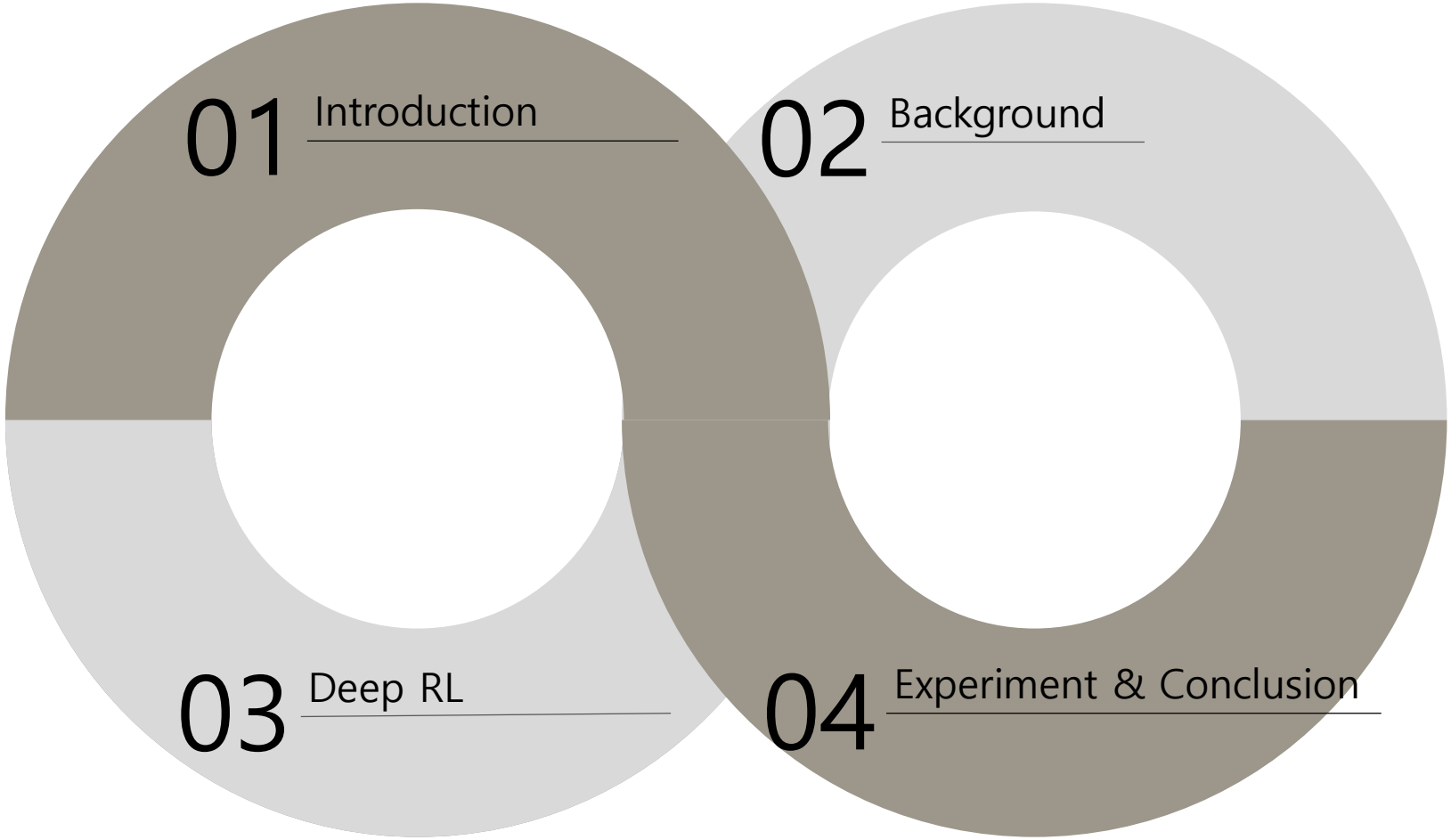
차세대 데이터베이스

- Playing Atari with Deep Reinforcement Learning -

2019/03/26 Tuesday

컴퓨터공학과
201972220 조민규

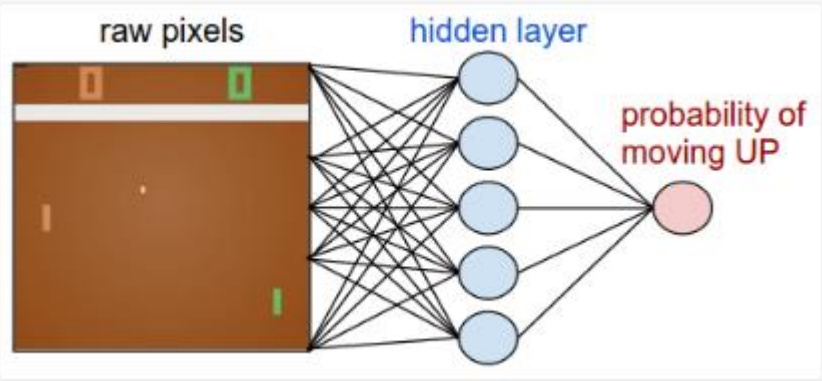
Contents


01 Introduction
02 Background
03 Deep RL
04 Experiment & Conclusion

Part
01

Abstraction

[논문 요약]

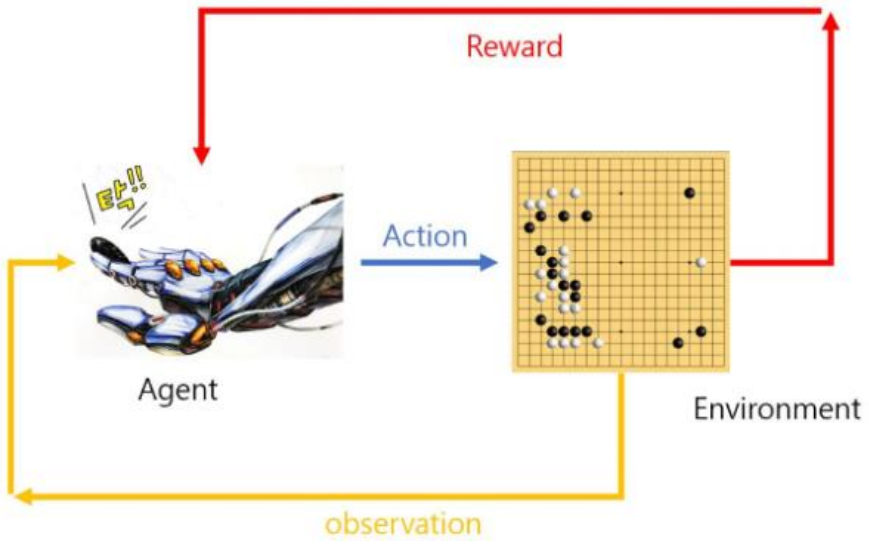


- ✓ High-Dimensional Sensory Input에 대해 RL을 사용하여 Control Policy를 성공적으로 학습하는 DL Model을 선보임
- ✓ CNN 모델을 사용하고, 변형된 Q-Learning을 사용하여 학습함
- ✓ 2600개가 넘는 다양한 게임을 학습시키는데 동일한 Model과 동일한 Learning Algorithm을 사용

Part
01

Base Knowledge

[Q-Learning]

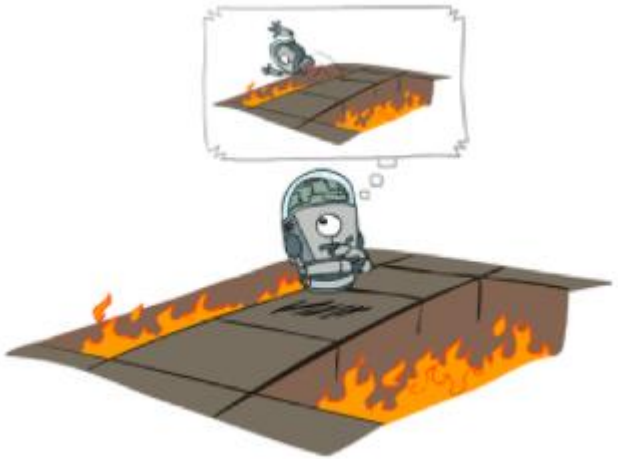


- ✓ Model-Free Reinforcement Learning Algorithm
- ✓ 여러 가지의 서로 다른 행동에 대해 피드백을 받으며 그 행동이 최고의 보상을 주는지 학습
- ✓ Ex) 걷는 법을 배우는 로봇은 넘어짐을 반복하면서 강화
- ✓ Alphago Zero는 RL을 잘 적용한 대표적인 사례

Part
01

Base Knowledge

[Model-Based Learning]



- ✓ Model-based Learning은 Environment에 대해 알고 있으며 Action에 따른 Environment의 변화를 아는 상태
- ✓ 어떤 State에서 어떤 Action이 최고의 Reward를 주는지 알 수 있음
- ✓ Environment에 대해 알고 있으므로 Exploration이 필요 없음

Part
01

Base Knowledge

[Model-Free Learning]

- ✓ Model-Free Learning은 Environment에 대해 모르며 Action에 따른 Next State와 Next Reward를 수동적으로 받음
- ✓ Environment를 모르므로 Exploration(탐험)을 통한 Trial and Error로 Policy Function을 점차 학습시켜야 함
- ✓ 이러한 과정을 통해 Expected sum of future reward를 최대로 하는 Policy Function을 구하고자 함

Part
01

Base Knowledge

[Q-Value Function]

- ✓ 어떤 상태 s에서 어떤 행동 a를 했을 때, 그 행동의 가치를 나타내는 Q(s,a) 함수를 사용
- ✓ 미래의 보상을 계산하기 위해 0~1 사이의 γ(Discount Factor)를 사용
- ✓ 현재 상태로부터 Δt 시간이 흐른 후에 얻는 보상r은 $\gamma^{\Delta t}$만큼 Discounted된 $r * \gamma^{\Delta t'}$
- ✓ Q-Value는 어떤 시간 t에서 전략 π를 따라 행동 a를 할 때 미래의 보상들의 총합의 기대값

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

Part
01

Base Knowledge

[Q-Learning]

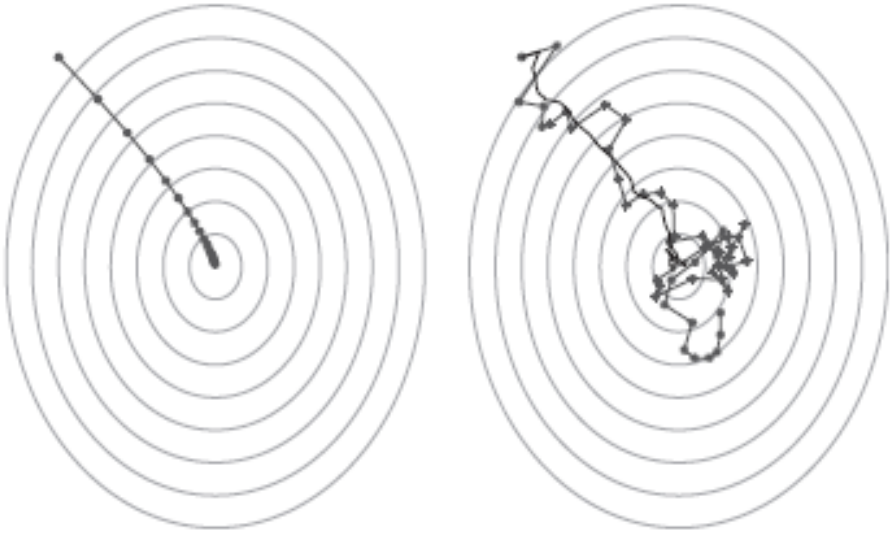
- ✓ FMDP에서 Agent가 Expected sum of future reward를 극대화하도록 Policy를 학습하는 것
- ✓ Q는 고정된 값으로 시작하여, Agent의 Action a_t으로 얻은 Reward r_t를 통해 갱신됨
- ✓ 이전의 값과 새로운 정보의 Weighted Sum을 이용하는 Value Iteration 기법

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Part
01

Base Knowledge

[Stochastic Gradient Descent]



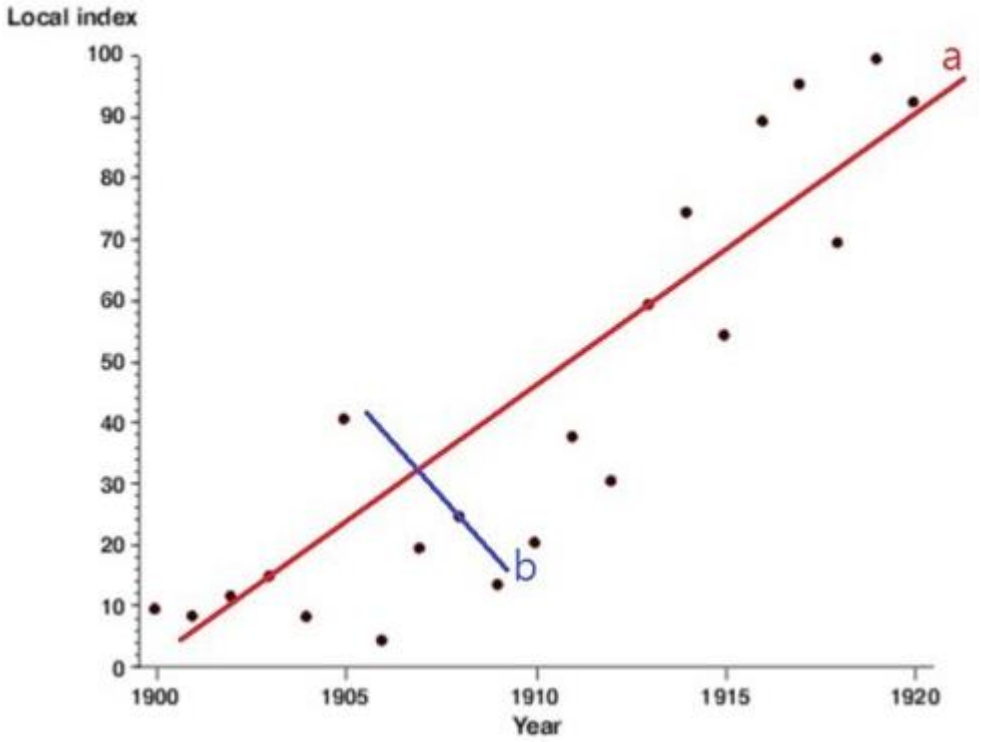
<BGD, SGD>

- ✓ Gradient Descent를 전체 데이터(Batch)가 아닌 일부 데이터의 모음(Mini-Batch)를 사용하는 방법
- ✓ BGD(Batch Gradient Descent)는 하나의 step을 위해 전체 데이터를 계산 하므로 계산량이 많음
- ✓ SGD(Stochastic Gradient Descent)는 Mini-Batch를 사용하여 다소 부정확할 수는 있지만 계산 속도가 훨씬 빠르기 때문에, 같은 시간에 더 많은 Step을 나아갈 수 있음
- ✓ Local Minima에 빠지지 않고 Global Minima에 수렴할 가능성이 더 높음

Part
01

Base Knowledge

[Experience Replay Memory]



- ✓ 전체 데이터의 분포를 보면 a가 가장 정답에 근접한 직선이지만, b근처의 데이터만 보면 b가 정답에 가까운 직선이 됨
- ✓ 고르게 분포된 데이터를 사용해야 정답에 근접한 직선을 찾을 수 있음
- ✓ RL에서는 환경과 상호작용하는 Data가 들어오기 때문에 b와 같은 경우가 빈번히 발생할 수 있음
- ✓ Experience Replay Memory에 데이터(Experience)를 저장한 후 Random하게 뽑아서 학습을 진행

Part
01

Base Knowledge

[Greedy Algorithm]

- ✓ 미래를 생각하지 않고 각 단계에서 가장 최선의 선택을 하는 것
- ✓ 미래를 고려하지 않기 때문에 항상 최선의 결과 반환 X
- ✓ 왼쪽과 같은 결과가 나왔을 때, 다음에 6이 나온 주사위를 선택
- ✓ Exploration이 충분히 이루어지지 않았기 때문에 최상의 결과가 나올 것이라는 확신 X

Part
01

Base Knowledge

[∈ - Greedy Algorithm]

- ✓ 0~1사이의 ∈라는 Hyper-Parameter를 통해 행동 결정
- ✓ 일정한 확률 ∈로는 Random으로 선택
  $1-\epsilon$의 확률로는 최상의 결과를 냈던 행동을 선택
- ✓ ∈ =0.5일 때, 50%의 확률로는 무작위로 주사위를 선택하고,
  50%의 확률로는 6이 나온 주사위를 선택한다.

Part
01

Introduction

[논문 요약]

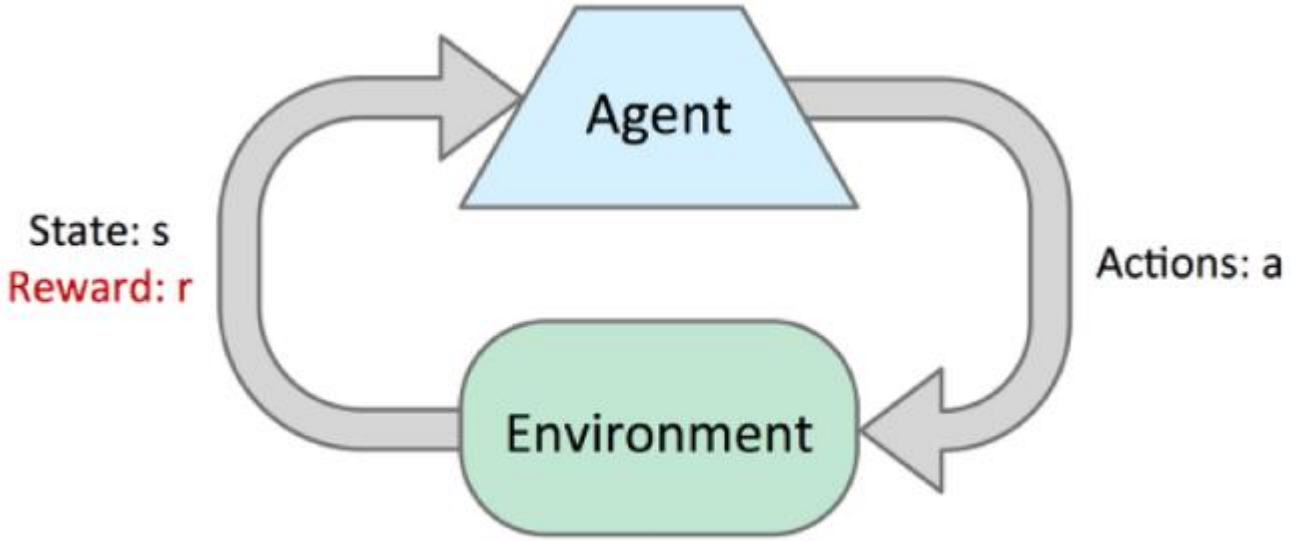
- ✓ CNN이 복잡한 RL 환경에서 원시 비디오로부터 성공적으로 Control Policy를 학습할 수 있음을 증명
- ✓ Variant Q-Learning을 사용하며, weigh를 update하기 위해 SGD(Stochastic Gradient Descent) 사용
- ✓ Correlated Data & Non-stationary distributions의 문제를 약화시키기 위해 Experience Replay Memory 사용
- ✓ 하나의 Neural Network를 사용하였고, 게임에 대한 특정 정보나 데이터를 제공하지 않고
  시각 데이터와 Reward 그리고 터미널로부터 오는 신호, 행동들로 만 학습을 진행
- ✓ 동일한 Network Architecture와 Hyperparameter를 사용하여 다양한 게임의 학습 진행

Part
02

BackGround

[논문 요약]

- ✓ Agent가 time-step마다 할 수 있는 행동들 (A={1, …, K}중에서 한가지(a_t)를 선택
- ✓ 입력(Image)를 바탕으로 어떤 행동(a_t)을 통해 보상 r_t 를 받고 State를 갱신
- ✓ 현재의 행동이 미래에 영향을 받으므로 행동의 sequence $s_t = x_1, a_1, x_2, a_2, \ldots, a_{t-1}, x_t$를 통해 학습 진행
- ✓ Future Reward를 최대화하는 행동을 선택하도록 학습



[그림 3] 강화 학습이 가정하는 세상

Part
02

BackGround

[논문 요약]

- ✓ 시간 t에서 Discount Factor(γ)가 적용된 보상 $r_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (T는 게임의 종료 시점)
  ex) $r_3 = r_3 + \gamma^1 r_4 + \gamma^2 r_5 + \gamma^3 r_6 + \ldots + \gamma^{T-3} r_T$

- ✓ 새로 정의한 action-value function $Q^*(s,a) = max_\pi E[R_t | s_t = s, a_t = a, \pi]$ (s_t는 Sequence, a는 선택한 Action)
  즉, Policy π를 통해 얻을 수 있는 Reward 의 Maximum Expected Value를 구함

- ✓ Bellman Equation에 따라 all possible action a′에 대해 next time-step에서 Optimal $Q^*(s,a)$를 안다면
  Optimal Strategy는 $\mathrm{r} + \gamma * Q^*(s,a)$를 Maximize하는 것이다.

- ✓ RL은 위와 같은 Optimal Strategy를 iterative update하게 사용, $\mathrm{converge}(Q_i \rightarrow Q^*)$ as $(i \rightarrow \infty)$

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

Part
02

BackGround

[논문 요약]

- ✓ But 이러한 Action-Value Function은 각 Sequence에 대해 독립적으로 estimate되므로 비현실적이다.
- ✓ θ를 Weight로 갖는 Non-Linear Network Function Approximator를 통해 Approximate $Q(s, a; \theta) \simeq Q^*(s, a)$.
- ✓ θ를 학습시키기 위해 i번째 iteration에서 다음과 같은 loss function을 갖게 하여 θ가 Converge 할 때 까지 반복시키고, 여기서 y_i는 iteration의 target value이다.

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q(s, a; w_i)\right)^2\right], \quad \text{where, } y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s', a'; w_{i-1}) \mid s, a\right]$$

- ✓ θ_{i-1} 은 $L_i(\theta_i)$ 를 optimizing 할 때 정해진다.

Part
02

BackGround

[논문 요약]

✓ 결과적으로 아래와 같은 Gradient를 얻게 됨

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

✓ Emulation에 대해 고려하지 않은 Model-Free Algorithm

✓ Learning Policy와 Behavior Policy가 다른 Off-Policy(θ vs ϵ)

Part
02

Deep Reinforcement Learning

[논문 요약]

- ✓ Agent가 매 time-step마다 했던 Experience(e_t)들을 Experience Replay에 저장
  $e_t = (s_t,\ a_t, r_t,\ s_{t+1})$ stored in $D = e_1,\ \ldots, e_N$
- ✓ Deep-Q Algorithm에서는 D에서 random하게 sampling한 e를 사용하여 학습을 진행
- ✓ Sampling한 e를 바탕으로 e-greedy Policy를 통해 Action을 선택 및 수행
- ✓ ø 함수를 통해 임의의 e에 따라 다른 Arbitrary Input Length를 Fixed Length로 변환하여 사용

Part
02

Deep Reinforcement Learning

[논문 요약]

Deep-Q Learning 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**

Algorithm 1 Deep-Q Learning with Experience Memory

1. Replay Memory D를 크기 N으로 초기화 한다.
2. Q(s, a) 를 random weight로 초기화한다.
3. Episode를 1~M까지 반복한다.
4. Sequence s_1을 Image x_1으로 초기화하고, ∅함수로를 전처리하여 $\emptyset_1$을 구한다.
5. 해당 에피소드에 대해 t= 1 ~ T까지 반복한다.
6. ∈-greedy 알고리즘에 따라 Action을 선택한다.
7. Emulator에서 action a_t 를 수행하고, reward r_t 와 image x_{t+1} 을 받는다.
8. $s_{t+1} = s_t, a_t, x_{t+1}$로 설정하고 전처리하여 $\emptyset_{t+1}$을 구한다.
9. Experience Memory(D)에 transition($\emptyset_t$, a_t, r_t , $\emptyset_{t+1}$)을 저장한다.
10. D에 저장된 sample들 중에서 minibatch의 개수만큼 random하게 뽑는다.
11. 전처리한 결과인 $\emptyset_{j+1}$ 이 목표 지점에 도달하면 $y_j = r_j$ 로
목표지점에 도달하지 못했으면 $y_j = r_j + \gamma max_{a'} Q(\emptyset_{j+1}, a'; \theta)$로 저장한다.
12. 방정식 3을 따라 gradient descent step을 수행한다.
13. t=T가 되면 반복문을 종료한다.
14. Episode=M이 되면 반복문을 종료한다.

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

<Equation 3>

Part
02

Deep Reinforcement Learning

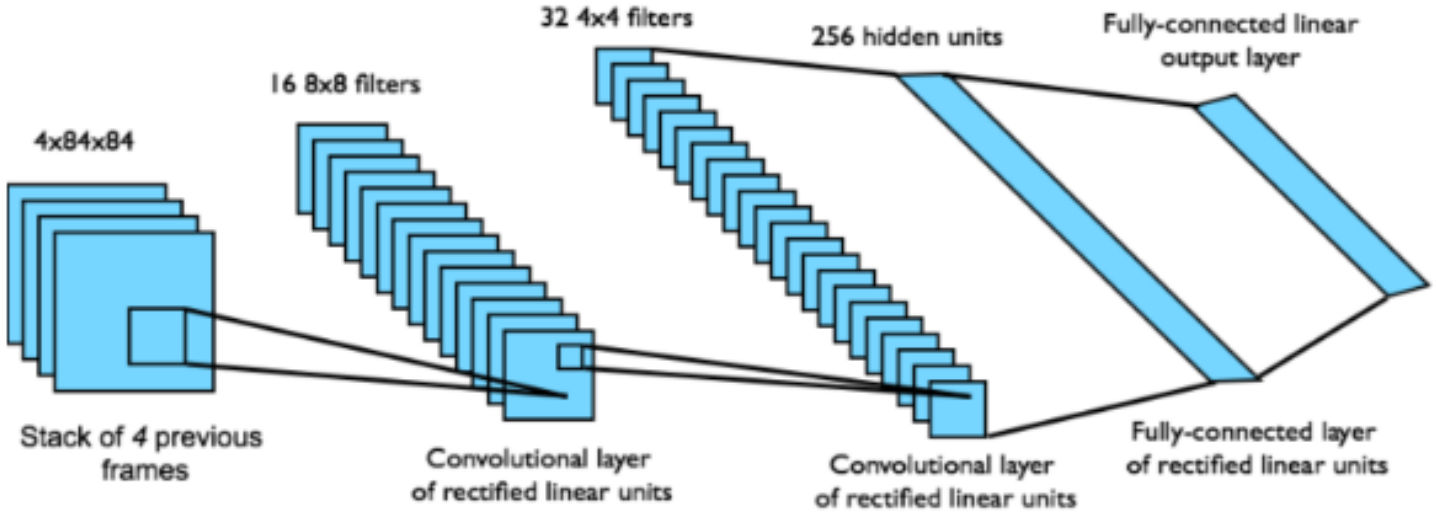
[논문 요약]

Preprocessing

Atari Game은 128 color palette의 210x160 pixel 이미지로 구성된다.
이를 직접 작업하는 것은 엄청난 계산양을 필요로 하므로 Preprocessing 과정을 적용한다.

1. RGB로 표현된 이미지를 Gray-Scale의 이미지로 변환한다.
2. 210x160 pixel의 이미지를 110x84 pixel의 이미지로 down-sampling 시킨다.
3. GPU 처리를 위해 110x84픽셀의 이미지를 84x84로 crop한다.

=> Last 4 frame에 대해 preprocessing하여 stack에 넣어둔다.



Part
02

Deep Reinforcement Learning

[논문 요약]

Model Architecture

Q-Value를 구하는 방법으로는 아래의 두가지가 있다.

1. History와 action을 input으로 하고, 그에 대해 예측된 Q-Value를 구하는 것
2. History만을 input으로 하여 모든 행동에 대해 예측된 Q-Value를 구하는 것

=> 1번을 사용하면 들어온 action에 대해 separate forward pass를 진행해야 하고, 들어온 action에 따라 연산의 양도 Linear하게 늘어난다. 그러므로 2번을 통해 한번의 single forward pass로 처리하여 연산의 양을 줄인다.

Part
02

Deep Reinforcement Learning

[논문 요약]

Advantages of DQN

1. Each Step의 Experience가 potentially 많은 weight updates에 재사용된다.
   →Experience를 한번만 사용했던 기존의 방법보다 훨씬 Data Efficiency하다.

2. 연속적인 sample들로부터 학습을 진행하는 것은 데이터들 간의 high correlations때문에 비효율적이다.
   →e-greedy Algorithm을 통해 데이터를 random하게 추출하여 correlations를 break하고 update 효율을 높인다.

3. 기존의 on-policy를 사용하면 매개변수가 학습된 다음, 데이터 샘플을 결정한다.
   즉, 이전 행동에 dominate 되어 training distribution이 그에 따라 바뀌고, local minimum으로 수렴할 수 있다.
   → Experience Replay를 통해 training distribution이 균형을 이뤄 학습을 원활하게 한다.

Part
02

Deep Reinforcement Learning

[논문 요약]

Disadvantages of DQN

1. Experience Replay에 마지막 N개의 Experience만 저장되며, Update를 위해 무작위로 추출한다.
   →Transition의 중요성에 대한 차별화 없이 Finite 크기의 Memory에 overwrite한다.

2. 마찬가지로 Data를 Sampling 할 때, 우선순위를 두지 않는다.
   →더 좋고 많은 학습을 할 수 있는 Transition에 대한 정교한 Sampling 전략이 부족하다.

Part
02

Experiment

[논문 요약]

1. Reward Structure: 양의 보상은 1, 음의 보상은 -1로 수정

2. RMSProp Algorithm & ∈-greedy Algorithm: 32 크기의 mini-batch를 RMSProp에 적용하였고,
   ∈ 값을 1~100만 번째 프레임까지는 1에서 0.1 까지 동일한 비율로 감소하고, 이후에는 0.1로 고정

3. Frame Skipping Technique: Agent가 모든 Frame을 보고 Action을 취하는 것이 아니라 K번째 프레임을 보고 액션을 고르게 하였고, 마지막 행동은 skipped된 frames에 반복 적용시켰다.

Part
02

Experiment

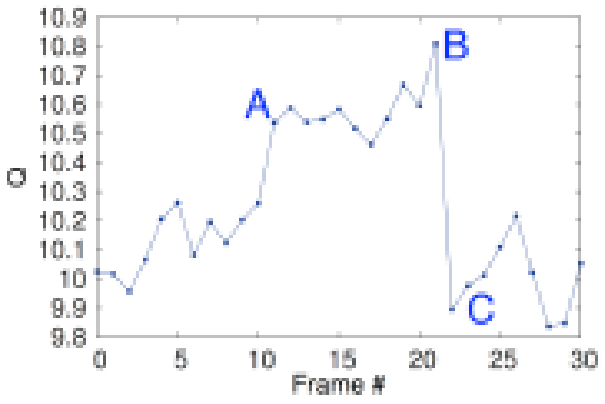
[논문 요약]

Visualizing the Value Function

Point A: Screen의 왼쪽에 enemy가 등장하였을 때, predicted value가 jump함

Point B: enemy를 발견하여 발사한 미사일이 적을 맞추려고 할 때 predicted value가 상승함

Point C: Screen에서 enemy가 사라졌을 때 predicted value가 다시 감소함



Part
02

Experiment

[논문 요약]

Main Evaluation

	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

Part
02

Conclusion

[논문 요약]

- ✓ RL을 위한 새로운 Deep Learning Model을 소개함
- ✓ Raw pixel들 만을 입력으로 사용하여 2600개가 넘는 어려운 Control Policy를 학습
- ✓ 학습을 위해 Stochastic Gradient Descent에 Experience Memory를 적용한 Deep Q-Learning 소개
- ✓ Architecture나 Hyper-parameter의 변화 없이 7개 중 6개의 게임에서 경이적인 결과를 도출

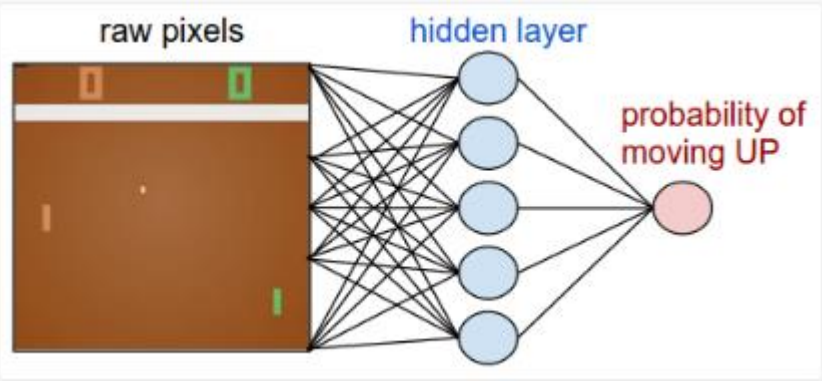
THANK YOU

For your attention

Part
01

Base Knowledge

[Deep Learning]

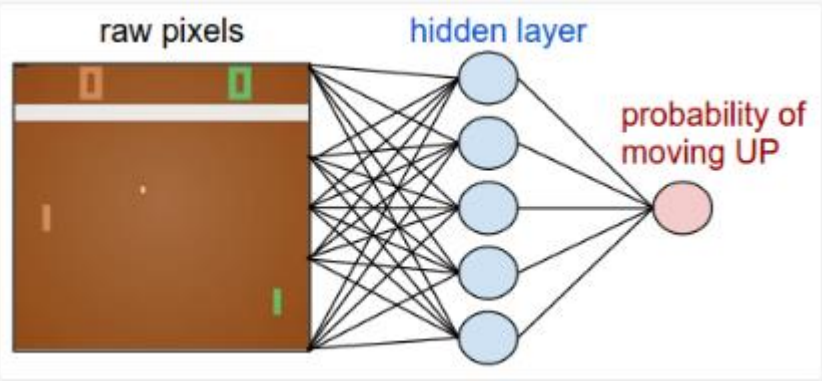


- ✓ Autonomous, Self-teaching System으로 Pattern을 찾기 위한 Algorithm을 학습시키기 위해 존재하는 데이터를 사용
- ✓ 이미지에서 edges, shapes, colors와 같은 특징 데이터를 사용하여 학습
- ✓ Ex) 이미지에서 고양이를 찾는 알고리즘에서 사용 가능
- ✓ Apple의 Face ID는 DL을 잘 적용한 대표적인 사례

Part
01

Base Knowledge

[Reinforcement Learning]



- ✓ Autonomous, Self-teaching System으로 Trial and Error를 통해 학습하여, 보상의 극대화를 통해 최고의 결과를 얻기 위해 학습
- ✓ 여러 가지의 서로 다른 행동에 대해 피드백을 받으며 그 행동이 최고의 보상을 주는지 학습
- ✓ Ex) 걷는 법을 배우는 로봇은 넘어짐을 반복하면서 강화
- ✓ Alphago Zero는 RL을 잘 적용한 대표적인 사례

Part
01

Base Knowledge

[DL vs RL]

	Deep Learning	Reinforcement Learning
공통점	• Autonomous, Self-Teaching System	
차이점	• Training Set으로부터 학습 • 학습을 새로운 데이터에 적용	• 최고의 보상을 위한 행동 선택 • 동적으로 학습하며 행동을 조정