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

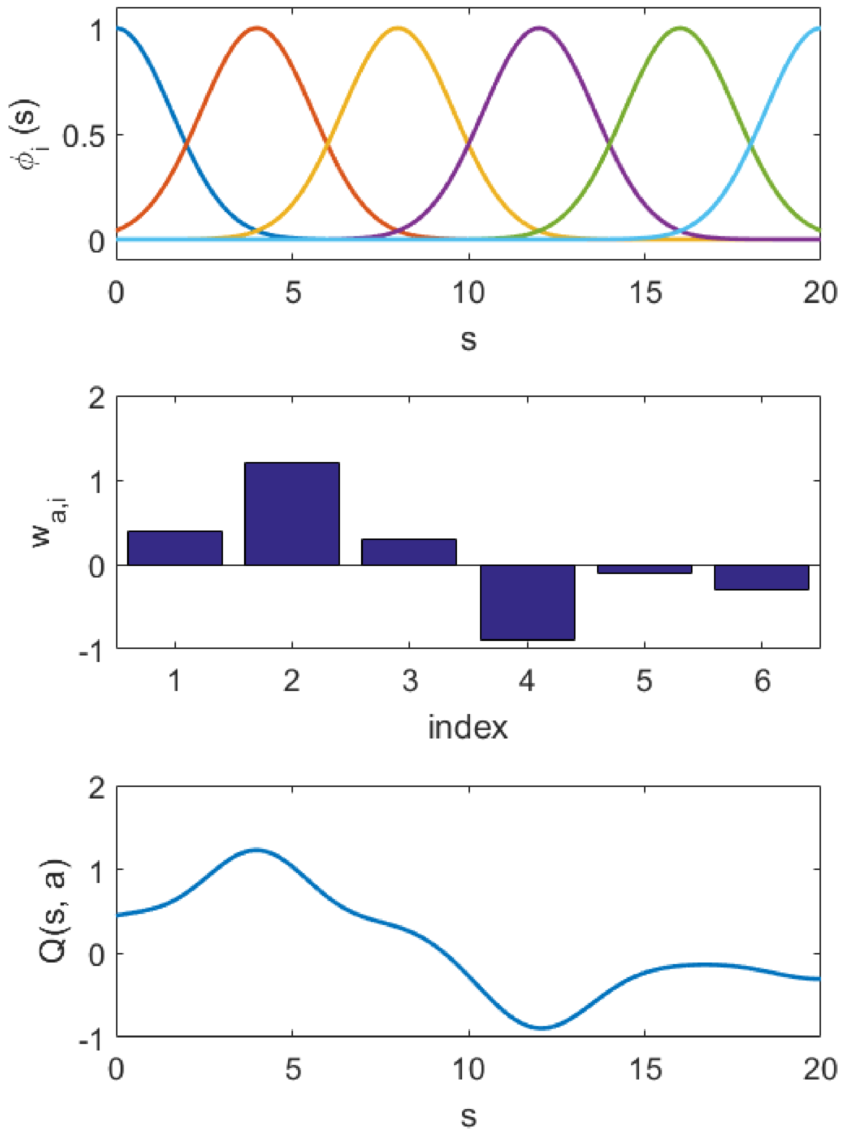
- So far we have assumed that states were discrete and it was possible to represent value functions and policy by a lookup table
  - Every state s has an entry $V(s)$
  - Every state-action pair s, a has an entry $Q(s, a)$ and/or $\pi(a \mid s)$
- It is NOT true for realistic tasks
  - Go: 10^{170} states
  - Helicopter/Mountain Car: continuous state space
  - Robots: informal state space
- So we need to approximate value functions and policy

Linear Function Approximation

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

$$\hat{q}(s, a, \boldsymbol{w}) = \boldsymbol{w}_a^\top \boldsymbol{\phi}(s)$$

  - $\boldsymbol{w}_a$: weight parameter vector for action a
  - $\boldsymbol{w} = \{\boldsymbol{w}_a\}$
  - $\boldsymbol{\phi}(s)$: basis function vector
- Under some assumptions, convergence proof is given



Function approximation

- It is usually infeasible to discretize a state space because the number of states grows exponentially
- 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 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?

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 result in divergence because activation functions such as a sigmoid function is not localized

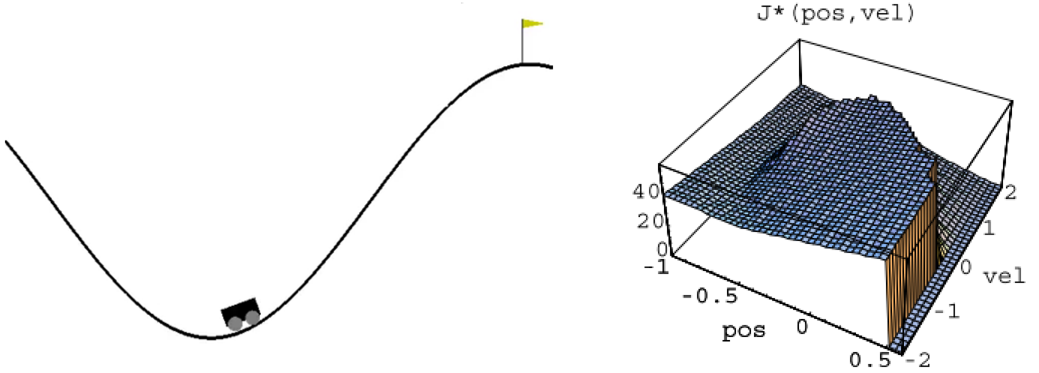


Figure 5: The car-on-the-hill domain. When the velocity is below a threshold, the car must reverse up the left hill to gain enough speed to reach the goal, so J^* is discontinuous.

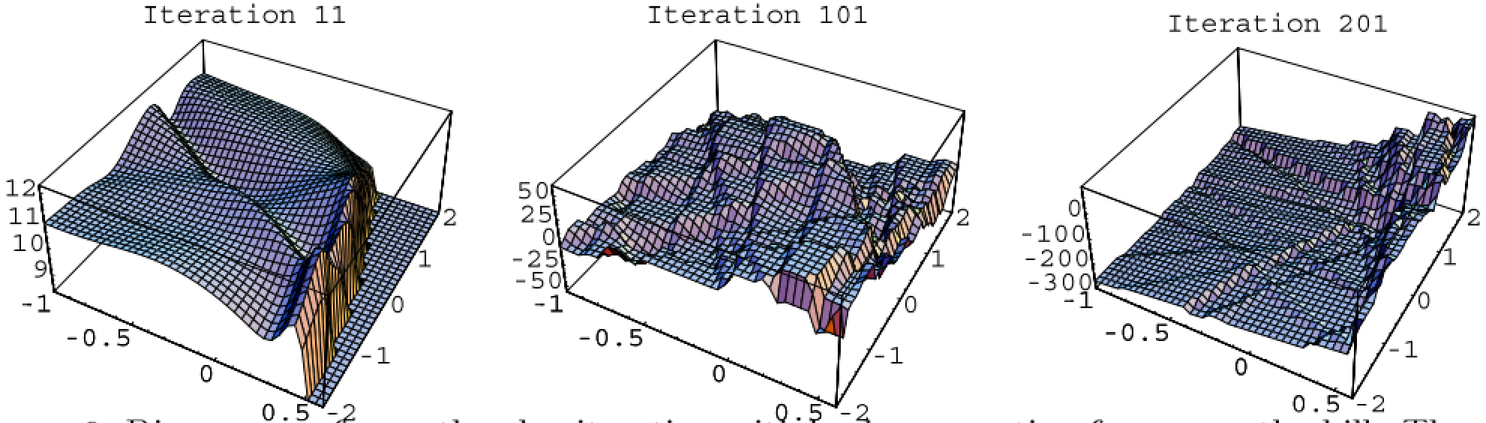
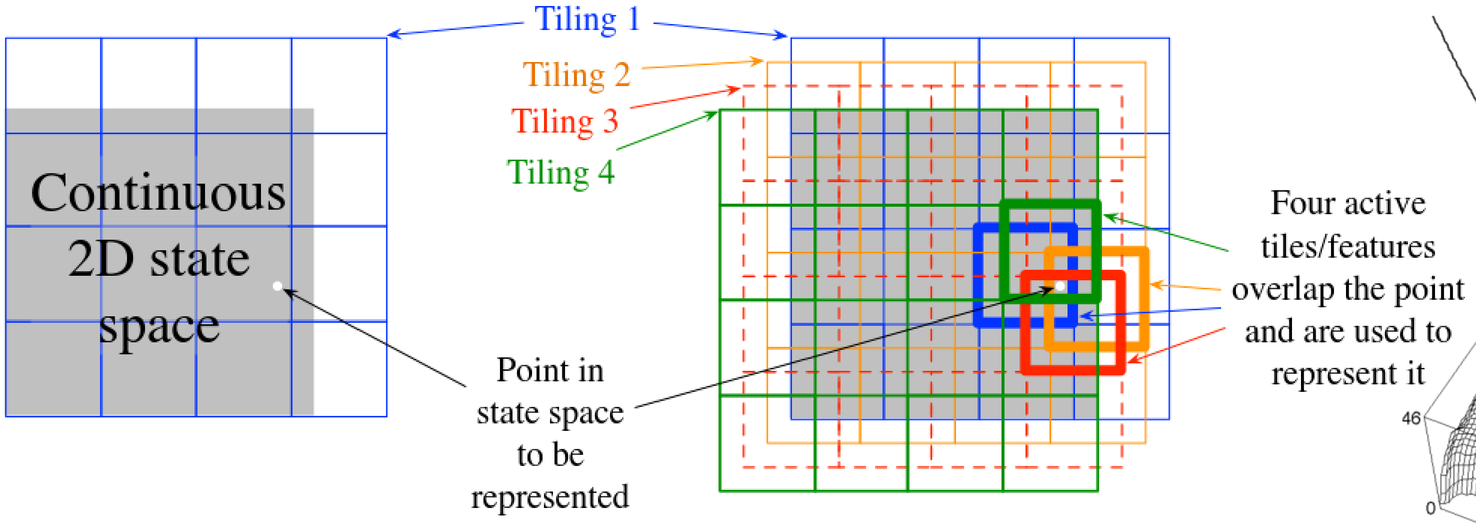


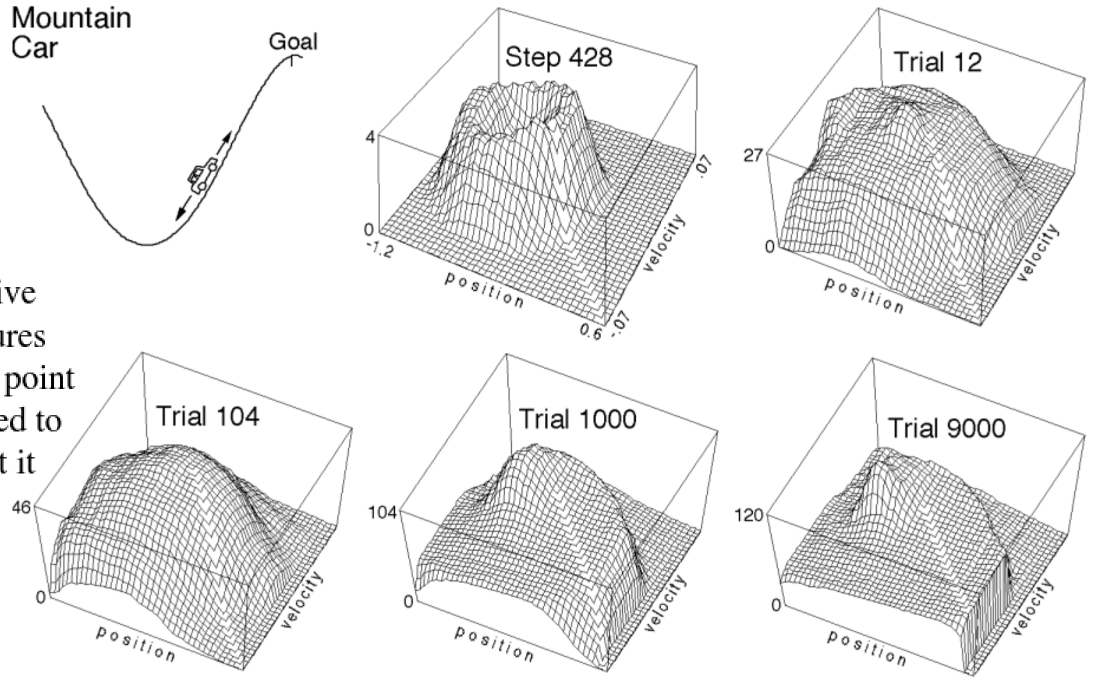
Figure 6: Divergence of smooth value iteration with backpropagation for car-on-the-hill. The neural net, a 2-layer MLP with 80 hidden units, was trained for 2000 epochs per iteration.

[Boyan and Moore, 1995]

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

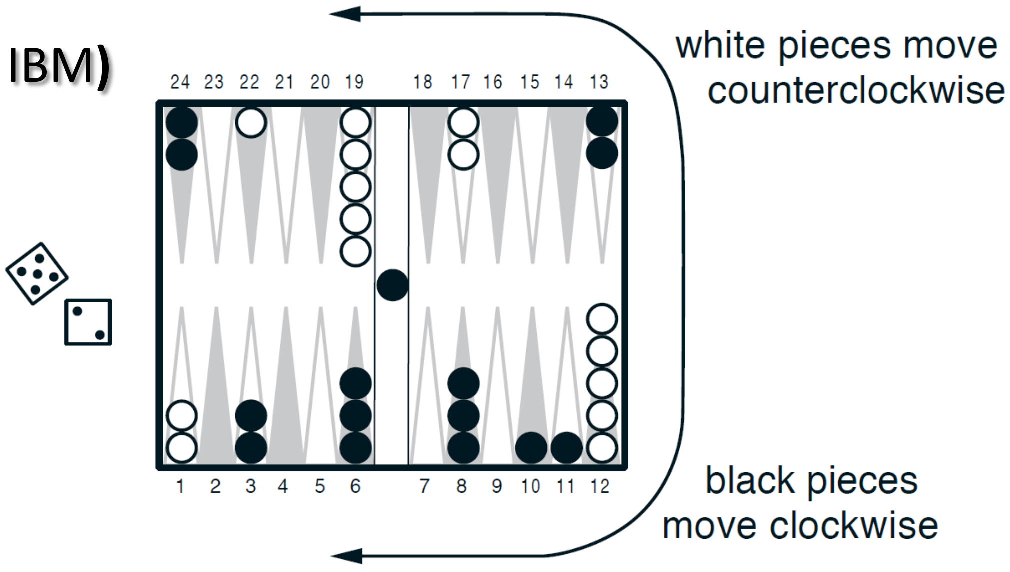
[Sutton, 1996]

What is deep reinforcement learning?

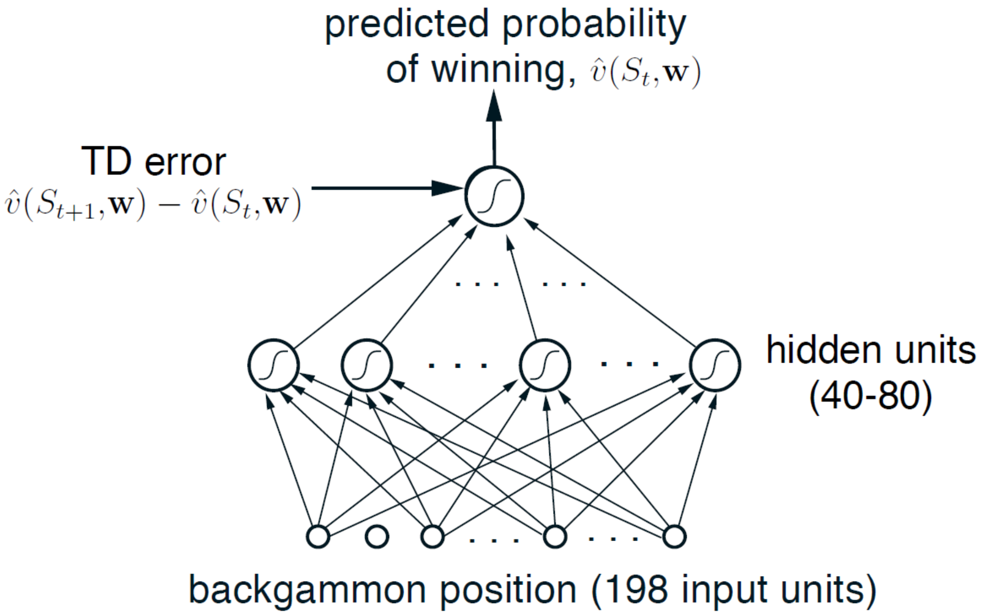
- Reinforcement learning + deep learning
  - RL algorithm with deep neural network function approximation
- End-to-end learning
  - Learning from raw state input, (e.g., image pixels)
- Same architecture across different tasks
  - All Atari 2600 games
  - Different task for a robot using the same sensor input (e.g., camera images)

1991-95: TD-gammon (Gerry Tesauro, IBM)

- RL backgammon program
- The big early success of RL + NN
- TD(λ) with NN
  - 2- or 3-ply search
- 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



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

1995-2013: Almost nothing

- Rich Sutton’s Q&A (2001-04):

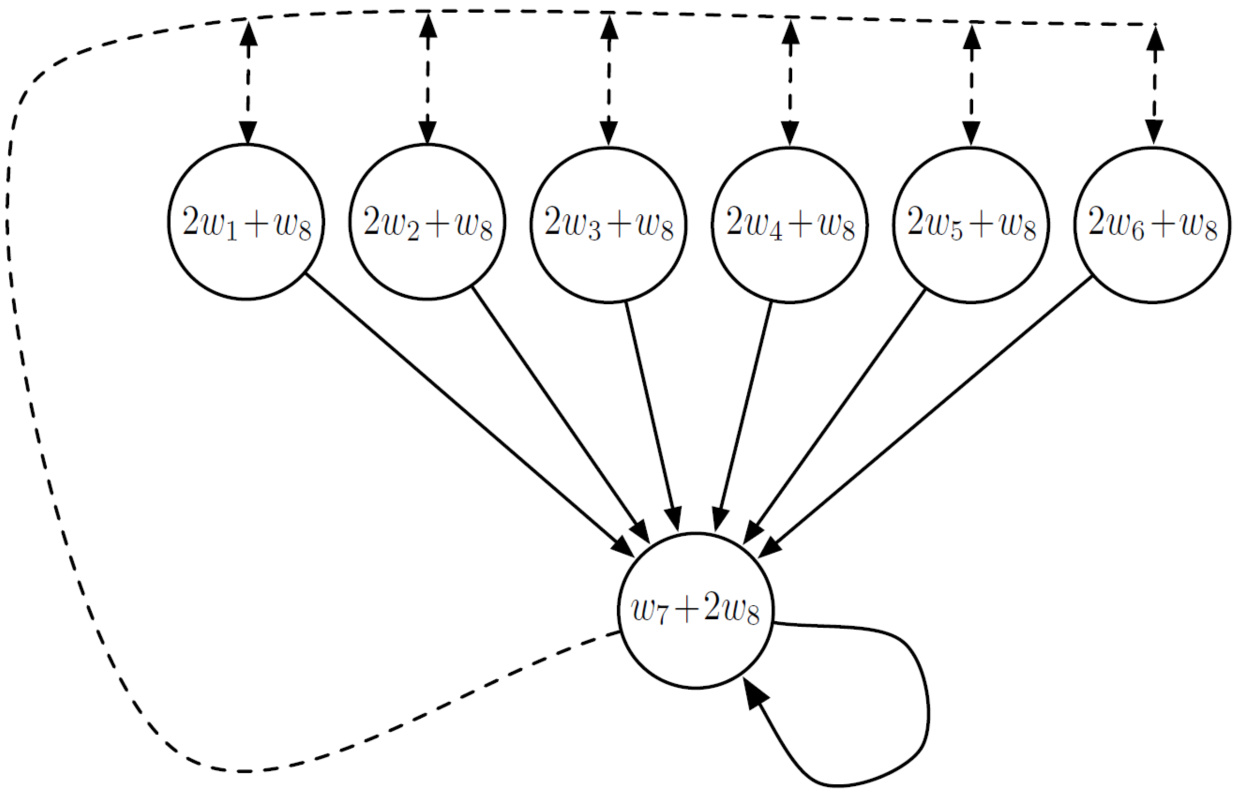
I am doing RL with a backpropagation neural network and it doesn't work; what should I do?

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 special case:
  - Games always ends after 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

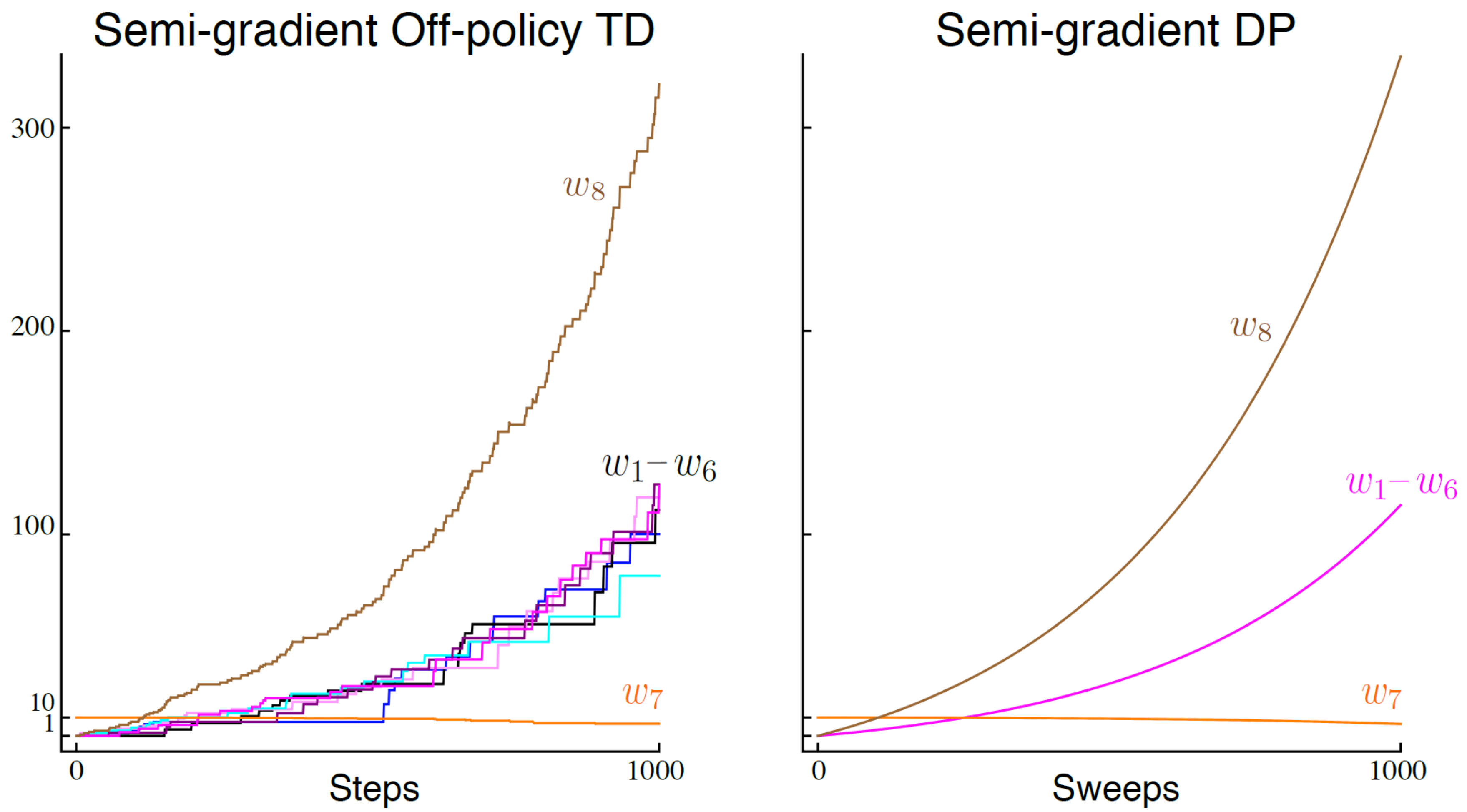


Figure 11.2: Demonstration of instability on Baird's counterexample. Shown are the evolution of the components of the parameter vector $\mathbf{w}$ of the two semi-gradient algorithms. The step size was $\alpha = 0.01$, and the initial weights were $\mathbf{w} = (1, 1, 1, 1, 1, 1, 10, 1)^\top$.

2013-15: Deep Q networks (DQN)(Google DeepMind)

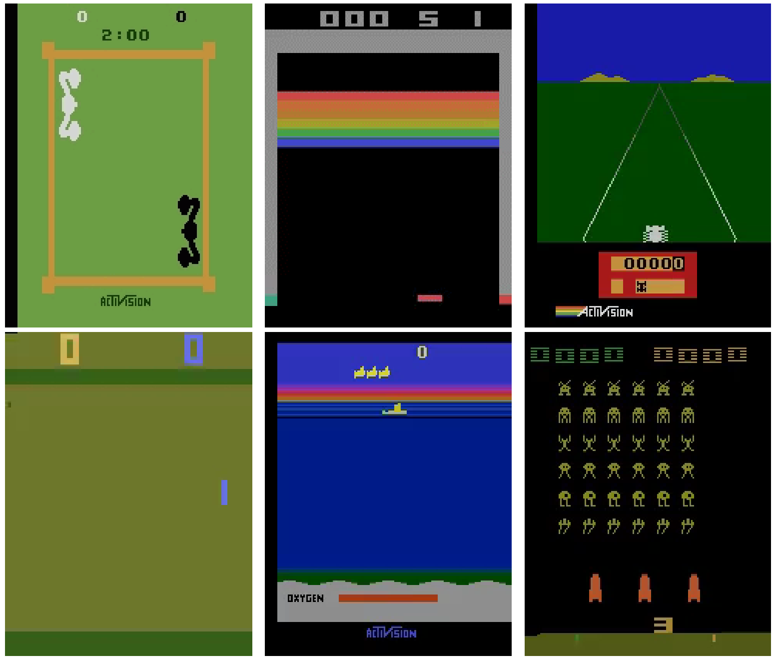
- DQN: Q-learning + CNN + target network + experience replay using GPUs
- Target network
  - Different and fixed network (θ^-) for computing the target Q
  - $\delta \leftarrow r + \gamma \max_{a'} Q(s', a' | \theta^-) - Q(s, a | \theta)$
  - In practice: after every fixed number of learning updates
    - $\theta^- = \theta$
- Experience replay
  - Store experiences: (s, a, r, s')
  - Update in mini-batches by uniform sampling
  - Reuse of samples: More effective use of experiences
  - Enables the use of GPUs and modern optimization techniques from classification

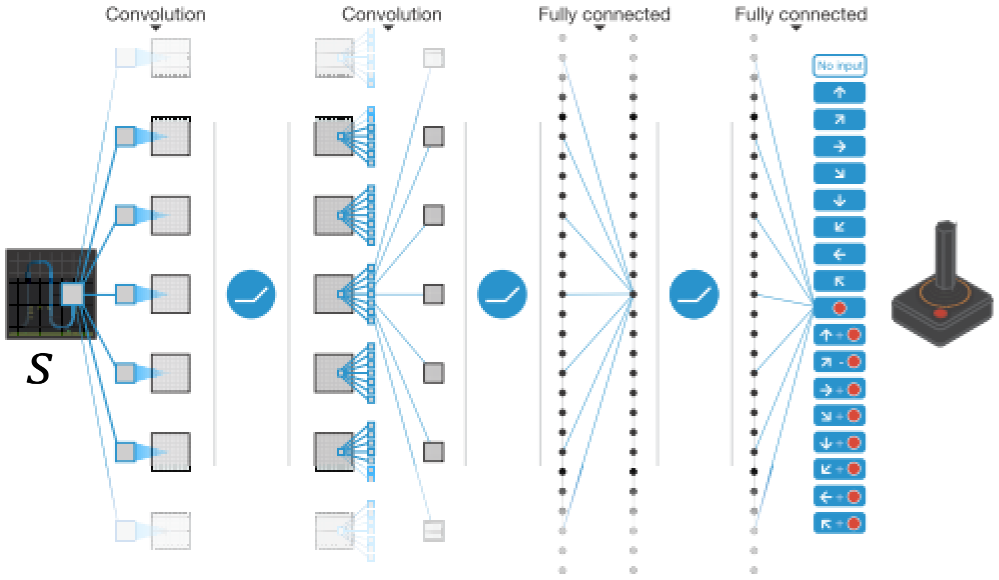
DQN playing ATARI 2600

- Home video console released in 1977
- Arcade Learning Environment (ALE)
- 210 x 160 color video at 60 Hz
- Pac-Man (‘82) sold 7.7 million copies

Deep Q Network (DQN)

- DQN represents q_π by a deep convolutional neural network
  - The input consists of 4 consecutive images of size 84x84
- DQN performed at a level that is comparable to professional human game testers



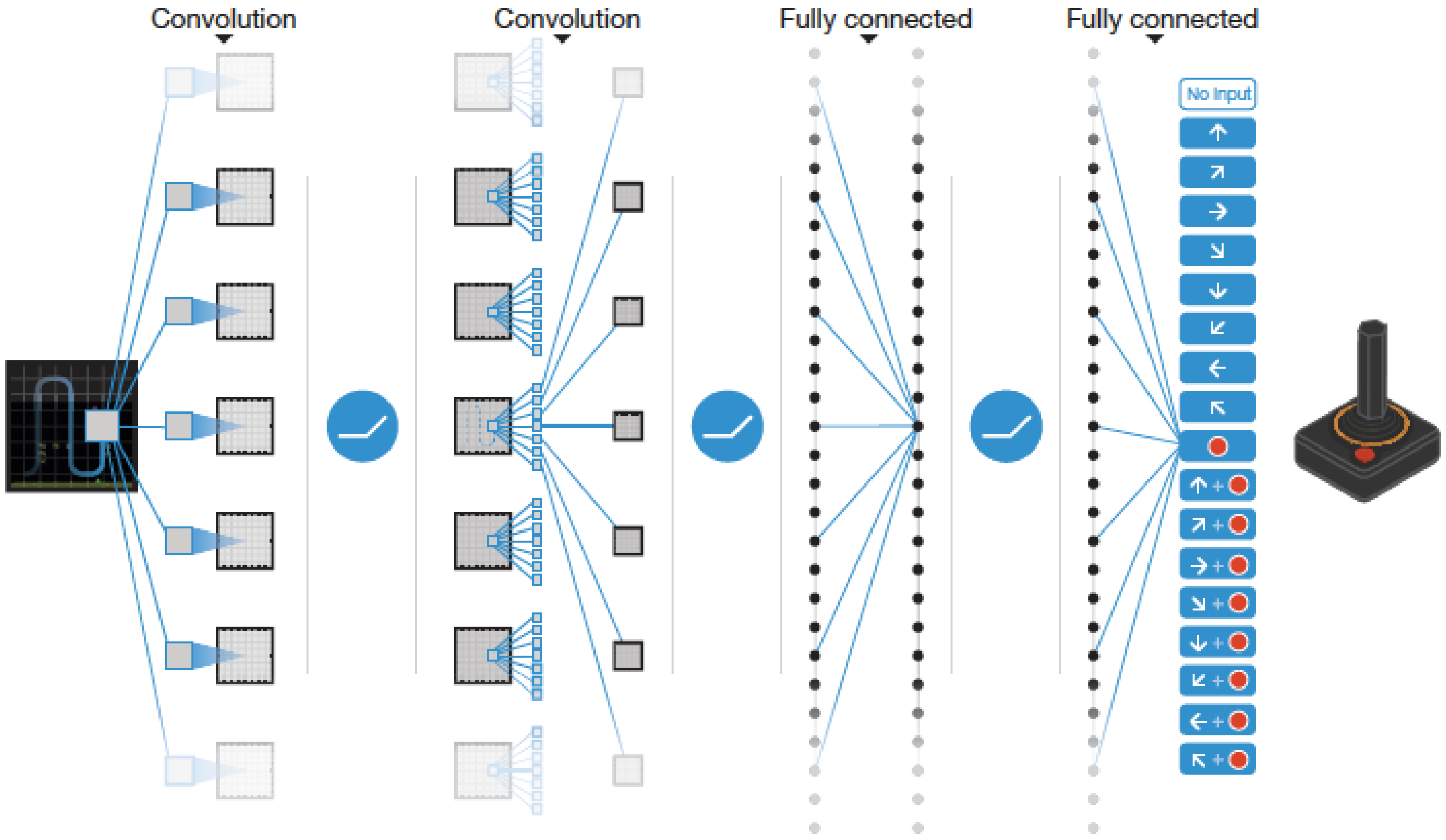


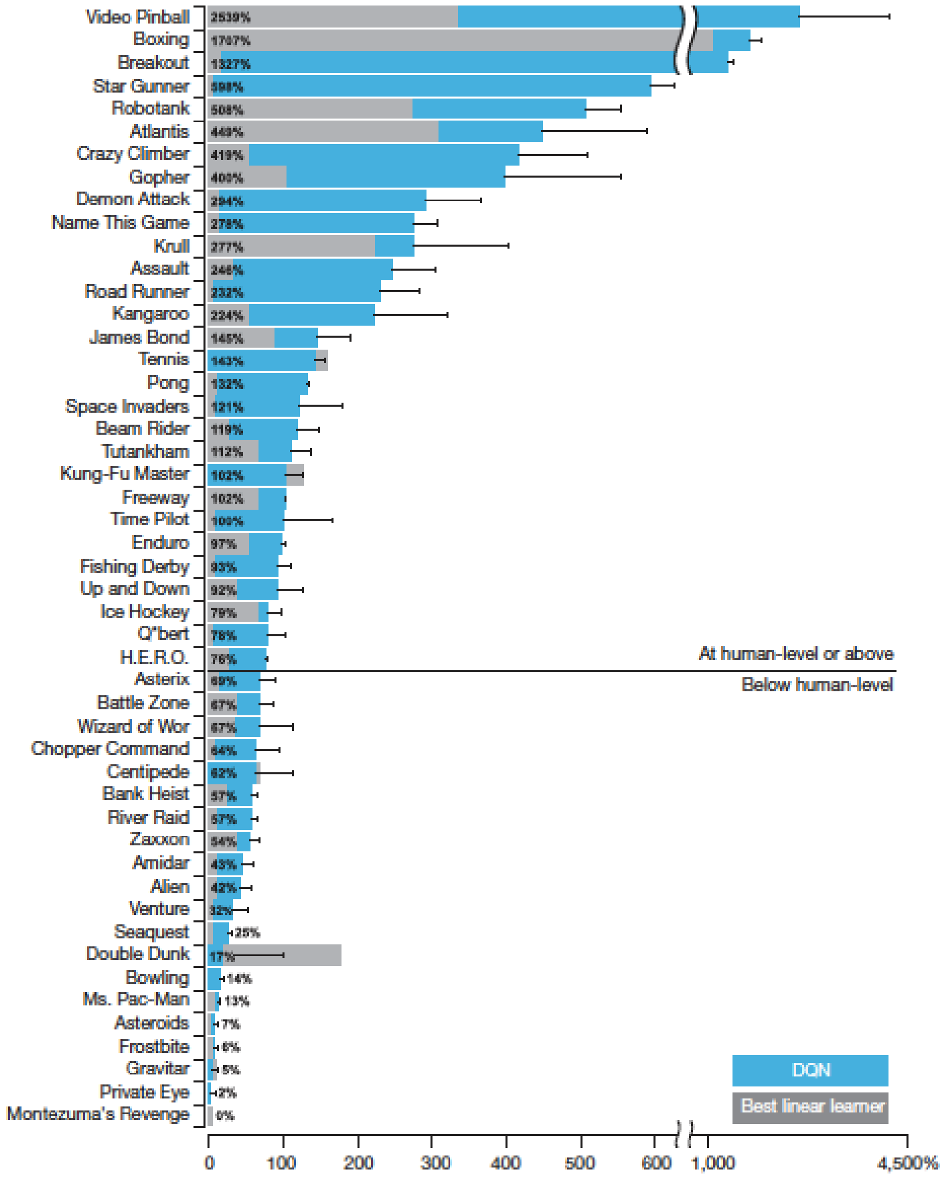
$q_\pi(s, a)$

[Mnih et al., 2013; 2015]

DQN playing ATARI 2600

Human level: 75% of human level



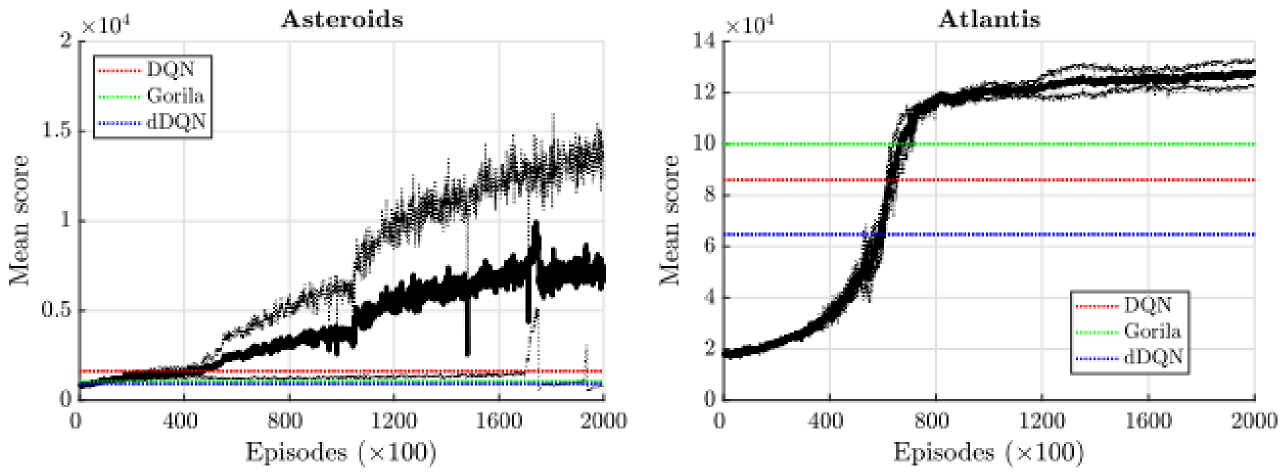


Mnih et al. Nature. 2015

What is the key issues in DQN?

- Neural fitted Q iteration: Convert reinforcement learning problems to supervised learning ones
- Experience replay: Store experiences and reuse them later
  - We should use off-policy reinforcement learning algorithms
- (Clipping the values of rewards)

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

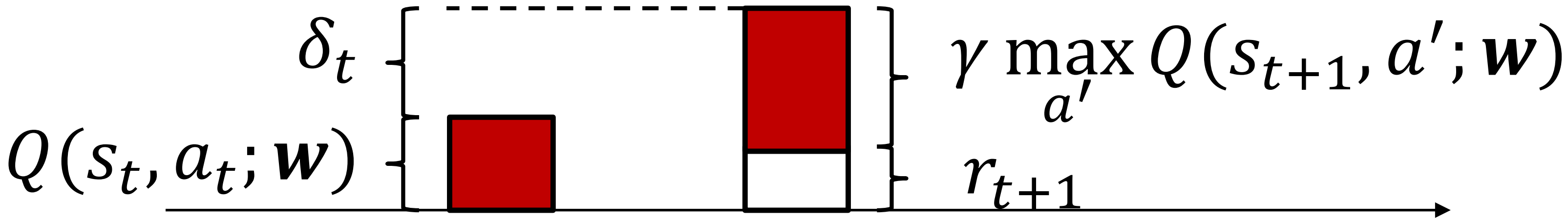


[Elfwing et al., 2018]

Why are nonlinear approximators unstable

- The target value changes when the parameters are updated!

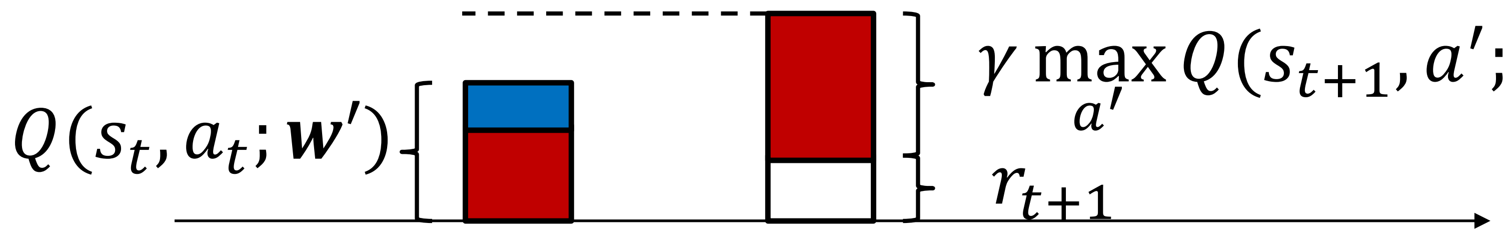
$$\delta_t = r_{t+1} + \gamma \max_{a'} \tilde{Q}(s_{t+1}, a'; \boldsymbol{w}) - \tilde{Q}(s_t, a_t; \boldsymbol{w})$$



Why are nonlinear approximators unstable

- The target value changes when the parameters are updated!

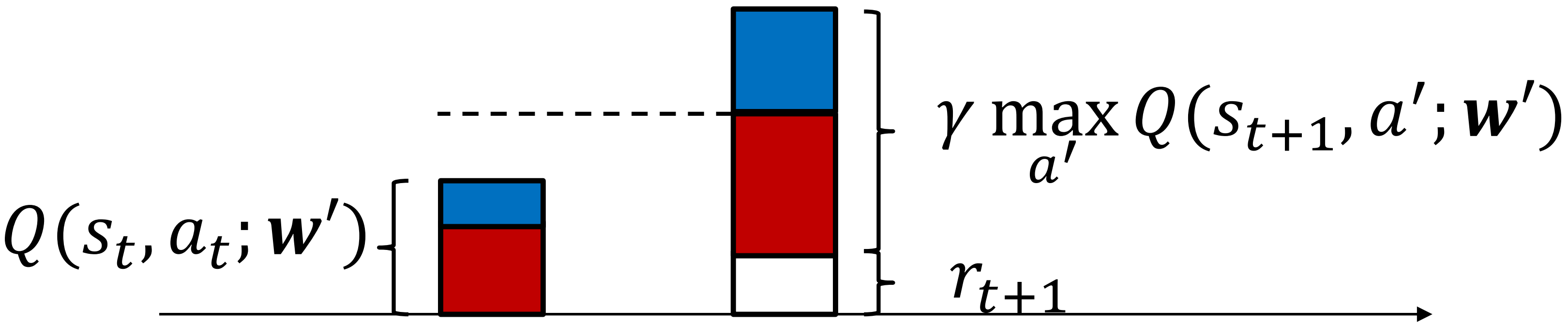
$$\delta_t = r_{t+1} + \gamma \max_{a'} \tilde{Q}(s_{t+1}, a'; \boldsymbol{w}) - \tilde{Q}(s_t, a_t; \boldsymbol{w})$$



Why are nonlinear approximators unstable

- The target value changes when the parameters are updated!

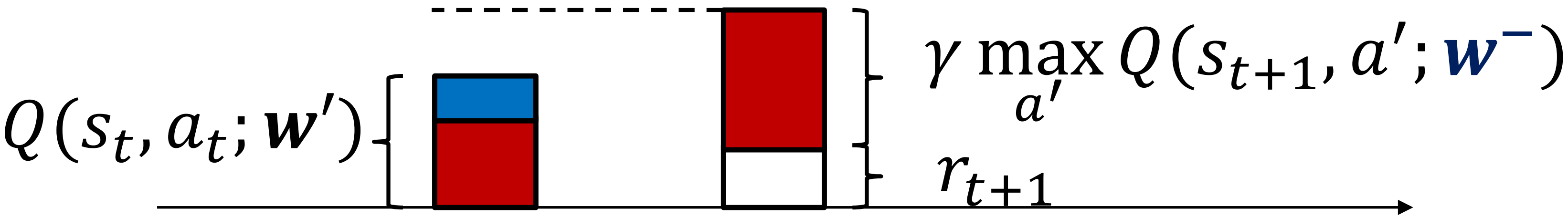
$$\delta_t = r_{t+1} + \gamma \max_{a'} \tilde{Q}(s_{t+1}, a'; \boldsymbol{w}) - \tilde{Q}(s_t, a_t; \boldsymbol{w})$$



Neural fitted Q-iteration

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

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



- Frequency to update the target network is critical
  $\Rightarrow$ Fast, but unstable learning for frequent update
  $\Rightarrow$ Stable, but slow learning when we rarely update $\boldsymbol{w}^{-} \leftarrow \boldsymbol{w}$

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 $\theta^{-} = \theta$

DQN: Importance of target network and replay

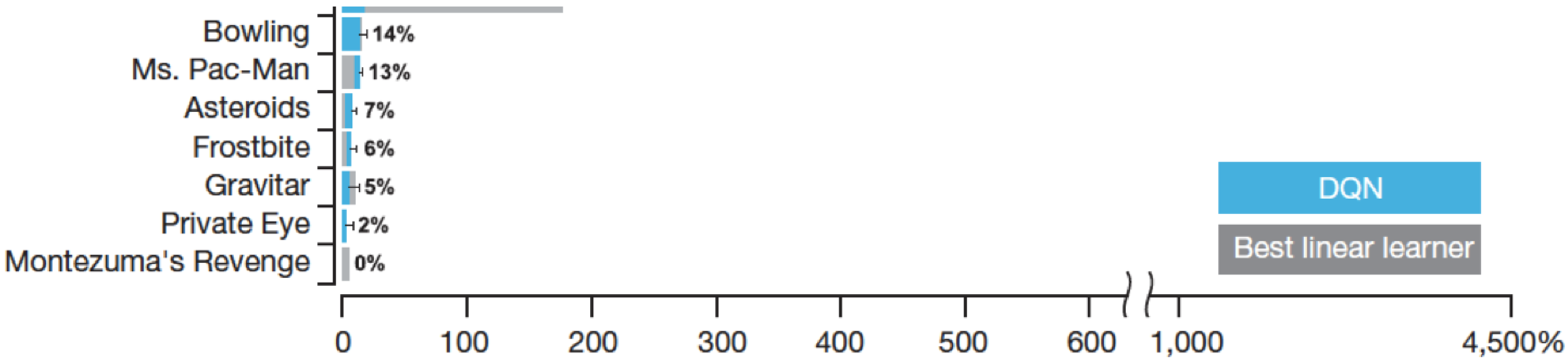
Extended Data Table 3 | The effects of replay and separating the target Q-network

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).

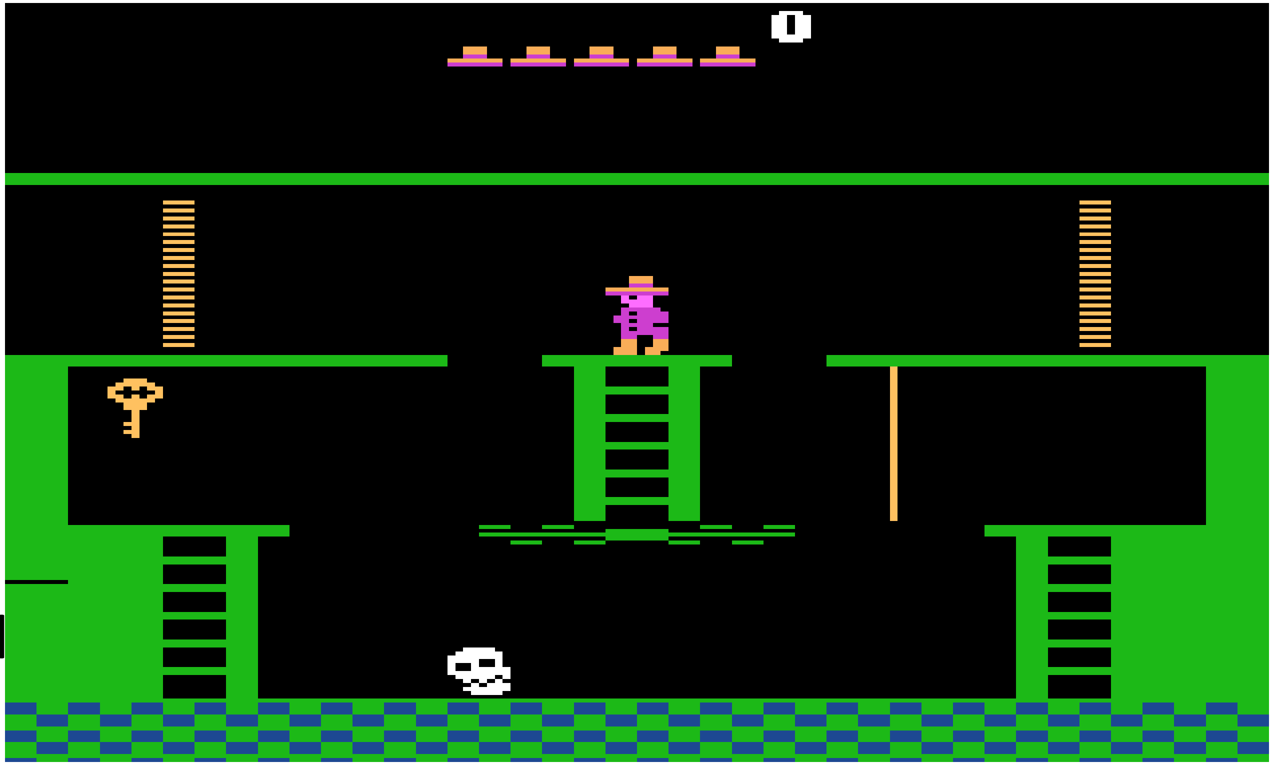
DQN in Atari: bad games

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



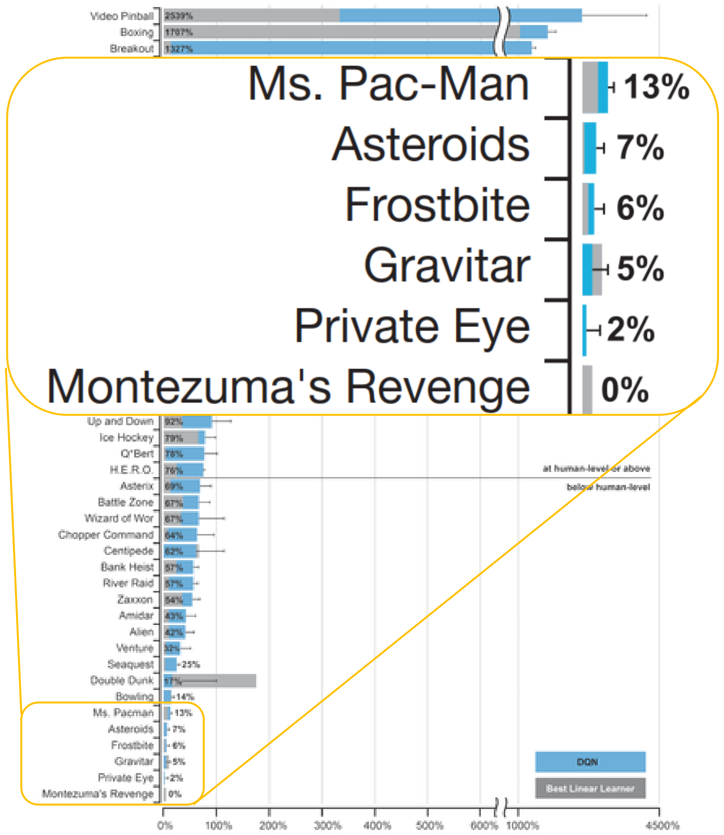
Montezuma’s revenge

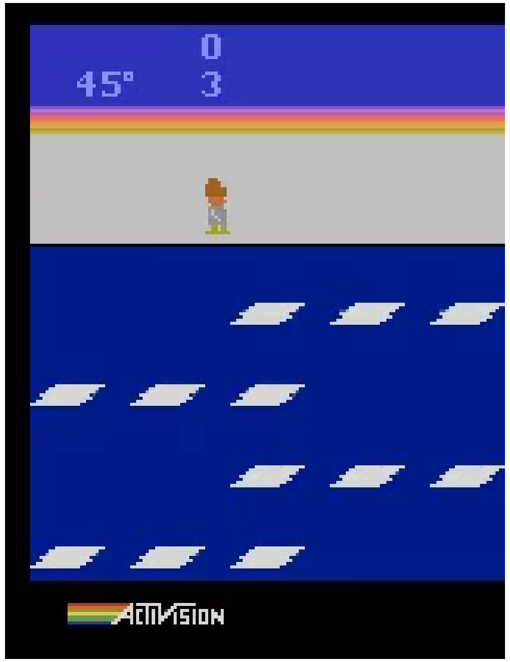
- Random exploration
  - ~0 reward
- Very long delayed rewards
- Have to do things in exact order
  - Eg, get key to exit screen
- New screen
  - complete new environment
- Requires memory
  - Part of screen played in darkness until

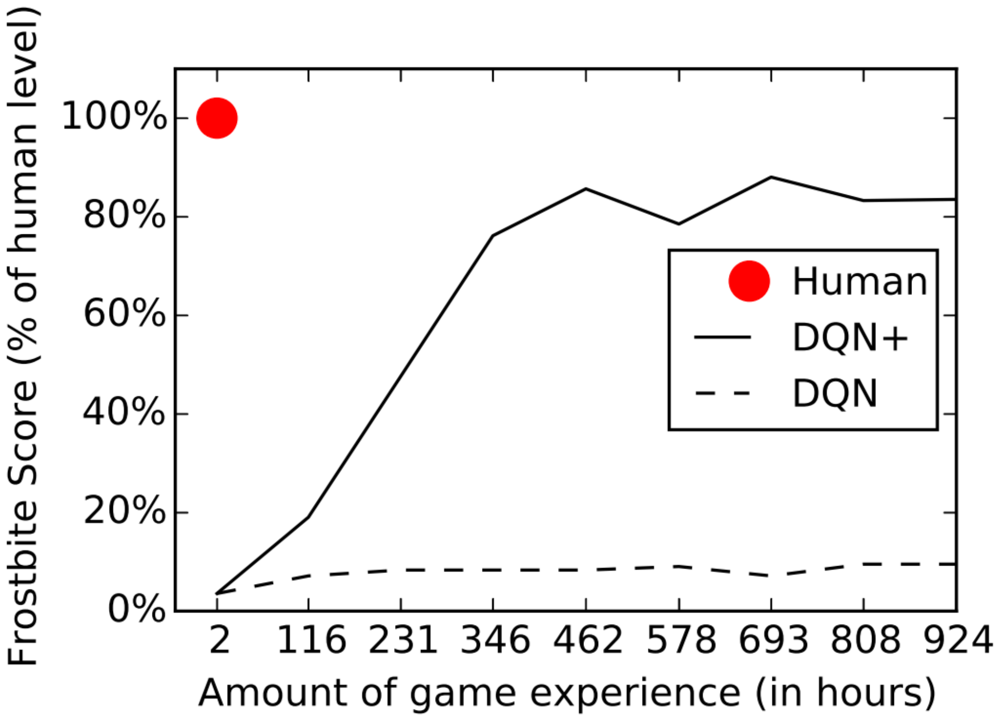


(Deep) Reinforcement Learning is Sample-Inefficient

- 463 hours to reach 80% of human level



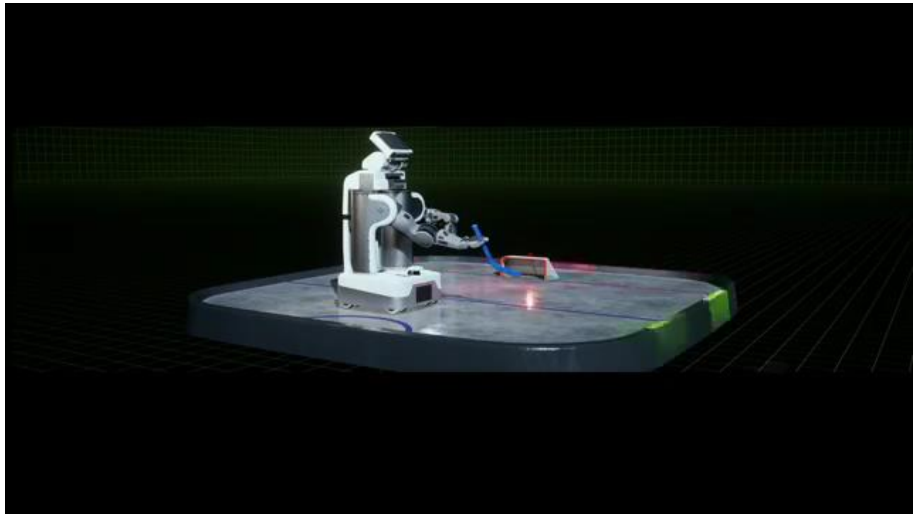




[Lake et al., 2017]

Why is deep reinforcement learning difficult?

- In Deep Learning, we can prepare a huge amount of training data beforehand
- 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
- Using multiple real robots in parallel
  - Expensive
  - The amount of data is still limited



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

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

Extension of Deep Q Networks

Rainbow: Integration of extended DQNs

- Double DQN (AAAI 2016)
- Dueling network (ICML 2016)
- Prioritized experience replay (ICLR 2016)
- Noisy network (ICLR 2018)
- Multi-step prediction
- Distributional reinforcement learning (ICML 2017)

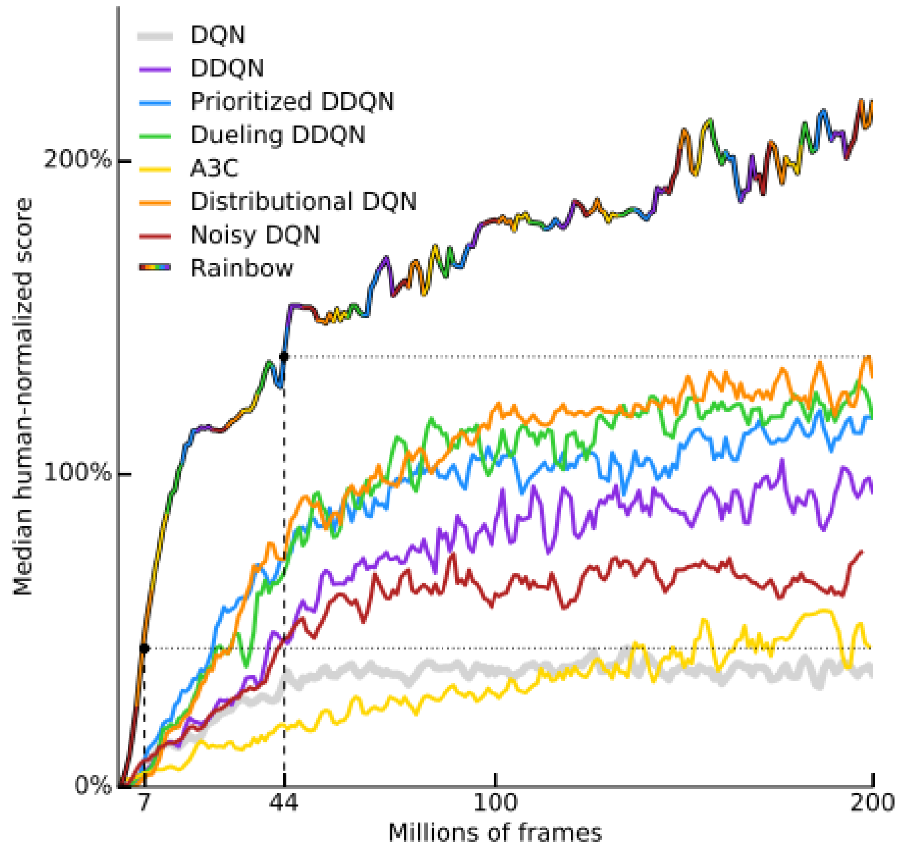
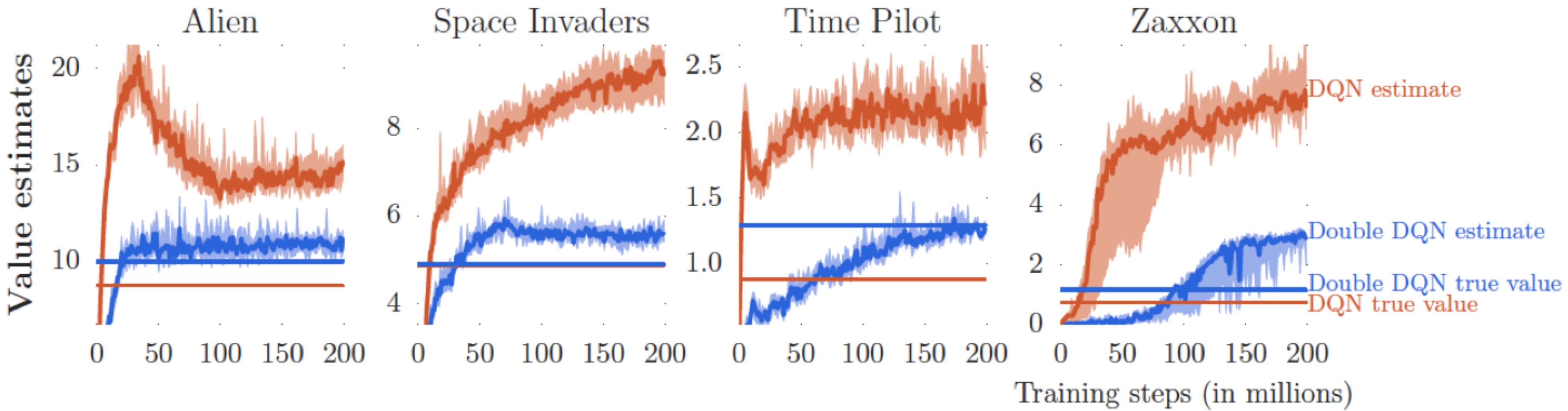


Figure 1: **Median human-normalized performance** across 57 Atari games. We compare our integrated agent (rainbow-

[Hessel et al., 2018]

Double DQN (Hasselt et al. '15)

- Q-learning can overestimate the Q-values
  - max-operator in TD-error
- Solution: separate networks action (θ) and value estimation (θ^-)
- $v^{DQN} = r + \gamma \max_{a'} Q(s', a'|\theta) = r + \gamma Q\left(s', \operatorname{argmax}_{a'} Q(s', a'|\theta)|\theta\right)$
- $v^{doubleDQN} = r + \gamma Q\left(s', \operatorname{argmax}_{a'} Q(s', a'|\theta)|\theta^-\right)$

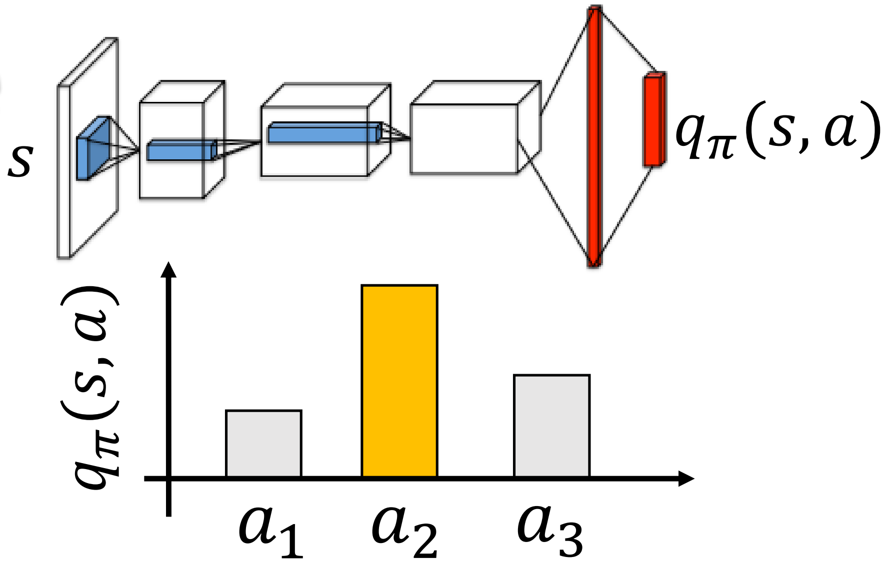


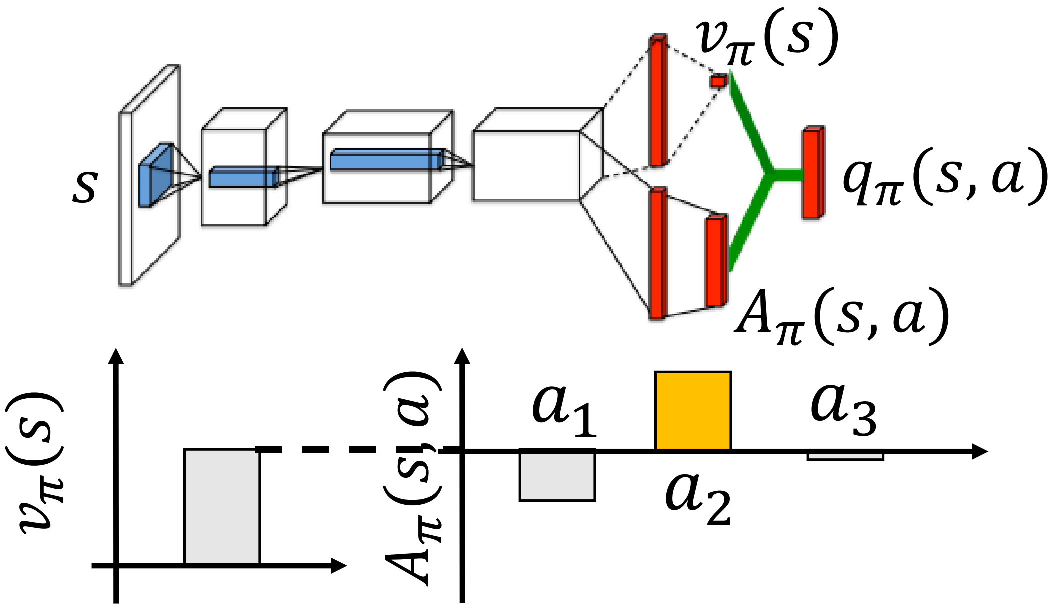
Dueling network architecture

- $q_\pi(s,a)$ is decomposed into two streams: $v_\pi(s)$ and $A_\pi(s,a)$ that is called the advantage function

$$q_\pi(s,a) = A_\pi(s,a) + v_\pi(s)$$

- Interpretation of $A_\pi(s,a)$
  - positive for good actions, and negative for bad actions
- However, q_π is not uniquely divided into v_π and A_π
  - Use the constraint $\mathbb{E}_\pi[A^\pi(\boldsymbol{x},a)] = 0$
- Dueling network improved the performance significantly





[Wang et al., 2016]

Prioritized Experience Replay

- In the original Experience Replay, experience transitions were uniformly sampled from a replay buffer
  ➡ replay transitions even if they are not significant
- The priority of transition is determined by TD error:
  - proportional
    $$P(i) \propto [|\delta_i| + \varepsilon]^\alpha$$
  - rank-based
    $$P(i) \propto [\text{rank of } |\delta_i|]^\alpha$$
- Correction by importance sampling

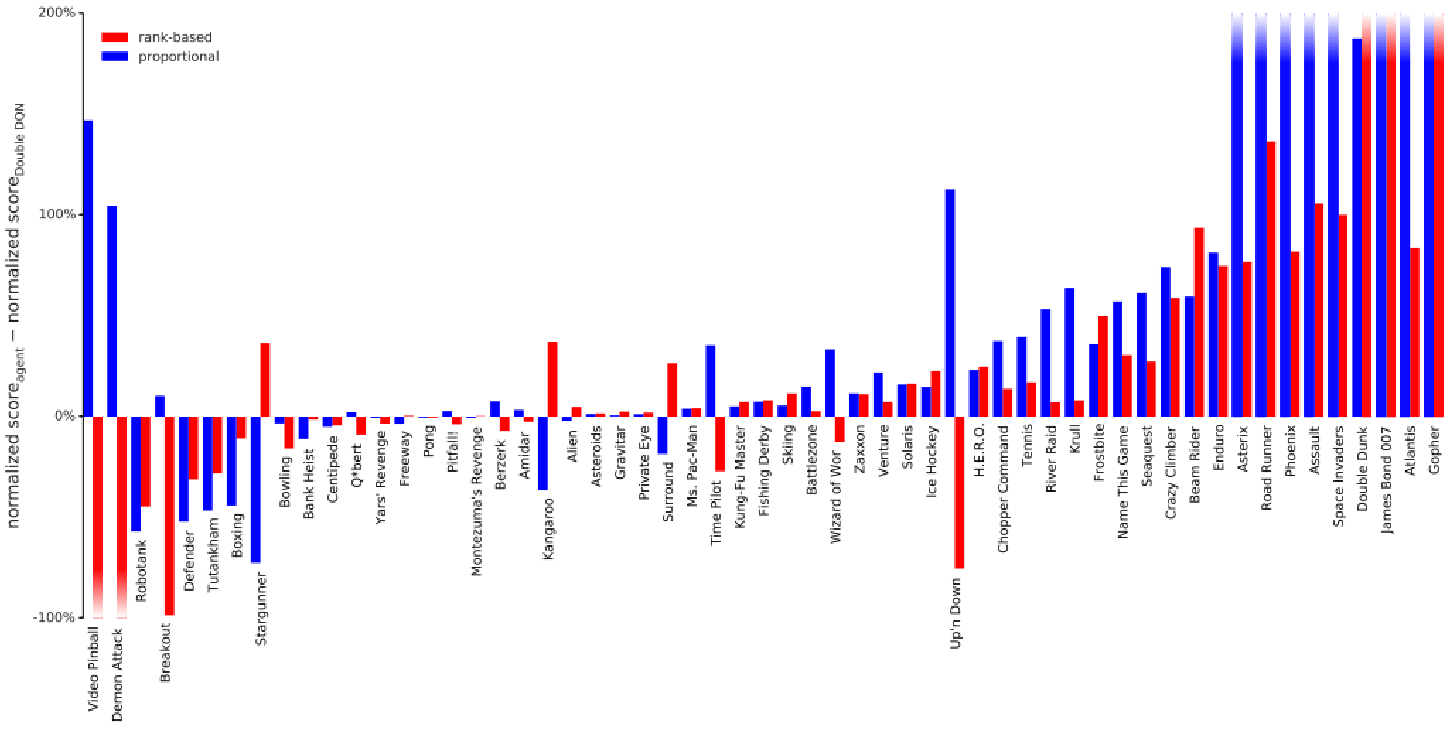


Figure 3: Difference in normalized score (the gap between random and human is 100%) on 57 games with human starts, comparing Double DQN with and without prioritized replay (rank-based variant in red, proportional in blue), showing substantial improvements in most games. Exact scores

[Schaul et al., 2016]

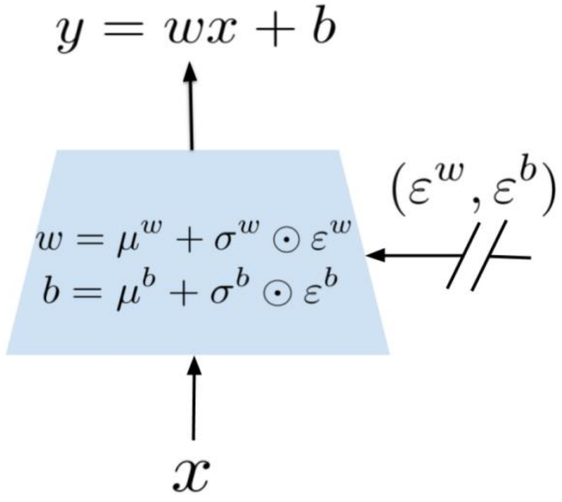
Exploration in the parameter space

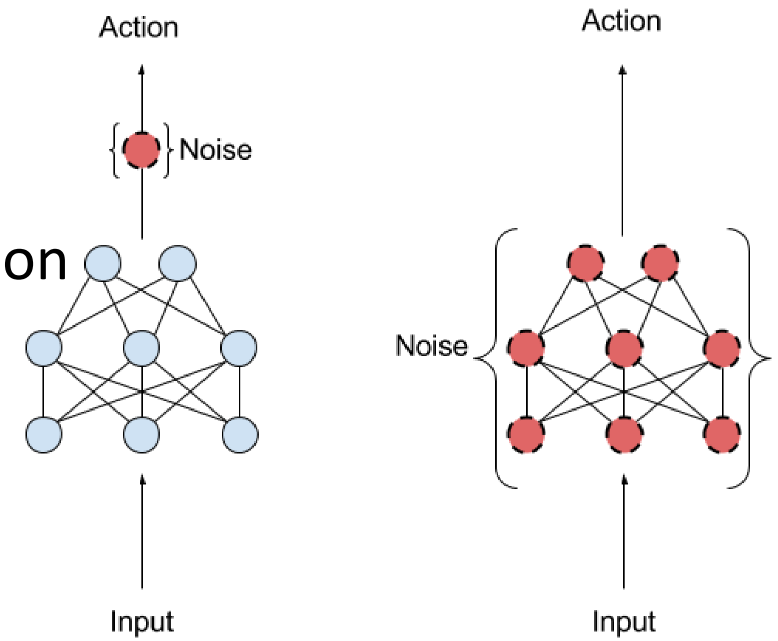
- Action selection in value-based RL such as DQN
  - softmax
  - ε-greedy

  Exploration in the action space
  modify a probability of action selection

  RL cannot tran the hyper parameters such as
  ε and β that control randomness

- Disturbance of the weights of the neural network





noise in the action space

noise in the parameter space

[OpenAI Blog]

[Fortunato et al., 2018; Plappert et al., 2018]

References

- Boyan, J.A., and Moore, A.W. Generalization in Reinforcement Learning: Safely Approximating the Value Function. NIPS 7, pp.369-376, 1995.
- Finn, C., Levine, S., and Abbeel, P. Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization. In Proc. of ICML, 2016.
- Fortunato, M., et al. Noisy networks for exploration. In Proc. of ICLR, 2018.
- Lake, B.M., Ullman, T.D., Tenenbaum, J.B., and Gershman, S.J. (2017). Building machines that learn and think like people. Behavioral and Brain Sciences, vol. 40, e253.
- Peters, J., Mülling, K., Altün, Y. Relative Entropy Policy Search, AAAI 2010.
- Plappert, M., et al. Parameter space noise for exploration. In Proc. of ICLR, 2018.
- Ross and Bagnell, 2010.
- Riedmiller, M. Neural fitted Q iteration - First experiences with a data efficient neural reinforcement learning method. In Proc. of ECML, pp. 317–328, 2005.
- Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In Proc. of ICLR, 2016.
- Silver, D., et al. Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484–489, 2016.

References

- Silver, D., et al. Mastering the game of Go without human knowledge. Nature, 550(7676), 354–359, 2017.
- Sutton, R.S.. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. NIPS 8, pp. 1038-1044, 1996.
- Tsurumine, Y., Cui, Y., Uchibe, E., and Matsubara, T. Deep Dynamic Policy Programming for Robot Control with Raw Images. In Proc. of IROS, 2017.
- Uchibe, E. and Doya, K. Inverse reinforcement learning using dynamic policy programming. In Proc. of ICDL and Epirob, 2014.
- Uchibe, E. Model-Free Deep Inverse Reinforcement Learning by Logistic Regression. Neural Processing Letters, 47(3): 891-905, 2018.
- Wang, Z., Schaul, T., Hessel, M., van Hasselt, H. Lanctot, M., and de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In Proc. of ICML.
- Wigness, M., Rogers III, J.G., Navarro-Serment, L.E. Robot Navigation from Human Demonstration: Learning Control Behaviors. In Proc. of ICRA 2018.

References

- Zhang, M., McCarthy, Z., Finn, C., Levine, S., and Abbeel, P. Learning Deep Neural Network Policies with Continuous Memory States. In Proc. of ICRA, 2016.
- Ziebart, B., et al. Maximum entropy inverse reinforcement learning. In Proc. of AAAI, 2008.
- Ziebart, B., et al. Planning-based predictions for pedestrians. In Proc. of IEEE/RSJ IROS, 2009.