TAVE 서기

|  |  |  |  |
| --- | --- | --- | --- |
| **서기 내용** | | | |
| **서기 일자** | 22.1.9 | **서기** | 8기 고성호 |
| **주제** | 강화학습 | | |
| **시간** | 21:00 ~23:00 | **장소** | Zoom 미팅 |
| **스터디**  **인원** | 고성호, 권기호, 이아현, 서가을 : 시작    고성호, 권기호, 이아현, 서가을 : 종료 | | |
| **내용** | | | |
| **배운 내용** | **Chapter 18 강화학습**  **18.8 시간차 학습**  **18.9 Q-learning**  **18.10 심층 Q-learning 구현하기**  **18.11 심층 Q-learning의 변종**  **18.13 그 외 유명한 강화학습 알고리즘** | | |
| **18.8 시간차 학습(Temporal Difference Learning : TD Learning)** \* 독립적인 행동으로 이뤄진 강화 학습 문제는 보통 Marcov Decision Process로 modeling가능하다. 하지만, 문제점이 있는데 초기에 Agent들이 transfer probability T(s, a, s')를 알지 못하고, Reward $R(s, a, s')$에 대해 알지 못한다는 점이다. 보상에 대해 알기 위해서는 적어도 한 번은 각 state와 transfer를 경험해야 한다. 또한 transfer probability에 대해 신뢰할 만한 추정을 얻으려면 여러 번 경험을 해야 한다.  \* TD learning은 Q-value iteration algorithm과 비슷하지만, Agent가 MDP에 대해 일부만 알고 있는 경우를 다룬다. Agent는 탐험 정책(exploaration policy)을 사용해 MDP를 탐험한다. 탐험이 진행될 수록 TD learning algorithm이 실제로 관측된 값들(transfer probability, reward)에 근거해 상태 가치의 추정값을 update한다.    \* 여기서 alpha는 learning rate이다.  \* r + gamma V\_k(s')는 TD target이다.  \* delta\_k (s, r, s')는 TD error라고 부른다.  \* 이 식의 첫 번째 형태를 더 간단히 쓰는 법은 아래 표기법을 사용하는 것이다.    \* 이 표기법을 사용하면 첫 번째 TD learning algorithm 식을 다음과 같이 변환할 수 있다.    \* 각 state s에서 algorithm은 agent가 이 state를 떠났을 때의 (당장의 reward)와 (나중에 기대할 수 있는 reward)의 moving average를 저장한다. **18.9 Q-Learning** \* Q learning 또한 transfer probability와 reward를 초기에 알지 못한 상황에서 학습이 진행되는 model-free RL(초기에 MDP에 대한 정보가 주어지지 않는 RL)이다. Q-value iteration algorithm을 사용하고, Agent의 플레이를 보고 점진적으로 Q-value 추정을 해가는 모델이다. 이후 정확한 Q-value estimation을 얻게 될 때 optimal policy는 가장 높은 Q-value를 가지는 action을 선택하는 것으로 설정한다(탐욕적 정책).    \* 각 state-action (s, a) 쌍마다 action a를 선택해 state s를 떠났을 때(다음 state s'는 random하게 choice한다.) 받을 수 있는 reward r과 discount된 미래 reward의 합(현재까지 추정된 Q value들을 참고해 random하게 선택된 다음 state s'의 가장 보상이 큰 action을 선택)을 더한 moving average를 저장한다. 여기서 미래 보상의 합은 target policy가 이후로 최적으로 행동한다고 가정하고 다음 state s'에 대한 Q-value estimation의 최댓값(state s'에서 취할 수 있는 모든 action과 다음 state조합에서 가장 보상이 큰 조합)을 선택한다. **18.9.1 탐험 정책**epsilon - greedy 탐색 \* 바로 전에 사용한 random한 탐색 정책에서 random하게 MDP를 충분히 탐색하기 위해서는 많은 반복이 소요된 것을 확인할 수 있었다. 이를 보완하기 위해 사용되는 탐험 정책이 epsilon - greedy 정책이다. epsilon의 확률로 random하게 다음 탐색을 선택하거나, 1-epsilon의 확률로 그 순간 가장 최선인 것으로 행동한다. epsilon- greedy policy는 Q-value estimate가 점점 정확해지면서 더 관심있는 부분을 살펴보게 되지만, 한 편으로 MDP의 알려지지 않은 부분을 random 하게 epsilon 의 확률로 탐색하게 된다. 보통 이 epsilon 값은 1.0과 같이 높게 잡고 0.05정도까지 점점 감소시키는 것이 일반적이다. Q - value estimate에 보너스 추가   \* N(s', a')는 상태 s'에서 행동 a'를 선택한 횟수를 count한다.  \* f(Q, N)은 f(Q, N) = Q + kappa / (1+N) 과 같은 탐험함수이다. 여기서 kappa는 agent가 알려지지 않은 곳에 얼마나 흥미를 느끼는지 나타내는 파라미터이다. **18.9.2 근사 Q-learning과 Deep Q-learning**   \* Q-learning의 주요한 문제는 방대한 state , action이 존재하는 state space를 대상으로 사용하기 어렵다는 점이다. 예를들어 위 팩맨의 경우를 보면 팩맨이 먹을 수 있는 먹이는 150개이다. 먹이는 먹혀서 존재하지 않을수도 있고 아직 안 먹혀서 존재할수도 있다. 약 10^45 개의 state가 존재한다. 팩맨과 유령이 있을 수 있는 위치의 경우의 수 까지 더하면 가능한 state는 지구에 있는 원자의 수 보다도 많아지기 때문에 탐색 방법을 어떤 것을 사용하는지와 상관 없이 Q-value로 기록할 수 있는 방법이 없다.  \* 이러한 Q-learning의 단점을 보완하기 위해 등장한 것이 근사 Q-learning이다. 근사 Q-learning은 어떤 state-action (s,a) 쌍의 Q-value를 근사하는 Q(s, a)를 적절한 개수의 파라미터를 사용해 찾는 방법을 사용한다. 오랫동안 state에서 뽑아 낸 feature들을 선형 조합하는 방식을 사용한다.  \* 일반적인 근사 Q-learning보다 더 성능이 좋았던 방법은 DNN기반의 Deep Q-Network(이하 DQN)를 활용한 Deep Q-learning(이하 DQL)이다. DQL은 근사 Q-learning 방법을 사용하되 Q-value 추정에 DQN을 활용하게 된다.    \* DQL의 근사 Q - value 또한 state s 에서 action a를 선택했을 때 얻게 되는 관측된 보상 r과 할인된 미래 보상의 합을 통해 추정된다. 이 때 할인된 미래 보상을 구하는 방법이 Q-learning과 좀 다르다. **심층 Q-Learning 구현**   \* cartpole environment의 Q-Learning에 사용할 DQN이다. 이론적으로는 DQN이 (state, action)을 입력으로 받고 이에 대한 Q-value를 출력하는 것이 맞지만, 실제로 구현할 때는 state를 입력으로 받고 마지막 Dense layer의 unit을 action의 개수로 설정해 Q-value를 한 번에 출력하는 것이 효율적이다.    \* agent의 exploration policy로 epsilon greedy pollicy를 사용한다. epsilon 의 확률로 random하게 action을 선택하고 1- epsilon의 확률로 모델을 통해 Q\_value를 출력해 Q\_value가 큰 action을 선택한다.    \* replay buffer(=replay memory)를 사용해 학습이 최근 경험(여기서는 observation값)에 영향을 받지 않도록 설정한다. buffer는 deque으로 구성하였다. deque은 linked list로 구현되어 있고, 길이가 길어질수록 random access가 느려지기 때문에 큰 replay buffer가 필요하다면 circular buffer를 사용하면 된다.    \* replay buffer에서 experience를 random sampling하는 함수이다.    \* one step을 진행하는 함수이다.    \* training을 하는 함수이다.    \* 600 episode 동안 훈련을 진행하였다. 그 중 50개의 episode는 replay buffer를 채우기 위해 훈련을 시행하지 않는다.      \* 약 400epoch 이후부터는 어느 정도 성능이 잘 나오는 것을 알 수 있다.    \* 다른 random seed로 시행할 때 가끔 이런 그래프가 그려지기도 한다. 배운 것을 잊어버리고 성능이 갑자기 바닥으로 내려가는 현상인데, 이를 catastropic forgetting(최악의 망각)이라고 한다. 이를 해결하기 위해서는 replay buffer 크기를 늘리거나, learning rate를 감소시키는 등의 방법이 있다. **18.11 심층 Q-Learning의 변종**고정된 Q-value target   \* 기본 DQL에서 model은 prediction을 만들고, target을 설정하는데 모두 사용된다. 위 코드는 앞서 살펴봤던 DQL의 training step에 해당하는 함수이다. DQN에서 target\_Q\_values는 model의 prediction을 통해 생성되었고, Q\_values는 model이러한 feedback loop는 발산, 진동, 동결 등의 문제가 생길 수 있기 때문에 network를 불안정하게 만든다.  \* 떄문에 딥마인드 연구자들은 2013년 논문에서 두 개의 DQN을 사용하는 방법을 사용하였다. 첫 번째는 각 step에서 학습하고 agent를 움직이는 데 사용하는 online model과 두 번째는 target을 정의하기 위해서만 사용하는 target model이다.  target model은 online model의 단순 복사본으로, target model은 학습이 이뤄지지 않기 때문에 50 epoch마다 학습이 이뤄지는 online model의 가중치를 target model로 복사한다.  \* 이러한 구조는 target model이 online model보다 자주 update되지 않으므로 Q-value target이 안정적이고, feedback loop 문제를 완화한다. 더블 DQN \* 2015년 딥마인드 연구자들은 DQN 알고리즘을 개선한 Double DQN을 만들어 내었다.  \* Target model은 Q-value를 과대평가하는 경향이 있다. 예를 들어 모든 행동이 좋은 행동이라고 가정했을 때, 추정된 Q-value가 모두 같아야 하지만, 근삿값이기 때문에 약간의 차이가 발생하게 된다. 이 때 target model은 항상 가장 큰 Q-value를 선택하므로, 평균 Q-value보다 조금 더 큰 값을 선택하게 된다. 이 때 Q-value가 과대평가 된다.    \* 이를 방지하기 위해 다음 state의 최선의 action을 선택할 때 online model을 사용한다. target model은 최선의 action에 대한 Q-value를 추정할 때만 사용한다. **18.11.3 우선 순위 기반 경험 재생** \*우선순위 기반 경험 재생 또는 중요도 샘플링은 replay buffer에서 더 중요한 경험을 더 자주 sampling하는 방식을 뜻한다.  \*여기서 중요한 경험의 정도는 학습을 빠르게 만드는 정도로 측정한다. TD 오차 delta의 크기가 크면 전이가 놀랍다는 것을 의미하고, 전이에서 배울 가치가 있다는 것을 의미한다.  \*경험이 replay buffer에 저장될 때 최초 우선순위를 높게 설정하여 적어도 한 번은 sampling되도록 설정한다. 그 다음부터는 TD 오차 로 우선순위(확률을 나타내는 p가 아니다)를 설정한다. 이 때 p가 0이 되지 않도록 작은 상수를 더해준다.  \*우선순위가 p인 경험을 sampling할 확률 P는 에 비례한다. 여기서 zeta는 우선순위 sampling을 얼마나 탐욕적으로 할 것인지, 즉 우선순위만 고려해 sampling할 것인지에 대한 정도를 의미하는 parameter이다. zeta가 0이면 우선순위를 반영하지 않고 uniform한 확률로 sampling하게 되고, 1이라면 완전한 중요도 sampling이 된다. 논문에서는 zeta 값을 0.6으로 설정하였다.  \* 이렇게 replay buffer로부터 우선순위를 반영해 sampling한 경험은 훈련시에 낮은 가중치를 부여해 보상해줘야 한다. 이를 위해 훈련 가중치 w는 로 설정해야 한다. n은 replay buffer에 있는 경험의 개수이고, beta는 우선순위 sampling 편향을 얼마나 조정할지에 대한 parameter이다. beta가 0에 가까울 수록 조정하지 않는 것이 된다. 논문에서는 훈련에서 초기에 beta=0.4를  사용하고, 훈련 마지막에는 beta=1까지 선형적으로 증가시켰다. **18.11.4 Dueling DQN**    \* Dueling DQN에서 (s, a)의 Q value는 위와 같이 state s에서 최적으로 행동했을 때 얻을 수 있는 할인된 총 보상에 대한 기댓값을 나타내는 value function V(s)와 state s에서 action a를 선택했을 때 다른 action들을 선택했을 때 보다 갖게 되는 이점을 나타내는 A(s, a)로 구성되어 있다. 위 식에서 alpha, beta, theat는 network의 weight이다. 두 개의 함수를 통해 추정된 estimator가 Q value를 구성하기 때문에 Dueling DQN이라고 부른다.value function의 estimator는 scalar값을 가지고, advantage function의 estimator는 vector값을 가진다.  \* 위 식을 그대로 이용하지는 않고 다음과 같이 바꿔 사용한다. 최선의 행동이 a\*일 때 A(s, a\*)=0이기 때문에 위에서 살펴본 식에서 빼 주는 것이 가능하다.    \* 이를 코드로 구현하면 아래와 같다.     * 1. **그 외 유명한 강화 학습 알고리즘**   **액터-크리틱 알고리즘**  \*정책 그레이디언트와 심층 Q-네트워크를 결합한 강화 학습 알고리즘  \*정책 네트워크와 크리틱 네트워크를 사용  DQN : 에이전트의 경험을 통해 보통과 같이 훈련  정책 : 일반적인 정책 그레이디언트와 다르게 빠르게 훈련  \*여러 에피소드를 진행해서 각 행동의 가치를 추정하고, 각 행동의 할인된 미래 보상을 합하여 정규화하는 대신, 에이전트는 DQN이 추정한 행동 가치에 의존함 -> DQN의 도움을 받아 훈련하는 에이전트와 비슷  **A3C(Asynchronous Advantage Actor-Critic)**  \*비동기적 어드밴티지 액터크리틱으로, 2016년 구글 딥마인드가 발표한 \*알고리즘학습 데이터간의 상관관계를 깨기 위해 DQN에서와 같이 리플레이 메모리를 사용하는 것이 아니라, 샘플을 수집하는 여러 개의 ACTOR-LEARNER라는 에이전트를 사용하는 Actor-Critic 방법  장점  \*강화학습 문제에서 더 좋은 보상을 달성 가능  \*기존 방식과 동일한 액션 공간에서 연속적으로나 별도로 동작하는 것이 가능  \*학습 속도가 빠름  \*DGN보다 시간이 단축되고, 학습 성능이 뛰어남    1. 동일한 신경망 모델로 되어 있는 각 actor-learner가 서로 다른 환경에서 비동기적으로 일정 시간동안 행동하면서 샘플을 생성  2. 여러 actor-learner가 생성한 샘플들을 이용하여 신경망 모델을 학습시킴  3. 학습된 신경망 모델을 다시 actor-learne에 복사  리플레이 메모리를 사용하는 효과  **A2C(Advantage Actor-Critic)**  \*비동기성을 제거한 A3C 알고리즘의 변종  \*Actor는 상태가 주어졌을 때 행동을 결정하고, Critic은 상태의 가치를 평가  \*모든 모델이 동기적으로 업데이트됨 -> GPU 의 성능을 십분 활용해 큰 배치에 대해 그레이디언트 업데이터를 수행 가능  **SAC(Soft Actor-Critic)**  보상뿐만 아니라 행동의 엔트로피를 최대화하도록 학습하는 모델  -> 가능한 많은 보상을 얻으려 하지만 가능한 예측이 어렵도록 만듦  -> 환경을 탐색하여 훈련 속도를 높이고, DQN 이 불완전한 추정을 만들 때, 같은 행동을 반복해 실행하지 않도록 도움  \*놀라운 샘플링 효율성을 보임  **PPO(Proximal Policy Optimization)**  \*훈련을 불안정하게 만드는 큰 가중치 업데이트를 피하기 위해 손실 함수를 클리핑하는 A2C 기반의 알고리즘  \*TRPO알고리즘의 간소화 버전  **호기심 기반 탐색(Curiosity-Based Exploration)**  \*보상의 희소성 : 학습을 느리고 비효율적으로 만듦  \*해결 : 보상을 무시, 즉 에이전트가 순수한 호기심만으로 환경을 탐색하면 보상이 환경에서 오는 것이 아니라 에이전트 자체의 성질이 됨  How?  에이전트가 끊임없이 자신의 행동의 결과를 예측하려고 시도합니다. 그리고 결과가 예측과 맞지 않는 상황을 찾음 -> if, 결과가 예측 가능 다른 곳 으로 이동else에이전트가 제어할 방법이 없다는 것을 인지But, 시간이 지나면 이것도 예측 가능해짐. | | |
| **과제할당** | 없음 | | |
| **특이사항** | 없음 | | |
| **비고** | 없음 | | |