

AI and Deep Learning

딥 러닝

제주대학교

변영철

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

#----- a neuron

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

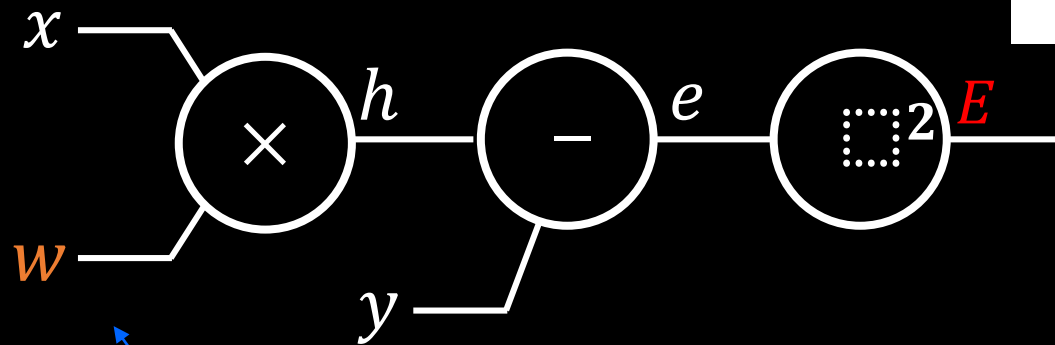
```
hypo = w * x_data
```

#----- learning

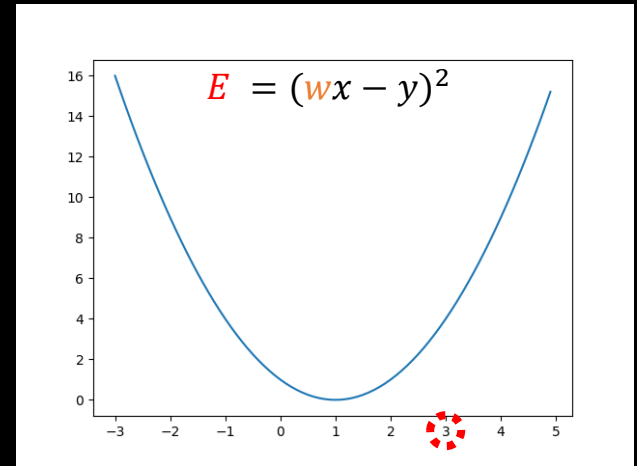
```
cost = (hypo - y_data) ** 2 #cost = Error ( $E$ )
```

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

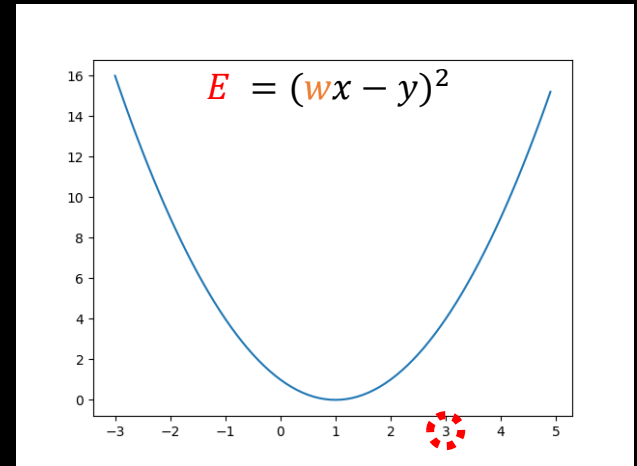
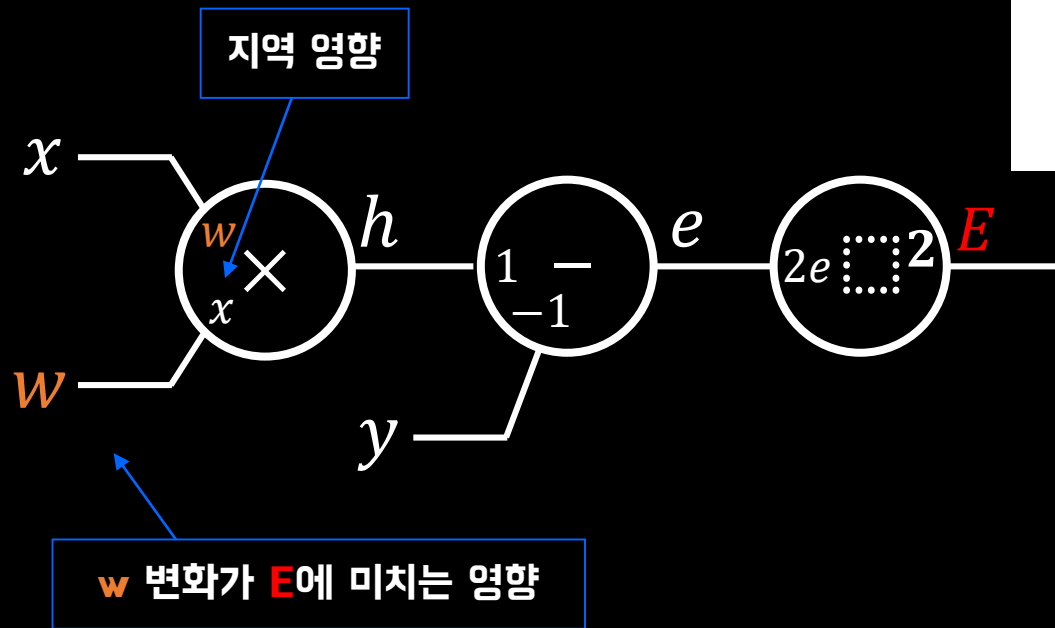
오류 계산 그래프



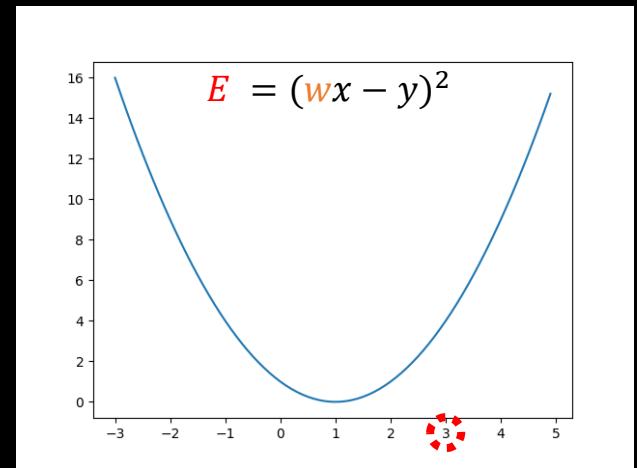
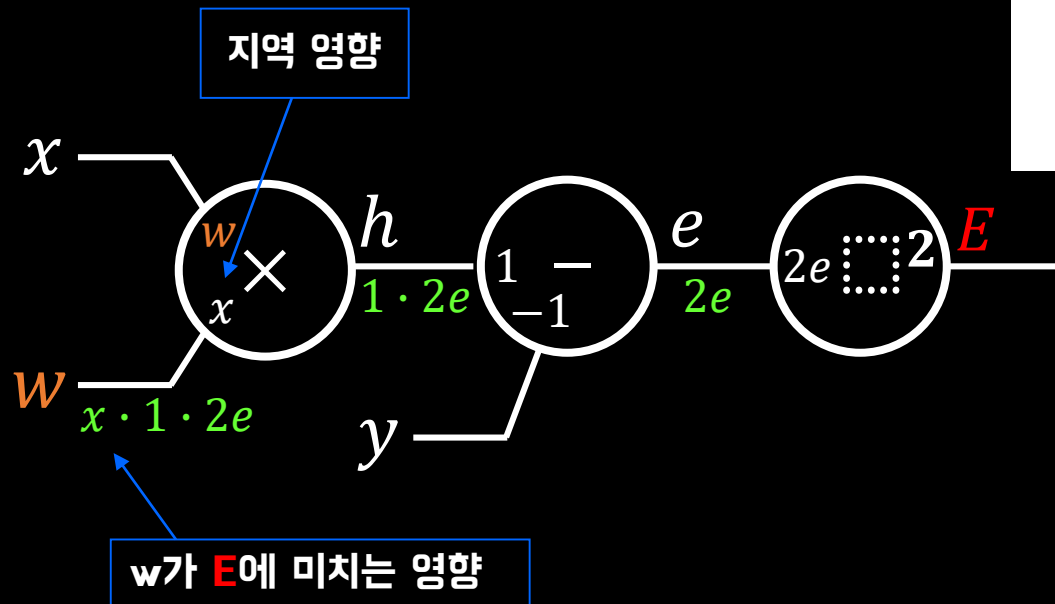
w 변화가 E 에 미치는 영향



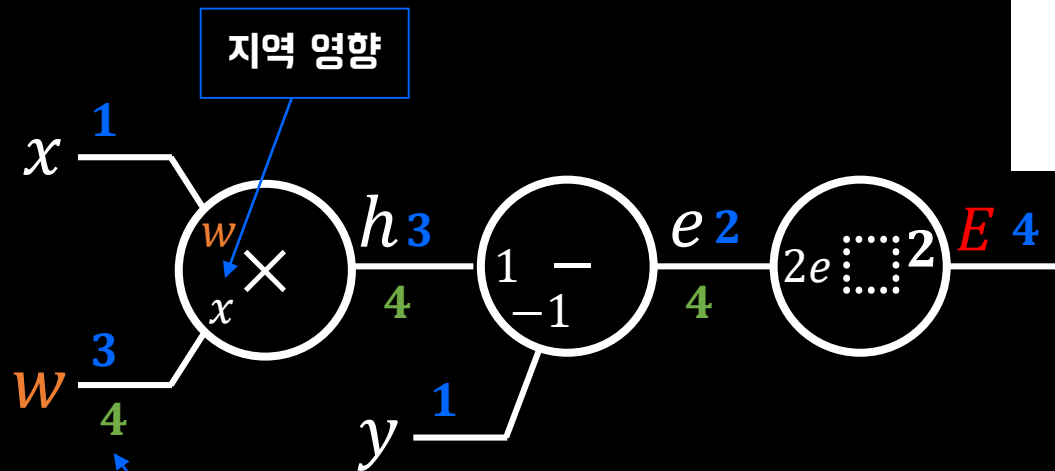
오류 계산 그래프



오류 계산 그래프

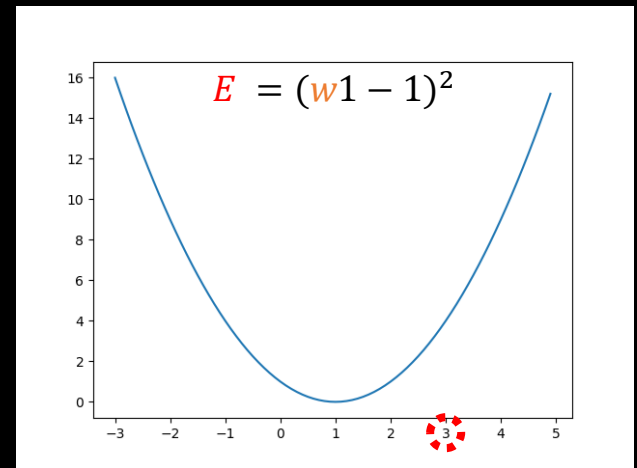


오류 계산 그래프

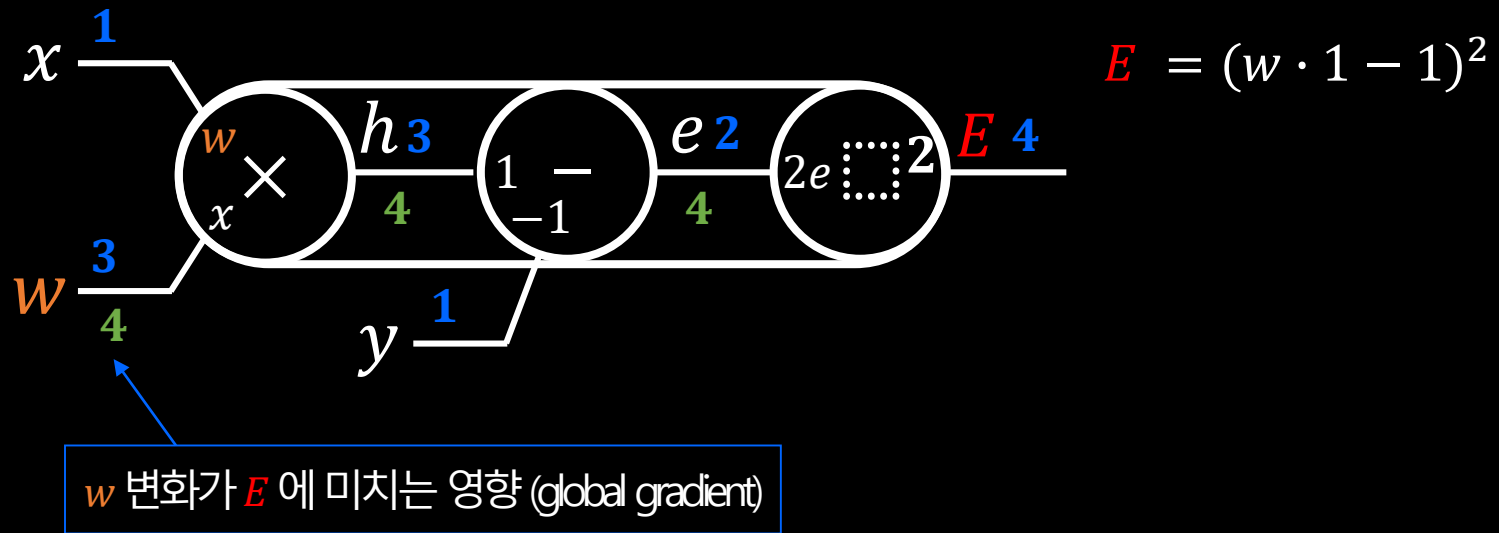


w가 E에 미치는 영향

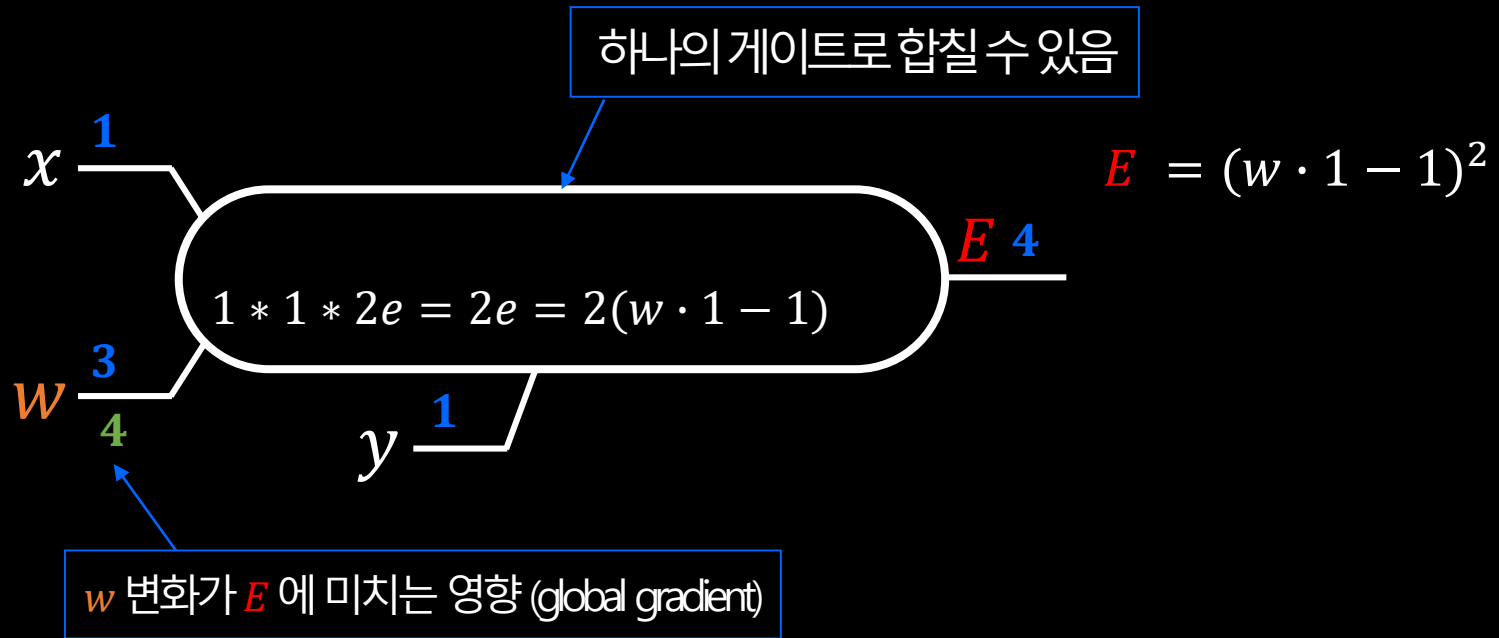
$$1 * 1 * 2e = 1 * 1 * 2(w1 - 1) = 2(w1 - 1)$$



연산 합치기

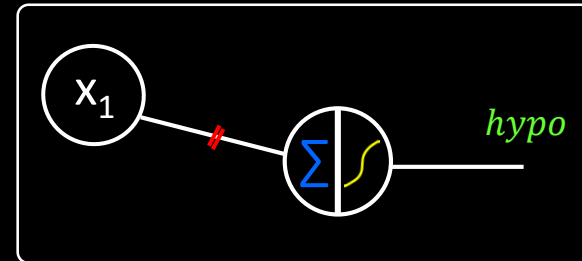


연산 합치기



오류 계산 그래프

로지스틱 리그레션
신경 세포 1개 있을 때



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

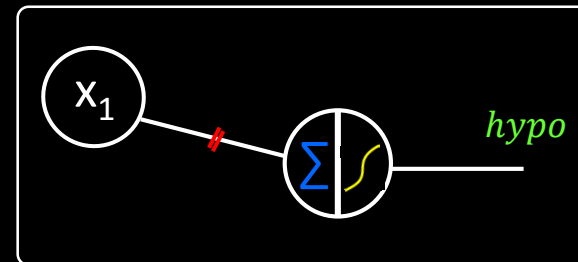
$$hypo = \frac{1}{1 + e^{-wx}}$$

w 변화가 $cost$ 에 미치는 영향은?

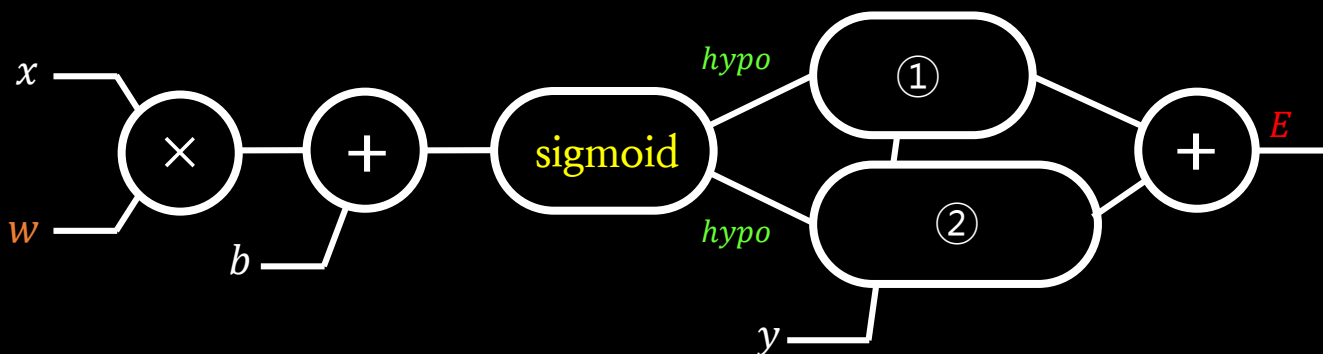
w 를 아주 조금 바꿨을 때 $cost$ 는 어떻게 변하나?

계산 그래프를 활용하거나 또는 직접 미분하거나...

오류 계산 그래프

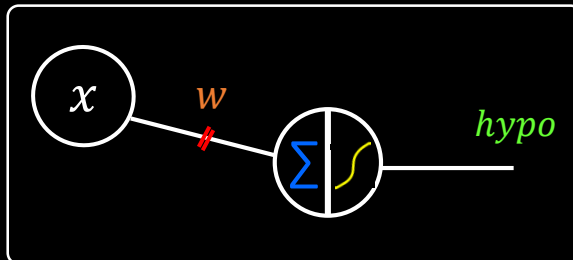
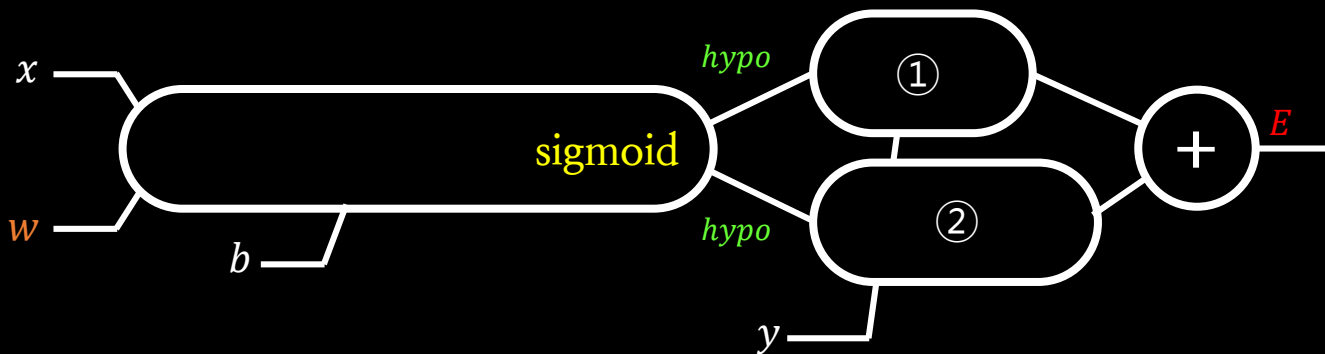


$$E = \overset{\textcircled{1}}{- (y \cdot \log(\textit{hypo}))} + \overset{\textcircled{2}}{(1 - y) \cdot \log(1 - \textit{hypo}))}$$



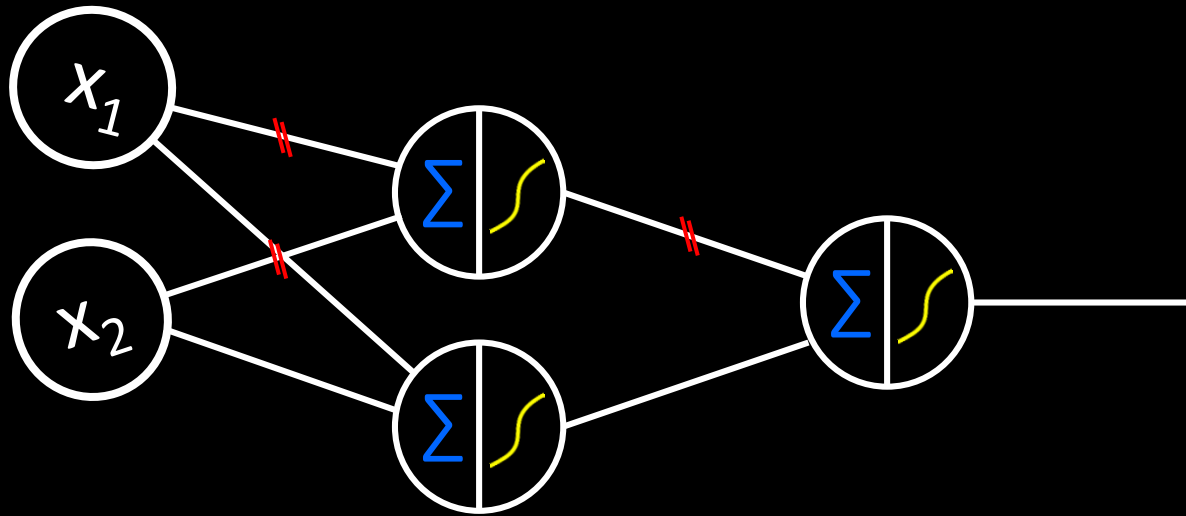
오류 계산 그래프

$$E = \overset{\textcircled{1}}{- (y \cdot \log(\text{hypo}))} + \overset{\textcircled{2}}{(1 - y) \cdot \log(1 - \text{hypo}))}$$

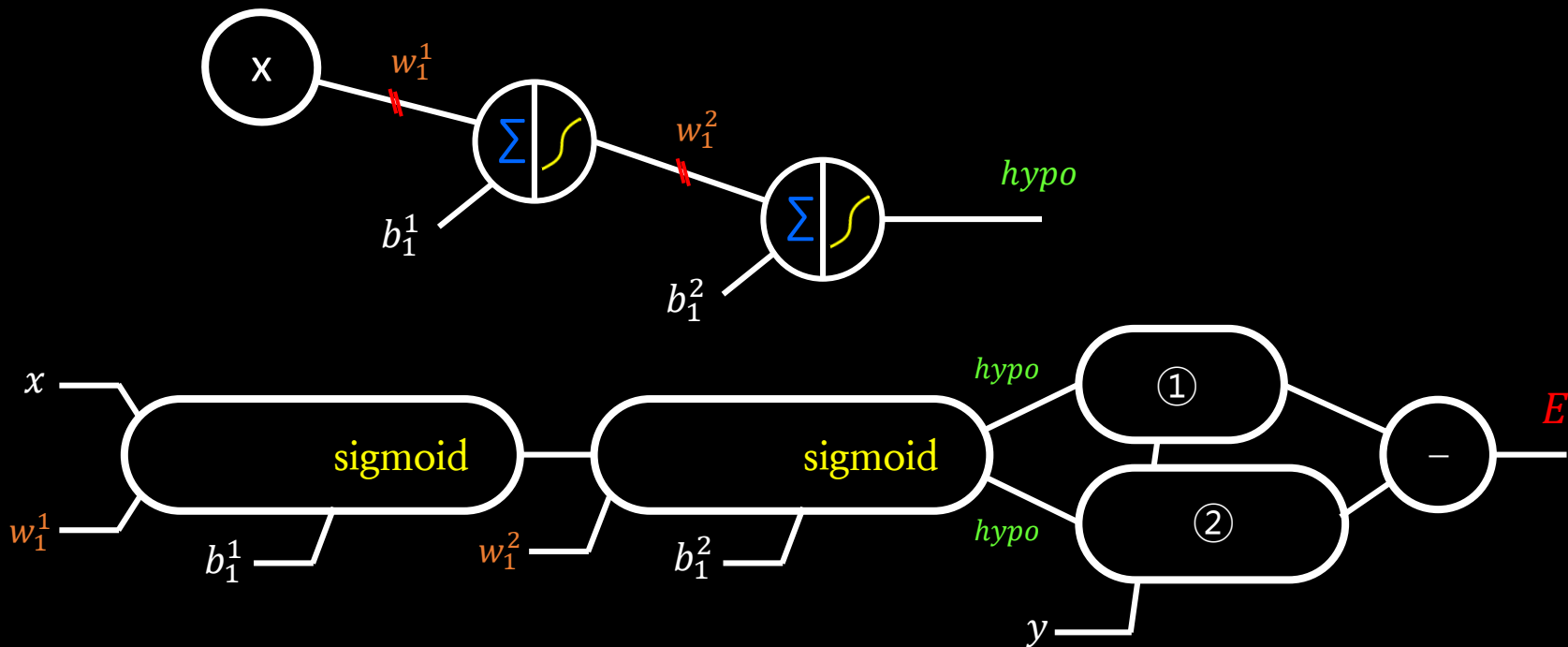


$$\text{hypo} = \frac{1}{1 + e^{-wx}}$$

오류 계산 그래프



오류 계산 그래프



사라지는 영향력

- Local gradient는 뉴런의 **sigmoid** function을 미분한 것
- **sigmoid**를 미분하면 $(1 - \text{sigmoid}(x)) * \text{sigmoid}(x)$
- **sigmoid**는 입력 값을 0~1 사이의 값으로 squash
- 100층 신경망의 경우 200개의 **sigmoid** 곱

사라지는 영향력

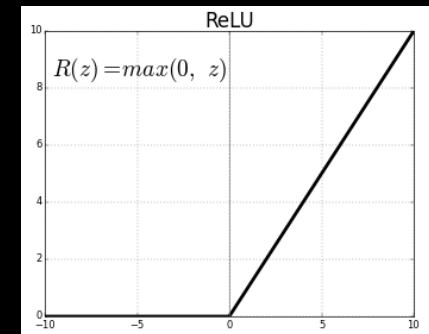
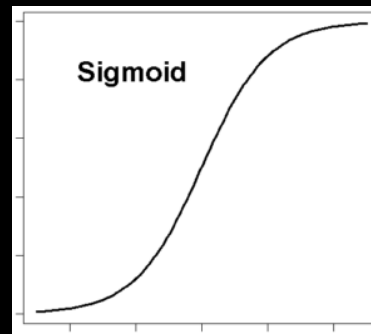
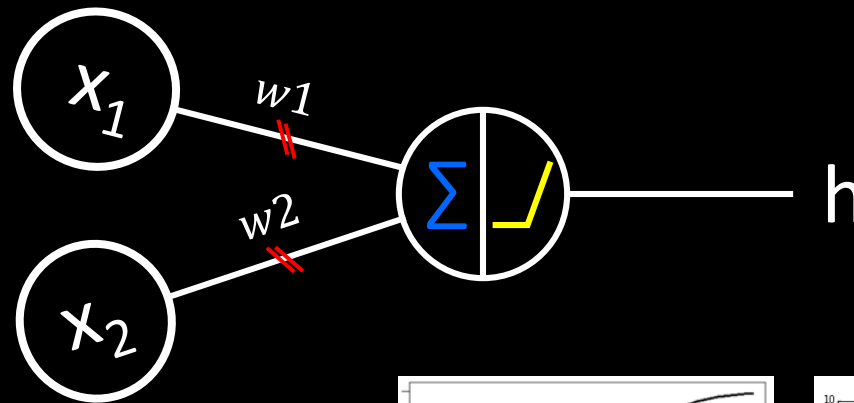
- w 가 E 에 미치는 영향은 수많은 1 이하의 곱
- 신경망의 왼쪽으로 갈수록 1 이하의 곱이 많아져서 w 변화가 E 에 미치는 영향(기울기, gradient)은 거의 0
- 즉, 영향력이 사라짐 (Vanishing Gradient)
- $w = w - \alpha \times$ (미치는 영향~거의 0)
- 신경망 왼쪽에 있는 뉴런들의 w, b 가 거의 갱신(학습)되지 않음

(실습) 18.py

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

ReLU

Logistic 함수 대신 **ReLU**를
사용함으로써 Vanishing
Gradient 문제 해결



(실습) 19.py

- ReLU를 이용하여 deep 신경망에서도 역전파 학습이 잘 됨을 보임.

MNIST

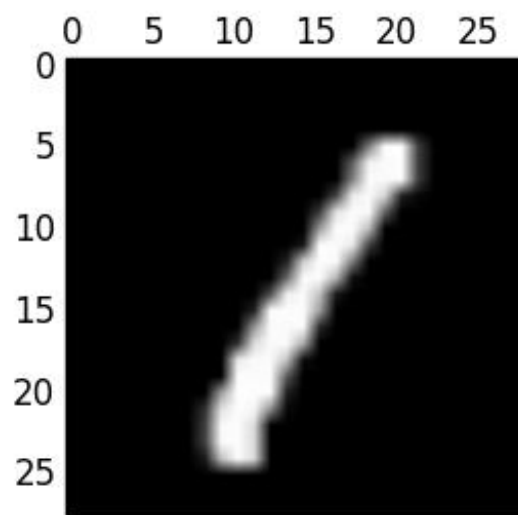
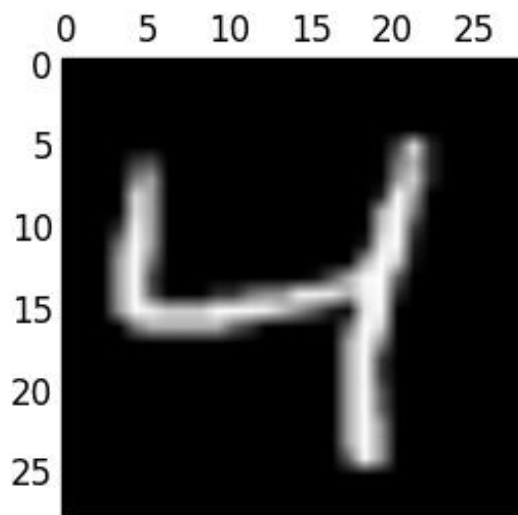
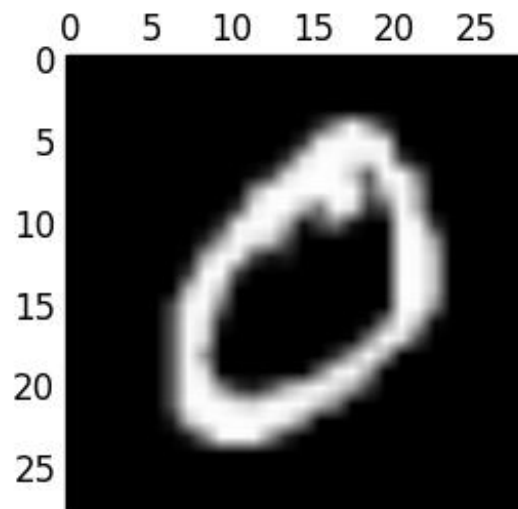
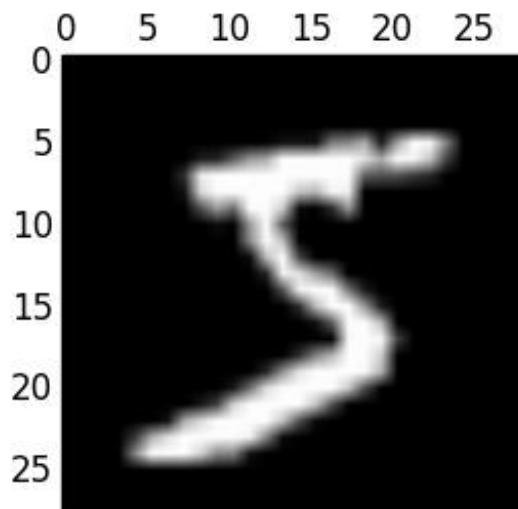


Modified National Institute of
Standards and Technology
(USA)



MNIST

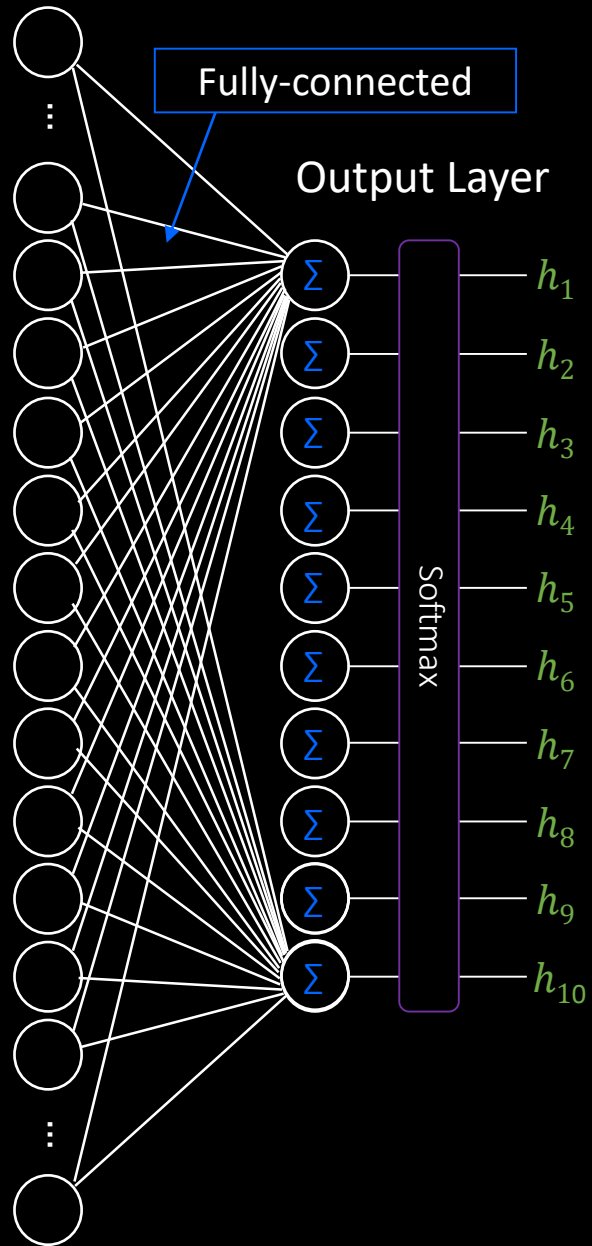


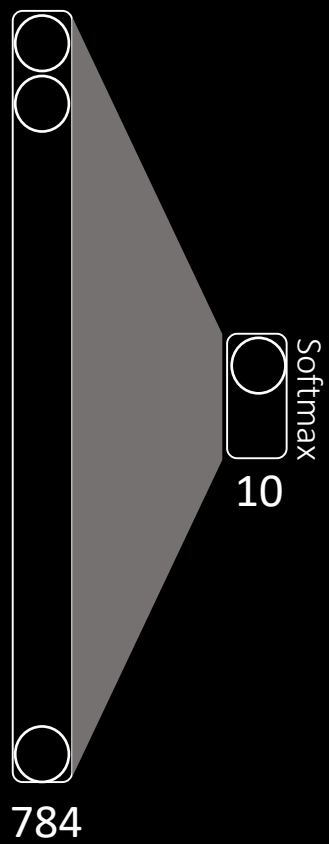


(Lab) 20.py

- 60,000 training images + 10,000 testing images
- Input image : $28 * 28$ pixels \rightarrow 784 pixels
- 784 dimension
- 10 classes (output: 0 ~ 9)
- Softmax
- 90.23% of recognition rate

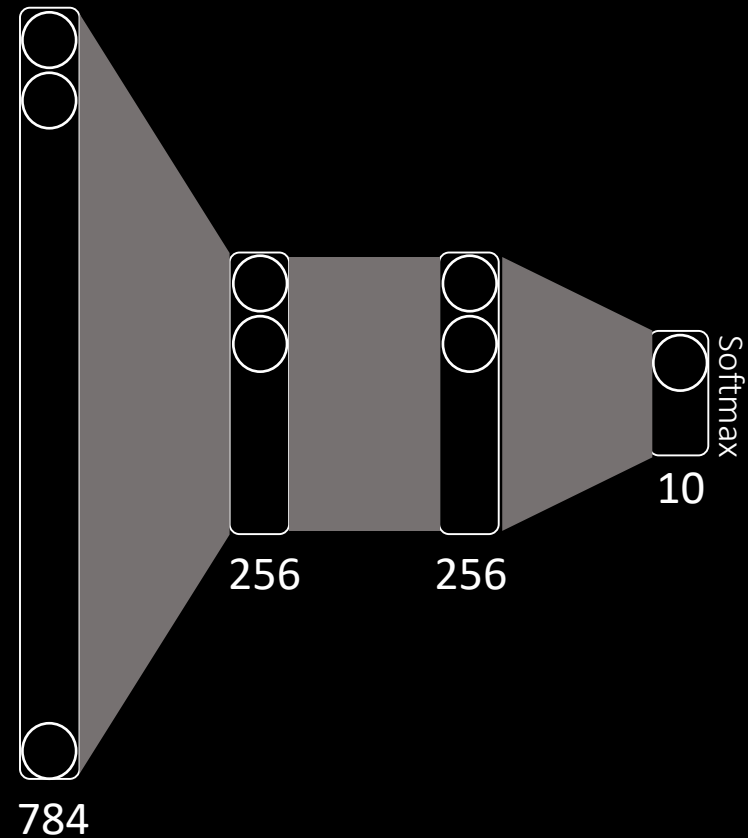
Input Layer





(Lab) 21.py

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

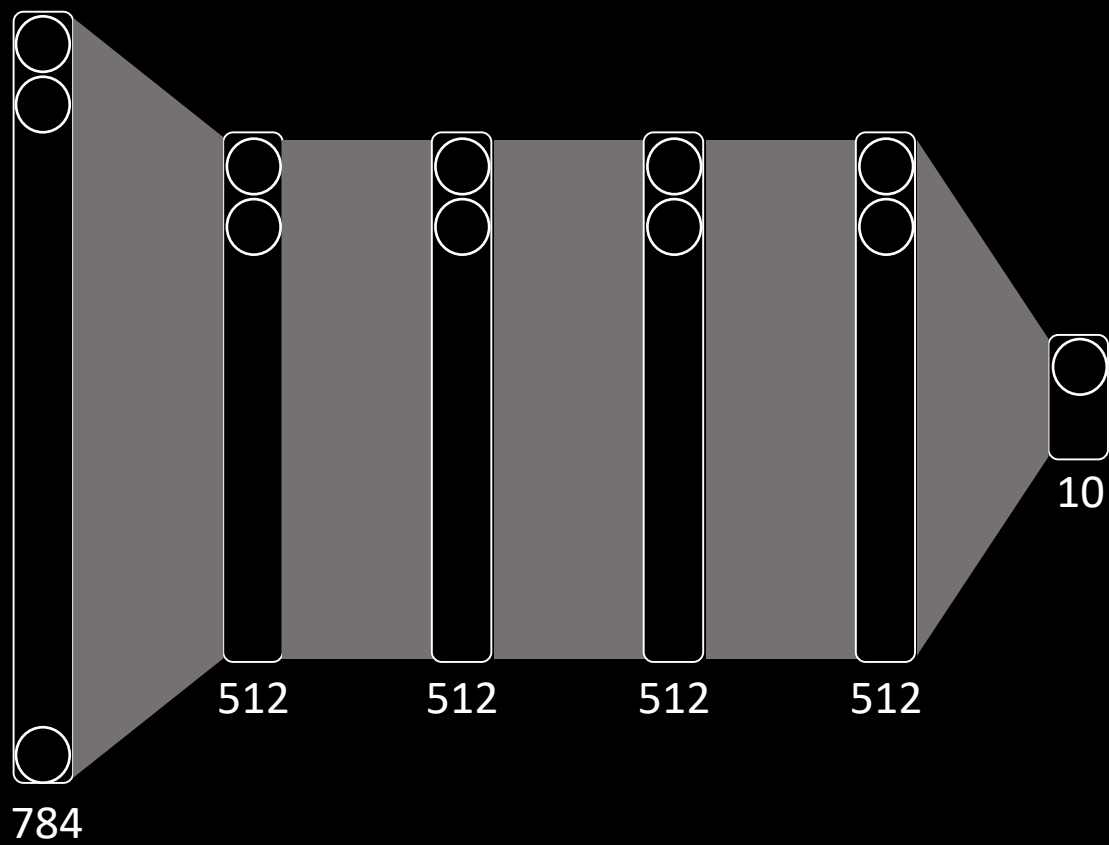


(Lab) 22.py

- 시냅스 가중치(w)와 바이어스(b)를 적절히 초기화
- Accuracy: 97.83%

(Lab) 23.py

- 시냅스 가중치(w)와 바이어스(b)를 적절히 초기화
- Deeper (DNN) -> 6 layers
- 97.23% of accuracy



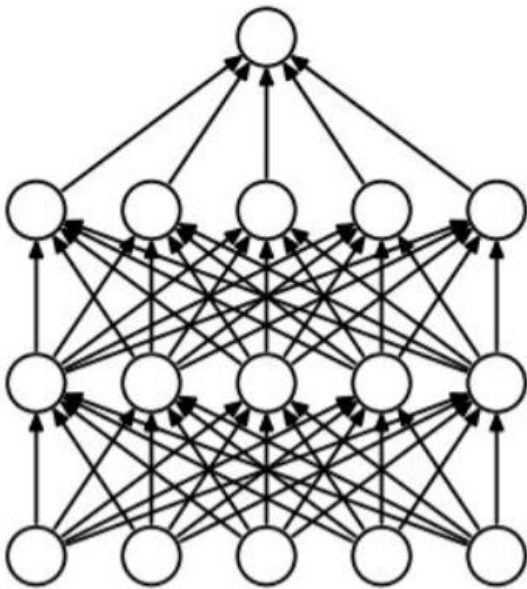
오버 피팅(over-fitting)

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

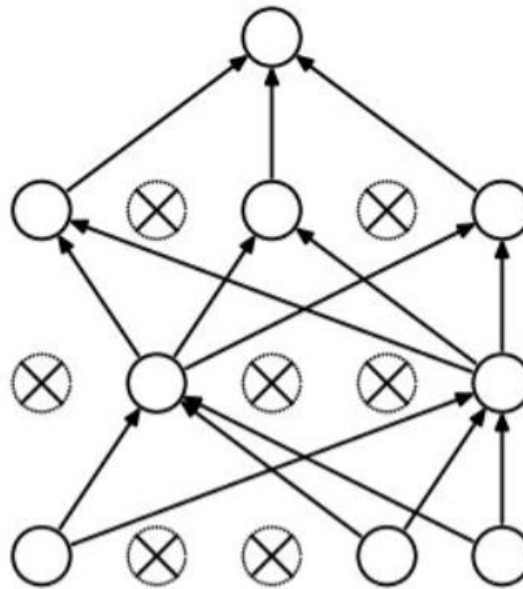
신경 세포가 많을 수록(deep & wide) 결정 경계
복잡하므로 **학습 시** 신경세포를 배제(drop-out)

Regularization: **Dropout**

“randomly set some neurons to zero in the forward pass”



(a) Standard Neural Net

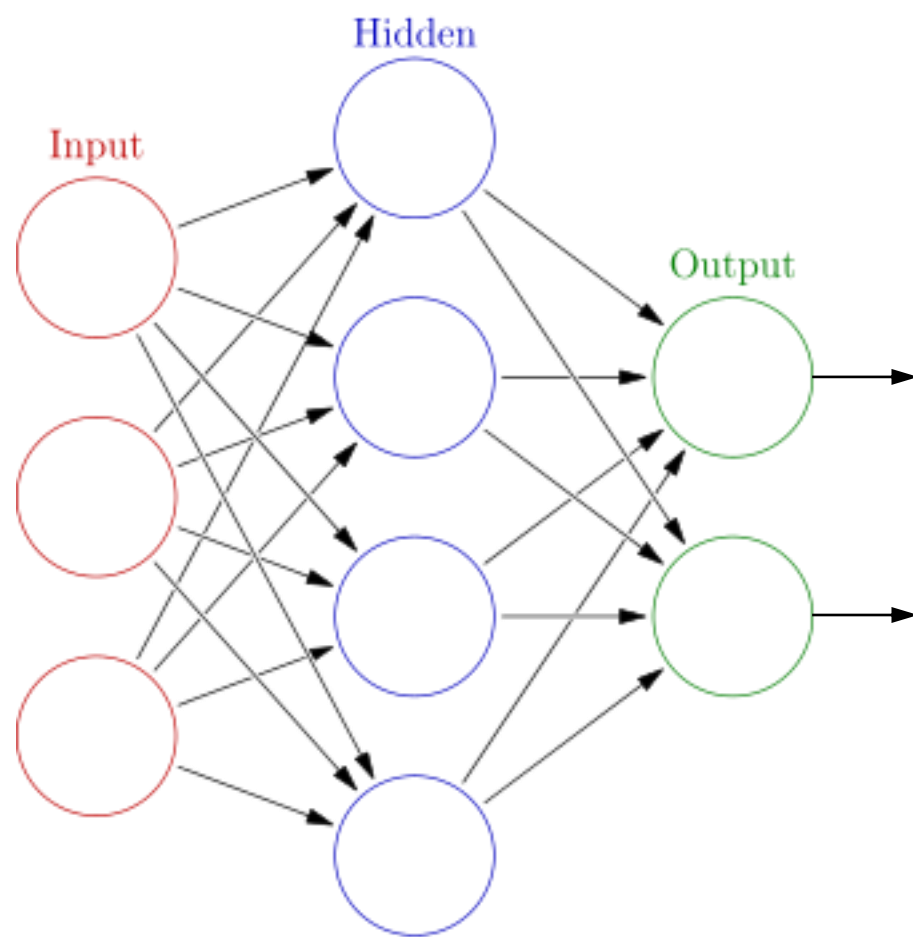


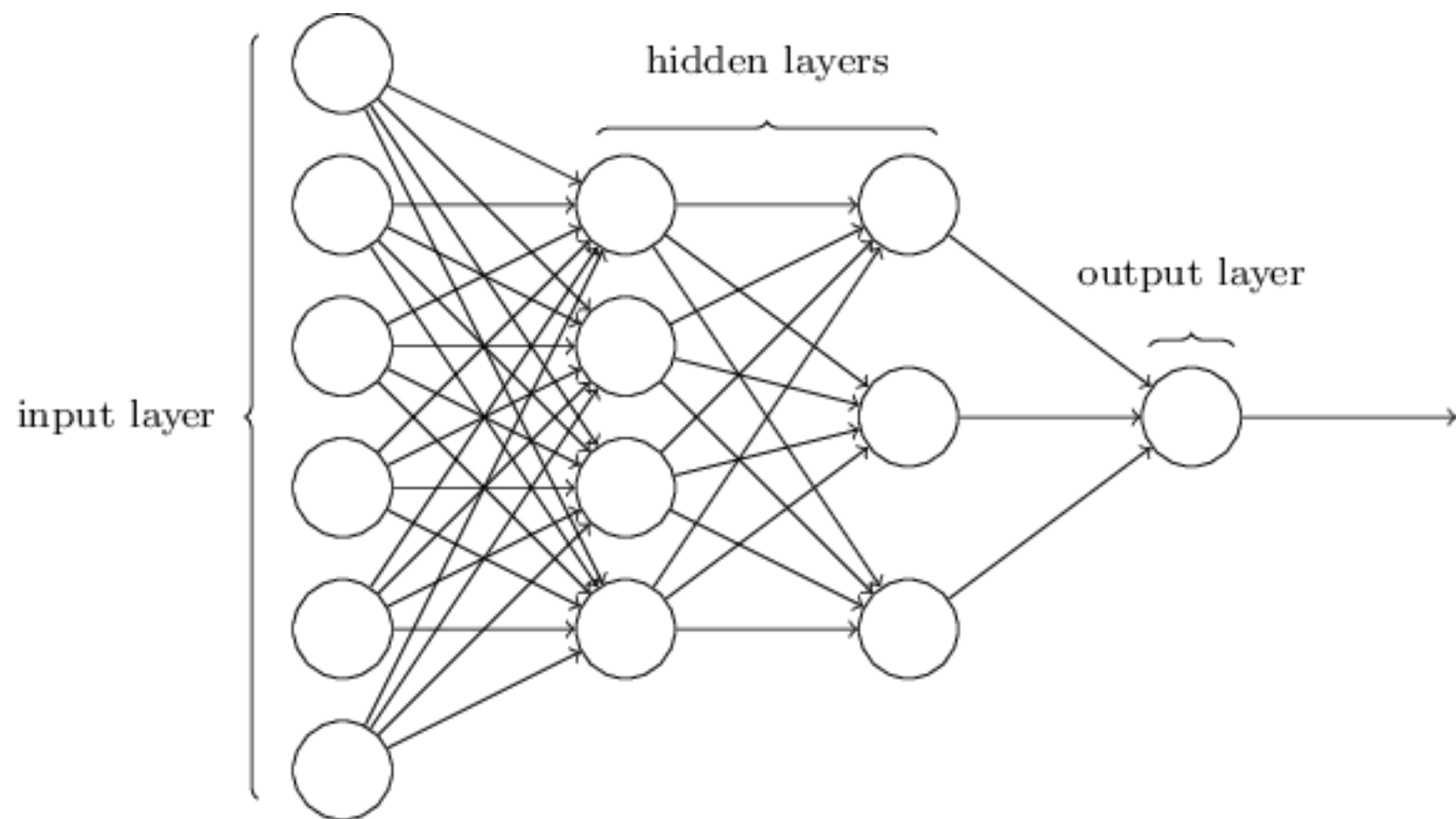
(b) After applying dropout.

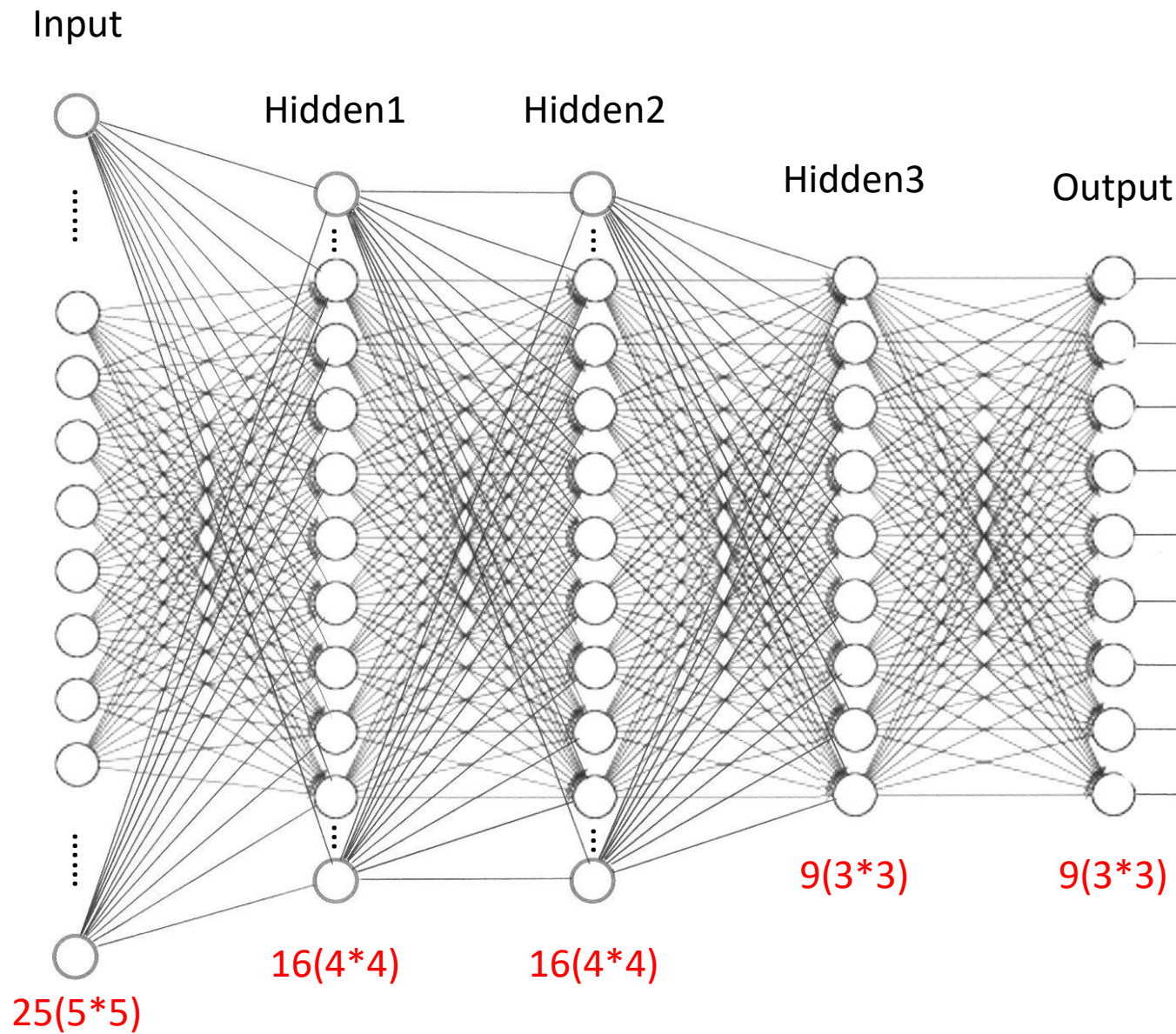
[Srivastava et al., 2014]

(실습) 24.py

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



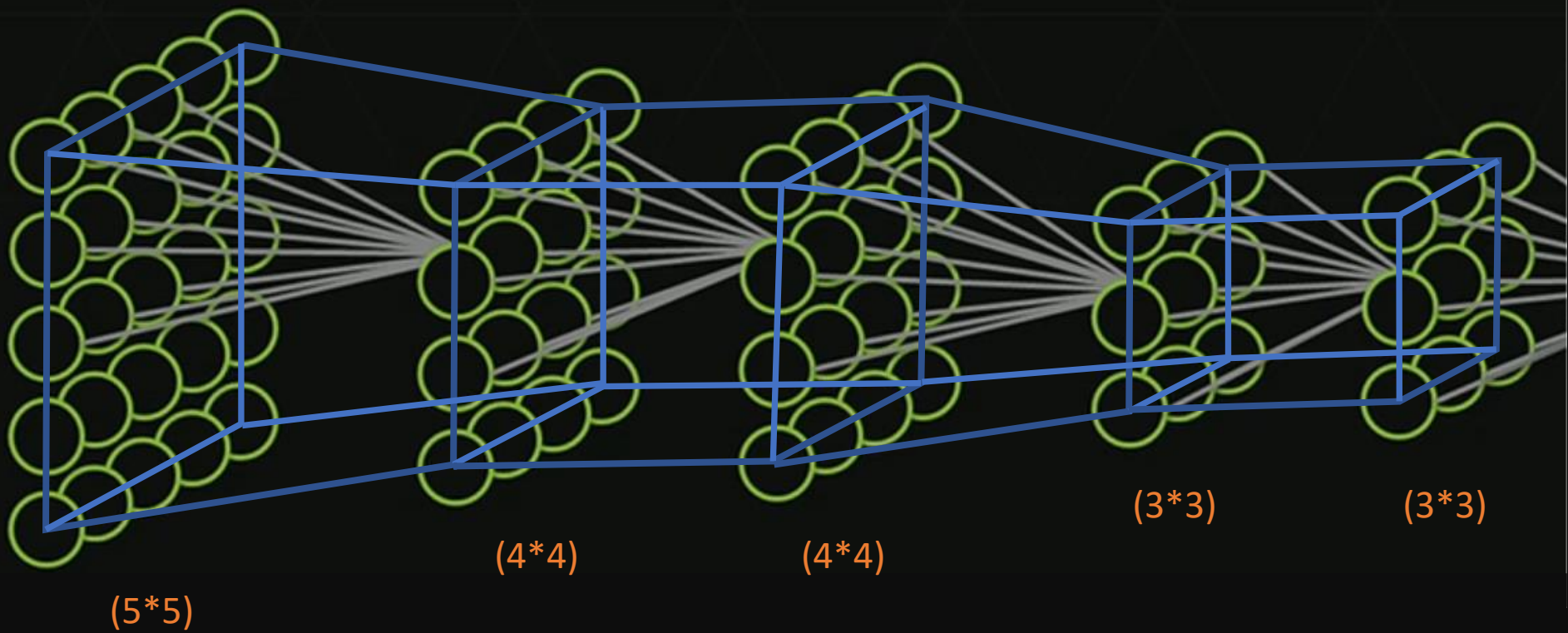




Fully connected, so how many connections(parameters) are there?

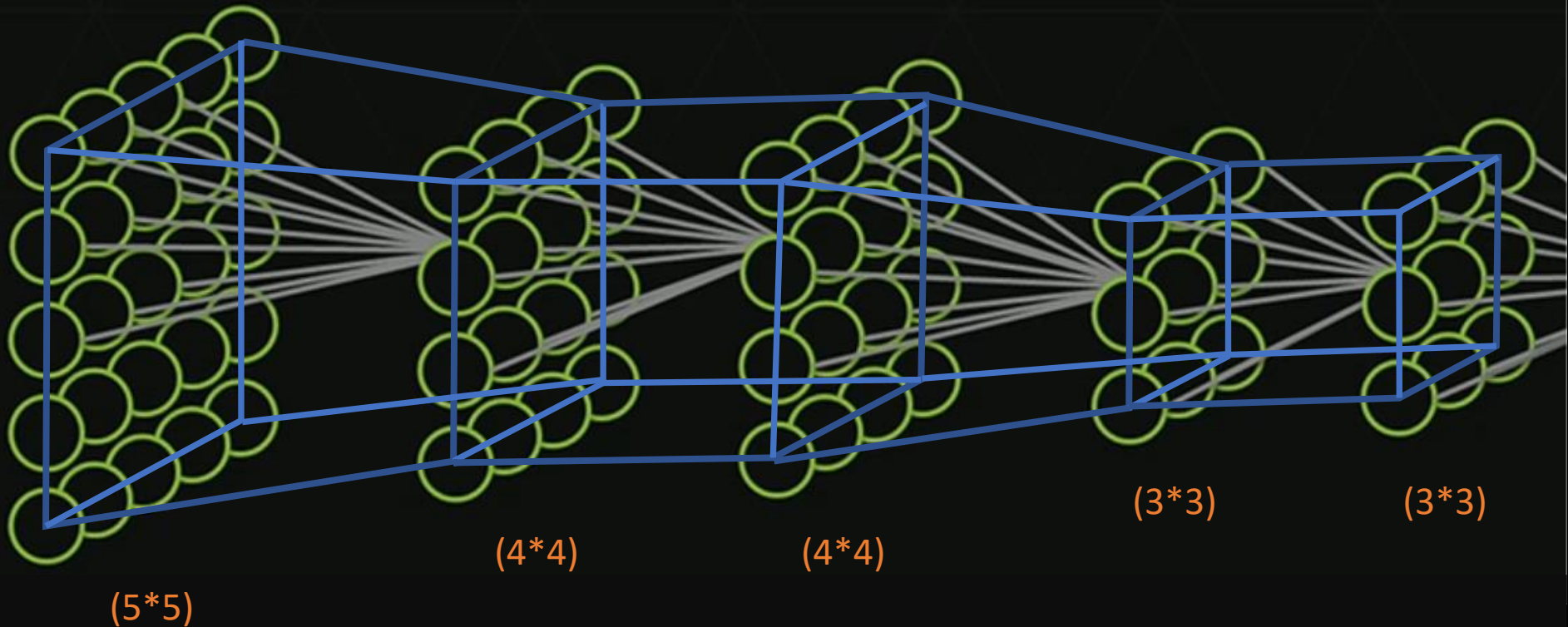
$$25 * 16 + 16 * 16 + 16 * 9 + 9 * 9 = 881$$





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



Deep Learning

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