Actor-Critic
Hung-yi Lee
Asynchronous Advantage Actor-Critic (A3C)

Review – Policy Gradient

\[ \nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \left( \sum_{t'=t}^{T_n} \gamma^{t'-t} r^n_{t'} - b \right) \nabla \log p_\theta(a^n_t | s^n_t) \]

\( G^n_t \): obtained via interaction

*Very unstable*

With sufficient samples, approximate the expectation of \( G \).

Can we estimate the expected value of \( G \)?

\[ G = 100 \]
\[ G = 3 \]
\[ G = 1 \]
\[ G = 2 \]
\[ G = -10 \]
Review – Q-Learning

- State value function $V^\pi(s)$
  - When using actor $\pi$, the *cumulated* reward expects to be obtained after visiting state $s$

- State-action value function $Q^\pi(s, a)$
  - When using actor $\pi$, the *cumulated* reward expects to be obtained after taking $a$ at state $s$

$V^\pi$ scalar

$Q^\pi$

Estimated by TD or MC

for discrete action only

$Q^\pi(s, a = \text{left})$

$Q^\pi(s, a = \text{right})$

$Q^\pi(s, a = \text{fire})$
Actor-Critic

\[ Q^{\pi_\theta}(s^n_t, a^n_t) - V^{\pi_\theta}(s^n_t) \]

\[ \nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \left( \sum_{t'=t}^{T_n} \gamma^{t'-t} r^n_{t'} - b \right) \nabla \log p_\theta(a^n_t | s^n_t) \]

\[ E[G^n_t] = Q^{\pi_\theta}(s^n_t, a^n_t) \]
Advantage Actor-Critic

Estimate two networks? We can only estimate one.

Only estimate state value
A little bit variance

\[ Q^n(s^n_t, a^n_t) = E[r^n_t + V^n(s^n_{t+1})] \]
\[ Q^n(s^n_t, a^n_t) = r^n_t + V^n(s^n_{t+1}) \]
Advantage Actor-Critic

\[ \pi \text{ interacts with the environment} \]

\[ \pi = \pi' \]

Update actor from \( \pi \rightarrow \pi' \) based on \( V^\pi(s) \)

Learning \( V^\pi(s) \)

\[ \nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \left( r_t^n + V^\pi(s_{t+1}^n) - V^\pi(s_t^n) \right) \nabla \log p_\theta(a_t^n|s_t^n) \]
Advantage Actor-Critic

• Tips
  • The parameters of actor $\pi(s)$ and critic $V^\pi(s)$ can be shared
  • Use output entropy as regularization for $\pi(s)$
    • Larger entropy is preferred $\rightarrow$ exploration
Asynchronous Advantage
Actor-Critic (A3C)

The idea is from 李思叡
Asynchronous

1. Copy global parameters
2. Sampling some data
3. Compute gradients
4. Update global models

\[ \theta^1 + \eta \Delta \theta \] (other workers also update models)
Pathwise Derivative Policy Gradient


Another Way to use Critic

**Original Actor-critic**

**Pathwise derivative**

**Policy gradient**

From Q function we know that taking $a'$ at state $s$ is better than $a$. We know the parameters of Q function.
Action $a$ is a continuous vector

$$a = \arg \max_a Q(s, a)$$

Actor as the solver of this optimization problem

http://www.cartomad.com/comic/109000081104011.html
Pathwise Derivative Policy Gradient

\[ \pi'(s) = \arg \max_a Q^\pi(s, a) \quad \text{a is the output of an actor} \]

Gradient ascent:
\[ \theta'^\pi \leftarrow \theta^\pi + \eta \nabla_{\theta^\pi} Q^\pi(s, a) \]

Update \( \pi \to \pi' \)

This is a large network
\( \pi \) interacts with the environment

Learning: \( Q^\pi(s, a) \)

Replay Buffer

Exploration

Find a new actor \( \pi' \) "better" than \( \pi \)

Actor

\( \pi \)

Update \( \pi \rightarrow \pi' \)

Update: \( \theta^{\pi'} \leftarrow \theta^\pi + \eta \nabla_{\theta^\pi} Q^\pi(s, a) \)

Replay Buffer

\( s \)

\( a \)

\( Q^\pi \)

\( Q^\pi(s, a) \)
Q-Learning Algorithm

• Initialize Q-function $Q$, target Q-function $\hat{Q} = Q$

• In each episode
  • For each time step $t$
    • Given state $s_t$, take action $a_t$ based on $Q$ (exploration)
    • Obtain reward $r_t$, and reach new state $s_{t+1}$
    • Store $(s_t, a_t, r_t, s_{t+1})$ into buffer
    • Sample $(s_i, a_i, r_i, s_{i+1})$ from buffer (usually a batch)
    • Target $y = r_i + \max_a \hat{Q}(s_{i+1}, a)$
    • Update the parameters of $Q$ to make $Q(s_i, a_i)$ close to $y$ (regression)

• Every $C$ steps reset $\hat{Q} = Q$
Q-Learning Algorithm ➔ Pathwise Derivative Policy Gradient

- Initialize Q-function $Q$, target Q-function $\hat{Q} = Q$, actor $\pi$, target actor $\hat{\pi} = \pi$
- In each episode
  - For each time step $t$
    1. Given state $s_t$, take action $a_t$ based on $Q$ (exploration)
    2. Obtain reward $r_t$, and reach new state $s_{t+1}$
    3. Store $(s_t, a_t, r_t, s_{t+1})$ into buffer
    4. Sample $(s_i, a_i, r_i, s_{i+1})$ from buffer (usually a batch)
  - Target $y = r_i + \max_a \hat{Q}(s_{i+1}, a)$
    1. Update the parameters of $Q$ to make $Q(s_i, a_i)$ close to $y$ (regression)
    2. Update the parameters of $\pi$ to maximize $Q(s_i, \pi(s_i))$
- Every C steps reset $\hat{Q} = Q$
- Every C steps reset $\hat{\pi} = \pi$
Connection with GAN

<table>
<thead>
<tr>
<th>Method</th>
<th>GANs</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freezing learning</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Label smoothing</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Historical averaging</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Minibatch discrimination</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Batch normalization</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Target networks</td>
<td>n/a</td>
<td>yes</td>
</tr>
<tr>
<td>Replay buffers</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Entropy regularization</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Compatibility</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>