Introduction to Deep Q-Learning

Alexander Van de Kleut[1]

[1][1]NeuroCog Lab Cheriton School of Computer Science University of Waterloo

IEEE Control Systems Magazine, 2002

Outline

The Reward Hypothesis

All of reinforcement learning is based on the idea that:

Reward Hypothesis

Every action of a rational agent can be thought of as seeking to maximize some cumulative scalar reward signal

We formalize this idea using a **Markov Decision Process**.

Markov Process

- A **Markov process** is formally a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$
- $\mathcal{S}$ is a set of states
- $\mathcal{P} : \mathcal{S}^2 \to [0, 1]$ is a transition probability distribution
  - $$\mathcal{P}(s, s') = \mathbb{P}\left[s' | s\right]$$

    the probability of transitioning to state s' given the current state s
- Markov processes are used model stochastic sequences of states $s_1, s_2, \ldots, s_T$ satisfying the **Markov property**:
  - $$\mathbb{P}\left[s_{t+1} | s_1, s_2, \ldots, s_t\right] = \mathbb{P}\left[s_{t+1} | s_t\right]$$

    the probability of transitioning from state s_t to state s_{t+1} is independent of previous transitions.
- We can generate **trajectories** of states using $\mathcal{P}$ of the form $\langle s_1, s_2, \ldots, s_T \rangle$

Markov Reward Process

- A **Markov reward process** is formally a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R} \rangle$ that allows us to associate with each state transition $\langle s_t, s_{t+1} \rangle$ some reward.
- $$\mathcal{R}(s_t, s_{t+1}) = \mathbb{E}\left[r_t | s_t, s_{t+1}\right]$$
  where r_t is the "instantaneous reward"
- We often simplify this to $\mathcal{R}(s_t)$ the reward of being in a particular state s_t
- Given a trajectory beginning at time step t $\langle s_t, s_{t+1}, \ldots, s_T \rangle$ there is an associated sequence of rewards $\langle r_t, r_{t+1}, \ldots, r_T \rangle$
- According to the reward hypothesis, we are interested in trajectories of states that maximize the **return** R_t

Return and Discounted Return

- The **return** R_t is just the cumulative rewards along a trajectory beginning at time stept t
- $$R_t = \sum_{k=t}^{T} r_k$$
- For finite T, we say the trajectory has a **finite time horizon** and is **episodic**
- For infinite T (trajectories are never-ending) we say the trajectory has an **infinite time horizon**
- In this case, R_t might not converge
- Instead we use the **discounted return** G_t
- $$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$$
- where γ is a discount factor between 0 and 1

Markov Decision Process

- A **Markov decision process** (MDP) extends Markov reward processes to make state transitions conditional on some action a_t
- Formally a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \mathcal{A} \rangle$
- $\mathcal{A}$ is a set of actions available to the agent
- State transitions are now

$$\mathcal{P}(s_t, a_t, s_{t+1}) = \mathbb{P}\left[s_{t+1} | s_t, a_t\right]$$

- The probability of transitioning from state s_t to state s_{t+1} after choosing action a_t
- Rewards are now

$$\mathcal{R}(s_t, a_t) = \mathbb{E}\left[r_t | s_t, a_t\right]$$

- The reward depends on the state you're in **and** the action you choose

Value Function

- We can use the expected value of G_t to determine the **value** of being in a state s_t
- $$V(s_t) = \mathbb{E}\left[G_t | s_t\right]$$
- We can decompose $V(s_t)$ into two parts: the immediate reward r_t and the discounted value of being in the next state s_{t+1}
- $$\begin{aligned} V(s_t) &= \mathbb{E}\left[G_t | s_t\right] \\ &= \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t\right] \\ &= \mathbb{E}\left[r_t + \gamma (r_{t+1} + \gamma r_{t+2} + \ldots) | s_t\right] \\ &= \mathbb{E}\left[r_t + \gamma G_{t+1} | s_t\right] \\ V(s_t) &= \mathbb{E}\left[r_t + \gamma V(s_{t+1}) | s_t\right] \end{aligned}$$

  which is known as the **Bellman Equation**

Action-Value Function

- We can also use the expected value of G_t to determine the **quality** of taking a certain action a_t in a state s_t
- $$Q(s_t, a_t) = \mathbb{E}\left[G_t | s_t, a_t\right]$$
- We can decompose $Q(s_t, a_t)$ the same way as $V(s_t)$ using the Bellman equation:

$$Q(s_t, a_t) = \mathbb{E}\left[r_t + \gamma Q(s_{t+1}, a_{t+1}) | s_t, a_t\right]$$

- Then we can see that

$$V(s_t) = \mathbb{E}_{a_t}\left[Q(s_t, a_t)\right]$$

the value of being in state s_t is just the expected quality of being in state s_t over all actions a_t

Policies

- We want to design an agent capable of maximizing the return in a Markov decision process
- We therefore need a method of selecting actions a_t at each timestep t
- We call this the **policy** of the agent
- Policies can be **deterministic**, so that the action taken is a function of the current state:

$$\mu : \mathcal{S} \to \mathcal{A}$$

  with $a_t = \mu(s_t)$
- or can be **stochastic**, where the action is samples from a probability distriution:

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

  with $a_t \sim \pi(\cdot|s_t)$
- In general we can refer to policies as π since μ is a special case of a stochastic policy

Policies

- Given a policy π we define
- $$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[G_t | s_t\right]$$
- $$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\left[G_t | s_t, a_t\right]$$
- Where the superscript π in $V^{\pi}(s_t)$ and $Q^{\pi}(s_t, a_t)$ as well as the subscript π in the expectations simply means "assuming subsequent actions are chosen according to a policy π"

Optimal Policy

- What does it mean for an agent to have an **optimal policy** π^*?
- If Q^* is the best possible Q-value achievable by a policy π then:

$$Q^*(s_t, a_t) = \max_{\pi} \mathbb{E}_{\pi}\left[G_t | s_t, a_t\right]$$

- Then the optimal policy is just

$$\pi^*(a_t|s_t) = \begin{cases} 1 & a_t = \arg\max_{a_t} Q^*(a_t, s_t) \\ 0 & \text{otherwise} \end{cases}$$

i.e., choose the action which maximizes the optimal Q-value

Learning Optimal Policies

- In order to learn the optimal policy π^* we need to learn Q^π, so we can choose the actions that maximize it
- However, determining Q^π exactly is often impossible in practice
- How would you compute the expected value of being in a certain state at a certain time?
- Reinforcement learning does not assume that you have access to the underlying MDP transition dynamics $\mathcal{P}$
- Different approaches have been used to learn Q^π

Q-Learning

- One approach is to try to use **dynamic programming** to learn Q^π
- Assumes that $\mathcal{A}$ and $\mathcal{S}$ are **finite**
- Begin with an approximation $\widehat{Q}^\pi$ which is successively improved to approach Q^π in the limit
- Assume that $\mathcal{S}$ and $\mathcal{A}$ are finite
- Then we can use dynamic programming to approximate Q^π
- Create a table of size $|\mathcal{S}| \times |\mathcal{A}|$
- Store estimates $\widehat{Q}^\pi(s_t, a_t)$ for each pair (s_t, a_t) in the table

Q-Learning

- Consider the following agent following a greedy policy π
  1. You are in a state s_t
  2. You choose an action a_t which maximizes $\widehat{Q}^\pi(s_t, a_t)$ (greedy policy)
  3. You get a reward r_t
  4. You transition to the next state s_{t+1}
- At this point, you have a slightly better estimate for $Q^\pi(s_t, a_t)$ according to the Bellman equation, namely

$$r_t + \gamma \max_{a_{t+1}} \widehat{Q}^\pi(s_{t+1}, a_{t+1})$$

  (since we are following a greedy policy)
- Update the table to be closer to the better estimate with some learning rate α

$$\widehat{Q}^\pi(s_t, a_t) \leftarrow (1 - \alpha)\widehat{Q}^\pi(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a_{t+1}} \widehat{Q}^\pi(s_{t+1}, a_{t+1}) \right)$$

Deep Q-Learning

- The issue with Q-learning is that it assumes that both $\mathcal{S}$ and $\mathcal{A}$ are finite
- We can apply the methods of Q-learning to MDPs with infinite $\mathcal{S}$ by using function approximation
- Instead of a table, $\widehat{Q}^{\pi}_{\theta}$ is a function approximator (such as a neural network) with parameters θ
- Given a state-action pair (s_t, a_t) our function approximator makes a prediction $\widehat{Q}^{\pi}_{\theta}(s_t, a_t)$
- We have a target

$$y = r_t + \gamma \max_{a_{t+1}} \widehat{Q}^{\pi}_{\theta}(s_{t+1}, a_{t+1})$$

- We want minimize the loss (e.g. via SGD)

$$L(\theta) = \left(\widehat{Q}^{\pi}_{\theta}(s_t, a_t) - y\right)^2$$

Target Networks

- Generally in an MDP, two consecutive states s_t and s_{t+1} are similar
- If our function approximator is differentiable (like a neural network), then:
  - If for a state s_t we update $\widehat{Q}^{\pi}_{\theta}(s_t, a_t)$ to be slightly higher, this will **also** make it so that $\widehat{Q}^{\pi}_{\theta}(s_{t+1}, a_{t+1})$ is higher, since s_{t+1} tends to be "nearby" to s_t
- This leads to instability in the training with runaway Q-values
- To solve this, we use **two** function approximators with parameters θ and θ^-
- θ^- is a copy of θ that we only use to generate the target value y, and it only gets synchronized every n_θ updates.

Replay Buffer

- If we update the parameters θ at each timestep, then we are only training on the most recent state-action pairs (s_t, a_t)
- This can result in **catastrophic forgetting**, where we perform worse on state-action pairs that we saw a long time ago
- To alleviate this, we can instead train each timestep on a **batch** of data stored in a **replay buffer** $\mathcal{D}$
- Assuming a greedy policy, we need to store transitions $\langle s_t, a_t, r_t, s_{t+1} \rangle$
- We can sample batches of transitions randomly from the replay buffer and train the function approximator on them

Terminal States

- Some states are **terminal states**, meaning that once you reach state s_t, every following state is equal to S_t and the reward is 0
- In these cases, we can reset the agent to a new starting state and treat this as s_{t+1}
- We can augment each transition with a **done** flag d_t indicating whether or not we entered a terminal state
- We can provide the agent with information about "doneness" when calculating the target:

$$y = r_t + (1 - d_t)\gamma \max_{a_{t+1}} \widehat{Q}^{\pi}_{\theta}(s_{t+1}, a_{t+1})$$

where multiplying by $(1 - d_t)$ helps the agent learn that $G_t = r_t$ (there are no more rewards after this transition)

ϵ-Greedy Policies

- When we first start out, our estimate for Q^{π} is probably very wrong
- It does not make sense to follow a greedy policy that chooses the maximal $\widehat{Q}_{\theta}^{\pi}$
- Instead we can start by selecting actions randomly to get an idea of how good they are
- Over time we can rely more and more on our greedy policy
- We can use an ϵ-greedy policy: Take a random action with some probability ϵ, otherwise follow the policy π
- One strategy is to linearly anneal ϵ from some value ϵ_i to ϵ_f over a number of iterations n_{ϵ}

Deep Q-learning algorithm

$\theta \leftarrow \theta_0$
$s_t \leftarrow s_0$
$\epsilon \leftarrow \epsilon_i$
for $t \leftarrow 0 \dots T$ **do** :
    if $t \mod n_\theta \equiv 0$ **then**:
        $\theta^- \leftarrow \theta$
    $a_t \sim \pi(\cdot|s_t)$ with ϵ-greedy policy
    $r_t \sim \mathcal{R}(s_t, a_t)$
    $s_{t+1} \sim \mathcal{P}(s_t, a_t, \cdot)$
    $d_t \leftarrow$ terminal state flag
    $\mathcal{D}$.push$(\langle s_t, a_t, r_t, s_{t+1}, d_t \rangle)$
    $B \leftarrow$ batch of data from $\mathcal{D}$
    for $s_k, a_k, r_k, s_{k+1}, d_k$ in B **do**:
        $y \leftarrow r_k + \gamma \max_{a_{k+1}} \widehat{Q}^\pi_{\theta^-}(s_{k+1}, a_{k+1})$
        minimize $L(\theta) = \left(\widehat{Q}^\pi_\theta(s_k, a_k) - y\right)^2$ with respect to θ
    Anneal ϵ towards ϵ_f