AI and Deep Learning

Deep Learning

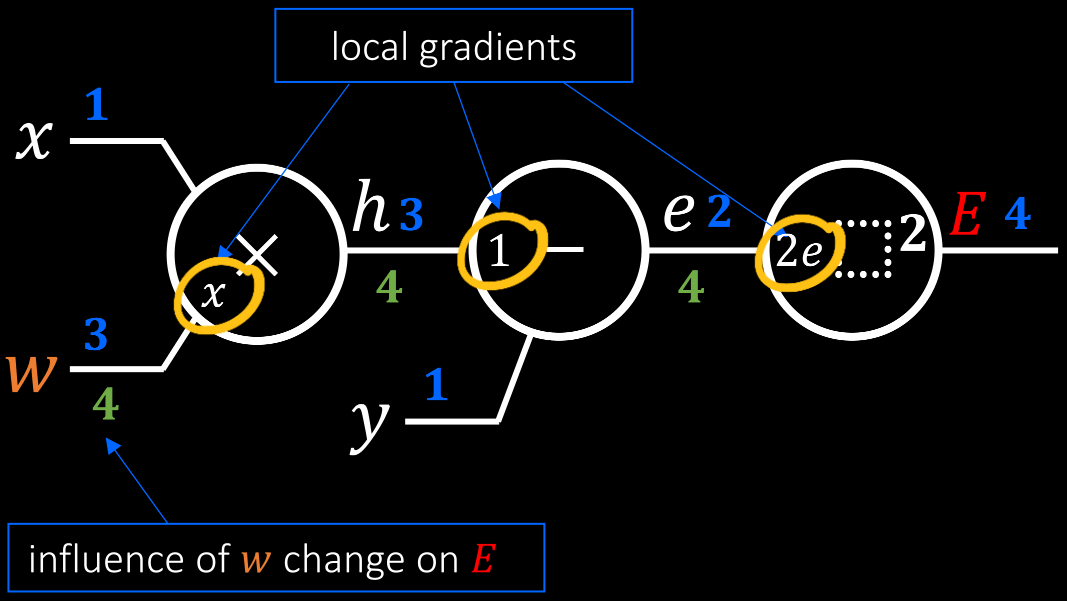
Jeju National University

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



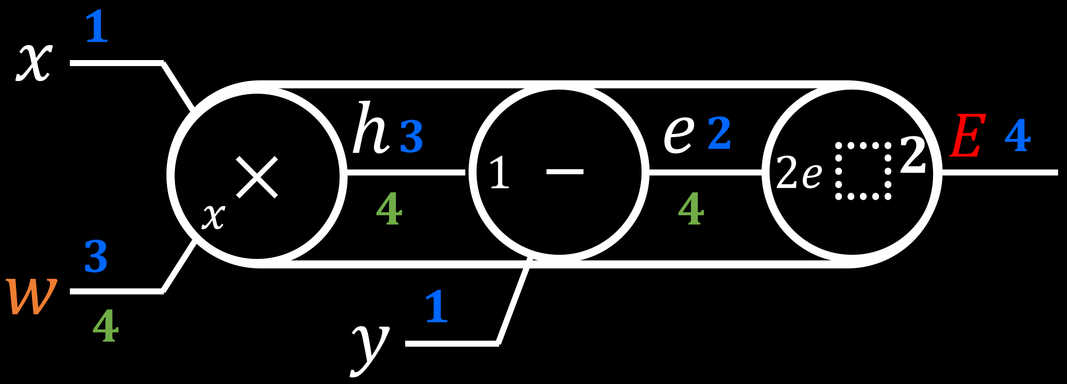
local gradients

influence of w change on E

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

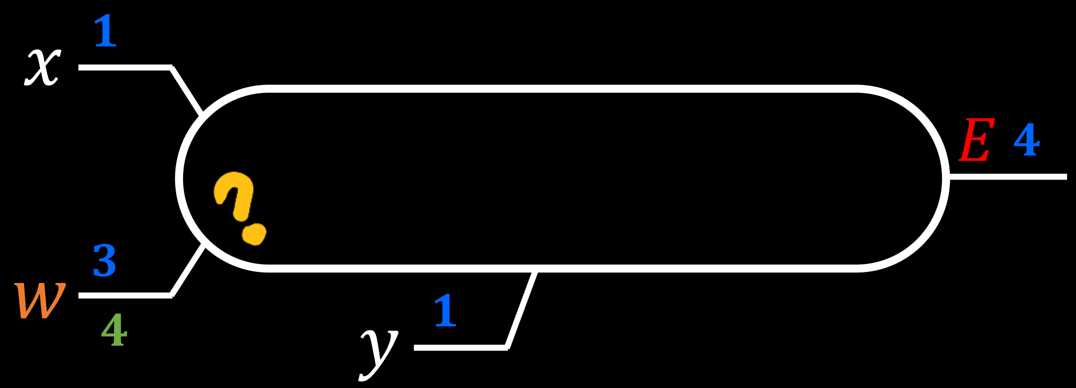
Influence of w change on E

Merging gates

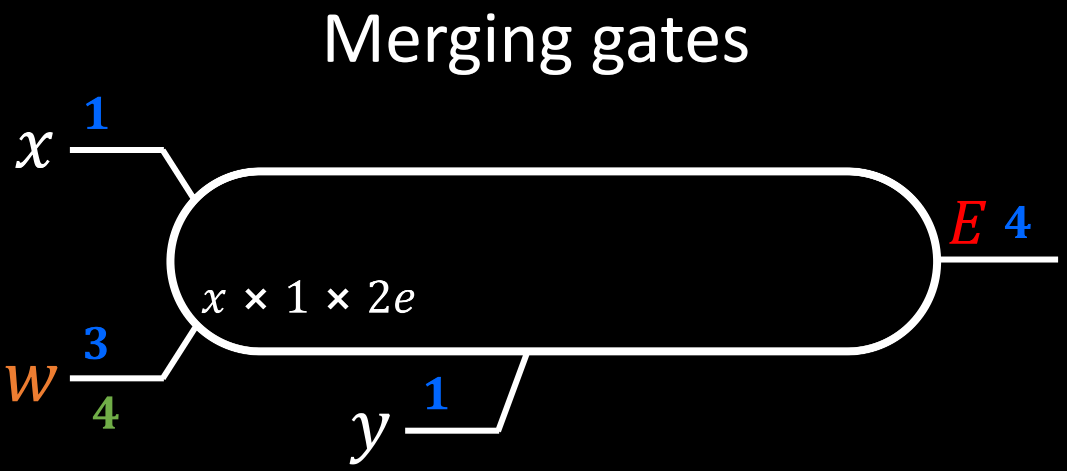


Influence of w change on E

Merging gates



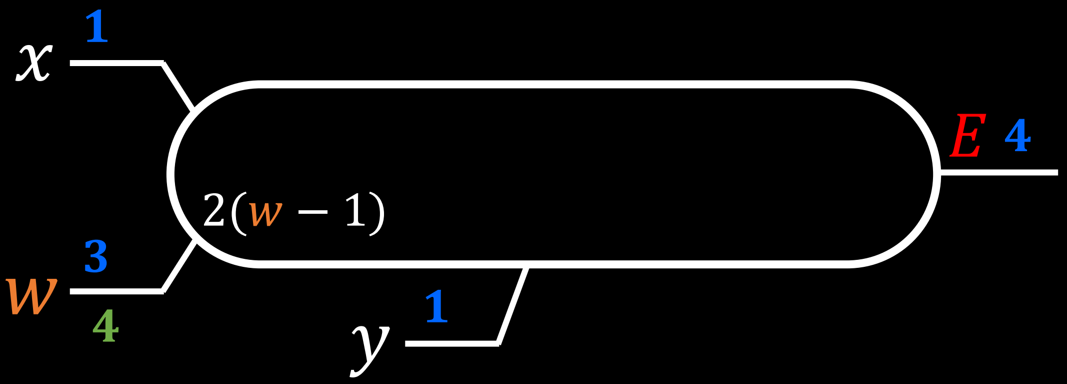
Influence of w change on E



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

Influence of w change on E

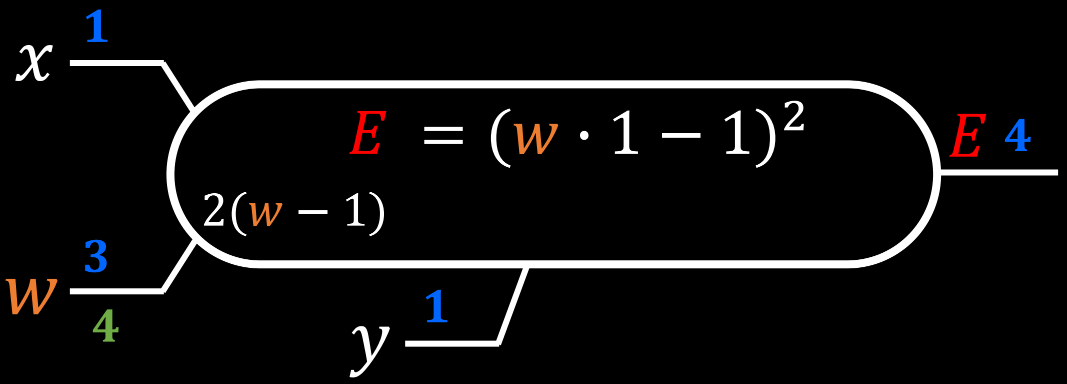
Merging gates



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

Influence of w change on E

Merging gates



Therefore, the local gradient is derivative of the E.

Influence of w change on E

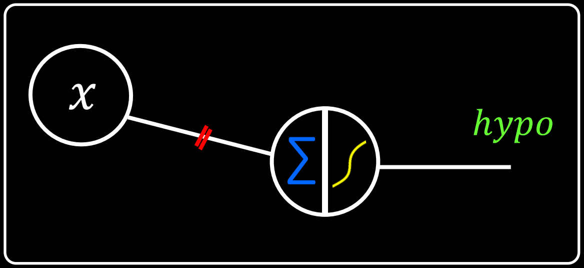
$$E = (w \cdot 1 - 1)^2$$

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



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

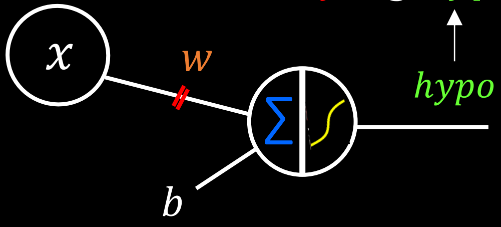
$$E = -y\log(hypo) - (1 - y)\log(1 - hypo)$$

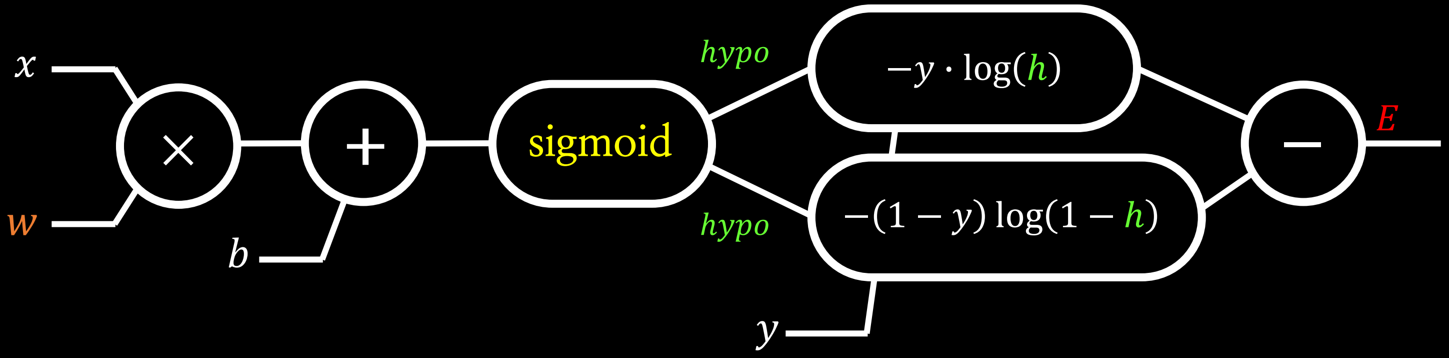
Binary Cross Entropy

Computational Graph

Binary Cross Entropy 오류함수(loss function)

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



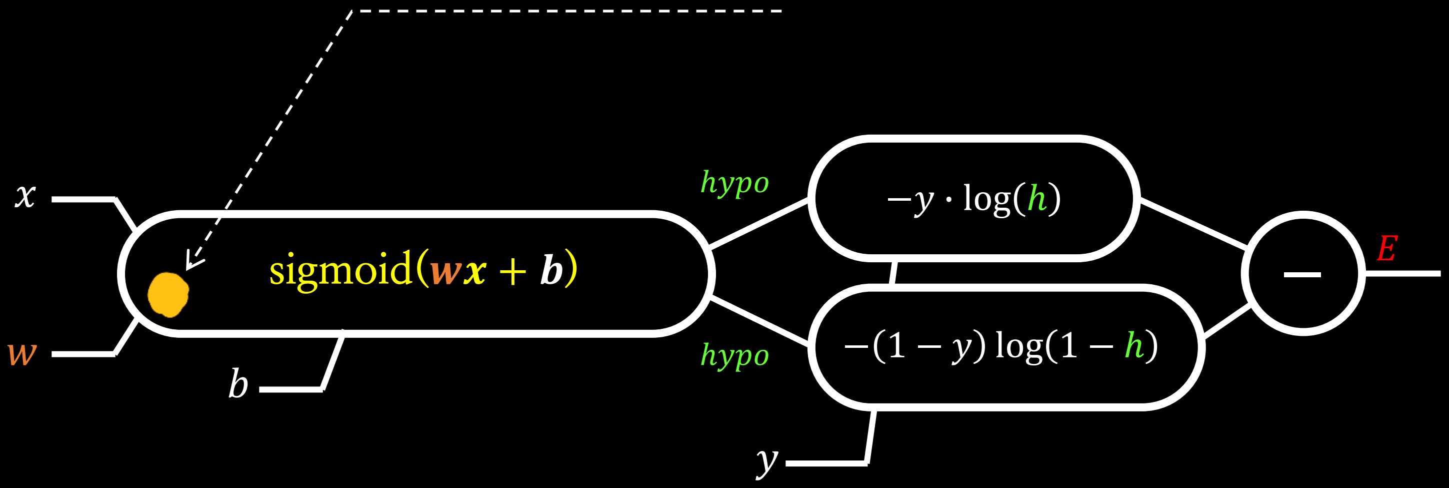


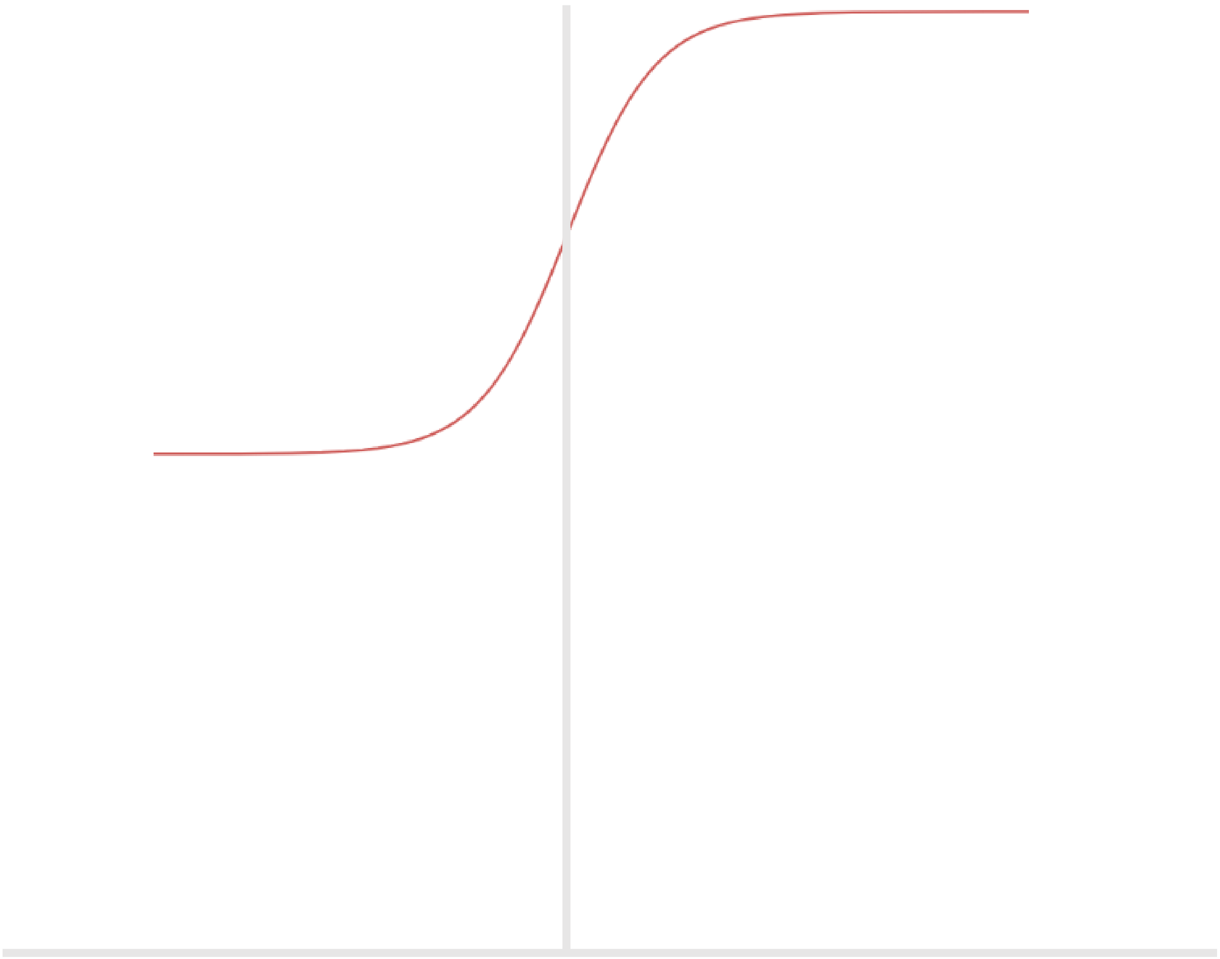
$$\frac{\partial E}{\partial w} =$$

Computational Graph

Merging gates

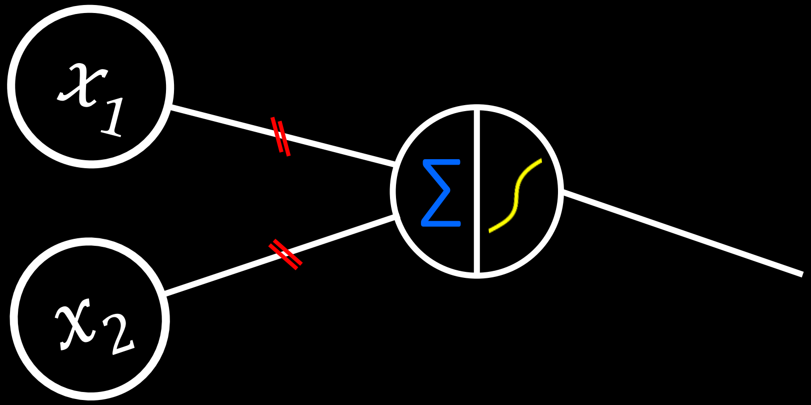
How we get the local gradient of the merged gate(sigmoid)?



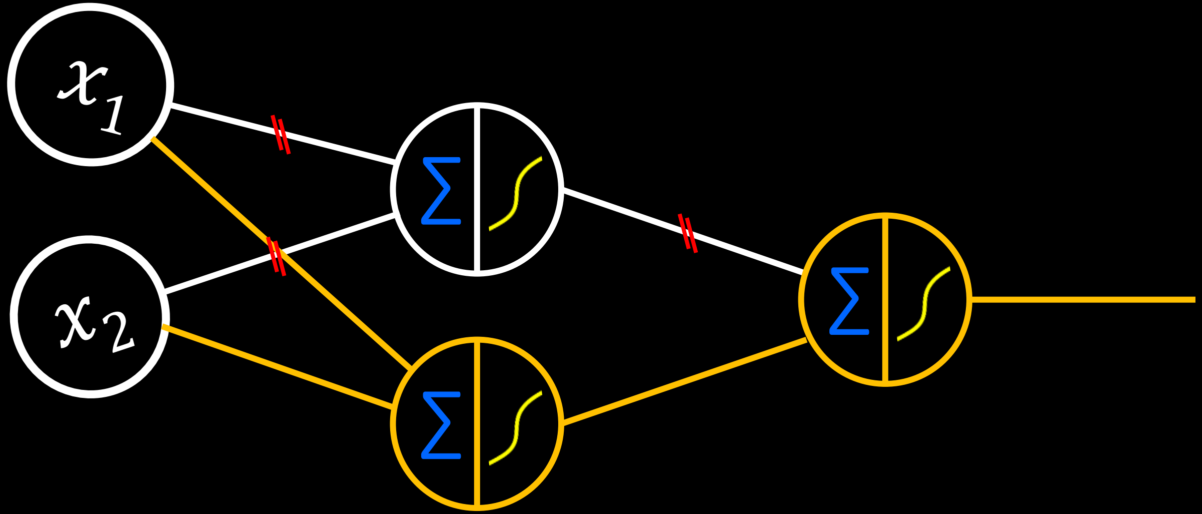


$$\frac{\partial sigmoid}{\partial w} = sigmoid(1 - sigmoid)$$

1 Neuron(2-layer)

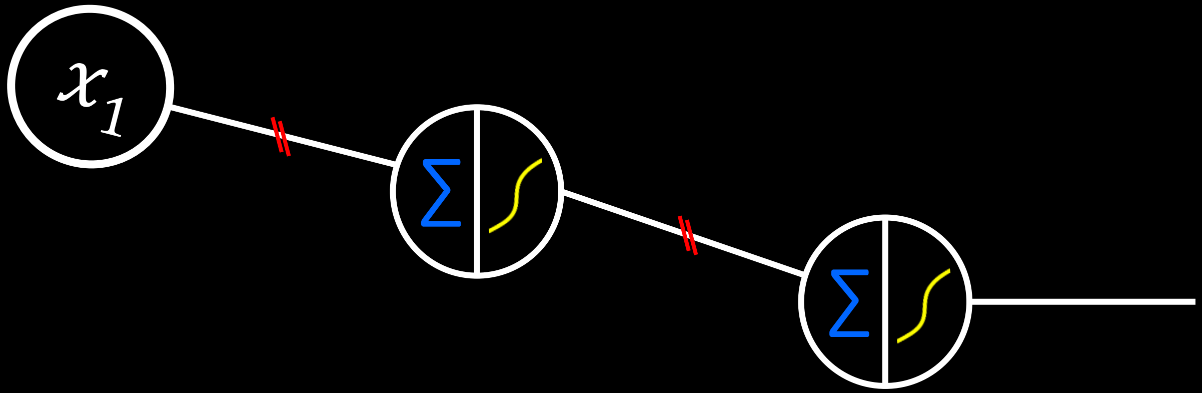


3-layer NN



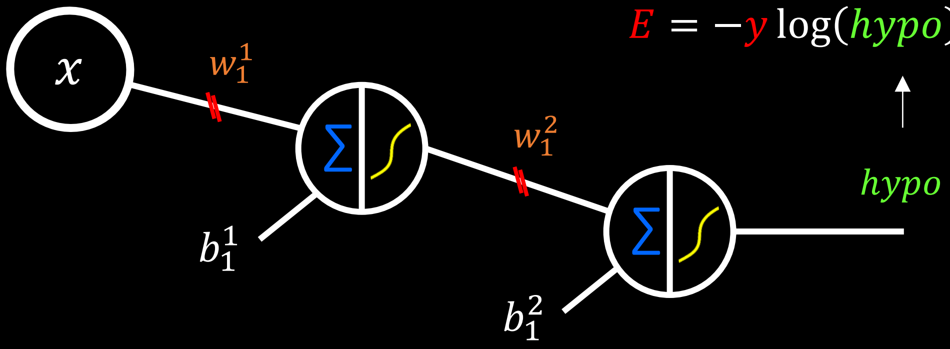
Why? what for?

3-layer NN (simplified)

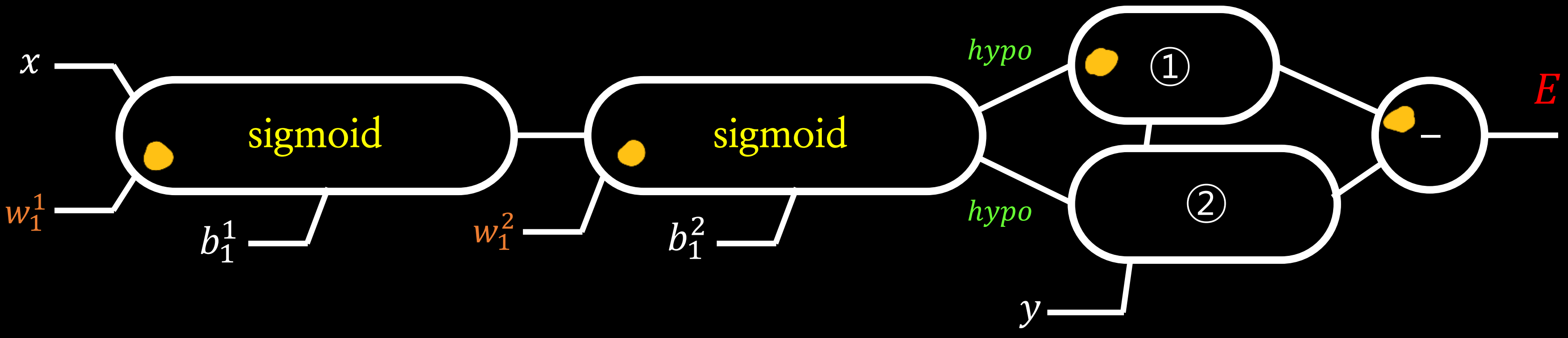


Influence of w change on E

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



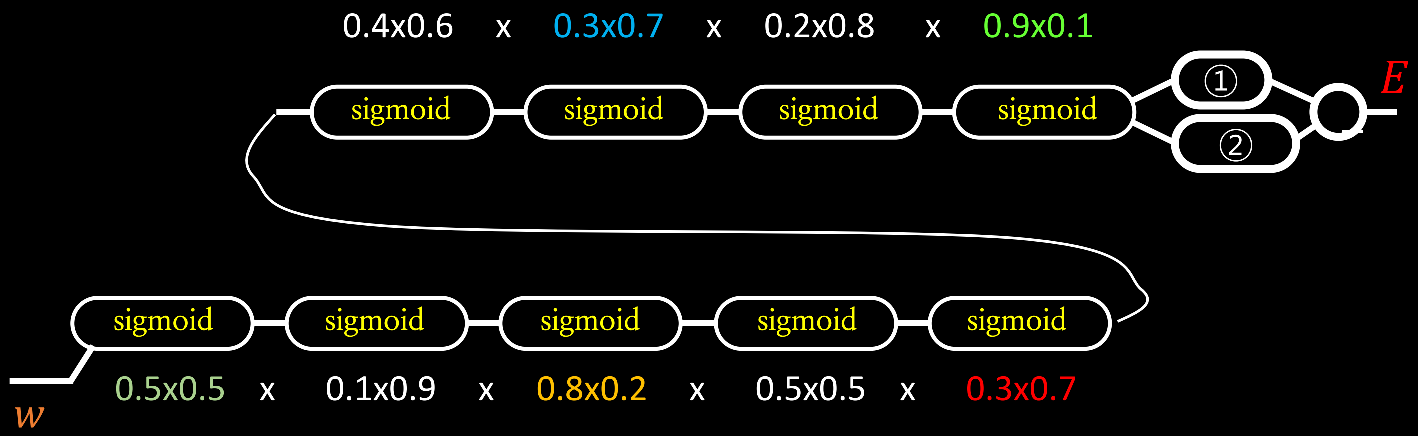
3-layer NN → 2 sigmoid



10-layer Neural Network

The giant monster, computational graph!

Influence of *w* change on *E*



0.4x0.6 x 0.3x0.7 x 0.2x0.8 x 0.9x0.1
sigmoid
sigmoid
sigmoid
sigmoid
①
②
E
sigmoid
sigmoid
sigmoid
sigmoid
sigmoid
w
0.5x0.5 x 0.1x0.9 x 0.8x0.2 x 0.5x0.5 x 0.3x0.7

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

Hint: chain rule!
Multiplication of the 9 derivative of *sigmoid*

Vanishing Gradient

- The **derivative** of a sigmoid function is sigmoid x (1-sigmoid)
- 2 multiplication of sigmoid for a single neuron, 18 multiplications for 9 connected neurons
- Each sigmoid gives us the value between 0 and 1.

Ex) 0.5x0.5 x 0.1x0.9 x 0.8x0.2 x 0.5x0.5 x 0.3x0.7 x 0.4x0.6 x 0.3x0.7 x 0.2x0.8 x 0.9x0.1
= 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**
- $w = w - \alpha \cdot$ Slope (≈ 0)
- $b = b - \alpha \cdot$ Slope (≈ 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 AI

(~ 2006)

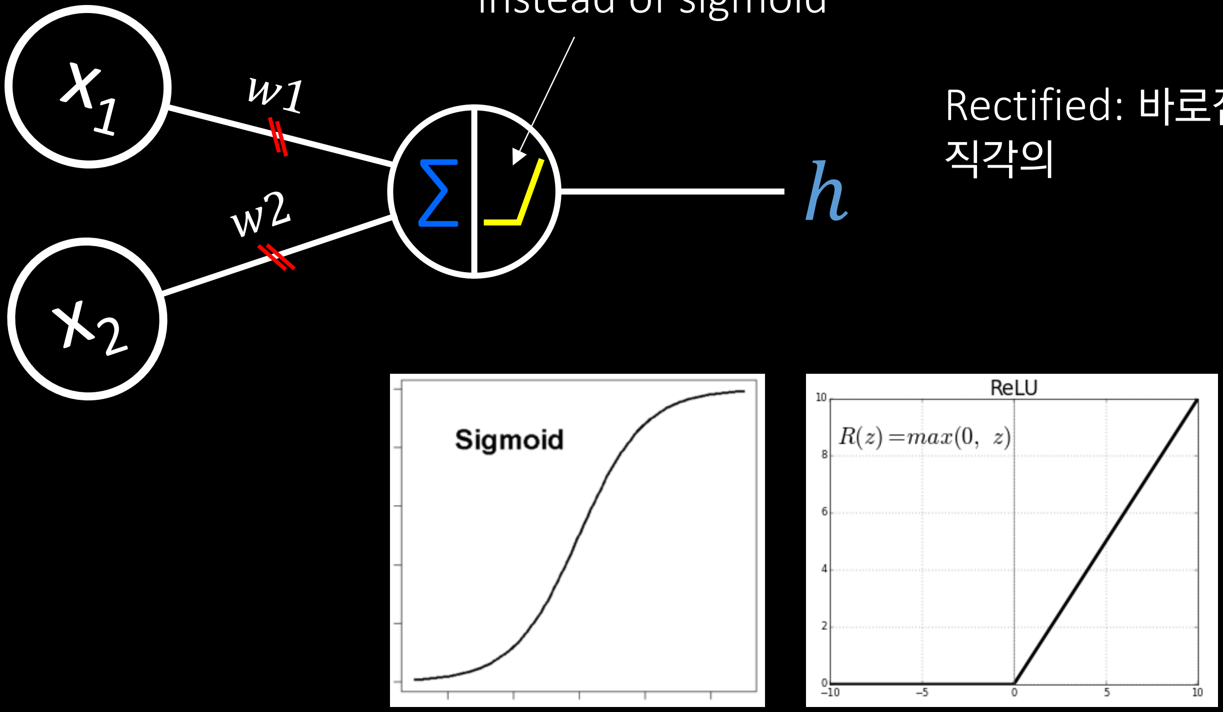
since back-propagation by Hinton in 1986

ReLU

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

using ReLU (Rectified Linear Unit)
instead of sigmoid

Rectified: 바로잡은,
직각의



(Lab) 19.py

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

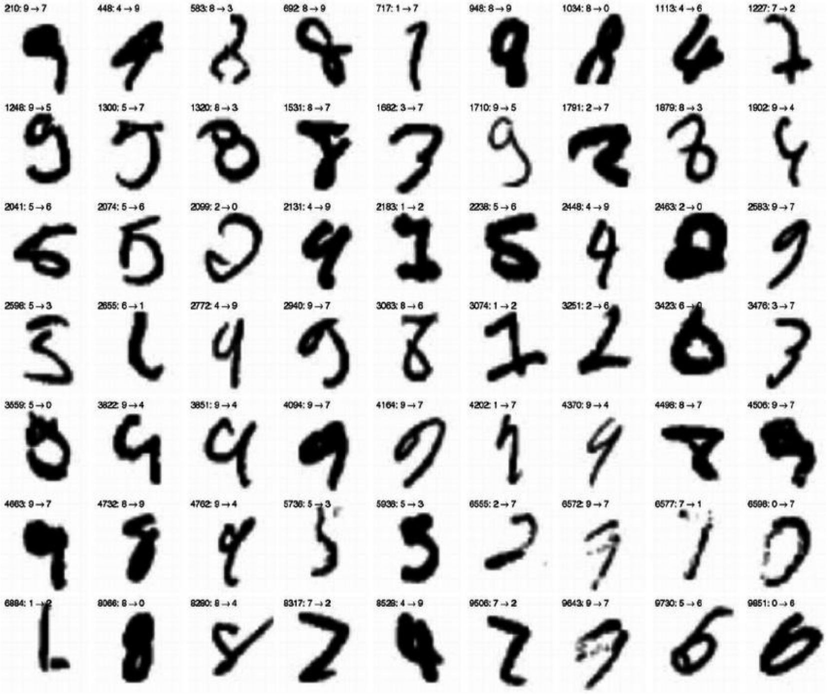
https://github.com/yungbyun/neuralnetworks

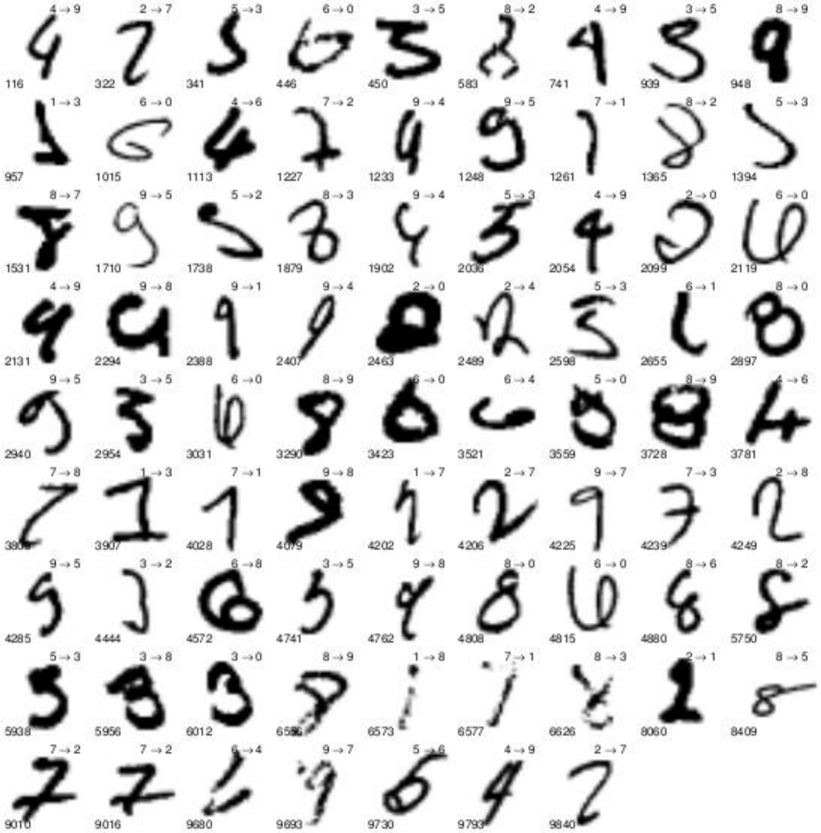
So, now can go deeper.

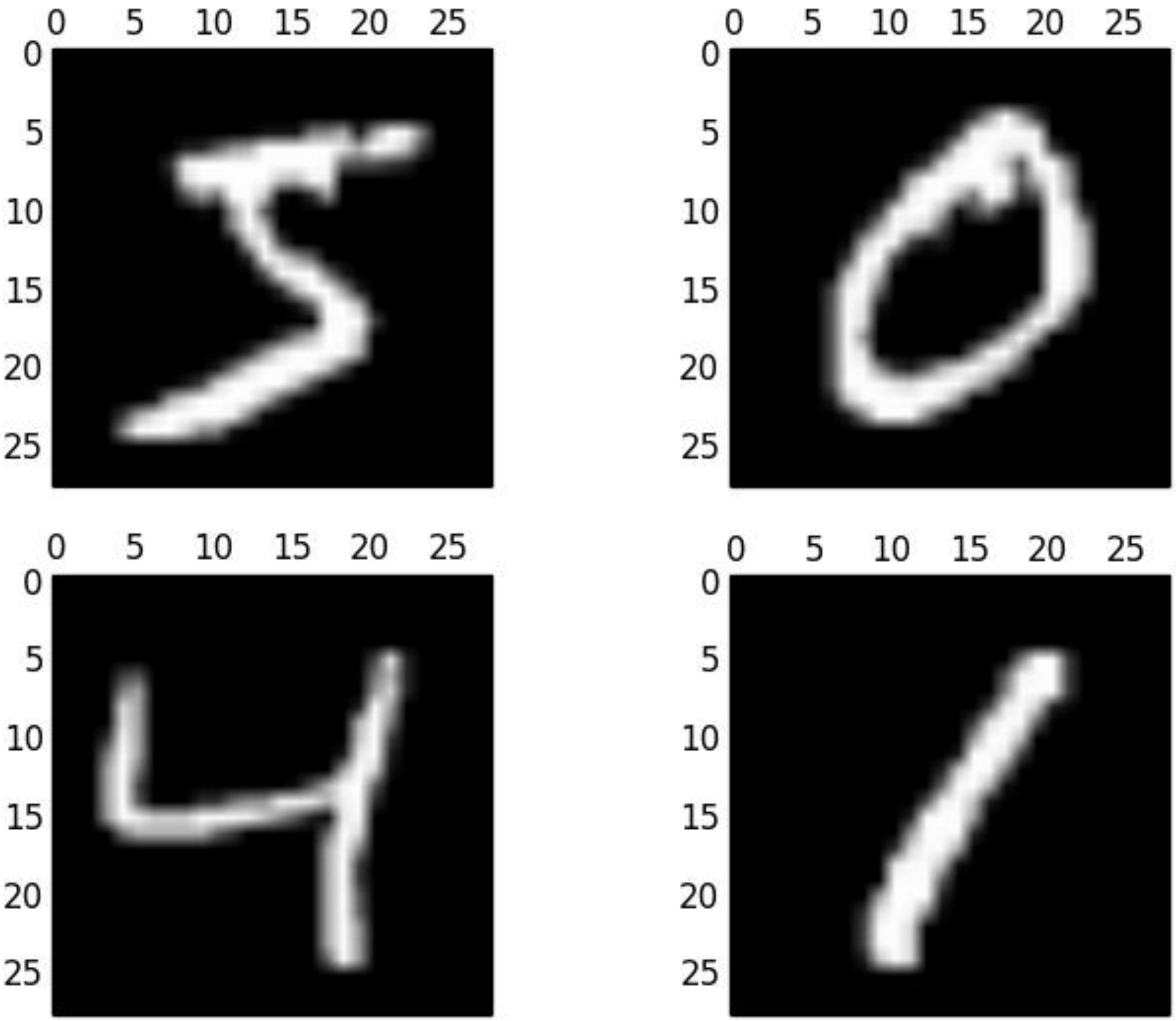
MNIST

Modified National Institute of Standards and Technology (USA)

MNIST


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
5736: 5→3
5938: 5→3
6555: 2→7
6572: 9→7
6577: 7→1
6598: 0→7
6884: 1→2
8066: 8→0
8290: 8→4
8317: 7→2
8528: 4→9
9506: 7→2
9643: 9→7
9730: 5→6
9851: 0→6

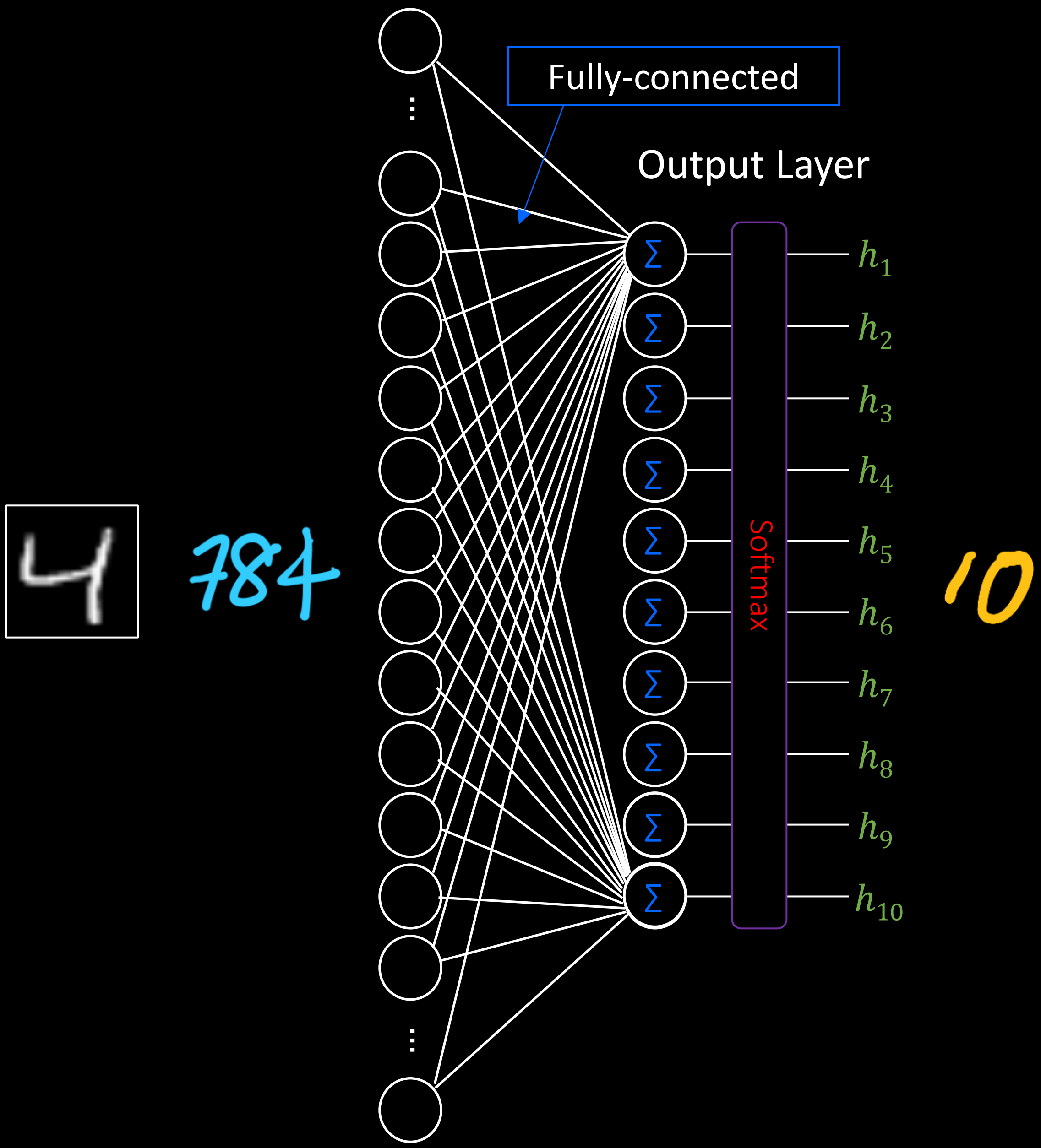

4→9 116
2→7 322
5→3 341
6→0 446
3→5 450
8→2 583
4→9 741
3→5 939
8→9 948
1→3 957
6→0 1015
4→6 1113
7→2 1227
9→4 1233
9→5 1248
7→1 1261
8→2 1365
5→3 1394
8→7 1531
9→5 1710
5→2 1738
8→3 1879
9→4 1902
5→3 2036
4→9 2054
2→0 2099
6→0 2119
4→9 2131
9→8 2294
9→1 2388
9→4 2407
2→0 2463
2→4 2489
5→3 2598
6→1 2655
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 6555
1→8 6573
7→1 6577
8→3 6626
2→1 8060
8→5 8409
7→2 9010
7→2 9016
6→4 9680
9→7 9693
5→6 9730
4→9 9793
2→7 9840


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
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 input dimension
- 10 classes (output: 0 ~ 9)
- Softmax
- 90.23% of recognition rate

https://github.com/yungbyun/neuralnetworks



Input Layer

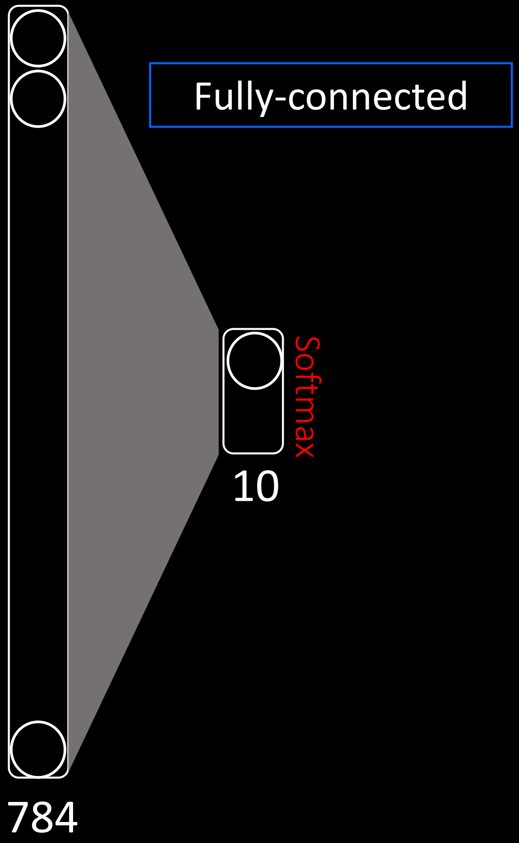
Fully-connected

Output Layer

Softmax

784

10



Fully-connected

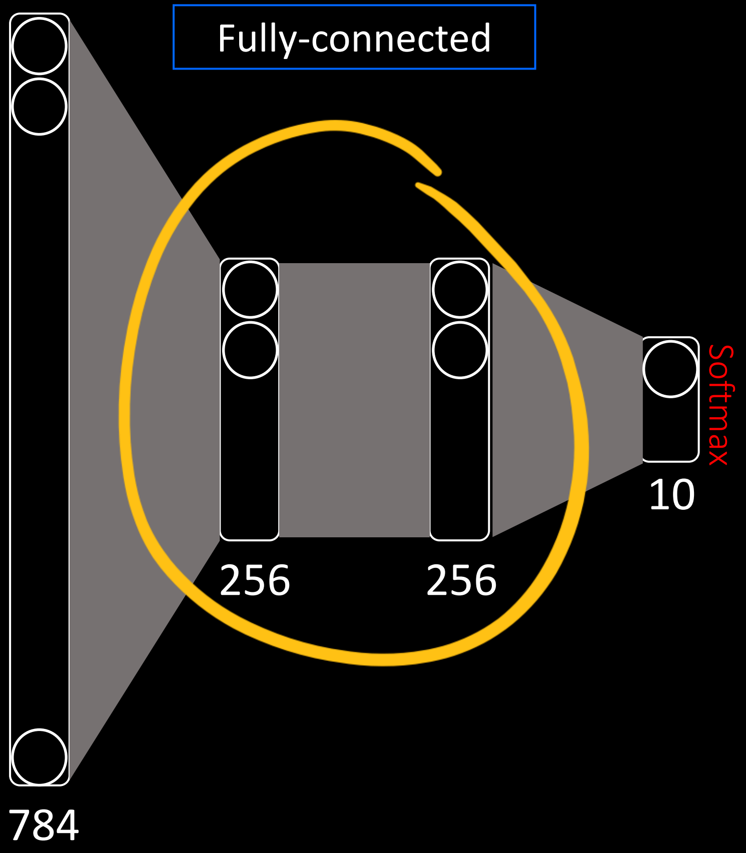
784

10

Softmax

(Lab) 21.py

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



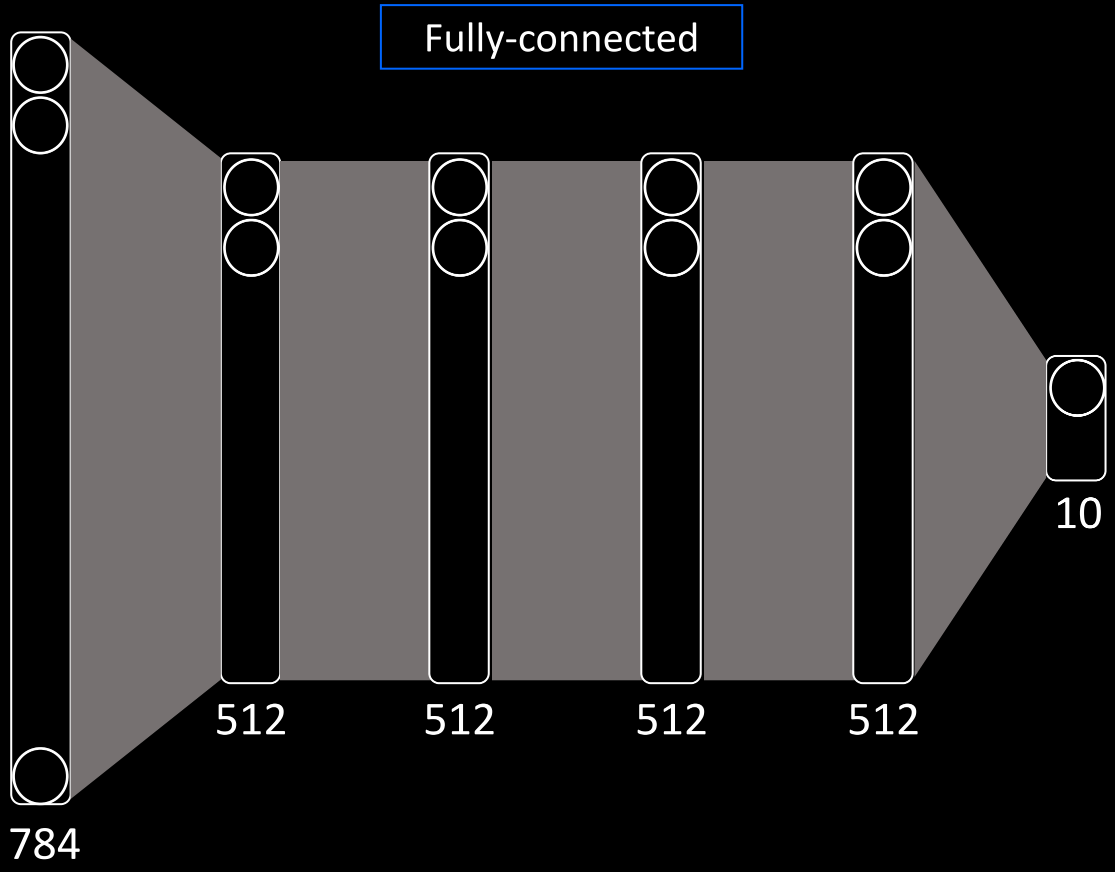
https://github.com/yungbyun/neuralnetworks

(Lab) 22.py

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

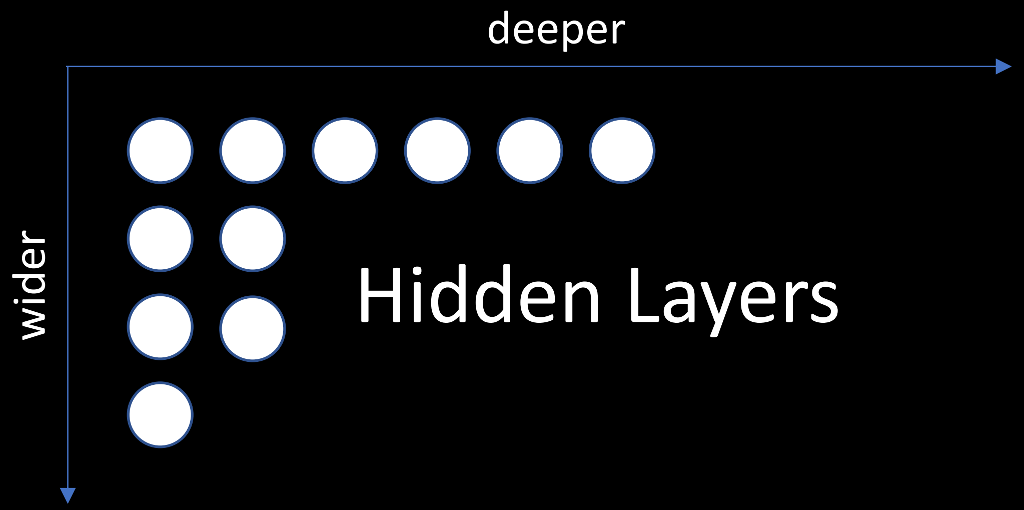
https://github.com/yungbyun/neuralnetworks

(Lab) 23.py



- Applying initialization
- 6-layer deep neural networks
- 97.83% of accuracy

Decision Boundry

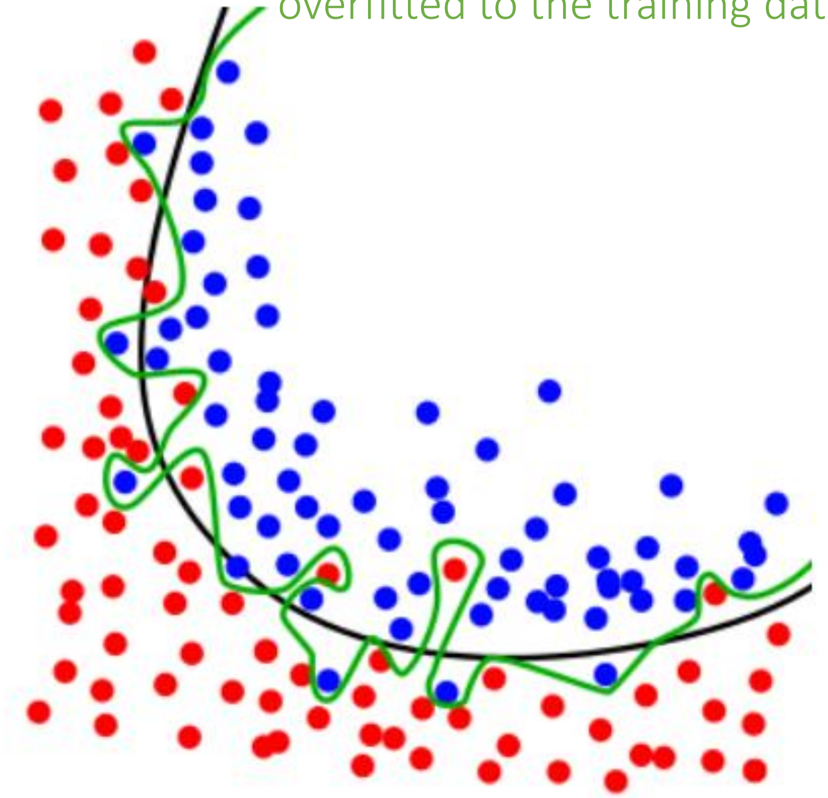


> Lots of neurons (connections, synapses) give us so complicated decision boundary.

Which do you think is desirable decision boundary?

overfitted to the training data

View from the sky



Training Data

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

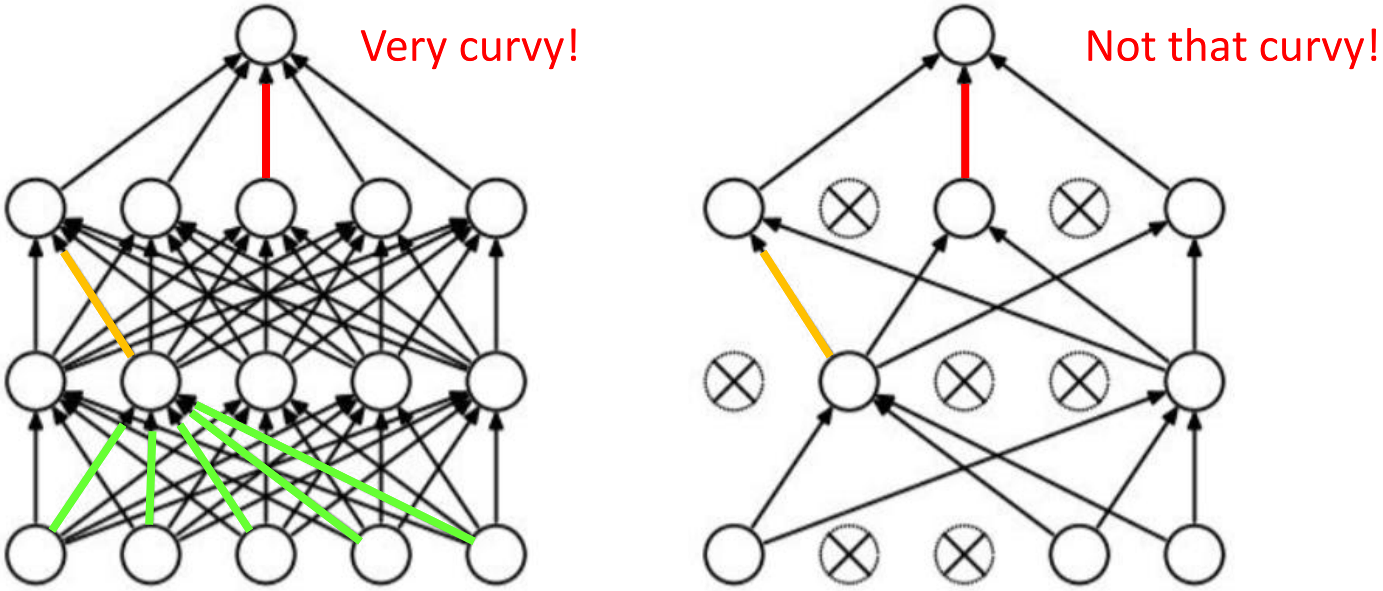
https://elitedatascience.com

Overfitting and drop-out

- The deeper the network is, the more the decision boundary is complex.
- Doing good at training data but errors for test 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]

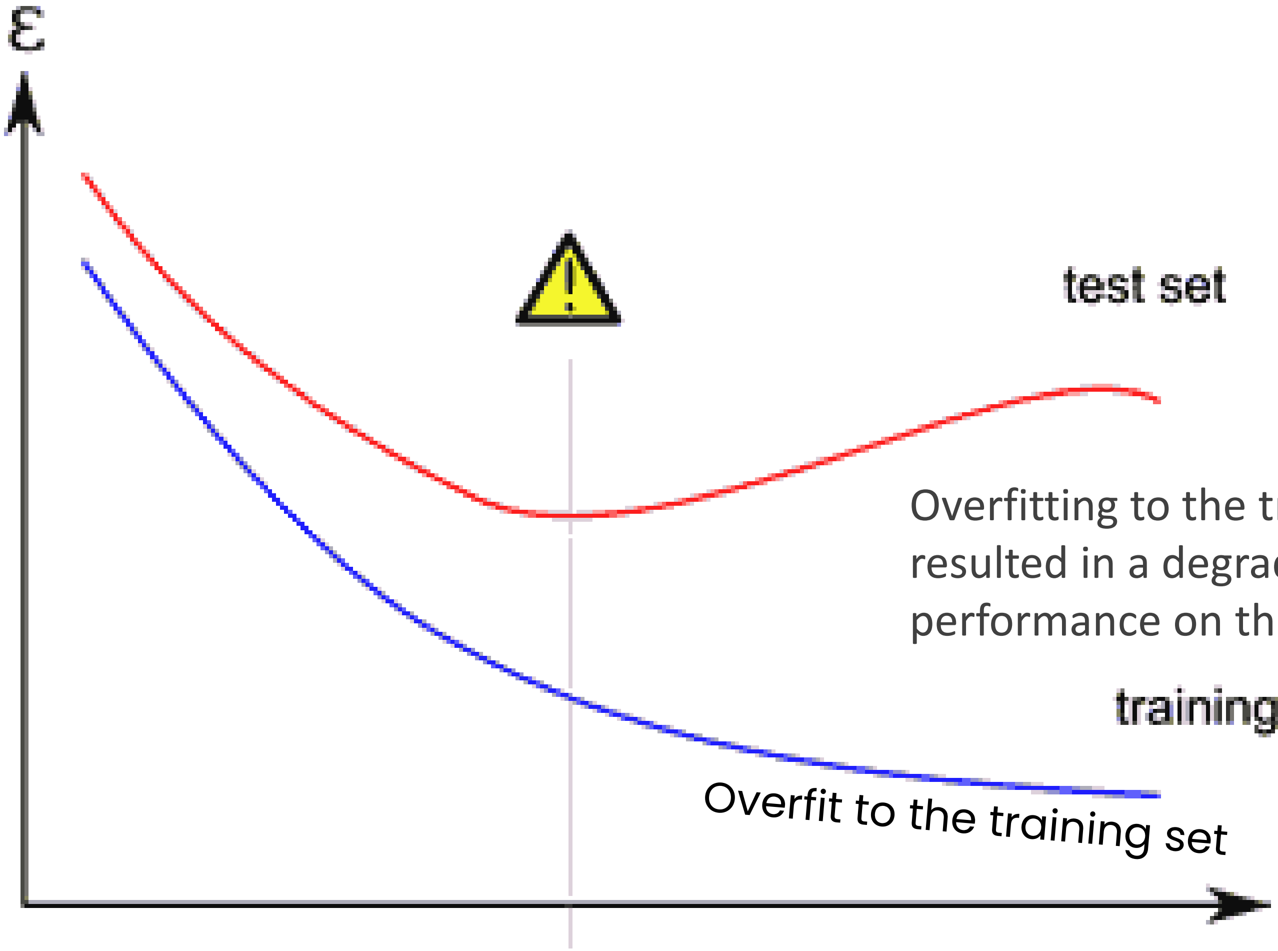
(Lab) 24.py

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

How to Prevent Overfitting

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

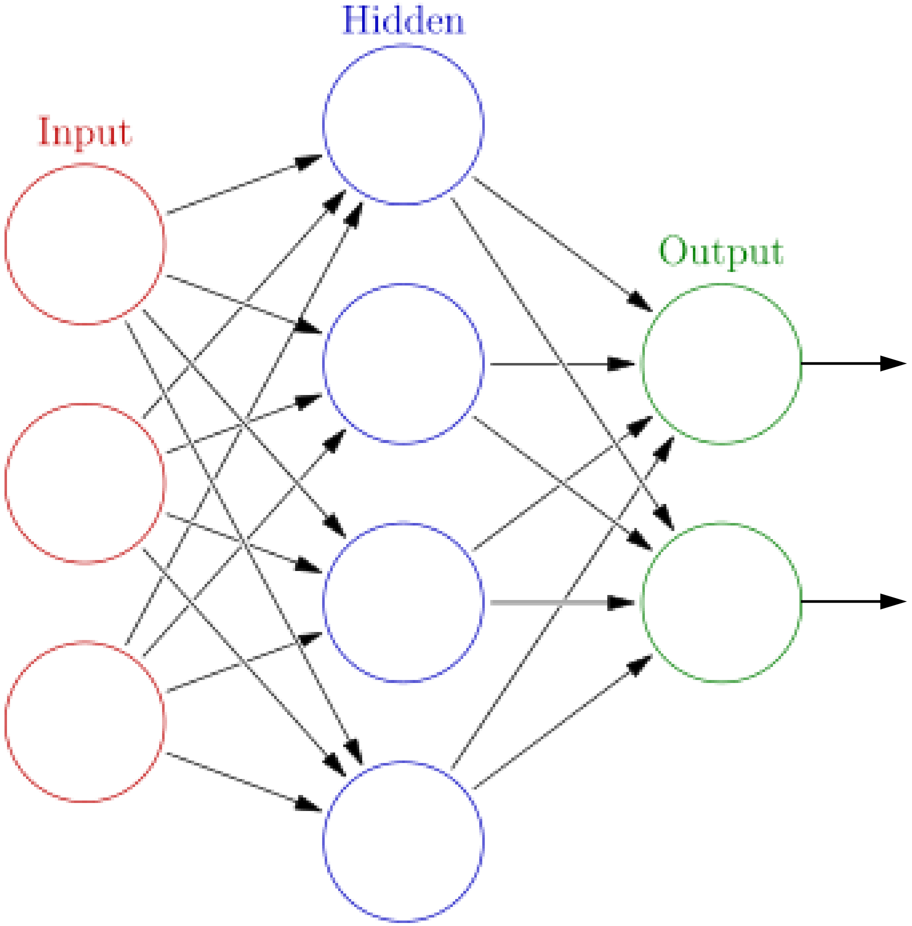
Early stopping



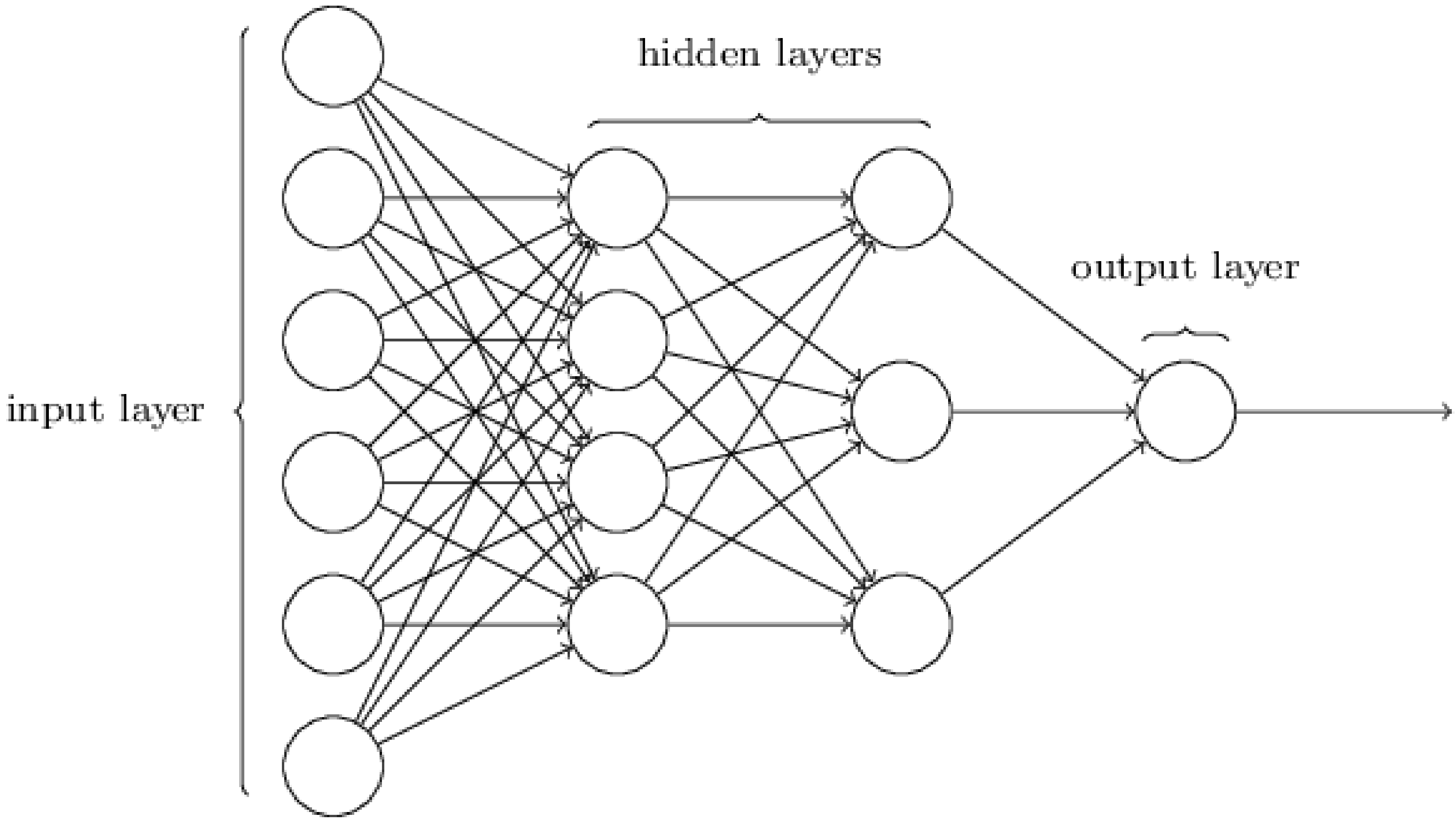
Overfitting to the training data resulted in a degraded performance on the test set.

Overfit to the training set

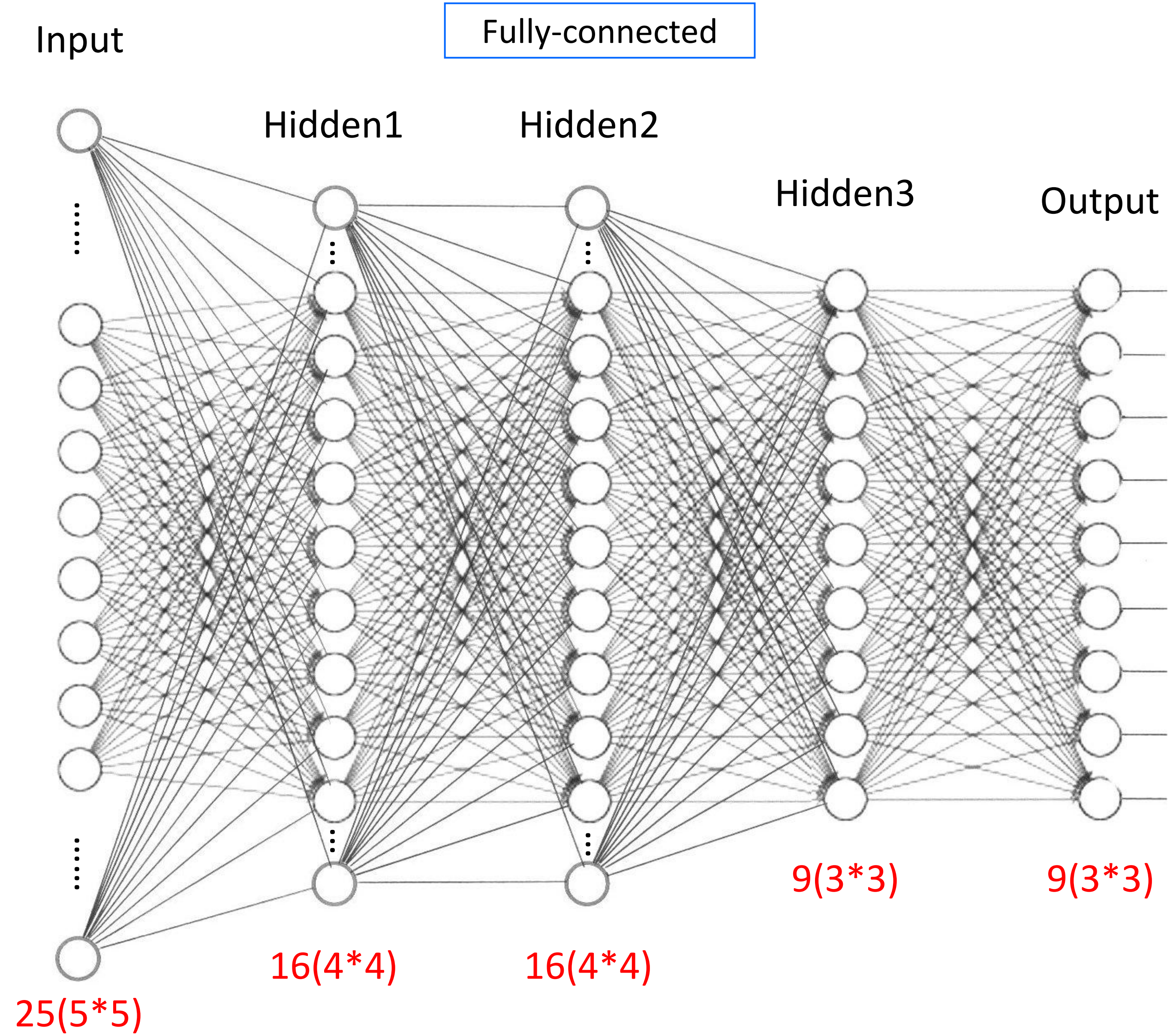
Fully-connected



Fully-connected



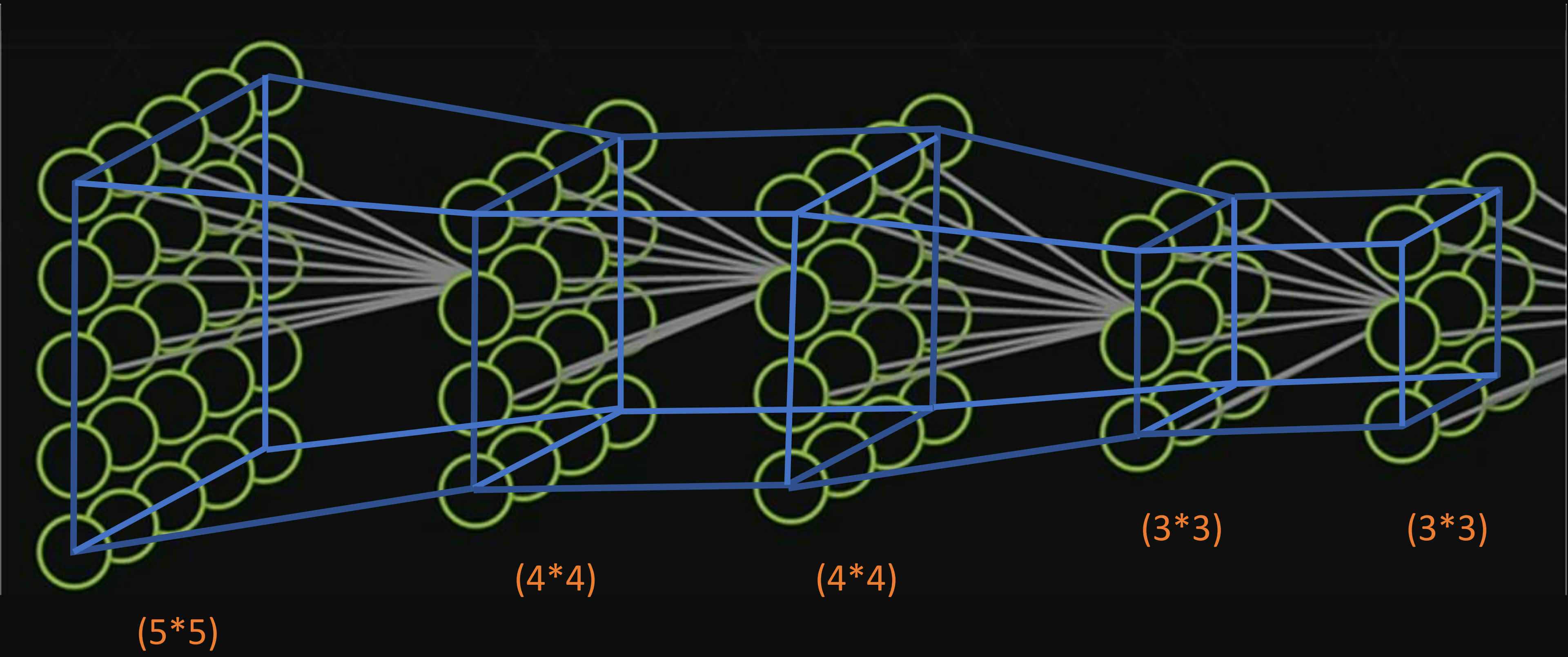
Fully-connected



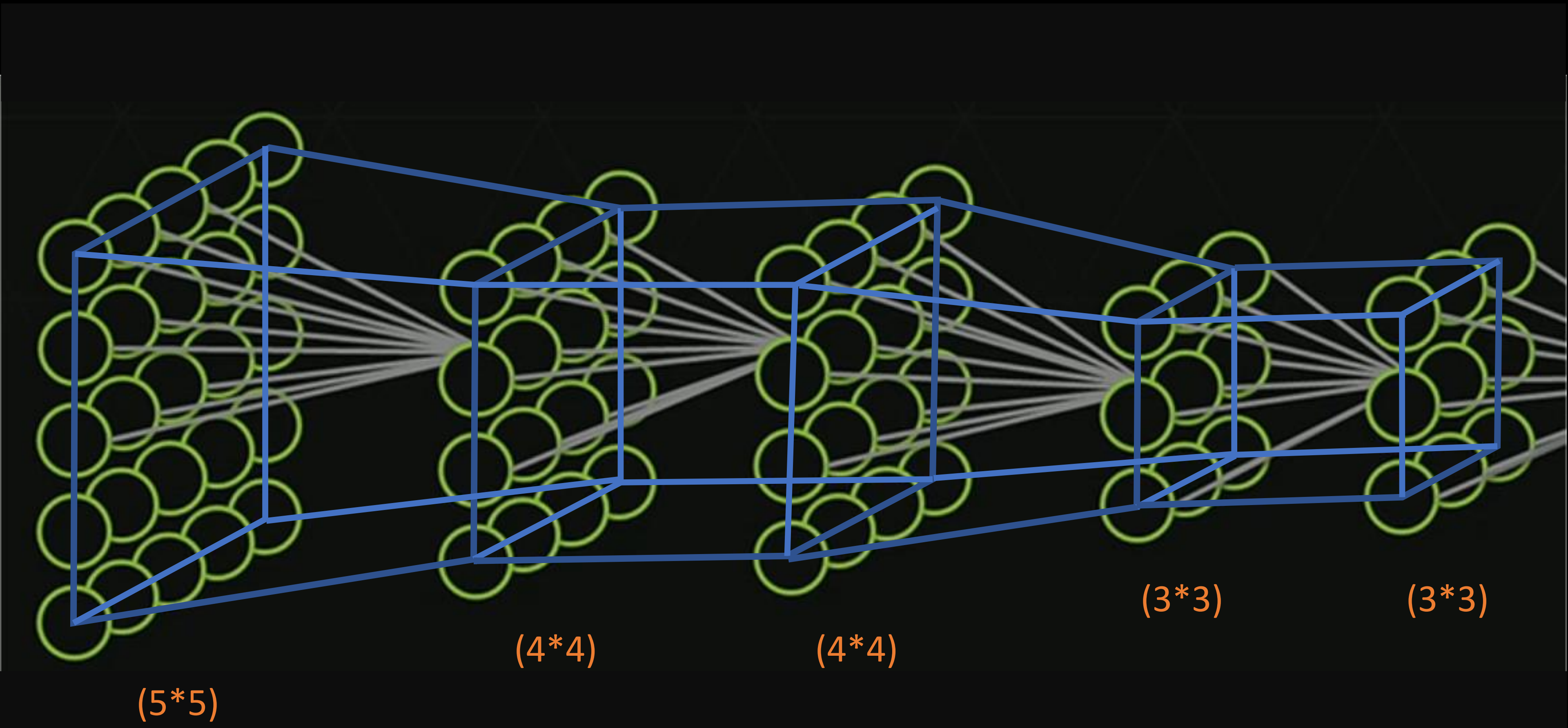
Fully connected, then how many connections(synapses, parameters) are there?

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

Fully-connected



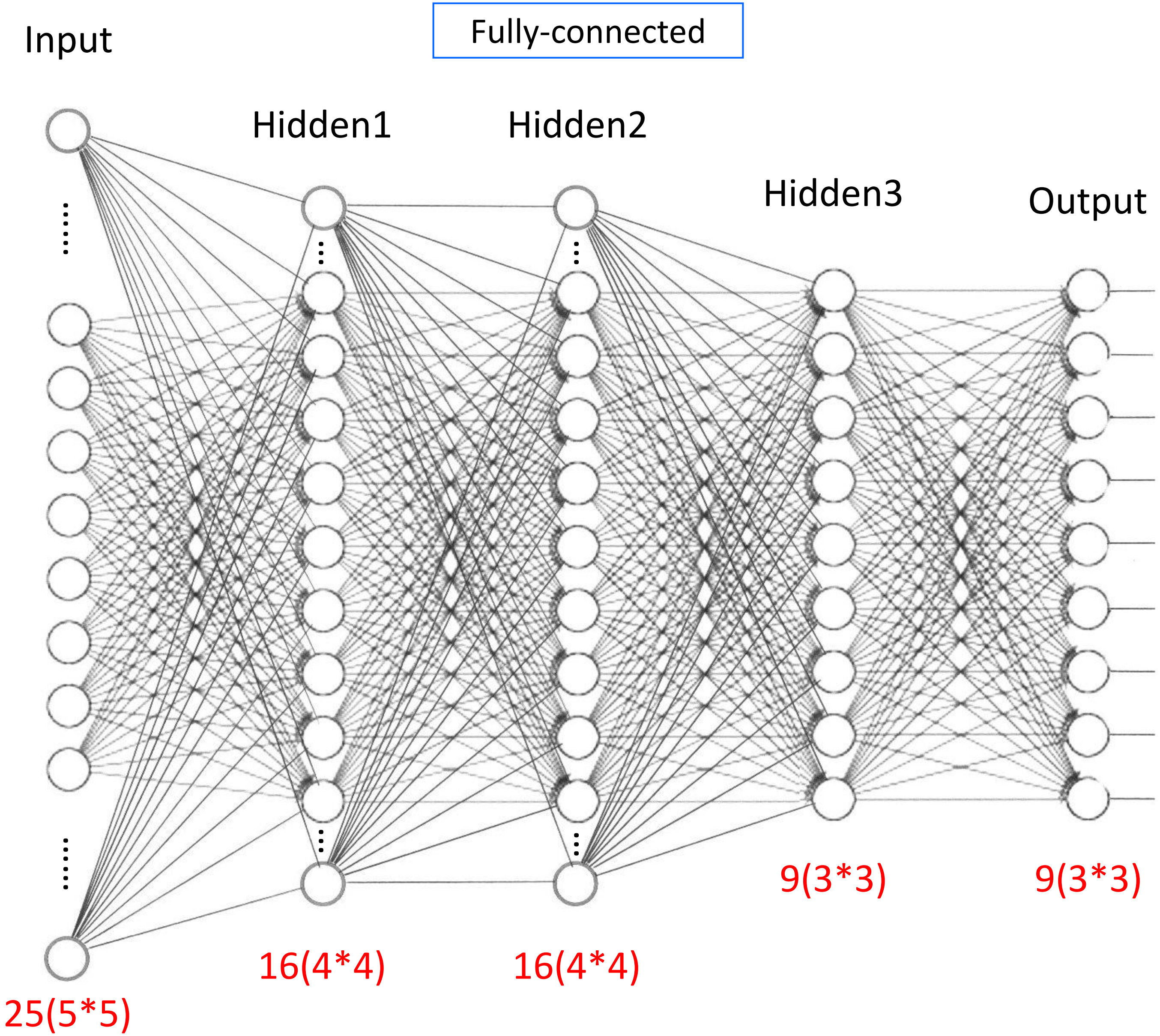
Fully-connected



Fully connected, so how many connections are there?

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

Fully-connected



Input

Hidden1

Hidden2

Hidden3

Output

25(5*5)

16(4*4)

16(4*4)

9(3*3)

9(3*3)

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

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

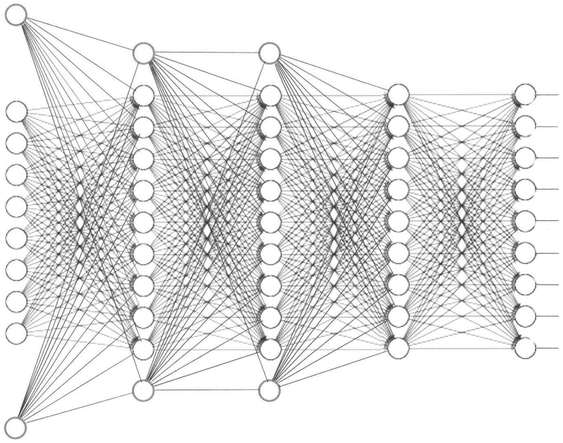
Geoffrey Hinton, Yann LeCun, Yoshua Bengio, Andrew Ng

Google Meta Mila amazon

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

Fully-connected Neural Networks

Any problem? Yes, so **CNN**.