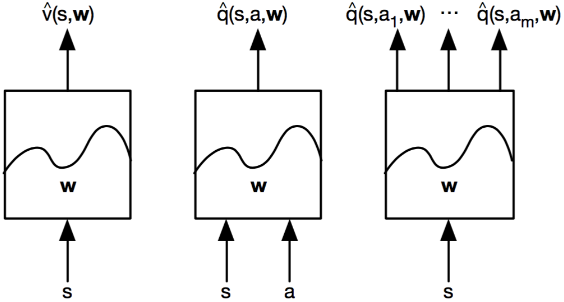
香港中文大學
The Chinese University of Hong Kong

Reinforcement Learning: function approximation

Zihao DENG
Wenqian ZHAO

Department of Computer Science and Engineering
The Chinese University of Hong Kong

Value Function Approximation



- Tablular methods: impossible to record all states for real word problems
- Function approximation: generalize from seen states to unseen states

Value Function Approximation

- Goal: find parameter vector w minimising mean-squared error between approximate value function $\hat{v}(S, w)$ and true value function $v_\pi(S)$

$$J(w) = ||v_\pi(S) - \hat{v}(S, w)||_2^2 \tag{1}$$

- Stochastic gradient descent

$$\Delta w = \alpha(v_\pi(s) - \hat{v}(S, w))\nabla_w \hat{v}(S, w) \tag{2}$$

- In reality we don't have the true value function $v_\pi(S)$
  - For Monte-Carlo, use discounted return G_t
  - For TD, use $R_{t+1} + \lambda \hat{v}(S_{t+1}, w)$

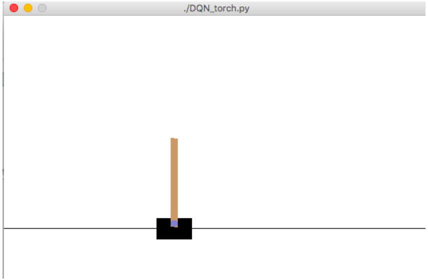
Deep Q-Networks (DQN)

- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in memory D
- Sample random mini-batch of transitions (s, a, r, s') from D
- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimise MSE between Q-network and Q-learning targets

$$L(w) = \mathrm{E}_{s,a,s,r' \sim D_i}[(r + \gamma \max_{a'} Q(s', a'; w^-) - Q(s, a; w))^2] \tag{3}$$

- Two important tricks in ensuring convergence: experience replay and fixed target

Deep Q-Networks(DQN): play games in OpenAI gym



- States are represented by 4-element tuples (position,cart velocity,angle,tip velocity)
- Actions can be either moving left or right
- Function approximator is a feed foward neural network
- 1 hidden layer with 10 neurons, 2 output neurons representing value estimation for two actions
- Implemented using torch and tensorflow, can stay alive for 1 minute

Policy based method

Value based method (previous sildes):

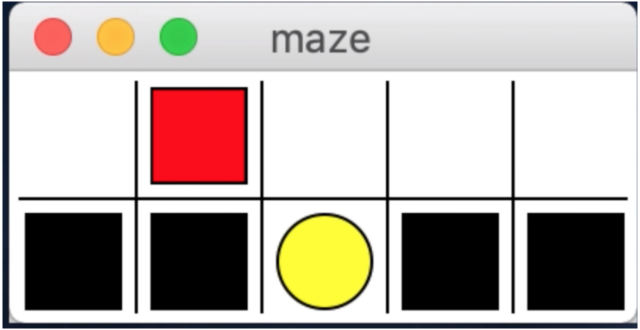
- Main focus is on state-action value evaluation
- Policy improvement is based on greedy or ϵ-greedy strategy w.r.t state-action values
- Return deterministic policy

Policy based method (slides after this page):

- Policy is a function of observations
- Policy improvement is based on gradient w.r.t some objective function
- State-action value not neccessary for policy updates
- Return stochastic policy

Policy based method

What's wrong with value based methods?



- Main problem: deterministic policy
- No good for partially observable environment
- Should I go left or right?

Policy Gradient: problem formulation

- Policy is a function of observation: $\pi_\theta(\cdot)$
- Trajectory τ: $\{s_0, a_0, r_0, s_1, a_1, r_1, ...\}$ is treated as random variable
- Distribution of τ is determined by policy π_θ
- For each trajectory, total reward is defined as $R(\tau)$
- Ultimate goal: optimize expectation $E_{\pi_\theta}[R(\tau)]$ w.r.t θ

Policy Gradient: approximate the gradient

- What does the gradient look like?

$$
\begin{aligned}
\nabla_\theta E_{\pi_\theta}[R(\tau)] &= \nabla_\theta \sum_\tau P_\theta(\tau)R(\tau) \\
&= \sum_\tau \nabla_\theta P_\theta(\tau)R(\tau) \\
&= \sum_\tau P_\theta(\tau)\frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)}R(\tau) = E_{\pi_\theta}[\nabla_\theta \ln P_\theta(\tau)R(\tau)]
\end{aligned} \tag{4}
$$

- Equation 4 tell us: gradient can be represented as an expectation
- Why it is important: expectation can be approximated by sampling

Policy Gradient: approximate the gradient

- Why the gradient even exists?

$$P(\tau) = P(s_0) \prod_{i=0}^{\infty} \pi_\theta(a_i, s_i) P(s_{i+1}|s_i, a_i) \tag{5}$$

- Assumption: there is an underlying MDP specifying $P(s_{i+1}|s_i, a_i)$ and $P(s_0)$

$$\begin{aligned}
\nabla_\theta \ln P(\tau) &= \nabla_\theta \ln[P(s_0) \prod_{i=0}^{\infty} \pi_\theta(a_i, s_i) P(s_{i+1}|s_i, a_i)] \\
&= \nabla_\theta \ln P(s_0) + \nabla_\theta \sum_{i=0}^{\infty} [\ln \pi_\theta(a_i, s_i) + \ln P(s_{i+1}|s_i, a_i)] \\
&= \nabla_\theta \sum_{i=0}^{\infty} \ln \pi_\theta(a_i, s_i)
\end{aligned} \tag{6}$$

Policy Gradient: understanding the formula

Combine all equation in previous slides, one important formula:

$$\nabla_\theta E_{\pi_\theta}[R(\tau)] = E_{\pi_\theta}[R(\tau)\nabla_\theta \sum_{s_i,a_i\in\tau} \ln \pi_\theta(a_i, s_i)] \tag{7}$$

Intuition from equation 7, adjustment magnitude of policy on $\pi_\theta(a, s)$:

- In proportion to the total reward gained from trajectories containing (a, s)
  - Rationale: good actions lead to good trajectories, while bad actions lead to bad ones
  - What about good actions in trajectories with bad overall performance? work on it later
- In inverse proportion to the probability of performing action a on state s
  - consider actions sampled frequently but with small positive effect
  - mitigate case where 'not-so-good' actions are rewarded frequently

Vanilla Policy Gradient: **REINFORCE**

So far, we obtain the first policy gradient algotithm called **REINFORCE** [Williams, R. J.]

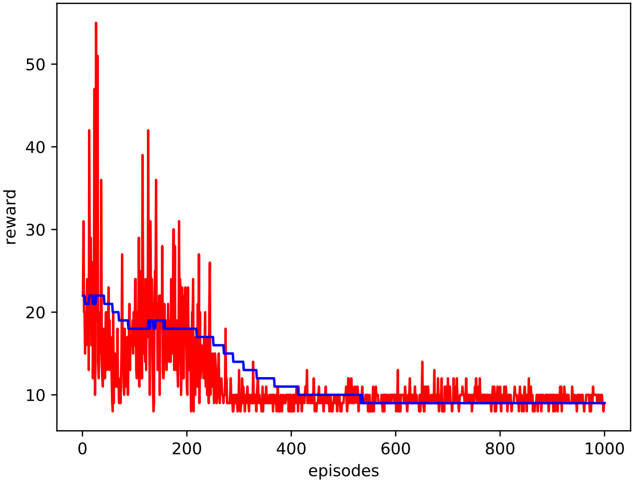
Algorithm 1 Generic Policy Gradient

1: Initialize policy parameters θ, learning rate α
2: **for** each iteration **do**
3: Collect trajectories $\{\tau_1, \tau_2, \tau_3, \ldots \tau_k\}$ using policy π_θ
4: Estimate gradient $\hat{grad} = \frac{1}{k} \sum_{i=1}^{k} [R(\tau_i) \nabla_\theta \sum \ln \pi_\theta(a, s)]$
5: Update policy $\theta \leftarrow \theta + \alpha \cdot \hat{grad}$
6: **end for**
7: **Return** policy π_θ

Vanilla Policy Gradient: **REINFORCE**

Experiment on CartPole game:

Bad performance, even not converge after 1000 episodes of training



Improvement for **REINFORCE**: baseline

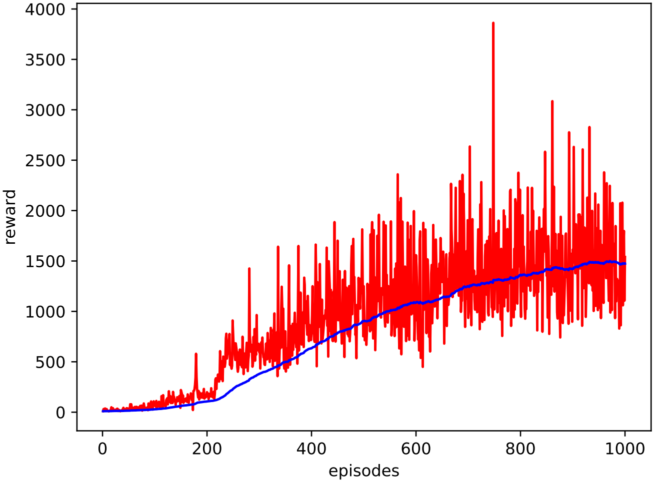
- One natural quetion: what if reward is always positive?
- Do we have to always increase $\pi_\theta(a, s)$ because $R(\tau)$ is positive (as in formula 7)?
- Actually, we only cares about the relative performance of trajectories
- Observation from formula 8: we can remove any constant term A from the expectation with out introducing bias.
- A can be the average performance for all trajectories, it is referenced as a baseline

$$
\begin{aligned}
E_{\pi_\theta}[\sum_a A \cdot \nabla \ln \pi(a, S)] &= \sum_a \pi_\theta(a, S) A \frac{\nabla_\theta \pi_\theta(a, S)}{\pi_\theta(a, S)} \\
&= A \sum_a \nabla_\theta \pi(a, S) \\
&= A \cdot \nabla_\theta \sum_a \pi_\theta(a, S) = A \nabla_\theta(1) = 0
\end{aligned} \tag{8}
$$

Improvement for **REINFORCE**: baseline

Experiment on CartPole game:

Better than before: an upgoing trend of rewards



Improvement for **REINFORCE**: advantage function

- Why we award/punish an action (s,a) based on the entire trajectory reward?
- Markov property: the action a_t only affects rewards after time t.

$$E_{\pi_\theta}[R_{0:i-1}(\tau)\nabla_\theta \ln \pi_\theta(a_i, s_i)] = 0 \tag{9}$$

- Actually, we can exploit the markov property to refine formula 7

$$\begin{aligned} \nabla_\theta E_{\pi_\theta}[R(\tau)] &= E_{\pi_\theta}[\sum(\nabla_\theta \ln \pi_\theta(a_i, s_i)(R_{0:i-1}(\tau) + R_{i:\infty}(\tau)))] \\ &= E_{\pi_\theta}[\sum(\nabla_\theta \ln \pi_\theta(a_i, s_i)R_{i:\infty}(\tau))] \end{aligned} \tag{10}$$

- Recall that subtraction of baseline doesn't change the expectation

$$\nabla_\theta E_{\pi_\theta}[R(\tau)] = E_{\pi_\theta}[\sum \nabla_\theta \ln \pi_\theta(a_i, s_i)(R_{i:\infty}(\tau) - V_{\pi_\theta}(s_i))] \tag{11}$$

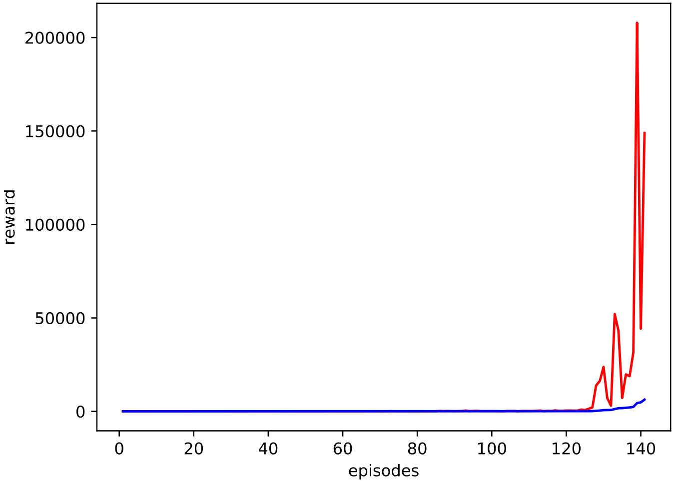
Improvement for **REINFORCE**: advantage function

- As shown in equation 11, $R_{i:\infty}(\tau) - V_{\pi_\theta}(s_i)$ is the actual term that determines the magnitude we adjust probability $\pi_\theta(a_i, s_i)$
- $R_{i:\infty}(\tau) - V_{\pi_\theta}(s_i)$ is also known as the advantage function
- Rationale: extra reward gained when performing certain action a_i on state s_i compared to average reward from that state under policy π_θ

Improvement for **REINFORCE**: advantage function

Experiment on CartPole game:

Only after 141 episodes of training, surviving time boosted to 20k!



Actor-Critic: a combination

So far, we mainly focused on pure value-based and pure policy-based methods ...

- Value-based: problem of deterministic policy in partially observed environments
- Policy-based: credit assignment problem (delay between action and reward)
- Why not combine them?
- Still use parameterized function for policy
- But also add an estimator for state values to approximate advantage function
- Stochastic policy agent without problem of update delay!

Actor-Critic: algorithm

Now there are two function to learn: policy function π_θ is known as actor and value estimator $\hat{V}_\varphi$ is known as critic, hence the model named **Actor-Critic**.

The overall algorithm can be like this (there are many variants, this is what we use in our experiment)

Algorithm 2 Actor-Critic

Initialize actor parameters θ, critic parameters φ, learning rate α
for each episode **do**
    Interact with environment for some time, get trajectory $\tau : \{s_0, a_0, r_0, s_1, a_0, r_0, ..., s_k\}$
    If s_k is Terminal, $R_{k:\infty} \leftarrow 0$ **else** $R_{k:\infty} \leftarrow V_\varphi\hat{(}s_k)$
    for each step i in $k-1:0$ **do**
        Estimate return $R_{i:\infty} \leftarrow r_i + R_{i+1:\infty}$
        Estimate advantage $A(a_i, s_i) \leftarrow R_{i:\infty} - V_\varphi\hat{(}s_i)$
    end for
    Estimate policy gradient $gr\hat{a}d = \frac{1}{k-1}\sum_{i=1}^{k-1}\sum[A(a_i, s_i)\nabla_\theta \ln \pi_\theta(a_i, s_i)]$
    Update actor $\theta \leftarrow \theta + \alpha \cdot gr\hat{a}d$
    Calculate loss for critic: $l(\varphi) \leftarrow \frac{1}{k-1}\sum_{i=0}^{k-1} A^2(a_i, s_i)$
    Update critic $\varphi \leftarrow \varphi + \alpha \cdot \nabla_\varphi l(\varphi)$
end for
Return policy π_θ

Further improvement: clipped objective function

- In previous section, policy gradient method works by computing gradient estimator in form

$$\hat{g} = \hat{E}_t[\nabla_\theta \ln \pi_\theta(a_t, s_t)\hat{A}_t] \tag{12}$$

- The estimator $\hat{g}$ can be obtained by differentiating the objective

$$L^{PG}(\theta) = \hat{E}_t[\ln \pi_\theta(a_t, s_t)\hat{A}_t] \tag{13}$$

- Multiple steps to optimize L^{PG} on same trajectory: destructively large policy updates [Schulman, John et al.].
- To improve sample efficiency, they adopt strategy of clipping the surrogate objective function in form $L^{CPI}(\theta)$:

Further improvement: clipped objective function

$$L^{CPI}(\theta) = \hat{E}_t[\frac{\pi_\theta(a_t, s_t)}{\pi_{\theta_{old}}(a_t, s_t)}\hat{A}_t] \tag{14}$$

- $\pi_{\theta_{old}}$: fixed term generated by old policy
- π_θ: current policy being optimized.
- The ratio $\frac{\pi_\theta(a_t,s_t)}{\pi_{\theta_{old}}(a_t,s_t)}$ is denoted as $r_t(\theta)$
- $r_t(\theta)$ measures the difference between current policy and old policy
- we don't want too big a update step, hence some constraint based on $r_t(\theta)$
- In practise we use the gradient of following objective function

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{15}$$

Further improvement: clipped objective function

The algorithm is known as Proximal Policy Optimization [Schulman, John et al.]

Only with minor modification on previous Actor-Critic algorithm:

Algorithm 3 Proximal Policy Optimization

Initialize actor parameters θ, critic parameters φ, learning rate α, clip coefficient ϵ
Initialize old policy $\pi_{\theta_{old}} \leftarrow \pi_\theta$
for each episode **do**
 for each time period in episode **do**
  denote current state as s_0, continue interacting with environment for some time
  get trajectory $\tau : \{s_0, a_0, r_0, s_1, a_0, r_0, ..., s_k\}$
  If s_k is Terminal, $R_{k:\infty} \leftarrow 0$ **else** $R_{k:\infty} \leftarrow \hat{V_\varphi(s_k)}$
  for each step i in $k-1:0$ **do**
   Estimate return $R_{i:\infty} \leftarrow r_i + R_{i+1:\infty}$
   Estimate advantage $A(a_i, s_i) \leftarrow R_{i:\infty} - \hat{V_\varphi(s_i)}$
  end for
  calculate loss for actor $l(\theta) \leftarrow \frac{1}{k}\sum_{t=0}^{k-1} \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$
  apply multiple steps gradient ascent on $l(\theta)$ to update θ
  calculate loss for critic: $l(\varphi) \leftarrow \frac{1}{k-1}\sum_{i=0}^{k-1} A^2(a_i, s_i)$
  apply multiple steps gradient descent on $l(\varphi)$ to update φ
  renew old policy $\pi_{\theta_{old}} \leftarrow \pi_\theta$
 end for
end for
Return policy π_θ

Further improvement: high score buffer replay