An Introduction to Deep Reinforcement Learning

Part 1 – MDPs, Dynamic Programming, Q-Learning, Deep Q-Learning

Karim Bouyarmane
The *subject* of Reinforcement Learning are **Markov Decision Processes (MDP)**
More precisely, Reinforcement Learning is a Machine Learning approach to solving MDPs.
MDP: simplest possible probabilistic model of “something” that can “take actions”/decisions and act on itself or on the world

agent/world with states $s \in S$ and possible actions $a \in A$

(e.g. physical robot, trading agent, video-game playing agent, continuous decision maker in dynamic and uncertain environment, etc.)

Two models and one parameter are necessary to fully characterize the MDP:

- a transition model $P(s' | s, a) = T(s, a, s')$ (a.k.a dynamics model, “what is the effect of an action?”)
- a reward model at state $s$ : $R(s) \in \mathbb{R}$ (“what is our objective? what state are we trying to reach?”) (or $R(s, a)$, or even $R(s, a, s')$, etc.)
- a discount factor $0 < \gamma \leq 1$ (trade-off between immediate reward and delayed reward, “cost of delayed reward”)

\[
\begin{array}{cccc}
\text{s} & \text{a} & \text{s'} \\
\downarrow & \text{r} & \downarrow
\end{array}
\]
**MDP:** simplest possible probabilistic model of “something” that can “take actions”/decisions and act on itself or on the world

agent/world with states \( s \in \mathcal{S} \) and possible actions \( a \in \mathcal{A} \)

(e.g. physical robot, trading agent, video-game playing agent, continuous decision maker in dynamic and uncertain environment, etc.)

Two models and one parameter are necessary to fully characterize the MDP:

- a **transition model** \( P(s' | s, a) = T(s, a, s') \) (a.k.a dynamics model, “what is the effect of an action?”)
- a **reward model** at state \( s : R(s) \in \mathbb{R} \) (“what is our objective? what state are we trying to reach?”) (or \( R(s, a) \), or even \( R(s, a, s') \), etc.)
- a **discount factor** \( 0 < \gamma \leq 1 \) (trade-off between immediate reward and delayed reward, “cost of delayed reward”)
MDP = (S, A, T, R, γ)
MDP

\[ s \xrightarrow{a} s' \]

\[ s \xrightarrow{r} r \]

\[ a_0 \]

\[ s_0 \]

\[ a_1 \]

\[ s_1 \]

\[ s_2 \]

\[ a_2 \]

\[ s_2 \]

\[ s_3 \]

\[ \ldots \]

\[ a_0 \]

\[ s_0 \]

\[ a_1 \]

\[ s_1 \]

\[ s_2 \]

\[ a_2 \]

\[ s_2 \]

\[ s_3 \]

\[ \ldots \]
\[
\sum_{t=0}^{+\infty} \gamma^t r_t
\]

return
**Policy** = deciding what action to take at every state  \( \pi: s \mapsto a \)

(a.k.a. “feedback loop”, “control law”, “control policy”, “decision function”, etc.)

“autonomous agent” = agent that follows (“is endowed with”) a policy \( \pi \)
“Solving” an MDP = solving for a **policy** $\pi : S \rightarrow \mathcal{A}$

$$\pi(s) = a$$
autonomous agent that follows a policy $\pi$
The autonomy feedback loop revisited

\[ \begin{align*}
\text{Sense} & \rightarrow \text{Plan} \\
\text{Plan} & \rightarrow \text{Act} \\
\text{Act} & \rightarrow \text{Sense}
\end{align*} \]
We don’t want to find any policy, we want to find a good policy

A good policy is a policy that takes actions that make the agent maximize its long-term rewards (or returns), i.e. that makes the agent realize a certain objective (the objective being encoded in the reward/returns model)

The art of formulating a good MDP is thus formulating a good reward model that captures the desired objective

A good policy can also be interpreted a policy that minimizes cost ($cost = -reward$)

Examples of long term rewards:
- Winning a game
- Accomplishing a task successfully
- Reaching a goal position
- Making stock gains at a certain maximum horizon
Definition: A sequence of states $s_t$ follows a policy $\pi$ if

$$\forall t \geq 0, \quad s_{t+1} \sim P(s_{t+1} | s_t, \pi(s_t))$$

We write $s_t \sim \pi$
So, we want to find the **optimal** policy $\pi^*$

$$
\pi^* = \arg\max_{\pi} \mathbb{E}_{s_t \sim \pi} \left[ \sum_{t=0}^{+\infty} \gamma^t R(s_t) \mid \pi \right]
$$

Where

$s_0 \sim$ given distribution

$s_{t+1} \sim P(s_{t+1} \mid s_t, \pi(s_t))$
Solving for the optimal policy is thus an **optimization problem (optimal control)** over the space of policies ($\mathcal{A}^S$)

Different families of methods for solving MDP

- **Non-ML MDP Solving: Dynamic programming** methods
  - Value iteration
  - Policy iteration

- **Q-learning** methods (DQN)

- **Policy gradient** methods (Actor-Critic)

- **Evolution strategies** or DFO: Derivative-Free Optimization (CMA-ES)
Value of a state (or Utility of a state) V-value (or U-value)

\[ V(s) \text{ (or } U(s)) = \mathbb{E}_{s_t \sim \pi^*} \left[ \sum_{t=0}^{+\infty} \gamma^t R(s_t) | s_0 = s, \pi^* \right] \]
Value of a state (or Utility of a state) V-value (or U-value):

“Best returns we can hope for in average, if we start from the state”

(meaning that we start from the state, and follow the optimal policy)
If we knew the V-value of every state, then the optimal policy at any given state is to take the action that gives you the best chance to land on the highest-value state

\textit{i.e.} \textbf{optimal policy = “follow the V-values”}
If we knew the V-value of every state, then the optimal policy at any given state is to take
the action that gives you the best chance to land on the highest-value state

\[ \pi^*(s) = \arg\max_a \sum_{s'} T(s, a, s')V(s') \]

\textit{i.e. optimal policy = “follow the V-values”}
If we knew the V-value of every state, then the optimal policy is

\[ \pi^*(s) = \arg\max_a \sum_{s'} T(s, a, s')V(s') \]
\[ V(s) = \mathbb{E} \left[ \sum_{t=0}^{+\infty} \gamma^t R(s_t) \mid s_0 = s, \pi^* \right] \] \[ \iff \quad \pi^*(s) = \arg\max_a \sum_{s'} T(s, a, s') V(s') \]

"\( V = f(\pi^*) \)" \[ \iff \quad \pi^* = f^{-1}(V)\]
Introducing Gridworld®

\[ \mathcal{A} = \{\uparrow, \leftarrow, \downarrow, \rightarrow\} \]
\[ \mathcal{S} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\} \]

for every state \( s \in \mathcal{S} \), let us denote:

- \( s^\uparrow \) the state immediately to the north of \( s \) (if it exists)
- \( s^\downarrow \) the state immediately to the south of \( s \) (if it exists)
- \( s^\leftarrow \) the state immediately to the west of \( s \) (if it exists)
- \( s^\rightarrow \) the state immediately to the east of \( s \) (if it exists)

the transition model is:

\[
P(s^\uparrow | s, \uparrow) = 0.8 \quad P(s^\rightarrow | s, \rightarrow) = 0.8
\]
\[
P(s^\downarrow | s, \downarrow) = 0.1 \quad P(s^\leftarrow | s, \leftarrow) = 0.1 \quad \ldots
\]
\[
P(s^\rightarrow | s, \uparrow) = 0.1 \quad P(s^\downarrow | s, \leftarrow) = 0.1
\]

+ If the robot bumps into a wall, it stays in the same state

the reward model is:

\[
R(11) = +1
\]
\[
R(10) = -1
\]
\[
R(s \neq 10 \text{ and } 11) = -0.04
\]

the discount factor is

\[ \gamma = 1 \]
Optimal policy $\pi^*$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>←</td>
<td>↑</td>
<td>↑</td>
<td>←</td>
</tr>
<tr>
<td>↑</td>
<td></td>
<td></td>
<td>←</td>
</tr>
<tr>
<td>←</td>
<td></td>
<td></td>
<td>←</td>
</tr>
</tbody>
</table>
Value of every state $V$

<table>
<thead>
<tr>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.812</td>
<td>0.868</td>
<td>0.918</td>
<td>+1</td>
</tr>
<tr>
<td>0.762</td>
<td>0.660</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0.705</td>
<td>0.655</td>
<td>0.611</td>
<td>0.388</td>
</tr>
</tbody>
</table>
\[ V(s) = \mathbb{E} \left[ \sum_{t=0}^{+\infty} \gamma^t R(s_t) \mid s_0 = s, \pi^* \right] \]

\[ \pi^*(s) = \arg\max_a \sum_{s', a} T(s, a, s') V(s') \]

**Value of every state \( V \)**

<table>
<thead>
<tr>
<th>0.812</th>
<th>0.868</th>
<th>0.918</th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.762</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.705</td>
<td>0.655</td>
<td>0.611</td>
<td>0.388</td>
</tr>
</tbody>
</table>

**Optimal policy \( \pi^* \)**

\[ \rightarrow \rightarrow \rightarrow +1 \]

\[ \uparrow \uparrow \rightarrow \times \]

\[ \uparrow \leftarrow \leftarrow \leftarrow \]
The value function or the optimal policy, completely characterize the optimal solution of an MDP.
Bellman equation:

\[ V(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') V(s') \]
Bellman equation:

\[ V(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') V(s') \]

In the example state (the one with value 0.918), if the agent follows the optimal action (which is “go to right”), then it has 80% chance of actually going to the right, landing in a state of value 1, 10% chance of going up, bumping into the wall, and thus staying in the same state with value 0.918, and 10% chance of going down, landing in the state of value 0.660, i.e.

\[ 0.8 \times 1 + 0.1 \times 0.660 + 0.1 \times 0.918 - 0.04 = 0.918 \]
Bellman equation:

\[
V(s_1) = R(s_1) + \gamma \max_a T(s_1, a, s_1)V(s_1) + T(s_1, a, s_2)V(s_2) + T(s_1, a, s_3)V(s_3) + \ldots
\]

\[
V(s_2) = R(s_2) + \gamma \max_a T(s_2, a, s_1)V(s_1) + T(s_2, a, s_2)V(s_2) + T(s_2, a, s_3)V(s_3) + \ldots
\]

\[
V(s_3) = R(s_3) + \gamma \max_a T(s_3, a, s_1)V(s_1) + T(s_3, a, s_2)V(s_2) + T(s_3, a, s_3)V(s_3) + \ldots
\]

;
Solving Bellman equation ⇒ Solving for $V$ ⇒ Obtaining $\pi^*$
The Bellman equation

\[ V(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') V(s') \]

is a fixed-point equation

“\( V = Bellman(V) \)”
To solve a fixed-point equation, we apply the iteration method:

Initialize a random $V_0$

$$V_{k+1} = Bellman(V_k)$$

$$\lim_{k \to +\infty} V_k = V$$
**Value-iteration** (for finding the optimal policy of an MDP)

- Initialize $V_0(s)$ at some random values at all states $s$
  - Apply Iterative Bellman equation
    $V_{k+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')V_k(s')$
  - Loop
  - Until convergence of $V_k(s)$ to some value $V(s)$
  - Apply
    $\pi^*(s) = \arg\max_a \sum_{s'} T(s, a, s')V(s')$
There is another method very similar to Value-iteration, that solves directly for $\pi^*$ called **Policy-iteration**.
Reminder - Bellman equation:

\[ V(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') V(s') \]

Linear Bellman equation, by definition of \( \pi^* \):

\[ V(s) = R(s) + \gamma \sum_{s'} T(s, \pi^*(s), s') V(s') \]
Policy-iteration (for finding the optimal policy of an MDP)

- Initialize $\pi_0(s)$ at some random values at all states $s$
  - Solve linear Bellman equation for $V_k$, given optimal policy $\pi_k$
    \[ V_k(s) = R(s) + \gamma \sum_{s'} T(s, \pi_k(s), s') V_k(s') \]
  - Update
    \[ \pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') V_k(s') \]
  - Loop
  - Until convergence of $\pi_k(s)$ to some value $\pi^*(s)$
Value-iteration and Policy-iteration are **exact methods** to solve MDPs, they are not Machine Learning approaches.
Why would we need Machine Learning to solve MDPs anyways?

Usually we don’t know the transition and reward model a priori, we don’t know $T$ and $R$

We can only observe some sample data from $T$ and $R$ by making the agent actually perform in real world or simulate different actions $a$ at different states $s$, then record what state $s'$ we ended up in and what reward $r$ we got as a result from that action at that state

Records of observed data from experiences will be in the form of tuples $(s, a, s', r)$
Classical MDP solving: Model-based
input: model \((T, R)\), output: policy

\[ T, R \rightarrow \pi^*, V \]

Reinforcement Learning for solving MDPs: Data-based
Input experience records (data) \(\langle s, a, s', r \rangle\), output policy

\[ \langle s_1, a_1, s'_1, r_1 \rangle \]
\[ \langle s_2, a_2, s'_2, r_2 \rangle \]
\[ \langle s_3, a_3, s'_3, r_3 \rangle \]
\[ \langle s_4, a_4, s'_4, r_4 \rangle \]
\[ \langle s_5, a_5, s'_5, r_5 \rangle \]
\[ \langle s_6, a_6, s'_6, r_6 \rangle \]
\[ \ldots \]
Reinforcement learning template:

- Start with a random policy
  - Following this policy (exploitation) interleaved with some random actions from time to time (exploration), make the agent collect experience record tuples
  - Refine the policy based the knowledge received from these actions and these observations
  - Loop
- Until the policy converges
What is it exactly that we “learn”?

- **Not directly the model** $T$ and $R$, since we only care about the policy $\pi$

- **Maybe learn V-value of every state?** too coarse, we don’t have experience data directly associated with states $s$, but with actions $a$ taken at state $s$

- We introduce a new quantity that refines V-values ⇒ by giving value to a pair of <action, state> the **Q-value of an action $a$ at state $s$**
V-value of a state:

\[ V(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') V(s') \]
Q-value of an action at a state:

\[ Q(s, a) = R(s) + \gamma \sum_{s'} T(s, a, s') V(s') \]
Q-value of an action at a state:

\[ Q(s, a) = R(s) + \gamma \sum_{s'} T(s, a, s') V(s') \]

Q-value is also known as action-value, as opposed to V-value which is known as state-value.
**Q-value of an action** at a state:

Best returns we can hope for if we take action $a$ at state $s$

(meaning that we take action $a$ at state $s$ and then start following the optimal policy from whatever state $s'$ we land at)
Value of every state $V$

<table>
<thead>
<tr>
<th></th>
<th>0.812</th>
<th>0.868</th>
<th>0.918</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.762</td>
<td>0.660</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.705</td>
<td>0.655</td>
<td>0.611</td>
<td>0.388</td>
</tr>
</tbody>
</table>

Optimal policy $\pi^*$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>→</td>
<td>→</td>
<td>+1</td>
</tr>
<tr>
<td>↑</td>
<td>↑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>↑</td>
<td>←</td>
<td>←</td>
<td>←</td>
</tr>
</tbody>
</table>

Q-value of every action in every state

*Not all values displayed, just example values*
Value of every state $V$

<table>
<thead>
<tr>
<th></th>
<th>0.812</th>
<th>0.868</th>
<th>0.918</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.762</td>
<td>0.660</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.705</td>
<td>0.655</td>
<td>0.611</td>
<td>0.388</td>
</tr>
</tbody>
</table>

Optimal policy $\pi^*$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>→</td>
</tr>
<tr>
<td>↑</td>
<td></td>
<td>↑</td>
<td></td>
</tr>
<tr>
<td>→</td>
<td></td>
<td></td>
<td>↑</td>
</tr>
</tbody>
</table>

Q-value of every action in every state

<table>
<thead>
<tr>
<th></th>
<th>...</th>
<th>...</th>
<th>0.812</th>
<th>...</th>
<th>...</th>
<th>0.868</th>
<th>...</th>
<th>...</th>
<th>0.881</th>
<th>0.812</th>
<th>0.918</th>
<th>...</th>
<th>...</th>
<th>0.675</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.762</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.705</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Not all values displayed, just example values*
Q-value of every action in every state

<table>
<thead>
<tr>
<th></th>
<th>0.812</th>
<th>0.868</th>
<th>0.881</th>
<th>0.918</th>
<th>0.675</th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.762</td>
<td></td>
<td></td>
<td>0.660</td>
<td></td>
<td>0.415</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.705</td>
<td>0.655</td>
<td>0.611</td>
<td></td>
<td>-0.740</td>
<td></td>
</tr>
<tr>
<td>0.655</td>
<td></td>
<td></td>
<td></td>
<td>0.388</td>
<td>0.209</td>
<td></td>
</tr>
</tbody>
</table>

Optimal policy $\pi^*$

- →
- ↑
- ←
If we knew the Q-value of every action at every state, then finding the optimal policy is straightforward

$$\pi^*(s) = \arg\max_a Q(s, a)$$

And finding the V-value of a state is also straightforward

$$V(s) = \max_a Q(s, a)$$

The optimal policy is guided by the Q values

*i.e. optimal policy = “follow the Q-values”*
Relationship between $V$, $Q$, $\pi^*$

$$V(s) = Q(s, \pi^*(s))$$
Bellman equation for the Q-value

$$Q(s, a) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')$$
Bellman equation for the Q-value

\[
Q(s, a) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')
\]

\[
Q(s, a) = \mathbb{E}_{s'} \left[ R(s) + \gamma \max_{a'} Q(s', a') | s, a \right]
\]
Q-learning (for finding the optimal policy of an MDP) with learning rate $\alpha$

- Initialize $Q(s, a)$ at some random values at all states $s$ and actions $a$ (i.e. initialize random policy)
- Start in state $s_0$
- Set current state $s = s_0$
  - From current state $s$, choose action $a$ by picking one of these two choices ($\epsilon$-greedy strategy):
    - [Exploitation, Being greedy] Either by following the current policy $\arg\max_a Q(s, a)$
    - [Exploration, with probability $\epsilon$] Or by picking a completely random action $a$
  - Execute $a$
  - Observe the landed state $s'$, and the obtained reward $r$ (we have now collected an experience record data point $(s, a, s', r)$)
  - From this observation, update value of $Q(s, a)$ by taking a stochastic gradient step towards
    $$\hat{Q}_{Bellman}(s, a|s', r) = r + \gamma \max_{a'} Q(s', a')$$
    $$Q(s, a) = Q(s, a) + \alpha [\hat{Q}_{Bellman}(s, a|s', r) - Q(s, a)]$$
  - Update current state $s = s'$
  - Loop
- Until convergence of $Q$/convergence of $\pi^*$

Keep in mind: policy $\equiv$ Q-value $\pi^*(s) = \arg\max_a Q(s, a)$
This approach is called **Tabular Q-learning**, which means it tries to build a table of Q-values for every (state, action) pair.

Problematic with continuous state spaces or continuous action spaces.

Even with discretization and finite state space: huge number of states (Tetris has $10^{60}$ states × 3 actions).
Solution: use function approximation with parametric model

instead of learning $Q(s,a)$ for every $(s,a)$, parameterize $Q$ as $Q_\theta$ (for example linear model, neural network), and learn the parameter $\theta$ from the observations, this is called Approximate Q-learning

when $Q_\theta$ is a deep learning model (for example a CNN, taking the raw pixels of the game as the state of the game), then we talk about Deep Reinforcement Learning
Deep RL = Q-value of every action as a deep-learning regression model, called the Q-network

Note that it is different from a supervised learning problem as a classification problem on the actions from the observation of the actions taken by human agents. Here there is no human agent, the agent generates the data it needs and learns a Q-value function.
Approximate Q-learning algorithm (for finding the optimal policy of an MDP) with learning rate $\alpha$

- Initialize $\theta$ at some random values (i.e. initialize random policy)
- Start in state $s_0$
- Set current state $s = s_0$
  - From current state $s$, choose action $a$ by picking one of these two choices:
    - [Exploitation] Either by following the current policy $\arg\max_a Q_\theta(s, a)$
    - [Exploration] Or by picking a completely random action $a$
  - Execute/simulate $a$
  - Observe the landed state $s'$, and the obtained reward $r$ (we have now collected an experience record data point $(s, a, s', r)$)
  - From this observation, update value of $Q_\theta(s, a)$ by taking a stochastic gradient step towards
    $$
    \hat{Q}_{\text{Bellman}}(s, a|s', r) = r + \gamma \max_{a'} Q_\theta(s', a')
    $$
    $$
    \theta = \theta - \alpha \frac{\partial}{\partial \theta} \left[ \hat{Q}_{\text{Bellman}}(s, a|s', r) - Q_\theta(s, a) \right]^2 \bigg|_{\theta}
    $$
- Update current state $s = s'$
- Loop
- Until convergence of $Q$ / convergence of $\pi^*$

Keep in mind: policy $\equiv$ Q-value

\[ \pi^*(s) = \arg\max_a Q_\theta(s, a) \]
Approximate Q-learning algorithm (for finding the optimal policy of an MDP) with learning rate $\alpha$

- Initialize $\theta$ at some random values (i.e. initialize random policy)
- Start in state $s_0$
- Set current state $s = s_0$
  - From current state $s$, choose action $a$ by picking one of these two choices:
    - [Exploitation] Either by following the current policy $\text{argmax}_a Q_\theta(s, a)$
    - [Exploration] Or by picking a completely random action $a$

- Execute/simulate $a$
- Observe the landed state $s'$, and the obtained reward $r$ (we have now collected an experience record data point $<s, a, s', r>$)
- From this observation, update value of $Q_\theta(s, a)$ by taking a stochastic gradient step towards
  $$\hat{Q}_{\text{Bellman}}(s, a|s', r) = r + \gamma \max_{a'} Q_\theta(s', a')$$
  $$\theta = \theta - \alpha \frac{\partial}{\partial \theta} [\hat{Q}_{\text{Bellman}}(s, a|s', r) - Q_\theta(s, a)]$$

- Update current state $s = s'$
- Loop
- Until convergence of $Q$ / convergence of $\pi^*$

Keep in mind: policy $\equiv$ Q-value
$$\pi^*(s) = \text{argmax}_a Q_\theta(s, a)$$

Chasing a moving target

Failed iid assumption for SGD

Two problems with this naïve approach
To stabilize Approximate Q-learning, Minh et al, 2015, introduce two improvements:

• **Experience replay**, store 1M transitions (experience data point) in memory buffer, then sample minibatches from those for SGD, don’t use current current transition for SGD, store it in memory buffer

• Use **target network** to compute the target of Q, update target network with Q-network every 10000 iterations
DQN algorithm with experience replay (for finding the optimal policy of an MDP) with learning rate $\alpha$

- Initialize $\theta$ (Q-network) at some random values (i.e. initialize random policy), initialize $\theta$ to $\theta$ ($\theta$ is the target network, target network=Q-network at initialization)
- Start in state $s_0$
- Set current state $s=s_0$
  - From current state $s$, choose action $a$ by picking one of these two choices:
    - [Exploitation] Either by following the current policy $\text{argmax}_a Q_\theta(s, a)$
    - [Exploration] Or by picking a completely random action $a$
  - Execute/simulate $a$
  - Observe the landed state $s'$, and the obtained reward $r$ (we have now collected an experience record data point$(s, a, s', r)$)
  - Store $(s, a, s', r)$ in replay buffer $\mathcal{D}$ (buffer capacity 1M, FIFO)
  - Sample minibatches of 32 tuples $(s_i, a_i, s'_i, r_i)$ of iid from $\mathcal{D}$ to perform SGD
  - [Update Q-network only, not target network] On that minibatch, update value of $Q_\theta(s_i, a_i)$ by taking a stochastic gradient step towards

$$\hat{Q}_{\text{Bellman},i}(s_i, a_i | s'_i, r_i) = r_i + \gamma \max_a Q_{\theta^-}(s'_i, a')$$

$$\theta = \theta - \alpha \frac{\partial}{\partial \theta} \mathbb{E}_{(s_i, a_i, s'_i, r_i)} \left[ \hat{Q}_{\text{Bellman},i}(s_i, a_i | s'_i, r_i) - Q_\theta(s_i, a_i) \right]^2 \bigg|_\theta$$

- Every 10000 iteration reset $\theta$ to $\theta$ (reset target network to Q-network)
- Update current state $s=s'$
- Loop
- Until convergence of $Q$/convergence of $\pi^*$