Advanced Policy Gradient

Contents

- A3C
- MaxEnt RL
- TRPO
- PPO
- DDPG
- TD3

A3C(Asynchronous Advantage Actor Critic)

- Actor-Critic method is an effective, but online learning method
- Not easy to generate many data
- Not easy to train in batch
- Experience replay is one solution, but sometime memory becomes too big.
- Another method of reducing dependency in data is A3C
- Another advantage of A3C is it can be done in parallel.

A3C

- **Asynchronous** stands for the principal difference of this algorithm from DQN, where a single neural network interacts with a single environment.
- In A3C, we’ve got a global network with multiple agents having their own set of parameters.
- It creates every agent’s situation interacting with its environment and harvesting the different and unique learning experience for overall training.
- That deals partially with sample correlation, a big problem for neural networks, which are optimized under the assumption that input samples are independent of each other
- Asynchronous: the algorithm involves executing a set of environments in parallel to increase the diversity of training data, and with gradient updates performed in a Hogwild! style procedure.
- No experience replay is needed, though one could add it if desired.

A3C

- A3C idea: Sample for data can be parallelized using several copies of the same agent
  - Use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - After some time, pass gradients to a global network that updates actor and critic using the gradients of all agents
  - After some time the worker copy the weights of the global network

- This parallelism decorrelates the agents' data, so no Experience Replay Buffer needed

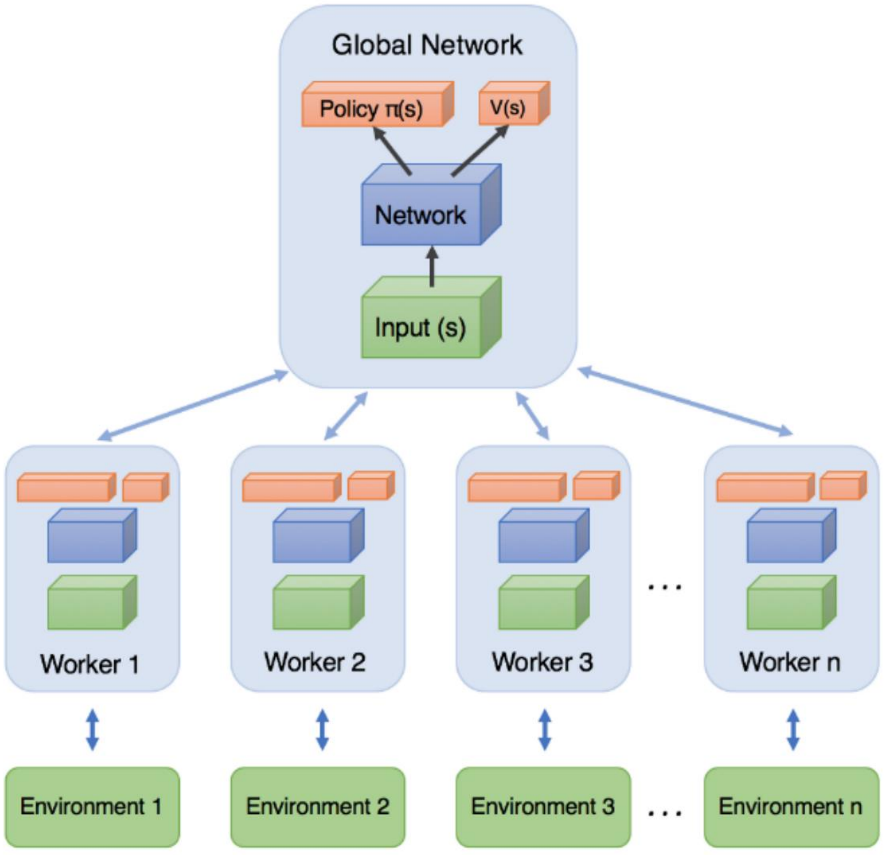
- Even one can explicitly use different exploration policies in each actor-learner to maximize diversity

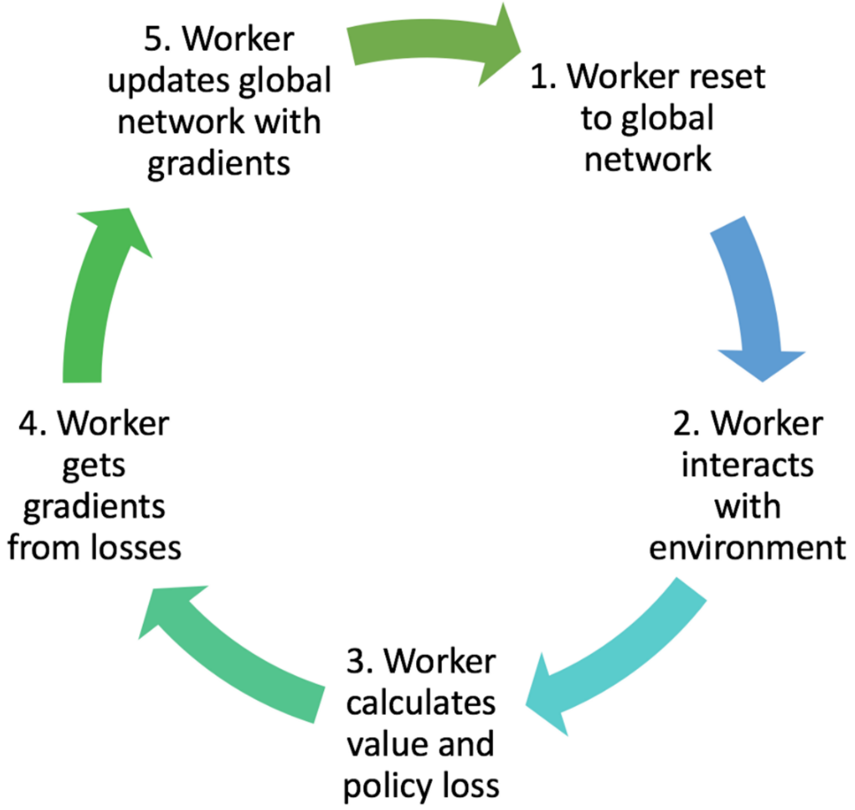
- Asynchronism can be extended to other update mechanisms (SARSA, Q-learning, etc) but it works better in Advantage Actor Critic setting

A3C

- A3C consists of **multiple independent workers**(agents, networks) with their own weights, who interact with a different copy of the environment in parallel.
- Thus, they can explore a bigger part of the state-action space in much less time.
- The agents (or workers) are trained in parallel and update periodically a global network, which holds shared parameters.
- The updates are not happening simultaneously and that’s where the asynchronous comes from.
- After each update, the agents resets their parameters to those of the global network and continue their independent exploration and training for n steps until they update themselves again.

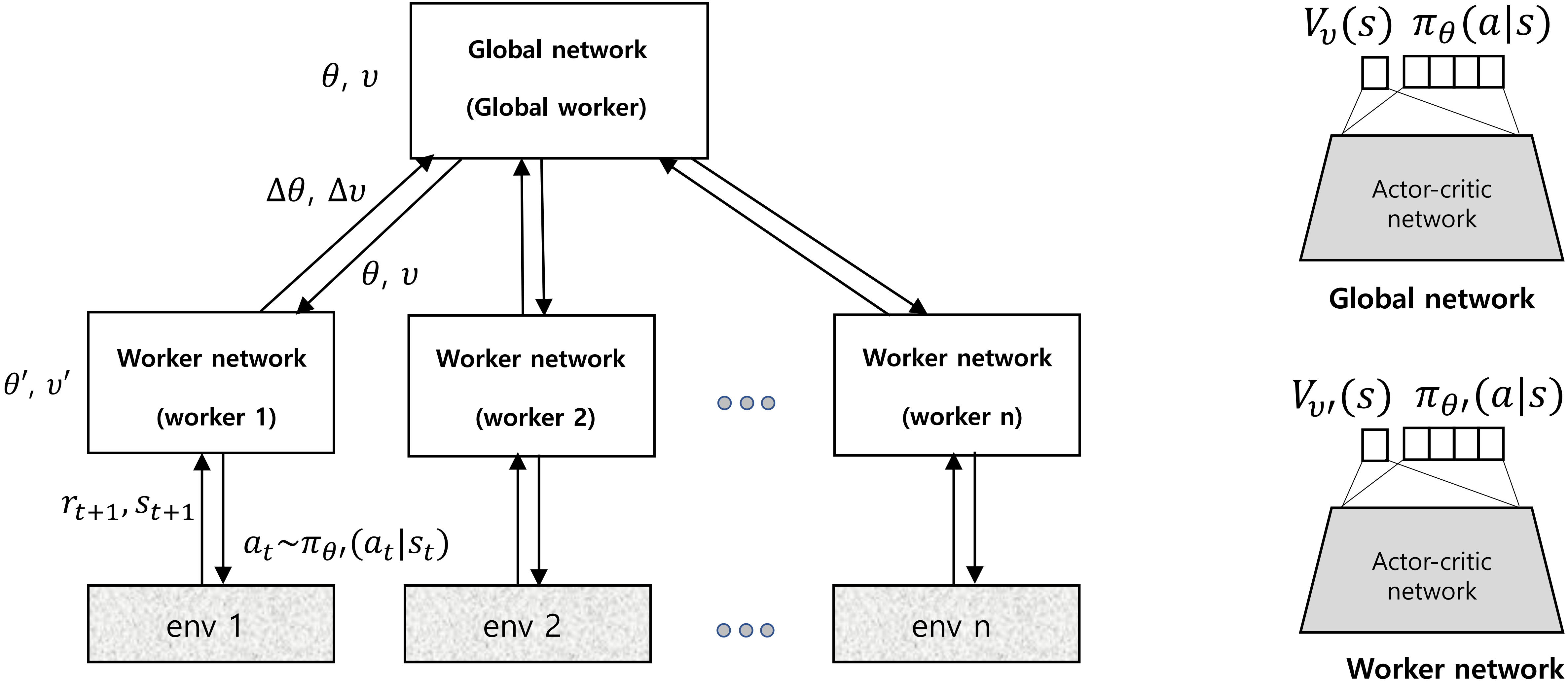
A3C





https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2

A3C


Global network
(Global worker)
$\theta,\ \upsilon$
$\Delta\theta,\ \Delta\upsilon$
$\theta,\ \upsilon$
$\theta',\ \upsilon'$
Worker network
(worker 1)
Worker network
(worker 2)
Worker network
(worker n)
r_{t+1}, s_{t+1}
$a_t \sim \pi_{\theta'}(a_t|s_t)$
env 1
env 2
env n
$V_\upsilon(s)\ \ \pi_\theta(a|s)$
Actor-critic network
Global network
$V_{\upsilon'}(s)\ \ \pi_{\theta'}(a|s)$
Actor-critic network
Worker network

Actor Network Update

- Recap: Advantage Actor-Critic (p. 41 in Chapter 9)

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, \delta^{\pi_\theta}]$$
$$\text{where } \delta^{\pi_\theta} = r + \gamma V_{\pi_\theta}(s') - V_{\pi_\theta}(s)$$

- Therefore, each worker (agent) in A3C computes policy gradient as follows

$$\nabla_{\theta'} \log \pi_{\theta'}(a_t|s_t)\, \delta_t \text{ where } \delta_t = r_{t+1} + \gamma v_{\upsilon'}(s_{t+1}) - v_{\upsilon'}(s_t)$$

  - θ' and υ' : parameters of policy network and value network of workers, respectively

- However, A3C uses n-step return (instead of single-step advantage δ^{π_θ})
  - n-step return in state s_t : $G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n v_{\upsilon'}(s_{t+n})$
  - n-step return has the following property

$$G_t^{(n)} = r_{t+1} + \gamma G_{t+1}^{(n-1)}$$

- Using $G_t^{(n)}$, A3C worker agent computes policy gradient as follows

$$\nabla_{\theta'} \log \pi_{\theta'}(a_t|s_t)\, \delta_t^{(n)} \text{ where } \delta_t^{(n)} = G_t^{(n)} - v_{\upsilon'}(s_t)$$

Actor Network Update

- A3C uses a trick to compute policy gradient efficiently
- Worker moves from state s_t to s_{t+n} with the following n actions (A3C uses n-step return)

$$(a_t, a_{t+1}, \dots, a_{t+n-1})$$

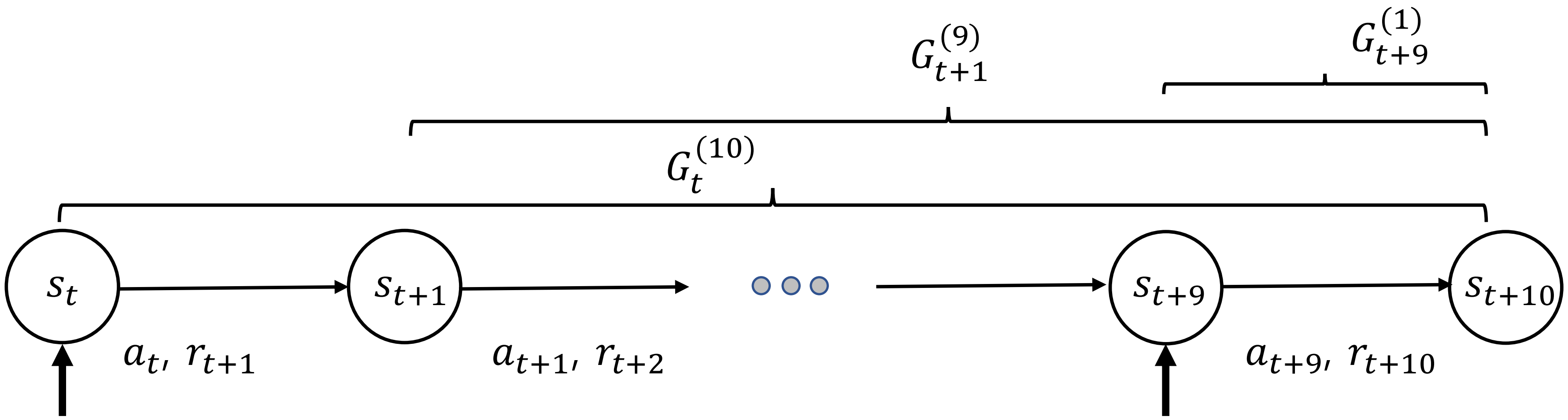
- In each of intermediate state s_{t+k} $(0 \le k \le n-1)$, it computes $G_{t+k}^{(n-k)}$
  - $G_{t+k}^{(n-k)}$: value of $(n-k)$-step return at (intermediate) state s_{t+k}
- For each intermediate state, agent computes policy gradient
  - $\Delta\theta$ is the summation of these gradients, defined as follows

$$\Delta\theta = \sum_{k=0}^{n-1} \nabla_{\theta'} \log \pi_{\theta'}(a_{t+k}|s_{t+k}) \left(G_{t+k}^{(n-k)} - v_{v'}(s_{t+k})\right)$$

- Above $\Delta\theta$ is forwarded to global worker to update global network
- After that, worker receives new parameters from global network and starts over

- Each worker works on a little different parameters from global networks.
- As long as learning constant is small, it works fine (*HogWild update)*

Computing Policy Gradients in Intermediate States



$$\Delta\theta = \sum \Big[\ \nabla_{\theta'} \log \pi_{\theta'}(a_t|s_t)\left(G_t^{(10)} - v_{v'}(s_t)\right) \quad \cdots \quad \nabla_{\theta'} \log \pi_{\theta'}(a_{t+9}|s_{t+9})\left(G_{t+9}^{(1)} - v_{v'}(s_{t+9})\right)$$

$$\Delta\theta = \sum_{k=0}^{9} \nabla_{\theta'} \log \pi_{\theta'}(a_{t+k}|s_{t+k})\left(G_{t+k}^{(10-k)} - v_{v'}(s_{t+k})\right) =$$

$$\nabla_{\theta'} \log \pi_{\theta'}(a_t|s_t)\left(r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^9 r_{t+10} + \gamma^{10} v_{v'}(s_{t+10}) - v_{v'}(s_t)\right) +$$

$$\cdots$$

$$\nabla_{\theta'} \log \pi_{\theta'}(a_{t+9}|s_{t+9})\left(r_{t+10} + \gamma v_{v'}(s_{t+10}) - v_{v'}(s_{t+9})\right)$$

Critic Network Update

- Worker should update critic network as well
- Error function of critic network (with single-step TD(0)) is given as follows (using MSE)

$$E_{\pi_{\theta'}}\left[\left(r_{t+1} + \gamma v_{v'}(s_{t+1}) - v_{v'}(s_t)\right)^2\right]$$

- Previously, worker computed multi-step return in each intermediate step
- We reuse these multi-step returns in critic network learning
- Therefore, in time step $t + k$, error function is defined as

$$\left(G_{t+k}^{(n-k)} - v_{v'}(s_{t+k})\right)^2$$

- Therefore, the total error function of critic network is

$$J(v') = \sum_{k=0}^{n-1} \left(G_{t+k}^{(n-k)} - v_{v'}(s_{t+k})\right)^2$$

- Its derivative is

$$\nabla_{v'} J(v') = \sum_{k=0}^{n-1} \nabla_{v'} \left(G_{t+k}^{(n-k)} - v_{v'}(s_{t+k})\right)^2$$

A3C

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state s_t
    repeat
        Perform a_t according to policy $\pi(a_t|s_t;\theta')$
        Receive reward r_t and new state s_{t+1}
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal s_t **or** $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
    for $i \in \{t-1, \ldots, t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i;\theta'_v))^2/\partial\theta'_v$
    end for
    Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

$$G_t^{(n)} = r_{t+1} + \gamma G_{t+1}^{(n-1)}$$

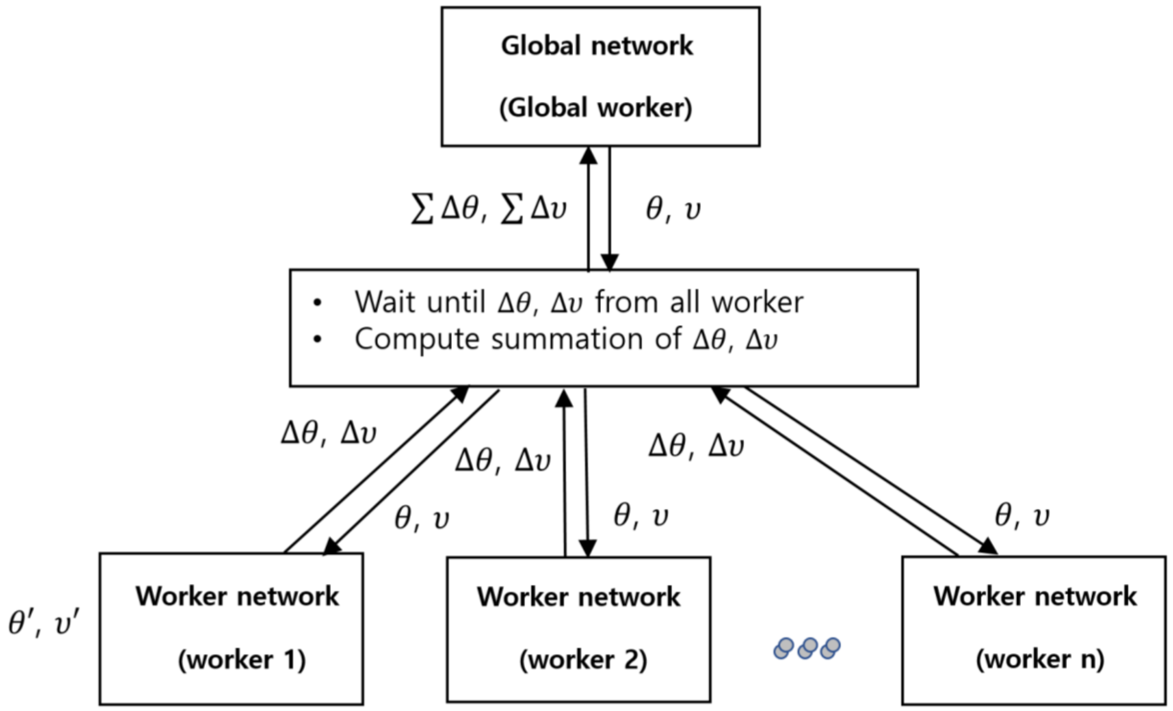
A2C(Advantage Actor Critic)

- In A3C, some agents will be playing with an older version of the parameters.

- Of course, the update may not happen asynchronously but at the same time.

- In that case, we have an improved version of A2C with multiple agents instead of one.

- A2C will wait for all the agents to finish their segment and then update the global network weights and reset all the agents.



MaxEnt RL

- In Maximum Entropy RL, the aim is to learn the optimal policy that can achieve the highest cumulative reward and maximum entropy.
- In Maximum Entropy RL, it enables more exploration and chances to avoid converging to local optima is higher.

- Original objective function

$$E_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

changes to

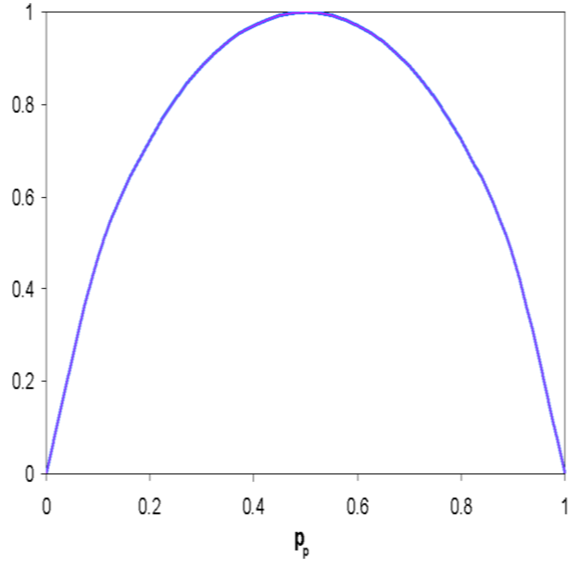
$$J(\theta) = E_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) + \beta H(\pi_\theta(\cdot\,|s_t))\right]$$

MaxEnt RL

- Why add an entropy regularizer?
- Why is maximizing entropy desirable in RL?

$$H(x) = -\sum_{x} p(x) \log p(x)$$

$$H(\pi_\theta) = E_{a \sim \pi_\theta}[-\log \pi_\theta(a|s)] = -\sum_{a} \pi_\theta(a|s) \log \pi_\theta(a|s)$$



Trust Region Policy Optimization(TRPO)

- In supervised learning, even if the parameter moves to a very bad state, it can recover from the learning of the next batch. (Each data is independent of each other by IID)
- However, policy gradient is basically an online learning method.
- Therefore, if you make a mistake in updating the parameters and move to a very bad state (such as setting the wrong step size), learning often becomes almost impossible from the next time.

- There are considerations when defining the confidence interval: 1) whether to define a confidence interval for the change difference in the parameter, 2) or a confidence interval for the change difference in the policy (as determined by the parameter)
- First, defining a confidence interval for a change in parameter values is simple to calculate the confidence interval, but the change in the objective function often changes significantly even with a slight change in the parameter value.
- On the other hand, in general, compared to changes in policy, the objective function changes relatively smoothly.

Motivation – Problem in REINFORCE

Initialize policy network π_θ
loop
    Generate episodes $\tau \sim s_0, a_0, r_1, \dots, s_{n-1}, a_{n-1}, r_n$ with π_θ
    $\Delta\theta = 0$
    for every step $t = 0,1,2,\dots,n-1$ in episode **do**

$$G_t = \sum_{i=0}^{n-t} \gamma^i r_{t+i}$$

$$\Delta\theta \leftarrow \Delta\theta + \gamma^t \nabla_\theta \log \pi_\theta(s_t, a_t)\, G_t$$

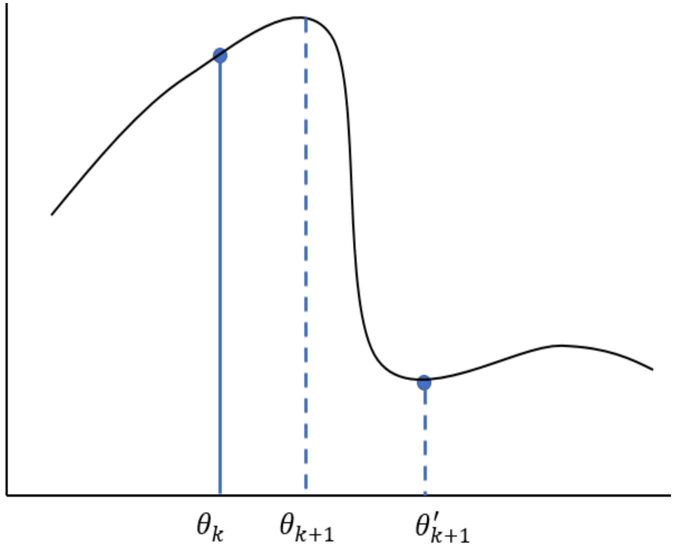
    end for
$\theta \leftarrow \theta + \alpha\Delta\theta$
end loop
return π_θ

- How to choose the step size(α)?
  - too large? 1) bad policy -> 2) collected data under bad policy
  - too small? cannot leverage data sufficiently
- Can’r recover!

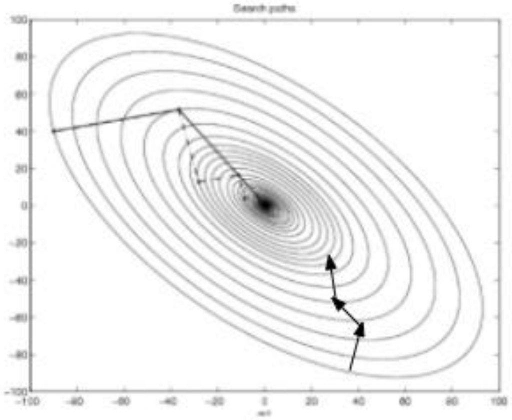
Why Trust Region?

Line search
(like gradient ascent)

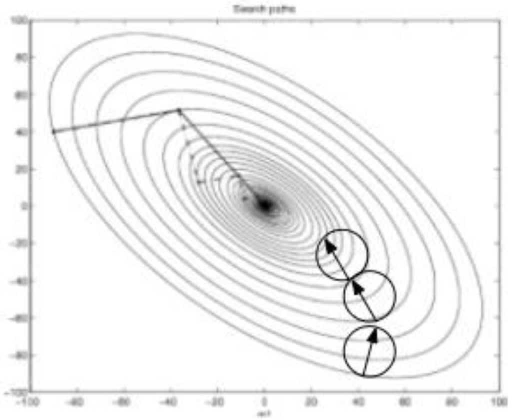
Trust region



θ_k -> θ'_{k+1}
(Once you fail, you can't recover)



LINE SEARCH METHOD



TRUST REGION METHOD

Instead of following the derivative,
1) define a trust(safe) region at current point and
2) move to the optimal point within the trust region

TRPO

- MDP model $< S, A, P, R, \rho_0, \gamma >$ where ρ_0 is prob. of starting states
  - $\eta(\pi)$: objective function of policy π
  - θ and $\tilde{\theta}$: current parameter and new parameter of policy network, respectively
  - π, π_θ: current policy
  - $\tilde{\pi}$, $\pi_{\tilde{\theta}}$: new policy

- Objective function

$$\eta(\theta) = E_{s_0, a_0, \ldots \sim \pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right] \qquad s_0 \sim \rho_0(s_0),\ a_t \sim \pi(a_t|s_t),\ s_{t+1} \sim P(s_{t+1}|s_t, a_t)$$

- Advantage Policy Gradient

$$\nabla_\theta \eta(\theta) = E_{s_0, a_0, \ldots \sim \pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) A_{\pi_\theta}(s_t, a_t)\right]$$

TRPO

- Advantage Policy Gradient

$$\nabla_\theta \eta(\theta) = E_{s_0,a_0,\ldots\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) A_{\pi_\theta}(s_t,a_t)\right]$$

  1) online learning: difficult to increase the number of data
  2) small change in parameter may cause big change in policy

- Define a region of parameter change that can limit the change in policy
  - Trust Region

- **Theorem:** Improvement of new policy over current policy

$$\eta(\tilde{\theta}) = \eta(\theta) + E_{s_0,a_0,\ldots\sim\pi_{\tilde{\theta}}}\left[\sum_{t=0}^{\infty}\gamma^t A_{\pi_\theta}(s_t,a_t)\right]$$

(*) Proof

Since $A_{\pi_\theta}(s_t, a_t) = r(s_t, a_t) + \gamma V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t)$

$$E_{s_0,a_0,\ldots\sim\pi_{\tilde{\theta}}}\left[\sum_{t=0}^{\infty}\gamma^t A_{\pi_\theta}(s_t, a_t)\right]$$

$$= E_{s_0,a_0,\ldots\sim\pi_{\tilde{\theta}}}\left[\sum_{t=0}^{\infty}\gamma^t \left(r(s_t, a_t) + \gamma V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t)\right)\right]$$

$$= \eta\left(\tilde{\theta}\right) + E_{s_0,a_0,\ldots\sim\pi_{\tilde{\theta}}}\left[\sum_{t=0}^{\infty}\gamma^{t+1} V_{\pi_\theta}(s_{t+1}) - \sum_{t=0}^{\infty}\gamma^t V_{\pi_\theta}(s_t)\right]$$

$$= \eta\left(\tilde{\theta}\right) + E_{s_0,a_0,\ldots\sim\pi_{\tilde{\theta}}}\left[\sum_{t=1}^{\infty}\gamma^t V_{\pi_\theta}(s_t) - \sum_{t=0}^{\infty}\gamma^t V_{\pi_\theta}(s_t)\right]$$

$$= \eta\left(\tilde{\theta}\right) - E_{s_0,a_0,\ldots\sim\pi_{\tilde{\theta}}}\left[V_{\pi_\theta}(s_0)\right] = \eta\left(\tilde{\theta}\right) - \eta(\theta)$$

TRPO

- Due to the theorem, objective function is now

$$E_{s_0, a_0, \ldots \sim \pi_{\tilde{\theta}}}\left[\sum_{t=0}^{\infty} \gamma^t A_{\pi_\theta}(s_t, a_t)\right]$$

- Above objective function is evaluated using data from new policy $\pi_{\tilde{\theta}}$

- Can we somehow use the data generated from previous policy π_θ?
  - how to change $\pi_{\tilde{\theta}}$ to π_θ

Discounted Visitation Frequency

- Discounted visitation frequency: the probability of agent staying state s using policy π_θ

$$\rho_{\pi_\theta}(s) = P_{\pi_\theta}(s_0 = s) + \gamma P_{\pi_\theta}(s_1 = s) + \gamma^2 P_{\pi_\theta}(s_2 = s) + \cdots = \sum_{t=0}^{\infty} \gamma^t P_{\pi_\theta}(s_t = s)$$

- Objective function is redefined using this frequency

$$J(\tilde{\theta}) = \eta(\tilde{\theta}) - \eta(\theta) = \sum_s \rho_{\pi_{\tilde{\theta}}}(s) \sum_a \pi_{\tilde{\theta}}(a|s) A_{\pi_\theta}(s,a)$$

- Can define this function using data from old policy π_θ?
- We can assume $\rho_{\pi_{\tilde{\theta}}}(s) \approx \rho_{\pi_\theta}(s)$
- Therefore,

$$L(\tilde{\theta}) = \sum_s \rho_{\pi_\theta}(s) \sum_a \pi_{\tilde{\theta}}(a|s) A_{\pi_\theta}(s,a)$$

- Now

$$L(\tilde{\theta}) = \sum_s \rho_{\pi_\theta}(s) \sum_a \pi_{\tilde{\theta}}(a|s) A_{\pi_\theta}(s,a)$$

$$= \sum_s \rho_{\pi_\theta}(s) \sum_a \pi_\theta(a|s) \left[\frac{\pi_{\tilde{\theta}}(a|s)}{\pi_\theta(a|s)} A_{\pi_\theta}(s,a) \right] = E_{\pi_\theta}\left[\frac{\pi_{\tilde{\theta}}(a|s)}{\pi_\theta(a|s)} A_{\pi_\theta}(s,a) \right]$$

(*) Importance Sampling Perspective

- We know

$$E_{X\sim P}[f(X)] = \sum P(X)f(X) = \sum Q(X)\frac{P(X)}{Q(X)}f(X) = E_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]$$

- From

$$J(\tilde{\theta}) = \sum_s \rho_{\pi_{\tilde{\theta}}}(s)\sum_a \pi_{\tilde{\theta}}(a|s)A_{\pi_\theta}(s,a) = E_{\pi_{\tilde{\theta}}}[A_{\pi_\theta}(s,a)]$$

- Substituting

$P(X)$-> $\pi_{\tilde{\theta}}(s|a)$, $Q(X)$-> $\pi_\theta(s|a)$, $f(X)$-> $A_{\pi_\theta}(s,a)$

- We have

$$J(\tilde{\theta}) = E_{\pi_{\tilde{\theta}}}[A_{\pi_\theta}(s,a)] = E_{\pi_\theta}\left[\frac{\pi_{\tilde{\theta}}(a|s)}{\pi_\theta(a|s)}A_{\pi_\theta}(s,a)\right]$$

$$= \sum_s \rho_{\pi_\theta}(s)\sum_a \pi_\theta(a|s)\left[\frac{\pi_{\tilde{\theta}}(a|s)}{\pi_\theta(a|s)}A_{\pi_\theta}(s,a)\right]$$

Trust Region

- However $E_{\pi_\theta}\left[\frac{\pi_{\tilde{\theta}}(a|s)}{\pi_\theta(a|s)}A_{\pi_\theta}(s,a)\right]$ has high variance

- We know that $Var[Z] = E\left[Z^2\right] - E[Z]^2$
- Let $\mathrm{Z} = \frac{P(X)}{Q(X)}f(X)$, then

$$Var[Z] = E_{X\sim Q}\left[\left(\frac{P(X)}{Q(X)}f(X)\right)^2\right] - E_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]^2 = E_{X\sim P}\left[\frac{P(X)}{Q(X)}f(X)^2\right] - E_{X\sim P}[f(X)]^2$$

- Let $P(X) \Rightarrow \pi_{\tilde{\theta}}(a|s)$, $Q(X) \Rightarrow \pi_\theta(a|s)$, $f(X) \Rightarrow A_{\pi_\theta}(s,a)$

- Therefore, big changes in policy causes high variance

$$\frac{P(X)}{Q(X)} = \frac{\pi_{\tilde{\theta}}(a|s)}{\pi_\theta(a|s)}$$

Trust Region

- Difference between policies should be small

- We define a Trust region that is based on difference between two policies
  - We use Kullback-Leibler(KL) divergence

- Kullback-Leibler(KL) divergence between two probabilities $p(x)$ and $q(x)$

$$D_{KL}(p||q) = E_{p(x)}\left[\log\frac{p(x)}{q(x)}\right] = \sum_x p(x)\log\frac{p(x)}{q(x)}$$

- Difference between two policies using KL divergence

$$D_{KL}\left(\pi_\theta(\cdot\,|s)||\pi_{\tilde{\theta}}(\cdot\,|s)\right) = E_{\pi_\theta}\left[\log\frac{\pi_\theta(a|s)}{\pi_{\tilde{\theta}}(a|s)}\right] = \sum_a \pi_\theta(a|s)\log\frac{\pi_\theta(a|s)}{\pi_{\tilde{\theta}}(a|s)}$$

Trust Region

- Therefore new parameter $\tilde{\theta}$ should satisfy the following condition

$$\max_{s} D_{KL}\left(\pi_{\theta}(\cdot\,|s)||\pi_{\tilde{\theta}}(\cdot\,|s)\right) \leq \delta$$

- This condition is too strict. Therefore we normally use average value

$$E_{\pi_{\theta}}\left[D_{KL}\left(\pi_{\theta}(\cdot\,|s)||\pi_{\tilde{\theta}}(\cdot\,|s)\right)\right] \leq \delta$$

- Finally TRPO objective function with restriction is as follows

$$\underset{\tilde{\theta}}{\operatorname{argmax}}\, E_{\pi_{\theta}}\left[\frac{\pi_{\tilde{\theta}}(a|s)}{\pi_{\theta}(a|s)} A_{\pi_{\theta}}(s,a)\right]$$

$$\text{subject to } E_{\pi_{\theta}}\left[D_{KL}\left(\pi_{\theta}(\cdot\,|s)||\pi_{\tilde{\theta}}(\cdot\,|s)\right)\right] \leq \delta$$

Computing TRPO

- Computing $\tilde{\theta}$ is not easy
- We use approximate function for objective function($L(\tilde{\theta})$) and trust region($l(\tilde{\theta})$)
- Functions are approximated using Taylor Series:

$$\tilde{f}(x) = f(c) + \nabla f(c)^T(x-c) + \frac{1}{2!}(x-c)^T H(x-c)$$

- Let $L(\tilde{\theta}) = E_{\pi_\theta}\left[\frac{\pi_{\tilde{\theta}}(a|s)}{\pi_\theta(a|s)} A_{\pi_\theta}(s,a)\right]$
- Using Taylor Series, $L(\tilde{\theta})$ is approximated as follows

$$L(\tilde{\theta}) \approx L(\theta) + \nabla_{\tilde{\theta}} L(\theta)^T(\tilde{\theta} - \theta)$$

$$L(\tilde{\theta}) \approx \nabla_{\tilde{\theta}} L(\theta)^T(\tilde{\theta} - \theta)$$

Computing TRPO

- Now define approximate function for trust region
- Let $l(\tilde{\theta}) = E_{\pi_\theta}\left[D_{KL}(\pi_\theta(\cdot|s)||\pi_{\tilde{\theta}}(\cdot|s))\right] = E_{\pi_\theta}\left[\sum_a \pi_\theta(a|s)\log\frac{\pi_\theta(a|s)}{\pi_{\tilde{\theta}}(a|s)}\right]$
- Using Taylor Series

$$l(\tilde{\theta}) \approx l(\theta) + \nabla_{\tilde{\theta}} l(\theta)^T(\tilde{\theta}-\theta) + \frac{1}{2}(\tilde{\theta}-\theta)^T \nabla_{\tilde{\theta}}^2 l(\theta)(\tilde{\theta}-\theta)$$

- We know that

1) $l(\theta) = E_{\pi_\theta}\left[D_{KL}(\pi_\theta(\cdot|s)||\pi_\theta(\cdot|s))\right] = 0$

2) **(*)** $\nabla_{\tilde{\theta}} l(\theta) = \nabla_{\tilde{\theta}} D_{KL}(\pi_\theta(\cdot|s)||\pi_{\tilde{\theta}}(\cdot|s))\Big|_{\tilde{\theta}=\theta} = \nabla_{\tilde{\theta}} \sum_a \pi_\theta(a|s)\log\frac{\pi_\theta(a|s)}{\pi_{\tilde{\theta}}(a|s)}\Bigg|_{\tilde{\theta}=\theta}$

$$= -\sum_a \nabla_{\tilde{\theta}} \pi_{\tilde{\theta}}(a|s) = -\nabla_{\tilde{\theta}} \sum_a \pi_{\tilde{\theta}}(a|s) = 0$$

- Due to 1) and 2), approximation function of $l(\tilde{\theta})$ is

$$l(\tilde{\theta}) \approx \frac{1}{2}(\tilde{\theta}-\theta)^T \nabla_{\tilde{\theta}}^2 l(\theta)(\tilde{\theta}-\theta)$$

Computing TRPO

- Solving in terms of $\tilde{\theta}$ (H: Hessian matrix of $D_{KL}\left(\pi_\theta(\cdot\,|s)||\pi_{\tilde{\theta}}(\cdot\,|s)\right)$

$$\tilde{\theta} = \theta + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

  - This is called natural policy gradient

- Above is obtained based on the approximate function using the Taylor series.
- Accordingly, the range of variation of $\tilde{\theta}$ value may deviate from the trust region

- When $\tilde{\theta}$ value is out of trust region, the j value is increased by one so that the policy change by $\tilde{\theta}$ is within trust region ($x \approx H^{-1} g$)

$$\tilde{\theta} = \theta + \alpha^j \sqrt{\frac{2\delta}{x^T H x}} x$$

  - This is called backtracking line search

Proximal Policy Optimization(PPO)

- TRPO needs a lot of computation
  - Computing Hessian Matrix and inverse matrix
- PPO: Approximate method of TRPO

PPO Adaptive KL Penalty

Method 1)

- TRPO problem is given as

$$\underset{\tilde{\theta}}{\operatorname{argmax}}\, E_{\pi_\theta}\left[\frac{\pi_{\tilde{\theta}}(a|s)}{\pi_\theta(a|s)} A_{\pi_\theta}(s,a)\right] \text{ subject to } E_{\pi_\theta}\left[D_{KL}\left(\pi_\theta(\cdot\,|s)||\pi_{\tilde{\theta}}(\cdot\,|s)\right)\right] \le \delta$$

- Using Lagrangian dual, objective function is transformed as

$$\underset{\tilde{\theta}}{\operatorname{argmax}}\, E_{\pi_\theta}\left[\frac{\pi_{\tilde{\theta}}(a|s)}{\pi_\theta(a|s)} A_{\pi_\theta}(s,a)\right] - \beta\left(E_{\pi_\theta}\left[D_{KL}\left(\pi_\theta(\cdot\,|s)||\pi_{\tilde{\theta}}(\cdot\,|s)\right)\right]\right)$$

PPO

Method 2)

Another way of computing β

- Let

  $$d = E_{\pi_\theta}\left[D_{KL}\left(\pi_\theta(\cdot\,|s), \pi_{\tilde{\theta}}(\cdot\,|s)\right)\right]$$

- Adjust β using the following

  $$\beta = \begin{cases} \beta/2, & d < \dfrac{\delta}{1.5} \\ \beta * 2, & d > \delta * 1.5 \end{cases}$$

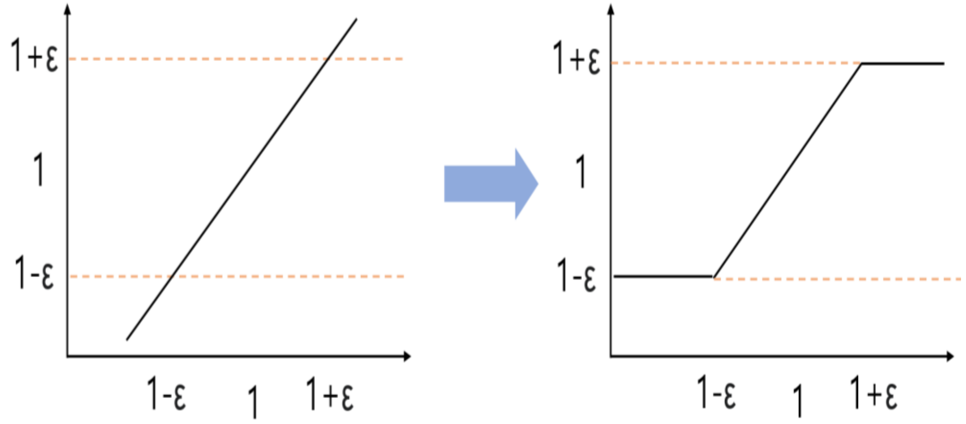
PPO

PPO with Clipped Objective

- If the ratio of the two policies exceeds a certain range, clip the value and forcing it to be a small value

- Clip function is defined as

$$clip(x, 1-\epsilon, 1+\epsilon) = \begin{cases} 1-\epsilon, & \text{if } x < 1-\epsilon \\ x, & \text{if } 1+\epsilon \leq x \leq 1-\epsilon \\ 1+\epsilon, & \text{if } x > 1+\epsilon \end{cases}$$



- Object function of PPO is given as

$$\mathcal{L}^{\text{CLIP}}(\tilde{\theta}) = E_{\pi_\theta}\left[\min\left(r_t(\tilde{\theta}), clip\left(r_t(\tilde{\theta}), 1-\epsilon, 1+\epsilon\right)\right) A_{\pi_\theta}(s_t, a_t)\right]$$

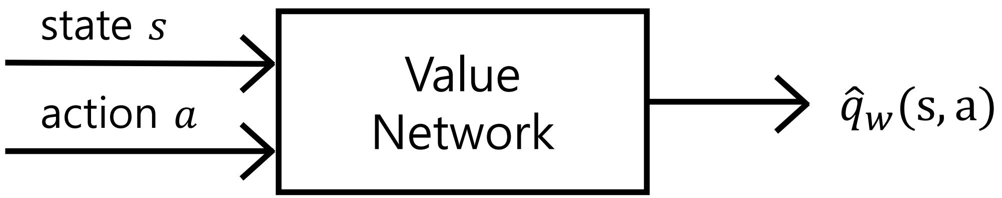
where $r_t(\tilde{\theta}) = \dfrac{\pi_{\tilde{\theta}}(a_t|s_t)}{\pi_\theta(a_t|s_t)}$

DDPG(Deep Deterministic Policy Gradient)

- Computing the maximum over actions in the target is impossible in continuous action spaces.
- DDPG deals with this by using a **target policy network** to compute an action which approximately maximizes
- The target policy network is found the same way as the target Q-function:
- DPG(Deterministic policy gradient): computing gradient on continuous action
- DDPG(Deep deterministic policy gradient): improved DPG using DQN techniques and others

DDPG

- Value network (DQN)



- Error function of value network is (w: parameter of value network)

$$J(w) = E_{s,a,r,s'}\left(r + \gamma \max_{a'} \hat{q}_w(s', a') - \hat{q}_w(s, a)\right)^2$$

- When using target value network in DQN (w': parameter of target network)

$$J(w) = E_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} \hat{q}_{w'}(s', a') - \hat{q}_w(s, a)\right)^2\right]$$

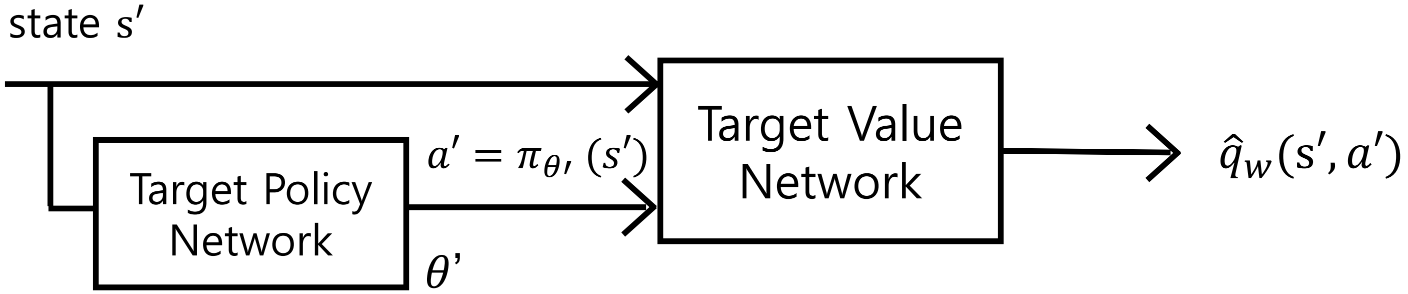
- Since actions are continuous, $\max_{a'} \hat{q}_{w'}(s', a')$ is not computable

DDPG

- In DDPG target network, finding best a' in $\max_{a'} \hat{q}_{w'}(s', a')$ is done by target policy network
- Target policy network: A new policy network that finds the optimal behavior a' from the state s'

- The target policy network is a policy network that outputs an action a' that maximizes the Q value for a particular state input s (θ' is parameter of target policy network): $a' = \pi_{\theta'}(s)$

- Now, in target value network, $\max_{a'} \hat{q}_{w'}(s', a')$ is replaced by $\hat{q}_{w'}(s', \pi_{\theta'}(s'))$



Training Value Network

- Error function of value network is defined as ($\max_{a'} \hat{q}_{w'}(s', a')$ -> $\hat{q}_{w'}(s', \pi_{\theta'}(s'))$)

$$J(w) = E_{s,a,r,s'\sim D}\left[\left(r + \gamma \underbrace{\hat{q}_{w'}(s', \underbrace{\pi_{\theta'}(s')}_{\text{target policy network}})}_{\text{target value network}} - \underbrace{\hat{q}_w(s,a)}_{\text{value network}}\right)^2\right]$$

- Differentiate $\hat{q}_w(s, a)$ only (semi-gradient)

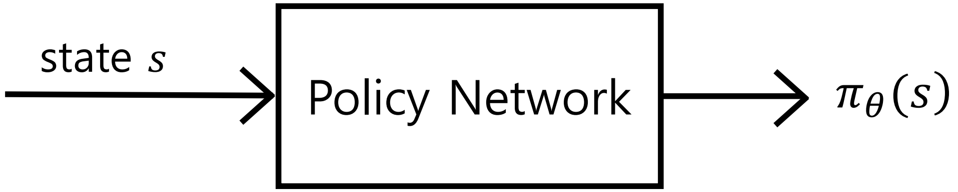
- Update w using

$$w = w - \alpha \nabla_w J(w)$$

- Periodically update target network parameter with value network
  - Or Polyak update

Training Policy Network

- Now train policy network
- Since actions are continuous, there is only one output in policy network



  - θ: parameter of policy network
  - handles deterministic policy only

- The learning of the policy network: learn the parameter θ of the policy network so that Q value of $(s, \pi_\theta(s))$ is maximized.

- Therefore, objective function of policy network is

$$q_w(s, \pi_\theta(s))$$

- Using experience replay D

$$J(\theta) = E_D[q_w(s, \pi_\theta(s)]$$

Training Policy Network

- Given objective function of policy network

$$J(\theta) = E_D[q_w(s, \pi_\theta(s))]$$

$$\nabla_\theta J(\theta) = E_D\left[\nabla_{\pi_\theta(s)} q_w(s, \pi_\theta(s)) \nabla_\theta \pi_\theta(s)\right]$$

- About $\nabla_{\pi_\theta(s)} q_w(s, \pi_\theta(s))$: differentiate value network q_w in terms of network input $\pi_\theta(s)$

- This means how much the input value a affects in learning q_w

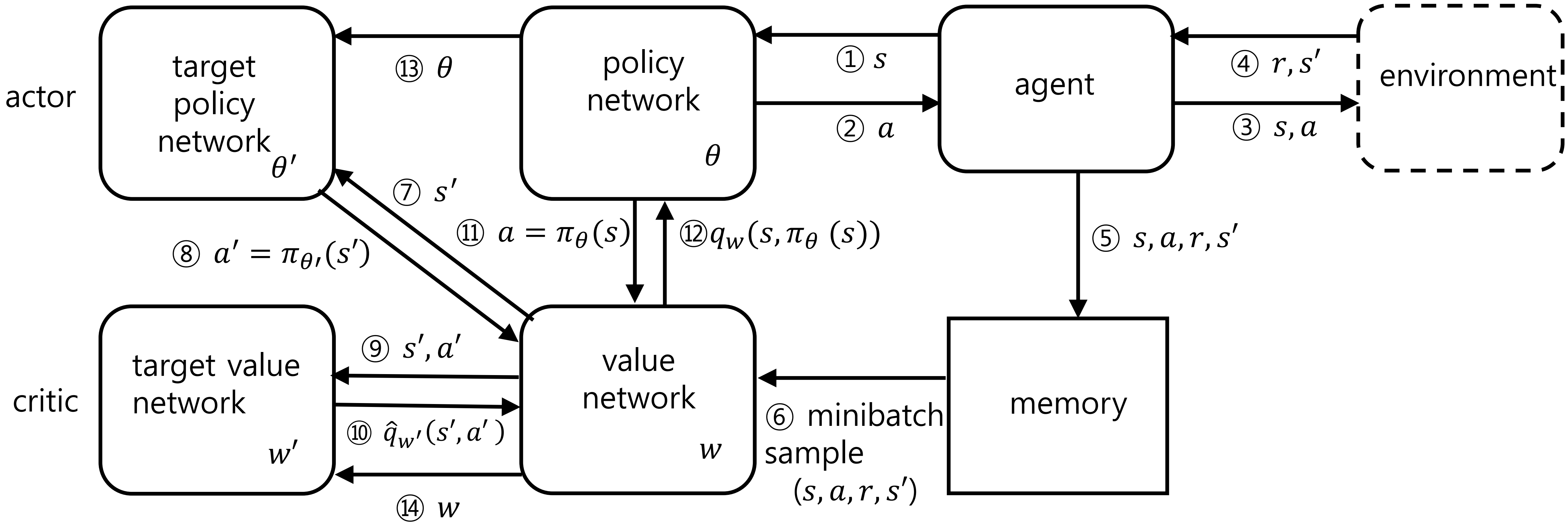
- Differentiating networks with respect to inputs are easily implemented in deep learning libraries(e.g., Pytorch).

- When choosing action a from policy network, we use the following to increase randomness

$$a = \pi_\theta(s) + e$$

- Periodically, parameters(θ) of policy network are copied to those(θ') of target policy network
  - Polyak update

DDPG


actor
target policy network θ'
policy network θ
agent
environment
① s
② a
③ s, a
④ r, s'
⑤ s, a, r, s'
⑬ θ
⑦ s'
⑧ $a' = \pi_{\theta'}(s')$
⑪ $a = \pi_\theta(s)$
⑫ $q_w(s, \pi_\theta(s))$
critic
target value network w'
value network w
memory
⑨ s', a'
⑩ $\hat{q}_{w'}(s', a')$
⑭ w
⑥ minibatch sample (s, a, r, s')

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$

repeat

  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$

  Execute a in the environment

  Observe next state s', reward r, and done signal d to indicate whether s' is terminal

  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$

  If s' is terminal, reset environment state.

  if it's time to update **then**

    for however many updates **do**

      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with Polyak update

$$\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$$

    end for

  end if

until convergence

https://spinningup.openai.com/en/latest/algorithms/ddpg.html

TD3(Twin Delayed Deep Deterministic Policy Gradient)

- TD3 is very similar to DDPG
- But, DDPG has a few problems:
  1) DDPG has Q value overestimation problem
  2) Since DDPG learns deterministic policy, it is necessary to increase randomness in action selection

Q overestimation problem:

- In DQN, it maintains two independent networks Q_A and Q_B
- Q_A is used for evaluating and Q_B is used for actual Q value

- In TD3, we choose minimum value between Q_A and Q_B
- $\min(Q_A, Q_B)$ becomes target of Q_A and Q_B network, respectively

TD3

Randomness in action

- Since DDPG handles deterministic policy, we need to increase the degree of exploration
- DDPG also adds randomness when choosing action a (p. 40)
  - when choosing action a from policy network, it adds a bit of random value

- In TD3, it add randomness when choosing action a' from state s' from experience replay data as well

$$(s, a, r, s')$$

- Therefore, action a' is selected using

  $a' = \pi_{\theta'}(s') + \epsilon,$ where $\epsilon \sim clip(N(0, \sigma), -\mathrm{c}, \mathrm{c})$
  - Assume noise ϵ follows Gaussian distribution $N(0, \sigma)$
  - To preventive excess noise, we also clip the noise

    $\epsilon \sim clip(N(0, \sigma), -\mathrm{c}, \mathrm{c})$ (usually $\sigma = 0.2$, $c = 0.5$)
- Also to prevent error from happening, action values are clipped as well.

$$a_{low} \leq a' \leq a_{high}$$

- Therefore, action a' is chosen based on

$$a' = clip(\pi_{\theta'}(s') + clip(\epsilon, -\mathrm{c}, \mathrm{c}), a_{low}, a_{high})$$

TD3

- Both Q-functions use a single target, calculated using whichever of the two Q-functions gives a smaller target value

$$y(r, s') = r + \gamma \min\left\{Q_{w'_A}(s', a'), Q_{w'_B}(s', a')\right\}$$

- And then both are learned by regressing to this target

$$E_{s,a,r,s'\sim D}\left[\left(y(r, s') - Q_{w_A}(s, a)\right)^2\right]$$
$$E_{s,a,r,s'\sim D}\left[\left(y(r, s') - Q_{w_B}(s, a)\right)^2\right]$$

- The policy is learned just by maximizing

$$J(\theta) = E_{s,a,r,s'\sim D}\left[q_{w_A}(s, \pi_\theta(s))\right]$$
$$\nabla_\theta J(\theta) = E_{s,a,r,s'\sim D}\left[\nabla_{\pi_\theta(s)} q_{w_A}(s, \pi_\theta(s)) \nabla_\theta \pi_\theta(s)\right]$$

  - same as DDPG

- In TD3, the policy is updated less frequently than the Q-functions are.

TD3

Initialize value network parameter w_A, w_B; Initialize policy network parameter θ
Target value network $w_A' \leftarrow w_A$, $w_B' \leftarrow w_B$; target policy network parameter $\theta' \leftarrow \theta$
for $t = 1$ to T do
  choose action a in state s $(a \sim \pi_\theta(s) + \epsilon$ where $\epsilon \sim \mathrm{clip}(N(0,\sigma), -c, c))$
  generate reward r and next state s'
  store (s, a, r, s') in memory R
  sample minibatch from R
  for each (s_i, a_i, r_i, s_i') in R
    $a_i' = \mathrm{clip}\left(\pi_{\theta'}(s_i') + \mathrm{clip}(N(0,\sigma), -c, c), a_{low}, a_{high}\right)$
    $y(r_i, s_i') = r_i + \gamma \min\left\{Q_{w_A'}(s_i', a_i'), Q_{w_B'}(s_i', a_i')\right\}$
  update value network parameter w_A and w_B, respectively
    $\underset{w}{\mathrm{argmin}} \frac{1}{N} \sum_i \left(y(r_i, s_i') - Q_{w_A}(s_i, a_i)\right)^2$ & $\underset{w}{\mathrm{argmin}} \frac{1}{N} \sum_i \left(y(r_i, s_i') - Q_{w_B}(s_i, a_i)\right)^2$
  if ($t \bmod d$) then
    update policy network parameter θ

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_i \nabla_{\pi_\theta(s_i)} q_{w_A}(s_i, \pi_\theta(s_i)) \nabla_\theta \pi_\theta(s_i)$$

    update target network parameters using Polyak method

$$w_A{}' = \kappa w_A + (1-\kappa) w_A{}'$$
$$w_B{}' = \kappa w_B + (1-\kappa) w_B{}'$$
$$\theta' = \kappa\theta + (1-\kappa)\theta'$$

  end if
end for