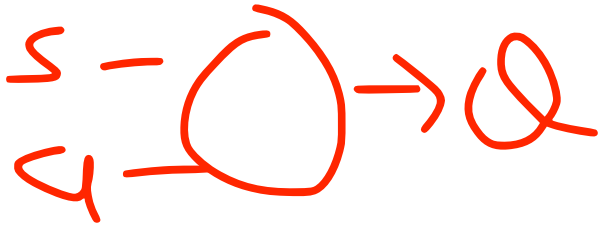
Q-Network

Prof. Tae-Hyoung Park
Dept. of Intelligent Systems & Robotics, CBNU

Q Learning

- Value function $q_*(s, a)$
  - 모든 State $s \in S$, action $a \in A$ 에 대하여 계산 필요
    - 3 x 4 grid world: 12 (states) * 4 (actions)
    - Chess board: 10^{123} (states)
  - State 가 많은 경우 Table 또는 dictionary 로 관리 어려움
  - 모든 (s, a) 에 대하여 독립적으로 평가하고 개선하는 것 어려움
  - ➢ 근사적으로 계산할 수 있는 함수 필요함

    ⇒ 신경망 (Regression) 적용 가능



- Q-Network
  - 신경망을 사용하여 Value function 을 근사적으로 계산하는 Q learning 알고리즘
  - Q learning + Neural network (regression)

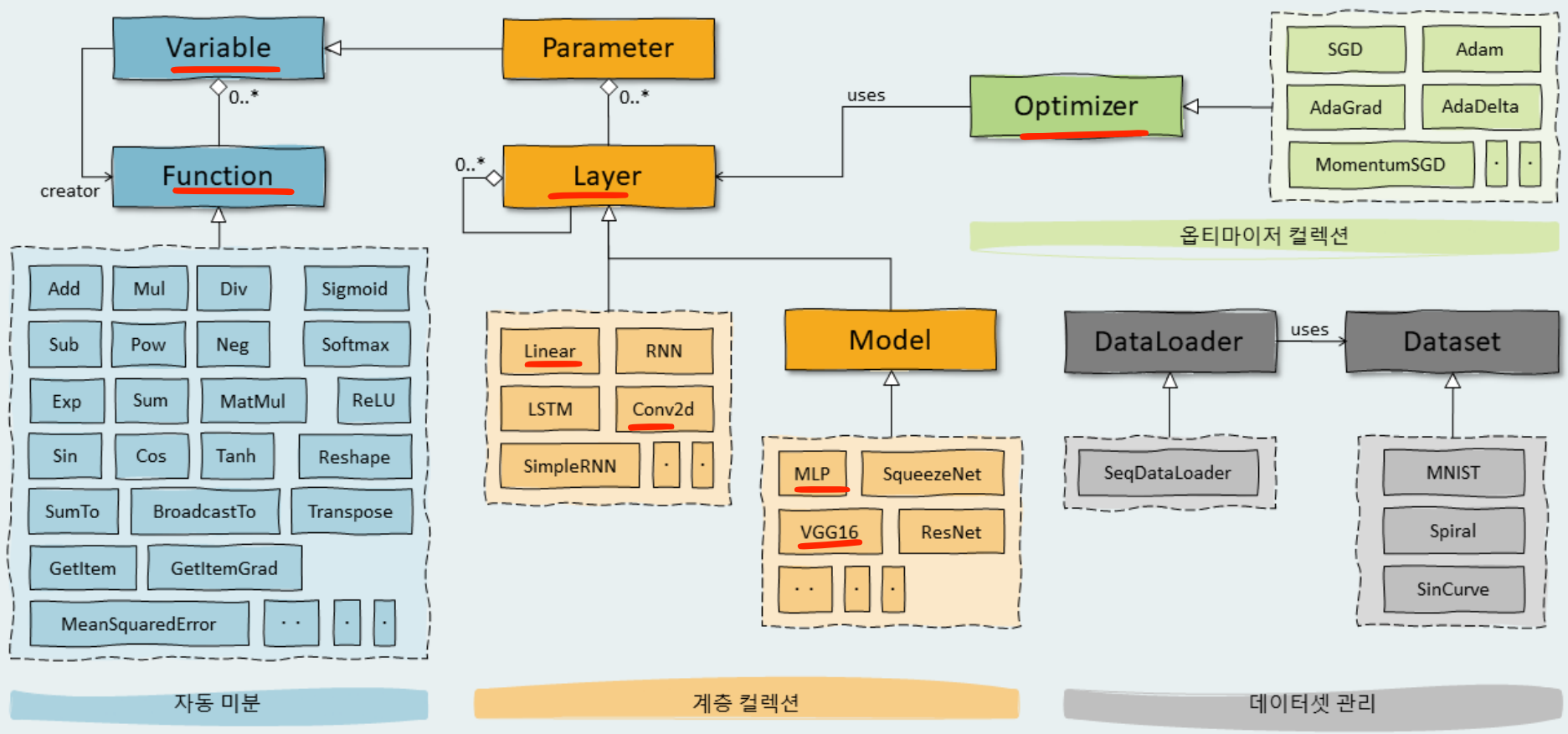
DeZero

- Deep Learning Frameworks
  - PyTorch
  - Tensorflow
  - ……
- DeZero
  - PyTorch 기반으로 알기 쉽게 설계된 신경망용 프레임 워크
  - $ pip install dezero (numpy 1.23.0 이전 버전과 호환)

```
$ pip show numpy
$ pip install numpy==1.23.0
```

DeZero

- 클래스 구조



DeZero

- Variable Class
  - 다차원 배열 (np.ndarray) 를 감싸는 클래스
  - backward() : 미분 계산 method

```
import numpy as np          # 1.23.0
from dezero import Variable  # ❶ dezero 모듈에서 Variable 임포트

x_np = np.array(5.0)
x = Variable(x_np)  # ❷ Variable 인스턴스 생성

y = 3 * x ** 2  # ❸ 넘파이 다차원 배열처럼 사용
print(y)
```

출력 결과

```
variable(75.0)
```

```
y.backward()
print(x.grad)
```

출력 결과

```
variable(30.0)
```

$$y = 3x^2$$

$$\frac{dy}{dx}_{(x=5)} = 6x_{(x=5)} = 30$$

```
class Variable:
    __array_priority__ = 200

    def __init__(self, data, name=None):
        if data is not None:
            if not isinstance(data, array_types):
                raise TypeError('{} is not supported'.format(type(data)))

        self.data = data
        self.name = name
        self.grad = None
        self.creator = None
        self.generation = 0
```

DeZero

- Function Class
  - 함수: Add(), Exp(), Mul() 등 Variable 객체에 대한 연산 수행
  - 가상함수: forward() backward()

```python
import numpy as np
from dezero import Variable
import dezero.functions as F

# 벡터의 내적
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
a, b = Variable(a), Variable(b)
c = F.matmul(a, b)
print(c)

# 행렬의 곱
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
c = F.matmul(a, b)
print(c)
```

1x4 + 2x5 + 3x6 = 32

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\begin{bmatrix}5 & 6\\7 & 8\end{bmatrix} = \begin{bmatrix}19 & 22\\43 & 50\end{bmatrix}$$

```
variable(32)
variable([[19 22]
          [43 50]])
```

DeZero

- Gradient 계산
  - Resenbrock function

$$y = 100(x_1 - x_0^2)^2 + (x_0 - 1)^2 \qquad (x_0^*, x_1^*) = (1,1)$$

  - Gradient $\left(\frac{\partial y}{\partial x_0}, \frac{\partial y}{\partial x_1}\right)$ at $x_0 = 0,\ x_1 = 2$

```
import numpy as np
from dezero import Variable

def rosenbrock(x0, x1):
    y = 100 * (x1 - x0 ** 2) ** 2 + (x0 - 1) ** 2
    return y

x0 = Variable(np.array(0.0))
x1 = Variable(np.array(2.0))

y = rosenbrock(x0, x1)
y.backward()
print(x0.grad, x1.grad)
```

$$\frac{dy}{dx_0} = -400x_0(x_1 - x_0^2) + 2(x_0 - 1)$$

$$\frac{dy}{dx_1} = 200(x_1 - x_0^2)$$

$$\left(\frac{dy}{dx_0}, \frac{dy}{dx_1}\right)_{x_0=0, x_1=2} = (-2,\ 400)$$

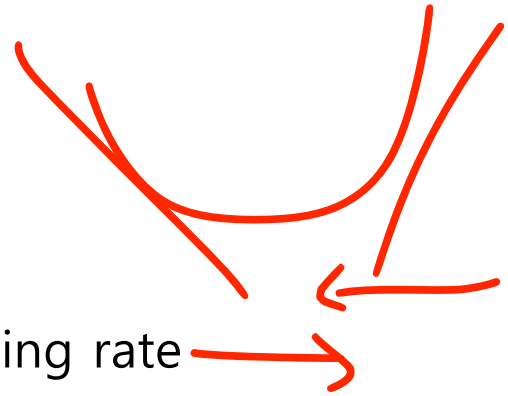
```
variable(-2.0) variable(400.0)
```

DeZero

- Gradient Descent

$$x_0^{k+1} = x_0^k - \alpha \frac{\partial y}{\partial x_0}(x_0^k, x_1^k), \qquad k = 1,2,\cdots$$

$$x_1^{k+1} = x_1^k - \alpha \frac{\partial y}{\partial x_1}(x_0^k, x_1^k), \qquad k = 1,2,\cdots$$



α: learning rate

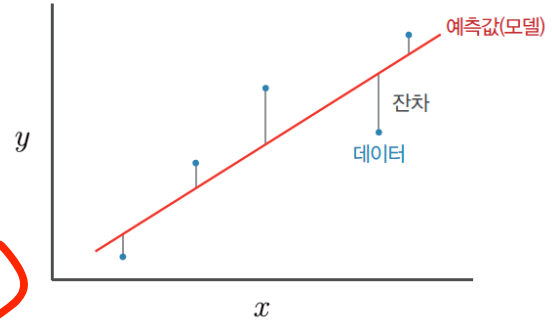
```python
import numpy as np
from dezero import Variable

def rosenbrock(x0, x1):
    y = 100 * (x1 - x0 ** 2) ** 2 + (x0 - 1) ** 2
    return y

x0 = Variable(np.array(0.0))
x1 = Variable(np.array(2.0))

iters = 10000  # 반복 횟수
lr = 0.001     # 학습률

for i in range(iters):  # 갱신 반복
    y = rosenbrock(x0, x1)

    # 이전 반복에서 더해진 미분 초기화
    x0.cleargrad()
    x1.cleargrad()

    # 미분(역전파)
    y.backward()

    # 변수 갱신
    x0.data -= lr * x0.grad.data
    x1.data -= lr * x1.grad.data

print(x0, x1)
```

$(x_0^*, x_1^*) = (1,1)$

```
variable(0.99449843677782456) variable(0.9890050527419593)
```

Linear Regression

- 응용 예 1: Linear Regression
  - Model: $y = Wx + b$
  - Data set: $(x_1, y_1), \cdots, (x_N, y_N)$
  - $(W^*, b^*) = \arg\min_{(W,b)} L$



$$L = \frac{1}{N}\sum_{i=1}^{N}(Wx_i + b - y_i)^2$$ Mean squared error / Loss

  - Gradient descent

$$W^{k+1} = W^k - \alpha \frac{\partial L}{\partial W}(x_k, y_k) \qquad b^{k+1} = b^k - \alpha \frac{\partial L}{\partial b}(x_k, y_k)$$

실습

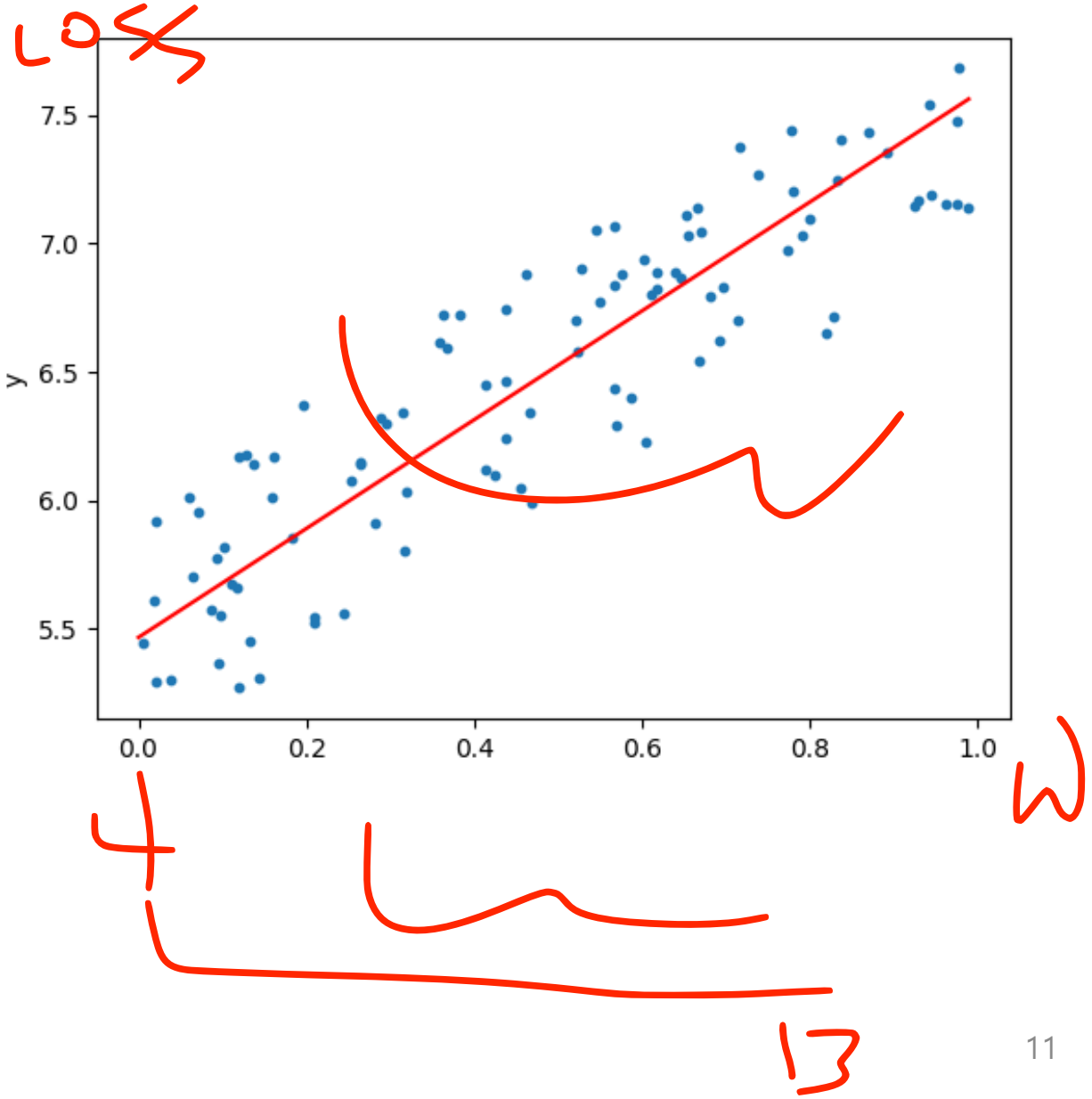
실습 #1 dezero3.py $y = 2x + 5$

```python
import numpy as np
import matplotlib.pyplot as plt
from dezero import Variable
import dezero.functions as F

# 토이 데이터셋
np.random.seed(0)
x = np.random.rand(100, 1)
y = 5 + 2 * x + np.random.rand(100, 1)
x, y = Variable(x), Variable(y)  # 생략 가능

# 매개변수 정의
W = Variable(np.zeros((1, 1)))
b = Variable(np.zeros(1))

# 예측 함수
def predict(x):
    y = F.matmul(x, W) + b  # 행렬의 곱으로 여
    return y

# 평균 제곱 오차 계산 함수
def mean_squared_error(x0, x1):
    diff = x0 - x1
    return F.sum(diff ** 2) / len(diff)

# 경사 하강법으로 매개변수 갱신
lr = 0.1
iters = 100
```

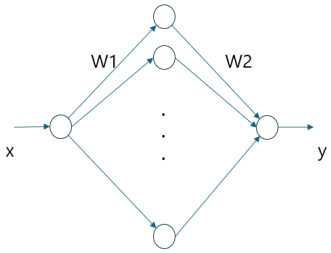
```python
for i in range(iters):
    y_pred = predict(x)
    loss = mean_squared_error(y, y_pred)
    # 또는 loss = F.mean_squared_error(y, y_pred)

    W.cleargrad()
    b.cleargrad()
    loss.backward()

    W.data -= lr * W.grad.data
    b.data -= lr * b.grad.data

    if i % 10 == 0:  # 10회 반복마다 출력
        print(loss.data)

print('====')
print('W =', W.data)
print('b =', b.data)

# [그림 7-9] 학습 후 모델
plt.scatter(x.data, y.data, s=10)
plt.xlabel('x')
plt.ylabel('y')
t = np.arange(0, 1, .01)[:, np.newaxis]
y_pred = predict(t)
plt.plot(t, y_pred.data, color='r')
plt.show()
```

실습

```
42.296340129442335
0.249157319775611 34
0.100789749543016 52
0.094618598030406 94
0.090266713813731 1
0.086945854839646 15
0.084410842064932 75
0.082475710222291 21
0.080998504540410 51
0.079870862186250 04
====
W = [[2.11807369]]
b = [5.46608905]
```



Nonlinear Regression



- 응용 예 2: Nonlinear Regression
  - Model: MLP $y = \sin 2\pi x$

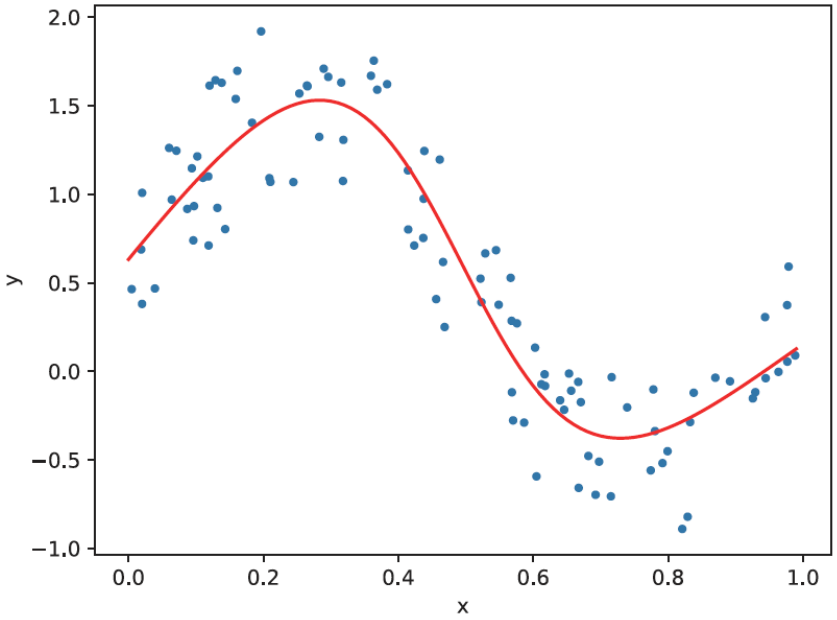
```
# 데이터셋
np.random.seed(0)
x = np.random.rand(100, 1)
y = np.sin(2 * np.pi * x) + np.random.rand(100, 1)

# ❶ 매개변수 초기화
I, H, O = 1, 10, 1  # I=입력층 차원 수, H=은닉층 차원 수, O=출력층 차원 수
W1 = Variable(0.01 * np.random.randn(I, H))  # 첫 번째 층의 가중치
b1 = Variable(np.zeros(H))                   # 첫 번째 층의 편향
W2 = Variable(0.01 * np.random.randn(H, O))  # 두 번째 층의 가중치
b2 = Variable(np.zeros(O))                   # 두 번째 층의 편향

# ❷ 신경망 추론
def predict(x):
    y = F.linear(x, W1, b1)                  # affine transformation
    y = F.sigmoid(y)
    y = F.linear(y, W2, b2)
    return y
```

```
lr = 0.2
iters = 10000

# ❸ 신경망 학습(매개변수 갱신)
for i in range(iters):
    y_pred = predict(x)
    loss = F.mean_squared_error(y, y_pred)

    W1.cleargrad()
    b1.cleargrad()
    W2.cleargrad()
    b2.cleargrad()

    loss.backward()

    W1.data -= lr * W1.grad.data
    b1.data -= lr * b1.grad.data
    W2.data -= lr * W2.grad.data
    b2.data -= lr * b2.grad.data

    if i % 1000 == 0:  # 1000회마다 출력
        print(loss.data)
```

Nonlinear Regression

- Nonlinear Regression

```
0.8165178492839196
0.24990280802148895
...
0.07618764131185574
```



Nonlinear Regression

- Layer Class & Model Class 사용

```python
import numpy as np
from dezero import Model
import dezero.layers as L
import dezero.functions as F

# 데이터셋 생성
np.random.seed(0)
x = np.random.rand(100, 1)
y = np.sin(2 * np.pi * x) + np.random.rand(100, 1)

lr = 0.2
iters = 10000

class TwoLayerNet(Model):  # 2층 신경망
    def __init__(self, hidden_size, out_size):
        super().__init__()
        self.l1 = L.Linear(hidden_size)
        self.l2 = L.Linear(out_size)

    def forward(self, x):
        y = F.sigmoid(self.l1(x))
        y = self.l2(y)
        return y
```

```python
model = TwoLayerNet(10, 1)  # 신경망 모델 생성

for i in range(iters):
    y_pred = model.forward(x)  # 또는 y_pred = model(x)
    loss = F.mean_squared_error(y, y_pred)

    model.cleargrads()
    loss.backward()

    for p in model.params():
        p.data -= lr * p.grad.data

    if i % 1000 == 0:
        print(loss)
```

실습

실습 #2 dezero4.py

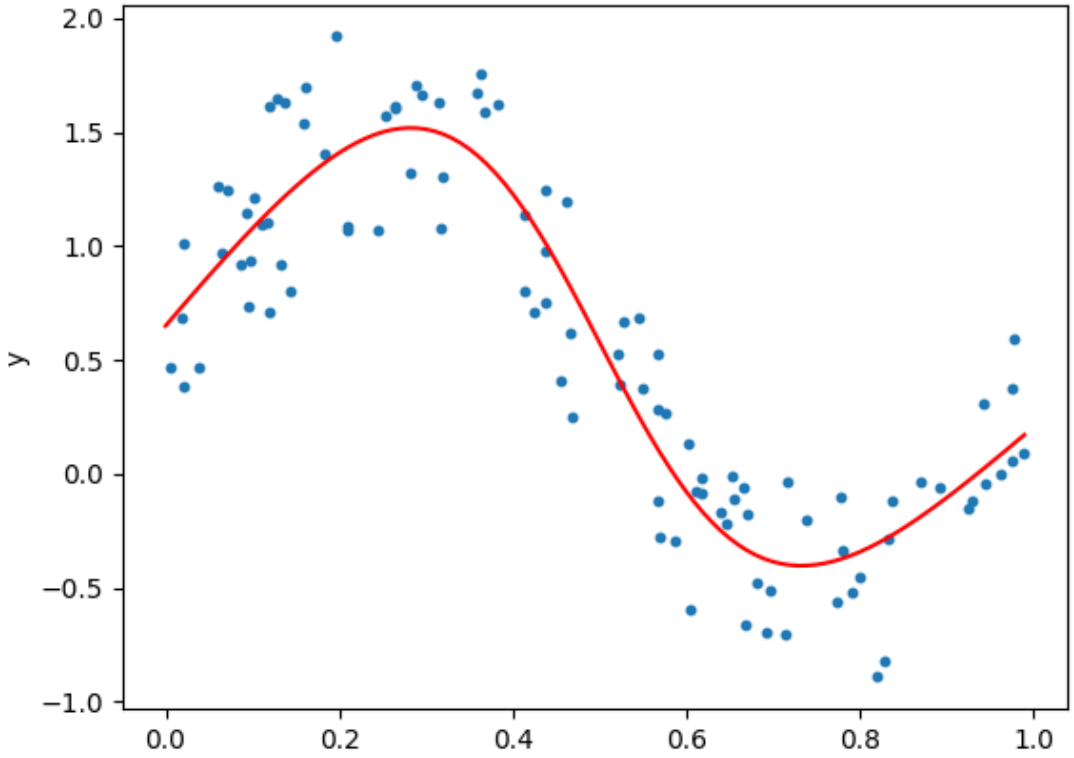
```python
import numpy as np
import matplotlib.pyplot as plt
from dezero import Model
from dezero import optimizers  # 옵티마이저들이 들어
import dezero.layers as L
import dezero.functions as F

# 데이터셋 생성
np.random.seed(0)
x = np.random.rand(100, 1)
y = np.sin(2 * np.pi * x) + np.random.rand(100, 1)

lr = 0.2
iters = 10000

class TwoLayerNet(Model):
    def __init__(self, hidden_size, out_size):
        super().__init__()
        self.l1 = L.Linear(hidden_size)
        self.l2 = L.Linear(out_size)

    def forward(self, x):
        y = F.sigmoid(self.l1(x))
        y = self.l2(y)
        return y
```

```python
model = TwoLayerNet(10, 1)
optimizer = optimizers.SGD(lr)  # 옵티마이저 생성
optimizer.setup(model)          # 최적화할 모델을
                                # 옵티마이저에 등록

for i in range(iters):
    y_pred = model(x)
    loss = F.mean_squared_error(y, y_pred)

    model.cleargrads()
    loss.backward()

    optimizer.update()  # 옵티마이저로 매개변수 갱신
    if i % 1000 == 0:
        print(loss.data)

# 그래프로 시각화([그림 7-12]와 같음)
plt.scatter(x, y, s=10)
plt.xlabel('x')
plt.ylabel('y')
t = np.arange(0, 1, .01)[:, np.newaxis]
y_pred = model(t)
plt.plot(t, y_pred.data, color='r')
plt.show()
```

실습

- 실행결과

```
0.8165178492839196
0.24990280802148895
0.24609876581126014
0.2372159081431807
0.20793216413350174
0.12311905720649353
0.07888166506355147
0.07655073683421636
0.07637803086238222
0.0761876413118557
```



Quiz

<Q1> 위 실습에서 다음의 optimizer 를 실행하여 결과를 비교하라.

```
class MomentumSGD(Optimizer):
class AdaGrad(Optimizer):
class Adam(Optimizer):
```

<Q2> $y = \sin(4\pi x)\ (0 \le x \le 1)$ 에 대한 loss 가 최소화 되도록 신경망을 최적화하고 결과를 출력하라.

하이퍼 파라미터 튜닝 필요.

Q-Network

- One-Hot Vector
  - 여러 원소 중 하나만 '1' 이고 다른 원소는 모두 '0' 인 벡터
  - 범주형 데이터 처리에 사용

    (ex) 옷 사이즈 S, M, L => (1,0,0), (0,1,0), (0,0,1) : 1 x 3 vector

    (ex) 3 x 4 grid world 에서 agent 의 상태 (y, x) : (0,0) ~ (2,3)

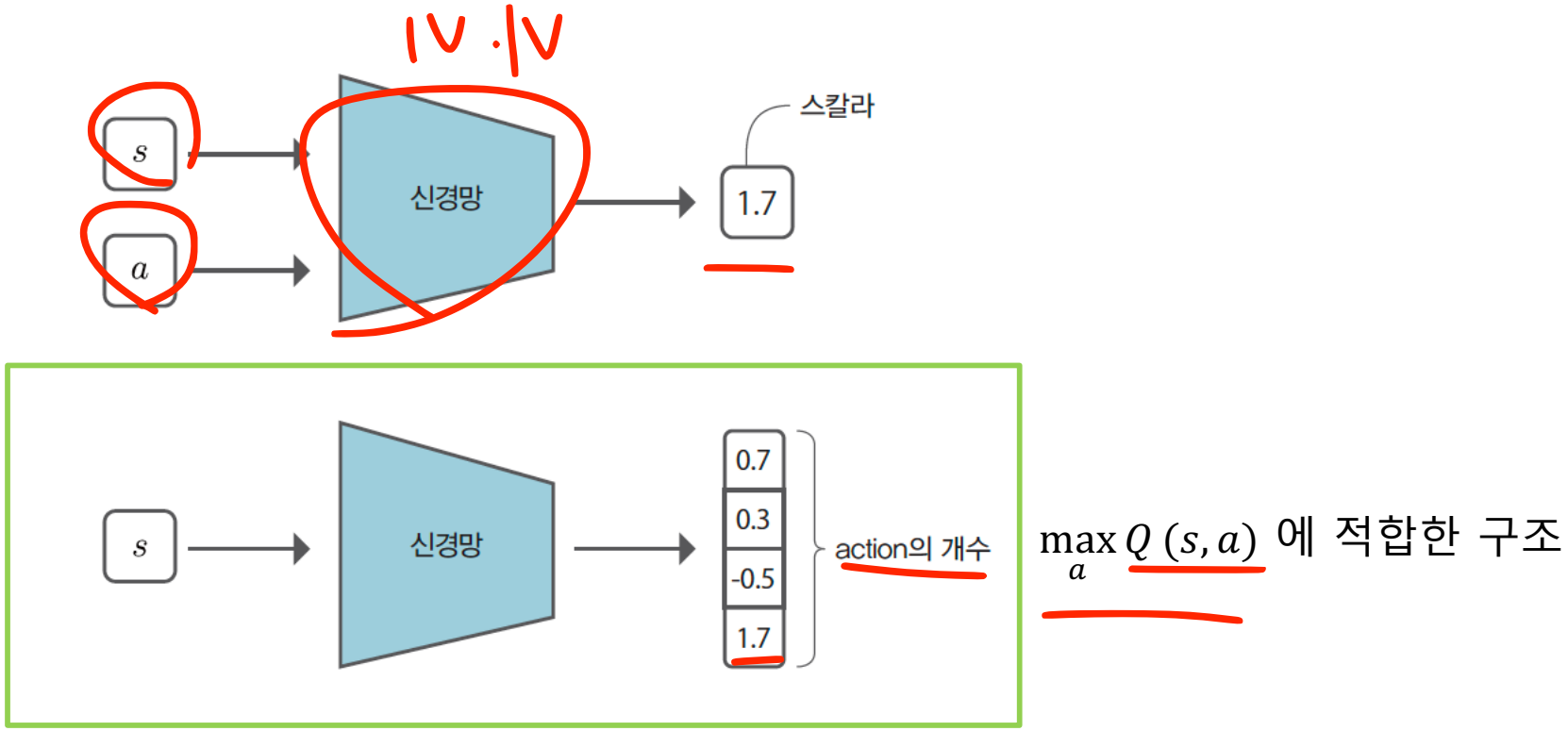
    => (1,0,0,0,0,0,0,0,0,0,0,0) ..... (0,0,0,0,0,0,0,0,0,0,0,1) : 1 x 12 vector

```python
def one_hot(state):
    # ❶ 벡터 준비
    HEIGHT, WIDTH = 3, 4
    vec = np.zeros(HEIGHT * WIDTH, dtype=np.float32)

    # ❷ state에 해당하는 원소만 1.0으로 설정
    y, x = state
    idx = WIDTH * y + x
    vec[idx] = 1.0

    # ❸ 배치 처리를 위해 새로운 축 추가
    return vec[np.newaxis, :]
```

```python
state = (2, 0)
x = one_hot(state)

print(x.shape)  # [출력 결과] (1, 12)
print(x)        # [출력 결과] [[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
```

Q-Network

- Q function: $Q(s, a)$
  - 테이블 (defaultdict) 로 구현
    - state/action 이 간단한 문제에만 적용가능
  - $Q(s, a)$ 의 계산 ⇒ Nonlinear regression 문제 ⇒ 신경망 구현

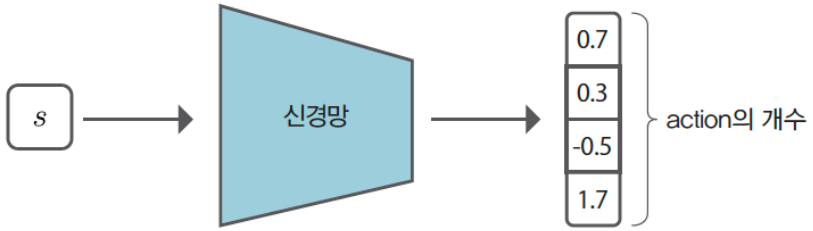


$\max_a Q(s, a)$ 에 적합한 구조

Q-Network

- QNet

```python
from dezero import Model
import dezero.functions as F
import dezero.layers as L

class QNet(Model):
    def __init__(self):
        super().__init__()
        self.l1 = L.Linear(100)  # 중간층의 크기
        self.l2 = L.Linear(4)    # 행동의 크기(가능한 행동의 개수)

    def forward(self, x):
        x = F.relu(self.l1(x))
        x = self.l2(x)
        return x

qnet = QNet()

state = (2, 0)
state = one_hot(state)  # 원-핫 벡터로 변환

qs = qnet(state)
print(qs.shape)  # [출력 결과] (1, 4)
```



Q-Network

- Q Function 의 학습
  - Update equation

$$Q'(S_t, A_t) = Q(S_t, A_t) + \alpha\{R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\}$$

$$\Rightarrow \quad Q'(S_t, A_t) = Q(S_t, A_t) + \alpha\{T - Q(S_t, A_t)\}$$

  - 입력 (S_t, A_t) 일때 출력 T 가 되도록 QNet 학습 : T = 정답, label

```
class QLearningAgent:
    ...

    def update(self, state, action, reward, next_state, done):
        # ❶ 다음 상태에서 최대가 되는 Q 함수의 값(next_q) 계산
        if done:  # ❷ 목표 상태에 도달
            next_q = np.zeros(1)  # ❸ [0.] (목표 상태에서의 Q 함수
        else:     # 그 외 상태
            next_qs = self.qnet(next_state)
            next_q = next_qs.max(axis=1)
```

S_t A_t R_t S_{t+1} : $Q(S_{t+1}, a)$: $\max_a Q(S_{t+1}, a)$

```
# ❺ 목표
target = self.gamma * next_q + reward
# ❻ 현재 상태에서의 Q 함수 값(q) 계산
qs = self.qnet(state)
q = qs[:, action]
# ❼ 목표(target)와 q의 오차 계산
loss = F.mean_squared_error(target, q)

# ❽ 역전파 → 매개변수 갱신
self.qnet.cleargrads()
loss.backward()
self.optimizer.update()

return loss.data
```

$\gamma \max_a Q(S_{t+1}, a) + R_t$: $Q(S_t, a)$: $Q(S_t, A_t)$

Q-Network

- Q Learning vs Q-Network

Q Learning

```
self.Q = defaultdict(lambda: 0)
```

Table

```
def update(self, state, action, reward, next_state, done):
    if done:  # 목표에 도달
        next_q_max = 0
    else:     # 그 외에는 다음 상태에서 Q 함수의 최댓값 계산
        next_qs = [self.Q[next_state, a] for a in range(self.action_size)]
        next_q_max = max(next_qs)

    # Q 함수 갱신
    target = reward + self.gamma * next_q_max
    self.Q[state, action] += (target - self.Q[state, action]) * self.alpha
```

Q-Network

```
class QNet(Model):
```

Netwok

```
def update(self, state, action, reward, next_state, done):
    # 다음 상태에서 최대가 되는 Q 함수의 값(next_q) 계산
    if done:  # 목표 상태에 도달
        next_q = np.zeros(1)  # [0.]  # [0.] (목표 상태에서
    else:     # 그 외 상태
        next_qs = self.qnet(next_state)
        next_q = next_qs.max(axis=1)
        next_q.unchain()  # next_q를 역전파 대상에서 제외

    # 목표
    target = self.gamma * next_q + reward
    # 현재 상태에서의 Q 함수 값(q) 계산
    qs = self.qnet(state)
    q = qs[:, action]
    # 목표(target)와 q의 오차 계산
    loss = F.mean_squared_error(target, q)

    # 역전파 → 매개변수 갱신
    self.qnet.cleargrads()
    loss.backward()
    self.optimizer.update()

    return loss.data
```

실습

실습 #3 q_learning_nn.py

```python
import matplotlib.pyplot as plt
import numpy as np
from dezero import Model
from dezero import optimizers
import dezero.functions as F
import dezero.layers as L
from common.gridworld import GridWorld

def one_hot(state):
    HEIGHT, WIDTH = 3, 4
    vec = np.zeros(HEIGHT * WIDTH, dtype=np.float32)
    y, x = state
    idx = WIDTH * y + x
    vec[idx] = 1.0
    return vec[np.newaxis, :]

class QNet(Model):
    def __init__(self):
        super().__init__()
        self.l1 = L.Linear(100)  # 중간층의 크기
        self.l2 = L.Linear(4)    # 행동의 크기(가능한 행동의 개수)

    def forward(self, x):
        x = F.relu(self.l1(x))
        x = self.l2(x)
        return x
```

```python
class QLearningAgent:
    def __init__(self):
        self.gamma = 0.9
        self.lr = 0.01
        self.epsilon = 0.1
        self.action_size = 4

        self.qnet = QNet()                          # 신경망 초기화
        self.optimizer = optimizers.SGD(self.lr)  # 옵티마이저 생성
        self.optimizer.setup(self.qnet)           # 옵티마이저에 신경망 등록

    def get_action(self, state_vec):
        if np.random.rand() < self.epsilon:
            return np.random.choice(self.action_size)
        else:
            qs = self.qnet(state_vec)
            return qs.data.argmax()

    def update(self, state, action, reward, next_state, done):
        # 다음 상태에서 최대가 되는 Q 함수의 값(next_q) 계산
        if done:  # 목표 상태에 도달
            next_q = np.zeros(1)  # [0.]  # [0.] (목표 상태에서의 Q 함수는 항
        else:     # 그 외 상태
            next_qs = self.qnet(next_state)
            next_q = next_qs.max(axis=1)
            next_q.unchain()  # next_q를 역전파 대상에서 제외

        # 목표
        target = self.gamma * next_q + reward
        # 현재 상태에서의 Q 함수 값(q) 계산
        qs = self.qnet(state)
        q = qs[:, action]
        # 목표(target)와 q의 오차 계산
        loss = F.mean_squared_error(target, q)

        # 역전파 → 매개변수 갱신
        self.qnet.cleargrads()
        loss.backward()
        self.optimizer.update()

        return loss.data
```

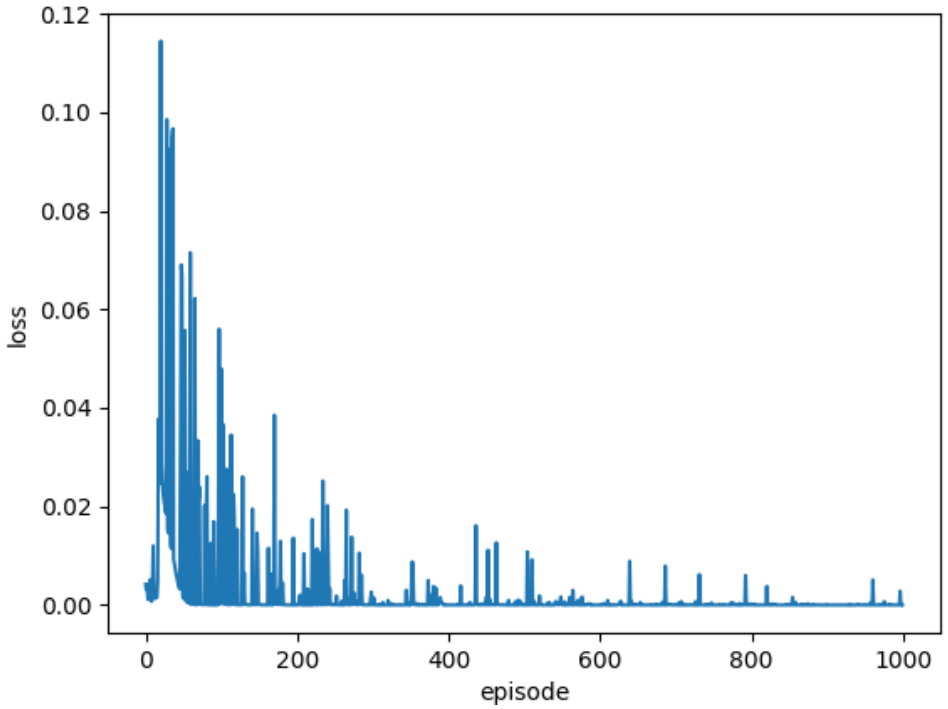
실습

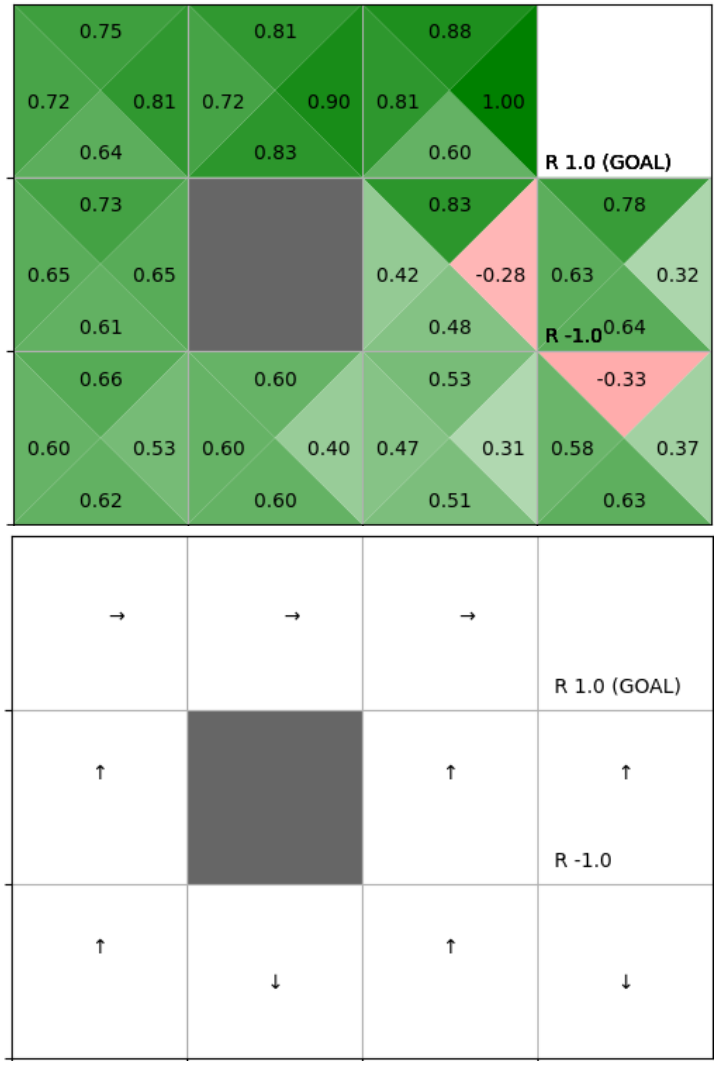
```
env = GridWorld()
agent = QLearningAgent()

episodes = 1000  # 에피소드 수
loss_history = []

for episode in range(episodes):
    state = env.reset()
    state = one_hot(state)
    total_loss, cnt = 0, 0
    done = False

    while not done:
        action = agent.get_action(state)
        next_state, reward, done = env.step(action)
        next_state = one_hot(next_state)

        loss = agent.update(state, action, reward, next_state, done)
        total_loss += loss
        cnt += 1
        state = next_state

    average_loss = total_loss / cnt
    loss_history.append(average_loss)
```

```
# 에피소드별 손실 추이
plt.xlabel('episode')
plt.ylabel('loss')
plt.plot(range(len(loss_history)), loss_history)
plt.show()

#  신경망을 이용한 Q 러닝으로 얻은 Q 함수와 정책
Q = {}
for state in env.states():
    for action in env.action_space:
        q = agent.qnet(one_hot(state))[:, action]
        Q[state, action] = float(q.data)
env.render_q(Q)
```

실습

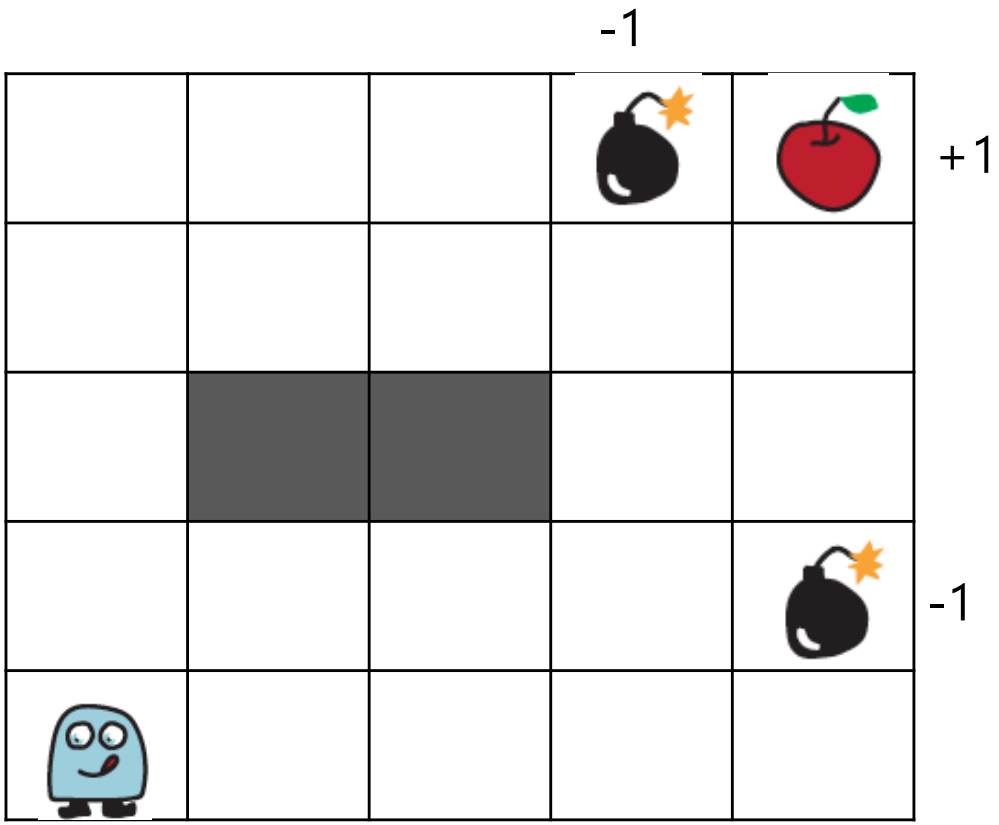
- 실행결과


0.12
0.10
0.08
0.06
0.04
0.02
0.00
loss
0
200
400
600
800
1000
episode


0.75
0.72
0.81
0.64
0.81
0.72
0.90
0.83
0.88
0.81
1.00
0.60
R 1.0 (GOAL)
0.73
0.65
0.65
0.61
0.83
0.42
-0.28
0.48
0.78
0.63
0.32
R -1.0
0.64
0.66
0.60
0.53
0.60
0.53
0.62
0.60
0.60
0.40
0.60
0.53
0.47
0.31
0.51
-0.33
0.58
0.37
0.63
→
→
→
R 1.0 (GOAL)
↑
↑
↑
R -1.0
↑
↓
↑
↓

Quiz

(Q3) Q-Network 를 적용하여 5x5 Grid World 에 대한 Q 테이블을 완성하고 policy 를 구하라. 단, 신경망의 최적화를 위한 파라메터를 설정하라.



요약

- Q-Learning vs. Q-Network

항목	Q-Learning	Q-Network
기본개념	테이블 기반 Q-value 업데이트	신경망 기반 Q-value 업데이트
Q-함수 표현	State-action 쌍에 대한 Q 값을 테이블로 저장 (작은/이산적 상태공간에 적합)	신경망 모델(regression) 로 Q 값 근사화 (큰/연속적 상태공간도 가능)
Update 방식	$Q'(S_t,A_t) = Q(S_t,A_t) + \alpha\{R_t + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)\}$	신경망 출력값=$Q(S_t,A_t)$ 신경망 목표값= $\gamma \max_a Q(S_{t+1},a) + R_t$ => Loss 를 기반으로 신경망 학습
행동 선택	ε-greedy 등 간단 전략 사용	ε-greedy 등 간단 전략
문제점	크거나 연속 상태공간 문제에 적용 어려움	신경망 성능에 결과 의존
적용 예	Grid World, 간단한 게임	Atari 게임, 로봇제어 등