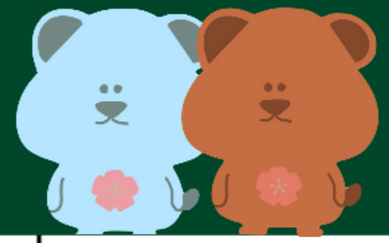
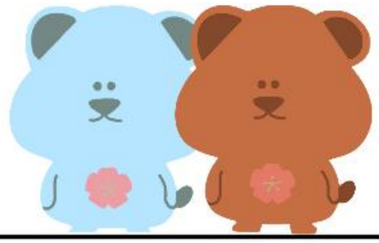


Playing Atari with Deep Reinforcement Learning

고급심화 차수빈

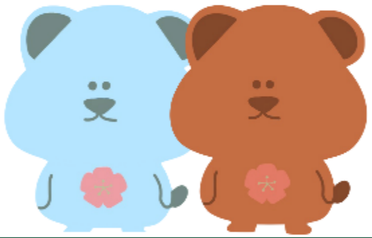
EWHA EURON

Contents



EWHA EURON

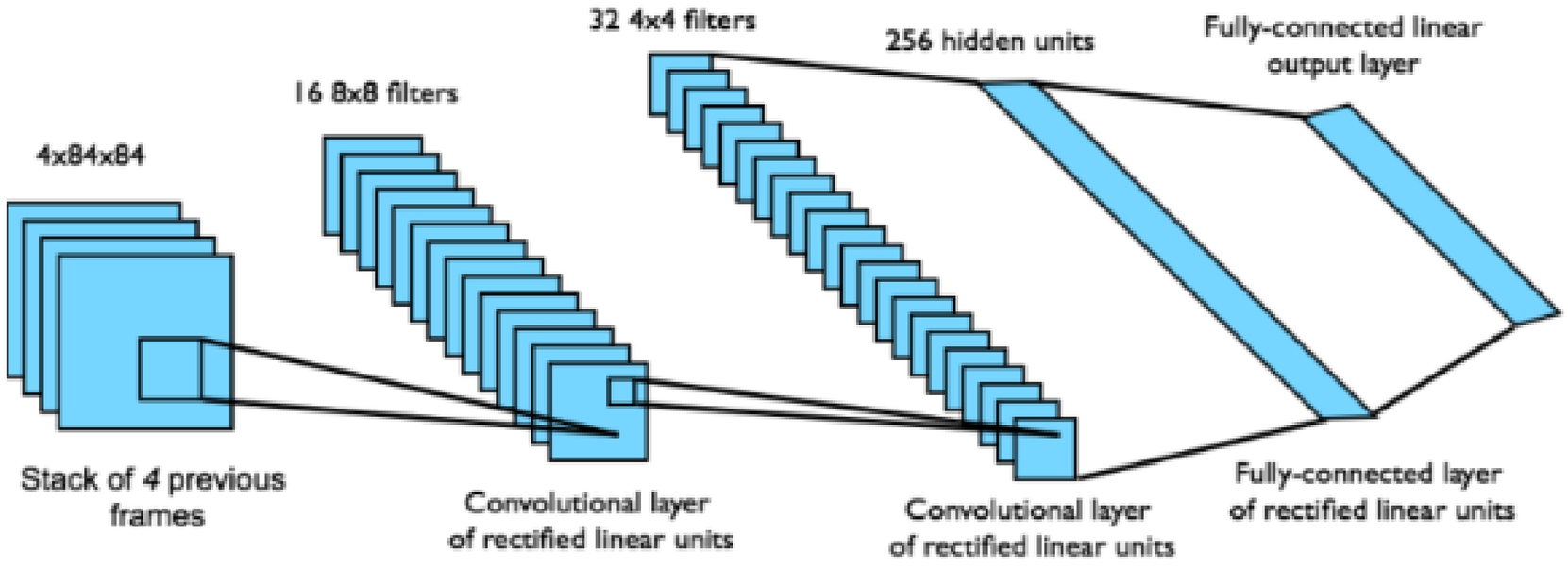
1. Backgrounds



EWHA EURON

#0. Abstract

- High-Dimensional Sensory Input에 대해 강화학습을 사용하여 Control Policy를 성공적으로 학습하는 DL Model을 선보임
- CNN 모델을 사용하고, 변형된 Q-Learning을 사용하여 Atari 게임을 수행해 나가는 방법을 학습
- 2600개가 넘는 다양한 게임을 학습시키는데 동일한 Model과 동일한 Learning Algorithm을 사용

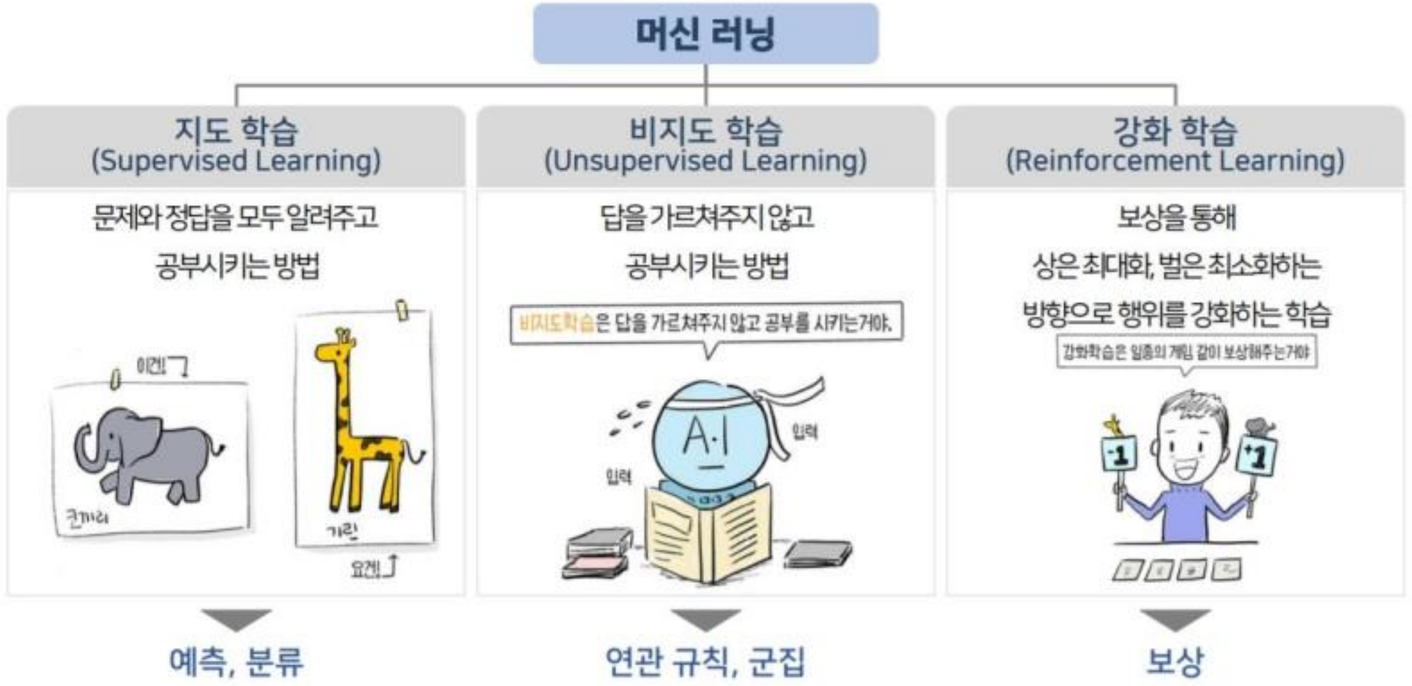


#1. Backgrounds

- **강화학습**

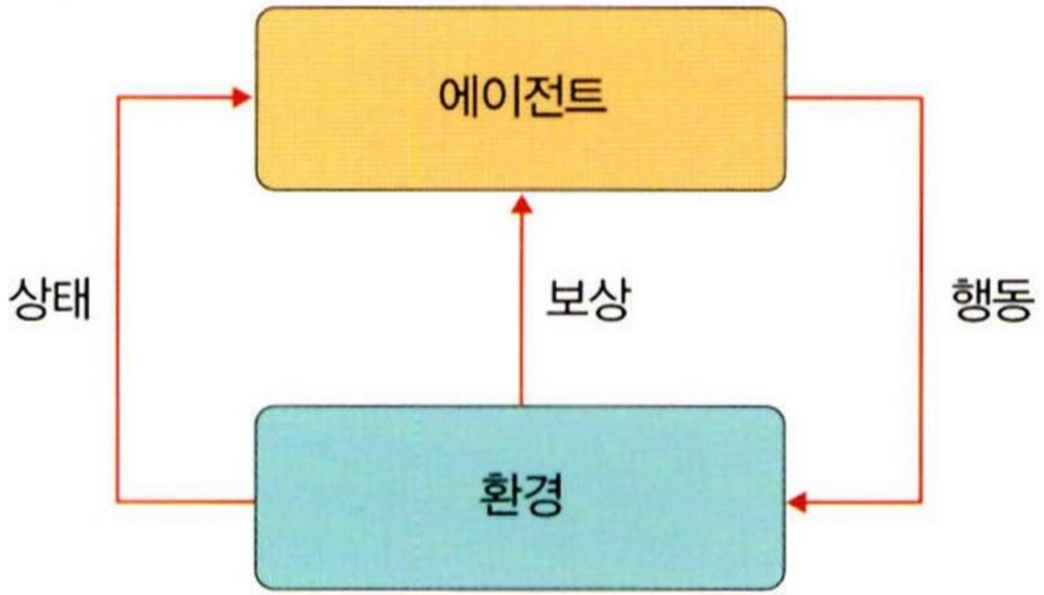
  환경과 상호 작용하는 에이전트를 학습시키자.

  - 어떤 환경에서 어떤 행동을 했을 때 그것이 잘된 행동인지 잘못된 행동인지를 판단하고 보상(or 벌칙)을 주는 과정을 반복해서 스스로 학습하게 하는 머신러닝 분야
  - **목표)** 환경과 상호 작용하는 에이전트를 학습시키는 것
    → 학습을 통해 누적 보상을 최대화하는 policy를 탐색



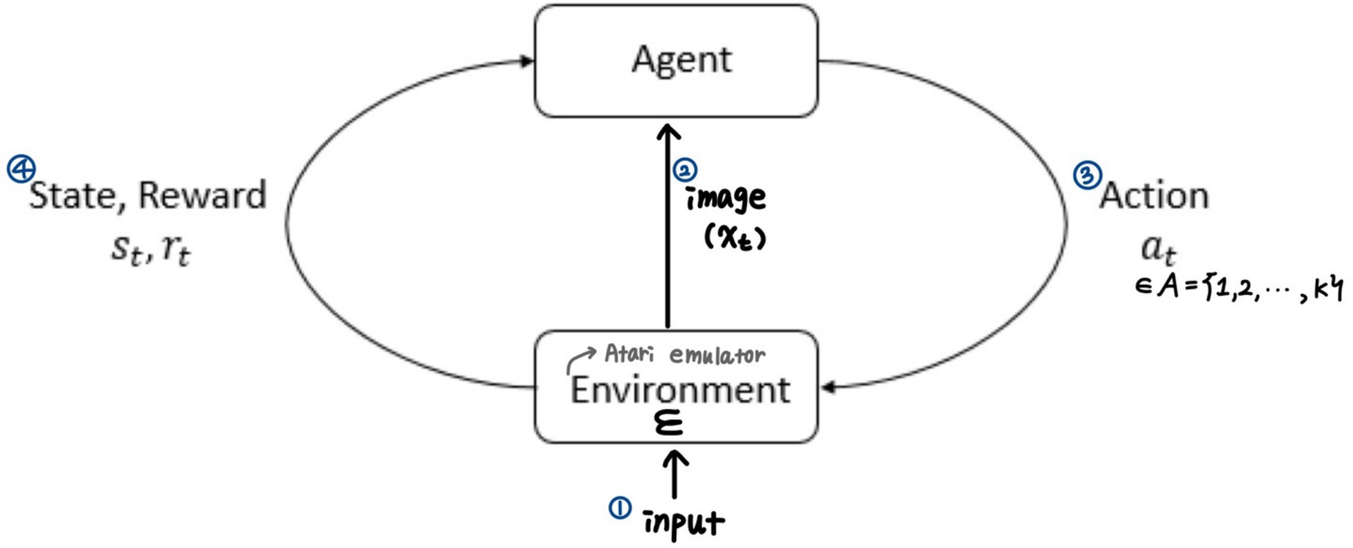
#1. Backgrounds

- **용어정리**
  - 환경(environment, ε)
    - 에이전트가 다양한 행동을 해보고, 그에 따른 결과를 관측할 수 있는 시뮬레이터
  - 에이전트(agent)
    - 상태(state)라고 하는 다양한 상황 안에서 행동(action)을 취하며 조금씩 학습해 나가는 주체
    - 행동에 대한 응답으로 양(+)이나 음(-) 또는 0의 보상(reward)를 돌려받음
  - 상태(state, s_t)
    - 에이전트가 관찰할 수 있는 상태의 집합
      $S_t = s\{s \in S\}, S: set\ of\ all\ states$
    - 시간에 따라 변화
  - 행동(action, a_t)
    - 에이전트가 상태 s_t에서 취할 수 있는 동작
      $A_t = a\{a \in A\}, A: set\ of\ all\ actions$
  - 보상(reward, r_t)
    - 에이전트의 행동에 대한 피드백
    - 리턴(return): 각 상태에서의 보상에 대한 총합
  - 정책(policy, π)
    - 에이전트가 특정 state에서 어떤 action을 취할 지 결정하게 하는 규칙
    - 각각의 상태마다 행동 분포(행동이 선택될 확률)를 표현하는 함수
      $\pi(a|s) = P(A_t = a|S_t = s), s: state, a: action$



#1. Backgrounds

- **강화학습의 기본 매커니즘**
  - 입력(image, Atari emulator의 내부 사정에 대한 간접적 정보)을 바탕으로 agent는 각 time step(t)마다 할 수 있는 행동들($A = 1, \cdots, K$) 중 한 가지를 선택하여 행함(a_t)
  - 이후 행동(a_t)에 대한 보상(r_t)를 받고 상태를 갱신(s_t)
  - agent가 오직 현재의 장면만을 관찰하면 전체적인 상황을 이해하기 어려움
    → action을 sequence($s_t = x_1, a_1, x_2, a_2, \cdots, a_{t-1}, x_t$)로 관찰하고 이를 통해 학습을 진행
  - 미래 보상을 최대화하는 행동을 선택하도록 학습을 진행해 나감
  - 강화 학습의 문제들은 마르코프 결정 과정으로 표현됨
    → 마르코프 결정 과정에 학습 개념을 추가한 것

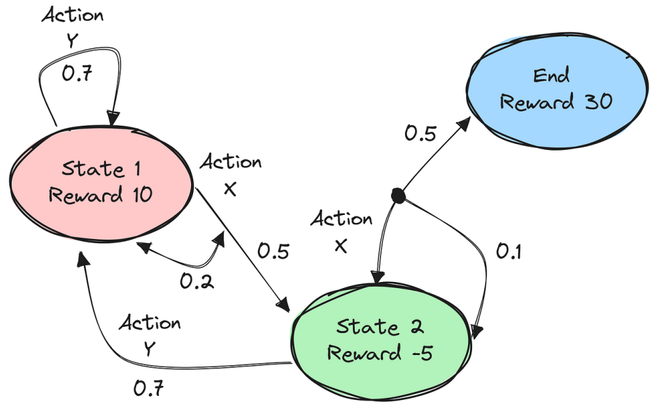


#1. Backgrounds

- **마르코프 결정 과정**
  - **마르코프 프로세스(MP, = Markov Chain)**
    - 어떤 상태가 일정한 간격으로 변하고, 다음 상태는 현재 상태에만 의존하는 확률적 변화 상태
    - 과거에 무슨 일이 일어났는지에 대해서는 전혀 관심 x

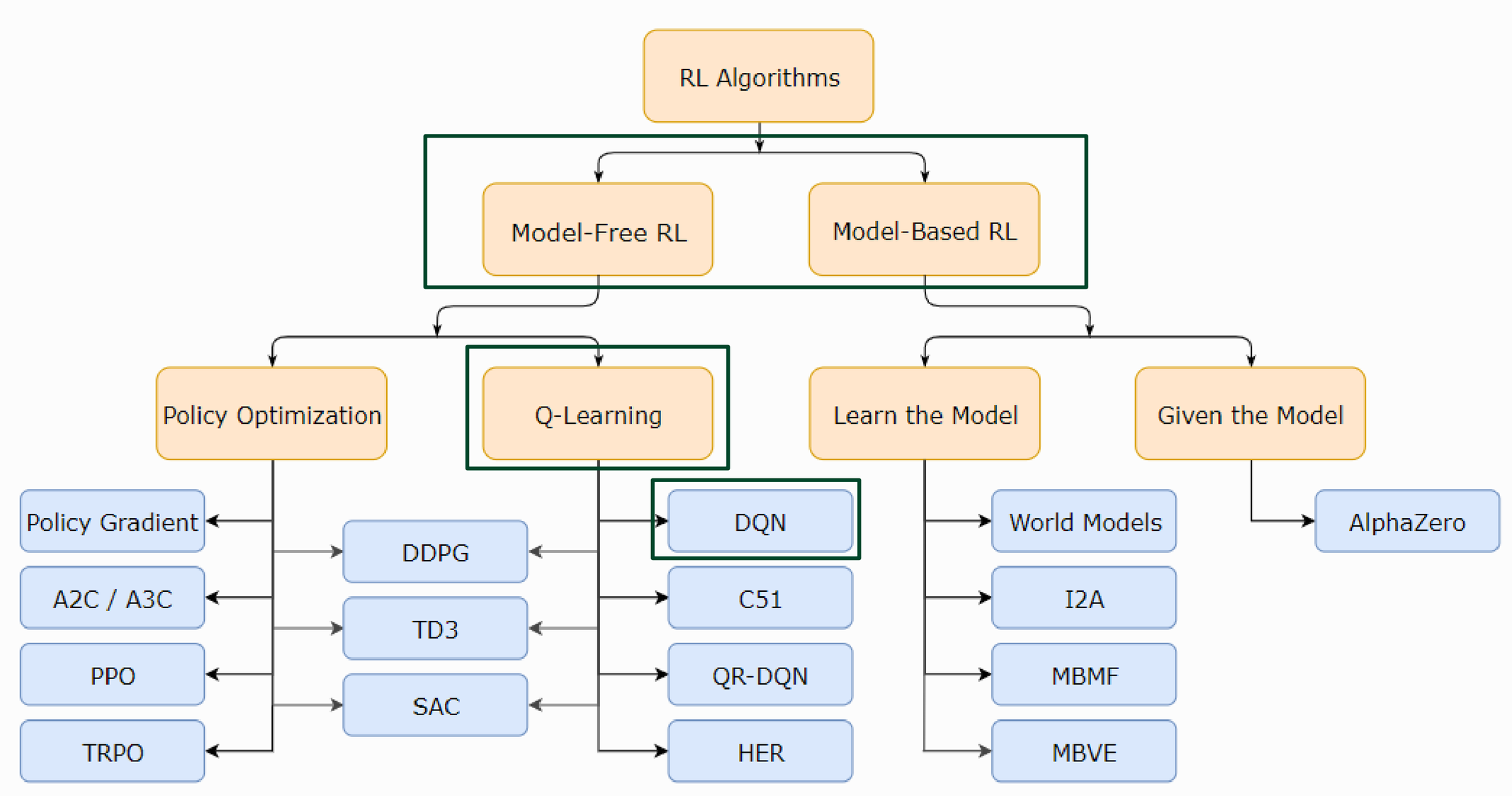
$$P(S_{t+1} = j|S_0 = s_0, S_1 = s_1, \cdots, S_{t-1} = s_{t-1}, S_t = i) = P(S_{t+1} = j|S_t = i)$$

  - **마르코프 보상 과정(MRP)**
    - 마르코프 과정 + **보상**(reward, 좋고 나쁨)
    - 이동 결과의 좋고 나쁨에 대해 보상(혹은 벌칙)을 주는 것
  - **마르코프 결정 과정(MDP)**
    - 마르코프 보상 과정 + **행동**
    - 정의된 문제에 대해 각 상태마다 전체적인 보상을 최대화하는 행동이 무엇인지 결정하는 것



#1. Backgrounds

- RL 구조도



#1. Backgrounds

- **Model-Based vs Model-Free**

Model-Based

- MDP에 대한 정보가 있을 때, 즉 r_s^a와 $P_{ss'}^a$를 알 때
- 환경에 대해 알고 있으며, 행동에 따른 환경의 변화를 아는 알고리즘
- 어떤 state에서 어떤 action이 최고의 reward를 주는지 이미 알 때
- 환경이 어떻게 동작하는지 알기에 exploration이 필요 x

Model-Free

- MDP에 대한 정보가 없을 때, 즉 r_s^a와 $P_{ss'}^a$를 모를 때
- 환경에 대해 알지 못하고, 환경이 알려주는 다음 단계와 보상을 수동적으로 얻게 됨
- 환경이 어떻게 동작하는지 모르기 때문에 exploration(탐사)를 해야 함
- 이러한 과정을 통해 가치를 최대화하는 정책(policy) 함수를 구현하고자 함

#1. Backgrounds

- **Policy-Based vs Value-Based**

Policy-Based

"Optimal Policy를 구해서 행동하자"

- 정책(= 주어진 환경에서 에이전트가 행동을 선택하는 방법)을 직접적으로 최적화
  - ㄴ 보상을 최대화하기 위해 에이전트의 정책(행동 선택 규칙)을 조정하는 것에 초점
- 보통 정책은 신경망 형태로 표현되며, 보상을 최대화하는 방향으로 정책을 업데이트하기 위해 보통의 최적화 알고리즘(ex. 확률적 경사 하강법)을 사용

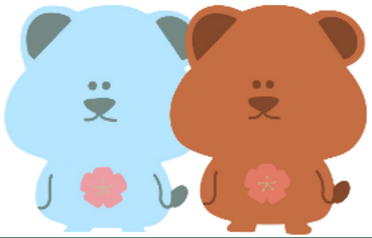
Value-Based

"Value를 따라 행동하자"

- 상태-행동 쌍의 가치를 통해 정책을 간접적으로 최적화
  - ㄴ 각 상태에서의 최적 행동을 학습
    → 이를 통해 최적 정책 학습
- 각 상태에서 가능한 행동들의 가치를 나타내는 Q 함수를 학습
  - ㄴ 에이전트는 환경에서 행동을 취하고, 그에 대한 보상과 다음 상태에서의 최적 행동에 대한 Q 값을 이용하여 Q 함수를 업데이트

  ⇒ 탐험과 활용 사이의 균형을 맞춰나감

2. Introduction



EWHA EURON

#2. Introduction

- **Deep Learning & RL**

- 딥러닝이 발전함에 따라 Vision, Speech와 같은 고차원의 데이터들을 추출하는 것이 가능해졌음
- 딥러닝에서는 이러한 고차원의 데이터들을 입력으로 사용하여 CNN, Multi-Layer Perceptrons, restricted Boltzmann machines, recurrent neural networks 등을 통해 지도학습/비지도학습에 사용하였음

❓ 그러면 이러한 데이터를 강화 학습에는 사용을 못할까

→ high-dimensional sensory inputs로부터 agent를 학습시키는 것

- 그러나 딥러닝을 강화학습에 적용하는 과정에서 몇 가지 문제점에 직면함

#2. Introduction

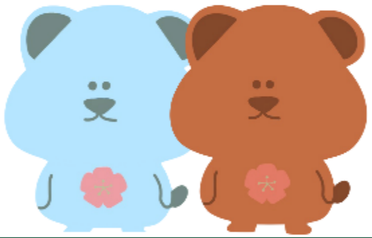
- **Deep Learning vs Reinforcement Learning**
  왜 DL을 RL에 적용하기 어려운가

  - 1. 학습에 대한 접근
    - 둘 다 Autonomous, Self-teaching system
    - DL ⇒ 정답(pattern) 학습 vs RL ⇒ 행동(action) 학습
    - 딥러닝은 input에 대한 결과가 직접 작성되어 계산의 시간이 적지만,
      RL에서는 어떠한 행위를 하면 trial and error를 통해 그 행위에 대한 결과를 알기까지 시간이 필요함
      ⇒ delay 발생
  - 2. 데이터의 독립성
    - DL: 각 데이터들은 독립적임을 가정
    - RL: 하나의 행위는 다른 행위에 영향
      ㄴ 현재 상태의 행동이 다음 상태의 보상에 영향을 미치는 등
    → 데이터 간의 연관성이 높음
  - 3. 데이터 분포의 변화
    - DL: 데이터의 분포가 고정되어 있다고 가정
    - RL: 알고리즘이 새로운 behavior를 배울 때마다 데이터의 분포가 변함

#2. Introduction

- 해당 논문에서는 CNN이 이러한 복잡한 RL 환경에서의 문제점을 개선하고, 원시 비디오로부터 성공적인 control policy를 학습할 수 있음을 증명하였음
  - CNN은 변형된 Q-Learning을 통해 학습되며, weight update에 SGD 사용
  - 또한, correlated data 문제와 non-stationary distribution 문제를 개선하기 위해 experience replay memory를 도입
    - 무작위로 이전의 transition을 추출하여 training distribution을 안정화
  - 하나의 Neural Network를 사용
    - 게임에 대한 특징 정보나 데이터 제공 없이 시각 데이터와 reward, 그리고 터미널로부터 오는 신호와 행동들만으로 학습 진행
- 동일한 Network Architecture와 Hyperparameter를 사용하여 다양한 게임을 학습

3. Q-Learning



EWHA EURON

#3. Q-Learning

- **Q-Learning**
  - 여러 실험(episode)을 반복하여 최적 정책을 학습하는 강화학습의 한 방법
  - 절차

큐-테이블		행동				
		$Action_1$	$Action_2$	···	$Action_{n-1}$	$Action_n$
상태	$State_1$	0	0	···	0	0
	$State_2$	0	0	···	0	0
	···	···	···	···	···	···
	$State_{n-1}$	0	0	···	0	0
	$State_n$	0	0	···	0	0

1. 초기화: Q-table에 있는 모든 큐 값을 0으로 초기화
   - Q-table) 모든 상태(state)와 행동(action)에 대한 기록을 담고 있는 테이블

한 실험에서..

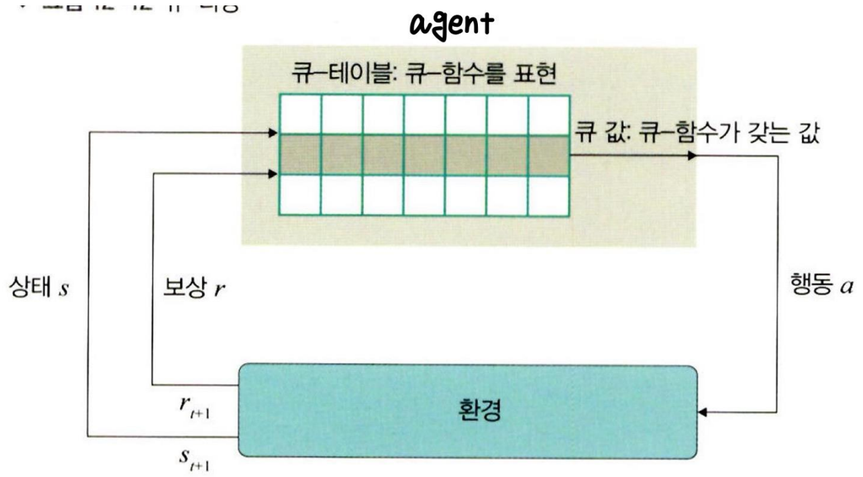
2. 각 상태에서 에이전트는 행동(a)을 선택하고 실행
   - 이때 행동은 임의 선택을 따름 ⇒ 가보지 않은 곳을 탐험(exploration)하며 최적 경로 탐색
   - 탐욕 알고리즘(greedy algorithm) 활용
     ㄴ 0 ~ 1 사이로 랜덤하게 난수를 추출해서 해당 값이 특정 임계치(threshold)보다 낮으면 랜덤하게 행동을 취함
3. 보상(r)과 다음 상태(s')를 관찰
4. 상태(s')에서 가능한 모든 행동에 대해 가장 높은 큐 값을 갖는 행동(a')선택
5. 상태에 대한 큐 값을 업데이트

$$Q_t(s_t, a_t) \leftarrow Q_{t-1}(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_{a'} Q_t(s', a') - Q_{t-1}(s_t, a_t)]$$

   - R_{t+1}: 현재 상태(s)에서 어떤 행동(a)을 취했을 때 얻는 즉각적 보상
   - $\max_{a'} Q_t(s', a')$: 미래에 보상이 가장 클 행동을 했다고 가정하고 얻은 다음 단계의 가치

   ⇒ 목표값(target value): $R_{t+1} + \max_{a'} Q_t(s', a')$

   ⇒ 목표값과 실제 관측값($Q_{t-1}(s_t, a_t)$)의 차이만큼 업데이트 진행
6. 종료 상태에 도달할 때까지 2 ~ 5 반복



#3. Q-Learning

- **Q-Function**
  - 에이전트가 주어진 상태(S_t)에서 행동(a_t)를 취했을 경우 받을 수 있는 보상의 기댓값($E(r_t)$)을 예측하는 함수
- **Q-Function Optimization**
  - 시간이 지날수록 보상의 가치는 점점 감소함 ⇒ 할인율(discount factor, γ) 적용
    - 0에 가까워지면, 미래에 받게 될 보상들이 모두 0으로 근사 → 바로 다음의 보상만 추구(근시안적)
    - 1에 가까울수록 미래 보상들을 더 많이 고려 → 미래 가치에 대해 할인을 고려 x(원시안적)
  - t 시점에서의 discount factor(γ)가 적용된 리턴(R_t)
    $R_t = \sum_{t=t'}^{T} \gamma^{t'-t} r_{t'} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T$
    (T: 게임 종료 시점, $t+1$ 시점부터 discounting factor 적용)
  - 이후 정책(π)을 통해 얻을 수 있는 최대 보상(r_t)의 기댓값으로 최적 action-value function을 재정의
    $Q^*(s,a) = \max_{\pi} E[R_t | s_t = s, a_t = \alpha, \pi]$ (S_t: sequence, a_t: action)
  - 이는 Bellman Equation을 따름

#3. Q-Learning

- **Q-Function Optimization**

  Bellman Equation

  - Idea) sequence(s')의 다음 time-step에서의 최적의 $Q^*(s,a)$ 값이 모든 행동(a')에 대해 알려져 있다면, 최적의 전략은 $r+\gamma Q^*(s',a')$의 기댓값을 최대화하는 것이다.

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

  - 많은 강화학습 알고리즘에서는 Q-function을 추정하기 위해 Bellman Equation을 반복적으로(iterative) 업데이트

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

  - 해당 과정을 무한 번 반복($i \rightarrow \infty$)하여 Q-Function을 최적화($Q_i \rightarrow Q^*$)

  - 그러나 기존의 value iteration algorithm은 각 sequence에 대해 action-value function을 독립적으로 추정
    ⇒ function approximator를 사용하여 action-value function을 적절히 근사하자!

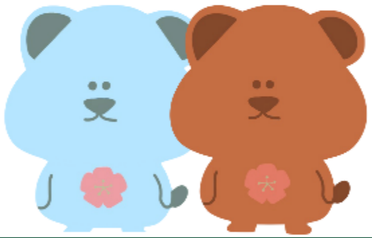
    $$Q(s,a;\theta) \simeq Q^*(s,a)$$

  ❓ 어떤 function approximator를 사용해야 할까

    ⇒ 일반적으로는 linear function을 사용하기는 함

    ⇒ Deep Q-Network: non-linear function approximator

4. Deep RL(& DQN)



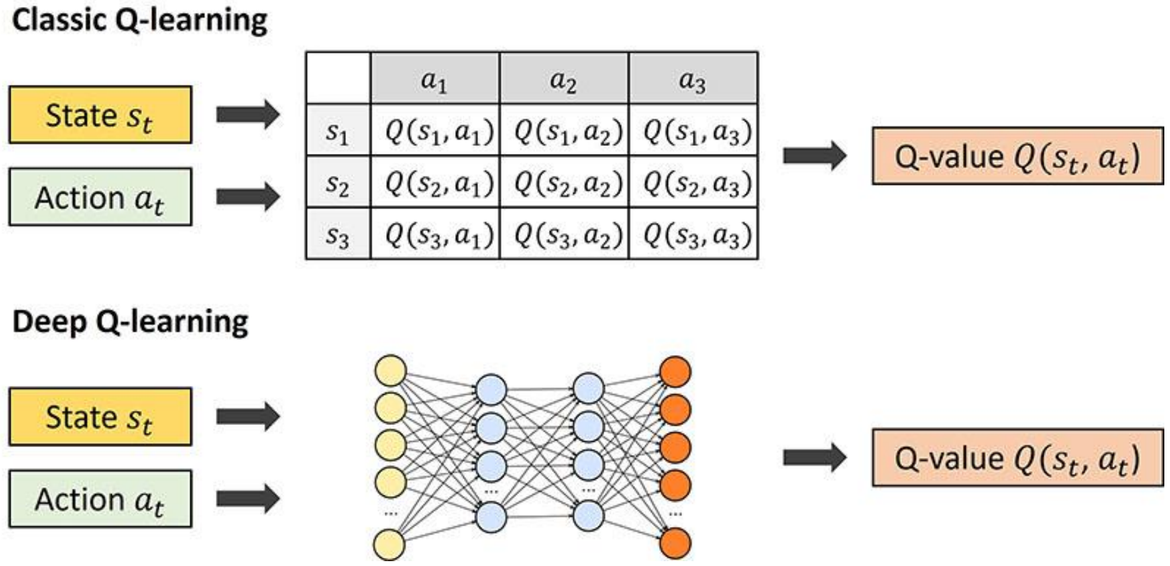
EWHA EURON

#4. Deep RL

- **Why Deep?**
  - 기존 Q-Learning의 한계
    - 에이전트가 취할 수 있는 상태 개수가 많은 경우(= S의 차원이 큰 경우) Q-table 구축에 한계가 있음
    - 데이터 간 높은 상관관계 → 학습 속도 저하
    - 큐-함수가 학습이 진행됨에 따라 계속 변화 → non-stationary targets

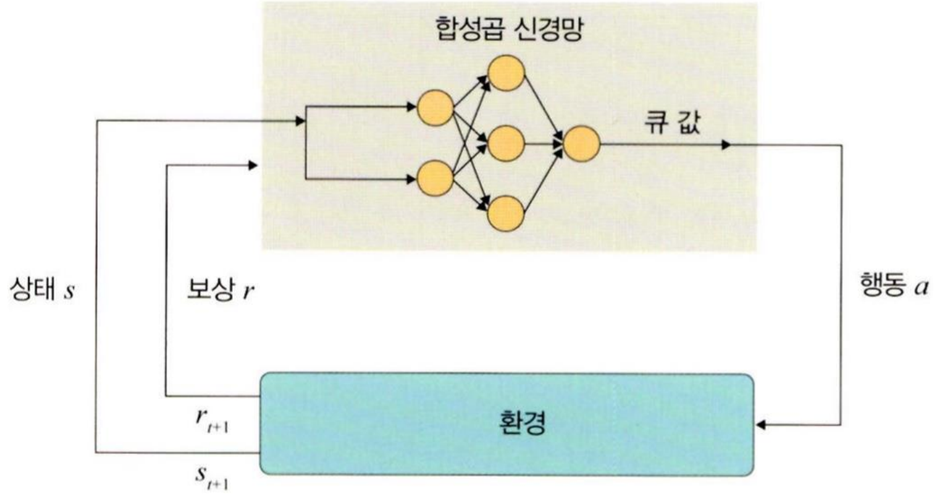
    ⇒ Deep Q-Network에서 개선

  - 해결
    1. 데이터 간 높은 상관관계 → experience replay(+ replay memory)
    2. non-stationary targets → Q-network 분리(Q-network vs target Q-network)



#4. Deep RL

- **Deep Q-Network**
  - Deep Q-Network
    - 합성곱 신경망(CNN)을 이용하여 Q-Function을 학습하는 강화 학습 기법
      → 합성곱 층을 “깊게” 하여 훈련 시 큐 값의 정확도를 높이는 것을 목표로 함
    - idea) Q-function에 활용되는 action-value function을 근사하는 function approximator로 비선형 함수인 Neural Network를 사용해 보자.



- learning policy와 behavior policy가 다른 Off-Policy

On-policy

Target policy와 Behavior policy가 같은 경우

$SARSA$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Off-policy

Target policy와 Behavior policy가 다른 경우

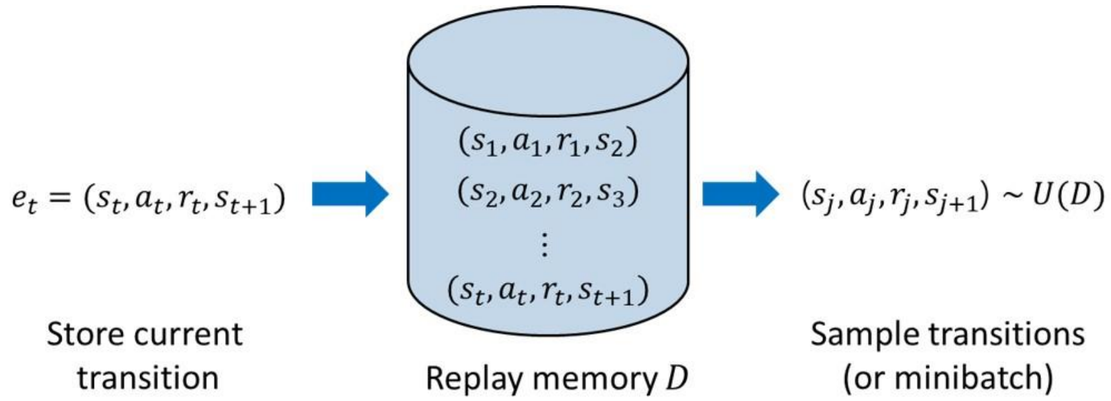
$Q - Learning$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

출처: 고려대학교 DSBA 연구실

#4. Deep Q-Network(DQN)

- **Problem Solution 1) experience replay(replay memory)**
  action에 대한 평가를 의도적으로 지연시켜보자!
  - 기존의 큐-러닝에서는 데이터 간의 상관관계로 인해 학습 속도가 느려지는 문제가 존재하였음
    - DQN에서는 에이전트의 상태가 변경되어도 즉시 훈련하는 것이 아닌 일정 수의 데이터가 수집되는 동안 기다림
    - 이후 일정 수의 데이터가 리플레이 메모리(D)에 쌓이게 되면 랜덤하게 데이터(e)를 추출하여 미니 배치를 통해 학습 진행
  - replay memory
    - agent가 매 time-step마다 했던 experience(e_t)들을 저장해 두는 저장소
    - 하나의 데이터에는 상태, 행동, 보상, 다음 상태가 저장됨



  - sampling 된 e를 바탕으로 e-greedy policy를 통해 action을 선택/수행
    - ϕ 함수를 통해 임의의 e에 따라 다른 임의의 입력 길이를 고정된 길이로 변환하여 사용
  - 데이터 여러 개로 훈련을 수행한 결과들을 '모두' 수렴하여 결과 도출 ⇒ 상관관계 문제 완화

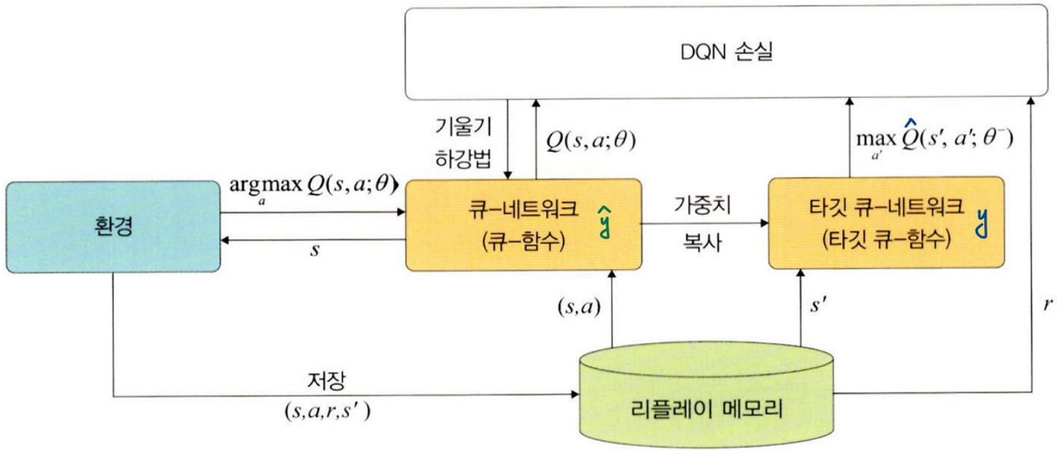
#4. Deep Q-Network(DQN)

- **Problem Solution 2) target Q-Network**
  예측을 위한 Q-network와 정답을 위한 Q-network(target Q-network)를 분리하자!
  - 정답(y)을 저장해 둘 target Q-network를 추가로 활용 → 고정
  - 예측($\hat{y}$)에 Q-network 사용 → optimal Q-value에 가깝도록
  - 원활한 수렴을 위해 target Q-network는 일정한 주기에 따라 한 번씩만 업데이트
    - 가중치 복사
    - every C gradient descent step
  - 손실 함수: MSE($\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$) 활용

$$L_i(\theta_i) = \underset{\text{보상의 기댓값 (expectation)}}{E_{(s,a,r,s')\sim U(D)}}[(\underbrace{r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_i^-)}_{y\text{(target)}} - \underbrace{Q(s, a; \theta_i)}_{\hat{y}\text{(prediction)}})^2]$$

target value　　action value

- $\hat{Q}(s', a'; \theta^-)$: target action-value function, 정답(target) 계산
- $Q(s, a; \theta)$: action-value function, 예측 계산



#4. Deep Q-Network(DQN)

- Algorithm

Algorithm 1 Deep Q-learning with Experience Replay

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

1. Replay Memory D를 크기 N으로 초기화
2. Q(s, a) 를 random weight로 초기화
3. Episode를 1 ~ M까지 반복
4. sequence s_1을 image x_1로 초기화하고, ϕ 함수로 s_1를 전처리하여 ϕ_1을 구함
5. 해당 에피소드에 대해 t = 1 ~ T까지 반복
6. ϵ-greedy 알고리즘에 따라 action을 선택
7. emulator에서 action a_t를 수행하고, reward r_t와 image x_{t+1}을 받음
8. $s_{t+1} = s_t, a_t, x_{t+1}$로 설정하고 전처리하여 ϕ_{t+1}을 구함
9. experience memory(D)에 transition($\phi_t, a_t, r_t, \phi_{t+1}$)을 저장
10. D에 저장된 sample들 중에서 minibatch의 개수만큼 random하게 선택
11. 전처리 결과인 ϕ_{j+1}이 목표 지점에 도달하면 $y_j = r_j$로, 목표지점에 도달하지 못했으면 $y_j = r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta)$로 저장
12. 방정식 3을 따라 gradient descent step을 수행

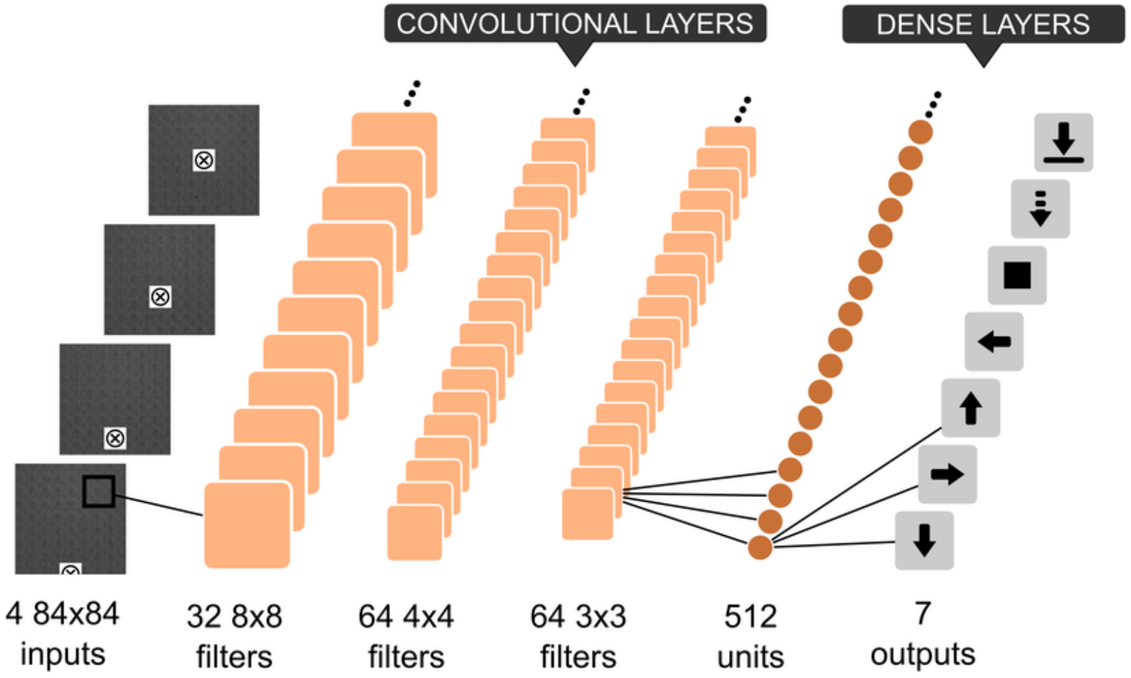
$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right]$$

13. t=T가 되면 반복문 종료
14. episode = M이 되면 반복문 종료

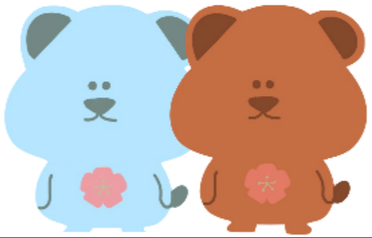
#4. Deep Q-Network(DQN)

- **DNN 구조**

1. Input: ϕ를 통해 전처리된 84 x 84 x 4 이미지(4 frames)
2. 1st Hidden Layer
   - input image에 stride 4를 포함한 16x8x8(16 channels with 8x8 filters)로 합성곱 연산을 수행
   - 이후 rectifier non-linearity(ex. relu) 적용
3. 2nd Hidden Layer
   - stride 2를 포함한 32x4x4(32 channels, 4x4 filters)로 합성곱 연산 수행
   - rectifier non-linearity(ex. relu) 적용
4. 마지막 Hidden Layer
   - fully-connected 됨
   - 256개의 rectifier 유닛으로 구성됨
5. Output layer
   - 각 수행 가능한 행동에 대해 single output을 갖는 fully-connected linear layer



5. Experiments



EWHA EURON

#5. Experiments

- **전처리**
  - Atari Game은 128 color palette의 210x160 pixel 이미지로 구성됨
    - 이를 직접 작업하는 것은 엄청난 계산양을 필요로 하므로 전처리 과정을 적용

1. RGB로 표현된 이미지를 gray-scale의 이미지로 변환
2. 210x160 pixel의 이미지를 110x84 pixel의 이미지로 down-sampling
3. GPU 처리를 위해 110x84픽셀의 이미지를 84x84로 crop

⇒ 마지막 4개의 frame에 대해 전처리를 수행하여 stack에 넣어둠
(4개의 프레임이 1개의 화면을 구성하기에 4개 프레임을 기준으로 처리)

- **Q-Value 계산**
  - 두 가지 방법이 존재
    1. input: history, action → 그에 대해 예측된 Q-Value 계산
       ㄴ 들어온 action에 대해 separate forward pass를 진행해야 함 → action의 수에 따라 연산 비용이 선형적으로 증가
    2. input: history → 모든 행동에 대해 예측된 Q-Value를 계산
       ㄴ 한번의 single forward pass로 처리 가능

⇒ input으로 history만을 받아 계산 → 연산량 감소

#5. Experiments

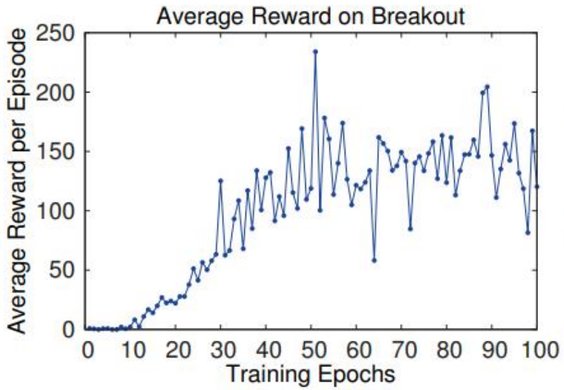
- **Algorithm & Hyper-parameter settings**

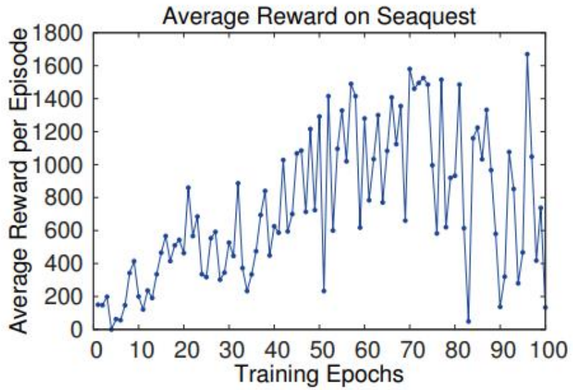
1. Reward Structure: 양의 보상은 1, 음의 보상은 -1, 변화 없음은 0으로 수정
2. RMSProp Algorithm & ε-greedy Algorithm
   - 최적화 알고리즘) batch_size = 32의 mini-batch를 RMSProp에 적용
   - behavior policy) ε 값을 1~100만 번째 프레임까지는 1에서 0.1까지 동일한 비율로 감소시키고, 이후에는 0.1로 고정
3. Frame Skipping Technique
   - agent가 모든 frame을 보고 행동을 취하는 것이 아닌 k번째 프레임을 보고 행동을 고르도록 함
   - 마지막 행동은 skipped된 frames에 반복 적용시켰음
   - Space Invaders를 제외한 모든 게임에서 k=4 로 지정
     - Space Invaders의 경우 게임 내부의 laser blinking 등의 문제 발생으로 인해 k = 3 지정

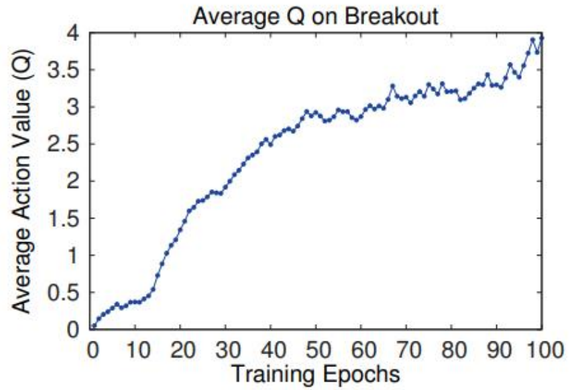
#5. Experiments

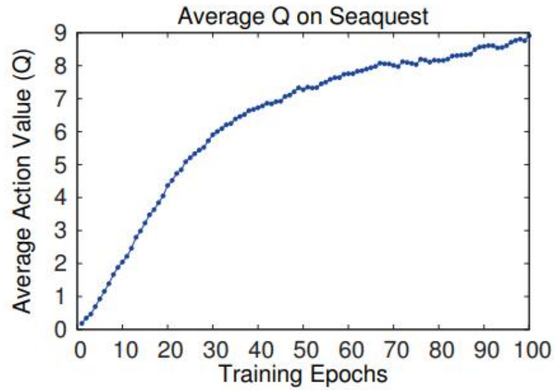
- **Training & Stability**
  - reinforcement learning에서의 모델 성능 측정은 supervised learning에서의 모델 성능 측정보다 어려움
    ⇒ 이를 해결하기 위해 게임에서 얻은 평균 보상을 기반으로 한 평가 척도를 제안
  - 보상 자체는 noise로 인해 정확도가 낮을 수 있음
    - steady한 progress를 만들어내지 못했으며, 상당히 noisy함을 확인할 수 있었음
  - 따라서 해당 논문에서는 policy의 성능을 측정하기 위해 Q-function을 사용하는 방법을 제안
    - 실험 결과, Q값을 사용한 방법이 보상에 비해 훨씬 안정적이며 발산하지 않는 것으로 나타났음

⇒ RL과 SGD를 사용하여 Neural Network를 stable하게 학습시킬 수 있음을 확인









#5. Experiments

- Visualizing the Value Function
  - Point A: Screen의 왼쪽에 enemy가 등장하였을 때, predicted value가 jump
  - Point B: enemy를 발견하여 발사한 미사일이 적을 맞추려고 할 때 predicted value가 상승
  - Point C: Screen에서 enemy가 사라졌을 때 predicted value가 다시 감소

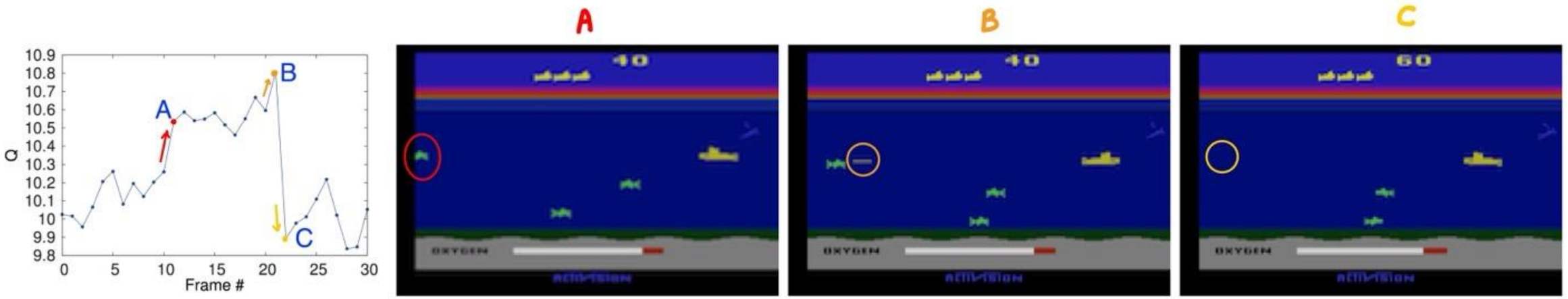


Figure 3: The leftmost plot shows the predicted value function for a 30 frame segment of the game Seaquest. The three screenshots correspond to the frames labeled by A, B, and C respectively.

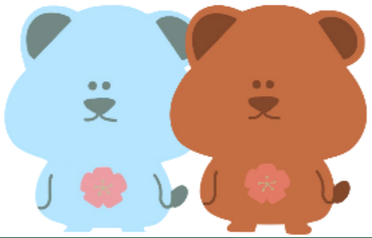
#5. Experiments

- Main Evaluation
  - 사전 지식을 거의 사용하지 않고도 다른 방법들보다 우월한 결과를 보임을 확인
  - 인간이 직접 play한 것과 비교해 보아도 상당한 성능을 보임

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ-greedy policy with $\epsilon = 0.05$.

6. Conclusion



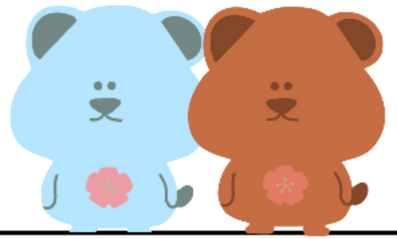
EWHA EURON

#6. Conclusion

- **DQN 장점**
  1. 각 step의 경험이 잠재적으로 많은 weight updates에 재사용됨
     → 경험을 한 번만 사용했던 기존의 방법보다 data efficient 함
  2. high correlation 문제 해결
     → e-greedy algorithm을 통해 데이터를 random하게 추출
     → correlations를 break하고 update 효율을 높임
  3. training distribution 변화 문제 해결
     - 기존의 on-policy 방식을 사용하면 매개변수가 학습된 다음 데이터 샘플을 결정
     - 이전 행동에 dominate 되어 training distribution이 그에 따라 바뀌고, local minimum으로 수렴하는 문제가 발생할 수 있음
     → experience replay를 통해 training distribution이 균형을 이루도록 함
     → 원활한 학습을 도모

- **DQN 단점**
  1. experience replay에는 마지막 N개의 experience만 저장되며, 이는 update를 위해 무작위로 추출됨
     → transition의 중요성에 대한 차별화 없이 유한한 크기의 memory에 overwrite 함
  2. 데이터 sampling 시 우선순위 x
     → 더 좋고 많은 학습을 할 수 있는 transition에 대한 정교한 sampling 전략이 부족함

THANK YOU



EWHA EURON