CHAPTER 1. 신경망 복습

1.1 수학과 파이썬 복습

1.1.1 벡터와 행렬

W = np.array([[1,2,3], [4,5,6]])

W.shape → (2,3) 다차원 배열의 형상

W.ndim → 2 차원수

1.1.2 행렬의 원소별 연산

1.1.3 브로드캐스트

1.1.4 벡터의 내적과 행렬의 곱

$$x \cdot y = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

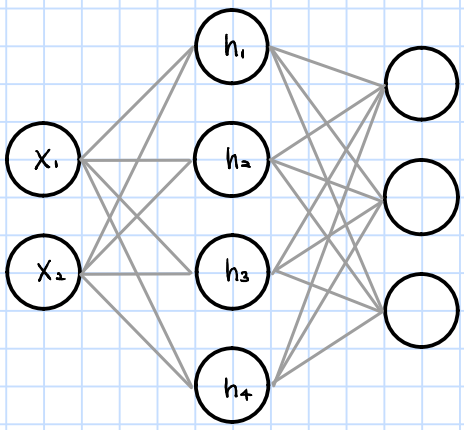
- 행렬의 곱 : 왼쪽 행렬의 행벡터와 오른쪽 행렬의 열벡터의 내적.

1.1.5 행렬 형상 확인

1.2 신경망의 추론

- "학습" 과 "추론"

1.2.1 신경망 추론 전체 그림



- $h_1 = x_1 w_{11} + x_2 w_{21} + b_1$
- $(h_1, h_2, h_3, h_4) = (x_1, x_2) \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{pmatrix} + (b_1, b_2, b_3, b_4)$

(1,4) (1,2) (2,4)

간소화

- $h = xW + b$

- 미니배치

X W = h

N x 2 2x4 N x 4

일치

- '비선형' 효과를 부여하기 위해 활성화함수 사용.

예) 시그모이드

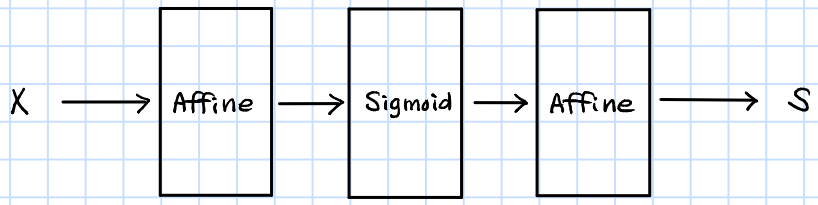
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

1. 2. 2. 계층을 클래스화 및 순전파 구현

- Affine : 완전연결 계층에 의한 변환
- Sigmoid : 시그모이드 함수에 의한 변환

✡ 구현 규칙

- 모든 계층은 forward() 와 backward() 메서드를 가진다.
- 모든 계층은 인스턴스 변수인 params 와 grads 를 가진다.
  - params : 가중치, 편향 등을 담는 리스트
  - grads : params에 저장된 각 매개변수에 대응하여, 해당 매개변수의 기울기를 보관하는 리스트



↳ TwoLayerNet 클래스로 추상화

```
x = np.random.randn(10, 2)
model = TwoLayerNet(2, 4, 3)
s = model.predict(x)
```

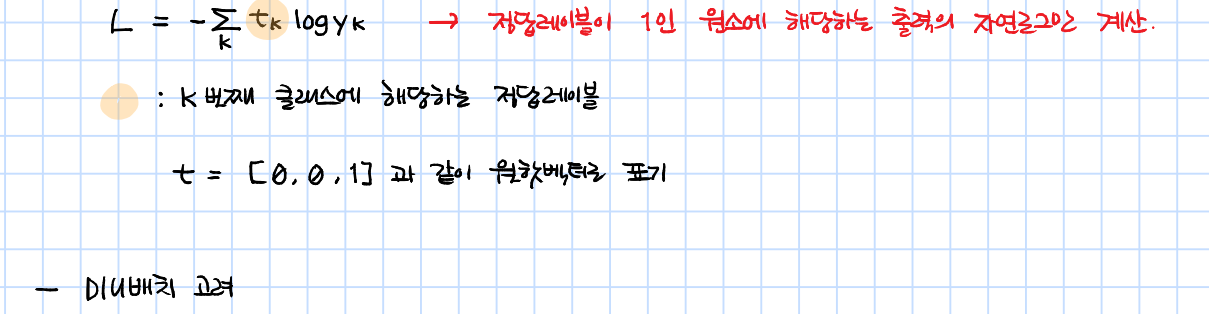
1.3 신경망의 학습

- 최적의 매개변수의 값을 찾는 작업.

1.3.1 손실함수

- 손실 : 학습데이터 (학습 시 주어진 정답데이터) 와 신경망이 예측한 결과를 비교,
  예측이 얼마나 나쁠가를 산출한 단일값 (스칼라)

- 손실함수 : 다중의 경우 Cross Entropy Error



- 소프트맥스 함수

$$y_k = \frac{\exp(s_k)}{\sum_{i=1}^{n} \exp(s_i)}$$

출력이 총 n개일 때,
k번째 출력 y_k를 구하는 계산식

y_k는 k번째 클래스에 해당하는 소프트맥스 함수의 출력.

$\exp(s_k)$: 점수 s_k의 지수 함수

$\sum_{i=1}^{n} \exp(s_i)$: 모든 입력신호의 지수 함수의 총합.

↓ 소프트맥스의 출력 (= 확률) 이 교차 엔트로피 오차에 입력됨.

- 교차 엔트로피의 수식

$$L = -\sum_k t_k \log y_k$$ → 정답레이블이 1인 원소에 해당하는 출력의 자연로그만 계산.

t_k : k번째 클래스에 해당하는 정답레이블

$t = [0, 0, 1]$ 과 같이 원핫벡터로 표기

- MI배치 고려

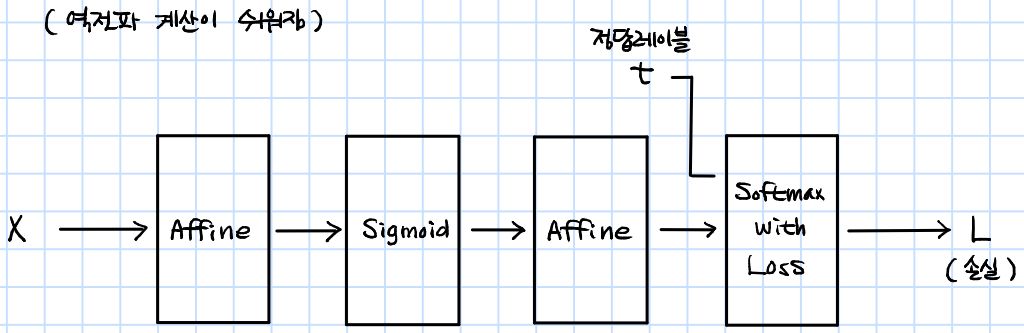
$$L = -\frac{1}{N}\sum_n \sum_k t_{nk} \log y_{nk}$$

t_{nk} : n번째 데이터의 k 차원 째 값 의미, 정답레이블

y_{nk} : 신경망의 출력.

- N개짜리로 확장.
- N으로 나눠서 '평균 손실 함수'를 구함.
  → MI배치의 크기에 관계없이 항상 일관된 척도를 얻을 수 있음.

- 소프트맥스와 교차 엔트로피 오차를 Softmax with Loss 계층으로 구현
  (역전파 계산이 수월함)



완전연결층에 의한 변환의 미니배치 버전

```
import numpy as np

W1 = np.random.randn(2,4)        # 가중치
b1 = np.random.randn(4)          # 편향
x = np.random.randn(10, 2)       # 입력        : 각 샘플데이터
h = np.matmul(x, W1) + b1
```

브로드캐스트 됨.

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

a = sigmoid(h)
```

종합

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

x = np.random.randn(10, 2)       # 입력        : 각 샘플데이터
W1 = np.random.randn(2,4)        # 가중치
b1 = np.random.randn(4)          # 편향
W2 = np.random.randn(4,3)        # → 은닉층 뉴런 4개, 출력층 뉴런 3개이므로 4x3 형상으로 설정.
b2 = np.random.randn(3)

h = np.matmul(x, W1) + b1
a = sigmoid(h)
s = np.matmul(a, W2) + b2
```

s → 최종출력, (10, 3)

: 10개의 데이터가 한꺼번에 처리,
각 데이터는 3차원 데이터로 변환됨.

```
import numpy as np

class Sigmoid:
    def __init__(self):
        self.params = []

    def forward(self, x):
        return 1 / (1 + np.exp(-x))
```

```
class Affine :

    def __init__(self, W, b):

        self.params = [W, b]

    def forward(self, x):

        W, b = self.params

        out = np.matmul(x, W) + b

        return out
```

→ 초기화될 때 가중치와 편향을 받음.
(신경망이 학습될 때 수시로 개선)

```
Class  TwoLayerNet :

    def __init__ (self, input_size, hidden_size, output_size):

        I, H, O = input_size, hidden_size, output_size

        # 가중치와 편향 초기화

        W1 = np.random.randn (I, H)

        b1 = np.random.randn (H)

        W2 = np.random.randn (H, O)

        b2 = np.random.randn (O)

        # 계층 생성

        self.layer = [ Affine (W1, b1),

                       Sigmoid (),

                       Affine (W2, b2) ]

        # 모든 가중치를 리스트에 모은다.

        self.params = []

        for layer in self.layers :

            self.params += layer.params        → 모든 계층은 자신의 학습 매개변수들을
                                                  인스턴스 변수인 params에 보관하고 있으므로
                                                  변수들을 더해주기만 하면 됨!

def predict (self, x) :

    for layer in self.layers :

        x = layer.forward (x)

    return x
```