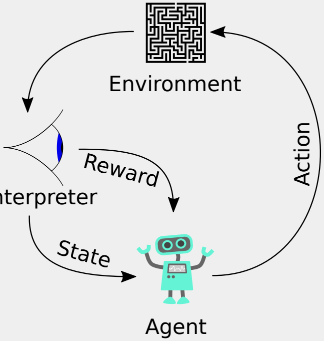
Introduction to Reinforcement Learning

Multiarmed Bandit Problem, Finite MDPs

Felix Wagner

Seminar: Machine Learning
TU Vienna

27 11 2018

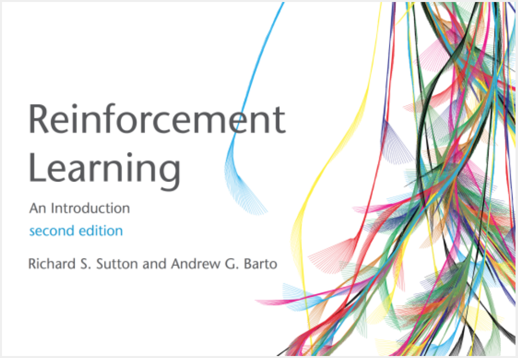


Overview

Contents:

Overview

Sutton S. Barto & Andrew G. Barto, Reinforcement Learning: An Introduction



Introduction

Categorization

Categorization

Supervised:

- Labeled data
- Regression or Classification
- e.g. "Which picture shows a cat?"

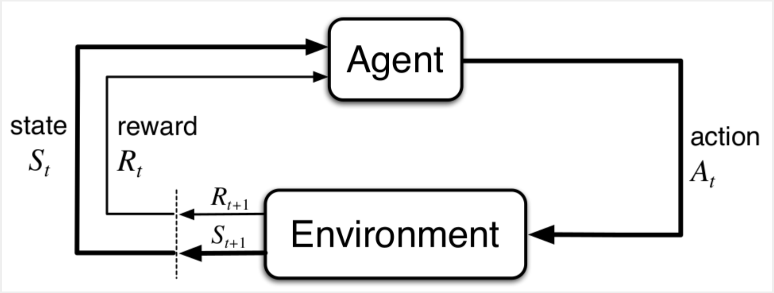
Unsupervised:

- Unlabeled data
- Finding structure
- e.g. Clustering data into similar groups to save data space

Reinforcement:

- No data
- Finding policy for decisions
- e.g. "Which is the best strategy to win tic-tac-toe?"

AGENT, ENVIRONMENT



For every timestep:

Agent sets action, based on state of the environment

Environment answers with reward and a new state

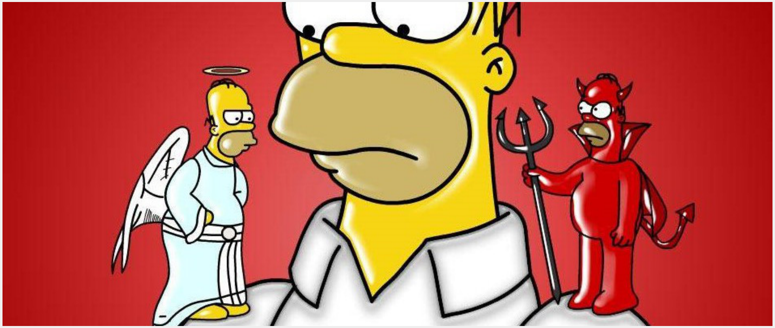
$S_0 \rightarrow A_0 \rightarrow \{R_1, S_1\} \rightarrow A_1 \rightarrow \{R_2, S_2\} \rightarrow A_2 \rightarrow \{R_3, S_4\} \dots$

Agent, Environment

Agent tries to maximize the rewards over time.

The reward contains no information about the expected future rewards from the new state.

Policy



The policy π tells the agent which action to set, depending on the state.
Deterministic policy $\pi(s)$: Returns action a for given state s.
Stochastic policy $\pi(a|s)$: Returns probability p of action a in state s.

Value Function

Value Function $v_\pi(s)$ assigns single numerical values to every state.
Action-Value Function $q_\pi(a, s)$ assigns single numerical value to every action - state pair.

Def.: An optimal policy π_* leades to expected returns that are greater than or equal to those of all other policies, for every state.

$v_*(s)$ denotes the value of a state under an optimal policy.
$q_*(a, s)$ denotes the value of an action-state pair under an optimal policy.

Exploration, Exploitation

At $t = 0$ the agent knows nothing about its environment: All values are set to some non-informative initial value.

Exploration: Try new actions, to get familiar with the environment.

Exploitation: Always use the action, that leads to the highest rewards over time (to your current knowledge). These are called greedy actions.
$A_t = \max_a q_\pi(a, s)$

Exploration, Exploitation

Exploitation leads to immediately higher rewards, but maybe some even better options are never discovered.

Exporation leads to immediately lower rewards, but in the long run the best options will be discovered.

Conclusio: The agent needs to do both. The less it knows about its environment, the more it needs to explore (and vice versa).

Exploration, Exploitation

The agent wants to enter the house. He tries climbing through the window first and it works. If he only uses greedy actions, he will never try to use the door.

Optional: Model of the Environment

Give agent some a-priori knowledge about the environment.

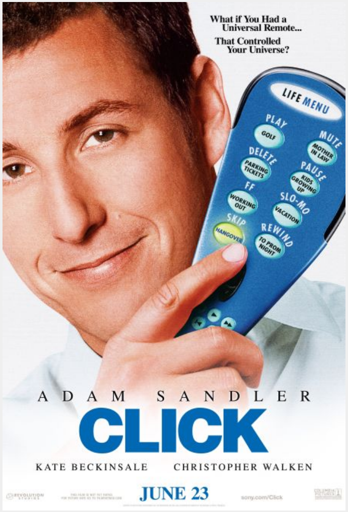
e.g. An agent plays a tic-tac-toe game. The opponent is part of the environment.

Without model: The environment seems to change arbirarily.

With model: The agent is aware, that the opponent will try to win. So the actions of the opponent can be modelled after the decisions, that the agent would make in his place.

Strategic Decisions

On which level is the decison made? Where is the border between agent and environment?
Which action produces which reward?
What information does the state hold?
Is there a model of the environment?



Reward Hypothesis and Human Decision Making

Reward Hypothesis

"That all of what we mean by goals and purpose can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signel (called reward)."

Human Decision Making

Feeling lonely: $R = -1$,
Feeling angry: $R = -0.5$
Getting an A on the exam: $R = 2.5$;
So an A is worth two times feeling lonely and one time feeling angry...?

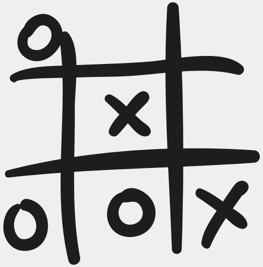
Obviousliy, human decision making is more complex and cannot be described by single numerical rewards.

Example: Tic-Tac-Toe

Two players on a 3×3 board,
3 stones in a row win. The agent is player X.

Game has $3^9 = 19683$ possible states.
Value function: Set up a table with a number (value) for each state.

The agent is player X, so:
Initialize all states with three X in a row to 1, all with three O to 0, all others to 0.5.



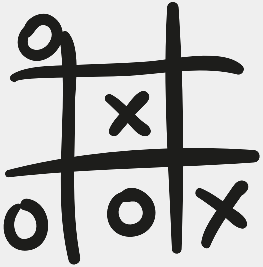
Example: Tic-Tac-Toe

Play many games against the opponent: Use greedy moves with probability $1 - \epsilon$ and explorative moves with probability ϵ. Update values after every greedy move:

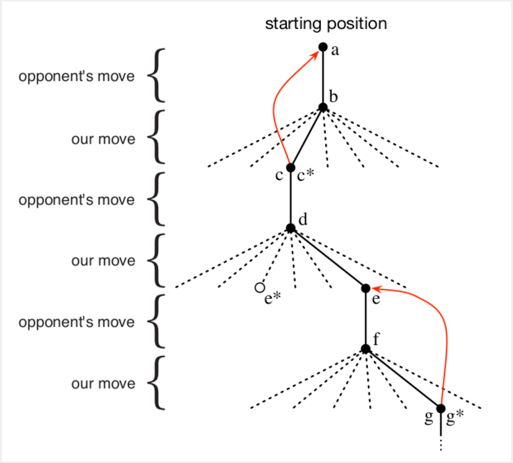
State Value Update Rule

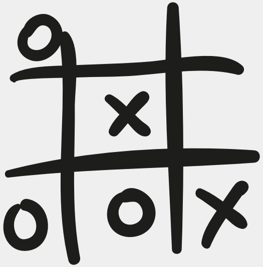
$$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)]$$

This is a form of temporal differences learning and a state value method.



Example: Tic-Tac-Toe





State Value Update Rule

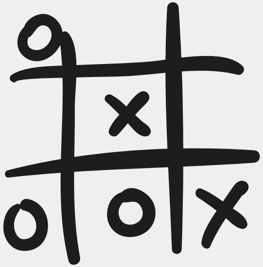
$$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)]$$

Example: Tic-Tac-Toe

State Value Update Rule

$$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)]$$

We reduce the stepsize parameter α over time:
The values converge towards the probabilities of winning from the current state.
(implementation on github)



Notice:

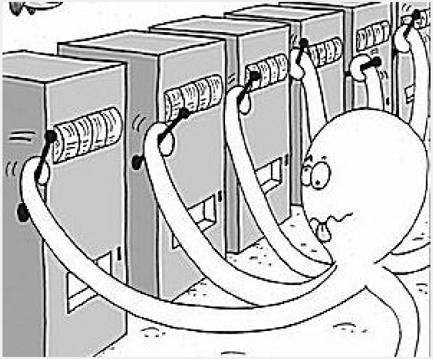
Model-free example.
Reward-free example.
No action values.
Suppose opponent makes mistakes.

Multiarmed Bandit Problem

Multiarmed Bandit

Slot machine with multiple levers, e.g. 10 levers.

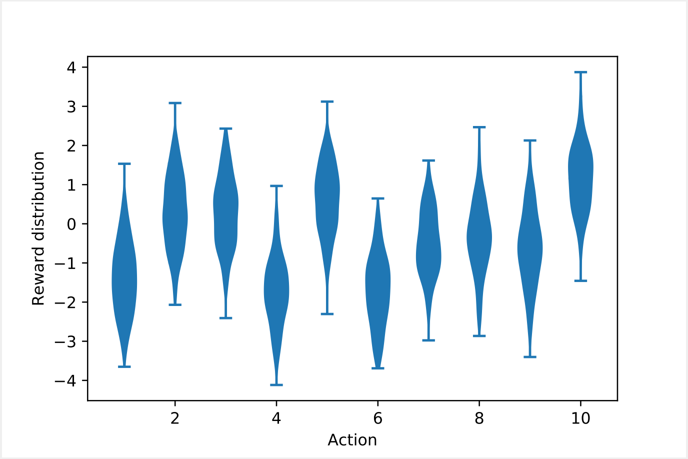
Notice:
Only one possible state.
Objective:
Maximize reward over *k* timesteps.



Estimate action values:

$Q_t(a) =$ (sum of rewards when *a* taken prior to *t*)/(number of times *a* taken prior to *t*)

Reward Distribution



Rewards: $R_n \propto \mathcal{N}(\mu = q_*(n), \sigma = 1)$ ... normal distribution
Real Action values: $q_*(n) \propto \mathcal{N}(\mu = 0, \sigma = 1)$

METHODS FOR POLICY LEARNING

optimal policy π_* ... a policy that maximizes the rewards over time (not necessarily unique!)

How to balance exploration and exploitation such that we arraive at an optimal policy?

Methods (e.g.):

- ϵ-Greedy Method
- Optimistic Initial Values
- UCB Action Selection
- Gradient Bandit Algorithm

ϵ-greedy Method

Reminder:

Greedy Action ... Action with the highest estimated value.

Algorithm

- Set up a table with initial estimations for every action value, e.g. $Q(n) = 0$.
- Choose $\epsilon \in [0, 1]$
- For every step:
-     With probability ϵ take a random non-greedy action.
-     Else: Take greedy action and update value.

ϵ-greedy Method

Update value according to

Estimate action values:

$Q_t(a) =$ (sum of rewards when a taken prior to t)/(number of times a taken prior to t)

This can be implemented recursively and efficiently:

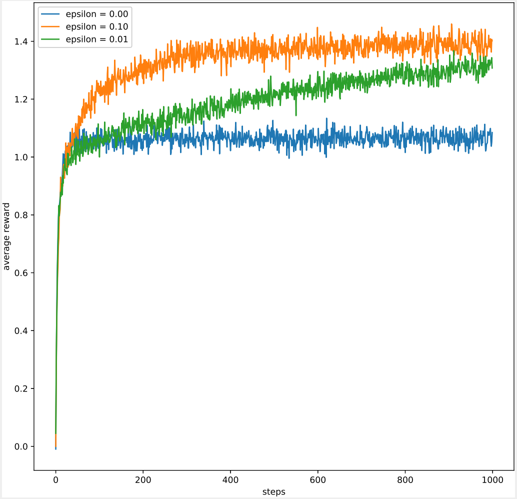
Action Value Update Rule

$$Q_{n+1} = Q_n + \frac{1}{n}\left[R_n - Q_n\right]$$

n ... number of times a taken prior to t
Q_n ... estimation of the true action value q_* after n received rewards
R_n .. n'th received reward for the action

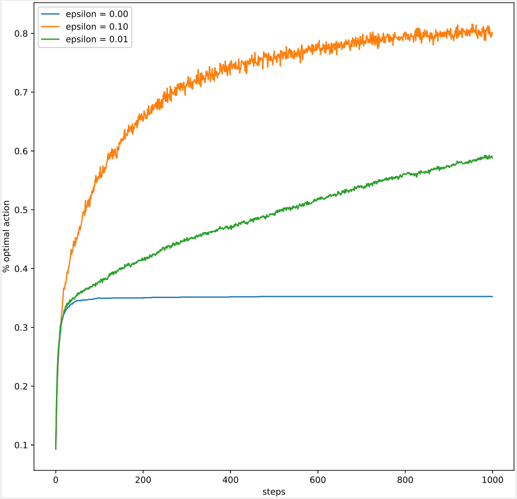
ϵ-greedy Method

Averaged performance over time step of 2000 indepentend 10 armed bandit problems.



ϵ-greedy Method

Averaged performance over time step of 2000 indepentend 10 armed bandit problems.



Optimistic Initial Values

Only one difference to ϵ-greedy method:

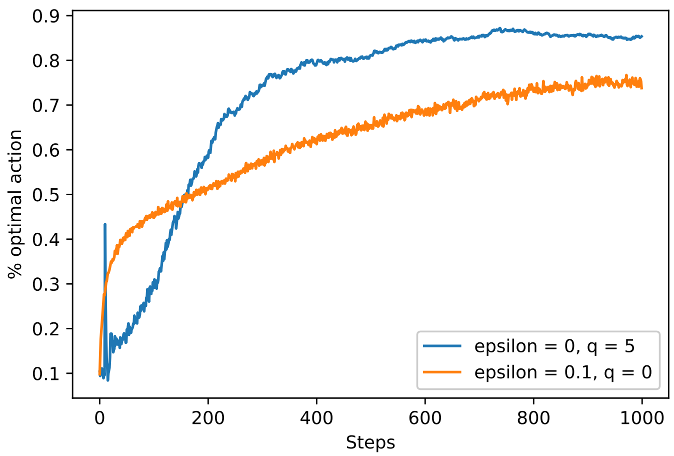
Algorithm

- Set up a table with **unrealistic high** initial estimations for every action value,
  e.g. $\mathbf{Q}(\mathbf{n}) = \mathbf{5}$.
- Choose $\epsilon \in [0, 1]$
- For every step:
- With probability ϵ take a random non-greedy action.
- Else: Take greedy action and update value.

In the pure version, optimistic value method uses $\epsilon = 0$.

Optimistic Initial Values

Averaged performance over time step of 2000 indepentend 10 armed bandit problems.



Upper Confidence Bound Action Selection

Until now, we distiguished between greedy and non-greedy actions. Now we look closer:

Select between non-greedy actions according to their potential for beeing optimal. This depends on:

- How close is their estimate to being maximal.
- Uncertainties in those estimates.

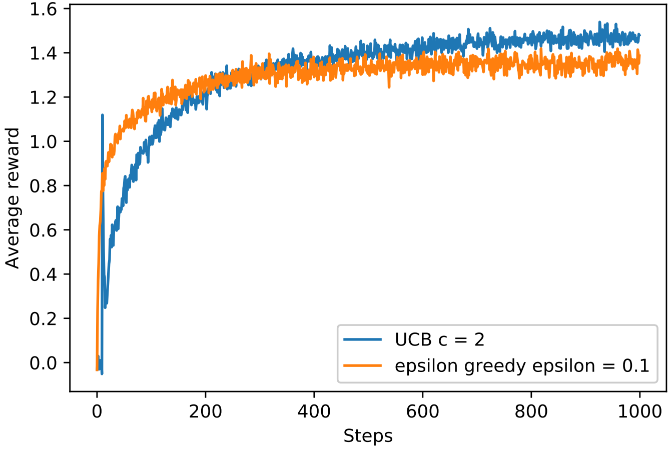
Action Selection Rule

$$A_t = arg\max_a \left[Q_t(a) + c\sqrt{\frac{ln(t)}{N_t(a)}} \right]$$

$N_t(a)$ ... number of times a was selected prior to time step t
$c > 0$ ... controls the degree of exploration
If $N_t(a) = 0$, then a is considered to be an maximizing action.

Upper Confidence Bound Action Selection

Averaged performance over time step of 2000 indepentend 10 armed bandit problems.



Gradient Bandit Algorithms

Value function $q(a)$ is an estimation of the rewards to come. Different approach: Learn preferences $H_t(a)$ for the actions instead of values. Only their relative difference is relevant!

Action Selection Rule (Probabilistic!)

$$Pr\{A_t = a\} = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} = \pi_t(a)$$

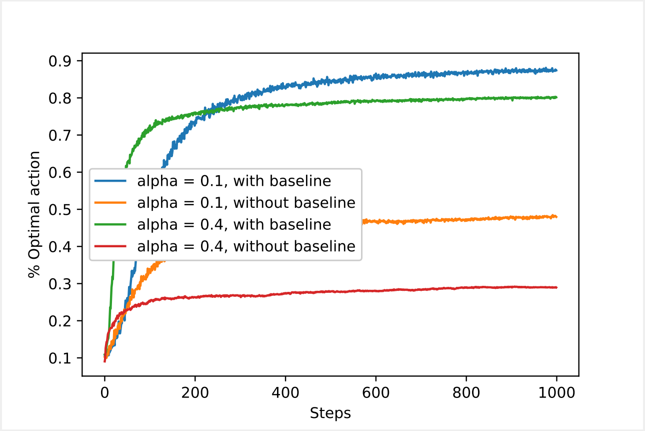
Preferences Update Rule

$H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \overline{R_t})(1 - \pi_t(A_t))$, and
$H_{t+1}(a) = H_t(a) + \alpha(R_t - \overline{R_t})\pi_t(a)$ for all $a \neq A_t$

$\alpha > 0$ ... step size parameter
$\overline{R}_t \in \mathbb{R}$ ... average of all rewards including step t (“baseline”)

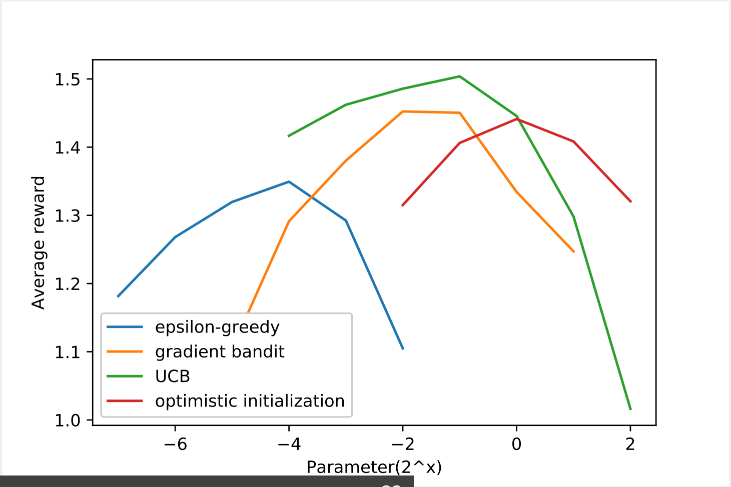
Gradient Bandit Algorithms

Averaged performance over time step of 2000 indepentend 10 armed bandit problems. Here, the true values $q_*(a)$ are $\propto \mathcal{N}(4, 1)$. With baseline this makes no difference on the performance.



Comparison of the Methods

Parameter Study: Summarize the learning curves by averaging over the first 1000 steps. Parameters: ϵ (ϵ-greedy), α (gradient bandit), c (UCB), Q_0 (optimistic initialization)



Recap: Policies

All the algorithms have a policy π that tells the algorithm which action in which state to choose (can also be probabilistic).

- ϵ-Greedy: $\pi(a_{greedy}) = 1 - \epsilon$, $\pi(a_{non-greedy}) = \epsilon$
- Optimistic Initial Values: same as ϵ-Greedy
- UCB Action Selection: $A_t = arg \max_a \left[Q_t(a) + c\sqrt{\frac{ln(t)}{N_t(a)}}\right]$
- Gradient Bandit Algorithm: $\pi_t(a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}}$

Recap: Update Rules, Values and Preferences

We want the policy to improve over time, so there is always an update rule for the policy.

- action value methods: $Q_{n+1} = Q_n + \frac{1}{n}\left[R_n - Q_n\right]$
- action preference methods:
  $H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \overline{R_t})(1 - \pi_t(A_t))$, and
  $H_{t+1}(a) = H_t(a) + \alpha(R_t - \overline{R_t})\pi_t(a)$ for all $a \neq A_t$

In the Tic Tac Toe Example we learned State Values $V(S_t)$, in the Bandit Problem we learned Action Values $Q_t(a)$. In the Gradient Bandit we learned Action Preferences instead of values.

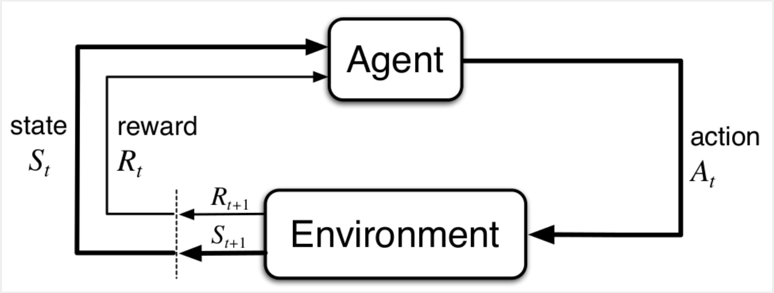
General Form for Update Rules

$$NewEstimate \leftarrow OldEstimate + Stepsize\left[Target - OldEstimate\right]$$

Finite Markov Decision Processes

Motivation

Generalization and rigorous framework for the Reinforcement Learning Problem.



Trajectory:
$S_0 \rightarrow A_0 \rightarrow \{R_1, S_1\} \rightarrow A_1 \rightarrow \{R_2, S_2\} \rightarrow A_2 \rightarrow \{R_3, S_4\}...$

Markov Decision Process

Def.: State “space” $\mathcal{S}$, action space $\mathcal{A}$ and reward space $\mathcal{R}$. We assume all as finite, the theory also works for countable infinite.

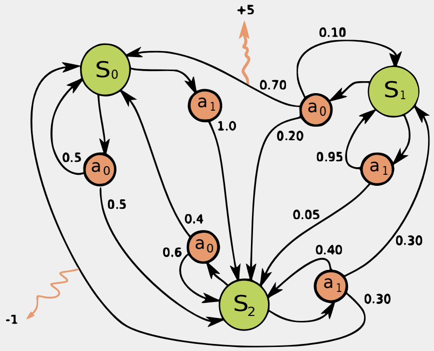
A Markov decision process is a 4-tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where

- $p : \begin{cases} \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0,1] \\ (s', r, s, a) \mapsto Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\} \end{cases}$
  is the dynamics function (conditional probability!).
- $r : \begin{cases} \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R} \\ (s, a, s') \mapsto r(s, a, s') \end{cases}$
  is the expected reward received after transitioning from state s to state s', due to action a.

Markov Decision Process

Two important characterstics:

- Well defined probabilities for all possible transitions.
- Each state has the Markov Property: The probabilities depend only on the current state, not on the past states. The state holds all necessary information for the future.



Returns and Discouting

The agent seeks to maximize the expected rewards:

Return

G_t, a function of the rewards, expected after time step t.
Simples case: $G_t = R_{t+1} + R_{t+2} + \cdots + R_T$

Simple sum makes only sense for episodic task (task ends after finitely many steps). For continuing task: Introduce discounting rate $\gamma \in [0, 1]$ and discounted return:

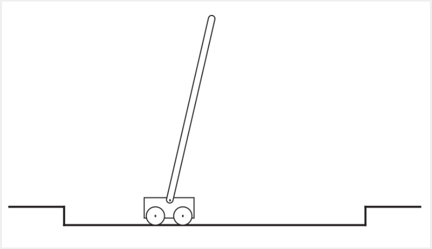
$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad \text{(converges)}$$

Returns satisfy recursive relation: $G_t = R_{t+1} + \gamma G_{t+1}$

Continuing and Episodic Tasks

Example of continuing task: Pole balancing
Idea for Rewards: $R = -1$ everytime failure occures, else $R = 0$

Possible actions: accelerate left and right
State: Angle of the pole to the floor, position on x-axis



Policies and Value Functions

The policy π is a mapping from states to probabilities of selecting each possible action:

Policy

$\pi(a|s) = Pr(A_t = a|S_t = s)$

Reminder:
Value function is the expected return when starting in s and following π thereafter:

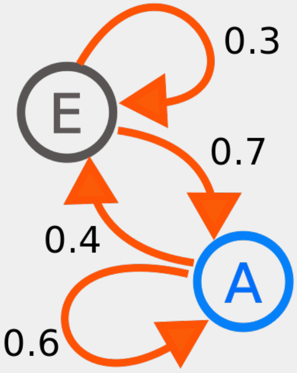
$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$$

Action value function is the expected return when starting in s, taking a and following π thereafter:

$$q_\pi(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a] = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

Policy and reduction to Markov Chain

MDP's are a generalization of markov chains, they introduce more than one possible action in any state. Together with a fixed policy the MPD reduces to a Markov chain.



Bellman Equation

The value function satisfies a recursive relation as well:

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}\big[G_t | S_t = s\big] && (1)\\
&= \mathbb{E}\big[R_{t+1} + \gamma G_{t+1} | S_t = s\big] && (2)\\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\big[r + \gamma \mathbb{E}[G_{t+1}|S_{t+1} = s']\big] && (3)\\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma v_\pi(s')\big] && (4)
\end{aligned}$$

The result is called the

Bellman Equation

$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma v_\pi(s')\big]$

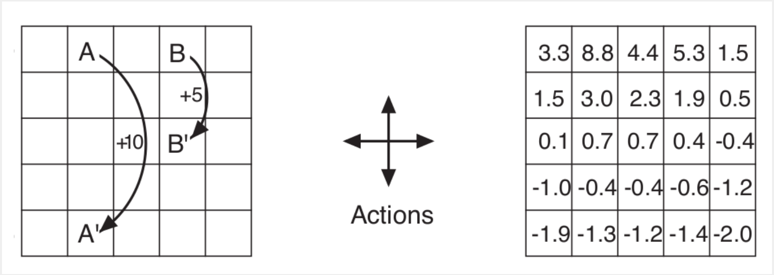
The value function v_π is its unique solution and can be found by solving the system of linear equations. For larger systems, there are various methods of learning v_π.

Gridworld with Random Policy

Example: 5×5 Grid.
Possible actions: u, l, d, r

$$\text{Rewards: } R = \begin{cases} -1 & \text{for bumping into the walls,} \\ 10 & \text{for jump from A to A',} \\ 5 & \text{for jump from B to B',} \\ 0 & \text{else} \end{cases}$$

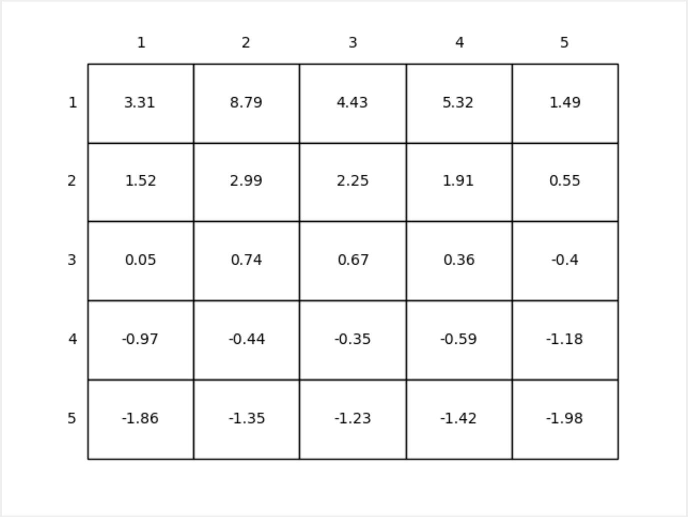
Policy: Choose action from uniform distribution.



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Gridworld with Random Policy

Compare with code from github: Same results.



	1	2	3	4	5
1	3.31	8.79	4.43	5.32	1.49
2	1.52	2.99	2.25	1.91	0.55
3	0.05	0.74	0.67	0.36	-0.4
4	-0.97	-0.44	-0.35	-0.59	-1.18
5	-1.86	-1.35	-1.23	-1.42	-1.98

Optimal Policies and Optimal Value Functions

Optimal Policies

Def.: A policy π is better than π' if $v_\pi(s) \geqslant v_{\pi'}(s)$ for all s. An optimal policy π_* is better than or equal to all others. There is always at least one such.

Optimal policies are sharing the same optimal action value function $q_*(s,a) = \max_\pi q_\pi(s,a)$ for all a, s. (unique!)
For the state action pair (s,a) this function gives the expected return for taking action a in state s and thereafter following an optimal policy:

$$q_*(s,a) = \mathbb{E}\big[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\big]$$

Bellman Optimality Equation

Bellman equation for v_* is called Bellman optimality equation:

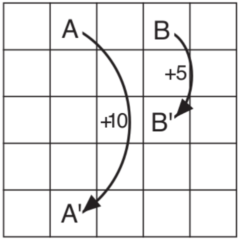
$$\begin{aligned}
v_*(s) &= \max_a q_{\pi_*}(s,a) && (5)\\
&= \max_a \mathbb{E}_{\pi_*}\big[G_t | S_t = s, A_t = a\big] && (6)\\
&= \max_a \mathbb{E}_{\pi_*}\big[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a\big] && (7)\\
&= \max_a \mathbb{E}\big[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\big] && (8)\\
&= \max_a \sum_{s',r} p(s', r|s,a)\big[r + \gamma v_*(s')\big] && (9)
\end{aligned}$$

Bellman optimality equation for q_*:

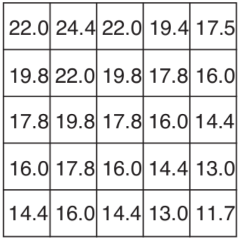
$$\begin{aligned}
q_*(s,a) &= \mathbb{E}\big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a\big] && (10)\\
&= \sum_{s',r} p(s', r|s,a)\big[r + \gamma \max_{a'} q_*(s', a')\big] && (11)
\end{aligned}$$

Gridworld with Optimal Policy

The Bellman Optimality Equation is again just a system of linear equations. Solved Bellman equation for v_* in the Gridworld Example: The value function is unique, the policy isn't.

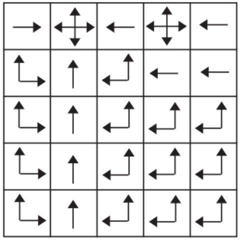


Gridworld



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

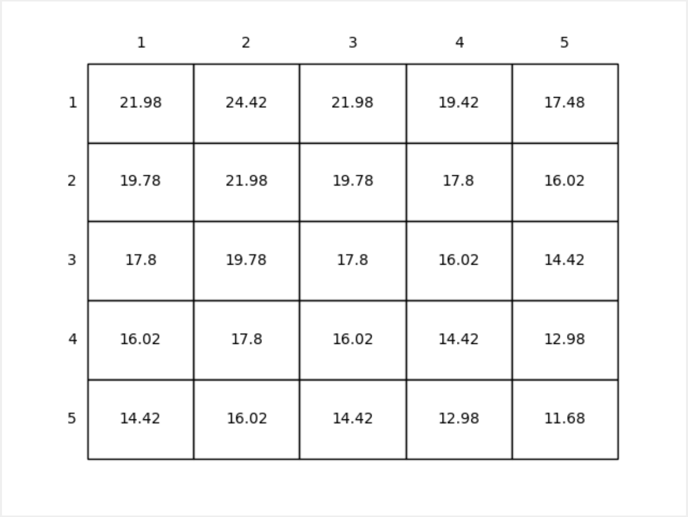
v_*



π_*

Gridworld with Optimal Policy

Compare with code from github: Same results.



	1	2	3	4	5
1	21.98	24.42	21.98	19.42	17.48
2	19.78	21.98	19.78	17.8	16.02
3	17.8	19.78	17.8	16.02	14.42
4	16.02	17.8	16.02	14.42	12.98
5	14.42	16.02	14.42	12.98	11.68

Recap: Finite MDP

Finite MDP: Finite set of states, actions and rewards together with well defined transition probabilities, depending on the current state and the taken action. The current state holds all information about the past (Markov Property).
Policy: Tells the agent which action to take in which state.
Return: Estimation of the future rewards.
(Action) Value function: Assign estimated Returns to states (action state pairs).
Bellman Equation: Recursive relation of the value function, can be solved as system of linear equations.

What was not in this chapter?
Strategies to arrive at an optimal policy!
Before: Update rules for Tic Tac Toe and Multiarmed Bandit were strategies to solve the Bellman Equation and arrive at an optimal policy at the same time.

Projects

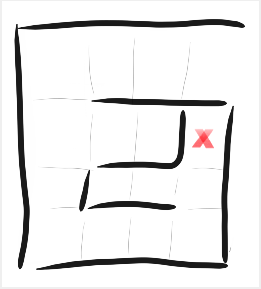
Labyrinth Escape

Agent starts at the red cross and needs to escape the labyrinth.

Possible actions: $\{U, R, D, L\}$

Rewards:

Bump into wall: $R = -1$
Escape: $R = 100$
Other actions: $R = 0$



Labyrinth Escape

ϵ-greedy method, with over episodes linearily decreasing $\epsilon \in [0, 1]$.

Q-Learning Algorithm

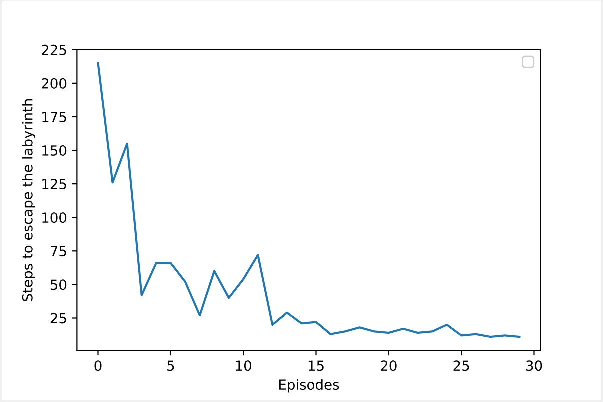
Define q-table containing a field for every action-state value.

Update action-state values after every greedy action:
$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a)]$

α ... Learning Rate, γ ... Discounting Factor
Notice: The value update rule is actually the Bellman Equation.

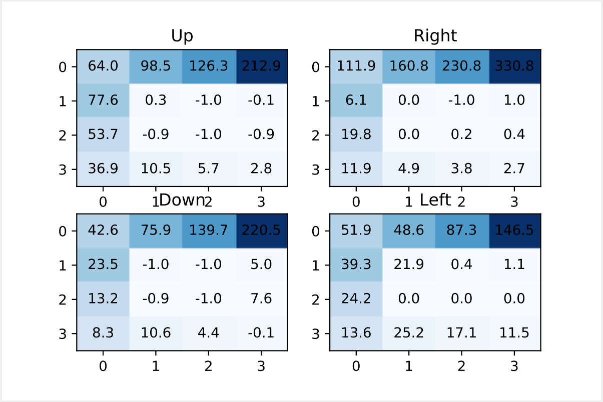
Labyrinth: Learning Curve

After 30 episodes the agent finds an optimal policy.



Labyrinth: Action Values

Action values, learned by the q learning algorithm.



Up

	0	1	2	3
0	64.0	98.5	126.3	212.9
1	77.6	0.3	-1.0	-0.1
2	53.7	-0.9	-1.0	-0.9
3	36.9	10.5	5.7	2.8

Right

	0	1	2	3
0	111.9	160.8	230.8	330.8
1	6.1	0.0	-1.0	1.0
2	19.8	0.0	0.2	0.4
3	11.9	4.9	3.8	2.7

Down

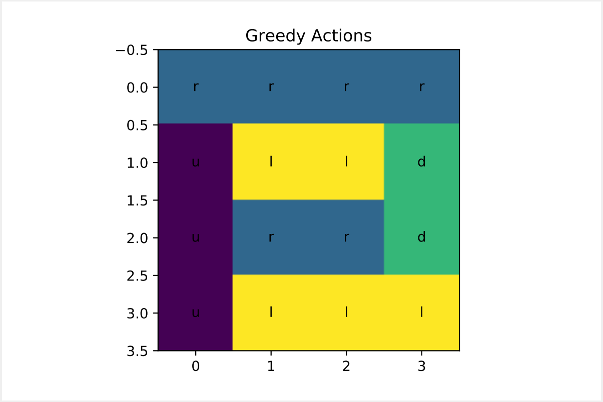
	0	1	2	3
0	42.6	75.9	139.7	220.5
1	23.5	-1.0	-1.0	5.0
2	13.2	-0.9	-1.0	7.6
3	8.3	10.6	4.4	-0.1

Left

	0	1	2	3
0	51.9	48.6	87.3	146.5
1	39.3	21.9	0.4	1.1
2	24.2	0.0	0.0	0.0
3	13.6	25.2	17.1	11.5

Labyrinth: Greedy Directions

Greedy actions for every state.



Travelling Salesman Problem (ongoing work)

A Salesman wants to visit certain cities (e.g. all capitital cities in austria) while driving as less kilometers as possible.

Set Startpoint and wanted endpoint.
Define Q-table for action-state values.
Consider Markov Property!
State = { current city , cities already visited }

Tschechien
Deutschland
Slowakei
Linz
Krems an der Donau
Klosterneuburg
St. Pölten
Wien
Wels
Oberösterreich
Amstetten
Niederösterreich
Steyr
Baden
Wiener Neustadt
Eisenstadt
Salzburg
Hallein
Kufstein
Wörgl
Bregenz
Dornbirn
Feldkirch
Vorarlberg
Innsbruck
Tirol
Salzburg
Kapfenberg
Leoben
Steiermark
Graz
Burgenland
Ungarn
Wolfsberg
Kärnten
Lienz
Klagenfurt am Wörthersee
Villach
Schweiz
Italien
Slowenien
2017 Copyright © KartePlan.com

Set Rewards:

$R(\text{A to B}) = -(\text{distance in kilometers})$

$R(\text{arriving at endponit when all cities visited}) = 10000$

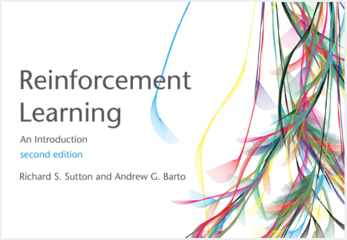
Update Rule as above.

Travelling Salesman Problem (ongoing work)

to be continued ... (on github and in the paper!)

References

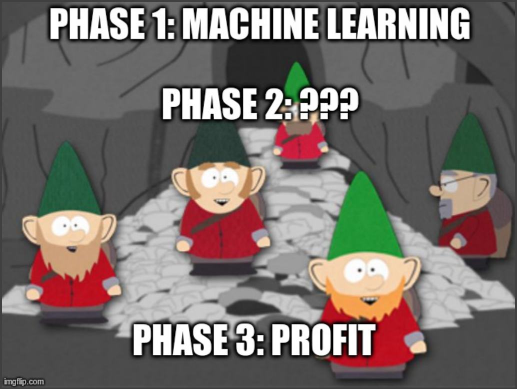
Sutton S. Barto & Andrew G. Barto, Reinforcement Learning: An Introduction



Implementation of Sutton & Barto's examples on github: github.com/ShangtongZhang/reinforcement-learning-an-introduction

Presentation and Implementation of Labyrinth & Travelling Salesman on github: github.com/fewagner/ReinforcementLearning

Presentation template from http://www.LaTeXTemplates.com (Pasquale Africa), some icons made by Freepik from www.flaticon.com, some pictures taken from Wikipedia



Questions?