

AI and Deep Learning

선형회귀(2)

오류 계산 그래프에서의 역전파

제주대학교

변영철

<http://github.com/yungbyun/mllecture>

학습 방법(w 업데이트)

- w 값 난수 초기화 (ex, 4)
- w 에서의 기울기 구함.
- 기울기로 w 를 업데이트

$$w = w - \alpha * (\text{기울기})$$

α : 반영 비율 (learning rate)

TensorFlow



- Tensor : '잡아당기다'라는 뜻인 라틴어 'tensus'
- 무언가를 잡아당기면 그 주위에 굉장히 복잡한 변형이 일어나는데, 이를 기술하는 수학적 언어가 Tensor
- 오류를 줄이기 위해 데이터에 굉장히 복잡한 변형이 일어나며 이를 텐서라고 부름.
- Tensor는 **계산그래프**에서 전달되는 데이터(벡터, 행렬 등)를 포함
- 따라서 TensorFlow는 텐서(데이터)의 흐름(flow)하며, 특히 오류 계산 그래프 내의 텐서 흐름을 통해 가중치 **w** 파라미터를 튜닝하는(학습) 머신러닝 프레임워크

TF를 이용한 선형 회귀 학습

③

`w = tf.Variable(tf.random_normal([1]))`

x

x

w

①

`x = [1]`

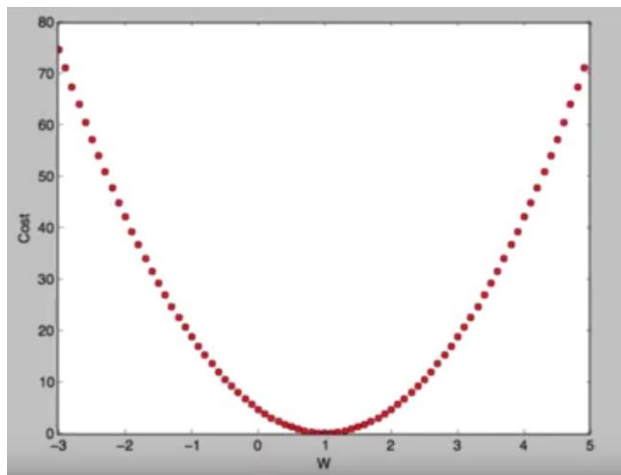
`hypo = w * x`

h

④

`y = [1]`

②



`cost_function = (hypo - y) ** 2`

⑤

$$E = (\text{hypo} - y)^2$$

myml.git 다운로드

- 1) 명령 프롬프트 실행
- 2) 원하는 폴더로 이동
- 3) `git clone https://github.com/yungbyun/myml.git`
- 4) PyCharm으로 오픈 (File | Open...)

(문제)

1시간 일할 경우 1달러를, 2시간 일할 경우 2달러를, 일한 시간만큼 시급을 지급한다고 하자. 이때 일한 시간을 줄 경우 시급을 알아 맞추는 프로그램을 작성하시오.

01.py


Finding w in
linear regression

```
import tensorflow as tf
```

```
#----- 학습데이터 설정  
x_data = [1]  
y_data = [1]
```

```
#----- 신경세포 만들기  
w = tf.Variable(tf.random_normal([1]))  
hypo = w * x_data
```

오류를 최소화하도록
w를 업데이트하는 train
오퍼레이션 반환



```
#----- 학습 시키기  
cost = (hypo - y_data) ** 2  
  
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)  
  
sess = tf.Session()  
sess.run(tf.global_variables_initializer())  
  
for i in range(1001):  
    sess.run(train) # 1-run, 1-update of w -> 1001 updates  
  
    if i % 100 == 0:  
        print('w:', sess.run(w), 'cost:', sess.run(cost))
```

```
#----- 학습 후 테스트하기  
x_data = [2]  
print(sess.run(x_data * w))
```



```
import tensorflow as tf
```

```
#----- 학습 데이터 설정
```

```
x_data = [1]
```

```
y_data = [1]
```

- `import` : 라이브러리 모듈을 찾아 필요할 경우 초기화하고 해당 코드에서 어떤 이름으로 사용되는지를 지정함.
- `x_data`: 여러 입력 데이터 보관하는 리스트(list)
- `y_data`는 여러 정답들을 보관하는 리스트
- `xxx = [1, 'hello', 2, 'world']`
- `print (xxx)`
- `xxx[1] = 'hi'`
- `xxx.append('jeju')`
- `print (xxx)`

#----- 신경세포 만들기

w = tf.Variable(tf.random_normal([1]))

hypo = w * x_data

- random_normal: 난수 텐서를 만들어 반환하는 함수
- Variable: 텐서를 담는 그릇인 변수를 만듦.
- w: 텐서를 담는 변수로서 계산 그래프를 구성하는 한 요소
- hypo: 입력데이터(x_data)와 w에 있는 텐서를 곱한 것.
텐서에 무엇인가를 곱한 이것도 텐서

#----- 학습 시/키/기

cost = (hypo - y_data) ** 2

**train = tf.train.GradientDescentOptimizer
(learning_rate=0.01).minimize(cost)**

- cost: 뉴런의 대답과 실제 정답 간의 차이를 제공한 것으로 오류값을 계산하기 위한 텐서를 계산하는 오퍼레이션
- 계산 그래프로 구성됨.
- GradientDescentOptimizer.minimize 함수는 경사하강 법으로 오류값 cost를 최소화하는 오퍼레이션을 리턴하는 함수
- train: 오류값 cost를 최소화하는 오퍼레이션으로 계산 그래프를 이용하여 경사하강법으로 w를 업데이트함(학습)

```
sess = tf.Session()  
sess.run(tf.global_variables_initializer())
```

- Session: 앞서 텐서를 구하기 위해 구성된 계산 그래프를 실행하여 실제 값을 구하기 위한 클래스
- tf.global_variables_initializer 함수: 텐서값을 갖는 변수들을 초기화하는 오퍼레이션을 반환함.
- sess.run 함수: 계산 그래프, 오퍼레이션을 실행하는 함수

```
for i in range(1001):  
    sess.run(train)  
    if i % 100 == 0:  
        print(sess.run(w), sess.run(cost))
```

- for loop: 파이썬 for loop로서 i를 0에서 1000까지 (1001번) 바꾸면서 반복문을 수행함.
- sess.run(train): 경사하강법을 이용하여 학습을 한번 실행함(w값이 한번 업데이트됨).
- if i % 100 == 0: i가 0, 100, 200, 300 등과 같이 나머지가 0일 때 참이되어 print 문장이 실행됨.

#----- 학습 후 테스트하기

x_data = [2]

print(sess.run(x_data * w))

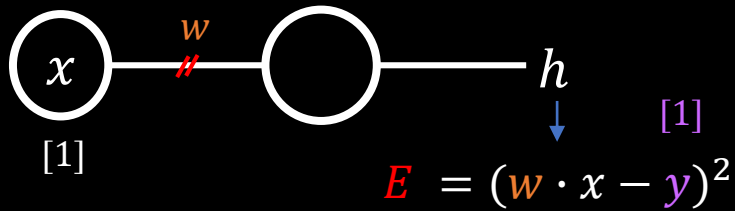
- 학습이 끝났을 때 결과는 무엇? 바로 여러 번 반복해서 업데이트된 w 값임.
- 학습된 w 값과 우리가 원하는 입력합을 곱할 경우 반환되는 값이 예측값임.

#----- 학습 후 테스트하기

x_data = [2]

print(sess.run(hypo))

- 계산 그래프가 처음 만들어질 때 한번만 값들이 설정되어 x_data에 새로운 값이 할당되어도 반영되지 않음.
- 이전에 할당한 x_data가 여전히 사용됨.
- 이를 해결하기 위해서는 처음부터 계산 그래프에 데이터를 설정하는 것이 아니라 필요한 시점에서 데이터를 전달하는 방법이 요구됨.
- 이를 위한 것이 Placeholder 방법임.



(1) (w)

(1 · w)

(1 · w - 1)²

#----- training data

$x_data = [1]$

$y_data = [1]$

#----- a neuron

$w = \text{tf.Variable}(\text{tf.random_normal}([1]))$

$\text{hypo} = w * x_data$

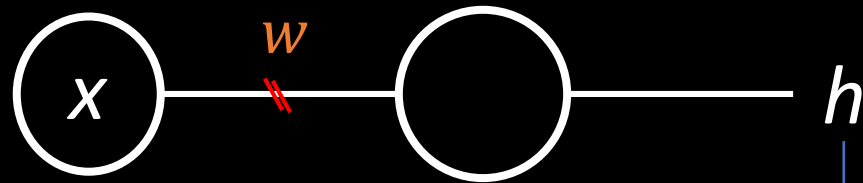
02.py

Drawing
cost function

오류 함수 생각하기

$$E = (wx - y)^2$$

- 어느 부분이 뉴런인가?
- 시냅스는?
- 입력 데이터는?
- 뉴런의 출력
- 정답(answer, ground truth)은?
- 가설(hypothesis)은?
- 오류 함수의 의미는?
- 뉴런 입력이 여러 개일 경우
- 데이터로 주어지는 것은?
- 우리가 튜닝해야 하는 것은?



$$E = (h - y)^2$$

$$E = (w \cdot x - y)^2$$

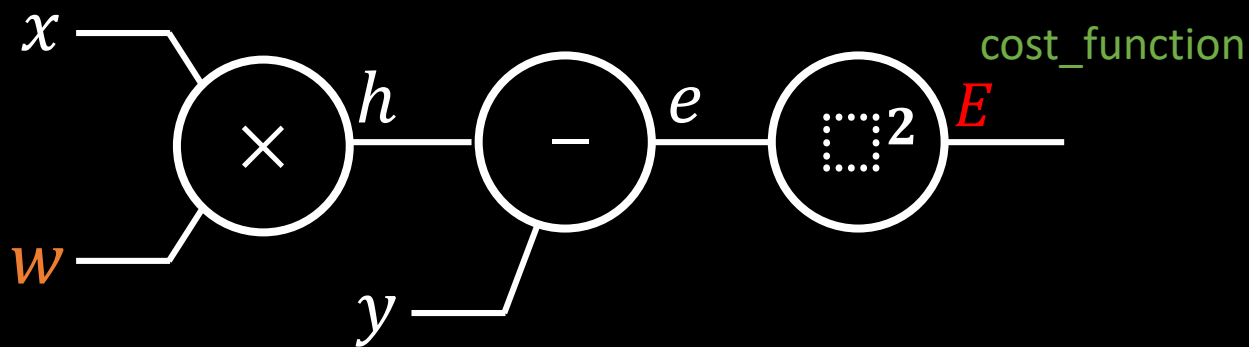
$$E = (w \cdot 1 - 1)^2$$

오류 계산 그래프

$$E = (wx - y)^2$$

hypo = $w * x$

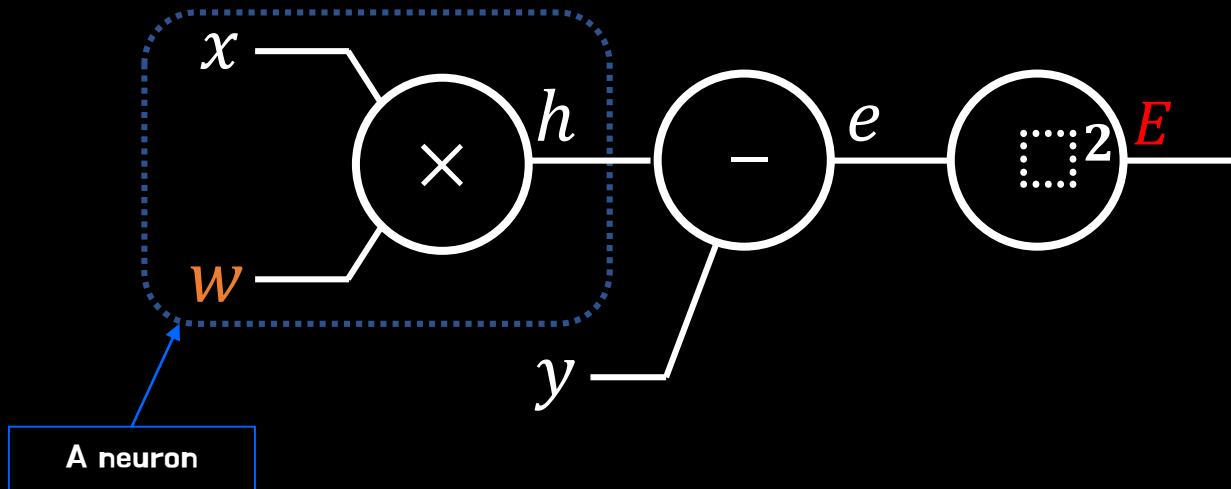
cost_function(E) = (hypo - y) ** 2)



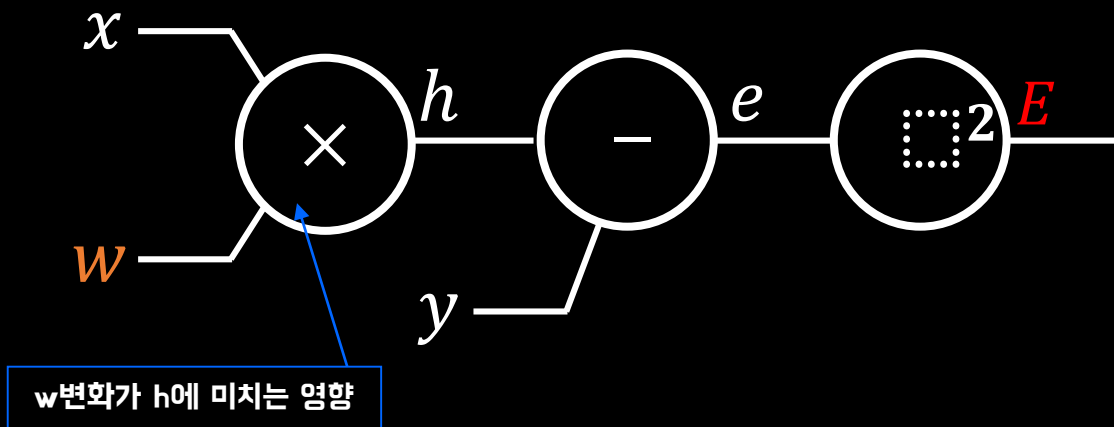
텐서란 무엇이고, 텐서 플로우란 무엇인가?
텐서플로우 프레임워크가 파라미터 튜닝

오류 계산 그래프와 미치는 영향

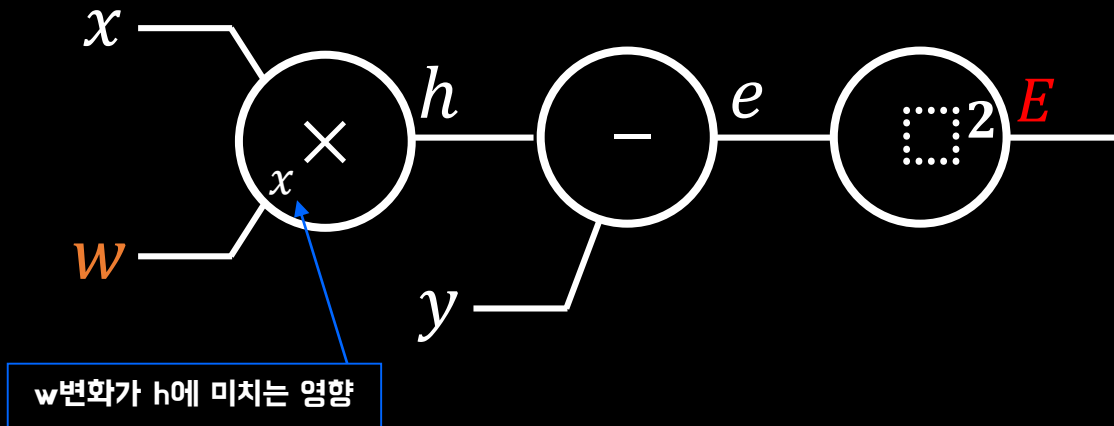
계산 그래프에서 w 가 E에 미치는 영향을 쉽게 알 수 있다.
그러면 오류를 줄일 수 있도록 w 를 조절할 수 있다.



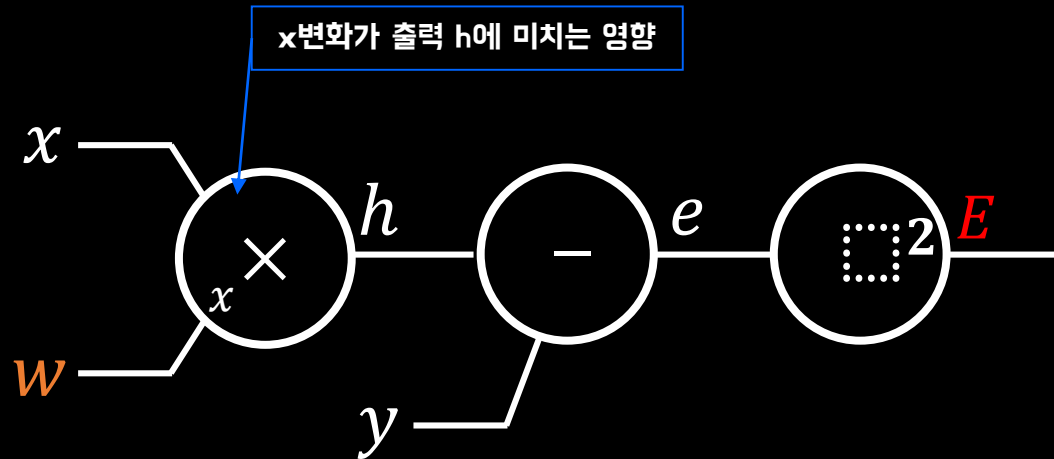
오류 계산 그래프와 미치는 영향



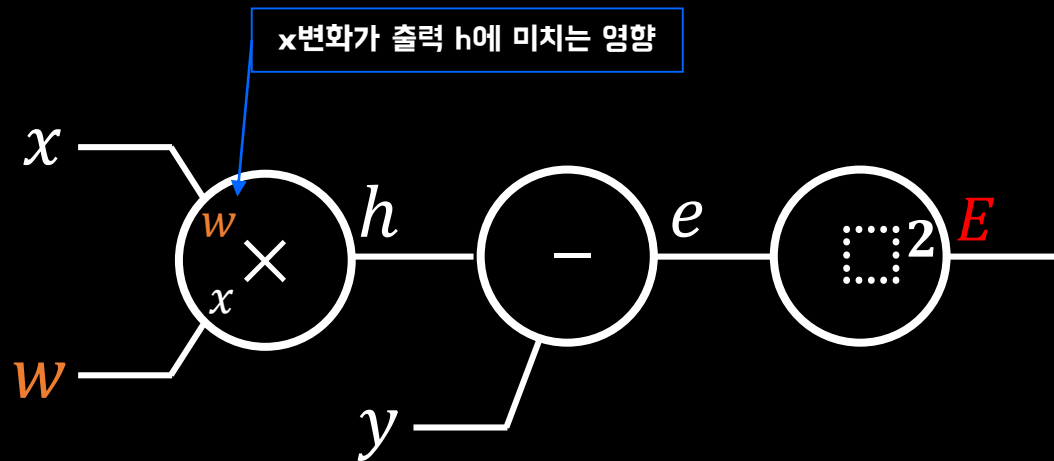
오류 계산 그래프와 미치는 영향



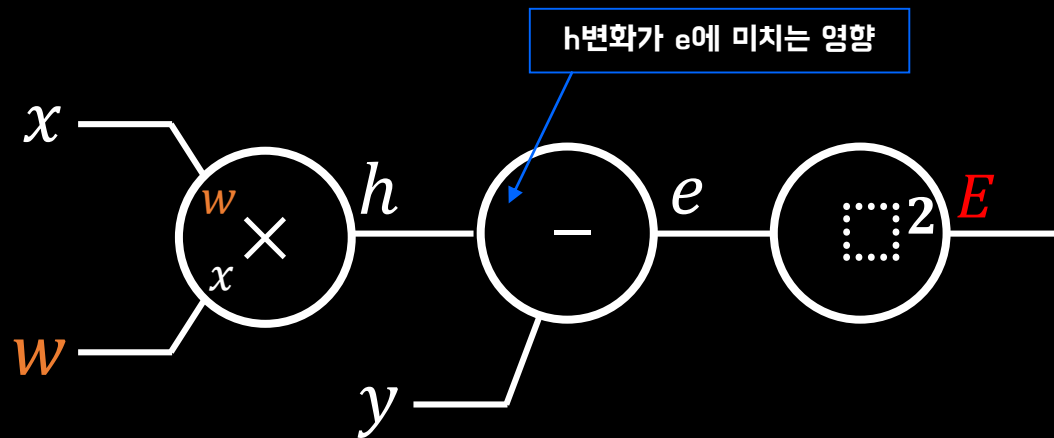
오류 계산 그래프와 미치는 영향



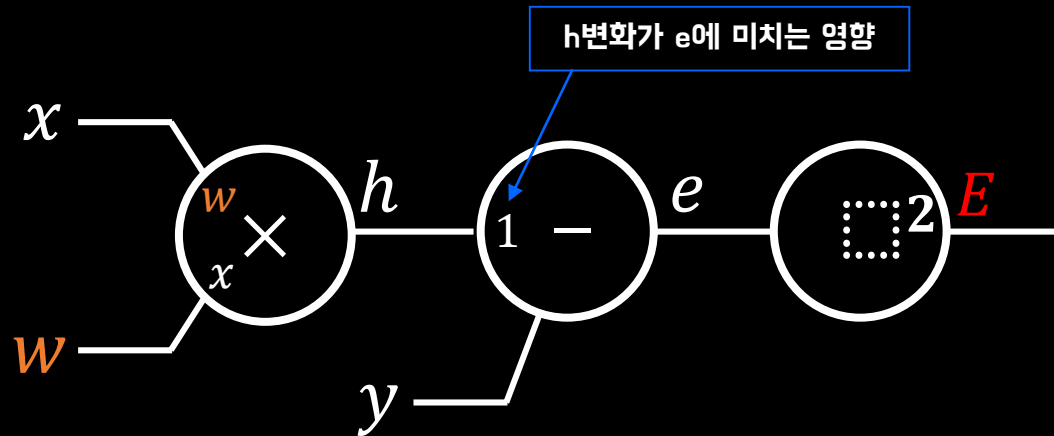
오류 계산 그래프와 미치는 영향



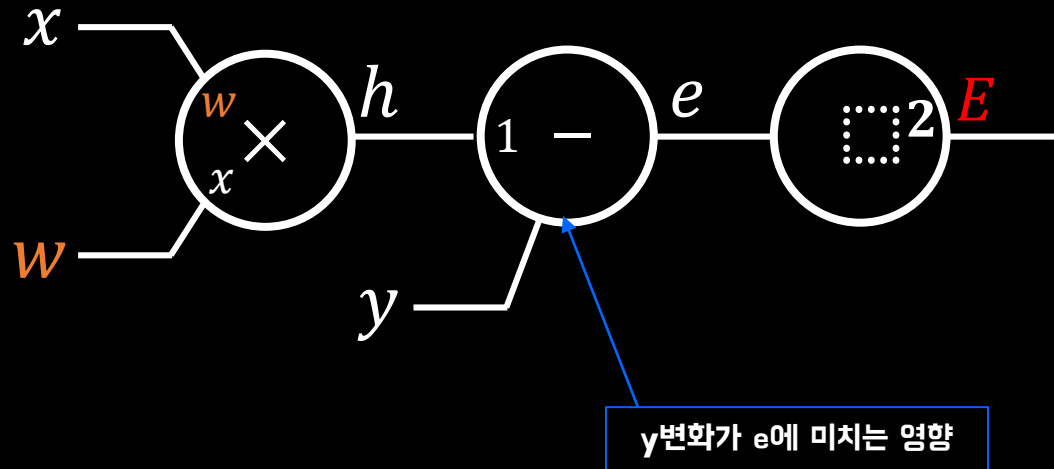
오류 계산 그래프와 미치는 영향



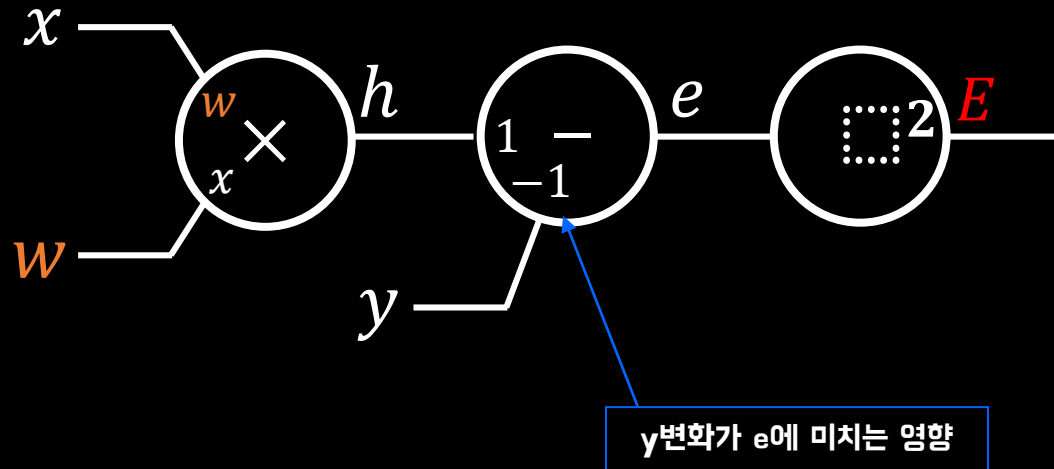
오류 계산 그래프와 미치는 영향



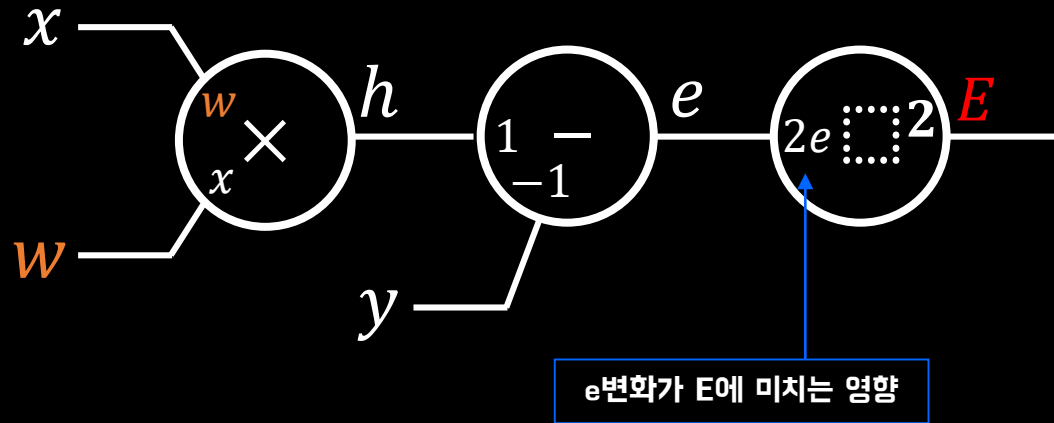
오류 계산 그래프와 미치는 영향



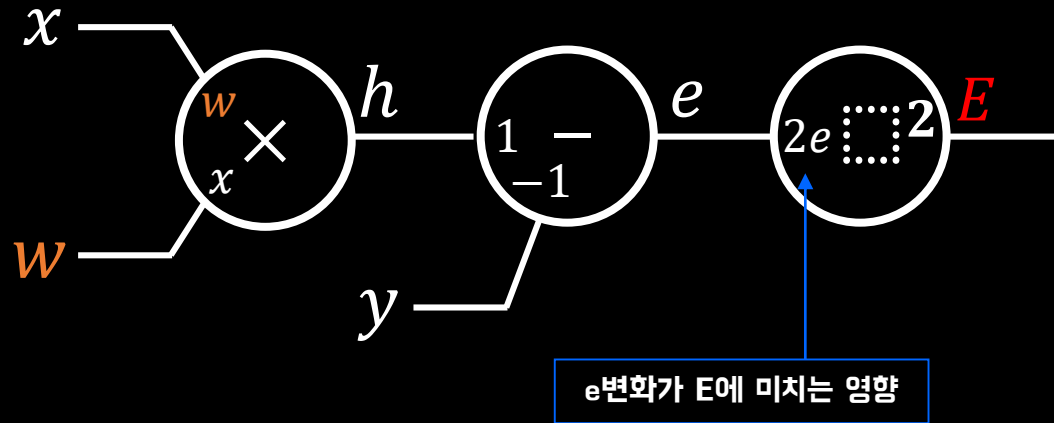
오류 계산 그래프와 미치는 영향



오류 계산 그래프와 미치는 영향

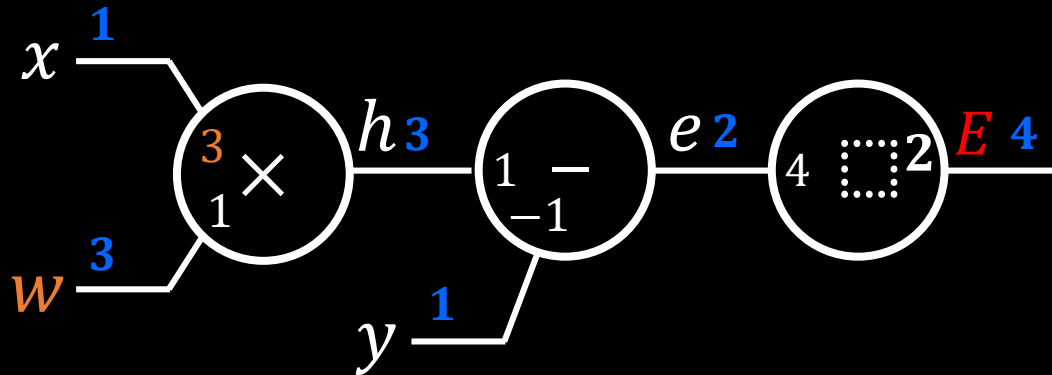


오류 계산 그래프와 미치는 영향



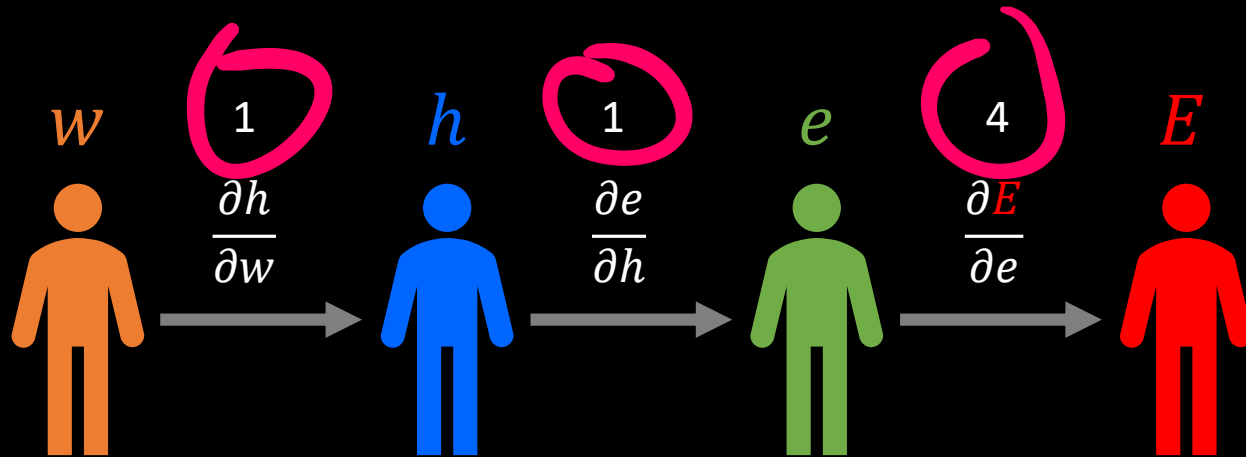
오류 계산 그래프와 미치는 영향

$(x, y) = (1, 1)$ 이고 w 는 3일 경우



각 게이트(연산) 별로 지역(local) 미치는 영향은 알았다.
이로부터 w 변화가 E 에 미치는 영향력은 알 수 있을까?

사람 사이의 영향력

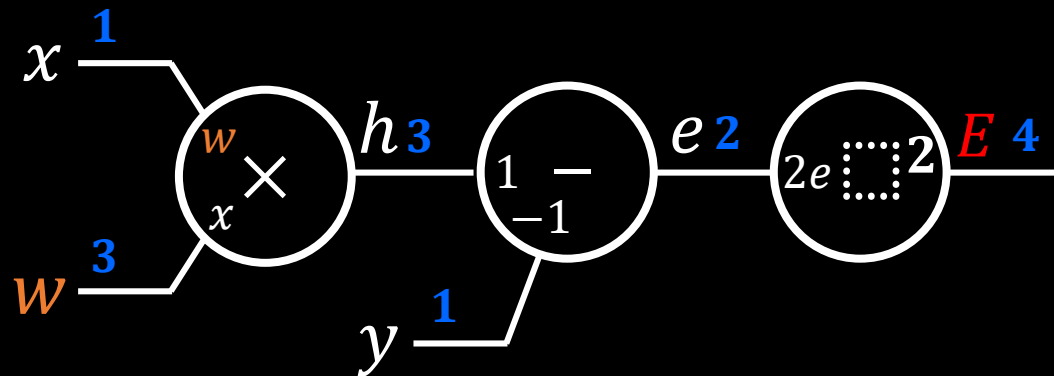


$$\frac{\partial E}{\partial w} = \frac{\partial h}{\partial w} \cdot \frac{\partial e}{\partial h} \cdot \frac{\partial E}{\partial e} = 1 \cdot 1 \cdot 4 = 4$$

체인 룰 (chain rule)

앞으로 전파

$(x, y) = (1, 1)$ 이고 w 는 3일 경우 **에러** 값은?

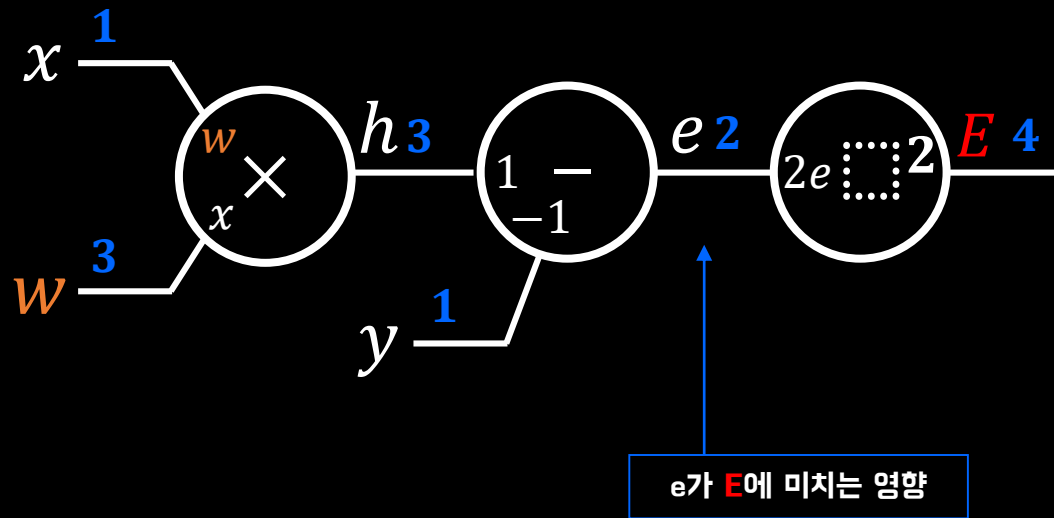


에러(E)가 크다.

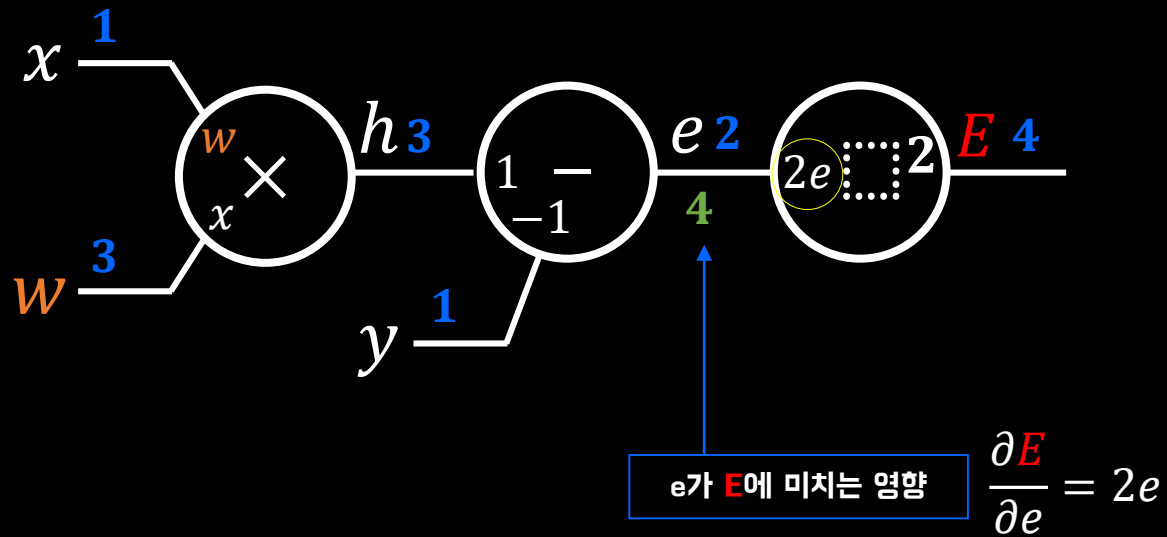
에러(E)가 줄어들도록 w 를 조절하자. 어떻게???

w 변화가 **E**에 미치는 영향(기울기)을 구한 후 $w = w - \alpha * (기울기)$

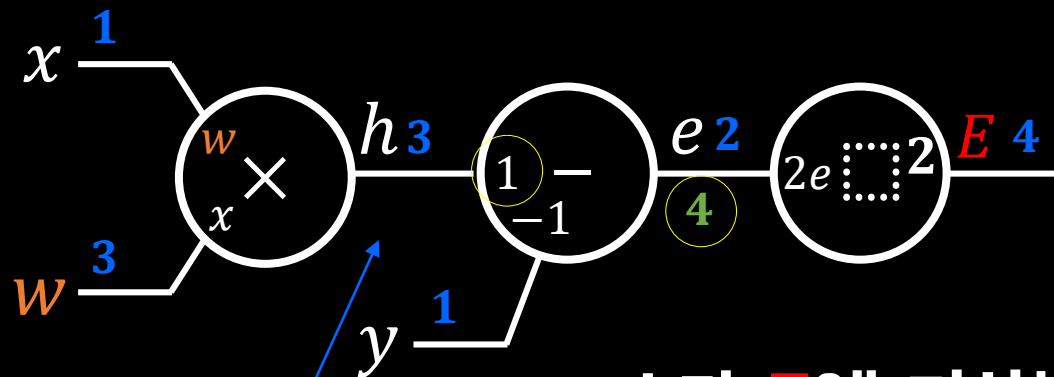
역전파와 체인룰



역전파와 체인룰



역전파와 체인룰

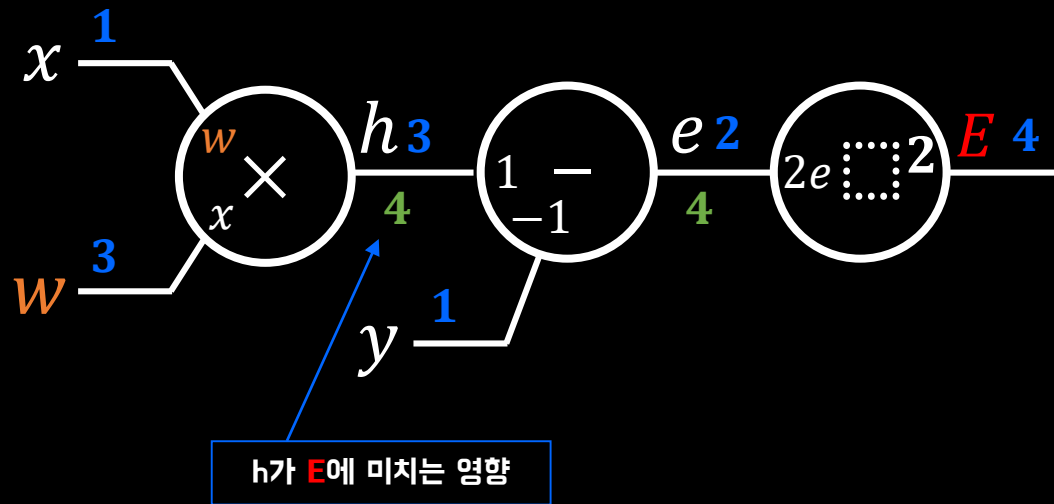


h가 **E**에 미치는 영향

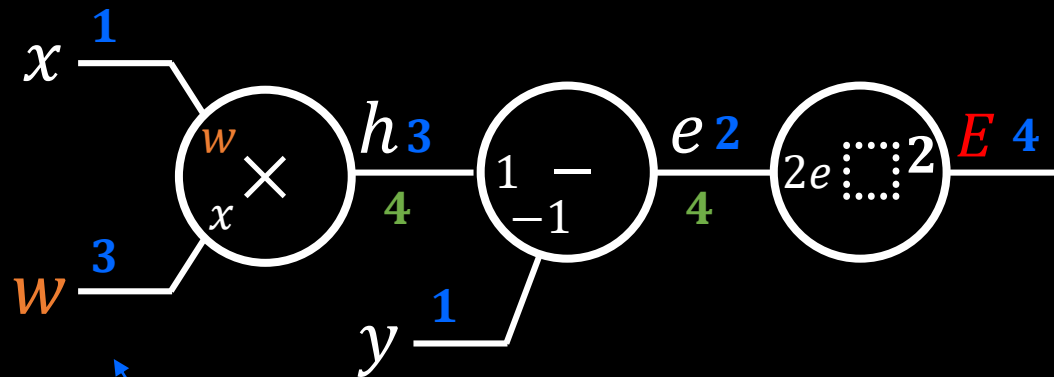
h가 **E**에 미치는 영향

$$\frac{\partial E}{\partial h} = \frac{\partial e}{\partial h} \cdot \frac{\partial E}{\partial e} = 1 \cdot 4$$

역전파와 체인룰



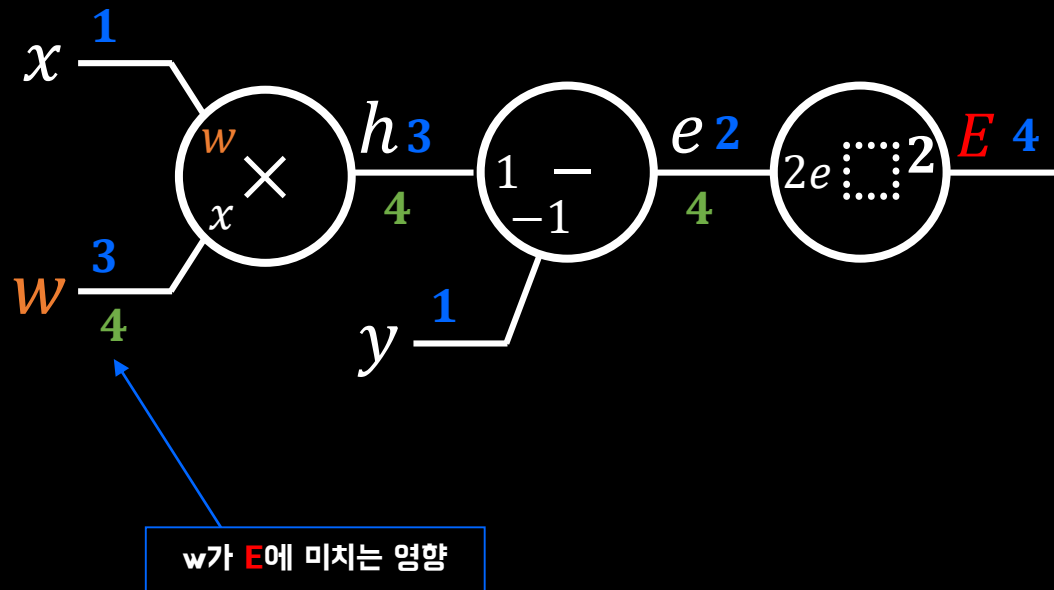
역전파와 체인룰



w가 E에 미치는 영향

$$\frac{\partial E}{\partial w} = \frac{\partial h}{\partial h} \cdot \frac{\partial e}{\partial h} \cdot \frac{\partial E}{\partial e} = x \cdot 1 \cdot 4$$

역전파와 체인룰



따라서 역전파 (back-propagation)

체인 룰(chain rule)을 적용하여 w 가
오류 E 에 얼마나 영향을 미치는지(기
울기)를 알아내는 과정

$$\frac{\partial E}{\partial w}$$

$$W = 3 - 0.1 * 4^{\frac{\partial E}{\partial w}}$$
$$W = 2.6$$




Tuned parameter
after 1 step learning.

```
import tensorflow as tf
```

```
#----- training data  
x_data = [1]  
y_data = [1]
```

```
#----- a neuron / neural network  
w = tf.Variable(tf.random_normal([1]))  
hypo = w * x_data
```

train operation to
update w to
minimize cost(error)



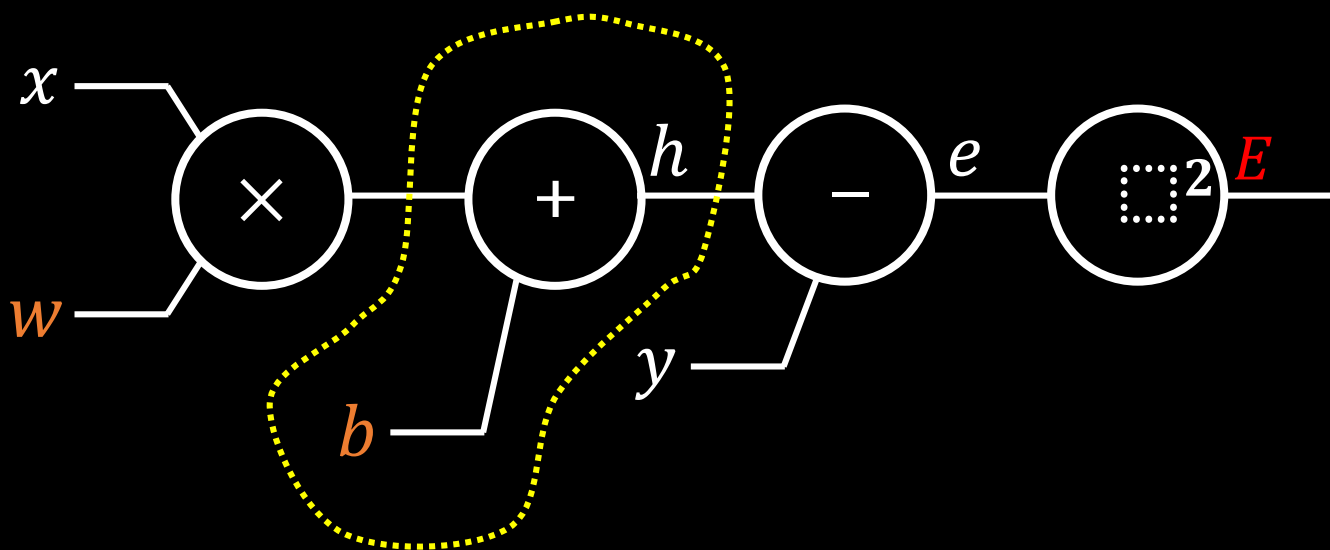
```
#----- learning  
cost = (hypo - y_data) ** 2  
  
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)  
  
sess = tf.Session()  
sess.run(tf.global_variables_initializer())  
  
for i in range(1001):  
    sess.run(train) # 1-run, 1-update of w -> 1001 updates  
  
    if i % 100 == 0:  
        print('w:', sess.run(w), 'cost:', sess.run(cost))
```

```
#----- testing(prediction)  
x_data = [2]  
print(sess.run(x_data * w))
```

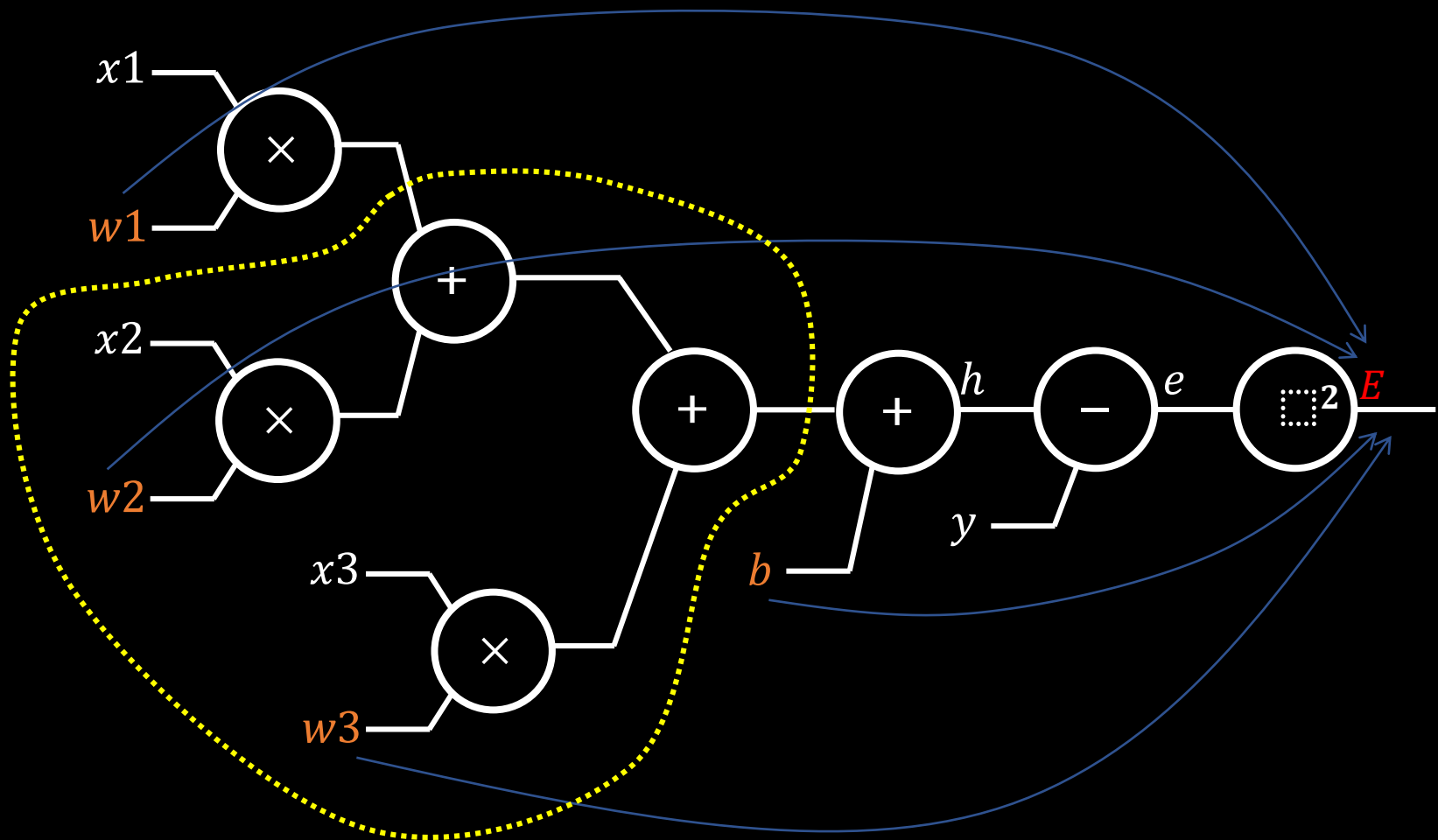
계산 그래프 확장

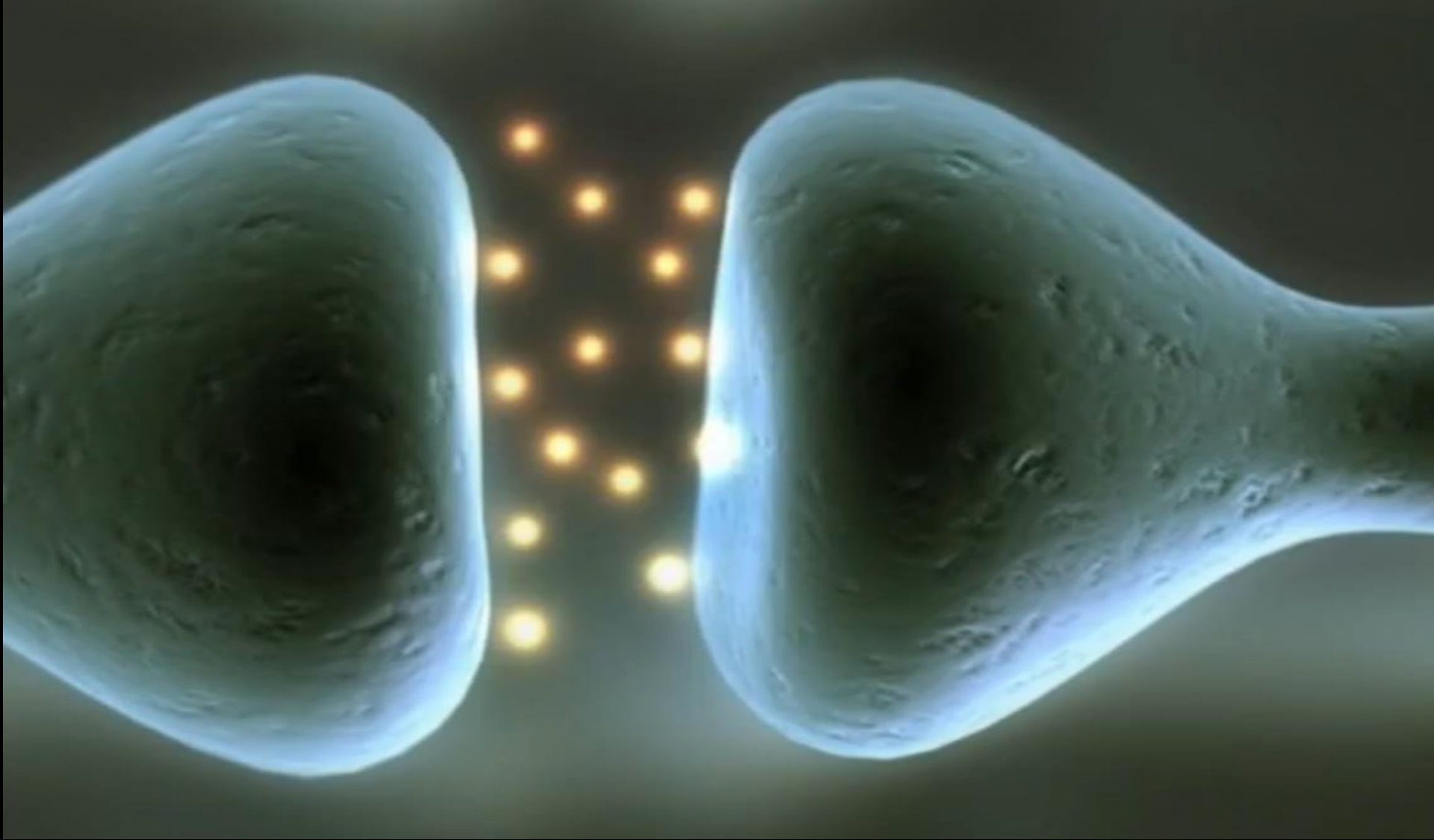
- bias가 있을 경우 (+ 게이트)
- 뉴런 입력이 3개일 때 (+ 게이트)
- 뉴런이 2개일 때
- 튜닝할 파라미터는 모두 몇 개?

$$E = ((wx + b) - y)^2$$



$$E = ((w_1x_1 + w_2x_2 + w_3x_3) + b) - y)^2$$





학습, 더 새롭고 좋은 연결을 만드는 것

Meaning of cost(error)

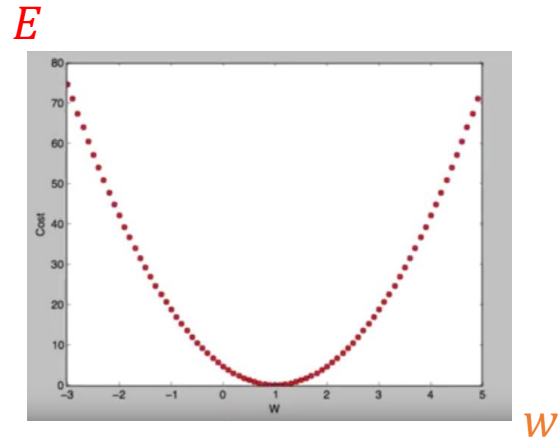
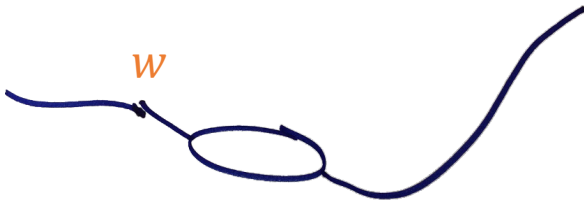
- 기울기가 큼 → 최저점에서 멀리 떨어짐 → 오류가 큼 → **bad!** → big penalty → 스트레스/고통이 큼 → big update(w)
- 기울기가 작음 → 최저점에 가까움 → 오류가 작음 → **not bad!** → small penalty → 스트레스/고통이 작음 → small update(w)
- 기울기 0 → 최저점! → 정답을 맞춤! → **great!** → no penalty → no update(w) → learning ended!

우리 마음 속의 cost(error, loss, stress) function

‘좋다’, ‘나쁘다’를 느끼게 하는 기저

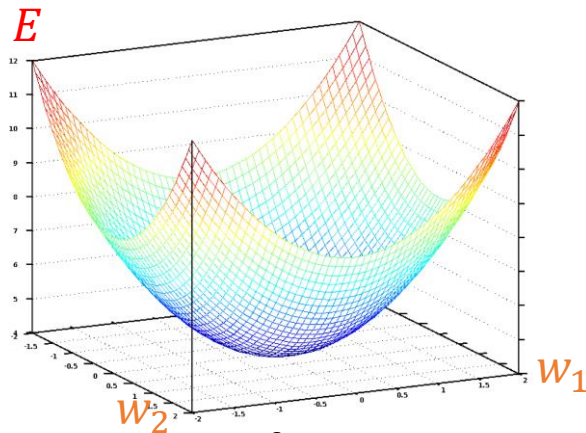
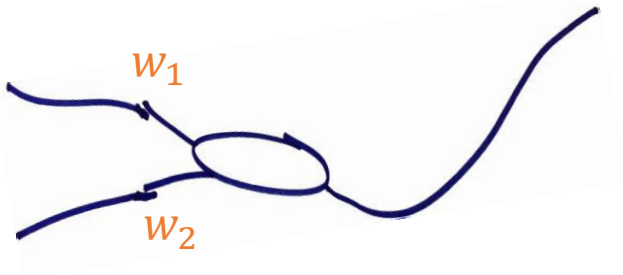
Cost(Error) Graph

$$E = (w \cdot 1 - 1)^2$$



convex function

$$E = (W \cdot 1 - 1)^2$$



convex function

02.with_bias.py

Parameter tuning
including bias

03.py

Using multiple
data

04.py

Training a neuron
having multiple
inputs