4. Understanding A3C

3. Before that, understand A-C or A2C

2. Before that, understand REINFORCE with baseline and see how it differs with A-C

1. Before that, understand REINFORCE : how policy gradient method works

1. Before that, understand REINFORCE : how policy gradient method works

Sum(of the rewards?) over states weighted by how often the states occur under policy pi

$$\begin{aligned}\nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) \\ &= \mathbb{E}_\pi\left[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t,\boldsymbol{\theta})\right].\end{aligned}$$

This starts from objective function…

1. Before that, understand REINFORCE : how policy gradient method works

$$J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0),$$

= true value function for policy pi determined by theta

This starts from objective function…

1. Before that, understand REINFORCE : how policy gradient method works

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta})$$
$$= \mathbb{E}_\pi\left[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t,\boldsymbol{\theta})\right].$$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \sum_a \hat{q}(S_t,a,\mathbf{w}) \nabla \pi(a|S_t,\boldsymbol{\theta}),$$

So then the update for parameters defining the policy would be like this

1. Before that, understand REINFORCE : how policy gradient method works

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta})$$
$$= \mathbb{E}_\pi\left[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t,\boldsymbol{\theta})\right].$$

But then to simplify

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi\left[\sum_a \pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a) \frac{\nabla \pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})}\right]$$
$$= \mathbb{E}_\pi\left[q_\pi(S_t,A_t) \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}\right] \qquad \text{(replacing } a \text{ by the sample } A_t \sim \pi\text{)}$$
$$= \mathbb{E}_\pi\left[G_t \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}\right], \qquad \text{(because } \mathbb{E}_\pi[G_t|S_t,A_t] = q_\pi(S_t,A_t)\text{)}$$

1. Before that, understand REINFORCE : how policy gradient method works

Then to update is

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}.$$

$$\nabla \ln x = \frac{\nabla x}{x}.$$

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

2. Before that, understand REINFORCE with baseline and see how it differs with A-C

Why baseline helps

Ex) if there is a state where all actions return high values, the baseline can help differentiate the higher valued action out of them.

Better if baseline can have roughly median or average of the possible value for all the actionsc

2. Before that, understand REINFORCE with baseline and see how it differs with A-C

REINFORCE with Baseline (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
        $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$ baseline
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

2. Before that, understand REINFORCE with baseline and see how it differs with A-C

So why need to compare A-C and baseline REINFORCE?

From the idea that the baseline is the state-value function…

And updating parameters for state-value function

And updating another set of parameters for the policy

Seems similar to Actor(policy) and critic (state-value func)

BUT IT IS NOT A-C

2. Before that, understand REINFORCE with baseline and see how it differs with A-C

Baseline REINFORCE is not A-C because

State-value func in baseline REINFORCE is not a critic because

State-value func is not used for bootstrapping but only as a baseline

3. Before that, understand A-C

Policy gradient method with bootstrapping critic

Temporal-difference method (it was Monte Carlo for REINFORCE)

So faster and more suitable for online problems

3. Before that, understand A-C

From just a state-value func for baseline

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big(G_{t:t+1} - \hat{v}(S_t,\mathbf{w})\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \tag{13.12}$$

$$= \boldsymbol{\theta}_t + \alpha\Big(R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \tag{13.13}$$

bootstrapping

3. Before that, understand A-C

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
- Initialize S (first state of episode)
- $I \leftarrow 1$
- Loop while S is not terminal (for each time step):
  - $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
  - Take action A, observe S', R
  - $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
  - $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla\hat{v}(S,\mathbf{w})$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\delta\nabla \ln\pi(A|S,\boldsymbol{\theta})$
  - $I \leftarrow \gamma I$
  - $S \leftarrow S'$

Note that delta can be seen as advantage = Advantage Actor Critic = A2C

4. Understanding A2C, A3C

Asynchronous advantage actor-critic

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// 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 $T > T_{max}$

Difference from just A-C:
Multiple A-C agents with a global shared parameter

Paper says that this is same as advantage

Accumulate gradient over multiple timesteps before applying

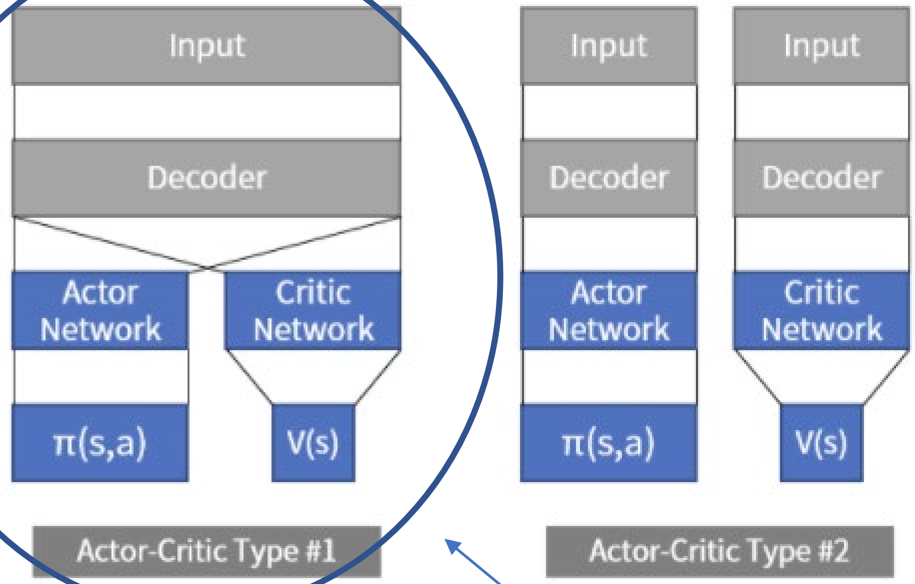
4. Understanding A2C, A3C

Further things to take note

Q. Reason for being asynchronous?

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// 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** In reverse order for proper bootstrapping
    $R \leftarrow r_i + \gamma R$
    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: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2/\partial\theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$



Really depends on the model but this ones seems to be this

Possible idea: incorporating experience replay into the asynchronous reinforcement learning framework to increase data efficiency

Debatable if it will get faster

코드!

https://github.com/laphisboy/RL_fall/tree/master/fall_week_4