7주차 논문 리뷰: Playing Atari with Deep Reinforcement Learning

0. Abstract

- 딥러닝 모델을 사용하여 고차원 sensory input으로부터 강화학습을 통해 직접적으로 제어 정책을 학습하는 첫 번째 모델을 제시
- **Q-learning**의 변형을 통해 훈련된 합성곱 신경망(CNN)
- 원시 픽셀 데이터를 입력으로 받아 future rewards를 추정하는 value function을 출력으로 함

1. Introuduction

1. 배경 & 문제 정의

- RL(강화학습)에서 시각적 입력(vision)과 음성 입력(speech) 같은 고차원 센서 데이터를 통해 Agent가 직접 제어를 학습하는 것은 오랫동안 어려운 과제로 여겨져옴
- 대부분의 성공적인 강화학습 응용 프로그램: hand-crafted features or linear value functions, policy representations 사용
    - 입력 데이터를 인간이 해석할 수 있는 features로 변환하는데, 그만큼 성능은 이런 특성의 품질에 크게 의존하게됨

2. 딥러닝 & 강화학습

- 딥러닝의 발전으로 인해, 원시 센서 데이터로부터 high-level features을 추출하는 것이 가능해짐→ 이로 인해 컴퓨터 비전과 음성 인식 분야에서 큰 성과

→ 합성곱 신경망(CNN), **다층 퍼셉트론(multilayer perceptrons)**, **restricted Boltzmann machines**, **순환 신경망(RNN)** 등 다양한 신경망 구조를 포함

- 딥러닝의 대부분의 성공적인 응용은 **라벨이 지정된 많은 양의 데이터**가 필요하지만, 강화학습에서는 이와 달리, 수천 번의 time steps 동안 **매우 희소하고 노이즈가 많은 스칼라 보상 신호(scalar reward signal)**를 학습해야 함
- 딥러닝은 일반적으로 **입력과 목표 간의 직접적인 연관성**을 학습하지만, 강화학습에서는 행동과 그로 인한 보상 간의 delay가 매우 길 수 있음
- 딥러닝은 **독립적인 데이터 샘플**을 가정하고 학습하는 반면, 강화학습에서는 **순차적으로 연결된 데이터 샘플**을 다루어야 함

- 이 논문은 **합성곱 신경망(CNN)을 사용하여 복잡한 강화학습 환경에서 원시 비디오 데이터로부터 제어 정책을 성공적으로 학습하는 방법**을 제안
  - Q-러닝의 변형을 사용하여 훈련
  - 강화학습에서 발생하는 **correlated data**와 **비정상적 분포(non-stationary distributions)** 문제를 완화
  - **experience replay**을 사용하여 과거 행동 데이터를 무작위로 샘플링하고, 이를 통해 학습 데이터의 분포를 안정화
  - Arcade Learning Environment (ALE)에서 구현된 다양한 Atari 2600 게임에 이 방법을 적용
    - **Atari 2600:** 고차원 시각적 입력을 제공하는 복잡한 강화학습 testbed(210×160 RGB 비디오 데이터(초당 60 프레임)로 구성)
  - 연구의 목표: 인간처럼 여러 게임을 학습할 수 있는 **단일 신경망 에이전트**를 만드는 것
  - 특정 게임에 대한 정보나 사람이 설계한 시각적 특징을 받지 않았고, 에뮬레이터의 내부 상태에 접근 X
  - 모든 게임에서 네트워크 구조와 학습에 사용된 하이퍼파라미터는 동일하게 설정

💡 **<추가 개념>**

1. Q-learning

- RL에서 사용하는 알고리즘 중 하나로, Agent가 특정 상태에서 어떤 행동을 할 때 얻을 수 있는 장기적인 보상을 학습
- Q-러닝의 목표: **Q함수**라 불리는 상태-행동 가치 함수를 최적화하여 최적의 정책을 찾는 것

2. correlated data

- RL에서는 시간 순서대로 수집되는 데이터가 종종 상관관계를 가지므로 독립적이지 않은 경우가 많음

3. non-stationary distributions

- RL 환경에서는 에이전트의 행동에 따라 데이터의 분포가 계속해서 변화

→ 이 상황을 non-stationary distributions라고 함. (모델이 학습하는 동안 데이터의 패턴이 계속 변할 수 있다는 의미)

4. Experience Replay Mechanism

- 강화학습에서는 에이전트가 환경과 상호작용하며 수집한 데이터를 학습
- Experience Replay Mechanism은 과거의 경험을 저장하고 이를 무작위로 샘플링하여 학습에 사용하는 방법
- 시계열 상의 연관성이 줄어들어 데이터가 독립적(i.i.d.)이라는 가정을 충족 & 학습 안정성을 높이는 데 도움

5. Arcade Learning Environment (ALE)

- 다양한 Atari 2600 게임을 통해 강화학습 알고리즘을 테스트할 수 있는 환경(주로 연구 목적으로 사용됨)
- 에이전트가 시각적으로 복잡한 환경에서 어떻게 학습하고 행동하는지 평가할 수 있도록 고안됨

2. Background

- 에이전트는 Atari 에뮬레이터와 같은 환경과 상호작용하며, 시간마다 가능한 행동 중 하나를 선택
  - 선택된 행동은 환경의 상태와 게임 점수에 영향을 주며, 에이전트는 현재 화면의 픽셀 데이터(이미지)와 보상(점수 변화)을 관찰
  - 일반적으로 게임의 상태는 시간에 걸쳐 달라질 수 있기 때문에, 행동의 효과는 수천 번의 시간 단계 후에 나타날 수 있음
- **MDP(마르코프 결정 과정)와 상태 표현**
  - 에이전트는 현재 화면만 관찰할 수 있기 때문에 현재 상황을 완전히 이해하는 데 한계가 있음
  - 이를 해결하기 위해, 모든 행동과 관찰의 순서를 상태 표현으로 고려하여 마르코프 결정 과정(MDP)을 사용 → 표준 강화학습 방법을 적용 가능해짐
- **Agent 목표**: 가능한 많은 보상을 얻도록 행동을 선택하는 것
  - 목표 달성을 위해 Q(s,a)라는 action-value function 정의

    → 특정 상태에서 특정 행동을 했을 때 기대되는 미래 보상의 최대값 나타냄
- **Bellman equation을 통한 Q-값 계산**
  - Q-값을 반복적으로 업데이트
  - 반복적인 Q-값 계산은 최적의 행동 가치 함수에 수렴
  - 최적 행동 가치 함수 정의

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

  - **Q*(s,a):** 상태 s에서 행동 a를 선택했을 때의 최적 행동 가치 함수
    - 상태-행동 쌍이 장기적으로 가져올 보상의 기대값을 나타냄
  - E: 다음 상태 s'가 환경 ε에서 결정된 확률에 따라 발생할 때의 기댓값
  - r: 현재 상태-행동 쌍에서 발생하는 즉각적인 보상
  - γ: 미래 보상에 대한 현재 가치의 중요도를 조절하는 매개변수(1에 가까울수록 먼 미래의 보상도 현재 가치에 반영되며, 0에 가까울수록 현재 보상에만 집중)

- 손실 함수 정의

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

- Q-러닝에서는 Q(s,a) 값을 근사하기 위해 신경망(Q-네트워크) 사용
- Li(theta_i): i번째 반복에서의 손실 함수
- y_i: i번째 반복에서의 타겟 Q값(현재 보상 r에 다음 상태 s′에서 가능한 행동 중 최적의 Q-값을 γ 와 함께 추가하여 타겟 Q-값을 계산)
- ρ(·): 행동과 상태의 확률 분포로, 행동과 상태가 경험 재생 메커니즘을 통해 무작위로 샘플링된다는 것을 나타냄

- 손실 함수의 기울기 계산

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

- Q--네트워크의 가중치를 업데이트하기 위해 손실 함수 기울기 계산

- **Q-Network 학습**
  - Q-러닝에서는 Q-함수를 추정하기 위해 **신경망**을 사용
  - 각 반복에서 신경망의 손실을 최소화하여 Q-네트워크의 가중치를 업데이트
  - 손실 함수는 현재 예측 Q-값과 타겟 Q-값의 차이의 제곱

💡

1. **Markov Decision Process(MDP)**
   - 강화학습의 수학적 모델로, 상태(state), 행동(action), 보상(reward), 상태 전이 확률로 구성됨
   - 각 상태에서 특정 행동을 했을 때 다음 상태로 전이될 확률과 이에 대한 보상을 정의하여, 최적의 행동을 선택하는 기준이 됨

2. **Bellman Equation**
   - 현재 상태에서 최적의 행동을 선택하기 위한 수학적 수식
   - 이 방정식을 반복적으로 풀어가며 최적의 Q-값을 계산하게 되며, 이는 Q-러닝의 핵심

3. Off-Policy 학습
   - 에이전트가 다른 정책에 따라 행동하면서도 최적의 정책을 학습할 수 있게 함
   - 이 논문에서는 **ε-greedy** 방식을 사용하여 ε 확률로 랜덤한 행동을 선택하여 탐색하고, 나머지 확률로 현재 최적의 행동을 선택하는 방식

3. Related Work

1. TD-gammon

- 강화학습의 성공 사례
- backgammon 게임에서 강화학습과 자기 학습(self-play)만으로 학습하여 인간 수준 이상의 성능을 보임
- TD-gammon은 **Q-러닝과 유사한 모델 기반 강화학습 알고리즘**을 사용했고, **다층 퍼셉트론**(Multi-layer Perceptron)을 이용해 가치 함수(value function)를 근사
- 체스, 바둑, 체커 등 다른 게임에서는 성공X

2. 비선형 함수 근사와 Q-러닝의 불안정성 문제

- Q-러닝을 비선형 함수 근사와 결합하거나 Off-policy 학습 방식으로 사용하면 Q-네트워크가 발산(divergence)할 수 있다는 문제가 발견됨
- 많은 연구들이 수렴 특성이 더 좋은 **선형 함수 근사(linear function approximators)**에 집중

3. 딥러닝과 강화학습의 결합 연구

- Restricted Boltzmann Machines과 같은 신경망을 통해 가치 함수나 정책을 근사하려는 연구가 진행됨
- **Gradient Temporal-Difference(GTD)** 방법을 사용하여 Q-러닝에서의 발산 문제를 해결하려는 접근도 이루어짐(GTD 방법은 비선형 함수 근사를 사용할 때 수렴을 보장하는 기법)

4. Neural Fitted Q-learning (NFQ)

- 이 논문과 가장 유사한 선행 연구
- NFQ는 손실 함수를 최적화하기 위해 RPROP 알고리즘을 사용하여 Q-네트워크의 가중치를 업데이트
- 배치 업데이트(batch update)를 사용하여 데이터셋 크기에 비례하는 계산 비용이 발생
    - 확률적 경사 하강법이 아닌 배치 업데이트를 채택

5. Atari 2600 에뮬레이터의 강화학습 플랫폼 활용

- Atari 2600 에뮬레이터는 강화학습 연구에서 고차원 시각적 특징을 테스트하는 플랫폼으로 사용
- 이 환경에서는 여러 게임에서 선형 함수 근사와 일반적인 시각적 특징을 사용하여 강화학습을 적용하는 연구가 이루어짐
- 해시(Hashing) 방법이나 진화적 아키텍처(예: HyperNEAT)를 통해 성능을 개선하고자 함

4. Deep Reinforcement Learning

- Deep Reinforcement Learning 개요
  - 최신 컴퓨터 비전 및 음성 인식 기술에서는 깊은 신경망을 훈련하여 많은 양의 데이터에서 학습을 통해 표현력을 높이는 방법을 사용
  - 이 접근법을 강화 학습에 적용하면 에이전트가 직접 원시 데이터(RGB 이미지 등)에서 학습하고 업데이트를 수행할 수 있게됨
  - **Stochastic Gradient Descent(확률적 경사 하강법)**을 통해 효율적으로 훈련할 수 있는 신경망을 사용하는 것이 핵심

- **TD-Gammon**
  - 초기 연구 중 하나인 TD-Gammon은 비정책적(on-policy) 샘플링을 통해 학습
  - 에이전트는 주어진 상태에서의 행동에 따라 바로 다음 상태를 학습하며, 이로 인해 지속적인 학습 업데이트가 가능
  - 이 방식은 시간이 지남에 따라 **학습 데이터가 특정 행동이나 상태에 편향되기 쉽고, 샘플 간의 상관관계가 강해짐**
  - Deep Reinforcement Learning에서는 이런 문제를 해결하기 위해 **Experience Replay**와 같은 방법이 도입됨

- **Experience Replay의 도입 및 장점 & 단점**
  - Experience Replay는 기존의 학습 방식과 달리 **에이전트가 경험한 모든 데이터(상태, 행동, 보상, 다음 상태)를 데이터셋 D에 저장**하고, 미니배치로 **무작위로 샘플링**하여 학습하는 방식

    → **데이터 독립성**: 샘플 간 상관성을 줄여 학습 과정이 안정화됨

    → **데이터 효율성**: 이전 경험을 재사용하여 더 많은 학습이 이루어지며, 중요하지 않은 데이터나 특정 행동에 대한 과잉 학습을 방지 가능

    → **Off-policy 학습 가능**: Q-Learning처럼 특정 정책을 따르지 않고, 다양한 정책에서 얻은 데이터로 학습이 가능

  - 모든 샘플이 동일한 빈도로 사용되기 때문에 중요한 샘플이 덜 학습될 수 있음

→ 이를 개선하기 위해 중요한 경험을 더 자주 샘플링하는 "**Prioritized Experience Replay**" 기법이 존재

- **Algorithm 1: Deep Q-learning with Experience Replay**:

  Algorithm 1에서는 Experience Replay를 사용하는 Deep Q-Learning의 전체적인 절차를 보여줌

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

  - 초기화 단계: Replay memory D를 용량 N으로 초기화 & Q-function을 무작위 가중치로 초기화
  - 반복
    - **탐색과 학습**: 에이전트는 일정 확률에 따라 무작위 행동을 선택하거나, 현재 Q-function의 예측에 기반하여 행동을 선택(epsilon-greedy)
    - **경험 저장**: 에이전트가 얻은 경험을 Replay memory에 저장
    - **미니배치 샘플링**: Replay memory에서 랜덤하게 미니배치를 샘플링하여 Q-learning 업데이트를 수행
    - **Q-value 업데이트**: 보상 r_j와 다음 상태의 최적 Q-value를 이용하여 손실 함수를 최소화하도록 Q-value를 업데이트
  - **데이터의 무작위성 확보**: Experience Replay를 통해 샘플 간 상관성을 줄여 주기적인 샘플 편향을 줄일 수 있음
  - **데이터 균형 유지**: 경험이 Replay memory에 저장되므로, 특정 행동이 반복적으로 학습되기보다는 다양한 경험이 고르게 학습됨

- **버퍼 크기 제한**: N 크기의 Replay memory에 최신 경험만 유지하기 때문에 메모리 제한이 있으며, 오래된 정보는 덮어씌워짐 ⇒ 중요한 정보가 손실될 수 있다는 한계

💡

1. **Deep Q-Learning**
   - Q-Learning에 심층 신경망을 결합한 방식
   - Q-Learning은 상태-행동 쌍에 대한 Q-value를 업데이트하지만, Deep Q-Learning은 심층 신경망을 통해 복잡한 입력 데이터를 처리하여 Q-value를 예측

2. **Replay Memory**
   - 에이전트가 경험한 모든 데이터를 메모리에 저장하는 방식
   - 무작위 샘플링을 통해 과거 경험을 재활용하여 더 효율적인 학습을 유도

3. **Epsilon-Greedy**
   - 행동 선택 시 무작위 탐색과 현재 정책 기반 탐색을 혼합하는 방식
   - 작은 확률(epsilon)로 무작위 행동을 선택하여 탐색을 하고, 나머지 확률로는 Q-function의 최대 값을 기준으로 행동을 선택

4. **Prioritized Experience Replay**
   - 모든 경험을 동일하게 사용하는 대신, 학습에 더 중요한 경험(학습에 큰 영향을 미치는 경험)에 우선순위를 부여하여 더 자주 학습에 사용하게 함

1. Preprocessing and Model Architecture

- **전처리 단계**: Atari 게임의 원본 화면 크기는 210*160 픽셀에 128개의 색상 팔레트를 가지고 있어 계산량이 크기 때문에, 먼저 입력 이미지의 크기를 줄이는 전처리 과정을 거침
  - **그레이스케일 변환**: RGB 이미지에서 그레이스케일로 변환하여 채널 수를 줄이기 ⇒ 계산 부담 감소
  - **크기 조정**: 110*84 픽셀로 다운샘플링 ⇒ 이미지 차원 감소
  - **이미지 크롭**: 게임의 실제 플레이 영역을 포착하기 위해 84*84 영역으로 잘라내기

- **프레임 스태킹**: 마지막 4개의 프레임을 쌓아서 Q-네트워크의 입력으로 사용 ⇒ 에이전트가 물체의 움직임을 이해할 수 있도록 시간적 정보를 제공

- **Q-함수의 파라미터화**: Q-함수는 주어진 상태에서 각 행동에 대한 Q-value를 출력
  - **별도 전달 방식**: 각 행동에 대해 Q-값을 계산하기 위해 별도의 전달이 필요하며, 행동의 수가 증가할수록 비용이 커지는 단점이 있음
  - **통합 출력 방식**: 네트워크의 출력에 각 가능한 행동에 대한 개별 Q-value를 나타내는 출력을 구성하여 forward pass로 모든 행동의 Q-값을 계산 가능

  ⇒ 효율적이며, DQN이 선택한 방식

- **신경망 구조 (Architecture)**
  - 입력: 84*84*4 크기의 이미지(4개의 프레임이 스택된 상태)
  - 첫 번째 은닉층: 16개의 8*8 필터, 스트라이드 4를 적용하여 특징을 추출 ⇒ **Rectifier 비선형성(ReLU)을 적용하여 활성화**
  - 두 번째 은닉층: 32개의 4*4 필터, 스트라이드 2로 컨볼루션을 수행 ⇒ **ReLU 활성화를 적용**
  - 세 번째 은닉층: 256개의 뉴런을 가진 완전 연결된 층으로 구성 ⇒ **ReLU 활성화를 적용**
  - 출력층: 게임의 각 가능한 행동에 대한 Q-value를 예측하는 하나의 출력을 생성하는 완전 연결된 층. ⇒ 가능한 행동의 수는 게임에 따라 4에서 18까지 다양

5. Experiments

- 실험 게임
  - DQN을 Atari 게임 7개(Beam Rider, Breakout, Enduro, Pong, Q*bert, Seaquest, Space Invaders)에서 테스트
  - 특정 게임에 맞춘 튜닝 없이 일관된 네트워크 구조, 학습 알고리즘, 하이퍼파라미터를 사용하여 다양한 게임에서 DQN이 얼마나 잘 작동하는지 확인하는 것

- 보상 구조 수정

    - 게임마다 점수의 크기가 다르기 때문에, 모든 **양의 보상은 1로, 음의 보상은 -1로 고정**
    - 0 보상은 변경 X

    ⇒ 게임 간의 점수 차이에 따른 학습 불균형을 줄이고, 여러 게임에서 동일한 학습 속도를 사용할 수 있도록

    ⇒ 보상의 크기를 구별할 수 없게 만들기 때문에, 보상 차이에 따른 미세한 학습이 어렵다는 한계

- 학습 설정
    - **RMSProp 알고리즘**을 사용하여 미니배치 크기 32로 학습
    - **epsilon-greedy**: 초기에는 ε을 1에서 시작해 100만 프레임 동안 점진적으로 0.1까지 감소시킨 후, 이후에는 0.1로 고정하여 학습
    - **학습 프레임**: 총 1,000만 프레임 동안 학습 & 최근 100만 프레임을 replay memory에 저장하여 경험을 샘플링

- frame-skipping 기법
    - 모든 k번째 프레임마다 에이전트가 행동을 선택하도록 하여 계산 비용 줄임

      ⇒ 프레임마다 선택하는 것보다 계산량을 줄이면서도 학습 효율을 유지할 수 있는 방법
    - Space Invaders에서는 k=3, 나머지는 k=4

1. Training and Stability

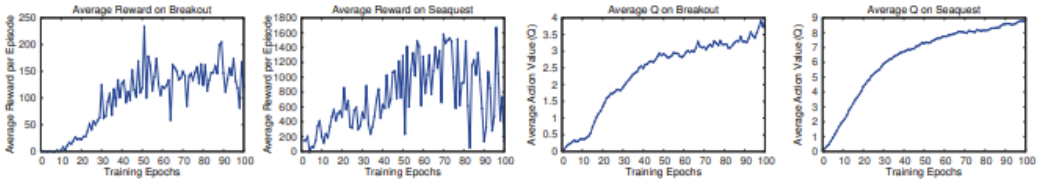
- 학습 평가의 어려움
    - 강화 학습에서는 모델의 성능을 평가하기 어렵
    - **에피소드 당 총 보상**이나 **여러 에피소드에서 평균 보상**과 같은 지표를 주기적으로 계산

      ⇒ 이 값들은 상당히 불안정하여 학습의 진전이 명확하게 보이지 않을 수 있음

- 평가 metric

- **Average Total Reward**: 에이전트가 특정 에피소드나 여러 게임에서 얻은 **평균 보상**을 계산
  ⇒ 학습 중 작은 변화에도 민감하게 반응하여
  일관성 있는 진전을 파악하기 어렵
- **Estimated Action-Value Function (Q)**: 에이전트가 **특정 상태**에서 정책을 따라 얻을 수 있는 할인된 보상

  ⇒ Q 함수 값은 매끄럽게 증가하여 학습이 안정적으로 진행되는지 판단하는 데 유용한 지표로 사용됨



- 왼쪽의 두 그래프: Breakout과 Seaquest 게임에서 에피소드 당 평균 보상
- 오른쪽의 두 그래프: 학습되지 않은 상태 집합에서의 Q 함수 값
- Q 함수의 평균 예측 값이 비교적 매끄럽게 증가하는 것을 통해 학습의 안정성을 확인할 수 있음

2. Visualizing the Value Function

- Seaquest 게임의 특정 프레임에서 에이전트가 학습한 Q 값을 시각화한 것
- 적이 화면에 등장했을 때 Q 값이 증가하고, 적을 공격하거나 사라지면 Q 값이 감소
- DQN이 게임 환경에서 변화하는 상황에 따라 Q 값을 유연하게 조정하는 방법을 학습했음을 알 수 있음

3. Main Evaluation

- **DQN 성능 평가와 비교 대상**
  - **Sarsa**: "State-Action-Reward-State-Action"의 약자로, 상태와 행동 쌍의 시퀀스를 통해 정책을 학습.
    - Sarsa 알고리즘은 특정 피처 세트를 사용하여 선형 정책을 학습하며, DQN처럼 심층 신경망을 사용하지 X
    - 사전에 정의된 피처(특징) 세트를 이용하기 때문에, 일반화된 환경에서는 성능이 떨어질 수 있음
  - **Contingency**: Sarsa와 비슷하지만, 학습되는 피처 세트를 조정하여 에이전트의 통제 아래에 있는 화면의 부분을 나타내도록 함.
    - 에이전트가 게임 속에서 제어할 수 있는 특정 객체나 위치 정보를 중심으로 피처를 설계하는 방식
  - **HNeat**: 진화적 알고리즘을 사용하여 정책을 탐색하는 방법
    - HNeat는 Atari 화면에서 8색 채널을 사용하여 각 객체의 위치와 종류를 나타내도록 설계됨
    - 객체를 탐지하는 데는 효과적이지만, 이러한 방식은 게임의 불확실성을 다루는 데는 한계가 있으며, 새로운 상황에 대한 일반화가 어려움

- **DQN의 학습 접근 방식과 HNeat와의 차이점**

  <DQN>
  - 단순히 RGB 원시 이미지 입력만을 사용해 학습한다는 점에서 기존 알고리즘들과 차별화됨
  - 사람이 수동으로 객체를 정의하고 피처를 만드는 대신, **에이전트가 이미지로부터 객체의 위치와 중요성을 직접 학습**하도록

  <HNeat>
  - Atari 화면의 각 객체를 미리 정의된 특수한 채널로 구분하여 학습
  - 적이나 장애물 같은 객체를 특정 색상으로 표현해 에이전트가 쉽게 인식하도록 도움
  - 특정한 구조에 최적화된 모델을 만드는 데 유리하지만, 랜덤한 환경 변화나 일반화된 상황에서는 성능이 떨어질 수 있음

- **실험결과**
  - **기본 평가 방법**: 각 알고리즘은 동일한 ε-greedy policy을 사용하여 평가됨
    - ε-탐욕 정책: 행동을 선택할 때 최적의 행동을 할 확률을 높게 설정하되, 일부 확률(ε)로 임의의 행동을 선택하게 하여 새로운 행동을 시도할 가능성을 유지하는 방식

      ⇒ 에이전트가 과도하게 특정 행동에 집착하지 않도록 방지
  - **DQN의 성능**: DQN은 Beam Rider, Breakout, Enduro, Pong 등의 게임에서 상당히 우수한 성능을 보임.

    ⇒ DQN이 단순한 피처가 아닌 원시 이미지 데이터를 학습하면서도 성공적으로 강화 학습을 수행했음을 의미
  - **어려운 게임에서의 성능 한계**: Q*bert, Seaquest, Space Invaders 같은 게임에서는 인간의 성능에 미치지 못하는 결과

    ⇒ 단기 보상보다는 장기적인 전략과 계획이 필요한 경우가 많아, 단순한 Q-learning 알고리즘이 어려움을 겪을 수 있음

6. Conclusion

- DQN이 다양한 Atari 게임에 적용될 수 있는 일반화된 강화 학습 모델로 자리매김
- 기존 알고리즘과 달리, **DQN은 별도의 피처 엔지니어링 없이도 높은 성능을 보이며, 이는 딥러닝이 원시 데이터에서 복잡한 관계를 학습하는 데 강력하다는 점을 보여줌**
- DQN은 학습하는 동안 모든 상태와 행동을 저장해두고 반복해서 학습하는Experience Replay)을 활용하여, 같은 데이터를 여러 번 학습할 수 있게 함

  ⇒ 학습 데이터가 부족하거나 비효율적으로 학습되는 문제를 해결하는 데 큰 도움