Deep Reinforcement Learning

Course: Deep Learning

Prof. Edgar F. Roman-Rangel.

Instituto Tecnologico Autonomo de Mexico, ITAM.
Semester: Spring 2020.

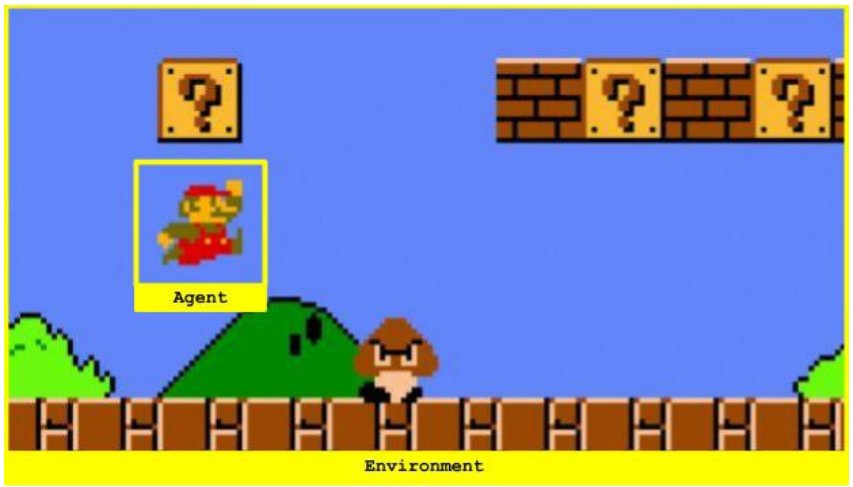
May 7^{th}, 2020.

Outline

Definition

Reinforcement Learning, RL:

Type of Machine Learning in which, for a given **agent**, interacting with an **environment**, we wish it to **learn** to optimize its **behavior**, so it can maximize some **reward**.

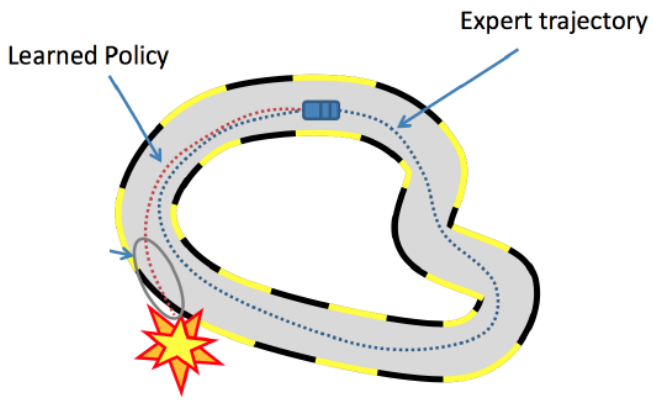


We might be interested in maximizing a total cumulative reward, and not only the immediate reward.

Constraints

We do not tell the agent how to behave, but only provide it with rewards as the response of executing specific actions, constrained by the **state** of the environment.

RL, *is not* learning the solving style of a human agent (imitation learning), but learning a **policy** to maximize a reward.



Notation

The notion of training pairs (x, y) is no longer evident. Now we have tuples (s_t, a_t, r_t, s_{t+1}), where,

- s_t: represents the state of the environment at time t.
- a_t: represents the action executed by the agent at time t.
- r_t: represents the reward returned by the environment as a response to the pair (s_t, a_t).

Environment examples

State-action-reward examples:

- https://gym.openai.com/
- https://github.com/openai/gym/wiki/Environments

Objective

Given the environment $e : s_t \times a_t \rightarrow r_t$,

Goal: find a policy π^* that maximizes $\sum_t \gamma^t r_t$,

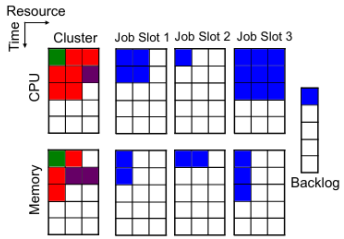
where, γ is a discount factor $(0 < \gamma < 1)$, which makes recent rewards be more relevant than older rewards.

$$\pi : s_t \rightarrow a_t.$$

Quality of π^* might be evaluated per **episode**.

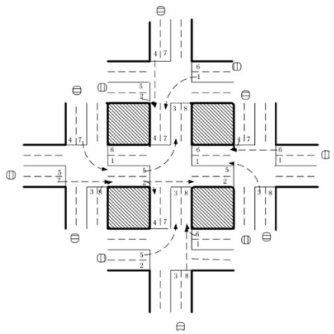
Episodes are defined by trajectories: $(s_0, a_0, r_1, s_1, a_1, r_2, \ldots)$.
$\rightarrow$ RL is often modeled as a Markov Decision Process (MDP).

Applications

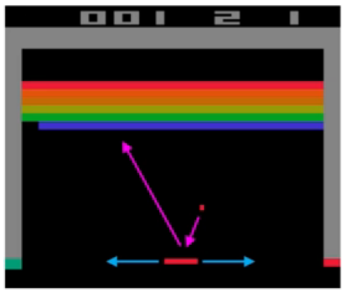


Resource Management

Robotics



Traffic light control



Game playing

See:

- https://www.youtube.com/watch?v=gn4nRCC9TwQ
- https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-architectures-4c1b3fb7f756

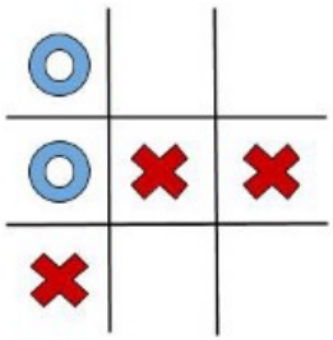
Outline

Markov Decision Process (MDP)

We assume all states are *Markov States*:

> "Any state depends solely on the state that came before it, and the transition probability from that state to the current one".

Example:



To choose our next move (next state), we don't care how we got to this current state, all we care about is the current state itself and the probability of choosing the next state.

Expected return

Sum of future discounted rewards (from time t).

$$R_t = \sum_{k=t}^{T} \gamma^k r_k,$$

where T is the final time step.

For the initial time step (s_0, a_0), we can hope,

$$\begin{aligned} R_0 &= \sum_{k=0}^{T} \gamma^k r_k, \\ &= r_0 + \gamma R_1. \end{aligned}$$

For the final time step (s_T, a_T),

$$R_T = r_T.$$

Value functions (1/2)

Value functions are function of states or (state, action) pairs. There are two types.

State-value function (v_π)

Value of a state under π.

$$\begin{aligned} v_\pi(s_t) &= \mathbb{E}\left[R_t | s_t\right] \\ &= \mathbb{E}\left[\sum_{k=t}^{T} \gamma^k r_k | s_t\right]. \end{aligned}$$

How good it is for the agent to be at given state.

Value functions (2/2)

Q-function and Q-value.

Action-value function (q_π)

Value of an action, given a state, under π.

$$q_\pi(s_t, a_t) = \mathbb{E}\left[R_t | s_t, a_t\right]$$
$$= \mathbb{E}\left[\sum_{k=t}^{T} \gamma^k r_k | s_t, a_t\right].$$

How good it is for the agent to perform a given action while been at a given state.

Optimal policy

$$\pi^* = \arg\max_{\pi} \{v_\pi(s)\}.$$

Optimal state-value function

$$v^*(s) = \max_{\pi} v_\pi(s).$$

Optimal action-value function

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

Bellman equation

Optimality for q^*.

$$q^*(s_t, a_t) = \mathbb{E}\left[r_t + \gamma \max_a q^*(s_{t+1}, a_{t+1})\right],$$

Our holy grail!

Example

current state: $s_t = 2$.
best action: $a_t = 3$.
$\gamma = 0.9$

next state: $s_{t+1} = 4$.
(returned by the environment).
next best action: $a_{t+1} = 5$.

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

Matrix R or rewards.

$$q^*(s_t, a_t) = r_t + \gamma \max_a q^*(s_{t+1}, a_{t+1})$$

$$q^*(2,3) = 0 + 0.9 \max_a q^*(4, a)$$

$$q^*(2,3) = 0 + 0.9(100)$$

$$q^*(2,3) = 90$$

Outline

Q-table

- Memory of what the agent has learned through experience.
- Transition probabilities.

	0	1	2	3	4	5
0	0	0	0	0	80	0
1	0	0	0	64	0	100
2	0	0	0	64	0	0
3	0	80	51	0	80	0
4	64	0	0	64	0	100
5	0	80	0	0	80	100

Matrix Q[1].

- Initialize to zeros.
- Estimate new q-values by trial and error.
- Update per episode.
- Find a good trade-off between exploitation and exploration.

[1] http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Update Q-table

Make q-values as close as possible to the right-hand side of Bellman equation, so the q-values eventually converge to q^*.

$$loss = \mathbb{E}\left[r_t + \gamma \max_a q^*(s_{t+1}, a)\right] - \mathbb{E}\left[\sum_t \gamma^t r_t\right].$$

Actual update:

$$q(s, a) = (1 - \alpha)q(s, a) + \alpha(r_t + \gamma \max_a q(s', a'))$$

Once q converges, we get the optimal q-function (optimal policy).

Training and testing

Learning:

Run several episodes updating q-values.

Testing:

Freeze q-values and run several episodes.

Exploration vs exploitation

Choosing new actions for training, can be done at random (exploration) or by using current knowledge (exploitation).

ϵ-greedy approach

Only explore the environment with probability of ϵ.

Initially $\epsilon \approx 1$, and decrease it as more knowledge is acquired.

Outline

Motivation

State space might be too large to fit it in a q-table or run simulations.
e.g., states are given by images: frames of 64×64 pixels, with values between 0 and 255 $\rightarrowtail$ number of states per frame $= (64 \times 64)^{256}$.

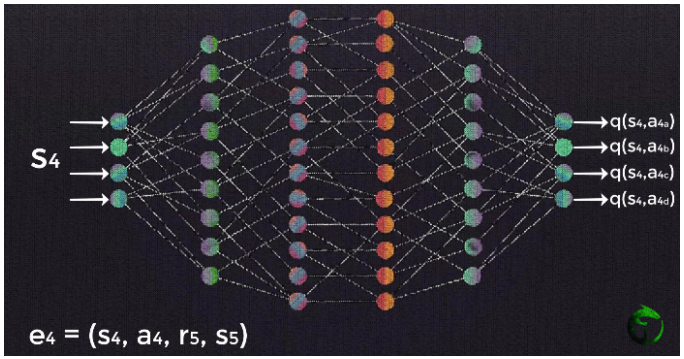
Motivation

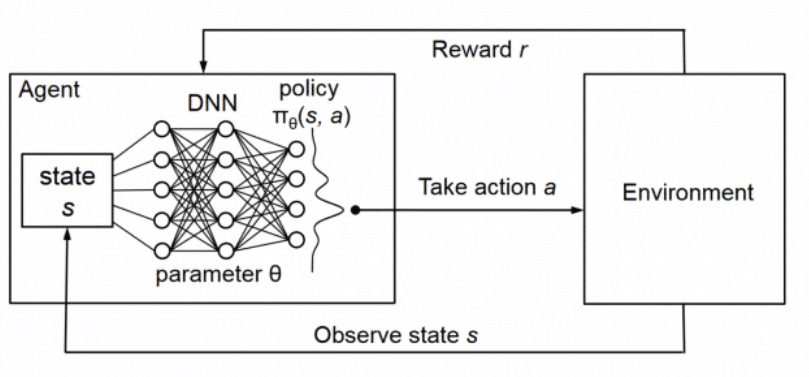
Multi-agent scenario.

Deep RL

Enter Deep Learning:
universal approximation function – Learn to approximate q-tables.
(Agent: Deep Neural Network)

Terms: Deep Q Learning (DQL) and Deep Q Network (DQN).





Let's take a second to think about input and output shapes.

Deep Q-Learning

We no longer have a Q-table, we do not need it. Instead, we pass s through a network to predict $q(s, a)$.

Policy learning vs., value learning.

Update (gradient descent)

loss $= q^*(s, a) - q(s, a)$

But we don't really know $q^*(s, a)$.

Alternating, we do both: train the network to predict $q(s, a)$ and learn $q(s, a)$ itself.

Experience replay

- Store agent experiences at each time step in a **replay memory**, $e_t = (s_t, a_t, r_t, s_{t+1})$.
- Randomly sample this replay memory to train the agent (network).
- Agent will get s_t to produce $q(s_t, a_t) \rightarrow a_t$.
- Feed a_t to the system to generate s_{t+1}.
- Agent will get s_{t+1} to produce $q(s_{t+1}, a_{t+1})$.
- Retrain agent to predict $q(s_{t+1}, a_{t+1})$ directly from s_t.

Stability

Since we do not really know the optimal q-values, and the agent 'brain' gets updated every time step, there is the risk of falling into stability issues.

Target network

To address this potential problem, we rely on a **target network**, which is used to predict $q(s_{t+1}, a_{t+1})$, and only update every N time steps.

Code

Code.

A few more thoughts

- DRL agents learn to perform tasks y trial and error.
- No human intervention, other than defining rewards.
- Shaping a reward function is critical (avoid cobra effect).
- Expected reward might be a long-term concept.
- RL seems to be closer to a human-like approach than supervised learning.
- RL is appropriate for tasks that are simple to evaluate but difficult to specify.
- Training requires simulation environments.
- RL is still far from reality in many scenarios (self-driving cars).
- Training might require huge computational loads.

Q&A

Thank you!

References

- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing Atari with Deep Reinforcement Learning". NIPS, 2013.
- Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, David Meger. "Deep Reinforcement Learning that Matters". AAAI, 2018.
- Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". AAAI, 2016.
- Charu C. Aggarwal. "Neural Networks and Deep Learning". Springer. 2018. Chap. 9.

References

- Maxim Lapan. "Deep Reinforcement Learning Hands-On". 2nd Ed. Packt Publishing. 2020.
- Alexander Zai and Brandon Brown. "Deep Reinforcement Learning in Action". Manning publications. 2020.
- H. Mao, A. Menache, I. Menache, and S. Kandula. "Resource Management With deep Reinforcement Learning". In ACM Workshop on Hot Topics in Networks, 2016.
- I. Arel, C. Liu, T. Urbanik, and A. Kohls, "Reinforcement learning-basedmulti-agent system for network traffic signal control". IET Intelligent Transport Systems, 2010.
- J. Kober, J. A. D. Bagnell, J. Peters. "Reinforcement Learning in Robotics: A survey". Int. J. Robot. Res. Jul. 2013.