Advanced Machine Learning
Summer 2019

Part 5 – Deep Reinforcement Learning
17.04.2019

Jonathon Luiten

Prof. Dr. Bastian Leibe

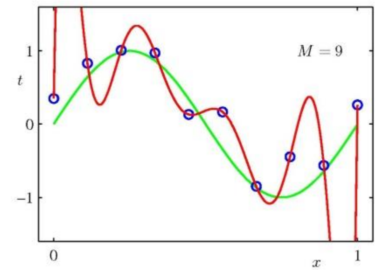
RWTH Aachen University, Computer Vision Group
http://www.vision.rwth-aachen.de

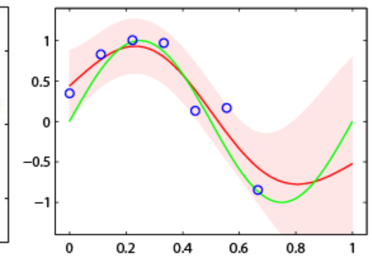
Visual Computing Institute | RWTH AACHEN UNIVERSITY

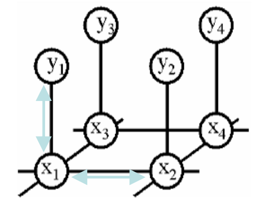
Course Outline

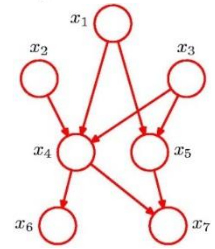
- Regression Techniques
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
- Deep Reinforcement Learning
- Probabilistic Graphical Models
  - Bayesian Networks
  - Markov Random Fields
  - Inference (exact & approximate)
- Deep Generative Models
  - Generative Adversarial Networks
  - Variational Autoencoders

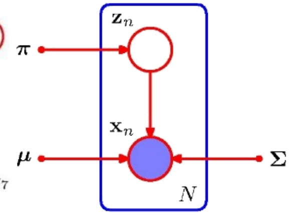
$f : \mathcal{X} \to \mathbb{R}$

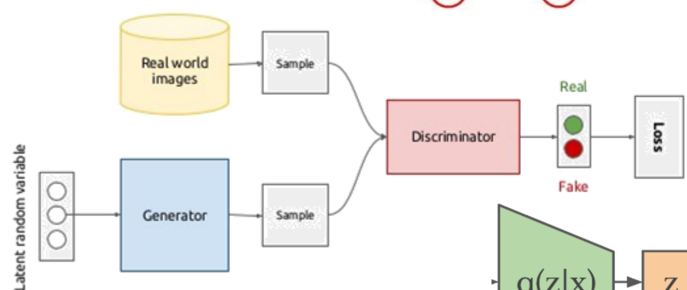


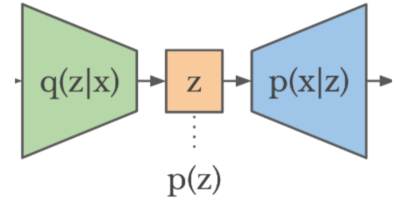












Jonathon Luiten
Visual Computing Institute | Prof. Dr . Bastian Leibe

Visual Computing Institute

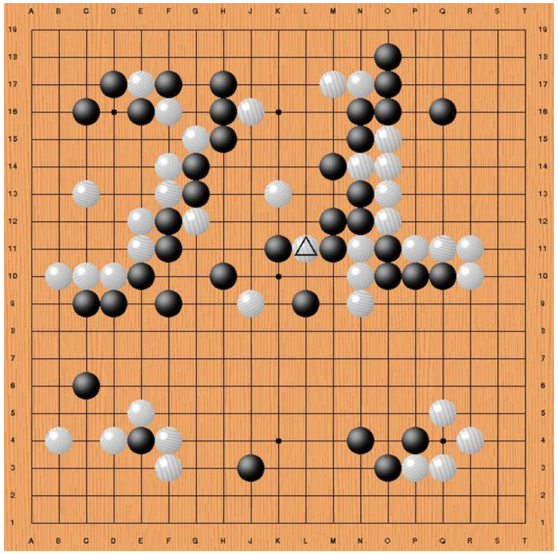
RWTH AACHEN UNIVERSITY

Topics of These Lectures

- Reinforcement Learning
  - Introduction
  - Key Concepts
  - Optimal policies
  - Exploration-exploitation trade-off
- Temporal Difference Learning
  - SARSA
  - Q-Learning
- Deep Reinforcement Learning
  - Value based Deep RL
  - Policy based Deep RL
  - Model based Deep RL
- Applications

What is Reinforcement Learning?

- Learning how to act from a reinforcement signal.
- Humans do this too.
- And it works: Atari games, Alpha Go, Dota2/Starcraft, Drone Control, Robot Arm Manipulation, etc.

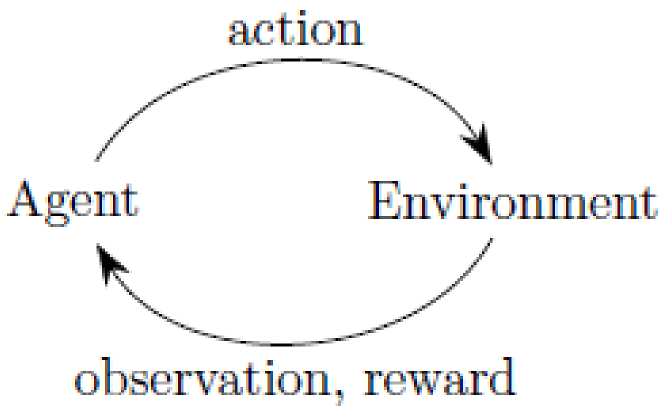


Jonathon Luiten
Visual Computing Institute | Prof. Dr . Bastian Leibe

Visual Computing Institute
RWTH AACHEN UNIVERSITY

Reinforcement Learning

- Motivation
  - General purpose framework for decision making.
  - Basis: Agent with the capability to interact with its environment
  - Each action influences the agent's future state.
  - Success is measured by a scalar reward signal.
  - Goal: select actions to maximize future rewards.



  - Formalized as a partially observable Markov decision process (POMDP)

Slide adapted from: David Silver, Sergey Levine

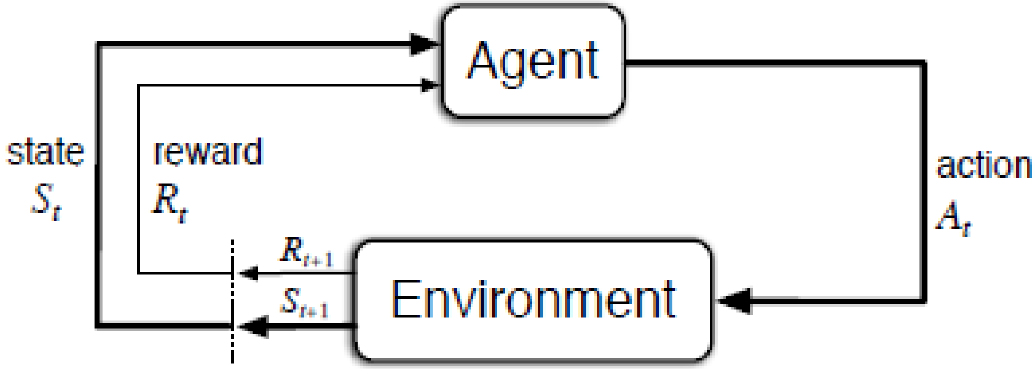
Reinforcement Learning

- Differences to other ML paradigms
  - There is no supervisor, just a reward signal
  - Feedback is delayed, not instantaneous
  - Time really matters (sequential, non i.i.d. data)
  - Agent’s actions affect the subsequent data it receives

$\Rightarrow$ *We don’t have full access to the function we’re trying to optimize, but must query it through interaction.*

Slide adapted from: David Silver, Sergey Levine

The Agent–Environment Interface



- Let's formalize this
  - Agent and environment interact at discrete time steps $t = 0, 1, 2, \ldots$
  - Agent observes state at time t: $S_t \in \mathcal{S}$
  - Produces an action at time t: $A_t \in \mathcal{A}(S_t)$
  - Gets a resulting reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$
  - And a resulting next state: S_{t+1}

Slide adapted from: Sutton & Barto

Note about Rewards

- Reward
  - At each time step t, the agent receives a reward R_{t+1}
  - This is the training signal
  - Provides a measure for the consequences of actions
  - Reward may be obtained only after a long sequence of actions
  - Goal: choose actions to maximize future accumulated reward.

- Important note
  - We need to provide those rewards to truly indicate what we want the agent to accomplish.
  - E.g., learning to play chess:
    - The agent should only be rewarded for winning the game.
    - Not for taking the opponent's pieces or other subgoals.
    - Else, the agent might learn a way to achieve the subgoals without achieving the real goal.

$\Rightarrow$ *This means, non-zero rewards will typically be very rare!*

Reward vs. Return

- Objective of learning
  - We seek to maximize the expected return G_t as some function of the reward sequence $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$
  - Standard choice: expected discounted return

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

  where $0 \leq \gamma \leq 1$ is called the discount rate.

- Difficulty
  - We don't know which past actions caused the reward.

  $\Rightarrow$ Temporal credit assignment problem

Markov Decision Process (MDP)

- Markov Decision Processes
  - We consider decision processes that fulfill the Markov property.
  - I.e., where the environments response at time t depends only on the state and action representation at t.

- To define an MDP, we need to specify
  - State and action sets
  - One-step dynamics defined by state transition probabilities

$$p(s'|s,a) = \Pr\{S_{t+1} = s' | S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a)$$

  - Expected rewards for next state-action-next-state triplets

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

Policy

- Definition
  - A policy determines the agent's behavior
  - Map from state to action $\pi: \mathcal{S} \rightarrow \mathcal{A}$

- Two types of policies
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \Pr\{A_t = a | S_t = s\}$

- Note
  - $\pi(a|s)$ denotes the probability of taking action a when in state s.

Jonathon Luiten
Visual Computing Institute | Prof. Dr . Bastian Leibe

Value Function

- Idea
  - Value function is a prediction of future reward
  - Used to evaluate the goodness/badness of states
  - And thus to select between actions

- Definition
  - The value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter.

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

  - The value of taking action a in state s under a policy π, denoted $q_\pi(s,a)$, is the expected return starting from s, taking action a, and following π thereafter.

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

Bellman Equation

- Recursive Relationship
  - For any policy π and any state s, the following consistency holds

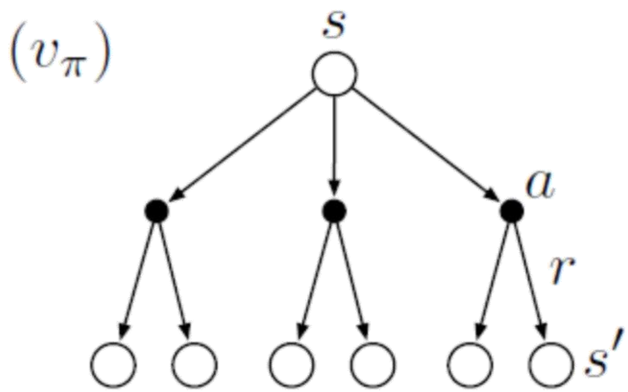
$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \middle| S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \middle| S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a)[r + \gamma v_\pi(s')], \qquad \forall s \in \mathcal{S}
\end{aligned}
$$

  - This is the Bellman equation for $v_\pi(s)$.

Bellman Equation

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \qquad \forall s \in \mathcal{S}$$

- Interpretation
  - Think of looking ahead from a state to each successor state.



  - The Bellman equation states that *the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way*.
  - We will use this equation in various forms to learn $v_\pi(s)$.

Jonathon Luiten
Visual Computing Institute | Prof. Dr . Bastian Leibe

Optimal Value Functions

- For finite MDPs, policies can be partially ordered
  - There will always be at least one optimal policy π_*.
  - The optimal state-value function is defined as

$$v_*(s) = \max_{\pi} \mathrm{v}_{\pi}(\mathrm{s})$$

  - The optimal action-value function is defined as

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

Optimal Value Functions

- Bellman optimality equations
  - For the optimal state-value function v_*:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a)$$

$$= \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

  - v_* is the unique solution to this system of nonlinear equations.
  - For the optimal action-value function q_*:

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$$

  - q_* is the unique solution to this system of nonlinear equations.

$\Rightarrow$ If the dynamics of the environment $p(s',r|s,a)$ are known, then in principle one can solve those equation systems.

Optimal Policies

- Why optimal state-value functions are useful
  - *Any policy that is greedy w.r.t. v_* is an optimal policy.*

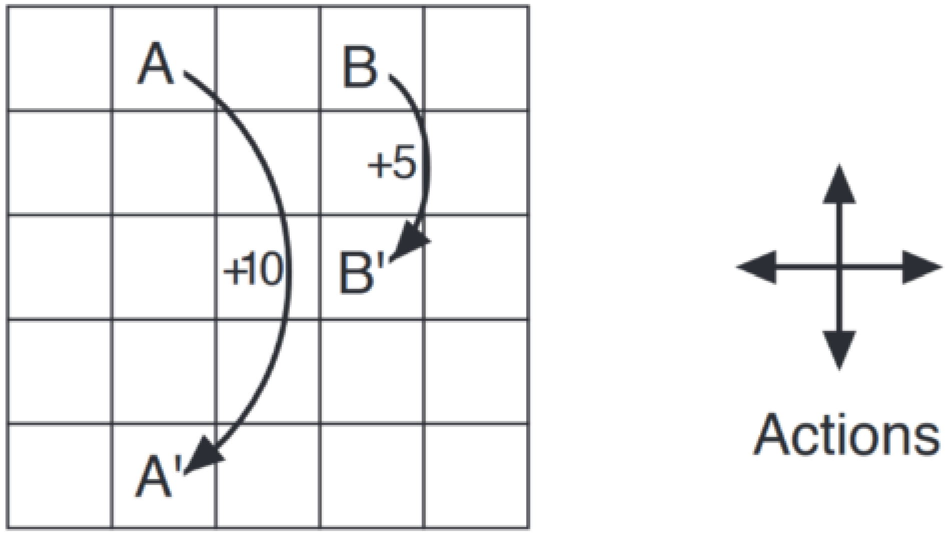
    $\Rightarrow$ Given v_*, one-step-ahead search produces the long-term optimal results.

    $\Rightarrow$ Given q_*, we do not even have to do one-step-ahead search

$$\pi_*(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}}\, q_*(s, a)$$

- Challenge
  - Many interesting problems have too many states for solving v_*.
  - Many Reinforcement Learning methods can be understood as approximately solving the Bellman optimality equations, using actually observed transitions instead of the ideal ones.

Example

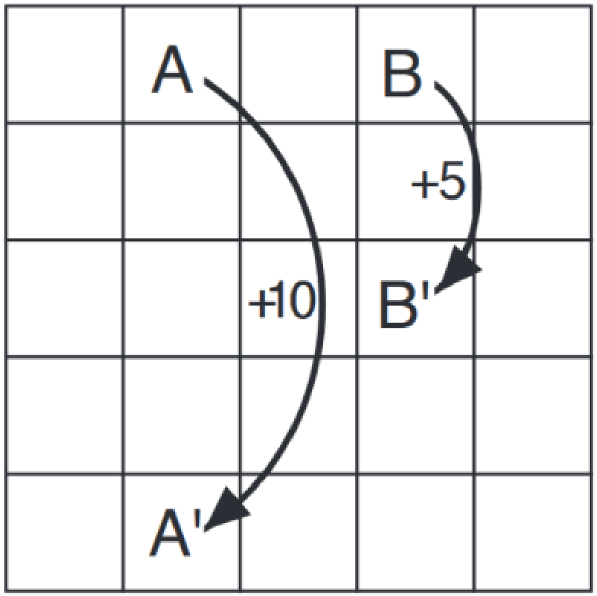


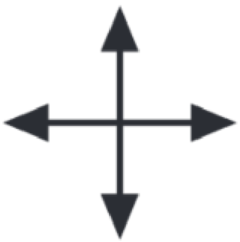
- Let‘s assume the following MDP
  - 4 actions: up, down, left, right
  - Deterministic state transitions on actions, but
    - Move from A / B transitions to A‘ / B‘ respectively
    - Move into border of the grid moves back to current location
  - Reward:
    - -1 for moving into border of the grid
    - +10 / +5 after transition from A / B respectively

Figure source: Sutton and Barto, 2012

Policy 1

- Value function for a policy that takes each action with equal probability ($\gamma = 0.9$)



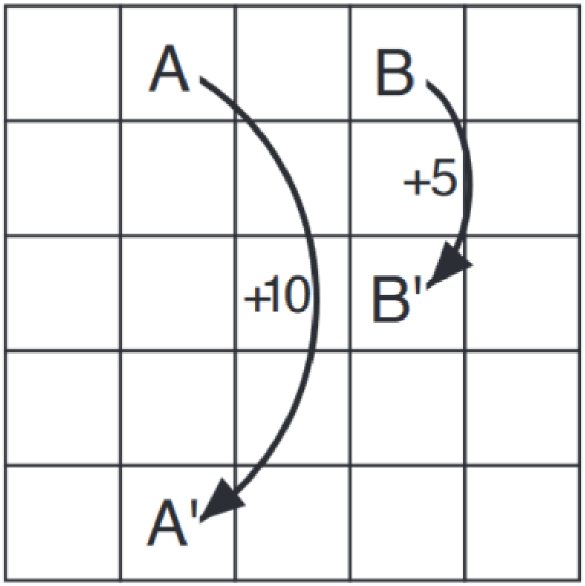


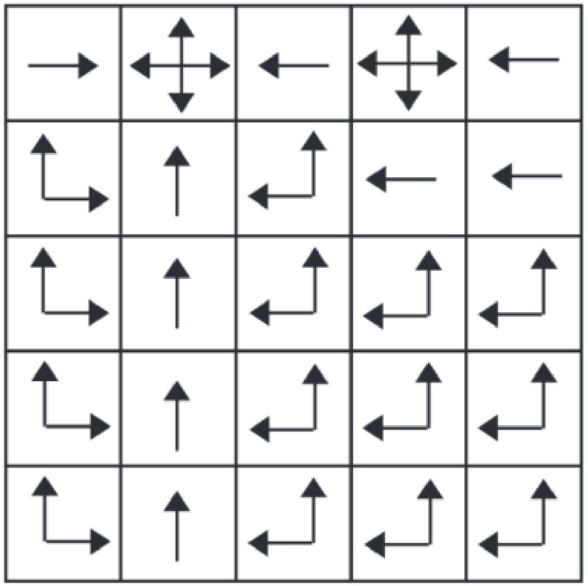
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Figure source: Sutton and Barto, 2012

Policy 2

- Optimal value function and policy for the grid world





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

Figure source: Sutton and Barto, 2012

Tabular vs. Approximate methods

- For problems with small discrete state and action spaces:
  - Value function or Policy function can be expressed as a table of values.
- If we cannot enumerate our states or actions we use function approximation.
  - Kernel methods
  - Deep Learning / Neural Networks
- Want to solve large problems with huge state spaces, e.g. chess: 10^{120} states.
- Tabular methods don't scale well - they're a lookup table
  - Too many states to store in memory
  - Too slow to learn value function for every state/state-action.

Model-based vs Model-free

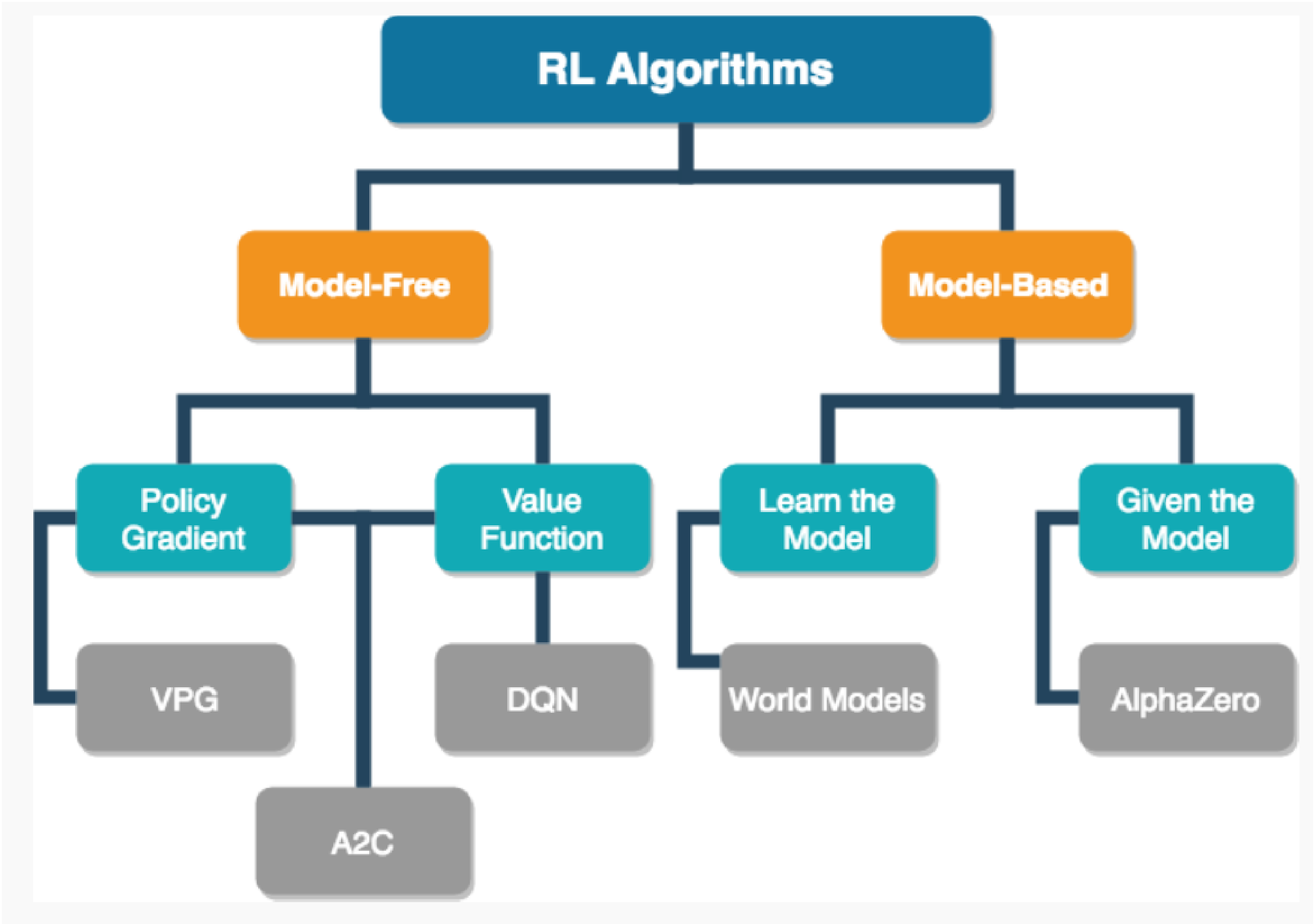
- Model-based
  - Has a model of the environment dynamics and reward
  - Allows agent to plan: predict state and reward before taking action
  - Pro: Better sample efficiency
  - Con: Agent only as good as the environment - Model-bias

- Model-free
  - No explicit model of the environment dynamics and reward
  - Less structured. More popular and further developed and tested.
  - Pro: Can be easier to implement and tune
  - Cons: Very sample inefficient

Value-based RL vs Policy-based RL

- RL methods can directly estimate a policy: Policy Based
  - A direct mapping of what action to take in each state.
  - $\pi(a|s) = \mathrm{P}(\mathrm{a}|\mathrm{s}, \theta)$
- RL methods can estimate a value function and derive a policy from that: Value Based
  - Either a state-value function
    - $\hat{V}(s;\theta) \approx V^{\pi}(\mathrm{s})$
  - Or an action-state value function (q function)
    - $\hat{Q}(s, a;\theta) \approx Q^{\pi}(\mathrm{s}, \mathrm{a})$
- Or both simultaneously: Actor-Critic
  - Actor-Critic methods learn both a policy (actor) and a value function (critic)

Taxonomy of RL methods

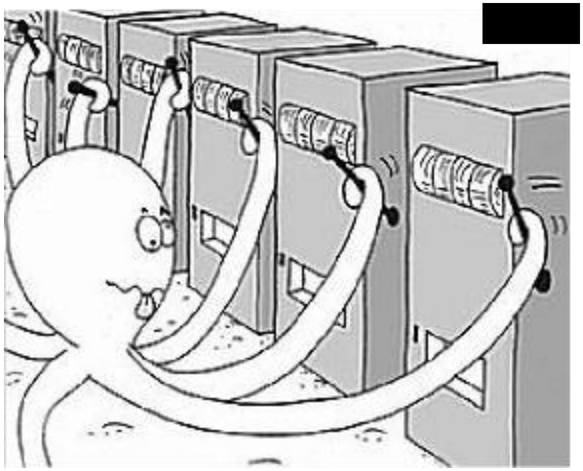


Jonathon Luiten
Visual Computing Institute | Prof. Dr . Bastian Leibe

Slide Credit: Zac Kenton

Exploration-Exploitation Trade-off

- Example: N-armed bandit problem
  - Suppose we have the choice between N actions $a_1, \ldots, a_N$.
  - If we knew their value functions $q_*(s, a_i)$, it would be trivial to choose the best.
  - However, we only have estimates based on our previous actions and their returns.



- We can now
  - Exploit our current knowledge
    - And choose the greedy action that has the highest value based on our current estimate.
  - Explore to gain additional knowledge
    - And choose a non-greedy action to improve our estimate of that action's value.

Jonathon Luiten
Visual Computing Institute | Prof. Dr . Bastian Leibe

Image source: research.microsoft.com

Simple Action Selection Strategies

- ϵ-greedy
  - Select the greedy action with probability $(1-\epsilon)$ and a random one in the remaining cases.

  $\Rightarrow$ In the limit, every action will be sampled infinitely often.

  $\Rightarrow$ Probability of selecting the optimal action becomes $> (1-\epsilon)$.
  - But: many bad actions are chosen along the way.

- Softmax
  - Choose action a_i at time t according to the softmax function

$$\frac{e^{q_t(a_i)/\tau}}{\sum_{j=1}^{N} e^{q_t(a_j)/\tau}}$$

    where τ is a temperature parameter (start high, then lower it).
  - Generalization: replace q_t by a preference function H_t that is learned by stochastic gradient ascent ("gradient bandit").

On-Policy vs. Off-Policy

- On-policy methods
  - Attempt to evaluate or improve the policy used to make decisions.
  - “Learn while on the job”

- Off-policy methods
  - Policy used to generate behavior (behavior policy) is unrelated to the policy that is evaluated and improved (estimation policy)
  - Can we learn the value function of a policy given only experience “off” the policy?
  - “Learn while looking over someone else’s shoulder”

Topics of These Lectures

- Reinforcement Learning
  - Introduction
  - Key Concepts
  - Optimal policies
  - Exploration-exploitation trade-off
- Temporal Difference Learning
  - SARSA
  - Q-Learning
- Deep Reinforcement Learning
  - Value based Deep RL
  - Policy based Deep RL
  - Model based Deep RL
- Applications

Policy Evaluation

- Policy evaluation (the prediction problem)
  - How good is a given policy?
  - For a given policy π, compute the state-value function v_π.
  - Once we know how good a policy is, we can use this information to improve the policy

- If we know the model:
  - $V_{k+1}^{\pi}(s_t) =$
  $$\sum_{a_t} \pi(a_t \mid s_t) \sum_{s_{t+1}} p(s_{t+1} \mid s_t, a_t) \left(r(s_t, a_t, s_{t+1}) + \gamma V_k^{\pi}(s_{t+1}) \right)$$
  - This can be shown to converge to the actual V^π as $K \to \infty$

Policy Evaluation

- If we do not know the model, then we have to approximate it using observations
- One option: Monte-Carlo methods
  - Play through a sequence of actions until a reward is reached, then backpropagate it to the states on the path.
  - Update after whole sequence (episodic)

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$ Target: the actual return after time t

- Or: Temporal Difference Learning (TD Learning) – TD(λ)
  - Directly perform an update using the estimate $V(S_{t+\lambda+1})$.
  - Bootstraps the current estimate of the value function
  - Can update every step

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

Target: an estimate of the return (here: TD(0))

Jonathon Luiten
Visual Computing Institute | Prof. Dr . Bastian Leibe

SARSA: On-Policy TD Control

- Idea
  - Turn the TD idea into a control method by always updating the policy to be greedy w.r.t. the current estimate

- Procedure
  - Estimate $q_\pi(s,a)$ for the current policy π and for all states s and actions a.
  - TD(0) update equation

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

  - This rule is applied after every transition from a nonterminal state S_t.
  - It uses every element of the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.
    $\Rightarrow$ *Hence, the name SARSA*.

Jonathon Luiten
Visual Computing Institute | Prof. Dr . Bastian Leibe

Image source: Sutton & Barto

SARSA: On-Policy TD Control

- Algorithm

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action a, observe r, s'
        Choose a' from s' using policy derived from Q (e.g., ε-greedy)
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    until s is terminal

Image source: Sutton & Barto

Q-Learning: Off-Policy TD Control

- Idea
  - Directly approximate the optimal action-value function q_*, independent of the policy being followed.

- Procedure
  - TD(0) update equation

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

  - Dramatically simplifies the analysis of the algorithm.
  - All that is required for correct convergence is that all pairs continue to be updated.

Image source: Sutton & Barto

Q-Learning: Off-Policy TD Control

- Algorithm

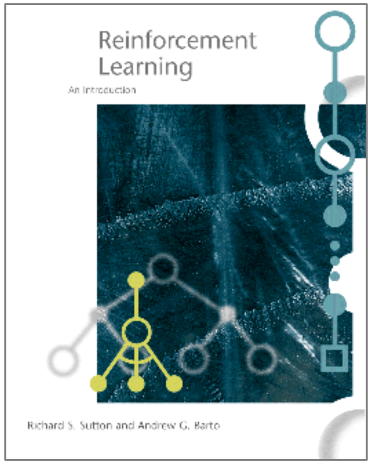
Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$;
    until s is terminal

Image source: Sutton & Barto

References and Further Reading

- More information on Reinforcement Learning can be found in the following book

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



- The complete text is also freely available online

https://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html

References and Further Reading

- DQN paper
  - www.nature.com/articles/nature14236

- AlphaGo paper
  - www.nature.com/articles/nature16961