5장

```python
import numpy as np

X = np.random.rand(5)
W = np.random.rand(5,9)
B = np.random.rand(9)

print(X)
print(W)
print(B)
```

```
[0.16061312 0.76206797 0.80134166 0.34947798 0.16709922]
[[0.51248447 0.13851247 0.79873295 0.43156687 0.48280852 0.5936369
  0.36324797 0.10980919 0.9127913 ]
 [0.82499349 0.36863136 0.07961857 0.99564781 0.57083533 0.26854116
  0.23541487 0.08200948 0.02620347]
 [0.55411233 0.79450971 0.35448589 0.1347516  0.22715001 0.11609005
  0.42756784 0.50149354 0.62965233]
 [0.84092671 0.06455441 0.96245314 0.90095084 0.71506776 0.19882273
  0.75952533 0.8961215  0.66617918]
 [0.77969457 0.76671412 0.31513666 0.42006237 0.24123473 0.55040548
  0.56006686 0.43534709 0.94495543]]
[0.86213131 0.63106572 0.58774316 0.65796372 0.20682857 0.51670451
 0.08042245 0.19610318 0.90962898]
```

```python
import numpy as np

X = np.random.rand(5)
W = np.random.rand(5,9)
B = np.random.rand(9)

print(X.shape)
print(W.shape)
print(B.shape)
Y = np.dot(X, W) + B
print(Y)
```

```
(5,)
(5, 9)
(9,)
[1.80874688 1.57178127 2.06560117 2.06707152 1.31830579 1.11001387
 1.72466339 1.88727654 0.7761859 ]
```

```python
#배치용 affine 계층에서 편향의 순전파 역전파
import numpy as np

X_dot_W = np.array([[0,0,0], [10,10,10]])
B = np.array([1,2,3])
Y = X_dot_W + B
print(Y)
```

```
[[ 1  2  3]
 [11 12 13]]
```

```python
dY = np.array([[1,2,3],[11,12,13]])
dB = np.sum(dY, axis=0)
print(dB)
```

```
[12 14 16]
```

```python
class Affine:
    def __init__(self,W,b):
        self.W = W
        self.b = b
        self.x = None
        self.dW =None
        self.db = None

        def forwar(self, x):
            self.x = x
            out + np.dot(x,self.W) + self.b
            return out

        def backward(self, out):
            dx = np.dot(dout, self.W.T)
            delf.dW = np.dot(Self.x.T, dout)
            self.db = np.sum(dout, axis=0)
            return dx
```

6장

```python
class Momentum:

    def __init__(self, lr=0.01, momentum=0.9):
        self.lr = lr
        self.momentum = momentum
        self.v = None
```

```python
def update(self, params, grads):
    if self.v is None:
        self.v = {}
        for key, val in params.items():
            self.v[key] = np.zeros_like(val)

    for key in params.keys():
        self.v[key] = self.momentum*self.v[key] - self.lr*grads[key]
        params[key] += self.v[key]
```

```python
class AdaGrad:

    def __init__(self, lr=0.01):
        self.lr = lr
        self.h = None

    def update(self, params, grads):
        if self.h is None:
            self.h = {}
            for key, val in params.items():
                self.h[key] = np.zeros_like(val)

        for key in params.keys():
            self.h[key] += grads[key] * grads[key]
            params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)
```

질문

affine 계층에서 행렬의 곱 계산 시 대응하는 차원의 원소 수를 일치시켜야 하는 이유는?

-변환을 행렬로 표현할 때, 입력 벡터와 출력 벡터의 차원이 일치해야 하기 때문이다. 일치하지 않으면 변환된 공간의 모양이나 크기가 유지되지 않는다. 행렬 연산의 관점에서 볼 때도 대응하는 차원의 원소 수를 일치시켜야 선형 변환을 할 수 있다.

오차역전파법의 핵심 아이디어는?

-오차역전파법의 핵심 아이디어는 기울기를 효율적으로 계산하여 신경망의 가중치를 업데이트하는 것이다. 손실 함수의 기울기를 신경망을 통해 역방향으로 전파하여 각 가중치와 편향에 대한 손실 함수의 기울기를 계산한다. 이 기울기를 사용하여 가중치를 조정하여 손실을 최소화한다.

모멘텀 최적화의 한계는?

-모멘텀 최적화의 성능은 모멘텀 값에 의존된다. 이 때, 모멘텀 값이 너무 작으면 시간이 오래 걸리고, 너무 크면 불안정해진다. 또한, 초기 가중치나 모멘텀 값을 잘못 설정하면 수렴에 문제가 생길 수 있다. 이 초기 설정을 조정하는 것이 어렵다.

모멘텀이 경사 하강법에 어떻게 도움을 주는가?

-성능을 향상시키는데 도움을 준다. 수렴 속도 향상, 경사 하강법의 효율성 향상, 불규칙한 그래디언트에 따른 튀는 현상 줄이기, 지역 최적점에 갇히는 문제를 완화 시킨다.