Reinforcement Learning Cheatsheet (Work in progress)

Alexandre Thomas
Mines ParisTech & Sorbonne University
alexandre.thomas@mines-paristech.fr

November 14, 2021

Contents

1 Bandits

The *multi-armed bandits problem* is a simplified setting of RL where actions do not affect the world state. In other words, the current state does not depend on previous actions, and the reward is immediate. Like any RL problem with unknown MDP, a successful agent should solve the *exploration-exploitation dilemma*, i.e. find a balance between exploiting what it has already learned to improve the reward and exploring in order to find the best actions.

◇ **Settings** Bandits problems can be stationary or non-stationary, and the setting can be stochastic or adversarial. Agents typically learn in an online setting and, in the case of a non-associative task (no need to associate different actions with different situations), they try to find a single best action out of a finite number of actions (also called "arms"). For associative tasks, *contextual bandits* make use of additional information which can be global or individual context (i.e. per arm). Context can be fixed or variable.

◇ **ϵ-greedy** The ϵ-greedy strategy selects a random action with probability ϵ. TODO: add equations

◇ **Upper Confidence Bounds (UCB)** UCB [1] follows an optimistic strategy and selects the best arm in the best case scenario, i.e. according to the upper bounds $B_t(i)$ on the arms value estimates:

$$\pi_t = \arg\max_i B_t(i) \quad \text{with} \quad B_t(i) = \hat{\mu_{i,t}} + \sqrt{\frac{2\log t}{\sum_{s=0}^{t} \mathbb{1}_{\pi_s=i}}}$$

LinUCB [2] follows the UCB strategy but considers a linear and individual context $x_{i,t}$. We have $\mathbb{E}[r_{i,t}|x_{i,t}] = \theta_i^T x_{i,t}$ and parameters θ_i are estimated with Ridge Regression on previously observed contexts and rewards.

◇ **Thompson Sampling** Thompson Sampling [3] follows a Bayesian approach and considers a parametric model $P(\mathcal{D}|\theta)$ with a prior $P(\theta)$. For instance, in the linear case [4]: $P(r_{i,t}|\theta) = \mathcal{N}(\theta^T x_{i,t}, v^2)$ and $P(\theta) = \mathcal{N}(0, \sigma^2)$. Then, at each iteration t, we sample θ from $P(\theta|\mathcal{D}) \propto P(\mathcal{D}|\theta)P(\theta)$ and select the arm $\pi_t = \arg\max_i \mathbb{E}[r_{i,t}|x_{i,t}, \theta]$.

2 RL Framework

◇ **Markov Decision Process (MDP)** A Markov Decision Process is a tuple (S, A, R, P, ρ_0) where

- S is the set of all valid states
- A is the set of all valid actions
- $R : S \times A \times S \to \mathbb{R}$ is the *reward function*, such that $r_t = R(s_t, a_t, s_{t+1})$. In the case the reward is stochastic and r_t is a random variable, we have $R(s, a, s') = \mathbb{E}[r_t|s_t = s, a_t = a, s_{t+1} = s']$
- $P : S \times A \times S \to [0, 1]$ the *transition probability function*, such that $P(s'|s, a)$ is the probability of transitioning into state s' if you are in state s and take action a
- $\rho_0 : S \to [0, 1]$ is the starting state distribution

◇ **Markov property** Transitions only depend on the most recent state and action, and no prior history : $P(s_{t+1}|s_t, a_t, \ldots, s_1, a_0, s_0) = P(s_{t+1}|s_t, a_t)$. This assumption does not always hold, for instance when the observed state does not contain all necessary information (*Partially Observable Markov Decision Process*), or when P and R actually depend on t (*Non-Stationary Markov Decision Process*).

When the MDP is known (e.g. small tabular environments), optimal policies can be found offline without interacting with the environment, using Dynamic Programming (DP) algorithms. But this is generally not the case and RL algorithms have to do trial-and-error search (like bandits problems), and have to deal with *delayed rewards*. Actions may affect not only the immediate reward, but also the next state and therefore all subsequent rewards.

◇ **Definitions**

- The *policy* π determines the behavior of our agent, who will take actions $a_t \sim \pi(\cdot|s_t)$. Policies can be derived from an action-value function or can be explicitly parameterized and denoted by π_θ. They can also be deterministic, in which case they are sometimes denoted by μ_θ, with $a_t = \mu_\theta(s_t)$.
- A *trajectory* $\tau = (s_0, a_0, s_1, \dots)$ is a sequence of states and actions in the world, with $s_0 \sim \rho_0$ and $s_{t+1} \sim P(\cdot|s_t, a_t)$. It is sampled from π if $a_t \sim \pi(\cdot|s_t)$ for each t. Trajectories are also called *episodes*.
- The return $R(\tau)$ is the cumulative reward over a trajectory and is the quantity to be maximized by our agent. It can refer to the *finite-horizon undiscounted return* $R(\tau) = \sum_{t=0}^{T} r_t$ or the *infinite-horizon discounted return* $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$ for instance. Parameter γ is called the *discount factor*.
- The *on-policy value function*: $V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s]$
- The *on-policy action-value function*: $Q^\pi(s,a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]$. We have $V^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[Q^\pi(s,a)]$.
- The *advantage function*: $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$
- The optimal value and action-value functions are obtained by acting according to an optimal policy π^*: $V^*(s) = \max_\pi V^\pi(s)$, $Q^*(s,a) = \max_\pi Q^\pi(s,a)$. We have $V^*(s) = \max_a Q^*(s,a)$, and a deterministic optimal policy can be obtained with $\pi^*(s) = \arg\max_a Q^*(s,a)$.

◇ **Bellman Equations**

$$V^\pi(s) = \mathop{\mathbb{E}}_{\substack{a\sim\pi(\cdot|s)\\ s'\sim P}}[R(s,a,s') + \gamma V^\pi(s')] \tag{1}$$

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{s'\sim P}\left[R(s,a,s') + \gamma \mathop{\mathbb{E}}_{a'\sim\pi(\cdot|s')}[Q^\pi(s',a')]\right] \tag{2}$$

$$V^*(s) = \max_a \mathop{\mathbb{E}}_{s'\sim P}[R(s,a,s') + \gamma V^*(s')] \tag{3}$$

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s'\sim P}\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right] \tag{4}$$

Where $s' \sim P$ is a shorthand for $s' \sim P(\cdot|s,a)$

3 Dynamic Programming

◇ **Policy Evaluation Algorithm** For (small) tabular environments with known MDP, Bellman equations can be computed exactly and one can converge on V^π by applying equation 1 repeatedly:

$$V_{i+1}(s) = \sum_{a\in A}\sum_{s'\in S} \pi(a|s)P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$$

◇ **Policy Iteration Algorithm** Being *greedy* with respect to current value function V^{π_k} makes it possible to define a better (deterministic) policy:

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'\in S} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_k}(s')] \qquad \textit{policy improvement}$$

In the policy iteration algorithm, the policy evaluation algorithm is applied until convergence to V^{π_k} and followed by one policy improvement step. This is repeated until convergence to π^* (characterized by stationarity). The idea of having these two policy evaluation and improvement processes interact is found in many RL algorithms (not necessarily as separate steps, but also simultaneously) and is called *general policy iteration*.

◇ **Value Iteration Algorithm** Equation 3 is applied repeatedly to converge directly on V^*:

$$V_{i+1}(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$$

And π^* is obtained by being greedy to V^*, which is possible since we know the MDP. Q-Value iteration (with equation 4) requires storing more values but is possible as well, and makes model-free approaches possible when the MDP is unknown (Value-Based section).

4 Value-Based

Value-based methods typically aim at finding Q^* first, which then gives the optimal policy $\pi^*(s) = \arg\max_a Q^*(s,a)$. For this, they require finite action spaces.

4.1 Tabular environments

In the case of tabular environments (i.e. S is finite as well), Q^π can actually be represented by a matrix.

◇ **Tabular Q-learning** Starting from a random value-action function Q, tabular Q-learning consists in interacting with the environment and applying equation 4 to converge to Q^* (algorithm 1). Q-learning is an *off-policy* method, i.e. learns the value of the optimal policy independently of the agent's actions, and π can be any policy as long as all state-action pairs are visited enough.

Algorithm 1: Tabular Q-learning

for *a number of episodes* **do**
  initialize s_0
  while *episode not done* **do**
    take action $a_t \sim \pi(s_t)$, observe r_t and s_{t+1}
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) \right]$

◇ **SARSA** The SARSA algorithm is the *on-policy* equivalent of Q-learning as it learns the value of the current agent's policy π. Instead of using Bellman equation 4, it uses equation 2 for the update step, and converges to the optimal policy π^* as long as policy π becomes greedy in the limit.

◇ **Exploration strategies** Used by value-based algorithms (SARSA, Q-learning) to balance between exploration and exploitation when interacting with the environment.

- *ϵ-greedy strategy*

$$\pi(s) = \begin{cases} \arg\max_a Q(s,a) & \text{be greedy with probability } \epsilon \\ a \textit{ random action}, & \text{otherwise} \end{cases}$$

- *Boltzmann selection*

$$\pi(a|s) = \frac{\exp(Q(s,a)/\tau)}{\sum_{a'} \exp(Q(s,a')/\tau)}$$

- Others: UCB-1, pursuit strategy. Comparison in [5] (for stochastic mazes).

◇ **Temporal Difference (TD) learning** TD methods combine Monte-Carlo (MC) sampling [1] with the *bootstrapping* of DP methods [2], and are used to learn V^π or Q^π. The following quantity

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

is called the *TD-error*.

[1] Since we don't know the MDP, we have to approximate the expectation using trajectory samples
[2] Bootstrapping: using our current approximation of $V^\pi(s')$ to estimate $V^\pi(s)$

- *TD(0) learning* is the most straightforward method. After acting according to π, the value function is updated with:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)]$$

  Since $\mathbb{E}[r_t + \gamma V^\pi(s_{t+1})] = V^\pi(s_t)$, V will eventually converge to V^π.

- *Multi-steps learning.* When the reward is delayed and is observed several steps after the decisive action, TD(0) is slow since values only propagate from one state to the previous one. With:

$$G_t^{(n)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V^\pi(s_{t+n})$$

  the *n-step return*, it is also true that $\mathbb{E}[G_t^{(n)}] = V^\pi(s_t)$. Therefore, the update steps becomes:

$$V(s_t) \leftarrow V(s_t) + \alpha[G_t^{(n)} - V(s_t)]$$

  Multi-steps learning can converge faster depending on the problem. As $n \to \infty$, $G_t^{(n)}$ approaches the unbiased MC estimate of V^π, at the cost of higher variance.

- *TD(λ) learning.* Instead of choosing the right n, a better way to navigate between TD(0) and MC updates is to average all n-step returns with a decay λ. For $\lambda \in [0, 1)$, the *λ-return* is defined as

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

  and also verifies $\mathbb{E}[G_t^\lambda] = V^\pi(s_t)$, therefore can be used in the update step. Choosing $\lambda = 0$ is equivalent to TD(0) updates, and $\lambda \to 1$ approaches MC updates.

TODO: TD learning algo ? (forward view)

◇ **Eligibility traces** Eligibility traces e_t are an equivalent but more convenient way of implementing TD(λ) learning. Updates are done online (*backward view*) instead of having to wait the end of the trajectory (*forward view*).

Algorithm 2: Eligibility traces, also called online TD(λ), tabular version

for *a number of episodes* **do**
  initialize s_0 and set $e_0(s) = 0, \forall s$
  while *episode not done* **do**
    take action $a_t = \pi(s_t)$, observe r_t and s_{t+1}
    for *all s* **do**
      $e_t(s) \leftarrow \lambda\gamma e_{t-1}(s) + \mathbb{1}_{s_t=s}$ `// this leads to` $e_t(s) = \sum_{k=0}^{t} (\lambda\gamma)^{t-k} \mathbb{1}_{s_k=s}$
      $V(s_t) \leftarrow V(s_t) + \alpha e_t(s) \delta_t$

TODO: version with weights

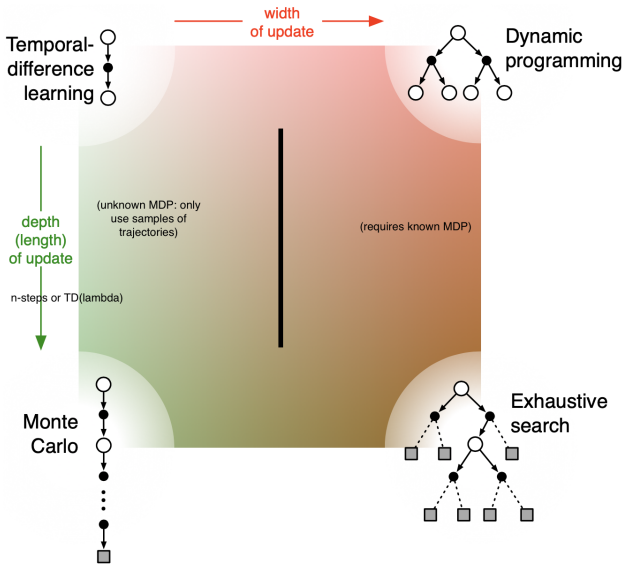

Figure 1: Taken from [6]

4.2 Approximate Q-learning

Tabular learning approaches have trouble learning in large environments since there is no generalization between similar situations, and storing Q or V can even be an issue. Approximate approaches allow working with large finite environments and even with continuous state space S, by considering

$$Q(s,a) \approx Q_\theta(\phi(s),a)$$

with $\phi(s) \in \mathbb{R}^d$ a vector of features. Features ϕ can be hand-crafted, but in Deep Reinforcement Learning (DRL) they are typically learned using neural networks and we simply note $Q(s,a) \approx Q_\theta(s,a)$.

◇ **Deep Q-Network (DQN) [7][8]** DQN was the first successful attempt at applying DRL on high-dimensional state spaces, and uses 2D convolutions with an MLP to extract features. It overcomes stability issues thanks to:

- *Experience-replay.* Within a trajectory, transitions are strongly correlated but gradient descent algorithms typically assume independent samples, otherwise gradient estimates might be biased. Storing transitions and sampling from a memory buffer D reduces correlation, and even allows mini-batch optimization to speed up training. Re-using past transitions also limits the risk of catastrophic forgetting.
- *Target network.* Based on eq. 4, values $Q_\theta(s,a)$ can be learned by minimizing the error[3] to the target $r + \gamma \max_{a'} Q_\theta(s',a')$. In this case, θ is continuously updated and we are chasing a non-stationary target. Learning can be stabilized by using a separate network Q_{θ^-} called the target network, with weights θ^- updated every k steps to match θ.

[3]This can be the mean square error, or Huber loss for more stability

Algorithm 3: Deep Q-learning with experience replay and target network (DQN)

Init: replay memory D with capacity M, Q_θ with random weights, $\theta^- = \theta$
for *a number of episodes* **do**
  initialize s_0
  while *episode not done* **do**
    take action $a_t \sim \pi(s_t)$, observe r_t and s_{t+1} `// π can be ϵ-greedy for instance`
    store transition (s_t, a_t, r_t, s_{t+1}) in D
    sample random mini-batch of transitions $((s_j, a_j, r_j, s_{j+1}))_{j=1,\dots,N}$ from D
    set $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is a terminal state} \\ r_j + \gamma \max_{a'} Q_{\theta^-}(s_{j+1}, a') & \text{otherwise} \end{cases}$
    $\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{N} \sum_{j=1}^{N} (y_j - Q_\theta(s_j, a_j))^2$
    every k steps, update $\theta^- = \theta$

◇ **Prioritized Experience Replay (PER) [9]** Improve experience replay: rather than sampling transitions t_i uniformly from the memory buffer, prioritize the ones that are the most informative, i.e. with the largest error.

$$i \sim P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \quad \text{with} \quad p_i = |\delta_i| + \epsilon$$

And $\delta_t = r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$ the TD-error again. In practice when implementing PER, errors δ_i are stored in memory with their associated transition t_i and are only updated with current Q_θ when t_i is sampled. A SumTree structure can be used to efficiently sample from $P(i)$, in $O(\log |D|)$. Since PER introduces a bias[4], authors use *importance sampling* and correct the loss with a term $w_i = 1/(NP(i))^\beta$.

◇ **Double DQN [10]** Because it is using the same value function Q_θ for both action selection and evaluation in the target $y_t = r_t + \gamma \max_a Q_\theta(s_{t+1}, a)$, Q-learning tends to overestimate action values as soon as there is any estimation error. This is particularly the case when the action space is large (figure 2). Double Q-learning [11] decouples action selection and evaluation using 2 parallel networks Q_θ and $Q_{\theta'}$. Double DQN [10] improves DQN performance and stability by doing it simply with the online network Q_θ and target network Q_{θ^-}:

$$y_t = r_t + \gamma Q_{\theta^-}(s_{t+1}, \arg\max_a Q_\theta(s_{t+1}, a))$$

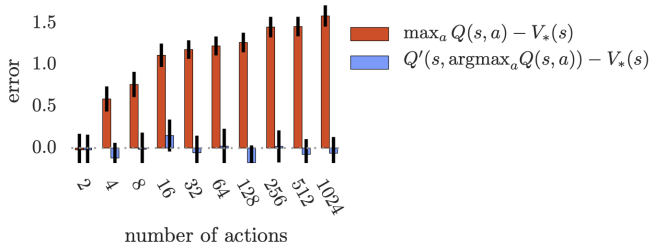

Figure 2: Taken from [10]

◇ **Rainbow [12]** Rainbow studies and combines a number of improvements to the DQN algorithm: multi-steps returns, prioritized experience replay, Double Q-learning, as well as dueling networks (using a value function $V_\theta(s)$ and an advantage function $A_\theta(s, a)$ to estimate $Q(s, a)$), noisy nets (adaptive exploration by adding noise to the parameters of the last layer, instead of being ϵ-greedy) and distributional RL (approximate the distribution of returns instead of the expected return).

◇ **Deep Recurrent Q-Network [13]** By using a RNN to learn the features (e.g. after the convolutional layers), DQRN keeps in memory a representation of the world according to previous observations. This makes it possible to go beyond the Markov property and work with POMDPs, but it can be harder to train.

[4] This blog post goes into more details

5 Policy Gradients

Policy gradient algorithms directly optimize the policy $\pi_\theta(\cdot|s)$. The goal is to maximize the expected return under π_θ:

$$\max_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)]$$

Using the log-derivative trick ($\nabla f = f\nabla \log f$) and the likelihood of a trajectory τ:

$$\pi_\theta(\tau) = \rho_0(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

the *policy gradient* can be derived:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(\tau)R(\tau)] \\ &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} R(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)\right] \end{aligned} \tag{5}$$

Learning the optimal policy can be easier than learning all the action values $Q(s, a)$, and directly optimizing π_θ makes it possible to have smoother updates and more stable convergence than Q-learning. In addition, all policy gradient algorithms work with continuous/infinite action spaces, and can encourage exploration with additional rewards (entropy). However, they can be less adapted to tabular environments and are often less sample-efficient than value-based approaches, which typically use memory buffers.

5.1 On-Policy

◇ **REINFORCE** Based on eq. 5, the REINFORCE algorithm simply uses a Monte-Carlo estimate of $\nabla_\theta J(\theta)$ to optimize π_θ, and increase the likelihood of trajectories with high rewards.

Algorithm 4: Vanilla REINFORCE

for *a number of epochs* **do**
  while *not enough samples* **do**
    collect trajectory $\tau_i = (s_0^i, a_0^i, \dots, s_{T_i}^i)$ by running active policy π_θ
  $\hat{g} \leftarrow \sum_i \sum_t R(\tau^i)\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)$ `// compute policy gradient estimate`
  $\theta \leftarrow \theta + \alpha\hat{g}$

It is unbiased but typically has very high variance and is slow to converge. Variance can be reduced using alternatives expressions of the gradient (all three can be combined):

- *Causality (don't let the past distract you)* Intuitively, rewards obtained before taking an action do not bring any information, they are just noise. Indeed $R(\tau)$ can be replaced by the *reward-to-go* $R_t(\tau) = \sum_{t'=t}^{T} r'_t$ without making the policy gradient expression biased (⋆).
- *Baseline* For any function $b : \mathcal{S} \to \mathbb{R}$, we have:

  $$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} b(s_t)\nabla_\theta \log \pi_\theta(a_t|s_t)\right] = 0$$

  Hence (⋆)we can use $(R(\tau) - b(s_t))$ instead of $R(\tau)$, and the variance is minimal when choosing $b(s) = V^\pi(s)$ (e.g. by fitting $b(s_t)$ to $R_t(\tau)$).
- *Discount* Using a discounted return $R(\tau) = \sum_{t=0}^{T} \gamma r_t$ can also reduce the variance but, unlike the 2 previous techniques, it makes the gradient biased.

More generally, the policy gradient can be expressed as:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \Psi_t\nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{6}$$

Where Ψ_t may be the reward $R(\tau)$, the reward-to-go $R_t(\tau)$, with or without a baseline (e.g. $R_t(\tau) - b(s_t)$).

◇ **Actor-Critic** The policy gradient (eq. 6) can also be expressed (⋆)with:

- $\Psi_t = Q^\pi(s_t, a_t)$: state-action value function
- $\Psi_t = A^\pi(s_t, a_t)$: advantage function
- $\Psi_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) = \delta_t$: TD error

Actor-critic methods combine the strengths of policy gradient and value-based methods by estimating a critic (Q^π, V^π and/or A^π) in order to better optimize the actor (policy π_θ). Adding a critic reduces the variance, but estimation errors introduce a bias as soon as the critic is not perfect.

◇ **Actor-Critic with compatible functions** We can try to approximate A^π with a function $f_\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. According to the *Compatible Function Approximation Theorem* [14] (⋆), if f_ϕ satisfies

$$\forall s, a, \nabla_\phi f_\phi(s, a) = \nabla_\theta \log \pi_\theta(a|s) \qquad \text{(i.e. } f_\phi \text{ is } \textit{compatible}\text{)} \tag{7}$$

$$\mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t (Q^\pi(s_t, a_t) - f_\phi(s_t, a_t)) \nabla_\theta \log \pi_\theta(a_t|s_t) \right] = 0 \tag{8}$$

then the policy gradient is exact:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t f_\phi(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

Eq. 7 can be satisfied with $f_\phi(s, a) = \nabla_\theta \log \pi_\theta(s, a)^T \phi$, and eq. 8 by solving:

$$\min_{\phi, w} \mathop{\mathbb{E}}_{s, a \sim \pi_\theta} \left[\left(Q^\pi(s, a) - f_\phi(s, a) - v_w(s) \right)^2 \right]$$

for some function $v_w : \mathcal{S} \to \mathbb{R}$ (⋆). In practice, TD is used to estimate V^π with v_w, and Q^π with $f_\phi + v_w$, so that $f_\phi(s, a) \approx A^\pi(s, a)$.

◇ **Actor-Critic with Generalized Advantage Estimation (GAE) [15]** This method only requires estimating V^π (easier to learn than $Q^\pi(s, a) \forall s, a$, especially in high dimension) to obtain an estimate of A^π. Similar to multi-steps learning, we can define the *n-steps advantage estimate*:

$$A_t^{(n)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n V^\pi(s_{t+n}) - V^\pi(s_t) \tag{9}$$

and control the bias-variance trade-off with n ($n = 1$: lower variance but higher bias when the estimate of V^π is wrong, $n \to \infty$: Monte-Carlo estimate, no bias but higher variance). Like TD(λ) learning, GAE averages the estimates and uses $\lambda \in [0, 1]$ to control the trade-off:

$$A_t^{GAE(\gamma,\lambda)} = (1 - \lambda) \sum_{n=0}^{\infty} \lambda^n A_t^{(n)} = \sum_{n=0}^{\infty} (\lambda\gamma)^n \delta_{t+n}$$

Here as well, eligibility traces are a way to implement GAE(γ, λ). Indeed, we can re-write the sample estimate of the policy gradient:

$$\sum_{t=0}^{\infty} A_t^{GAE(\gamma,\lambda)} \nabla_\theta \log \pi_\theta(a_t|s_t) = \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{n=0}^{\infty} (\gamma\lambda)^n \delta_{t+n}$$

Algorithm 5: Actor-critic with GAE, using eligibility traces

for *a number of episodes* **do**
  initialize s_0 and set $e_0(s) = 0, \forall s$
  while *episode not done* **do**
    take action $a_t = \pi(s_t)$, observe r_t and s_{t+1}
    for *all* s **do**
      $e_t(s) \leftarrow \lambda\gamma e_{t-1}(s) + \mathbb{1}_{s_t = s}$ `// this leads to` $e_t(s) = \sum_{k=0}^{t} (\lambda\gamma)^{t-k} \mathbb{1}_{s_k = s}$
      $V(s_t) \leftarrow V(s_t) + \alpha e_t(s) \delta_t$

TODO: algos? learn V^π?

◇ **A2C / A3C** *Asynchronous Advantage Actor Critic* (A3C) [16] and its synchronous version A2C are two widely used actor-critic methods. They both run n environments in parallel to get better estimates of the returns (more samples = less variance).

- Both a policy π_θ and a value function $V^\pi_{\theta_v}$ are learned. Most of the layers are shared between them (fig. 3).
- *Asynchronous*. The A3C updates are done asynchronously: each agent sends its gradient to the global network every k steps and updates its local weights after that.
- *Advantage*. Use the n-steps return (forward view, eq. 9) to learn $V^\pi_{\theta_v}$ and to estimate the advantage function A^π.

The A2C algorithm works the same way but with synchronous, deterministic updates. It waits for all agents to be done with the k steps, before performing a batch update and updating all weights at the same time.

TODO: give actual formula (and algo) of A3C/A2C: https://arxiv.org/pdf/1611.01224.pdf

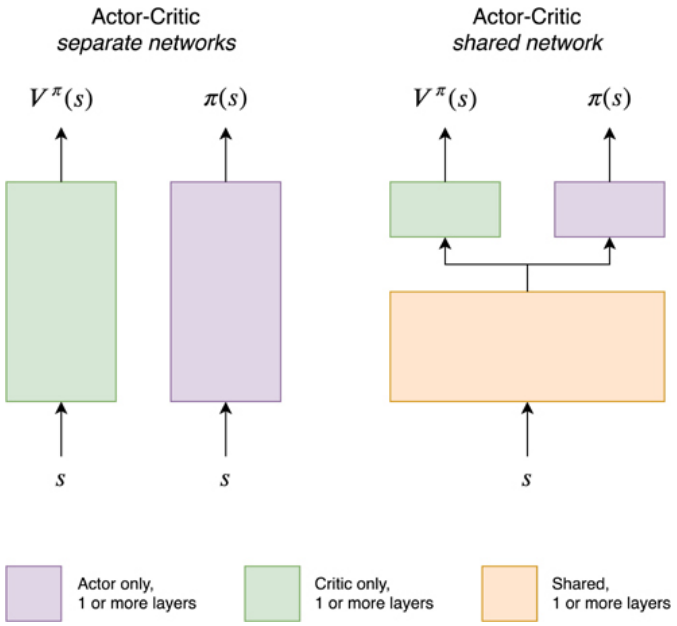

Figure 3: Network architectures for actor-critic methods. todo: re-make

◇ **Entropy** To encourage exploration and avoid converging on a deterministic policy too fast, an entropy cost can be added to the policy gradient update:

$$\theta \leftarrow \theta + \alpha(\nabla_\theta J(\theta) + \nabla_\theta H_\theta(s_t))$$

With

$$H_\theta(s_t) = -\sum_a \pi_\theta(a_t|s_t) \log \pi_\theta(a_t|s_t)$$

the entropy term to maximize.

◇ **Relative Policy Performance Bound** Instead of looking directly for $\max_\pi J(\pi)$, we can optimize the *relative performance* between π and and arbitrary policy π_{old}: $\max_\pi J(\pi) - J(\pi_{old})$. For any policies $\pi = \pi_\theta$, $\pi_{old} = \pi_{\theta_{old}}$,

we have the *relative performance bound*:

$$\begin{aligned} J(\pi) - J(\pi_{old}) &= \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi_{old}}(s_t,a_t)\right] \\ &= \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{s\sim d^{\pi}(s),a\sim\pi_{old}(a|s)}\left[\frac{\pi(a|s)}{\pi_{old}(a|s)}A^{\pi_{old}}(s,a)\right] \\ &\geq \underbrace{\frac{1}{1-\gamma}\mathop{\mathbb{E}}_{s\sim d^{\pi_{old}}(s),a\sim\pi_{old}(a|s)}\left[\frac{\pi(a|s)}{\pi_{old}(a|s)}A^{\pi_{old}}(s,a)\right]}_{=L_{\theta_{old}}(\theta)} - CD_{KL}^{max}(\pi||\pi_{old}) \end{aligned} \tag{10}$$

with

- The *discounted state distribution* of policy π

$$d^{\pi}(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_t = s|\pi) \tag{11}$$

  (TODO: explain that it is the same as the *stationary distribution* $d^{\pi}(s) = \lim_{t\to\infty} P(s_t = s \mid \pi)$)

- $D_{KL}^{max}(\pi||\pi_{old}) = \max_s D_{KL}[\pi(\cdot|s)||\pi_{old}(\cdot|s)]$
- C a factor that depends on γ and π_{old}

A number of algorithms (e.g. TRPO, PPO) maximize this lower bound of the relative performance, by keeping π (current policy) and π_{old} (old policy) close and maximizing objective $L_{\theta_{old}}(\theta)$. This has the advantages of:

1. Re-using past samples, since s, a are sampled from the old policy π_{old} in the training objective $L_{\theta_{old}}(\theta)$. This is more sample-efficient.
2. Controlling the policy updates in the space of distributions instead of the space of parameters, thanks to the KL divergence term. This makes updates smoother and training more stable.

◇ **Trust Region Policy Optimization (TRPO) [17]** TRPO considers an approximate objective to eq. 10. Since the penalty coefficient C can be very large and lead to very small updates, it uses a hard constraint of the D_{KL} (trust region), and since D_{KL}^{max} is hard to optimize it uses $\overline{D}_{KL}$ instead.

$$\max_{\theta} L_{\theta_{old}}(\theta) \quad \text{subject to} \quad \underbrace{\mathop{\mathbb{E}}_{s\sim d^{\pi_{old}}}[D_{KL}(\pi(\cdot|s)||\pi_{old}(\cdot|s)]}_{=\overline{D}_{KL}(\pi||\pi_{old})} \leq \delta \tag{12}$$

for some hyper-parameter δ. TODO: check the order of distributions in kl div Using Taylor approximation, this training objective (eq. 12) is replaced by

$$\max_{\theta} g^T(\theta - \theta_{old}) \quad \text{s.t.} \quad \frac{1}{2}(\theta-\theta_{old})^T F(\theta-\theta_{old}) \leq \delta \tag{13}$$

with

$$\begin{aligned} g &= \nabla_\theta L_{\theta_{old}}(\theta)\big|_{\theta=\theta_{old}} \\ &= \nabla_\theta J(\theta)\big|_{\theta=\theta_{old}} \qquad \text{(only at } \theta = \theta_{old}) \\ &= \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{s\sim d^{\pi_{old}},a\sim\pi_{old}(\cdot|s)}\left[\nabla_\theta \log \pi_\theta(a|s)\big|_{\theta=\theta_{old}} A^{\pi_k}(s,a)\right] \end{aligned}$$

and

$$\begin{aligned} F &= \nabla_\theta^2 \overline{D}_{KL}(\pi||\pi_{old})\big|_{\theta=\theta_{old}} \\ &= \mathop{\mathbb{E}}_{s\sim d^{\pi_{old}}}\left[\nabla_\theta^2 D_{KL}(\pi(\cdot|s)||\pi_{old}(\cdot|s))\big|_{\theta=\theta_{old}}\right] \end{aligned}$$

At each iteration, sample estimates of the gradient g and Fisher matrix F can be computed, and the constrained optimization objective of eq. 13 is solved approximately. The solution [5] (obtained by deriving KKT conditions) is:

$$\theta = \theta_{old} + \beta F^{-1} g \qquad \text{with} \qquad \beta = \sqrt{\frac{2\delta}{g^T F^{-1} g}}$$

Instead of directly inverting F (impossible with deep and large models), the conjugate gradient method solves $Fx = g$ (max d steps with $\theta \in \mathbb{R}^d$) and obtains an estimate of $x = F^{-1}g$, called the natural gradient [6]. The update steps becomes:

$$\theta = \theta_{old} + \sqrt{\frac{2\delta}{x^T F x}} x$$

Finally, a *backtracking line search* adjusts the step size in order to make the largest update that effectively improves the objective $L_{\theta_{old}}(\theta)$ and respects the constraint $\overline{D}_{KL}(\pi||\pi_{old}) \leq \delta$ (which may be violated due to the approximations).

◇ **Natural Gradient with compatible functions** When using a compatible function for the advantage estimation (eq. 7 and 8), it follows that $F\phi = \nabla_\theta J(\theta)$ (⋆), and $\phi = F^{-1}\nabla_\theta J(\theta)$ is actually the natural gradient (cf TRPO).

An incremental algorithm is suggested by [19], using learning rate α_i for the update of the critic (parameters ϕ and w, cf *Actor-Critic with compatible functions*), and learning rate β_i for the policy: $\theta \leftarrow \theta + \beta_i \phi$. This ensures that the critic converges faster than the actor. In practice however, the learning rates are hard to tune (approaches doing batch updates and adding an entropy cost have been suggested to make training more stable [20]).

◇ **Proximal Policy Optimization (PPO) [21]** Second-order optimization methods like natural gradients perform well, but are computationally expensive and complex to implement. PPO mimics the reliable trust-region update of TRPO, but using a simpler first-order method. Two versions are proposed:

- *Adaptive KL penalty.* Consider the following unconstrained objective to maximize:

$$\begin{aligned}\mathcal{L}^{KL}(\theta) &= L_{\theta_{old}}(\theta) - \beta_k \overline{D}_{KL}(\pi||\pi_{old}) \\ &= \mathop{\mathbb{E}}_{s,a\sim\pi_{old}} \left[\frac{\pi(a|s)}{\pi_{old}(a|s)} A^{\pi_{old}}(s,a) - \beta_k D_{KL}[\pi(\cdot|s)||\pi_{old}(\cdot|s)] \right]\end{aligned}$$

  With β_k an adaptive KL penalty term ensuring that $\overline{D}_{KL}(\pi||\pi_{old})$ stays close to δ most of the time (cf TODO: algo).

- *Clipped objective.*

$$\mathcal{L}^{CLIP}(\theta) = \mathop{\mathbb{E}}_{s,a\sim\pi_{old}} [\min(r_t(\theta)A_t^{\pi_{old}}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t^{\pi_{old}})]$$

  With $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t,s_t)}$ and $A_t^{\pi_{old}} = A^{\pi_{old}}(s_t, a_t)$. As illustrated in figure 4, this objective is a pessimistic lower bound of $\frac{\pi(a|s)}{\pi_{old}(a|s)} A^{\pi_{old}}(s,a)$, and effectively discourage too large improvements of policy π_θ by clipping the probability ratio. Updates in the wrong direction, i.e. that deteriorate π_θ, remain fully penalized.

Compared to TRPO, PPO makes it straightforward to share parameters between the policy and value functions. Also, re-using the sampled data $\mathcal{D}_k$ for K update steps makes it more data-efficient.

5.2 Off-Policy

Although actor-critic methods such as PPO (TODO: and A2C/A3C/TRPO?) are already re-using a couple recent samples, they are not as sample-efficient as off-policy value-based methods. *Off-policy policy gradient* methods improve this, and make it possible to define better exploration strategies.

In this section, samples are collected with a *behavior policy* μ, typically different from our learned policy π_θ.

[5] This update step does not take into account architectures using dropout or shared parameters between policy and value function.

[6] The natural gradient is effectively a gradient in the space of the distributions (using distance $D_{KL}(\pi_\theta||\pi_{\theta_{old}})$), rather than the space of parameters (with euclidean distance on parameters θ). More details here [18].

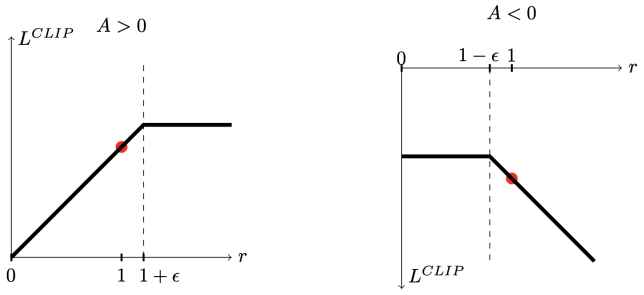

Figure 4: *PPO w/ clipped objective.* Plots showing $\mathcal{L}^{CLIP}$ as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. More detailed explanations here [22]. Taken from [21].

◇ **Off-policy policy gradient theorem** Given a behavior policy μ and its stationary distribution $d^\mu(s) = \lim_{t\to\infty} P(s_t = s \mid \mu)$ (see eq. 11), we can define an *off-policy performance objective* [7]:

$$J_\mu(\theta) = \mathop{\mathbb{E}}_{s_0\sim\mu}[V^{\pi_\theta}(s_0)]$$

The off-policy policy gradient can then be expressed as:

$$\begin{aligned}
\nabla_\theta J_\mu(\theta) &= \nabla_\theta \mathop{\mathbb{E}}_{s\sim d^\mu}\left[\sum_a \pi_\theta(a \mid s) Q^{\pi_\theta}(s,a)\right] \\
&= \mathop{\mathbb{E}}_{s\sim d^\mu}\left[\sum_a \nabla_\theta \pi_\theta(a \mid s) Q^{\pi_\theta}(s,a) + \underbrace{\pi_\theta(a \mid s)\nabla_\theta Q^{\pi_\theta}(s,a)}_{\approx 0}\right] \\
&\approx \mathop{\mathbb{E}}_{s,a\sim\mu}\left[\frac{\pi_\theta(a \mid s)}{\mu(a \mid s)} Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a \mid s)\right]
\end{aligned} \tag{14}$$

The approximation is necessary since $\nabla_\theta Q^{\pi_\theta}(s,a)$ is typically hard to compute. However, this biased gradient still improves the policy and allows converging to the right solution, at least in the tabular case (off-policy policy gradient theorem, [23]).

Note. Eq. 14 closely resembles the on-policy gradient (eq. 5), but with an added weight $\frac{\pi_\theta(a|s)}{\mu(a|s)}$, and using the stationary distribution. The *policy gradient theorem* [8] actually states that:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathop{\mathbb{E}}_{s\sim d^{\pi_\theta}}\left[\sum_a Q^{\pi_\theta}(s,a)\nabla_\theta \pi_\theta(a \mid s)\right] \\
&= \mathop{\mathbb{E}}_{s,a\sim\pi_\theta}[Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a \mid s)]
\end{aligned} \tag{15}$$

◇ **Estimating Q^π in off-policy** *Tabular case (discrete action and state spaces).* Using a slightly different TD error $\delta_t = r_t + \gamma \mathbb{E}_{a\sim\pi(|s_{t+1}}[Q(s_{t+1})] - Q(s_t,a_t)$, we can estimate Q^π with TD-0 learning, even if our samples (s_t, a_t, s_{t+1}) are not collected with policy π:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\delta_t$$

Multi-step learning is harder however, since the next actions a_{t+k} are sampled from the behavior policy μ:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \mathop{\mathbb{E}}_{s_t,a_t,s_{t+1}\sim\mu,\forall t}\left[\sum_{t=0}^{k}\gamma^t\left(\prod_{i=0}^{t} c_i\right)\left(r_t + \gamma \mathop{\mathbb{E}}_{a_{t+1}\sim\pi}[Q(s_{t+1},a_{t+1})] - Q(s_t,a_t)\right) \mid s_0 = s, a_0 = a\right]$$

With c_i a coefficient that can be

[7] This is slightly different from the on-policy reward that we have been considering so far: $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \mathbb{E}_{s_0\sim\rho_0}[V^{\pi_\theta}(s_0)]$.
[8] Proof can be found here or here

- $c_i = \lambda$: regular $Q(\lambda)$ algorithm, but biased since we are off-policy, and only converges if μ and π are close
- $c_i = \frac{\pi(a_i|s_i)}{\mu(a_i|s_i)}$: unbiased thanks to importance sampling, but not stable (high variance)
- $c_i = \lambda\pi(a_i \mid s_i)$: *Tree Backup* [24] works even when μ and π are different, but not efficient due to premature "cuts" if $\pi(a_i \mid s_i)$ becomes small.
- $c_i = \lambda \min(1, \frac{\pi(a_i|s_i)}{\mu(a_i|s_i)})$: *Retrace* [25] combines the best of both worlds, with controlled variance $c_i \leq \lambda$), no premature "cuts", and even has guarantees of convergence.

Approximate case (e.g. continuous state space). Here as well we can use the Retrace coefficient c_i, but learning $Q_\theta(s, a)$ requires a target value Q^{ret} in order to compute an error (e.g. least square) and a gradient. For each trajectory τ, the target can be defined recursively (starting from the final value $Q^{ret}(s_{T-1}, a_{T-1}) = r_{T-1}$):

$$Q^{ret}(s_t, a_t) = r_t + \gamma c_{t+1}[Q^{ret}(s_{t+1}, a_{t+1}) - Q_\theta(s_{t+1}, a_{t+1})] + \gamma E_{a\sim\pi} Q_\theta(s_{t+1}, a)$$

◇ **Actor-Critic with Experience Replay (ACER)** [26] Similar to on-policy actor critic methods (e.g. A3C/A2C), learning an estimate of Q^π in eq. 14 can reduce the variance. ACER decomposes $\nabla_\theta J_\mu(\theta)$ into

$$\begin{aligned}\nabla_\theta J_\mu(\theta) = \mathop{\mathbb{E}}_{s_t\sim\mu}\Bigg[&\mathop{\mathbb{E}}_{a_t\sim\mu}[\min(c, \omega_t(a_t))Q^\pi(s_t, a_t)\nabla_\theta \log \pi_\theta(a_t \mid s_t)] \\ &+ \mathop{\mathbb{E}}_{a\sim\pi}\left[\max\left(0, \frac{\omega_t(a) - c}{\omega_t(a)}\right) Q^\pi(s_t, a)\nabla_\theta \log \pi_\theta(a \mid s_t)\right]\Bigg]\end{aligned}$$

With c a hyper-parameter and $\omega_t(a) = \frac{\pi(a|s_t)}{\mu(a|s_t)}$ the importance sampling ratio. The left term clips this ratio to bound the variance, the right term ensures that the estimate is unbiased. Using our estimates of Q^π, and substracting our estimate V_{θ_v} of V^π to further reduce the variance, the gradient estimate becomes:

$$\begin{aligned}g_t^{\text{acer}} = &\min(c, \omega_t(a_t))(Q^{ret}(s_t, a_t) - V_{\theta_v}(s_t))\nabla_\theta \log \pi_\theta(a_t \mid s_t) \\ &+ \mathop{\mathbb{E}}_{a\sim\pi}\left[\max\left(0, \frac{\omega_t(a) - c}{\omega_t(a)}\right)(Q_{\theta_v}(s_t, a) - V_{\theta_v}(s_t))\nabla_\theta \log \pi_\theta(a \mid s_t)\right]\end{aligned}$$

In addition to this, ACER does trust-region updates (like TRPO) but actually considers the gradient $g_{\phi_\theta(s_t)}$ w.r.t. the policy distribution parameters $\phi_\theta(s)$ (e.g. logits or probability vector of discrete actions) instead of the model parameters θ. With π_{θ_a} a smoothly updated average policy, the following optimisation problem is solved:

$$\min_z \frac{1}{2}||g_{\phi_\theta(s_t)} - z||^2 \quad \text{s.t.} \quad \nabla_{\phi_\theta(s_t)} D_{KL}[\pi_{\theta_a}(\cdot \mid s_t) \mid\mid \pi_\theta(\cdot \mid s_t)]^T z \leq \delta$$

Which has a closed form solution z^*. Policy parameters θ are updated using the gradient $g_\theta = z^* \nabla_\theta \phi_\theta(s_t)$.

5.3 Continuous Action Spaces

Previous policy gradients methods can be used with either a discrete or continuous action space (e.g. $\mathcal{A} \in \mathbb{R}^d, \pi(a \mid s) = \mathcal{N}(a; \mu_\theta(s), \Sigma_\theta(s))$). But continuous action spaces actually allow taking the gradient w.r.t. actions $\nabla_a Q_\theta(s, a)$ and deriving specific algorithms.

◇ **Deterministic Policy Gradients (DPG/DDPG)** [27] [28] In addition to a continuous action space, DPG considers a non-stochastic policy taking actions $a = \mu_\theta(s)$. The on-policy policy gradient becomes:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{s\sim d^{\pi_\theta}}\left[\nabla_a Q^{\pi_\theta}(s, a)\Big|_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta(s)\right] \tag{16}$$

And critic Q^{π_θ} can be learned with SARSA updates.

In order to add exploration and to be more sample-efficient, an off-policy version is also proposed:

$$\nabla_\theta J_\beta(\theta) \approx \mathop{\mathbb{E}}_{s\sim d^\beta}\left[\nabla_a Q^{\pi_\theta}(s, a)\Big|_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta(s)\right] \tag{17}$$

Doing the same approximation as in 14 (and this time we call β the behavior policy). Critic can be learned with Q-learning updates.

DDPG simply implements the off-policy version of DPG with deep neural networks for Q^π and μ_θ, and adding noise to μ_θ to construct the exploration policy. Stability is achieved by using tricks from DQN: replay buffer, target networks and soft updates. A couple extensions use additional tricks: 2 value networks Q^π like Double DQN (TD3 [29]), Prioritized replay and a distributional critic (D4G [30]).

◇ Policy gradient with an off-policy critic (Q-Prop)

References

[1] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[2] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

[3] Emilie Kaufmann, Nathaniel Korda, and Rémi Munos. Thompson sampling: An asymptotically optimal finite-time analysis. In *International conference on algorithmic learning theory*, pages 199–213. Springer, 2012.

[4] Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *International Conference on Machine Learning*, pages 127–135. PMLR, 2013.

[5] Arryon D Tijsma, Madalina M Drugan, and Marco A Wiering. Comparing exploration strategies for q-learning in random stochastic mazes. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016.

[6] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[9] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[10] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[11] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23:2613–2621, 2010.

[12] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[13] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

[14] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[15] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[17] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[18] Julien Vitay. Deep reinforcement learning - natural gradients. `https://julien-vitay.net/deeprl/NaturalGradient.html`. Accessed: 2021-07-24.

[19] Shalabh Bhatnagar, Mohammad Ghavamzadeh, Mark Lee, and Richard S Sutton. Incremental natural actor-critic algorithms. *Advances in neural information processing systems*, 20:105–112, 2007.

[20] Joni Pajarinen, Hong Linh Thai, Riad Akrour, Jan Peters, and Gerhard Neumann. Compatible natural gradient policy search. *Machine Learning*, 108(8):1443–1466, 2019.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[22] Ppo clipped objective, explained. https://stackoverflow.com/a/50663200. Accessed: 2021-07-25.

[23] Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

[24] Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[25] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G Bellemare. Safe and efficient off-policy reinforcement learning. *arXiv preprint arXiv:1606.02647*, 2016.

[26] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[27] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[28] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[29] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[30] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.