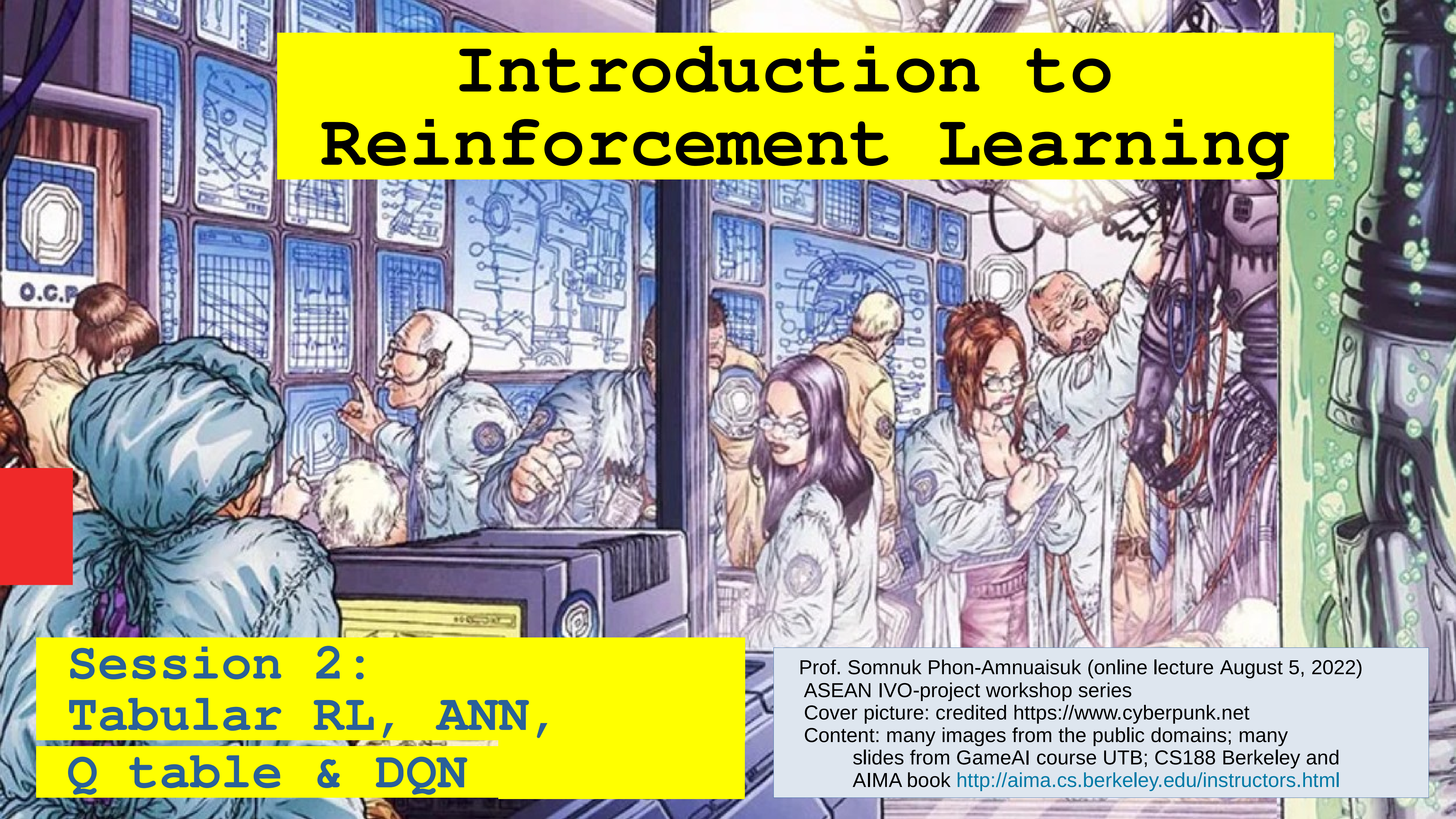
Introduction to Reinforcement Learning



Session 2: Tabular RL, ANN, Q table & DQN

Prof. Somnuk Phon-Amnuaisuk (online lecture August 5, 2022)
ASEAN IVO-project workshop series
Cover picture: credited https://www.cyberpunk.net
Content: many images from the public domains; many slides from GameAI course UTB; CS188 Berkeley and AIMA book http://aima.cs.berkeley.edu/instructors.html

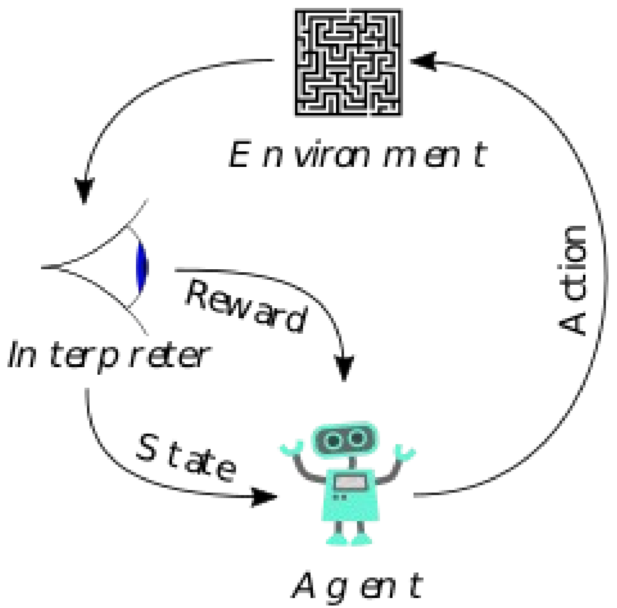

meme
A meme (/' mi: m/ meem), a neologism coined by Richard Dawkins, is "an idea, behavior, or style that spreads from person to person within a culture". A meme acts as a unit for carrying cultural ideas, symbols, or practices that can be transmitted from one mind to another through writing, speech, gestures, rituals, or other imitable phenomena with a mimicked theme.

Outline

- Overview of the previous session
- Q-learning
- Artificial neural network
- Deep Q-learning
- OpenAI Gym

Some Historical Background

- **Reinforcement learning** (**RL**) is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward.
- RL does not assume knowledge of an exact mathematical model of the MDP and it targets large MDPs where exact methods become infeasible.



Search in a Noisy Landscape

- Traditional search is not designed to handle noise from sensors.
- Reinforcement learning could learn to approximate a better representative of the noisy environment.
- A reinforcement learning system identifies the following elements from reward signals: a policy, a value function, and, optionally, a model of the environment.
- Model is optional since RL could be framed as either a model-based or model-free approach.

Markov Decision Process

- A Markov decision process is a 5-tuple (S,A,T(.,.,.),R(.,.,.),gamma) where
- S is a finite set of states
- A is a finite set of actions
- T (s,a,s’) = P(s(t+1) = s’ | s(t) = s, a(t) = a)

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

- R (s,a,s’) is the expected reward after taking action a
- Gamma in [0,1] is a discount factor

Definitions

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are *stationary* (time-independent), $A_t \sim \pi(\cdot|S_t), \forall t > 0$

Slide credit: David Silver

Definitions

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Definition

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

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

Slide credit: David Silver

Bellman Equation - compute V(s)

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \quad \forall s \in \mathcal{S},
\end{aligned}
$$

Q Learning

- Q-learning is a model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances.
- It does not require a model (hence the connotation "model-free") of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations.
- We will discuss more about the notion of ‘Model’ in RL in future RL lectures.

https://en.wikipedia.org/wiki/Q-learning

Reinforcement Learning Model

- A model of the environment: the agent’s representation of the environment

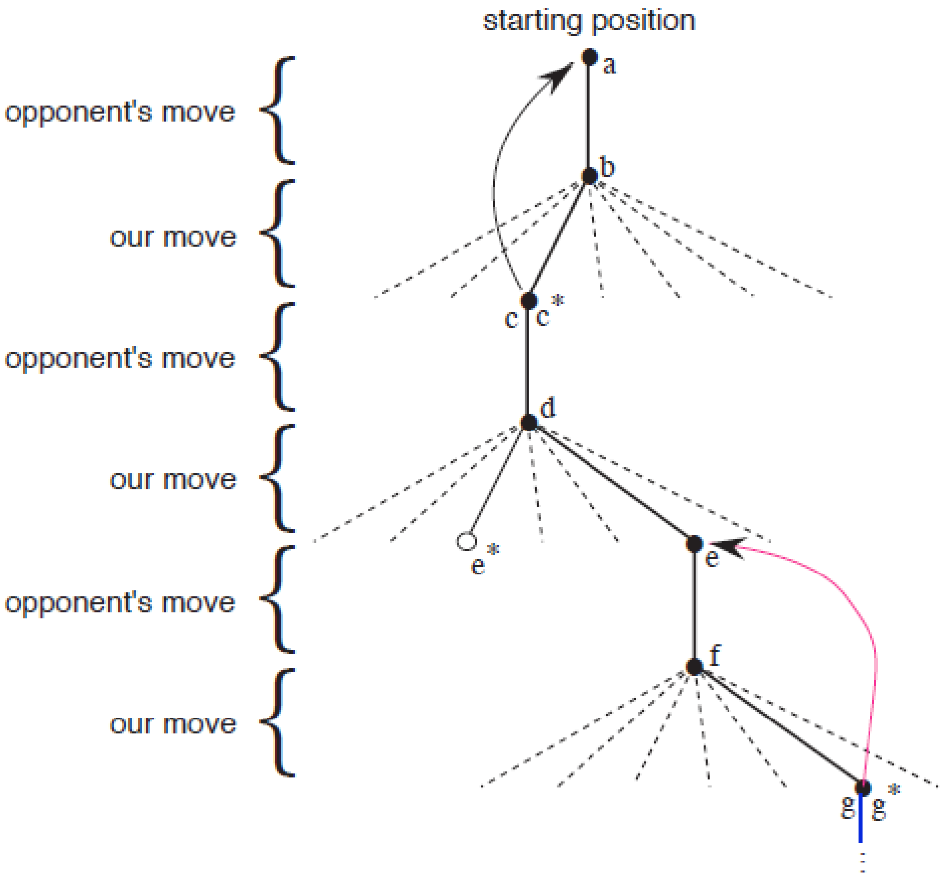
$$p(s',r|s,a) \doteq \Pr\{S_{t+1}=s', R_{t+1}=r \mid S_t=s, A_t=a\}.$$

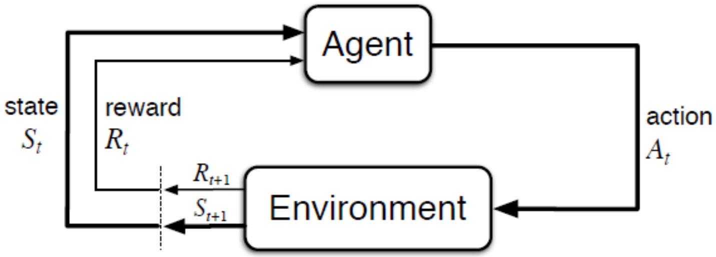
$$r(s,a) \doteq \mathbb{E}[R_{t+1} \mid S_t=s, A_t=a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s',r|s,a),$$

$$p(s'|s,a) \doteq \Pr\{S_{t+1}=s' \mid S_t=s, A_t=a\} = \sum_{r\in\mathcal{R}} p(s',r|s,a),$$

$$r(s,a,s') \doteq \mathbb{E}\left[R_{t+1} \mid S_t=s, A_t=a, S_{t+1}=s'\right] = \frac{\sum_{r\in\mathcal{R}} r\, p(s',r|s,a)}{p(s'|s,a)}.$$

Reinforcement Learning





Initialized

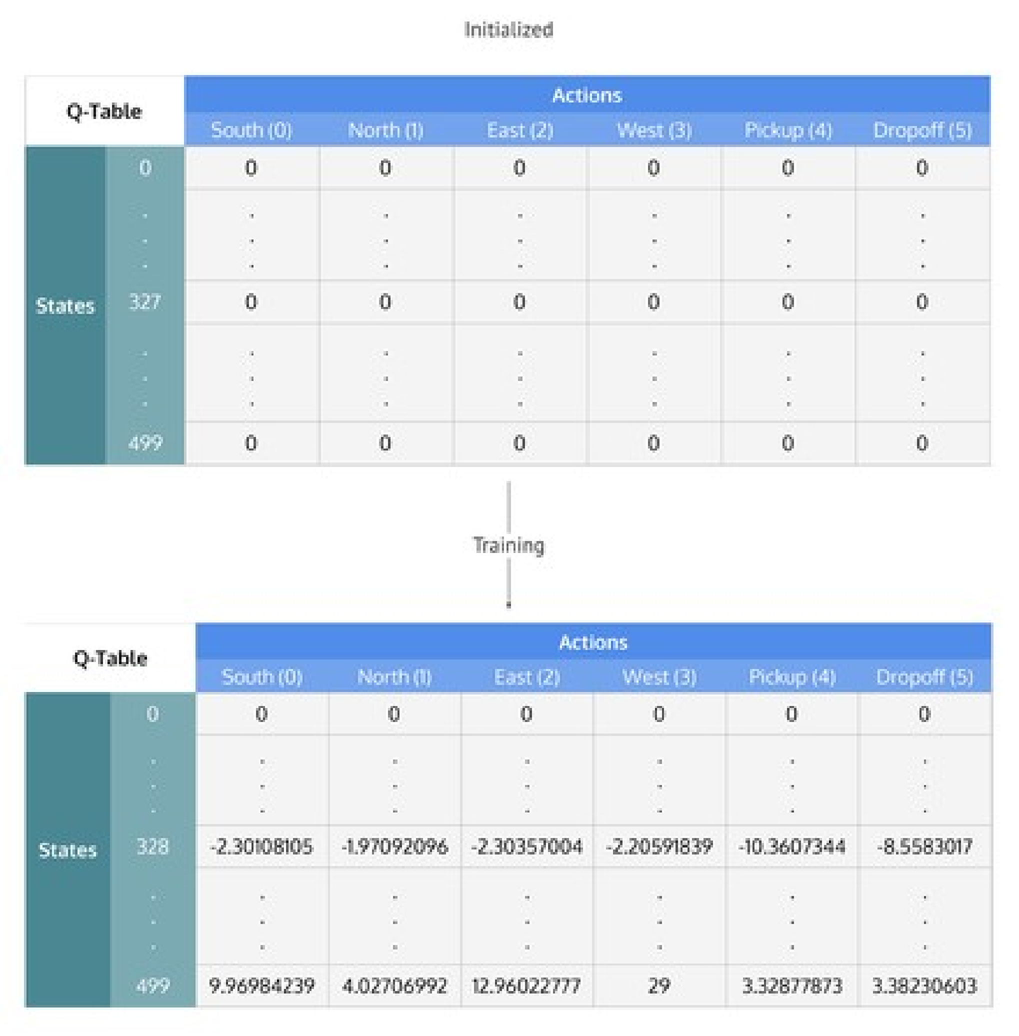
Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Q Learning

- Q-learning was introduced by Chris Watkins in 1989.
- Watkins was addressing “Learning from delayed rewards”, the title of his PhD thesis.
- The standard Q-learning algorithm (using a Q table) applies only to discrete action and state spaces. Discretization of these values leads to inefficient learning.



Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	327	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	0	0	0	0	0	0

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

https://en.wikipedia.org/wiki/Q-learning

Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[sample\right]$$

Recap: Q Learning

Algorithm:

Start with $Q_0(s,a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = R(s,a,s')$$

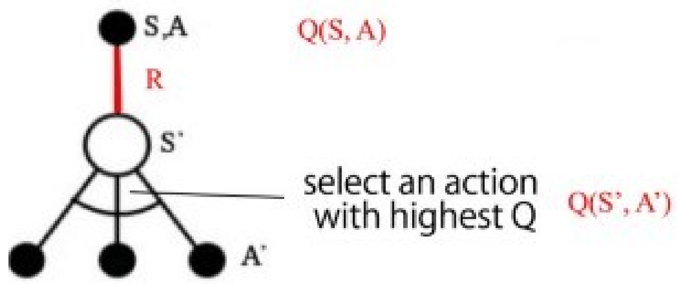
Sample new initial state s'

else:

$$\text{target} = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

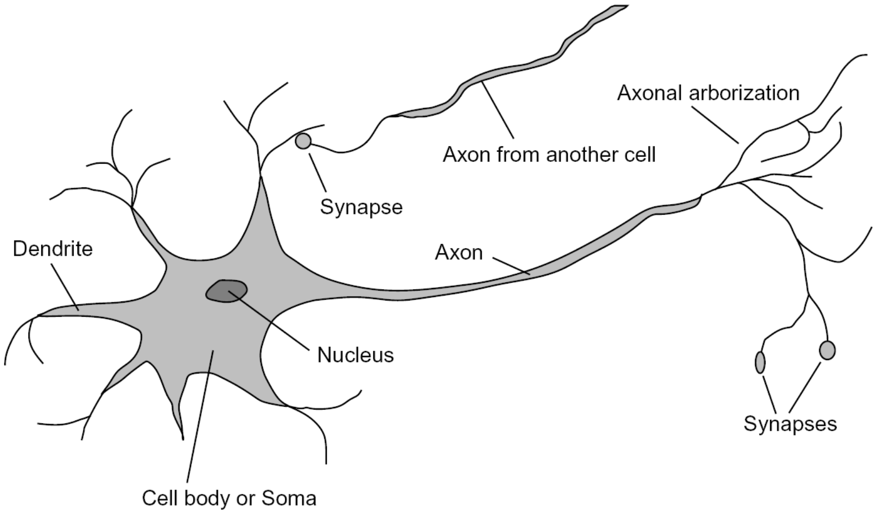
$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\,[\text{target}]$$

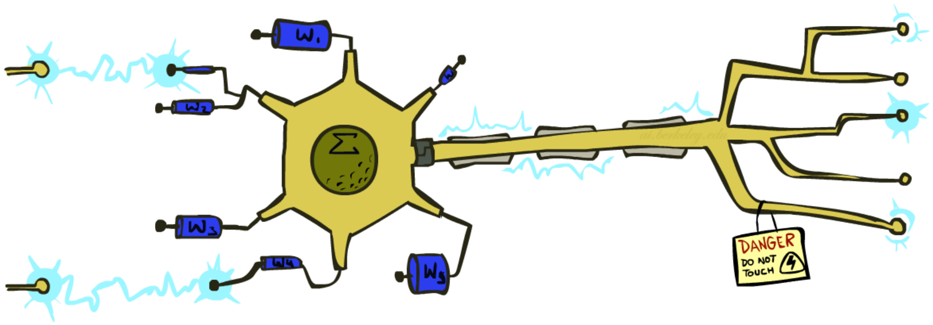
$$s \leftarrow s'$$



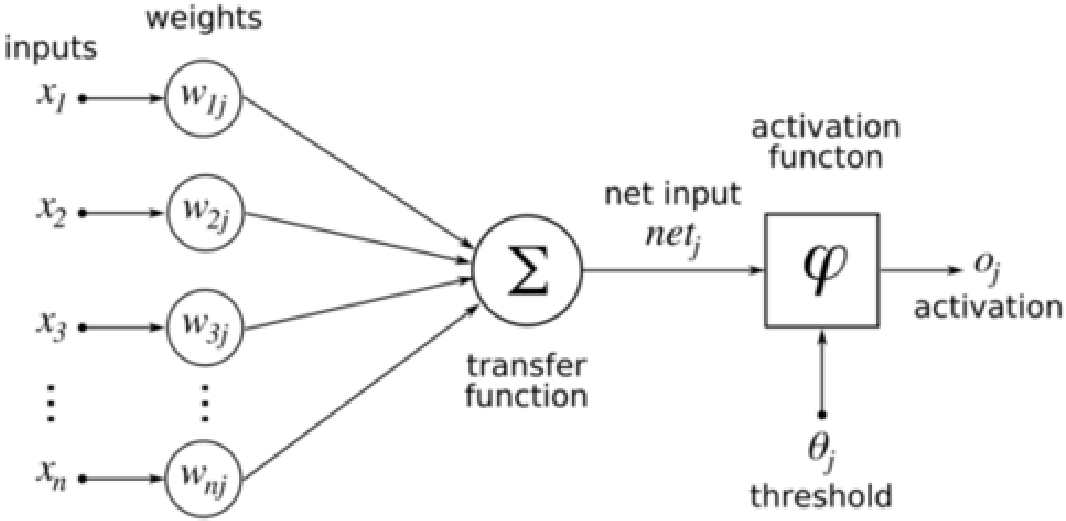
Introduction to Perceptrons and ANNs

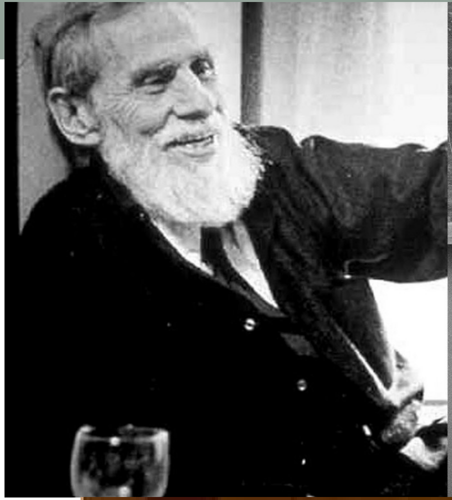
- Very loose inspiration: human neurons

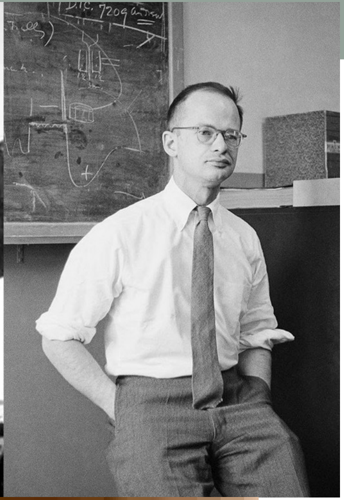




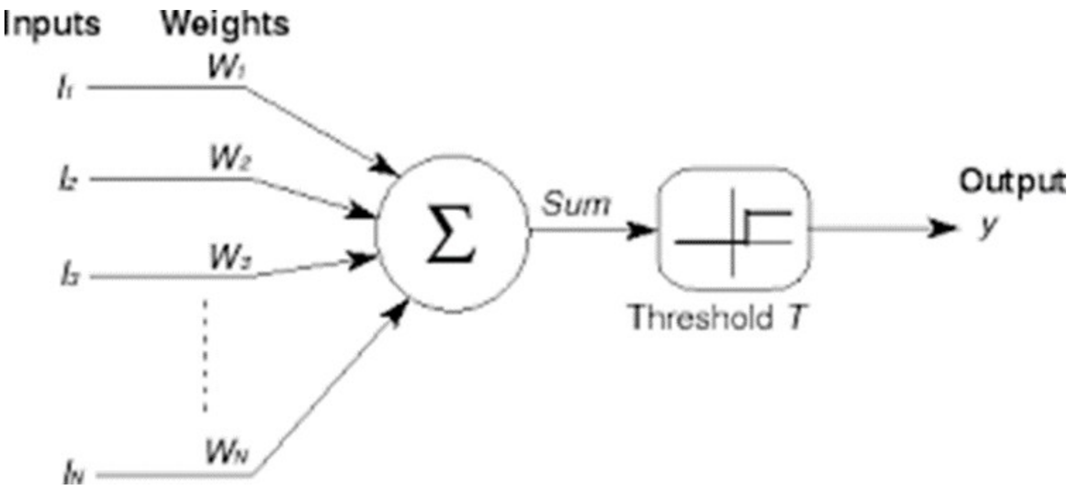
Perceptrons

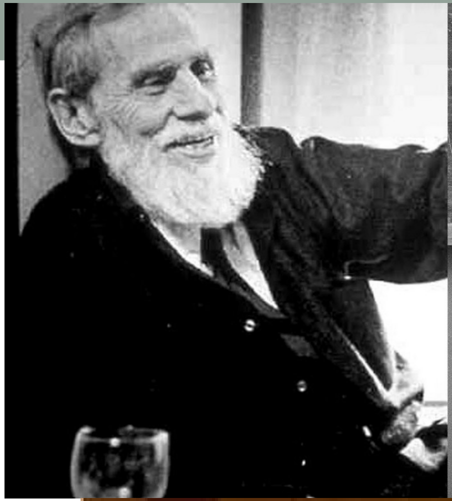


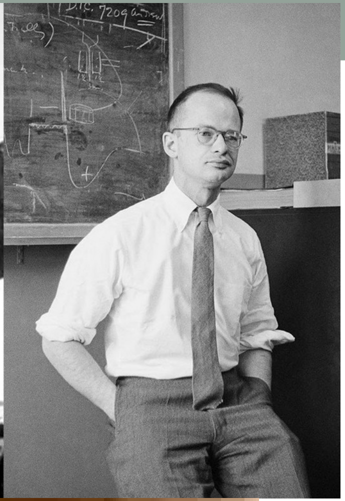




Perceptrons







Perceptrons

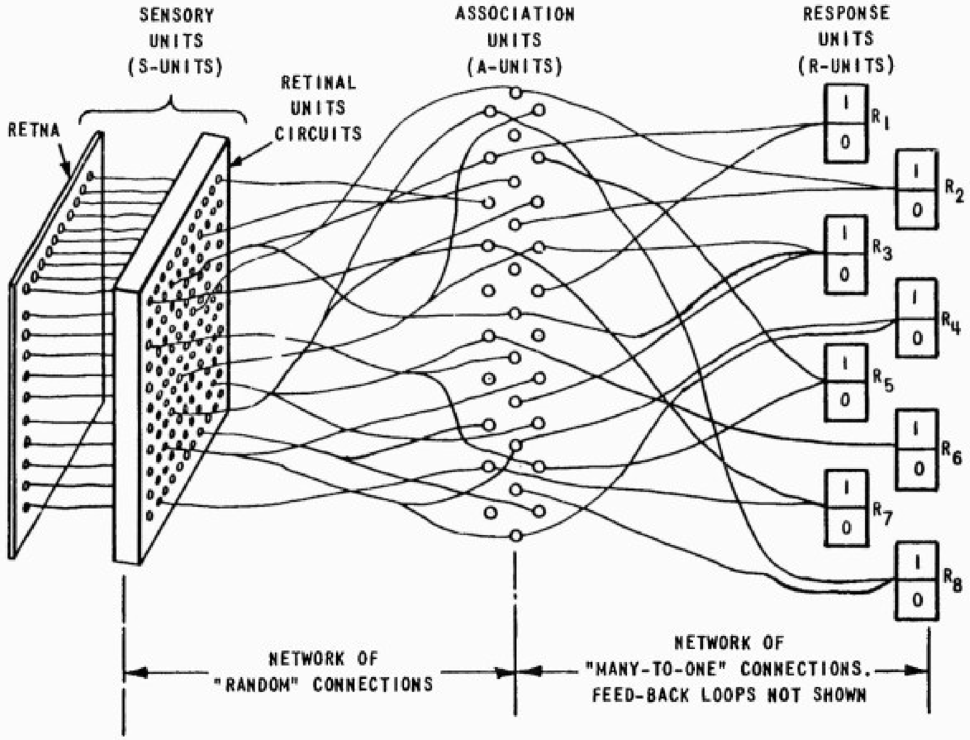


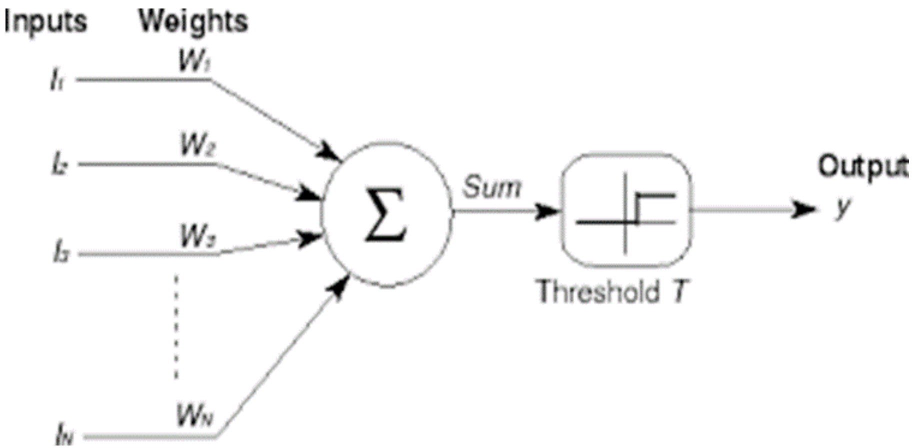
Figure I ORGANIZATION OF THE MARK I PERCEPTRON

How to determine the values of W?

- How to solve for W

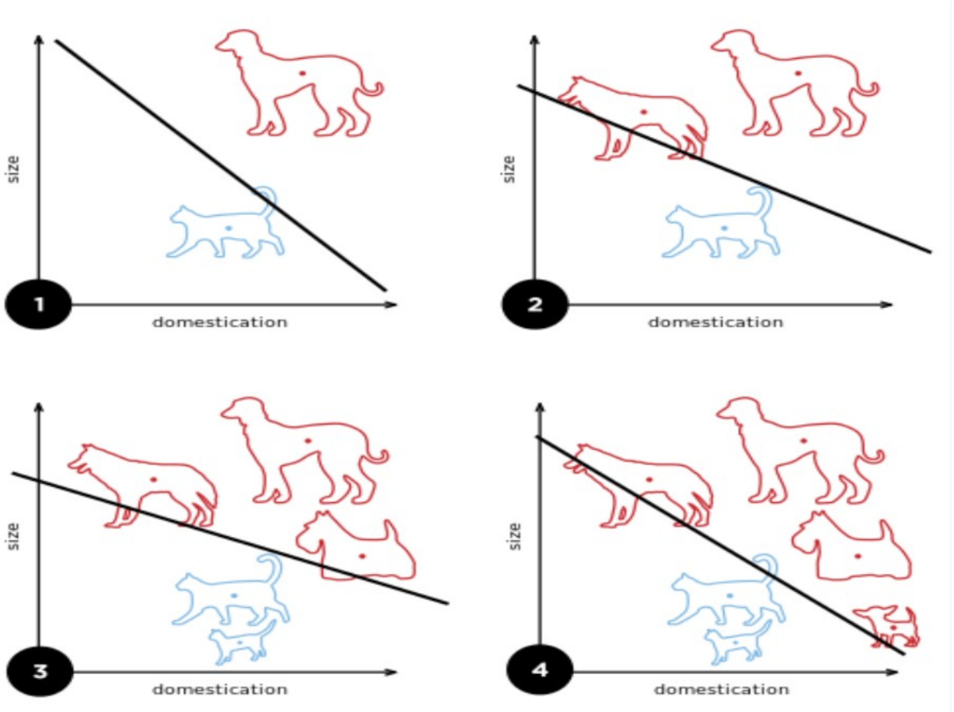
$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{w} \cdot \mathbf{x} \text{ is } \sum_{i=1}^{m} w_i x_i$$



- Randomly
- Analytically
- Optimization algorithm e.g., gradient descent

Rosenblatt’s Perceptron Learning Rule

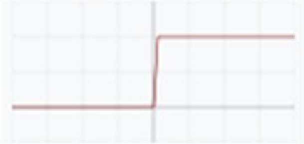
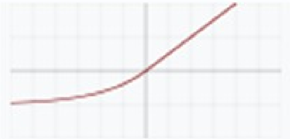


$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}$$

https://en.wikipedia.org/wiki/Perceptron

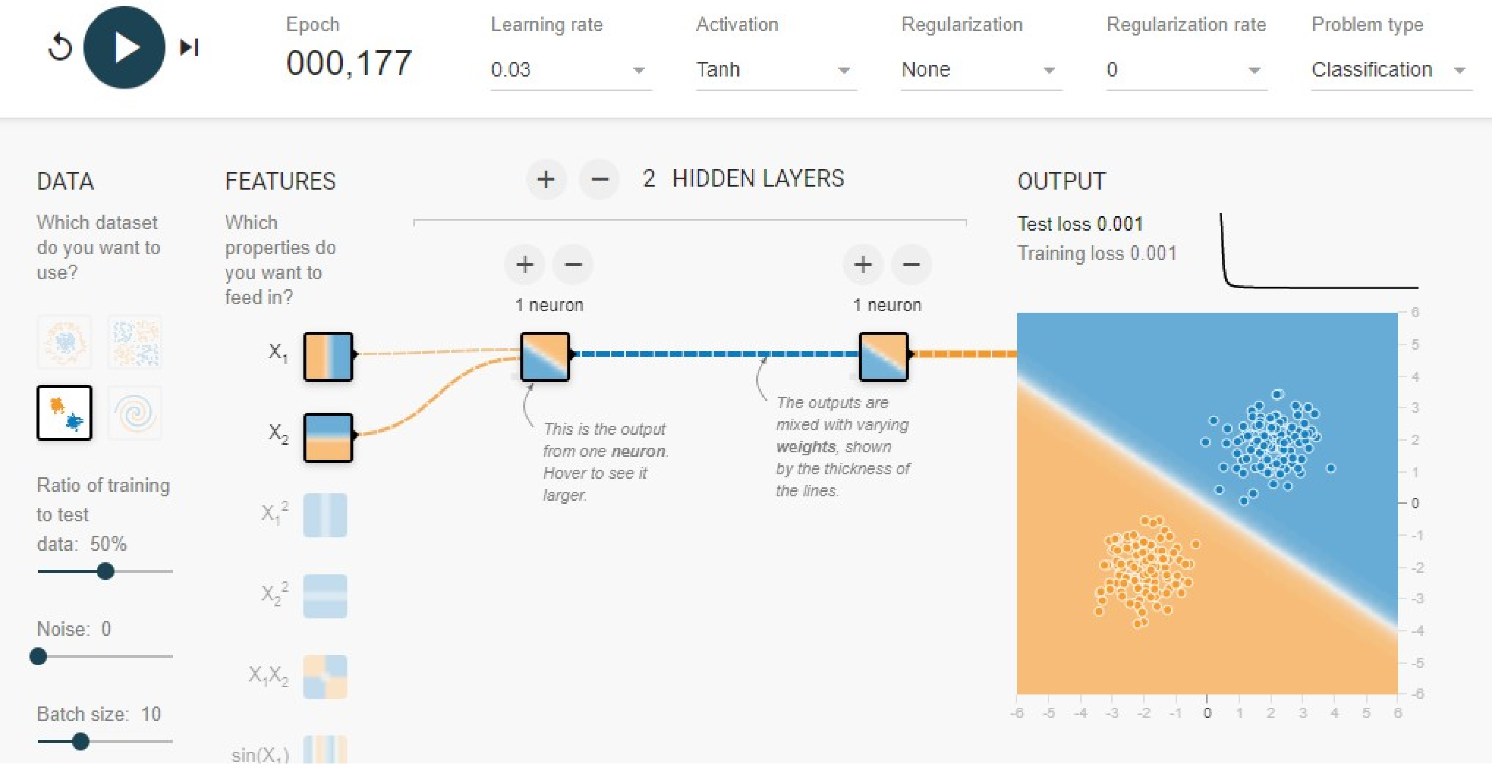
- The perceptron is an artificial neuron using the Heaviside step function as the activation function.
- The perceptron learning rule is an algorithm for learning a classifier function. It was invented in 1958 by Frank Rosenblatt.

Activation Function

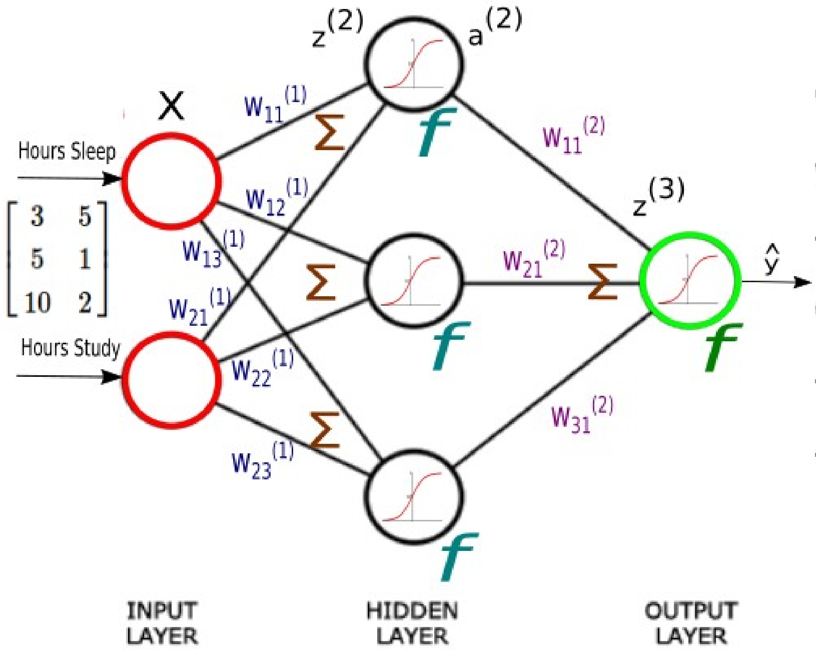
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$ [1]	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$
Rectified linear unit (ReLU)[15]		$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$
Exponential linear unit (ELU)[20]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$

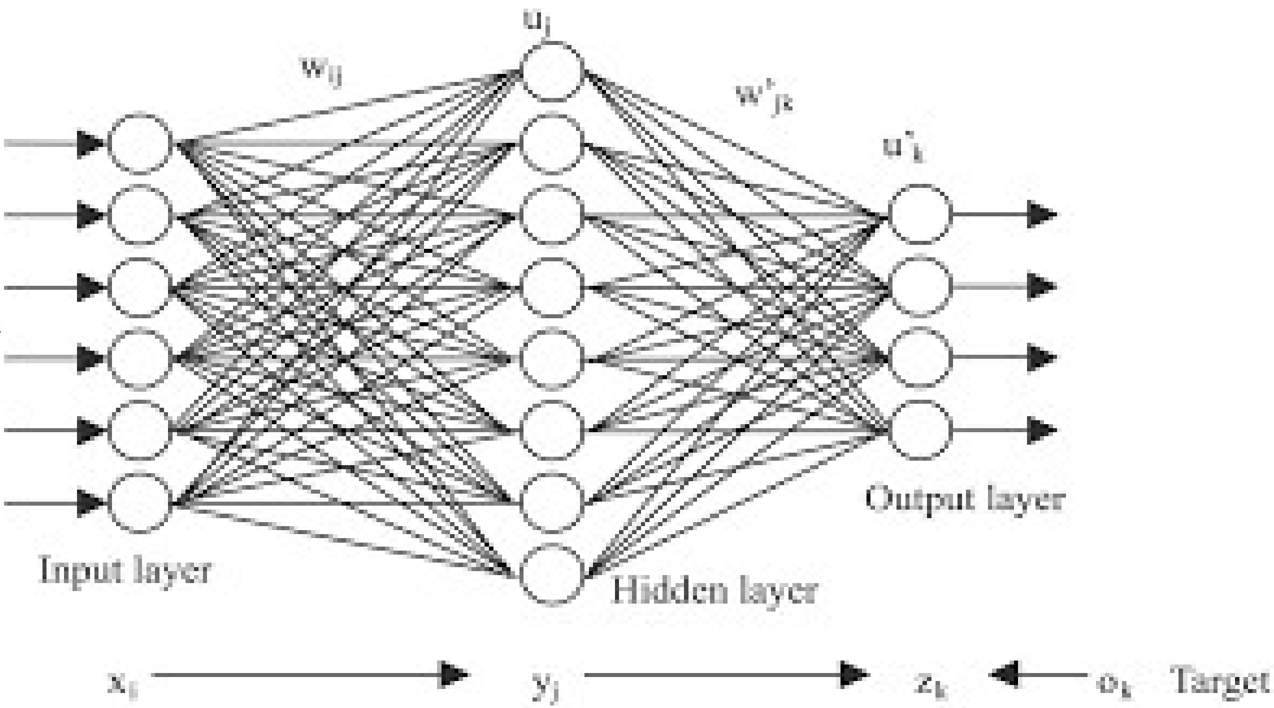
TensorFlow ANN Playground

- https://playground.tensorflow.org



Artificial Neural Network





Backpropagation (Rumelhart 1986)

$$E = \tfrac{1}{2}(t - y)^2 \qquad \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j}\frac{\partial \mathrm{net}_j}{\partial w_{ij}}$$

$$o_j = \varphi(\mathrm{net}_j) = \varphi\left(\sum_{i=1}^{n} w_{ij} o_i\right)$$

$$\varphi(z) = \frac{1}{1 + e^{-z}} \qquad \frac{d\varphi}{dz}(z) = \varphi(z)(1 - \varphi(z))$$

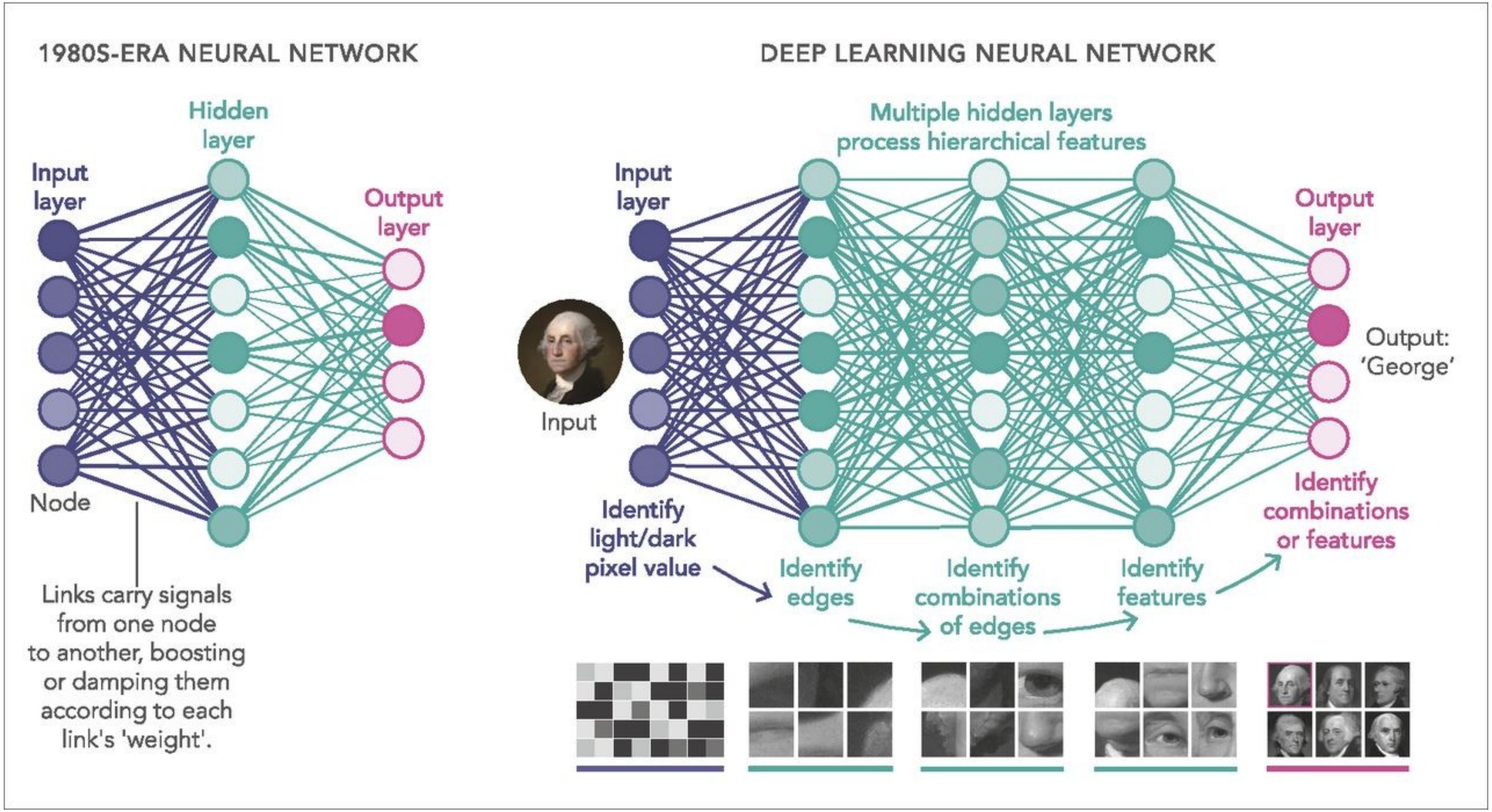
$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}(t - y)^2 = y - t$$

$$\frac{\partial o_j}{\partial \mathrm{net}_j} = \frac{\partial}{\partial \mathrm{net}_j}\varphi(\mathrm{net}_j) = \varphi(\mathrm{net}_j)(1 - \varphi(\mathrm{net}_j))$$

$$\frac{\partial \mathrm{net}_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{i=1}^{n} w_{ij} o_i\right) = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i$$

$$\frac{\partial E}{\partial w_{ij}} = (o_j - t)\, o_j (1 - o_j)\, o_i$$

Deep Learning

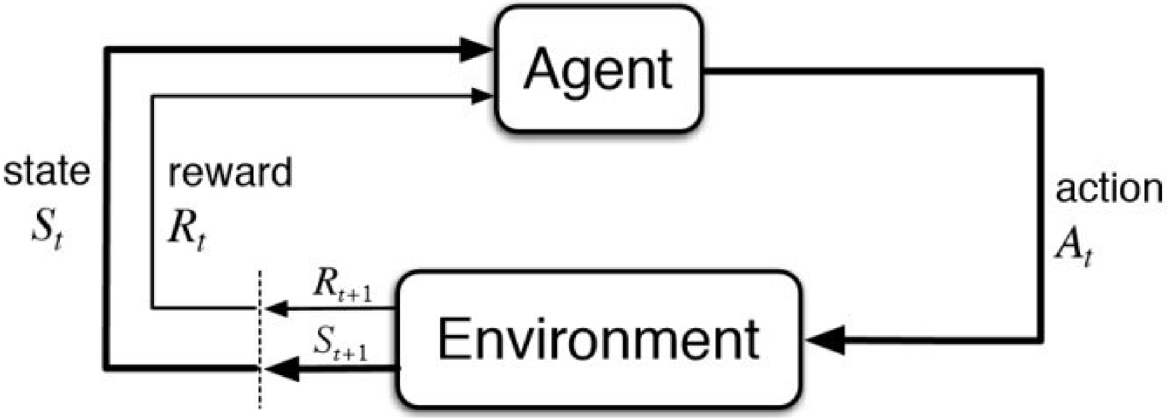


credit: Lucy Reading-Ikkanda (artist). https://www.pnas.org/doi/10.1073/pnas.1821594116

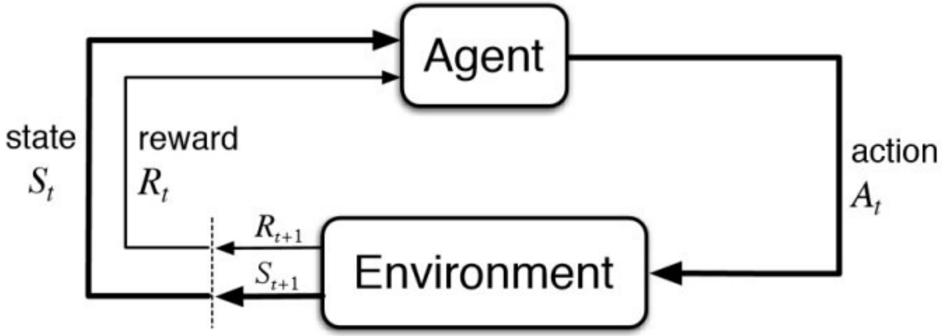
Reinforcement Learning

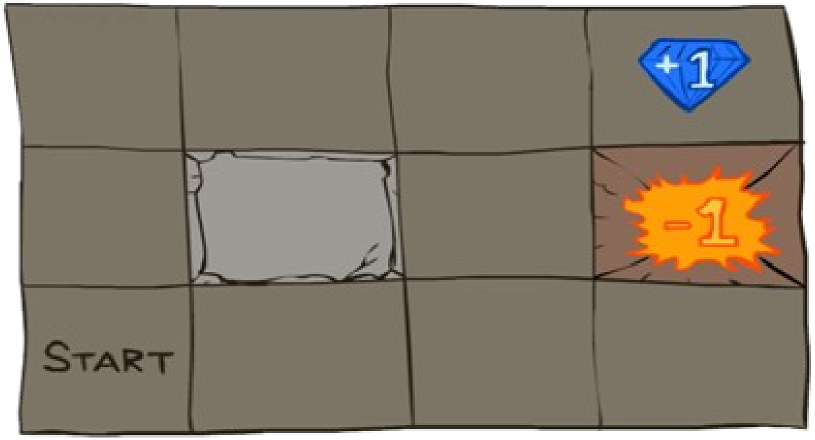
- Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

  (Wikipedia)

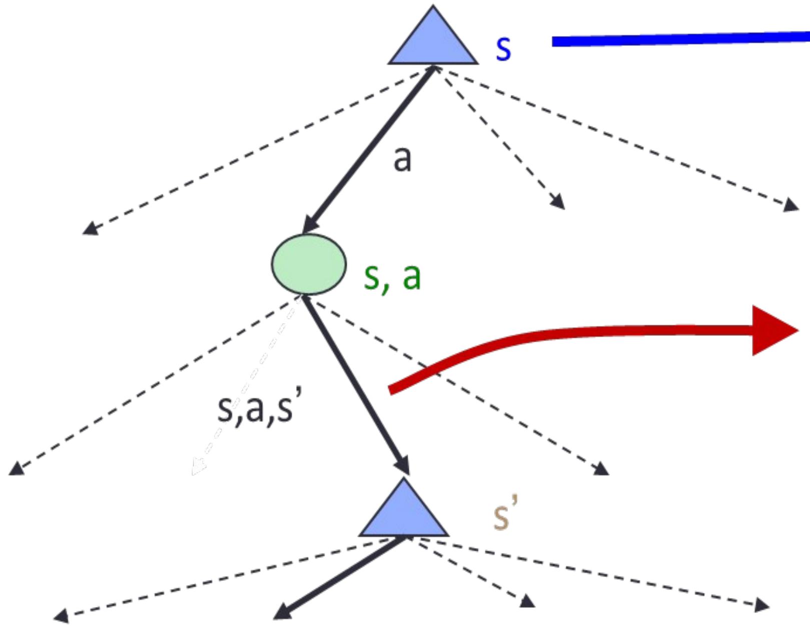


Reinforcement Learning





Reinforcement Learning

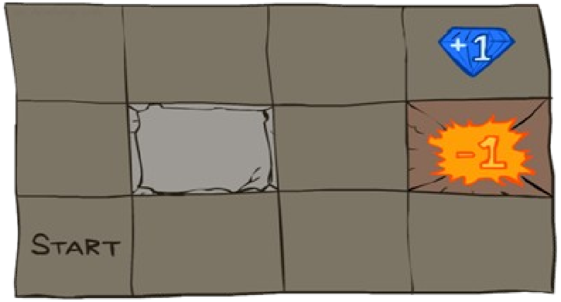


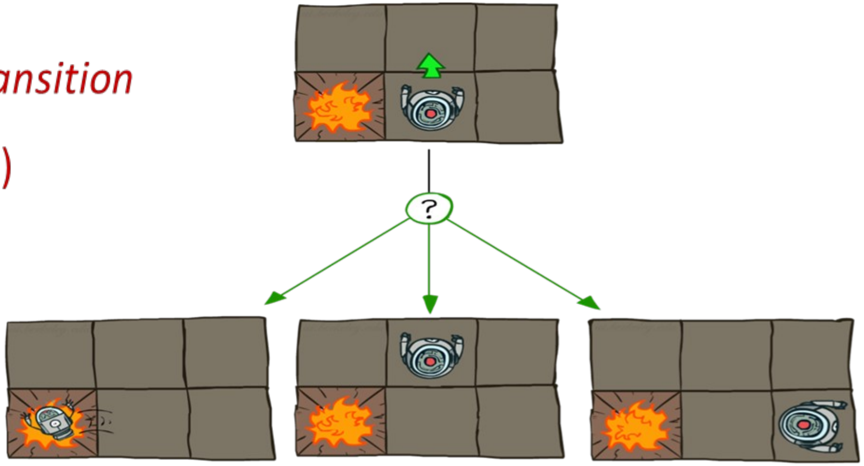
s is a *state*

(s,a,s') called a *transition*

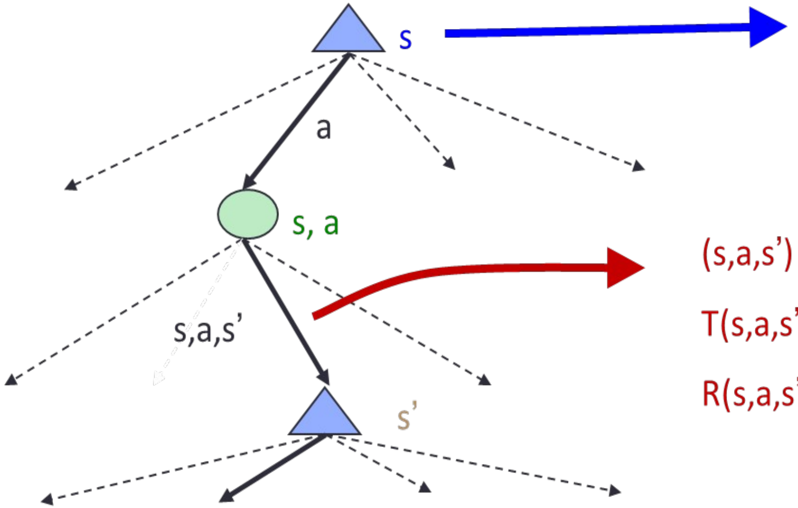
T(s,a,s') = P(s'|s,a)

R(s,a,s')





Reinforcement Learning

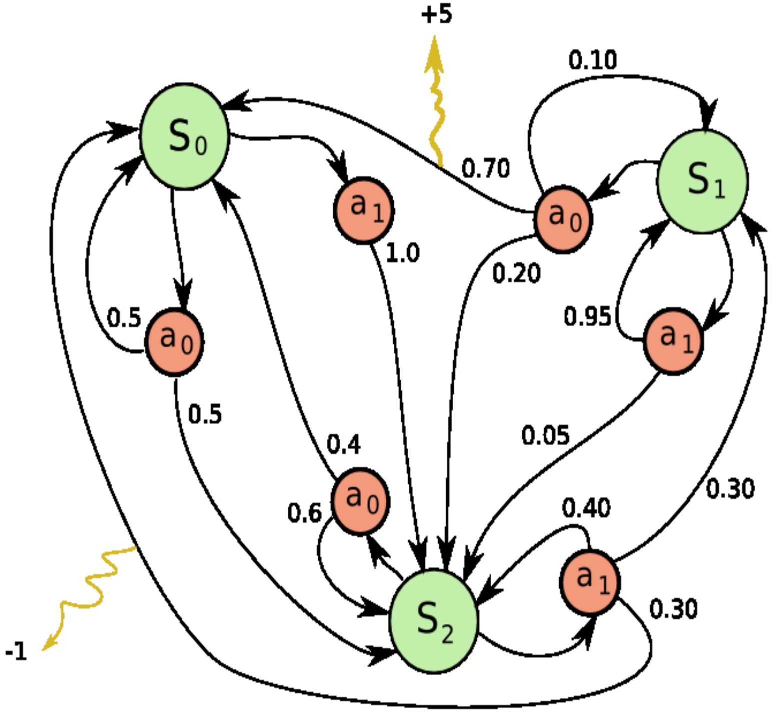


s is a *state*

(s,a,s') called a *transition*

T(s,a,s') = P(s'|s,a)

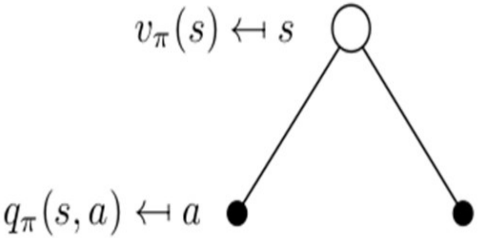
R(s,a,s')



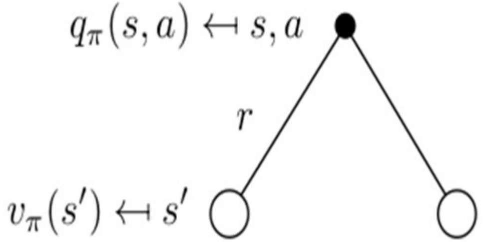
$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]$$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

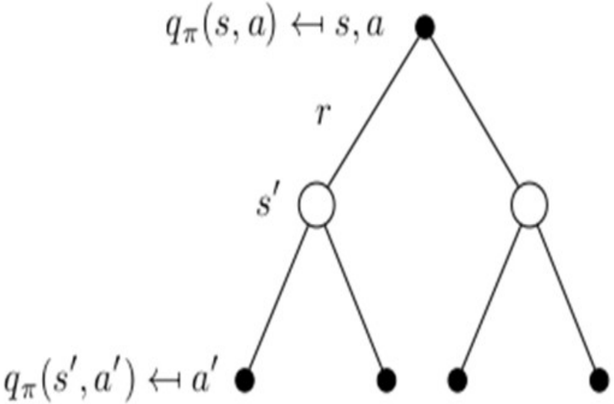
Reinforcement Learning



$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) q_\pi(s,a)$$



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a'\in\mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

Reinforcement Learning

- Below are the Bellman equations, and they characterize optimal values.
- V*(s) = expected utility starting in s and acting optimally
- Q*(s,a) = expected utility starting out having taken action a from state s and (thereafter) acting optimally

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

Tabular RL

Tabular RL approach is a common choic[e] of implementation in various RL learning techniques:

- Temporal different learning (TD)
- Q-learning, and
- SARSA

Actions / States	A_1	A_2	$\dots$	A_n
S_1	$q_{1,1}$	$q_{1,2}$	$\dots$	$q_{1,n}$
S_2	$q_{2,1}$	$q_{2,2}$	$\dots$	$q_{2,n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
S_m	$q_{m,1}$	$q_{m,2}$	$\dots$	$q_{m,n}$

Q Learning

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \bigg)}^{\text{temporal difference}}$$

In 2014 Google DeepMind patented an application of Q-learning to deep learning, titled "deep reinforcement learning" or "deep Q-learning" that can play Atari 2600 games at expert human levels.

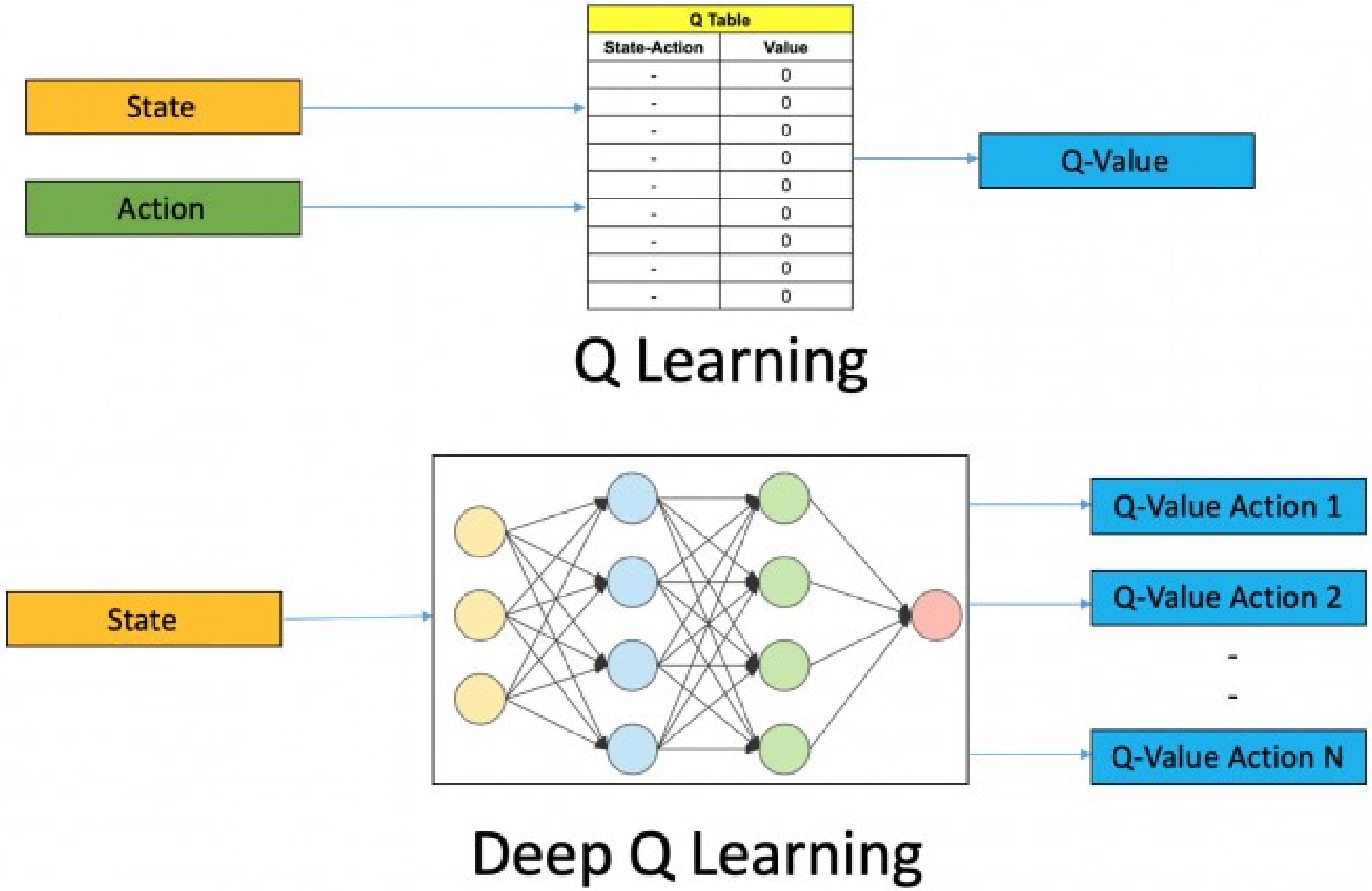
Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Q and Deep Q Learning



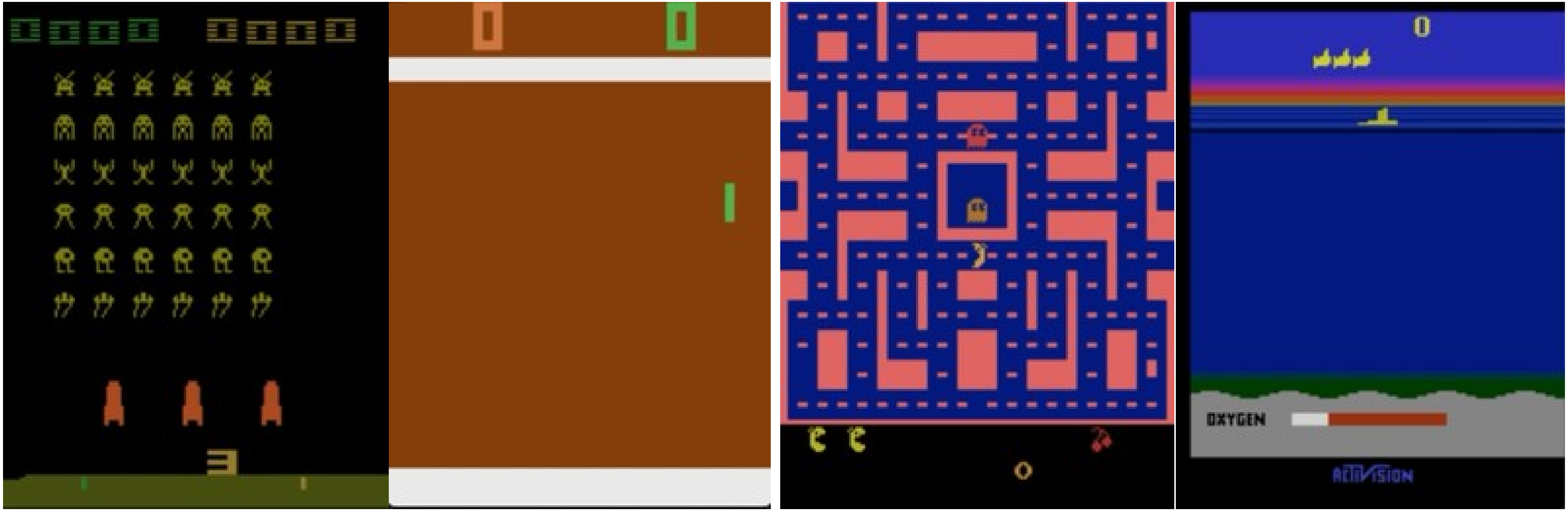
Policy Gradient Learning

- REINFORCE - Tesuro
- Actor-Critic
- Asynchronous Advantage Actor-Critic (A3C)

Reinforcement Learning

Algorithm	Description	Model	Policy	Action Space	State Space	Operator
Monte Carlo	Every visit to Monte Carlo	Model-Free	Off-policy	Discrete	Discrete	Sample-means
Q-learning	State–action–reward–state	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA	State–action–reward–state–action	Model-Free	On-policy	Discrete	Discrete	Q-value
Q-learning - Lambda	State–action–reward–state with eligibility traces	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA - Lambda	State–action–reward–state–action with eligibility traces	Model-Free	On-policy	Discrete	Discrete	Q-value
DQN	Deep Q Network	Model-Free	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Model-Free	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Advantage Actor-Critic Algorithm	Model-Free	On-policy	Continuous	Continuous	Advantage
NAF	Q-Learning with Normalized Advantage Functions	Model-Free	Off-policy	Continuous	Continuous	Advantage
TRPO	Trust Region Policy Optimization	Model-Free	On-policy	Continuous	Continuous	Advantage
PPO	Proximal Policy Optimization	Model-Free	On-policy	Continuous	Continuous	Advantage
TD3	Twin Delayed Deep Deterministic Policy Gradient	Model-Free	Off-policy	Continuous	Continuous	Q-value
SAC	Soft Actor-Critic	Model-Free	Off-policy	Continuous	Continuous	Advantage

Deep Q Learning (around 2013-2015)



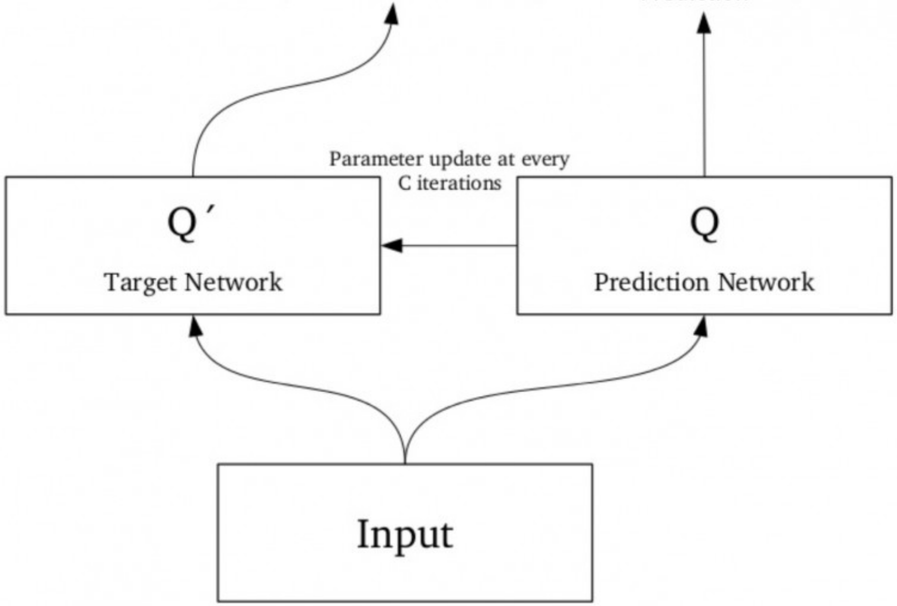
Deep Q Learning

- The DeepMind system used a deep convolutional neural network, with layers of tiled convolutional filters to mimic the effects of receptive fields.
- **Reinforcement learning is unstable** or divergent when a nonlinear function approximator such as a neural network is used to represent Q. This instability comes from the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and the data distribution, and the correlations between Q and the target values.
- The technique used **experience replay**, a biologically inspired mechanism that uses a random sample of prior actions instead of the most recent action to proceed. This removes correlations in the observation sequence and smooths changes in the data distribution. Iterative updates adjust Q towards target values that are only periodically updated, further reducing correlations with the target.

Deep Q Learning

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Bigg)}^{\text{temporal difference}}$$

$$\left[\left(\underbrace{\boxed{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}}_{\text{Target}} - \underbrace{\boxed{Q(s, a; \theta_i)}}_{\text{Prediction}} \right)^2 \right]$$



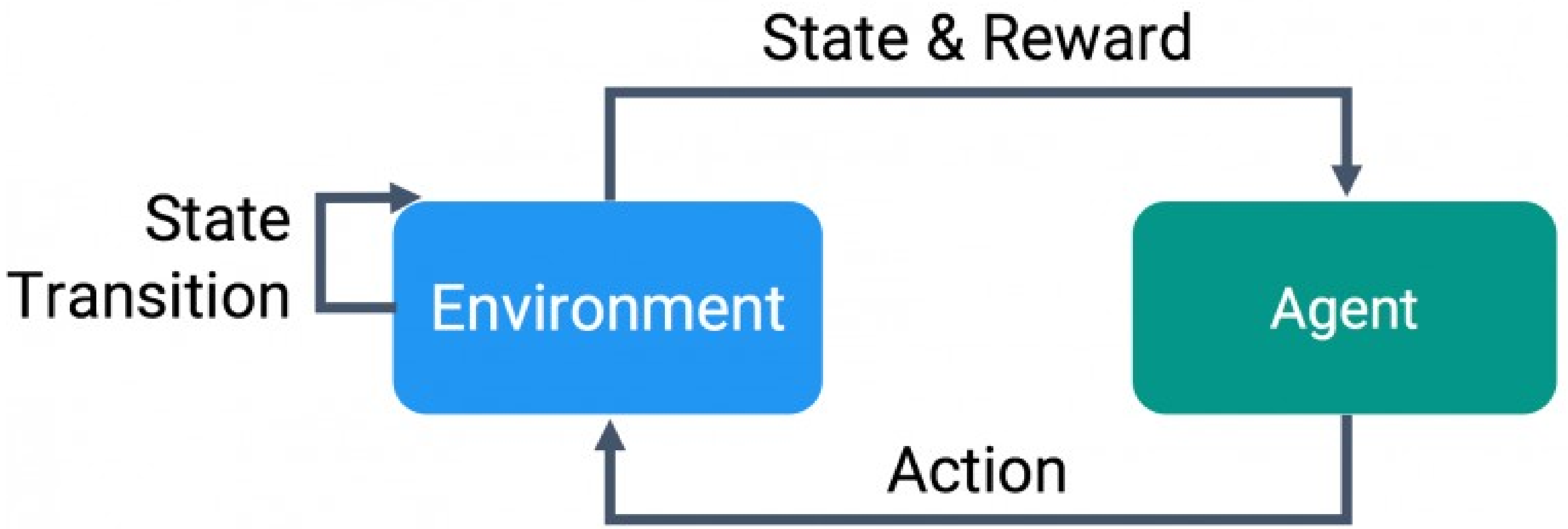
1. Select an action from possible Q-values actions, select using the epsilon-greedy policy.
2. Perform this action **a** in a state **s** and move to a new state **s’** to receive a reward **r**.
3. Record this transition in our replay buffer as <s,a,r,s’>
4. After C iterations, sample data randomly from the replay-buffer and train the ANN using fix target.
5. Update the target Q network
6. Repeat

https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

Deep Q Learning (breakout)

- The Deepmind paper trained for "a total of 50 million frames (that is, around 38 days of game experience in total)". Note that this paper is published in 2015.
- A Q-Learning Agent learns to perform its task such that the recommended action maximizes the potential future rewards.
- This method is considered an "Off-Policy" method, meaning its Q values are updated assuming that the best action was chosen, even if the best action was not chosen.

Reinforcement Learning in Gym



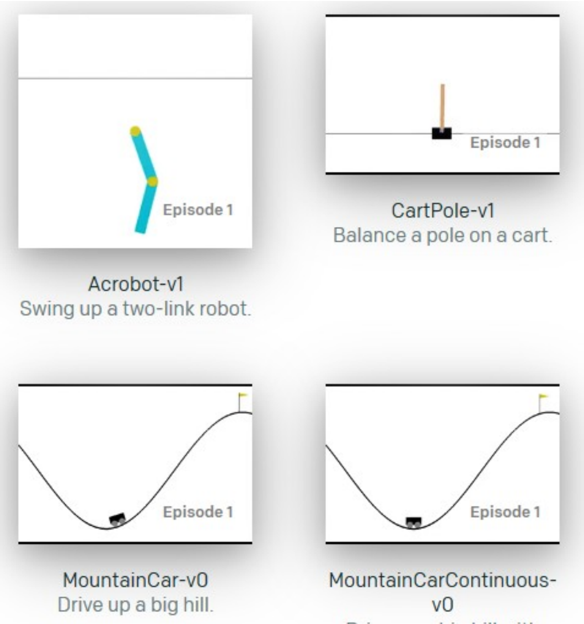
Gym

- Gym is a toolkit for developing and comparing reinforcement learning algorithms.
- Gym makes no assumptions about the structure of your agent, and is compatible with any numerical computation library, such as TensorFlow or Theano.
- The gym library is a collection of test problems — environments — that you can use to work out your reinforcement learning algorithms

Available Environments

Gym comes with a diverse suite of environments that range from easy to difficult and involve many different kinds of data. View the full list of environments to get the birds-eye view.

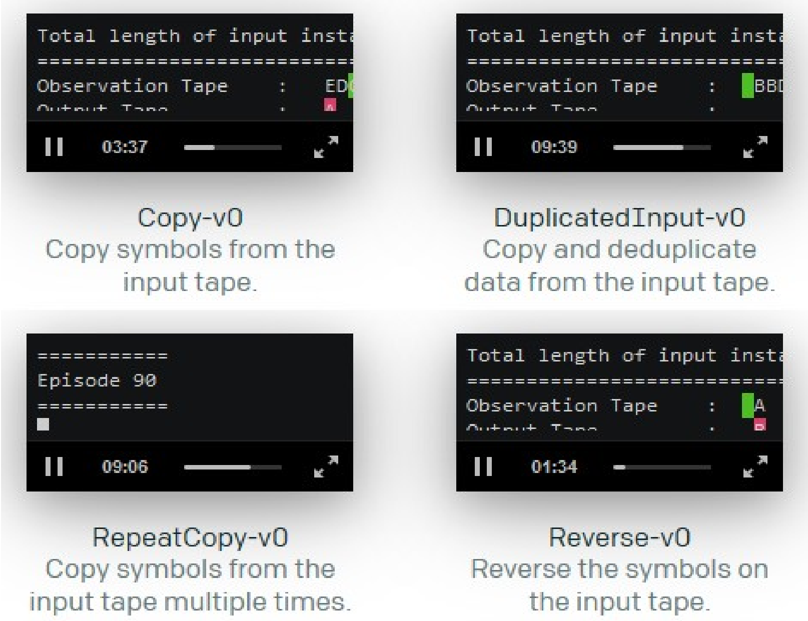
- **Classic control and toy text**
- Algorithmic
- Atari
- 2D and 3D robots



Available Environments

Gym comes with a diverse suite of environments that range from easy to difficult and involve many different kinds of data. View the full list of environments to get the birds-eye view.

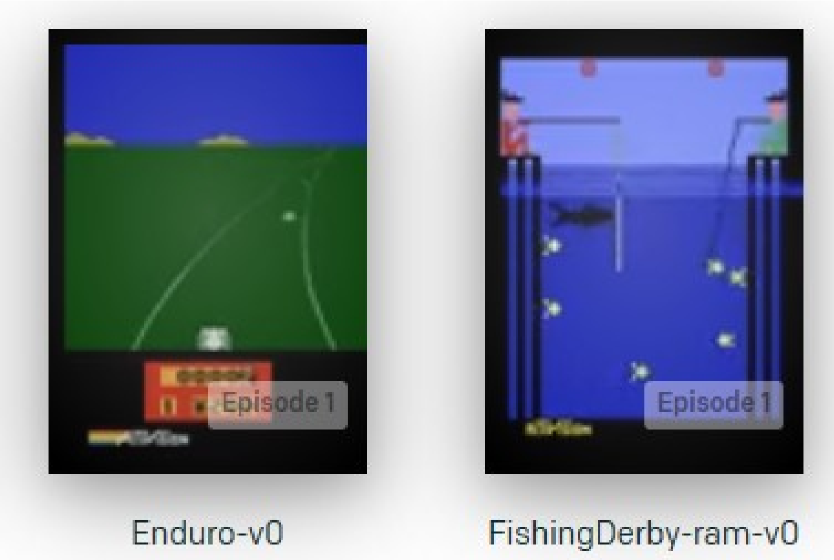
- Classic control and toy text
- **Algorithmic**
- Atari
- 2D and 3D robots



Available Environments

Gym comes with a diverse suite of environments that range from easy to difficult and involve many different kinds of data. View the full list of environments to get the birds-eye view.

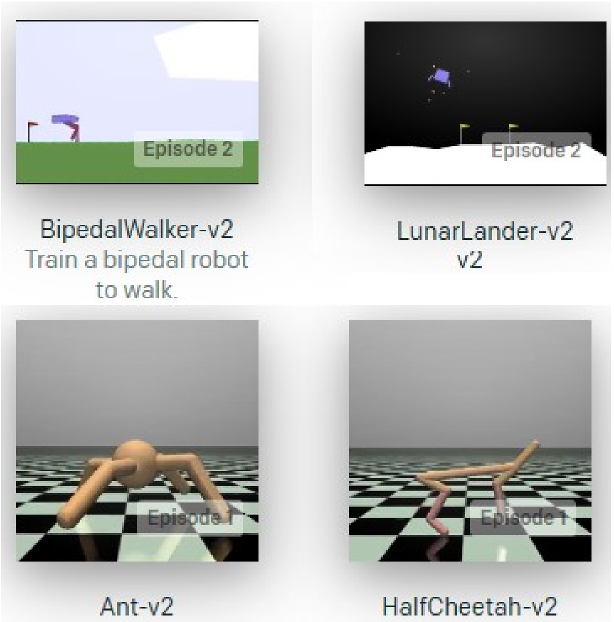
- Classic control and toy text
- Algorithmic
- **Atari**
- 2D and 3D robots



Available Environments

Gym comes with a diverse suite of environments that range from easy to difficult and involve many different kinds of data. View the full list of environments to get the birds-eye view.

- Classic control and toy text
- Algorithmic
- Atari
- **2D and 3D robots**



Gym

The main OpenAI Gym class. It encapsulates an environment with arbitrary behind-the-scenes dynamics. An environment can be partially or fully observed.

The main API methods that users of this class need to know are:

- **step**
- **reset**
- **render**
- **close**
- **seed**

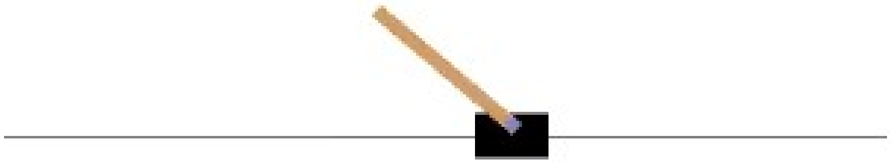
And set the following attributes:

- **action_space**: The Space object corresponding to valid actions
- **observation_space**: The Space object corresponding to valid observations
- **reward_range**: A tuple corresponding to the min and max possible rewards

Note: a default reward range set to [-inf,+inf] already exists. Set it if you want a narrower range.
The methods are accessed publicly as "step", "reset", etc...

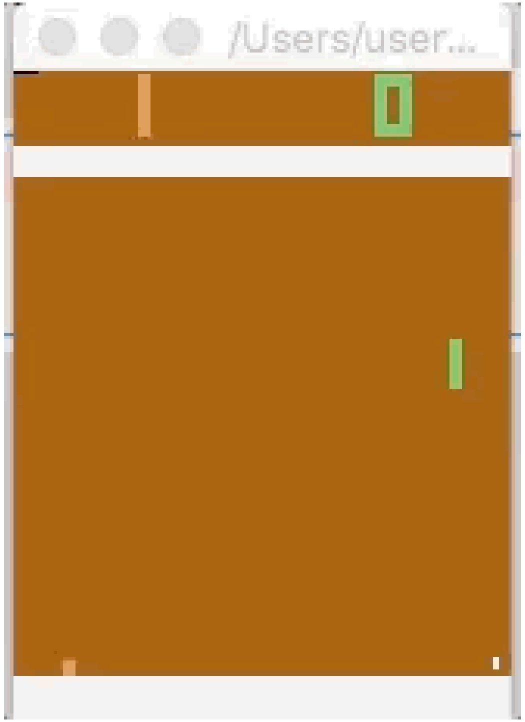
Environment – Cart Pole

```
import gym
env = gym.make('CartPole-v0')
env.reset()
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample())
env.close()
```



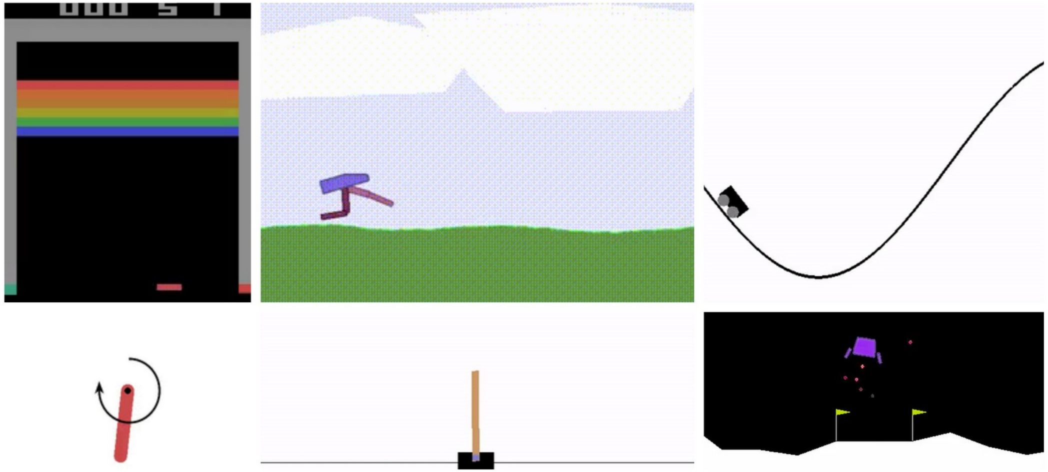
Environment – Pong

```
import gym
env = gym.make(‘Pong-v0')
env.reset()
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample())
env.close()
```



Observations

- Observation (object): an environment - specific object representing your observation of the environment. For example, pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game.



env.step(<action>)

- The environment’s step function returns exactly what we need. In fact, step returns four values. These are:
  - observation (object)
  - reward (float)
  - done (Boolean)
  - info (dict)

Spaces

- We have been sampling random actions from the environment’s action space.
- But what actually are those actions? Every environment comes with an action_space and an observation_space.
- These attributes are of type Space, and they describe the format of valid actions and observations.

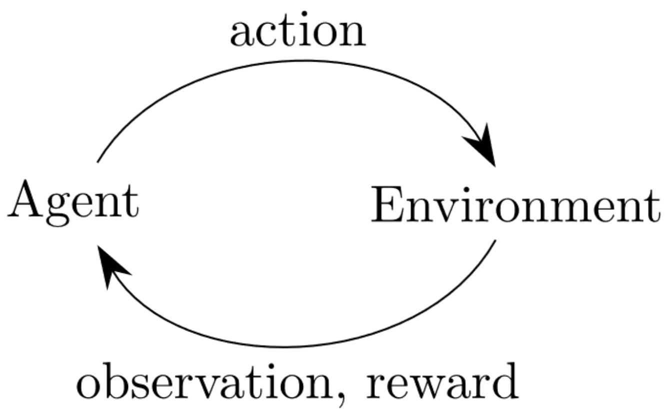
```
import gym
env = gym.make('CartPole-v0')

print(env.action_space)
#> Discrete(2)

print(env.observation_space)
#> Box(4,)
```

Agent – Action - Reward

```
import gym
env = gym.make('CartPole-v0')
for i_episode in range(20):
    observation = env.reset()
    for t in range(100):
        env.render()
        print(observation)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
env.close()
```



Q & A