Day 2. Deep Q-Network

NPEX Reinforcement Learning

July 27, 2021

Jaeuk Shin, Minkyu Park

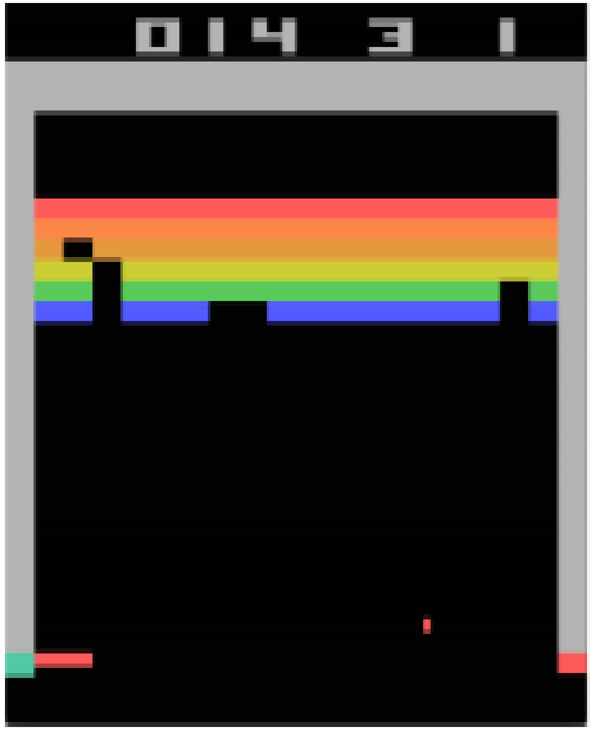
CORE
Control + Optimization Research Lab

Deep Q-Network : Why so Successful?

Is DQN merely a Q-learning algorithm with a neural network?

Two important techniques used in DQN:

1. use of **experience replay**
   - increase sample efficieny
2. use of **target Q-network**
   - mitigate training stability



DQN agent playing Atari Pong

$\Longrightarrow$ We will see how these features are implemented in practice!

Deep Q-Network : Implementation

Goal. Use a **neural network** to approximate a Q-function! (How?)

If there are m possible actions, there are two options:

1. convert a_j into **one-hot vector** $\mathbf{e}_j$, and feed $(s, \mathbf{e}_j)$ to a network which returns $Q(s, a_j; \theta)$

2. network returns m Q-values:

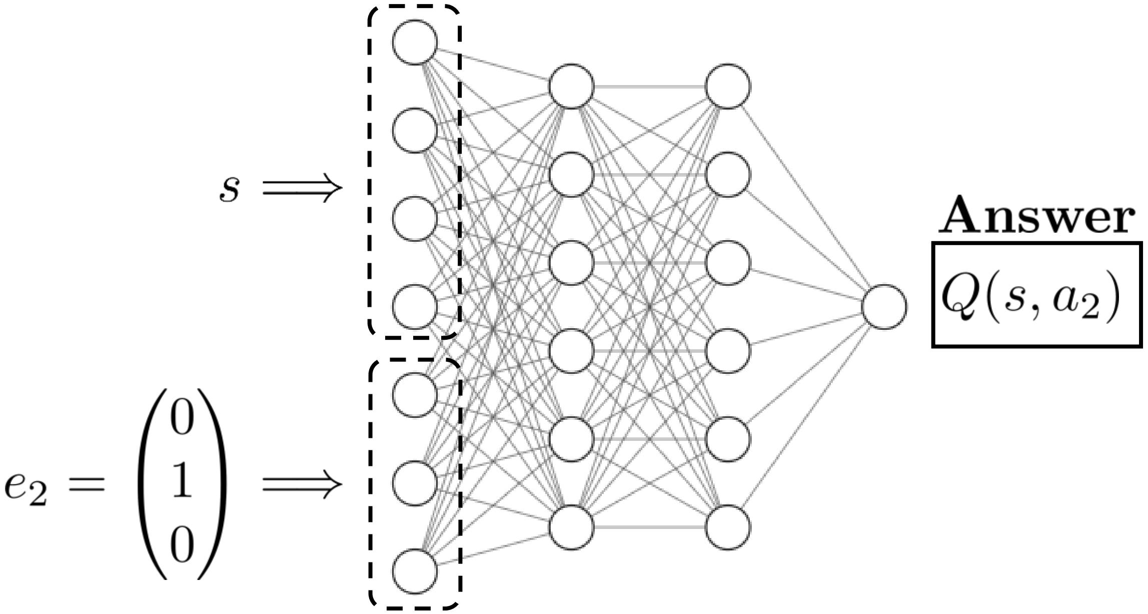
$$Q(s; \theta) = (Q(s, a_0; \theta), Q(s, a_1; \theta), \cdots Q(s, a_{m-1}; \theta))^\top.$$

Deep Q-Network : Implementation

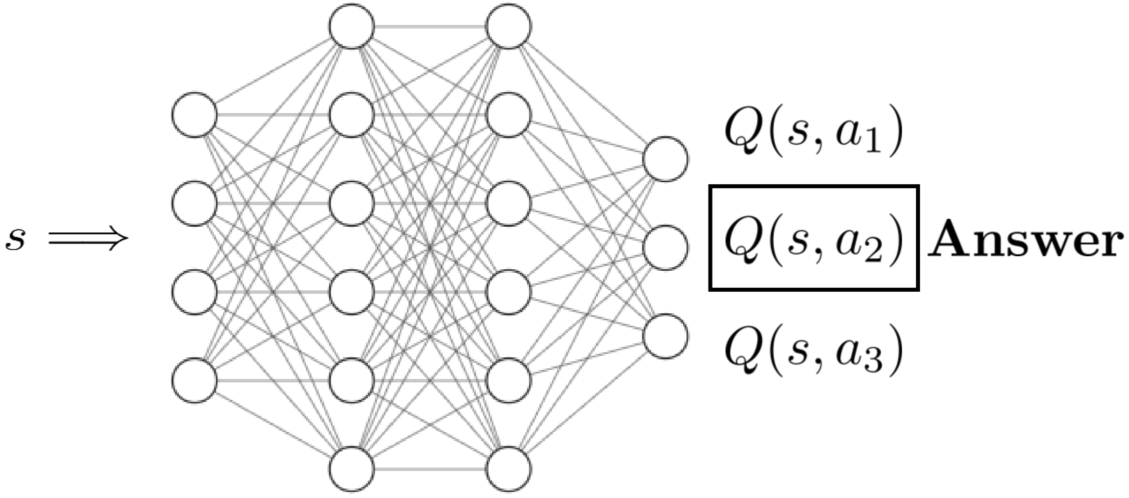
ex. 4 dim. state input & 3 actions

Query 1. Given a state s, What is $Q(s, a_2)$?

1. using one-hot vector encoding:



2. using multiple output architecture:

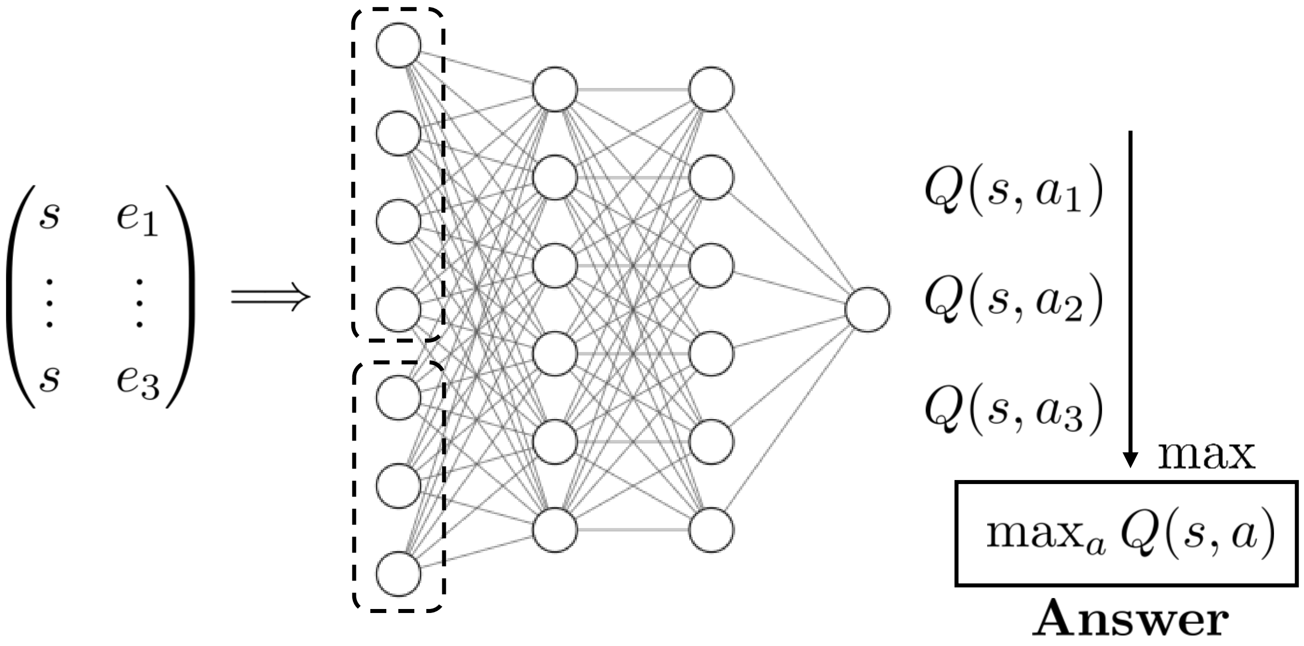


Deep Q-Network : Implementation

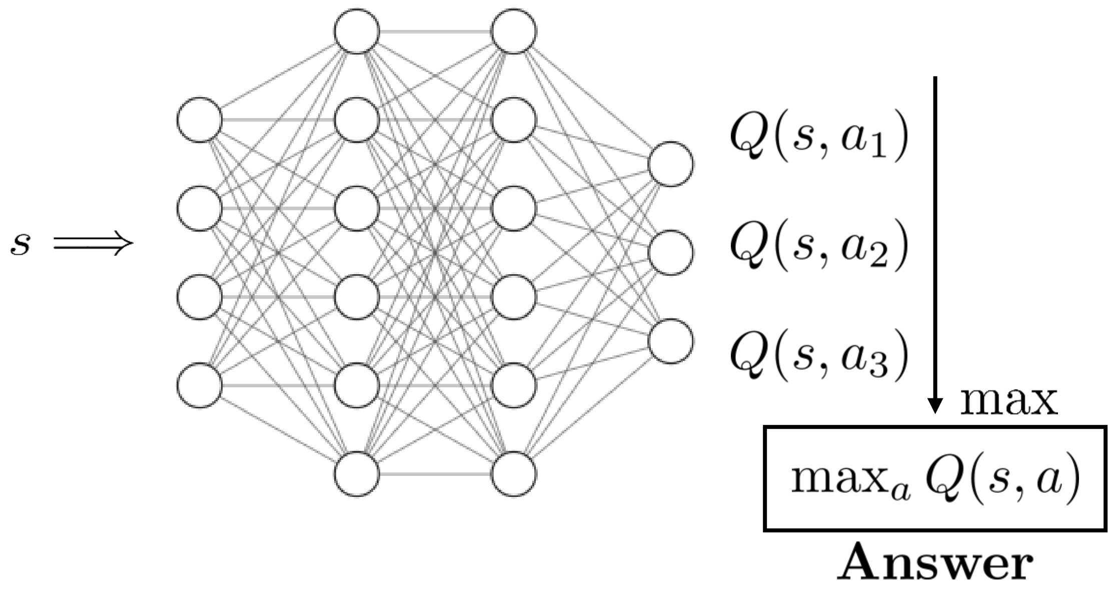
ex. 4 dim. state input & 3 actions

Query 2. Given a state s, What is $\max_a Q(s,a)$?

1. using one-hot vector encoding:



2. using multiple output architecture:



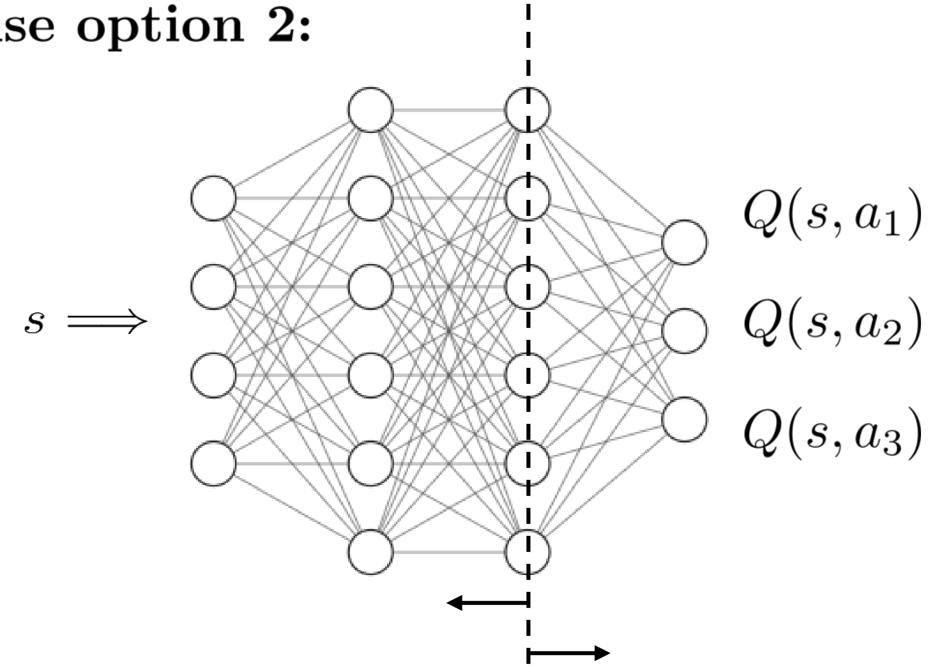
Deep Q-Network : Implementation

ex. 4 dim. state input & 3 actions

Query 3. Given a batch $s_1, \cdots, s_{256}$, What is $\max_a Q(s_1, a), \cdots, \max_a Q(s_{256}, a)$?

$\Longrightarrow$ Definitely, option 2 is better in this case.

In practice, we use option 2:



What if action is continuous?

Deep Q-Network : Implementation

2-layer neural network in PyTorch

```python
class Critic(nn.Module):
    def __init__(self, state_dim, num_action, hidden_size1, hidden_size2):
        super(Critic, self).__init__()
        self.fc1 = nn.Linear(state_dim, hidden_size1)
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.fc3 = nn.Linear(hidden_size2, num_action)

    def forward(self, state):
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))
        q = self.fc3(x)
        return q
```

neural network parameters

Given s, compute a vector of Q-values.

other non-linearities : tanh, etc.

Deep Q-Network : Implementation

```python
class DQNAgent:
    def __init__(self, obs_dim, num_act, hidden1, hidden2):
        self.obs_dim = obs_dim
        self.num_act = num_act
        self.critic = Critic(obs_dim, num_act, hidden1, hidden2).to(device)

    def act(self, state, epsilon=0.0):
        if np.random.rand() < epsilon:
            return np.random.randint(self.num_act)
        else:
            self.critic.eval()
            s = torch.Tensor(state).view(1, self.obs_dim).to(device)
            q = self.critic(s)
            return np.argmax(q.cpu().detach().numpy())
```

ε-greedy method

Deep Q-Network : Implementation

```python
def update(agent, replay_buf, gamma, critic_optim, target_critic, tau, batch_size):
    agent.critic.train()
    batch = replay_buf.sample_batch(batch_size)

    with torch.no_grad():
        observations = torch.Tensor(batch['state']).to(device)
        actions = torch.tensor(batch['action'], dtype=torch.long).to(device)
        rewards = torch.Tensor(batch['reward']).to(device)
        next_observations = torch.Tensor(batch['next_state']).to(device)
        terminals = torch.Tensor(batch['done']).to(device)
        mask = 1.0 - terminals

        next_q = torch.unsqueeze(target_critic(next_observations).max(1)[0], 1)
        target = rewards + gamma * mask * next_q

    out = agent.critic(observations).gather(1, actions)

    loss_ftn = MSELoss()
    loss = loss_ftn(out, target)
    critic_optim.zero_grad()
    loss.backward()
    critic_optim.step()

    for p, targ_p in zip(agent.critic.parameters(), target_critic.parameters()):
        targ_p.data.copy_((1. - tau) * targ_p + tau * p)
    return
```

unroll batch

training target construction

$$y_j = r_j + \gamma \max_{a'} Q\left(s'_j, a; \boxed{\theta^-}\right)$$

target network

mean squared loss:

$$\frac{1}{N}\sum_{j=1}^{N}(Q(s_j, a_j; \theta) - y_j)^2$$

gradient descent

update target network

$$\theta^- \leftarrow (1-\tau)\theta^- + \tau\theta$$

Deep Q-Network : Implementation

technical details:

$N \times d$

$\boldsymbol{s_1}$
$\boldsymbol{s_2}$
$\boldsymbol{s_3}$
$\boldsymbol{s_4}$
$\boldsymbol{s_5}$
$\boldsymbol{s_6}$

next_observations

$N \times m$

$\boldsymbol{Q(s_1, a_m)}$	...	$\boldsymbol{Q(s_1, a_m)}$
$\boldsymbol{Q(s_2, a_m)}$	...	$\boldsymbol{Q(s_2, a_m)}$
$\boldsymbol{Q(s_3, a_m)}$	...	$\boldsymbol{Q(s_3, a_m)}$
$\boldsymbol{Q(s_4, a_m)}$	...	$\boldsymbol{Q(s_4, a_m)}$
$\boldsymbol{Q(s_5, a_m)}$	...	$\boldsymbol{Q(s_5, a_m)}$
$\boldsymbol{Q(s_6, a_m)}$	...	$\boldsymbol{Q(s_6, a_m)}$

target_critic(next_observations)

N

$\max_{\boldsymbol{a}} \boldsymbol{Q(s_1, a)}$	$\max_{\boldsymbol{a}} \boldsymbol{Q(s_2, a)}$	$\max_{\boldsymbol{a}} \boldsymbol{Q(s_3, a)}$	$\max_{\boldsymbol{a}} \boldsymbol{Q(s_4, a)}$	$\max_{\boldsymbol{a}} \boldsymbol{Q(s_5, a)}$	$\max_{\boldsymbol{a}} \boldsymbol{Q(s_6, a)}$

target_critic(next_observations).max(1)[0]

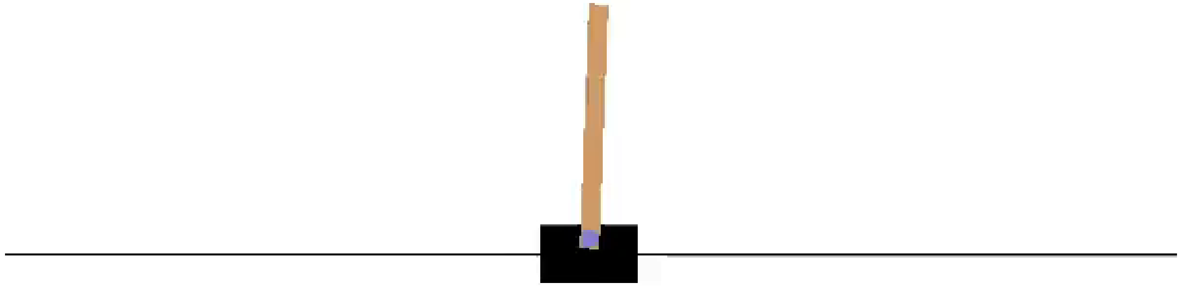
$N \times 1$

$\max_{\boldsymbol{a}} \boldsymbol{Q(s_1, a)}$
$\max_{\boldsymbol{a}} \boldsymbol{Q(s_2, a)}$
$\max_{\boldsymbol{a}} \boldsymbol{Q(s_3, a)}$
$\max_{\boldsymbol{a}} \boldsymbol{Q(s_4, a)}$
$\max_{\boldsymbol{a}} \boldsymbol{Q(s_5, a)}$
$\max_{\boldsymbol{a}} \boldsymbol{Q(s_6, a)}$

torch.unsqueeze(..., 1)

Deep Q-Network : Example

OpenAI Gym CartPole-v1



Let's try it!

Deep Q-Network : Further Improvement?

Limitation?

- stability of training
- sample efficiency
- overestimation bias
- exploration scheme

How to overcome these issues?

1. use variants of DQN, e.g., double DQN (Hasselt, '15), prioritized experience replay (Schaul, '15)
2. combine it with policy gradient method $\Longrightarrow$ **actor-critic algorithms**

CORE
Control + Optimization Research Lab