

S&DS 365 / 665  
Intermediate Machine Learning

# Deep reinforcement Learning

April 6

Yale

# Reminders

- Quiz 3 available at 1pm today
  - Available until Friday at 1pm
  - 25 minutes once started
- Assignment 3 due week from today

# Outline

- Quick recap
  - Q-learning
  - Illustration on taxi problem
  - RL concepts
- Bellman equation
- Deep reinforcement learning

# Reinforcement learning

- An agent interacts with an environment
- The actions the agent takes change the state of the environment
- The agent receives rewards for each action, and seeks to maximize the total cumulative reward

*Reinforcement learning is a framework for sequential decision making to achieve a long-term goal.*

# Reinforcement learning: Formalization

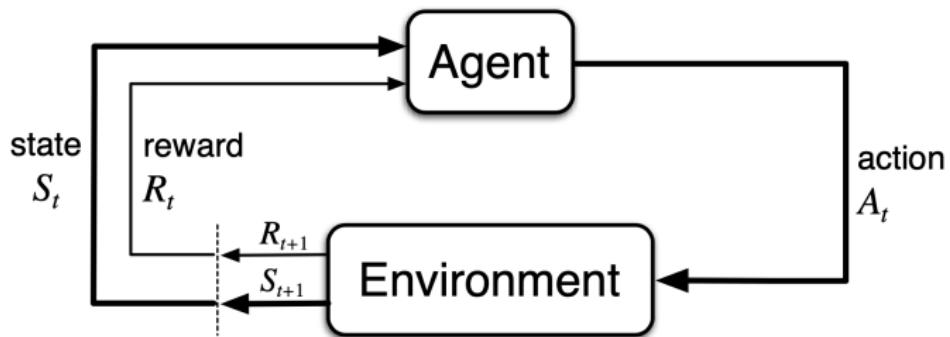
- The environment is in state  $s$  at a given time
- The agent takes action  $a$
- The environment transitions to state  $s' = \text{next}(s, a)$
- The agent receives reward  $r = \text{reward}(s, a)$

# Reinforcement learning: Formalization

- The environment is in state  $s$  at a given time
- The agent takes action  $a$
- The environment transitions to state  $s' = \text{next}(s, a)$
- The agent receives reward  $r = \text{reward}(s, a)$

This is said to be a *Markov decision process*. It's "Markov" because the next state only depends on the current state and the action selected. It's a "decision process" because the agent is making choices of actions in a sequential manner.

# Keep in mind the basic setup



# Important RL concepts

*Policy*: A mapping from states to actions. An algorithm/rule to make decisions at each time step, designed to maximize the long term reward.

# Important RL concepts

*Value function*: A mapping from states to total reward. The total reward the agent can expect to accumulate in the future, starting from that state.

Rewards are short term. Values are predictions of future rewards.

# Q-learning

- Maintains a “quality” variable  $Q(s, a)$  for taking action  $a$  in state  $s$

# Q-learning

- Maintains a “quality” variable  $Q(s, a)$  for taking action  $a$  in state  $s$
- A measure of the cumulative rewards obtained by the algorithm when it takes action  $a$  in state  $s$

# Q-learning

- Maintains a “quality” variable  $Q(s, a)$  for taking action  $a$  in state  $s$
- A measure of the cumulative rewards obtained by the algorithm when it takes action  $a$  in state  $s$
- Quality should not be assessed purely based on the reward the action has in the current time step

# Q-learning

- Maintains a “quality” variable  $Q(s, a)$  for taking action  $a$  in state  $s$
- A measure of the cumulative rewards obtained by the algorithm when it takes action  $a$  in state  $s$
- Quality should not be assessed purely based on the reward the action has in the current time step
- Need to take into account the future rewards

# Q-learning

- Maintains a “quality” variable  $Q(s, a)$  for taking action  $a$  in state  $s$
- A measure of the cumulative rewards obtained by the algorithm when it takes action  $a$  in state  $s$
- Quality should not be assessed purely based on the reward the action has in the current time step
- Need to take into account the future rewards
- A type of gradient ascent algorithm

## Q-Learning

Algorithm parameters: step size  $\alpha$ , exploration probability  $\varepsilon$

Initialize  $Q(s, a)$  arbitrarily, except  $Q(\text{terminal}, \cdot) = 0$

**Loop** for each episode:

    Initialize  $S$

**Loop** for each step of episode

        Choose action  $A$  using  $Q$  with  $\varepsilon$ -greedy policy

        Take action  $A$ , observe reward  $R$ , new state  $S'$

$$Q(S) \leftarrow Q(S, A) + \alpha (R + \gamma \max_a Q(S', a) - Q(S, A))$$

$$S \leftarrow S'$$

**Until**  $S$  is terminal

# Recall: Taxi problem

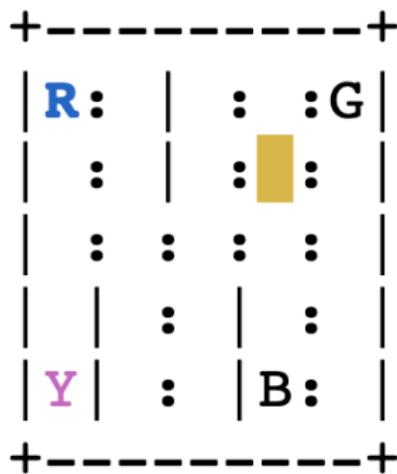
A taxicab drives around the environment, picking up and delivering a passenger at four locations



# Recall: Taxi problem

A taxicab drives around the environment, picking up and delivering a passenger at four locations

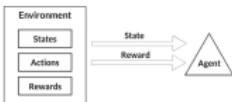
"Ascii art" rendition:



+ Code + Text

## Reinforcement Learning

In reinforcement learning, an agent interacts with the environment, experiencing a series of rewards based on its actions. The agent seeks to maximize its rewards by developing a strategy that learns to choose appropriate actions in each state.

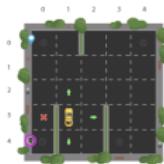


In this notebook we demo the Q-learning algorithm, one of the fundamental algorithms of reinforcement learning. We illustrate Q-learning on the taxicab problem formulated by Tom Dietterich in the paper "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition", as developed in the code from [OpenAI gym](#). Our presentation follows [this tutorial](#).

We'll need the OpenAI gym package. This can be installed as shown below. We'll display some simple graphics using `IPython.display`.

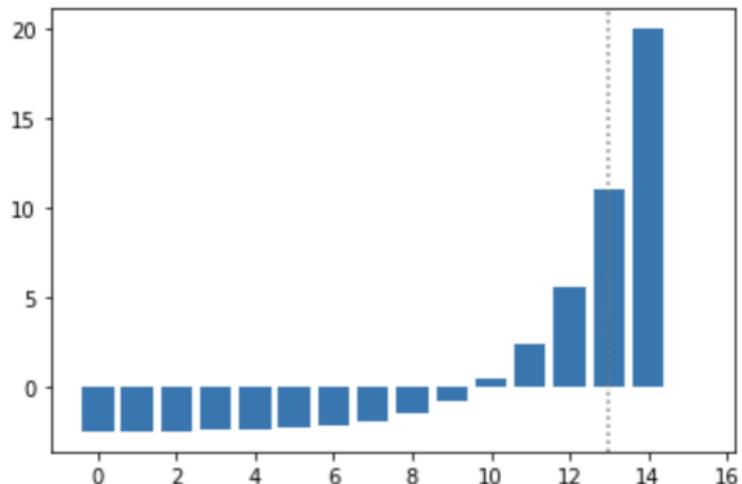
```
[ ] #!pip install gym
import gym
import numpy as np
from IPython.display import clear_output
from time import sleep
```

The environment is a simple grid, with some barriers inserted to make things more interesting. A taxicab drives around the environment, picking up and delivering a passenger at four locations. A graphic of the environment is shown below.



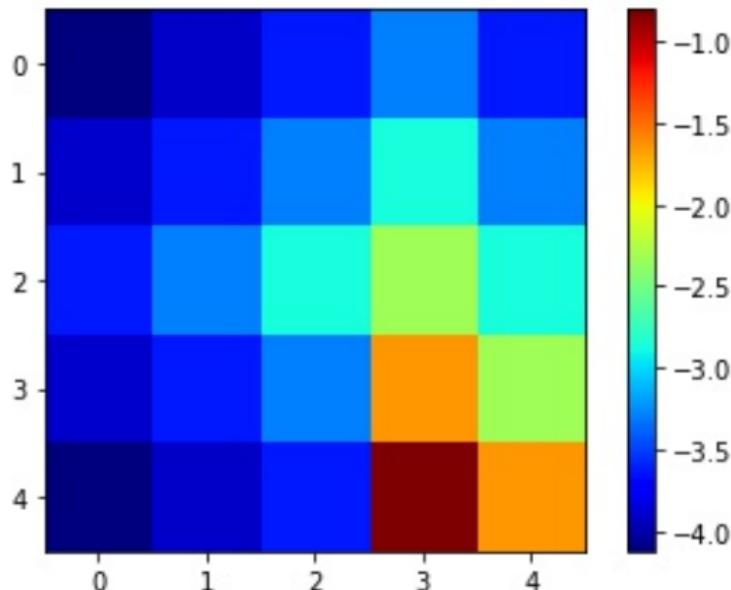
R:	:	:	: G
:	:	:	 G
:	:	:	
:	:	:	
Y	:	B:	

(East)



# Question

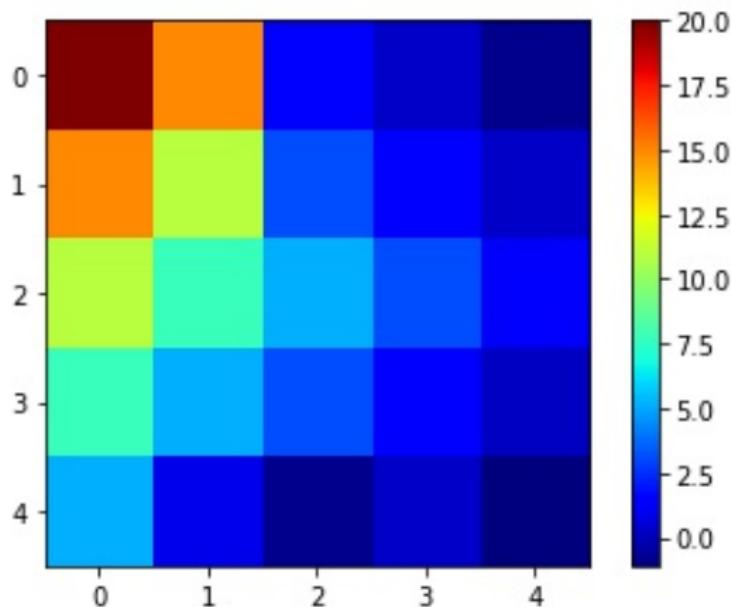
For a fixed passenger location and destination, the value function  $v(\text{row}, \text{col})$  assigns a value to each of the  $25 = 5 \times 5$  grid points.



Is the passenger waiting, or in the taxi?

# Question

For a fixed passenger location and destination, the value function  $v(\text{row}, \text{col})$  assigns a value to each of the  $25 = 5 \times 5$  grid points.



Is the passenger waiting, or in the taxi?

# **$Q$ -learning update**

**Update:**

$$Q(s, a) \leftarrow$$

$$Q(s, a) + \alpha \left( \text{reward}(s, a) + \gamma \max_{a'} Q(\text{next}(s, a), a') - Q(s, a) \right)$$

# **$Q$ -learning update**

## Update:

$$Q(s, a) \leftarrow$$

$$Q(s, a) + \alpha \left( \text{reward}(s, a) + \gamma \max_{a'} Q(\text{next}(s, a), a') - Q(s, a) \right)$$

- When action  $a$  is taken in state  $s$ , reward  $\text{reward}(s, a)$  is given
- Then, the algorithm moves to a new state  $\text{next}(s, a)$

# **$Q$ -learning update**

## Update:

$$Q(s, a) \leftarrow$$

$$Q(s, a) + \alpha \left( \text{reward}(s, a) + \gamma \max_{a'} Q(\text{next}(s, a), a') - Q(s, a) \right)$$

- Cumulative future reward of this action is  $\max_{a'} Q(\text{next}(s, a), a')$
- Future rewards discounted by factor  $\gamma < 1$
- Trades off short-term against long-term rewards

# Bellman equation



The optimal value function is the largest expected discounted long term reward starting from that state.

# Bellman equation: Deterministic case

The optimality condition for the value function  $v_*$  is

$$v_*(s) = \max_a \left\{ \text{reward}(s, a) + \gamma v_*(\text{next}(s, a)) \right\}$$

# Bellman equation: Deterministic case

The optimality condition for the  $Q$ -function is

$$Q_*(s, a) = \text{reward}(s, a) + \gamma \max_{a'} Q_*(\text{next}(s, a), a')$$

and then  $v_*(s) = \max_{a'} Q_*(s, a')$

# **$Q$ -learning update**

Note how this makes sense in terms of the update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left( \text{reward}(s, a) + \gamma \max_{a'} Q(\text{next}(s, a), a') - Q(s, a) \right)$$

# Bellman equation: Deterministic case

*If we know  $Q_*$ , we know  $v_*$ :*

# Bellman equation: Deterministic case

If we know  $Q_*$ , we know  $v_*$ :

$$\begin{aligned}v_*(s) &= \max_a Q_*(s, a) \\&= \max_a \left\{ \text{reward}(s, a) + \gamma \max_{a'} Q_*(\text{next}(s, a), a') \right\} \\&= \max_a \left\{ \text{reward}(s, a) + \gamma v_*(\text{next}(s, a)) \right\}\end{aligned}$$

which is the Bellman equation

# Bellman equation: Random environments

The optimality condition for the value function  $v_*$  is

$$\begin{aligned}v_*(s) &= \max_a \sum_{s',r} p(s', r | s, a) \left\{ r + \gamma v_*(s') \right\} \\&= \max_a \mathbb{E} \left[ R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a \right]\end{aligned}$$

# Bellman equation: Random environments

## Value function optimality

$$v_*(s) = \max_a \mathbb{E} \left[ R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a \right]$$

# Bellman equation: Random environments

The optimality condition for the  $Q$ -function

$$\begin{aligned} Q_*(s, a) &= \sum_{s', r} p(s', r | s, a) \left\{ r + \gamma \max_{a'} Q_*(s', a') \right\} \\ &= \mathbb{E} \left[ R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right] \end{aligned}$$

# Bellman equation: Random environments

## **Q-function optimality**

$$Q_*(s, a) = \mathbb{E} \left[ R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right]$$

# Comment on Q-learning

- Q-learning is an example of *temporal difference (TD) learning*
- It is an “off-policy” approach that is practical if the space of actions is small
- Value iteration is analogous approach for learning the value function for a given policy  $\pi$ , a (possibly random) choice of action for each state

# Deep reinforcement learning: Motivation

- Direct implementation of Q-learning only possible for small state and action spaces
- For large state spaces we need to map states to “features”
- Deep RL uses a multilayer neural network to learn these features and the  $Q$ -function

# Starting point: Bellman equation

$$Q(s, a; \theta) = \mathbb{E} \left[ R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; \theta) \mid S_t = s, A_t = a \right]$$

- The parameters  $\theta$  are weights in a neural network
- The state  $S_{t+1}$  is the input to the network
- Each possible action  $a$  is assigned a value by the network

*How do we solve this implicit equation for the network parameters?*

# Strategy

Objective:

$$Q(s, a; \theta) = \mathbb{E} \left[ R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; \theta) \mid S_t = s, A_t = a \right]$$

Let  $y_t$  be a sample from this conditional distribution:

$$y_t = R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; \theta_{\text{current}})$$

Adjust the parameters  $\theta$  to make the squared error small (SGD):

$$(y_t - Q(s, a; \theta))^2$$

# Strategy

Adjust the parameters  $\theta$  to make the squared error small

$$(y_t - Q(s, a; \theta))^2$$

How? Carry out SGD

$$\theta \longleftarrow \theta + \eta (y_t - Q(s, a; \theta)) \nabla_{\theta} Q(s, a; \theta)$$

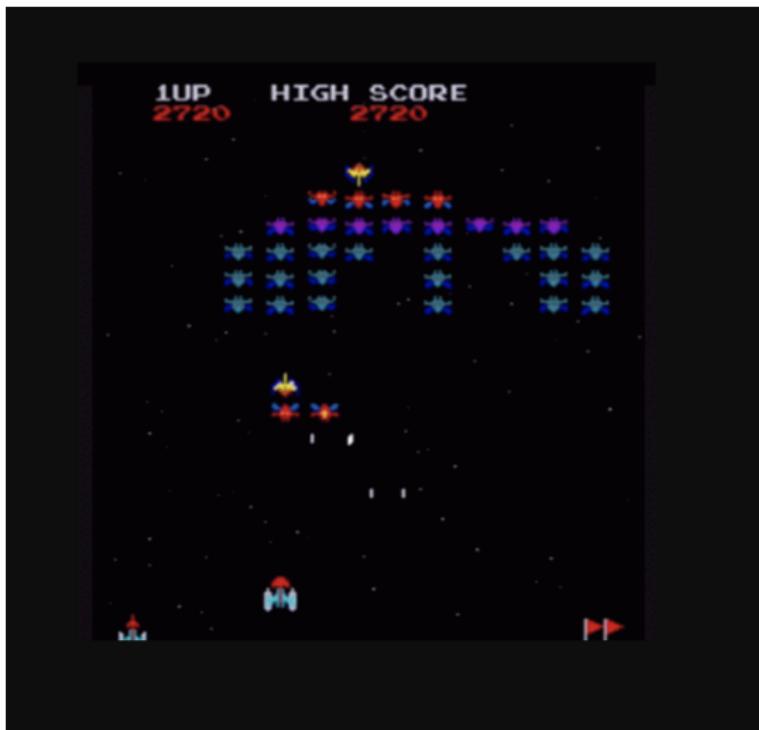
using backpropagation!

# When does learning take place?

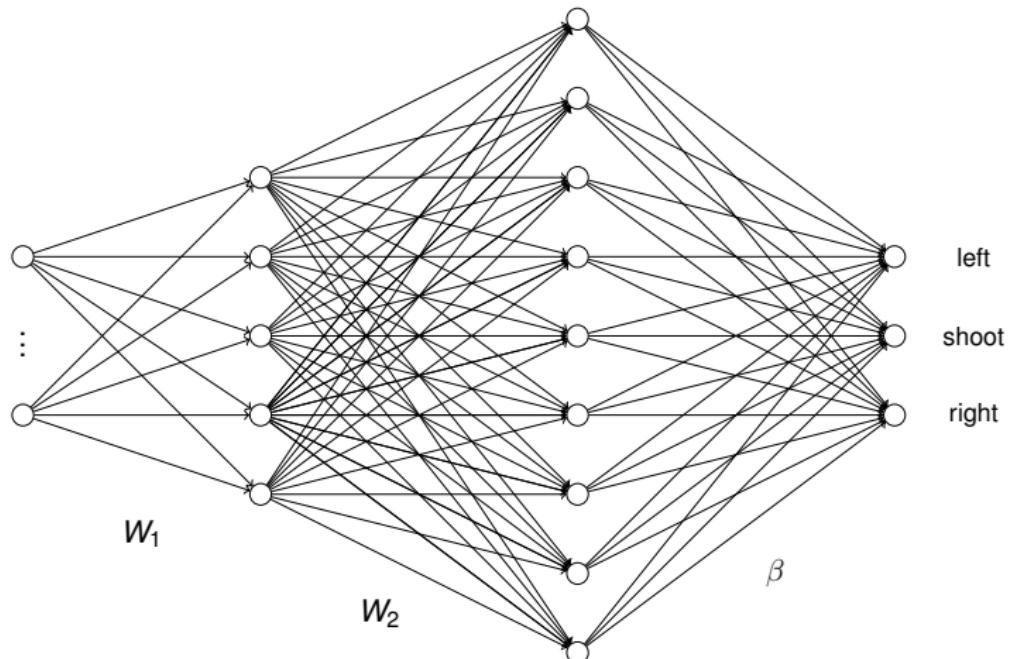
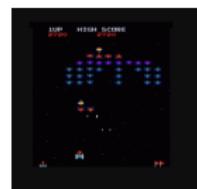
Recall from Bellman equation that  $y_t$  is an expectation.

*Learning takes place when expectations are violated. The receipt of the reward itself does not cause changes.*

# Space Invaders



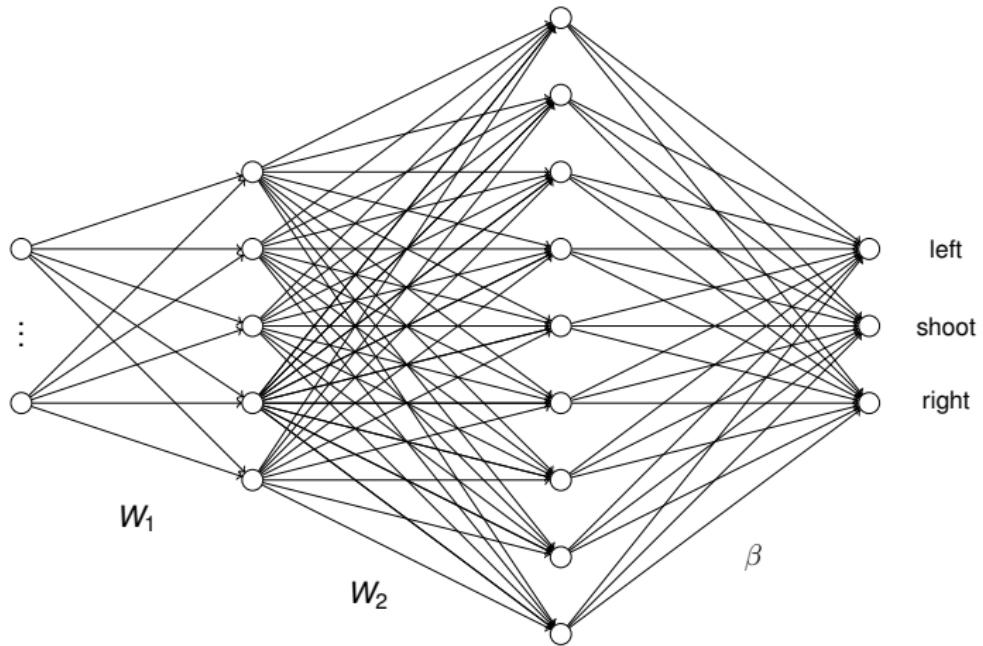
# Space Invaders: Q-learning framework



# Space Invaders: Q-learning framework



(multiple frames)



# DeepMind work

- Images cropped to  $84 \times 84$  pixels; 128 color palette; input sequence of 4 frames; reward is score
- 3-layer convolutional neural network, ReLU nonlinearity, final layer fully connected, 256 neurons
- Q-learning carried out over minibatches of playing sequences that are “remembered and replayed”

---

“Playing Atari with Deep Reinforcement Learning,” Mnih et al., DeepMind Technologies, 2013. Larger versions reached human level performance on 29/49 games. Google acquired DeepMind and used Deep Q-learning to save energy (and money) in data centers.

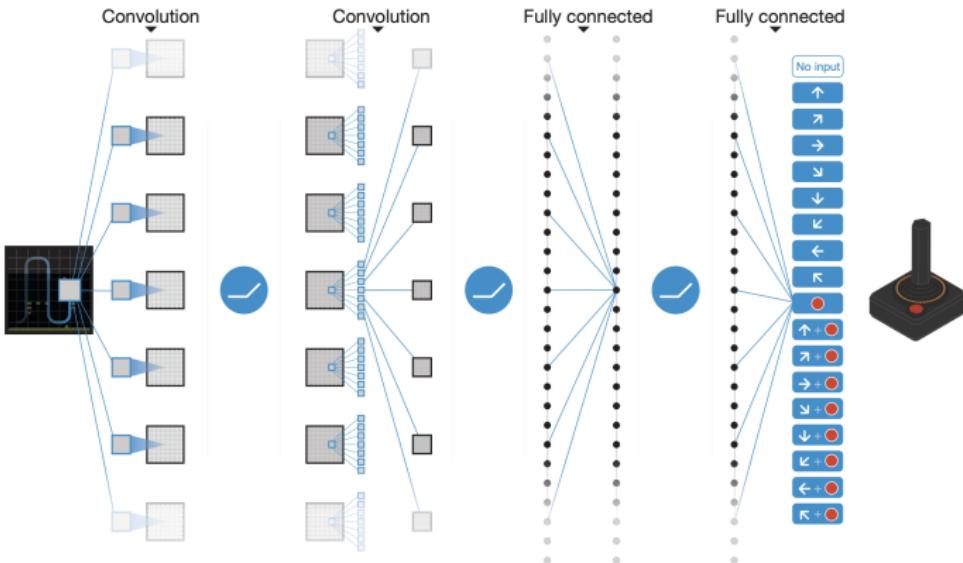
# DeepMind work

- Identical architecture used for seven games (Beam Rider, Breakout, Enduro, Pong, Q\*bert, Seaquest, Space Invaders)
- Each game trained for 100 epochs (50,000 minibatch weight updates / epoch), about 50 hours
- Surpasses human expert on three of seven games

---

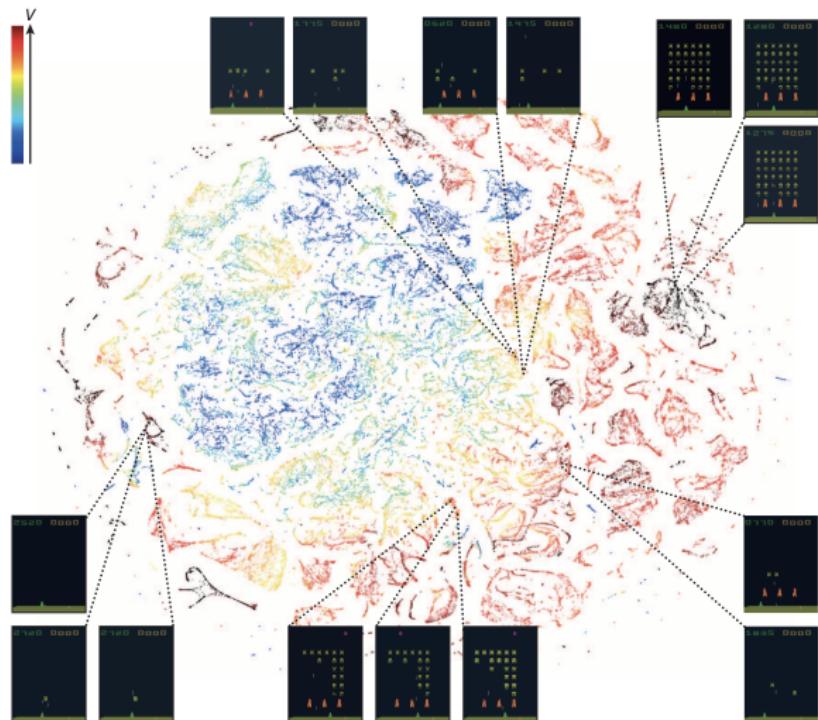
"Playing Atari with Deep Reinforcement Learning," Mnih et al., DeepMind Technologies, 2013. Larger versions reached human level performance on 29/49 games. Google acquired DeepMind and used Deep Q-learning to save energy (and money) in data centers.

# Second generation DQN



<https://storage.googleapis.com/deepmind-data/assets/papers/DeepMindNature14236Paper.pdf>

# Second generation DQN: Interpretation



t-SNE representations of last layer for Space Invaders, color-coded for  $v_*$ .

# Further examples

BLOG POST  
RESEARCH

## Deep Reinforcement Learning

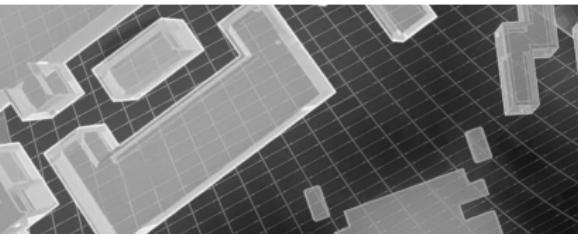
---

SHARE

---

AUTHORS

David Silver



Humans excel at solving a wide variety of challenging problems, from low-level motor control through to high-level cognitive tasks. Our goal at DeepMind is to create artificial agents that can achieve a similar level of performance and generality. Like a human, our agents learn for themselves to achieve successful strategies that lead to the greatest long-term rewards. This paradigm of learning by trial-and-error, solely from rewards or punishments, is known as [reinforcement learning \(RL\)](#). Also like a human, our agents construct and learn their own knowledge directly from raw inputs, such as vision, without any hand-engineered features or domain heuristics. This is achieved by [deep learning of neural](#)

<https://deepmind.com/blog/article/deep-reinforcement-learning>

## A Neural Substrate of Prediction and Reward

Wolfram Schultz, Peter Dayan, P. Read Montague\*

The capacity to predict future events permits a creature to detect, model, and manipulate the causal structure of its interactions with its environment. Behavioral experiments suggest that learning is driven by changes in the expectations about future salient events such as rewards and punishments. Physiological work has recently complemented these studies by identifying dopaminergic neurons in the primate whose fluctuating output apparently signals changes or errors in the predictions of future salient and rewarding events. Taken together, these findings can be understood through quantitative theories of adaptive optimizing control.

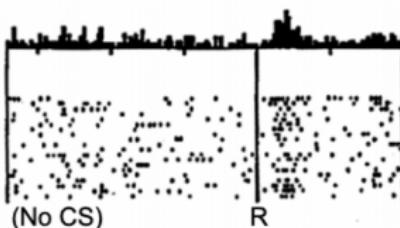
---

Science 1997

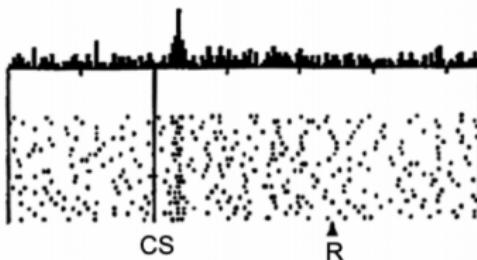
# Neuroscience connection

Do dopamine neurons report an error  
in the prediction of reward?

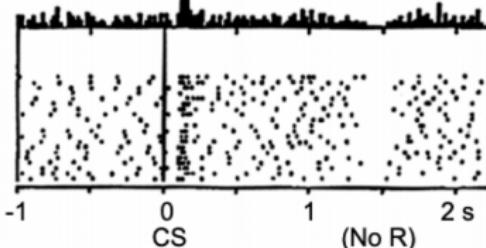
No prediction  
Reward occurs



Reward predicted  
Reward occurs



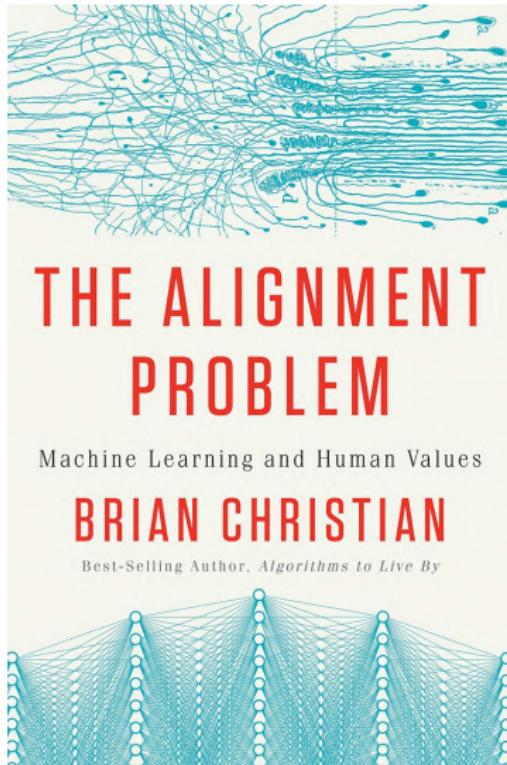
Reward predicted  
No reward occurs



# Dopamine

- Dopamine neurons fire when there is an error — expectations are violated
- Initially they fire when a reward is received when none was expected
- When a light is flashed before the reward (conditioning stimulus CS), dopamine firing moves to the stimulus, as the animal learns the association
- When the reward is removed, an error in expectation results; dopamine is again released

# Recommended reading



# Summary

- For large state spaces, can't estimate  $Q$ -function directly
- Deep  $Q$ -learning: Using a (deep) neural network with state as input, value assigned to each possible action
- Success at games (Backgammon, Atari, Go)
- Close connection to neuroscience of behavior and reward