Actor-Critic Method
In this lecture...

The actor-critic architecture

Least-Squares Policy Iteration

Natural actor-critic
Actor-critic methods

- Actor-critic methods implement *generalised policy iteration* - alternating between a policy evaluation and a policy improvement step.
- There are two closely related processes of
  - actor improvement which aims at improving the current policy
  - critic evaluation which evaluates the current policy
  If the critic is modelled by a bootstrapping method it reduces the variance so the learning is more stable than pure policy gradient methods.
Relation to other RL methods

Value-based methods:
► estimate the value function
► policy is implicit (e.g., $\epsilon$-greedy)

Policy-based methods
► estimate the policy
► no value function

Actor-critic methods
► estimate the policy
► estimate the value function
Actor-critic architecture
The policy used to generate the samples (behaviour policy) could be different from the one which is evaluated and improved (target policy).

This allows the critic to learn about the actions not preferred by the target policy and therefore improve the target policy.

This is impossible to achieve if behaviour policy is the same as target policy and if they are both deterministic.

In the case that the behaviour and the target policy are the same but stochastic the estimation on low-probability states might be poor.

If behaviour policy is completely random it might not visit important parts of the space.

The best choice of the behaviour policy is to add exploration into the target policy.
Implementing actor-critic architecture

Boltzmann policy estimated in a tabular way.

Small state-action space The critic is a Q-function estimator and the actor is $\epsilon$-greedy or Boltzmann policy estimated in a tabular way.

Large state-action spaces Both the critic and the actor use function approximation

$|A| \ll |S|$ sample complexity
Implementing a critic

- The critic estimates the value of the current policy – *prediction problem*
- Since the actor uses Q-values to choose actions, the critic must estimate the Q-function
- For small state-spaces we could use tabular TD algorithms to estimate the Q-function (SARSA, Q-learning, etc)
- For large state-spaces we could use LSTD to estimate the Q-function.

Small state-spaces: TD algorithms to estimate Q-fn. SARSA, Q-learning, ...
Implementing an actor

Policy improvement can be implemented in two ways:

- **Greedy improvement**: Moving the policy towards the greedy policy underlying the Q-function estimate obtained from the critic.
- **Policy gradient**: Perform policy gradient directly on the performance surface underlying the chosen parametric policy class.
Greedy improvement

- For small state-action spaces the policy is greedy with respect to the obtained Q-value
- For large state-action spaces the policy is parametrised and the greedy action is computed on the fly
Algorithm 1 Least-Squares Policy Iteration

1: Input: parametrisation of $Q(\cdot, \cdot; \theta) = \theta^T \phi(\cdot, \cdot)$
2: Initialise $\theta$ arbitrarily
3: repeat
4: \[ \pi(s) = \arg \max_a \theta^T \phi(s, a) \] \{policy improvement\}
5: \[ \theta = LSTD(\pi, \phi, \theta) \] \{policy evaluation\}
6: until convergence
Policy gradient

- Policy gradient methods perform stochastic gradient descent on the performance surface of the parametrised policy.
- Policy gradient theorem (last lecture) gives

\[
\nabla J(\omega) = E_\pi \left[ \gamma^t R_t \nabla \omega \log \pi(a|s, \omega) \right] \\
= E_\pi \left[ \gamma^t Q_\pi(s, a) \nabla \omega \log \pi(a|s, \omega) \right] \\
= E_\pi \left[ \gamma^t (Q_\pi(s, a) - V_\pi(s)) \nabla \omega \log \pi(a|s, \omega) \right]
\]

PROOF

- **Advantage function** \( A_\pi(s, a) \) is defined as

\[
A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)
\]
Compatible function approximation

**actor** policy that takes actions parametrised with $\omega$, for example

$$\pi(a|s, \omega) = \frac{\exp(\omega^T \psi(s, a))}{\sum_{a'} \exp(\omega^T \psi(s, a'))}$$

**critic** Advantage function that evaluates the actor parametrised with $\theta$

$$\gamma^t A(s_t, a, \theta) = \theta^T \phi(s_t, a)$$

such that the choice of the approximation is *compatible* with the policy parametrisation: if $\omega$ changes $\theta$ changes too.
Limitations of vanilla policy gradient

- Vanilla policy gradient methods are not always stable as (large) changes in the parameters can result in unexpected policy moves.
- Convergence can be very slow.

(More in next lecture.)
Natural actor-critic [Peters and Schaal, 2008]

- Uses compatible function approximation for actor and critic
- A modified form of gradient – *natural gradient* is used to find the optimal parameters
Natural Policy Gradient

- Advantage function is parametrised with parameters $\theta$ such that the direction of change is the same as for the policy parameters $\omega$

$$\gamma^t \nabla_\theta A(s_t, a, \theta) = \nabla_\omega \log \pi(s_t, a, \omega)$$

- Then by replacing

$$\gamma^t A(s_t, a, \theta) = \nabla_\omega \log \pi(s_t, a, \omega)^T \theta$$

in Eq 3

- It can be shown

$$\theta = G_\omega^{-1} \nabla_\omega J(\omega)$$

where $G_\omega$ is the Fisher information matrix

$$G_\omega = E_{\pi(\omega)} \left[ \nabla \log \pi(b, a, \omega) \nabla \log \pi(b, a, \omega)^T \right]$$

- $\theta$ is the natural gradient of $J(\omega)$
Natural gradient [Amari, 1998]

- Distance in Riemann space: \(|d\omega|^2 = d\omega^T G_\omega d\omega\), where \(G_\omega\) is a metric tensor
- Direction of steepest descent in Riemann space for some loss function \(L(\omega)\) is \(G_\omega^{-1} \nabla_\omega L(\omega)\)
- If \(\omega\) is used to optimise the estimate of a probability distribution \(p(x|\omega)\) then the optimal metric tensor is Fisher information matrix as this give distances invariant to scaling of the parameters.

\[
G_\omega = E(\nabla \log p(x|\omega) \nabla \log p(x|\omega)^T)
\]

- It can be shown that \(KL(p(x|\omega)||p(x|\omega + d\omega)) \approx d\omega^T G_\omega d\omega\)
Algorithm 2 Episodic Natural Actor Critic

1: Input: parametrisation of $\pi(\omega)$
2: Input: parametrisation of $\gamma^t A(\theta) = \theta^T \phi$
3: Input: step size $\alpha > 0$
4: Initialise $\omega$ and $\theta$
5: repeat
6: Execute the episode according to the current policy $\pi(\omega)$
7: Obtain sequence of states $s_t$, actions $a_t$ and return $R$
8: **Critic evaluation** Choose $\theta$ and $J$ to minimise $$(\sum_t \theta^T \phi(s_t, a_t) + J - R)^2$$
9: **Actor update** $\omega \leftarrow \omega + \alpha \theta$
10: until convergence

In practice the update is not performed after every episode but rather after a number of episodes to improve stability and efficiency.
Actor-critic methods implement generalised policy iteration where the actor aims at improving the current policy and the critic evaluates the current policy.

For large state-action spaces, both the actor and the critic are parametrised functions.

The actor and the critic can be estimated using compatible function approximation, where their parameters depend on each other and are estimated using stochastic gradient descent.

Instead of the vanilla gradient which has low convergence rates, the natural gradient can be used and this yields natural actor-critic algorithm.