Machine Learning
ML. 4 Intro to reinforcement learning

David Ríos Insua and Roi Naveiro

Objectives and schedule

Introduce key concepts about reinforcement learning. Markov decision processes, dynamic programming, Q-learning, Deep RL

Contents

- Motivation.
- Concepts. MDPs and dynamic programming
- Q-learning
- Deep reinforcement learning
- Lab

Refs

Sutton, Barto (2018) RL: An intro
https://www.csee.umbc.edu/courses/graduate/678/spring17/RL-3.pdf

Brilliant summary in

- https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html

Videos

- https://www.youtube.com/watch?v=V1eYniJ0Rnk DeepMind DRL. Mnih et al
- https://www.youtube.com/watch?v=WXuK6gekU1Y Alphago
- https://www.youtube.com/watch?v=tCpf5wDr0UE AlphaZero

General AI

- https://bdtechtalks.com/2021/06/07/deepmind-artificial-intelligence-reward-maximization/

Motivation

RL: features

- Learning by interaction with environment
  - ‘Cause-Effect’ relations
  - Consequences of actions
  - What to do to achieve goals.
- Goal directed learning: what to do to maximize a reward
  - Discover actions that yield most reward by trying them (trial and error search)
  - Actions affect not only immediate reward but also affect environment (delayed reward)

RL features

- Optimal control of incompletely known Markov decision processes
  - Schemes for sense-act-respond
  - Exploration (collect more info)-exploitation (best action)
  - Uncertainty about evolution of environment and rewards achieved
  - Sequential learning

Examples

Example	+ Reward	- Reward
Defeat Go world champion	Winning a game	Losing
Make humanoid robot walk	Moving forward	Falling
Drive ADS	Reaching the target	Crashing
Dominating pension fund market *	Market share increases	Market share decreases
Virus mutations*	Mutation succeeds	Does not succeed

A multi-armed bandit example. Statement

10 slot machines at casino. Play for free.

Give a random reward between 0 and 10 €. Each different average payout.

Figure out the best machine in this *10-armed bandit problem* (to make the most money)

A multi-armed bandit example. Ingredients

Actions. a. Pulling one out of 10 arms.

Plays. At each time instant, make an action

Reward. R after action, receives a reward between 0 and 10. Each arm has a unique probability distribution over rewards

Policy. Play a few times, choosing different bandits, observe rewards. Then, choose bandit with largest observed average reward

Expected reward at play *k* for action *a*. $Q_k(a)$. *Value function, action-value function, Q function.*

A multi-armed bandit example. Ingredients

Exploration. At start, play game and observe rewards from machines.

Exploitation. Use knowledge about which machine produces largest expected reward to keep playing on it

Need proper *balance* between exploration and exploitation

For exploitation: maximise the Q value. *Greedy strategy*

Epsilon greedy strategy. With probability epsilon, choose action at random; with 1-epsilon choose best bandit so far

Update the Q values. Maintained as an array

A multi-armed bandit example. Algo (Q-learning)

Choose epsilon, initialise Q values at 0

Until terminating condition (no. trials, stability of Q-values,…)

    If rand <eps Choose bandit i randomly

        else Choose bandit i maximising Q

    Play bandit i

    Update Q value (of bandit i)

Play on bandit maximising Q

A multi-armed bandit example. With *softmax*

Rather than choosing at random (uniformly) use probabilities proportional to

$$exp(\, Q_k\,(a) \,/\, \tau \,)$$

where τ is a 'temperature' scaling the probability

A *contextual* multi-armed bandit example

Consider advertisement placement. Whenever you visit a website with ads, ads maximizing probability that you will click on them will pop up

Rewards depend on *states* in a *state space*. State = info available in environment useful to make decision.

State-action pair

Q values Q(s,a) over state-action pairs. Same strategy… but bigger table…

Uncertainty over state evolution. Similar strategy based on expected values (and learning about evolution probabilities)… but bigger

…..

Deep reinforcement learning. Using DNNs to approximate Q values (+ some twists)

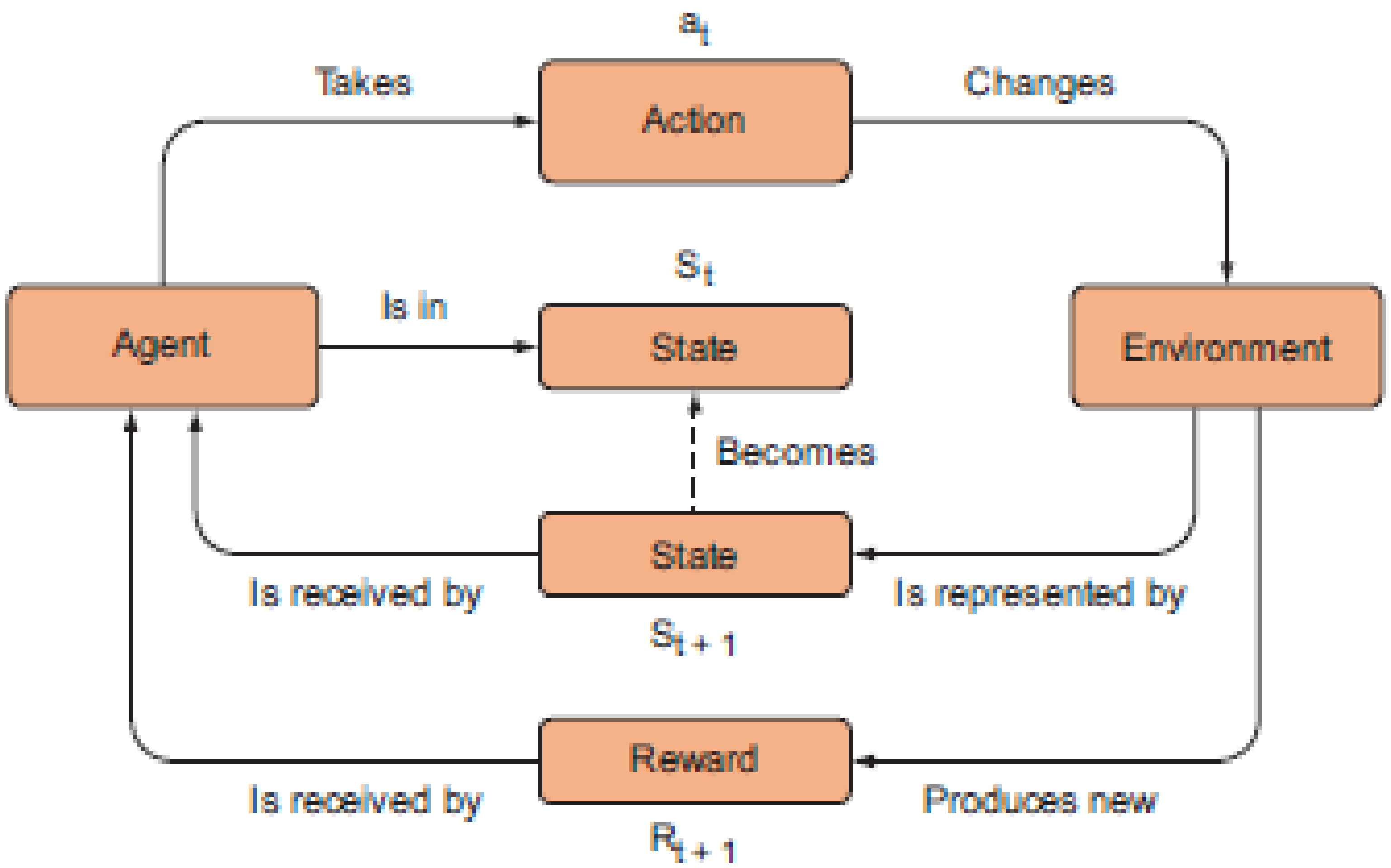
RL elements

RL elements

- Agent
- Environment with states
- Policy
- Reward signal
- Value function
- Model of environment

Goal: Learn a good policy for the agent from experiment trials and simple feedback received; with optimal policy, agent capable to actively adapt to environment to maximize future rewards.

RL: the broad picture



RL Element: MDPs

- States $s \in \mathcal{S}$
- Actions $a \in \mathcal{A}$
- Reward $r \in \mathcal{R}$
- History $S_1, A_1, R_2, S_2, A_2, \ldots, S_T$

RL elements: MDPs

Transition

(s, a, s', r).

$$P(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]$$

$$P^a_{ss'} = P(s'|s, a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} P(s', r|s, a)$$

$$R(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r|s, a)$$

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]$$

Policy

Deterministic

$$\pi(s) = a$$

Stochastic

$$\pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$$

RL elements. Value functions

Value function. Discounted future reward or return

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

State value of a state

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

Action value of state-action pair (Q value)

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

Relation State value vs Action value

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

Advantage function

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

RL elements: Optimal value and polciy

Optimal value function

$$V_*(s) = \max_\pi V_\pi(s), Q_*(s,a) = \max_\pi Q_\pi(s,a)$$

Optimal policy

$$\pi_* = \arg\max_\pi V_\pi(s), \pi_* = \arg\max_\pi Q_\pi(s,a)$$

With, obviously,

$$V_{\pi_*}(s) = V_*(s) \text{ and } Q_{\pi_*}(s,a) = Q_*(s,a)$$

Bellman equations

Decomposing value function into immediate reward and future value

$$\begin{aligned}
V(s) &= \mathbb{E}[G_t | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \dots) | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]
\end{aligned}$$

$$\begin{aligned}
Q(s, a) &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a \sim \pi} Q(S_{t+1}, a) \mid S_t = s, A_t = a]
\end{aligned}$$

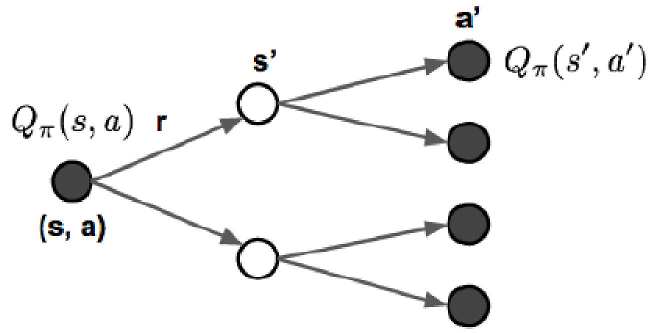
Bellman expectation equations

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

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_\pi(s')$$

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_\pi(s') \right)$$

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s',a')$$



Bellman optimality equations

$$V_*(s) = \max_{a \in \mathcal{A}} Q_*(s, a)$$

$$Q_*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_*(s')$$

$$V_*(s) = \max_{a \in \mathcal{A}} \big(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_*(s') \big)$$

$$Q_*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} \max_{a' \in \mathcal{A}} Q_*(s', a')$$

If complete info available (R and P), dynamic programming

If not, can’t apply these, but guides solution!!!

Dynamic programming

Model fully known. Apply Bellman equations

Policy evaluation for a given policy

$$V_{t+1}(s) = \mathbb{E}_\pi[r + \gamma V_t(s') | S_t = s] = \sum_a \pi(a|s) \sum_{s',r} P(s', r|s, a)(r + \gamma V_t(s'))$$

Policy improvement

$$Q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a] = \sum_{s',r} P(s', r|s, a)(r + \gamma V_\pi(s')) \qquad \pi'(s) = \arg\max_{a\in\mathcal{A}} Q_\pi(s,a)$$

Policy iteration

$$\pi_0 \xrightarrow{\text{evaluation}} V_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{\text{evaluation}} V_{\pi_1} \xrightarrow{\text{improve}} \pi_2 \xrightarrow{\text{evaluation}} \cdots \xrightarrow{\text{improve}} \pi_* \xrightarrow{\text{evaluation}} V_*$$

$$\begin{aligned} Q_\pi(s, \pi'(s)) &= Q_\pi(s, \arg\max_{a\in\mathcal{A}} Q_\pi(s,a)) \\ &= \max_{a\in\mathcal{A}} Q_\pi(s,a) \geq Q_\pi(s, \pi(s)) = V_\pi(s) \end{aligned}$$

Q-learning

RL elements: Approaches

When complete info not available

- Model-based. Model (of environment) is learnt
- Model-free. No model (of environment) is learnt
- On-policy. Use outcomes form target policy to train
- Off-policy. Evaluate or improve policy different from that used to generate

Q-learning. Off-policy TD learning

1. Initialize $t = 0$.
2. Starts with S_0.
3. At time step t, we pick the action according to Q values, $A_t = \arg\max_{a \in \mathcal{A}} Q(S_t, a)$ and ϵ-greedy is commonly applied.
4. After applying action A_t, we observe reward R_{t+1} and get into the next state S_{t+1}.
5. Update the Q-value function:
   $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$.
6. $t = t + 1$ and repeat from step 3.

Watkins (1992) proves that converges to optimal Q, from which a policy is deduced

Deep reinforcement learning

Motivation

- Q(s,a) is a table that is updated each time
- But what if A, S are large? Even continuous?
- Approximate Q(s,a) with a model Q(s,a,θ) in particular a neural net

DQN

DQN adds two features to improve convergence

- *Experience replay*. Store episodes $e_t = (S_t, A_t, R_t, S_{t+1})$ in replay memory $D_t = \{e_1, \ldots, e_t\}$ During Q-L updates randomly sample from memory (data efficiency, reduce correlation, smooths)
- *Periodically updated target*. Q network frozen every C iterations. Stabilizes training.

Loss is

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

DQN

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Further topics

Policy gradient methods

Before: Learn value function and deduce policy accordingly

PG methods learn policy directly with a parametrized function

$\pi(a|s;\theta)$

Use objective function $\mathcal{J}(\theta) = V_{\pi_\theta}(S_1) = \mathbb{E}_{\pi_\theta}[V_1]$

Use gradient ascent

Multi-agent reinforcement learning

Several agents. Competitive marketing, Cybersecurity, Virus-human competition

- Common knowledge. Nash Q-learning,…
- No common knowledge. Adversarial modelling, adversarial machine learning

Keep in touch!!

david.rios@icmat.es
@davidrinsua

roi.naveiro@icmat.es