AI and Deep Learning

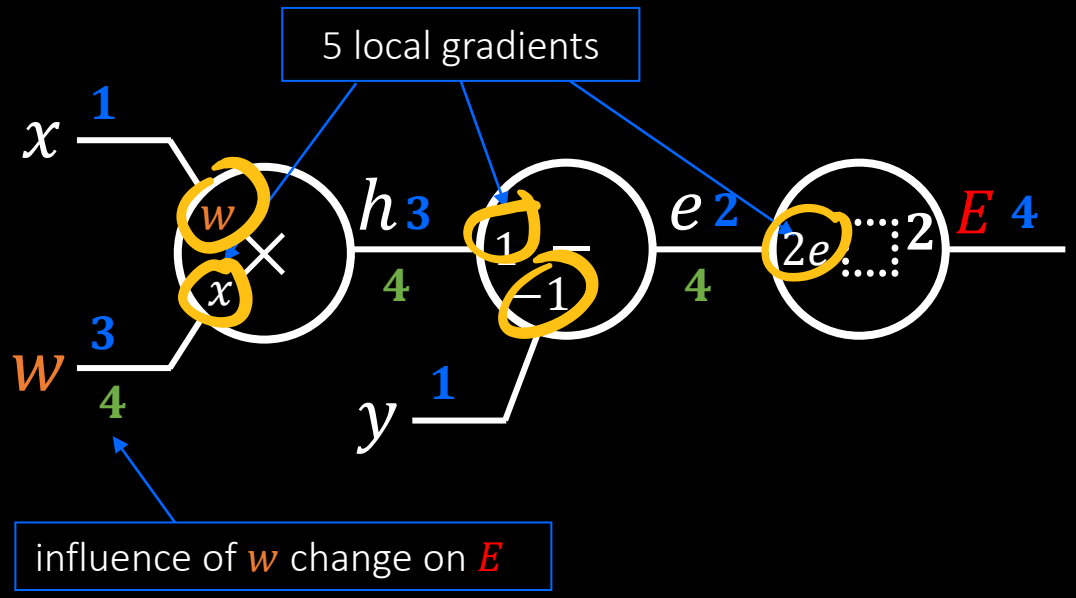
# Deep Learning

Jeju National University
Yungcheol Byun

# Agenda

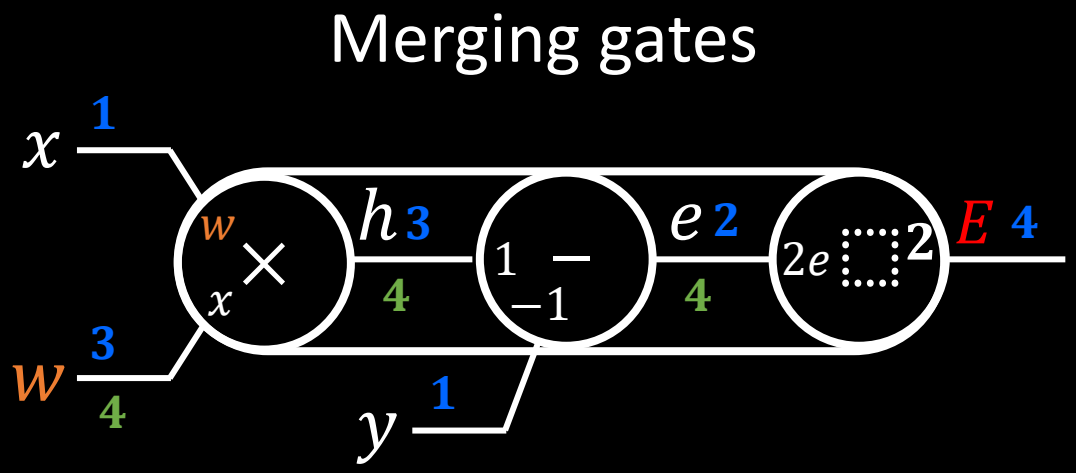
- Merging gates in a computation graph
- Vanishing gradient and ReLU
- MNIST application
- Overfitting and drop-out
- Deep Learning

# Influence of $w$ change on $E$

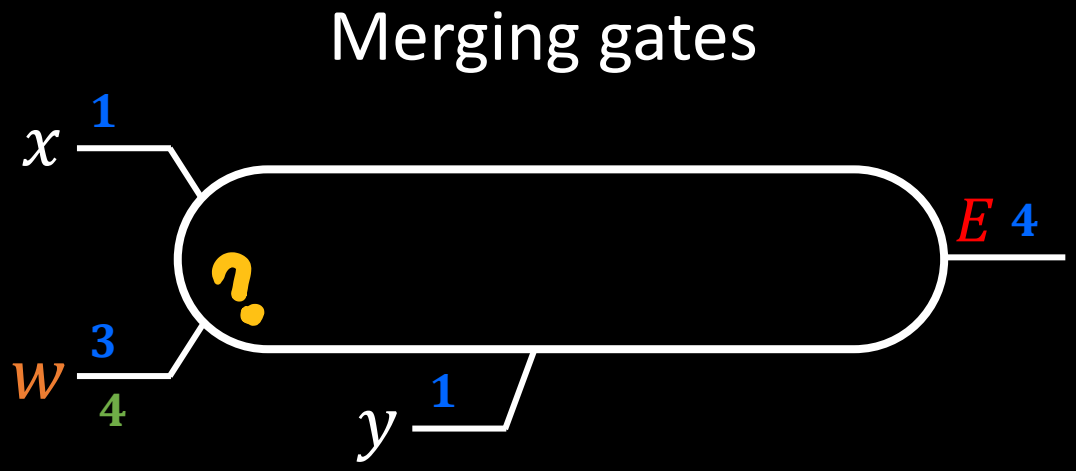



is multiplication of all the local gradients in the graph (chain rule)

# Influence of $w$ change on $E$

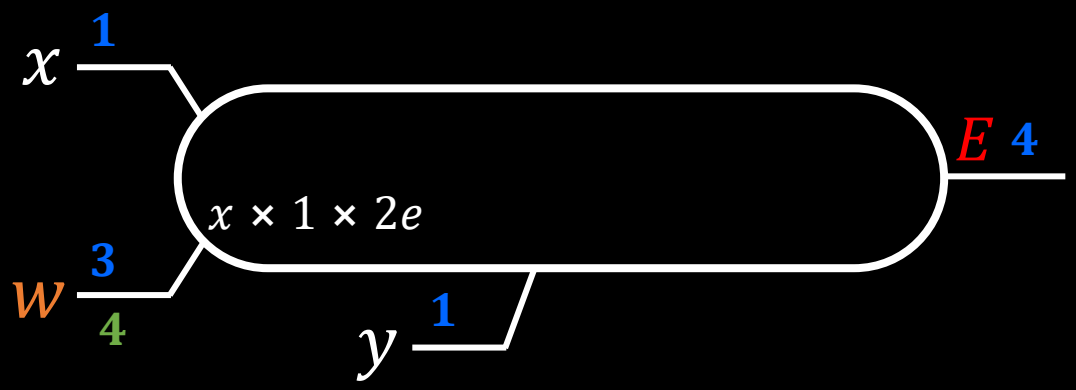


# Influence of $w$ change on $E$



# Influence of $w$ change on $E$

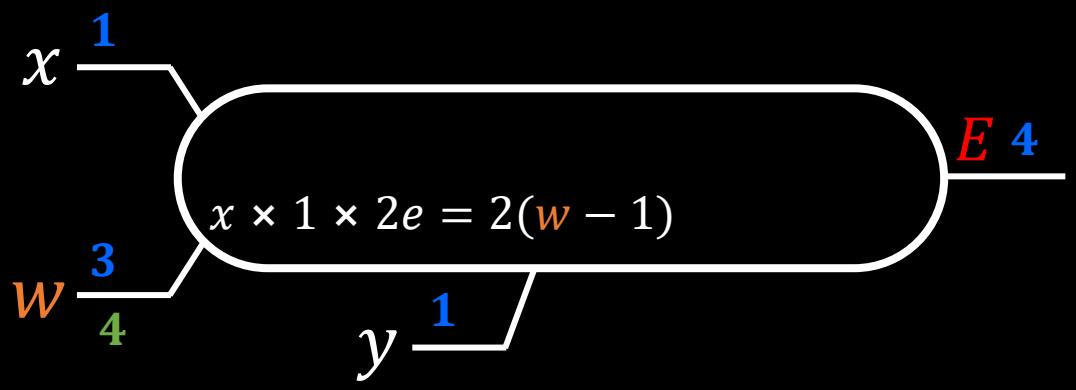
Merging gates




is multiplication of all the local gradients in the graph (chain rule)

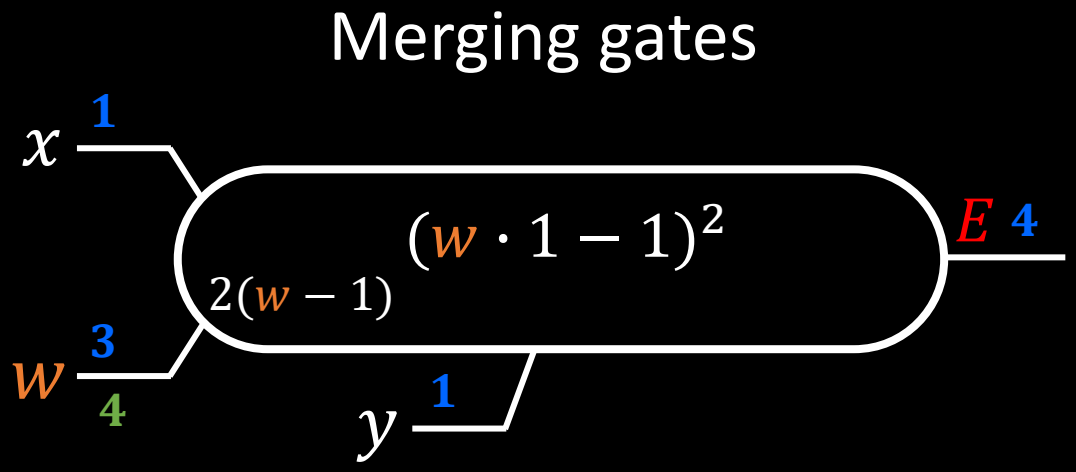
# Influence of $w$ change on $E$

Merging gates




is multiplication of all the local gradients in the graph (chain rule)

# Influence of $w$ change on $E$




Therefore, the local gradient is derivative of the function $E$.

# Influence of $w$ change on $E$

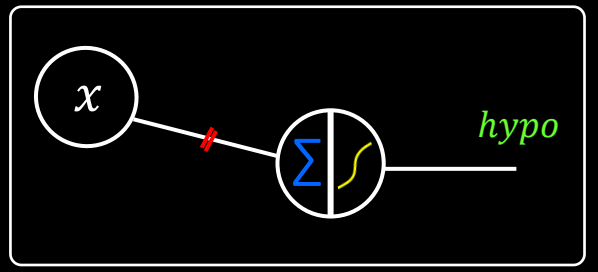
$$E = (w \cdot 1 - 1)^2$$

Derivative of $E$ with respect to $w$

$$\frac{\partial E}{\partial w} = \frac{\partial}{\partial w}(w \cdot 1 - 1)^2 = 2(w - 1)$$

# Cost/Error function

for logistic regression



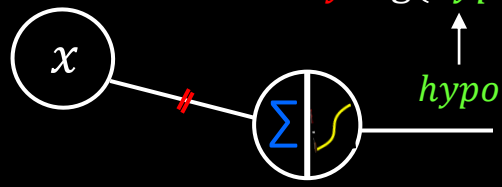

$$hypo = \frac{1}{1 + e^{-wx}}$$

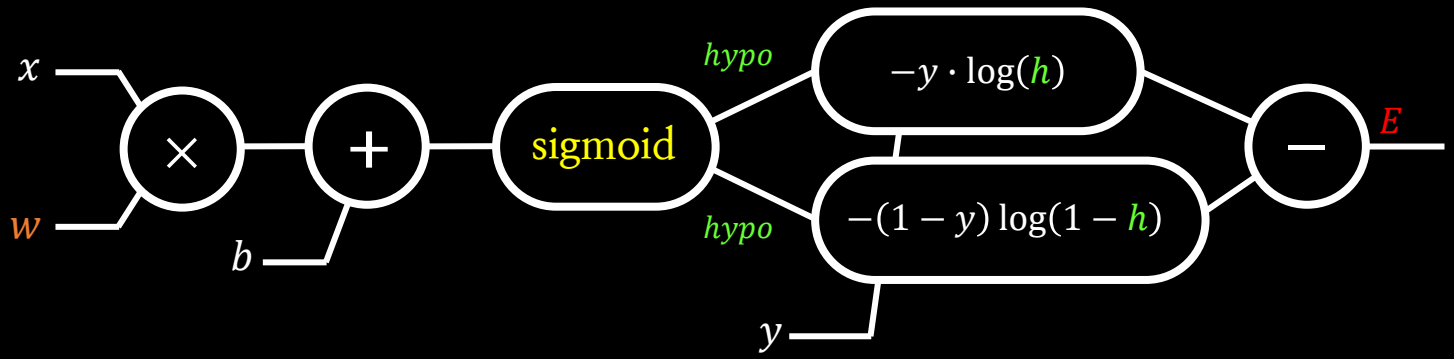
$$E = -y \log(hypo) - (1 - y)\log(1 - hypo)$$

# Computational Graph

**Binary Cross Entropy 오류함수(loss function)**

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



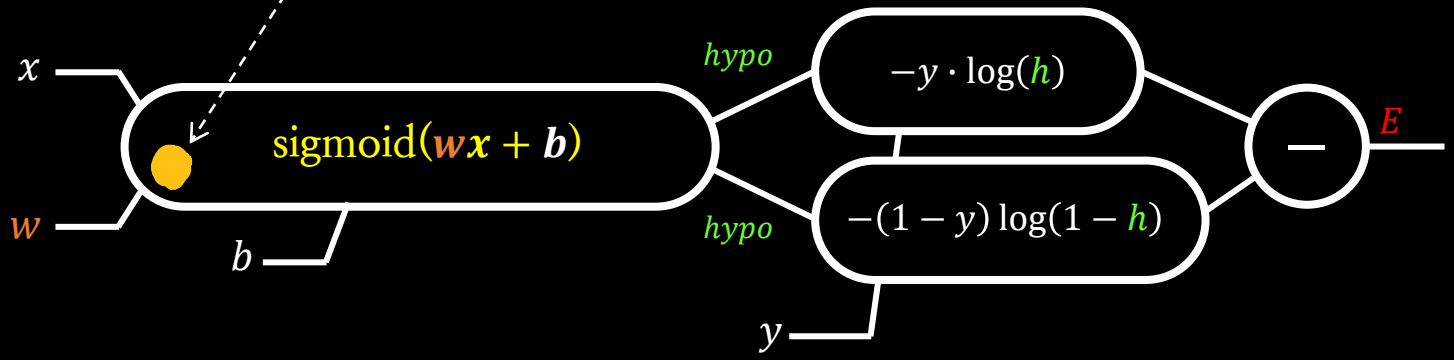



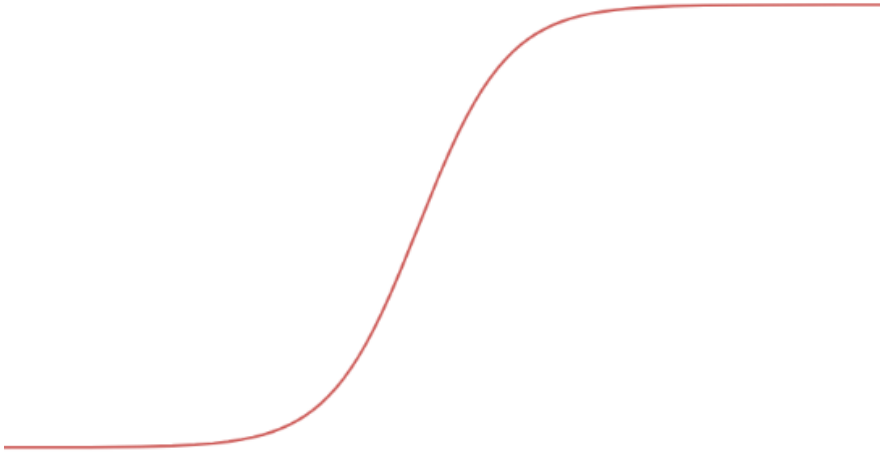

$$\frac{\partial E}{\partial w} =$$

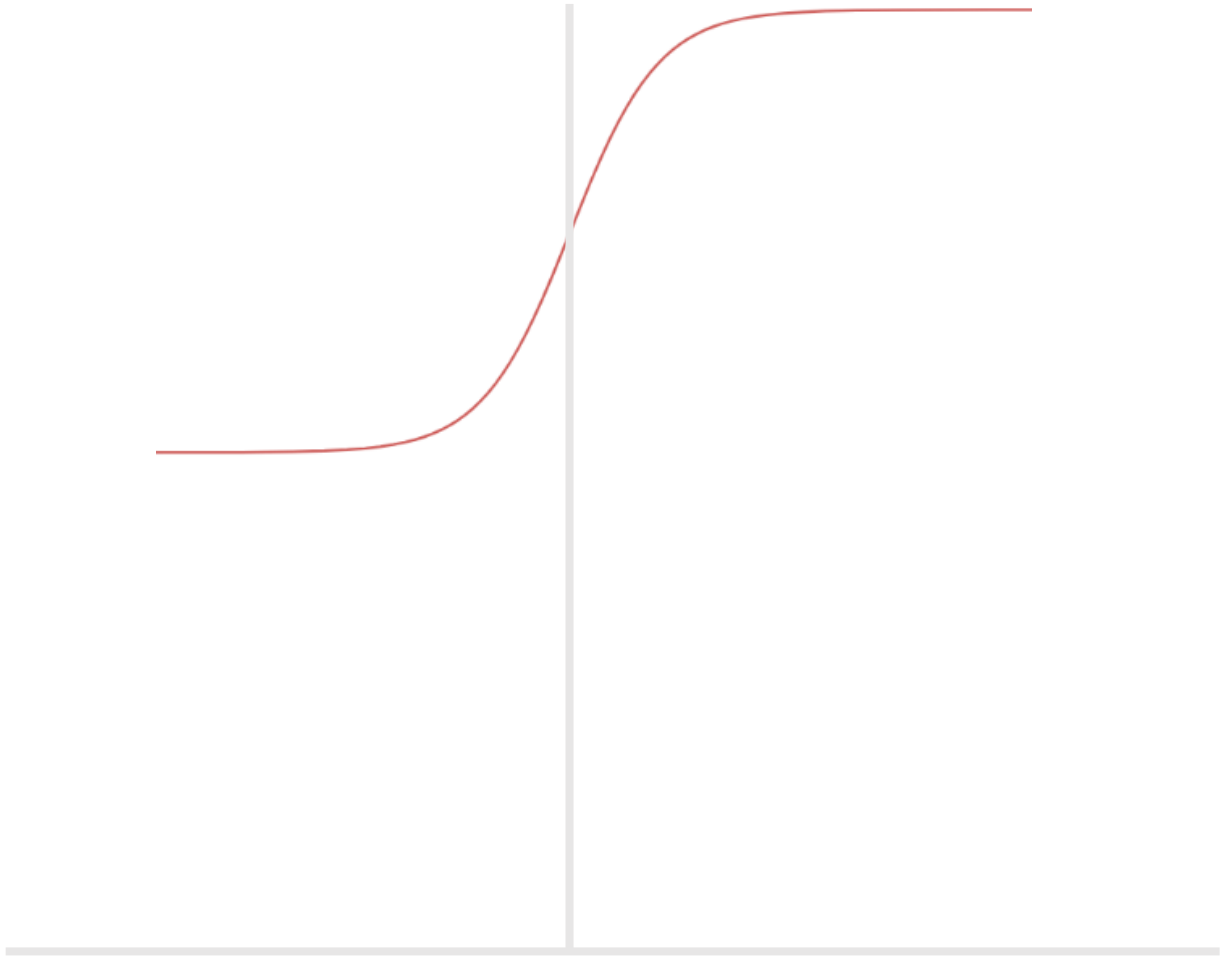
# Computational Graph

Merging gates

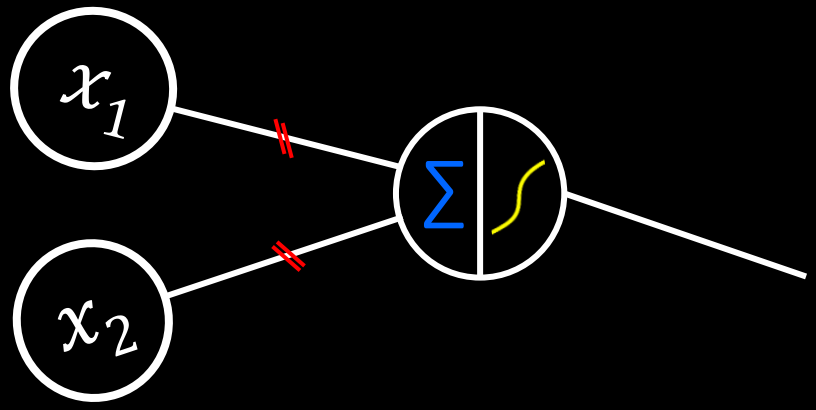
How we get the local gradient of the merged gate(sigmoid)?



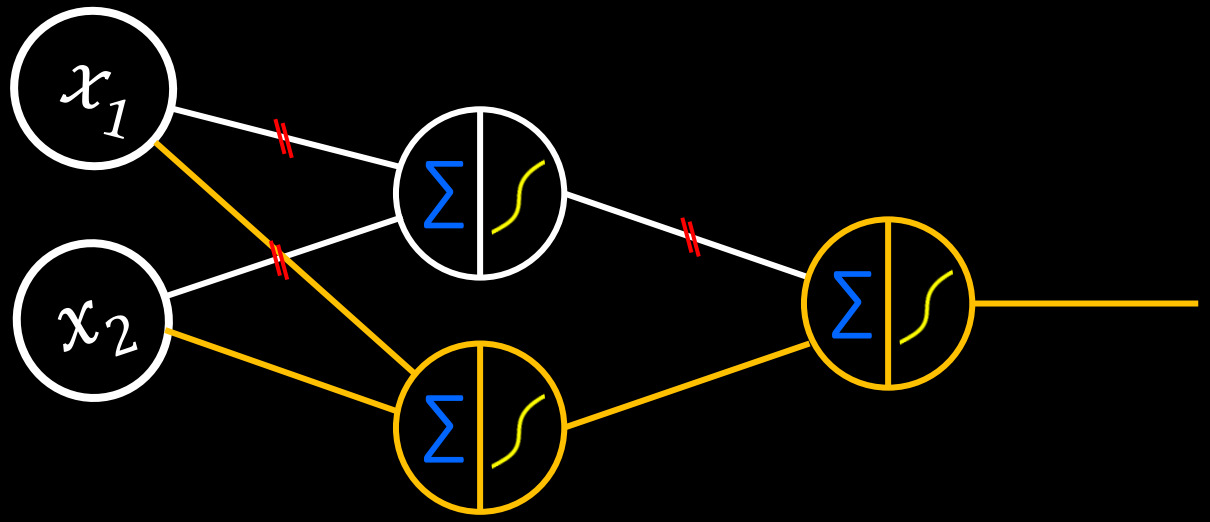




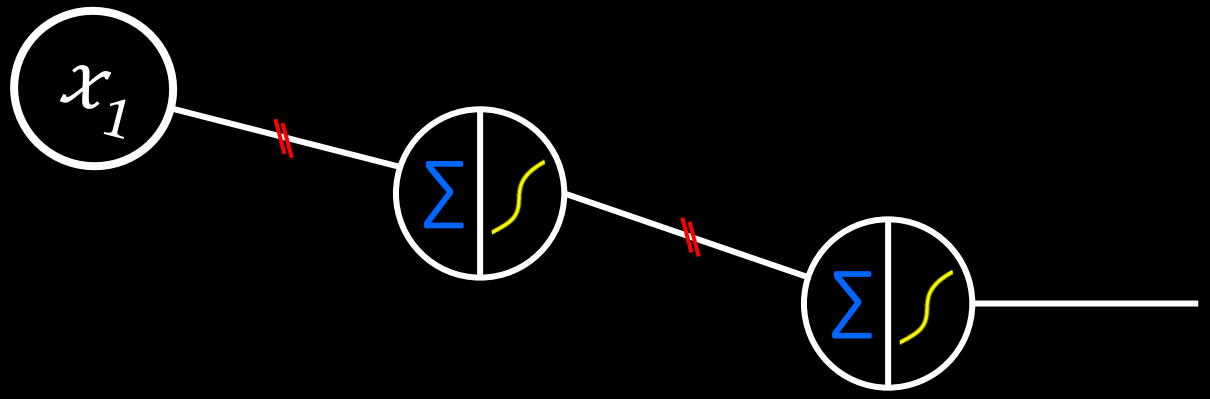
# 1 Neuron (2-layer)



# 3-layer NN



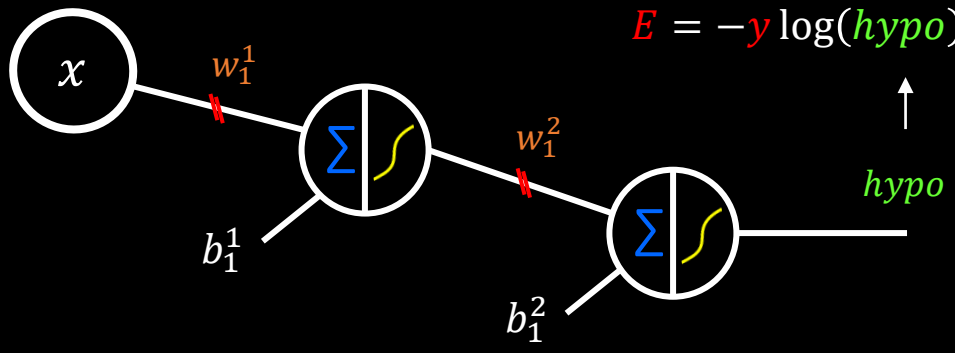
# 3-layer NN (simplified)

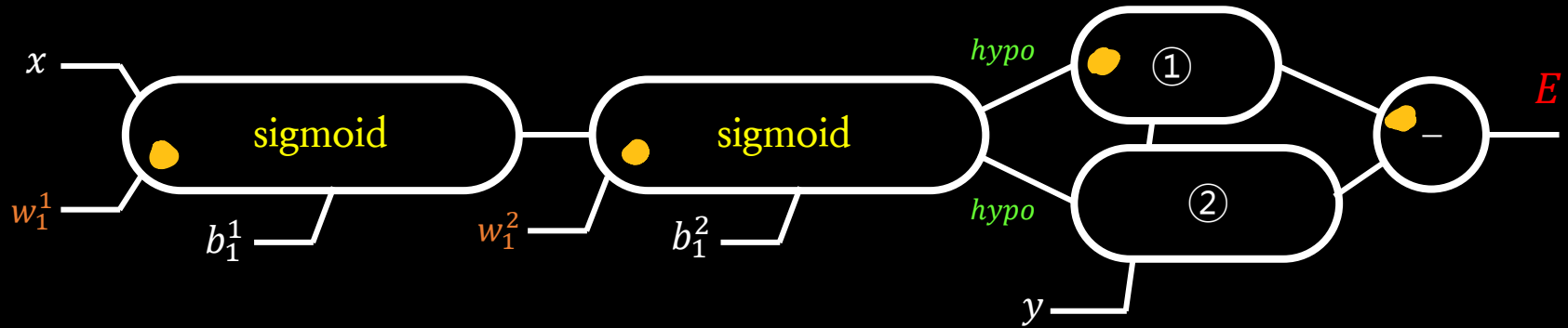


# Influence of $w$ change on $E$

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

① ②

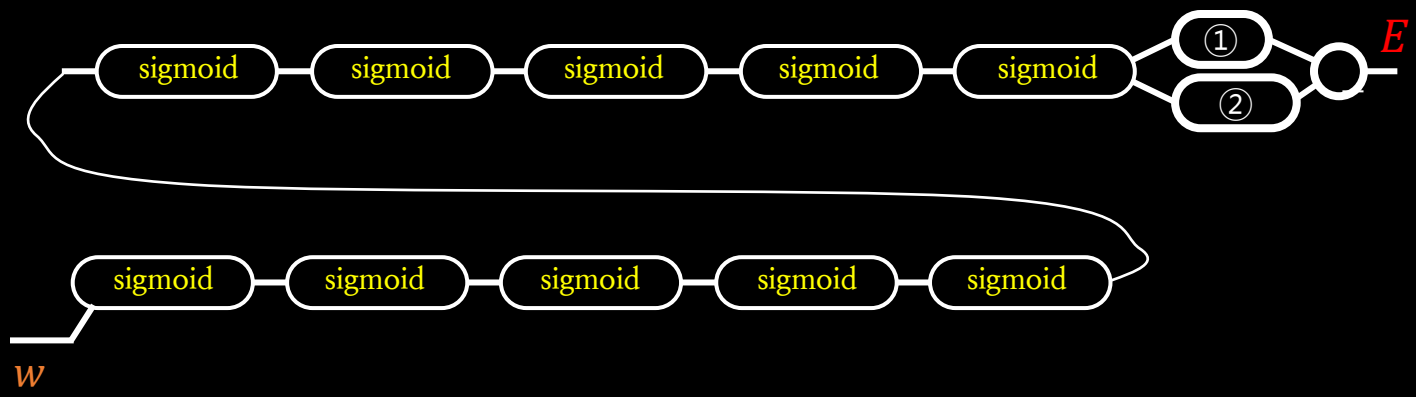


# 10-layer Neural Network

The giant monster, computational graph!

# Influence of $w$ change on $E$




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

*Hint*: chain rule!

# Vanishing Gradient

- The derivative of sigmoid function is sigmoid x (1-sigmoid)
- Two multiplication of sigmoid for a single neuron, 20 multiplications for 10 connected neurons
- Each sigmoid gives us the value between 0 and 1.

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

- The influence of $w$ change on $E$ is calculated through *many* multiplications of the values between 0 and 1, which gives us almost 0.
- Vanishing Gradient
- $w = w - \alpha \cdot$ (almost 0)
- $b = b - \alpha \cdot$ (almost 0)
- Therefore, no updates in $w$ and $b$

# (Lab) 18.py

- XOR problem using 4-layer neural networks
- Failed owing to vanishing gradient

https://github.com/yungbyun/neuralnetworks

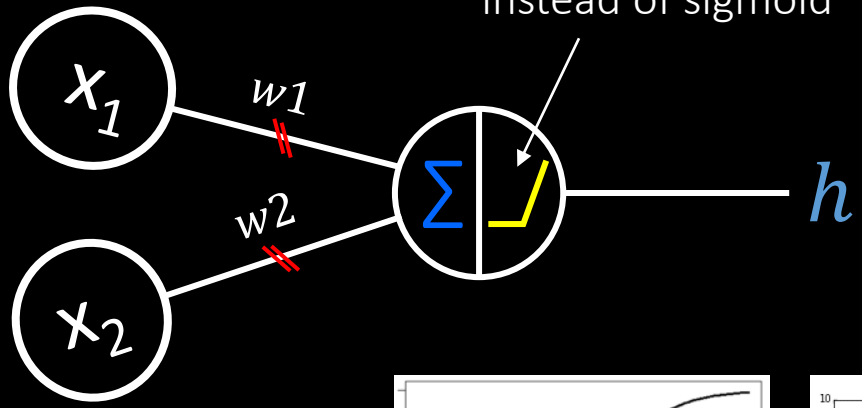
# The Dark Age in Artificial Intelligence and Neural Networks (~2006)

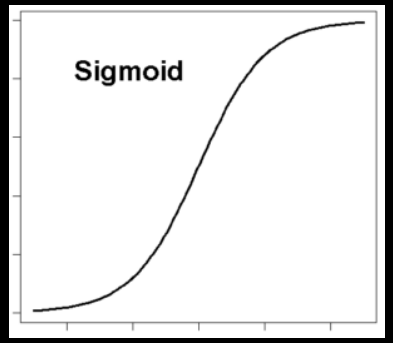
since back-propagation by Hinton in 1986

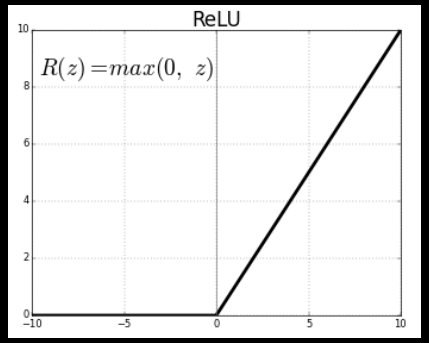
# ReLU

proposed by Hahnloser in 2000 and demonstrated for deep networks in 2011

using ReLU (Rectified Linear Unit) instead of sigmoid



# (Lab) 19.py

- Solving vanishing gradient problem using ReLU activation function
- Back-propagation is working by using ReLU.

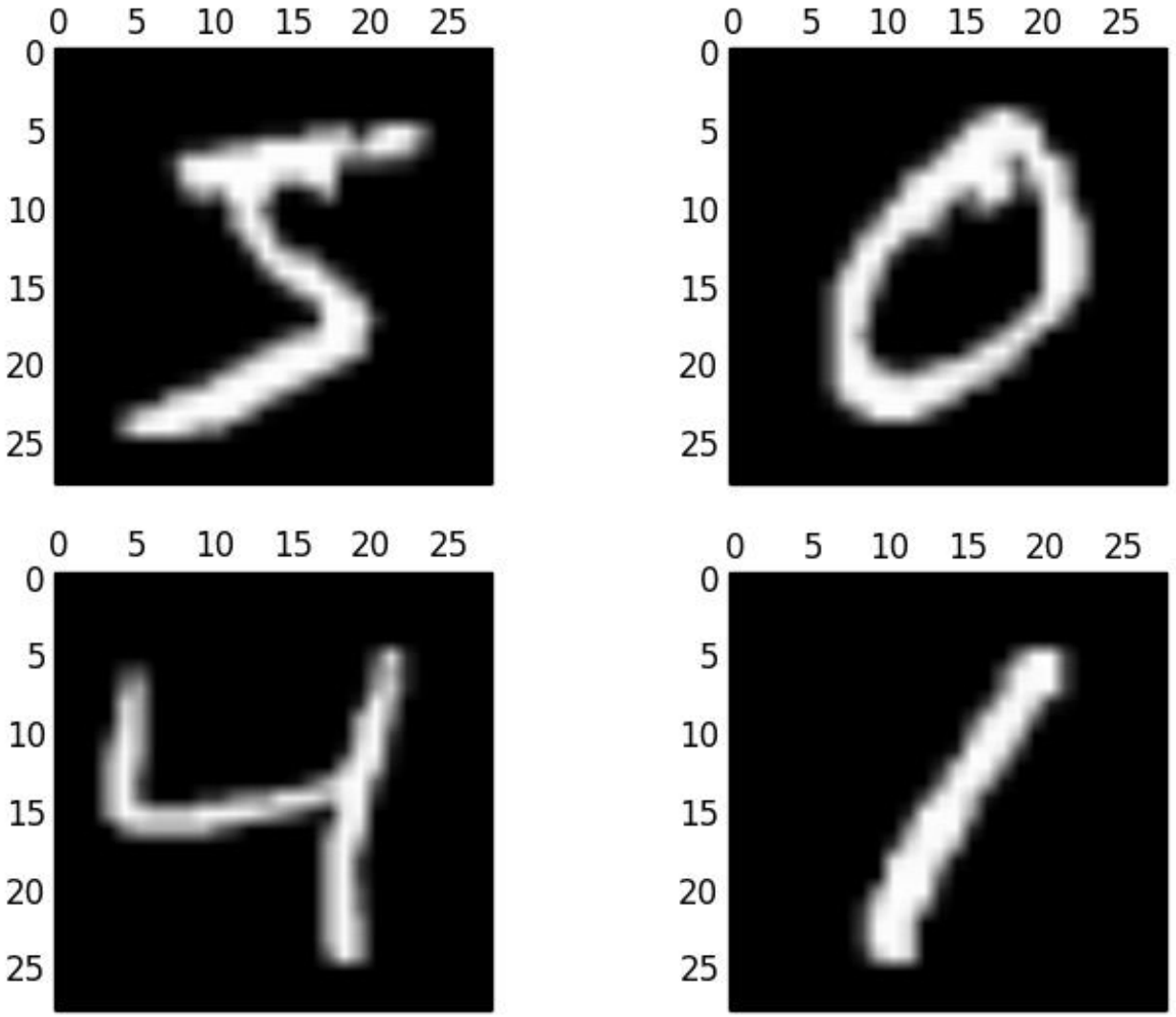
https://github.com/yungbyun/neuralnetworks

So, now can go deeper.

# MNIST

Modified National Institute of Standards and Technology (USA)

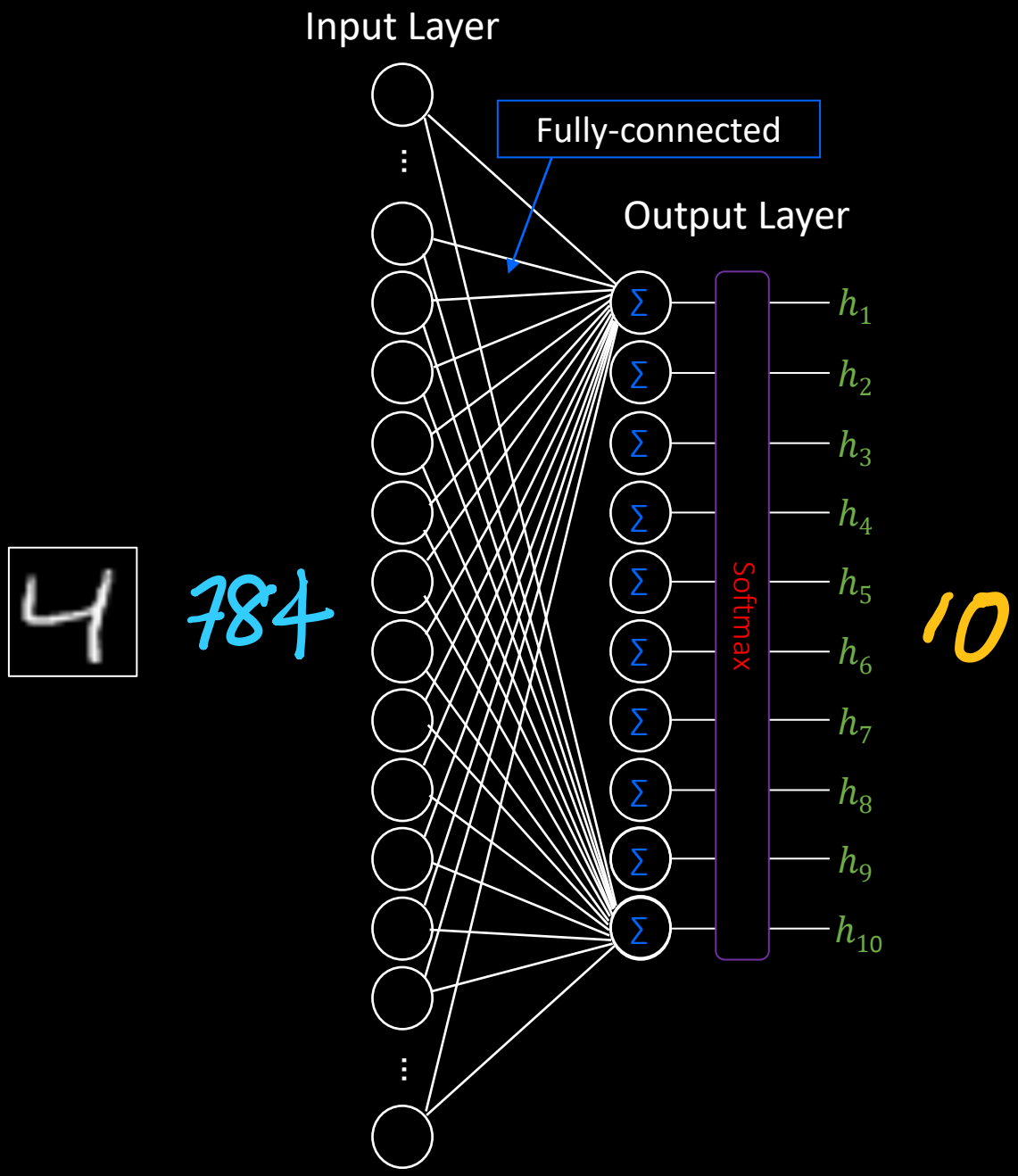
# MNIST

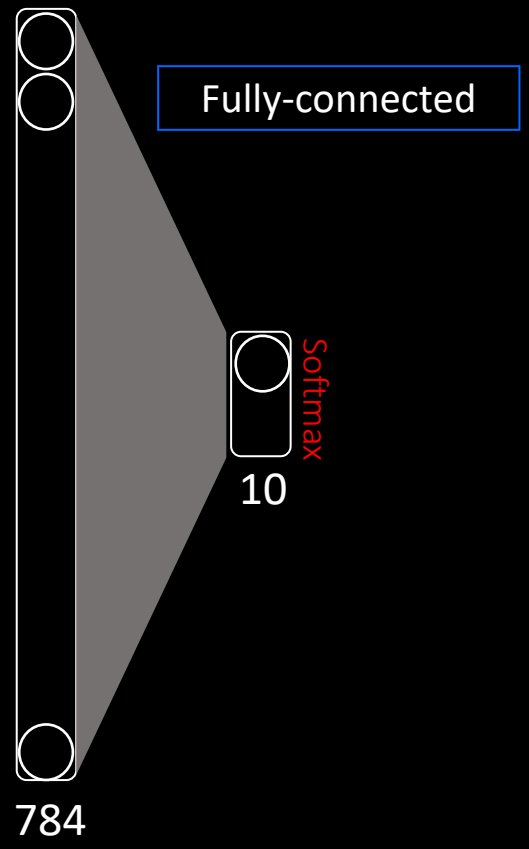

0 5 10 15 20 25
0
5
10
15
20
25

# (Lab) 20.py

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

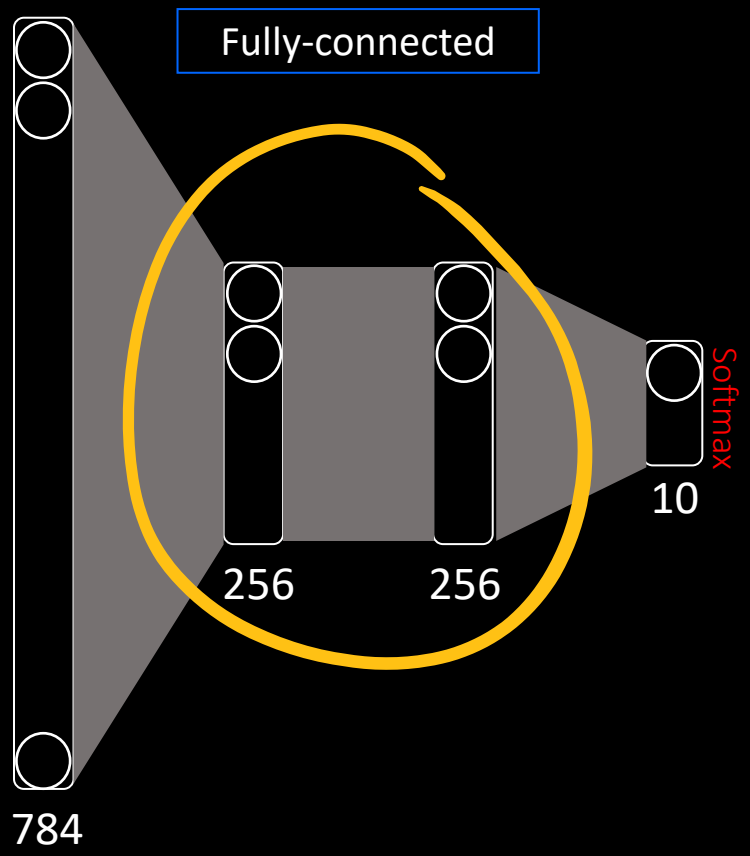
https://github.com/yungbyun/neuralnetworks


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


Fully-connected
Softmax
10
784

# (Lab) 21.py

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



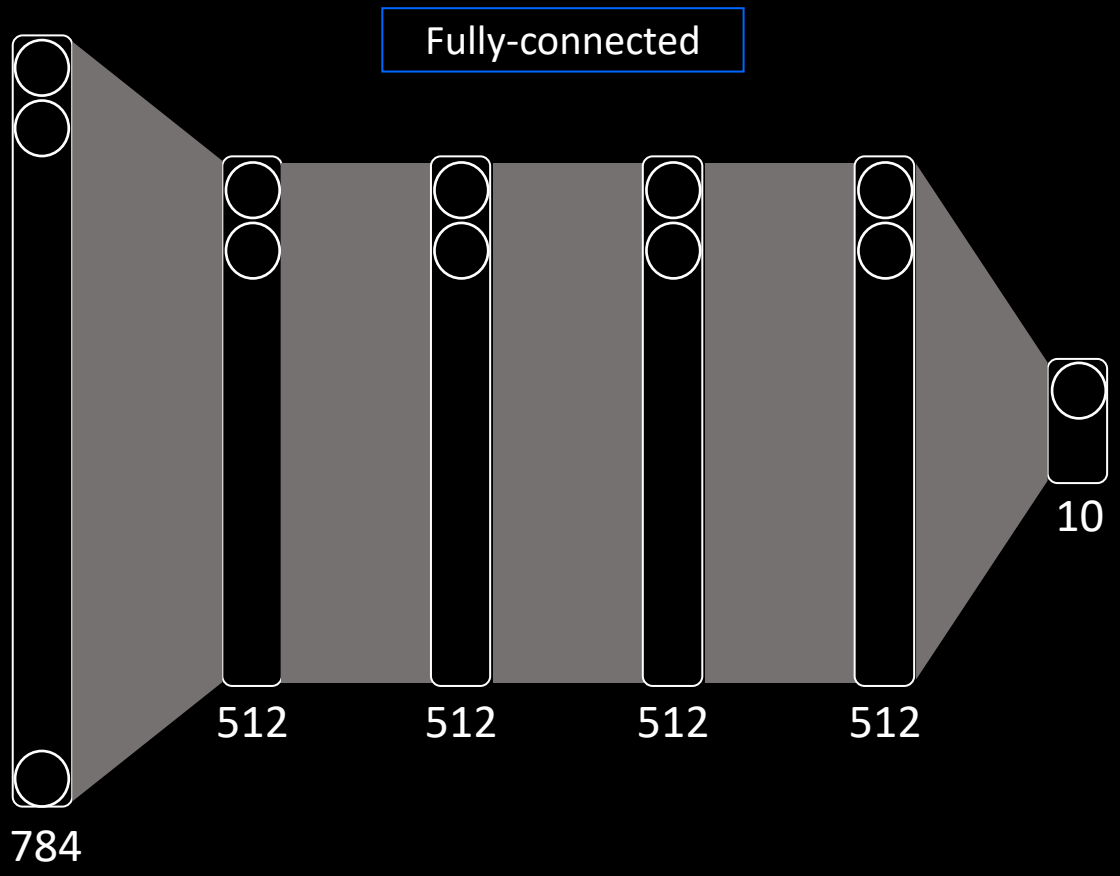

https://github.com/yungbyun/neuralnetworks

# (Lab) 22.py

- Applying initialization method for $w$ and $b$, not randomly
- 97.23% of accuracy

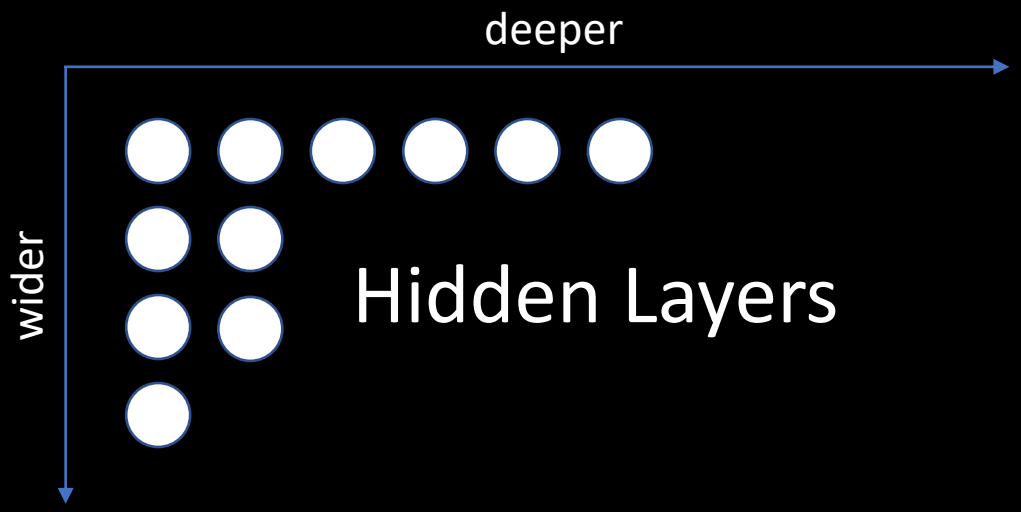
https://github.com/yungbyun/neuralnetworks

# (Lab) 23.py




- Applying initialization method for $w$ and $b$, not randomly
- 6-layer deep neural networks
- 97.83% of accuracy

# Decision Boundry

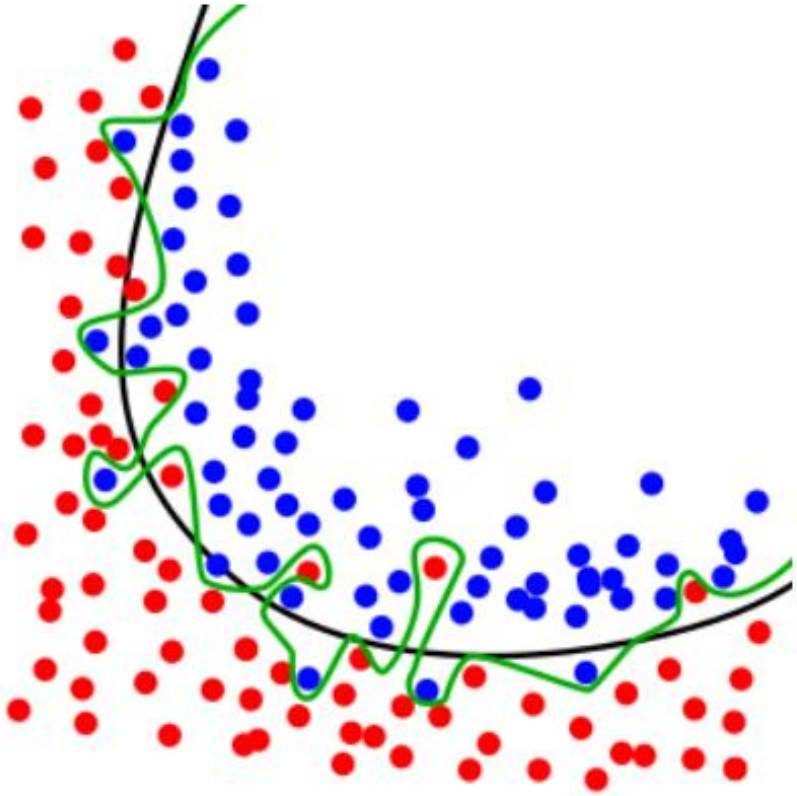



“

Lots of neurons (connections, synapses) give us so complicated decision tree

# Which do you think is desirable decision boundary?

View from above



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

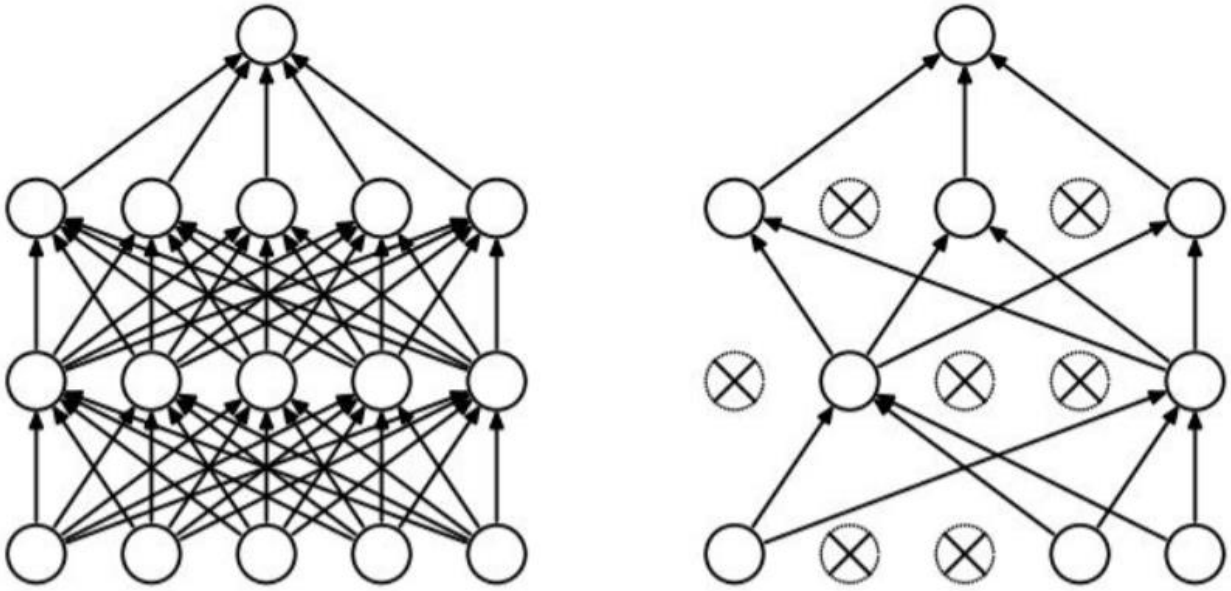
https://elitedatascience.com

# Overfitting and drop-out

- The deeper the network is, the more the decision boundary is complex.
- Good at learning data but errors for testing data → overfitted to the training data
- Making it less complex by drop-out some neurons while learning.

# Regularization: **Dropout**

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



(a) Standard Neural Net

(b) After applying dropout.

*[Srivastava et al., 2014]*

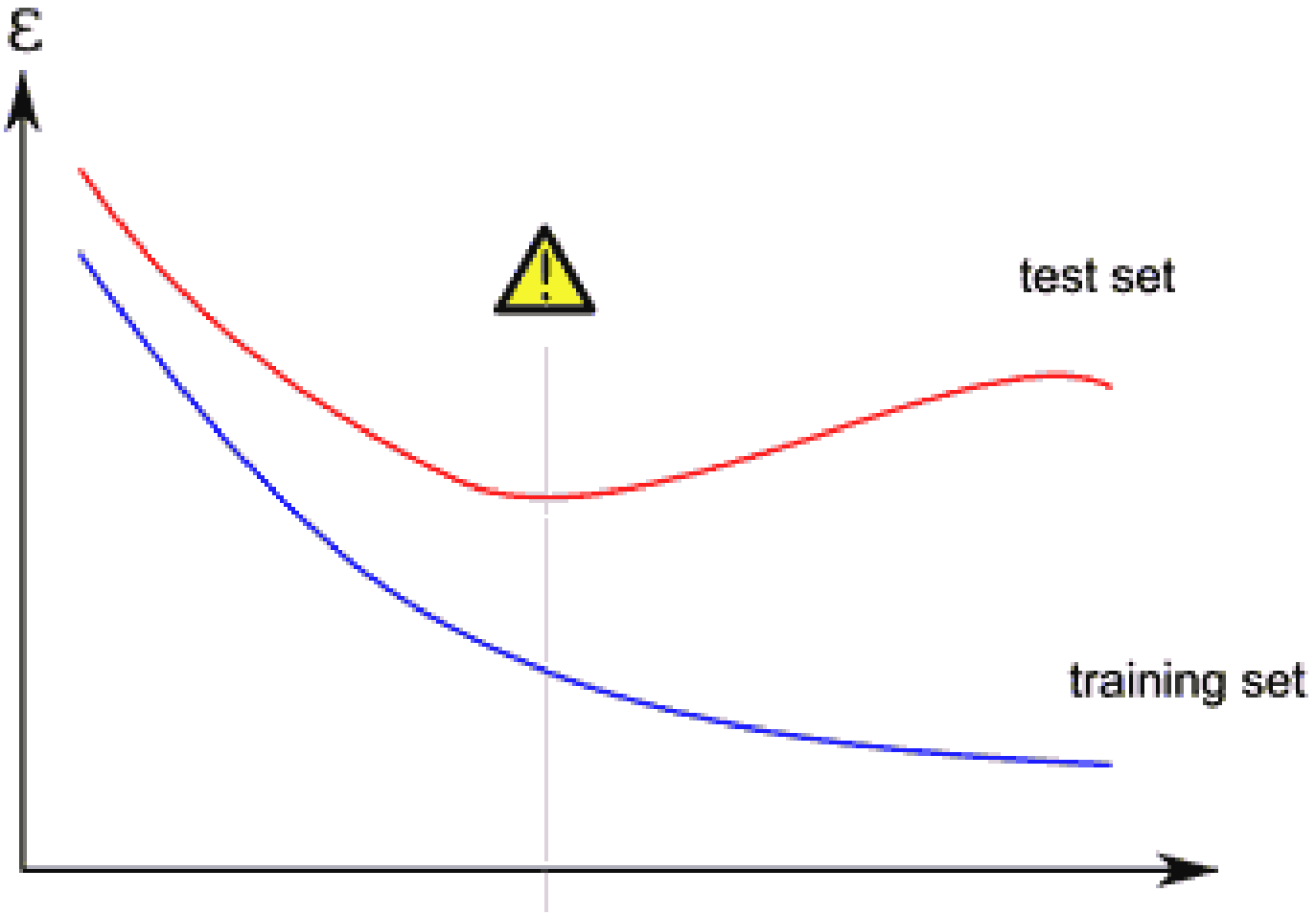
# (Lab) 24.py

- Applying dropout
- 98.13% (←97.83%) of recognition accuracy

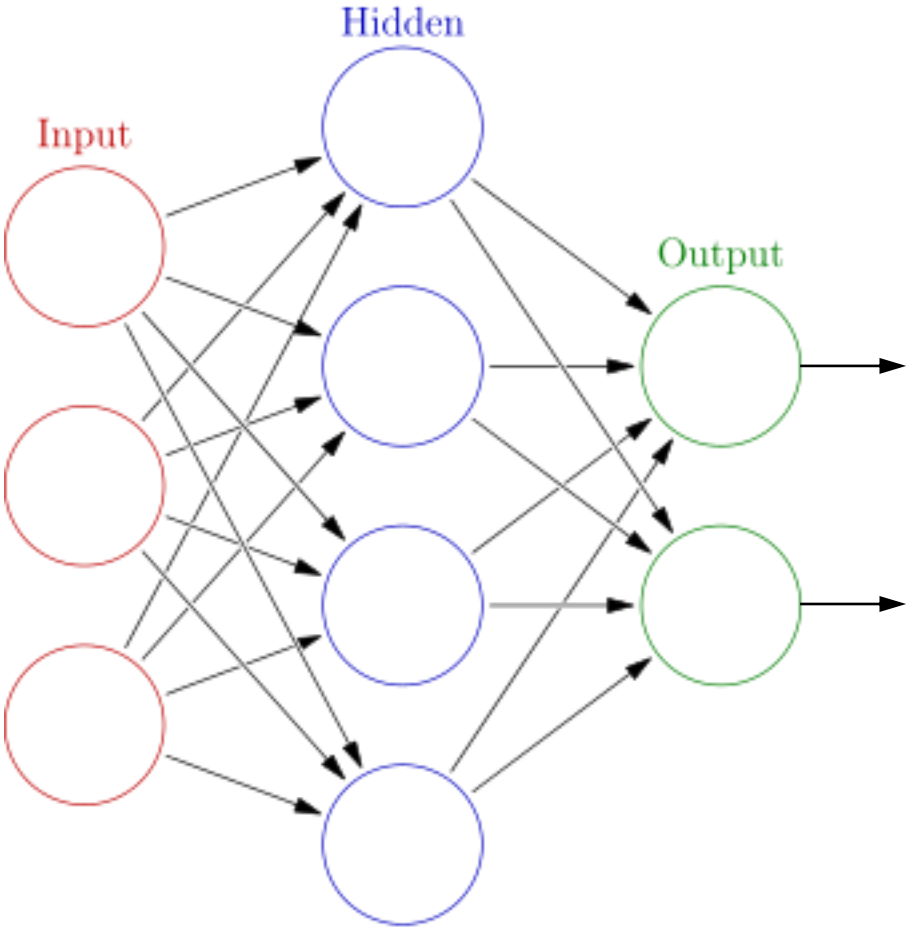
# How to Prevent Overfitting

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

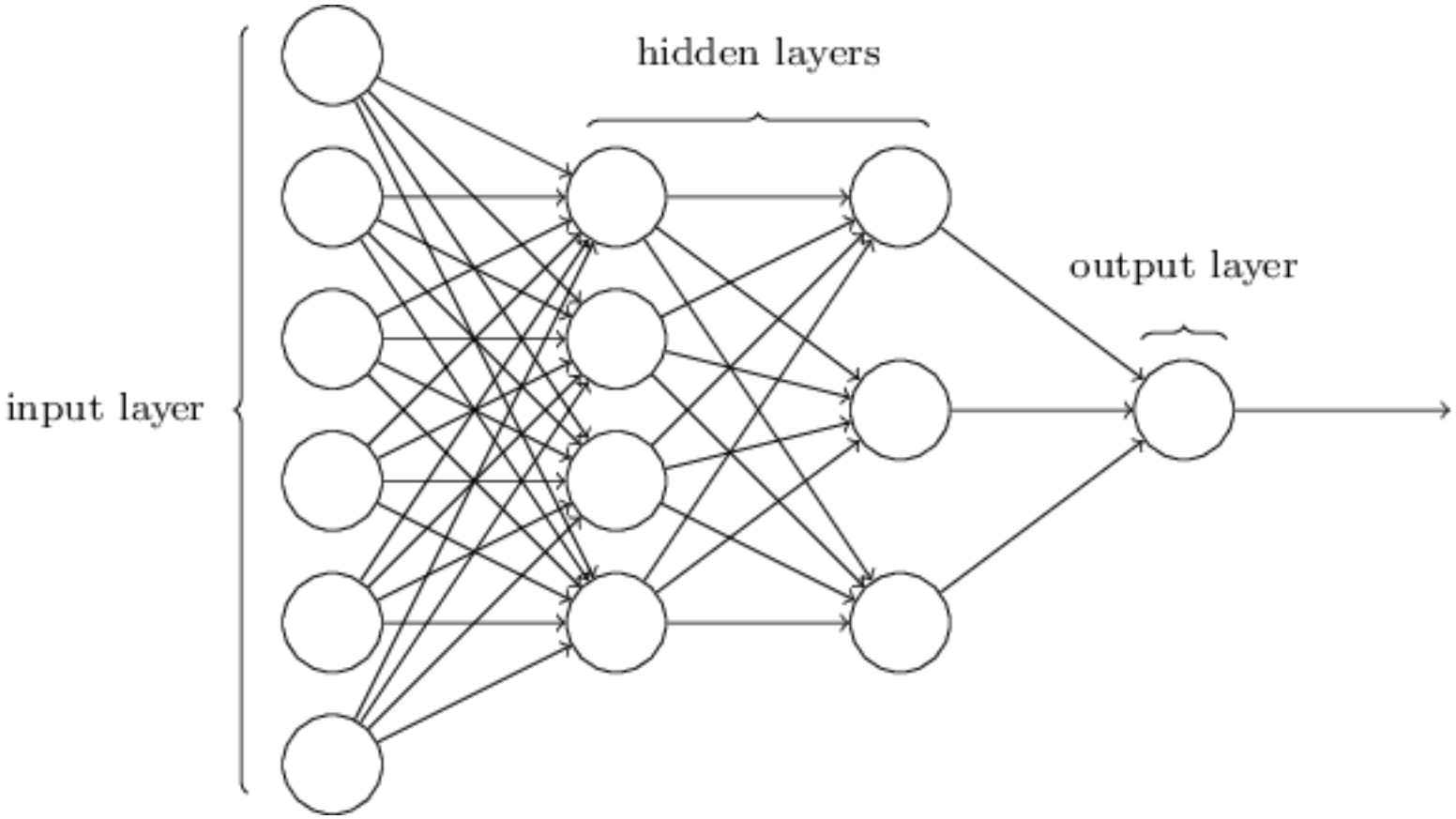
# Early stopping

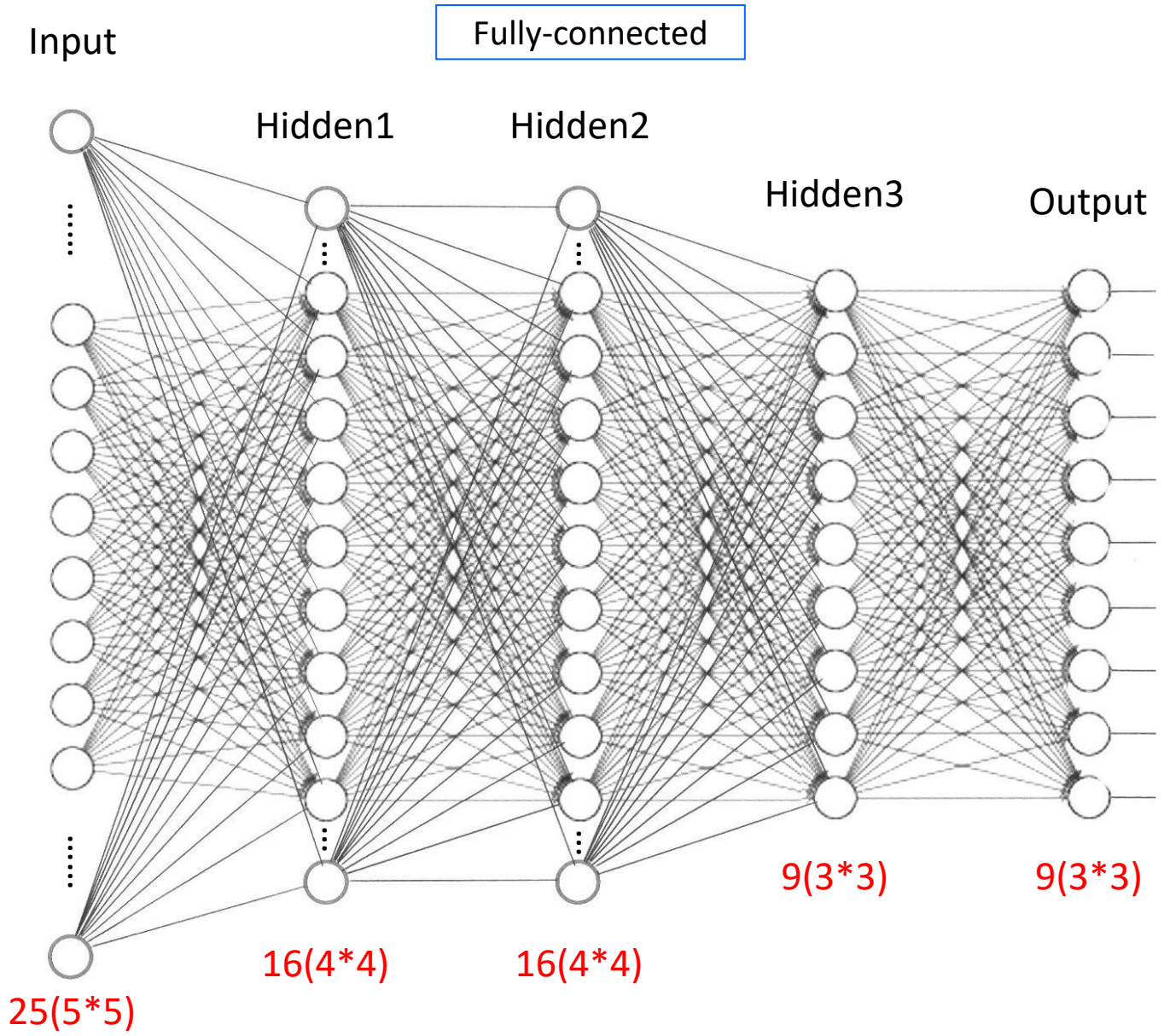


Fully-connected



Fully-connected

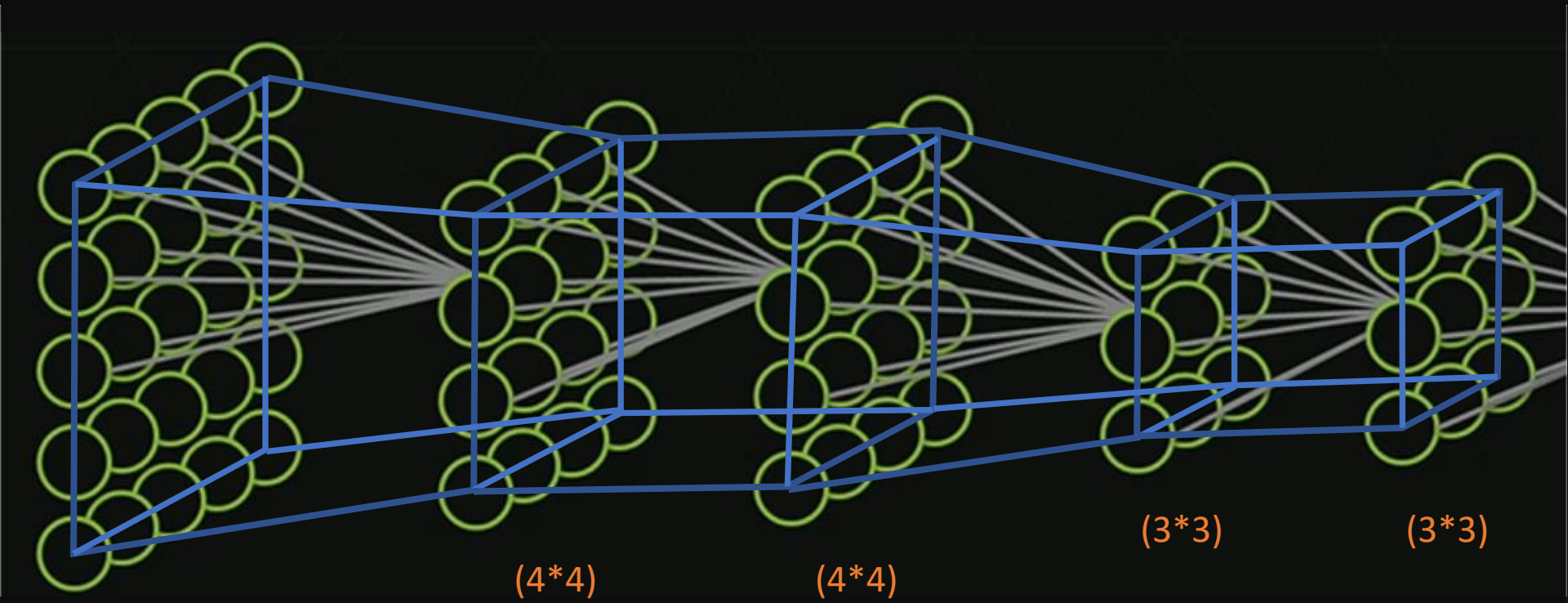




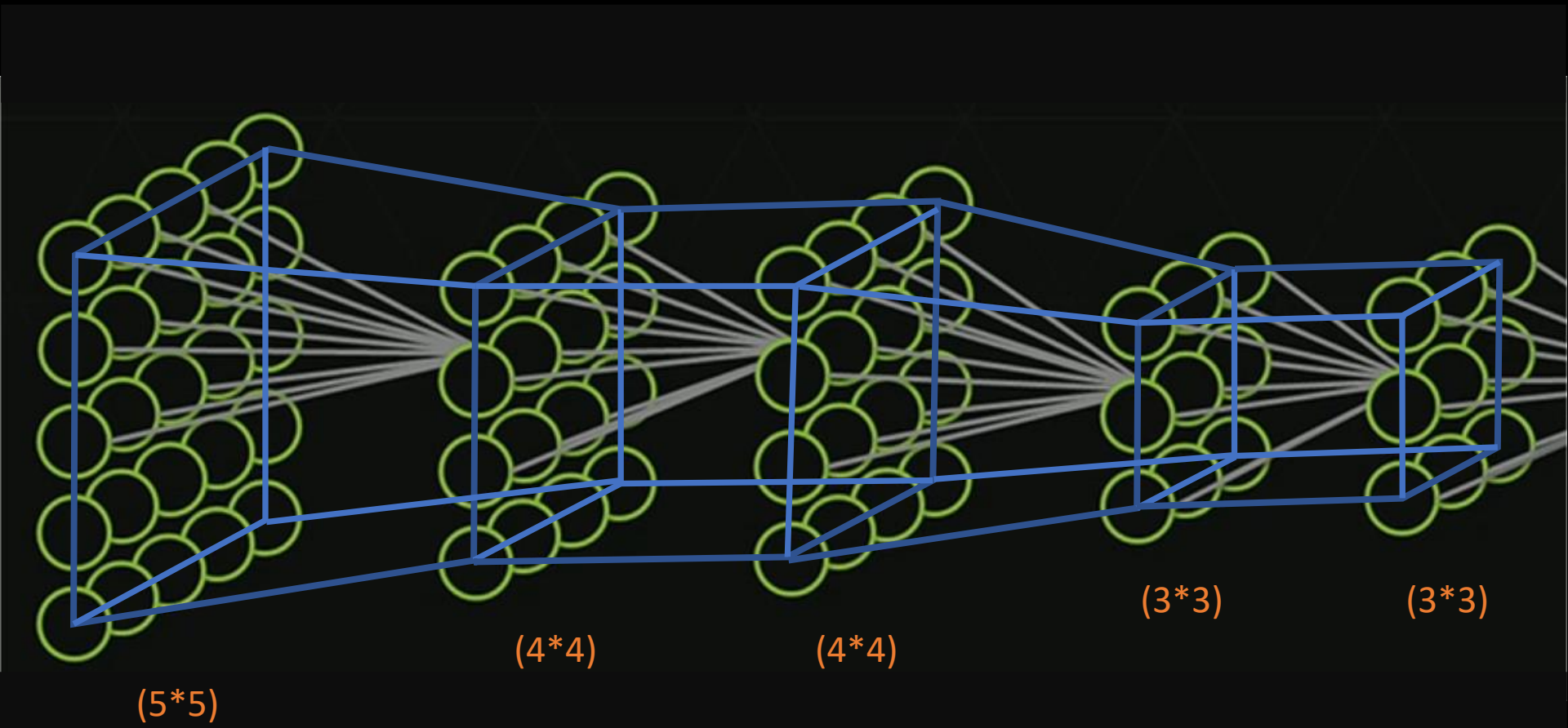

Fully connected, then how many connections(synapses, parameters) are there?

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

Fully-connected

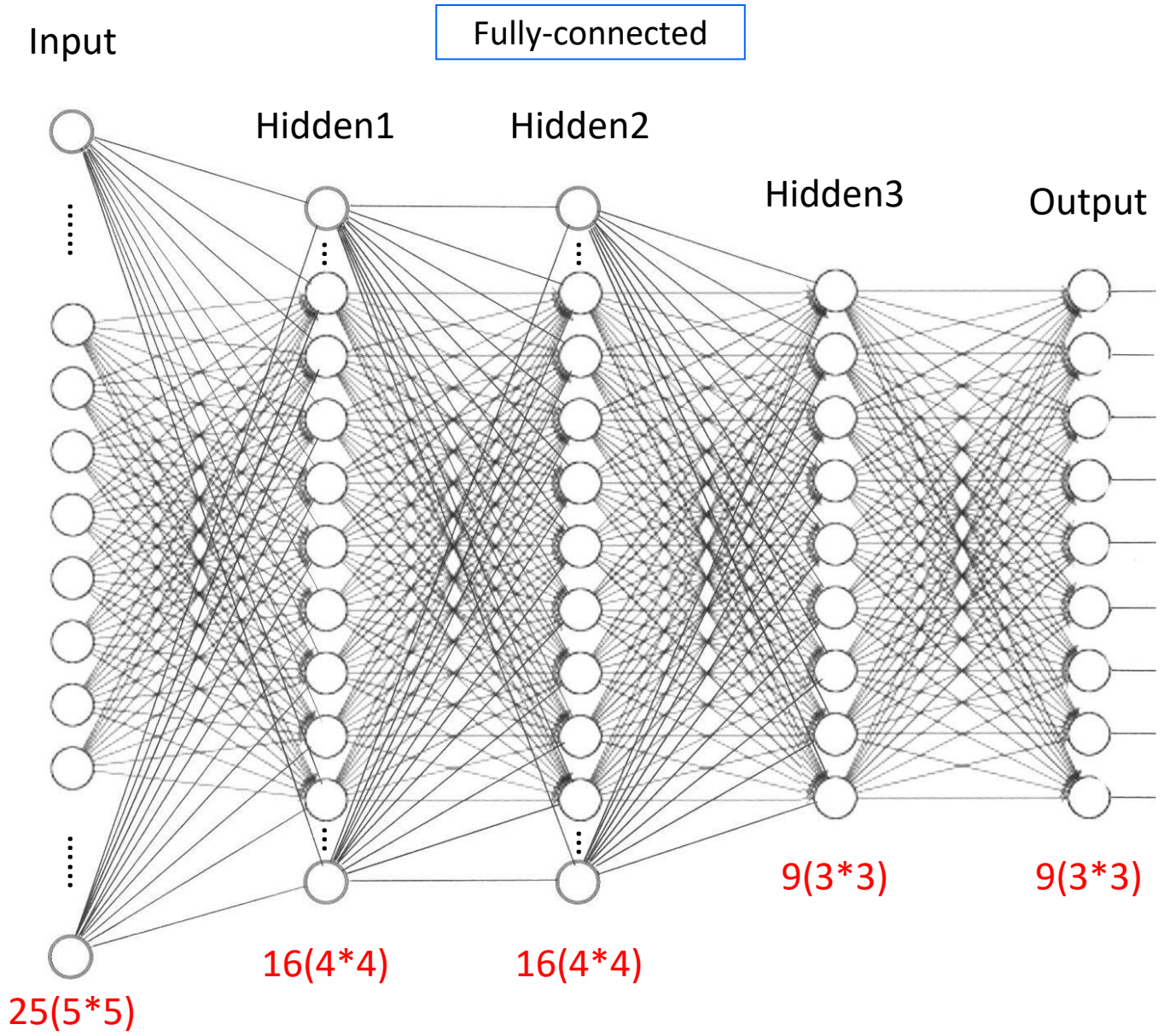


Fully-connected




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
- Activation functions (ReLU)
- Weight initialization methods
- Dropout (2014)
- Big data
- GPU

Fully-connected

# FCNN

Any problem?