**[Playing Atari with Deep Reinforcement Learning]**

1. **Abstract**

* Reinforcement Learning(RL)을 이용하여 high-dimensional sensory input으로부터 control policeies 를 성공적으로 학습한 최초의 deep learning model이다.
* CNN 모델이 사용되었으며, Q-learning의 한 variant로 학습되었다.
* 모델의 input은 raw pixels이고, output은 미래의 보상을 예측하는 value function이다.
* 7종류의 2600개의 Atari 게임을 학습한 결과, 7개 중 6개는 전의 결과를 뛰어 넘었으며 그 중에서 3개는 인간의 기대를 능가했다.

1. **Introduction**

* Vision이나 Speech와 같은 high-dimensional sensory input에서 agent를 제어해 학습하는 것은 오랜 과제였다.
* 이전까지 이러한 분야에서의 성공적인 RL application은 hand-crafted features에 의존적이었다. 즉, feature representation의 질(quality)에 따라 성능이 좌지우지 되었다.
* 최근에는 DL의 발전으로 computer vision이나 speech recognition과 같은 raw-sensory data에서 high-level feature을 추출하는 것이 가능해졌다.
* 하지만 RL은 DL의 관점에서 몇가지 문제점들이 존재한다.
* 먼저 성공적인 DL이 되려면 hand-labelled된 많은 양의 training data가 요구된다. 반면에 RL 알고리즘에서는 대부분 sparse, noisy, delayed인 scalar reward signal 로부터 학습되어야만 한다.
* DL 알고리즘에서는 데이터들이 독립적이라는 가정이 수반되지만 RL에서는 강하게 연관되어 있는 상태를 전형적으로 다루게 된다.
* 새로운 behavior을 학습할 때마다 데이터 분포가 변하는 RL의 특성은 고정된 분포를 가정하는 딥러닝과 상충된다.
* 이 논문에서는 CNN이 raw video data가 복잡한 RL 환경에서 성공적인 control policies를 통해 위 문제점을 해결 가능하다는 것을 보여준다.
* 이 Network는 가중치를 업데이트하기 위해 Stochastic Gradient Descent를 사용한 Q-learning의 한 variant로 학습되었다. 또한 상호연관된 데이터와 정제되지 않은 분포의 문제를 완화하기 위해서, 무작위로 이전의 transitions를 뽑는 experience replay mechanism을 사용하였다. 이는 수많은 이전 behaviors의 training distribution을 완화시킬 수 있다.
* 목표는 가능한 많은 게임을 성공적으로 학습시켜 하나의 neural network agent를 만드는 것이다.
  + Atari 2600 game은 high-dimensional visual input을 가진 agent로 표현되며, 사람에게도 어려운 환경이 주어진다.
  + Network에는 게임의 구체적인 정보나 hand-designed visual feature들을 제공되지 않았으며, 오로지 초기상태에서 video input과 player로부터 얻을 수 있는 reward, terminal signals, 그리고 set of possible actions들로부터 학습된다.

1. **Background**

* Agent가 environment(환경)에서의 task를 보면, 각 time-step마다 agent는 action을 선택하게 된다. Action이 전달되면 내부 state와 게임 score가 변하게 된다.
* 일반적으로 환경은 stochastic이기에, agent는 내부 state를 알지 못하고, 현재 화면의 raw pixel vector과 score의 변화만을 알 수 있다.
* 하지만 게임의 score는 현재의 action뿐만 아니라 일련의 action들과 observations에 의해 결정되고 action에 대한 feedback은 수천번의 time-step이 진행된 후에 받게 된다.
* 그렇기에 일련의 actions들과 observations에 의존하는 game strategies를 학습하게 되며 각 sequence가 고유의 state를 의미하는 MDP를 적용하게 된다.
* State를 표현하는 시간 t로 전체 sequence를 사용함으로써, 기본적인 RL 학습방법인 MDP를 적용할 수 있다.
* 결국 agent의 목표는 미래의 보상을 최대화하기 위해 actions을 선택하고 emulator와 interact하는 것이다.

1. **Related Work**

* 이전에 RL이 적용된 가장 잘 알려진 사례는 TD-gammon이었다. 강화학습을 통해 스스로 플레이하는 방법을 학습하고 인간 수준의 플레이를 보여주기도 했다. TD-gammon은 Q-learning과 유사한 model-free RL 알고리즘을 사용했으며, multi-layer perceptron을 사용하여 value function을 계산하였다.
* 하지만 chess나 Go에서 TD-gammon의 방식은 좋지 못한 결과를 보였으며, TD-gammon은 backgammon에서만 작동되는 특수케이스로 볼 수 있다고 여겨지게 되었다.
* 그리고 최근에야 DL과 RL을 융합시키는 분야가 다시 나타나기 시작했다.  Deep Neural Network는 환경 𝐸E 를 측정하기 위해 사용되게 되었고, restricted Boltzmann 기계들은
* 그리고 최근에야 DL과 RL을 융합시키는 분야가 다시 나타나기 시작했다.  Deep Neural Network는 환경 𝐸E 를 측정하기 위해 사용되게 되었고, restricted Boltzmann 기계들은 value function이나 policy를 측정하기 위해 사용되기 시작하였다.
* 게다가 발산문제는 gradient temporal-difference에 의해 다루어지게 되었고, 이것은 non-linear function approximator로 fixed policy를 사용할 때나 제한적인 Q-learning의 변형을 활용하여 linear function approximation과 함께 control policy를 학습할 때 수렴함을 증명하였다.
* 그러나 이것은 non-linear control까지 확정되지는 않은 상태였다.
* 우리의 접근법과 가장 유사했던 이전의 작업으로는 Neural fitted Q-Learning(NFQ)가 있다. NFQ는 Q-Network의 Parameter들을 갱신시키기 위해 RPROP 알고리즘을 사용하여 2번 방정식에서의 Loss Function을 최적화시켰다.
* 그러나 이것은 Iteration을 돌기 위해 많은 계산 양을 필요로하는 Batch Gradient Descent를 사용하였지만, 이 논문에서는 Stochastic Gradent Descent를 사용하여 Iteration을 돌기 위해 필요한 계산 양을 줄였고, 큰 Data-Set까지 학습을 Scale-up 시켰다.
* NFQ는 또한 처음으로 deep autoencoder를 사용함으로써 task의 low dimensional representation을 학습하였고, 시각입력을 사용함으로써 real-world control task를 성공적으로 NFQ 알고리즘에 적용하였다.
* 그러나 이와 반대로 우리는 시각적 입력으로부터 직접적으로 철저한 강화학습을 적용시켰고, 결과적으로 Action-Value를 판별하는 것과 같은 특징들을 학습하였다.
* RL 플랫폼으로 사용된 Atari 2600 에뮬레이터는 visual features와 linear function approximtion을 standard RL에 적용시켰던 [3] 논문에서부터 사용되기 시작하였습니다.
* 결과적으로 많으 수의 특징들은 lower-dimensional space에 적용시킴으로 결과는 개선되었다. HyperNEAT evolutionary architecture는 Atari Platform에도 적용되었으며, 이 플랫폼은 각 게임마다 전략을 나타내는 Neural Network를 발전시키는데 사용되었다.
* Emulator의 리셋 기능을 사용하여 결정을 내리는 Sequence에 대해 반복적으로 교육을 받았을 때, 이러한 전략은 여러 아타리 게임에서의 디자인 결함을 악용할 수 있었다.

1. Deep Reinforcement Learning

**4. Deep Reinforcement Learning**

Computer Vision과 Speech Recognition에서 이뤄낸 최근의 성과는 매우 큰 training sets를 활용하여 deep neural network를 효과적으로 훈련시킨 결과였다. 대부분의 성공적인 approaches는 Stochastic Gradient Descent를 기반으로 lightweight update함으로써, raw input들로부터 직접 학습된 것들이였다. Deep Neural Network에 충분한 양의 Data를 제공함으로써, handcrafted된 features보다 많은 representation들을 학습할 수 있었고, 이러한 성공들을 바탕으로 RL에 대한 우리의 접근법을 생각해냈다. 우리의 목표는 RL 알고리즘을 Deep Neural Network와 연결하여 RGB Image들에 직접적으로 작동하고, Stochastic Gradient Updates를 사용하여 Traning Data를 효율적으로 처리하는 것이다.

TD-Gammon과 달리 우리는 experience replay라는 기술을 활용하였는데, Agent가 매 Time-Step마다 했던 Experience(Episode)들을 Dataset에 저장을 시키고, 수많은 Episode들이 replay memory에 쌓이게 된다. 그리고 알고리즘 내부에서 샘플들이 저장된 풀로부터 임의로 하나를 샘플링하여 학습(Q-Learning, Mini-Batch)에 적용시켰다. 이후에(experience replay 후) Agent는 e-greedy policy에 따라 행동을 선택하고 수행한다. Neural Network의 입력으로써 가변적인 history를 사용하는 것은 어렵지만, Deep-Q Algorithm에서는 𝜙ϕ 함수를 사용하여 고정 길이의 history를 입력으로 사용한다.

이러한 DQN은 기존의 Q-Learning보다 아래와 같은 장점을 지닌다.

1. each step의 Experience가 잠재적으로 많은 weight update에 재사용되므로, Experience를 weight update 한번만 사용하는 기존의 방법보다 훨씬 data efficiency 하다.
2. 연속적인 sample들로부터 학습을 진행하는 것은 데이터들간의 high correlations때문에 비효율적이다. 하여 sample들을 e-greedy 알고리즘을 통해 randomize하여 sample들의 high correlations를 break하여 update의 효율성을 높인다.
3. 기존의 on-policy를 통해 학습을 하면, 매개변수가 학습된 다음 데이터 샘플을 결정한다. 예를 들어, 만약 보상을 극대화하는 행동이 왼쪽으로 움직이는 것이라면 training sample들은 왼쪽의 샘플들로 dominate 될 것이다. 만약 극대화하는 행동이 오른쪽으로 바뀌면 traning distribution 역시 오른쪽으로 바뀔 것이다. 이런 것들을 통해 원하지않는 feedback loops가 발생하고, parameter들이 local minumum으로 수렴하거나 발산함을 파악할 수 있다. experience replay를 사용함으로써, behavior distribution이 균형을 이루게 되고 parameter의 발산이나 진동을 피하고 학습을 매끄럽게 진행한다. 또한 experience replay를 통해 학습하기 위해서는 Q-Learning의 choice를 원활히하는 off-policy를 사용해야 한다. 왜냐하면 Behavior Policy의 Parameter 𝜖ϵ 과 Learning Policy의 Parameter 𝜃θ 가 다르기 때문이다.

DQN에서는 replay memory 안에 마지막 N개의 exprience만을 저장하고, update를 하기위해 무작위로 Data Set으로부터 추출한다. 이러한 접근법은 Memory Buffer가 중요한 Transition에 차별점을 두지 않고 항상 제한된 크기 N의 버퍼에 최근의 Transition을 덮어 씌운다는 접에서 한계가 있다. 마찬가지로, uniform sampling은 replay memory안의 모든 transition에 동일한 중요성을 부여한다. 더욱 정교한 Sophisticated Sampling 전략은 우선순위를 매기는 것과 유사하게 우리에게 가장 중요한 Transition을 중요시 할 것이다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

1) replay memory D와 이것의 capacity를 N으로 초기화한다.   
2) action-value function Q를 random weight로 초기화한다.   
3) episode를 1~M까지 반복한다.   
4) sequence s1을 t=1일때의 이미지 𝑥1x1으로 초기화하고, 전처리과정을 통해 𝜙1ϕ1 을 구한다.   
5) t를 1~T까지 반복한다.    
6) e-greedy 알고리즘을 따라 무작위 action 또는 이전에 최선의 결과를 냈던 action 중 하나를 𝑎𝑡at  선택한다.   
7) emulator에서 action 𝑎𝑡at 를 수행하고, reward 𝑟𝑡rt와 다음 image 𝑥𝑡+1xt+1 를 observe한다.   
8) 그리고 현재의 State 𝑠𝑡st, 현재의 Action 𝑎𝑡at, 새로운 image인 𝑥𝑡+1xt+1 을 𝑠𝑡+1st+1 로 저장하고, 𝑠𝑡+1st+1 에 대해 pre-processing을 한다.   
9) 그리고 replay memory D에 현재의 상태를 전처리한 값 𝜙𝑡ϕt, 𝑎𝑡at, 𝑟𝑡rt, 𝜙𝑡+1ϕt+1 을 저장한다.   
10) D에 저장된 Sample들 중에서 minibatch의 개수만큼 random하게 뽑는다.   
11) 여기서 𝑦𝑗yj를 정의하는데, 전처리한 결과인 𝜙𝑗+1ϕj+1 이 목표 지점에 도달하면 𝑟𝑗rj 로, 목표지점이 아니라면 𝑟𝑗+𝛾𝑚𝑎𝑥𝑎′𝑄(𝜙𝑗+1,𝑎′;𝜃)rj+γmaxa′Q(ϕj+1,a′;θ) 로 설정한다.   
12) 그리고 방정식 3을 따라 Loss Function을 정의하고 gradient desenct를 수행한다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

**4.1 Preprocessing and Model Architecture**

**[ Preprocessing ]**

128 color palette를 가진 210 \* 160 pixel images인 raw Atari frames로 직접 작업하는 것은 정말 많은 계산양을 필요로 한다. 그래서 먼저 input의 dimensionality를 줄이는 basic preprocessing step을 적용한다. 이러한 과정은 먼저 RGB로 표현된 이미지를 Gray-Scale로 변환하고, 110 \* 84의 이미지로 down-sampling 시킨다. 이후에 게임의 진행 부분만 보이도록 84 \* 84로 잘라내서 Final Input값을 추출한다. 이 알고리즘을 적용하기 위해서는 마지막 84 \* 84로 잘라내는 기술이 반드시 필요한데, 사용하는 GPU에서 정사각형 사진만 GPU 연산이 가능하기 때문이다. 위의 알고리즘의 전처리 함수인 𝜙()ϕ() 에서 마지막 4개의 frames만 전처리를 하여 Stack에 넣어두고, 이를 입력에 대한 Q-function의 값을 구하기 위해 사용한다.   
  
=> 4개의 프레임이 1개의 화면을 구성하기에 4개의 프레임을 기준으로 처리한 것

스크린샷이(가) 표시된 사진

자동 생성된 설명

**[ Model Architecture ]**

Q-Value를 구하는 방법으로는 2가지가 있다.

1. istory와 action을 input으로 하고, output으로 history와 그 action에 대해 예측된 Q-value를 구하는 것
2. history만을 input으로 하여 output으로 각 행동에 대해 예측된 Q-Value를 구하는 것

1번을 사용하면 입력으로 들어온 action에 대해 seperate forward pass연산을 진행해야 하고, action의수가 증가함에 따라 linear하게 연산의 양도 증가하는 단점이 있다. 그래서 2번의 방법을 사용하는데, Neural Network의 입력으로 state를 사용하여 각 state에서 수행가능한 모든 Action에 대한 Q-value를 계산하기 때문에 좀 더 효율적이다. 우리는 어떤 상태를 입력으로 주고 원하는 행동에 대한 Q-value를 사용하면 된다. 즉, 주어진 상태에서 가능한 모든 Action에 대한 Q-value값을 한번의 single forward pass로 처리하는 이점을 얻을 수 있다.

조류이(가) 표시된 사진

자동 생성된 설명

**[ DNN Architecture ]**

1. Neural Network의 Input은 𝜙ϕ 를 통해 전처리된 84 x 84 x 4 이미지(4 frames)이다.
2. 첫 번째 Hidden Layer는 input image에 stride 4를 토함한 16 8x8(16 channels with 8x8 filters)로 합성곱 연산을 한 후에, rectifier non-linearity(ex. relu)를 적용한 것이다.
3. 두번째 Hidden Layer는 stride 2를 포함한 32 4x4(32 channels with 4x4 filters)로 합성곱 연산을 하고 rectifier non-linearity(ex. relu)를 적용한 것이다.
4. 마지막 Hidden Layer는 funnly-connected되고, 256개의 rectifier 유닛으로 구성된다.
5. 최종적으로 Output layer는 각 수행가능한 행동에 대해 single output을 갖는 fully-connected linear layer이다.

텍스트, 지도이(가) 표시된 사진

자동 생성된 설명

**5. Experiments**

Experiment를 진행하면서 algorithm 및 hyper-parameter에 대한 3가지 setting이 나온다.

1. Reward Structure: Training을 하면서 Reward Structure에 한가지 변화를 주었다. 양의 보상은 1, 음의 보상은 -1, 변화 없음은 0으로 수정하였는데, 이를 통해 error derivatives(오류 도함수)의 스케일을 제한하고, 모든 게임에 동일한 learning rate를 적용할 수 있었다. 그러나 보상의 강도 역시 제한함으로써 에이전트의 성능의 제약을 가져올 수 있다.
2. RMSProp Algorithm & 𝜖ϵ - greedy Algorithm: 최적화 알고리즘으로는 크기 32의 Minibatch을 RMSProp 알고리즘에 적용하였고, Behavior Policy로는 처음부터 100만번째 프레임까지는 1에서 0.1까지 동일한 비율로 감소하는 epsilone 값을 통해 e-greedy 알고리즘을 사용하였고, 이후에는 0.1로 고정하였다.
3. Frame Skipping Technique: Agent가 모든 Frame을 보고 Action을 취하는 것이 아니라 K번째 프레임을 보고 액션을 고르게 하였고, 마지막 행동은 skipped된 frames에 반복 적용시켰다. 액션을 선택하기 위해서는 많은 양의 계산을 필요로 하지만, 한 스텝을 forward하는 것은 훨씬 적은 연산을 필요로 한다. 그러므로 이러한 기술을 적용함으로써 실행시간은 같지만 약 k배 많이 게임을 진행시킬 수 있었다. Space Invaders를 제외한 모든 게임에서 k=4 로 정해주었는데, 이 게임에서는 게임 내부의 laser blinking과 문제가 생겨 특별히 k=3으로 처리를 하였다.

**5.1 Training and Stability**

Supervised Learning에서는 training sets와 validation sets를 통해 학습 중에 model의 성능을 쉽게 확인할 수 있었다. 그러나 Reinforcement Learning에서는 학습 중에 agent의 progress를 정확히 측정하는 것이 매우 어렵다. 이 논문에서 Evaluation Metric(평가의 척도)은 수많은 Games를 통해 평균화된 Reward를 토대로, Game 또는 Episode에서 평균적으로 얻은 보상이기 때문에 training동안에 모은 total reward를 주기적으로 계산해야 한다. 하지만 학습을 진행하면서 Policy의 Weights를 아주 조금 변화시키는 것이 state의 distribution에 큰 변화를 줄 수 있어서, 여기서 사용하는 average total reward metric은 때때로 noisy하다.    
  
Figure2에서 왼쪽의 두 그래프는 Sequest, Breakout이라는 게임에서 학습을 하는동안 totalreward가 어떻게 변화하는지 보여준다. 두 그래프는 steady한 progress를 만들어내지 못했으며, 상당히 noisy하다. 하지만 Policy에 대한 Action-Value를 예측하는 Q-function은 상당히 stable한데, 이 함수는 Agent가 어떤 state에서 Policy를 따랐을 때, 얼마의 discounted reward를 얻을 수 있는지를 제공한다. 우리는 traning이 시작하기 전에 random policy를 적용하고 게임을 실행하여 몇개의 fixed state를 수집하였고 이런 state들에 대해 maximum predicted Q의 값을 도출하였다. 오른쪽의 두 그래프는 Average Predicted Q가 Agent를 통해 얻은 average total reward보다 훨씬 smooth하게 증가함을 보여주고, 다른 게임에서도 그래프를 그려보았을때 유사하게 smooth하게 증가하였다. 또한 실험에서 Q 값이 발산하는 경우는 없었다. 비록 반드시 수렴한다는 이론적인 검증은 없지만, 우리의 방식이 RL과 SGD를 사용하여 Neural Network를 stable하게 학습시킴을 알 수 있다.

스크린샷, 회로이(가) 표시된 사진

자동 생성된 설명

**5.2 Visualizing the Value Function**

Figure3는 Seaquest 게임에서 학습된 value function을 시각화한 것이다. 해당 그림은 screen의 왼쪽에 enemy가 등장하였을 때, predicted value가 jump함을 보여준다(Point A). 적을 발견한 Agent는 Enemy를 향해 미사일을 발사하고, 발사된 미사일이 적을 맞추려고 할 때, predicted value가 오름을 보여준다(Point B). 그리고 적이 사라졌을 때 predicted value는 원래의 값으로 떨어지게 된다(Point C). Figure 3는 우리의 value function이 꽤나 복잡한 일련의 이벤트들에 대해 어떻게 진화해 나가야 할지 학습할 수 있음을 보인다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

**5.3 Main Evaluation**

Table1은 일정한 step까지 0.05의 e값을 갖는 e-greedy 알고리즘을 적용하여 다양한 학습 알고리즘을 적용한 결과이다. 아래의 표는 HNeat와 DQN에 대해 Episode의 실행 결과가 가장 좋았던 경우이다.

=> Sarsa: Sarsa알고리즘은 Atari task를 처리하기 위해 수작업으로 설계된 여러 가지 특징들을 위해 Linear Policy를 학습하였고, 가장 성능이 좋은 Feature Set의 점수를 report 하였다.

**6. Conclusion**

이 논문은 RL을 위한 새로운 Deep Learning Model을 소개하였고, 이 모델을 바탕으로 raw pixel들만을 입력으로 사용하여 2600가지가 넘는 아타리 게임을 위한 어려운 Control Policy를 학습하는 것이 가능함을 증명하였다. 또한 deep network의 학습을 쉽게하기 위해 Stochastic Minibatch Gradient Descent에 Experience Replay Memory를 적용한 Q-Learning의 변형을 소개하였다. 이러한 접근법은 Architecture나 Hyperparameter의 변화없이, 7개의 게임 중 6개의 게임에서 경이적인 결과를 도출하였다.

1. ㅁㅁ