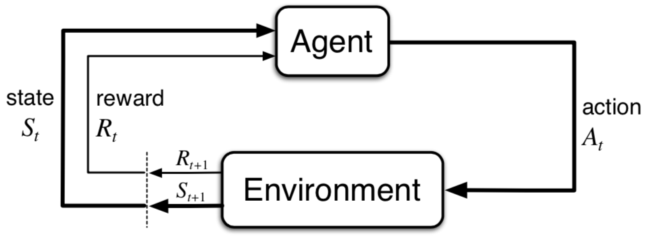
Reinforcement Learning

Reinforcement learning is the filed of learning to make decisions. The goal of reinforcement learning is for an agent to learn how to evolve in an environment.



The Agent at each step t receives a representation of the environment's *state*, $S_t \in S$ and it selects an action $A_t \in A(s)$. Then, as a consequence of its action the agent receives a *reward*, $R_{t+1} \in R \in \mathbb{R}$.
Policy
A *policy* is a mapping from a state to an action

$$\pi_t(s|a) \tag{1}$$

That is the probability of select an action $A_t = a$ if $S_t = s$.
Reward
The total *reward* is expressed as:

$$G_t = \sum_{k=0}^{H} \gamma^k r_{t+k+1} \tag{2}$$

Where γ is the *discount factor* and H is the *horizon*, that can be infinite.

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

1 Exploration Exploitation Dilemma

1.1 Exploration

Exploring potential hypotheses for how to choose actions

1.2 Exploitation

Exploiting limited knowledge about what is already known should work well.

2 Episodic vs. Continuing Tasks

Episodic task: interaction ends at some time step T.

$$S_0, A_0, R_1, S_1, A_1, ..., R_T, S_T$$

When the episode ends the agent look at the total reward he received to figure out how well he did. Then he's able to stare from scratch with the knowledge of the paste live.
Continuing task: interaction continues without limit.

$$S_0, A_0, R_1, S_1, A_1, ...$$

3 Markov Decision Process

A **Markov Decision Process**, MDP, is a 5-tuple (S, A, P, R, γ) where:

$$\begin{aligned} &\text{finite set of states:} \\ &s \in S \\ &\text{finite set of actions:} \\ &a \in A \\ &\text{state transition probabilities:} \\ &p(s'|s,a) = Pr\{S_{t+1} = s' | S_t = s, A_t = a\} \\ &\text{expected reward for state-action-next\_state:} \\ &r(s', s, a) = \mathbb{E}[R_{t+1} | S_{t+1} = s', S_t = s, A_t = a] \end{aligned} \tag{3}$$

4 Value Function

Value function describes *how good* is to be in a specific state s under a certain policy π. For MDP:

$$V_\pi(s) = \mathbb{E}[G_t | S_t = s] \tag{4}$$

Informally, is the expected return (expected cumulative discounted reward) when starting from s and following π

4.1 Optimal

$$V_*(s) = \max_\pi V_\pi(s) \tag{5}$$

5 Action-Value (Q) Function

We can also denoted the expected reward for state, action pairs.

$$q_\pi(s,a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] \tag{6}$$

5.1 Optimal

The optimal value-action function:

$$q_*(s,a) = \max_\pi q^\pi(s,a) \tag{7}$$

Clearly, using this new notation we can redefine V^*, equation 5, using $q^*(s,a)$, equation 7:

$$V_*(s) = \max_{a \in A(s)} q_{\pi*}(s,a) \tag{8}$$

Intuitively, the above equation express the fact that the value of a state under the optimal policy **must be equal** to the expected return from the best action from that state.

6 Bellman Equations

An important recursive property emerges for both Value (4) and Q (6) functions if we expand them.

6.1 Value Function

$$\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi\left[G_t|S_t=s\right] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t=s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}+\gamma\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}|S_t=s\right] \\
&= \underbrace{\sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)}_{\text{Sum of all probabilities } \forall \text{ possible } r} \qquad (9) \\
&\left[r+\gamma\underbrace{\mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}|S_{t+1}=s'\right]}_{\text{Expected reward from } s_{t+1}}\right] \\
&= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)\left[r+\gamma V_\pi(s')\right]
\end{aligned}$$

Similarly, we can do the same for the Q function:

$$\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t=s,A_t=a\right] \\
&= \mathbb{E}_\pi\left[R_{t+1}+\gamma\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}|S_t=s,A_t=a\right] \\
&= \sum_{s',r} p(s',r|s,a)\left[r+\gamma\mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}|S_{t+1}=s'\right]\right] \\
&= \sum_{s',r} p(s',r|s,a)\left[r+\gamma V_\pi(s')\right]
\end{aligned} \qquad (10)$$

6.2 Bellman Expectation Equations

$$\boxed{v_\pi(s) = \sum_{a\in\mathcal{A}(s)}\pi(a|s)\sum_{s'\in\mathcal{S},r\in\mathcal{R}}p(s',r|s,a)(r+\gamma v_\pi(s'))}$$

$$q_\pi(s,a) = \sum_{s'\in\mathcal{S},r\in\mathcal{R}}p(s',r|s,a)(r+\gamma\sum_{a'\in\mathcal{A}(s')}\pi(a'|s')q_\pi(s',a'))$$

6.3 Bellman Optimality Equations

$$\boxed{v_*(s) = \max_{a\in\mathcal{A}(s)}\sum_{s'\in\mathcal{S},r\in\mathcal{R}}p(s',r|s,a)(r+\gamma v_*(s'))}$$

$$\boxed{q_*(s,a) = \sum_{s'\in\mathcal{S},r\in\mathcal{R}}p(s',r|s,a)(r+\gamma\max_{a'\in\mathcal{A}(s')}q_*(s',a'))}$$

6.4 Useful Formulas for Deriving the Bellman Equations

$$v_\pi(s) = \sum_{a\in\mathcal{A}(s)}\pi(a|s)q_\pi(s,a)$$

$$v_*(s) = \max_{a\in\mathcal{A}(s)}q_*(s,a)$$

$$q_\pi(s,a) = \sum_{s'\in\mathcal{S},r\in\mathcal{R}}p(s',r|s,a)(r+\gamma v_\pi(s'))$$

$$q_*(s,a) = \sum_{s'\in\mathcal{S},r\in\mathcal{R}}p(s',r|s,a)(r+\gamma v_*(s'))$$

$$\begin{aligned}
q_\pi(s,a) &\doteq \mathbb{E}_\pi[G_t|S_t=s,A_t=a] && (1)\\
&= \sum_{s'\in\mathcal{S},r\in\mathcal{R}}\mathbb{P}(S_{t+1}=s',R_{t+1}=r|S_t=s,A_t=a) \\
&\quad \mathbb{E}_\pi[G_t|S_t=s,A_t=a,S_{t+1}=s',R_{t+1}=r] && (2)\\
&= \sum_{s',r}p(s',r|s,a)\mathbb{E}_\pi[G_t| \\
&\quad S_t=s,A_t=a,S_{t+1}=s',R_{t+1}=r] && (3)\\
&= \sum_{s'\in\mathcal{S},r\in\mathcal{R}}p(s',r|s,a)\mathbb{E}_\pi[G_t|S_{t+1}=s',R_{t+1}=r] && (4)\\
&= \sum_{s'\in\mathcal{S},r\in\mathcal{R}}p(s',r|s,a)\mathbb{E}_\pi[R_{t+1}+\gamma G_{t+1}| \\
&\quad S_{t+1}=s',R_{t+1}=r] && (5)\\
&= \sum_{s'\in\mathcal{S},r\in\mathcal{R}}p(s',r|s,a)(r+\gamma\mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']) && (6)\\
&= \sum_{s'\in\mathcal{S},r\in\mathcal{R}}p(s',r|s,a)(r+\gamma v_\pi(s')) && (7)
\end{aligned}$$

The reasoning for the above is as follows:

- (1) by definition ($q_\pi(s,a)\doteq\mathbb{E}_\pi[G_t|S_t=s,A_t=a]$)
- (2) Law of Total Expectation
- (3) by definition ($p(s',r|s,a)\doteq\mathbb{P}(S_{t+1}=s',R_{t+1}=r|S_t=s,A_t=a)$)
- (4) $\mathbb{E}_\pi[G_t|S_t=s,A_t=a,S_{t+1}=s',R_{t+1}=r]=\mathbb{E}_\pi[G_t|S_{t+1}=s',R_{t+1}=r]$
- (5) $G_t=R_{t+1}+\gamma G_{t+1}$
- (6) Linearity of Expectation
- (7) $v_\pi(s')=\mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']$

7 Dynamic programming

Dynamic programming (**DP**) is a technique for solving complex problems. Instead of solving complex problems directly, we break the problem into simple sub-problems. DP helps in significantly minimizing the computation time.

7.1 Policy Iteration

We can now find the optimal policy

Algorithm 1: Policy Iteration

```
1. Initialisation
$V(s) \in \mathbb{R}$, (e.g $V(s) = 0$) and $\pi(s) \in A$ for all $s \in S$,
$\Delta \leftarrow 0$
2. Policy Evaluation
while $\Delta \geq \theta$ (a small positive number) do
    foreach $s \in S$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end
end
3. Policy Improvement
policy-stable $\leftarrow$ true
foreach $s \in S$ do
    old-action $\leftarrow \pi(s)$
    $\pi(s) \leftarrow_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s')\right]$
    policy-stable $\leftarrow$ old-action $= \pi(s)$
end
if policy-stable return V $\approx V_*$ and $\pi \approx \pi_*$, else go to
  2
```

7.2 Value Iteration

We can avoid to wait until $V(s)$ has converged and instead do policy improvement and truncated policy evaluation step in one operation

Algorithm 2: Value Iteration

```
Initialise $V(s) \in \mathbb{R}$, e.g$V(s) = 0$
$\Delta \leftarrow 0$
while $\Delta \geq \theta$ (a small positive number) do
    foreach $s \in S$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end
end
ouput: Deterministic policy $\pi \approx \pi_*$ such that
$\pi(s) =_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s')\right]$
```

8 Bandits algorithms

Bandit models first appeared in the 1930s to model gambling. The term bandit comes from gambling where slot machine can be thought as one-armed bandits. In the so-called one-armed bandit problem, a user faces $K \geq 2$ slot machines. Each giving an average reward that the user does not know a priori. For each of these actions, he will select a machine that will maximize his winnings.

Problem: which slot machine should we play at each turn when their payoffs are not necessarily the same and initially unknown?

This kind of problem is far from being unique to gambling. It is present in many situations where we have to choose how to allocate resources sequentially among several options without knowing the outcomes of those options. Below is a list of the most frequent applications:

- **Design of experiments**:this is particularly the case for *clinical tests*, where the effectiveness of treatments is not known and the only way to find out is to test them on real patients.
- **Website Optimization**: online ad placement, web page personalization [2].
- **Cognitive radio**: is a radio that can be programmed and configured dynamically to use the best wireless channels [1]
- **Network Routing**: Routing is the mechanism by which paths are selected in a network to route data from a sender to one or more recipients. Allocation of channels to the right users, such that the overall throughput is maximised, can be formulated as a Multi-Armed Bandit Problem (MABP).
- **RecommenderSystems**

9 Temporal Difference: Q Learning

Temporal Difference (TD) methods learn directly from raw experience without a model of the environment's dynamics. TD substitutes the expected discounted reward G_t from the episode with an estimation:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{11}$$

The following algorithm gives a generic implementation.

Algorithm 3: Q Learning

```
Initialise $Q(s, a)$ arbitrarily and
 $Q(terminal - state, ) = 0$
foreach episode $\in$ episodes do
    while s is not terminal do
        Choose $a$ from $s$ using policy derived from $Q$
          (e.g., $\epsilon$-greedy)
        Take action $a$, observer $r, s'$
        $Q(s, a) \leftarrow$
          $Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
        $s \leftarrow s'$
    end
end
```

10 Deep Q Learning

Created by *DeepMind*, Deep Q Learning, DQL, substitutes the Q function with a deep neural network called *Q-network*. It also keep track of some observation in a *memory* in order to use them to train the network.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \left[\left(\underbrace{r + \gamma \max_a Q(s', a'; \theta_{i-1})}_{\text{target}} - \underbrace{Q(s, a; \theta_i)}_{\text{prediction}} \right)^2 \right] \tag{12}$$

Where θ are the weights of the network and $U(D)$ is the experience replay history.

Algorithm 4: Deep Q Learning

```
Initialise replay memory $D$ with capacity $N$
Initialise $Q(s, a)$ arbitrarily
foreach $episode \in episodes$ do
    while $s$ is not terminal do
        With probability $\epsilon$ select a random action
          $a \in A(s)$
        otherwise select $a = \max_a Q(s, a; \theta)$
        Take action $a$, observer $r, s'$
        Store transition $(s, a, r, s')$ in $D$
        Sample random minibatch of transitions
          $(s_j, a_j, r_j, s'_j)$ from $D$
        Set $y_j \leftarrow$
          $\begin{cases} r_j & \text{for terminal } s'_j \\ r_j + \gamma \max_a Q(s', a'; \theta) & \text{for non-terminal } s'_j \end{cases}$
        Perform gradient descent step on
          $(y_j - Q(s_j, a_j; \Theta))^2$
        $s \leftarrow s'$
    end
end
```

11 Monte Carlo Methods

Monte Carlo (MC) is a *Model Free* method, It does not require complete knowledge of the environment. It is based on **averaging sample returns** for each state-action pair. The following algorithm gives the basic implementation

Algorithm 5: Monte Carlo first-visit

```
Initialise for all $s \in S, a \in A(s)$ :
  $Q(s, a) \leftarrow$ arbitrary
  $\pi(s) \leftarrow$ arbitrary
  $Returns(s, a) \leftarrow$ empty list
while forever do
    Choose $S_0 \in S$ and $A_0 \in A(S_0)$, all pairs have
      probability $> 0$
    Generate an episode starting at $S_0, A_0$ following
      $\pi$ foreach pair $s, a$ appearing in the episode do
        $G \leftarrow$ return following the first occurrence of
          $s, a$
        Append $G$ to $Returns(s, a))$
        $Q(s, a) \leftarrow average(Returns(s, a))$
    end
    foreach $s$ in the episode do
        $\pi(s) \leftarrow_a Q(s, a)$
    end
end
```

For non-stationary problems, the Monte Carlo estimate for, e.g, V is:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right] \tag{13}$$

Where α is the learning rate, how much we want to forget about past experiences.

References

[1] Wassim Jouini et al. "Multi-armed bandit based policies for cognitive radio's decision making issues". In: *3rd international conference on Signals, Circuits and Systems (SCS)*. Djerba, Tunisia, Nov. 2009, 6 pages. URL: https://hal-supelec.archives-ouvertes.fr/hal-00421252.

[2] J.M. White. *Bandit Algorithms for Website Optimization: Developing, Deploying, and Debugging*. O'Reilly Media, 2012. ISBN: 9781449341589. URL: https://books.google.fr/books?id=xnAZLjqGybwC.