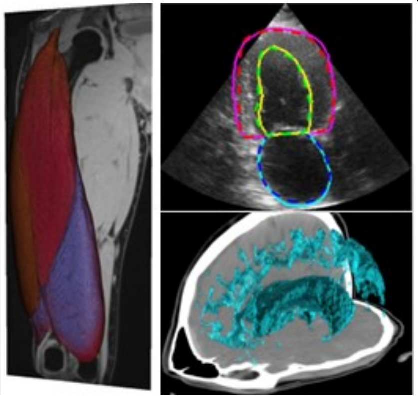
Deep learning for medical imaging school 2021

April 19—24 2021

Virtual edition

LABEX PRIMES
UNIVERSITÉ DE LYON



Basics of deep learning part 2

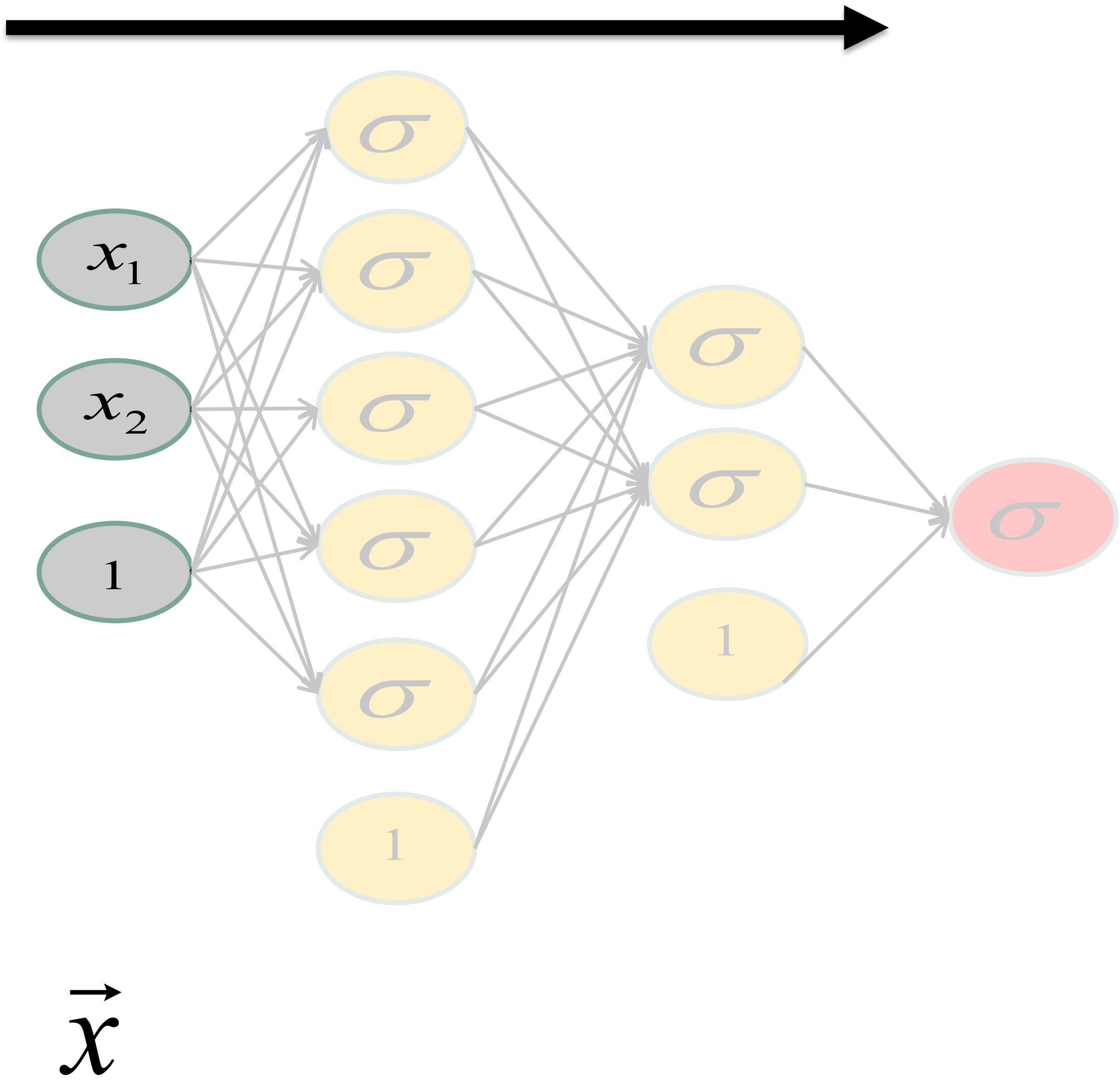
By

Christian Desrosiers

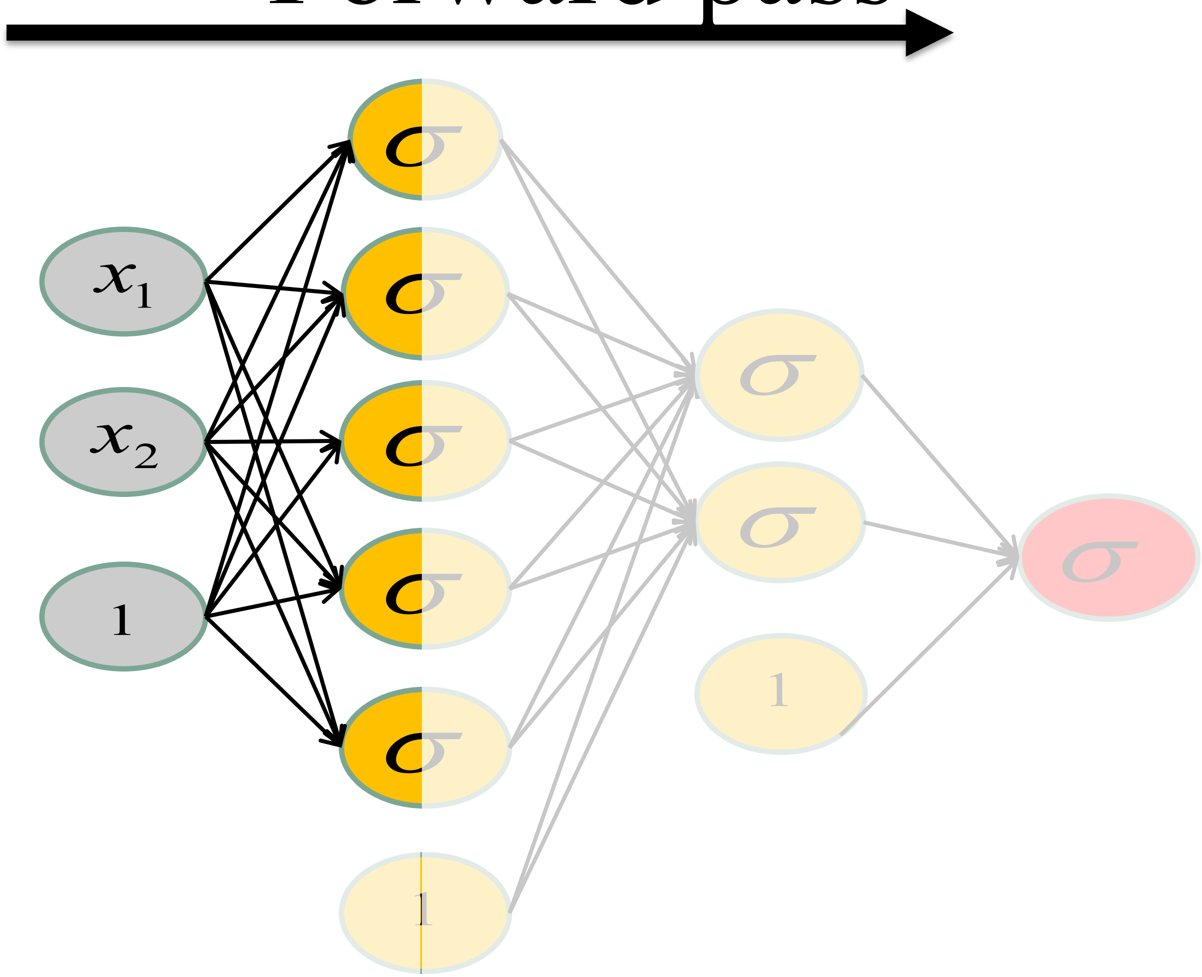
ÉTS
Le génie pour l'industrie

How to train the network?

Forward pass

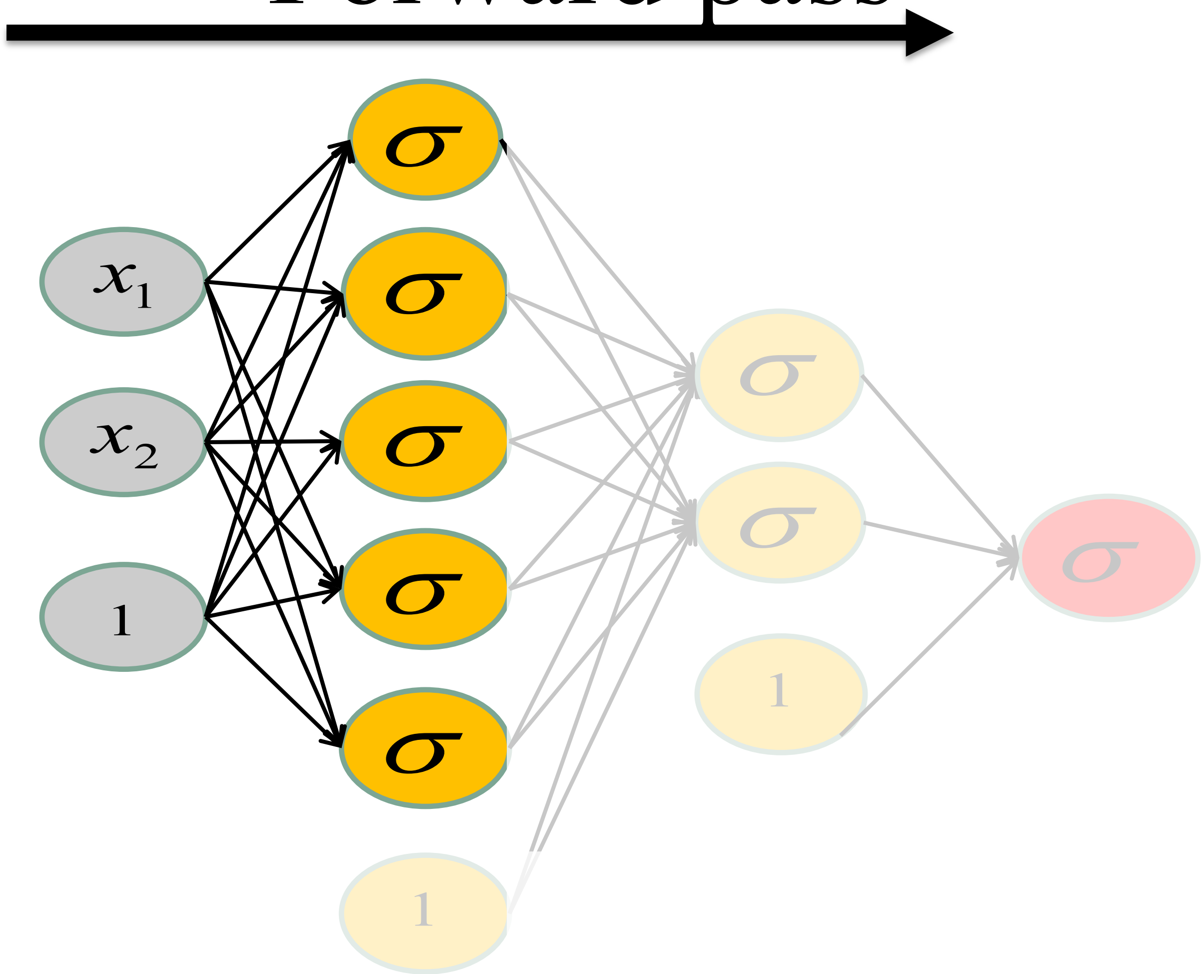


Forward pass



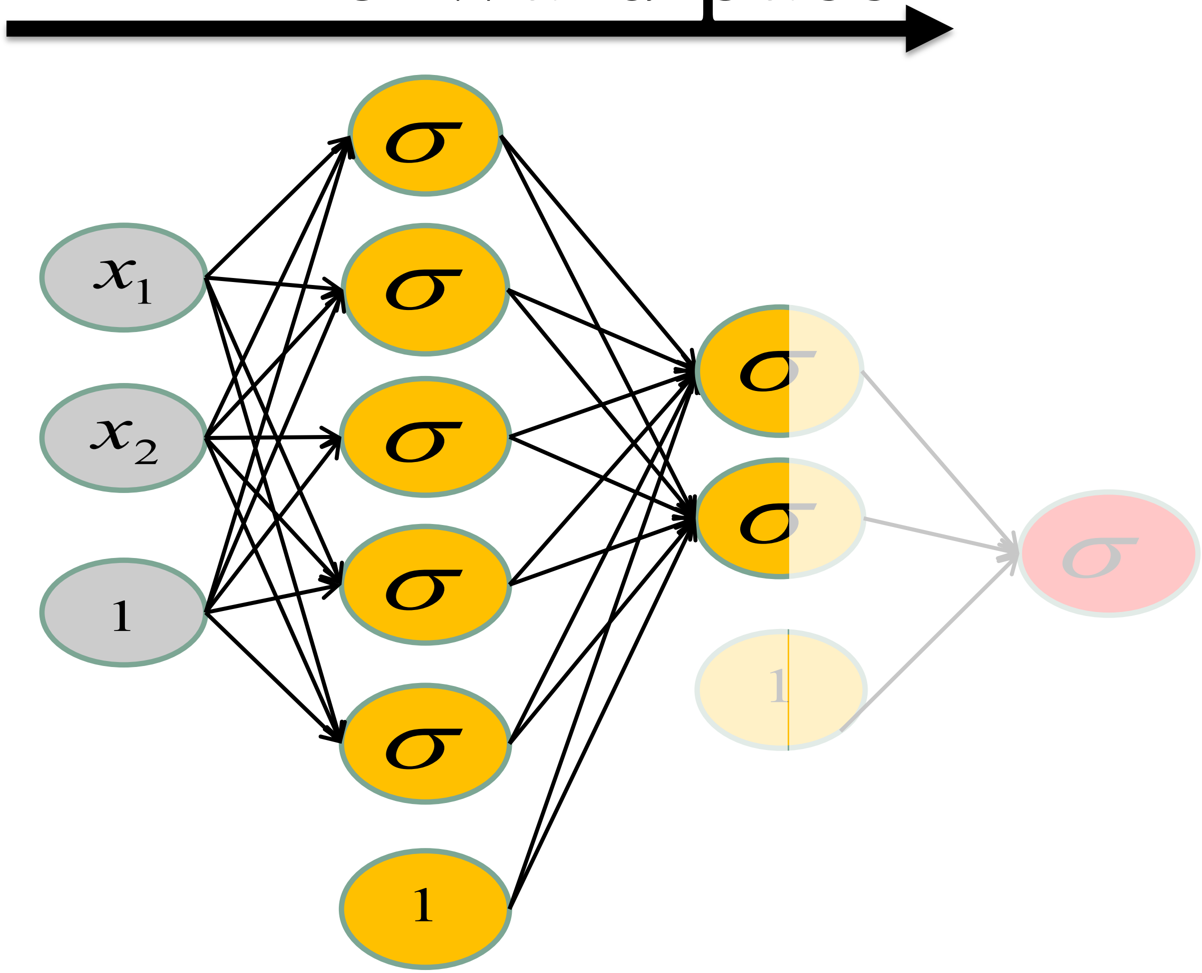
$$W^{[0]}\vec{x}$$

Forward pass



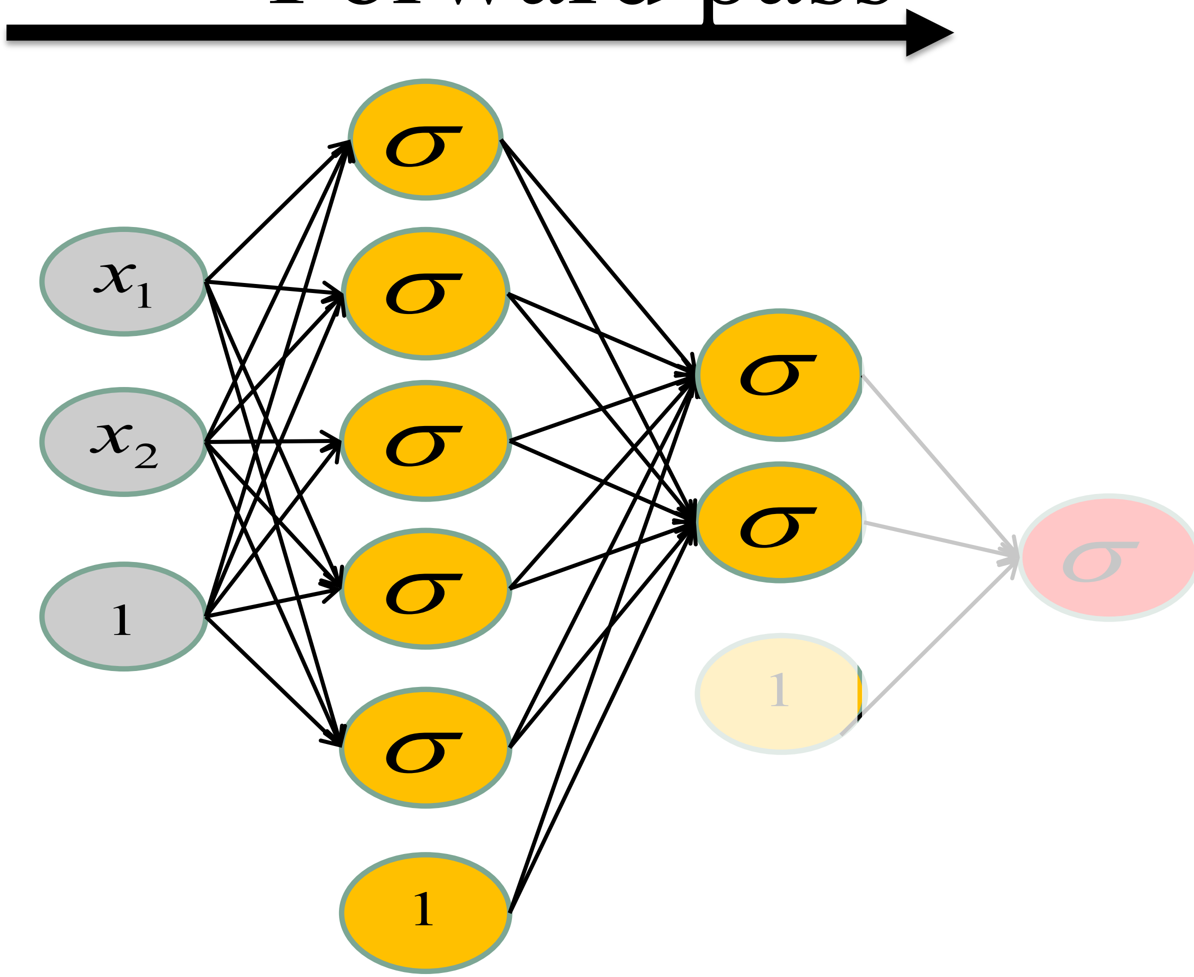
$$\sigma\left(W^{[0]}\vec{x}\right)$$

Forward pass



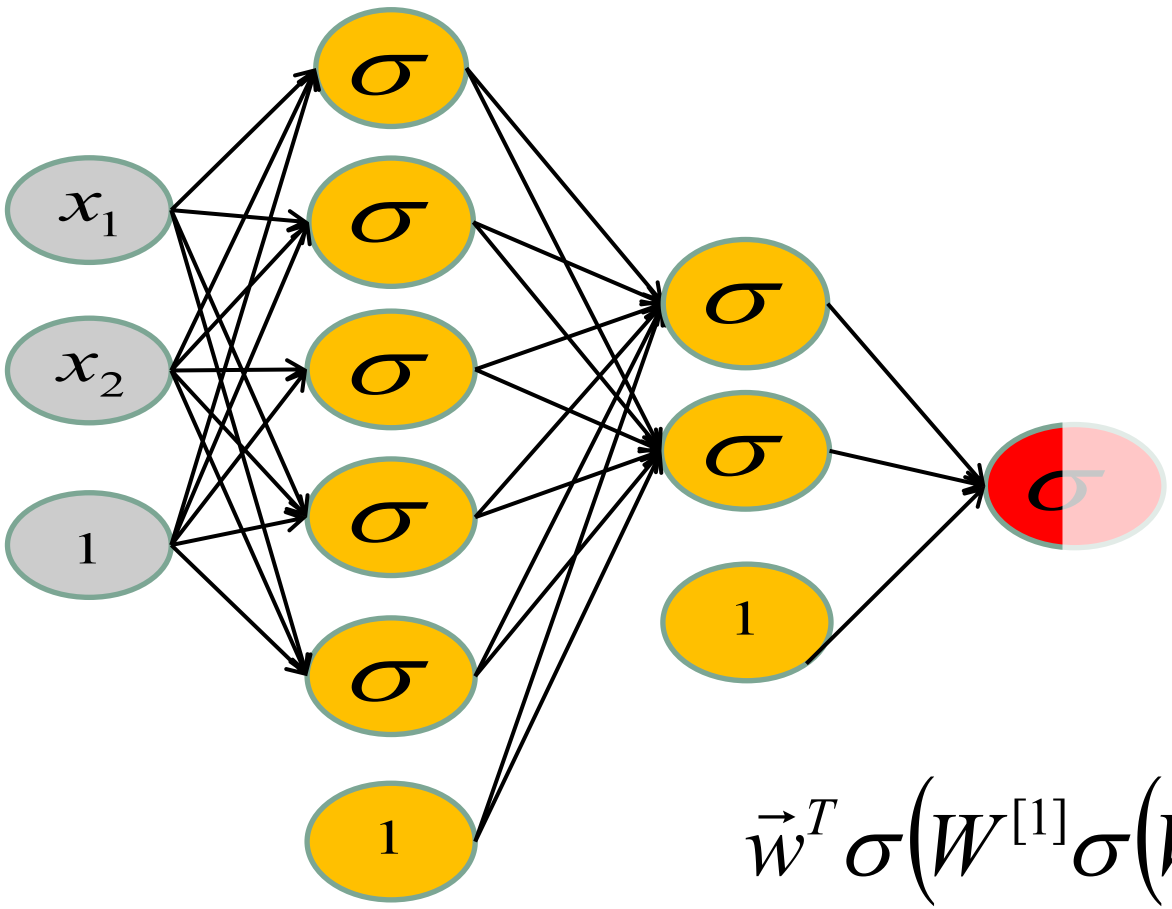
$$W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)$$

Forward pass



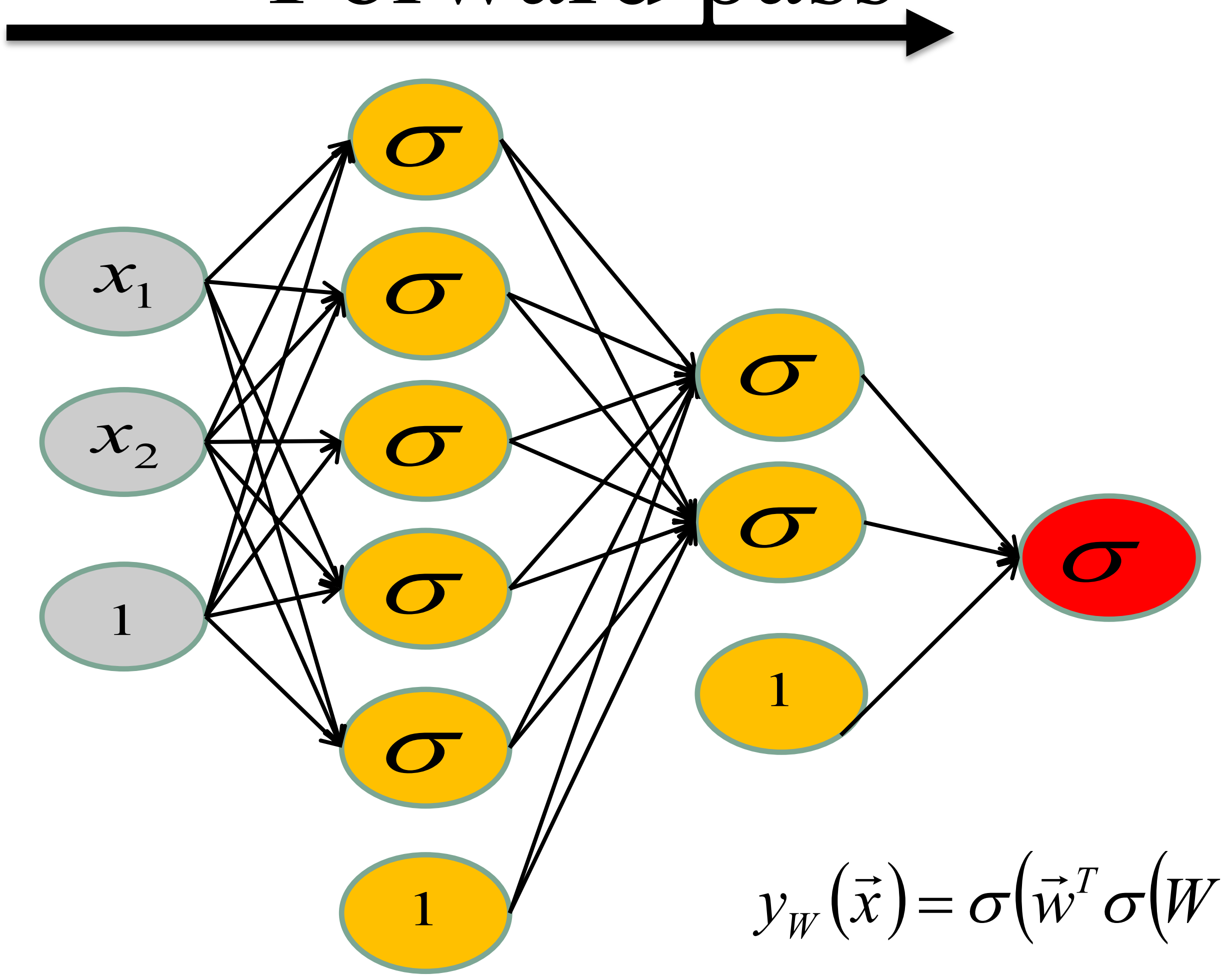
$$\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)$$

Forward pass



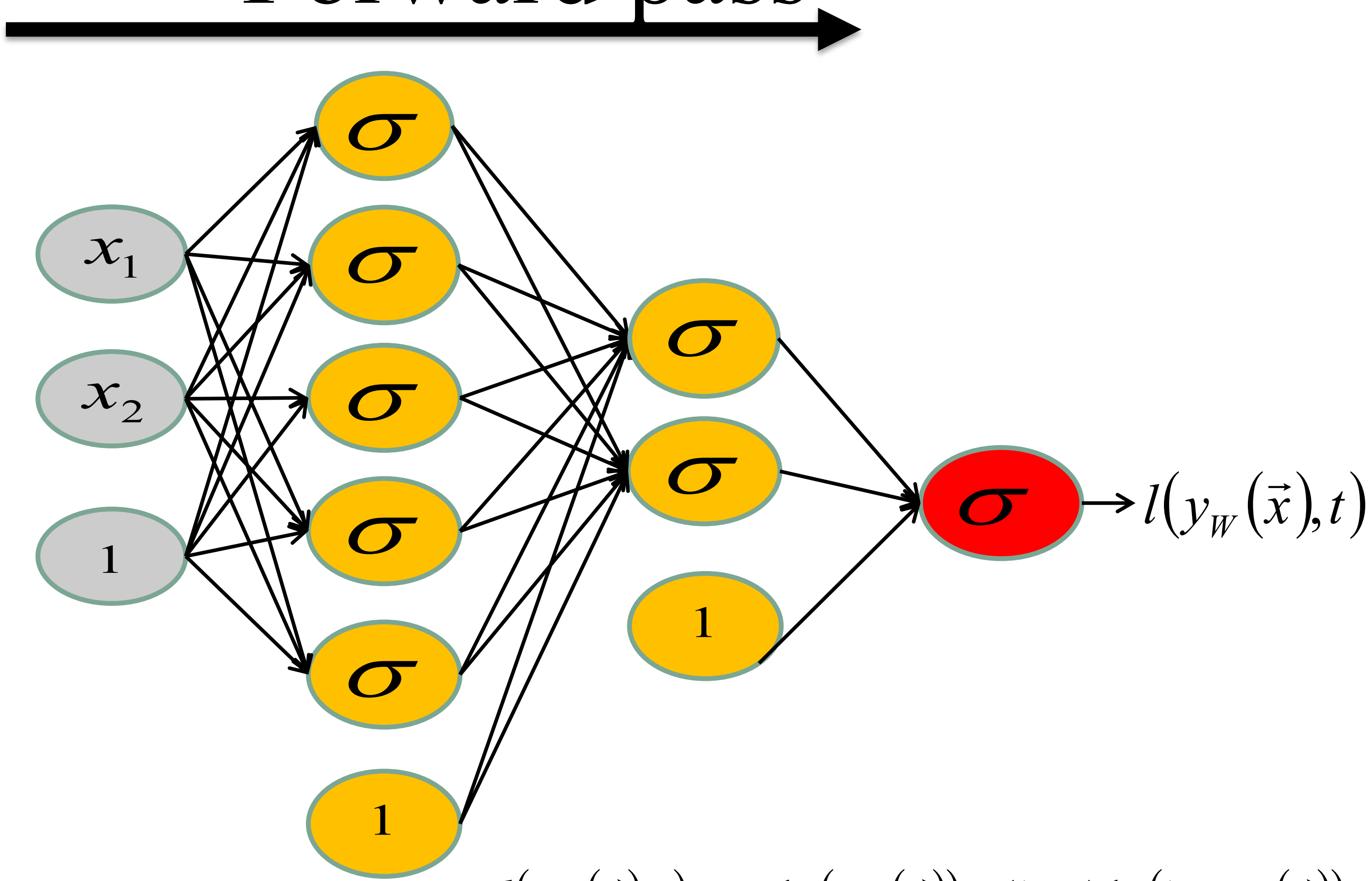
$$\vec{w}^T \sigma\left(W^{[1]} \sigma\left(W^{[0]} \vec{x}\right)\right)$$

Forward pass



$$y_W(\vec{x}) = \sigma\left(\vec{w}^T \sigma\left(W^{[1]} \sigma\left(W^{[0]} \vec{x}\right)\right)\right)$$

Forward pass



$$l\left(y_W\left(\vec{x}\right),t\right) = -t\ln\left(y_W\left(\vec{x}\right)\right) - (1-t)\ln\left(1-y_W\left(\vec{x}\right)\right)$$

How to optimize the network?

1- From

$$W = \arg\min_{W} = \sum_{n=1}^{N} l\left(y_W\left(\vec{x}_n\right), t_n\right) + \lambda R\left(W\right)$$

Choose a regularization function

$$R\left(W\right) = \left\|W\right\|_1 \text{ or } \left\|W\right\|_2$$

How to optimize the network?

2- Choose a loss $l\left(y_W\left(\vec{x}_n\right), t_n\right)$ for example

Hinge loss
Cross entropy

Do not forget to adjust the output layer with the loss you have choosen.

cross entropy => Softmax

How to optimize the network?

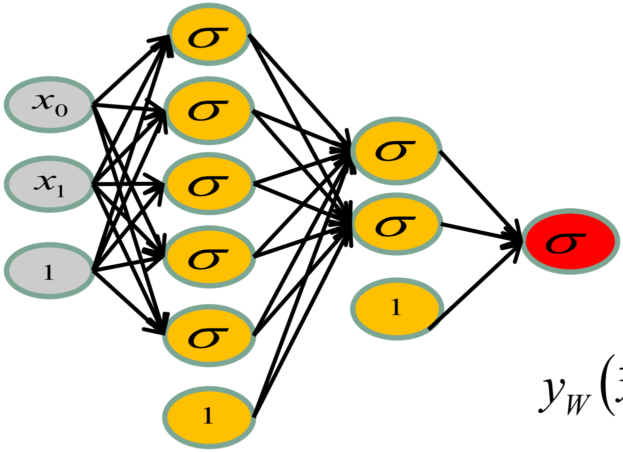
3- Compute the gradient of the loss with respect to each parameter

$$\frac{\partial\left(\sum_{n=1}^{N} l\left(y_W\left(\vec{x}_n\right), t_n\right) + \lambda R(W)\right)}{\partial W_{a,b}^{[c]}}$$

and launch a gradient descent algorithm to update the parameters.

$$W_{a,b}^{[c]} := W_{a,b}^{[c]} - \eta \frac{\partial\left(\sum_{n=1}^{N} l\left(y_W\left(\vec{x}_n\right), t_n\right) + \lambda R(W)\right)}{\partial W_{a,b}^{[c]}}$$

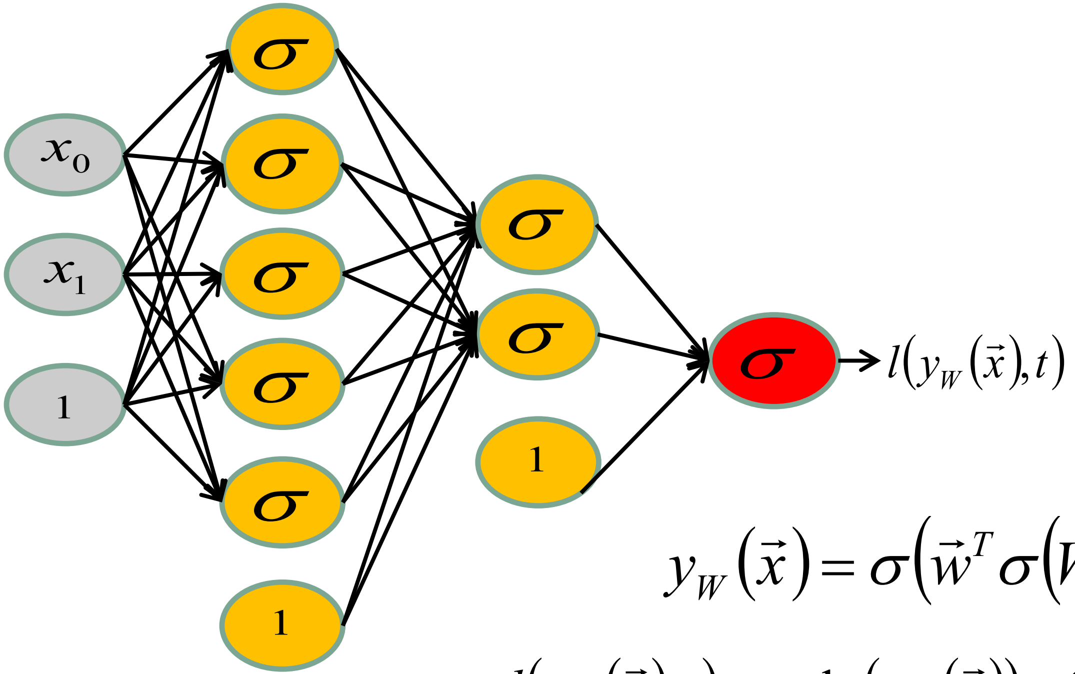
How to optimize the network?



$$y_W(\vec{x}) = \sigma\left(\vec{w}^{[2]}\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)\right)$$

$$\frac{\partial\left(\sum_{n=1}^{N} l\left(y_W(\vec{x}_n), t_n\right) + \lambda R(W)\right)}{\partial W_{a,b}^{[c]}}$$

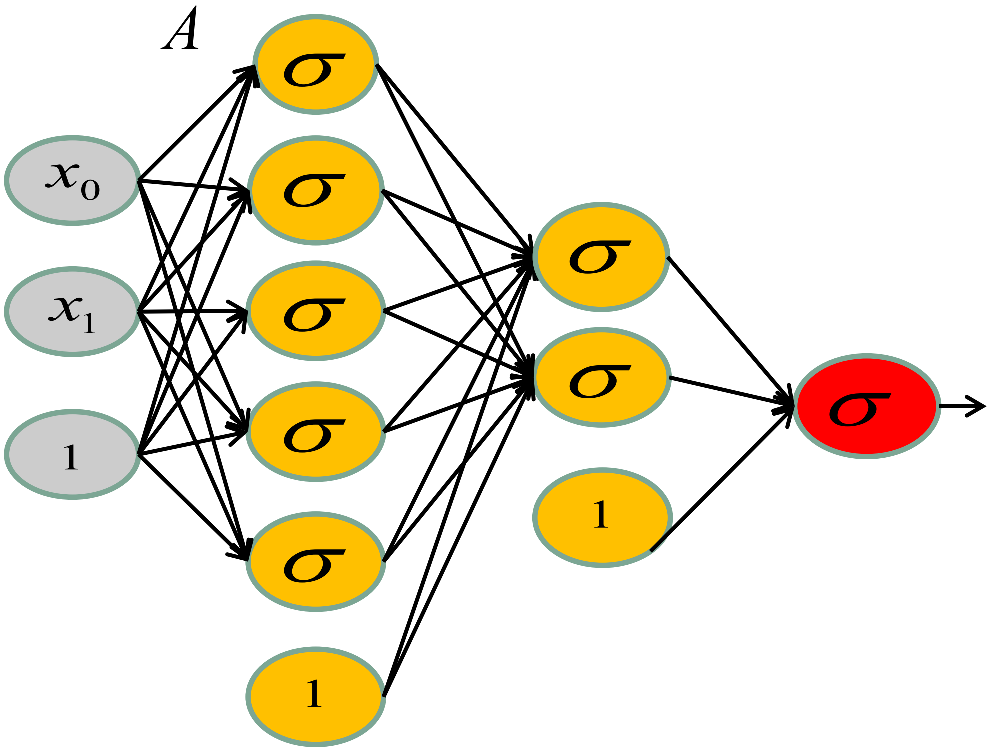
➡ Backpropagation



$$y_W\left(\vec{x}\right) = \sigma\left(\vec{w}^T \sigma\left(W^{[1]} \sigma\left(W^{[0]} \vec{x}\right)\right)\right)$$

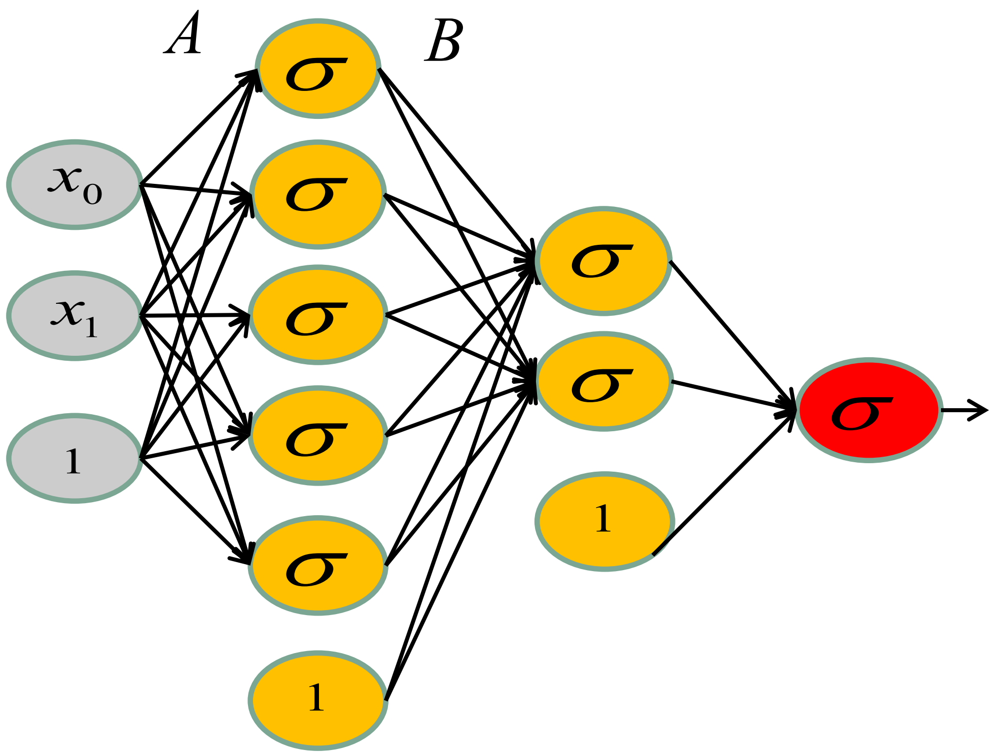
$$l\left(y_W\left(\vec{x}\right), t\right) = -t \ln\left(y_W\left(\vec{x}\right)\right) - (1-t) \ln\left(1 - y_W\left(\vec{x}\right)\right)$$

$$A = W^{[0]}\vec{x}$$

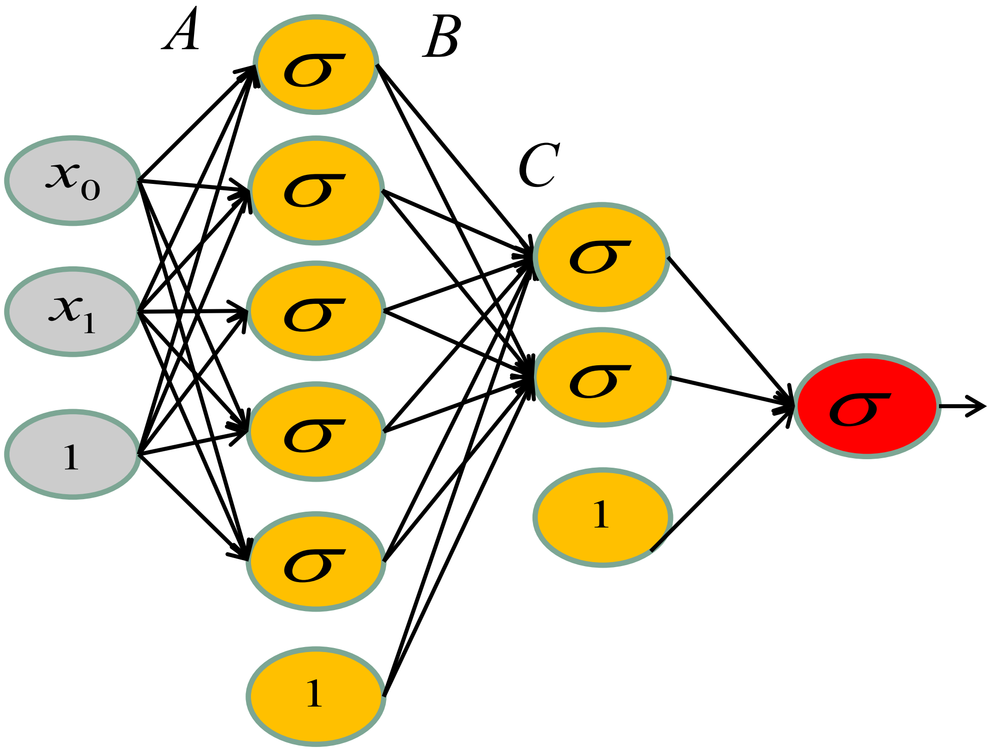


$$A = W^{[0]}\vec{x}$$

$$B = \sigma(A)$$



$$A = W^{[0]}\vec{x}$$
$$B = \sigma(A)$$
$$C = W^{[1]}B$$

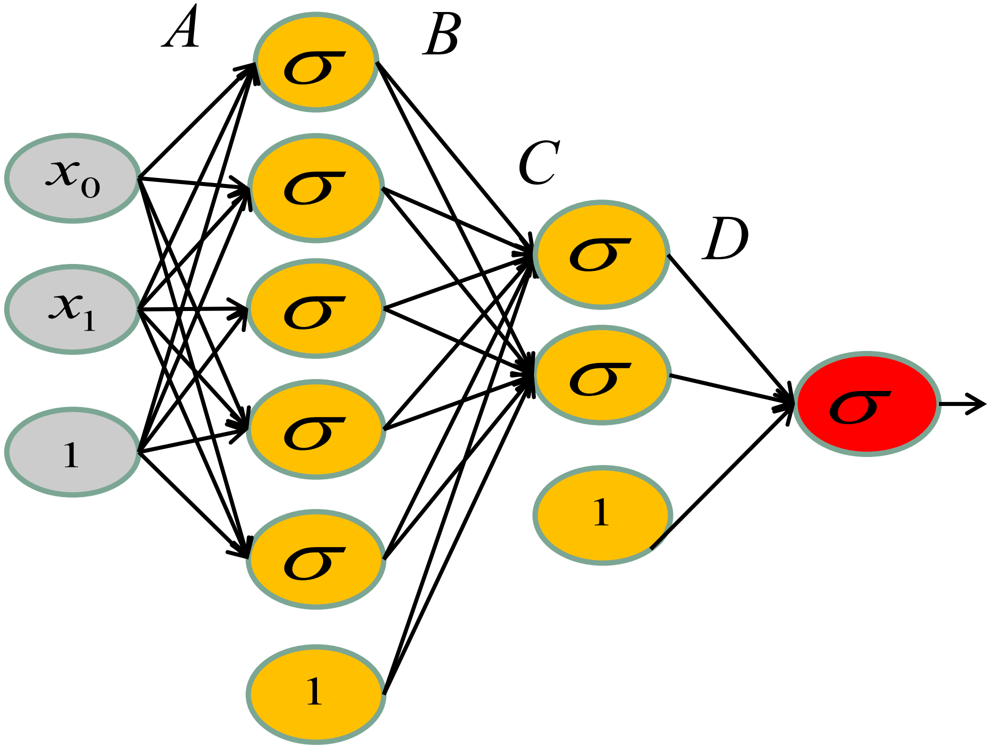


$$A = W^{[0]}\vec{x}$$

$$B = \sigma(A)$$

$$C = W^{[1]}B$$

$$D = \sigma(C)$$



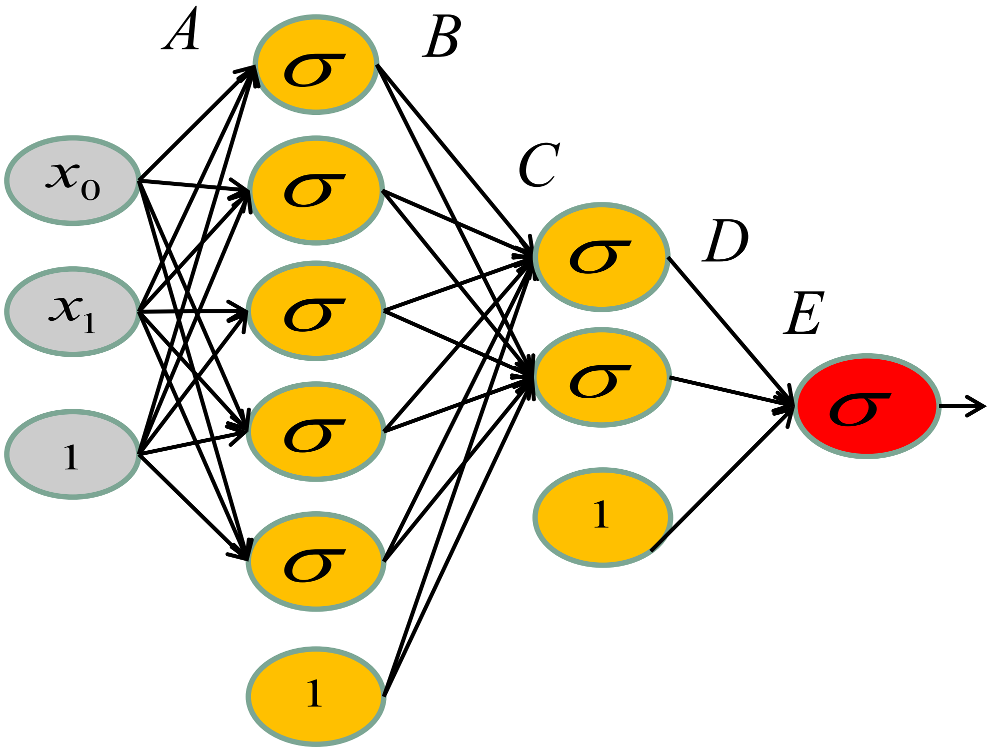
$$A = W^{[0]}\vec{x}$$

$$B = \sigma(A)$$

$$C = W^{[1]}B$$

$$D = \sigma(C)$$

$$E = \vec{w}^T D$$



$$A = W^{[0]}\vec{x}$$

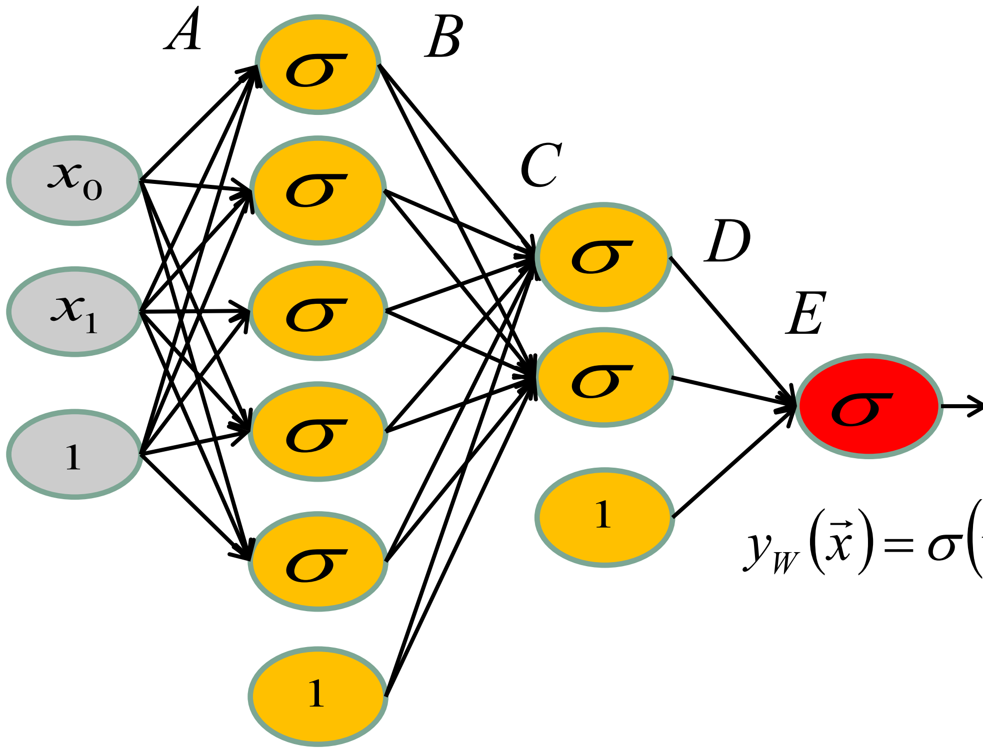
$$B = \sigma(A)$$

$$C = W^{[1]}B$$

$$D = \sigma(C)$$

$$E = \vec{w}^T D$$

$$y_W(\vec{x}) = \sigma(E)$$



$$y_W(\vec{x}) = \sigma\left(\vec{w}^T\sigma\left(W^{[1]}\sigma\left(W^{[0]}\vec{x}\right)\right)\right)$$

$$A = W^{[0]}\vec{x}$$

$$B = \sigma(A)$$

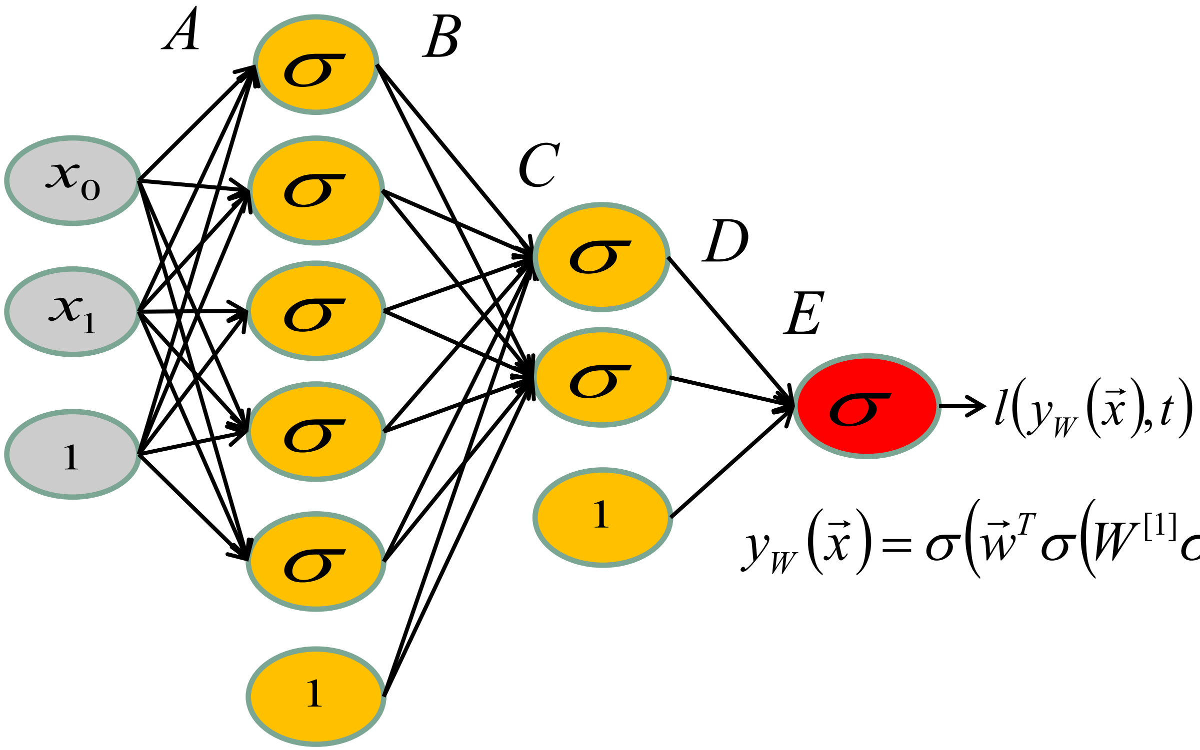
$$C = W^{[1]}B$$

$$D = \sigma(C)$$

$$E = \vec{w}^T D$$

$$y_W(\vec{x}) = \sigma(E)$$

$$l(y_W(\vec{x}), t)$$



$$y_W(\vec{x}) = \sigma\left(\vec{w}^T \sigma\left(W^{[1]} \sigma\left(W^{[0]}\vec{x}\right)\right)\right)$$

$$\frac{\partial\left(\sum_{n=1}^{N} l\left(y_W\left(\vec{x}_n\right), t_n\right)\right)}{\partial W^{[l]}} \quad ?$$

Chain rule

Chain rule recap

$$\left.\begin{array}{l} f(u) = u^2 \\ u(v) = 2v \\ v(x) = 1/x \end{array}\right\} \quad \frac{\partial f}{\partial x} = ? \quad \left\} \begin{aligned} \frac{\partial f}{\partial x} &= \frac{\partial f}{\partial u} \times \frac{\partial u}{\partial v} \times \frac{\partial v}{\partial x} \\ &= 2u \times 2 \times \left(-\frac{1}{x^2} \right) \end{aligned}\right.$$

$$A = W^{[0]}\vec{x}$$

$$B = \sigma(A)$$

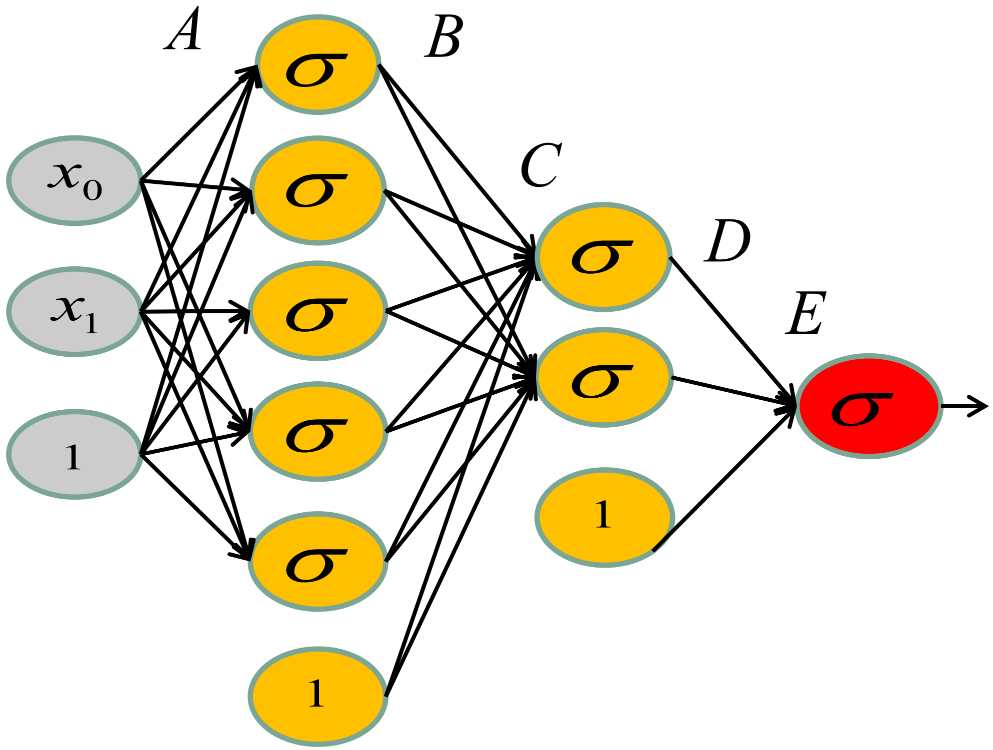
$$C = W^{[1]}B$$

$$D = \sigma(C)$$

$$E = \vec{w}^T D$$

$$y_W(\vec{x}) = \sigma(E)$$

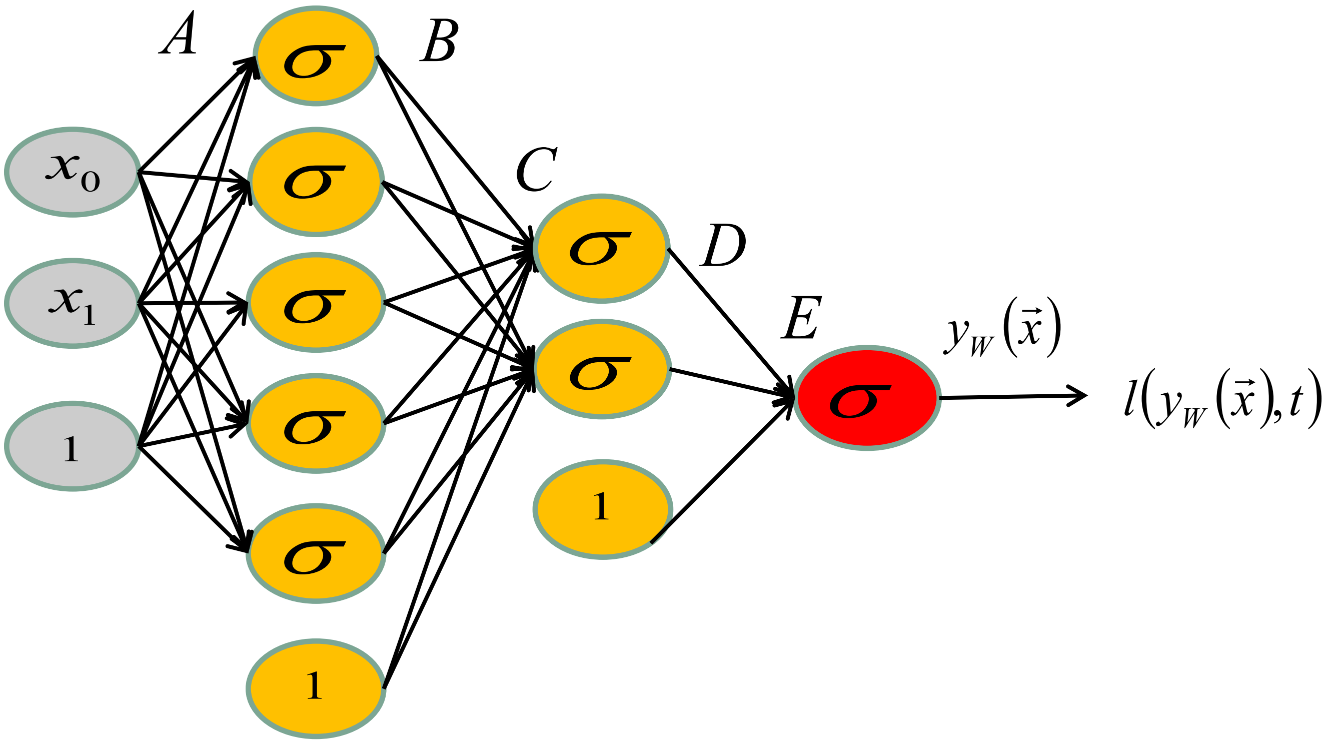
$$l(y_W(\vec{x}), t)$$



$$\frac{\partial(l(y_W(\vec{x}),t))}{\partial W^{[0]}} = \frac{\partial(l(y_W(\vec{x}),t))}{\partial y_W(\vec{x})} \frac{\partial(y_W(\vec{x}))}{\partial E} \frac{\partial(E)}{\partial D} \frac{\partial(D)}{\partial C} \frac{\partial(C)}{\partial B} \frac{\partial(B)}{\partial A} \frac{\partial(A)}{\partial W^{[0]}}$$

Back propagation

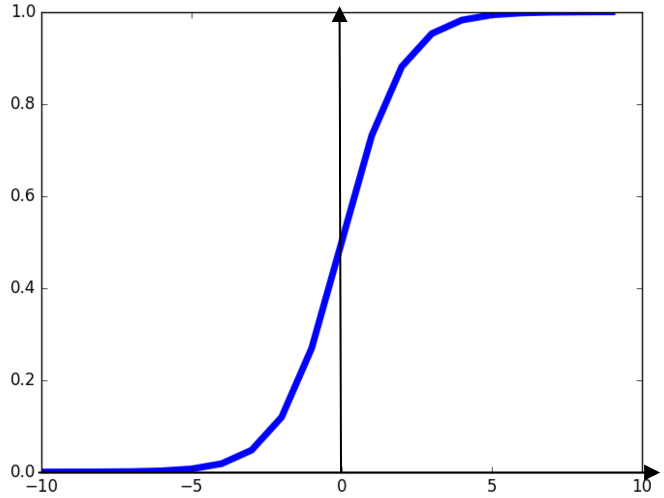
$$\frac{\partial\left(l\left(y_W(\vec{x}),t\right)\right)}{\partial W^{[0]}}=\frac{\partial\left(l\left(y_W(\vec{x}),t\right)\right)}{\partial y_W(\vec{x})}\frac{\partial\left(y_W(\vec{x})\right)}{\partial E}\frac{\partial(E)}{\partial D}\frac{\partial(D)}{\partial C}\frac{\partial(C)}{\partial B}\frac{\partial(B)}{\partial A}\frac{\partial(B)}{\partial W^{[0]}}$$



Activation functions

Activation functions

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

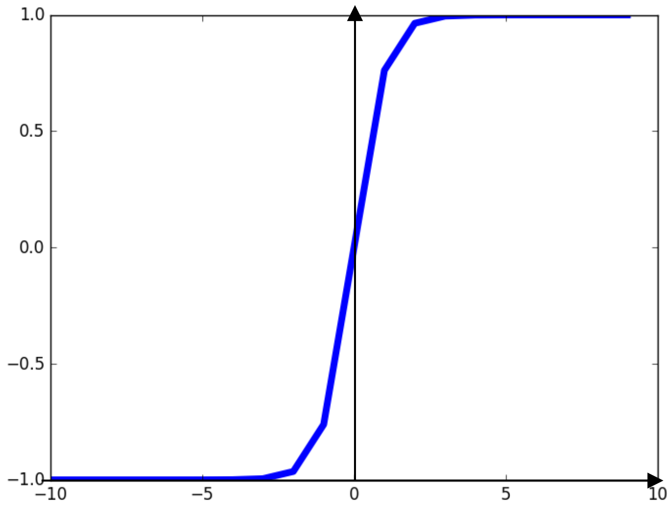


Sigmoid

3 Problems :

- Gradient saturates when input is large
- Not zero centered
- *exp*() is an expensive operation

Activation functions



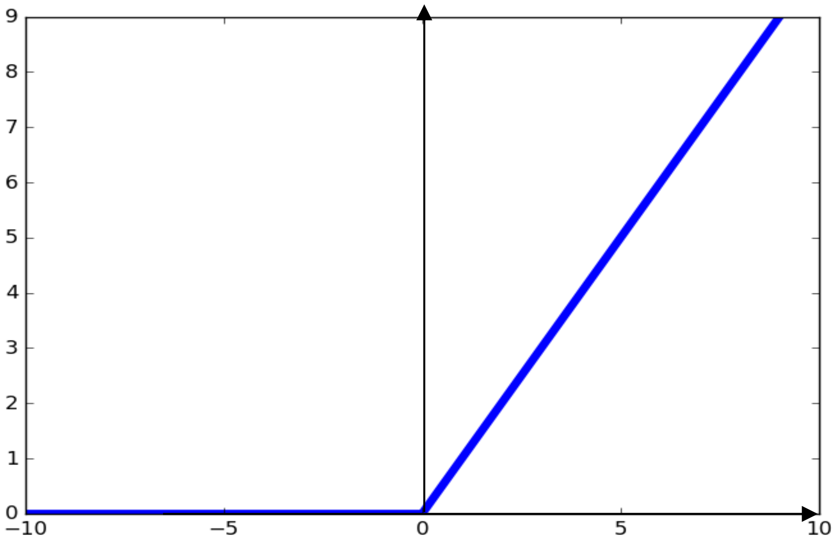
Tanh(x)

- **Output is zero-centered ☺**
- **Small gradient** when input is large ☹

[LeCun et al., 1991]

Activation functions

$$\mathrm{ReLU}(x) = \max(0, x)$$



- **Large gradient** for $x > 0$ ☺
- Super **fast** ☺

- Output **non centered at zero** ☹
- **No gradient** when x < 0? ☹

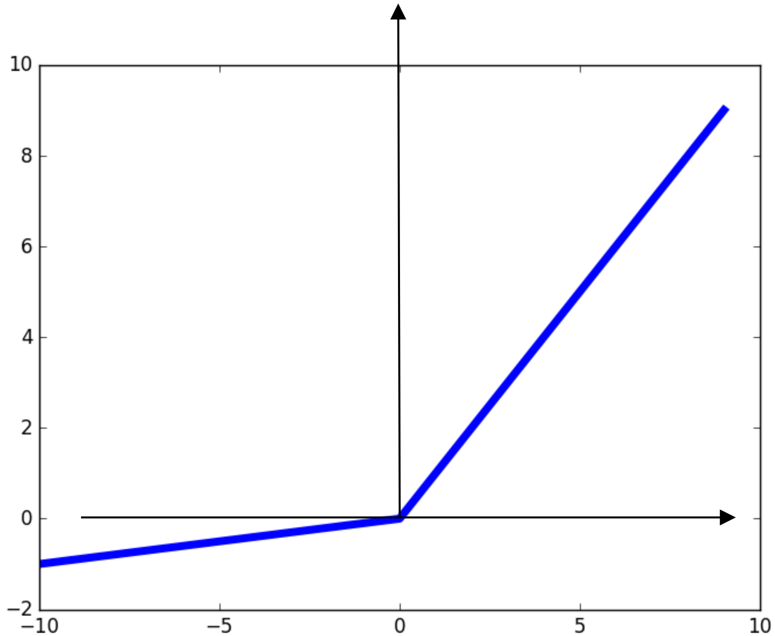
ReLU(x)

(Rectified Linear Unit)

[Krizhevsky et al., 2012]

Activation functions

$$\mathrm{LReLU}\left(x\right) = \max\left(0.01x, x\right)$$



Leaky ReLU(x)

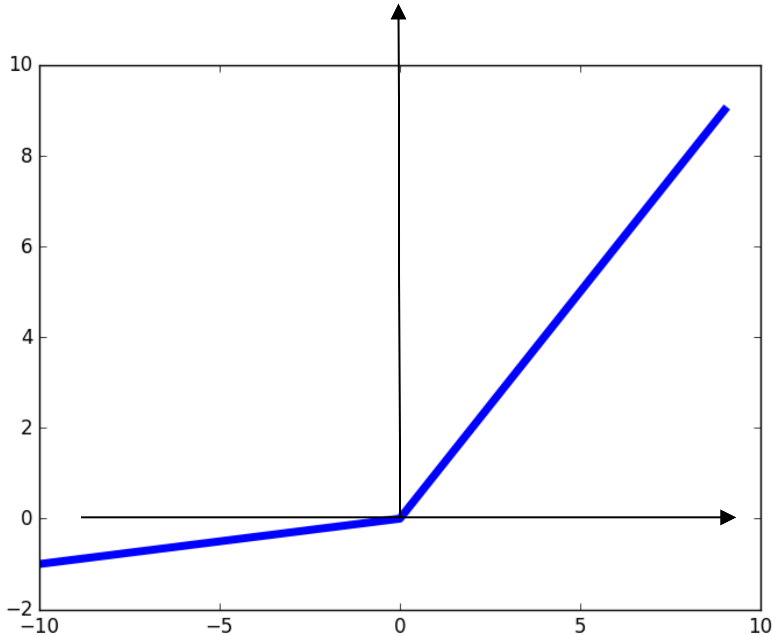
- no **gradient saturation** ☺
- Super **fast** ☺

- 0.01 is a **hyperparameter** 😐

[Mass et al., 2013]
[He et al., 2015]

Activation functions

$$\text{PReLU}(x) = \max(\alpha x, x)$$



Parametric ReLU(x)

- no **gradient saturation** ☺
- Super **fast** ☺

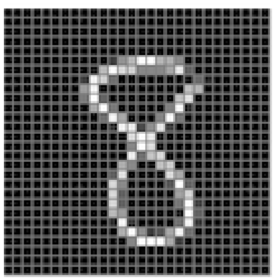
- α **learned** with backprop ☺

[Mass et al., 2013]
[He et al., 2015]

In practice

- By default, people use **ReLU**.
- Try **Leaky ReLU / PReLU**
- Try **tanh** but might be sub-optimal
- **Do not use sigmoid** except at the output of a 2 class net.

How to classify an image?

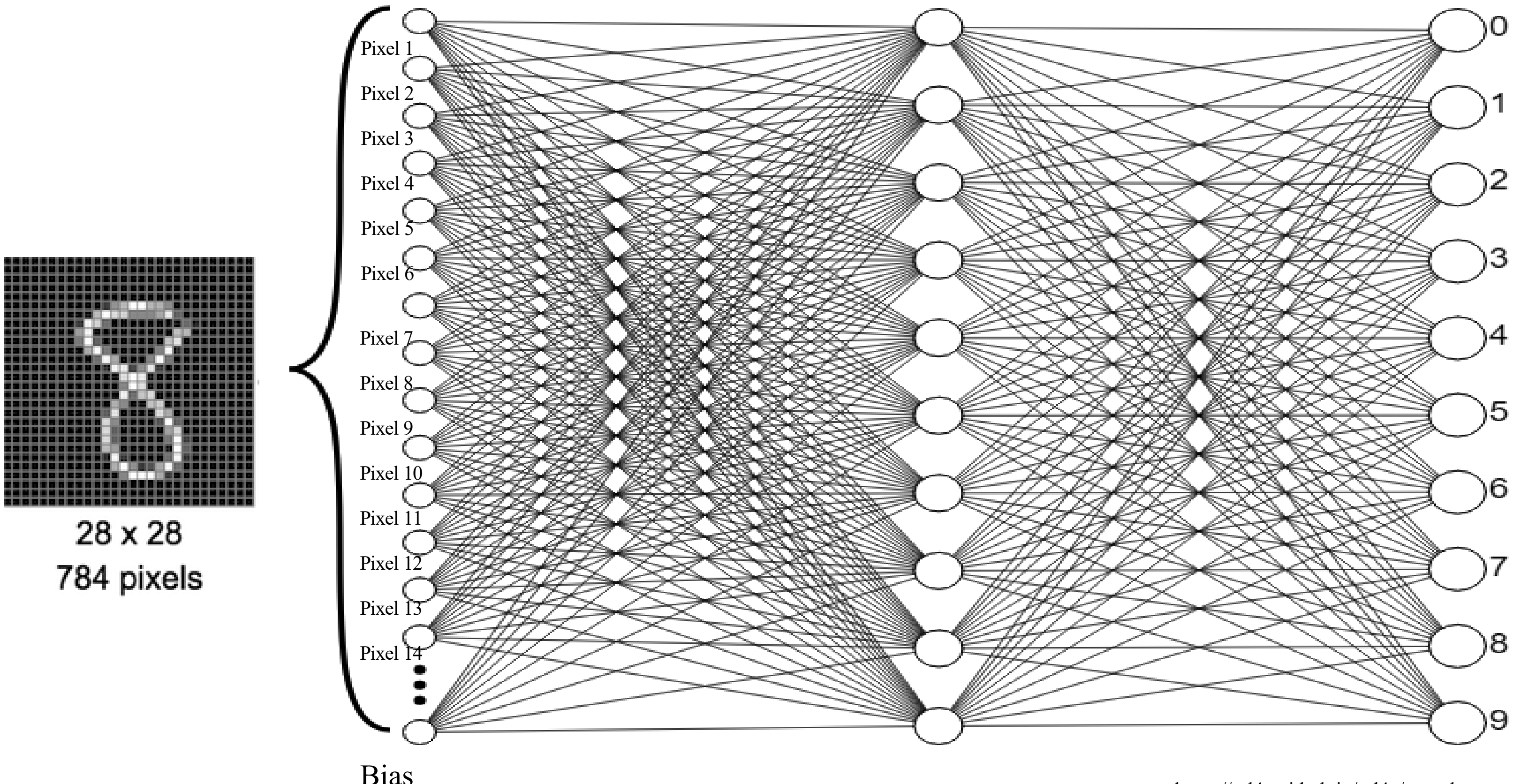


28 x 28
784 pixels

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

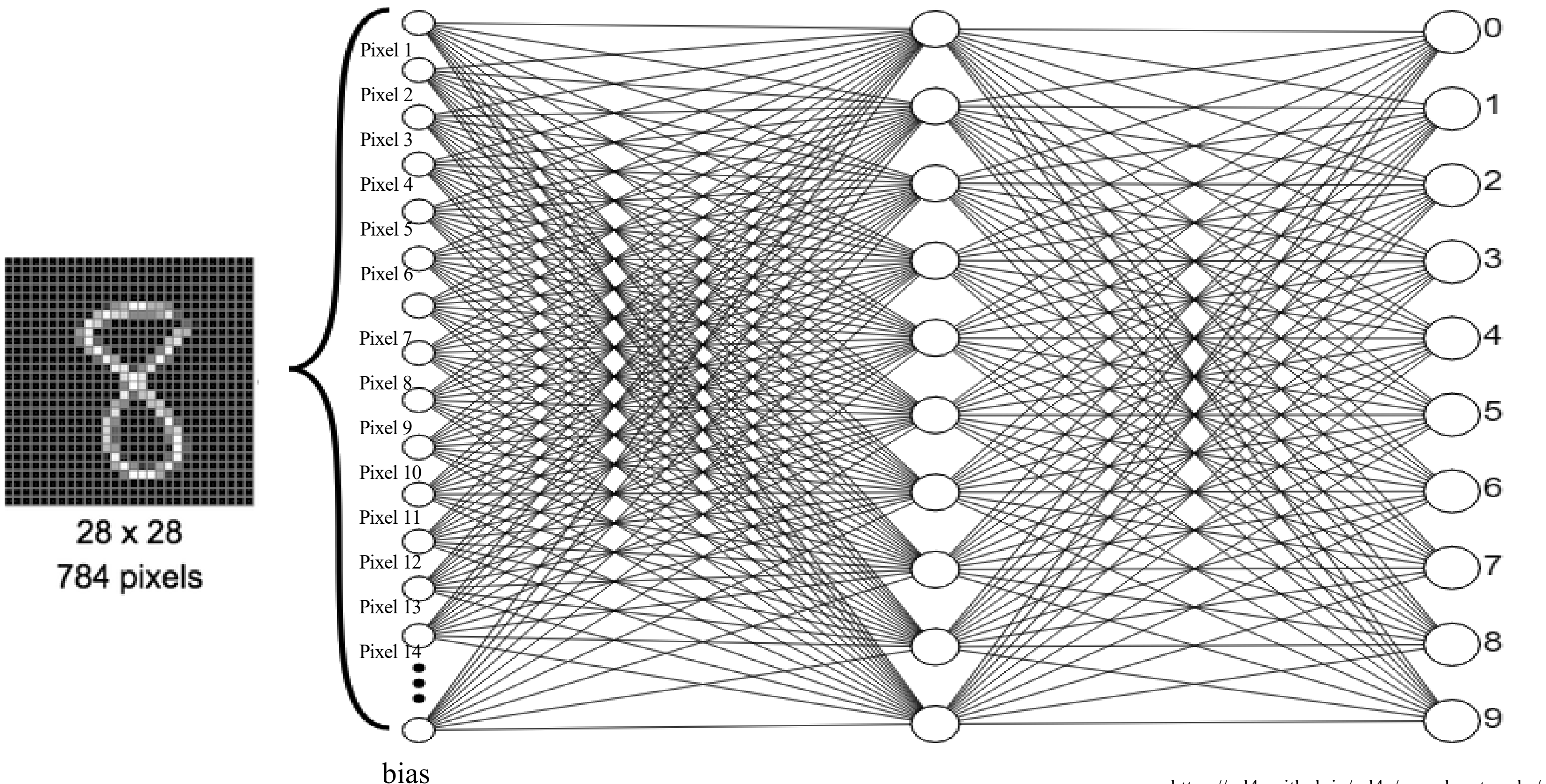
https://ml4a.github.io/ml4a/neural_networks/

How to classify an image?



https://ml4a.github.io/ml4a/neural_networks/

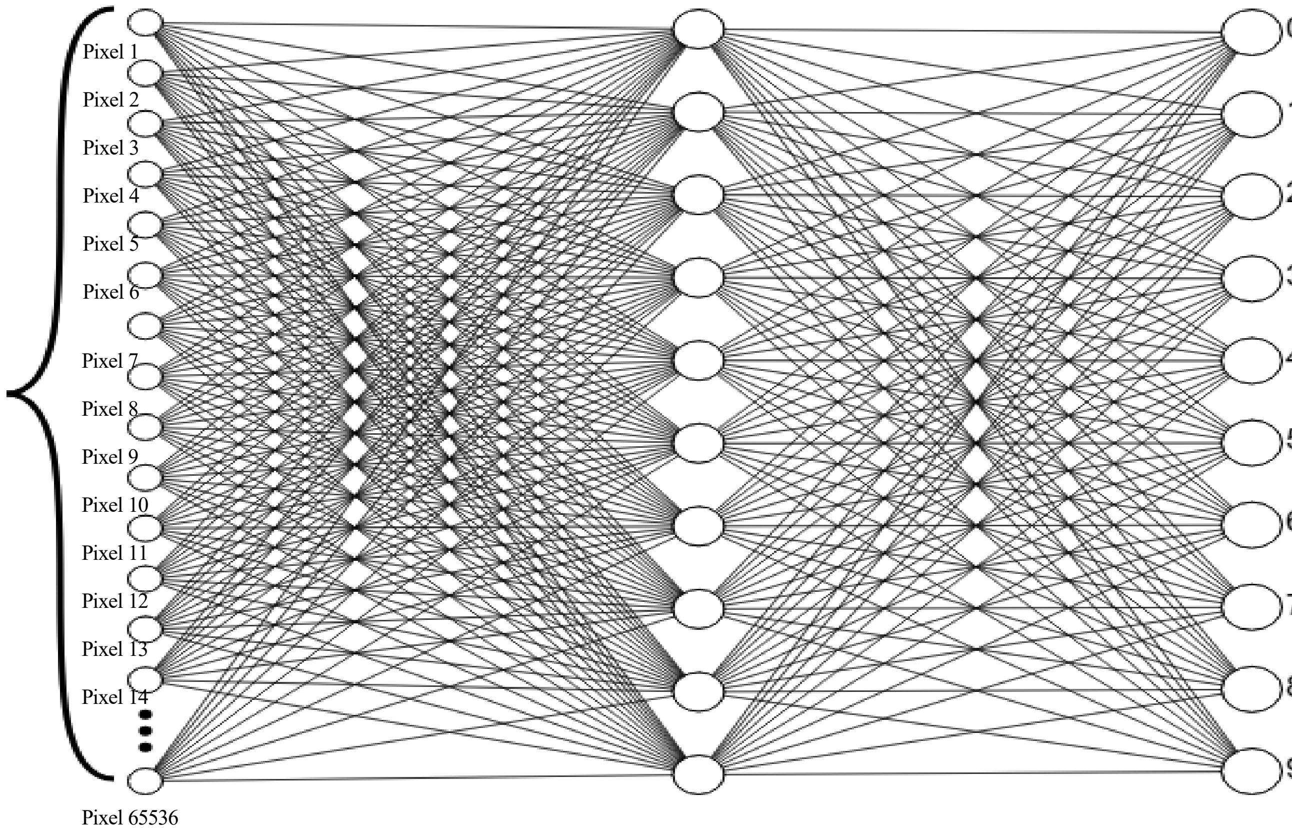
Many parameters (7850 in Layer 1)



https://ml4a.github.io/ml4a/neural_networks/

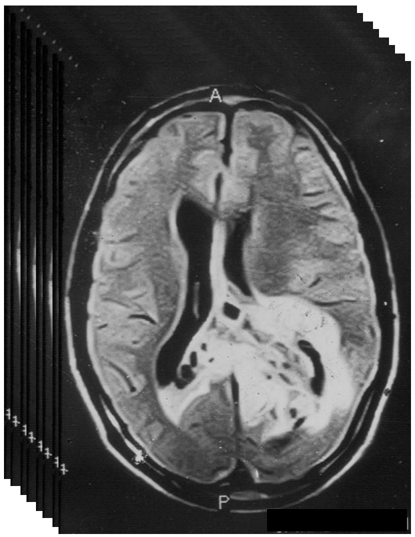
Too many parameters (655,370 in Layer 1)

256x256

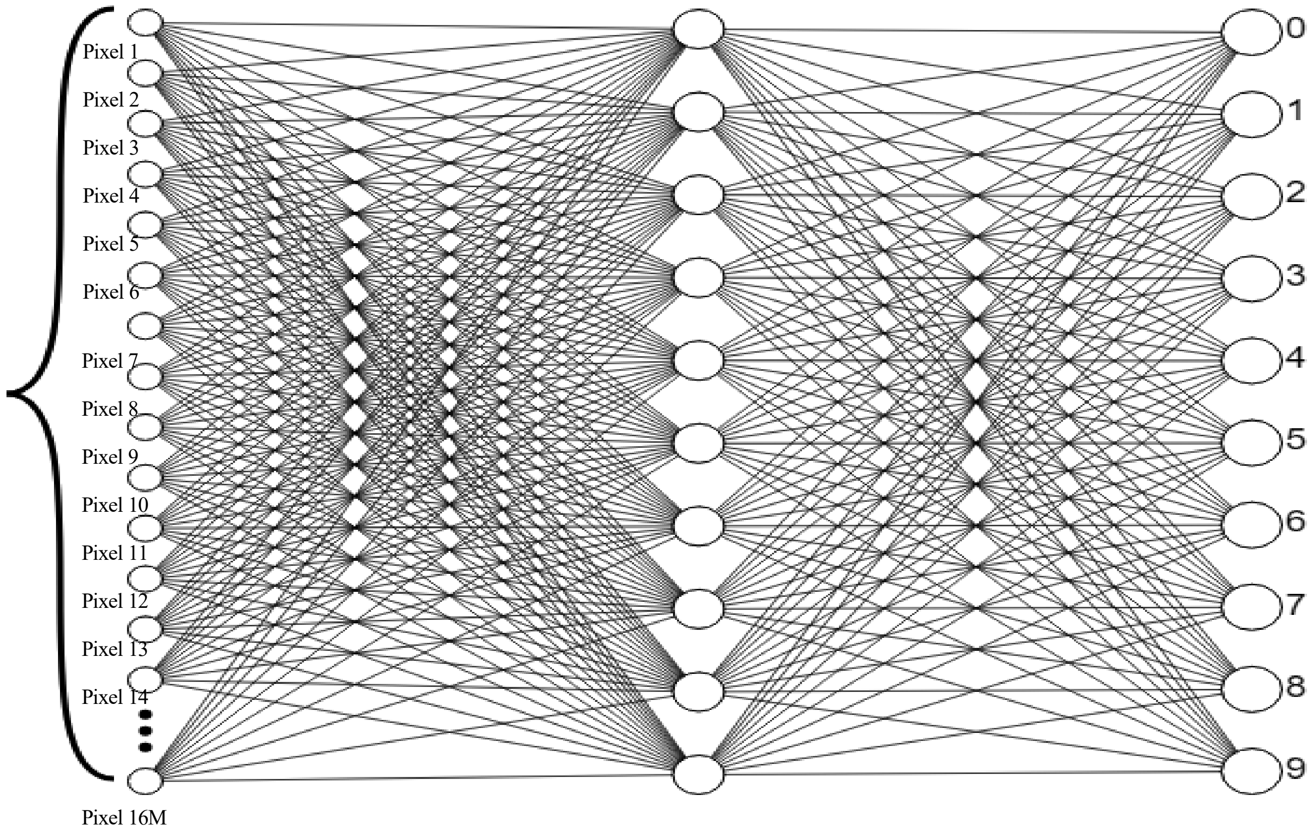


https://ml4a.github.io/ml4a/neural_networks/

Waaay too many parameters (160M in Layer 1)

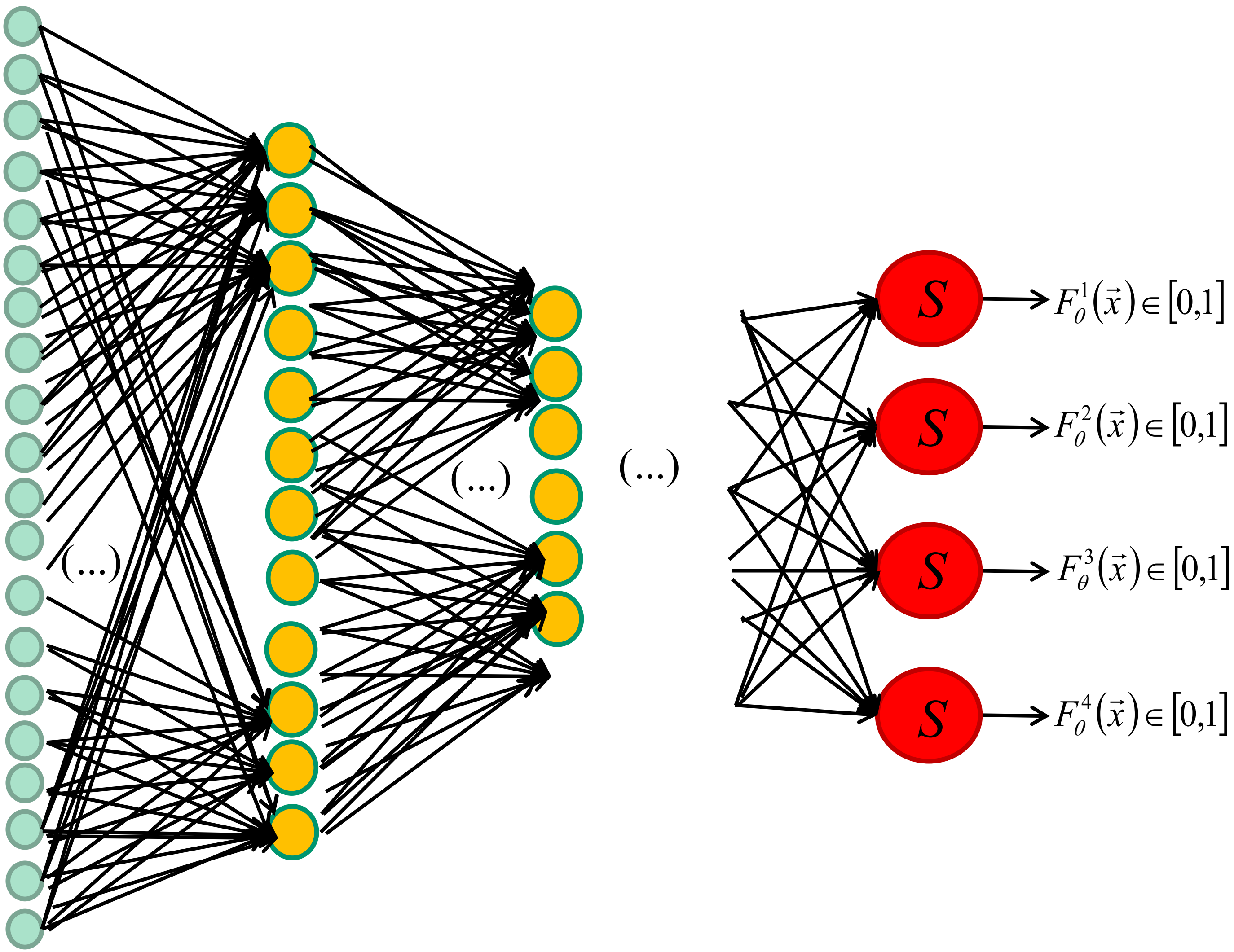


256x256x256



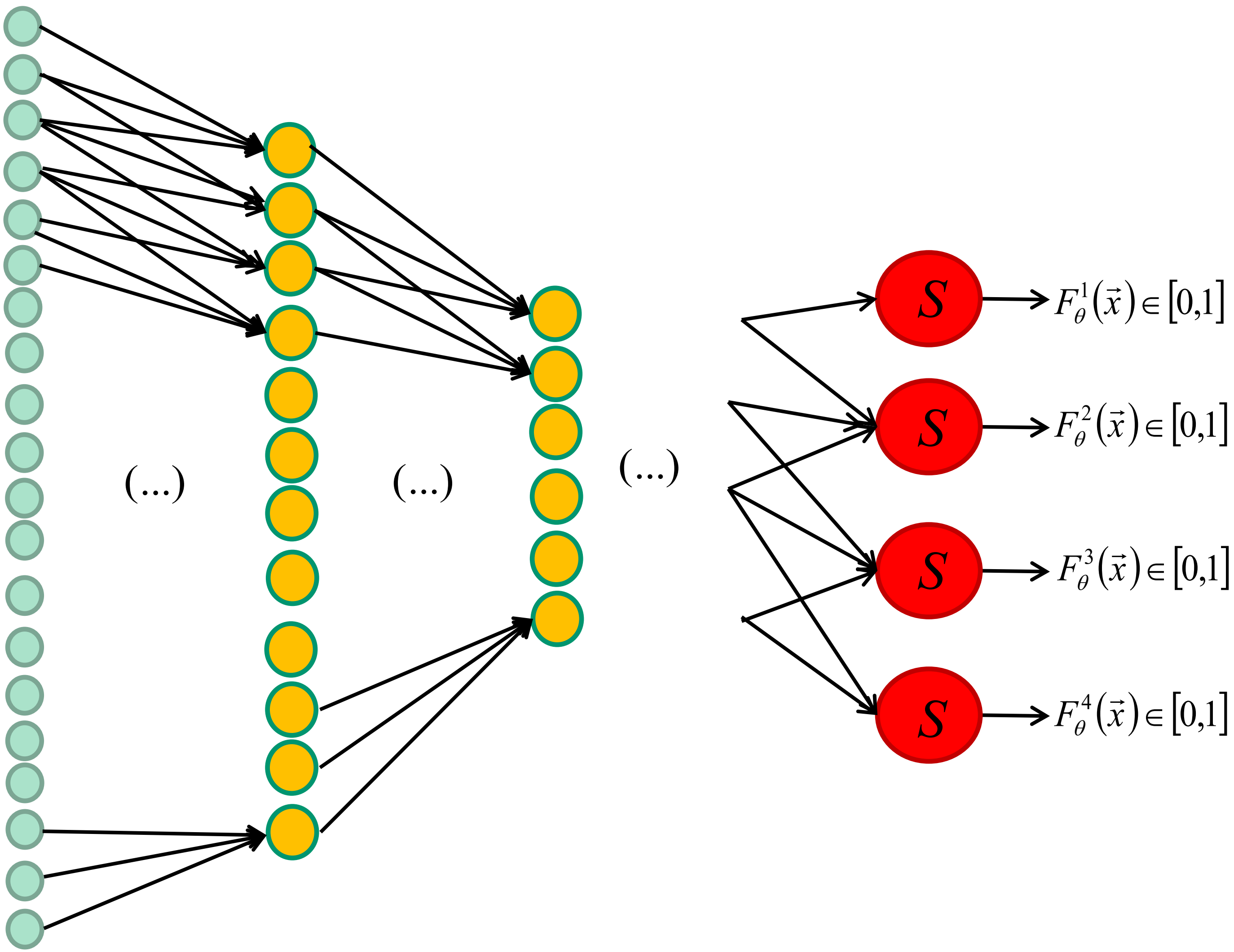
https://ml4a.github.io/ml4a/neural_networks/

Full connections are too many



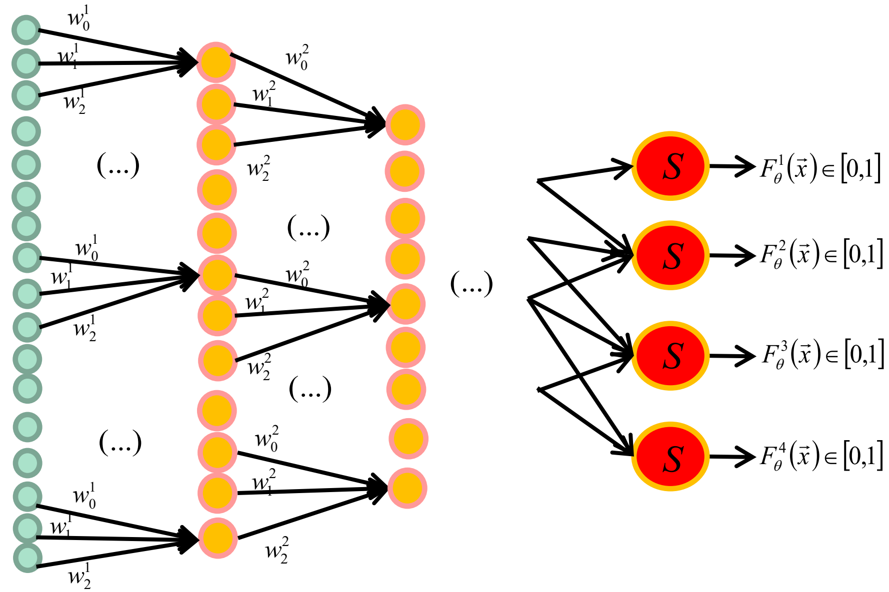
150-D input vector with 150 neurons in Layer 1 => **22,500 parameters!!**

No full connection



150-D input vector with 150 neurons in Layer 1 => **450 parameters!!**

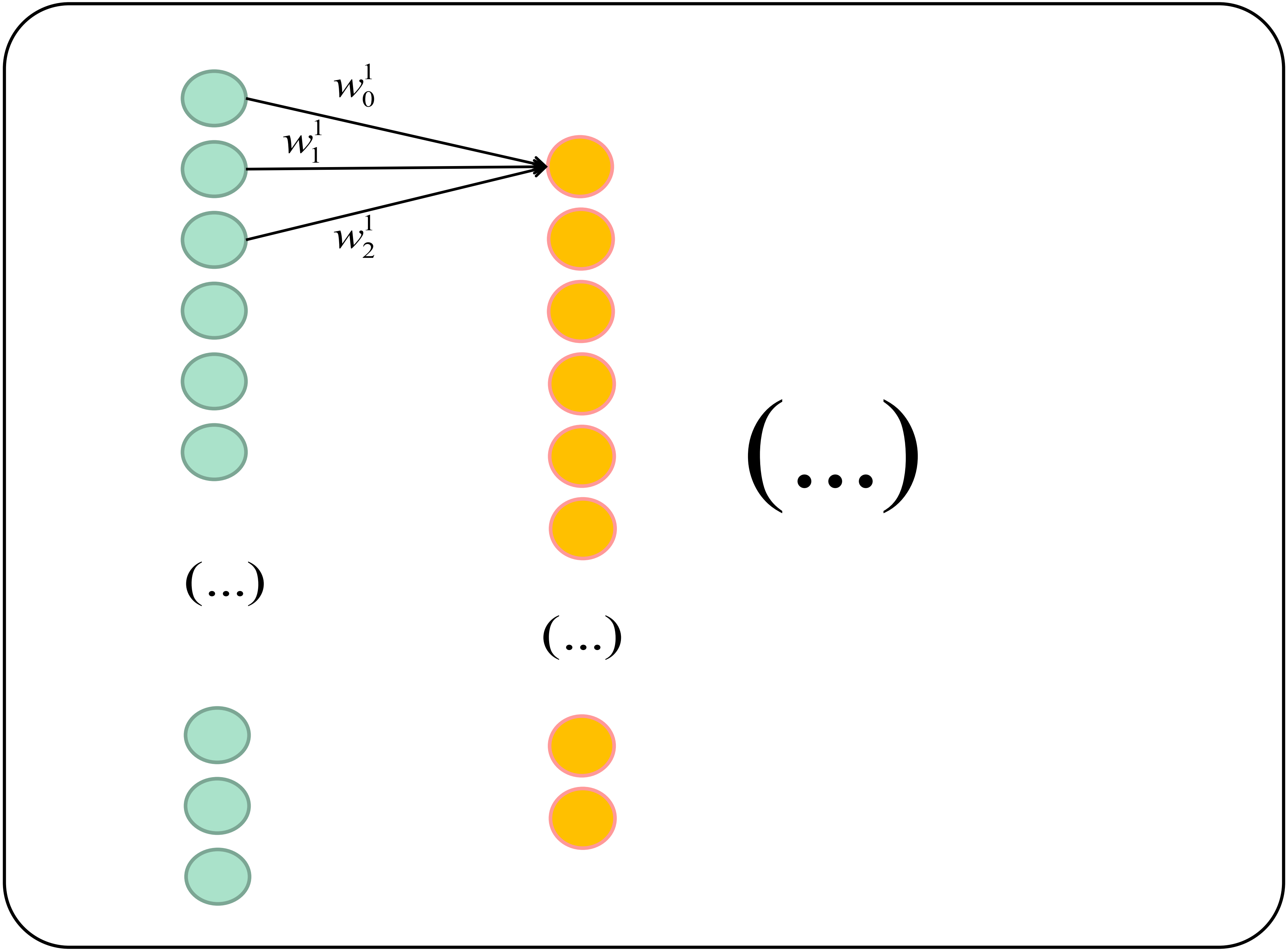
Share weights

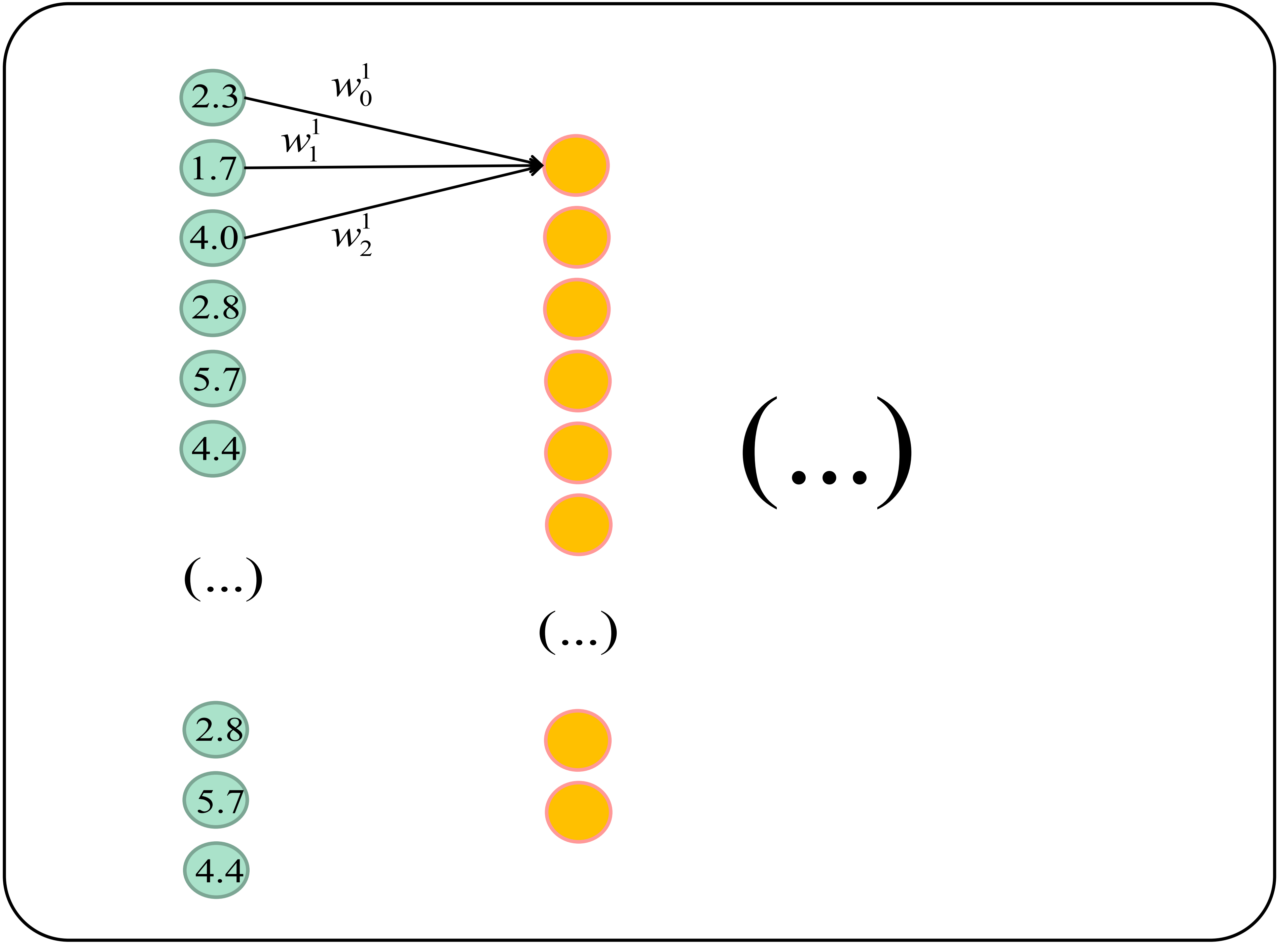


1- Learning convolution filters!

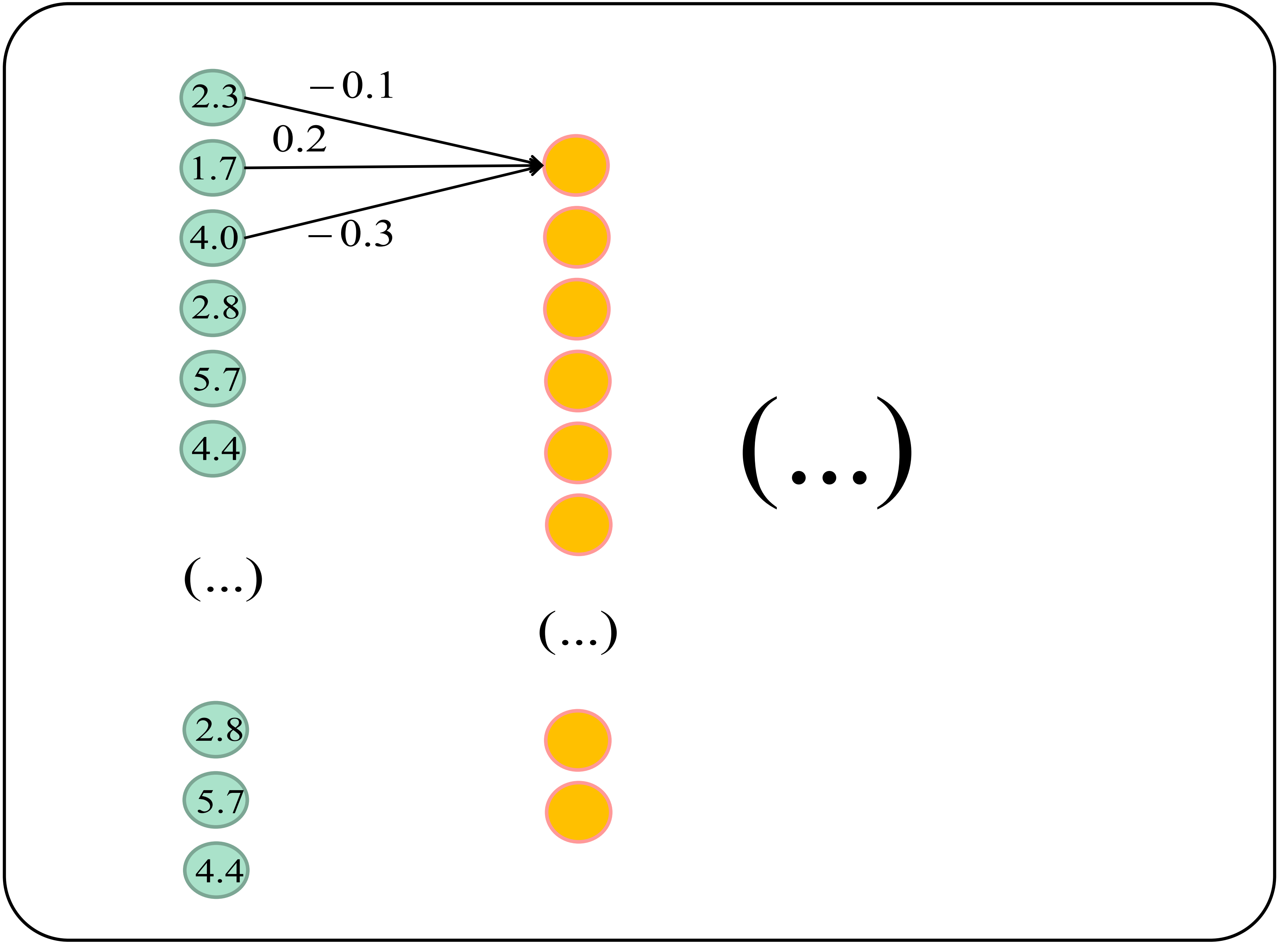
2- Small number of parameters = can make it deep!

150-D input vector with 150 neurons in Layer 1 => **3 parameters!!**

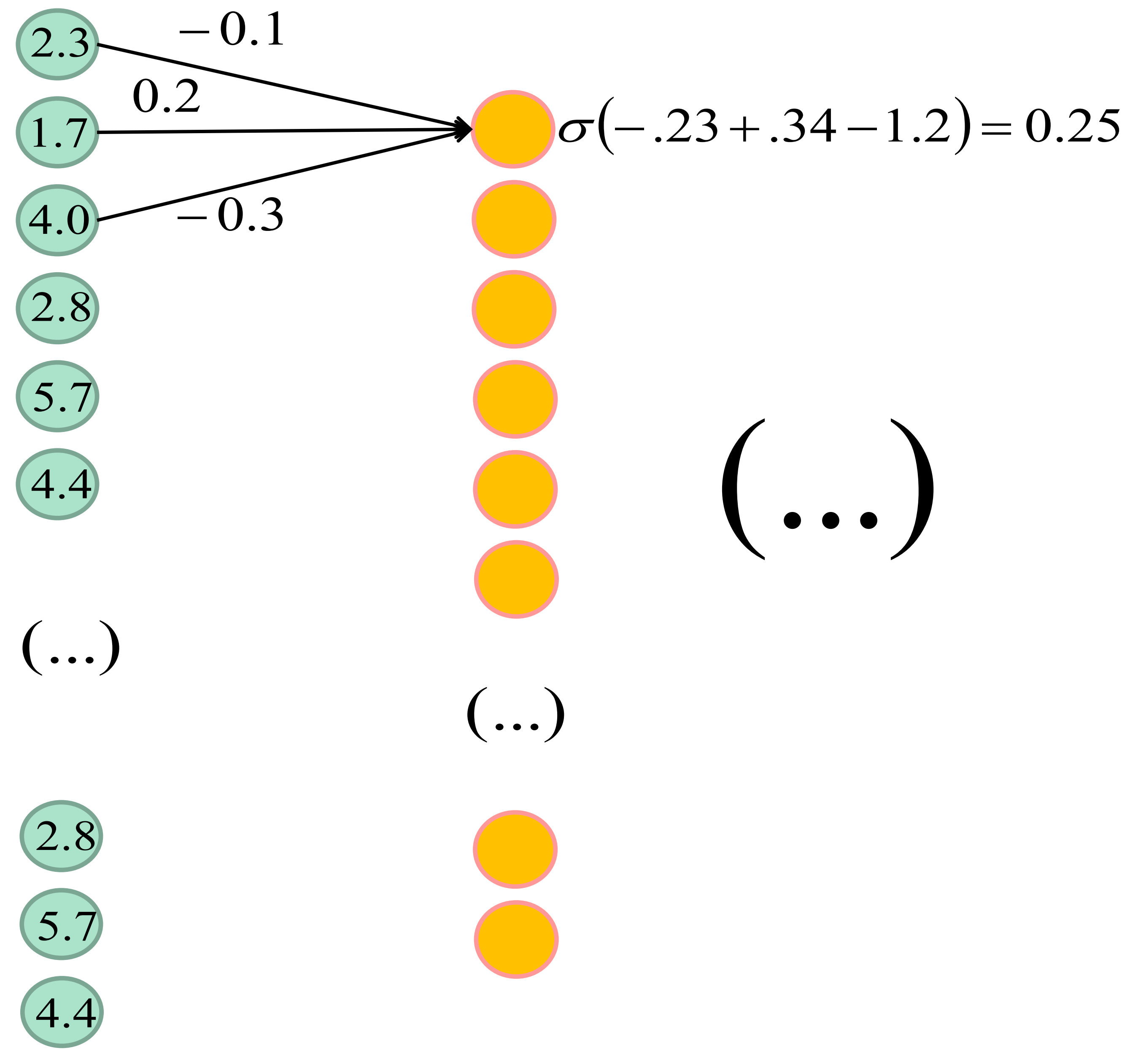

w_0^1
w_1^1
w_2^1
(...)
(...)
(...)

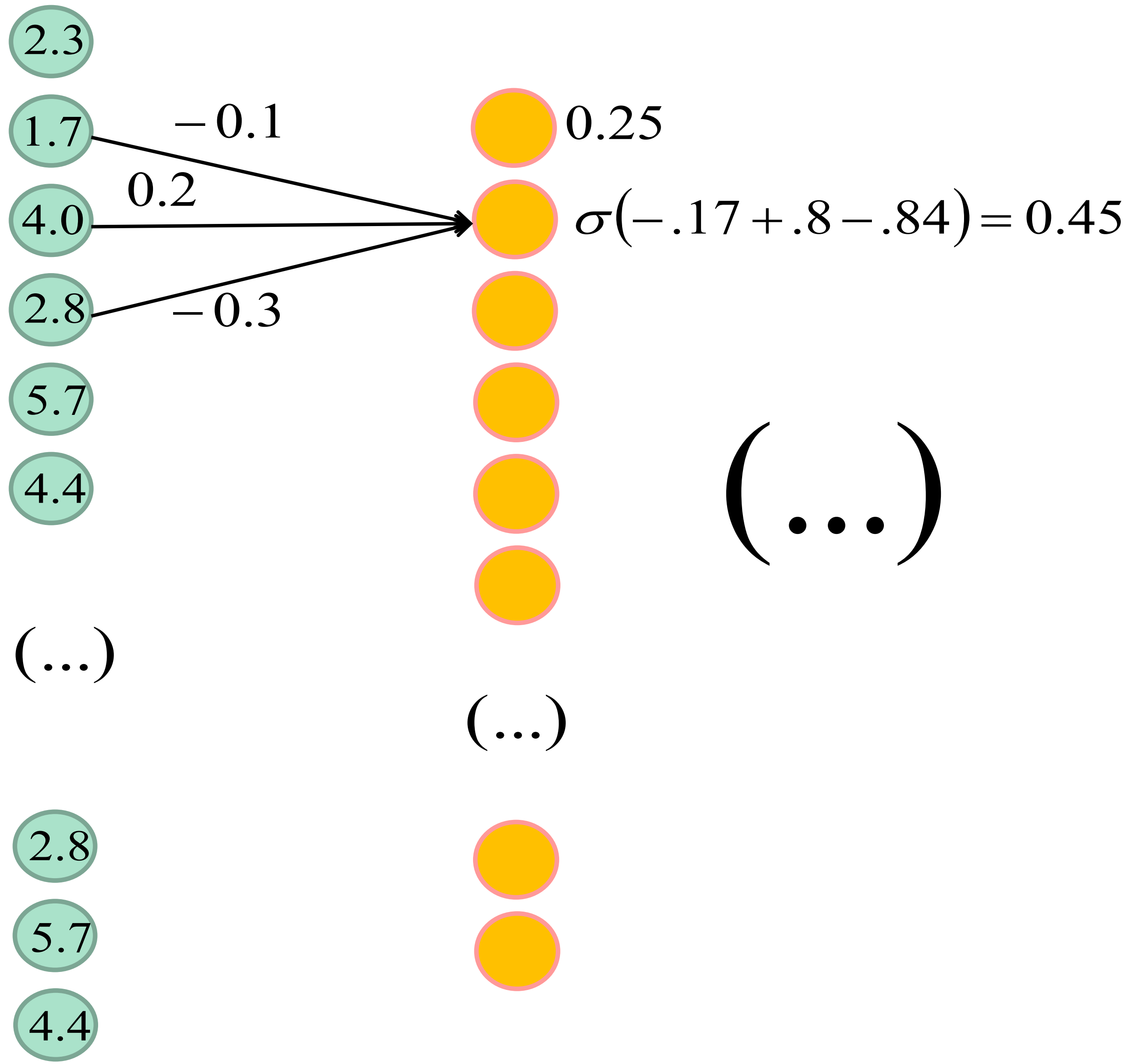


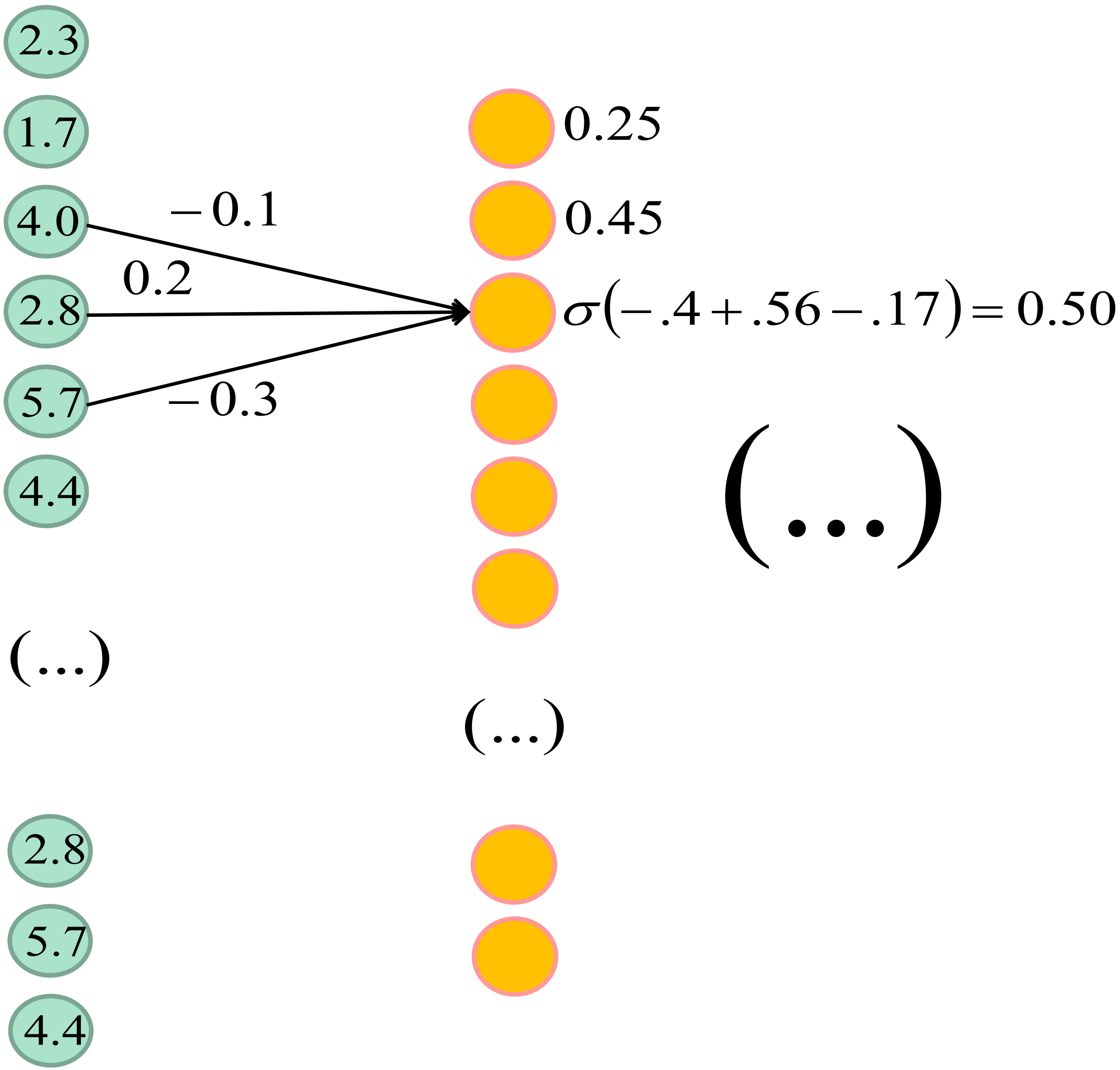
2.3
1.7
4.0
2.8
5.7
4.4
(...)
2.8
5.7
4.4
w_0^1
w_1^1
w_2^1
(...)
(...)



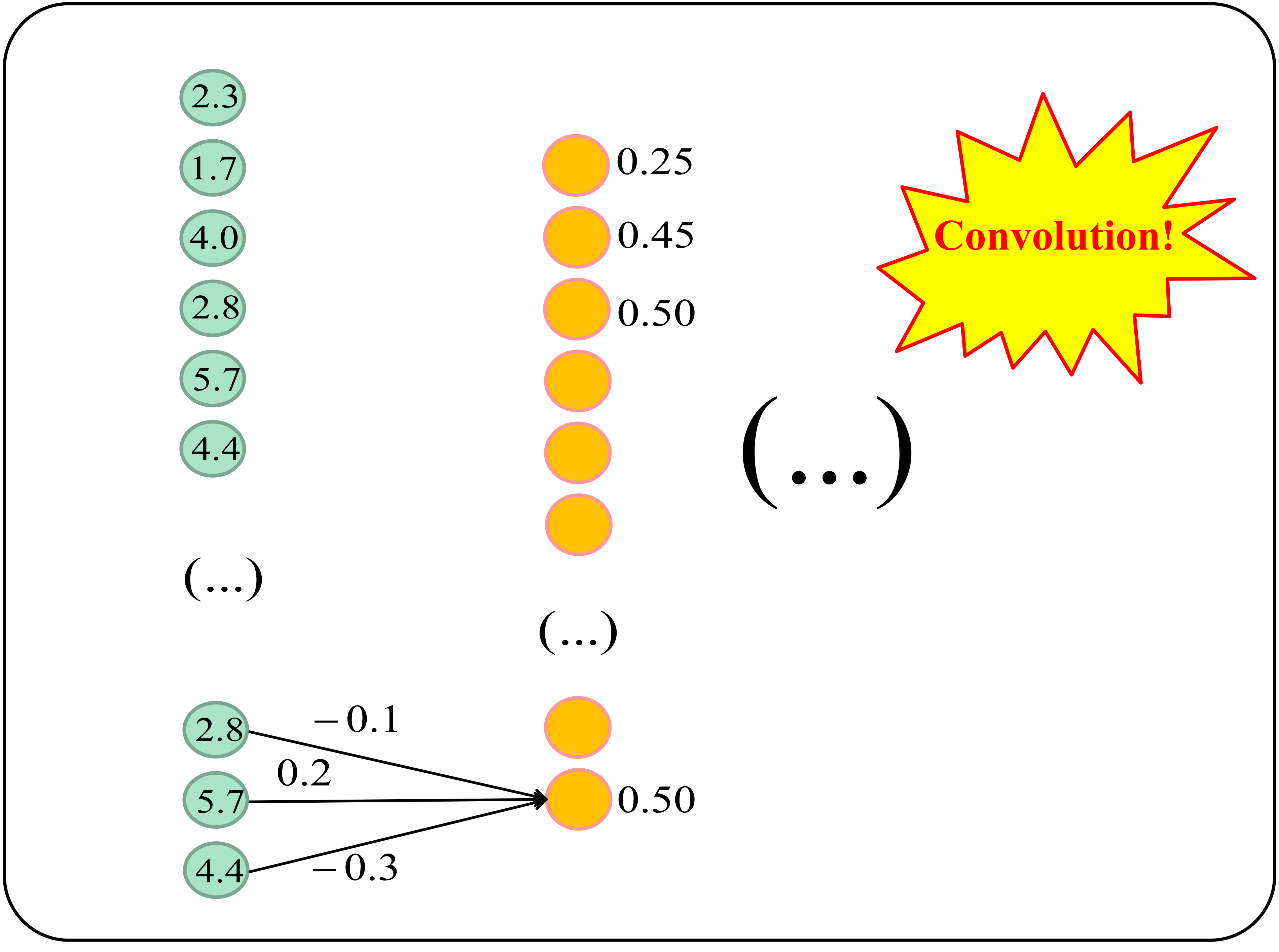
2.3
1.7
4.0
2.8
5.7
4.4
(...)
2.8
5.7
4.4
− 0.1
0.2
− 0.3
(...)
(...)


2.3
1.7
4.0
2.8
5.7
4.4
(...)
2.8
5.7
4.4
−0.1
0.2
−0.3
σ(−.23 + .34 − 1.2) = 0.25
(...)
(...)

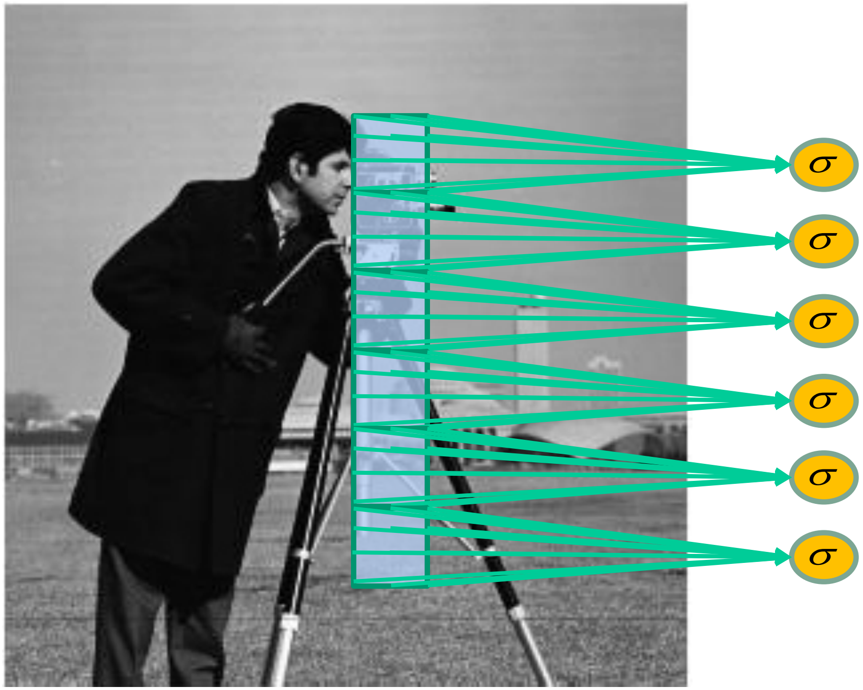

2.3
1.7
4.0
2.8
5.7
4.4
(...)
2.8
5.7
4.4
−0.1
0.2
−0.3
0.25
σ(−.17 + .8 − .84) = 0.45
(...)
(...)



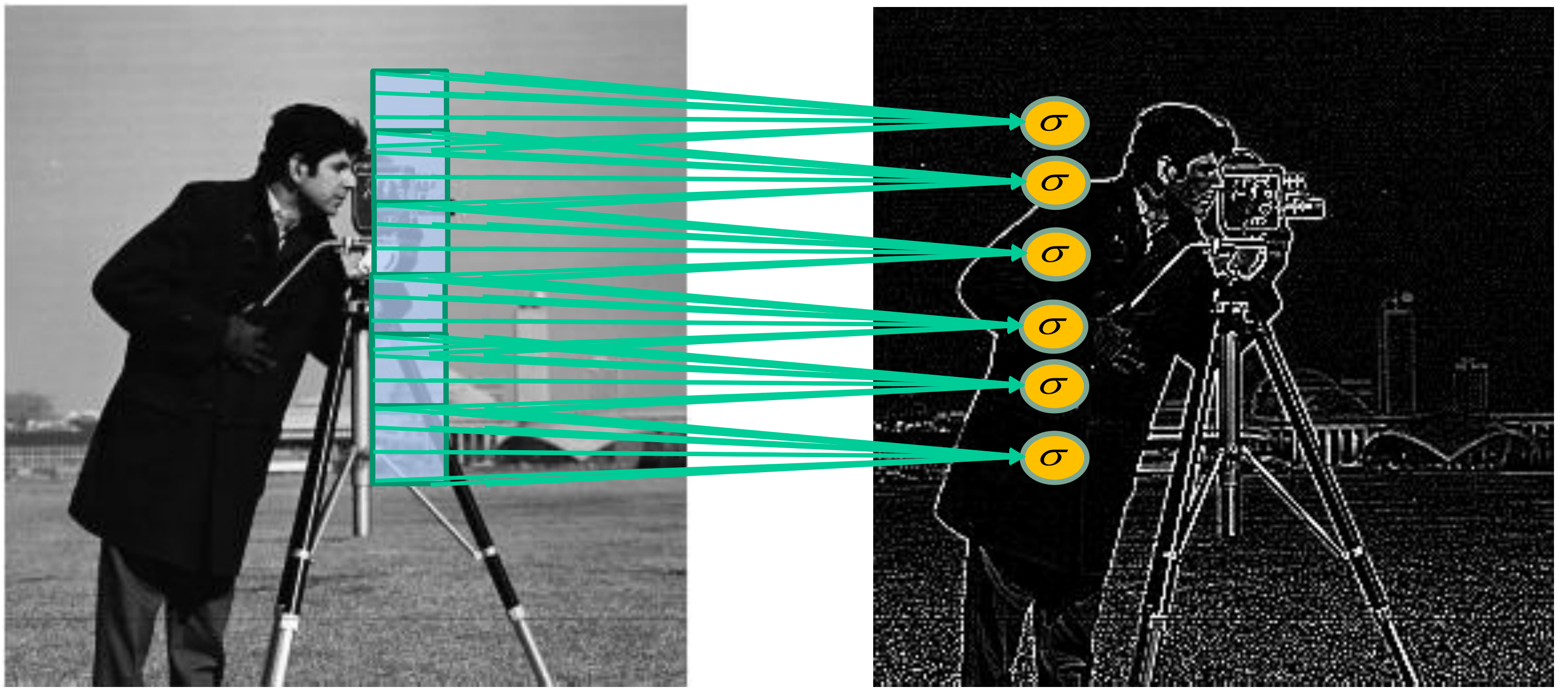
2.3
1.7
4.0
2.8
5.7
4.4
(...)
2.8
5.7
4.4
−0.1
0.2
−0.3
0.25
0.45
σ(−.4 + .56 − .17) = 0.50
(...)
(...)



2.3
1.7
4.0
2.8
5.7
4.4
(...)
0.25
0.45
0.50
(...)
Convolution!
(...)
2.8
5.7
4.4
−0.1
0.2
−0.3
0.50

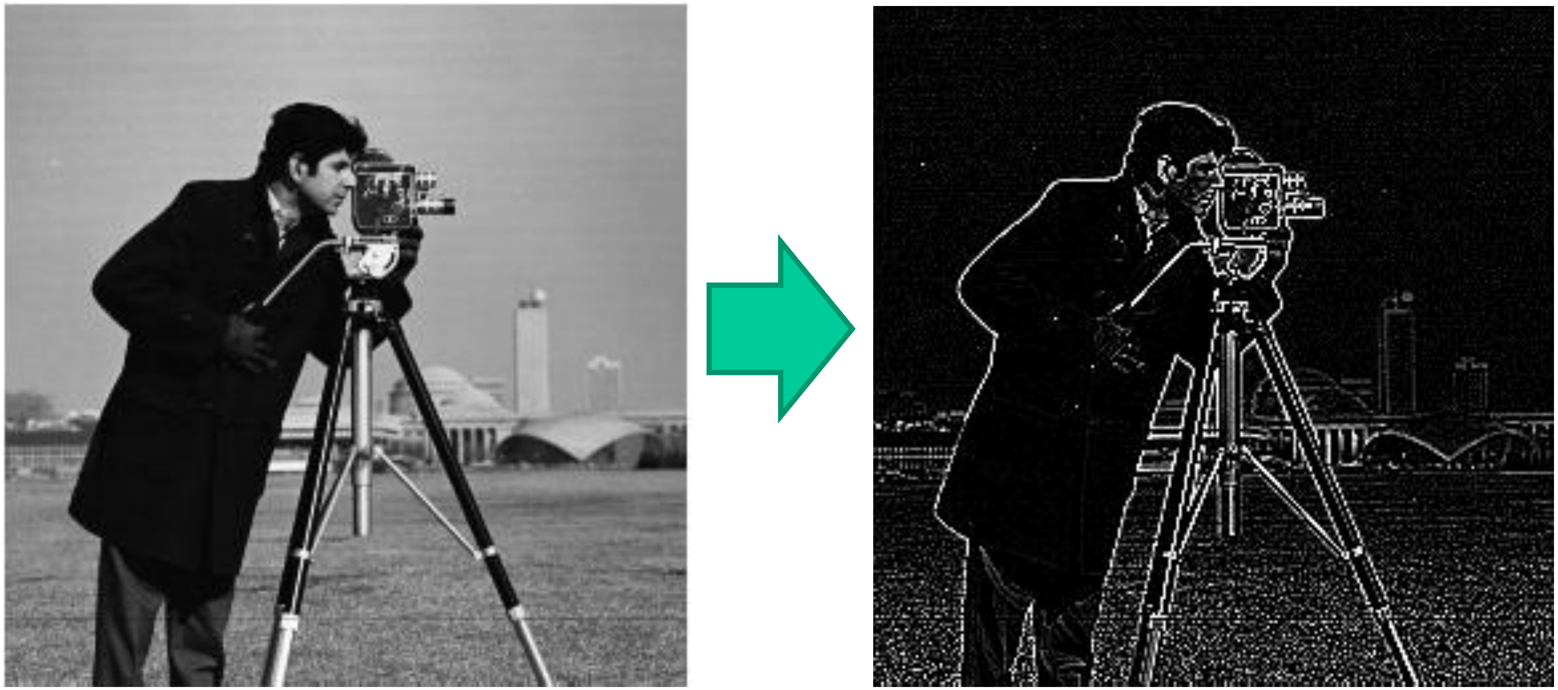


Each neuron of layer 1 is connected to 3x3 pixels, layer 1 has **9 parameters!!**

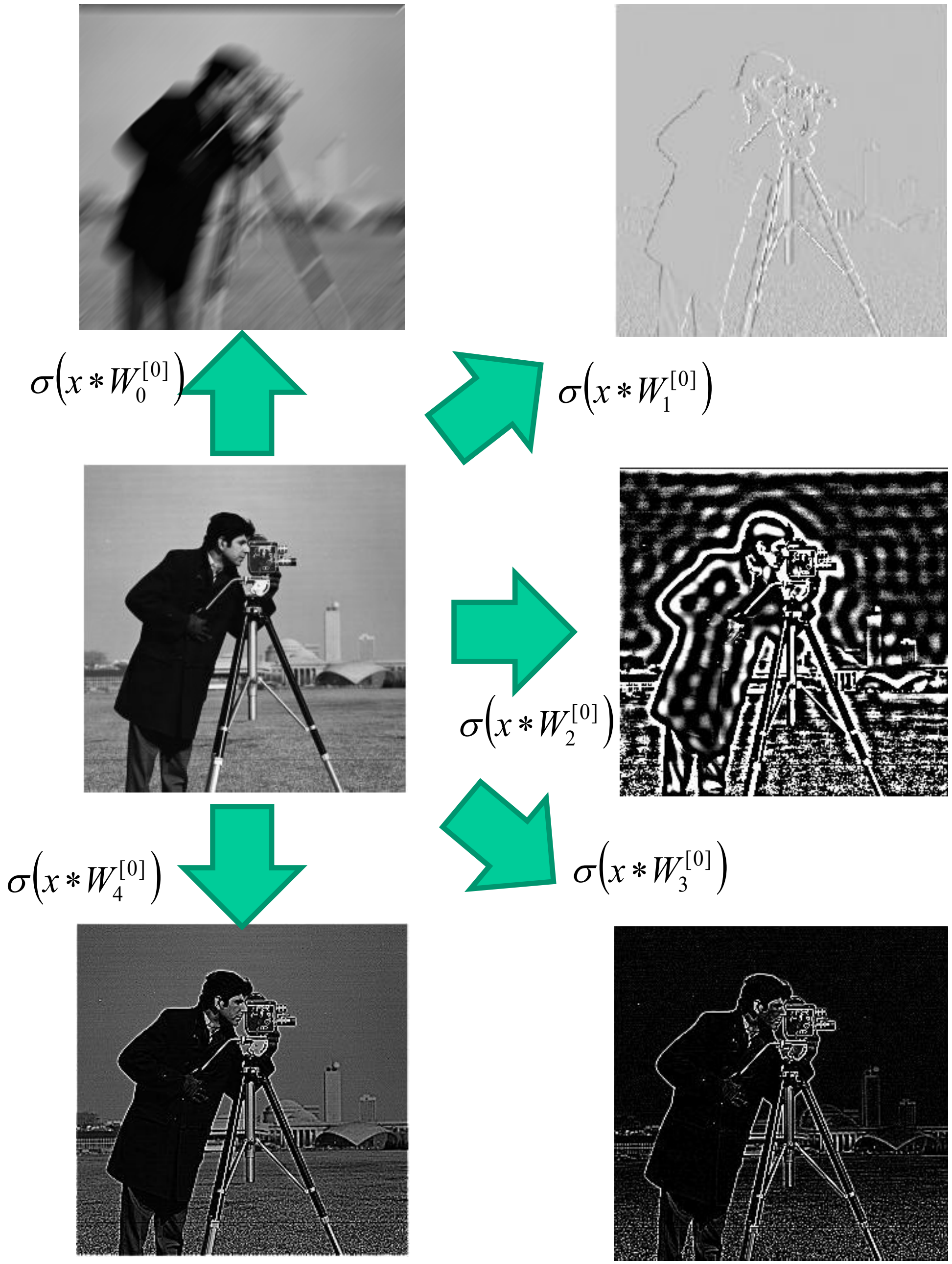

σ
σ
σ
σ
σ
σ

Convolution operation

Feature map

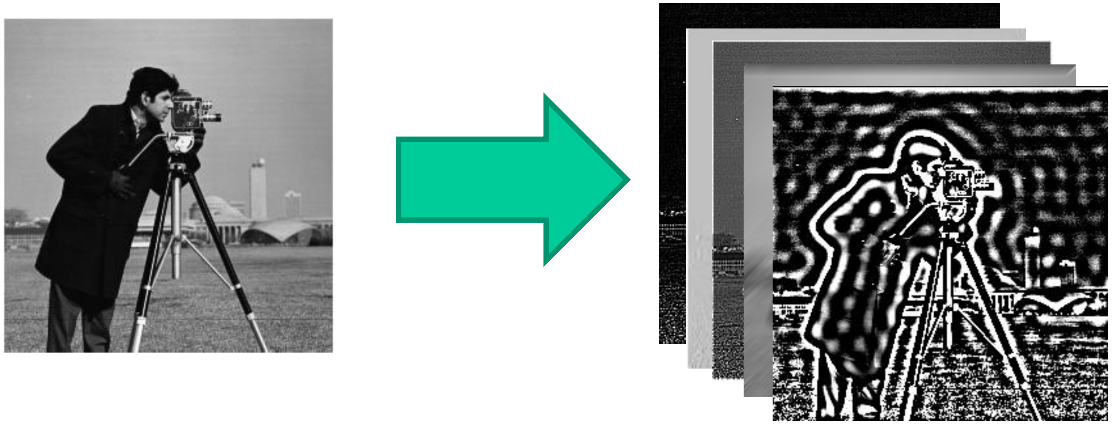


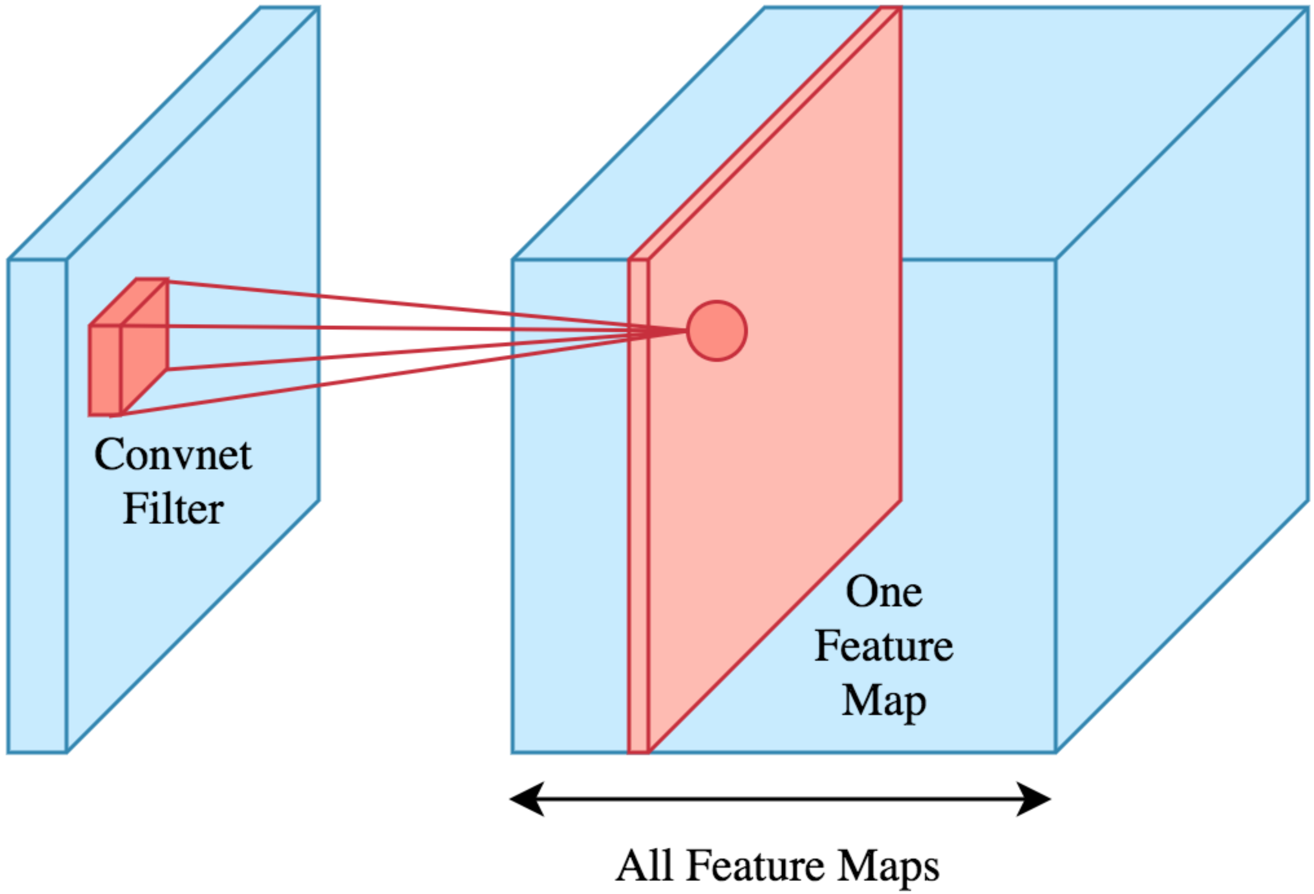
$$F = \sigma\left(x * W^{[0]}\right)$$



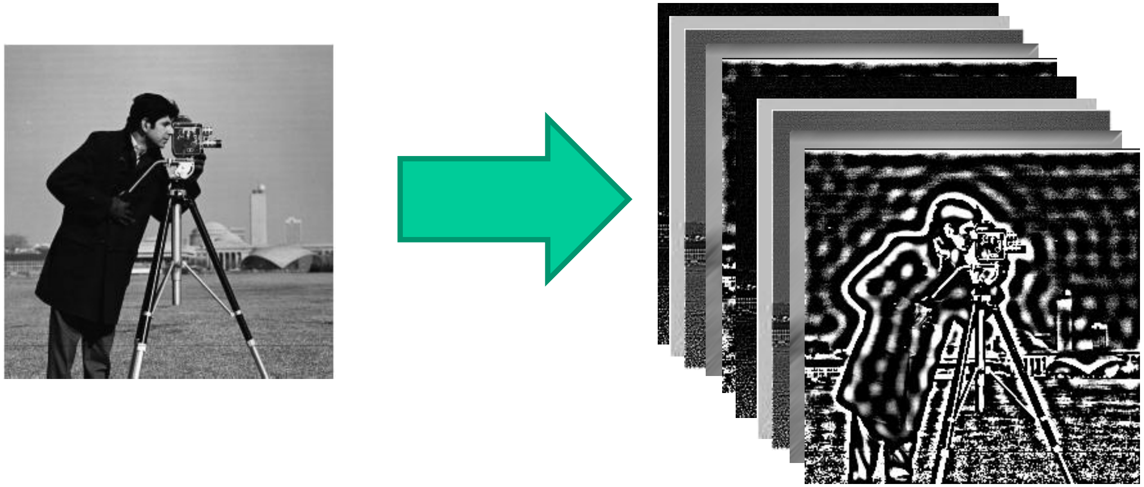
$\sigma\left(x * W_0^{[0]}\right)$
$\sigma\left(x * W_1^{[0]}\right)$
$\sigma\left(x * W_2^{[0]}\right)$
$\sigma\left(x * W_3^{[0]}\right)$
$\sigma\left(x * W_4^{[0]}\right)$

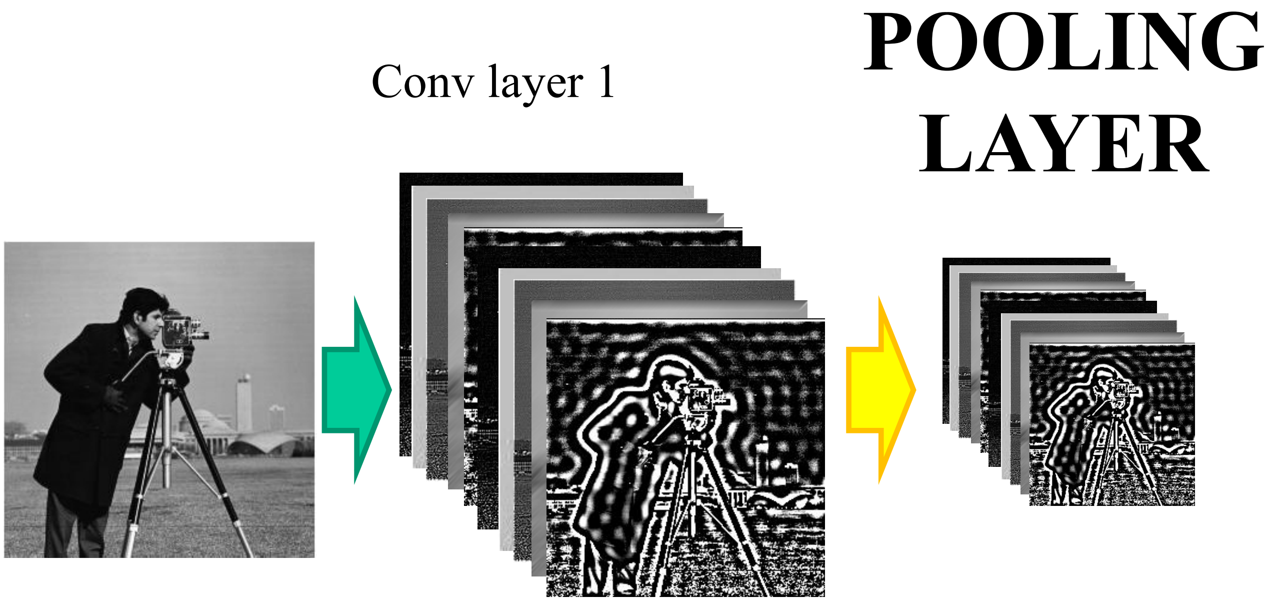
5-feature map convolution layer



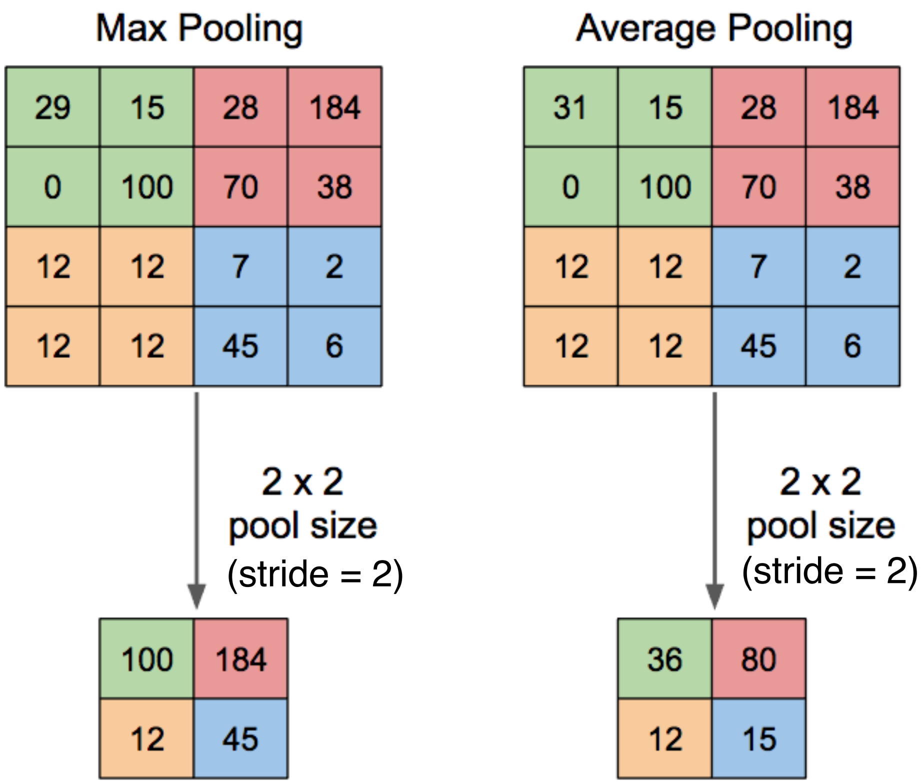

Convnet
Filter
One
Feature
Map
All Feature Maps

K-feature map convolution layer




Conv layer 1
POOLING LAYER

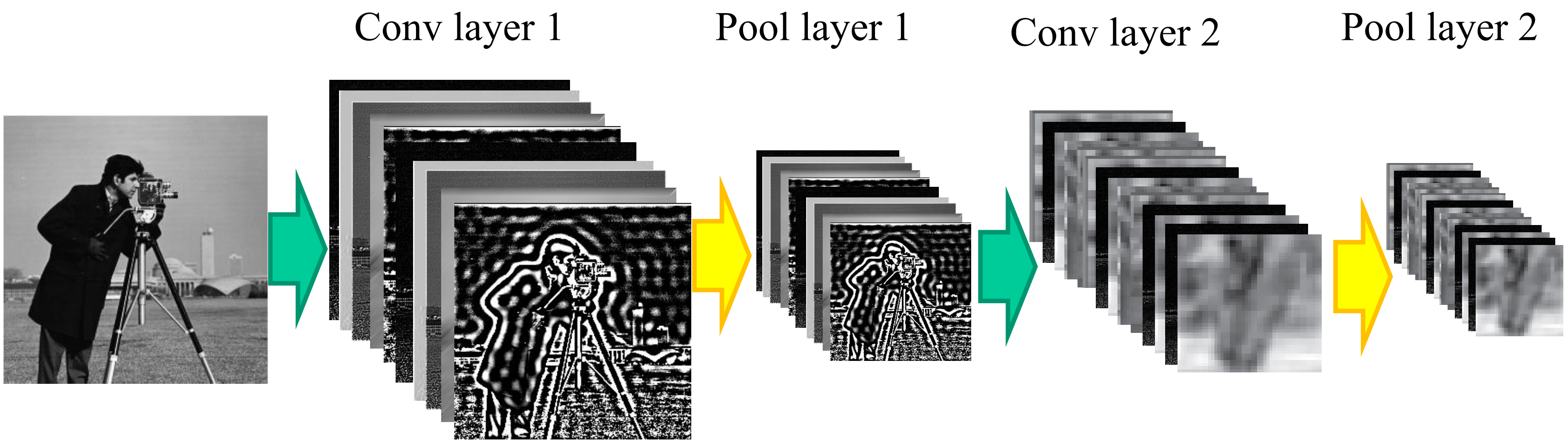
Pooling layer



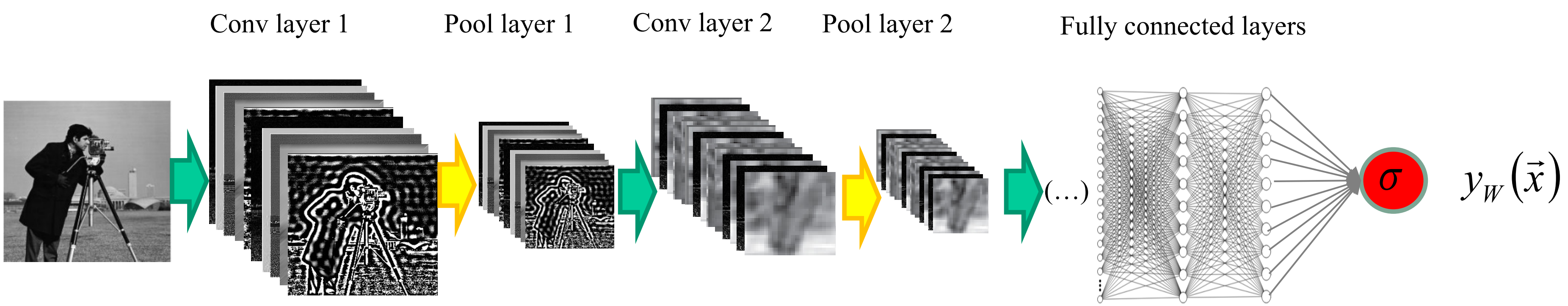
Goals

- Reduce the spatial resolution of feature maps
- Lower memory and computation requirements
- Provide partial invariance to position, scale and rotation

Images: https://pythonmachinelearning.pro/introduction-to-convolutional-neural-networks-for-vision-tasks/

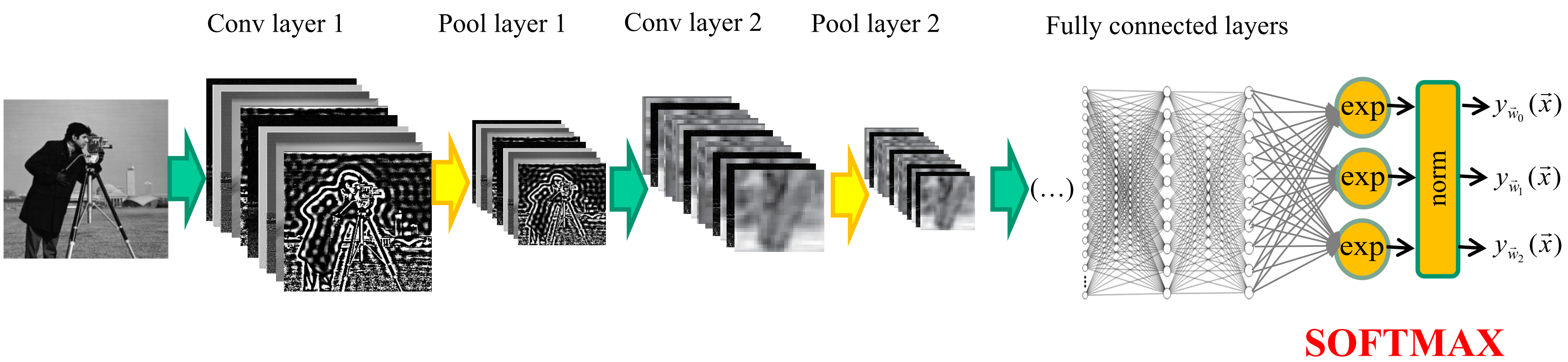

Conv layer 1
Pool layer 1
Conv layer 2
Pool layer 2

2 Class CNN



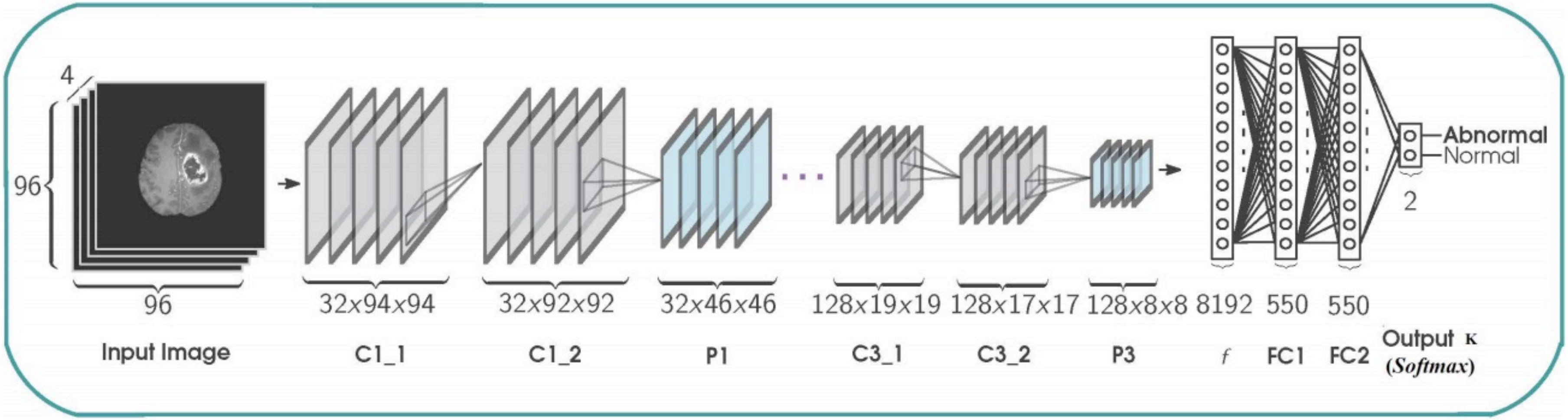
$$l\left(y_W(\vec{x}), t\right) = -t \ln\left(y_W(\vec{x})\right) - (1-t)\ln\left(1 - y_W(\vec{x})\right)$$

K Class CNN



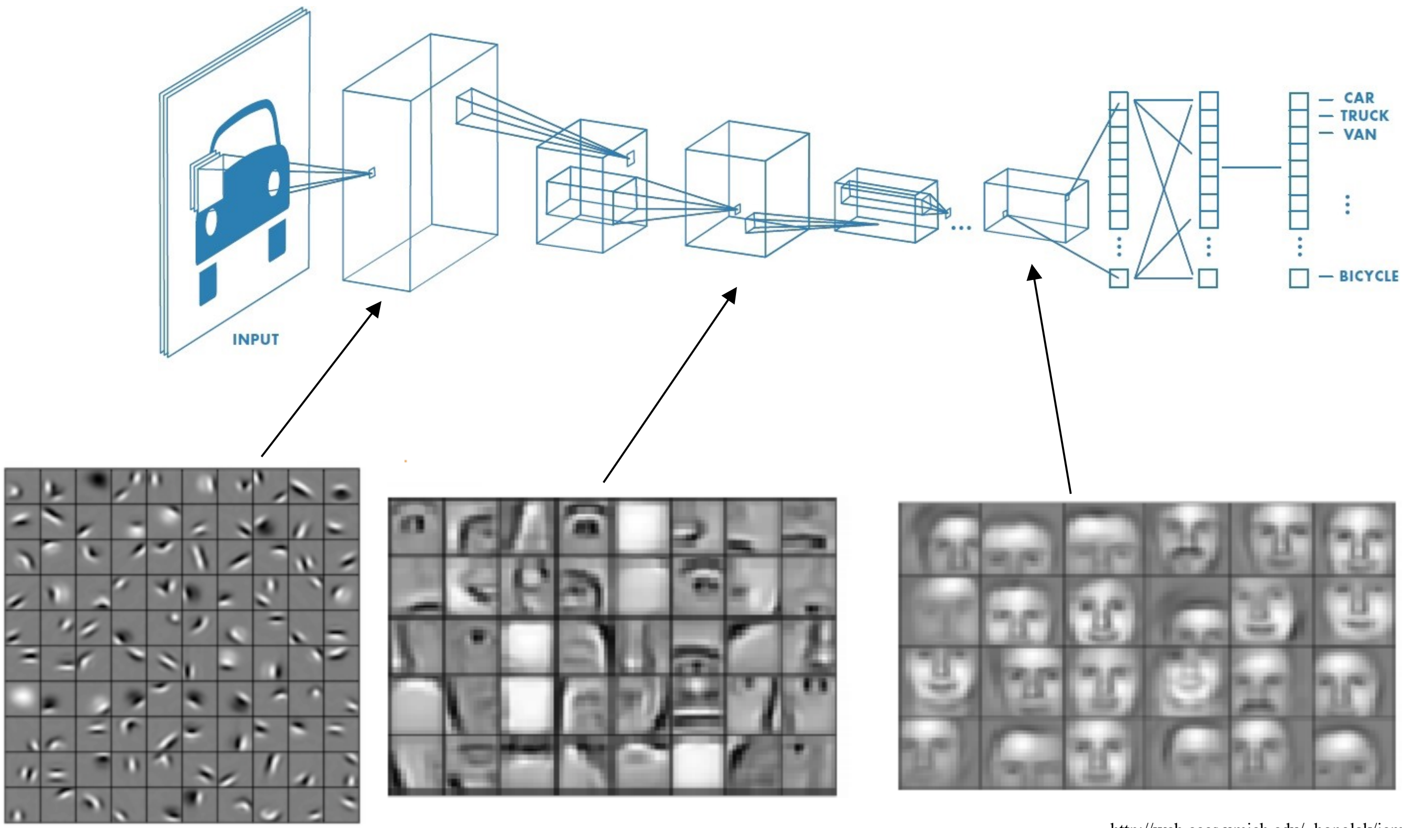
$$l\left(y_W\left(\vec{x}\right), t\right) = -t\ln\left(y_W\left(\vec{x}\right)\right) - (1-t)\ln\left(1 - y_W\left(\vec{x}\right)\right)$$

Nice example from the litterature



S. Banerjee, S. Mitra, A. Sharma, and B. U Shankar, A CADe System for Gliomas in Brain MRI using Convolutional Neural Networks, arXiv:1806.07589v, 2018

Learn image-based caracteristics

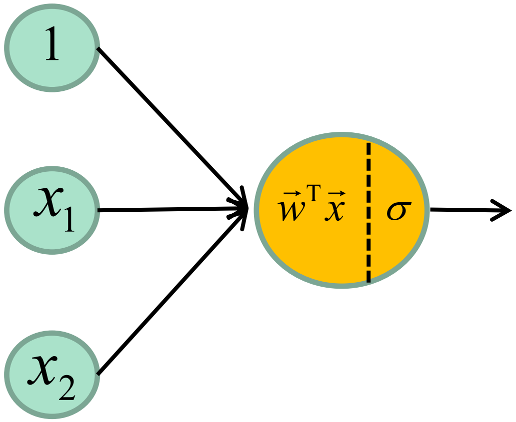


http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf

Batch processing

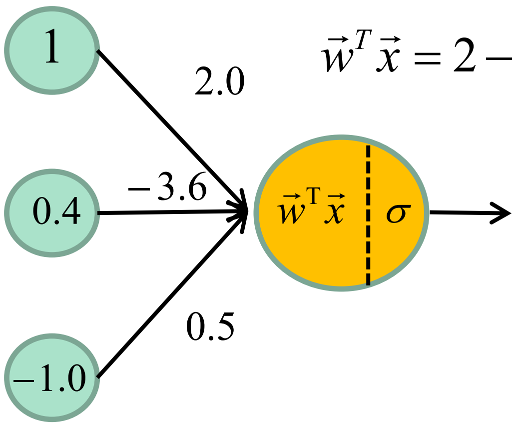
$$\vec{x} = (0.4, -1.0)$$

$$\vec{w} = [2.0, -3.6, 0.5]$$



$$\vec{x} = (0.4, -1.0)$$

$$\vec{w} = [2.0, -3.6, 0.5]$$

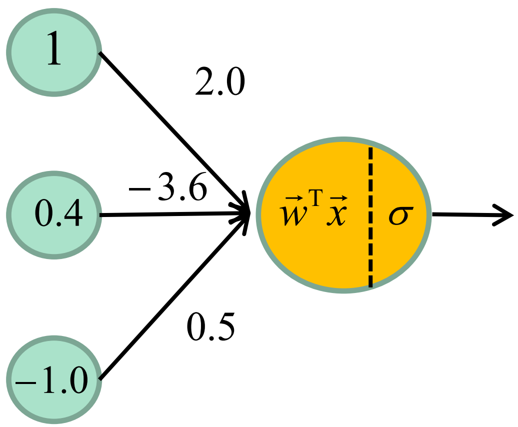


$$\vec{w}^T\vec{x} = 2 - 3.6 * 0.4 - 0.5 = -1.94$$

$$y_{\vec{w}}(\vec{x}) = \sigma(-1.94) = \frac{1}{1+e^{1.94}} = 0.125$$

$$\vec{x} = (0.4, -1.0)$$

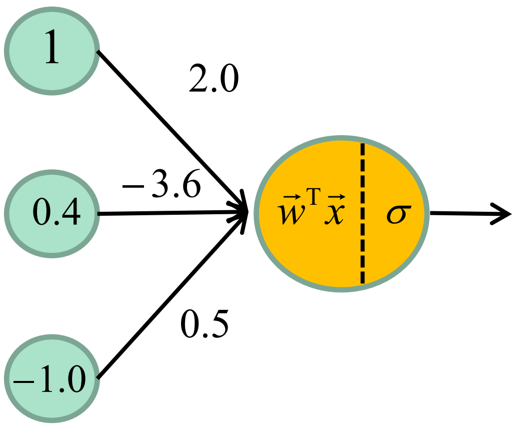
$$\vec{w} = [2.0, -3.6, 0.5]$$



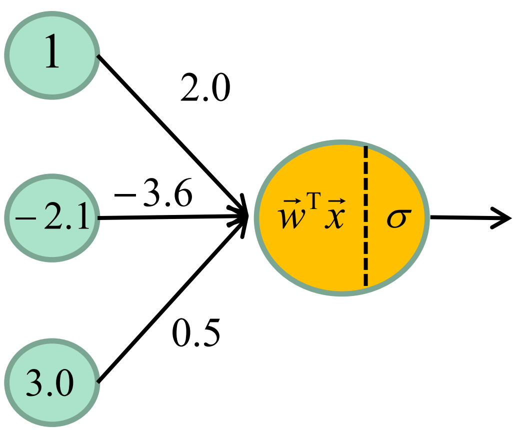
$$y_{\vec{w}}(\vec{x}_a) = \sigma\left((2.0, -3.6, 0.5) \begin{pmatrix} 1 \\ 0.4 \\ -1 \end{pmatrix} \right) = 0.125$$

$$\vec{x}_a = (0.4, -1.0),\ \vec{x}_b = (-2.1, 3.0)$$

$$\vec{w} = [2.0, -3.6, 0.5]$$

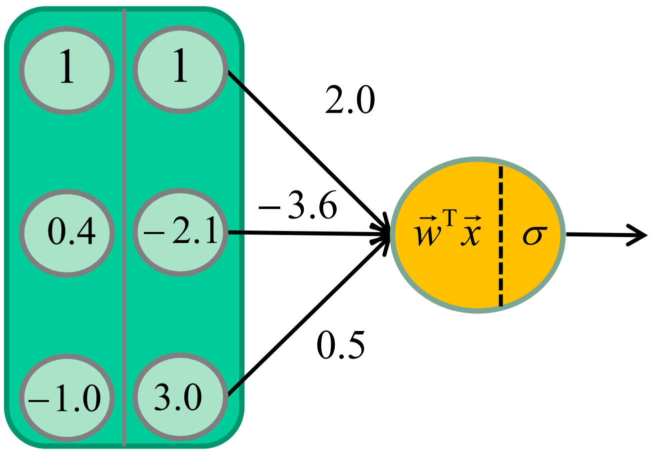


$$y_{\vec{w}}(\vec{x}_a) = \sigma\left((2.0, -3.6, 0.5) \begin{pmatrix} 1 \\ 0.4 \\ -1 \end{pmatrix} \right) = 0.125$$



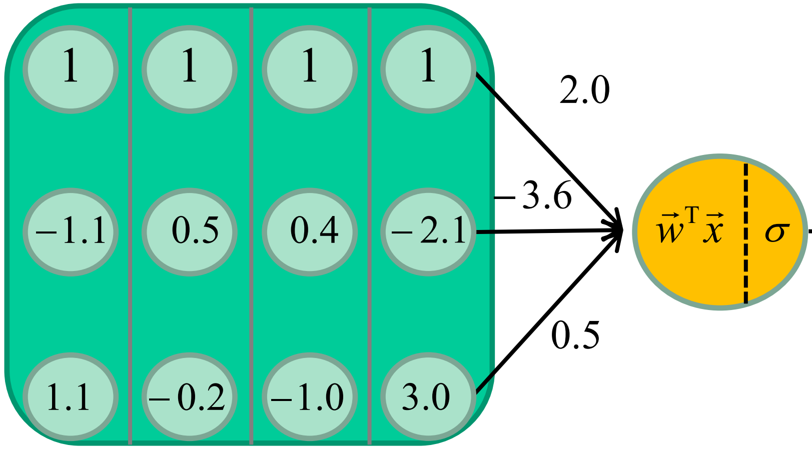
$$y_{\vec{w}}(\vec{x}_b) = \sigma\left((2.0, -3.6, 0.5) \begin{pmatrix} 1 \\ -2.1 \\ 3.0 \end{pmatrix} \right) = 0.99$$

Mini-batch processing



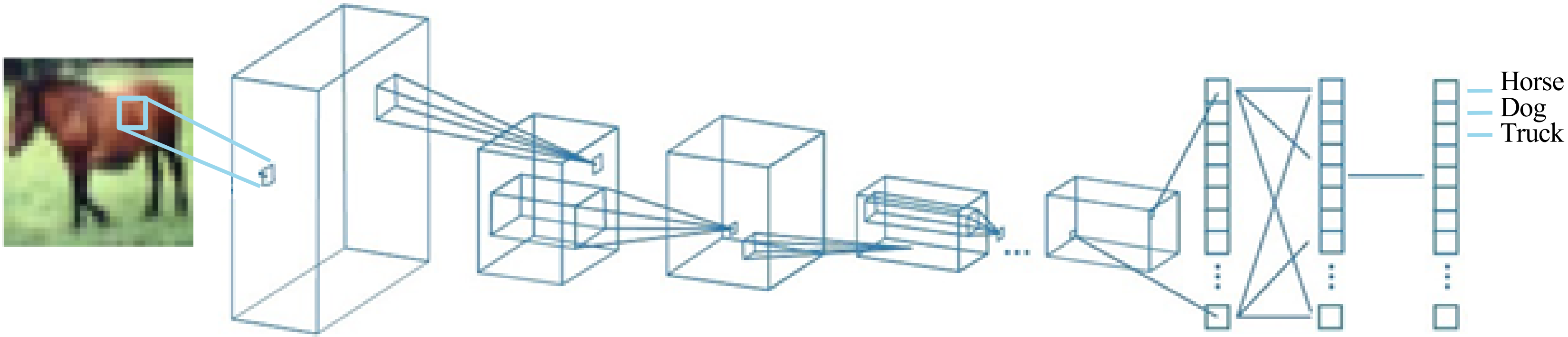
$$y_{\vec{w}}(\vec{x}_a) = \sigma\left((2.0,-3.6,0.5) \begin{pmatrix} 1 & 1 \\ 0.4 & -2.1 \\ -1 & 3.0 \end{pmatrix} \right) = (0.125, 0.99)$$

Mini-batch processing

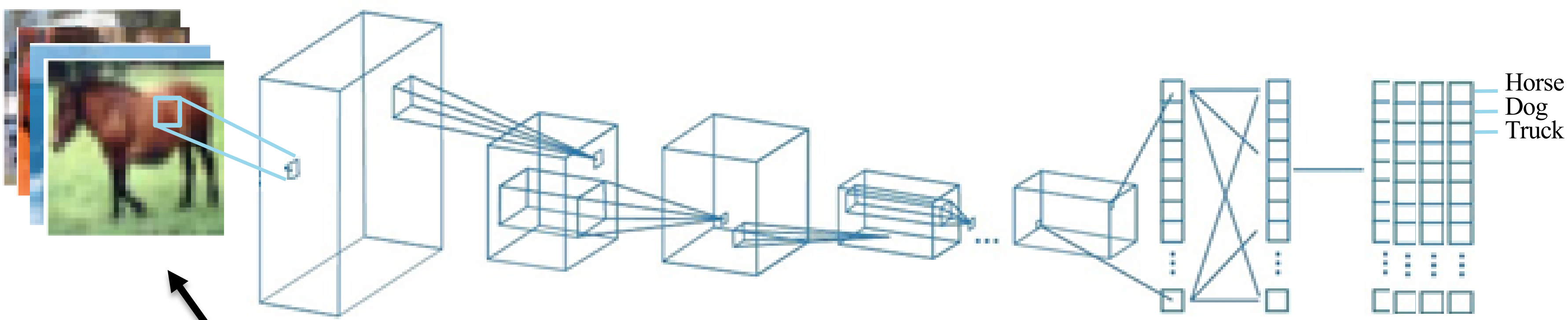


$y_{\vec{w}}(\vec{x}_a) = (0.89, 0.2, 0.125, 0.99)$

Mini-batch processing



Mini-batch processing

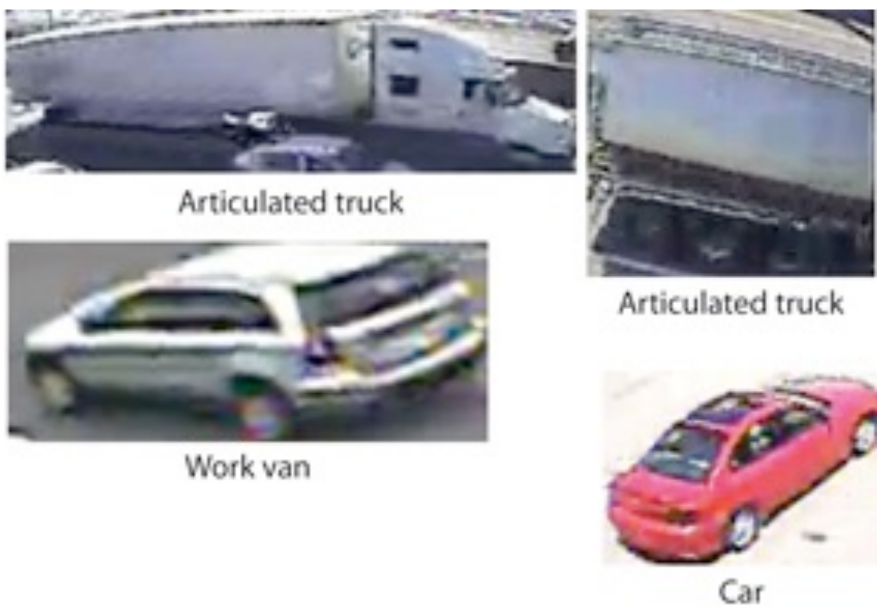


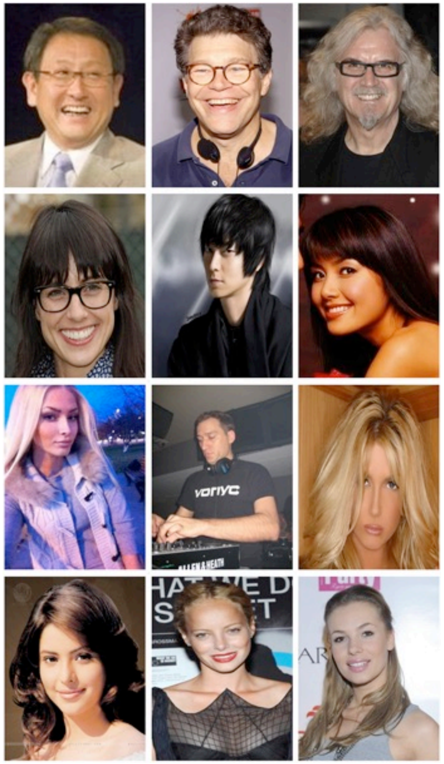
Mini-batch of
4 images

4 predictions

Classical applications of ConvNets

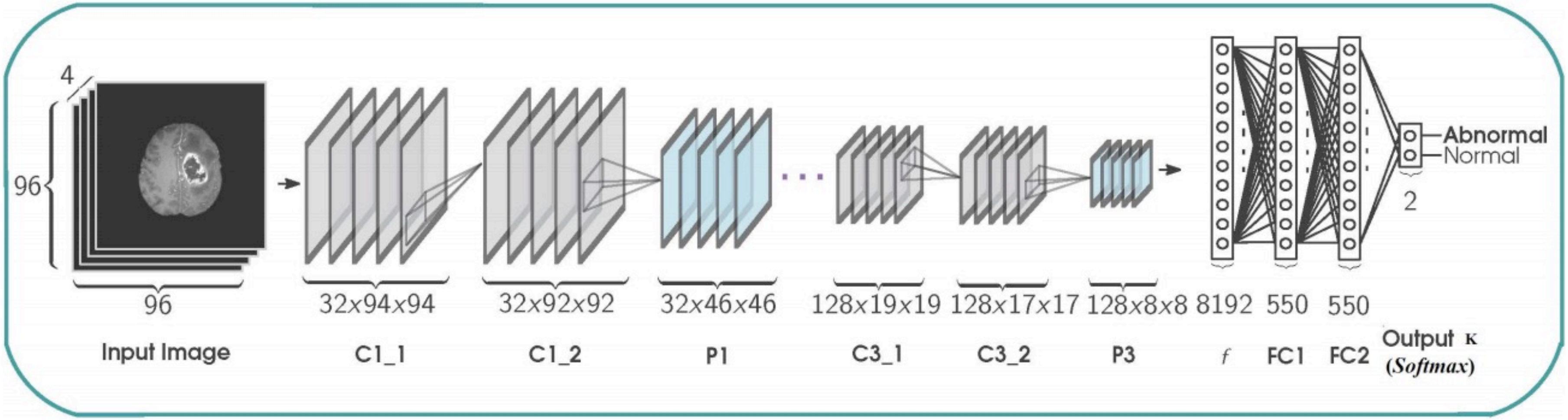
Classification.





Classical applications of ConvNets

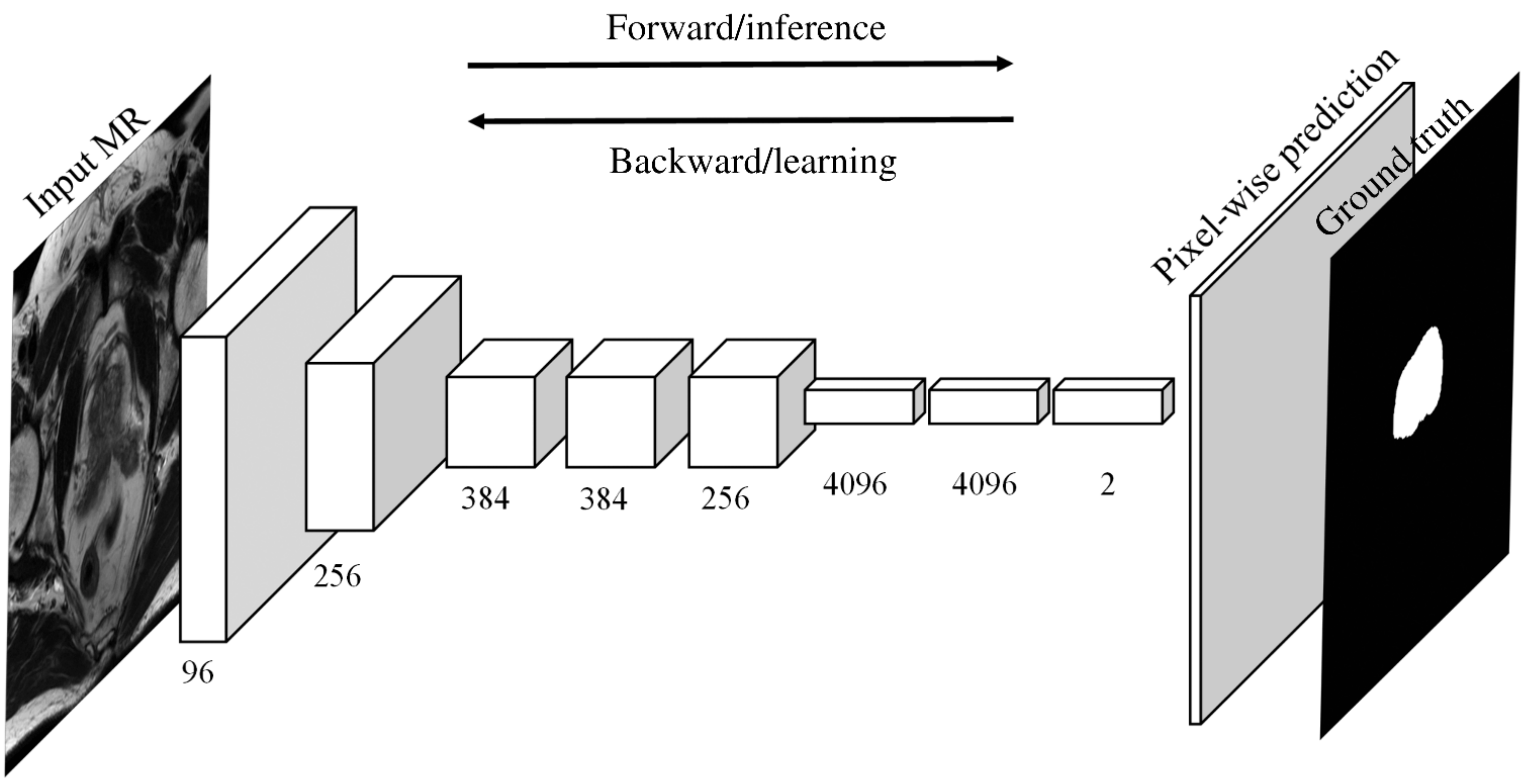
Classification.



S. Banerjee, S. Mitra, A. Sharma, and B. U Shankar, A CADe System for Gliomas in Brain MRI using Convolutional Neural Networks, arXiv:1806.07589v, 2018

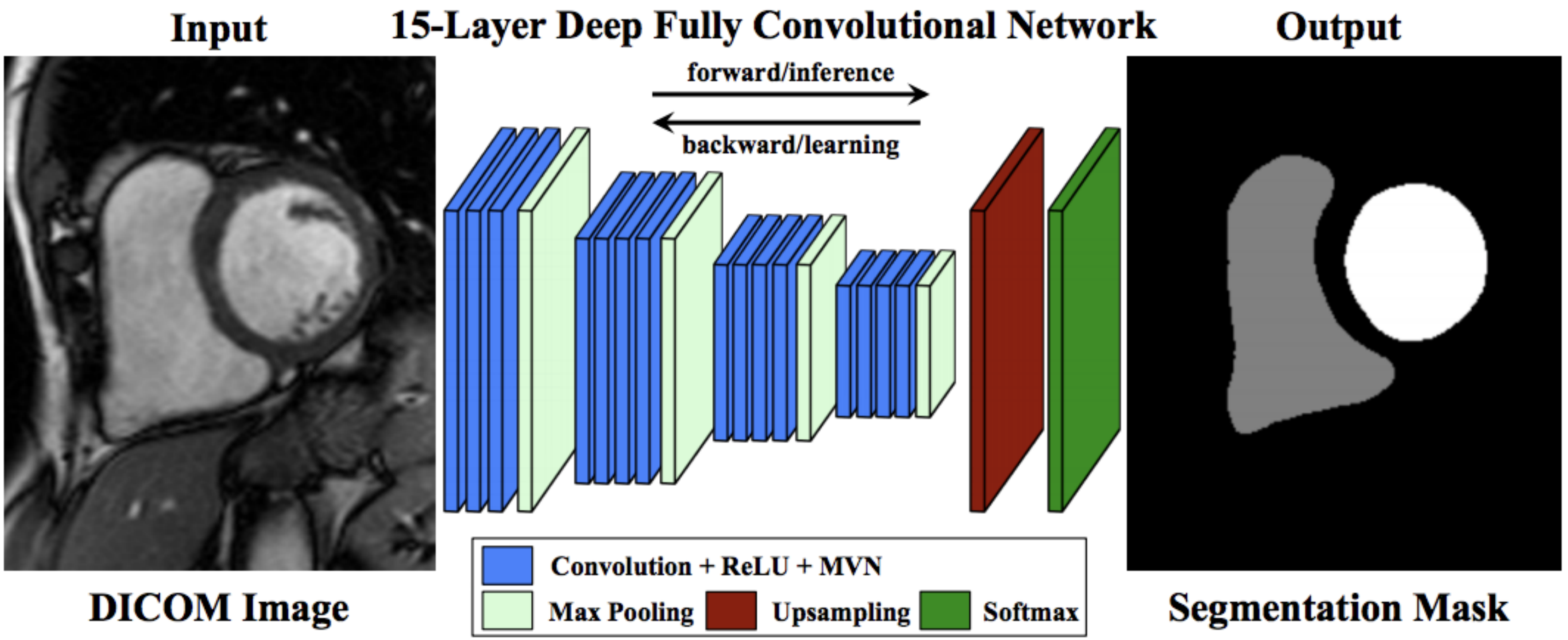
Classical applications of ConvNets

Image segmentation



Classical applications of ConvNets

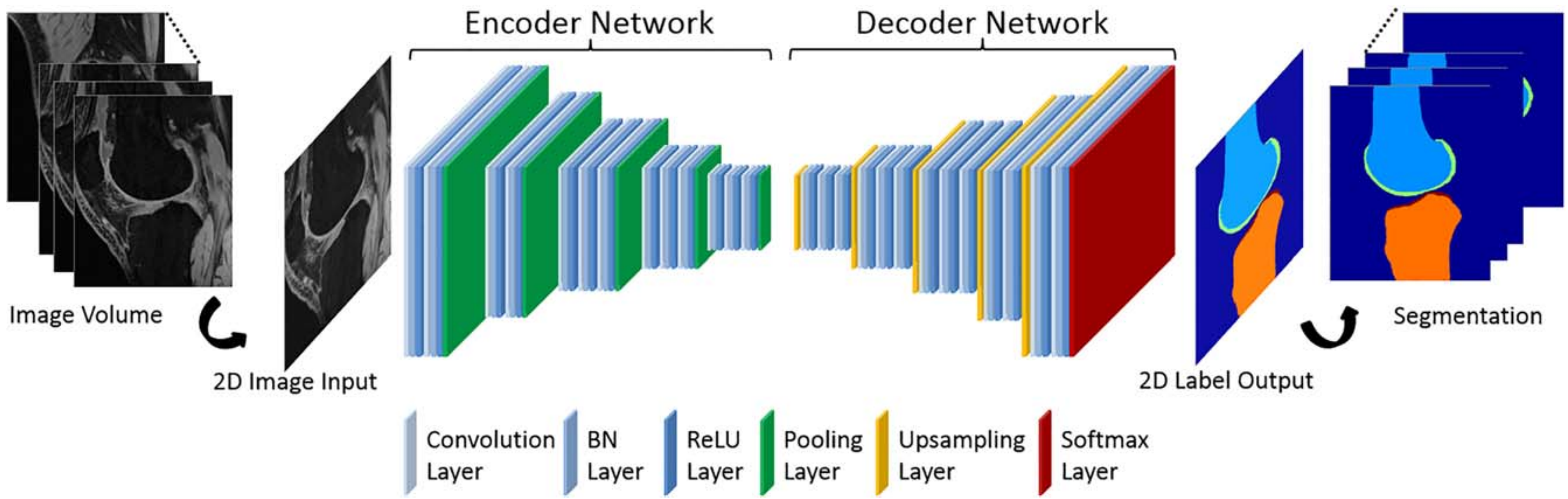
Image segmentation



Tran, P. V., 2016. A fully convolutional neural network for cardiac segmentation in short-axis MRI. arXiv:1604.00494.

Classical applications of ConvNets

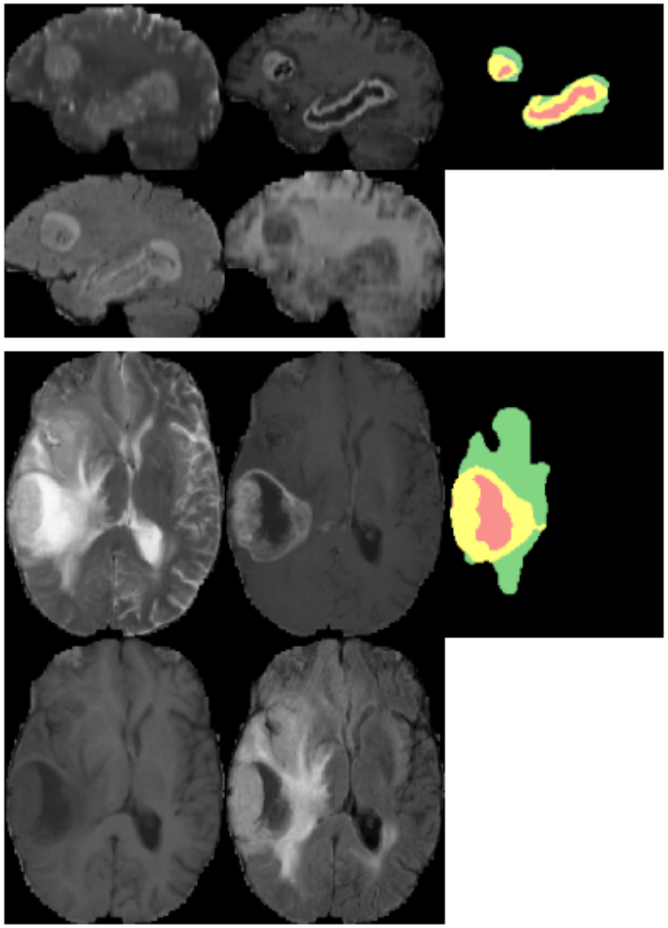
Image segmentation



Fang Liu, Zhaoye Zhou, +3 authors, Deep convolutional neural network and 3D deformable approach for tissue segmentation in musculoskeletal magnetic resonance imaging. in Magnetic resonance in medicine 2018 DOI:10.1002/mrm.26841

Classical applications of ConvNets

Image segmentation



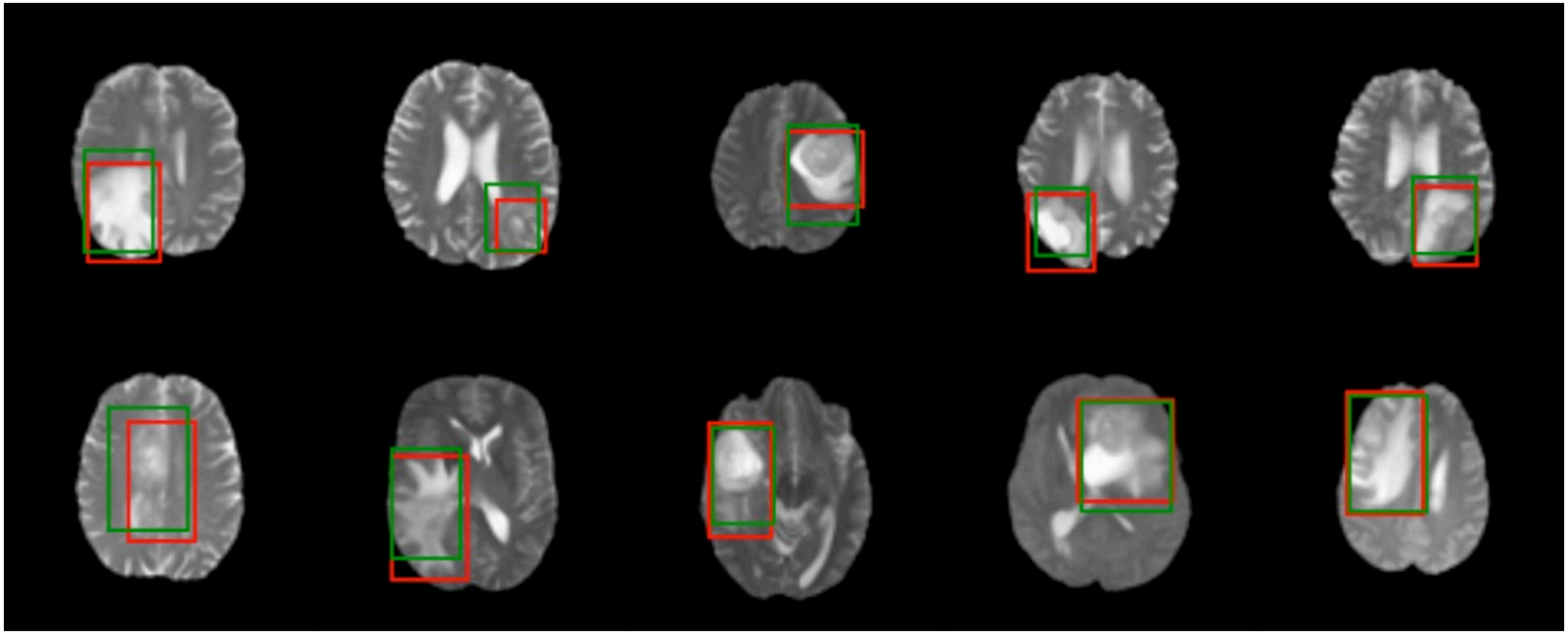
Havaei M., Davy A., Warde-Farley D., Biard A., Courville A., Bengio Y., Pal C., Jodoin P-M, Larochelle H. (2017)
Brain Tumor Segmentation with Deep Neural Networks, Medical Image Analysis, Vol 35, 18-31

Classical applications of ConvNets

Localization

Classical applications of ConvNets

Localization



S. Banerjee, S. Mitra, A. Sharma, and B. U Shankar, A CADe System for Gliomas in Brain MRI using Convolutional Neural Networks, arXiv:1806.07589v, 2018

Merci