S&DS 365 / 665

Intermediate Machine Learning

Deep Reinforcement Learning

November 1

Yale

Reminders

- Assignment 3 due tonight at midnight
- Assignment 4 out today (GNNs and Deep RL)
- Quiz 4 next week

Outline

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

Main reference: “Reinforcement Learning: An Introduction” (Second Edition), Richard S. Sutton and Andrew G. Barto

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

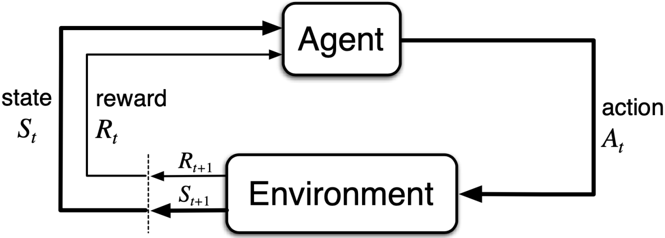
- The environment is in state s at a given time
- The agent takes action a
- The environment transitions to state $s' = \text{next}(s, a)$
- The agent receives reward $r = \text{reward}(s, a)$

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



“Reinforcement Learning: An Introduction” (Second Edition), Richard S. Sutton and Andrew G. Barto

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.

"Reinforcement Learning: An Introduction" (Second Edition), Richard S. Sutton and Andrew G. Barto

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.

"Reinforcement Learning: An Introduction" (Second Edition), Richard S. Sutton and Andrew G. Barto

Q-learning

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

History: Proposed by Chris Watkins; Watkins and Dayan proved convergence (1992)

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

History: Proposed by Chris Watkins; Watkins and Dayan proved convergence (1992)

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

History: Proposed by Chris Watkins; Watkins and Dayan proved convergence (1992)

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

History: Proposed by Chris Watkins; Watkins and Dayan proved convergence (1992)

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

History: Proposed by Chris Watkins; Watkins and Dayan proved convergence (1992)

Q-Learning

Parameters: step size α, exploration probability ε, discount factor γ
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 ε-greedy policy

    Take action A; observe reward R and new state S'

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

    $S \leftarrow S'$

  Until S is terminal

Q-learning update

Update:

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

Q-learning update

Update:

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

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

Q-learning update

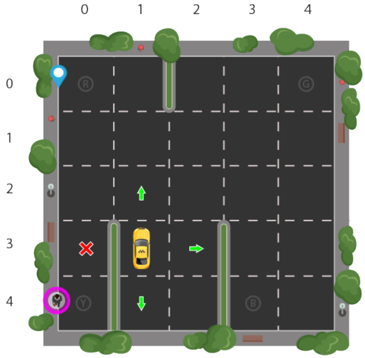
Update:

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

- 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

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:

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

+ 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.

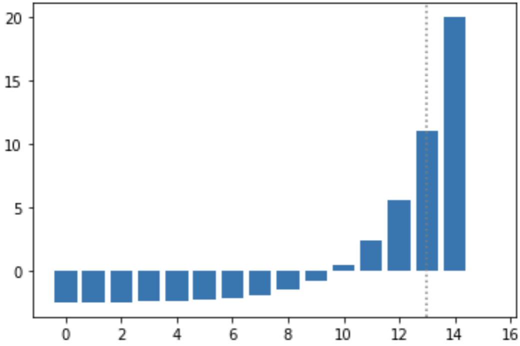
In this notebook we demo the Q-learning algorithm, one of the fundamental algorithms of reinforcement learning. We illustrate Q-learning on the taxicab problem formulated by Tom Dietterich in the paper "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition", as developed in the 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
```

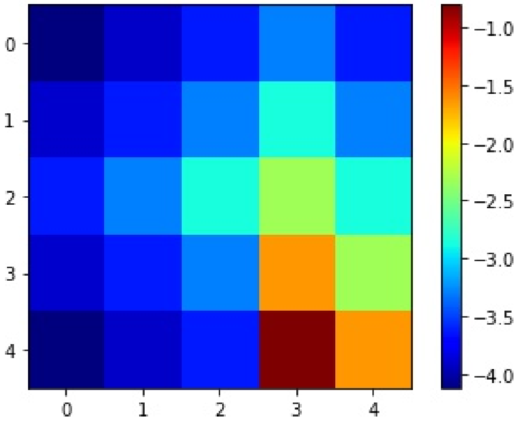
The environment is a simple grid, with some barriers inserted to make things more interesting. A taxicab drives around the environment, picking up and delivering a passenger at four locations. A graphic of the environment is shown below.

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
```



Question

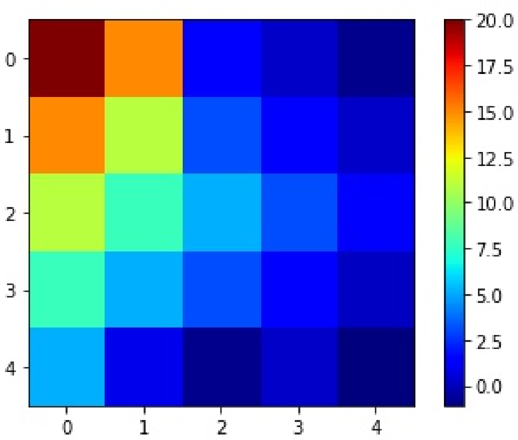
For a fixed passenger location and destination, the value function $v(\texttt{row}, \texttt{col})$ assigns a value to each of the $25 = 5 \times 5$ grid points.



Is the passenger waiting, or in the taxi?

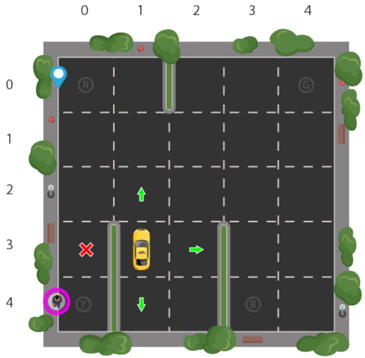
Question

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



Is the passenger waiting, or in the taxi?

Look Mom, no map!



In Q-learning, the agent learns to navigate optimally without learning an explicit map.

Proof of convergence: Watkins and Dayan, “Q-Learning,” *Machine Learning*, 1992

Bellman equation

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

https://en.wikipedia.org/wiki/Richard_E._Bellman

Bellman equation: Deterministic case

The optimality condition for the value function v_* is

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

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:

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

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

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)\Big\{r + \gamma v_*(s')\Big\}$$
$$= \max_a \mathbb{E}\Big[R_{t+1} + \gamma v_*(S_{t+1}) \;\mid\; S_t = s, A_t = a\Big]$$

Bellman equation: Random environments

Value function optimality

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

Bellman equation: Random environments

The optimality condition for the Q-function

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

Bellman equation: Random environments

Q-function optimality

$$Q_*(s,a) = \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') \;\big|\; S_t = s, A_t = a\Big]$$

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 π, 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

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

- The parameters θ 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:

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

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 θ to make the squared error small (SGD):

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

Strategy

Adjust the parameters θ to make the squared error small

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

How? Carry out SGD

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

using backpropagation!

When does learning take place?

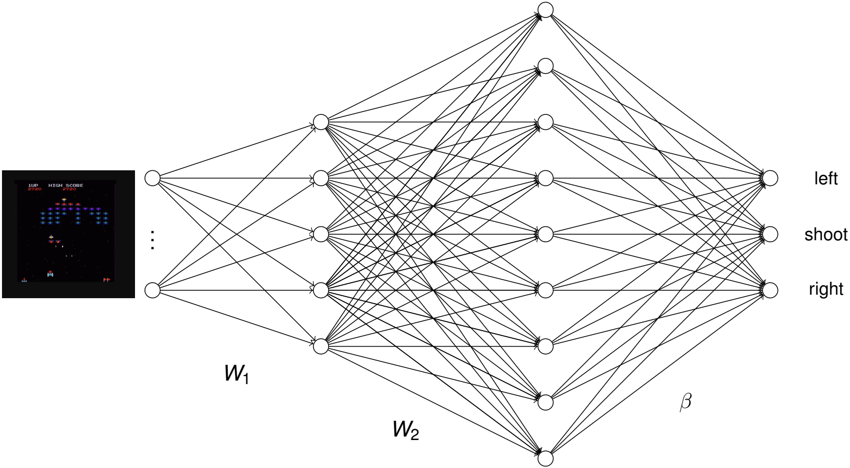
Recall from Bellman equation that y_t is an expectation.

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

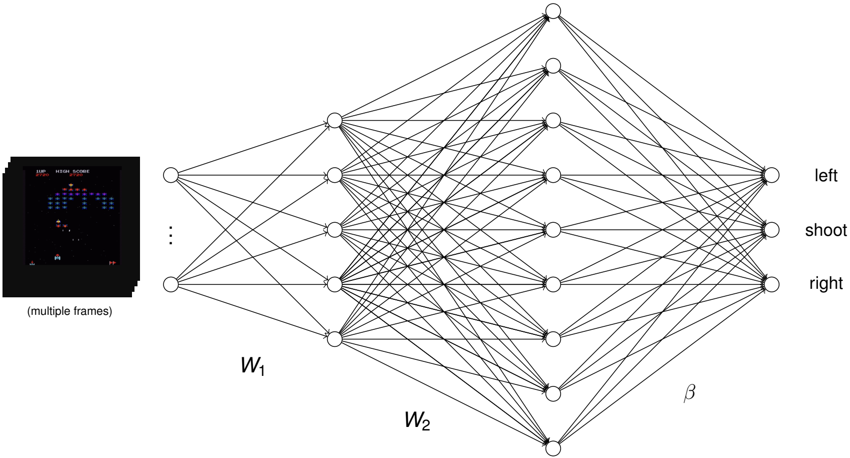
See neuroscience connection below

Space Invaders

Space Invaders: Q-learning framework



Space Invaders: Q-learning framework



Early DeepMind work

- Images cropped to 84×84 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.
https://patentimages.storage.googleapis.com/71/91/4a/c5cf4ffa56f705/US20150100530A1.pdf
Google acquired DeepMind and used Deep *Q*-learning to save energy (and money) in data centers.

Early 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.
https://patentimages.storage.googleapis.com/71/91/4a/c5cf4ffa56f705/US20150100530A1.pdf
Google acquired DeepMind and used Deep *Q*-learning to save energy (and money) in data centers.

Payoff

SUSTAINABILITY

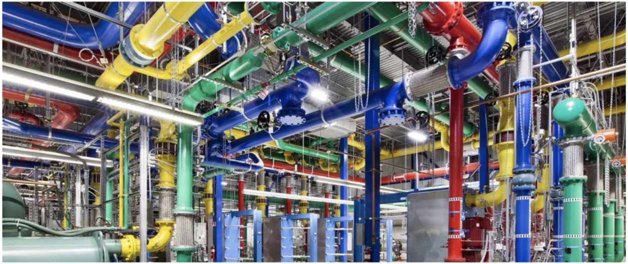
DeepMind AI reduces energy used for cooling Google data centers by 40%

Jul 20, 2016 · 4 min read

R Rich Evans
Research Engineer,
DeepMind

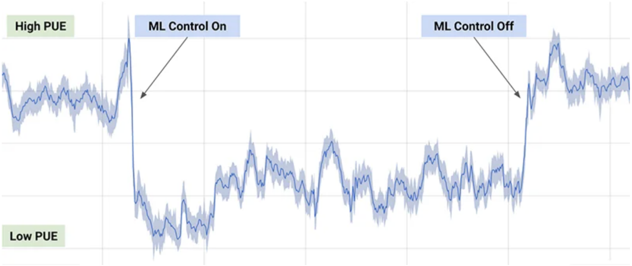
Jim Gao
Data Center Engineer,
Google

Share



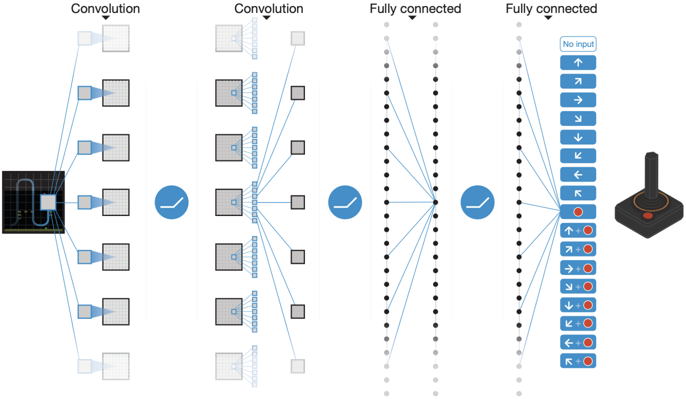
Payoff

We tested our model by deploying on a live data center. The graph below shows a typical day of testing, including when we turned the machine learning recommendations on, and when we turned them off.



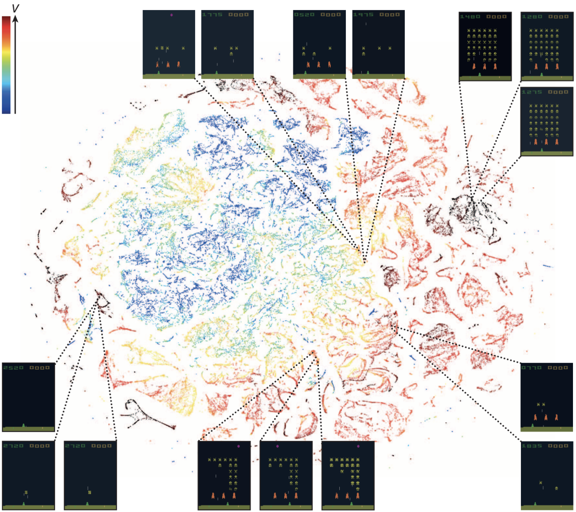
Google DeepMind graph showing results of machine learning test on power usage effectiveness in Google data centers

Second generation DQN



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

Second generation DQN: Interpretation

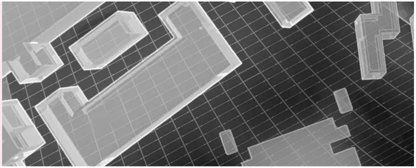


t-SNE represenations of last layer for Space Invaders, color-coded for v_*.

Further examples

BLOG POST
RESEARCH

17 JUN 2016



Deep Reinforcement Learning

SHARE

AUTHORS

DS 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

Neuroscience connection

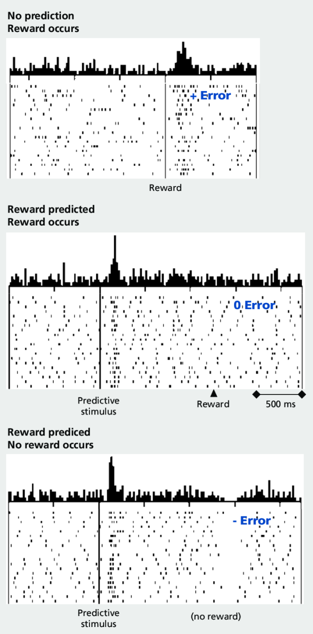
A Neural Substrate of Prediction and Reward

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

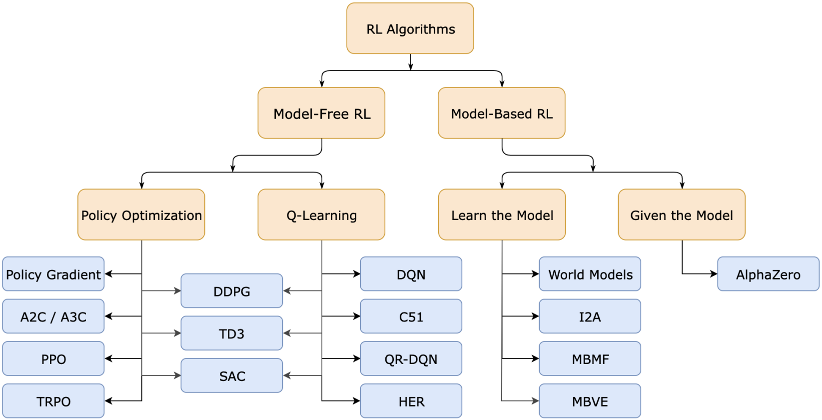
Neuroscience connection



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

RL Framework Zoo



https://spinningup.openai.com/en/latest/index.html

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