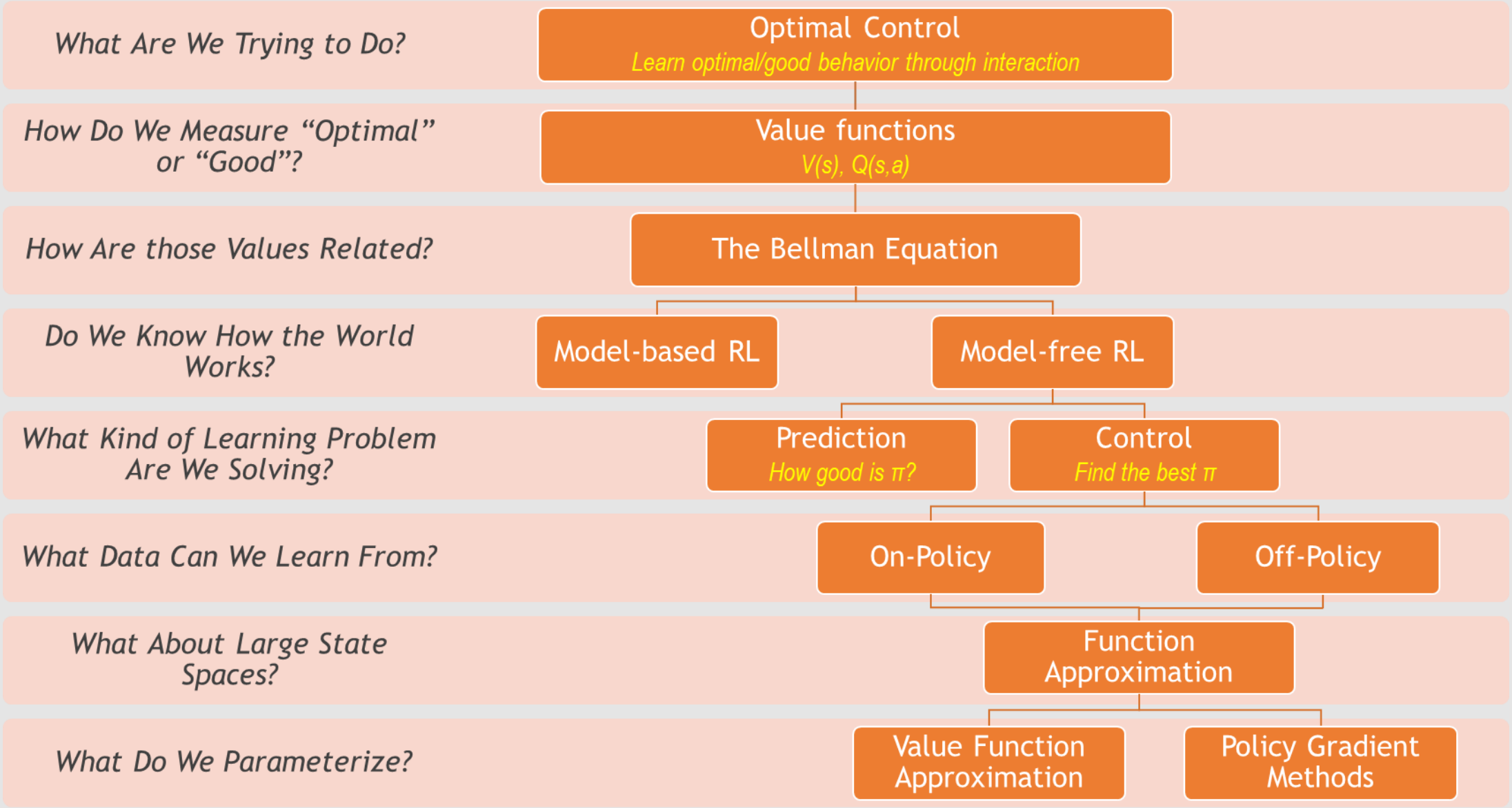
Value Function Approximation

Iftekharul Islam

Contents

Motivation

Our RL Journey So Far



Motivation: Problems with Tabular Learning

- So far, we've been representing state and action values by tables.

State	s_1	s_2	$\cdots$	s_n
Value	$v_\pi(s_1)$	$v_\pi(s_2)$	$\cdots$	$v_\pi(s_n)$

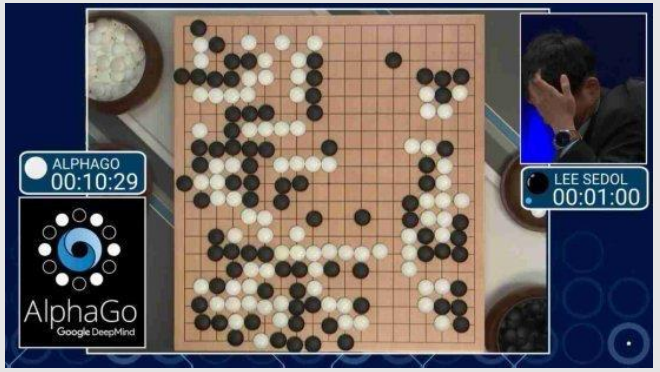
	a_1	a_2	a_3	a_4	a_5
s_1	$q_\pi(s_1, a_1)$	$q_\pi(s_1, a_2)$	$q_\pi(s_1, a_3)$	$q_\pi(s_1, a_4)$	$q_\pi(s_1, a_5)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_9	$q_\pi(s_9, a_1)$	$q_\pi(s_9, a_2)$	$q_\pi(s_9, a_3)$	$q_\pi(s_9, a_4)$	$q_\pi(s_9, a_5)$

- **Advantage:** Intuitive and simpler to analyze

Image: Shiyu Zhao, *Value Function Methods*

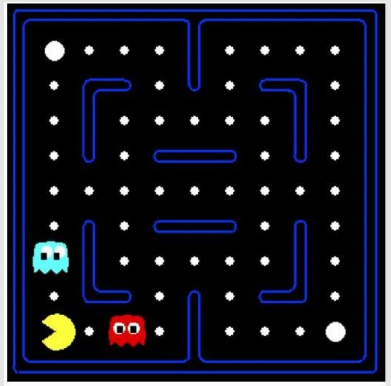
Motivation: Problems with Tabular Learning

- **Disadvantages:** impractical for real-world problems
  1. **Storage**
     - Backgammon: 10^{20} states
     - Computer Go: 10^{170} states
     - Robot arm: infinite number of states! (continuous)
  2. **generalization ability**



www.youtube.com/watch?v=HT-UZkiOLv8

Let's say we discover through experience that this state is bad:



In naive Q-learning we know nothing about this state:

Or even this one:

Images: Pieter Abbeel, CS 188 Introduction to Artificial Intelligence. Fall 2018

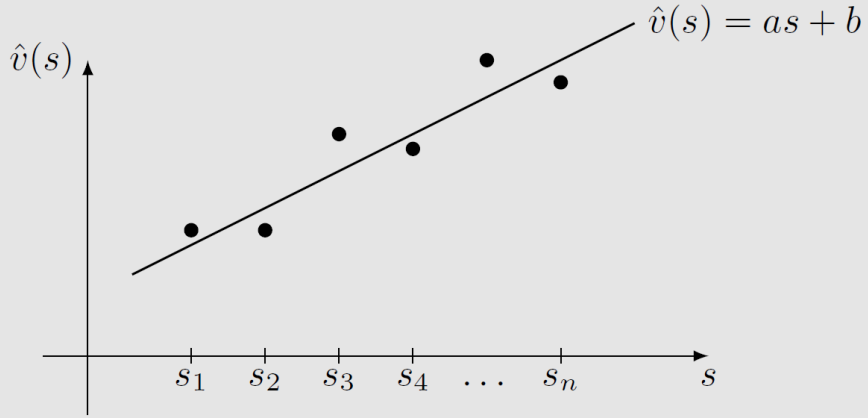
Motivation: From Table to Function

Example:

- n states: $s_1, \ldots, s_n$
- State values are $v_\pi(s_1), \ldots, v_\pi(s_n)$, where π is a given policy.
- n is very large!
- We want to use a simple curve to approximate these values.

Motivation: From Table to Function

We can fit a **straight line** to the dots.



Suppose the equation of the straight line is

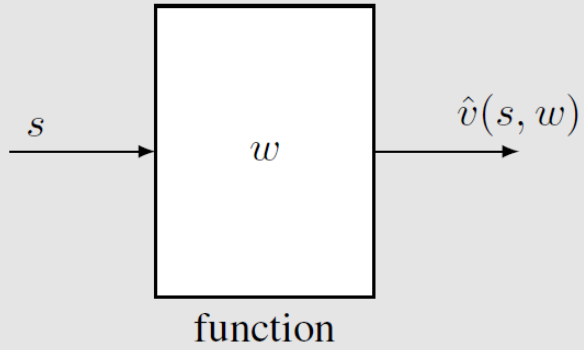
$$\hat{v}(s,w) = as + b = \underbrace{[s,1]}_{\phi^T(s)} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{w} = \phi^T(s)w$$

w is the parameter vector; φ(s) the feature vector of s; $\hat{\boldsymbol{v}}(\boldsymbol{s},\boldsymbol{w})$ is linear in w.

Image: Shiyu Zhao, *Value Function Methods*

Motivation: From Table to Function

- **Retrieving a state's value:**
- **Tabular method:** Direct lookup from the stored table
- **Functional method:** Compute by passing state through the function

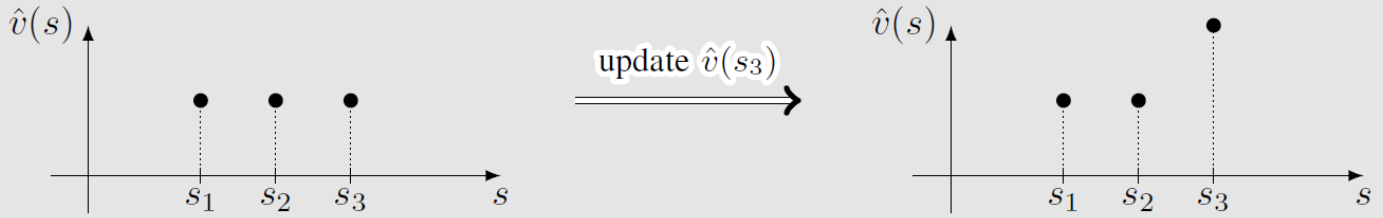


- **Advantage:** Memory efficiency. Rather than maintaining |S| separate values, we only store a compact parameter vector w.

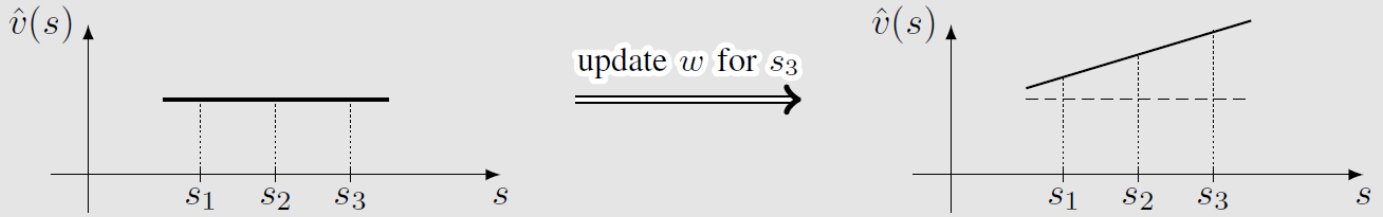
Image: Shiyu Zhao, *Value Function Methods*

Motivation: From Table to Function

- **Key distinction in updating values:**



(a) Tabular method



(b) Function method

- **Advantage:** Ability to generalize. Adjusting w for one state automatically influences estimates for similar states.

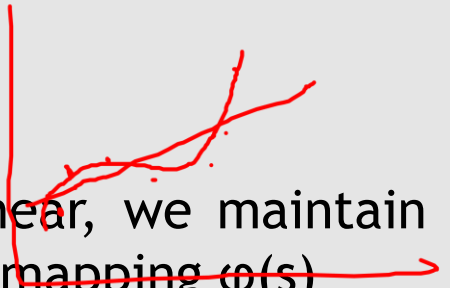
Image: Shiyu Zhao, *Value Function Methods*

Motivation: From Table to Function

- The benefits come at a cost: **perfect representation isn't achievable.**
- We can achieve more accurate fits **using higher-degree polynomials:**

$$\hat{v}(s,w) = as^2 + bs + c = \underbrace{[s^2, s, 1]}_{\phi^T(s)} \underbrace{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}_{w} = \phi^T(s)w.$$

- **Key observations:**
  - Increasing dimensions of w and φ(s) can improve accuracy
  - While the relationship between states and values may be nonlinear, we maintain linearity in parameters by encoding complexity within the feature mapping φ(s)



Key Ideas

- **Core concept:** Use parameterized functions to approximate values:

$$\hat{v}(s, w) \approx v_\pi(s)$$

  where $w \in \mathbb{R}^m$ contains the parameters

- **Fundamental distinction:** How we access and modify v(s)

- **Key benefits:**
  - **Memory efficiency:** The dimensionality of w can be far smaller than |S|
  - **Generalization capability:** Updating w for a visited state propagates information to unvisited states

Algorithm for State Value Estimation

Formal Problem Setup

Let's formalize the approach:

- $v_\pi(s)$ denotes the true state value
- $\hat{v}(s, w)$ represents our approximation
- **Objective:** Identify optimal w that minimizes approximation error across all states

Finding optimal w requires two components:

1. Defining an **objective function**
2. Developing **optimization algorithms**

Objective Function

- Our **objective function:** $J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$
- **Goal:** Find the $\boldsymbol{w}$ that minimizes $\boldsymbol{J}(\boldsymbol{w})$
  - We can use the **gradient-descent** algorithm:

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

  - Gradient descent finds a local minimum
  - To find global minimum, we use **Stochastic gradient descent (SGD)** which samples the gradient:

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

  - Expected update is equal to full gradient update

Algorithms for Prediction

- But we do not know the true value function $v_\pi(s)$
- Replace $v_\pi(s)$ with a ***target***:

$$\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S, \mathbf{w})$$

Method	Target
Monte-Carlo	G_t
TD(0)	$R_{t+1} + \gamma\hat{v}(s_{t+1}, \mathbf{w})$
TD(λ)	G_t^λ

Algorithms for Control

- We can use similar approach and replacements for $\boldsymbol{q_\pi(s, a)}$:

$$\Delta \mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$$

Method	Target
Monte-Carlo	G_t
SARSA	$R_{t+1} + \gamma\hat{q}(s_{t+1}, a_{t+1}, \mathbf{w})$
Q-learning	$R_{t+1} + \gamma max_{a_{t+1}}\hat{q}(s_{t+1}, a_{t+1}, \mathbf{w})$

- Control with Function Approximator
  - policy evaluation: approximated policy evaluation $\widehat{\boldsymbol{Q}}(.,.,\boldsymbol{w}) \approx \boldsymbol{q_\pi}$
  - policy improvement: ϵ-greedy

Linear Value Function Approximation

- We represent value function by a linear combination of features

- **Features** are functions from states to real numbers that capture important properties of the state

- Example features for Pac Man:
  - Distance to closest ghost
  - Distance to closest dot
  - Number of ghosts

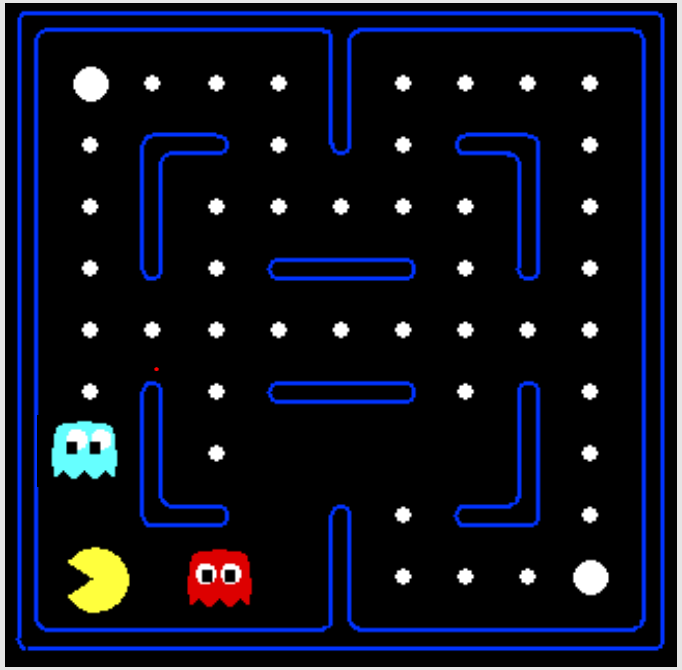


Image: Shiyu Zhao, *Value Function Methods*

Linear Value Function Approximation

- We represent value function by a linear combination of features

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \varphi(S)^\top \mathbf{w})^2 \right]$$

- Stochastic gradient descent converges on global optimum
- Update rule:

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \varphi(S)$$

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \varphi(S)$$

- Update = step-size × prediction error × feature value

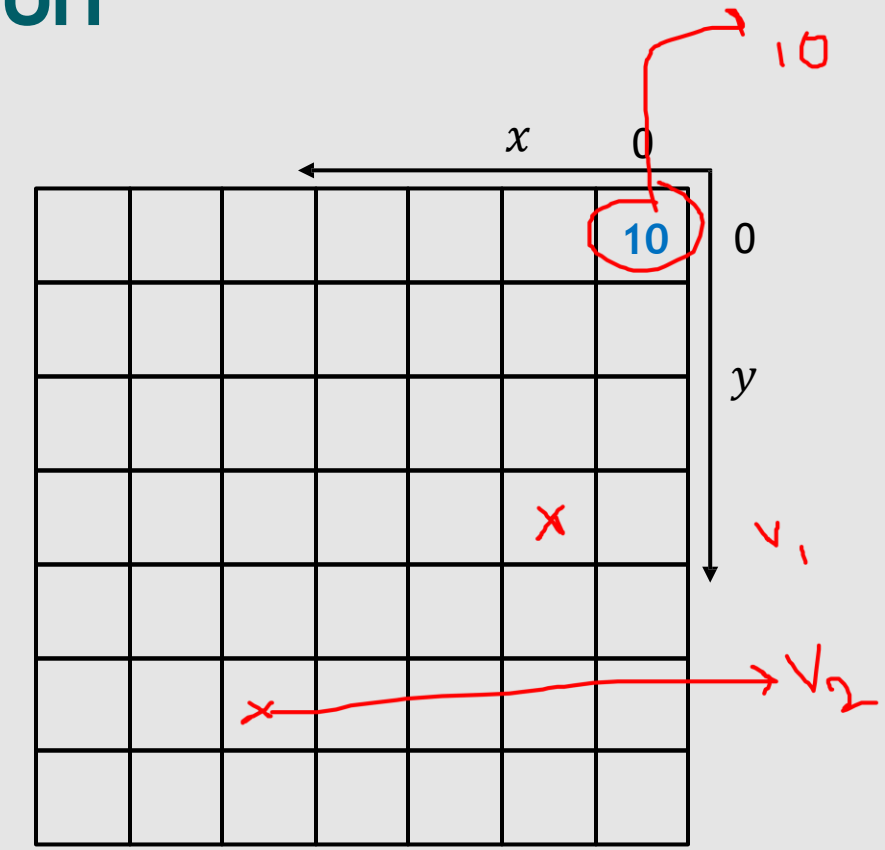
Linear Value Function Approximation

- Two key questions to address:
  1. Can we approximate any V-/Q-value function with a linear FA?
     - **Yes!** (But the proof is out of this class.)
  2. Is it easy to find such a linear FA?

Linear Value Function Approximation

Example: Grid world problem

- No obstacles
- Actions (deterministic): Up/Down/Left/Right
- No discounting
- Reward: **+10** at goal, **−1** everywhere else



- Represent state s by a feature vector:

$$\phi(s) = \begin{bmatrix} 1 \\ x \\ y \end{bmatrix}$$

- Perform linear VFA:

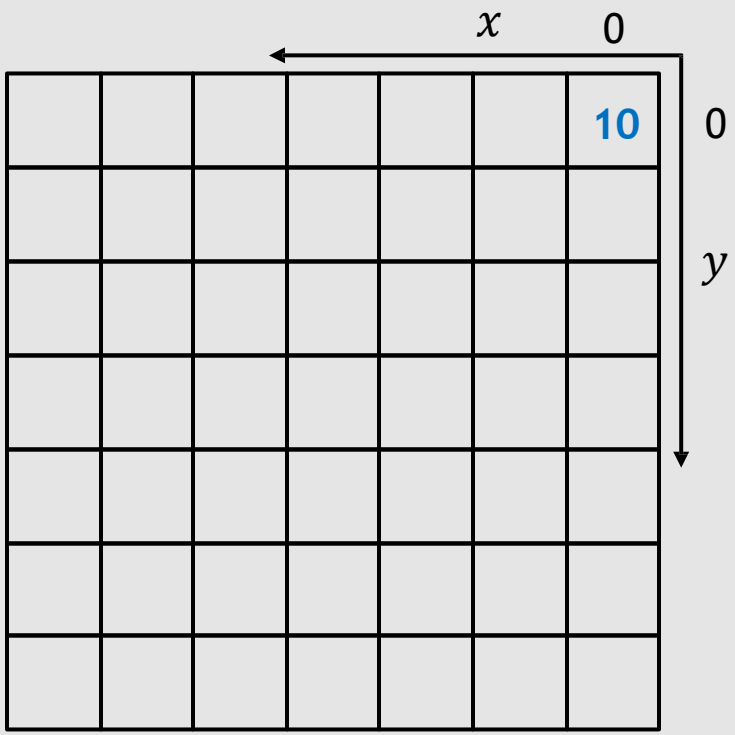
$$\hat{v}(s,w) = \phi(s)^T w = \begin{bmatrix} 1 & x & y \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

$$= w_0 + w_1 x + w_2 y$$

Linear Value Function Approximation

Example: Grid world problem

- No obstacles
- Actions (deterministic): Up/Down/Left/Right
- No discounting
- Reward: **+10** at goal, **−1** everywhere else



$$\hat{v}(s,w) = \phi(s)^T w = \begin{bmatrix} 1 & x & y \end{bmatrix} \begin{bmatrix} 10 \\ -1 \\ -1 \end{bmatrix}$$

$$= 10 - x - y$$

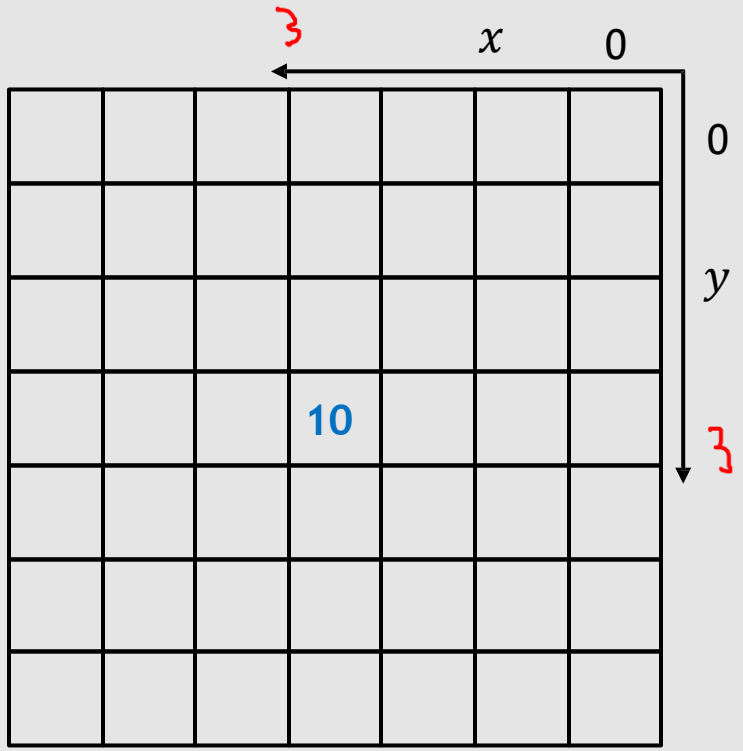
- (Manhattan distance.)

Is there a good linear approximation? → **YES**

Linear Value Function Approximation

Example: Grid world problem

- No obstacles
- Actions (deterministic): Up/Down/Left/Right
- No discounting
- Reward: **+10** at goal, **−1** everywhere else



- But what if we move the reward?

$$\hat{v}(s,w) = \phi(s)^T w = \begin{bmatrix} 1 & x & y \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

$$= w_0 + w_1 x + w_2 y$$

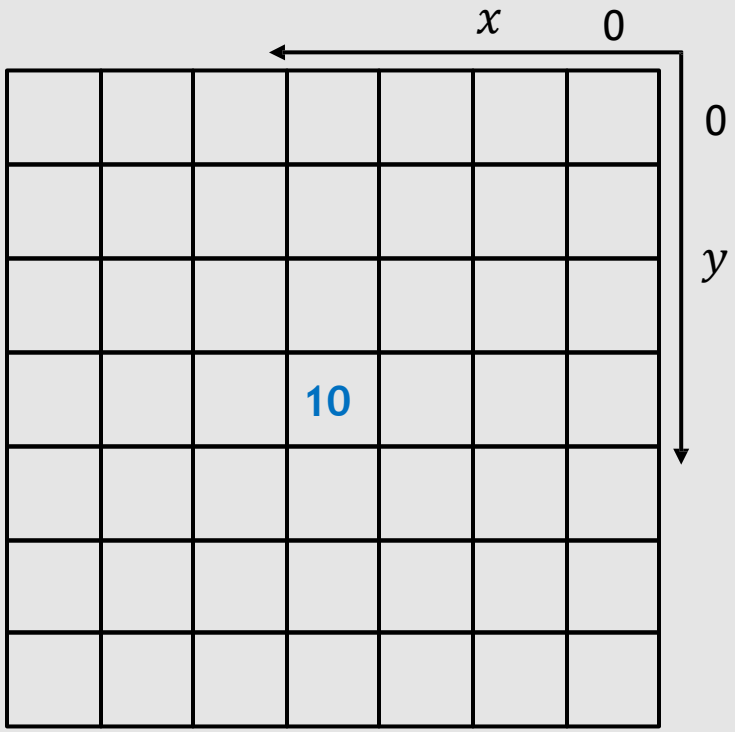
Is there a good linear approximation? → **NO**

Unless…

Linear Value Function Approximation

Example: Grid world problem

- No obstacles
- Actions (deterministic): Up/Down/Left/Right
- No discounting
- Reward: **+10** at goal, **−1** everywhere else



…we add a new feature z.

$$\hat{v}(s,w) = \phi(s)^T w = \begin{bmatrix} 1 & x & y & z \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$$= w_0 + w_1 x + w_2 y + w_3 z$$

Is there a good linear approximation now? → **YES**

$$z = |3 - x| + |3 - y|$$

(I'll leave the figuring out the w_i values to you.
***Hint**: one or more w_i will be 0.)*

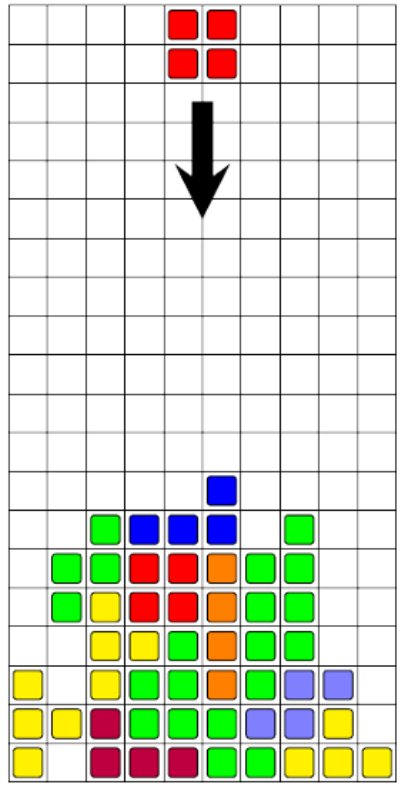
Linear Value Function Approximation

Example: Tetris

- state: board configuration + shape of the falling piece ~2^{200} states!
- action: rotation and translation applied to the falling piece
- 22 features aka basis functions ϕ_i
  - Ten basis functions, 0, . . . , 9, *mapping the state to the height h[k] of each* column.
  - Nine basis functions, 10, . . . , 18, *each mapping the state to the absolute difference* between heights of successive columns: $|h[k+1] - h[k]|$, $k = 1, \ldots, 9$.
  - One basis function, 19, that maps state to the maximum column height: $\max_k h[k]$
  - One basis function, 20, that maps state to the number of 'holes' in the board.
  - One basis function, 21, that is equal to 1 in every state.

$$\hat{V}_\theta(s) = \sum_{i=0}^{21} \theta_i \phi_i(s) = \theta^\top \phi(s)$$

[Bertsekas & Ioffe, 1996 (TD); Bertsekas & Tsitsiklis 1996 (TD); Kakade 2002 (policy gradient); Farias & Van Roy, 2006 (approximate LP)]



Is it easy to find such a linear FA? → **No!**

Image: Pieter Abbeel, *Advanced Robotics, Fall 2019*

Linear Value Function Approximation

Example: Pacman

V(s) = θ_0

+ θ_1 "distance to closest ghost"
+ θ_2 "distance to closest power pellet"
+ θ_3 "in dead-end"
+ θ_4 "closer to power pellet than ghost"
+ ...

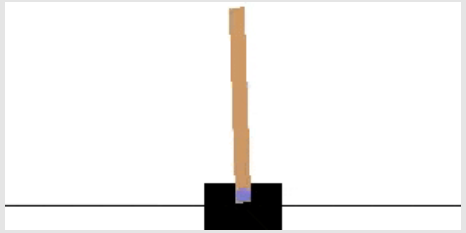
$$= \sum_{i=0}^{n} \theta_i \phi_i(s) = \theta^\top \phi(s)$$

Is it easy to find such a linear FA? → **No!**

Image: Pieter Abbeel, *Advanced Robotics, Fall 2019*

Linear Value Function Approximation

Example: Cartpole



We applied LSPI with a set of 10 basis functions for each of the 3 actions, thus a total of 30 basis functions, to approximate the value function. These 10 basis functions included a constant term and 9 radial basis functions (Gaussians) arranged in a 3×3 grid over the 2-dimensional state space. In particular, for some state $s = (\theta, \dot{\theta})$ and some action a, all basis functions were zero, except the corresponding active block for action a which was

$$\left(1,\ e^{-\frac{\|s-\mu_1\|^2}{2\sigma^2}},\ e^{-\frac{\|s-\mu_2\|^2}{2\sigma^2}},\ e^{-\frac{\|s-\mu_3\|^2}{2\sigma^2}},\ \ldots,\ e^{-\frac{\|s-\mu_9\|^2}{2\sigma^2}} \right)^{\mathsf{T}},$$

where the μ_i's are the 9 points of the grid $\{-\pi/4,\ 0,\ +\pi/4\} \times \{-1,\ 0,\ +1\}$ and $\sigma^2 = 1$.

Lagoudakis, M. G. and Parr, R.: Least-Squares Policy Iteration. JMLR. 2003

Is it easy to find such a linear FA? → **No!**

Linear Value Function Approximation

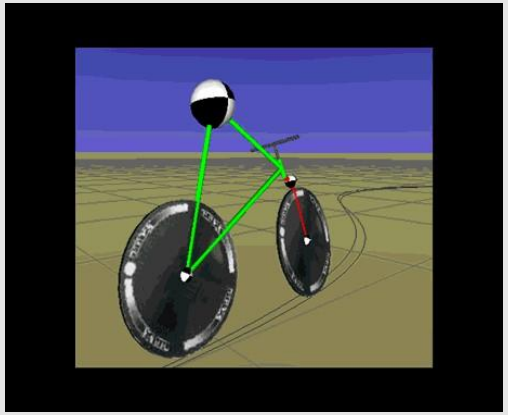
Example: Bicycle balancing

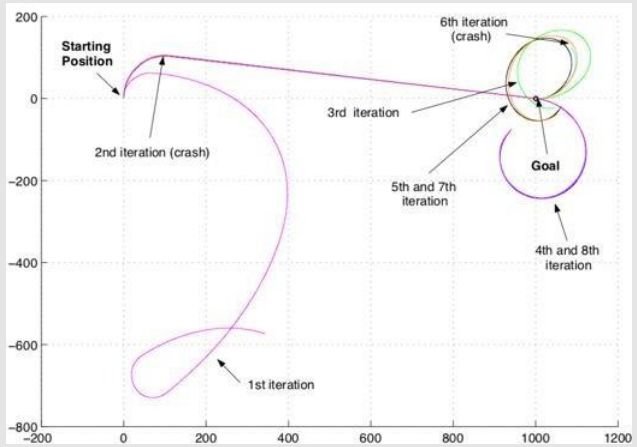
The goal in the *bicycle balancing and riding* problem (Randløv and Alstrøm, 1998) is to learn to balance and ride a bicycle to a target position located 1 km away from the starting location. Initially, the bicycle's orientation is at an angle of 90° to the goal. The state description is a six-dimensional real-valued vector $(\theta, \dot{\theta}, \omega, \dot{\omega}, \ddot{\omega}, \psi)$, where θ is the angle of the handlebar, ω is the vertical angle of the bicycle, and ψ is the angle of the bicycle to the goal. The actions are the torque τ applied to the handlebar (discretized to $\{-2, 0, +2\}$) and the displacement of the rider υ (discretized to $\{-0.02, 0, +0.02\}$). In our experiments, actions are restricted so that either $\tau = 0$ or $\upsilon = 0$ giving a total of 5 actions.[12] The noise in the system is a uniformly distributed term in $[-0.02, +0.02]$ added to the displacement component of the action. The dynamics of the bicycle are based on the model of Randløv and Alstrøm (1998) and the time step of the simulation is set to 0.01 seconds.

The state-action value function $Q(s, a)$ for a fixed action a is approximated by a linear combination of 20 basis functions:

$$\left(1,\ \omega,\ \dot{\omega},\ \omega^2,\ \dot{\omega}^2, \omega\dot{\omega},\ \theta,\ \dot{\theta},\ \theta^2,\ \dot{\theta}^2,\ \theta\dot{\theta},\ \omega\theta,\ \omega\theta^2,\ \omega^2\theta,\ \psi,\ \psi^2,\ \psi\theta,\ \bar{\psi},\ \bar{\psi}^2,\ \bar{\psi}\theta\right)^{\intercal},$$

where $\psi = \pi - \psi$ for $\psi > 0$ and $\bar{\psi} = -\pi - \psi$ for $\psi < 0$. Note that the state variable $\ddot{\omega}$ is completely ignored. This block of basis functions is repeated for each of the 5 actions, giving a total of 100 basis functions (and parameters).





Lagoudakis, M. G. and Parr, R.: Least-Squares Policy Iteration. JMLR. 2003

Is it easy to find such a linear FA? → **No!**

Non-linear Value Function Approximation

- Why if we replace linear approximation with NNs?
  - Theory tells us that this doesn‘t work out

Convergence of Prediction Algorithms

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

G_t

Convergence of Control Algorithms

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗

(✓) = chatters around near-optimal value function

Images: David Siver, *Reinforcement Learning, 2016*

Non-linear Value Function Approximation

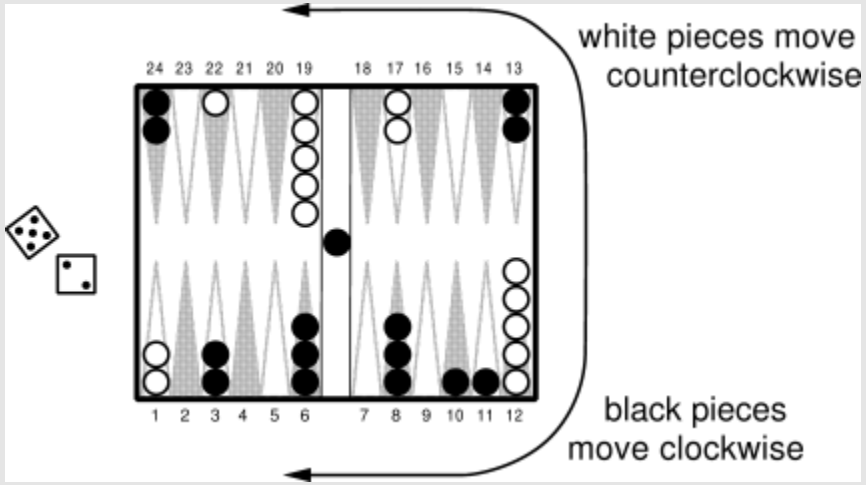
- Why if we replace linear approximation with NNs?
  - Theory tells us that this doesn‘t work out

- **The deadly triad**:
  1. **Function approximation**
  2. **Bootstrapping**
  3. **Off-policy learning**

  If you have **all three at once**, learning can fail (even when each pair alone might be fine).

Non-linear Value Function Approximation

- Why if we replace linear approximation with NNs?
  - Theory tells us that this doesn't work out
  - But in some cases, it did! ☺
  - Example: Gerry Tesauro's TD-Gammon (1992)
    - Neural network (NN) with 80 hidden units
    - Used RL for 300,000 games of self-play
    - One of the top players in the world!



Tesauro, G.: Temporal difference learning and TD-Gammon. 1995.

Image: http://incompleteideas.net/book/first/ebook/node108.html

Non-linear Value Function Approximation

- Why if we replace linear approximation with NNs?
  - Theory tells us that this doesn‘t work out

- Besides some few hand-crafted and tuned successes people did not manage to apply NNs “as is” to RL

Until…

Deep Q-Network (DQN)

DQN

Letter | Published: 25 February 2015

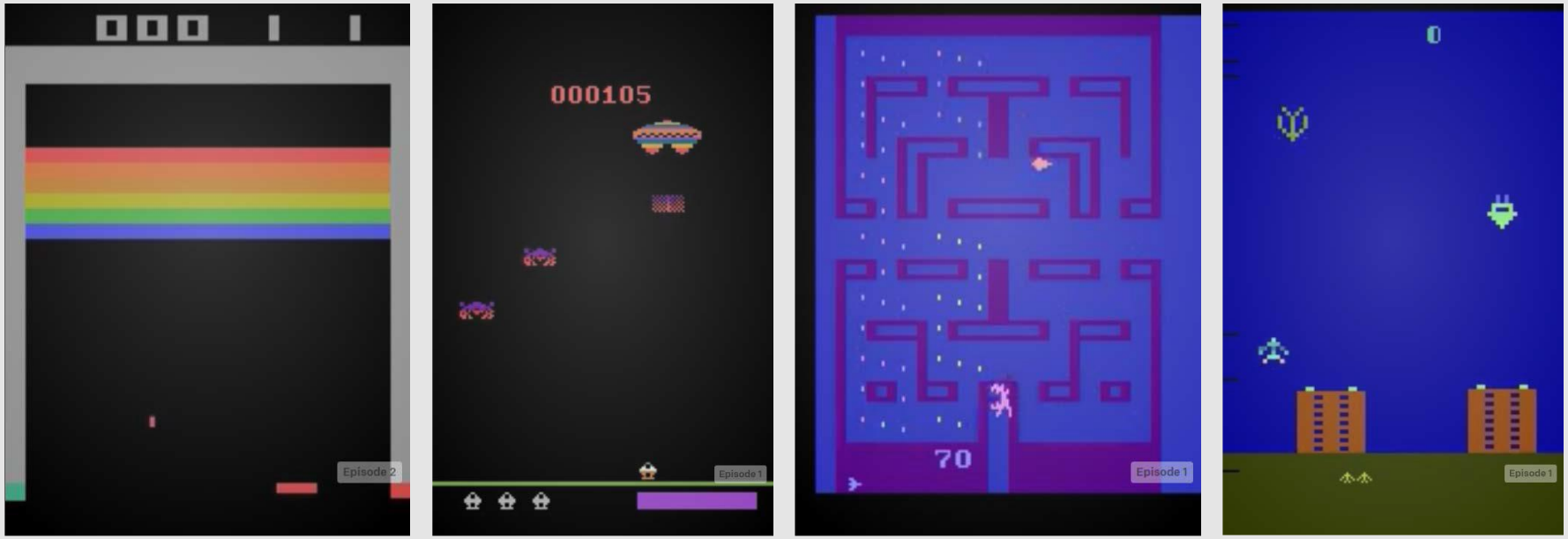
Human-level control through deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

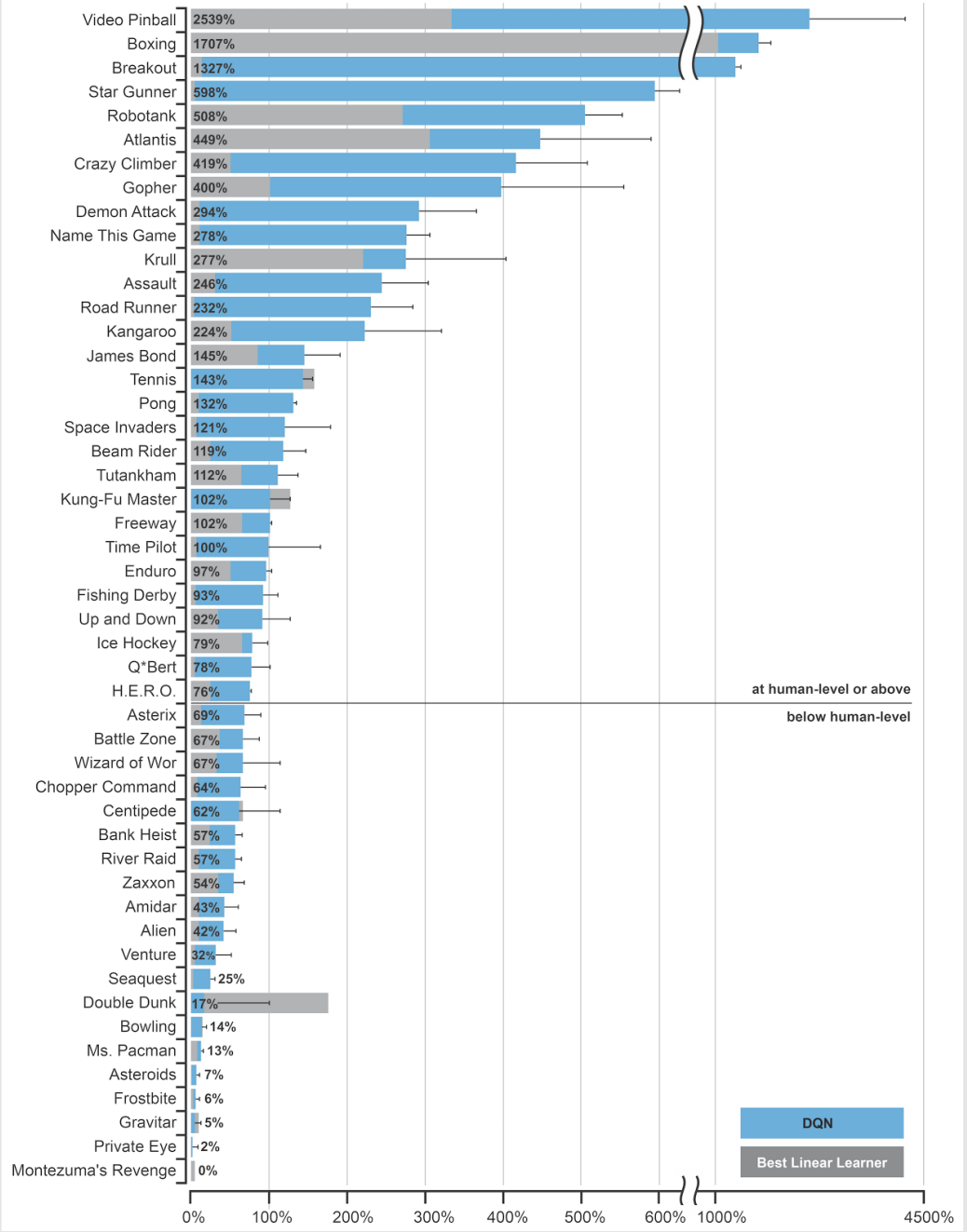
Nature **518**, 529–533 (2015) | Cite this article

DQN

- Surpassed human-level performance in 29 games of the Atari 2600 series
- **Same RL architecture** to learn a policy in each game
- **End-to-end learning** with just image pixels as input
- “The” contribution that started a round of huge investments in RL



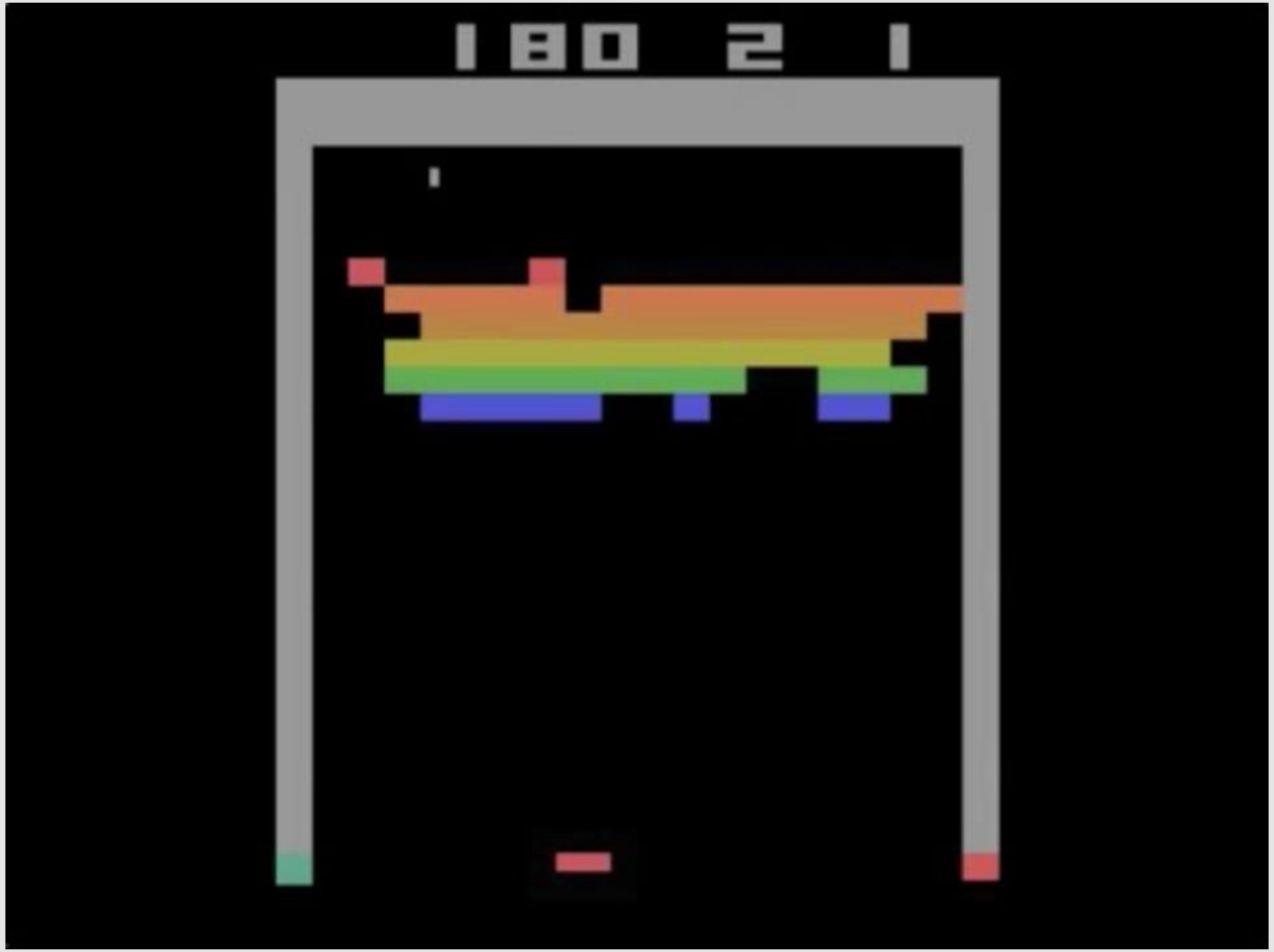
DQN



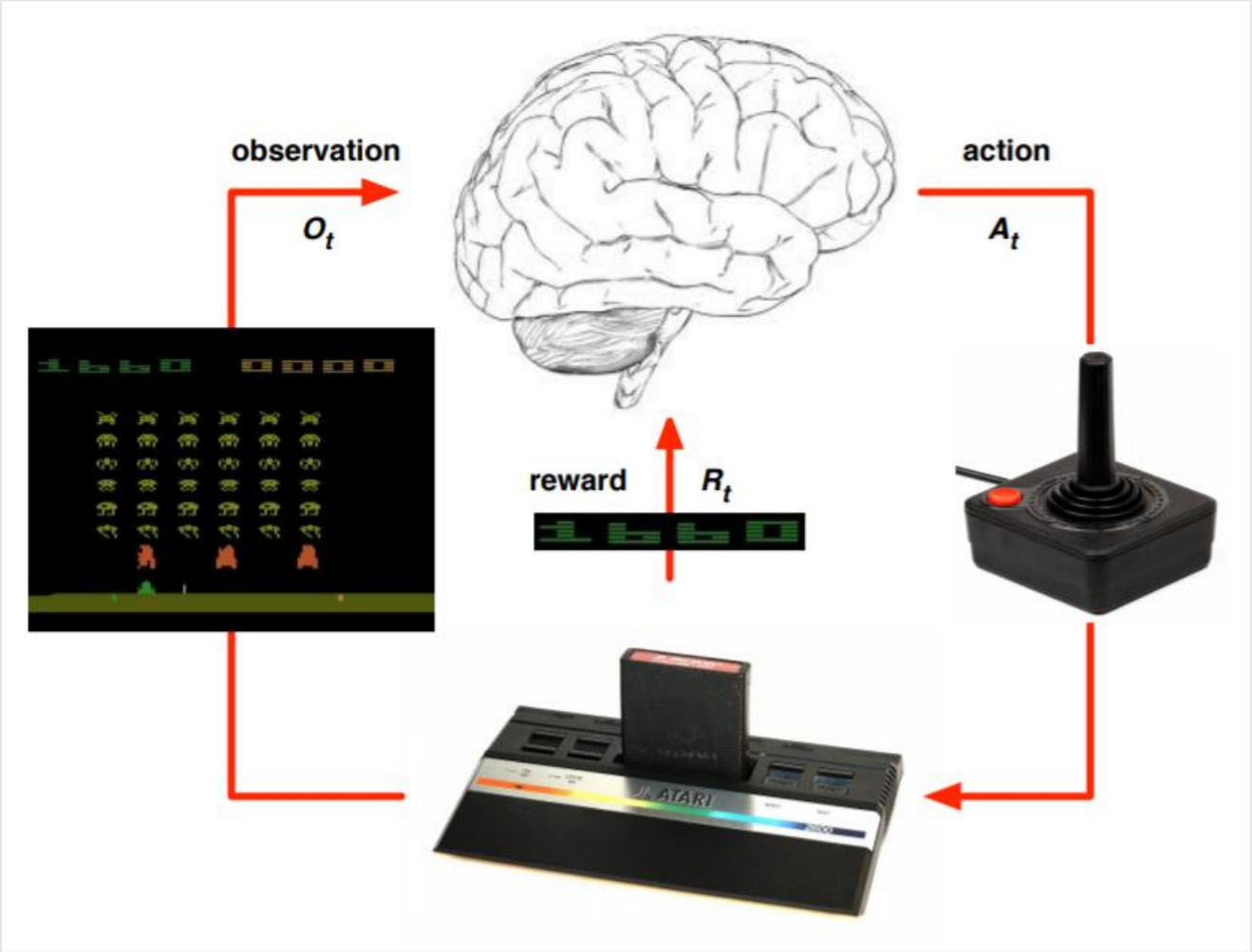
To learn more:

Mnih, Volodymyr, et al. "***Human-level control through deep reinforcement learning.***" nature 518.7540 (2015): 529-533.

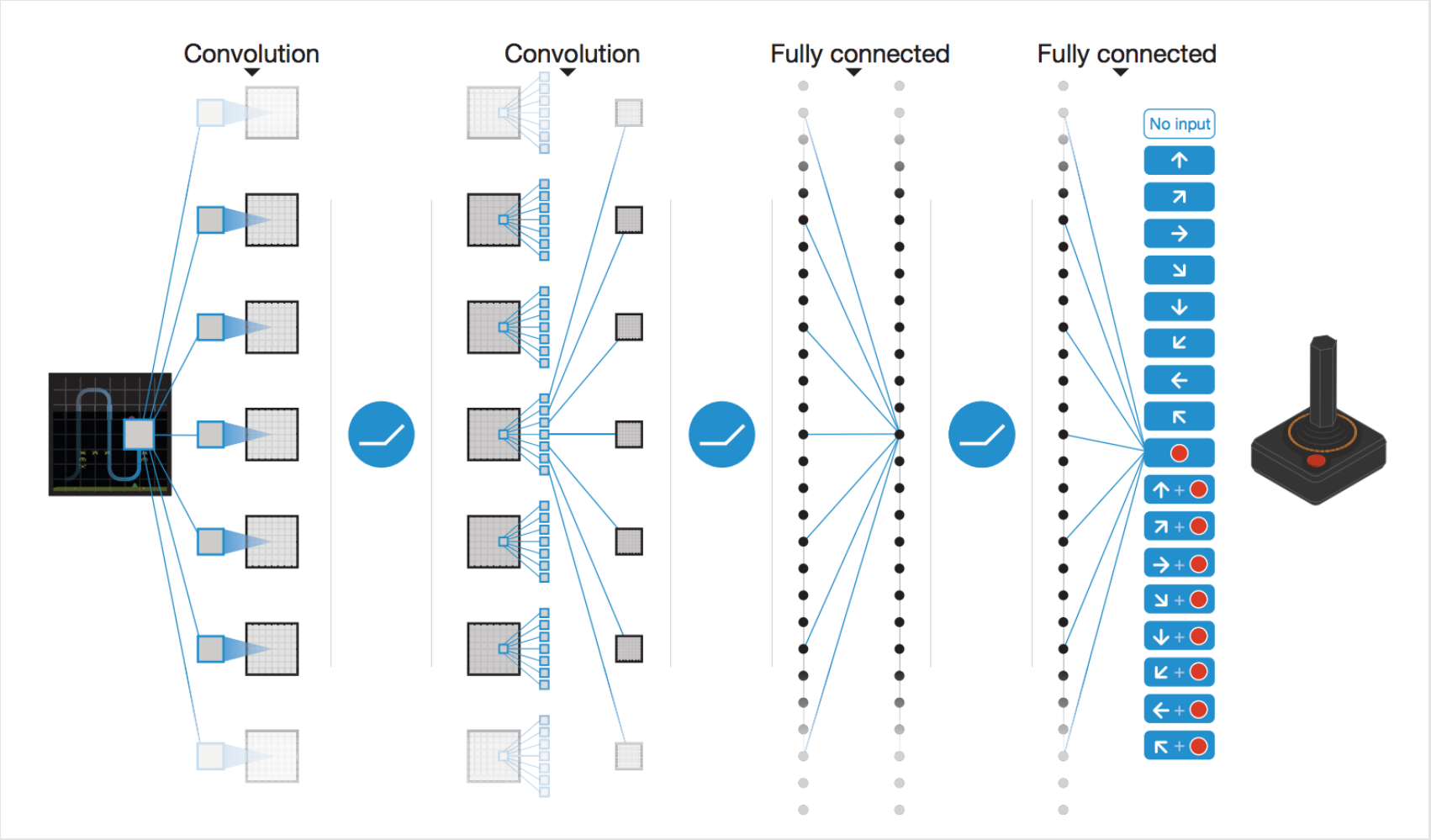
DQN



DQN



DQN

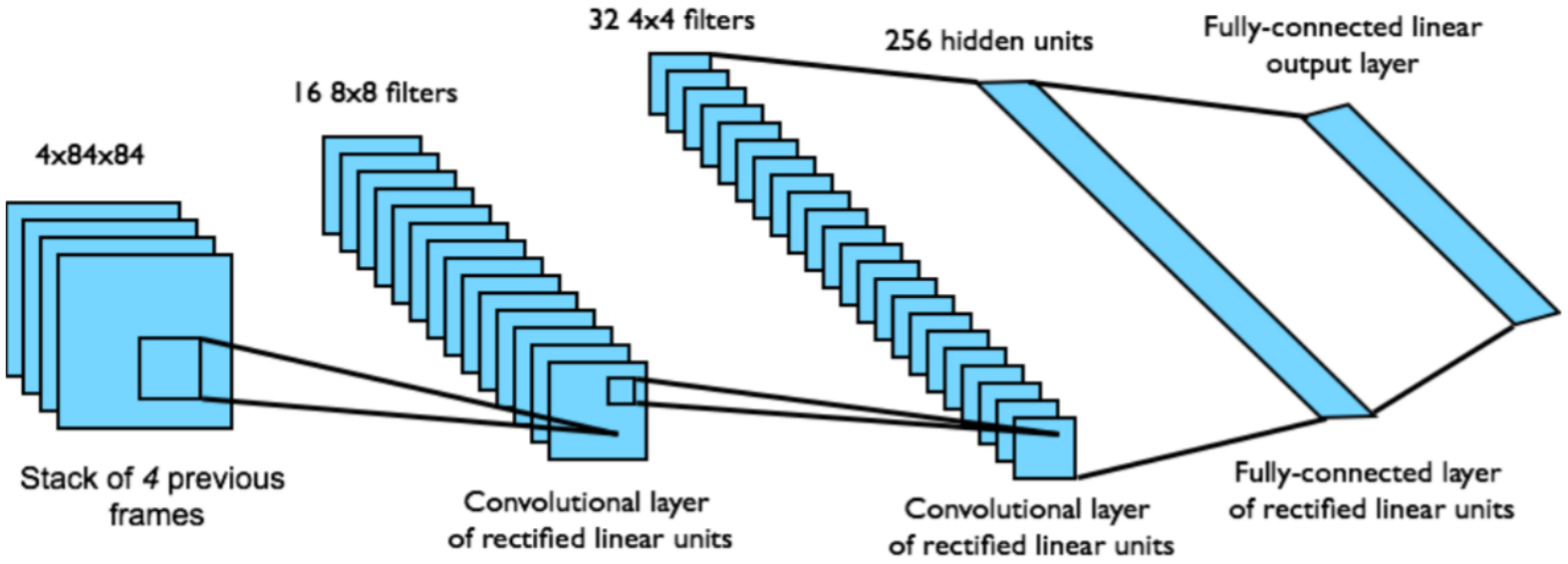


To learn more:

Mnih, Volodymyr, et al. "***Human-level control through deep reinforcement learning.***" nature 518.7540 (2015): 529-533.

DQN

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

DQN: How it works

- DQNs are essentially "Q-Learning on steroids" (Deep NN as VFA)
- **Objective function:** $L(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i}[(y_i - Q(s,a,w))^2]$

Q-Learning with Linear VFA

- Approximated Q-function with parameters w and feature vector ϕ:
  $$Q(s,a) \approx w^T \phi(s,a)$$
- Target value:
  $$y_i = r + \gamma \max_{a' \in \mathcal{A}} w_i^T \phi(s',a')$$
- Updating:
  $$w_{i+1} = w_i + \alpha[y_i - w_i^T \phi(s,a)]\phi(s,a)$$

Q-Learning with NN VFA

- Approximated Q-function with NN and parameters w:
  $$Q(s,a) \approx \hat{Q}(s,a;w^-)$$
- Target value:
  $$y_i = r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s',a';w^-)$$
- Updating:
  $$w_{i+1} = w_i + \alpha\left[y_i - \hat{Q}(s,a;w_i)\right]\nabla_{w_i}\hat{Q}(s,a;w_i)$$
  Every k steps: $w^- \leftarrow w_i$

Image: Christopher Mutschler, *Reinforcement Learning, 2025*

DQN: How it works

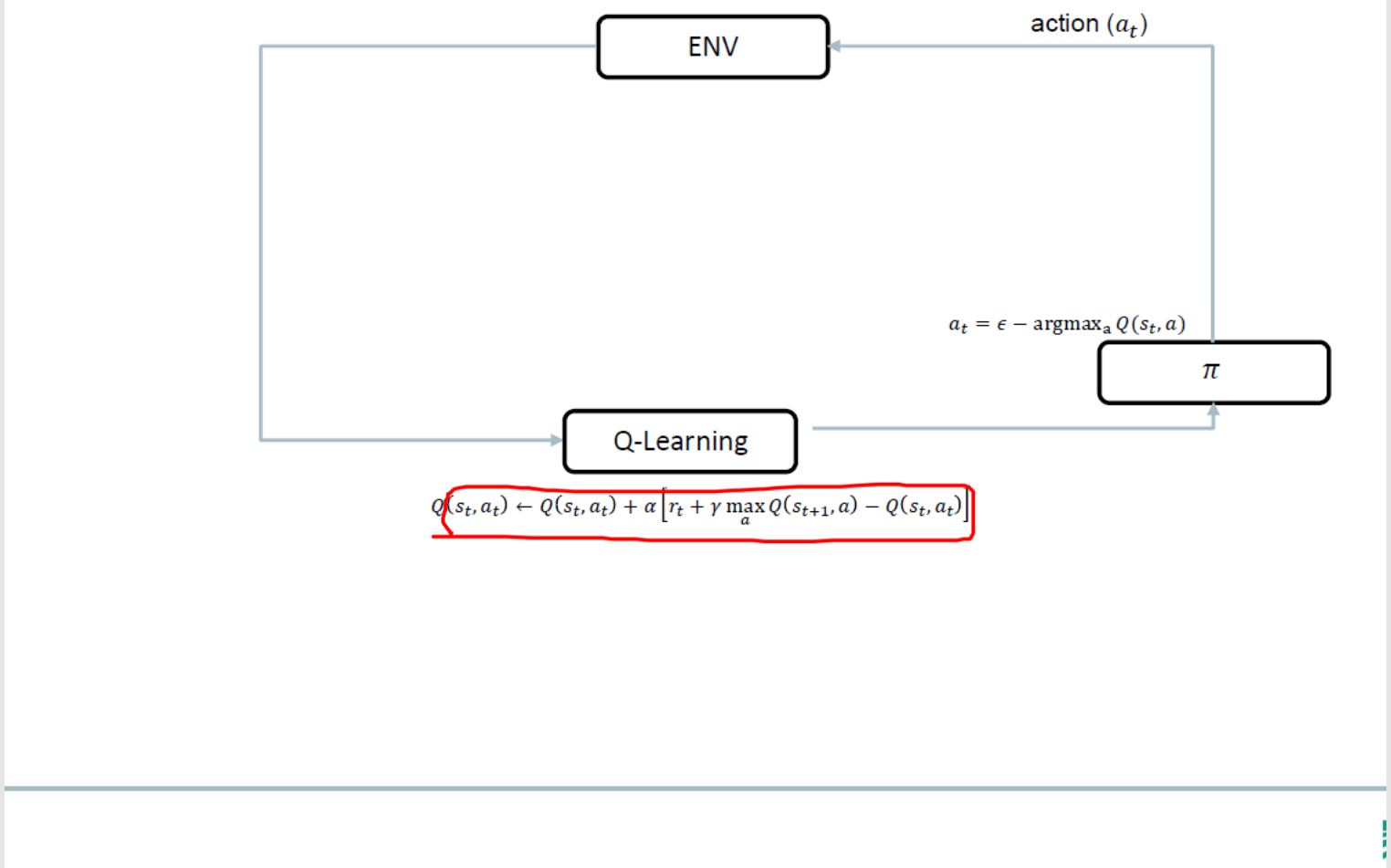


Image: Christopher Mutschler, *Reinforcement Learning, 2025*

DQN: How it works

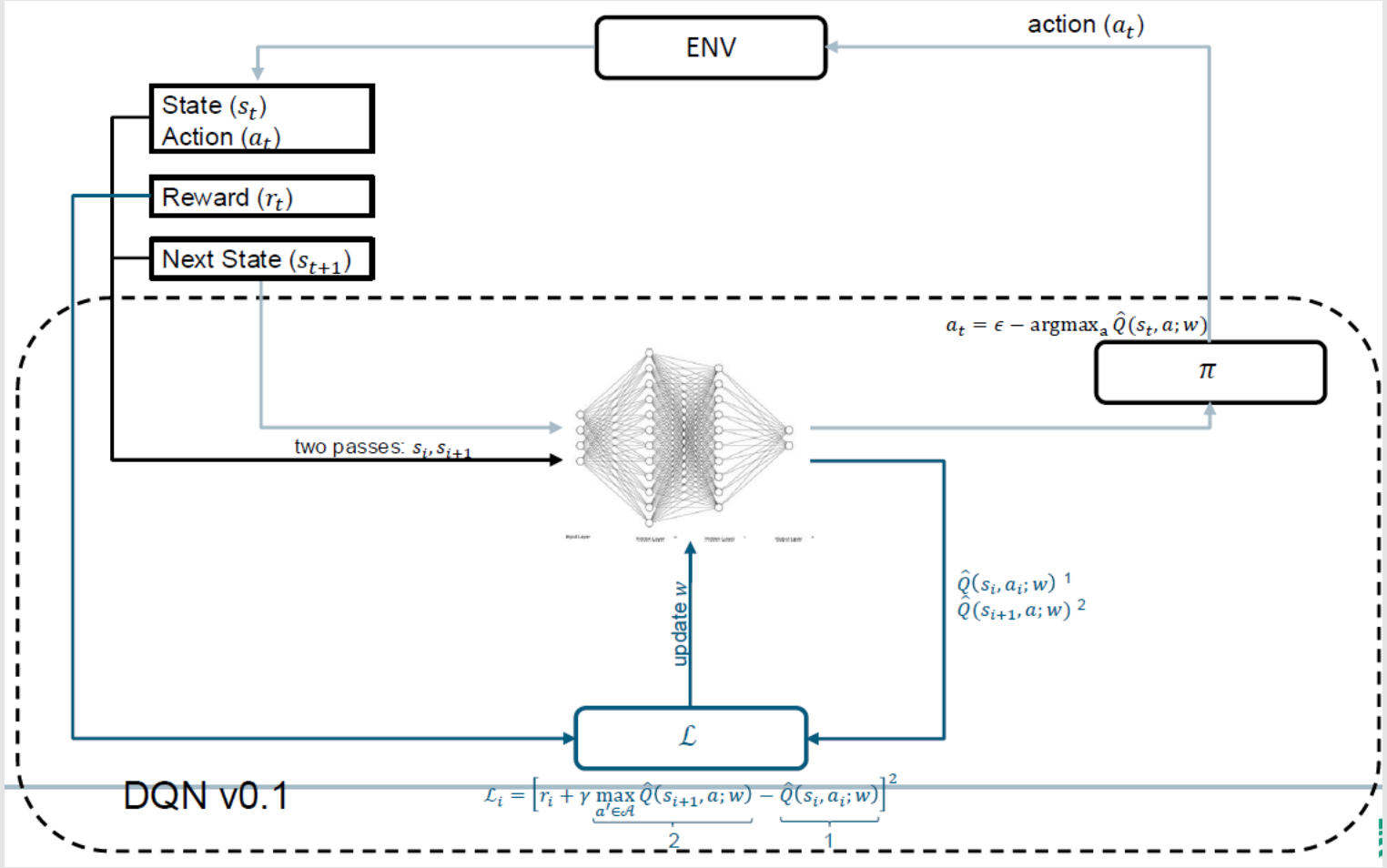
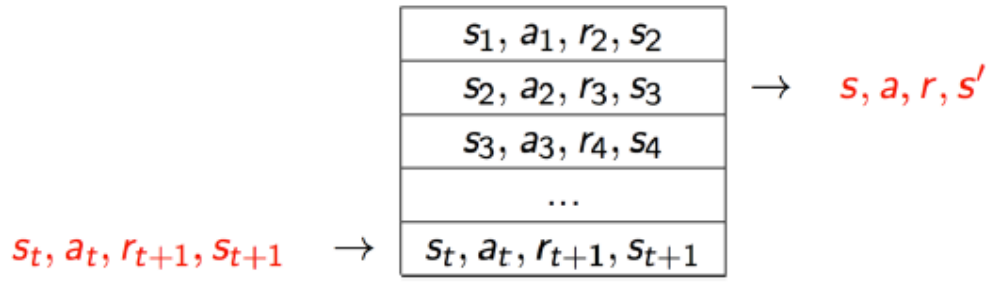


Image: Christopher Mutschler, *Reinforcement Learning, 2025*

DQN: How it works

- How does it work?
- A **bag of tricks** for stabilizing learning:
  1. Experience Replay:
     - **Problem:** Experiences are correlated over time.
       → *Oscillations* and *divergence* during learning.
     - **Solution:** Random sampling of experience mini-batches from a memory.
       → Samples can be re-used to *increase data efficiency*.
       → Breaking correlations by randomization *reduces variance*.



http://www0.cs.ucl.ac.uk/staff/d.silver/web/Resources_files/deep_rl.pdf

Image: Christopher Mutschler, *Reinforcement Learning, 2025*

DQN: How it works

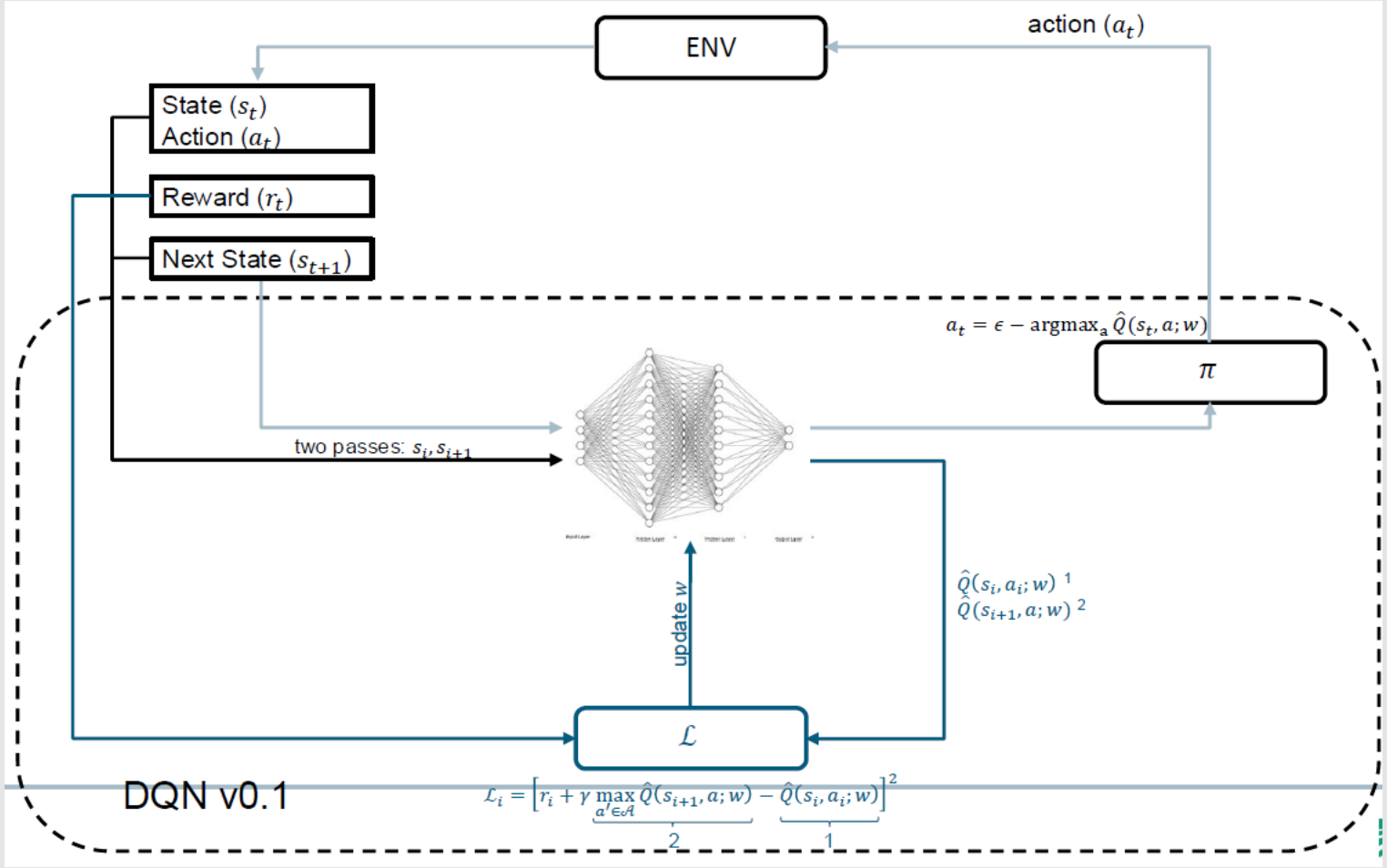


Image: Christopher Mutschler, *Reinforcement Learning, 2025*

DQN: How it works

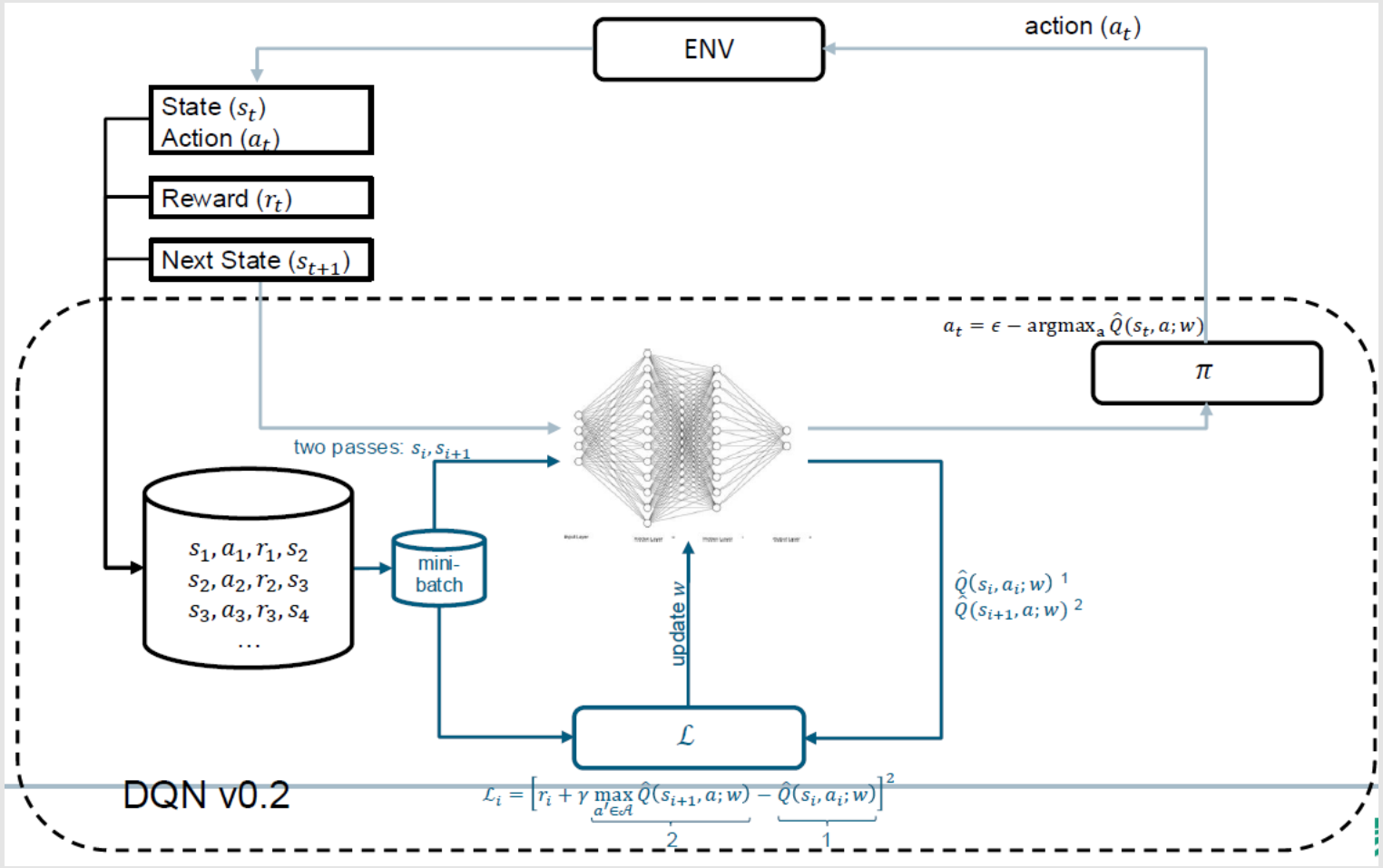


Image: Christopher Mutschler, *Reinforcement Learning, 2025*

DQN: How it works

- How does it work?
- A **bag of tricks** for stabilizing learning:
  1. Experience Replay.
  2. Separate, frozen target Q-network:
     - **Problem:** Target Q-values $y_i = r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s', a'; w_{i-1})$ change constantly.

       → *Oscillations* and *divergence* during learning.

Image: Christopher Mutschler, *Reinforcement Learning, 2025*

DQN: How it works

- How does it work?
- A **bag of tricks** for stabilizing learning:
  1. Experience Replay.
  2. Separate, frozen target Q-network:
     - **Solution:** Two Q-networks:
       → Frozen Target Q-network with parameters w^- predicts Q-learning targets $\hat{Q}(s', a'; w_i^-)$.
       → Dynamic Main Q-network with parameters w evaluates current Q-values $\hat{Q}(s', a'; w_{i+1})$
       → Perform a gradient descent step (w.r.t. w) towards $\left(y_i - \hat{Q}(s, a; w_i)\right)^2$
     - Added delay breaks correlations between Q-network and target.
     - *Avoids oscillations* by having fixed targets.
       (Note: We periodically update the target Q-network by copying the weights $w^- \leftarrow w_i$.)
     - *Reduces* chance of *divergence*.

Image: Christopher Mutschler, *Reinforcement Learning, 2025*

DQN: How it works

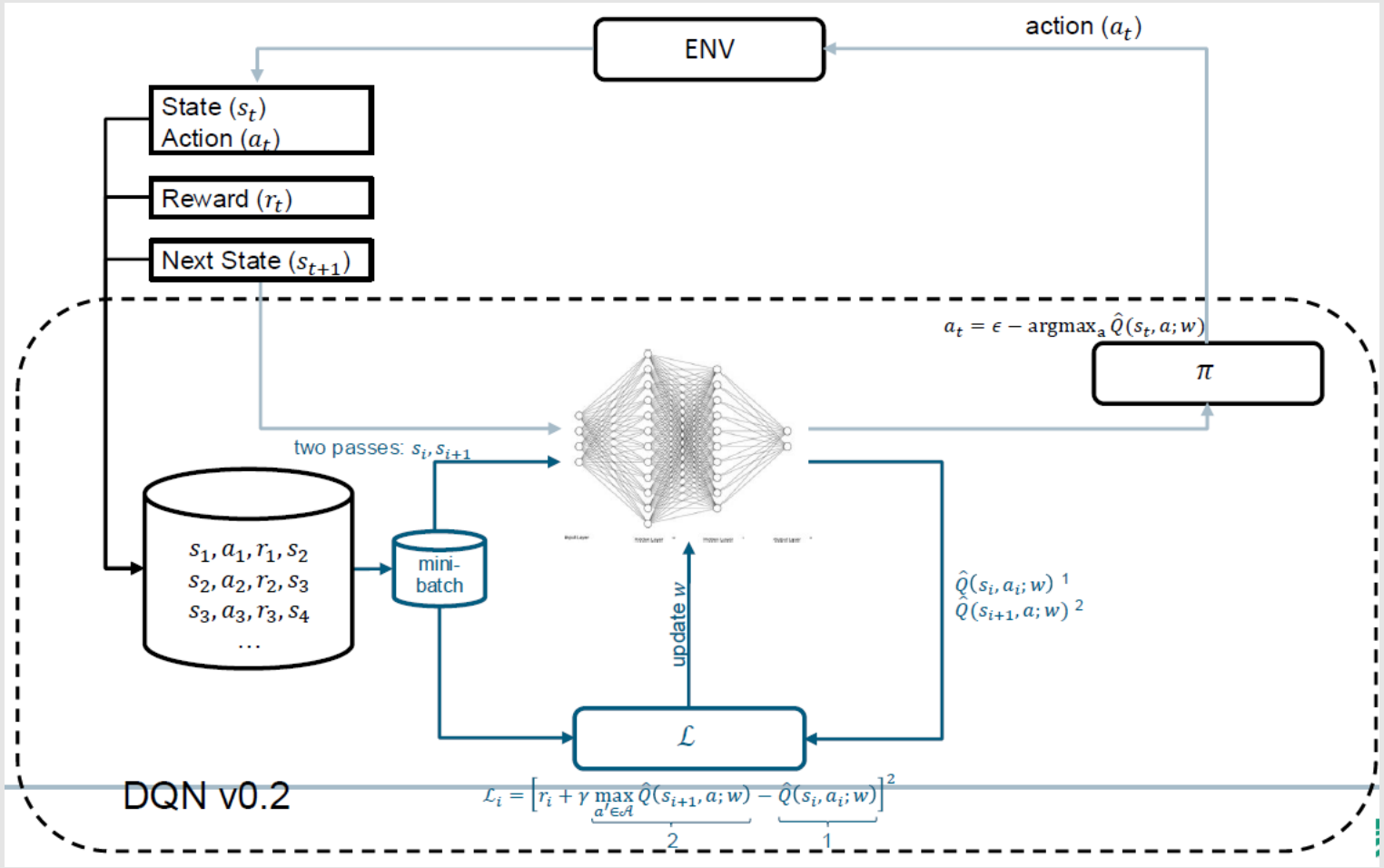


Image: Christopher Mutschler, *Reinforcement Learning, 2025*

DQN: How it works

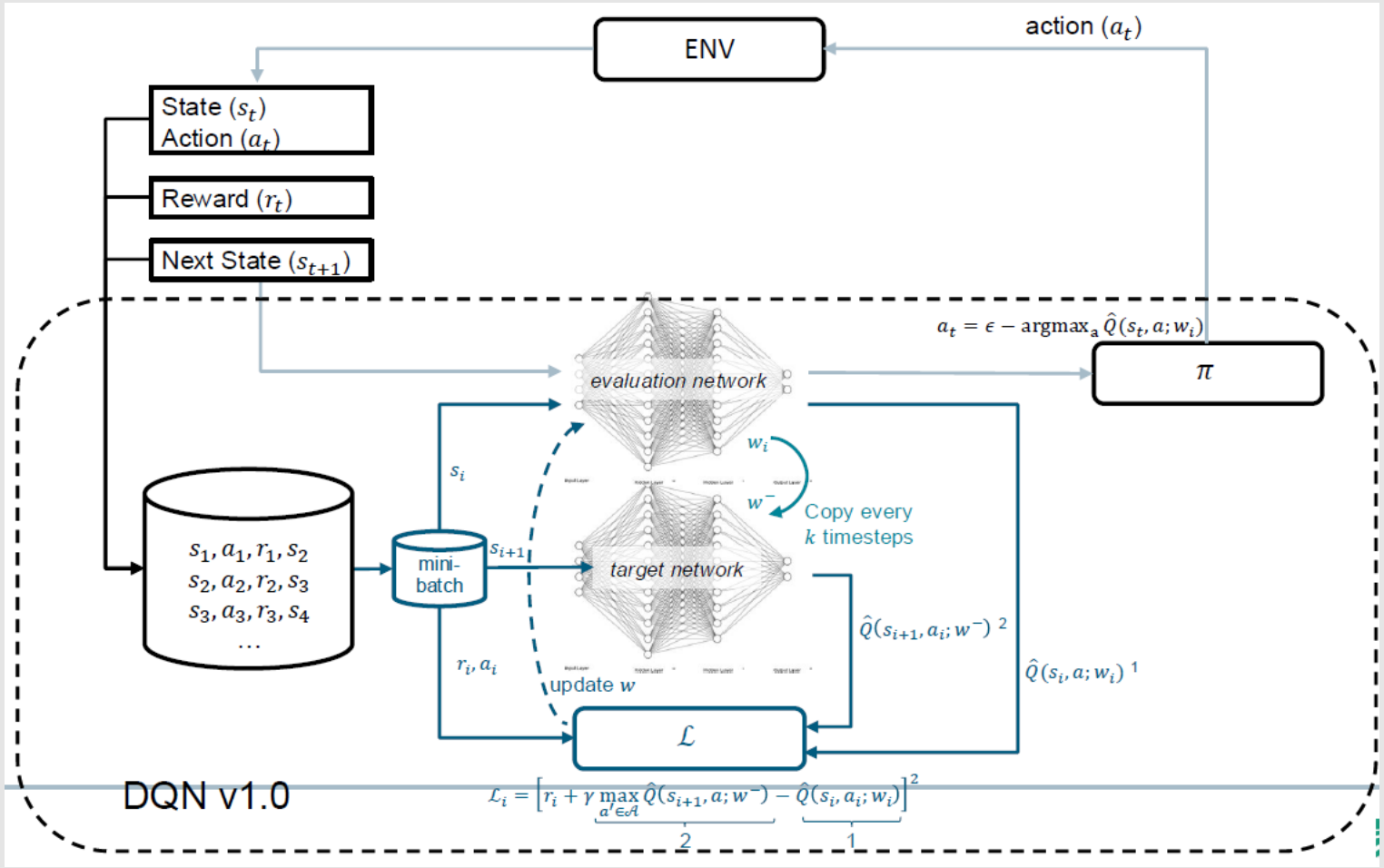


Image: Christopher Mutschler, *Reinforcement Learning, 2025*

DQN: How it works

- How does it work?
- A **bag of tricks** for stabilizing learning:
  1. Experience Replay.
  2. Separate, frozen target Q-network.
  3. Apply reward clipping:
     - **Problem:** Large rewards result in large variances in Q-values.
     - Different games have different reward values.
     - *Oscillations* and *divergence* during learing.

     - **Solution:** Clip the rewards (and loss terms) to a range $[-1.0, 1.0]$.
     - Avoids *oscillations* by normalizing rewards when training for multiple games.
     - Prevents Q-values from becoming too large.

Image: Christopher Mutschler, *Reinforcement Learning, 2025*

Extensions: Double DQN

- DQN overestimates Q-values due to using the same network for action selection and evaluation.

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)$$

- Double DQN separates these roles:

$$y_t^{\text{Double}} = r_t + \gamma Q(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'; \theta), \theta^-)$$

- Online network (θ) selects; target network (θ^-) evaluates.
- Reduces bias, giving more stable and accurate estimates.

To learn more:

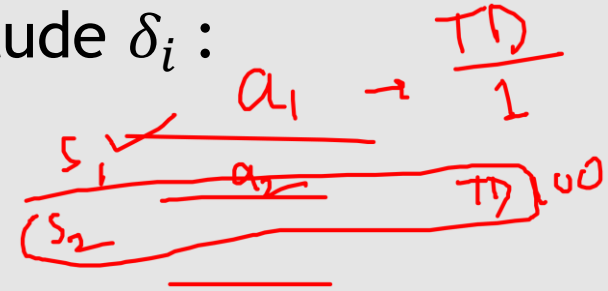
Van Hasselt, Hado, Arthur Guez, and David Silver. "***Deep reinforcement learning with double q-learning.***" Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.

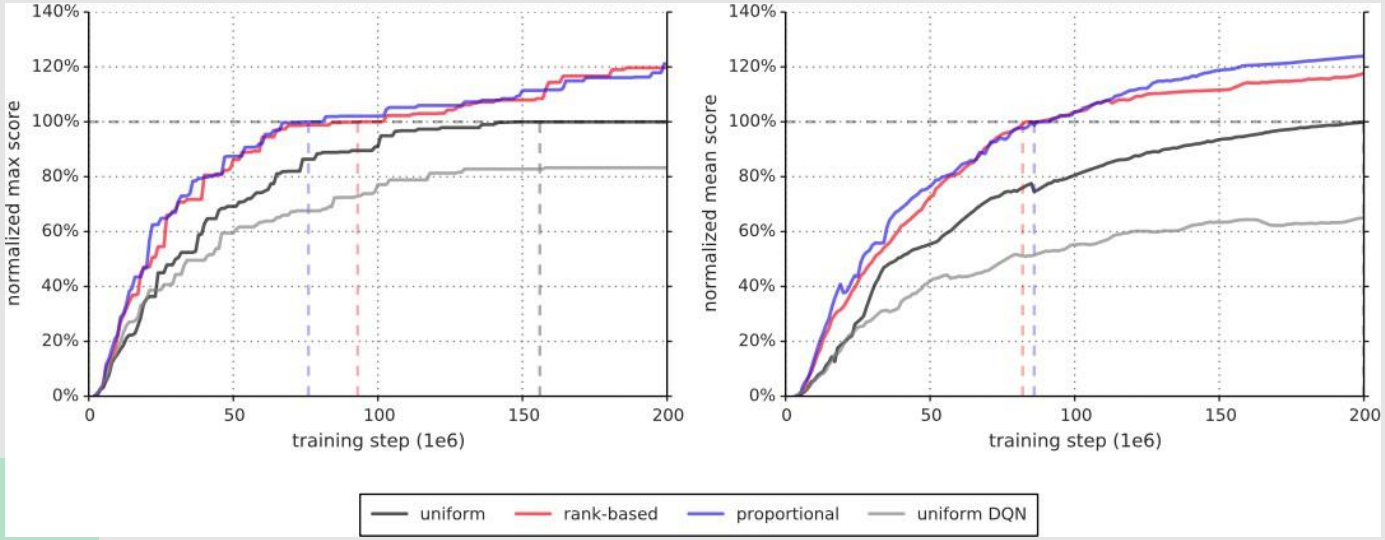
Extensions: Prioritized Experience Replay

- Uniform replay wastes updates on uninformative samples.
- Prioritized Replay samples experiences by TD error magnitude δ_i :

$$P(i) \propto | \delta_i |^{\alpha}$$

- Focuses on transitions the agent predicts poorly.
- Importance sampling corrects bias → faster, smarter learning.





To learn more:

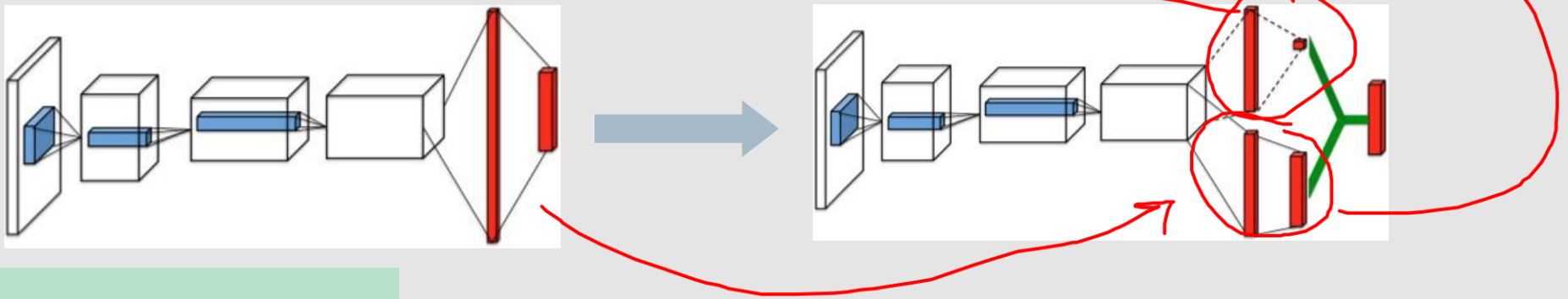
Schaul, Tom, et al. "***Prioritized experience replay***." arXiv preprint arXiv:1511.05952 (2015).

Extensions: Dueling DQN

- DQN learns Q-values for all actions — even when most don't matter.
- Dueling DQN splits Q into value and advantage streams:

$$Q(s,a) = V(s) + (A(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a'))$$

- Learns which states are valuable independently of specific actions.
- Improves generalization and stability.

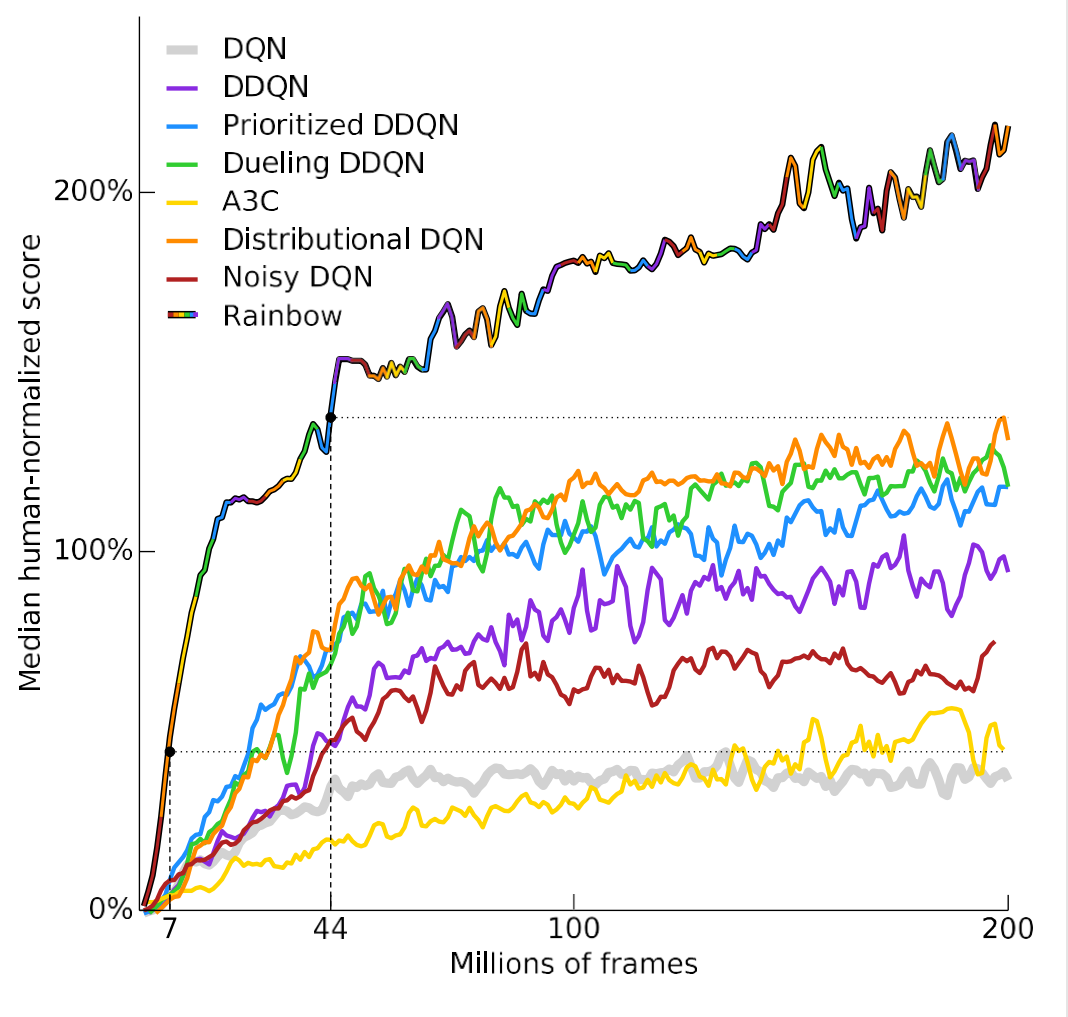


To learn more:

Wang, Ziyu, et al. "***Dueling network architectures for deep reinforcement learning.***" International conference on machine learning. PMLR, 2016.

Extensions: Rainbow DQN

- Combines the best ideas from earlier variants:
  1. Double DQN – reduces overestimation
  2. Prioritized Replay – smarter sampling
  3. Dueling Net – efficient value learning
  4. Multi-step Returns – faster credit assignment
  5. Noisy Nets – exploration via parameter noise
  6. Distributional RL – models return distribution
- Unified, high-performing DQN achieving state-of-the-art results.



To learn more:

Hessel, Matteo, et al. "***Rainbow: Combining improvements in deep reinforcement learning.***" Proceedings of the AAAI conference on artificial intelligence. Vol. 32. No. 1. 2018.

Summary

- Tabular methods don't scale. State spaces are too large, generalization is impossible.

- Function approximation, especially with deep learning, enables RL to tackle real-world problems with complex, high-dimensional state spaces.