Reinforcement Learning

Introduction

Boreiko Valentyn

Deep Learning with Tensorflow

January 18, 2018

Reinforcement Learning(RL) - Motivation

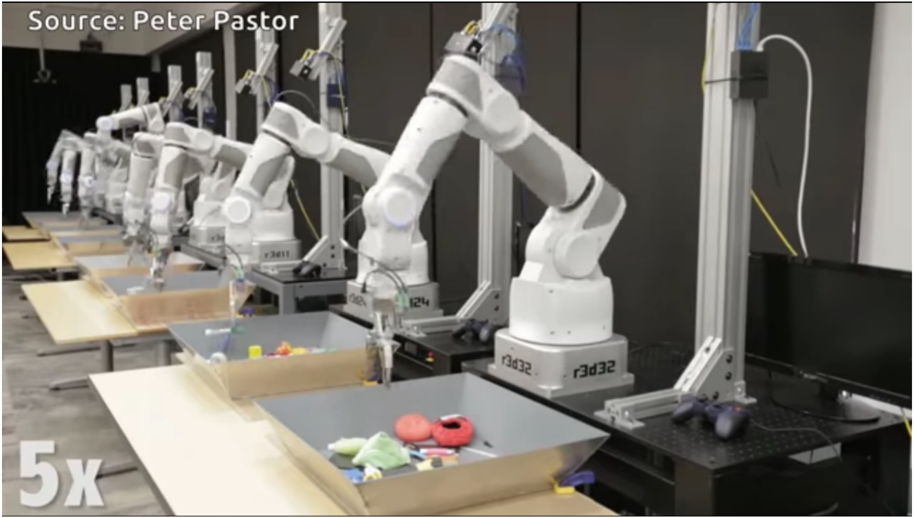


Figure: Application of Reinforcement Learning

What is Reinforcement Learning

RL is the general-purpose framework for decision-making:

- There is an agent that can **act**

What is Reinforcement Learning

RL is the general-purpose framework for decision-making:

- There is an agent that can **act**
- Every action influences the future **state** of the agent

What is Reinforcement Learning

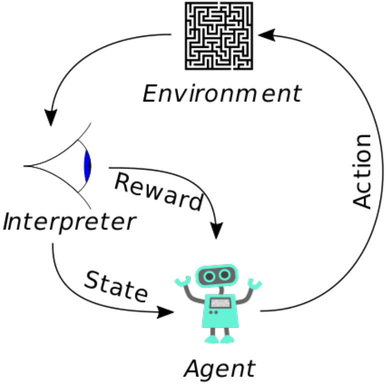
RL is the general-purpose framework for decision-making:

- There is an agent that can **act**
- Every action influences the future **state** of the agent
- Success is measured by a **reward**

What is Reinforcement Learning

RL is the general-purpose framework for decision-making:

- There is an agent that can **act**
- Every action influences the future **state** of the agent
- Success is measured by a **reward**
    - **Objective:** choose actions that **maximize future reward**



Where did it come from

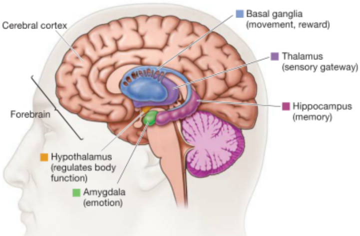
R. Sutton, A. Barto - inspired by Pavlovian and instumental conditioning.

Where did it come from

R. Sutton, A. Barto - inspired by Pavlovian and instumental conditioning.
D. Bertsekas and J. Tsitsiklis - stochastic approximations to dynamic programming methods (called "neuro-dynamic programming").

Where did it come from

R. Sutton, A. Barto - inspired by Pavlovian and instumental conditioning.
D. Bertsekas and J. Tsitsiklis - stochastic approximations to dynamic programming methods (called "neuro-dynamic programming").
Neural implementation - dopamine dependent learning in the basal ganglia.



Markov decision process (MDP)

Reinforcement learning is defined as a MDP task.
The basic assumption is that:

Definition

A state s_t is Markov iff:

$$P[s_{t+1}|s_t] = P[s_{t+1}|s_1, ..., s_t]$$

This means, that the state captures all relevant information from history.

Markov decision process(MDP)

Definition

Under this assumption, the the Markov decision process in which all states are Markov is given a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where:

Markov decision process(MDP)

Definition

Under this assumption, the the Markov decision process in which all states are Markov is given a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where:

- $\mathcal{S}$ is a space of states

Markov decision process(MDP)

Definition

Under this assumption, the the Markov decision process in which all states are Markov is given a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where:

- $\mathcal{S}$ is a space of states
- $\mathcal{A}$ is a space of actions

Markov decision process(MDP)

Definition

Under this assumption, the the Markov decision process in which all states are Markov is given a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where:

- $\mathcal{S}$ is a space of states
- $\mathcal{A}$ is a space of actions
- $\mathcal{P} : s \times a \to s'$ is a state transition function (potentially stochastic)

Markov decision process(MDP)

Definition

Under this assumption, the the Markov decision process in which all states are Markov is given a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where:

- $\mathcal{S}$ is a space of states
- $\mathcal{A}$ is a space of actions
- $\mathcal{P} : s \times a \to s'$ is a state transition function (potentially stochastic)
- $\mathcal{R} : s \times a \to \mathbb{R}$ is a reward function (potentially stochastic)

Solving RL

In order to solve RL task we need the following functions (the following notation is used s - state, a - action, r_t - reward at time t, $\gamma \in [0, 1]$ - the discount):

Solving RL

In order to solve RL task we need the following functions (the following notation is used s - state, a - action, r_t - reward at time t, $\gamma \in [0, 1]$ - the discount):

- **Policy** - models the agent's action selection. It can be:

Solving RL

In order to solve RL task we need the following functions (the following notation is used s - state, a - action, r_t - reward at time t, $\gamma \in [0, 1]$ - the discount):

- **Policy** - models the agent's action selection. It can be:
  - Deterministic policy - $\pi(s)$

Solving RL

In order to solve RL task we need the following functions (the following notation is used s - state, a - action, r_t - reward at time t, $\gamma \in [0, 1]$ - the discount):

- **Policy** - models the agent's action selection. It can be:
  - Deterministic policy - $\pi(s)$
  - Stochastic policy - $\pi : A \times S \to [0, 1], \pi(a|s) = P[a_t = a|s_t = s]$

Solving RL

In order to solve RL task we need the following functions (the following notation is used s - state, a - action, r_t - reward at time t, $\gamma \in [0, 1]$ - the discount):

- **Policy** - models the agent's action selection. It can be:
  - Deterministic policy - $\pi(s)$
  - Stochastic policy - $\pi : A \times S \rightarrow [0, 1], \pi(a|s) = P[a_t = a|s_t = s]$
- **Value function** - gives the expected utility of taking a certain action at a certain step. It can be:

Solving RL

In order to solve RL task we need the following functions (the following notation is used s - state, a - action, r_t - reward at time t, $\gamma \in [0,1]$ - the discount):

- **Policy** - models the agent's action selection. It can be:
  - Deterministic policy - $\pi(s)$
  - Stochastic policy - $\pi : A \times S \to [0,1], \pi(a|s) = P[a_t = a | s_t = s]$
- **Value function** - gives the expected utility of taking a certain action at a certain step. It can be:
  - **Action value function** - expected discounted total reward given an action and a state

Solving RL

In order to solve RL task we need the following functions (the following notation is used s - state, a - action, r_t - reward at time t, $\gamma \in [0,1]$ - the discount):

- **Policy** - models the agent's action selection. It can be:
  - Deterministic policy - $\pi(s)$
  - Stochastic policy - $\pi : A \times S \rightarrow [0,1], \pi(a|s) = P[a_t = a | s_t = s]$
- **Value function** - gives the expected utility of taking a certain action at a certain step. It can be:
  - **Action value function** - expected discounted total reward given an action and a state

$$q_\pi(s,a) = E[r_{t+1} + \gamma r_{t+2} + ... | s_t = s, a_t = a]$$

Solving RL

In order to solve RL task we need the following functions (the following notation is used s - state, a - action, r_t - reward at time t, $\gamma \in [0, 1]$ - the discount):

- **Policy** - models the agent's action selection. It can be:
  - Deterministic policy - $\pi(s)$
  - Stochastic policy - $\pi : A \times S \rightarrow [0, 1], \pi(a|s) = P[a_t = a|s_t = s]$
- **Value function** - gives the expected utility of taking a certain action at a certain step. It can be:
  - **Action value function** - expected discounted total reward given an action and a state

$$q_\pi(s, a) = E[r_{t+1} + \gamma r_{t+2} + ...|s_t = s, a_t = a]$$

  - **State-value function** - expected discounted total reward given a state

Solving RL

In order to solve RL task we need the following functions (the following notation is used s - state, a - action, r_t - reward at time t, $\gamma \in [0, 1]$ - the discount):

- **Policy** - models the agent's action selection. It can be:
  - Deterministic policy - $\pi(s)$
  - Stochastic policy - $\pi : A \times S \to [0, 1], \pi(a|s) = P[a_t = a|s_t = s]$
- **Value function** - gives the expected utility of taking a certain action at a certain step. It can be:
  - **Action value function** - expected discounted total reward given an action and a state

$$q_\pi(s, a) = E[r_{t+1} + \gamma r_{t+2} + ...|s_t = s, a_t = a]$$

  - **State-value function** - expected discounted total reward given a state

$$v_\pi(s) = E[r_{t+1} + \gamma r_{t+2} + ...|s_t = s]$$

- **Model** - agent's representation of environment. It consists of:

- **Model** - agent's representation of environment. It consists of:
  - State transition function:

$$\mathcal{P}(s, s', a) = P[s_{t+1} = s' | s_t = s, a_t = a]$$

- **Model** - agent's representation of environment. It consists of:
  - State transition function:

$$\mathcal{P}(s, s', a) = P[s_{t+1} = s' | s_t = s, a_t = a]$$

  - Expected reward:

$$\mathcal{R}(s, a) = E[r_{t+1} | s_t = s, a_t = a]$$

- **Model** - agent's representation of environment. It consists of:
  - State transition function:

$$\mathcal{P}(s, s', a) = P[s_{t+1} = s' | s_t = s, a_t = a]$$

  - Expected reward:

$$\mathcal{R}(s, a) = E[r_{t+1} | s_t = s, a_t = a]$$

Disclaimer : all three can be approximated by Neural Networks!

- **Model** - agent's representation of environment. It consists of:
  - State transition function:

$$\mathcal{P}(s, s', a) = P[s_{t+1} = s' | s_t = s, a_t = a]$$

  - Expected reward:

$$\mathcal{R}(s, a) = E[r_{t+1} | s_t = s, a_t = a]$$

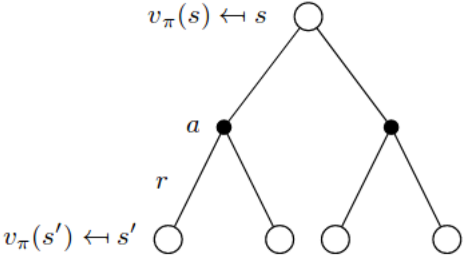
Disclaimer : all three can be approximated by Neural Networks!
Objective : maximizing expected reward, discounted with $\gamma \in [0, 1]$ for numerical stability reasons.

Bellman expectation equation

State or action value function can be decomposed into immediate reward plus the discounted value of successor rate:

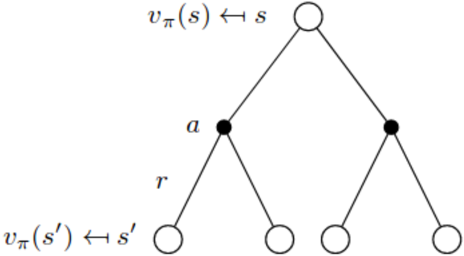
Bellman expectation equation

State or action value function can be decomposed into immediate reward plus the discounted value of successor rate:



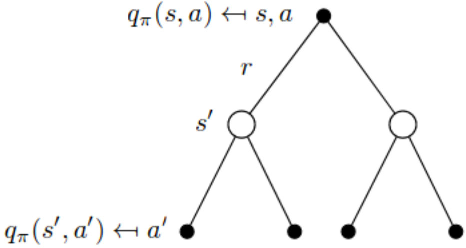
Bellman expectation equation

State or action value function can be decomposed into immediate reward plus the discounted value of successor rate:

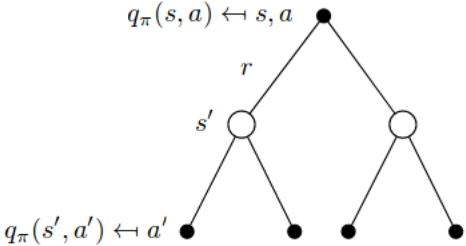


$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(\mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{P}(s,s',a) v_\pi(s'))$$

Bellman expectation equation



Bellman expectation equation



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

Optimal value function

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies:

Optimal value function

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies:

$$v_*(s) = max_\pi v_\pi(s)$$

Optimal value function

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies:

$$v_*(s) = max_\pi v_\pi(s)$$

The *optimal action-value function* $v_*(s)$ is the maximum action-value function over all policies:

Optimal value function

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies:

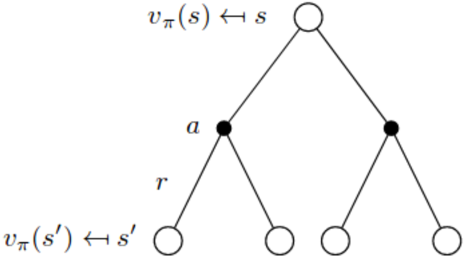
$$v_*(s) = max_\pi v_\pi(s)$$

The *optimal action-value function* $v_*(s)$ is the maximum action-value function over all policies:

$$q_*(s, a) = max_\pi q_\pi(s, a)$$

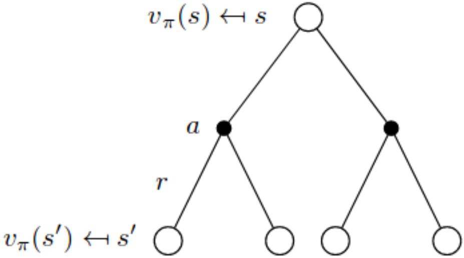
Bellman optimality equation

Optimal value functions are recursively related:



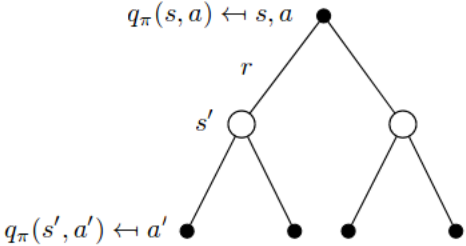
Bellman optimality equation

Optimal value functions are recursively related:

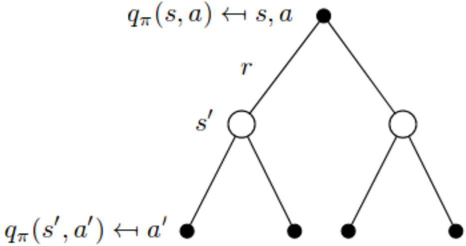


$$v_*(s) = \max_{a \in \mathcal{A}} \mathcal{R}(s, a) + \gamma \sum_{s' \in S} \mathcal{P}(s, s', a) v_*(s')$$

Bellman optimality equation

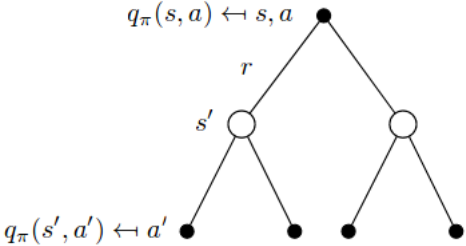


Bellman optimality equation



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

Bellman optimality equation



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

Both can be solved by iterative algorithm:

Iteration algorithm

- Assign each state a random value

Iteration algorithm

- Assign each state a random value
- For each state, find its value function based on its neighbors' values

Iteration algorithm

- Assign each state a random value
- For each state, find its value function based on its neighbors' values
- We do so by:

$$v_{i+1}(s) = \max_{a \in \mathcal{A}} \mathcal{R}(s, a) + \gamma \sum_{s' \in S} \mathcal{P}(s, s', a) v_i(s')$$

Iteration algorithm

- Assign each state a random value
- For each state, find its value function based on its neighbors' values
- We do so by:

$$v_{i+1}(s) = \max_{a \in \mathcal{A}} \mathcal{R}(s, a) + \gamma \sum_{s' \in S} \mathcal{P}(s, s', a) v_i(s')$$

- If no value changes by more than some $\epsilon > 0$, we stop.

Q-learning - model-free

The *model-free* framework has no knowledge of MDP transitions/rewards.

Q-learning - model-free

The *model-free* framework has no knowledge of MDP transitions/rewards. The agent must experience each state and each transition at least once to know the rewards.

Q-learning - model-free

The *model-free* framework has no knowledge of MDP transitions/rewards. The agent must experience each state and each transition at least once to know the rewards.
And the Q-Learning is the approach to implement the *value iteration update* in this case, having the *learning rate* α:

$$q(s_t, a_t) = (1 - \alpha)q(s_t, a_t) + \alpha(r_t + \gamma max_{a \in \mathcal{A}} q(s_{t+1}, a))$$

Unique fixed point - Banach fixed-point theorem

We have that the value iteration converges to v_*, which is also a unique solution. It is true for the Bellman expectation operator, i.e.:

Unique fixed point - Banach fixed-point theorem

We have that the value iteration converges to v_*, which is also a unique solution. It is true for the Bellman expectation operator, i.e.:

$$T^\pi(v) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v,$$

Unique fixed point - Banach fixed-point theorem

We have that the value iteration converges to v_*, which is also a unique solution. It is true for the Bellman expectation operator, i.e.:

$$T^\pi(v) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v,$$

which operates on the Banach spaces. This operator is a contraction:

Unique fixed point - Banach fixed-point theorem

We have that the value iteration converges to v_*, which is also a unique solution. It is true for the Bellman expectation operator, i.e.:

$$T^\pi(v) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v,$$

which operates on the Banach spaces. This operator is a contraction:

$$\|T^\pi(v) - T^\pi(u)\| \leq \gamma \|P^\pi\| \|u - v\|_\infty \leq \gamma \|u - v\|_\infty,$$

Unique fixed point - Banach fixed-point theorem

We have that the value iteration converges to v_*, which is also a unique solution. It is true for the Bellman expectation operator, i.e.:

$$T^\pi(v) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v,$$

which operates on the Banach spaces. This operator is a contraction:

$$\|T^\pi(v) - T^\pi(u)\| \leq \gamma \|P^\pi\| \|u - v\|_\infty \leq \gamma \|u - v\|_\infty,$$

which together with the Banach fixed-point theorem gives us the desired statement.

The following are the problems, that can be modelled as MDPs:

- Robocup soccer

Q-learning - model-free

The following are the problems, that can be modelled as MDPs:

- Robocup soccer
- Quake

Q-learning - model-free

The following are the problems, that can be modelled as MDPs:

- Robocup soccer
- Quake
- Game of Go

Q-learning - model-free

The following are the problems, that can be modelled as MDPs:

- Robocup soccer
- Quake
- Game of Go
- Walking robot

Q-learning - model-free

The following are the problems, that can be modelled as MDPs:

- Robocup soccer
- Quake
- Game of Go
- Walking robot
- Portfolio management

Q-learning - model-free

The following are the problems, that can be modelled as MDPs:

- Robocup soccer
- Quake
- Game of Go
- Walking robot
- Portfolio management
- Helicopter

Q-learning - model-free

The following are the problems, that can be modelled as MDPs:

- Robocup soccer
- Quake
- Game of Go
- Walking robot
- Portfolio management
- Helicopter

For most of these and similar problems we have the following issues:

Q-learning - model-free

The following are the problems, that can be modelled as MDPs:

- Robocup soccer
- Quake
- Game of Go
- Walking robot
- Portfolio management
- Helicopter

For most of these and similar problems we have the following issues:

- MDP model is unknown, but experience can be sampled

Q-learning - model-free

The following are the problems, that can be modelled as MDPs:

- Robocup soccer
- Quake
- Game of Go
- Walking robot
- Portfolio management
- Helicopter

For most of these and similar problems we have the following issues:

- MDP model is unknown, but experience can be sampled
- MDP model is known, but is too big to use, if only by samples

Q-learning - model-free

The following are the problems, that can be modelled as MDPs:

- Robocup soccer
- Quake
- Game of Go
- Walking robot
- Portfolio management
- Helicopter

For most of these and similar problems we have the following issues:

- MDP model is unknown, but experience can be sampled
- MDP model is known, but is too big to use, if only by samples

Model-free RL can solve these issues!

Q-learning example by hand

- $$R = \begin{array}{c|cccccc} & \multicolumn{6}{c}{\textbf{Action}} \\ \textbf{State} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & 100 \\ 2 & -1 & -1 & -1 & 0 & -1 & -1 \\ 3 & -1 & 0 & 0 & -1 & 0 & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & 0 & -1 & -1 & 0 & 100 \end{array}$$

Q-learning example by hand

- $$R = \begin{array}{c|cccccc} \text{State} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & 100 \\ 2 & -1 & -1 & -1 & 0 & -1 & -1 \\ 3 & -1 & 0 & 0 & -1 & 0 & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & 0 & -1 & -1 & 0 & 100 \end{array}$$
  (Action: columns 0–5)

- $$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Q-learning example by hand

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

- $$Q = \begin{array}{c c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- $$Q = \begin{array}{c c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Q-learning example by hand

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

- $$Q = \begin{array}{c|c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- $$Q = \begin{array}{c|c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- $$Q = \begin{array}{c|c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

Q-learning example by hand

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

- $$Q = \begin{array}{cc} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0\\1\\2\\3\\4\\5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- $$Q = \begin{array}{cc} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0\\1\\2\\3\\4\\5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- $$Q = \begin{array}{cc} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0\\1\\2\\3\\4\\5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

- $$Q = \begin{array}{cc} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0\\1\\2\\3\\4\\5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

Q-learning example by hand

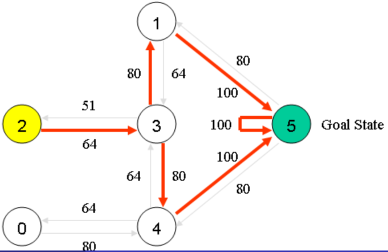
- $$R = \begin{array}{cc} & \text{Action} \\ \text{State} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1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

- $$Q = \begin{array}{cc} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- $$Q = \begin{array}{cc} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- $$Q = \begin{array}{cc} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

- $$Q = \begin{array}{cc} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

- 

Q-learning - code example

```
import numpy.random as rnd
learning_rate0 = 0.05
learning_rate_decay = 0.1
n_iterations = 20000
s = 0 # start in state 0
Q = np.full((3, 3), -np.inf)
# -inf for impossible actions

for state, actions in enumerate(possible_actions):

    Q[state, actions] = 0.0
```

Q-learning - code example

```
for iteration in range(n_iterations):

    a = rnd.choice(possible_actions[s])
    # choose an action (randomly)
    sp = rnd.choice(range(3), p=T[s, a])
    # pick next state using T[s, a]
    reward = R[s, a, sp]
    learning_rate = learning_rate0 /\
    (1 + iteration * learning_rate_decay)
    print learning_rate
    Q[s, a] = learning_rate * Q[s, a] \
    + (1 - learning_rate) * ( reward + \
    discount_rate * np.max(Q[sp]) )
    s = sp # move to next state
```