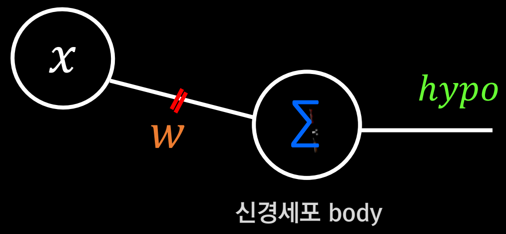
AI and Deep Learning

딥 러닝
Deep Learning

제주대학교

변 영 철

http://github.com/yungbyun/ml

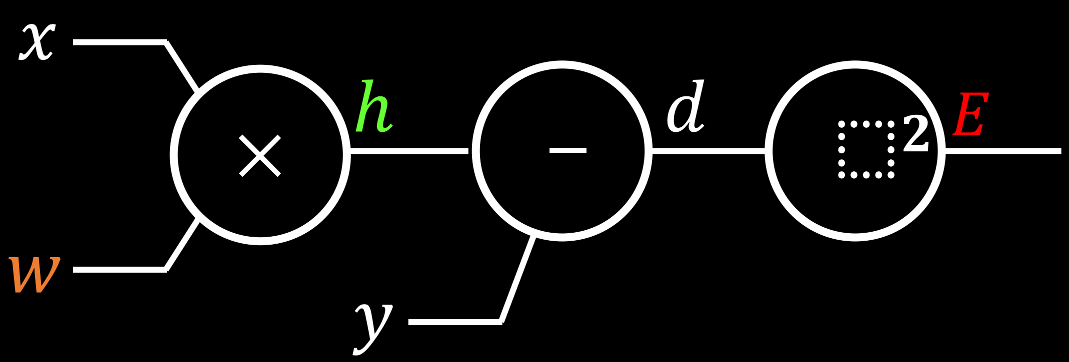


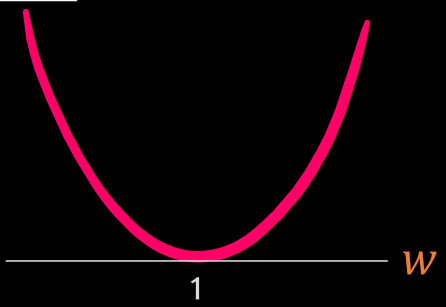

```
#----- a neuron
w = tf.Variable(tf.random_normal([1]))
hypo = w * x_data

#----- learning
cost = (hypo - y_data) ** 2
```

입력 x가 1, 정답 y가 1일 때

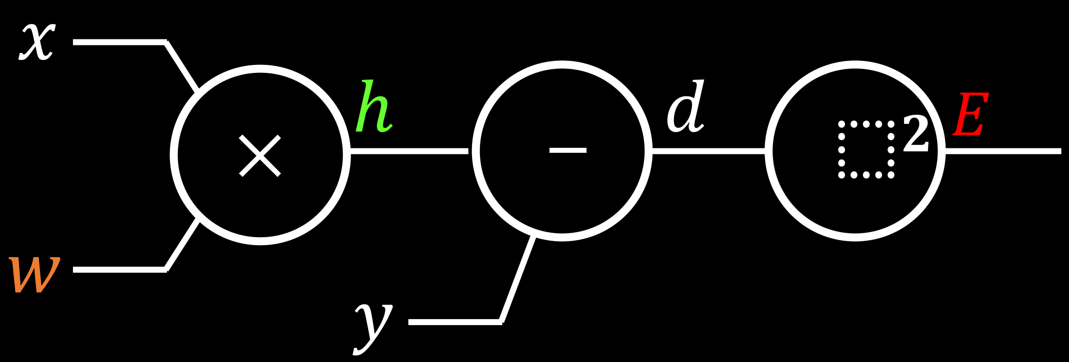
$$\boxed{cost(E) = (w \cdot 1 - 1)^2}$$

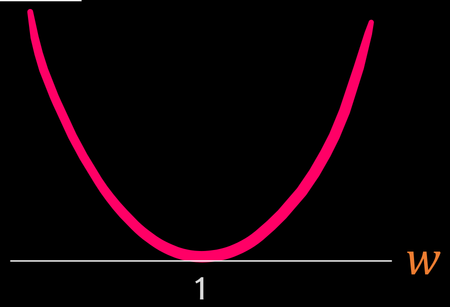
오류 $cost(E)$ 계산 그래프





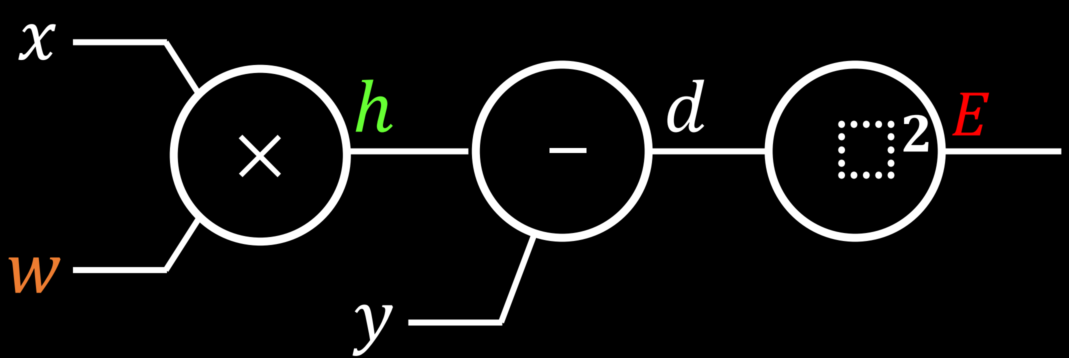
오류 $cost(E)$ 계산 그래프

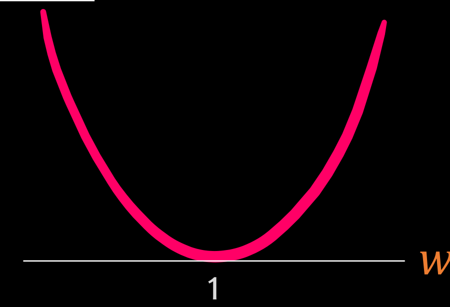




w를 조절하는 것 = 학습

오류 $cost(E)$ 계산 그래프

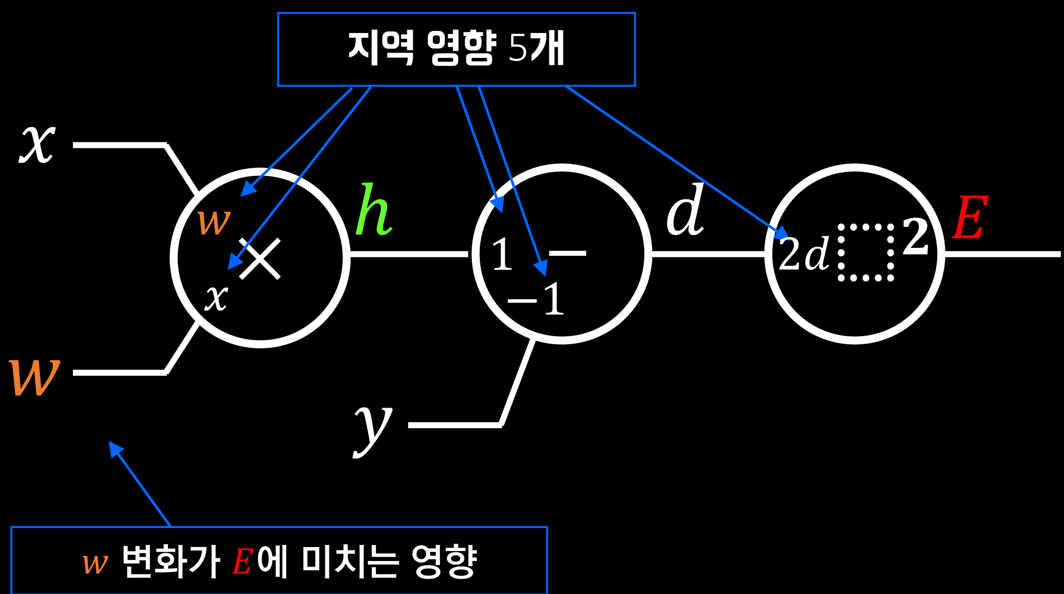


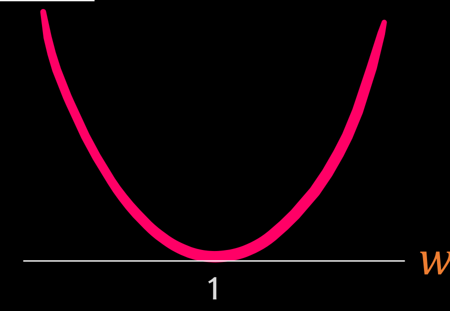


w를 조절하는 것 = 학습

텐서플로우에서는 어떻게 w를 조절할까? → 오류 계산 그래프 이용

오류 $cost(E)$ 계산 그래프





= 계산 그래프에서 경로 상에 있는 지역 영향의 곱

$= x \cdot 1 \cdot 2d$

$= x \cdot 1 \cdot 2(w \cdot x - y)$

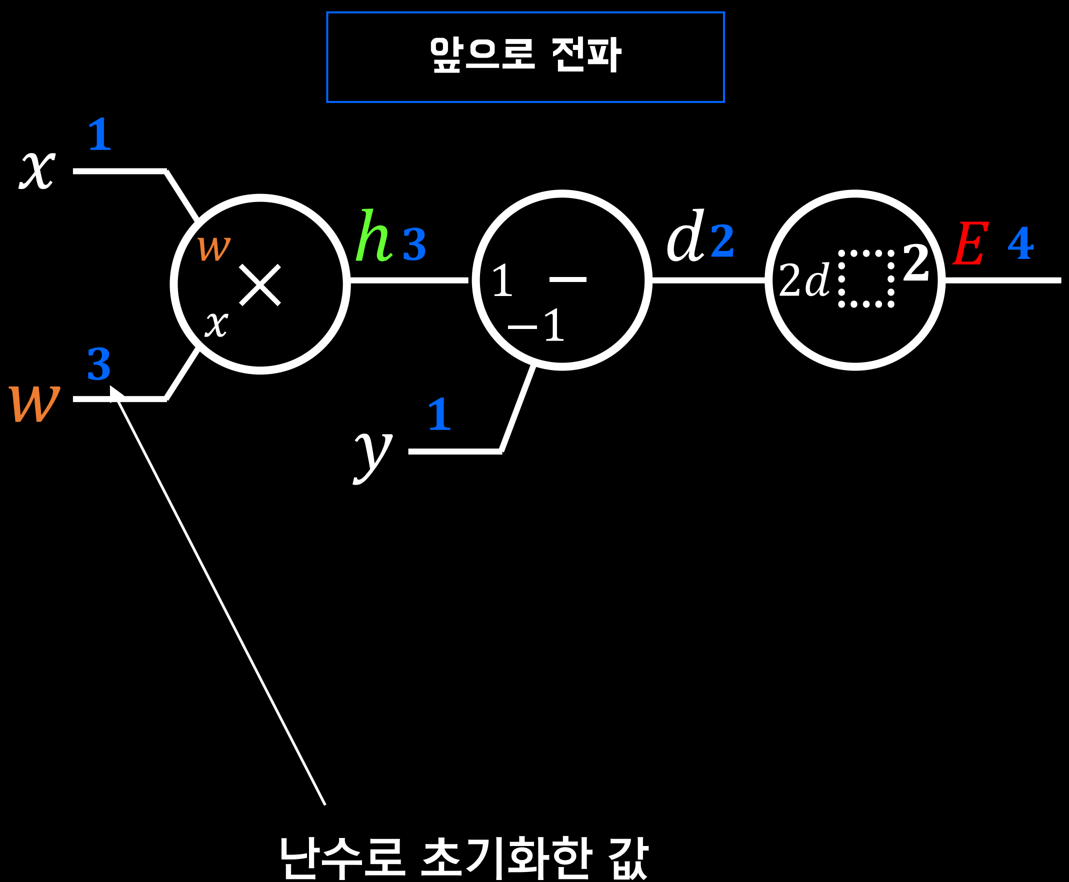
$= 1 \cdot 1 \cdot 2(w \cdot 1 - 1)$

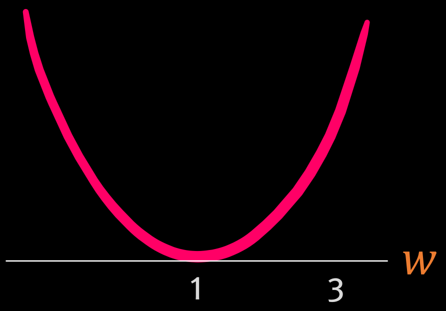
$= 2(w - 1)$

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

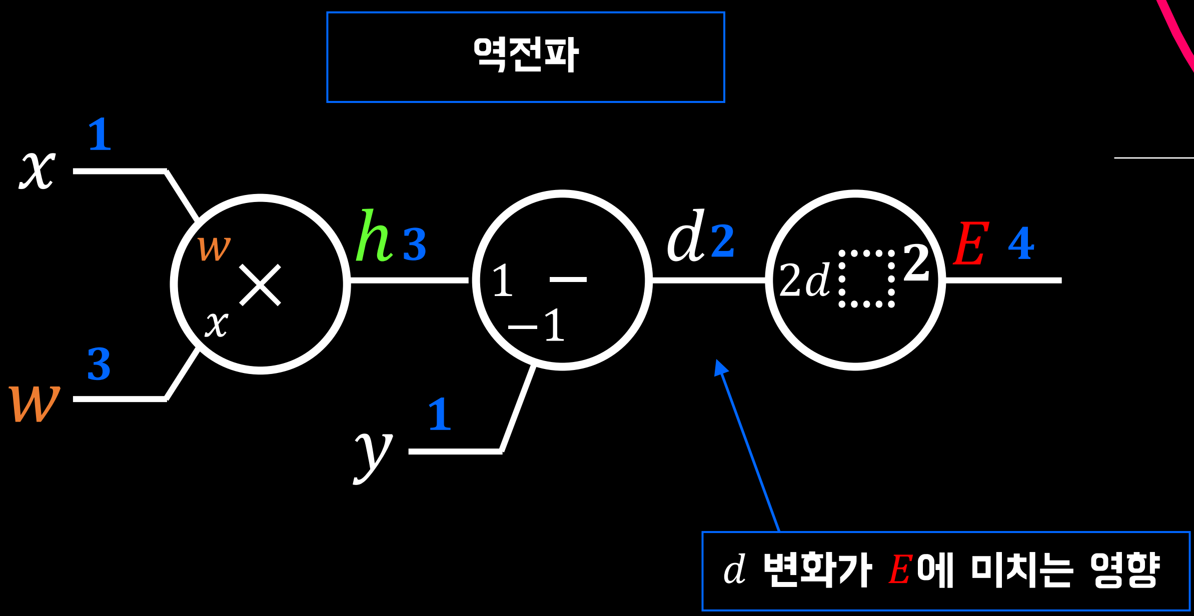
1. 계산 그래프 체인룰
2. 미분 방정식 방법

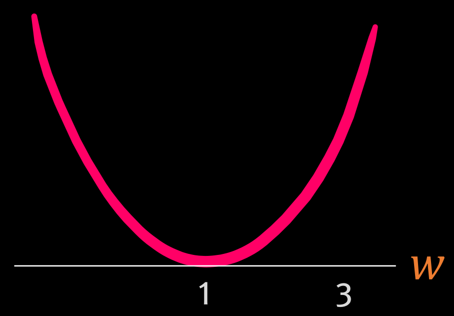
오류 E 계산 그래프



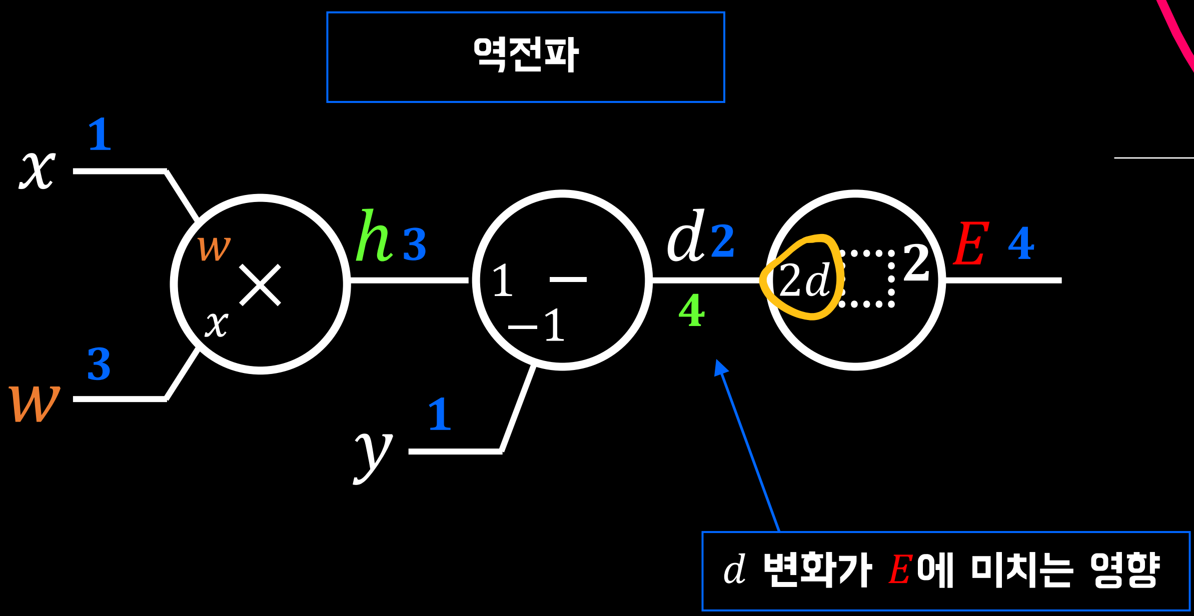


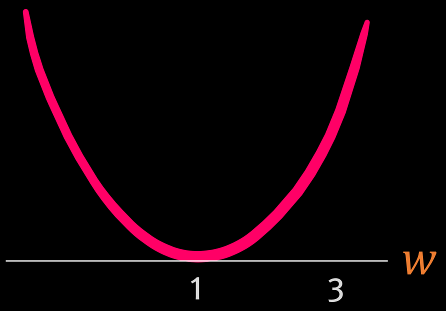
오류 E 계산 그래프



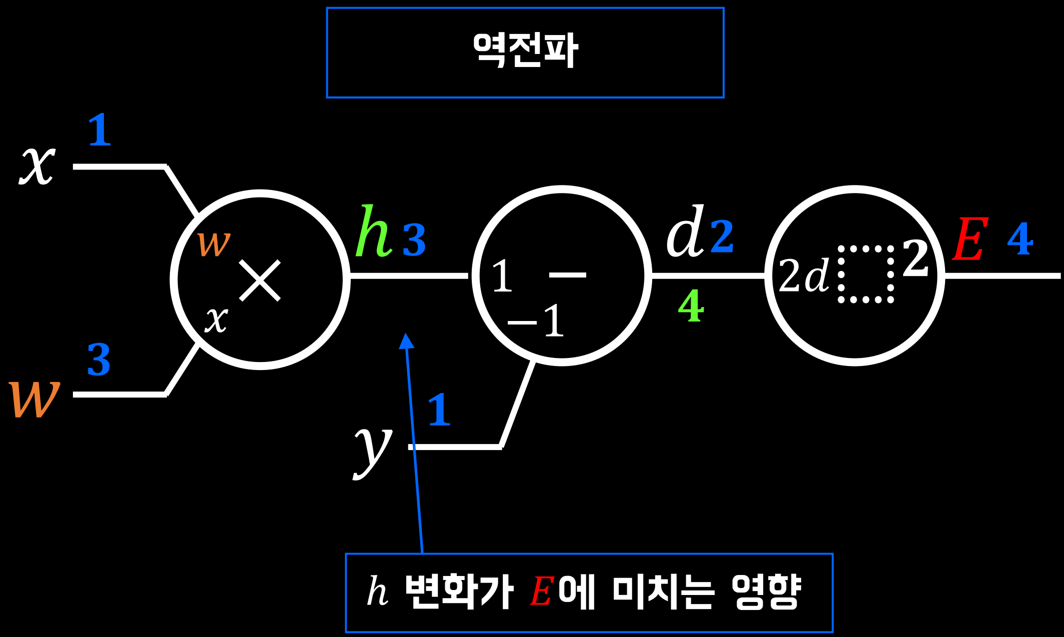


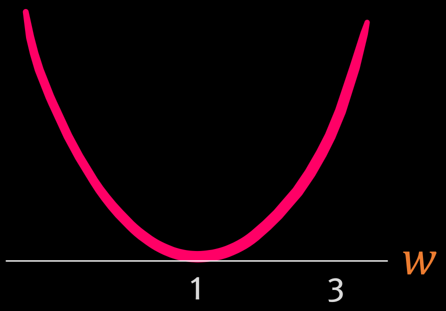
오류 E 계산 그래프



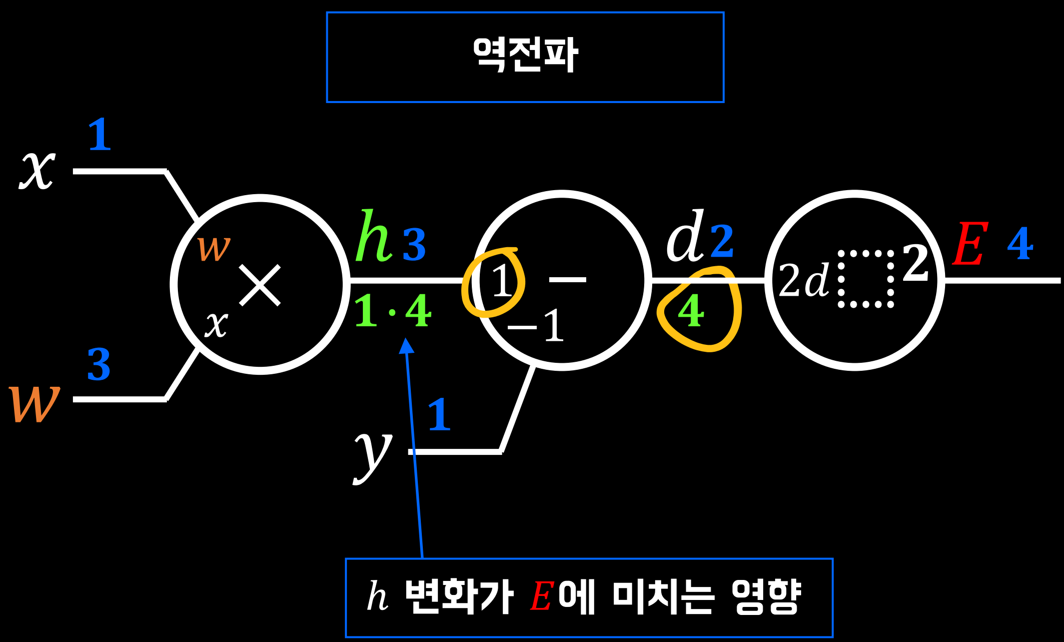


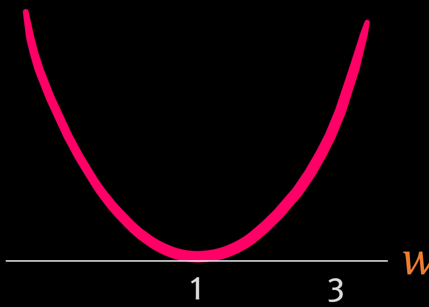
오류 E 계산 그래프



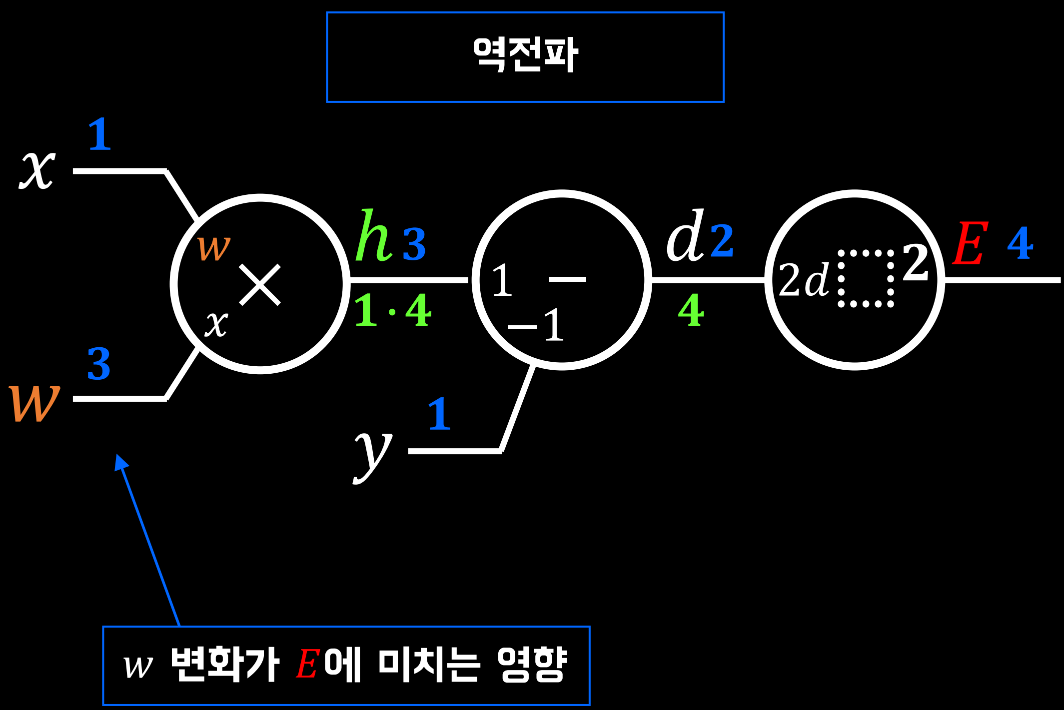


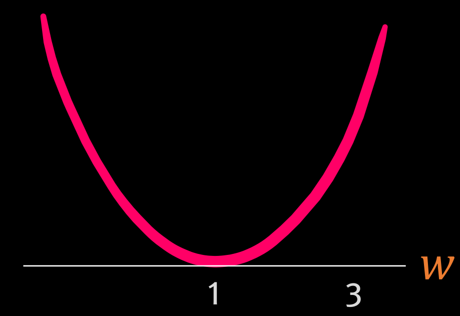
오류 E 계산 그래프



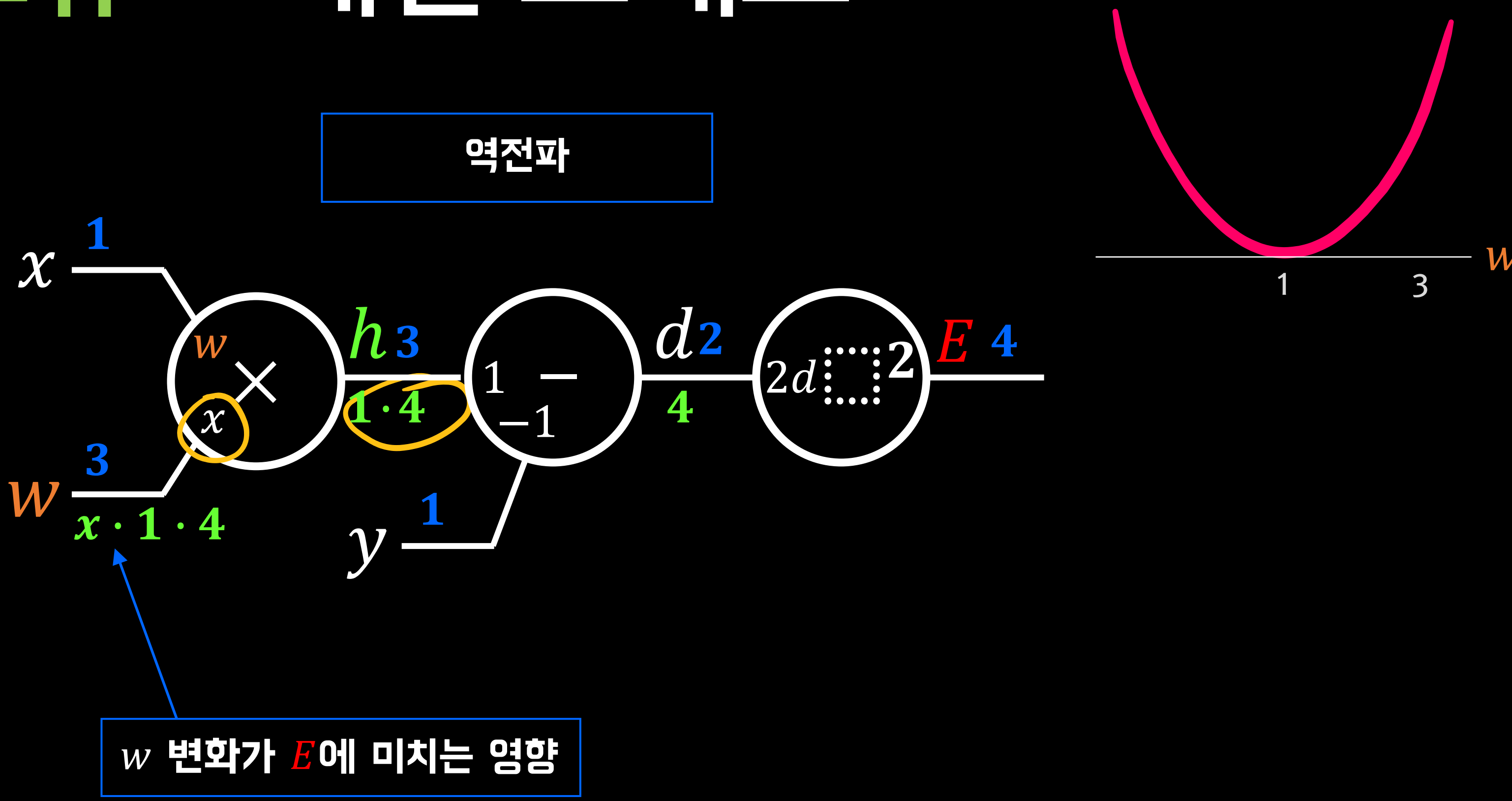


오류 E 계산 그래프



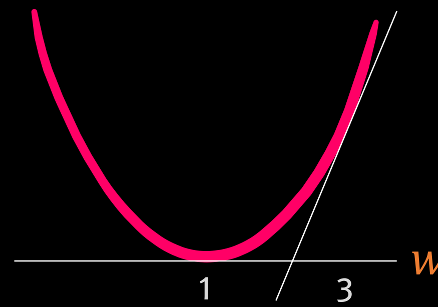


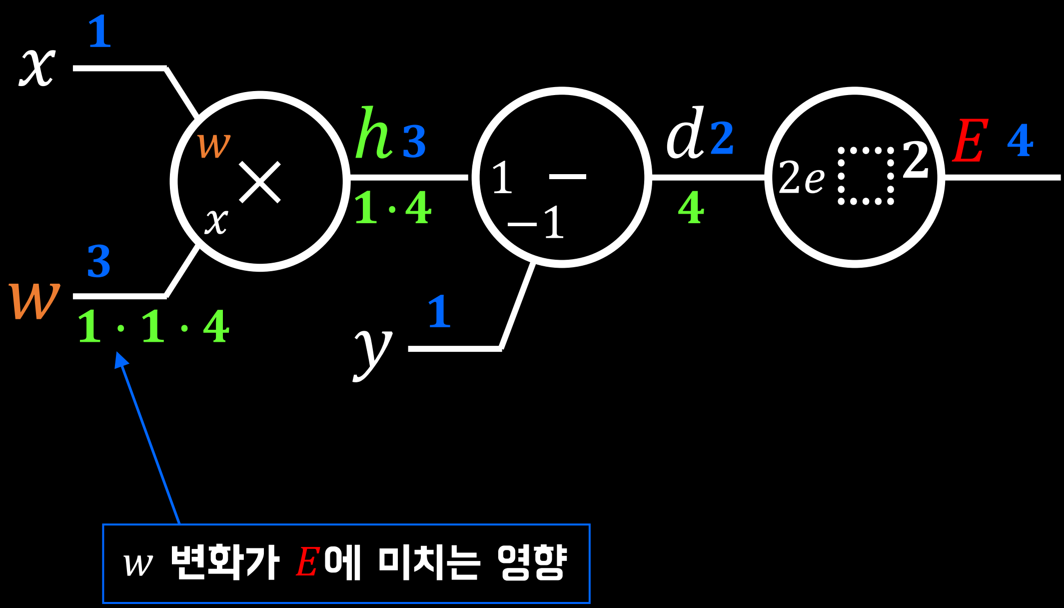
오류 E 계산 그래프



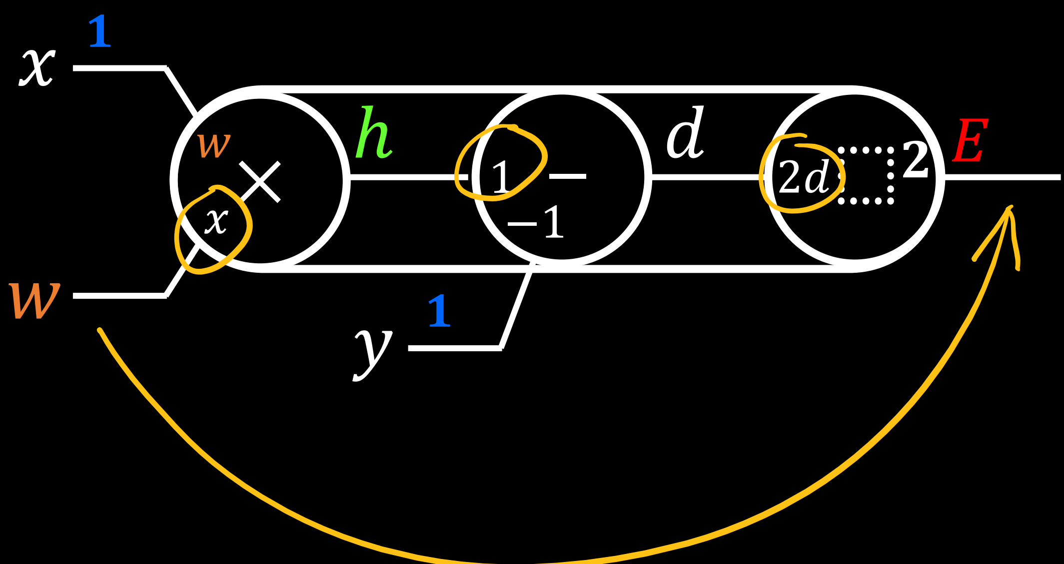
오류 E 계산 그래프

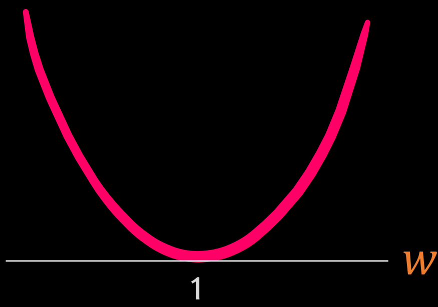
$E = (w \cdot 1 - 1)^2$



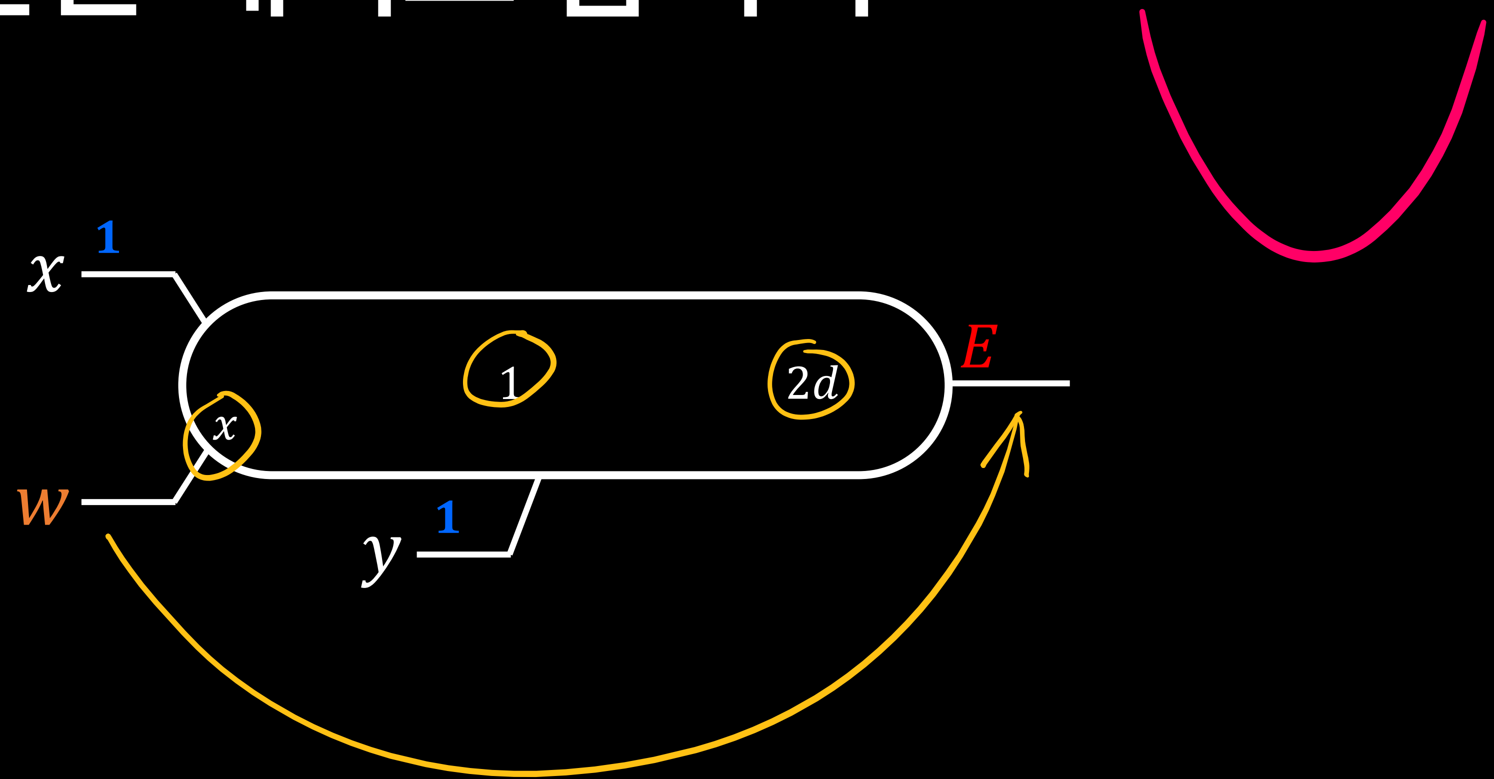


연산 게이트 합치기

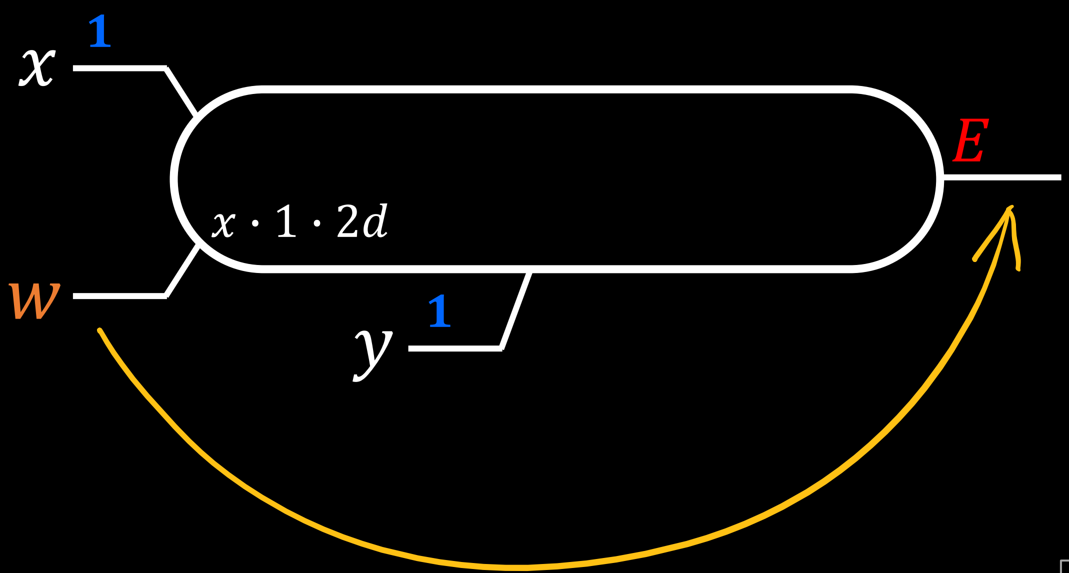


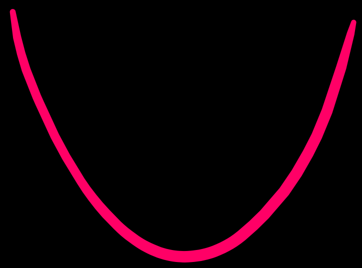


연산 게이트 합치기



연산 게이트 합치기

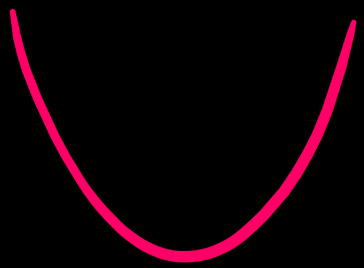


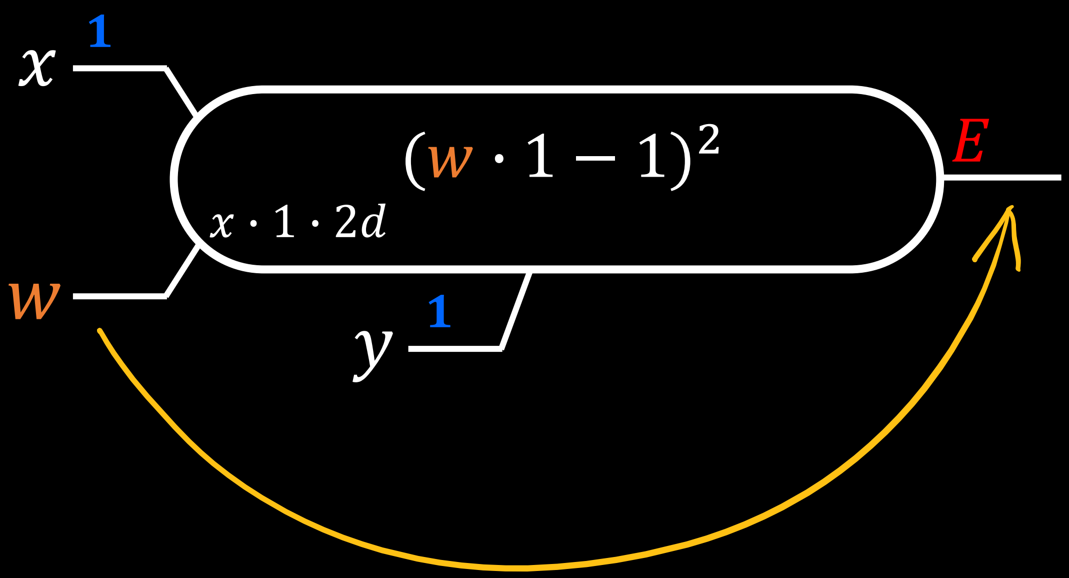


1. 계산 그래프 체인룰
2. 미분 방정식 방법

연산 게이트 합치기

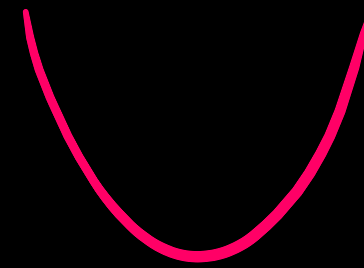
$$E = (w \cdot 1 - 1)^2$$

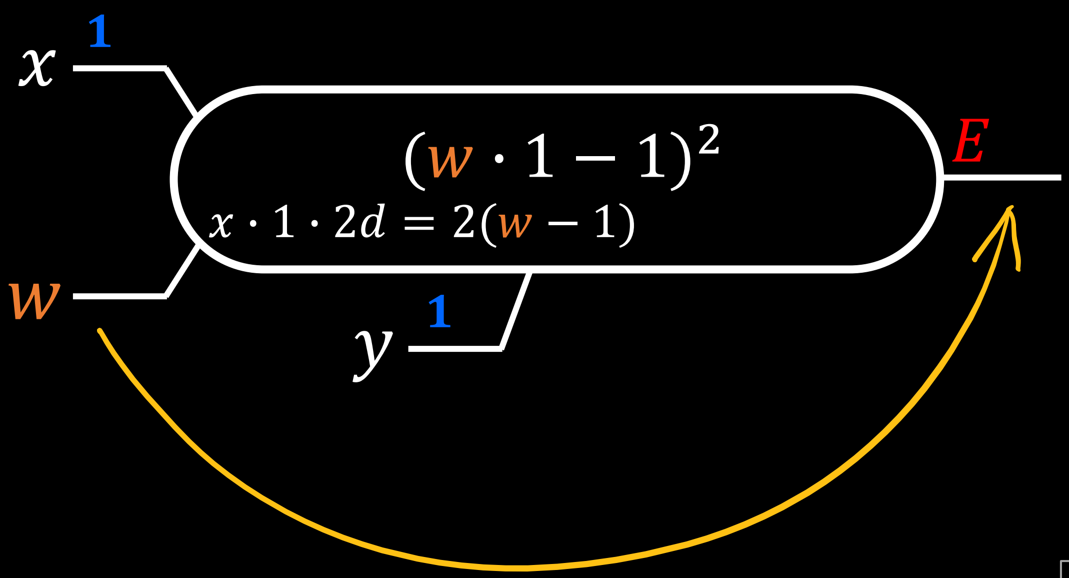




연산 게이트 합치기

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

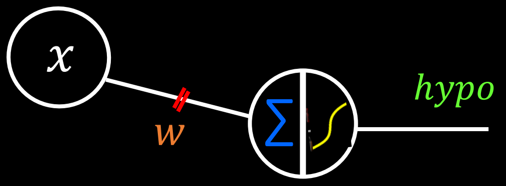




1. 계산 그래프 체인룰
2. 미분 방정식 방법

오류 E 계산 그래프

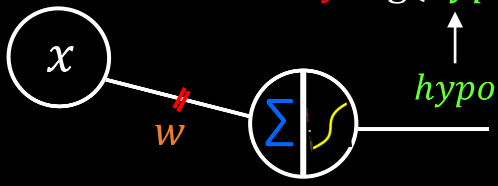
Linear Regression(선형회귀) vs. Logistic Regression(논리회귀)



오류 E 계산 그래프

Binary Cross Entropy 오류함수(loss function)

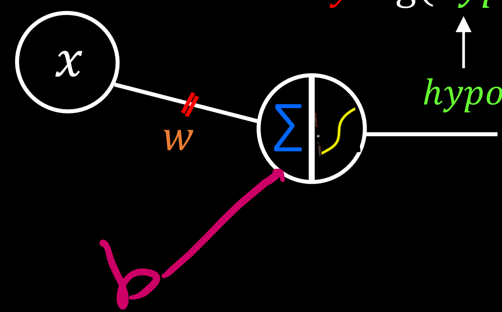
$$E = -y\log(hypo) - (1 - y)\log(1 - hypo)$$

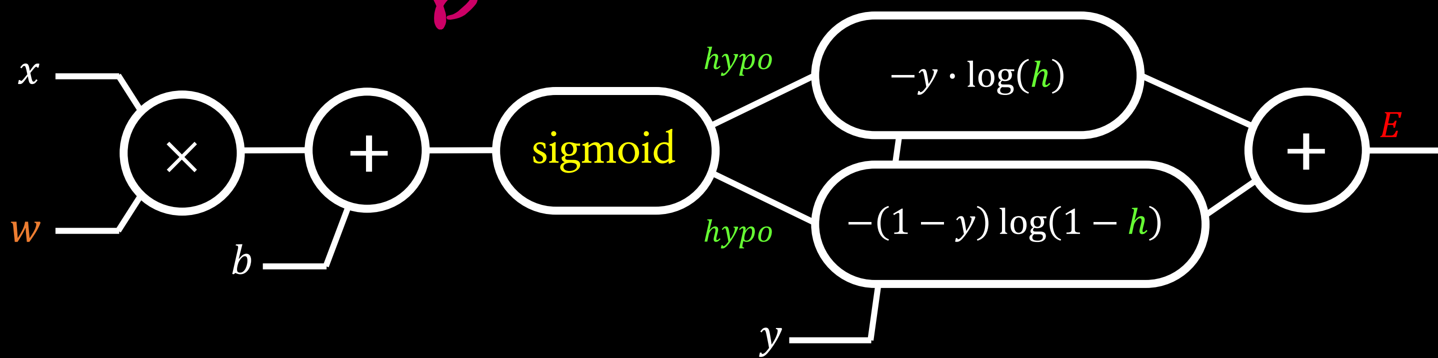


오류 E 계산 그래프

Binary Cross Entropy 오류함수(loss function)

$$E = -y\log(hypo) - (1-y)\log(1-hypo)$$

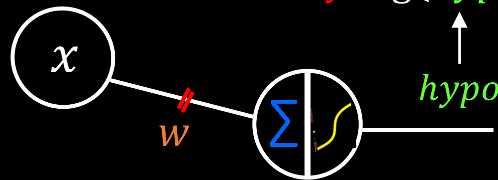


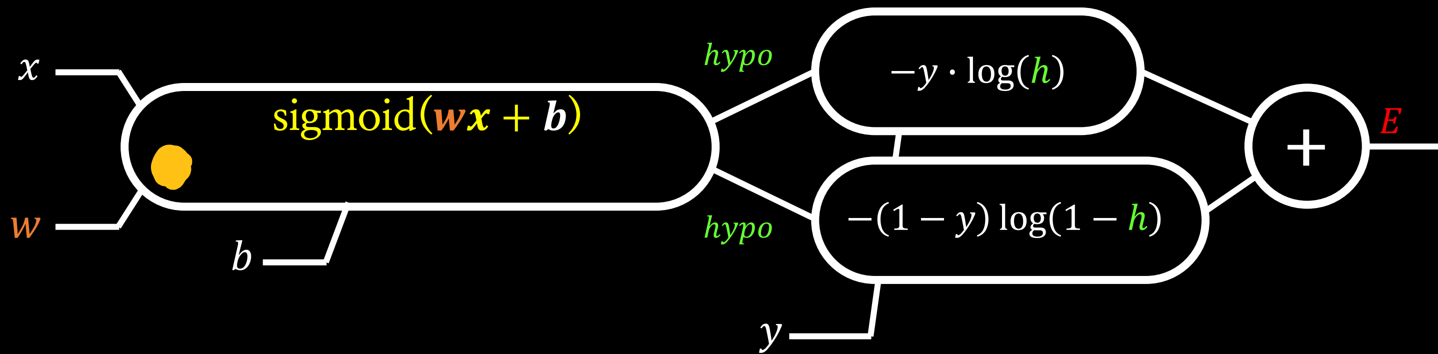


오류 E 계산 그래프

Binary Cross Entropy 오류함수(loss function)

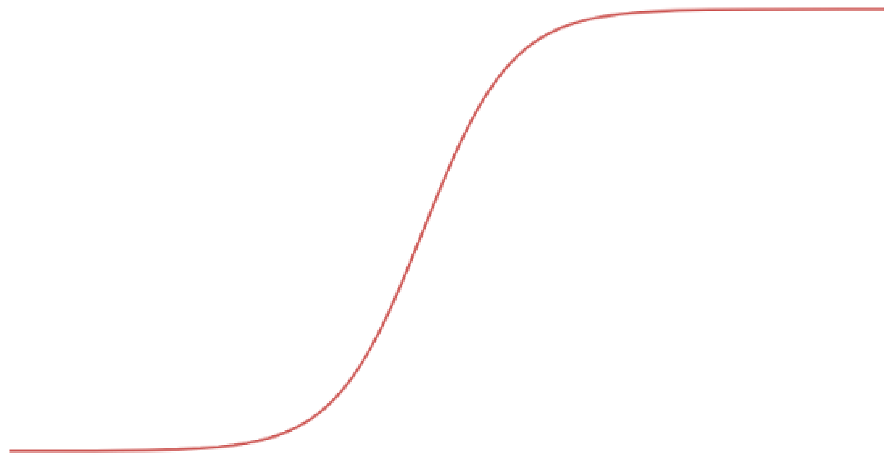
$$E = -y\log(hypo) - (1-y)\log(1-hypo)$$

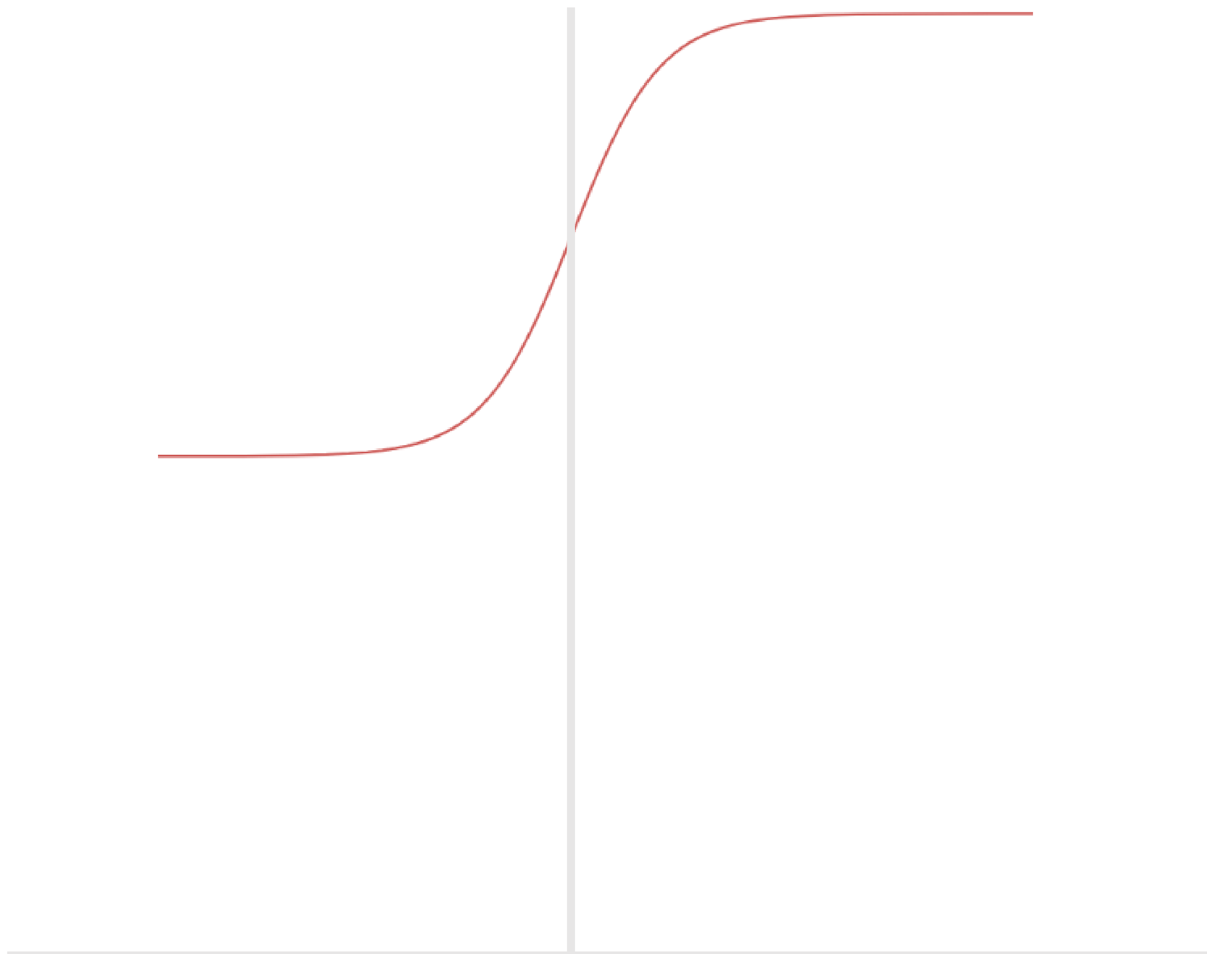




$$\frac{\partial E}{\partial w} =$$

$$\frac{\partial hypo}{\partial w} =$$

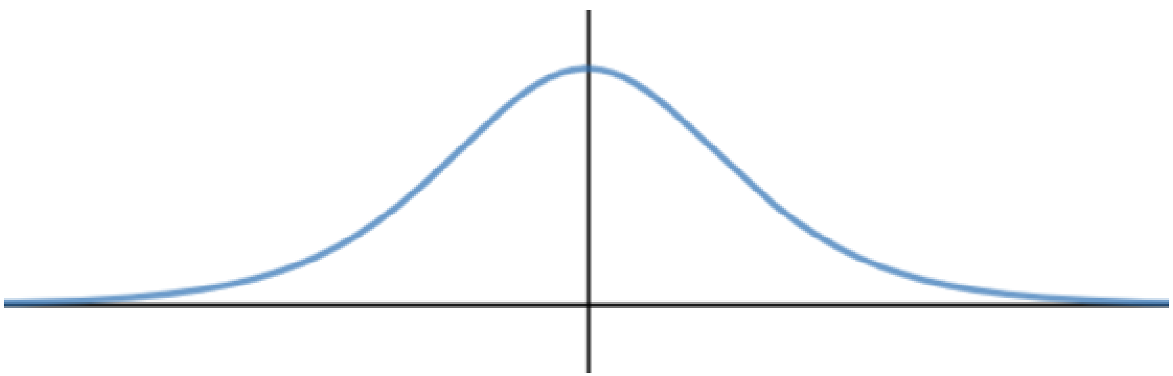




σ: *sigmoid*

$$(\sigma)(1-\sigma)$$

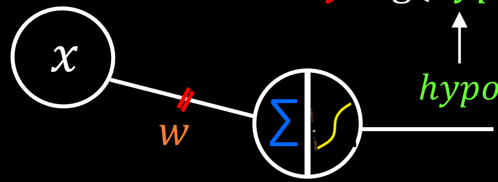
$$= \left(\frac{1}{1+e^{-wx}}\right)\left(1-\frac{1}{1+e^{-wx}}\right)$$

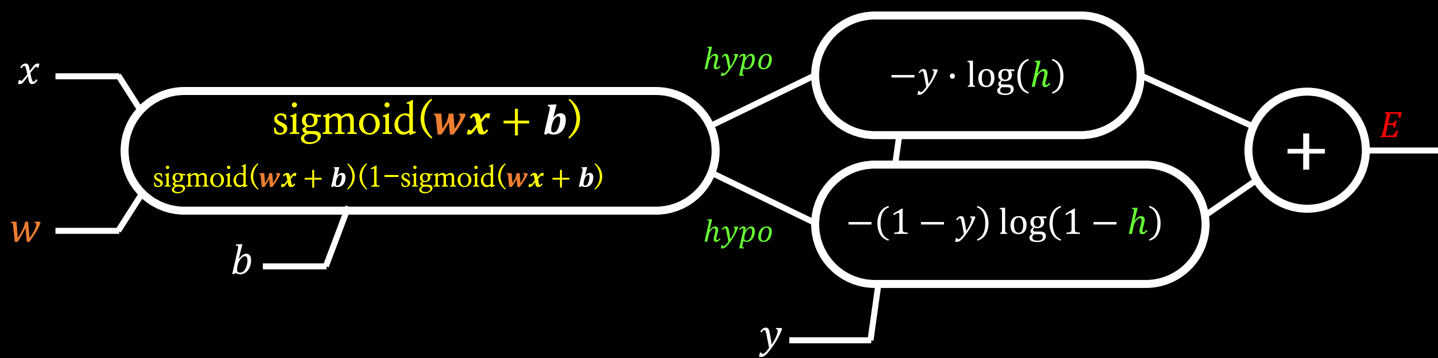


오류 E 계산 그래프

Binary Cross Entropy 오류함수(loss function)

$$E = -y\log(hypo) - (1-y)\log(1-hypo)$$

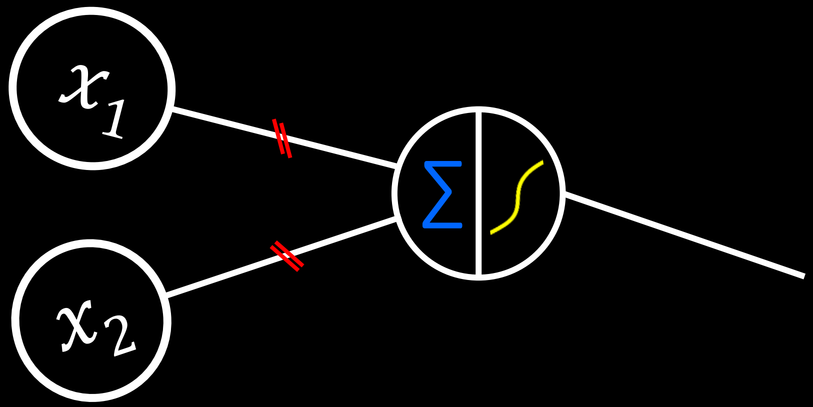




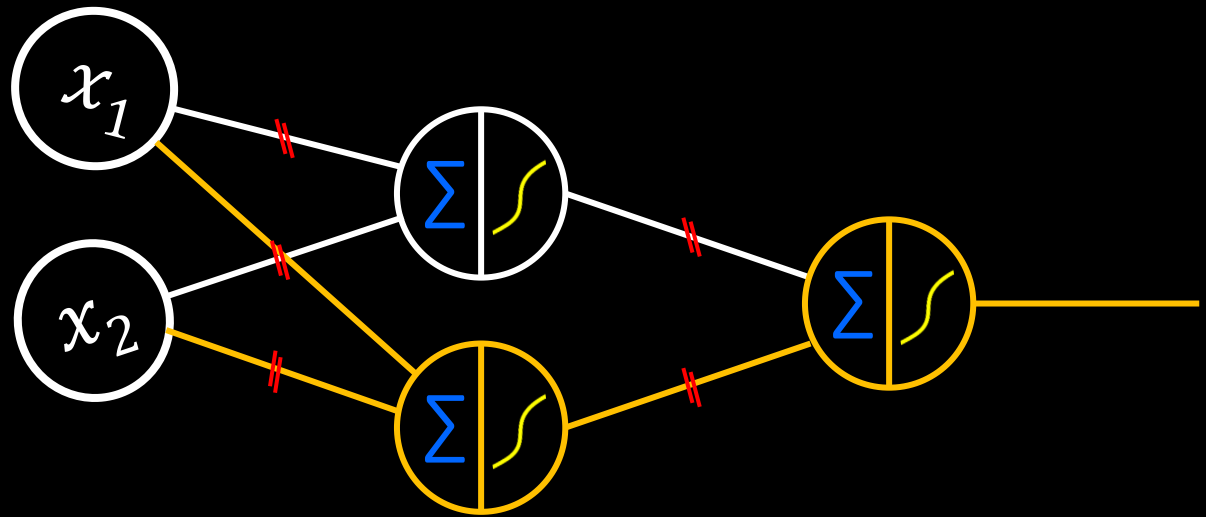
$$\frac{\partial E}{\partial w} =$$

$$\frac{\partial hypo}{\partial w} =$$

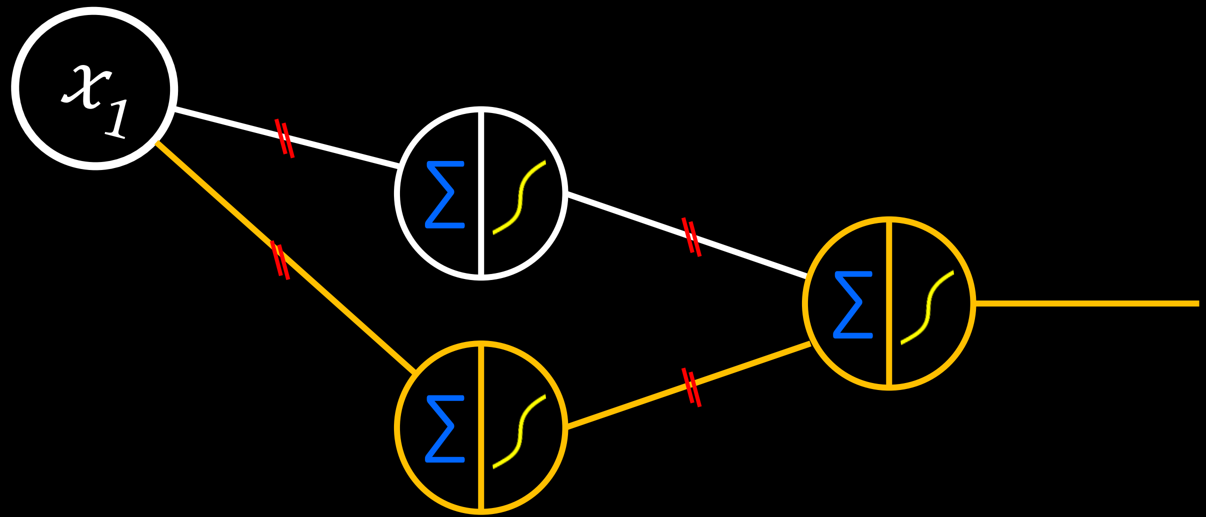
뉴런 1개 (2 입력)



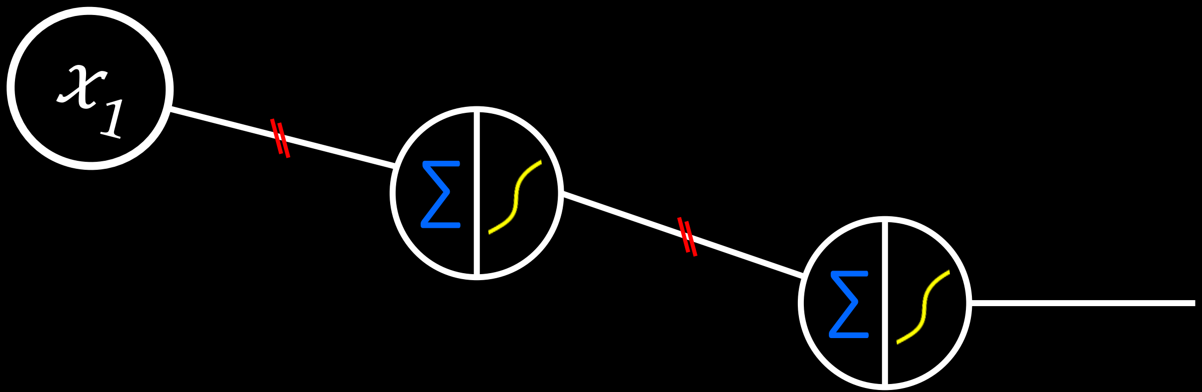
뉴런 3개 (3층 신경망)



3층 신경망 (단순화)

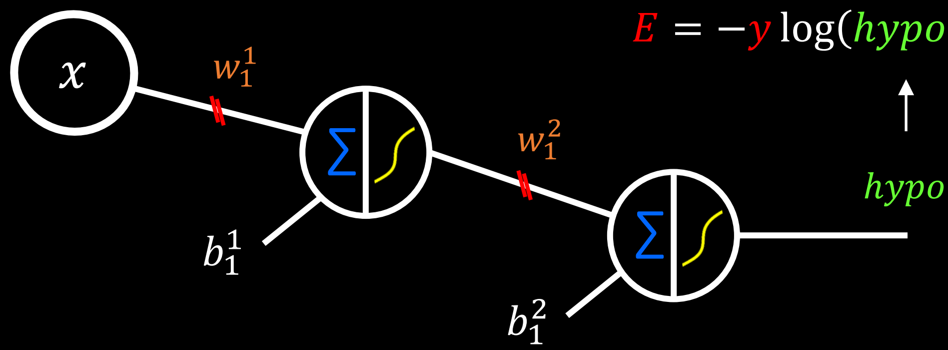


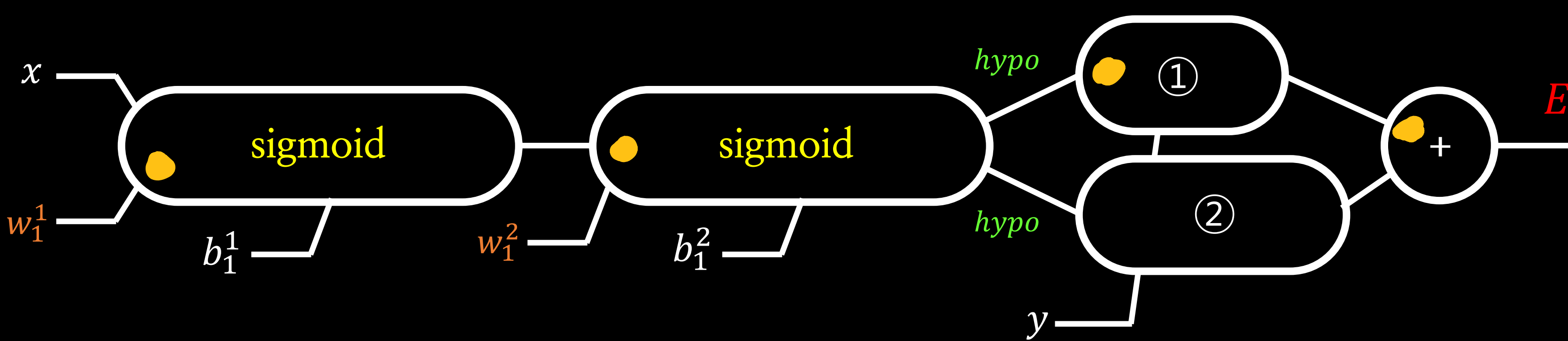
3층 신경망 (단순화)



오류 계산 그래프

$$E = \overset{①}{-y\log(hypo)} - \overset{②}{(1-y)\log(1-hypo)}$$

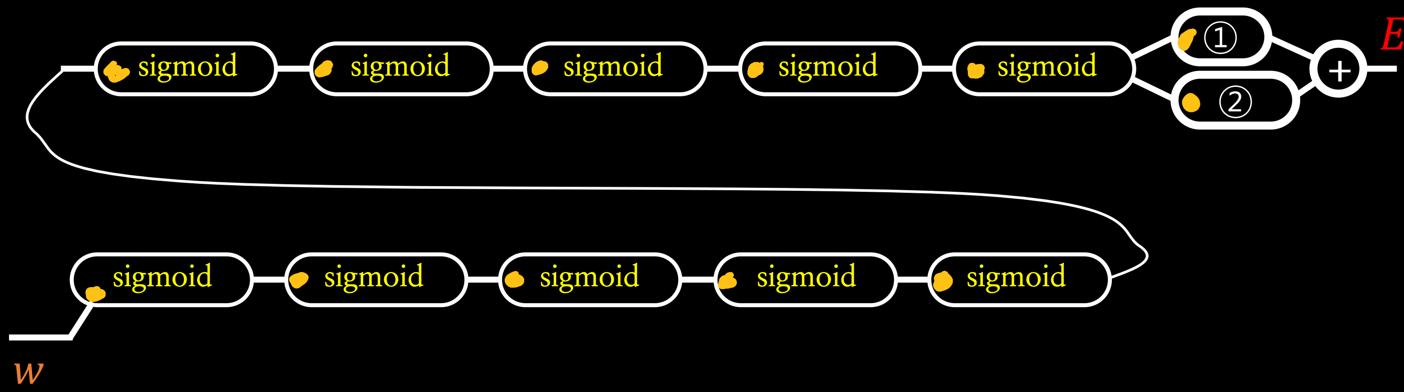




10 층 신경망

The giant monster, computational graph!

오류 E 계산 그래프



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

힌트: chain rule!

Vanishing Gradient 사라지는 영향력

- sigmoid 함수의 기울기를 구하면(미분) sigmoid x (1-sigmoid)
- 한 뉴런에 대해 두번의 sigmoid 곱, 따라서 10층의 뉴런의 경우 20번의 sigmoid 곱
- sigmoid 함수는 0과 1 사이의 값을 반환

0.5x0.5 x 0.1x0.9 x 0.8x0.2 x 0.5x0.5 x 0.3x0.7 x 0.4x0.6 x 0.5x0.5 x 0.2x0.8 x 0.5x0.5 x 0.6x0.4
= 0.00000010886 x ①

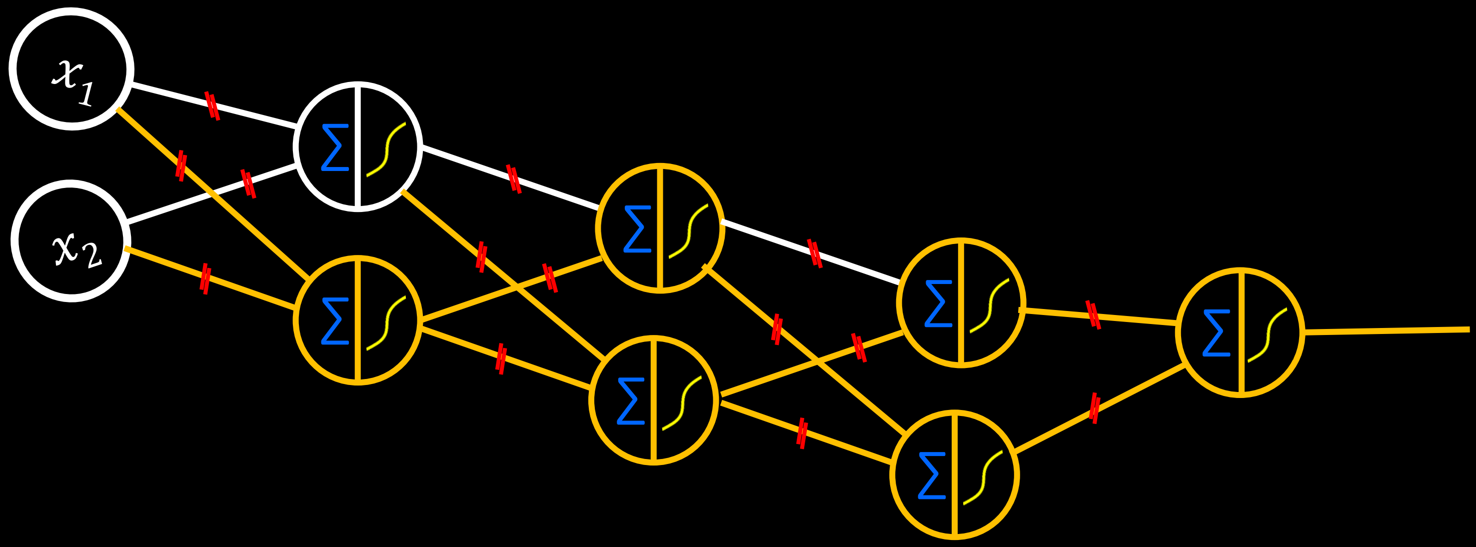
Vanishing Gradient 사라지는 영향력

- 따라서 w가 E에 미치는 영향을 구하려면 수많은 sigmoid 함수를 곱해야 하며 결과는 거의 0에 가까움.
- 사라지는 영향력, Vanishing Gradient
- $w = w - \alpha \cdot$ (거의 0)
- $b = b - \alpha \cdot$ (거의 0)
- 따라서, w와 b가 수정되지 않아 학습이 이뤄지지 않음.

(실습) 19.py

https://github.com/yungbyun/myml

- 5층으로 구성된 신경망으로 XOR 문제를 해결하고자 했으나 Vanishing Gradient 때문에 실패



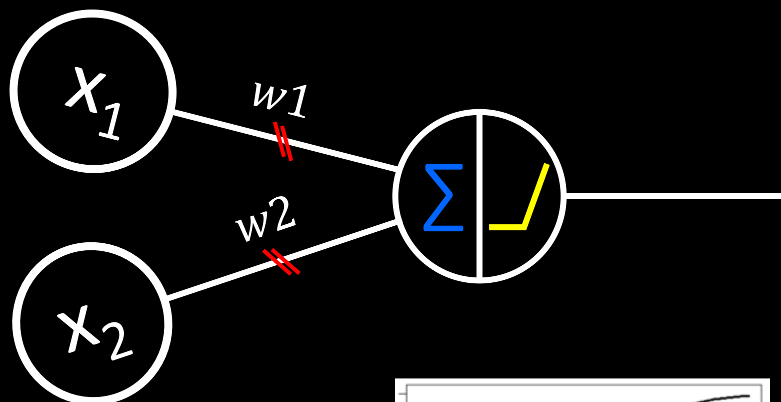
1986년, Hinton 교수가
역전파 알고리즘(back-propagation)을
제안한 이후

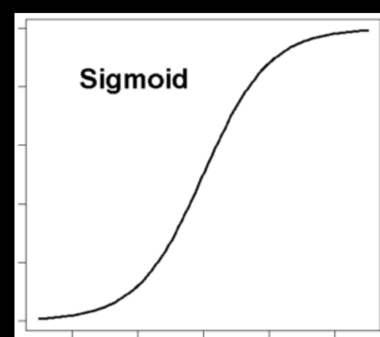
두번째 맞은 인공지능 암흙의 시대 (~2006)

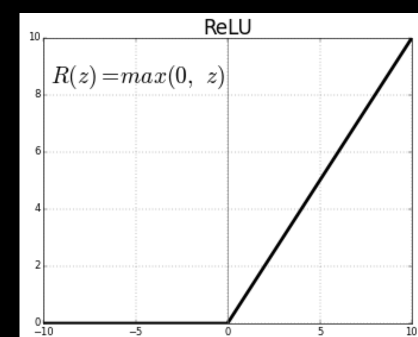
ReLU

Rectified Linear Unit

Logistic 함수 대신 ReLU라는 활성화 함수를 사용함으로써 Vanishing Gradient 문제 해결







(실습) 20.py

https://github.com/yungbyun/myml

- ReLU를 이용하여 deep 신경망에서도 역전파 학습이 잘 됨을 보임.

이제는 깊게(deep) 만들 수 있다.

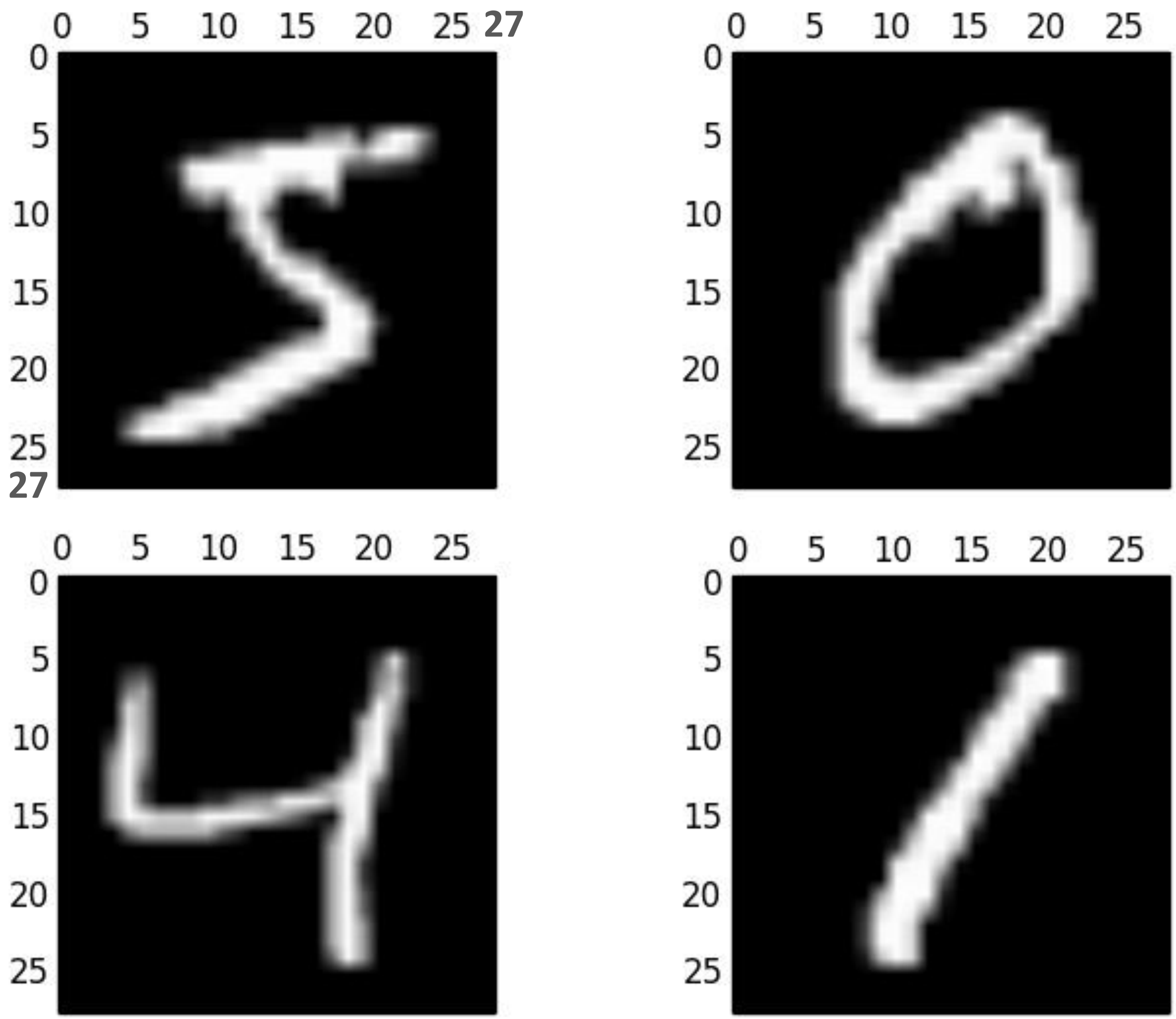
Deep Neural Network
Deep Learning

MNIST

Modified National Institute of Standards and Technology (USA)

MNIST

28 x 28 = 784 픽셀

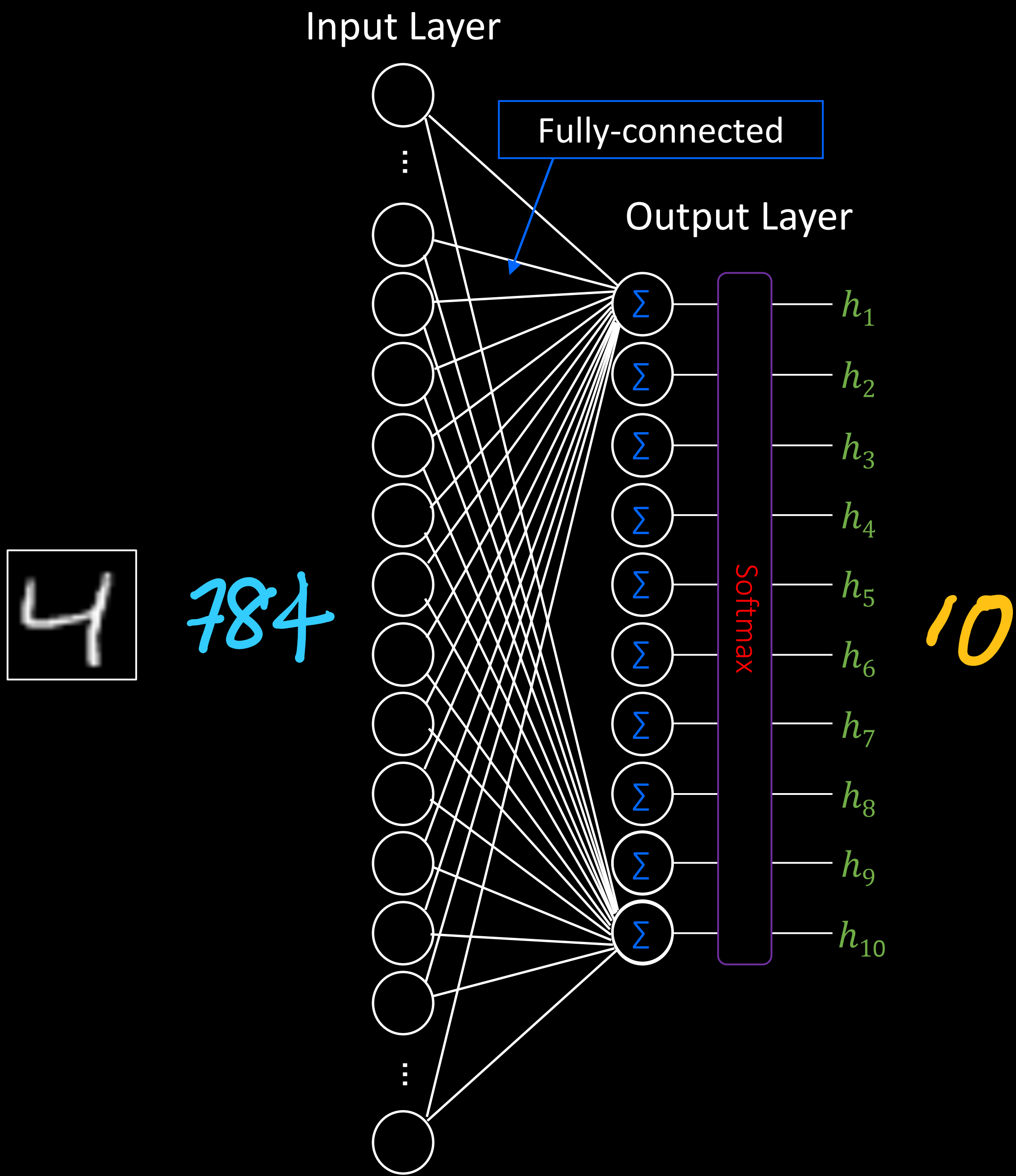


0 5 10 15 20 25 27
0 5 10 15 20 25 27
0 5 10 15 20 25
0 5 10 15 20 25
0 5 10 15 20 25
0 5 10 15 20 25
0 5 10 15 20 25
0 5 10 15 20 25

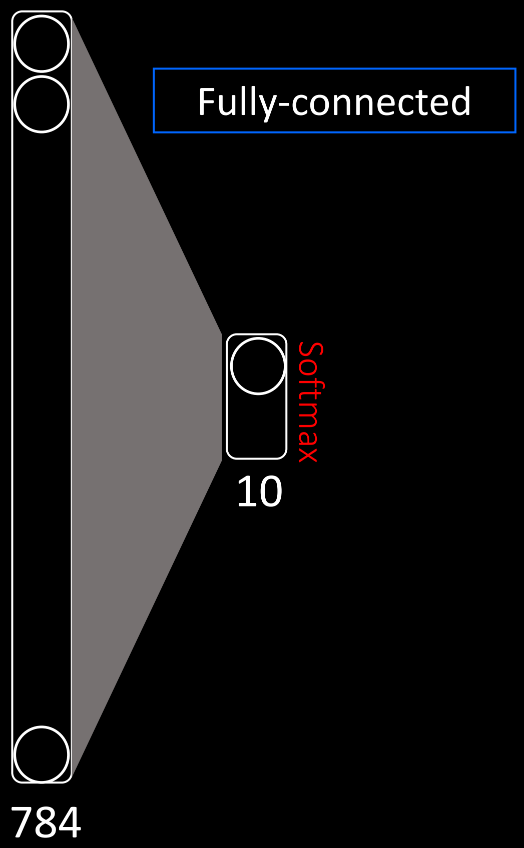
(Lab) 21.py

https://github.com/yungbyun/myml

- 60,000 학습 이미지 + 10,000 테스트 이미지
- 입력이미지 : 28 * 28 픽셀 → 784 픽셀 (차원)
- 10 클래스 (0 ~ 9)
- Softmax
- 90.23% 인식률

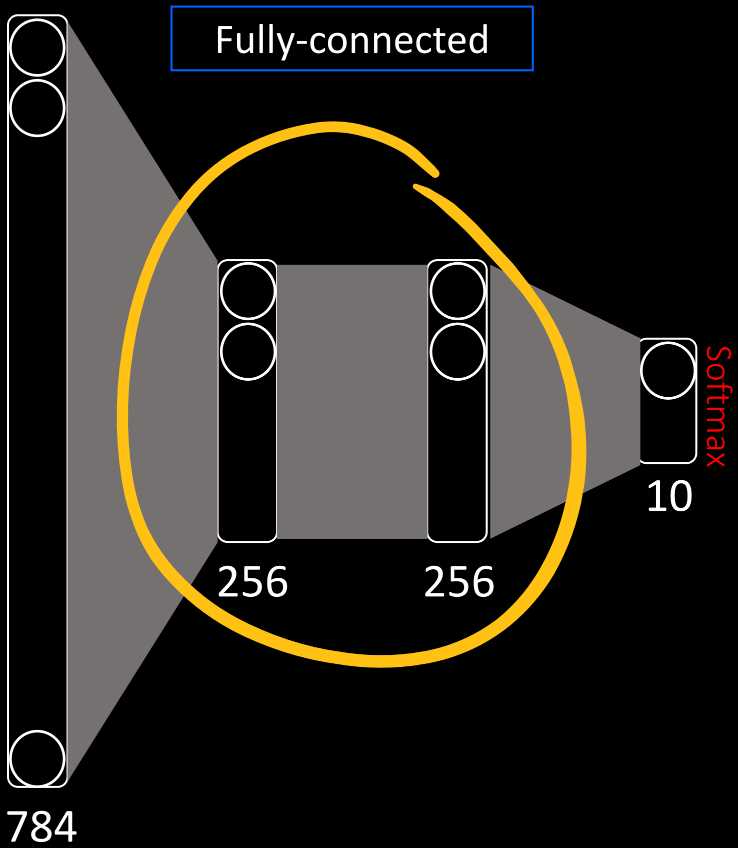


Input Layer
Fully-connected
Output Layer
784
Softmax
h_1
h_2
h_3
h_4
h_5
h_6
h_7
h_8
h_9
h_{10}
10


Fully-connected
Softmax
10
784

(Lab) 22.py

- Deep Neural Network (4-layer)
- ReLU
- 94.55% accuracy



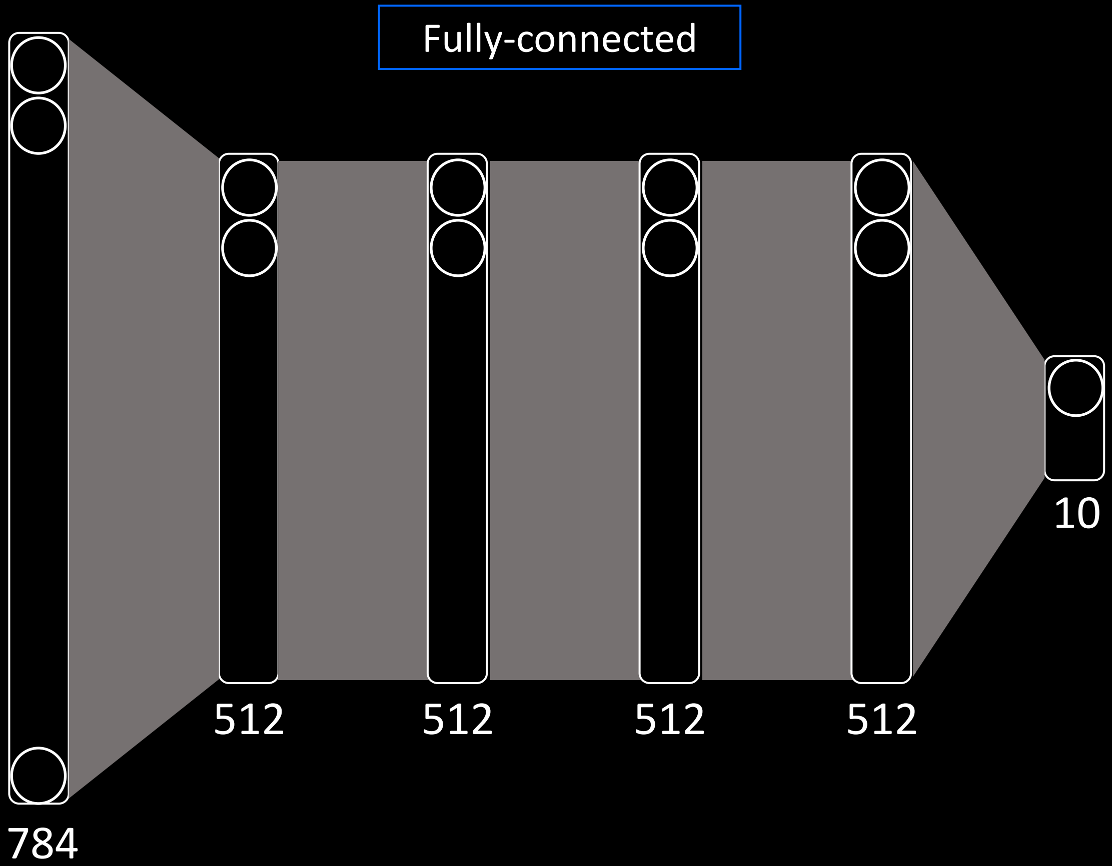
https://github.com/yungbyun/myml

(Lab) 23.py

- 파라미터 w, b 난수 초기화가 아닌 새로운 초기화 방법
- 97.23% 인식률

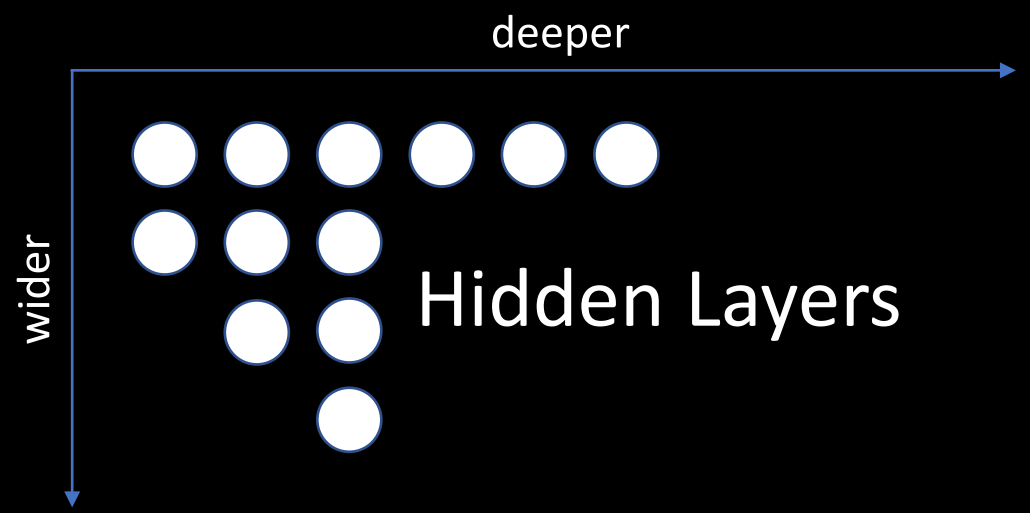
https://github.com/yungbyun/myml

(Lab) 24.py



- 파라미터 w, b 난수 초기화가 아닌 새로운 초기화 방법
- 6-layer deep neural networks
- 97.83% of accuracy

결정 경계

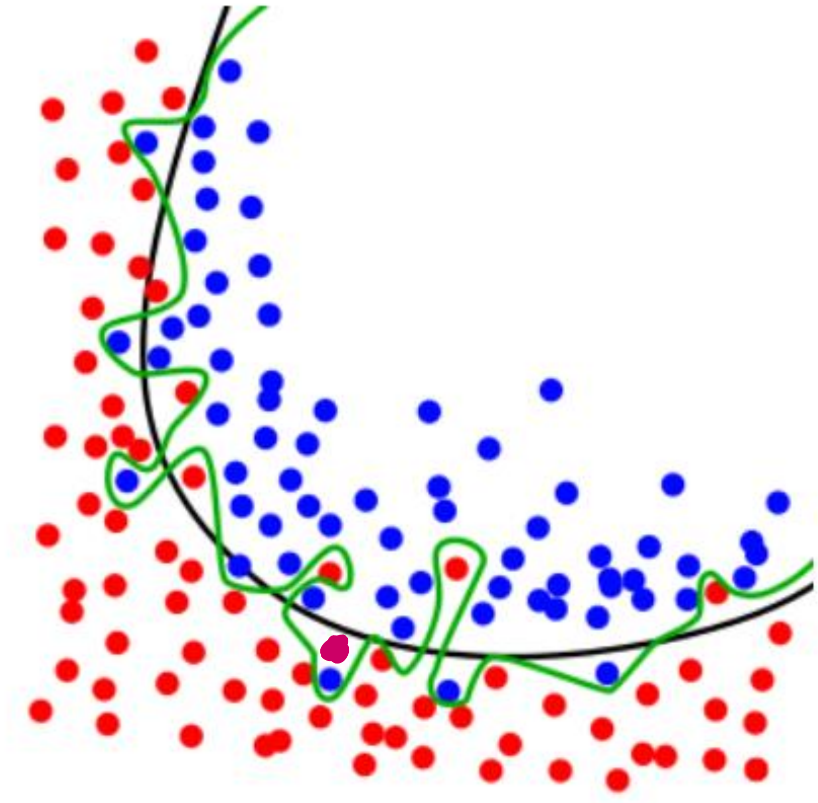


“

수많은 뉴런, 여러 연결로 인한
복잡한 결정경계

초록색과 검정색, 어느 결정경계가 바람직할까?

하늘에서
본 모습



While the black line fits the data well,
the green line is overfit.

https://elitedatascience.com

결정경계 복잡도

뉴런 수가 많으면?
뉴런 수가 적으면?

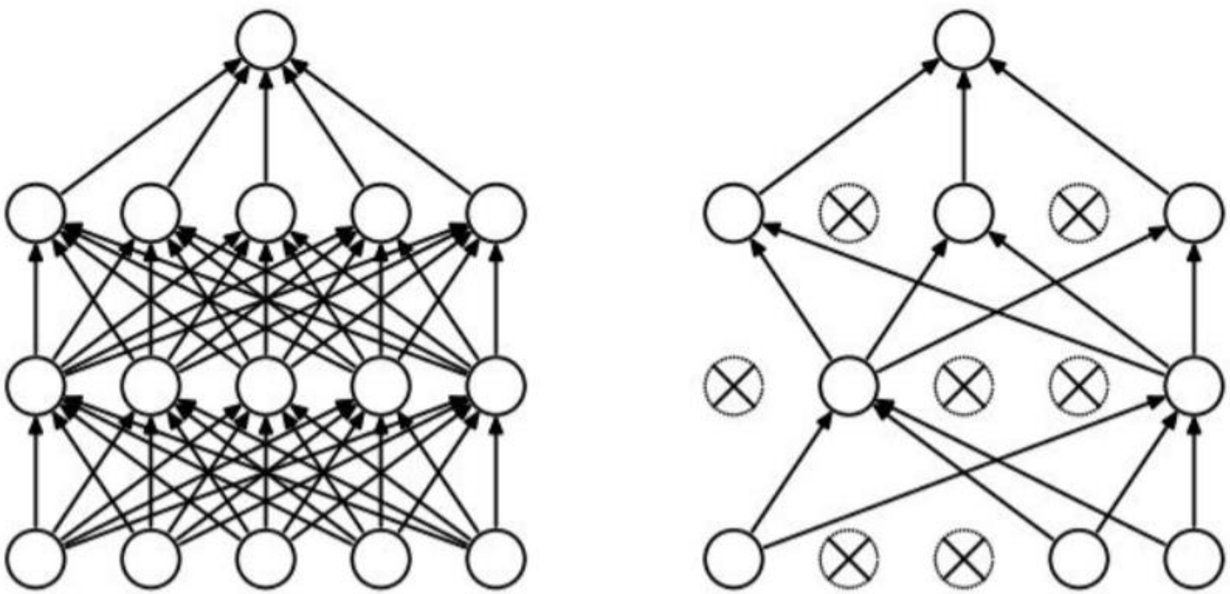
오버 피팅(over-fitting)

- 신경망의 깊이와 너비가 클 수록(deep & wide) 결정 경계는 매우 복잡
- 학습 데이터에 대해서는 지나치게 학습하여 기가 막히게 잘 인식함.
- 하지만 테스트 데이터에 대해서는 에러가 많이 남
- 이를 해결하려면? → 결정경계를 너무 복잡하지 않게 (검정색 결정경계)

신경 세포가 많을 수록(deep & wide) 결정 경계
복잡하므로 학습 시 신경세포를 배제(drop-out)

Regularization: **Dropout**

"randomly set some neurons to zero in the forward pass"



(a) Standard Neural Net

(b) After applying dropout.

[Srivastava et al., 2014]

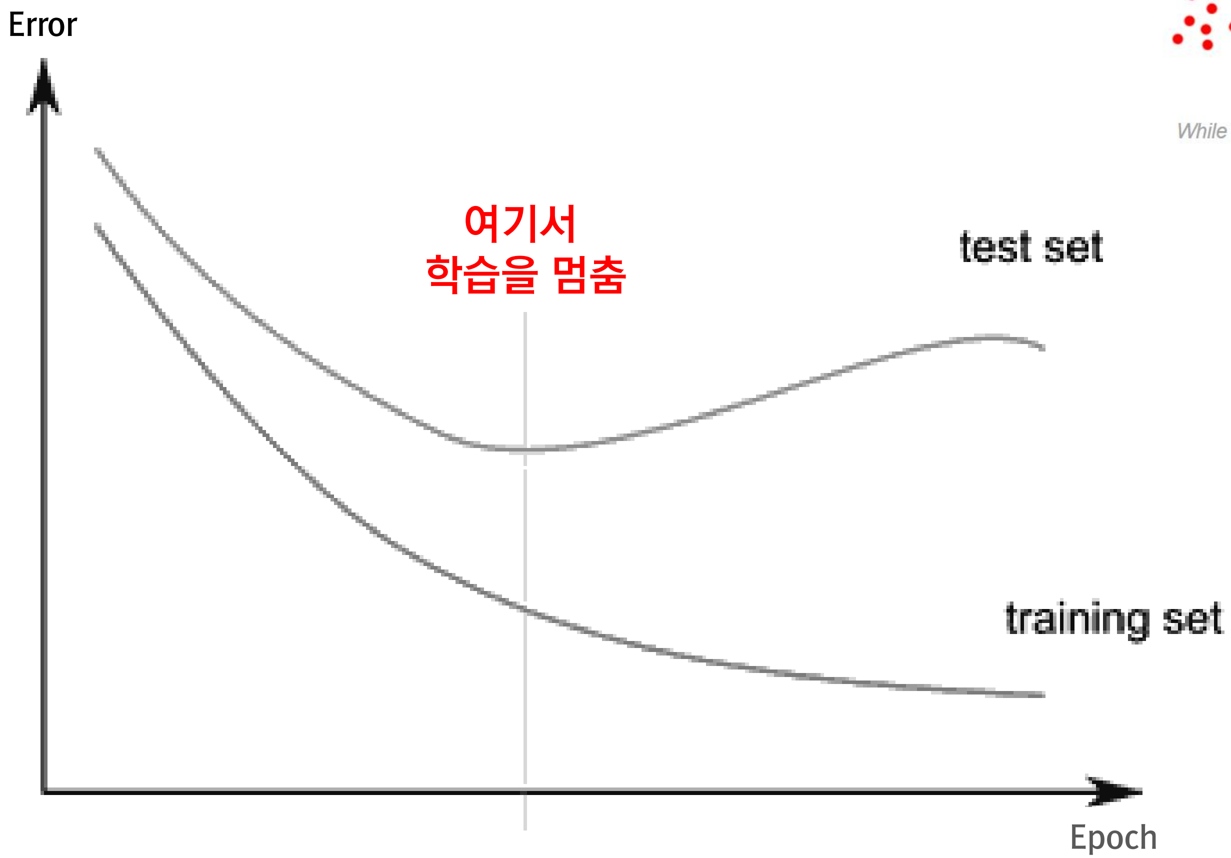
(실습) 25.py

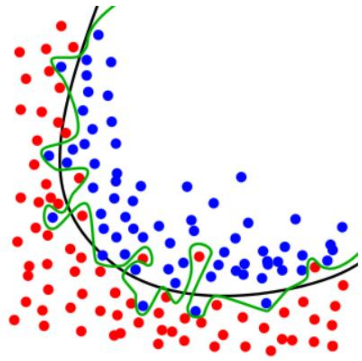
- 시냅스 가중치(w)와 바이어스 (b)를 적절히 초기화
- Deeper (DNN) -> 6개 층
- Dropout
- 98.13% of accuracy

How to prevent overfitting

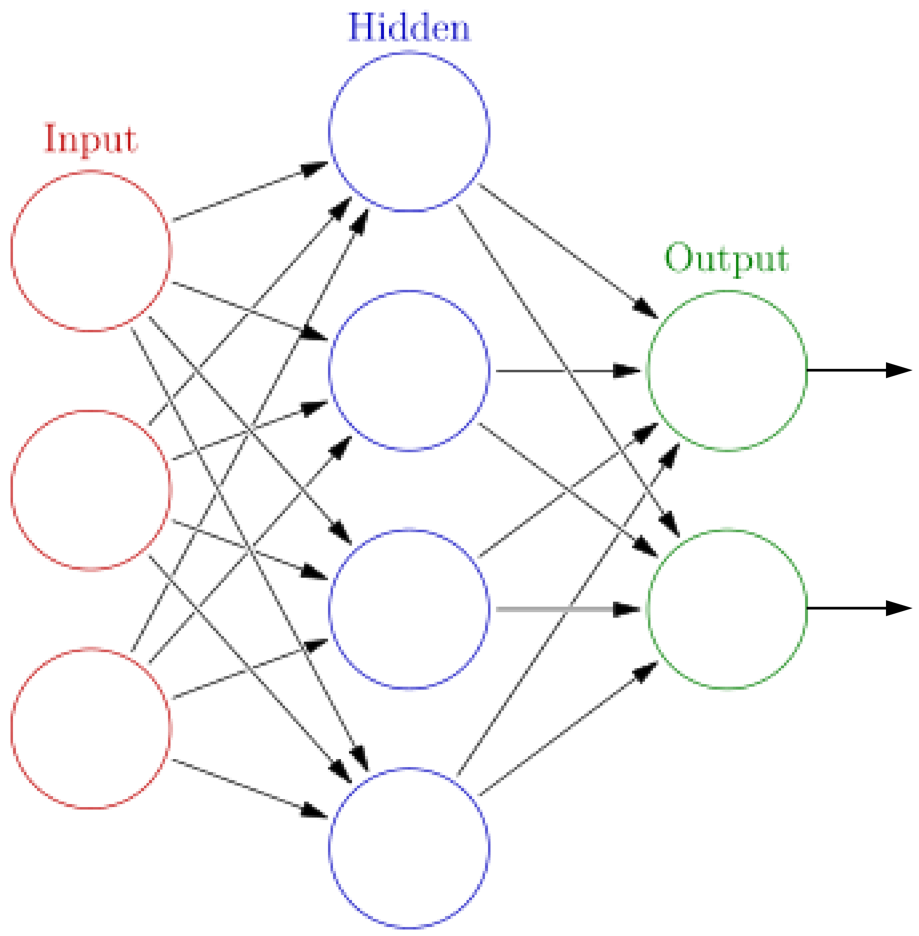
- Train with more data
- Reduce features
- Early stopping
- Ensemble
- Regularization

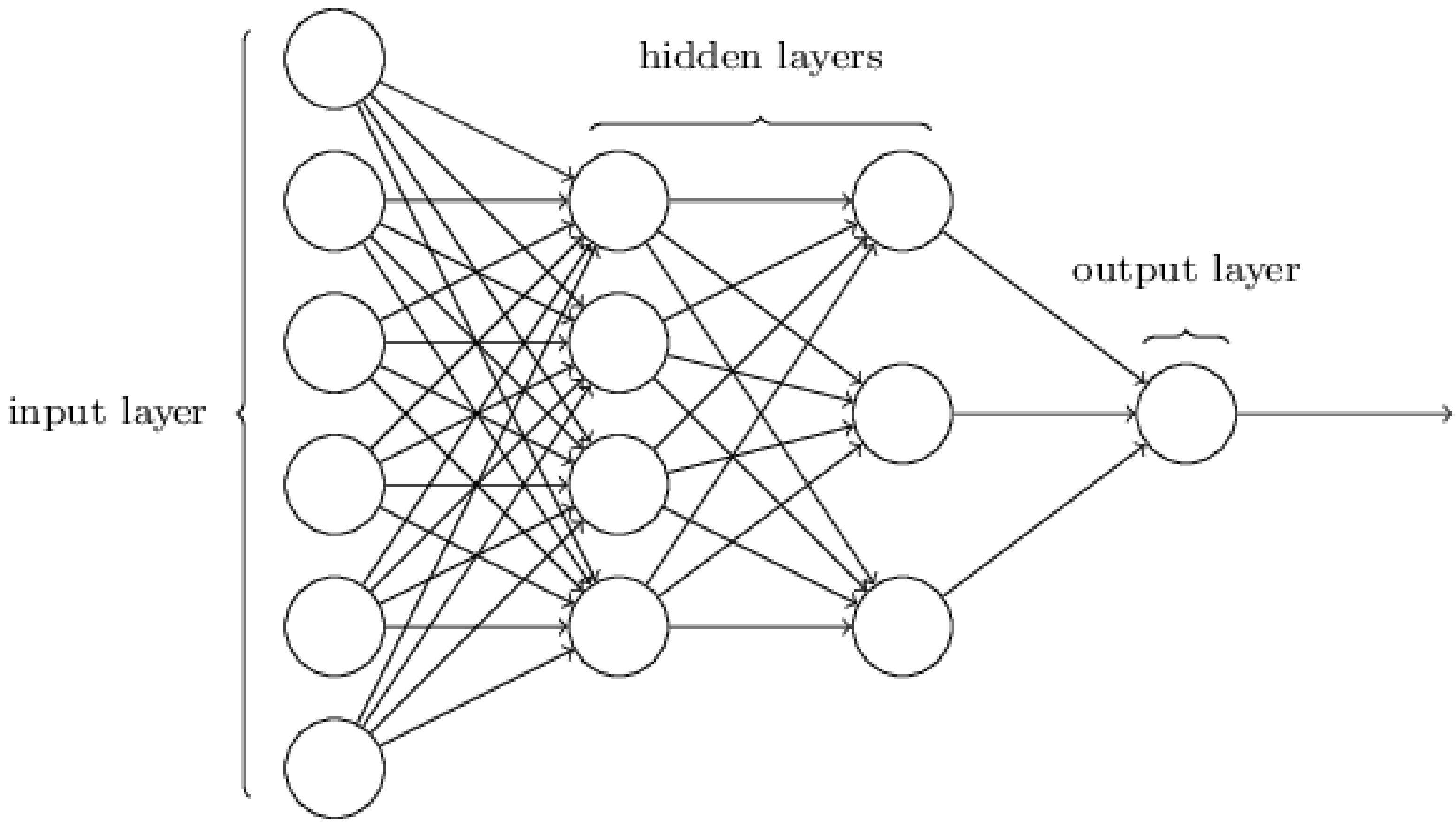
Early stopping



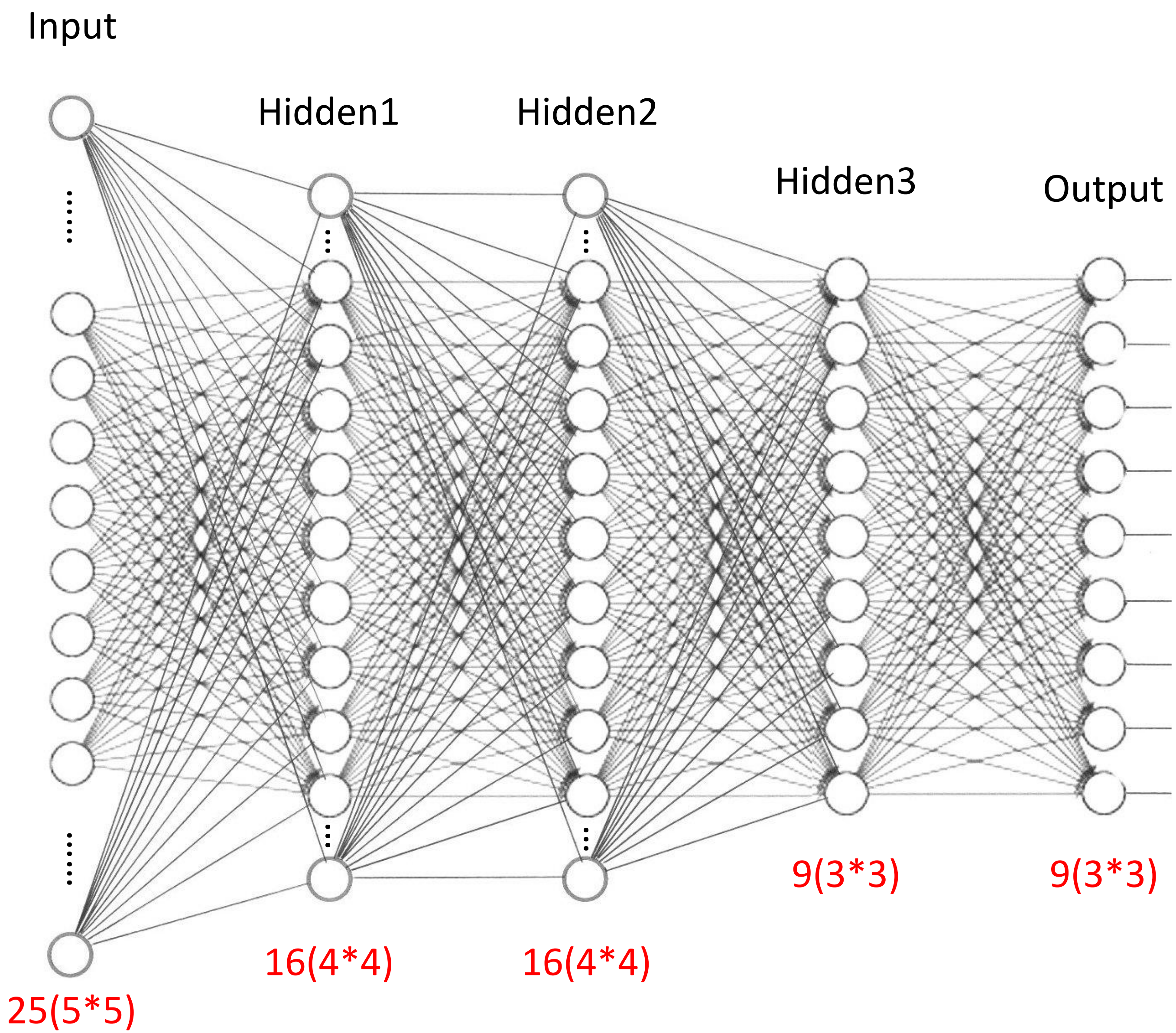


While the black line fits the data well,
the green line is overfit.


Input
Hidden
Output



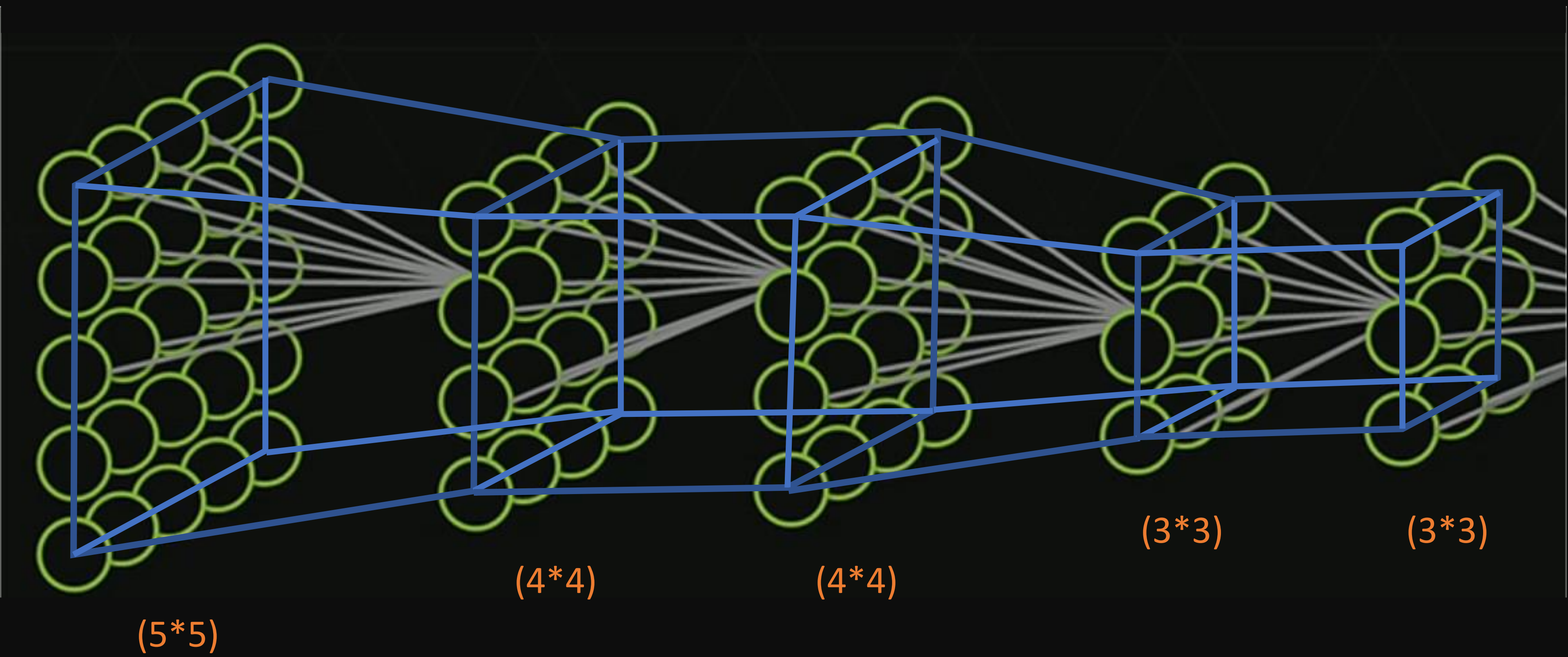
input layer
hidden layers
output layer



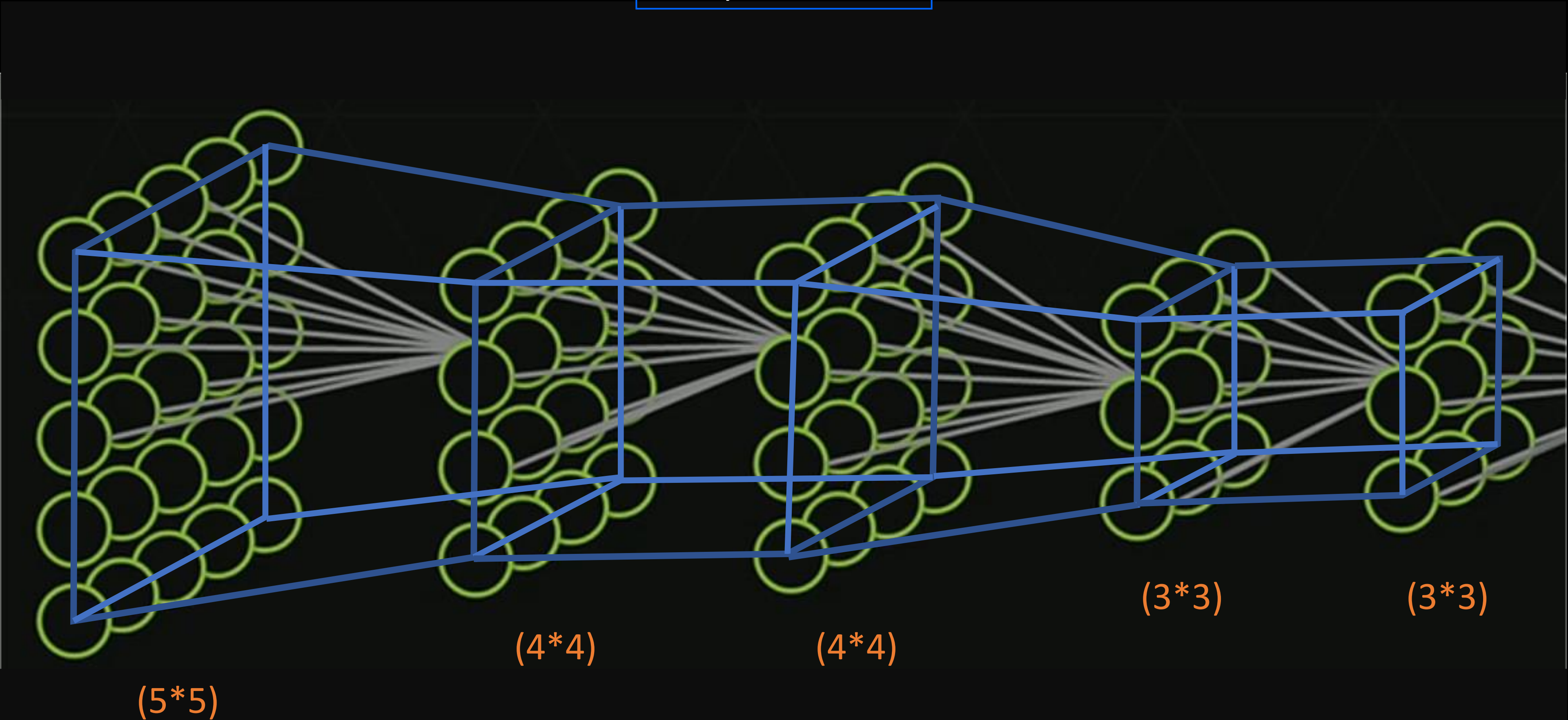
Fully connected, so how many connections(parameters) are there?

25 * 16 + 16 * 16 + 16 * 9 + 9 * 9 = 881

Fully-connected

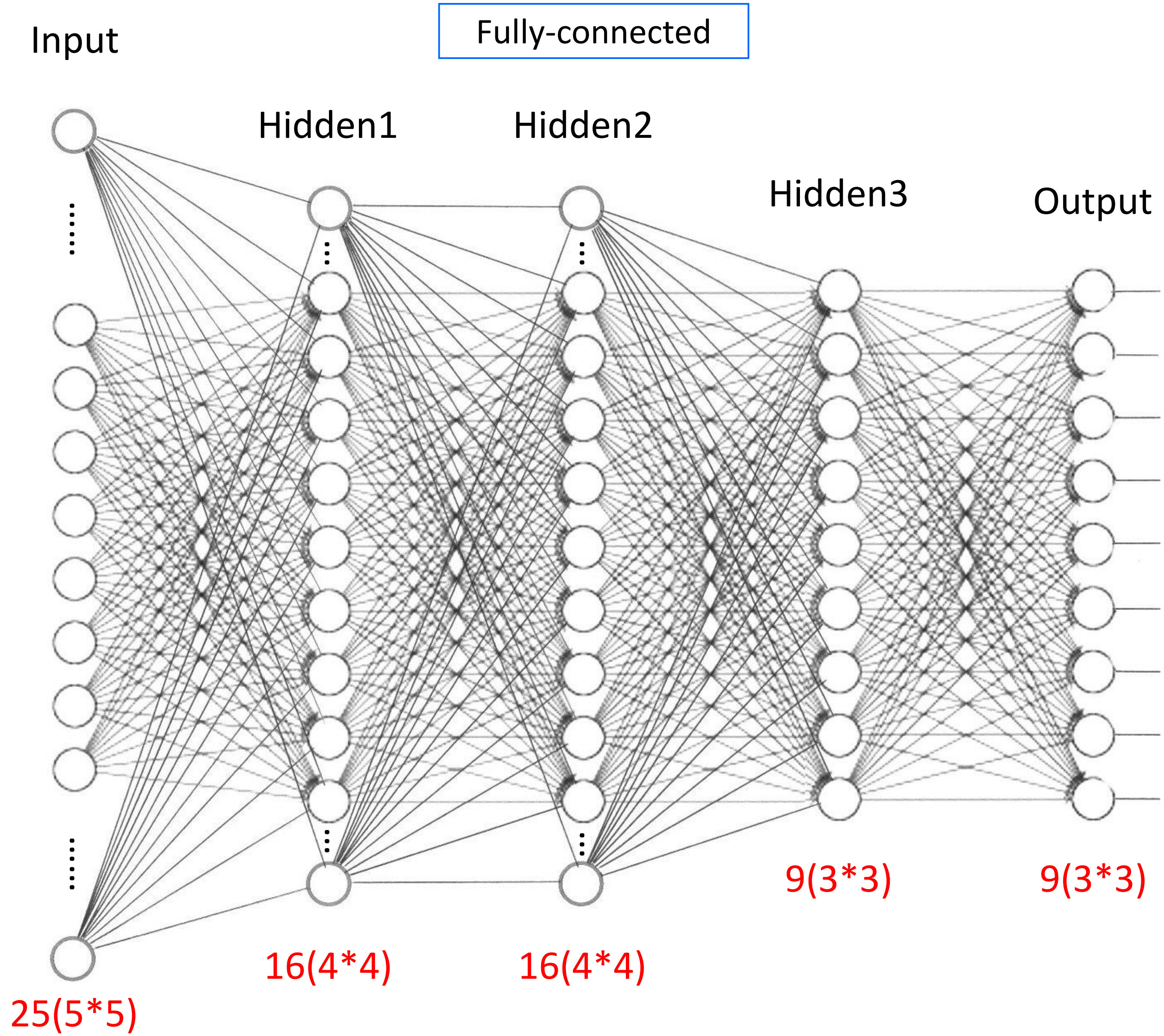


Fully-connected



Fully connected, so how many connections are there?

25 * 16 + 16 * 16 + 16 * 9 + 9 * 9 = 881



Fully connected, then how many connections(synapses, parameters) are there?

25 * 16 + 16 * 16 + 16 * 9 + 9 * 9 = 881



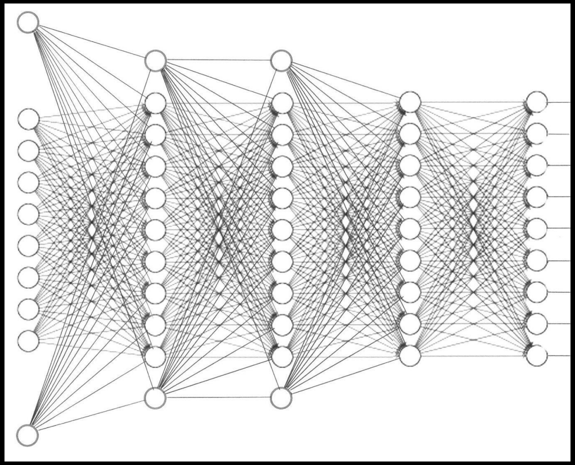

Geoffrey Hinton, Yann LeCun, Yoshua Bengio, Andrew Ng

Google Baidu百度 facebook IBM

Deep Learning

- in early 2000s (2006, 2010, 2012)
- Deep Neural Networks
- Weight initialization methods
- Activation functions (ReLU)
- Dropout (2014)
- Big data
- GPU

Fully-connected



FCNN

Any problem?