AI IN THE BUILT ENVIRONMENT
DCP4300

Week 9: Robotics

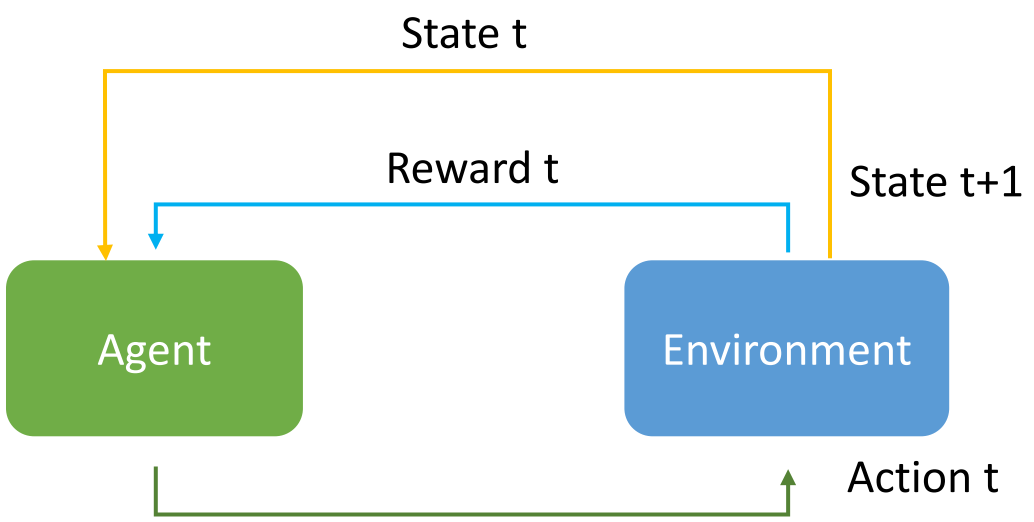
Part B: Reinforcement Learning

Dr. Chaofeng ‘Charles' Wang

University of Florida
College of Design Construction and Planning

Reinforcement Learning:

Train the Agent to learn to React to an Environment by **trial and error**.



Reinforcement Learning:

Train the Agent to learn to React to an Environment by **trial and error**.

Has broad applications. Some examples:

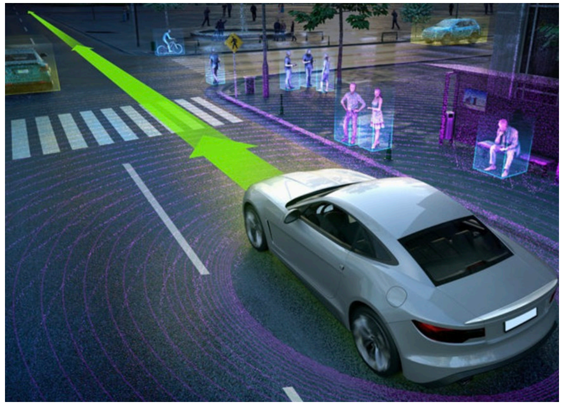
Autonomous Driving

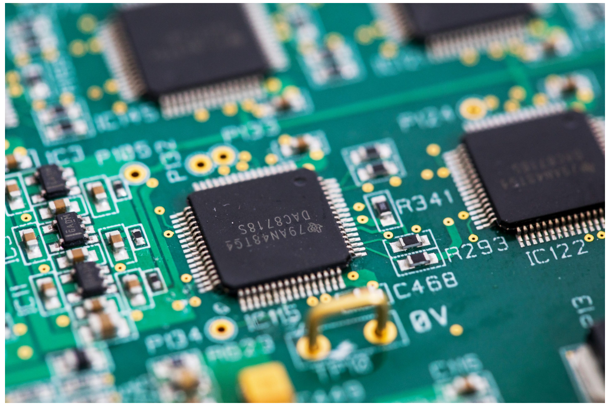
Gaming AI

Robotics

Design

…





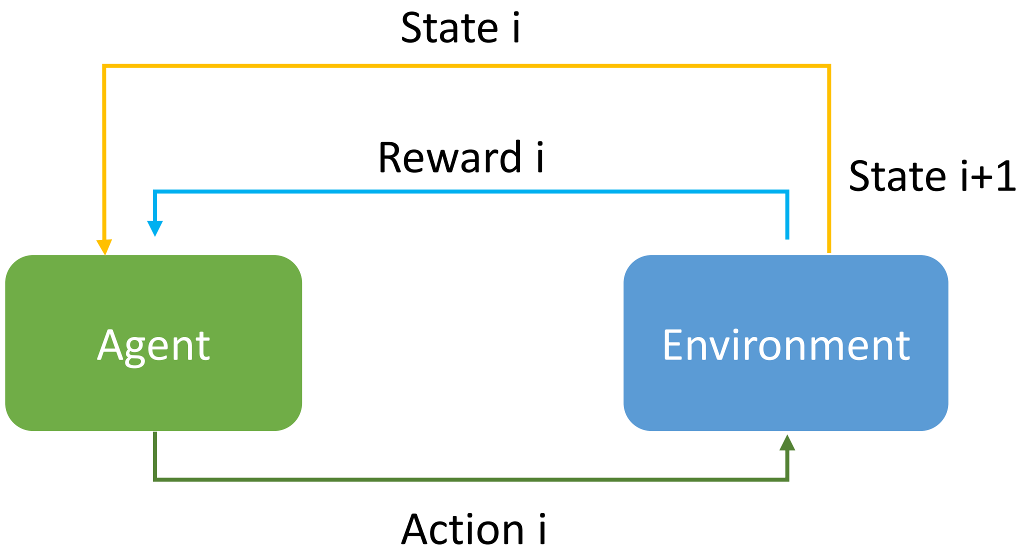
Supervised Learning Vs Reinforcement Learning

They are both trained to learn some functions: $f: X \rightarrow Y$

x1	x2	x3	x4	x5	y
0.21	0.20	0.65	0.87	0.29	0.22
0.83	0.47	0.14	0.77	0.43	0.63
0.42	0.31	0.41	0.43	0.11	0.92
0.83	0.49	0.52	0.01	0.94	0.17
0.99	0.05	0.47	0.72	0.01	0.60
0.31	0.31	0.74	0.41	0.93	0.13
0.29	0.03	0.32	0.16	0.24	0.35
0.91	0.91	0.24	0.23	0.51	0.23
0.47	0.04	0.17	0.77	0.34	0.08
0.10	0.10	0.73	0.82	0.32	0.23
0.09	0.66	0.10	0.98	0.21	0.66
0.00	0.35	0.38	0.18	0.89	0.02

X is a space of [x1,x2,x3,x4,x5]

Y is a space of [y]



X is a space of [state]

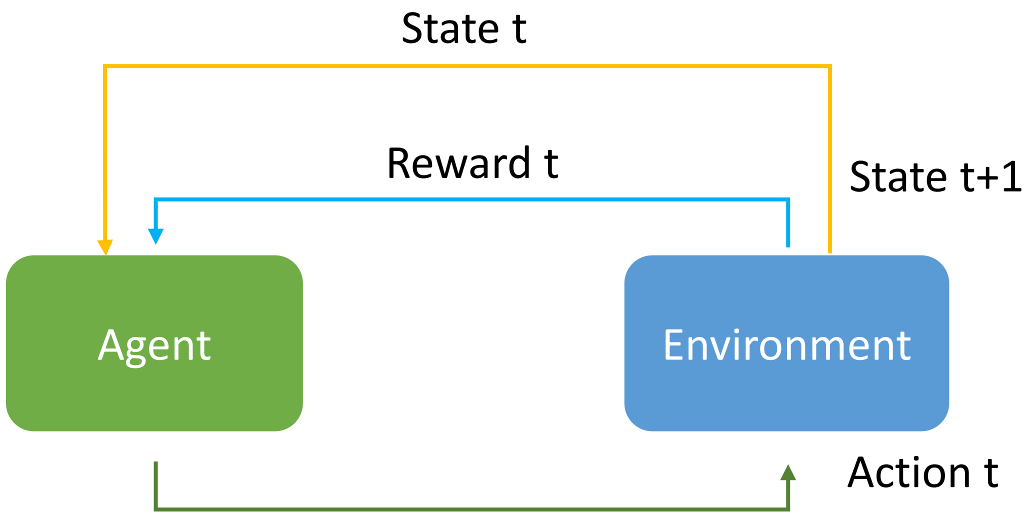
Y is a space of [action]

The differences:

Training data is labeled:
pairs of ([x1,x2,x3,x4,x5], [y])

No training data.
The agent has to interact with environment to generate the data on the run.
And the generated data is unlabeled.

Reinforcement Learning

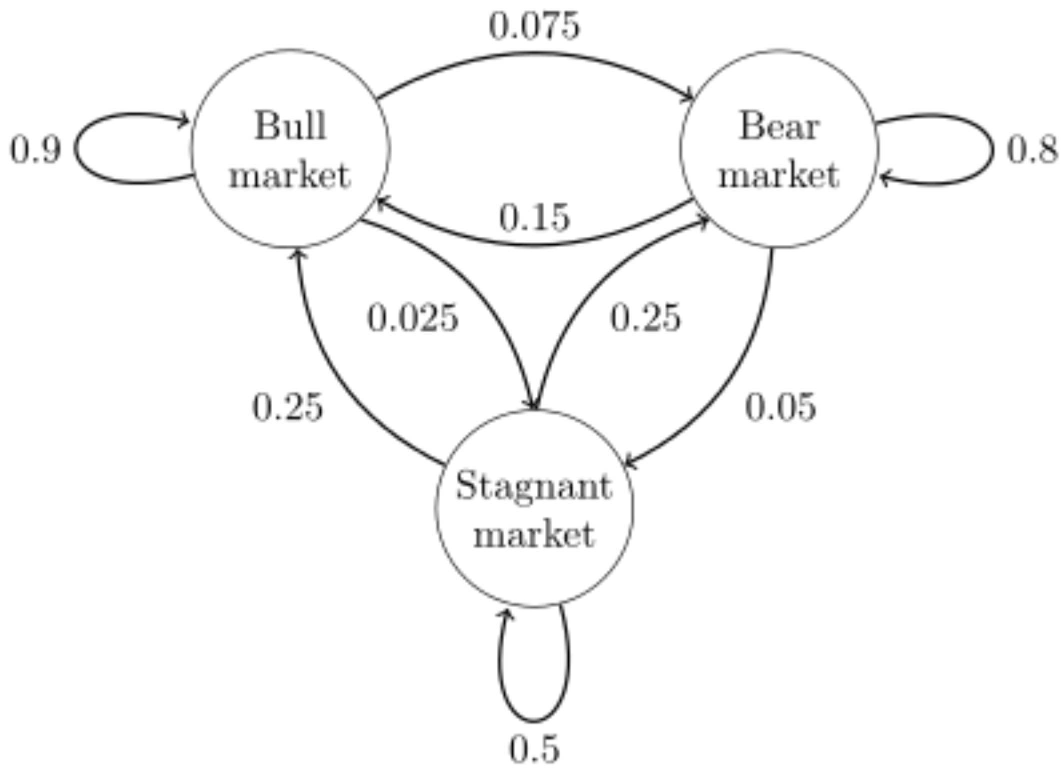


(state, action, reward) trajectory

Markov Process (Markov Chain)

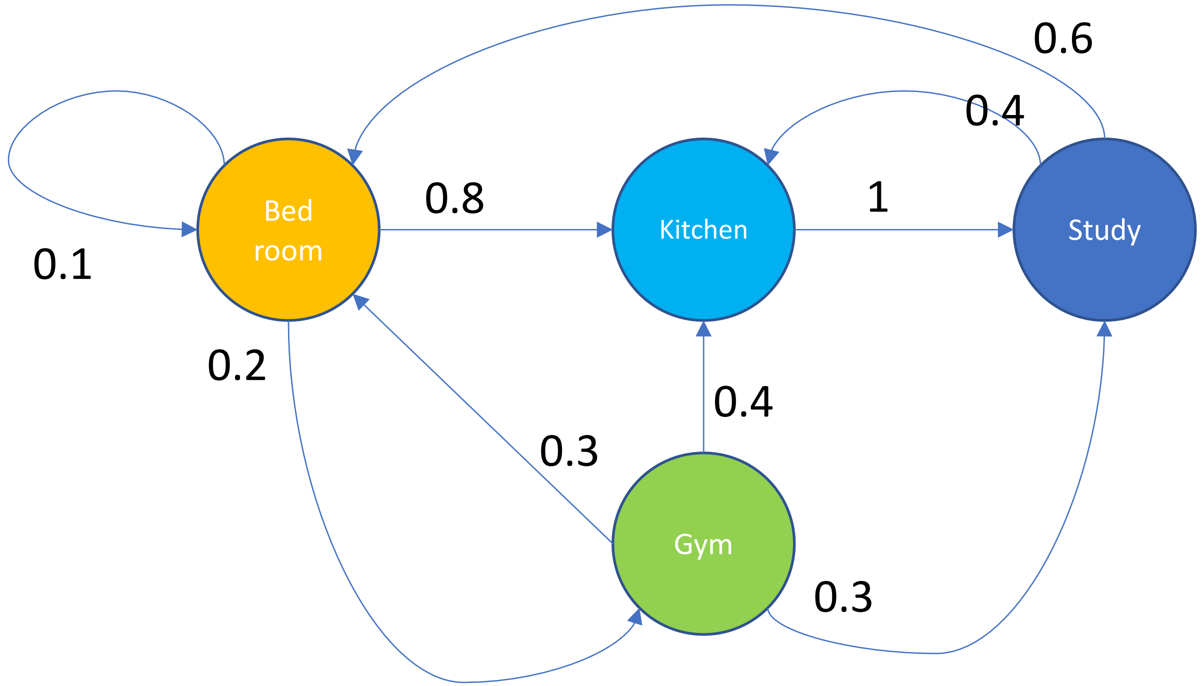
A stochastic process consisting of a sequence of events (state), where the event depends only on the previous event.

Markov Process (Markov Chain)



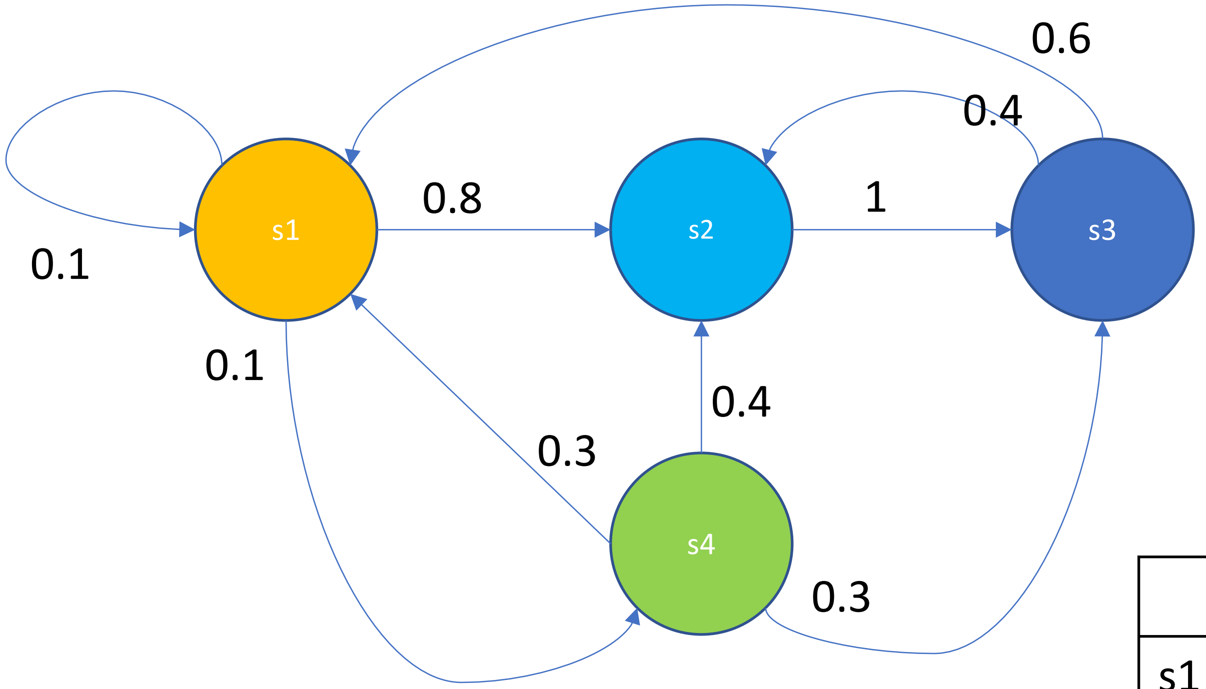
Stock market

Markov Process (Markov Chain)



A day

Markov Process (Markov Chain)



Transition matrix

Next

Current

	s1	s2	s3	s4
s1	0.1	0.8	0	0.1
s2	0	0	1	0
s3	0.6	0.4	0	0
s4	0.3	0.4	0.3	0

Any environment model (simplified)

State transition

$$s_t \xrightarrow{a_t} s_{t+1}$$

It's a probability:

$$p(s'|s,a) = P(S' = s | S = s, A = a)$$

The randomness comes from the environment.

Policy, π

It is a function, $\pi: (a, s) \rightarrow [0,1]$

$$\pi(a|s) = P(A = a|S = s)$$

The output of the function means the probability of taking the action $A = a$ given $S = s$

The agent will take an action, which is guided by the policy function π .

What does it mean? *The agent will draw a sample from the distribution.*

Return

Cumulative future rewards

$$u_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} \cdots$$

The return depends on the current and all future actions ($a_t, a_{t+1}, a_{t+2} \ldots$), current state and all future states ($s_t, s_{t+1}, s_{t+2} \ldots$)

Action can be sampled from the policy function $\pi(a|s)$

State can be sampled from the state transition function $p(s'|s, a)$

Value functions

Action-value function

$$Q_\pi(s_t, a_t) = \mathbb{E}[u_t | S_t = s_t, A_t = a_t]$$

Optimal action-value function

$$Q^*(s_t, a_t) = max_\pi\ Q_\pi(s_t, a_t)$$

State-value function

$$V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A\ )] = \sum_a [\pi(a|s_t) \cdot Q_\pi(s_t, a)] \quad \textit{Action is discrete}$$

$$V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A\ )] = \int [\pi(a|s_t) \cdot Q_\pi(s_t, a)] da \quad \textit{Action is continuous}$$

Value functions

Action-value function

$$Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$$

Given policy π, how good it is for the agent to take the action a_t in the state s_t

State-value function

$$V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A\ )]$$

Given policy π, how good the situation is in the state s_t

How does the agent work?

There are two types:

Policy-based

If we know the policy function π:
The agent can take an action by sampling:

$$a_t \sim \pi(\cdot \mid s_t)$$

Value-based

If we know the optimal action-value function $Q^*(s_t, a_t)$:
The agent can take an action that maximize Q^* :

$$a_t = argmax_a Q^*(s_t, a)$$

Value-based RL

If we know the optimal action-value function $Q^*(s_t, a_t)$:
The agent can take an action that maximize Q^* :

$$\boldsymbol{a_t = argmax_a} Q^*(s_t, a)$$

$Q^*(s_t, a_t)$ can be calculated by looping over all ***possible future paths***, **for simplest cases**.

A practical method is to approximate it **iteratively**.

Action

State

Q	a_1	a_2	a_3	a_4
s_1	$Q(s_1, a_1)$	$Q(s_1, a_2)$	$Q(s_1, a_3)$	$Q(s_1, a_4)$
s_2	$Q(s_2, a_1)$	$Q(s_2, a_2)$	$Q(s_2, a_3)$	$Q(s_2, a_4)$
s_3	$Q(s_3, a_1)$	$Q(s_3, a_2)$	$Q(s_3, a_3)$	$Q(s_3, a_4)$
.	.	.	.	.
.	.	.	.	.

Our objective is to learn the values $Q(s_t, a_t)$ which represents the policy

Q-learning Algorithm

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

Q-learning Algorithm

Action

State

Q	a1	a2	a3	a4
s1	0	0	0	0
s2	0	0	0	0
s3	0	0	0	0
.	.	.	.	.
.	.	.	.	.

s

$Q(s,a_1)$
$Q(s,a_2)$
$Q(s,a_3)$
$Q(s,a_4)$

e-greedy sampling

a

Update Q(s,a) in table

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_* Q(s', *) - Q(s,a)]$$

r

s

Bellman Equation

$$Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma max_{*} Q(s',*)-Q(s,a)]$$

Bellman Equation

$$u_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} \ldots$$

$$u_t = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots)$$

$$u_t = r_t + \gamma u_{t+1}$$

$$E(u_t) = E(r_t + \gamma u_{t+1})$$

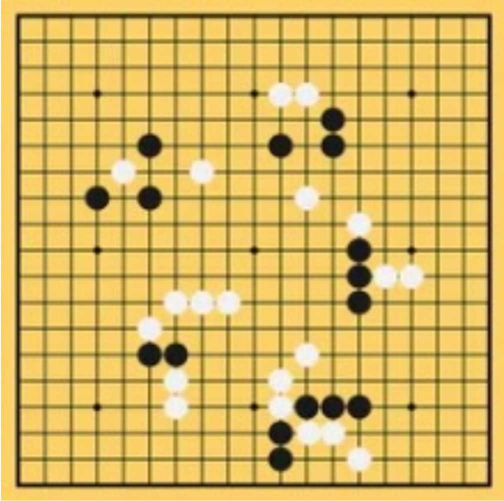
$$E(u_t) = r_t + \gamma E(u_{t+1})$$

$$Q(s_t, a_t) = r_t + \gamma Q(s_{t+1}, a_{t+1})$$

$$Q(s, a) = r + \gamma Q(s', a')$$

In Q-learning, we use a table to store and calculate the optimal Q values.
This is ok for simple cases.

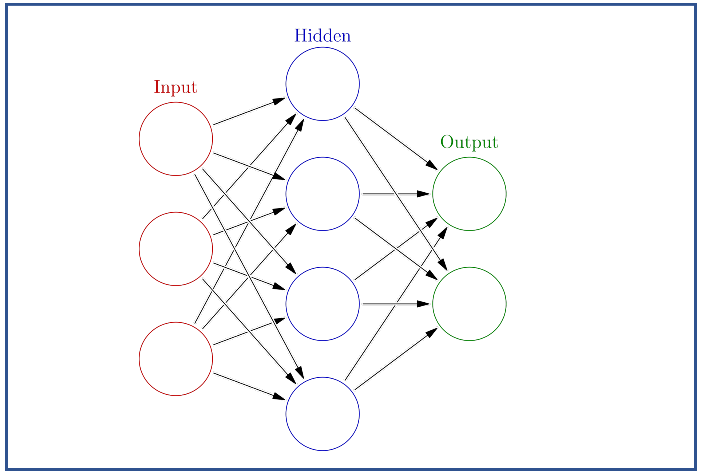
But for more complicated cases, this is not doable because the dimension of the table can the calculation needed are too large.



Instead of using a table, we can use a neural network to approximate the real $Q^*(s_t, a_t)$.

The method is called Deep Q Network (DQN)

Deep Q Network (DQN)



s

$Q(s,a_1,w)$
$Q(s,a_2,w)$
$Q(s,a_3,w)$
$Q(s,a_4,w)$

e-greedy sampling

a

Update w in neural network using backpropagation

$$L=\frac{1}{2}[r+\gamma max_* Q(s',*,w)-Q(s,a,w)]^2$$

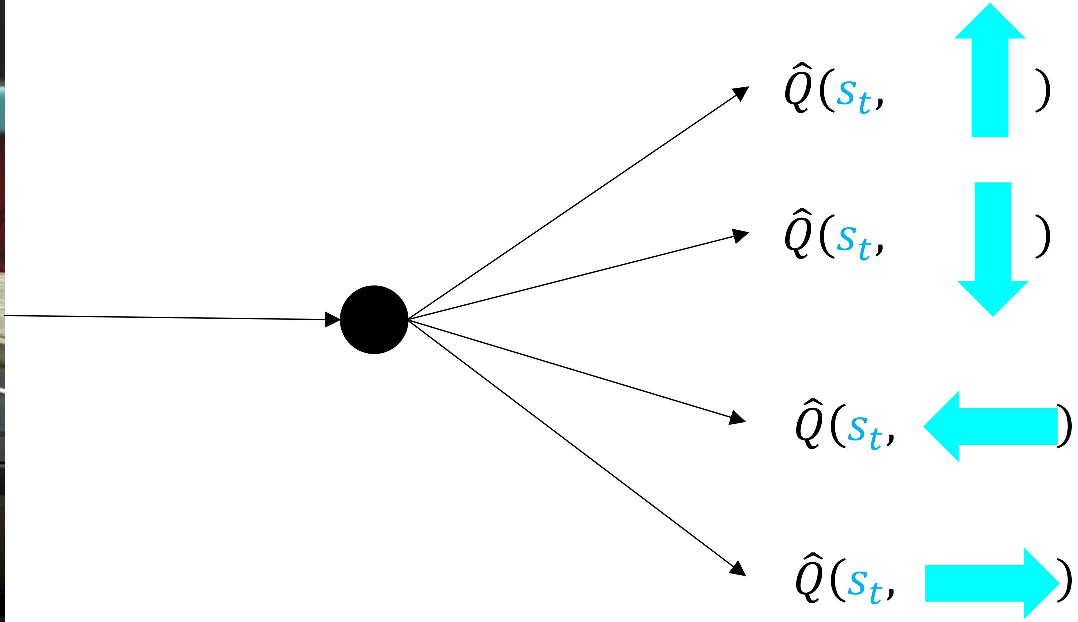
r

s

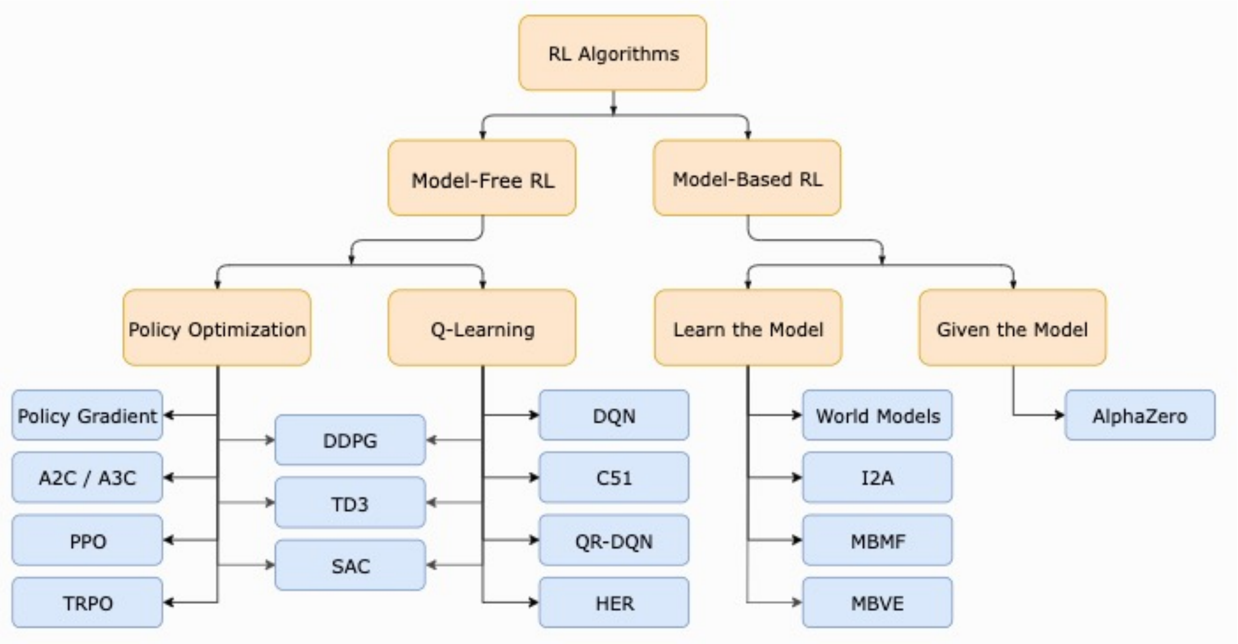
Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602. https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

Deep Q Network (DQN)

s_t



More RL



OpenAI Spinning Up

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

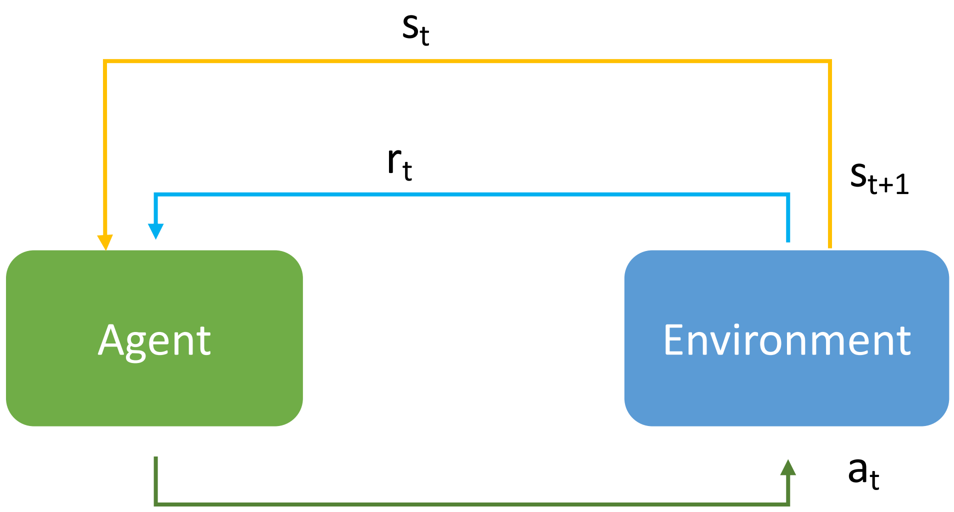
DeepMind x UCL Reinforcement Learning Lecture Series

https://deepmind.com/learning-resources/reinforcement-learning-series-2021

Collection of implementations of RL
https://stable-baselines3.readthedocs.io/en/master/

Tensorflow Agents also has implementations
https://www.tensorflow.org/agents

Create A Reinforcement Learning Workflow



OpenAI Gym
DeepMind-control
Atari
Your own

Exercise: Build your own environment and RL workflow

Tensorflow Agents: Environment: https://www.tensorflow.org/agents/tutorials/2_environments_tutorial

Tensorflow Agents: DQN https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial

Workflow:
https://colab.research.google.com/drive/1wkklY2cj_-qvA7AJvRUnTYWuE_GNJvvt?usp=sharing