AI and Deep Learning

Deep Learning

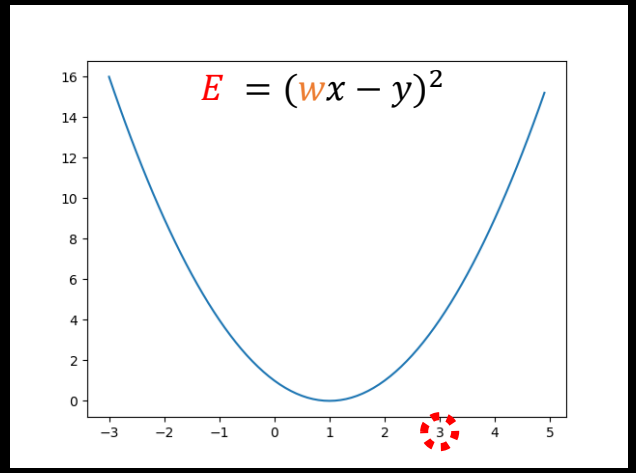
Jeju National University
Yung-Cheol Byun

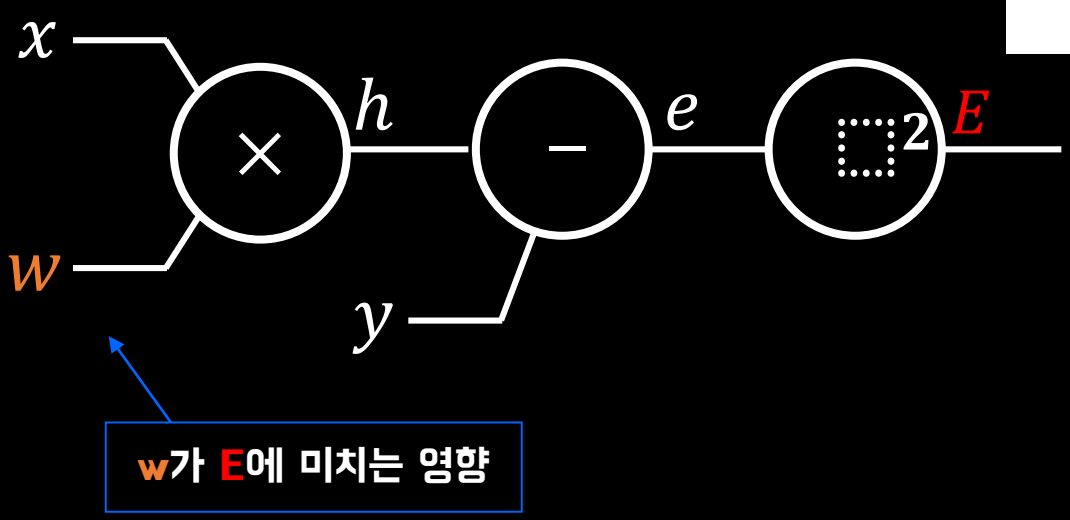
```
#----- a neuron
w = tf.Variable(tf.random_normal([1]))
hypo = w * x_data

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

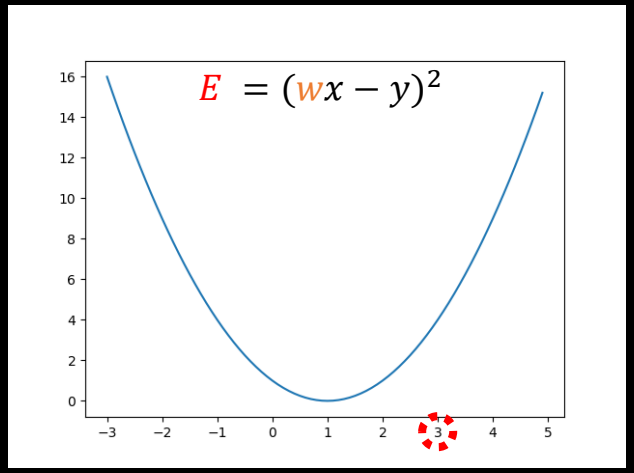
$$E = (wx - y)^2$$

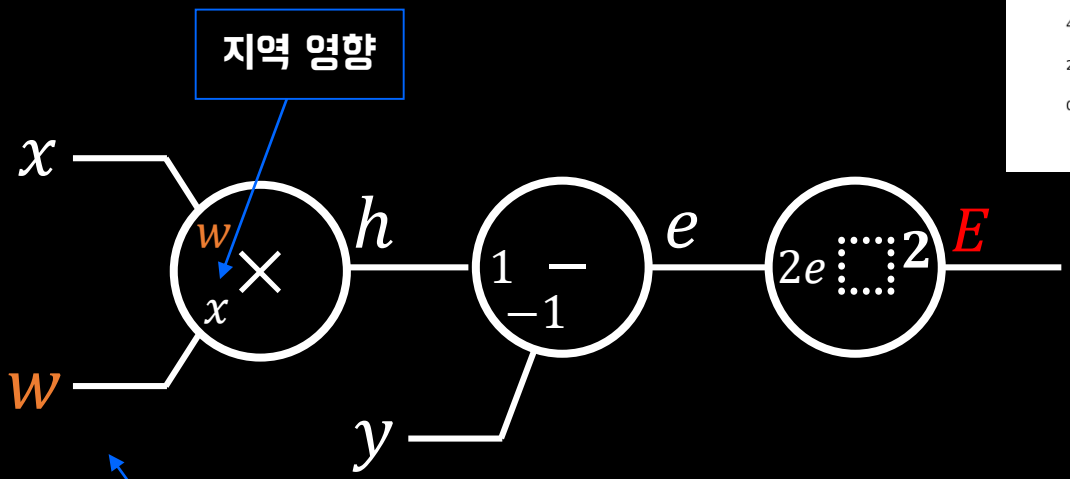
오류 계산 그래프





오류 계산 그래프

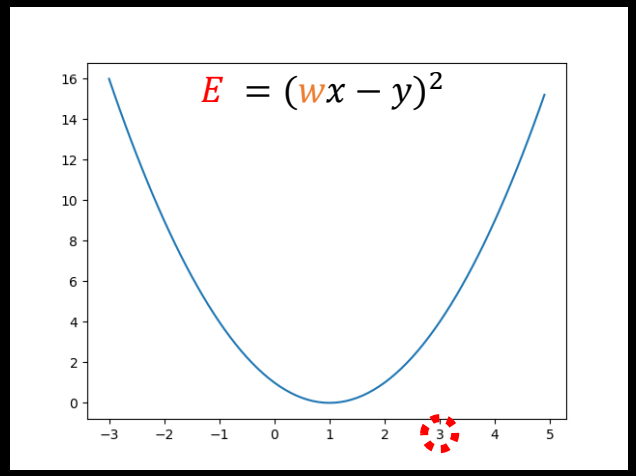


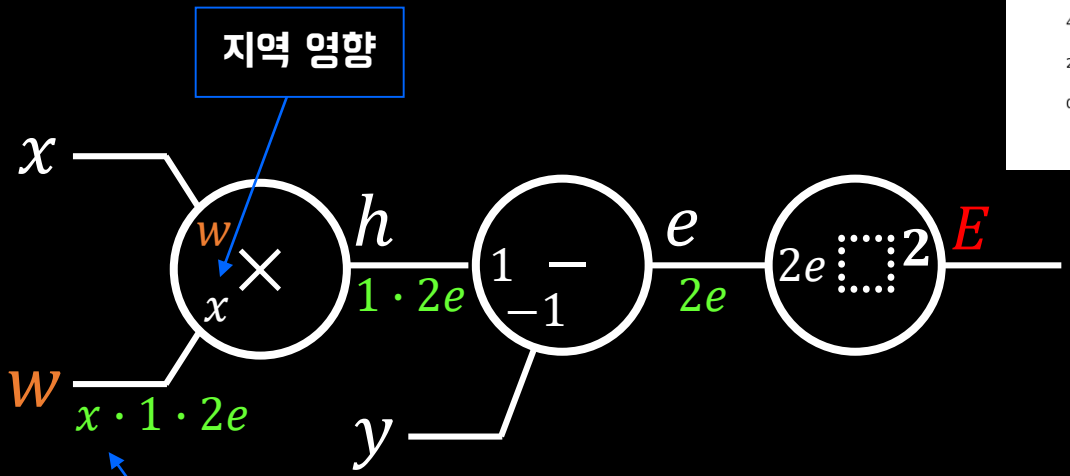


w가 E에 미치는 영향
= 모든 지역 영향을 곱함 (체인룰)
= w의 지역 영향 * h가 E에 미치는 영향

$x * 1 * 2e$

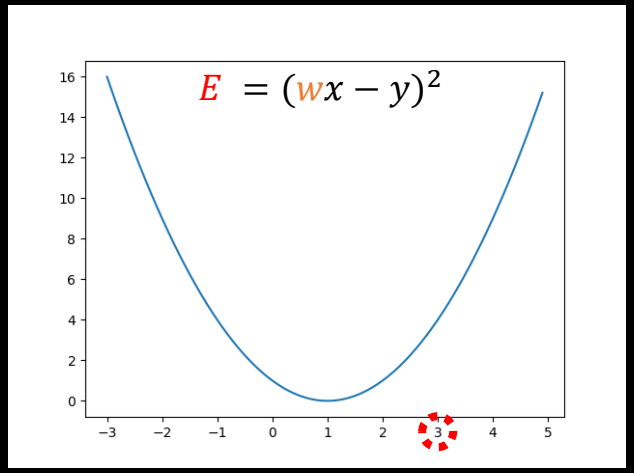
오류 계산 그래프

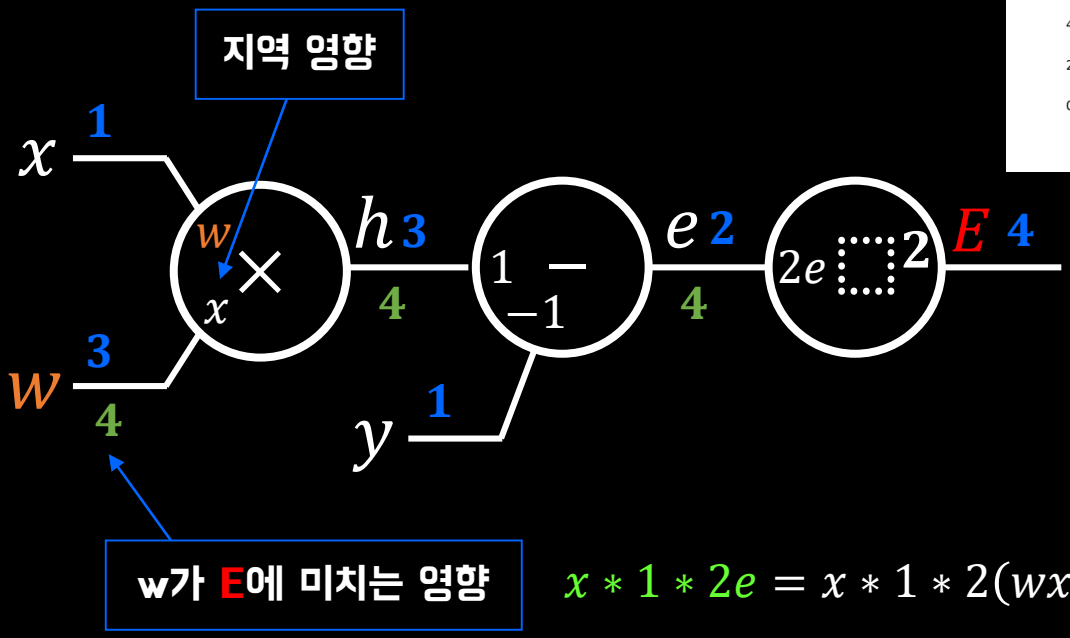




w가 E에 미치는 영향
= 모든 지역 영향을 곱함 (체인룰)
= w의 지역 영향 * h가 E에 미치는 영향

오류 계산 그래프



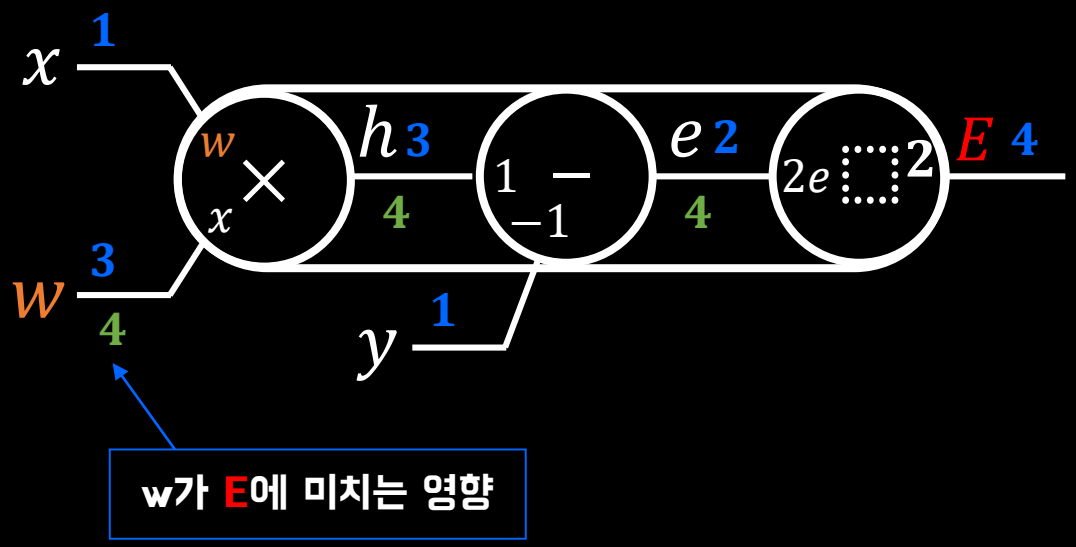


$x * 1 * 2e = x * 1 * 2(wx - y) = 2x(wx - y)$

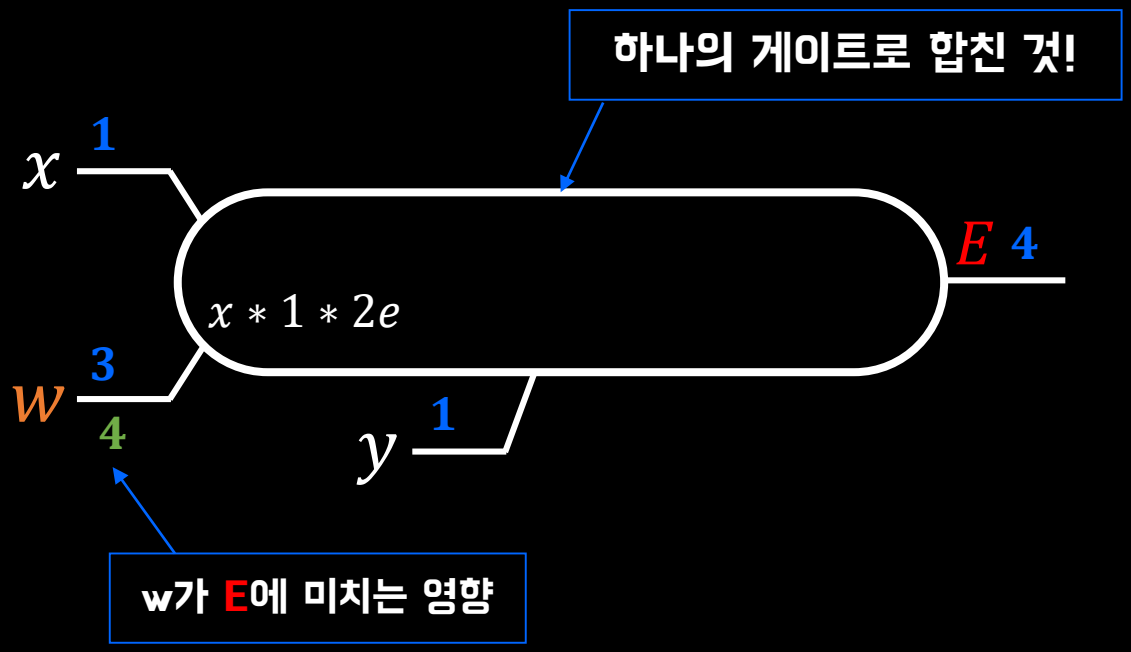
오류 계산 그래프

- 지역 영향(local gradient)는 어떻게 구했나?
- 입력을 1만큼 증가해서 출력이 얼마나 증가하는지 확인해 봄.
- 또는 직접 미분할 수도 있음.

오류 계산 그래프



오류 계산 그래프



※$(x+y)^2 = x^2 + 2xy + y^2$

$E = (wx - y)^2$를 미분하면?

$(wx - y)^2 = w^2x^2 - 2wxy + y^2$

미분결과 : $2wx^2 - 2xy = 2x(wx - y)$

$x * 1 * 2e = x * 1 * 2(wx - y) = 2x(wx - y)$

계산 그래프 또는 미분

- cost(E) 계산 그래프를 그린 후 w가 E에 미치는 영향 구하거나
- 또는 cost(E)를 직접 w에 대하여 미분하여 값을 구하거나

오류 계산 그래프

로지스틱 리그레션
신경 세포 1개만 있을 때

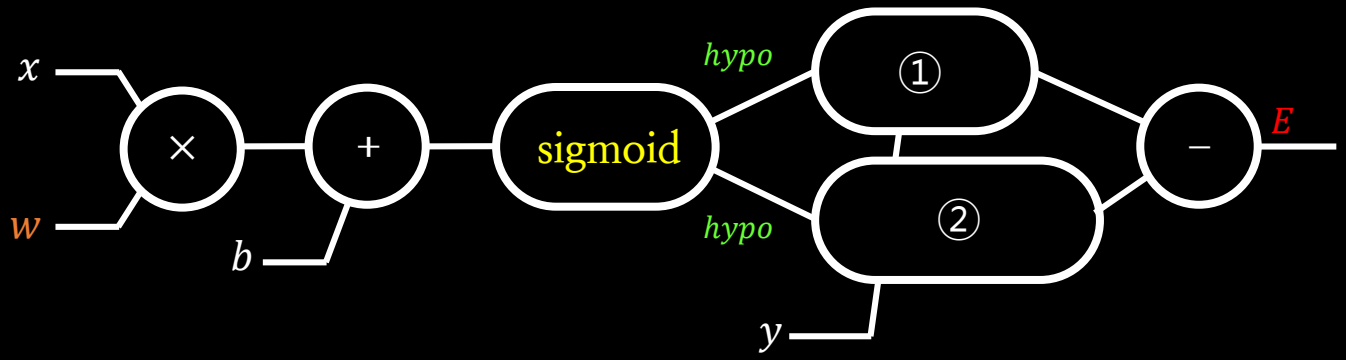
$$cost(E) = -y\log(hypo) - (1-y)\log(1-hypo)$$

$$hypo = \frac{1}{1+e^{-wx}}$$

w가 cost에 미치는 영향은?
w를 아주 조금 바꿨을 때 cost는 어떻게 변하나?
계산 그래프를 그리거나 또는 직접 미분하거나...

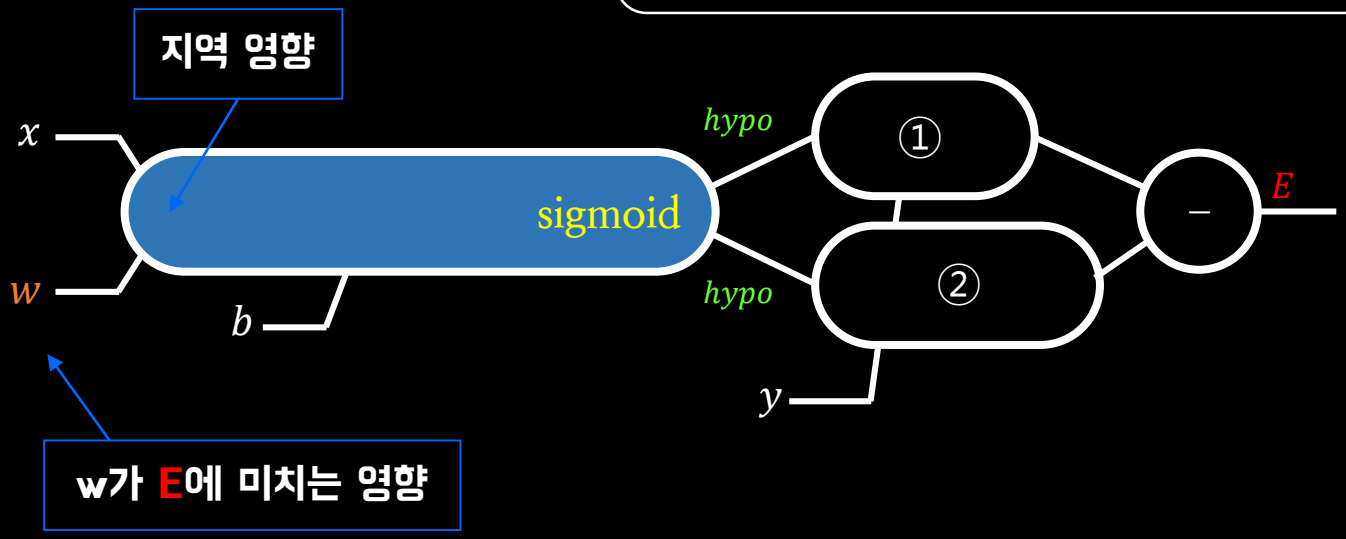
오류 계산 그래프

$$E = -y \cdot \log(hypo) - \underbrace{(1-y) \cdot \log(1-hypo)}_{②}$$

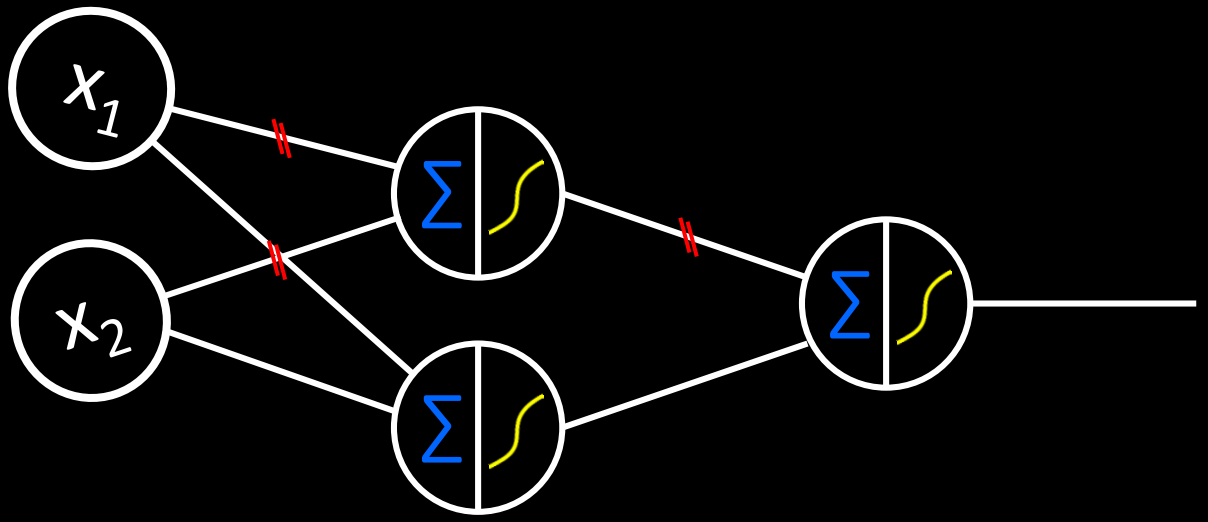


오류 계산 그래프

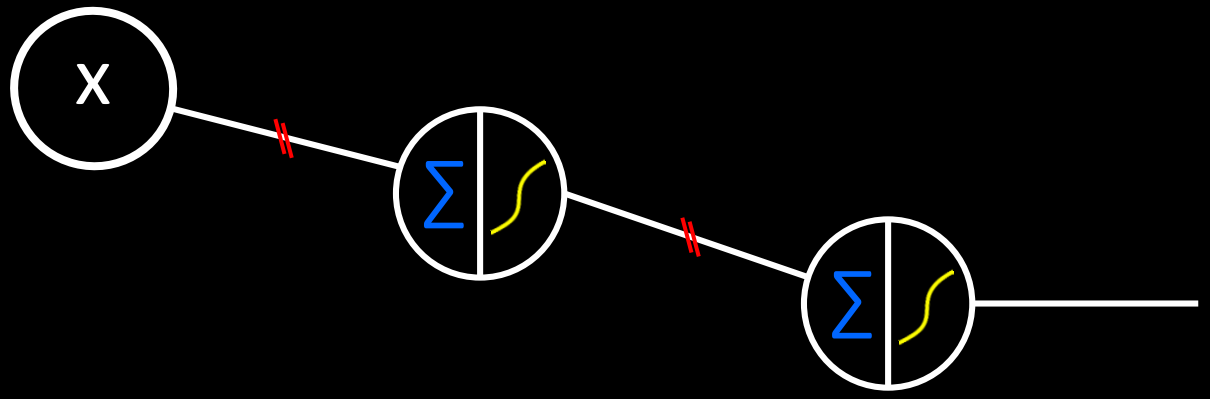
$$E = \underset{①}{-y \cdot \log(hypo)} \underset{②}{-(1-y) \cdot \log(1-hypo)}$$



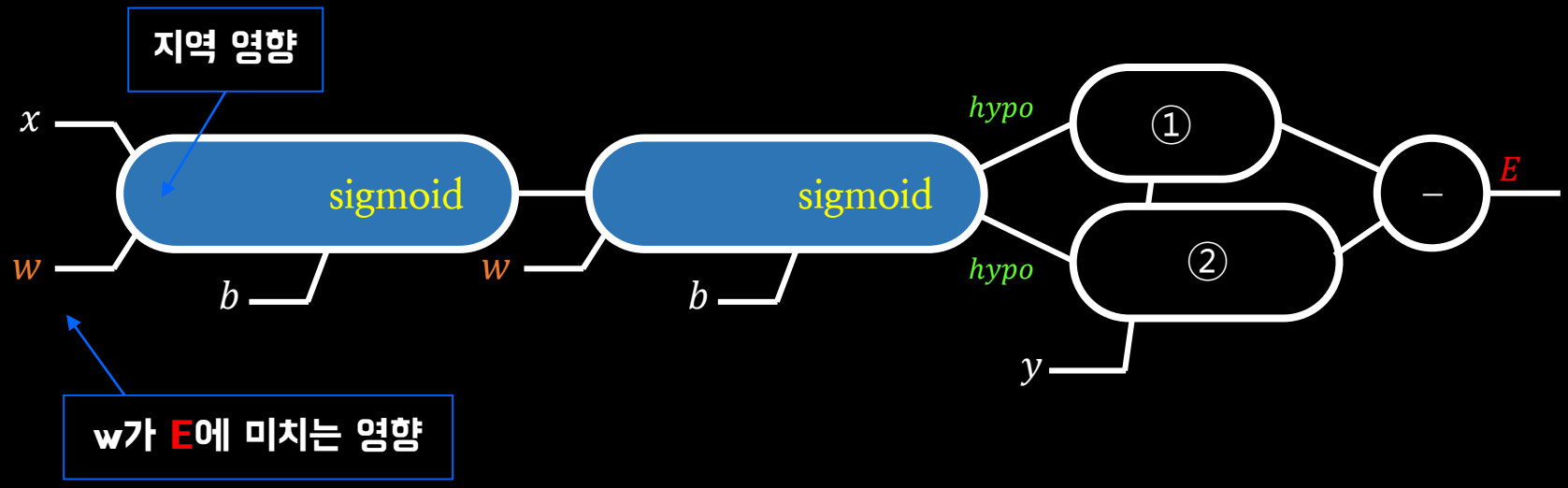
오류 계산 그래프



오류 계산 그래프



오류 계산 그래프



사라지는 영향력 문제

- 수많은 뉴런이 연결
- 그 안에 들어 있는 sigmoid(logistic) function들
- local gradient? logistic function을 미분한 것 (1-logistic(x)) * logistic(x)
- 왼쪽의 w가 오류에 미치는 영향은?
- 모든 local gradient 를 곱한 것(체인룰)

사라지는 영향력 문제

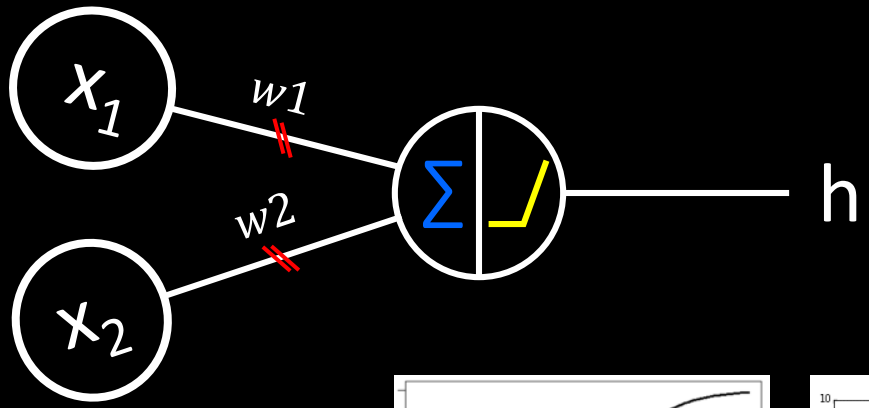
- 왼쪽의 w는 수많은 1 이하 수의 곱
- 오른쪽 w는 큰 문제가 없음
- 왼쪽으로 갈 수록 w가 E에 미치는 영향(기울기=gradient)은 매우 작음.
- 즉, 영향력이 사라짐 (Vanishing Gradient)
- w = w - α * (기울기)
- 따라서 w, b가 거의 갱신이 되지 않음

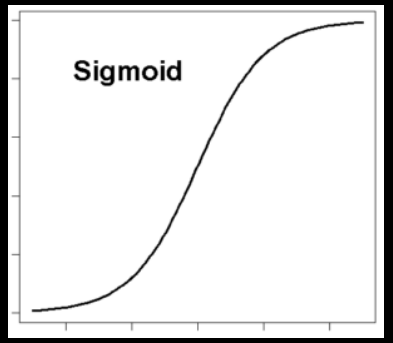
(실습) 18.py

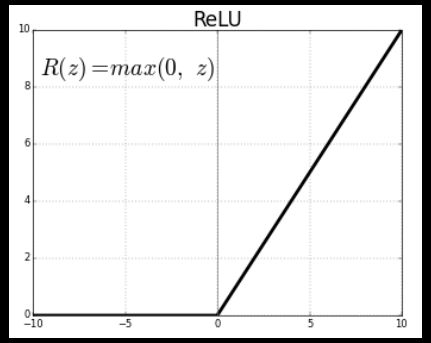
- 4층으로 구성된 신경망으로 XOR 문제를 해결하고자 했으나 Vanishing Gradient 때문에 실패

ReLU

Logistic 함수 대신 ReLU를 사용함으로써 Vanishing Gradient 문제 해결







(실습) 19.py

- ReLU를 이용하여 deep 신경망에서도 역전파 학습이 잘 됨을 보임.

MNIST

Modified National Institute of Standards and Technology (USA)

MNIST


210: 9 → 7
448: 4 → 9
583: 8 → 3
692: 8 → 9
717: 1 → 7
948: 8 → 9
1034: 8 → 0
1113: 4 → 6
1227: 7 → 2
1248: 9 → 5
1300: 5 → 7
1320: 8 → 3
1531: 8 → 7
1682: 3 → 7
1710: 9 → 5
1791: 2 → 7
1879: 8 → 3
1902: 9 → 4
2041: 5 → 6
2074: 5 → 6
2099: 2 → 0
2131: 4 → 9
2183: 1 → 2
2238: 5 → 6
2448: 4 → 9
2463: 2 → 0
2583: 9 → 7
2598: 5 → 3
2655: 6 → 1
2772: 4 → 9
2940: 9 → 7
3063: 8 → 6
3074: 1 → 2
3251: 2 → 6
3423: 6 → 0
3476: 3 → 7
3559: 5 → 0
3822: 9 → 4
3851: 9 → 4
4094: 9 → 7
4164: 9 → 7
4202: 1 → 7
4370: 9 → 4
4498: 8 → 7
4506: 9 → 7
4663: 9 → 7
4732: 8 → 9
4762: 9 → 4
5730: 5 → 3
5938: 5 → 3
6555: 2 → 7
6572: 9 → 7
6577: 7 → 1
6598: 0 → 7
6884: 1 → 2
8066: 8 → 0
8280: 8 → 4
8317: 7 → 2
8528: 4 → 9
9506: 7 → 2
9643: 9 → 7
9730: 5 → 6
9851: 0 → 6

MNIST


4 → 9
116
2 → 7
322
5 → 3
341
6 → 0
446
3 → 5
450
8 → 2
583
4 → 9
741
3 → 5
939
8 → 9
948
1 → 3
957
6 → 0
1015
4 → 6
1113
7 → 2
1227
9 → 4
1233
9 → 5
1248
7 → 1
1261
8 → 2
1365
5 → 3
1394
8 → 7
1531
9 → 5
1710
5 → 2
1738
8 → 3
1879
9 → 4
1902
5 → 3
2035
4 → 9
2054
2 → 0
2099
6 → 0
2119
4 → 9
2131
9 → 8
2294
9 → 1
2388
9 → 4
2407
2 → 0
2463
2 → 4
2489
5 → 3
2598
6 → 1
2655
8 → 0
2897
9 → 5
2940
3 → 5
2954
6 → 0
3031
8 → 9
3290
6 → 0
3423
6 → 4
3521
5 → 0
3559
8 → 9
3728
4 → 6
3781
7 → 8
3808
1 → 3
3907
7 → 1
4028
9 → 8
4079
1 → 7
4202
2 → 7
4206
9 → 7
4225
7 → 3
4239
2 → 8
4249
9 → 5
4285
3 → 2
4444
6 → 8
4572
3 → 5
4741
9 → 8
4762
8 → 0
4808
6 → 0
4815
8 → 6
4880
8 → 2
5750
5 → 3
5938
3 → 8
5956
3 → 0
6012
8 → 9
6556
1 → 8
6573
7 → 1
6577
8 → 3
6626
2 → 1
8060
8 → 5
8409
7 → 2
9010
7 → 2
9016
4 → 9
9680
9 → 7
9693
5 → 6
9730
4 → 9
9793
2 → 7
9840

(실습) 20.py

- 입력으로 주어지는 이미지는 28 * 28 = 784 픽셀
- 따라서 784 차원 입력
- 클래스가 10개(0~9)이므로 신경세포 수는 10개
- Softmax

(실습) 21.py

- Deep Neural Network (4층)
- ReLU

(실습) 22.py

- 시냅스 가중치(W)와 바이어스(b)를 적절히 초기화

(실습) 23.py

- 시냅스 가중치(W)와 바이어스(b)를 적절히 초기화
- More Deep (DNN) -> 6개 층

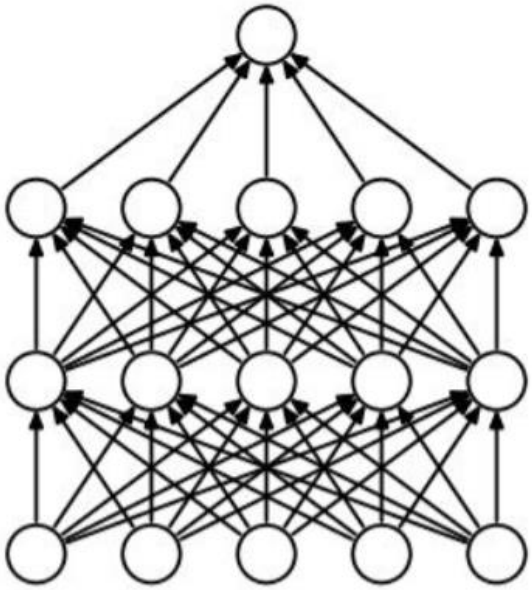
오버 피팅(over-fitting)

- 신경망의 깊이와 너비가 클 수록(deep & wide) 결정 경계는 매우 복잡
- 학습 데이터에 대해 지나치게 학습하여 기가 막히게 잘됨
- 하지만 테스트 데이터에 대해서는 에러가 많이 남 -> 오버 피팅

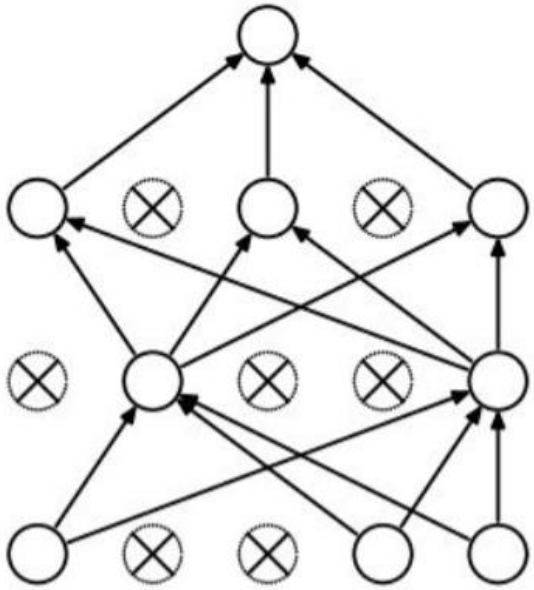
신경 세포가 많을 수록(deep & wide) 결정 경계
복잡하므로 학습 시 신경세포를 배제(drop-out)

Regularization: **Dropout**

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



(a) Standard Neural Net

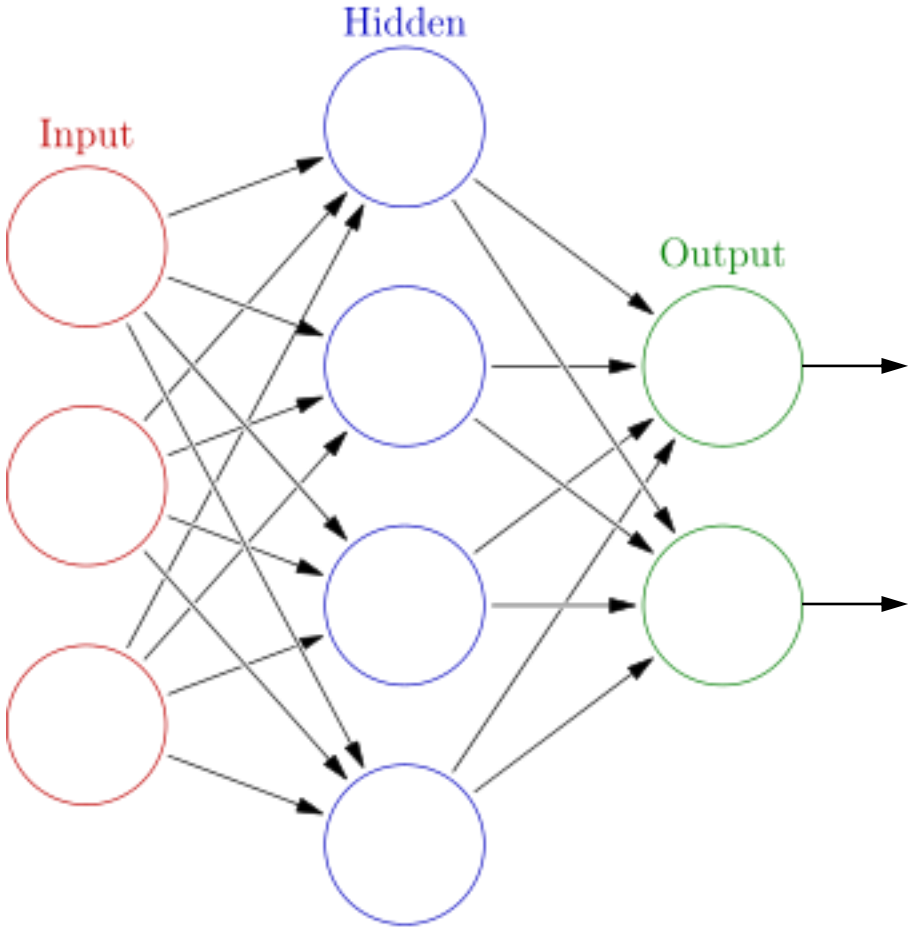


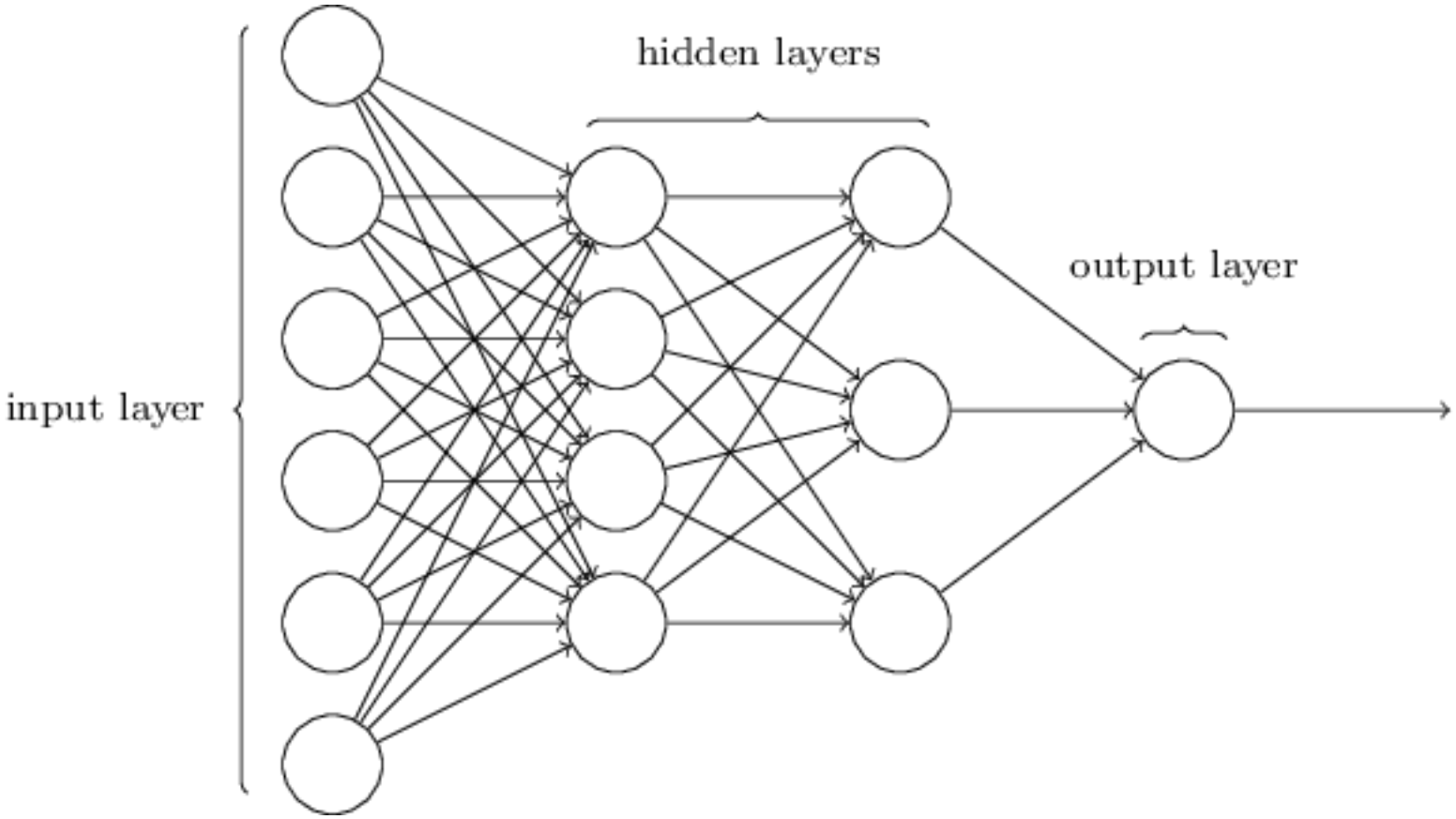
(b) After applying dropout.

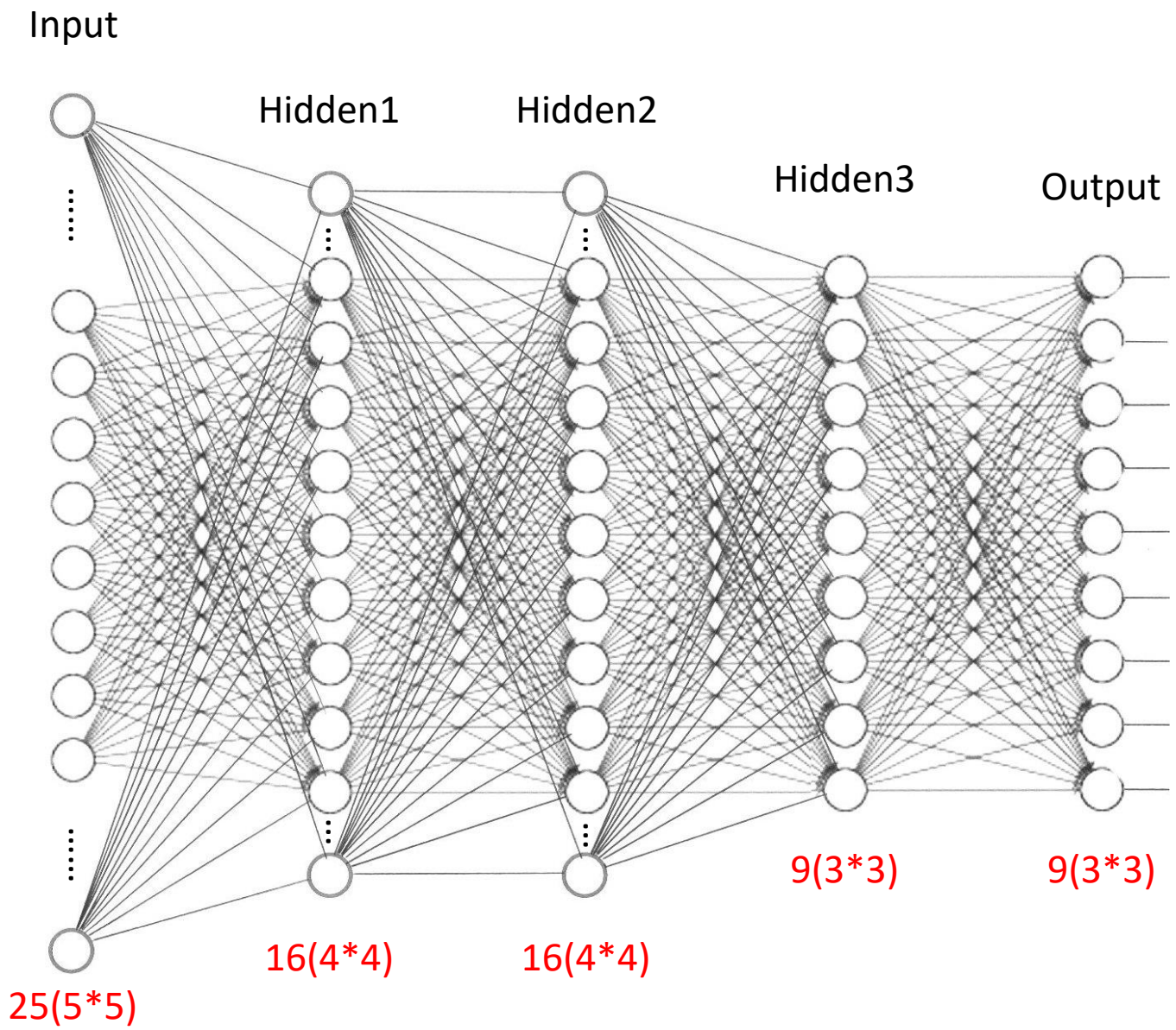
[Srivastava et al., 2014]

(실습) 24.py

- 시냅스 가중치(W)와 바이어스(b)를 적절히 초기화
- More DNN -> 6개 층

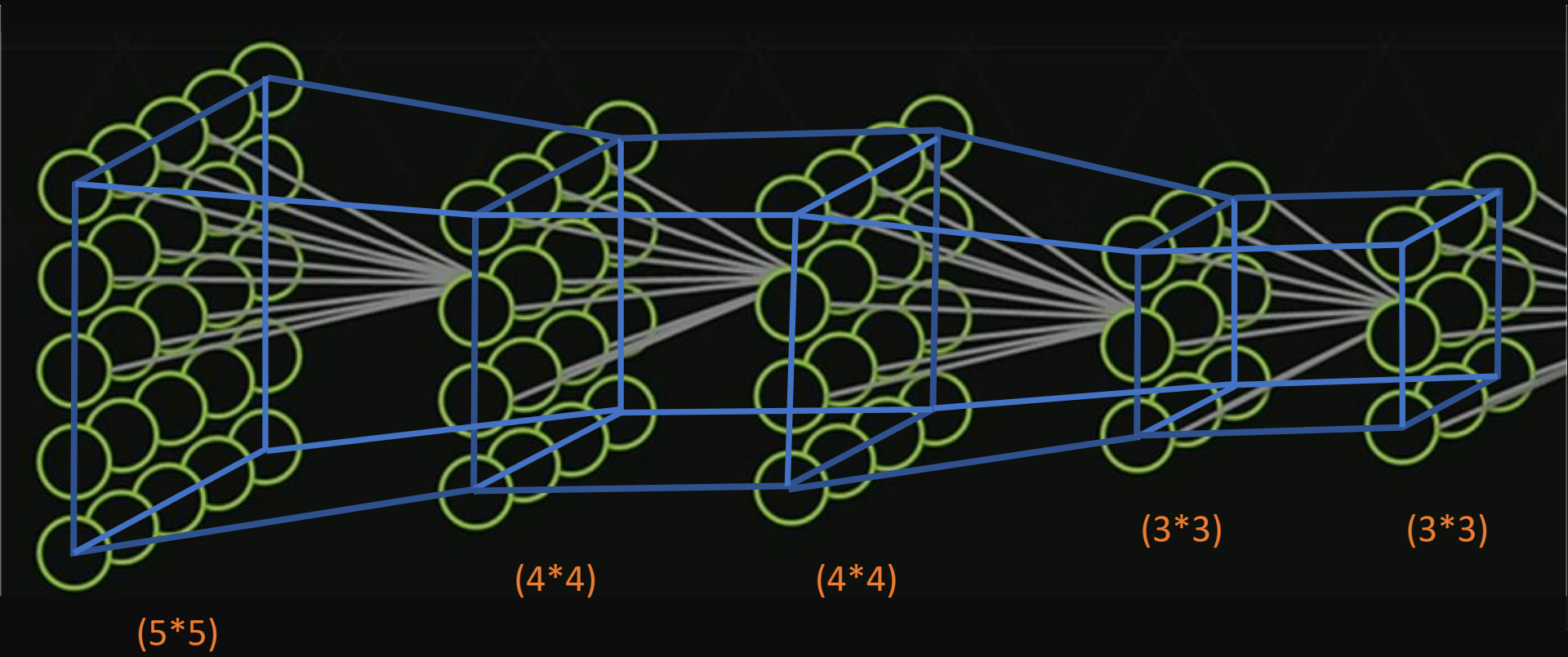

Hidden
Input
Output

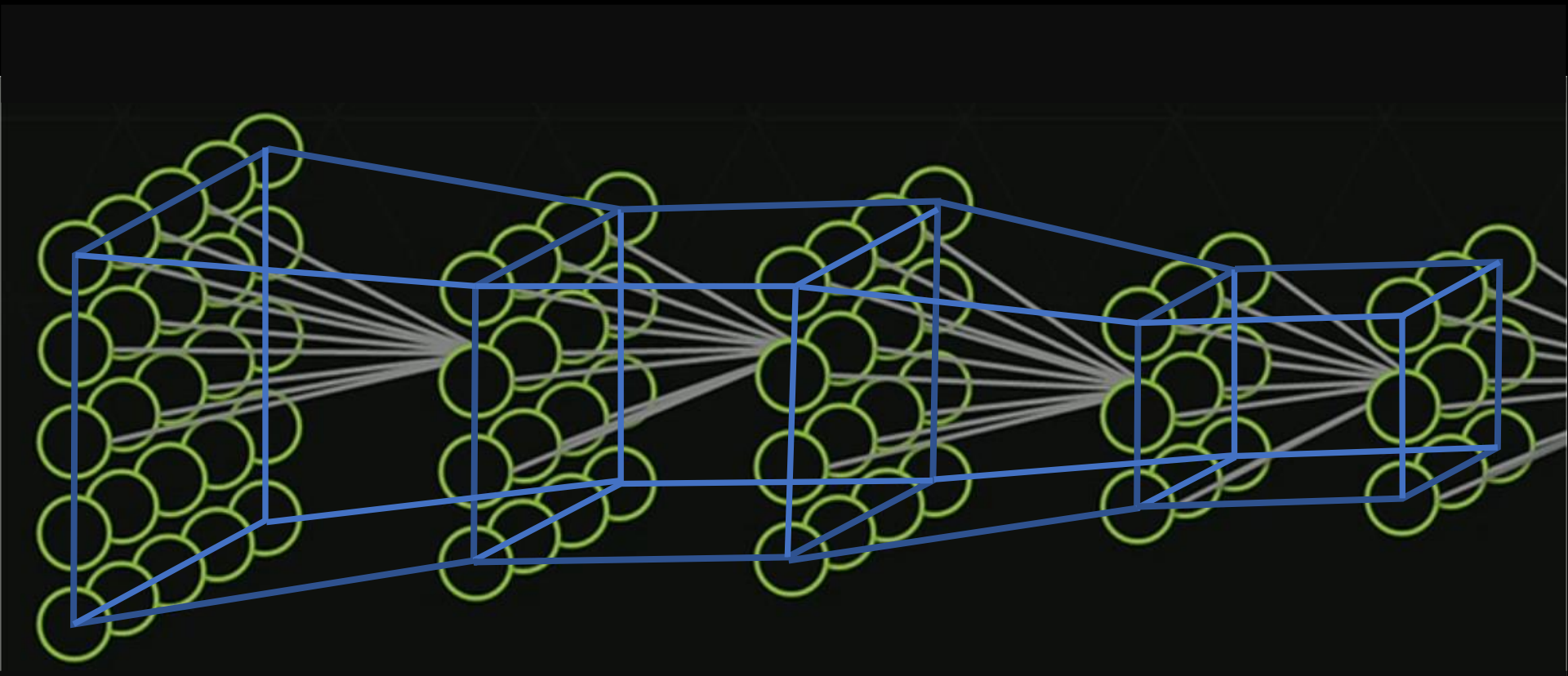

hidden layers
output layer
input layer



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

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


(5*5)
(4*4)
(4*4)
(3*3)
(3*3)



Fully connected, so how many connections 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