AIoT AutoCar Prime 으로 배우는

온디바이스 AI 프로그래밍

7. 인공지능

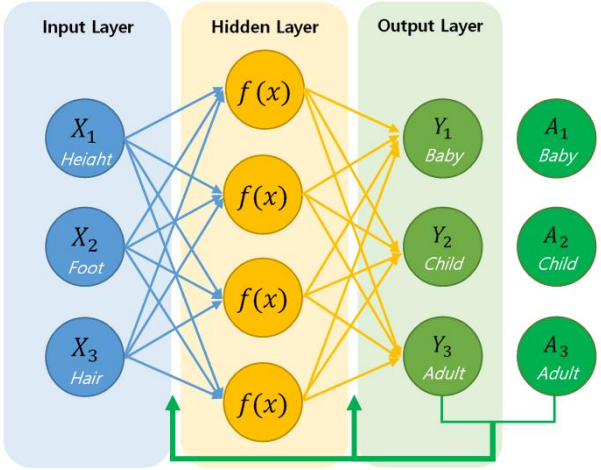
7.3 강화 학습

강화 학습

- 학습 초기에는 우연에 의존하여 학습. 점차 보상을 최대화하도록 학습
- 학습 초기 오차는 매우 높음. 시간이 지날수록 급격히 오차가 줄어듦
- 학습 데이터 가공이 필요 없고 필요한 선택만 하도록 학습
  - 고성능 환경에서 짧은 시간에 많은 학습 가능
- 초기 학습 모델에 의해 최종 학습 모델이 결정
- 학습 특징은 비지도 학습과 매우 유사하지만, 그 적용 대상이 다름
- 데이터 식별보다는 데이터 시뮬레이션 용도에 적합

Deep Q-Network

- 일반적인 딥러닝 알고리즘은 강화학습 불가능
  - 역전파 : 신경망 모델이 출력한 결과와 정답 데이터를 비교해 어떤 가중치 변수를 얼마나 조절할지 결정하고, 해당 가중치 변수에 전달하는 과정



Deep Q-Network

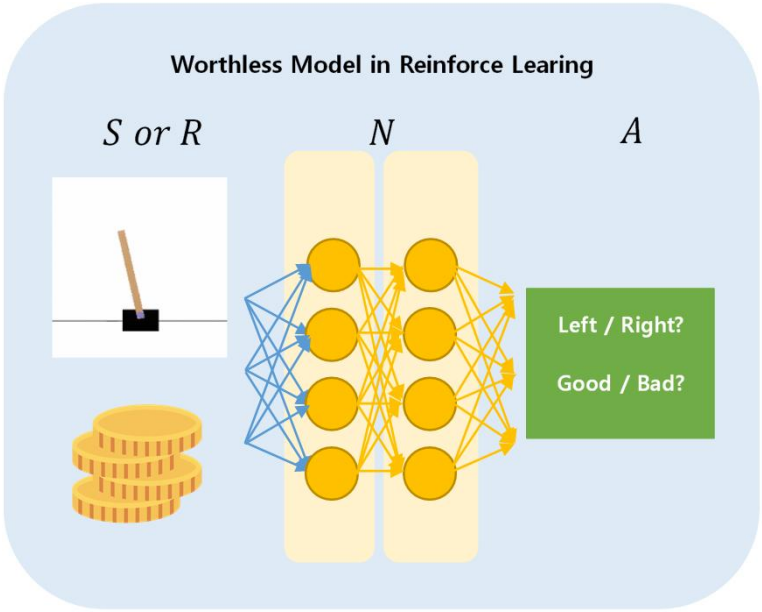
- 역전파를 하려면 ?
  - 모델의 출력 데이터와 비교할 정답 데이터가 같은 의미를 가진 데이터여야 함
  - 의미 있는 학습을 위해서는 가중치가 정답 데이터와 연관이 있어야 함

Deep Q-Network

- 강화 학습 과정 (벽돌 깨기를 가정)
  - 학습(train)하는 시점에는 '보상' 입력
  - 행동(run)하는 시점에는 '상태' 입력
  - 행동 입력과 학습 입력이 다르므로 입력에 대한 가중치는 하나로 수렴 불가
    - '상태'를 학습한 신경망 모델은 '보상'을 전혀 학습하지 못함
    - '보상'을 학습한 신경망 모델은 '상태'에 대한 행동을 전혀 하지 못함
  - 신경망의 핵심인 역전파 과정이 불가능한 학습

Deep Q-Network

- 강화 학습 과정 (벽돌 깨기를 가정)

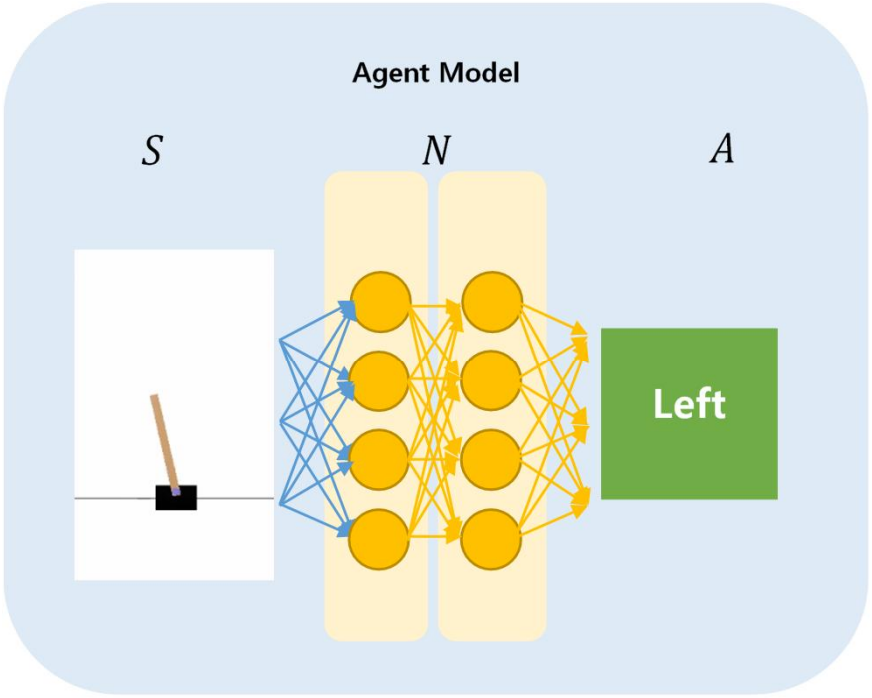


Deep Q-Network

- DQN 알고리즘
  - 게임을 진행하는 모델과 보상에 대해 학습하는 모델을 분리
  - 리플레이라는 개념을 추가해 문제 해결
    - 에이전트 : 게임을 플레이하는 행동 모델
    - 리플레이 : 이전에 진행했던 게임 상태 기록, 에이전트 행동 기록, 보상 기록
  - 에이전트의 행동에 대하여 게임은 어떤 보상을 줬는지 기록

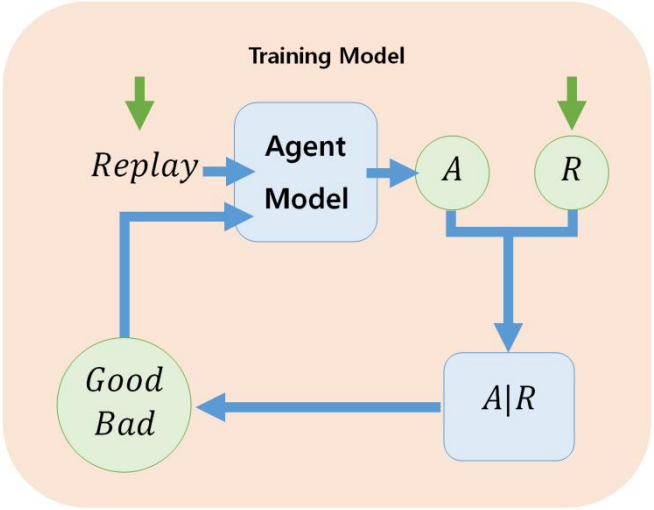
Deep Q-Network

- 에이전트 : 게임 상태S에서 신경망N에 입력되어 결과로 행동 A출력



Deep Q-Network

- 학습 모델
  - 보상 R이 주어졌을 때 리플레이 데이터를 불러와 에이전트에 입력
  - 출력된 행동 데이터와 보상 R을 비교해 에이전트의 행동 결과를 판단하여 가중치 조절



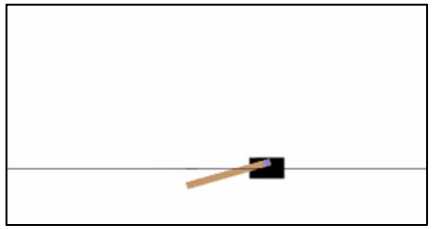
Deep Q-Network

- 강화 학습에 유용한 라이브러리인 OpenAI Gym을 사용하여 DQN을 실습
  - OpenAI Gym : 강화 학습을 위해 그래픽 요소, 행동과 보상 등을 제공
  - 알고리즘 구현에 집중할 수 있는 환경 제공
  - Pop.AI 라이브러리를 이용해 DQN을 구현

OpenAI Gym

- CartPole 예제
  - Gym에서 간단한 물리 엔진을 포함한 게임을 표시
  - 카트가 빠른 속도로 오른쪽으로 움직여 사라짐
  - 아무런 게임 규칙이 적용되지 않아 리셋 안됨

```
01:     import gym
02:
03:     env = gym.make('CartPole-v1')
04:     env.reset()
05:
06:     for _ in range(1000):
07:         env.render()
08:         env.step(env.action_space.sample()) # take a random action
09:     env.close()
```

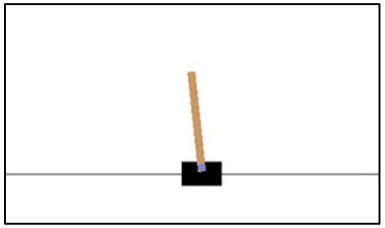


OpenAI Gym

- 게임의 규칙 적용
  - 게임 종료 : 막대가 15도 이상 기울거나 500 스텝 이상 진행될 경우 게임 종료
  - 보상 : 막대가 15도 이하로 직립한 스텝에서 1의 보상이 주어짐
  - 행동 : env.step 메소드에 True 또는 False를 입력해 좌우로 움직일 수 있음
  - 결과 : env.step 메소드를 실행하면 결과값들이 반환
  - state : 행동으로 인해 변화된 상태.
    - 리스트로 반환되며 차례대로 카트의 위치, 카트의 속도, 막대의 각도, 막대의 속도
  - reward : 해당 스텝의 보상. 0 또는 1이 반환
  - done : 게임이 종료되었는지 여부

OpenAI Gym

```
01:  import gym
02:  env = gym.make('CartPole-v1')
03:  for i_episode in range(20):
04:      state = env.reset()
05:      for t in range(100):
06:          env.render()
07:
08:          action = env.action_space.sample()
09:          state, reward, done, _ = env.step(action)
10:          if done:
11:              print("Episode finished after {} timesteps".format(t+1))
12:              break
13:  env.close()
```



OpenAI Gym – pop.AI

- DQN 객체의 파라미터
  - state_size: 상태 데이터의 크기. 최하위 차원의 크기 입력 (필수 입력)
  - hidden_size: 은닉층의 노드 수 (기본값: 5)
    - hidden_size를 조절하여 더 복잡한 학습이 가능하지만 크기가 커질수록 학습 속도는 느려짐
  - output_size: 결과 데이터의 크기. 최하위 차원의 크기 입력 (기본값: 1)
  - layer_level: 은닉층의 수 (기본값: 1)
    - layer_level을 조절하여 더 깊은 신경망을 만들어 복잡한 학습이 가능
    - 은닉층 차원이 커질수록 학습 속도는 느려지고 과적합 현상이 쉽게 발생할 가능성이 커짐
  - restore: 최근 모델에 이어서 학습할지에 대한 여부를 Boolean으로 입력 (기본값: False)
  - ckpt_name: 저장 및 불러올 모델 파일의 이름 (기본값: DQN)

OpenAI Gym – pop.AI

- Pop.AI 라이브러리 import
- DQN이라는 변수에 생성
  - AI모듈에서 DQN 객체의 state_size 파라미터를 4로 설정
- gym의 CartPole 환경을 생성하여 env변수에 저장

```
01:     import gym
02:     from pop import AI
03:
04:     DQN=AI.DQN(state_size=4)
05:
06:     env = gym.make('CartPole-v1')
```

OpenAI Gym – pop.AI

- 게임을 1000번 플레이하도록 for 루프 생성
- env객체의 reset 메소드로 환경을 초기화
- step과 total_reward를 0으로 초기화
- 리플레이 구현을 위해 상태, 보상, 행동을 기록할 리스트 생성

```
07:         for i_episode in range(1000):
08:             state = env.reset()
09:             step = 0
10:             total_reward = 0
11:
12:             states, rewards, actions=[], [], []
```

OpenAI Gym – pop.AI

- 한 게임을 진행할 때 게임이 종료될 때까지 행동하도록 무한 루프 생성
- env객체의 render메소드로 현재 상황을 그래픽으로 출력
- DQN객체의 run메소드에 현재 상태state를 입력하면 에이전트의 행동 값 출력
- 이 행동 값을 env.step에 입력하면
  - CartPole환경에서 행동으로 인해 변화된 상태와 보상 출력
- 상태, 보상, 행동 기록을 리플레이 리스트에 추가
- 총 보상과 스텝 갱신

OpenAI Gym – pop.AI

```
13:                 while True:
14:                     env.render()
15:
16:                     action=DQN.run([state])
17:
18:                     state, reward, done, _ = env.step(action)
19:
20:                     states.append(state)
21:                     rewards.append(reward)
22:                     actions.append(action)
23:
24:                     total_reward+=reward
25:                     step+=1
```

OpenAI Gym – pop.AI

- done 변수가 True인 경우
  - 한 스텝의 게임 종료
- 게임 종료 시
  - 이번 게임에서 몇 스텝까지 진행했는지 출력
  - DQN객체의 train메소드에 리플레이 리스트들을 입력하여 에이전트 가중치 조절
    - train메소드 파라미터 : states, rewards, actions

OpenAI Gym – pop.AI

```
26:                                 if done:
27:                                     print("Done after {} steps".format(step+1))
28:
29:                                     loss=DQN.train(states, rewards, actions)
30:                                     print('episode '+ str(i_episode + 1)+" reward : ", total_reward, ", loss : ",loss)
31:                                     break
```

- 게임이 모두 끝나면 CartPole 환경 종료

```
32:             env.close()
```

OpenAI Gym – pop.AI

전체 코드

```
01: import gym
02: from pop import AI
03:
04: DQN=AI.DQN(state_size=4)
05:
06: env = gym.make('CartPole-v1')
07: for i_episode in range(1000):
08:         state = env.reset()
09:         step = 0
10:         total_reward = 0
11:
12:         states, rewards, actions=[], [], []
13:         while True:
14:             env.render()
15:
16:             action=DQN.run([state])
17:
18:             state, reward, done, _ = env.step(action)
19:
20:             states.append(state)
21:             rewards.append(reward)
22:             actions.append(action)
23:
24:             total_reward+=reward
25:             step+=1
26:
27:             if done:
28:                 print("Done after {} steps".format(step+1))
29:
30:                 loss=DQN.train(states, rewards, actions)
31:                 print('episode '+ str(i_episode + 1)+" reward : ", total_reward, ", loss : ",loss)
32:                 break
33:
34: env.close()
```

내용 정리

- 강화 학습
  - 상황에 대한 데이터가 주어진 경우
  - 어떤 행동을 하면 그에 따른 보상 및 벌이 발생
  - 상황, 행동, 보상의 관계를 분석해 머신러닝 모델을 최적화 하는 방법
- DQN 알고리즘
  - 게임을 진행하는 모델과 보상을 학습하는 모델을 분리
  - 진행했던 게임 기록으로 보상이 발생한 행동을 학습하는 기법

내용 정리

- 에이전트
  - 게임을 플레이하는 행동 모델
- 리플레이
  - 이전에 진행했던 게임의 상태 기록, 에이전트 행동 기록, 보상 기록
- OpenAI Gym
  - 강화학습에 대해 쉽게 배우고 개발할 수 있도록 그래픽 환경을 제공하는 패키지

연습문제

- 문제 44. 다음 문장들을 읽고 빈 칸을 채워보세요.
  - A. 신경망 모델이 출력한 결과를 비교해 가중치를 조절하는 과정을 _____라 한다.
  - B. 강화 학습은 _____을 최대화하도록 학습한다.
  - C. DQN에서 게임을 플레이하는 모델을 _____라 한다.
  - D. DQN은 상태, 행동, 보상 기록을 통해 학습하고, 이 기록들을 _____라 한다.

연습문제

- 문제 45. 다음 코드는 Gym 라이브러리를 이용하여 강화 학습 환경을 구현한 코드입니다. 질문을 읽고 답해보세요.

```
01: import gym
02: from pop import Util
03:
04: env = gym.make('CartPole-v1')
05: for i_episode in range(20):
06:     state = env.reset()
07:     for t in range(100):
08:         img = env.render(mode = 'rgb_array')
09:         Util.imshow(“CartPole”, img, mode='RGB')
10:
11:         action = env.action_space.sample()
12:         state, reward, done, _ = env.step(action)
13:         if done:
14:             print("Episode finished after {} timesteps".format(t+1))
15:             break
16: env.close()
```

연습문제

- A. 코드를 실행해보고 게임 20회의 최대 보상값과 평균 보상값을 답해보세요.
- B. Pop.AI 라이브러리의 DQN 클래스를 사용해 보상을 최대화하는 코드를 작성해보세요.
- C. B에서 작성한 코드를 실행해 최대 보상값과 평균 보상값을 답해보세요.