Playing Atari with Deep Reinforcement Learning Review

- **Abstract**
  - ➔ 강화학습을 사용하며 고차원의 sensory input으로부터 control policies를 성공적으로 학습하는 딥러닝 모델을 처음으로 소개한다. 모델은 variant Q-learning으로 학습된 CNN구조의 모델이다. 7개의 Atari 2600게임들을 순수 픽셀 값만 입력하여 어떠한 수동적인 조작없이 적용했다.
- **Introduction**
  - ➔ 강화학습에 있어 고차원의 시각 혹은 청각의 sensory inputs으로부터 학습하는 것은 되게 어려운 일이다. 기존에 있는 이러한 domain을 적용한 성공적인 강화학습은 linear value function이나 policy representation을 결합한 hand-crafted features에 의존을 많이 했다.
  - ➔ 최근 딥러닝 연구는 순수 sensory data로부터 고차원의 특징들을 추출할 수 있게 되었다. 이것들은 주로 신경망 구조를 사용하였다.
  - ➔ 하지만 강화학습은 딥러닝의 관점에서 바라보고 적용하려고 하면 몇가지 문제점들이 존재한다. 첫째로, 대부분의 딥러닝 기술들은 많은 양의 hand-labelled training data가 필요하지만, 강화학습 알고리즘은 sparse, noisy 그리고 delayed 된 scalar reward로부터 학습을 한다. 둘째로, 대부분의 딥러닝 알고리즘은 데이터가 서로 독립적임을 가정을 두고 학습을 하는데, 강화학습은 데이터의 sequences 값들이 매우 관련이 크다. 마지막으로, 강화학습에서 데이터 분포는 알고리즘이 새로운 행동을 학습할 때마다 바뀌는 반면, 딥러닝에서의 데이터 분포는 고정된 분포로 가정을 둔다.
  - ➔ 위 논문에서는 CNN이 이러한 문제들을 해결해서 순수 비디오 데이터로부터 control policies들을 학습을 하여 복잡한 강화학습 모델에 적용하는 것을 보여줄 것이다. 네트워크는 variant Q-learning으로 학습이 된다. 데이터의 상관성이 큰 큰 것 비정적 분포를 따르는 문제를 완화시키기 위해서 'an experience replay mechanism'을 도입했다. 이는 이전 상태들을 랜덤하게 추출하여, 학습 분포를 smooth하게 만들어주는 역할을 한다.
  - ➔ 이 실험의 목표는 모든 게임에서도 적용될 수 있는 단일 신경망 네트워크 agent를 만드는 것이다. 네트워크는 어떠한 game-specific information이나 hand-designed visual features 등이 주어지지 않은체, 비디오 입력 값으로만 학습이된다. 게다가, 네트워크 구조와 hyperparameters들은 모두 고정 값으로 사용을 하였고, 6개의 게임에서 기존보다 나은 성과를 보여주었다.
- **Background**

➔ Model-free technique 중 하나인 Q-learning 알고리즘을 신경망 네트워크를 접목시키고 잇다. 이 실험에서 환경은, Atari emulator 자체가 되고, 이는 actions, rewards, image pixels가 환경을 구성한다.

➔ 논문에서는 vision data를 활용하여 state를 정의하는데, 하나의 화면에서만 보고 얻는 정보에는 한계가 있으므로 이전의 vision과 action의 history data까지 활용한다. 그래서 t 시점의 state는 action과 vision data의 sequence로 본다.

➔ Policy를 학습할 때, 이전 sequence까지 고려하여 결정하는데, 유한한 결과값이 있으므로, 일종의 MDP (Markov Decision Process)로 볼 수 있다.

➔ Reward Function은 과거 시점을 고려하여 특정 시점 때마다, discount factor를 고려해서 합치는 식이 된다.

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}.$$

➔ Q-function은 pi를 st에서 at를 mapping하는 policy function으로 보면, optimal Q-function은 다음과 같다.

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[R_t \middle| s_t = s, a_t = a, \pi\right]$$

➔ 이 식은, Bellman Equation을 따르는데, 만약 s'의 optimal Q-function이 a' action으로 알려질 수 있으면 이는 이렇게 표현이 될 수 있다.

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

➔ Q-function을 측정할 때, Bellman Equation은 value iteration 알고리즘을 사용하여 구하는데, 이는 sequence에 대해 모두 독립적으로 시행하기 때문에 이론적으로는 가능하지만 실제로는 불가능한 성격을 띄고 있다. 그래서 action-value function을 적절한 approximator를 사용하여 측정한다. 여기서 쓰이는 approximator는 주로 linear function으로도 쓰이지만, non-linear function으로 모델링 할 때는 신경망 모델을 활용하는데, 이를 Q-network로 부른다.

➔ Q-network는 sequence의 loss function을 최소화 시키는 방향으로 학습을 하는데 이는,

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

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

➔ 최적화 할 때, θ가 update될 때, loss function의 이전 θ값은 고정되어서 'free target Q-

network'라고 부른다. 이는 지도학습과는 다르게 target이 θ에 영향을 많이 받아, 안정적인 학습을 하기 위해 θ을 고정하기 때문이다.

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

➔ 위에 수식은, Q-network의 loss의 gradient를 구한 식이다.

- **Deep Reinforcement Learning**

➔ 이 논문은 강화학습 알고리즘을 stochastic gradient updates를 사용하며 학습 데이터를 처리하는 DNN과 연결하는 것이 최종 목표이다.

➔ TD-Gammon과는 다르게 이 논문은 experience replay를 활용했다. 이것은 agent의 experience(st,at,rt,s(t+1))를 각 시점마다 D라는 data-set에 저장을 하여 replay memory에 쌓아 올리는 것이다. 이 알고리즘의 안에 있는 부분은 Q-learning updates 혹은 minibatch updates를 표본의 experience에 적용한다. Experience replay를 실행한 후에, agent는 ∈-greedy policy에 따라 행동을 실행한다. Q-function은 φ함수에 의해 생성된 histories의 고정된 표현을 활용한다.

➔ Deep Q-learning with Experience Replay algorithm

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

➔ 그리고 연속적인 표본으로부터 직접적으로 학습하는 것은 높은 상관관계가 있어 비효율적이기에, 표본들을 랜덤화 시켜서 이러한 상관관계를 부수고 매 반복마다 표본의 분산을 줄인다.

➔ On-policy를 학습할 때, 현재 parameters는 다음 데이터 샘플을 결정한다. 예를 들어, action이 왼쪽으로 움직이는 것으로 극대화 시키면, 학습 표본들은 왼쪽으로 가는 것으로 지배된다.

➔ Experience replay를 사용함으로써 행동 분포는 이전의 상태에 대해 평균화 되고,

학습을 smoothing시키고 parameters의 진동과 발산을 피하게 한다.

➔ 실제로, 이 알고리즘은 지난 N개의 experience를 replay memory에 저장을 하고, 업데이트를 진행할 때, D로부터 랜덤하게 표본을 추출한다. 이것은 한계가 있는데, memory buffer는 유한한 N 메모리 크기 때문에 중요한 transitions을 차별화시키지 않고 항상 최근 transitions위에 작성하기 때문이다.

- **Preprocessing and Model Architecture**

  ➔ 128개의 color palette인 210 x 160 pixel 크기의 Atari frame 자체로부터 일이 진행되기에, 이것은 계산 복잡도가 부담될 수 있다. 그래서 연구진은 간단한 전처리 단계를 도입했다. RGB Atari frames은 gray-scale로 표현을 바꾸고 이미지를 110 x 84 크기로 줄였다. 그리고 나서 cropping을 통해 정사각형 크기인 84 x 84크기로 줄였고 이것이 최종적으로 input에 들어갈 이미지가 된다. 이것은 history의 4 frames들이 스태킹되어 Q-function 알고리즘에 input으로 들어간다.

  ➔ Q는 history action paris를 Q-value의 측정값으로 매핑시키기 때문에, history와 action은 NN에 inputs으로 들어간다. 이런 구조의 가장 주된 결점은 분리된 forward pass가 Q-value를 각 action마다 계산하기 위해서는 필요한 것이다. 그래서 연구진은 각 action마다 separate output이 있는 구조를 활용했고, state representation은 NN의 input으로 사용했다. Output은 input에 각 예측된 action의 Q-values에 대응된다. 이 구조의 가장 주된 장점은 Q-values를 모든 action마다 한 개의 forward pass만을 사용하며 계산할 수 있다는 점이다.

  ➔ NN의 input은 84 x 84 x 4 크기의 이미지이고, 첫번째 hidden layer는 16개의 4x4필터이고 두번째 hidden layer는 32개의 4x4필터이고 마지막 hidden layer는 256개의 노드가 있는 FC layer이다.

- **Experiments and Results**

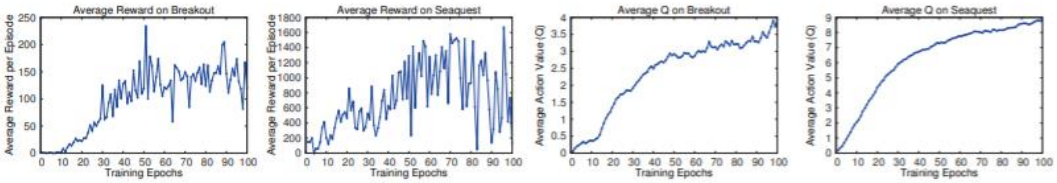

Figure 2: The two plots on the left show average reward per episode on Breakout and Seaquest respectively during training. The statistics were computed by running an ϵ-greedy policy with $\epsilon = 0.05$ for 10000 steps. The two plots on the right show the average maximum predicted action-value of a held out set of states on Breakout and Seaquest respectively. One epoch corresponds to 50000 minibatch weight updates or roughly 30 minutes of training time.

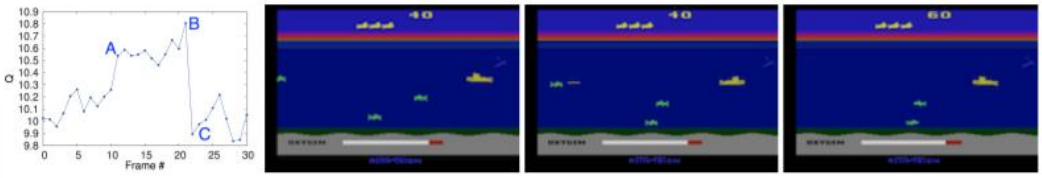

Figure 3: The leftmost plot shows the predicted value function for a 30 frame segment of the game Seaquest. The three screenshots correspond to the frames labeled by A, B, and C respectively.

	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 $\epsilon = 0.05$.