Reinforcement Learning

A brief introduction

Heejin Jeong

University of Pennsylvania

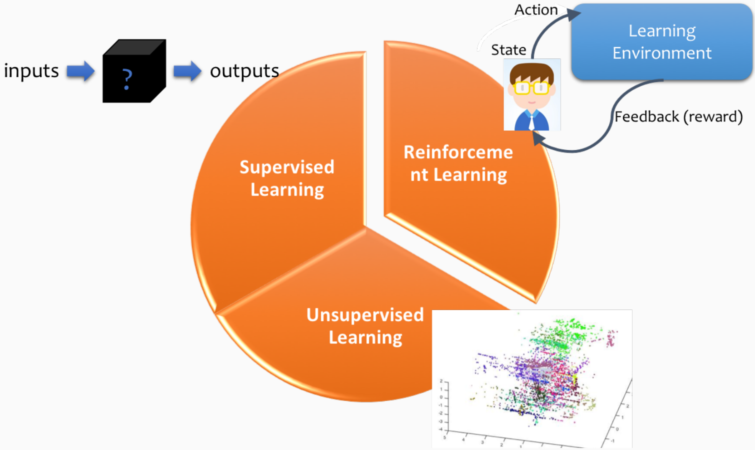
References:
Reinforcement Learning: An Introduction by A. Barrto and R. S. Sutton
Reinforcement Learning Course Slides by David Silver, UCL and Deepmind

Table of contents

1. What is Reinforcement Learning?
2. Markov Decision Process
3. Dynamic Programming
4. Exploration-Exploitation Trade-off
5. Monte-Carlo Methods
6. Q-learning
7. Value Function Approximation
8. Policy Gradient Methods

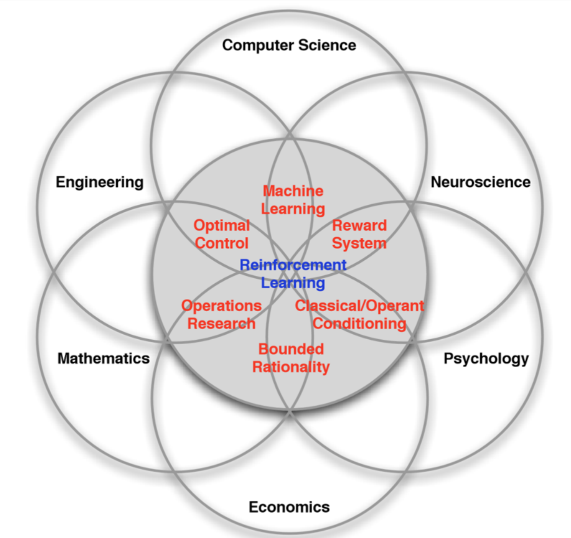
What is Reinforcement Learning?

Machine Learning

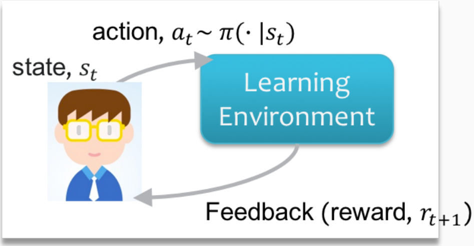


→ RL gives a mathematical framework for sequential decision making!

Reinforcement Learning Origins



Reinforcement Learning Framework

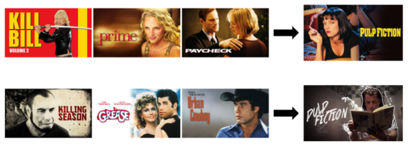


Goal of RL agent: To learn an optimal policy, π^* which maximizes its expected total discounted future reward

- Trial-and-error Search
- Delayed Reward

Examples of Reinforcement Learning Applications

- Playing Video/Board/Strategy games
- Finance
- Robotics
- Medicine
- Recommendation system




nature medicine

News & Views | Published: 05 November 2018

HEALTH CARE

Individualized sepsis treatment using reinforcement learning

Markov Decision Process

Finite Markov Decision Process (MDP)

Markov Assumption

The future is independent of the past given the present

$$P(x_{t+1}|x_0, \cdots, x_t) = P(x_{t+1}|x_t)$$

MDP Tuple: $\mathcal{M} =< \mathcal{S}, \mathcal{A}, P, R, \gamma >$

- State space, $\mathcal{S}$: a finite set of states. $s_t \in \mathcal{S}$
- Action space, $\mathcal{A}$: a finite set of actions. $a_t \in \mathcal{A}$
- Transition Probability Kernel, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$

$$p(s'|s, a) = P(s_{t+1} = s'|s_t = s, a_t = a)$$

- Reward function, $R : \mathcal{S} \times \mathcal{A} \rightarrow R$ or $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow R$

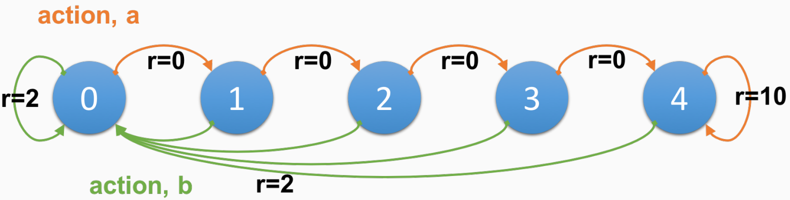
$$R(s, a) = \mathbb{E}[r_{t+1}|s_t = s, a_t = a]$$

$$R(s, a, s') = \mathbb{E}\left[r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'\right]$$

- Discount factor, $\gamma \in [0, 1]$

Finite Markov Decision Process (MDP) - example

Can you guess?



With probability P=0.1, the other action is executed

Finite Markov Decision Process (MDP)

Policy

- Stochastic Policy, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$
- Deterministic Policy, $\pi : \mathcal{S} \to \mathcal{A}$

Return

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-1} r_{t+T}$$

$T < \infty$ for an episodic task, $T = \infty$ for a continuing task

Value Function

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}) \,|s_t = s \right] = \mathbb{E}_\pi \left[G_t \,|s_t = s \right]$$

Action Value Function (Q value function)

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}) \,|s_t = s, a_t = a \right] = \mathbb{E}_\pi \left[G_t \,|s_t = s, a_t = a \right]$$

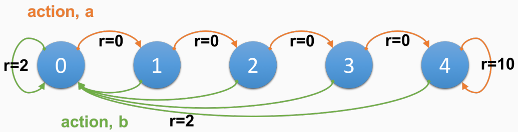
Bellman Expectation Equation

$$\begin{aligned}
G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \cdots = r_{t+1} + \gamma G_{t+1} \\
V^{\pi}(s) &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_{\pi} \left[G_{t+1} | s_{t+1} = s' \right] \right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma V^{\pi}(s') \right] \\
&= \mathbb{E} \left[r + \gamma V^{\pi}(s') \right]
\end{aligned}$$

Bellman Optimality Equation

$$\begin{aligned}
V^*(s) &= \max_{\pi} V^{\pi}(s) \\
Q^*(s, a) &= \mathbb{E} \left[R(s, a) + \gamma V^*(s') \right] \\
&= \mathbb{E} \left[R(s, a) + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \right] \\
V^*(s) &= \max_{a \in \mathcal{A}} Q^*(s, a)
\end{aligned}$$

Markov Decision Process - Optimality



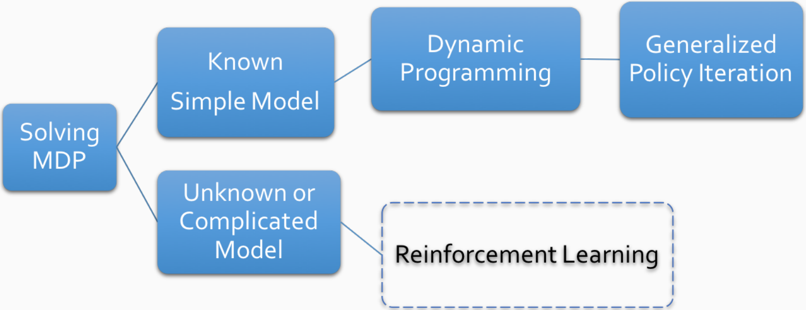
With probability P=0.1, the other action is executed

Q*	a	b
0	40.7	38.9
1	45.5	39.4
2	51.4	40.1
3	58.7	40.9
4	67.7	41.9

Optimal policy

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s, a)$$

How to solve MDPs?



Dynamic Programming

Dynamic Programming

Dynamic Programming

- Break down into sub-problems
- Solve the sub-problems
- Combine the sub-problem solutions

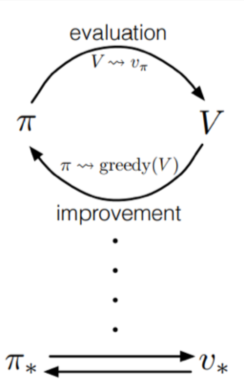
Applying to an MDP (but not limited to)

- Bellman equation is a *recursive decomposition*
- Dynamic Programming can solve an MDP with full knowledge of the MDP

Policy Iteration

1. Policy Evaluation : $V^{\pi}(s) = \mathbb{E}\left[r + \gamma V^{\pi}(s')\right]$
   For all $s \in \mathcal{S}$,
   $V_{k+1}(s) = \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V_k(s')\right]$
2. Policy Improvement
   For all $s \in \mathcal{S}$

$$\pi(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(s')\right]$$



Principle of Optimality

Generalized Policy Iteration (GPI) is the general idea of interacting **Policy Evaluation** and **Policy Improvement** independent of the granularity of the two processes. Almost all reinforcement learning methods are well described as GPI.

Policy Iteration

Policy iteration (using iterative policy evaluation)

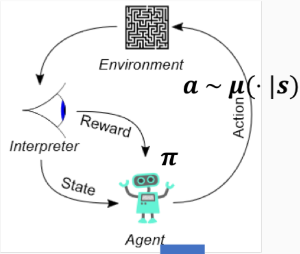
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

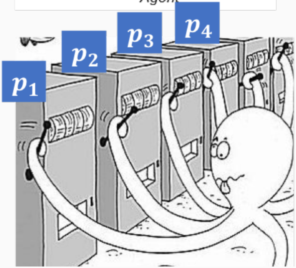
2. Policy Evaluation
   Repeat
   $\quad\Delta \leftarrow 0$
   $\quad$For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$*old-action* $\leftarrow \pi(s)$
   $\quad\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   $\quad$If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Exploration-Exploitation Trade-off

When should I stop exploring?





- State: $|S| = 1$
- Action: a_k: pulling k-th arm
- Gambling Machines: Return 1 with unknown probability p_k and 0 otherwise
- Reward = 1 or 0
- Cost : wasting in playing a suboptimal pull

Should I select the best arm based on my current knowledge?
Or,
Should I explore other arms?

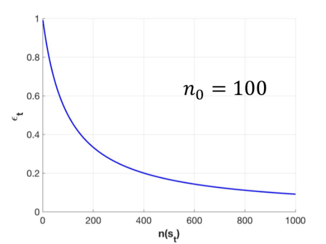
Control in RL

- Action (behavior) policy, μ : policy for choosing an action
- Target policy, π : policy that we want to update
- On-policy Control
  : Learning about a policy π using experience sampled from π (i.e. $\mu = \pi$)
- Off-policy Control
  : Learning a policy π using experience sampled from μ (i.e. $\mu \neq \pi$)
    - Safe Exploration
    - Learn from observing others

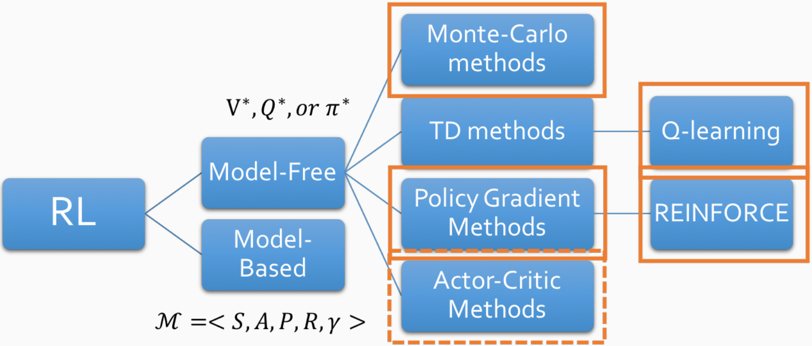
Example: ϵ-greedy Exploration

- Continual Exploration
  - With probability ϵ perform a randomly selected action
  - With probability $1 - \epsilon$ perform a greedy action
- For any ϵ-greedy policy, the ϵ-greedy policy μ with respect to Q^π is an improvement
- Time-varying $\epsilon = \epsilon_t$

$$\epsilon_t = \frac{n_0}{n_0 + visits(s_t)} \qquad \text{where} \quad n_0 : \text{constant}$$



Learning Methods

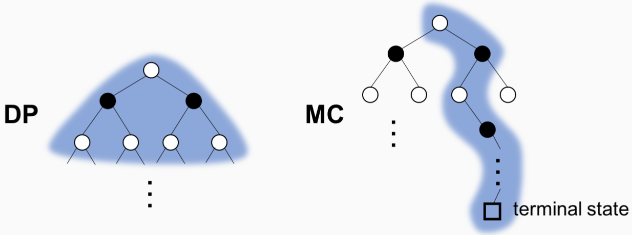


Monte-Carlo Methods

Monte Carlo Methods in RL

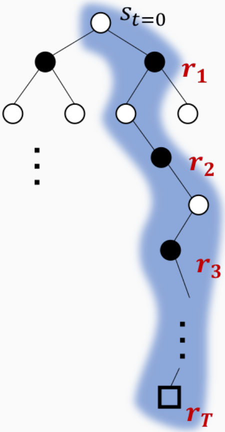
- *Monte Carlo* – repeated random sampling to obtain numerical results
- Ways of solving RL problems based on averaging complete sample returns
- Instead of using the expectation, we compute a complete return
- Therefore, defined only for episodic environments

[Backup Diagrams]



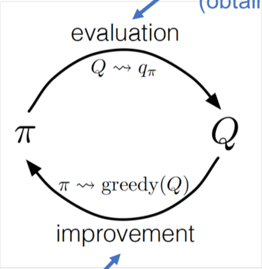
Monte Carlo Prediction

- Return, $G_t = r_{t+1} + \cdots + \gamma^{T-1} r_{t+T}$
  In MC, use empirical mean return starting from s_t or (s_t, a_t) instead of expected return is used for $V^\pi(s_t)$ or $Q^\pi(s_t, a_t)$
- $V^\pi(s) = \mathbb{E}\left[G_t | s_t = s\right]$ = average of the returns following all the visits to s in a set of episodes
- $Q^\pi(s, a) == \mathbb{E}\left[G_t | s_t = s, a_t = a\right]$ = average of the returns following all the visits to (s, a) pair in a set of episodes



Monte Carlo Updates

Sample an episode following the current action policy (obtain a **return**, G_t, of the episode)



Update the action value with the average of $[G_1, G_2, \cdots, G_N]$

Monte Carlo Updates

Incremental Monte Carlo Updates

Suppose we have a sequence of episode samples for s, a and consider only the first visit to s, a
: $G^{(1)}(s, a), \cdots, G^{(n)}(s, a)$ where $G^{(k)}$ is the return sample from the kth episode.
Then, the update rule for $Q_n(s, a)$ is:

$$
\begin{aligned}
Q_{n+1}(s, a) = \text{Average Return } &= \frac{\sum_{k=1}^{n} w_k G^{(k)}(s, a)}{\sum_{k=1}^{n} w_k} \\
&= Q_n(s, a) + \alpha \left(G^{(n)}(s, a) - Q_n(s, a) \right)
\end{aligned}
$$

α : learning rate

Off-policy Monte Carlo Control Algorithm

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
 $Q(s,a) \leftarrow$ arbitrary
 $C(s,a) \leftarrow 0$
 $\pi(s) \leftarrow$ a deterministic policy that is greedy with respect to Q

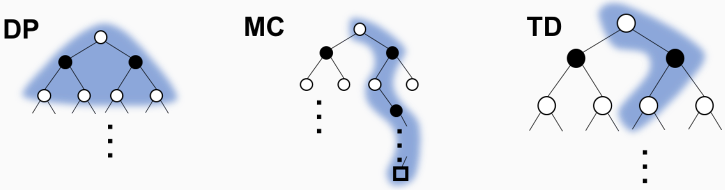
Repeat forever:
 Generate an episode using any soft policy μ:
  $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$
 $G \leftarrow 0$
 $W \leftarrow 1$
 For $t = T-1, T-2, \ldots$ downto 0:
  $G \leftarrow \gamma G + R_{t+1}$
  $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t,A_t)} [G - Q(S_t, A_t)]$
  $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
  If $A_t \neq \pi(S_t)$ then ExitForLoop
  $W \leftarrow W \frac{1}{\mu(A_t|S_t)}$

Monte Carlo Control **converges** with *action policy which is greedy in the limit* if all $(s, a) \in (\mathcal{S}, \mathcal{A})$ pairs are visited infinitely often.

Q-learning

Temporal Difference Prediction

- Combination of DP and MC
- Unlike MC, TD learns from your current predictions rather than waiting until termination
- $TD(0)$: One-step look ahead
  - TD target : $r_{t+1} + \gamma V(s_{t+1})$
  - TD error : $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - Q(s_t, a_t)$



Q-learning : Off-policy TD(0)

- On experience $< s_t, a_t, r_{t+1}, s_{t+1} >$ with greedy target policy π

$$
\begin{aligned}
Q(s_t, a_t) &\leftarrow Q(s_t, a_t) + \alpha \cdot \text{TD error} \\
&\leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma V(s_{t+1}) - Q(s_t, a_t)\right)
\end{aligned}
$$

  where $\alpha \in (0, 1)$ is a learning rate.

  Since it is off-policy, $V(s_{t+1}) = \max_{a'} Q(s_{t+1}, a')$.
- Convergence is guaranteed for discrete $\mathcal{S}, \mathcal{A}$ if:
  - $\alpha \in (0, 1)$
  - $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$
  - All (s, a) pairs are visited infinitely often

Q-learning : Off-policy TD(0)

Q-learning Algorithm

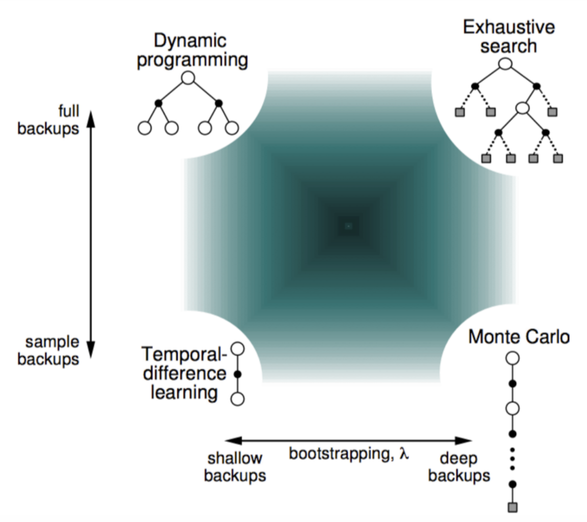
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$
    until S is terminal

When your subsequent state $s_{t+1} = S'$ is a terminal state, your expected future total reward is just the immediate reward:
TD target = r_{t+1}

MC vs. Q-learning

- MC: High Variance, Low Bias
  $\rightarrow$ Less sensitive to initial Q values
- Q-learning (TD): Low Variance, High Bias
    - Online learning is available. We wait only one time step!
    - Applications with long episodes : delaying all learning until an episode's end is too slow
    - Non-episodic (continuing) tasks
- It considers experimental actions
- Not theoretically proven, but in practice, TD methods converges faster than constant α MC methods on stochastic tasks

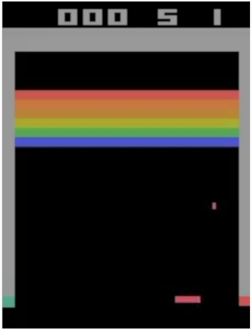
Summary



Dynamic programming
Exhaustive search
full backups
sample backups
Temporal-difference learning
Monte Carlo
shallow backups
bootstrapping, λ
deep backups

Value Function Approximation

Curse of Dimensionality



Value Function Approximation

Solution for large MDPs:

$$\hat{V}(s;\theta) \approx V^{\pi}(s)$$
$$\hat{Q}(s,a;\theta) \approx Q^{\pi}(s,a)$$

- Generate from seen states to unseen states
- Update parameter θ using MC or TD learning

Therefore, we learn the parameter of the function which has s or s, a as an input and $\hat{V}$ or $\hat{Q}$ as an output.

Function Approximators

- Linear Combinations of features
- Neural Network
- Decision Tree
- Nearest Neighbor
- Fourier / Wavelet basis

Differentiable?

Value Function Approximation by Stochastic Gradient Descent

Suppose $J(\theta)$ is a differentiable function of parameter θ:

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

The goal is to find θ^* which minimizes the *Mean Square Value Error*:

$$J(\theta) = \mathbb{E}_{s\sim\mu(\cdot)}\left[\left(V^\pi(s) - \hat{V}(s;\theta)\right)^2\right]$$

$$\frac{\partial J(\theta)}{\partial \theta} = 2\mathbb{E}_{s\sim\mu(\cdot)}\left[V^\pi(s) - \hat{V}(s;\theta)\right]\left(-\nabla_\theta \hat{V}(s;\theta)\right)$$

Then, update θ with the direction of minimizing the error:

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta J(\theta)$$

Value Function Approximation by Stochastic Gradient Descent

Stochastic Gradient Descent, (SGD)

Instead of computing the exact expectation, **sample** a value

$$\Delta\theta = \alpha \left(V^{\pi}(s) - \hat{V}(s;\theta) \right) \nabla_{\theta} \hat{V}(s;\theta)$$

$\rightarrow$ Its expected update is equal to the full gradient update!

Feature Vector

How do we compute $\hat{V}(s;\theta)$?

Represent state by a feature vector

$$\phi(s) = \begin{bmatrix} \phi_1(s) \\ \vdots \\ \phi_n(s) \end{bmatrix}$$

For example,

- Trends in the stock market
- Distance of robot from landmarks:
  s is robot's position and positions of the landmarks
- Principled Component Analysis
- Representation learning

Linear Value Function Approximator

The value function is represented by :

$$\hat{V}(s;\theta) = \phi(s)^T\theta$$

Then,

$$J(\theta) = \mathbb{E}_\pi\left[\left(V^\pi(s) - \phi(s)^T\theta\right)^2\right]$$

$\rightarrow$ quadratic in θ, therefore linear in θ in its gradient!

$$\Delta\theta = \alpha\left(V^\pi(s) - \phi(s)^T\theta\right)\phi(s)$$

How about Table Look-up Features?
How do we compute $V^\pi(s)$?

Online (Incremental) Prediction Algorithm

Monte-Carlo with Value Function Approximation

$$\text{Return, } G_t = r_{t+1} + \gamma r_{t+2} + \cdots \gamma^{T-1} r_{t+T}$$

TD(0) with Value Function Approximation

Similar to MC, but instead of G_t, use:

$$r_{t+1} + \gamma \hat{V}(s_{t+1}; \theta)$$

	Table Lookup	Linear	Non-Linear
MC Control on-policy	Optimal	Optimal	Diverge
TD(0) on-policy	Optimal	Diverge	Diverge
MC Control off-policy	Optimal	Optimal	Diverge
TD(0) off-policy	Optimal	Diverge	Diverge

Control with Function Approximation

Consider $Q^\pi(s,a)$ and s, a instead of $V^\pi(s)$ and s.

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta J(\theta) = \alpha E_\pi\left[Q^\pi(s,a) - \hat{Q}(s,a;\theta)\right]\nabla_\theta\hat{Q}(s,a;\theta)$$

Its SGD update for the linear function approximation:

$$\Delta\theta = \alpha\left(Q^\pi(s,a) - \hat{Q}(s,a;\theta)\right)\phi(s,a)$$

Online (Incremental) Control Algorithm

- For MC, $Q^\pi(s,a)$ target : G_t
- For off-policy TD(0), $Q^\pi(s,a)$ target : $r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a; \theta)$

Least Square Prediction

Collect Agent's experience, $\mathcal{D} := \{(s_1, V_1^\pi), \cdots, (s_T, V_T^\pi)\}$

Least square algorithm:

$$
\begin{aligned}
& \textit{minimize}_\theta \qquad LS(\theta) \\
\text{where } LS(\theta) =& \sum_{t=1}^{T} \left(V_t^\pi - \hat{V}(s_t; \theta) \right)^2 \\
=& \mathbb{E}_{\mathcal{D}} \left[\left(V^\pi - \hat{V}(s; \theta) \right)^2 \right]
\end{aligned}
$$

SGD with Experience Replay

Repeat,

(1) Sample a pair, $(s, V^\pi) \sim \mathcal{D}$

(2) Apply SGD, $\Delta\theta = \alpha \left(V^\pi - \hat{V}(s; \theta) \right) \nabla_\theta \hat{V}(s; \theta)$

Convergence

Value Prediction Algorithms:

	Table Lookup	Linear	Non-Linear
MC Control on-policy	Optimal	Optimal	Diverge
TD(0) on-policy	Optimal	Diverge	Diverge
MC Control off-policy	Optimal	Optimal	Diverge
TD(0) off-policy	Optimal	Diverge	Diverge

Control Algorithms:

	Table Lookup	Linear	Non-Linear
MC Control	Optimal	Near-optimal	Diverge
Q-learning	Optimal	Diverge	Diverge

Figure 1: Successful Deep RL Examples: TD Gammon, Atari Games, Game of Go

Deep Q-network

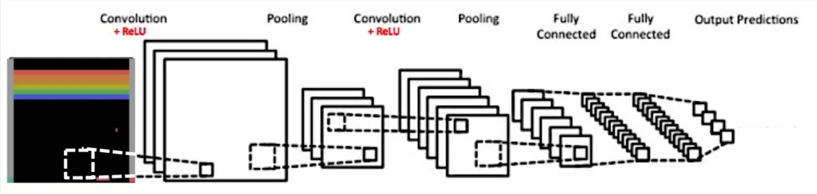
Major Features of DQN : Experience Replay and fixed Q-targets

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

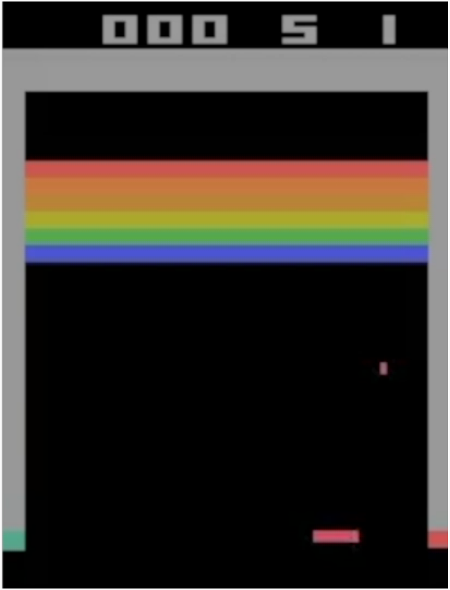
Target Value, $y_j = r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta^-)$ where θ^- are target network parameters.

Deep Q-network in Atari



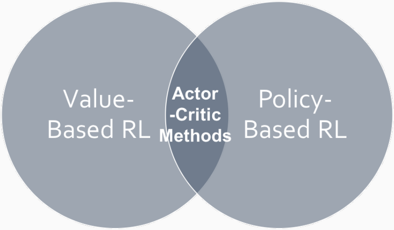
- state : a stack of raw pixel images from the last 4 frames
- action : 4-18 joystick/button positions
- reward : score

Deep Q-network in Atari



Policy Gradient Methods

Policy-Based RL



Instead of $\pi^*(s) = \text{argmax}_a Q^*(s, a)$, we want to explicitly learn an optimal policy:

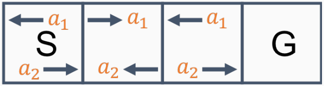
$$\pi_\theta(s, a) = Pr(a|s, \theta)$$

Finding θ which maximizes a performance measure $J(\theta) \rightarrow$ optimization problem

Policy Gradient: Optimize using stochastic gradient ascent

$$\theta_{t+1} = \theta_t + \alpha \nabla \hat{J}(\theta_t)$$

Policy-Based RL



With a function approximation, $x(s, a_1) = [1, 0]^T, x(s, a_2) = [0, 1]^T$, we need a **stochastic** policy.

- Advantages of Policy-based RL
  - Stochastic Policies (for POMDP)
  - Better Convergence Properties (at least local optima)
  - Effective in high-dimensional or continuous action spaces
- Disadvantages of Policy-based RL
  - Typically converge to a local rather than global optimum
  - Sample inefficient and high variance

Policy Objective Functions

Measure of the quality of a policy π_θ

1. Episodic Environments with a starting state, s_0

$$J(\theta) = V^{\pi_\theta}(s_0) = \mathbb{E}_{\pi_\theta}(V_0)$$

2. Continuing Environments
   - Average Value

$$J_{ave,V}(\theta) = \sum_s \rho^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

   - Average Reward per time-step

$$J_{ave,R}(\theta) = \sum_s \rho^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) R(s,a)$$

ρ^{π_θ} : stationary distribution of Markov chain for π_θ

Policy Gradient Methods

This is an optimization problem : Find θ that maximize $J(\theta)$.

Gradient Ascent:

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

Policy Gradient:

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

How to estimate the gradient?

- Computing Gradients by Finite Differences

$$\frac{\partial J(\theta)}{\partial \theta_n} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

where u_k is unit vector.
$\rightarrow$ Simple but noisy and inefficient

Policy Gradient Theorem

For any differentiable policy $\pi_\theta(s, a)$, for any of the policy objective functions $J(\theta)$, the policy gradient is:

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

Loglikelihood Trick, Score Function

Assuming that:

1. π_θ is differentiable whenever it is non-zero
2. $\nabla_\theta \pi_\theta(s, a)$

$$\nabla_\theta \pi_\theta(s, a) = \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)}$$

$\frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} = \nabla_\theta \log(\pi_\theta(s, a)) \rightarrow$ Score Function

Policy Examples

Softmax Policy

$\phi(s,a)^T\theta$: linear combination

$$\pi_\theta(s,a) \propto e^{\phi(s,a)^T\theta}$$

Then, the score function is:

$$\nabla_\theta \log(\pi_\theta(s,a)) = \phi(s,a) - \mathbb{E}_{\pi_\theta}\left[\phi(s,\cdot)\right]$$

Gaussian Policy

The most common policy for continuous action spaces.

$$\mu(s) = \phi(s)^T\theta$$

Then, an action is selected by $a \sim \mathcal{N}(\mu(s), \sigma^2)$.

The score function:

$$\nabla_\theta \log(\pi_\theta(s,a)) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$

REINFORCE : Monte Carlo Policy Gradient

$$\begin{aligned}\nabla_\theta J(\theta) =& \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s_t, a_t) Q^{\pi_\theta}(s_t, a_t)\right] \\ =& \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s_t, a_t) G_t\right]\end{aligned}$$

Stochastic Gradient Ascent Algorithm:

$$\theta_{t+1} = \theta_t + \alpha G_t \log \pi_\theta(s_t, a_t)$$

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

(Williams, 1992)

Actor-Critic Algorithm

Approximating Policy Gradient using **Critic** in order to reduce the large variance.

- Actor: Update the policy parameter θ (Policy Improvement)
- Critic: Update the Q-function, $Q(s, a; w)$ (Policy Evaluation)

$$\begin{aligned}\nabla_\theta J(\theta) =& \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s_t, a_t) Q^{\pi_\theta}(s_t, a_t)\right] \\ \approx& \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s_t, a_t) Q(s_t, a_t; w)\right]\end{aligned}$$

where w is a parameter of a function approximator of Q