An Introduction to Reinforcement Learning

Wang, Yuanyuan

School of Statistics and Mathematics
Central University of Finance and Economics
Beijing, China

07 December 2018

The big picture

- The Story of AlphaGo: learning through **reinforcement learning** (RL)
- A long history since 1996 or earlier
- Renaissance of RL
- the combination of deep neural networks and reinforcement learning (deep RL)
- A real hope for artificial intelligence

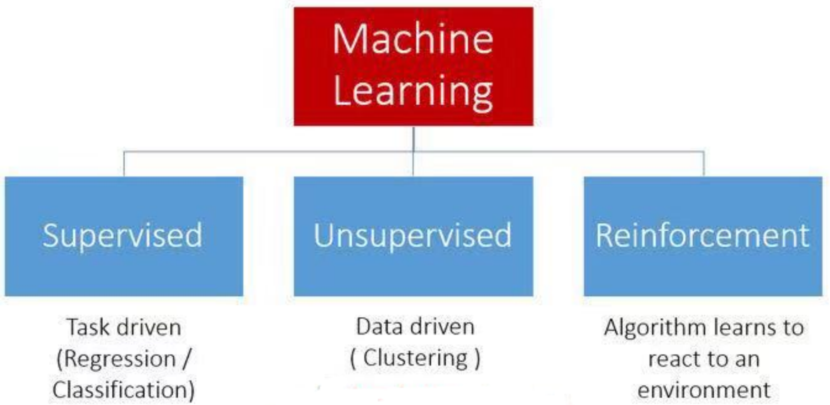
AlphaGo's 4-1 victory against Mr Lee Sedol in Seoul, South Korea, in March 2016

Outline

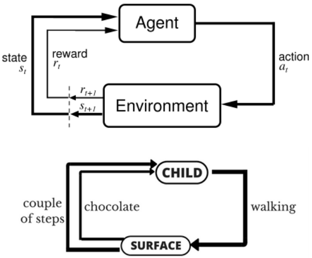
Outline

RL: A Branch of Machine Learning

Branches of Machine Learning:



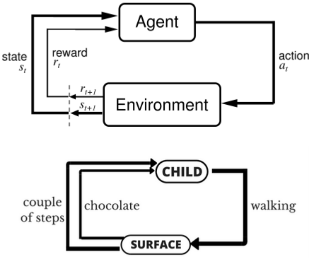
Elements of RL



elements of RL

- Agent: (who) the learner and decision maker
- State: (where) representation of the environment
- Action: (what to do)
- Reward: (what agent get) the goal in a reinforcement learning problem
- Policy: (how to do) defines the learning agent's way of behaving at a given time

Learning Process of RL



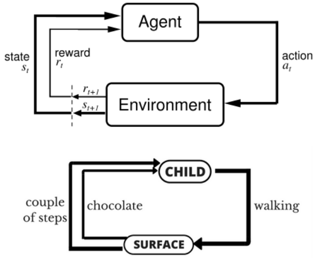
learning process of RL

At each time step t, the agent:

- Observes current state s_t
- Agent takes a sequence of actions (a_t) .
- Observes outcomes (state s_{t+1}, rewards r_{t+1}) of those actions.
- Statistically estimates relations between action choice and outcomes, $P_r(s_t|s_{t-1}, a_{t-1})$
- Ultimate goal: maxmize future rewards

- After some time... learns action selection policy $\pi(s, a) = P(a_t = a|s_t = s)$, that optimizes selected outcomes $argmax_\pi E_\pi(r_0 + r_1 + \ + r_T|s_0)$

Learning Process of RL



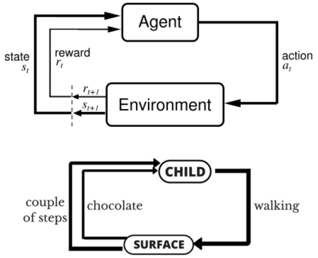
learning process of RL

At each time step t, the agent:

- Observes current state s_t
- Agent takes a sequence of actions (a_t) .
- Observes outcomes (state s_{t+1}, rewards r_{t+1}) of those actions.
- Statistically estimates relations between action choice and outcomes, $P_r(s_t|s_{t-1}, a_{t-1})$
- Ultimate goal: maxmize future rewards

- After some time... learns action selection policy $\pi(s, a) = P(a_t = a|s_t = s)$, that optimizes selected outcomes $argmax_{\pi} E_{\pi}(r_0 + r_1 + + r_T|s_0)$

Learning Process of RL



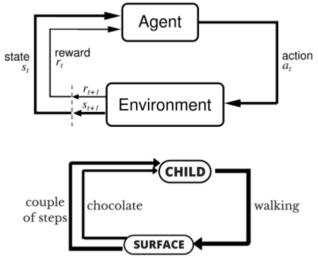
learning process of RL

At each time step t, the agent:

- Observes current state s_t
- Agent takes a sequence of actions (a_t) .
- Observes outcomes (state s_{t+1}, rewards r_{t+1}) of those actions.
- Statistically estimates relations between action choice and outcomes, $P_r(s_t|s_{t-1}, a_{t-1})$
- Ultimate goal: maxmize future rewards

- After some time... learns action selection policy $\pi(s, a) = P(a_t = a|s_t = s)$, that optimizes selected outcomes $argmax_\pi E_\pi(r_0 + r_1 + \quad + r_T|s_0)$

Learning Process of RL



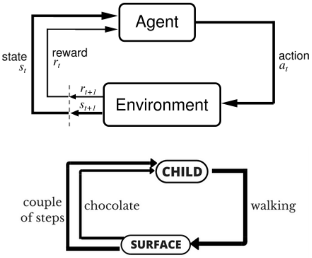
learning process of RL

At each time step t, the agent:

- Observes current state s_t
- Agent takes a sequence of actions (a_t) .
- Observes outcomes (state s_{t+1}, rewards r_{t+1}) of those actions.
- Statistically estimates relations between action choice and outcomes, $P_r(s_t|s_{t-1}, a_{t-1})$
- Ultimate goal: maxmize future rewards

- After some time... learns action selection policy $\pi(s, a) = P(a_t = a|s_t = s)$, that optimizes selected outcomes $argmax_{\pi} E_{\pi}(r_0 + r_1 + \ + r_T|s_0)$

Learning Process of RL

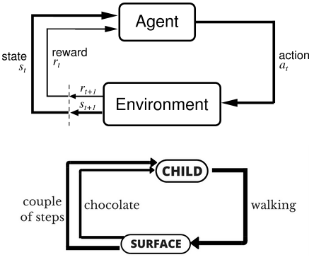


learning process of RL

At each time step t, the agent:

- Observes current state s_t
- Agent takes a sequence of actions (a_t) .
- Observes outcomes (state s_{t+1}, rewards r_{t+1}) of those actions.
- Statistically estimates relations between action choice and outcomes, $P_r(s_t|s_{t-1}, a_{t-1})$
- Ultimate goal: maxmize future rewards
- After some time... learns action selection policy $\pi(s, a) = P(a_t = a|s_t = s)$, that optimizes selected outcomes $argmax_\pi E_\pi(r_0 + r_1 + + r_T|s_0)$

Learning Process of RL



learning process of RL

At each time step t, the agent:

- Observes current state s_t
- Agent takes a sequence of actions (a_t) .
- Observes outcomes (state s_{t+1}, rewards r_{t+1}) of those actions.
- Statistically estimates relations between action choice and outcomes, $P_r(s_t|s_{t-1}, a_{t-1})$
- Ultimate goal: maxmize future rewards

- After some time... learns action selection policy $\pi(s,a) = P(a_t = a|s_t = s)$, that optimizes selected outcomes $argmax_\pi E_\pi(r_0 + r_1 + + r_T|s_0)$

What Makes RL Differ from Other Algorithms

- No right answer to refer, only a reward signal
- Consequences is delayed
- Only through trail and error
- Non-stationarity: Agent's actions affect the subsequent data it receives, data is changed after every step

RL Applications

- Robotics: A mobile robot decides which direction it should head towards
- Financial: Manage an investment portfolio, reward for each increment on account
- Games: Play many games better than humans
- NLP: dialogue system, machine translation, sequence generation
- Computer vision: recognition, motion analysis, scene understanding

Outline

Outline

The Markov property

A discrete time stochastic control process has the Markov property:

- $P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t, ..., , s_0, a_0)$, and
- $P(r_{t+1}|s_t, a_t) = P(r_{t+1}|s_t, a_t, ..., , s_0, a_0)$

Requirements of MDP:

- It is possible to estimate the desired states
- Multiple tires are allowed
- The future of the process only depends on the current states and actions

Markov Decision Process

MDP in RL: a 5-tuple (S,A,T,R,γ):

- S: state space,
- A: action space,
- P: transition function,
  (eg. P_{sa} denotes the probability distribution from current state s after action a, $P_{s'|sa}$ denotes the probabilty from state s after action a to state s')
- R : reward function, the immediate reward after a certain action
- $\gamma \in [0,1]$: is the discount factor
  1. There is a $1-\gamma$ chance that the agent dies afterwards, thus cannot receive rewards afterwards.
  2. Receiving a reward tomorrow, is worth less than today by a factor of γ

Markov Decision Process

the agent:

- In a initial state S_0
- Choose an action a_0
- Transit to next state S_1 by transition function P_{sa}
- Repeat ....

- $S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} S_2 \xrightarrow{A_2} S_3 \xrightarrow{A_3} \ldots$

Markov Decision Process

the agent:

- In a initial state S_0
- Choose an action a_0
- Transit to next state S_1 by transition function P_{sa}
- Repeat ....

- $S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} S_2 \xrightarrow{A_2} S_3 \xrightarrow{A_3} \ldots$

Markov Decision Process

the agent:

- In a initial state S_0
- Choose an action a_0
- Transit to next state S_1 by transition function P_{sa}
- Repeat ....

- $S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} S_2 \xrightarrow{A_2} S_3 \xrightarrow{A_3} \ldots$

Markov Decision Process

the agent:

- In a initial state S_0
- Choose an action a_0
- Transit to next state S_1 by transition function P_{sa}
- Repeat ....
- $S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} S_2 \xrightarrow{A_2} S_3 \xrightarrow{A_3} \ldots$

The Expected Return: V-value Function

To find a policy that maximizes the reward, we need to estimate **the expected return** for every policy π,

$$V_\pi(s) = E_\pi(\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, \pi)$$

- it is the the expectation of future returns at a time t given state s.
- it is to evaluate the value of a certain state or state-action pairs.

Optimal expected return:

$$V^*(s) = max_\pi V^\pi(s)$$

Bellman's equation

Current state return consists of two parts:

- immediate reward
- future expected reward

Bellman's equation for **a state** with a fixed policy

$$V_\pi(s) = \sum_{a \in A} \pi(s,a)[\overbrace{r(s,a)}^{\text{immediate}} + \gamma \overbrace{\sum_{s' \in S} P(s'|s,a) V_\pi(s')}^{\text{future}}]$$

Bellman's optimal equation for **a state-action pair**:

$$Q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q_\pi(s',a')$$

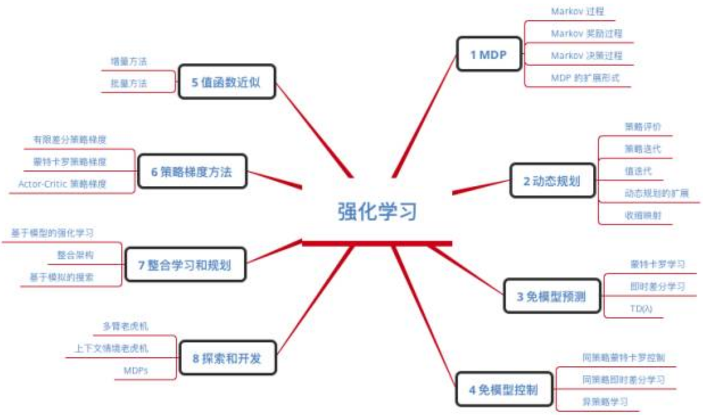
Outline

1. What is Reinforcement Learning

2. Reinforcement Learning and Markov Decision Process

3. Solution Methods for Reinforcement Learning
   - Solution Methods Overview
   - Q-Learning
   - Deep Q-Learning

An Overview of Solution Methods



- Value-based approach: Q-Learning, Deep Q-Learning
- Policy-based approach: find policy function
- Model-based: estimate changes in environment

Q-Table: Element of Q-Learning

$$Q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q_\pi(s',a')$$

1. Update $Q(s,a)$ through iteration, until $Q(s,a)$ converge.
2. So the policy is given: which action brings highest Q.

What Q-Learning do is to save $Q(s,a)$ in a table, like this:

	a1	a2	a3	a4
s1	Q(1,1)	Q(1,2)	Q(1,3)	Q(1,4)
s2	Q(2,1)	Q(2,2)	Q(2,3)	Q(2,4)
s3	Q(3,1)	Q(3,2)	Q(3,3)	Q(3,4)
s4	Q(4,1)	Q(4,2)	Q(4,3)	Q(4,4)

- Row: state
- Column: action
- Value: $Q(s,a)$

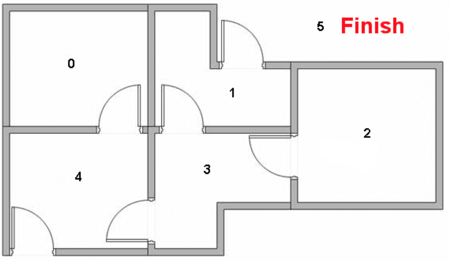
Underlying Q-Learning: Value Iteration

Require: initialize V arbitrarily (e.g. $V(s) := 0, \forall s \in S$)
  repeat
    $\Delta := 0$
    for each $s \in S$ **do**
      $v := V(s)$
      for each $a \in A(s)$ **do**
        $Q(s,a) := \sum_{s'} T(s,a,s') \left(R(s,a,s') + \gamma V(s') \right)$
      $V(s) := \max_a Q(s,a)$
      $\Delta := \max(\Delta, |v - V(s)|)$
  until $\Delta < \sigma$

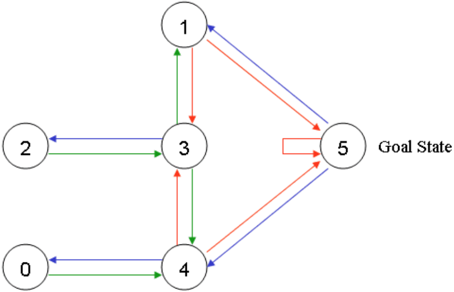
Value Iteration Pseudocode

- Initialize $Q(s,a) = 0$
- Update Q by Bellman's equation
- Until convergence

Value Iteration: A Simple Instance

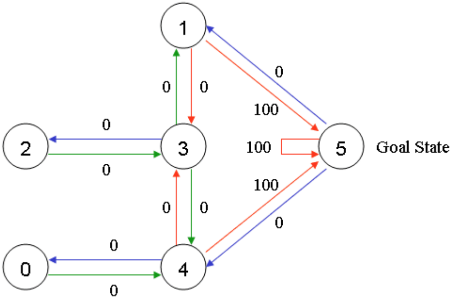


a RL instance



simple version

Value Iteration: A Simple Instance



initial

$$
\begin{array}{cc}
 & \text{Action} \\
\text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array}
\end{array}
$$

$$
R= \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{bmatrix}
-1 & -1 & -1 & -1 & 0 & -1 \\
-1 & -1 & -1 & 0 & -1 & 100 \\
-1 & -1 & -1 & 0 & -1 & -1 \\
-1 & 0 & 0 & -1 & 0 & -1 \\
0 & -1 & -1 & 0 & -1 & 100 \\
-1 & 0 & -1 & -1 & 0 & 100
\end{bmatrix}
$$

$$
\begin{array}{c} \\ \\ \end{array}
\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array}
$$

$$
Q= \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

initial R and Q

Value Iteration: A Simple Instance

Randomly choose s=1:

$$Q = \begin{array}{c c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0\\1\\2\\3\\4\\5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Also randomly choose s=3:

$$Q = \begin{array}{c c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0\\1\\2\\3\\4\\5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Final result:

$$Q = \begin{array}{c c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0\\1\\2\\3\\4\\5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

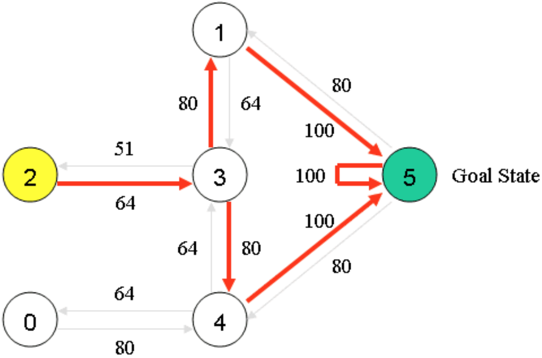
Final result after normalization:

$$Q = \begin{array}{c c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0\\1\\2\\3\\4\\5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

$Q(1,5) = r(1,5) + 0.8 * Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100$
$Q(3,1) = r(3,1) + 0.8 * Max[Q(1,2), Q(1,5)] = 0 + 0.8 * Max(0,100) = 80$
......

Value Iteration: A Simple Instance

Best policy:



Q-Learning: Improvemens

1. Value Iteration:

$$Q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q_\pi(s',a')$$

2. Q-Learning is to iterate within finite states and actions:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \underbrace{(\overbrace{r_t + \gamma \max_a Q(s_{t+1},a)}^{\text{estimated optimal value}} - Q(s_t,a_t))}_{\Delta Q}$$

- Like gradient descending, its gradient 'ascending'
- α: like learning rate
- time-saving

Q-Learning Algorithm

Q-Learning Pseudocode:

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'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

Exploration and Exploitation

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):
        $\boxed{\text{Choose } A \text{ from } S \text{ using policy derived from } Q \text{ (e.g., } \epsilon\text{-greedy)}}$
        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

Exploration and Exploitation

Policies:

- stochastic: exploration
- greedy policy: exploitation, $\pi(s_{t+1}) = argmax_a Q(s_{t+1}, a)$
- $\epsilon-$greedy: combine exploration and exploitation, ϵ is often a samll value, is the probability of choosing stochastic action, determines the proportion of exploration and exploitation

Solution Categories:

- Off-policy vs On-policy
- Model-free vs Model-based
- ...

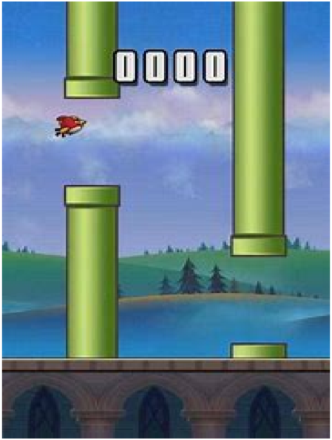
Value Function Approximation

Tabular vs Function approximation:

Tabular: save Q value in a table, but when:

- Too many states: raw image, Go, ...
- Continuous state spaces.

Go: 10^{170}states



A $10 * 10$ pixes 8-bit gray-scale image: 256^{100} states

Value Function Approximation

1. Q table Update $\rightarrow$ Value Function Fitting
2. Function approximator: $Q(s, a; \omega)$
   - For example, in linear model:

$$Q(s, a) = w_1 s + w_2 a + b$$

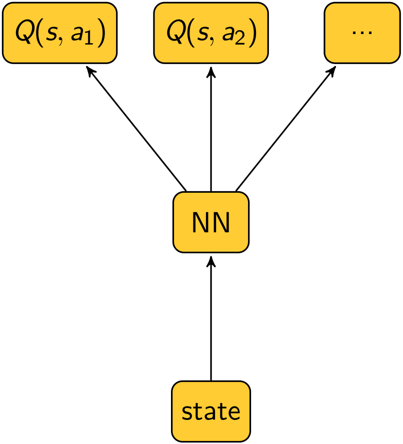
   - Let ω denote the parameters in f

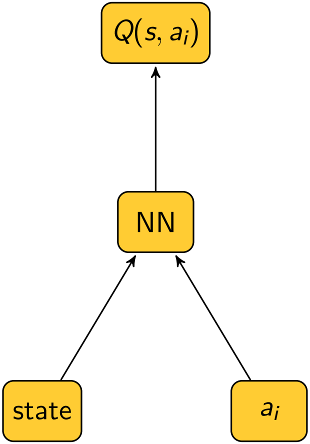
$$Q(s, a; \omega) \approx Q'(s, a)$$

3. Supervised learning:
   - Define a loss function
   - Caculate gradient of loss function
   - update ω through SGD or else

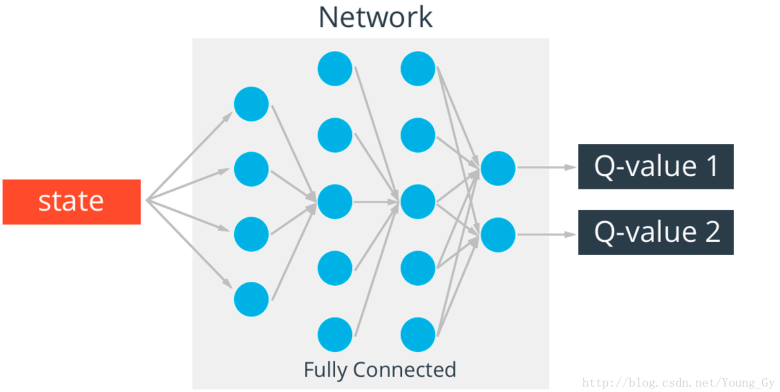
Deep Q-Learning (NIPS 2013 and Nature 2015)

Use deep neural network to approximate Q: $Q(s, a; \omega) \approx Q'(s, a)$

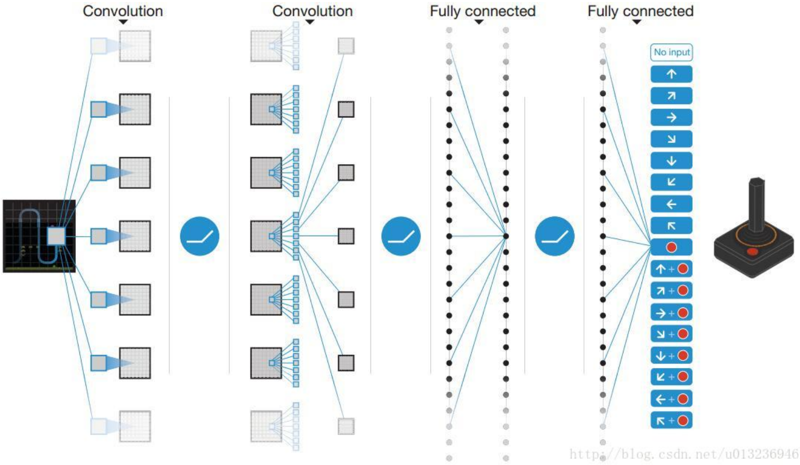




DQN: DEMO 1


Network
state
Fully Connected
Q-value 1
Q-value 2
http://blog.csdn.net/Young_Gy

Deep: DEMO 2


Convolution
Convolution
Fully connected
Fully connected
No input
http://blog.csdn.net/u013236946

Deep Q-Learning

$Q(s, a; \omega) \approx Q'(s, a)$

Questions:

- High-volume labeled data
- Independent sample vs trajectory data
- Sationary vs nonsationary distribution
- Unstable

Strategies

- Use Q-Learning data as labels
- Experience replay
- Fixed Q-targets

DQN: Use Q-Learning labels

Train the Q-Network:

- High-volume labeled data

$$Q_{label} = r + \gamma \max_{a'} Q(s', a', \omega)$$

- Loss function: MSE

$$L(\omega) = E[(r + \gamma \max_{a'} Q(s', a', \omega) - Q(s, a, \omega))^2]$$

- update ω through gradient descending

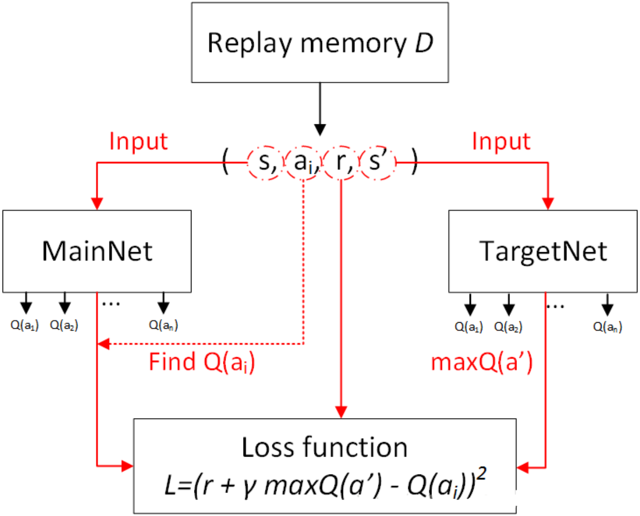
DQN: Pseudocode code (Nature 2015)

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 $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$
    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
End For

DQN: Experience replay and Fixed Q-targets

- Experience replay: like a memory pool
  - it is to deal with trajectory data and nonsationary distribution
  - Experience replay is a short-term memory mechanism, later follows LSTM
- Fixed Q-targets:
  - MainNet: $Q(s, a; \theta_i)$ produce current output
  - TargetNet: $Q(s, a; \theta_i^-)$ produce TargetQ
  - Purpose: by using TargetNet, target Q value is constant within C steps, lower the relations between Current Q and Target Q, makes Network more stable

DQN: Chart Flow



Thanks!

Thank you for your time and attention.

And have a nice weekend!

Questions?

References

- https://deepmind.com/research/alphago/
- Reinforcement Learning An Introduction (2nd Edition)
- Human-level control through deep reinforcement learning
- Mastering the game of Go with deep neural networks and tree search