Fraunhofer IIS

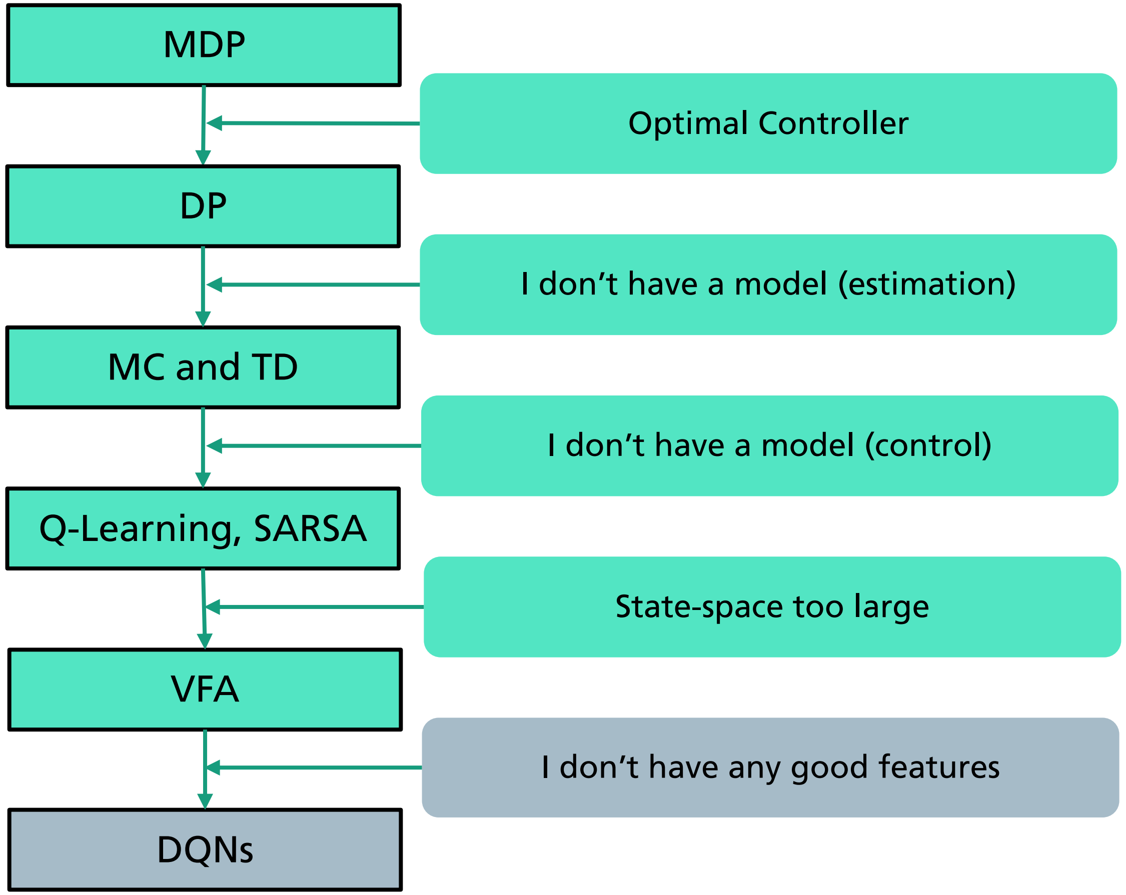
Fraunhofer-Institut für Integrierte
Schaltungen IIS

Reinforcement Learning

Lecture 5: Value-function Approximation

Christopher Mutschler

Overview



MDP

Optimal Controller

DP

I don’t have a model (estimation)

MC and TD

I don’t have a model (control)

Q-Learning, SARSA

State-space too large

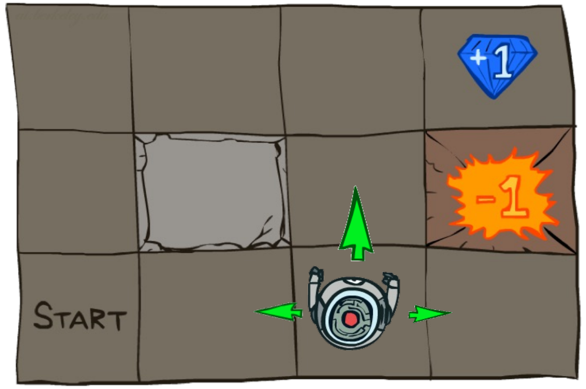
VFA

I don’t have any good features

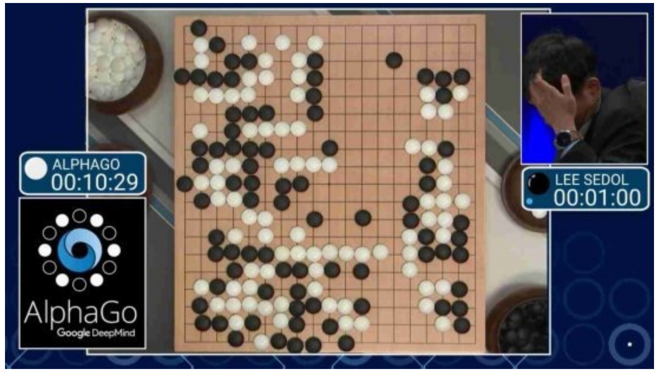
DQNs

Value Function Approximation

- Challenge #1: In real world problems, the state space can be large
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Robot arm: **infinite** number of states! (continuous)
- Problems with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually



http://ai.berkeley.edu/lecture_slides.html

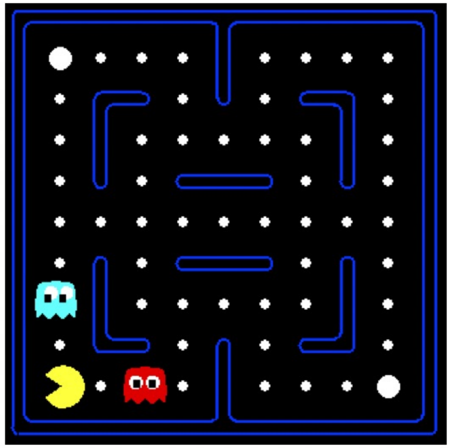


https://www.youtube.com/watch?v=HT-UZkiOLv8

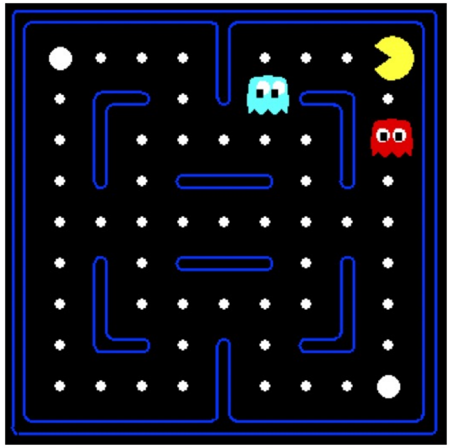
Value Function Approximation

- Challenge #2: Generalization across states

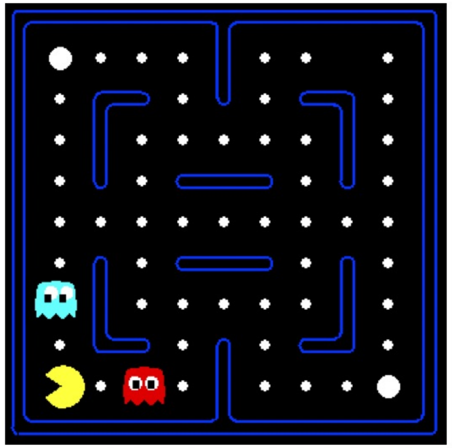
Let‘s say we discover through experience that this state is bad:



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



Or even this one:



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

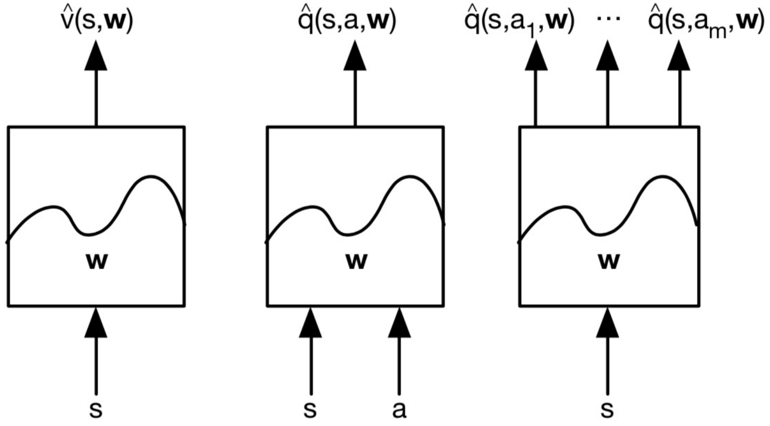
Value Function Approximation

- Value Function Representations
- Exact:
  - A table with a distinct value for each case
  - V: one entry per s
  - Q: one entry for each (s, a) pair
- Approximate:
  - Approximate V or Q with a function approximator (e.g., NN, polynomials, RBF, …)

$$\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$$
$$\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$$

+ **We only need to store the approximator parameters**

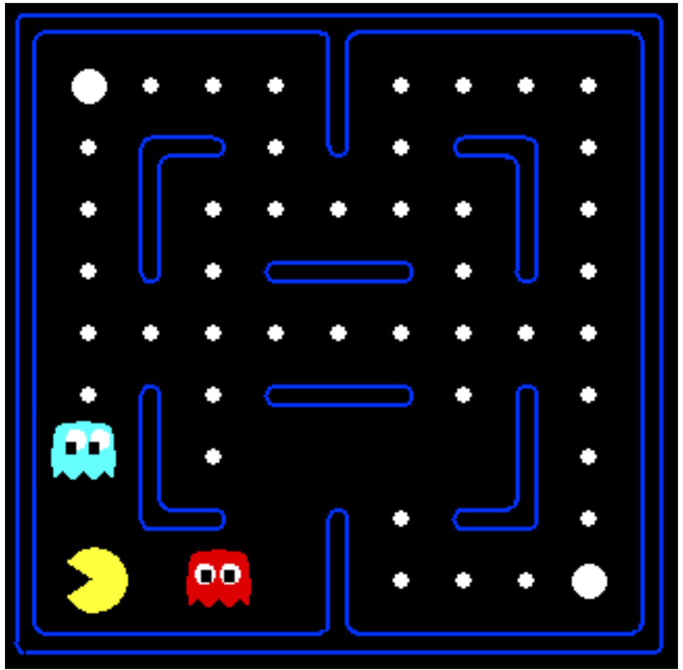
- **Convergence properties do not hold anymore**



David Silver. 2016.

Value Function Approximation

- VFA: Describe a state using a vector 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
  - …



Value Function Approximation

- Our goal is to learn good parameters w that approximate the true value function well:

$$C = \left(Q^{+}(s,a) - \hat{Q}^{\pi}(s,a;w)\right)^{2}$$
$$= (Q^{+}(s,a) - \ \phi(s,a)^{T}w)^{2}$$

$$\frac{\partial C}{\partial w} = -2\ \phi(s,a)(Q^{+}(s,a) - \phi(s,a)^{T}w)$$

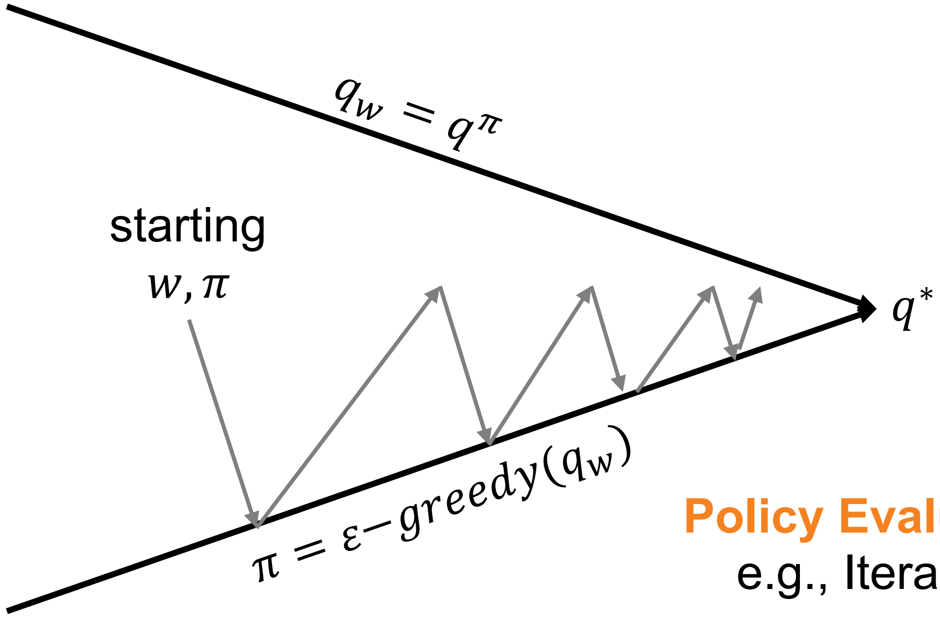
$$w \leftarrow w\ - \eta \frac{\partial C}{\partial w}$$
$$w \leftarrow w + 2\ \eta\ \phi(s,a)\left(Q^{+}(s,a)\ - \hat{Q}^{\pi}(s,a;w)\right)$$

$Q^{+}(s,a) =$

$r + \ \gamma\ max_{a'}Q(s',a')$	Q-Learning with Linear VFA
$r + \ \gamma\ Q(s',a')$	SARSA with Linear VFA
G_t	MC with Linear VFA

Value Function Approximation

- Our goal is to learn good parameters w that approximate the true value function well:

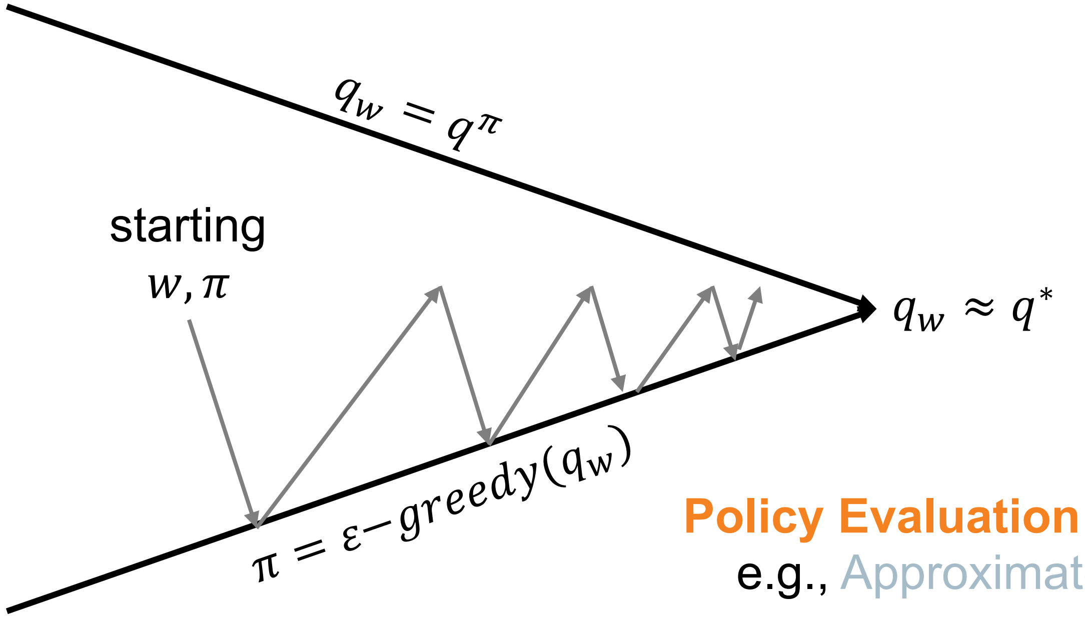


Policy Evaluation: Estimate q_π
e.g., Iterative Policy Evaluation

Policy Improvement: Generate $\pi' \geq \pi$
e.g., ϵ-greedy Policy Improvement

Value Function Approximation

- Our goal is to learn good parameters w that approximate the true value function well:



Policy Evaluation: Estimate $\hat{\boldsymbol{q}}(\cdot,\cdot,\boldsymbol{w}) \approx q_\pi$
e.g., Approximate Policy Evaluation

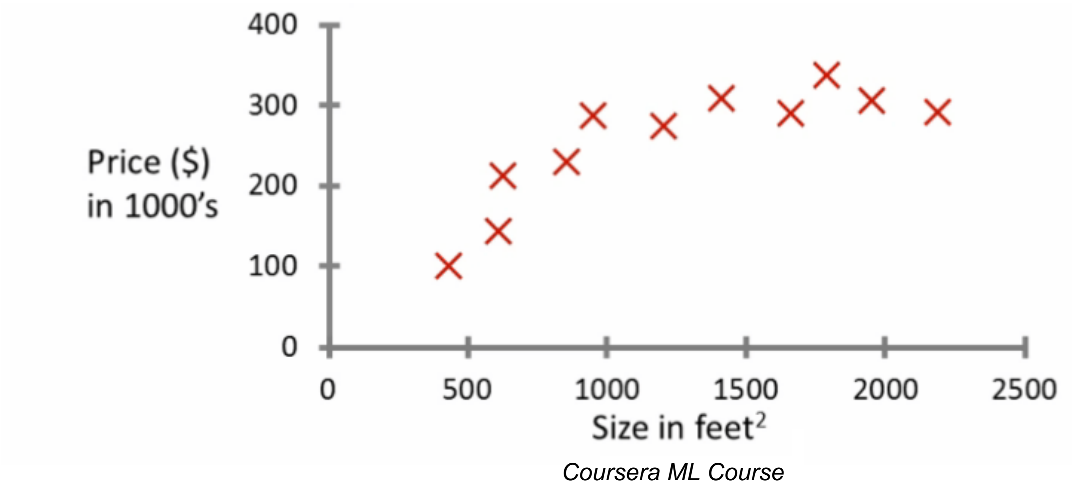
Policy Improvement: Generate $\pi' \geq \pi$
e.g., ϵ-greedy Policy Improvement

Value Function Approximation

- Linear Value Function Approximation (careful: non-linear features)

$$\hat{Q}^{\pi}(s, a; w) = \phi(s, a)^T w$$

Housing Price Prediction



Coursera ML Course

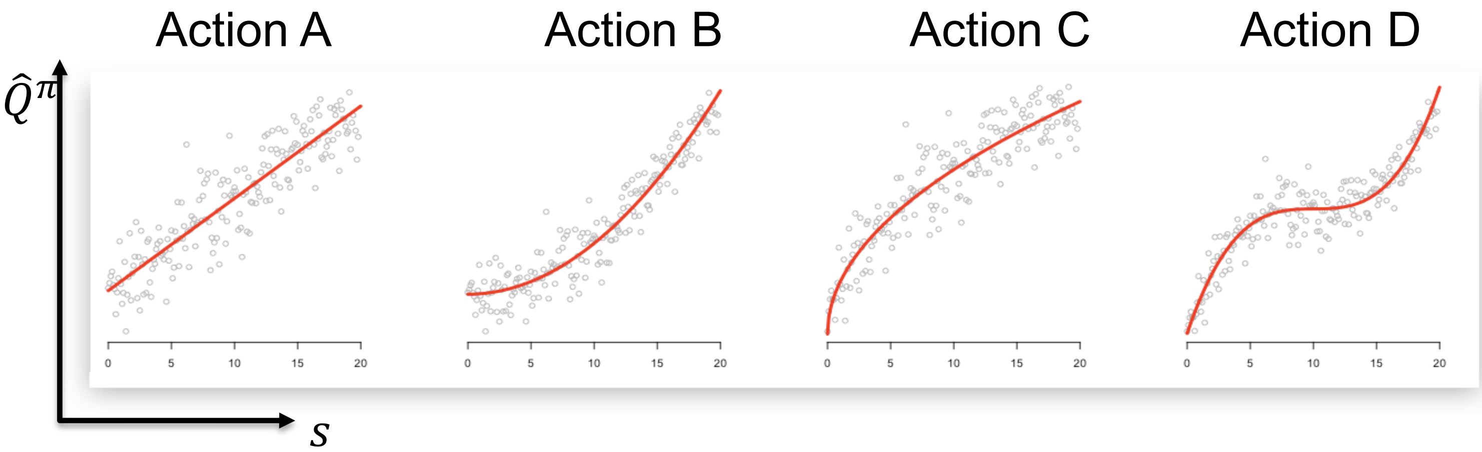
Value Function Approximation

- Linear Value Function Approximation (careful: non-linear features)

$$\hat{Q}^{\pi}(s,a;w) = \phi(s,a)^T w$$

- Example features: Polynomial Basis

$$\phi_i(s,a_l) = \prod_{j=1}^{k} s_j^{c_{i,j}}, \qquad c_{i,j} \in \{0,1,\dots,n\}, a_l \in \mathcal{A}$$



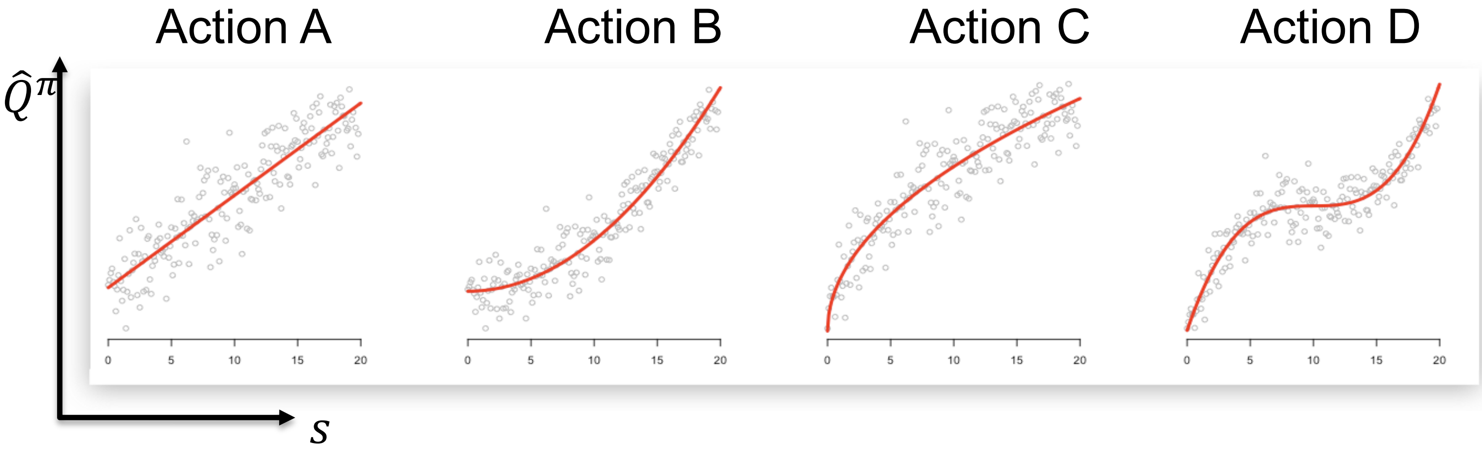
Value Function Approximation

- Linear Value Function Approximation (careful: non-linear features)

$$\hat{Q}^{\pi}(s,a;w) = \phi(s,a)^T w$$

- Example features: Polynomial Basis, for instance:

$$(s_1,s_2)^T \rightarrow (1,s_1,s_2,s_1s_2)^T$$
$$(s_1,s_2)^T \rightarrow (1,s_1,s_2\,,s_1s_2,s_1^2,s_2^2,s_1s_2^2,s_1^2s_2,s_1^2s_2^2)$$



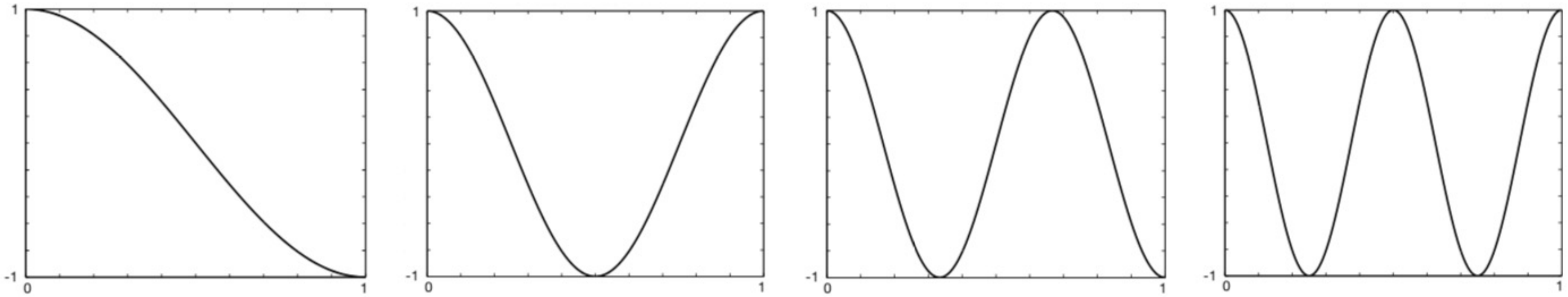
Value Function Approximation

- Linear Value Function Approximation (careful: non-linear features)

$$\hat{Q}^{\pi}(s, a; w) = \phi(s, a)^T w$$

- Example features: Fourier Basis

$$\phi_i(s, a_j) = \cos(i\pi s), \qquad s \in [0,1], a_j \in \mathcal{A}$$

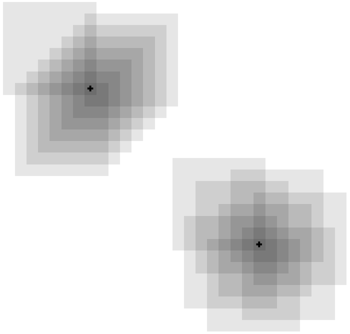


Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

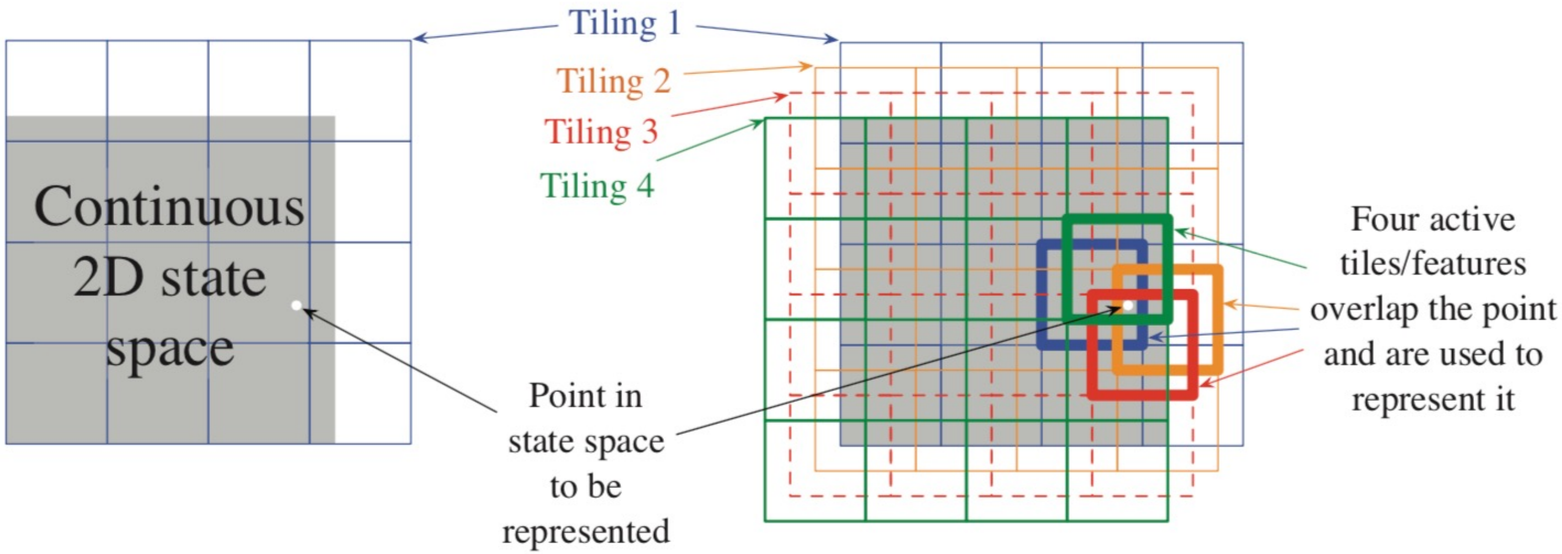
Value Function Approximation

- Linear Value Function Approximation (careful: non-linear features)

$$\hat{Q}^{\pi}(s, a; w) = \phi(s, a)^T w$$



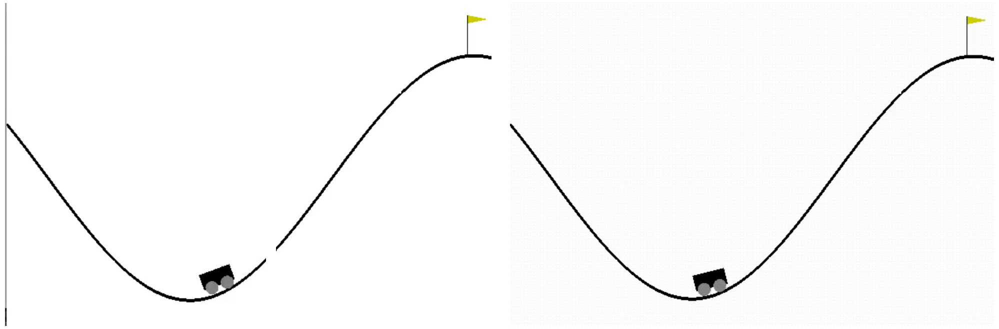
- Example features: Tile Coding

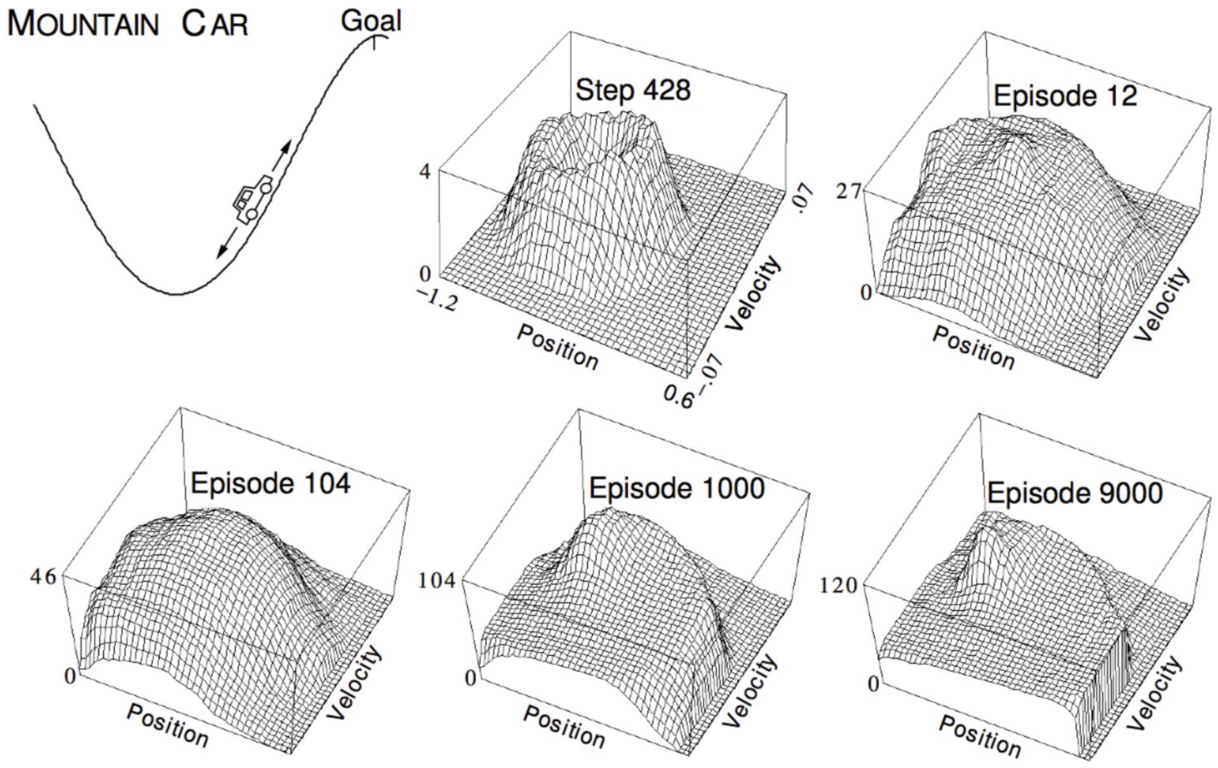

Tiling 1
Tiling 2
Tiling 3
Tiling 4
Continuous 2D state space
Point in state space to be represented
Four active tiles/features overlap the point and are used to represent it

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Value Function Approximation

- Linear Sarsa with Coarse Coding[1] in Mountain Car





David Silver. 2016.

[1] *„Tile Coding with circles“*

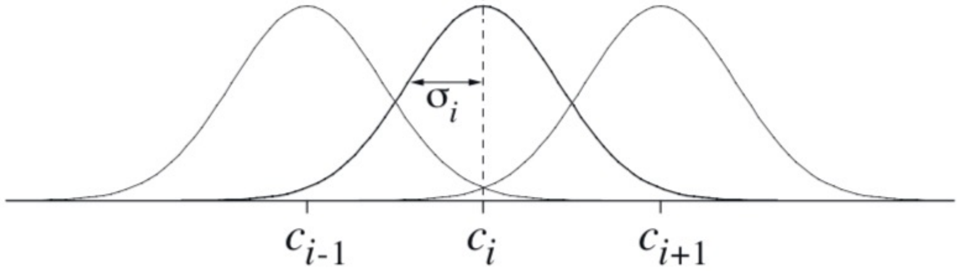
Value Function Approximation

- Linear Value Function Approximation (careful: non-linear features)

$$\hat{Q}^{\pi}(s,a;w) = \phi(s,a)^T w$$

- Example features: Radial Basis Functions (RBFs)

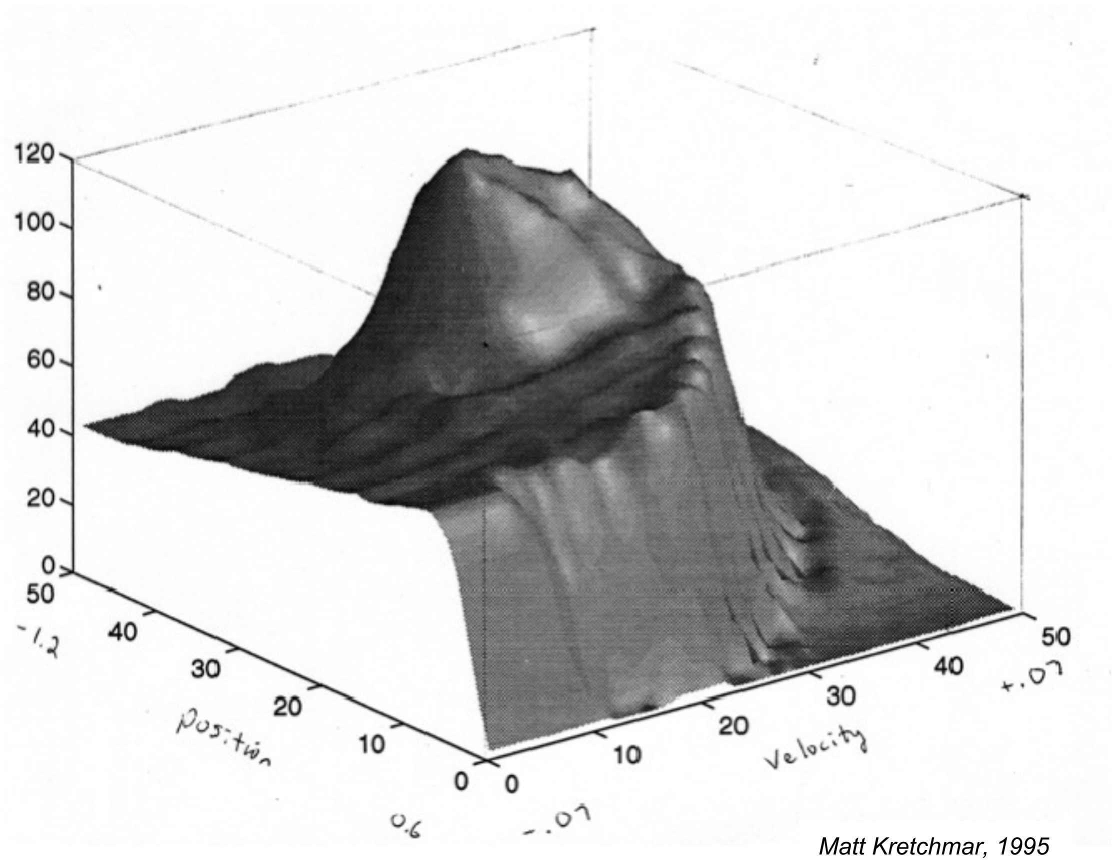
$$\phi_i(s,a_j) \doteq \exp\left(-\frac{\|s-c_i\|^2}{2\sigma_i^2}\right), \qquad a_j \in \mathcal{A}$$



Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Value Function Approximation

- Linear Sarsa with Radial Basis Functions in Mountain Car



Matt Kretchmar, 1995

Value Function Approximation

- Why Linear VFA?
- Convergence of Control Algorithms

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

(✓) = chatters around near-optimal value function

Value Function Approximation (VFA)

- There are two important questions to answer with VFA:
  1. Can we approximate any V-/Q-value function with a linear FA?
  2. Is it easy to find such a linear FA?

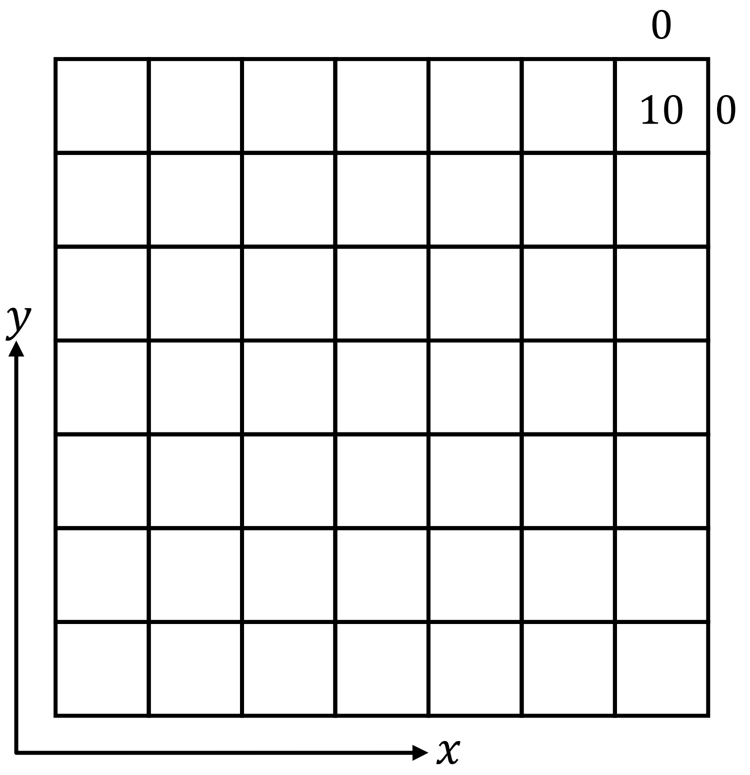
Value Function Approximation (VFA)

- There are two important questions to answer with VFA:
  1. Can we approximate any V-/Q-value function with a linear FA?
     → **YES!** (But the proof is out of the scope of this class…)

Value Function Approximation (VFA)

- There are two important questions to answer with VFA:
  1. Can we approximate any V-/Q-value function with a linear FA?
  2. Is it easy to find such a linear FA? → **YES!**

- Example Gridworld problem:
  - No obstacles,
  - deterministic actions (UDLR)
  - no discounting
  - reward is -1 everywhere except $+10$ at goal.



Value Function Approximation (VFA)

- There are two important questions to answer with VFA:
  1. Can we approximate any V-/Q-value function with a linear FA?
  2. Is it easy to find such a linear FA? → **YES!**

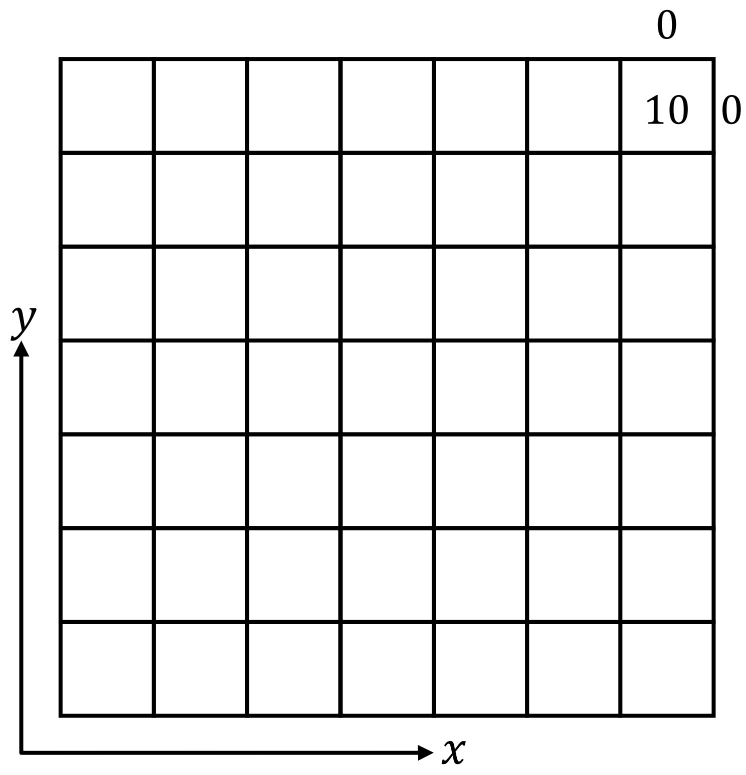
- Represent state s by a feature vector:

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

- Perform linear VFA:

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

$$\hat{V}(s;w) = w_0 + w_1 x + w_2 y$$



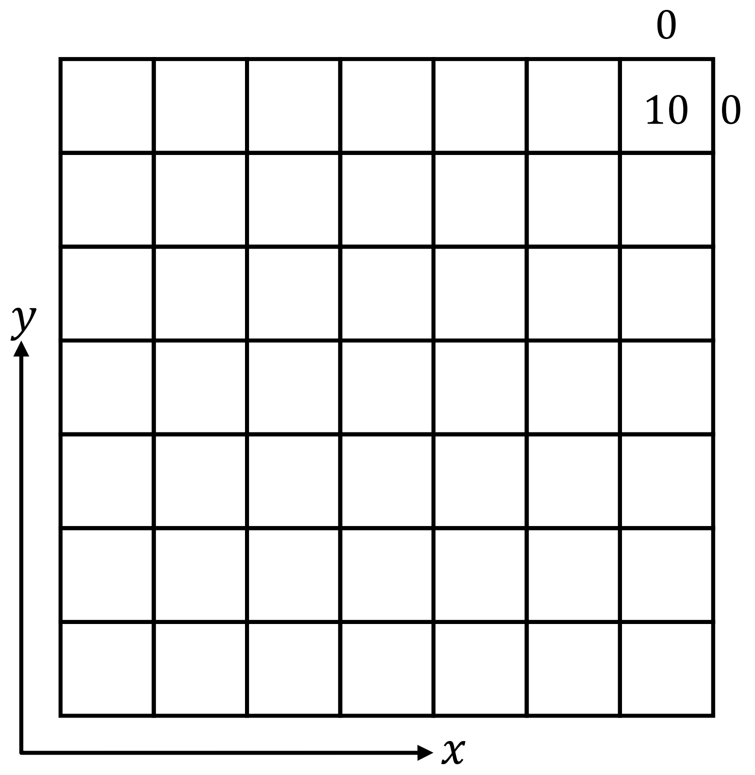
Value Function Approximation (VFA)

- There are two important questions to answer with VFA:
  1. Can we approximate any V-/Q-value function with a linear FA?
  2. Is it easy to find such a linear FA? → **YES!**

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

$$\begin{aligned}\hat{V}(s;w) &= \begin{bmatrix}1 & x & y\end{bmatrix}\begin{bmatrix}10\\-1\\-1\end{bmatrix}\\ &= 10 - x - y = 10 - |x + y|\end{aligned}$$

- Note: Manhattan Distance.



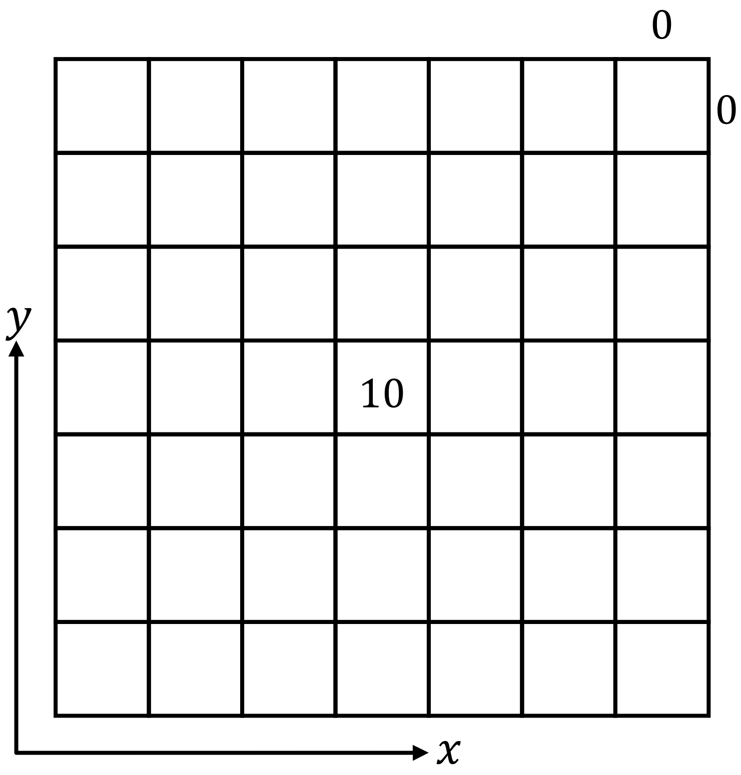
Value Function Approximation (VFA)

- There are two important questions to answer with VFA:
  1. Can we approximate any V-/Q-value function with a linear FA?
  2. Is it easy to find such a linear FA? → **YES!**

- What if the reward changes (see Fig.)?
- Linear VFA:

$$\begin{aligned}\hat{V}(s;w) &= \phi(s)^T w \\ &= \begin{bmatrix}1 & x & y\end{bmatrix}\begin{bmatrix}w_0 \\ w_1 \\ w_2\end{bmatrix} \\ &= w_0 + w_1 x + w_2 y\end{aligned}$$

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



Value Function Approximation (VFA)

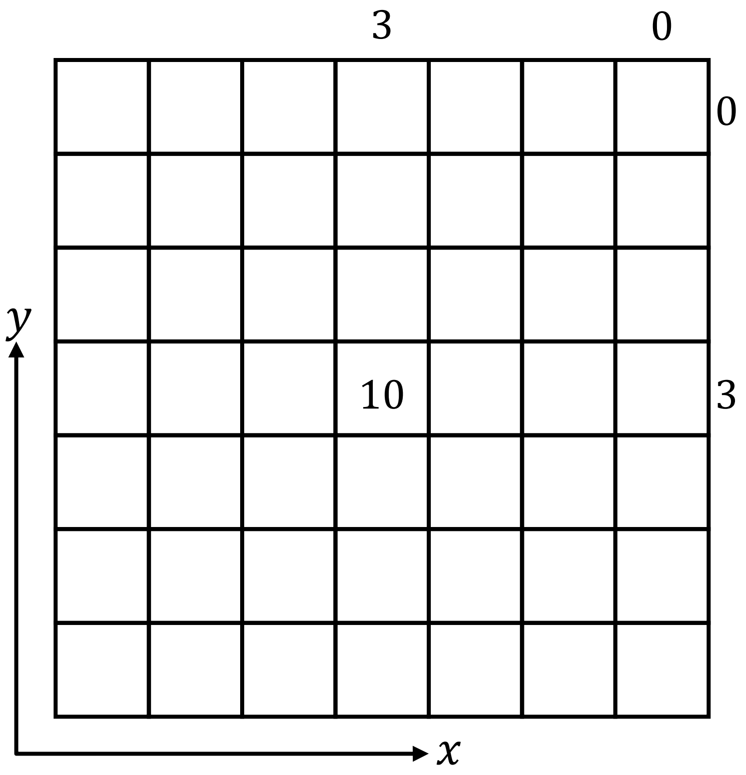
- There are two important questions to answer with VFA:
  1. Can we approximate any V-/Q-value function with a linear FA?
  2. Is it easy to find such a linear FA? → **YES!**

- Linear VFA with a new feature $\boldsymbol{z}$:

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

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

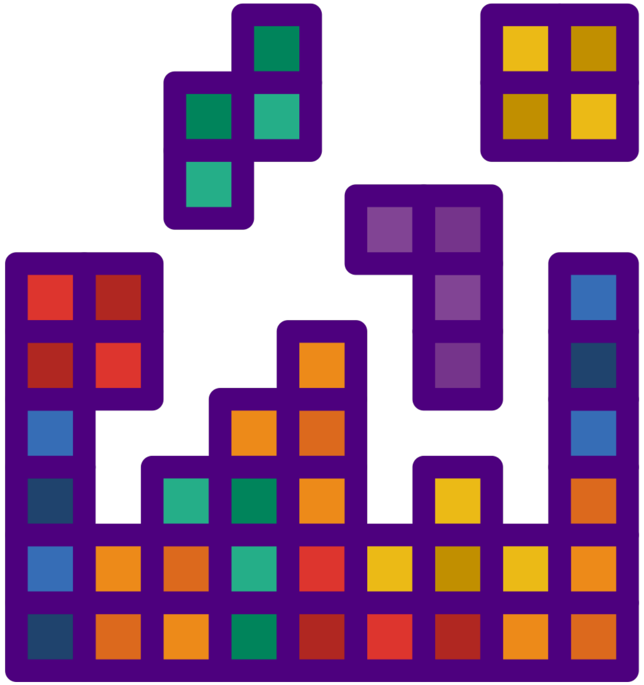
$$z = |3 - x| + |3 - y|$$



Value Function Approximation (VFA)

- There are two important questions to answer with VFA:
  1. Can we approximate any V-/Q-value function with a linear FA?
  2. Is it easy to find such a linear FA? → **NO!**

Feature	Description
Landing Height	Height of last piece is added.
Eroded Piece Cells	#rows eliminated in the last move multiplied with the #bricks eliminated from the last piece added.
Row Transitions	#horizontal full to empty or empty to full transitions between the cells on the board.
Columns Transitions	Same thing for vertical transitions.
Holes	#empty cells covered by at least one full cell.



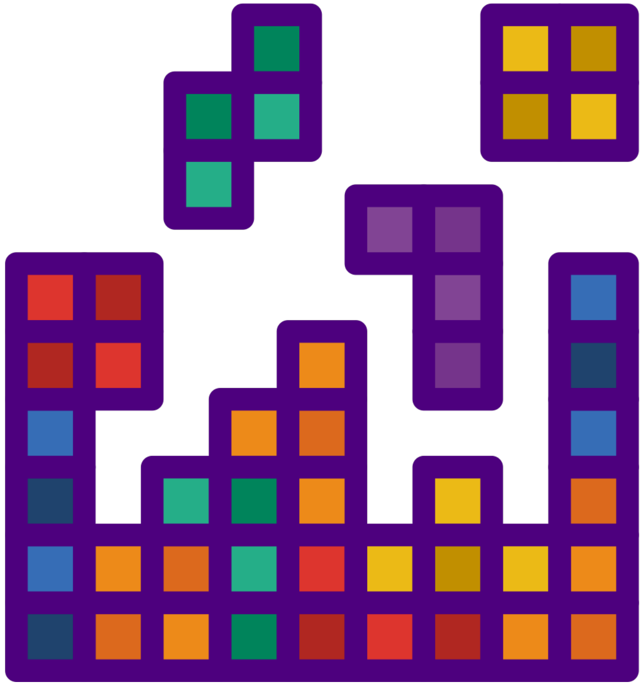
https://www.flaticon.com/free-icon/tetris_1006985

Thierry, C. and Scherrer, B.: Improvements on Learning Tetris with Cross-Entropy. 2010.

Value Function Approximation (VFA)

- There are two important questions to answer with VFA:
  1. Can we approximate any V-/Q-value function with a linear FA?
  2. Is it easy to find such a linear FA? → **NO!**

Feature	Description
Board Wells	$\sum_{w \in wells} (1 + 2 + \cdots + depth(w))$
Column Height	Height of the *p*th column of the board.
Column Difference	Absolute difference $\lvert h_p - h_{p+1} \rvert$ between adjacent columns.
Maximum Height	Maximum pile height: $\max_p h_p$.
Holes	Number of empty cells covered by at least one full cell.



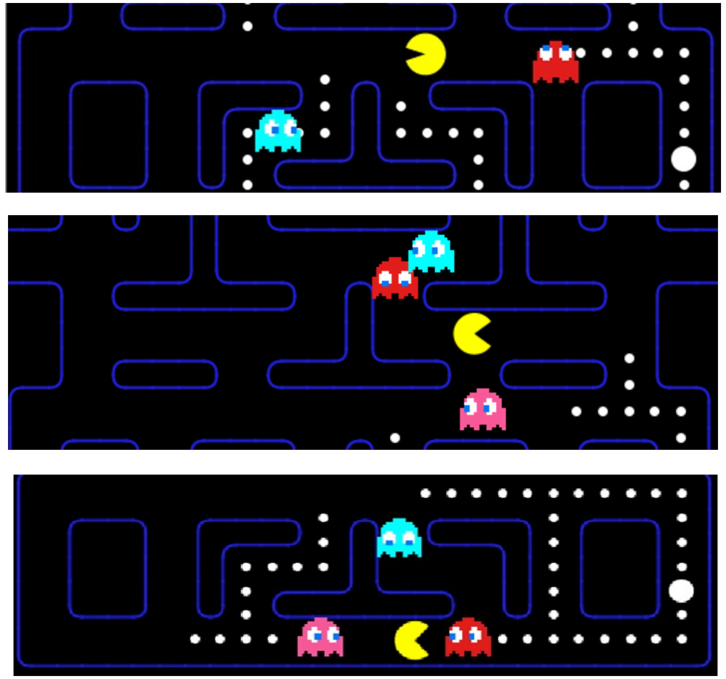
https://www.flaticon.com/free-icon/tetris_1006985

Thierry, C. and Scherrer, B.: Improvements on Learning Tetris with Cross-Entropy. 2010.

Value Function Approximation (VFA)

- There are two important questions to answer with VFA:
  1. Can we approximate any V-/Q-value function with a linear FA?
  2. Is it easy to find such a linear FA? → **NO!**

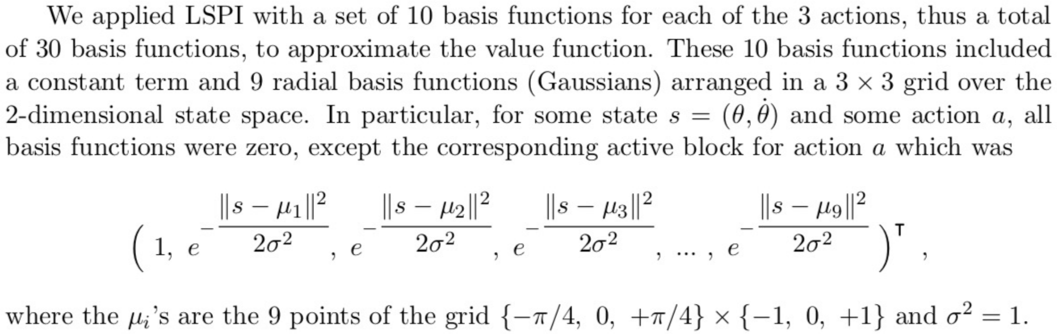
Feature	Description
Nearby Walls	Existence of wall to all four wind directions.
Nearest Target	Direction of the nearest *target* where it is preferable for the Ms. Pac-Man to move.
Nearby Ghosts	Existence of a ghost to all four wind directions.
Existance of Escape	Describes if Ms. Pac-Man can move freely or if she is trapped.



Tziortziotis, N. and Tziortziotis, K. and Blekas, K.: Play Ms. Pac-Man Using an Advanced Reinforcement Learning Agent. 2014.

Value Function Approximation (VFA)

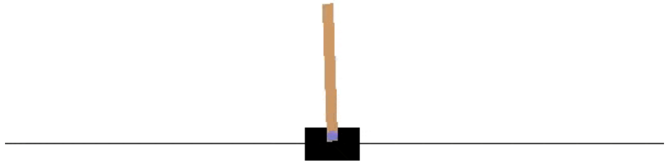
- There are two important questions to answer with VFA:
  1. Can we approximate any V-/Q-value function with a linear FA?
  2. Is it easy to find such a linear FA? → **NO!**



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

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

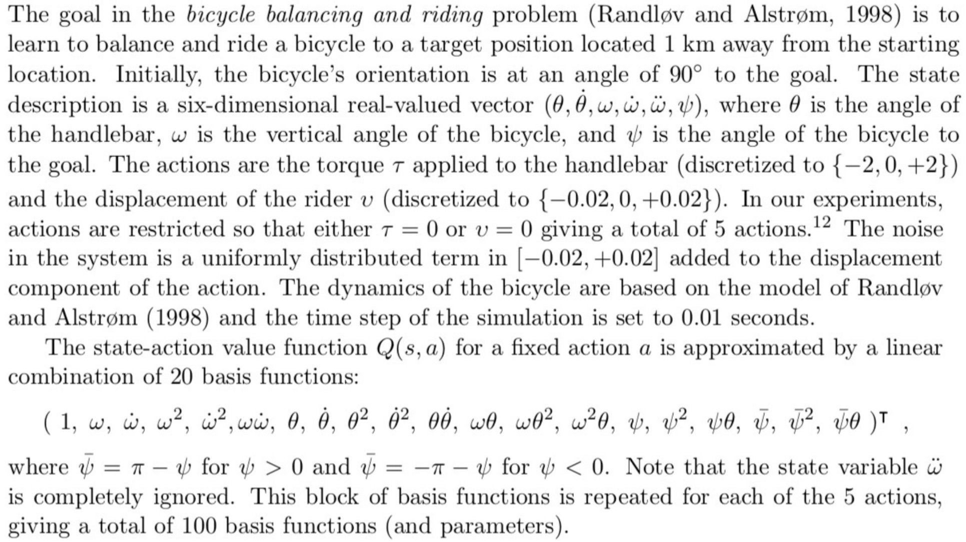
where the μ_i's are the 9 points of the grid $\{-\pi/4,\ 0,\ +\pi/4\} \times \{-1,\ 0,\ +1\}$ and $\sigma^2 = 1$.



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

Value Function Approximation (VFA)

- There are two important questions to answer with VFA:
  1. Can we approximate any V-/Q-value function with a linear FA?
  2. Is it easy to find such a linear FA? → **NO!**

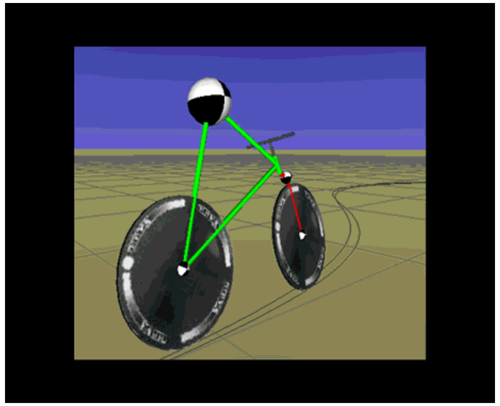


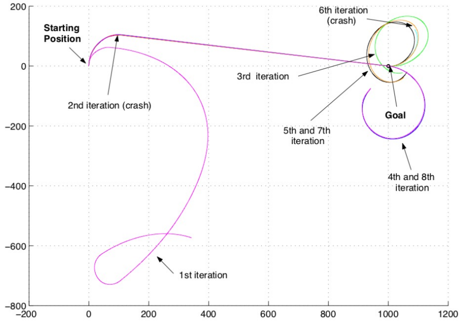
The goal in the *bicycle balancing and riding* problem (Randløv and Alstrøm, 1998) is to learn to balance and ride a bicycle to a target position located 1 km away from the starting location. Initially, the bicycle's orientation is at an angle of 90° to the goal. The state description is a six-dimensional 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:

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

where $\bar{\psi} = \pi - \psi$ 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

Value Function Approximation (VFA)

- Idea: Why don‘t we replace linear approximation with NNs?
  - Because theory tells us that this doesn‘t work out

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

(✓) =chatters around near-optimal value function

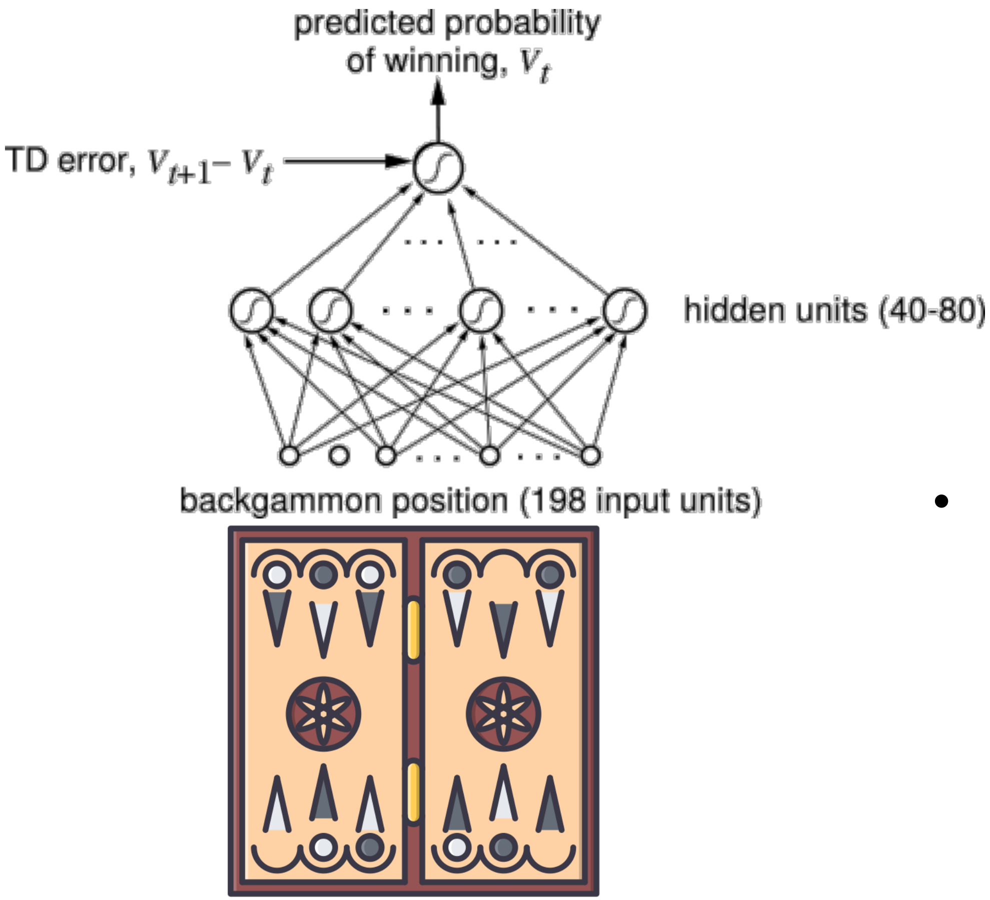
Value Function Approximation (VFA)

- Idea: Why don‘t we replace linear approximation with NNs?
  - Because theory tells us that this doesn‘t work out
  - But in some applications, it did ☺!
- World‘s Best Backgammon Player:
  - Neural network (NN) with 80 hidden units
  - Used RL for 300.000 games of self-play
  - One of the top players in the world!

https://www.flaticon.com/free-icon/backgammon_683899

Value Function Approximation (VFA)

Parenthesis: Gerry Tesauro‘s TD-Gammon (‘92, ’94, ’95)


predicted probability of winning, V_t
TD error, $V_{t+1} - V_t$
hidden units (40-80)
backgammon position (198 input units)

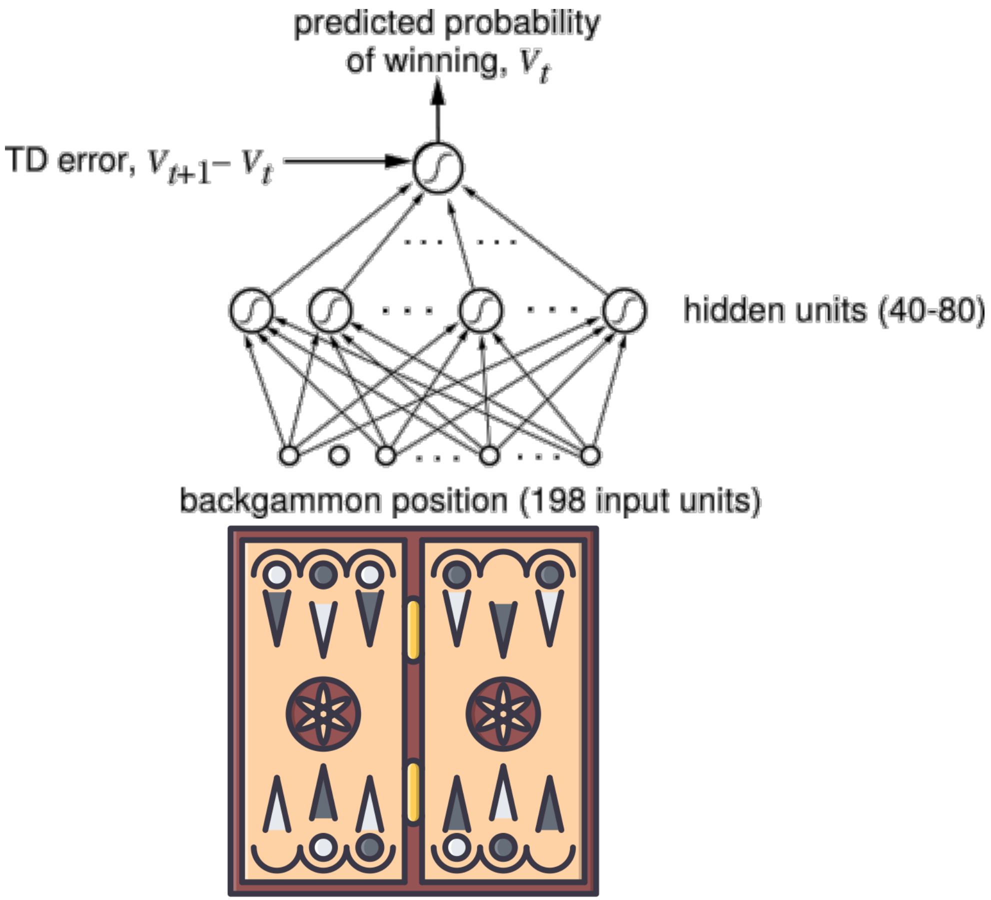
- Rule to update the network weights w:

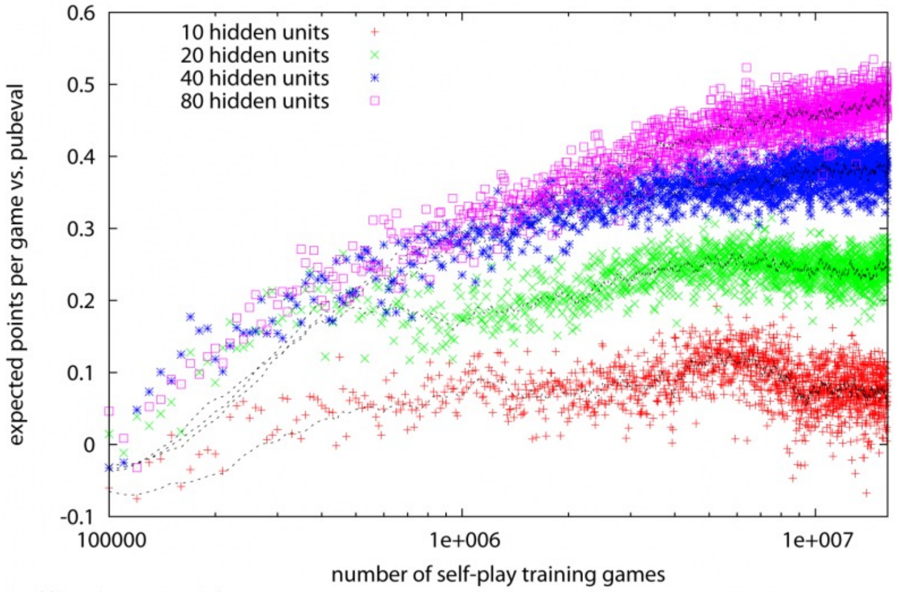
$$w \leftarrow w + \alpha(V_{t+1} - V_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w V_k$$

Tesauro, G.: Temporal difference learning and TD-Gammon. 1995.
see also: https://users.auth.gr/kehagiat/Research/GameTheory/12CombBiblio/BackGammon.html

Value Function Approximation (VFA)

Parenthesis: Gerry Tesauro's TD-Gammon ('92, '94, '95)


predicted probability of winning, V_t
TD error, $V_{t+1} - V_t$
hidden units (40-80)
backgammon position (198 input units)



Tesauro, G.: Temporal difference learning and TD-Gammon. 1995.
see also: https://users.auth.gr/kehagiat/Research/GameTheory/12CombBiblio/BackGammon.html

Value Function Approximation (VFA)

- Idea: Why don‘t we replace linear approximation with NNs?
  - Because theory tells us that this doesn‘t work out

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

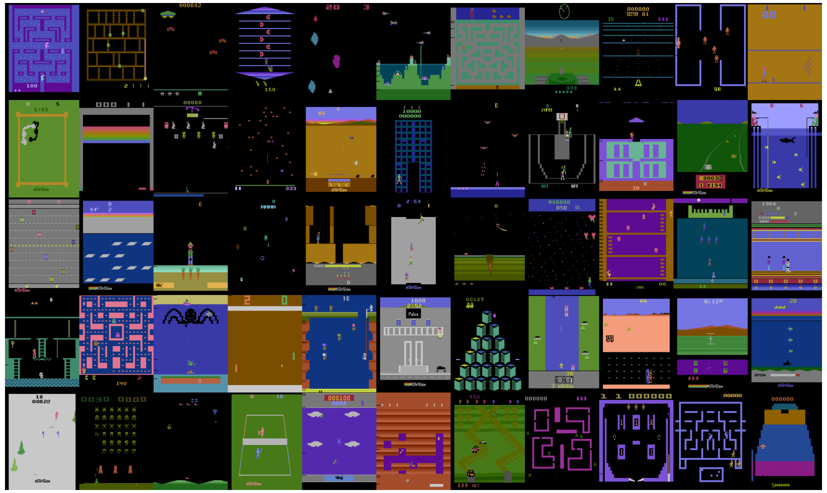
(✓) =chatters around near-optimal value function

- **Besides some few hand-crafted and tuned successes NNs have not been managed to be applied “as is” to RL**

Deep Q-Networks (DQNs)

- Then a game-changing result was published in Nature:
  DQN from DeepMind (now Google DeepMind)
  - Surpassed human player in 49 games of the Atari 2600 series
  - Same RL algorithm to learn a policy in each game
  - **End-to-end:** Only image pixels as input
  - “The“ contribution that initiated a round of huge investments in RL

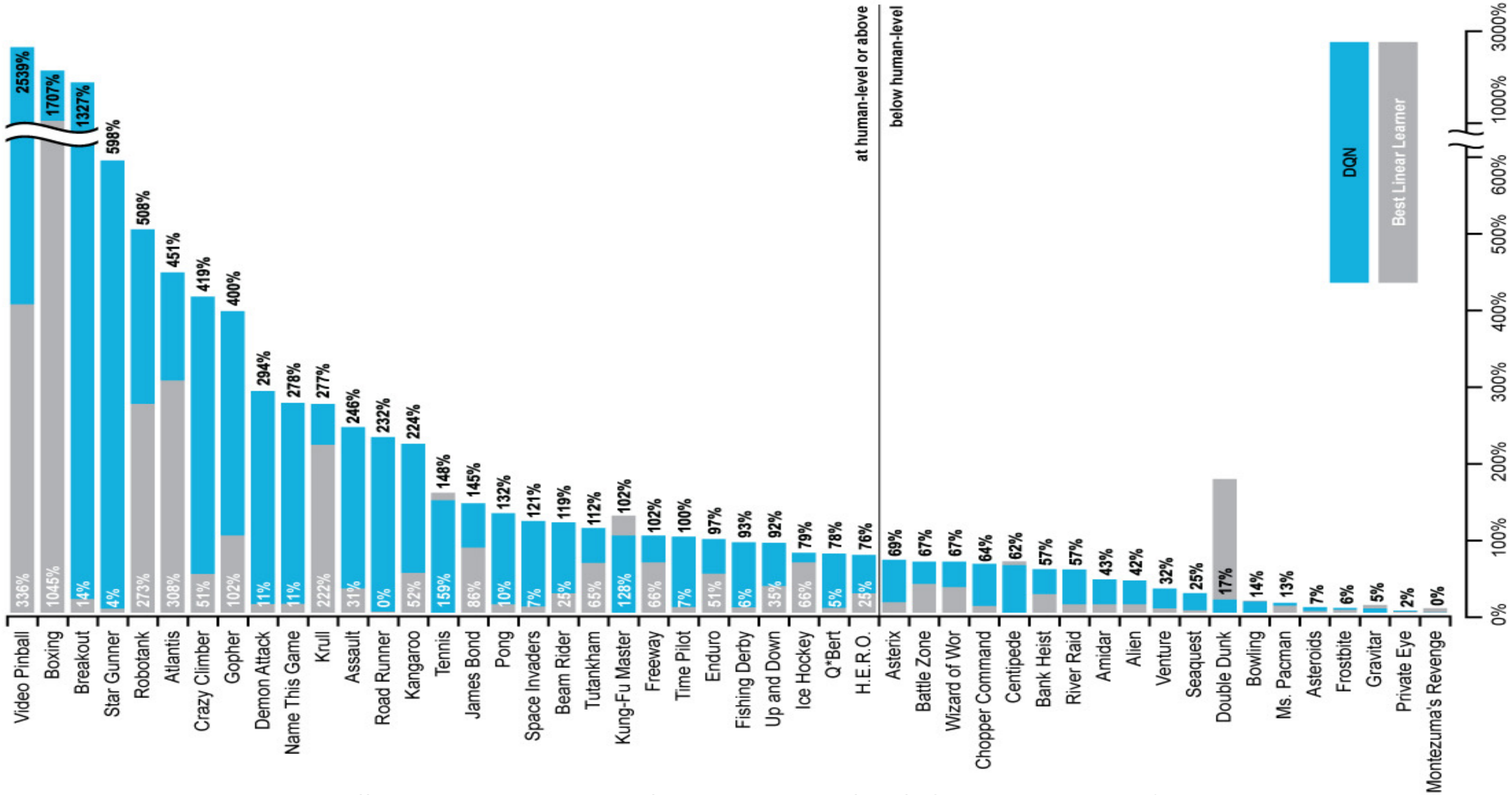
https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf



https://www.aarondefazio.com/adefazio-rl2014.pdf

Deep Q-Networks (DQNs)

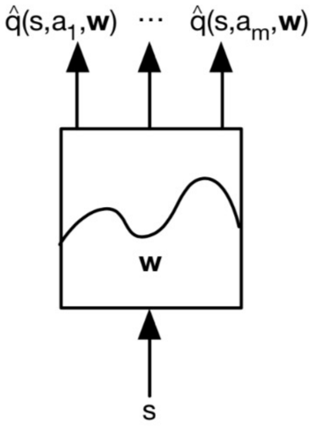
- Then a game-changing result was published in Nature:
  DQN from DeepMind (now Google DeepMind)



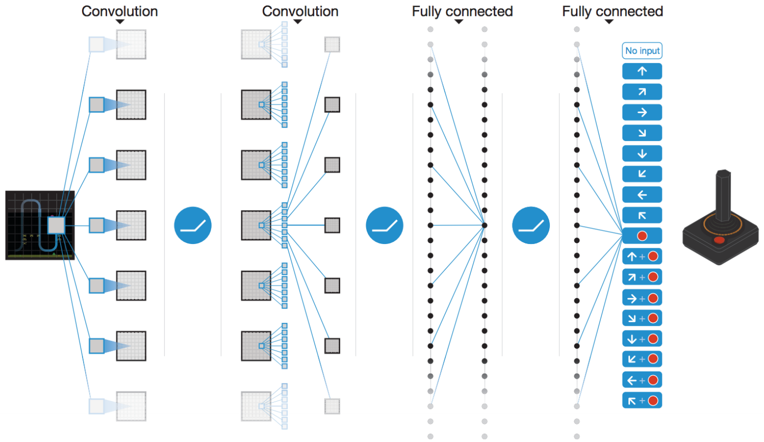
https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

Deep Q-Networks (DQNs)

- How does it work?
  - A convolutional neural network reads the image from the game (i.e., a framestack that uses the last $N = 4$ frames).
  - The CNN is a value function approximator for the $Q(s, a)$ function.
  - The reward is the game score.
  - The network weights are tuned using backpropagation signals of the rewards.



http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf



https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

Deep Q-Networks (DQNs)

- How does it work?
  - DQNs are „Q-Learning on steroids“ (Deep NN as VFA)
  - Training possible in Tensorflow (or Pytorch, Keras, …)
  - Objective function for gradient descent:

$$L(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i}[(y_i - Q(s,a,w))^2]$$

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: $w^- \leftarrow w_i$

Q-Learning with Linear VFA

- Approximated Q-function with parameters w and feature vector ϕ:

$$Q(s,a) \approx w^T\phi(s,a)$$

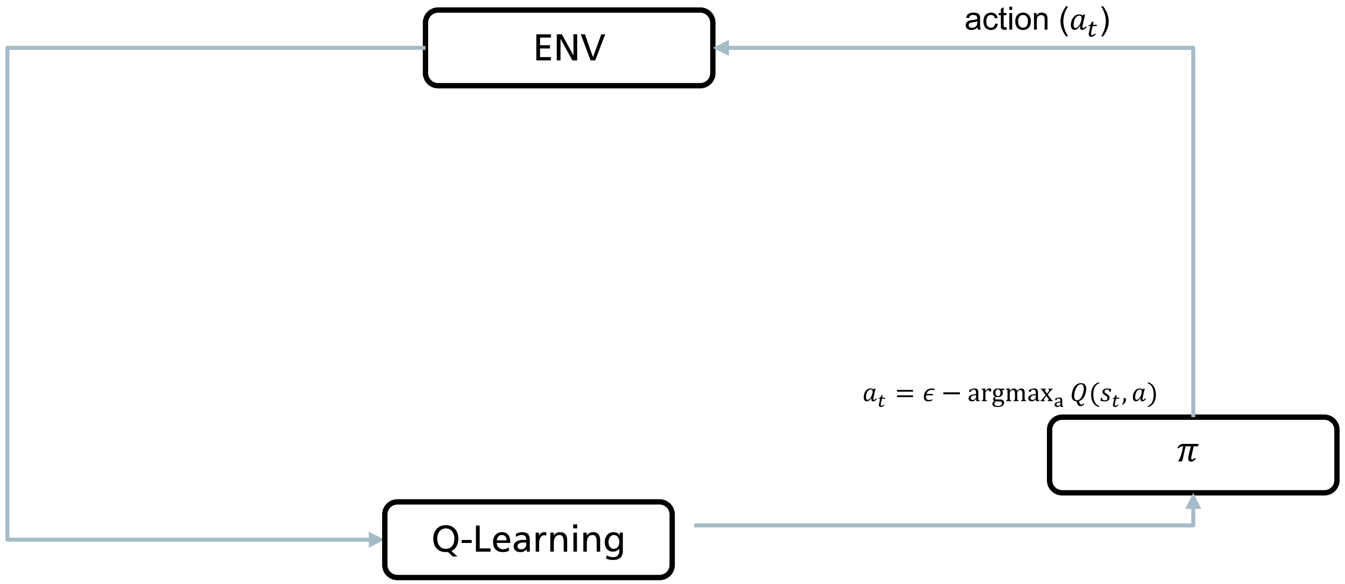
- Target value:

$$y_i = r + \gamma \max_{a' \in \mathcal{A}} w_i^T\phi(s',a')$$

- Updating:

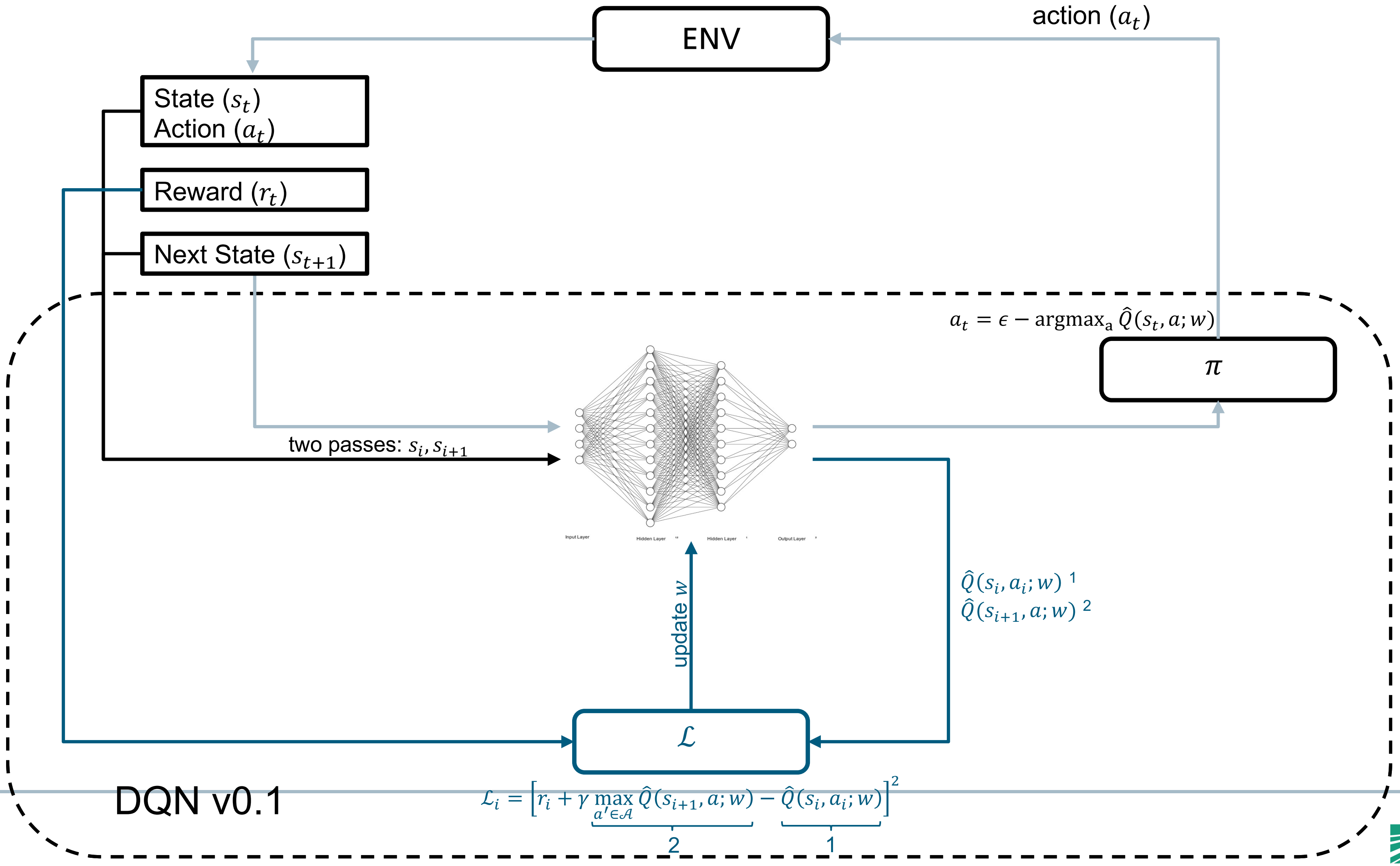
$$w_{i+1} = w_i + \alpha[y_i - w_i^T\phi(s,a)]\phi(s,a)$$

Deep Q-Networks (DQNs)



$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Deep Q-Networks (DQNs)



action (a_t)

ENV

State (s_t)
Action (a_t)

Reward (r_t)

Next State (s_{t+1})

two passes: s_i, s_{i+1}

$a_t = \epsilon - \mathrm{argmax_a}\, \hat{Q}(s_t, a; w)$

π

update w

$\hat{Q}(s_i, a_i; w)$ 1
$\hat{Q}(s_{i+1}, a; w)$ 2

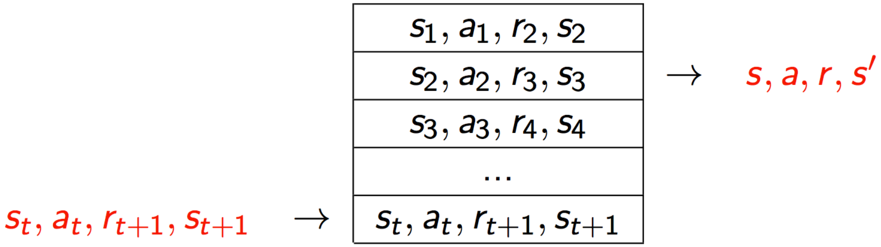
$\mathcal{L}$

$$\mathcal{L}_i = \left[r_i + \gamma \underbrace{\max_{a' \in \mathcal{A}} \hat{Q}(s_{i+1}, a; w)}_{2} - \underbrace{\hat{Q}(s_i, a_i; w)}_{1} \right]^2$$

DQN v0.1

Deep Q-Networks (DQNs)

- 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*.



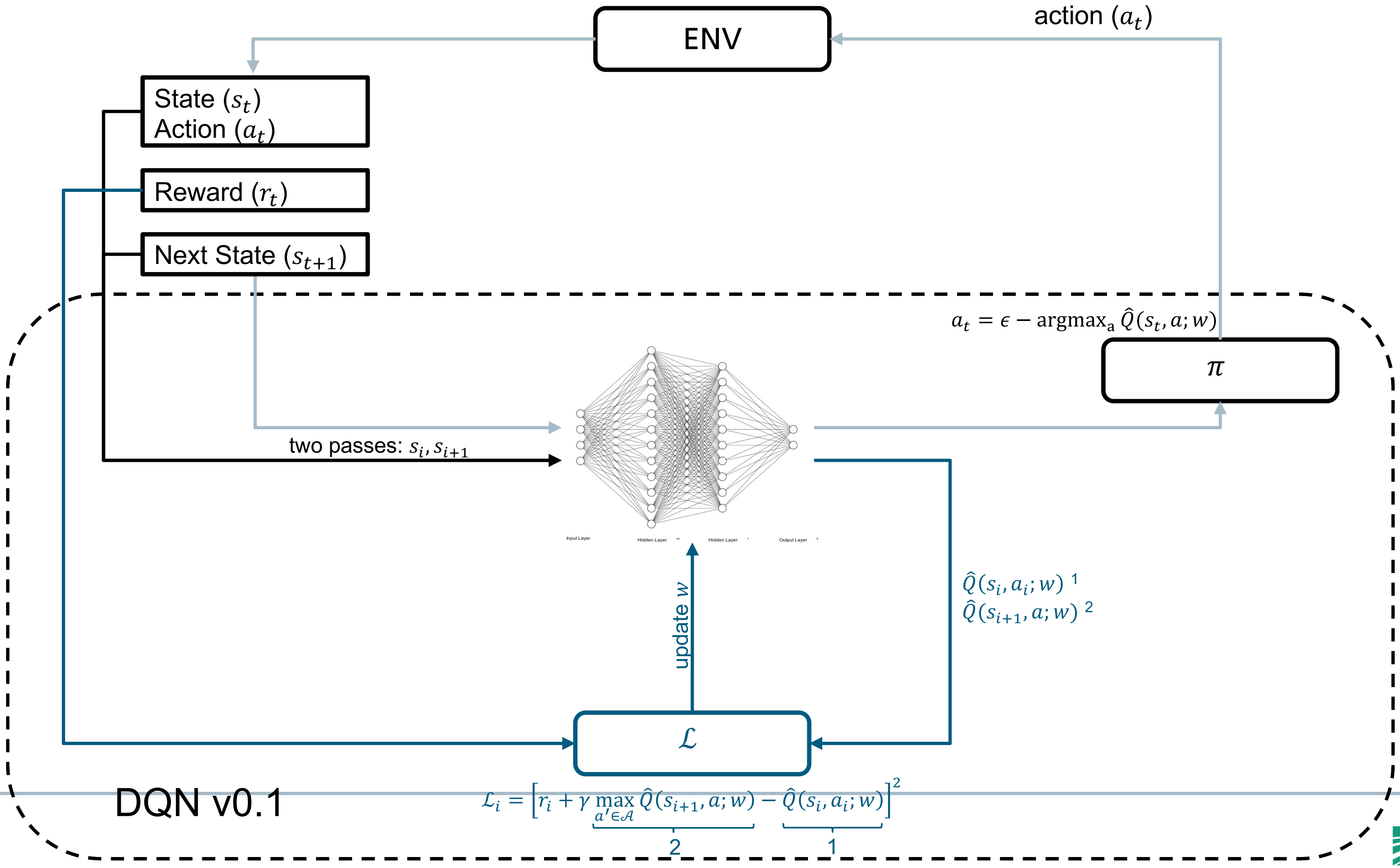
s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$s_t, a_t, r_{t+1}, s_{t+1} \rightarrow$

$\rightarrow s, a, r, s'$

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

Deep Q-Networks (DQNs)



ENV

action (a_t)

State (s_t)
Action (a_t)

Reward (r_t)

Next State (s_{t+1})

$a_t = \epsilon - \mathrm{argmax_a}\, \hat{Q}(s_t, a; w)$

π

two passes: s_i, s_{i+1}

$\hat{Q}(s_i, a_i; w)$ 1
$\hat{Q}(s_{i+1}, a; w)$ 2

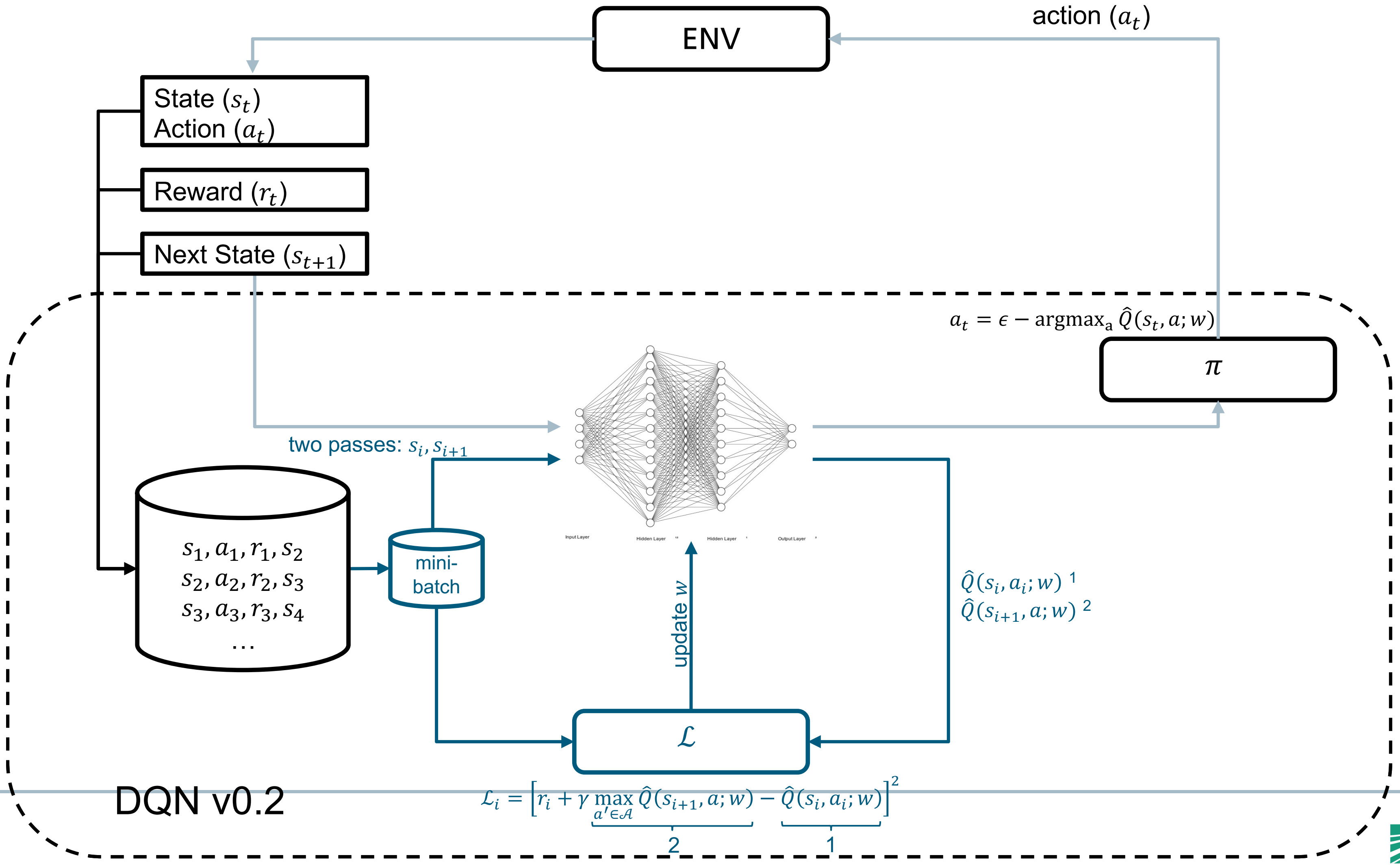
update w

$\mathcal{L}$

$$\mathcal{L}_i = \left[r_i + \gamma \underbrace{\max_{a' \in \mathcal{A}} \hat{Q}(s_{i+1}, a; w)}_{2} - \underbrace{\hat{Q}(s_i, a_i; w)}_{1} \right]^2$$

DQN v0.1

Deep Q-Networks (DQNs)



$$\mathcal{L}_i = \left[r_i + \gamma \underbrace{\max_{a' \in \mathcal{A}} \hat{Q}(s_{i+1}, a; w)}_{2} - \underbrace{\hat{Q}(s_i, a_i; w)}_{1}\right]^2$$

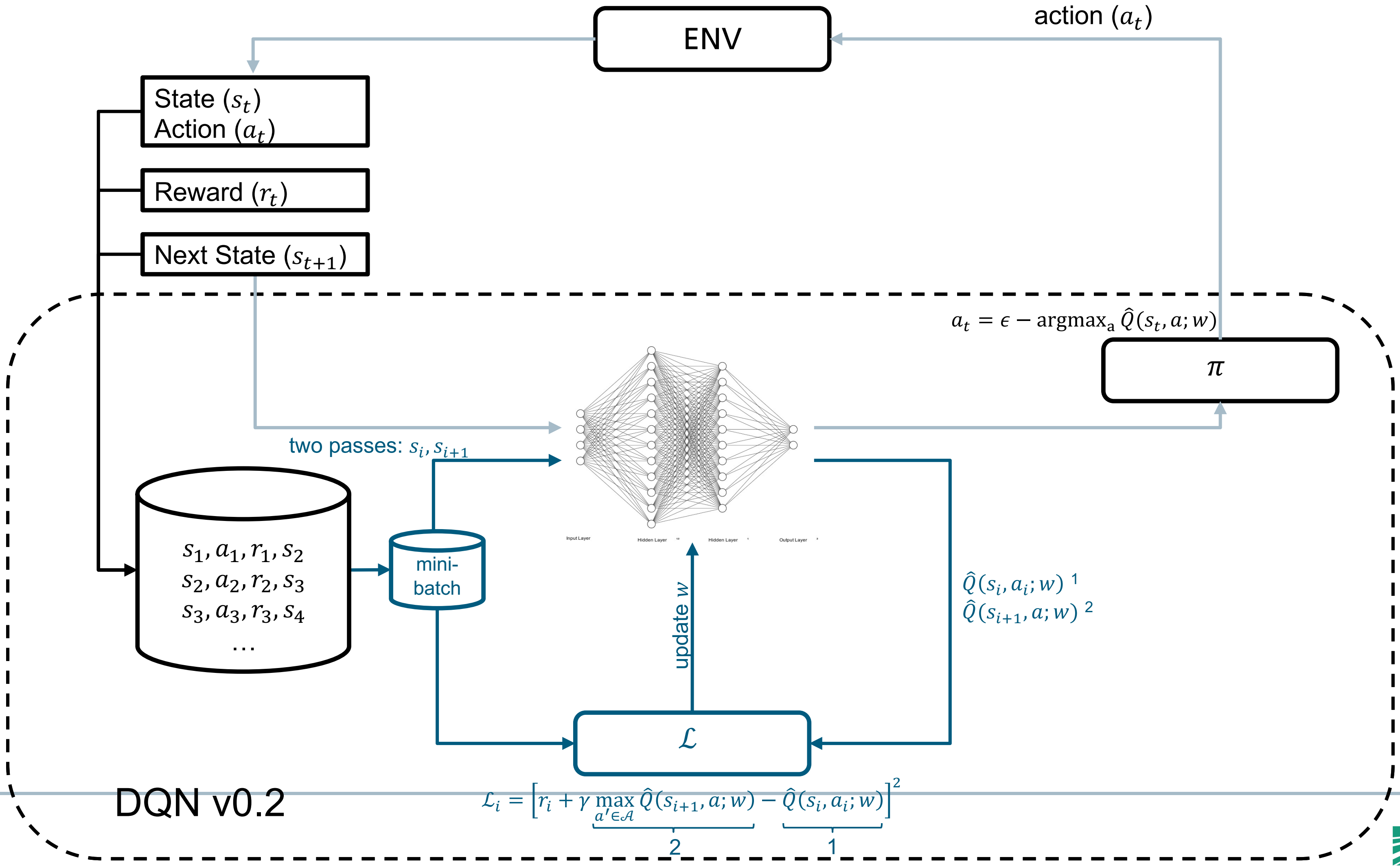
Deep Q-Networks (DQNs)

- 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.

Deep Q-Networks (DQNs)

- 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*.

Deep Q-Networks (DQNs)



ENV

action (a_t)

State (s_t)
Action (a_t)

Reward (r_t)

Next State (s_{t+1})

$a_t = \epsilon - \text{argmax}_a\, \hat{Q}(s_t, a; w)$

π

two passes: s_i, s_{i+1}

s_1, a_1, r_1, s_2
s_2, a_2, r_2, s_3
s_3, a_3, r_3, s_4
$\dots$

mini-batch

update w

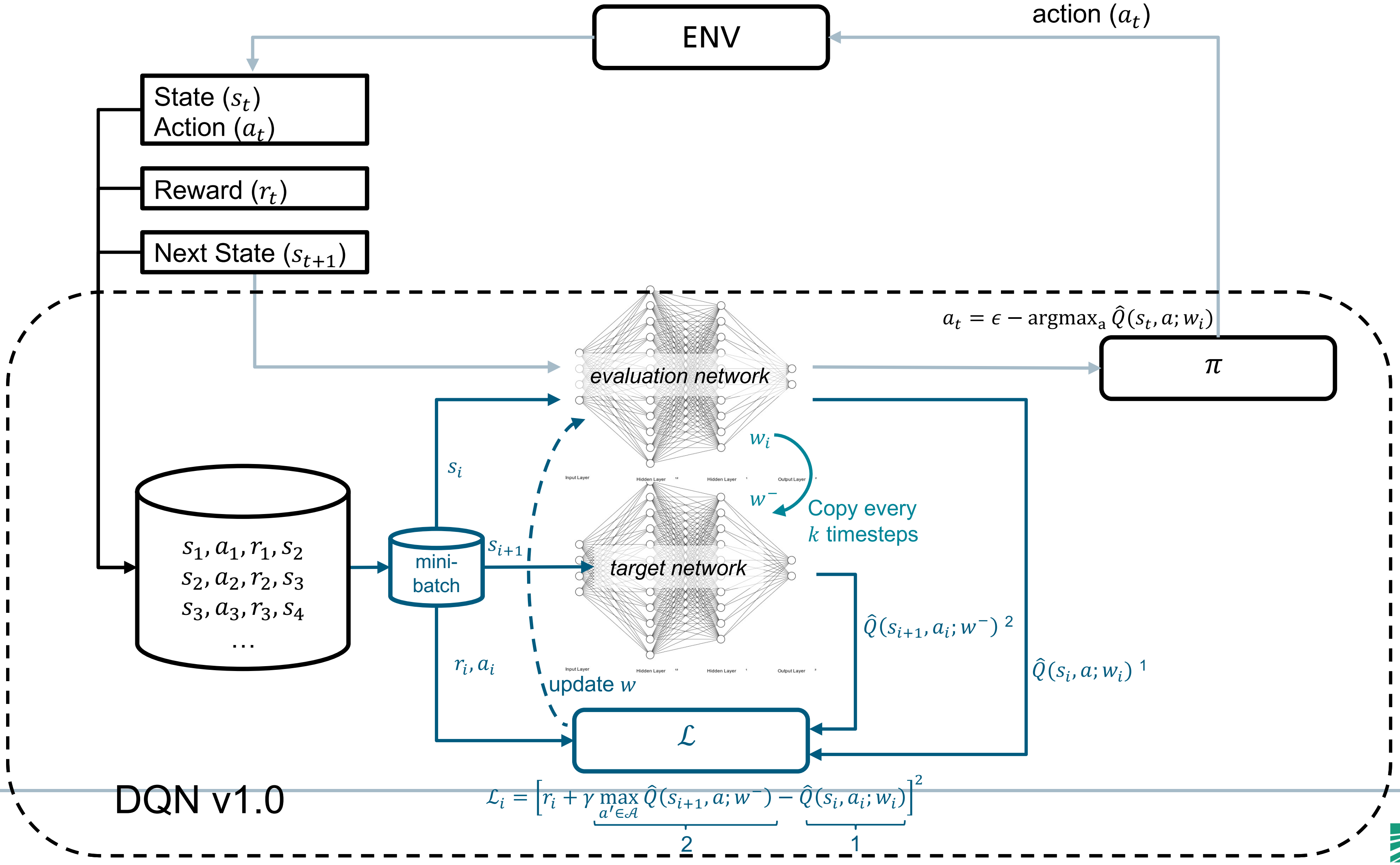
$\hat{Q}(s_i, a_i; w)$ 1
$\hat{Q}(s_{i+1}, a; w)$ 2

$\mathcal{L}$

$$\mathcal{L}_i = \left[r_i + \underbrace{\gamma \max_{a' \in \mathcal{A}} \hat{Q}(s_{i+1}, a; w)}_{2} - \underbrace{\hat{Q}(s_i, a_i; w)}_{1} \right]^2$$

DQN v0.2

Deep Q-Networks (DQNs)

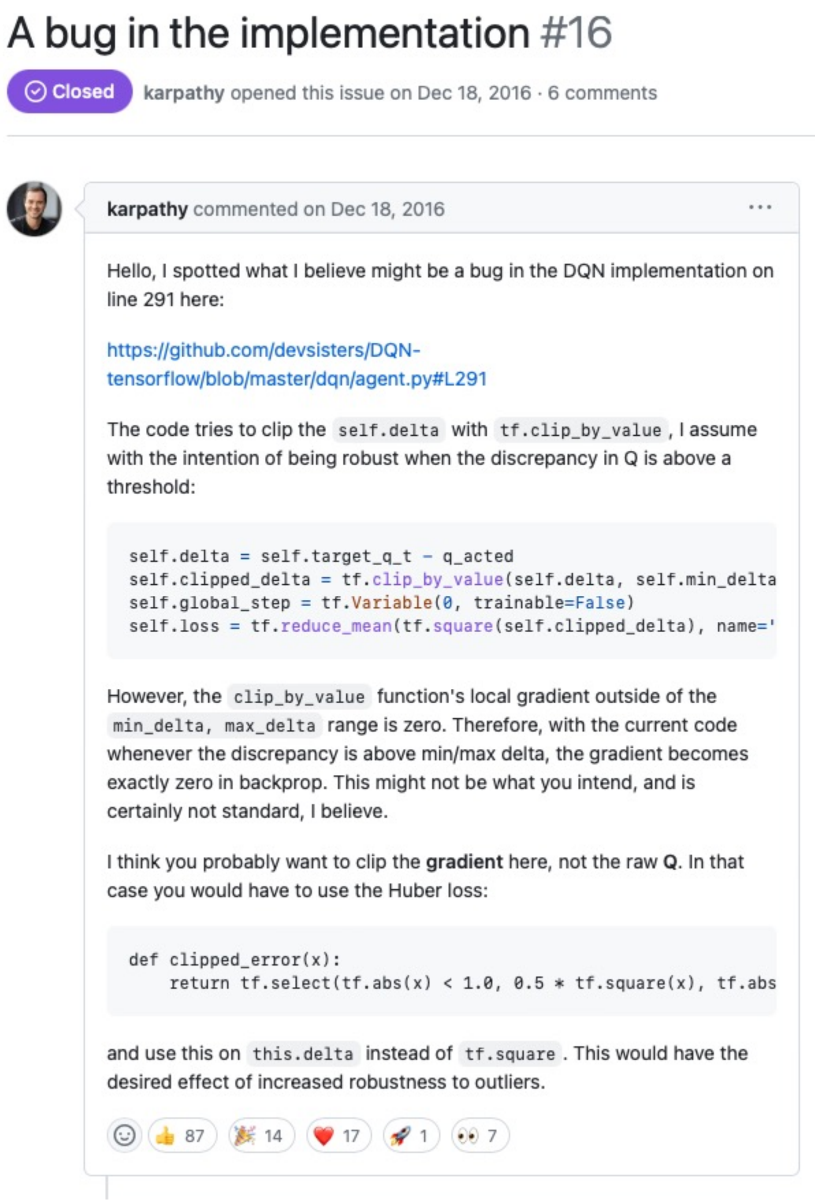


$$\mathcal{L}_i = \Big[r_i + \underbrace{\gamma \max_{a' \in \mathcal{A}} \hat{Q}(s_{i+1}, a; w^-)}_{2} - \underbrace{\hat{Q}(s_i, a_i; w_i)}_{1}\Big]^2$$

Deep Q-Networks (DQNs)

- 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 gam
     - Prevents Q-values from becoming too large.



A bug in the implementation #16

karpathy opened this issue on Dec 18, 2016 · 6 comments

karpathy commented on Dec 18, 2016

Hello, I spotted what I believe might be a bug in the DQN implementation on line 291 here:

https://github.com/devsisters/DQN-tensorflow/blob/master/dqn/agent.py#L291

The code tries to clip the `self.delta` with `tf.clip_by_value`, I assume with the intention of being robust when the discrepancy in Q is above a threshold:

```
self.delta = self.target_q_t - q_acted
self.clipped_delta = tf.clip_by_value(self.delta, self.min_delta
self.global_step = tf.Variable(0, trainable=False)
self.loss = tf.reduce_mean(tf.square(self.clipped_delta), name='
```

However, the `clip_by_value` function's local gradient outside of the `min_delta, max_delta` range is zero. Therefore, with the current code whenever the discrepancy is above min/max delta, the gradient becomes exactly zero in backprop. This might not be what you intend, and is certainly not standard, I believe.

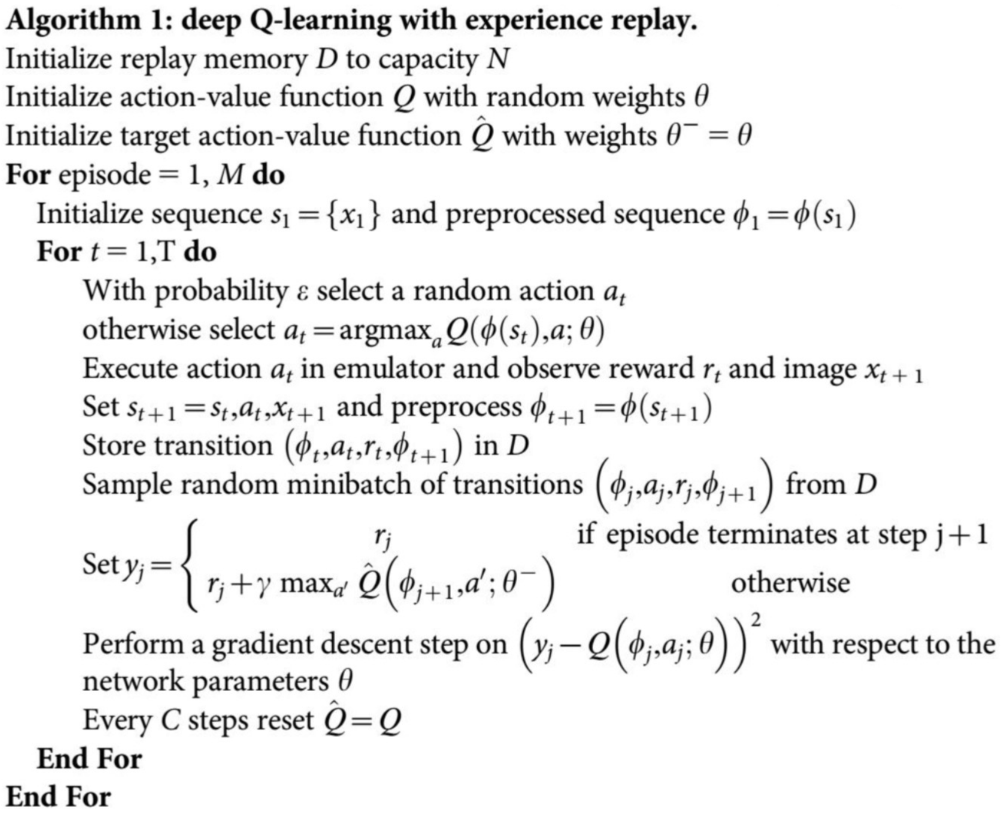
I think you probably want to clip the **gradient** here, not the raw **Q**. In that case you would have to use the Huber loss:

```
def clipped_error(x):
    return tf.select(tf.abs(x) < 1.0, 0.5 * tf.square(x), tf.abs
```

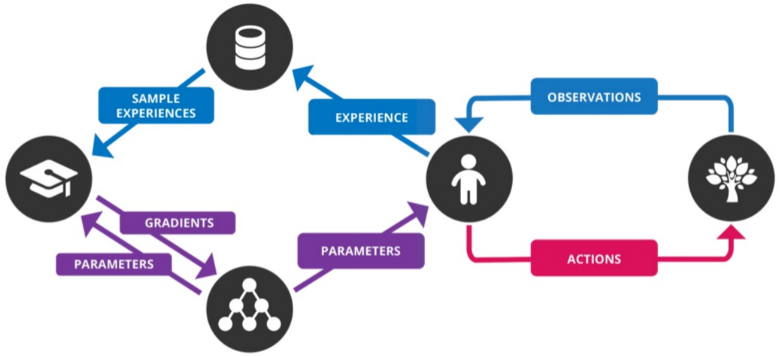
and use this on `this.delta` instead of `tf.square`. This would have the desired effect of increased robustness to outliers.

Deep Q-Networks (DQNs)

How does it work?



```
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
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 $\varepsilon$ 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 $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    End For
End For
```



https://sites.google.com/view/deep-rl-bootcamp/lectures

Deep Q-Networks (DQNs)

- **Extension #1: Double DQN (DDQN)**
- **Problem:** Upward positive bias (overestimation of Q-values) in targets:

$$y_i = r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s', a'; \vartheta_i)$$

- DDQN splits action selection and action evaluation:

$$y_i = r + \gamma \hat{Q}\left(s', \underset{a' \in \mathcal{A}}{argmax}\, \hat{Q}(s', a'; \theta_i); \vartheta_i\right)$$

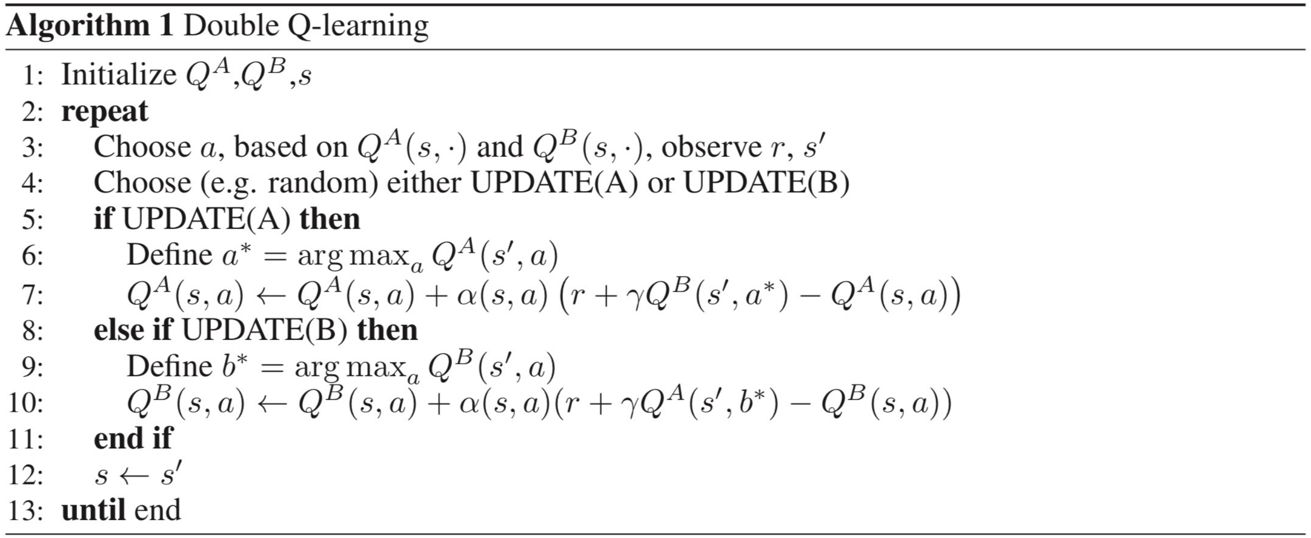
  - Use estimations from main Q-network to select actions
  - Use estimations from target Q-network to evaluate actions

Deep Q-Networks (DQNs)

- **Extension #1: Double DQN (DDQN)**
- DDQN splits action selection and action evaluation:

$$y_i = r + \gamma \hat{Q}\left(s', \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \hat{Q}(s', a'; \theta_i)\, ; \vartheta_i\right)$$

  - Use estimations from main Q-network to select actions
  - Use estimations from target Q-network to evaluate actions



Algorithm 1 Double Q-learning

```
Initialize $Q^A$,$Q^B$,$s$
**repeat**
    Choose $a$, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe $r$, $s'$
    Choose (e.g. random) either UPDATE(A) or UPDATE(B)
    **if** UPDATE(A) **then**
        Define $a^* = \arg\max_a Q^A(s',a)$
        $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
    **else if** UPDATE(B) **then**
        Define $b^* = \arg\max_a Q^B(s',a)$
        $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)(r + \gamma Q^A(s',b^*) - Q^B(s,a))$
    **end if**
    $s \leftarrow s'$
**until** end
```

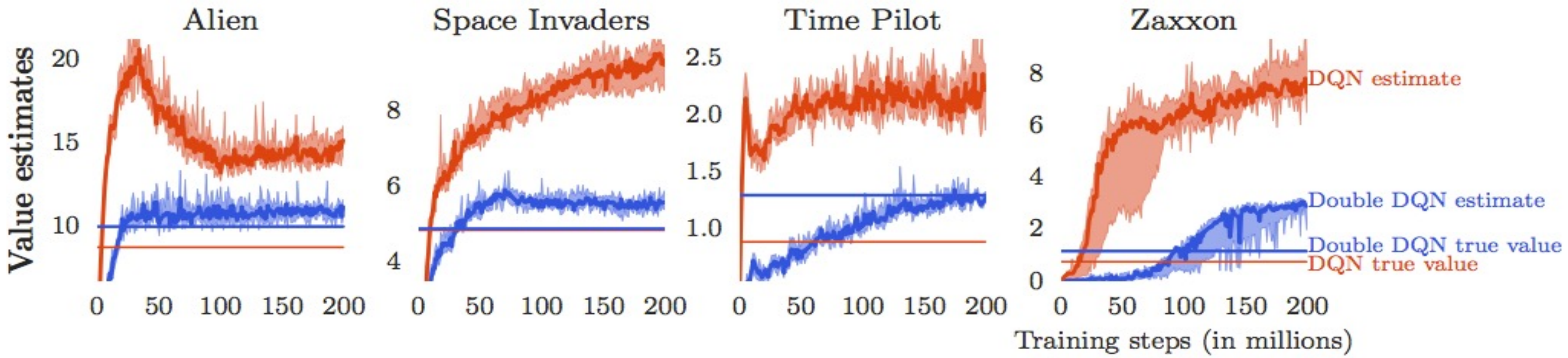
Hasselt, H. V. (2010). Double Q-learning. In *Advances in Neural Information Processing Systems* (pp. 2613-2621).

Deep Q-Networks (DQNs)

- **Extension #1: Double DQN (DDQN)**
- DDQN splits action selection and action evaluation:

$$y_i = r + \gamma \hat{Q}\left(s', \underset{a' \in \mathcal{A}}{\operatorname{argmax}} \hat{Q}(s', a'; \theta_i); \vartheta_i\right)$$

  - Use estimations from main Q-network to select actions
  - Use estimations from target Q-network to evaluate actions



Hasselt, H. V. (2015). Deep Reinforcement Learning with Double Q-learning. In *AAAI* (pp. 2094-2100).

Deep Q-Networks (DQNs)

- **Extension #2: Prioritized Experience Replay**
- Some experiences retain more information for learning than others
- **Problem:** Experience Replay Sampling is uniform sampling
- **Prioritized Experience Replay** samples mini-batches of based on their absolute Bellman error e:

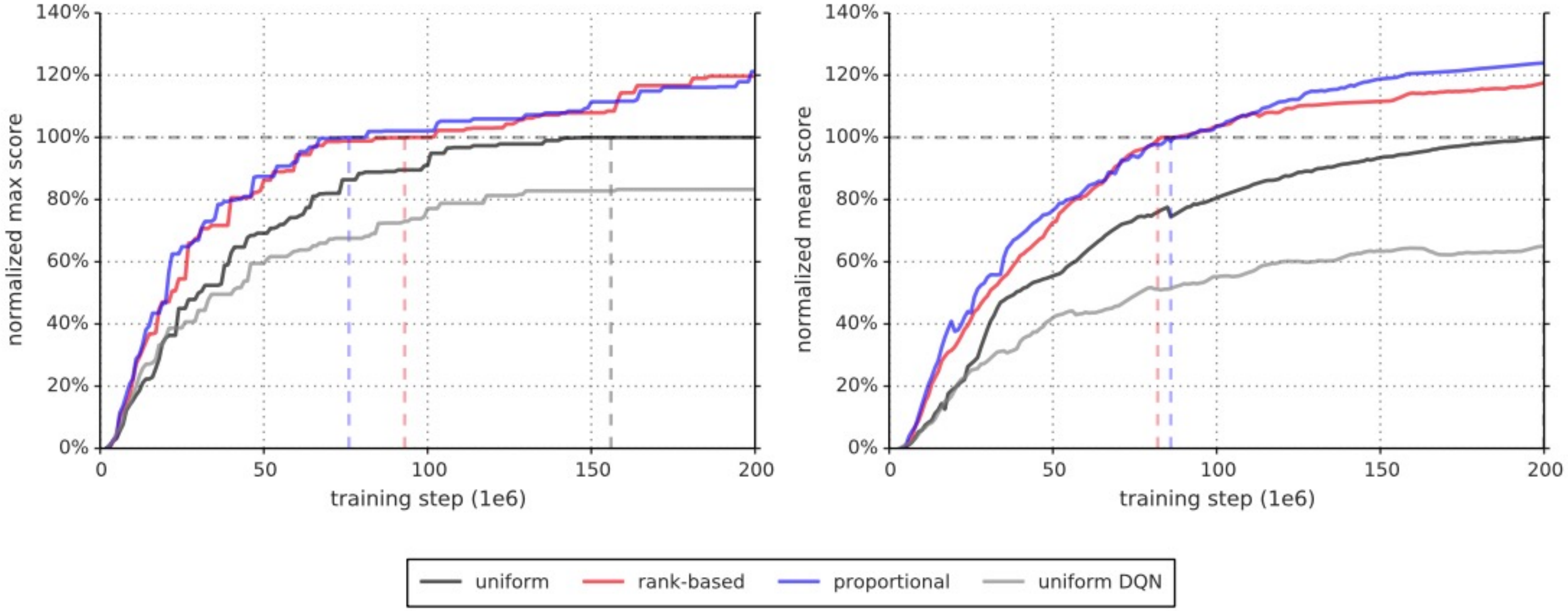
$$\delta = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a)$$
$$e = |\delta|$$

- Using DDQN notation:

$$y_i = r + \gamma \hat{Q}\left(s', \underset{a' \in \mathcal{A}}{argmax}\, \hat{Q}(s', a'; \theta_i); \vartheta_i\right)$$
$$\delta = y_i - Q(s, a; \theta_i)$$
$$e = |\delta|$$

Deep Q-Networks (DQNs)

- **Extension #2: Prioritized Experience Replay**
- Some experiences retain more information for learning than others
- **Problem:** Experience Replay Sampling is uniform sampling
- **Prioritized Experience Replay** leads to much faster learning

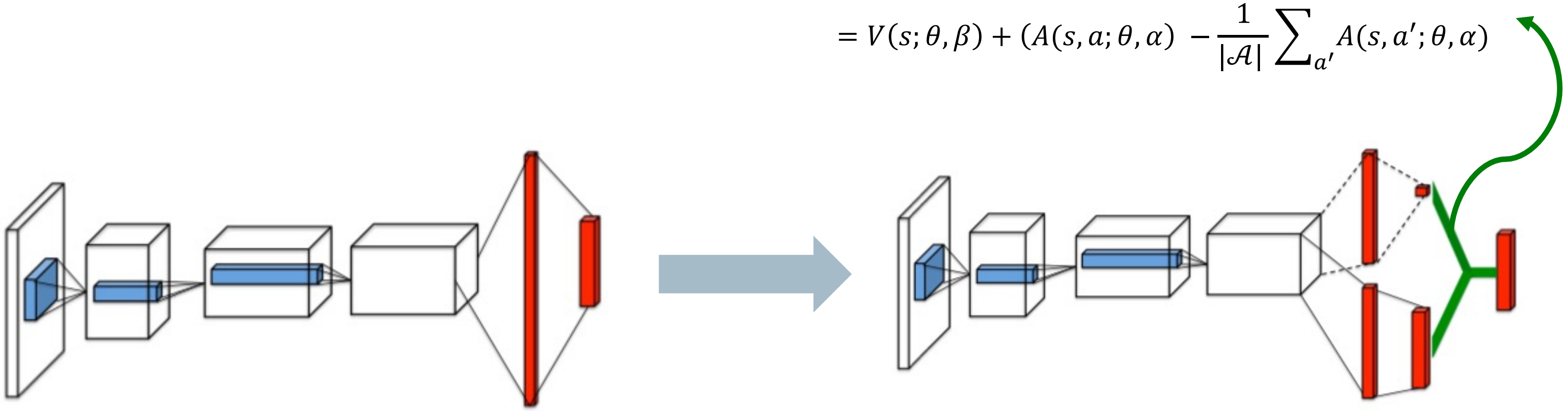


https://arxiv.org/pdf/1511.05952.pdf

Deep Q-Networks (DQNs)

- **Extension #3: Dueling Architectures**
- Split Q-network into two channels:
  - Action-independent value function $V(s; \mathbf{v})$
  - Action-dependent advantage function $A(s, a; \mathbf{w})$

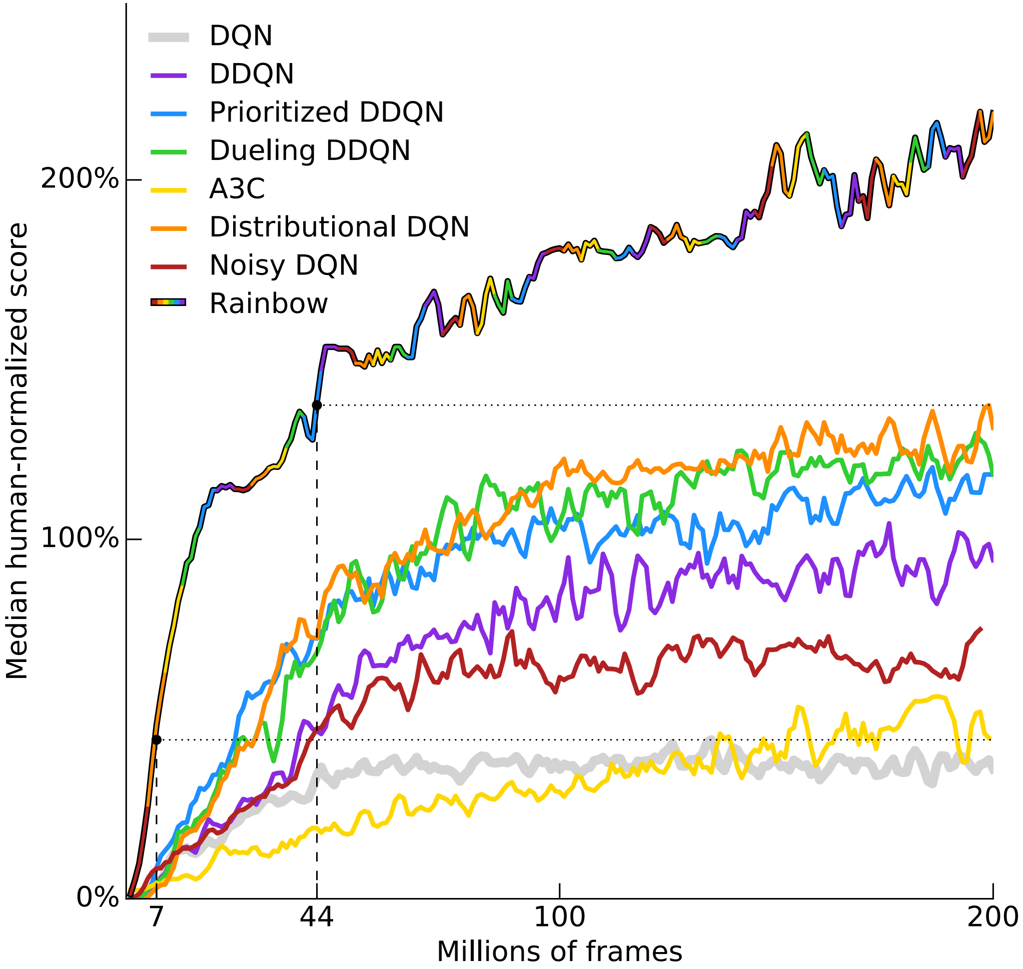
  → $Q(s, a) = V(s; \mathbf{v}) + A(s, a; \mathbf{w})$



$$Q(s,a;\theta,\alpha,\beta)$$
$$= V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum\nolimits_{a'} A(s,a';\theta,\alpha)$$

Wang et al.: Dueling Network Architectures for Deep Reinforcement Learning

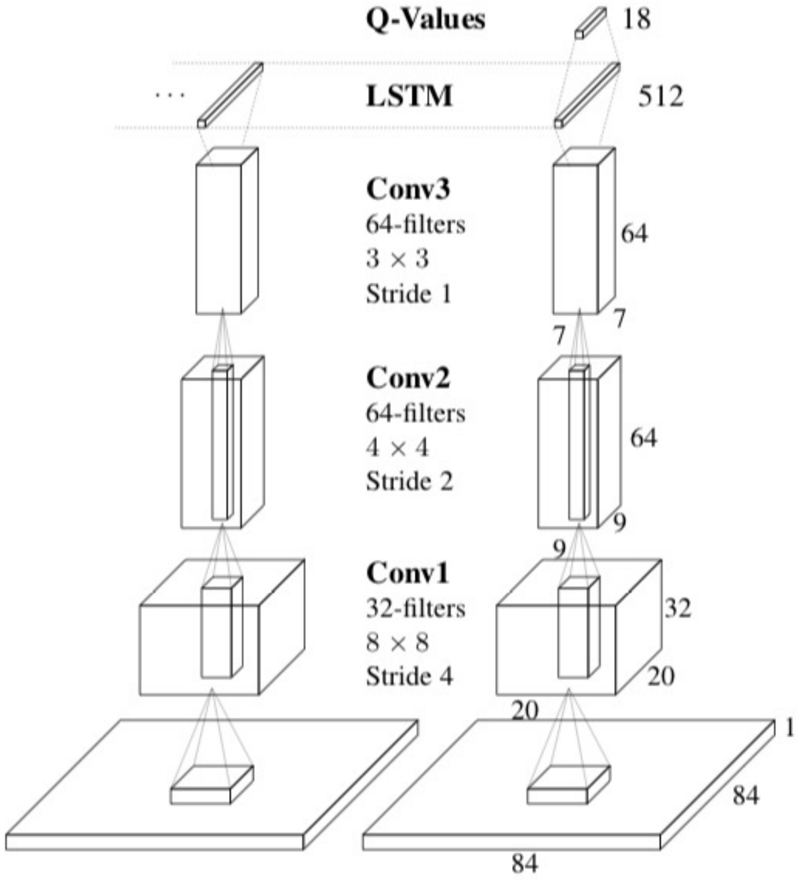
Deep Q-Networks (DQNs)



Hessel et al.: Rainbow: Combining Improvements in Deep Reinforcement Learning

Deep Q-Networks (DQNs)

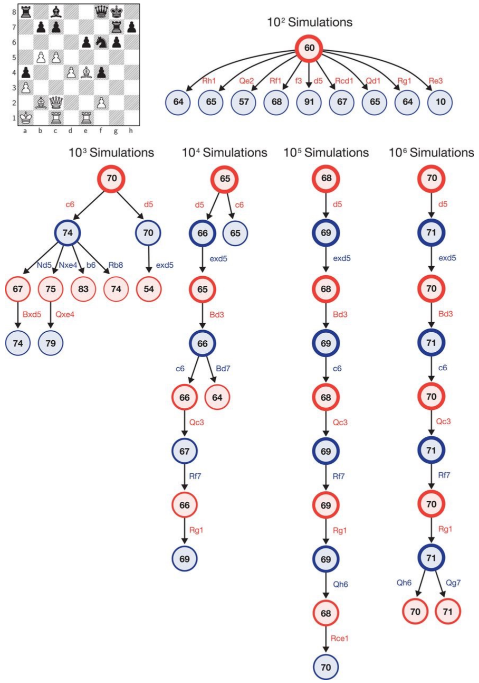
- **Extension: Deep Recurrent Q-Learning Networks (DRQNs)**
- DQN in Atari needed to process more frames (4) in order to get an estimate of hidden state (e.g., ball velocity in pong)
- Many real-world problems have partially observable states (POMDPs)
- So instead of augmenting states in the POMDP case, why don't we use an RNN (LSTM) instead?
- Caution: use only when you have a true POMDP problem as it adds significant complexity during training. If you are not sure, try augmenting states first!

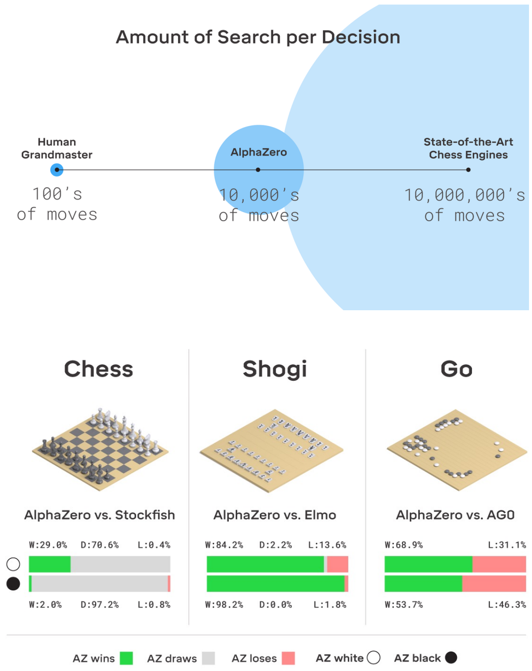


https://arxiv.org/pdf/1507.06527.pdf

Deep Q-Networks (DQNs)

Beyond just an extension: AlphaZero





https://deepmind.com/blog/alphazero-shedding-new-light-grand-games-chess-shogi-and-go/

Value Function Approximation

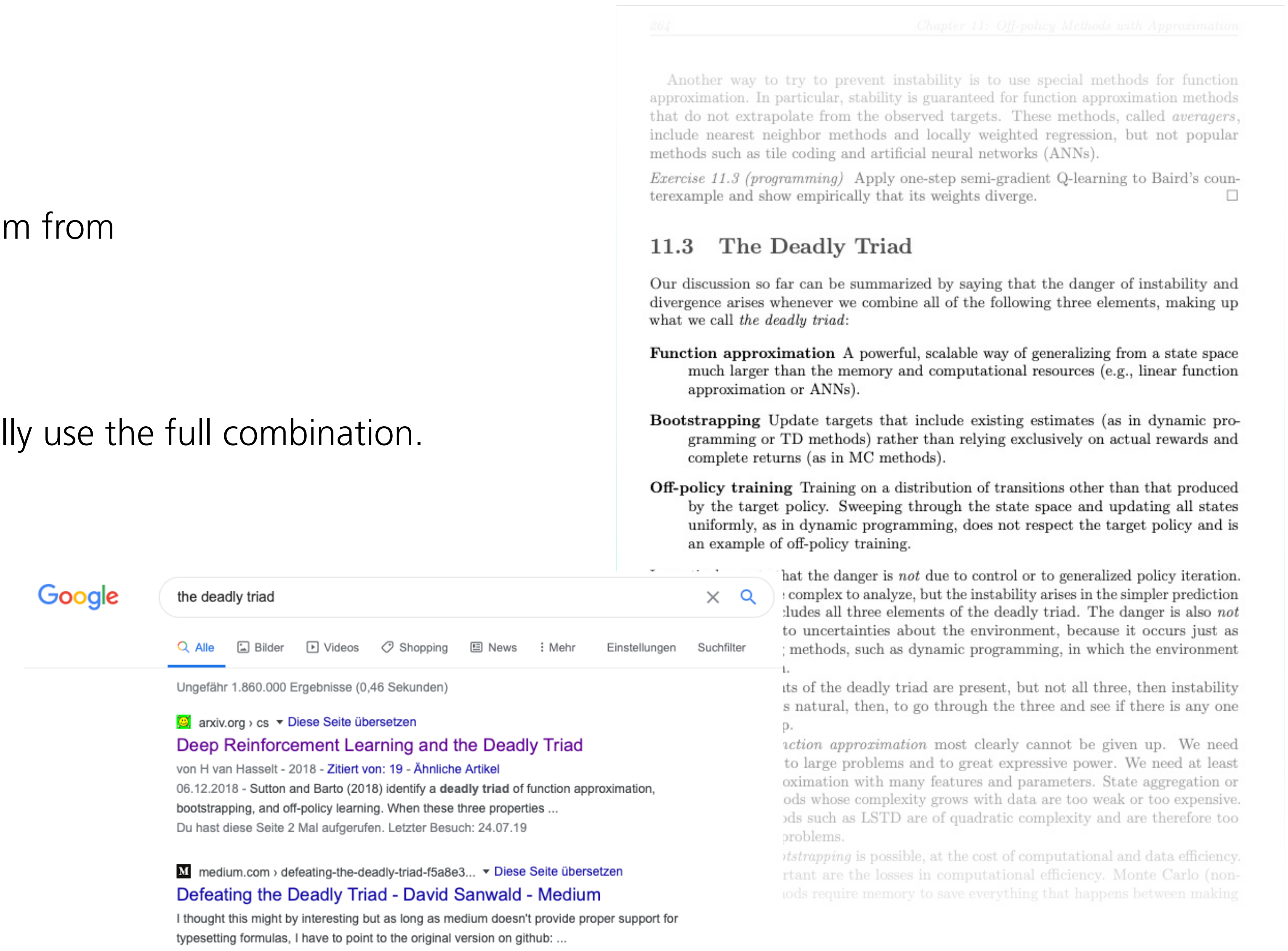
- In practice we often need to approximate the value function, because:
  - (Explicit) tabular representations require too much space
  - We want to generalize information across state (see also: POMDPs!)

- For linear function approximation almost all convergence guarantees hold
  - For non-linear function approximation such guarantees cannot be given
  - But careful scheduling and several tricks help to stabilize training

- But:
  - Non-linear function approximation is very sensitive to hyper-parameter tuning!
  - See also: https://www.youtube.com/watch?v=Vh4H0gOwdIg
    (not directly related but definitely worth watching!)
  - And also: https://www.alexirpan.com/2018/02/14/rl-hard.html
    (but please read with humor)

The Deadly Triad

- Stability in RL is a very serious thing!
- Instability and divergence in RL mainly stem from
  1. Function Approximation.
  2. Bootstrapping.
  3. Off-policy training.
- Unfortunately, in most of the case we really use the full combination.



264 *Chapter 11: Off-policy Methods with Approximation*

Another way to try to prevent instability is to use special methods for function approximation. In particular, stability is guaranteed for function approximation methods that do not extrapolate from the observed targets. These methods, called *averagers*, include nearest neighbor methods and locally weighted regression, but not popular methods such as tile coding and artificial neural networks (ANNs).

Exercise 11.3 (programming) Apply one-step semi-gradient Q-learning to Baird's counterexample and show empirically that its weights diverge. □

11.3 The Deadly Triad

Our discussion so far can be summarized by saying that the danger of instability and divergence arises whenever we combine all of the following three elements, making up what we call *the deadly triad*:

Function approximation A powerful, scalable way of generalizing from a state space much larger than the memory and computational resources (e.g., linear function approximation or ANNs).

Bootstrapping Update targets that include existing estimates (as in dynamic programming or TD methods) rather than relying exclusively on actual rewards and complete returns (as in MC methods).

Off-policy training Training on a distribution of transitions other than that produced by the target policy. Sweeping through the state space and updating all states uniformly, as in dynamic programming, does not respect the target policy and is an example of off-policy training.

hat the danger is *not* due to control or to generalized policy iteration.
complex to analyze, but the instability arises in the simpler prediction
ludes all three elements of the deadly triad. The danger is also *not*
to uncertainties about the environment, because it occurs just as
methods, such as dynamic programming, in which the environment

its of the deadly triad are present, but not all three, then instability
s natural, then, to go through the three and see if there is any one
p.

iction approximation most clearly cannot be given up. We need
to large problems and to great expressive power. We need at least
oximation with many features and parameters. State aggregation or
ods whose complexity grows with data are too weak or too expensive.
ods such as LSTD are of quadratic complexity and are therefore too
problems.

tstrapping is possible, at the cost of computational and data efficiency.
rtant are the losses in computational efficiency. Monte Carlo (non-
iods require memory to save everything that happens between making

Google

the deadly triad

Ungefähr 1.860.000 Ergebnisse (0,46 Sekunden)

arxiv.org › cs ▾ Diese Seite übersetzen

Deep Reinforcement Learning and the Deadly Triad

von H van Hasselt - 2018 - Zitiert von: 19 - Ähnliche Artikel

06.12.2018 - Sutton and Barto (2018) identify a **deadly triad** of function approximation, bootstrapping, and off-policy learning. When these three properties ...

Du hast diese Seite 2 Mal aufgerufen. Letzter Besuch: 24.07.19

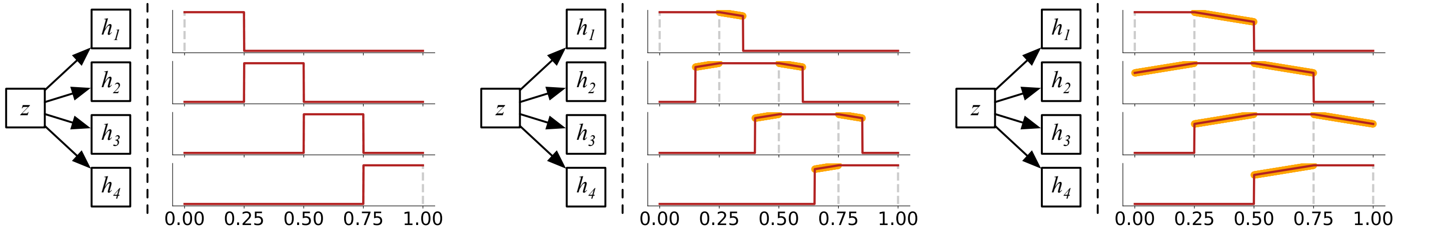
medium.com › defeating-the-deadly-triad-f5a8e3... ▾ Diese Seite übersetzen

Defeating the Deadly Triad - David Sanwald - Medium

I thought this might by interesting but as long as medium doesn't provide proper support for typesetting formulas, I have to point to the original version on github: ...

Fuzzy Tiling Activations

- DQNs need target networks to reduce the chance of divergence
- Main reason:
  - The Q-Targets are non-stationary and moving over subsequent gradient descent steps
- Idea:
  - Introduce a special activation function that produces sparse representations!
    i.e., updates on a particular Q-value does no affect nearby Q-values that much.
  - FTA layers stack k-dimensional sparse encodings for each element h_1W_2 ($h_1 = xW_1$)



(a) TA, $k = 4$ (b) FTA, $k = 4, \eta = 0.1$ (c) FTA, $k = 4, \eta = 0.25$

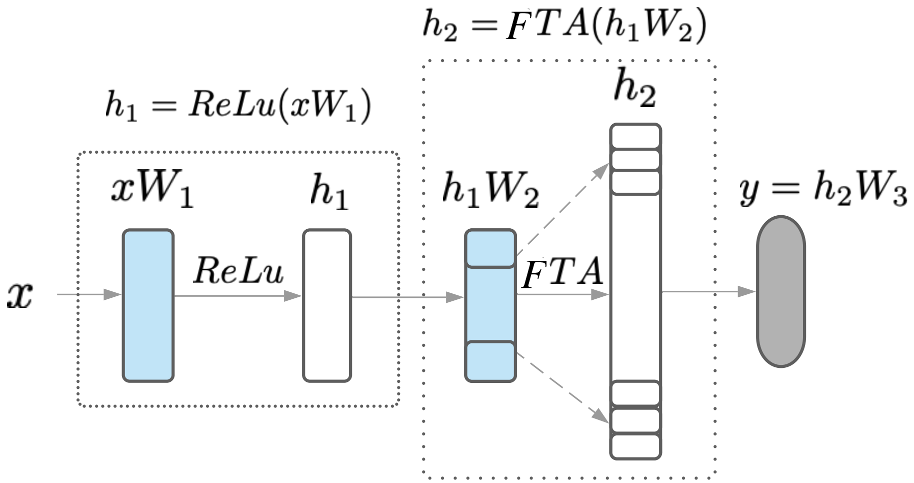


Figure 2: A visualization of an FTA layer

Pan et al.: Fuzzy Tiling Activations: A Simple Approach to Learning Sparse Representations Online. ICLR 2021.

Lesson of today

“Be careful with (non-linear) function approximation“

References

- Going Deeper Into Reinforcement Learning: Understanding Q-Learning and Linear Function Approximation; https://danieltakeshi.github.io/2016/10/31/going-deeper-into-reinforcement-learning-understanding-q-learning-and-linear-function-approximation/
- Watkins, C. J. C. H. (1989). Learning from delayed rewards(Doctoral dissertation, King's College, Cambridge): https://link.springer.com/content/pdf/10.1007/BF00992698.pdf
- Lin, L. J. (1993). Reinforcement learning for robots using neural networks (No. CMU-CS-93-103). Carnegie-Mellon Univ Pittsburgh PA School of Computer Science: http://www.dtic.mil/dtic/tr/fulltext/u2/a261434.pdf
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529: http://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf
- Van Hasselt, H., Guez, A., & Silver, D. (2016, February). Deep Reinforcement Learning with Double Q-Learning. In AAAI (Vol. 2, p. 5): http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/download/12389/11847
- Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581: https://arxiv.org/abs/1511.06581
- Hausknecht, M., & Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. CoRR, abs/1507.06527, 7(1): http://www.aaai.org/ocs/index.php/FSS/FSS15/paper/download/11673/11503
- Van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., & Modayil, J. (2018). Deep Reinforcement Learning and the Deadly Triad. arXiv preprint arXiv:1812.02648.: https://arxiv.org/abs/1812.02648