本文汇总了几个 deep reinforcement learning 的方法

一．基于 DQN

1. DQN

和传统 rl 一样，DQN 的任务同样是学习如何选择 action 使 return 的期望最大。使用了 CNN 来近似 optimal action-value function，据此在 state 下可以选择使 Q*(s,a)其最大的 action。

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi\right]$$

我们知道基于时序差分的 Q-learning 方法为：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

现在我们不用大量采样的方法对 Q 进行估计，而是将 Q 建模为一个参数为θ的神经网络，在第 i 个迭代的更新θ使用的 loss function 为：

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

approximate value function 为 Q(s,a,θ)，state、action 作为 Q 的输入，括号中前两项是对应的 target，所以这是一个回归问题，本文使用 l2 loss。使用网络近似的方法极大地提高了训练的效率，不需要每个 s-a 对出现过很多次然后基于采样求均值，对只出现一次甚至没有出现过的 state 也可以进行估计，具有很好的泛化性。DQN 使用了下面两个技巧：

1. Target network

如果令 target 中的 Q network 和得到 value 估计值的 network 是同一个，会造成训练的不稳定，当我们更新θ使 $Q(s_t,a_t)$增大，这时可能也同样增大了 $Q(s_{t+1},a_t)$，也就改变了这次训练的 target。所以我们将得到 target 的网络固定住，称为 target network Q^，当 Q 更新 C 次后，将 Q 的参数拷贝给 Q^。

2. Experience Replay

我们的训练数据是(s_t,a_t,r_t,s_{t+1})，现在我们使用一个 buffer D 存这种数据。在每个 iteration 中，选择 action 与环境交互，将新的数据存入 buffer，然后训练网络并不只使用新得到的数据，而是从 buffer 中采样一个 batch 来训练网络。这样做的优势是 1. 提高了数据利用率，产生的数据可能会反复使用 2. 使 batch 中的数据更加 diverse，因为 batch 中包含不同时期的 policy 采样的 action。

注意，这种更新方式其实是 off-policy 的，因为训练当前 policy 的数据是由早期（不同）policy 得到的，但是理论上这不会产生误差，原因是我们只使用(s_t,a_t,r_t,s_{t+1})这个片段进行训练，而不是整个 trajectory（基于 MC 方法）。回忆 off-policy 的 MC 方法，用重要性采样求 state-value，我们将 behavior policy 产生的 state-action 对的 return 加权，权值是 $P(s_t,a_t,r_t,s_{t+1}\ldots\ldots| s_t)$在两个 policy 下的比值，由于环境动态是相同的，所以比值为π(ai|si)从 i=t 开始连乘的比值；而要求 action value，$P(s_t,a_t,r_t,s_{t+1}\ldots\ldots| s_t,a_t)$在两个 policy 下的比值为π(ai|si)从 i=t+1 开始连乘。现在只有一个片段，重要性采样比为 1。所以这种方法的影响只是对当前 policy 不会产生概率小的 action 进行的更多的探索，而不会产生误差。

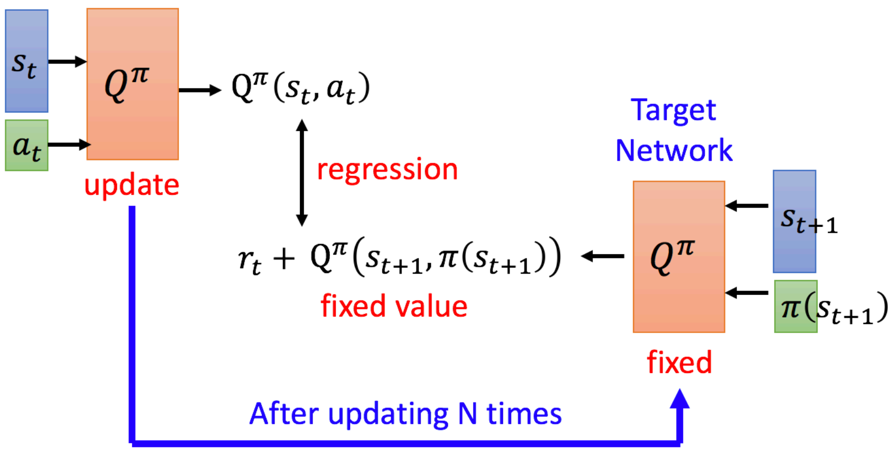
Q-learning 是一种可以进行 off-policy 学习的方法，而其他方法比如 sarsa，n-step 方法，actor-critic 方法只能进行 on-policy 学习，而不能利用 experience Replay。比如 sarsa 是基于 action value 的 bellman 方程：

$$Q^{\pi}(s_t,a_t)=\mathbb{E}_{r_t,s_{t+1}\sim E}\left[r(s_t,a_t)+\gamma\,\mathbb{E}_{a_{t+1}\sim\pi}\left[Q^{\pi}(s_{t+1},a_{t+1})\right]\right]$$

saras 利用五元组(s_t,a_t,r_t,s_{t+1}, a_{t+1})由上式进行 value estimate，这个五元组由真实的轨迹给出，而这个轨迹是通过贪心的选择 action 产生，也就相当于 value improvement。所以整体是一个 value iteration 的过程。其中，关于 state 和 action 两部分的期望都用采样代替，而关于 action 的期望不用采样代替的称为 excepted sarsa。上式的迭代过程相当于策略评估步骤，而评估出 state 下不同 action 的 q value 后，策略会修改为选择 q 最大的 action，进一步影响对 action 求期望的步骤。

而 Q-learning 使用四元组(s_t,a_t,r_t,s_{t+1})，a_{t+1} 由一个确定性函数贪心的产生 μ:S→A，（真实的轨迹中 next action 并不一定是 a_{t+1}）也就是说里面的关于 action 的期望就避免了：$Q^{\mu}(s_t,a_t)=\mathbb{E}_{r_t,s_{t+1}\sim E}\left[r(s_t,a_t)+\gamma Q^{\mu}(s_{t+1},\mu(s_{t+1}))\right]$。这个迭代本身相当于 value iteration 步骤，因为对 action 取 max 代替了期望。随着 q 的改变，产生样本的 policy 也会修改，但这个 policy 本身只是提供一个探索的方向。可以当做是 value iteration 的副产品，可以不当做是 policy improvement 步骤。

这是 Q-learning 基于的公式，这时由于期望只与环境有关，所以是可以进行 off-policy 学习的。对于那些 on-policy 方法，不能使用 Experience Replay 来减小数据的相关性，可以异步的并行执行多个 agent，见 A3C。



最后给出算法伪代码：

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

2. Double DQN

在 Q learning 的 target 中有一个 max 操作，使用了同一个 value function 挑选 action 和评估 action：

$$Y_t^{\mathrm{Q}} = R_{t+1} + \gamma Q(S_{t+1}, \operatorname{argmax} Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}_t)$$

这会使算法最倾向于选择最被高估的 action 的 value 值，target 是高估的，从而导致 estimated value 被高估。也就是说，由于将 action 的估计值中最大的那一个当做是真实值最大的 action 的估计，导致我们学习到的 value 值总是被高估。解决这个问题的方法是将选择最大的 action 和对这个 action 进行估计的两个任务分开：

$$Y_t^{\mathrm{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t)$$

其中基于 θ 确定的当前 value 来使用 greedy policy 选择 action，θ‘用来对确定该 action 的 value。更新的是当前 value 的参数 θ，两个 value function 轮流交换任务。

在 DQN 中，我们天然的拥有两个网络。我们可以用 target network 对 action 进行评估，此时由于 target network 本身也会周期性的进行更新，所以不需要交换两个网络的功能：

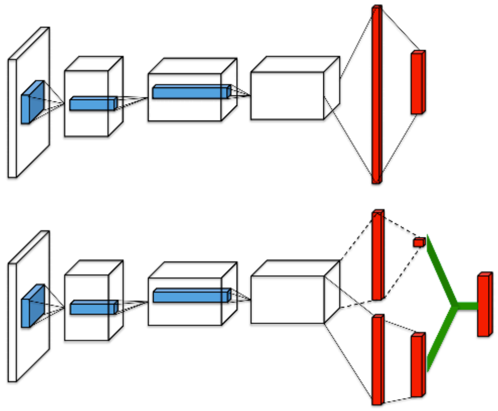
$$Y_t^{\mathrm{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^{-})$$

引申：如果不用 maxQ(s’,a)，而是也用ϵ greedy policy 选 a’，则可以看做是在 policy 下求期望，则不会由于选择最被高估的 action，是不是也能解决 positive bias 的问题？

3. Dueling DQN

这是一种适用于 DQN 的新的网络结构（下图二），dueling network Q(s,a)有

两个部分组成，一个是 state value function V(s)，另一个是在某个 state 下，衡量 action 的 function A(s,a)，最后将两个部分结合起来得到 Q。本网络的优势是提高了对 state 下不同 action 评估的泛化性，在更新 V(s)时对没出现的 action 的 value 也进行了更新。



现在已知 Q(s,a) = V(s)+A(s,a)，由于 $V(s) = E_a[Q(s,a)]$，所以 $E_a[A(s,a)] = 0$。网络的输入为 state，V 网络输出一个标量，A 网络输出一个 vector，每一维对应一个 action；两部分共同的卷积层参数为 θ，两部分各自的全连接层参数分别为 β,α，则聚合模型为 Q(s,a|θ,β,α) = V(s|θ,β)+A(s,a|θ,α)。但是实际这样做效果不佳，因为一个 Q function 并不对应着唯一的 state value 和 A 的值，我们将 V 任意增大一个常数，同时为该 state 对应的所有 action 减小这个常数，仍能得到原来的 Q。实际上，本方法贡献在于提出 V 这个网络，是模型对不同 action 有更好的泛化性，但由于上述问题，我们不确定 V 能学到多少信息，如果 V 不存在任何信息，A 网络就相当于原来的 Q 网络，模型退化为 DQN。

本文提出的解决办法就是为 A 网络增加限制，迫使 V 学习更多。前面提到如果 V 是 state value，则 $V(s) = E_a[Q(s,a)]$，所以 $E_a[A(s,a)] = 0$，我们为 A 网络加一个 normalization 层，令其对所有的 action 的加和为 0：

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)\right)$$

对 A 的限制也可以为对所有 action 令最大的 advantage 为 0，此时 value 是 optimal value function：

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a'\in|\mathcal{A}|} A(s,a';\theta,\alpha)\right)$$

注意，dueling DQN 只是将 V 和 A 作为网络结构的一部分，loss function 仍和 DQN 相同。

最后，上述方法都基于 one step Q learning，因为这些方法都是基于 one step return 来更新 action value。这样的缺点是得到的 reward 只直接影响上一步的 state-action 对的 value 值 Q(s,a)，其他的 state-action 对的 value 值是通过更新后的 Q(s,a)间接的收到影响。由于需要通过很多次更新才能将 reward 传递给前面的 state-action，学习过程非常缓慢。n-step Q learning 可以更有效率的传播 reward。

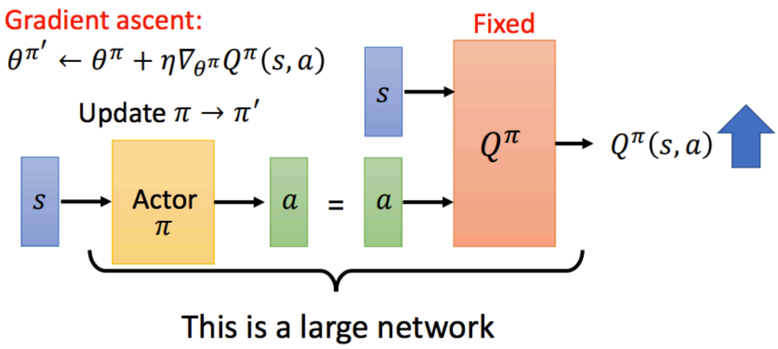
二．Policy gradient

4. Deterministic Policy Gradient

DPG 算法将 policy gradient framework 拓展到了确定性 policy。

我们知道 Q-learning 适用于动作空间是离散的情况，因为 policy improvement 步骤要选取 a = arg max$_a$ Q$_\pi$(s,a)，对于离散空间，比较每一个 action 的 q value 值即可，而对于连续的 action 空间，需要找到使 Q 全局最大的 action，这可以看做是解一个 optimization 问题，我们将 a 当做参数利用 gradient ascent 来解，但是容易陷入局部最优，且每个 step 都要进行一次 gradient ascent，运算量大。

我们而一种简单的代替的方法是，另外学习一个网络 actor 来解这个 optimization 问题，该网络输入为 s 输出 a，是一个确定性的 policy，记做 a = μ_θ(s)。我们希望训练该网络输出使 q value 最大的 a，将该 policy 参数向 Q 的梯度的方向移动。



对于每一个 state，θ 都要用$\nabla_\theta Q^{\mu^k}(s,\mu_\theta(s))$更新，每个 state 会给出不同的更新方向，所以利用 state 在 ergodic MDP 下的稳态分布求期望：

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}_{s\sim\rho^{\mu^k}}\left[\nabla_\theta Q^{\mu^k}(s,\mu_\theta(s))\right]$$

根据链式法则：

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}_{s\sim\rho^{\mu^k}}\left[\nabla_\theta \mu_\theta(s)\ \nabla_a Q^{\mu^k}(s,a)\Big|_{a=\mu_\theta(s)}\right]$$

然而更改了 policy，state 的分布 ρ_μ 也随之改变，所以我们不能保证 policy 是提升的。但是类似于 policy gradient theorem，我们不用求出 state 分布的梯度，而是依据 policy 采样，会以当前 policy 下 state 的稳态分布采样到 state。用确定性的 policy，现在我们得到的 performance 为：

$$\begin{aligned} J(\mu_\theta) &= \int_{\mathcal{S}} \rho^{\mu}(s) r(s,\mu_\theta(s)) \mathrm{d}s \\ &= \mathbb{E}_{s\sim\rho^{\mu}}\left[r(s,\mu_\theta(s))\right] \end{aligned}$$

比较 REINFROCE 方法的 performance：

$$\begin{aligned} J(\pi_\theta) &= \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \pi_\theta(s,a) r(s,a) \mathrm{d}a\mathrm{d}s \\ &= \mathbb{E}_{s\sim\rho^{\pi},a\sim\pi_\theta}\left[r(s,a)\right] \end{aligned}$$

该 performance 相当于将 policy gradient 方法中的 performance 对 action 求期望的部分由一个确定的 policy 产生的 action 替代，这是因为如果 policy 具有随机性，会涉及到对 action 的采样，则 Q 的梯度无法直接传过来。梯度为：

$$\nabla_\theta J(\mu_\theta) = \int_S \rho^\mu(s)\nabla_\theta\mu_\theta(s)\ \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}$$
$$= \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta\mu_\theta(s)\ \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right]$$

我们知道 policy gradient 算法是调整 policy 的参数 θ 使其向 performance gradient 的方向移动，performance 在没有 episode 的情况下为 average reward，有 episode 时定义为初始 state 的 value，得到的都是下式，但是 ρ 的定义不同：

$$J(\pi_\theta) = \int_S \rho^\pi(s)\int_{\mathcal{A}} \pi_\theta(s,a) r(s,a)\mathrm{d}a\mathrm{d}s$$
$$= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[r(s,a)]$$

$$\nabla_\theta J(\pi_\theta) = \int_S \rho^\pi(s)\int_{\mathcal{A}} \nabla_\theta\pi_\theta(a|s) Q^\pi(s,a)\mathrm{d}a\mathrm{d}s$$
$$= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_\theta \log\pi_\theta(a|s) Q^\pi(s,a)]$$

事实上，deterministic policy gradient 是 stochastic policy gradient 的一个特例，当 action 空间是连续的，policy π 是在该空间上的概率密度函数，比如建模为高斯分布：

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{1}{\sigma(s,\boldsymbol{\theta})\sqrt{2\pi}}\exp\left(-\frac{(a-\mu(s,\boldsymbol{\theta}))^2}{2\sigma(s,\boldsymbol{\theta})^2}\right)$$

当 stochastic policy 的均值为 deterministic policy，方差 ρ→0 时，则两个 policy 相等$\pi_{\mu_\theta,0} \equiv \mu_\theta$。此时 stochastic policy gradient 收敛到 deterministic policy gradient：

$$\lim_{\sigma\downarrow 0}\nabla_\theta J(\pi_{\mu_\theta,\sigma}) = \nabla_\theta J(\mu_\theta)$$

以上是 DGP 的理论基础，DGP 可用于 on-policy 情况或者 off-policy 情况，但是这两种方法在学习 critic 时，由于使用了函数来近似真正的 value function，可能会引入 bias，可能会存在收敛问题。

On-policy Deterministic actor-critic，on-policy 情况下可以使用 on-policy 算法，比如 sarsa，也可以用于 off-policy 算法，比如 Q-learning。Critic 使用 sarsa 更新

$$\delta_t = r_t + \gamma Q^w(s_{t+1}, a_{t+1}) - Q^w(s_t, a_t)$$
$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t)$$
$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta\mu_\theta(s_t)\ \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)}$$

off-policy 情况下，使用 behavior policy β 选择 action，对于原本具有随机性的 actor-critic 方法，performance 定义修改为在 behavior policy 得到的 state 分布下，用 target policy 得到的 state value 的期望值：

$$J_\beta(\pi_\theta) = \int_S \rho^\beta(s) V^\pi(s)\mathrm{d}s$$
$$= \int_S\int_A \rho^\beta(s)\pi_\theta(a|s) Q^\pi(s,a)\mathrm{d}a\mathrm{d}s$$

$$\nabla_\theta J_\beta(\pi_\theta) \approx \int_S\int_{\mathcal{A}} \rho^\beta(s)\nabla_\theta\pi_\theta(a|s) Q^\pi(s,a)\mathrm{d}a\mathrm{d}s \qquad (4)$$
$$= \mathbb{E}_{s\sim\rho^\beta, a\sim\beta}\left[\frac{\pi_\theta(a|s)}{\beta_\theta(a|s)}\nabla_\theta \log\pi_\theta(a|s) Q^\pi(s,a)\right]$$

而在确定性 policy 的情况下：

$$\begin{aligned} J_\beta(\mu_\theta) &= \int_{\mathcal{S}} \rho^\beta(s) V^\mu(s) \mathrm{d}s \\ &= \int_{\mathcal{S}} \rho^\beta(s) Q^\mu(s, \mu_\theta(s)) \mathrm{d}s \end{aligned}$$

$$\begin{aligned} \nabla_\theta J_\beta(\mu_\theta) &\approx \int_{\mathcal{S}} \rho^\beta(s) \nabla_\theta \mu_\theta(a|s) Q^\mu(s,a) \mathrm{d}s \\ &= \mathbb{E}_{s\sim\rho^\beta}\left[\nabla_\theta \mu_\theta(s)\ \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right] \end{aligned}$$

可见由于此时对 action 求积分的步骤没有了，所以在学习 actor 时避免了重要性采样；而使用 off-policy 的算法学 critic，比如 Q-learning，则在 critic 学习中也避免了重要性采样。

$$\begin{aligned} \delta_t &= r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t) \\ w_{t+1} &= w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \\ \theta_{t+1} &= \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t)\ \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)} \end{aligned}$$

deep DGP 利用 DQN 对 DGP 进行了修改，使其适用于在大的 state 和 action 空间中使用神经网络函数近似器。

1. 用 nn 来进行 rl 学习的一个问题是，大部分 optimization 算法都假设样本是 i.i.d.的，而利用一个 agent 顺序的得到采样显然是不满足这个假设的。为了解决这个问题 DQN 使用了 replay buffer，每次更新时从 buffer 中采样一个 batch，这同样可用在 DPG 中，因为这是一个 off-policy 的算法。
2. 同样使用 target network，但是不是隔 C 步将参数复制过去，而是用了一种 soft 的更新方式：θ’ = τθ+(1-τ)θ’，使训练更稳定
3. 由于是 off-policy 方法，可以独立的用一个噪声去进行探索，得到采样，然后加入 actor policy 即可。

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2)$
    Update the actor policy using the sampled gradient:

$$\nabla_{\theta^\mu} \mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

比较 DDGP 与 actor-critic 方法，后者令 actor 自己去选择 action，critic 给出这个选择是否是好的 action；而前者则是直接引导 actor 去选择能够得到较大的 value 的 action。

下面分析一下 action 是连续还离散空间如何影响我们选择确定性的还是有随机性的 policy，注意 REINFORCE 是一个离散 action 空间学习具有随机性 policy 的方法，DPG 是一个连续空间学习确定性 policy 的方法：

action 空间是离散的情况：

1. Q-learning（value-based）主要适用于这种情况，在连续空间中选择使 Q 最大的 action 是困难的，value-based 方法只能学到确定性的 policy（ϵ-greedy 方法是为了更好的探索，找到这个最优 policy），因为只有在这个 action 下我们预计得到最大的 return。
2. Policy-based 方法中，可以学习具有随机性的 policy（REINFORCE），π(a|s)是一个分布函数，按照这个概率选择 action 后，可以利用 Q(s,a)对该选择评估好坏，以更新 policy。**而不可以使用确定性的 policy，因为用 Q 对 a 求梯度没有意义，无法指导 policy 的学习。**

Action 空间连续：

可以学习确定性 policy（DPG 算法应用的唯一情况），此时可以用 Q 对 a 求梯度，然后这个梯度指导 policy 的参数更新。理论上也可以学习具有随机性的 policy，此时 π(a|s)是一个概率密度函数，仍然可以据此选择 action，但是不能用 REINFORCE 方法训练 policy，因为 π(a|s)不再是一个概率值，无法直接用 log π(a1|s)求梯度使 a1 出现的概率增大，这种情况要先对 π(a|s)建模，比如高斯分布。