Introduction to Deep Reinforcement Learning

https://github.com/racousin/rl_introduction

Raphael Cousin

January 11, 2024

Reinforcement Learning Objective

RL aims to optimize decision-making in environments without a known transition model P.

Objective

Find the optimal policy π^*, that maximizes the expected return, $J(\pi)$:

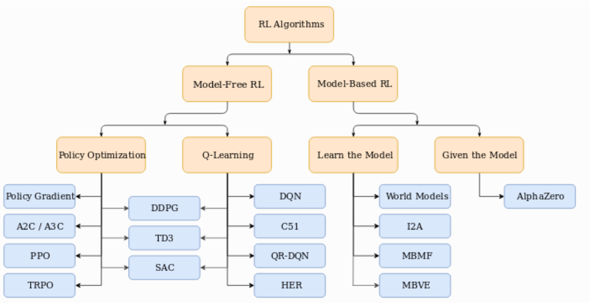
$$\pi^* = \arg\max_{\pi} J(\pi)$$

$$J(\pi) = \mathbb{E}_{\tau \sim \pi}[G(\tau)] = \int_{\tau} \mathbb{P}(\tau|\pi) G(\tau)$$

First Glossary of RL

- Model free/ Model based
- Q-learning/Policy Optimization
- On-policy/Off-policy
- ϵ-Greedy

Overview of RL Algorithms



- **Model free:** learn the policy π^* directly
- **Model based:** use an environment model P^* to learn π^*

Key Strategies in Model-Free RL

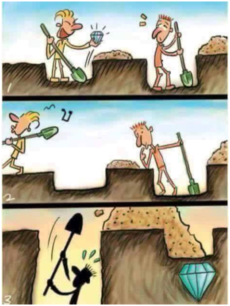
- **Q-learning:** Learn the action-value function Q to determine the best action given a state:

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q_\pi(s, a)$$

- **Policy Optimization:** Directly learn the policy π that maximizes the expected return.

Exploration-Exploitation

Knowledge of the environment comes from interaction. There are trade-offs to be made between using what we know and further exploration.



The ϵ-Greedy Strategy

The ϵ-greedy strategy is a simple yet effective method for balancing exploration and exploitation by choosing:

Given a state s, at step t the policy π is defined as:

- With probability ϵ, choose an action at random (exploration).
- With probability $1 - \epsilon$, choose the action with the highest estimated value (exploitation).

Definition

- **On-policy:** Directly learns from and improves the policy it executes.
- **Off-policy:** Learns a different policy from the executed one, allowing for learning from observations.

Second Glossary of RL

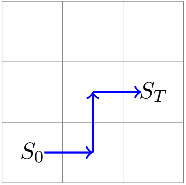
- Monte-Carlo
- Temporal difference
- SARSA
- Q-learning

- To evaluate $V_\pi(s) = E_{\tau\sim\pi}[G_t \,| s_t = s]$
- Generate an episode with the policy π $S_1, A_1, R_2, \ldots, S_T$ to compute $G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$.
- The empirical value function is : $V_\pi(s) = \frac{\sum_{t=1}^{T} \mathbb{1}[S_t=s] G_t}{\sum_{t=1}^{T} \mathbb{1}[S_t=s]}$
- As, well, the empirical action-value function is :
- $Q_\pi(s,a) = \frac{\sum_{t=1}^{T} \mathbb{1}[S_t=s, A_t=a] G_t}{\sum_{t=1}^{T} \mathbb{1}[S_t=s, A_t=a]}$

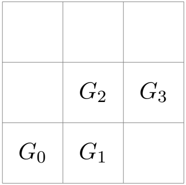
Initialize Q $Q(s,a) \forall s, a$.

1. Generate an episode with the policy π (extract from Q ϵ-greedy)
2. Update Q using the episode:
   $q_\pi(s,a) = \frac{\sum_{t=1}^{T} \left(\mathbb{1}[S_t = s, A_t = a] \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \right)}{\sum_{t=1}^{T} \mathbb{1}[S_t = s, A_t = a]}$
3. Iterate

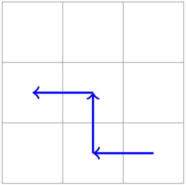
Visual Steps in Monte Carlo



1. Generate episode following $\arg\max Q(s, a)$



2. Evaluate Q



3. Iterate

Temporal Difference (TD)

Monte Carlo and dynamic programming ideas, using bootstrapping for value updates.

- **Bellman equations**:

$$V(S_t) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]$$

$$Q(s, a) = \mathbb{E}[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

- **TD Target**: unbiased estimate:
  - $V(S_t)$: $R_{t+1} + \gamma V(S_{t+1})$
  - $Q(S_t, A_t)$: $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

Value function estimation with TD Learning

TD Error (δ_t)

The difference between the TD target and the current value estimate:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

Update Rule

Incrementally update the state value with the learning rate (α):

$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t$$

Action-Value function estimation with TD Learning

TD Error (δ_t)

The difference between the TD target and the current value estimate:

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

Update Rule

Incrementally update the state value with the learning rate (α):

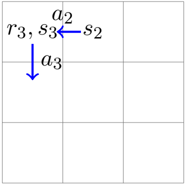
$$Q(S_t) \leftarrow Q(S_t) + \alpha\delta_t$$

SARSA Algorithm

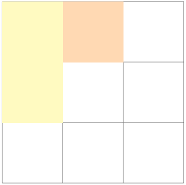
Initialize Q function $Q(s,a) \forall s, a$
$S_t =$ initial state, act with π (extract from Q ϵ-greedy) to get A_t, R_{t+1}, S_{t+1}

1. Act with π (extract from Q ϵ-greedy) to get $A_{t+1}, R_{t+2}, S_{t+2}$
2. Update Q using the observation step:
   $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
3. Iterate

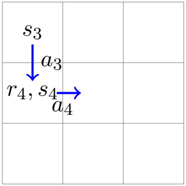
Visual Steps in SARSA



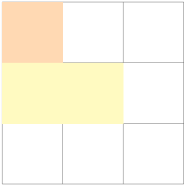
1. Choose action $a_3 = \arg\max Q(s_3, a)$



2. Update $Q(s_2, a_2)$ with $r_3 + \gamma Q(s_3, a_3)$



3. Choose action $a_4 = \arg\max Q(s_4, a)$



4. Update $Q(s_3, a_3)$ with $r_4 + \gamma Q(s_4, a_4)$

Q-learning

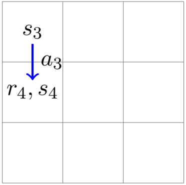
- Remember $Q_{\pi^*}(S_t, A_t) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a')|S_t = s, A_t = a]$
- So $R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')$ is an unbiased estimate for $Q(S_t, A_t)$
- $R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')$ is called the Q target.
- α improvement: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$

Q-learning Algorithm

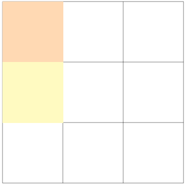
Initialize Q function $Q(s, a) \forall s, a$
$S_t =$ initial state

1. Act with π (extract from Q ϵ-greedy) to get A_t, R_{t+1}, S_{t+1}
2. Update Q using the observation step:
   $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max Q(S_{t+1}, a) - Q(S_t, A_t))$
3. Iterate

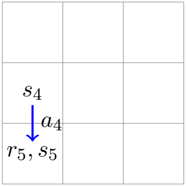
Visual Steps in Q-Learning



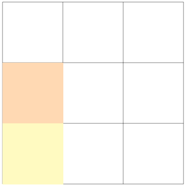
1. Choose action $a_3 = \arg\max Q(s_3, a)$



2. Update $Q(s_3, a_3)$ with $r_4 + \gamma \max Q(s_4, a)$



3. Choose action $a_4 = \arg\max Q(s_4, a)$



4. Update $Q(s_4, a_4)$ with $r_5 + \gamma \max Q(s_4, a)$

Coding session
Temporal Difference

Third Glossary of RL

- Reinforce/VPG
- Deep Q-learning
- Actor-Critic

Limitations of Traditional Q Learning

Q Learning faces challenges when scaling to complex problems:

- High-dimensional state spaces lead to slow convergence.
- Inapplicable to environments with continuous action spaces.

Deep Q Learning Overview

Deep Q Learning extends Q Learning by using neural networks:

- Parametrize Q function with θ, $Q_\theta : S \times A \to \mathbb{R}$.
- Objective: Find θ^* that approximates the optimal Q function.
- Define Q target as: $y = R_{t+1} + \gamma \max_{a'} Q_\theta(S_{t+1}, a')$.
- Minimize loss (e.g., MSE): $L(\theta) = \mathbb{E}_{s,a\sim Q}[(y - Q(s,a,\theta))^2]$.

Executing the Deep Q Learning Algorithm

Steps to implement Deep Q Learning:

1. For current state S_t, compute $Q_\theta(S_t, a)$ for all actions.
2. Take action A_t with highest Q value, observe reward and next state.
3. Compute target y for S_{t+1} and minimize loss $L(\theta)$.
4. Iterate to refine θ towards optimal.

Improving Deep Q Learning Stability

Key techniques for enhancing DQL:

- **Experience Replay**: Store transitions $(S_t, A_t, R_{t+1}, S_{t+1})$ and sample randomly to break correlation in sequences.
- **Target Network**: Use a separate, slowly updated network to stabilize targets.
- **Additional Improvements**: Epsilon decay for exploration, reward clipping, Double Q Learning to reduce overestimation.

Coding session
Deep Q - learning

Policy Optimization

- Parametrization of policy, π_θ.
- We aim to maximize the expected return $J(\pi_\theta) = E_{\tau \sim \pi_\theta}[G(\tau)]$.
- Gradient ascent:
  $\theta_{k+1} = \theta_k + \alpha \left. \nabla_\theta J(\pi_\theta) \right|_{\theta_k}$.
- We can proof that:
  $\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)]$
  `https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html`

Reinforce/VPG algorithm

Initialize policy π_θ

1. Generate episodes $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$ with the policy π_θ
2. Compute gradient approximation
   $\hat{\nabla} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$
3. Update policy (apply gradient ascent) $\theta \leftarrow \theta + \alpha \hat{\nabla}$
4. Iterate

Introduction to Actor-Critic Models

Actor-Critic models combine the benefits of policy-based and value-based approaches:

- The **Actor** updates the policy distribution in the direction suggested by the **Critic**.
- The **Critic** estimates the value function (V or Q) to critique the actions taken by the Actor.
- This interaction enhances learning by using the Critic's value function to reduce the variance in policy gradient estimates.

Policy Gradient in Actor-Critic

The policy gradient in Actor-Critic models can be written as:

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t\right]$$

Where Φ_t represents:

- Total return G_t.
- Advantage function: $R_{t+1} - V(s_t)$ or $R_{t+1} - Q(s_t, a_t)$.

Using Φ_t improves policy updates by evaluating actions more effectively.

Actor-Critic Algorithm Steps

Implementing the Actor-Critic algorithm involves:

1. Initializing parameters for both the Actor (θ) and the Critic (ϕ).
2. For each episode:
   1. Generate an action A_t using the current policy π_{θ_t}.
   2. Update the Actor by applying gradient ascent using the Critic's feedback.
   3. Update the Critic by minimizing the difference between estimated and actual returns.
3. Repeat the process to refine both Actor and Critic.

References

Course:

http://incompleteideas.net/book/the-book-2nd.html

https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html

http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/

https://github.com/kengz/awesome-deep-rl

Framework:

https://spinningup.openai.com/en/latest/

https://gym.openai.com/envs/#atari

https://github.com/deepmind/bsuite