13. 강화학습

강화학습은 원래 심리학의 개념으로 보상을 통해 행동에 '강화'를 받는 방식으로 학습하는 것을 말한다. 예를 들면 강아지가 재롱을 부리면(행동) 주인이 간식을 주고(보상) 이를 통해 강아지는 재롱을 더 많이 부리게 된다(강화). 기계학습에서 강화학습은 행동에 따르는 보상을 최대화하저는 방식으로 학습하는 것이다.

13.1. 마코프 결정 과정

강화학습은 보통 마코프 결정 과정(Markov Decision Process: 이하 MDP)라는 형식으로 문제를 정의하고 그 안에서 기대되는 보상의 합을 최대화하는 정책을 찾는 방식으로 문제를 푼다.

MDP에는 은 여러 가지 **상태**(state)가 있으며 상태에 따라 할 수 있는 행동도 다르고, 그에 따라 보상도 다르다고 가정한다. MAB는 상태의 종류가 1가지 밖에 없는 아주 단순한 형태로 볼 수 있다.

또한 이전 상태에서 어떤 행동을 하느냐에 따라 다음 상태가 달라질 수 있다. 이때 일정한 **전이확률**(transition probability)에 따라 똑같은 상태에 똑같은 행동을 하더라도 다음 상태는 다양하게 될 수 있다. 단순성을 위해 전이확률이 직전 상태와 행동에 좌우되는 **마코프 속성**(Markovian property)을 가정한다.

그러므로 보상을 최대화하기 위해서는 상태에 따라 행동을 달리해야 한다. 상태에 따른 행동을 **정책**(policy)이라고 한다. 즉, 강화학습은 기대보상을 최대화하는 정책 최적화 문제로 볼 수 있다.

강화학습이 한 번의 행동에 한 번의 보상으로 끝나는 것이 아니라 이후에 이어지는 상태들까지 고려해야 한다. 수업을 빼먹고 낮잠을 자면 단기적으로는 큰 보상을 꿀잠을 잘 수 있지만, 장기적으로는 나쁜 학점을 받을 수도 있다. 이럴 때 단기적 보상과 장기적 보상의 달리 평가할 수 있다. 똑같은 100만원이라도 당장 손 안에 있는 100만원과 1년 후의 100만원은 다른 것이다. 이 차이를 보완하기 위해 **할인**(discount)을 한다. 할인은 이자율의 반대 개념이다. 만약 연리 2%면 현재 100만원은 1년 후의 102만원과 같다. 반대로 말하면 1년후의 102만원을 2% 할인하면 현재의 100만원과 같아진다. 할인을 많이 할 수록 미래보다 현재를 우선하는 것이 된다.

강화학습에서는 한 상태에서 다른 상태로, 그 상태에서 또 다른 상태로, 여러 상태를 거쳐가게 된다. 이렇게 여러 상태에서 받은 보상들을 모두 더한 것을 **수익**(return)이라고 한다. 보상은 시점에 따라 할인을 하므로 수익은 할인된 보상들의 합이다.

13.2. OpenAI gym 소개

강화학습은 MAB와 마찬가지로 미리 데이터가 주어진 것이 아니라 행동에 따라 다른 데이터를 얻게 된다. MAB에서는 실습 때 모의실험을 위한 코드를 직접 작성했지만, 강화학습의 경우에는 OpenAI에서 개발한 `gym`이라는 라이브러리가 있어 시뮬레이션을 할 수 있게 하고 있다. `gym`은 고전 비디오 게임이나 물리 시뮬레이션 등을 제공한다.

`gym`은 Python 패키지 이므로 `pip`를 이용해 설치한다. 먼저 명령창을 연다. 윈도에서는 윈도+R을 눌러 `cmd`를 입력한다. 맥과 리눅스에서는 터미널을 연다. 이후 다음 명령으로 설치한다.

```
pip install gym
```

설치가 끝나면 R에서 `reticulate` 라이브러리를 이용해 Python과 연동을 해서 `gym`을 불러온다.

```
library(reticulate)
gym <- import('gym')
```

`gym`은 다양한 환경을 제공한다. 이번 시간에 해볼 환경은 '블랙잭(blackjack)'이라는 간단한 카드 게임이다.

```
env <- gym$make('Blackjack-v0')
```

블랙잭의 규칙은 다음과 같다.

- 2~9까지 카드는 숫자대로 점수를 계산한다. J, Q, K는 10으로 계산한다. A는 1 또는 11로 계산할 수

있다.
- 플레이어와 딜러는 각각 2장의 카드를 받는다
- 딜러는 2장 중 1장을 보이게 하고 1장은 덮어둔다
- 플레이어는 원할 때까지 카드를 더 받을 수 있다. 단, 점수가 21을 넘기면 진다.
- 플레이어가 카드 받기를 멈추면 딜러는 점수가 17을 넘을 때까지 계속 더 받는다.
- 딜러가 21을 넘기면 플레이어가 이긴다.
- 둘 다 21이하인 경우는 점수가 높은 쪽이 이긴다.

이제 블랙잭 게임을 실제로 해보자. `env$reset()` 함수를 이용해 게임을 시작할 수 있다. 블랙잭의 경우 게임을 시작하면 다음과 같은 리스트가 반환되는 데 이것이 게임의 현재 상태라고 할 수 있다.

```
[[1]]
[1] 9

[[2]]
[1] 10

[[3]]
[1] FALSE
```

`[[1]]` 값은 플레이어의 점수, `[[2]]`는 딜러가 공개한 카드의 점수다. `[[3]]`은 A를 가지고 있는지를 나타낸다.

`env$step(as.integer(0))`을 하면 카드를 그만 받는다. `env$step(as.integer(1))`을 하면 카드를 한 장 더 받는다. `step`을 할 경우 다음과 같은 리스트가 반환된다.

```
[[1]]
[[1]][[1]]
[1] 13

[[1]][[2]]
[1] 10

[[1]][[3]]
[1] FALSE

[[2]]
[1] 0

[[3]]
[1] FALSE

[[4]]
named list()
```

먼저 `[[1]]`은 행동을 통해 변경된 상태를 나타낸다. 각각의 내용은 `reset`의 경우와 같다. `[[2]]`는 이번에 얻은 보상이다. 게임 중에는 항상 0의 보상을 얻고, 게임이 끝났을 때만 이기면 1, 지면 -1의 보상을 얻는다. `[[3]]`이 `TRUE`면 게임이 종료, `FALSE`면 게임의 종료를 나타낸다. `[4]`는 추가 정보인데 여기서는 쓰지 않는다.

위에서 `[[1]]`, `[[2]]`, `[[3]]`, `[[4]]`의 용도는 환경의 종류와 상관없이동일하다. 다만 `[[1]]`의 내용은 환경에 따라 달라진다.

13.3. 강화학습의 종류

13.3.1. 진화 알고리즘

강화학습을 푸는 가장 단순한 방법은 진화 알고리즘이다. 여러 가지 정책으로 여러 번 시도를 한다. 가장 보상이 큰 정책을 남기고 다른 정책은 버린다. 보상이 가장 큰 정책을 조금씩 다르게 만들어서 다시 여러 가지 정책을 만든다. 이를 계속 반복하면 보상이 가장 큰 정책을 찾을 수도 있다. 진화 알고리즘의 장점은 무엇보다도 간단하다는 것이지만, 다른 문제에 진화 알고리즘을 적용한 경우와 마찬가지로 학습에 방향성이 없기 때문에 느리다는 것이 단점이다. 대부분 강화학습은 문제가 어렵고, MAB와 마찬가지로 현실에 적용해야 데이터를 얻을 수 있기 때문에 학습 속도가 느린 것은 대단히 치명적이다. 현실에서는 진화

알고리즘은 잘 쓰이지 않으나, 최근에는 진화 알고리즘의 효율성을 높이는 방향의 연구도 활발히 이뤄지고 있다.

13.3.2. 동적계획법

동적계획법(Dynamic Programming)은 큰 문제를 작은 문제로 나누어 푸는 여러 가지 기법들을 가리킨다. 강화학습에서는 현재 상태의 가치는 수익의 기대값과 같다. 그리고 수익은 그 상태에서 받을 수 있는 보상과 다음 상태의 수익을 할인한 것을 더한 것과 같다. 다음 상태의 수익의 기대값이 다음 상태의 가치이므로 아래와 같이 식이 전개된다.

$$v(s) = E[G_t|S_t = s] = E[R_{t+1} + \gamma G_{t+1}|S_t = s] = E[R_{t+1} + \gamma v(S_{t+1})|S_t = s] = \sum a\pi(a|s)\sum s',rp(s', r|s, a)[r + \gamma v(s')]$$

위의 식에서 $\pi(a|s)$는 상태 s에서 행동 a를 할 정책이다. 그리고 $p(s', r|s, a)$는 상태 s에서 행동 a를 했을 때 다음 상태 s'로 넘어가면서 보상 r을 받을 전이확률이다.

동적계획법의 장점은 모든 경우의 수를 끝까지 다 따라갈 필요 없다는 것이다. 현재 상태의 가치의 식은 다음 상태의 가치의 식을 포함하므로 각 상태의 가치만 구하면 된다.

동적계획법에서 정책은 각 상태에서 가치가 가장 큰 행동을 하는 탐욕 알고리즘으로 개선할 수 있다.

다만 동적계획법은 전이확률을 추정해야하고, 다음 상태와 보상의 경우의 수를 모두 더해야 하므로($\sum_{s',r}$) 계산량이 많다는 단점이 있다. 그래서 현실적으로는 잘 사용하지 않는다.

13.3.3. 몬테카를로법

강화학습에서 몬테카를로법은 다음과 같은 방법으로 한다. 정책을 정하고 처음부터 끝까지 한 에피소드(episode)를 정책대로 해본다. 에피소드란 다음과 같이 시작에서 끝까지 상태, 행동, 보상의 과정이다.

$$s_1 \rightarrow a_1 \rightarrow r_1 \rightarrow s_2 \rightarrow a_2 \rightarrow r_2 \rightarrow \cdots \rightarrow s_t \rightarrow a_t \rightarrow r_t$$

이렇게 거쳐간 상태를 기억해뒀다가, 각 상태의 평균 수익을 구하면 된다. 이를 반복하면 정책을 정확히 평가할 수 있다.

몬테카를로법은 MAB와 비슷한 문제가 있어서 해보지 않은 행동들의 가치는 알 수 있는 방법이 없다. 특히, 시작 상태가 충분히 다양하지 않다면 상태들의 가치를 추정하는데 편향이 생길 수 있다.

On-Policy vs. Off-Policy

몬테카를로법에서는 On-Policy 방법과 Off-Policy 방법 2가지가 있다. On-Policy 방법은 정책을 직접 적용해서 얻은 에피소드 데이터로 정책을 개선하는 방법이다. MAB에서 다뤘던 알고리즘들은 모두 On-Policy 방법에 속한다. On-Policy 방법은 상대적으로 단순하지만 검증되지 않은 정책을 직접 적용하는데 따른 위험성이 있다.

Off-Policy 방법은 행동 정책(behavior policy)과 대상 정책(target policy)을 나눈다. 행동 정책을 적용해서 얻은 에피소드 데이터로 대상 정책을 개선하는 방법이다. 기존의 데이터를 활용할 수 있다는 장점이 있지만 실제 적용이 훨씬 복잡해진다. 몬테카를로법에서 쓰는 Off Policy 방법으로는 중요도 샘플링(importance sampling)이 있다.

중요도 샘플링

중요도 샘플링은 대상 정책에서 행동의 확률 $\pi(A_t|S_t)$을 행동 정책에서 행동의 확률 $b(A_t|S_t)$로 나눈다. 이것을 전체 에피소드에 적용하면 중요도 샘플링 비(importance sampling ratio) $\rho_{t:T-1}$를 구할 수 있다.

$$\rho_{t:T-1} = T-1\prod k=t\pi(A_t|S_t)b(A_t|S_t)$$

그러면 행동 정책으로 얻은 수익이 G_t라고 하면 대상 정책으로는 $\rho_{t:T-1}G_t$라고 추정할 수 있게 된다.

좀 더 쉽게 설명해보자. 친구가 A, B, C의 행동을 순서대로 해서 100의 이익을 얻는 것을 보았다고 해보자. 그런데 똑같은 상황에 A를 할 가능성은 친구는 30%지만 나는 60%다. 그러면 나는 이익을 2배 더 얻을 수 있을 것이다. 그렇지만 같은 상황에서 C를 할 가능성은 친구는 40%지만 나는 20%다. 그러면 나는 이익을 1/2 밖에 얻지 못할 것이다. 이런 식으로 확률을 보정해서 이익을 추정하는 것이다.

실습: 입실론 탐욕 알고리즘

실습에서는 입실론 탐욕 알고리즘을 사용해보도록 하자. 일단 데이터 프레임의 형태로 가치표를 초기화 한다.

```
# 가치를 초기화
value.df = data.frame(
  player = rep(1:21, 10, each=4),
  dealer = rep(1:10, each=84),
  ace = rep(c(T, F), 210, each=2),
  action = rep(c(0, 1), 420),
  value = 0
  )
```

주어진 조건에서 가능한 행동을 찾는 함수를 만든다

```
available.actions <- function(value.df, status){
  cond <- value.df$player == status[[1]] &
    value.df$dealer == status[[2]] &
    value.df$ace == status[[3]]

  value.df$value[cond]
}
```

다음으로 입실론 탐욕 알고리즘으로 하나의 에피소드를 수집하는 함수를 작성한다.

```
# 입실론 탐욕 알고리즘으로 하나의 에피소드를 수집한다
epsilon.greedy <- function (env, value.df, epsilon = 0.1){

  episode = list()
  status <- env$reset()
  done = FALSE

  while(!done){

    if(runif(1) < epsilon){
      action <- sample(c(0, 1), 1)
    } else {
      action <- which.max(available.actions(value.df, status)) - 1
    }

    # 상태와 행동을 에피소드에 기록한다
    t <- length(episode) + 1
    status$action <- action
    episode[[t]] <- status

    # 행동을 실행하고 결과를 확인한다
    result <- env$step(as.integer(action))

    status <- result[[1]]
    status$reward <- result[[2]]
    done <- result[[3]]
  }

  list(episode = episode, G = result[[2]])
}
```

행동의 가치를 찾는 함수를 만든다

```
action.value.cond <- function(value.df, status){
```

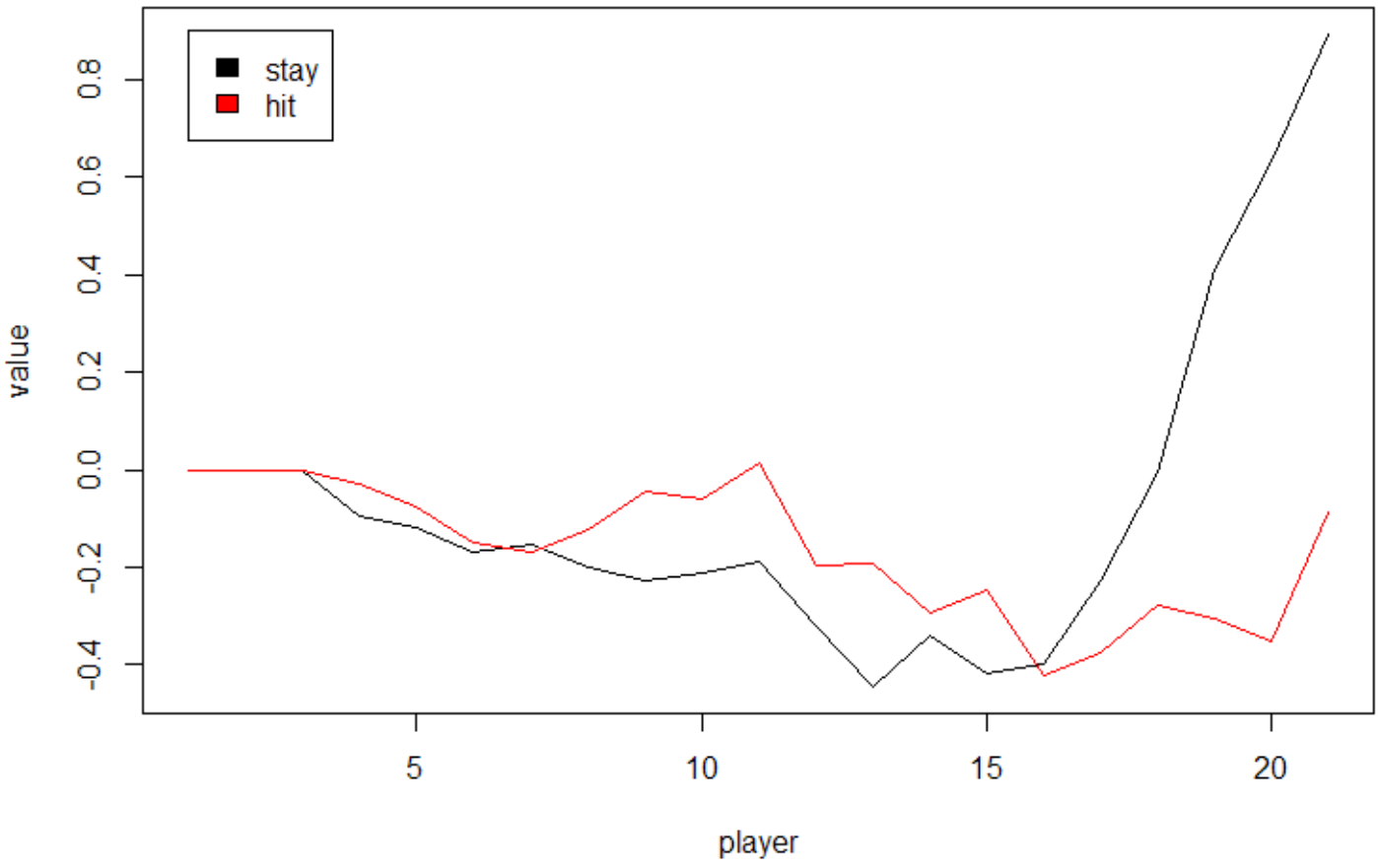
```
  value.df$player == status[[1]] &
    value.df$dealer == status[[2]] &
    value.df$ace == status[[3]] &
    value.df$action == status$action
}
```

이제 에피소드를 수집하고, 가치를 업데이트 하는 과정을 1000회 반복한다.

```
alpha = 0.1
for(i in 1:1000){
  # 한 게임의 에피소드를 수집한다
  ep <- epsilon.greedy(env, value.df)

  # 가치를 업데이트 한다
  for(status in ep$episode){
    cond <- action.value.cond(value.df, status)
    V <- value.df$value[cond]
    value.df$value[cond] <- V + alpha * (ep$G - V)
  }
}
```

결과를 시각화해보자.

```
library(dplyr)

# 플레이어 점수에 따라 더이상 받지 않는 행동(0)과 더 받는 행동(1)의 평균 가치를 구한다
v <- value.df %>% group_by(player, action) %>% summarise(mean(value))
v0 <- v %>% filter(action == 0)
v1 <- v %>% filter(action == 1)

# 그래프로 그려서 비교해본다
plot(v0$player, v0$`mean(value)`, typ='l', xlab = 'player', ylab = 'value')
lines(v1$player, v1$`mean(value)`, col=2)
legend('topleft', legend = c('stay', 'hit'), fill = c(1, 2))
```



위의 그래프를 보면 플레이어의 점수가 낮을 때는 카드를 더 받는 것(hit, 빨간 선)이, 점수가 높을 때는 더 받지 않는 것(stay, 검은 선)이 보다 더 가치가 높은 행동이다.

13.3.4. 시간차 학습

한 상태의 가치는 그 상태 이후 얻은 수익의 평균이다. MAB에서 다뤘듯이 평균은 다음과 같은 방법으로 바꿔서 구할 수 있다.

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

그런데 현 단계의 가치 $V(S_t)$, 수익 G_t과 보상 R_{t+1}, 그리고 다음 단계의 가치 $V(S_{t+1})$ 사이에는 다음과 같은 관계가 성립한다.

$$V(S_t) = E(G_t) = E(R_{t+1} + \gamma V(S_{t+1}))$$

이를 이용해서 다음과 같은 방법으로 가치를 추정하는 것을 시간차 학습(temporal difference learning) 이라고 한다.

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(St+1) - V(S_t)\Big]$$

시간차 학습의 장점은 수익 G_t이 필요없기 때문에 에피소드의 끝까지 기다리지 않아도 다음 상태로 넘어가기만 하면 바로 가치 추정을 바꿀 수 있다는 것이다.

SARSA

시간차 학습에서 실제로 행동을 하려면 상태 가치인 $V(S_t)$가 아닌 상태 행동 가치 $Q(S_t, A_t)$를 추정해야한다. 이를 SARSA라고 한다. 아래 식에 나오는 변수들(S_t, A_t, R_t, S_{t+1}, A_{t+1})에서 딴 이름이다.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

일상적인 표현을 이용해서 설명해보자. 만약 국민대에서 서울역까지 50분이면 간다고 생각하고 있었다. 현재의 정책은 국민대에서 길음역까지 택시를 타고, 길음역에서 서울역까지 지하철 4호선을 타는 것이다. 그런데 이번에 택시를 타니 길이 막혀서 길음역까지 30분이 걸렸다(R_t). 그러면 서울역까지 다 가보지 않더라도 길음역에서 서울역까지 4호선을 타고 걸리는 시간의 기존 추정치($Q(S_{t+1}, A_{t+1})$을 이용하면 국민대~서울역까지의 추정시간을 업데이트할 수 있다.

Q-Learning

Q-Learning은 시간차학습에서 사용하는 Off-Policy 학습 방법이다.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_t + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\Big]$$

SARSA와 달라진 부분은 실제 행동 A_{t+1} 대신 대상 정책에서 최선의 행동 $A' = \text{argmax}$을 사용한다는 점이다. 다시 국민대~서울역의 예를 사용해보자. 친구는 길음역까지 택시를 타고 길음역에서는 전철을 타지만, 나는 길음역에서 서울역까지 버스를 탄다. 그렇다면 내가 길음역까지 택시를 타는 것의 가치를 계산할 때는 친구가 행동한 것처럼 길음역~서울역에서 지하철을 타는 것(A_{t+1})이 아니라 나라면 했을 행동인 버스를 타는 것(A')을 기준으로 계산하게 된다.

Q-Learning은 행동 정책이 입실론 탐욕 알고리즘처럼 탐색을 하더라도 대상 정책은 탐색을 할 필요가 없기 때문에 더 효율적인 정책을 학습할 수 있다.

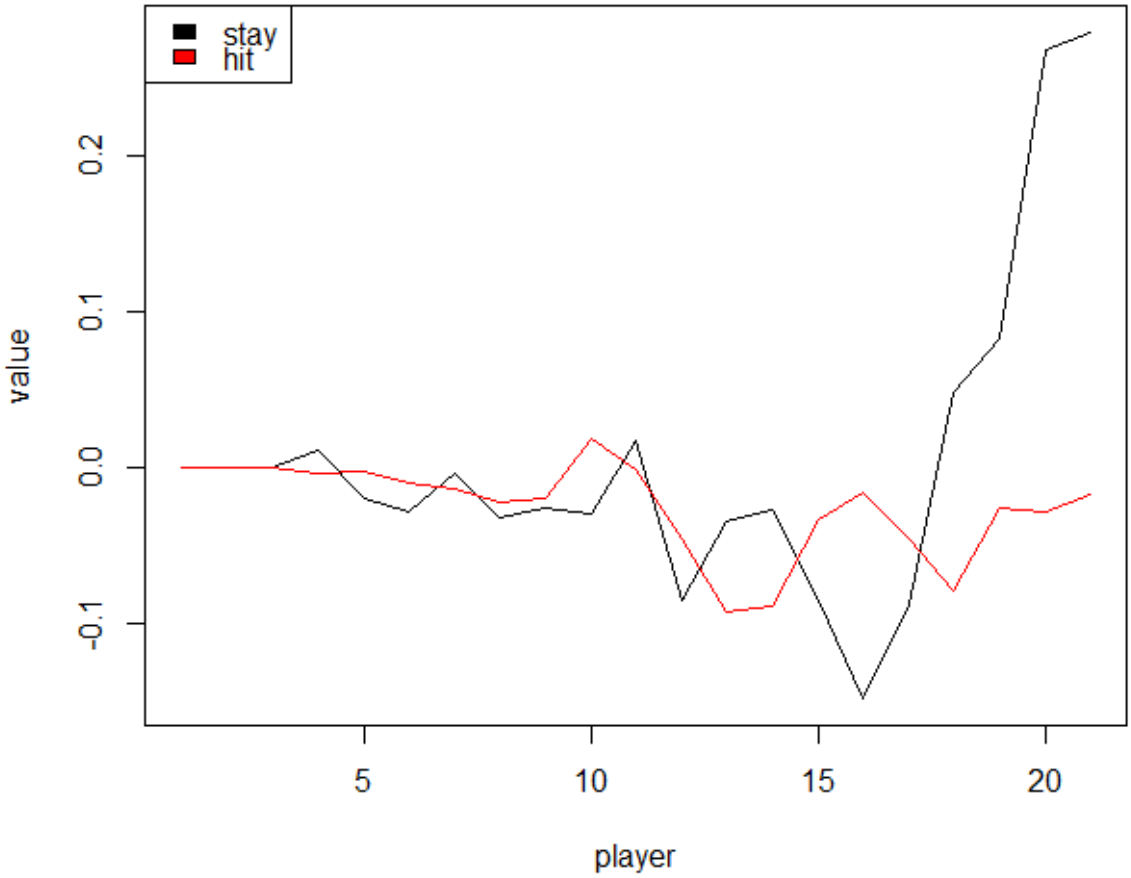
Q-Learning을 실습해보자.

```
gamma = 1  # 할인은 하지 않는다
```

```
# 가치를 초기화
qlearn.df = data.frame(
  player = rep(1:21, 10, each=4),
  dealer = rep(1:10, each=84),
  ace = rep(c(T, F), 210, each=2),
  action = rep(c(0, 1), 420),
  value = 0
  )

gamma = 1
for(i in 1:1000){
  # 행동 정책으로 한 게임의 에피소드를 수집한다
  ep <- epsilon.greedy(env, value.df)

  # 대상 정책의 가치를 업데이트 한다
  N = length(ep$episode)
  for(i in 1:N){
    # 현재 상태의 기존 가치 추정치를 찾는다
    cond1 <- action.value.cond(qlearn.df, ep$episode[[i]])
    V1 <- qlearn.df$value[cond1]

    if(i == N){  # 마지막 상태면
      R = ep$G  # 게임의 수익이 보상
      V2 = 0  # 다음 상태가 없으므로 가치도 0
    } else {  # 마지막 상태가 아니면
      R = 0  # 게임 중이므로 보상은 없다

      # 대상 정책 기준으로 다음 상태의 가치를 찾는다
      cond2 <- action.value.cond(qlearn.df, ep$episode[[i + 1]])
      V2 <- qlearn.df$value[cond2]
    }

    # 업데이트
    qlearn.df$value[cond1] <- V1 + alpha * (R + gamma * V2 - V1)
  }
}
```

학습된 가치함수를 시각화 해보자.

```
v <- qlearn.df %>% group_by(player, action) %>% summarise(mean(value))
v0 <- v %>% filter(action == 0)
v1 <- v %>% filter(action == 1)

# 그래프로 그려서 비교해본다
plot(v0$player, v0$`mean(value)`, typ='l', xlab = 'player', ylab = 'value')
lines(v1$player, v1$`mean(value)`, col=2)
legend('topleft', legend = c('stay', 'hit'), fill = c(1, 2))
```



13.3.5. 가치 함수 근사

지금까지는 상태별로 상태 가치를 추정하거나 상태-행동별로 상태-행동 가치를 추정하는 것으로 가정하고 설명했다. 그런데 상태나 상태-행동의 조합이 매우 많다면 현실적으로 모든 조합의 가치를 다 추정하는 것은 불가능하다. 위의 블랙잭만 하더라도 가능한 상태-행동의 조합이 모두 840가지나 된다. 각각의 경우의 수마다 가치를 추정하자면 매우 많은 데이터가 필요하다.

그래서 지도 학습을 통해 가치 함수 Q를 근사하는 방법을 사용하게 된다. 특히 최근에 많이 주목을 받는 방법은 Q-Learning에서 Q함수를 딥러닝으로 대체한 Deep Q-Learning이다.

가치 함수 근사는 매우 까다롭다. Q-Learning의 식을 보면 가치 함수를 추정하는데 다시 가치 함수를 사용하는데 가치 함수가 계속 바뀌기기 때문에 학습 결과가 매우 불안정해진다. 또한 실제 데이터에서는 S_t, S_{t+1}, \cdots 등이 모두 비슷비슷하게 된다. 컴퓨터 게임을 생각해보면 몇 초나 몇 분 전 후의 상태라는 것은 거의 비슷하기 마련이다. 지도 학습은 다양한 데이터가 고르게 들어와야 잘 되는데, 비슷한 데이터가 몰려서 들어오니 학습이 잘 되지 않는다.

13.3.6. 정책 경사

앞서 다룬 방법들은 모두 상태-행동의 가치를 계산하고 이를 토대로 현재 상태에서 가치가 가장 높은 행동을 하는 방식으로 문제를 풀었다. 이와 반대로 정책을 직접 개선하는 방법을 정책 경사(policy gradient)라고 한다. 이 방법은 MAB의 소프트맥스 알고리즘과 동일하다.

정책 경사는 가치를 추정하는 방법보다 상대적으로 단순하다. 왜냐하면 수익을 최대화하는 방향으로 경사상승법을 쓰는 것 뿐이기 때문이다. 이러한 특성을 이용하면 지도 학습에서 손실함수에 어떤 특성을 임의로 추가함으로써 강화학습적인 측면을 추가할 수 있다.

챗봇을 구현할 때 사용하는 방법 한 가지는 실제 질문-답변 사례를 지도학습시키고, 고객의 질문이 들어왔을 때 그 질문에 맞는 답변을 예측하고, 그 답변으로 대답하는 것이다. 이 방법으로 실제 챗봇을 만들어보면 "I don't know"처럼 '틀리지 않는 비슷비슷한' 답변을 내놓는 경향이 있다. 왜냐하면 지도학습은 예측의 오차를 줄이는 방향으로 학습이 이뤄지기 때문이다.

이러한 문제를 정책 경사를 통해 보완할 수 있다. 예를 들면 손실함수에 예측의 오차만이 아니라 이전에 했던 답변과 유사성도 추가해주면 정확하면서도 좀 더 다양한 답변을 예측하게 된다.

13.4. 다른 개념들과 차이

강화학습과 다른 개념들의 차이를 알아보자. 먼저 지도학습은 데이터가 주어진 상황에서 예측의 오차를 최소화하는 것이 목적이다. 반면 강화학습은 행동을 통해 직접 데이터를 얻어야 하고, 수익의 최대화가 목적이다.

MAB와 강화학습은 여러 면에서 비슷하다. MAB는 상태가 오로지 하나 뿐이고 행동을 하면 바로 에피소드가 끝나는 특수한 강화학습으로 볼 수도 있다. Contextual Bandit 문제는 상태가 여럿이지만 행동에 따라 상태가 달라지지는 않는다.

가장 큰 차이는 한 번의 행동으로 얻는 보상을 최대화할 것이냐(MAB), 아니면 계속해서 이어지는 행동들로 얻는 최종적인 수익을 최대화할 것이냐(강화학습)이다. A/B 테스팅 문제에 적용한다면 구매 버튼을 잘 만들어서 고객들이 구매를 많이 하게 만드는 것이 MAB 문제의 관심사라면, 고객들이 당장은 구매를 하지 않더라도 웹사이트에 계속해서 오랫동안 방문하며 장기적으로 구매를 많이 하게 만드는 것이 강화학습의 관심사라고 할 수 있다.

마지막으로 MCTS와 강화학습의 차이를 알아보자. MCTS는 가치를 여러 번의 모의실험을 통해 추정한다. 반면에 강화학습은 학습을 통해 상태-행동의 가치를 추정한다.

13.5. 어려움

강화학습은 매우 매력적인 개념이지만 아직까지 현실에서 성공사례가 많지 않다. 강화학습 자체가 지도학습에 비해 훨씬 어렵기 때문이다.

우선 강화학습은 학습 속도가 매우 느리다. 지도학습과 달리 데이터가 주어진 상황에서 학습이 이뤄지는 것이 아니라 시행착오를 거치면서 데이터를 스스로 만들어내야 하는 것이 이러한 느린 학습 속의 한 가지 원인이다. 컴퓨터 게임에서 인간이 몇 분만 플레이 하면 도달하는 수준의 실력을 갖추려면, Deep Q-Learning으로는 80여시간 분량의 게임을 플레이해야 한다.

물론 학습 속도가 느리더라도 바둑이나 컴퓨터 게임 등은 인간이 도저히 할 수 없을만큼의 분량의 게임을 해서 실력을 키울 수 있다. 그러나 로봇이나 자율주행 자동차처럼 현실과 상호작용이 필요한 경우 제대로 학습되지 않은 강화학습을 사용할 수 없기 때문에 여러 가지 어려움이 있다.

또한 강화학습은 수익을 극대화하는 방향으로 학습이 이뤄지기 때문에 우리가 원하지 않았던 의외의 형태로 작동하는 경우가 자주 있다. 예를 들면 컴퓨터 게임에서 점수를 높게 받도록 하면 게임을 잘 하는 대신 점수를 무한대로 올릴 수 있는 버그를 찾아내는 것을 예로 들 수 있다.