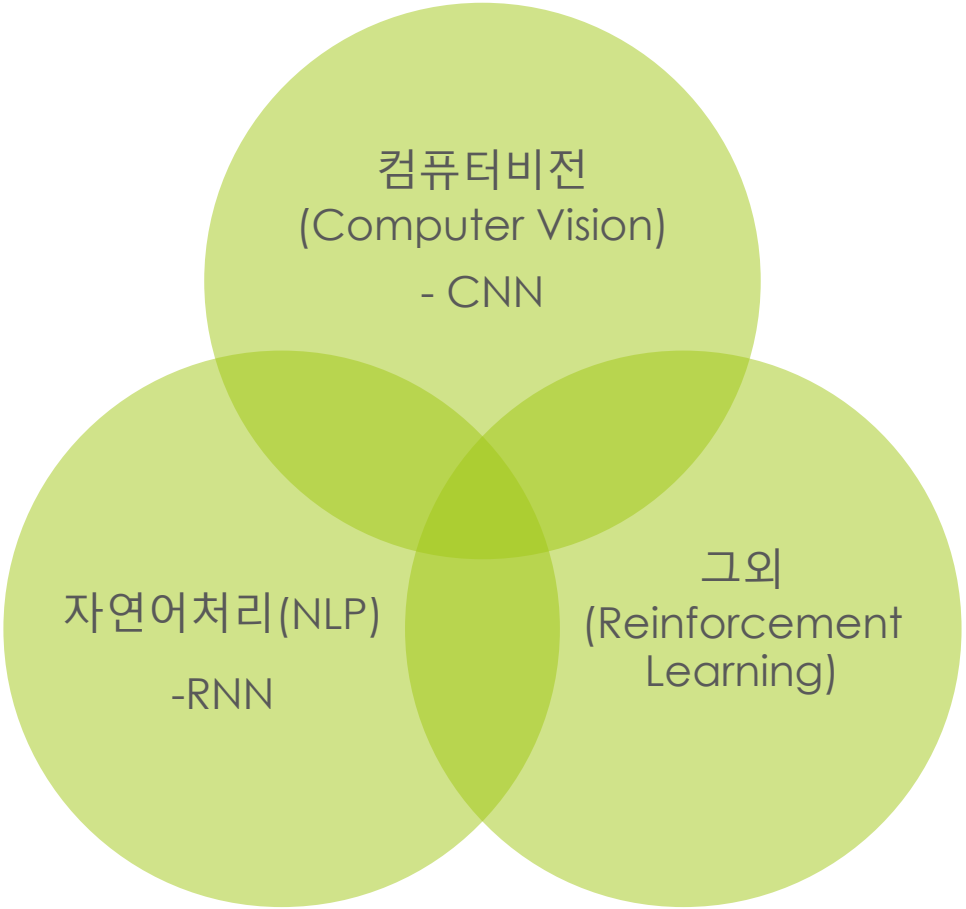
Reinforcement Learning

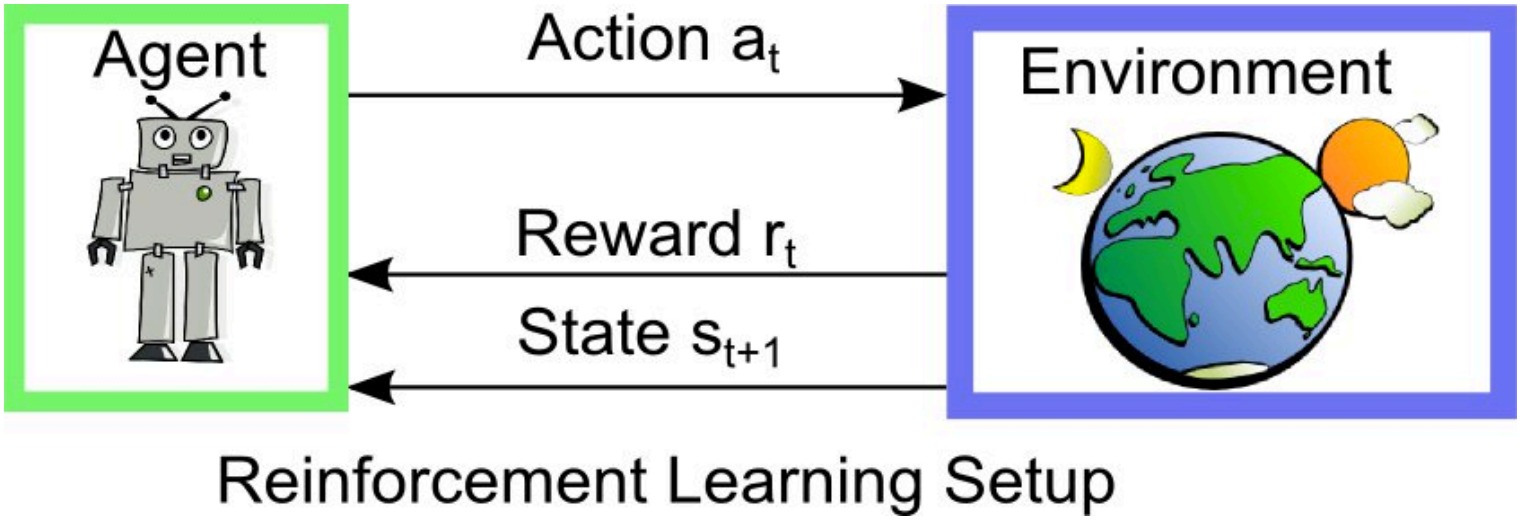
Solaris
(http://solarisailab.com)

딥러닝의 다양한 응용분야들



Reinforcement Learning

- Reinforcement Learning 알고리즘은 에이전트(Agent)가 환경(Environment)과 상호작용하면서 정보를 학습할 수 있도록 하고 이를 통해, 최적의 행동(Optimal Policy)을 찾도록 도와준다.



Reinforcement Learning Setup

Reward Hypothesis

- 강화학습의 기본 가정은 어떤 목표(Goal)든 보상(Reward)를 최대화하는 것으로 표현 될 수 있다는 것이다.

- **강화 가정(Reward Hypothesis) – 모든 목표는 보상을 최대화하는 것으로 나타낼 수 있다.**

- 간단한 예로, 인간은 기본적으로 어떤 행동을 할 때, 보상이 최대화 되는 방향으로 행동한다. 예를 들어 우리의 목표가 10만원을 버는 것이라고 생각해보자. 이때, A라는 상점에서 하루 동안 청소를 하면 10만원을 주고, B라는 상점에서 하루 동안 청소를 하면 8만원을 준다고 생각해보자. 자연스럽게 우리는 A라는 상점에서 일을 하기로 결정할 것이다.

- 이제 문제는 이런 상식적인 명제를 구체적인 수학적인 형태로 표현하는 것이다. 이를 위해서 강화 학습(Reinforcement Learning)은 **상태 가치 함수(State-Value Function)**와 **행동 가치 함수(Action-Value Function)**라는 개념을 사용한다.

State-Action Value

- **상태 가치 함수(State-Value Function)**는 현재 상태의 좋고/나쁨을 표현한다. 상태 가치 함수를 수학적으로 표현하면 아래와 같다.

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s]$$

- 위의 식이 의미하는 바를 해석해보자. 어떤 시간 t에서 전략 π를 따를때 기대되는(E_π-nondeterministic일 경우 평균-) 어떤 상태 $s(V_\pi(s), S_t = s)$의 가치는 미래의 보상들의 총합$(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots)$으로 표현된다.

- 여기서 **γ -감마-는 discount factor**이다. γ 는 보통 0에서 1의 값을 부여한다. 만약 γ 가 0이라면 오직 다음 시간$(t+1)$의 보상만을 고려한다. 이때의 장점은 빨리 최적의 행동을 결정할 수 있다는 점이다. γ 가 1이라면 미래의 보상도 바로 다음의 보상만큼 중요하게 생각한다. 이 경우, 당장의 보상은 최대화 할 수 없지만 미래의 수까지 내다보면서 행동을 할 수 있다는 장점이 있다. 실제 상황에서는 문제에 따라 최적의 γ 이 다르고, 실험을 통해 최적의 γ를 설정해 주어야 한다.

- 정리하자면, 상태 가치 함수 $V_\pi(s)$ 는 현재 상태 s 의 가치를 정량적으로 나타낸다.

Action-Value Function

- **행동 가치 함수(Action-Value Function)**는 수학적으로 다음과 같이 정의 된다.

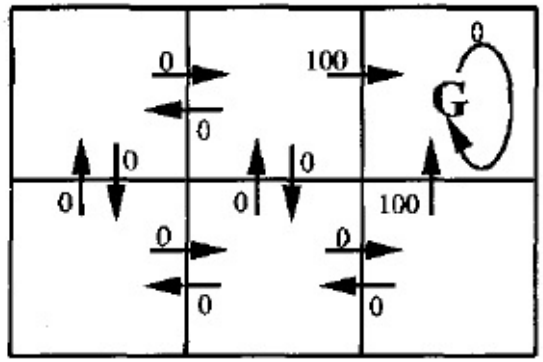
$$Q_\pi(s,a) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s, A_t = a]$$

- 위의 식이 의미하는 바를 해석해보자. 어떤 시간 t 일때 전략 π를 따를때 기대되는(E_π - nondeterministic일 경우 평균-)어떤 상태 $s(S_t = s)$에서 어떤 행동$(A_t = a)$을 했을때의 가치$(Q_\pi(s,a))$는 미래의 보상들의 총합$(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots)$이다.이제 예제를 통해, 상태 가치 함수(State-Value Function)와 행동 가치 함수(Action Value Function)에 대한 개념을 구체적으로 이해해보자.

Grid-World Example

- **예제 1 – 우리는 Goal 지점이 출구인 미로-그리드월드(Grid World)-에 갖혀 있다. Goal에 도달 할때는 100점의 보상(Reward)를 얻고 그 외의 지점에서는 0의 보상(Reward)를 얻는다.**

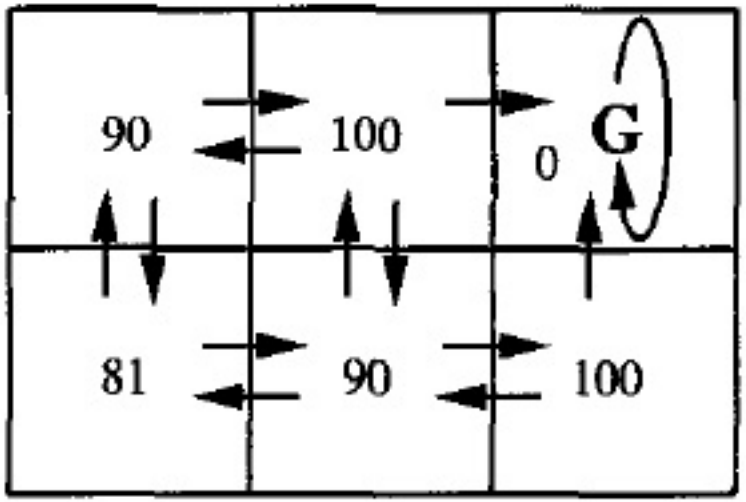
- 위의 예제1에서 Goal에 도달 했을때 100점의 보상을 얻는다. 따라서 우리의 최종 목표는 최대한 적은 이동을 통해 Goal에 도달해 미로를 탈출하는 것이다. 상태 가치 함수와 행동 가치 함수를 이용해서 이 상황을 수학적으로 표현해보자.



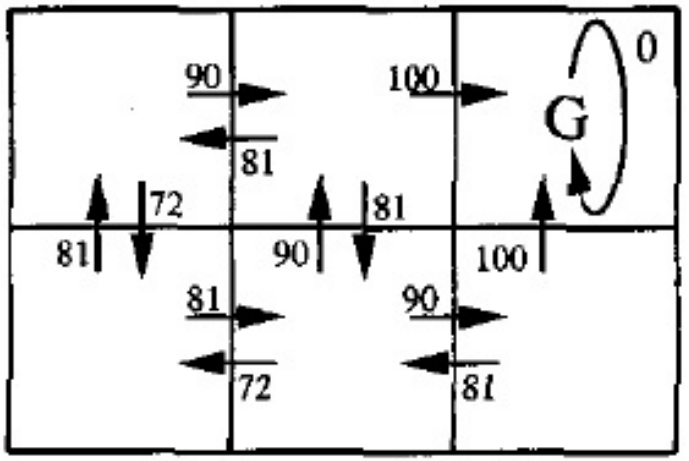
$r(s, a)$ (immediate reward) values

Grid-World Example

- 예제 1 – 우리는 Goal 지점이 출구인 미로-그리드월드(Grid World)-에 갖혀 있다. Goal에 도달 할때는 100점의 보상(Reward)를 얻고 그 외의 지점에서는 0의 보상(Reward)를 얻는다.



$V^*(s)$ values



$Q(s, a)$ values

Q-Learning

Bellman equation

The optimal Q-value function Q* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Q* satisfies the following **Bellman equation**:

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right]$$

Intuition: if the optimal state-action values for the next time-step Q*(s',a') are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s',a')$

The optimal policy **π*** corresponds to taking the best action in any state as specified by Q*

Slide credit: CS231n (https://goo.gl/QSA36t)

Q-Learning

Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

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

Q_i will converge to Q* as i -> infinity

What's the problem with this?
Not scalable. Must compute Q(s,a) for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

Solution: use a function approximator to estimate Q(s,a). E.g. a neural network!

Slide credit: CS231n (https://goo.gl/QSA36t)

Q-Learning

Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

function parameters (weights)

If the function approximator is a deep neural network => **deep q-learning**!

Slide credit: CS231n
(https://goo.gl/QSA36t)

Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') | s, a \right]$$

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s,a;\theta_i))^2 \right]$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s, a \right]$

Iteratively try to make the Q-value close to the target value (y_i) it should have, if Q-function corresponds to optimal Q* (and optimal policy π^*)

Backward Pass

Gradient update (with respect to Q-function parameters θ):

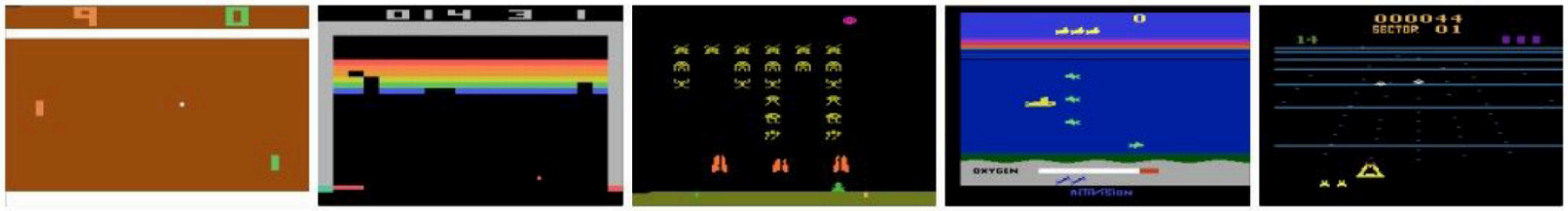
$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)) \nabla_{\theta_i} Q(s,a;\theta_i) \right]$$

Slide credit: CS231n (https://goo.gl/QSA36t)

DQN

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Case Study: Playing Atari Games



Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

Slide credit: CS231n (https://goo.gl/QSA36t)

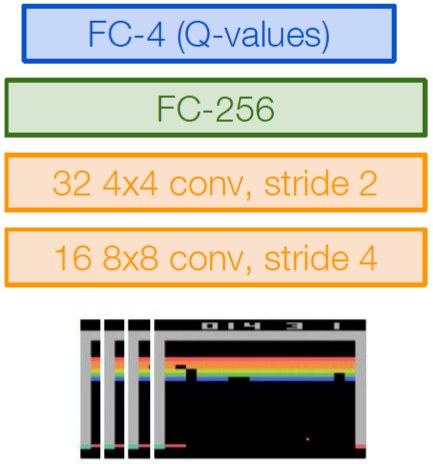
DQN

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Q-network Architecture

$Q(s, a; \theta)$:
neural network
with weights θ

A single feedforward pass to compute Q-values for all actions from the current state => efficient!



Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$

Number of actions between 4-18 depending on Atari game

Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

Slide credit: CS231n
(https://goo.gl/QSA36t)

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Training the Q-network: Experience Replay

Learning from batches of consecutive samples is problematic:

- Samples are correlated => inefficient learning
- Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

Address these problems using **experience replay**

- Continually update a **replay memory** table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

Each transition can also contribute to multiple weight updates
=> greater data efficiency

Slide credit: CS231n (https://goo.gl/QSA36t)

DQN

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Putting it together: Deep Q-Learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

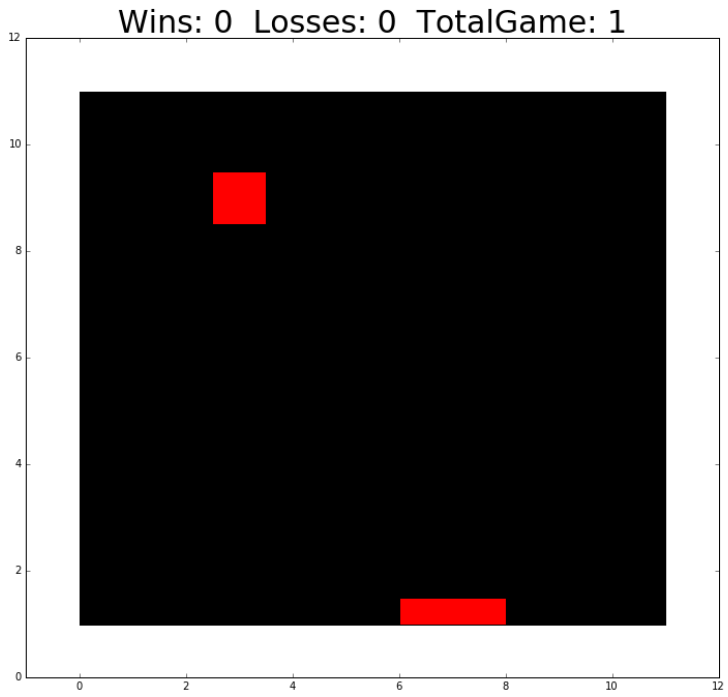
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise 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

Experience Replay: Sample a random minibatch of transitions from replay memory and perform a gradient descent step

Slide credit: CS231n (https://goo.gl/QSA36t)

TensorFlow를 이용한 DQN 구현

- https://github.com/solaris33/deep-learning-tensorflow-book-code/tree/master/Ch12-DQN



Questions & Answers

Thank You!