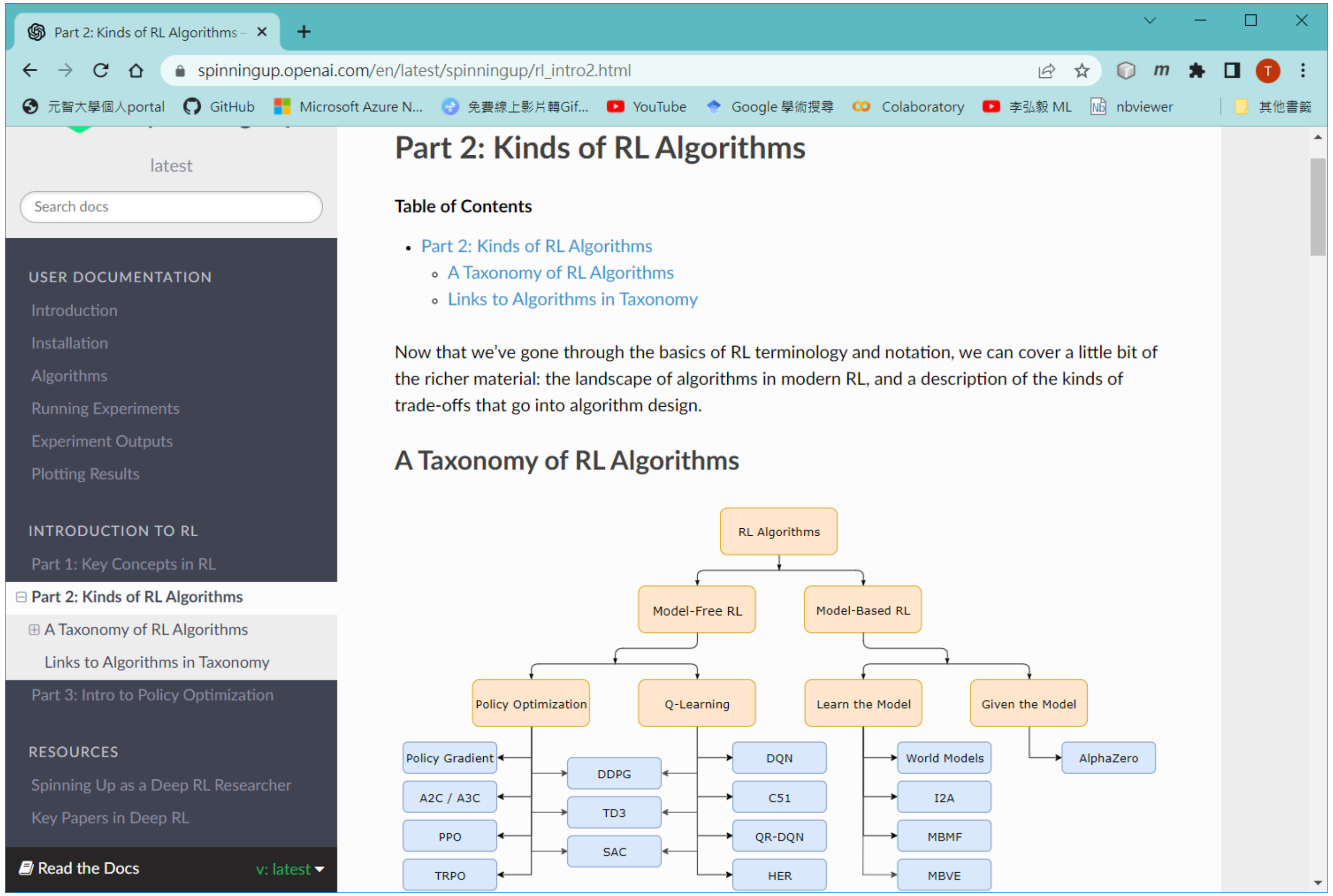
Policy Optimization

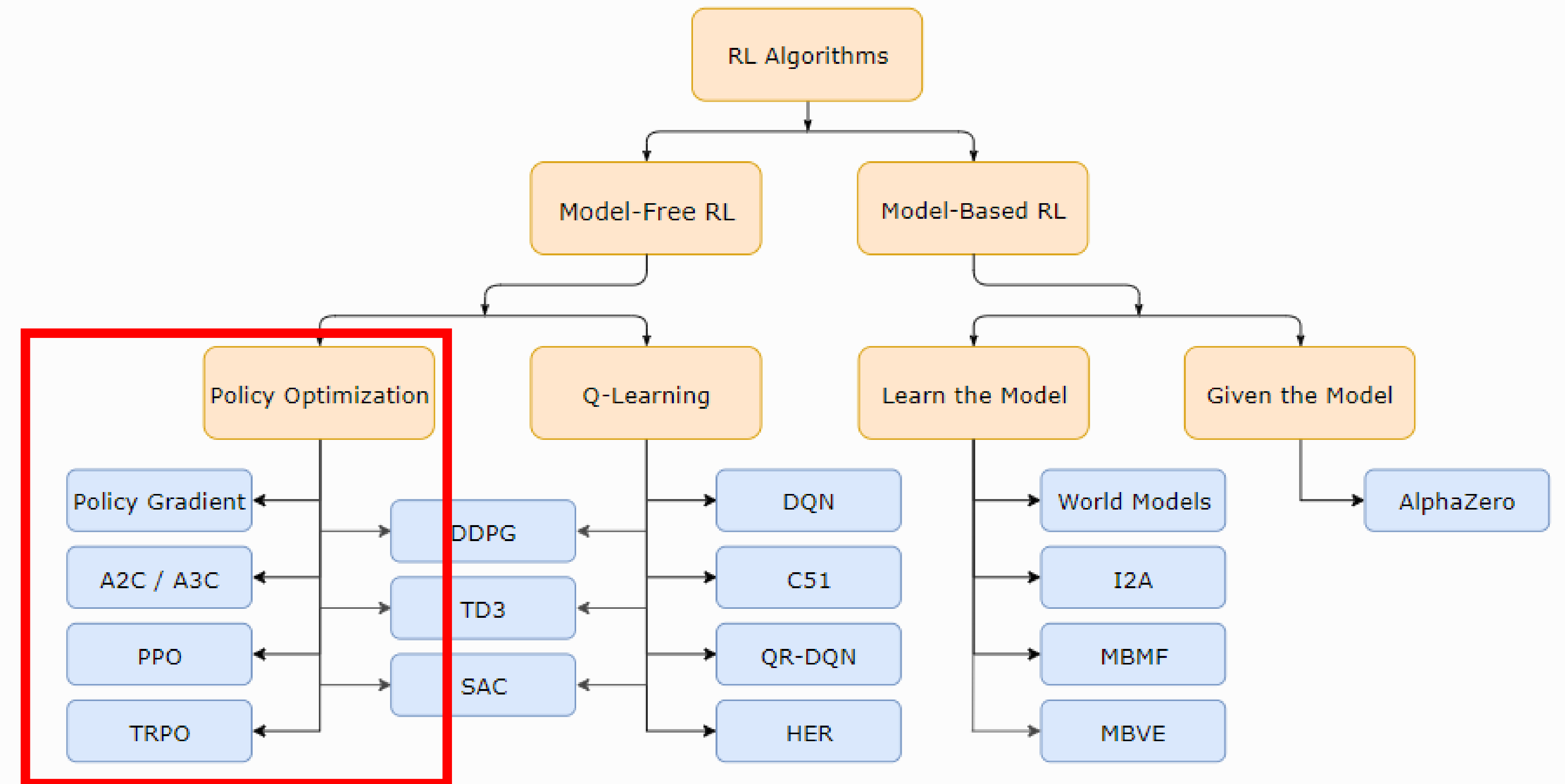
Model free RL includes 1) Policy Optimization and 2) Q-Learning.



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

Policy Optimization

This lecture note discusses policy optimization, including PG, 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	$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$

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

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

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

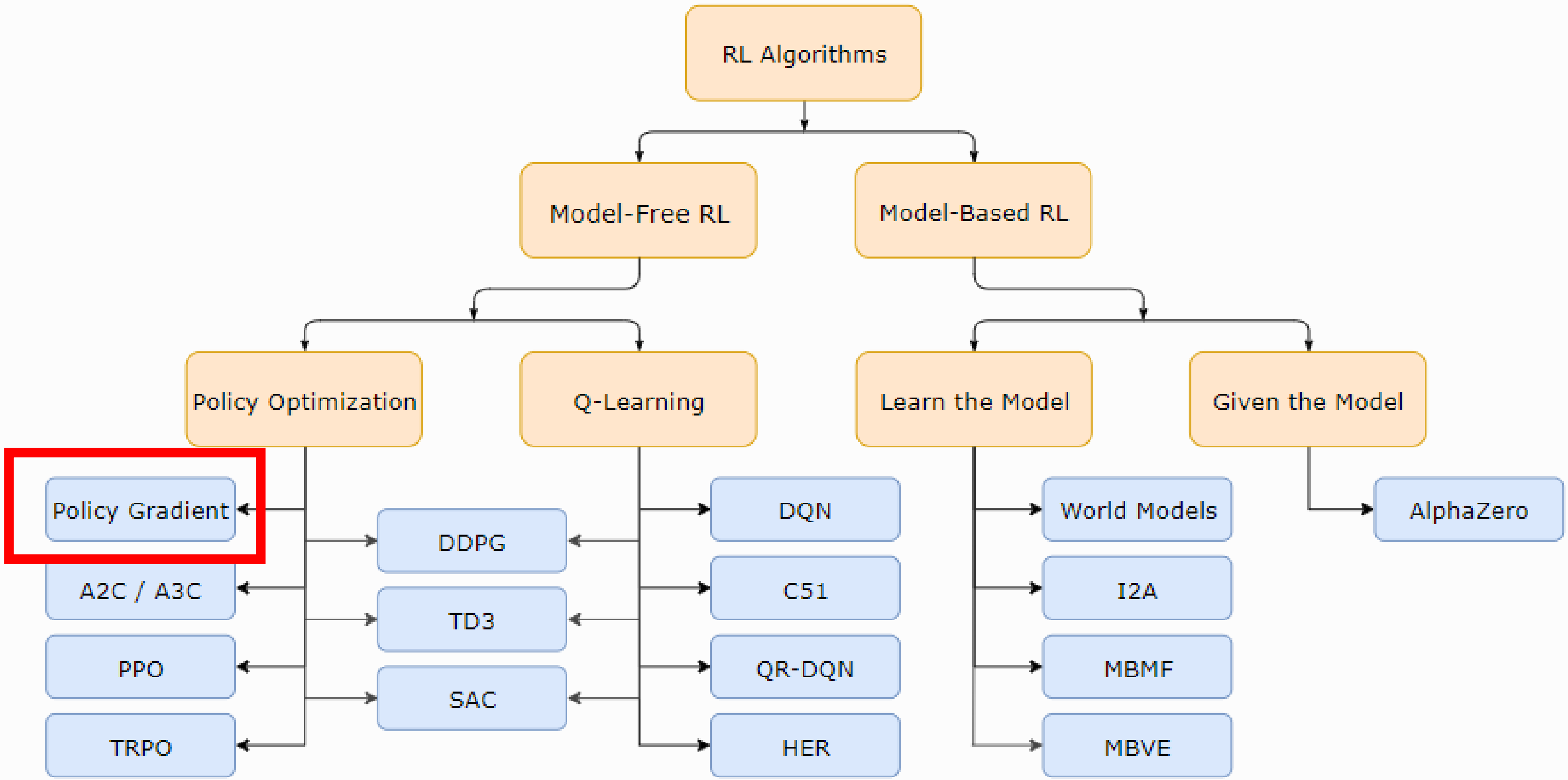
$$Q^\pi(s,a) = E_{\tau\sim\pi}\,(R(\tau)|s_0 = s, a_0 = a)$$

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

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

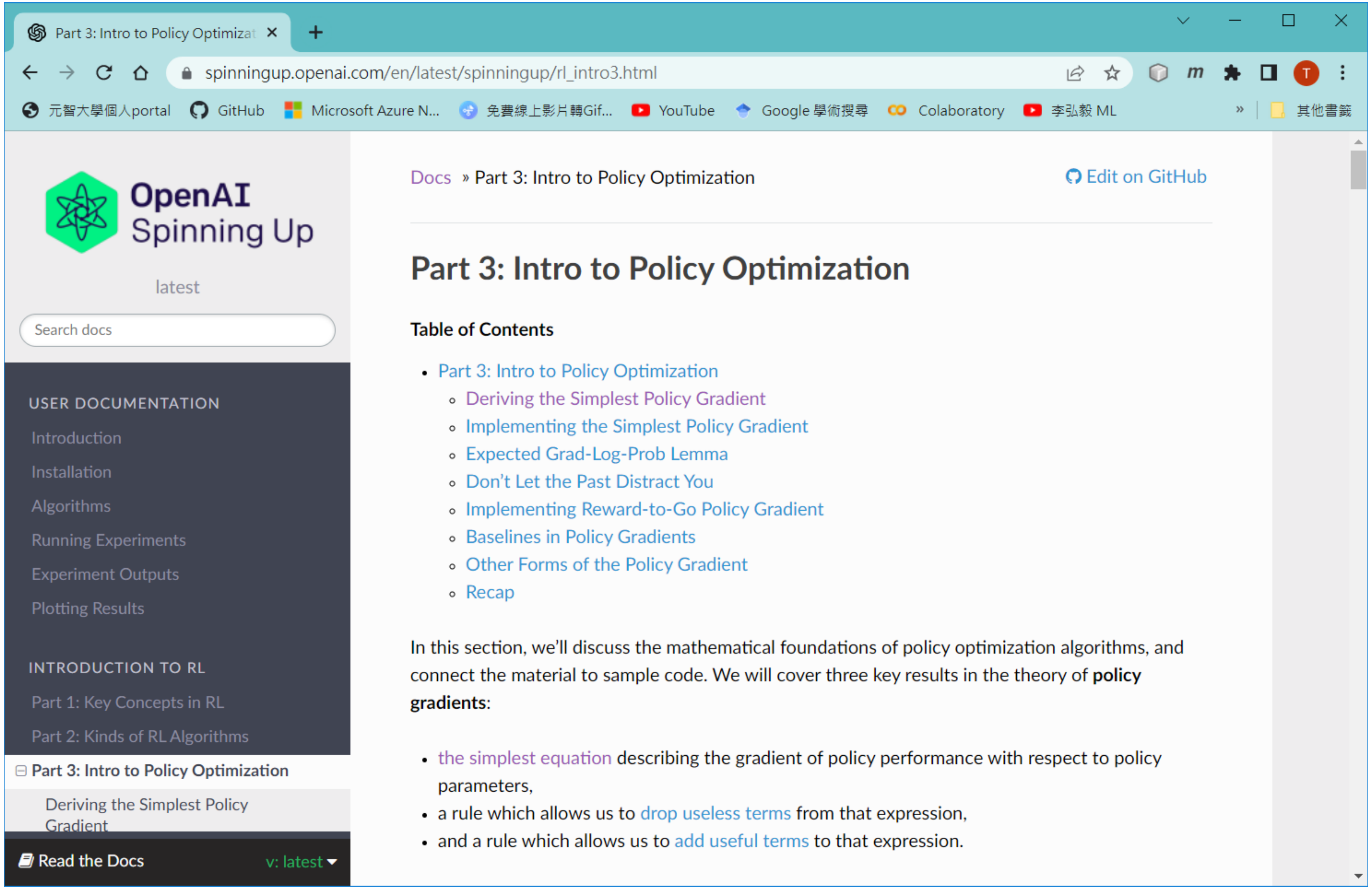
$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Policy gradient



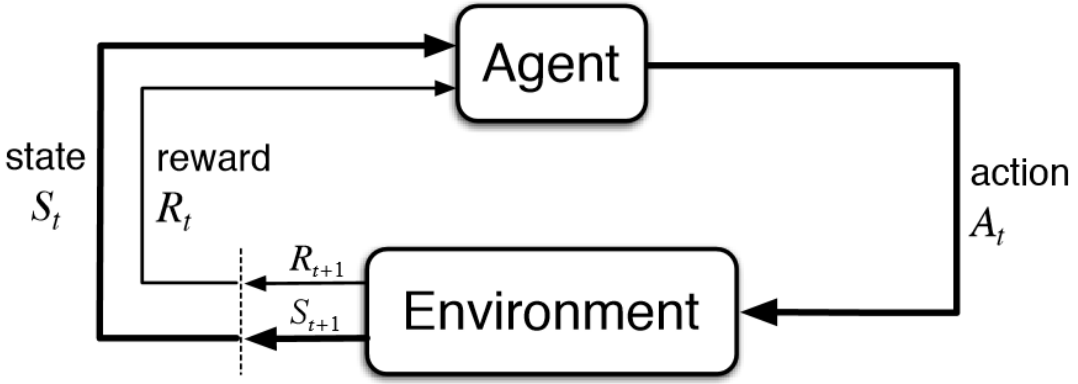
A non-exhaustive, but useful taxonomy of algorithms in modern RL. Citations below.

Policy gradient

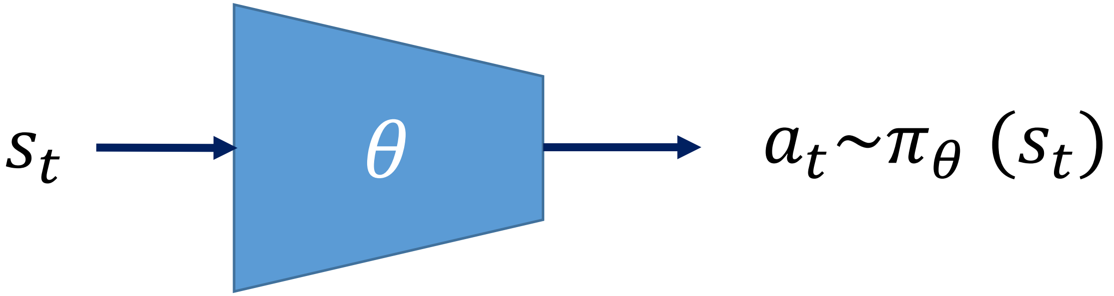


How to train NN to learn the best policy?

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



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



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

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

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

How to maximize cumulative reward?

We need gradient of the expected cumulative reward to do gradient descent.

$$\max J(\pi) = E_{\tau\sim\pi}\left(R(\tau)\right) \qquad \theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)$$

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

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

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

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

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

How to calculate $\nabla_\theta J(\pi_\theta)$?

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

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t) \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})\right]$$

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)\left(\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)\right)\right]$$

How to calculate $\nabla_\theta J(\pi_\theta)$?

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

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

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

$$\Phi_t = R(\tau)$$

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

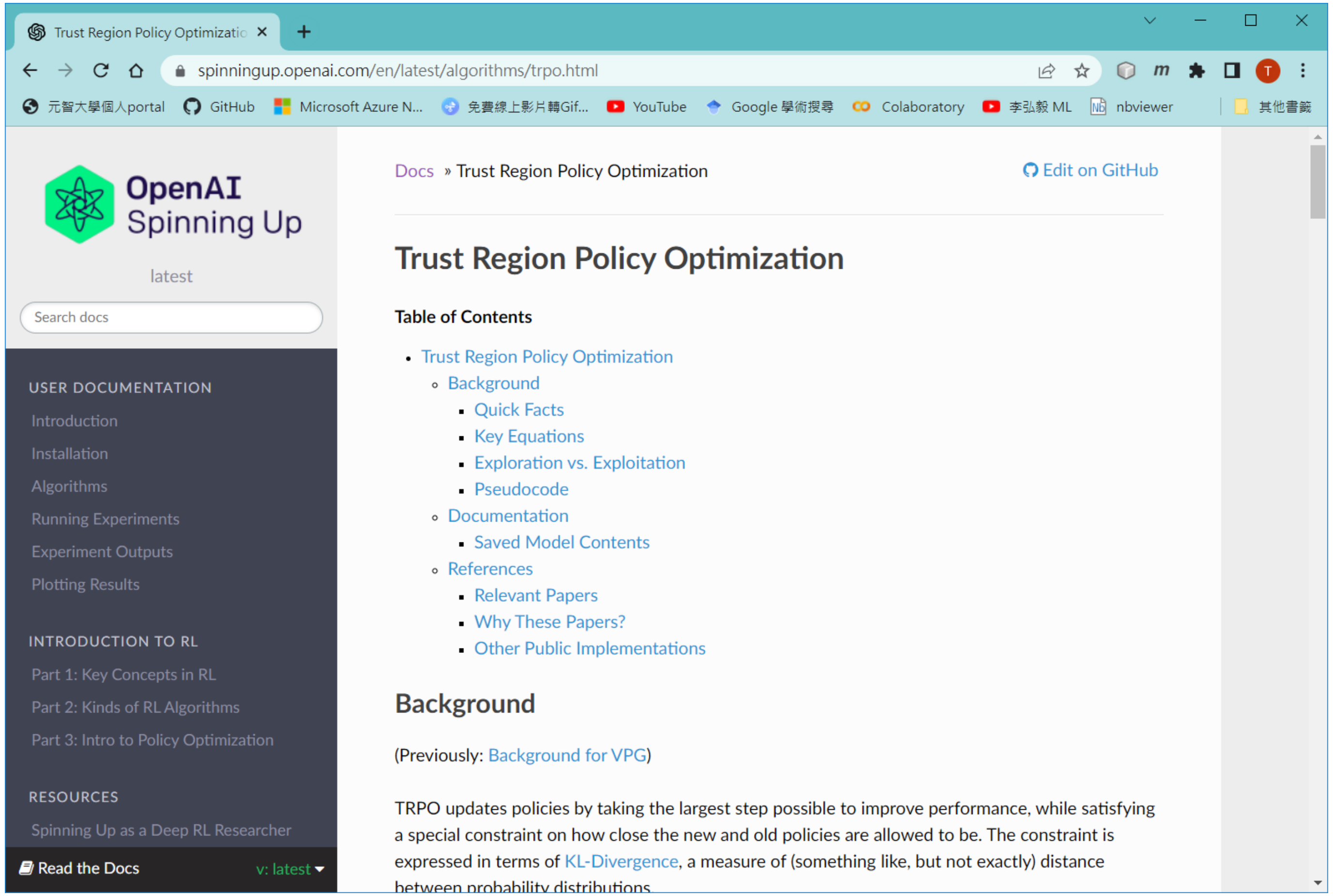
$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$$

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

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

TRPO

How can we make policy optimization more sampling efficient?



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

Important sampling

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

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

$$\mathrm{VAR[X] = E(X^2) - (E[X])^2}$$

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

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

李弘毅: https://youtu.be/OAKAZhFmYoI

TRPO

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

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

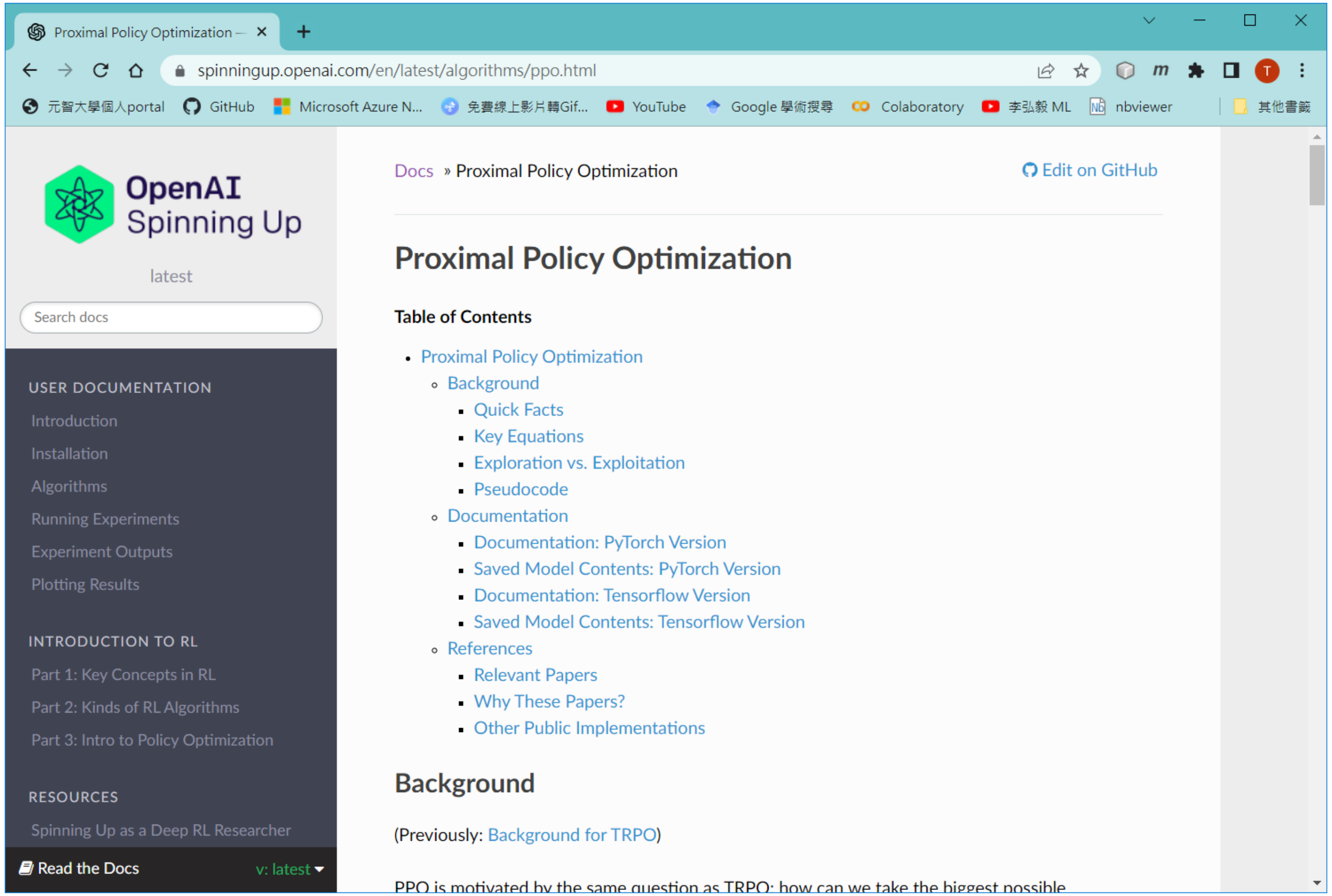
$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}(\theta_k, \theta)$$
$$s.t.\ \bar{D}_{\mathrm{KL}}(\theta \parallel \theta_k) \le \delta$$

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

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

PPO

How can we make policy optimization more sampling efficient?



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

PPO

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

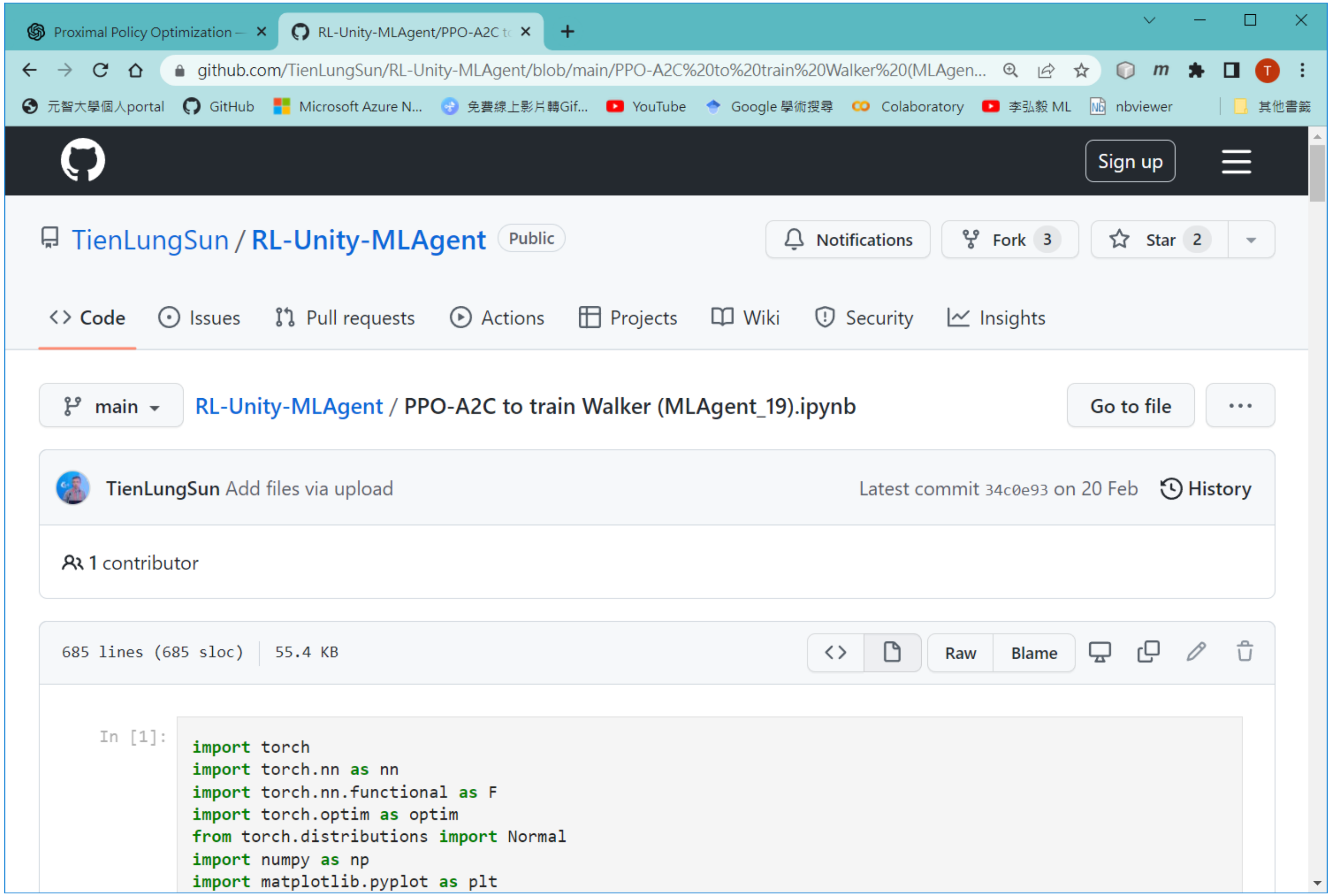
$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)$$

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

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

PyTorch Implementation

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



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

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

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

…

$$\Delta_1 = r_1 + (\gamma * v_2 * mask_1 - v_1)$$
$$gae_{1\sim20} = \Delta_1 + \gamma * \tau * mask_1 * gae_{2\sim20}$$
$$return_1 = gae_{1\sim20} + v_1$$

Sampling a batch of data to train NN

```
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])
```

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

Select one batch of train data to optimize NN

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

```
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 NN

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

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 NN

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

```
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 NN

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

```
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>)
```

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

Hyper parameters

Unity ML agent → Walker.yaml

3DBall.yaml

```yaml
behaviors:
  3DBall:
    trainer_type: ppo
    hyperparameters:
      batch_size: 64
      buffer_size: 12000
      learning_rate: 0.0003
      beta: 0.001
      epsilon: 0.2    ε
      lambd: 0.99    τ
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99    γ
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 50000
    time_horizon: 1000
    summary_freq: 5000
    threaded: true
```