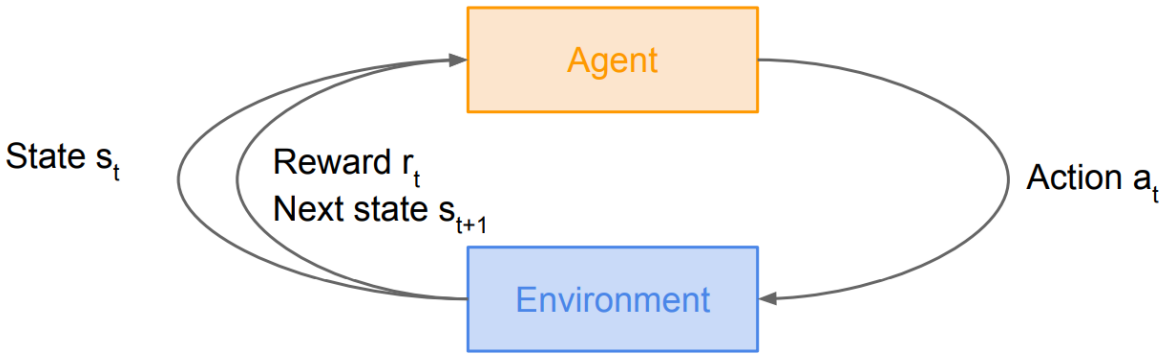
Lec 14. Deep Reinforcement Learning

Reinforcement Learning

- agent: environmnet에서 action하는 주체, action에 따른 rewards 받음
- agent의 rewards를 최대화하는 action을 학습하는 것
- 어떠한 environment에서 agent는 어떤 action을 취함 → environment는 action에 따라 agent에게 reward를 주고 다음 상태 부여 → agent가 terminal state가 될 때까지 반복



- example: Cart-Pole Problem, Robot Locomotion, Atari games, etc.

Markov Decision Process

- Markov property: 현재 상태만으로 전체 상태를 나타내는 성질
- Markov property를 만족, reinforcement problem을 수식화할 수 있음
- 아래는 속성 정의 및 MDP 작동 방식

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

- 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 $r_t \sim R(\cdot \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$
  - 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 최대화 하는 π∗를 찾는 것이 목표 (→ cumulative discounted reward를 최대화)
- ex) Grid World
  - random policy: 어떤 방향으로 움직이든 무작위로 방향 결정
  - optimal policy: 좀 더 terminal state에 가까워지도록 방향 결정
- How do we handle the randomness?
  - A: rewards의 합에 대한 기댓값을 최대화 (아래의 수식 참고)

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

- Value function: 어떤 상태 S와 정책 π가 주어졌을 때 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 행동이 가장 좋은지 알려주는 함수

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

Bellman equation

- reinforcement learning에서 중요한 요소 중 하나
- 현재 state의 value function과 next state의 value function 사이의 관계식

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

- Q*는 어떤 행동을 취했을 때 미래에 받을 보상의 최대치. 최적의 policy를 구하는 방법은?
- → "Value iteration algorithm"
  - 반복적인 update에 Bellman equation 이용
  - BUT, not scalable
  - sol) Q(s, a)를 computable하게 근사시키기(→ Neural Network 이용)

Q-Learning

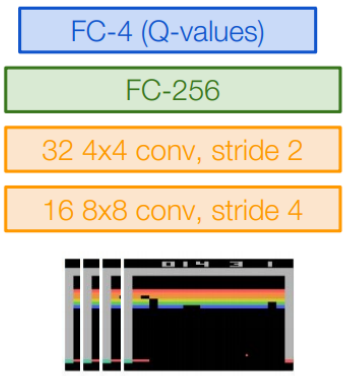
- NN을 이용해 Q(s, a)를 근사시키는 것
- action value function 이용

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

function parameters (weights)

- Deep Q-learning: deep NN 이용한 것
- NN으로 근사시킨 Q-function을 학습시켜서 Bellman equation의 error가 최소가 되도록 함
  - forward pass: loss function 계산 (loss가 최소가 되면 좋음)
  - backward pass: loss를 기반으로 θ update
  - 반복적 update를 통해 Q-function이 target과 가까워지도록 학습
- ex) Playing Atari Games
  - Q-network Architecture

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



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

    - input: 여러 전처리 과정을 통해 84×84 크기로 만들어 4 frames 누적
    - output: input이 들어왔을 때, 각 행동의 Q-value값
      - 여기서는 위, 아래, 왼쪽, 오른쪽 네 방향으로, 4개의 벡터
    - 장점: 한 번의 forward pass만으로 모든 함수에 대한 Q-value 계산 가능
- experience replay: Q-network에서 발생할 수 있는 문제를 다룸
  - Q-network를 학습시킬 때 하나의 배치에서 연속적인 sample로 train하는 것은 bad
    - correlated samples ← ineffective to train
    - bad feedback loops
  - sol> "Experience Replay": replay memory 이용: correlation problem 해결, 데이터 효율 높아짐

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

Policy Gradients

- Q-learning에서 Q-function이 매우 복잡하다는 단점이 있었음
- policy 자체를 학습시킬 수 있냐는 의문에서 시작된 접근 방식

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

For each policy, define its value:

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

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

- policy parameter에 대해 gradient ascent 수행

REINFORCE algorithm

REINFORCE algorithm

Expected reward: $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)] = \int_\tau r(\tau)p(\tau;\theta)\mathrm{d}\tau$

Now let's differentiate this: $\nabla_\theta J(\theta) = \int_\tau r(\tau)\nabla_\theta p(\tau;\theta)\mathrm{d}\tau$ Intractable! Gradient of an expectation is problematic when p depends on θ

However, we can use a nice trick: $\nabla_\theta p(\tau;\theta) = p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$

If we inject this back:

$$\nabla_\theta J(\theta) = \int_\tau (r(\tau)\nabla_\theta \log p(\tau;\theta))\, p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)\nabla_\theta \log p(\tau;\theta)]$$

Can estimate with Monte Carlo sampling

- 경로에 대한 feature reward를 기댓값으로 나타냄
- 경로는 어떤 policy에 따라 설정이 됨
- Gradient ascent를 이용하기 위해 미분 → 계산 불가능한 식 → 트릭을 이용해 식 변형 → Monte Carlo sampling에 의해 경로를 샘플링할 수 있게 됨
- 구체적인 Q-value를 모르더라도 policy 자체의 gradient를 구해 최적의 policy를 찾을 수 있음
- 분산이 높다는 문제 발생 → 분산을 줄이는 것이 policy gradient에서 가장 중요한 것 중 하나
    1. 해당 state로부터 받을 미래 보상만을 고려하여 어떤 행동을 취할 확률을 높임
    2. 지연된 보상에 의해 할인율 적용
    3. Baseline: 얻은 rewards가 예상했던 것보다 좋은지 판단하는 것
        1. baseline function - 해당 state에서 얼마만큼의 보상을 원하는지 알려주는 함수
        2. 정의 방법 중 가장 간단한 것은 지금까지의 rewards에 대해 moving average 값을 취함
        3. 혹은 value function을 baseline에 적용 → Q-function과 Value function을 조합해 training 가능 → "Actor-Critic Algorithm"

Actor-Critic Algorithm

- Actor: policy; 어떤 state를 결정할지 정해줌
- Critic: Q-function; action을 판단하고 조절 방향을 알려줌

- Advantage function: action이 예상보다 얼마나 큰 rewards를 가져왔는지 판단함
- 기존에는 모든 조합에 대해 training했지만, 여기서는 정해준 조합에 대해서만 train

Initialize policy parameters θ, critic parameters ϕ
For iteration=1, 2 … **do**
Sample m trajectories under the current policy
$\Delta\theta \leftarrow 0$
For i=1, …, m **do**
For t=1, … , T **do**

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$$

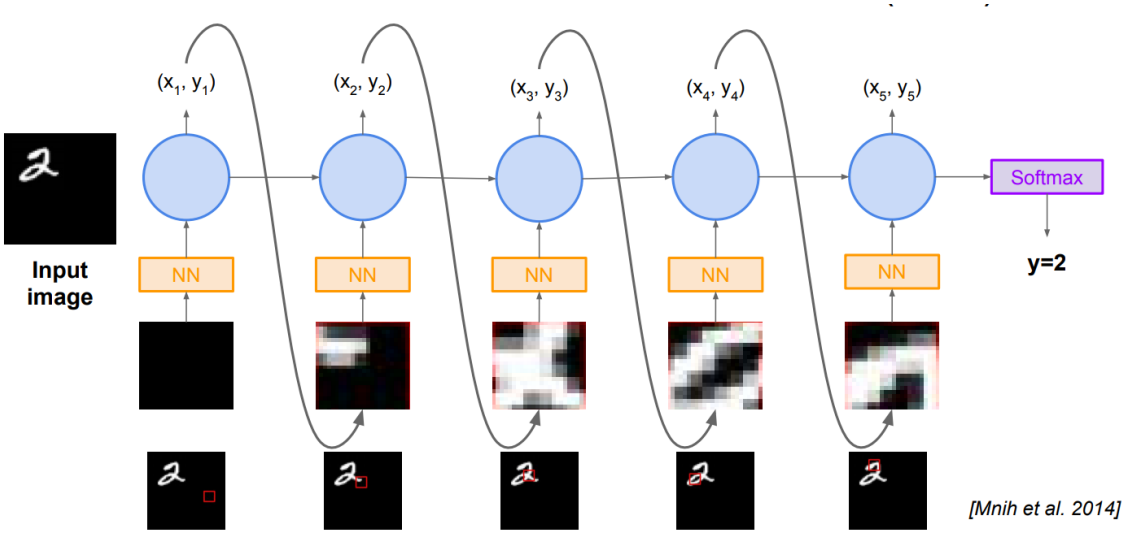
$$\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$$

$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$$

$\theta \leftarrow \alpha\Delta\theta$
$\phi \leftarrow \beta\Delta\phi$
End for

- Recurrent Attention Model
  - image classification과 연관
  - hard attention과도 관련
    - image의 glimpses만 가지고 classify
    - 이미지 전체가 아닌 지역적 부분만을 보는 것이며, 선택적으로 집중할 수 있음
      - 인간의 안구 운동에 따른 지각 능력과 연관
    - computational resources 절약 가능
    - classification 성능을 높임
  - state: glimpses

- action: image 내에서 어떤 부분을 볼 것인지 선택
- reward: 옳으면 1, 그렇지 않으면 0
- reinforcement learning이 필요한 이유: image에서 glimpses를 뽑아내는 것은 미분 불가능한 연산임
- state modeling에 RNN 이용



- gradient 구하는 법은 기존과 동일. 경로 sampling, compute gradient, backprop

- AlphaGo
  - 바둑 두는 agent
  - 지도학습, 강화학습이 섞인 모델
  - Monte Carlo Tree Search, deep RL 방법도 섞임
  - input: 바둑판, 바둑 돌의 위치
  - train (initialize AlphaGo): 프로 바둑기사의 기보를 지도학습으로 학습하여 초기화 → 바둑판의 상태를 보고 다른 행동을 어떻게 취할지 결정, 프로 바둑기사가 두는 수를 supervised mapping으로 학습
  - initialize policy network: 바둑판의 상태를 input으로 받아 output으로 action을 냄. policy gradients를 이용해 지속적으로 학습시킴
  - train value network