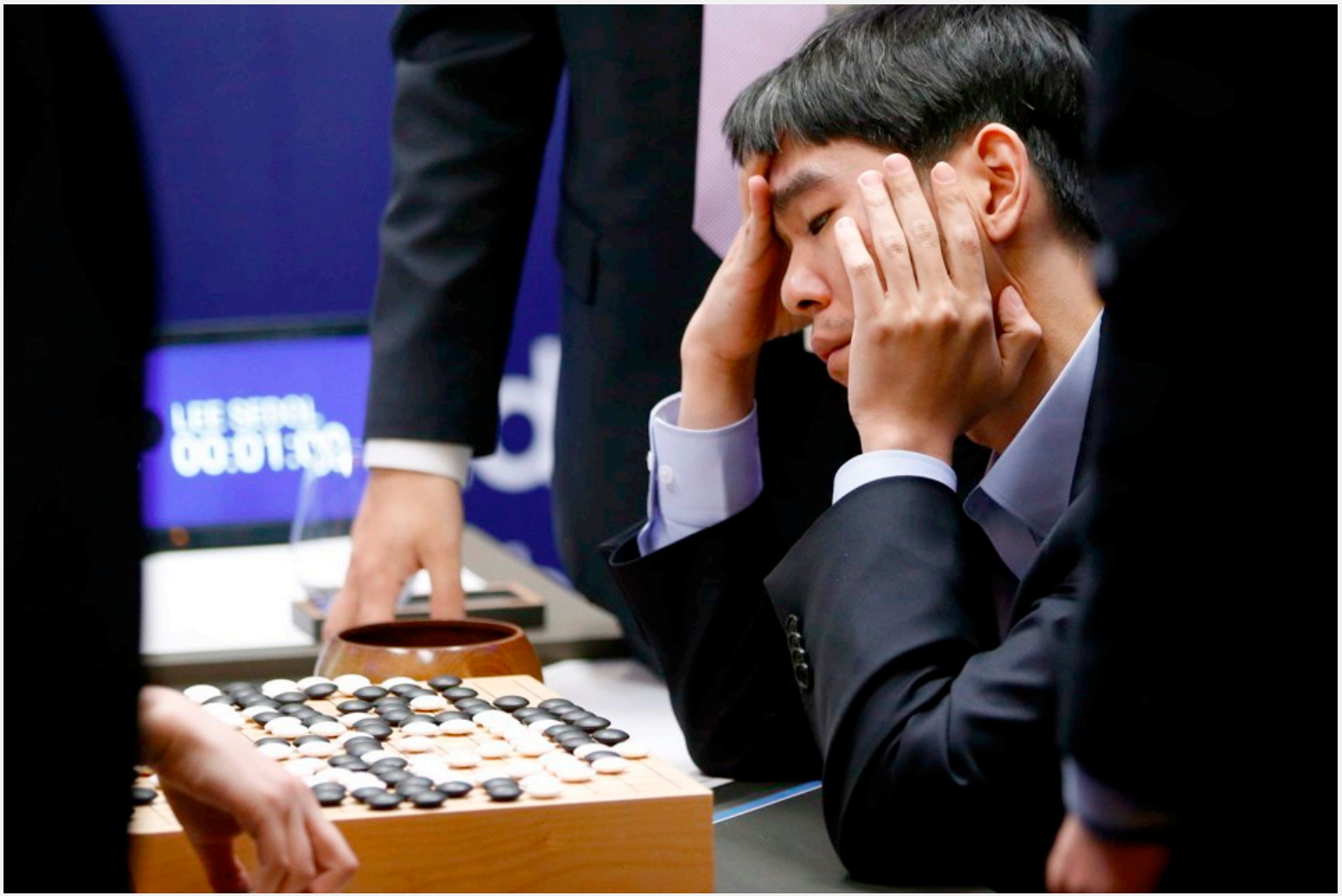
Reinforcement Learning

Ekapol Chuangsuwanich
Department of Computer Engineering
Chulalongkorn University

Deep blue vs Garry Kasparov 1997

AlphaGo vs Lee Sedo 2016



AlphaGo vs Ke Jie 2017

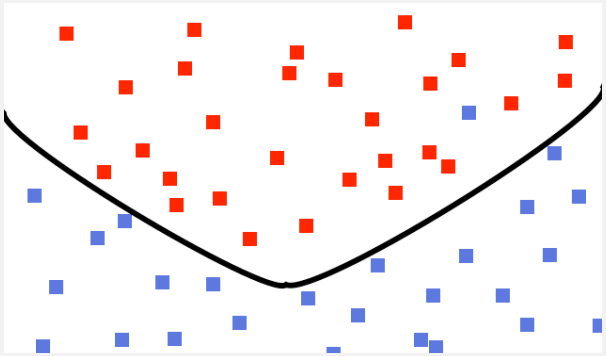
Deep Blue	AlphaGo
1997	2016
IBM	DeepMind
Super computer	Cloud
Chess	Go
10^{40}	10^{170}
Search algorithm	Machine learning algorithm
Hard-coded with specialized rules	Reinforcement learning
Can only play chess	Generalizable

Google buys UK artificial intelligence startup Deepmind for £400m

Google makes its biggest EU purchase yet with the technology that aims to make computers think like humans

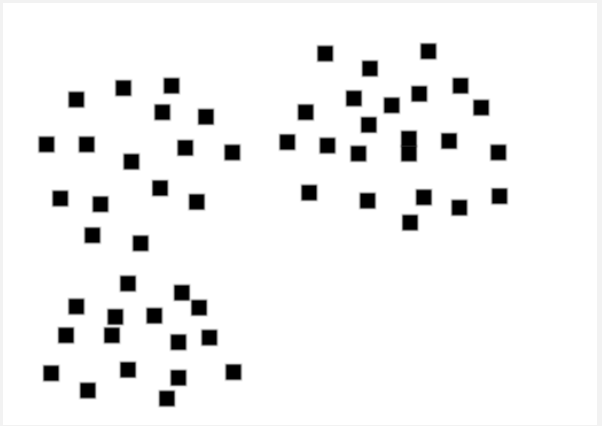
https://www.theguardian.com/technology/2014/jan/27/google-acquires-uk-artificial-intelligence-startup-deepmind

Reinforcement learning

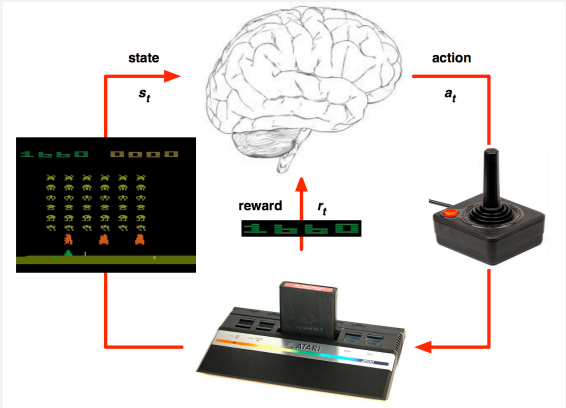


Supervised Learning

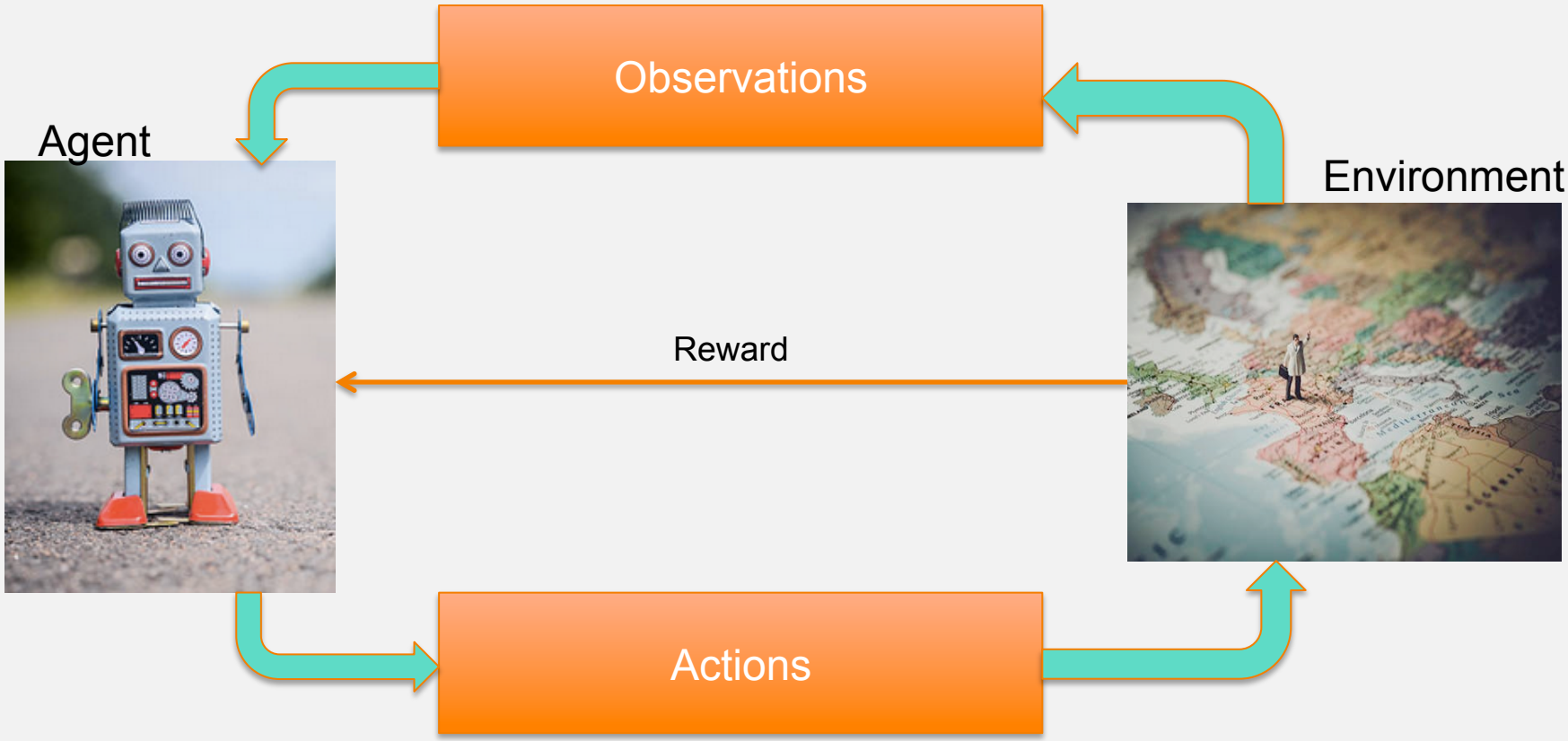
Reinforcement Learning



Unsupervised Learning



Reinforcement learning framework

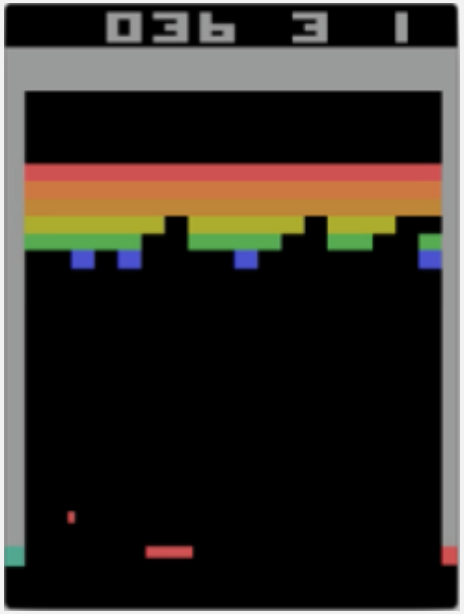


Learning through trial and error

Rewards-based learning

Maximize the rewards

$R_t = \Delta\text{distance}$



$R_t = \Delta\text{score}$

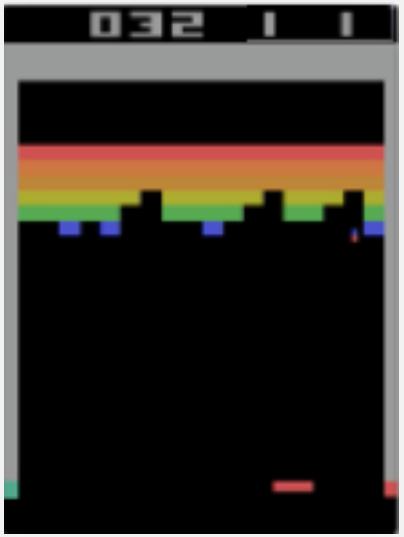
$$R_T = \begin{cases} 1 & \text{, win} \\ -1 & \text{, lose} \end{cases}$$

Credit assignment problem

An action can have consequences further away in time

Some movements might not have any effect on the outcome

10 time steps later



$r_t = 0$

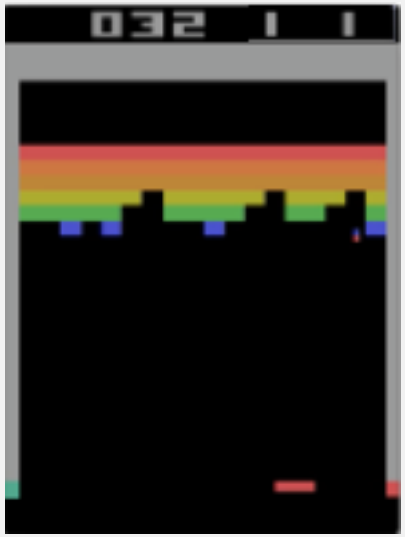
$r_{t+10} = 1$

Cumulative rewards (Return)

Maximize the expected cumulative rewards with discounting

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_t$$

$r_t = 0$



$r_{t+10} = 1$

$r_{t+34} = -1$

Markov Decision Process (MDP)

S,A,P,R,γ

S – Set of states

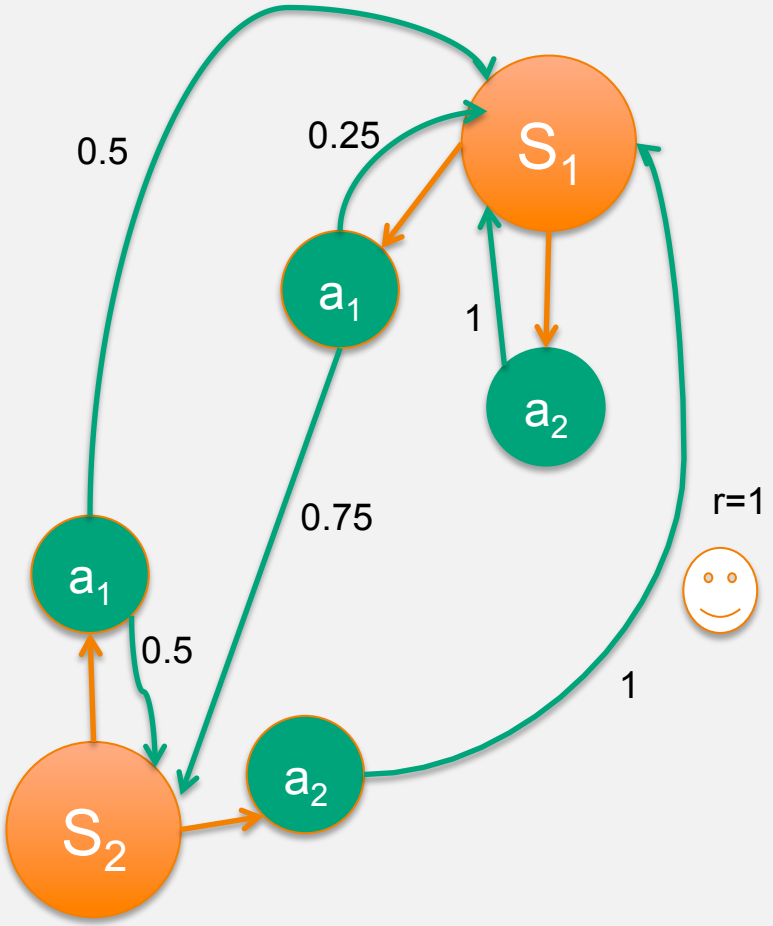
A – Set of actions

P – Transition between states given an action

$$P^{a}_{s,s'} = \text{Prob}[s_{t+1}=s' \mid s_t=s, a_t=a]$$

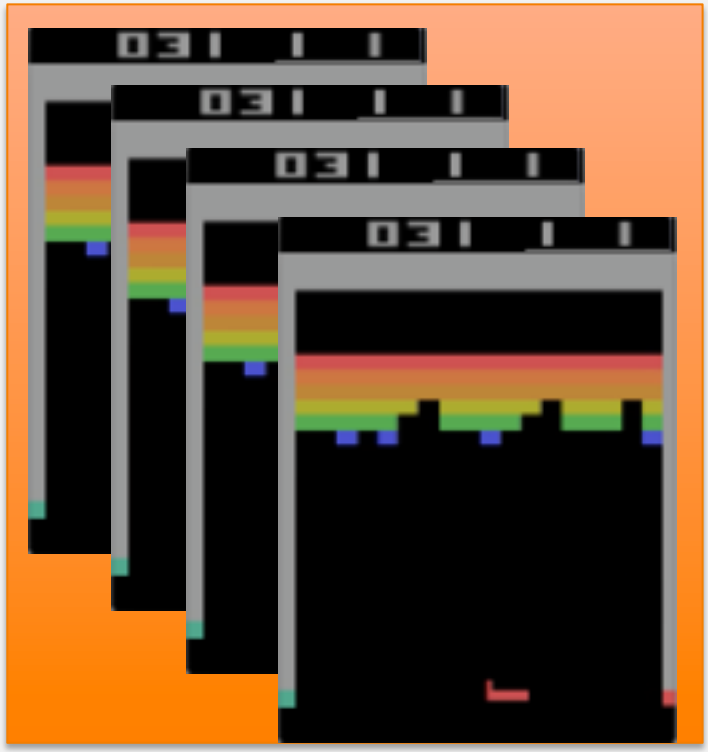
R – Rewards associated with actions and states

γ – Discount factor



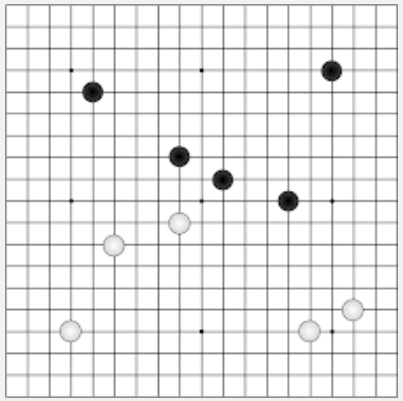
Markov State

Markov property: All information from past is captured in the current state



A state

For Go, a board position
For simple video games, stack multiple frames



Policy

A strategy to perform an action according to a particular state

$$\pi(s) = a$$

A policy defines the behavior of the agent

In MDP, we want to learn the best policy for the task

Action-Value Function (Q Function)

Expected total rewards starting from state s taking action a then follows a given policy π

$$Q_\pi(s,a) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s, a_t = a]$$

$$= E[r_t + \gamma Q_\pi(s',a') \mid s_t = s, a_t = a]$$

$$= R_s^a + \gamma \sum_{s'} P_{s,s'}^a \max_{a' \in A} Q_\pi(s',a')$$

$$Q^*(s,a) = \max_\pi Q_\pi(s,a)$$

Q* determines the optimal policy

Q-learning

Q-learning is used to learn Q* by sampling the states s' and rewards r through events

Starts with an original belief of Q values

Takes random actions to explore the possibilities

Update your belief of Q depending on the rewards from the environment

No need to explicitly learn the transition probability or the expected rewards

"Learns by trial and error"

Temporal difference Q-learning

Update Q by the reward received from taking action a in state s.

State s' is the state after taking action a.

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

Recall

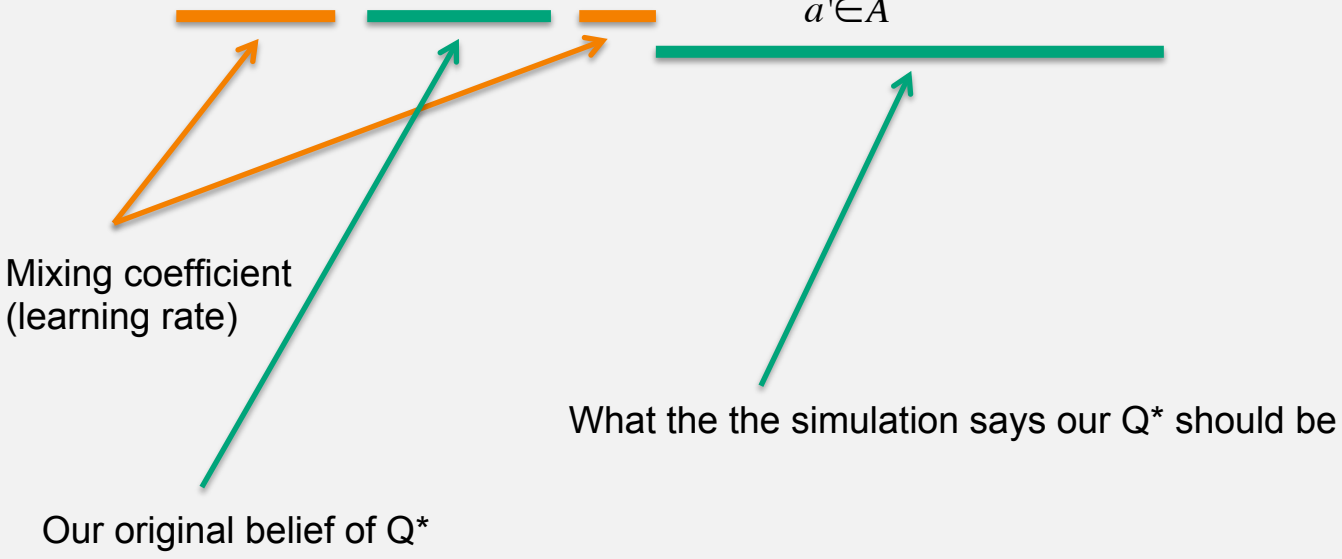
$$Q(s,a) = R_s^a + \gamma \sum_{s'} P_{s,s'}^a \max_{a' \in A} Q(s',a')$$

R (the expected rewards) becomes r (the sampled reward)
The transition probability, P, is dropped because we sampled one outcome
This is a proxy for the sampled Q*

Temporal difference Q-learning

Update Q by the reward received from taking action a in state s.

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$



Temporal difference Q-learning

Update Q by the reward received from taking action a in state s.

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

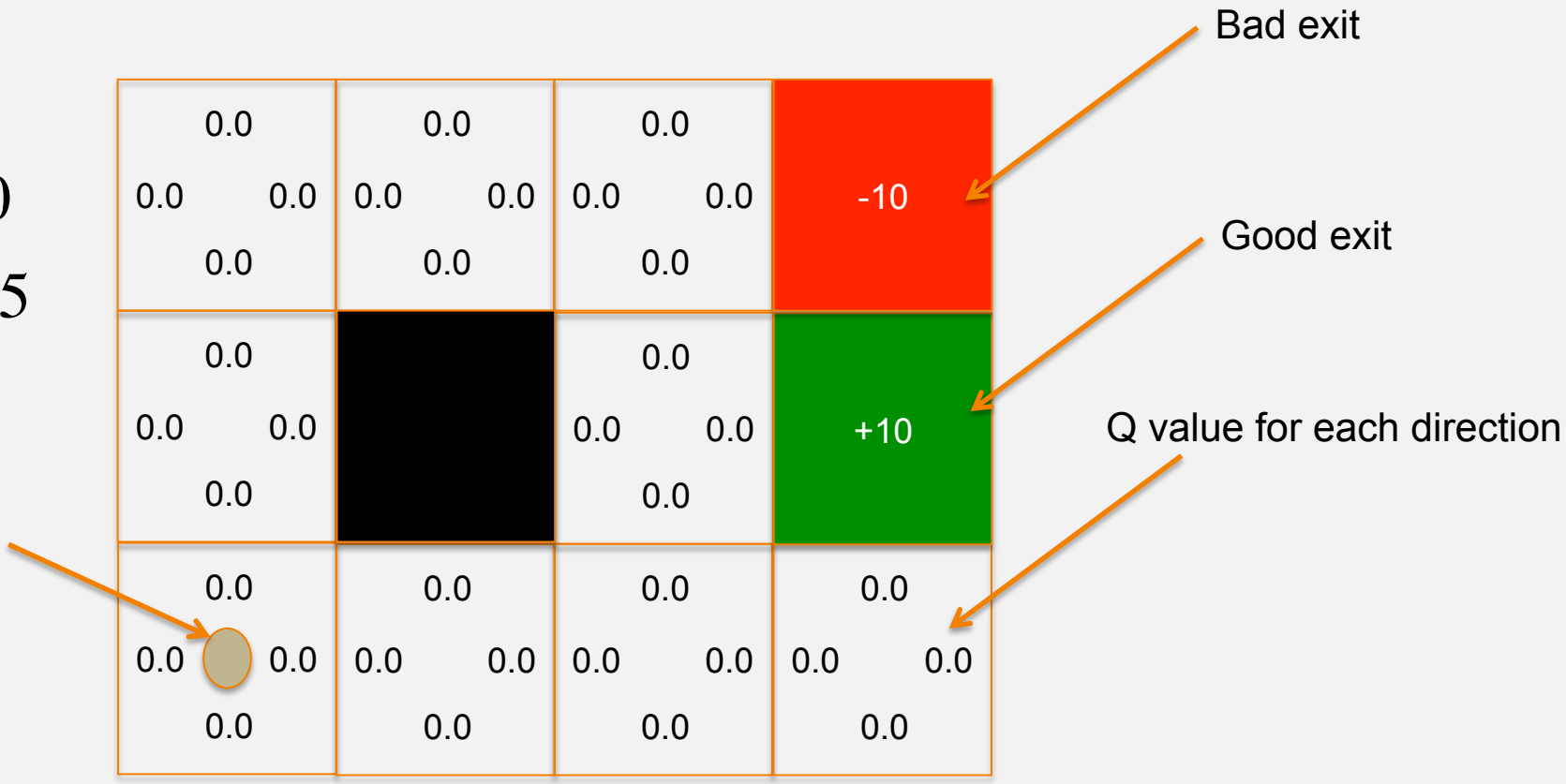
Iterating this equation infinitely is guaranteed to converge Q to the optimal Q* (under certain assumptions)

Toy example

The states are the different positions in the room
4 actions: up, down, left, right

$\gamma = 1.0$

$\alpha = 0.5$



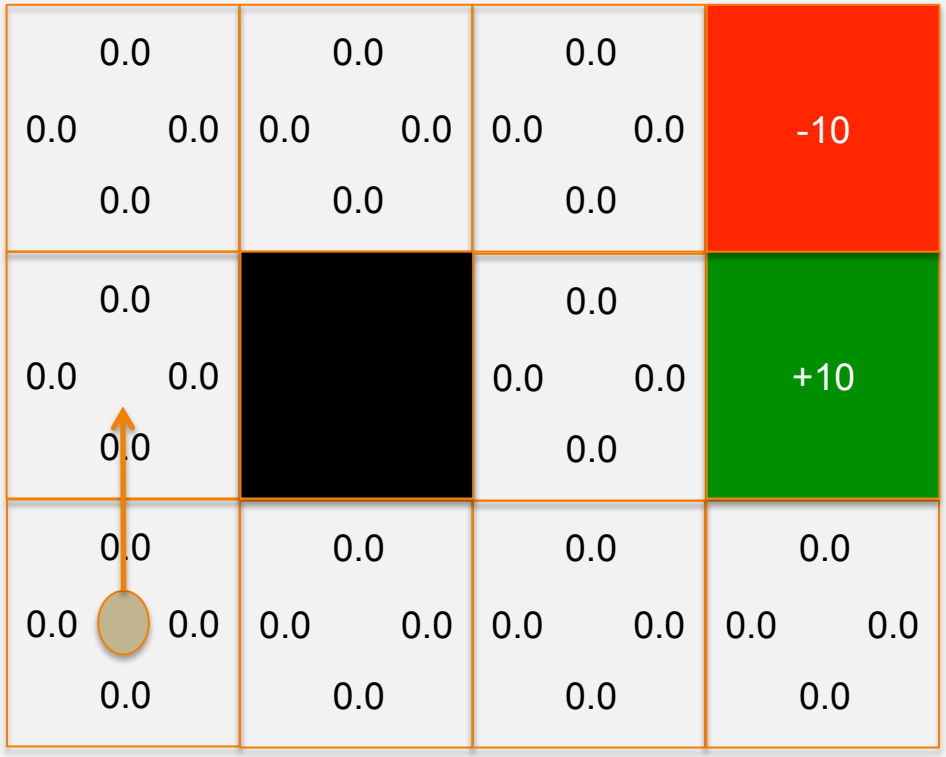
A reward of -1 for every move (penalty for time wasted)

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



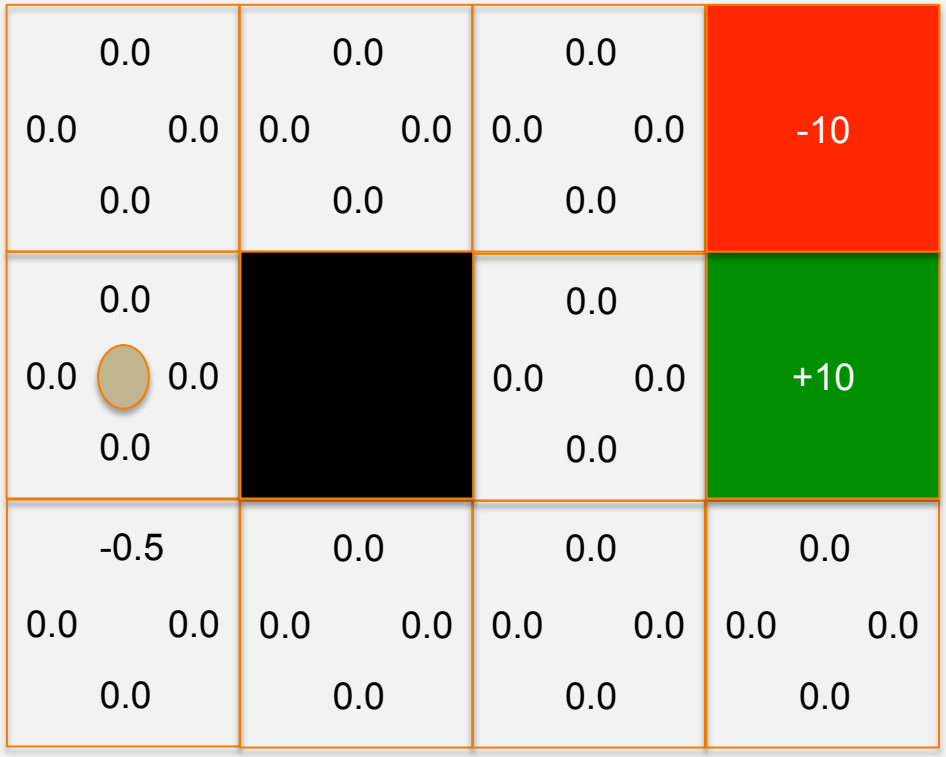
(1-0.5)0+0.5[-1 + 1* 0] = -0.5

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$

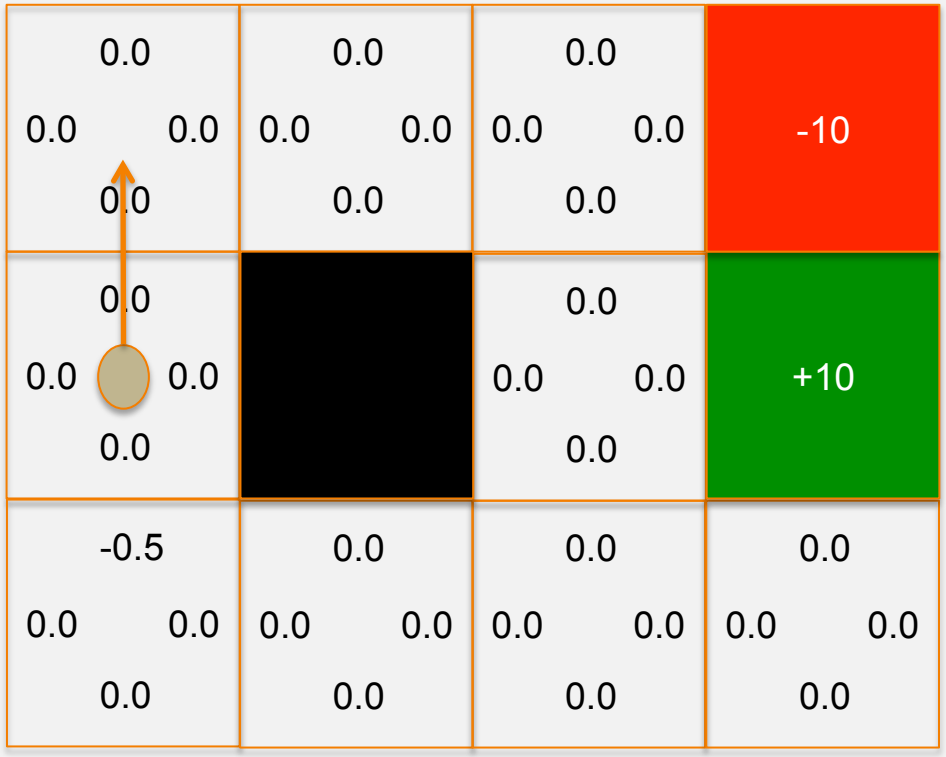


Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



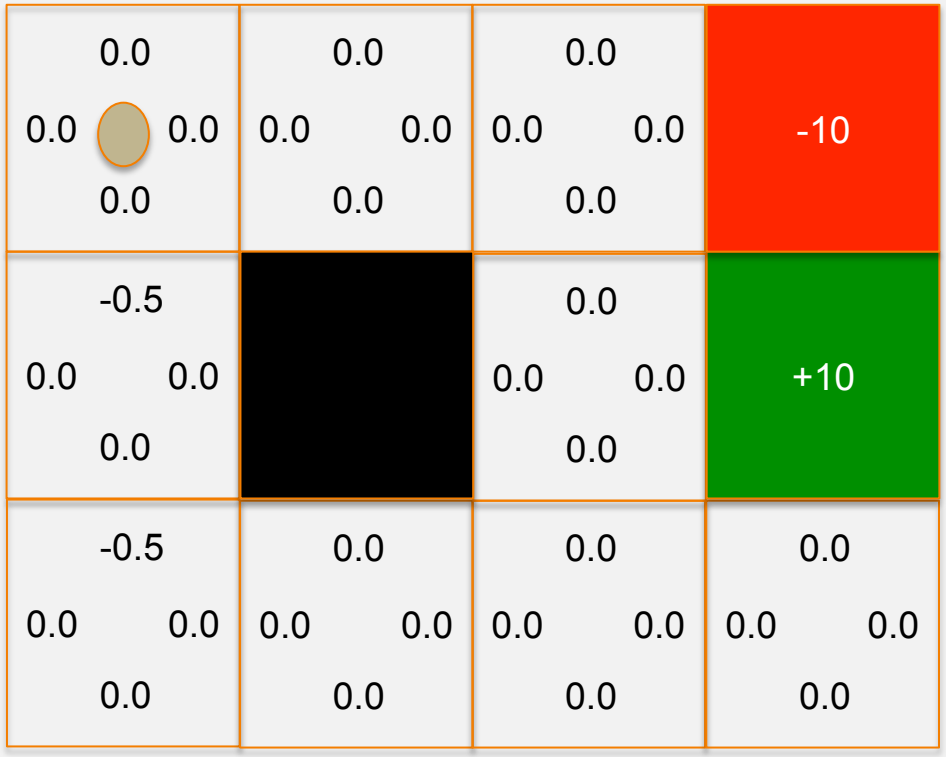
(1-0.5)0+0.5[-1 + 1* 0] = -0.5

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$

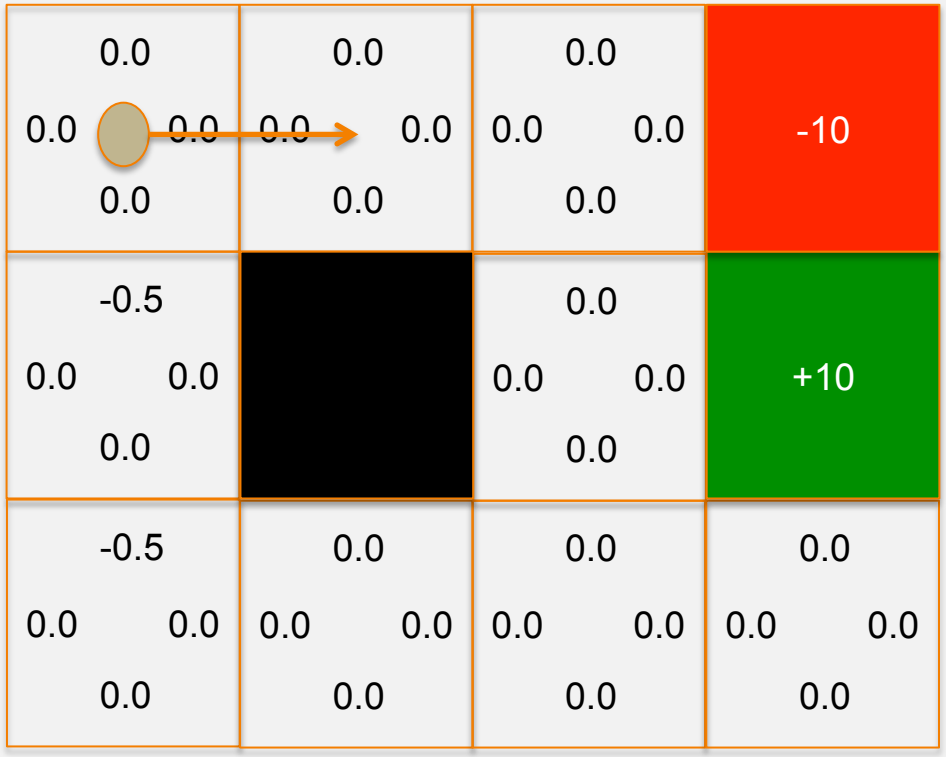


Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



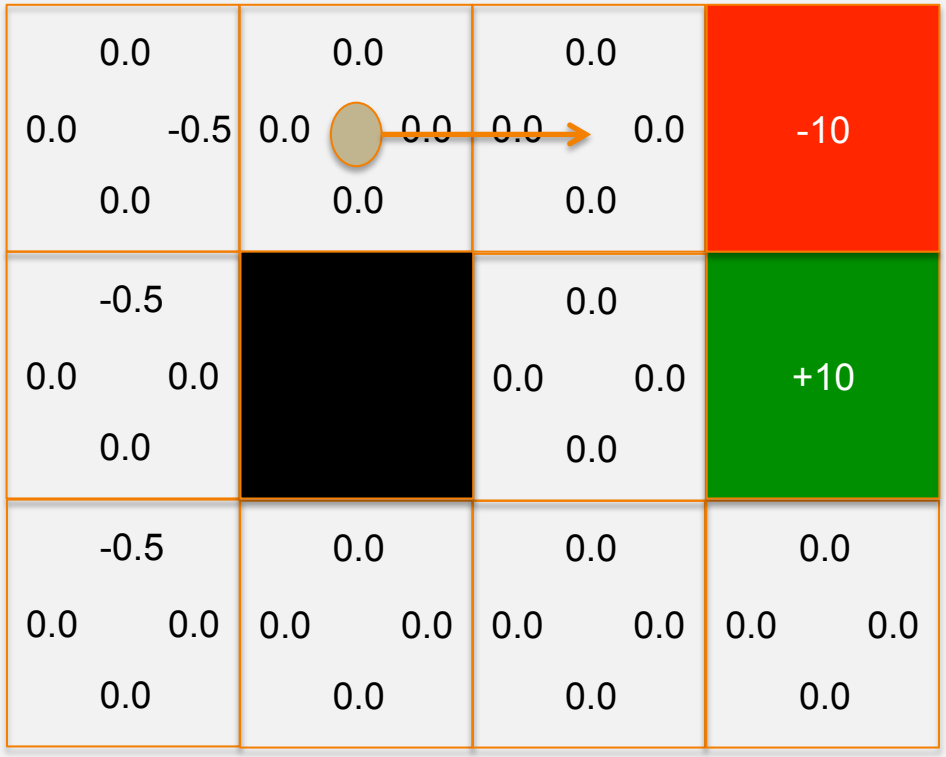
(1-0.5)0+0.5[-1 + 1* 0] = -0.5

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



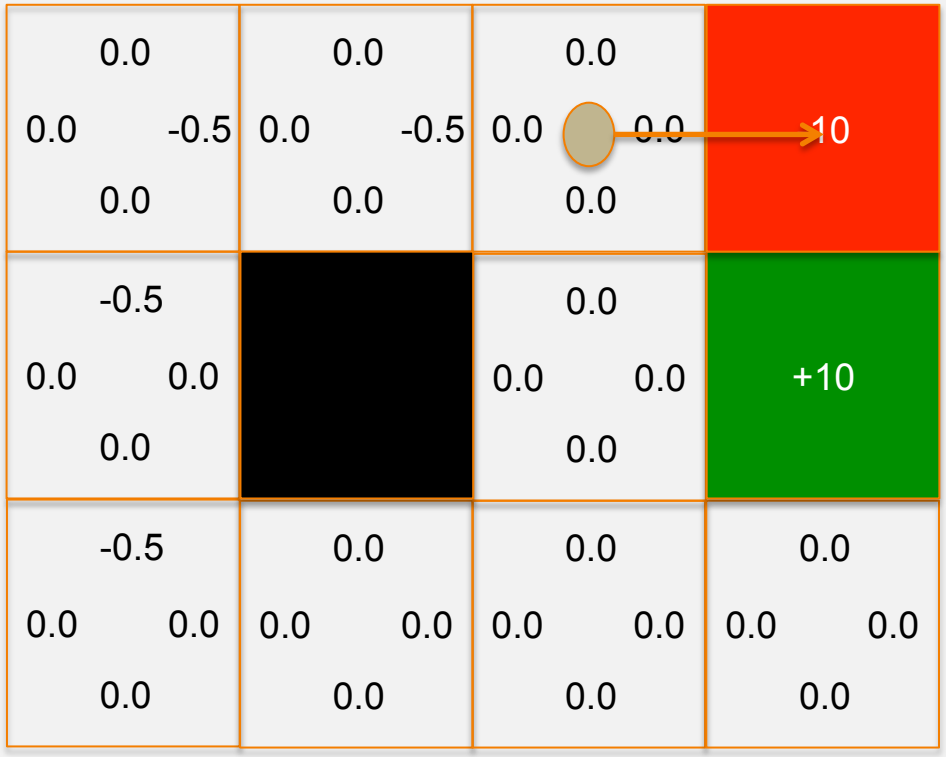
(1-0.5)0+0.5[-1 + 1* 0] = -0.5

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



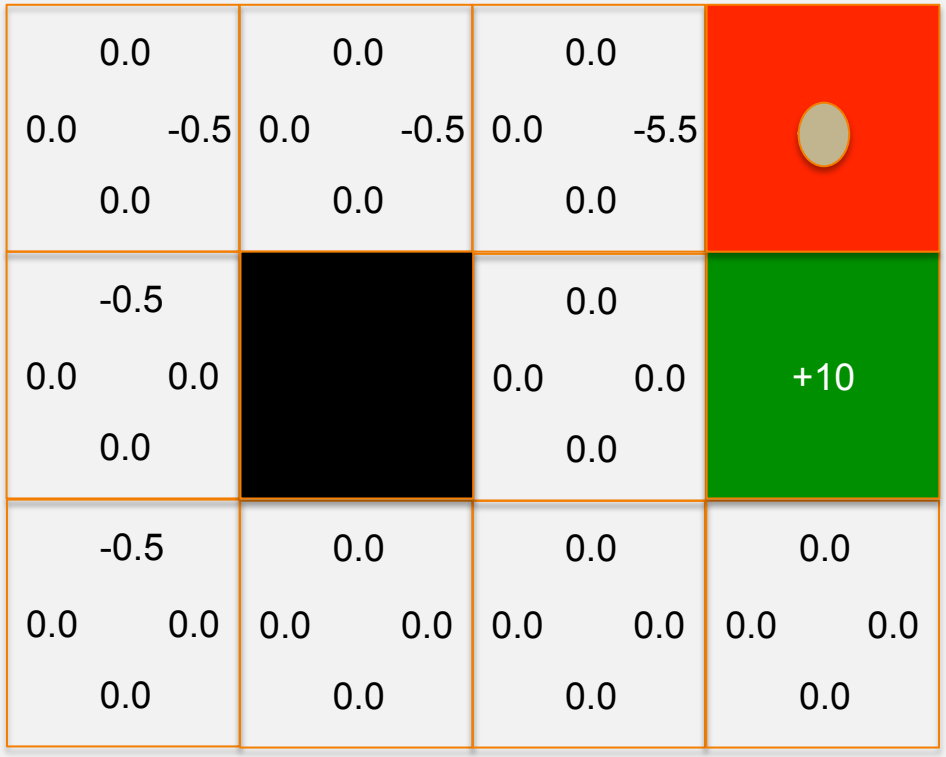
(1-0.5)0+0.5[-1 + 1* -10] = -5.5

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$

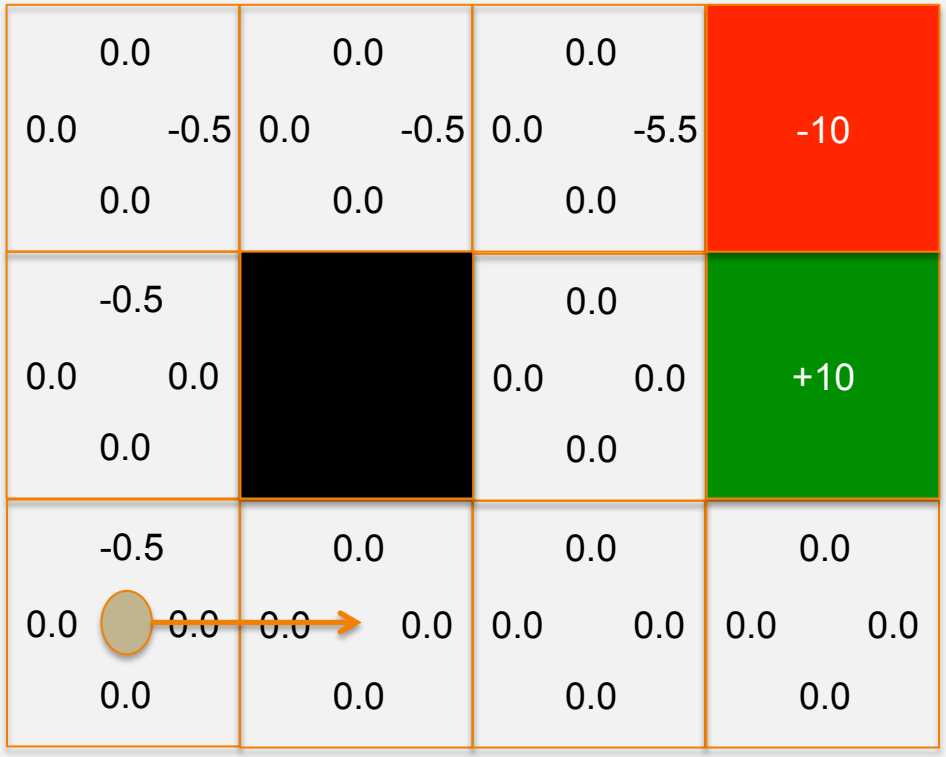


Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$

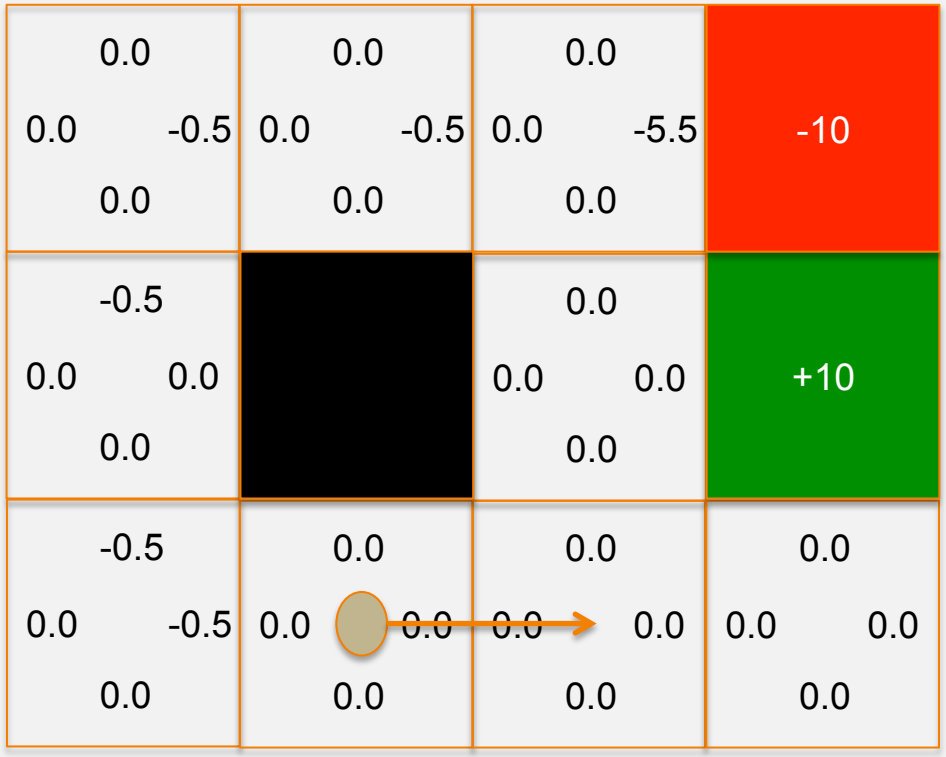


Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$

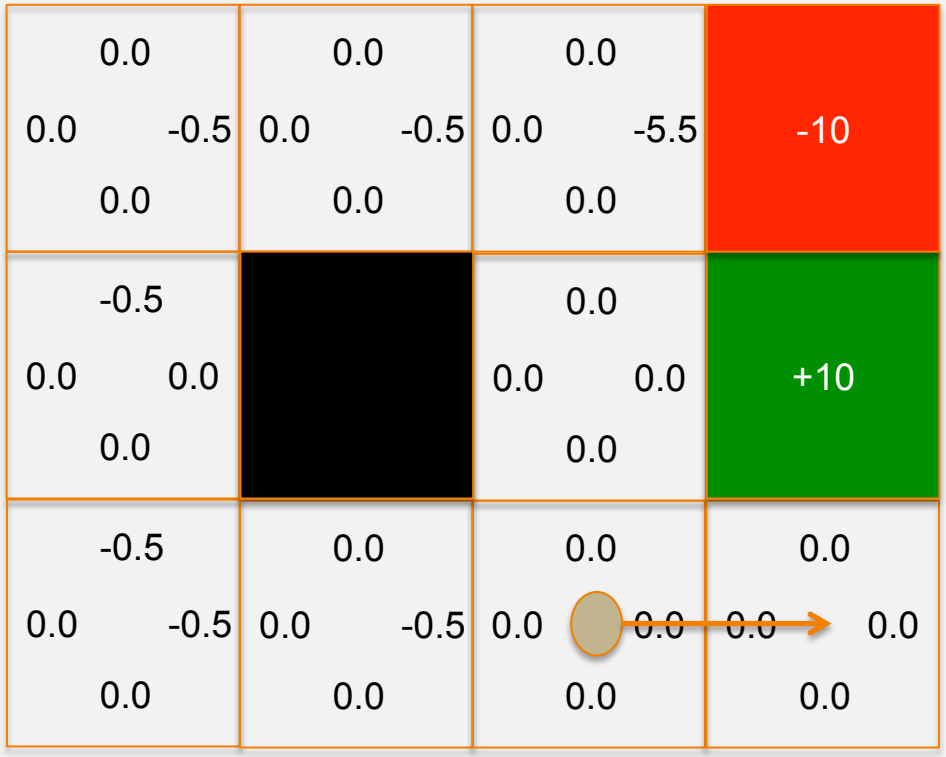


Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$

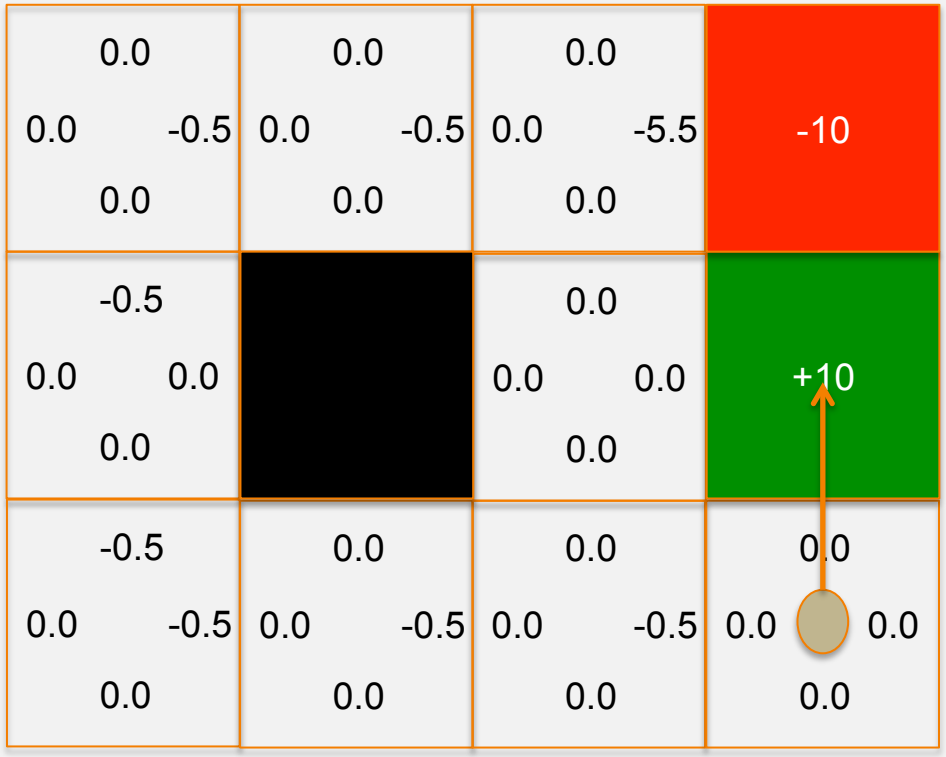


Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



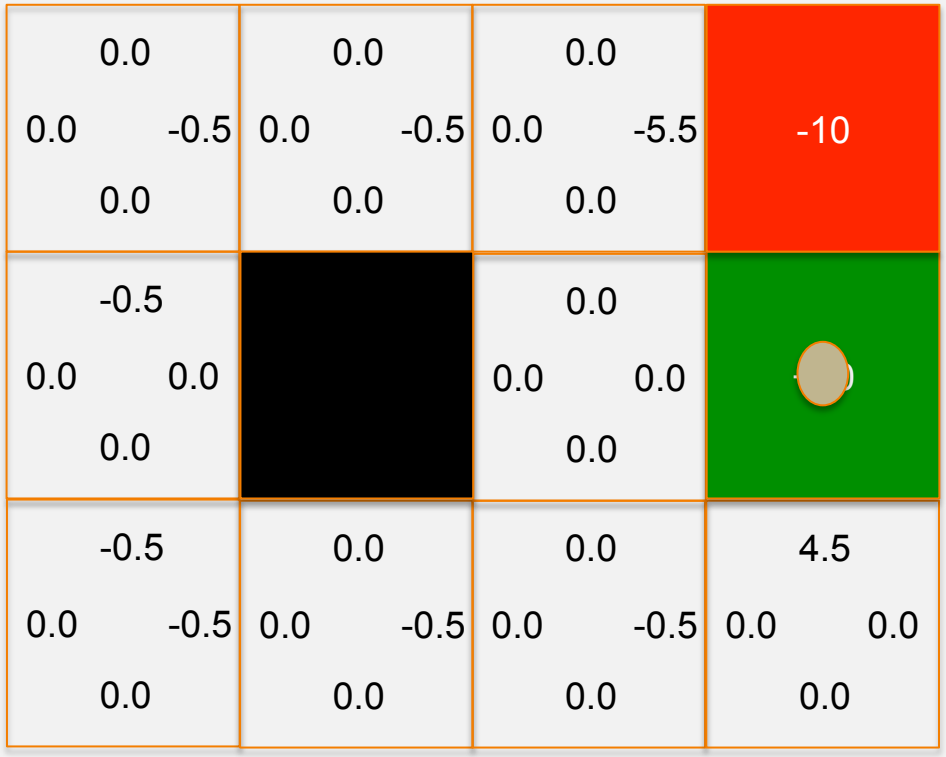
(1-0.5)0+0.5[-1 + 1* 10] = 4.5

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$

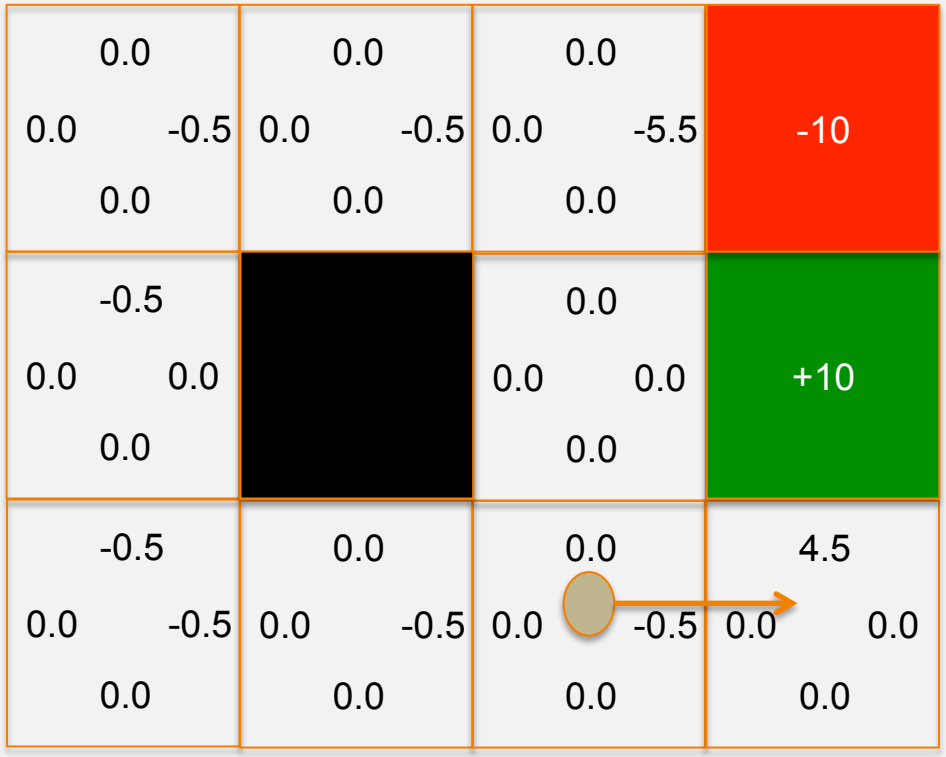


Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



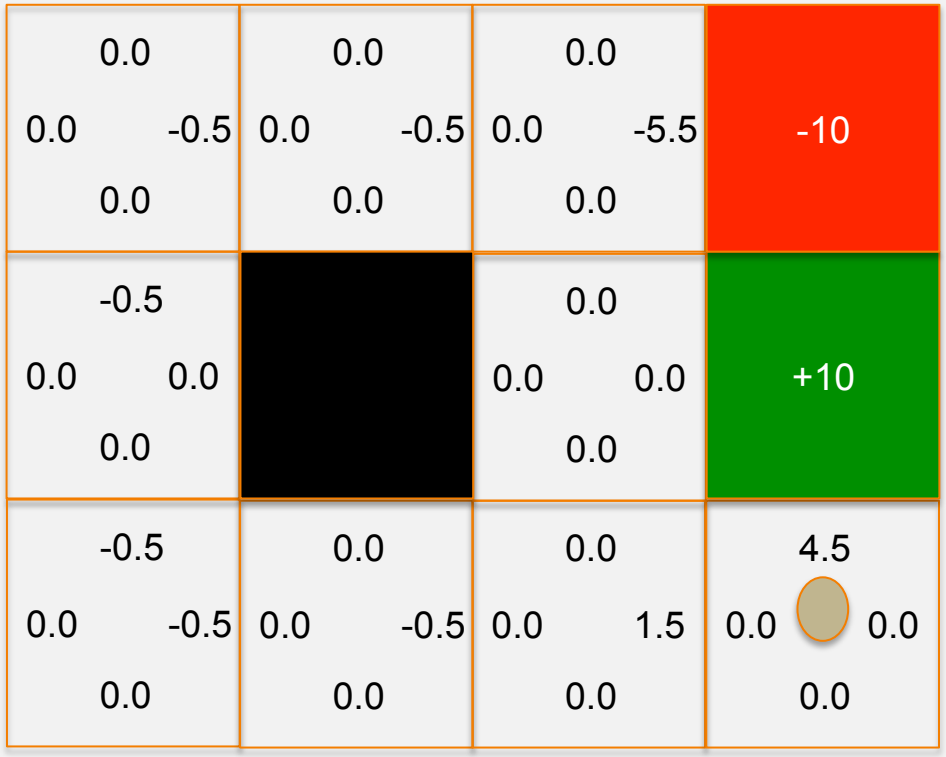
(1-0.5)(-0.5)+0.5[-1 + 1* 4.5] = 1.5

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$

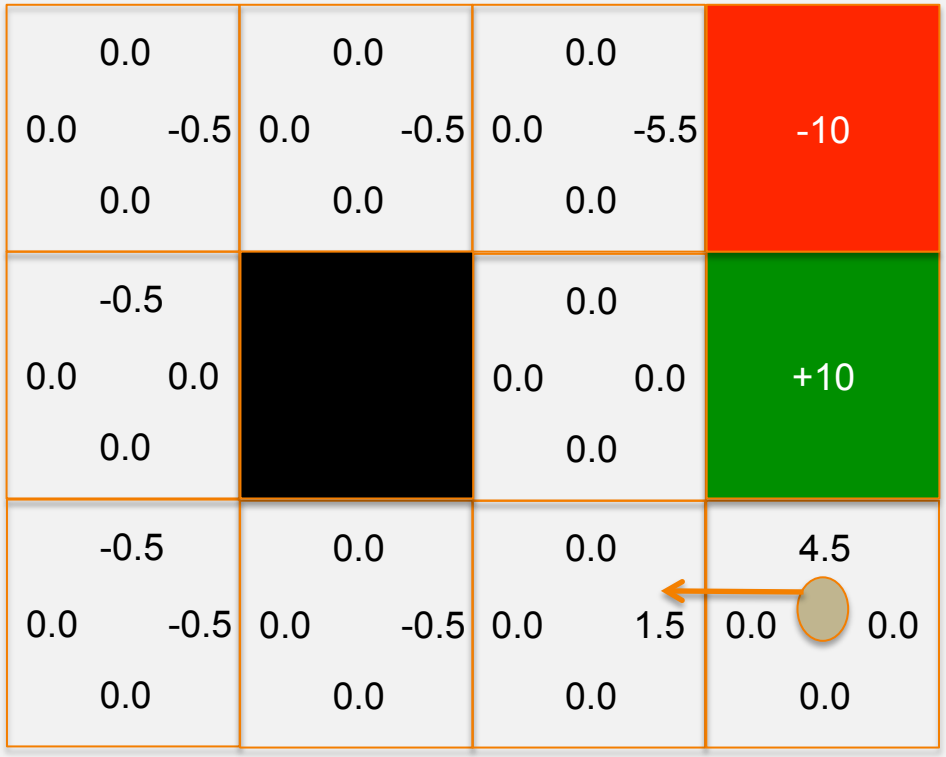


Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



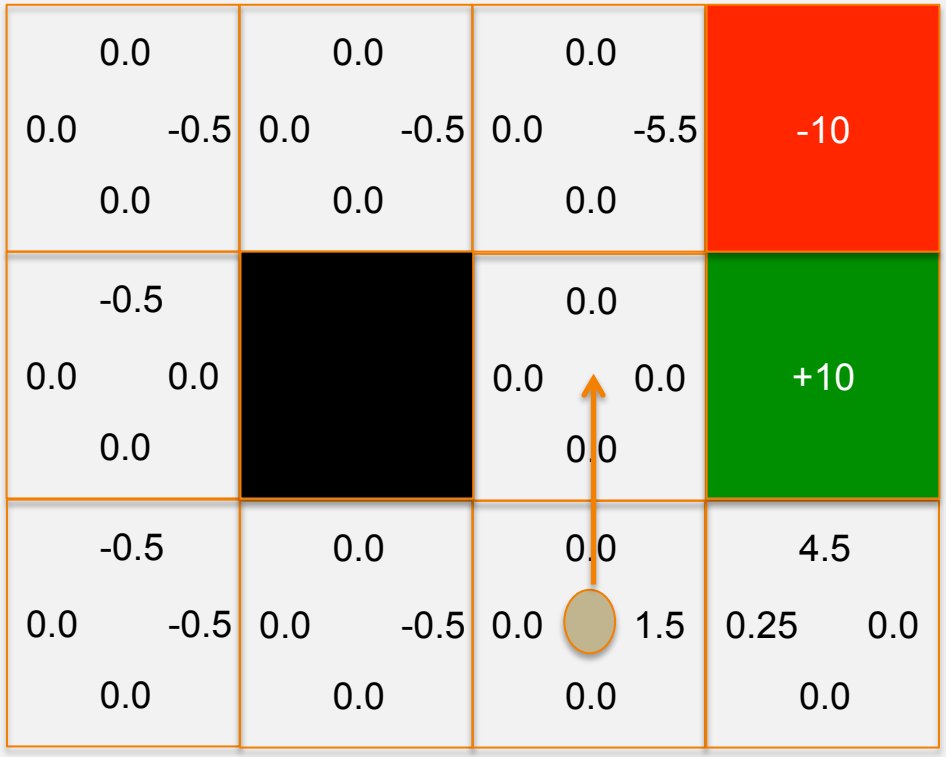
(1-0.5)(0.0)+0.5[-1 + 1* 1.5] = 0.25

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



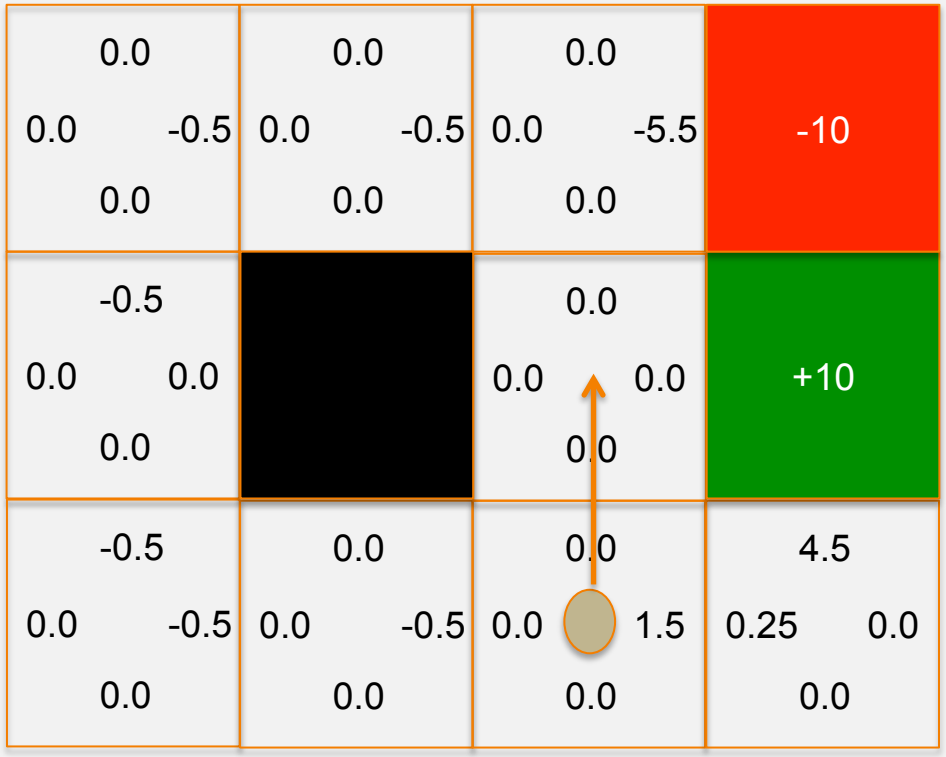
(1-0.5)(0.0)+0.5[-1 + 1* 1.5] = 0.25

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



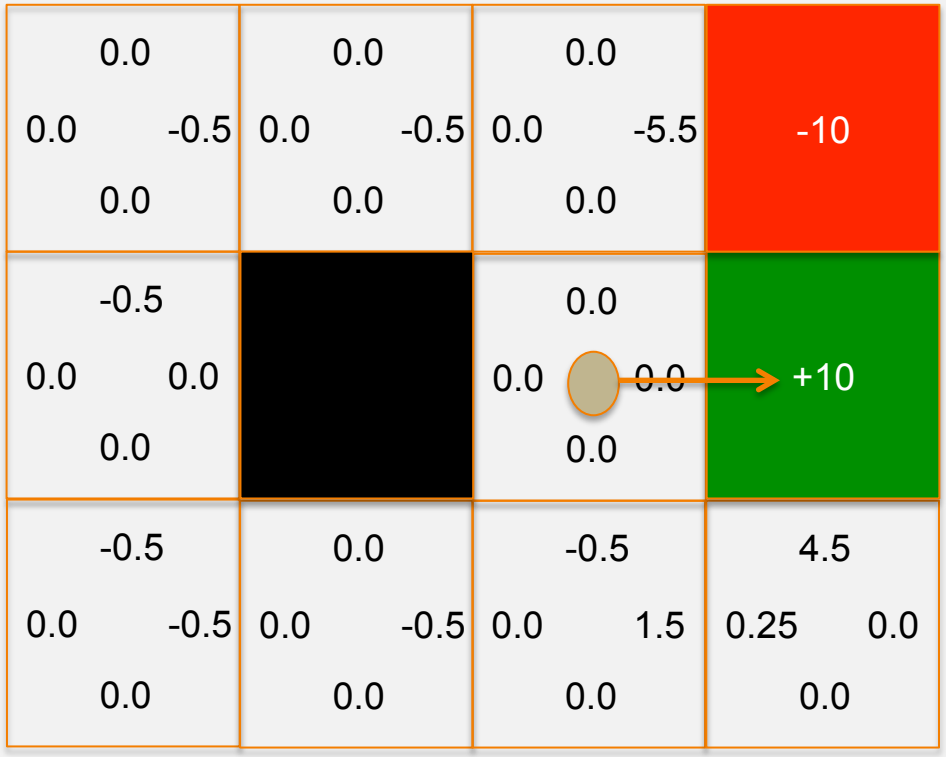
(1-0.5)(0.0)+0.5[-1 + 1* 0.0] = -0.5

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



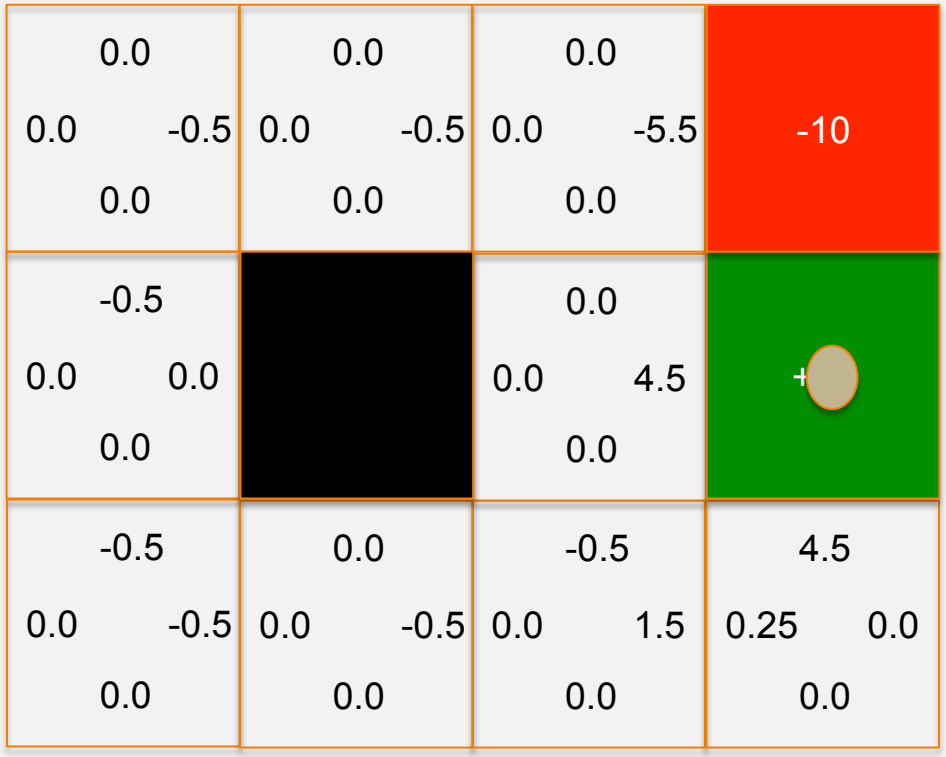
(1-0.5)(0.0)+0.5[-1 + 1* 10] = 4.5

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



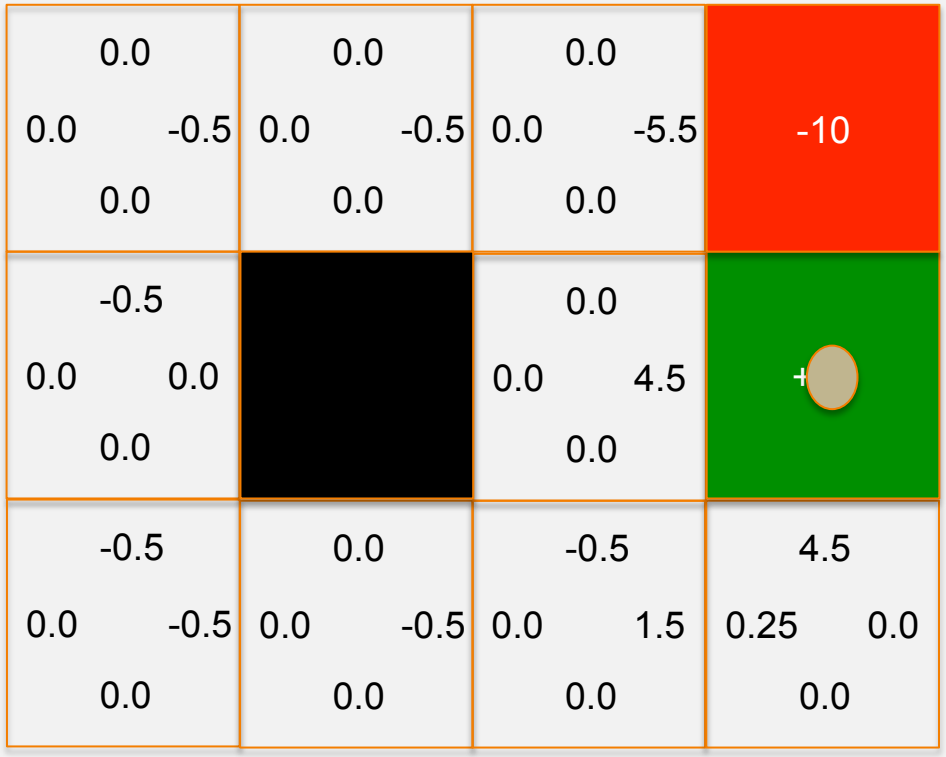
(1-0.5)(0.0)+0.5[-1 + 1* 10] = 4.5

Toy example

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

$\gamma = 1.0$

$\alpha = 0.5$



(1-0.5)(0.0)+0.5[-1 + 1* 10] = 4.5

Toy example

$\gamma = 1.0$

$\alpha = 0.5$

Many moves later

0.0 0.0 -0.5 0.0	0.0 0.0 0.37 0.0	0.0 -0.5 -8.25 2.625	-10
-0.5 0.0 0.0 0.0		0.375 0.0 6.75 0.25	+10
-0.5 0.0 -0.5 0.0	0.0 0.0 0.5 0.0	1.5 0.0 2.5 0.0	6.75 0.25 0.0 0.0

Toy example

$\gamma = 1.0$

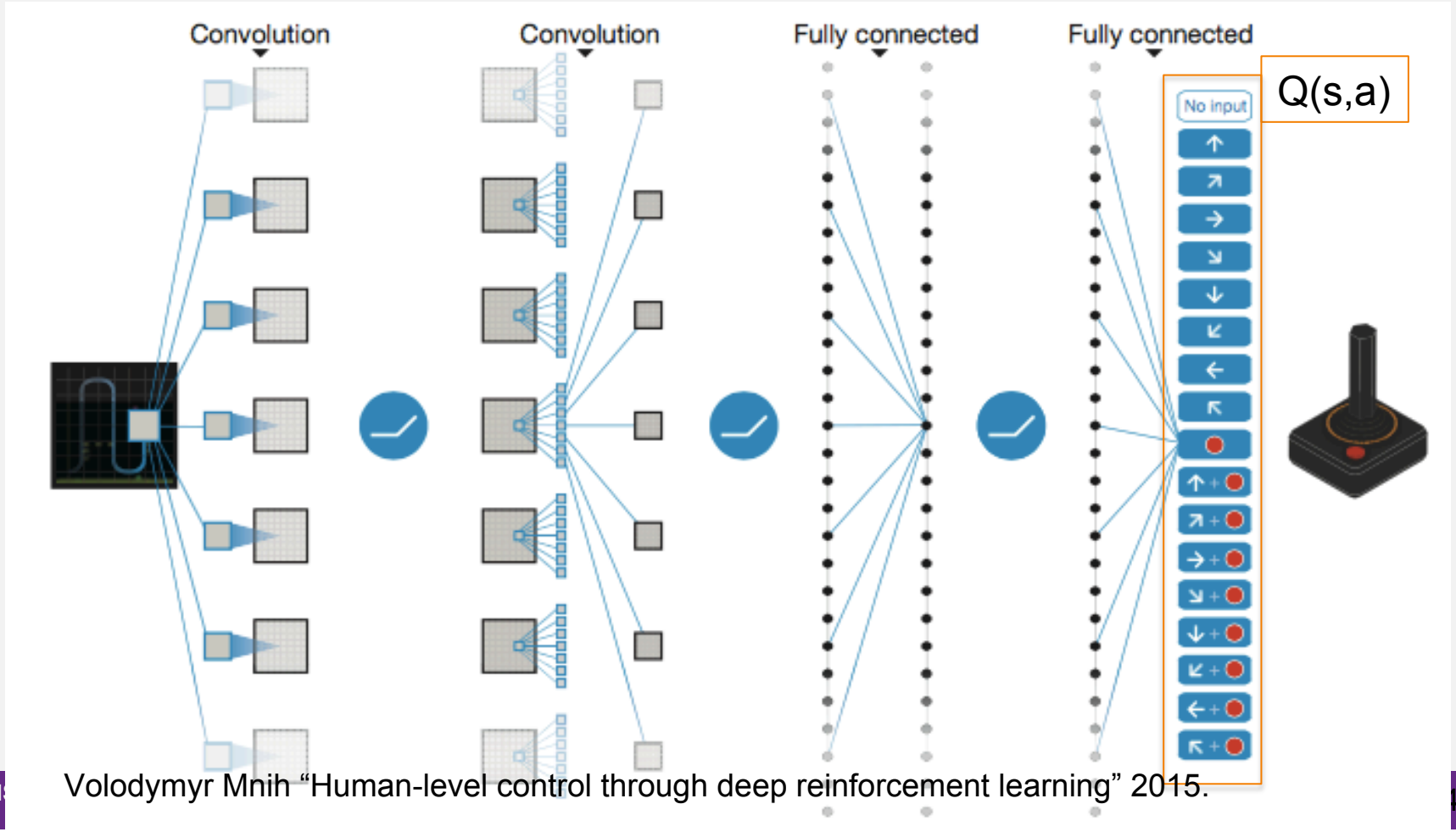
$\alpha = 0.5$

Many many more moves later

0.0 0.0 6.0 4.0	0.0 5.0 7.0 0.0	0.0 6.0 -11 8.0	-10
5.0 0.0 0.0 3.0		7.0 0.0 9.0 7.0	+10
4.0 0.0 6.0 0.0	0.0 5.0 7.0 0.0	8.0 6.0 8.0 0.0	9.0 7.0 0.0 0.0

Deep Q learning (DQN)

Use a DNN to learn the Q function. Outputs are the possible actions.



Volodymyr Mnih “Human-level control through deep reinforcement learning” 2015.

DQN Loss Function

Similar to the update equation of Q-learning

$$\text{new}Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

We want to move our current belief (the Q generated from the network) towards the sampled Q

$$Loss = \{Q(s,a) - [r + \gamma \max_{a' \in A} Q(s',a')]\}^2$$

A couple more tricks

Experience replay

- Adjacent frames in a game are highly correlated
- Saves past experiences and re-use them in the mini-batch

Target network

- Updating the Q belief too quickly leads to instability
- Keeps a target network that updates less frequently.

$$Loss = \{Q(s,a) - [r + \gamma \max_{a' \in A} \hat{Q}(s',a')]\}^2$$

Recent developments

Double Q-Learning, 2015
https://arxiv.org/abs/1509.06461

Dueling Q-network, 2015
https://arxiv.org/abs/1511.06581

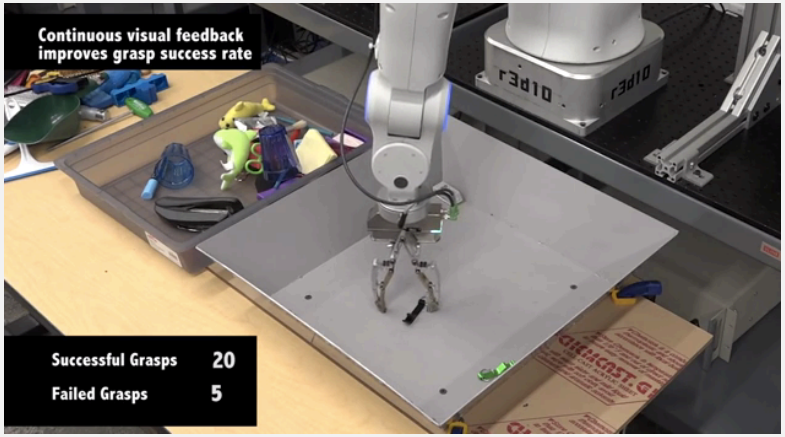
Prioritized experience replay, 2015
https://arxiv.org/abs/1511.05952

Neural episodic control, 2017
https://arxiv.org/abs/1703.01988

Other applications using reinforcement learning

Robotics

http://spectrum.ieee.org/automaton/robotics/drones/drone-uses-ai-and-11500-crashes-to-learn-how-to-fly



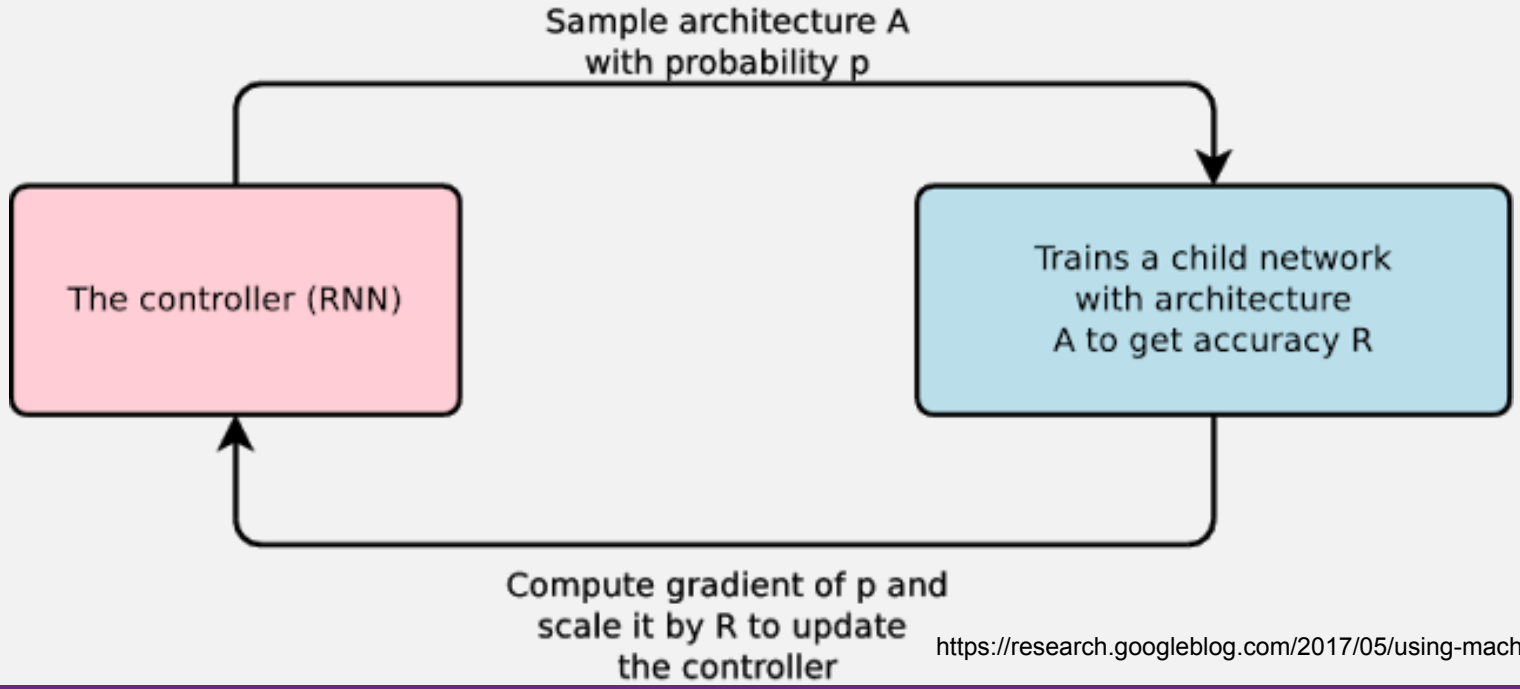
https://research.googleblog.com/2016/03/deep-learning-for-robots-learning-from.html

Model selection for deep architectures

Tuning a network takes time

Let machine learning learns how to tune a network

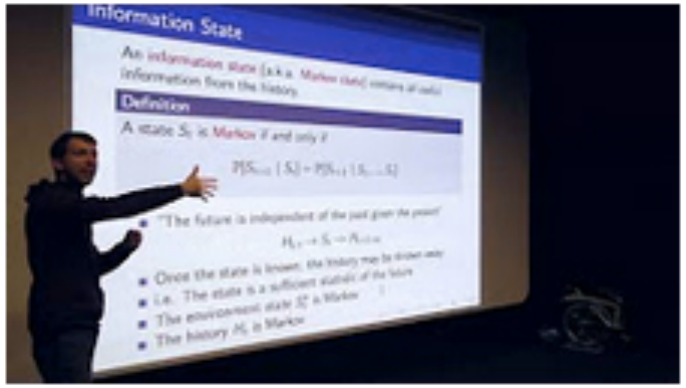
Matches or outperforms ML experts performance



https://research.googleblog.com/2017/05/using-machine-learning-to-explore.html

Where to learn more?

https://www.youtube.com/playlist?list=PLLHj4I_rxKAV_F-rJGI_NQmpdcFwFi-Z7



David Silver: RL Course

Kerawit • 10 videos • 29 views • Last updated on Mar 19, 2017

▶ Play all Share + Save

Skynet yet?

Yann LeCun

March 14, 2016 ·

Follow

Statement from a Slashdot post about the AlphaGo victory: "We know now that we don't need any big new breakthroughs to get to true AI"

That is completely, utterly, ridiculously wrong.

As I've said in previous statements: most of human and animal learning is unsupervised learning. If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake.

We need to solve the unsupervised learning problem before we can even think of getting to true AI. And that's just an obstacle we know about. What about all the ones we don't know about?

#deeplearning #AI #AlphaGo

https://www.facebook.com/yann.lecun/posts/10153426023477143

Conclusion

Reinforcement learning learns the optimal behavior through trial and error

Q-learning is a framework to learn the optimal policy based on states, actions, and rewards

Deep Q learning uses deep learning to approximate the Q function

Reinforcement learning framework