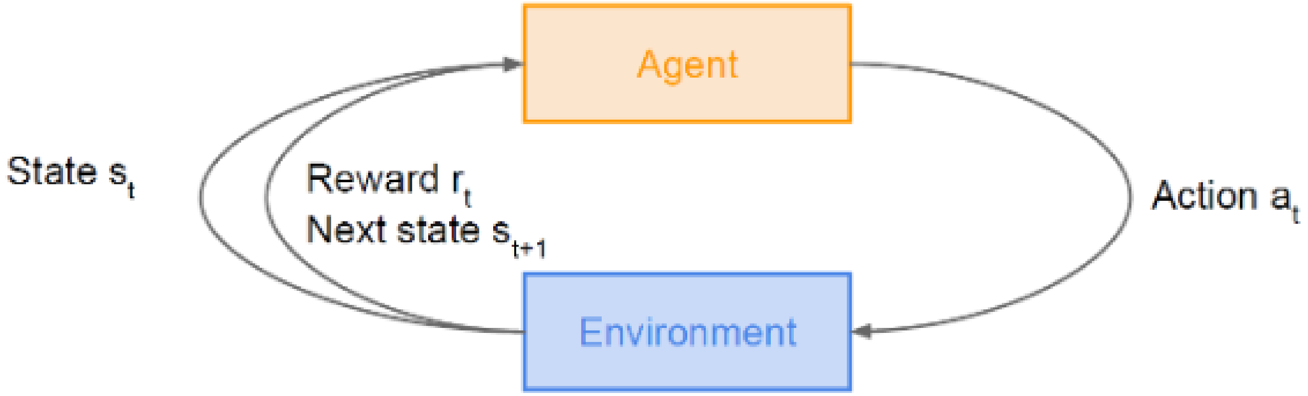
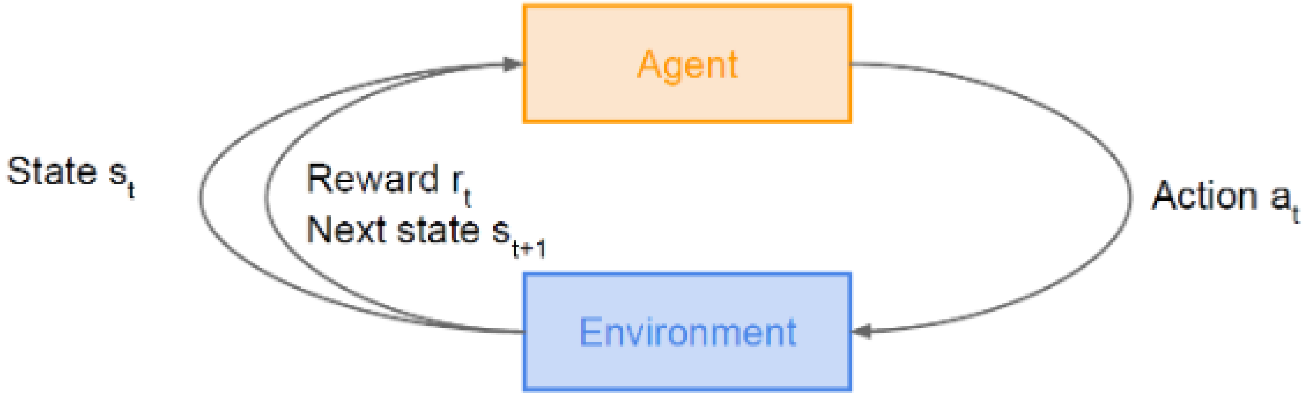
CS231N 14강
Reinforcement Learning

Reinforcement Learning(강화학습)



- Agent: Environment에서 Action 취할 수 있는 물체
- Environment: Agent와 상호작용하여 Agent state 설정
- 순서:
  1) Environment->Agent state부여
  2) Agent가 Action
  3) Action에 대해 Agent가 보상받음
  4) State 부여 받음

Reinforcement Learning(강화학습)



강화 학습으로 풀 수 있는 문제
- Cart-Pole Problem (Cart위에서 Pole 균형잡기)
- Robot Locomotion(로봇을 앞으로 가기)
- Atari Games(가장 높은 점수로 게임 끝내기)
- Go(바둑 게임 이기기)

Markov Decision Processes
:강화학습 방법 수식화

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

Markov Decision Process

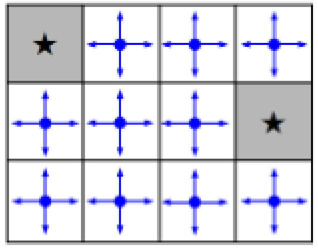
- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(. \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. \mid s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

- A policy **π** is a function from S to A that specifies what action to take in each state
- **Objective**: find policy **π*** that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

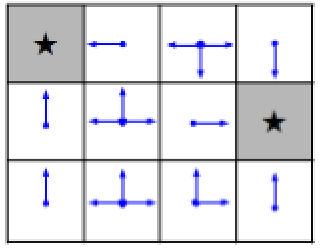
- 누적 보상을 최대화하는 π* 찾는것을 목표

Markov Decision Processes

A simple MDP: Grid World



Random Policy



Optimal Policy

누적 보상을 최대화하는 π* 찾는것을 목표
-> 미래에 내가 받을 보상들의 합이 최대로!

Formally: $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right]$ with $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$

Markov Decision Processes

- **Value function**:어떤 상태S와 정책 π가 주어졌을때, 계산되는 누적 보상의 기댓값

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

- **Q-Value function**: 상태 S에서 어떤 행동을 해야 가장 좋은지 알려주는 함수
  (상태,행동) -> 보상?

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

Markov Decision Processes

- **Bellman equation**: 현재 state의 value functio과 다음 state의 value function 사이의 관계식을 나타냄
- Value iteration algorithm : 반복적인 update로 벨만 방정식을 이용하여 점차적으로 Q*를 최적화시키는 방법

  문제점

  - 계산량이 많음 -> Q(s,a) 근사시키는 방법 필요

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**!

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

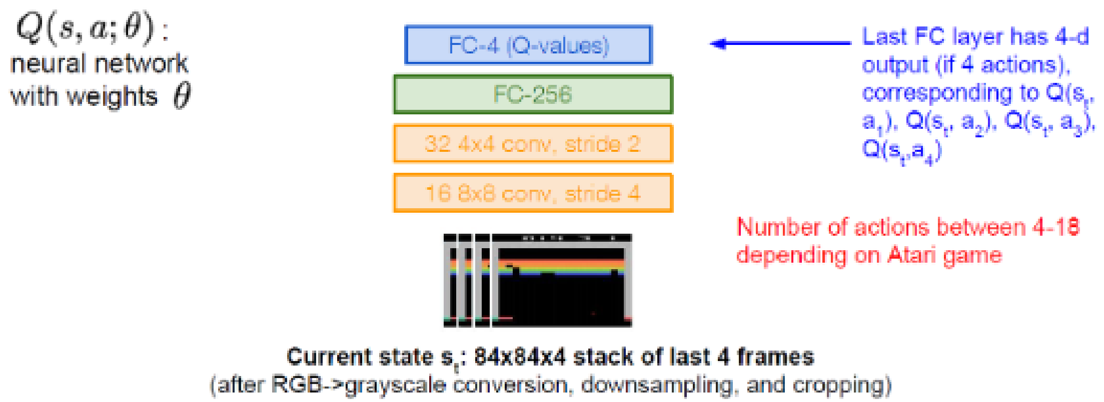
Backward Pass

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

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

- Neural Network로 Q(s,a)를 근시시키는 방법 = Deep Q-Learning

Q-learning : Atari Games



- 게임 화면 84*84->4프레임정도 누적시켜 넣음
- 출력은 입력이 들어왔을때 각 행동의 Q-value, 위,아래,왼,오
- 한번의 forward pass만으로 모든 함수에 대한 Q-value값 계산

Q-learning : Atari Games

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

- Training 시 Experience Replay:
  Replay Memory table(상태,행동,보상,다음행동)을 만들어 계속 Update함. 연속 시간 샘플x 임의로 샘플링된 샘플

Policy Gradients:
정책(policy)자체를 학습시키는 방법

Policy Gradients

Formally, let's define a class of parametrized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

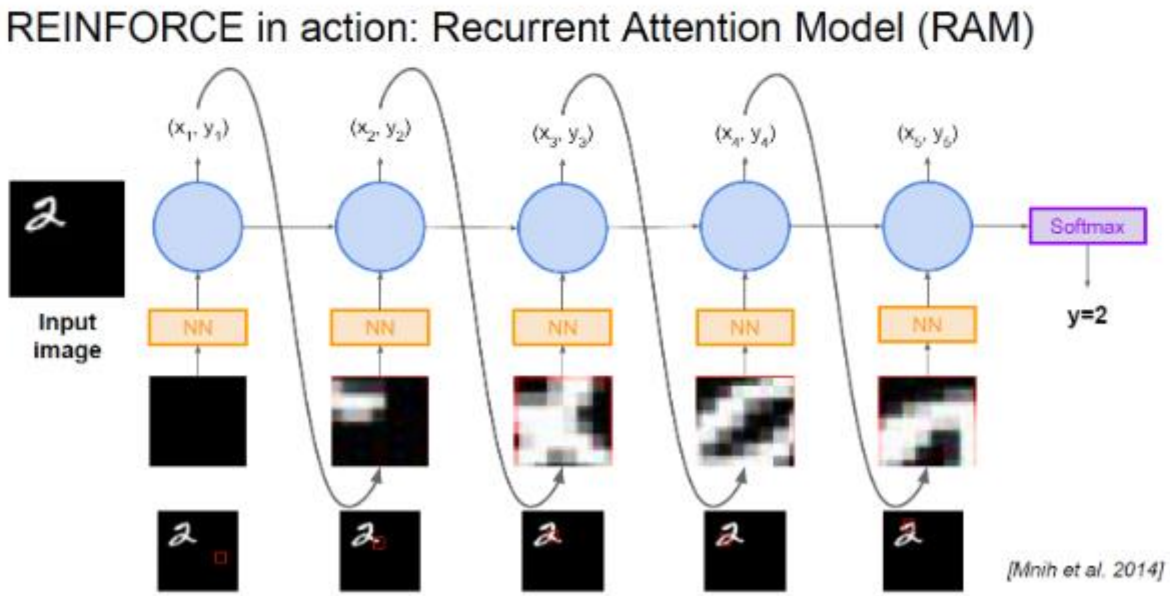
$$J(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta\right]$$

- 구체적인 값(Q-value)없이 정책 자체의 gradient를 구해 최적의 정책을 찾음
- 문제점: 분산이 너무 높음

Policy Gradients:
정책(policy)자체를 학습시키는 방법

- 분산이 너무 높은 문제를 해결하는 방법
  1) 해당 상태로부터 받을 미래 보상만을 고려하여 어떤 행동을 취할 확률을 키워주는 방법
  2) 지연된 보상에 의해서 할인률 적용
  3) Baseline: 현재까지 경험한 보상들에 대해 moving average 값 취함

RAM(Recurrnet Attention Model)



- 강화학습으로 풀기
  State: 지금까지 관찰한 glimpses
  Action: 다음으로 어떤 부분을 볼것인지 결정
  보상: Classification 성공 유무
- State 선택에 RNN 사용