REINFORCEMENT LEARNING: AN INTRODUCTION

Ianis Lallemand, 24 octobre 2012

This presentation is based largely on the book:

Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, MIT Press, Cambridge, MA, 1998

CONTENTS

1. General definition
2. Elements of reinforcement learning
3. Example 1: Tic-Tac-Toe
4. Example 2: *n*–armed bandit
5. The full reinforcement learning problem
6. Dynamic programming
7. Monte Carlo Methods
8. Temporal Difference learning

GENERAL DEFINITION

GENERAL DEFINITION

"Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them."

GENERAL DEFINITION

GENERAL DEFINITION

"Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them."

GENERAL DEFINITION

GENERAL DEFINITION

"Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them."

SUPERVIZED LEARNING

Reinforcement learning is different from supervized learning (pattern recognition, neural networks, etc).

Supervized learning is learning from examples provided by a knowledgeable external supervizor.

In reinforcement learning the agent learns from his own behavior.

GENERAL DEFINITION

AGENTS

The AGENT performs the reinforcement learning task.

1. It has explicite goals (problem for music…).

2. It can sense aspect of the ENVIRONMENT (environment described in terms of STATES).

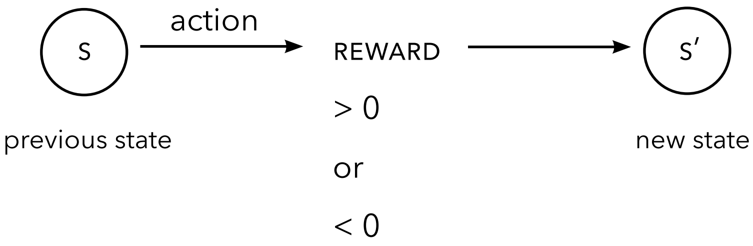
3. It performs ACTIONS to influence the environment.

ELEMENTS OF REINFORCEMENT LEARNING

REWARD FUNCTION

It defines the goal in a reinforcement learning problem.

It gives the agent a sense of what is good in an immediate sense (pleasure / pain).



s

action

REWARD

> 0

or

< 0

s′

previous state

new state

ELEMENTS OF REINFORCEMENT LEARNING

VALUE FUNCTION

It gives the agent a sense of what is good in the long run.

It is either:

1. A function of the environment's STATES (STATE VALUE FUNCTION).
2. A function of the environment's STATES and of the agent's ACTIONS (ACTION VALUE FUNCTION).

INTERPRETATION

The VALUE of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

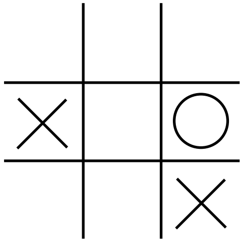
ELEMENTS OF REINFORCEMENT LEARNING

MODEL OF THE ENVIRONMENT

It is used to predict the states the environment will be in after the agent performs its actions.

In reinforcement learning, the agent often uses the model to compute series of potential state–action sequences: it projects himself in the future to decide which action to perform in the present.

EXAMPLE 1: TIC—TAC—TOE



APPROACH

Reinforcement learning with approximate value functions.

REWARD

+1 for winning the game.

VALUE FUNCTION

A table storing the last estimated probability of our winning from each state of the game (init at 0.5).

EXAMPLE 1: TIC—TAC—TOE

GREEDY MOVES

1. Look at states that could result from our possible moves.

2. Look at VALUE FUNCTION values in those states (expected reward from these states).

3. Select action leading to state with highest value (GREEDY MOVE).

EXAMPLE 1: TIC—TAC—TOE

GREEDY MOVES

1. Look at states that could result from our possible moves.

2. Look at VALUE FUNCTION values in those states (expected reward from these states).

3. Select action leading to state with highest value (GREEDY MOVE).

EXPLORATORY MOVES

Once in a while, perform a random move (EXPLORATORY MOVE).

Important to force the agent to explore new solutions (avoid local maximum).

EXAMPLE 1: TIC—TAC—TOE

LEARNING

Play many games.

EXAMPLE 1: TIC—TAC—TOE

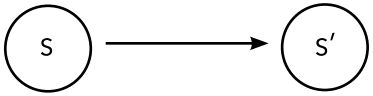
LEARNING

Play many games.

UPDATE

After each move, change the value of the state prior to the move

(re-estimate our probability of winning from that state)



previous state new state

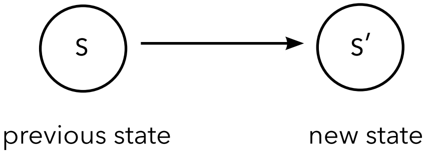
EXAMPLE 1: TIC—TAC—TOE

LEARNING

Play many games.

UPDATE

After each move, change the value of the state prior to the move (re-estimate our probability of winning from that state).



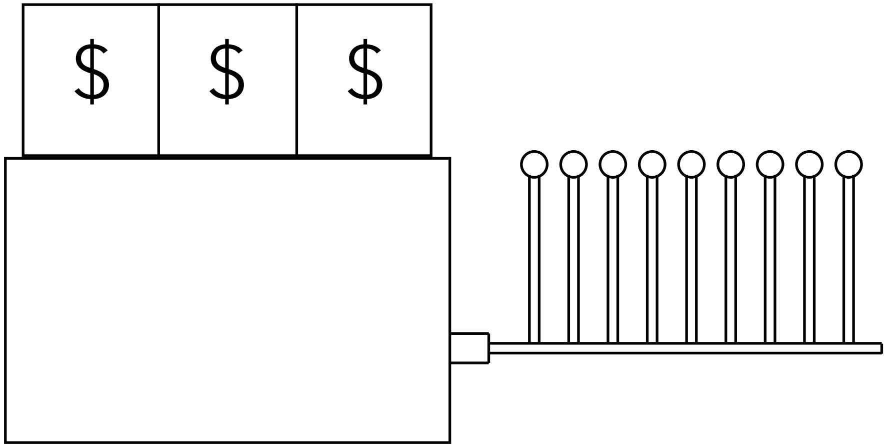
BACK-UP

$V(s) = V(s) + a\,(V(s') - V(s))$

a: STEP-SIZE

If a decreases over time, converges to true probabilities of winning.

EXAMPLE 2: *N*–ARMED BANDIT



SYSTEM

An *n*–armed "bandit" casino machine.

Each arm gives a numerical reward sampled from its own stationary probability distribution.

GOAL

Find the best way to play (find the best arm).

EXAMPLE 2: *N*–ARMED BANDIT

REMARK

Since distributions are stationary, the system is always in the same state.

Rewards are not associated with values alone, but with ACTIONS AND VALUES.

VALUE FUNCTION

The value function is an ACTION-VALUE FUNCTION.

It gives the expected reward after selecting an action (which arm to pull).

EXAMPLE 2: *N*–ARMED BANDIT

REMARK

Since distributions are stationary, the system is always in the same state.

Rewards are not associated with values alone, but with ACTIONS AND VALUES.

VALUE FUNCTION

The value function is an ACTION-VALUE FUNCTION.

It gives the expected reward after selecting an action (which arm to pull).

APPROACH

Reinforcement learning with tabular ACTION-VALUE FUNCTION.

Store in a table the current estimated values of each action.

The true value of an action is the average reward received when this action is selected (i.e. the mean of the arm’s stationary distribution).

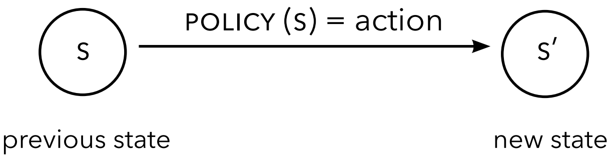
THE FULL REINFORCEMENT LEARNING PROBLEM

POLICY (π)

It is the fourth basic element of reinforcement learning.

It is a mapping from environment states to actions.

It defines the agent’s way of behaving at a given time.



THE FULL REINFORCEMENT LEARNING PROBLEM

RETURN

It is the expected total reward in the long run.

It can be estimated from predicted rewards.

THE FULL REINFORCEMENT LEARNING PROBLEM

RETURN	It is the expected total reward in the long run.
	It can be estimated from predicted rewards.
DISCOUNTED RETURN	$R_t = r_{t+1} + \gamma\ r_{t+2} + \gamma^2\ r_{t+3} + \ldots$
	Where $\gamma < 1$.

THE FULL REINFORCEMENT LEARNING PROBLEM

RETURN

It is the expected total reward in the long run.

It can be estimated from predicted rewards.

DISCOUNTED RETURN

$R_t = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + \ldots$

Where $\gamma < 1$.

VALUE ESTIMATION

State–value function under policy π:

$V^\pi(s) = E_\pi\{R_t | s_t = s\}$

Action–value function under policy π:

$Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\}$

DYNAMIC PROGRAMMING

KEY IDEA

Use of value functions to organize and structure the search for good policies.

POLICY ITERATION

Alternate between:

1. POLICY EVALUATION: iterative computation of value functions for a given policy.
2. POLICY IMPROVEMENT: computation of an improved policy given the value function for that policy.

DYNAMIC PROGRAMMING

POLICY EVALUATION

Goal: computing the state-value function for a given policy.

BELLMAN OPTIMALITY EQUATION

The value of a state under an optimal policy π* must equal the expected return for the best action from that state under this policy :

$$V^* (s) = \mathbf{max}\ Q^{\pi^*}(s, a)$$

Where **max** is computed amongst all actions that can be taken from s.

ITERATIVE POLICY EVALUATION

Iterative version of Bellman optimality equation.

The update of V (s) is based on old estimates of V (s) : BOOTSTRAPPING.

DYNAMIC PROGRAMMING

POLICY IMPROVEMENT

If, in a given state s, there is an action $a \neq \pi(s)$ such that :

$Q^{\pi}(s, a) > V^{\pi}(s)$

Then we should change the policy π to select action a each time s is encountered.

THEOREM

For deterministic policies π and π′,

if $Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s)$ (π′ would be built from π as explained above)

Then π′ must be as good, or better than, π (i.e. $V^{\pi'}(s) \geq V^{\pi}(s)$ for all s)

Policy improvement gives better policies except when the current policy is already optimal.

MONTE CARLO METHODS

MONTE CARLO

Learn value functions and optimal policies in the form of SAMPLE EPISODES.

(only for EPISODIC TASKS: task for which there exists a final state, e.g. Tic-Tac-Toe)

GENERAL IDEA

If an agent:

1. follows π and maintains an average of actual returns that have followed each encountered state, the average converges to the STATE-VALUE FUNCTION for policy π.

2. maintains an average of actual returns that have followed each action taken from all encountered states, the average converges to the ACTION-VALUE FUNCTION for policy π.

MONTE CARLO METHODS

CONSTANT-α MONTE CARLO

Perform actions until the end of the episode is reached.

At the end of the episode, R_t is known for all t (all rewards are known).

For all t, update the state-value function with:

$$V (s_t) = V (s_t) + \alpha (R_t - V (s_t))$$

INTERPRETATION

Update V (s_t) towards R_t , which is the average reward received starting from state t in the episode.

(recall that V (s_t) should be equal to the true average reward received starting from state s_t , which we estimate here in the sample episode by R_t)

TEMPORAL DIFFERENCE LEARNING

TD LEARNING METHODS

Combination of ideas from Monte Carlo and Dynamic Programming.

They can learn without a model of the environment (like Monte Carlo), through sampling.

They bootstrap (like Dynamic Programming).

BASIC EXAMPLE

When in state s_{t+1}, update $V (s_t)$ by:

$$V (s_t) = V (s_t) + \alpha (r_{t+1} + \gamma V (s_{t+1}) - V (s_t))$$

(note that r_{t+1} is received *before* reaching state s_{t+1})

TEMPORAL DIFFERENCE LEARNING

SARSA

SARSA is an ON-POLICY CONTROL method.

CONTROL: estimating ideal policies through the action-value function.

ON-POLICY: evaluate or improve the policy that is used to make decisions.

UPDATE RULE

$$Q (s_t, a_t) = Q (s_t, a_t) + \alpha (r_{t+1} + \gamma Q (s_{t+1}, a_{t+1}) - Q (s_t, a_t))$$

$s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$: sarsa

TEMPORAL DIFFERENCE LEARNING

Q-LEARNING

Q-LEARNING is an OFF-POLICY CONTROL method.

OFF-POLICY: use two different policies. The policy used to generate behavior (BEHAVIOR POLICY) may be unrelated to the policy that is evaluated and improved (ESTIMATION POLICY).

UPDATE RULE

$$Q (s_t, a_t) = Q (s_t, a_t) + \alpha (r_{t+1} + \gamma \max_{\mathbf{a}} \mathbf{Q (s_{t+1}, a)} - Q (s_t, a_t))$$

E.g.:

1. Use a EPSILON-GREEDY policy (behavior policy) to select a_t from s_t.

2. Update is performed by greedy selection of action a = **max Q (s_{t+1}, a)** . Action a_{t+1} is not taken (it will be taken at next iteration following the EPSILON-GREEDY policy).