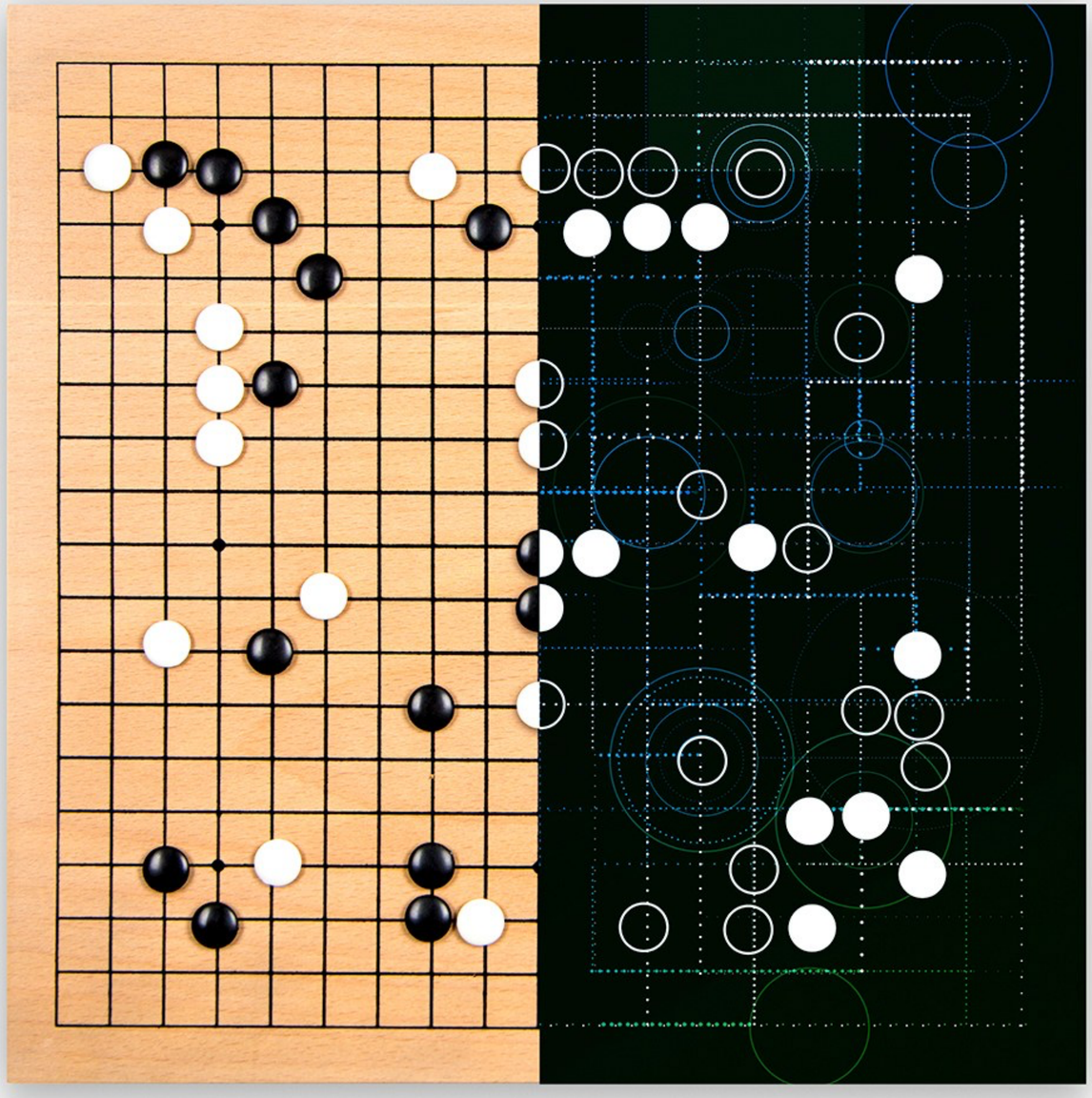
Introduction to Reinforcement Learning 🤖

Piotr Januszewski

Where is it used?

- Robotics
- Video games
- Conversational systems
- Medical intervention
- Algorithm improvement
- Improvisational theatre
- Autonomous driving
- Prosthetic arm control
- Financial trading
- Query completion

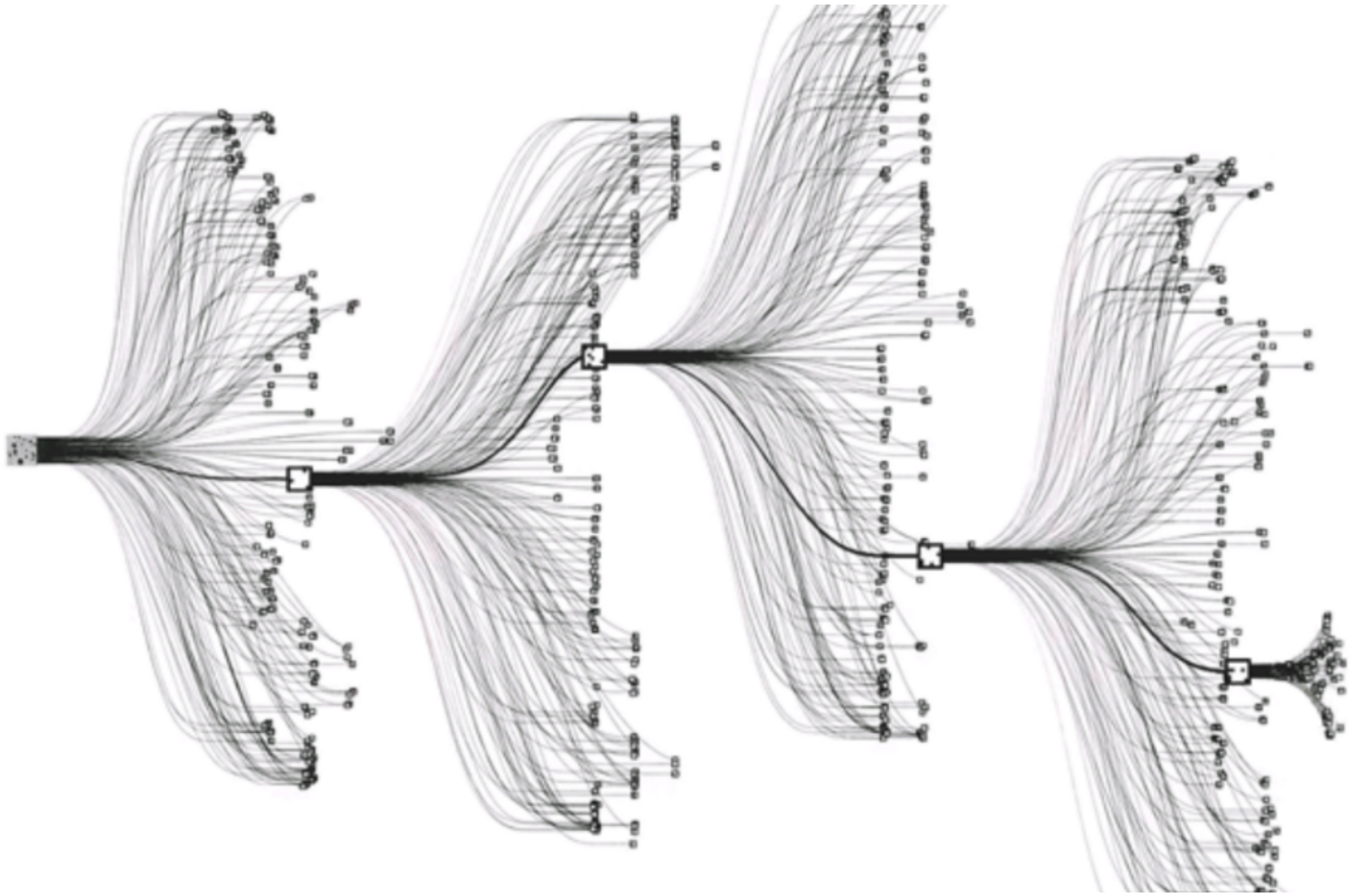


Source: https://www.wired.com/2016/01/in-a-huge-breakthrough-googles-ai-beats-a-top-player-at-the-game-of-go/

DeepMind’s AlphaGo

AlphaGo Lee version of this algorithm became **the first AI to beat a human grandmaster of Go** in March of 2016.

It is huge achievement as for a long time people thought that **Go is the holy grail of artificial intelligence** and we were still many years before AI will beat humans in this game!

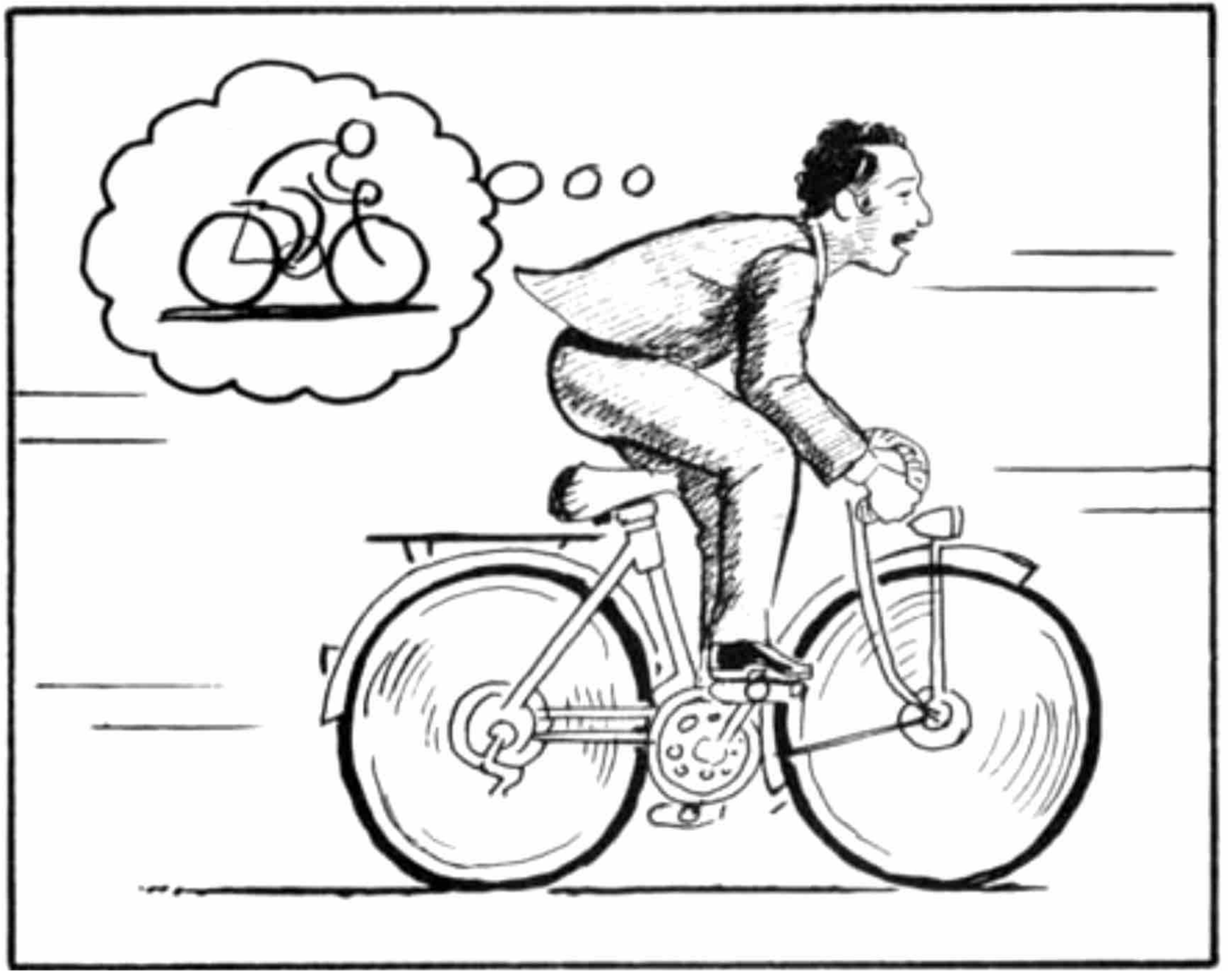


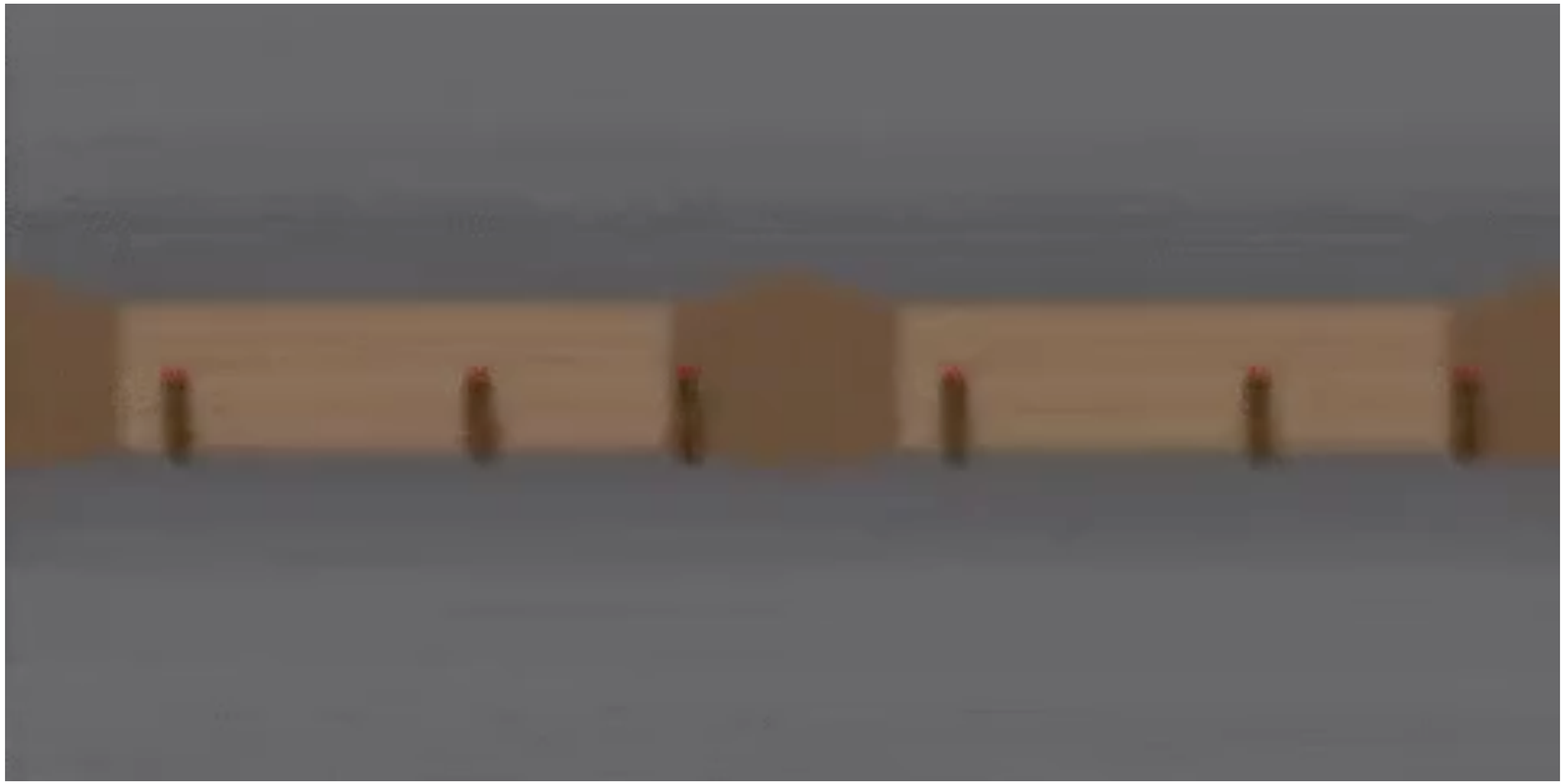
The ancient Game of **Go has more states then there are atoms in the known universe** and it’s branching factor is more then seven times bigger then the one of chess.

DeepMind’s AlphaZero

In October of 2017, DeepMind published new AlphaGo version. **AlphaGo Zero had defeated AlphaGo 100–0**. Incredibly, it had done so **by learning solely through self-play**. No longer was a database of human expert games required to build a super-human AI.

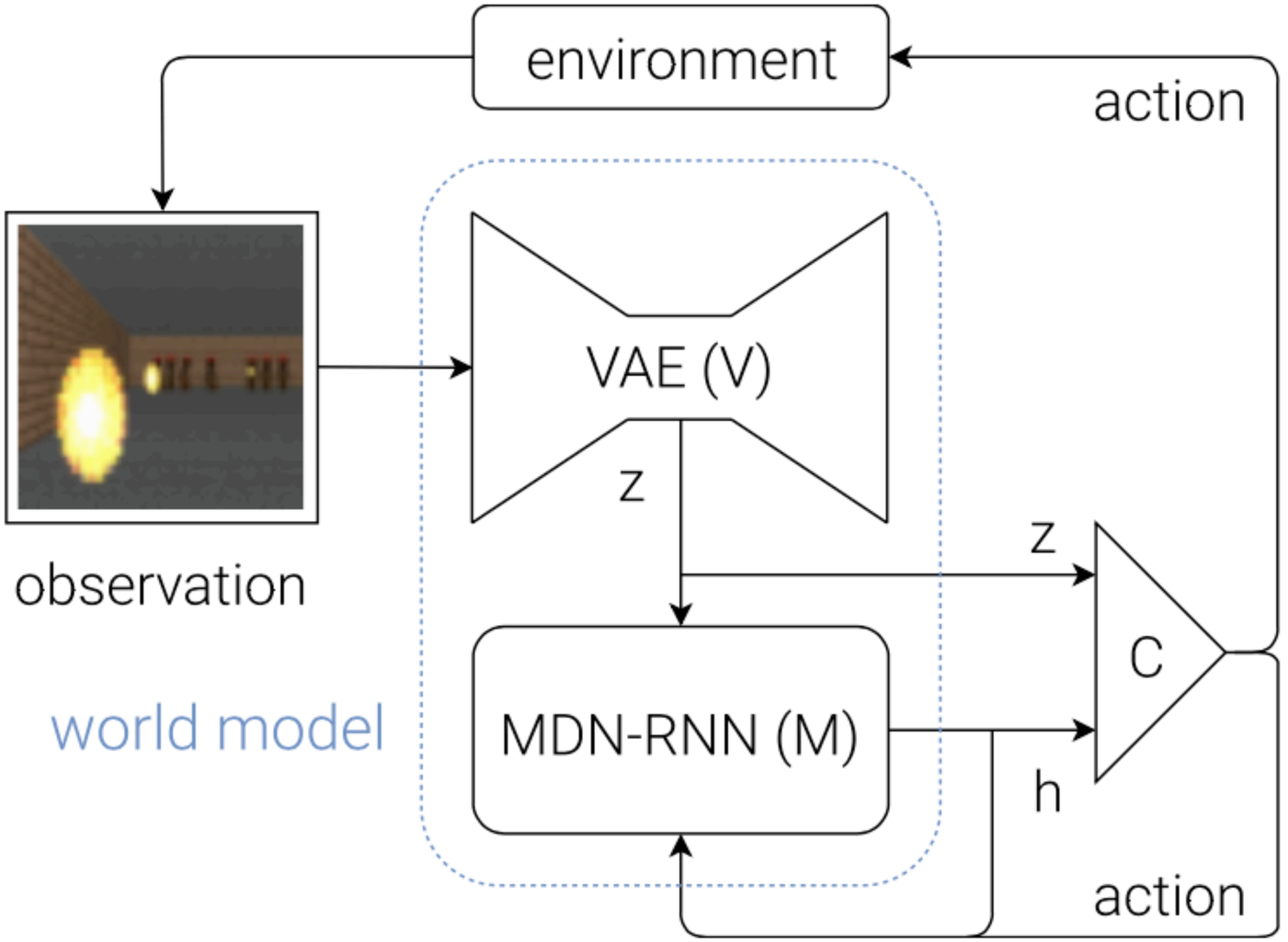
48 days later they publish current state-of-the-art version AlphaZero, which can play other games like Chess and Shogi. **It can beat the world-champion programs StockFish** (for Chess) **and Elmo** (for Shogi) **learning through self-play for only about 4 and 2 hours respectively!**



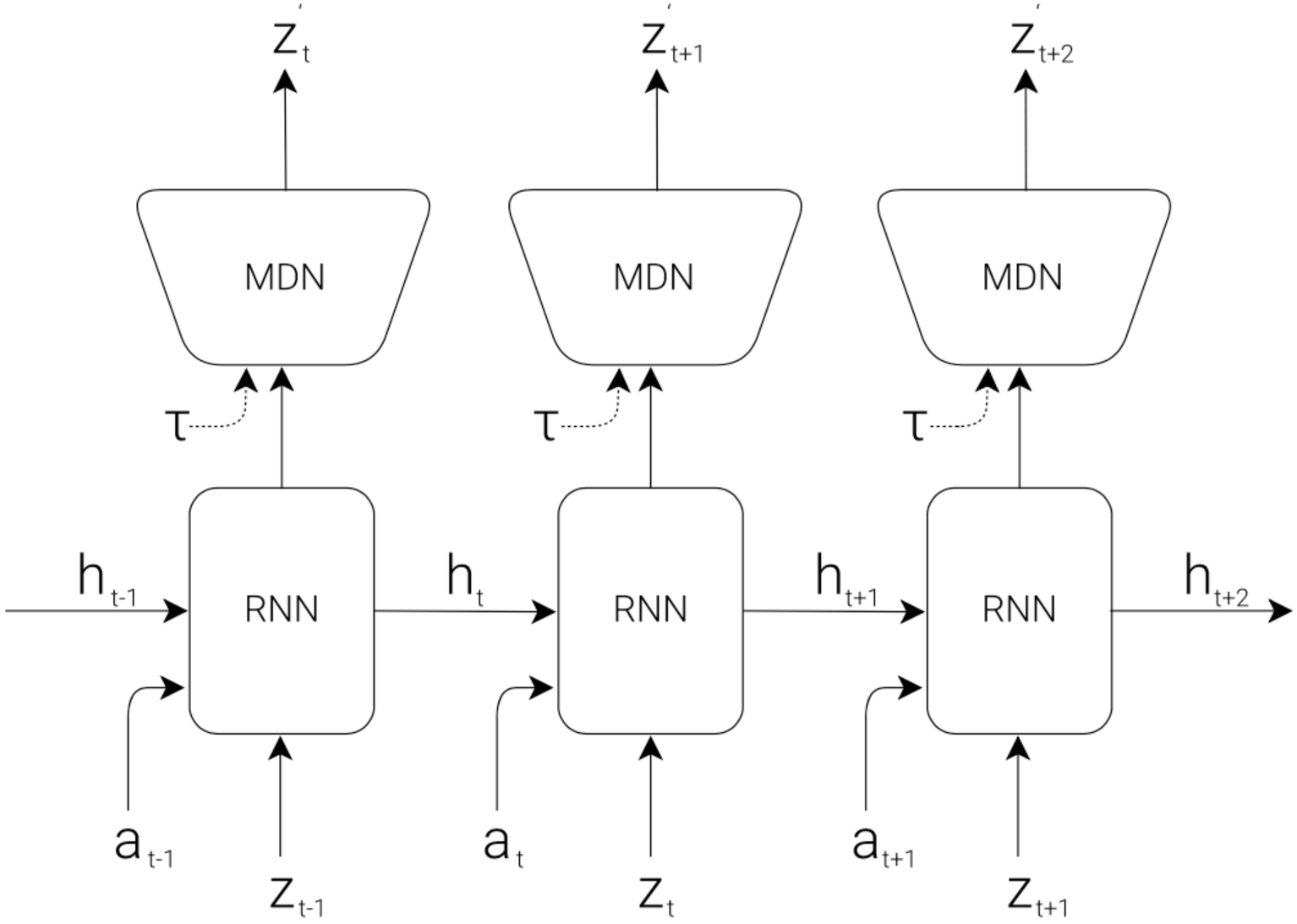


They were able to train agent avoid fireballs in the environment. But the training took place in **agent's imagination!**

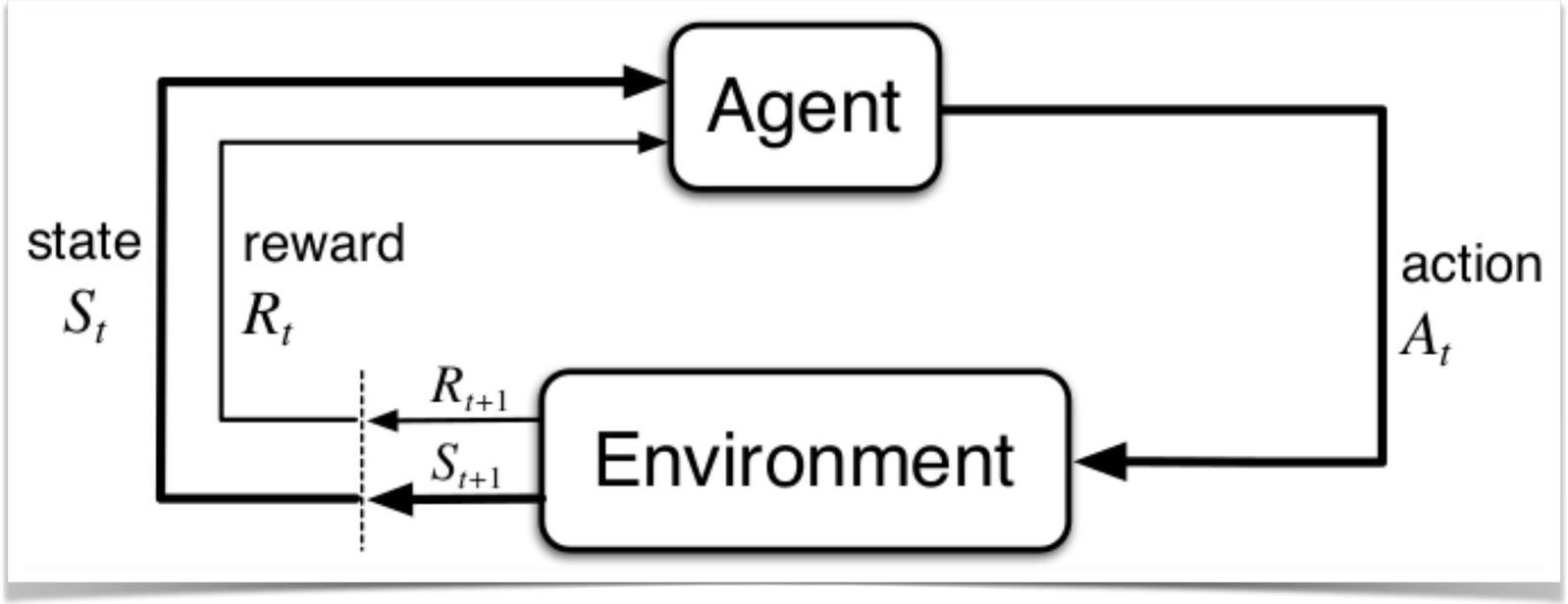
Agent architecture



Memory architecture



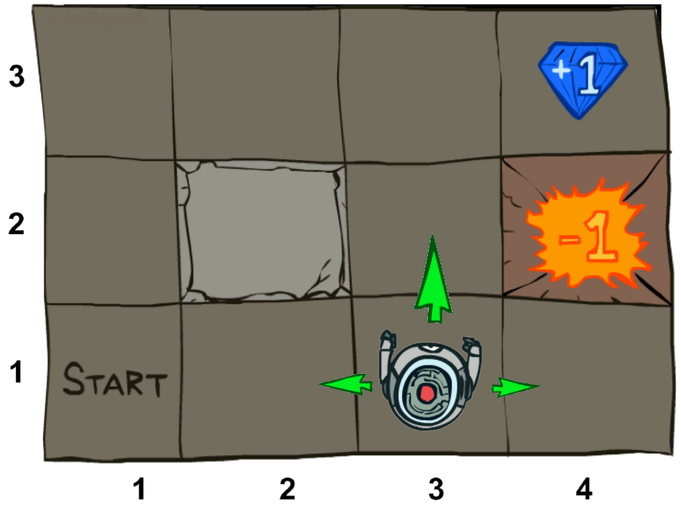
Reinforcement Learning



- Learning by trial-and-error, in real-time.
- Improves with experience
- Inspired by psychology – Agent + Environment
  – Agent selects actions to maximise total reward (return)

Example: Grid World

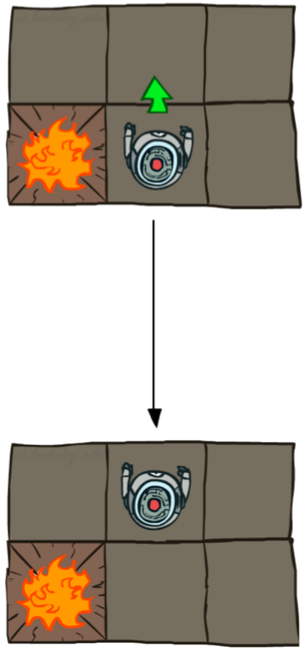
- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
  - States are robot positions
- The agent receives rewards each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximise sum of rewards



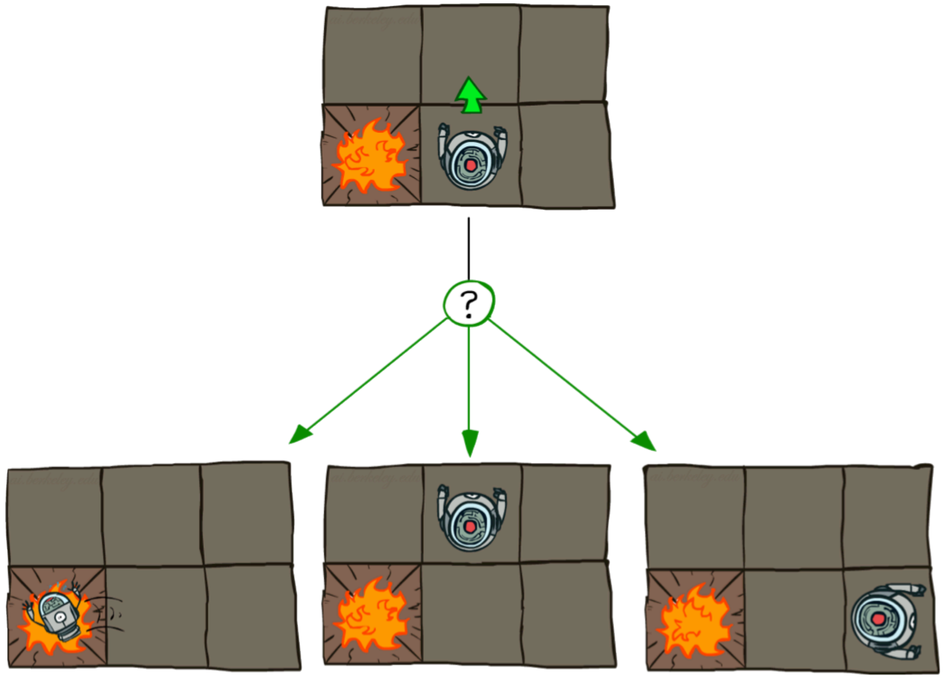
Source: UC Berkeley CS188 Intro to AI

Grid World Actions

Deterministic Grid World

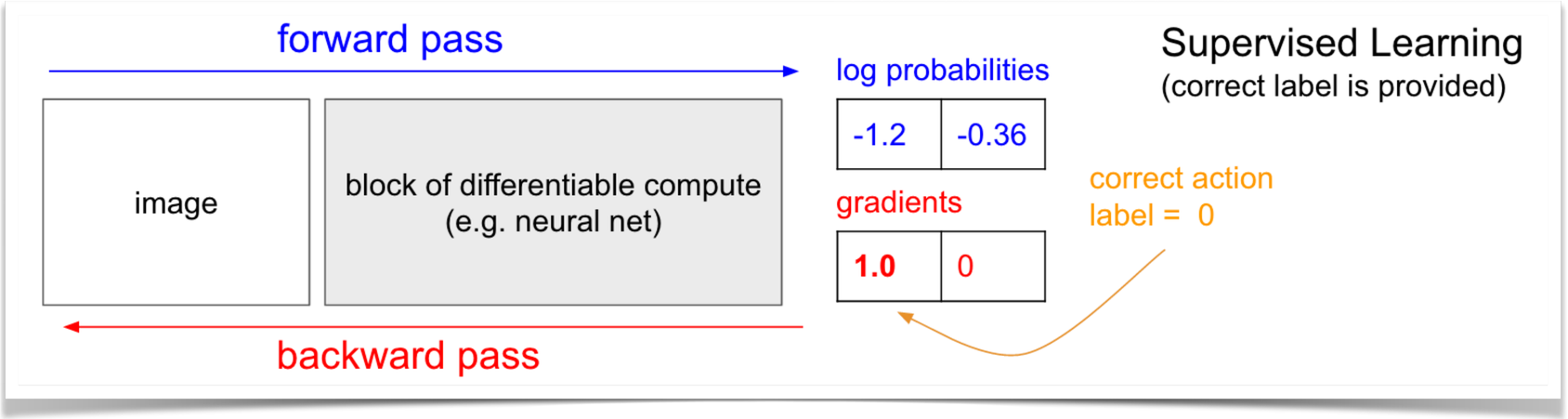


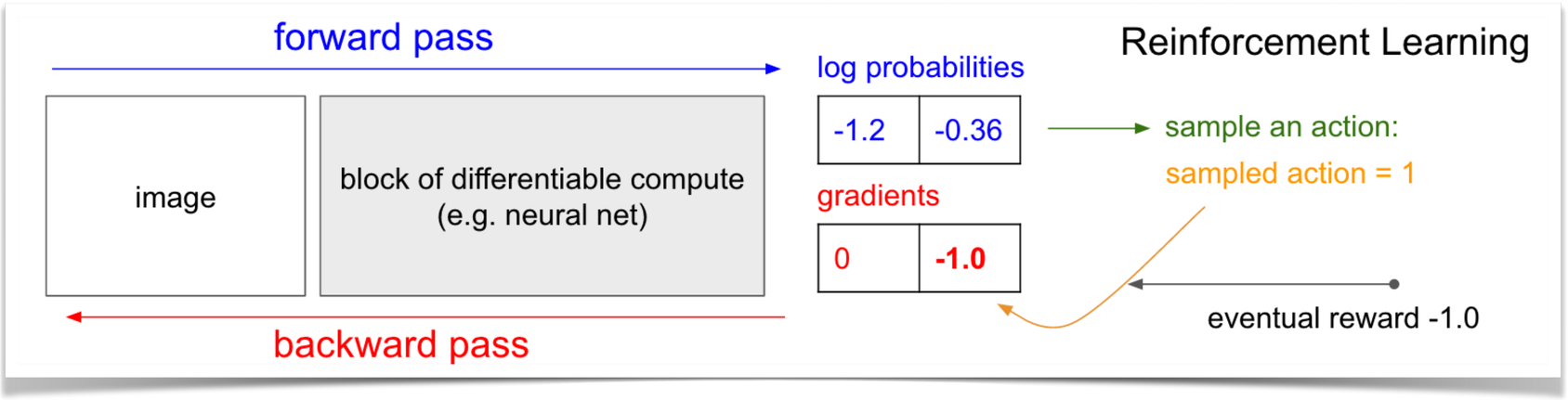
Stochastic Grid World



Source: UC Berkeley CS188 Intro to AI

How is RL related to SL?



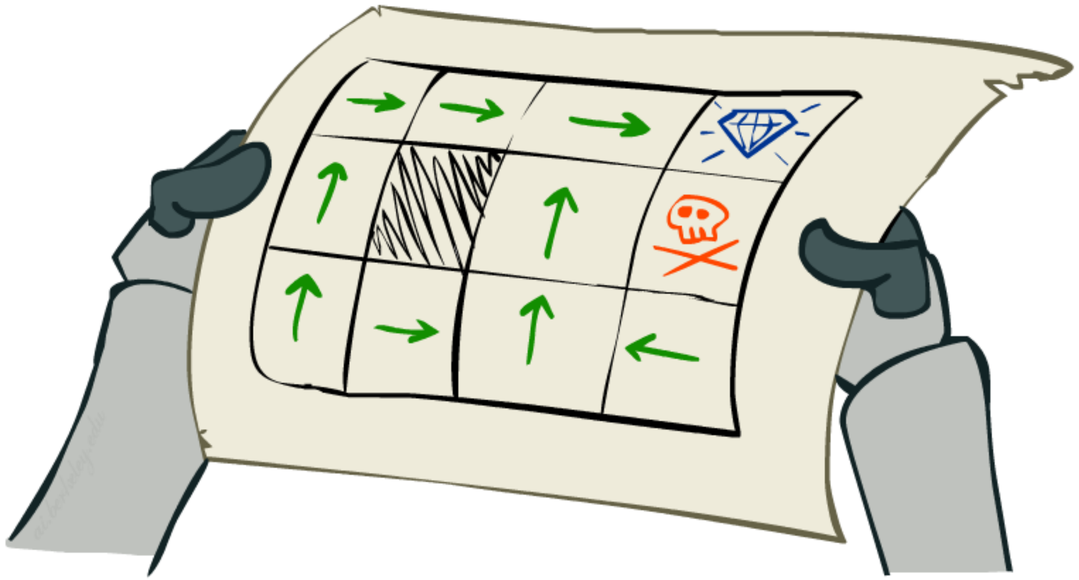


When to use RL?

- Data in the form of trajectories.
- Need to make a sequence of (related) decisions.
- Observe (partial, noisy) feedback to choice of actions.
- Tasks that require both learning and planning.

Policy Iteration 🔁

Policy tells agent how it should behave



Expected return

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

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

G_t – **future expected (discounted) return from time t**

R_{t+i} – **random variable, reward at time t + i**

γ – **discount factor**

1

Worth Now

γ

Worth Next Step

γ^2

Worth In Two Steps

State value

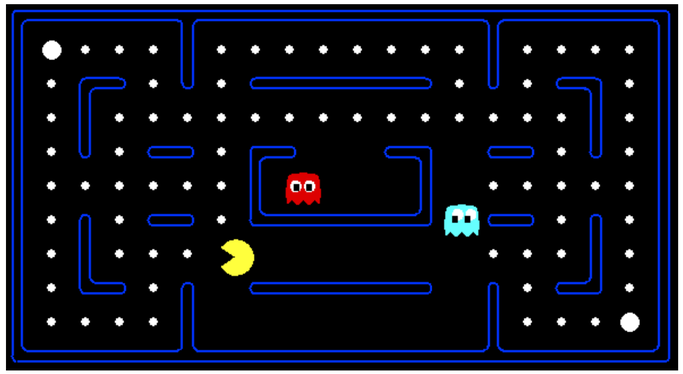
$$v_\pi(s) = \mathbf{E}_\pi[G_t \,|\, S_t = s] = R_{t+1} + R_{t+2} + R_{t+3} + \ldots = R_{t+1} + v_\pi(S')$$

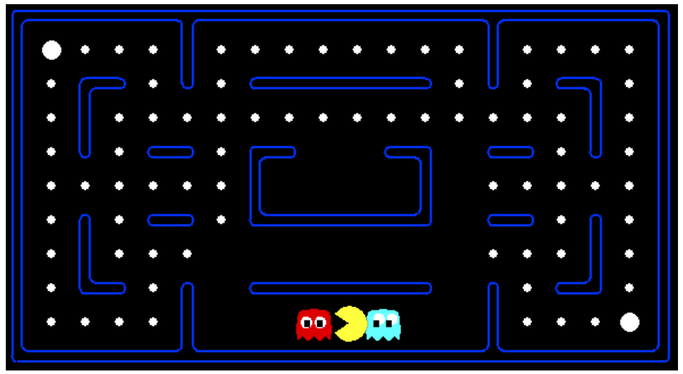
$$v_\pi(s) = R_{t+1} + v_\pi(S') = \sum_{s'} P(s, a, s')[r + v_\pi(s')], a = \pi(s), r = R(s, a, s')$$

π – **policy, decide on action a in state s**

P – **dynamics model, simulates environment**

R – **reward function, returns transition reward**





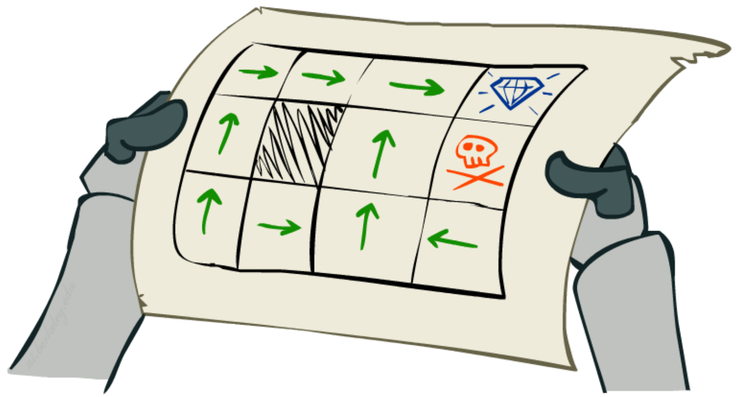
Policy evaluation

$$v_{k+1}(s) \doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s]$$

Policy improvement

$$
\begin{aligned}
\pi'(s) &\doteq \arg\max_a q_\pi(s,a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]
\end{aligned}
$$

q_π – **state-action quality function**



AlphaZero is a General Policy Iteration algorithm

Q-Learning

Learn Q-values

$$q_\pi(s,a) = \sum_{s'} P(s,a,s')[r + \max_{a'} q_\pi(s',a')], r = R(s,a,s')$$

Rewrite with expectation

$$q_\pi(s,a) = \mathbf{E}_{s' \sim P(s,a)}[R_{t+1} + \max_{a'} q(s',a') \,|\, S_t = s, A_t = a]$$

and...

Monte Carlo!

Learn Q-values

$$q_\pi(s,a) = \sum_{s'} P(s,a,s')[r + \max_{a'} q_\pi(s',a')],\ r = R(s,a,s')$$

Rewrite with expectation

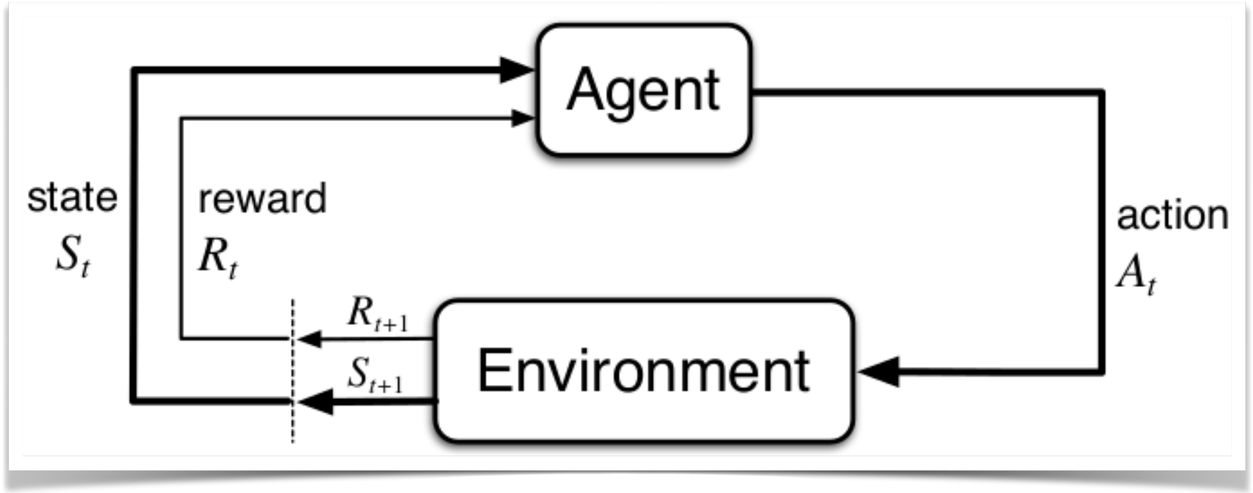
$$q_\pi(s,a) = \mathbf{E}_{s' \sim P(s,a)}[R_{t+1} + \max_{a'} q(s',a') \,|\, S_t = s, A_t = a]$$

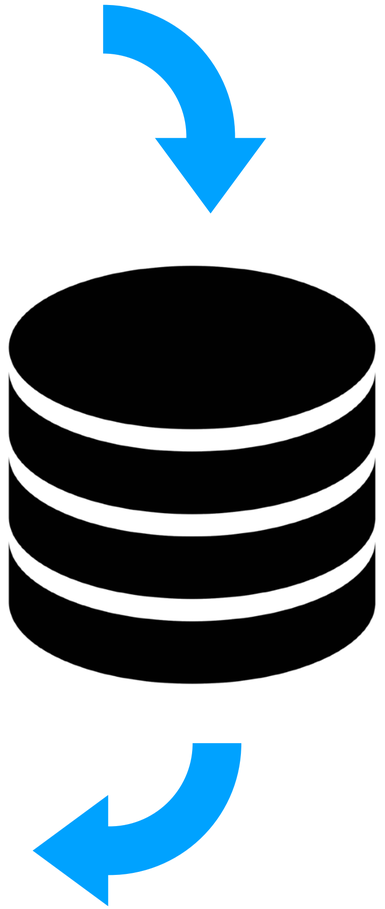
Monte Carlo running average

$$target = R(s,a,s') + \gamma \max_{a'} q_\pi(s',a')$$

$$q_{k+1} = (1-\alpha) \cdot q_k(s,a) + \alpha \cdot target$$

Q-Learning





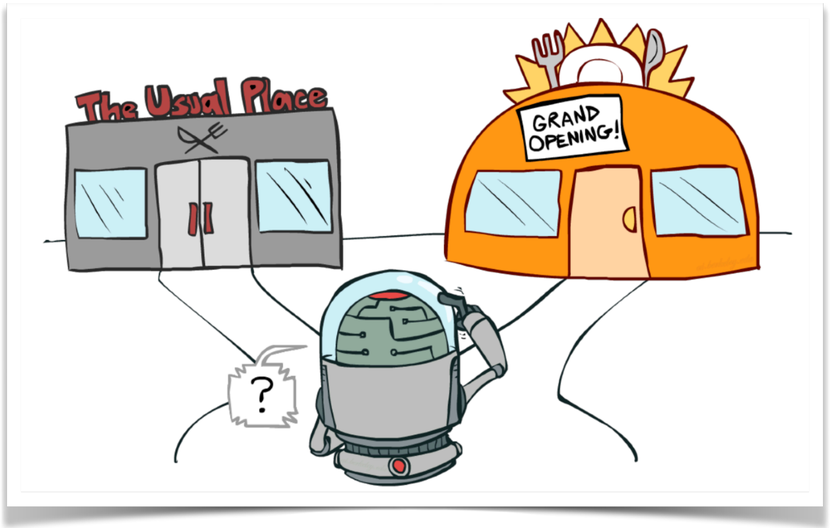
$$target = R(s, a, s') + \gamma \max_{a'} q_\pi(s', a')$$

$$q_{k+1} = (1 - \alpha) \cdot q_k(s, a) + \alpha \cdot target$$

$$\pi'(s) \doteq \arg\max_a q_\pi(s, a)$$

Exploration-exploitation

$$\pi'(s) \doteq \max q_\pi(s, a)$$



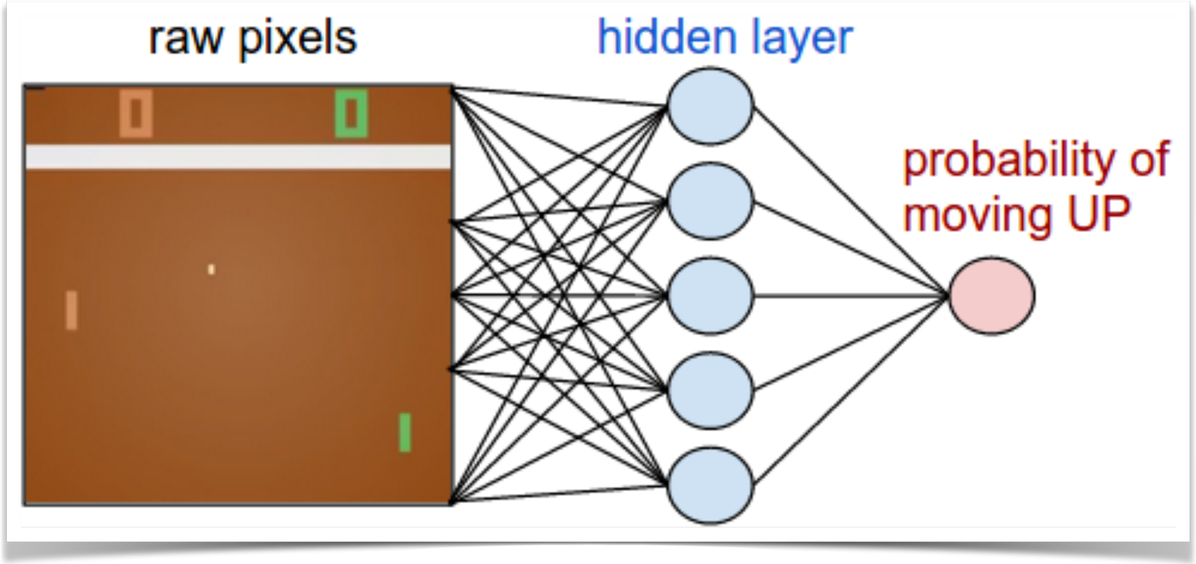
Off-policy learning: pros and cons

It doesn’t matter what actions you take (optimal or not). You “just” need to explore a lot to see possible states and what happens there. Q-value will converge and you can take *argmax* on it to get policy.

We train policy indirectly. At the end, exact state-action values aren’t needed to get optimal policy. What matters is ordering of actions. Policy often converges faster to the optimal one than Q-values.

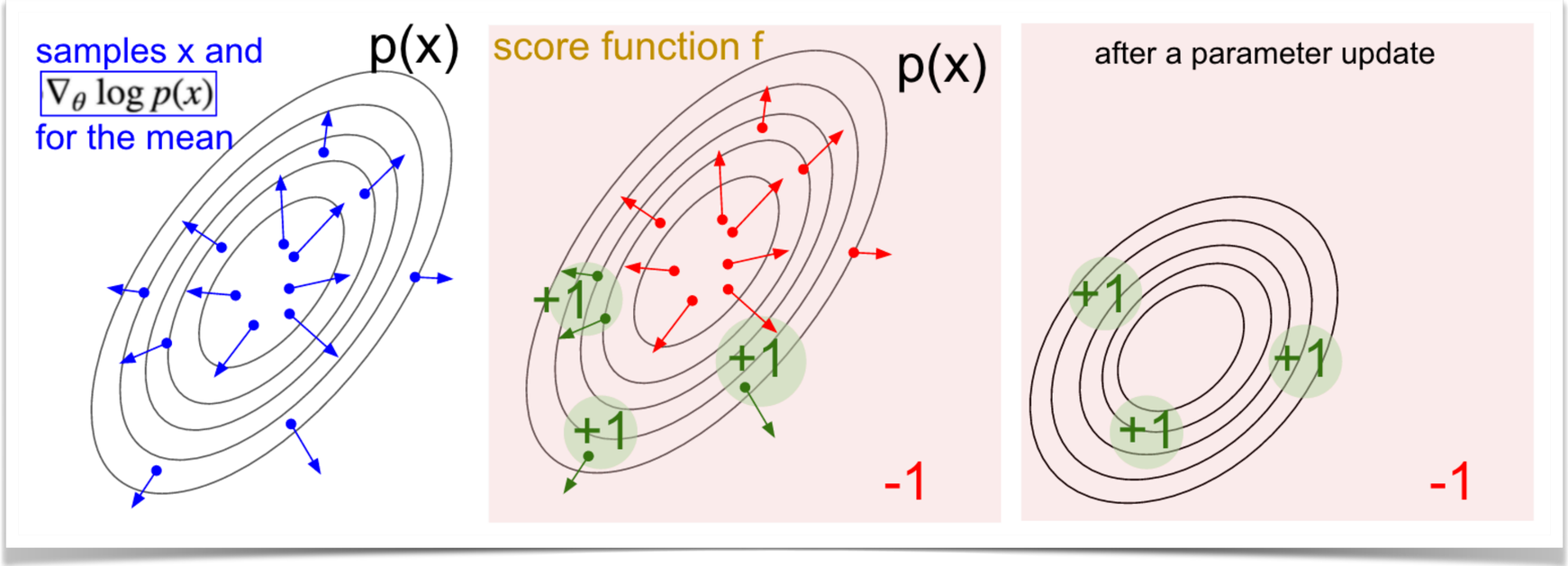
Policy Gradients 🎯

Why to learn some values, if all we care about is policy?



Our NN takes current game state, processes it and returns probabilities of particular actions (parametrised distribution over actions). Then we sample from this distribution to decide what action to take.

Plug-in reward as a label



What happens is we shift action distribution (which depends on current state) to areas where actions result in higher rewards. We encourage NN to take “good” actions and discourage it to take “bad” actions.

On-policy learning: pros and cons

We train NN policy directly, that’s great! But also we learn on-policy, it means that what we do has high impact on what we learn.

If you e.g. has too big learning rate and you will take too big step in parameter space (remember that we work with estimates of true gradient!) in some seemingly promising direction, it can turn out that you jumped right into worse region of our distribution and you are doing worse then before! There is know easy way back, you have to learn again how to play better.

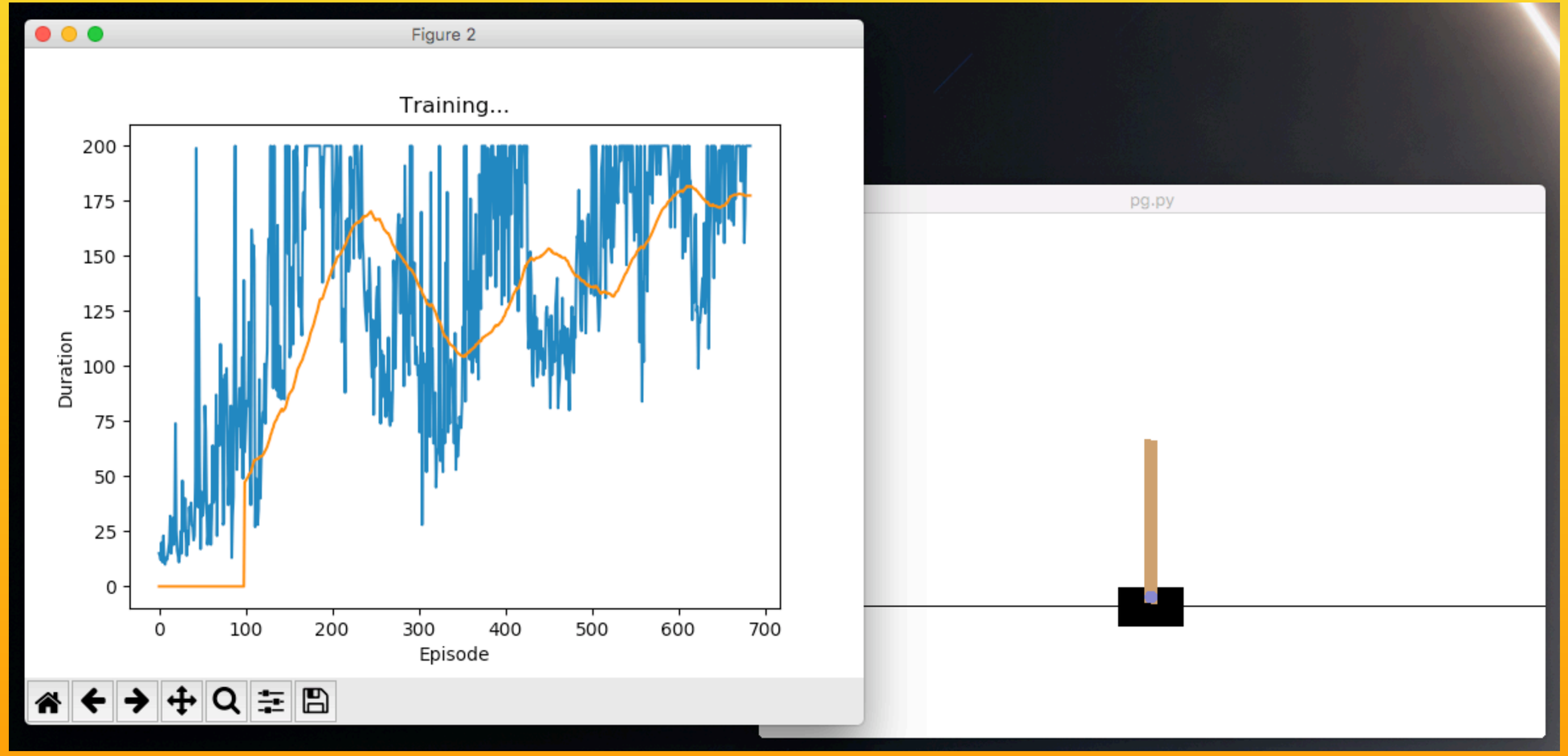


Figure 2
Training...
Duration
Episode
0
25
50
75
100
125
150
175
200
0
100
200
300
400
500
600
700
pg.py

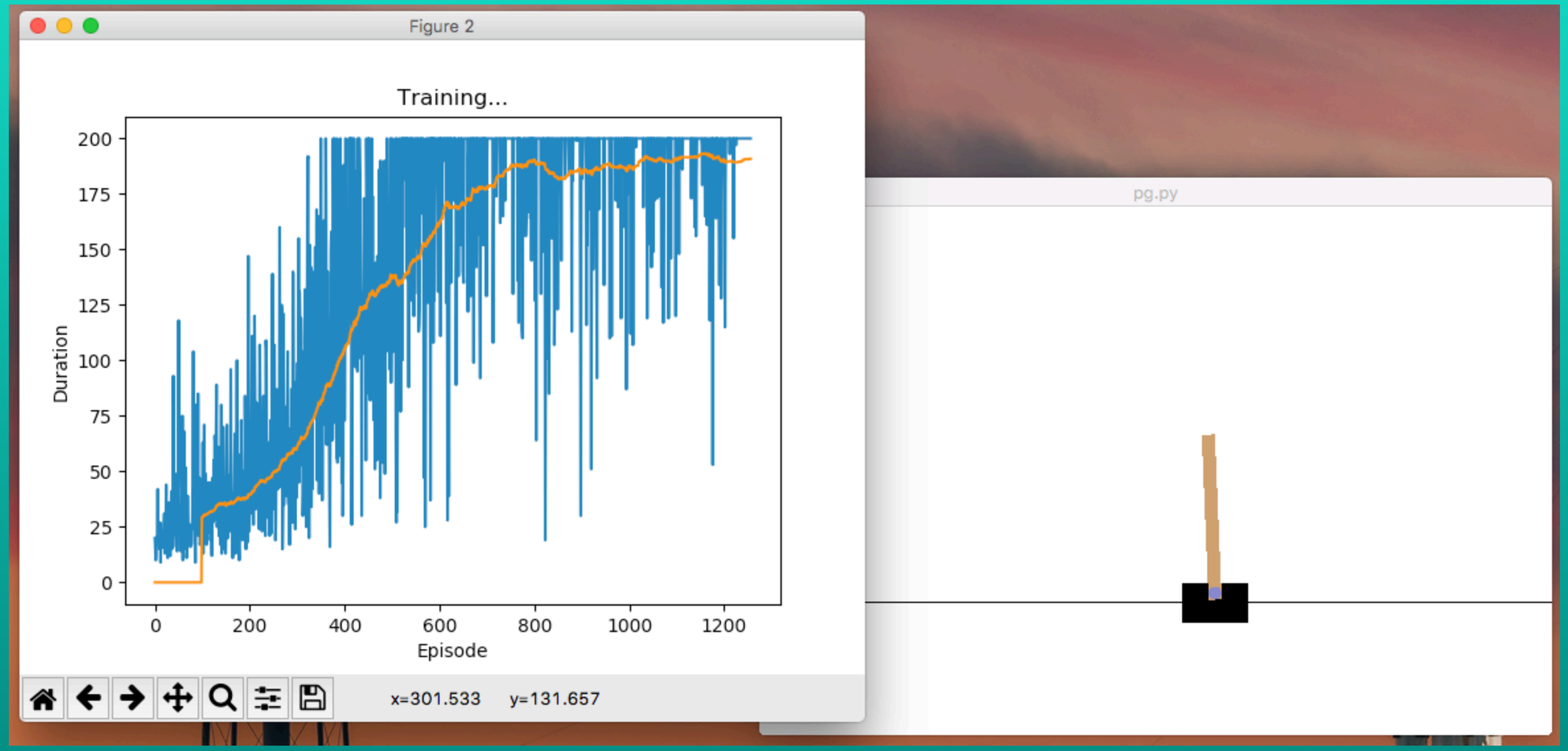


Figure 2
Training...
Duration
Episode
0
25
50
75
100
125
150
175
200
0
200
400
600
800
1000
1200
x=301.533
y=131.657
pg.py

Further reading/watching

- **Deep Reinforcement Learning: Pong from Pixels by Andrej Karpathy**
- **Reinforcement Learning: An Introduction 2nd edition**
- **Reinforcement Learning course by David Silver**
- AlphaGo Zero - How and Why it Works by Tim Wheeler
- CS188 Intro to AI, Berkeley
- Deep RL Bootcamp, Berkeley 2017
- Deep Learning and Reinforcement Learning Summer School, Montreal 2017
- Deep Learning and Reinforcement Learning Summer School, Toronto 2018
- AlphaGo movie on Netflix 📽️

Dynamic Programming

in Colab