TAVE 서기

|  |  |  |  |
| --- | --- | --- | --- |
| **서기 내용** | | | |
| **서기 일자** | 22.1.7 | **서기** | 이아현 |
| **주제** | 강화학습 | | |
| **시간** | 20:30 – 22:30 | **장소** | Zoom 미팅 |
| **스터디**  **인원** | 고성호, 권기호, 이아현, 서가을 : 시작    고성호, 권기호, 이아현, 서가을 : 종료 | | |
| **내용** | | | |
| **배운 내용** | **[목차]**  **Chapter 18. 강화학습**  **18.1 보상을 최적화하기 위한 학습**  **18.2 정책 탐색**  **18.3 OpenAI 짐**  **18.4 신경망 정책**  **18.5 행동 평가 : 신용 할당 문제**  **18.6 정책 그레이디언트**  **18.7 마르코프 결정 과정** | | |
| **Chapter18 강화학습**  **18.1 보상을 최적화하기 위한 학습**  **- 강화학습의 요소**  - **에이전트** : 강화학습에서 환경에 대해 특정 행동을 하고 학습하는 프로그램이나 로봇  - **관측** : 강화학습에서 학습하는 주체가 위치하거나 감지하고 있는 상태의 정보  - **환경** : 강화학습을 이용해 풀고자 하는 대상이나 문제  - **행동** : 에이전트가 상태 (S)에서 할 수 있는 행동  - **보상** : 에이전트가 행동을 취하면 환경은 그 행동에 대해 좋고 나쁨을 평가하고 평가 결과를 수치화해서 즉각적으로 에이전트에게 전달하는데, 이처럼 에이전트가 취한 행동에 대해 환경으로부터 좋고 나쁨의 평가를 수치적으로 받는 것을 의미  **- 강화학습** : 에이전트가 주어진 환경 안에서 관측과 행동을 통해 시행착오를 겪으며 보상의 장기간 기대치를 최대로 만드는 행동을 학습하는 것  **- 강화학습의 사례**  - 로봇 제어 프로그램  - 게임 프로그램  - 스마트 온도조절기  - 자동 매매 프로그램  - <https://www.youtube.com/watch?v=nReMgotclXU> (볼링)  **- 강화학습의 활용**  - 자율주행 자동차  - 추천 시스템  - 온라인 광고 시스템  - 이미지 분류 시스템  **18.2 정책 탐색**    **- 정책** : 소프트웨어 에이전트가 행동을 결정하기 위해 사용하는 알고리즘  (예시 : 관측을 입력으로 받고 수행할 행동을 출력하는 신경망)   1. 생각할 수 있는 어떤 알고리즘도 정책이 될 수 있음 2. 결정적인 필요 없음 3. 환경을 관측할 필요가 없는 경우도 존재   **에이전트가 어떻게 행동을 선택하는지 정의한다. (보통 π로 정의)**   |  |  | | --- | --- | | **확정적 정책** | **확률적 정책** | |  | Understanding Policy Iteration Algorithm For Reinforcement Learning | by  Abhishek Suran | Artificial Intelligence in Plain English | | 1. 각 상태당 정해진 행동이 존재 2. 에이전트는 각기 다른 상태에서 같은 행동을 선택할 수도 있다. 3. 에이전트는 어떠한 행동은 아예 선택하지 않을 수도 있다. 4. π(s)=a 5. 함수의 정의(자판기) 6. 죄를 지으면 정책상 감옥에 가야한다 | 1. 각 상태에서 각 행동을 취할 가능성을 확률로 표현 2. 0<π(a|s)≤1 3. ∑a∈A(s)π(a|s)=1 4. 죄를 지었지만 경찰에게 발견이 될 확률은? | | - **에이전트가 집으로 가야하는 문제**  **- 에이전트는 상하좌우로 움직일 수 있음** | | |  |  | | * 그림의 각 방향 화살표는 하나의 정책이 된다 * 각 화살표는 각 칸에서 어떤 방향으로 움직여야 하는지 알려준다 | * 각 상태별 행동의 선택 확률 결정 * 확률적 정책은 현재 상태가 행동을 선택할 모든 정보를 가지고 있다고 가정 |  * 로봇 진공청소기의 정책은 매 초마다 어떤 확률 p만큼 전진함 * (1-p)의 확률로 랜덤하게 회전함 * 회전 각도는 –r과 +r 사이의 랜덤한 각도   => 두 개의 정책 파라미터 : p, r  **1. 정책 탐색(policy search)**  - 파라미터에 많은 다른 값을 대입해보고 가장 성능이 좋은 조합을 고름  - 정책 공간(policy space)이 매우 클 경우 효율적이지 못함  **2. 유전 알고리즘(genetic algorithm)**  - 임의의 개수의 정책을 랜덤하게 생성해서 시도해본 다음, 성능이 낮은 정책을 버리고 성능이 좋은 정책을 살려 각각 자식 정책을 생산하게 함  - 1세대 정책 100개 => 성능 낮은 80개 버리고, 20개를 살려 각각 자식 정책 4개 생산  - 이를 여러 세대에 걸쳐 반복함  **3. 정책 그레이디언트(policy gradient, PG)**  - 정책 파라미터에 대한 보상의 그레이디언트를 평가해서 높은 보상의 방향을 따르는 그레이디언트로 파라미터를 수정하는 최적화 기법  - P를 증가 시켜 봄 => 로봇이 수집한 먼지 양이 증가했는가? => yes : p를 증가  => no : p를 감소  **18.3 OpenAI 짐**  **- 강화학습에서 어려운 점?**  : 에이전트를 훈련하기 위해 먼저 작업 환경을 마련해야 하는 것 -> 이에 훈련을 위한 **최소한의 시뮬레이션 환경이 필요**  **-** **OpenAI 짐** : 다양한 종류의 시뮬레이션 환경을 제공하는 툴킷  **1. CartPole 환경 만들기**    - cartPole 환경의 경우 각 관측은 네 개의 실수를 담은 1D 넘파이 배열임  **[카트의 수평 위치(0.0 = 중앙), 카트의 속도(양수는 우측 방향을 의미), 막대의 각도(0.0 = 수직), 막대의 각속도(양수는 시계 방향을 의미)]**  render()메소드를 활용해 환경을 출력해야 하는데, 한 가지 문제점이 있음  -> Carpole과 같은 환경들은 사용자의 display에 접근해 창을 띄워 환경에 대한 정보를 나타내야 하는데, 본 스터디에서 사용한colab은 가상환경을 사용하는 headlesss server임 -> Xvfb, Xdummy 같은 가짜 X server를 install 해야함. 본 스터디에서는 Xvfb(X virtual framebuffer) server를 사용할 것임      - render() 메서드에서 반환된 렌더링된 이미지를 넘파이 배열로 받으려면 **mode=”rgb\_array”**로 지정해야 함(특이하게 CartPole 환경은 스크린으로도 렌더링을 출력)    - **env.action\_space**를 통해 가능한 행동이 몇 개인지 파악할 수가 있음  - 0과 1 두개의 행동이 가능하고 각각 왼쪽 가속, 오른쪽 가속에 해당함    - **step() 메서드**는 주어진 행동을 실행하고 네 가지 값을 반환함   * **obs** : action을 취함으로서 얻은 관측값들을 뜻한다. obs[1]이 음수가 되었으므로 카트가 오른쪽으로 움직이게 될 것이다. obs[2]를 보면 막대는 오른쪽 방향으로 아직 기울어져 있다. 하지만 각속도가 음수가 되었으므로 obs[3] < 0 다음 스텝 이후에는 왼쪽으로 기울어질 가능성이 크다. * **reward** : agent의 목표는 카트에서 막대를 오랫동안 떨어트리지 않는 것이므로 어떤 action(오른쪽이던, 왼쪽이던)을 취해도 reward는 1이 될 것이다. * **done** : 이 값이 True면 이 eposide가 done인 상태가 된다(종료상태). 막대가 너무 기울어지거나 화면 밖으로 나갈 경우 혹은 정해진 최대 step인 200 step이 넘어갈 경우 episode는 종료된다. episode 종료 이후 꼭 reset을 통해 값을 초기화 해줘야 한다. * **info** : 현재 cartpole 환경에서는 info가 비어있지만, 다른 환경의 경우 agent 생명이 몇 번 남았는지 등의 정보가 담긴다.   - 한 환경을 다 사용했다면 **close() 메서드를 호출해 자원을 반납해야 함**.    - 간단한 정책(막대가 왼쪽으로 기울어지면 카트를 왼쪽으로 가속하고 오른쪽으로 기울어지면 오른쪽으로 가속)을 하드코딩함  - 이 정책으로 episode 500번 실행해서 얻은 평균 보상을 확인  => 500번을 시도해도 72번의 step을 넘지 못 함 (좋지 않은 결과)  **18.4 신경망 정책**  - 관측을 입력으로 받고 실행할 행동에 대한 확률을 추정하여 출력하는 신경망  => 추정된 확률에 따라 랜덤하게 행동을 선택할 것  - 왜 가장 높은 점수의 행동을 그냥 선택하지 않고 신경망이 만든 확률을 기반으로 **랜덤하게 행동을 선택**할까?  => 이는 에이전트가 새로운 행동을 탐험하는 것과 잘 할 수 있는 행동을 활용하는 것 사이에 균형을 맞추게 함  - 어떤 상태가 숨겨져 있다면? 과거의 행동과 관측도 고려  - 잡음이 있을 경우 : 지난 관측 몇 개를 사용하는 것이 좋음  **tf.keras를 사용하여 신경망 정책을 구현하는 코드**    - 간단한 Sequential 모델을 사용해 정책 네트워크를 정의  - 입력의 개수 : 관측 공간의 크기  - 하나의 확률이 필요하므로 시그모이드 활성화 함수를 사용한 하나의 출력 뉴런을 둠  - 가능한 행동이 두 개보다 많으면 행동마다 하나의 출력 뉴런을 두고 소프트맥스 활성화 함수를 사용해야 함  **18.5 행동 평가 : 신용 할당 문제**  - **신용 할당 문제** : 에이전트가 보상을 받았을 때 어떤 행동 덕분인지(혹은 탓인지) 알기가 어려움  - 이를 해결하기 위해 흔히 사용되는 전략은 행동이 일어난 후 각 단계마다 **할인계수(감마)**를 적용한 보상을 모두 합하여 행동을 평가하는 것이 있음  - 할인된 보상의 합을 **대가**라고 부름    - 할인계수가 0에 가까우면 먼 미래의 보상이 현재의 보상만큼 중요하게 취급 X  - 할인계수가 1에 가까울수록 미래의 보상은 현재의 보상보다 중요함  - 전형적인 할인 계수의 값은 0.9 ~ 0.99  - 진짜 좋은 행동 뒤에 운이 나쁘게 금방 넘어질 경우! → 낮은 Return값 반환  **=> 많은 횟수를 반복하여 평균적으로 좋은 행동과 나쁜 행동의 대가를 정규화하여 행동을 평가**  **18.6 정책 그레이디언트**  **- 정책 그레이디언트** : 높은 보상을 얻는 방향의 그레이디언트를 따르도록 정책의 파라미터를 최적화하는 알고리즘  **- Reinforce 알고리즘** : 가장 많이 사용하는 알고리즘임  1. 신경망 정책이 일단 여러 번 게임을 플레이 해보고, 매 스텝마다 선택된 행동이 더 높은 가능성을 가지도록 만드는 그레이디언트 계산  2. 에피소드를 몇 번 실행한 다음, 각 행동의 이익을 계산  3. 그 이익이 양수면 계산한 그레이디언트 적용 / 음수면 그것과 반대의 그레이디언트 적용  4. 마지막으로 모든 결과 그레이디언트 벡터를 평균 내어서 경사 하강법 스텝 적용  → Trial - Error 방식            - discount\_rewards() 함수 : 정확히 기대한 값이 반환  - discount\_and\_normalize\_rewards() : 두 에피소드의 각 행동에 대해 정규화 된 행동 이익을 반환    => 더 크고 복잡한 문제에는 잘 적용되지 못 함(샘플 효율성이 좋지 X)  - 액터-크리틱 알고리즘 같은 더 강력한 알고리즘도 있음  - 에이전트가 각 상태에 대한 할인된 미래의 대가를 추정하도록 학습됨. 또는 각 상태에 있는 각 행동에 대한 할인된 미래의 대가를 추정하도록 학습. 그리고 이 지식을 사용하여 어떻게 행동할지 결정하는데 이는 먼저 **마르코프 결정 과정**을 알아야 함  **18.7 마르코프 결정 과정**   |  |  | | --- | --- | | **마르코프 연쇄** | **마르코프 결정 과정** | |  |  | | - 메모리가 없는 확률 과정  - 정해진 개수의 상태를 가지고 있으며,  각 스텝마다 한 상태에서 다른 상태로 랜덤하게 전이  - 상태 s에서 s’로 전이하기 위한 확률은 고정되어 있으며, 과거 상태에는 상관없이 (s, s’) 쌍에만 의존  - 종료 상태 : 다양한 상태를 여러 번 오갈 수 있지만 결국 수렴하는 상태 | - 각 스텝에서 에이전트는 여러 가능한 행동 중 하나를 선택할 수 있고, 전이 확률을 선택된 행동에 따라 달라짐,  또한 어떤 상태 전이는 보상을 반환함  - 에이전트의 목적은 시간이 지남에 따라 보상을 최대화하기 위한 정책을 찾는 것  - 최적의 상태 가치 V\*(s) : 에이전트가 상태 s에 도달한 후 최적으로 행동한다고 가정하고 평균적으로 기대할 수 있는 할인된 미래 보상의 합 |   **- 벨먼 최적 방정식**    T는 Transpose의 약자로 에이전트가 행동 a를 선택했을 때 상태 s에서 s’로 전이될 확률을 의미하며, 오른쪽은 그때 에이전트가 받을 수 있는 보상을 의미함. 여기에 s’로 이동 이후에 받을 미래의 보상 또는 최적으로 이동한다고 가정하고 s’의 최적의 상태의 가치에 할인계수를 곱한 것을 더함.  **- 가치 반복 알고리즘** : 정책을 평가할 때 유용하지만 정책을 알려주지는 않음 -> 최적의 정책을 이미 알고 있는 경우에 사용    **- Q-가치 반복 알고리즘** : 상태와 행동의 쌍 (s, a)을 가지고 그에 대한 가치를 추정함 -> 최적의 정책을 구할 수 있음    **- 최적의 정책** : 에이전트가 상태 s에 도달했을 때 가장 높은 Q-값을 가진 행동을 선택하면 됨    **- Q-가치 반복 알고리즘을 코드로 표현**      상태 s0에서는 행동 a0을 선택하고, 상태 s1에서는 행동 a0을 선택하고, 상태 s2에서는 행동 a1을 선택할 때 최적임을 의미함  **- 할인계수를 0.95로 바꾸면?**    할인 계수를 0.95로 높이면 0, 2, 1이 나오게 되는데 이는 상태 s1에서 최선의 행동은 불을 건너는 것임  **=> 미래 보상에 더 가치를 둘수록 미래의 행복을 위해 당장의 고통을 견디려 하기 때문임** | | |
| **과제할당** | 18.8 박제윤  18.9 서가을  18.10 권기호  18.11 고성호  18.12 이문기  18.13 안세윤 | | |
| **특이사항** | 없음 | | |
| **비고** | 없음 | | |