0 . 강화학습 개념정리먼저 해봅시다.

1. 강화학습을 푼다는 것은 **최적의 정책함수(=미래에 얻을수 있는 보상함수의 기댓값을 최대로 하는 행동)**를 찾는것과 같다.
2. 특정 환경과 상호작용하여, **보상을 최대화** 하는 행동 또는 행동 순서를 학습
3. 미래에 얻어지는 보상도 다 고려해야한다.

4. state : 상태 , 선택하기 위해 받는 정보
5. reward : agent 가 action을 하였을때 받는 보상
6. environment : 모든 환경
7. action : agent가 특정 환경에서 취할수 있는 행동
8. policy : 특정 state 에서 action 을 골라주는 함수 (어떤 액션을 골라줄것인가?)
9. episode : 시작 ~ 종료까지 agent가 거친 sequence

10. 강화학습에는 라벨이 없다. 시행착오를 겪으면서 학습 데이터를 스스로 모은다!
11. 강화학습의 데이터는 time series data : 각각의 데이터들이 독립적이지 않다는것. 상관관계를 가지고 있다.
12. 보상이 바로 정해지지 않는다. 여러 행동을 조합해서 reward를 받기도 한다. (그전에 내 행동이 맞는지 아닌지 알기 어렵다.)
13. action에 따라 다음 state는 불확실하다. (상대방이 어떻게 행동을 취할지 불확실하다.)
14. 현재의 결정, action이 이후에 올 데이터에 영향을 미친다.

15. **MDP : 마르코프 의사결정과정**
    - 현재가 주어졌을때, 과거와 미래가 독립적임
    - **t+1 번째의 state 는 오직 t번째 state 및 action 에만 의존함**
    - 현재의 나의 상태에는 과거의 상태가 "반영"은 되어있음. 그렇다고 미래를 결정하는것이 과거라는건 아님
16. 전이함수 : 특정 상태에서 특정 행동을 했을때, 다음번에 도달할 상태들의 확률을 나타낸것. (상대방이 어디에 둘것이냐 ?)
17. reward function : t+1번째 reward를 예측하는 함수

18. **Bellman equation : 벨만 방정식**
    - sequence s' 의 다음 time stamp 에서의 optimal Q-function 값이 모든 action a ' 에 대해서 알려져 있다면, optimal strategy는 $r+\gamma Q*(s',a')$의 expected value를 maximize하는 것이라는 것이다.
19. MDP 가 주어졌을때, 최적의 정책 함수를 찾는 가장 기본적인 방법

20. **Value function** : 현재의 어떤 state에서 출발하여 얻을수 있는 모든 보상들의 합의 기댓값.
    - V(s)
    - 해당 state 인 s의 가치를 매기기 위한 함수이고 이를 위해 가치반복, 정책 반복을 사용
21. **할인률** : 같은 보상을 얻을수 있다면 빨리 얻는것이 좋음.

22. 가치반복
    - 현재의 policy가 optimal하다고 전재하고, max를 취한다.
    - policy 가 optimal 하다는건 >>
23. 정책반복 : 정책 평가, 정책 개선 (π)
    - policy에 따라 value function이 확률적으로 주어진다.
    - 기댓값으로 value function을 구하게 된다.

24. **Q learning : 해당 state 에서 어떤 action을 취하는게 최적인지 구하기 위함**.
    - Q(s,a)
    - state 에서 취할수 있는 각 action 들의 가치를 알수 있다.

25. TD learning (temporal difference learning : 시간차 학습)
    - 대부분의 문제는 **model free** 이므로, 직접 여러번 각 state 에서 action 을 계속 취해보면서 전이함수 T 를 직접 구해야 한다.
    - 이때 활용하는것이 TD learning 이다.
    - **새로운 값에 대한 추정값이 들어오면 기존의 값과 우리가 정한 a비율로 섞이게 된다.**
    - 미래 가치에 대한 추정값을 사용해서 학습한다.

1. 논문리뷰 시작

👉 Playing Atari with Deep Reinforcement Learning

00. Abstract

- 강화학습을 통해 **successfully learn control policies directly from high demensionoal sensory input**
- Atari는 CNN 모델을 사용하며, 변형된 Q-learning을 사용하여 학습되었습니다.
    - input : raw pixels

    - output : value function estimating future rewards (미래의 보상을 예측하는 가치함수)
- 게임을 학습할때 픽셀 값들을 입력으로 받고, 각 행위에 대해 점수를 부여하고, 어떤 행동에 대한 결과값을 함수를 통해 받게된다.
- Atari games중에서 7개의 게임환경을 통해 이 방법을 적용했고, 7개중 6개의 게임에서 이전 모든 접근들을 능가했고, 6개 중에서 3개는 인간을 넘어섰다.

01. Introduction

- **vision , speech** 같은 **high demensionoal sensory input** 으로 부터 agent를 학습시키는것은 강화학습의 오랜 과제였다.
- 딥러닝이 발전함에 따라 Vision, Speech와 같은 고차원의 데이터들을 추출하는 것이 가능해졌다.
- 하지만 딥러닝을 강화학습에 적용하는 과정에서 몇가지 문제점을 발견하게 되었다,
    - deep learning : label 된 많은 양의 training 데이터를 필요로 함.
    - reinforce learning : sparse, noisy, **delayed (between action and rewards)****한 reward signal 라는 scalar 값을 통해 학습한다.
    - > RL에서는 어떠한 행위를 하면 그 행위에 대한 결과를 알기까지 **시간**이 필요하다. 그리고 이런 **delay**는 어려움을 유발 한다는것이죠~!~!

    - **☆딥러닝 알고리즘에서 각 데이터 들은 독립적이지만, 강화학습에서는 하나의 행위가 다른 것들과 연관성이 높다.**

    - RL 에서는 알고리즘이 새로운 행동을 배울때마다 data 의 distribution이 변하게 되는데, 이것은 데이터의 분포가 고정되어있다고 가정하는 딥러닝의 가정과 충돌하여 문제가 될수 있다.

- BUT, 이 논문에서는 그것을 극복한, CNN이 복잡한 RL 환경에서 성공적인 학습을 할수 있음을 증명한다.
- ☆변형된 **Q-Learning을** 통해 학습되며, weight를 update하기 위해 **stochastic gradient descent**를 사용합니다.
- correlated data문제를 해결하기 위해 **experience replay mechanism**을 사용한다.
    - 이전에 학습한 데이터들을 저장하여 random한 샘플로 사용하고, 그렇게 함으로써 training distribution이 smooth 하도록 만드는 기법.

- 우리의 목표는 👧 **to create single neural network agent that is able to sucessfully learn to paly as many of the games as possible.** (가능한 많은 게임들을 성공적으로 학습할수 있는 NN을 만드는것이다.)

- **☆NN은 어떠한 게임에 대한 특정 정보나 게임의 우위를 위한 데이터등을 제공받지 않는다.**
- **☆오직 비디오의 시각 데이터와 Reward 그리고 터미널로부터 오는 신호 그리고 가능한 몇개의 행동으로만 학습을 진행.**
- 또한 다양한 게임들에 동일한 network architecture과 hyper parameter를 사용했다.

02. Background

- Reinforcement Learning을 위해서는 먼저 **환경 E**을 정의해야한다. 이 논문에서는 **Atari 에뮬레이터**가 환경이 될 것이다.
- 매 시간마다 agent는 legal game action A={1,...,K} 중에서 action 을 하나 선택한다.
- 게임을 하는동안 컨트롤러의 버튼을 누르는 action이 모이게 되면 현재 점수에 어떻게든 영향을 주게되고, 그 결과로 최종 score가 결정된다. (action 이 모여서 score 결정)

- **즉, Atari 게임 환경에서 reward 는 게임 score이다.**
- 현재 내가 선택한 action이 바로 reward 에 반영되는것이 아니라 나중에 반영될수도 있음. (received after many thousands of time-steps have elapsed.)

- 이 논문에서는 **vision 데이터를 사용해서 state를 정의**하는데, 간단하게 xt를 state로 삼으면 될 것 같지만 그렇지 않다.
- ☆실제로는 화면 하나만 보고 알 수 있는 정보가 제한적이고 **현재 상태를 정확하게 판단하기 위해서**는 vision정보와 내가 행한 action을 포함한 **과거 history**들까지 모두 있지 않으면 안되기 때문에 이 논문은 **state 를 action과 image의 sequence로 정의한다.** ☆
  - ☆이 사실때문에 state를 image와 action 의 sequence로 정의 한다.

- 그리고 게임은 언젠가 끝나기 때문에 finite 한 MDP가 된다.
- 목표 : 가장 높은 점수를 획득하는것. (! 시간이 오래 지날수록 해당 reward의 가치는 내려간다. = 할인률)

- $Q*(s,a)=\max_{\pi}E[Rt|st=s,at=\alpha,\pi]$: **정책 π 를 통해 얻을수 있는 reward의 maximum expected value**
  - 최적의 Q fuction은 bellman equation이라는 특성을 따른다.
  - MDP 에서는 이 **optimal action value function 혹은 optimal Q-function 하나만 제대로 알고 있다면**, 반드시 **항상 optimal한 action을 고를수 있다.**
  - 이때 optimal strategy 는 $r+\gamma Q*(s',a')$ 를 maximize 하는것
  - Q function : 각 state에서 어떤 행동 a를 했을때, 기대되는 미래 보상의 총합(가치)

- RL에서는 모든 state와 action에 대한 labeled data가 없기 때문에 이를 어떻게 다뤄야 하는지를 모델에서 고려해야만한다. 또한 현재까지 연구된 많은 deep learning structure들은 data가 i.i.d. 하다고 가정하지만, **실제 RL 환경에서는 state들이 엄청나게 correlated되어있기 때문에 제대로 된 learning이 어렵다.**

- 이 논문에서 문제를 해결하기 위해 사용한 방법
  1. **Freeze taget Q-network:**

     optimization 과정에서 parameter θ가 update되는 동안 loss function $Li(\theta i)$ 의 이전 iteration paramter $\theta i-1$은 고정된다는 것이다.

     이렇게하는 이유는 supervised learning과는 다르게, target의 값이 θ의 값에 (민감하게) 영향을 받기 때문에 stable한 learning을 위하여 θ값을 고정하는 것이다.

  2. **Experience replay :**

     agent 의 experience 를 각 time stamp 마다 다음과 같은 튜플 형태의 **메모리에 저장한 후 이를 다시 이용하는것이다.**

     Experience replay를 사용함으로써 data의 correlation을 깰 수 있고, 조금 더 i.i.d.한 세팅으로 network를 train할 수 있게 된다. 또한 방대한 과거 데이터가 한 번만 update 되고 버려지는 비효율적 접근이 대신에, **지속적으로 추후 update에도 영향을 줄 수 있도록** 접근하기 때문에 데이터 사용도 훨씬 효율적이라는 장점이 있다.

- 3. **Clip reward or normalize network adaptively to sensible range**

  reward의 값을 [-1,0,1] 중에서 하나만 선택하도록 강제하는 아이디어이다.

  내가 100점을 얻거나 10000점을 얻거나 항상 reward는 +1 이다. 'highest' score를 얻는 것은 불가능하지만, 이렇게 설정함으로써 조금 더 stable한 update가 가능해진다.

03. Deep Reinforcement Learning

- 👉 **experience replay**
  - " 인접한 학습데이터 사이의 correlation으로 인한 비효율성 극복하기 위한 기법"
  - agent 가 매 time step 마다 했던 episode 경험들을 dataset에 저장시켜, 수많은 episode 들이 replay memory에 싸이게 된다.
    - **agent의 경험 데이터 (s,a,r,s')를 replay memory에 저장**
  - 그리고 알고리즘 내부에서 샘플들이 저장된 풀로 부터 임의로 하나를 샘플링하여 학습에 적용시켰다.
  - 이러한 DQN 은 replay memory 안에 마지막 N개의 exprience만을 저장하고, **update를 하기위해 무작위로 Data Set으로부터 추출한다. (동일한 중요성을 부여한다는것이다.)**
  - ☆ **experience replay를 사용함으로써, behavior distribution이 균형을 이루게 되고 parameter의 발산이나 진동을 피하고 학습을 매끄럽게 진행한다.** ☆
    - 만약에 experience replay 를 사용하지 않고 on-policy인 경우 ❓) 예를 들어, 만약 보상을 극대화하는 행동이 왼쪽으로 움직이는 것이라면 training sample들은 왼쪽의 샘플들로 dominate 될 것이다. >> 이렇게 되면 will be dominated by samples from the left-hand side . >> **stuck in poor local minimum, or enver dicerge catastrophically.**
  - ☆ **By using experiece replay the behavior distribution is averaged over many of its previous states.** 이 문장이

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

- 1. 초기화 한다.
  2. episode를 1~M까지 반복한다.
  3. sequence s1 을 t=1 일때의 이미지 x1 으로 초기화 하고, 전처리한것을 초기화 한다.
  4. time step 을 1부터 T까지 반복
  5. e-greedy 알고리즘을 따라 무작위 action 또는 이전에 최선의 결과를 냈던 action 하나를 선택한다.
     - sample들을 e-greedy 알고리즘을 통해 randomize하여 sample들의 high correlations(충돌) 를 break하여 update의 효율성을 높인다. (딥 러닝은 correlations 를 깨야하니까 이런 방식이 효율적이라고 하는것 같음.)
  6. action at 를 수행하고, reward rt와 다음 image xt+1 를 받는다.
  7. 그리고 현재의 State st, 현재의 Action at, 새로운 image인 xt+1 을 st+1 로 저장하고, st+1 에 대해 pre-processing을 한다.
  8. replay memory D에 현재의 상태를 전처리한 값 ϕt, at, rt, ϕt+1 을 저장한다.
  9. D에 저장된 Sample들 중에서 minibatch의 개수만큼 random하게 뽑는다.
  10. 이 목표 지점에 도달하면 rj 로, 목표지점이 아니라면 rj+γmaxa'Q(ϕj+1,a';θ) 로 설정한다.

04. Experiments

- visualizing the value function : 학습된 value function을 시각화한것.

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.

- 적이 왼쪽에 등장했을때 A : predicted value가 상승
- 적을 발견하여 미사일이 적을 맞추려고 할때 B : predicted value 가 상승
- screen 에서 적이 사라졌을때 C : predicted value 감소

05. Conclusion

- 이 모델을 바탕으로 raw pixel 들만을 입력으로 사용하여 2600개가 넘는 atari 게임을 위한 control policy를 학습하는것이 가능함을 증명하였다.
- deep network에 Stochastic Gradient Descent 에 Experience Replay Memory 를 적용한 Q learning 의 변형을 소개했다.
- 좋은 결론을 도출했다~~~

2️⃣나의 한줄 정리 : 🗝️

가장 인상 깊었던 부분은 state를 image와 action의 sequece로 정의했다는점! 그래서 자동으로 문제를 해결했다는게 신기하다. atari 영상을 보고도 놀랐다! 공략법을 찾아서 구멍을 내고... 신기방기.. 😳

experience replay 라는 기법을 사용한다. (이게 이 논문의 핵심) RL 의 가장 중요한 특성인(?) 보상이 바로바로 정해지지 않음 때문에 이런 기법을 생각해 낸것이 아닌가 싶다. 그니까 한쪽으로 결과가 치우쳐지지 않게 계속 random한 값을 넣어주는것이 아닐까!?

암튼 논문 하나를 읽으면서 참고 설명들과, 영어 글씨들을 열심히 읽어 내려갔지만 역시나 어렵긴 어려웠다.

다음에 다시 이 논문을 보게 된다면 수식적인 부분도 좀더 열심히 살펴봐야겠다!

3️⃣ 참고자료

https://mangkyu.tistory.com/60

http://hugrypiggykim.com/2019/03/10/playing-atari-with-deep-reinforcement-learning/

http://sanghyukchun.github.io/90/