Lee Sedol playing Go against AlphaGo

AN INTRODUCTION TO REINFORCEMENT LEARNING

Guest lecture by Francis wyffels

GHENT UNIVERSITY

@fwyffels - francis.wyffels@UGent.be

CC BY SA

airplane

automobile

bird

cat

deer

dog

frog

horse

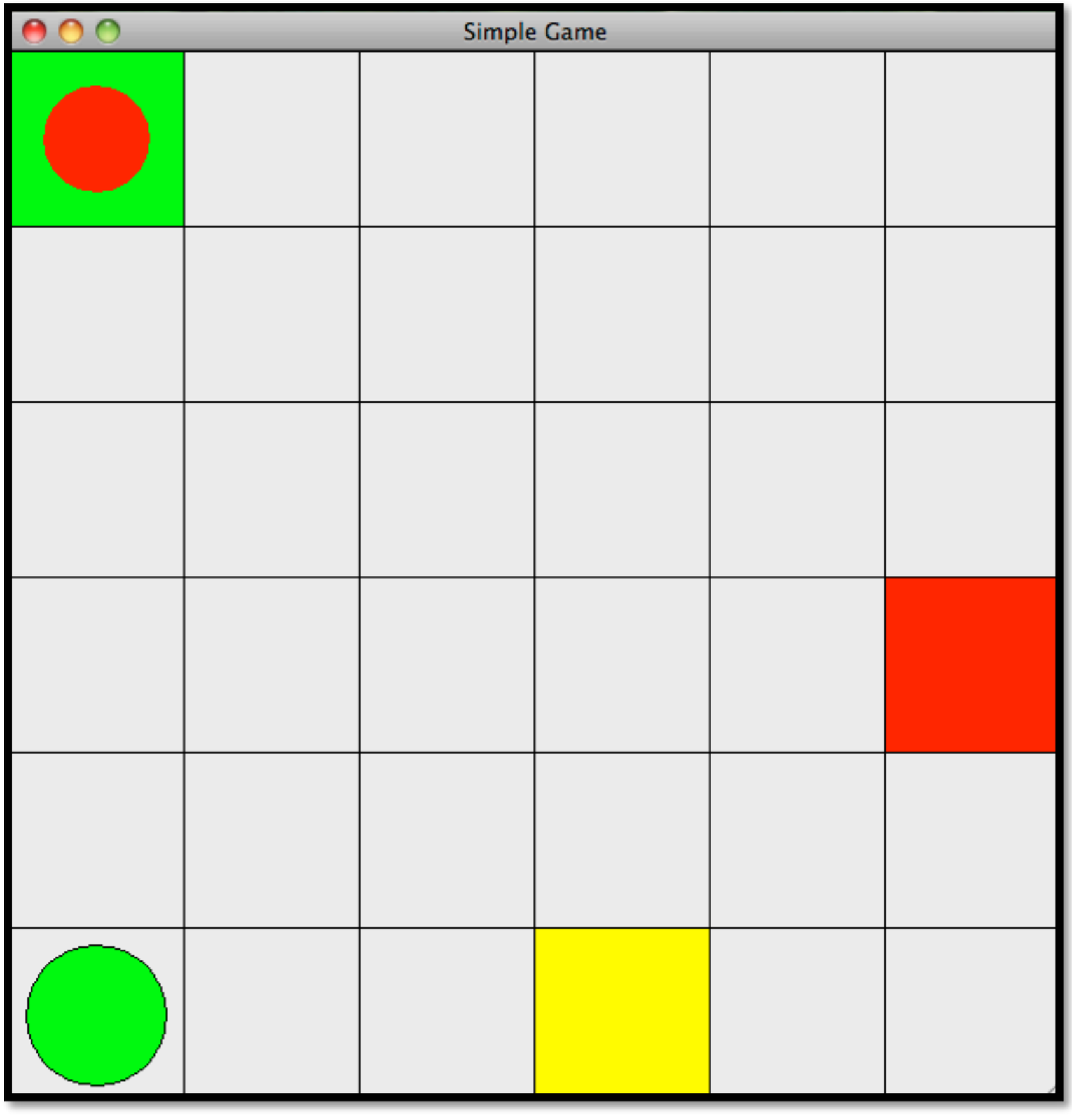
ship

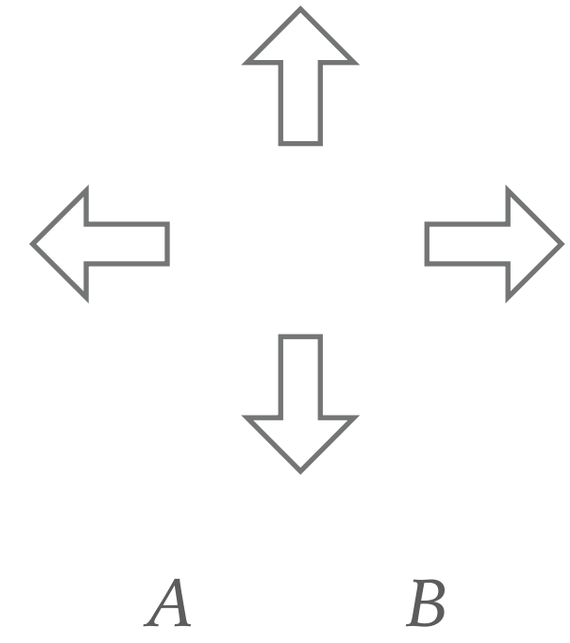
truck

[CIFAR10]

[Stanford University - AI]

LET'S PLAY A GAME





A *B*

> Reinforcement learning is an interesting solution for a class of problems where *no explicit solution* is available and for which a *control policy* has to be found in order to maximise the *cumulative reward*.

PART I

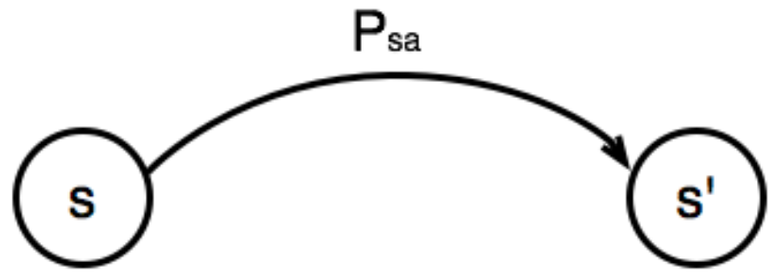
Discrete state space & discrete action space

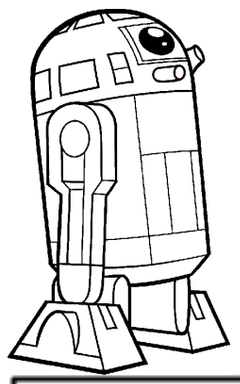
[Civilisation 2]

MARKOV DECISION PROCESSES

- Also known as MDP's
- Defined by (S, A, $\{P_{SA}\}$, γ, R)
  - ✓ S: sate
  - ✓ A: action
  - ✓ $\{P_{SA}\}$: state transition probability distribution
  - ✓ γ: discount factor
  - ✓ R: reward

$$\sum_{\forall s'} P_{sa}(s') = 1$$
$$P_{sa}(s') \geq 0$$





NORTH

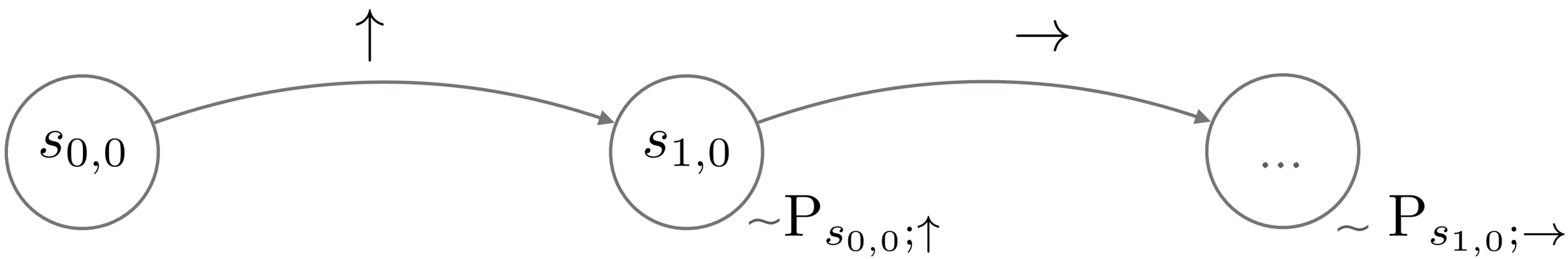
NORTH	80%
EAST	10%
WEST	10%

-0.02	-0.02	-0.02	+1
-0.02		-0.02	-1
START	-0.02	-0.02	-0.02

Possible actions: NORTH, EAST, SOUTH, WEST

[Russel & Norvig, Artificial Intelligence: a modern approach]

MARKOV DECISION PROCESSES: HOW IT WORKS



$$\mathrm{P}_{s_{0,0};\uparrow}(s_{1,0}) = 0.8$$

$$\mathrm{P}_{s_{0,0};\uparrow}(s_{0,1}) = 0.1$$

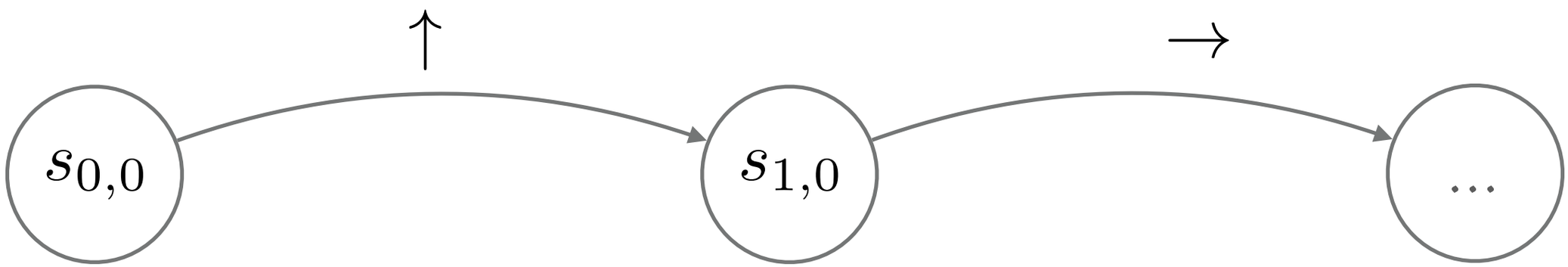
$$\mathrm{P}_{s_{0,0};\uparrow}(s_{0,0}) = 0.1$$

-0.02	-0.02	-0.02	+1
-0.02		-0.02	-1
START	-0.02	-0.02	-0.02

Possible actions: NORTH, EAST, SOUTH, WEST

[Russel & Norvig, Artificial Intelligence: a modern approach]

MARKOV DECISION PROCESSES: MAXIMISE THE TOTAL PAYOFF



$$\mathrm{R}(s_{0,0}) + \gamma \mathrm{R}(s_{1,0}) + \gamma^2 \mathrm{R}(s_{...}) + ...$$

Goal: choose actions to maximise the expected value of the total payoff!

$$\mathrm{E}\left(\mathrm{R}(s_{0,0}) + \gamma \mathrm{R}(s_{1,0}) + \gamma^2 \mathrm{R}(s_{...}) + ...\right)$$

For this we need to find a policy:

$$\Pi : S \rightarrow A$$

-0.02	-0.02	-0.02	+1
-0.02		-0.02	-1
START	-0.02	-0.02	-0.02

Possible actions: NORTH, EAST, SOUTH, WEST

[Russel & Norvig, Artificial Intelligence: a modern approach]

THE BELLMAN EQUATIONS

- Value function for a specific policy π

$$\mathrm{V}^{\pi}(s) = \mathrm{R}(s) + \gamma \sum_{\forall s'} \mathrm{P}_{s\pi(s)}(s')\mathrm{V}^{\pi}(s')$$

derived from to total expected payoff for a specific policy starting from a specific state!

- Optimal value function

$$\mathrm{V}^{*}(s) = \mathrm{R}(s) + \max_{a} \gamma \sum_{\forall s'} \mathrm{P}_{sa}(s')\mathrm{V}^{*}(s')$$

- Optimal policy

$$\pi^{*}(s) = \arg\max_{a} \sum_{\forall s'} \mathrm{P}_{sa}(s')\mathrm{V}^{*}(s')$$

VALUE ITERATION

1. Initialise $\mathrm{V}(s) = 0 \;\; \forall s$

2. For every s update

$$\mathrm{V}(s) = \mathrm{R}(s) + \max_a \gamma \sum_{\forall s'} \mathrm{P}_{sa}(s')\mathrm{V}(s')$$

3. Repeat until there are no changes

$$\mathrm{V}(s) \rightarrow \mathrm{V}^*(s)$$

Afterwards, get the optimal policy:

$$\pi^*(s) = \arg\max_a \sum_{\forall s'} \mathrm{P}_{sa}(s')\mathrm{V}^*(s')$$

POLICY ITERATION

1. Initialise policy π arbitrarily

2. Solve $\mathrm{V} := \mathrm{V}^{\pi}$

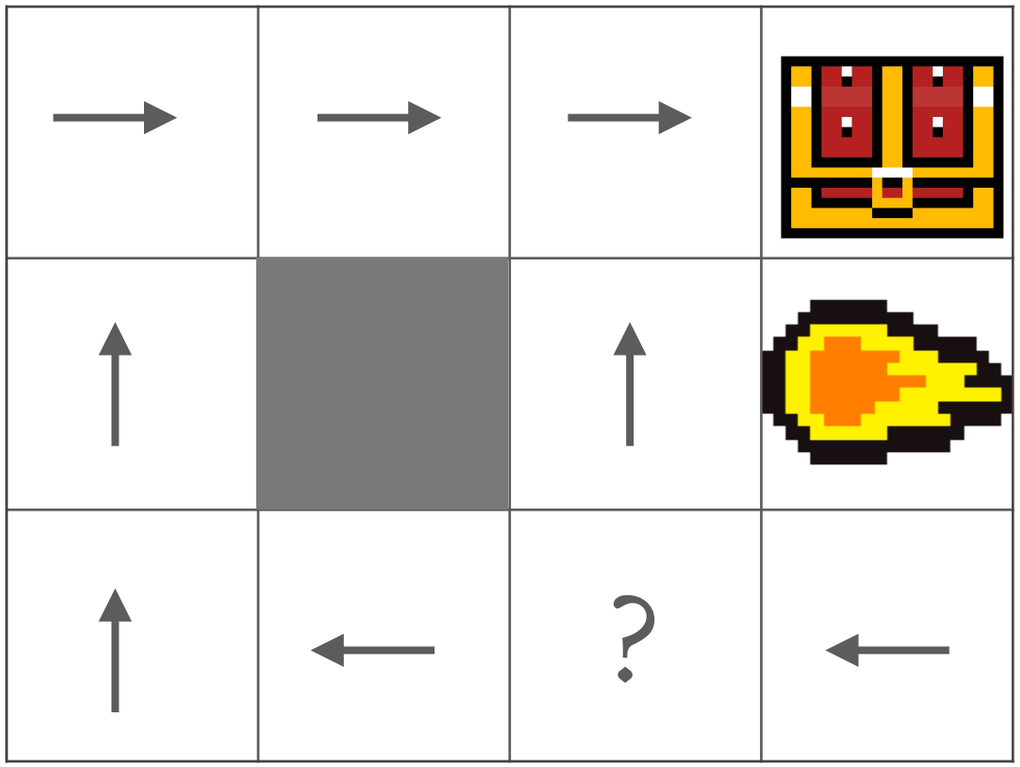
3. Let $\pi(s) := \arg\max_a \sum_{\forall s'} \mathrm{P}_{sa}(s')\mathrm{V}(s')$

4. Repeat until there are no changes

$$\mathrm{V}(s) \rightarrow \mathrm{V}^*(s) \qquad\qquad \pi(s) \rightarrow \pi^*(s)$$

SOME EXAMPLES

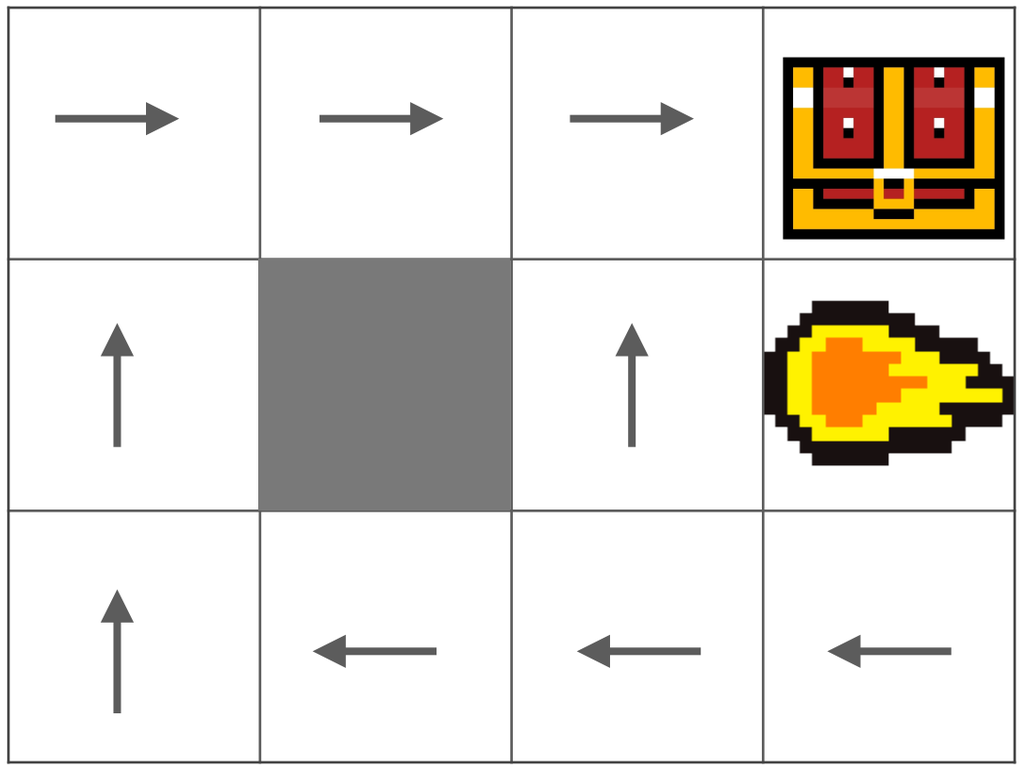
WHAT IS THE OPTIMAL ACTION?



0.86	0.90	0.93	+1
0.82		0.69	-1
0.78	0.75	0.71	0.49

- $P_{SA} \sim \{0.8, 0.1, 0.1\}$
- $\gamma = 0.99$

GOING WEST IS OPTIMAL

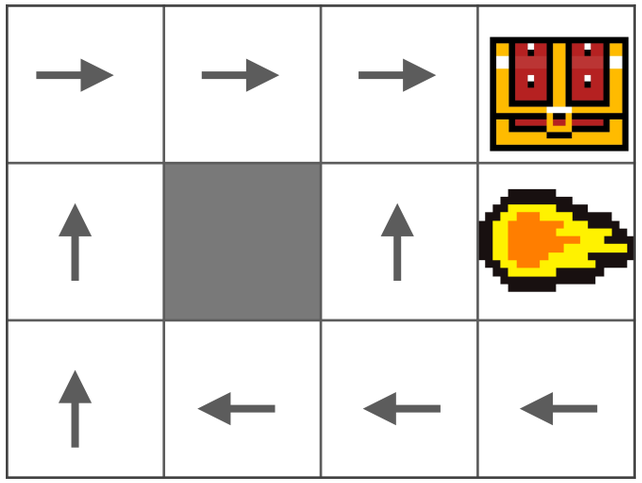


0.86	0.90	0.93	+1
0.82		0.69	-1
0.78	0.75	0.71	0.49

- $P_{SA} \sim \{0.8, 0.1, 0.1\}$
- $\gamma = 0.99$

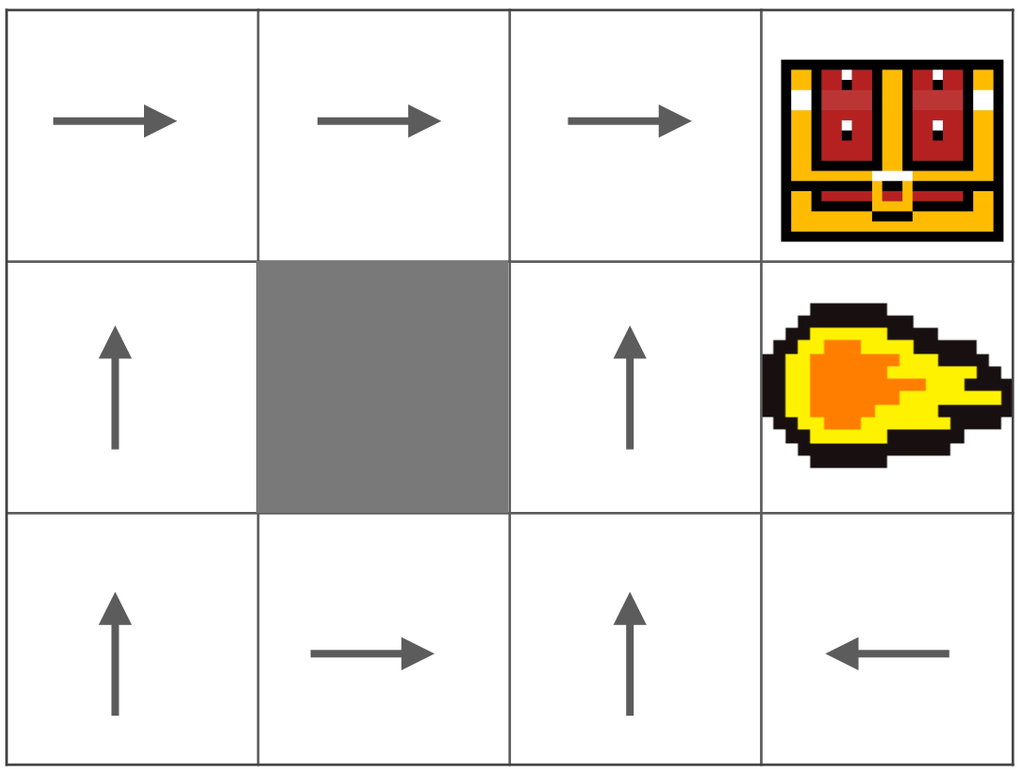
↑ $0.8*0.69 + 0.1*0.75 + 0.1*0.49 = 0.676$

← $0.8*0.75 + 0.1*0.71 + 0.1*0.69 = 0.74$



0.86	0.90	0.93	+1
0.82		0.69	-1
0.78	0.75	0.71	0.49

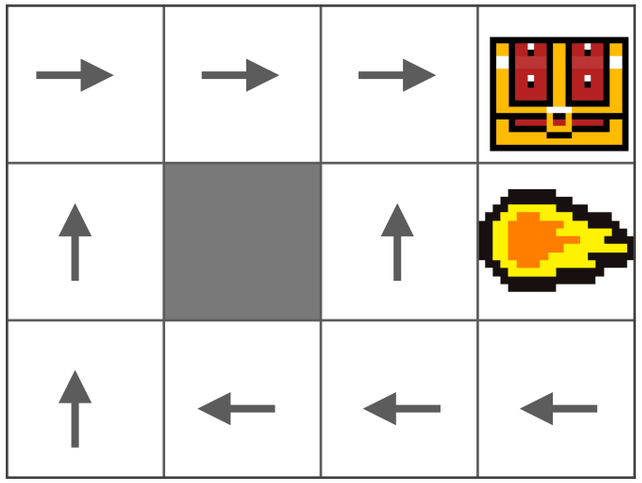
- $P_{SA} \sim \{0.8, 0.1, 0.1\}$
- $\gamma = 0.99$



0.91	0.94	0.97	+1
0.88		0.93	-1
0.85	0.87	0.90	0.86

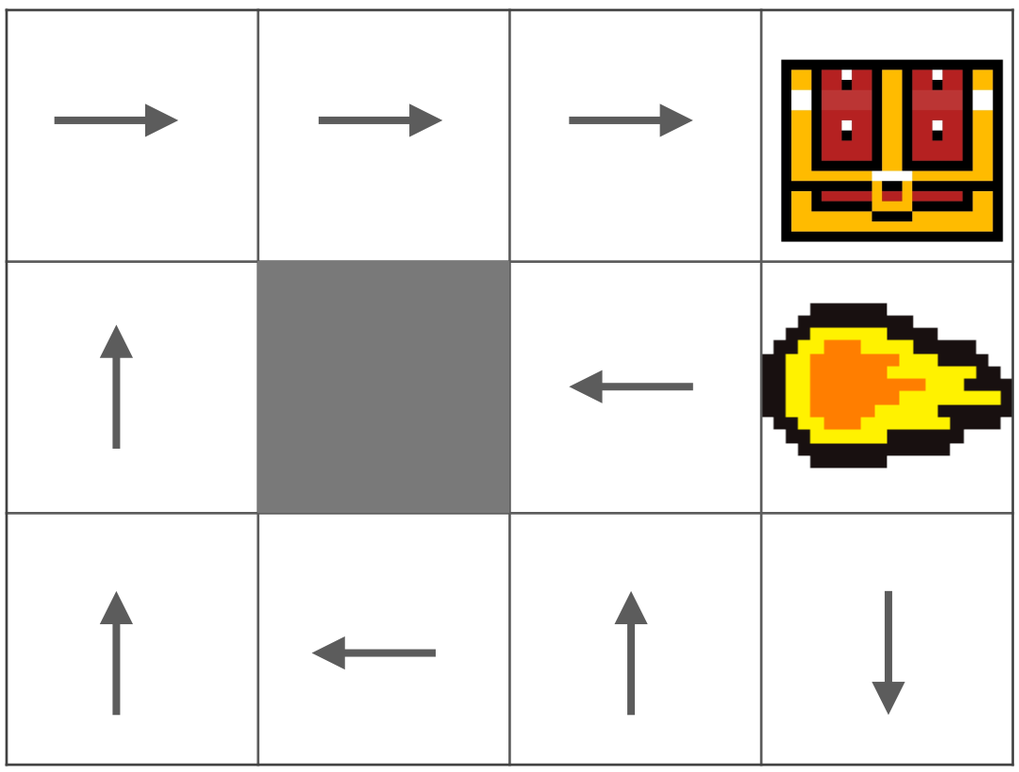
- $P_{SA} \sim \{0.99, 0.005, 0.005\}$
- $\gamma = 0.99$

the robot is very accurate



0.86	0.90	0.93	+1
0.82		0.69	-1
0.78	0.75	0.71	0.49

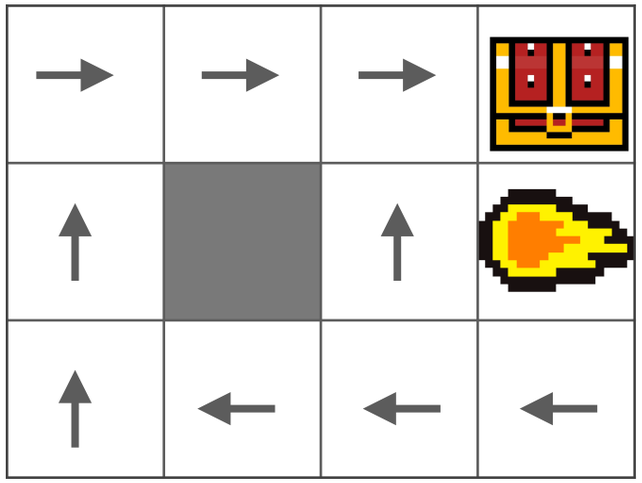
- $P_{SA} \sim \{0.8, 0.1, 0.1\}$
- $\gamma = 0.99$



0.70	0.78	0.83	+1
0.65		0.69	-1
0.57	0.52	0.53	0.43

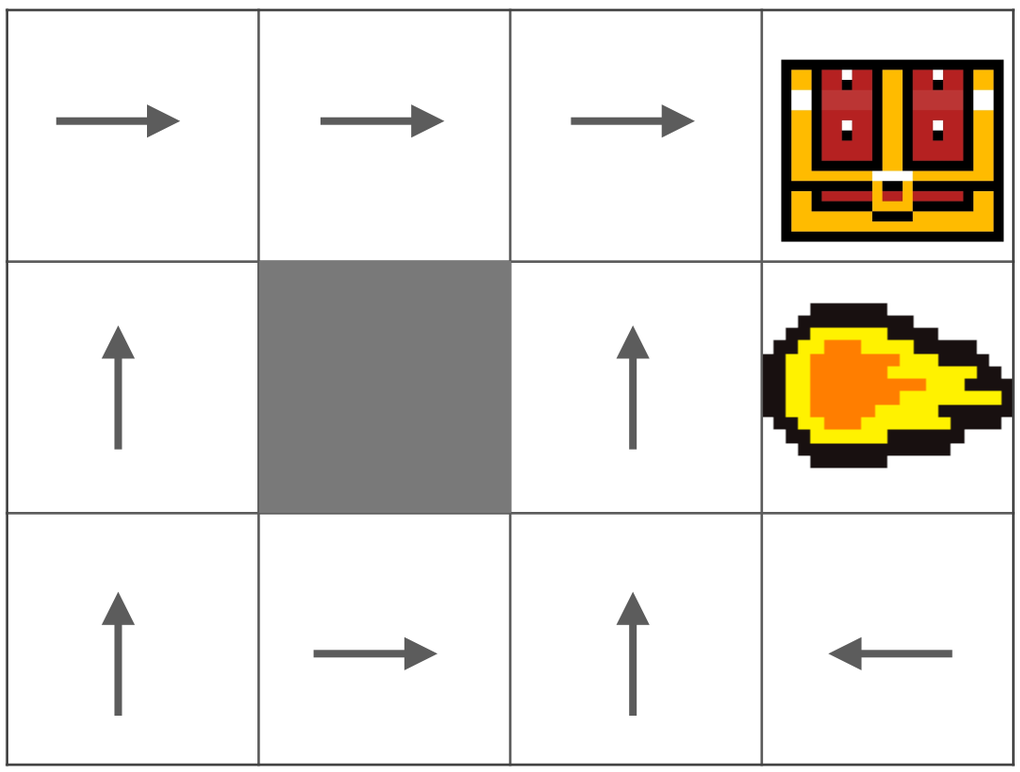
- $P_{SA} \sim \{0.5, 0.25, 0.25\}$
- $\gamma = 0.99$

the robot is very sloppy



0.86	0.90	0.93	+1
0.82		0.69	-1
0.78	0.75	0.71	0.49

- $P_{SA} \sim \{0.8, 0.1, 0.1\}$
- $\gamma = 0.99$



0.58	0.70	0.82	+1
0.48		0.52	-1
0.39	0.34	0.41	0.20

- $P_{SA} \sim \{0.8, 0.1, 0.1\}$
- $\gamma = 0.9$

slight decrease in discount, dramatic effects!

BUT IN ROBOTICS WE DO NOT ALWAYS HAVE ACCURATE MODELS

There is a reason that ATLAS costs USD 2.000.000

WHAT IF THERE IS NO MODEL AVAILABLE OF THE ENVIRONMENT

- $P_{sa}(s')$ is not available
  ➜ value iteration & policy iteration won't work

- One solution is to estimate $P_{sa}(s')$ based on interactions within the environment

$$P_{sa}(s') = \frac{\#\ a \text{ taken in } s \text{ and got to } s'}{\#\ a \text{ taken in } s}$$

- Use Temporal Difference (TD) learning

TEMPORAL DIFFERENCE LEARNING: TD(0)

1. Initialise V(s) arbitrarily, evaluate policy π
2. Repeat until convergence
    1. Initialise s
    2. Repeat (until s is terminal)
        - a is an action given by π
        - do *a*, observe reward *r* and next state

$$\mathrm{V}(s) = \mathrm{V}(s) + \alpha(r + \gamma \mathrm{V}(s') - \mathrm{V}(s))$$

        - update s to s'

No need for a model of the environment, implemented in an online, incremental fashion

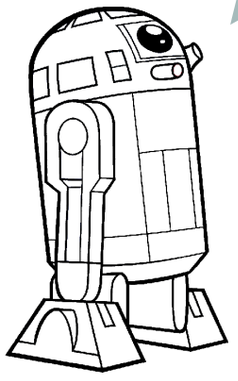
TEMPORAL DIFFERENCE LEARNING: TD(0)

1. Initialise V(s) arbitrarily, evaluate policy π

2. Repeat until

   - $V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$
   - update s to s'

What about finding the optimal control policy?



No need for a model of the environment, implemented in an online, incremental fashion

POLICY IMPROVEMENT

- It's all about the question whether action a improves the policy π or not!
- Define the quality (the payoff) of a state-action combination:

$$Q : S \times A \rightarrow \mathbb{R}$$

$$Q^{\pi}(s, a) = \mathrm{E}[r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s, a_t = a]$$

SOLUTION: Q-LEARNING

- Q-learning: initialise Q randomly and update the Q function every time step k using the following equation

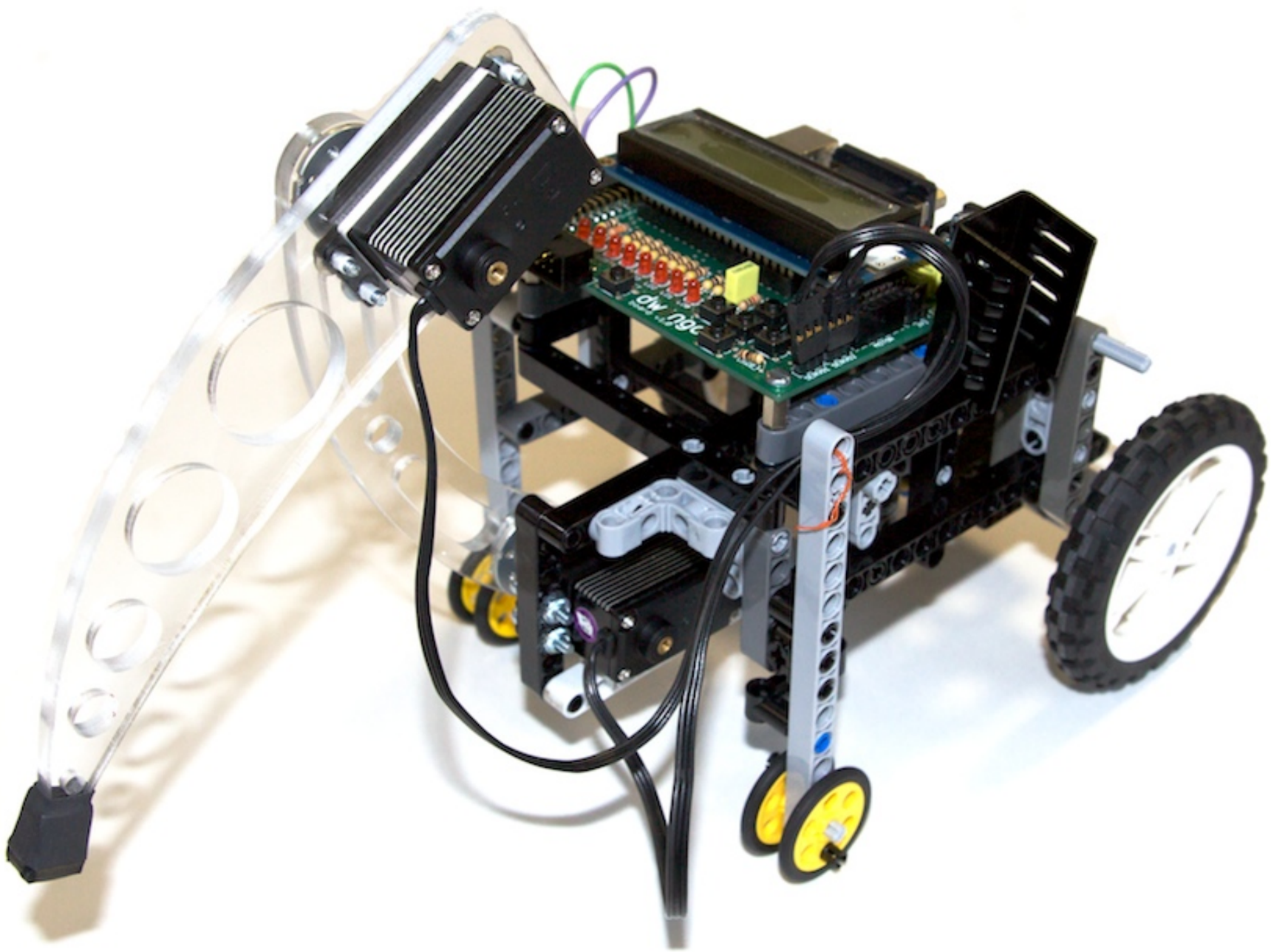
$$Q(s_k, a_k) := (1 - \alpha)Q(s_k, a_k) + \alpha[r_{k+1} + \gamma \max_a Q(s_{k+1}, a)]$$

off-policy learner

- Selection of the action can be done in an ε-greedy way
- This involves an exploration-exploitation trade-off

avoid that the agent gets stuck!

A CRAWLING ROBOT EXAMPLE



[https://www.youtube.com/watch?v=2iNrJx6IDEo]

A CRAWLING ROBOT EXAMPLE

1st UP	1st	2nd UP	2nd
-1.2	0.6	0.9	1.2
0.2	-0.8	-0.5	-1.6
-1.4	0.5	1	-1.1
0.9	-0.9	0.6	0.6
-0.1	1.1	-0.1	0.2
0.2	1	0.9	-1
1.1	-0.1	-0.7	1.9
-0.8	-0.8	-1.1	0.5
0.5	-0.3	1	-0.3
-0.6	1.2	-0.6	-0.9
-1	1.3	0.2	-0.1
-0.1	-0.5	-0.9	1
0.9	0.2	1.1	-0.6
0.8	-0.1	0.3	1.3
-0.1	-1.4	1	-0.4
-1	-0.1	-1.1	0.8

A CRAWLING ROBOT EXAMPLE

1st UP	1st	2nd UP	2nd
-1.2	0.6	0.9	1.2
0.2	-0.8	-0.5	-1.6
-1.4	0.5	1	-1.1
0.9	-0.9	0.6	0.6
-0.1	1.1	-0.1	0.2
0.2	1	0.9	-1
1.1	-0.1	-0.7	1.9
-0.8	-0.8	-1.1	0.5
0.5	-0.3	1	-0.3
-0.6	1.2	-0.6	-0.9
-1	1.3	0.2	-0.1
-0.1	-0.5	-0.9	1
0.9	0.2	1.1	-0.6
0.8	-0.1	0.3	1.3
-0.1	-1.4	1	-0.4
-1	-0.1	-1.1	0.8

1st UP	1st	2nd UP	2nd
5	-15	-29	10
12	16	3	28
41	19	19	24
6	11	-8	-18
-4	5	36	11
0	18	13	-16
-2	35	17	**112**
124	37	6	15
-35	-80	17	-32
-16	-10	25	-9
-3	**93**	1	0
-59	-6	**22**	-40
-4	38	13	1
8	-4	17	14
16	5	15	23
-35	-34	-39	-12

PART II

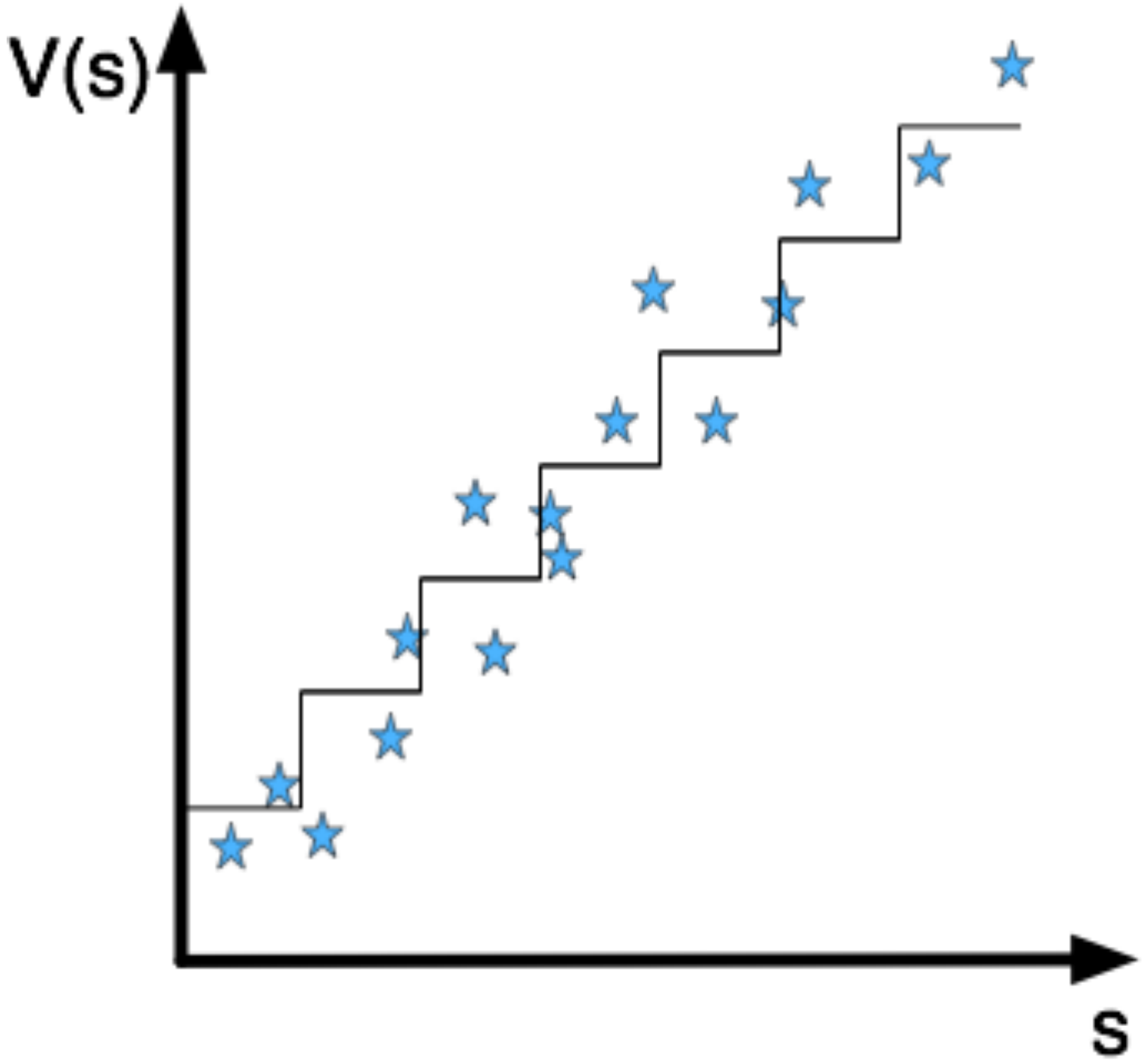
Continuous state space & discrete action space

[Blog Forbes]

CONTINUOUS STATE MDP'S

$$x,\ y,\ z,\ \alpha,\ \beta,\ \gamma,\ \dot{x},\ \dot{y},\ \dot{z},\ \dot{\alpha},\ \dot{\beta},\ \dot{\gamma}$$

DISCRETISATION LEADS TO POOR REPRESENTATIONS OF THE DATA



DISCRETISATION: A CURSE

- the curse of dimensionality
- Let's say $s \in \mathbb{R}^n$
- If s can be divided into k grids, then k^n possibilities
- For chess is this already 64^32 (minus some illegal positions…)

Tip: see https://en.wikipedia.org/wiki/Game_complexity for game complexities

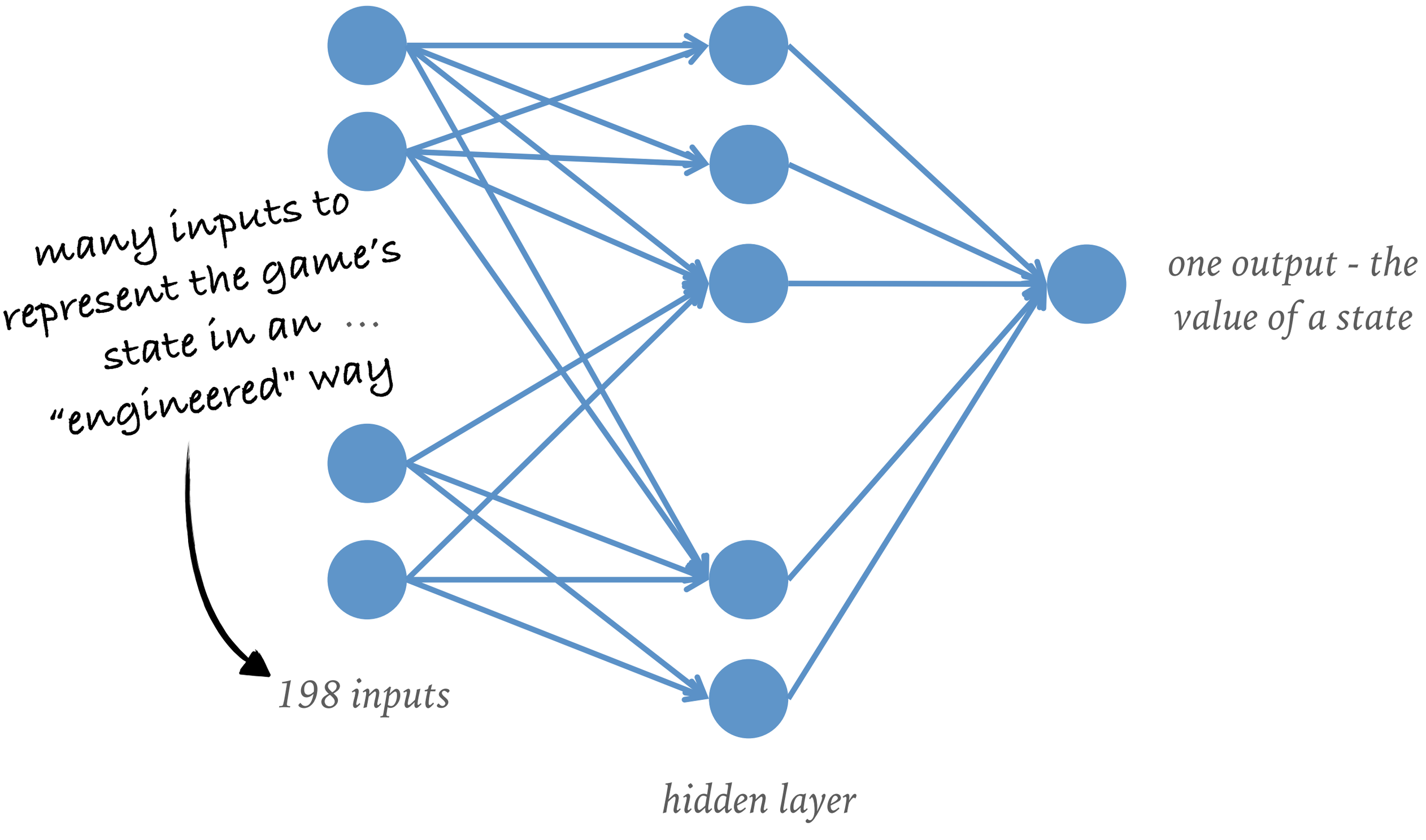
IN SUMMARY: CONTINUOUS STATE SPACE

- Too many states to visit
- Too many states to memorise
- Solution: generalise!

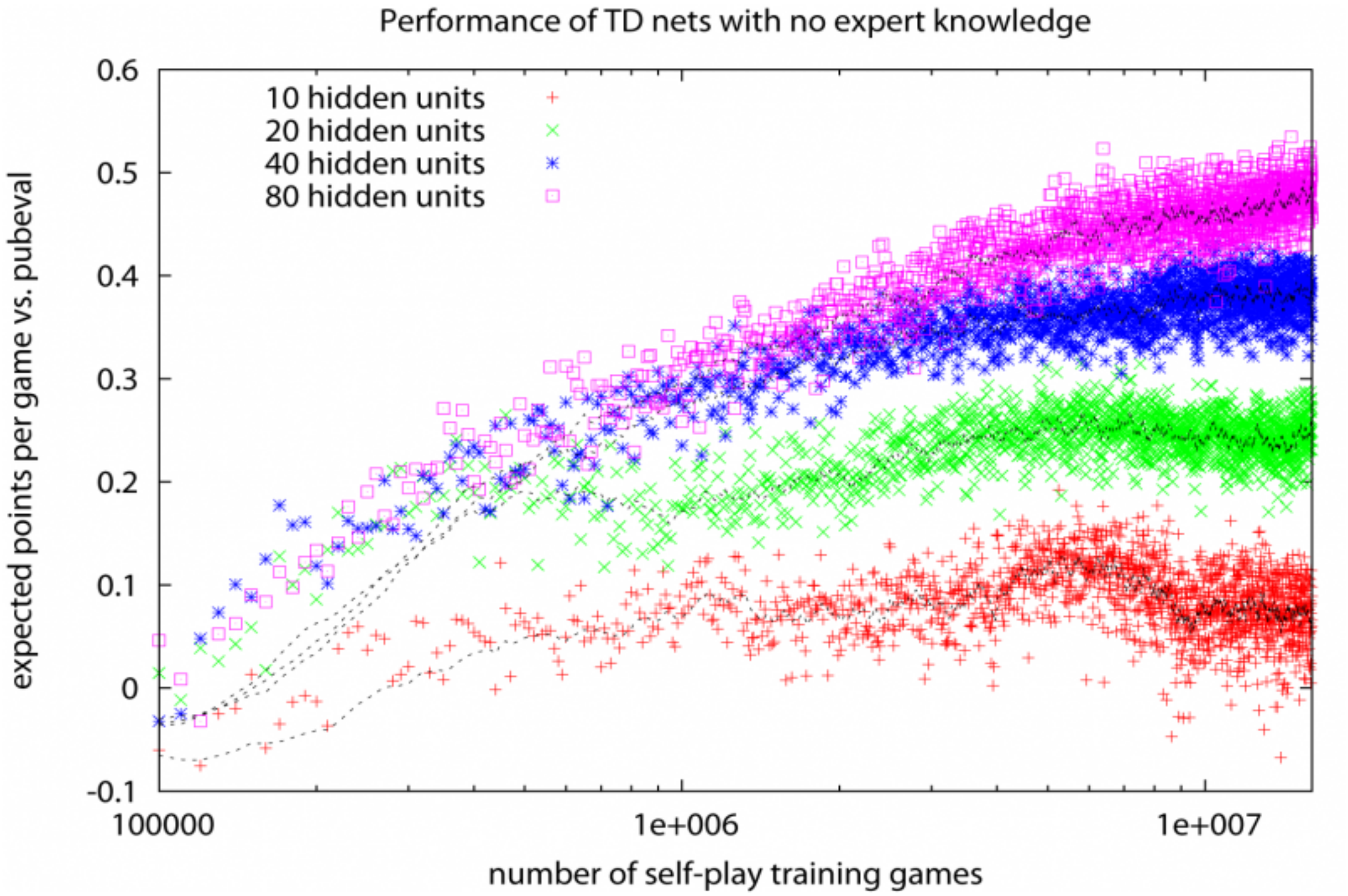
THE BACKGAMMON CASE

THE TD-GAMMON CASE BY TESAURO, 1995

APPROXIMATION OF THE VALUE-FUNCTION WITH A NEURAL NETWORK



TD-GAMMON – PERFORMANCE



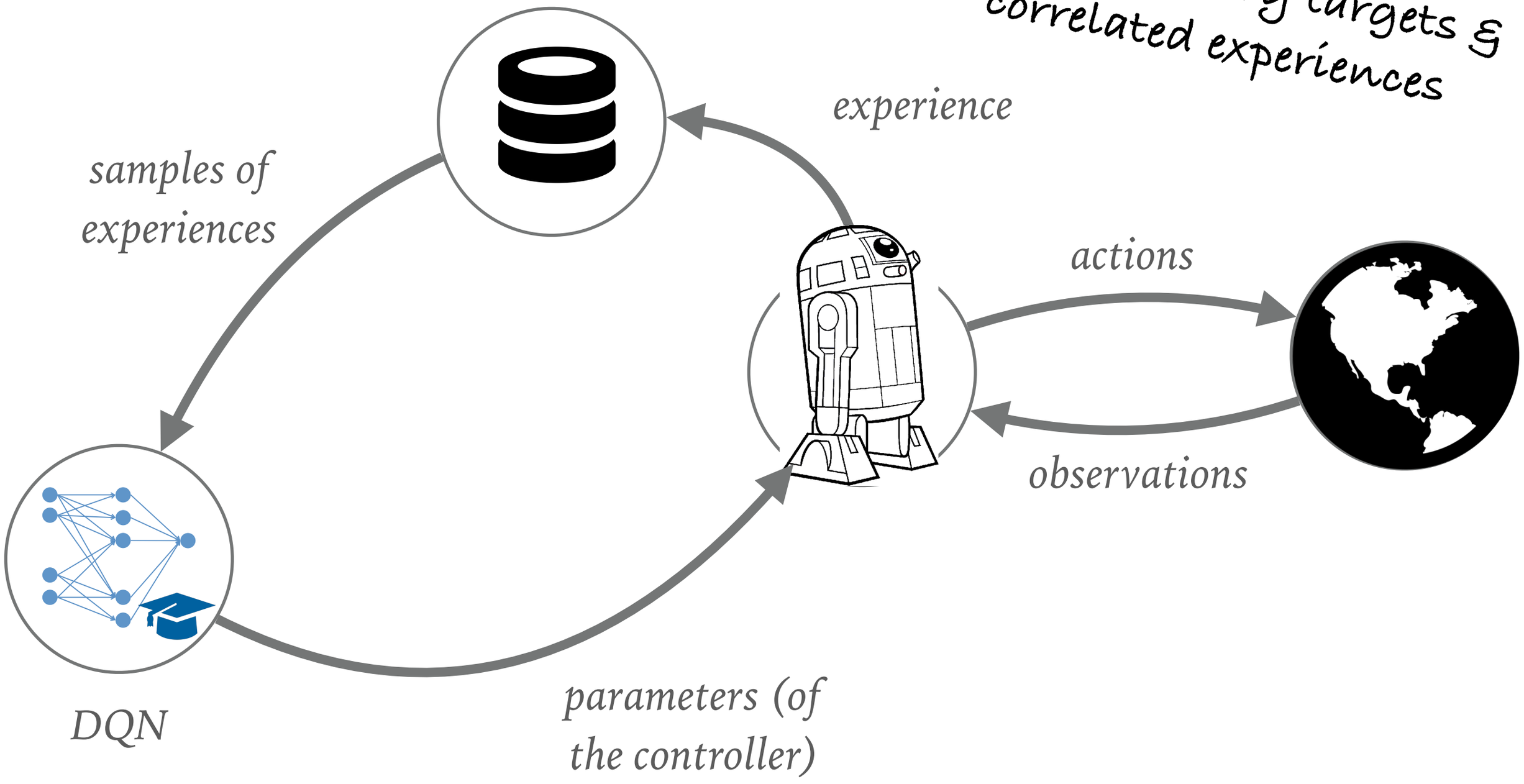
TD-GAMMON GREATLY INFLUENCED HOW HUMAN PLAYERS PLAY TODAY

similar to the achievements of AlphaGo Zero today

version	hidden neurons	trained games	human opponent	result
1	80	300000	Robertie	-13after 51 games
2	40	800000	Variousmasters	-7 after 38 games
2.1	80	1500000	Robertie	-1 after 40 games
3	80	1500000	Kazaros	6 after 20 games

(NEURAL) FITTED Q-ITERATION

convergence is not guaranteed, problems: non-stationary targets & correlated experiences



FITTED Q-ITERATION

random sampling experiences in order to avoid correlations

- Approximate Q(s,a) instead of discretisation
- Update is done with a set of historical data (interactions) also known as the replay memory $H = \{\langle s_i, a_i, r_i, s'_i \rangle\}$

$$\mathcal{L}_i(\theta_i) = \mathrm{E}_{s,a,r,s' \sim H}\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2$$

- Apply a well known regression method (e.g. neural network)
- Use bootstrapping to incorporate already well known information

we keep the netwerk fixed to enforce some stationarity

Learning to Swing-Up and Balance from Scratch

"Neuroinformatics Group"
University of Osnabrück
Prof. Dr. Martin Riedmiller

THAT IS SO NINETIES, NOW EVERYTHING IS DEEP . . .

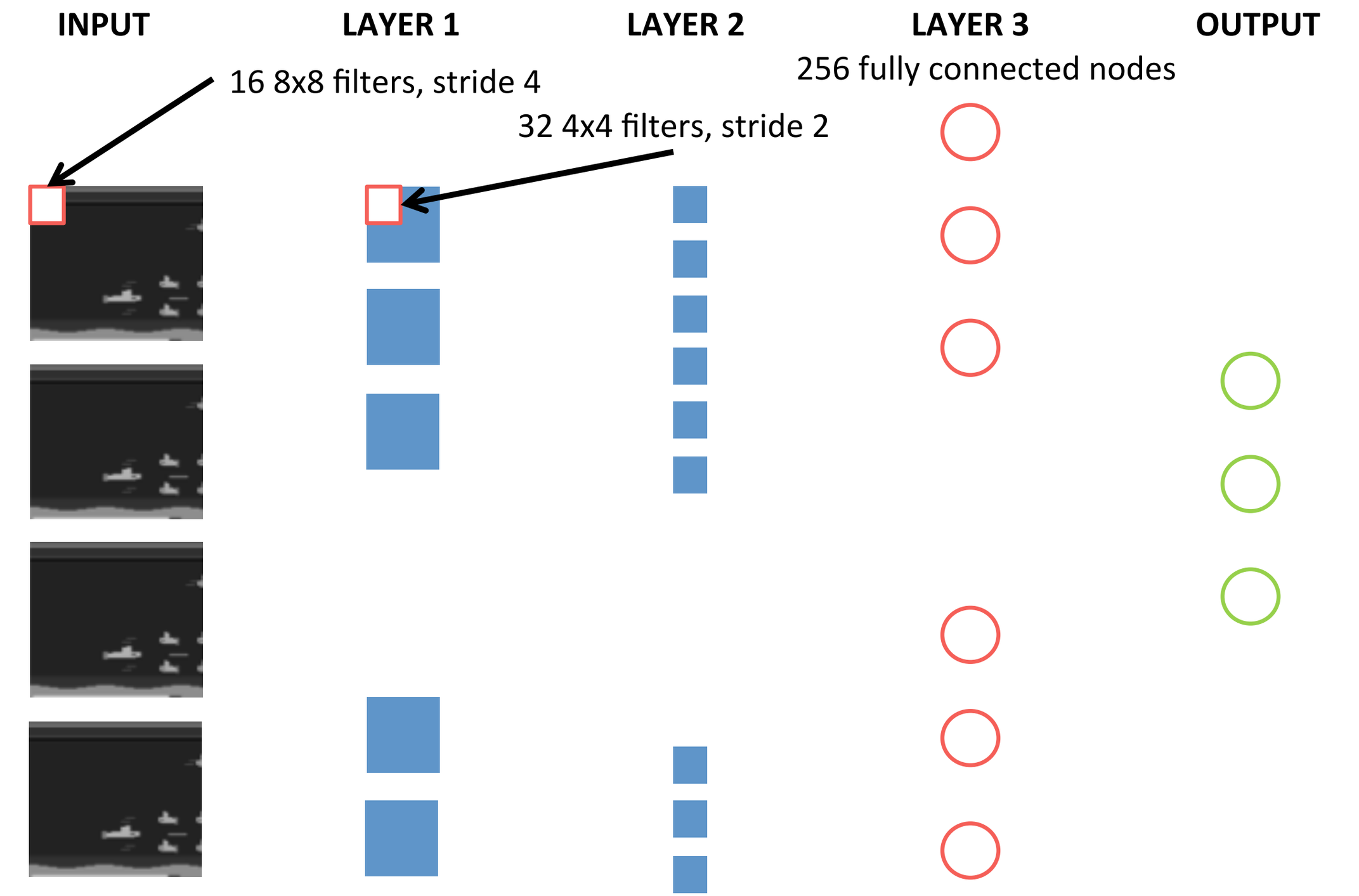
DEEP Q-LEARNING: THE ARCADE GAME CASE

- Approximate the Q-function with a deep neural network
- State = image
- Output = Q-value for each possible action

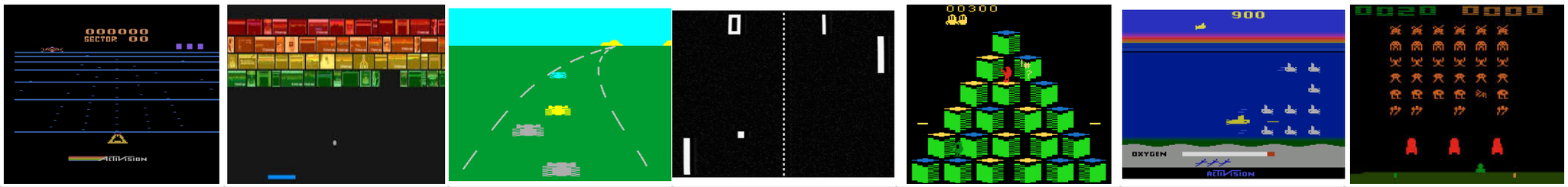
210 x 160 pixels - 128 colours

84 x 84 pixels - 4 colours

DEEP Q-LEARNING: THE ARCADE GAME CASE



DEEP Q-LEARNING: THE ARCADE GAME CASE

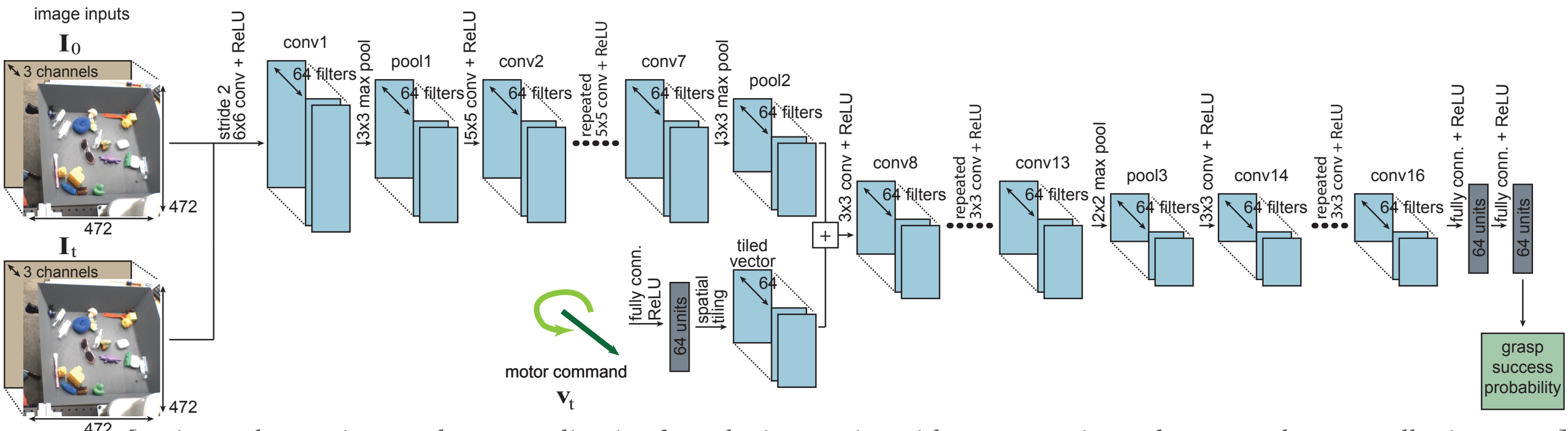


Random	354	1.2	0	-20.4	157	110	179
SARSA	996	5.2	129	-19	614	665	271
Contingency	1743	6	159	-17	960	723	268
DQN	4092	**168**	**470**	**20**	1952	1705	581
HUMAN	**7456**	31	368	-3	**18900**	**28010**	**3690**

[Mnih et al, Playing Atari with Deep Reinforcement Learning, 2014]

[Google Deepmind - Arcade Games]

THE GRASPING ROBOTS CASE: 14 ROBOTS, 2 MONTHS, > 800.000 ATTEMPTS



[Levine et al, Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection, 2016]

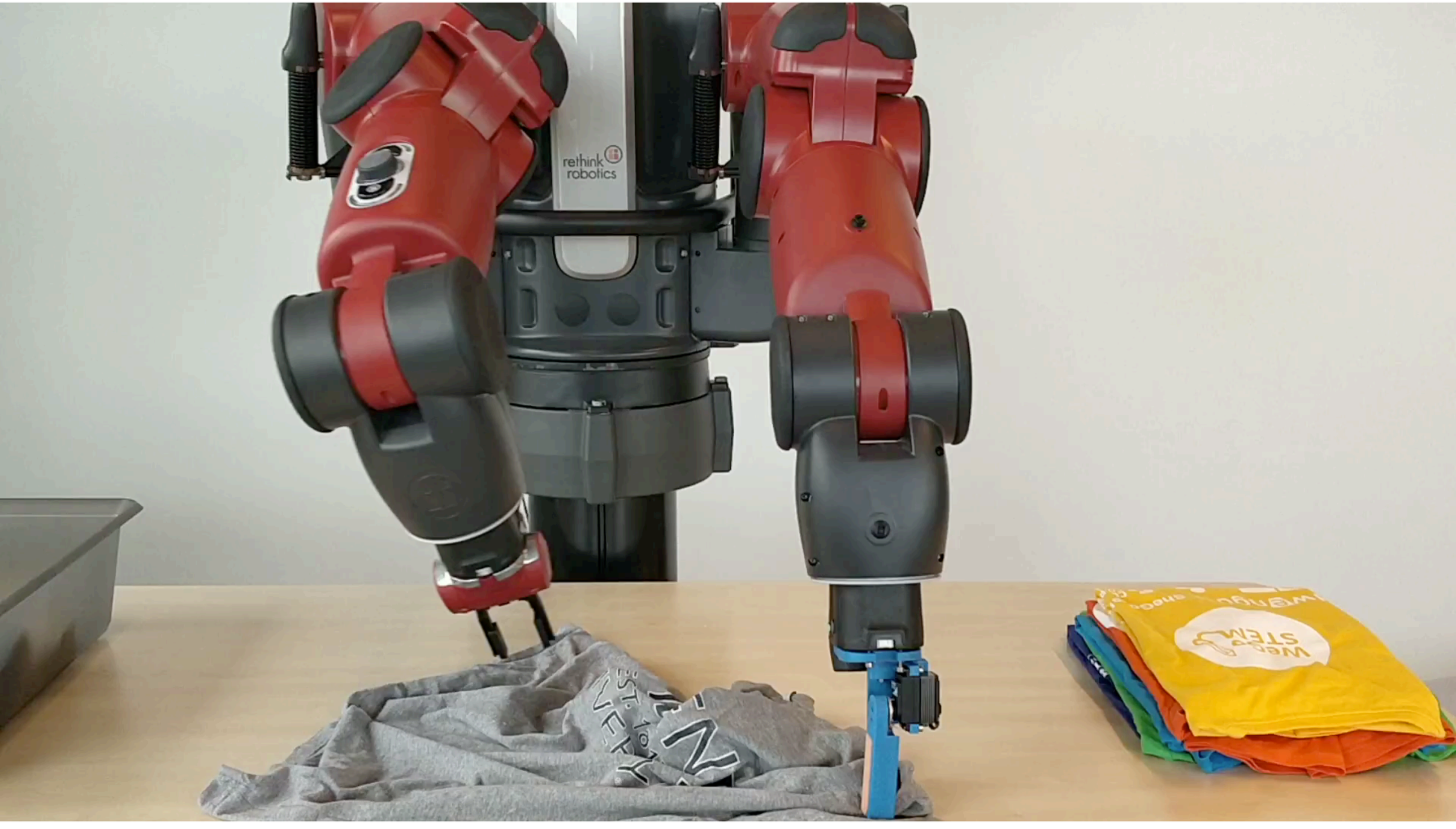
HOE LEREN
ROBOTS
VAN MENSEN?

PART III

What if you don't know the reward function?

[picture by Wouter Van Vooren]

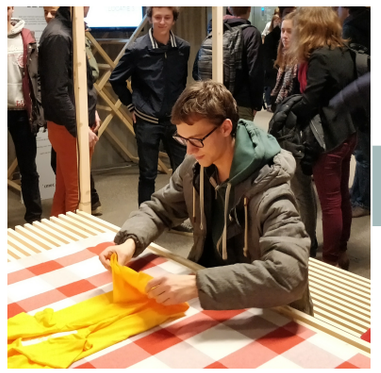
LET'S EVALUATE FOLDING SKILLS



[https://www.youtube.com/watch?v=VAhTgKT86W4]

[https://www.youtube.com/watch?v=VAhTgKT86W4]

data collection



clustering

segmentation of the most important steps

inverse reinforcement learning

reinforcement learning

folding robot

INVERSE REINFORCEMENT LEARNING

- We try to find the reward function R that explains the expert behaviour
- Assume that the expert is indeed optimal
- Assume we can enumerate all policies -> computational challenge!
- Ongoing research topic

AVAILABLE FOR EVERYONE?

DEEP Q-LEARNING: PATENTED BY GOOGLE

Methods and apparatus for reinforcement learning

US 20150100530 A1

Publication number	US20150100530 A1
Publication type	Application
Application number	US 14/097,862
Publication date	Apr 9, 2015
Filing date	Dec 5, 2013
Priority date	Oct 8, 2013
Also published as	CN105637540A, EP3055813A1, WO2015054264A1
Inventors	Volodymyr Mnih, Koray Kavukcuoglu
Original Assignee	Google Inc.
Export Citation	BiBTeX, EndNote, RefMan

Patent Citations (10), Non-Patent Citations (6), Referenced by (2), Classifications (6), Legal Events (2)

External Links: USPTO, USPTO Assignment, Espacenet

ABSTRACT

We describe a method of reinforcement learning for a subject system having multiple states and actions to move from one state to the next. Training data is generated by operating on the system with a succession of actions and used to train a second neural network. Target values for training the second neural network are derived from a first neural network which is generated by copying weights of the second neural network at intervals.

IMAGES (11)

“

There has, nonetheless, been an attempt to use a multilayer perceptron to approximate a Q-value function, in “Neural fitted Q iteration—first experiences with a data efficient neural reinforcement learning method”, Machine Learning: ECML 2005, Springer 2005, pages 317-328, Riedmiller, Martin. The technique described there is based on the principle of storing and reusing transition experiences but has some significant practical disadvantages: broadly speaking a neural network is trained based on the stored experience, and when the experience is updated with a new (initial state—action—resulting state) triple the previous neural network is discarded and an entirely new neural network is trained on the updated experience.

-Excerpt from US 20150100530 A1

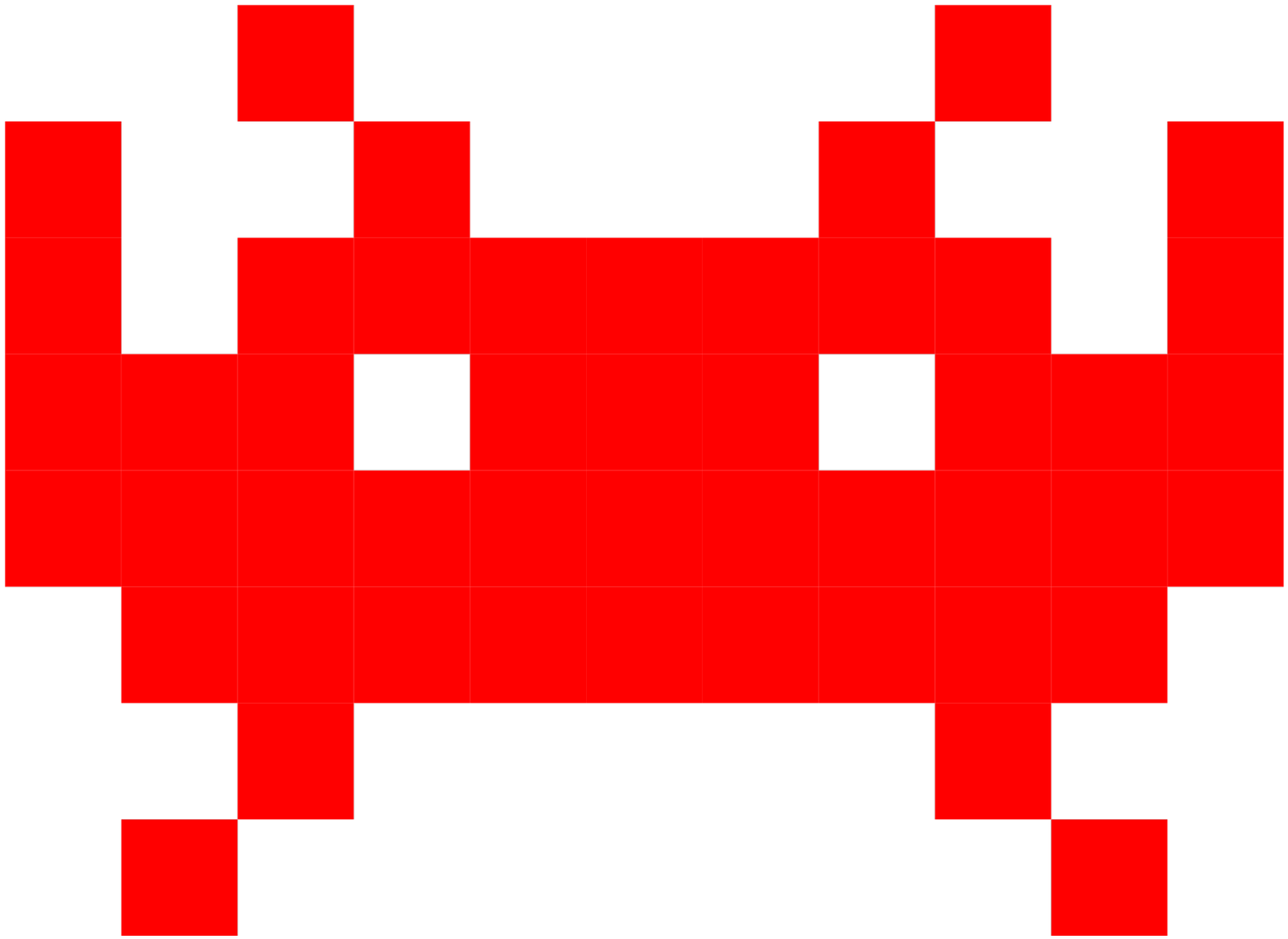
TAKE HOME MESSAGES

1. Use RL in problems for which you don't have explicit solutions on the condition that you can define some reward ~ learning with a critic

2. Is there a model of the world
   - ✓ YES -> PI or VI
   - ✓ NO -> TD-learning

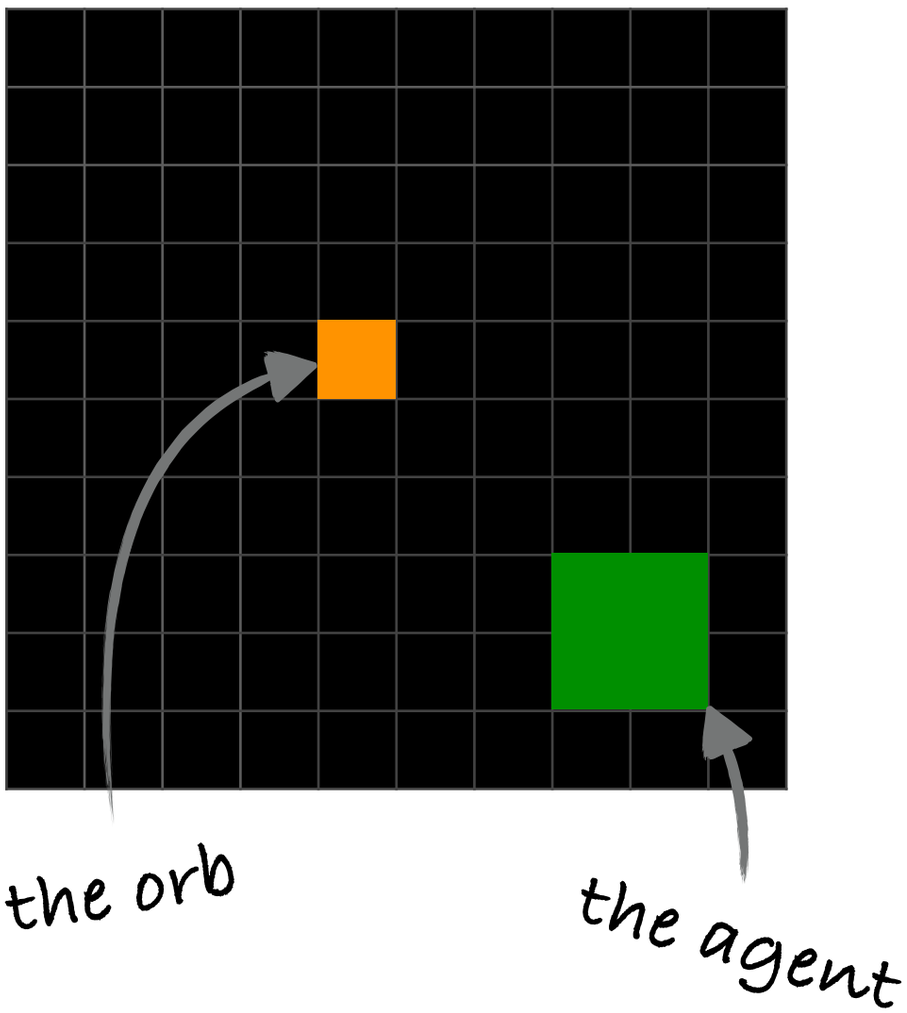
3. Continuous state space? Try to approximate the value function (or Q function)

4. There are many interesting challenges out there!



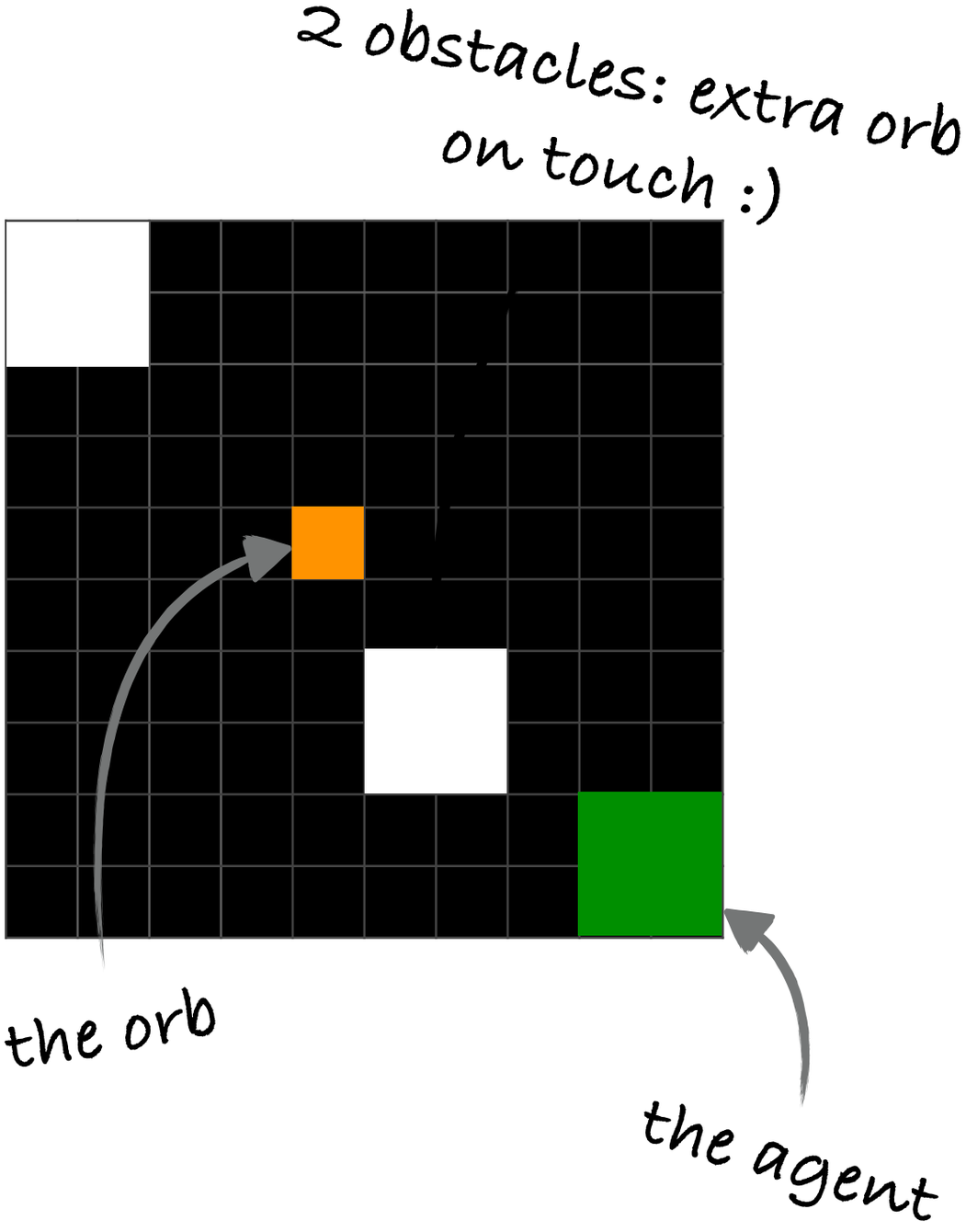
PROJECT ASSIGNMENT

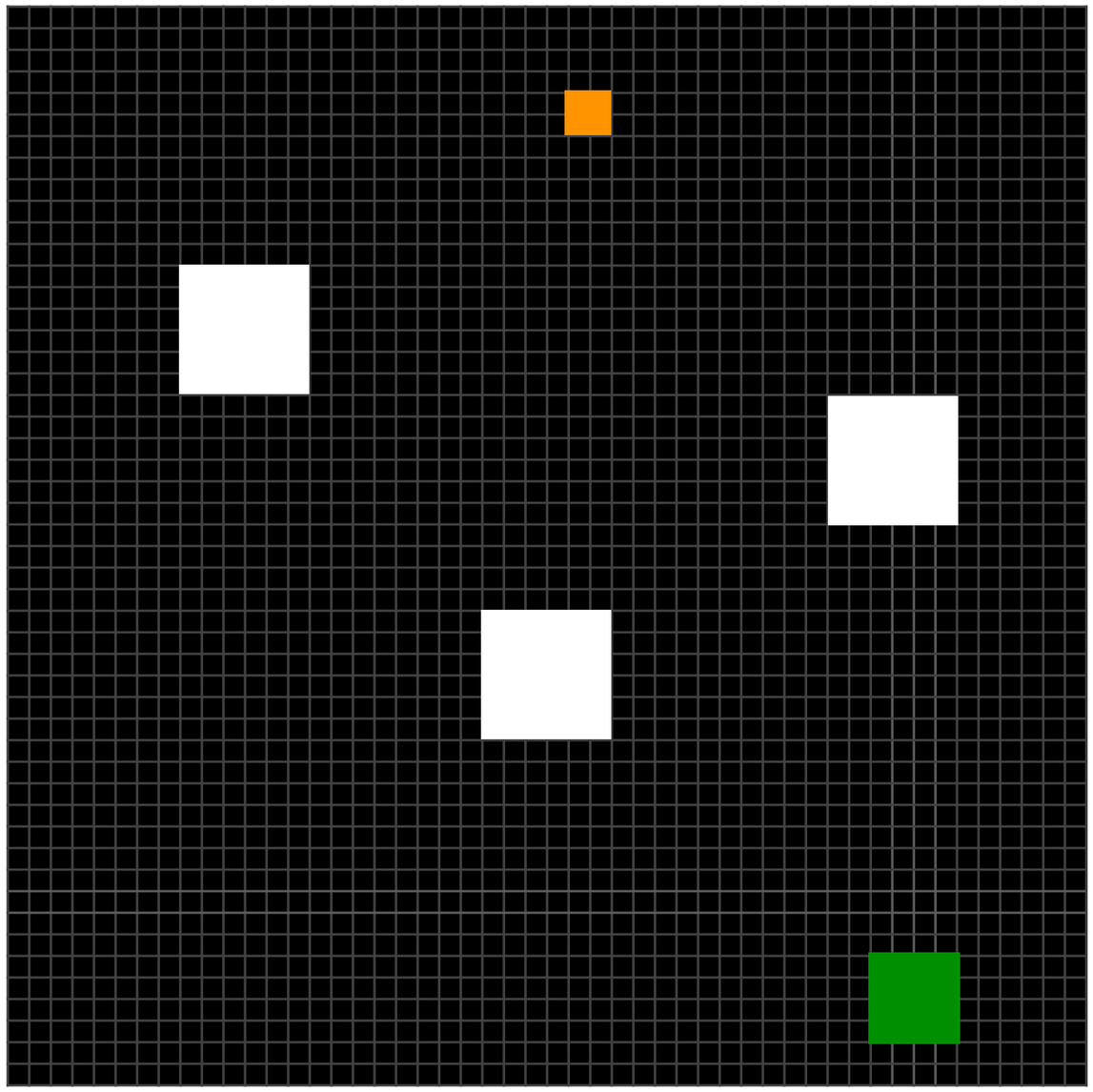
ORB CATCHING GAME



- Hands-on experience with the principles of reinforcement learning
- Starts with fixed tasks
  - ✓ reward function
  - ✓ preprocessing
  - ✓ DQN architecture
  - ✓ policy
  - ✓ debugging
- Possible questions on your exam!

ORB CATCHING GAME: ADDING HIGH LEVEL GOALS & INCREASING SIZE





M.A.R.C. a quadruped robot

WILL YOU TRAIN THE NEXT GENERATION ROBOTS?

Guest lecture by Francis wyffels

@fwyffels - francis.wyffels@UGent.be

GHENT UNIVERSITY

CC BY SA