



Intro to Deep Reinforcement Learning

(Beginner Friendly)

Indaba X Ghana

Kale-ab Tessera - Research Engineer, InstaDeep
September 23, 2022

Overview

- ❖ Why RL
- ❖ RL Flow and Intuition
- ❖ Tabular RL
- ❖ Q-Learning
- ❖ Deep Q-Learning

Why RL

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



[AlphaZero](#)

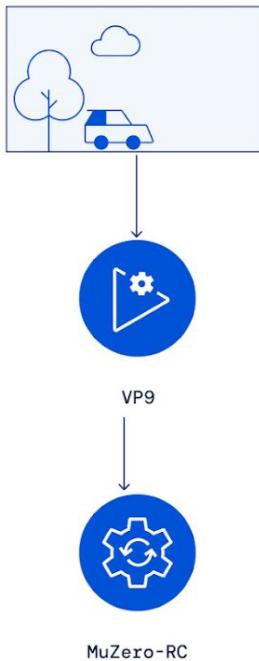
MuZero - Mastering Go, chess, shogi and Atari without rules



[MuZero](#)

Beyond Games

MuZero - YouTube to optimise video compression



[MuZero Youtube](#)

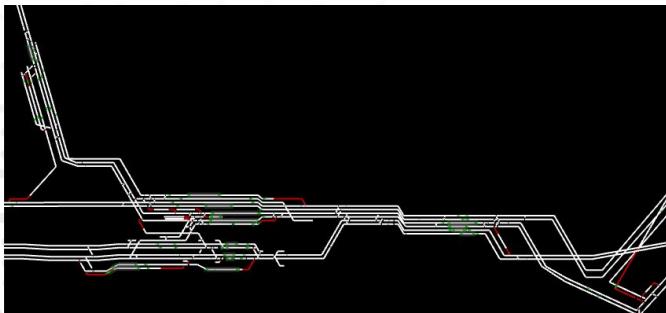
Train Scheduling with InstaDeep &



BAHN

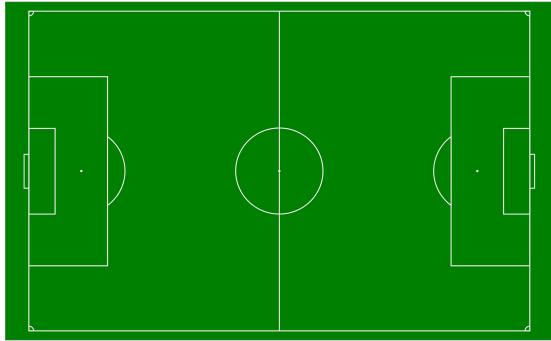
Solving Train Scheduling using Multi-agent Reinforcement Learning

- Key Challenge: **How to efficiently manage dense traffic on complex railway networks?**
- Why this problem is difficult:
 - Many trains and potential routes - solution must **scale**
 - Must accommodate delays, train breakdowns, etc - solution must be **robust**
 - Trains must be on time - solution must be **efficient**



RL Flow and Intuition

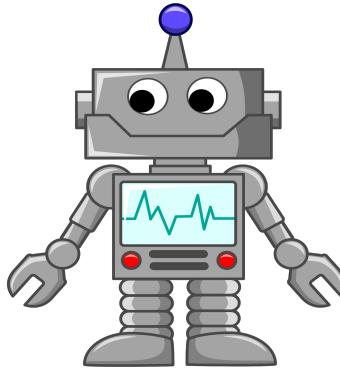
Reinforcement Learning (RL) Loop



Environment:

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

Reinforcement Learning (RL) Loop



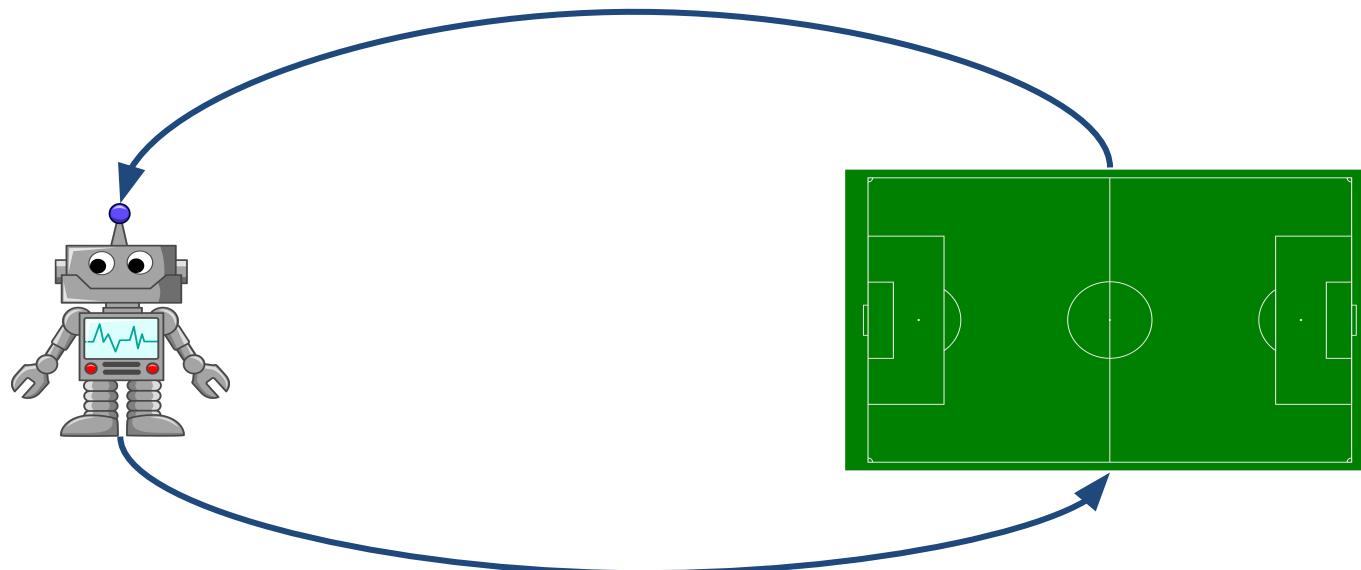
Agent:

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

Reinforcement Learning (RL) Loop

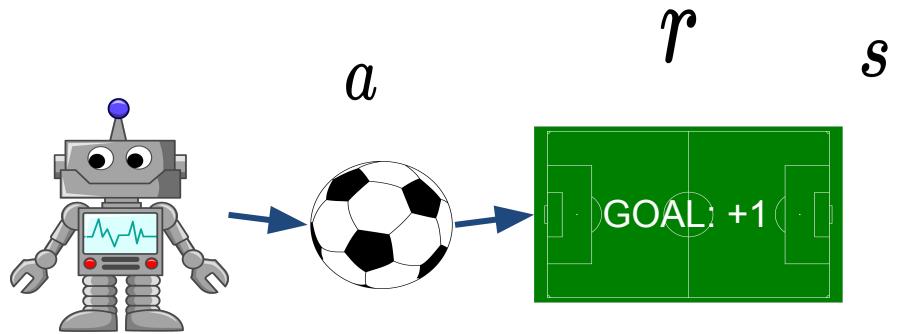
$s \in S$ Receive state from possible states.

r Receive reward.

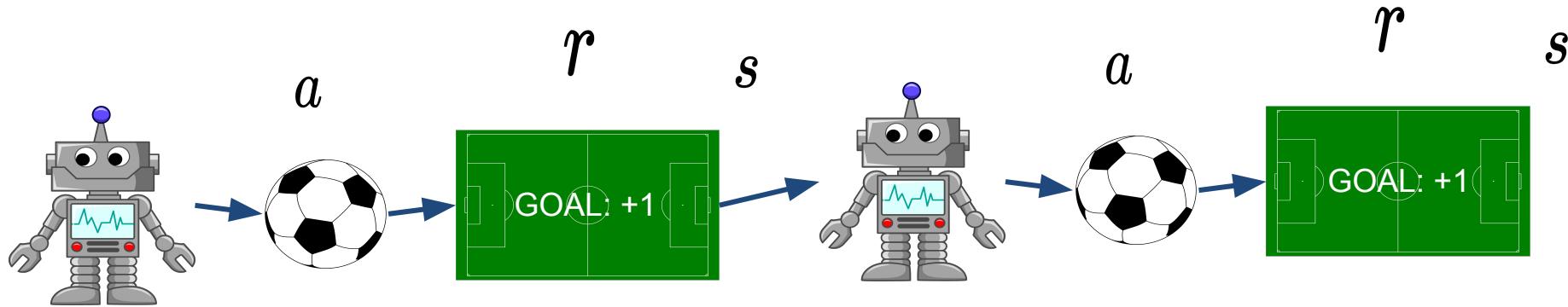


$a \in A$ Take action from available actions.

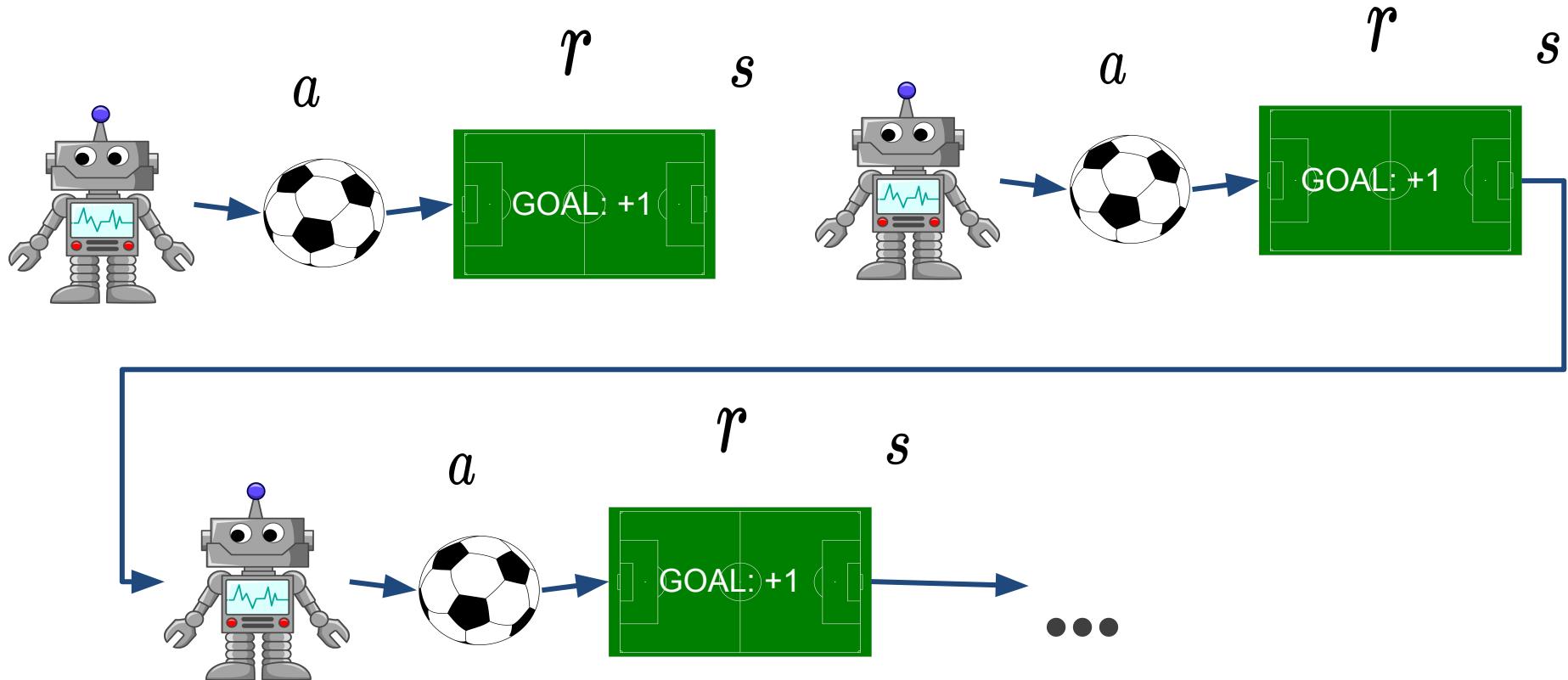
Reinforcement Learning (RL) Loop



Reinforcement Learning (RL) Loop

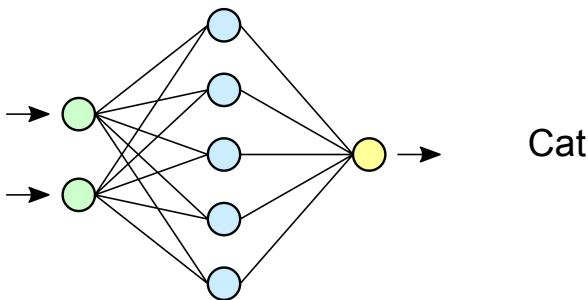


Reinforcement Learning (RL) Loop



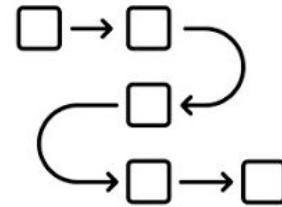
RL compared to Supervised Learning (SL) - Decisions

- SL - One-shot



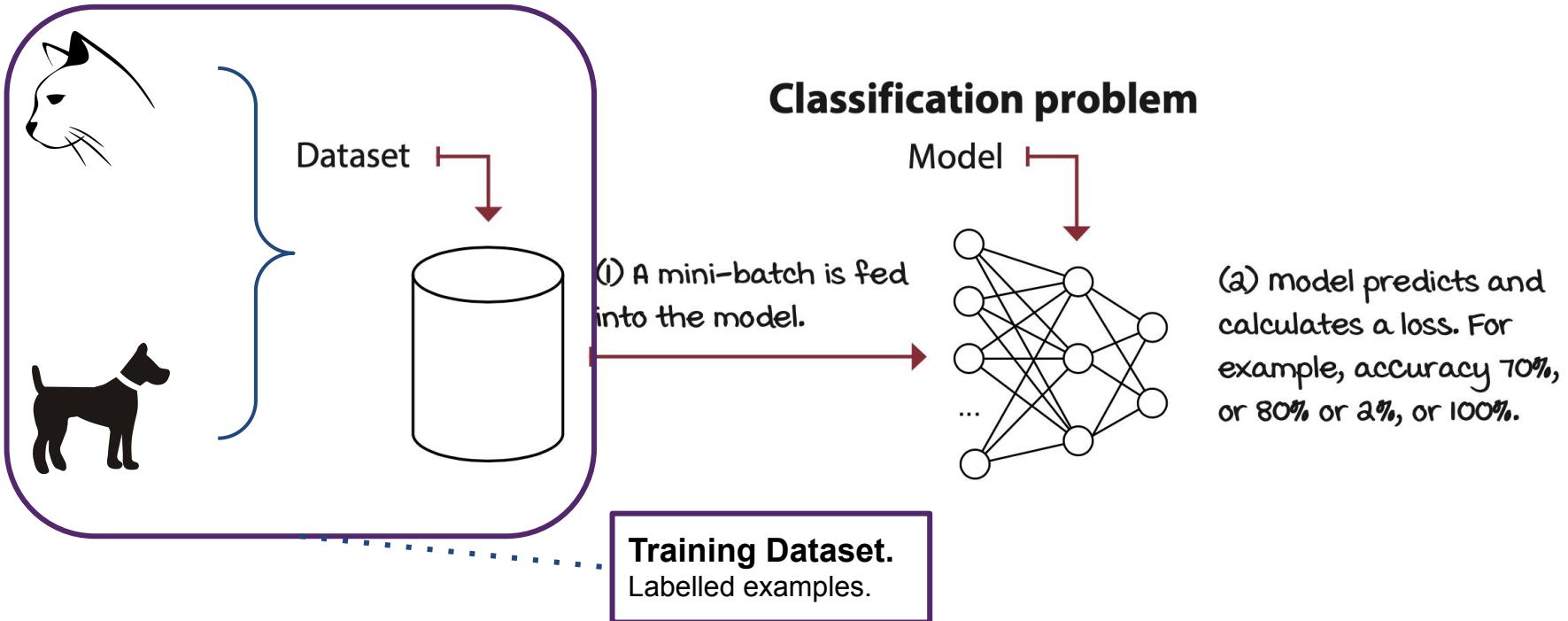
Cat

- RL - Sequential



RL compared to Supervised Learning - Training

- SL - Learn from labelled examples.

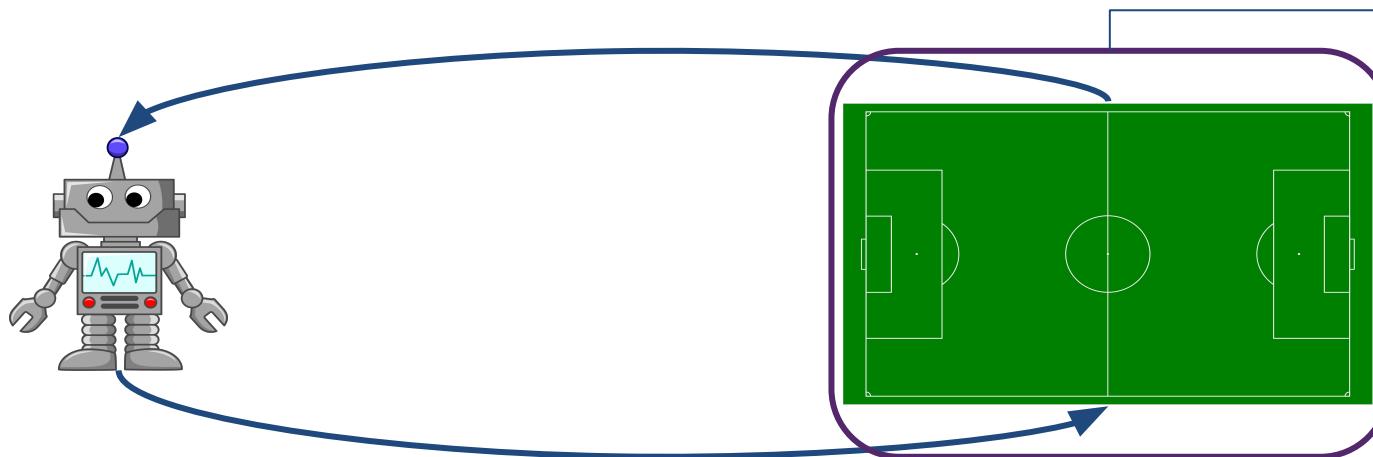


RL compared to Supervised Learning - Training - Trial and Error

- Learn from interacting with an environment.

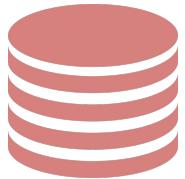
Environment/Simulator

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



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%.

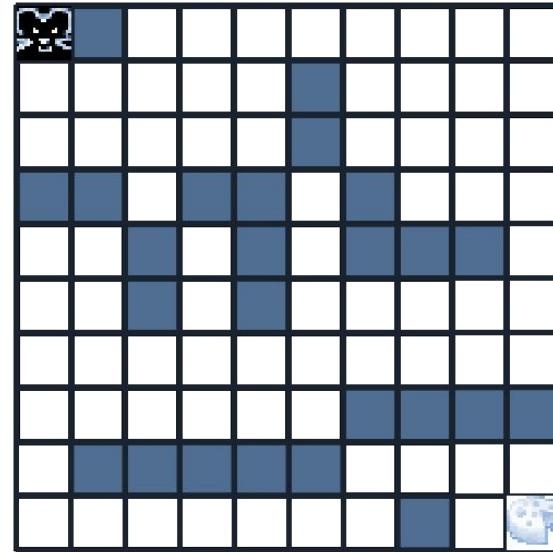
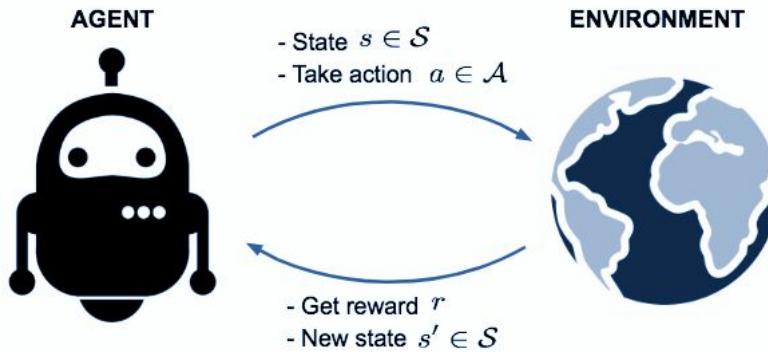
- RL - Maximize Cumulative Reward (Return).



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*).

Tabular RL

Markov Decision Process (MDP)

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

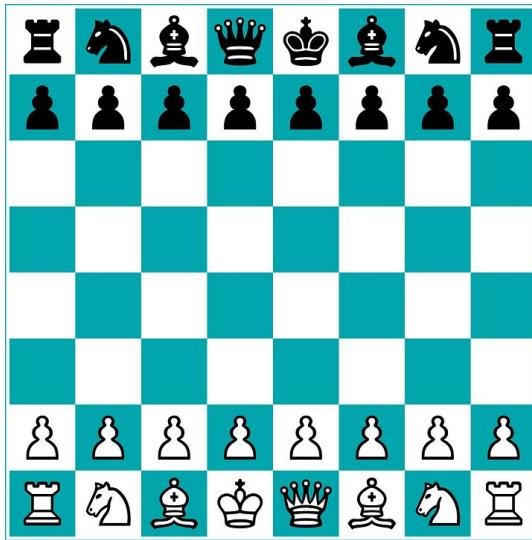
Markov Property: Transitions only depend on the most recent state and action, and no prior history (**current state contains all necessary information**).

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

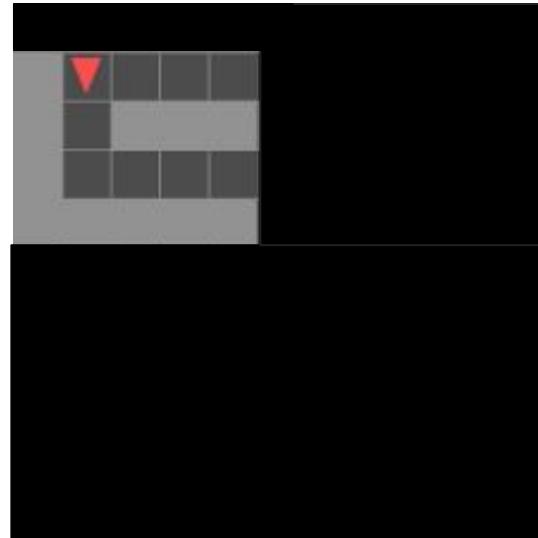
Probability of next state given current state = Probability of next state given whole history

Markov Property

Chess - Markovian.



Partially Observable Grid world - Not Markovian.



Markov Decision Process (MDP)

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

	-1	-1	-1	-1	-1
-10					
-10					
-10					
-10					
-10					
-10					
-10	-10	-10	-10	-10	GOAL 0

Markov Decision Process (MDP)

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

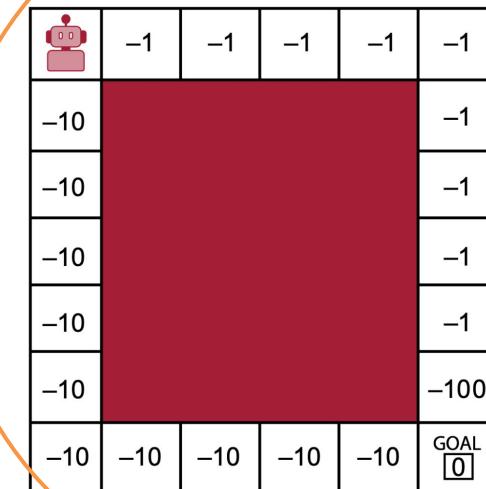
S - state space - is a finite set of states.

$s \in \mathbf{S}$, 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.



Markov Decision Process (MDP)

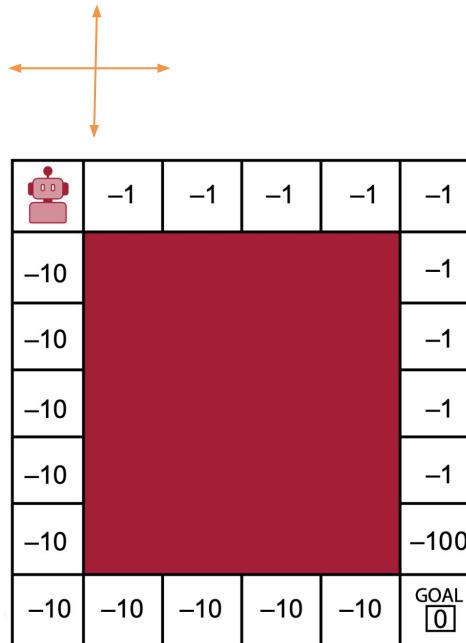
$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$$

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

$a \in \mathbf{A}$, what our agent does (discrete or continuous).

e.g.

- # 1. LEFT: 0
- # 2. DOWN = 1
- # 3. RIGHT = 2
- # 4. UP = 3



Markov Decision Process (MDP)

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

T - transition probability.

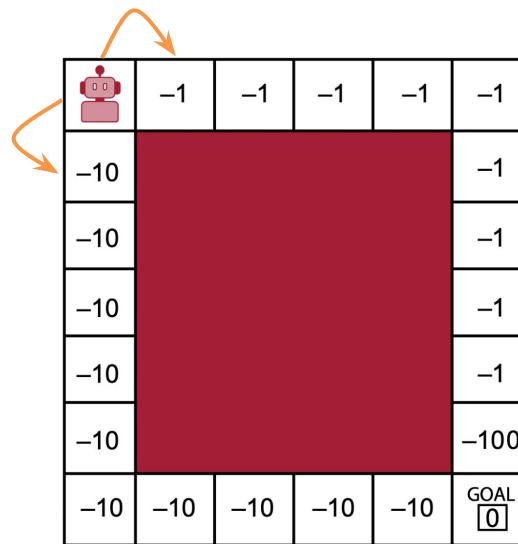
$$\mathcal{P}(s'|s, a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

Deterministic

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

Stochastic

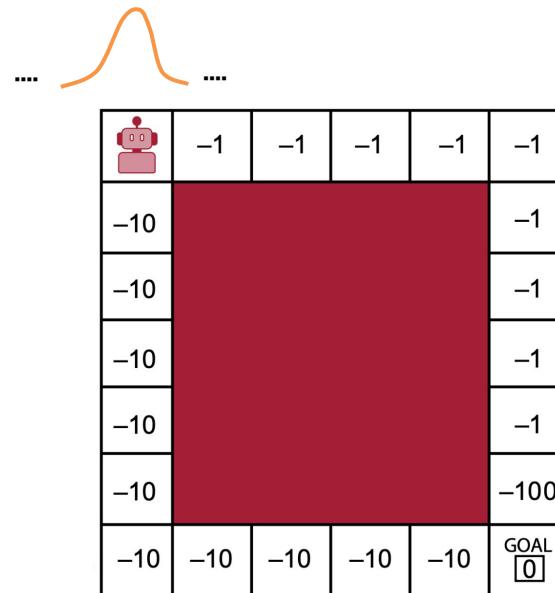
Probability distribution over transitions e.g. if you decide to go left, you will go left **50%** of the time, stay in your location **50%** of the time.



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?



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.

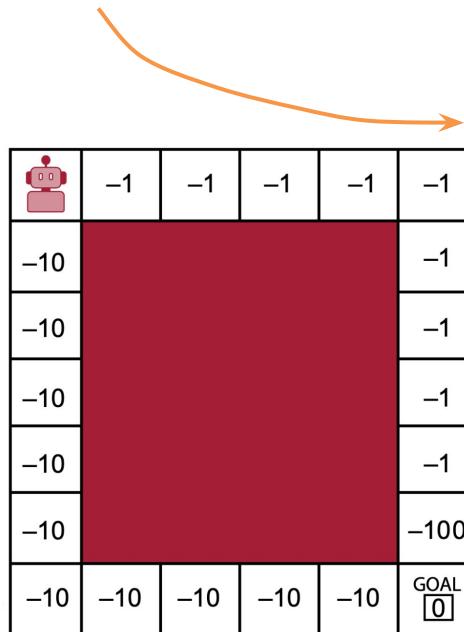
$$r(s_t, a_t)$$

	-1	-1	-1	-1	-1	-1
-10						-1
-10						-1
-10						-1
-10						-1
-10						-100
-10	-10	-10	-10	-10	GOAL 0	

Markov Decision Process (MDP)

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

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



Policy - Agents Brain

Policy: Mapping from (perceived) 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 :

$$\pi_{\theta}$$

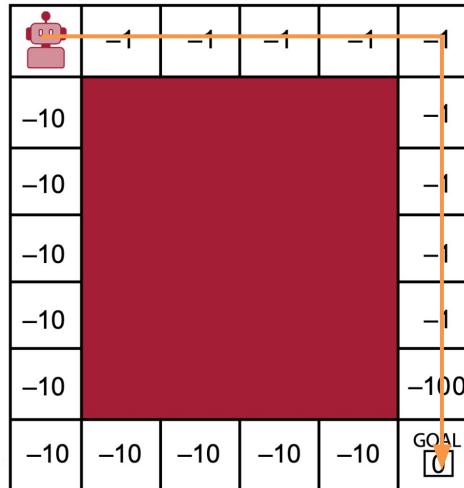


Markov Decision Process (MDP)

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

Trajectory

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



Markov Decision Process (MDP)

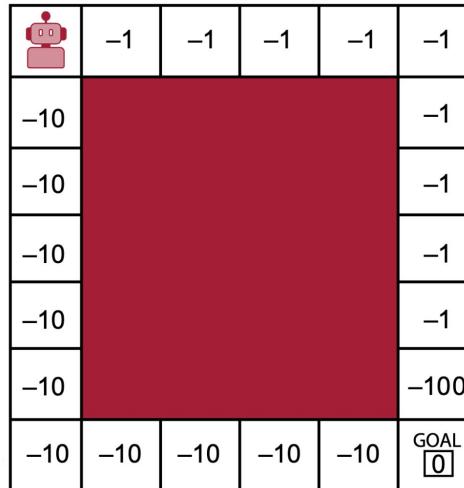
$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$$

Trajectory

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

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

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

Trajectory distribution

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

Probability of a specific trajectory.

	-1	-1	-1	-1	-1
-10					
-10					
-10					
-10					
-10					
-10					
-10	-10	-10	-10	-10	GOAL 0

RL Objective

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^T \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Trajectory sampled from trajectory distribution.

Discounted return (reward over time) following these trajectories.

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

Return vs Reward

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

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

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

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

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:

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

Action-value function:

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

Efficiently estimating values is critical to RL.

Kinds of RL Algorithms (Model-free)

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

Dynamic Programming - Bellman Equation

Value functions can be split into 2 parts:

$$Q^\pi(s, a) = \mathbb{E}_\pi[r(s, a) + \gamma Q^\pi(s_{t+1}, a_{t+1}) \mid S_t = s, A_t = a]$$

Immediate Reward.

Discounted value of next state.

Dynamic Programming

Bellman Operator

$$\vec{Q}^\pi = \mathcal{B}^\pi \vec{Q}^\pi \quad \lim_{k \rightarrow \infty} \vec{Q}_k^\pi = \vec{Q}^\pi$$

Policy Iteration

Actions

States

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

Policy Iteration

Actions

States

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)

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

Policy Iteration

Actions

States

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)

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

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

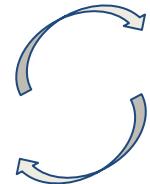
Actions

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

States

Policy Evaluation (Prediction)

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



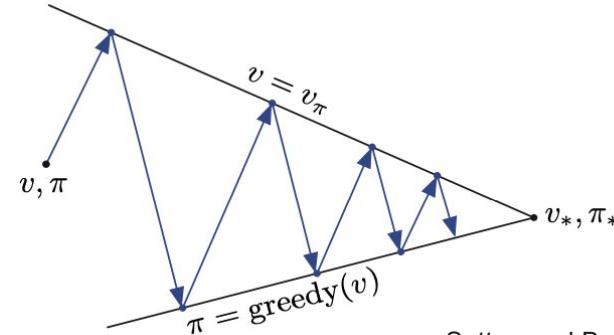
Policy Improvement (Control)

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

Policy Iteration

Actions

Q	States				
	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)

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



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

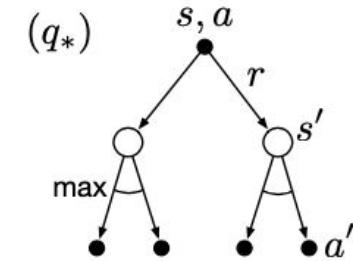
Bellman Optimality Equation

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

Value Iteration

Bellman Optimality Equation

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

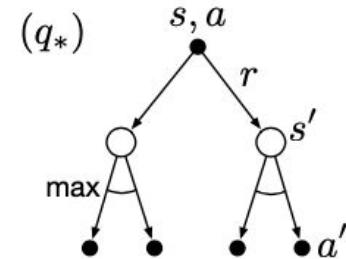


Value Iteration

Bellman Optimality Equation

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

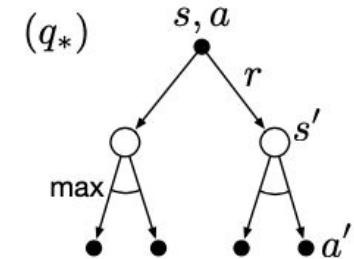
Need a model



Value Iteration

Bellman Optimality Equation

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



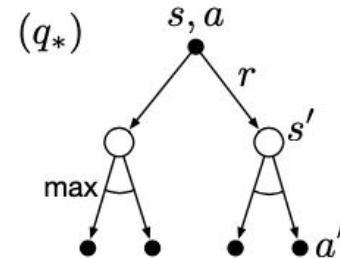
Value Iteration -> Temporal Difference (TD) learning

Bellman Optimality Equation

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

Incrementally estimate
using samples

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



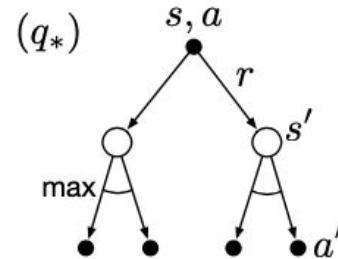
Value Iteration -> Temporal Difference (TD) learning

Bellman Optimality Equation

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

Incrementally estimate
using samples

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



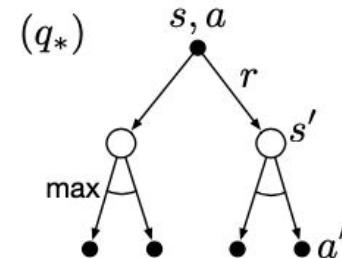
Value Iteration -> Temporal Difference (TD) learning

Bellman Optimality Equation

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

Incrementally estimate
using samples

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



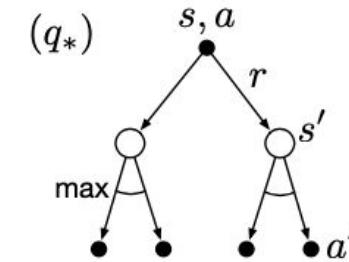
Value Iteration -> Temporal Difference (TD) learning

Bellman Optimality Equation

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

Incrementally estimate
using samples

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



TD Error

Q-Learning (Off-policy TD Learning)

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

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

The diagram illustrates the Q-Learning update rule. The formula is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

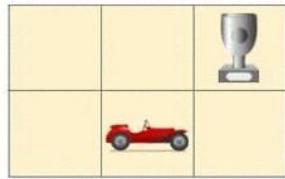
The components are labeled as follows:

- Old Estimate. (Purple box)
- Step Size. (Green box)
- Target (Orange box)
- Old Estimate. (Purple box)

A red bracket labeled **TD Error** spans the difference between the Target and Old Estimate.

Q-Learning (Off-policy TD Learning)

Game Board:



Current state (s): $\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$

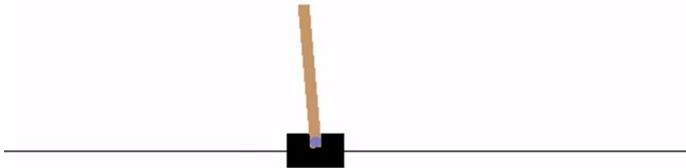
Q Table:

$\gamma = 0.95$

	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](#)

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:

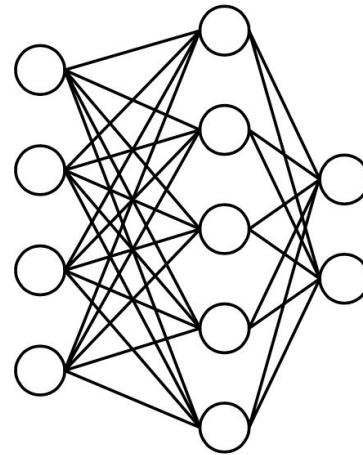
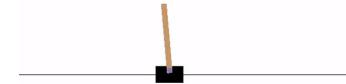
- Too many states to store in memory
- Too slow to update and estimate values for each state

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

Approx. Dynamic Programming using function approximation

The goal of function approximation

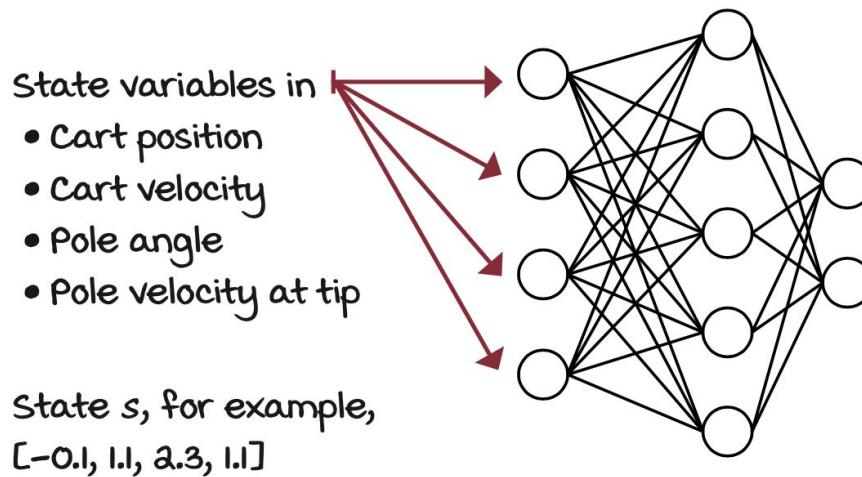
Approximate the values of states using a **parameterised function**



The goal of function approximation

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

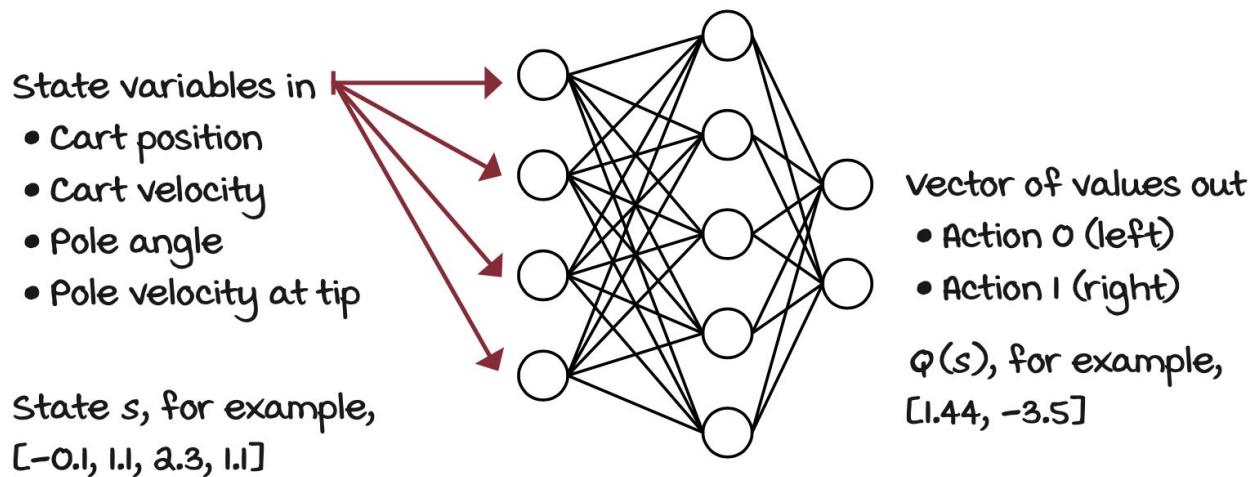
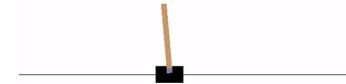
- Input: state features



The goal of function approximation

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

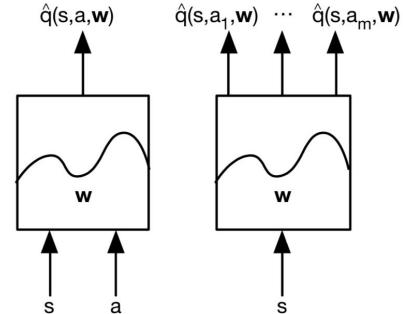
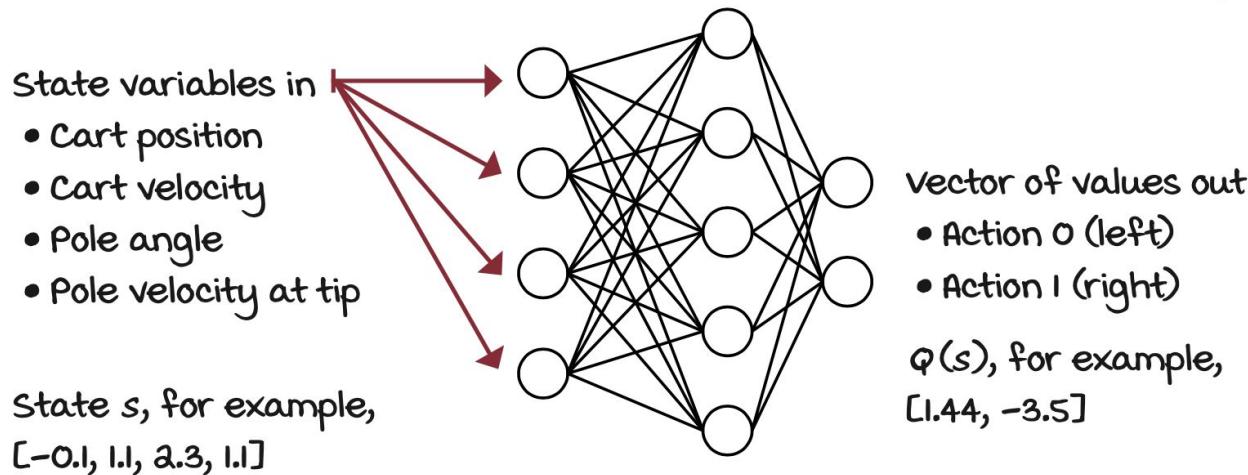
- **Input:** state features
- **Output:** estimated Q-values



The goal of function approximation

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

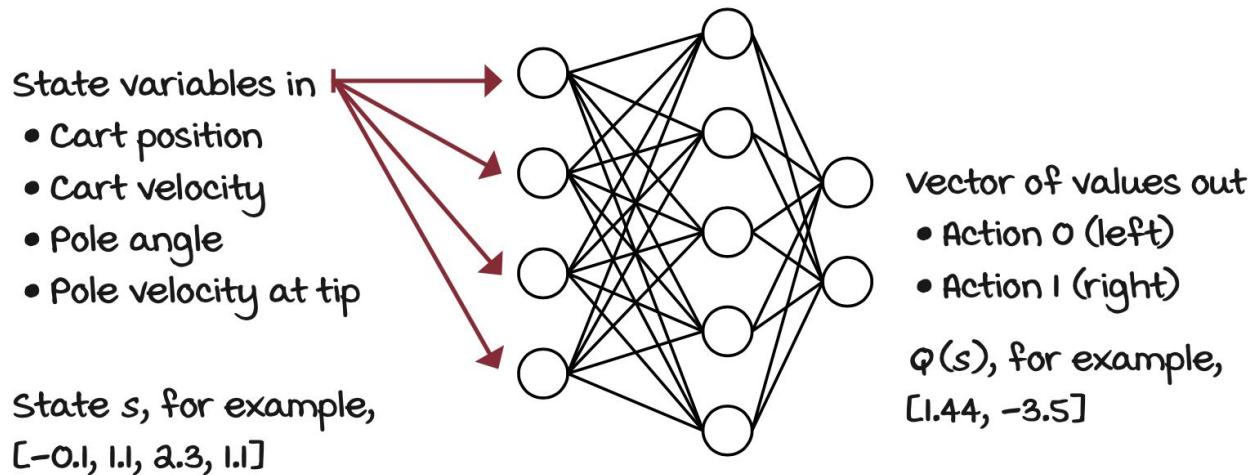
- Input: state features
- Output: estimated Q-values



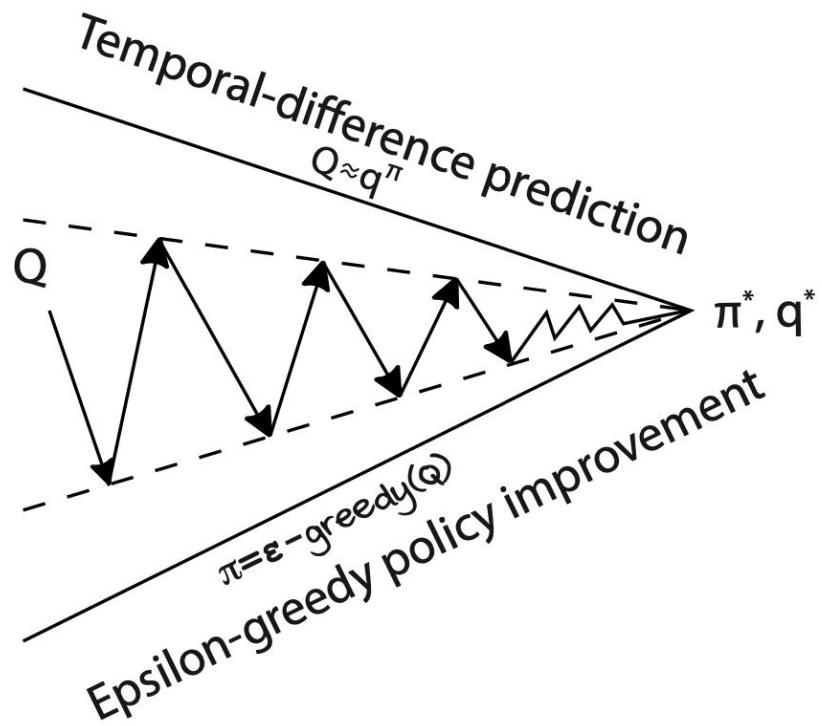
The goal of function approximation

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

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



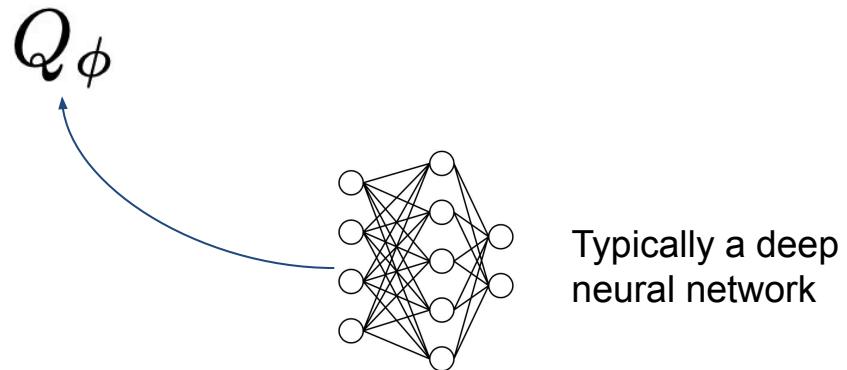
Approximate Dynamic Programming



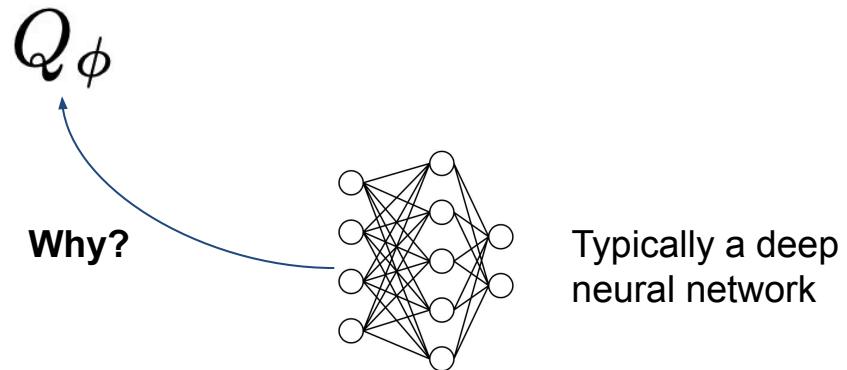
Approximate Dynamic Programming

$$Q_\phi$$

Approximate Dynamic Programming

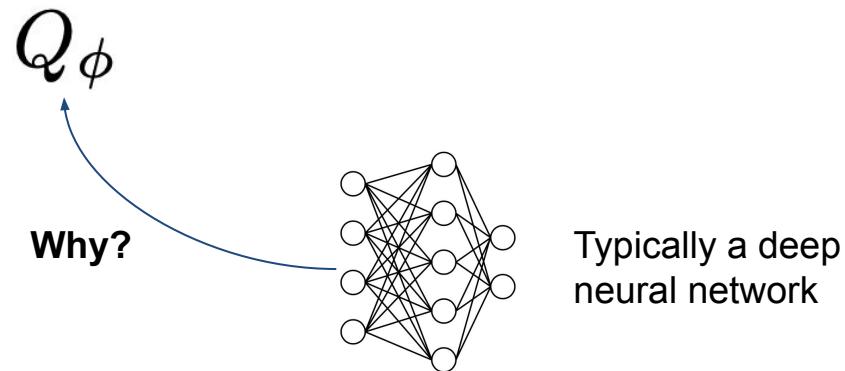
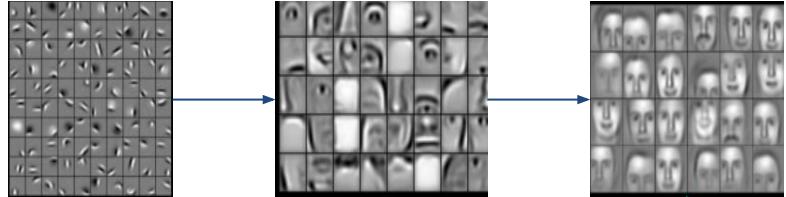


Approximate Dynamic Programming



Approximate Dynamic Programming

- Known to discover useful features

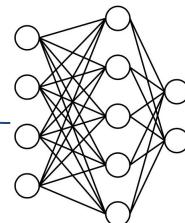


Approximate Dynamic Programming

- Known to discover useful features

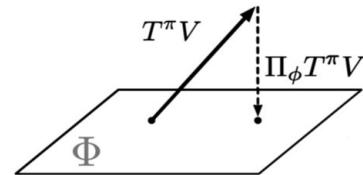
Q_ϕ

Why?

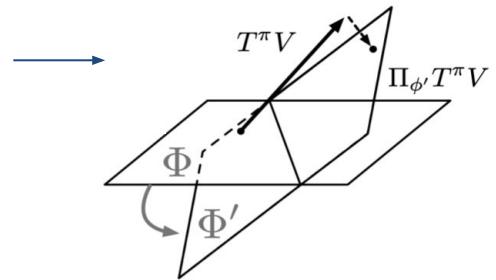


Typically a deep
neural network

Linear RL

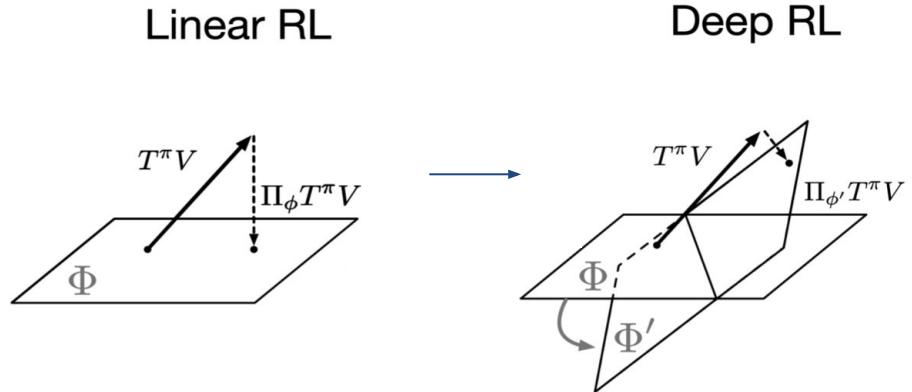
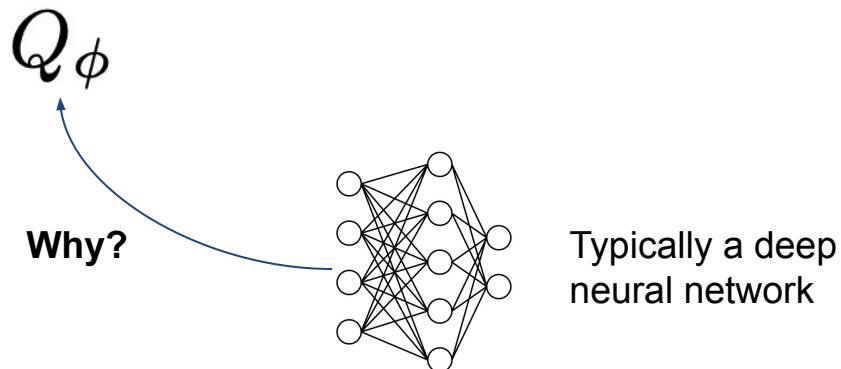


Deep RL

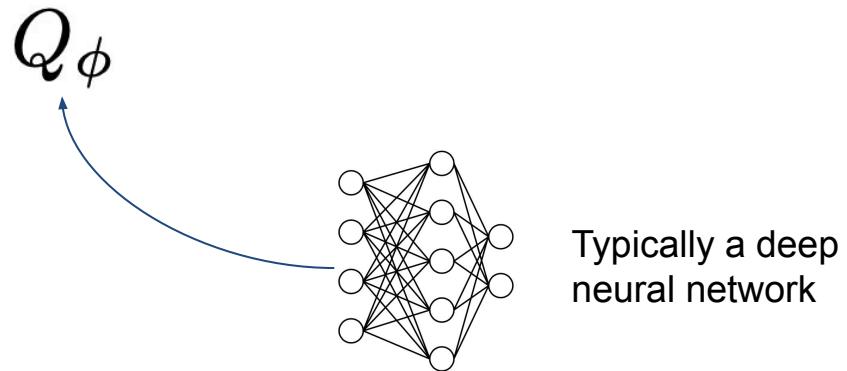


Approximate Dynamic Programming

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



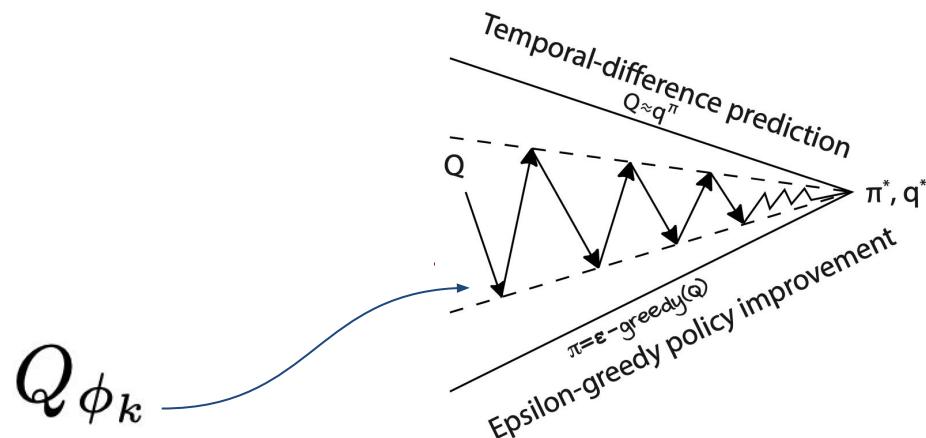
Approximate Dynamic Programming



Approximate Dynamic Programming

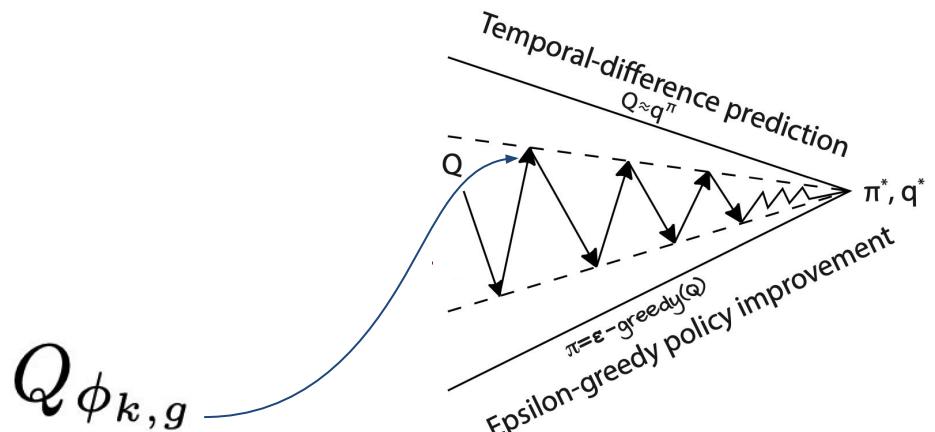
$$Q_{\phi_k}$$

Approximate Dynamic Programming



Controls policy
improvement step

Approximate Dynamic Programming



Controls policy
evaluation step

Approximate Dynamic Programming

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

Approximate Dynamic Programming

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

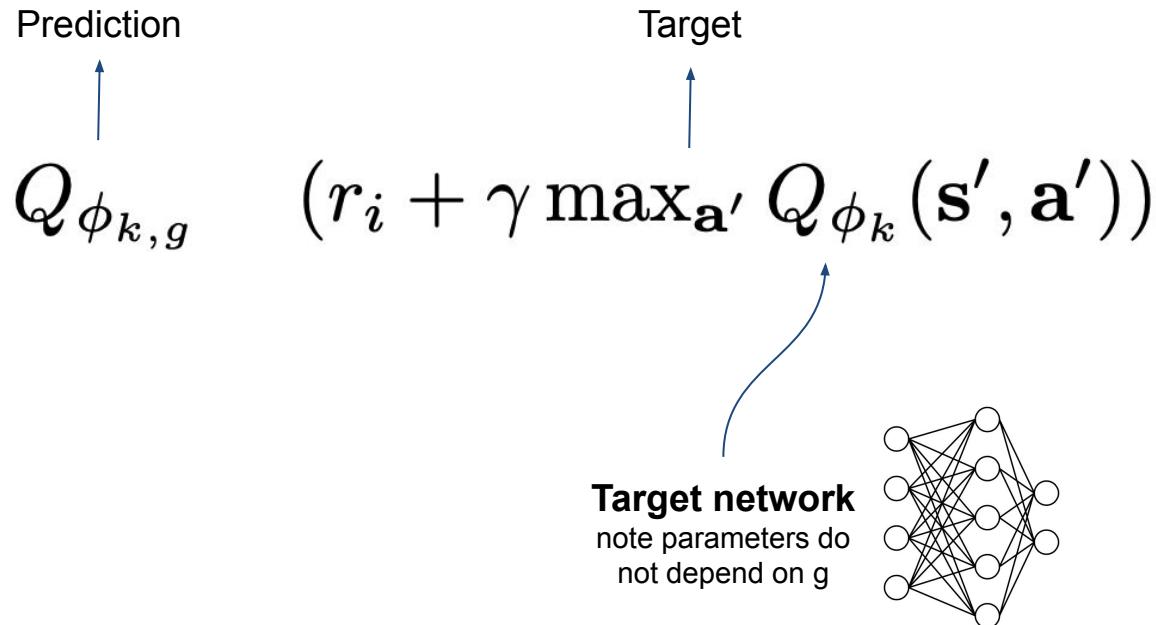
Approximate Dynamic Programming

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

Prediction Target

The diagram illustrates the components of a Q-learning update. On the left, the word "Prediction" is positioned above a blue upward-pointing arrow. To the right of the arrow is the mathematical expression for the predicted Q-value, $Q_{\phi_{k,g}}$. On the right, the word "Target" is positioned above another blue upward-pointing arrow. To the left of this arrow is the mathematical expression for the target Q-value, $(r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}'))$.

Approximate Dynamic Programming



Approximate Dynamic Programming

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

Prediction Target



Approximate Dynamic Programming

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

↑ ↑
Prediction Target

Approximate Dynamic Programming

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

TD Error

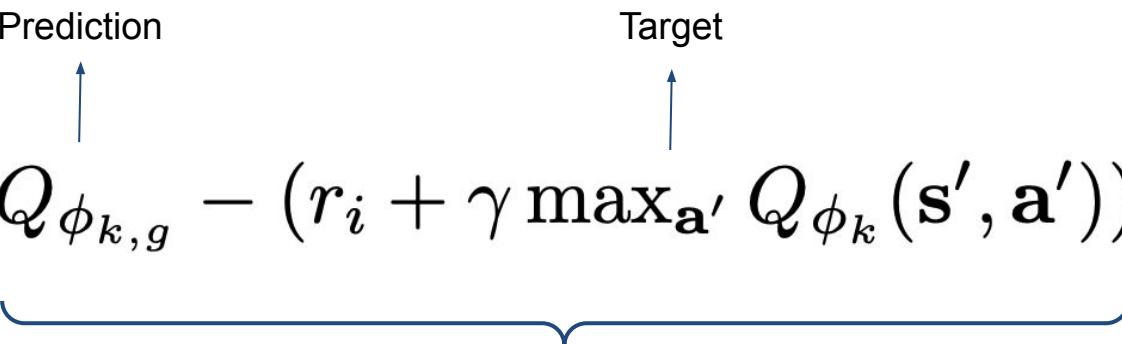
Prediction

Target

Approximate Dynamic Programming

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

TD Error



Approximate Dynamic Programming

Squared-error loss as in *supervised learning*

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

TD Error

The diagram illustrates the squared-error loss function used in Approximate Dynamic Programming. It features a curly brace grouping the two main components of the equation: the prediction term ($Q_{\phi_k, g}$) and the target term ($r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')$). Two vertical arrows point upwards from the ends of the grouped terms to the labels 'Prediction' and 'Target' respectively. The entire expression is enclosed in parentheses and squared.

Approximate Dynamic Programming

Squared-error loss as in *supervised learning*

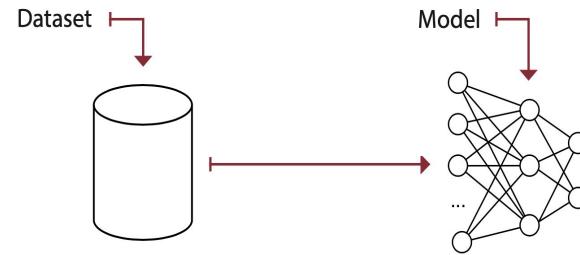
$$\sum_i \left(Q_{\phi_k, g} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')) \right)^2$$

Prediction

Target

TD Error

The diagram illustrates the training process of a machine learning model. A 'Dataset' is represented by a cylinder, which feeds into a 'Model' (a neural network). The 'Model' outputs a 'Prediction'. The 'Prediction' and a 'Target' (also labeled 'TD Error') are compared to calculate the squared-error loss. A blue curly brace groups the prediction term and the target term, while another blue bracket below groups the entire squared difference expression.



Approximate Dynamic Programming

Squared-error loss as in *supervised learning*

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

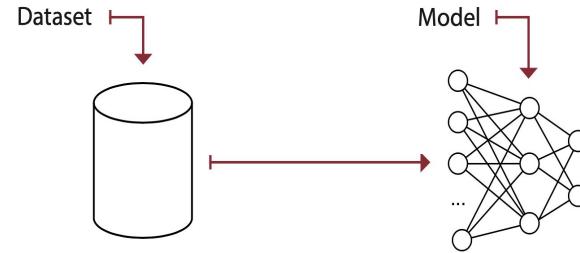
Prediction

Target

TD Error

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

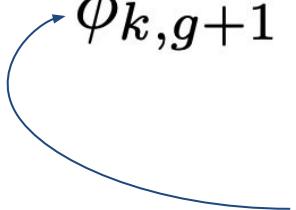
$$\sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')) \right)^2$$

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

Approximate Dynamic Programming

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

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



Update the parameters using **gradient descent**

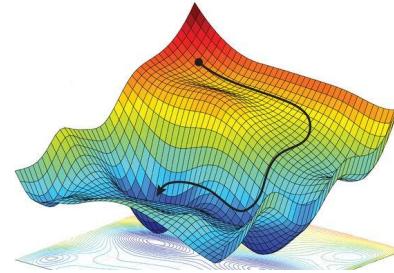
Approximate Dynamic Programming

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

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

Update the parameters using **gradient descent**

Approximate Dynamic Programming



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

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

Update the parameters using **gradient descent**

Approximate Dynamic Programming

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

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

Approximate Dynamic Programming

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

sample batch B

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

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

end for

Approximate Dynamic Programming

for gradient step $g \in [0, \dots, G - 1]$ **do**
 sample batch B

$$\sum_i (Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')))^2$$
$$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$$

end for



Approximate
Policy Evaluation

Approximate Dynamic Programming

for gradient step $g \in [0, \dots, G - 1]$ **do**
 sample batch B

$$\sum_i (Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')))^2$$
$$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$$

end for



Approximate
Policy Evaluation

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

Approximate Dynamic Programming

for gradient step $g \in [0, \dots, G - 1]$ **do**
 sample batch B

$$\sum_i (Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')))^2$$
$$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$$

end for



Approximate
Policy Evaluation

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

Act (ε) greedy with
respect to new
parameters

Approximate Dynamic Programming

for gradient step $g \in [0, \dots, G - 1]$ **do**
 sample batch B

$$\sum_i (Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')))^2$$
$$\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$$

end for

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

Act (ε) greedy with
respect to new
parameters

Approximate
Policy Improvement

Approximate
Policy Evaluation

Approximate Dynamic Programming

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

sample batch B

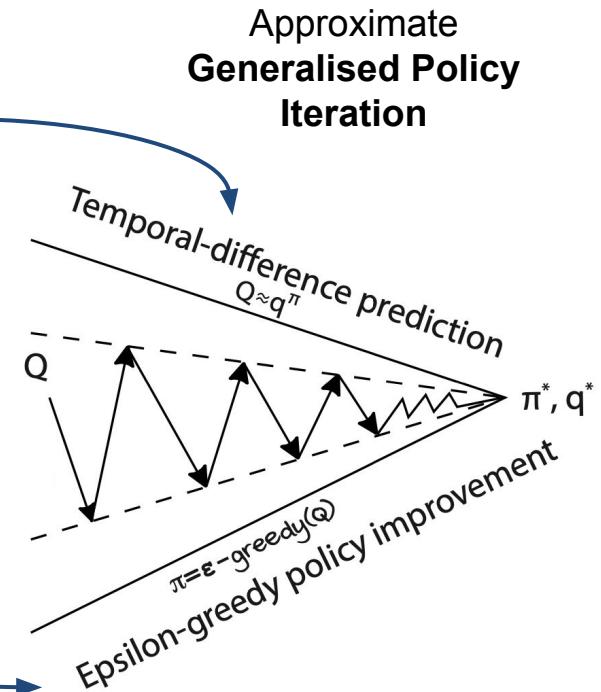
$$\sum_i (Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')))^2$$

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

end for

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

Act (ε) greedy with
respect to new
parameters



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 $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ \triangleright 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 $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ \triangleright 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 $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ \triangleright 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 $k \in [0, \dots, K]$ **do**

Generic Q-learning algorithm

- 1: initialize ϕ_0
- 2: initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ \triangleright 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 $k \in [0, \dots, K]$ **do**
- 6: **for** step $s \in [0, \dots, S - 1]$ **do**

Generic Q-learning algorithm

- ```

1: initialize ϕ_0
2: initialize $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$ \triangleright 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 $k \in [0, \dots, K]$ do
6: for step $s \in [0, \dots, S - 1]$ do
7: $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$ \triangleright sample action from exploration policy

```

# Generic Q-learning algorithm

- 1: initialize  $\phi_0$
- 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  $\epsilon$ -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  $k \in [0, \dots, K]$  **do**
- 6:   **for** step  $s \in [0, \dots, S - 1]$  **do**
- 7:      $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$                                   ▷ sample action from exploration policy
- 8:      $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$                                   ▷ sample next state from MDP

# Generic Q-learning algorithm

- ```

1: initialize  $\phi_0$ 
2: initialize  $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$      $\triangleright$  Use  $\epsilon$ -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  $k \in [0, \dots, K]$  do
6:   for step  $s \in [0, \dots, S - 1]$  do
7:      $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$                                  $\triangleright$  sample action from exploration policy
8:      $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$                        $\triangleright$  sample next state from MDP
9:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$      $\triangleright$  append to buffer, purging old data if buffer too big

```

Generic Q-learning algorithm

```
1: initialize  $\phi_0$ 
2: initialize  $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$      $\triangleright$  Use  $\epsilon$ -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  $k \in [0, \dots, K]$  do
6:   for step  $s \in [0, \dots, S - 1]$  do
7:      $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$                                  $\triangleright$  sample action from exploration policy
8:      $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$                    $\triangleright$  sample next state from MDP
9:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$      $\triangleright$  append to buffer, purging old data if buffer too big
10:   end for
```

Generic Q-learning algorithm

```
1: initialize  $\phi_0$ 
2: initialize  $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$      $\triangleright$  Use  $\epsilon$ -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  $k \in [0, \dots, K]$  do
6:   for step  $s \in [0, \dots, S - 1]$  do
7:      $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$                                  $\triangleright$  sample action from exploration policy
8:      $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$                    $\triangleright$  sample next state from MDP
9:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$      $\triangleright$  append to buffer, purging old data if buffer too big
10:   end for
11:    $\phi_{k,0} \leftarrow \phi_k$ 
```

Generic Q-learning algorithm

```
1: initialize  $\phi_0$ 
2: initialize  $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$      $\triangleright$  Use  $\epsilon$ -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  $k \in [0, \dots, K]$  do
6:   for step  $s \in [0, \dots, S - 1]$  do
7:      $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$                                  $\triangleright$  sample action from exploration policy
8:      $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$                    $\triangleright$  sample next state from MDP
9:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$      $\triangleright$  append to buffer, purging old data if buffer too big
10:    end for
11:     $\phi_{k,0} \leftarrow \phi_k$ 
12:    for gradient step  $g \in [0, \dots, G - 1]$  do
```

Generic Q-learning algorithm

```
1: initialize  $\phi_0$ 
2: initialize  $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$      $\triangleright$  Use  $\epsilon$ -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  $k \in [0, \dots, K]$  do
6:   for step  $s \in [0, \dots, S - 1]$  do
7:      $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$                                  $\triangleright$  sample action from exploration policy
8:      $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$                    $\triangleright$  sample next state from MDP
9:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$      $\triangleright$  append to buffer, purging old data if buffer too big
10:    end for
11:     $\phi_{k,0} \leftarrow \phi_k$ 
12:    for gradient step  $g \in [0, \dots, G - 1]$  do
13:      sample batch  $B \subset \mathcal{D}$                                  $\triangleright B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$ 
```

Generic Q-learning algorithm

```
1: initialize  $\phi_0$ 
2: initialize  $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon \mathcal{U}(\mathbf{a}) + (1 - \epsilon) \delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$      $\triangleright$  Use  $\epsilon$ -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  $k \in [0, \dots, K]$  do
6:   for step  $s \in [0, \dots, S - 1]$  do
7:      $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$                                  $\triangleright$  sample action from exploration policy
8:      $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$                    $\triangleright$  sample next state from MDP
9:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$      $\triangleright$  append to buffer, purging old data if buffer too big
10:    end for
11:     $\phi_{k,0} \leftarrow \phi_k$ 
12:    for gradient step  $g \in [0, \dots, G - 1]$  do
13:      sample batch  $B \subset \mathcal{D}$                        $\triangleright B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$ 
14:      estimate error  $\mathcal{E}(B, \phi_{k,g}) = \sum_i (Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')))^2$ 
```

Generic Q-learning algorithm

```
1: initialize  $\phi_0$ 
2: initialize  $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon \mathcal{U}(\mathbf{a}) + (1 - \epsilon) \delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$      $\triangleright$  Use  $\epsilon$ -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  $k \in [0, \dots, K]$  do
6:   for step  $s \in [0, \dots, S - 1]$  do
7:      $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$                                  $\triangleright$  sample action from exploration policy
8:      $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$                    $\triangleright$  sample next state from MDP
9:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$      $\triangleright$  append to buffer, purging old data if buffer too big
10:    end for
11:     $\phi_{k,0} \leftarrow \phi_k$ 
12:    for gradient step  $g \in [0, \dots, G - 1]$  do
13:      sample batch  $B \subset \mathcal{D}$                              $\triangleright B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$ 
14:      estimate error  $\mathcal{E}(B, \phi_{k,g}) = \sum_i (Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')))^2$ 
15:      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

```
1: initialize  $\phi_0$ 
2: initialize  $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$      $\triangleright$  Use  $\epsilon$ -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  $k \in [0, \dots, K]$  do
6:   for step  $s \in [0, \dots, S - 1]$  do
7:      $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$                                  $\triangleright$  sample action from exploration policy
8:      $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$                    $\triangleright$  sample next state from MDP
9:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$      $\triangleright$  append to buffer, purging old data if buffer too big
10:    end for
11:     $\phi_{k,0} \leftarrow \phi_k$ 
12:    for gradient step  $g \in [0, \dots, G - 1]$  do
13:      sample batch  $B \subset \mathcal{D}$                                  $\triangleright B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$ 
14:      estimate error  $\mathcal{E}(B, \phi_{k,g}) = \sum_i (Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')))^2$ 
15:      update parameters:  $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$ 
16:    end for
```

Generic Q-learning algorithm

```
1: initialize  $\phi_0$ 
2: initialize  $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$      $\triangleright$  Use  $\epsilon$ -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  $k \in [0, \dots, K]$  do
6:   for step  $s \in [0, \dots, S - 1]$  do
7:      $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$                                  $\triangleright$  sample action from exploration policy
8:      $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$                    $\triangleright$  sample next state from MDP
9:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$      $\triangleright$  append to buffer, purging old data if buffer too big
10:    end for
11:     $\phi_{k,0} \leftarrow \phi_k$ 
12:    for gradient step  $g \in [0, \dots, G - 1]$  do
13:      sample batch  $B \subset \mathcal{D}$                                  $\triangleright B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$ 
14:      estimate error  $\mathcal{E}(B, \phi_{k,g}) = \sum_i (Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')))^2$ 
15:      update parameters:  $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$ 
16:    end for
17:     $\phi_{k+1} \leftarrow \phi_{k,G}$                                  $\triangleright$  update parameters
```

Generic Q-learning algorithm

```
1: initialize  $\phi_0$ 
2: initialize  $\pi_0(\mathbf{a}|\mathbf{s}) = \epsilon\mathcal{U}(\mathbf{a}) + (1 - \epsilon)\delta(\mathbf{a} = \arg \max_{\mathbf{a}} Q_{\phi_0}(\mathbf{s}, \mathbf{a}))$      $\triangleright$  Use  $\epsilon$ -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  $k \in [0, \dots, K]$  do
6:   for step  $s \in [0, \dots, S - 1]$  do
7:      $\mathbf{a} \sim \pi_k(\mathbf{a}|\mathbf{s})$                                  $\triangleright$  sample action from exploration policy
8:      $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$                    $\triangleright$  sample next state from MDP
9:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r(\mathbf{s}, \mathbf{a}))\}$      $\triangleright$  append to buffer, purging old data if buffer too big
10:    end for
11:     $\phi_{k,0} \leftarrow \phi_k$ 
12:    for gradient step  $g \in [0, \dots, G - 1]$  do
13:      sample batch  $B \subset \mathcal{D}$                                  $\triangleright B = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_t)\}$ 
14:      estimate error  $\mathcal{E}(B, \phi_{k,g}) = \sum_i (Q_{\phi_{k,g}} - (r_i + \gamma \max_{\mathbf{a}'} Q_{\phi_k}(\mathbf{s}', \mathbf{a}')))^2$ 
15:      update parameters:  $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$ 
16:    end for
17:     $\phi_{k+1} \leftarrow \phi_{k,G}$                                  $\triangleright$  update parameters
18: 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

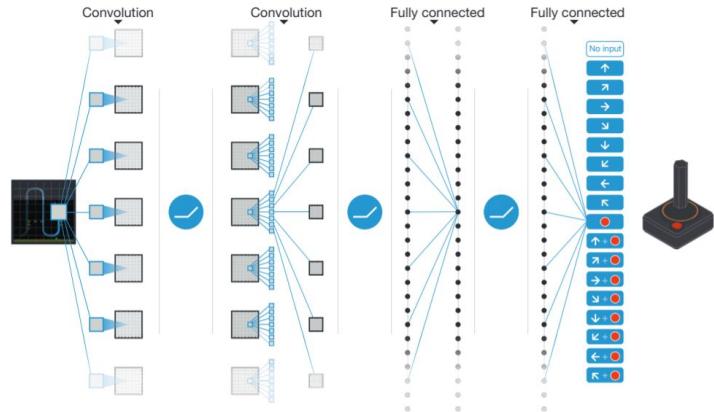
We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function 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.

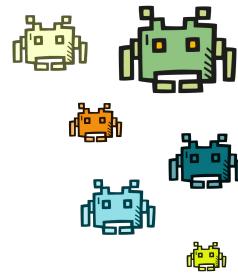
LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih^{1*}, Koray Kavukcuoglu^{1*}, David Silver^{1*}, Andrei A. Rusu¹, Joel Veness¹, Marc G. Bellemare¹, Alex Graves¹, Martin Riedmiller¹, Andreas K. Fidjeland¹, Georg Ostrovski¹, Stig Petersen¹, Charles Beattie¹, Amir Sadik¹, Ioannis Antonoglou¹, Helen King¹, Dharshan Kumaran¹, Daan Wierstra¹, Shane Legg¹ & Demis Hassabis¹





Stability Issues with Deep Q-Networks

Naive Q-Learning oscillates or diverges with neural networks:

- ❑ Data is sequential:
Successive samples 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

Special cases of the generic Q-learning algorithm

Classic Q-learning (Watkins and Dayan, 1992)

buffer size = 1

$S = 1$

$G = 1$

Special cases of the generic Q-learning algorithm

Classic Q-learning (Watkins and Dayan, 1992)

buffer size = 1

$S = 1$

$G = 1$

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

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)

buffer size = 1

$S = 1$

$G = 1$

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

buffer size = S (sampling size is a hyperparameter)

$G = \infty$ (until convergence)

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

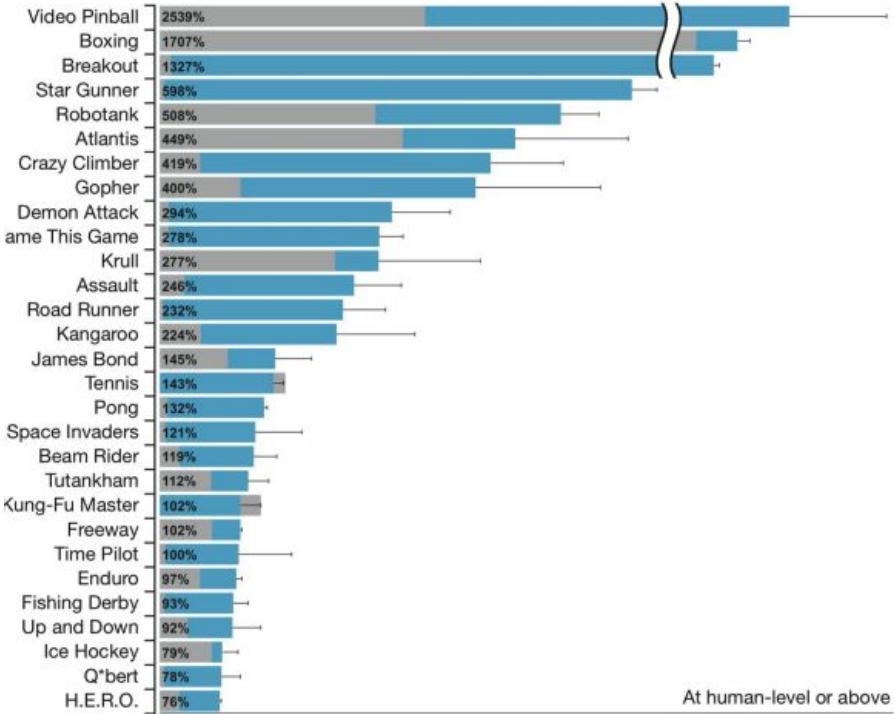
buffer size, S , G (all hyperparameters)

Collect transitions and run gradient steps concurrently

Sample random batches from experience replay ——> decorrelate transitions

Lagging update of target network ——> fix target network to stabilise learning

Deep Q-Networks results



Next steps? Deep RL Prac



Thank you Questions?



Slides by Kale-ab Tessera and Arnu Pretorius