Indian Institute of Information Technology, Allahabad





Introduction to Machine Learning

Introduction to Reinforcement Learning



By

Dr. Shiv Ram Dubey
Associate Professor
Computer Vision and Biometrics Lab
Department of Information Technology
Indian Institute of Information Technology, Allahabad
Web: https://profile.iiita.ac.in/srdubey/

DISCLAIMER

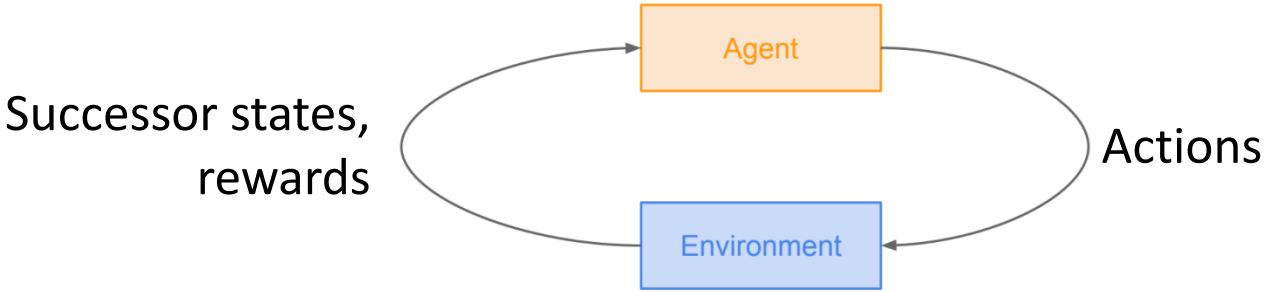
The content (text, image, and graphics) used in this slide are adopted from many sources for academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.

Outline

- Introduction to Reinforcement Learning
- Markov Decision Process (MDP) formalism
- The Bellman Equation
- Q-learning
- Deep Q Networks (DQN)

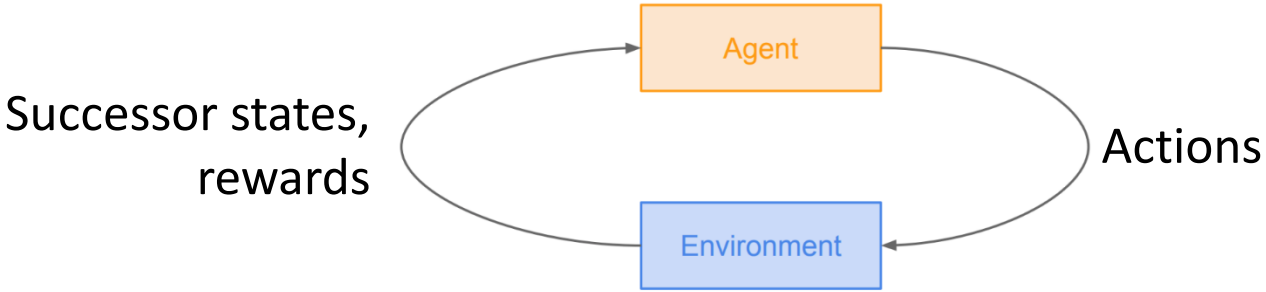
Reinforcement learning (RL)

- Setting: agent that can take *actions* affecting the *state* of the environment and observe occasional *rewards* that depend on the state



Reinforcement learning (RL)

- Setting: agent that can take *actions* affecting the *state* of the environment and observe occasional *rewards* that depend on the state
- Goal: learn a *policy* (mapping from states to actions) to maximize expected reward over time



RL vs. Supervised Learning

- **Reinforcement learning loop**
  - From state s, take action a determined by *policy* $\pi(s)$
  - Environment selects next state s' based on *transition model* $P(s'|s,a)$
  - Observe s' and reward $r(s')$, update policy

- **Supervised learning loop**
  - Get input x_i sampled from data distribution
  - Use model with parameters w to predict output y
  - Observe target output y_i and loss $l(w, x_i, y_i)$
  - Update w to reduce loss: $w \leftarrow w - \eta \nabla l(w, x_i, y_i)$

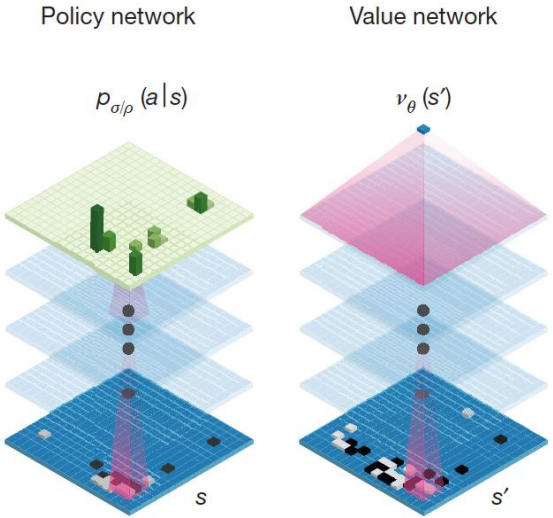
RL vs. Supervised Learning

- **Reinforcement learning loop**
  - Agent's actions affect the environment and help to determine next observation
  - Rewards may be sparse
  - Rewards are not differentiable w.r.t. model parameters
- **Supervised learning loop**
  - Next input does not depend on previous inputs or agent's predictions
  - There is a supervision signal at every step
  - Loss is differentiable w.r.t. model parameters

Example Applications of Deep RL

- AlphaGo and AlphaZero

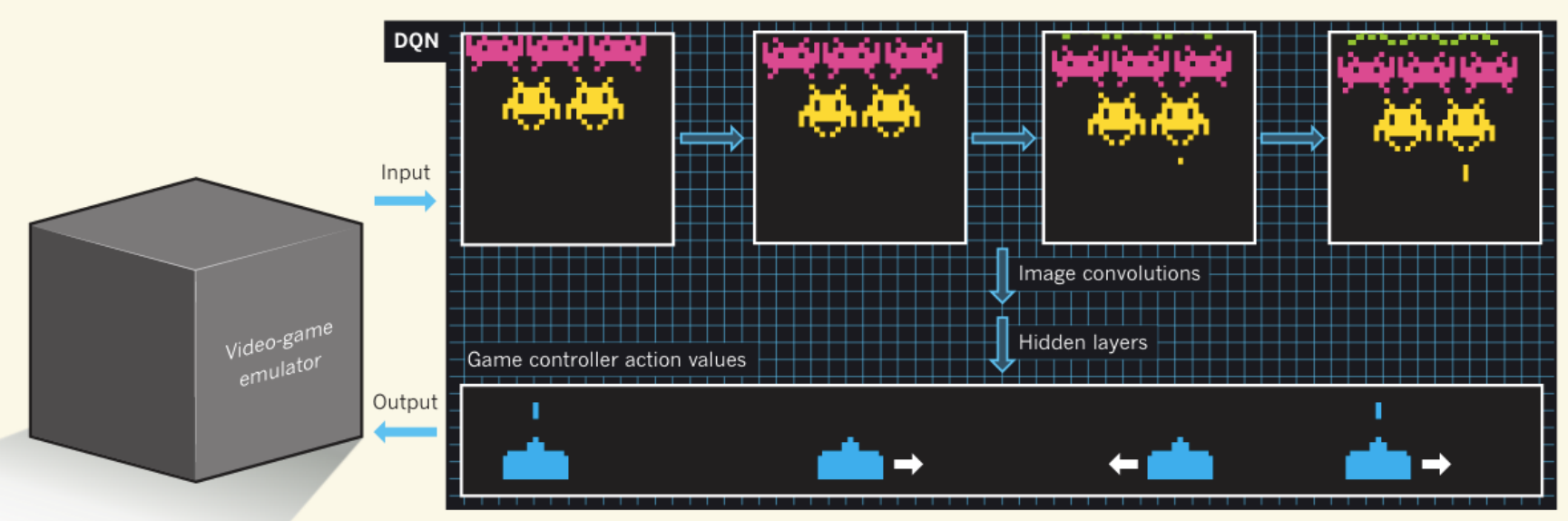




https://deepmind.com/research/alphago/

Example Applications of Deep RL

- Playing video games



V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Human-level control through deep reinforcement learning, *Nature* 2015

Example Applications of Deep RL

- Sensorimotor learning

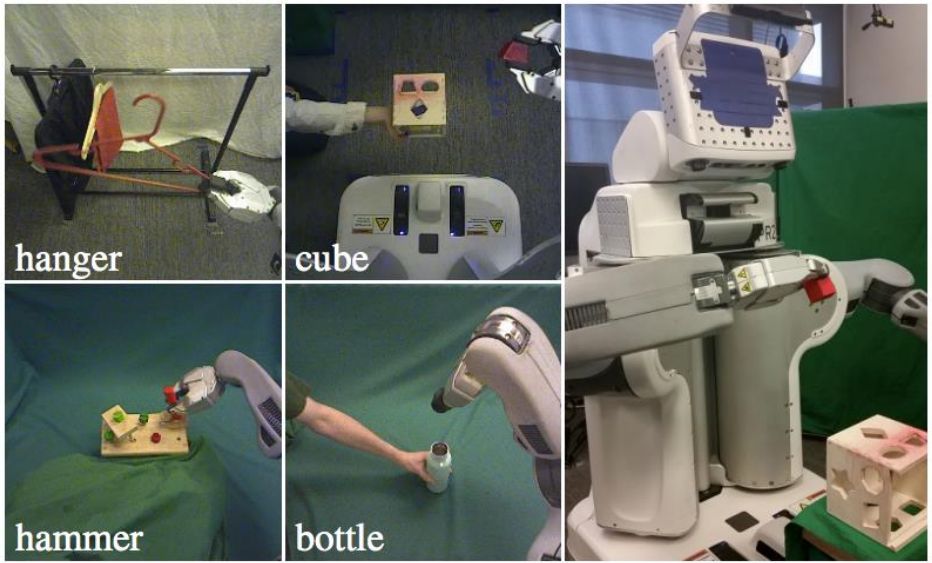


Fig. 1: Our method learns visuomotor policies that directly use camera image observations (left) to set motor torques on a PR2 robot (right).

S. Levine, C. Finn, T. Darrell and P. Abbeel, End-to-End Training of Deep Visuomotor Policies, JMLR 2016

Example Applications of Deep RL

- Autonomous Driving



(a) Urban driving



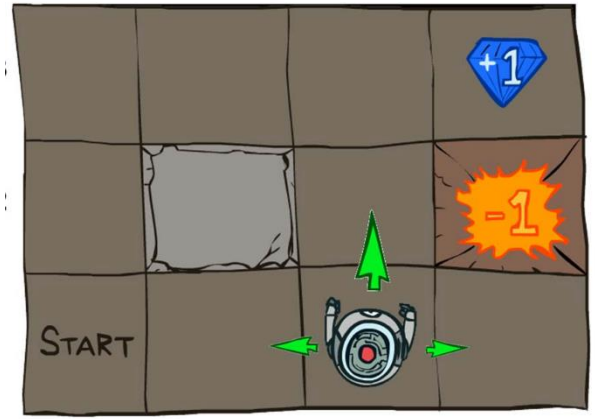
(b) Off-road traversal

B. Zhou, P. Krähenbühl, and V. Koltun, Does Computer Vision Matter for Action? Science Robotics, 4(30), 2019

Formalism: Markov Decision Processes

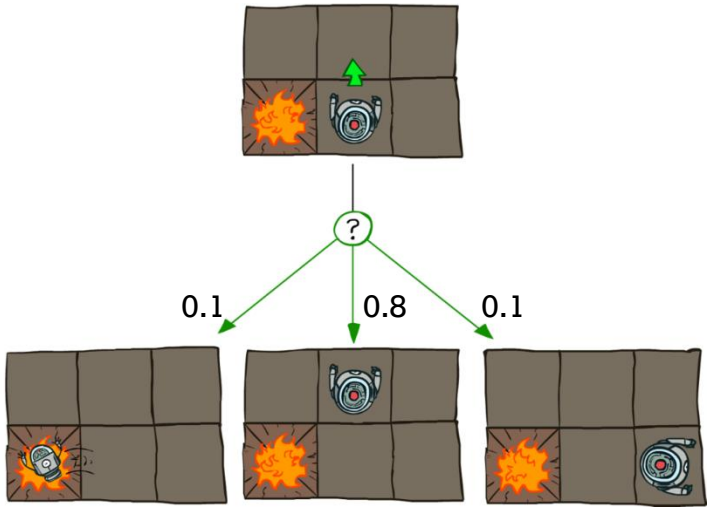
- Components:
  - **States** s, beginning with initial state s_0
  - **Actions** a
  - **Transition model** $P(s' \mid s, a)$
    - *Markov assumption*: the probability of going to s' from s depends only on s and a and not on any other past actions or states
  - **Reward function** $r(s)$
- **Policy** $\pi(s)$: the action that an agent takes in any given state
  - The "solution" to an MDP

Example MDP: Grid World



$r(s) = -0.04$ for every non-terminal state

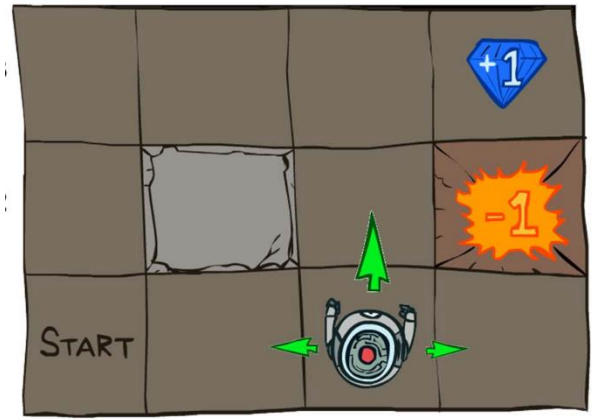
Transition model:

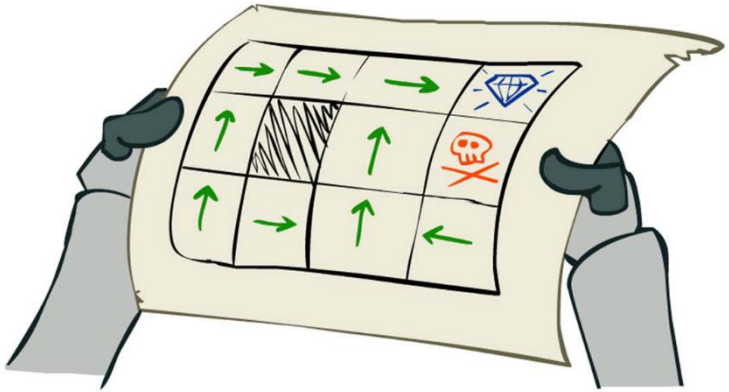


Source: P. Abbeel and D. Klein

Example MDP: Grid World

- Goal: find the best policy





Source: P. Abbeel and D. Klein

Rewards of State Sequences

- Suppose that following policy π starting in state s_0 leads to a sequence $s_0, s_1, s_2, \ldots$
- The *cumulative reward* of the sequence is $\sum_{t \geq 0} r(s_t)$
- **Problem:** state sequences can vary in length or even be infinite
- **Solution:** redefine cumulative reward as sum of rewards *discounted* by a factor γ



Image source: P. Abbeel and D. Klein

Discounting

- Discounted cumulative reward:

$$r(s_0) + \gamma\, r(s_1) + \gamma^2 r(s_2) + \gamma^3 r(s_3) + \cdots$$
$$= \sum_{t \geq 0} \gamma^t\, r(s_t)\,, \qquad 0 < \gamma \leq 1$$

- Sum is bounded by $\frac{r_{\max}}{1-\gamma}$
- Helps algorithms converge

Value Function

- The *value function* $V^{\pi}(s)$ of a state s w.r.t. policy π is the expected cumulative reward of following that policy starting in s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t \, r(s_t) \mid s_0 = s, \pi\right]$$

with $a_t = \pi(s_t), s_{t+1} \sim P\left(\cdot \mid s_t, a_t\right)$

Value Function

- The *value function* $V^{\pi}(s)$ of a state s w.r.t. policy π is the expected cumulative reward of following that policy starting in s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t \, r(s_t) \mid s_0 = s, \pi\right]$$

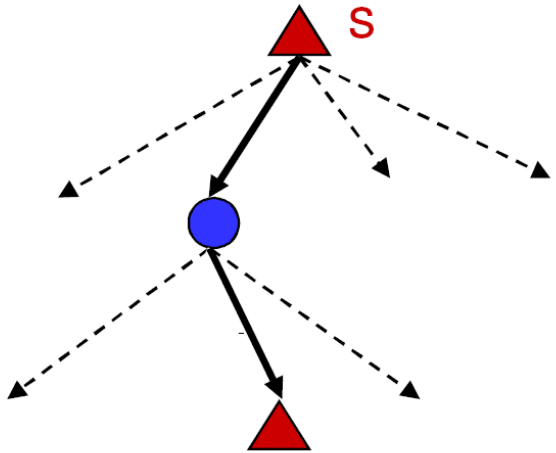
with $a_t = \pi(s_t), s_{t+1} \sim P\left(\cdot \mid s_t, a_t\right)$

- The ***optimal value*** of a state is the value achievable by following the best possible policy:

$$V^{*}(s) = \max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t \, r(s_t) \mid s_0 = s, \pi\right]$$

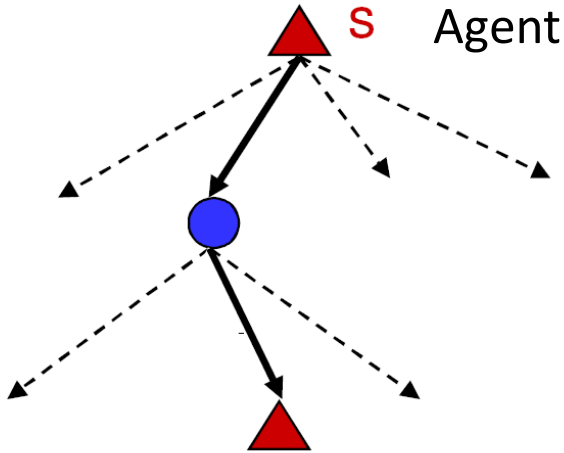
The Bellman Equation

- Recursive relationship between optimal values of successive states:



The Bellman Equation

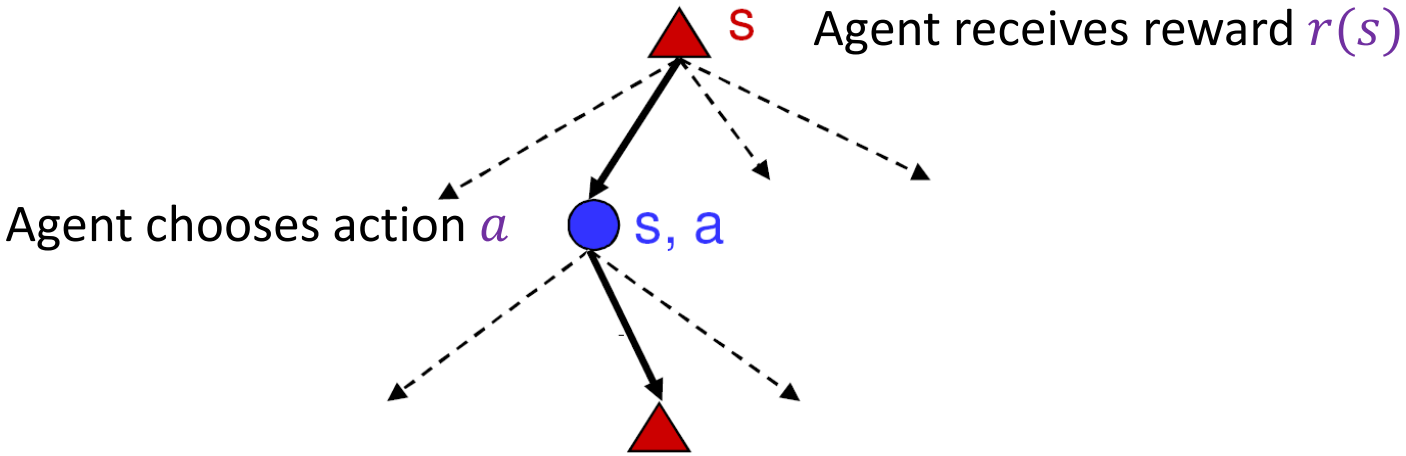
- Recursive relationship between optimal values of successive states:



Agent receives reward $r(s)$

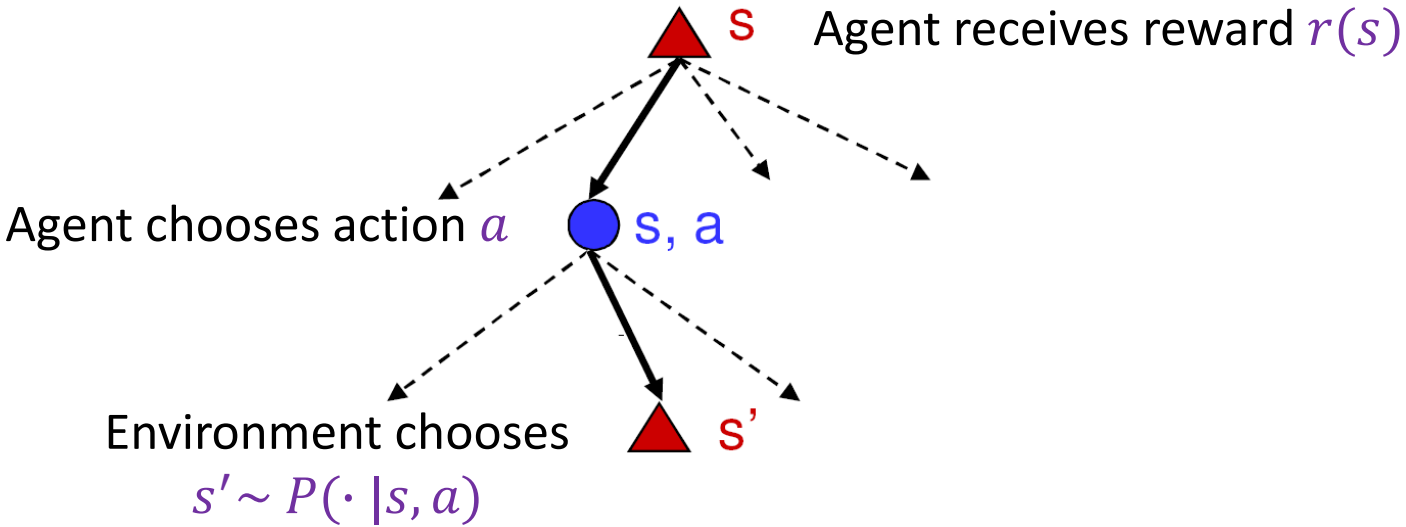
The Bellman Equation

- Recursive relationship between optimal values of successive states:



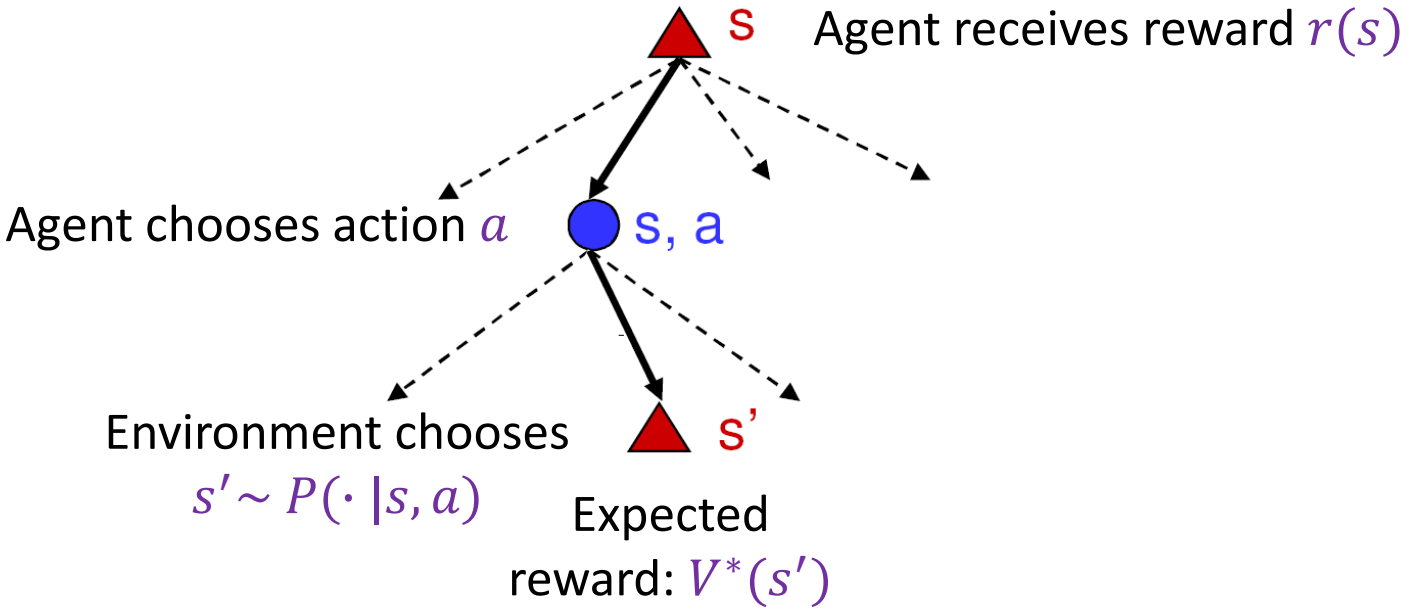
The Bellman Equation

- Recursive relationship between optimal values of successive states:



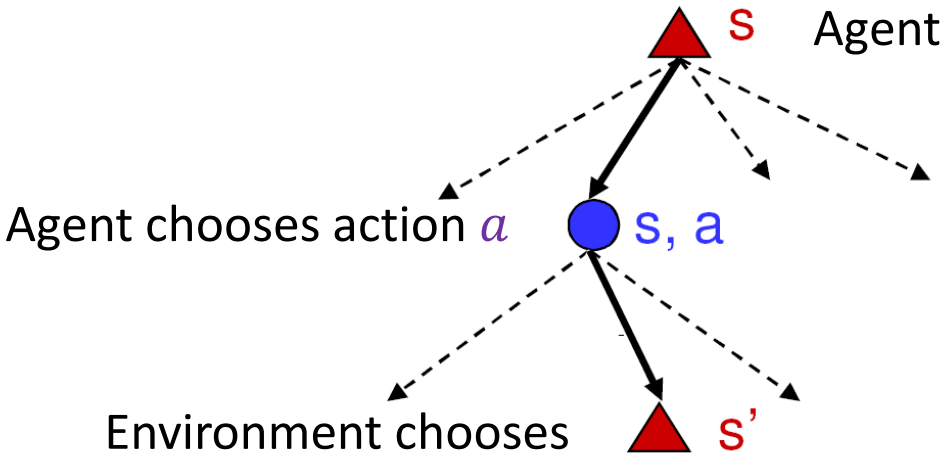
The Bellman Equation

- Recursive relationship between optimal values of successive states:



The Bellman Equation

- Recursive relationship between optimal values of successive states:

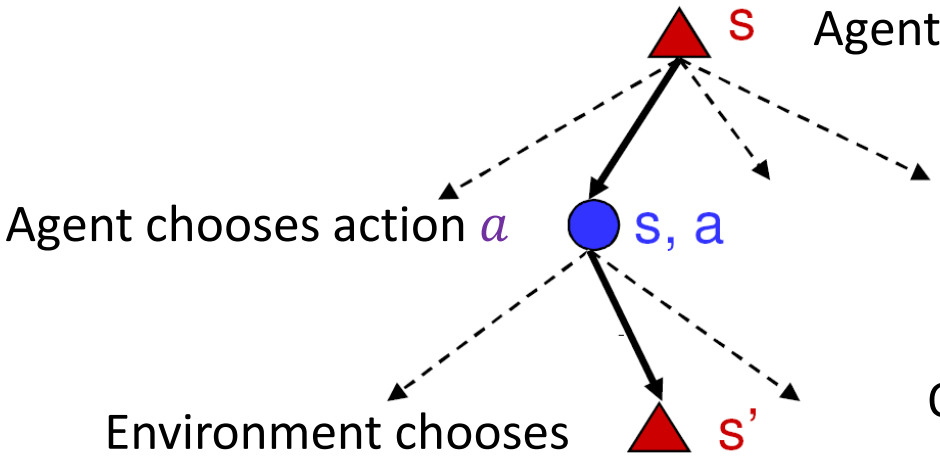


Expected value for action a:

$$\sum_{s'} P(s'|s,a)V^*(s')$$

The Bellman Equation

- Recursive relationship between optimal values of successive states:



Expected value for action a:

$$\sum_{s'} P(s'|s,a)V^*(s')$$

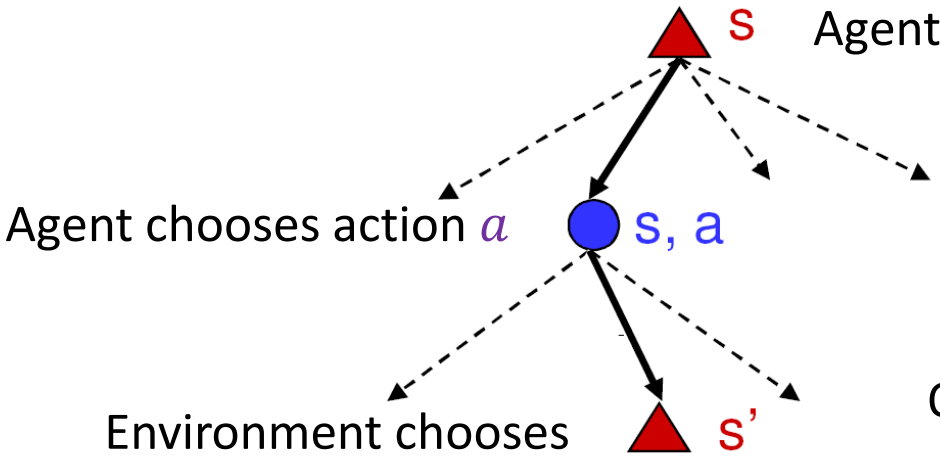
Optimal policy:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)V^*(s')$$

The Bellman Equation

- Recursive relationship between optimal values of successive states:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V^*(s')$$



Agent receives reward $r(s)$

Agent chooses action a

Expected value for action a:

$$\sum_{s'} P(s'|s,a)V^*(s')$$

Environment chooses

$s' \sim P(\cdot\,|s,a)$

Expected reward: $V^*(s')$

Optimal policy:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)V^*(s')$$

The Bellman Equation

- Recursive relationship between optimal values of successive states:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V^*(s')$$

Reward in current state

Discounted expected future reward assuming agent follows the optimal policy

Value Function: Issue

- Optimal policy in terms of the state value function:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)V^*(s')$$

To use this in practice, we need to know the transition model

Value Function: Issue

- Optimal policy in terms of the state value function:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)V^*(s')$$

To use this in practice, we need to know the transition model

Solution: Q-learning

- It is more convenient to define the value of a *state-action pair*:

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t\, r(s_t)\,|s_0 = s, a_0 = a, \pi\right]$$

Q-value function

- The *optimal* Q-value:

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t\, r(s_t) \mid s_0 = s, a_0 = a, \pi\right]$$

Q-value function

- The *optimal* Q-value:

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t\, r(s_t) \mid s_0 = s, a_0 = a, \pi\right]$$

- The relationship between $V^*(s)$ and $Q^*(s,a)$:

$$V^*(s) = \max_a Q^*(s,a)$$

- The optimal policy:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

Bellman Equation for Q-values

$$V^*(s) = \max_a Q^*(s,a)$$

- Regular Bellman Equation:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$$

- Bellman Equation for Q-values:

$$Q^*(s,a) = r(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$$

$$= \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[r(s) + \gamma \max_{a'} Q^*(s',a') | s,a \right]$$

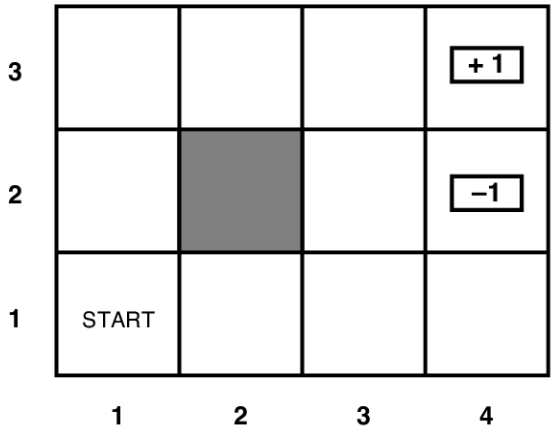
Finding the optimal policy

- The Bellman equation is a constraint on Q-values of successive states:

$$Q^*(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[r(s) + \gamma \max_{a'} Q^*(s',a') | s,a\right]$$

- We could think of $Q^*(s,a)$ as a table indexed by states and actions, and try to solve the system of Bellman equations to fill in the unknown values of the table

Q-value function: Example



(1,1)	(1,2)	(1,3)	+1
(2,1)	WALL	(2,3)	−1
START	(3,2)	(3,3)	(3,4)

Q-value function: Example

Parameter	Value
Actions	Up, Down, Left, Right
Intended direction succeeds	80%
Each perpendicular direction	10%
Reward per nonterminal transition	-0.04
Positive terminal	$+1$
Negative terminal	-1
Discount factor	$\gamma = 0.9$
Initial Q-values	$Q_0(s, a) = 0$

Q-value function: Example

Q-iteration formula

For each nonterminal state s and action a:

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a) \left[r(s,a,s') + 0.9 \max_{a'} Q_k(s',a') \right]$$

Define the maximum action value:

$$V_k(s) = \max_a Q_k(s,a)$$

Then the update can be written as:

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[r + 0.9 V_k(s')]$$

Q-value function: Example

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[r + 0.9V_k(s')]$$

$$V_k(s) = \max_a Q_k(s,a)$$

Iteration 1: Calculate the first Q-values

Since $Q_0(s,a) = 0$, the future-value term is initially zero.

For the **cell (1,3)**, taking action **Right**:

$$Q_1\big((1,3),R\big) = 0.8(1) + 0.1(-0.04) + 0.1(-0.04)$$
$$= 0.792$$

Q-value function: Example

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[r + 0.9V_k(s')]$$

$$V_k(s) = \max_a Q_k(s,a)$$

Iteration 1: Calculate the first Q-values

Since $Q_0(s,a) = 0$, the future-value term is initially zero.

For the **cell (1,3)**, taking action **Right**:

$$Q_1((1,3),R) = 0.8(1) + 0.1(-0.04) + 0.1(-0.04) = 0.792$$

Thus, the maximum action values after Iteration 1 are: $V_1(s) = \max_a Q_1(s,a)$

Iteration 1

−0.040	−0.040	0.792	+1
−0.040	WALL	−0.040	−1
−0.040	−0.040	−0.040	−0.040

Q-value function: Example

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[r + 0.9V_k(s')]$$

$$V_k(s) = \max_a Q_k(s,a)$$

Iteration 2: Propagate the discounted future reward

For the **cell (1,2)**, taking action **Right**:

$$\begin{aligned} Q_2\big((1,2),R\big) = {} & 0.8(-0.04 + 0.9(0.792)) \\ & +0.1(-0.04 + 0.9(-0.04)) \\ & +0.1(-0.04 + 0.9(-0.04)) = 0.52304 \end{aligned}$$

Q-value function: Example

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[r + 0.9V_k(s')]$$

$$V_k(s) = \max_a Q_k(s,a)$$

Iteration 2: Propagate the discounted future reward

For the **cell (1,2)**, taking action **Right**:

$$Q_2((1,2),R) = 0.8(-0.04 + 0.9(0.792)) + 0.1(-0.04 + 0.9(-0.04)) + 0.1(-0.04 + 0.9(-0.04)) = 0.52304$$

The complete maximum-value grid after Iteration 2 is:

Iteration 2

−0.076	0.523	0.860	+1
−0.076	WALL	0.431	−1
−0.076	−0.076	−0.076	−0.076

Notice how the positive value has propagated from (1,3) to (1,2), and the effect is beginning to spread through other cells.

Q-value function: Example

Converged values and optimal policy: After repeated updates, the maximum Q-values converge approximately to the following values:

0.611	0.766	0.928	+1
0.487	WALL	0.585	−1
0.374	0.327	0.428	0.189

Q-value function: Example

Converged values and optimal policy: After repeated updates, the maximum Q-values converge approximately to the following values:

0.611	0.766	0.928	+1
0.487	WALL	0.585	−1
0.374	0.327	0.428	0.189

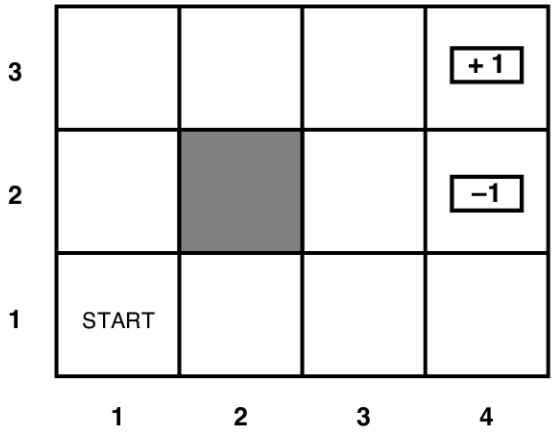
Hence, the approximate optimal policy is:

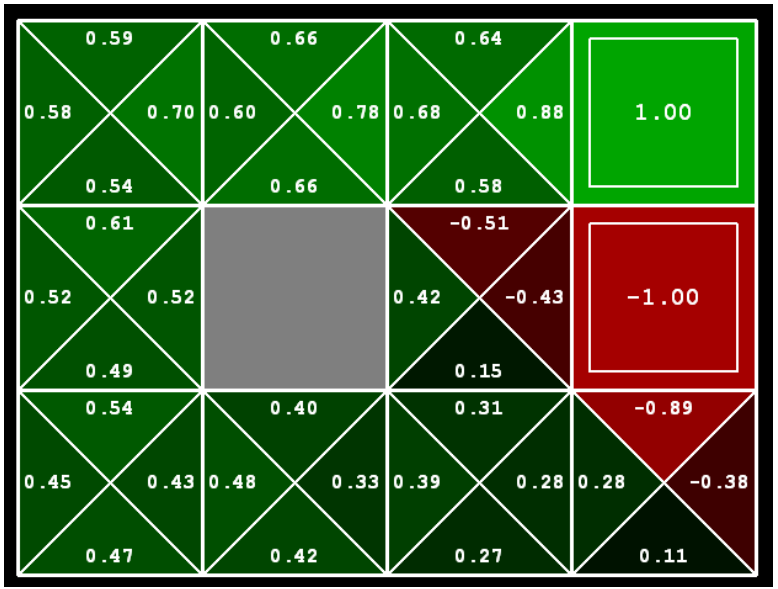
→	→	→	+1
↑	WALL	↑	−1
↑	→	↑	←

Arrows indicate the action with the highest converged Q-value.

Q-value function

- $Q^*(s,a) = \max_\pi \mathbb{E}[\sum_{t\geq 0} \gamma^t \, r(s_t) \mid s_0 = s, a_0 = a, \pi]$
- $\pi^*(s) = \arg\max_a Q^*(s,a)$





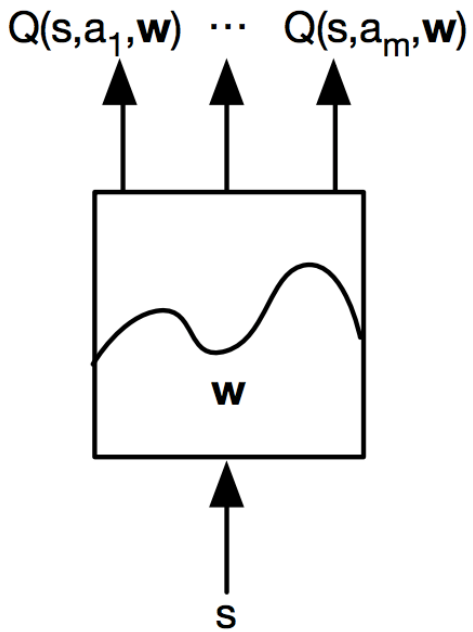
Finding the optimal policy

- **Problem:** state spaces for interesting problems are huge
- **Solution:** approximate Q-values using a parametric function:

$$Q^*(s, a) \approx Q_w(s, a)$$

Deep Q-learning

- Train a deep neural network to estimate Q-values:



Source: D. Silver

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Human-level control through deep reinforcement learning, *Nature* 2015

Deep Q-learning

$$Q^*(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[r(s) + \gamma \max_{a'} Q^*(s',a')|s,a\right]$$

- Idea: at each iteration i of training, update model parameters w_i to "nudge" the left-hand side toward the right-hand "target":

$$y_i(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[r(s) + \gamma \max_{a'} Q_{w_{i-1}}(s',a')|s,a\right]$$

- Loss function:

$$L_i(w_i) = \mathbb{E}_{s,a \sim \rho}\left[(y_i(s,a) - Q_{w_i}(s,a))^2\right]$$

where ρ is a *behavior distribution*

Deep Q-learning

Target: $y_i(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[r(s) + \gamma \max_{a'} Q_{w_{i-1}}(s',a') | s,a \right]$

Loss: $L_i(w_i) = \mathbb{E}_{s,a \sim \rho} \left[(y_i(s,a) - Q_{w_i}(s,a))^2 \right]$

- Gradient update:

$$\begin{aligned} \nabla_{w_i} L(w_i) &= \mathbb{E}_{s,a \sim \rho} \left[(y_i(s,a) - Q_{w_i}(s,a)) \nabla_{w_i} Q_{w_i}(s,a) \right] \\ &= \mathbb{E}_{s,a \sim \rho, s'} \left[(r(s) + \gamma \max_{a'} Q_{w_{i-1}}(s',a') \right. \\ &\quad \left. - Q_{w_i}(s,a)) \nabla_{w_i} Q_{w_i}(s,a) \right] \end{aligned}$$

- SGD training: replace expectation by sampling *experiences* (s,a,s') using behavior distribution and transition model

Deep Q-learning in Practice

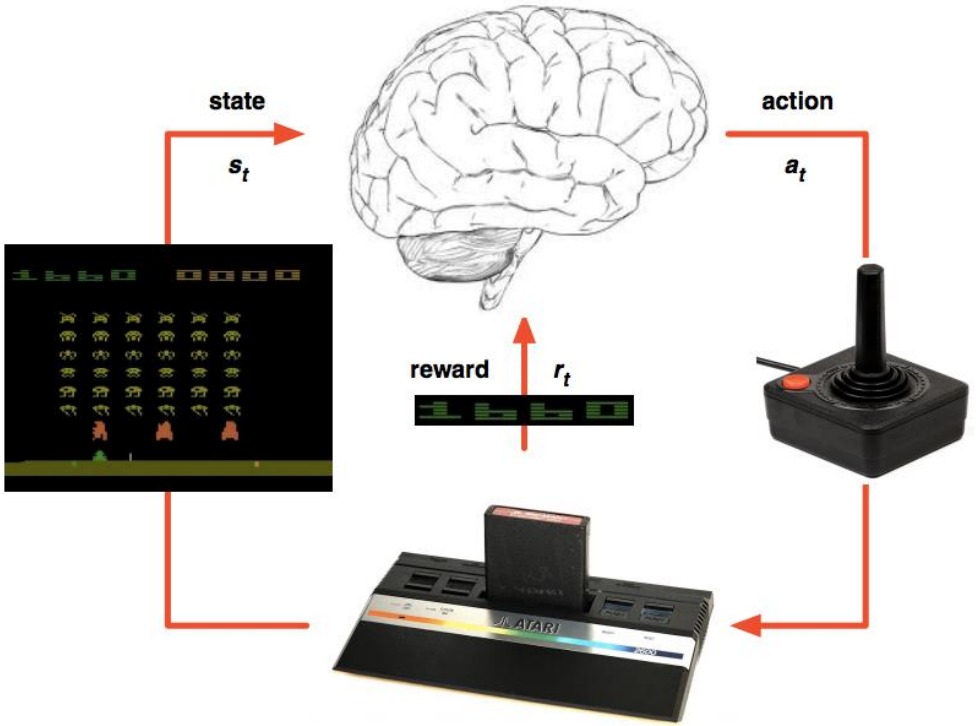
- Training is prone to instability
  - Unlike in supervised learning, the targets themselves are moving!
  - Successive experiences are correlated and dependent on the policy
  - Policy may change rapidly with slight changes to parameters, leading to drastic change in data distribution
- Solutions
  - Freeze target Q network
  - Use *experience replay*

Experience Replay

- At each time step:
  - Take action a_t according to *epsilon-greedy policy*
  - Store experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in *replay memory buffer*
  - Randomly sample *mini-batch* of experiences from the buffer

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

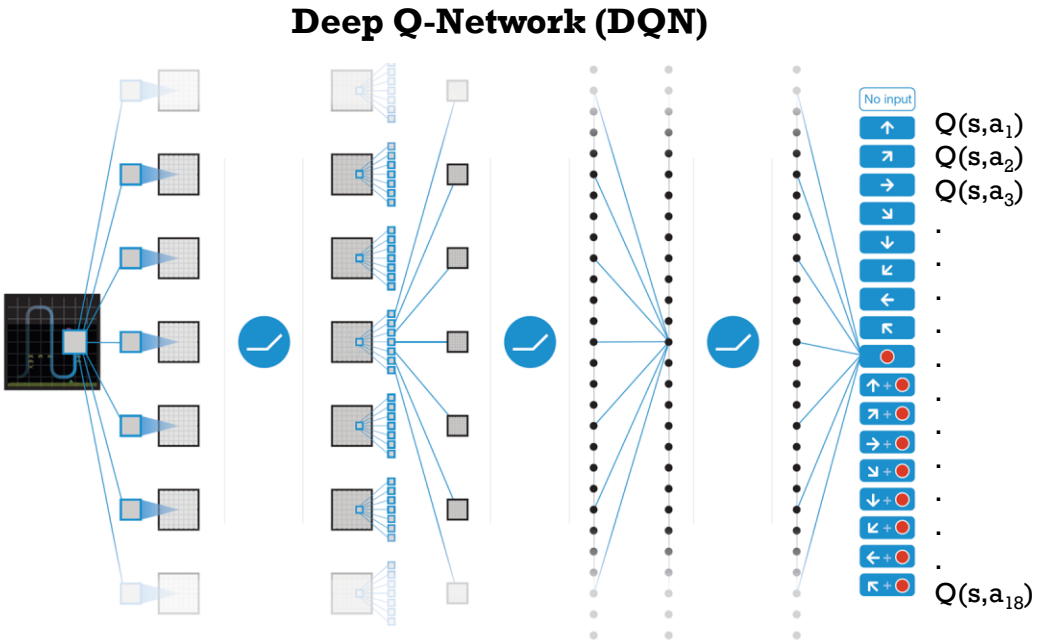
Deep Q-learning in Atari



V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller,
Human-level control through deep reinforcement learning, *Nature* 2015

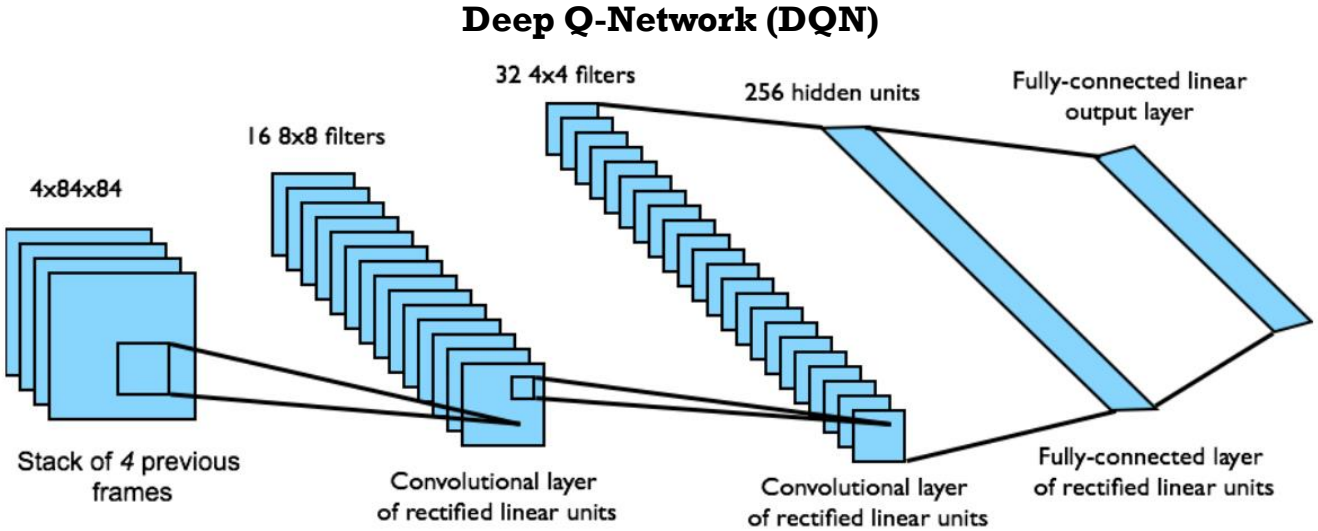
Deep Q-learning in Atari

- End-to-end learning of $Q(s,a)$ from pixels s
- Output is $Q(s,a)$ for 18 joystick/button configurations
- Reward is change in score for that step



Deep Q-learning in Atari

- Input state is stack of raw pixels (grayscale) from last 4 frames
- Network architecture and hyperparameters fixed for all games



Breakout Demo



https://www.youtube.com/watch?v=TmPfTpjtdgg

Acknowledgement

Most of the slides are based on the ML/DL course of:

- Deep Learning, Stanford University
- Introduction to Deep Learning, University of Illinois at Urbana-Champaign
- Introduction to Deep Learning, Carnegie Mellon University
- Convolutional Neural Networks for Visual Recognition, Stanford University
- NVDIEA Deep Learning Teaching Kit
- And many more…

THANK YOU