Reinforcement Learning (RL)

Chapter 10:

Deep Reinforcement Learning (DRL)

Policy Gradient algorithms

Saeed Saeedvand, Ph.D.

Contents

In this Chapter:

- ✓ Policy Gradient
- ✓ REINFORCE algorithm
- ✓ Vanilla Policy Gradient
- ✓ Different PG methods to update parameters
- ✓ Actor-Critic algorithms
- ✓ Proximal Policy Optimization (PPO)

Aim of this chapter:

- ✓ Understand the general concepts of the Policy Gradient algorithms and importance of use of them and Introducing main flow and concept of the Actor-Critic algorithms, and PPO.

Policy gradients (PG)

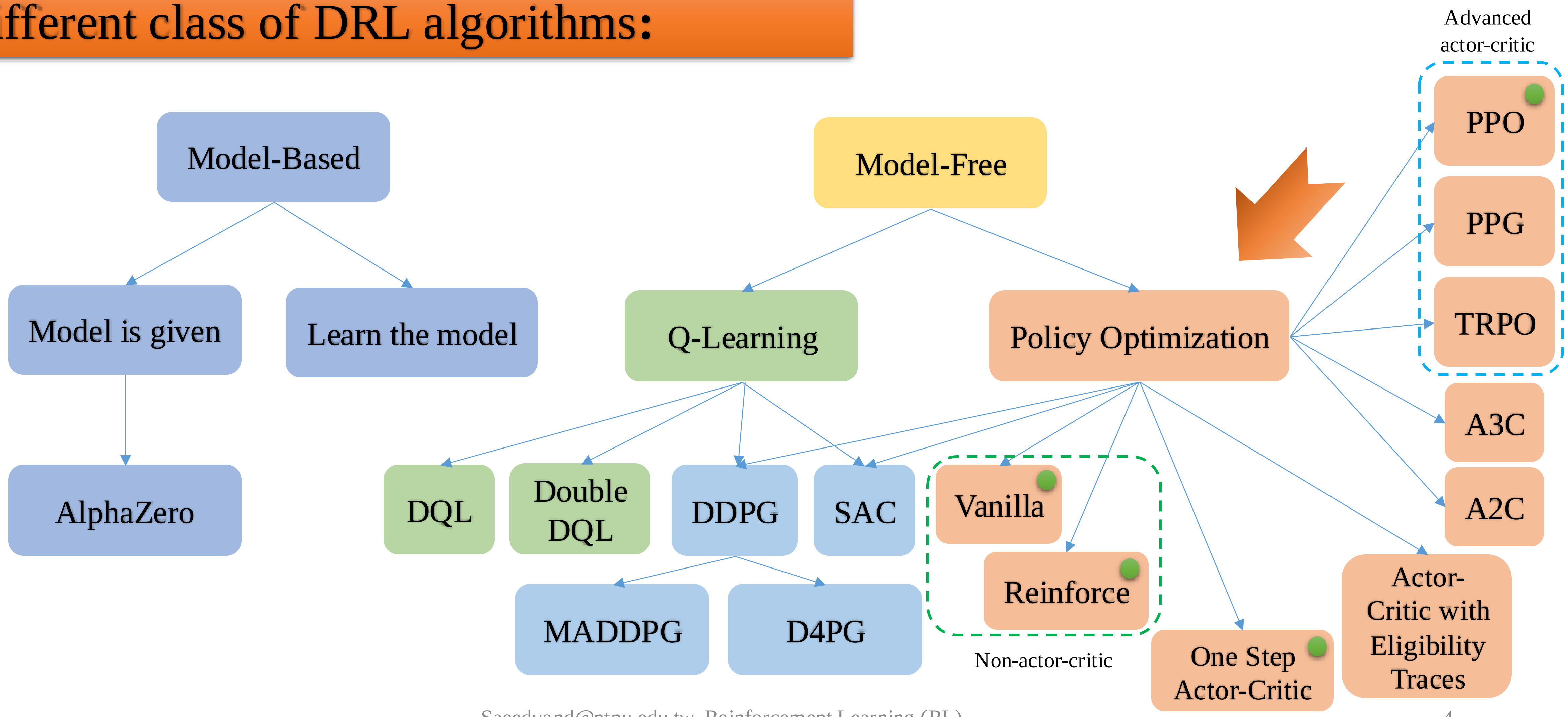
What is the Policy gradients

- ✓ In RL we want to find an optimal policy for the agent to get maximum rewards.
- ✓ Policy gradients (PG) is a range of algorithms for **solving RL problems**.
- ✓ **PG** family of algorithms **directly optimize the policy** instead of state-value or action-value.
- ✓ In PG we usually **model the policy** with a **parameterized function**.
- ✓ So we select actions without calculating a value function.

Important Note: A value function may still be **used to learn the policy parameter**, but is not required for action selection.

Deep Reinforcement Learning

Different class of DRL algorithms:



Policy Gradients (PG)

Value based approaches

- ✓ **Choose actions greedily** with respect to the value function and deterministic way (still are good for many applications).

Policy based approaches

- ✓ Learning a **probability distribution** over actions by getting observations and produce **stochastic policies.**

Policy Gradients (PG)

Policy based approaches

- ✓ **Ability** to work in continuous action spaces **without discretization**.
- ✓ Value-based approaches are **computationally very expensive if we want** to use them in the continuous space.
  - ✓ Because of **infinite number of actions and/or states** to **estimate the values and** getting **max** of next actin is very expensive**.**

$$\Delta w = \alpha[R(s_t, a_t) + \gamma \boxed{\mathrm{Max}_a[\hat{Q}(s_{t+1}, a_{t+1}; w)]} - \hat{Q}(s_t, a_t; w)\nabla_w \hat{Q}(s_t, a_t; w)]$$

Policy Gradients (PG)

- ✓We consider the **parameterized policy $\boldsymbol{\pi_\theta}$** for stochastic policy.
  - ✓ Usually ANN; but Logistic Regression, and … can be used too.
- ✓So, the objective function is to **maximize the expected return:**

$$max_\theta \ \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)\right]$$

It is state-action sequence in a **complete trajectory** representing T steps.

Policy Gradients (PG)

$$\boxed{max_{\theta}}\ \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)\right]$$

- ✓We aim to find the proper or max of parameters θ (e.g. ANN) that maximizes the expected reward (**finding best policy**).

- ✓**Expected value** can be computed by trajectories generated with **policy** $\boldsymbol{\pi_\theta}$.

How to maximizes the expected reward in PG

Gradient Ascent algorithm

- **Gradient Ascent** is similar **Gradient Decent** which instead of minimization performs maximizing steps.
- **If we can** calculate the **gradient** regarding the parameters we have:

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)]$$

Then we can update the parameters θ of network in the direction of the gradient!

$$\theta = \theta + \alpha \nabla_\theta \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)]$$

All algorithms that follow this called **Policy Gradient** techniques

How to calculate gradient from policy? Next slides

How to maximizes the expected reward in PG

How to calculate gradient from policy?

- ✓ **Calculating gradient from policy** is challenging because:
  1. Depends on the **selected actions** directly determined policy π_θ.
  2. Depends on the **state distribution** implicitly determined policy π_θ (**transition probability**).

- ✓ **Finding state distribution** for the policy update **is hard** because **environment** is generally **unknown**.
- ✓ Therefore, we need multiple mathematical steps including finding **probability of a trajectory**.
- ✓ **With PG theorem we** reform **derivatives of the objective function** and remove the need for state distribution (simplify the gradient computation).

Policy Gradients (PG) algorithms

REINFORCE algorithm

✓ Using Monte Carlo sampling of trajectories we can **compute the Policy Gradient**:

Log idea is to prevent exponentiation and make calculations simpler.

$$\nabla_\theta \mathbb{E}[G_t | S_t = s_0] = \sum_{t=0}^{T-1} G_t \nabla_\theta log\pi_\theta(a_t | s_t)$$

Monte Carlo sampling

$$G_t = \sum_{t'=t+1}^{T} \gamma^{k-t-1} R(s_{t'}, a_{t'})$$

Probability that action a is taken at:
state s_t and parameter θ of network at time t
comes from network by e.g. softmax

Proof of simplification:
Chapter 13, reinforcement learning, Sutton book

Policy Gradients (PG) algorithms

REINFORCE Policy Gradient algorithm

How to Update parameters θ?

$$\nabla_\theta \mathbb{E}[G_t|S_t = s_0] = \sum_{t=0}^{T-1} G_t \nabla_\theta log\pi_\theta(a_t|s_t)$$

$$\theta_{t+1} = \theta_t + \alpha \sum_{t=0}^{T-1} G_t \nabla_\theta log\pi_\theta(a_t|s_t)$$

REINFORCE Policy Gradient algorithm

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for $\pi(a|s,\theta)$

Input: a differentiable policy parameterization
Algorithm parameter: step size $\alpha > 0$
Initialize: policy parameter θ (e.g. 0)

Loop forever (for each episode):
Generate an episode $S_0, A_0, R_0, \dots, S_{T-1}, A_{T-1}, R_T$

Loop for each step of the episode $t = 0, 1, \dots, T-1$

$$G_t = \sum_{t'=t+1}^{T} \gamma^{k-t-1} R(s_{t'}, a_{t'})$$

$$\theta_{t+1} = \theta_t + \alpha\gamma^t G_t \nabla_\theta log\pi_\theta(a_t|s_t, \theta)$$

REINFORCE is on-policy

Since its training samples are collected by target policy

Policy Gradients (PG) algorithms

How to calculate gradient from policy to maximize reward?

Vanilla Policy Gradient algorithm

Total reward **after action $\boldsymbol{a_t}$** of the trajectory

Trajectories

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} R(\tau) = \boxed{\mathbb{E}_{\tau\sim\pi_\theta}} [\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} \gamma^{t'-t} R(s_{t'}, a_{t'})\ ]$$

Expectation can be written as average of trajectory size: $\frac{1}{|\tau|}$ over multiple episodes

Log idea is to prevent exponentiation and make calculations simpler

Probability that action a is taken at: state s_t and parameter θ of network at time t

Policy Gradients (PG) algorithms

Vanilla Policy Gradient algorithm

How to Update parameters θ?

Vanilla is also on-policy

$$\nabla_\theta \mathbb{E}_{\pi_\theta} R(\tau) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)\gamma^t \sum_{t'=t}^{T-1} \gamma^{t'-t} R(s_{t'}, a_{t'})]$$

$$\theta_{t+1} = \theta_t + \alpha \frac{1}{|\tau|}[\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} \gamma^{t'-t} R(s_{t'}, a_{t'})]$$

Policy Gradients (PG) algorithms

Different PG methods to update parameters

$$\nabla_\theta \mathbb{E}[G_t|S_t = s_0] = \sum_{t=0}^{T-1} \sum_{t'=t+1}^{T} \gamma^{k-t-1} R(s_{t'}, a_{t'}) \nabla_\theta log\pi_\theta(a_t|s_t)$$

Total reward of a_t with the trajectory sampeling

Reinforce

$$\nabla_\theta \mathbb{E}_{\pi_\theta} R(\tau) = \frac{1}{|\tau|} [\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} \gamma^{t-1} R(s_t, a_t)]$$

Total reward of the trajectory (averaged)

Vanilla

$$\nabla_\theta \mathbb{E}[G_t|S_t = s_0] = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t) \gamma^t Q(s_t, a_t) - V(s_t)]$$

Advantage function $A(s_t, a_t)$

A common variation of REINFORCE is to **subtract a bias or baseline** value $(Q(s_t, a_t) - V(s_t))$
It **reduces the variance of gradient** estimation by keeping bias unchanged.

PG Methods

Different PG methods to update parameters

$$\nabla_\theta \mathbb{E}[G_t|S_t = s_0] = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)\gamma^t\, R(s_t, a_t) + V(s_{t+1}) - V(s_t)]$$

Temporal deference's residual

$$\nabla_\theta \mathbb{E}[G_t|S_t = s_0] = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)\gamma^t\, Q(s_t, a_t)]$$

State-action function

Policy Gradients (PG)

- ✓ We should be able to **run the policy in the environment to collect** the trajectory dataset.
- ✓ We should represent our policy in a way that **allows us to calculate gradient of policy.**
- ✓ So with update step we can compute the policy gradient

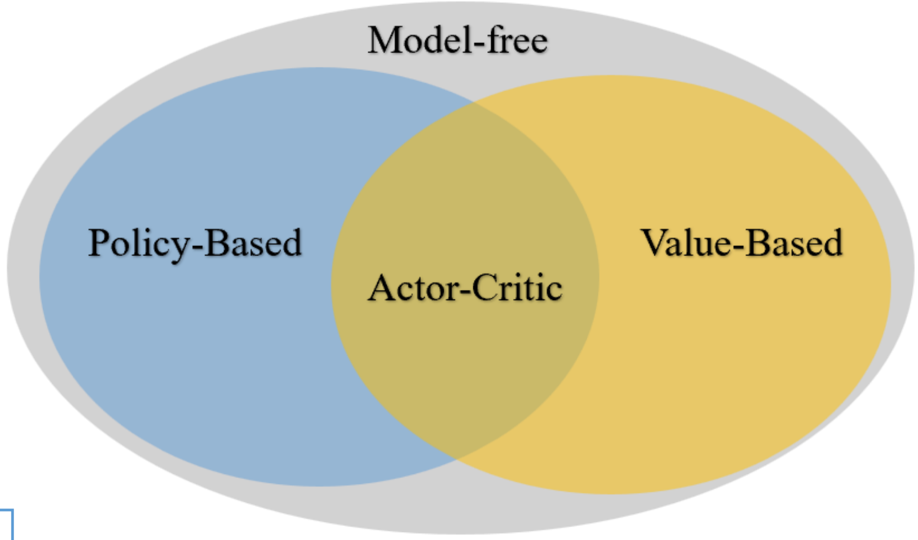
Note: PG algorithms produces probabilities of actions
For example usually using of softmax layer at the end.

Policy Gradients (PG)

Actor-Critic algorithms

- ✓ Using the **value function** beside **policy gradient** can help the policy update.
- ✓ Methods that learn approximations to both **policy** **and** **value functions** **together** are called **Actor-Critic** methods.

- ✓ **There are different versions:**
  - One-step Actor–Critic (TD(0))
  - Actor–Critic with Eligibility Traces
  - Actor–Critic with Eligibility Traces (continuing)
  - …



> **Note:** REINFORCE-with-baseline method **learns both a policy and a state-value function**, but it is not an actor–critic method because it use state-value function as baseline only not as a critic.

Policy Gradients (PG)

Actor-Critic Networks

- **Actor** is a reference to the updating the policy parameters θ of $\boldsymbol{\pi}_{\theta}(\boldsymbol{s}, \boldsymbol{a})$.

- **Critic** refers to the updating value function parameters w and can be one of state-value $\hat{V}(s, w)$ or **action-value** $\hat{Q}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{w})$ functions.

One Step Actor-Critic (Online)

Input: a differentiable policy parameterization $\pi(a|s,\theta)$
Input: a differentiable state-value function parameterization $\pi(a|s,w)$
Parameters: step sizes $\alpha^{\theta} > 0$ and $\alpha^{w} > 0$
Initialize: Policy parameter weights $\theta \; \in \mathbb{R}$
Initialize: State-value weights w ∈ $\mathbb{R}$
Loop forever (for each episode):

Initialize s (first state of episode)

$I = 1$

Loop while S is not terminal (for each time step):

$a \sim \pi(.|s,\theta)$

Take action a, observe s' , and R

$\delta = R + \gamma\hat{V}(s',w) - \hat{V}(s,w)$ if (s' is terminal) $\hat{V}(s',w) = 0$

$w = w + \alpha^{w} I \, \delta \, \nabla \, \hat{V}(s,w)$

$\theta = \theta + \alpha^{\theta} I \, \delta \, \nabla \log \pi(a|s,\theta)$

$I = \gamma I$

$s = s'$

Calculation of TD error for state-value using **critic network**

Update weighs of the **critic network**

Update weighs of the **actor network (policy parameters)**
It is relying on gradient state-value from critic network

Policy Gradients (PG) algorithms

Proximal Policy Optimization (PPO)

- ✓ **Tries to Avoid** parameter updates that resuls in **big change** of the **policy at one step.**
- ✓ **Adding constraint** on the **size of policy update** at each iteration.
- ✓ Trust region policy optimization (TRPO) does the same but complex to implement.
- ✓ PPO is using a clipped surrogate objective with **similar performance** to TRPO.
- ✓ There are different versions of PPO with success and failures in different scenarios (We look at general concept here).

Schulman, John, et al. "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347* (2017).

Policy Gradients (PG) algorithms

Proximal Policy Optimization (PPO)

- ✓ We can compute probability ratio between old and new policies as:

$$r(\theta) = \frac{\pi_{\theta_{new}}(a|s)}{\pi_{\theta_{old}}(a|s)}$$

- ✓ Then, we can state the objective function (surrogate objective):

$$J(\theta) = \mathbb{E}[r(\theta)A_{\theta_{old}}(s,a)]$$

advantage function

- ✓ With this extremely large parameter updates and big policy ratio change can happen which brings instability.

Policy Gradients (PG) algorithms

Proximal Policy Optimization (PPO)

- ✓ In PPO we add a constraint hyperparameter to limit updates to a small interval $[1-\varepsilon, 1+\varepsilon]$, so we have:

Advantage estimates

$$L_{clip}(\theta) = \mathbb{E}[\min(r(\theta)A_{\theta_{old}}(s,a), (clip(r(\theta), 1-\varepsilon, 1+\varepsilon)A_{\theta_{old}}(s,a))]$$

The minimum one between the original value and the clipped one

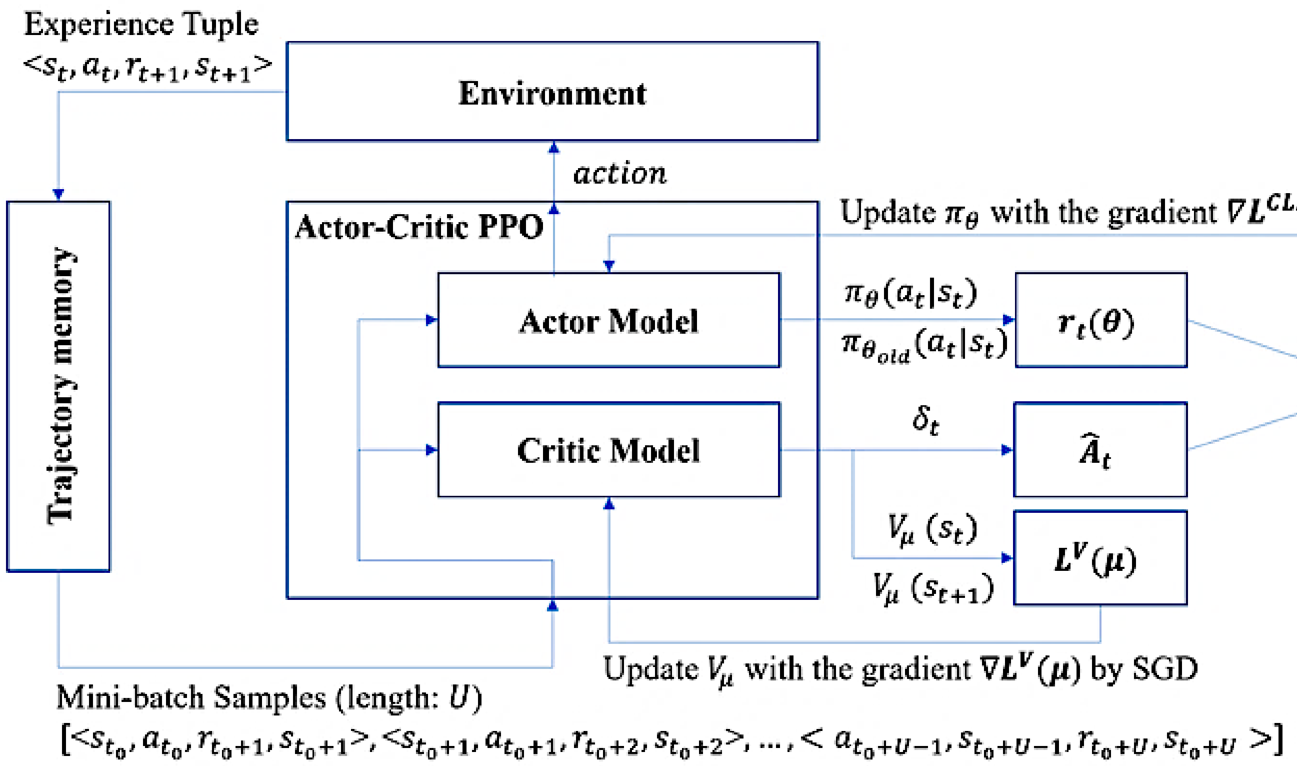
Clips between $[1-\varepsilon, 1+\varepsilon]$

- ✓ Here an extreme increasing of the policy update due to high rewards will be limited.

Policy Gradients (PG) algorithms

Proximal Policy Optimization (PPO)

✓ PPO is sharing parameters for both policy (actor) and value (critic) networks



Lim, Hyun-Kyo, et al. "Federated reinforcement learning for training control policies on multiple IoT devices." *Sensors* 20.5 (2020): 1359.

Policy Gradients (PG) algorithms

Proximal Policy Optimization (PPO)

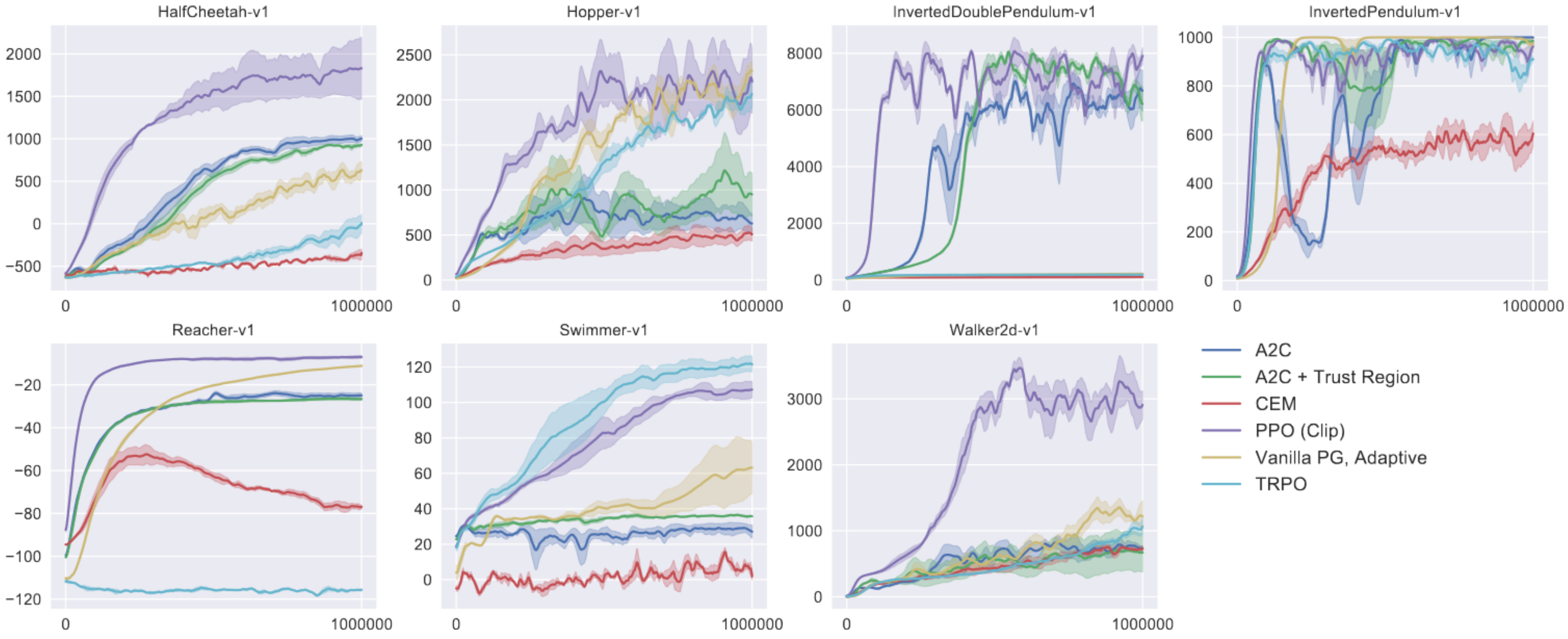
Algorithm 1 PPO, Actor-Critic Style

for iteration$=1,2,\ldots$ **do**
  for actor$=1,2,\ldots,N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1,\ldots,\hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Policy Gradients (PG) algorithms

Proximal Policy Optimization (PPO)

✓ Training for one million timesteps on different MuJoCo environments.



Policy Gradients (PG) algorithms

Policy Parameterization for Continuous Actions

- ✓ Policy-based methods are a **practical way** of dealing with **large actions spaces**
- ✓ Continuous spaces with an infinite number of actions also are good options for PG algorithms.
- ✓ **Instead** of computing probabilities for each action, network can learn **statistics of the probability distribution**.

Policy Gradients (PG) algorithms

Proximal Policy Optimization (PPO)

✓ Design choices for PPO policy parameterization.

Gaussian	$\pi_\theta(a\|s) := \mathcal{N}\left(\mu_\theta(s), \sigma_\theta^2(s)\right)$
Beta	$\pi_\theta(a\|s) := f\left(\frac{a-l}{r-l}, \alpha_\theta(s), \beta_\theta(s)\right)$ with $f(x,\alpha,\beta) := \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x^{\beta-1})$
Softmax	$\pi_\theta(a\|s) := \frac{1}{c_s} e^{\phi_\theta(s,a)}$ with $c_s = \sum_{a'\in\mathcal{A}} e^{\phi_\theta(s,a')}$

Hsu, Chloe Ching-Yun, Celestine Mendler-Dünner, and Moritz Hardt. "Revisiting design choices in proximal policy optimization." arXiv preprint arXiv:2009.10897 (2020).

Continuous Actions (Review note:)

Normal Distribution:

- ✓ During any measurement **values will follow a normal distribution.**
- ✓ We can use it to **describe physical events.**
- ✓ Also, known as **Gaussian Distribution** or **bell curve.**

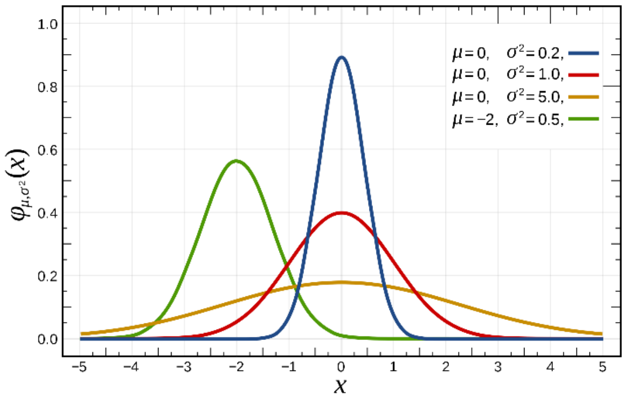
Density of the probability

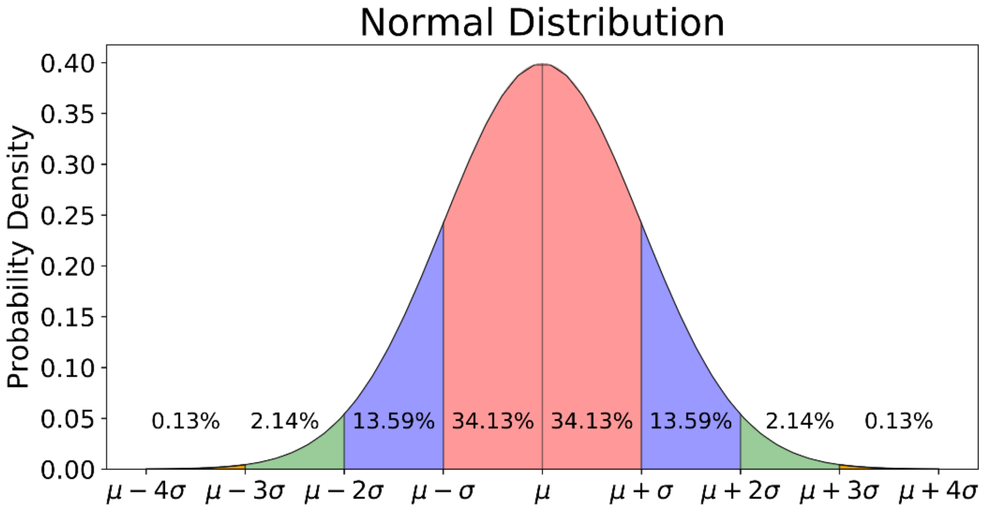
mean

$$p(x) \doteq \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Standard deviation

Just the number 3.14159

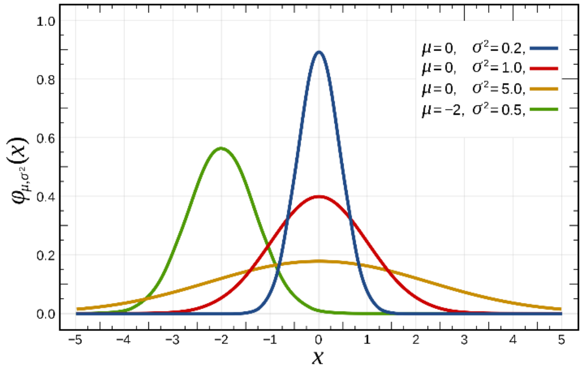




Continuous Actions

Defining Continues action space:

✓ For normal distribution If we write **probability density function (PDF)** as:



$$p(x) \doteq \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

✓ For **policy parameterization**, the **policy** can be defined as the **normal probability density** over a real-valued scalar actions:

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

Parameterized function approximators $\theta = [\theta_\mu, \theta_\sigma]^T$

So, **Mean** and **Standard** deviation are parametric function approximators representing action space

Summery

- ✓ We discussed Policy Gradient to directly improve policy
- ✓ We saw REINFORCE algorithm and Vanilla Policy Gradient as two PG algorithms.
- ✓ We discussed how to use benefits of both policy and value function together in Actor-Critic algorithms.
- ✓ Concept of Continues action Space.
- ✓ How Proximal Policy Optimization (PPO) is working.