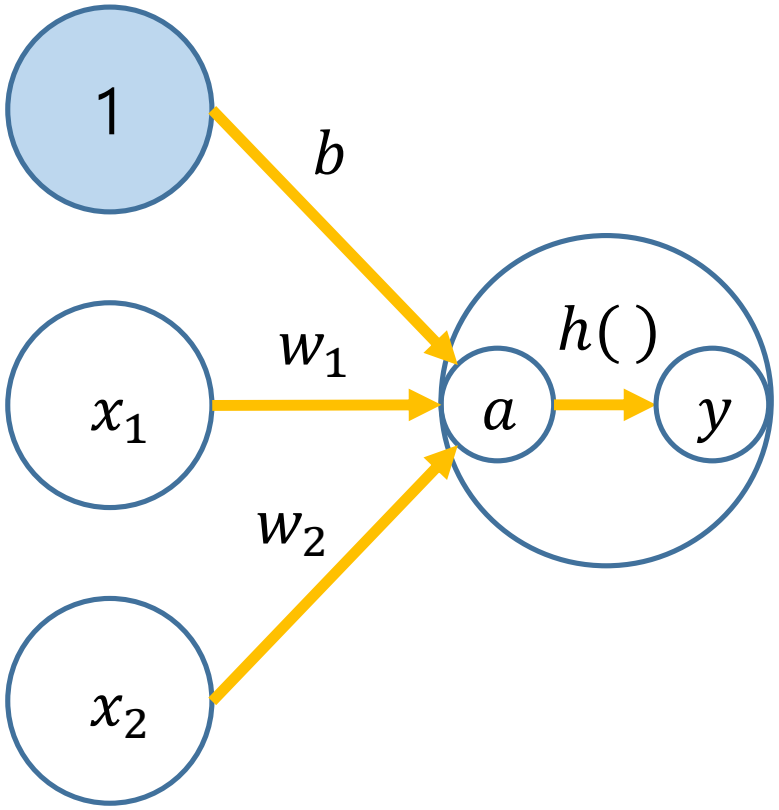
신경망 : Neural Network

[Deep Learning from scratch] - 사이토 고키

중요 내용 요약 : 김진수

신경망 : Neural Network



- 퍼셉트론에서의 y : Step function 값 $(0\ or\ 1)$

$$y = \begin{cases} 0\ (b + w_1x_1 + w_2x_2 < 0) \\ 1\ (b + w_1x_1 + w_2x_2 \geq 0) \end{cases}$$

- 신경망에서의 y : 비선형의 활성화 함수(h) 값

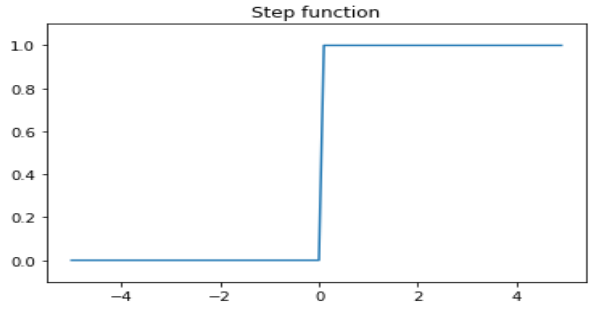
$$a = b + w_1x_1 + w_2x_2$$

$$y = h(a)$$

활성화 함수 : Activation function

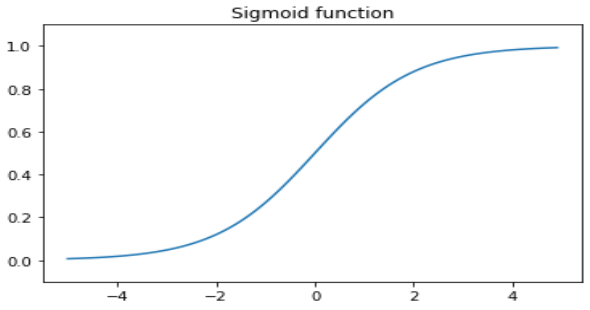
- Step function

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



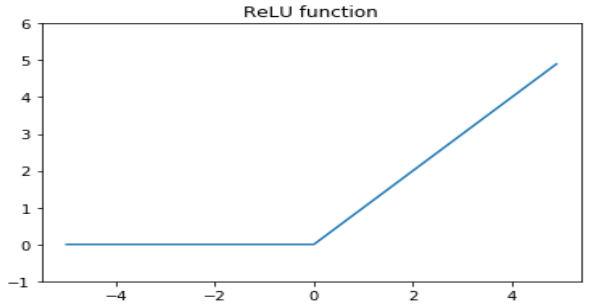
- Sigmoid

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



- ReLU (Rectified Linear Unit)

$$h(x) = \begin{cases} 0\ (x < 0) \\ x\ (x \geq 0) \end{cases}$$



활성화 함수로는 반드시

비선형 함수를 사용한다

→ 선형 함수는 층을 깊게

하더라도 의미가 없기 때문

활성화 함수 : Code

- Step function

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

$x > 0$ 의 True/False를 $(numpy)int$로 변환하여 0 또는 1의 값으로 return

- Sigmoid

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

$sigmoid(x) = \frac{1}{1+\exp(-x)}$ 식의 값을 return

- ReLU (Rectified Linear Unit)

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

0과 x를 비교하여 큰 값을 출력한다.
$x >= 0$일 때는 x를, $x < 0$일 때는 0을 출력

다차원 행렬의 곱

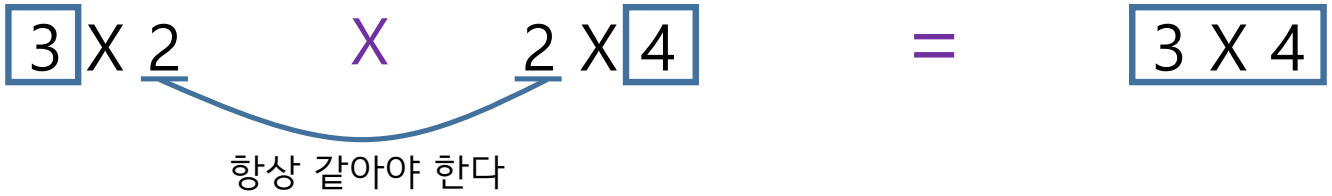
$$\begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \end{pmatrix} \begin{pmatrix} w_1 & w_2 \\ w_3 & w_4 \end{pmatrix} = \begin{pmatrix} x_1w_1 + x_2w_3 & x_1w_2 + x_2w_4 \\ x_3w_1 + x_4w_3 & x_3w_2 + x_4w_4 \end{pmatrix}$$

X W Y

- $(np.array$이라면$)\ X, W$ 의 행렬 곱 $Y = np.dot(X, W)$ (신경망에서는+Bias)

  ⇒ dot는 '점곱', '내적', 또는 '스칼라곱'이라고도 한다.

- X.shape : (3,2), W.shape : (2,4)일 때, Y.shape : (3,4)



출력층 설계하기

- 분류 (Classification) : 입력 데이터가 **어느 Class**에 속하는 지에 대한 문제

  ex) 인물 사진 -> 남?/녀?

- 회귀 (Regression) : 입력 데이터에서 **연속적인 수치**를 예측하는 문제

  ex) 인물 사진 -> 57.4kg?

- 출력층에서 사용하는 활성화 함수
  - 항등 함수 : 입력을 그대로 출력 (Identity function)
  - Softmax 함수 : 지수 함수를 활용한 분류기. 각 요소의 값은 0~1, 합은 1이 된다.

$$y_k = \frac{exp(a_k)}{\sum_{i=1}^{n} exp(a_i)} = \frac{\exp(a_k - C)}{\sum_{i=1}^{n} \exp(a_i - C)}$$ (임의의 정수 C는 지수 오버플로 방지 대책, 입력 최댓값 사용이 일반적)

Softmax 함수 : Code

```
def softmax(x):
    c = np.max(x)
    exp_x = np.exp(x - c) # overflow prevention
    sum_exp_x = np.sum(exp_x)
    return exp_x / sum_exp_x
```

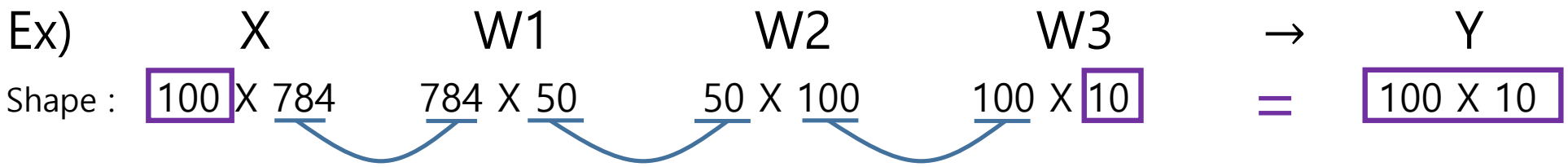
- 지수 함수(exp)를 사용하기 때문에 오버플로 (inf 값 발생)가 날 수 있다.

  => x중에서의 $np.max$값 c를 빼서 값이 무수히 커지는 것을 막아준다.

- $softmax(x) = \frac{\exp(x_k - C)}{\sum_{i=1}^{n} \exp(x_i - C)}$의 값을 return

배치 처리

- 수치 계산 라이브러리가 큰 배열을 효율적으로 처리할 수 있으므로, 배치 처리를 통해 시간을 대폭 줄여줄 수 있다.
- 데이터의 I/O 횟수가 줄어, 병목 현상을 막아주게 되므로, 순수 계산을 수행하는 비율이 높아진다.
- 결론적으로 큰 배열을 한번에 계산 하는 것이 작은 배열을 여러 번 계산하는 것보다 빠르다.
- Ex)



신경망의 추론(Forward) : Code

```python
def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=True, one_hot_label=False)
    return x_test, t_test
```

- $load\_mnist$: Mnist data를 load하는 함수, flatten (28*28 ⇒ 784), normalize (/255)

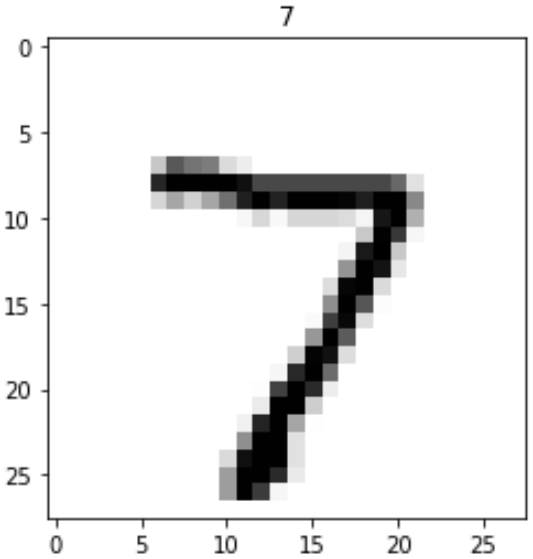
```python
def init_network():
    with open('sample_weight.pkl', 'rb') as f:
        network = pickle.load(f)

    return network
```

- 학습된 신경망의 W(가중치)와 b(편향) 값을 $pickle$ 파일로부터 불러오는 함수

```python
x, t = get_data() # mnist data load
network = init_network() # saved weight load

plt.imshow(x[0].reshape(28, 28), cmap='Greys')
plt.title('%d' % t[0])
plt.show()
```



- Data 와 신경망을 load 시킨 후에, 첫 Data인 image[0]를 plot한 결과

```python
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
```

- x(입력), W(가중치), b(편향)을 가지는 신경망을 이용하여 행렬 계산을 수행하고, 활성화 함수로 $sigmoid$ 함수를 사용
- 출력층에서 $softmax$의 결과를 return 해준다.

신경망의 추론(Forward) : Code

```
accuracy_cnt = 0
for i in range(len(x)):
    y = predict(network, x[i])
    p = np.argmax(y) # index of max value
    if p == t[i]:
        accuracy_cnt += 1

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

```
Accuracy:0.9352
```

- x(입력)을 1개씩 신경망으로 추론시켰을 때의 결과 y
- y의 결과에서 가장 큰 값의 index : p

  ⇒ p가 정답(t)과 같다면($True(1)/False(0)$), 누적하여 전체 개수로 나누어 준 정확도를 출력

```
# batch processing
batch_size = 100
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)))
```

```
Accuracy:0.9352
```

- x_batch(100개)를 신경망으로 추론시켰을 때의 결과 y_batch(100개)
- y_batch(100개)의 결과에서 가장 큰 값의 index : p(100개)

  ⇒ $argmax$ 시에 100개의 index를 내주기 위해서 $axis = 1$을 기준으로, 각각의 p가 정답(t)과 같다면($True(1)/False(0)$), 누적하여 전체 개수로 나누어 준 정확도를 출력, 결과는 배치 처리 전과 같다.