DL-Neural Network

Computational Linguistics @ Seoul National University

DL from Scratch

By Hyopil Shin

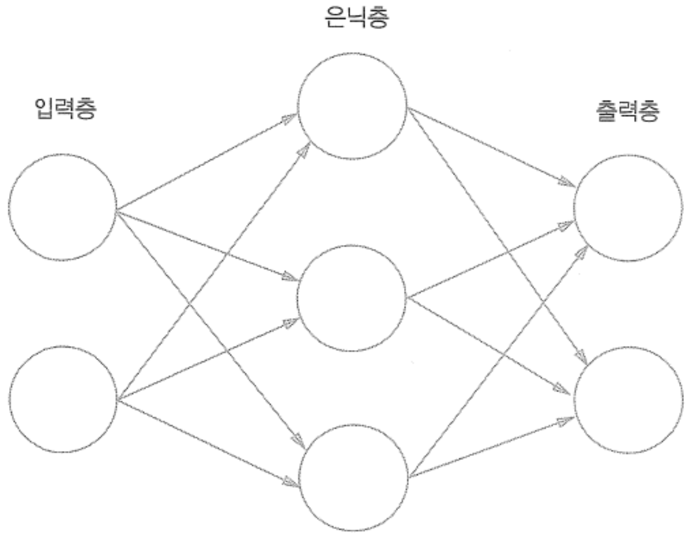
perceptron

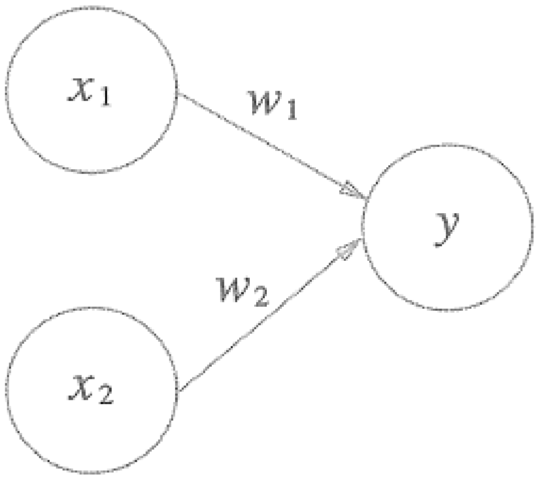
- Perceptron으로 복잡한 처리도 표현 가능
- 가중치를 결정하는 것이 복잡하고 수동으로 이루어짐
- Neural Network
  - 가중치 매개변수의 적절한 값을 데이터로부터 자동으로 학습하는 능력

Neural Network

- Simple Neural Network
  - 입력층, 은닉층, 출력층

그림 3-1 신경망의 예





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

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

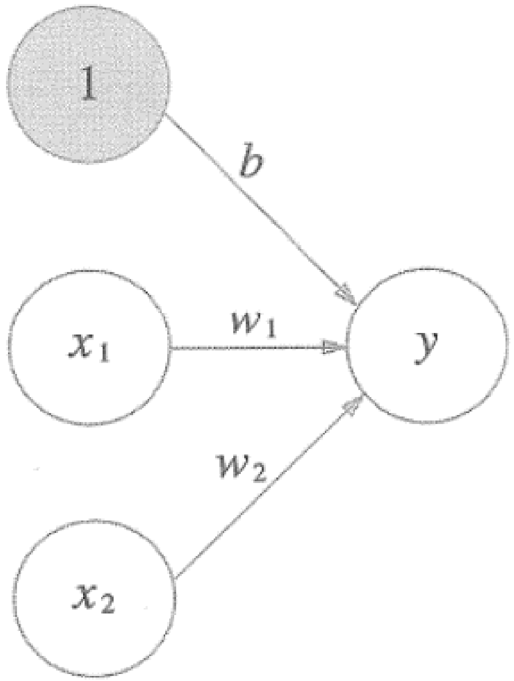
Neural Network

- X1, x2, 1이라는 세 개의 신호가 뉴런에 입력되어, 각 신호에 가중치를 곱한 후, 다음 뉴런에 전달
- Activation Function(활성화 함수)

$$y = h(b + w_1x_1 + w_2x_2)$$

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

그림 3-3 편향을 명시한 퍼셉트론



Activation Function

- 활성화 함수: 입력 신호의 총합이 활성화를 일으키는지 결정하는 역할을 하는 함수

$$a = b + w_1x_1 =+ w_2x_2$$
$$y = h(a)$$

그림 3-5 왼쪽은 일반적인 뉴런, 오른쪽은 활성화 처리 과정을 명시한 뉴런(a는 입력 신호의 총합, $h()$는 활성화 함수, y는 출력)

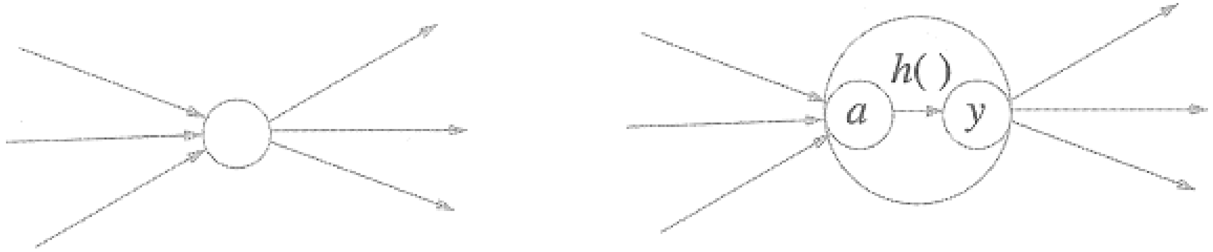
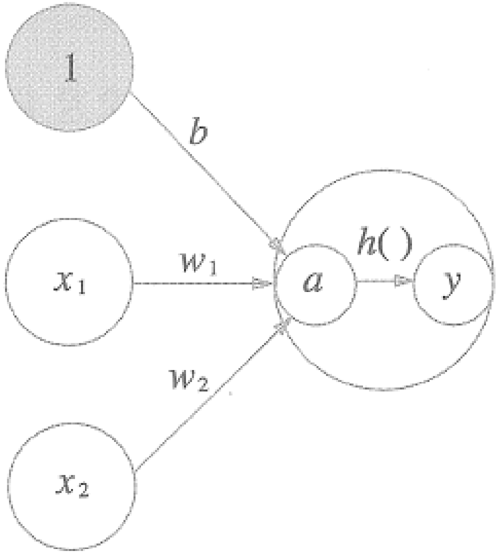


그림 3-4 활성화 함수의 처리 과정



Activation Function

- Step function
  - 활성화 함수는 임계값을 경계로 출력이 바뀜
  - 이를 계단 함수(step function) → perceptron
  - 다른 함수는?
- Sigmoid Function

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

Step function

- step_function.py

```
def step_function(x):
    if x > 0:
        return 1
    else:
        return 0
```

```
def step_function(x):
    y = x > 0
    return y.astype(np.int)
```

```
>>> import numpy as np
>>> x = np.array([-1.0, 1.0, 2.0])
>>> x
array([-1.,  1.,  2.])
>>> y = x > 0
>>> y
array([False,  True,  True], dtype=bool)
```

```
>>> y = y.astype(np.int)
>>> y
array([0, 1, 1])
```

Activation Function

- Step function Graph
- [Sigmoid function]
- [Sig_step_compare]

그림 3-6 계단 함수의 그래프

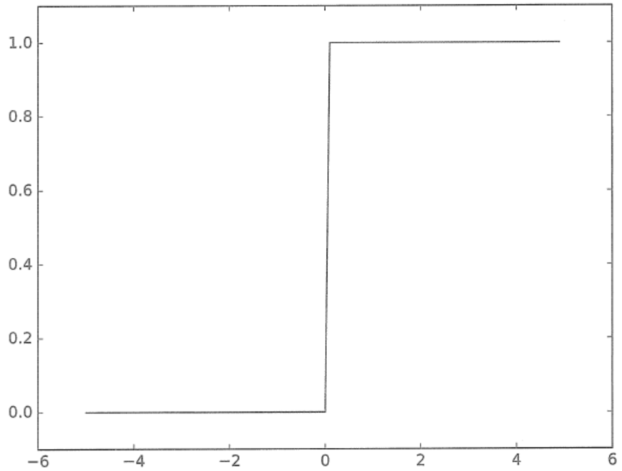


그림 3-7 시그모이드 함수의 그래프*

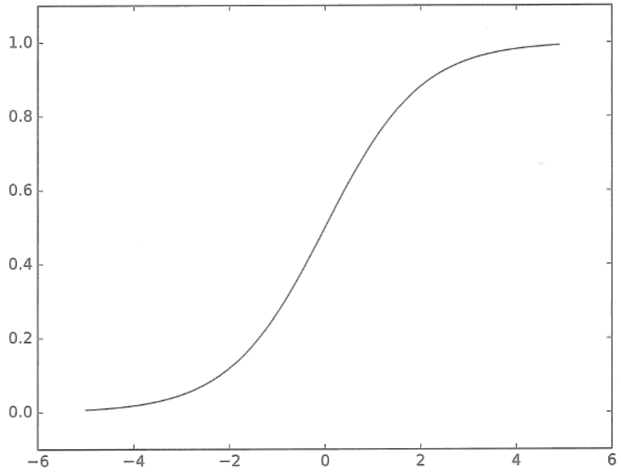
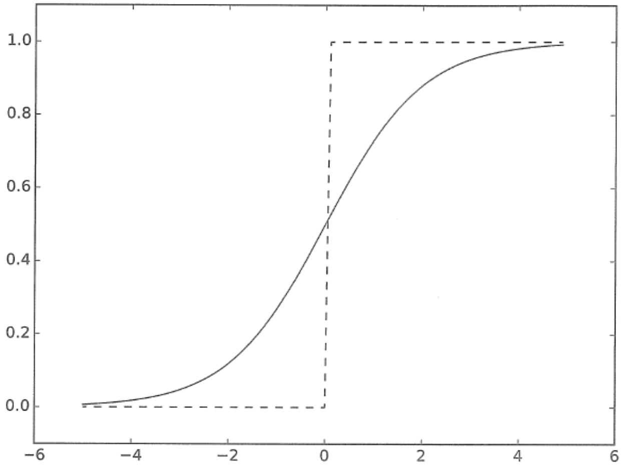


그림 3-8 계단 함수(점선)와 시그모이드 함수(실선)

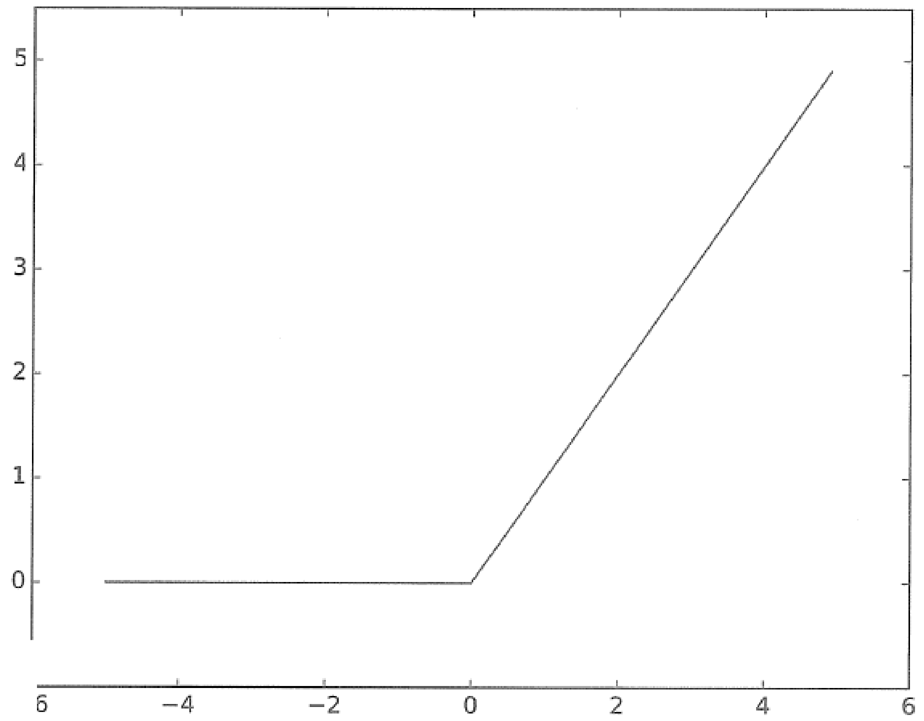


Activation Function

- Non-linear function
  - Step, sigmoid function
  - For hidden layers
- ReLU (Rectified Linear Unit)
  - 입력이 0을 넘으면 그 입력을 그대로 출력하고, 0이하이면 0을 출력하는 함수
  - [Relu.py]()

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

그림 3-9 ReLU 함수의 그래프*

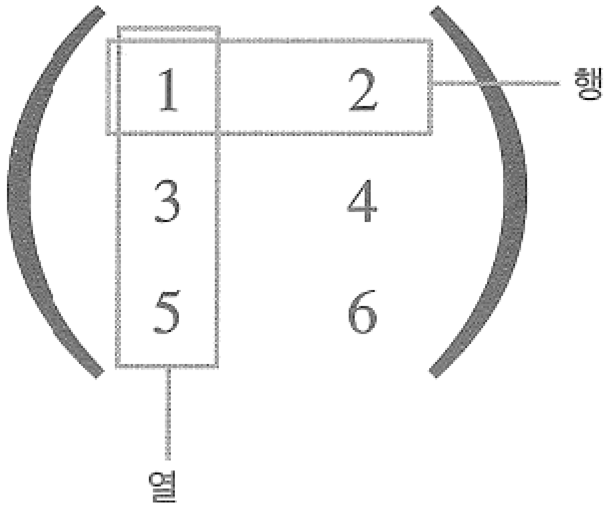


N-dimensional Array

```
>>> import numpy as np
>>> A = np.array([1, 2, 3, 4])
>>> print(A)
[1 2 3 4]
>>> np.ndim(A)
1
>>> A.shape
(4,)
>>> A.shape[0]
4
```

```
>>> B = np.array([[1,2], [3,4], [5,6]])
>>> print(B)
[[1 2]
 [3 4]
 [5 6]]
>>> np.ndim(B)
2
>>> B.shape
(3, 2)
```

그림 3-10 2차원 배열(행렬)의 행(가로)과 열(세로)



1차 배열 주의

```
a1 = np.array( [1, 2, 3] ) #크기 (3,)인 1차원 배열
a2 = np.array( [ [1, 2, 3] ] ) #크기 (1,3)인 2차원 배열 (행벡터)
a3 = np.array( [ [1], [2], [3] ] ) #크기 (3,1)인 2차원 배열 (열벡터)
```

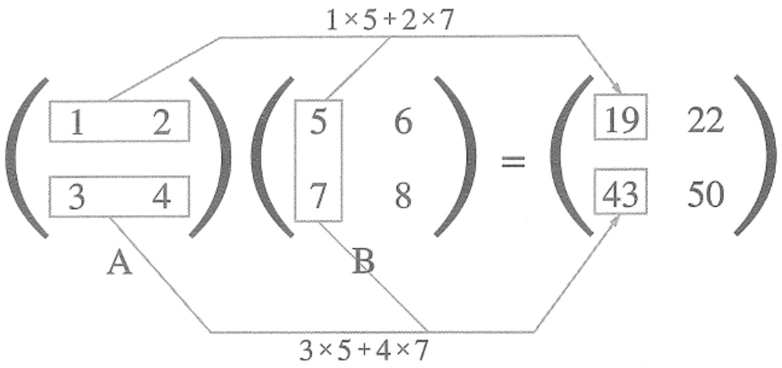
- **a1.T 는 동작하지 않는다**. 반면 a2.T 와 a3.T는 동작한다. 1차 배열은 행벡터나 열벡터 두 가지 모두로 취급되기도 한다.

- 1차 배열은 행벡터나 열벡터 둘 다로 취급할 수 있다. dot(A,v) 에서 v는 열벡터로 다루어지고 dot(v,A)에서는 행벡터로 취급된다. 따라서 전치를 복잡하게 수행할 필요가 없다.

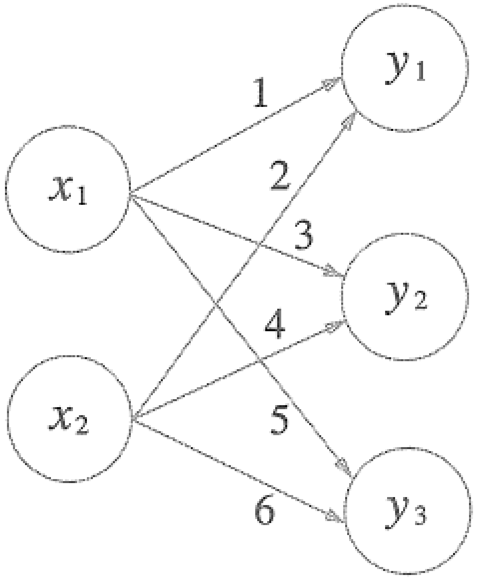
N-dimensional Array

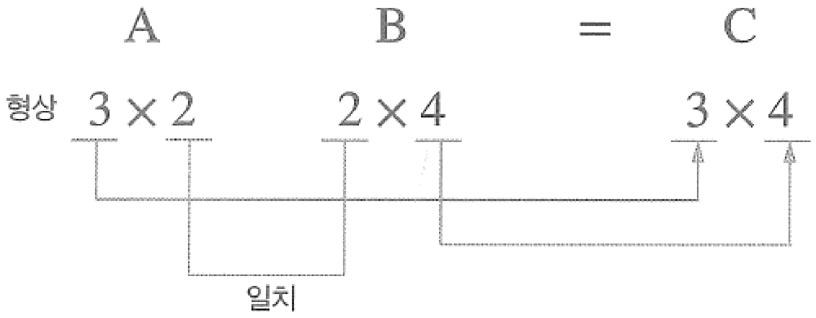
- Dot product

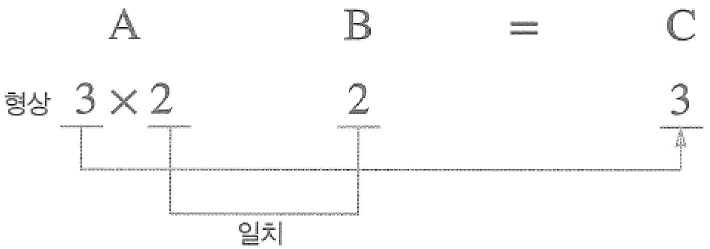
그림 3-11 행렬의 내적 계산 방법

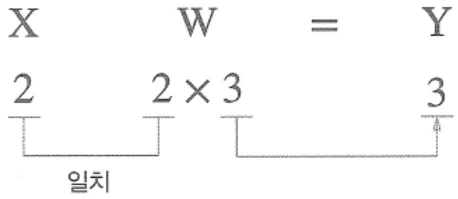



```
>>> A = np.array([[1,2], [3,4]])
>>> A.shape
(2, 2)
>>> B = np.array([[5,6], [7,8]])
>>> B.shape
(2, 2)
>>> np.dot(A, B)
array([[19, 22],
       [43, 50]])
```









Three-layer Neural Network

그림 3-15 3층 신경망 : 입력층(0층)은 2개, 첫 번째 은닉층(1층)은 3개, 두 번째 은닉층(2층)은 2개, 출력층(3층)은 2개의 뉴런으로 구성된다.

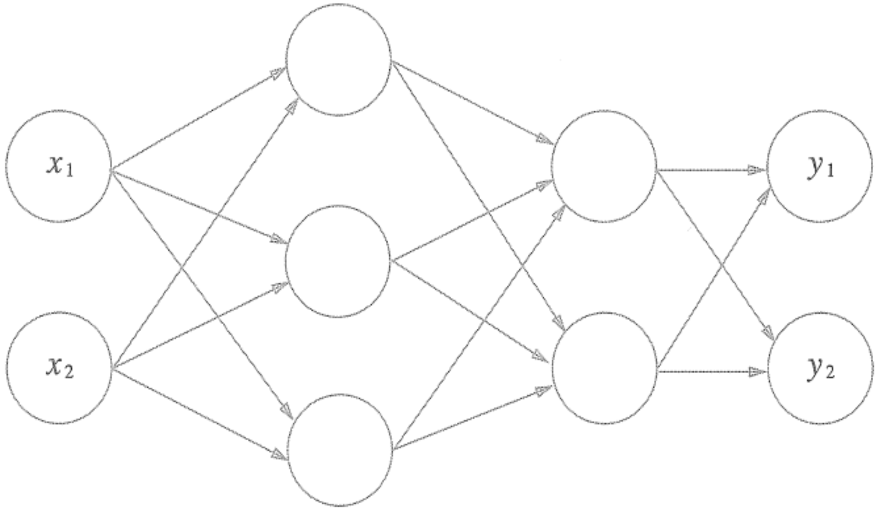
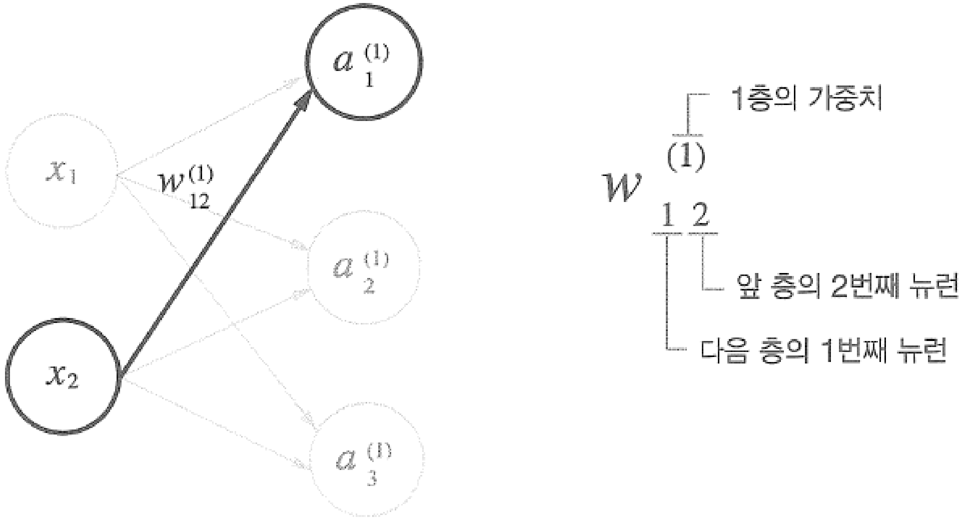
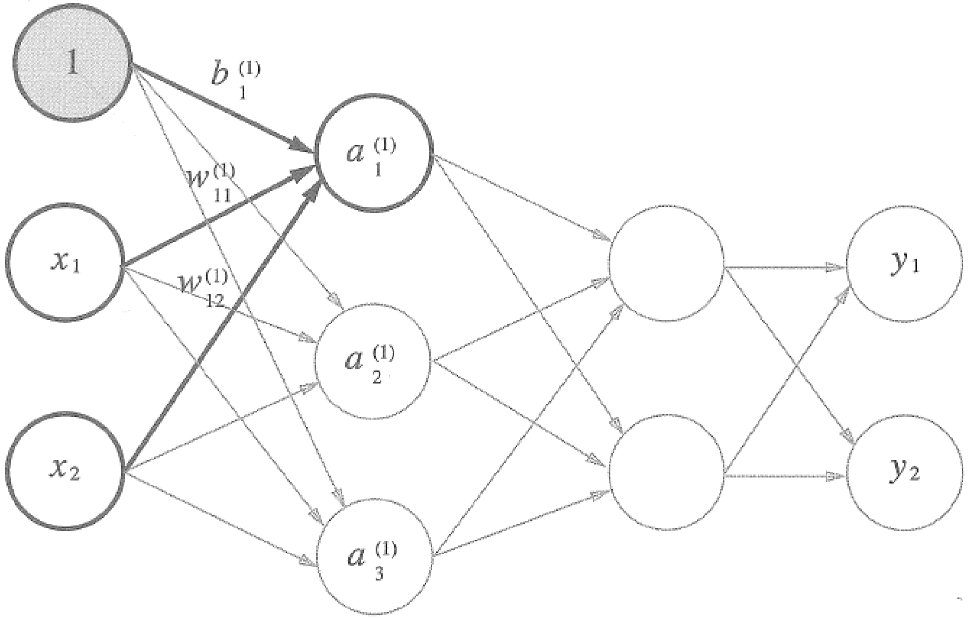


그림 3-16 중요한 표기



Three-layer Neural Network

그림 3-17 입력층에서 1층으로 신호 전달



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

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

$$\mathbf{A}^{(1)} = \begin{pmatrix} a_1^{(1)} & a_2^{(1)} & a_3^{(1)} \end{pmatrix},\ \mathbf{X} = \begin{pmatrix} x_1 & x_2 \end{pmatrix},\ \mathbf{B}^{(1)} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} & b_3^{(1)} \end{pmatrix}$$

$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \end{pmatrix}$$

Three-layer Neural Network

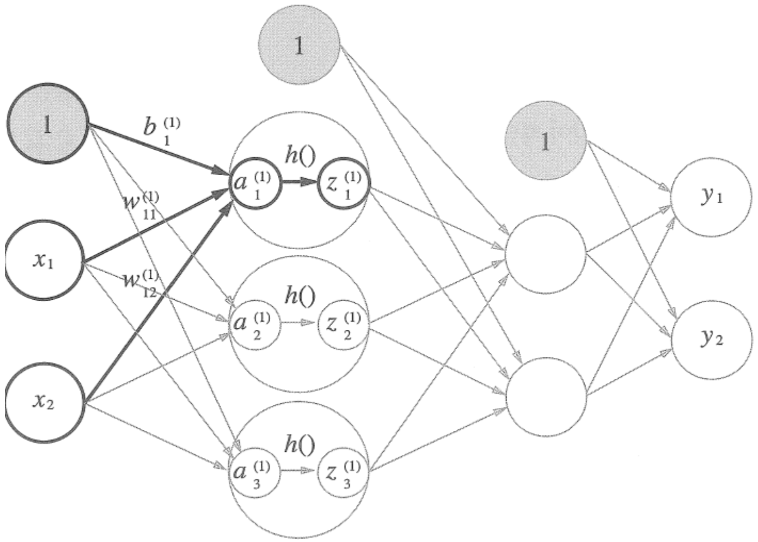


그림 3-18 입력층에서 1층으로의 신호 전달

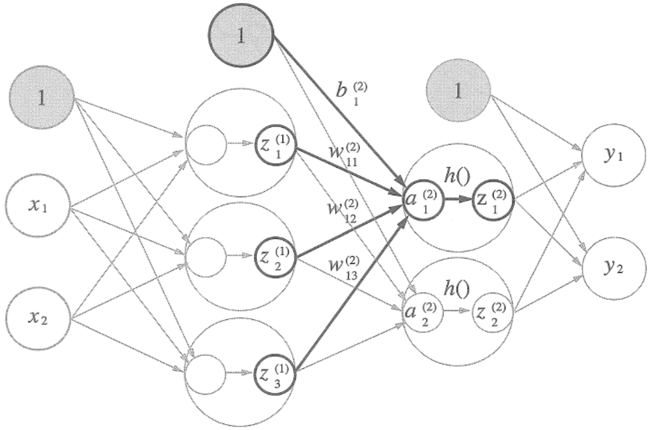


그림 3-19 1층에서 2층으로의 신호 전달

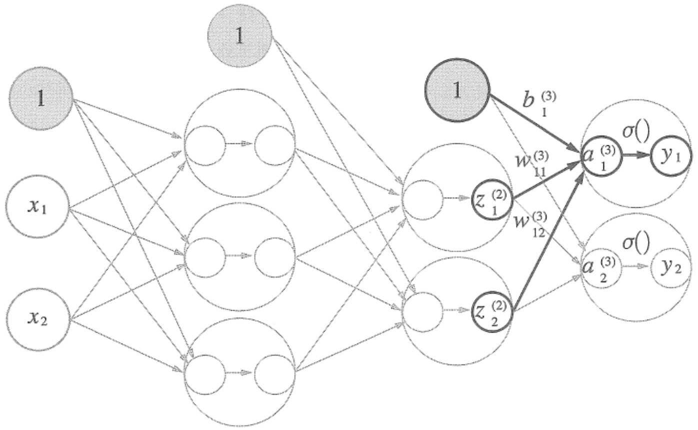


그림 3-20 2층에서 출력층으로의 신호 전달

Three-layer Neural Network

```
def init_network():
    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

def forward(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 = identity_function(a3)

    return y

network = init_network()
x = np.array([1.0, 0.5])
y = forward(network, x)
print(y) # [ 0.31682708  0.69627909]
```

```
def identity_function(x):
    return x
```

Output Layer

- Classification or regression?
- Identity function and softmax function

```
def softmax(a):
    exp_a = np.exp(a)
    sum_exp_a = np.sum(exp_a)
    y = exp_a / sum_exp_a

    return y
```

그림 3-21 항등 함수

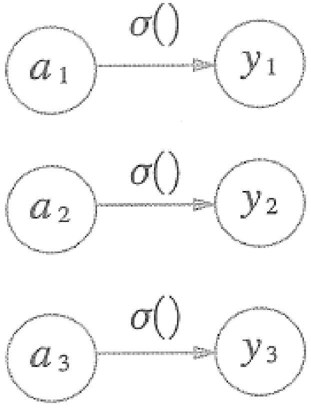
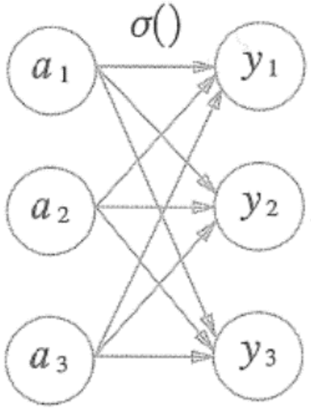


그림 3-22 소프트맥스 함수



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

Softmax Function

- Overflow문제
  - 어떤 정수를 더해 overflow 방지

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

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

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

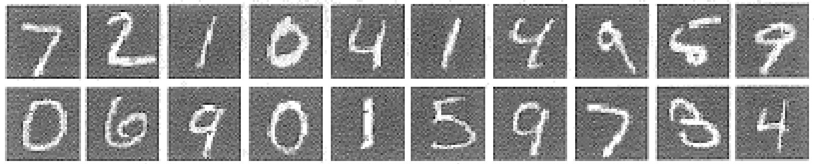
```
>>> a = np.array([1010, 1000, 990])
>>> np.exp(a) / np.sum(np.exp(a)) # 소프트맥스 함수의 계산
array([ nan,  nan,  nan])          # 제대로 계산되지 않는다.
>>>
>>> c = np.max(a)                  # c = 1010 (최댓값)
>>> a - c
array([  0, -10, -20])
>>>
>>> np.exp(a - c) / np.sum(np.exp(a - c))
array([  9.99954600e-01,   4.53978686e-05,   2.06106005e-09])
```

```
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
```

MNIST

- MNIST data set
  - 기계학습의 대표적인 자료
  - 0-9까지의 숫자 이미지로 구성
  - Training 이미지 60,000 장 Test image 10,000장
  - Image data는 28*28크기의 회색 이미지(1채널), 각 픽셀은 0에서 255까지의 값을 취함
  - 각 image에는 ‘7’,’2’, ‘1’과 같은 이미지가 실제로 나타내는 숫자가 레이블로 붙어 있음
  - mnist.py (mnistPy2.py) : mnist 데이터를 가져오기 위한 것

그림 3-24 MNIST 이미지 데이터셋의 예



MNIST

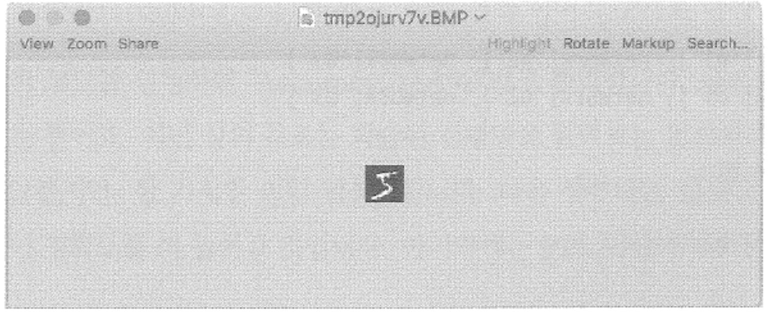
- load_mnist()
  - (훈련 이미지, 훈련, 레이블), (시험이미지, 시험 레이블) 형식
  - Parameter: normalize, flatten, one_hot_label

```
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
from dataset.mnist import load_mnist

# 처음 한 번은 몇 분 정도 걸립니다.
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True,
normalize=False)

# 각 데이터의 형상 출력
print(x_train.shape) # (60000, 784)
print(t_train.shape) # (60000,)
print(x_test.shape)  # (10000, 784)
print(t_test.shape)  # (10000,)
```

MNIST

- 03-mnist_show.py

그림 3-25 MNIST 이미지 중 하나



- 03-neuralnet_mnist.py

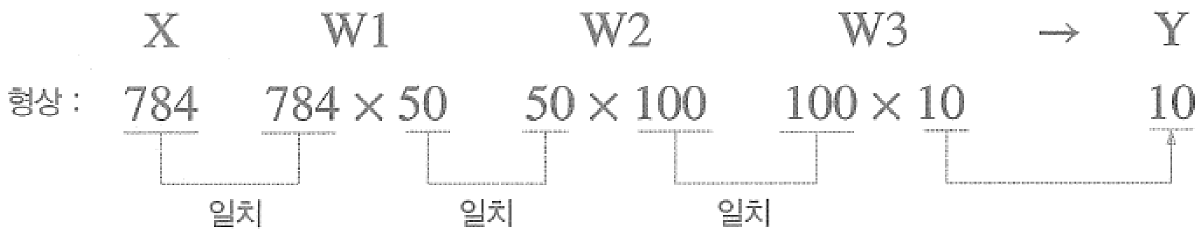
```
import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
from PIL import Image

def img_show(img):
    pil_img = Image.fromarray(np.uint8(img))
    pil_img.show()

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True,
normalize=False)

img = x_train[0]
label = t_train[0]
print(label) # 5

print(img.shape)              # (784,)
img = img.reshape(28, 28)     # 원래 이미지의 모양으로 변형
print(img.shape)              # (28, 28)

img_show(img)
```

MNIST

- 매개변수의 형상

```
>>> x, _ = get_data()
>>> network = init_network()
>>> W1, W2, W3 = network['W1'], network['W2'], network['W3']
>>>
>>> x.shape
(10000, 784)
>>> x[0].shape
(784,)
>>> W1.shape
(784, 50)
>>> W2.shape
(50, 100)
>>> W3.shape
(100, 10)
```

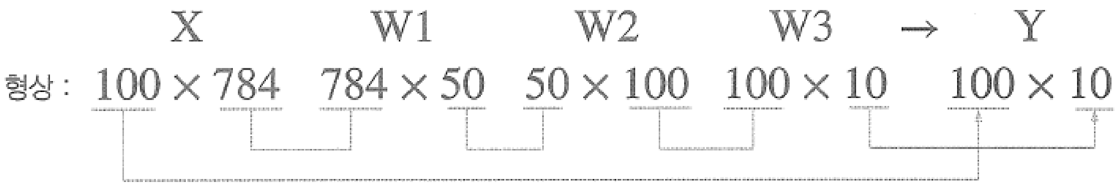
그림 3-26 신경망 각 층의 배열 형상의 추이



MNIST

- Batch 처리

그림 3-27 배치 처리를 위한 배열들의 형상 추이




```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]
    y_batch = predict(network, x_batch)
    p = np.argmax(y_batch, axis=1)
    accuracy_cnt += np.sum(p == t[i:i+batch_size])

print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
```

MNIST

```
>>> list( range(0, 10) )
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list( range(0, 10, 3) )
[0, 3, 6, 9]
```

```
>>> x = np.array([[0.1, 0.8, 0.1], [0.3, 0.1, 0.6],
...     [0.2, 0.5, 0.3], [0.8, 0.1, 0.1]])
>>> y = np.argmax(x, axis=1)
>>> print(y)
[1 2 1 0]
```

```
>>> y = np.array([1, 2, 1, 0])
>>> t = np.array([1, 2, 0, 0])
>>> print(y==t)
[True True False True]
>>> np.sum(y==t)
3
```

Round Up

- 신경망에서는 활성화 함수로 sigmoid함수와 ReLU 함수 같은 매끄럽게 변화하는 함수를 이용
- Numpy의 다차원 배열을 잘 사용하면 신경망을 효율적으로 구현할 수 있다
- 기계학습 문제는 크게 회귀와 분류로 나눌 수 있다
- 출력층의 활성화 함수로는 회귀에서는 주로 항등 함수를, 분류에서는 주로 Softmax함수를 이용한다
- 분류에서는 출력층의 뉴런 수를 분류하려는 클래스 수와 같게 설정한다
- 입력 데이터를 묶은 것을 배치라 하며, 추론 처리를 이 배치 단위로 진행하면 결과를 훨씬 빠르게 얻을 수 있다