Playing Atari with Deep Reinforcement Learning

00 Abstract

first deep learning model - control policies directly

pixel 입력 받아 미래 보상을 계산하는 value function 출력

강화학습 도전과제로 사용되는 Atari 2600 게임 중 7가지 게임에 적용하였으며 - 6개 게임에서 기존 연구보다 뛰어난 성능을, 3개 게임에서는 실제 사람(전문가)의 능력을 뛰어넘는 결과를 내었다.

01 Introduction

RL의 도전과제인 vision & voice : high-dimensional sensory input 를 다룰 것

기존 RL model → hand-crafted features + linear value function / policy representation 에 의존

- 선형 가치 함수 : 입력 피쳐의 선형 결합으로 value 계산

DL 기술의 발전은 RL에게 이익이 될 수 있지만, 곧바로 RL에 적용하는 것을 박는 몇가지 문제점이 존재

1. 막대한 데이터 - sparse, noisy and delyed :

   특히 입력-보상 사이에는 수많은 timesteps(delay)가 존재하는데, 라벨과 즉시 매핑되는 supervised learning에 비해 직접적인 연관성이 저해될 수 있다.
2. data samples - not independent : RL 에서는 correlated 된 데이터 사용
3. changing data distribution : 새로운 행동을 만들어내는 과정에서 데이터 분포의 변화가 계속됨

논문에서는..

- CNN

- a variant of the Q-learning + stochastic gradient descent → weight update
- experience replay mechanism (correlated data & non-stationary distribution 해결)

Atari 2600 games?

- in ALE
- (210 × 160 RGB video at 60Hz) - 인풋으로 사용

연구의 최종 목적은 수많은 게임 플레이가 가능한 하나의 NN model을 만드는 것.

02 Background

- environment = Atari emulator

  agent 는 emulator가 보여주는 장면으로 환경을 파악한다. 따라서 environment를 완전히 이해하지 못하기 때문에, St = x1, a1, x2, a2, ..., at-1, xt - 각 시점마다 어떤 고유한 상태를 나타내는 시퀀스 S를 사용하여 학습을 진행한다.

  각 시퀀스는 유한한 time-steps 가 끝나면 종료된다
- a large but finite Markov decision process(MDP) 성립하며 → 일반적인 RL 알고리즘 사용 가능

- agent의 목표는 보상을 최대화하는 액션을 선택하는 것
  - gamma : t에 따라 감소하는 미래 보상을 결정 (감소하는 정도)
  - pi : policy(sequence - action mapping)
  - optimat action-value function Q*(s,a) : 어떤 전략을 사용했을 때 기대할 수 있는 최대 보상

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

    Bellman equation : 현재 보상과 미래 보상(불확실성 - discounted) 합의 최댓값 표현. r은 현재 보상, 감마~ 미래보상

value iteration algo (update Q) → 반복이 무한대로 가면 최적 값으로 수렴한다는 가정

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

- approximator 사용 가능

$$Q(s,a;\theta) \approx Q^*(s,a)$$

보통 선형이지만, NN사용하여 비선형 모델로 ⇒ Q-network

- Q-networt
  - Loss function

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

θ : Q-network(s-a 예측)의 가중치. 학습의 목표가 이 최적 theta를 찾는 것.

Li(θi) : 최적 theta를 찾기 위해 최적화할 값 - theta가 최적 값에서 얼마나 떨어져 있는지 표현

yi : target. 이때 타겟은 네트워크 가중치에 따라 달라진다(항상 같지 않다)

  - L을 미분하여 최적점 찾기

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

- model-free
  - 환경 설계 대신 emulator 사용
- off-policy
  - greedy stage $a = \max_a Q(s,a;\theta)$
  - ϵ-greedy strategy : $1-\epsilon$ → greedy, ϵ → random

    - exploration 보장

03 Related work

- TD-gammon : backgammon(주사위 게임) playing program, self-play하여 인간 능가하는 성능을 달성
    - Q-learning과 비슷한 model-free, multi layer perceptron으로 가치 함수 추정
    - 다른 게임(체스, 바둑)에 적용 같은 접근법 사용 어려움 - 주사위 사용
    - Q-learning에 non-linear function approximator를 사용하는 것이 발산을 일으킴
    - DNN + RL 새로운 시도
- DL + RL : DNN → estimate the environment or policy
    - divergence → gradient temporal-difference 메소드로 해결
        - 고정된 정책 평가 - 비선형 approximator 사용해도 수렴
        - nonlinear control 로 확장하기는 어려움

- NFQ : neural fitted Q-learning
    - RPROP - 손실함수 최적화/Q-network 업데이트
    - batch update (계산 비용)
    - deep autoencoder → 저차원 표현
- 이 연구에서
    - stochastic update
    - end-to-end : visual input 가지고 직접 학습 (오토인코더 없음) → action-value를 직접적으로 식별할 수 있도록
    - raw visual input → experience replay 적용 (기존에는 저차원 상태에서 진행했었다)

04 Deep Reinforcement Learning

> Our goal is to connect a reinforcement learning algorithm to a deep neural network which operates directly on RGB images and efficiently process training data by using stochastic gradient updates.

Experience replay

- time-step 별로 agent의 경험을 episode로 replay memory 에 저장 → 무작위 샘플링, Q-learning update
- ϵ-greedy policy 에 따라 리플레이 후 행동 선택
- 다른 길이의 input 사용 어려움 - fixed length representation of histories 사용

⇒ deep Q-learning

위 접근의 장점

- data efficiency : 각 시간에서 경험이 가중치 업데이트에 잠재적으로 사용됨
- 랜덤 샘플 - break correlation → reduce the var of the update

  진동, 발산 방지 : 무작위 샘플링 - 평균화.
  - on-policy는 현재 학습 중인 파라미터가 다음 학습 데이터 샘플을 결정하므로 feedback loop 또는 poor local minimum에 빠질 수 있다.

⇒ off-policy : 현재 파라미터와 샘플링에 쓰이는 파라미터가 다르므로.

- 최신 N개의 experience 저장(순서에만 의존)
- 모두 같은 확률로 샘플링 - 모든 경험이 동일한 중요도를 가지게 된다

  → priority sweeping : 학습 효과가 높은 경험에 중요도 부여하여 해결

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

04.01 Preprocessing and Model Architecture

reducing input dimensionality (원본은 210x160 pixel with a 128 color palette)

- gray-scale
- down-sampling → 110×84
- GPU-2d convolution 인풋은 squre → 84×84 로 크롭

기존 방식에서는 각 행동의 가치 값(Q)을 계산할 때마다 순전파가 필요 → 액션의 수에 비례한 비용 증가

→ 입력으로는 상태만 입력, 행동 별로 output unit을 두어 한번의 순전파로 모든 행동의 Q 값 계산

architecture

- input : 84*84*4, produced by φ (reducing input dimensionality파트의 과정)
- hidden layer
  - stride 4, 8×8 filter, 16 convolution - recrifier nonlinearity
  - stride 2, 4×4 filter, 32 convolution - rectifier nonlinearity

    - fully-connected(256 rectifier units)
- output : 각 액션 별로 하나의 출력 - fully connected linear layer

05 Experiments

Beam Rider, Breakout, Enduro, Pong, Q*bert, Seaquest, Space Invaders

모두 같은 모델(세팅)으로 학습시킨다

- 각 게임 별로 score의 차이가 있기 때문에 승리 +1, 패배 -1, 변화없음 0 으로 통일하여 학습
- RMSProp, minibatch(32)
- epsilon ; 0.1
- 10 million frames and used a replaymemory of one million most recent frames
- frame-skipping : k 번째 프레임만 보도록 k=3 사용

0501 Training and Stability

RL - 학습 중 성능 평가 어려움

Average total reward metric - 작은 가중치의 변화가 쌓여 큰 분포의 변화를 만들어낼 수 있다. noisy

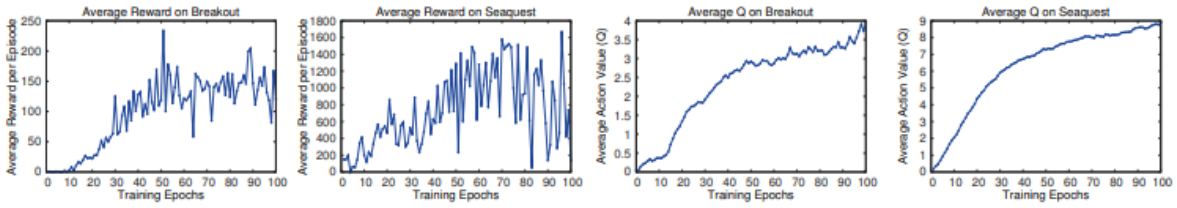
→ steady progress 없음

Policy’s estimated action-value function Q

- 그 정책을 따랐을 때 에이전트가 얻는 총 보상을 기준으로 판단 (discounted reward?)

→ 보다 안정적인 진행

→ 발산하는 문제가 있음



0502 Visualizing the Value Function

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.

(Seaquest) - 적이 보이면 올라가서 발사, 사라지면 내려온다

0503 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

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 $\epsilon = 0.05$.

Sarsa

- hand-engineered feature 사용

Contigency

- Sarsa + 컨트롤 화면 일부 추가

두 알고리즘은 background subtraction, 색상별 분리된 채널을 사용한다. - 이는 아타리 게임에서 한 객체가 하나의 색으로 표현되기 때문임.

⇒ DQN은 채널 분리 없이 raw RGB screenshot 사용

실제 사람의 플레잉 human performance 사용/비교

⇒ 사람을 능가하는 케이스 존재

evolutionary policy search approach

HNeat : 화면 내 객체 감지

- HNeat Best : hand-engineered object detector
- HNeat Pixel : 특정 8개의 color channel 사용
- 이는 한 에피소드에서만 최고 점수를 목표로 하기 때문에, 랜덤에 일반화 X

⇒ DQN은 넓은 범위에서 성능을 보임

06 conclusion

- DL model → RL (Atari)
- raw pixel input 만을 사용
- online Q-learning + stochastic minibatch update with experience replay memory