Intro to Reinforcement Learning

University of Pretoria

By Kale-ab Tessera, Divanisha Patel and Arnu Pretorius.
May, 2023

Overview

- Why RL
- RL Flow and Intuition
- RL Formalism
- Q-Learning
- Deep Q-Learning
- RL at InstaDeep

Why RL

Go

Strategy board game where two players try to surround opponents pieces.

Likely the world's oldest board game, is thought to have originated in China 4,000 years ago. [1]

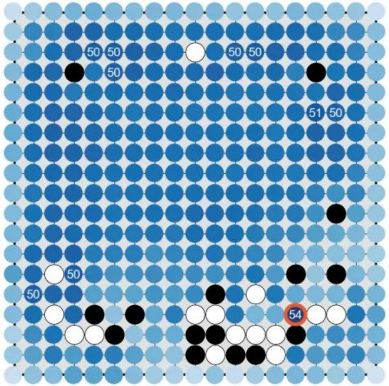
Attempts to solve Go!

- Many attempts to solve, but unsuccessful.
- Number of configurations of board - 10^170 - “more than number of atoms in the universe” - Alpha GO (chess - ~10^50 possible positions)

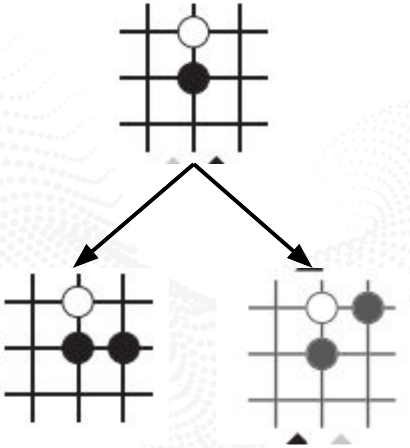
Image, 1] https://www.britannica.com/topic/go-game

AlphaGo - Deep RL Based Computer Program Plays Go

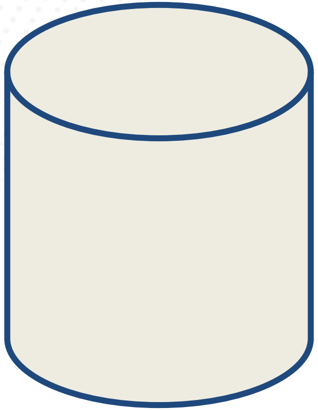
RL Problem



MCTS (Monte Carlo Tree Search)



Database of Expert Knowledge (~30 million moves) + Self-Play



Mastering the game of Go with deep neural networks and tree search

AlphaGo - 2016 - Beat World Champion

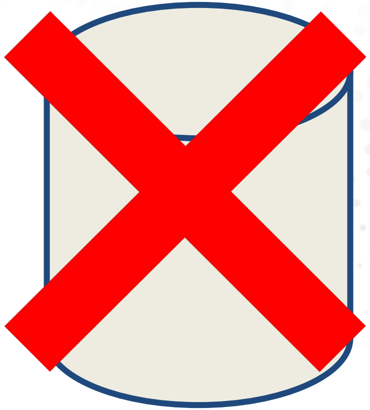
"I thought AlphaGo was based on probability calculation and that it was merely a machine. But when I saw this move, I changed my mind. Surely, AlphaGo is creative."

- Lee Sedol - Winner of 18 world Go titles

AlphaZero - Learn from Self-Play (No Human Knowledge)

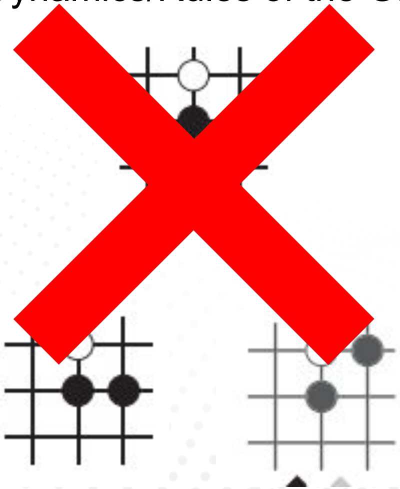
AlphaZero

No Database of Expert Knowledge



MuZero - Mastering Go, chess, shogi and Atari without rules

Dynamics/Rules of the Game.



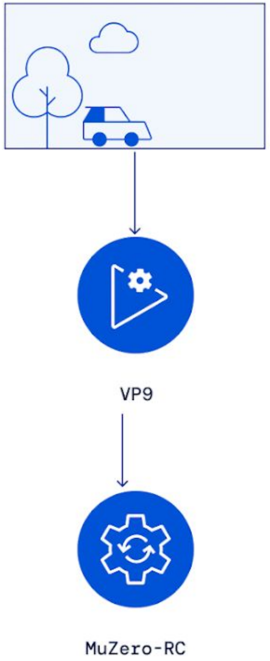
Real world settings - the rules or dynamics are typically unknown and complex.

MuZero

Beyond Games

InstaDeep™

MuZero - YouTube to optimise video compression



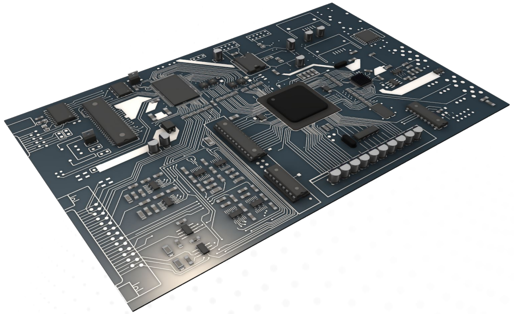
MuZero Youtube

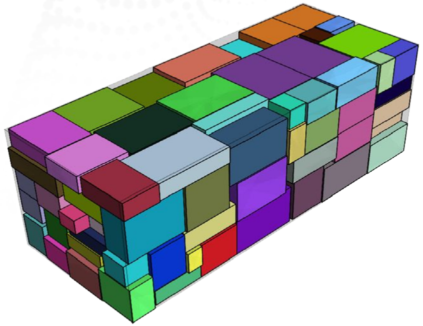
RL at InstaDeep

Deep**PCB**™
(Hardware/IOT)

Deep**Pack**™
(Logistics/Supply Chain)

Deep**Rail**™
(Fleet Management)





Design complex printed circuit boards in less than 24 Hours

Accelerates the product cycle in IOT and consumer electronics

Pack items more efficiently to improve supply chain logistics

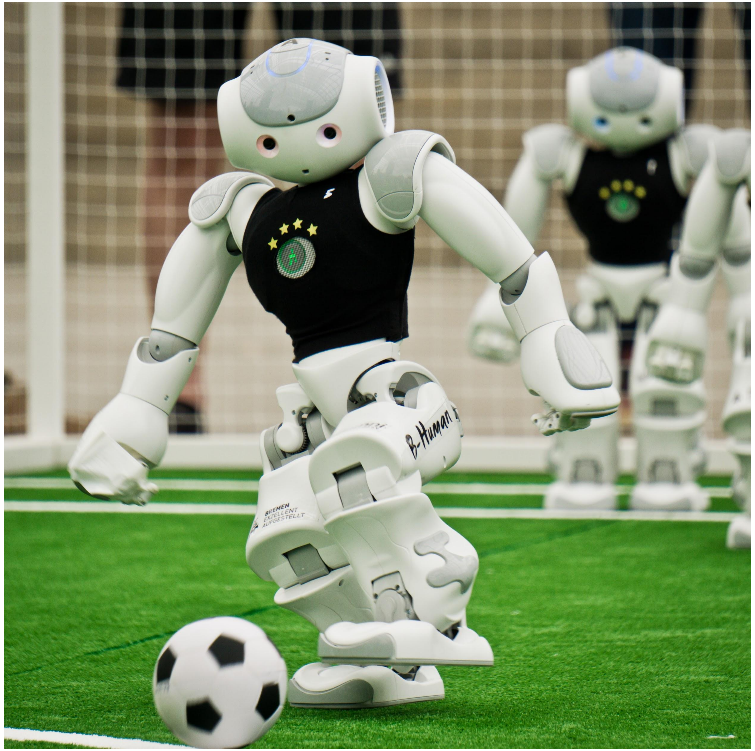
Save money on transport costs for large shipments

Optimize train scheduling and mobility fleet management

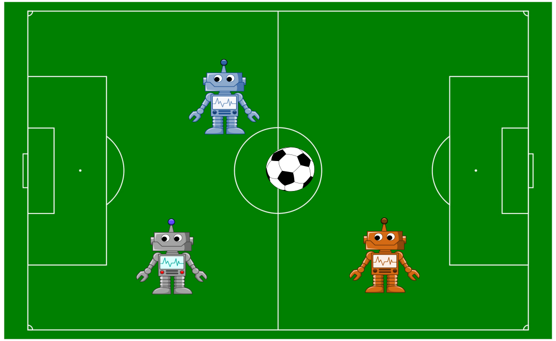
Reduces passenger delays, better yields on infrastructure projects

RL Flow and Intuition

Practical Setting - Robot Playing Football



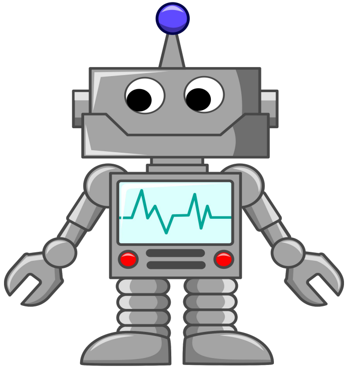
Reinforcement Learning (RL) Loop



Environment:

- The system we care about - returns our reward signal.
- What our "agent" **sees** and **interacts** with.

Reinforcement Learning (RL) Loop



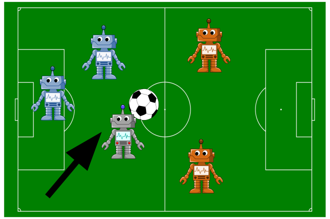
Agent:

- **Interacts** with the environment.
- Entity that makes **decisions**, adapts and **learns**.

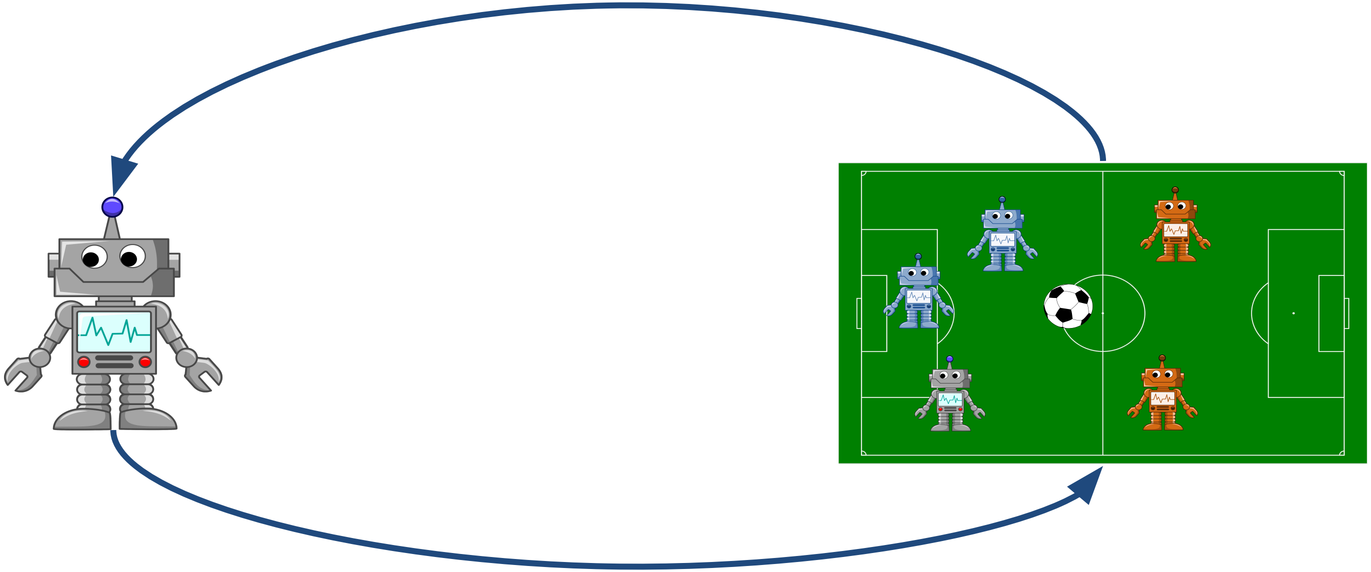
InstaDeep™

Reinforcement Learning (RL) Loop

$s \in S$ Receive state from possible states.



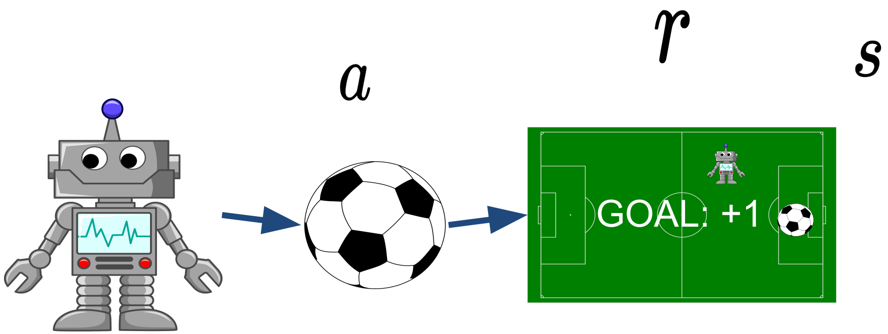
r Receive reward.



$a \in A$ Take action from available actions.

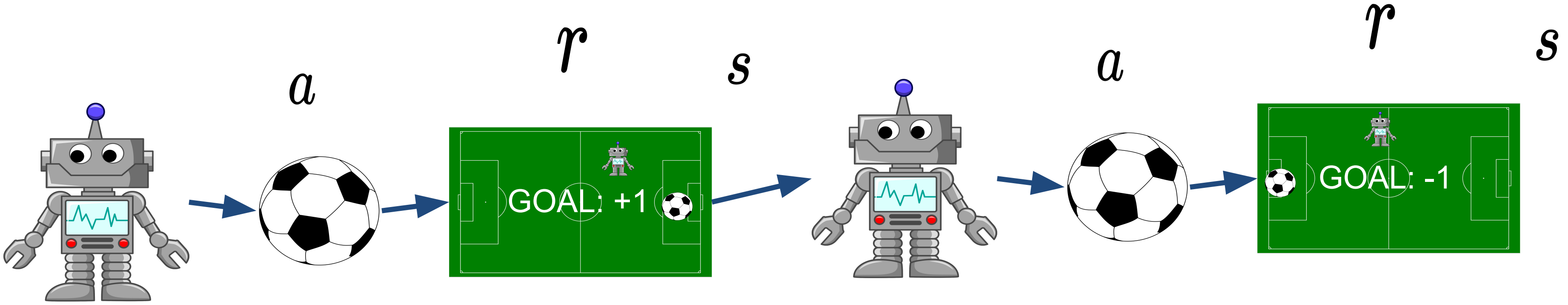
InstaDeep™

Reinforcement Learning (RL) Loop



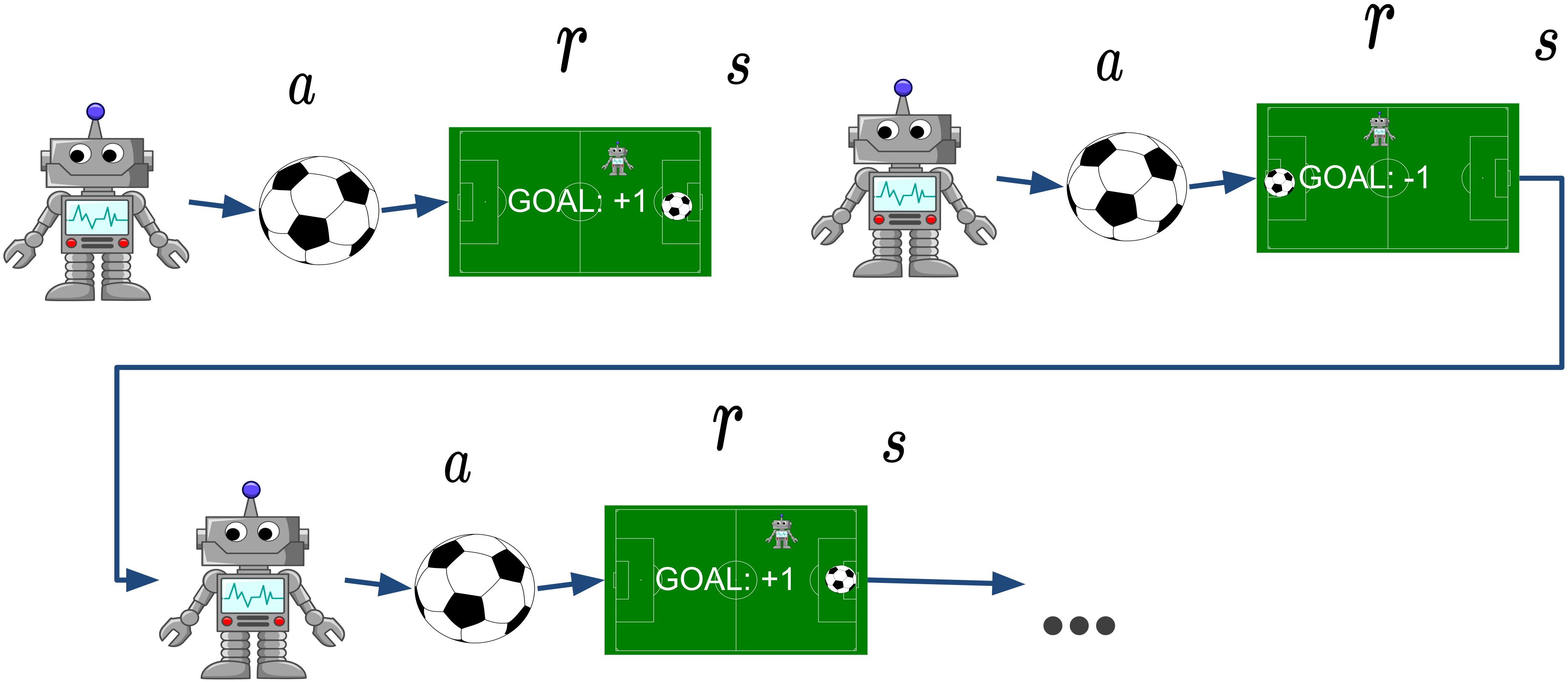
InstaDeep™

Reinforcement Learning (RL) Loop



InstaDeep™

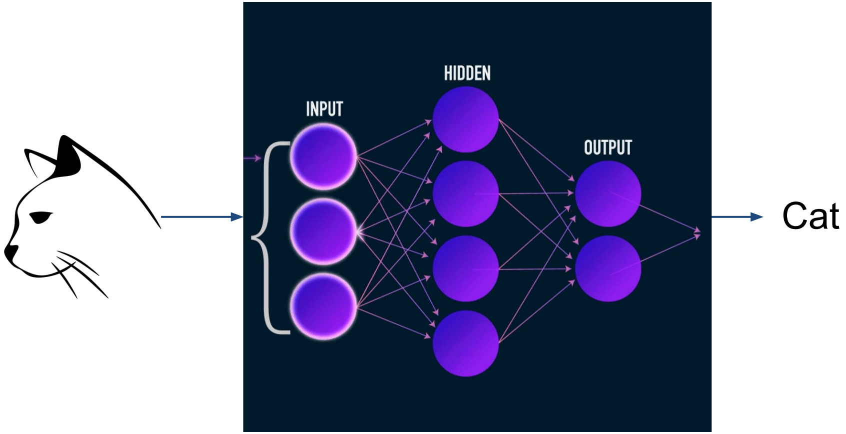
Reinforcement Learning (RL) Loop



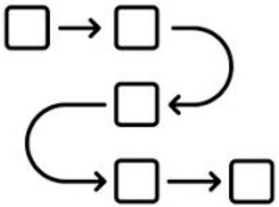
InstaDeep™

RL compared to Supervised Learning (SL) - Decisions

- SL - One-shot

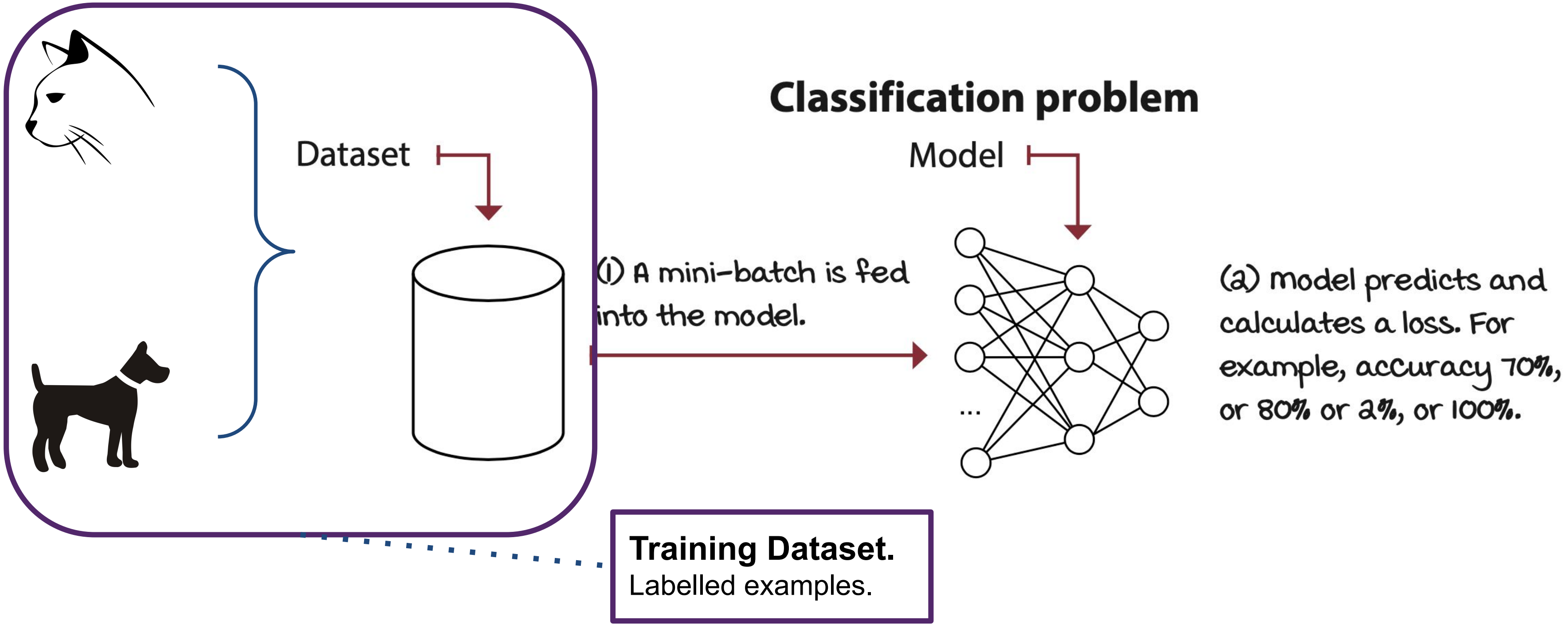


- RL - Sequential



RL compared to Supervised Learning - Training

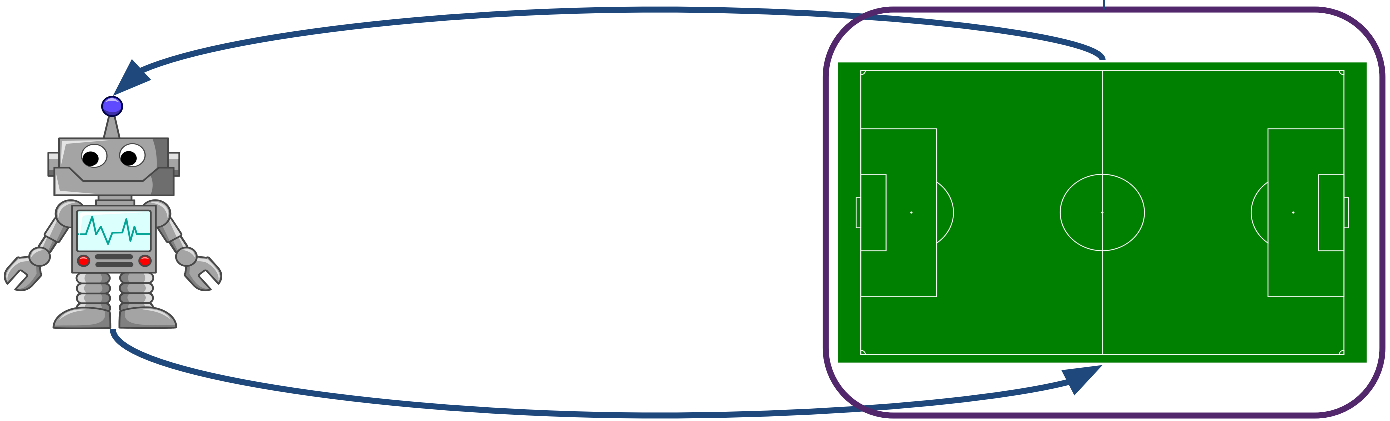
- SL - Learn from labelled examples.



RL compared to Supervised Learning - Training - Trial and Error

- Learn from interacting with an environment.

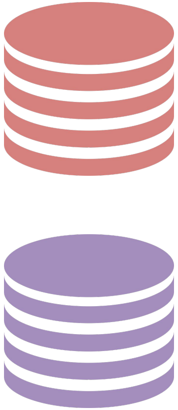
Environment/Simulator
- Reward signal.
- Possible states and actions.
- Rules or dynamics of the environment.



InstaDeep™

RL compared to Supervised Learning - Objectives

- SL - Performance on Test Set (e.g. Test Accuracy/Loss).

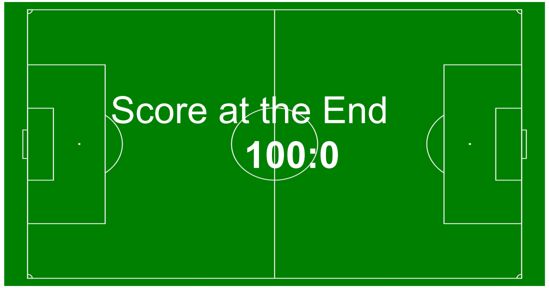


Train Data

Test Data

e.g. test accuracy 78%.

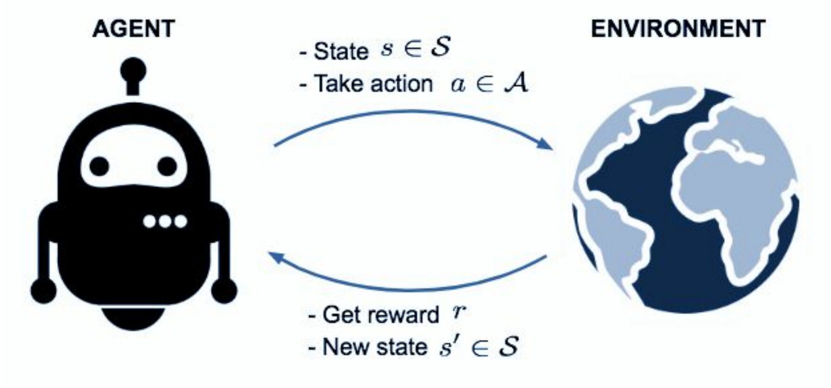
- RL - Maximize Cumulative Reward (Return).

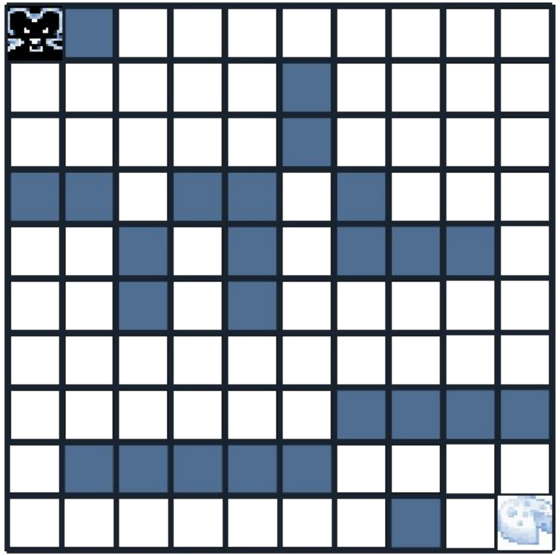


e.g. return is 100 (scored 100 goals in a match) or mean episode return is 50 (played two games - game 1 scored 75, game 2 scored 25).

Reinforcement Learning (RL)

RL agent-environment interaction loop





RL is **goal-directed learning from interaction** (trial and error).

Learn - **what to do** (*how to map situations to actions*, as to *maximize a numerical reward*).

InstaDeep™

RL Formalism

Markov Decision Process (MDP)

MDPs: Formal way to describe an RL environment (or any sequential decision-making systems).

Markov Property: Transitions only depend on the most recent state and action, and no prior history (**current state contains all necessary information**).

"|" - Given

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]$$

Probability of next state given current state = Probability of next state given whole history

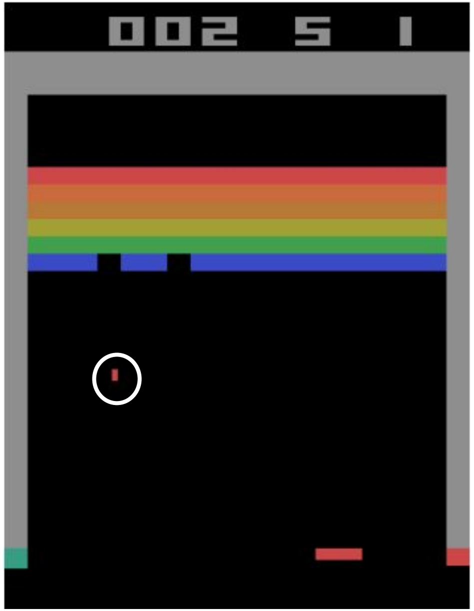
InstaDeep™

Markov Property

Do we need history for Chess?

Chess - Markovian.

Which direction is the ball going?

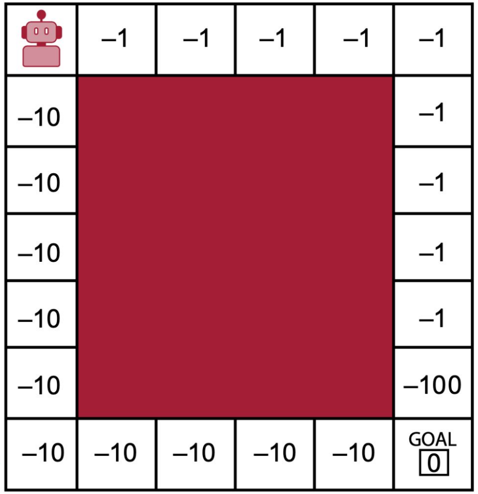


Atari Breakout - Not Markovian.

InstaDeep™

Markov Decision Process (MDP)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$$



Markov Decision Process (MDP)

$$\mathcal{M} = (\underline{\mathcal{S}}, \mathcal{A}, T, d_0, r, \gamma)$$

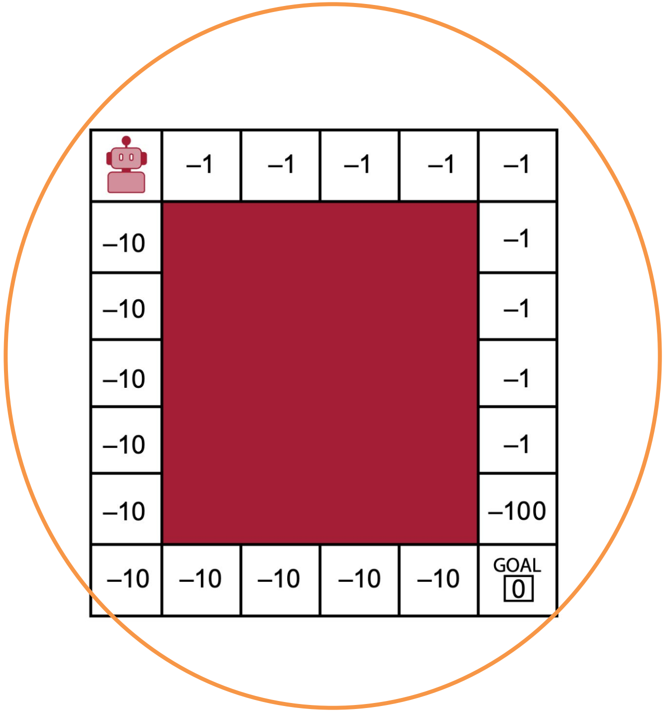
S - state space - is a finite set of states.

s ∈ **S ,** full description/representation of the environment at a particular time (discrete or continuous).

e.g.

x	0	0	0
0	B	B	0
…	…	…	…
0	0	0	T

where x - agent, T - terminal, B - blocked, 0 - open space.



InstaDeep™

Markov Decision Process (MDP)

$$\mathcal{M} = (\mathcal{S}, \underline{\mathcal{A}}, T, d_0, r, \gamma)$$

A - action space - is a finite set of actions.

a ∈ **A ,** what our agent does (discrete or continuous).

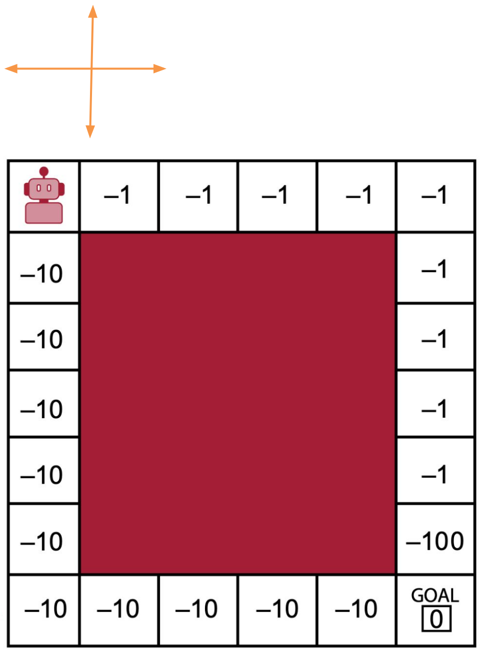
e.g.

\# 1. LEFT= 0

\# 2. DOWN = 1

\# 3. RIGHT = 2

\# 4. UP = 3

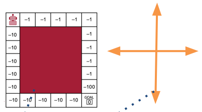


InstaDeep™

Markov Decision Process (MDP)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \underline{T}, d_0, r, \gamma)$$

T - transition probability.



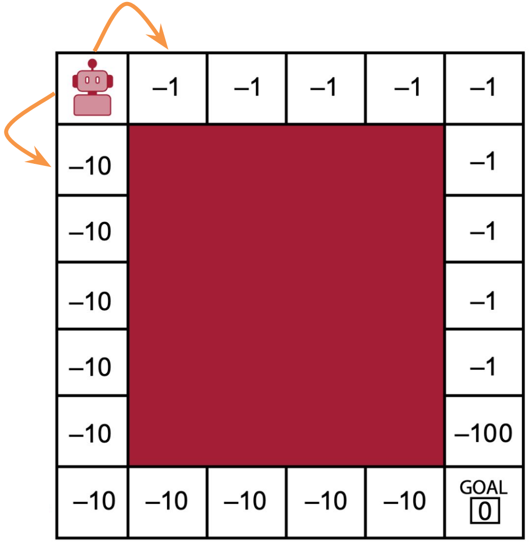
$$\mathcal{P}(s'|s,a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

Deterministic

If you decide to go left you'll go left.

Stochastic

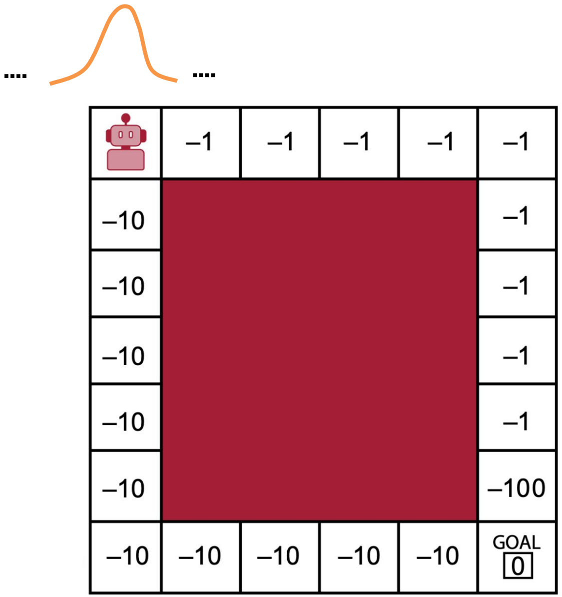
Probability distribution over transitions e.g. if you decide to go left, you will go left **50%** of the time, stay in your location **50%** of the time.



Markov Decision Process (MDP)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, \underline{d_0}, r, \gamma)$$

d0 - distribution of initial states - do you always start in the same place?



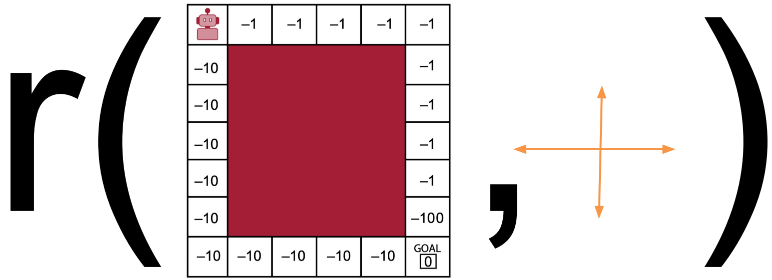
InstaDeep™

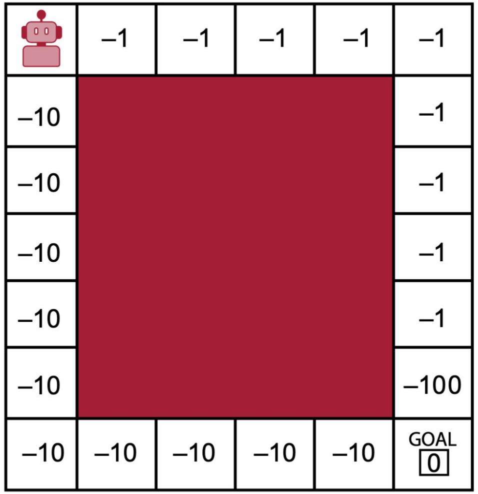
Markov Decision Process (MDP)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, \underline{r}, \gamma)$$

r - reward function - how good our current state/action was.

$$r(s_t, a_t)$$



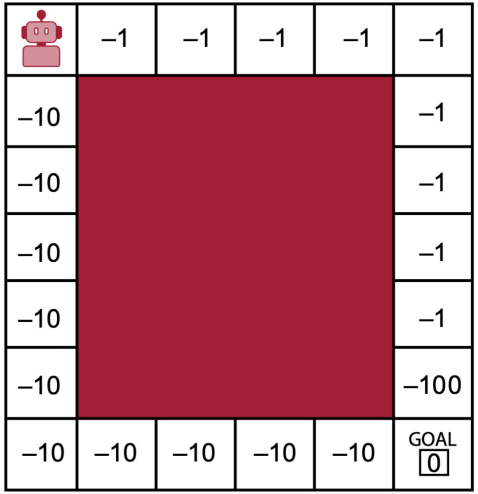


InstaDeep™

Markov Decision Process (MDP)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$$

$\gamma \in [0,1]$ is a discount factor, that penalise rewards in the future.



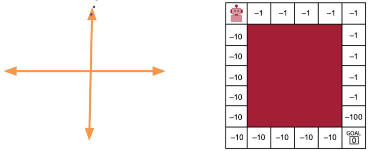
InstaDeep™

Policy - Agents - What to Do

Policy: Mapping from states to actions.

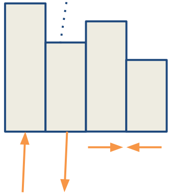
Deterministic:

$$a = \pi(s)$$



Stochastic:

$$\pi(a|s) = P[A_t = a | S_t = s]$$



$$a \sim \pi(A|s)$$

In Deep RL - policies are parameterized by the weights of Neural Network Θ:

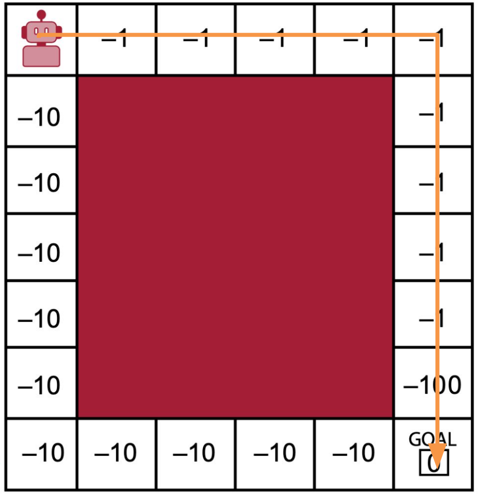
$$\pi_\theta$$

Markov Decision Process (MDP)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$$

Trajectory

$$\tau = (\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_H, \mathbf{a}_H)$$



InstaDeep™

Markov Decision Process (MDP)

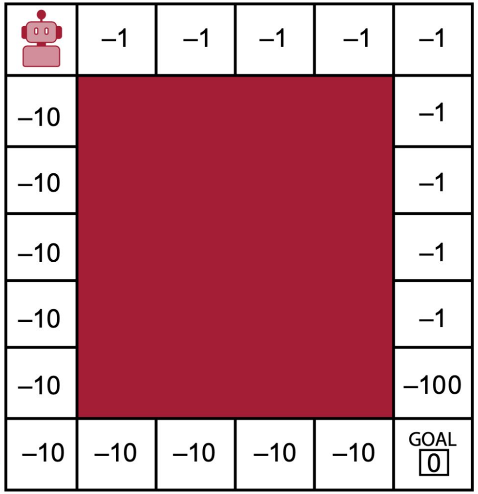
$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$$

Trajectory

$$\tau = (\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_H, \mathbf{a}_H)$$

Trajectory distribution

$$p_\pi(\tau) = d_0(\mathbf{s}_0) \prod_{t=0}^{H} \pi(\mathbf{a}_t|\mathbf{s}_t) T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$



Markov Decision Process (MDP)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$$

Trajectory

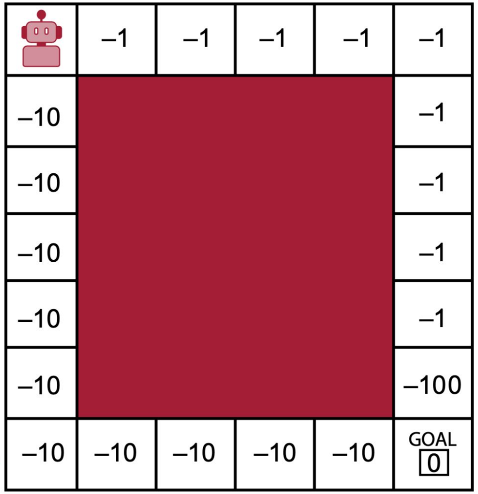
$$\tau = (\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_H, \mathbf{a}_H)$$

Trajectory distribution

Product notation.

$$p_\pi(\tau) = d_0(\mathbf{s}_0) \prod_{t=0}^{H} \underbrace{\pi(\mathbf{a}_t | \mathbf{s}_t)}_{\text{Policy}} T(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

Probability of a specific trajectory.



Return vs Reward

Reward - how good our **current state/action** is.

$$r_t = r(s_t, a_t)$$

Return - expected cumulative **reward over time**.

$$R_i(\tau) = \sum_{t=i}^{T} \gamma^t r_t$$

InstaDeep™

RL Objective

Sum of rewards weighted by their probability.

$$J(\pi) = \mathbb{E}_{\boxed{\tau \sim p_\pi(\tau)}} \boxed{[R(t)]}$$

Trajectory sampled from trajectory distribution according to policy.

Return (reward over time) following these trajectories.

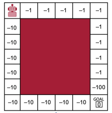
Maximise the total expected return per episode.
E.g. football - score most goals in a match or over many matches.

How our agents learns - Value

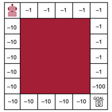
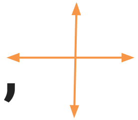
Value: What is good in the **long run**.

Value of state(s) /state-action (s,a): How good is the s or s,a pair, i.e. the expected return (G_t) if you start at s or s,a and then act according to your policy.

State-value function:



$$V^{\pi}(\mathbf{s}) = \mathbb{E}_{\tau \sim p_{\pi}(\tau|\mathbf{s})}\left[R(\tau)|s_0 = s\right]$$

Action-value (Q) function: ( , )

$$Q^{\pi}(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\tau \sim p_{\pi}(\tau|\mathbf{s},\mathbf{a})}\left[R(\tau)|s_0 = s, a_0 = a\right]$$

Efficiently estimating values is critical to RL.

Kinds of RL Algorithms (Model-free)

	Value-Based Methods	Policy-Based Methods	Actor-Critic Methods
LEARN	$Q_\theta(s,a)$	$\pi_\theta(a\|s)$	Actor $\pi_\theta(a\|s)$ Critic $Q_w(s,a)$
ACT	$a = \underset{a}{argmax}\, Q(s,a)$	$a \sim \pi_\theta(a\|s)$	$a \sim \pi_\theta(a\|s)$
E.G.	DQN.	Reinforce.	A2C/A3C, DDPG, PPO, etc.

InstaDeep™

Dynamic Programming - Bellman Equation

Value functions can be split into 2 parts:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[\underbrace{r(s,a)}_{\text{Immediate Reward.}} + \underbrace{\gamma Q^{\pi}(s_{t+1}, a_{t+1})}_{\text{Discounted value of next state.}} \mid S_t = s, A_t = a]$$

InstaDeep™

Dynamic Programming

Bellman Operator

$$\vec{Q}^{\pi} = \mathcal{B}^{\pi}\vec{Q}^{\pi}$$

$$\lim_{k\to\infty} \vec{Q}_k^{\pi} = \vec{Q}^{\pi}$$

Policy Iteration

States

Actions

Q	0	1	2	3
0	-1.5	-0.2	1.2	5.7
1	4.2	-2.1	2.7	6.1

InstaDeep™

Policy Iteration

States

Actions

Q	0	1	2	3
0	-1.5	-0.2	1.2	5.7
1	4.2	-2.1	2.7	6.1

Policy Evaluation (Prediction)

$$\vec{Q}^{\pi}_{k+1} = \mathcal{B}^{\pi} \vec{Q}^{\pi}_{k}$$

InstaDeep™

Policy Iteration

States

Actions

Q	0	1	2	3
0	-1.5	-0.2	1.2	5.7
1	4.2	-2.1	2.7	6.1

Policy Evaluation (Prediction)

$$\vec{Q}^{\pi}_{k+1} = \mathcal{B}^{\pi}\vec{Q}^{\pi}_{k}$$

Policy Improvement (Control)

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \delta(\mathbf{a}_t = \arg\max Q(\mathbf{s}_t, \mathbf{a}_t))$$

Policy Iteration

States

Actions

Q	0	1	2	3
0	-1.5	-0.2	1.2	5.7
1	4.2	-2.1	2.7	6.1

Policy Evaluation (Prediction)

$$\vec{Q}^{\pi}_{k+1} = \mathcal{B}^{\pi}\vec{Q}^{\pi}_{k}$$

Policy Improvement (Control)

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \delta(\mathbf{a}_t = \arg\max Q(\mathbf{s}_t, \mathbf{a}_t))$$

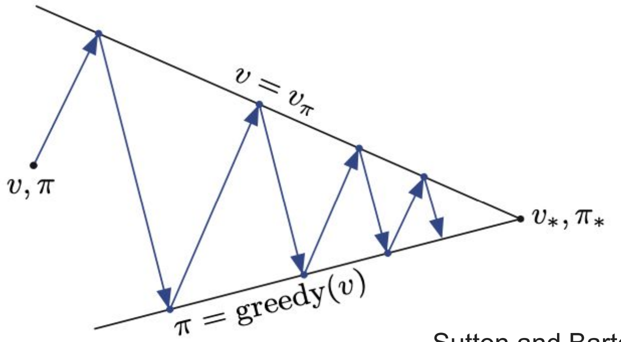
InstaDeep™

Policy Iteration

States

Actions

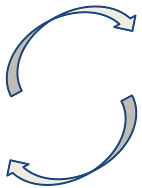
Q	0	1	2	3
0	-1.5	-0.2	1.2	5.7
1	4.2	-2.1	2.7	6.1



Sutton and Barto, 2018.

Policy Evaluation (Prediction)

$$\vec{Q}^{\pi}_{k+1} = \mathcal{B}^{\pi} \vec{Q}^{\pi}_{k}$$



Policy Improvement (Control)

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \delta(\mathbf{a}_t = \arg\max Q(\mathbf{s}_t, \mathbf{a}_t))$$

Value Iteration

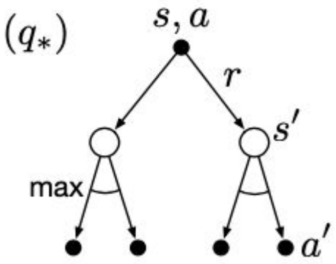
Bellman Optimality Equation

$$Q^\star(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \left[\max_{\mathbf{a}_{t+1}} Q^\star(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \right]$$

Value Iteration

Bellman Optimality Equation

$$Q^\star(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \left[\max_{\mathbf{a}_{t+1}} Q^\star(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \right]$$



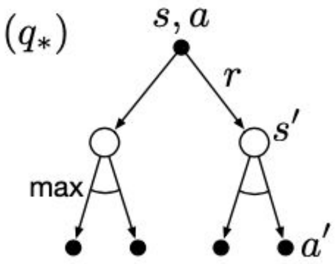
InstaDeep™

Value Iteration

Bellman Optimality Equation

$$Q^\star(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \left[\max_{\mathbf{a}_{t+1}} Q^\star(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \right]$$

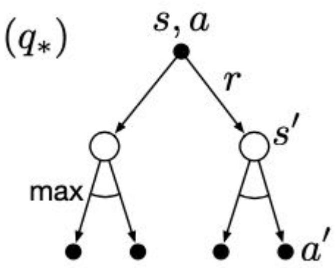
Need a model



Value Iteration

Bellman Optimality Equation

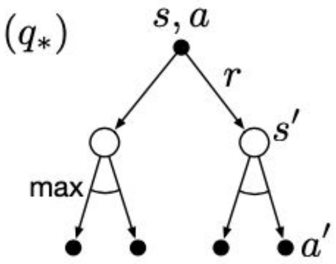
$$Q^\star(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim T(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} \left[\max_{\mathbf{a}_{t+1}} Q^\star(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \right]$$



Value Iteration -> Temporal Difference (TD) learning

Bellman Optimality Equation

$$Q^\star(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \left[\max_{\mathbf{a}_{t+1}} Q^\star(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \right]$$



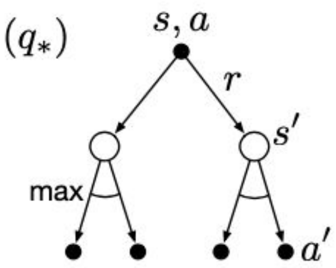
Incrementally estimate
using samples

$$NewEstimate \leftarrow OldEstimate \; + \; StepSize \left[Target - OldEstimate \right]$$

Value Iteration -> Temporal Difference (TD) learning

Bellman Optimality Equation

$$Q^\star(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \left[\max_{\mathbf{a}_{t+1}} Q^\star(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \right]$$



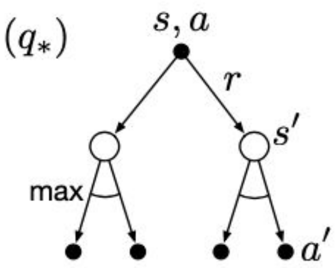
Incrementally estimate
using samples

$$NewEstimate \leftarrow OldEstimate + StepSize \left[Target - OldEstimate \right]$$

Value Iteration -> Temporal Difference (TD) learning

Bellman Optimality Equation

$$\underline{Q^\star(\mathbf{s}_t, \mathbf{a}_t)} = \underline{r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim T(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} \left[\max_{\mathbf{a}_{t+1}} Q^\star(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \right]}$$



Incrementally estimate
using samples

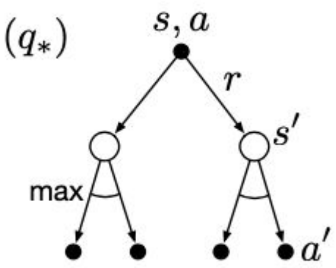
$$NewEstimate \leftarrow \underline{OldEstimate} + StepSize \left[\underline{Target} - \underline{OldEstimate} \right]$$

InstaDeep™

Value Iteration -> Temporal Difference (TD) learning

Bellman Optimality Equation

$$\underline{Q^\star(\mathbf{s}_t, \mathbf{a}_t)} = \underline{r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \left[\max_{\mathbf{a}_{t+1}} Q^\star(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \right]}$$



Incrementally estimate using samples

$$NewEstimate \leftarrow \underline{OldEstimate} + StepSize \underbrace{\left[\underline{Target} - \underline{OldEstimate} \right]}_{\textit{TD Error}}$$

InstaDeep™

Q-Learning (Off-policy TD Learning)

Update the value estimates in part based on other estimates: “Learning a guess from a guess”.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{Old Estimate.}} + \underbrace{\alpha}_{\text{Step Size.}} \underbrace{\Big(\underbrace{r(s_t, a_t) + \gamma \max_a Q\left(s_{t+1}, a\right)}_{\text{Target}} - \underbrace{Q(s_t, a_t)}_{\text{Old Estimate.}} \Big)}_{\textit{TD Error}}$$

Q-Learning (Off-policy TD Learning)

Game Board:

Q Table: $\gamma = 0.95$

	0 0 0 1 0 0	0 0 0 0 1 0	0 0 0 0 0 1	1 0 0 0 0 0	0 1 0 0 0 0	0 0 1 0 0 0
↑						
↓						
→						
←						

[Link](#)

Q-Learning (Off-policy TD Learning)

Game Board:

Current state (s): 0 0 0
0 1 0

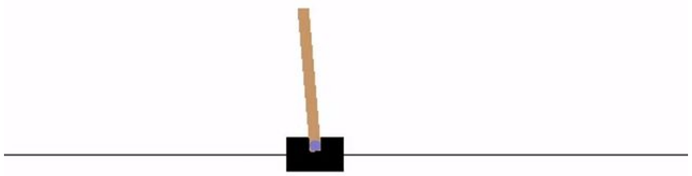
Q Table: $\gamma = 0.95$

	0 0 0 1 0 0	0 0 0 0 1 0	0 0 0 0 0 1	1 0 0 0 0 0	0 1 0 0 0 0	0 0 1 0 0 0
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

[Link]()

InstaDeep™

Q-learning in large state spaces?



Its state space is comprised of four variables:

- The cart position on the track (x-axis) with a range from –2.4 to 2.4
- The cart velocity along the track (x-axis) with a range from –inf to inf
- The pole angle with a range of ~–40 degrees to ~ 40 degrees
- The pole velocity at the tip with a range of –inf to inf

Tabular RL does **not** scale to large complex problems:

- Too many states to store in memory
- Too slow to update and estimate values for each state

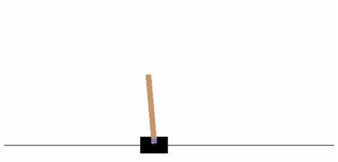
Need to use an approach able to **generalise** across many states

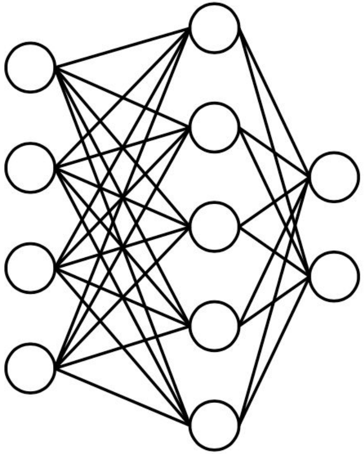
Approx. Dynamic Programming

using function approximation

The goal of function approximation

Approximate the values of states using a **parameterised function**

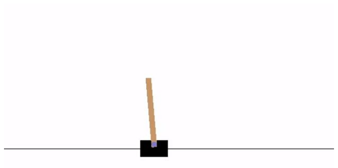


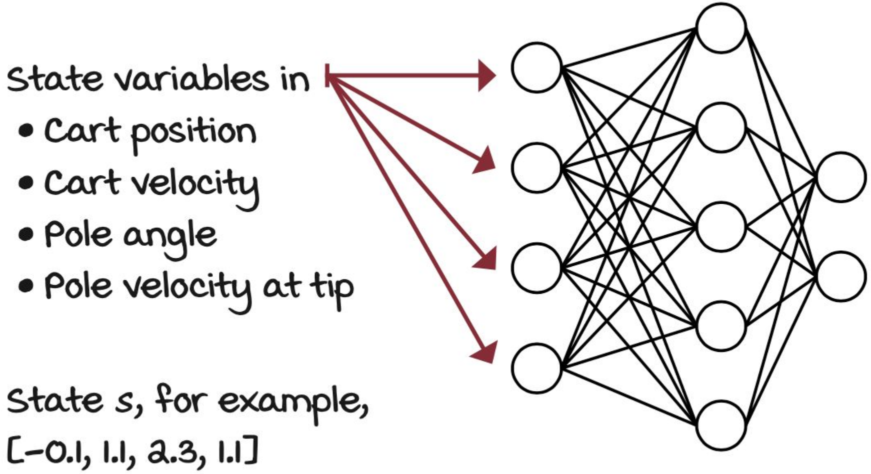


The goal of function approximation

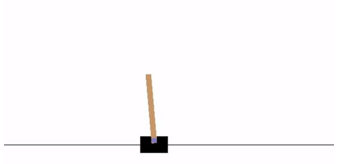
Approximate the values of states using a **parameterised function**

- **Input**: state features



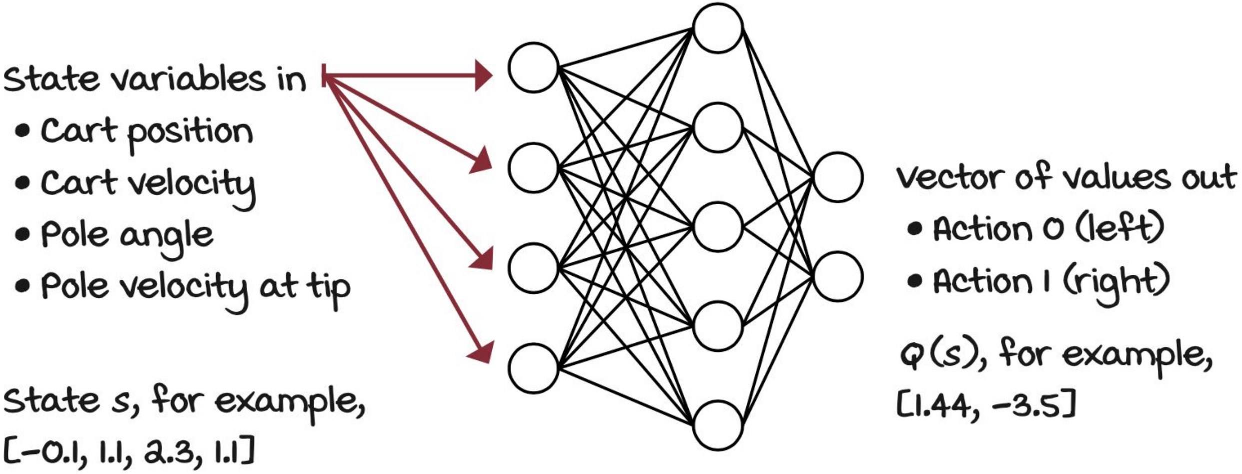


The goal of function approximation



Approximate the values of states using a **parameterised function**

- **Input**: state features
- **Output**: estimated Q-values

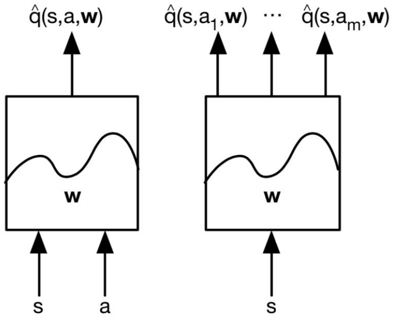


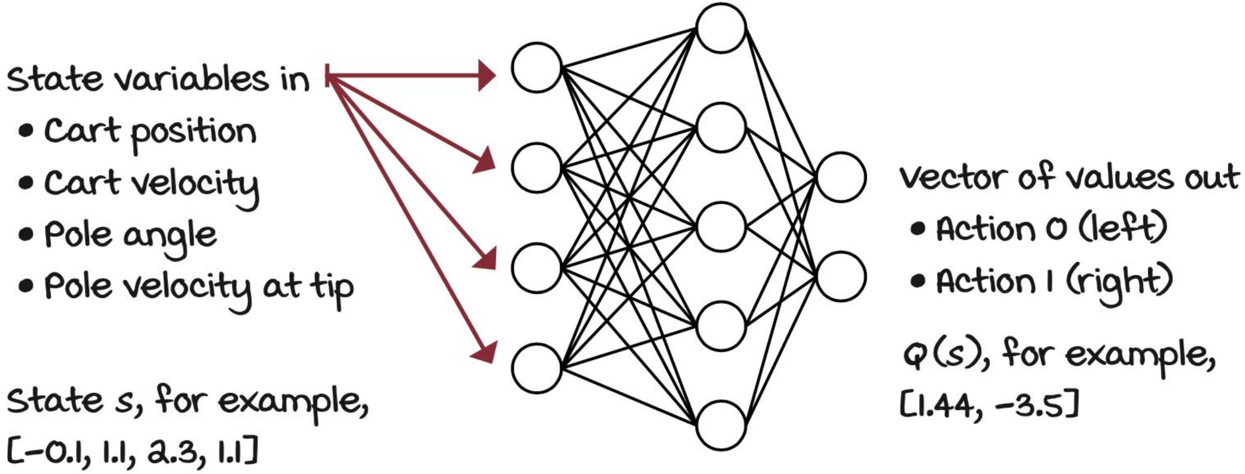
InstaDeep™

The goal of function approximation

Approximate the values of states using a **parameterised function**

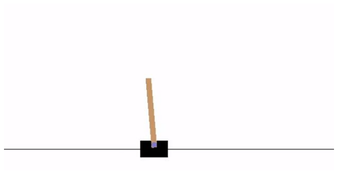
- **Input**: state features
- **Output**: estimated Q-values





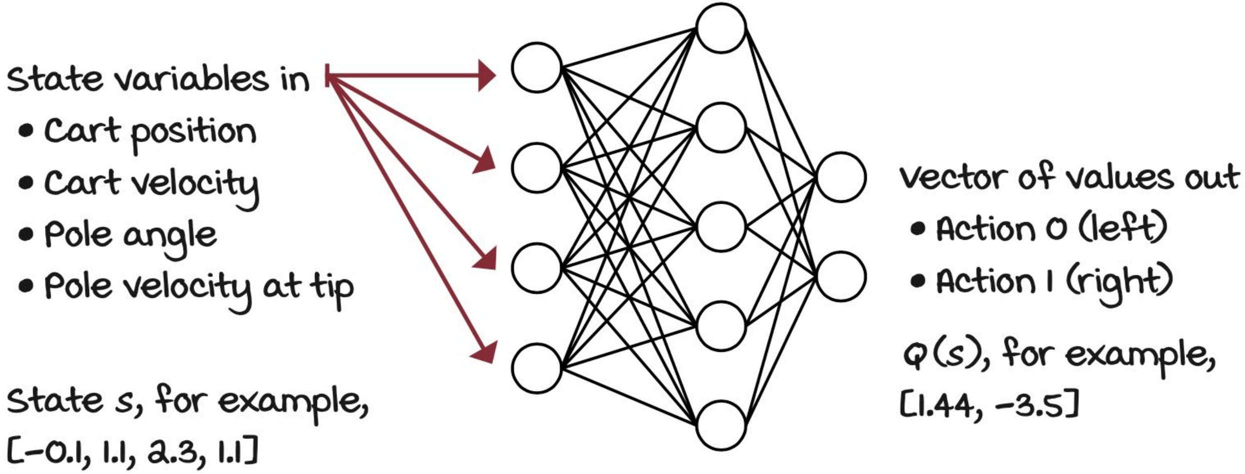
InstaDeep™

The goal of function approximation



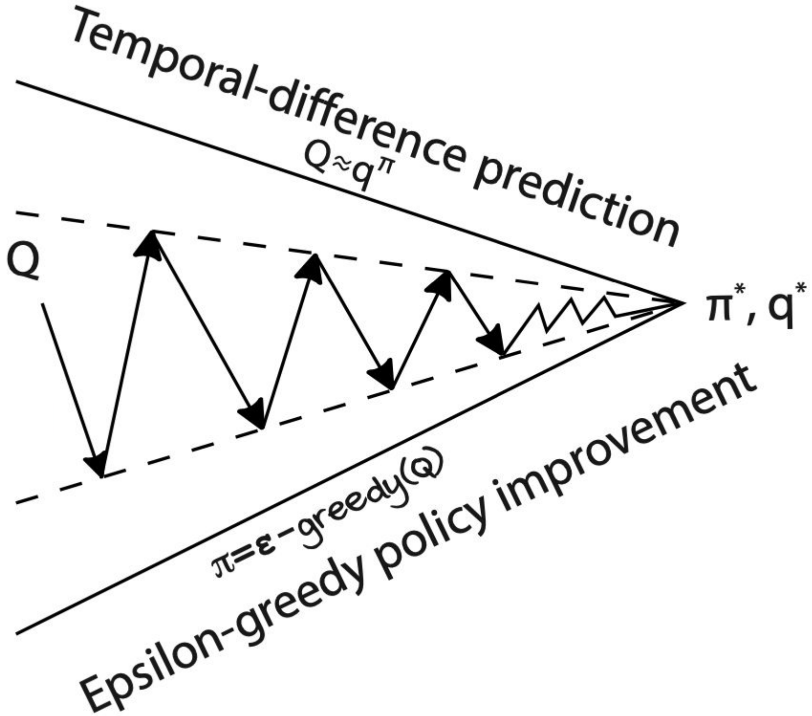
Approximate the values of states using a **parameterised function**

- **Input**: state features
- **Output**: estimated Q-values
- **Target**: reward to go



InstaDeep™

Approximate Dynamic Programming

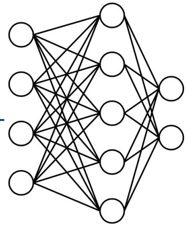


Approximate Dynamic Programming

$$Q_\phi$$

Approximate Dynamic Programming

$$Q_\phi$$

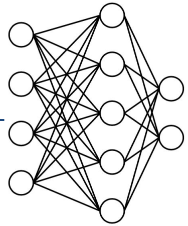


Typically a deep neural network

Approximate Dynamic Programming

$$Q_\phi$$

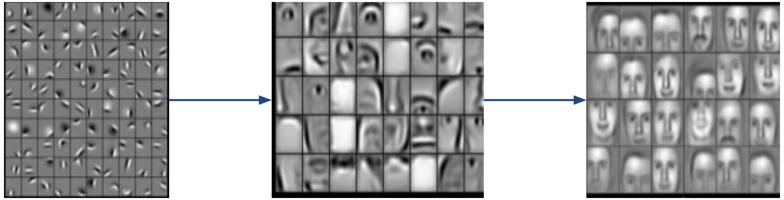
Why?



Typically a deep
neural network

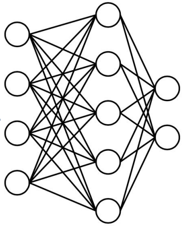
Approximate Dynamic Programming

- Known to discover useful features



$$Q_\phi$$

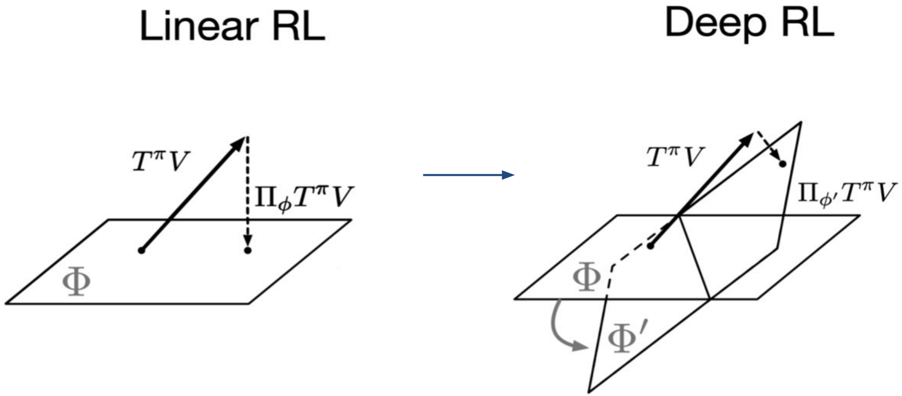
Why?



Typically a deep neural network

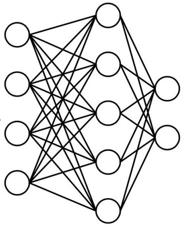
Approximate Dynamic Programming

- Known to discover useful features



Q_ϕ

Why?

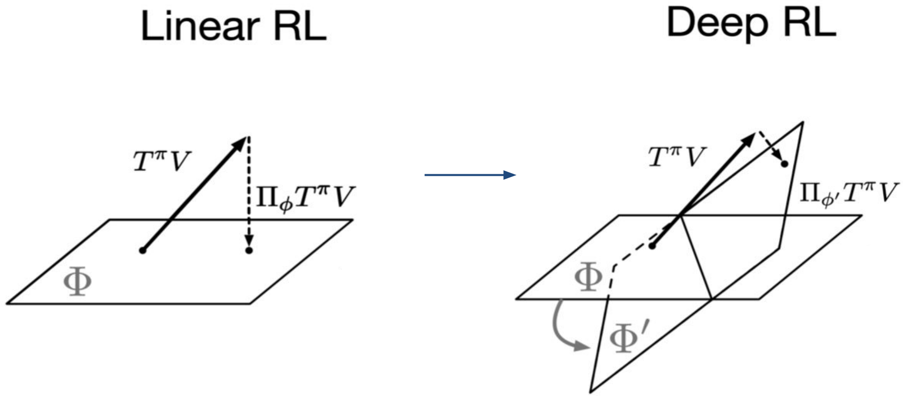


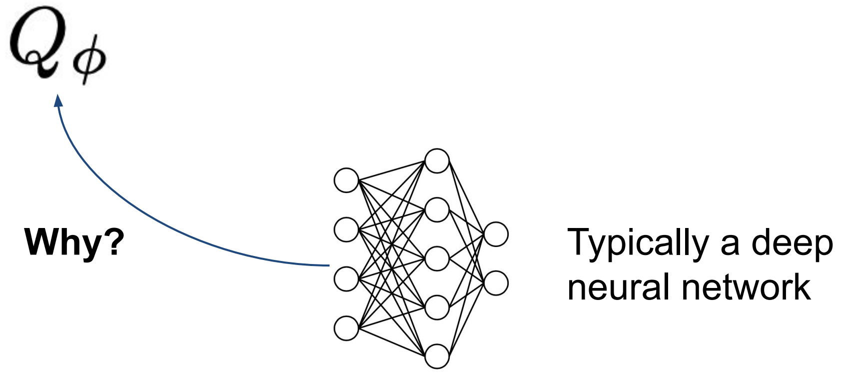
Typically a deep neural network

InstaDeep™

Approximate Dynamic Programming

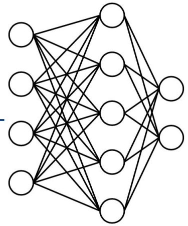
- Known to discover useful features
- Wealth of research in DL that can be directly be applied to RL





Approximate Dynamic Programming

$$Q_\phi$$



Typically a deep neural network

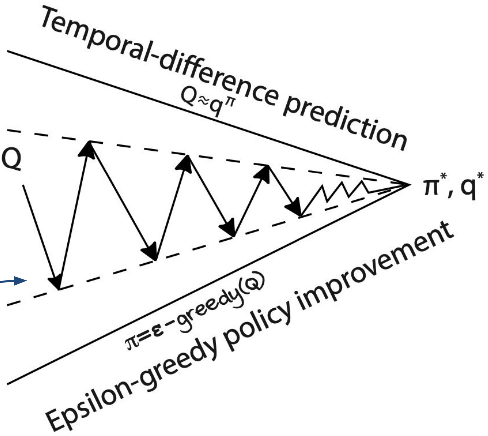
Approximate Dynamic Programming

$$Q_{\phi_k}$$

InstaDeep™

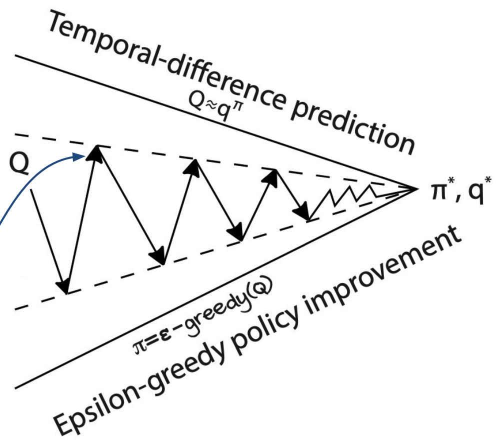
Approximate Dynamic Programming

$$Q_{\phi_k}$$



Controls policy
improvement step

Approximate Dynamic Programming



$$Q_{\phi_{k,g}}$$

Controls policy
evaluation step

Approximate Dynamic Programming

$$Q_{\phi_{k,g}}$$

Approximate Dynamic Programming

$$Q_{\phi_{k,g}} \qquad (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))$$

Approximate Dynamic Programming

Prediction

Target

$$Q_{\phi_{k,g}} \qquad (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))$$

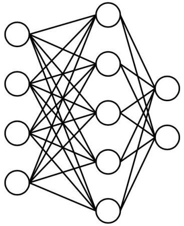
Approximate Dynamic Programming

Prediction

Target

$$Q_{\phi_{k,g}} \qquad (r_i + \gamma \max\nolimits_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))$$

Target network
note parameters do not depend on g



Approximate Dynamic Programming

Prediction

Target

$$Q_{\phi_{k,g}} \qquad (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))$$

Approximate Dynamic Programming

Prediction

Target

$$Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))$$

Approximate Dynamic Programming

$$\underbrace{\overset{\text{Prediction}}{Q_{\phi_{k,g}}} - (\overset{\text{Target}}{r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')})}_{\text{TD Error}}$$

InstaDeep™

Approximate Dynamic Programming

$$\sum_i \underbrace{\Big(\overbrace{Q_{\phi_{k,g}}}^{\text{Prediction}} - \overbrace{(r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}',\mathbf{a}'))}^{\text{Target}}\Big)}_{\text{TD Error}}{}^2$$

InstaDeep™

Approximate Dynamic Programming

Squared-error loss as in *supervised learning*

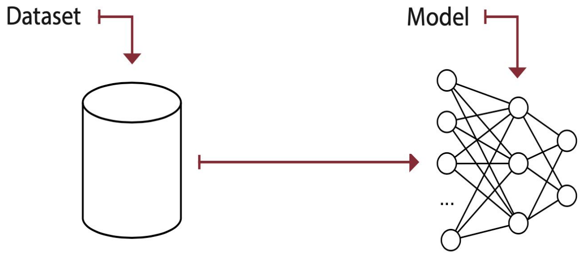
Prediction

Target

$$\underbrace{\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2}_{\text{TD Error}}$$

InstaDeep™

Approximate Dynamic Programming



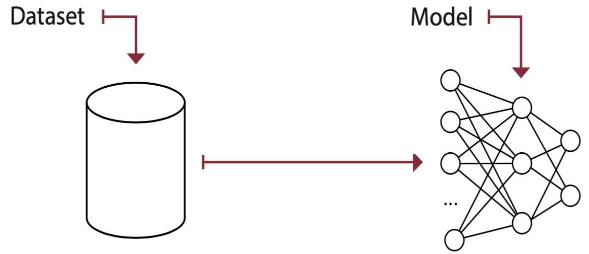
Squared-error loss as in *supervised learning*

Prediction

Target

$$\sum_i \underbrace{\left(Q_{\phi_{k,g}} - \left(r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')\right)\right)}_{\text{TD Error}}^2$$

Approximate Dynamic Programming



Squared-error loss as in *supervised learning*

Prediction

Target

$$\sum_i \Big(\underbrace{Q_{\phi_{k,g}} - \big(r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')\big)}_{\text{TD Error}}\Big)^2$$

Sum over
transition data

$$B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$$

Approximate Dynamic Programming

$$\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$$

Approximate Dynamic Programming

$$\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$$

$$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$$

InstaDeep™

Approximate Dynamic Programming

$$\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$$

$$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$$

Update the parameters using **gradient descent**

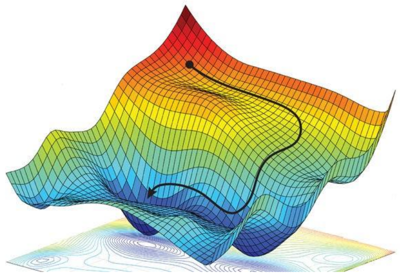
Approximate Dynamic Programming

$$\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$$

$$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \overbrace{\mathcal{E}(B, \phi_{k,g})}$$

Update the parameters using **gradient descent**

Approximate Dynamic Programming



$$\sum_i \left(Q_{\phi_{k,g}} - \left(r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')\right)\right)^2$$

$$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \overbrace{\mathcal{E}(B, \phi_{k,g})}$$

Update the parameters using **gradient descent**

InstaDeep™

Approximate Dynamic Programming

$$\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$$

$$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$$

Approximate Dynamic Programming

for gradient step $g \in [0, \ldots, G-1]$ **do**

sample batch B

$$\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$$

$$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$$

end for

Approximate Dynamic Programming

for gradient step $g \in [0, \ldots, G-1]$ **do**

sample batch B

$$\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$$

$$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$$

end for

Approximate
Policy Evaluation

Approximate Dynamic Programming

for gradient step $g \in [0, \ldots, G-1]$ **do**
    sample batch B
    $\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$
    $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$
end for

Approximate
Policy Evaluation

$$\phi_{k+1} \leftarrow \phi_{k,G}$$

Approximate Dynamic Programming

for gradient step $g \in [0, \ldots, G-1]$ **do**

sample batch B

$\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$

$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$

end for

Approximate
Policy Evaluation

$$\phi_{k+1} \leftarrow \phi_{k,G}$$

Act (ε) greedy with respect to new parameters

Approximate Dynamic Programming

$$
\begin{aligned}
&\textbf{for } \text{gradient step } g \in [0, \ldots, G-1] \textbf{ do} \\
&\quad \text{sample batch } B \\
&\quad \textstyle\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2 \\
&\quad \phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g}) \\
&\textbf{end for}
\end{aligned}
$$

Approximate
Policy Evaluation

$$\phi_{k+1} \leftarrow \phi_{k,G}$$

Act (ε) greedy with respect to new parameters

Approximate
Policy Improvement

InstaDeep™

Approximate Dynamic Programming

for gradient step $g \in [0, \ldots, G-1]$ **do**

sample batch B

$$\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$$

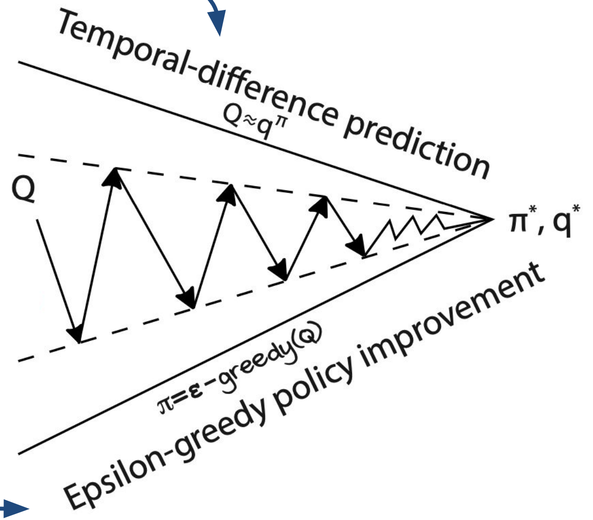
$$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$$

end for

$$\phi_{k+1} \leftarrow \phi_{k,G}$$

Act (ε) greedy with respect to new parameters

Approximate **Generalised Policy Iteration**



InstaDeep™

Generic Q-learning algorithm

1: initialize ϕ_0

Generic Q-learning algorithm

1: initialize ϕ_0
2: initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ ▷ Use ϵ-greedy exploration

Generic Q-learning algorithm

1: initialize ϕ_0
2: initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ ▷ Use ϵ-greedy exploration
3: initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size

Generic Q-learning algorithm

1: initialize ϕ_0
2: initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ ▷ Use ϵ-greedy exploration
3: initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
4: initialize $\mathbf{s} \sim d_0(\mathbf{s})$

Generic Q-learning algorithm

1: initialize ϕ_0
2: initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ ▷ Use ϵ-greedy exploration
3: initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
4: initialize $\mathbf{s} \sim d_0(\mathbf{s})$
5: **for** iteration $k \in [0, \ldots, K]$ **do**

InstaDeep™

Generic Q-learning algorithm

1: initialize ϕ_0
2: initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ ▷ Use ϵ-greedy exploration
3: initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
4: initialize $\mathbf{s} \sim d_0(\mathbf{s})$
5: **for** iteration $k \in [0, \ldots, K]$ **do**
6:     **for** step $s \in [0, \ldots, S-1]$ **do**

InstaDeep™

Generic Q-learning algorithm

1: initialize ϕ_0
2: initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ ▷ Use ϵ-greedy exploration
3: initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
4: initialize $\mathbf{s} \sim d_0(\mathbf{s})$
5: **for** iteration $k \in [0, \ldots, K]$ **do**
6:     **for** step $s \in [0, \ldots, S-1]$ **do**
7:         $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ ▷ sample action from exploration policy

InstaDeep™

Generic Q-learning algorithm

initialize ϕ_0

initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s},\mathbf{a}))$ $\triangleright$ Use ϵ-greedy exploration

initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size

initialize $\mathbf{s} \sim d_0(\mathbf{s})$

for iteration $k \in [0,\ldots,K]$ **do**

  for step $s \in [0,\ldots,S-1]$ **do**

    $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ $\triangleright$ sample action from exploration policy

    $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})$ $\triangleright$ sample next state from MDP

Generic Q-learning algorithm

initialize ϕ_0
initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ ▷ Use ϵ-greedy exploration
initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
initialize $\mathbf{s} \sim d_0(\mathbf{s})$
for iteration $k \in [0, \ldots, K]$ **do**
    for step $s \in [0, \ldots, S-1]$ **do**
        $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ ▷ sample action from exploration policy
        $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ ▷ sample next state from MDP
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$ ▷ append to buffer, purging old data if buffer too big

Generic Q-learning algorithm

initialize ϕ_0

initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s},\mathbf{a}))$ $\triangleright$ Use ϵ-greedy exploration

initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size

initialize $\mathbf{s} \sim d_0(\mathbf{s})$

for iteration $k \in [0, \ldots, K]$ **do**

    for step $s \in [0, \ldots, S-1]$ **do**

        $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ $\triangleright$ sample action from exploration policy

        $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})$ $\triangleright$ sample next state from MDP

        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s},\mathbf{a},\mathbf{s}',r(\mathbf{s},\mathbf{a}))\}$ $\triangleright$ append to buffer, purging old data if buffer too big

    end for

Generic Q-learning algorithm

initialize ϕ_0

initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ ▷ Use ϵ-greedy exploration

initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size

initialize $\mathbf{s} \sim d_0(\mathbf{s})$

for iteration $k \in [0, \ldots, K]$ **do**

    for step $s \in [0, \ldots, S-1]$ **do**

        $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ ▷ sample action from exploration policy

        $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ ▷ sample next state from MDP

        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$ ▷ append to buffer, purging old data if buffer too big

    end for

    $\phi_{k,0} \leftarrow \phi_k$

Generic Q-learning algorithm

initialize ϕ_0

initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s},\mathbf{a}))$ ▷ Use ϵ-greedy exploration

initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size

initialize $\mathbf{s} \sim d_0(\mathbf{s})$

for iteration $k \in [0, \ldots, K]$ **do**

  for step $s \in [0, \ldots, S-1]$ **do**

    $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ ▷ sample action from exploration policy

    $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})$ ▷ sample next state from MDP

    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s},\mathbf{a},\mathbf{s}',r(\mathbf{s},\mathbf{a}))\}$ ▷ append to buffer, purging old data if buffer too big

  end for

  $\phi_{k,0} \leftarrow \phi_k$

  for gradient step $g \in [0, \ldots, G-1]$ **do**

Generic Q-learning algorithm

initialize ϕ_0
initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ $\triangleright$ Use ϵ-greedy exploration
initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
initialize $\mathbf{s} \sim d_0(\mathbf{s})$
for iteration $k \in [0, \ldots, K]$ **do**
  for step $s \in [0, \ldots, S-1]$ **do**
    $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ $\triangleright$ sample action from exploration policy
    $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ $\triangleright$ sample next state from MDP
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$ $\triangleright$ append to buffer, purging old data if buffer too big
  end for
  $\phi_{k,0} \leftarrow \phi_k$
  for gradient step $g \in [0, \ldots, G-1]$ **do**
    sample batch $B \subset \mathcal{D}$ $\triangleright$ $B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$

Generic Q-learning algorithm

initialize ϕ_0
initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ $\triangleright$ Use ϵ-greedy exploration
initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
initialize $\mathbf{s} \sim d_0(\mathbf{s})$
for iteration $k \in [0, \ldots, K]$ **do**
  for step $s \in [0, \ldots, S-1]$ **do**
    $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ $\triangleright$ sample action from exploration policy
    $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ $\triangleright$ sample next state from MDP
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$ $\triangleright$ append to buffer, purging old data if buffer too big
  end for
  $\phi_{k,0} \leftarrow \phi_k$
  for gradient step $g \in [0, \ldots, G-1]$ **do**
    sample batch $B \subset \mathcal{D}$ $\triangleright$ $B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$
    estimate error $\mathcal{E}(B, \phi_{k,g}) = \sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$

Generic Q-learning algorithm

initialize ϕ_0

initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s},\mathbf{a}))$ $\triangleright$ Use ϵ-greedy exploration

initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size

initialize $\mathbf{s} \sim d_0(\mathbf{s})$

for iteration $k \in [0, \ldots, K]$ **do**

  for step $s \in [0, \ldots, S-1]$ **do**

    $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ $\triangleright$ sample action from exploration policy

    $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})$ $\triangleright$ sample next state from MDP

    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s},\mathbf{a},\mathbf{s}', r(\mathbf{s},\mathbf{a}))\}$ $\triangleright$ append to buffer, purging old data if buffer too big

  end for

  $\phi_{k,0} \leftarrow \phi_k$

  for gradient step $g \in [0, \ldots, G-1]$ **do**

    sample batch $B \subset \mathcal{D}$ $\triangleright$ $B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$

    estimate error $\mathcal{E}(B, \phi_{k,g}) = \sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$

    update parameters: $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha\nabla_{\phi_{k,g}}\mathcal{E}(B, \phi_{k,g})$

Generic Q-learning algorithm

initialize ϕ_0
initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s},\mathbf{a}))$ ▷ Use ϵ-greedy exploration
initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
initialize $\mathbf{s} \sim d_0(\mathbf{s})$
for iteration $k \in [0, \ldots, K]$ **do**
  for step $s \in [0, \ldots, S-1]$ **do**
    $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ ▷ sample action from exploration policy
    $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})$ ▷ sample next state from MDP
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s},\mathbf{a},\mathbf{s}',r(\mathbf{s},\mathbf{a}))\}$ ▷ append to buffer, purging old data if buffer too big
  end for
  $\phi_{k,0} \leftarrow \phi_k$
  for gradient step $g \in [0, \ldots, G-1]$ **do**
    sample batch $B \subset \mathcal{D}$ ▷ $B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$
    estimate error $\mathcal{E}(B, \phi_{k,g}) = \sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}',\mathbf{a}'))\right)^2$
    update parameters: $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha\nabla_{\phi_{k,g}}\mathcal{E}(B,\phi_{k,g})$
  end for

Generic Q-learning algorithm

initialize ϕ_0

initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ ▷ Use ϵ-greedy exploration

initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size

initialize $\mathbf{s} \sim d_0(\mathbf{s})$

for iteration $k \in [0, \ldots, K]$ **do**

    for step $s \in [0, \ldots, S-1]$ **do**

        $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ ▷ sample action from exploration policy

        $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ ▷ sample next state from MDP

        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$ ▷ append to buffer, purging old data if buffer too big

    end for

    $\phi_{k,0} \leftarrow \phi_k$

    for gradient step $g \in [0, \ldots, G-1]$ **do**

        sample batch $B \subset \mathcal{D}$ ▷ $B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$

        estimate error $\mathcal{E}(B, \phi_{k,g}) = \sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))\right)^2$

        update parameters: $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$

    end for

    $\phi_{k+1} \leftarrow \phi_{k,G}$ ▷ update parameters

Generic Q-learning algorithm

initialize ϕ_0
initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1-\epsilon)\delta(\mathbf{a} = \arg\max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s},\mathbf{a}))$ $\triangleright$ Use ϵ-greedy exploration
initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
initialize $\mathbf{s} \sim d_0(\mathbf{s})$
for iteration $k \in [0,\ldots,K]$ **do**
    for step $s \in [0,\ldots,S-1]$ **do**
        $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ $\triangleright$ sample action from exploration policy
        $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})$ $\triangleright$ sample next state from MDP
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s},\mathbf{a},\mathbf{s}',r(\mathbf{s},\mathbf{a}))\}$ $\triangleright$ append to buffer, purging old data if buffer too big
    end for
    $\phi_{k,0} \leftarrow \phi_k$
    for gradient step $g \in [0,\ldots,G-1]$ **do**
        sample batch $B \subset \mathcal{D}$ $\triangleright$ $B = \{(\mathbf{s}_i,\mathbf{a}_i,\mathbf{s}'_i,r_t)\}$
        estimate error $\mathcal{E}(B,\phi_{k,g}) = \sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}',\mathbf{a}'))\right)^2$
        update parameters: $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha\nabla_{\phi_{k,g}}\mathcal{E}(B,\phi_{k,g})$
    end for
    $\phi_{k+1} \leftarrow \phi_{k,G}$ $\triangleright$ update parameters
end for

Deep Q-Networks

as special case of generic Q-learning

Deep Q-Networks

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

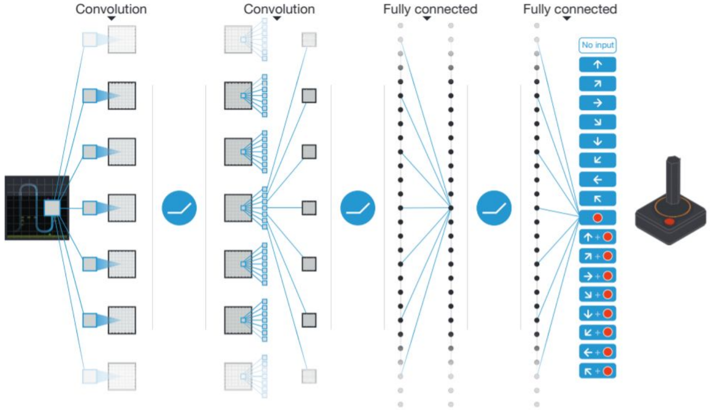
We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

LETTER

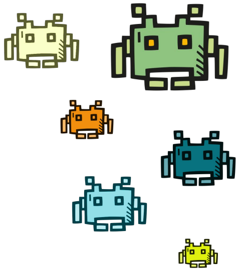
doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]



Stability Issues with Deep Q-Networks



Naive Q-Learning oscillates or diverges with neural networks:

- ❏ Data is sequential:
  Successive sample are correlated, non-iid.
- ❏ Policy changes rapidly with slight changes to Q-values
- ❏ Scale of rewards and Q-values is unknown
  Naive Q-learning gradients can be large and unstable when backpropagated.
- ❏ Exploration is greedy

DQN provides a stable solution to deep value-based RL:

- ❏ Use experience replay
  Break correlations in data, bring us back to iid setting
  Learn from all past policies
- ❏ Freeze target Q-network
  Avoid oscillations
  Break correlations between Q-network and target
- ❏ Clip rewards or normalize network adaptively to sensible range
  Robust gradients
- ❏ Use Epsilon Greedy Exploration

nature Human-level control through deep reinforcement learning, *V. Mnih et al, nature 2015*

InstaDeep™

Special cases of the generic Q-learning algorithm

Classic Q-learning (Watkins and Dayan, 1992)

$$\text{buffer size} = 1$$

$$S = 1$$

$$G = 1$$

Special cases of the generic Q-learning algorithm

Classic Q-learning (Watkins and Dayan, 1992)

$$\text{buffer size} = 1$$

$$S = 1$$

$$G = 1$$

Fitted Q-iteration (Ernst et al., 2005, Riedmiller et al., 2005)

$\text{buffer size} = S$ (sampling size is a hyperparameter)

$G = \infty$ (until convergence)

Special cases of the generic Q-learning algorithm

Classic Q-learning (Watkins and Dayan, 1992)

$$\text{buffer size} = 1$$
$$S = 1$$
$$G = 1$$

Fitted Q-iteration (Ernst et al., 2005, Riedmiller et al., 2005)

$\text{buffer size} = S$ (sampling size is a hyperparameter)

$G = \infty$ (until convergence)

Deep Q-Networks (Mnih et al., 2013)

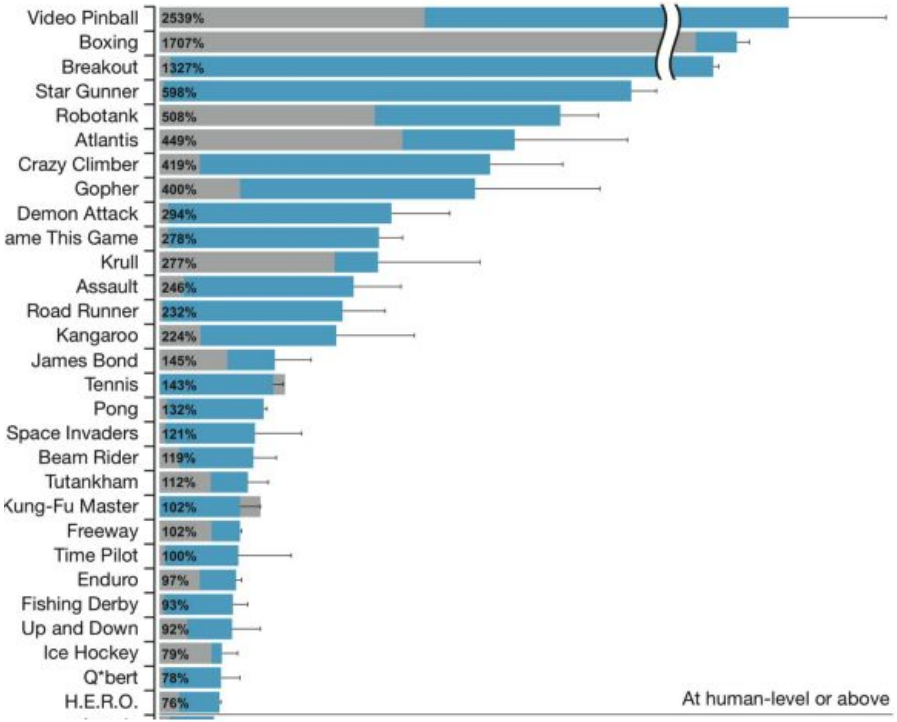
$\text{buffer size},\ S,\ G$ (all hyperparameters)

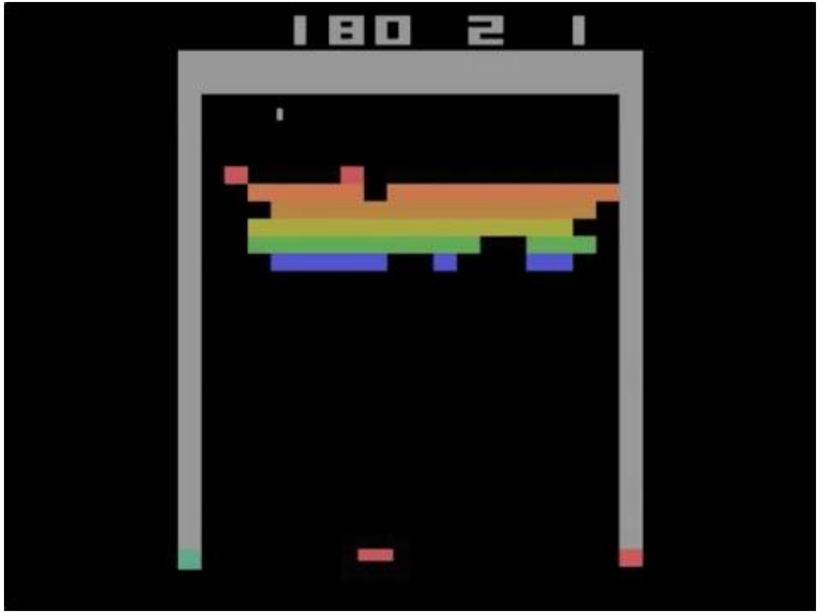
Collect transitions and run gradient steps concurrently

Sample random batches from experience replay $\longrightarrow$ decorrelate transitions

Lagging update of target network $\longrightarrow$ fix target network to stabilise learning

Deep Q-Networks results





InstaDeep™

Next steps?
Deep RL Prac

InstaDeep™

Other excellent sources:

- Reinforcement Learning: An Introduction by Richard S. Sutton and Andrew G. Barto
- OpenAI Spinning Up
- David Silver UCL Course

InstaDeep™

Questions?