Introduction to Reinforcement Learning

github.com/kengz/openai_lab

Wah Loon Keng
Laura Graesser

9 Dec 2017

RL system

- Agent (OpenAI Lab)
- Environment (OpenAI gym)
- SARS (state, action, reward, state_next) time transition

Characteristics

- Goal oriented, reward signals
- On-line, interactive
- Stateful environment (MDP)
- No full access to the function to optimize (no labeled data)

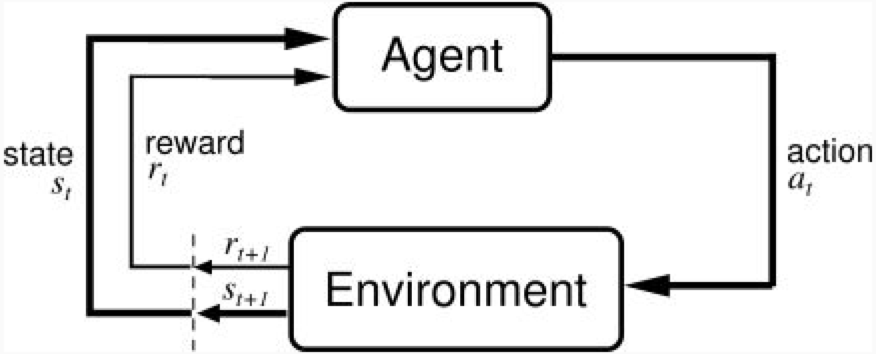


Image from http://wiki.ubc.ca/Course:CPSC522/Reinforcement_Learning

RL as MDP

Can be specified with:

S: state space (discrete/cont.)

A: action space (discrete/cont.)

R: reward distribution, $(\cdot|s,a)$

P: transition probability, $(\cdot|s,a)$

MDP:

1. Init, $t = 0$, env samples state $s_0 \sim p(s_0)$
2. For $t = 0$ until done:
    2.1. Agent selects action, a_t
    2.2. Env samples reward, $r_t \sim R(\cdot|s_t, a_t)$
    2.3. Env samples next state, $s_{t+1} \sim P(\cdot|s_t, a_t)$
    2.4. Agent receives r_t, s_{t+1}

CartPole-v0

State space (continuous, 4 dims):

- position
- velocity
- angle
- angular velocity
- e.g.: [0.023, 0.004, -0.061, 0.028]

Action space (discrete, 2 actions):

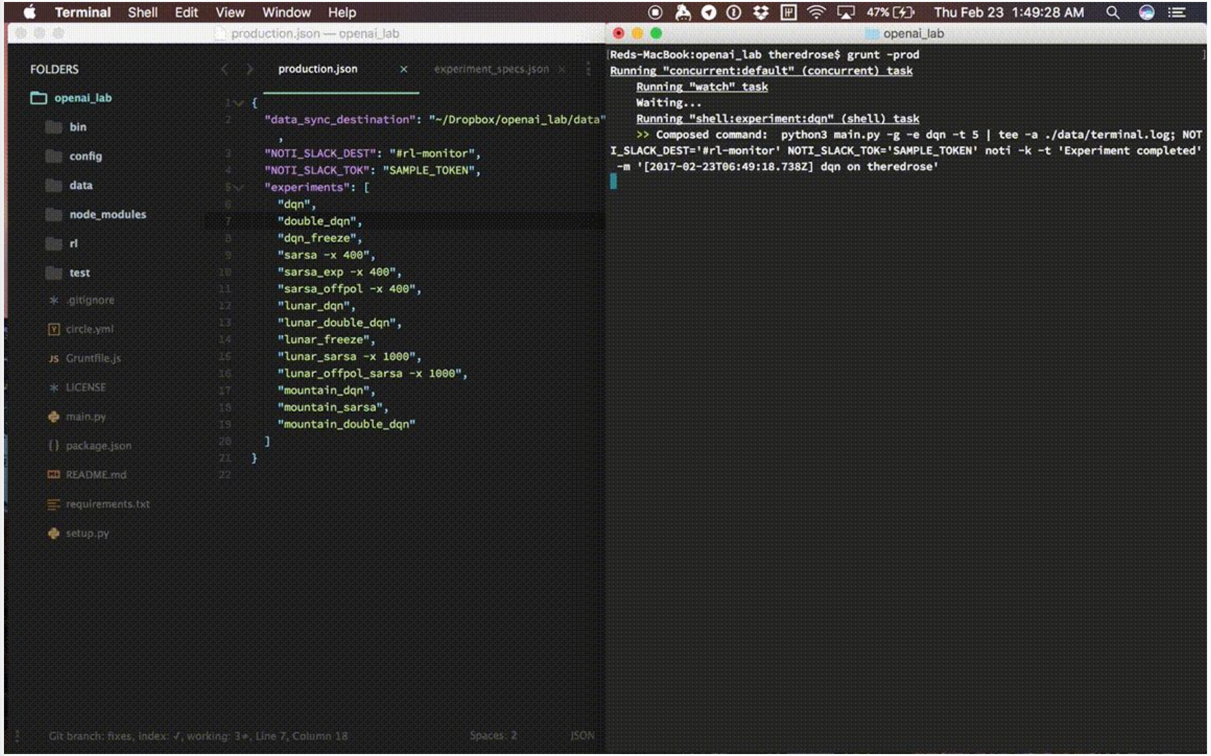
- push left: 0
- push right: 1
- e.g.: 0

Reward: +1 per timestep for staying up

Solution/epi: total rewards = 200

Solution: mean rewards > 195.0 over 100 consecutive episodes

github.com/kengz/openai_lab



What to learn in an RL problem?

1. What action to take? → **Policy** based algorithms
2. How good actions are given states? → **Value** based algorithms
3. The dynamics of the system? → **Model** based algorithms

Some or all of the above?

Deep-RL hint: use deep neural networks to approximate these functions.

Policy, Objective, Value

A policy is a function $\pi : S \rightarrow A$

Find a policy π^* that max. the cum. discounted reward $\sum_{t \geq 0} \gamma^t r_t$

$$\pi^* = arg \max_{\pi} E[\sum_{t \geq 0} \gamma^t r_t | \pi]$$

with $s_0 \sim p(s_0), a_t \sim \pi(\cdot | s_t), s_{t+1} \sim p(s_t, a_t)$

Value function: how good is a state?

$$V^{\pi}(s) = E[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi]$$

Q-value function: how good is a state-action pair?

$$Q^{\pi}(s, a) = E[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi]$$

Source: CS294, Deep Reinforcement Learning, 2017, Sergey Levine. https://www.youtube.com/watch?v=Q4kF8sfggoI&index=1&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3

Q-learning

Q-Learning

Use a neural net to approximate the Q-function

$$Q^{\pi}(s,a) = E\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

$$\text{Neural net approx. with param } \theta : Q(s,a;\theta) \approx Q^*(s,a)$$

With a perfect Q function we know how to act optimally - policy for free

$$\pi^*(s) = \max_a Q(s,a)$$

How to learn it? Bootstrapping

Bellman equation for optimal policy:

$$Q^*(s,a) = r(s,a) + \gamma E\left[V(s'|s,a)\right]$$

$$Q^*(s,a) \approx r(s,a) + \gamma \max_{a'} Q^*(s',a')|s,a$$

Source: CS294, Deep Reinforcement Learning, 2017, Sergey Levine. https://www.youtube.com/watch?v=Q4kF8sfggoI&index=1&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3

DQN Algorithm

Algorithm DQN

For i = 1 .... N:

Gather data (s_i, a_i, r_i, s'_i) by acting in the environment using some policy

for j = 1 ... K:

1. Calculate target values for each example

$$y_i = r_i + \gamma \max_{a'} Q(s'_i, a'; \theta_{i-1}) | s_i, a_i$$

2. Update network parameters, using MSE loss

$$L_j(\theta) = \frac{1}{2} \sum_i ||(y_i - Q(s_i, a_i; \theta_i))||^2$$

Source: CS294, Deep Reinforcement Learning, 2017, Sergey Levine. https://www.youtube.com/watch?v=Q4kF8sfggoI&index=1&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3

Properties

- Value-based
- Off-policy
- Model-free
- Online/Offline

Pros & Cons

Pros:

- Low variance - Q function estimates the expected cumulative discounted rewards from (s^t, a^t)
- Fairly sample efficient

Cons:

- High bias (critic imperfect)
- Q function has no convergence guarantee
- Exploration problem

Source: CS294, Deep Reinforcement Learning, 2017, Sergey Levine. https://www.youtube.com/watch?v=Q4kF8sfggoI&index=1&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3

Improving Q-learning

- Replay memory: somewhat decorrelates data, more sample efficient
- Target networks: make the moving target move less
- Different networks to select next state action and to evaluate it

Source: CS294, Deep Reinforcement Learning, 2017, Sergey Levine. https://www.youtube.com/watch?v=Q4kF8sfggoI&index=1&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3

Policy Gradient

Policy Gradient Formalism

Define a class of parametrized policies, $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each π_θ, define its value $J(\theta) = E\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$

Find the optimal policy $\theta^* = arg \max_\theta J(\theta)$

For trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$ and scalar score function $f(\tau)$

$$J(\theta) = E_{\tau \sim p(\tau;\theta)}\left[f(\tau)\right] = \int_\tau f(\tau)p(\tau;\theta)d\tau$$

Weights the probability for action a given s

Gradient update for convergence (with some magic):

$$\nabla_\theta J(\theta) = E_{\tau \sim p(\tau;\theta)}\left[f(\tau)\nabla_\theta logp(\tau;\theta)\right]$$
$$\approx \sum_{t\geq 0} f(\tau)\nabla_\theta log\pi_\theta(a_t|s_t)$$

Source: CS294, Deep Reinforcement Learning, 2017, Sergey Levine. https://www.youtube.com/watch?v=Q4kF8sfggoI&index=1&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3

Policy Gradient Algorithm (REINFORCE, Williams 1992)

Algorithm REINFORCE:

Initialize weights θ, learning rate α

for each episode (trajectory) $\tau = \{s_0, a_0, r_0, s_1, \cdots, r_T\} \sim \pi_\theta$

  for $t = 0$ to T do

    $\theta \leftarrow \theta + \alpha \; f(\tau)_t \nabla_\theta log\pi_\theta(a_t|s_t)$

  end for

end for

Source: CS294, Deep Reinforcement Learning, 2017, Sergey Levine. https://www.youtube.com/watch?v=Q4kF8sfggoI&index=1&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3

Properties

- Policy-based
- On-policy
- Model-free
- Offline (samples whole episodes)

Pros & Cons

Pros:

- Unbiased, uses direct sampling

Cons:

- High variance since the sampled reward varies per episode, credit assignment
- Inefficient sampling

Source: CS294, Deep Reinforcement Learning, 2017, Sergey Levine. https://www.youtube.com/watch?v=Q4kF8sfggoI&index=1&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3

Improving REINFORCE by lowering the variance:

Given $\nabla_\theta J(\theta) \approx \sum_{t\geq 0} f(\tau)\nabla_\theta log\pi_\theta(a_t|s_t)$, improve baseline with:

1. reward as weightage $f(\tau) = \sum_{t'\geq t} r_{t'}$

2. add discount factor $f(\tau) = \sum_{t'\geq t} \gamma^{t'-t} r_{t'}$

3. introduce baseline $f(\tau) = \sum_{t'\geq t} \gamma^{t'-t} r_{t'} - b(s_t)$

4. advantage function $f(\tau) = Q^\pi(s_t, a_t) - V^\pi(s_t) = A^\pi(s_t, a_t)$

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)\right)\nabla_\theta log\pi_\theta(a_t|s_t)$$

Source: CS294, Deep Reinforcement Learning, 2017, Sergey Levine. https://www.youtube.com/watch?v=Q4kF8sfggoI&index=1&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3

Actor-Critic

What to fit for the critic?

A, Q, or V?

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)\right) \nabla_\theta log\pi_\theta(a_t|s_t)$$

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

Q can be approximated with the reward from the current state and action, and V

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)}\left[V^\pi(s_{t+1})\right]$$

$$Q^\pi(s_t, a_t) \approx r(s_t, a_t) + V^\pi(s_{t+1})$$

$$A^\pi(s_t, a_t) = r(s_t, a_t) + V^\pi(s_t') - V^\pi(s_t)$$

We can just learn V

How to learn V? The bootstrap again!

$$V(s_t) = r(s_t, a_t) + V(s_t')$$

Source: CS294, Deep Reinforcement Learning, 2017, Sergey Levine. https://www.youtube.com/watch?v=Q4kF8sfggoI&index=1&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3

Actor-Critic Algorithm

For i = 1 .... N:

1. Gather data (s_i, a_i, r_i, s'_i) by acting in the environment using your policy
2. Update V

for j = 1 ... K:

Calculate target values for each example

$y_i = r_i + V(s'_i)$

Update network parameters, using MSE loss

$L_j(\theta) = \frac{1}{2} \sum_i ||(y_i - V(s_i; \theta_i))||^2$

3. Evaluate A, $A^\pi(s_i, a_i) = r(s_i, a_i) + V^\pi(s'_i) - V^\pi(s_i)$
4. Calculate gradient, $\nabla_\phi J(\phi) \approx \sum_i A^\pi_t(s_i, a_i) \nabla_\phi log\pi_\phi(a_i|s_i)$
5. Use gradient to update parameters ϕ

Source: CS294, Deep Reinforcement Learning, 2017, Sergey Levine. https://www.youtube.com/watch?v=Q4kF8sfggoI&index=1&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3

Properties

- Value and policy -based
- On-policy
- Model-free
- Online/offline

Pros & Cons

Pros:

- Low bias
- Low variance

Cons:

- Sample inefficient (from the policy gradient)

OpenAI Lab Demo

github.com/kengz/openai_lab

Appendix

Experience Replay

Memory unit

At each time step store <S, A, R, S’>

- State, S
- Action taken in S
- Reward after taking action A
- Next state, S’

Parameters

- Max memory length

Advantages

- Can learn from past experiences as well as present experience
- Can learn about optimal policy whilst following an exploratory policy
- Decorrelates data (somewhat)