European Commission
Horizon 2020
European Union funding
for Research & Innovation

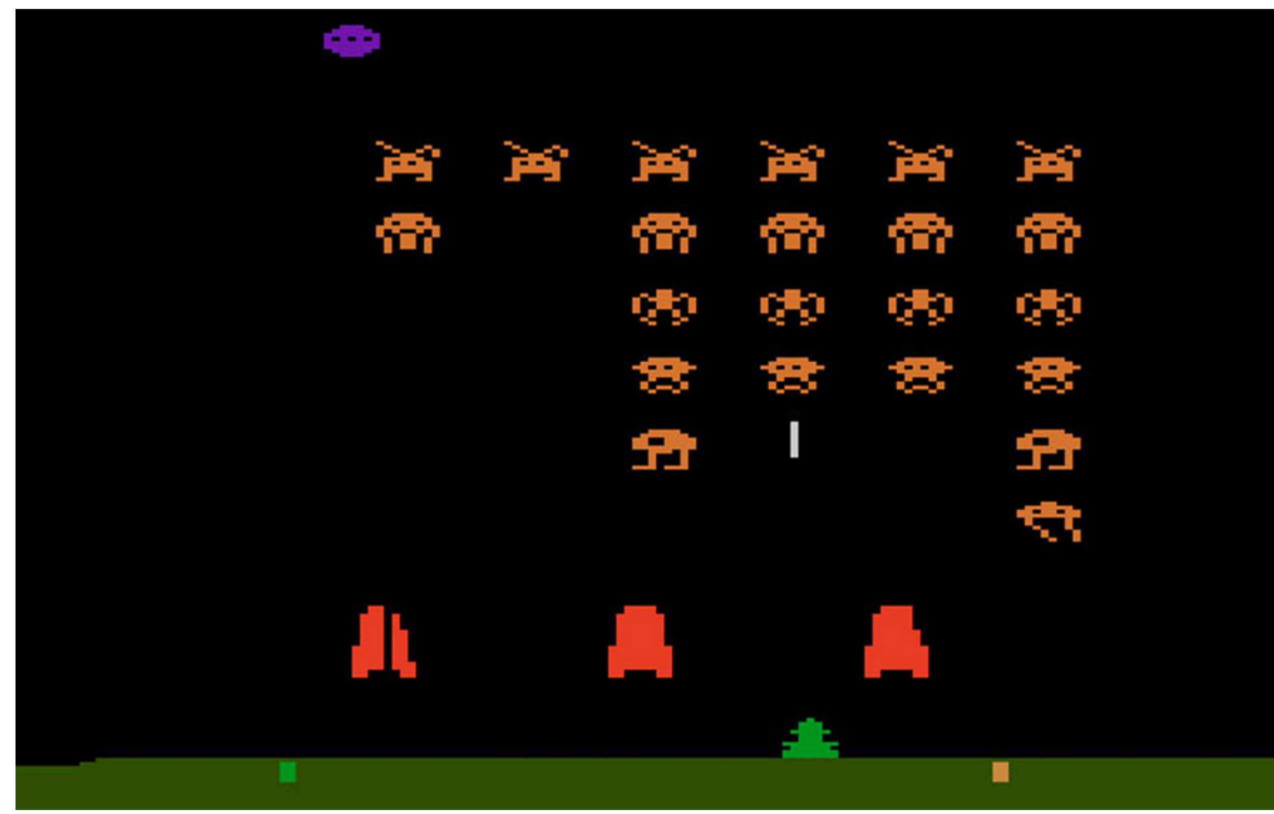
Introduction to Deep Reinforcement Learning

Ludovic Righetti (MPI / NYU)
Nicolas Mansard (LAAS/CNRS)

What is reinforcement learning?

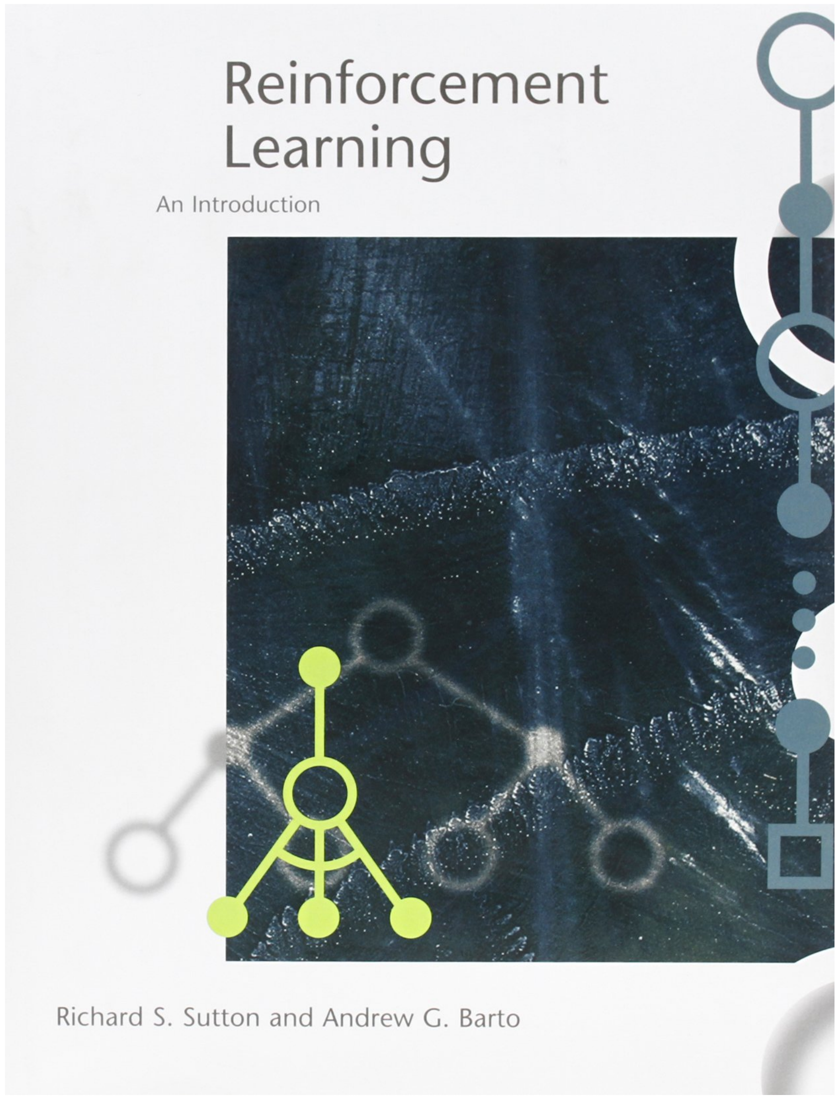
Learning "how to act" from direct interaction with the environment with the goal to maximize a "reward"

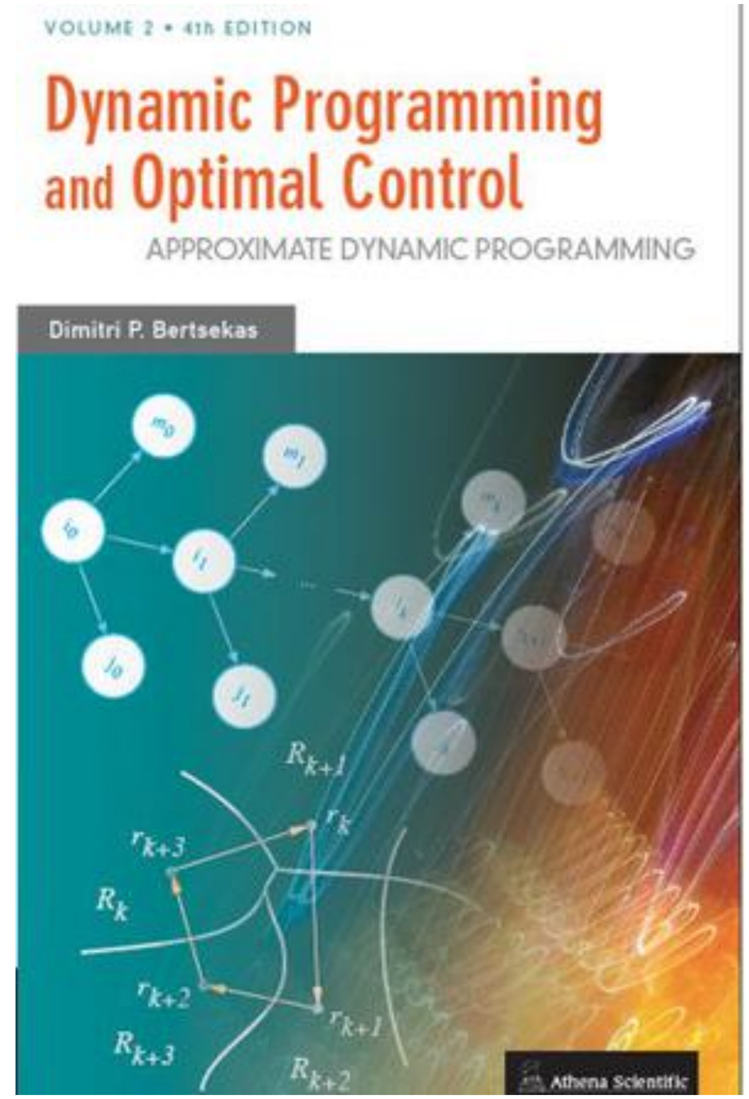
[Silver et al., Nature, 2016]



[Mnih et al., Nature, 2015]

What is reinforcement learning?

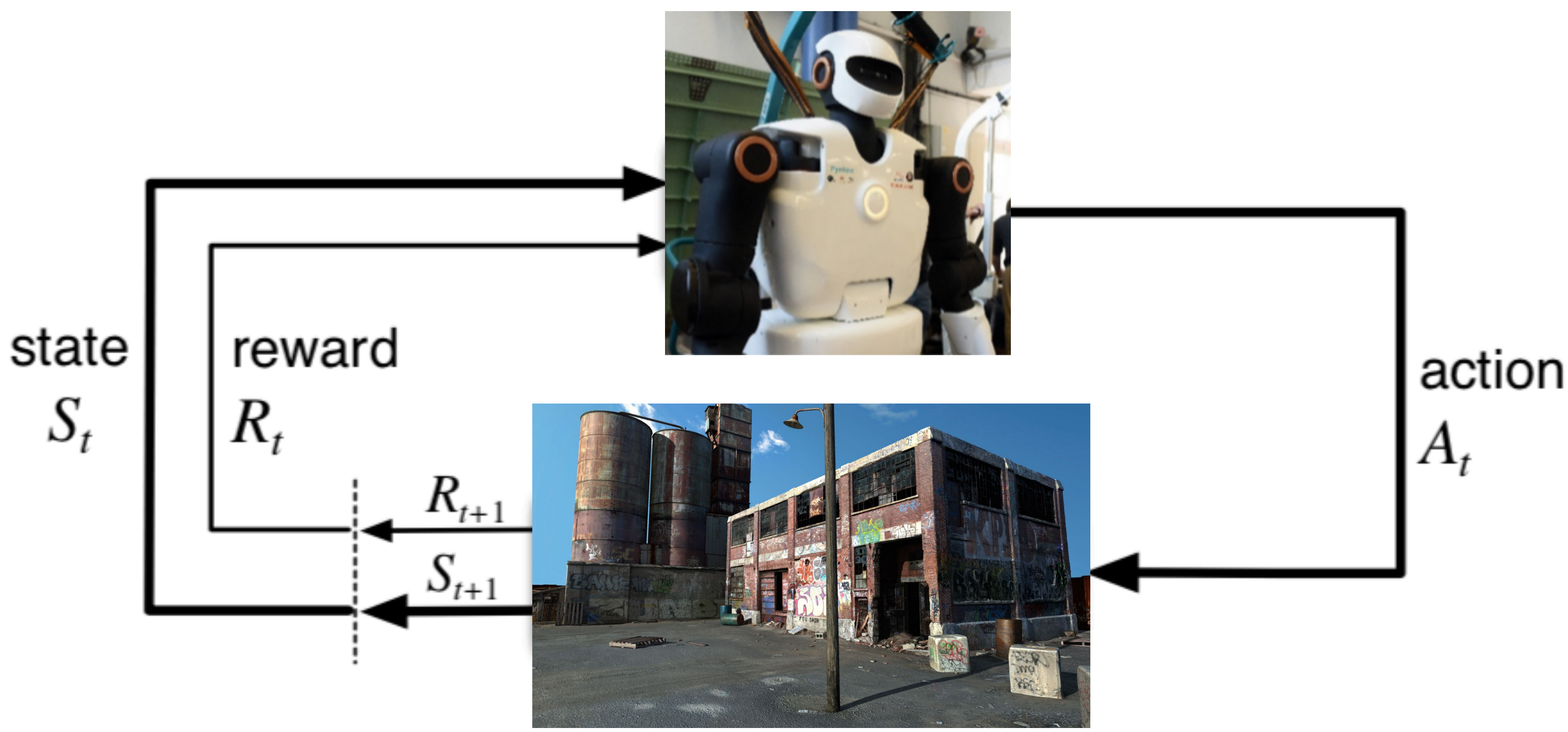




Jens Kober et al. "Reinforcement Learning in Robotics: A Survey", IJRR 2013

What is reinforcement learning?

$$\max \sum_t R_t(S_t, A_t)$$



$$S_{t+1} = f(S_t, A_t)$$

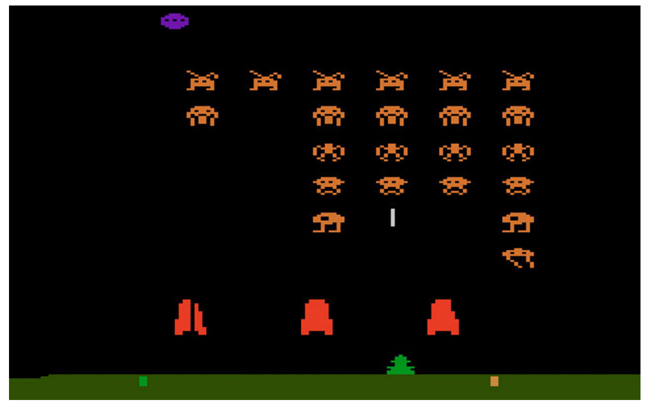
Markov Decision Process

Policy

$$A_t = \pi(S_t) \, ?$$

What is reinforcement learning?

Typical RL problems



State is discrete (countable)

Set of actions is discrete

Robotics RL problems

State is continuous

Action space is continuous

Most methods designed for discrete state/action models do not carry over to continuous sate/action models

What is the difference with O.C.?

Reinforcement learning

$$\max \sum_t R_t(S_t, A_t)$$

$$\min \sum_t -R_t(S_t, A_t)$$

$$S_{t+1} = \text{?}$$

In RL we do not know the dynamic model

Optimal Control

$$\min \sum_t C_t(S_t, A_t)$$

$$S_{t+1} = f(S_t, A_t)$$

Our typical optimal control setup (cf. DDP tutorial)

Model-based reinforcement learning

Reinforcement learning

$$\min \sum_t C_t(S_t, A_t)$$

$$S_{t+1} = \textbf{?}$$

Model-based reinforcement learning => Sample the world, learn a model (i.e. system identification) and then do optimal control

[Schaal and Atkeson, Control Systems Magazine, 1994]

[Levine and Koltun, ICML, 2013]

Model-free reinforcement learning

Reinforcement learning

$$\min \sum_t C_t(S_t, A_t)$$

$$S_{t+1} = \textbf{?}$$

Ignore the model and learn directly the policy (or an approximation of the value function to infer the policy)
=> Q-learning, actor-critic algorithms, policy gradients, etc

Let's go back to Bellman

$$\min_{A_t} \sum_{t=0}^{\infty} \gamma^t C_t(S_t, A_t)$$

γ Discount factor

$$S_{t+1} = f(S_t, A_t)$$

Bellman's Equations (infinite horizon case)

$V(S_t)$

Valu

i.e. optimal

obtained

opti

V(S_t) is the unique solution of this equation within the class of bounded functions (necessary and sufficient condition for optimality)

Value Iteration

$$V(S_t) = \min_{A_t} \left(C_t(S_t, A_t) + \gamma V(S_{t+1})\right)$$

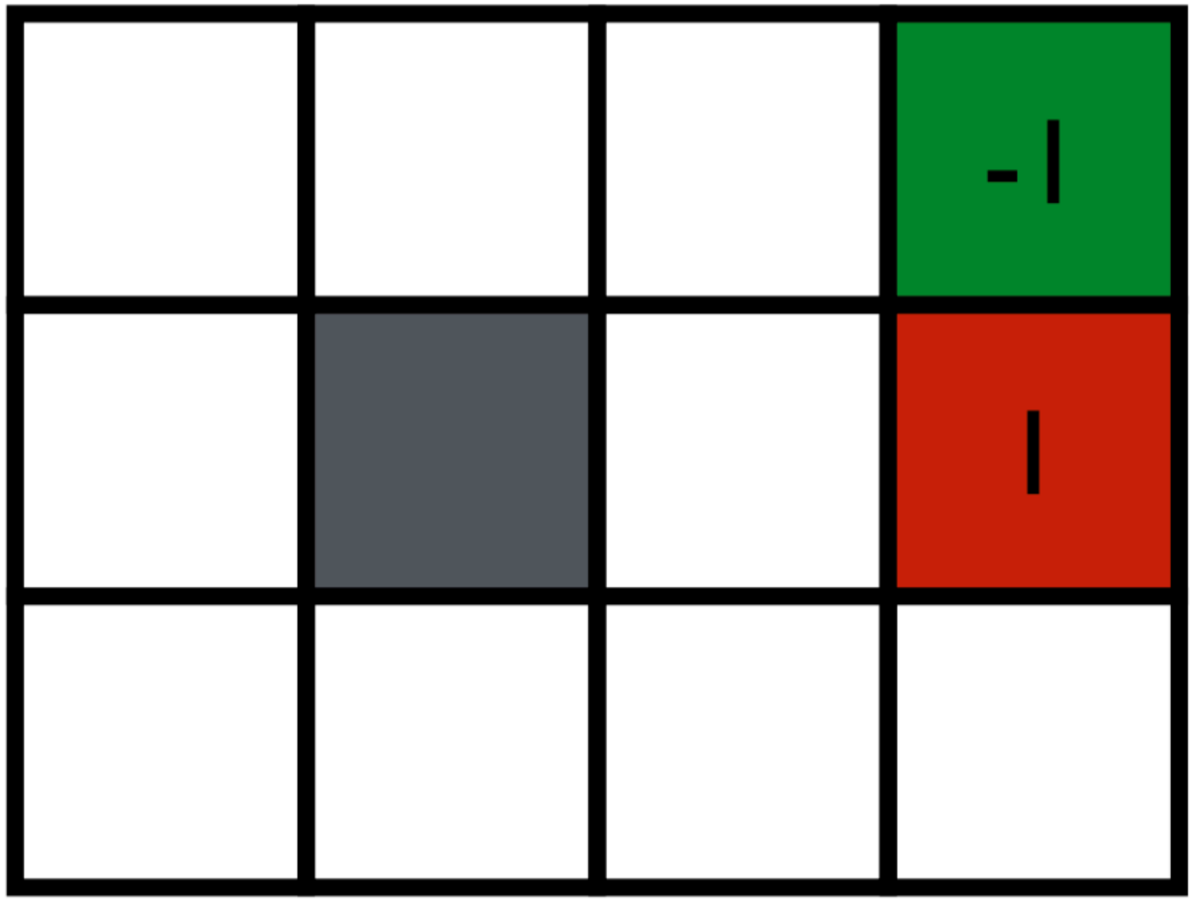
Start with an initial guess for V $\rightarrow V_0$

Iterate the Bellman Equation for each state S_t

$$V_{n+1}(S_t) = \min_{A_t}(C_t(S_t, A_t) + \gamma V_n(S_{t+1}))$$

One can show that $V_n \rightarrow V$ when $n \rightarrow \infty$

Canonical Example



Get out of the maze

- Red is bad (+1 cost)
- Green is good (-1 cost)
- Possible actions (N,E,W,S)
- $\gamma = 0.9$

Canonical Example

0	0	0	-1
0		0	1
0	0	0	0

- Initialize V_0

Canonical Example

0	0	-0.9	-1.9
0		0	1.9
0	0	0	0

- First iteration of Bellman (we update every state)

Canonical Example

0	-0.81	-1.71	-2.71
0		-0.81	2.71
0	0	0	0

- 2nd Iteration

Canonical Example

-0.73	-1.54	-2.44	-3.44
0		-1.54	3.44
0	0	-0.73	0

- 3rd Iteration

Canonical Example

-1.39	-2.2	-3.1	-4.1
-0.66		-2.2	4.1
0	-0.66	-1.39	-0.66

- 4th Iteration

Canonical Example

-4.15	-4.96	-5.86	-6.86
-3.42		-4.96	6.86
-2.77	-3.42	-4.15	-3.42

- 10th Iteration

Canonical Example

-7.29	-8.1	-9	-10
-6.56		-8.1	10
-5.9	-6.56	-7.29	-6.56

- 100th Iteration

Canonical Example

-7.29	-8.1	-9	-10
-6.56		-8.1	10
-5.9	-6.56	-7.29	-6.56

- 1000th Iteration

Canonical Example

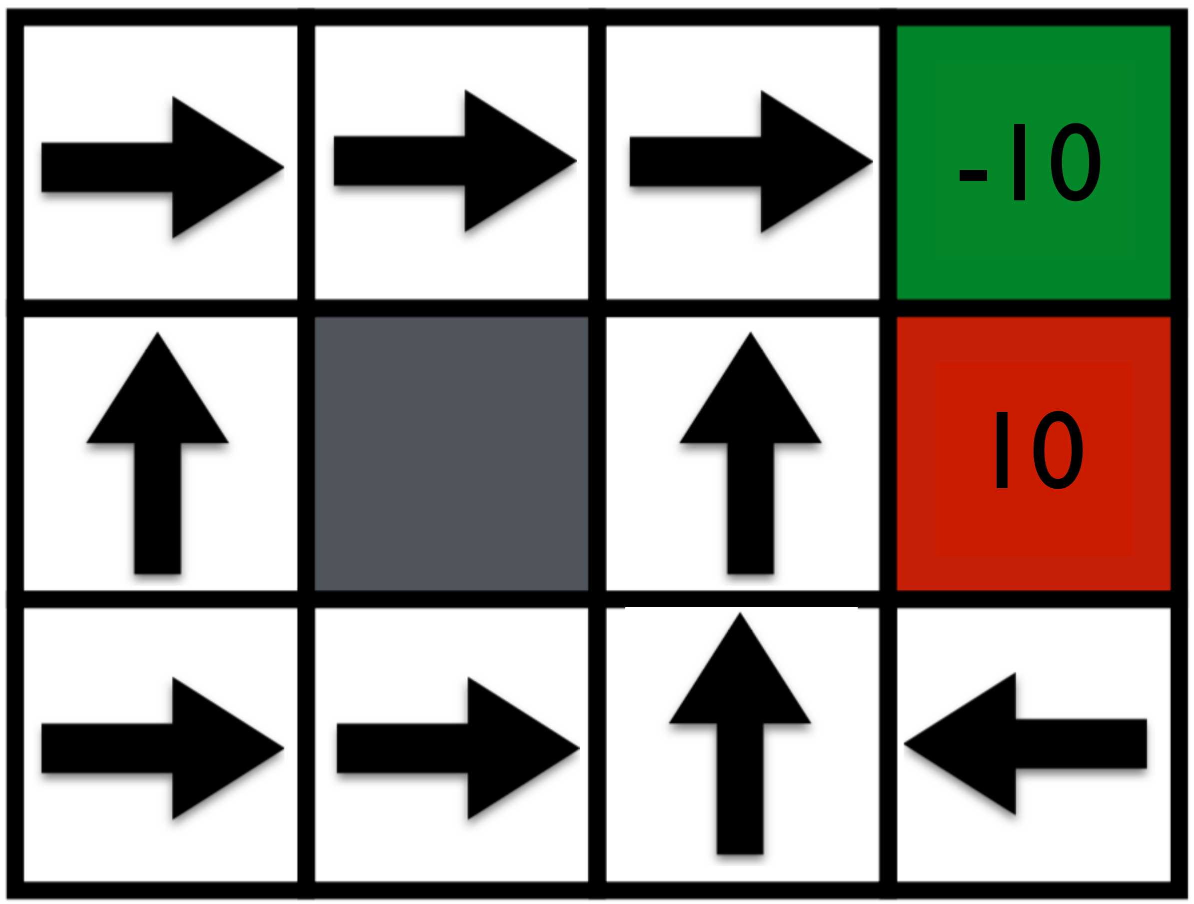
-7.29	-8.1	-9	-10
-6.56		-8.1	10
-5.9	-6.56	-7.29	-6.56

- 1000th Iteration

We have converged and found the optimal value function

The policy is read out by following the action that creates the lowest next value + current cost

Canonical Example



- 1000th Iteration

We have converged and found the optimal value function

The policy is read out by following the action that creates the highest next value

Policy Iteration

Start with an initial guess for the policy $\pi_0(S_t)$
and value function $V(S_t)$

1. Policy evaluation

   Iterate the Bellman Equation for each state S_t
   using the current policy for transition
   $$V_{n+1}(S_t) = C_t(S_t, \pi_k(S_t)) + \gamma V_n(S_{t+1})$$

2. Policy update

   Given the current guess for V find a new policy
   such that $\pi_{k+1}(S_t) = \arg\min C_t(S_t, A_t) + \gamma V(S_{t+1})$

What about learning?

Q-learning: value iteration with a twist

[Watkins 1989]

Introduce the Q-function $Q(S_t, A_t)$

which stores for each state/action pairs the cost of performing A_t and then using the optimal policy afterwards

$$Q(S_t, A_t) = C_t + \gamma V(S_{t+1}) \quad \Rightarrow \quad V(S_t) = \min_{A_t} Q(S_t, A_t)$$

Fortunately, the Q-function also satisfies Bellman's equation

$$Q(S_t, A_t) = C_t(S_t, A_t) + \gamma \min_{A_{t+1}} Q(S_{t+1}, A_{t+1})$$

Q-learning: value iteration with a twist

[Watkins 1989]

If we choose any random action A_t and observe what reward C_t we get then if our Q-function guess was correct we should have

$$Q(S_t, A_t) = C_t(S_t, A_t) + \gamma \min_{A_{t+1}} Q(S_{t+1}, A_{t+1})$$

So if this is not the case, then we can estimate "how wrong we are" and update Q accordingly

$$Q^{new}(S_t, A_t) \leftarrow (1-\alpha) Q(S_t, A_t) + \alpha (C_t(S_t, A_t) + \gamma \min_{A} Q(S_{t+1}, A))$$

α is the learning rate between 0 and 1

Q-learning: value iteration with a twist

[Watkins 1989]

When dealing with discrete state/action spaces (and moderate size for this space) we can store the Q-function as a table indexed by actions and states

Q-learning with a table

Choose $\alpha \in (0,1]$ and small $\epsilon > 0$ Initialize $Q(S,A)$ arbitrarily

For each episode:

Start from an initial state S_0

Loop for each step t of the episode:

Choose A_t from S_t using a policy using Q (e.g. ϵ-greedy policy)

Take action A_t and observe cost $C_t(S_t, A_t)$ and next state S_{t+1}

Update $Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha(C_t(S_t, A_t) + \gamma \min_A Q(S_{t+1}, A))$

Do until convergence

Practical Example

Break

Exercise on Q-learning with a table

Discretized states and actions

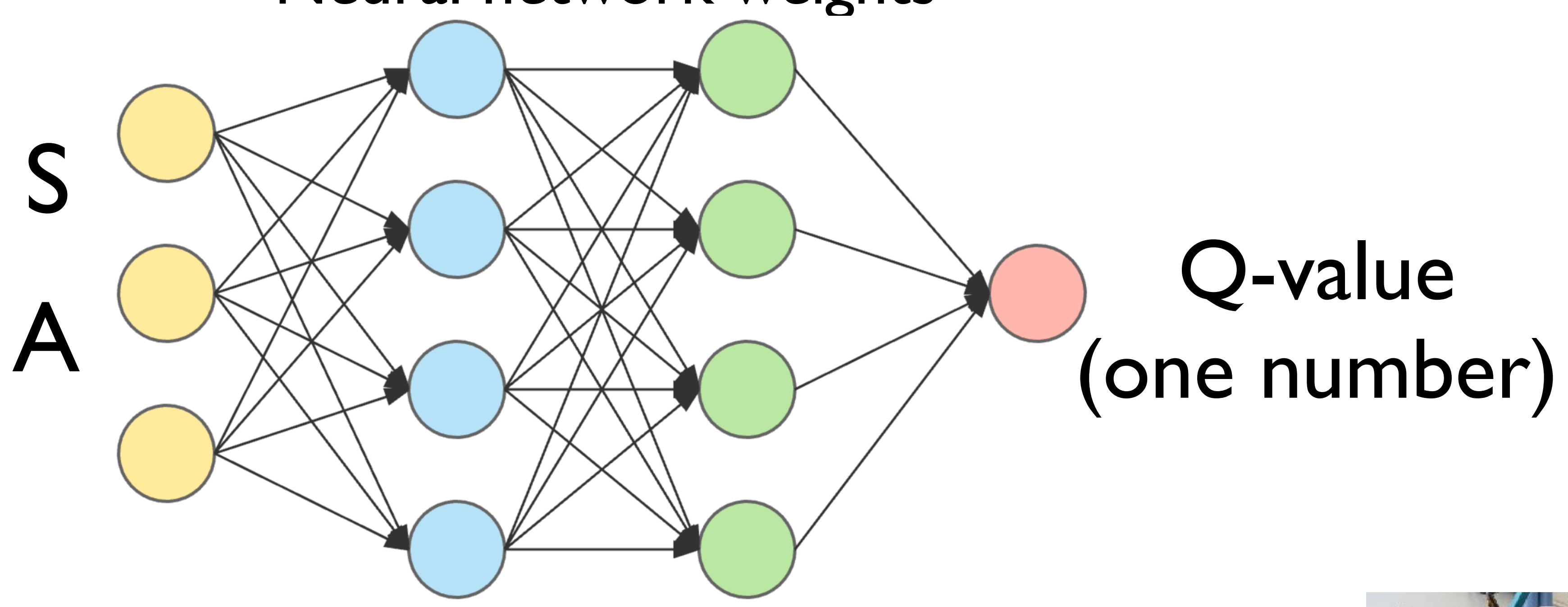
Q-learning with function approximation

Q-learning with a table cannot work for high-dimensional spaces nor for continuous state/action spaces!

Idea: replace the table with a function approximator (e.g. a neural network) - still assume discrete number of actions

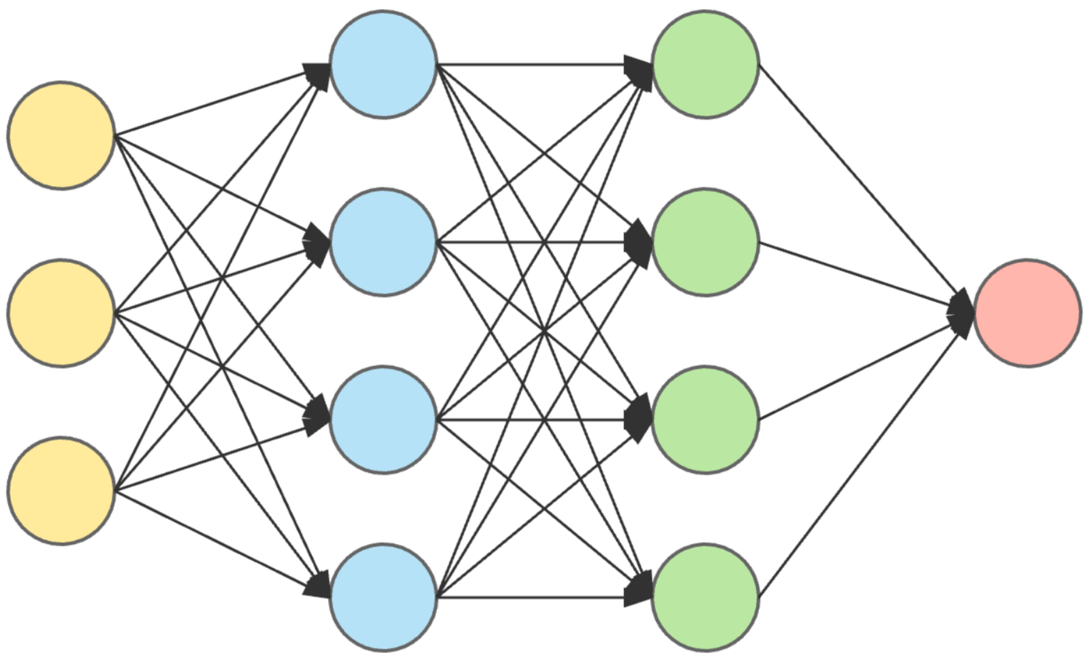
$$Q(S, A) \simeq Q(S, A, \theta)$$

Neural network weights



Q-learning with function approximation

$$Q(S, A) \simeq Q(S, A, \theta)$$



The problem can be written as a least square problem

$$\min_{\theta} ||Q(S_t, A_t, \theta) - C_t(S_t, A_t) - \gamma \min_{A_t} Q(S_{t+1}, A, \theta^{-})||^2$$

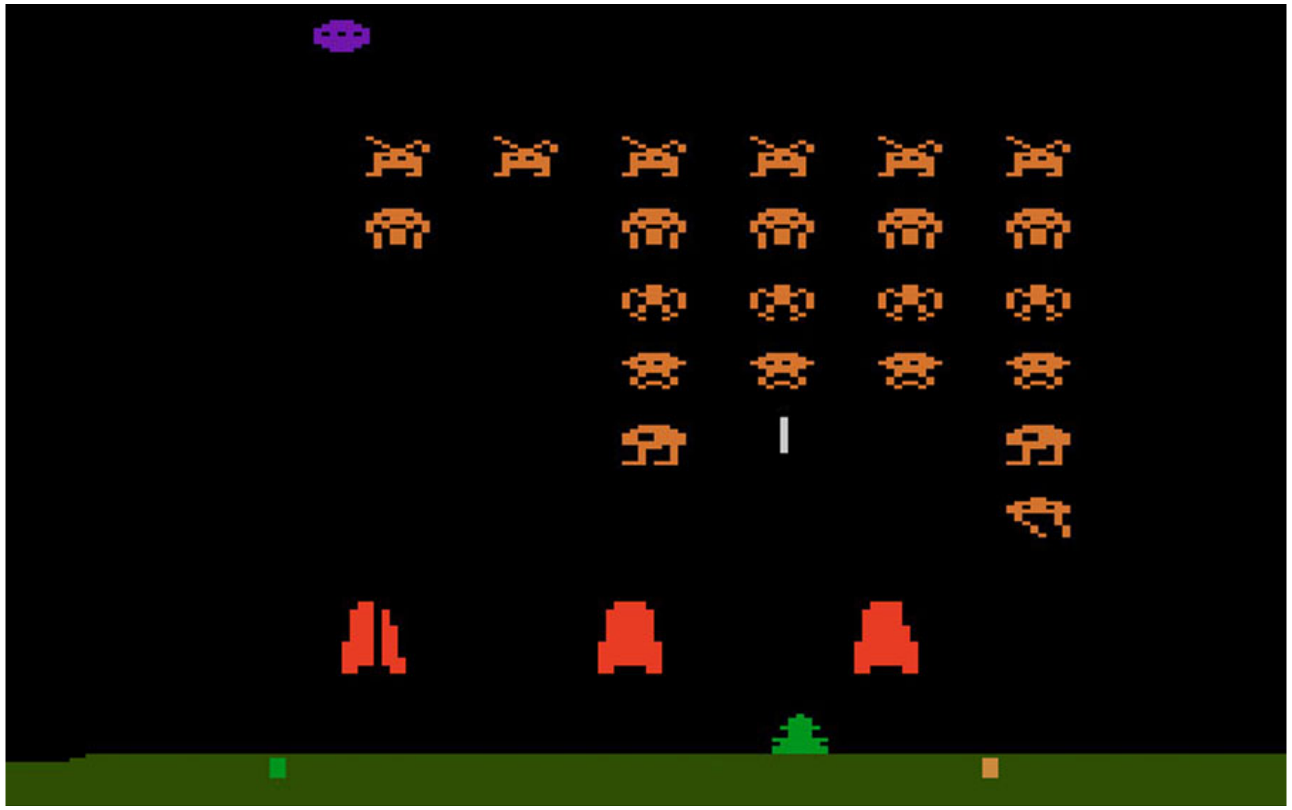
for set of observed (S_t, A_t, C_t, S_{t+1}) samples

Q-learning with function approximation

<u>Problem</u>: a direct (naive) approach using solely current episode data tend to be unstable (i.e. it diverges):

- The sequence of observations are correlated
- Small changes in Q can lead to large changes in policy

Deep Q-network (DQN)



[Mnih et al., Nature, 2015]

Deep Q-network (DQN)

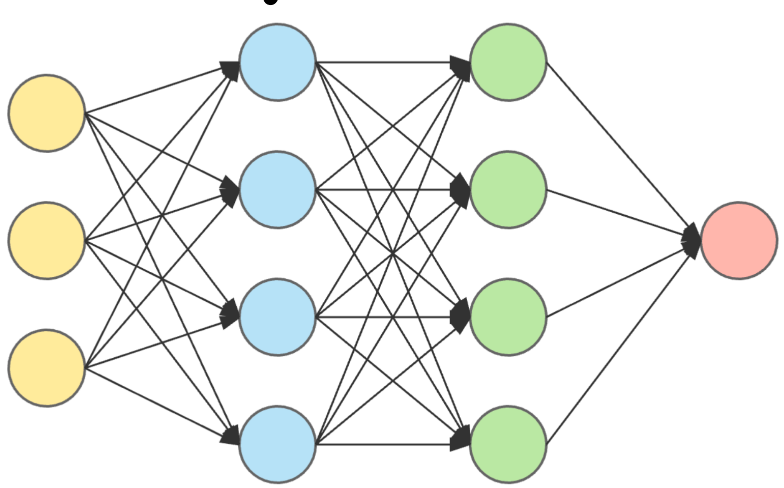
<u>Problem</u>: a direct (naive) approach using solely current episode data tend to be unstable (i.e. it diverges):

- The sequence of observations are correlated
- Small changes in Q can lead to large changes in policy

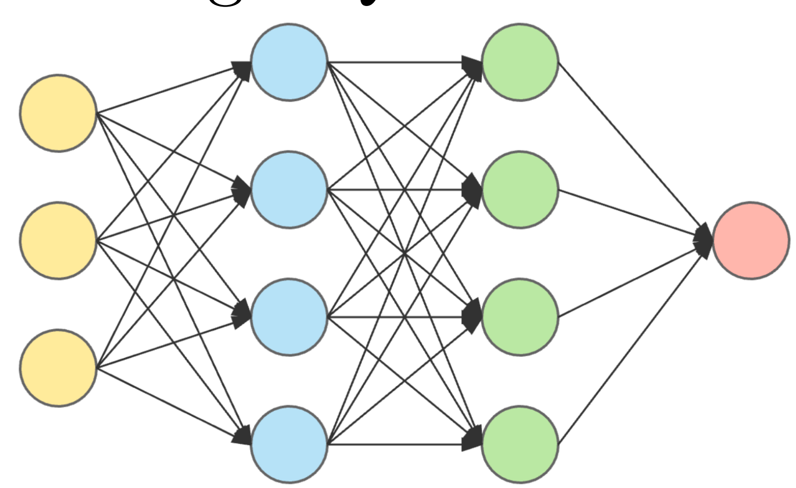
<u>Solution</u>

1) Use a "replay" memory of a previous samples from which we randomly sample the next training batch (remove correlations)
2) Use 2 Q-network to avoid correlations due to updates

Q-network



Target $\hat{Q}$-network



Deep Q-network (DQN)

[Mnih et al., Nature, 2015]

Initialize replay memory D of size N

Initialize Q-network with random weights θ

Initialize target $\hat{Q}$ function with weights $\theta^- = \theta$

For each episode:

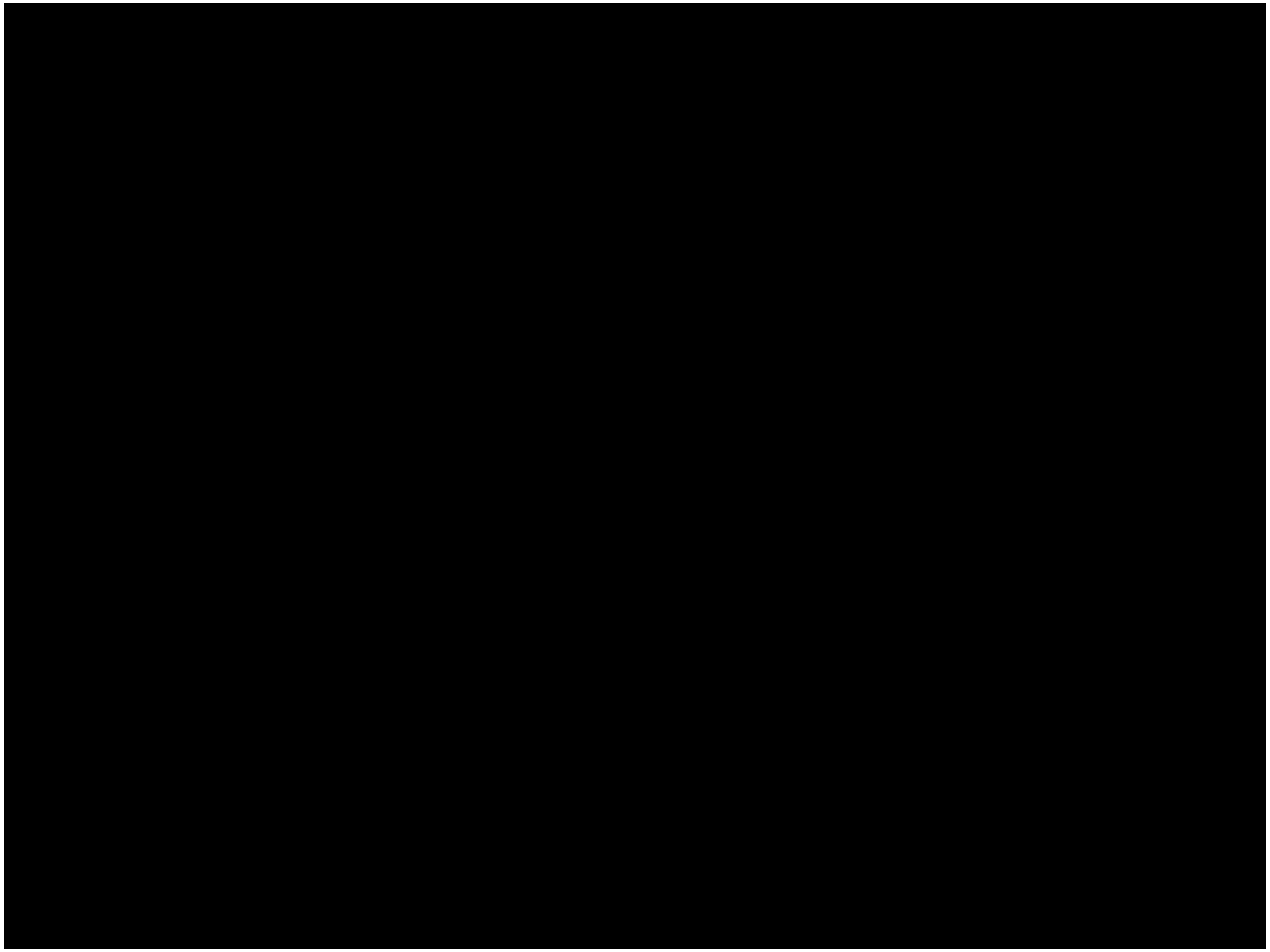
Start from an initial state S_0

Loop for each step t of the episode:

- Choose A_t from S_t using a policy using Q (e.g. ϵ-greedy policy)
- Take action A_t and observe cost $C_t(S_t, A_t)$ and next state S_{t+1}
- Store (S_t, A_t, C_t, S_{t+1}) in memory D
- Sample minibatch of transitions (S_j, A_j, C_j, S_{j+1}) from memory D
- Gradient descent on θ to minimize $||Q(S_j, A_j, \theta) - C_j - \gamma \min_A \hat{Q}(S_{j+1}, A, \theta^-)||^2$
- Every C steps reset $\theta^- = \theta$

Deep Q-network (DQN)

[Mnih et al., Nature, 2015]



Deep Q-network (DQN)

[Mnih et al., Nature, 2015]

Practical Example

Break

Let's replace the table with a neural network

1) Just replace Q-table with the NN
2) DQN (replay memory, target, etc)

What about continuous action space?

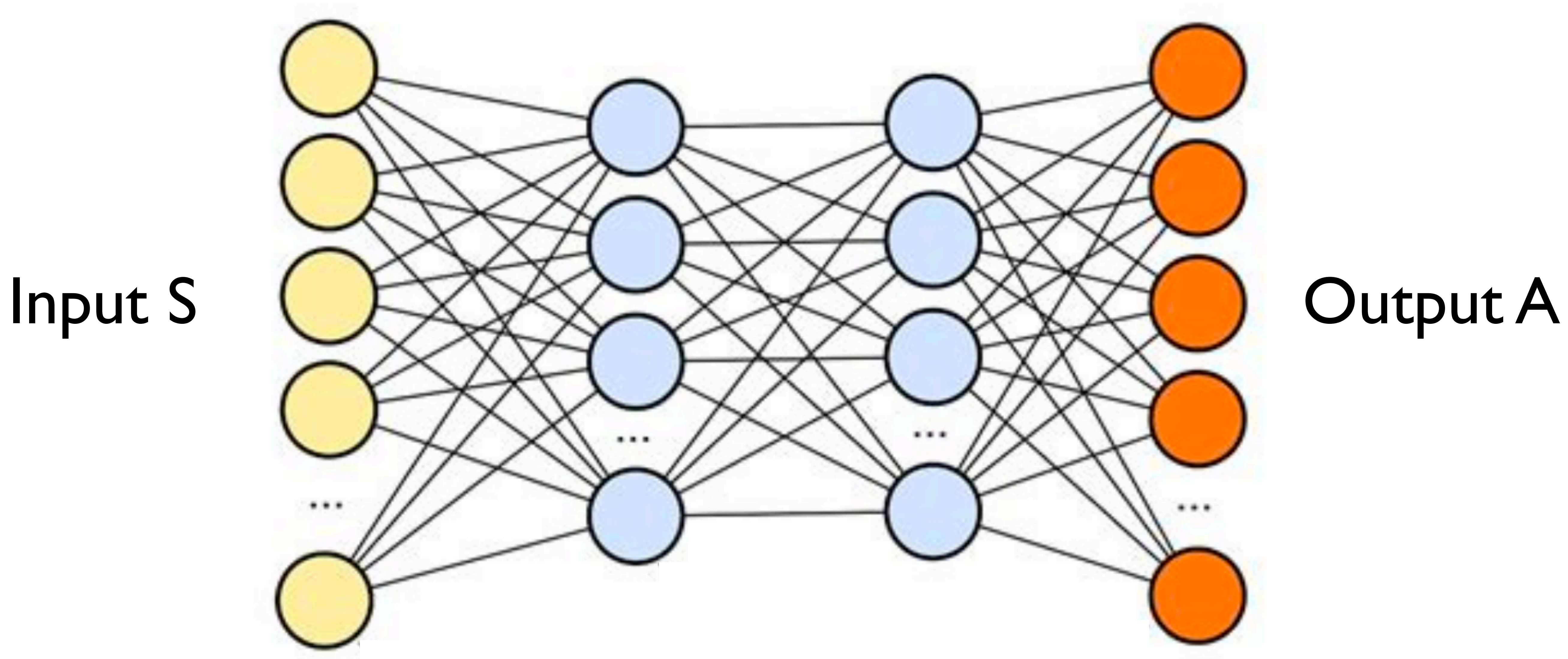
Problem: we need to evaluate the min to be able to do Q-learning with a function approximator

$$||Q(S_j, A_j, \theta) - C_j - \gamma \min_A \hat{Q}(S_{j+1}, A, \theta^-)||^2$$

Solution: use another neural network to approximate the min operator (i.e. to approximate the optimal policy)
=> Actor-critic algorithm

First we need a gradient for the policy

Let $\pi(S_t, \theta^\pi)$ an approximation of a policy with a NN (weights θ^π)



First we need a gradient for the policy

Let $\pi(S_t, \theta^\pi)$ an approximation of a policy with a NN (weights θ^π)

Let $J_\pi(S_t)$ be the value function under policy π

We would like to find the gradient $\nabla_{\theta^\pi} J_\pi$ to improve π

The policy gradient theorem (cf. Sutton-Barto book) tells us that

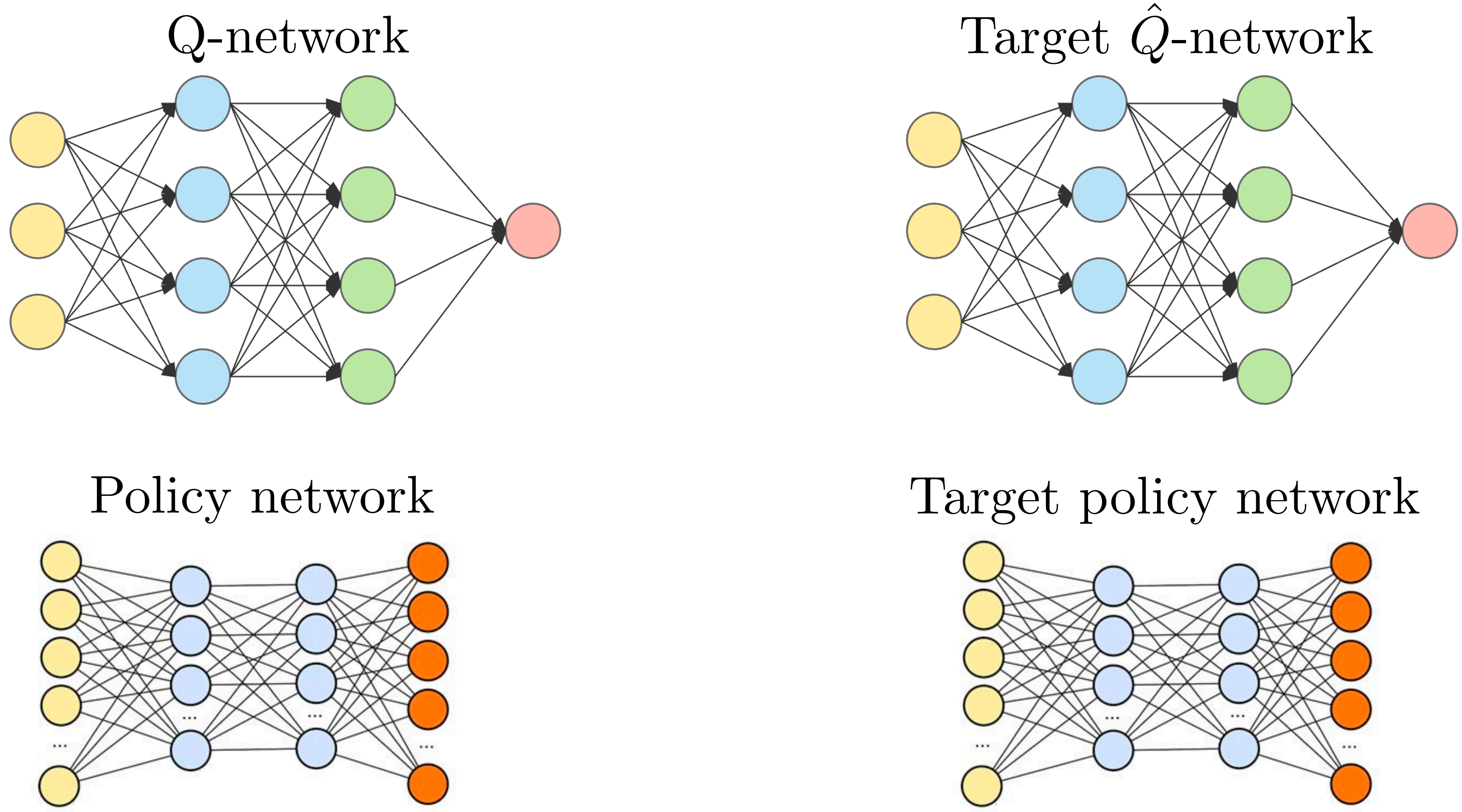
$$\nabla_{\theta^\pi} J \simeq \frac{1}{N} \sum_i \nabla_A Q(S, A, \theta^Q)|_{S=S_i, A=\pi(S_i)} \nabla_{\theta^\pi} \pi(S_i, \theta^\pi)$$

=> we can do gradient descent on the policy parameters to minimize the value function

DDPG (Deep Deterministic Policy Gradient)

[Lillicrap et al., ICML, 2016]

Policy network (actor) - Q-network (critic)
DDPG => Same as DQN + policy network



DDPG (Deep Deterministic Policy Gradient)

[Lillicrap et al., ICML, 2016]

Initialize replay memory D of size N
Initialize Q- and policy networks with random weights θ^Q and θ^π
Set target networks $\theta^{Q'} = \theta^Q$ and $\theta^{\pi'} = \theta^\pi$

For each episode:

Start from an initial state S_0

Loop for each step t of the episode:

Choose $A_t = \pi(S_t) + noise$ (to explore a bit)

Take action A_t and observe cost $C_t(S_t, A_t)$ and next state S_{t+1}

Store (S_t, A_t, C_t, S_{t+1}) in memory D

Sample minibatch of transitions (S_j, A_j, C_j, S_{j+1}) from memory D

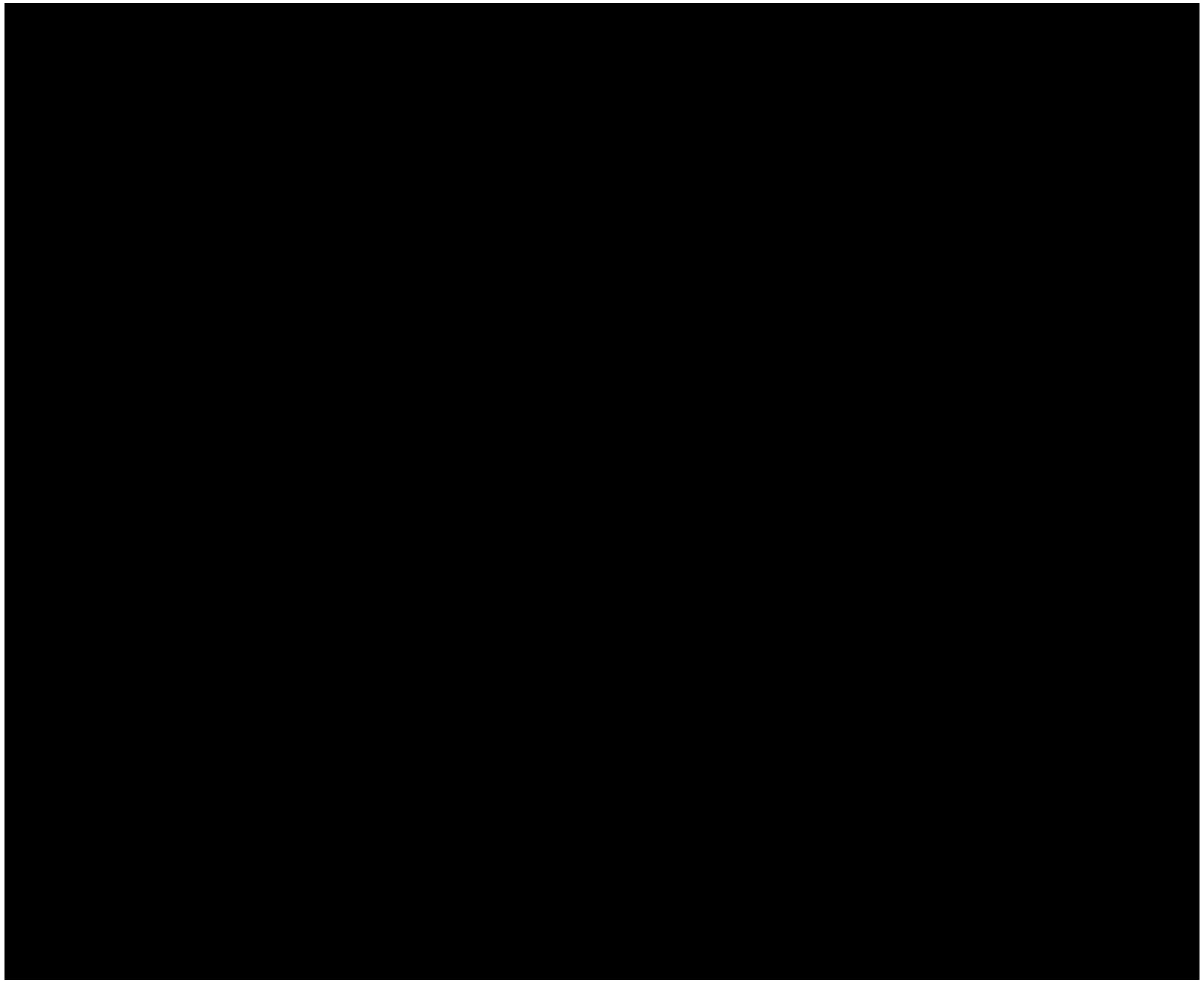
Gradient descent on θ^Q to minimize $||Q(S_j, A_j, \theta^Q) - C_j - \gamma Q'(S_{j+1}, \pi'(S_{j+1}))||^2$

Policy update $\nabla_{\theta^\pi} J \simeq \frac{1}{N} \sum_i \nabla_A Q(S, A, \theta^Q)|_{S=S_i, A=\pi(S_i)} \nabla_{\theta^\pi} \pi(S_i, \theta^\pi)$

Smooth update of target networks
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\pi'} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi'}$$

DDPG (Deep Deterministic Policy Gradient)

[Lillicrap et al., ICML, 2016]



In a nutshell…

All the presented algorithms are all variations on the same theme:
Use Bellman Equations to find iterative algorithms

Severe limitations still exist:

- Algorithms difficult to tune to ensure convergence
- Need lots of samples (not practical on real robots)
- Not clear how to efficiently explore
- Robots can break during learning
- Does not generalize (fixed policy/Q-function)

In a nutshell…

Learn only the Q-function (Q-learning) => DQN (Atari games)

Learn Q-function + policy function (actor-critic) => DDPG

Very long history of actor-critic algorithms in robotics:

[Doya, Neural Computation 2000]

Natural Actor-Critic [Peters et al. 2008]

Learn directly policy (use policy gradient) => TRPO or PPO

Also long history of policy gradient methods:

REINFORCE [Williams, 1992]

[Doya, Neural Computation 2000]

PI2 [Theodorou et al., JMLR 2010]

Practical Example

Break

Let’s add a critic and test DDPG