Deep Reinforcement Learning Lab

OpenAI Gym: CartPole-v0

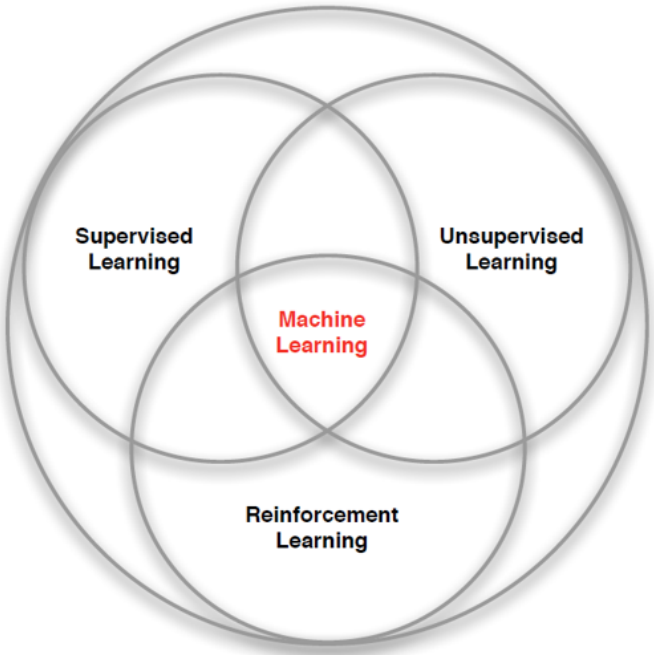
Wei Li

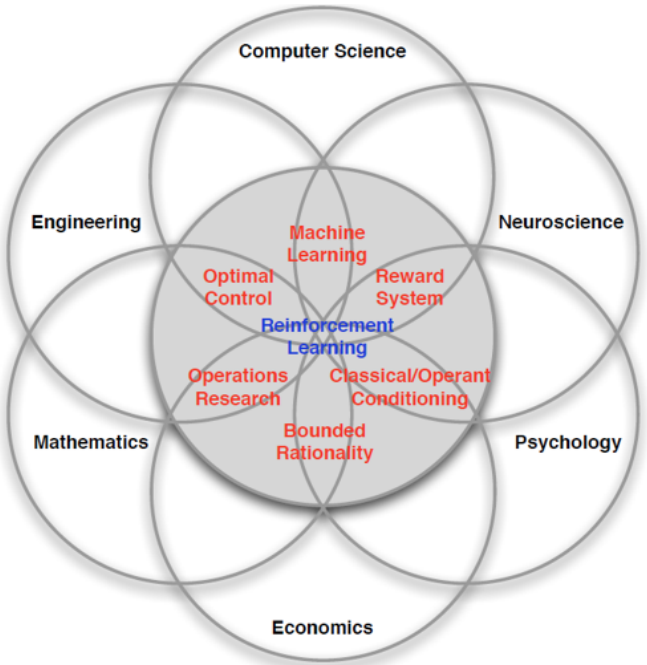
National Chiao-Tung University
Computer Games and Intelligence (CGI) Lab

December 13, 2017

Branches of Machine Learning

- Supervised Learning(SL)
  - learning from a training set of labeled examples provided by a knowledgeable external supervisor
- Unsupervised Learning(UL)
  - typically about finding structure hidden in collections of unlabeled data.
- Reinforcement Learning(RL)
  - learning from interaction.





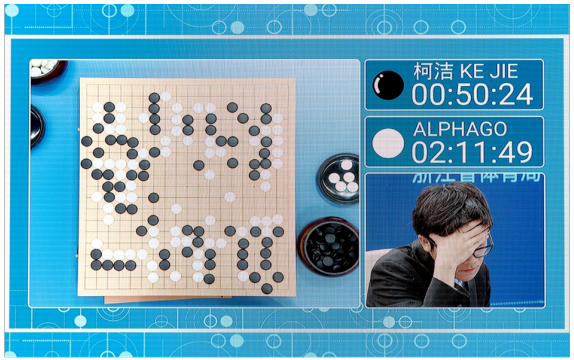
Characteristics of Reinforcement Learning

What makes reinforcement learning di erent from other machine learning paradigms?

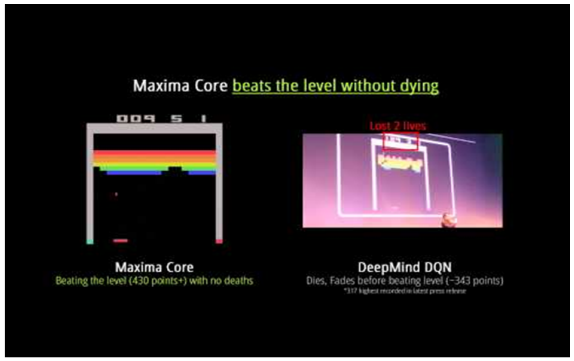
- There is no supervisor, only a reward signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions a ect the subsequent data it receives

Examples of Reinforcement Learning

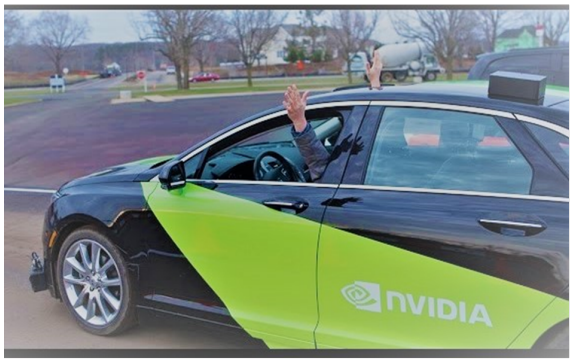
Train your dog



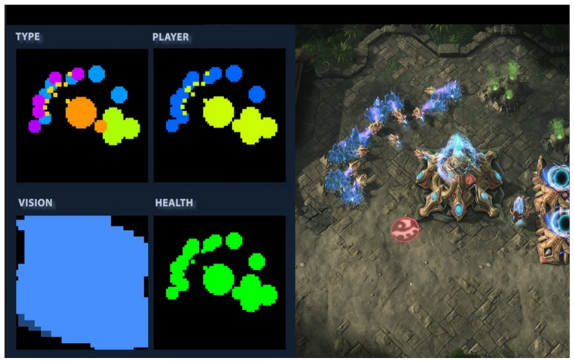
AlphaGo



Atari Game



Self–driving

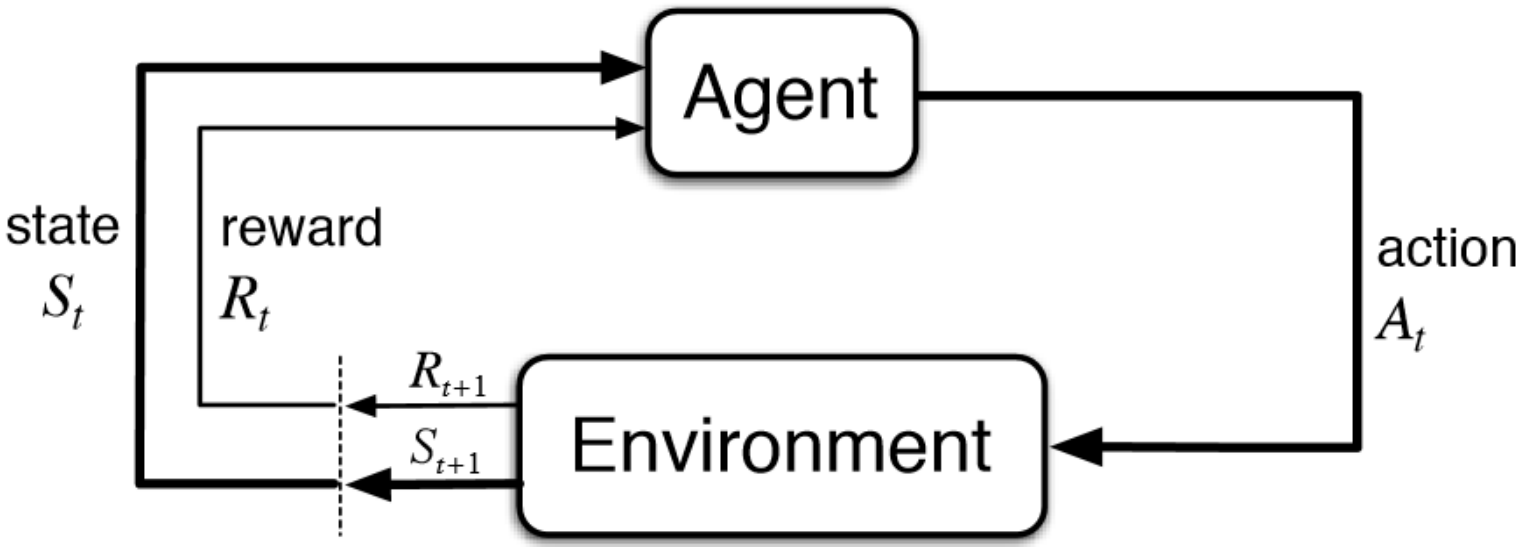


StarCraft2

Robotic Arm

What is reinforcement learning

- Reinforcement learning is learning what to do — how to map situations to actions — so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. —— Sutton



Markov Decision Processes

- Markov decision processes formally describe an environment for reinforcement learning
- Almost all RL problems can be formalized as MDPs

Definition

A Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix,
  - $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | s_t = s, A_t = a]$
- $\mathcal{R}$ is reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | s_t = s, A_t = a]$
- γ is a discount factor, $\gamma \in [0, 1]$.

Return

Definition

The return G_t is the total discounted reward from time-step t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- This values immediate reward above delayed reward.
  - close to 0 leads to "myopic" evaluation
  - close to 1 leads to "far-sighted" evaluation

Why discount

Most Markov reward and decision processes are discounted. Why?

- Mathematically convenient to discount rewards
- Avoids in nite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is nancial, immediate rewards may earn more interest than delayed rewards
- Animal/human behaviour shows preference for immediate reward
- It is sometimes possible to use undiscounted Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate.

Value Function

Definition

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

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

Definition

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

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

Optimal Value Function

Definition

The optimal state-value function $v(s)$ is the maximum value function over all policies

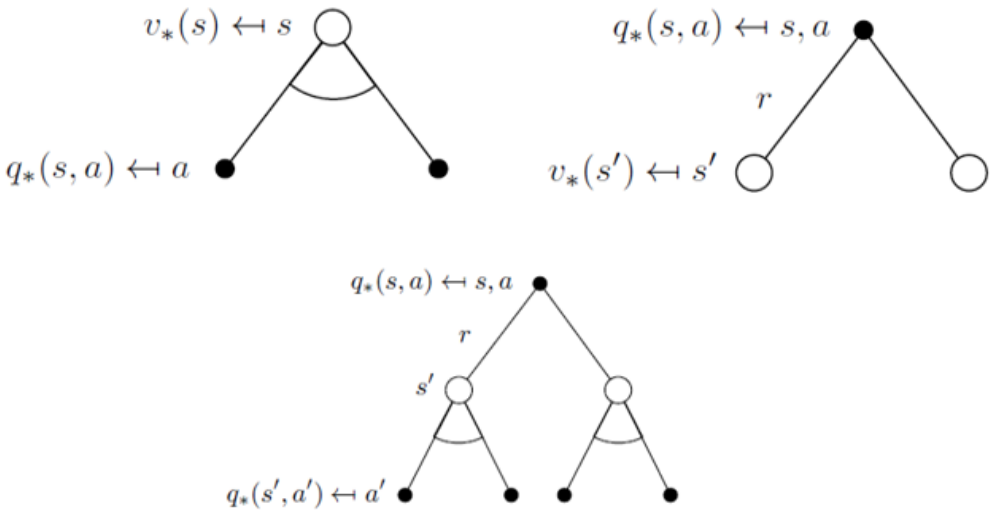
$$v_*(s) = \max_\pi v_\pi(s)$$

Definition

The optimal action-value function $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

Bellman Optimality Equation



$$v_*(s) = \max_a q_*(s,a)$$

$$q_*(s,a) = \max_\pi q_\pi(s,a) = \max_\pi \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

$$= R_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

$$= R_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a')$$

Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa

Approaches To Reinforcement Learning

- Value-based RL
- Policy-based RL
- Value-based + Policy-base (Actor-Critic) RL
- Model-free RL
- Model-based RL

DL + RL = Deep Reinforcement Learning

- Use deep neural networks to represent
  - Value function
  - Policy
  - Model
- Optimise loss function by stochastic gradient descent

A Quick Overview

- Value-based
  - (DQN) Deep Q Network
    - Playing Atari with Deep Reinforcement Learning(2013)
    - Human-level control through deep reinforcement learning
  - Other improvements
    - (DDQN) Deep Reinforcement Learning with Double Q-learning
    - Prioritized Prioritized Experience Replay
    - Dueling Network Architectures for Deep Reinforcement Learning
- Actor-Critic
  - (DDPG) Deep Deterministic Policy Gradient
  - (A3C) Asynchronous Methods for Deep Reinforcement Learning
  - (ACER) Sample Efficient Actor-Critic with Experience Replay
- More
  - TRPO, DPPO, Rainbow etc.

A Quick Overview

- Value-based
  - (DQN) Deep Q Network
    - Playing Atari with Deep Reinforcement Learning(2013)
    - Human-level control through deep reinforcement learning
  - Other improvements
    - (DDQN) Deep Reinforcement Learning with Double Q-learning
    - Prioritized Prioritized Experience Replay
    - Dueling Network Architectures for Deep Reinforcement Learning
- Actor-Critic
  - (DDPG) Deep Deterministic Policy Gradient
  - (A3C) Asynchronous Methods for Deep Reinforcement Learning
  - (ACER) Sample Efficient Actor-Critic with Experience Replay
- More
  - TRPO, DPPO, Rainbow etc.

Q-learning

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

- See more in Reinforcement Learning/Q-learning

Q-Networks

- Iterative Update

$$Q_{i+1}(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q_i(s',a')|s,a]$$

  can converge to optimal action-value function when $i \to \infty$
- Represent value function by Q-network with weights $\mathbf{w}$

$$Q(s,a,\mathbf{w}) \approx Q^*(s,a)$$

Deep Q-Networks(DQN)

- Employs a neural network to estimate the value function of each discrete action
  - Action Value: $Q(s, a, \mathbf{w})$
- When acting, selects the maximally valued output for a given state
  - Action Value: $\arg\max_a Q(s, a, \mathbf{w})$
- Loss Function: $L_i(w_i) = \mathbb{E}[y_i - Q(s, a, w_i)^2]$

Deep Q-Networks(first Ver.)

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Deep Q-Networks(Target Q)

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$,T **do**

With probability ε select a random action a_t

otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

End For

Double DQN

The idea of Double Q-learning is to reduce overestimations by decomposing the max operation in the target into action selection and action evaluation.

Evaluate the greedy policy according to the online network.

Using the target network to estimate its value.

$$Y_t^{DQN} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta_t^-})$$

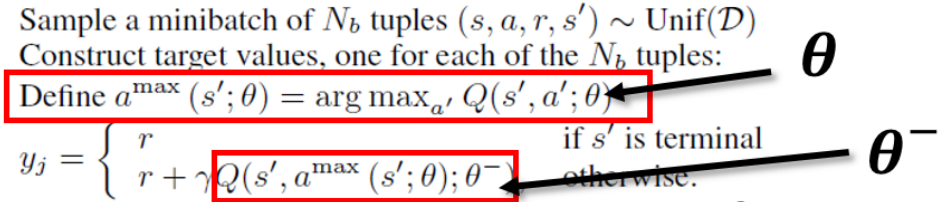
$$Y_t^{DQN} \equiv R_{t+1} + \gamma Q\left(S_{t+1}, \operatorname{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta_t^-}); \boldsymbol{\theta_t^-}\right)$$

$$Y_t^{DoubleDQN} \equiv R_{t+1} + \gamma Q\left(S_{t+1}, \operatorname{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta_t}); \boldsymbol{\theta_t^-}\right)$$

Double DQN

Algorithm 1: Double DQN Algorithm.

input : $\mathcal{D}$ – empty replay buffer; θ – initial network parameters, θ^- – copy of θ
input : N_r – replay buffer maximum size; N_b – training batch size; N^- – target network replacement freq.
for *episode* $e \in \{1, 2, \ldots, M\}$ **do**
  Initialize frame sequence $\mathbf{x} \leftarrow ()$
  for $t \in \{0, 1, \ldots\}$ **do**
    Set state $s \leftarrow \mathbf{x}$, sample action $a \sim \pi_{\mathcal{B}}$
    Sample next frame x^t from environment $\mathcal{E}$ given (s, a) and receive reward r, and append x^t to $\mathbf{x}$
    if $|\mathbf{x}| > N_f$ **then** delete oldest frame $x_{t_{min}}$ from $\mathbf{x}$ **end**
    Set $s' \leftarrow \mathbf{x}$, and add transition tuple (s, a, r, s') to $\mathcal{D}$,
      replacing the oldest tuple if $|\mathcal{D}| \geq N_r$
    Sample a minibatch of N_b tuples $(s, a, r, s') \sim \text{Unif}(\mathcal{D})$
    Construct target values, one for each of the N_b tuples:
    Define $a^{\max}(s'; \theta) = \arg\max_{a'} Q(s', a'; \theta)$
    $y_j = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma Q(s', a^{\max}(s'; \theta); \theta^-) & \text{otherwise.} \end{cases}$
    Do a gradient descent step with loss $\|y_j - Q(s, a; \theta)\|^2$
    Replace target parameters $\theta^- \leftarrow \theta$ every N^- steps
  end
end



About our lab

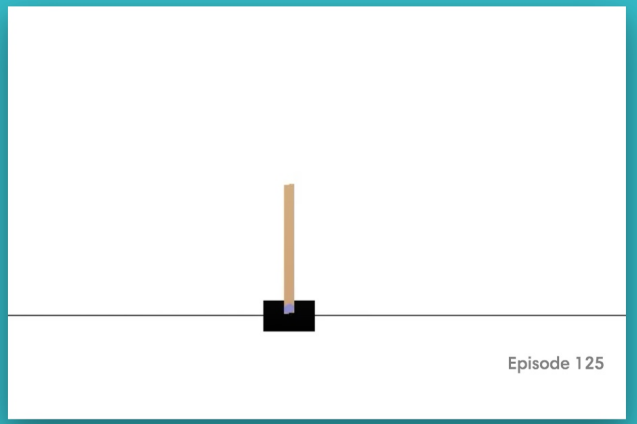
- OpenAI Gym
- Implement DQN algorithm to Train CartPole-v0

CartPole-v0

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

CartPole-v0 defines "solving" as getting average reward of 195.0 over 100 consecutive trials.

This environment corresponds to the version of the cart-pole problem described by Barto, Sutton, and Anderson [Barto83].



Svalorzen's algorithm, Took 42 episodes to solve the environment. 200.00 ± 0.00 2 months ago

- See more in SPEC