in \mathbb{R}^\zeta$, $\theta = \theta^- \in \mathbb{R}^d$ arbitrarily, memory $D$

**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 $r_{k+1}$;

    store tuple $\langle x_k, u_k, r_{k+1}, x_{k+1} \rangle$ in $D$;

    sample mini-batch $D_b$ from $D$ (after initial memory warmup);

    **for** $i = 1, \ldots, b$ **samples** **do** calculate $Q$-targets

    **if** $x_{i+1}$ is terminal **then** $y_i = r_{i+1}$;

    **else**

    $u' = \text{clip}(\mu_{\theta^-}(x_{i+1}) + \text{clip}(\mathcal{N}(0, \Sigma), -c, c), u, \bar{u})$;

    $y_i = r_{i+1} + \gamma \min_{l=1,2} \hat{q}(x_{i+1}, u', w_l^-)$;

    fit $w_l$ on loss $\mathcal{L}(w_l) = [y - \hat{q}(x, u, w_l)]_D^2$ with step size $\alpha_w \ \forall l$;

    **if** $k \mod k_w = 0$ **then**

    $\theta \leftarrow \theta + \alpha_{\theta}[\nabla_{\theta} \mu(x, \theta) \nabla_u \hat{q}(x, u, w_1)|_{u=\mu_{\theta}(x)}]D_b$;

    $w_l^- \leftarrow (1 - \tau)w_l^- + \tau w_l$, $\theta^- \leftarrow (1 - \tau)\theta^- + \tau\theta$;

---

**Algo. 13.2:** Twin delayed deep deterministic policy gradient (output: parameter vectors $\theta^*$ for $\mu^*(x, \theta^*)$) and $w^*$ for $\hat{q}^*(x, u, w^*)$)
<table>
<thead>
<tr>
<th></th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Deep Deterministic Policy Gradient (DDPG)</td>
</tr>
<tr>
<td>2</td>
<td>Twin Delayed Deep Deterministic Policy Gradient (TD3)</td>
</tr>
<tr>
<td>3</td>
<td>Trust Region Policy Optimization (TRPO)</td>
</tr>
<tr>
<td>4</td>
<td>Proximal Policy Optimization (PPO)</td>
</tr>
</tbody>
</table>
In contrast to the previous two algorithms, we will focus on stochastic policies $\pi(u|x)$ in the following.

First, we rewrite the performance metric (12.7) to obtain

$$J_\pi = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_k \right].$$  \hfill (13.16)

Using the advantage $a_\pi(x, u) = q_\pi(x, u) - v_\pi(x)$ we can calculate the performance of an updated policy $\pi \to \tilde{\pi}$.

$$J_{\tilde{\pi}} = J_\pi + \int_{\mathcal{X}} p_{\tilde{\pi}}(x) \int_{\mathcal{U}} \tilde{\pi}(u|x)a_\pi(x, u).$$  \hfill (13.17)

While for finite MDPs, the policy improvement theorem guaranteed $J_{\tilde{\pi}} \geq J_\pi$ for each policy update, there might be some states where $\int_{\mathcal{U}} \tilde{\pi}(u|x)a_\pi(x, u) < 0$ for continuous MDPs using function approximation.

\footnote{proof from: S. Kakade and J. Langford, \textit{Approximately optimal approximate reinforcement learning}, ICML, vol. 2, pp 267-274, 2002}
Reinterpreting the Stochastic Policy Gradient (2)

- For easier calculation, we introduce a local approximation to (13.17)

$$L_{\pi}(\hat{\pi}) = J_{\pi} + \int_{\mathcal{X}} p^\pi(x) \int_{\mathcal{U}} \tilde{\pi}(u|x) a_{\pi}(x, u)$$  \hspace{1cm} (13.18)

where $p^\pi(x)$ is used instead of $p^\tilde{\pi}(x)$, i.e., neglecting the state distribution change due to a policy update.

- For any parametrized and differentiable policy $\pi_{\theta}(u|x)$, it can be shown that

$$L(\pi_{\theta_0}) = J(\pi_{\theta_0}),$$

$$\nabla_{\theta} L_{\pi_{\theta_0}}(\pi_{\theta})|_{\theta=\theta_0} = \nabla_{\theta} J(\pi_{\theta})|_{\theta=\theta_0}$$  \hspace{1cm} (13.19)

for any initial parameter set $\theta_0$.

- For a sufficiently small step size, improving $L_{\pi_{\theta_0}}$ will also improve $J$.

However, we do not know how much the actual stochastic policy will change while moving through the parameter space. Hence, we do not have a good decision basis to choose the policy gradient step size.
From the previous discussion it can be concluded that we want a metric describing how much a policy is changed in the action space when updating the policy in the parameter space.

Against this background, we make use of the Kullback-Leibler divergence (also called relative entropy)

\[ D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \left( \frac{p(x)}{q(x)} \right) \, dx \]  

(13.20)

defined for continuous distributions \( P \) and \( Q \) with their probability densities \( p \) and \( q \).

Example: for two multivariate Gaussian distributions of equal dimensions \( d \), with means \( \mu_0, \mu_1 \) and with (non-singular) covariance matrix \( \Sigma_0, \Sigma_1 \) we receive

\[ D_{KL}(\mathcal{N}_0 \parallel \mathcal{N}_1) = \frac{1}{2} \left( \text{tr} \left( \Sigma_1^{-1} \Sigma_0 \right) + (\mu_1 - \mu_0)^T \Sigma_1^{-1} (\mu_1 - \mu_0) \right) \\
- d + \ln \left( \frac{\det \Sigma_1}{\det \Sigma_0} \right) \].
Adding a Trust Region Constraint (2)

- The trust region policy optimization (TRPO) updates the policy parameters while constraining the KL divergence between the new and the old policy distribution:

\[
\begin{align*}
\max_{\theta} & \quad \mathcal{L}_{\theta_k}(\theta), \\
\text{s.t.} & \quad D_{\text{KL}}(\theta_k, \theta) \leq \kappa
\end{align*}
\]  

with

\[
D_{\text{KL}}(\theta_k, \theta) = D_{\text{KL}}(\pi_{\theta_k}, \pi_\theta) = \mathbb{E}_{\pi_{\theta_k}} \left[ D_{\text{KL}}(\pi_{\theta_k}(\cdot|X) \parallel \pi_\theta(\cdot|X)) \right].
\]

- Hence, we want to limit the average KL divergence w.r.t. the states visited by the old policy.

- The constraint \( \kappa \) is a TRPO hyperparameter (typically \( \kappa \ll 1 \)).

- Although (13.21) does not provide any formal convergence guarantee, we at least have a link between changes in the parameter and policy distribution space. Therefore, we can use this tool to prevent erratic policy changes.
To actually solve (13.21) we will make use of samplings from Monte-Carlo rollouts. Expanding the objective yields

$$\max_\theta \mathcal{L}_{\theta_k}(\theta) = \max_\theta J_{\pi_k} + \int_\mathcal{X} p^\pi_k(\mathbf{x}) \int_\mathcal{U} \pi_\theta(\mathbf{u}|\mathbf{x}) a_{\pi_k}(\mathbf{x}, \mathbf{u}).$$  \hspace{1cm} (13.22)$$

The first term $J_{\pi_k}$ can be dropped, since it is irrelevant for the optimization result (constant). Using samples we can approximate

$$\int_\mathcal{X} p^\pi_k(\mathbf{x}) \approx \frac{1}{1-\gamma} \mathbb{E}_{\pi_{\theta_k}}[X].$$

Moreover, \( \int_\mathcal{U} \pi_\theta(\mathbf{u}|\mathbf{x}) a_{\pi_k}(\mathbf{x}, \mathbf{u}) \approx \mathbb{E}_{\pi_{\theta_k}} \left[ \frac{\pi_\theta(U|X)}{\pi_{\theta_k}(U|X)} a_{\pi_k}(X, U) \right] $$
is also approximated applying importance sampling based on data from the old policy. Hence, the sampled objective is

$$\max_\theta \mathbb{E}_{\pi_{\theta_k}} \left[ \frac{\pi_\theta(U|X)}{\pi_{\theta_k}(U|X)} a_{\pi_k}(X, U) \right].$$  \hspace{1cm} (13.23)$$
**Smooth Policy Updates via TRPO**

Fig. 13.4: Simplified representation of the policy evolution for a scalar action given some fixed state. Left: TRPO-style updates finding the optimal action with increasing probability. Right: Unmonitored policy distributions not converging towards an optimal policy (‘policy chattering’).
Applying the previous sample-based estimation we obtain

\[ \theta_{k+1} = \arg \max_{\theta} \mathbb{E}_{\pi_{\theta_k}} \left[ \frac{\pi_{\theta}(U|X)}{\pi_{\theta_k}(U|X)} a_{\pi_k}(X, U) \right], \]

s.t. \[ \mathbb{E}_{\pi_{\theta_k}} [D_{KL}(\pi_{\theta_k}(\cdot|X) \parallel \pi_{\theta}(\cdot|X))] \leq \kappa. \]  

Hence, we have a three-step procedure for each TRPO update:

1. Use Monte-Carlo simulations based on the old policy to obtain data.
2. Use the data to construct (13.24).
3. Solve the constrained optimization problem to update the policy parameter vector.

Solving (13.24) is generally a nonlinear optimization problem. The original TRPO implementation uses a local objective and constraint approximation together with conjugate gradient and line search algorithms. However, many other constrained-nonlinear solvers are also applicable.
Generalized Advantage Estimation

Having data \( \langle x, u, r, x' \rangle \) in \( D \) from a Monte Carlo rollout available, an important problem is to estimate \( a_{\pi_k}(x, u) \) in (13.24).

A particular suggestion in the TRPO context is to use a generalized advantage estimator (GAE) \(^1\) defined as

\[
\hat{a}_k(\gamma, \lambda) = \sum_{i=0}^{\infty} (\gamma \lambda)^i \delta_{k+i}.
\]  

Here, \( \delta_k = r_k + \gamma v(x_{k+1}) - v(x_k) \) is a single advantage sample.

Hence, the GAE is the exponentially-weighted average of the discounted advantage samples with an additional weighting \( \lambda \).

Similar formulation compared to TD(\( \lambda \)) but instead of the state value the estimator’s target is the advantage.

The choice of \( (\gamma \lambda) \) trade-offs the bias and variance of the estimator.

The TRPO’s key facts are:

- The TRPO constrains policy distribution changes when updating the policy parameters (for stochastic policies and on-policy learning).
- The objective is to enable a monotonically improving learning process.
- Using trust regions, erratic policy updates should be prevented.

The TRPO’s main hurdles are:

- Constructing the objective function and constraint requires Monte Carlo rollouts (time consuming, data inefficient).
- When the sampled optimization problem is set up, a nonlinear and constrained optimization step is required (no simple policy gradient).
- For speedy implementations, only approximate solutions of the TRPO problem are possible.

We will not provide any specific TRPO implementation suggestion at this point, since this is rather cumbersome. Instead we will move forward to a similar algorithm which is pursuing the same goal (prevent erratic policy changes) with a much simpler implementation.
Table of Contents

1. Deep Deterministic Policy Gradient (DDPG)
2. Twin Delayed Deep Deterministic Policy Gradient (TD3)
3. Trust Region Policy Optimization (TRPO)
4. Proximal Policy Optimization (PPO)
The upcoming proximal policy optimization (PPO) algorithm tries to mimic the constrained TRPO problem based on related unconstrained problems.

\[
\theta_{k+1} = \arg \max_\theta \mathbb{E}_{\pi_{\theta_k}} \left[ \frac{\pi_\theta(U|X)}{\pi_{\theta_k}(U|X)} a_{\pi_k}(X, U) \right],
\]

s.t. \[
\mathbb{E}_{\pi_{\theta_k}} \left[ D_{KL}(\pi_{\theta_k}(\cdot|X) \parallel \pi_\theta(\cdot|X)) \right] \leq \kappa.
\]

Hence, the objective will be reformulated to incorporate mechanisms preventing excessively large variations of the policy distribution during a parameter update (leading to an updated policy with sufficient proximity to the old one).

Moreover, PPO incorporates two variants which we will discuss:

1. Clipping the surrogate objective,
Clipped Surrogate Objective

The first approach is based on the following objective:

\[
\mathbb{E}_{\pi_{\theta_k}} \left[ \min \left\{ \frac{\pi_{\theta}(U|X)}{\pi_{\theta_k}(U|X)} a_{\pi_k}(X,U), \operatorname{clip} \left( \frac{\pi_{\theta}(U|X)}{\pi_{\theta_k}(U|X)}, 1 - \epsilon, 1 + \epsilon \right) a_{\pi_k}(X,U) \right\} \right].
\]

(13.26)

Above, \( \epsilon < 1 \) is a PPO hyperparameter serving as a regularizer.

The first element of \( \min \{ \cdot \} \) is the previous TPRO objective.

The second element of \( \min \{ \cdot \} \) modifies the surrogate objective by clipping the importance sampling ratio \( \pi_{\theta}/\pi_{\theta_k} \).

The latter should remove the incentive for moving the importance sampling ratio outside of the interval \( [1 - \epsilon, 1 + \epsilon] \).

The modified objective is therefore a lower bound of the unclipped TRPO objective.
Consider a single sample \((x, u)\) with a positive advantage \(a_{\pi_k}(x, u)\):
\[
\max_{\theta} \min_{\pi} \left\{ \frac{\pi_\theta(u|x)}{\pi_{\theta_k}(u|x)} a_{\pi_k}(x, u), \text{clip} \left( \frac{\pi_\theta(u|x)}{\pi_{\theta_k}(u|x)}, 1 - \epsilon, 1 + \epsilon \right) a_{\pi_k}(x, u) \right\}.
\]

Because the advantage is positive, the objective will increase if the action becomes more likely, i.e., if \(\pi_\theta(u|x)\) increases.

If \(\pi_\theta(u|x) > (1 + \epsilon)\pi_{\theta_k}(u|x)\) the clipping becomes active.

Hence, the objective reduces to
\[
\max_{\theta} \min_{\pi} \left\{ \frac{\pi_\theta(u|x)}{\pi_{\theta_k}(u|x)}, 1 + \epsilon \right\} a_{\pi_k}(x, u).
\]

Due to the \(\min\{\cdot\}\) operator, the entire objective is therefore limited to \((1 + \epsilon)a_{\pi_k}(x, u)\).

Interpretation: the new policy does not benefit from going further away from the old policy.
Consider a single sample \((x, u)\) with a negative advantage \(a_{\pi_k}(x, u)\):

\[
\max_{\theta} \min \left\{ \frac{\pi_{\theta}(u|x)}{\pi_{\theta_k}(u|x)} a_{\pi_k}(x, u), \text{clip} \left( \frac{\pi_{\theta}(u|x)}{\pi_{\theta_k}(u|x)}, 1 - \epsilon, 1 + \epsilon \right) a_{\pi_k}(x, u) \right\}.
\]

Because the advantage is negative, the objective will increase if the action becomes less likely, i.e., if \(\pi_{\theta}(u|x)\) decreases.

If \(\pi_{\theta}(u|x) < (1 - \epsilon)\pi_{\theta_k}(u|x)\) the clipping becomes active.

Hence, the objective reduces to

\[
\max_{\theta} \max \left\{ \frac{\pi_{\theta}(u|x)}{\pi_{\theta_k}(u|x)}, 1 - \epsilon \right\} a_{\pi_k}(x, u).
\]

Due to the \(\max\{\cdot\}\) operator, the entire objective is limited to \((1 - \epsilon)a_{\pi_k}(x, u)\).

Interpretation: the new policy does not benefit from going further away from the old policy.
Adaptive KL Penalty

- The second PPO variant makes use of the following KL-penalized objective

\[
\mathbb{E}_{\pi_{\theta_k}} \left[ \frac{\pi_{\theta}(U|X)}{\pi_{\theta_k}(U|X)} a_{\pi_k}(X,U) - \beta D_{KL}(\pi_{\theta_k}(\cdot|X) \parallel \pi_{\theta}(\cdot|X)) \right]. \quad (13.27)
\]

- Transfers the KL-based constraint into a penalty for large policy distribution changes.

- The parameter \( \beta \) weights the penalty against the policy improvement.

- The original PPO implementation suggests an adaptive tuning of \( \beta \) w.r.t. the sampled average KL divergence \( \overline{D}_{KL}(\theta_k, \theta) \) estimated from previous experience

\[
\overline{D}_{KL}(\theta_k, \theta) < \overline{D}^*_{KL} : \beta \leftarrow \beta/2, \\
\overline{D}_{KL}(\theta_k, \theta) > \overline{D}^*_{KL} : \beta \leftarrow \beta \cdot 2.
\]

(13.28)

with some target value of the KL divergence \( \overline{D}^*_{KL} \) (additional hyperparameter).
**Algo. Implementation: PPO**

**input:** diff. stochastic policy fct. $\pi(u|x, \theta)$ and value fct. $\hat{v}(x, w)$

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

**init:** weights $w \in \mathbb{R}^\zeta$ and $\theta \in \mathbb{R}^d$ arbitrarily, memory $\mathcal{D}$

for $j = 1, 2, \ldots, \text{(sub-)episodes}$ do

- initialize $x_0$ (if new episode);
- collect a set of tuples $\langle x_k, u_k, r_{k+1}, x_{k+1} \rangle$ by running $\pi(u|x, \theta_j)$;
- store them in $\mathcal{D}$;
- estimate the advantage $\hat{a}_{\pi_j}(x, u)$ based on $\hat{v}(x, w_j)$ and $\mathcal{D}$ (e.g., GAE);
- $\theta_{j+1} \leftarrow$ policy gradient update on (13.26) or (13.27);
- $w_{j+1} \leftarrow$ minimizing the mean-squared TD errors using $\mathcal{D}$;
- delete entries in $\mathcal{D}$;

**Algo. 13.3:** Proximal policy optimization (output: parameter vectors $\theta^*$ for $\pi^*(u|x, \theta^*)$) and $w^*$ for $\hat{v}^*(x, w^*)$)
Some PPO Remarks

- Clipping the surrogate objective (13.26) was reported to achieve higher performances than the KL penalty (13.27).\(^1\)

- Like TRPO, PPO is an on-policy algorithm. Hence, the memory \(\mathcal{D}\) is not a rolling replay buffer (cf. off-policy algorithms like DQN, DDPG or TD3) but a rollout buffer using one fixed policy.

- These rollouts are likely to result in an increased sample demand either using a simulator or a real experiment.

Although PPO is derived from a TRPO background pursuing monotonically increasing policy performance, its realization is based on multiple heuristics and approximations. Hence, there is no guarantee on achieving this goal and the specific performance of the PPO algorithm must be evaluated empirically given a certain application.

---

Exemplary Performance Comparison

Fig. 13.5: Learning curves for OpenAI Gym continuous control tasks. The shaded region represents half a standard deviation of the average evaluation over ten trials (source: S. Fujimoto et al., *Addressing Function Approximation Error in Actor-Critic Methods*, 2018).
Outlook: Other Contemporary Algorithms (1)

The selection of algorithms appears endless:
- DQN variants such as
  - (Prioritized) dueling DQN
  - Noisy DQN
  - Distributional DQN
- Rainbow (combining multiple DQN extensions)
- Soft actor-critic (SAC)
- Actor critic using Kronecker-factored trust region (ACKTR)
- Asynchronous advantage actor-critic (A3C)
- ...

Remarks:
- You have already learned the basic building blocks in order to make yourself familiar with any value-/policy-based or hybrid RL approach.
- Use this knowledge!
- Focus on primary scientific literature for self-studying and not on arbitrary blogs or other possible non-reliable sources!
Outlook: Other Contemporary Algorithms (2)

Algorithm collections with tutorial-style documentation:

▶ Intel Reinforcement Learning Coach
▶ OpenAI Spinning Up

Algorithm collections with decent application-oriented documentation:

▶ Acme
▶ Garage
▶ Google Dopamine
▶ RLlib (Ray)
▶ Stable Baselines3
▶ Tensorforce
▶ TF-Agents
▶ ...
The deep deterministic policy gradient (DDPG) approach ‘transfers’ many deep $Q$-network (DQN) ideas to continuous action spaces. It mainly combines DQN + deterministic policy gradients + policy and value target networks (plus additional minor tweaks).

However, the DDPG actor-critic suffers from value overestimation and high variance during learning. Hence, sampled policy gradients might not be optimal (pointing towards overrated action values).

Twin delayed DDPG (TD3) adds clipped double $Q$-learning, delayed policy updates and target policy smoothing to counteract these issues.

Trust region policy optimization (TRPO) pursues monotonically increasing policy performance by limiting policy distribution changes. This results in a nonlinear constrained optimization problem adding computational complexity (no simple policy gradients).

Proximal policy optimization (PPO) converts the TRPO idea into an unconstrained optimization problem by a modified objective. Likewise, the PPO’s objective is to prevent erratic policy distribution changes.
Thanks for your attention and have a nice week!