Reinforcement Learning introduction for AI safety

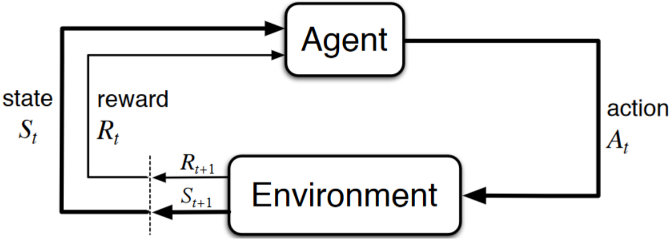
Aude Maier

Swiss AI Safety Camp 2023

Program of the day

- Introduction to Markov Decision Processes
- TD-learning
  - Tabular Q-learning
  - Deep Q-learning
  - Coding exercise on deep Q-learning
- Policy gradient
  - Reinforce algorithm
  - Coding exercise on Reinforce
  - Actor Critic architecture
  - Proximal Policy Optimization

Markov Decision Process (MDP)



- S_t, A_t, R_t are Random Variables;
  $S_t \in \mathcal{S}$, $A_t \in \mathcal{A}$, $R_t \in \mathbb{R}$.
- The interaction between Agent and Environment creates the **Trajectory**: $S_0, A_0, R_0, S_1, A_1, R_1, \ldots$
- R_t, S_t only depend on S_{t-1}, A_{t-1} (Markov Property), they are distributed according to the **Dynamic** of the MDP:

$$p(s', r|s, a) := Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

Markov Decision Process (MDP)

Definitions

- **Return**:

$$G_t := R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ...$$

  What we want to maximize (in expected value).

- **Policy**:

$$\pi(a|s) := Pr\{A_t = a|S_t = s\}$$

  Rule according to which the agent selects an action. "Making the agent learn" means modifying its policy in order to maximize the expected return.

- **V-values**:

$$V_\pi(s) := \mathbb{E}_\pi[G_t|S_t = s]$$

- **Q-values**:

$$Q_\pi(s, a) := \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

  Property - $V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s, a)$

Markov Decision Process (MDP)

Commonly used policies

- Greedy: $\pi(s, a) = \mathbb{1}\{a = \mathrm{argmax}_a Q_(s, a)\}$ (deterministic)
- Epsilon-greedy: follow the greedy policy with probability $0 < \varepsilon < 1$ and choose a random action with probability ε (stochastic)
- Softmax: $\pi(s, a) = \frac{\exp(\beta Q(s,a)}{\sum_{\tilde{a}} \exp(\beta Q(s,\tilde{a})}$ (stochastic)

Markov Decision Process (MDP)

- **Optimal policy**: a policy π_* is optimal if $V_{\pi_*}(s) \geq V_\pi(s)$ for any state s and policy π

Theorem

There always exists at least one optimal policy.

- **Optimal V/Q-values**: $v_*(s) := v_{\pi_*}(s) = max_\pi V_\pi(s)$, $q_*(s, a) := q_{\pi_*}(s, a) = max_\pi q_\pi(s, a)$

Exercises - Build the Bellman Equation

1. Write G_t as a function of G_{t+1}.
2. Write a recursive equation for the V-values $v_\pi(s)$. Hint: plug your result to the previous exercise in the definition of the V-values.
3. Do the same for the Q-values.

Model free / Model based RL

- **Model based**: The agent has an explicit representation of $p(s', r|s, a)$, allows explicit planning
- **Model free**: No explicit representation of $p(s', r|s, a)$, learns heuristics (e.g. taking action a in state s usually leads to high reward later in the episode), learns directly v/q-values or policy

Model based Reinforcement Learning (optional

- Given the model:
  Optimization problem - can be solved optimally by dynamic programming methods, not always computationally tractable therefore we use decision time planning methods such as monte carlo tree search. This is the case of AlphaZero
- Learns the model:
  Approximate the dynamic and learn the policy / v-values using the learned dynamic. For a complex game such as go, use a latent space to represent states. This is the case of MuZero

Model free Reinforcement Learning

- **Temporal Difference (TD)-learning**
  First learn the optimal Q-values using the Bellman equation. Then find the optimal policy by taking $\pi_*(a|s) = \mathbb{1}\{a = \mathrm{argmax}_a Q_*(s, a)\}$.
- Policy learning
  Learn directly the policy $\pi(a|s)$ such as to maximize the expected return.

TD-learning

Bellman Optimality Equation

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}[R_t + \gamma \mathrm{max}_{a'} Q_*(S_{t+1}, a') | S_t = s, A_t = a] \\
&= \sum_{r,s'} p(r, s'|s,a) \left(r + \gamma \mathrm{max}_{a'} Q_*(s', a')\right)
\end{aligned}
$$

Theorem

1. There always exists a unique set of Q-values $\{Q(s,a)\}_{s \in \mathcal{S}, a \in \mathcal{A}}$ that satisfy the Bellman Optimality Equation;
2. The Q-values that satisfy the Bellman Optimality Equation are the optimal Q-values.

In other words - A set of Q-values $\{Q(s,a)\}_{s \in \mathcal{S}, a \in \mathcal{A}}$ satisfy the Bellman Optimality Equation if and only if they are the optimal Q-values.

TD-learning

Idea

Update the Q-values iteratively to minimze the difference between the RHS and LHS of the Bellman equation. If the iteration converges, it means that we have found the optimal Q-values.

Update Rule

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_t + \gamma \mathrm{max}_a Q(S_{t+1}, a) - Q(S_t, A_t))$$

TD-learning

Algorithm

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop 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

Exercise tabular Q-learning

Consider a rat navigating in a 1-armed maze (=linear track). The rat is initially placed at the upper end of the maze (state s), with a food reward at the other end. This can be modeled as a one-dimensional sequence of states with a unique reward ($r = 1$) as the goal is reached. For each state, the possible actions are going up or going down. When the goal is reached, the trial is over, and the rat is picked up by the experimentalist and placed back in the initial position s and the exploration starts again.

- Describe what happens during the first episode.
- Compute the Q-values at the end of the first episode.
- Compute the Q-values at the end of the second episode.
- Give the Q-values that will reached after an arbitrarily large number of episodes.

Speed up the propagation of the values - Optional

- N-step TD-learning - Write the Q-values $Q(S_t, A_t)$ as a function of $Q(S_{t+n}, A_{t+n})$.
- Eligibility traces - associate a variable to every Q-value, called its trace, which represents the contribution of the corresponding state-action pair to the reward obtained.

TD-learning with function approximation: deep Q-learning

Problems of tabular Q-learning

- Many entries
- Cannot handle continuous spaces
- Does not take into account the proximity between states

Idea

Define a neural network that takes a continuous state as input and returns the Q-value of every action.
$Q(s, a)$ becomes $Q(s, a|\vec{\theta})$ where $\vec{\theta}$ are the parameters of the Q-network.

Question

Consider a continuous version of the previous exercise, what would be the input and output dimension of the Q-network?
Same question for a 2D version of the exercise.

Deep Q-learning - training process

Bellman Equation reminder

On average, we want $Q(S_t, A_t) = R_t + \gamma \mathrm{max}_a Q(S_{t+1}, a)$.

- Use a mean squared error between the RHS and LHS.
  - Loss function:
    $E = \frac{1}{2} \sum_t \left[R_t + \gamma \mathrm{max}_a Q(S_{t+1}, a|\vec{\theta}) - Q(S_t, A_t|\vec{\theta}) \right]^2$
  - Update rule: $\theta_i = \theta_t + \frac{\partial E}{\partial \theta_i}$
  - Subtlety - Semi-gradient rule: treat $\mathrm{max}_a Q(S_{t+1}, a|\vec{\theta})$ as a constant w.r.t. $\vec{\theta}$

Replay Buffer

Deep Q-learning pseudocode

Algorithm 1 Deep Q-Learning with Replay Buffer

Input: Environment p

Initialise replay buffer $\mathcal{D}$ to capacity N
Initialize state-action value network $Q(\cdot\,;\theta)$ with parameters θ
for episode $= 1$ to M **do**
    Sample initial state s from environment
    $d :=$ False
    Initalize target parameters $\theta_{\text{target}} := \theta$
    while $d =$ False **do**
        $a := \begin{cases} \arg\max_a Q(s, a; \theta) & \text{prob } 1 - \epsilon \\ \text{random action} & \text{prob } \epsilon \end{cases}$
        Sample (s_{new}, r, d) from environment given (s, a).
        Store experience $(s, a, r, s_{\text{new}}, d)$ in $\mathcal{D}$.
        if Learning on this step **then**
            Sample minibatch $B := \{(s^i, a^i, r^i, s^i_{\text{new}}, d^i)\}$ from $\mathcal{D}$
            $y^j := r^j + \gamma \max_{a'} Q(s^i_{\text{new}}, a'; \theta_{\text{target}})[\![d^i = \text{False}]\!]$
            Let loss $L(\theta) := \frac{1}{|B|} \sum_{i=1}^{|B|} (y^i - Q(s^i, a^i; \theta))^2$
            Gradient descent step $\theta := \theta - \eta \nabla_\theta L(\theta)$
        end if
        if Update target this step **then**
            $\theta_{\text{target}} := \theta$
        end if
    end while
end for

Exercise DQN

Model free Reinforcement Learning

- Temporal Difference (TD)-learning
  First learn the optimal Q-values using the Bellman equation.
  Then find the optimal policy by taking
  $\pi_*(a|s) = \mathbb{1}\{a = \mathrm{argmax}_a Q_*(s,a)\}$.
- **Policy learning**
  Learn directly the policy $\pi(a|s)$ such as to maximize the expected return.

Policy learning

- Directly learn the policy using a neural network that takes the state as input and returns the probability distribution over all possible actions.
- Change the parameters of the network so as to maximize the expected return.