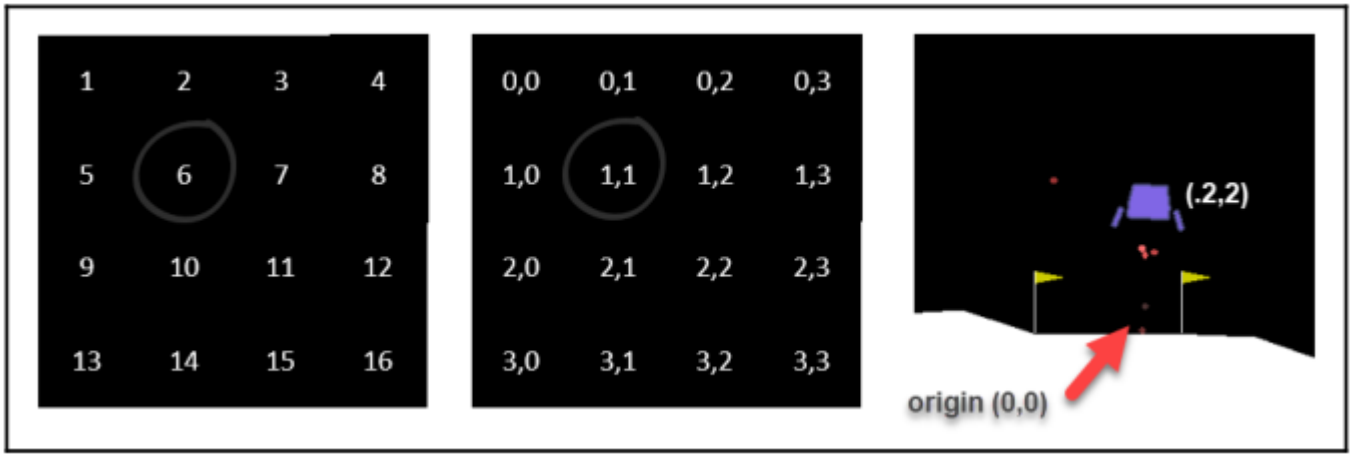
Hands-On Reinforcement Learning for Games

Ch7, Going Deeper with DDQN

최우진

2022/01/25

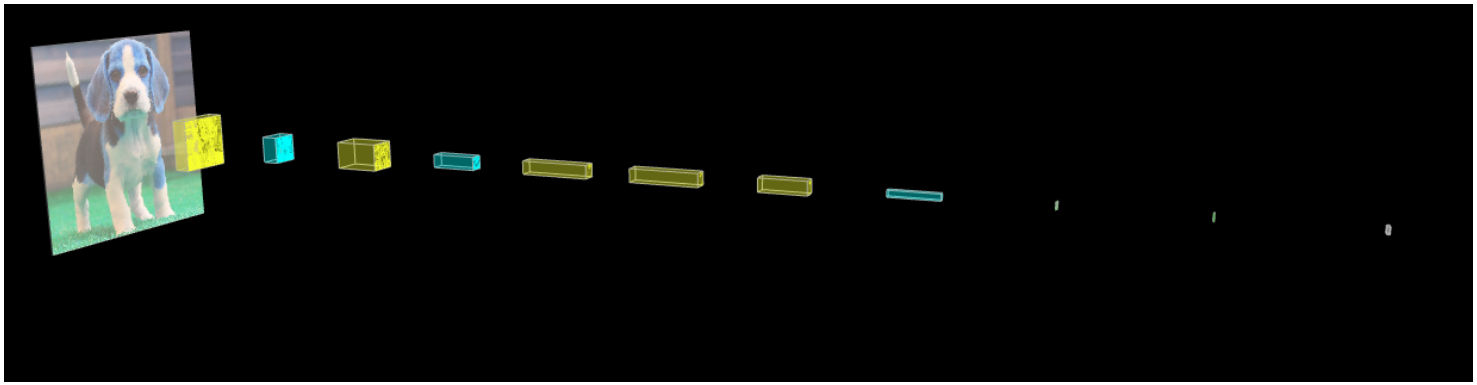
Understanding visual state



- 환경에서 상태에 대한 정보를 수집하여 특정한 값들로 표현하는 것이 인코딩이다.
- 인코딩의 방식이 위 그림처럼 여러가지가 존재할 수 있다.
- 딥러닝을 사용해서 환경에서 직접 상태에 대한 정보를 수집하도록 하는 방법도 존재한다.

Introducing CNNs

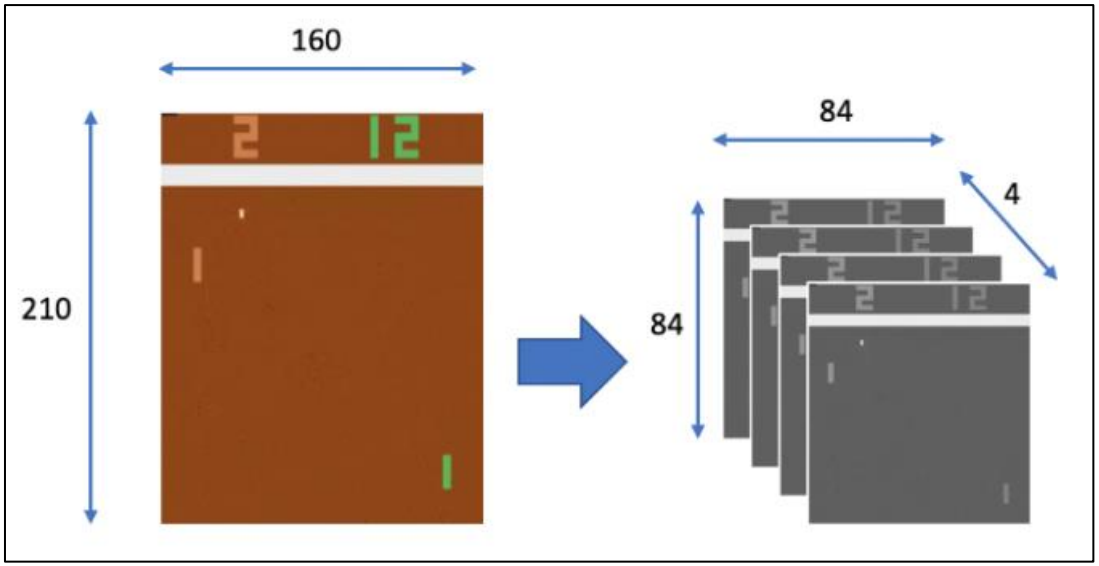
- CNN: 사진에서 특징을 추출하기 위해 사용한 딥러닝 모델 중 대표적인 모델
- AlexNet (tensorspace.org)



- Yellow: CNN layers, Blue: pooling layers

Working with a DQN on Atari

- 게임: Pong
- 게임 규칙: 두 플레이어가 게임을 하는데, 한 플레이어가 먼저 21점에 도달하면 게임 종료
- Agent: 초록색 바, 행동: 위, 아래, 정지
- 주어지는 환경 크기: [210(높이), 160(너비), 3(차원)] -> [84(높이), 84(너비), 1(차원)]



Adding CNN layers

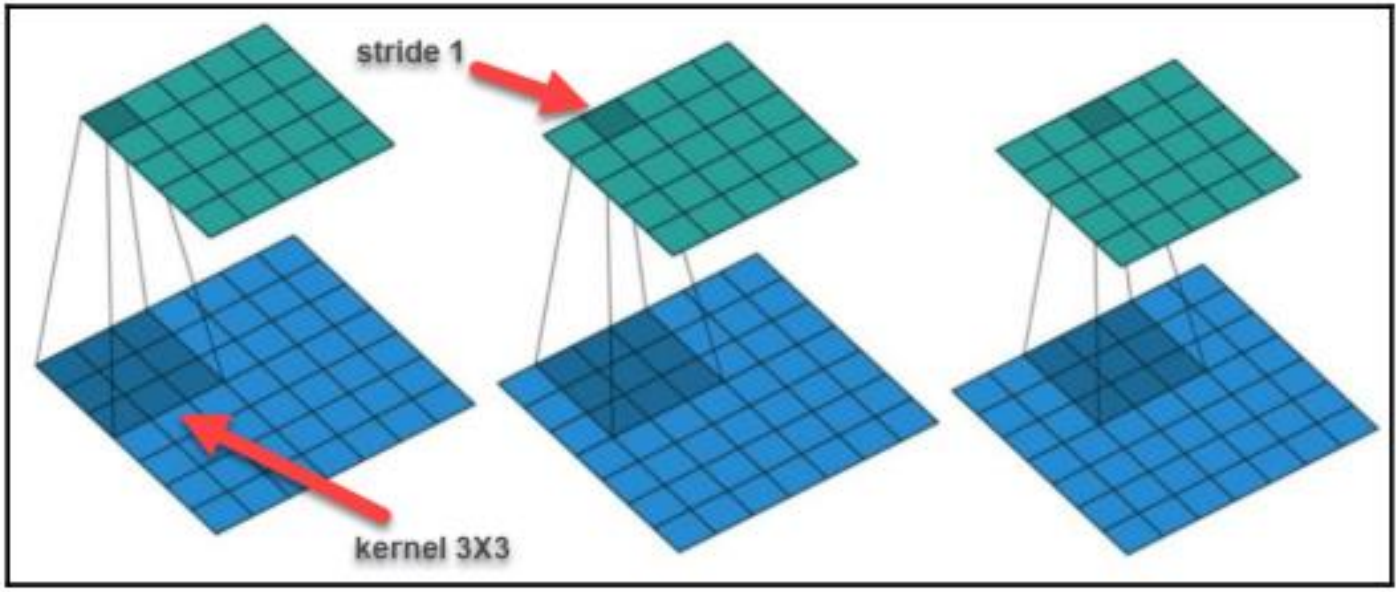
- input_shape: (1, 84, 84), self.feature_size(): 3136

```
self.features = nn.Sequential(
    nn.Conv2d(input_shape[0], 32, kernel_size=8, stride=4),  # (batch_size, 1, 84, 84)
    nn.ReLU(),
    nn.Conv2d(32, 64, kernel_size=4, stride=2),  # (batch_size, 32, 20, 20)
    nn.ReLU(),
    nn.Conv2d(64, 64, kernel_size=3, stride=1),  # (batch_size, 64, 9, 9)
    nn.ReLU()
)  # (batch_size, 64, 7, 7)

self.fc = nn.Sequential(
    nn.Linear(self.feature_size(), 512),  # (batch_size, 64 * 7 * 7 = 3136)
    nn.ReLU(),
    nn.Linear(512, self.num_actions)  # (batch_size, 512)
)  # (batch_size, num_actions(6))
```

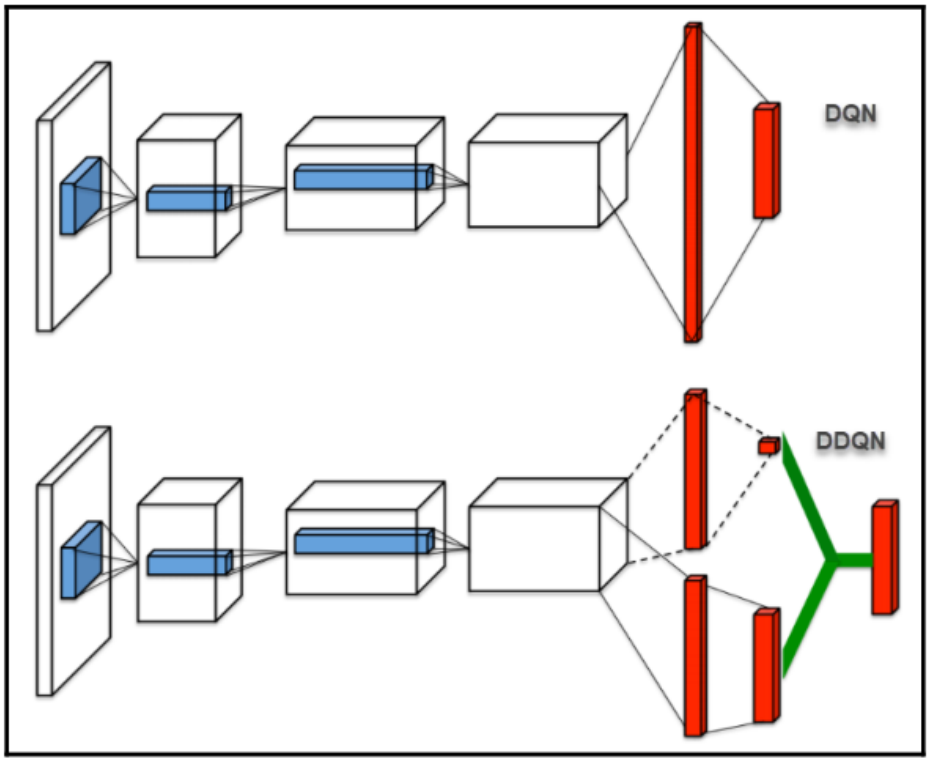
Adding CNN layers

- strided convolution process



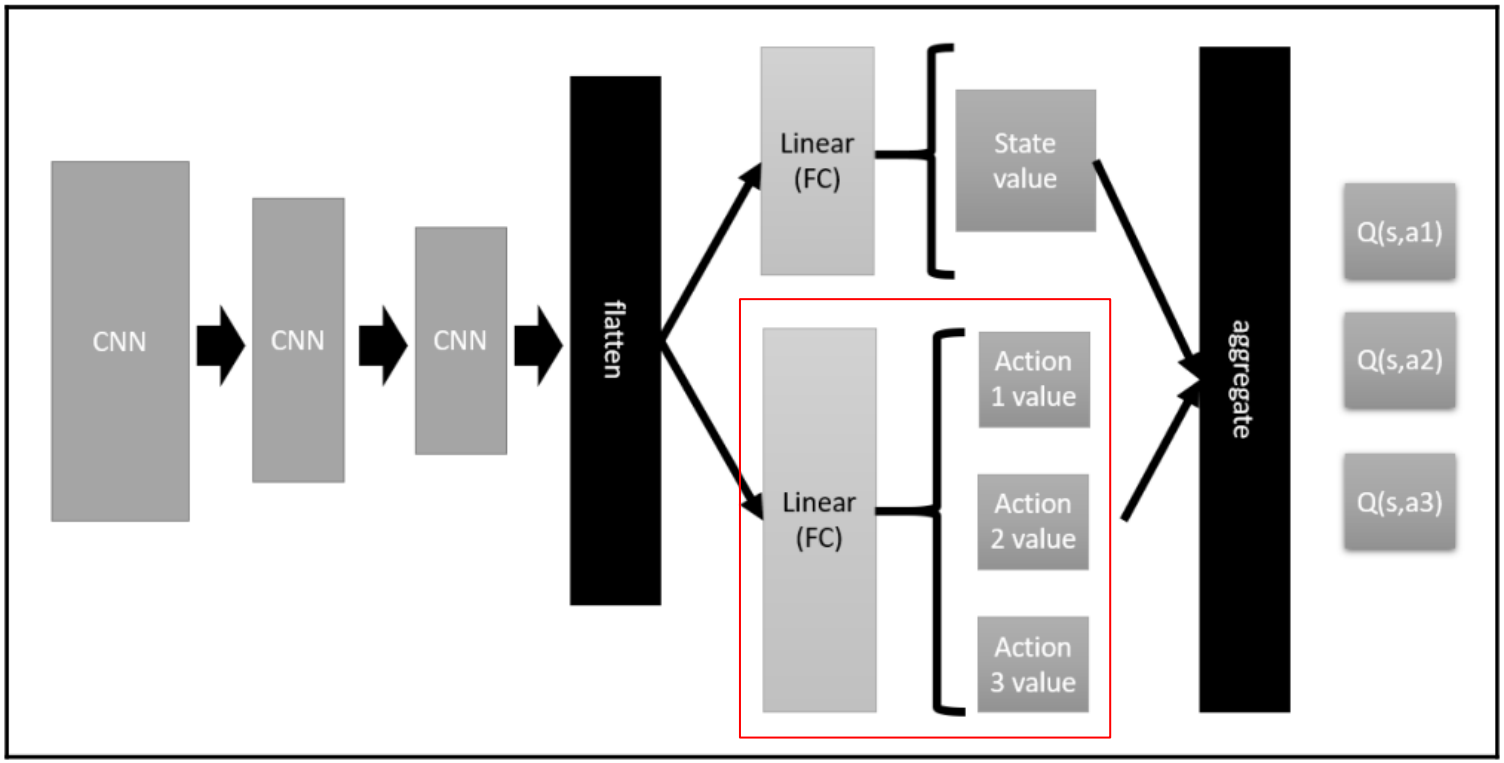
Introducing DDQN

- DDQN은 dueling DQN이며, double DQN이랑 다르다.
- Double DQN은 두 개의 다른 DQN 모델을 사용한다.



Dueling DQN or the real DDQN

- 한 행동을 선택해서 Q값을 구하는 것이 아니라 여러 행동을 통해 얻은 Q값을 모두 사용하고자 하는 모델



Advantage

Dueling DQN or the real DDQN

- Value: state s에서 받을 보상의 크기
- Advantage: 다른 action에 비해 해당 action이 얼마나 좋은 지에 대한 척도

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

- Q값에 영향을 주는 값이 무엇인지 알기 힘들다는 단점 발생
- Optimal action a^*을 선택하면 $Q(s,a) = V(s)$로 설정

$$Q(s,a) = V(s) + A(s,a) - \max_a A(s,a)$$

Dueling DQN or the real DDQN

- 게임: 달 착륙

```
class DDQN(nn.Module):
    def __init__(self, num_inputs, num_outputs):
        super(DDQN, self).__init__()

        self.feature = nn.Sequential(
            nn.Linear(num_inputs, 128),
            nn.ReLU()
        )

        self.advantage = nn.Sequential(
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, num_outputs)
        )

        self.value = nn.Sequential(
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )
```

```
def forward(self, x):
    x = self.feature(x)
    advantage = self.advantage(x) # (batch_size, 4)
    value     = self.value(x)     # (batch_size, 1)
    return value + advantage - advantage.mean()
```

Extending replay with prioritized experience replay

- Replay buffer의 데이터를 랜덤에서 우선순위를 정하는 식으로 변경
- 따라서, 중요한 경험을 자주 재사용
- 우선순위는 td_loss로 설정 -> loss가 큰 값을 더 자주 뽑도록 설정

```
def compute_td_loss(batch_size, beta):
    state, action, reward, next_state, done, indices, weights = replay_buffer.sample(batch_size
q_values      = current_model(state)
next_q_values = target_model(next_state)

q_value          = q_values.gather(1, action.unsqueeze(1)).squeeze(1)
next_q_value     = next_q_values.max(1)[0]
expected_q_value = reward + gamma * next_q_value * (1 - done)

loss = (q_value - expected_q_value.detach()).pow(2).mean()
prios = loss + 1e-5
loss  = loss.mean()

optimizer.zero_grad()
loss.backward()
replay_buffer.update_priorities(indices, prios.data.cpu().numpy())
optimizer.step()
```

Reference

- https://taek-l.tistory.com/37
- Hands-On Reinforcement Learning for Games, Ch 7.
- https://towardsdatascience.com/deep-q-network-dqn-i-bce08bdf2af

Ch7, Going Deeper with DDQN

감사합니다