Introduction to Reinforcement Learning

A mini course @ HCMUS, Vietnam

Lectures 4-6 (cont’d)

Long Tran-Thanh

Long.tran-thanh@warwick.ac.uk

Policy gradient

Last lecture

Function approximation:

- In large scale RL, we approximate V(s) and Q(s,a) with v(s, w) and q(s,a,w)
- We implicitly derive a policy from there: main goal is to learn the values. Policy is just a helper to do this learning efficiently (e.g., we use epsilon-greedy)
- This is called **value-based RL**

Value-based vs. policy-based RL

Function approximation:

- In large scale RL, we approximate V(s) and Q(s,a) with v(s, w) and q(s,a,w)
- We implicitly derive a policy from there: main goal is to learn the values. Policy is just a helper to do this learning efficiently (e.g., we use epsilon-greedy)
- This is called **value-based RL**

Question: Can we learn the policy in a more explicit way?

Idea: Remember policy iteration?

- Iterate over **deterministic** policies
- Challenge: search on large policy space

Value-based vs. policy-based RL

Function approximation:

- In large scale RL, we approximate V(s) and Q(s,a) with v(s, w) and q(s,a,w)
- We implicitly derive a policy from there: main goal is to learn the values. Policy is just a helper to do this learning efficiently (e.g., we use epsilon-greedy)
- This is called **value-based RL**

Question: Can we learn the policy in a more explicit way?

Idea: Remember policy iteration?

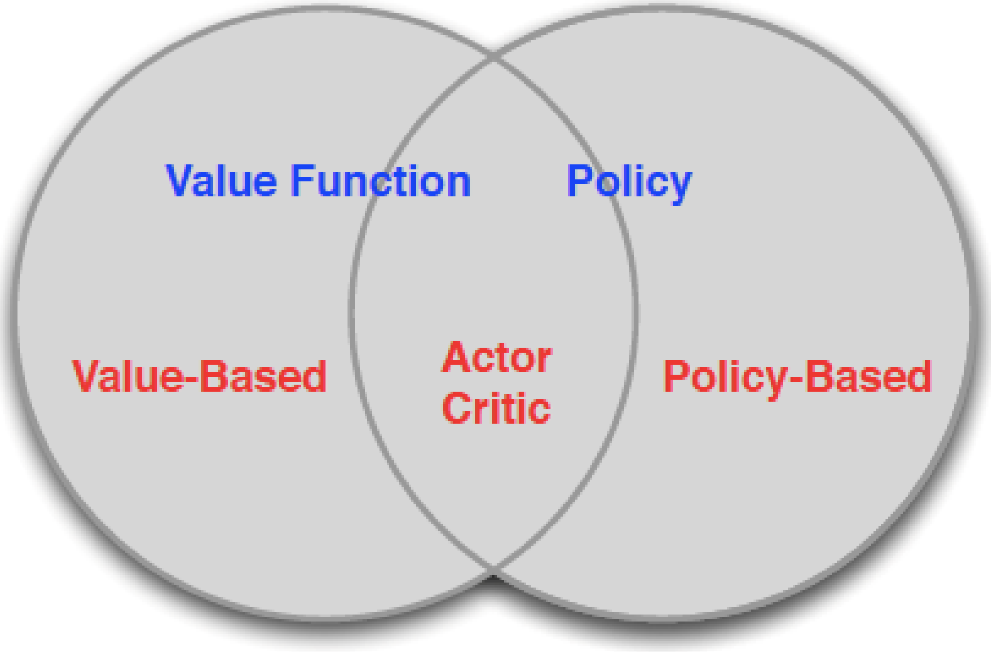
- Iterate over **deterministic** policies
- Challenge: search on large policy space

Idea 2: Why not parametrise the policies as well?

- More compact representation
- Can do efficient search directly on the policy space (using gradient descent)
- This is called **policy-based RL**

Value-based vs. policy-based RL

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



Taken from David Silver's slides

Policy-based RL

Advantages:

- Better convergence properties (converges faster)
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies
- Can deal with adaptive/adversarial environments where deterministic policies fail

Disadvantages:

- Typically converge to a local rather than global optimum (also gets stuck at saddle points in many cases)
- Evaluating a policy is typically inefficient and high variance

Stochastic policies vs. deterministic policies

Example: Two-player game of rock-paper-scissors

- Scissors beats paper
- Rock beats scissors
- Paper beats rock

Consider policies for iterated rock-paper-scissors

- A deterministic policy is easily exploited
- A uniform random policy of choosing each with 1/3 probability is optimal (i.e. Nash equilibrium)

What is the best policy?

We parametrise policies with parameter vector $\theta \in \Theta$

Goal: Find $\theta^* \in \Theta$ with best policy

But how do we measure the quality of a policy ? We need to define a utility function $J(\theta)$

What is the best policy?

We parametrise policies with parameter vector $\theta \in \Theta$

Goal: Find $\theta^* \in \Theta$ with best policy

But how do we measure the quality of a policy ? We need to define a utility function $J(\theta)$

In episodic environments we can use the start value: $J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}[G_1^{\pi_\theta}]$

What is the best policy?

We parametrise policies with parameter vector $\theta \in \Theta$

Goal: Find $\theta^* \in \Theta$ with best policy

But how do we measure the quality of a policy ? We need to define a utility function $J(\theta)$

In episodic environments we can use the start value: $J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}[G_1^{\pi_\theta}]$

In continuing environments we can use the average value: $J_{\mathrm{avV}}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$

where $d^{\pi_\theta}(s)$ is the stationary distribution of the underlying Markov chain

What is the best policy?

We parametrise policies with parameter vector $\theta \in \Theta$

Goal: Find $\theta^* \in \Theta$ with best policy

But how do we measure the quality of a policy ? We need to define a utility function $J(\theta)$

In episodic environments we can use the start value: $$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}[G_1^{\pi_\theta}]$$

In continuing environments we can use the average value: $$J_{\mathrm{avV}}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

where $d^{\pi_\theta}(s)$ is the stationary distribution of the underlying Markov chain

Or the average reward per time-step: $$J_{\mathrm{avR}}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) R(s,a)$$

where R(s,a) is the expected reward received at (s,a)

How to find the best policy?

Gradient-free solutions:

- Hill climbing
- Simplex / amoeba / Nelder Mead
- Genetic algorithms

Greater efficiency often possible using gradient:

- Gradient descent
- Conjugate gradient
- Quasi-newton

We focus on gradient descent, many extensions possible

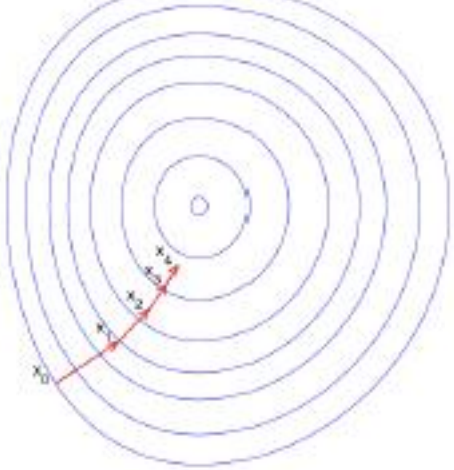
Recall gradient descent

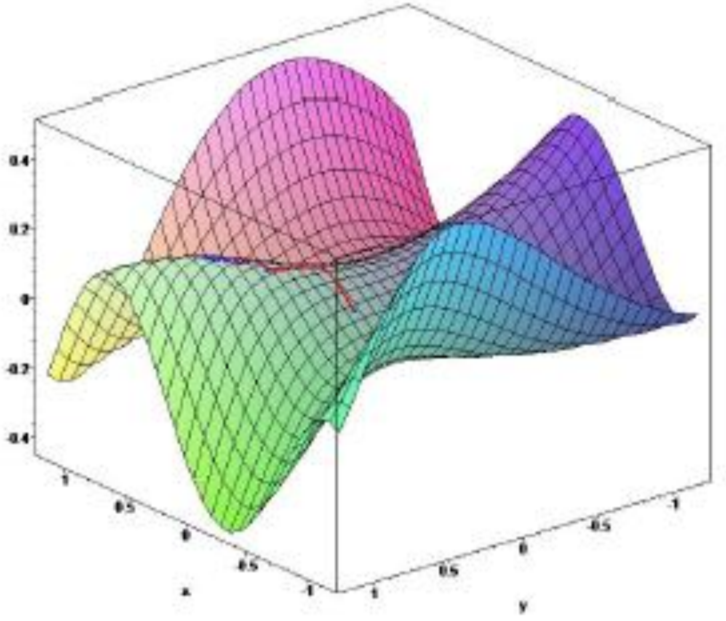
$J(\theta)$: a differentiable function of parameter vector θ
Goal: find local (global) **maximum** of $J(\theta)$

- Gradient of $J(\theta)$: $\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$

- Adjust θ in the **negative** direction of the gradient

$$\Delta\theta = -\alpha \nabla_\theta J(\theta)$$





Taken from David Silver's slides

How to compute the gradient?

- Finite differences approach

- Score function (with likelihood ratio)

Computing the gradient with finite differences

- Idea: $$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \varepsilon u_k) - J(\theta)}{\varepsilon}$$

  - $k \in [1, n], \ \ \varepsilon \to 0, \ \ u_k$ is the *k*th unit vector (with *1* in the *k*th component, and *0* elsewhere)

Computing the gradient with finite differences

- Idea: $$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \varepsilon u_k) - J(\theta)}{\varepsilon}$$
  - $k \in [1, n], \;\; \varepsilon \to 0, \;\; u_k$ is the *k*th unit vector (with *1* in the *k*th component, and *0* elsewhere)
- Uses n evaluations to compute policy gradient in n dimensions
- Simple, noisy, inefficient - but sometimes effective
- Works for arbitrary policies, even if policy is not differentiable

Computing the gradient with score function

- Consider a simple one-step MDP (we stop after 1 action)
  - Starting state follows distribution *d(s)*

$$J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s, a) R(s, a)$$

Computing the gradient with score function

- Consider a simple one-step MDP (we stop after 1 action)
  - Starting state follows distribution *d(s)*

$$J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) R(s,a)$$

  - Gradient: $$\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(s,a) R(s,a)$$

Computing the gradient with score function

- Consider a simple one-step MDP (we stop after 1 action)
  - Starting state follows distribution *d(s)*

$$J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) R(s,a)$$

  - Gradient: $$\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(s,a) R(s,a)$$
  - Simple trick: $$\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}$$
  - Using $\nabla \log(f(x)) = \dfrac{\nabla f(x)}{f(x)}$ for any $f(x)$:

Computing the gradient with score function

- Idea: consider a simple one-step MDP (we stop after 1 action)
  - Starting state follows distribution *d(s)*

$$J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) R(s,a)$$

  - Gradient: $$\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(s,a) R(s,a)$$
  - Simple trick: $$\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}$$
  - Using $\nabla \log(f(x)) = \dfrac{\nabla f(x)}{f(x)}$ for any $f(x)$:

$$\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \underbrace{\nabla_\theta \log(\pi_\theta(s,a))}_{\textbf{score function}}$$

Computing the gradient with score function

- Idea: consider a simple one-step MDP (we stop after 1 action)
  - Starting state follows distribution *d(s)*

$$J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) R(s,a)$$

  - Gradient: $$\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(s,a) R(s,a)$$

Computing the gradient with score function

- Idea: consider a simple one-step MDP (we stop after 1 action)
  - Starting state follows distribution *d(s)*

$$J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) R(s,a)$$

  - Gradient:

$$\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(s,a) R(s,a)$$

$$= \sum_s d(s) \sum_a \pi_\theta(s,a) \nabla_\theta \log(\pi_\theta(s,a)) R(s,a)$$

The policy gradient theorem

- It generalises the previous idea to general MDPs

Theorem

For any differentiable policy $\pi_\theta(s,a)$,
for any of the policy objective functions $J = J_1, J_{avR},$ *or* $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^{\pi_\theta}(s,a)\right]$$

The policy gradient theorem

- It generalises the previous idea to general MDPs

Theorem

For any differentiable policy $\pi_\theta(s,a)$,
for any of the policy objective functions $J = J_1, J_{avR},$ *or* $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^{\pi_\theta}(s,a)\right]$$

- It replaces R(s,a) with the expected Q-value of the policy

Approximating the gradient

- Gradient policy theorem: $\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) \nabla_\theta \log(\pi_\theta(s,a)) Q^{\pi_\theta}(s,a)$
- Challenge: calculate $Q^{\pi_\theta}(s,a)$

Approximating the gradient with MC

- Gradient policy theorem: $\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) \nabla_\theta \log(\pi_\theta(s,a)) Q^{\pi_\theta}(s,a)$
- Challenge: calculate $Q^{\pi_\theta}(s,a)$
- Solution: use MC method, use return $G_t^{\pi_\theta}$ as unbiased estimate of $Q^{\pi_\theta}(s_t,a_t)$
- This solution is called REINFORCE: MC + policy gradient theorem (Williams, 1992)

The REINFORCE algorithm

function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta - \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) G_t^{\pi_\theta}$
    end for
  end for
  return θ
end function

Approximating the gradient with value function approximation

- Gradient policy theorem: $\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) \nabla_\theta \log(\pi_\theta(s,a)) Q^{\pi_\theta}(s,a)$

- Challenge: calculate $Q^{\pi_\theta}(s,a)$

- Solution: use MC method, use return $G_t^{\pi_\theta}$ as unbiased estimate of $Q^{\pi_\theta}(s_t,a_t)$

- This solution is called REINFORCE: MC + policy gradient theorem (Williams, 1992)

    - Issue: MC has high variance -> requires lots of samples

Approximating the gradient with value function approximation

- Gradient policy theorem: $\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) \nabla_\theta \log(\pi_\theta(s,a)) Q^{\pi_\theta}(s,a)$
- Challenge: calculate $Q^{\pi_\theta}(s,a)$
- Solution: use MC method, use return $G_t^{\pi_\theta}$ as unbiased estimate of $Q^{\pi_\theta}(s_t,a_t)$
- This solution is called REINFORCE: MC + policy gradient theorem (Williams, 1992)
  - Issue: MC has high variance -> requires lots of samples
- Idea 2: use value function approximation: $\hat{q}(s,a,\mathbf{w}) \approx Q^{\pi_\theta}(s_t,a_t)$

Actor-critic algorithms

- Critic (responsible for the value function approximation part): Updates action-value function parameters ***w*** (using e.g., MC, TD(0), TS(lambda))

- Actor (responsible for the policy gradient part): Updates policy parameters θ , in direction suggested by critic

Actor-critic algorithms

- Critic (responsible for the value function approximation part): Updates action-value function parameters ***w*** (using e.g., MC, TD(0), TS(lambda))

- Actor (responsible for the policy gradient part): Updates policy parameters θ , in direction suggested by critic

- Actor-critic algorithms follow an approximate policy gradient:

$$\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) \nabla_\theta \log(\pi_\theta(s,a)) \hat{q}(s,a,w)$$

$$\Delta\theta = -\alpha \nabla_\theta \log(\pi_\theta(s,a)) \hat{q}(s,a,w)$$

Action-value Actor-critic (QAC – Crites & Barto, 1994)

- Critic: linear value function approx using TD(0)
- Actor: policy gradient

function QAC
  Initialise s, θ
  Sample $a \sim \pi_\theta$
  for each step **do**
    Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
    Sample action $a' \sim \pi_\theta(s', a')$
    $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
    $\theta = \theta - \alpha \nabla_\theta \log(\pi_\theta(s,a)) Q_w(s,a,w)$
    $w \leftarrow w + \beta \delta \phi(s, a)$
    $a \leftarrow a', s \leftarrow s'$
  end for
end function

Other Actor-critic algorithms

- A2C (Advantage Actor-Critic, Mnih *et al.*, 2016)

$$\theta = \theta - \alpha \nabla_\theta \log(\pi_\theta(s,a)) Q_w(s,a,w) \Longrightarrow A(s,a) = Q(s,a) - V(s)$$

  - Advantages: lower variance, in-line with theory

- A3C (Asynchronous Advantage Actor-Critic, Mnih *et al.*, 2016): multi-agent RL

- Deep Deterministic Policy Gradient (DDPG): for continuous action spaces + returns to Q-value (due to the deterministic policy)