Actor-Critic methods

Actor-Critic methods are a combination of

- Valued based
- Policy based

methods.

There are separate Neural Networks

- Critic Network

  Approximates the Value of each state

- Actor Network

  Implements the policy

with separate parameters and training objectives.

The Critic helps the Actor from operating blindly

- by providing estimates of the value of states
- to help the Actor to optimize policy in an informed direction

This collaborative effort

- reduces variance
- increases sample efficiency

The Actor-Critic method

- is *model-free*
- can be either
    - *on-policy*: guided by current Actor network parameters
        - using on-demand episode
    - *off-policy*
        - using a *replay buffer*
        - to store multiple episodes
            - perhaps accumulated with an earlier set of Actor network parameters
        - reduces noisiness of gradient for updates

Since both the Actor and Critic are Neural Networks

- they compute *functions* of state and action
- rather than storing tables (indexed by state and action)

Thus, they work well for continuously (rather than discrete) valued

- actions
- states

Comparison vs Valued-based and Policy-based methods

Method	Value Function Used	Policy Used Directly	Typical Action Spaces	Variance / Bias
Value-based	Yes	No (Implicit)	Discrete	Low variance
Policy-based	No	Yes	Continuous / Discrete	High variance
Actor-Critic	Yes	Yes	Both	Balanced

Vanilla Actor-Critic

The simplest Actor-Critic method is *Vanilla Actor-Critic*.

- Actor Loss is *exactly* that specified by the Policy Gradient Theorem
    - no additional terms added for practical/algorithmic considerations
    - hence, true Loss, rather than a Surrogate Loss
- Critic Loss is MSE: discrepancy between
    - current estimate of a state's value
    - an improved estimate of the state's value
        - improved with the knowledge from the current episode and action

Advantage for the Actor of Vanilla Actor-Critic

Recall the Unified Policy Gradient Formulation

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{=0}^{T-1} \nabla_\theta \log \pi_\theta(\backslash\mathrm{actseq}_{\tau,} | \backslash\mathrm{stateseq}_{\tau,t})\backslash\mathrm{advseq}_{\tau,}\right]$$

For Vanilla Actor-Critic, the advantage is:

$$\backslash\mathrm{advseq}_{\tau,t} = r_t + \gamma V(\backslash\mathrm{stateseq}_{\tau,+1}) - V(\backslash\mathrm{stateseq}_{\tau,})$$

Note that the Policy Gradient Theorem and Advantage are relevant

- **only** for the Policy Network

We can interpret the Advantage $\color{red}{\backslash\text{advseq}}_{\tau,t}$ by looking at each part

- $V(\color{red}{\backslash\text{stateseq}}_{\tau,})$ is the value of state $\color{red}{\backslash\text{stateseq}}_{\tau,}$
  - *before* the current episode
- $r_t + \gamma V(s_{\tau,t+1})$ is the *updated* value of state $\color{red}{\backslash\text{stateseq}}_{\tau,}$
  - after receiving reward r_t for taking action $\boxed{\text{\\actseq\_\\tt}}$ in the current episode

So the advantage is

- the *increment* to $V(\color{red}{\backslash\text{stateseq}}_{\tau,})$
- that occurs in step of the current episode

Note that is uses

- the *current* (pre-updating) value of successor state $\color{red}{\backslash\text{stateseq}}_{\tau,+1}$

Thus, the goal of updating the Actor network's parameters is to

- encourage actions with positive advantage
- where the advantage is the incremental return vs the current policy

Critic network goal

The Critic network's goal

- is to achieve the best estimate for the value of each state

This is expressed by minimizing the Loss

$$L_{\text{critic}} = \mathbb{E}_{s_t} \left[\left(V_w(s_t) - y_t \right)^2 \right]$$

where

- y_\tt is the *target value* for \stateseq_\tt
- often computed as

$$y_{=}\mathtt{r}_{+}\gamma\mathtt{V}_{\mathtt{w}}(\backslash\text{stateseq}_{+1})$$

  - where w are the *current* (pre-updating) values of the Critic network parameters

Re-writing

Thus the goal of updating the Critic network's parameters is to

- reduce the discrepancy between
- the current value of state $\backslash\mathrm{stateseq}_{\tau}$,
- and the target (updated) value y_\tt

Loss summary

Component	Purpose	Loss Function	Parameters Updated	
Actor (Policy)	Maximize expected return via policy gradient	$ L{\text{actor}} = - \log \pi\theta(a_t	s_t) \advseq_t $	θ (policy parameters)
Critic (Value)	Estimate value function to reduce variance	$L_{\text{critic}} = (V_w(s_t) - G_t)^2$ or Huber loss	w (value function parameters)	

Pseudo code for Vanilla Actor-Critic

Detailed Loss for Vanilla Actor-Critic

- Actor Loss

$$L_{\text{actor}} = -\mathbb{E}_{s_t, a_t} \left[\log \pi_\theta(a_t | s_t) \cdot A_t \right]$$

- Critic Loss

$$L_{\text{critic}} = \mathbb{E}_{s_t} \left[\left(V_w(s_t) - y_t \right)^2 \right]$$

where

- y_\tt is the *target value* for \stateseq_\tt
- often computed as

$$y_= \mathtt{r}_+ \gamma \mathtt{V}_{\mathtt{w}} (\backslash\text{stateseq}_{+1})$$

  - where w are the *current* (pre-updating) values of the Critic network parameters

```
# Initialize actor and critic neural networks
actor = initialize_actor_network()
critic = initialize_critic_network()

gamma = 0.99  # discount factor
max_episodes = 1000
max_steps = 500

for episode in range(max_episodes):
    state = env.reset()
    total_reward = 0

    for step in range(max_steps):
        # Get action probabilities from actor network
        action_probs = actor.predict(state)

        # Sample action from probabilities
        action = sample_action(action_probs)

        # Perform action, observe reward and next state
```

where

- `actor.predict(state)`
  - is the Actor network's action probability distribution $\pi(\cdot | \text{\stateseq})$
- `sample_action(action_probs)`
  - randomly chooses an action
  - based on the action probability distribution $\pi(\cdot | \text{\stateseq})$
  - exploration, not deterministic "choose max probability"
- `actor.predict(state)`
  - is the Critic's current estimate of the value of state `state`
- `actor.optimize(actor_loss)`, `critic.optimize(critic_loss)`
  - are Gradient Descent operators to minimize the loss (given in the respective arguments)

There is one notable element in the code

- the `stop_gradient` operator
- in the Actor Loss L_{actor}

```
actor_loss = -log_prob(action) * stop_gradient(advantage)
```

Note that

```
advantage = target - value
```

depends *only* on the parameters of the Critic Network.

The goal of the L_{actor} minimization is to change

- *only* the parameters of the Actor Network

We don't want any gradient *flowing to the Critic Network*

- via the `advantage` variable
- Critic network's parameters should *not be changed* when optimized the Actor network's parameters

The `stop_gradient(advantage)` operator

- prevents an upstream gradient (from `actor_loss` L_{actor})
- from flowing into `advantage`
  - and thus flowing into the Critic network

In [2]:

```
print("Done")
```

```
Done
```