18장 강화학습 (1부)

감사의 글

자료를 공개한 저자 오렐리앙 제롱과 강의자료를 지원한 한빛아카데미에게 진심어린 감사를 전합니다.

소개

- 머신러닝 분야에서 가장 흥미로우며 가장 오래된 분야
- 게임, 기계 제어 등 매우 다양한 애플리케이션에서 활용됨

주요 내용

- 강화학습(Reinforcement Learning) 소개

- 심층 강화학습의 주요 기법 두 가지
    - 정책 그레이디언트(policy gradients)
    - 심층 Q-네트워크
        - 마르코프 결정과정(Markov decision processes, MDPs)

- 실전 예제 1: 움직이는 카드(cart)에서 막대 균형잡기
    - OpenAI-Gym 소개
    - 정책 그레이디언트 활용

- 실전 예제 2: 브레이크아웃(Bradkout)이라는 아타리(Atari) 게임 플레이어 훈련시키기
    - TF-Agents 라이브러리 소개
    - 심층 Q-네트워크 활용

1절 보상 최적화 학습

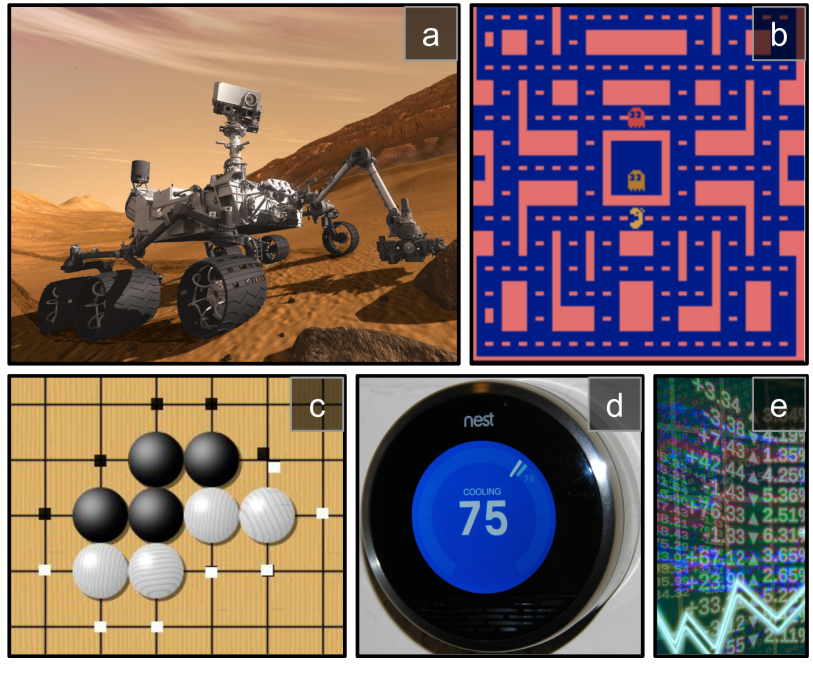
- 소프트웨어 에이전트(agent)가 주어진 환경에서 관측(observation) 후 행동(action)을 취하는 행위 반복
- 행위 결과에 따라 양(positive) 또는 음(negative)의 보상을 받음.
- 목표: 최대한의 (양의) 보상과 최소한의 (음의) 보상 받기

에이전트 학습 예제

주요 용어

- 에이전트
- 환경
- 관측
- 행동
- 보상

활용 사례



활용 사례 1: 로봇

- 에이전트: 로봇 제어 프로그램
- 환경: 실제 세상
- 관측: 카메라, 터치 센서 등을 이용하여 환경 관찰
- 행동: 모터를 구동하기 위해 시그널 전송
- 보상: 목적지에 도착할 때 양의 보상, 시간을 낭비하거나 잘못된 방향으로 향할 때 음의 보상 받음.

활용 사례 2: 미스 팩맨

- 에이전트: 미스 팩맨 제어 프로그램
- 환경: 아타리 게임 시뮬레이션
- 관측: 스크린샷
- 행동: 가능한 아홉 가지 조이스틱 위치
- 보상: 게임 점수

활용 사례 3: 바둑

- 미스 팩맨과 유사하게 작동

활용 사례 4: 온도조절기

- 에이전트: 온도제어 프로그램
- 환경: 주위 온도
- 관측: 온도
- 행동: 온도 조절
    - 사람의 요구를 예측하도록 학습된 결과에 따라 행동 취함
- 보상: 에너지를 절약하면 양의 보상, 사람이 온도를 조작할 필요가 발생하면 음의 보상을 받음

활용 사례 5: 주식 자동매매 프로그램

- 에이전트: 자동매매 프로그램
- 환경: 주식시장
- 관측: 주식시장 가격
- 행동: 매초 얼마나 사고팔아야 할지 결정
- 보상: 금전적 이익과 손실

주요 활용 영역

- 자율주행 자동차
- 추천 시스템
- 웹페이지 상에 광고배치
- 이미지 분류시스템 포커싱(주의집중) 영역 선정

보상의 종류

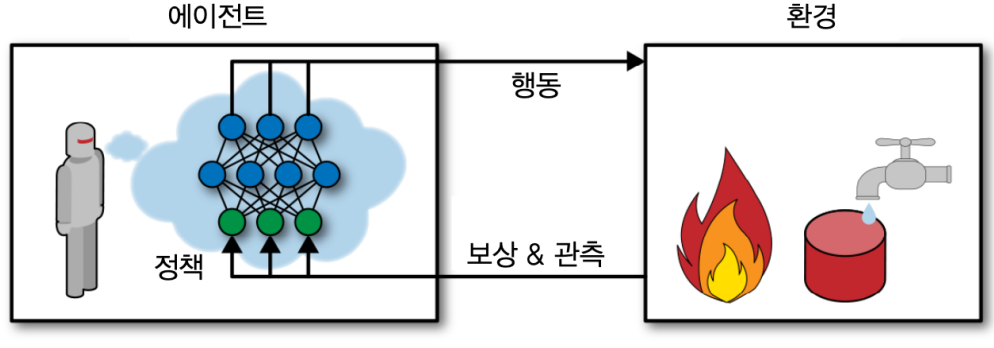
- 양의 보상: 기쁨
- 음의 보상: 아픔
- 양 또는 음의 보상이 전혀 없을 수도 있음
    - 미로 게임 에이전트는 타임스텝마다 음의 보상을 받기에 빠르게 탈출하도록 학습함.

2절 정책 탐색

정책

- 에이전트(agent)가 행동을 결정하기 위해 사용하는 알고리즘
  - 입력값: 관측
  - 출력: 행동

- 입력값이 필요없을 수도 있음
  - 예제: 30분 동안 수집한 먼지 양을 보상으로 받는 로봇진공청소기



확률적 정책

- 무작위성이 포함된 정책

- 예제: 로봇진공청소기
    - 매 초마다 지정된 확률 p 만큼 전진.
    - 무작위적으로 (1-p) 의 확률로 왼쪽 또는 오른쪽으로 회전하기.
    - 회전 각도는 -r 과 r 사이의 임의의 값.

정책 파라미터 탐색

- **정책 파라미터**: 정책에 사용되는 변경가능한 파라미터
    - 로봇진공청소기의 경우: p 와 r.

- **정책 탐색**: 가장 성능이 좋은 정책 마라미터 탐색

- **정책 공간**: 정책 파라미터가 취할 수 있는 값들의 집합

정책 탐색 기법: 유전 알고리즘

- 예제
    - 1세대: 정책 100개 랜덤하게 생성하여 사용해본 후 상위 20% 정책만 남김
    - 2세대: 남겨진 20개을 정책을 각각 4개씩 약간의 무작위성을 추가하여 복사.
    - 위 과정을 좋은 정책을 찾을 때까지 여러 세대에 걸쳐 반복.

- 주요 활용 예제: NEAT(NeuroEvolution of Augmenting Topologies) 알고리즘

정책 탐색 기법: 정책 그레이디언트(PG)

- 경사하강법과 유사한 방법
- 정책을 따른 결과인 보상이 최댓값을 갖도록 정책 파라미터들을 조금씩 변경하는 기법
- 보상이 최댓값을 갖도록 하기에 **경사상승법** 이라고도 불림.

- 예제: 로봇 진공청소기
    - p의 값을 조금 크게 한 경과 30분 동안 더 많은 먼지를 수집할 경우 p의 값을 좀 더 키움. 그렇지 않으면 p의 값을 조금 줄임.

- PG에 대해 이번 장에서 자세히 다룰 것임.

- PG를 TF(텐서플로우)로 구현하려면 에이전트가 활동할 환경을 먼저 세팅해야 함. 여기서는 **Open AI Gym** 사용.

3절 Open AI Gym

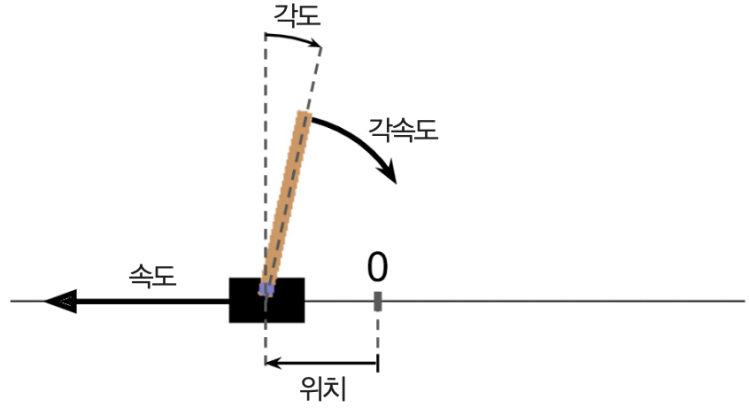
- 아타리 게임, 보드게임, 2D/3D 물리적 시물레이션 등을 위한 시뮬레이션 환경 제공
- 시뮬레이션 환경 내에서 에이전트에 대한 다양한 정책을 훈련하고 비교할 수 있음.

설치

- 설치 요령은 책과 코랩 노트북 참조

CartPole 환경

- 카드 위에 놓인 막대가 넘어지지 않도록 오른쪽/왼쪽으로 가속할 수 있는 2D 시뮬레이션 환경



CartPole 실행

환경 리셋하기

- CartPole-v1 환경을 생성한 후 리셋하여 환경 초기화 함.

```
import gym
env = gym.make("CartPole-v1")
obs = env.reset() # 초기화된 환경 관측치 할당
```

- CartPole-v1의 경우 `reset()` 메서드는 아래 모양의 초기화된 환경 관측값을 반환함.

```
array([-0.01258566, -0.00156614, 0.04207708, -0.00180545])
```

  - 0번 인덱스: 카트의 위치 (0.0은 중앙)
  - 1번 인덱스: 카트의 이동 속도 (음수는 왼쪽 방향으로의 이동 의미)
  - 2번 인덱스: 막대(pole)의 기울어진 각도(0.0은 수직)
  - 3번 인덱스: 막대의 각속도(양수는 시계방향 의미)

- gym 이 제공하는 전체 시뮬레이션 리스트는 아래와 같이 확인:

```
gym.envs.registry.all()
```

환경 렌더링

- 환경을 화면에 출력하려면 `render()` 메서드 실행

- `render()` 메서드에서 반환된 이미지를 넘파이 배열로 받으려면 `mode="rgb_array"` 설정

```
img = env.render(mode="rgb_array")
```

- `img` 에는 아래 모양의 컬러 사진이 저장됨.

```
(800, 1200, 3) # img.shape
```

행동(actions)

- 가능한 행동은 다음과 같이 확인

```
env.action_space
```

- CartPole-v1의 경우 다음과 0과 1 두 종류의 행동이 가능:

```
Discrete(2)
```

    - 0: 왼쪽으로 가속하기
    - 1: 오른쪽으로 가속하기

- 환경에 따라 다른 행동, 심지어 연속적인 값, 즉, 부동소수점으로 표현되는 행동도 가능.

- 예제: 초기 상태에서 막대가 오른쪽으로 쓰러지고 있기에 오른쪽으로 가속하는 행동 한 번 실행

```
action = 1  # 오른쪽으로 (살짝) 가속
  obs, reward, done, info = env.step(action)
```

- 행동 결과는 아래와 같음:

```
# obs: 한 번 오른쪽으로 가속한 후 관측값
array([-0.01261699,  0.19292789,  0.04204097, -0.28092127])

# reward: CartPole의 경우 보상은 항상 1
1.0

# done: 게임 종료 여부, 즉, 막대가 쓰러졌는지 여부
False

# info: 에이전트 관련 기타 정보. CartPole의 경우 없음.
{}
```

정책 활용 예제: 하드 코딩

- 정책: 막대가 왼쪽으로 쓰러지면 왼쪽으로 가속, 오른쪽으로 쓰러지면 오른쪽으로 가속

```
def basic_policy(obs):
      angle = obs[2]
      return 0 if angle < 0 else 1
```

- 위 정책을 이용하여 에피소드 500번 시뮬레이션 실행.
    - 매 에피소드마다 얻는 보상 저장
    - 여기서 누적되는 보상은 게임이 종료될 때까지의 행동 실행횟수를 가리킴.

```
totals = []
for episode in range(500):
  episode_rewards = 0
  obs = env.reset()
  for step in range(200):
      action = basic_policy(obs)
      obs, reward, done, info = env.step(action)
      episode_rewards += reward
      if done:
          break
  totals.append(episode_rewards)
```

- 평균적으로 41.7회, 최대 68회 행동 실행함.

```
# np.mean(totals), np.std(totals), np.min(totals), np.max(totals)
  (41.718, 8.858356280936096, 24.0, 68.0)
```

- 막대가 오래 서있지 못하고 좌우로 심하게 흔들리면서 쓰러짐.

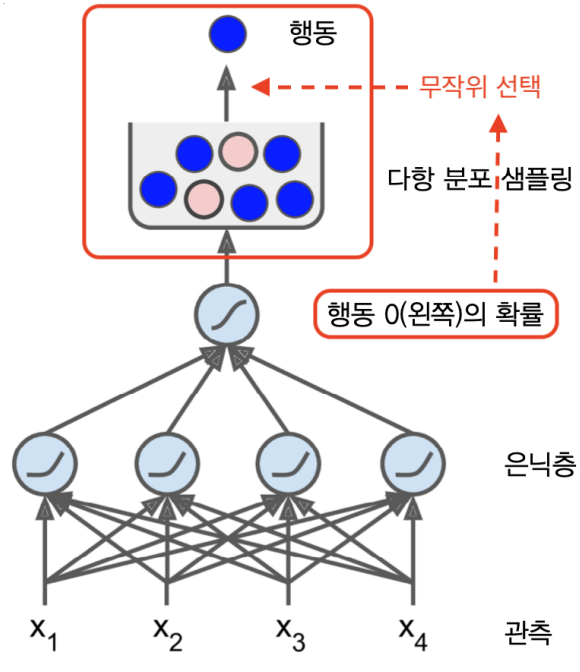
- 신경망을 활용한 정책 알고리즘을 이용하면 더 좋은 정책을 생성함.

4절 신경망 정책

- 관측을 입력받아 실행할 행동을 결정하는 데에 사용되는 값을 반환하는 신경망 정책함수 구현
- 신경망 정책함수의 반환값: 실행할 행동에 대한 확률

확률적 행동

- 실제 행동은 행동에 대한 확률에 의거하여 무작위적으로 선택
    - 이유: 새로운 활동에 대한 가능성을 열어 두었을 때 보다 나은 정책을 찾을 수도 있기 때문임.



행동 최종 결정

- 확률적으로 행동을 결정하는 것 이외에 과거의 행동과 관측을 행동을 결정하는 데에 활용할 수도 있음.

- CartPole 문제의 경우에는 해당되지 않음.

- 하지만 만약에 카트의 현재 위치만 관측된다면 현재 속도를 추정하기 위해 이전 관측도 활용해야 함.

예제

- tf.keras 를 활용한 신경망 정책 모델
    - 출력층: 한 개의 뉴런과 시그모이드 활성화 함수 사용
        - 좌우 움직임 이외의 다른 행동이 가능한 경우 그만큼의 뉴런 사용하며 최종적으로 소프트맥스 활성화 함수 활용
    - 출력값: 0과 1사이의 확률값 반환. 여기서는 왼쪽으로 행동할 확률 지정

```
n_inputs = 4 # 관측값 모양(env.observation_space.shape[0])

model = keras.models.Sequential([
    keras.layers.Dense(5, activation="elu", input_shape=[n_inputs]),
    keras.layers.Dense(1, activation="sigmoid"),
    ])
```

5절 행동 평가

지도학습 신경망 모델의 일반적인 학습법

- 손실함수(`loss`)를 기준으로 경사하강법(`optimizer`) 적용
- 평가지표
    - 회귀모델: (일반적으로) 손실 기준
    - 분류모델: 정확도(accuracy), ACU 등

- 회귀 모델

```
model.compile(loss="mse",
              optimizer="adam")
  model.fit()
```

- 분류 모델

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="nadam",
              metrics=["accuracy"])
  model.fit()
```

강화학습 신경망 모델 평가지표: 행동 이익

- 각 행동결정 단계에서의 가장 좋은 행동이 무엇인지 알고 있다면 일반적인 지도학습 활용 가능
    - `loss`를 예를 들어 추정된 확률과 타깃 확률 사이의 크로스 엔트로피로 정할 수 있음.

- 하지만 강화학습에서는 에이전트가 활용할 수 있는 것은 보상뿐임.

- 따라서 특정 행동의 영향력을 평가할 수 있는 기준이 요구됨.
    - 행동 평가 = 행동에 따른 보상 평가하기

신용할당 문제

- 특정 행동이 보상에 미치는 영향을 평가하기가 매우 어려움을 나타냄.
- 행동의 결과가 늦게 보상을 줄 수 있음.
- 또한 어떤 행동이 보상에 얼마나 영향을 미쳤는가를 파악하기 어려움.

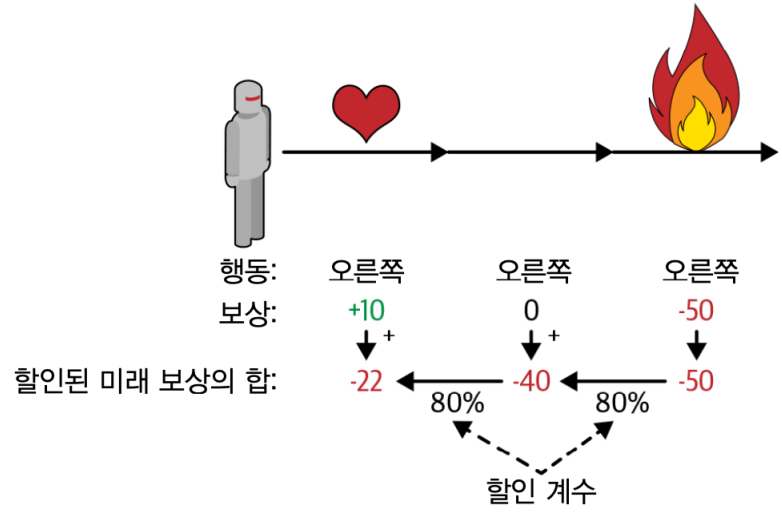
행동 대가(action return)

- 신용할당 문제의 해결 전략: 각 행동 결정단계마다 **할인계수**(discount factor) **감마**(γ) 를 적용한 보상을 모두 합하여 행동 평가

- **행동 대가**(action return): 할인된 미래 보상의 누적 합

- 예제
  - 아래 그림에서 에이전트가 오른쪽으로 세 번 움직이는 행동을 취함.
    - 첫째 스텝에서 10, 둘째 스텝에서 0, 셋째 스텝에서 -50의 보상을 받음.
  - 따라서 첫째 행동의 대가는 다음과 같이 계산됨:

$$10 + \gamma \cdot 0 + \gamma^2 \cdot (-50) = -22$$



할인계수

- 0과 1 사이의 값
- 1에 가까울 수록 보다 먼 미래의 보상이 보다 중요해짐.
- CartPole의 경우
    - 행동의 효과가 매우 짧은 기간안에 나타남.
    - 따라서 $\gamma = 0.95$ 가 적절해 보임. ($0.95^{13} \approx 0.5$)

행동이익(action advantage)

- 대가 정규화: 대가들의 평균과 표준편차를 계산한 후 표준점수로 변환하기

$$Z = \frac{X - \mu(X)}{\sigma(X)}$$

- 행동 이익: 게임을 충분히 많이 실행한 후 계산한 각 행동에 대한 대가를 정규화한 값
    - 많은 에피소드(시뮬레이션)를 실행하면서 얻은 각 행동에 대한 대가들의 평균값과 표준편차를 계산함.

행동 평가지표: 행동이익

- 양의 행동이익: 좋은 행동
- 음의 행동이익: 나쁜 행동

6절 정책 그레이디언트(Policy Gradients)

- PG: 높은 보상을 얻도록 신경망 정책 모델의 파라미터를 경사하강법으로 학습시키는 기법

REINFORCE 알고리즘

- 대표적인 PG 알고리즘

작동방식 요약

1. 신경망 정책 모델을 활용한 게임을 여러 번의 에피소드로 실행하면서 아래 데이터 저장
    A. 시뮬레이션의 매 스텝마다 파라미터별로 그레이디언트 계산해서 저장.
    B. 시뮬레이션의 매 스텝마다 보상을 확인해서 저장

2. 모든 에피소드 완료 후 매 스텝의 행동이익 계산

3. 매 스텝에 대해 행동이익을 저장된 파라미터 그레이디언트와 곱하기

4. 계산된 모든 그레이디언트의 평균값을 파라미터별로 계산

5. 계산된 파라미터별 그레이디언트의 평균값을 신경망 정책 모델에 경사하강법을 이용하여 적용

구현: 스텝 실행 결과 반환 함수

- 스텝을 한 번 실행할 때마다 관측과 그레이디언트 반환하기 함수

- 타깃확률(`y_target`): 0 또는 1
    - 정책 모델이 추천하는 방향을 타깃확률로 지정
    - 따라서 손실(`loss`)이 최소, 즉, 정책 모델이 가능한 좋은 확률로 행동을 추천하는 방향으로 유도함.

- 그레이디언트(`grads`): 실행된 스텝에서 각 파라미터에 대한 손실함수의 그레이디언트

```
def play_one_step(env, obs, model, loss_fn):
    with tf.GradientTape() as tape:
        left_proba = model(obs[np.newaxis])
        action = (tf.random.uniform([1, 1]) > left_proba)
        y_target = tf.constant([[1.]]) - tf.cast(action, tf.float32)
        loss = tf.reduce_mean(loss_fn(y_target, left_proba))
    grads = tape.gradient(loss, model.trainable_variables)
    obs, reward, done, info = env.step(int(action[0, 0].numpy()))
    return obs, reward, done, grads
```

구현: 여러 에피소드 실행 결과 반환 함수

- 에피소드 실행중에 발생하는 매 스텝의 결과인 보상과 그레이디언트를 리스트로 저장
- 모든 에피소드에 대해 반복

```
def play_multiple_episodes(env, n_episodes, n_max_steps, model, loss_fn):
    all_rewards = []
    all_grads = []
    for episode in range(n_episodes):
        current_rewards = []
        current_grads = []
        obs = env.reset()
        for step in range(n_max_steps):
            obs, reward, done, grads = play_one_step(env, obs, model, loss_fn)
            current_rewards.append(reward)
            current_grads.append(grads)
            if done:
                break
        all_rewards.append(current_rewards)
        all_grads.append(current_grads)
    return all_rewards, all_grads
```

구현: 행동이익 반환 함수

- `discount_rewards()` 함수
    - 하나의 에피소드 내에서 발생하는 스텝별 보상에 대한 행동대가(return) 계산

- `discount_and_normalize_rewards()` 함수
    - 모든 에피소드에 대해 스텝별 행동이익 계산

```
def discount_rewards(rewards, discount_rate):
    discounted = np.array(rewards)
    for step in range(len(rewards) - 2, -1, -1):
        discounted[step] += discounted[step + 1] * discount_rate
    return discounted

def discount_and_normalize_rewards(all_rewards, discount_rate):
    all_discounted_rewards = [discount_rewards(rewards, discount_rate)
                              for rewards in all_rewards]
    flat_rewards = np.concatenate(all_discounted_rewards)
    reward_mean = flat_rewards.mean()
    reward_std = flat_rewards.std()
    return [(discounted_rewards - reward_mean) / reward_std
            for discounted_rewards in all_discounted_rewards]
```

구현: 신경망 정책 모델 훈련

- `discount_rewards()` 함수
    - 하나의 에피소드 내에서 발생하는 스텝별 보상에 대한 행동대가(return) 계산

- `discount_and_normalize_rewards()` 함수
    - 모든 에피소드에 대해 스텝별 행동이익 계산

```
n_iterations = 150            # PG 적용을 150번 실행
n_episodes_per_update = 10    # 10번 에피소드마다 PG 적용
n_max_steps = 200             # 매 에피소드마다 최대 200번 행동
discount_rate = 0.95          # 할인계수

optimizer = keras.optimizers.Adam(lr=0.01)
loss_fn = keras.losses.binary_crossentropy  # 손실함수


model = keras.models.Sequential([
    keras.layers.Dense(5, activation="elu", input_shape=[4]),
    keras.layers.Dense(1, activation="sigmoid"),
    ])
```

```
env = gym.make("CartPole-v1")

# PG 적용: 150회
for iteration in range(n_iterations):
    # 에피소드 10번 실행
    all_rewards, all_grads = play_multiple_episodes(
        env, n_episodes_per_update, n_max_steps, model, loss_fn)
    all_final_rewards = discount_and_normalize_rewards(all_rewards,
                                                       discount_rate)
    all_mean_grads = []

    # 스텝별 그레이디언트에 스텝별 행동이익을 곱한 후 파라미터별 그레이디언트 평균값 계산
    for var_index in range(len(model.trainable_variables)):
        mean_grads = tf.reduce_mean(
            [final_reward * all_grads[episode_index][step][var_index]
             for episode_index, final_rewards in enumerate(all_final_rewards)
                 for step, final_reward in enumerate(final_rewards)], axis=0)
        all_mean_grads.append(mean_grads)

    # 계산된 파라미터별 그레이디언트를 기존의 파라미터에 더하는 방식으로 경사하강법 적용
    optimizer.apply_gradients(zip(all_mean_grads, model.trainable_variables))
```

결과

- 평균 보상이 200에 거의 가까울 정도로 학습이 잘됨(구글 코랩 노트북 참조).
- 하지만 학습을 위해 매우 많은 게임을 실행해야 함. 즉, 샘플 효율성이 매우 낮음.
- CartPole 용도의 정책 그레이디언트 알고리즘을 다른 문제에 활용하기에는 알고리즘이 너무 단순함.
- 하지만 나중에 짧게 소개할 Actor-Critic 알고리즘의 기초로 사용됨.