Deep Reinforcement Learning for control, investigating the application to a trajectory planning problem

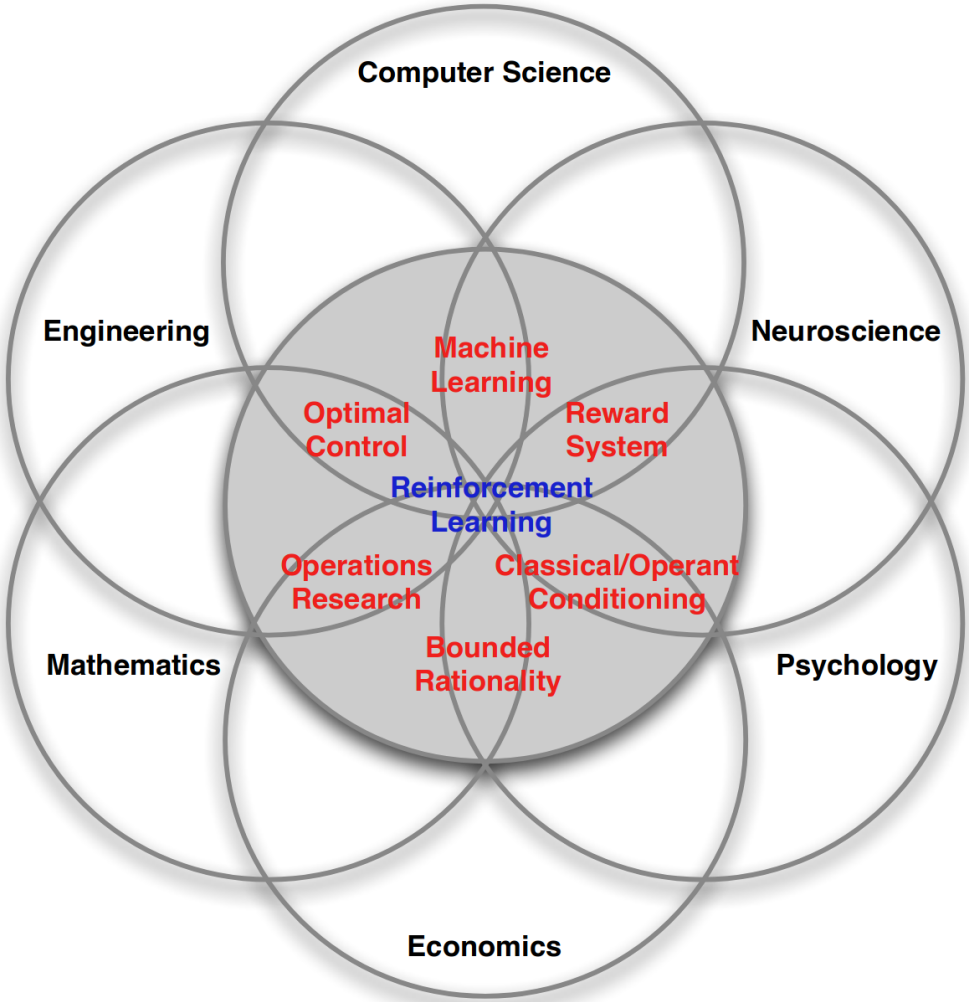
Lars Klein

RWTH

lars.klein@rwth-aachen.de

July 19, 2018

Orientation (Disorientation ?)

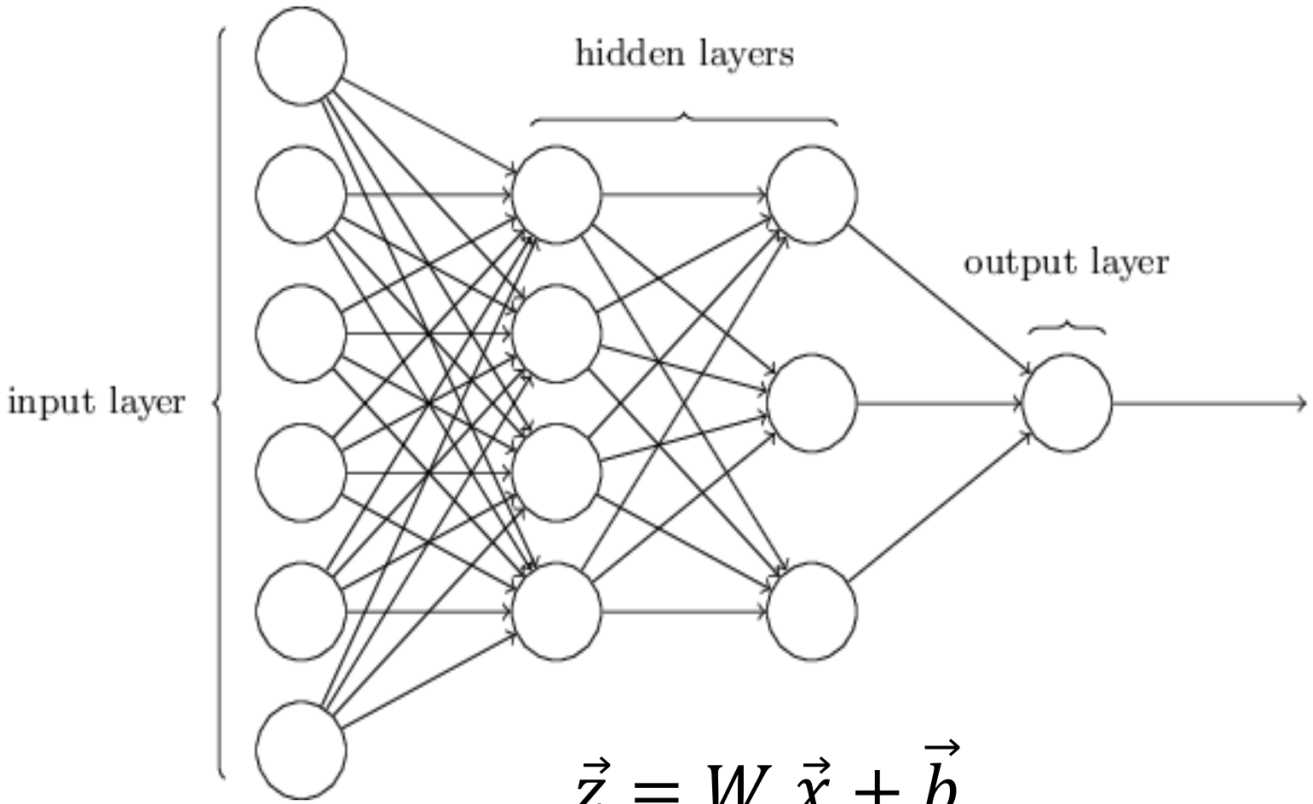


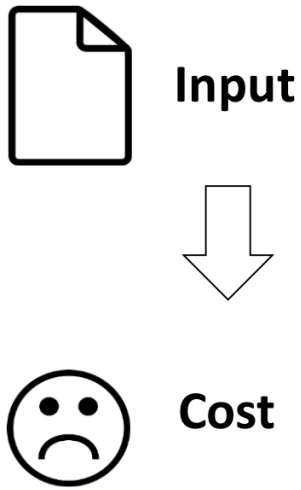
What is the scope of this presentation?

Reinforcement learning can be presented from many perspectives. An exhaustive discussion is beyond the scope of this presentation. Instead we will focus on

- A short intro to machine learning
- Description of a reinforcement learning problem
- Introduction to one of the key algorithms
- Demo of my simulation
- Discussion of results and problems

Intro to Machine Learning





Backpropagation

$$\vec{z} = W\,\vec{x} + \vec{b}$$
$$\vec{a} = \sigma(\vec{z})$$

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial f} \cdot \frac{\partial f}{\partial w}$$

Types of Machine Learning

Unsupervised

You are given only data. The goal is to find structure within this data. Typical tasks would be compression or outlier detection.

Supervised

You are given pairs of $(input, output)$. The goal is to learn a good prediction of outputs for new inputs. This is further divided into **Regression** *predictingcontinuousoutputs* and **Classification** *predictingdiscreteoutputs*.

Reinforcement learning

You are given a simulation that you can interact with, and a feedback signal that informs you how well you did. The goal is to explore behaviours and identify a strategy to maximize positive feedback.

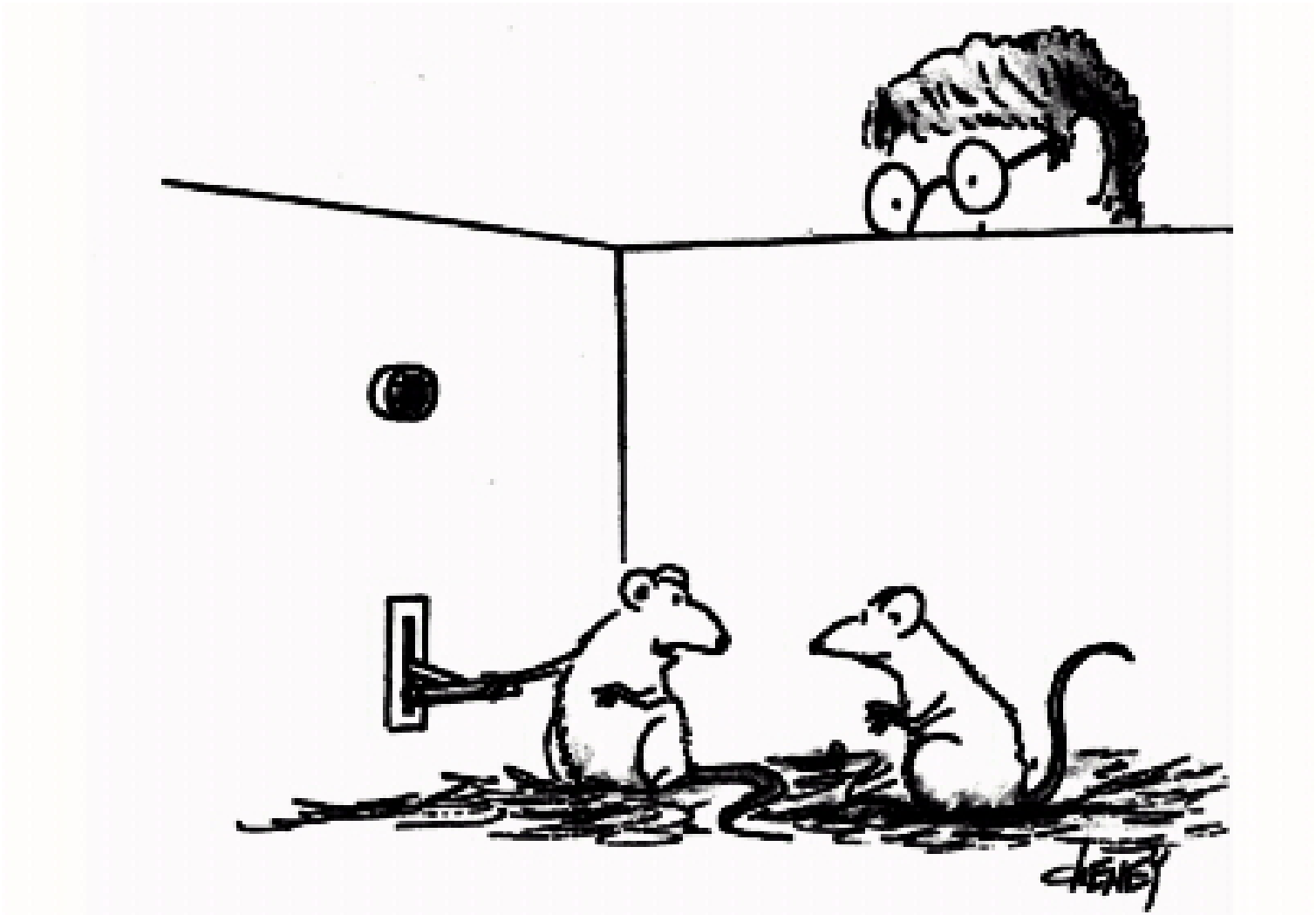
Challenges in reinforcement learning

The purpose of reinforcement learning is to predict values for behaviours.
But this is not just regression for reward signals:

- Data is created interactively.
- Reward can be sparse
- Reward can be delayed
- Reward can be non-deterministic
- State of the simulation can be very complex
- State of the simulation can be only partially observable

The reinforcement agent needs to discover viable strategies while exploring the simulation.
$\rightarrow$Simulation needs to be designed in tandem with the rl algorithm.

A typical RL agent



It's a rather interesting phenomenon. Every time I press this lever, that post-graduate student breathes a sigh of relief.

So how does it actually look ?

Definitions

An interaction with the environment: $S_t, a_t, r_t \rightarrow S_{t+1}$:

- $s_t \in S$ **state** (partial observation) at time t
- $a_t \in A$ **action** taken at time t
- r_t **reward** received at time t
- s_{t+1} state (partial observation) at time t+1
- $a_t \sim \pi(s_t) = P(a \in A | s_t \in S)$, actions are drawn from a distribution over the action space A, conditioned on the current state.
  this is called the **policy**

The reward r_t obtained at time t, as well as the next state s_{t+1} depend on the action a_t sampled from π.

Definitions

All interactions with the environment are based on a policy. We derive the following definitions:

- $v(s) = E_{\pi, s_t=s}(r_t) + \mu * E_{\pi, s_t=s}(v(s_{t+1})$, the **value** of a state.
  Defined as total expected future reward.
  A discount factor μ is introduced to avoid infinite values.
  The recursion stops at terminal states with a fixed value of 0.
- $Q(s, a) = E_{\pi, s_t=s, a_t=a}(v(s))$, **Q-values** are analogous to values, but action at time t is already given
- $A(s, a) = Q(s, a) - v(s)$, **advantage** of choosing action a at state s.

How to act ?

The goal is to maximize the total expected future reward:

- $\pi^\star$ is the ideal policy.
  $\rightarrow \quad a_t \sim \pi^\star(s_t)$.
- $Q^\star(a, s)$ are the Q-values associated with the ideal policy.
  $\rightarrow \quad a_t = argmax_a Q^\star(s_t, a)$
- $A^\star(a, s)$ are the advantages associated with the ideal policy.
  Please note: $A^\star <= 0$

For continous action spaces:

- π is a probability distribution, it translates nicely to the sampling of continous instead of discrete actions.
- acting according to the Q values becomes problematic:
  $\rightarrow \quad$ the argmax becomes a complex optimization problem.

Learning with policy gradients

Policy gradient methods predict:

- policy $\pi(s)$
- value $v(s)$

The value function is necessary for training the agent.
We use it to calculate an estimation of the advantage.
All notation here assumes empirical observations of rewards and states:

$$\delta_t = -v(s_t) + r_t + \gamma v(s_{t+1})$$

$$\hat{A}(s_0 = s, a_0 = 0) = \sum_{t=0}^{\inf} \delta_t (\mu\gamma)^t$$

If we collect enough such estimates, we can approximate the true advantage function (which was defined as an expectation).
Please note: we assume a given initial action.

Learning with policy gradients part 2

Positive advantage = action is better than current policy,
negative advantage = action is worse

$\rightarrow$ change the probability according to our advantage estimation.

In a machine learning setup:

- updates become training steps
- $\pi(s) = \pi_\theta(s)$,
  function approximator with parameters θ.
- estimate training gradients on empirical data:

$$\hat{g} = \hat{E}(\nabla_\theta(log(\pi_\theta(a_0, s_0))\, \hat{A}(a_0, s_0)$$

Learning with policy gradients part 3

In practice this leads to the following setup:

1. initialize random policy and random value prediction
2. interact with simulation and collect (lots of) observations
3. use observations to calculate empirical estimates, (advantages for each action)
4. train on empirical estimates $\rightarrow$ update policy and value estimates
5. back to step 2.

Learning with policy gradients part 4

Basically, we are repeating two steps:

- use π to get the value function $v = v^{\pi}$
- update the policy with the value function $\pi := greedy(v)$

This is supposed to guarantee convergence.
At least that's what they say here.

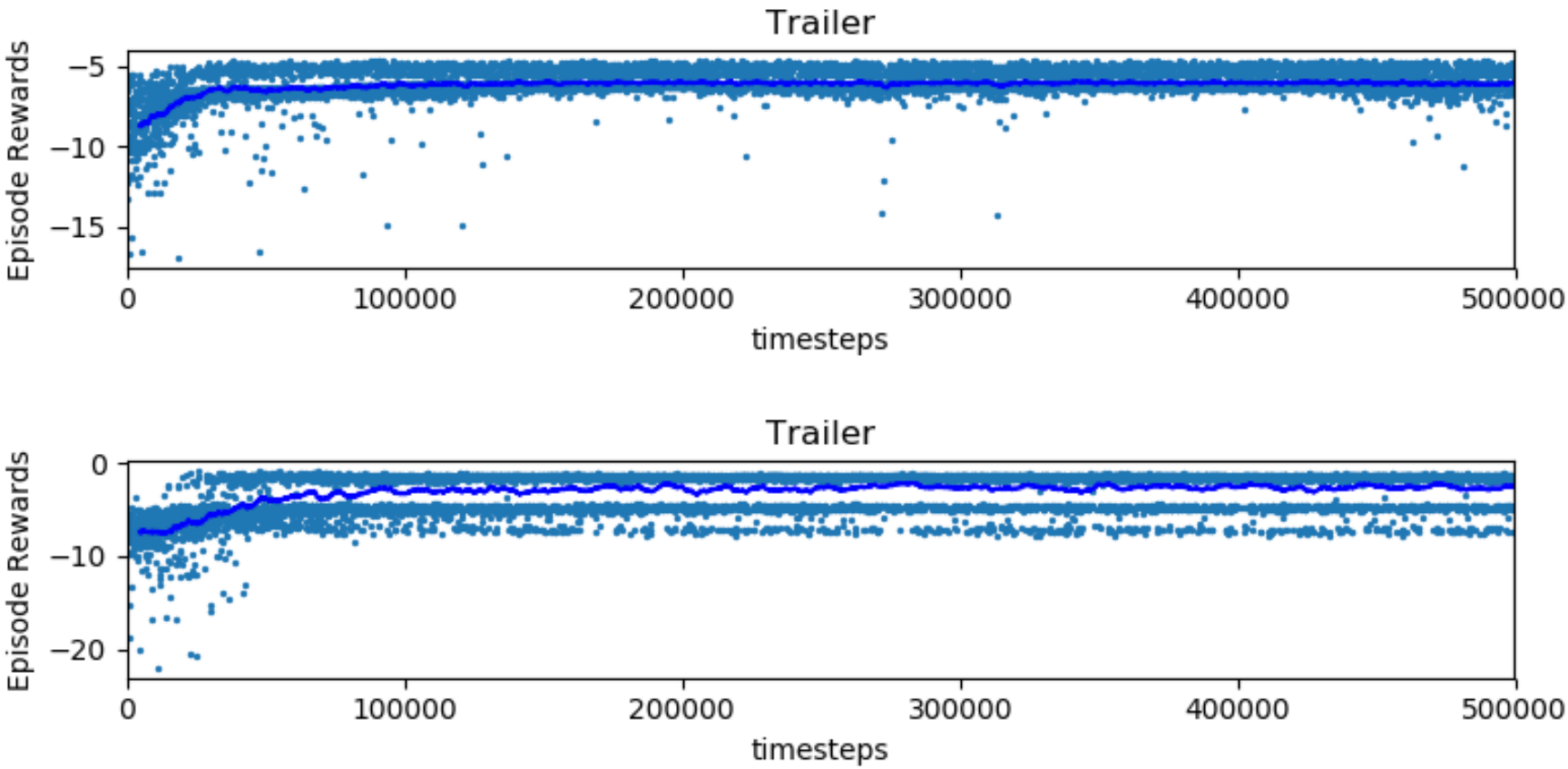
In practice, things look different ...

Demo

A concrete RL task.

Using OpenAI baseline algorithms

Two different reward setups:



Tuning

After reimplementing the algorithm and searching for better function approximators:

Questions ?

Sources for the graphics (click me):

- venn diagram on reinforcement learning
- rat comic
- the others are homemade