Reinforcement Learning

Chapter 6: Actor Critic Methods

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Fall 2025

Learning *Deterministic* Policy

From model-based RL we know that: *the optimal policy can be deterministic*

Why don't we train a policy network that learns a deterministic policy?

+ *You are contradicting yourself! You said that stochastic policy is a general case that includes deterministic policies as well! Now you want to get back to a deterministic policy?!*
– *Well! You're right! But there will be no harm in learning a deterministic policy! It might only be less effective!*
+ *Why we should do it then?*
– *It could give us some benefits, especially when we have continuous action-space*

Learning *Deterministic* Policy

With continuous action-space, policy is a density function

- *With discrete action-space, we can show policy by a finite vector*[1]

$$\mathbf{x}(s) \longrightarrow \boxed{\pi_{\boldsymbol{\theta}}(\cdot|s)} \longrightarrow \begin{bmatrix} \pi_{\boldsymbol{\theta}}(a^1|s) \\ \vdots \\ \pi_{\boldsymbol{\theta}}(a^M|s) \end{bmatrix}$$

- *Unlike discrete action-space, we cannot do this with continuous actions*
  - ↳ *We should learn a function from state feature, e.g.,*

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(a - \mathrm{DNN}(\mathbf{x}(s)|\boldsymbol{\theta}))}{2\sigma^2}\right\}$$

  - ↳ *We then sample from this learned density function, e.g.,*
    - ↳ *Draw a sample from Gaussian distribution with mean* $\mathrm{DNN}(\mathbf{x}(s)|\boldsymbol{\theta})$ *and variance* σ^2

[1] *We have seen this in Assignment 3*

Learning *Deterministic* Policy

There are several things that could go wrong

- *What if the generated sample is out of accepted range?*
  - ↳ *Our sample from Gaussian distribution is extremely large*
  - ↳ *In sensitive control settings, this could harm the system*
- *What if we only try a few samples?*
  - ↳ *We don't see the probabilities as opposed to continuous actions*
  - ↳ *We then cannot really reject too many samples*

With continuous actions, we usually prefer to learn a deterministic policy: its main feature is that it can be represented by a single action $a^\star$

$$\pi_{\boldsymbol{\theta}}\left(a|s\right) = \begin{cases} 1 & a = a^\star \\ 0 & a \neq a^\star \end{cases}$$

Deterministic Policy Network

Considering a deterministic policy, we only need to learn an estimate of optimal action in each state: we can revise our policy network into a deterministic policy network

$$\mathbf{x}(s) \longrightarrow \boxed{\mu_{\boldsymbol{\theta}}(\cdot)} \longrightarrow \hat{a}^\star(s)$$

Deterministic Policy Network

Deterministic policy network maps a state-features into a single action and can be realized by a DNN with input being the state feature representation and a single output

Deterministic Policy Network: *Training*

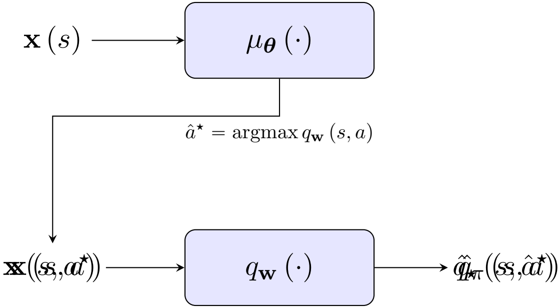
+ *How should we train these networks? Similar to other policy networks?*
– *We can look at them as a special form of policy networks and do the same thing, yes! However, it turns out that in this special case, there are better ways to do it! Especially as we always implement them actor-critic*
+ *So, we should start all over again?!*
– *Not really! We should basically use what we learned for DQL*

Deterministic Policy Network: *Training*

Recall the property of optimal policy: it gives maximum value and action-value

---

If we have a Q-network that estimates optimal action-value, we can say



The output satisfies

$$\hat{q}_\star(s, \hat{a}^\star) = \max_a \hat{q}_\star(s, a)$$

Deterministic Policy Network: *Training*

So, in actor-critic form with a Q-network, we could train the deterministic policy network as

$$\boldsymbol{\theta}^\star = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, Q_{\mathbf{w}}\left(s, \mu_{\boldsymbol{\theta}}\left(s\right)\right)$$

which we can solve using gradient ascent by updating as

$$\begin{aligned} \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} Q_{\mathbf{w}}\left(s, \mu_{\boldsymbol{\theta}}\left(s\right)\right) \\ &\leftarrow \boldsymbol{\theta} + \alpha \underbrace{\frac{\partial}{\partial a} Q_{\mathbf{w}}\left(s, a\right)\big|_{a=\mu_{\boldsymbol{\theta}}(s)}}_{\text{Backprop over Q-Net}} \quad \underbrace{\nabla \mu_{\boldsymbol{\theta}}\left(s\right)}_{\text{Backprop over policy}} \end{aligned}$$

Moral of Story

As long as we have an estimator of optimal action-value function, we can train deterministic policy network very easily!

Deterministic Policy Gradient

+ *But how can we can find such an estimator?*
– *Well! We have done this before!*
+ *You mean in DQL?!*
– *Exactly! In Q-learning we use Bellman's optimality equation to estimate optimal action-value function: we can do the same here*

Recall that Bellman's optimality equation indicate that

$$q_\star(S_t, A_t) = R_{t+1} + \gamma \mathbb{E}_{S_{t+1} \sim p(\cdot|S_t, A_t)} \left\{ \max_a q_\star(S_{t+1}, a) \right\}$$

and if we know the action $a^\star = \operatorname{argmax}_a q_\star(S_{t+1}, a)$*, we could write*

$$q_\star(S_t, A_t) = R_{t+1} + \gamma \mathbb{E}_{S_{t+1} \sim p(\cdot|S_t, A_t)} \left\{ q_\star(S_{t+1}, a^\star) \right\}$$

Deterministic Policy Gradient

If we use our deterministic policy network in one time step we sample

$$S_t, \mu_{\boldsymbol{\theta}}(S_{t+1}) \xrightarrow{R_{t+1}} S_{t+1}$$

We can then sample an estimator of optimal action-value at $a = \mu_{\boldsymbol{\theta}}(S_{t+1})$

$$\hat{Q}_t = R_{t+1} + \gamma Q_{\mathbf{w}}(S_{t+1}, \mu_{\boldsymbol{\theta}}(S_{t+1}))$$

Once we are over with sample trajectory: we update Q-network to minimize loss

$$\mathcal{L}(\mathbf{w}) = \frac{1}{T} \sum_{t=0}^{T-1} \left(Q_{\mathbf{w}}(S_t, \mu_{\boldsymbol{\theta}}(S_t)) - \hat{Q}_t \right)^2$$

And the life is much easier as compared to TRPO and PPO ☺

Deterministic Policy Gradient

We do very well know how to do this

$$\begin{aligned}
\mathbf{w} &\leftarrow \mathbf{w} - \beta \nabla \mathcal{L}(\mathbf{w}) \\
&\leftarrow \mathbf{w} + \beta \mathtt{mean}\left[\left(\hat{Q}_t - Q_{\mathbf{w}}\left(S_t, \mu_{\boldsymbol{\theta}}\left(S_t\right)\right)\right) \nabla Q_{\mathbf{w}}\left(S_t, \mu_{\boldsymbol{\theta}}\left(S_t\right)\right)\right] \\
&\leftarrow \mathbf{w} + \beta \mathtt{mean}\left[\Delta_t \nabla Q_{\mathbf{w}}\left(S_t, \mu_{\boldsymbol{\theta}}\left(S_t\right)\right)\right]
\end{aligned}$$

where $\Delta_t = \hat{Q}_t - Q_{\mathbf{w}}\left(S_t, \mu_{\boldsymbol{\theta}}\left(S_t\right)\right)$ *is the TD error*

Alternating between the two update rules, we end up with a

Deterministic Policy Gradient $\equiv$ *DPG*

algorithm: there are various DPG algorithms; we take a look into the famous one

A Basic DPG Algorithm

We can use these updates to write a simple online DPG algorithm

DPG_v1():

Initiate with $\boldsymbol{\theta}$ and $\mathbf{w}$, as well as factor $\alpha < 1$ and learning rate β
Initiate some initial state S_0 and draw action A_0 as $A_0 \leftarrow \mu_{\boldsymbol{\theta}}(S_0)$
while** interacting **do
  Sample a time step $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$
  Draw the next optimal action as $A_{t+1} \leftarrow \mu_{\boldsymbol{\theta}}(S_{t+1})$
  Compute $\Delta = R_{t+1} + \gamma Q_{\mathbf{w}}(S_{t+1}, A_{t+1}) - Q_{\mathbf{w}}(S_t, A_t)$
  Update value network as $\mathbf{w} \leftarrow \mathbf{w} + \beta \Delta \nabla Q_{\mathbf{w}}(S_t, A_t)$
  Update policy network as $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \frac{\partial}{\partial a} Q_{\mathbf{w}}(S_t, a)\big|_{a=A_t} \nabla \mu_{\boldsymbol{\theta}}(S_t)$
  Go for next state $S_t \leftarrow S_{t+1}$
  if** S_t is terminal **then
    Draw a new random state S_0 and $A_0 \leftarrow \mu_{\boldsymbol{\theta}}(S_0)$
  end if
end while

Basic DPG: *Practical Challenges*

Using our knowledge, we can easily detect challenges of this basic algorithm

- *Lack of exploration $\rightarrow$ ϵ-greedy improvement*
  - ↳ *We follow blindly the deterministic policy network*
  - ↳ *We do not give any chance for exploration*
  - ↳ *This can quickly stick us to a bad locally-optimal deterministic policy*
- *High-variance gradient estimators $\rightarrow$ experience replay*
  - ↳ *It's online and hence update the networks with single time step samples*
  - ↳ *We need more samples to compute better estimators*
  - ↳ *We would like to have independent samples*
- *Variation of training labels $\rightarrow$ target network*
  - ↳ *Each time we update, we change the label in the training batch*
  - ↳ *This can severely deteriorate the convergence of algorithm*

DPG: ϵ-Greedy Improvement

To have sufficient exploration of environment: we can follow ϵ-greedy approach

+ *But how does it work here? You said we have continuous actions!*
- *Well! We can add continuous randomness to our policy*

Say we get $A_t \leftarrow \mu_{\boldsymbol{\theta}}(S_t)$ at time t: then we replace our action with

$$A_t \leftarrow A_t + \sqrt{\epsilon} Z_t$$

where Z_t is random noise with mean zero and variance one

> *Classical choice of Z_t is zero-mean unit-variance Gaussian variable, i.e.,*
>
> $$Z_t \sim \mathcal{N}(0, 1)$$

Note that the noise term $\sqrt{\epsilon} Z_t$ is then zero-mean with variance ϵ

DPG: ϵ-Greedy Improvement

+ *But what if after adding $\sqrt{\epsilon} Z_t$, the action gets out of its allowed range? For instance, we get $A_t = 5$ and $\sqrt{\epsilon} Z_t = 3$, but we should have all actions between 2 and 6*

– *That's a valid question! We usually clip the action in this case*

To avoid out-of-range actions, we replace apply ϵ-greedy approach as

$$A_t \leftarrow \text{Clip}\left(\mu_{\boldsymbol{\theta}}\left(S_t\right) + \sqrt{\epsilon} Z_t, a_{\min}, a_{\max}\right)$$

where $a_{\min}$ and $a_{\max}$ are minimum and maximum allowed actions and

$$\text{Clip}\left(x, a_{\min}, a_{\max}\right) = \begin{cases} a_{\min} & x < a_{\min} \\ x & a_{\min} \leqslant x \leqslant a_{\max} \\ a_{\max} & x > a_{\max} \end{cases}$$

DPG: *Replay Buffer*

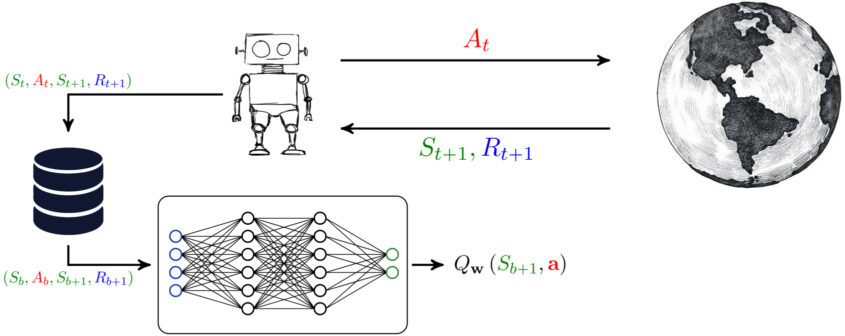
To reduce estimator's variance and enhance sample-efficiency, we can use experience replay as in DQL

- + *Wait a moment! But we talked about the fact that experience replay can increase estimator's variance in PGMs due to importance sampling argument! Now, we just ignore all those discussions?!*
- – *With stochastic policy yes! But here we have a deterministic policy*

Deterministic policy returns only one action

- *For each choice of $\boldsymbol{\theta}$, our policy chooses only one action*
  - ↳ *If we change $\boldsymbol{\theta}$ we only change this action*
- *Policy update does not change the probability of all actions*
  - ↳ *This is in fact why Q-learning does not suffer from high estimate variance*

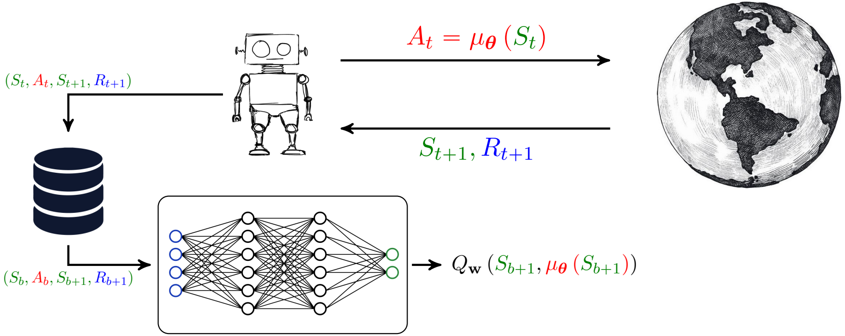
Recall: *Experience Replay in DQL*



We update DQN by randomly sampled mini-batches using TD error

$$\Delta_b \leftarrow R_{b+1} + \gamma \max_m Q_{\mathbf{w}}(S_{b+1}, a^m) - Q_{\mathbf{w}}(S_b, A_b)$$

DPG: *Experience Replay*



We now use the deterministic policy network to compute TD error

$$\Delta_b \leftarrow R_{b+1} + \gamma Q_{\mathbf{w}}\left(S_{b+1}, \mu_{\boldsymbol{\theta}}\left(S_{b+1}\right)\right) - Q_{\mathbf{w}}\left(S_b, A_b\right)$$

DPG: *Target Network*

The last thing to handle is to keep training dataset fixed for a while

- *After each mini-batch, we change both policy and Q-network*
  - ↳ *We update* $\mathbf{w}$ *and* $\boldsymbol{\theta}$
- *If we use the same networks to compute the estimate*

$$\hat{Q}_b = R_{b+1} + \gamma Q_{\mathbf{w}} \left(S_{b+1}, \mu_{\boldsymbol{\theta}} \left(S_{b+1} \right) \right)$$

then our next iteration runs over a different dataset

↳ *This can cause or training loop to diverge*

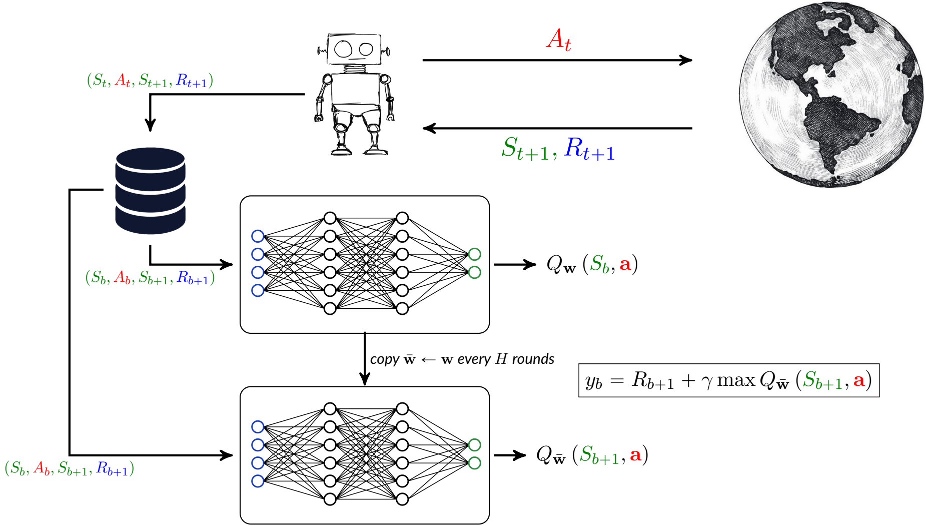
We have dealt with this in DQL using target network

Target Network in DPG

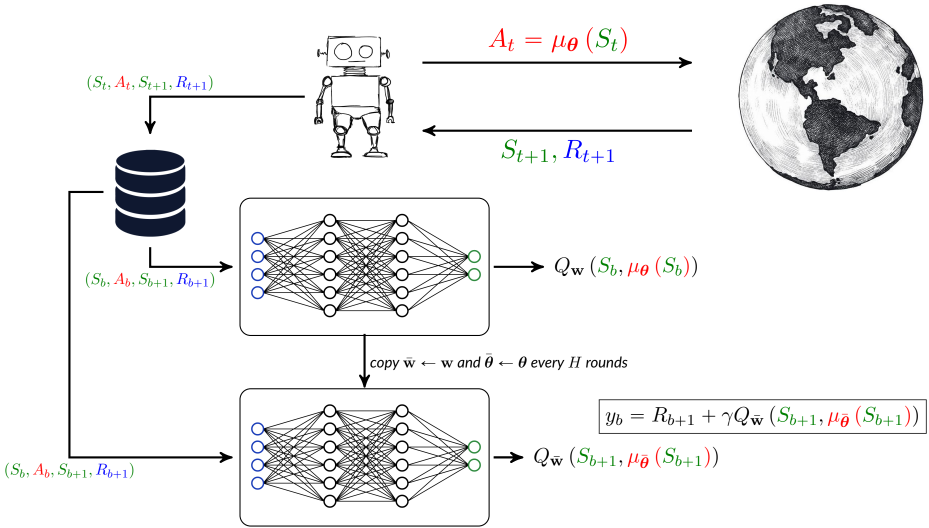
Copy DQN and policy network into the exactly same target networks

- *Use these target networks to compute the estimates*
- *Update them every multiple iterations by new copies of online networks*

Recall: *Target Network in DQL*



DPG: *Target Networks*



DDPG: *Deep DPG Algorithm*

DDPG():

Initiate with $\boldsymbol{\theta} = \bar{\boldsymbol{\theta}}$ *and* $\mathbf{w} = \bar{\mathbf{w}}$, *as well as factor* $\alpha < 1$ *and learning rate* β

Initiate state S_0 *and draw* $A_0 \leftarrow \mathrm{Clip}\left(\mu_{\boldsymbol{\theta}}\left(S_0\right) + \sqrt{\epsilon}\mathcal{N}\left(0,1\right)\right)$

while *interacting* ***do***

    Sample a time step $S_t, A_t \xrightarrow{R_{t+1}} S_{t+1}$ *and save in replay buffer*

    for *multiple iterations* ***do***

        Sample $S_b, A_b \xrightarrow{R_{b+1}} S_{b+1}$ *from replay buffer*

        Draw $A_{b+1} \leftarrow \mathrm{Clip}\left(\mu_{\bar{\boldsymbol{\theta}}}\left(S_{b+1}\right) + \sqrt{\epsilon}\mathcal{N}\left(0,1\right)\right)$

        Compute $\Delta = R_{b+1} + \gamma Q_{\bar{\mathbf{w}}}\left(S_{b+1}, A_{b+1}\right) - Q_{\mathbf{w}}\left(S_b, A_b\right)$

        Update value network as $\mathbf{w} \leftarrow \mathbf{w} + \beta \Delta \nabla Q_{\mathbf{w}}\left(S_t, A_t\right)$

        Update policy network as $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \frac{\partial}{\partial a} Q_{\mathbf{w}}\left(S_t, a\right)\big|_{a=A_t} \nabla \mu_{\boldsymbol{\theta}}\left(S_t\right)$

        if H *iterations passed* ***then***

            Copy $\bar{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}$ *and* $\bar{\mathbf{w}} \leftarrow \mathbf{w}$

        end if

    end for

end while

DDPG: *Overestimate Issue*

DDPG has been the key DPG algorithm and widely used for

- *Dealing with continuous action spaces*
- *Enabling off-policy learning with a deterministic version of PGM*

DDPG has shown a key issue; namely, overestimate of values

Value Overestimate of DDPG

Action-values estimated by DDPG can have significant biases

+ *We had it also in DQL and you said "it's not a big deal in general"! So, do we care about it here?!*
– *Well! Here is more important! Because, we use those estimates to update the policy and large biases would explode TD error!*
+ *So, shall we do double DQL then?!*
– *Pretty much yes!*

TD3: *Twin Delayed DDPG*

Value overestimate has been addressed in the extended version of DDPG

Twin Delayed DDPG $\equiv$ *TD3*

In TD3, we add three extra tricks to DDPG

1. *We use double DQN to suppress undesired bias*
   - ↳ *Remember that bias was mainly coming out of* $\max$ *operator*
   - ↳ *We came with a remedy called double Q-learning*
2. *We delay the policy update, i.e., update the policy less frequent*
   - ↳ *We update DQN after each mini-batch, but*
   - ↳ *We update policy network once every couple of mini-batches*
3. *We add extra noise to actions when we use them in target networks*
   - ↳ *This way we make the estimation error somehow independent*
   - ↳ *This leads to less bias*

From DPG to PGM

Both ideas of learning deterministic or stochastic policy have pros and cons

- *For deterministic policy we could say*
  - ↳ *We can efficiently use off-policy learning*
  - ↳ *We can get better sample efficiency*
  - ↳ *But we get less chance of being optimal*
    - ↳ *We do not search among possible random optimal policies*
- *For stochastic policy we could say*
  - ↳ *We search among much larger set of policies*
  - ↳ *We can converge to a better policy*
  - ↳ *But we have troubles with sample efficiency*
    - ↳ *We cannot easily learn off-policy due to limits of importance sampling*

+ *Is there any way to get good things of both worlds?*

– *Soft actor-critic approaches actually do this*

Recall: *Information Content and Entropy*

To understand the idea behind soft actor-critic, let's recap some definitions

Information Content

The information content of random variable $X \sim p(x)$ is

$$i(X) = \log \frac{1}{p(X)}$$

The information contents have some interesting properties

- *It's always non-negative, since $0 \leqslant p(x) \leqslant 1$*
- *The less likely outcome $X = x$ is, the more will be its information content*
  - ↳ *Think about it! You will find it very intuitive*

Recall: *Information Content and Entropy*

Entropy

For random variable $X \sim p(x)$, entropy is its average information content, i.e.,

$$H_p(X) = \mathbb{E}_p\left\{i(X)\right\} = \mathbb{E}_p\left\{\log \frac{1}{p(X)}\right\} = \int_x p(x) \log \frac{1}{p(x)}$$

Entropy quantifies how much confusion we have about X

- *If X is highly random, e.g., uniformly or Gaussian distributed,*
  - ↳ *Then $H_p(X)$ is very large*
- *If X is deterministic*
  - ↳ *Then $H_p(X) = 0$*

Redefining Value Function

After dealing with both deterministic and stochastic policies we might formulate the best policy as follows

- *It's globally deterministic*
  - ↳ *If one action gives better reward, it should go for it*
- *It's locally stochastic*
  - ↳ *Among actions with same rewards, it chooses one at random*

We could capture both these behaviors by looking into a new metric

> *Say we play with policy π: at time t, we are interested in*
>
> $$\tilde{R}_{t+1} = R_{t+1} + \xi H_\pi \left(A_t | S_t \right)$$
>
> *for some ξ, where $H_\pi \left(A_t | S_t \right)$ is entropy of action $A_t \sim \pi \left(\cdot | S_t \right)$*

Redefining Value Function

Say we play with policy π: at time t, we are interested in

$$\tilde{R}_{t+1} = R_{t+1} + \xi H_\pi \left(A_t | S_t\right)$$

for some ξ, where $H_\pi \left(A_t | S_t\right)$ is entropy of action $A_t \sim \pi\left(\cdot | S_t\right)$

This new modified reward incorporates both desires

- *Being globally deterministic*
  - ↳ *For actions with larger R_{t+1}, the modified $\tilde{R}_{t+1}$ is also larger*
- *Being locally stochastic*
  - ↳ *For actions with same R_{t+1}, policy with higher randomness has larger $\tilde{R}_{t+1}$*

Well! This might be a better reward!

SAC: *Soft Actor-Critic*

We can use either DPG or PGM to develop an actor-critic method for this new reward, i.e., we could define

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left\{ \sum_{i=0}^{\infty} \gamma^i \tilde{R}_{t+i+1} | S_t = s \right\} \\ &= \mathbb{E}_\pi \left\{ \sum_{i=0}^{\infty} \gamma^i \left[R_{t+i+1} + \xi H_\pi \left(A_{t+i} | S_{t+i} \right) \right] | S_t = s \right\} \end{aligned}$$

Interestingly, we end up in both cases with the same policy and value gradients!

The derived actor critic method is referred to as

Soft Actor-Critic $\equiv$ *SAC*

DRL Algorithms

Most DRL algorithms used in practice are actor-critic

We already discussed all main classes of actor-critic approaches

To each class, there are various extensions

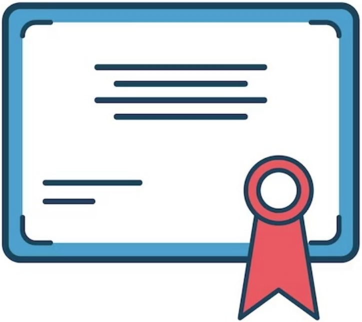
- *You are able now to follow all those extensions*
- *If necessary, you could come up with your own particular extension!*

A rule-of-thumb is

- *If you deal with discrete actions and have no concern on sample efficiency*
  - ↳ *Use stochastic-policy actor-critic approaches*
- *If you deal with continuous actions and/or need sample efficiency*
  - ↳ *Use DPG-like actor-critic approaches*

OpenAI: *Spinning Up in DRL*

Congratulations! You are now Deep RL experts!



Looking for some mini-projects for further practices? Take a look at OpenAI page