Deep Reinforcement Learning

Zhihua Zhang

Peking University

July 2019

Outline

Machine Learning Tasks

- *Supervised Learning* is the task of inferring a classification or regression from labeled training data.
  - Training data, validate data, test data
- *Unsupervised Learning* is the task of making inferences from unlabeled datasets. (Passive)
  - Clustering, dimension reduction
  - Density estimation, data generation
- *Reinforcement Learning* is the task of learning how agents make sequential decisions in an environment in order to obtain a maximum cumulative reward. (Active)

Related Subjects

- Artificial Intelligence: Reinforcement learning, Deep learning + Reinforcement learning
- Operations Research: Approximation dynamic programming (discrete state)
- Control Theory: Optimal control (continuous sate)

References

- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT, 2016
- Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction, Second edition. MIT, 2019.
- Dimitri P. Bertsekas and John N. Tsitsiklis. Neuro-Dynamic Programming. Athena Scientific, 1996.
- Vincent Francois-Lavet et al. An introduction to deep reinforcement learning. Foundations and Trends in Machine Learning, 11(3-4), 219-354, 2018
- Conferences: NeuroIPS, ICML, ICLR, etc.

Contents

- Markov Decision Processes (MDPs)
- Finite-horizon MDPs: Dynamic programming or backward induction
- Infinite-horizon MDPs: Value function, value Iteration, Policy Iteration
- Q-Learning: Q Function, Q-value, Q-Policy
- RL: Off-policy, on-policy, TD(0), TD(λ)
- Deep RL: DQN, Policy gradient Algorithms

Definition of MDP

A MDP model consists of five elements: decision epochs, states, actions, transition probabilities, and reward

- Decision Epochs and Periods: Decisions are made at points of time referred to as decision epochs. Let T denote the set of decision epochs: either discrete $T = \{1, 2, \ldots, N\}$ or continuum $T = [0, N]$, and either finite $N < \infty$ or infinite $N = \infty$.
- State and Action Sets: At each decision epoch t, the system occupies a state s_t. Let $\mathcal{S}$ denote the set of possible system states. At some decision epoch, the decision maker observes the system in state $s \in \mathcal{S}$, and he may choose action a from the set $\mathcal{A}_s$ of allowable actions in state s.
- Rewards and Transition Probabilities: As a result of choosing action $a \in \mathcal{A}_s$ in state s at decision epoch t, 1) the decision maker receives a reward $r_t(s, a)$, and 2) the system state at the next epoch is determined by the probability distribution $p_t(\cdot|s, a)$.

Remarks

- There should define a terminal reward $r_N(s)$ in the finite-horizon case. The function $p_t(\cdot|s,a)$ is called a transition probability.
- When the reward depends on the state of the system at the next decision epoch, we let $r_t(s,a,j)$ denote the value of the reward received when the state of the system at decision epoch t is s, action $a \in \mathcal{A}_s$ is selected, and the system occupies state j at decision epoch $t+1$. Its expected value at epoch t is

$$r_t(s,a) = \sum_{j \in \mathcal{S}} p_t(j|s,a) r_t(s,a,j).$$

- The qualifier "Markov" is used because the transition probability and reward functions depend on the past only through the current state of the system and the action selected by the decision maker in that state.

Decision Rules

A decision rule prescribes a procedure for action selection in each state at a specified decision epoch.

- Markovian and Deterministic (MD) $\pi_t : \mathcal{S} \to \mathcal{A}$. "Markovian" means that it depends on previous system states and actions only through the current state of the system, and "Deterministic" means it chooses an action with certainty.
- Markovian and Randomized (MR) $q_{\pi_t(s_t)}(a) = \Pr^{\pi}(Y_t = a | X_t = s_t)$.
- History dependent and Deterministic (HD). For $h_t = (s_1, a_1, \ldots, s_{t-1}, a_{t-1}, s_t) \in \mathcal{H}_t$, $\pi_t : \mathcal{H}_t \to \mathcal{A}$.
- History dependent and Randomized (HR).

Policies

- A policy, plan, or strategy specifies the decision rule to be used at all decision epochs.
- A policy π is a sequence of decision rules, i.e., $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_{N-1})$.
- We call a policy stationary if $\pi_t = d$ for all $t \in T$ and denote $d^\infty = (d, d, \ldots)$ in the infinite-horizon case.

Notation

In the finite-horizon MDPs, $T = \{1, 2, \ldots, N\}$ with $N < \infty$.

- Let $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_{N-1}) \in \Pi^{\mathrm{HR}}$ denote a randomized history-dependent policy.
- Let $R_t \triangleq r_t(X_t, Y_t)$ denote the random reward received in period $t < N$, $R_N \triangleq r_N(X_N)$ denote the terminal reward, and $R \triangleq (R_1, R_2, \ldots, R_N)$ denote a random sequence of rewards.

The Expected Total Reward Criterion

Let $v_N^{\boldsymbol{\pi}}(s)$ represent the expected total reward over the decision making horizon if policy π is used and the system is in state s at the first decision epoch. For a "HR" π, it is defined as

$$v_N^{\boldsymbol{\pi}}(s) = \mathbb{E}^{\boldsymbol{\pi}}\left\{ \sum_{t=1}^{N-1} r_t(X_t, Y_t) + r_N(X_N) \middle| X_1 = s \right\}.$$

For a "HD", it is

$$v_N^{\boldsymbol{\pi}}(s) = \mathbb{E}^{\boldsymbol{\pi}}\left\{ \sum_{t=1}^{N-1} r_t(X_t, \pi(H_t)) + r_N(X_N) \middle| X_1 = s \right\}.$$

Optimal Policies

- MDP theory and algorithms for finite-horizon model primarily concern determining a policy $\boldsymbol{\pi}^*$ with the largest expected total reward. That is, seek a policy $\boldsymbol{\pi}^*$ for which

$$v_N^{\boldsymbol{\pi}^*}(s) \geq v_N^{\boldsymbol{\pi}}(s), s \in \mathcal{S}$$

  for all $\boldsymbol{\pi}$. We refer to such a policy as an optimal policy.
- Such a policy need not exist, so instead seek an ε-optimal policy, that is, for an $\varepsilon > 0$, a policy $\boldsymbol{\pi}_\varepsilon^*$ such that

$$v_N^{\boldsymbol{\pi}_\varepsilon^*}(s) + \varepsilon > v_N^{\boldsymbol{\pi}}(s), s \in \mathcal{S}$$

  for all $\boldsymbol{\pi}$.
- Anyway, we can define the value v_N^* of MDP as

$$v_N^*(s) = \sup_{\boldsymbol{\pi} \in \Pi^{\mathrm{HR}}} v_N^{\boldsymbol{\pi}}(s), s \in \mathcal{S}.$$

Policy Evaluation

- Let $u_t^{\boldsymbol{\pi}} : \mathcal{H}_t \to \mathbb{R}$ denote the total expected reward obtained by using policy $\boldsymbol{\pi}$ at decision epochs $t, t+1, \ldots, N-1$. Then

$$u_t^{\boldsymbol{\pi}}(h_t) = \mathbb{E}^{\boldsymbol{\pi}}\left\{ \sum_{k=t}^{N-1} r_k(X_k, Y_k) + r_N(X_N) \middle| h_t \right\}$$

  and let $u_N^{\boldsymbol{\pi}}(h_N) = r_N(s)$ when $h_N = (h_{N-1}, a_{N-1}, s)$.
- When $h_1 = s$, $u_1^{\boldsymbol{\pi}}(s) = v_N^{\boldsymbol{\pi}}(s)$.

Compute $v_N^{\boldsymbol{\pi}}(s)$ by Inductively evaluating $u_t^{\boldsymbol{\pi}}$

1. Set $t = N$ and $u_N^{\boldsymbol{\pi}}(h_N) = r_N(s_N)$ for all $h_N = (h_{N-1}, a_{N-1}, s_N) \in \mathcal{H}_N$.
2. If $t = 1$, Stop. Otherwise go to Step 3.
3. Substitute $t-1$ for t and compute $u_t^{\boldsymbol{\pi}}(h_t)$ for each $h_t = (h_{t-1}, a_{t-1}, s_t) \in \mathcal{H}_t$ by

$$u_t^{\boldsymbol{\pi}}(h_t) = r_t(s_t, \pi_t(h_t)) + \sum_{j \in \mathcal{S}} p_t(j|s_t, \pi_t(h_t)) u_{t+1}^{\boldsymbol{\pi}}(h_t, \pi_t(h_t), j).$$

   Note that $(h_t, \pi_t(h_t), j) \in \mathcal{H}_{t+1}$.
4. Return to Step 2.

The Bellman Equation

Define

$$u_t^*(h_t) = \sup_{\boldsymbol{\pi} \in \Pi^{\mathrm{HR}}} u_t^{\boldsymbol{\pi}}(h_t).$$

Assume either finite or countable $\mathcal{S}$. The Bellman equation (optimality equation) is

$$u_t(h_t) = \sup_{a \in \mathcal{A}_{s_t}} \left\{ r_t(s_t, a) + \sum_{j \in \mathcal{S}} p_t(j|s_t, a) u_{t+1}(h_t, a, j) \right\}$$

and $u_N(h_N) = r_N(s_N)$ for $h_N = (h_{N-1}, a_{N-1}, s_N) \in \mathcal{H}_N$.

Theorem

We have that

$$v_N^*(s) = \sup_{\boldsymbol{\pi} \in \Pi^{\mathrm{HR}}} v_N^{\boldsymbol{\pi}}(s) = \sup_{\boldsymbol{\pi} \in \Pi^{\mathrm{MD}}} v_N^{\boldsymbol{\pi}}(s), \ s \in \mathcal{S}.$$

Backward Induction or Dynamic Programming

1. Set $t = N$ and $u_N^*(s_N) = r_N(s_N)$ for all $s_N \in \mathcal{S}$.
2. Substitute $t-1$ for t and compute $u_t^*(s_t)$ for each $s_t \in \mathcal{S}$ by

$$u_t^*(h_t) = \max_{a \in \mathcal{A}_t} \left\{ r_t(s_t, a) + \sum_{j \in \mathcal{S}} p_t(j|s_t, a) u_{t+1}^*(j) \right\}.$$

   Set

$$A_{s_t}^* = \operatorname*{argmax}_{a \in \mathcal{A}_{s_t}} \left\{ r_t(s_t, a) + \sum_{j \in \mathcal{S}} p_t(j|s_t, a) u_{t+1}^*(j) \right\}.$$

3. If $t = 1$, Stop. Otherwise, Return to Step 2.

The Value of MDP

- In the infinite-horizon model, $T = \{1, 2, \ldots, \infty\}$.
- The expected total discounted reward of policy $\boldsymbol{\pi} \in \Pi^{\mathrm{HR}}$ is defined:

$$u_\lambda^{\boldsymbol{\pi}}(s) = \lim_{N\to\infty} \mathbb{E}_s^{\boldsymbol{\pi}}\left\{\sum_{t=1}^{N} \lambda^{t-1} r(X_t, Y_t)\right\} = \mathbb{E}_s^{\boldsymbol{\pi}}\left\{\sum_{t=1}^{\infty} \lambda^{t-1} r(X_t, Y_t)\right\}$$

  for $\lambda \in [0, 1)$.
- In discounted models, the value of the MDP, $u_\lambda^*(s)$, is defined by

$$u_\lambda^*(s) \triangleq \sup_{\boldsymbol{\pi}\in\Pi^{\mathrm{HR}}} u_\lambda^{\boldsymbol{\pi}}(s).$$

Assumptions

1. Assume the rewards $r(s,a)$ and transition probabilities $p(j|s,a)$ are stationary, which do not vary as decision epochs.
2. Bounded rewards: $|r(s,a)| \leq M < \infty$ for all $a \in \mathcal{A}$ and $s \in \mathcal{S}$.
3. Discounting: future rewards are discounted according to a discount factor λ, with $0 \leq \lambda < 1$.
4. Assume discrete (finite or countable) $\mathcal{S}$.

Matrix Expression

- For $\boldsymbol{\pi} \in \Pi^{\mathrm{MD}}$, define $r_\pi(s)$ and $P_\pi(j|s)$ by

$$r_\pi(s) \triangleq r(s, \pi(s)) \text{ and } P_\pi(j|s) \triangleq p(j|s, \pi(s))$$

  and for $\boldsymbol{\pi} \in \Pi^{\mathrm{MR}}$, define $r_\pi(s)$ and $P_\pi(j|s)$ by

$$r_\pi(s) \triangleq \sum_{a \in \mathcal{A}_s} q_{\pi(s)}(a) r(s, a) \text{ and } P_\pi(j|s) \triangleq \sum_{a \in \mathcal{A}_s} q_{\pi(s)}(a) p(j|s, a)$$

- Let $\mathbf{r}_\pi$ denote the $|\mathcal{S}|$-vector, with the sth element $r_\pi(s)$, and P_π the $|\mathcal{S}| \times |\mathcal{S}|$ matrix with (s, j)th entry given by $P_\pi(j|s)$.
- Let $\mathbf{u}_\lambda^\pi$ denote the $|\mathcal{S}|$-vector, with the sth element $u_\lambda^\pi(s)$. Then

$$\mathbf{u}_\lambda^{\boldsymbol{\pi}} = \sum_{t=1}^{\infty} \lambda^{t-1} P_\pi^{t-1} \mathbf{r}_{\pi_t}.$$

The Bellman Equation

- Let $d^\infty = (d, d, \ldots)$. Then

$$\sup_{\boldsymbol{\pi} \in \Pi^{\mathrm{HR}}} u_\lambda^{\boldsymbol{\pi}}(s) = \sup_{\boldsymbol{\pi} \in \Pi^{\mathrm{MD}}} u_\lambda^{\boldsymbol{\pi}}(s) = \sup_{d \in D} u_\lambda^{d^\infty}(s).$$

- The optimality equation is

$$\mathbf{u} = \sup_{d \in D} \{\mathbf{r}_d + \lambda P_d \mathbf{u}\} = \mathcal{L}\mathbf{u} \quad \text{(or)}$$

$$\mathbf{u} = \max_{d \in D} \{\mathbf{r}_d + \lambda P_d \mathbf{u}\} = L\mathbf{u}$$

- For a stationary policy d^∞, its value satisfies

$$\mathbf{u} = \mathbf{r}_d + \lambda P_d \mathbf{u}.$$

Contraction Mapping

Theorem (Contraction Mapping)

Under Assumptions 1-4, $\mathcal{L}$ and L are contraction mapping; that is,

$$\|\mathcal{L}\mathbf{u} - \mathcal{L}\mathbf{v}\| \leq \lambda \|\mathbf{u} - \mathbf{v}\|,$$
$$\|L\mathbf{u} - L\mathbf{v}\| \leq \lambda \|\mathbf{u} - \mathbf{v}\|.$$

Banach Fixed-Point Theorem

Lemma (Banach Fixed-Point Theorem)

Suppose $\mathcal{U}$ is a Banach space (complete normed linear space) and $T : \mathcal{U} \to \mathcal{U}$ is a contraction mapping. Then

- *There exists a unique $\mathbf{u}^*$ in $\mathcal{U}$ such that $T\mathbf{u}^* = \mathbf{u}^*$; and*
- *For arbitrary $\mathbf{u}^{(0)}$ in $\mathcal{U}$, then sequence $\{\mathbf{u}^{(n)}\}$ defined by*

$$\mathbf{u}^{(n+1)} = T\mathbf{u}^{(n)} = T^{n+1}\mathbf{u}^{(0)}$$

converges to $\mathbf{u}^*$.

Theorem (Existence of Optimal Policies)

Suppose $\mathcal{S}$ is finite or countable, then for all $\varepsilon > 0$ there exists an ε-optimal deterministic stationary policy.

The Value Iteration Algorithm

1. Select $\mathbf{u}^{(0)} \in \mathcal{U}$, specify $\varepsilon > 0$, and set $n = 0$.
2. For each $s \in \mathcal{S}$, compute $u^{(n+1)}(s)$ by

$$u^{(n+1)}(s) = \max_{a \in \mathcal{A}_s} \left\{ r(s,a) + \sum_{j \in \mathcal{S}} \lambda p(j|s,a) u^{(n)}(j) \right\}.$$

3. If $\|\mathbf{u}^{(n+1)} - \mathbf{u}^{(n)}\| < \frac{1-\lambda}{2\lambda}\varepsilon$, go to Step 4. Otherwise increment n by 1 and return to Step 2.
4. For each $s \in \mathcal{S}$, choose

$$d_\varepsilon(s) = \operatorname*{argmax}_{a \in \mathcal{A}_s} \left\{ r(s,a) + \sum_{j \in \mathcal{S}} \lambda p(j|s,a) u^{(n+1)}(j) \right\}$$

and stop. Here d_ε^∞ is a deterministic ε-optimal policy.

The Policy Iteration Algorithm

1. Set $n = 0$, and select an arbitrary decision rule $d_0 \in D$.
2. (Policy Evaluation) Obtain $\mathbf{u}^{(n)}$ by solving

$$(\mathbf{I} - \lambda P_{d_n})\mathbf{u} = \mathbf{r}_{d_n}.$$

3. (Policy Improvement) Choose d_{n+1} to satisfy

$$d_{n+1} \in \operatorname*{argmax}_{d \in D} \left\{ \mathbf{r}_d + \lambda P_d \mathbf{u}^{(n)} \right\}$$

   setting $d_{n+1} = d_n$ if some stopping criterion is met.
4. If $d_{n+1} = d_n$, stop and let $d_* = d_n$. Otherwise increment n by 1 and return to Step 2.

The Modified Policy Iteration

1. Select $\mathbf{u}^{(0)} \in \mathcal{U}$ such that $L\mathbf{u}^{(0)} - \mathbf{u}^{(0)} \geq 0$, specify $\varepsilon > 0$, and set $n = 0$.
2. (Policy Improvement) Choose d_{n+1} to satisfy

$$d_{n+1} \in \operatorname*{argmax}_{d \in D} \left\{ \mathbf{r}_d + \lambda P_d \mathbf{u}^{(n)} \right\},$$

   setting $d_{n+1} = d_n$ if some stopping criterion is met.
3. (Partial Policy Evaluation)
   (a) Set $k = 0$, and $\mathbf{u}_n^{(0)} = \max_{d \in D} \{ \mathbf{r}_d + \lambda P_d \mathbf{u}^{(n)} \}$.
   (b) If $\| \mathbf{u}_n^{(0)} - \mathbf{u}^{(n)} \| < \frac{1-\lambda}{2\lambda} \varepsilon$, go to Step 4. Otherwise go to (c).
   (c) if $k = m_n$, go to (e). Otherwise compute $\mathbf{u}_n^{(k+1)}$ by

$$\mathbf{u}_n^{(k+1)} = \mathbf{r}_{d_{n+1}} + \lambda P_{d_{n+1}} \mathbf{u}_n^{(k)}.$$

   (d) Increment k by 1 and return to (c).
   (e) Set $\mathbf{u}^{(n+1)} = \mathbf{u}_n^{(m_n)}$, increment n by 1, go to Step 2.
4. Set $d_\varepsilon = d_{n+1}$ and Stop.

The Q-Factor Functions

- Note that the Bellman equation is

$$u(s) = \max_{a \in \mathcal{A}_s} \left\{ r(s,a) + \lambda \sum_{j \in \mathcal{S}} p(j|s,a) u(j) \right\}$$

for $\lambda \in [0,1)$. The Q-factor function is defined by

$$Q(s,a) = r(s,a) + \lambda \sum_{j \in \mathcal{S}} p(j|s,a) \max_{b \in \mathcal{A}_j} Q(j,b)$$

and $u(s) = \max_{a \in \mathcal{A}_s} Q(s,a)$.

- For a given policy π, its value is

$$u^{\pi}(s) = r(s,\pi(s)) + \lambda \sum_{j \in \mathcal{S}} p(j|s,\pi(s)) u^{\pi}(j)$$

while $Q^{\pi}(s,a) = r(s,a) + \lambda \sum_{j \in \mathcal{S}} p(j|s,\pi(s)) Q^{\pi}(j,\pi(j))$ and $u^{\pi}(s) = Q^{\pi}(s,\pi(s))$.

Advantage Functions

- For a given policy π, its advantage value $A^\pi(s,a)$ is defined as

$$A^\pi(s,a) = Q^\pi(s,a) - u^\pi(s).$$

  Thus, $A^\pi(s,\pi(s)) = Q^\pi(s,\pi(s)) - u^\pi(s) = 0$.
- The advantage function of MDP satisfies

$$Q(s,a) = u(s) + A(s,a) - \max_{a \in \mathcal{A}_s} A(s,a).$$

The Q-Factor Value Iteration Algorithm

1. Select $Q^{(0)}$ (e.g., $Q^{(0)}(s,a)=0$), specify $\varepsilon>0$, and set $n=0$.
2. For each $s\in\mathcal{S}$ and $a\in\mathcal{A}_s$, compute

$$Q^{(n+1)}(s,a)\leftarrow\sum_{j\in\mathcal{S}}p(j|s,a)\big[r(s,a,j)+\lambda\max_{b\in\mathcal{A}_j}Q^{(n)}(j,b)\big].$$

3. Calculate for $s\in\mathcal{S}$,

$$u^{(n+1)}(s)=\max_{b\in\mathcal{A}_s}Q^{(n+1)}(s,b)\text{ and }u^{(n)}(s)=\max_{b\in\mathcal{A}_s}Q^{(n)}(s,b).$$

   Then if $\|\mathbf{u}^{(n+1)}-\mathbf{u}^{(n)}\|<\frac{1-\lambda}{2\lambda}\varepsilon$, go to Step 4. Otherwise increment n by 1 and return to Step 2.
4. For each $s\in\mathcal{S}$, choose

$$d_\varepsilon(s)=\operatorname*{argmax}_{a\in\mathcal{A}_s}Q^{(n+1)}(s,a)$$

   where d_ε^∞ is the ϵ-optimal policy, and Stop.

The Robbins-Monro Algorithm

- Let us denote the ith independent sample of a random variable X by x_i, and the expected value by $\mathbb{E}(X)$. Then

$$\mathbb{E}(X) = \lim_{n\to\infty} \frac{\sum_{i=1}^{n} x_i}{n}, \quad \text{with probability } 1.$$

- Let $X^{(n)} = \frac{\sum_{i=1}^{n} x_i}{n}$. Then the Robbins-Monro method is

$$X^{(n+1)} = X^{(n)} - \alpha_{n+1}(X^{(n)} - x_{n+1})$$

where $\alpha_{n+1} = \frac{1}{n+1}$. In general, we just set $\alpha_n \in (0,1)$ such that

$$\sum_{n=1}^{\infty} \alpha_n = \infty \text{ and } \sum_{n=1}^{\infty} \alpha_n^2 < \infty.$$

Maximization and Expectation

- Recall that the Bellman equation is

$$\begin{aligned} u(s) &= \max_{a \in \mathcal{A}_s} \sum_{j \in \mathcal{S}} p(j|s,a) \big[r(s,a,j) + \lambda u(j) \big] \\ &= \max(\mathbb{E}(\cdot)) \end{aligned}$$

while the Q-factor function is defined by

$$\begin{aligned} Q(s,a) &= \sum_{j \in \mathcal{S}} p(j|s,a) \big[r(s,a,j) + \lambda \max_{b \in \mathcal{A}_j} Q(j,b) \big] \\ &= \mathbb{E}\big[r(s,a,j) + \lambda \max_{b \in \mathcal{A}_j} Q(j,b) \big] \\ &= \mathbb{E}(Sample) = \mathbb{E}(\max(\cdot)) \end{aligned}$$

and $u(s) = \max_{a \in \mathcal{A}_s} Q(s,a)$.

Model-Free Scheme

The Robbins-Monro method leads to a model-free scheme as

$$Q^{(n+1)}(s,a) \leftarrow (1-\alpha_{n+1})Q^{(n)}(s,a) + \alpha_{n+1}\left[r(s,a,j) + \lambda \max_{b\in\mathcal{A}_j} Q^{(n)}(j,b)\right]$$

for each (s,a) pair.

Q-Learning

1. Initialize the Q-factors (e.g., $Q(s,a)=0$), specify $n_{\max}$, and set $n=0$.
2. Let the current state be s. Select action a with a probability of $1/|\mathcal{A}_s|$ (or other choices).
3. Simulate action a. Let the next state be j. Increment n by 1. Then update α.
4. Update $Q(s,a)$ by

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r(s,a,j) + \lambda \max_{b\in\mathcal{A}_j} Q(j,b)\right]$$

5. If $n < n_{\max}$, set $s \leftarrow j$, and go to Step 2. Otherwise go to Step 6.
6. For each $l \in \mathcal{S}$, choose

$$d(l) = \operatorname*{argmax}_{b\in\mathcal{A}_l} Q(l,b).$$

The policy generated by the algorithm is d^∞, and Stop.

SARSA

1. Initialize Q-factors (e.g., $Q(s,a)=0$).
2. For $t \leftarrow 0$ to T do
   - $s \leftarrow SelectState()$
   - $a \leftarrow SelectAction(d(Q), s)$ Policy d derived from Q
   - for each step of epoch t do
     - $r' \leftarrow r(s,a)$
     - $s' \leftarrow NextState(s,a)$
     - $a' \leftarrow SelectAction(d(Q), s')$
     - $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r' + \lambda Q(s',a'))$
     - $s \leftarrow s'$
     - $a \leftarrow a'$
3. Return Q

Off-Policy and On-Policy

- RL algorithms include two components: a learning policy, which determines the action to take, and an update rule, which defines the new estimate of the optimal value function.
- For an off-policy algorithm, the update rule does not necessarily depend on the learning policy. Q-learning is an off-policy algorithm because its update rule is based on the max operator. SARSA is an on-policy algorithm

Temporal Difference TD(0) or TD(γ)

Recall that for a given policy π, its value is

$$u^{\pi}(s) = r(s, \pi(s)) + \lambda \sum_{j} p(j|s, \pi(s)) u^{\pi}(j)$$
$$= \mathbb{E}(r(s, \pi(s)) + \lambda u^{\pi}(j)).$$

TD(0): One Step

1. Initialize u
2. For $t \leftarrow 0$ to T do
   - $s \leftarrow SelectState()$
   - for each step of epoch t do
     - $r' \leftarrow r(s, \pi(s))$
     - $s' \leftarrow NextState(\pi, s)$
     - $u(s) \leftarrow (1 - \alpha)u(s) + \alpha[r' + \lambda u(s')]$
     - $s \leftarrow s'$
3. Return u

Actor-Critics Framework

- "Actor" refers to the learned policy, and "Critic" refers to the learned value function.
- The (modified) policy iteration can be viewed as actor-critic. The Policy improvement step is viewed as the work of an actor, who takes into account the latest policy. The policy evaluation step is viewed as the work of a critic, who evaluates the performance of the current policy.
- AC uses action-selection probability to guide the search. For each state-action pair (s, a), one stores $H(s, a)$, a surrogate for the action-selection probability. An action a is selected in state s with

$$q(a|s) = \frac{\exp(H(s, a))}{\sum_{b \in \mathcal{A}_s} \exp(H(s, b))}$$

An Actor-Critics Algorithm

1. Initialize the u-values and H-values, specify $n_{\max}$, and set $n = 0$.
2. Let the current state be s. Select action a with probability of $q(a|s)$.
3. (Critic Update) Simulate action a. Let the next state be j, and let $r(s, a, j)$ be the immediate reward earned in going to j from s under a. Update $u(s)$ by
$$u(s) \leftarrow u(s) + \alpha\big[r(s, a, j) + \lambda u(j) - u(s)\big]$$
4. (Actor Update) Update $H(s, a)$ using
$$H(s, a) \leftarrow H(s, a) + \beta\big[r(s, a, j) + \lambda u(j) - u(s)\big]$$
5. If $n < n_{\max}$, increment n by 1, set $s \leftarrow j$, and then go to Step 2. Otherwise go to Step 6.
6. For each $l \in \mathcal{S}$, choose
$$d(l) = \operatorname*{argmax}_{b \in \mathcal{A}_l} H(l, b).$$

The policy generated by the algorithm is d^∞, and Stop.

Policy Gradient Methods

- In policy gradient methods, the policy can be parameterized as $q(a|s,\theta) = \Pr(A_t = a|S_t = s, \theta)$, as long as $q(a|s,\theta)$ is differentiable w.r..t its parameters θ.
- Let $J(\theta) = u^{q_\theta}(s)$. Then policy gradient theorem says that An action a is selected in state s in

$$\begin{aligned}
\nabla J(\theta) &\propto \sum_s \mu(s) \sum_a q(s,a) \nabla q(a|s,\theta) \\
&= \mathbb{E}\Big[\sum_a q(S_t,a) \nabla q(a|S_t,\theta)\Big] \\
&= \mathbb{E}\Big[\sum_a q(a|S_t,\theta) q(S_t,a) \frac{\nabla q(a|S_t,\theta)}{q(a|S_t,\theta)}\Big] \\
&= \mathbb{E}\big[q(S_t,A_t) \nabla \ln q(A_t|S_t,\theta)\big].
\end{aligned}$$

REINFORCE

1. Input a differentiable policy parameterization $q(a|s,\theta)$, and initialize policy parameter θ.
2. For each episode
   - Generate $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $q(\cdot|\cdot,\theta)$
   - Fo each step of the episode $t = 0, 1, \ldots, T-1$:
     - $G \leftarrow \sum_{k=t+1}^{T} \lambda^{k-t-1} R_k$
     - $\theta \leftarrow \theta + \alpha \lambda^t G \nabla \ln q(A_t|S_t,\theta)$

REINFORCE with Baseline

- Note that

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a (q(s,a) - b(s)) \nabla q(a|s,\theta)$$

because

$$\sum_a b(s) \nabla q(a|s,\theta) = b(s) \nabla \sum_a q(a|s,\theta) = b(s) \nabla 1 = 0.$$

- A new version of REINFORCE with a baseline is

$$\theta_{t+1} = \theta_t + \alpha (G_t - b(S_t)) \nabla \ln q(A_t|S_t,\theta).$$

The Curse of Dimensionality

- Each Q-function is stored individually in a table in the computer's memory. For large state-action spaces, look-up tables are ruled out.
- Function approximation is instead employed. That is, $Q(s,a)$ is updated by using function approximation, and we use the following definition for $Q(s,a)$:

$$Q(s,a) = \sum_{j \in \mathcal{S}} p(j|s,a) \left[r(s,a,j) + \lambda \max_{b \in \mathcal{A}_j} Q_w(j,b) \right],$$

where Q_w can be defined by regression and neural networks.

The Function Fitting

- The so-called Bellman error (BE) is defined as

$$BE = \frac{1}{2}\sum_{(s,a)}[Q(s,a) - Q_w(s,a)]^2.$$

- The derivative of BE w.r.t. w is

$$\begin{aligned}
\frac{\partial BE}{\partial w(l,a)} &= -\sum_{(s,a)} \frac{\partial Q_w(s,a)}{\partial w(l,a)}[Q(s,a) - Q_w(s,a)] \\
&= -\sum_{(s,a)} \frac{\partial Q_w(s,a)}{\partial w(l,a)}\Big\{\sum_{j\in\mathcal{S}} p(j|s,a)\big[r(s,a,j) \\
&\quad + \lambda \max_{b\in\mathcal{A}_j} Q_w(j,b) - Q_w(s,a)\big]\Big\} \\
&= -\mathbb{E}\Big\{\sum_{(s,a)} \frac{\partial Q_w(s,a)}{\partial w(l,a)}\big[r(s,a,j)+\lambda \max_{b\in\mathcal{A}_j} Q_w(j,b)-Q_w(s,a)\big]\Big\}.
\end{aligned}$$

The Function Fitting

- The derivative of BE w.r.t. w is

$$\frac{\partial BE}{\partial w(l,a)}$$
$$= -\sum_{(s,a)} \frac{\partial Q_w(s,a)}{\partial w(l,a)} \left[r(s,a,j) + \lambda \max_{b \in \mathcal{A}_j} Q_w(j,b) - Q_w(s,a) \right] \text{(or)}$$
$$= -\frac{\partial Q_w(s,a)}{\partial w(l,a)} \left[r(s,a,j) + \lambda \max_{b \in \mathcal{A}_j} Q_w(j,b) - Q_w(s,a) \right].$$

- The update of $w(l,a)$ is

$$w(l,a) \leftarrow w(l,a) - \alpha \frac{\partial BE}{\partial w(l,a)}.$$

Deep Q-Networks

- The Q-values are parameterized with a neural network $Q(s, a; w)$. Then an approximation of the Q-value at the kth iteration $Q(s, a; w_k)$ is updated towards the target value

$$Y_k^Q = r + \lambda \max_{b \in \mathbf{A}_j} Q(j, b; w_k).$$

- The Q-learning update amounts in updating the parameters

$$w_{k+1} = w_k + \alpha[Y_k^Q - Q(s, a; w_k)] \nabla_{w_k} Q(s, a; w_k).$$

DQN and Variants

- In the DQN (Mnih et al., 2015), the target value is

$$Y_k^{DQN} = r + \lambda \max_{b \in \mathbf{A}_j} Q(j, b; w_k^-),$$

where its parameters w_k^- are updated only every C iterations with the following assignment: $w_k^- = w_k$.

- In Double DQN (DDQN) (Van Hasselt et al., 2016), the target value Y_k^Q is replaced by

$$Y_k^{DDQN} = r + \lambda Q(j, \operatorname*{argmax}_{b \in \mathbf{A}_j} Q(j, b; w_k); w_k^-).$$

- In the Dueling Network (Wang et al., 2015), it is based on the advantage function $A^\pi(s, a)$.

Policy Gradient Methods for Deep RL

- Deep Deterministic Policy Gradient DDPG (Silver et al., 2014)
- The Natural Policy Gradients
- Trust region Optimization TRPO (Schulman et al., 2015)
- Proximal Policy Optimization PPO (Schulman et al., 2017)
- Entropy-Regularized Optimization

Related Models

- Hidden Markov Models (HMM)
- Markov Decision Processes (MDPs)
- Partially Observed MDPs (POMDPs)

Potential Issues

- Bandits vs. MDPs
- Stochastic Optimization vs Simulation
- Multiple Agents vs. Games
- Rewards vs Meta Learning

Thanks

Thanks!

Comments & Questions?