Deep Learning

A course about theory & practice

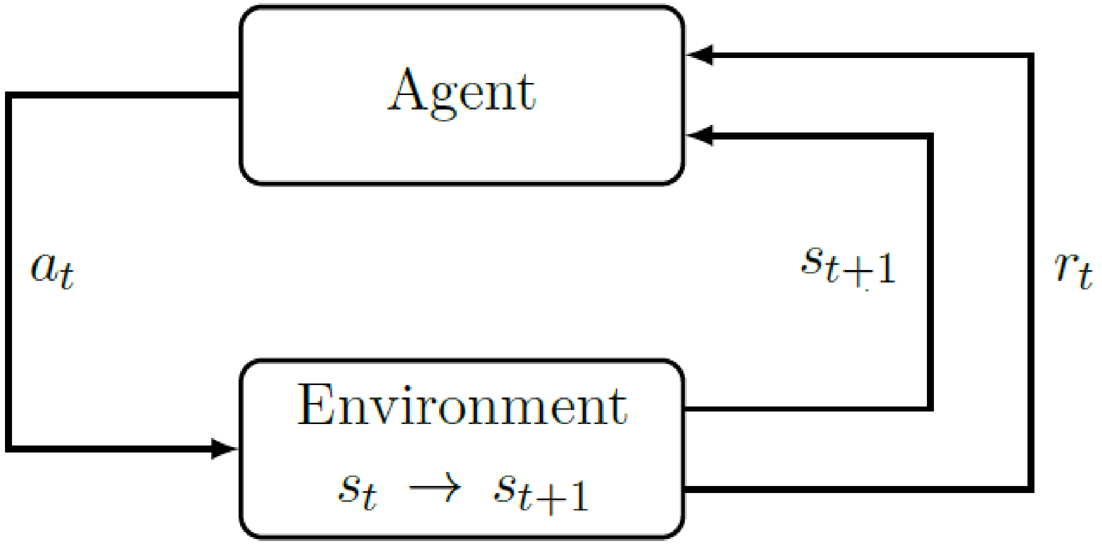
Reinforcement Learning

Marco Piastra

Markov Decision Process (MDP)

Basic assumptions

[image from: https://arxiv.org/pdf/1811.12560.pdf]



The **Environment**: is in *state* s_t — *time*

An **Agent** observes *state* s_t and performs *action* a_t

The **Environment** *state* transitions from $s_t \rightarrow s_{t+1}$

The **Agent** receives *reward* r_t

Cumulative reward: $R := \sum_{t=0}^{\infty} r_t$

Markov Decision Process (MDP)

Markov Decision Process: $< \mathcal{S}, \mathcal{A}, r, P, \gamma >$

A set of *states* : $\mathcal{S} = \{s_1, s_2, \dots\}$

A set of *actions* : $\mathcal{A} = \{a_1, a_2, \dots\}$

A *reward function* : $r : \mathcal{S} \to \mathbb{R}$

A *transition probability distribution* : $P(S_{t+1} \mid S_t, A_t)$ (also called a *model*)

Markov property: the transition probability depends only on the previous state and action

$$P(S_{t+1} \mid S_t, A_t) = P(S_{t+1} \mid S_t, A_t, S_{t-1}, A_{t-1}, S_{t-2}, A_{t-2}, \dots)$$

A *discount factor* : $0 \leq \gamma < 1$

Markov Decision Process (MDP): policies and values

The agent is supposed to adopt a *deterministic* policy : $\pi : \mathcal{S} \to \mathcal{A}$

In other words, the agent always chooses its *action* depending on the *state* alone

Given a policy π , the ***state value function*** is defined, for each state s as:

$$V^\pi(s) := \mathbb{E}[r(S_t) + \gamma r(S_{t+1}) + \gamma^2 r(S_{t+2}) + \ldots \mid \pi, S_t = s]$$

Note the role of the *discount factor*: a value $\gamma < 1$ means that that future rewards could be weighted less (by the agent) than immediate ones

Note also that all states S_t must be described by *random variables* :
i.e. the policy is deterministic, yet the state transition is not

Note also that when the reward is *bounded*, i.e. $r(S) \leq r_{\max}$

$$\sum_{t=0}^{\infty} \gamma^t \, r(S_t) \ \leq \ r_{\max} \sum_{t=0}^{\infty} \gamma^t \ = \ r_{\max} \frac{1}{1-\gamma}$$

for $\gamma < 1$ this is the *geometric series*

Bellman equations

By working on the definition of value function:

$$V^\pi(s) := \mathbb{E}[r(S_t) + \gamma r(S_{t+1}) + \gamma^2 r(S_{t+2}) + \ldots \mid \pi, S_t = s]$$
$$= \mathbb{E}[r(S_t) + \gamma(r(S_{t+1}) + \gamma r(S_{t+2}) + \ldots) \mid \pi, S_t = s]$$
$$= r(s) + \gamma \mathbb{E}[r(S_{t+1}) + \gamma r(S_{t+2}) + \ldots \mid \pi, S_t = s]$$
$$= r(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) \cdot \mathbb{E}[r(S_{t+1}) + \gamma r(S_{t+2}) + \ldots \mid \pi, S_{t+1} = s']$$
$$= r(s) + \gamma \sum_{S_{t+1}} P(S_{t+1} \mid s, \pi(s)) \cdot V^\pi(S_{t+1})$$

This means that in a Markov Decision Process:

$$V^\pi(s) = r(s) + \gamma \sum_{S_{t+1}} P(S_{t+1} \mid s, \pi(s)) \cdot V^\pi(S_{t+1})$$

This is true for any *state*, so there is one such equation for each of those

If the set of states is finite, there are exactly $|S|$ (linear) Bellman equations for $|S|$ variables:
in general, for any deterministic policy, V^π can be computed analytically

Optimal policy – Optimal value function

- Basic definitions

$$V^*(s) := \max_{\pi} V^{\pi}(s),\ \forall s \in S$$
$$\pi^*(s) := \mathrm{argmax}_{\pi} V^{\pi}(s),\ \forall s \in S$$

Property: for every MDP, there exists such an optimal deterministic policy (*possibly non-unique*)

With Bellman Equations:

$$\max_{\pi} V^{\pi}(s) = r(s) + \gamma \max_{\pi} \left(\sum_{S_{t+1}} P(S_{t+1} \mid s, \pi(s)) \cdot V^{\pi}(S_{t+1}) \right)$$
$$V^*(s) = r(s) + \gamma \max_{\pi} \left(\sum_{S_{t+1}} P(S_{t+1} \mid s, \pi(s)) \cdot V^*(S_{t+1}) \right)$$
$$= r(s) + \gamma \max_{a} \left(\sum_{S_{t+1}} P(S_{t+1} \mid s, a) \cdot V^*(S_{t+1}) \right)$$

Therefore:

$$\pi^*(s) = \mathrm{argmax}_{a} \left(\sum_{S_{t+1}} P(S_{t+1} \mid s, a) V^*(S_{t+1}) \right)$$

once V^* *has been determined,* π^* *can be determined as well*

Computing V^* *directly from these equations is unfeasible, however*

There are in fact $|\mathcal{A}|^{|\mathcal{S}|}$ *possible strategies ...*

Reinforcement Learning
(model-based)

Optimal value function: value iteration

- Value iteration algorithm

  Initialize: $V(s) := r(s), \ \forall s \in S$
  Repeat:

  Note that there is no policy: all actions must be explored

  1) For every state, update: $V(s) := r(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a) V(s')$

  Theorem: for every fair way (*i.e. giving an equal chance*) of visiting the states in S, this algorithm converges to V^*

Optimal policy: policy iteration

- Policy iteration algorithm

Initialize $\pi(s), \forall s \in S$ at random
Repeat:

This step is computationally expensive: either solve the equations or use value iteration (with fixed policy π)

1) For each state, compute: $V(s) := V^{\pi}(s)$
2) For each state, define: $\pi(s) := \mathrm{argmax}_a \sum_{s'} P(s' \mid s, a) V(s')$

Theorem: for every fair way (*i.e. giving an equal chance*) of visiting the states in S, this algorithm converges to π^*

As with the value iteration algorithm, this algorithm uses partial estimates to compute new estimates.
It is also greedy, in the sense that it exploits its current estimate $V^{\pi}(s)$

Policy iteration converges with very few number of iterations, but every iteration takes much longer time than that of *value iteration*

The tradeoff with value iteration is the action space:
when action space is large and state space is small, policy iteration could be better

Reinforcement Learning
(model-free)

Molde-based vs. model-free reinforcement learning

- *Value iteration* and *policy iteration* are offline algorithms

  The *model* , i.e. the Markov Decision Process is known

  What needs to be learnt is the optimal policy π^*

  In the algorithms, *visiting states* just means considering them:
  there needs not be an agent which *actually plays* the game

- Different conditions: *learning by doing …*

  Suppose the *model* (i.e. the MDP) is NOT known, or perhaps known only in part

  In particular, it might not be known the transition function $P(S_{t+1} \mid S_t, A_t)$

  Such scenario is also called 'model-free'

  The agent, then, must learn by doing… that is, actually playing the game

Action value function

An analogous of the value function V^π

Given a policy π , the ***action value function*** is defined, for each pair (s, a) as:

$$Q^\pi(s,a) := \sum_{S_{t+1}} P(S_{t+1} \mid s,a) \cdot V^\pi(S_{t+1})$$
$$= \sum_{S_{t+1}} P(S_{t+1} \mid s,a) \cdot \mathbb{E}[r(S_{t+1}) + \gamma r(S_{t+2}) + \ldots \mid \pi, S_{t+1}]$$
$$= \sum_{S_{t+1}} P(S_{t+1} \mid s,a) \cdot [r(S_{t+1}) + \mathbb{E}[\gamma r(S_{t+2}) + \ldots \mid \pi, S_{t+1}]]$$
$$= \sum_{S_{t+1}} P(S_{t+1} \mid s,a) \cdot [r(S_{t+1}) + \gamma Q^\pi(S_{t+1}, \pi(S_{t+1}))]$$

In other words, $Q^\pi(s,a)$ is the expected value of the reward in S_{t+1}
by taking action a in state s and then following policy π from that point on

Following a similar line of reasoning, the ***optimal*** *action value function* is

$$Q^*(s,a) = \sum_{S_{t+1}} P(S_{t+1} \mid s,a) \cdot [r(S_{t+1}) + \gamma \max_{a'} Q^*(S_{t+1}, a')]$$

Q-Learning

- Q-learning algorithm (ε-*greedy version*)

  Initialize $\hat{Q}(s,a)$ at random, put the agent is in a random state s
  Repeat:

  1) Select the action $\text{argmax}_a \hat{Q}(s,a)$ with probability $(1-\varepsilon)$
     otherwise, select a at random
  2) The agent is now in state s' and has received the reward r
  3) Update $\hat{Q}(s,a)$ by

$$\Delta\hat{Q}(s,a) = \alpha[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)]$$

Exponential Moving Average
(see later …)

Note that step 1) is closely similar to a ***multi-armed bandit****:*
in each state, the agent has to choose one among all actions in $\mathcal{A}$
and this will produce a random reward…

Q-Learning

- Q-learning algorithm

  Theorem (Watkins, 1989): in the limit of that each action is played infinitely often and each state is visited infinitely often and $\alpha \to 0$ as experience progresses, then

$$\hat{Q}(s, a) \to Q^*(s, a)$$

  with probability 1

The Q-learning algorithm bypasses the MDP entirely,
in the sense that the optimal strategy is learnt without learning the model $P(S_{t+1} \mid S_t, A_t)$

Q-Learning revisited

- Q-learning algorithm (*ε-greedy version*)

Initialize $\hat{Q}(s,a)$ at random, put the agent is in a random state s
Repeat:

1) Select the action $a = \text{argmax}_a \hat{Q}(s,a)$ with probability $(1-\varepsilon)$
   otherwise, select a at random
2) The agent is now in state s' and has received the reward r
3) Update $\hat{Q}(s,a)$ by

$$\Delta\hat{Q}(s,a) = \alpha[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)]$$

By rewriting step 3)

$$\hat{Q}(s,a) = \hat{Q}(s,a) + \Delta\hat{Q}(s,a) = \hat{Q}(s,a) + \alpha[r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)]$$
$$= \underbrace{\alpha[r + \gamma \max_{a'} \hat{Q}(s',a')] + (1-\alpha)\hat{Q}(s,a)}_{\text{Exponential Moving Average}}$$

Exponential Moving Average

compare with (see before):

$$Q^*(s,a) = \underbrace{\sum_{S_{t+1}} P(S_{t+1} \mid s,a) \cdot [r(S_{t+1}) + \gamma \max_{a'} Q^*(S_{t+1},a')]}_{\text{Expectation}}$$

Expectation