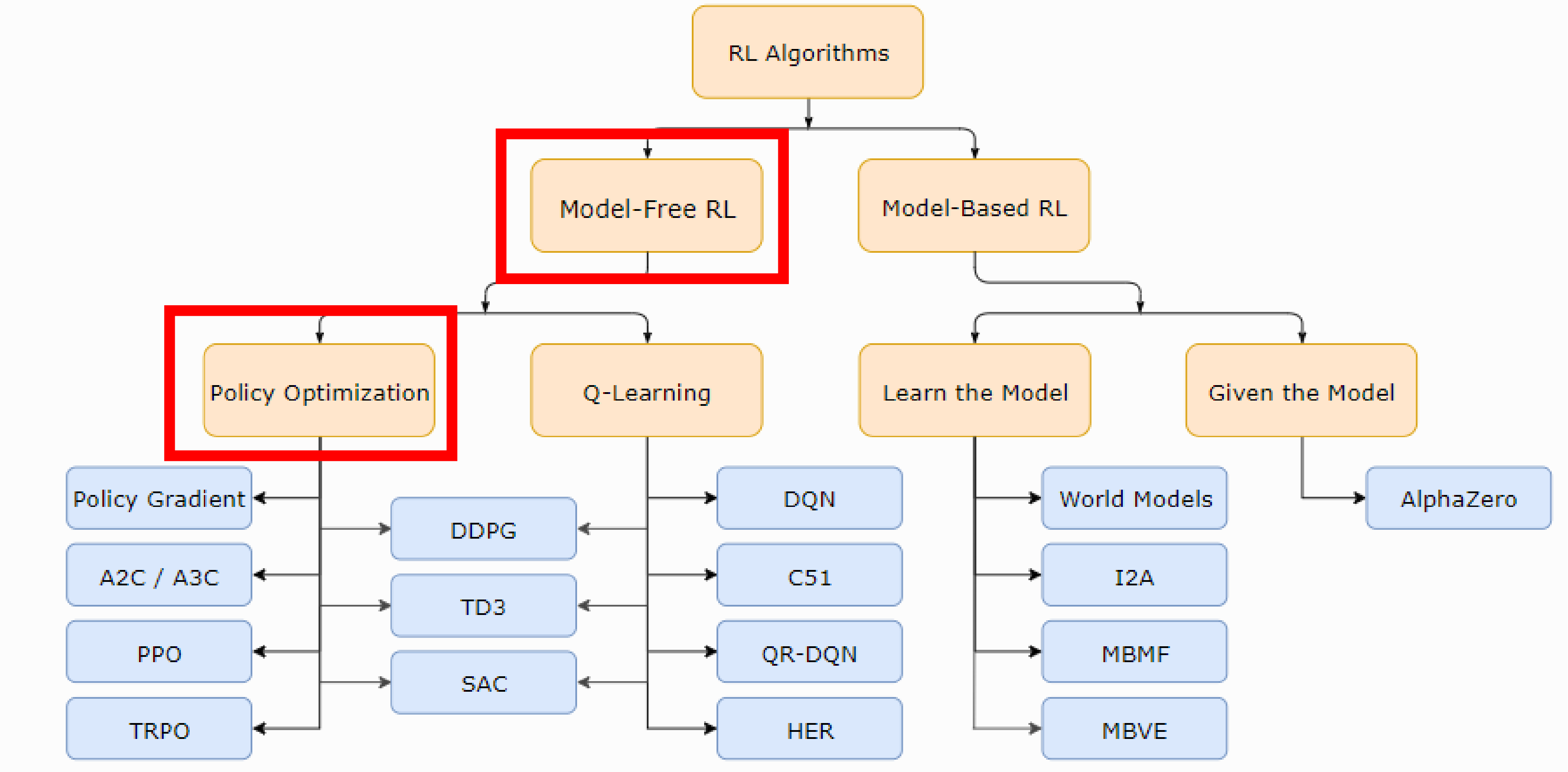
Policy optimization based RL

Model free RL algorithms include Policy Optimization and Q-Learning.

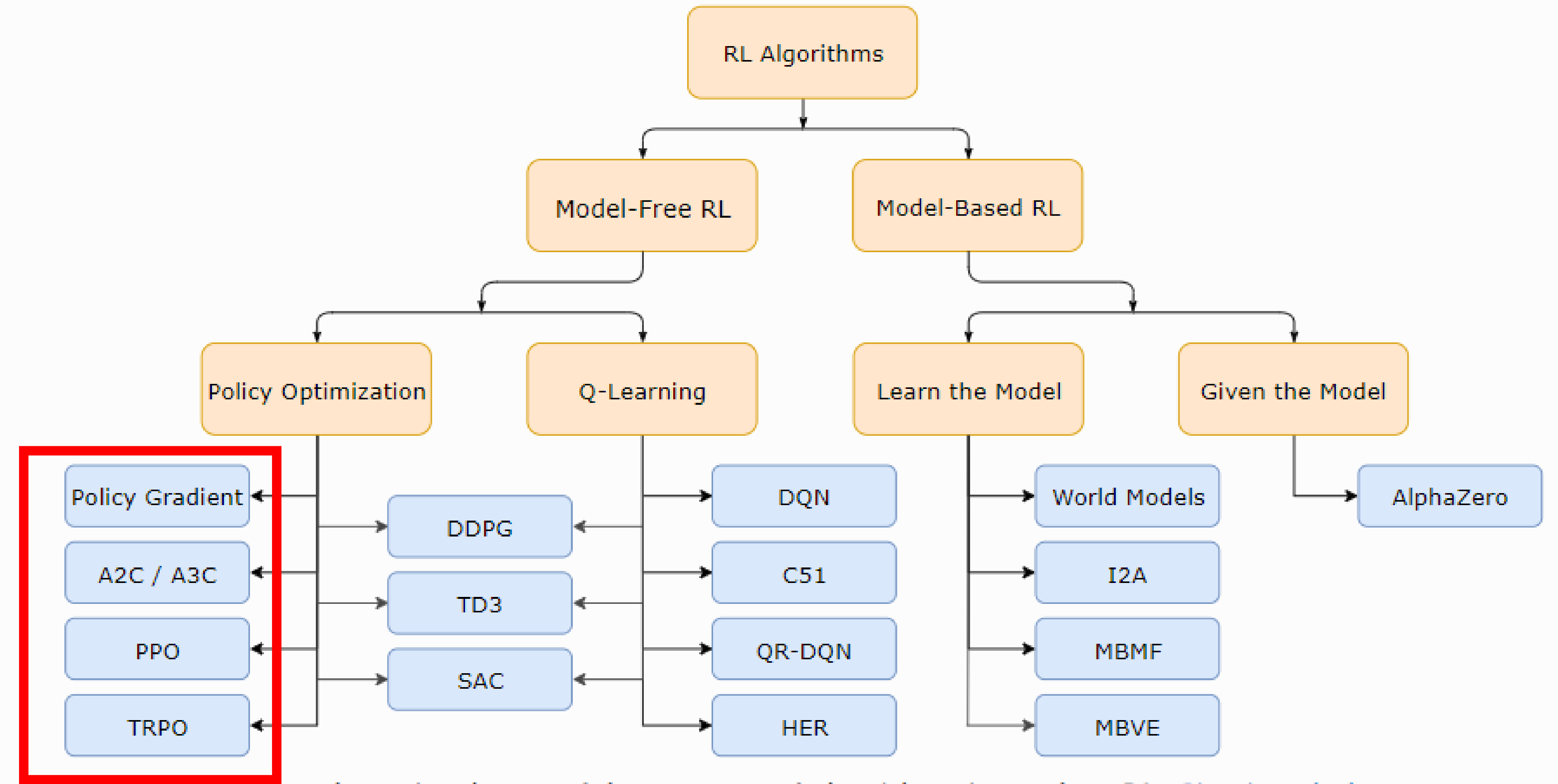


A non-exhaustive, but useful taxonomy of algorithms in modern RL. Citations below.

[Welcome to Spinning Up in Deep RL! — Spinning Up documentation (openai.com)]

Policy optimization based RL

Popular policy optimization based RL algorithms include PG, A2C/A3C, TRPO, and PPO.



A non-exhaustive, but useful taxonomy of algorithms in modern RL. Citations below.

Recap – Key concepts

Policies	$a_t \sim \pi_\theta(s_t)$
Trajectories	$\tau = (s_0, a_0, s_1, a_1, \dots)$
Reward	$r_t = R(s_t, a_t, s_{t+1})$
Return	$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$

$$P(\tau|\pi) = \rho_0(s_0) \cdot \pi(a_0|s_0) \cdot P(s_1|s_0, a_0) \cdot \pi(a_1|s_1) \cdot P(s_2|s_1, a_1) \cdot \cdots$$

$$J(\pi) = E_{\tau\sim\pi}\,(R(\tau))$$

$$V^\pi(s) = E_{\tau\sim\pi}\,(R(\tau)|s_0 = s)$$

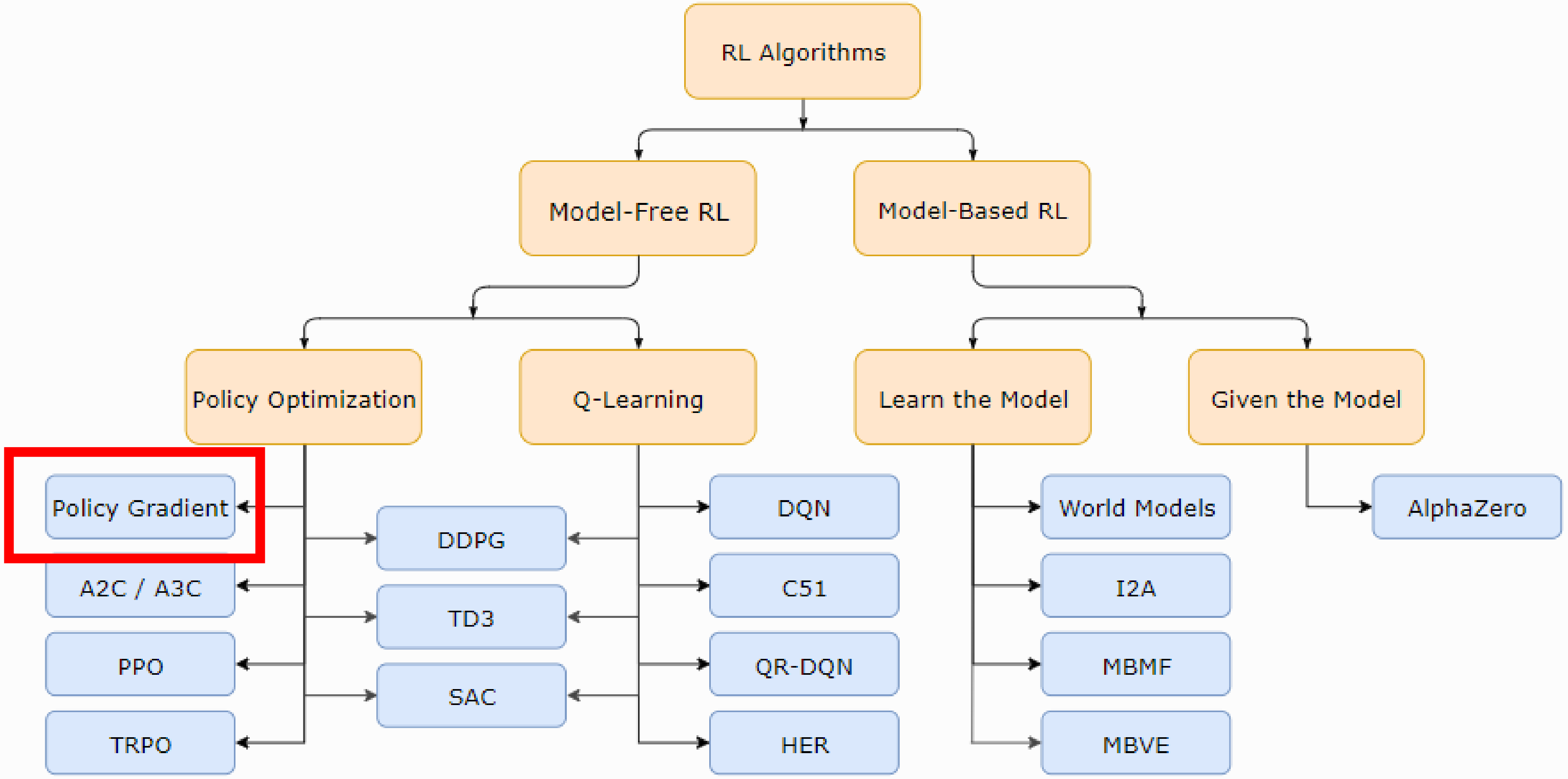
$$Q^\pi(s,a) = E_{\tau\sim\pi}\,(R(\tau)|s_0 = s, a_0 = a)$$

$$V^\pi(s) = E_{\substack{\tau\sim\pi \\ s'\sim P}}[r(s,a) + \gamma V^\pi(s')]$$

$$Q^\pi(s,a) = E_{s'\sim P}\Big[r(s,a) + \gamma E_{a'\sim\pi}[Q^\pi(s',a')]\Big]$$

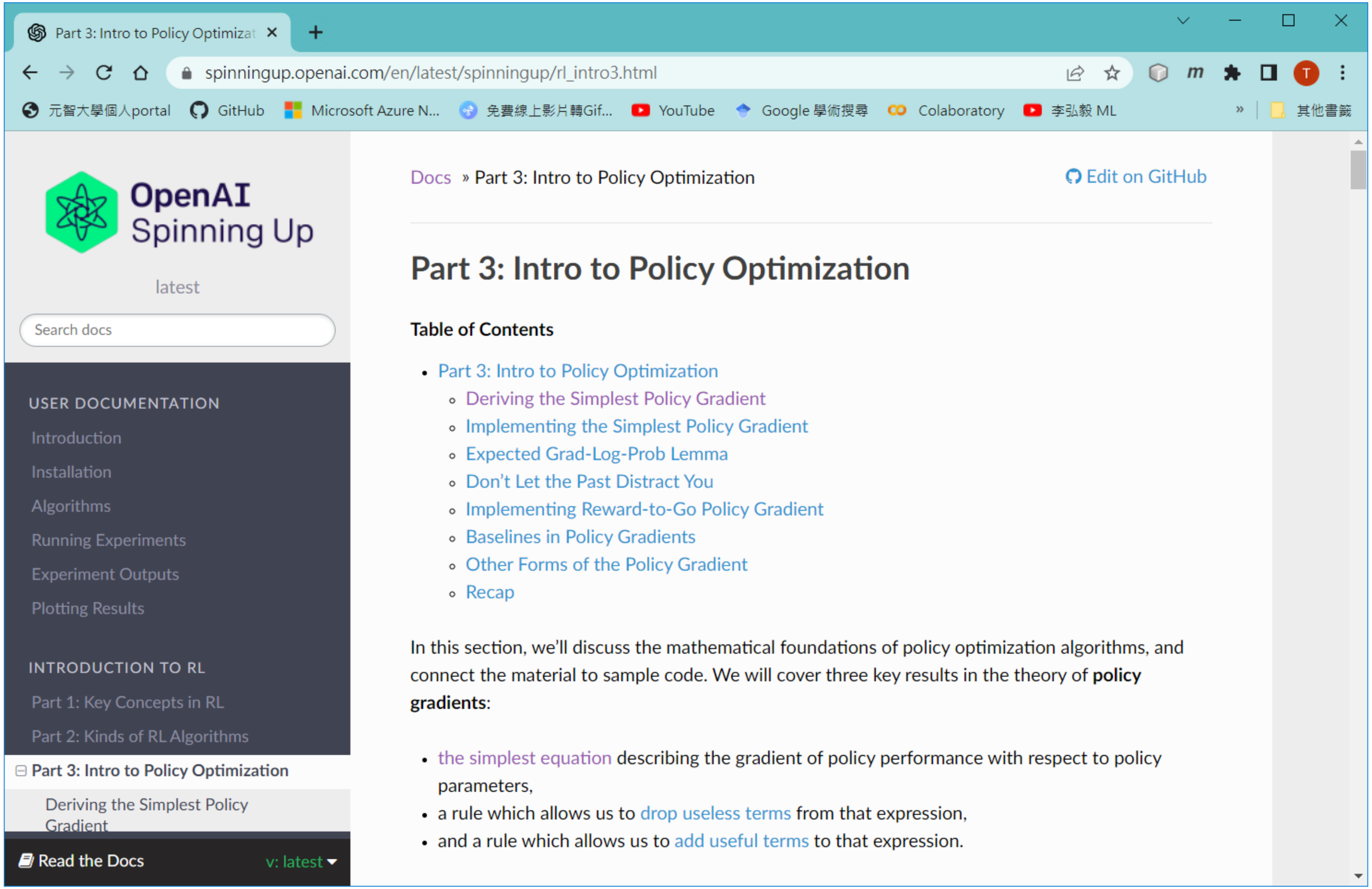
$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Policy gradient



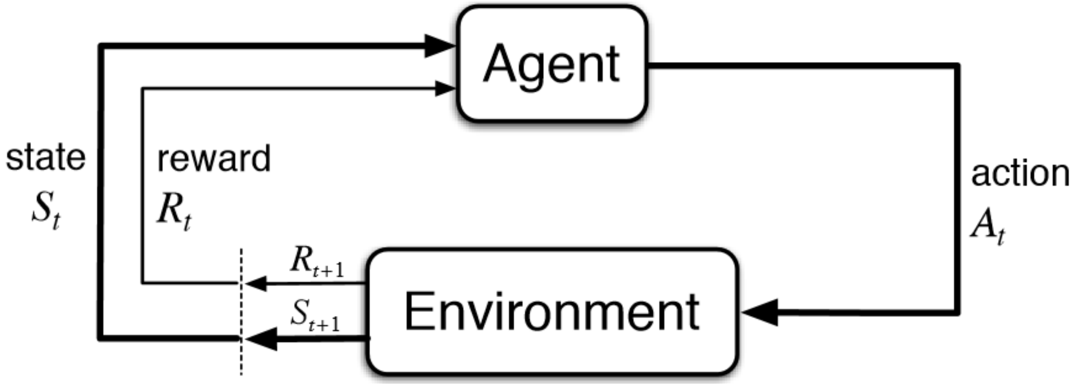
A non-exhaustive, but useful taxonomy of algorithms in modern RL. Citations below.

Policy gradient

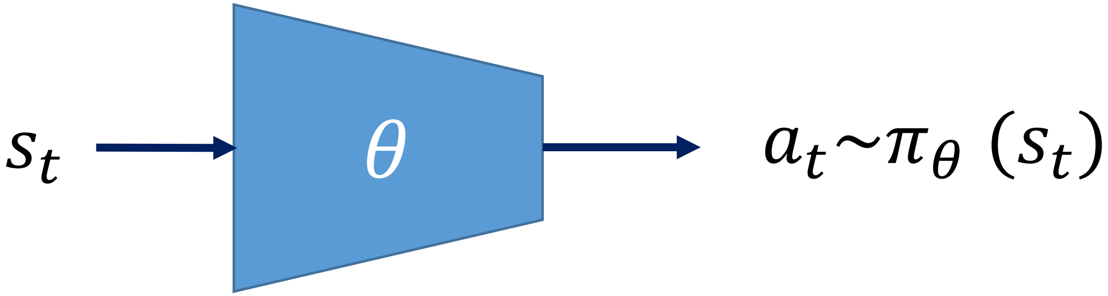


How to train NN to learn the best policy?

The NN should learn to generate actions that maximize cumulative rewards.



$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)$$



$$\max\ J(\pi) = E_{\tau \sim \pi}\ (R(\tau))$$

$$\tau = (s_0, a_0, s_1, a_1, \dots)$$

$$P(\tau|\pi) = \rho_0(s_0) \cdot \pi(a_0|s_0) \cdot P(s_1|s_0, a_0) \cdot \pi(a_1|s_1) \cdot P(s_2|s_1, a_1) \cdot \cdots$$

How to train NN with maximize cumulative reward?

To find π_θ that has maximum $J(\pi)$, we need $\nabla J(\pi)$

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)$$

$$\nabla J(\pi) = \nabla E_{\tau \sim \pi}\left(R(\tau)\right)$$

$$= \nabla \sum_{\tau \sim \pi} P(\tau|\pi) R(\tau)$$

$$= \sum_{\tau \sim \pi} \nabla P(\tau|\pi) R(\tau)$$

$$= \sum_{\tau \sim \pi} P(\tau|\pi) \nabla log P(\tau|\pi) R(\tau) \qquad \nabla f = f \nabla log f$$

How to train NN with maximize cumulative reward?

$$\nabla J(\pi) = \sum_{\tau \sim \pi} P(\tau|\pi) \nabla log P(\tau|\pi) R(\tau)$$

$$= \sum_{\tau \sim \pi} P(\tau|\pi) [\nabla_\theta log\pi(a_0|s_0) + \nabla_\theta log\pi(a_1|s_1) + \cdots \nabla_\theta log\pi(a_T|s_T)] R(\tau)$$

$$= \sum_{\tau \sim \pi} P(\tau|\pi) \left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t) \right] R(\tau)$$

$$= E_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t) R(\tau) \right]$$

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$$

$$P(\tau|\pi) = \rho_0(s_0) \cdot \pi(a_0|s_0) \cdot P(s_1|s_0, a_0) \cdot \pi(a_1|s_1) \cdot P(s_2|s_1, a_1) \cdot \cdots \pi(a_T|s_T) \cdot P(s_{T+1}|s_T, a_T)$$

$$log P(\tau|\pi) = log \rho_0(s_0) + log\pi(a_0|s_0) + log P(s_1|s_0, a_0) + log\pi(a_1|s_1) + \cdots$$

$$\nabla_\theta log P(\tau|\pi) = \nabla_\theta log\pi(a_0|s_0) + \nabla_\theta log\pi(a_1|s_1) + \cdots$$

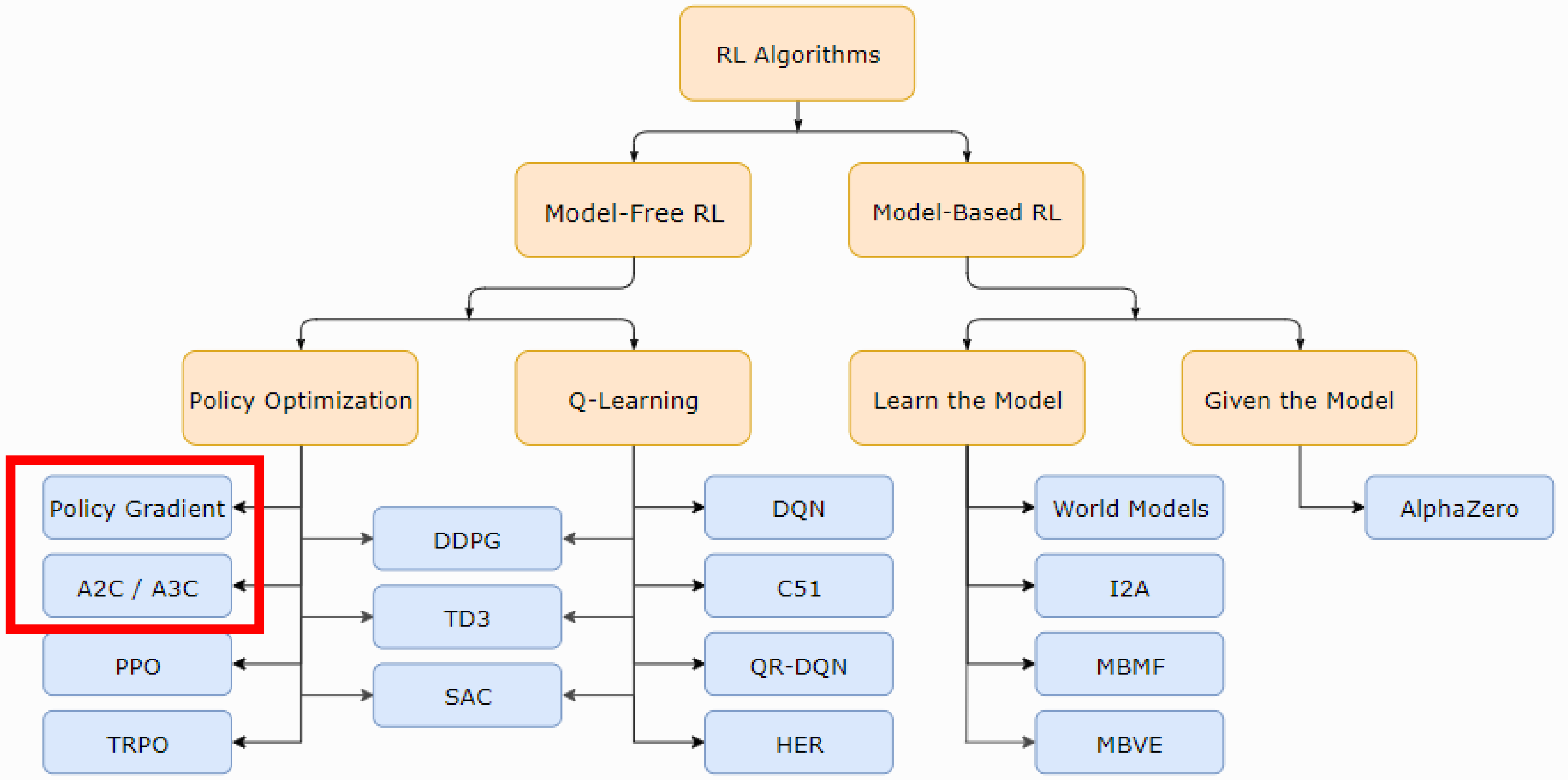
Problem with policy gradients

$$\nabla J(\pi) = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)R(\tau)\right]$$

- Monte Carlo updates (i.e. taking random samples) will introduce high variability in log probabilities and cumulative reward values. This will cause unstable learning and/or the policy distribution skewing to a non-optimal direction
- Another problem occurs if we have trajectories whose cumulative reward is 0.
- These issues contribute to the instability and slow convergence of vanilla policy gradient methods.

https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Reducing sampling variance with actor-critic



A non-exhaustive, but useful taxonomy of algorithms in modern RL. Citations below.

Use expected value to reduce sampling variance

$$\nabla J(\pi) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta log\pi_\theta(a_t|s_t)R(\tau)\right]$$

REINFORCE (Monte Carlo PG)

$$= E_{\tau\sim\pi_\theta}\ [\textstyle\sum_{t=0}^{T}\nabla_\theta log\pi_\theta(a_t|s_t)Q^\pi(s,a)]$$

Q Actor-Critic

$$= E_{\tau\sim\pi_\theta}\ [\textstyle\sum_{t=0}^{T}\nabla_\theta log\pi_\theta(a_t|s_t)A^\pi(s,a)]$$

Advantage Actor-Critic

$$= E_{\tau\sim\pi_\theta}\ [\textstyle\sum_{t=0}^{T}\nabla_\theta log\pi_\theta(a_t|s_t)\delta]$$

TD Actor-Critic

https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Q actor-critic

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)\Phi_t\right] \qquad \Phi_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} \nabla \log p_\theta(a_t^n|s_t^n)\left(\sum\nolimits_{t'}^{T_n} \gamma^{t'-t} r_{t'}^n\right)$$

$$G_t^n = \sum\nolimits_{t'}^{T_n} \gamma^{t'-t} r_{t'}^n$$

unstable when sampling amount is not large enough

$$E[G_t^n] = Q^{\pi_\theta}\,(s_t^n, a_t^n)$$

By definition, the expected value of G_t^n is Q

Ref: 李弘毅 https://youtu.be/j82QLgfhFiY

Reducing sampling variance with a baseline

$$\nabla J(\pi) = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)\,(R(\tau) - b(s_t))\right]$$

- Making the cumulative reward smaller by subtracting it with a baseline will make smaller gradients, and thus smaller and more stable updates..

https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Reducing sampling variance with a baseline

$$\nabla J(\pi) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)R(\tau)\right]$$

$$= E_{\tau\sim\pi_\theta}\ [\textstyle\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)\ (R(\tau) - b(s_t))\ ]$$

$$\begin{aligned} 0 &= \nabla_\theta \int_x P_\theta(x) \\ &= \int_x \nabla_\theta P_\theta(x) \\ &= \int_x P_\theta(x) \nabla_\theta \log P_\theta(x) \\ \therefore 0 &= \underset{x\sim P_\theta}{\mathrm{E}}\left[\nabla_\theta \log P_\theta(x)\right]. \end{aligned}$$

$$E_{a_t\sim\pi_\theta}\ [\nabla_\theta log\pi_\theta(a_t|s_t)b(s_t)] = 0$$

https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html

Advantage actor-critic

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)\Phi_t\right] \qquad \Phi_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} - b(s_t)$$

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} \nabla \mathrm{log} p_\theta(a_t^n|s_t^n)\left(\boxed{\sum\nolimits_{t'}^{T_n} \gamma^{t'-t} r_{t'}^n} - \boxed{b(s_t^n)}\right)$$

$$E[G_t^n] = Q^{\pi_\theta}(s_t^n, a_t^n) \qquad E[b(s_t^n)] = V^{\pi_\theta}(s_t^n)$$

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)\left(Q^\pi(s_t,a_t) - V^\pi(s_t)\right)\right]$$

how much better it is to take a specific action compared to the average, general action at the given state.

Advantage actor-critic

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)\big(Q^\pi(s_t,a_t) - V^\pi(s_t)\big)\right]$$

$$Q^\pi(s_t,a_t) = \mathrm{E}[r(s_t,a_t) + \gamma V^\pi(s_{t+1})\ ]$$

$$\begin{aligned} A^\pi(s_t,a_t) &= Q^\pi(s_t,a_t) - V^\pi(s_t) \\ &= r(s_t,a_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) \end{aligned}$$

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)(r(s_t,a_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)\ )\right]$$

https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Advantage actor-critic

REINFORCE (Monte Carlo PG)

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log\pi_\theta(a_t|s_t)\Phi_t\right] \qquad \Phi_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

Advantage Actor-Critic

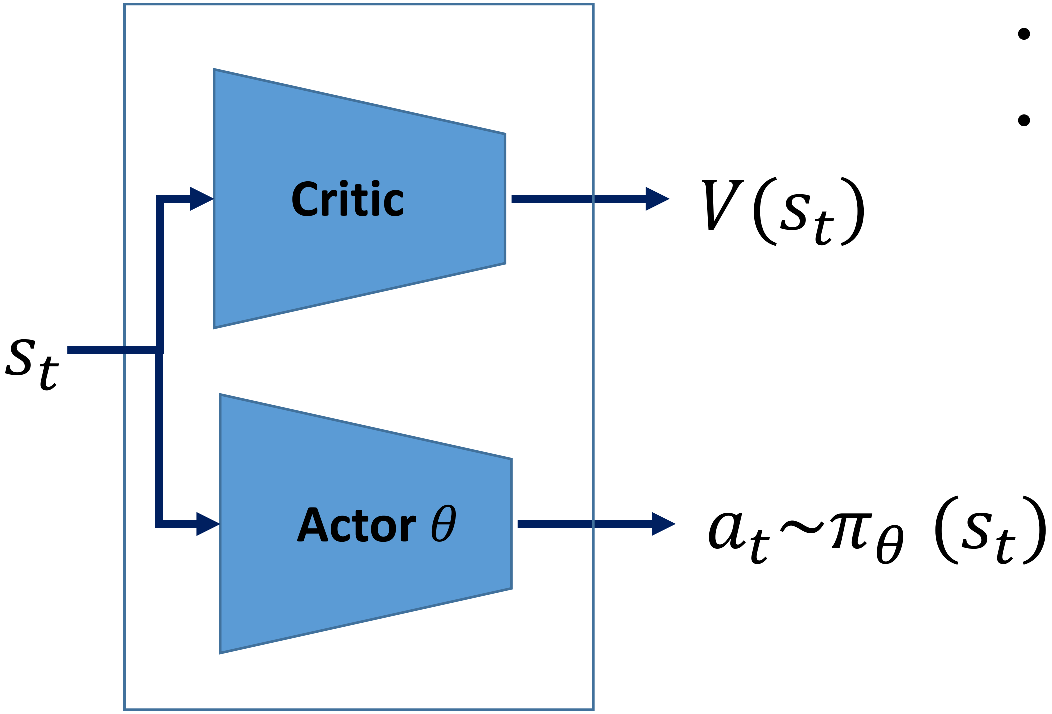
$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log\pi_\theta(a_t|s_t)\big(Q^\pi(s_t,a_t) - V^\pi(s_t)\big)\right]$$

$$= E_{\tau\sim\pi_\theta}\left[\textstyle\sum_{t=0}^{T} \nabla_\theta \log\pi_\theta(a_t|s_t)(r(s_t,a_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t))\right]$$

https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

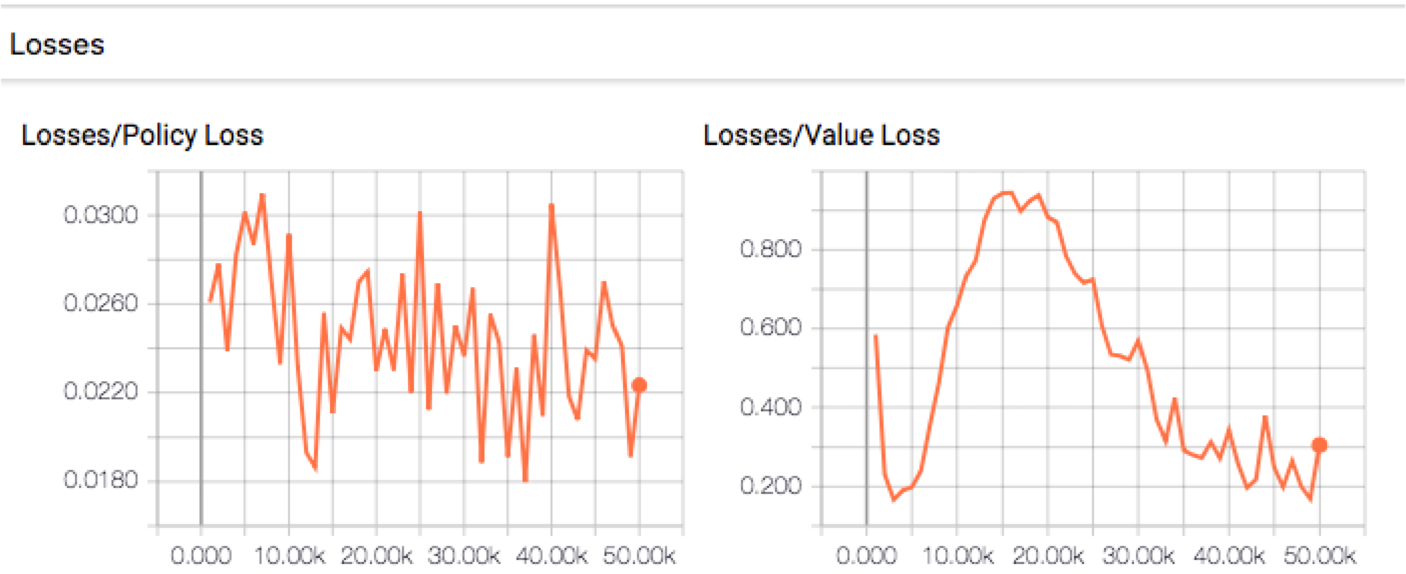
Advantage actor-critic (A2C)

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log\pi_\theta(a_t|s_t)\left(r(s_t,a_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)\right)\right]$$

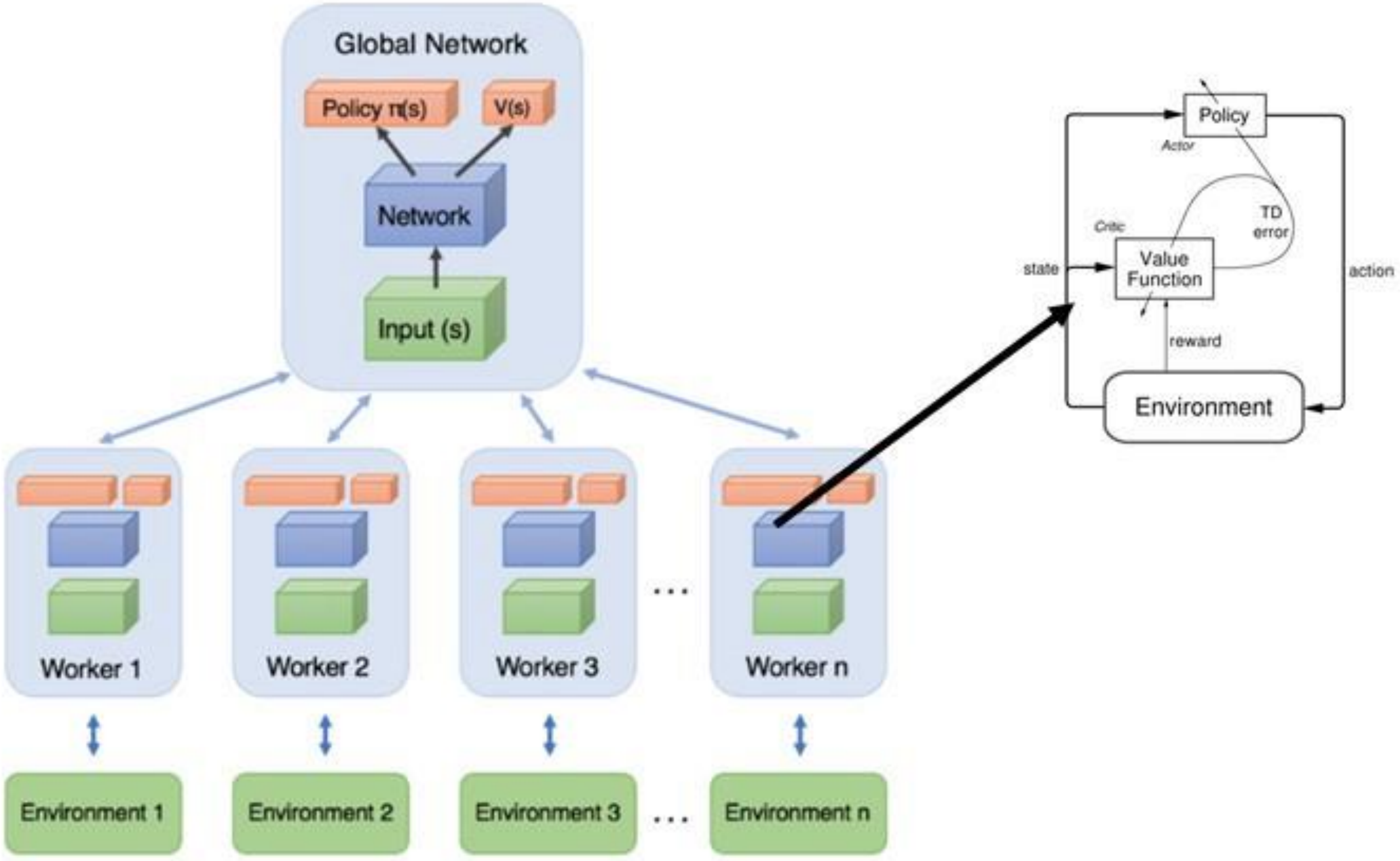


$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)$$

- The "Critic" estimates the value function.
- The "Actor" updates the policy distribution in the direction suggested by the Critic.



Asynchronous Advantage Actor Critic (A3C)



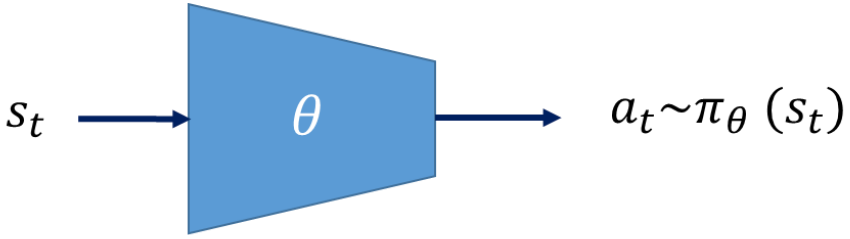
https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Sampling efficiency problem

REINFORCE (Monte Carlo PG)

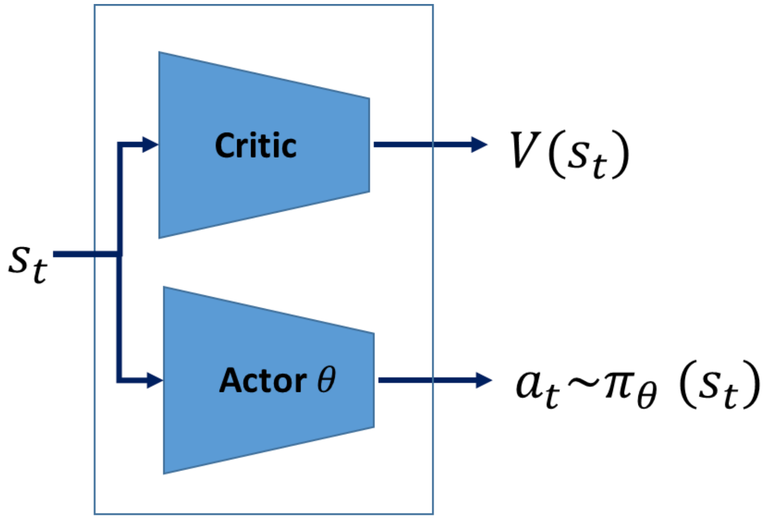
$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t\right] \qquad \Phi_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)$$



Advantage Actor-Critic (A2C)

$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\left(Q^\pi(s_t,a_t) - V^\pi(s_t)\right)\right]$$
$$= E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\left(r(s_t,a_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)\right)\right]$$



$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)$$

Important sampling

$$E_{x\sim \mathrm{p}}[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = E_{x\sim \mathrm{q}}\left[f(x)\frac{p(x)}{q(x)}\right]$$

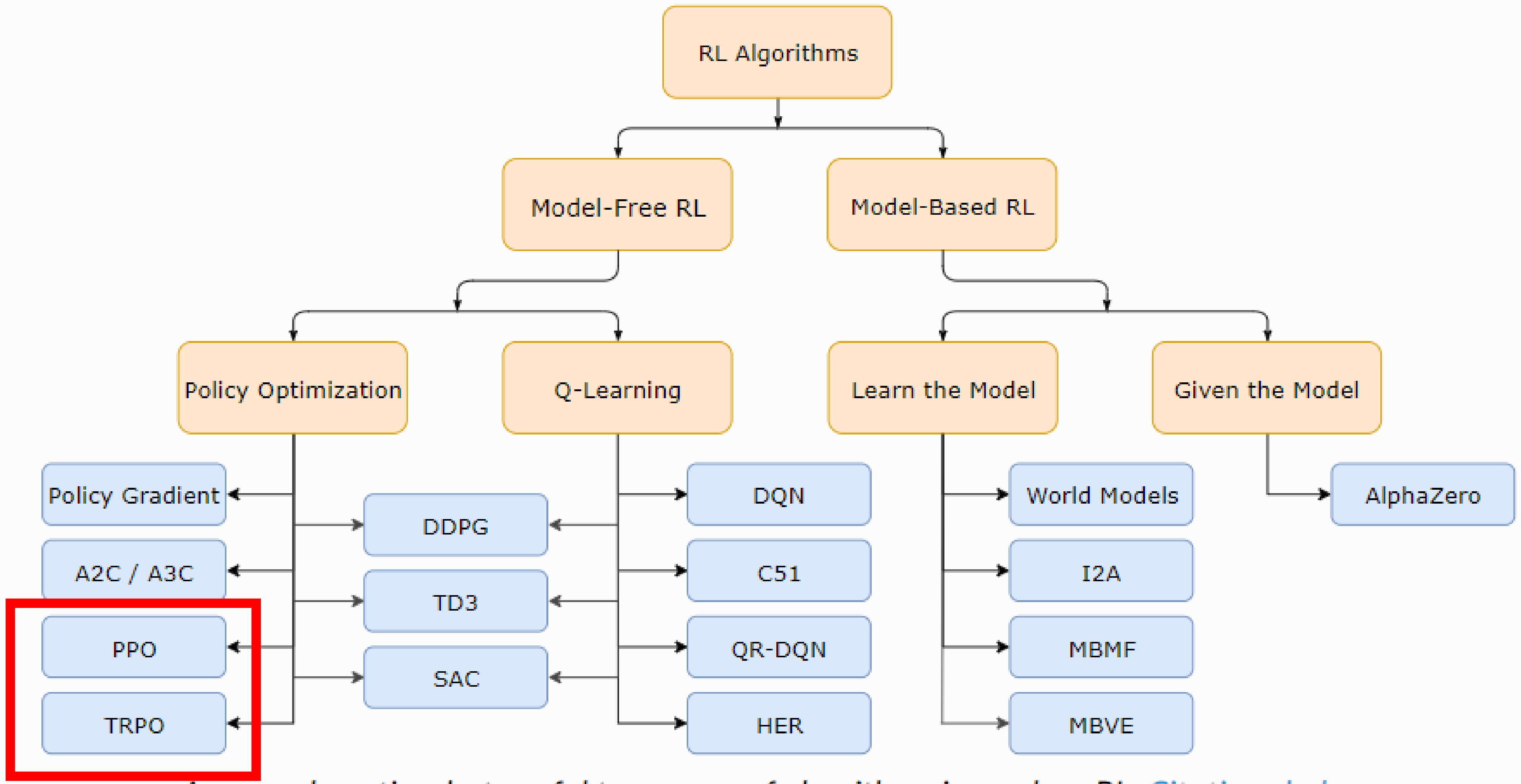
$$Var_{x\sim \mathrm{p}}[f(x)] = E_{x\sim \mathrm{p}}[f(x)^2] - \left(E_{x\sim \mathrm{p}}[f(x)]\right)^2 \qquad \mathrm{VAR[X] = E(X^2) - (E[X])^2}$$

$$Var_{x\sim \mathrm{q}}\left[f(x)\frac{p(x)}{q(x)}\right] = E_{x\sim \mathrm{q}}\left[\left(f(x)\frac{p(x)}{q(x)}\right)^2\right] - \left(E_{x\sim \mathrm{q}}\left[f(x)\frac{p(x)}{q(x)}\right]\right)^2$$

$$= E_{x\sim \mathrm{p}}\left[f(x)^2\frac{p(x)}{q(x)}\right] - \left(E_{x\sim \mathrm{p}}[f(x)]\right)^2$$

李弘毅: https://youtu.be/OAKAZhFmYoI

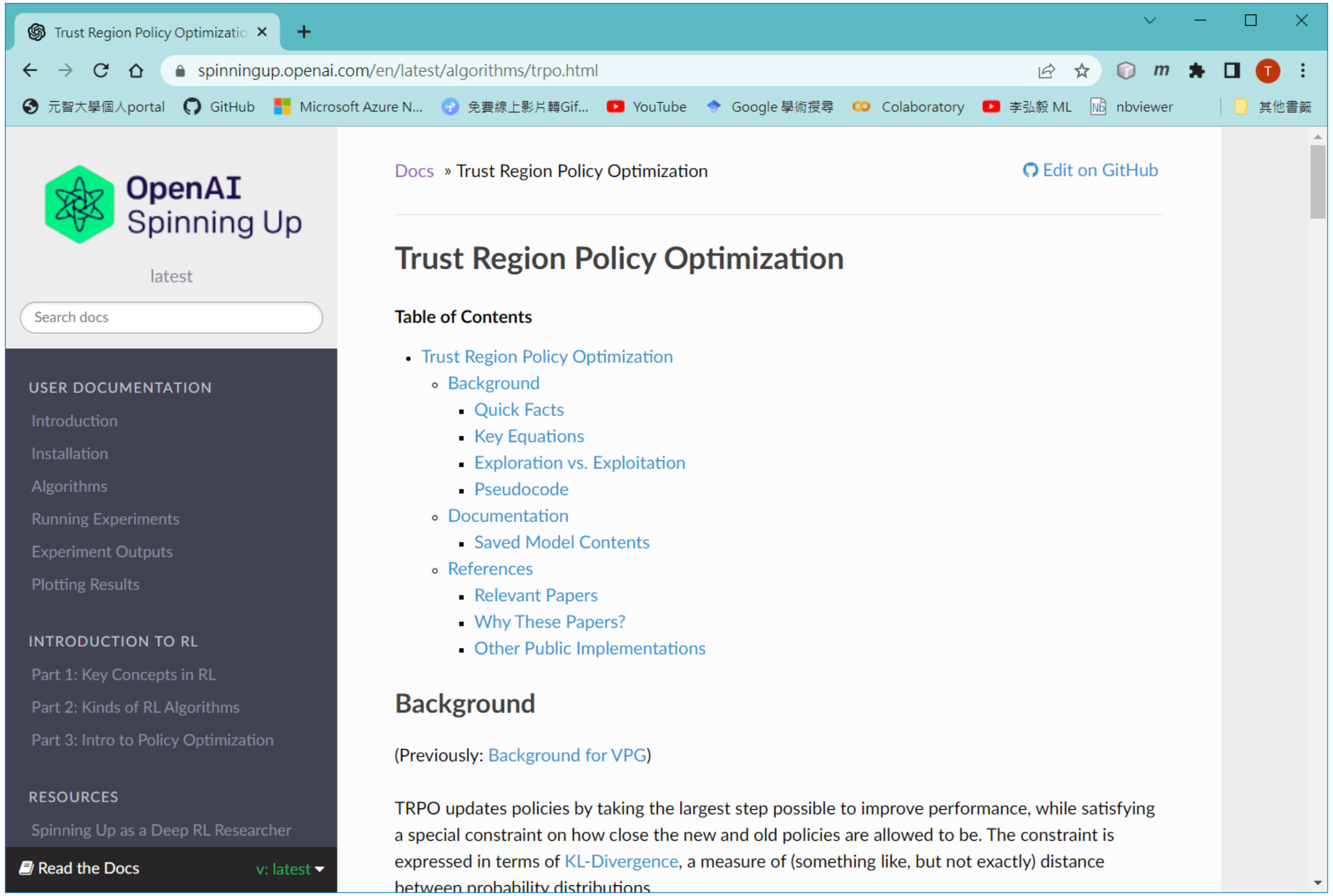
Solving sampling efficiency problem



A non-exhaustive, but useful taxonomy of algorithms in modern RL. Citations below.

TRPO

How can we make policy optimization more sampling efficient?



[Welcome to Spinning Up in Deep RL! — Spinning Up documentation (openai.com)]

TRPO

$$\max\ J(\pi) = E_{\tau\sim\pi}\left(R(\tau)\right)$$

$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)$$

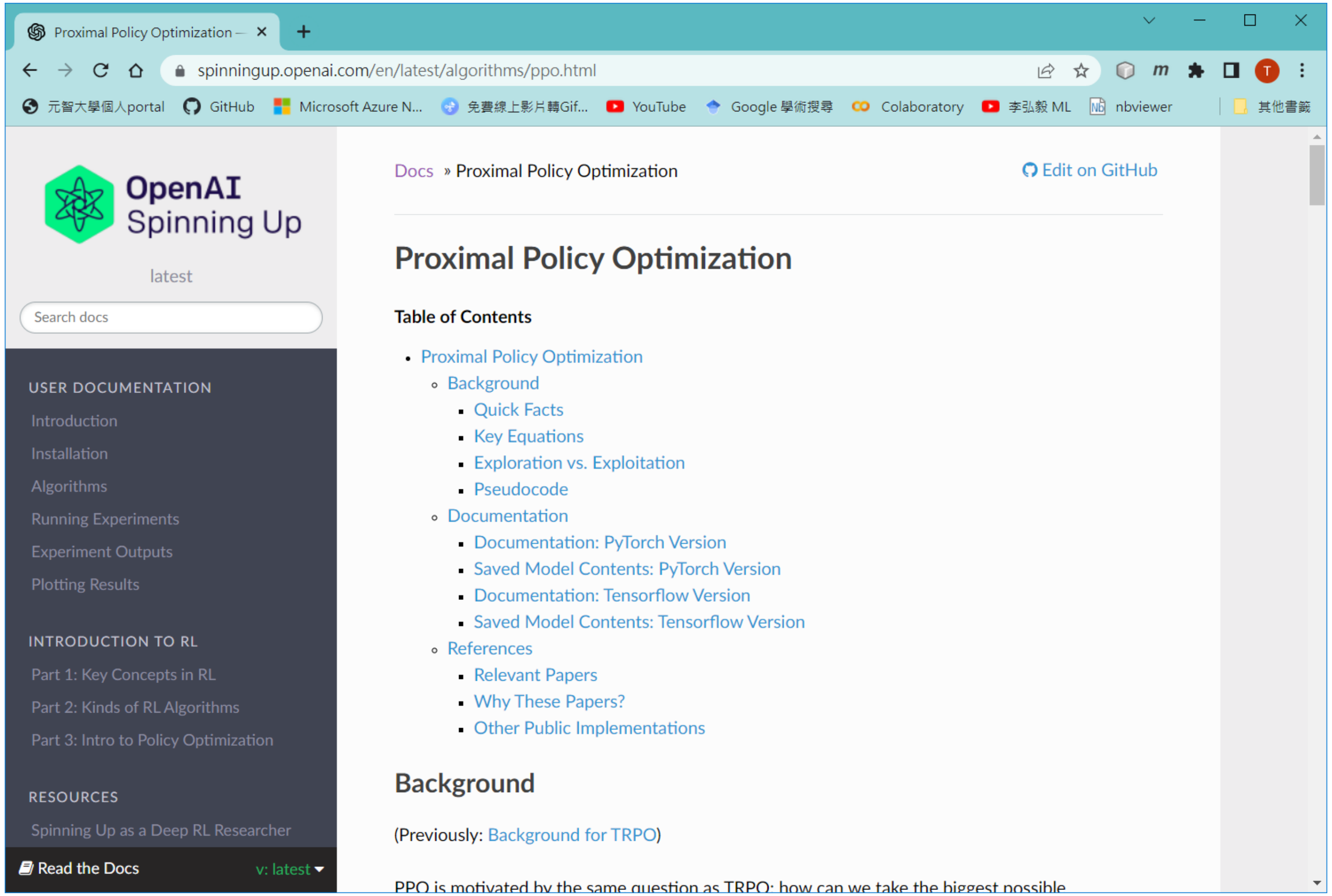
$$\theta_{k+1} = \arg\max_\theta \mathcal{L}(\theta_k, \theta)$$
$$s.t.\ \bar{D}_{\mathrm{KL}}(\theta \parallel \theta_k) \le \delta$$

$$\mathcal{L}(\theta_k, \theta) = E_{s,a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_\theta}(s,a)\right]$$

$$\bar{D}_{\mathrm{KL}}(\theta \parallel \theta_k) = E_{s\sim\pi_{\theta_k}}\left[D_{KL}\left(\pi_\theta(\cdot\,|s) \parallel \pi_{\theta_k}(\cdot\,|s)\right)\right]$$

PPO

How can we make policy optimization more sampling efficient?



[Welcome to Spinning Up in Deep RL! — Spinning Up documentation (openai.com)]

PPO

$$\max\ J(\pi) = E_{\tau\sim\pi}\left(R(\tau)\right)$$

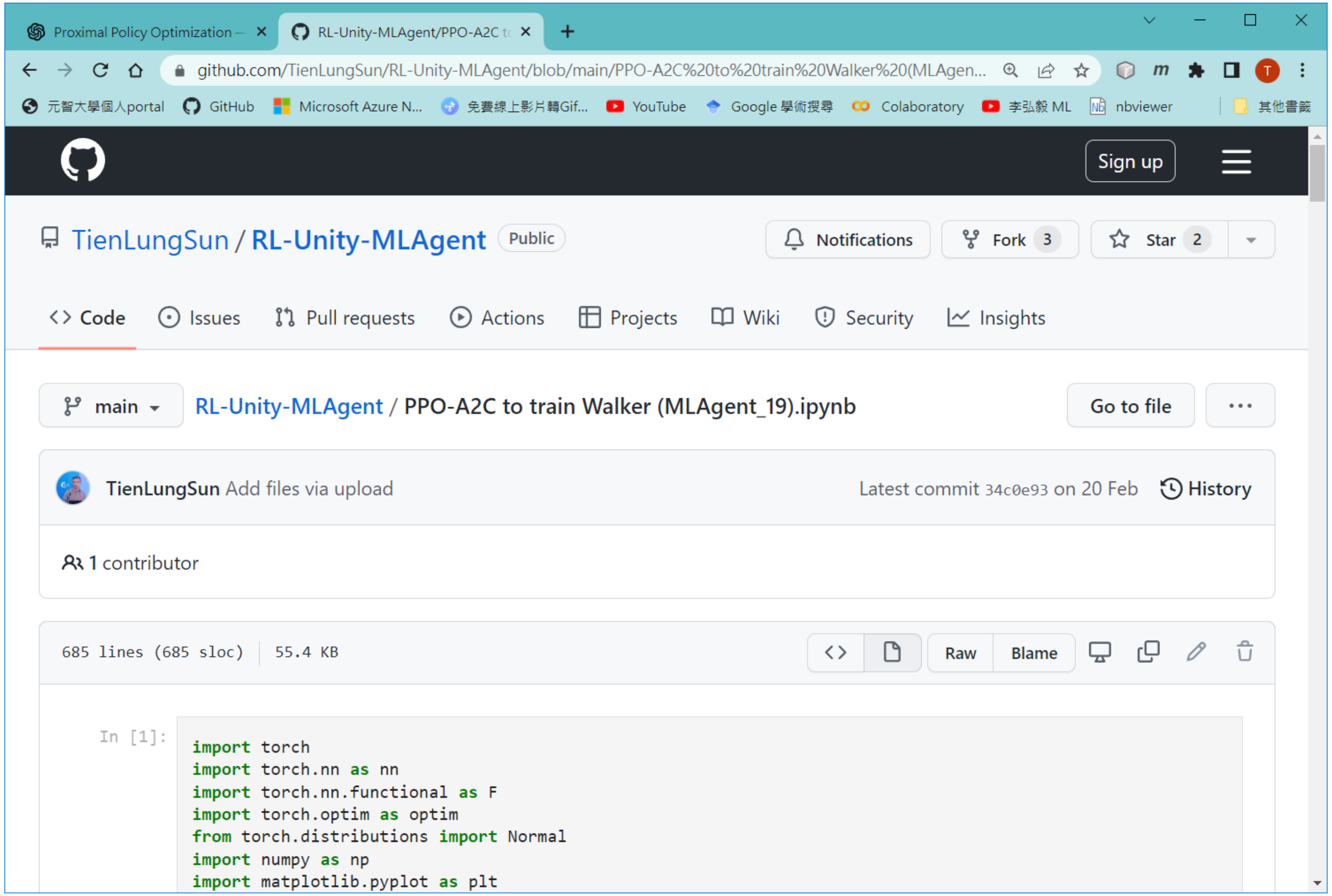
$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)$$

$$\theta_{k+1} = \arg\max_\theta E_{s,a\sim\pi_{\theta_k}}\left[L(s,a,\theta_k,\theta)\right]$$

$$[L(s,a,\theta_k,\theta)] = min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), clip\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\varepsilon, 1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right)$$

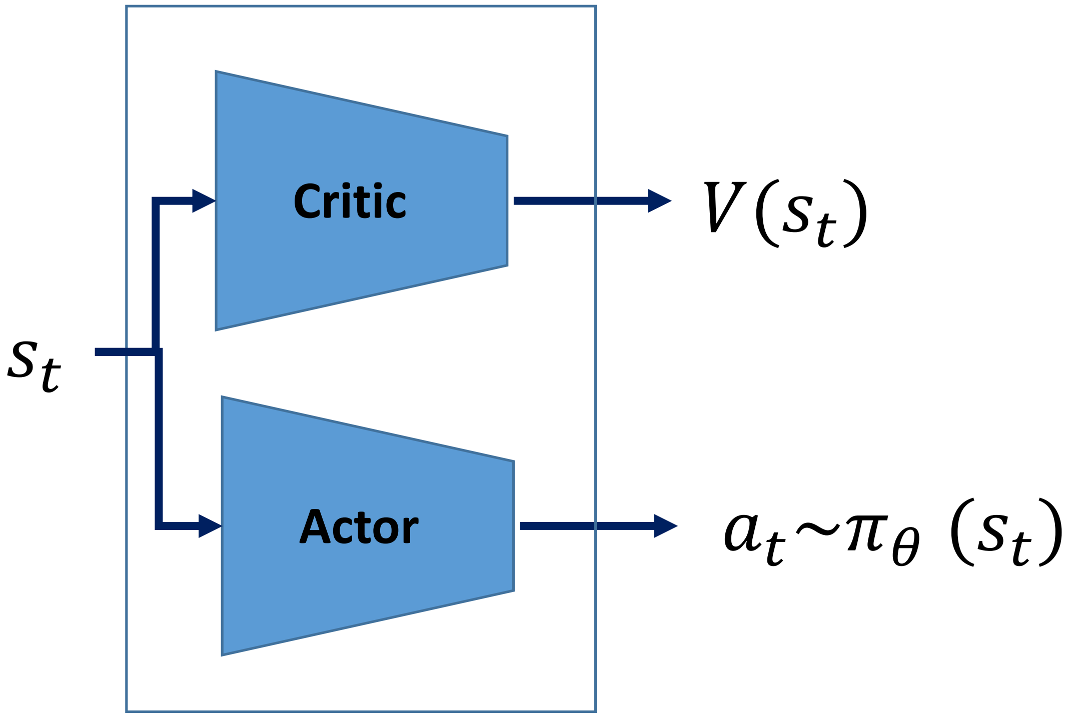
PyTorch Implementation

My GitHub → RL-Unity-MLAgent → PPO-A2C.ipynb

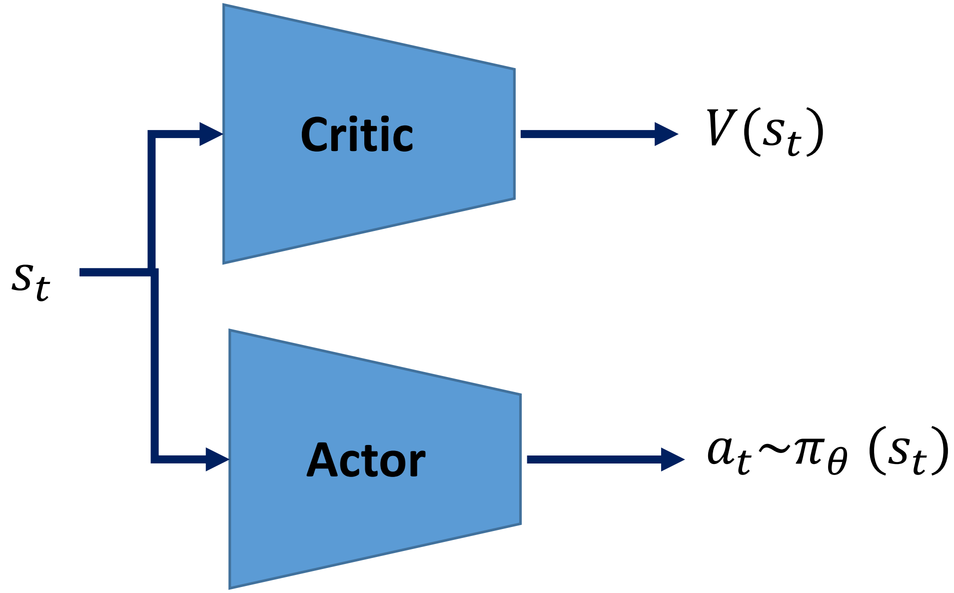


PyTorch implementation of A2C

```python
class Net(nn.Module):
    def __init__(self, ):
        super(Net, self).__init__()

        self.critic = nn.Sequential(
            nn.Linear(N_STATES, 128),
            nn.LayerNorm(128),
            nn.Linear(128, 128),
            nn.LayerNorm(128),
            nn.Linear(128, 1)
        )

        self.actor = nn.Sequential(
            nn.Linear(N_STATES, 128),
            nn.LayerNorm(128),
            nn.Linear(128, 128),
            nn.LayerNorm(128),
            nn.Linear(128, N_ACTIONS)
        )
        self.log_std = nn.Parameter(torch.ones(1,
        self.apply(init_weights)

    def forward(self, x):
        value = self.critic(x)
        mu    = self.actor(x)
        std   = self.log_std.exp().expand_as(mu)
        dist  = Normal(mu, std)
        return dist, value
```



Define loss function



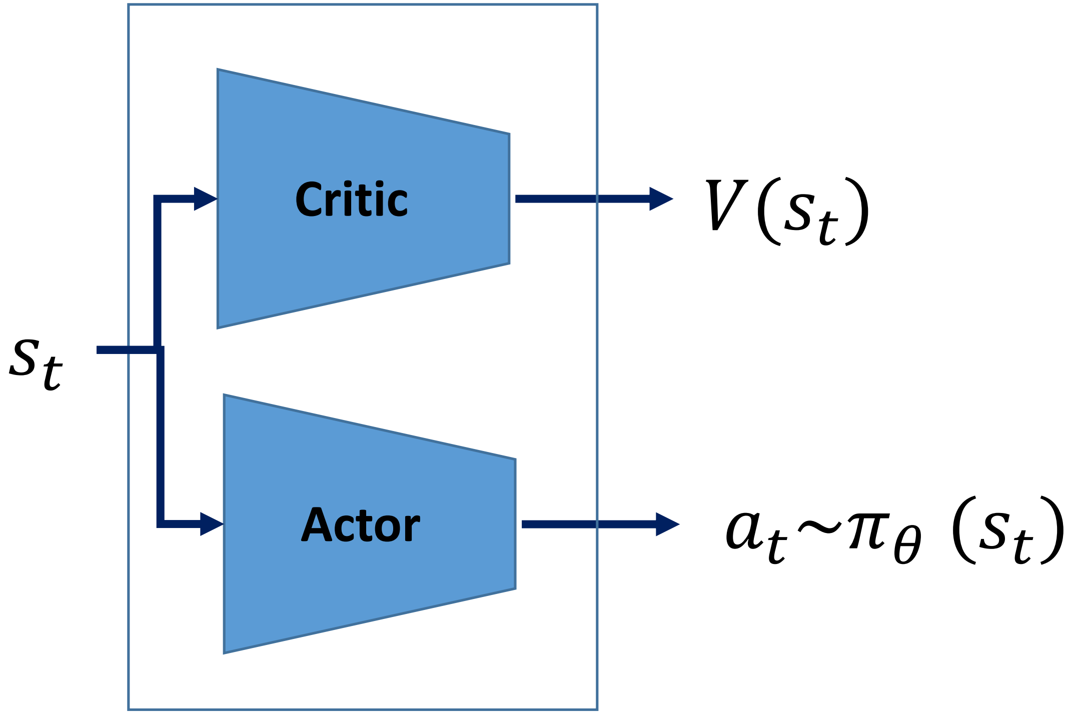
$$L = L_\pi + c_v L_v$$

$$Loss_\pi = \sum_{(s_t,a_t)} min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A^{\theta'}(s_t,a_t), clip\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right) A^{\theta'}(s_t,a_t)\right)$$

$$A^\theta(s_t,a_t) = r_t^n + \gamma V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)$$

$$Loss_v = \left(r_t^n + \gamma V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)\right)^2$$

Add entropy-based regularization



$$L = L_\pi + c_v L_v + c_{reg} L_{reg}$$

Define loss function

```
dist, value = net(batch_state.to(device))
critic_loss = (batch_return.to(device) - value).pow(2).mean()
entropy = dist.entropy().mean()
batch_action = dist.sample()
batch_new_log_probs = dist.log_prob(batch_action)
ratio = (batch_new_log_probs - batch_old_log_probs.to(device)).exp()
surr1 = ratio * batch_advantage.to(device)
surr2 = torch.clamp(ratio, 1.0 - clip_param, 1.0 + clip_param) * batc
actor_loss  = - torch.min(surr1, surr2).mean()
loss = 0.5 * critic_loss + actor_loss - 0.001 * entropy
```

$$L = L_\pi + c_v L_v + c_{reg} L_{reg}$$

$$Loss_v = \left(r_t^n + \gamma V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)\right)^2$$

$$Loss_\pi = \sum_{(s_t, a_t)} min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A^{\theta'}(s_t, a_t), clip\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right) A^{\theta'}(s_t, a_t)\right)$$

Combine data collected from different agents

Use PPO to update NN weights and biases

```
returns   = torch.cat(returns).detach()
log_probs = torch.cat(log_probs).detach()
values    = torch.cat(values).detach()
states    = torch.cat(states)
actions   = torch.cat(actions)
advantage = returns - values
```

```
print(len(returns), returns[0].shape)
print(len(log_probs), log_probs[0].shape)
print(len(values), values[0].shape)
print(len(states), states[0].shape)
print(len(actions), actions[0].shape)
print(len(advantage), advantage[0].shape)
```

```
60 torch.Size([1])
60 torch.Size([2])
60 torch.Size([1])
60 torch.Size([8])
60 torch.Size([2])
60 torch.Size([1])
```

N: no. of agents
K: time horizon

$$\begin{bmatrix} \vec{s}_{1,step1} \\ \vdots \\ \vec{s}_{N,step1} \\ \vdots \\ \vec{s}_{1,stepk} \\ \vdots \\ \vec{s}_{N,stepk} \end{bmatrix} \begin{bmatrix} \vec{a}_{1,step1} \\ \vdots \\ \vec{a}_{N,step1} \\ \vdots \\ \vec{a}_{1,stepk} \\ \vdots \\ \vec{a}_{N,stepk} \end{bmatrix} \quad \begin{bmatrix} v_{1,step1} \\ \vdots \\ v_{N,step1} \\ \vdots \\ v_{1,stepk} \\ \vdots \\ v_{N,stepk} \end{bmatrix} \begin{bmatrix} return_{1,step1} \\ \vdots \\ return_{N,step1} \\ \vdots \\ return_{1,stepk} \\ \vdots \\ return_{N,stepk} \end{bmatrix} \begin{bmatrix} gae_{1,step1} \\ \vdots \\ gae_{N,step1} \\ \vdots \\ gae_{1,stepk} \\ \vdots \\ gae_{N,stepk} \end{bmatrix}$$

Calculate GAE

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta log\pi_\theta(a_t|s_t)\Phi_t\right]$$

$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$$

$$\nabla\bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{20}\nabla \log p_\theta(a_t^n|s_t^n)A^{\pi_\theta}(s_t, a_t)$$

$$\Phi_t = A^{\pi_\theta}(s_t, a_t)$$

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$
$$= \left(r_t^n + V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)\right)$$

Δ = reward + expected accumulated reward
gae = Δ + accumulated gae
Return = gae + expected accumulated reward

$$\Delta_{20} = r_{20} + (\gamma * v_{21} * mask_{20} - v_{20})$$
$$gae_{20} = \Delta_{20} + \gamma * \tau * mask_{20} * gae_{initial}$$
$$return_{20} = gae_{20} + v_{20}$$

$$\Delta_{19} = r_{19} + (\gamma * v_{20} * mask_{19} - v_{19})$$
$$gae_{19\sim20} = \Delta_{19} + \gamma * \tau * mask_{19} * gae_{20}$$
$$return_{19} = gae_{19\sim20} + v_{19}$$

…

$$\Delta_{1} = r_{1} + (\gamma * v_{2} * mask_{1} - v_{1})$$
$$gae_{1\sim20} = \Delta_{1} + \gamma * \tau * mask_{1} * gae_{2\sim20}$$
$$return_{1} = gae_{1\sim20} + v_{1}$$

Sampling a batch of data to train NN

```python
batch_size = states.size(0)
for _ in range(batch_size // mini_batch_size):
    rand_ids = np.random.randint(0, batch_size, mini_batch_size)
    break
print(rand_ids)
print(states[rand_ids, :].shape)
print(actions[rand_ids, :].shape)
print(log_probs[rand_ids, :].shape)
print(returns[rand_ids, :].shape)
print(advantage[rand_ids, :].shape)
```

```
[39 52 11  8 45]
torch.Size([5, 8])
torch.Size([5, 2])
torch.Size([5, 2])
torch.Size([5, 1])
torch.Size([5, 1])
```

$$
\begin{bmatrix} \vec{s}_{1,step1} \\ \vdots \\ \vec{s}_{N,step1} \\ \vdots \\ \vec{s}_{1,step20} \\ \vdots \\ \vec{s}_{N,step20} \end{bmatrix}
\begin{bmatrix} \vec{a}_{1,step1} \\ \vdots \\ \vec{a}_{N,step1} \\ \vdots \\ \vec{a}_{1,step20} \\ \vdots \\ \vec{a}_{N,step20} \end{bmatrix}
\qquad
\begin{bmatrix} return_{1,step1} \\ \vdots \\ return_{N,step1} \\ \vdots \\ return_{1,step20} \\ \vdots \\ return_{N,step20} \end{bmatrix}
\begin{bmatrix} gae_{1,step1} \\ \vdots \\ gae_{N,step1} \\ \vdots \\ gae_{1,step20} \\ \vdots \\ gae_{N,step20} \end{bmatrix}
$$

Select one batch of train data to optimize actor

$$\theta_{k+1} = \arg\max_{\theta} E_{s,a\sim\pi_{\theta_k}}\left[min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), clip\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\varepsilon, 1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right)\right]$$

```
net = Net().to(device)
```

```
optimizer = optim.Adam(net.parameters(), lr=0.001)
```

```
actor_loss  = - torch.min(surr1, surr2).mean()
print(actor_loss)
```

```
tensor(-0.0410, device='cuda:0', grad_fn=<NegBackward>)
```

```
optimizer.zero_grad()
actor_loss.backward()
optimizer.step()
```

Select one batch of train data to optimize actor

$$\theta_{k+1} = \arg\max_{\theta} E_{s,a\sim\pi_{\theta_k}}\left[min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), clip\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\varepsilon, 1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right)\right]$$

select one batch and perform PPO optimization

```
for batch_state, batch_action, batch_old_log_probs, batch_re
    break
```

```
print(batch_state.shape, batch_action.shape)
```

```
torch.Size([5, 8]) torch.Size([5, 2])
```

```
dist = net(batch_state.to(device))
print(dist)
```

```
Normal(loc: torch.Size([5, 2]), scale: torch.Size([5, 2]))
```

Select one batch of train data to optimize actor

$$\theta_{k+1} = \arg\max_{\theta} E_{s,a\sim\pi_{\theta_k}}\left[min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), clip\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\varepsilon, 1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right)\right]$$

```
batch_action = dist.sample()
batch_new_log_probs = dist.log_prob(batch_action)
print(batch_new_log_probs.shape)
```

```
torch.Size([5, 2])
```

```
ratio = (batch_new_log_probs - batch_old_log_probs.to(device)).exp()
print(ratio)
```

```
tensor([[1.2427, 0.6327],
        [0.4962, 1.2191],
        [0.8360, 1.0056],
        [1.4339, 0.3089],
        [1.3568, 0.3191]], device='cuda:0', grad_fn=<ExpBackward>)
```

Select one batch of train data to optimize actor

$$\theta_{k+1} = \arg\max_{\theta} E_{s,a\sim\pi_{\theta_k}}\left[min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), clip\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\varepsilon, 1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right)\right]$$

```
surr1 = ratio * batch_advantage.to(device)
print(surr1)
```

```
tensor([[0.0606, 0.0309],
        [0.0242, 0.0595],
        [0.0408, 0.0491],
        [0.0699, 0.0151],
        [0.0662, 0.0156]], device='cuda:0', grad_fn=<MulBackward0>)
```

```
clip_param=0.2
surr2 = torch.clamp(ratio, 1.0 - clip_param, 1.0 + clip_param) * batc
print(surr2)
```

```
tensor([[0.0585, 0.0390],
        [0.0390, 0.0585],
        [0.0408, 0.0491],
        [0.0585, 0.0390],
        [0.0585, 0.0390]], device='cuda:0', grad_fn=<MulBackward0>)
```

```
actor_loss  = - torch.min(surr1, surr2).mean()
print(actor_loss)
```

```
tensor(-0.0410, device='cuda:0', grad_fn=<NegBackward>)
```

PPO-AC training loop

```
while (frame_idx < MAX_STEPS):
    print("\nframe idx = ", frame_idx)
    print("Interacts with Unity to collect training data")
    states, actions, log_probs, values, rewards, masks, next_state = collect_training_data (N_AGEN
    _, next_value = net(next_state.to(device))

    print("Compute GAE of these training data set")
    returns = compute_gae(TIME_HORIZON, next_value, rewards, masks, values, GAMMA, LAMBD)

    returns   = torch.cat(returns).detach()
    log_probs = torch.cat(log_probs).detach()
    values    = torch.cat(values).detach()
    states    = torch.cat(states)
    actions   = torch.cat(actions)
    advantages = returns - values

    print("Optimize NN with PPO")
    critic_loss, actor_loss = ppo_update(N_EPOCH, BATCH_SIZE, states, actions, log_probs, returns,
    CriticLossLst.append(critic_loss)
    ActorLossLst.append(actor_loss)

    frame_idx += TIME_HORIZON
```

Hyper parameters

OpenAI spinning up

Documentation: PyTorch Version

```
spinup.ppo_pytorch(env_fn, actor_critic=<MagicMock spec='str' id='140554322637768'>, ac_kwargs={}, seed=0, steps_per_epoch=4000, epochs=50, gamma=0.99, clip_ratio=0.2, pi_lr=0.0003, vf_lr=0.001, train_pi_iters=80, train_v_iters=80, lam=0.97, max_ep_len=1000, target_kl=0.01, logger_kwargs={}, save_freq=10)
```

Proximal Policy Optimization (by clipping),

with early stopping based on approximate KL

PPO Training parameters

```
behaviors:
  Walker:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
    network_settings:
      normalize: true
      hidden_units: 512
      num_layers: 3
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.995
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 30000000
    time_horizon: 1000
    summary_freq: 30000
```

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log\pi_\theta(a_t|s_t)\left(Q^\pi(s_t,a_t)-V^\pi(s_t)\right)\right]$$

$$= E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log\pi_\theta(a_t|s_t)\left(r(s_t,a_t)+\gamma V^\pi(s_{t+1})-V^\pi(s_t)\right)\right]$$

$$\theta_{k+1} = \arg\max_\theta E_{s,a\sim\pi_{\theta_k}}\left[L(s,a,\theta_k,\theta)\right]$$

$$[L(s,a,\theta_k,\theta)] = min\left(\begin{array}{c}\dfrac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a),\\ clip\left(\dfrac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)},1-\varepsilon,1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\end{array}\right)$$

- beta is the weight of entropy regularization
- lambd is the weight to calculate GAE

HW 3

- Use PPO to train walker
- Change reward setting to let walker have different behavior