Deep Reinforcement Learning

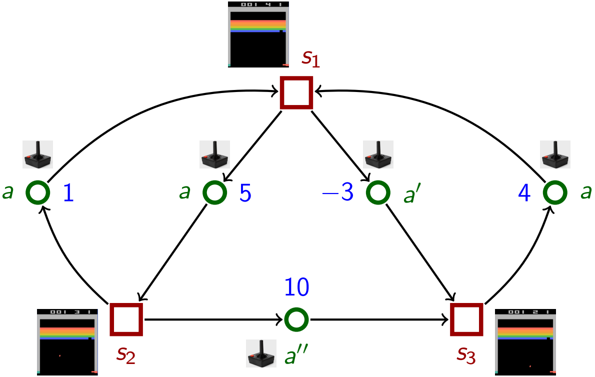
Tomáš Brázdil

2016

Based on V. Mnih et al, Human-level control through deep reinforcement learning. Nature (2015).

Left: `https://commons.wikimedia.org/wiki/File:Atari2600a.JPG`, **Right**: `http://www.opobotics.com/`.

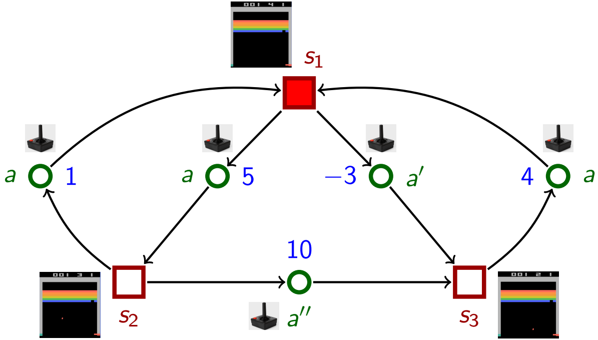
Deterministic Markov Decision Processes



- set of states S,
- set of actions A,
  each state is assigned a set of **enabled** actions,
- transition function $\delta : S \times A \to S$,
- reward function $R : S \times A \to \mathbb{R}$.

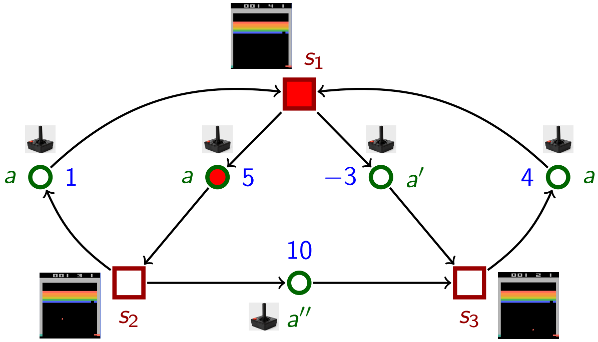
Deterministic Markov Decision Processes

Policy π chooses actions based on the current state.



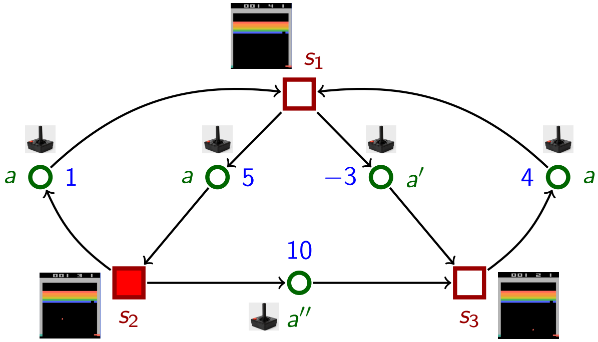
Deterministic Markov Decision Processes

Policy π chooses actions based on the current state.



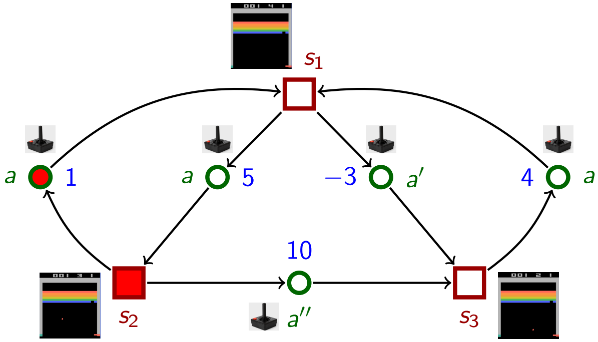
Deterministic Markov Decision Processes

Policy π chooses actions based on the current state.



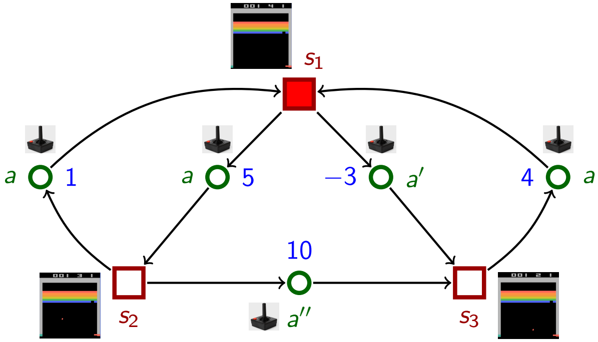
Deterministic Markov Decision Processes

Policy π chooses actions based on the current state.



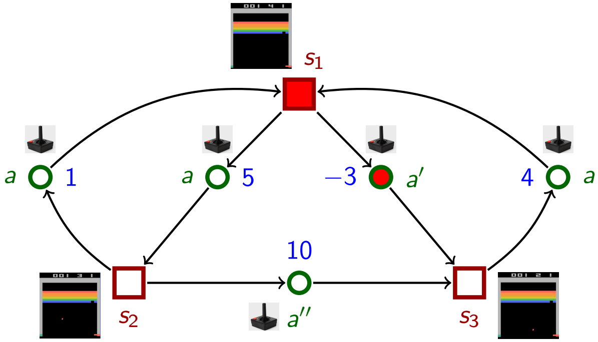
Deterministic Markov Decision Processes

Policy π chooses actions based on the current state.



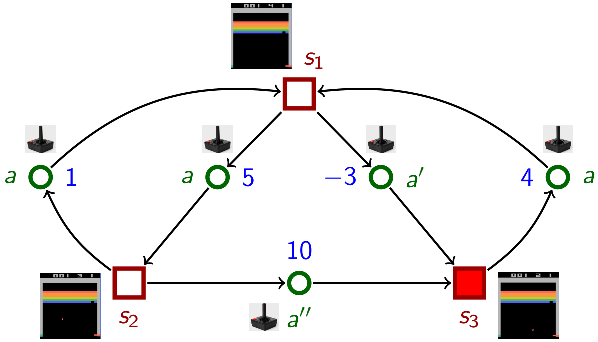
Deterministic Markov Decision Processes

Policy π chooses actions based on the current state.



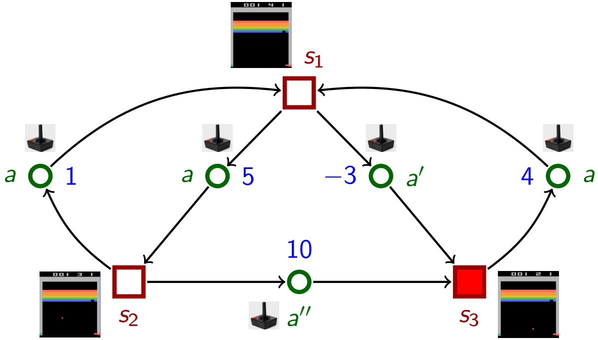
Deterministic Markov Decision Processes

Policy π chooses actions based on the current state.



Deterministic Markov Decision Processes

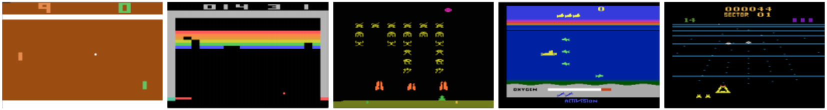
Policy π chooses actions based on the current state.



Notation:

- $S_1, S_2, \ldots$ where S_t is the t-th visited state
- $A_1, A_2, \ldots$ where A_t is the t-th taken action
- $R_1, R_2, \ldots$ where R_t is the t-th obtained reward

Encoding Atari games



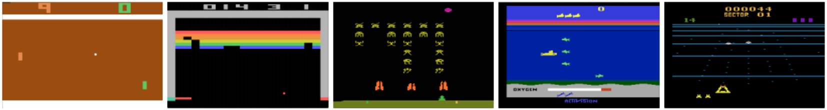
States correspond to preprocessed screenshots.

Original screenshots: 210×160 in 128 colors.
Preprocessing:

- Rescale and crop to 80×80,
- convert to gray-scale,
- use 4 most recent frames in a single state.

The states: Real vectors of dimension $80 \times 80 \times 4$.

Encoding Atari games



States correspond to preprocessed screenshots.

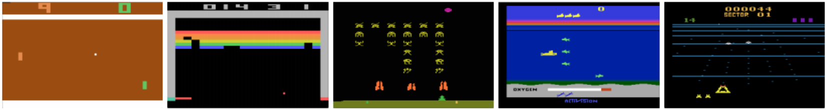
Original screenshots: 210×160 in 128 colors.
Preprocessing:

- Rescale and crop to 80×80,
- convert to gray-scale,
- use 4 most recent frames in a single state.

The states: Real vectors of dimension $80 \times 80 \times 4$.

Actions correspond to actions of the player: joystick position, position of fire buttons.

Encoding Atari games



States correspond to preprocessed screenshots.

Original screenshots: 210×160 in 128 colors.
Preprocessing:

- Rescale and crop to 80×80,
- convert to gray-scale,
- use 4 most recent frames in a single state.

The states: Real vectors of dimension $80 \times 80 \times 4$.

Actions correspond to actions of the player: joystick position, position of fire buttons.

Rewards correspond to changes in the game score.

Sqeezed into three values: 1 for positive, -1 for negative, 0 for 0.

Return and Value Functions

Definition

Return G *is the total discounted reward* $G = \sum_{k=0}^{\infty} \gamma^k R_{k+1}$.
Here $0 < \gamma < 1$ *is a* discount factor.

Return and Value Functions

Definition

Return *G is the total discounted reward $G = \sum_{k=0}^{\infty} \gamma^k R_{k+1}$.*
Here $0 < \gamma < 1$ is a discount factor*.*

Definition

Action-value function *$q_\pi(s, a)$ is*
return starting from state s, taking action a, and then following π.

Return and Value Functions

Definition

Return G *is the total discounted reward* $G = \sum_{k=0}^{\infty} \gamma^k R_{k+1}$.
Here $0 < \gamma < 1$ *is a* discount factor.

Definition

Action-value function $q_\pi(s, a)$ *is*
return starting from state s, taking action a, and then following π.

Optimal action-value function $q_*(s, a)$ *is the* maximum action-value function *over all policies:*

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

Return and Value Functions

Definition

Return G *is the total discounted reward* $G = \sum_{k=0}^{\infty} \gamma^k R_{k+1}$.
Here $0 < \gamma < 1$ *is a* discount factor.

Definition

Action-value function $q_\pi(s, a)$ *is*
return starting from state s**, taking action** a**, and then following** π.

Optimal action-value function $q_*(s, a)$ *is the* maximum action-value function *over all policies:*

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

Theorem

Define a policy π_* *which in every* $s \in S$ *chooses* $a \in A$ *so that*

$$q_*(s, a) = \max_{a'} q_*(s, a')$$

Then for all $s \in S$ *and* $a \in A$ *we have that* $q_{\pi_*}(s, a) = q_*(s, a)$
(i.e. π_* *is optimal).*

Value Iteration

Bellman equation (Bellman, 1957):

$$q_*(s, a) = R(s, a) + \gamma \max_{a'} q_*(s', a') \qquad \text{here } s' = \delta(s, a) \quad (1)$$

The true optimal values q_* form the unique solution of the above equation.

Value Iteration

Bellman equation (Bellman, 1957):

$$q_*(s,a) = R(s,a) + \gamma \max_{a'} q_*(s',a') \qquad \text{here } s' = \delta(s,a) \quad (1)$$

The true optimal values q_* form the unique solution of the above equation.

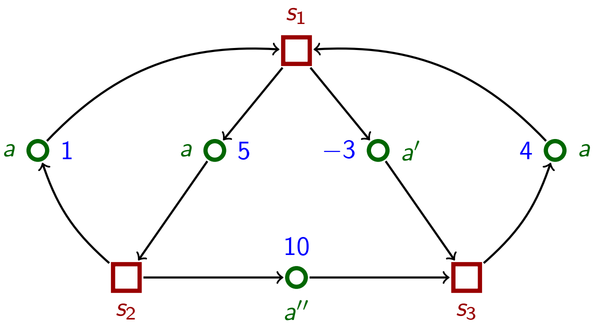
Value iteration algorithm:

- Start with $q_0(s,a) = 0$ for all s, a.
- Iteratively apply the right-hand-side of (1):

$$q_{k+1}(s,a) = R(s,a) + \gamma \max_{a'} q_k(s',a') \qquad \text{here } s' = \delta(s,a)$$

Then $q_*(s,a) = \lim_{k\to\infty} q_k(s,a)$.

Deterministic Markov Decision Processes



	q_0	q_1	q_2	$\cdots$
(s_1, a)	0	5	$5 + \gamma 10$	$\cdots$
(s_1, a')	0	-3	$\cdots$	
(s_2, a)	0	1	$\cdots$	
(s_2, a'')	0	10	$\cdots$	
(s_3, a)	0	4	$\cdots$	

$$q_2(s_1, a) = q_1(s_1, a) + \gamma \max\{q_1(s_2, a), q_1(s_2, a'')\} = 5 + \gamma \max\{1, 10\}$$

Criticism

Minor issue: The value iteration can be used only if the transition relation δ is known.

Major issue: State/action space is typically huge or infinite:

- Atari games: $128^{84\times 84\times 4} = 128^{28224}$ possible states!
- Go: 10^{170} states
- Helicopter control: Infinite!

Criticism

Minor issue: The value iteration can be used only if the transition relation δ is known.

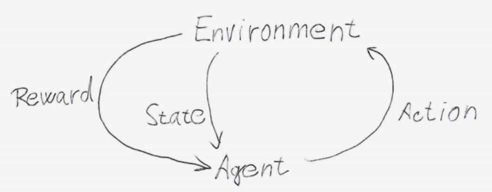
Major issue: State/action space is typically huge or infinite:

- Atari games: $128^{84 \times 84 \times 4} = 128^{28224}$ possible states!
- Go: 10^{170} states
- Helicopter control: Infinite!

We solve this problem in two steps:

1. Update our approximation of q_* only for "relevant" state-action pairs.
   (using reinforcement learning)
2. Represent our approximation of q_* succinctly.
   (using neural networks).

Reinforcement learning (roughly)



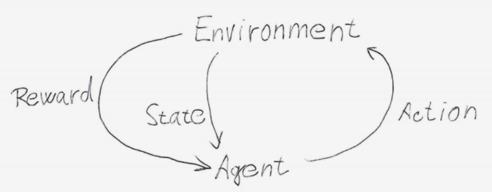
The problem:

How to learn q_* ?

Reinforcement learning (roughly)

The problem:

How to learn q_* ?



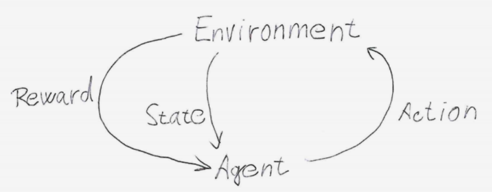
In general:

- Start with a policy $\hat{\pi}$ and an estimate Q of q_*.
- **While** (unhappy with the result) **do**
  - Simulate $\hat{\pi}$ and update the estimate Q based on "experience".
  - Update the policy $\hat{\pi}$ according to Q.

Reinforcement learning (roughly)

The problem:

How to learn q_* ?



In general:

- Start with a policy $\hat{\pi}$ and an estimate Q of q_*.
- **While** (unhappy with the result) **do**
  - Simulate $\hat{\pi}$ and update the estimate Q based on "experience".
  - Update the policy $\hat{\pi}$ according to Q.

We need to

- have a good rule for learning from experience (*exploit* your choice of actions),
- go through important parts of the state-space (*explore* the state space).

Q-learning

For exploration, consider ε-greedy (randomized) policy $\hat{\pi}$:

- With probability $1 - \varepsilon$, choose $a = \arg\max_{a'} Q(s, a')$.
- With probability ε, choose an arbitrary action uniformly in random.

Q-learning

For exploration, consider ε-greedy (randomized) policy $\hat{\pi}$:

- With probability $1-\varepsilon$, choose $a = \arg\max_{a'} Q(s, a')$.
- With probability ε, choose an arbitrary action uniformly in random.

Q-learning algorithm:

- Always follow $\hat{\pi}$.
- In every time instant t update Q by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t(q_*(S_t, A_t) - Q(S_t, A_t))$$

Q-learning

For exploration, consider ε-greedy (randomized) policy $\hat{\pi}$:

- With probability $1 - \varepsilon$, choose $a = \arg\max_{a'} Q(s, a')$.
- With probability ε, choose an arbitrary action uniformly in random.

Q-learning algorithm:

- Always follow $\hat{\pi}$.
- In every time instant t update Q by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t(q_*(S_t, A_t) - Q(S_t, A_t))$$

But we do not know $q_*(S_t, A_t)$ ...

Q-learning

For exploration, consider ε-greedy (randomized) policy $\hat{\pi}$:

- With probability $1-\varepsilon$, choose $a = \arg\max_{a'} Q(s, a')$.
- With probability ε, choose an arbitrary action uniformly in random.

Q-learning algorithm:

- Always follow $\hat{\pi}$.
- In every time instant t update Q by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t(q_*(S_t, A_t) - Q(S_t, A_t))$$

But we do not know $q_*(S_t, A_t)$ ... employ a bootstrap estimate:

$$q_*(S_t, A_t) \quad \approx \quad R_t + \gamma \max_a Q(S_{t+1}, a)$$

Q-learning

For exploration, consider ε-greedy (randomized) policy $\hat{\pi}$:

- With probability $1 - \varepsilon$, choose $a = \arg\max_{a'} Q(s, a')$.
- With probability ε, choose an arbitrary action uniformly in random.

Q-learning algorithm:

- Always follow $\hat{\pi}$.
- In every time instant t update Q by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t(q_*(S_t, A_t) - Q(S_t, A_t))$$

But we do not know $q_*(S_t, A_t)$ ... employ a bootstrap estimate:

$$q_*(S_t, A_t) \quad \approx \quad R_t + \gamma \max_a Q(S_{t+1}, a)$$

and obtain

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left(R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

Q-learning

For exploration, consider ε-greedy (randomized) policy $\hat{\pi}$:

- With probability $1-\varepsilon$, choose $a = \arg\max_{a'} Q(s, a')$.
- With probability ε, choose an arbitrary action uniformly in random.

Q-learning algorithm:

- Always follow $\hat{\pi}$.
- In every time instant t update Q by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t(q_*(S_t, A_t) - Q(S_t, A_t))$$

But we do not know $q_*(S_t, A_t)$ ... employ a bootstrap estimate:

$$q_*(S_t, A_t) \quad \approx \quad R_t + \gamma \max_a Q(S_{t+1}, a)$$

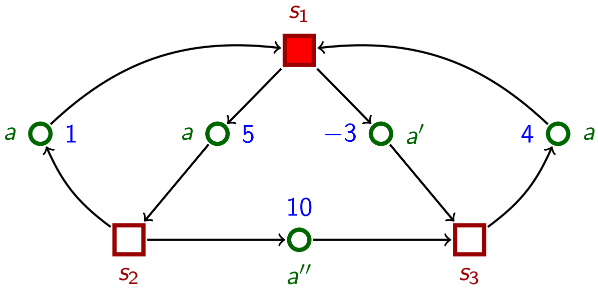
and obtain

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t \left(R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

Theorem (Watkins & Dayan 1992)

If S is finite and $\alpha_t = 1/t$, then each $Q(s, a)$ converges to $q_*(s, a)$.

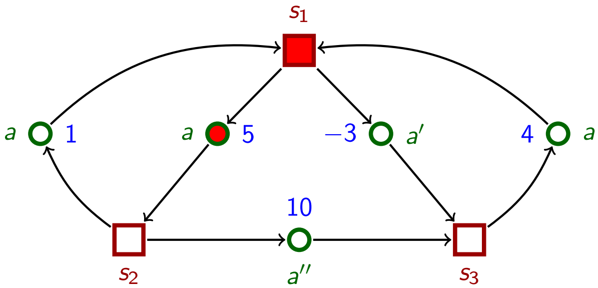
Deterministic Markov Decision Processes



t	0	1	2	$\cdots$
(s_1, a)	0			
(s_1, a')	0			
(s_2, a)	0			
(s_2, a'')	0			
(s_3, a)	0			

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

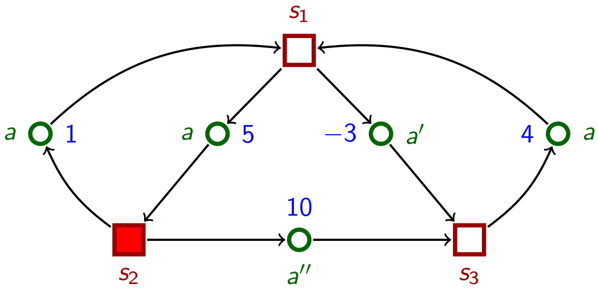
Deterministic Markov Decision Processes



t	0	1	2	$\cdots$
(s_1, a)	0	$0 + \alpha(5 + \gamma 0 - 0)$		
(s_1, a')	0	0		
(s_2, a)	0	0		
(s_2, a'')	0	0		
(s_3, a)	0	0		

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

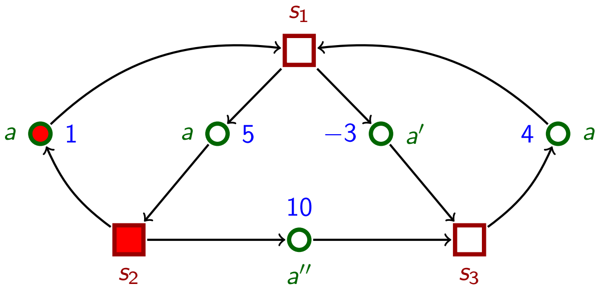
Deterministic Markov Decision Processes



t	0	1	2	$\cdots$
(s_1, a)	0	$0 + \alpha(5 + \gamma 0 - 0)$		
(s_1, a')	0	0		
(s_2, a)	0	0		
(s_2, a'')	0	0		
(s_3, a)	0	0		

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

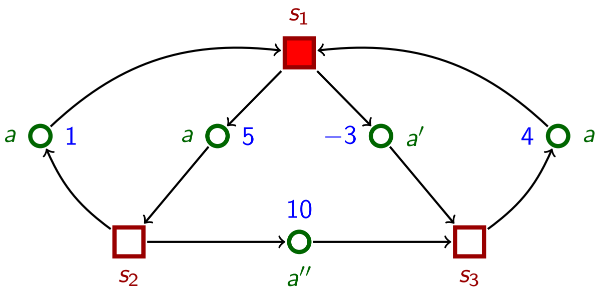
Deterministic Markov Decision Processes



t	0	1	2	$\cdots$
(s_1, a)	0	$0 + \alpha(5 + \gamma 0 - 0)$	$\alpha 5$	
(s_1, a')	0	0	0	
(s_2, a)	0	0	$0 + \alpha(1 + \gamma \alpha 5 - 0)$	
(s_2, a'')	0	0	0	
(s_3, a)	0	0	0	

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

Deterministic Markov Decision Processes



t	0	1	2	$\cdots$
(s_1, a)	0	$0 + \alpha(5 + \gamma 0 - 0)$	$\alpha 5$	
(s_1, a')	0	0	0	
(s_2, a)	0	0	$0 + \alpha(1 + \gamma\alpha 5 - 0)$	
(s_2, a'')	0	0	0	
(s_3, a)	0	0	0	

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

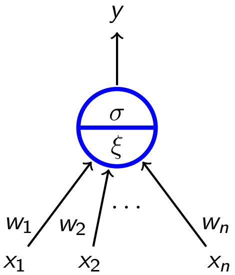
Q-learning with Function Approximation

The problem: How to represent Q ?

- linear combinations of (manually created) features [typical]
- decision trees
- SVM
- neural networks
- ...

Neural networks

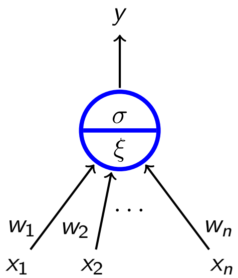
Neural network is a directed graph of interconnected neurons.

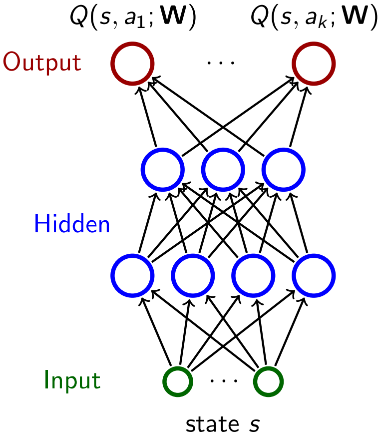


- $w_1, \dots, w_n \in \mathbb{R}$ are **weights**
- $x_1, \dots, x_n \in \mathbb{R}$ are **inputs**
- $\xi = \sum_{i=1}^{n} w_i x_i$
- $y = \sigma(\xi)$

Neural networks

Neural network is a directed graph of interconnected neurons.





- $w_1, \dots, w_n \in \mathbb{R}$ are **weights**
- $x_1, \dots, x_n \in \mathbb{R}$ are **inputs**
- $\xi = \sum_{i=1}^{n} w_i x_i$
- $y = \sigma(\xi)$

- $\mathbf{W}$ are weights of all neurons
- $Q(s, a; \mathbf{W})$ the Q value in (s, a) represented by the network

Q-learning with Neural Networks

Q-learning algorithm:

- Always follow ε-greedy $\hat{\pi}$.
- In every time instant t consider (S_t, A_t, R_t, S_{t+1}):

Q-learning with Neural Networks

Q-learning algorithm:

- Always follow ε-greedy $\hat{\pi}$.
- In every time instant t consider (S_t, A_t, R_t, S_{t+1}):
  - Freeze the current weights as $\mathbf{W}^-$, fix the "target" value $\tau := R_t + \gamma \max_a Q(S_{t+1}, a; \mathbf{W}^-)$.

Q-learning with Neural Networks

Q-learning algorithm:

- Always follow ε-greedy $\hat{\pi}$.
- In every time instant t consider (S_t, A_t, R_t, S_{t+1}):
  - Freeze the current weights as $\mathbf{W}^-$, fix the "target" value $\tau := R_t + \gamma \max_a Q(S_{t+1}, a; \mathbf{W}^-)$.
  - Update weights $\mathbf{W}$ so that $Q(S_t, A_t; \mathbf{W})$ gets closer to τ

$$\mathbf{W} \leftarrow \mathbf{W} + \alpha_t(\tau - Q(S_t, A_t; \mathbf{W}))\nabla_{\mathbf{W}} Q(S_t, A_t; \mathbf{W})$$

Q-learning with Neural Networks

Q-learning algorithm:

- Always follow ε-greedy $\hat{\pi}$.
- In every time instant t consider (S_t, A_t, R_t, S_{t+1}):
  - Freeze the current weights as $\mathbf{W}^-$, fix the "target" value $\tau := R_t + \gamma \max_a Q(S_{t+1}, a; \mathbf{W}^-)$.
  - Update weights $\mathbf{W}$ so that $Q(S_t, A_t; \mathbf{W})$ gets closer to τ

$$\mathbf{W} \leftarrow \mathbf{W} + \alpha_t(\tau - Q(S_t, A_t; \mathbf{W}))\nabla_{\mathbf{W}} Q(S_t, A_t; \mathbf{W})$$

How the above rule is derived?

Q-learning with Neural Networks

Q-learning algorithm:

- Always follow ε-greedy $\hat{\pi}$.
- In every time instant t consider (S_t, A_t, R_t, S_{t+1}):
  - Freeze the current weights as $\mathbf{W}^-$, fix the "target" value $\tau := R_t + \gamma \max_a Q(S_{t+1}, a; \mathbf{W}^-)$.
  - Update weights $\mathbf{W}$ so that $Q(S_t, A_t; \mathbf{W})$ gets closer to τ

$$\mathbf{W} \leftarrow \mathbf{W} + \alpha_t(\tau - Q(S_t, A_t; \mathbf{W}))\nabla_{\mathbf{W}} Q(S_t, A_t; \mathbf{W})$$

How the above rule is derived?

We want to adjust $\mathbf{W}$ to minimize the square error (here τ is constant!)

$$L(\mathbf{W}) = \frac{1}{2}(\tau - Q(S_t, A_t; \mathbf{W}))^2$$

Q-learning with Neural Networks

Q-learning algorithm:

- Always follow ε-greedy $\hat{\pi}$.
- In every time instant t consider (S_t, A_t, R_t, S_{t+1}):
  - Freeze the current weights as $\mathbf{W}^-$, fix the "target" value $\tau := R_t + \gamma \max_a Q(S_{t+1}, a; \mathbf{W}^-)$.
  - Update weights $\mathbf{W}$ so that $Q(S_t, A_t; \mathbf{W})$ gets closer to τ

$$\mathbf{W} \leftarrow \mathbf{W} + \alpha_t(\tau - Q(S_t, A_t; \mathbf{W}))\nabla_{\mathbf{W}} Q(S_t, A_t; \mathbf{W})$$

How the above rule is derived?

We want to adjust $\mathbf{W}$ to minimize the square error (here τ is constant!)

$$L(\mathbf{W}) = \frac{1}{2}(\tau - Q(S_t, A_t; \mathbf{W}))^2$$

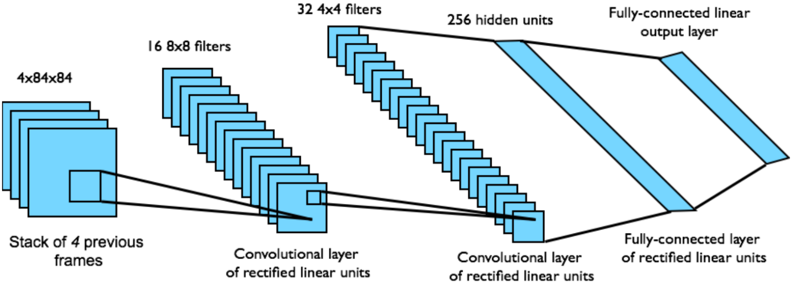
using gradient descent $\mathbf{W} = \mathbf{W} - \alpha \nabla_{\mathbf{W}} L(\mathbf{W})$ where

$$\begin{aligned}\nabla_{\mathbf{W}} L(\mathbf{W}) &= \nabla_{\mathbf{W}} \left((1/2)(\tau - Q(S_t, A_t; \mathbf{W}))^2\right) \\ &= (\tau - Q(S_t, A_t; \mathbf{W}))(-\nabla_{\mathbf{W}} Q(S_t, A_t; \mathbf{W}))\end{aligned}$$

$\nabla_{\mathbf{W}} Q(S_t, A_t; \mathbf{W})$ can be computed using standard backpropagation.

Convolutional Networks

In image processing, classical MLP has been superseded by *convolutional networks*.



First introduced in [LeCun et al., 1989d] for handwritten digits recognition.

Combined with powerful GPU powered computers ⇒ breakthrough in image processing.

Image: D. Silver, UCL Course on RL, `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html`

DQN

Note that the Q-learning algorithm adapts weights in every step.

May be unstable: The learning may be slow or even diverge since

- training samples obtained along simulations are strongly correlated,
- their distribution changes.

DQN

Note that the Q-learning algorithm adapts weights in every step.

May be unstable: The learning may be slow or even diverge since

- training samples obtained along simulations are strongly correlated,
- their distribution changes.

Replay memory:

- Store history of (state, action, reward, newState) tuples into a memory M.
- Train the network on training examples obtained by sampling from M.

DQN

Note that the Q-learning algorithm adapts weights in every step.

May be unstable: The learning may be slow or even diverge since

- training samples obtained along simulations are strongly correlated,
- their distribution changes.

Replay memory:

- Store history of (state, action, reward, newState) tuples into a memory M.
- Train the network on training examples obtained by sampling from M.

Dealyed target values:

- $\mathbf{W}^-$ is several steps old value of weights.
  (Previously $\mathbf{W}^-$ was the current weight vector.)

Both adjustments considerably improve learning (see results later).

Experiments

Training:

- 49 games, the same architecture of network (trained for each game).
- ε-greedy strategy with ε annealed linearly from 1.0 to 0.1 over the first million frames.
- Trained for 50 million frames (around 38 days of game experience in total).

Experiments

Training:

- 49 games, the same architecture of network (trained for each game).
- ε-greedy strategy with ε annealed linearly from 1.0 to 0.1 over the first million frames.
- Trained for 50 million frames (around 38 days of game experience in total).

Evaluation:

- Play each game 30 times for up to 5 min each time with different initial random conditions.
- ε-greedy policy with $\varepsilon = 0.05$.
- A random agent selecting actions at 10Hz used as baseline.
- Pro human tester under the same emulator, average reward for 20 episodes, max 5 minutes, following around 2h of practice playing each game.

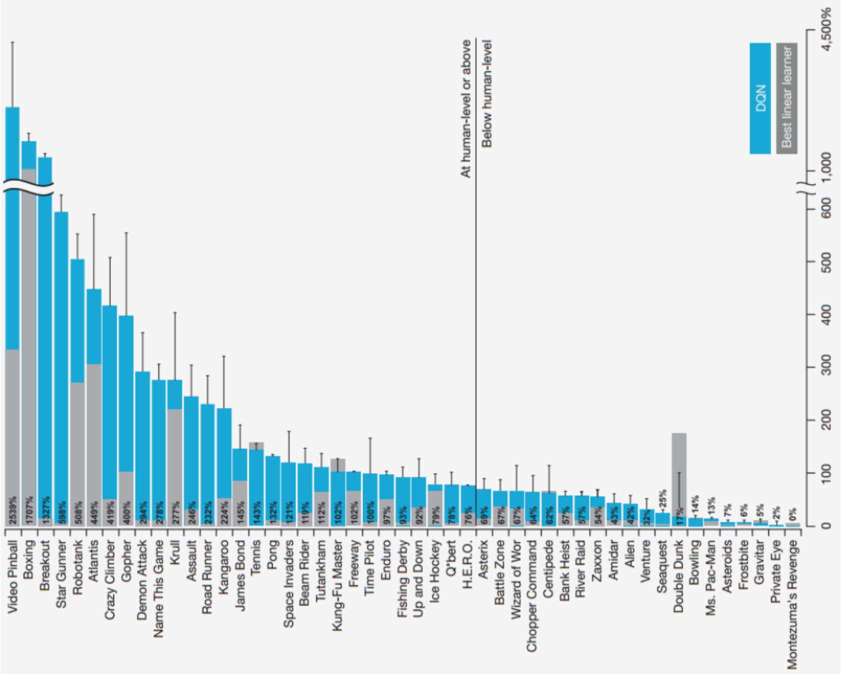


Image: V. Mnih et al, Human-level control through deep reinforcement learning. Nature (2015).

Results

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Sarsa and **Contingency** are other reinforcement learning methods.

HNeat Best and **HNeat Pixel** are methods based on evolutionary policy search.
These methods use a hand-engineered object detector algorithm that outputs the locations and types of objects on the Atari screen.

Image: D. Silver, UCL Course on RL, `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html`

Conceptual Limitations (as opposed to humans)

- Prior knowledge:
  - **Humans**: Huge amount, such as intuitive physics and intuitive psychology.
  - **RL**: Starts from scratch which is simultaneously impressive (because it works) and depressing (because we lack concrete ideas for how not to).

Conceptual Limitations (as opposed to humans)

- Prior knowledge:
  - **Humans**: Huge amount, such as intuitive physics and intuitive psychology.
  - **RL**: Starts from scratch which is simultaneously impressive (because it works) and depressing (because we lack concrete ideas for how not to).
- Abstraction and planning:
  - **Humans**: Build a rich, abstract model and plan within it.
  - **RL**: Brute force, where the correct actions are eventually discovered and internalized into a policy.

Conceptual Limitations (as opposed to humans)

- Prior knowledge:
  - **Humans**: Huge amount, such as intuitive physics and intuitive psychology.
  - **RL**: Starts from scratch which is simultaneously impressive (because it works) and depressing (because we lack concrete ideas for how not to).
- Abstraction and planning:
  - **Humans**: Build a rich, abstract model and plan within it.
  - **RL**: Brute force, where the correct actions are eventually discovered and internalized into a policy.
- Experience acquisition:
  - **Humans**: Can figure out what is likely to give rewards without ever actually experiencing the rewarding transition.
  - **RL**: Has to actually experience a positive reward.

A. Karpathy, Deep Reinforcement Learning: Pong from Pixels,

http://karpathy.github.io/2016/05/31/rl/.

Conclusions

- Current computers can learn to play games on old computer at (super)human level.
- The main algorithms used in solution:
  - Reinforcement learning
  - Convolutional networks
- It is a very active area of research, several better solutions than DQN have been recently presented.

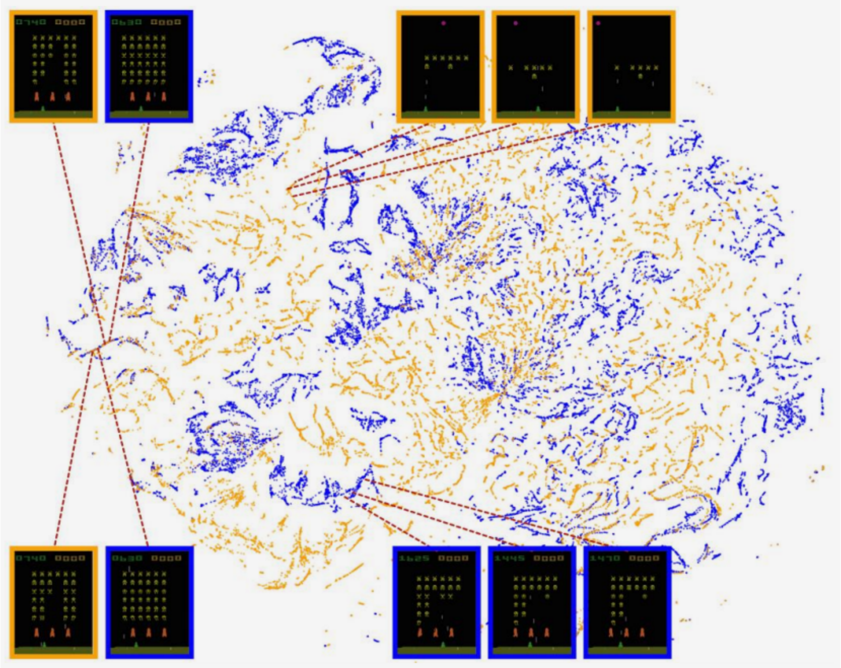


Image: V. Mnih et al, Human-level control through deep reinforcement learning. Nature (2015).