18장 강화학습 (2부)

감사의 글

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

7절 마르코프 결정과정

- PG 알고리즘: 보상을 증가시키기 위해 정책을 직접 최적화하는 방향으로 학습

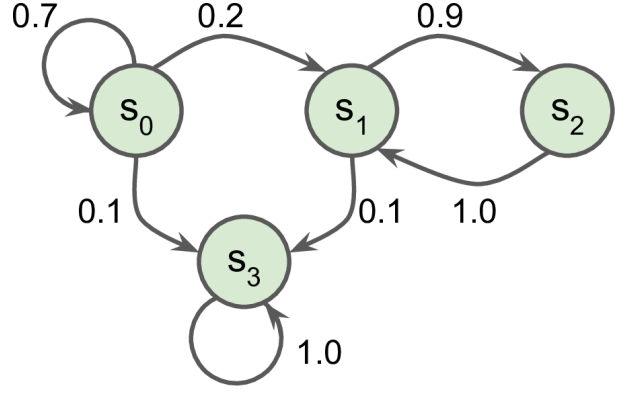
- 다른 알고리즘: PG 보다 덜 직접적으로 학습
    - 에이전트가 새로운 스텝을 실행하기 전의 (환경)상태에서 기대할 수 있는 대가를 추정하거나, 취할 수 있는 각각의 행동에 대한 대가를 추정함.
    - 예제:
        - 가치 반복 알고리즘
        - 시간차 학습
        - Q-러닝

- 마르코프 결정과정(Markov Decision Process, MDP)
    - 가치 반복 알고리즘, 시간차 학습, Q-러닝 등에 사용되는 행동 결정과정

마르코프 체인

- 20세기 초에 마르코프의 메모리 없는 확률과정 연구에 사용된 개념
    - 확률과정(stochastic process):
        - 확률공간에서 정의되는 확률변수들의 모임
        - 확률변수의 인덱스는 정수를 취하여 이산적일 수도 있고, 실수를 취하여 연속적일 수도 있음.
        - 확률변수 사이의 이동은 확률적으로 이루어짐.

- 예제
  - 확률변수: 상태(state)
    - S_0, S_1, S_2, S_3
  - 상태 사이의 이동확률: 두 개의 상태에만 의존함. (메모리 없음)
    - S_0 상태인 경우
      - 70%의 확률로 자신의 상태에 머무름
      - 20%의 확률로 S_1 상태로 이동
      - 10%의 확률로 S_3 상태로 이동
    - 기타 등등(아래 그림 참조)



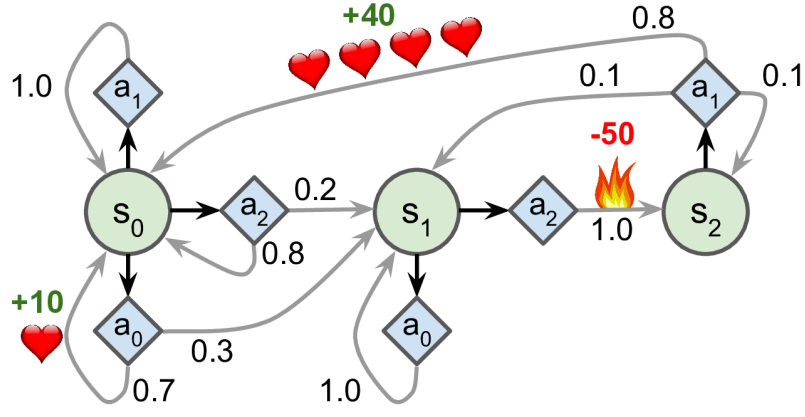
마르코프 결정과정(MDP)

- 1950년대에 Bellman에 의해 소개됨

- 마르코프 체인과 유사하지만, 각 상태에서 에이전트가 다양한 행동 중에 하나의 행동을 취할 수 있음.
    - 다른 상태로의 이동은 취한 행동에 의존함.
    - 다른 상태로 이동하면서 경우레 따라 보상을 받기도 함.

- 에이전트의 목표: 최대의 보상을 받는 정책 개발

MDP 예제



- 각 상태에서 시간이 흐르면서 최고의 보상을 받을 수 있는 전략은?
  - S_0: 행동 a_0 선택하기
  - S_1: 보상이 전혀 없는 행동 a_0 또는 위험하지만 궁극적으로 높은 보상의 가능성을 갖는 행동 a_2 선택 가능
  - S_2: 선택의 여지 없음

최적의 상태 가치(optimal state value)

- 정책 평가 용도로 중요

- $V^*(s)$: 상태 s 에서 에이전트가 최적의 행동을 선택한다고 가정했을 때 얻을 수 있는 할인된 미래 보상에 대한 기대치의 최댓값

$$V^*(s) = \max_a \sum_{s'} T(s, a, s') \cdot \left[R(s, a, s') + \gamma \cdot V(s') \right]$$

  - $T(s, a, s')$: 행동 a를 선택했을 때 상태 s에서 상태 s'로 전이될 확률
  - $R(s, a, s')$: 행동 a를 선택했을 때 상태 s에서 상태 s'로 이동되었을 때 받을 수 있는 보상
  - γ: 할인계수

- 동적계획법 활용: $V^*(s)$를 동적계획법으로 빠르게 계산 가능

$$V^*_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \cdot \left[R(s, a, s') + \gamma \cdot V_k(s') \right]$$

  - $V^*_k(s)$: 동적계획법 알고리즘의 k번째 반복에서 상태 s의 추정 상태 가치
  - $V^*_0(s) = 0$

Q-가치

- 최적의 상태-행동(state-action) 가치 계산

- $Q^*(s, a)$: 에이전트가 상태 s에 도착한 후에 행동 a를 선택할 때 얻을 수 있는 할인된 미래 보상에 대한 기대치
    - 에이전트를 위한 최적의 정책을 결정하는 데에 활용될 수 있음

- 동적계획법 활용: $Q^*(a, s)$를 동적계획법으로 빠르게 계산 가능

$$Q^*_{k+1}(a, s) \leftarrow \sum_{s'} T(s, a, s') \cdot \left[R(s, a, s') + \gamma \cdot \max_{a'} Q_k(s', a')\right]$$

    - $Q^*_0(a, s) = 0$

- $\pi^*(s)$: 상태 s에 도착했을 때 취할 수 있는 최선의 정책은 최고의 Q-가치를 갖는 행동 선택하기

$$\text{argmax}_a \, Q^*(s, a)$$

적용 예제

- $\pi^*()$ 함수를 위 그림에 있는 MDP에 적용하기

- MDP 정의

```
transition_probabilities = [ # 모양=[s, a, s']
        [[0.7, 0.3, 0.0], [1.0, 0.0, 0.0], [0.8, 0.2, 0.0]],
        [[0.0, 1.0, 0.0], None, [0.0, 0.0, 1.0]],
        [None, [0.8, 0.1, 0.1], None]]

rewards = [ # 모양=[s, a, s']
        [[+10, 0, 0], [0, 0, 0], [0, 0, 0]],
        [[0, 0, 0], [0, 0, 0], [0, 0, -50]],
        [[0, 0, 0], [+40, 0, 0], [0, 0, 0]]]

possible_actions = [[0, 1, 2], [0, 2], [1]]
```

- Q-가치 초기화

```
Q_values = np.full((3, 3), -np.inf) # 불가능한 행동: -np.inf

for state, actions in enumerate(possible_actions):
    Q_values[state, actions] = 0.0  # 가능한 행동: 0
```

- Q-가치 반복
    - 할인계수: gamma = 0.90

```
for iteration in range(50):
    Q_prev = Q_values.copy()
    for s in range(3):
        for a in possible_actions[s]:
            Q_values[s, a] = np.sum([
                    transition_probabilities[s][a][sp]
                    * (rewards[s][a][sp] + gamma * np.max(Q_prev[sp]))
                for sp in range(3)])
```

- 결과: Q-가치

```
array([[18.91891892, 17.02702702, 13.62162162],
       [ 0. , -inf, -4.87971488],
       [ -inf, 50.13365013, -inf]])
```

- gamma=0.90 인 경우: `np.argmax(Q_values, axis=1)`

```
array([0, 0, 1])
```

- gamma=0.95 인 경우: `np.argmax(Q_values, axis=1)`
  - 에이전트가 미래에 대한 보상을 보다 높게 간주함.
  - 상태 s_1에서 당장의 고통(불길, -50)을 감수하고 행동 a_2 선택

```
array([0, 2, 1])
```

8절 시간차 학습

- 학습 초기에 에이전트는 MDP에 대한 사전정보를 최소한만 알고 있음.
    - 가능한 상태와 가능한 행동은 안다고 가정.
    - 반면에 행동에 대한 보상과 전이확률은 모름.

- 시간차 학습(Time Difference Learning, TD 학습)을 통해 보상과 전이확률 추정
    - 보상: 한 번 이상 각각의 상태와 전이를 경험해서 확인
    - 전이 확률: 여러 번의 경험을 통해 추정

TD 알고리즘

- 탐험 정책: 완전히 랜덤한 정책 등을 이용하여 MDP를 탐험하는 정책

- 탐험이 진행하면서 실제로 관측된 전이와 보상에 근거하여 상태 가치 추정값 업데이트

$$V_{k+1}(s) \leftarrow (1-\alpha)V_k(s) + \alpha\Big(r + \gamma \cdot V_k(s')\Big)$$

- 아래와 같이 표현 가능:

  $$V_{k+1}(s) \leftarrow V_k(s) + \alpha \cdot \delta_k(s, r, s')$$

  단,

  $$\delta_k(s, r, s') = r + \gamma \cdot V_k(s') - V_k(s)$$

  - α: 학습률 (0.01 정도로 작게)
  - $r + \gamma \cdot V_k(s')$: TD 타깃
  - $\delta_k(s, r, s')$: TD 오차

- 아래 식을

$$V_{k+1}(s) \leftarrow (1-\alpha)V_k(s) + \alpha\Big(r + \gamma \cdot V_k(s')\Big)$$

다음과 같이 표현하는 것 선호됨

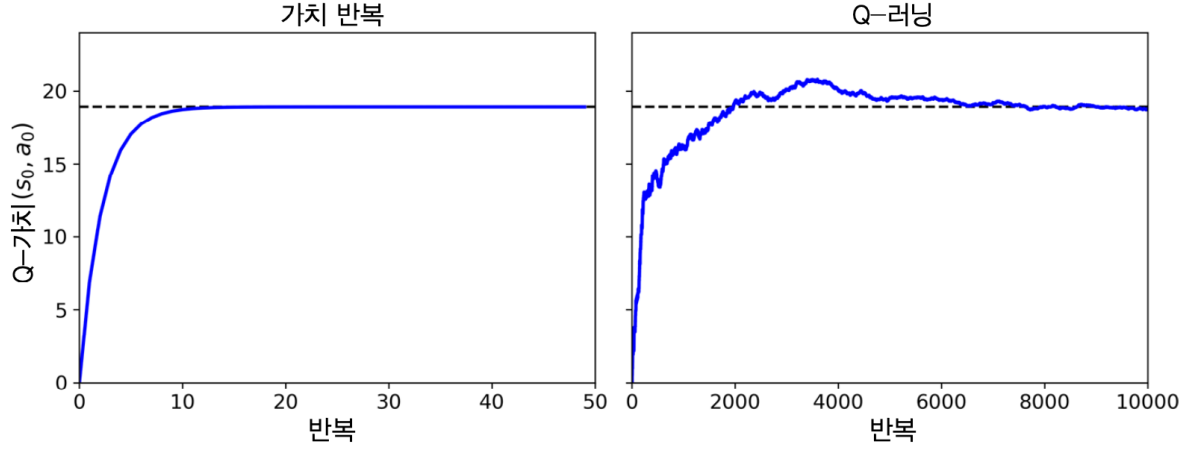
$$V(s) \underset{\alpha}{\leftarrow} r + \gamma \cdot V(s')$$

9절 Q-러닝

- TD 학습 방식을 Q-가치를 추청하는 데에 사용함.

$$Q(s,a) \underset{\alpha}{\leftarrow} r + \gamma \cdot \max_{a'} Q_(s', a')$$

- TD 학습을 통해 알아낸 보상과 전이확률을 이용하여 Q-러닝을 반복실행하면 최적의 Q-가치에 수렴함. 대신, 보다 훨씬 많은 반복이 요구됨.



off-policy 대 on-policy

- off-policy 알고리즘: 학습 과정중에 사용되는 정책이 반드시 최종 적으로 실행되는 정책이 아닐 수도 있는 알고리즘
    - 예제: Q-러닝 알고리즘

- on-policy 알고리즘: 학습 과정에 사용되는 정책이 항상 사용되는 알고리즘
    - 예제: PG 알고리즘

탐험 정책

- ε-탐욕 정책
- 탐험함수 적용 정책

ε-탐욕 정책

- 각 스텝에서 ε 확률로 랜덤하게 행동을 선택하거나 $(1-\varepsilon)$의 확률로 그 순간 가장 최선인 행동을 선택하는 정책

탐험함수 적용 정책

- 이전에 많이 시도하지 않았던 행동을 시도하도록 유도하는 정책

$$Q(s, a) \underset{\alpha}{\leftarrow} r + \gamma \cdot \max_{a'} f(Q_(s', a'), N(s', a'))$$

- $N(s', a')$: 상태 s'에서 행동 a'을 선택한 횟수
- $f(Q, N)$은 아래와 같은 탐험 함수

$$f(Q, N) = Q + \frac{\kappa}{1 + N}$$

  - κ: 탐험 호기심 정도를 나타내는 하이퍼파라미터

근사 Q-러닝과 심층 Q-러닝

근사 Q-러닝

- Q-러닝의 문제점: 중간규모 이상의 MDP에 적용하기 어려움. 이유: 너무 많은 상태의 수
- 임의의 상태-행동 (s, a)에 대한 근사 Q-가치 $Q_\theta(s, a)$를 대신 계산하여 활용

심층 Q-러닝

- 2013년 딥마인드가 제시한 심층신경망을 활용한 $Q_\theta(s, a)$ 추정 기법
- 심층 Q-네트워크(DQN, Deep Q-Network): Q-가치를 추정하기 위해 사용하는 DNN
- 심층 Q-러닝: 근사 Q-러닝을 위해 DQN을 활용하는 학습법

DQN 훈련 알고리즘

- 행동을 결정해야 하는 매 순간(상태)에 이전 경험을 바탕으로 정해진 타깃 Q-가치를 목표로 지도 학습 실행
    - 타깃 Q-가치는 정해진 배치(batch) 크기 만큼 무작위적으로 선택된 이전 경험으로 결정.

$$Q_{target}(s, a) = r + \gamma \cdot \max_{a'} Q_{\theta}(s', a')$$

- 다수의 에피소드를 통한 업데이트 반복 알고리즘으로 이해하려면 아래 식이 보다 적절함. $Q_{target}(s, a)$은 매 에피소드마다 업데이트됨.

$$Q_{target}(s, a) \leftarrow r + \gamma \cdot \max_{a'} Q_{\theta}(s', a')$$

10절 심층 Q-러닝 구현하기

구현: DQN 설정

```
env = gym.make("CartPole-v1")
input_shape = [4] # 관측 자료형 모양
n_outputs = 2 # 행동 종류 2개

# 출력층 뉴련 수: 2개.
# 즉, 현재 상태에서 취할 수 있는 모든 행동에 대한 확률값 반환
model = keras.models.Sequential([
    keras.layers.Dense(32, activation="elu", input_shape=input_shape),
    keras.layers.Dense(32, activation="elu"),
    keras.layers.Dense(n_outputs)
    ])
```

구현: ε-탐욕 정책 알고리즘

- DQN 모델 활용

```
def epsilon_greedy_policy(state, epsilon=0):
    if np.random.rand() < epsilon:
        return np.random.randint(2)
    else:
        Q_values = model.predict(state[np.newaxis])
        return np.argmax(Q_values[0])
```

구현: 지정된 크기의 경험 선택 알고리즘

```
def sample_experiences(batch_size):
    indices = np.random.randint(len(replay_memory), size=batch_size)
    batch = [replay_memory[index] for index in indices]
    states, actions, rewards, next_states, dones = [
        np.array([experience[field_index] for experience in batch])
        for field_index in range(5)]
    return states, actions, rewards, next_states, dones
```

구현: 스텝 실행 알고리즘

- DQN 모델을 활용하는 ε-탐욕 정책을 활용하여 한 스템 실행하기

```
def play_one_step(env, state, epsilon):
    action = epsilon_greedy_policy(state, epsilon)
    next_state, reward, done, info = env.step(action)
    replay_memory.append((state, action, reward, next_state, done))
    return next_state, reward, done, info
```

구현: 지정된 batch 크기의 경험을 이용하여 설정된 타깃 Q-가치를 활용한 경사하강법 실행 알고리즘

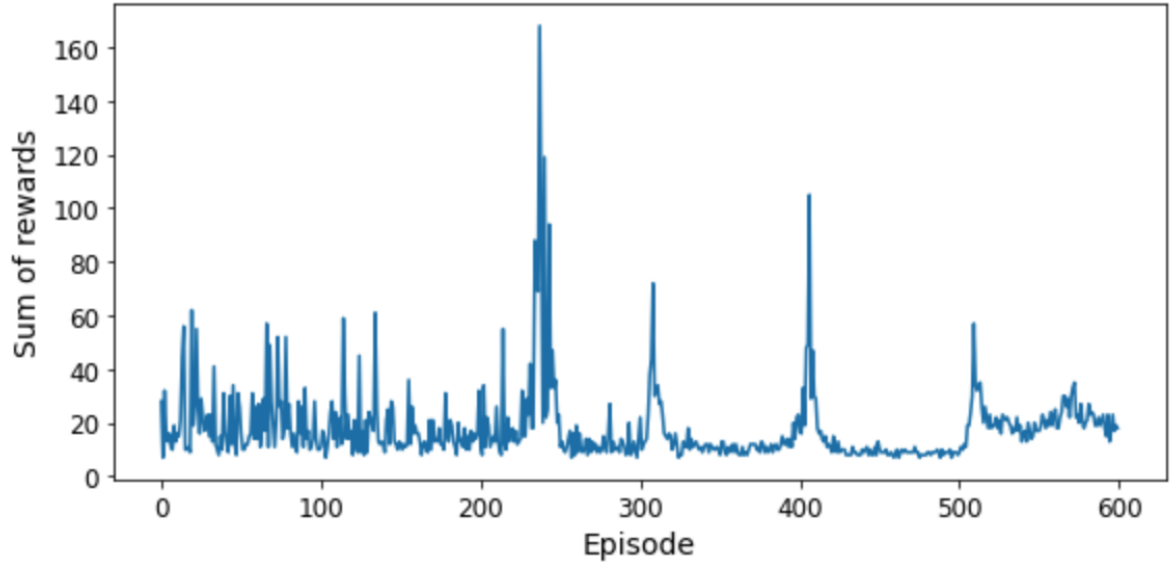
```
batch_size = 32
discount_rate = 0.95
optimizer = keras.optimizers.Adam(lr=1e-3)
loss_fn = keras.losses.mean_squared_error

def training_step(batch_size):
    experiences = sample_experiences(batch_size)
    states, actions, rewards, next_states, dones = experiences
    next_Q_values = model.predict(next_states)
    max_next_Q_values = np.max(next_Q_values, axis=1)
    target_Q_values = (rewards +
                       (1 - dones) * discount_rate * max_next_Q_values)
    target_Q_values = target_Q_values.reshape(-1, 1)
    mask = tf.one_hot(actions, n_outputs)
    with tf.GradientTape() as tape:
        all_Q_values = model(states)
        Q_values = tf.reduce_sum(all_Q_values * mask, axis=1, keepdims=True)
        loss = tf.reduce_mean(loss_fn(target_Q_values, Q_values))
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

CartPole의 DQN 모델 훈련

```
for episode in range(600):
    obs = env.reset()
    for step in range(200):
        epsilon = max(1 - episode / 500, 0.01)
        obs, reward, done, info = play_one_step(env, obs, epsilon)
        if done:
            break
    if episode > 50:
        training_step(batch_size)
```

- DQN 알고리즘의 학습곡선
    - 240 번의 에피소드 동안 발전이 전혀 없다가 갑자기 좋아짐.
    - 이전 Q-러닝 알고리즘보다 훨씬 빠르기 학습함
    - 단, 에피소드가 더 지마면 망각현상이 발생하여 성능이 오르락내리락 함. 이런 현상을 **재해성 망각**(catastrophic forgetting)이라 부름.



DQN 모델의 한계

- 훈련이 매우 어렵고 불안정한 경우가 일반적임

- 초기 하이퍼파라미터 값과 랜덤 시드에 영향을 많이 받음.즉, 운이 매우 좋아야 함.
    - 예제: CartPole의 경우 은닉층의 뉴런 수를 30 또는 34로 정하면 성능이 100 이상 나오지 않음.

- 그럼에도 불구하고 알파고와 아타리 게임 등 몇몇 실전 앱에서 훌륭하게 활용됨.

11절 심층 Q-러닝의 변종

- 앞서 설명한 CartPole의 DQN 모델은 너무 불안정함.

- 안정적이면서 빠른 훈련을 지원하는 심층 Q-러닝 알고리즘 존재

고정 Q-가치 타깃

- 앞서 설명한 모델은 하나의 모델이 타깃 Q-가치와 현재 상태에서의 예측을 함께 실행함. 따라서 자기 꼬리를 물려고 하는 강아지처럼 불안정한 피드백의 요인으로 작용하여 발산, 진동, 동경 등의 문제 발생 유발.

- 2013년 딥마인드 팀에서 아타리 게임 구현에 활용.

- 타깃 Q-가치를 정하는 모델을 별도로 사용
    - 온라인 모델: 각 스텝에서 학습하고 에이전트를 움직임에 사용되는 행동을 선택하는 모델
    - 타깃 모델: 타깃을 정의하기 위해서만 사용되는 모델. 온라인 모델의 복사본 사용.

```
next_Q_values = target.predict(next_states)
```

- 일정 숫자의 에피소드를 진행할 때마다 경사하강법을 적용하여 파라미터 조정함.

2013년 딥마인드 모델 (아타리 게임)

- 학습률: 0.00025
- 타깃 모델 업데이트 주기: 10,000 에피소드
- 경험 저장 버퍼 크기: 100만
- ϵ: 100만 스텝동안 1에서 0.1까지 매우 천천히 감소시킴.
- 매 에피소드에서의 스텝 수: 5천만

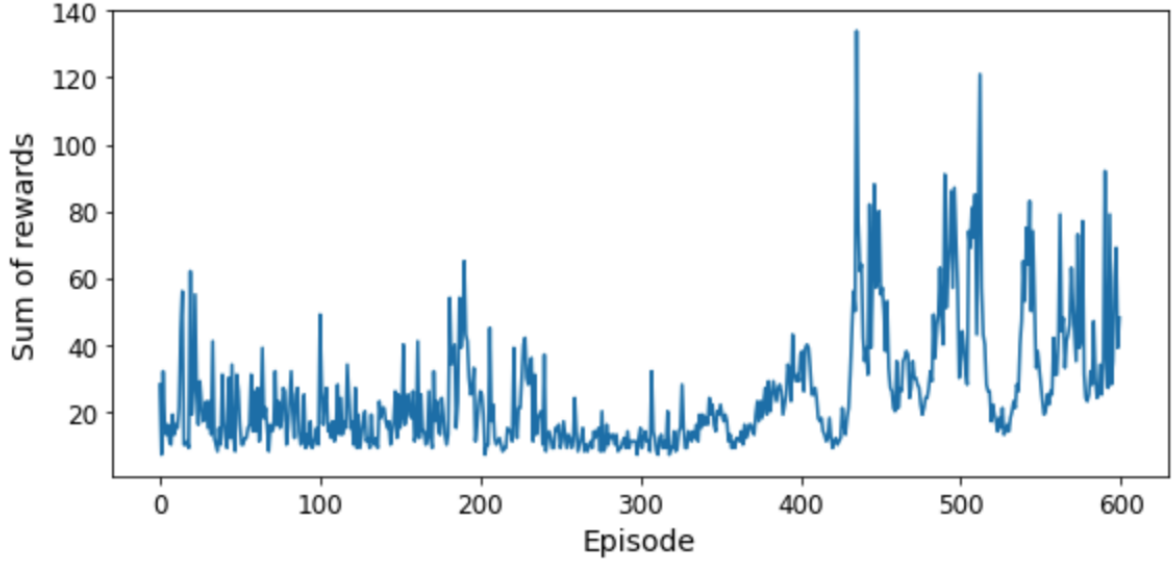
더블 DQN

- 2015년 딥마이드가 2013년 모델을 개선해서 제시함

- 2013년도 모델에서 사용된 타깃 모델의 타깃 Q-가치 계산 방법을 조금 수정함.
    - 온라인 모델이 선정한 최적의 행동을 기준으로 타깃 모델에서 타깃 Q-가치 계산

```
def training_step(batch_size):
    experiences = sample_experiences(batch_size)
    states, actions, rewards, next_states, dones = experiences
    next_Q_values = model.predict(next_states)
    best_next_actions = np.argmax(next_Q_values, axis=1)
    next_mask = tf.one_hot(best_next_actions, n_outputs).numpy()
    next_best_Q_values = (target.predict(next_states) * next_mask).sum(axis=1)
    target_Q_values = (rewards +
                       (1 - dones) * discount_rate * next_best_Q_values)
    target_Q_values = target_Q_values.reshape(-1, 1)
    mask = tf.one_hot(actions, n_outputs)
    [...] # 이전과 동일
```

- 보다 안정적으로 학습됨을 확인 할 수 있음.



우선순위 기반 경험 재생

- 2015년 딥마인드에서 제시한 개선 모델
- 타깃 Q-가치 계산에 사용되는 이전 경험을 무작위적으로 선택하는 것 대신에 중요한 경험을 보다 자주 선택하도록 유도하는 기법 적용
- 중요도 평가 기준: TD-오차
- 모델에 따라 중요도를 어떻게 활용할지 달라짐.

듀얼링 DQN

- 보통 DDQN 이라 부름.
    - 주의: 더블 DQN과 혼동하지 말 것.

- 2015년 딥마인드에서 제시한 개선 모델

기본 아이디어

- Q-가치가 아래처럼 계산될 수 있음에 주목함.

$$Q(s, a) = V(s) + A(s, a)$$

  - $V(s)$: 상태 s의 가치
  - $A(s, a)$: 상태 s에서 다른 가능한 모든 행동과 비교하여 행동 a를 취했을 때 얻는 이득 (advantage)

- 아래 식을 만족시키는 행동 a^* 존재

$$V(s) = Q(s, a^*) \quad \text{이고} \quad A(s, a^*) = 0$$

- DDQN 모델: 상태의 가치($V(s)$)와 모든 가능한 행동의 이득($A(s, a)$)을 계산하여 Q-가치 추정치 계산

```
K = keras.backend
input_states = keras.layers.Input(shape=[4])

hidden1 = keras.layers.Dense(32, activation="elu")(input_states)
hidden2 = keras.layers.Dense(32, activation="elu")(hidden1)

state_values = keras.layers.Dense(1)(hidden2)
raw_advantages = keras.layers.Dense(n_outputs)(hidden2)

advantages = raw_advantages - K.max(raw_advantages, axis=1, keepdims=True)

Q_values = state_values + advantages

model = keras.models.Model(inputs=[input_states], outputs=[Q_values])

target = keras.models.clone_model(model)
target.set_weights(model.get_weights())
```

강화학습 모델 활용법 소개

- 여러 모델을 조합하여 새로운 모델 생성하여 많이 활용됨.

- 딥마인드(2017년)
    - 6개의 기법을 조합하여 레인보우(Rainbow)라는 에이전트에 적용.

- 하지만 강화학습 모델을 훈련시키는 일은 일반적으로 매우 어려움.

- 따라서 TF-Agents 등 높은 확장성과 성능이 검증된 라이브러리 활용을 추천함.