Brain-Inspired Artificial Intelligence 5: Introduction to Deep Reinforcement Learning

Eiji Uchibe

Dept. of Brain Robot Interface

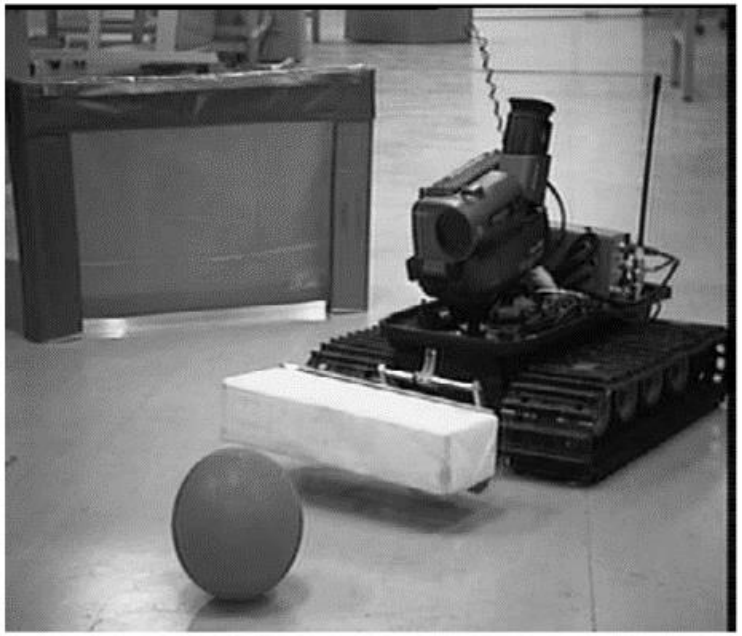
ATR Computational Neuroscience Labs.

Large-Scale Reinforcement Learning

- So far we have assumed that states and actions were discrete and it was possible to represent value functions and policy by a lookup table
- It is NOT true for realistic tasks
  - Go: 10^{170} states
  - Helicopter/Mountain Car: continuous state space
  - Robots: informal state space

Discretizing State Space

- When a state is given by a continuous vector, it is discretized manually
  - Soccer robot whose task is to shoot a ball into a goal



http://rraj.rsj-web.org/ja_history

M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. (1996). Purposive behavior acquisition for a real robot by vision-based reinforcement learning. Machine Learning, 23, 279-303.

Discretizing State Space

- 5 state variables: x-position and size of the ball and x-position, size and orientation of the goal

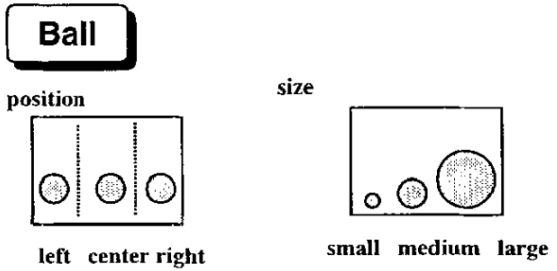
- The number of states grows exponentially as the number of features increases

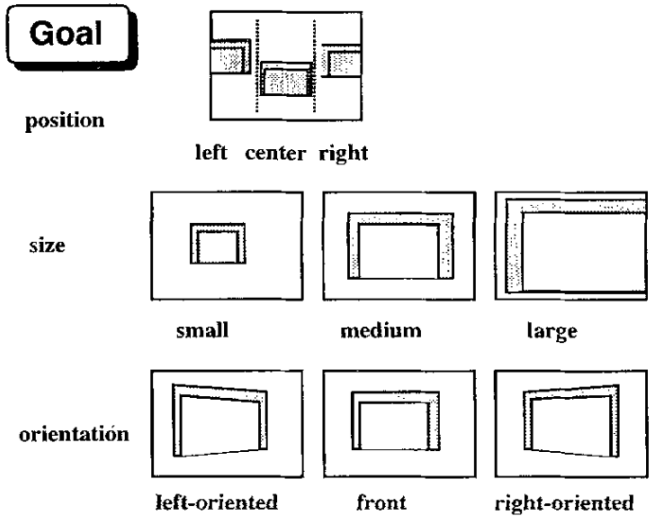
$$3^2 \times 3^3 = 243$$

the number of ball states

the number of goal states

the total number of states





- So we need to approximate value functions and policy

M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. (1996). Purposive behavior acquisition for a real robot by vision-based reinforcement learning. Machine Learning, 23, 279-303.

Function approximation

- Reminder: Update rule of Q-learning for discrete systems

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta_t$$
$$\delta = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)$$

- If $Q(s, a)$ is approximated by some function $\hat{q}(s, a, \boldsymbol{w})$ parameterized by $\boldsymbol{w}$, the update rule is given by

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\delta\nabla_{\boldsymbol{w}}\hat{q}(s, a, \boldsymbol{w})$$
$$\delta = R_{t+1} + \gamma \max_{a'} \hat{q}(s_{t+1}, a', \boldsymbol{w}) - \hat{q}(s_t, a_t, \boldsymbol{w})$$

- What kind of approximators should we use?

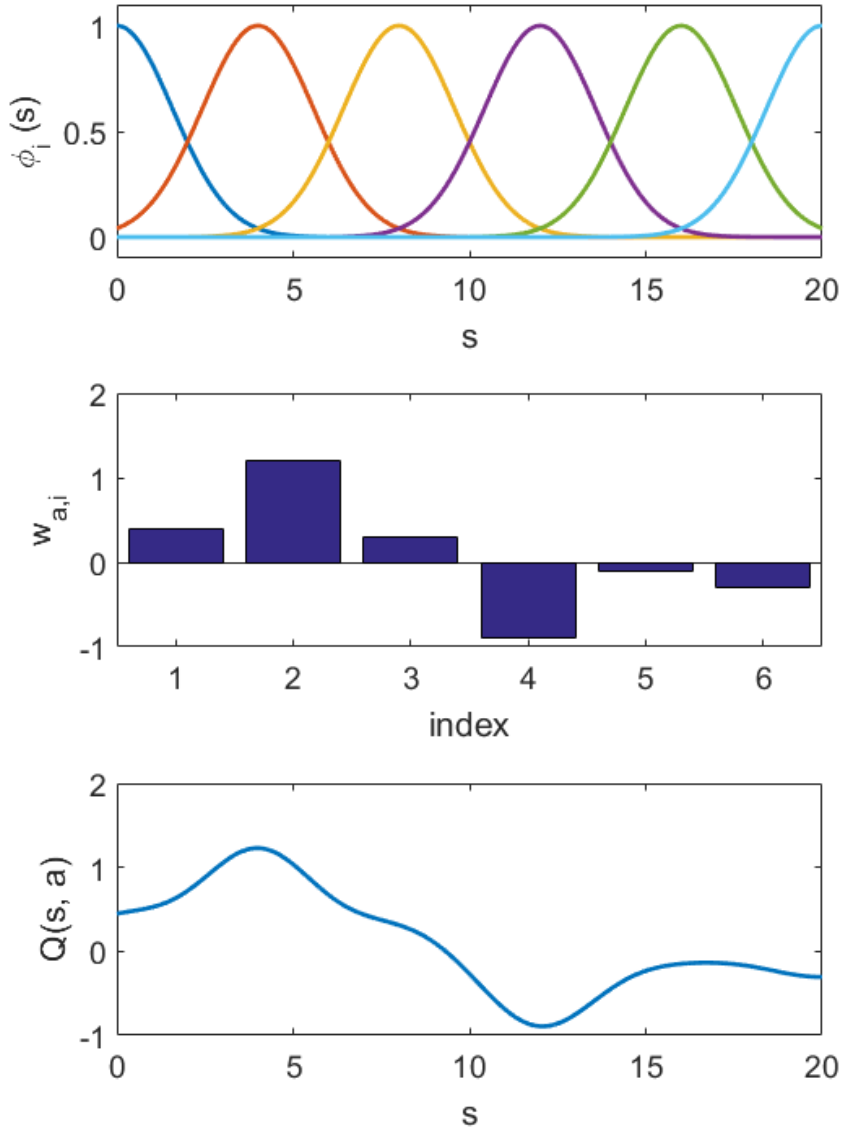
Linear Function Approximation

- A linear function approximator is introduced to deal with continuous states and discrete actions

$$\hat{Q}(s, a, \boldsymbol{w}) = \boldsymbol{w}_a^{\mathsf{T}} \boldsymbol{\phi}(s)$$

  - $\boldsymbol{w}_a$: weight parameter vector for action a
  - $\boldsymbol{w} = \{\boldsymbol{w}_a\}$
  - $\boldsymbol{\phi}(s)$: basis function vector

- Under some assumptions, convergence proof is given



Failure of Nonlinear Function Approximation

- Task: Mountain-Car
  - Driving an underpowered car up a steep mountain road
- V^{π} is approximated by a neural network with 80 hidden units
- Value iteration results in divergence

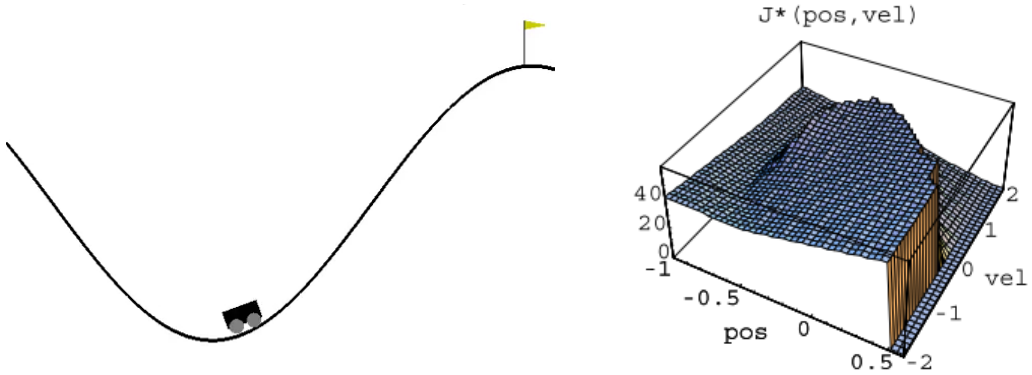


Figure 5: The car-on-the-hill domain. When the velocity is below a threshold, the car must reverse up the left hill to gain enough speed to reach the goal, so J^* is discontinuous.

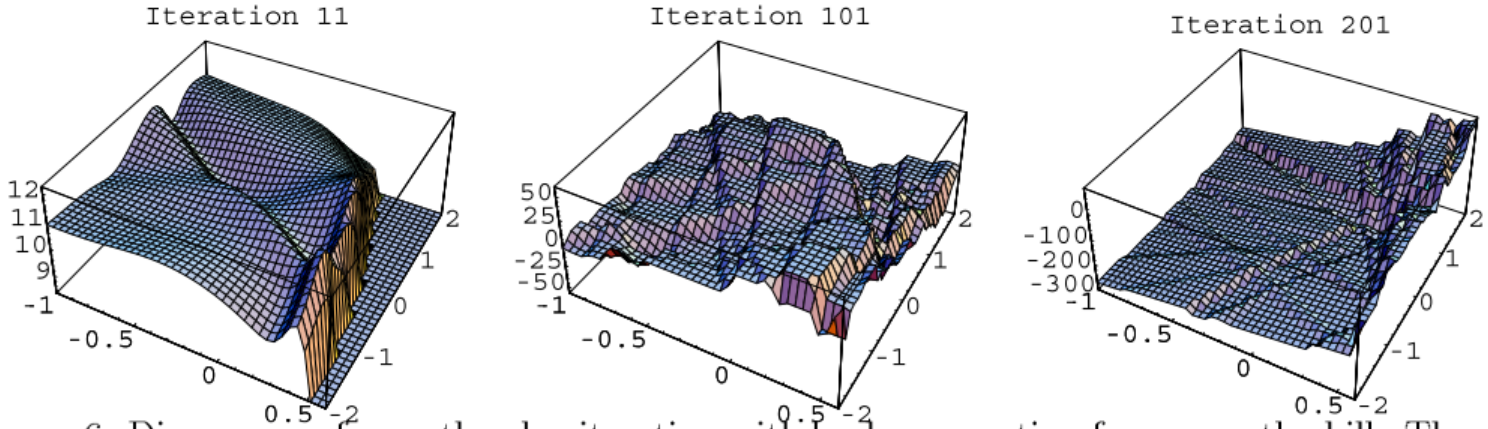
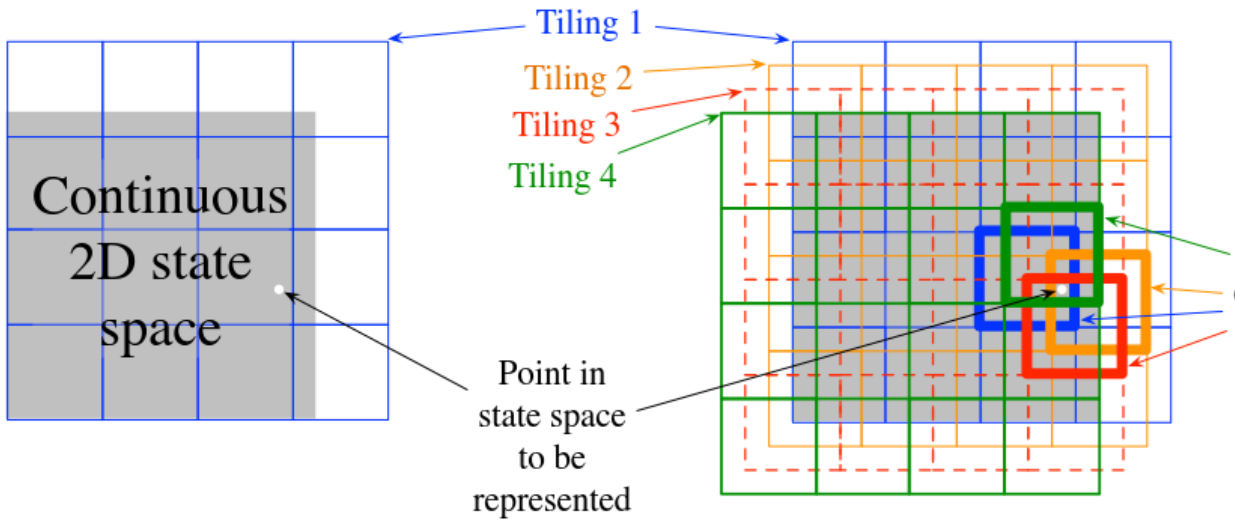


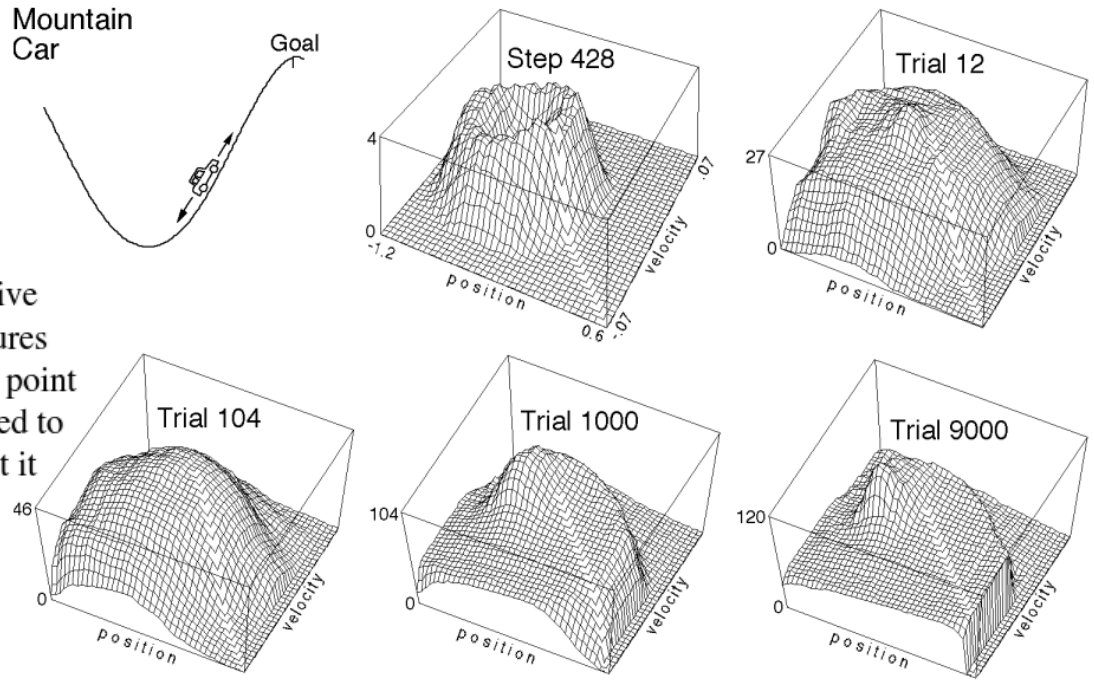
Figure 6: Divergence of smooth value iteration with backpropagation for car-on-the-hill. The neural net, a 2-layer MLP with 80 hidden units, was trained for 2000 epochs per iteration.

J.A. Boyan, and A.W. Moore. (1995). Generalization in Reinforcement Learning: Safely Approximating the Value Function. NIPS 7, pp.369-376.

Success of Linear Function Approximation

- Tile coding successfully approximated the optimal value function because basis functions are localized



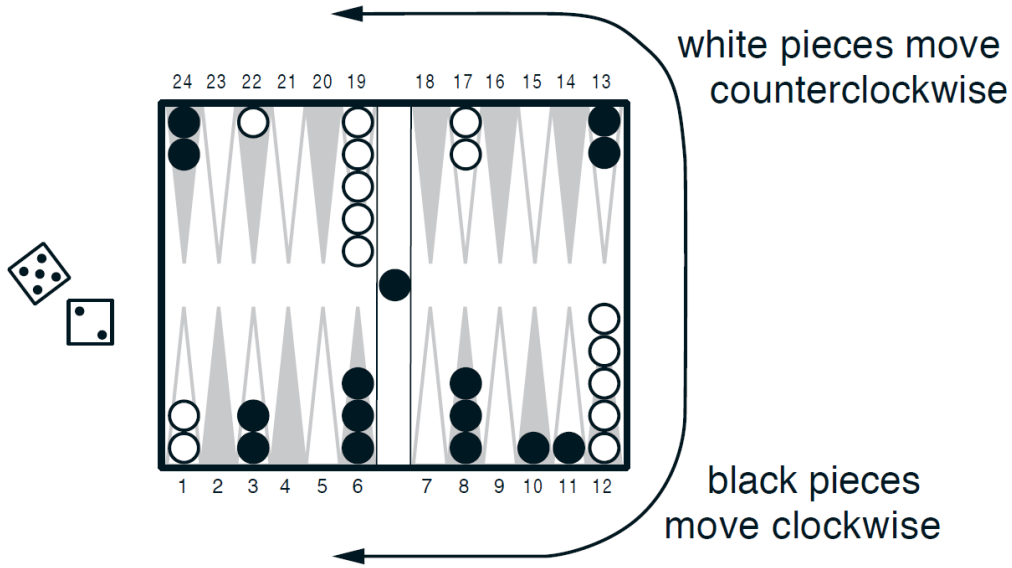


- Difficult to apply tile coding to a high-dimensional state space

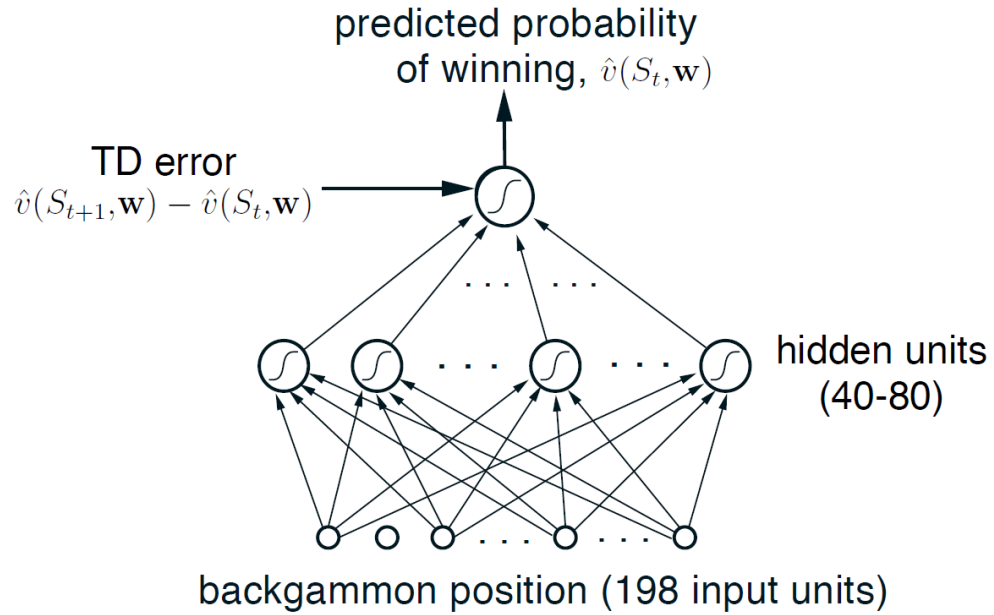
R.S. Sutton. (1996). Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. NIPS 8, pp. 1038-1044.

1991-95: TD-gammon

- RL backgammon program
- The big early success of RL + NN
- 40-80 sigmoid hidden units
- Up to 1,500,000 games of self-play
- Delayed rewards at end of the game
- At or near best human player level



A backgammon position



G. Tesauro. (1992). Practical issues in temporal difference learning. Machine Learning. Vol. 8, no. 3, pages 257-277.
R. Sutton and A.G. Barto. (2018). Reinforcement Learning: An Introduction. (2nd edition). MIT Press.

TD-Gammon results

Program	Hidden units	Training games	Opponents	Results
TD-Gammon 0.0	40	300,000	other programs	tied for best
TD-Gammon 1.0	80	300,000	Robertie, Magriel, ...	-13 pts / 51 games
TD-Gammon 2.0	40	800,000	various grandmasters	-7 pts / 38 games
TD-Gammon 2.1	80	1,500,000	Robertie	-1 pt / 40 games
TD-Gammon 3.0	80	1,500,000	Kazaros	+6 pts / 20 games

G. Tesauro. (1992). Practical issues in temporal difference learning. Machine Learning. Vol. 8, no. 3, pages 257-277.
R. Sutton and A.G. Barto. (2018). Reinforcement Learning: An Introduction. (2nd edition). MIT Press.

1995-2013: Almost nothing

- Rich Sutton’s Q&A (2001-04):

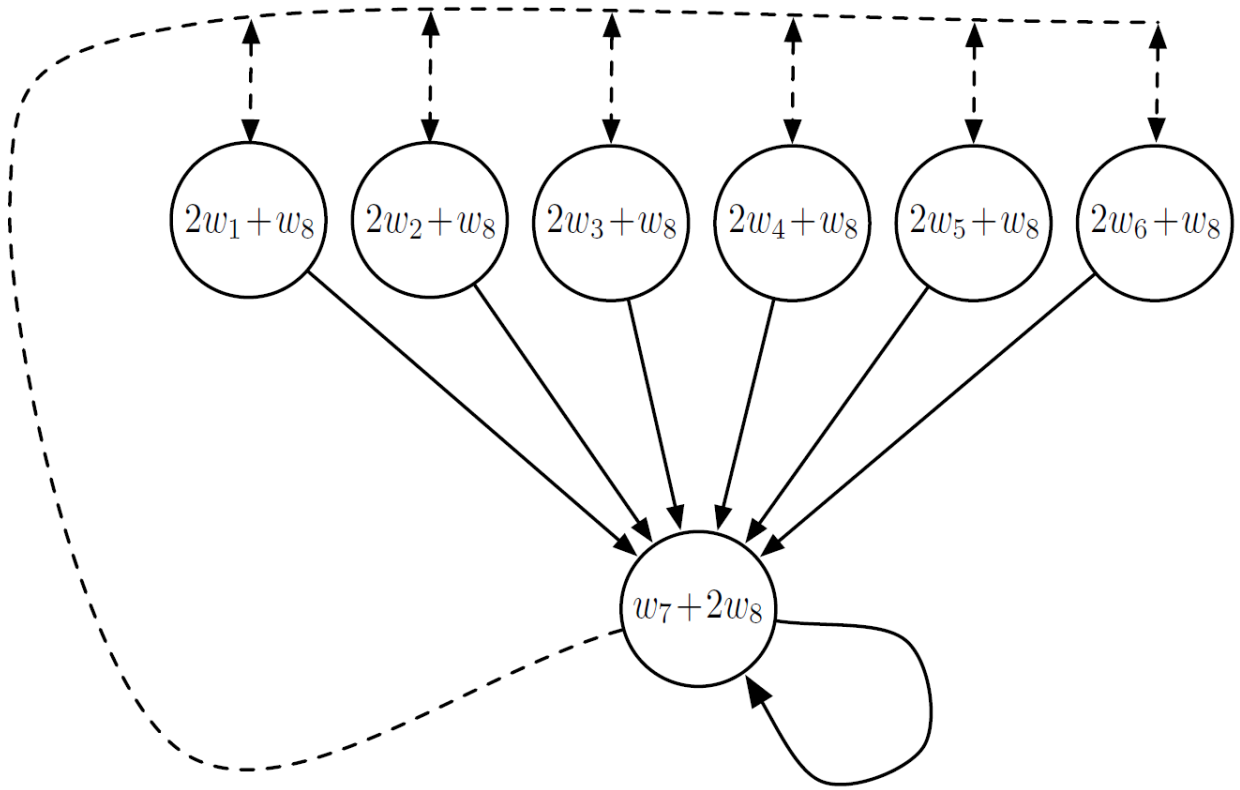
I am doing RL with a backpropagation neural network and it doesn't work; what should I do?

It is a common error to use a backpropagation neural network as the function approximator in one's first experiments with reinforcement learning, which almost always leads to an unsatisfying failure. The primary reason for the failure is that backpropation is fairly tricky to use effectively, doubly so in an online application like reinforcement learning. It is true that Tesauro used this approach in his strikingly successful backgammon application, but note that at the time of his work with TD-gammon, Tesauro was already an expert in applying backprop networks to backgammon. He had already built the world's best computer player of backgammon using backprop networks. He had already learned all the tricks and tweaks and parameter settings to make backprop networks learn well. Unless you have a similarly extensive background of experience, you are likely to be very frustrated using a backprop network as your function approximator in reinforcement learning.

Why no follow-up to TD-Gammon?

- Considered a special case:
  - Games always end after a reasonable number of episodes
  - Natural exploration (dice)
  - Tesauro's special NN skills
- Subsequent failures in other games
- Lack of theoretical proofs of convergence
  - E.g., Baird's counterexample shows parameter divergence for off-policy learning

Baird's counterexample



$\pi(\text{solid}|\cdot) = 1$

$b(\text{dashed}|\cdot) = 6/7$

$b(\text{solid}|\cdot) = 1/7$

$\gamma = 0.99$

Figure 11.1: Baird's counterexample. The approximate state-value function for this Markov process is of the form shown by the linear expressions inside each state. The solid action usually results in the seventh state, and the dashed action usually results in one of the other six states, each with equal probability. The reward is always zero.

Baird's counterexample

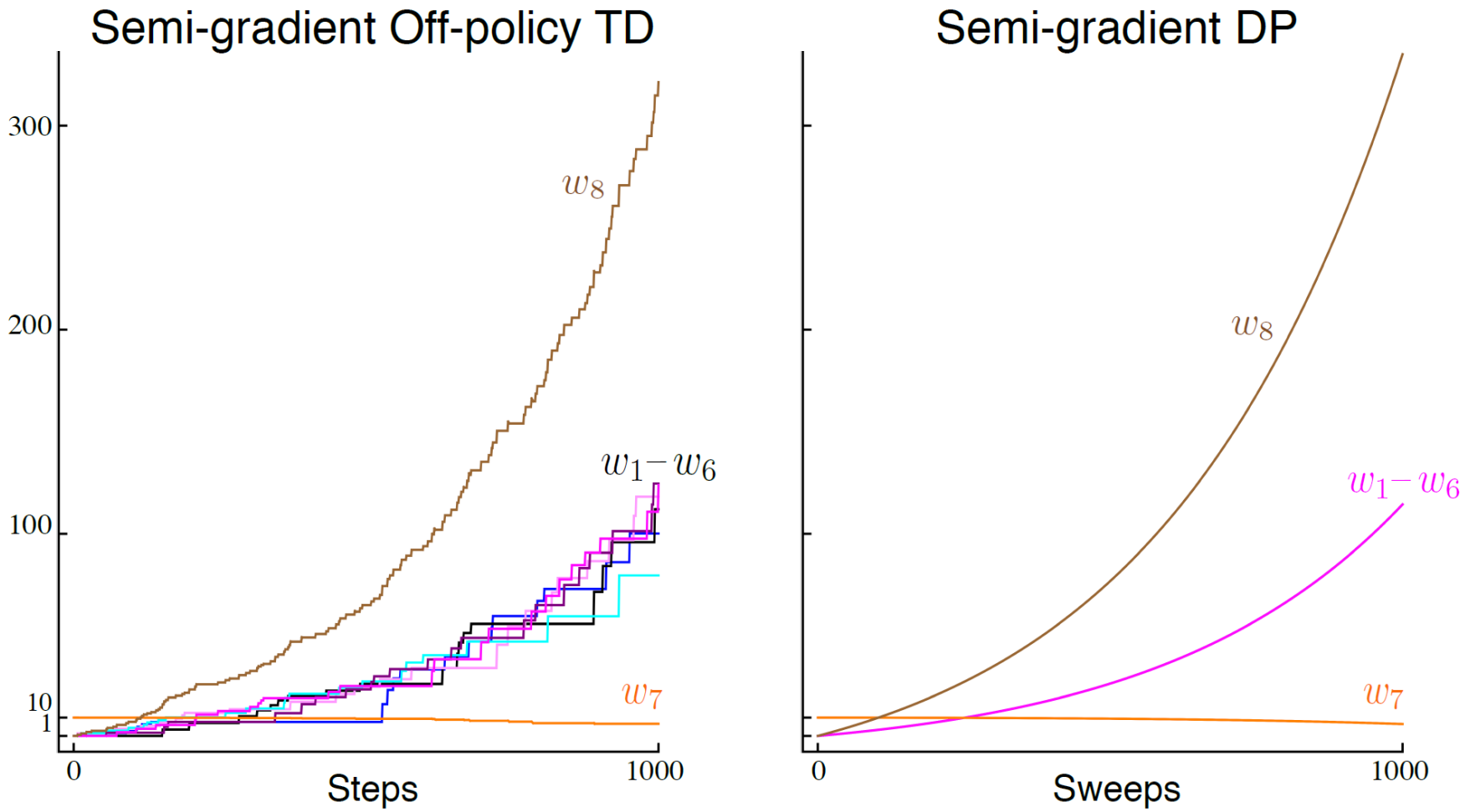
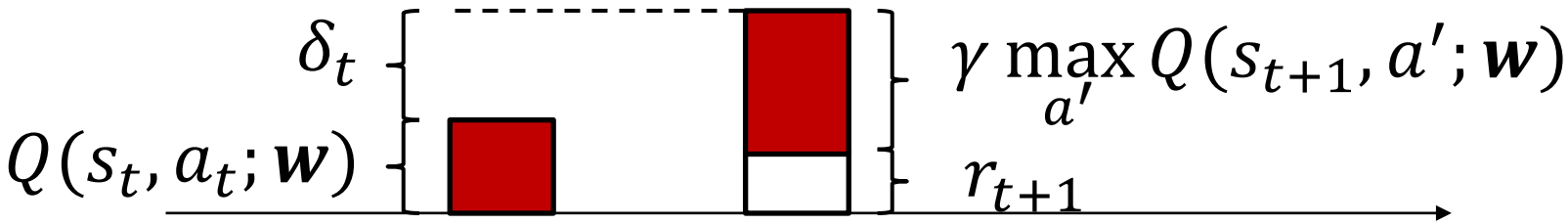


Figure 11.2: Demonstration of instability on Baird's counterexample. Shown are the evolution of the components of the parameter vector $\mathbf{w}$ of the two semi-gradient algorithms. The step size was $\alpha = 0.01$, and the initial weights were $\mathbf{w} = (1, 1, 1, 1, 1, 1, 10, 1)^\top$.

Why are nonlinear approximators unstable

- The target value changes when the parameters are updated!

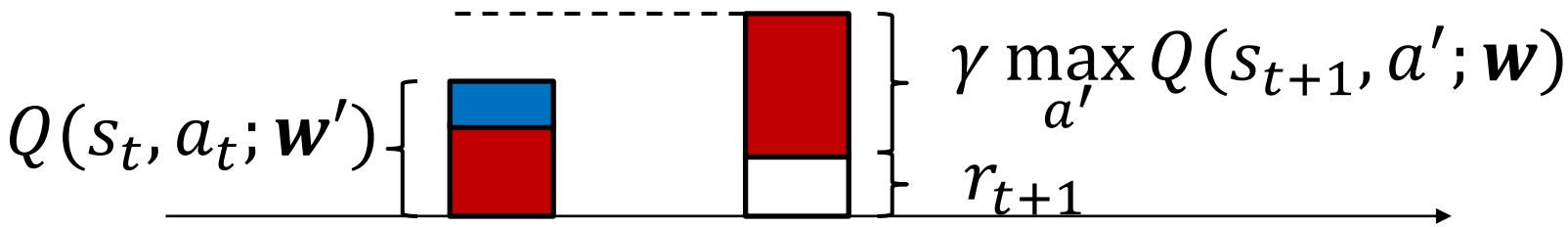
$$\delta_t = r_{t+1} + \gamma \max_{a'} \tilde{Q}(s_{t+1}, a'; \boldsymbol{w}) - \tilde{Q}(s_t, a_t; \boldsymbol{w})$$



Why are nonlinear approximators unstable

- The target value changes when the parameters are updated!

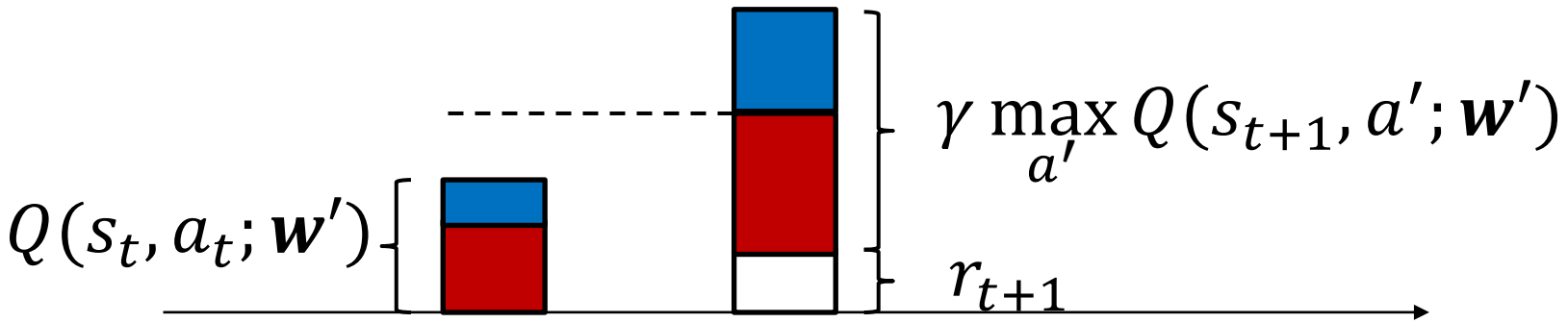
$$\delta_t = r_{t+1} + \gamma \max_{a'} \tilde{Q}(s_{t+1}, a'; \boldsymbol{w}) - \tilde{Q}(s_t, a_t; \boldsymbol{w})$$



Why are nonlinear approximators unstable

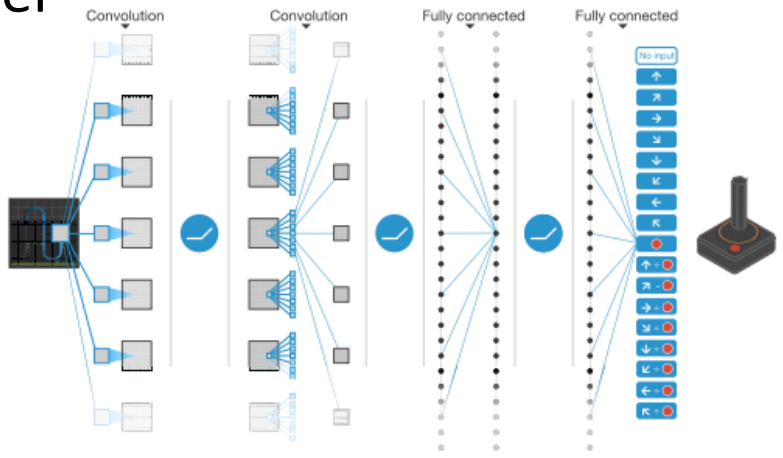
- The target value changes when the parameters are updated!

$$\delta_t = r_{t+1} + \gamma \max_{a'} \tilde{Q}(s_{t+1}, a'; \boldsymbol{w}) - \tilde{Q}(s_t, a_t; \boldsymbol{w})$$



2013-15: Deep Q network (DQN)

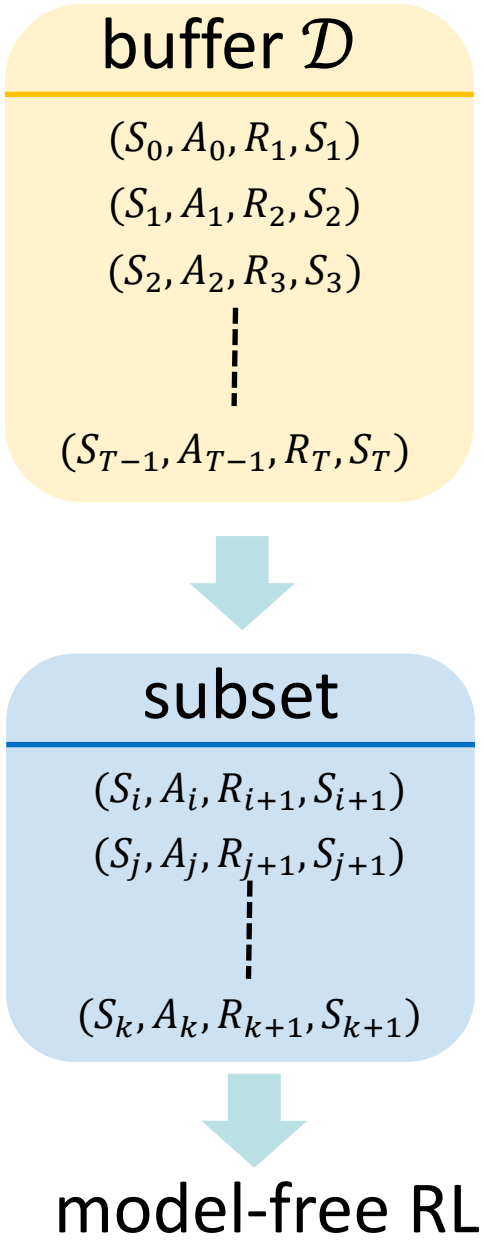
- DQN achieves remarkable success in computer games by learning deeply encoded representation from convolution networks
- What are the key issues in DQN?
  - Neural fitted Q iteration: Convert reinforcement learning to supervised learning
  - Experience replay: Store experiences and reuse them later
  - (Clipping the values of rewards)



V. Mnih et al. (2015). Human-level control through deep reinforcement learning. Nature, vol. 518, no. 7540, pp. 529–533.

Experience Replay

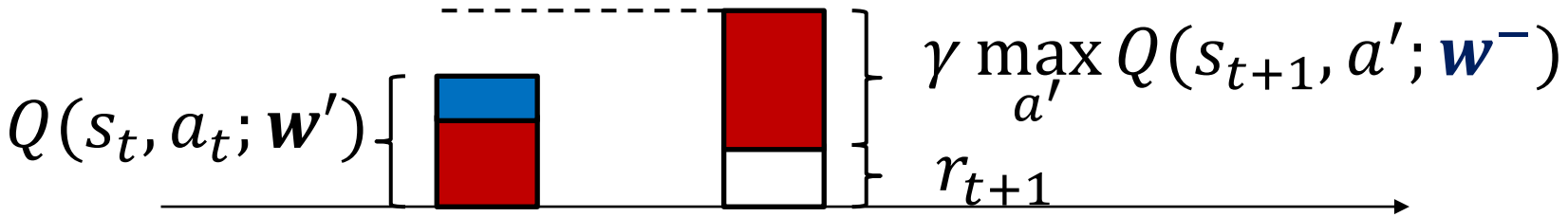
- Experienced transitions are stored in the buffer $\mathcal{D}$
- Same transitions are repeatedly sampled from $\mathcal{D}$ and use off-policy RL algorithms
- Uniform sampling is used
  - If there are N experienced transitions, the probability to select i-th transition is $1/N$
- One of widely used techniques in modern RL studies



L.-J. Lin. (1991). Programming Robots Using Reinforcement Learning and Teaching. In Proc. of AAAI, pages 781-786.
V. Mnih et al. (2015). Human-level control through deep reinforcement learning. Nature, vol. 518, no. 7540, pp. 529–533.

Neural fitted Q-iteration

- Two action-value functions are maintained
  - $Q(s, a, \boldsymbol{w}^-)$: Target Q function to compute the target value
  - $Q(s, a, \boldsymbol{w})$: Learning Q function to be trained
- Fix the target Q-function for a while when computing TD error



$$\delta_t = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'; \boldsymbol{w}^-) - Q(s_t, a_t; \boldsymbol{w})$$

M. Riedmiller. (2005). Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method. In Proc. of ECML, pages 317-328.

V. Mnih et al. (2015). Human-level control through deep reinforcement learning. Nature, vol. 518, no. 7540, pp. 529–533.

Neural fitted Q-iteration

- Frequency to update the target network is critical

  ⇒ Fast, but unstable learning for frequent update

  ⇒ Stable, but slow learning when we rarely update $\boldsymbol{w}^{-} \leftarrow \boldsymbol{w}$

M. Riedmiller. (2005). Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method. In Proc. of ECML, pages 317-328.
V. Mnih et al. (2015). Human-level control through deep reinforcement learning. Nature, vol. 518, no. 7540, pp. 529–533.

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$,T **do**

With probability ε select a random action a_t

otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

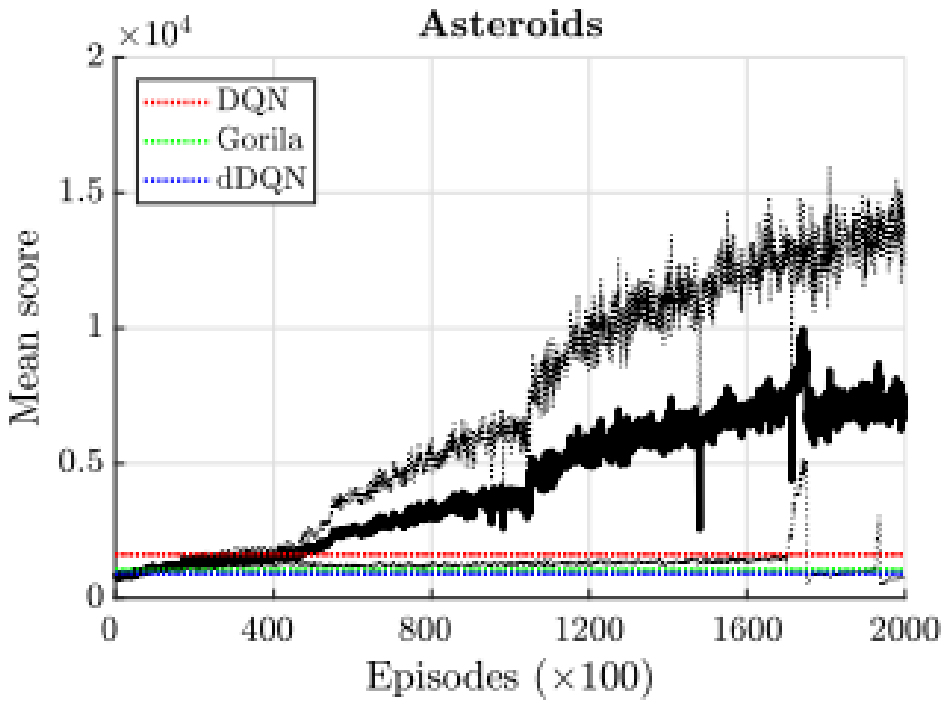
Every C steps reset $\hat{Q} = Q$

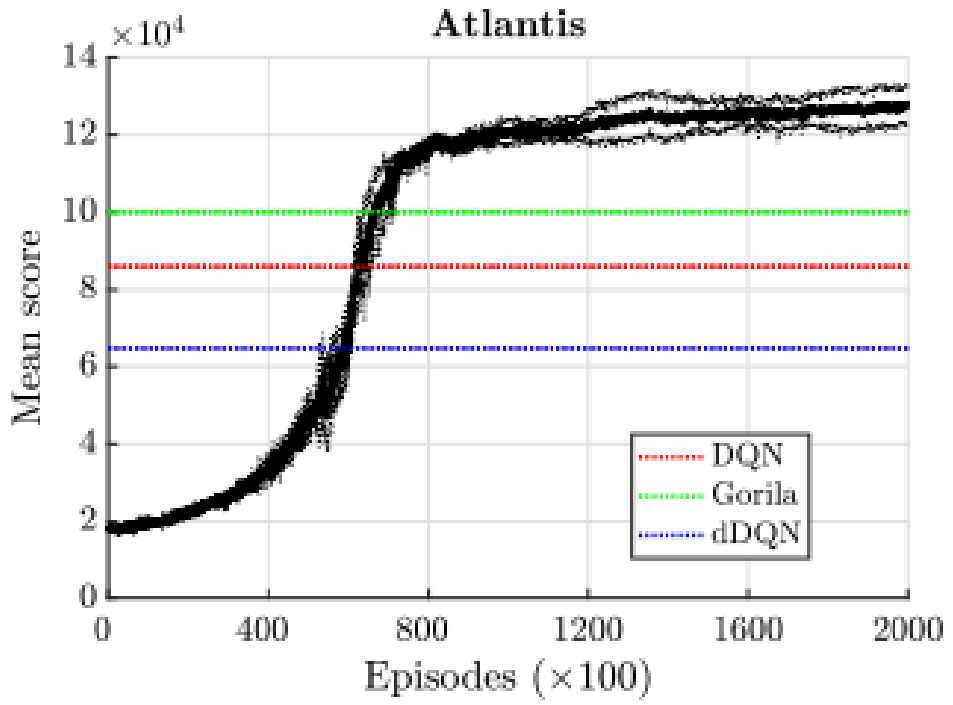
End For

End For

None: On-policy deep reinforcement learning

- However, it is reported that on-policy learning such as SARSA also achieved better performance than DQN

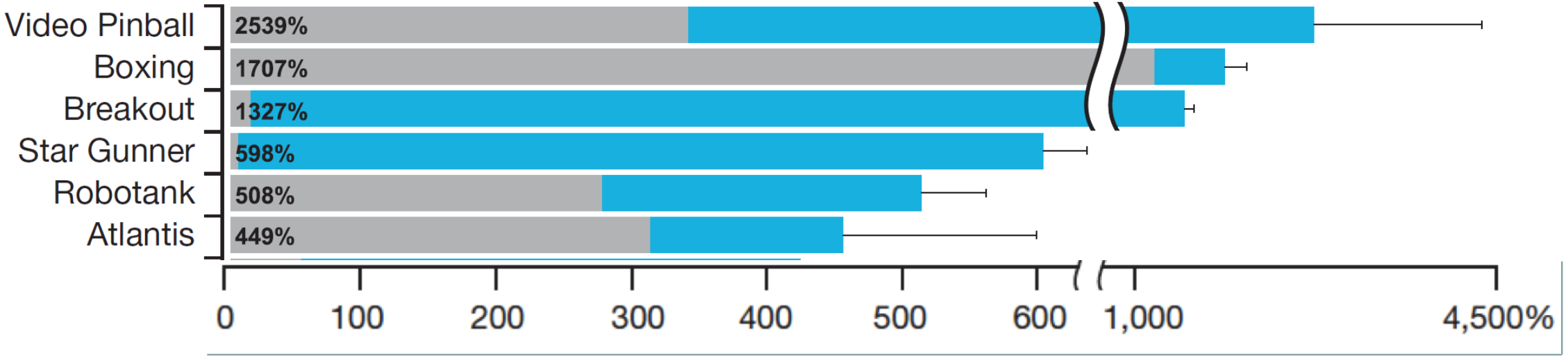




S. Elfwing, E. Uchibe, and K. Doya. (2018). Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning. Neural Networks. Vol. 107, pages 3-11.

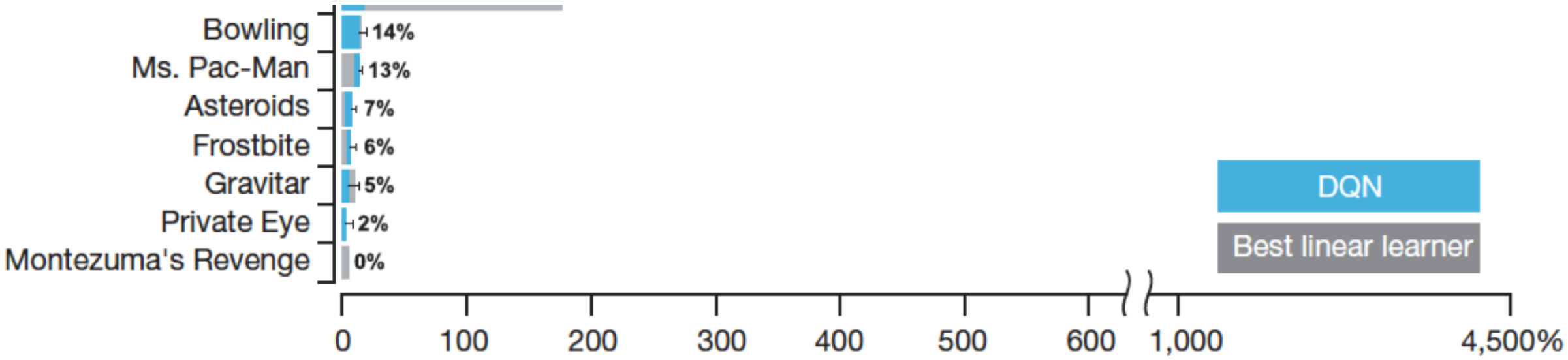
DQN in Atari: good games

- Very good in reactive games
  - Full state is visible and no longer planning required



DQN in Atari: bad games

- Not good in games that require
  - Longer term planning
  - More advanced exploration



DQN: Importance of target network and replay

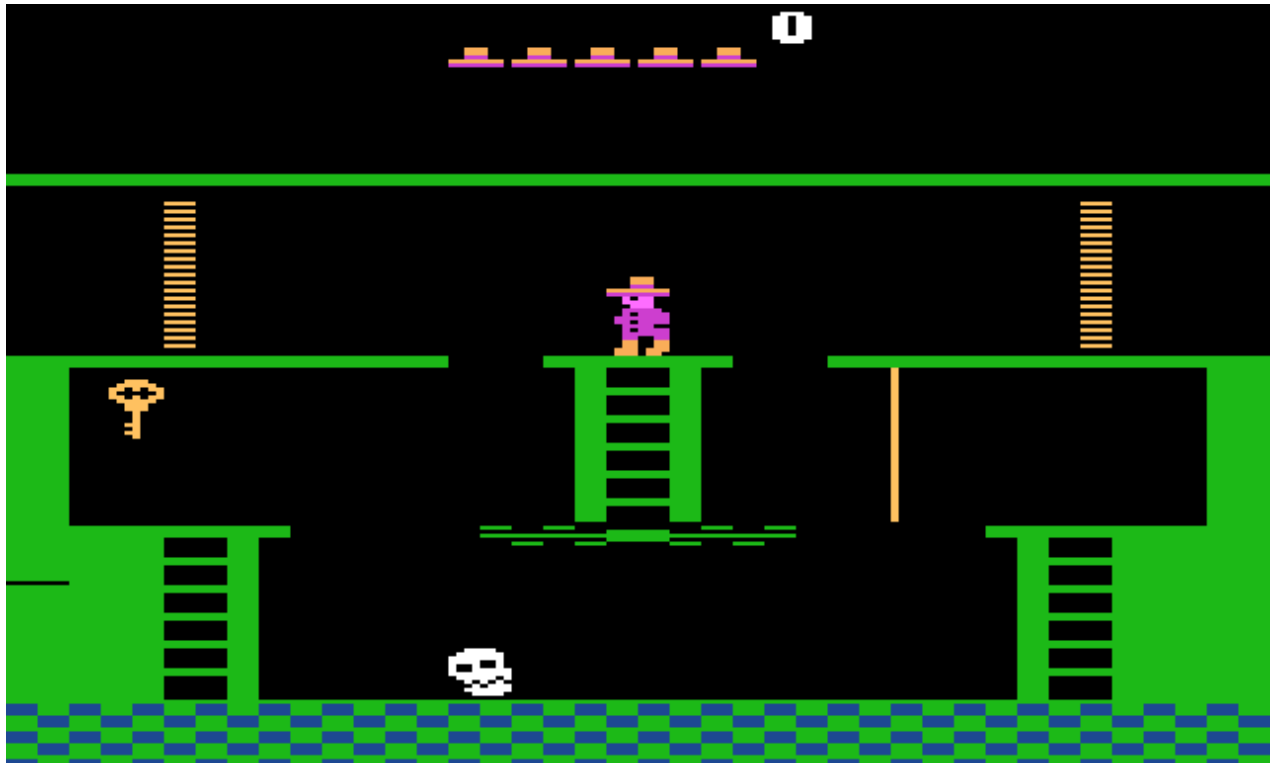
Extended Data Table 3 | The effects of replay and separating the target Q-network

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

DQN agents were trained for 10 million frames using standard hyperparameters for all possible combinations of turning replay on or off, using or not using a separate target Q-network, and three different learning rates. Each agent was evaluated every 250,000 training frames for 135,000 validation frames and the highest average episode score is reported. Note that these evaluation episodes were not truncated at 5 min leading to higher scores on Enduro than the ones reported in Extended Data Table 2. Note also that the number of training frames was shorter (10 million frames) as compared to the main results presented in Extended Data Table 2 (50 million frames).

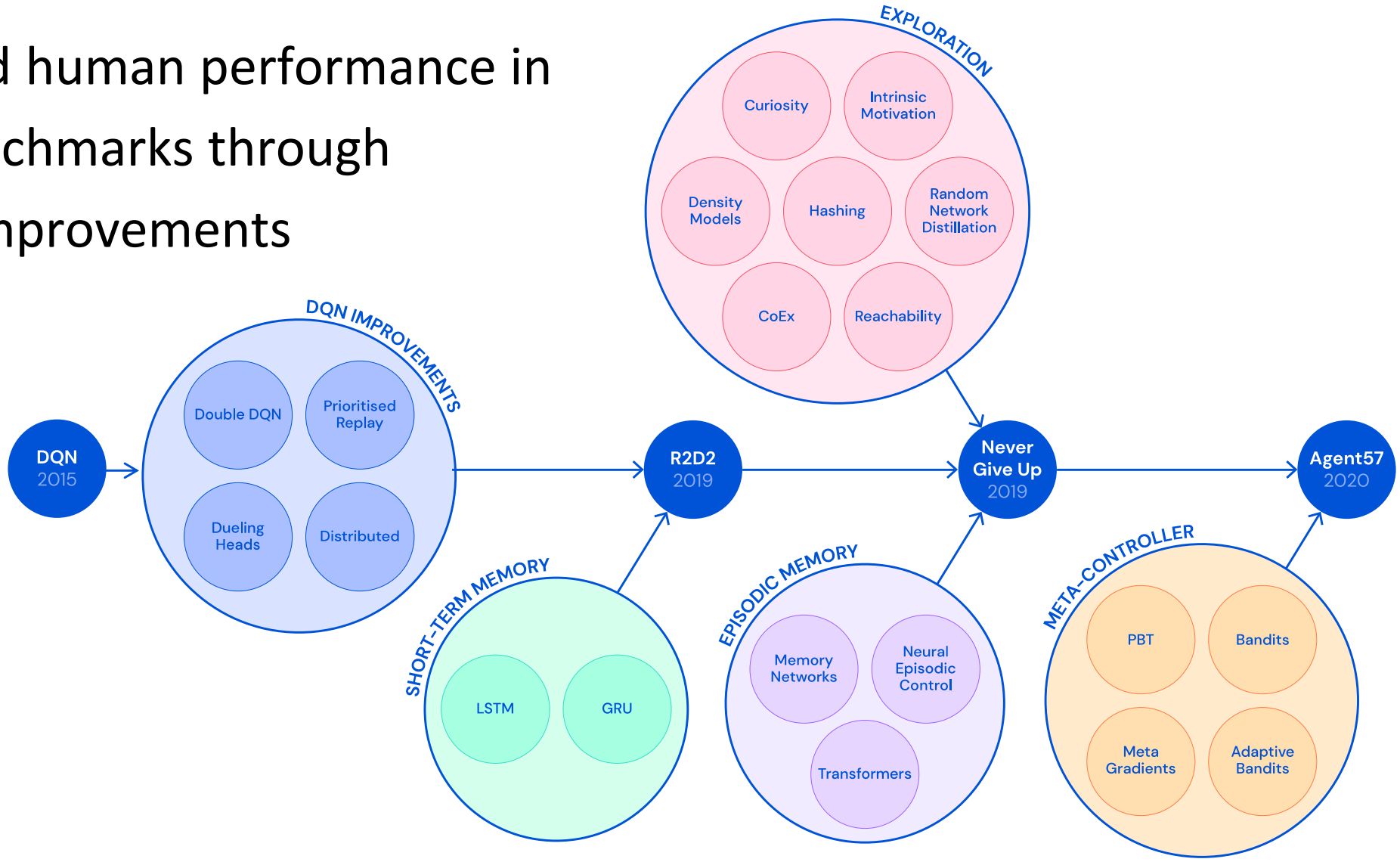
Montezuma's revenge

- Random exploration
  - ~0 reward
- Very long delayed rewards
- Have to do things in exact order
  - Eg, get key to exit screen
- New screen
  - complete new environment
- Requires memory
  - Part of screen played in darkness until torch is found



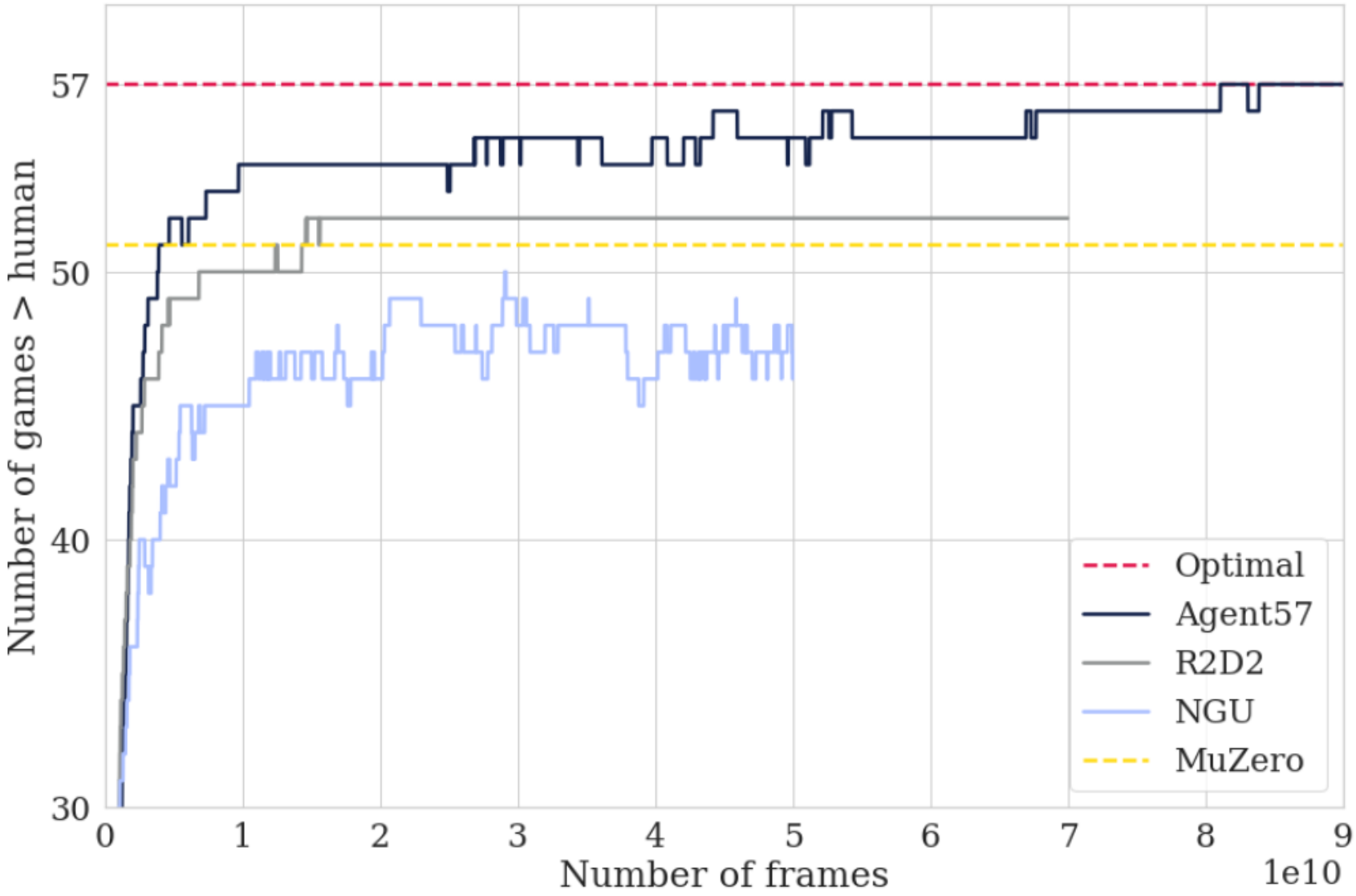
From DQN to Agent57

- Surpassed human performance in all 57 benchmarks through various improvements



modified from https://www.deepmind.com/blog/agent57-outperforming-the-human-atari-benchmark

From DQN to Agent57



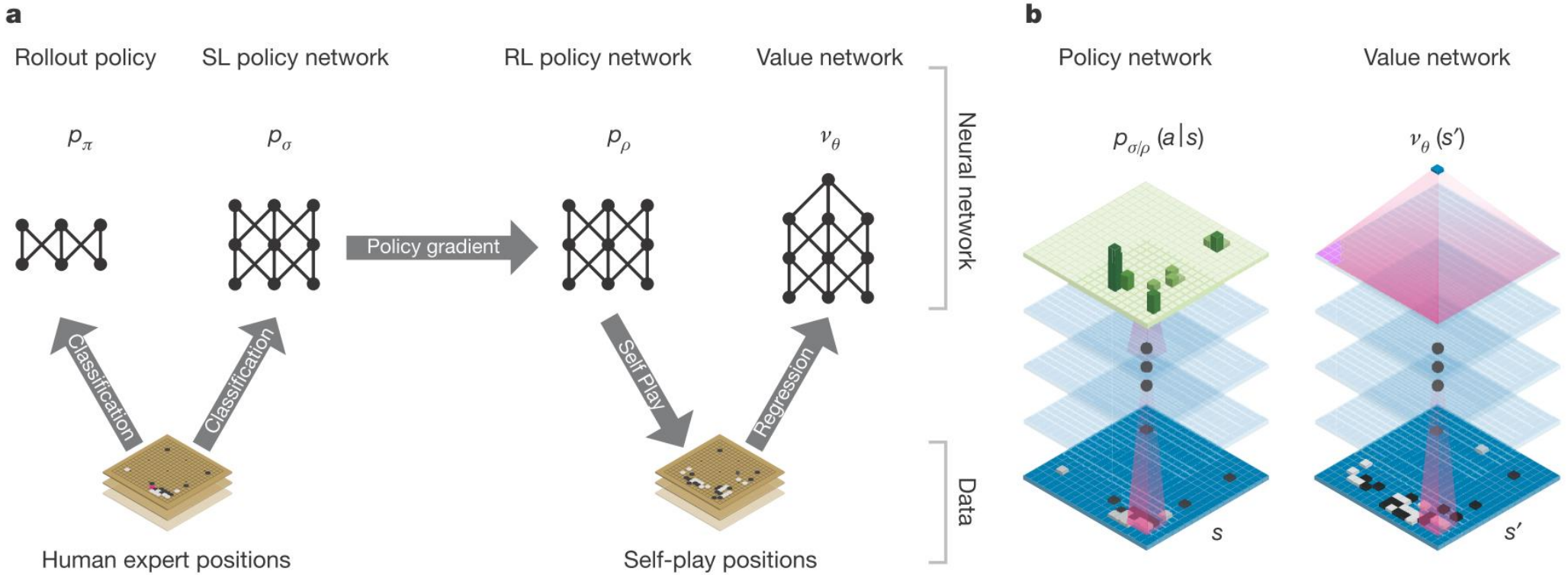
A. P. Badia et al. (2020). Agent57: Outperforming the Atari Human Benchmark. Proc. of ICML, pp. 507-517.

AlphaGo (Go)

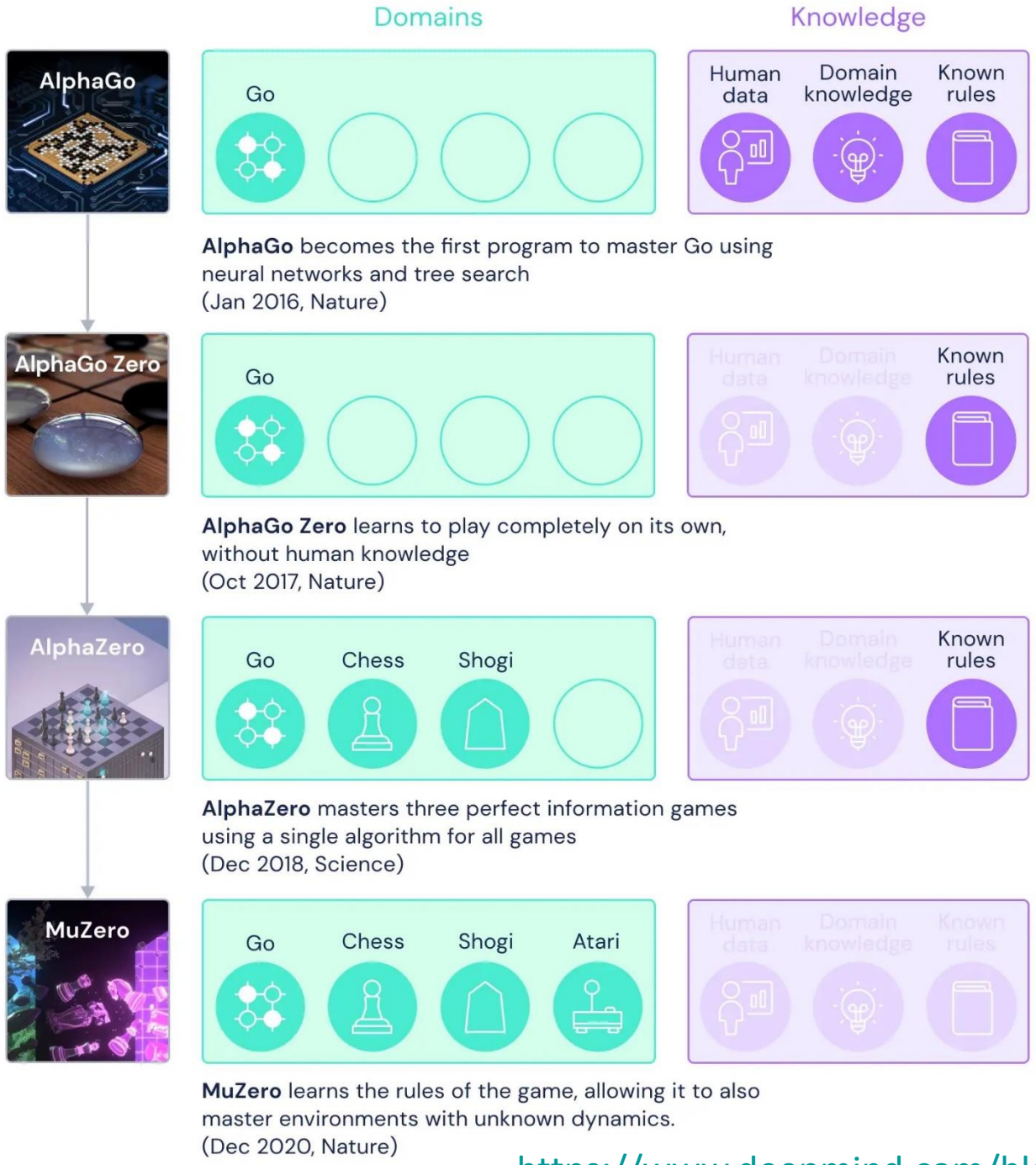
- Combination of imitation learning and reinforcement learning

Go ratings

Rank	Name	♂♀	Flag	Elo
1	Ke Jie	♂		3615
2	Google AlphaGo			3585
3	Park Jungwhan	♂		3569
4	Iyama Yuta	♂		3532
5	Lee Sedol	♂		3519



D. Silver et al. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, vol. 529, no. 7587, pp. 484–489.

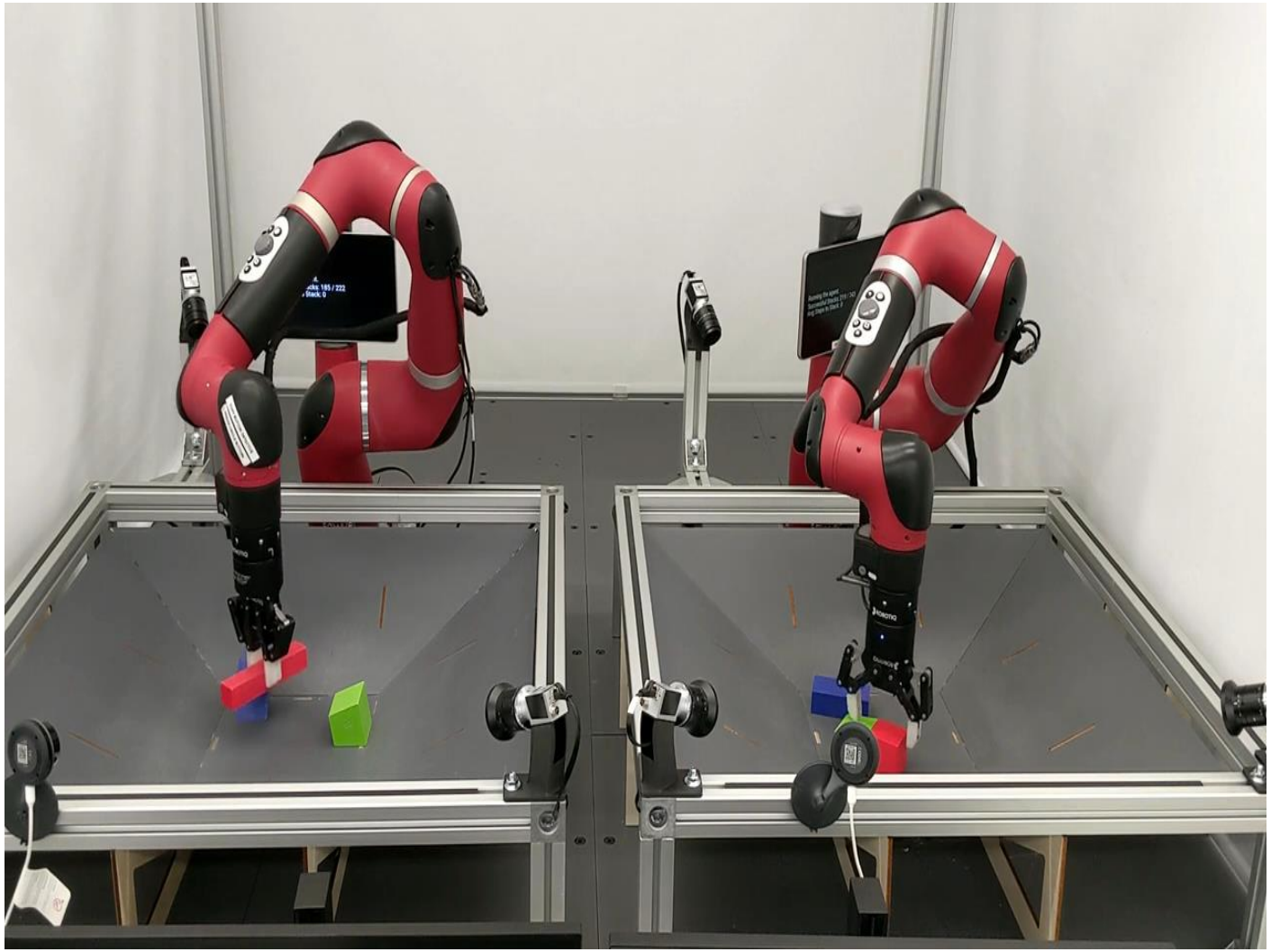


History of Alpha-series

- AlphaGo Zero does not use human-data
- AlphaZero can be applied to Chess and Shogi as well as Go
- MuZero does not know the game rules (model-free RL!)

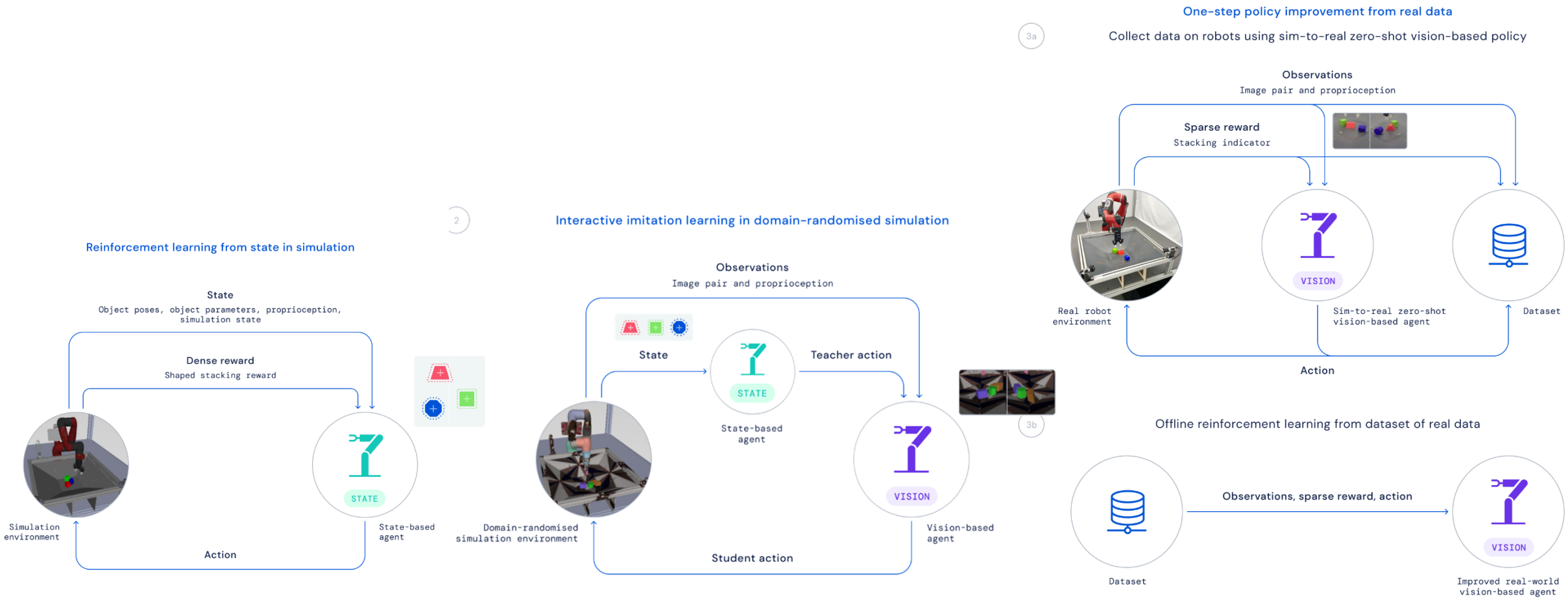
https://www.deepmind.com/blog/muzero-mastering-go-chess-shogi-and-atari-without-rules

Stacking objects



A. X. Lee et al. (2021). Beyond Pick-and-Place: Tackling Robotic Stacking of Diverse Shapes. Proc. of CoRL.

Stacking objects



A. X. Lee et al. (2021). Beyond Pick-and-Place: Tackling Robotic Stacking of Diverse Shapes. Proc. of CoRL.

Handling a non-rigid object

- Learn an optimal policy in the real environment, without using a computer simulator

Y. Tsurumine, Y. Cui, E. Uchibe, and T. Matsubara. (2019). Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation. Robotics and Autonomous Systems, vol. 112, pp. 72–83, 2019.

Robust locomotion in the wild

Experimental results on Digit

Robustness to external disturbance

G. A. Castillo, B. Weng, W. Zhang, and A. Hereid. (2022). Reinforcement Learning-Based Cascade Motion Policy Design for Robust 3D Bipedal Locomotion. IEEE Access, vol. 10, pp. 20135–20148.

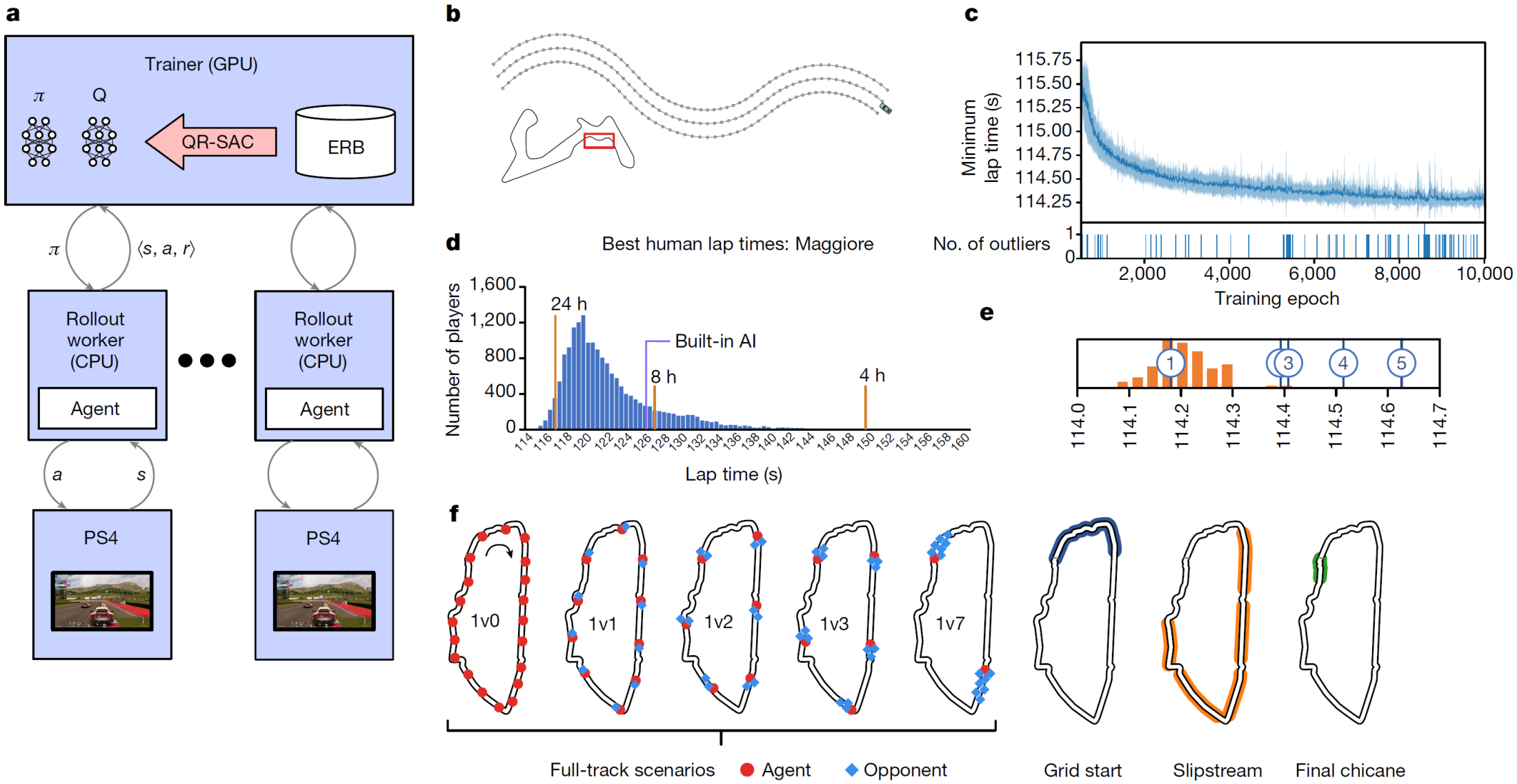
Gran Turismo Sophy

- Outperform the human-champion game players

https://www.gran-turismo.com/jp/gran-turismo-sophy

P.R. Wurman et al. (2022). Outracing champion Gran Turismo drivers with deep reinforcement learning, Nature, vol. 602, no. 7896, pp. 223–228.

Gran Turismo Sophy



P.R. Wurman et al. (2022). Outracing champion Gran Turismo drivers with deep reinforcement learning, Nature, vol. 602, no. 7896, pp. 223–228.

AlphaStar (Playing Star Craft II)

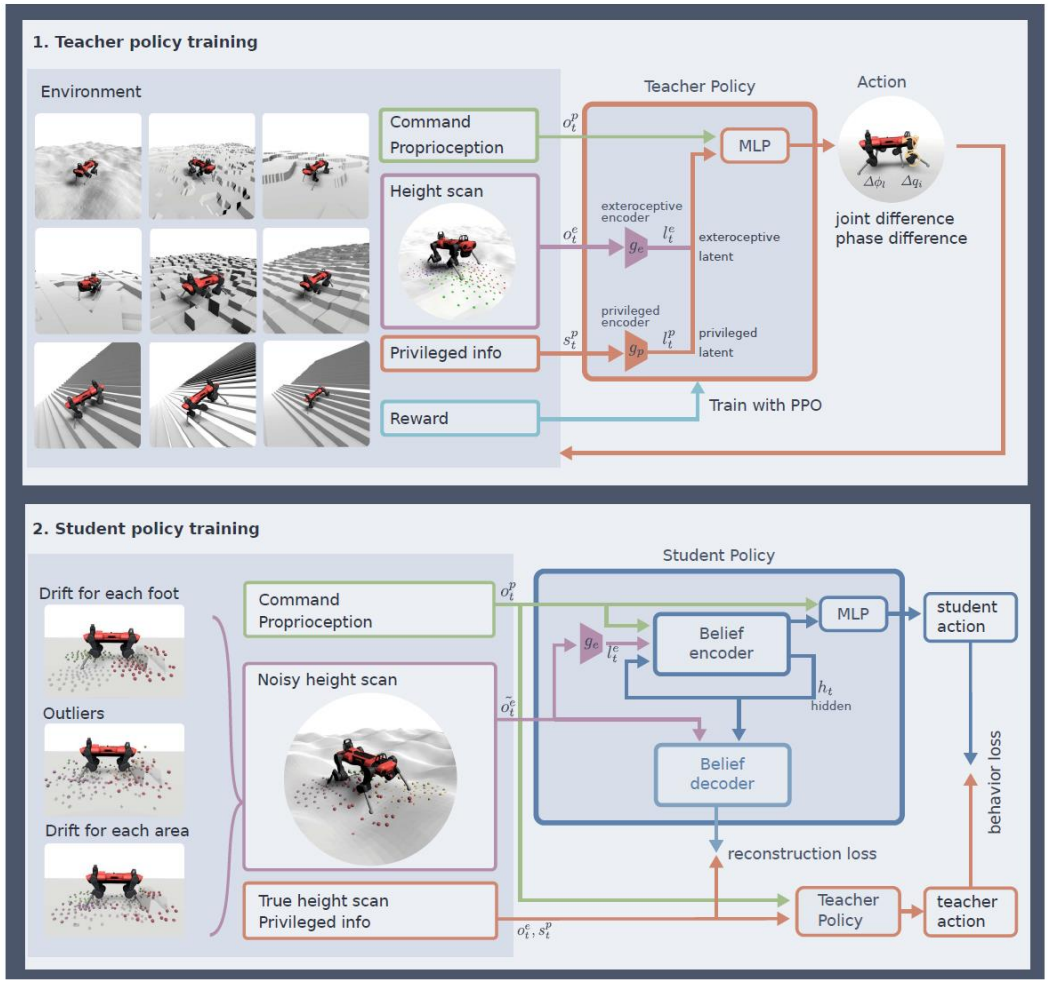
- AlphaStar (Vinyals et al., 2019)
- multiagent real-time strategy game RL + supervised learning
- 200 years

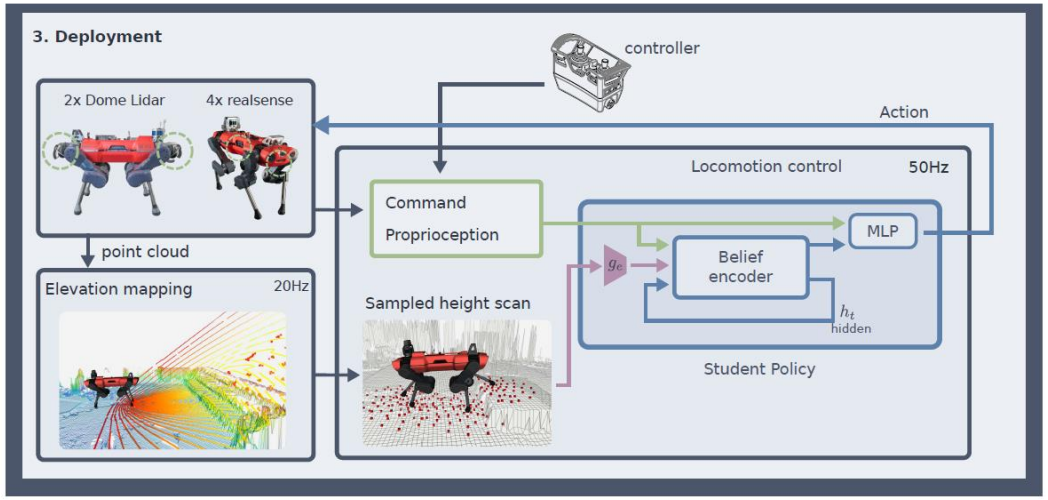
O. Vinyals, et al. (2019). Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning. Nature 575, no. 7782: 350–54.

Robust locomotion in the wild

T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. (2022). Learning robust perceptive locomotion for quadrupedal robots in the wild. *Sci. Robot.*, vol. 7, no. 62, p. eabk2822.

Robust locomotion in the wild





T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. (2022). Learning robust perceptive locomotion for quadrupedal robots in the wild. *Sci. Robot.*, vol. 7, no. 62, p. eabk2822.

Computational resources

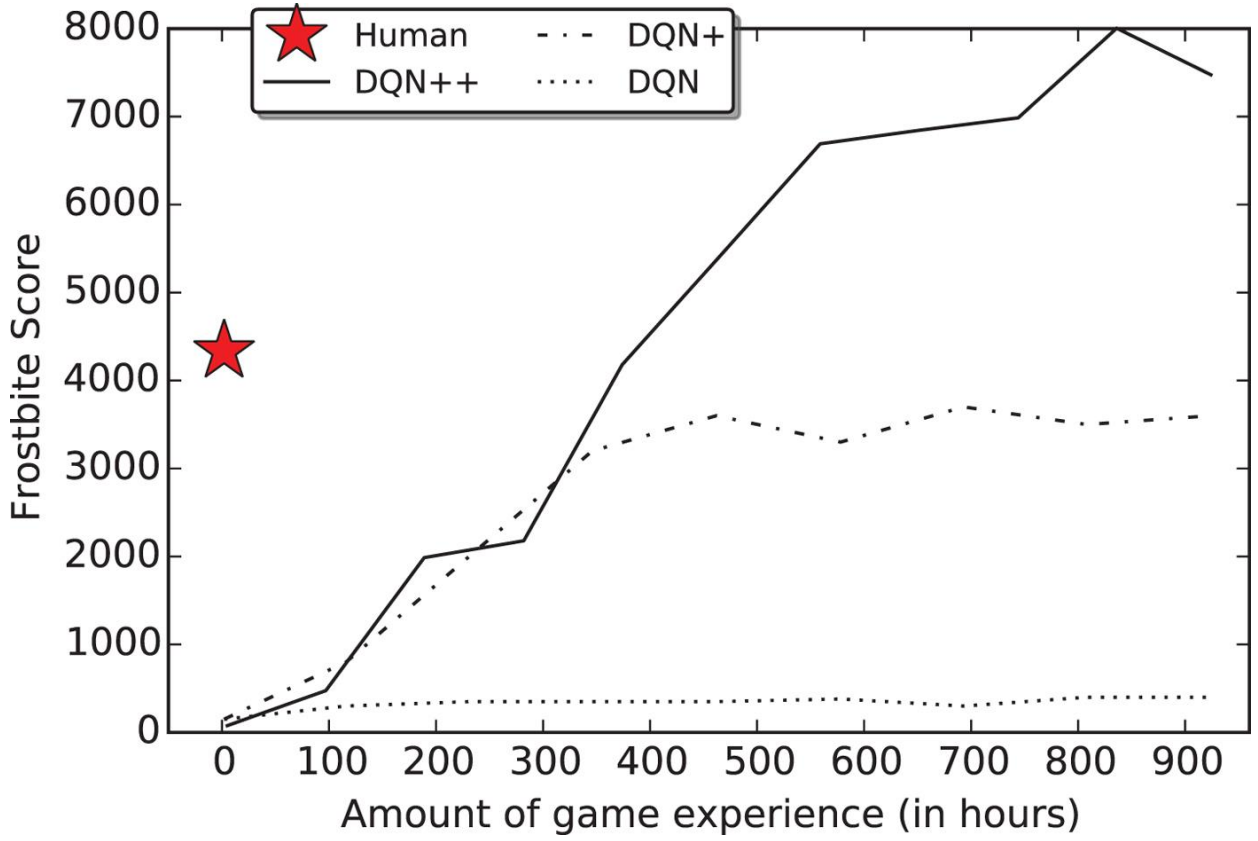
- AlphaZero independently learns to play chess, shogi, and go
  - 5,000 first-generation TPUs to generate the games
  - 64 second-generation TPUs to train the neural networks

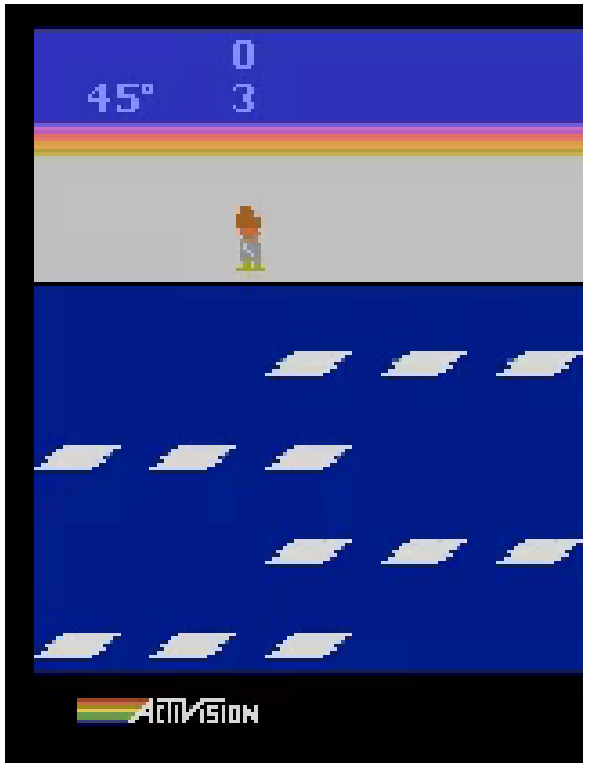
Reported in paper (1)	3-Day	40-Day
Hours of training	72	960
Matches played	4,900,000	29,000,000
Seconds per move	0.4	0.8
Training: # GPUs	64	64
Training: # CPUs	19	19
Self-play: # TPUs/player	4	4
GCP Data (2)		
TPU $/h	$6.50	$6.50
GPU $/h (best)	$0.31	$0.31
CPU $/h (best)	$0.01	$0.01
Estimates		
# moves in a Go match (3)	211	211
How many TPUs were required?		
Seconds/match/player	84.40	168.80
Matches/hour/player	42.65	21.33
Matches/total time/player	3071.09	20473.93
# Players required	1595.52	1416.44
# TPUs required	6382.10	5665.74
Costs		
Training cost (GPU)	$1,428.48	$19,046.40
Training cost (CPU)	$9.55	$127.32
Self-play: TPU cost	$2,986,822.22	$35,354,222.22
Final Estimate	$2,988,260.25	$35,373,395.94

https://www.yuzeh.com/data/agz-cost.html

Comparing learning speed for people versus DQNs

- 463 hours to reach 80% of human level

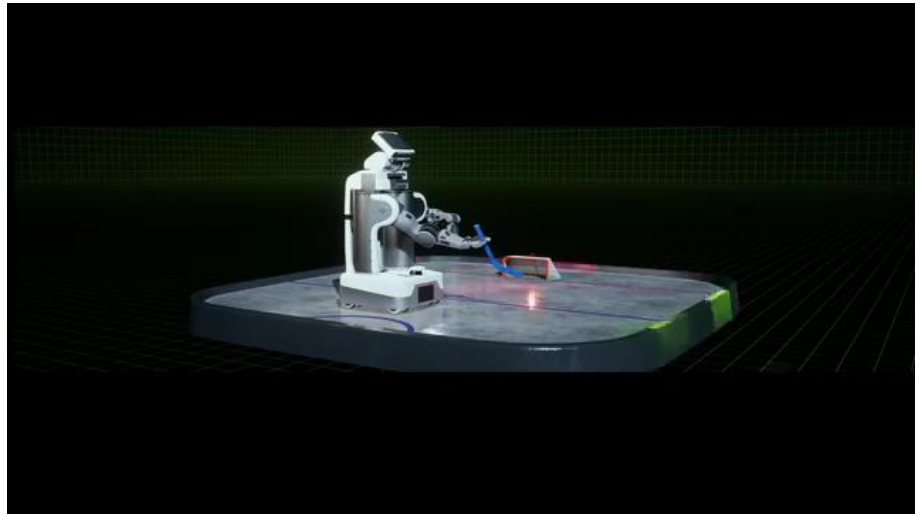




B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. (2017). Building machines that learn and think like people. Behavioral and Brain Sciences, vol. 4.

Why is deep reinforcement learning difficult?

- In Reinforcement Learning, the learning agent should collect data by itself
  - An initial policy, which is randomly initialized, cannot explore the environment efficiently, and therefore, useful data is not gathered
- Using simulators
  - Pay attention to simulation-to-reality gap



https://www.youtube.com/watch?v=oa__wkSmWUw

Why is deep reinforcement learning difficult?

- In Reinforcement Learning, the learning agent should collect data by itself
  - An initial policy, which is randomly initialized, cannot explore the environment efficiently, and therefore, useful data is not gathered
- Using multiple real robots in parallel
  - Expensive
  - The amount of data is still limited

https://www.youtube.com/watch?v=cXaic_k80uM

Summary

- Introduction to deep reinforcement learning
- Recent studies achieve remarkable success in games, but it is still challenging to apply to real problems
- Promising approaches
  - Imitation learning
  - Offline reinforcement learning

Report

- Please select one topic and write your report

1. Consider the application of reinforcement learning

2. How can we reduce the training data and time?