# Deep Learning with Python

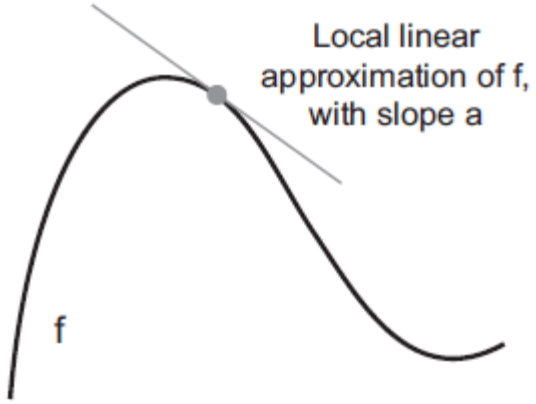
Ch2.4~6 The mathematical building blocks of neural networks

G201849026 신승엽

# 목차

- Gradient-based optimization
  - Derivative, Gradient, SGD, Backpropagation
- Looking back at our first example
  - Python codes

# Derivative




- derivative란?
  - continuous, smooth function f(x) = y
  - continuous : $f(x + \varepsilon_x) = y + \varepsilon_y$
    -> x에서 작은 변화량은 y에서도 작은 변화량
  - smooth : $f(x + \varepsilon_x) = y + a * \varepsilon_x$
    -> 충분히 작은 $\varepsilon_x$에 대해 f는 기울기 a 인 선형함수로 근사한다.
  - The slope a is the derivative of f in p
    -> f(x)값을 줄이기 위해서는 a의 반대방향으로 x를 업데이트

# Gradients

- gradients란?
  - Gradient는 tensor에 대한 derivative이다.
  - Derivative가 기울기라면 Gradient는 곡률로 이해할 수 있다.
  - w가 tensor인 f(w)=y, y를 작아지게 하는 w업데이트는 다음과 같다.
    -> $w_1 = w_0$ - step * gradient(f)($w_0$)
  - step(learning rate)이 필요한 이유는 gradient가 $w_0$의 작은 변화량에서의 근사이기 때문에 $w_0$로부터 멀리 떨어지게 하는 것을 막기 위함.

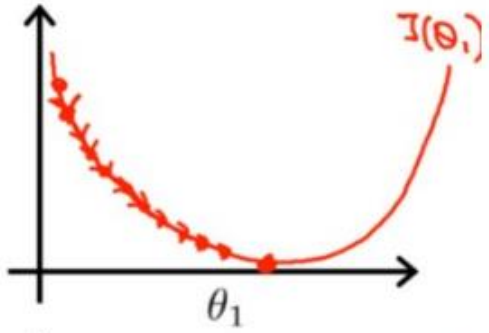
# SGD

- Stochastic Gradients Descent로 임의로 데이터 샘플을 가져와 학습에 사용하는 기법
- 등장배경: 데이터 수가 많은 경우 손실함수의 최소값을 찾는데 오래 걸림.
- 가져오는 데이터 샘플 수에 따라 trueSGD, mini-batchSGD, batchSGD로 나뉜다.
  - trueSGD: 하나의 데이터 샘플
  - mini-batchSGD : 일정량(mini-batch)의 샘플
  - batchSGD : 데이터셋의 모든 샘플(shuffle)

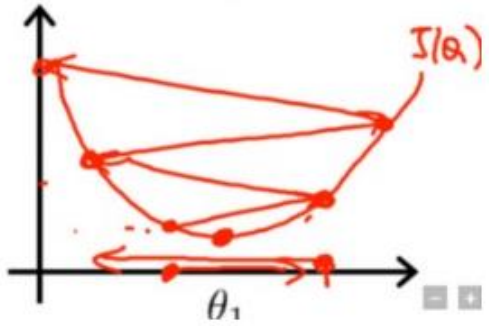
# Optimizer

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.




If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.




- Learning rate(step)의 중요성 부각
- 기존 SGD를 수정하여 learning rate 문제를 해결하는 기법
- Momentum, Adagrad, RMSProp 등이 있음

# Momentum

- 물리학의 관성의 성질을 이용
- 이전의 gradient가 충분히 크면 지역 극소점을 지나칠 것이라는 아이디어

```
past_velocity = 0.
momentum = 0.1
while loss > 0.01:
    w, loss, gradient = get_current_parameters()
    velocity = past_velocity * momentum + learning_rate * gradient
    w = w + momentum * velocity - learning_rate * gradient
    past_velocity = velocity
    update_parameter(w)
```

Constant momentum factor

Optimization loop

# Adagrad(adaptive gradient)

- 각각의 가중치($\theta$)마다 learning rate를 다르게 주는 방식
- 이전까지 많이 변화한 $\theta$는 적게 변화하도록 하고 이전까지 적게 변화한 w는 많이 변화하도록 함.
- 학습이 계속될수록 G값은 계속 커지므로 업데이트가 느려지는 단점

$$G_t = G_{t-1} + (\nabla_\theta J(\theta_t))^2$$

gradient

Learning rate

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

분모가 0이 안되게

# RMSProp

- Adagrad 단점을 보완
- G를 지수이동평균으로 설정.
- $\gamma$는 보통 0.99나 09로 설정

$$G = \gamma G + (1 - \gamma)(\nabla_\theta J(\theta_t))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{G + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

# Backpropagation

- 뉴럴넷의 층이 쌓이면 다음과 같은 수식을 갖는다
  ->f(W1, W2, W3) = a(W1, b(W2, c(W3)))

- 연쇄법칙을 사용해 뒤에서부터 손실 값을 전파하여 w를 업데이트한다.
  ->f(g(x)) = f'(g(x)) * g'(x)

## Looking back at our first example

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255


network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))


network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

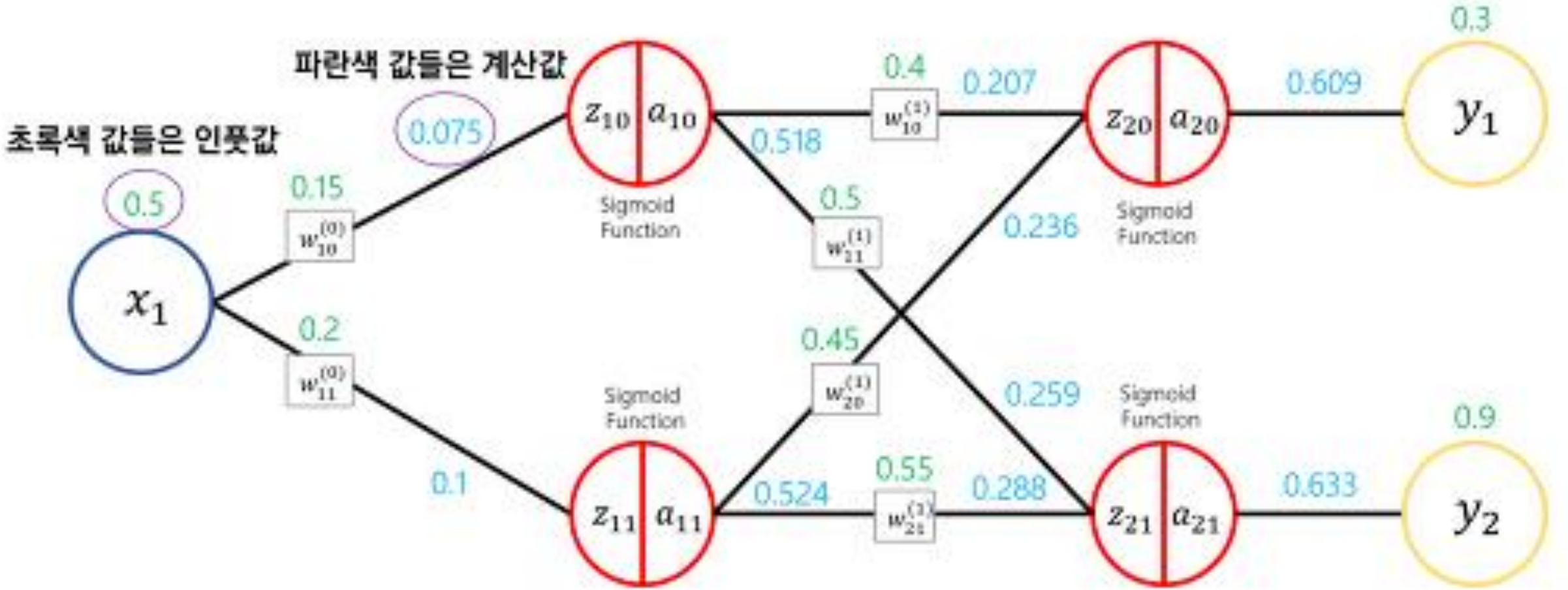
데이터 로드

Reshaping and scaling

2개의 층으로 구성되며 가중치는 (28*28,512),(512,10)이며 활성화 함수로 relu와 softmax를 적용.

Optimizer로 RMSProp 알고리즘, 손실함수로 크로스엔트로피, 학습의 성능을 파악하기 위해 accuracy를 적용.

fit함수를 call하여 학습을 실행하고 128개의 배치데이터 마다 가중치 갱신(60000/128=469번)을 하며 총 5번 반복 학습한다.




$$E_{tot} = \frac{1}{2}\sum(target - output)^2 = \frac{1}{2}(0.3 - 0.609)^2 + \frac{1}{2}(0.9 - 0.621)^2 = 0.087$$

$$\frac{\partial E_{tot}}{\partial w_{10}^{(1)}} = \frac{\partial E_{tot}}{\partial a_{20}} \frac{\partial a_{20}}{\partial z_{20}} \frac{\partial z_{20}}{\partial w_{10}^{(1)}}$$

$$= -\left(target_{y1} - a_{20}\right) * sigmoid(z_{20}) * \left(1 - sigmoid(z_{20})\right) * a_{10}$$

$$= -(0.3 - 0.609) * 0.609 * (1 - 0.609) * 0.518$$

$$= 0.0381$$

$$w_{10}^{(1)+} = w - \eta * \frac{\partial E_{tot}}{\partial w_{10}^{(1)}} = 0.4 - 0.5 * 0.0381 = 0.380$$

$w_{10}^{(1)}$**이 전체 에러에 미치는 영향은** 0.0381, **갱신한** $w_{10}^{(1)+}$**값은** 0.380

$$\frac{\partial E_{tot}}{\partial w_{10}^{(0)}} = \left(\frac{\partial E_1}{\partial a_{10}} + \frac{\partial E_2}{\partial a_{10}}\right) \frac{\partial a_{10}}{\partial z_{10}} \frac{\partial z_{10}}{\partial w_{10}^{(0)}}$$

$$\frac{\partial E_1}{\partial a_{10}} = \frac{\partial E_1}{\partial a_{20}} \frac{\partial a_{20}}{\partial z_{20}} \frac{\partial z_{20}}{\partial a_{10}}$$

$$= -\left(target_{y1} - a_{20}\right) * sigmoid(z_{20}) * \left(1 - sigmoid(z_{20})\right) * w_{10}^{(1)}$$

$$= -(0.3 - 0.609) * 0.609 * (1 - 0.609) * 0.4$$

$$= 0.0294$$

# $a_{10}$ 가 $E_1$ 에 미치는 영향은 0.0294

$$\frac{\partial E_{tot}}{\partial w_{10}^{(0)}} = \left(\frac{\partial E_1}{\partial a_{10}} + \frac{\partial E_2}{\partial a_{10}}\right) \frac{\partial a_{10}}{\partial z_{10}} \frac{\partial z_{10}}{\partial w_{10}^{(0)}}$$

$$\frac{\partial E_2}{\partial a_{10}} = \frac{\partial E_2}{\partial a_{21}} \frac{\partial a_{21}}{\partial z_{21}} \frac{\partial z_{21}}{\partial a_{10}}$$

$$= -\left(target_{y2} - a_{21}\right) * sigmoid(z_{21}) * \left(1 - sigmoid(z_{21})\right) * w_{11}^{(1)}$$

$$= -(0.9 - 0.633) * 0.633 * (1 - 0.633) * 0.5$$

$$= -0.0328$$

# $a_{10}$ 가 $E_2$ 에 미치는 영향은 -0.0328

$$\frac{\partial E_1}{\partial a_{10}} = 0.0305 \qquad \frac{\partial E_2}{\partial a_{10}} = -0.0328 \qquad \frac{\partial a_{10}}{\partial z_{10}} = sigmoid(z_{10}) * \left(1 - sigmoid(z_{10})\right)$$

---

$$\frac{\partial E_{tot}}{\partial w_{10}^{(0)}} = \left(\frac{\partial E_1}{\partial a_{10}} + \frac{\partial E_2}{\partial a_{10}}\right) \frac{\partial a_{10}}{\partial z_{10}} \frac{\partial z_{10}}{\partial w_{10}^{(0)}}$$

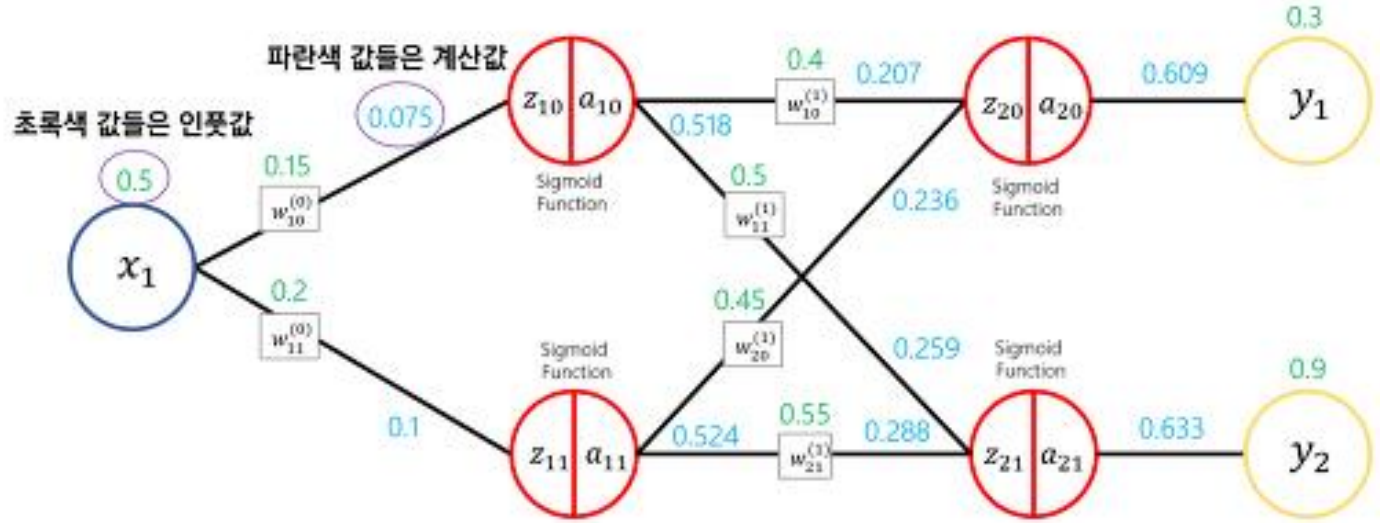
$$= (0.0294 - 0.0328) * 0.249 * 0.5$$

$$= -0.00042$$ # $w_{10}^{(0)}$가 $E_{tot}$ 에 미치는 영향은 -0.00042

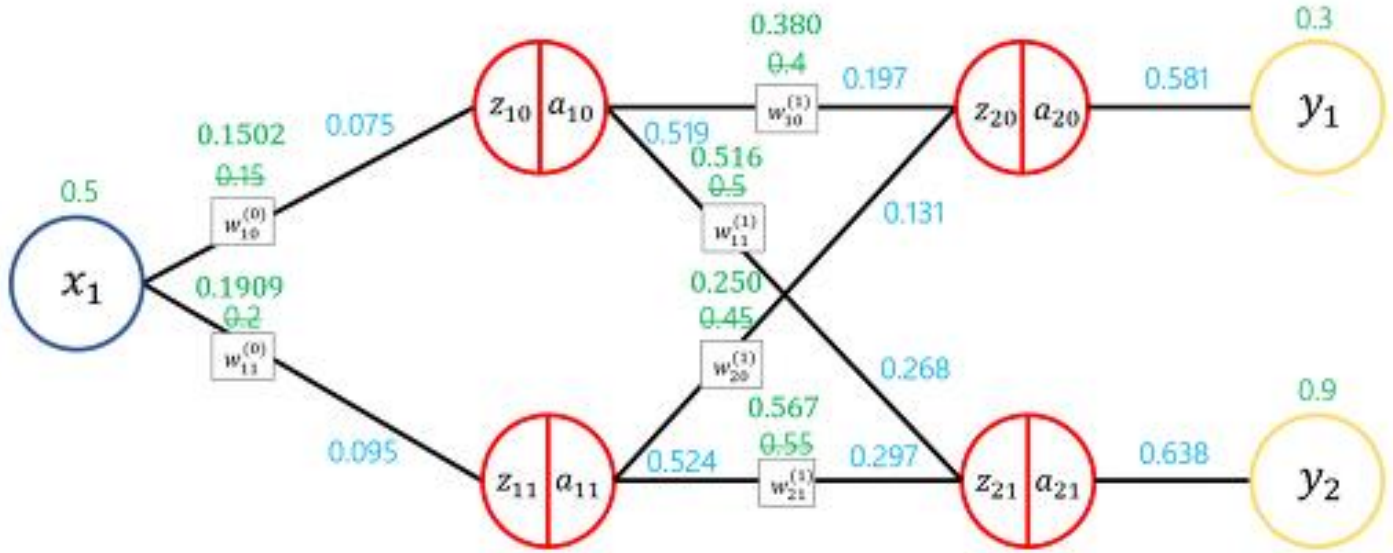
---

$$w_{10}^{(0)+} = w - \eta * \frac{\partial E_{tot}}{\partial w_{10}^{(0)}} = 0.15 - 0.5 * (-0.00042) = 0.1502$$

새로운 $w_{10}^{(0)+}$ 의 값은 0.1502




학습 전 신경망 - y_1 : 0.609, y_2 : 0.633




학습 후 신경망 - y_1 : 0.581, y_2 : 0.638