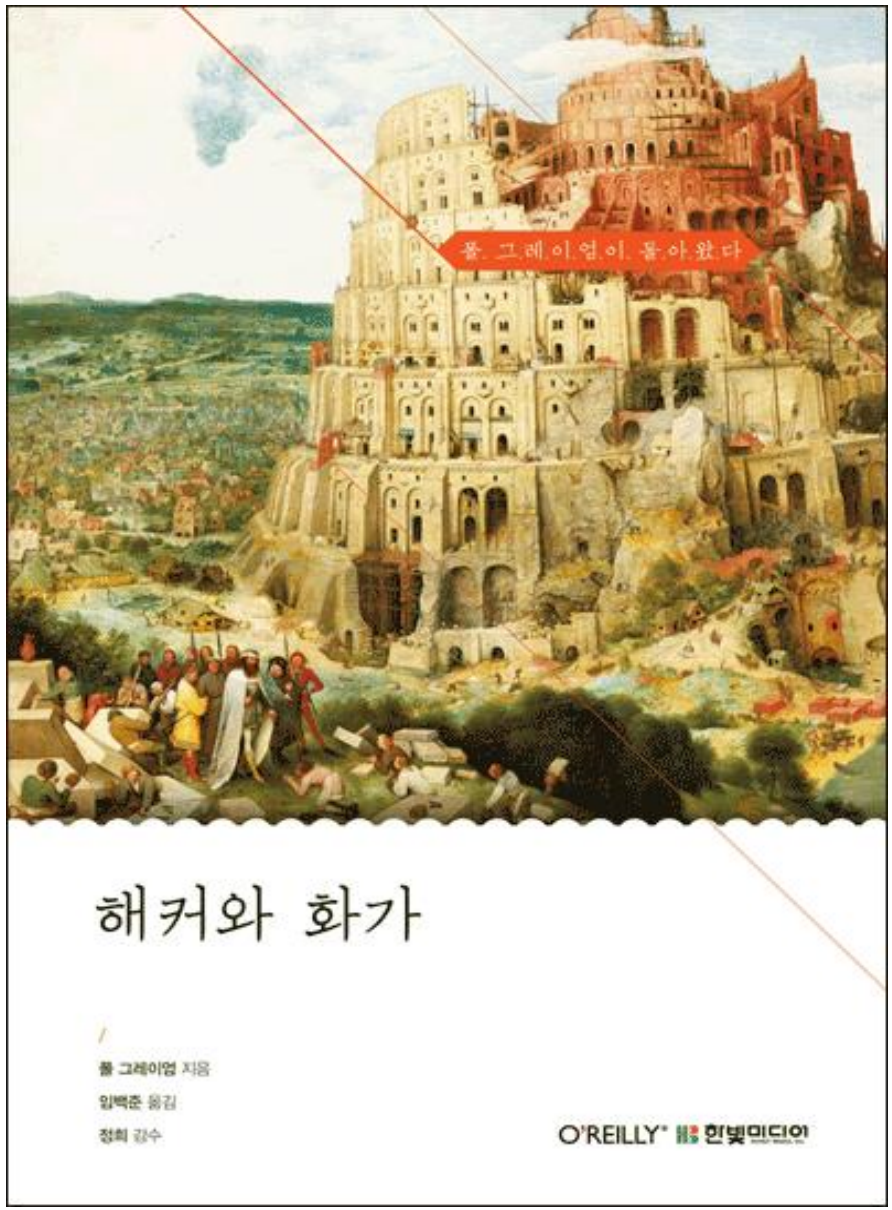
CHAPTER3 신경망



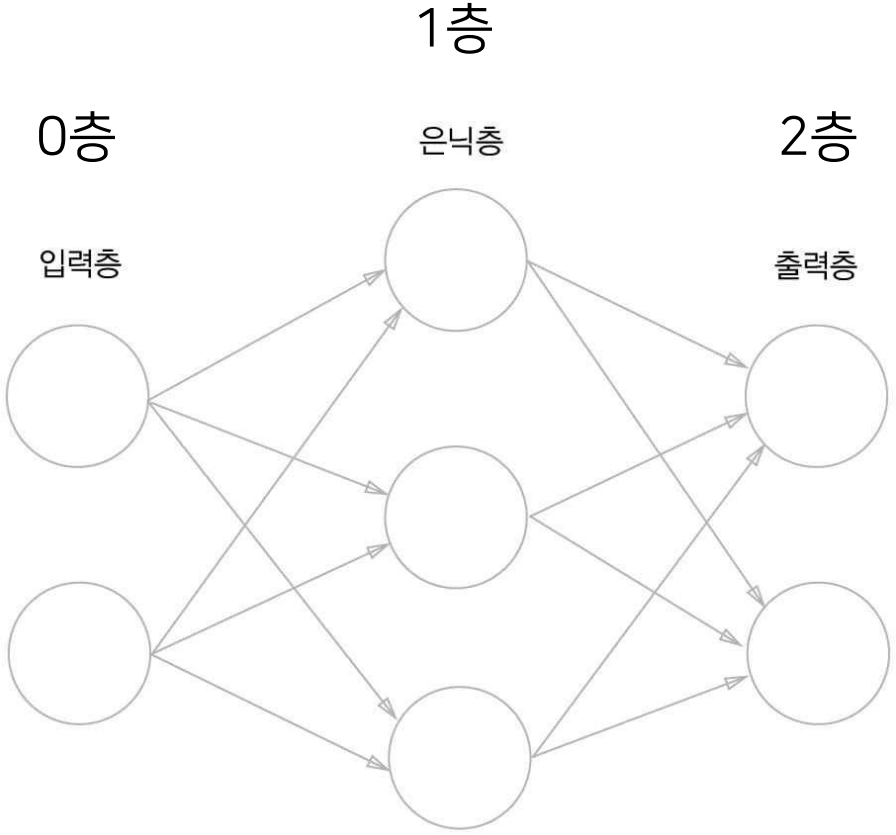
폴 그레이엄

폴 그레이엄은 프로그래머이자, 벤처 기업 투자가, 수필가이다. 리스프에 대한 그의 작업으로 유명하고, 지금은 야후! 스토어가 된 비아웹을 공동 창업한 것으로도 유명하다. 그 밖에 해커와 화가라는 수필집을 저술하였으며, 초기 투자 자금을 운용하는 회사인 와이 콤비네이터를 공동 창업하기도 했다. 가장 최근에는 해커 뉴스를 론칭했다. 위키백과

출생: 1965년, 영국 웨이머스

퍼셉트론에서 신경망으로

- 신경망의 예
  - 입력층 : 맨 왼쪽 줄
  - 은닉층 : 중간 줄(입력층과 중간층과 달리 사람눈에 보이지 않음)
  - 출력층 : 맨 오른쪽 줄



2층신경망 (가중치 갖는 층 기준)

퍼셉트론에서 신경망으로

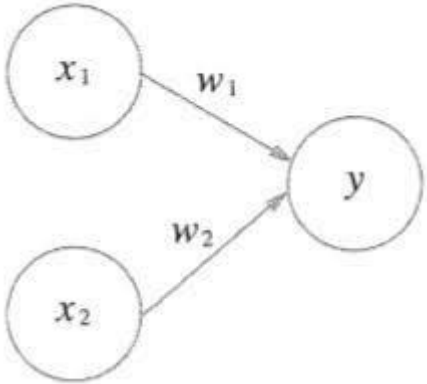
- 퍼셉트론 : 다수의 신호를 입력받아 하나의 신호로 출력하는 것

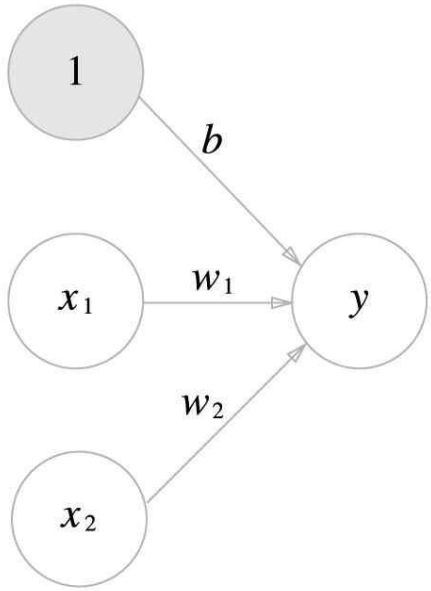
$$y = \begin{cases} 0 & (b + w_1 x_1 + w_2 x_2 \le 0) \\ 1 & (b + w_1 x_1 + w_2 x_2 > 0) \end{cases}$$

$$y = h(b + w_1 x_1 + w_2 x_2)$$

$$\Longrightarrow \; h(x) = \begin{cases} 0 & (x \le 0) \\ 1 & (x > 0) \end{cases}$$

- b : 편향, 뉴런이 얼마나 쉽게 활성화 되는냐를 제어
- w1, w2 : 가중치, 각 신호의 영향력을 제어



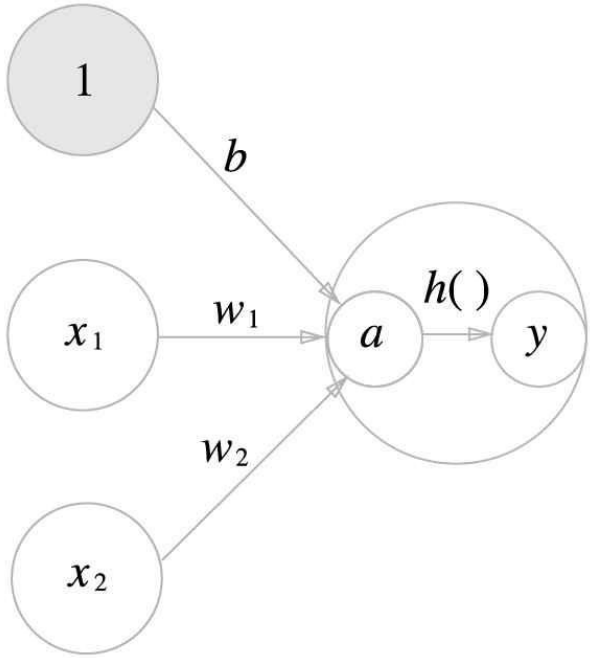


퍼셉트론에서 신경망으로

- 활성화 함수
  - 입력신호의 총합을 출력신호로 변환하는 함수
  - 입력신호의 총합이 활성화를 일으키는지 정하는 역할

1단계 $a = b + w_1 x_1 + w_2 x_2$

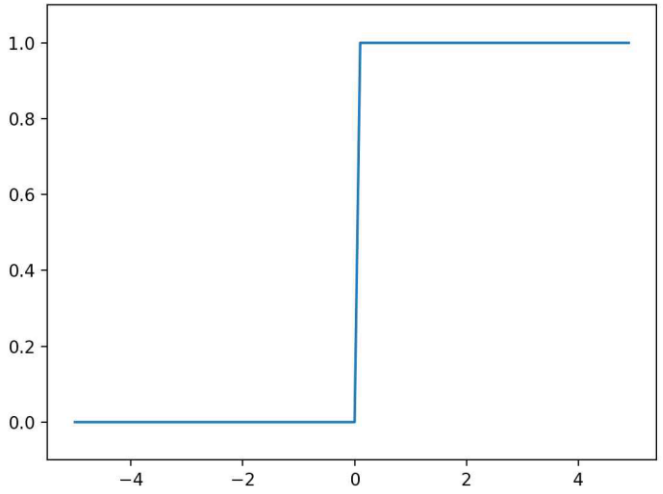
2단계 $y = h(a)$



활성화 함수

- 계단함수
  - 임계값을 경계로 출력이 바뀌는 함수
    - 입력이 0을 넘으면 1을 출력, 그 외에는 0을 출력
  - 퍼셉트론에서 활성화 함수로 계단함수를 이용

```
import numpy as np
import matplotlib.pylab as plt

def step_function(x):
    return np.array(x > 0, dtype = np.int)

x = np.arange(-5.0, 5.0, 0.1) # -5.0이상 5.0미만 0.1간격의 넘파이 배열 생성
y = step_function(x)
plt.plot(x,y)
plt.ylim(-0.1, 1.1) # y축의 범위 지정
plt.show()
```

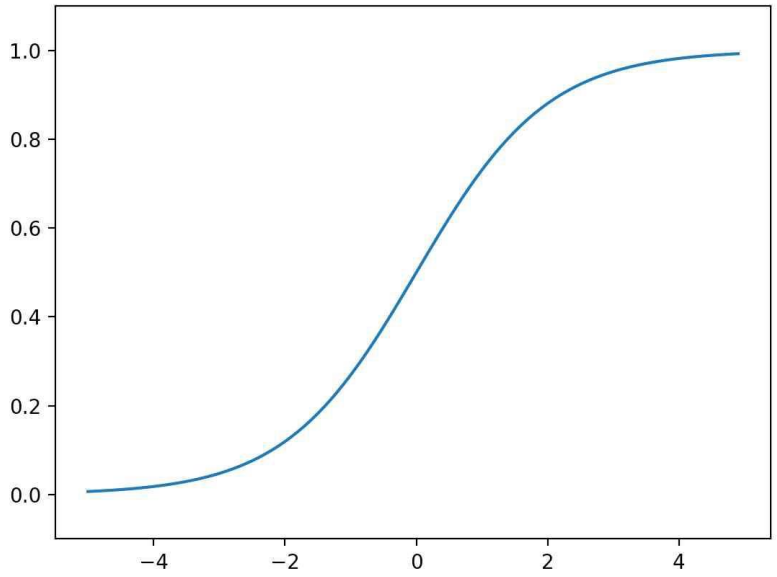


활성화 함수

- 시그모이드 함수(S자 함수)
  - 신경망에서 자주 이용하는 활성화 함수

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

```
import numpy as np
import matplotlib.pylab as plt

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

x = np.arange(-5.0, 5.0, 0.1)
y = sigmoid(x)
plt.plot(x, y)
plt.ylim(-0.1, 1.1)
plt.show()
```



활성화 함수

- 시그모이드 함수와 계단함수의 비교

		계단함수	시그모이드함수
차이점	매끄러움	매끄럽지 않음	매끄러움
	반환값	0 이나 1	실수
	활성화함수	퍼셉트론	신경망
공통점		입력된 값이 작을때 출력이 0에 가까워지고(혹은 0) 큰 값인 경우에는 1에 가까워짐	
		입력이 아무리 작거나 커도 출력은 0에서 1사이	
		활성화함수이다	
		비선형함수이다	

활성화 함수

- 선형함수
  - 출력이 입력의 상수배만큼 변하는 함수
  - f(x) = ax + b ➔ 곧은 1개의 직선
  - 층을 아무리 깊게 해도 은닉층이 없는 네트워크로 표현될 수 있음

- 비선형함수
  - 선형이 아닌 함수(직선 1개로 그릴 수 없는 함수)

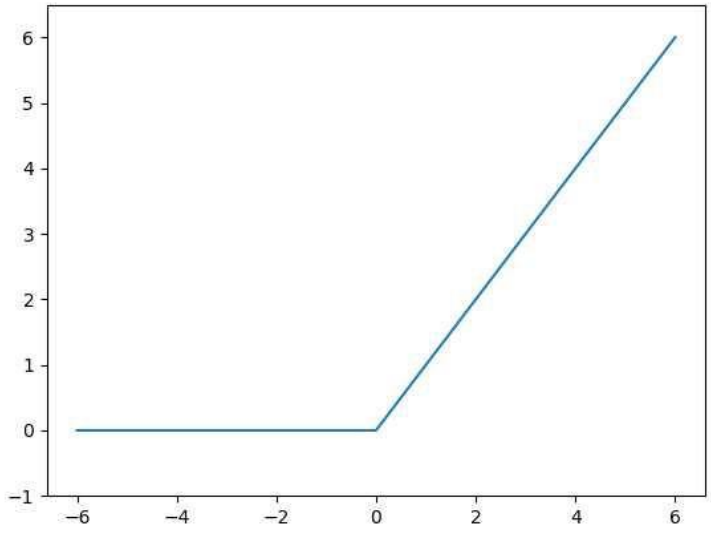
✔신경망에서는 활성화함수로 비선형함수를 사용해야함
  - 선형함수를 이용하면 신경망의 층을 깊게 하는 의미가 없어지기 때문

활성화 함수

- Relu함수
  - 입력이 0을 넘으면 그 입력을 그대로 출력, 0 이하이면 0을 출력

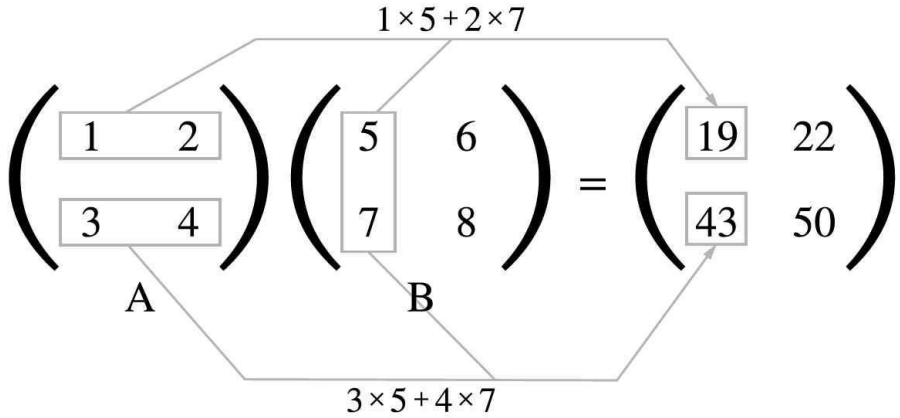
$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

```
import numpy as np
import matplotlib.pylab as plt

def relu(x):
    return np.maximum(0,x)

x = np.arange(-6.0, 7.0, 2.0)
y = relu(x)
plt.plot(x,y)
plt.ylim(-1.0, 6.5)
plt.show()
```

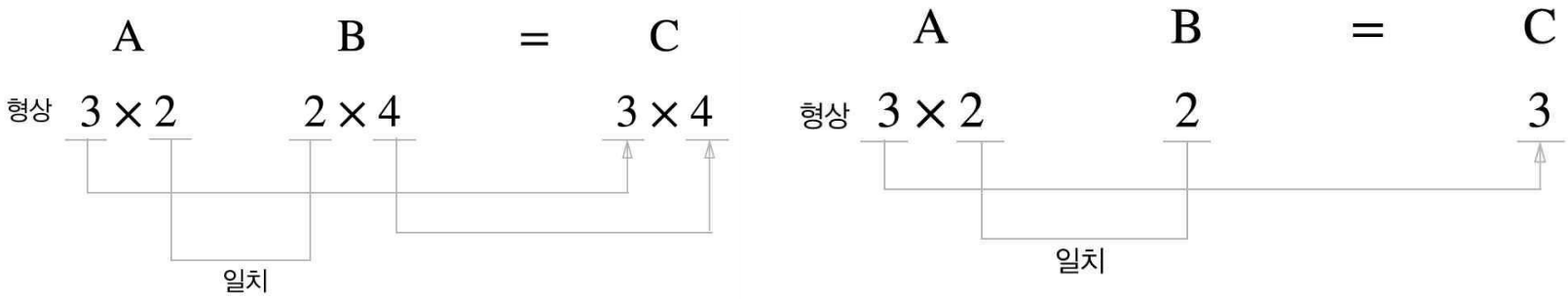


다차원 배열 계산

- <행렬>
  - 행렬의 내적



✔행렬의 곱에서는 대응하는 차원의 원소 수를 일치시켜야함



다차원 배열 계산

<A와 B가 모두 2차원 배열인 경우>

```
import numpy as np

A = np.array([[1, 2], [3, 4]])
print(A.shape) # 배열의 형상 확인
B = np.array([[5, 6], [7, 8]])
print(B.shape)
C = np.dot(A, B) # 배열의 내적
print(C)
print(np.ndim(C)) # 배열의 차원 수 확인
```

```
(2, 2)
(2, 2)
[[19 22]
 [43 50]]
2
```

<A가 2차원, B가 1차원 배열인 경우>

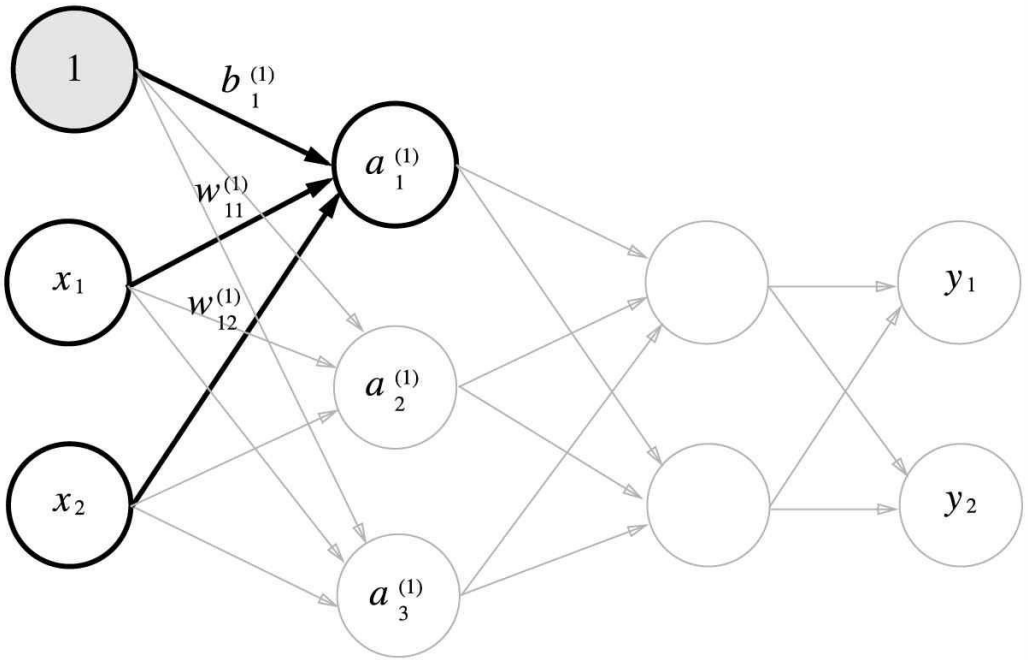
```
import numpy as np

A = np.array([[1, 2], [3, 4], [5, 6]])
print(A.shape) # 배열의 형상 확인
B = np.array([7, 8])
print(B.shape)
C = np.dot(A, B) # 배열의 내적
print(C)
print(np.ndim(C)) # 배열의 차원 수 확인
```

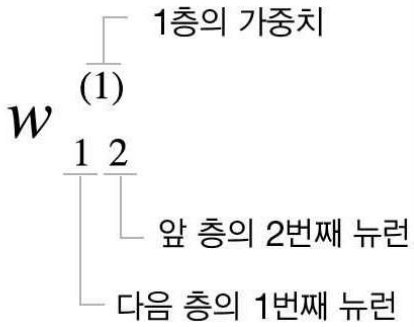
```
(3, 2)
(2,)
[23 53 83]
1
```

3층 신경망 구현하기

- 3층 신경망에서 수행되는 입력부터 출력까지의 처리(순방향 처리) 구현

<입력층에서 1층으로 신호 전달>



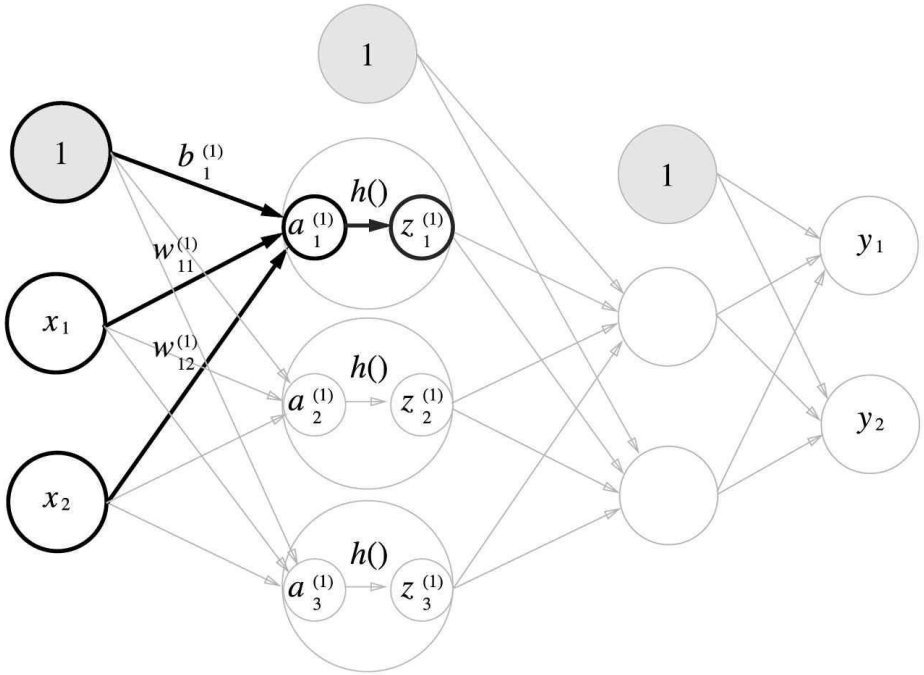


$$a_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)}$$

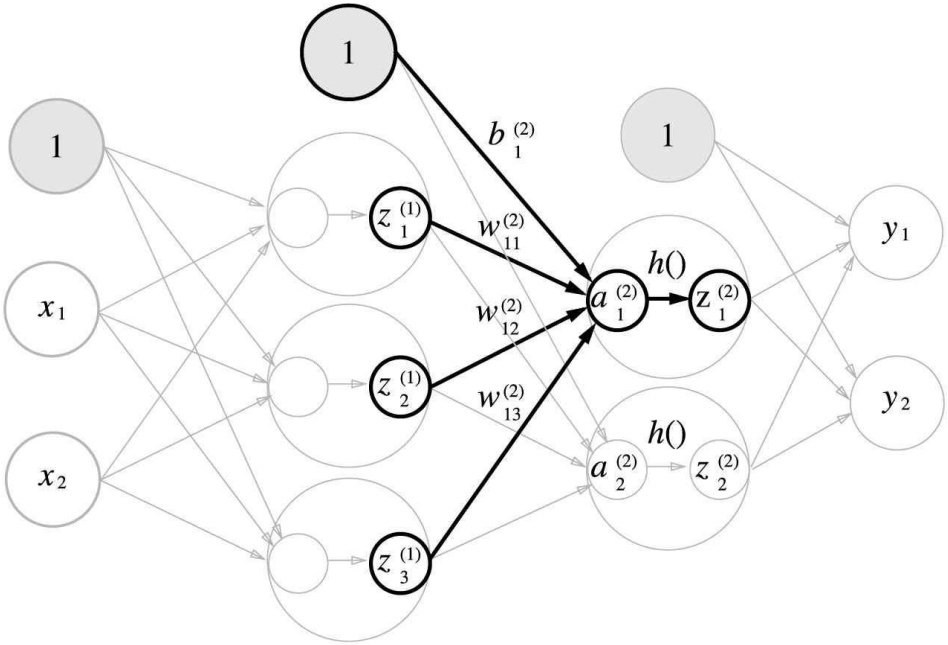
$$\mathbf{A}^{(1)} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{B}^{(1)}$$

3층 신경망 구현하기

<입력층에서 1층으로의 신호 전달>



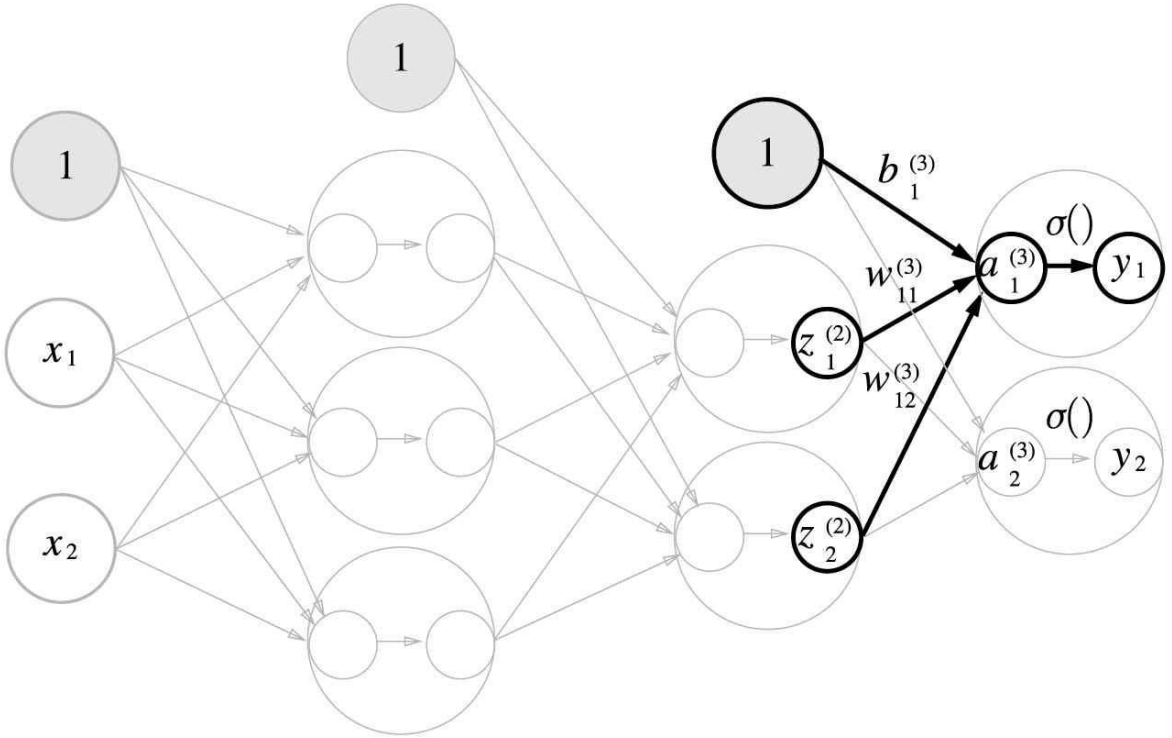
<1층에서 2층으로의 신호 전달>



※ 여기서는 활성화 함수로 시그모이드 함수를 사용!

3층 신경망 구현하기

<2층에서 출력층으로의 신호 전달>



※출력층의 활성화 함수로 항등함수(입력을 그대로 출력하는 함수)를 사용!

3층 신경망 구현하기

```python
import numpy as np

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

def identity_function(x): #항등함수
    return x

def init_network(): #가중치와 편향 초기화
    network = {}
    network['W1'] = np.array([[0.1, 0.3, 0.5], [0.2, 0.4, 0.6]])
    network['b1'] = np.array([0.1, 0.2, 0.3])
    network['W2'] = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
    network['b2'] = np.array([0.1, 0.2])
    network['W3'] = np.array([[0.1, 0.3], [0.2, 0.4]])
    network['b3'] = np.array([0.1, 0.2])

    return network
```

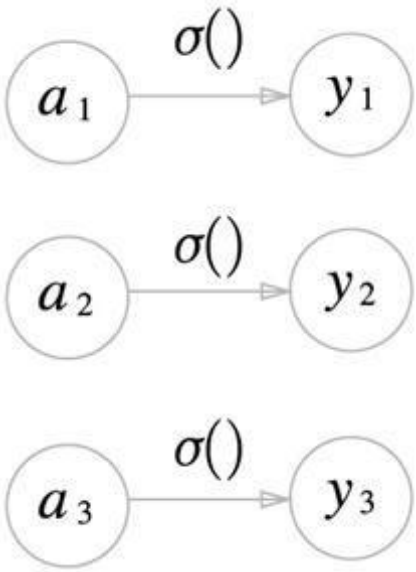
```python
def forward(network, x): #입력신호를 출력으로 변환
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    a1 = np.dot(x, W1) + b1
    z1 = sigmoid(a1)
    a2 = np.dot(z1, W2) + b2
    z2 = sigmoid(a2)
    a3 = np.dot(z2, W3) + b3
    y = identity_function(a3)

    return y

network = init_network()
x = np.array([1.0, 0.5]) #입력신호
y = forward(network, x) #출력
print(y)
```

```
[0.31682708 0.69627909]
```

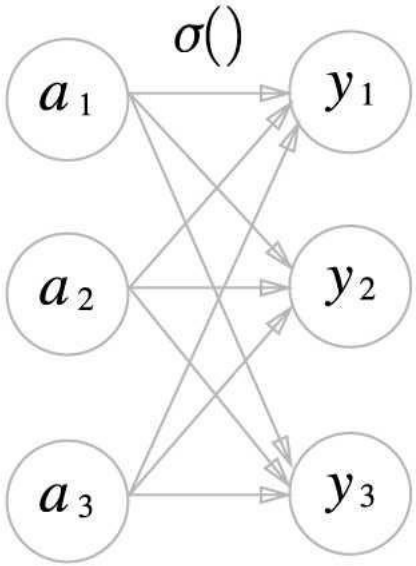
❝ 신경망에서의 계산을 행렬 계산으로 정리 가능 ❞

출력층 설계하기

<항등함수>
-회귀에 사용



<소프트맥스함수>
-분류에 사용



$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

출력층 설계하기

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n}\exp(a_i)} = \frac{C\exp(a_k)}{C\sum_{i=1}^{n}\exp(a_i)}$$

$$= \frac{\exp(a_k + \log C)}{\sum_{i=1}^{n}\exp(a_i + \log C)}$$

$$= \frac{\exp(a_k + C')}{\sum_{i=1}^{n}\exp(a_i + C')}$$

➔ 소프트맥스의 지수 함수를 계산할 때 어떤 정수를 더해도 (혹은 빼도) 결과는 바뀌지 않는다.

```python
import numpy as np

def softmax(a):
    c = np.max(a)
    exp_a = np.exp(a-c) # 오버플로 대책
    sum_exp_a = np.sum(exp_a)
    y = exp_a / sum_exp_a

    return y
```

출력층 설계하기

- 소프트맥스 함수의 특징

```
import numpy as np

def softmax(a):
    c = np.max(a)
    exp_a = np.exp(a-c) # 오버플로 대책
    sum_exp_a = np.sum(exp_a)
    y = exp_a / sum_exp_a

    return y

a = np.array([0.3, 2.9, 4.0])
y = softmax(a)
print(y)
print(np.sum(y))
```

⬇

```
[0.01821127 0.24519181 0.73659691]
1.0
```

✔ 함수의 출력은 0에서 1.0사이의 실수이다.
✔ 출력의 총합은 1이된다

⬇

❝ 확률로 해석가능 ❞

손글씨 숫자 인식

- MNIST 데이터셋
  - 0부터 9까지의 숫자 이미지로 구성됨
  - 훈련이미지가 6000장(학습), 시험이미지가 1000장(분류)
  - 28×28 크기의 회색조 이미지
  - 각 픽셀은 0~255까지의 값을 취함
  - 각 이미지에 그 이미지가 실제 의미하는 숫자가 레이블로 붙어있음

손글씨 숫자 인식

- 신경망의 추론 처리
  - 입력층 뉴런을 784개, 출력층 뉴런을 10개로 구성
    ※출력층의 뉴런 수는 풀려는 문제에 맞게 적절히 정해야함 (분류에서는 분류하고 싶은 클래스 수로 설정)
  - 은닉층은 총 2개, 각 은닉층 뉴런의 개수는 임의로 결정

  - 배치처리
    - 배치란?: 하나로 묶은 입력 데이터

※ 데이터를 배치로 처리함으로써 효율적이고 빠르게 처리할 수 있음
(일반적으로 100장에서 1000장을 묶음)

손글씨 숫자 인식

```python
import sys, os
sys.path.append(os.pardir) #부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import pickle
from mnist import load_mnist
from PIL import Image # 이미지 표시에 PIL모듈 사용

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

def softmax(x):
    c = np.max(x)
    exp_x = np.exp(x - c)
    sum_exp_x = np.sum(exp_x)
    y = exp_x/sum_exp_x

    return y

def get_data():
    # (훈련이미지, 훈련레이블), (시험이미지, 시험레이블)형식으로 반환
    (x_train, t_train), (x_test, t_test) = load_mnist(normalize = True, flatten = True, one_hot_label = False)
    return x_test, t_test

def init_network():
    with open("sample_weight.pkl", 'rb') as f:
        network = pickle.load(f)
    return network

def predict(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    a1 = np.dot(x, W1) + b1
    z1 = sigmoid(a1)
    a2 = np.dot(z1, W2) + b2
    z2 = sigmoid(a2)
    a3 = np.dot(z2, W3) + b3
    y = softmax(a3)

    return y
```

```python
x, t = get_data()
network = init_network()

batch_size = 100 # 배치 크기
accuracy_cnt = 0

for i in range(0, len(x), batch_size):
    x_batch = x[i:i+batch_size] # 앞에서 부터 100장씩 꺼내 묶어지게됨
    y_batch = predict(network, x_batch)
    p = np.argmax(y_batch, axis = 1) # 1번째 차원을 구성하는 각 원소 중 최댓값의 인덱스 찾도록 한 것
    accuracy_cnt += np.sum(p == t[i:i+batch_size])

print("Accuacy:" + str(float(accuracy_cnt)/len(x))) # 정확도 평가
```

결과 : Accuacy:0.9352

Exercise 1

- 다음 각 활성화 함수들을 설명하고 python으로 구현하시오.
  - Step function
  - Sigmoid
  - Relu
  - 단 각 함수들은 다차원의 nd.array를 받아 처리할수 있어야 한다.

- 출력층에서 다중 클래스 분류 문제에 주로 쓰이는 다음 함수의 개념을 설명하고 python으로 구현하시오.
  - Softmax (overflow 대책 포함)
  - 단, 각 함수는 다차원의 nd.array를 받아 처리할수 있어야 한다.

Exercise 2

- 3.4.3 절의 구현예제를 참고하여 다음 spec을 만족하는 신경망을 구현하시오.
  - Hidden layer의 개수는 4개로 한다. (4층)
  - 3층을 중간에 추가 하고 이때 활성화 함수는 Relu로 한다. 1,2 층 및 4층은 그대로 설정.
  - 3 의 W, B의 형상은 2층의 output과 np.dot이 가능하도록 설정하고, 값은 임의로 설정하시오.
  - 입력값 x=np.array([1.0, 0.5])를 넣었을때 y는 어떻게 변하는지 확인하시오.

Exercise 3

- 3.6.3 절의 MNIST 학습 예제에서 batch size를 200으로 설정하고 수행해 보시오 .

Exercise 4

- 1번에서 작성한 Account 클래스를 관리하는 무한은행 프로그램을 만드시오.
  - 각 개인의 계좌는 Account 클래스의 객체를 만들어 사용한다.
  - 생성한 개개인의 은행계좌들(Account 클래스의 객체)는 Dictionary에 넣어서 관리한다.
    - **Key: 계좌번호(account)**
    - **Value: Account 객체**
  - Deposit과 withdraw시 Dictionary에서 해당 계좌로 고객의 Account 클래스의 객체를 찾아 그 객체의 balance에 더하거나 차감.

Exercise 4

- 실행 예)

무한은행 Menu
1. 계좌개설
2. 입금
3. 출금
4. 고객 잔액 조회
5. 전체 고객 조회
6. 종료

Select menu: 1
Account? 1111
Name? 유재석
Success! "유재석 고객님!"
계좌개설을 축하합니다."
Now, Balance is 0.

무한은행 Menu
1. 계좌개설
2. 입금
3. 출금
4. 고객 잔액 조회
5. 전체 고객 조회
6. 종료

Select menu: 2
Account? 1111
How much? 10000
Success! "유재석 고객님!"
Now, Balance is 10000.

Exercise 4

무한은행 Menu
1. 계좌개설
2. 입금
3. 출금
4. 고객 잔액 조회
5. 전체 고객 조회
6. 종료

Select menu: 3
Account? 1111
How much? 5000
Success! "유재석 고객님!"
Now, Balance is 5000

무한은행 Menu
1. 계좌개설
2. 입금
3. 출금
4. 고객 잔액 조회
5. 전체 고객 조회
6. 종료

Select menu: 4
Account? 1111
Success! "유재석 고객님!"
Now Balance is 5000.

Exercise 4

무한은행 Menu
1. 계좌개설
2. 입금
3. 출금
4. 고객 잔액 조회
5. 전체 고객 조회
6. 종료

Select menu: 1
Account? 2222
Name? 박명수
Success! "박명수 고객님!"
계좌개설을 축하합니다."
Now, Balance is 0.

무한은행 Menu
1. 계좌개설
2. 입금
3. 출금
4. 고객 잔액 조회
5. 전체 고객 조회
6. 종료

Select menu: 2
Account? 2222
How much? 20000
Success! "박명수 고객님!"
Now, Balance is 20000.

Exercise 4

무한은행 Menu
1. 계좌개설
2. 입금
3. 출금
4. 고객 잔액 조회
5. 전체 고객 조회
6. 종료
Select menu: 5
Account: 1111
Name: 유재석
Balance: 5000

Account: 2222
Name: 박명수
Balance: 20000

무한은행 Menu
1. 계좌개설
2. 입금
3. 출금
4. 고객 잔액 조회
5. 전체 고객 조회
6. 종료
Select menu: 6
Bye~Bye~

References

- 밑바닥부터 시작하는 딥러닝 (파이썬으로 익히는 딥러닝 이론과 구현), 사이토 고키, 2017
- 소스코드
  - https://github.com/WegraLee/deep-learning-from-scratch
- 창원대학교 정보 시각화 연구실 (Information Visualization Lab) 2018 하계 딥러닝 세미나 자료
- 스터디 자료
  - http://cafe.naver.com/architect1.cafe
  - https://nbviewer.jupyter.org/github/SDRLurker/deep-learning/