

Policy Learning 2

offline learning

data is collected and reused before learning like imitation learning

online learning

data is collected along with the update of learning process

- on-policy learning: the data used for learning comes from current policy
- off-policy learning: the data used for learning can come from old policy
 - this can result in problems, because the action took at a state changes with the policy. Though I may take some action in the past, I don't do that now with current policy, then learning that data point will be of little use.

model-based learning

- trying to learning world model, a.k.a, $r(s_t, a_t)$ and $p(s_{t+1} | s_t, a_t)$.
- This method saves the cost of interacting with the environment, because we can choose the action from the optimum $r(s_t, a_t)$. So this method is worth trying for real world RL.

model-free learning

- trying to learn the optimum policy, a.k.a, θ which gives out $\pi_\theta(a_t | s_t)$.
- if classify the methods by having actor network or not, having critic network or not, there are three categories:
 - only actor: REINFORCE
 - actor-critic: A3C, PPO
 - in this method, critic network only helps the actor learn, it never plays a role in the decision making phase.
 - only-critic: value-based methods like Q-learning, DQN
 - these method gives out the best action by searching through the whole action space and selecting the one with the highest value for the current state. So it is mostly applied in discrete action space.
- two main challenges:
 - i. noisy gradient
 - ii. sample efficiency

On-policy learning

REINFORCE

1. optimization target: $\operatorname{argmax}_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} (\sum_{t=0}^T r(s_t, a_t))$
2. important formula:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [\sum_{t=0}^T r(s_t, a_t)] \quad (1)$$

$$= \int p_\theta(\tau) r(\tau) d\tau \quad (2)$$

$$\approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^T r(s_t, a_t) \quad (3)$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau \quad (4)$$

$$= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) d\tau \quad (5)$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} [\nabla_\theta \log p_\theta(\tau) r(\tau)] \quad (6)$$

3. pipeline:

- i. rollout the policy π_θ in the environment to collect data τ
- ii. calculate $\nabla_\theta J(\theta)$
- iii. update the policy θ using gradient ascent: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
- iv. go back to step 1 and repeat (about 1M step in total)

4. intuition:

- i. what's the relationship between this naive method(REINFORCE) and imitation learning?
 - what imitation learning learns:

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) \right)$$

- what REINFORCE learns:

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) \right) \left(\sum_{t=1}^T r(s_{i,t}, a_{i,t}) \right)$$

So basically, REINFORCE learns the weighted sum of the log-likelihood of action and the reward, while imitation learning just take all these weights to 1.

- ii. In REINFORCE, for the trajectory with high reward, **all** the state and action pairs (s_t, a_t) on this trajectory are encouraged, which makes the model choose a_t with a higher probability at s_t

5. drawbacks:

- i. high variance: $\sum_{i=1}^N \nabla_\theta \log \pi_\theta(\tau) r(\tau)$ changes dramatically for different trajectories! As mentioned about, all the actions on a trajectory are encouraged to the same extent, even though it is actually a bad decision, it is covered by the good actions. So the sum of gradients on all actions in this trajectory is pretty noisy.
- ii. baseline: if all trajectories have positive rewards, than all the trajectories are encouraged. we need a baseline to separate good trajectories from bad ones.

6. refinement:

- i. reducing variance: high variance results from same weight for all (action, state) pairs in a trajectory. we can use 'reward to go' to assign different weights to them. For taking action $a_{i,t}$ at state $s_{i,t}$, we have weight $\sum_{t'=t}^T r(s_{i,t'}, a_{i,t'})$.
- ii. baseline: simply, we can use the average of all trajectories' rewards as the baseline, those trajectories with rewards lower than the average will be punished, while those with higher rewards will be encouraged. More precisely, we can choose baseline b as

$$\operatorname{argmin}_b \operatorname{Var} = E[x^2] - E[x]^2 \quad (7)$$

$$= E_\tau [(\nabla_\theta \log \pi_\theta(\tau)(r(\tau) - b))^2] - \quad (8)$$

$$E_\tau [\nabla_\theta \log \pi_\theta(\tau)(r(\tau) - b)]^2 \quad (9)$$

, this gives out optimum b in theory by letting the grad to be 0:

$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]}$$

Actor-Critic

1. intuition: We want to further address the high variance drawback of REINFORCE. In the refinement of REINFORCE, we have 'reward to go' which is $\sum_{t'=t}^T r(s_{i,t'}, a_{i,t'})$ to better estimate the appropriate weights for each (state, action) pair. However, this estimated 'reward to go' also suffers from high variance, because it is calculated from only one trajectory. Now we want to use network to directly predict the 'reward to go', which is the values functions.

2. value functions:

- $Q^\pi(s_t, a_t) = r(s_{t+1}, a_{t+1}) + \mathbb{E}_{s_{t+1} \sim p(s_{t+1} | s_t, a_t)} [V^\pi(s_{t+1})]$
- $V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi_\theta(a_t | s_t)} [Q^\pi(s_t, a_t)]$
- $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$

NOTE: all the value functions are conditioned on the current policy π_θ

3. the new target using Q^π as reward to go, V^π as baseline:

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) (Q^\pi(s_{i,t}, a_{i,t}) - V^\pi(s_{i,t}))$$

4. critic network:

- i. which value function to fit?
 - if we fit Q^π , then $V^\pi = E[Q^\pi(s_t, a_t)]$. And we can apply supervision as $Q^\pi(s_t, a_t) = E[\sum_{t'=t}^T r(s_{i,t'}, a_{i,t'}) | s_t, a_t]$, we don't have multiple trajectories start from s_t and take actions a_t , so we simply take the average of all rewards after this timestep on this trajectory as supervision. This is the same as plain reward to go, but we can expect the network to learn a more exactly representation of it.

- if we fit V^π , then $Q^\pi = r(s_t, a_t) + E[V^\pi(s_{t+1})]$. The supervision will be $V^\pi(s_t) = E[r(s_{t+1}, a_{t+1})|s_t]$, again in practice, we simply take the average of all rewards after this timestep on this trajectories as supervision. And $A^\pi(s_t, a_t)$ is simplified as $r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t)$.

ii. here we fit V^π . And for all the pairs $\{(s_{i,t}, r(s_{i,t}, a_{i,t}) + \hat{V}_\phi^\pi(s_{i,t+1}))\} := \{(s_{i,t}, y_{i,t})\}$ the loss function will be $L(\phi) = \frac{1}{2} \sum_i ||\hat{V}_\phi^\pi(s_i) - y_{i,t}||^2$

5. discount factors:

- intuition: episode length can't grow forever, we wish to get rewards quicker.
- distinguish two forms:
 - option1:

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\sum_{k=t}^T \gamma^{k-t} r(s_{i,k}, a_{i,k}) \right)$$

- option2:

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\sum_{k=t}^T \gamma^{k-1} r(s_{i,k}, a_{i,k}) \right) \quad (10)$$

$$= \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \gamma^{t-1} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\sum_{k=t}^T \gamma^{k-t} r(s_{i,k}, a_{i,k}) \right) \quad (11)$$

NOTE: we don't want when the pair (s_t, a_t) appears influence the importance of that pair, we just start add the discount to the following rewards from where that pair appears. So option1 is right

6. pipeline

- batch actor-critic:
 - sample s_i, a_i from $\pi_\theta(a|s)$
 - fit $\hat{V}_\phi^\pi(s)$ to sampled reward sums
 - evaluate $\hat{A}^\pi(s_i, a_i) = r(s_i, a_i) + \gamma \hat{V}_\phi^\pi(s'_i) - \hat{V}_\phi^\pi(s_i)$
 - $\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_i|s_i) \hat{A}^\pi(s_i, a_i)$
 - update ϕ using gradient ascent: $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$
 - go back to step 1 and repeat
- online actor-critic:
 - take action $a \sim \pi_\theta(a|s)$, get (s, a, s', r)
 - update $\hat{V}_\phi^\pi(s)$ using $r + \gamma \hat{V}_\phi^\pi(s')$
 - evaluate $\hat{A}^\pi(s, a) = r(s, a) + \gamma \hat{V}_\phi^\pi(s') - \hat{V}_\phi^\pi(s)$
 - $\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(a|s) \hat{A}^\pi(s, a)$
 - update ϕ using gradient ascent: $\phi \leftarrow \phi + \alpha \nabla_\phi J(\phi)$
 - go back to step 1 and repeat

7. further refinement: GAE

- Actor-Critic use $r(s, a) + \gamma \hat{V}_\phi^\pi(s') - \hat{V}_\phi^\pi(s)$ as weight
 - low variance (due to learned critic)
 - not unbiased (the critic is not perfect)
- policy gradient use $(\sum_{t'=t}^T \gamma^{t'-t} r(s_{t'}, a_{t'})) - b$ as weight
 - high variance (only one trajectory)
 - no bias
- n-step returns:
 - $\hat{A}_n^\pi(s_t, a_t) = \sum_{k=t}^{t+n} \gamma^{k-t} r(s_k, a_k) + \gamma^{n+1} \hat{V}_\phi^\pi(s_{t+n+1}) - \hat{V}_\phi^\pi(s_t)$
 - the first n steps comes from the sample, which reduce the bias; using critic after n steps to reduce the variance.
- GAE(Generalized Advantage Estimation):
 - $\hat{A}_{GAE}^\pi(s_t, a_t) = \sum_{n=0}^{\infty} \omega^n \hat{A}_n^\pi(s_t, a_t)$, which means GAE is trying to combine all n-step returns by weight average them. Usually the weight will be exponential falloff: $\omega_n \sim \lambda^n$
 - the function is equal to: $\hat{A}_{GAE}^\pi(s_t, a_t) = \sum_{t'=t}^{\infty} (\lambda \gamma)^{t'-t} \delta_{t'}$, where $\delta_{t'} = r(s_{t'}, a_{t'}) + \gamma \hat{V}_\phi^\pi(s_{t'+1}) - \hat{V}_\phi^\pi(s_{t'})$

Review On-policy learning

basically, these methods tried to reduce the variance, but the sample efficiency is not considered.

- Actor-critic algorithms:
 - Actor: the policy
 - Critic: value function
 - Reduce variance of policy gradient

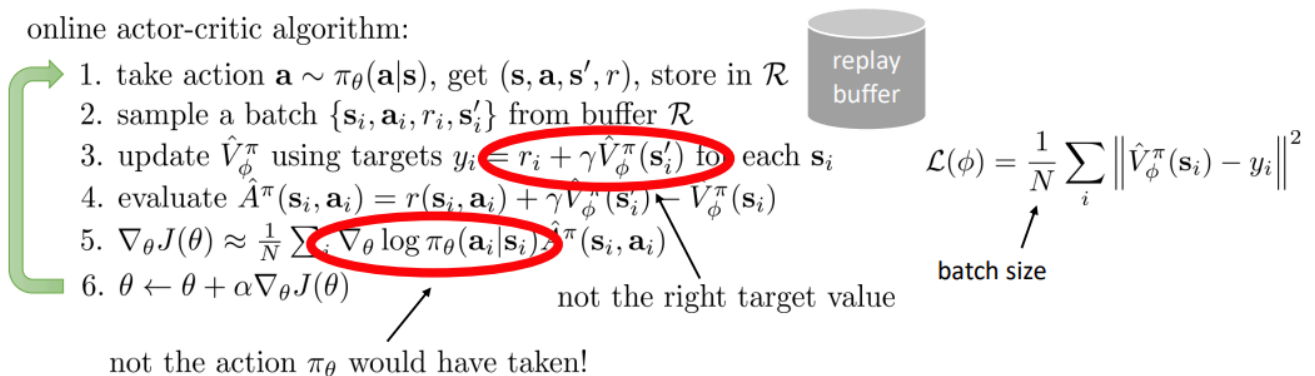
- Discount factors
 - encourage short-term rewards
 - also a variance reduction trick
- Policy evaluation
 - Fitting value function to policy
- State-dependent baselines (learn $\hat{V}_\phi^\pi(s)$ as baseline)
 - Another way to use the critic
 - Can combine: n-step returns or GAE
- Actor-critic algorithm design
 - One network (with two heads) or two networks
 - Batch-mode, or online (+ parallel)
 - Batch-mode: update once per batch (trajectory)
 - Online: update once per step, but running many robots in parallel, assemble all the (s, a) pair and update the network.

Off-policy learning

off-policy actor-critic

consider a actor-critic method with buffer:

online actor-critic algorithm:



This algorithm is broken!

Can you spot the problems?

there are a few problems with this method:

1. when we update critic network ($\hat{V}_\phi^\pi(s)$), we use the data (s, a, s', r) from old policy, this does not reflect the change in policy. So the predicted value function is not accurate.
 - we predict $\hat{Q}_\phi^\pi(s, a)$ instead of $\hat{V}_\phi^\pi(s)$. And consider $\hat{Q}_\phi^\pi(s, a) = r(s, a) + \gamma E[\hat{V}_\phi^\pi(s')]$, we can do Monte Carlo for a single time to estimate $E[\hat{V}_\phi^\pi(s')]$, which is $V(s') = E[Q_\phi^\pi(s', a')]$. We then do Monte Carlo again and we have the fitting target: $\hat{Q}_\phi^\pi(s, a) = r(s, a) + \gamma Q_\phi^\pi(s', a')$. But this time, we can reflect the change in policy by taking $a' = \pi'(s')$ instead of $a' = \pi(s')$. Consider the new policy choose action with different probability, we can expect the predicted Q to converge to the Q under current policy.
2. when we calculate the gradient of policy. We cannot use $\nabla_\theta \log(\pi_\theta(a|s))$, because this gradient does not reflect what the current policy will do under s , so we actually need to use $\nabla_\theta \log(\pi'_\theta(a|s))$ instead.
3. how to calculate the baseline when we are estimating Q rather than V ?
 - in this situation the baseline should be $V(s)$ and advantage will be $Q(s, a) - V(s)$. But the truth is we don't have access to $V(s)$, and we cannot even estimate it. So we just $Q(s, a)$ as weight instead of $Q(s, a) - V(s)$. This step introduce high variance.
4. last problem: s is not sampled from current policy. Nothing we can do here, just accept it. In fact, this gives us a optimal policy on a broader distribution, which helps solve transfer problem.

importance sampling

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

policy gradient + importance sampling

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)] = E_{\tau \sim p'_\theta(\tau)}\left[\frac{p_\theta(\tau)}{p'_\theta(\tau)}r(\tau)\right]$$

consider

$$p_{\theta}(\tau) = p(s_1) \prod_{t=1}^T \pi_{\theta}(a_t | s_t) p(s_{t+1} | s_t, a_t)$$

we have

$$\begin{aligned} \nabla'_{\theta} J(\theta') &= E_{\tau \sim p_{\theta}(\tau)} \left[\frac{p_{\theta'}(\tau)}{p_{\theta}(\tau)} \nabla_{\theta'} \log \pi_{\theta'}(\tau) r(\tau) \right] \\ &= E_{\tau \sim p_{\theta}(\tau)} \left[\left(\frac{\pi_{\theta'}(s_1)}{\pi_{\theta}(s_1)} \prod_{t=1}^T \frac{\pi_{\theta'}(a_t | s_t)}{\pi_{\theta}(a_t | s_t)} \right) \left(\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(a_t | s_t) \right) \left(\sum_{t=1}^T r(s_t, a_t) \right) \right] \end{aligned}$$

taking causality into consideration, we don't want future decisions to influence current weight, so we have

$$\nabla'_{\theta} J(\theta') = E_{\tau \sim p_{\theta}(\tau)} \left[\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(a_t | s_t) \left(\frac{\pi_{\theta'}(s_1)}{\pi_{\theta}(s_1)} \prod_{t'=1}^t \frac{\pi_{\theta'}(a_{t'} | s_{t'})}{\pi_{\theta}(a_{t'} | s_{t'})} \right) \left(\sum_{t'=t}^T r(s_{t'}, a_{t'}) \right) \right]$$

How to understand the 'importance' factor $\frac{\pi_{\theta'}(s_1)}{\pi_{\theta}(s_1)} \prod_{t'=1}^t \frac{\pi_{\theta'}(a_{t'} | s_{t'})}{\pi_{\theta}(a_{t'} | s_{t'})}$? It reflects the difference between old policy and current policy. If the current policy is more possible to take this trajectory and get to state s_t (which means the factor will be large), then the rewards to go should be higher weighted, because we need to examine in detail whether this tendency is good or not; however, if the current policy does not like to follow this trajectory, then we don't care the rewards to go that much. In a word, the importance factor show the frequency of the trajectory under current policy.

In practice, we don't want to calculate the long production, so we approximate the importance factor

$$\frac{\pi_{\theta'}(s_1)}{\pi_{\theta}(s_1)} \prod_{t'=1}^t \frac{\pi_{\theta'}(a_{t'} | s_{t'})}{\pi_{\theta}(a_{t'} | s_{t'})}$$

with

$$\frac{\pi_{\theta'}(a_t | s_t)}{\pi_{\theta}(a_t | s_t)}$$

we ignore $\pi_{\theta'}(s_1) / \pi_{\theta}(s_1)$ by assuming that they the two policies are similar.

TRPO (Trust Region Policy Optimization)

use hard KL-divergence constraint to guarantee that the policy won't change too much after one update.

- guarantee to converge to a local optimum
- complicate math

PPO (Proximal Policy Optimization)

- Following TRPO's intuition, PPO provides two methods to constraint policy distribution's change using the first-order algorithm
 - Clipped Surrogate Objective

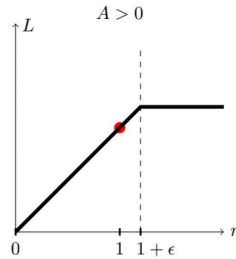
PPO – Clipped Surrogate Objective

- Clip the probability ratio to range $[1 - \epsilon, 1 + \epsilon]$

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

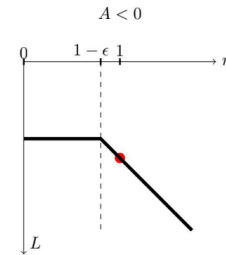
- Condition 1: $A^{\pi_{\theta_k}}$ is positive.

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



- Condition 2: $A^{\pi_{\theta_k}}$ is negative.

$$L(s, a, \theta_k, \theta) = \max \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}, (1 - \epsilon) \right) A^{\pi_{\theta_k}}(s, a)$$



◦ Adaptive KL Penalty Coefficient

23

PPO – Adaptive KL Penalty Coefficient

- KL constraint in TRPO is hard to apply.
 - Adaptive KL penalty coefficient has similar effect.
- Adaptive KL can be alternative to clipped surrogate objective.
- Achieve some target value of KL divergence d_{targ} .

$$L^{KL PEN}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t - \beta \text{KL}[\pi_{\theta_{\text{old}}}(\cdot | s_t), \pi_{\theta}(\cdot | s_t)] \right]$$

$$\text{Compute } d = \hat{\mathbb{E}}_t [\text{KL}[\pi_{\theta_{\text{old}}}(\cdot | s_t), \pi_{\theta}(\cdot | s_t)]]$$

- If $d < d_{\text{targ}}/1.5$, $\beta \leftarrow \beta/2$
- If $d > d_{\text{targ}} \times 1.5$, $\beta \leftarrow \beta \times 2$

24

- Soft constraint
 - Trade-off: reward monotonous and policy randomness
- Other tricks to make training more stable and better sample efficiency
 - Importance sampling
 - GAE

PPO – Generalized Advantage Estimation (GAE)

- Key idea: balances bias and variance in advantage estimation by combining multiple truncated advantage estimates.

- $\hat{A}_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t) \rightarrow$ High bias, low variance.
- $\hat{A}_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) \rightarrow$ Moderate bias/variance.
- $\hat{A}_t^{(\infty)} = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V(s_t) \rightarrow$ Unbiased, high variance.

- Combines these estimates with weights with $\omega_k = \lambda^{k-1}$

$$\hat{A}_t^{\text{GAE}} = \frac{\sum_k w_k \hat{A}_t^{(k)}}{\sum_k w_k} \quad \text{equals to} \quad \hat{A}_t^{\text{GAE}} = \delta_t + \gamma \lambda \delta_{t+1} + \dots + (\gamma \lambda)^{T-t+1} \delta_{T-1}$$

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

- Schulman, John, et al. "High-dimensional continuous control using generalized advantage estimation." arXiv preprint arXiv:1506.02438. 25

- implementation details

PPO Algorithm

Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go \hat{R}_t .
- 5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T (V_{\phi}(s_t) - \hat{R}_t)^2,$$

typically via some gradient descent algorithm.


- 8: **end for**

Usually use GAE here

$$g(\epsilon, A) = \begin{cases} (1 + \epsilon)A & A \geq 0 \\ (1 - \epsilon)A & A < 0 \end{cases}$$

Value update

PPO – Update tricks

- 
- 1, sample action-state pairs $\{s_t, a_t, s_{t+1}\} \rightarrow B$ from π_θ
 - 2, update $V_\phi, \pi_{\theta'}$ **for K epochs**, each epoch use all of data in B
 - 3, $\theta' \rightarrow \theta$

- In practice, K shouldn't be too large, otherwise the distribution mismatch would be large.
- Mini-batch: In each epoch, we divide buffer into several mini-batches (similar to batch gradient decent).
 - Too large mini-batch: stuck in local minima
 - Too small mini-batch: noisy gradient

28

PPO – Normalization / Scaling

- Advantage normalization: $A_t^{norm} = \frac{A_t - \mu_A}{\sqrt{\sigma_A^2 + \epsilon}}$
 - Only used in training time
 - Normalize in a minibatch
 - When the value function is not well trained, it helps A_t center around 0.
- State normalization : $s_t^{norm} = \frac{s_t - \mu_{run}}{\sqrt{\sigma_{run}^2 + \epsilon}}$
 - Need to restore the running mean of states and use it in test time.
 - Similar to Batch-Norm, it helps with training.
- Reward scaling : $r_t^{scale} = \frac{r_t}{\sqrt{\sigma_{run}^2 + \epsilon}}$
 - Scaled reward makes value function $V_\phi(s_t)$ training easier.

30

PPO – Initialization & Activation

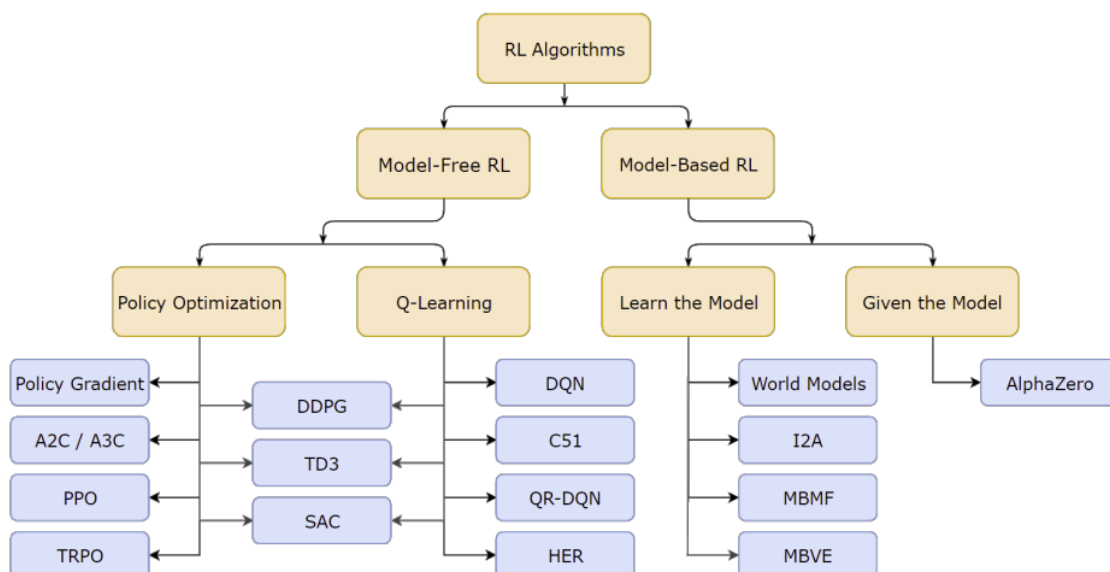
- Orthogonal Initialization
 - Step 1: Gaussian distribution($\mu = 0, \sigma = 1$) initialize W .
 - Step 2: Apply SVD, $W = U^T D V$, $V \rightarrow W$. W is an orthogonal matrix.
 - [Engstrom, Ilyas, et al., \(2020\)](#) find orthogonal initialization to outperform the default Xavier initialization.
 - [Andrychowicz, et al. \(2021\)](#) find centering the action distribution around 0 (i.e., initialize the policy output layer weights with 0.01”) to be beneficial
- Tanh activation
 - use Tanh activation instead of ReLU (PPO original paper)
 - only 2 or 3 hidden layer MLP, gradient vanishing may not have significant damage.

31

- pros
 - better sample efficiency
 - Importance sampling, minibatch
 - more stable training and better performance
 - Clipped Surrogate Objective, Adaptive KL, other implemental tricks
- cons:
 - Still poor sample efficiency compared with off-policy RL algorithm
 - Need to train in simulator, has domain gap to real-world environment
 - many task-specific tricks
 - Carefully choose hyperparameters

Summary of RL

A Taxonomy of RL Algorithms



ref: <https://spinningup.openai.com/en/latest/index.html>