Introduction to RL

Reference

- FML Chapter 14
- Sutton and Barto - Reinforcement Learning

Outline

1. Introduction
2. Bellman Equations
3. Temporal Difference (TD) Methods
4. Function Approximation for Value Functions
5. Actor-critic Methods
6. Deep Reinforcement Learning

Outline

1. Introduction
2. Bellman Equations
3. Temporal Difference (TD) Methods
4. Function Approximation for Value Functions
5. Actor-critic Methods
6. Deep Reinforcement Learning

Bellman Equation (under Markov Property)

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R_t \mid s_t = s] = \mathbb{E}_{\pi}[r_{t+1} + \gamma R_{t+1} \mid s_t = s]$$

$$= \underbrace{\mathbb{E}_{\pi}[r_{t+1} \mid s_t = s]}_{\text{(A)}} + \gamma \underbrace{\mathbb{E}_{\pi}[R_{t+1} \mid s_t = s]}_{\text{(B)}}$$

$$\text{(A)} = \sum_r r p(r \mid s) = \sum_r r \sum_{s',a} p(r, a, s' \mid s)$$

$$= \sum_r r \sum_{s',a} p(s', r \mid s, a)\pi(a \mid s)$$

$$= \sum_r r \sum_a \pi(a \mid s) \sum_{s'} p(s', r \mid s, a)$$

$$= \sum_a \pi(a \mid s) \sum_r \sum_{s'} p(s', r \mid s, a)\ r$$

$$\text{(B)} = \mathbb{E}_{\pi}[R_{t+1} \mid s_t = s]$$

$$= \sum_{s'} \mathbb{E}_{\pi}[R_{t+1} \mid s_{t+1} = s'] \Pr(s_{t+1} = s' \mid s_t = s)$$

$$= \sum_{s'} \sum_r V^{\pi}(s') \Pr(s_{t+1} = s', r_{t+1} = r \mid s_t = s)$$

$$= \sum_{s'} \sum_r V^{\pi}(s') \sum_a \Pr(s_{t+1} = s', r_{t+1} = r \mid s_t = s, a_t = a)\pi(a \mid s)$$

$$= \sum_a \pi(a \mid s) \sum_r \sum_{s'} p(s', r \mid s, a)\ V^{\pi}(s')$$

Therefore $V^{\pi}(s) = \sum_a \pi(a \mid s) \sum_r \sum_{s'} p(s', r \mid s, a)[r + \gamma v^{\pi}(s')]$

Bellman Equation (under Markov Property)

$$
\begin{aligned}
V^{\pi}(s) &= \sum_{a} \pi(a \mid s) \sum_{r} \sum_{s'} p(s', r \mid s, a)[r + \gamma V^{\pi}(s')] \\
&= \sum_{a} \pi(a \mid s) \Big[\sum_{r} p(r \mid s, a) r + \gamma \sum_{s'} p(s' \mid s, a) V^{\pi}(s') \Big] \\
&= \sum_{a} \pi(a \mid s) \Big[\mathbb{E}[r \mid s, a] + \gamma \sum_{s'} p(s' \mid s, a) V^{\pi}(s') \Big]
\end{aligned}
$$

$$
\begin{aligned}
Q^{\pi}(s, a) &= \sum_{r} \sum_{s'} p(s', r \mid s, a)[r + \gamma V^{\pi}(s')] \\
&= \mathbb{E}[r \mid s, a] + \gamma \sum_{s'} p(s' \mid s, a) V^{\pi}(s') \\
&= \mathbb{E}[r \mid s, a] + \gamma \sum_{s'} p(s' \mid s, a) \sum_{a'} \pi(a' \mid s') Q^{\pi}(s', a')
\end{aligned}
$$

Optimal Value Function

$$V^\star(s) = \max_\pi V^\pi(s) = \max_\pi \mathbb{E}[R_t \,|\, s_t = s]$$

$$Q^\star(s,a) = \max_\pi Q^\pi(s,a) = \max_\pi \mathbb{E}[R_t \,|\, s_t = s, a_t = a]$$

Bellman Optimality Equation

$$V^\star(s) = \max_a \sum_{r,s'} p(s', r \,|\, s, a)(r + \gamma V^\star(s'))$$

$$Q^\star(s,a) = \sum_{r,s'} p(s', r \,|\, s, a)(r + \gamma \max_{a'} Q^\star(s', a'))$$

RL Diagram

- Know transition dynamics $P(r_{t+1} = r', s_{t+1} = s' \mid s_t = s, a_t = a)$
  - Dynamic Programming
    - Policy Iteration

$$\begin{cases} \text{Policy evaluation} & V_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_k(s')] \\ \text{Plicy improvement} & \pi(a \mid s) = \arg\max_a Q^\pi(s, a) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V^\pi(s')] \end{cases}$$

    - Value Iteration

$$V_{k+1}(s) = \max_a \sum_{s',r} p(s', r \mid s, a)(r + \gamma V_k(s'))$$ no policy involved

RL Diagram

- Unknown transition dynamics $P(r_{t+1} = r', s_{t+1} = s' \mid s_t = s, a_t = a)$
  - Monte Carlo Prediction — Simulate $V^\pi(s), \forall s.$

    Let $V^\pi(s) \in [0, T]$. Using Hoeffding's inequality, given n trajectories/ samples for each state s

$$\Pr\left(\frac{1}{n}\sum_{i=1}^{n} \hat{V}^\pi(s) - V^\pi(s) \geq \epsilon\right) \leq e^{-\frac{2\epsilon^2}{nT^2}}$$

  - Estimate $Q^\pi(s, a)$ to do planning

On Policy and Off Policy

On Policy

Environment react to a policy μ

$$\begin{cases} \text{Evaluate } \mu \\ \text{Improve } \mu \end{cases}$$

Off Policy

Observe $\{(s_t, a_t, r_{t+1})\}_{t=1}^{T}$

generated by a behavior policy μ

However, goal is to

$$\begin{cases} \text{Evaluate } \pi \\ \text{Improve } \pi \end{cases}$$

without executing a target policy π

RL - TD Methods

Reference

- FML Chapter 14
- Sutton and Barto - Reinforcement Learning

Outline

1. Introduction
2. Bellman Equations
3. Temporal Difference (TD) Methods
4. Function Approximation for Value Functions
5. Actor-critic Methods
6. Deep Reinforcement Learning

Temporal Difference Learning

We will introduce

- SARSA: On-policy TD control
- Q-learning: Off-policy TD control

Introduction of TD Learning

TD methods are similar to

- Dynamic programming

  since TD methods update estimates based in part on other learned estimates, without waiting for the final outcome

- Monte Carlo methods

  since TD methods learn directly from raw experience without a model of the environment's dynamics

TD prediction

- TD methods only wait until the next time step to update the value estimates
- At time $t+1$ they immediately form a target and make an update using the observed reward r_{t+1} and the current estimate $V(s_{t+1})$:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$ where α is the step size.

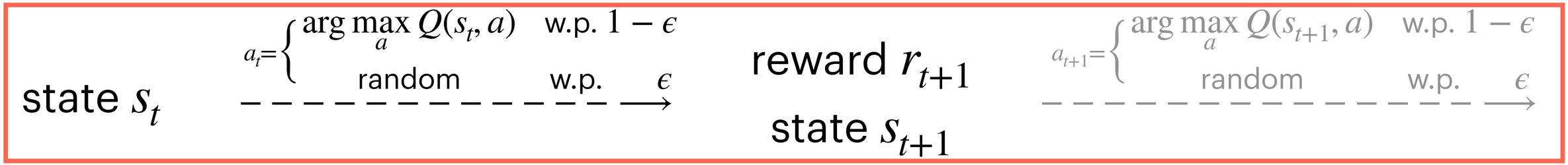
Notice that

- if the estimation is correct, then $\mathbb{E}[r_{t+1}] = V(s_t) - \gamma V(s_{t+1})$.
- Similar to MC update except that it takes place at every step
- Similar to DP methods, the TD method bases its update in part on an existing estimate — a bootstrapping method.

TD Error

- TD error is $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$, the error made in the estimate at time t.
- TD error δ_t at time t depends on the next state and the next reward, therefore, δ_t is not available until step $t + 1$.
- Updating the value function with the TD error is called a **backup**.
- Related to Bellman Equation.
  - **Group Discussion**

SARSA: On-policy TD Control



- On policy: execute an ϵ-**greedy** policy, evaluate an ϵ-**greedy** policy
- Balances between exploration and exploitation using ϵ-greedy policy
- Learns Q-function of an ϵ-greedy policy

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big)$$

- This update is done after every transition from a non-terminal state s_t. If s_{t+1} is terminal, $Q(s_{t+1}, a_{t+1})$ is defined as zero.

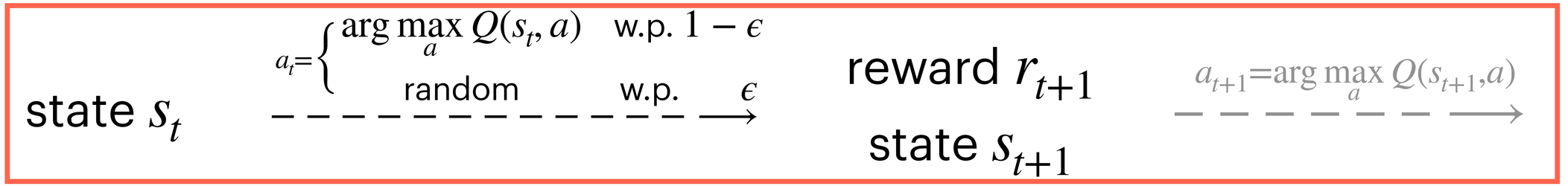
SARSA: On-policy TD Control

Algorithm 1 SARSA

Initialise Q arbitrarily, $Q(terminal, \cdot) = 0$
repeat
  Initialize s
  Choose a ϵ-greedily
  repeat
    Take action a, observe r, s'
    Choose a' ϵ-greedily
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma Q(s',a') - Q(s,a)\right)$
    $s \leftarrow s', a \leftarrow a'$
  until s is terminal
until convergence

- Maintain a table of $Q(s,a)$ values, $\forall(s,a)$.

Q-learning: Off-Policy TD Control



- Off policy: execute an ϵ-**greedy** policy μ, evaluate a **greedy** policy π
- Balances between exploration and exploitation using ϵ-greedy policy
- Learns the optimal Q-function

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right)$$

- This update is done after every transition from a non-terminal state s_t. If s_{t+1} is terminal, $Q(s_{t+1}, a_{t+1})$ is defined as zero.

Q-learning: Off-Policy TD Control

Algorithm 2 Q-learning

Initialise Q arbitrarily, $Q(terminal, \cdot) = 0$
repeat
  Initialize s
  repeat
    Choose a ϵ-greedily
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$
    $s \leftarrow s'$
  until s is terminal
until convergence

- Maintain a table of $Q(s, a)$ values, $\forall(s, a)$.

Expected SARSA: Variance Reduction

state s_t $\quad a_t = \begin{cases} \arg\max_a Q(s_t, a) & \text{w.p. } 1-\epsilon \\ \text{random} & \text{w.p. } \epsilon \end{cases}$ $\quad$ reward r_{t+1}, state s_{t+1} $\quad a_{t+1} = \begin{cases} \arg\max_a Q(s_{t+1}, a) & \text{w.p. } 1-\epsilon \\ \text{random} & \text{w.p. } \epsilon \end{cases}$

- SARSA - Update Q-function

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \Big)$$

- Expected SARSA - Update Q-function

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(r_{t+1} + \gamma \sum_{a'} \pi(a' \mid s_{t+1}) Q(s_{t+1}, a') - Q(s_t, a_t) \Big)$$

- Expected sparse evaluates the Q values for π - can be either on- or off-policy
- Q-learning is a special case of expected SARSA where π is a greedy policy

Outline

1. Introduction
2. Bellman Equations
3. Temporal Difference (TD) Methods
4. Function Approximation for Value Functions
5. Actor-critic Methods
6. Deep Reinforcement Learning

RL - Function Approximations

Outline

1. Introduction
2. Bellman Equations
3. Temporal Difference (TD) Methods
4. Function Approximation for Value Functions
5. Actor-critic Methods
6. Deep Reinforcement Learning

Function Approximation for Value Functions

We will introduce

- Value-function approximation
- Gradient methods
- Linear approximation and least-squares TD
- Policy gradient methods
- REINFORCE

Value Function Approximation

Value-function Approximation

- Representing value function as a table is not possible for large state spaces or continuous state spaces
- Parameterized function approximation with weight vector $\theta \in \mathbb{R}^n$

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

  - Dimension of θ is much less than the number of states ($n \ll |\mathrm{S}|$)

Quality of the Value-function Approximation

- Mean Squared Value Error MSVE(θ)

$$\text{MSVE}(\theta) = \sum_s d^\pi(s)\Big(V^\pi(s) - \hat{V}^\pi(s,\theta)\Big)^2$$

- Typically choose $d^\pi(s)$ to be the occupancy frequency — the fraction of time spent in s under the target policy π.
  - Let $h^\pi(s)$ denote the probability that an episode begins in state s
  - Let $e^\pi(s)$ denote the average time steps spent in state s in a single episode
  - We solve the system of equations $e^\pi(s) = h^\pi(s) + \sum_{s'} e^\pi(s') \sum_a \pi(a \mid s') p(s \mid s', a)$ to obtain $e^\pi(s)$
  - Then, $d^\pi(s) = \dfrac{e^\pi(s)}{\sum_{s'} e^\pi(s')}$

Gradient Method to Minimize MSVE(θ)

- If $\hat{V}^\pi(s, \theta)$ is differentiable w.r.t. $\theta = [\theta_1, \theta_2, \ldots, \theta_n]^\top$, we then update θ as follows

$$\theta_{t+1} = \theta_t - \frac{1}{2}\alpha \nabla_\theta \Big(V^\pi(s_t) - \hat{V}^\pi(s_t, \theta_t) \Big)^2$$
$$= \theta_t - \frac{1}{2}\alpha \Big(V^\pi(s_t) - \hat{V}^\pi(s_t, \theta_t) \Big) \nabla_\theta \hat{V}^\pi(s_t, \theta_t)$$

- A sample $s_t \mapsto V^\pi(s_t)$ consists of a state s_t and its true value under policy π
- We assume the states appear in examples with the same $d(s)$ distribution
- In practice, true value not available. Solution: use a **noisy** estimate. If unbiased estimator is used, θ_t is guaranteed to converge to a local optima for decreasing α.
- How to estimate the true value? Monte Carlo: $\hat{V}^\pi(s_t) = R_t$

Semi-gradient Methods

- Instead of MC, we use TD or DP updates for prediction using SGD, i.e., we perform bootstrapping, based on current value of the weight vector θ_t
- The prediction will be biased, not a true gradient-based method

Example - Linear Approximation

- Linear function approximation $\hat{V}^{\pi}(s, \theta) = \theta^{\top}\phi(s)$ where $\phi(s)$ is the feature function.
- Recall the SGD update $\theta_{t+1} = \theta_t - \frac{1}{2}\alpha\Big(V^{\pi}(s_t) - \hat{V}^{\pi}(s_t, \theta_t)\Big)\nabla_{\theta}\hat{V}^{\pi}(s_t, \theta_t)$
- Under linear function approximation, $\nabla_{\theta}\hat{V}^{\pi}(s_t, \theta_t) = \phi(s)$
- Therefore the SGD update for linear function approximation is

$$\theta_{t+1} = \theta_t - \frac{1}{2}\alpha\Big(V^{\pi}(s_t) - \theta_t^{\top}\phi(s_t)\Big)\phi(s_t)$$

Semi-gradient TD update for Linear Function Approximation

- We use TD updates to estimate the true value $V^{\pi}(s_t)$

$$\theta_{t+1} = \theta_t - \frac{1}{2}\alpha\Big(V^{\pi}(s_t) - \theta_t^{\top}\phi(s_t)\Big)\phi(s_t)$$

$$= \theta_t - \frac{1}{2}\alpha\Big(r_{t+1} + \gamma\hat{V}^{\pi}(s_{t+1}, \theta_t) - \theta_t^{\top}\phi(s_t)\Big)\phi(s_t)$$

$$= \theta_t - \frac{1}{2}\alpha\Big(r_{t+1} + \gamma\theta_t^{\top}\phi(s_{t+1}) - \theta_t^{\top}\phi(s_t)\Big)\phi(s_t)$$

$$= \theta_t - \frac{1}{2}\alpha\Big(r_{t+1}\phi(s_t) - \phi(s_t)\big(\phi(s_t) - \gamma\phi(s_{t+1})\big)^{\top}\theta_t\Big)$$

- Once the system has reached steady state, for any given θ_t , the expected next weight vector is $\mathbb{E}[\theta_{t+1} \mid \theta_t] = \theta_t + \alpha(\mathbf{b} - A\theta_t)$ where $\begin{cases} \mathbf{b} = \mathbb{E}[r_{t+1}\phi(s_t)] \\ A = \mathbb{E}[\phi(s_t)(\phi(s_t) - \gamma\phi(s_{t+1}))^{\top}] \end{cases}$

- If the system converges to θ, then $\mathbf{b} - A\theta = \mathbf{0}$

Semi-gradient TD update for Linear Function Approximation

- Least-Squares TD
  - TD with linear function approximation converges asymptotically, for appropriate decreasing step sizes, to the TD fix point:

$$\theta = A^{-1}\mathbf{b}$$

$$A = \mathbb{E}\left[\phi(s_t)\left(\phi(s_t) - \gamma\phi(s_{t+1})\right)^\top\right]$$

$$\mathbf{b} = \mathbb{E}[r_{t+1}\phi(s_t)]$$

  - Therefore, we don't need to compute the solution iteratively. Instead, we can calculate A and $\mathbf{b}$ and then find the fix point.

Policy Function Approximation

Policy Gradient Methods

- Policy gradient methods learn a parameterized policy that can select actions without needing to compute a value function
- Policy π is parameterized with $\omega \in \mathbb{R}^n$

$$\pi(a \mid s, \omega) = p(a_t = a \mid s_t = s, \omega_t = \omega)$$

For instance, a Gibbs policy $\pi(a \mid s, \omega) = \frac{\exp(\omega^\top \psi(s,a))}{\sum_{a'} \exp(\omega^\top \psi(s,a'))}$ where $\psi(s,a)$ denotes the feature functions.

- Given a performance measure $J(\omega)$, the gradient is

$$\omega_{t+1} = \omega_t + \alpha \nabla_\omega J(\omega_t)$$

- $J(\omega)$ is typically $\sum_s d^\pi(s)\ V^{\pi(\omega)}(s)$.

Compared with Value-based Methods

- Pros: better convergence properties, effective in high-dim or continuous action space, can learn stochastic policies
- Cons: usually converge to local optimum, inefficient to evaluate, high variance

Policy Gradient Theorem

- Reward $J(\omega) = \sum_s d^{\pi_\omega}(s) V^{\pi_\omega}(s) = \sum_s d^{\pi_\omega}(s) \sum_a Q^{\pi_\omega}(s,a) \pi_\omega(a \mid s)$

where d^{π_ω} is the on-policy distribution under π, i.e., the stationary distribution of Markov chain for π_ω: $d^{\pi_\omega}(s) = \lim_{t \to \infty} P(s_t = s \mid s_0, \pi_\omega)$

Theorem $\nabla_\omega J(\omega) \propto \sum_s d^{\pi_\omega}(s) \sum_a Q^{\pi_\omega}(s,a) \nabla_\omega \pi_\omega(a \mid s)$

Provides a nice reformation of the derivative of the objective function to not involve the derivative of the state distribution or the Q

Policy Gradient Theorem: Proof Sketch

(1) A recursive form of derivative of the state value fn.

$$\nabla_w V^\pi(s) = \sum_{a \in A} \left(\nabla_w \pi_w(a|s) Q^\pi(s,a) + \pi_w(a|s) \sum_{s'} P(s'|s,a) \nabla_\theta V^\pi(s') \right)$$

(2) Unroll the recursive representation of $\nabla_w V^\pi(s)$

$$\nabla_w V^\pi(s) = \sum_{x \in S} \sum_{k=0}^{\infty} \rho^\pi(s \to x, k) \phi(x)$$

where $\phi(x) = \sum_{a \in A} Q^\pi(x,a) \nabla_w \pi_w(a|x)$

(3) A form excluding the derivative of Q-value fn

$$\nabla_w J(w) \propto \sum_s d^\pi(s) \phi(s)$$

Policy Gradient Theorem: Proof Details (1)

$$
\begin{aligned}
&\nabla_\theta V^\pi(s) \\
=&\nabla_\theta \Big(\sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a) \Big) \\
=& \sum_{a \in \mathcal{A}} \Big(\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) + \pi_\theta(a|s) \nabla_\theta Q^\pi(s,a) \Big) && \text{; Derivative product rule.} \\
=& \sum_{a \in \mathcal{A}} \Big(\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) + \pi_\theta(a|s) \nabla_\theta \sum_{s',r} P(s',r|s,a)(r + V^\pi(s')) \Big) && \text{; Extend } Q^\pi \text{ with future state value.} \\
=& \sum_{a \in \mathcal{A}} \Big(\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) + \pi_\theta(a|s) \sum_{s',r} P(s',r|s,a) \nabla_\theta V^\pi(s') \Big) && P(s',r|s,a) \text{ or } r \text{ is not a func of } \theta \\
=& \sum_{a \in \mathcal{A}} \Big(\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) + \pi_\theta(a|s) \sum_{s'} P(s'|s,a) \nabla_\theta V^\pi(s') \Big) && \text{; Because } P(s'|s,a) = \textstyle\sum_r P(s',r|s,a)
\end{aligned}
$$

Policy Gradient Theorem: Proof Details (2)

* Def: k-step transition probability

the probability of transitioning from state s with policy π_w after k step: $\rho^{\pi}(s \to x, k)$

$$s \xrightarrow{a \sim \pi_w(\cdot|s)} s' \xrightarrow{a \sim \pi_w(\cdot|s')} s'' \quad \cdots\cdots$$

$$\rho^{\pi}(s \to x, k+1) = \sum_{s'} \rho^{\pi}(s \to s', k)\rho^{\pi}(s' \to x, 1)$$

Policy Gradient Theorem: Proof Details (2)

$$
\begin{aligned}
&\nabla_\theta V^\pi(s) \\
=&\phi(s) + \sum_a \pi_\theta(a|s) \sum_{s'} P(s'|s,a) \nabla_\theta V^\pi(s') \\
=&\phi(s) + \sum_{s'} \sum_a \pi_\theta(a|s) P(s'|s,a) \nabla_\theta V^\pi(s') \\
=&\phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) \nabla_\theta V^\pi(s') \\
=&\phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) \nabla_\theta V^\pi(s') \\
=&\phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) [\phi(s') + \sum_{s''} \rho^\pi(s' \to s'', 1) \nabla_\theta V^\pi(s'')] \\
=&\phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) \phi(s') + \sum_{s''} \rho^\pi(s \to s'', 2) \nabla_\theta V^\pi(s'') \text{ ; Consider } s' \text{ as the middle point for } s \to s'' \\
=&\phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) \phi(s') + \sum_{s''} \rho^\pi(s \to s'', 2) \phi(s'') + \sum_{s'''} \rho^\pi(s \to s''', 3) \nabla_\theta V^\pi(s''') \\
=&\ldots \text{; Repeatedly unrolling the part of } \nabla_\theta V^\pi(.) \\
=&\sum_{x \in S} \sum_{k=0}^{\infty} \rho^\pi(s \to x, k) \phi(x)
\end{aligned}
$$

Policy Gradient Theorem: Proof Details (3)

$$\nabla_\theta J(\theta) = \nabla_\theta V^\pi(s_0) \qquad \text{; Starting from a random state } s_0$$

$$= \sum_s \sum_{k=0}^{\infty} \rho^\pi(s_0 \to s, k)\phi(s) \qquad \text{; Let } \eta(s)=\textstyle\sum_{k=0}^{\infty} \rho^\pi(s_0 \to s,k)$$

$$= \sum_s \eta(s)\phi(s)$$

$$= \left(\sum_s \eta(s)\right) \sum_s \frac{\eta(s)}{\sum_s \eta(s)}\phi(s) \qquad \text{; Normalize } \eta(s), s\in\mathcal{S} \text{ to be a probability distribution.}$$

$$\propto \sum_s \frac{\eta(s)}{\sum_s \eta(s)}\phi(s) \qquad \textstyle\sum_s \eta(s) \text{ is a constant}$$

$$= \sum_s d^\pi(s) \sum_a \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) \qquad d^\pi(s)=\frac{\eta(s)}{\sum_s \eta(s)} \text{ is stationary distribution.}$$

Policy Gradient Theorem: Proof Details (3)

In the episodic case, the constant of proportionality ($\sum_s \eta(s)$) is the average length of an episode; in the continuing case, it is 1 (Sutton & Barto, 2017; Sec. 13.2). The gradient can be further written as:

$$
\begin{aligned}
\nabla_\theta J(\theta) &\propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s, a) \nabla_\theta \pi_\theta(a|s) \\
&= \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\
&= \mathbb{E}_\pi[Q^\pi(s, a) \nabla_\theta \ln \pi_\theta(a|s)] \qquad \text{; Because } (\ln x)' = 1/x
\end{aligned}
$$

Where $\mathbb{E}_\pi$ refers to $\mathbb{E}_{s \sim d_\pi, a \sim \pi_\theta}$ when both state and action distributions follow the policy π_θ (on policy).

REINFORCE

- Policy Gradient Theorem $\nabla_\omega J(\omega) = \sum_s d^\pi(s) \sum_a Q^\pi(s,a) \nabla_\omega \pi(a \mid s, \omega)$ gives us an exact expression for the gradient. We need a sampling method whose expectation equals or approximates this expression.
- We know that

$$\mathbb{E}_{\Pr(s_{t+1}, r_{t+1} \mid s_t), \pi(a_t \mid s_t, \omega), s_t}\left[\gamma^t R_t \frac{\nabla_\omega \pi(a_t \mid s_t, \omega)}{\pi(a_t \mid s_t, \omega)}\right] = \mathbb{E}_{s_t}\left[Q^\pi(s_t, \pi(a_t \mid s_t, \omega)) \frac{\nabla_\omega \pi(a_t \mid s_t, \omega)}{\pi(a_t \mid s_t, \omega)}\right]$$

$$= \sum_s d^\pi(s) \sum_a Q^\pi(s,a) \frac{\nabla_\omega \pi(a \mid s, \omega)}{\pi(a \mid s, \omega)} \pi(a \mid s, \omega) = \nabla_\omega J(\omega)$$

- Therefore, $\nabla_\omega J(\omega)$ can be evaluated through $\mathbb{E}_\pi[\gamma^t R_t \frac{\nabla_\omega \pi(a_t \mid s_t, \omega)}{\pi(a_t \mid s_t, \omega)}]$ ———— REINFORCE

REINFORCE

- Evaluate $\nabla_\omega J(\omega)$ through

$$\mathbb{E}_\pi[\gamma^t R_t \frac{\nabla_\omega \pi(a_t \mid s_t, \omega)}{\pi(a_t \mid s_t, \omega)}] = \mathbb{E}_\pi[\gamma^t R_t \nabla_\omega \log \pi(a_t \mid s_t, \omega)]$$

- Update the policy through

$$\omega_{t+1} = \omega_t + \alpha \gamma^t R_t \nabla_\omega \log \pi(a_t \mid s_t, \omega)$$

- In case π is a Gibbs policy

$$\nabla \log \pi(a_t \mid s_t, \omega) = \psi(s_t, a_t) - \sum_{a'} \pi(a' \mid s_t, \omega) \psi(s_t, a')$$

REINFORCE

Algorithm 3 REINFORCE

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\omega})$, $\alpha > 0$
Initialise $\boldsymbol{\omega}$
repeat
    Generate an episode $s_0, a_0, r_1, \cdots, s_T, a_T$ following $\pi(\cdot|\cdot, \boldsymbol{\omega})$
    for each step $t = 0, \cdots, T$ **do**
        $R_t \leftarrow$ return from step t
        $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \alpha\gamma^t R_t \nabla \log \pi(a|s_t, \boldsymbol{\omega})$
    end for
until convergence

- An episodic Monte-Carlo Policy-Gradient Method

Off-Policy Policy Gradient

1. The off-policy approach does not require full trajectories and can reuse any past episodes ("experience replay") for much better sample efficiency.
2. The sample collection follows a behavior policy different from the target policy, bringing better exploration.

Now let's see how off-policy policy gradient is computed. The behavior policy for collecting samples is a known policy (predefined just like a hyperparameter), labelled as $\beta(a|s)$. The objective function sums up the reward over the state distribution defined by this behavior policy:

$$J(\theta) = \sum_{s \in \mathcal{S}} d^{\beta}(s) \sum_{a \in \mathcal{A}} Q^{\pi}(s, a) \pi_{\theta}(a|s) = \mathbb{E}_{s \sim d^{\beta}} \Big[\sum_{a \in \mathcal{A}} Q^{\pi}(s, a) \pi_{\theta}(a|s) \Big]$$

where $d^{\beta}(s)$ is the stationary distribution of the behavior policy β; recall that $d^{\beta}(s) = \lim_{t \to \infty} P(S_t = s | S_0, \beta)$; and Q^{π} is the action-value function estimated with regard to the target policy π (not the behavior policy!).

Off-Policy Policy Gradient

Given that the training observations are sampled by $a \sim \beta(a|s)$, we can rewrite the gradient as:

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{s\sim d^\beta}\Big[\sum_{a\in\mathcal{A}} Q^\pi(s,a)\pi_\theta(a|s)\Big] \\
&= \mathbb{E}_{s\sim d^\beta}\Big[\sum_{a\in\mathcal{A}} \big(Q^\pi(s,a)\nabla_\theta\pi_\theta(a|s) + \pi_\theta(a|s)\nabla_\theta Q^\pi(s,a)\big)\Big] && \text{; Derivative product rule.} \\
&\overset{(i)}{\approx} \mathbb{E}_{s\sim d^\beta}\Big[\sum_{a\in\mathcal{A}} Q^\pi(s,a)\nabla_\theta\pi_\theta(a|s)\Big] && \text{; Ignore the red part: } \pi_\theta(a|s)\nabla_\theta Q^\pi(s,a). \\
&= \mathbb{E}_{s\sim d^\beta}\Big[\sum_{a\in\mathcal{A}} \beta(a|s)\frac{\pi_\theta(a|s)}{\beta(a|s)} Q^\pi(s,a)\frac{\nabla_\theta\pi_\theta(a|s)}{\pi_\theta(a|s)}\Big] \\
&= \mathbb{E}_\beta\Big[\frac{\pi_\theta(a|s)}{\beta(a|s)} Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)\Big] && \text{; The blue part is the importance weight.}
\end{aligned}
$$

where $\frac{\pi_\theta(a|s)}{\beta(a|s)}$ is the importance weight.

In summary, when applying policy gradient in the off-policy setting, we can simple adjust it with a weighted sum and the weight is the ratio of the target policy to the behavior policy,

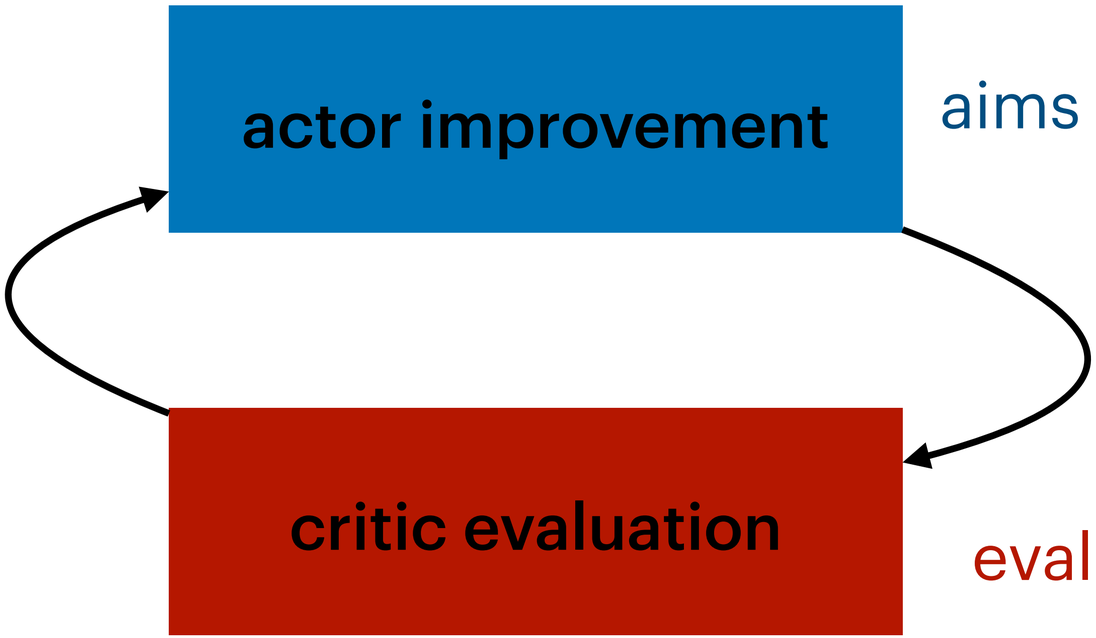
Outline

1. Introduction
2. Bellman Equations
3. Temporal Difference (TD) Methods
4. Function Approximation for Value Functions
5. Actor-critic Methods
6. Deep Reinforcement Learning

RL -Actor Critic & Intro to DRL

Actor-critic Methods

- A generalized policy iteration, **alternating** between a policy evaluation and a policy improvement step.
-

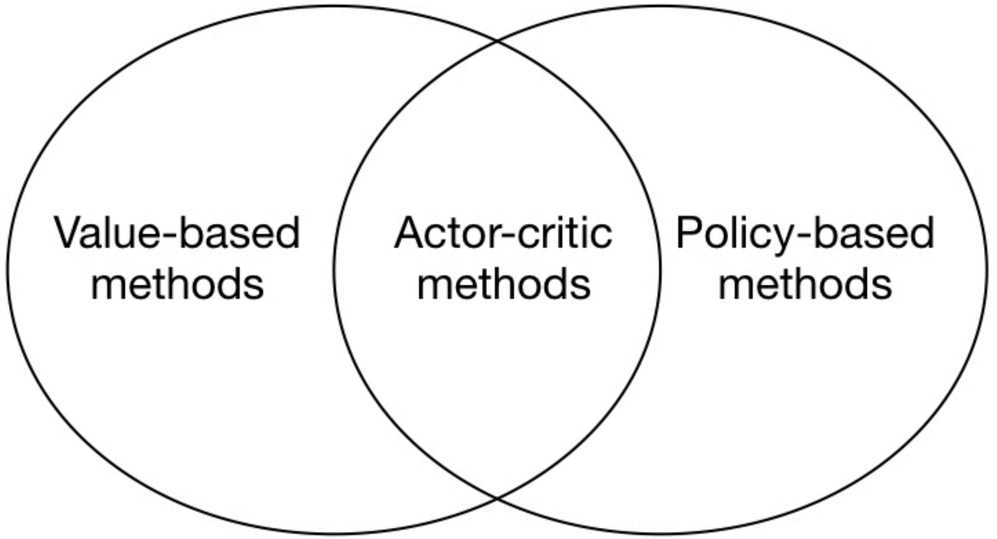


aims at improving current π

evaluates the current π

Relation to Other RL Methods

- Value-based methods
  - estimate the value function
  - policy is implicit (e.g., ϵ-greedy)
- Policy-based methods
  - estimate the policy
  - no value function
- Actor-critic methods
  - estimate the policy
  - estimate the value function



Implementing a Critic

- The critic estimates the value of the current policy - prediction problem
- Since the actor uses Q values to choose actions, the critic must estimate the Q function.
  - For **small state-spaces**, use tabular TD algorithms to estimate the Q function (SARSA, Q-learning, etc)
  - For **large state-spaces**, use LSTD to estimate the Q function.

Implementing an Actor

- Greedy improvement - move the policy toward the greedy policy underlying the Q function estimate obtained from the critic
  - For small state-action spaces: policy is greedy w.r.t. the obtained Q values
  - For large state-action spaces: policy is parameterized and the greedy action is computed on the fly
- Policy gradient - perform policy gradient directly on the performance surface $J(\omega)$ underlying the chosen parametric policy class

Variance Reduction in Policy gradient

- Instead of using $\nabla_\omega J(\omega) = \mathbb{E}_\pi[\gamma^t R_t \nabla_\omega \log \pi(a_t \mid s_t, \omega)]$ as in REINFORCE, we use $\nabla_\omega J(\omega) = \mathbb{E}_\pi[\gamma^t Q^\pi(s_t, a_t) \nabla_\omega \log \pi(a_t \mid s_t, \omega)]$ since we have estimated Q values in the Critic.
- Further we use an **advantage function** $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$ instead of $Q^\pi(s_t, a_t)$ since $\mathbb{E}_\pi[\gamma^t V^\pi(s_t) \nabla_\omega \log \pi(a_t \mid s_t, \omega)] = 0$

$$
\begin{aligned}
\mathbb{E}_{\tau\sim\pi_\theta}\Big[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)\Big] &= \mathbb{E}_{s_{0:t},a_{0:t-1}}\Big[\mathbb{E}_{s_{t+1:T},a_{t:T-1}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)\right]\Big] \\
&= \mathbb{E}_{s_{0:t},a_{0:t-1}}\Big[b(s_t) \cdot \underbrace{\mathbb{E}_{s_{t+1:T},a_{t:T-1}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\right]}_{E}\Big] \\
&= \mathbb{E}_{s_{0:t},a_{0:t-1}}\Big[b(s_t) \cdot \mathbb{E}_{a_t}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\right]\Big] \\
&= \mathbb{E}_{s_{0:t},a_{0:t-1}}\Big[b(s_t) \cdot 0\Big] = 0
\end{aligned}
$$

$$
\mathbb{E}_{a_t}\Big[\nabla_\theta \log \pi_\theta(a_t|s_t)\Big] = \int \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)} \pi_\theta(a_t|s_t) da_t = \nabla_\theta \int \pi_\theta(a_t|s_t) da_t = \nabla_\theta \cdot 1 = 0
$$

Outline

1. Introduction
2. Bellman Equations
3. Temporal Difference (TD) Methods
4. Function Approximation for Value Functions
5. Actor-critic Methods
6. Deep Reinforcement Learning

Deep Reinforcement Learning

- Deep reinforcement learning refers to using a neural network to approximate the value function, the policy or the model.

- Nonlinear function approximate might be “rich”
- However, it may not give any interpretation or the estimate might be stuck at the local optima due to the non-convexity of the optimization landscape

Value-Based Algorithms

Deep Neural Networks

Neural network transforms input vector $\mathbf{x}$ into an output $\mathbf{y}$:

$$\begin{aligned}\mathbf{h}_0 &= g_0(W_0\mathbf{x}^\mathsf{T} + b_0)\\ \mathbf{h}_i &= g_i(W_i\mathbf{h}_{i-1}^\mathsf{T} + b_i), 0 < i < m\\ \mathbf{y} &= g_m(W_m\mathbf{h}_{m-1}^\mathsf{T} + b_m)\end{aligned}$$

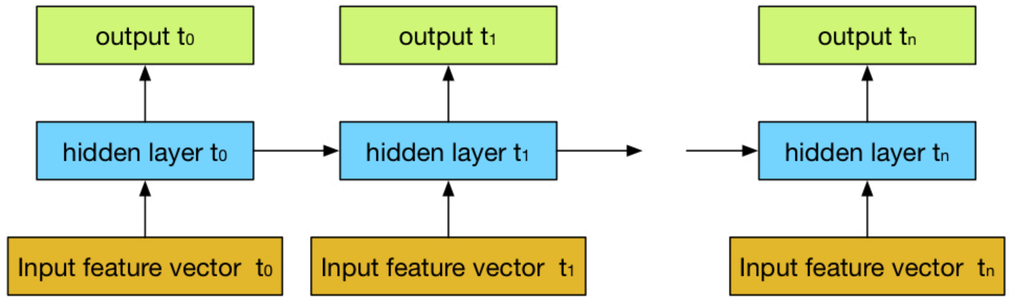
where

- g_i (differentiable) activation functions hyperbolic tangent tanh or sigmoid σ, $0 \leq i \leq m$
- W_i, b_i parameters to be estimated, $0 \leq i \leq m$

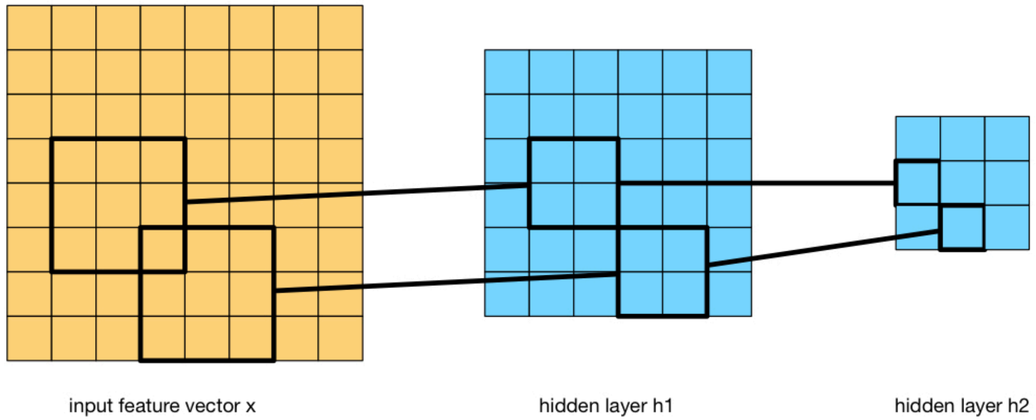
It is trained to minimise the loss function $L = |\mathbf{y}^* - \mathbf{y}|^2$ with stochastic gradient descent in the regression case. In the classification case, it minimises the cross entropy $-\sum_i y_i^* \log y_i$.

Weight Sharing in Deep Neural Networks

- Recurrent neural network shares weights between time-steps

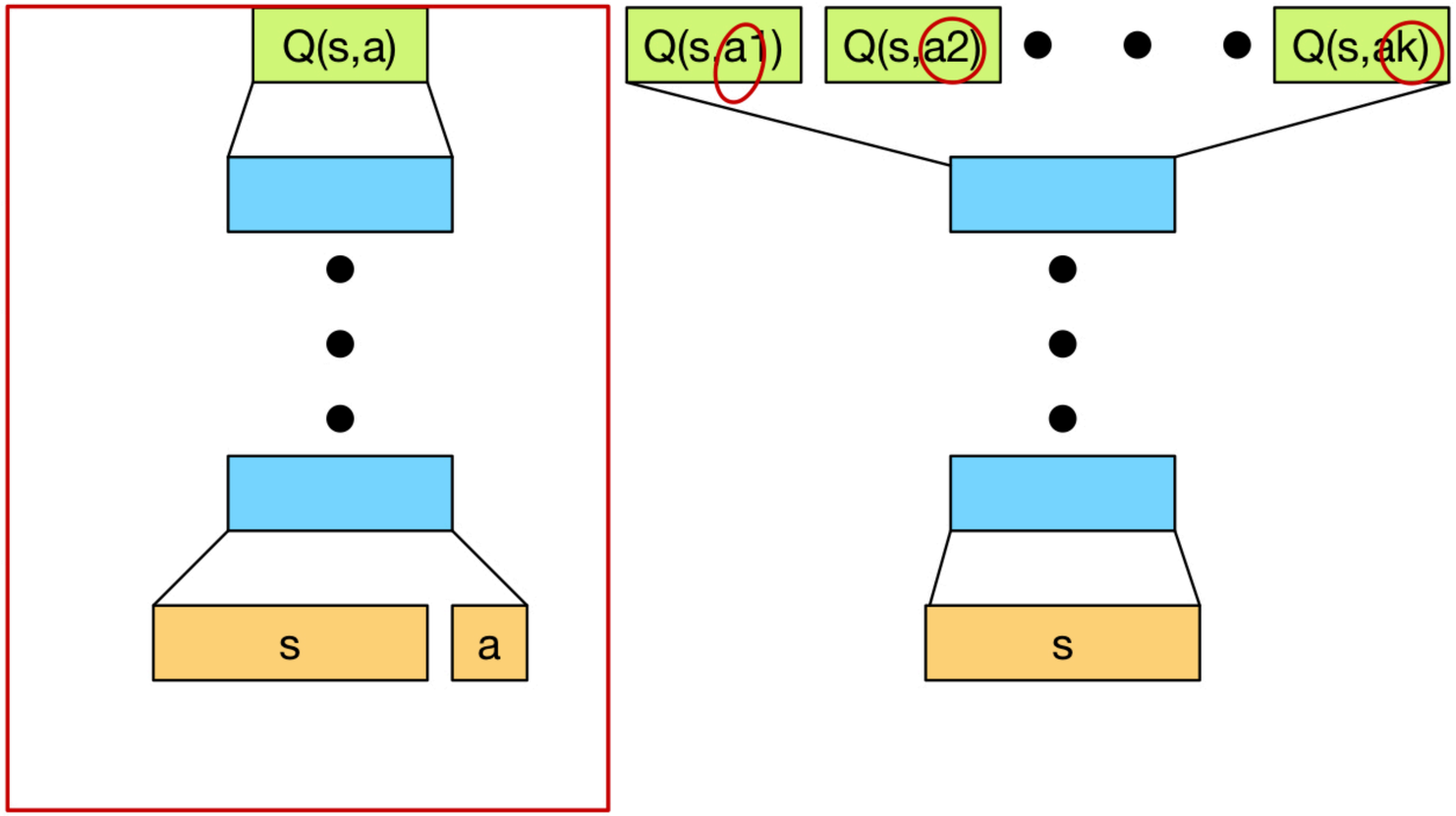


- Convolutional neural network shares weights between local regions

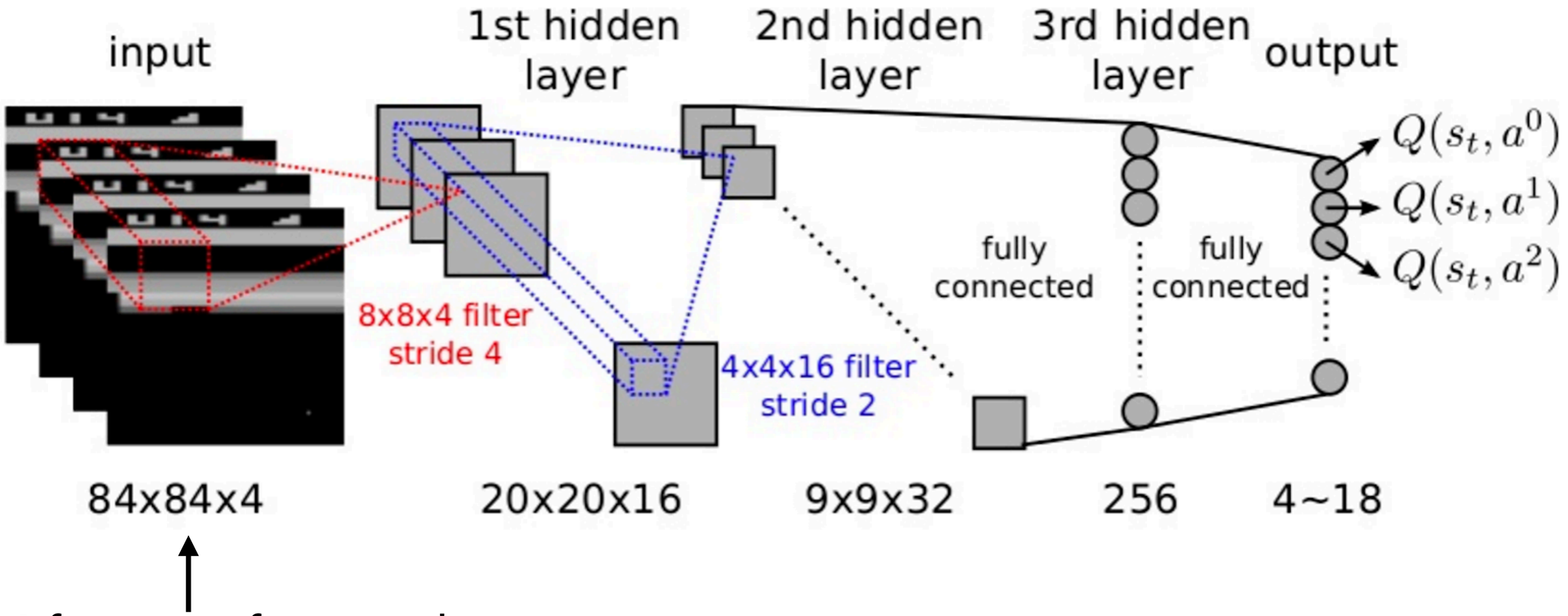


Q-networks

- Q-networks approximate the Q-function as a neural network
- There are two architectures:
  1. Q-network takes an input s, a and produces $Q(s, a)$
  2. Q-network takes an input s and produces a vector $Q(s, a_1), \cdots, Q(s, a_k)$



Deep Q-Learning (DQN)



last 4 frames of gameplay

Deep Q-networks

$Q(s, a, \theta)$ is a neural network.

$$MSVE = \left(r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta) \right)^2$$

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$$

i: # iteration

$$\text{where } y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

target

Gradient:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right]$$

Deep Q-networks

$Q(s, a, \theta)$ is a neural network.

$$MSVE = \left(r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta) \right)^2$$

- Q-learning algorithm where Q-function estimate is a neural network
- This algorithm provides a biased estimate

This algorithm diverges because

- States are correlated
- Targets are non-stationary

Deep Q-Learning (DQN)
Key mechanisms

- Experience replay: store (s,a,r,s') in a replay buffer, and randomly sample instances from the buffer to remove correlations in the data
  - In order to deal with the correlated states, the agent builds a dataset of experience and then makes random samples from the dataset.
  - In order to deal with non-stationary targets, the agent fixes the parameters $\boldsymbol{\theta}^-$ and then with some frequency updates them

$$MSVE = \left(r + \gamma \max_{a'} Q(s', a', \boldsymbol{\theta}^-) - Q(s, a, \boldsymbol{\theta}) \right)^2$$

Deep Q-Learning (DQN)
Key mechanisms

- Experience replay: store (s,a,r,s’) in a replay buffer, and randomly sample instances from the buffer to remove correlations in the data
- Target network: the parameters of the target network (network to compute Q-values) are updated periodically and held fixed in between, to reduce the correlations between action values Q and the target $r + \gamma \max_{a'} Q(s', a')$

Extensions of DQN

Prioritized Replay

[Schaul et al., 2015]

- Instead of randomly selecting experience order the experience by some measure of priority
- The priority is typically proportional to the TD-error

$$\delta = |r + \gamma \max_{a'} Q(s', a', \boldsymbol{\theta}^-) - Q(s, a, \boldsymbol{\theta})|$$

Double DQN

[van Hasselt et al., 2015]

- Remove upward bias caused by $\max_{a'} Q(s', a', \boldsymbol{\theta}^-)$
- The idea is to produce two Q-networks
  1. Current Q-network $\boldsymbol{\theta}$ is used to select actions
  2. Older Q-network $\boldsymbol{\theta}^-$ is used to evaluate actions

$$MSVE = \left(r + \gamma Q(s', \arg\max_{a'} Q(s', a', \boldsymbol{\theta}), \boldsymbol{\theta}^-) - Q(s, a, \boldsymbol{\theta}) \right)^2$$

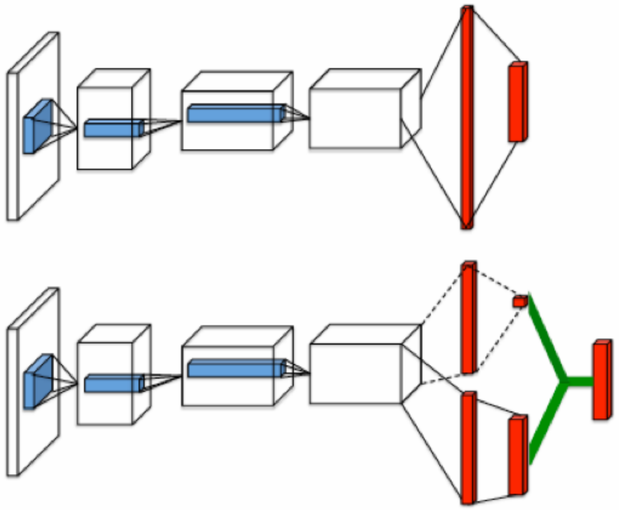
Dueling Q-network

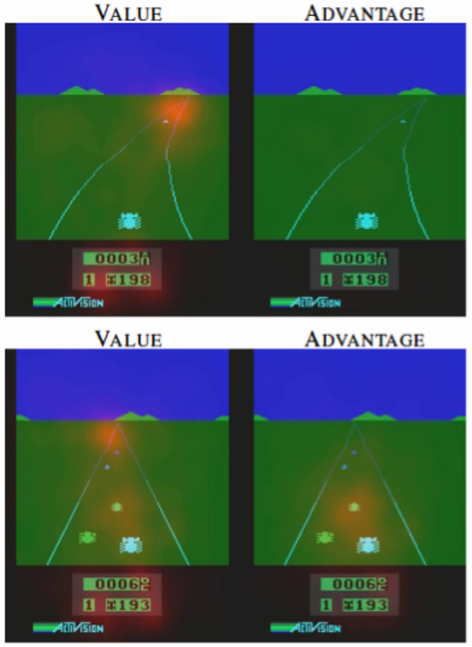
[Wang et al., 2015]

- Dueling Q-network combined two streams to produce Q-function:
  1. one for state values
  2. another for advantage function
- The network learns state values for which actions have no effect
- Dueling architecture can more quickly identify correct action in the case of redundancy

Dueling Q-network

[Wang et al., 2015]





- Traditional DQN and dueling DQN architecture

- The value stream learns to pay attention to the road.
- The advantage stream learns to pay attention only when there are cars immediately in front

DRL

Furong Huang

Outline

1. Introduction
2. Bellman Equations
3. Temporal Difference (TD) Methods
4. Function Approximation for Value Functions
5. Actor-critic Methods
6. Deep Reinforcement Learning

Policy Gradient Algorithms

Policy Approximation

- Policy π is a neural network parametrised with $\boldsymbol{\omega} \in \mathbb{R}^n$, $\pi(a, s, \boldsymbol{\omega})$
- Performance measure $J(\boldsymbol{\omega})$ is the value of the initial state $V_{\pi(\boldsymbol{\omega})}(s_0) = E_{\pi(\boldsymbol{\omega})}[r_0 + \gamma r_1 + \gamma^2 r_2, + \cdots]$
- The update of the parameters is

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha \nabla J(\boldsymbol{\omega}_t)$$

- And the gradient is given by the policy gradient theorem

$$\nabla J(\boldsymbol{\omega}) = E_\pi \left[\gamma^t R_t \nabla_{\boldsymbol{\omega}} \log \pi(a|s_t, \boldsymbol{\omega}) \right]$$

- This gives REINFORCE algorithm for a neural network policy

Advantage Actor-Critic (A2C)

Why does the policy gradient method have high variance?

For the sake of hypothetical example, lets say that $\nabla_\theta log \pi_\theta(\tau)$ is $[0.5, 0.2, 0.3]$ respectively for three trajectories, and $r(\tau)$ is $[1000, 1001, 1002]$.

Then the variance of the product of the two terms for these three samples is $Var(0.5 \times 1000, 0.2 \times 1001, 0.3 \times 1002)$ which according to WolframAlpha is around 23286.8.

Now what if we reduce all values of $r(\tau)$ by a constant, say, 1001? Then the variance of the product becomes $Var(0.5 \times 1, 0.2 \times 0, 0.3 \times 1)$, which is 0.1633, a much smaller value.

Advantage Actor-Critic (A2C)

Advantage function:

$$A(s_t, a_t) = Q_w(s_t, a_t) - V_v(s_t) = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)$$

(how better an action is compared to the others at a given state)

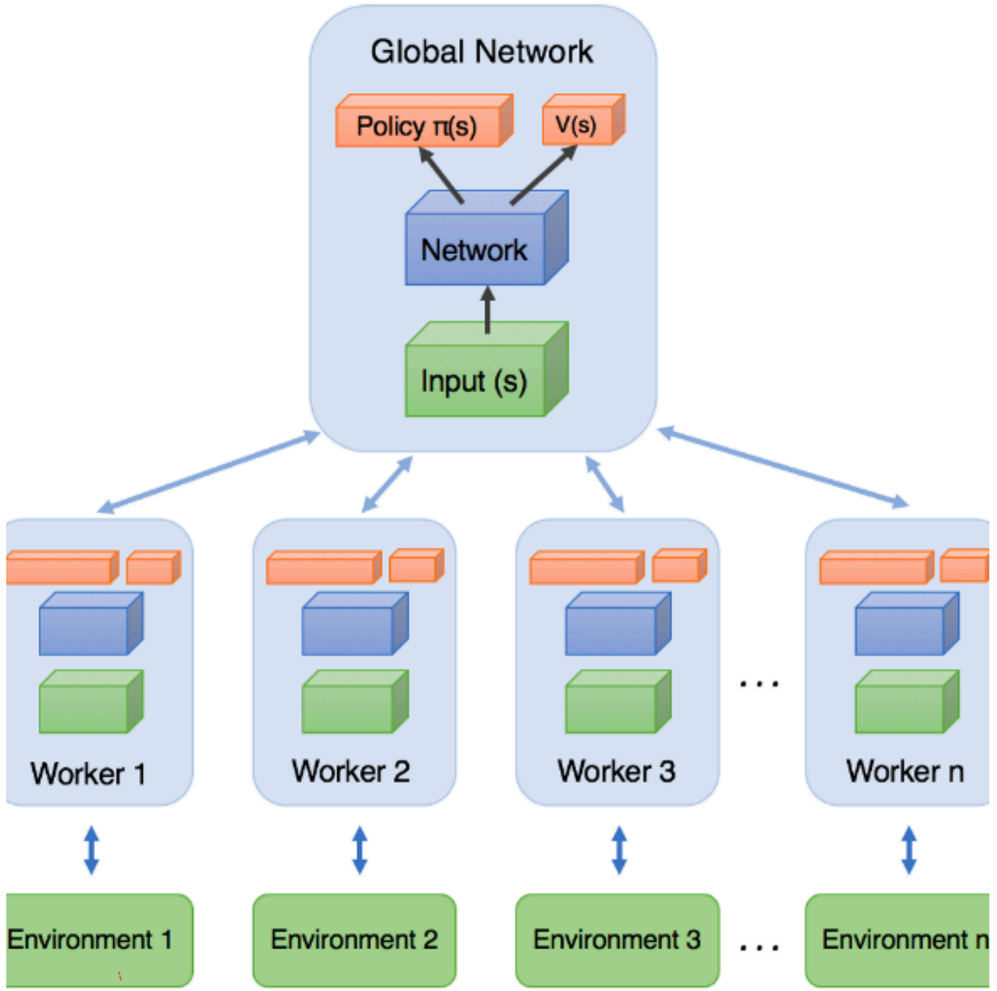
A2C: let the critic learn A(s,a) instead of Q(s,a)

$$\begin{aligned}\nabla_\theta J(\theta) &\sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)) \\ &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)\end{aligned}$$

Results: reduces the high variance of policy networks and stabilize the model.

Asynchronous Advantage Actor-Critic (A3C)
(Mnih et al., 2016)

- A3C consists of multiple independent agents(networks) with their own weights, who interact with a different copy of the environment in parallel.
- Results: they can explore a bigger part of the state-action space in much less time.



Deep Deterministic Policy Gradient (DDPG)
(Lillicrap et al. 2016)

- DDPG is an off-policy algorithm.
- DDPG can only be used for environments with continuous action spaces.
- DDPG can be thought of as being deep Q-learning for continuous action spaces.

DDPG ≈ Actor-Critic + DQN

Deep Deterministic Policy Gradient (DDPG)
(Lillicrap et al. 2016)

4 networks:

θ^Q : Q network

θ^μ : Deterministic policy function → directly maps a state to an action

$\theta^{Q'}$: target Q network

$\theta^{\mu'}$: target policy network → time-delayed copies

Deep Deterministic Policy Gradient (DDPG)
(Lillicrap et al. 2016)

- The Q-learning side:

  minimize the MSBE loss

  slowly updated target networks

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}} \left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s')) \right) \right)^2 \right]$$

  d: is or not terminal state?

- The policy-learning side:

  maximize the action values (parameters of Q are treated as constants)

$$\max_\theta \underset{s\sim\mathcal{D}}{\mathrm{E}} \left[Q_\phi(s, \mu_\theta(s)) \right]$$

Trust Region Policy Optimization (TRPO)

(Schulman et al. 2015)

monotonically improve policies with theoretical guarantee!

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right]$$

$$\text{subject to} \quad \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta.$$

have shown: policy iteration, policy gradient, and natural policy gradient (Kakade, 2002) are special cases of TRPO

Trust Region Policy Optimization (TRPO)

(Schulman et al. 2015)

Preliminaries

MDP: $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$

Let π denote a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, and let $\eta(\pi)$ denote its expected discounted reward:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right], \text{ where}$$

$$s_0 \sim \rho_0(s_0),\ a_t \sim \pi(a_t|s_t),\ s_{t+1} \sim P(s_{t+1}|s_t, a_t).$$

Trust Region Policy Optimization (TRPO)

(Schulman et al. 2015)

Preliminaries

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right],$$

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right],$$

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s), \text{ where}$$

$$a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t) \text{ for } t \geq 0.$$

Trust Region Policy Optimization (TRPO)

(Schulman et al. 2015)

Preliminaries

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \cdots \sim \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right]$$

$$= \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a).$$

visit frequencies

as long as >=0 for every state, \tilde{\pi} is guaranteed to increase \eta

problem: in approximate setting, it is typically unavoidable that for some state, this term is negative.

Trust Region Policy Optimization (TRPO)

(Schulman et al. 2015)

Local approximation to \eta

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a)$$

If we have a diffientiable function of \pi parameterized by \theta

$$L_{\pi_{\theta_0}}(\pi_{\theta_0}) = \eta(\pi_{\theta_0}),$$
$$\nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta)\big|_{\theta=\theta_0} = \nabla_\theta \eta(\pi_\theta)\big|_{\theta=\theta_0}$$

a sufficiently small step $\pi_{\theta_0} \rightarrow \tilde{\pi}$ that improves $L_{\pi_{\theta_{\text{old}}}}$ will also improve η

Trust Region Policy Optimization (TRPO)

(Schulman et al. 2015)

Main result 1:

$$\eta(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - CD_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}),$$

$$\text{where } C = \frac{4\epsilon\gamma}{(1-\gamma)^2}.$$

$$D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{\mathrm{KL}}(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s))$$

Algorithm 1 Policy iteration algorithm guaranteeing non-decreasing expected return η

Initialize π_0.

for $i = 0, 1, 2, \ldots$ until convergence **do**

Compute all advantage values $A_{\pi_i}(s, a)$.

Solve the constrained optimization problem

$$\pi_{i+1} = \arg\max_{\pi} \left[L_{\pi_i}(\pi) - CD_{\mathrm{KL}}^{\max}(\pi_i, \pi) \right]$$

$$\text{where } C = 4\epsilon\gamma/(1-\gamma)^2$$

$$\text{and } L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$$

end for

Trust Region Policy Optimization (TRPO)

(Schulman et al. 2015)

Monotonic Improvement Guarantee

$$\text{let } \bar{M}_i(\pi) = \quad L_{\pi_i}(\pi) - C D_{\mathrm{KL}}^{\max}(\pi_i, \pi).$$

$$\eta(\pi_i) = M_i(\pi_i)$$

$$\eta(\pi_{i+1}) \geq M_i(\pi_{i+1})$$

$$\eta(\pi_{i+1}) - \eta(\pi_i) \geq M_i(\pi_{i+1}) - M(\pi_i)$$

Minorize-Maximization (MM) algorithm

Trust Region Policy Optimization (TRPO)

(Schulman et al. 2015)

Main result 2:

- For policy parameterized by \theta

$$\underset{\theta}{\text{maximize}}\,[L_{\theta_{\text{old}}}(\theta) - CD_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta)]$$

- Practical solution: use a trust region constraint

$$\underset{\theta}{\text{maximize}}\, L_{\theta_{\text{old}}}(\theta)$$
$$\text{subject to } D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta) \leq \delta.$$

$$\underset{\theta}{\text{maximize}}\, L_{\theta_{\text{old}}}(\theta)$$
$$\text{subject to } \overline{D}_{\text{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leq \delta.$$

where $\overline{D}_{\text{KL}}^{\rho}(\theta_1, \theta_2) := \mathbb{E}_{s\sim\rho}\left[D_{\text{KL}}(\pi_{\theta_1}(\cdot|s) \,\|\, \pi_{\theta_2}(\cdot|s))\right]$

average KL divergence

Proximal Policy Optimization (PPO)
(Schulman et al. 2017)

TRPO: good theoretical performance, but complicated implementation and computation, because the surrogate objective function is the KL divergence between the old and the new policy.

PPO: propose a clipped surrogate function

The idea of TRPO's constraint is disallowing the policy to change too much. Therefore, instead of adding a constraint, PPO slightly modifies TRPO's objective function with a penalty for having a too large policy update.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$

- TRPO, PPO, A3C: on-policy, sample inefficient
- DQN, DDPG: off-policy, sensitive to hyperparameters

Model-based Methods

Why Model-based RL?

Why can human make good decisions?

the ability to “predict”

- Advantages of using model:
  - Sample efficiency
  - Model reuse and transfer
  - Find better representations
  - …

The Basic Approach

- What is a “model”?

  a dynamics model “predicts” the next state: $f(s_t, a_t) = s_{t+1}$

  (A similar predict model can also be built for reward.
  For simplicity, assume the reward function is known)

- Use the model

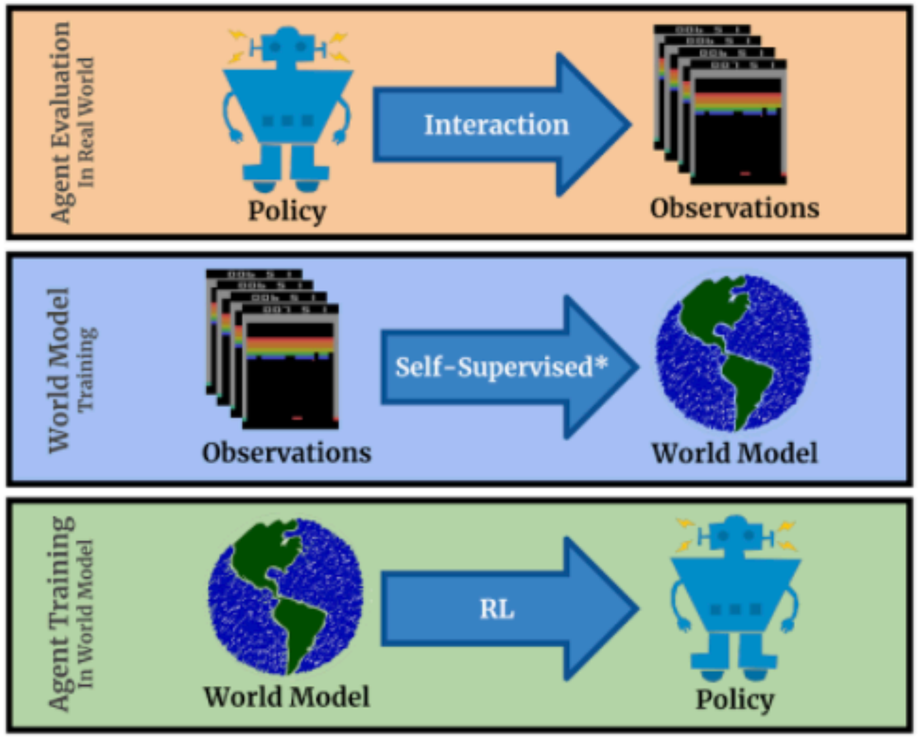
  Model predictive control (MPC):

$$(\mathbf{a}_t, \dots, \mathbf{a}_{t+H-1}) = \arg \max_{\mathbf{a}_t, \dots, \mathbf{a}_{t+H-1}} \sum_{t'=t}^{t+H-1} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$$

note: in practice, we compute a sequence of actions, but execute only a_t, and then replan at the next steps with updated state information

Save Samples with Model Learning

The "SimPLe" framework [Kaiser, et al. 2019]



require less samples from the environment

collect trajectories from the real environment

update the model with collected trajectoreis

improve the policy in the learned model

Kaiser L, Babaeizadeh M, Milos P, et al. Model-based reinforcement learning for atari[J]. arXiv preprint arXiv:1903.00374, 2019.

A Brief Summary

Summary

- Neural networks can be used to approximate the value function, the policy or the model in reinforcement learning.
- Any algorithms that assumes a parametric approximation can be applied with neural networks
- However, vanilla versions might not always converge due to biased estimates and correlated samples
- With methods such as prioritised replay, double Q-network or duelling networks the stability can be achieved
- Neural networks can also be applied to actor-critic methods
- Using them for model-based method does not always work well due to compounding errors

Deep RL Methods

- Model-free
  - Value-based: DQN, Double DQN, Rainbow DQN, PER, Retrace, ...
  - Policy-based (usually actor-critic): A2C, A3C, DDPG, TRPO, PPO, ...
- Model-based
  - use dynamics model to simulate
  - use model to initialize model-free learners
  - use model to regularize learned representation

On-policy and Off-policy

On-policy methods:

REINFORCE, A2C, A3C, TRPO, PPO, ...

FRAMEWORK

Initialize policy/value parameters, on-policy buffer D

for iteration = 1, 2, 3, ...
  collect set of trajectories, save to D
  update policy/value parameters with D
  clear D
endfor

Off-policy methods:

DQN, DDPG, SAC, Model-based, ...

FRAMEWORK

Initialize policy/value parameters, off-policy buffer D

for step = 1, 2, 3, ...
  take action, save the current transition to D
  draw a mini-batch from D
  update policy/value parameters with the mini-batch
endfor