Reinforcement Learning Basic

Week4. Deep Q-Network

박준영

Hanyang University
Department of Computer Science

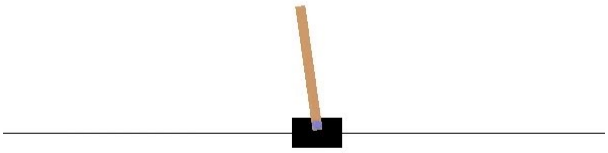
지난 시간

- 가치함수의 재귀적 정의
- 벨만 방정식(Bellman Equation)
- 시간차(TD) 학습
- 표를 이용한 강화학습

표를 이용한 기법의 한계

- 환경의 복잡도에 따라 메모리 사용량이 늘어난다.
  - 상태 가치 함수를 이용할 때: $O(|S|)$
  - 행동 가치 함수를 이용할 때: $O(|S| \times |A|)$
- 많은 환경이 상태, 행동의 개수가 매우 많거나 심지어 무한이다.
- 메모리 용량엔 한계가 존재한다.

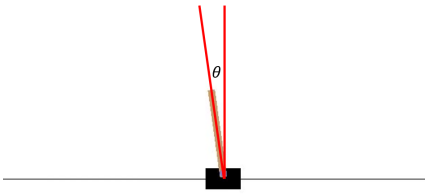
새로운 환경: CartPole-v0



새로운 환경: CartPole-v0

Observation : $[x, \theta, dx/dt, d\theta/dt]$

- x : 트랙 상에서 카트의 위치
- θ : 막대와 수직선이 이루는 각도
- dx/dt : 카트의 속도
- $d\theta/dt$: 막대의 각속도



새로운 환경: CartPole-v0

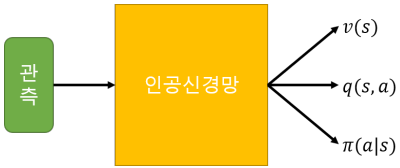
- 종료 조건
  - θ가 15° 이상
  - 원점으로 부터 거리가 2.4 units 이상
- Action : 좌우로 이동 (0 or 1)
- Reward: 에피소드가 유지된 시간

새로운 환경: CartPole-v0

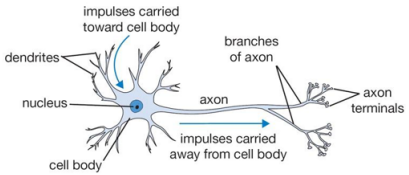
- FrozenLake와는 다르게 관측되는 값이 연속적인 값이다.
- State-Action 쌍의 개수가 무한히 많다.
- 테이블 방식으로 풀기는 어렵다.

근사적 해법

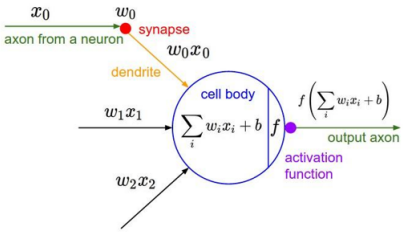
- 강화학습은 함수를 찾아나가는 과정이다.
  (ex) $v(s)$, $q(s,a)$, $\pi(a|s)$
- 딥러닝(인공 신경망)은 함수를 근사하는 도구이다.
  - 굳이 정확한 함수를 구할 필요까지는 없다.



인공신경망

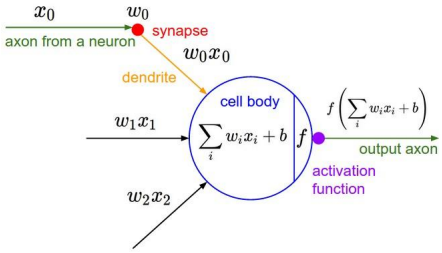


생명체의 뉴런



인공 신경망 뉴런

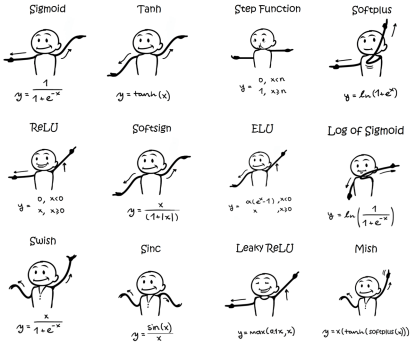
뉴런(Neuron)



인공신경망의 뉴런은 선형 함수와 활성화 함수로 이루어져있다.

$$
\begin{aligned}
z &= Wx + b \\
y &= f(z)
\end{aligned}
$$

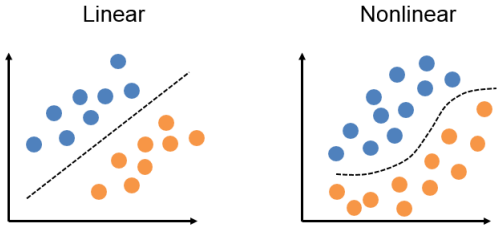
활성화 함수(Activation Function)



https://sefiks.com/2020/02/02/dance-moves-of-deep-learning-activation-functions/

활성화 함수(Activation Function)

활성화 함수들의 공통점 : 비선형 함수이다.
⟶ 비선형 함수를 활성화 함수로 사용하는 이유는?



인공신경망의 학습

- 예측: 인공신경망을 계산한 결과
- 목표: 정답 또는 갱신의 목표

인공신경망의 가중치를 예측이 목표에 가까워지도록 조절

= 학습

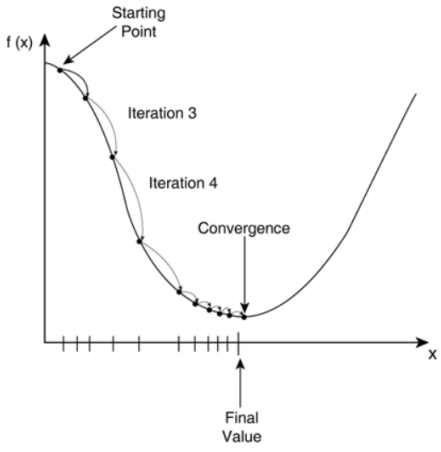
오차함수(Loss Function)

오차함수(Loss Function) : 목표와 예측 사이의 오차를 계산하는 함수.

(ex)

- MSE(Mean Squared Error) : $\mathcal{L} = (y - \hat{y})^2$
- Cross Entropy : $\mathcal{L} = -y \log \hat{y}$

경사 하강법(Gradient Descent)



Q-Learning + 인공신경망 알고리즘

1. $Q(s,a)$를 랜덤하게 초기화한다.
2. 환경과 상호작용하며 $< s, a, R, s' >$을 얻는다.
3. 오차 $\mathcal{L} = (R - \gamma \max_{a'} Q(s', a') - Q(s,a))^2$을 계산한다.
4. 확률적 경사 하강법(SGD)으로 $Q(s,a)$를 학습한다.
5. 수렴할 때까지 2번으로 돌아간다.

$\longrightarrow$ 하지만 학습이 잘 되지 않는다.

탐험(Exploration) 문제

- 탐욕적 정책을 사용한다면?
  - Q를 잘 만들었다면 에이전트는 유사한 경험만 할 것이다.
  - Q가 완벽하지 않다면 다른 행동을 시도해보지 않고 나쁜 경험만 계속 할 수 있다.
- 완전히 무작위로 탐험을 한다면?
  - 초반에는 탐욕적 정책보다 학습 속도가 빠르다.
  - 후반으로 갈수록 Q가 정확해지는데, 이용하지 않을 이유가 없다.

$\longrightarrow$ 탐험과 활용의 딜레마

SGD Optimization 문제

SGD를 사용하려면 학습 데이터가 독립 동일 분포(i.i.d.)이어야 한다.

- 에이전트의 경험은 각각이 연속되므로 독립이 아니다.
- 최적 정책이 만든 데이터 동일 분포가 아니다.

갱신의 목표 문제

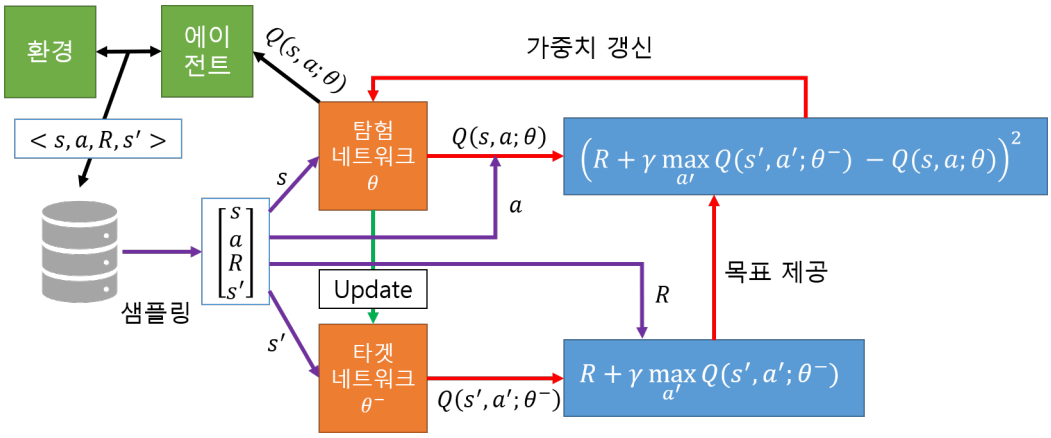
- $Q(s, a)$의 갱신에 $Q(s, a)$를 이용한다.
- 목표를 향해 이동하면 목표도 같이 바뀐다!

$\longrightarrow$ 학습이 상당히 불안정해질 수 있다.

Deep Q-Network

- 구글 딥마인드가 2015년 발표한 논문 「Human-level control through deep reinforcement learning」에 소개
- Q-Learning 알고리즘과 인공 신경망을 결합

Big Picture


환경
에이전트
$Q(s,a;\theta)$
가중치 갱신
$< s,a,R,s' >$
탐험 네트워크 θ
$Q(s,a;\theta)$
$\left(R+\gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta)\right)^2$
s
a
$\begin{bmatrix} s \\ a \\ R \\ s' \end{bmatrix}$
Update
R
목표 제공
샘플링
s'
타겟 네트워크 θ^-
$Q(s',a';\theta^-)$
$R+\gamma \max_{a'} Q(s',a';\theta^-)$

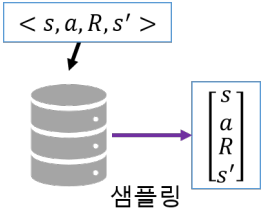
엡실론 탐욕 정책

- 어떤 기준 ϵ을 정해서 해당 확률만큼 임의로 행동을 한다.
- 그 이외에는 가치 함수에 탐욕적으로 행동한다.

- 처음에는 $\epsilon = 1$로 설정한다.
- 학습이 되어감에 따라 ϵ을 줄인다.

Replay Buffer

- 에이전트의 경험($< s, a, R, s' >$)을 저장하는 버퍼.
- 고정된 크기로, 버퍼가 가득 차면 오래된 데이터를 삭제한다.
- 학습 할 때마다 버퍼에서 랜덤하게 샘플링 해 미니배치를 구성한다.



Target Network

- 학습 데이터를 생성하는 신경망과 다른 별도의 신경망을 만든다.
  $\longrightarrow$ 타겟 인공신경망(Target Network)
- 일정 주기마다 메인 신경망의 가중치로 업데이트 한다.
- 메인 신경망의 가중치: θ, 타겟 신경망의 가중치: θ^-

$$\mathcal{L} = \left(R + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2$$

DQN 알고리즘

1. θ와 θ^-를 랜덤하게 초기화하고, 빈 replay buffer를 만든다.
2. ϵ의 확률만큼 임의로 행동하고, 그렇지 않으면 $a_t = \text{argmax}_a \, Q(s, a; \theta)$로 행동한다.
3. 얻은 경험 $< s, a, R, s' >$을 replay buffer에 저장한다.
4. replay buffer로부터 임의의 미니배치를 가져온다.
5. 오차 $\mathcal{L} = \left(R + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2$를 계산한다.
6. SGD를 이용해 θ를 갱신한다.
7. N step마다 $\theta^- \leftarrow \theta$로 갱신한다.
8. 수렴할 때까지 2번으로 돌아간다.

구현하기

```
class Network(nn.Module):
  def __init__(self):
    super(Network, self).__init__()

    self.net = nn.Sequential(
        nn.Linear(4, 64),
        nn.ReLU(inplace=True),
        nn.Linear(64, 64),
        nn.ReLU(inplace=True),
        nn.Linear(64, 2)
    )

  def forward(self, x):
    return self.net(x)
```

구현하기

```python
class ExperienceBuffer:
  def __init__(self, capacity):
    self.buffer = collections.deque(maxlen=capacity)

  def __len__(self):
    return len(self.buffer)

  def append(self, experience):
    self.buffer.append(experience)

  def sample(self, batch_size):
    indices = np.random.choice(len(self.buffer), batch_size,
                               replace=False)
    states, actions, rewards, dones, next_states = \
        zip(*[self.buffer[idx] for idx in indices])

    return np.array(states), np.array(actions), \
           np.array(rewards, dtype=np.float32), \
           np.array(dones, dtype=np.uint8), \
           np.array(next_states)
```

구현하기

```python
def play_step(self, net, epsilon=0.0, device="cpu"):
  # 중략

  if np.random.random() < epsilon:
    action = env.action_space.sample()
  else:
    state_a = np.array([self.state], copy=False)
    state_v = torch.FloatTensor(state_a).to(device)
    q_vals_v = net(state_v)
    _, act_v = torch.max(q_vals_v, dim=1)
    action = int(act_v.item())

  #중략
```

구현하기

```python
def calc_loss(batch, net, tgt_net, device="cpu"):
  states, actions, rewards, dones, next_states = batch

  # 중략

  state_action_values = net(states_v).gather(
    1, actions_v.unsqueeze(-1)).squeeze(-1)

  with torch.no_grad():
    next_state_values = tgt_net(next_states_v).max(1)[0]
    next_state_values[done_mask] = 0.0
    next_state_values = next_state_values.detach()

  expected_state_action_values = next_state_values * GAMMA + \
                                 rewards_v

  return F.mse_loss(state_action_values,
                    expected_state_action_values)
```

구현하기

```
while True:
  frame_idx += 1
  epsilon = max(EPSILON_FINAL, EPSILON_START -
                frame_idx / EPSILON_DECAY_LAST_FRAME)

  reward = agent.play_step(net, epsilon, device=device)

  #중략

  if len(buffer) < REPLAY_START_SIZE:
      continue

  if frame_idx % SYNC_TARGET_FRAMES == 0:
    tgt_net.load_state_dict(net.state_dict())

  optimizer.zero_grad()
  batch = buffer.sample(BATCH_SIZE)
  loss_t = calc_loss(batch, net, tgt_net, device=device)
  loss_t.backward()
  optimizer.step()
```