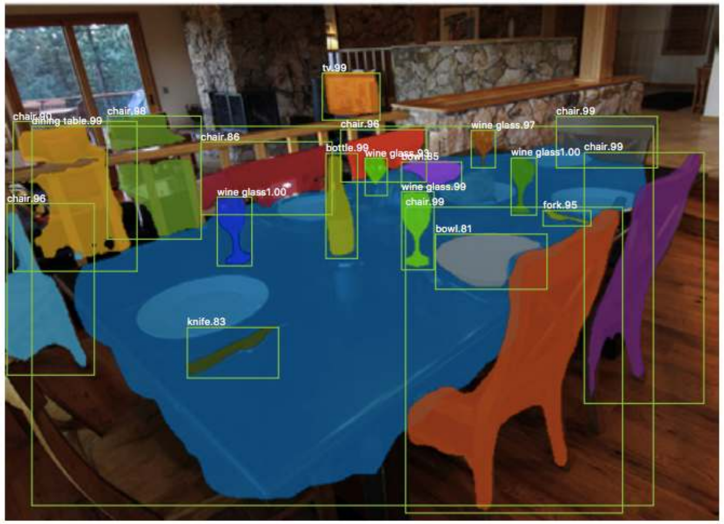
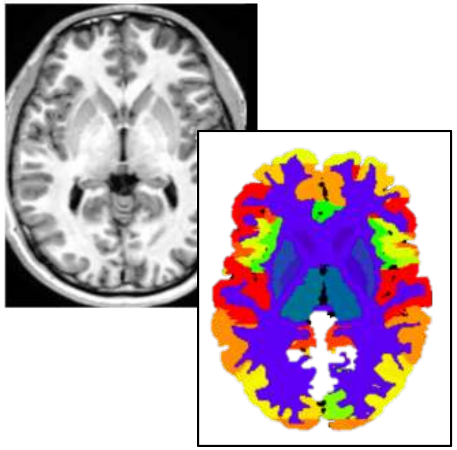
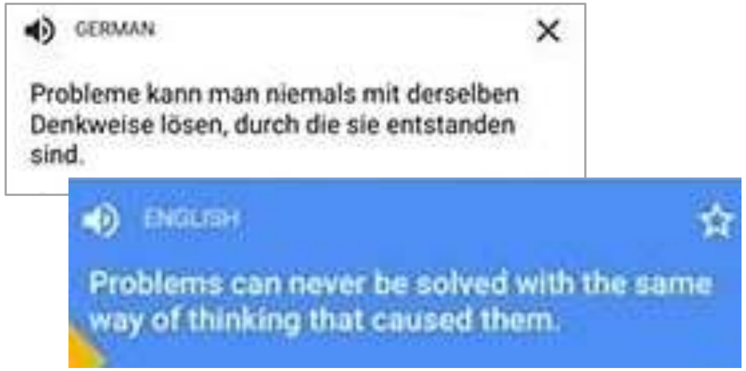
Introduction to Deep Learning

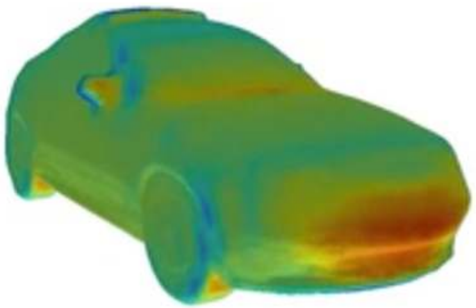
Vincent Lepetit

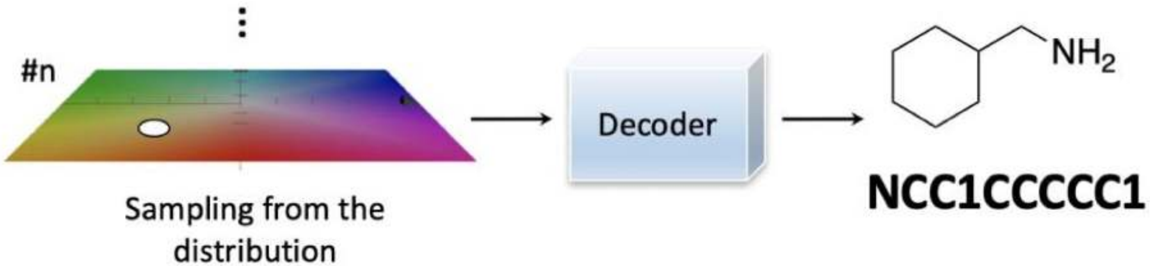






Google Assistant





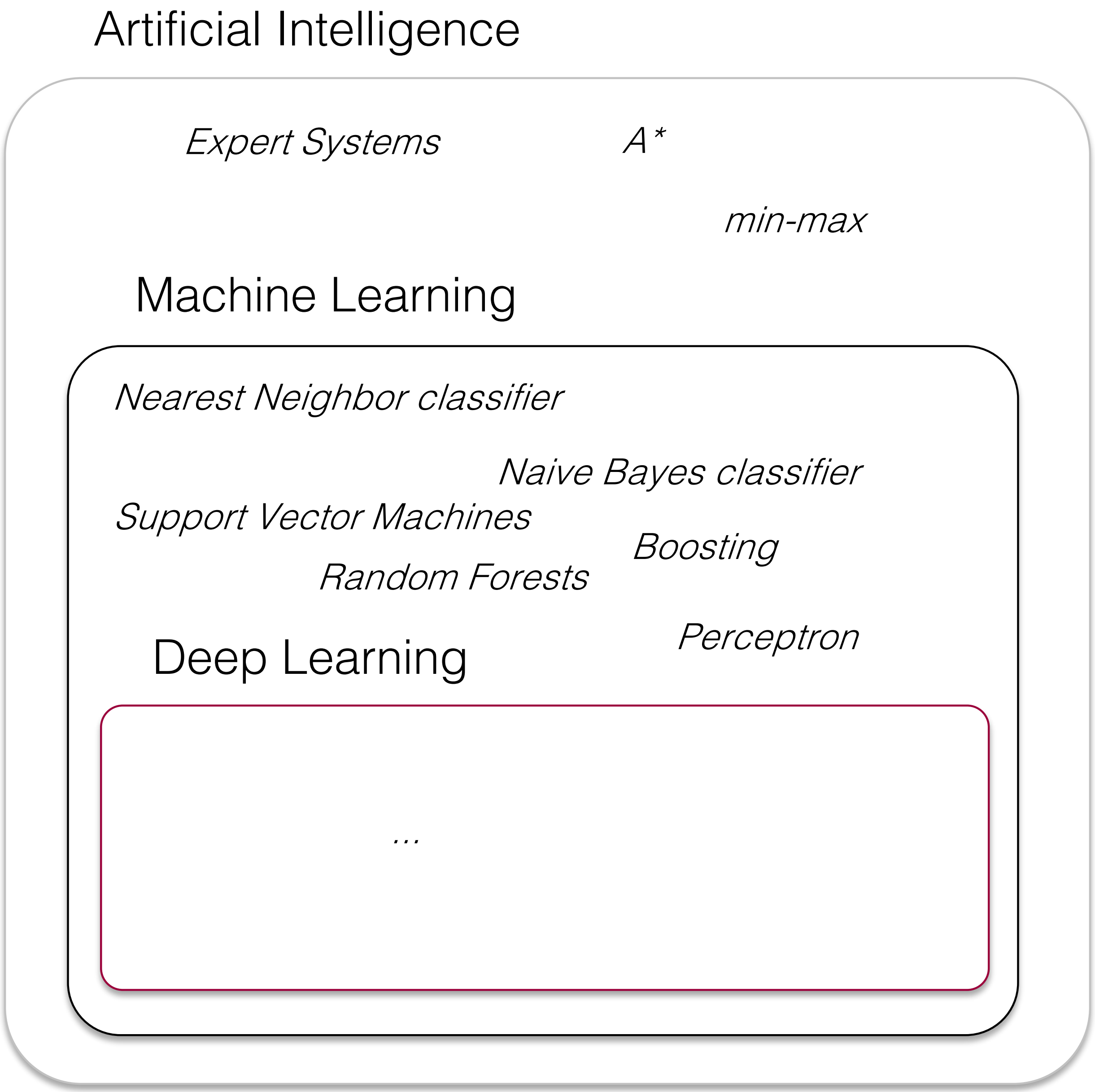
Why Deep Learning is Currently so Popular?

- No need to engineer features;

- Very flexible framework. Originally developed for supervised learning, but can be extended to many other problems.

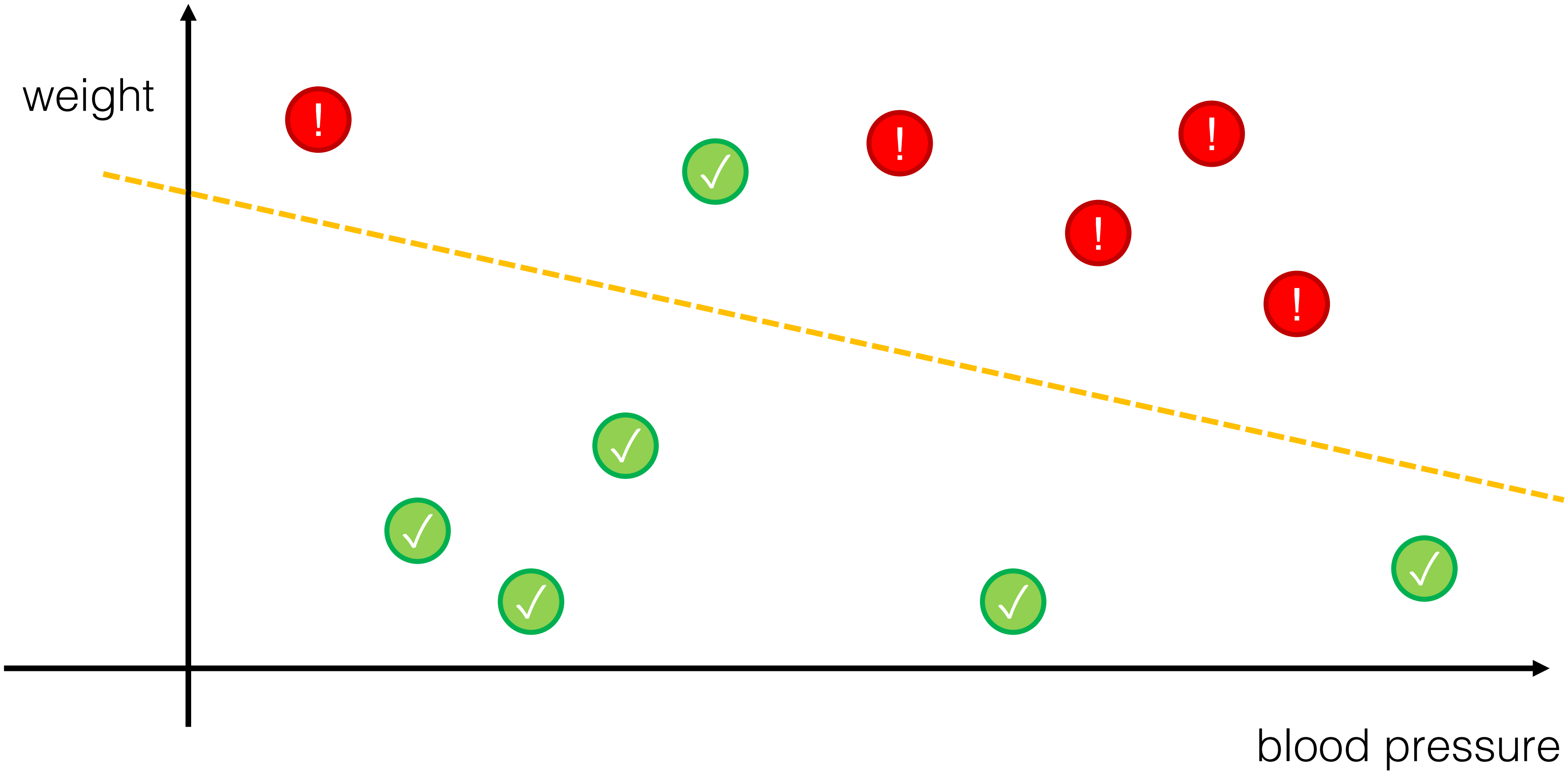
- *Why now?*
    - Faster computers (with GPUs); More training data; Better optimization algorithms; Easy to use and powerful libraries in Python; It took time to researchers to get convinced it actually works.

Artificial Intelligence/Machine Learning/Deep Learning



From Early Approaches to Modern Deep Learning

Linear Classifier / Perceptron



formalization

The equation of the separation (boundary) is

w_1 x (blood pressure) + w_2 x weight + $b = 0$

Samples such that

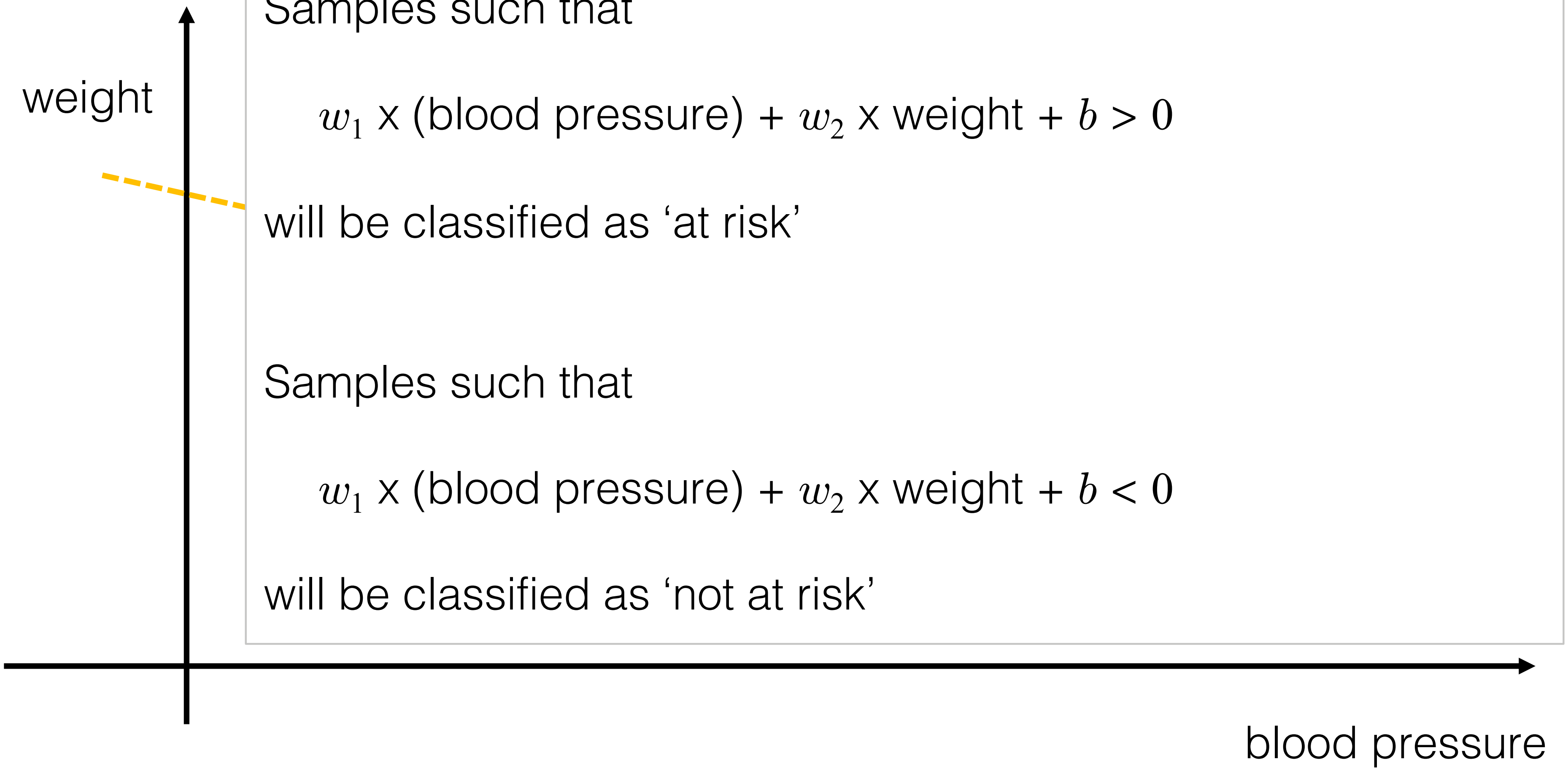
w_1 x (blood pressure) + w_2 x weight + $b > 0$

will be classified as 'at risk'

Samples such that

w_1 x (blood pressure) + w_2 x weight + $b < 0$

will be classified as 'not at risk'



formalization (2)

The equation of the separation (boundary) is

$$w_1 \times x_1 + w_2 \times x_2 + b = 0$$

Samples such that

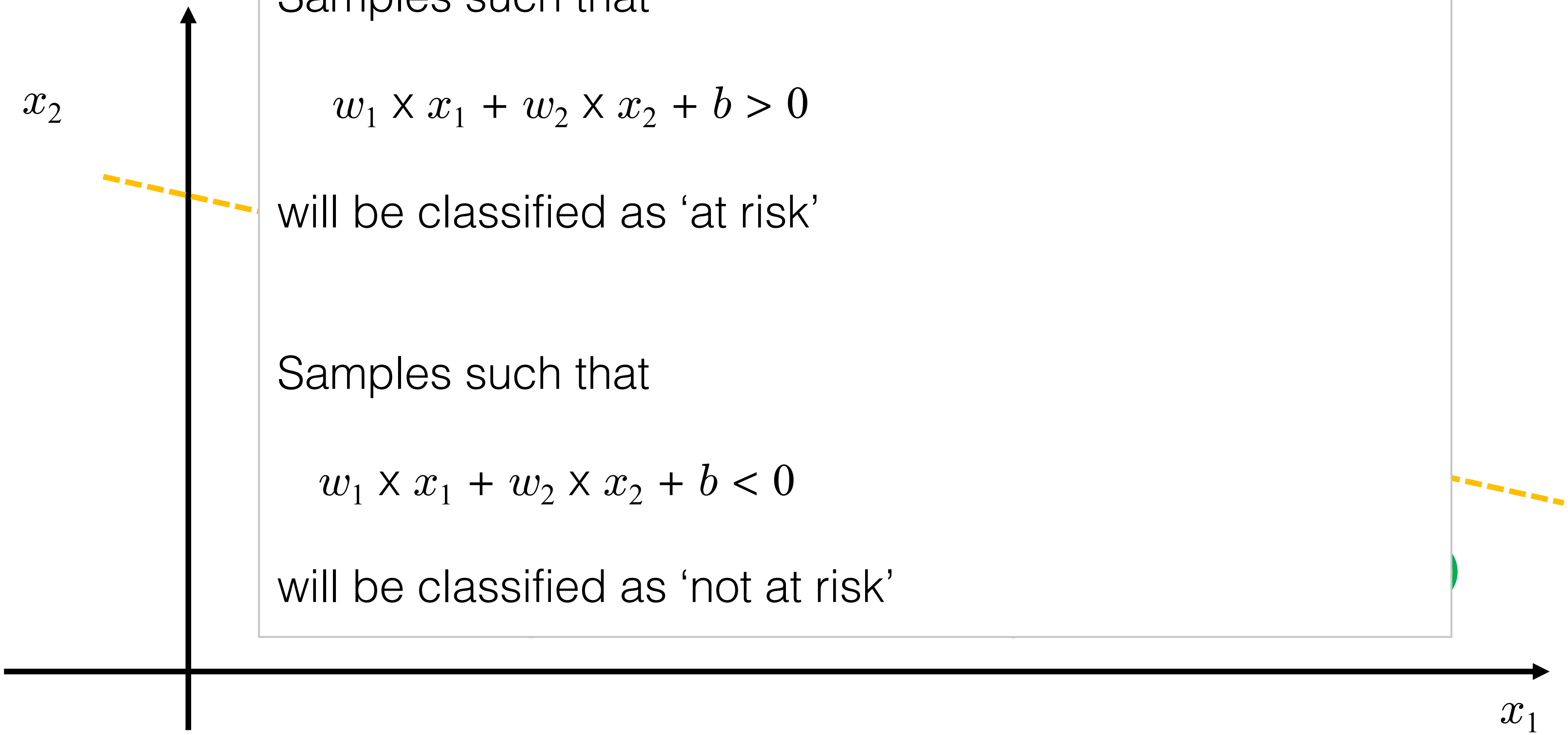
$$w_1 \times x_1 + w_2 \times x_2 + b > 0$$

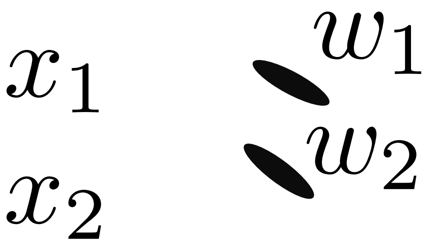
will be classified as ‘at risk’

Samples such that

$$w_1 \times x_1 + w_2 \times x_2 + b < 0$$

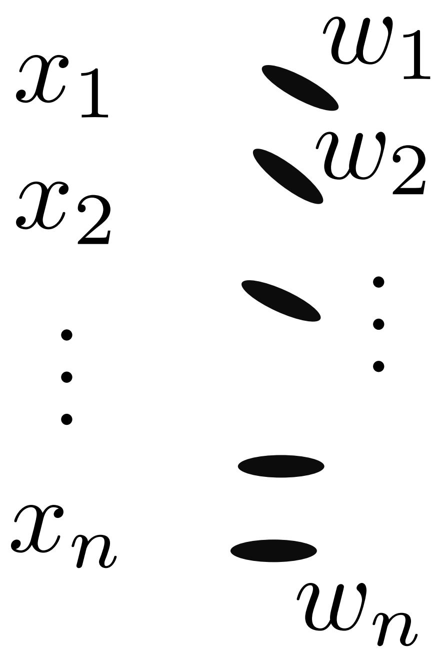
will be classified as ‘not at risk’





$$o = w_1 x_1 + w_2 x_2$$

Prediction based on the sign of o

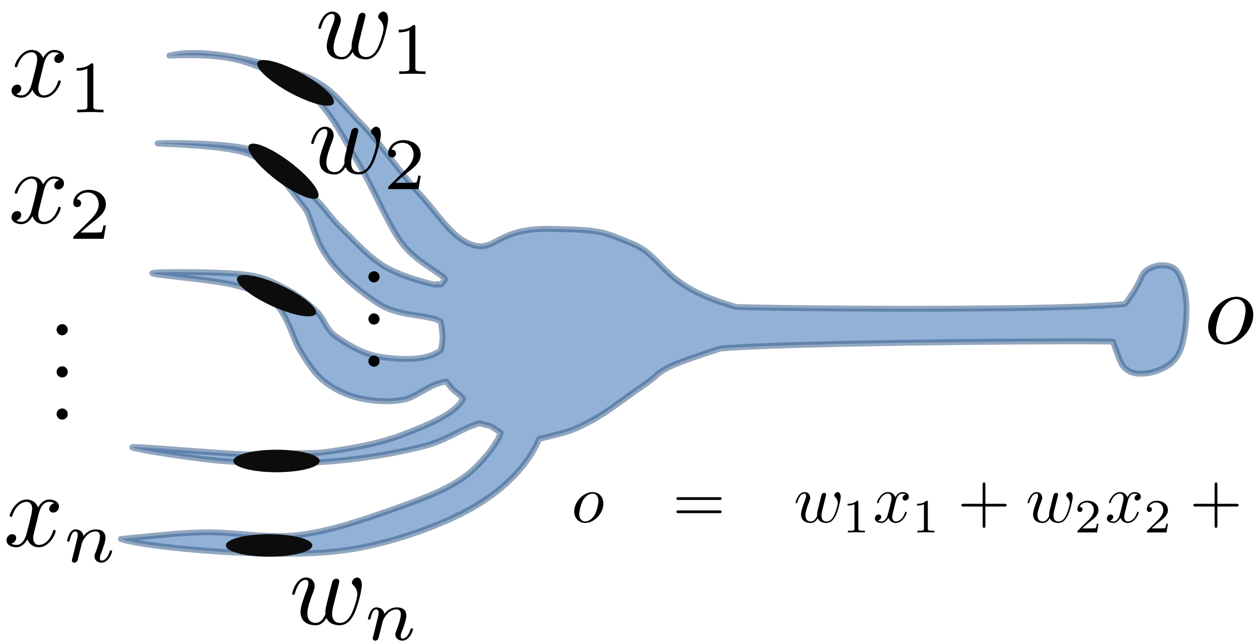


$$o = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

Prediction based on the sign of o

PERCEPTRON

Inspired by the work of Donald Hebb (1949)



$$o = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

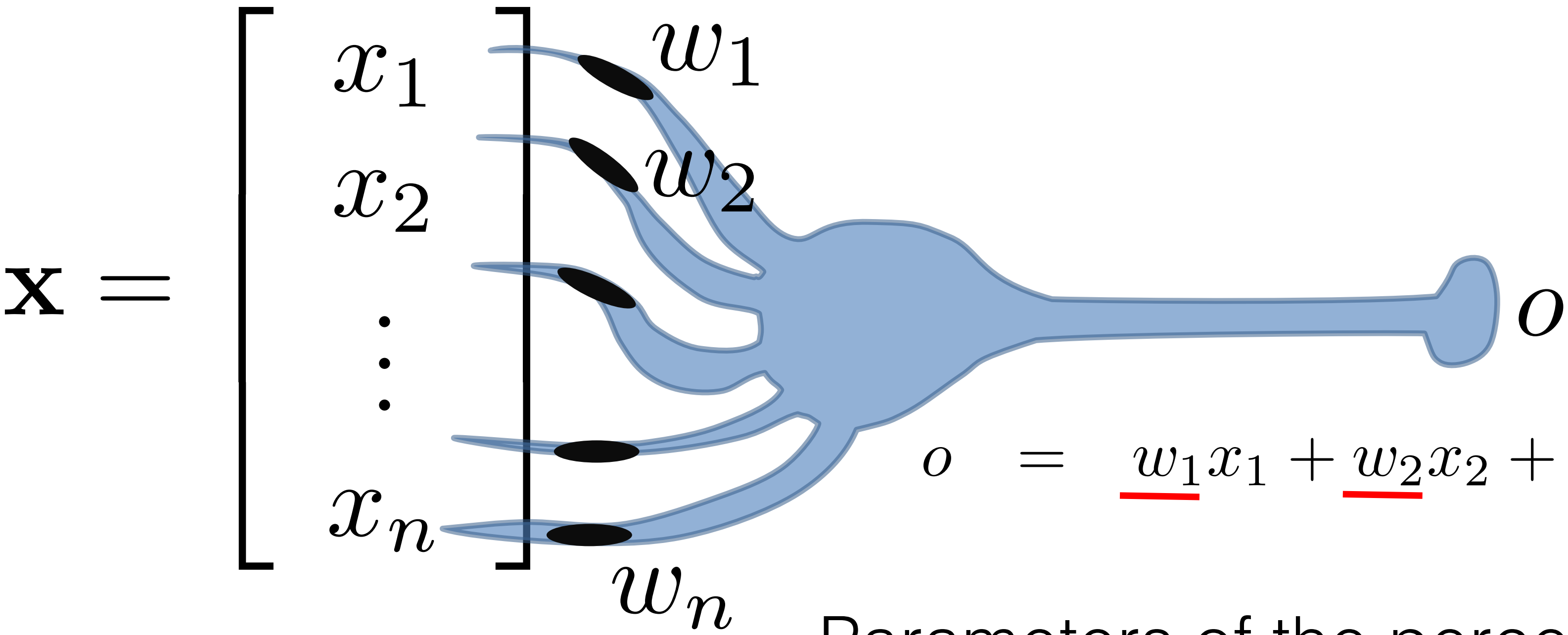
Prediction based on the sign of o

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\begin{aligned} o &= w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b \\ &= \textstyle\sum_i w_i x_i + b \\ &= \mathbf{w}^\top \mathbf{x} + b \end{aligned}$$

Prediction based on the sign of o

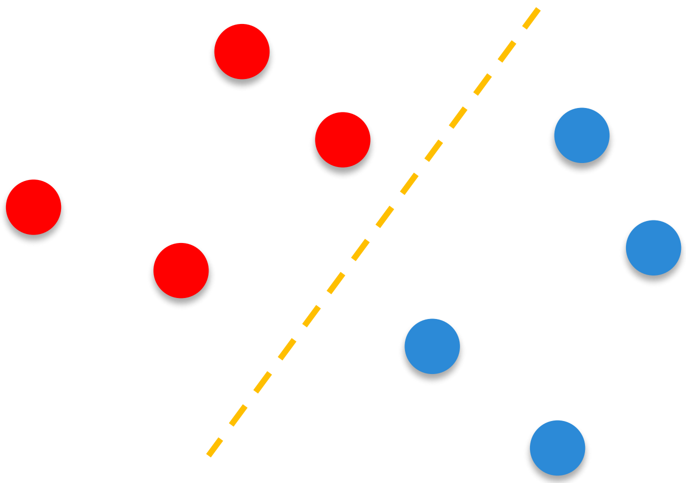
Perceptron



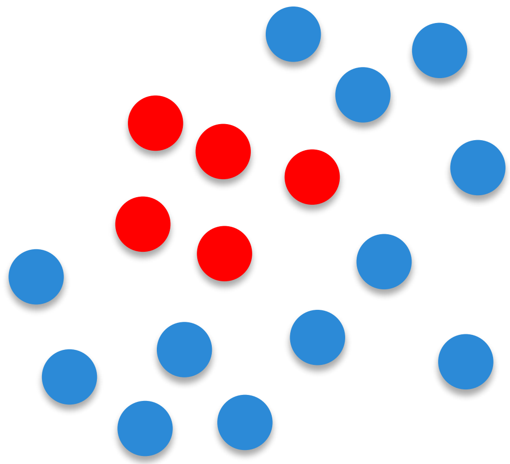
$$o = \underline{w_1} x_1 + \underline{w_2} x_2 + \cdots + \underline{w_n} x_n + \underline{b}$$

Parameters of the perceptron:
we need to find good values for them
(we will see that later)

A perceptron can only correctly classify data points that are linearly separable:

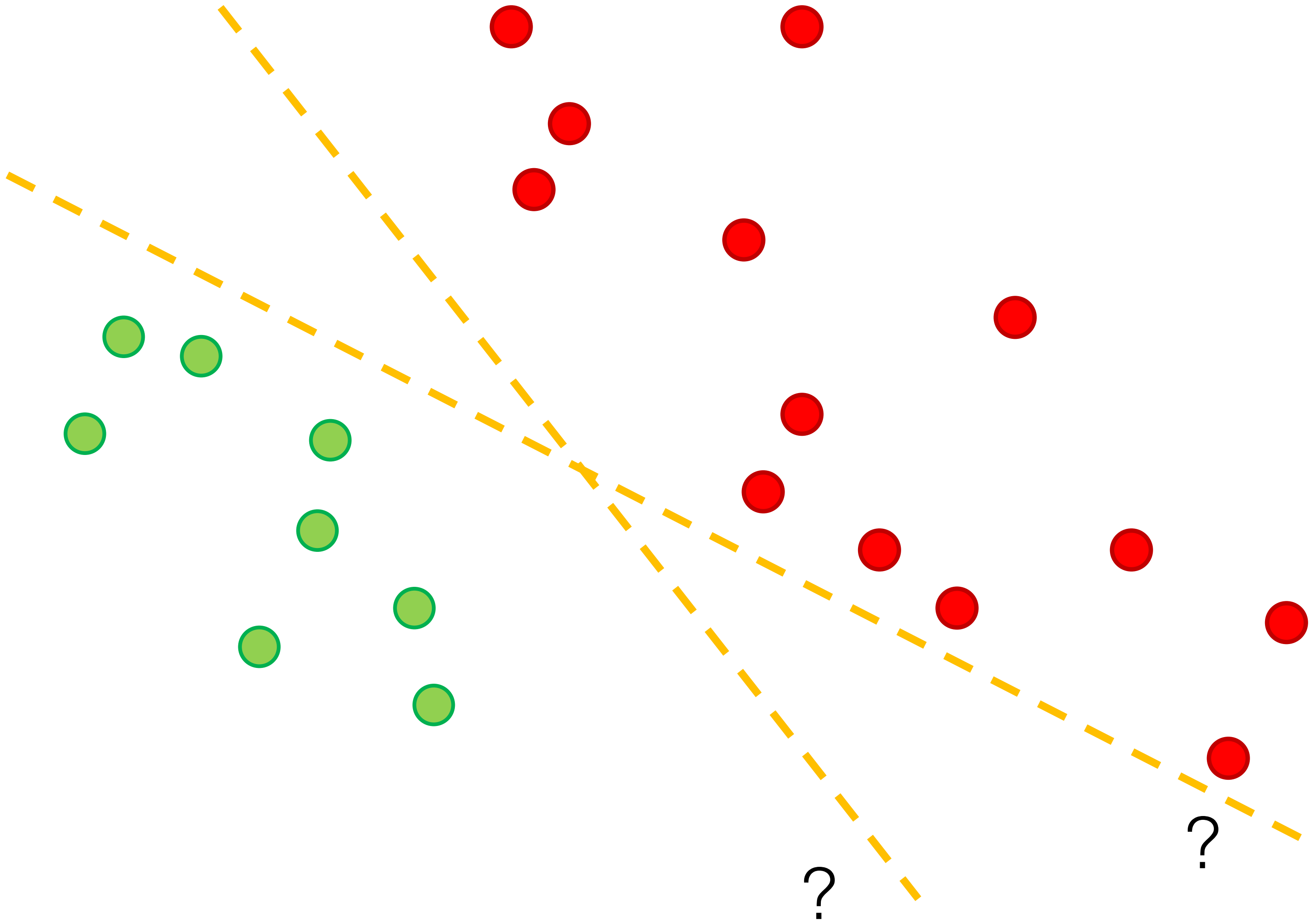


linearly separable

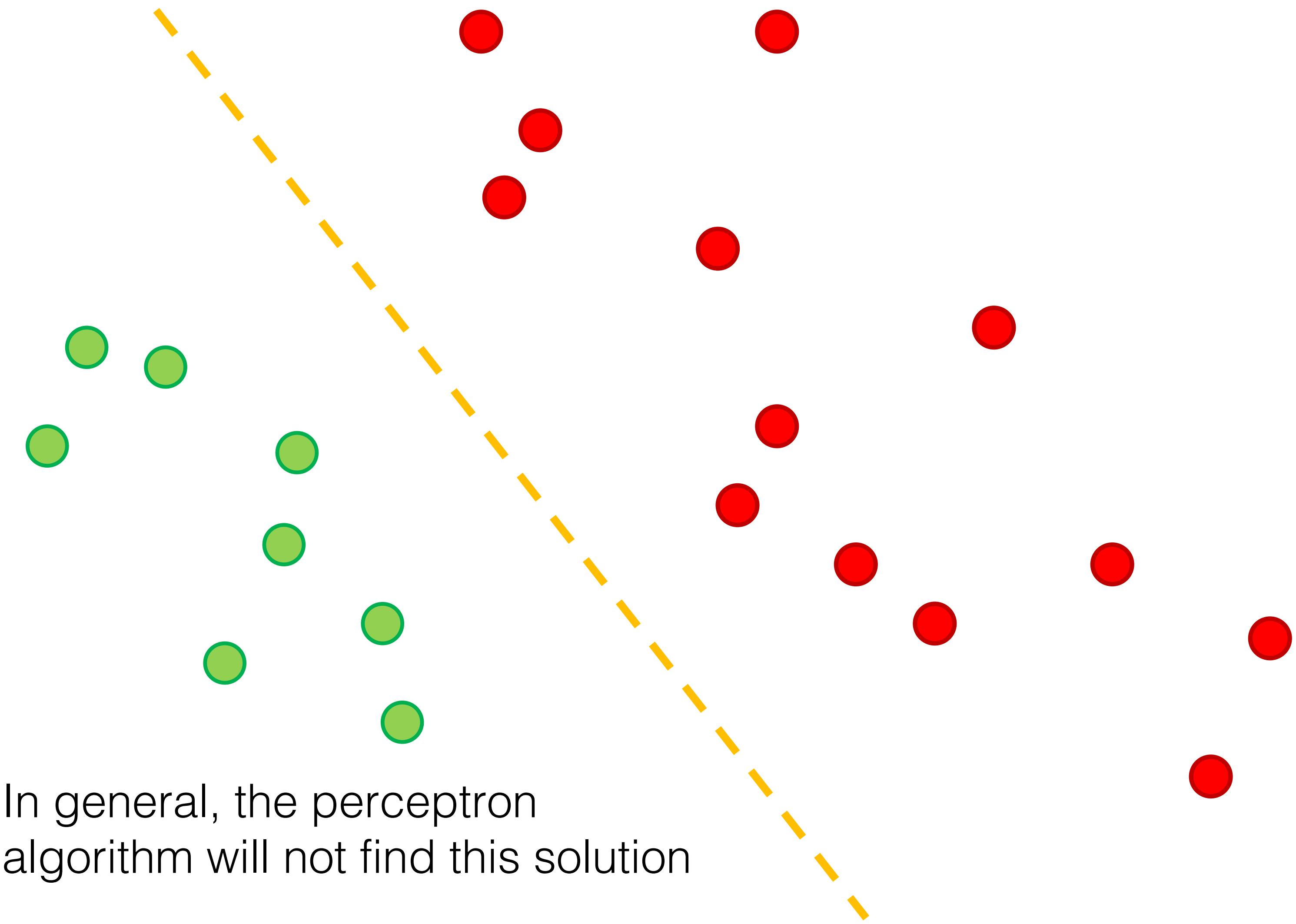


nonlinearly separable

What is the Best Linear Classifier?



What is the Best Linear Classifier?

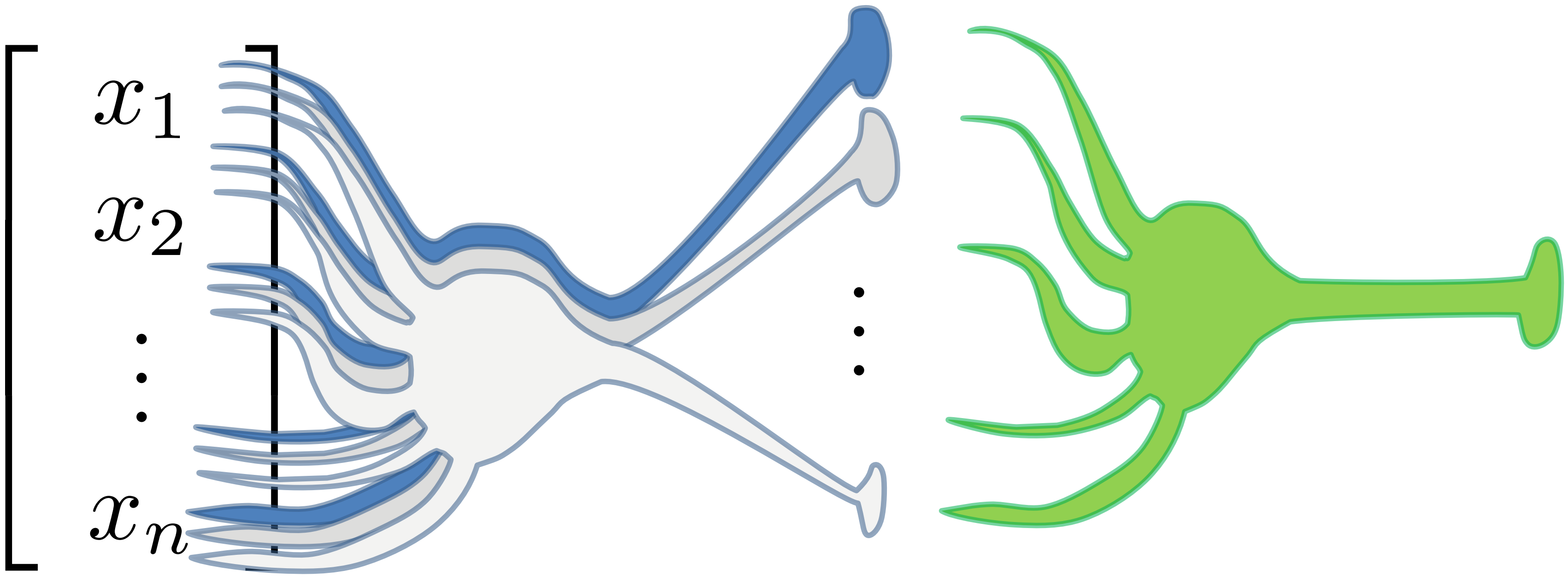


In general, the perceptron algorithm will not find this solution

TWO-LAYER NETWORKS,

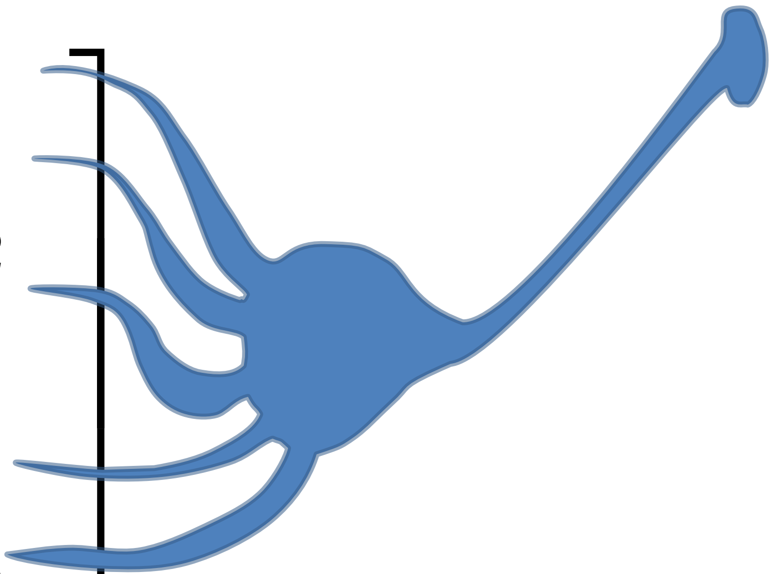
MULTI-LAYER PERCEPTRONS (MLP)

Two-Layer Network (~1980)



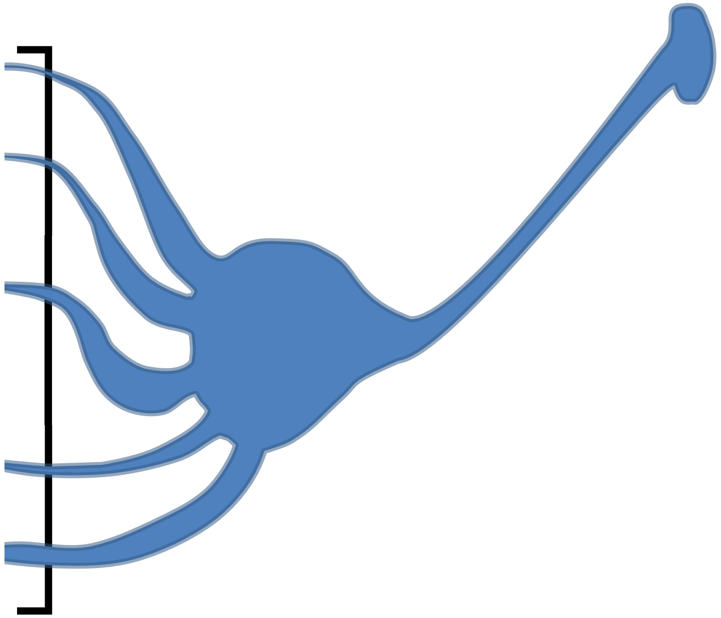
Two-Layer Network (~1980)

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$



$$o_1 = w_{1,1}x_1 + w_{1,2}x_2 + \ldots + w_{1,n}$$

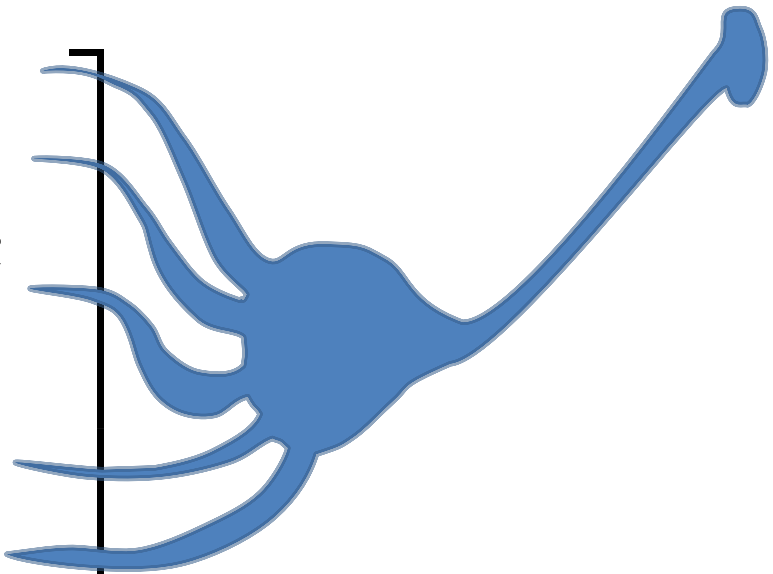
Two-Layer Network (~1980)



$$o_1 = w_{1,1}x_1 + w_{1,2}x_2 + \ldots + w_{1,n}x_n + b_1$$

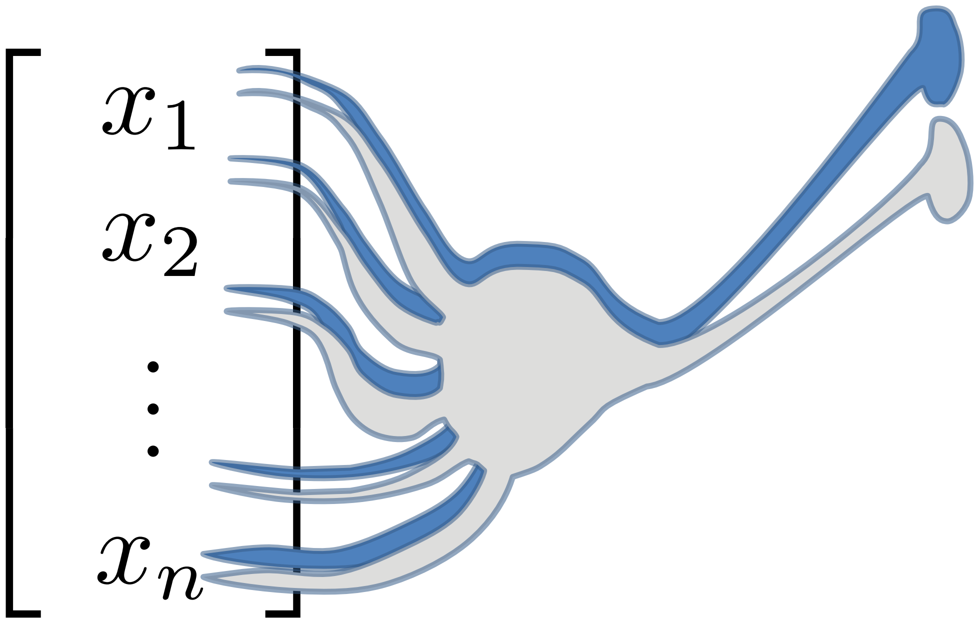
Two-Layer Network (~1980)

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$



$$o_1 = w_{1,1}x_1 + w_{1,2}x_2 + \ldots + w_{1,n}$$

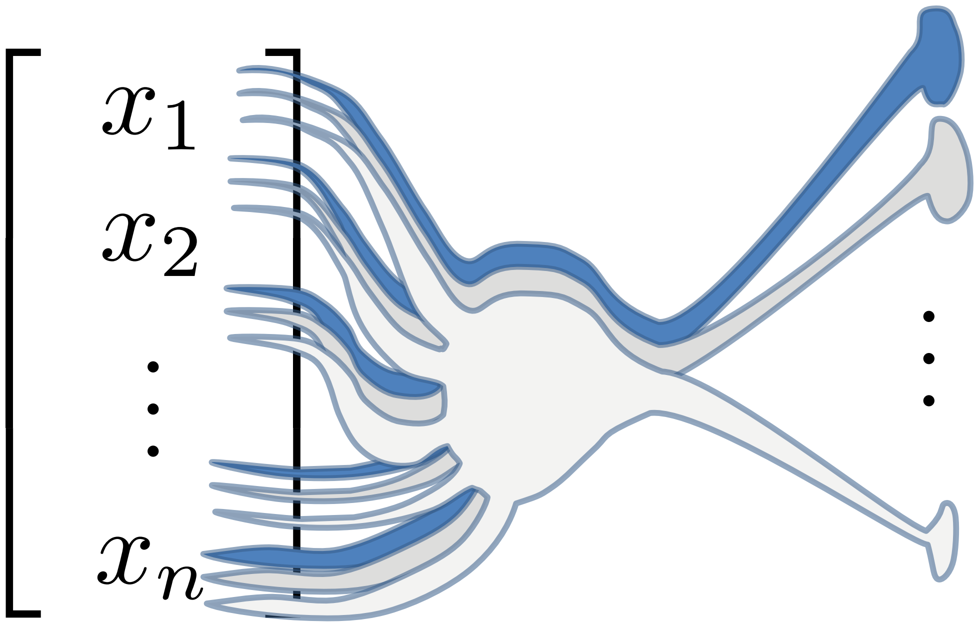
Two-Layer Network (~1980)



$$o_1 = w_{1,1}x_1 + w_{1,2}x_2 + \ldots + w_{1,n}$$

$$o_2 = w_{2,1}x_1 + w_{2,2}x_2 + \ldots + w_{2,n}$$

Two-Layer Network (~1980)



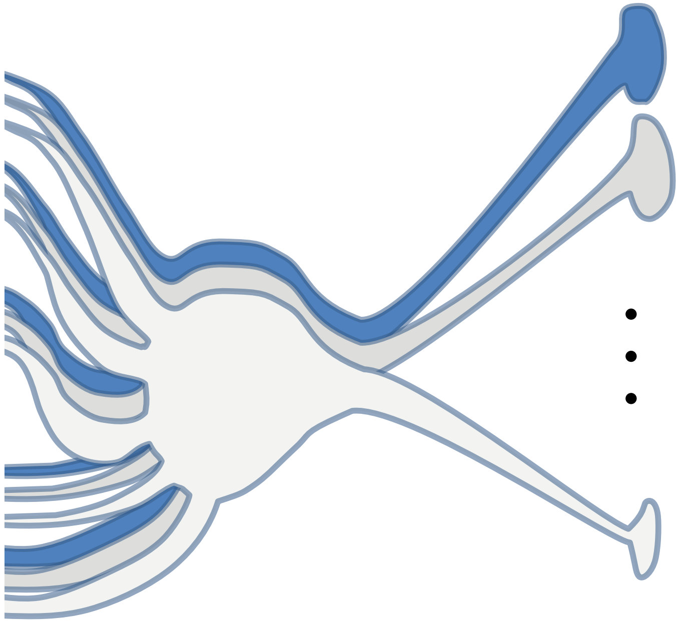
$$o_1 = w_{1,1}x_1 + w_{1,2}x_2 + \ldots + w_{1,n}$$

$$o_2 = w_{2,1}x_1 + w_{2,2}x_2 + \ldots + w_{2,n}x$$

$$\vdots$$

$$o_m = w_{m,1}x_1 + w_{m,2}x_2 + \ldots + w_m$$

the first network's parameters



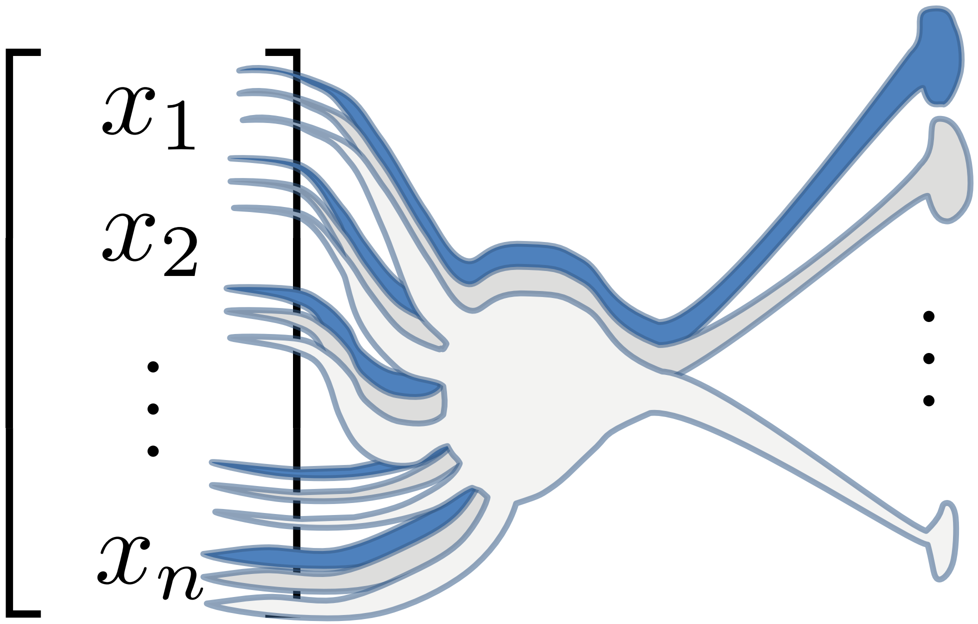
$$o_1 = \underline{w_{1,1}}x_1 + \underline{w_{1,2}}x_2 + \ldots + \underline{w_{1,n}}x_n + \underline{b_1}$$

$$o_2 = \underline{w_{2,1}}x_1 + \underline{w_{2,2}}x_2 + \ldots + \underline{w_{2,n}}x_n + \underline{b_2}$$

$$\vdots$$

$$o_m = \underline{w_{m,1}}x_1 + \underline{w_{m,2}}x_2 + \ldots + \underline{w_{m,n}}x_n + \underline{b_m}$$

Two-Layer Network (~1980)



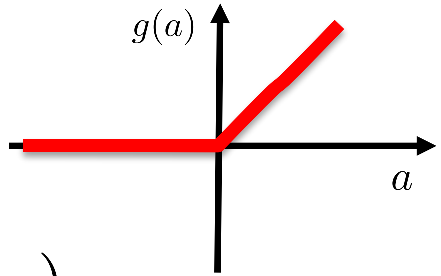
$$o_1 = w_{1,1}x_1 + w_{1,2}x_2 + \ldots + w_{1,n}$$

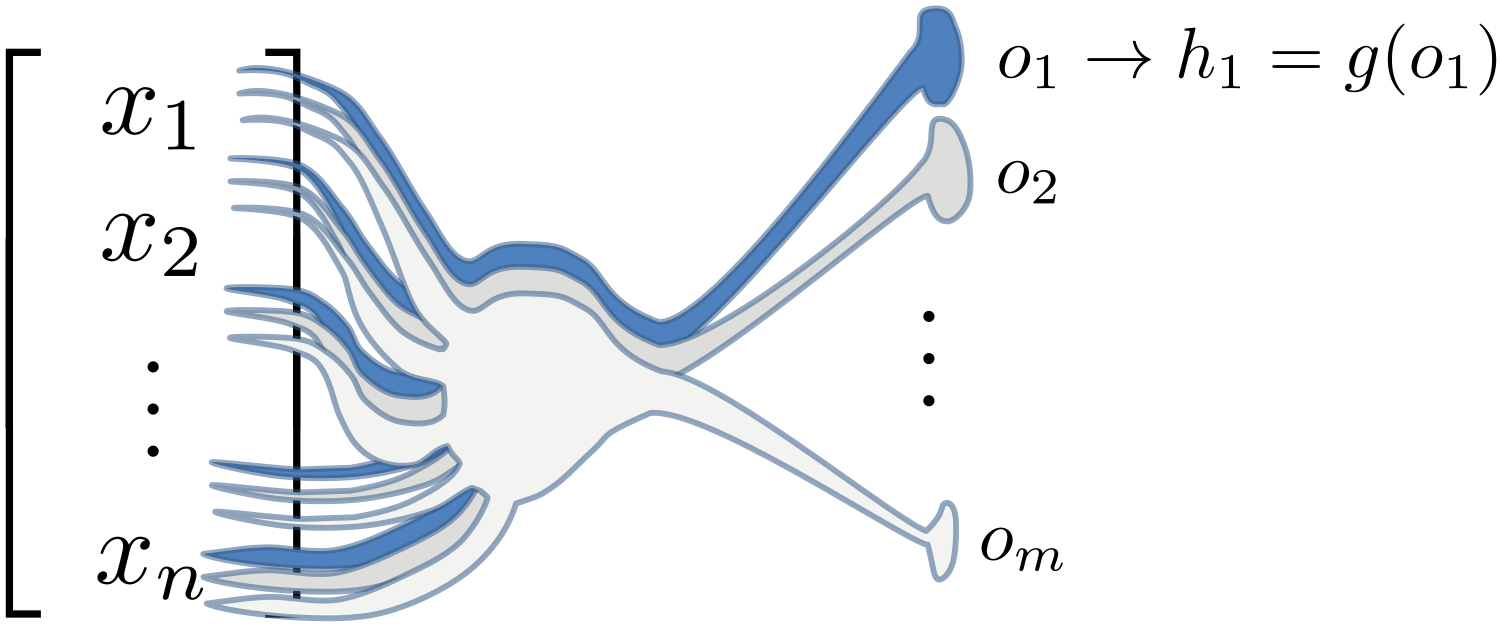
$$o_2 = w_{2,1}x_1 + w_{2,2}x_2 + \ldots + w_{2,n}$$

$$\vdots$$

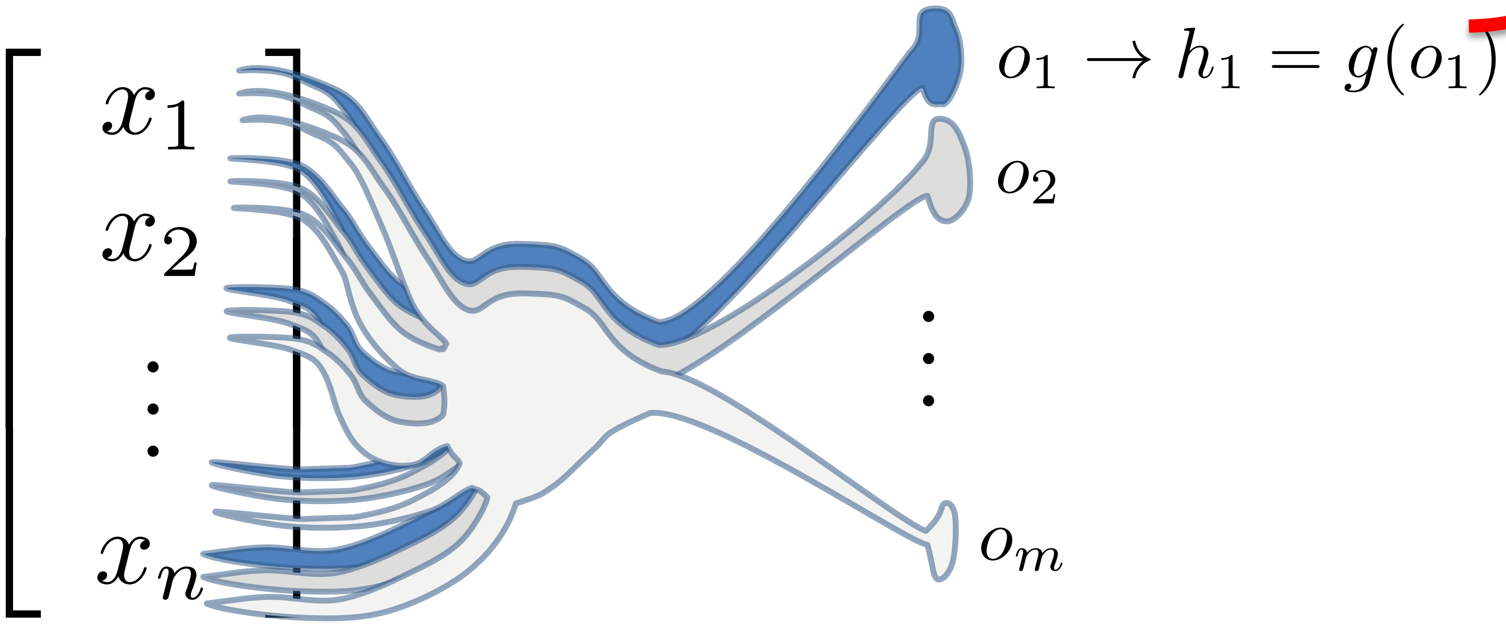
$$o_m = w_{m,1}x_1 + w_{m,2}x_2 + \ldots + w_m$$

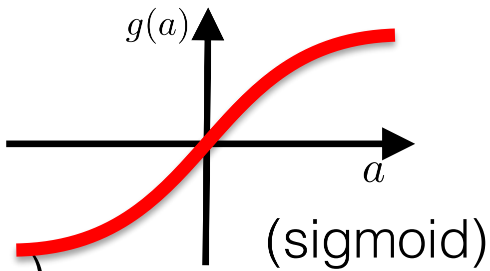
Two-Layer Network (~1980)



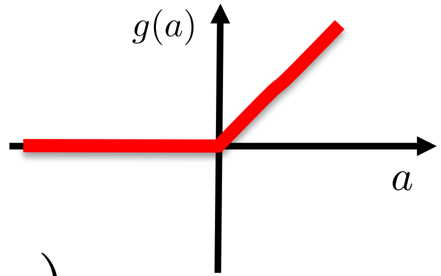


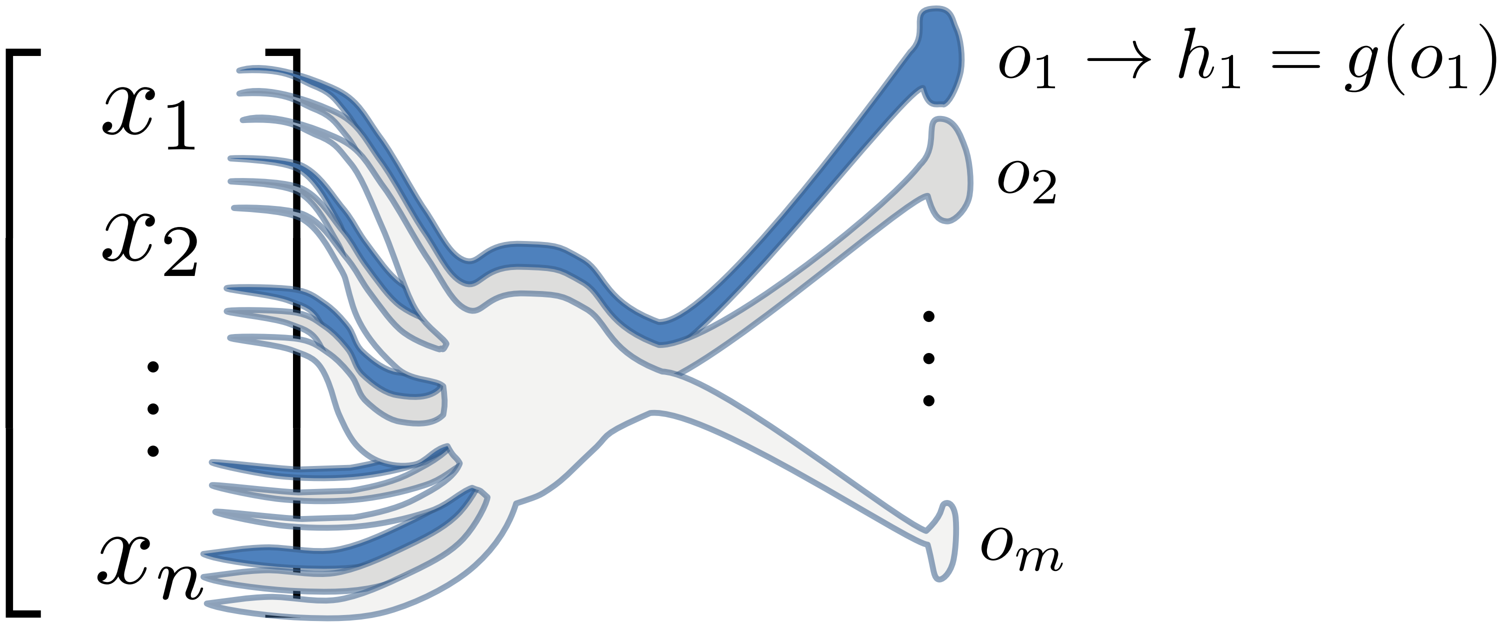
Two-Layer Network (~1980)



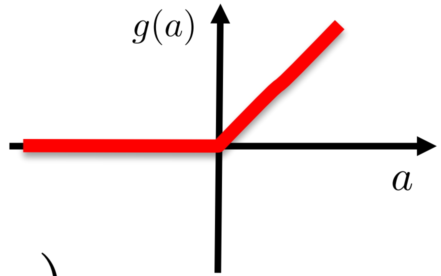


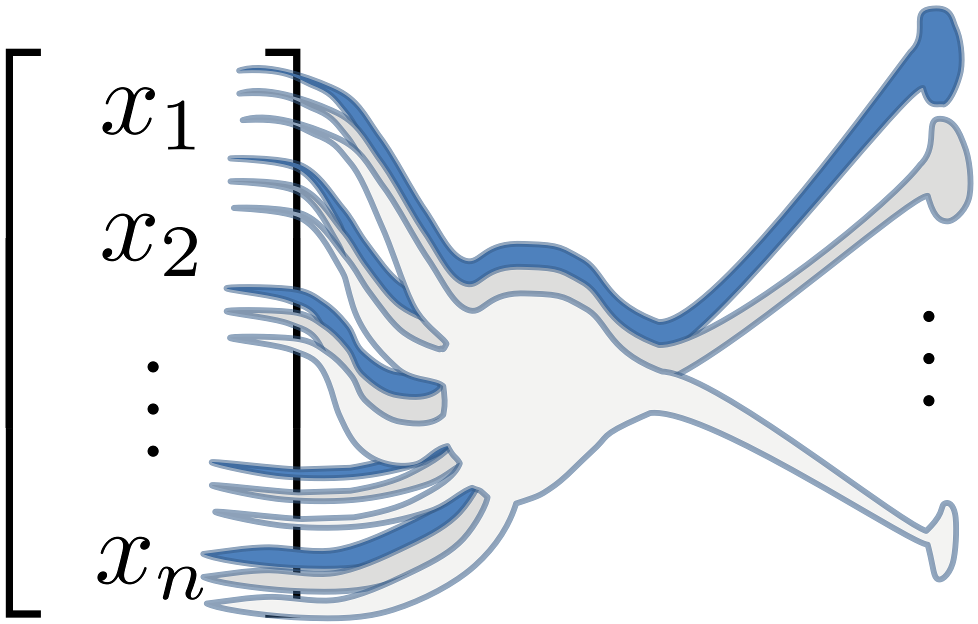
Two-Layer Network (~1980)





Two-Layer Network (~1980)





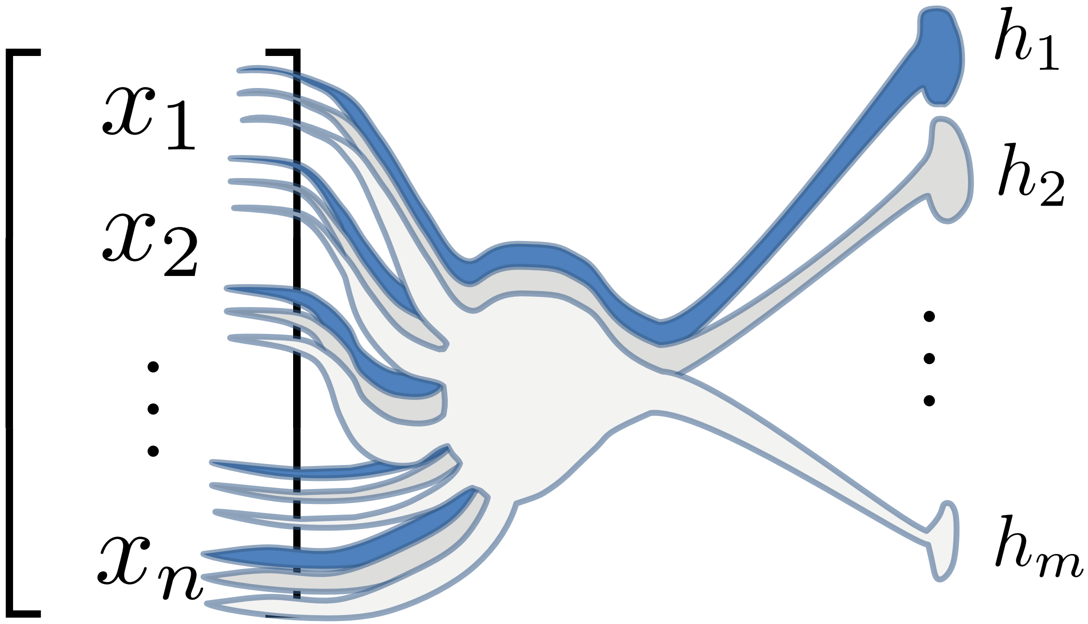
$$o_1 \rightarrow h_1 = g(o_1)$$

$$o_2 \rightarrow h_2 = g(o_2)$$

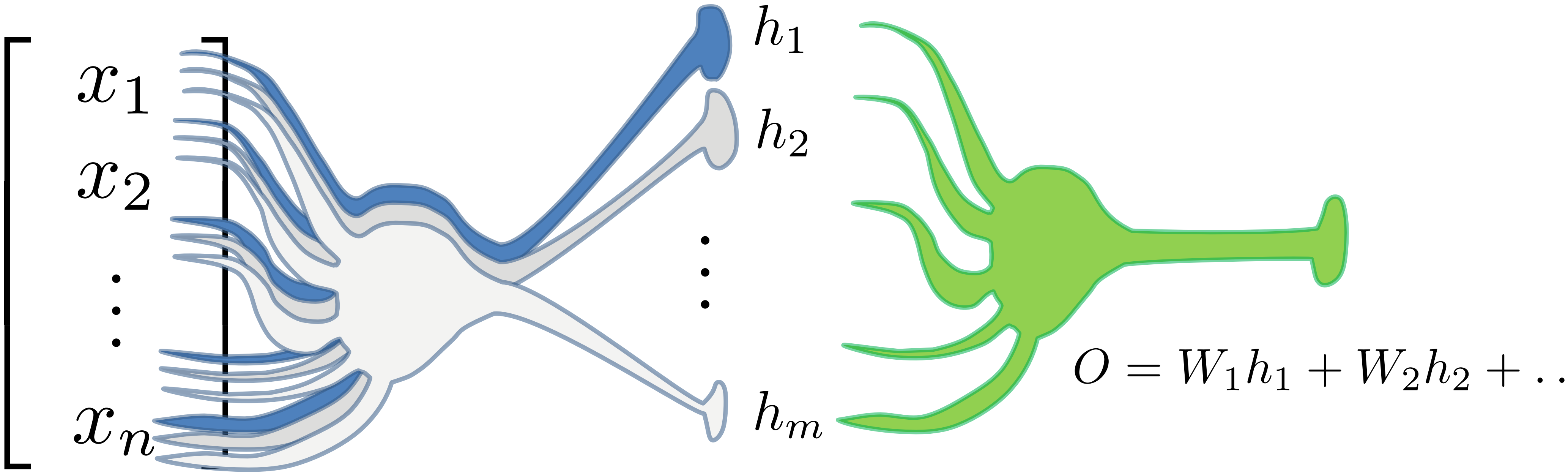
$$\vdots$$

$$o_m \rightarrow h_m = g(o_m)$$

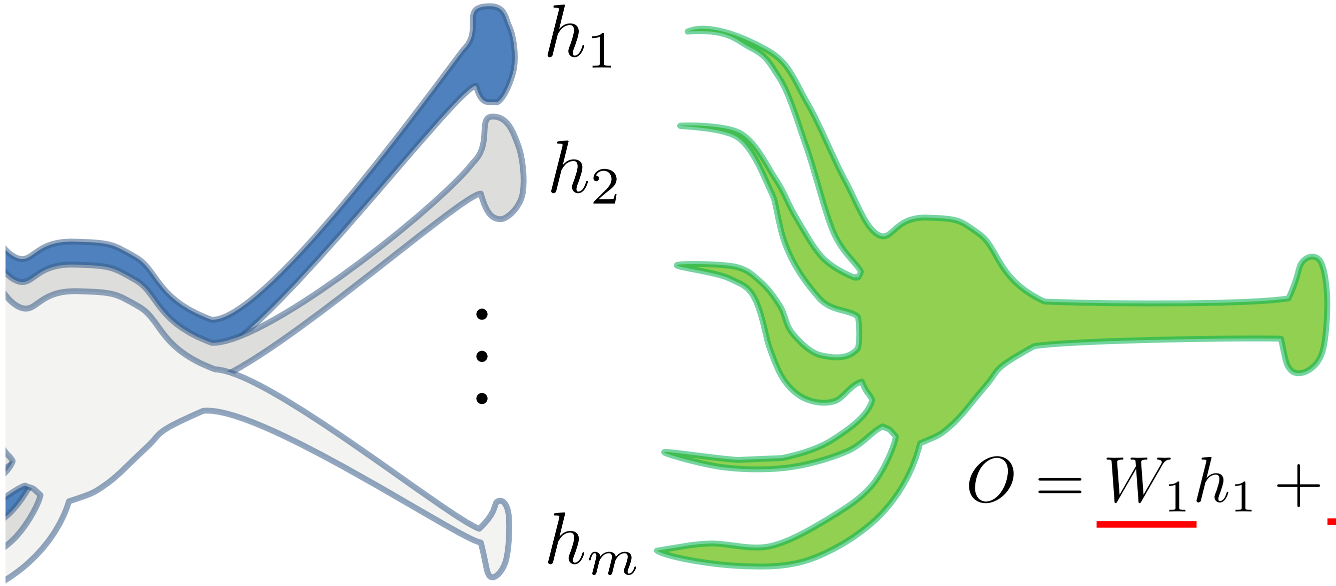
Two-Layer Network (~1980)



Two-Layer Network (~1980)

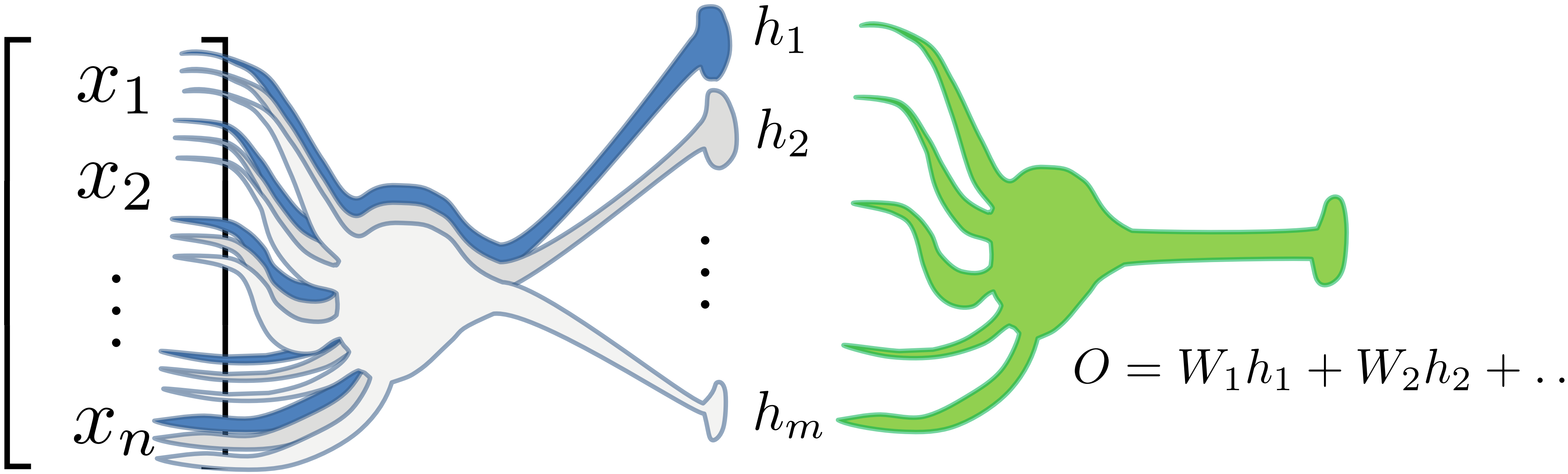


the rest of the network's parameters

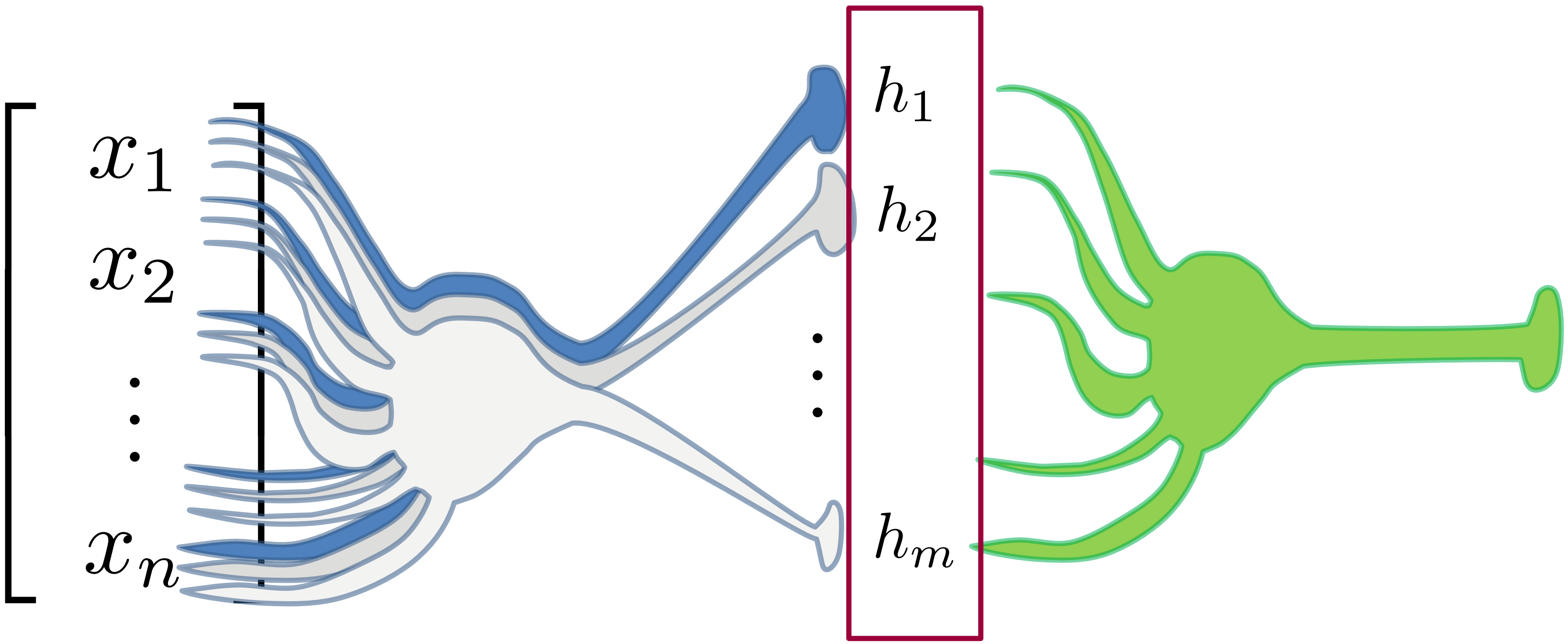


$$O = \underline{W_1} h_1 + \underline{W_2} h_2 + \ldots + \underline{W_m} h_m + \underline{B}$$

Two-Layer Network (~1980)



intermediate layer



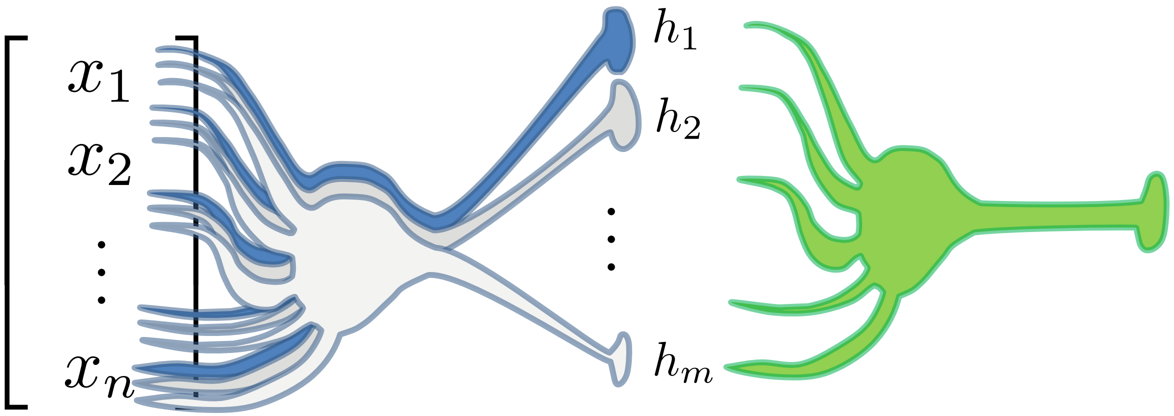
"features"

- the number m of "features" is a hyperparameter: it needs to be chosen;
- the values of the features are difficult to interpret (individually);

https://playground.tensorflow.org/

Universal Approximation Theorem

(almost) any classification problem can be solved by a two-layer network

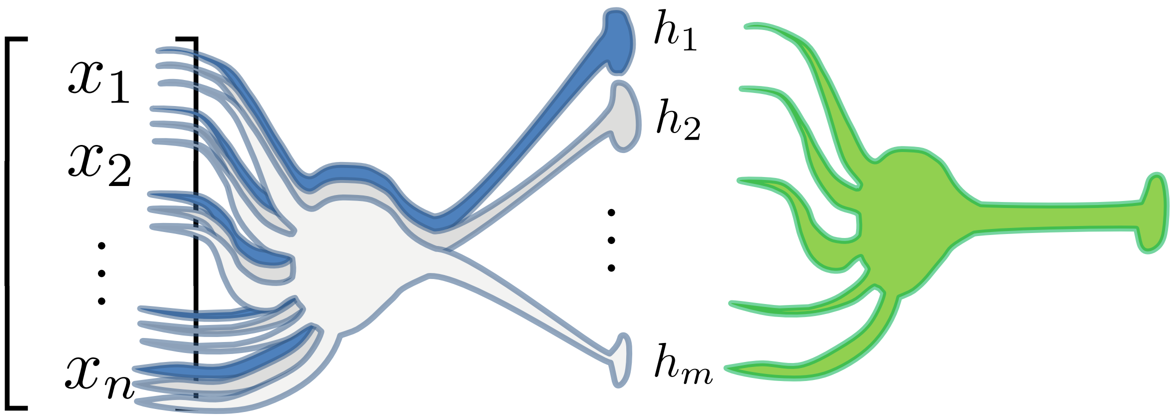


Universal Approximation Theorem

(almost) any classification problem can be solved by a two-layer network

BUT:

1. This assumes we have enough training data.
2. For difficult problems, using a two-layer network is intractable.

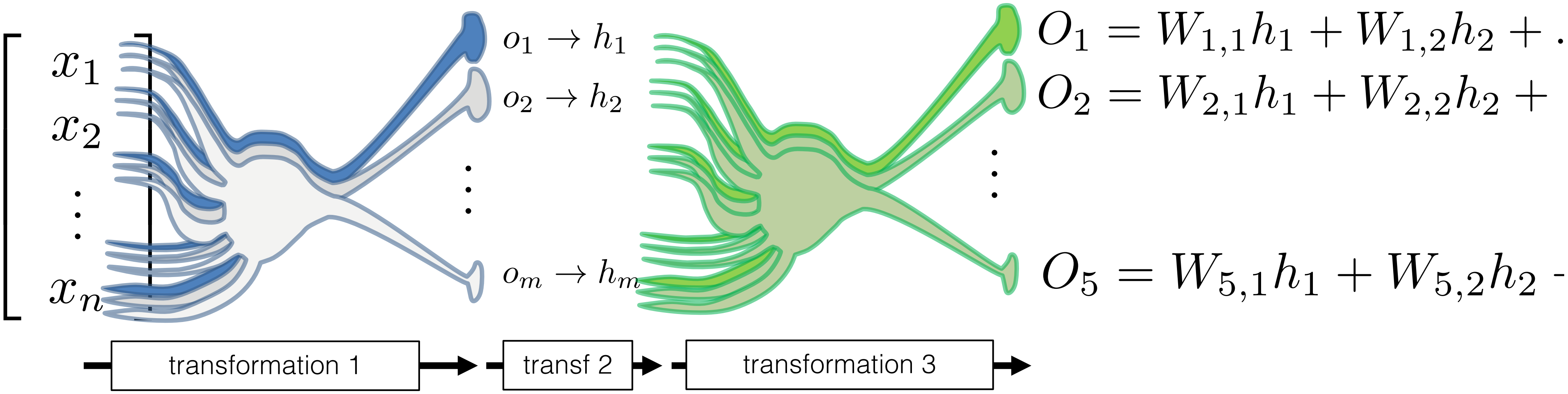


Fortunately, we can use more layers ie a deep network instead.

MULTI-CLASS PROBLEMS

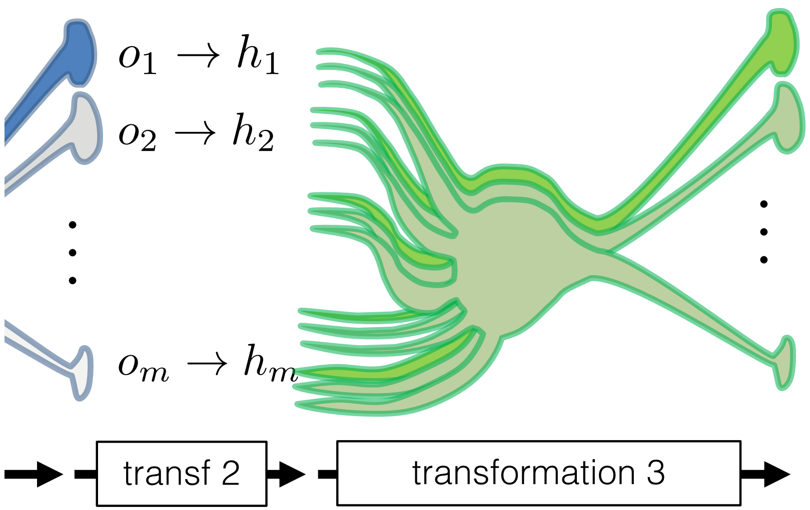
Multi-class problems

With a two-layer network, and 5 classes:



Multi-class problems

With a two-layer network, and 5 classes:



$$O_1 = W_{1,1}h_1 + W_{1,2}h_2 + \ldots + W_{1,m}h_m + B_1$$

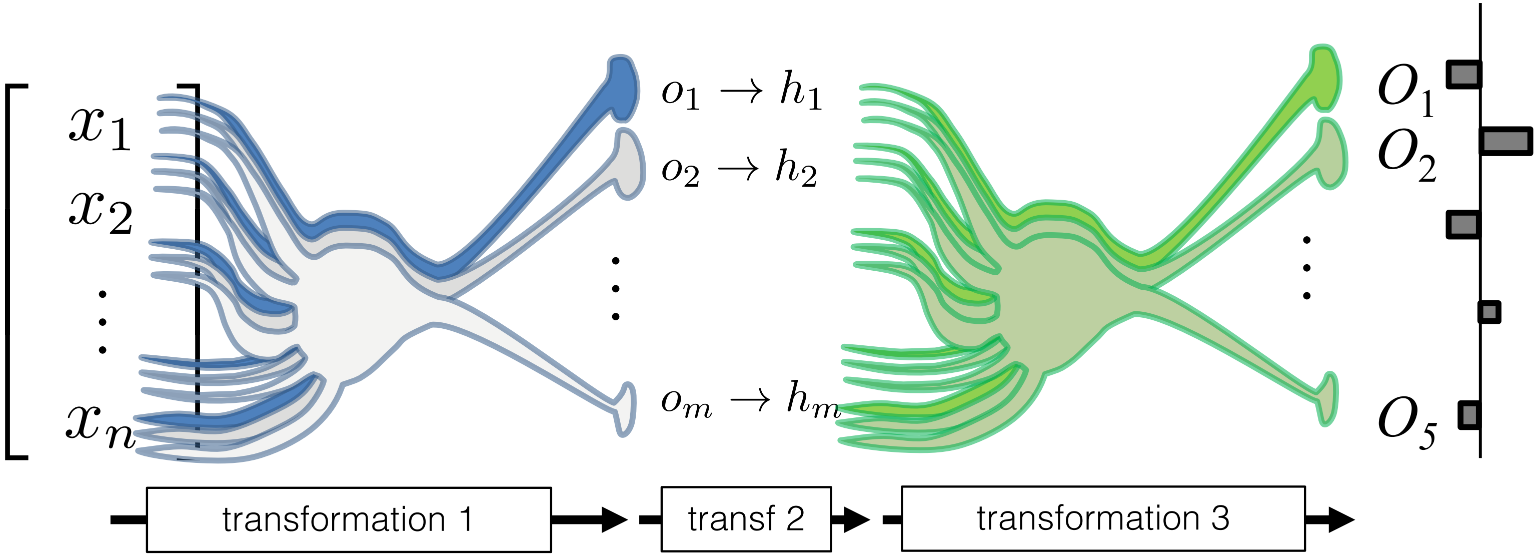
$$O_2 = W_{2,1}h_1 + W_{2,2}h_2 + \ldots + W_{2,m}h_m + B_2$$

$$\vdots$$

$$O_5 = W_{5,1}h_1 + W_{5,2}h_2 + \ldots + W_{5,m}h_m + B_5$$

Multi-class problems

With a two-layer network, and 5 classes:



the class predicted for this sample by the network is Class #2

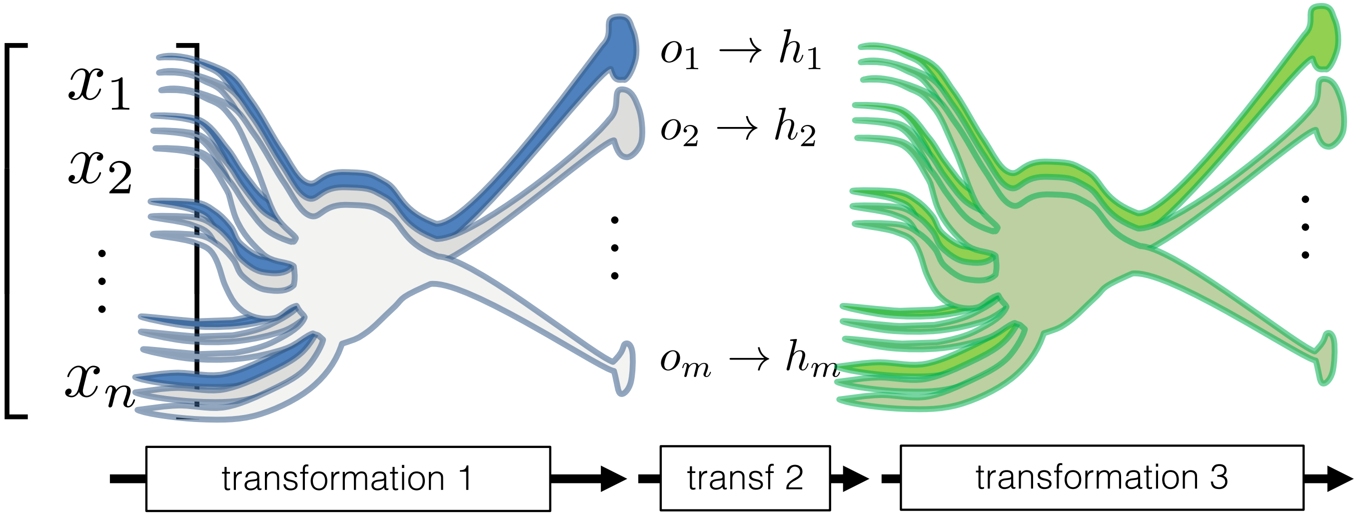
FINDING THE NETWORK'S PARAMETERS

the network's parameters

$$o_1 = \underline{w_{1,1}} x_1 + \underline{w_{1,2}} x_2 + \ldots + \underline{w_{1,n}} x_n + \underline{b_1}$$

$$o_2 = \underline{w_{2,1}} x_1 + \underline{w_{2,2}} x_2 + \ldots + \underline{w_{2,n}} x_n + \underline{b_2}$$

$$o_m = \underline{w_{m,1}} x_1 + \underline{w_{m,2}} x_2 + \ldots + \underline{w_{m,n}} x_n + \underline{b_m}$$



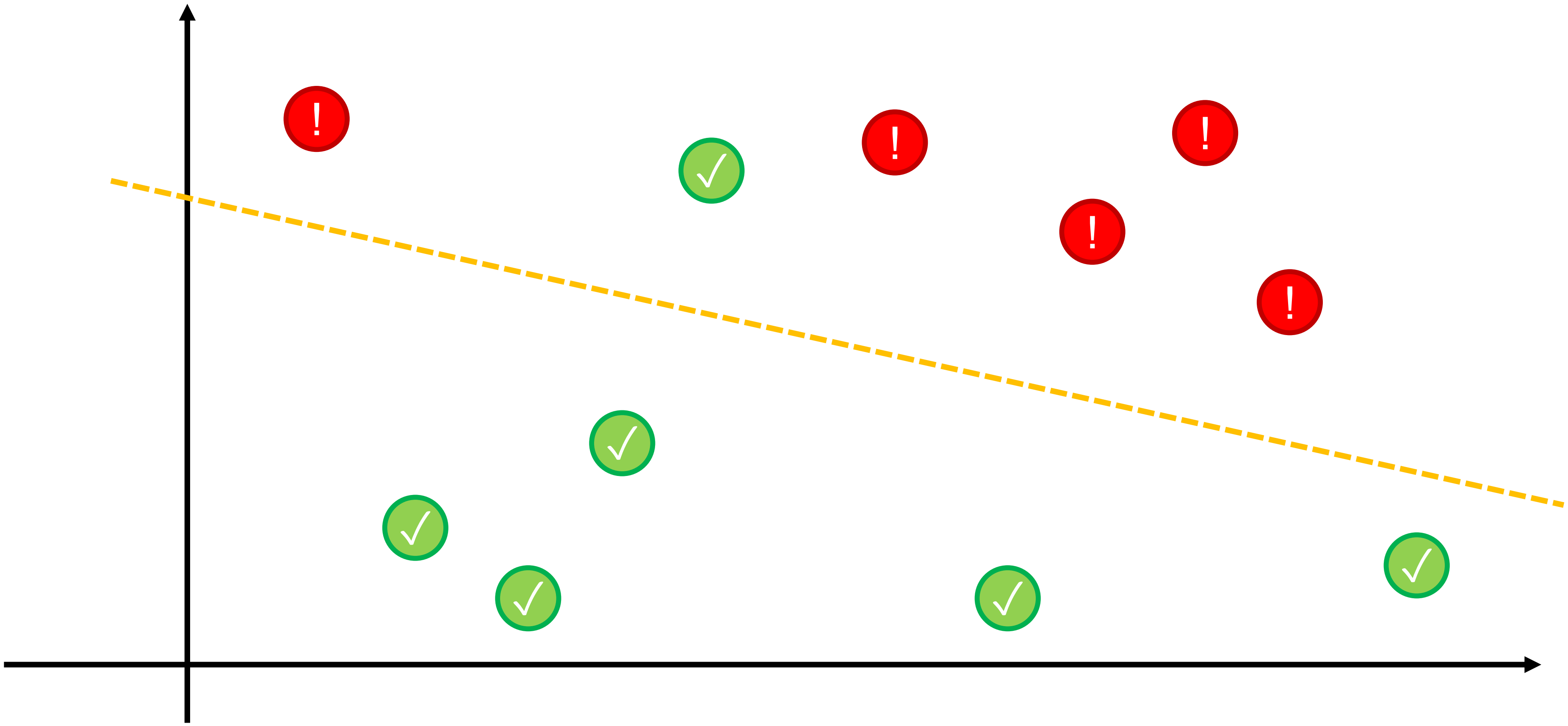
$$O_1 = \underline{W_{1,1}} h_1 + \underline{W_{1,2}} h_2 + .$$

$$O_2 = \underline{W_{2,1}} h_1 + \underline{W_{2,2}} h_2 +$$

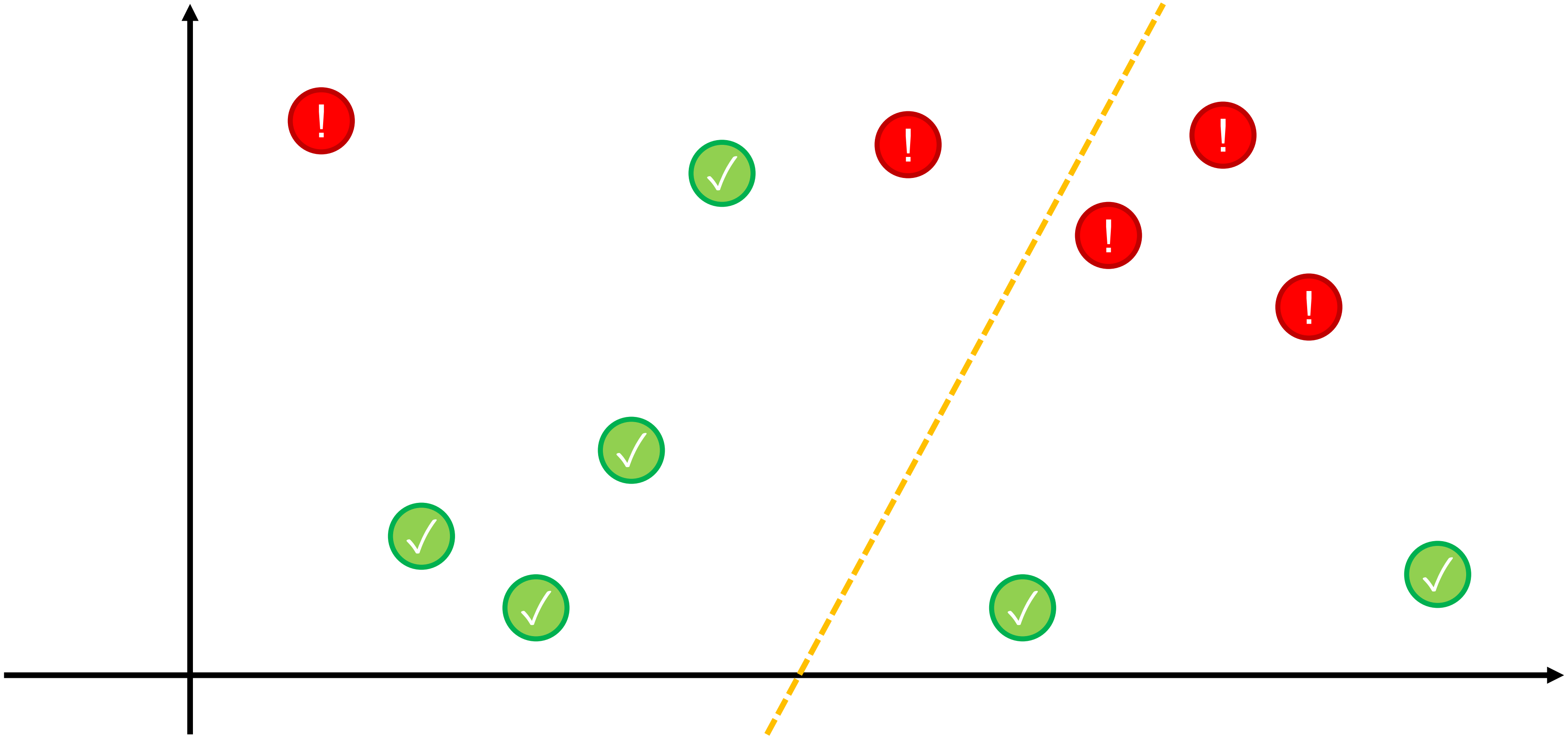
$$O_5 = \underline{W_{5,1}} h_1 + \underline{W_{5,2}} h_2 -$$

Given a training set, how can we find good values for these parameters?

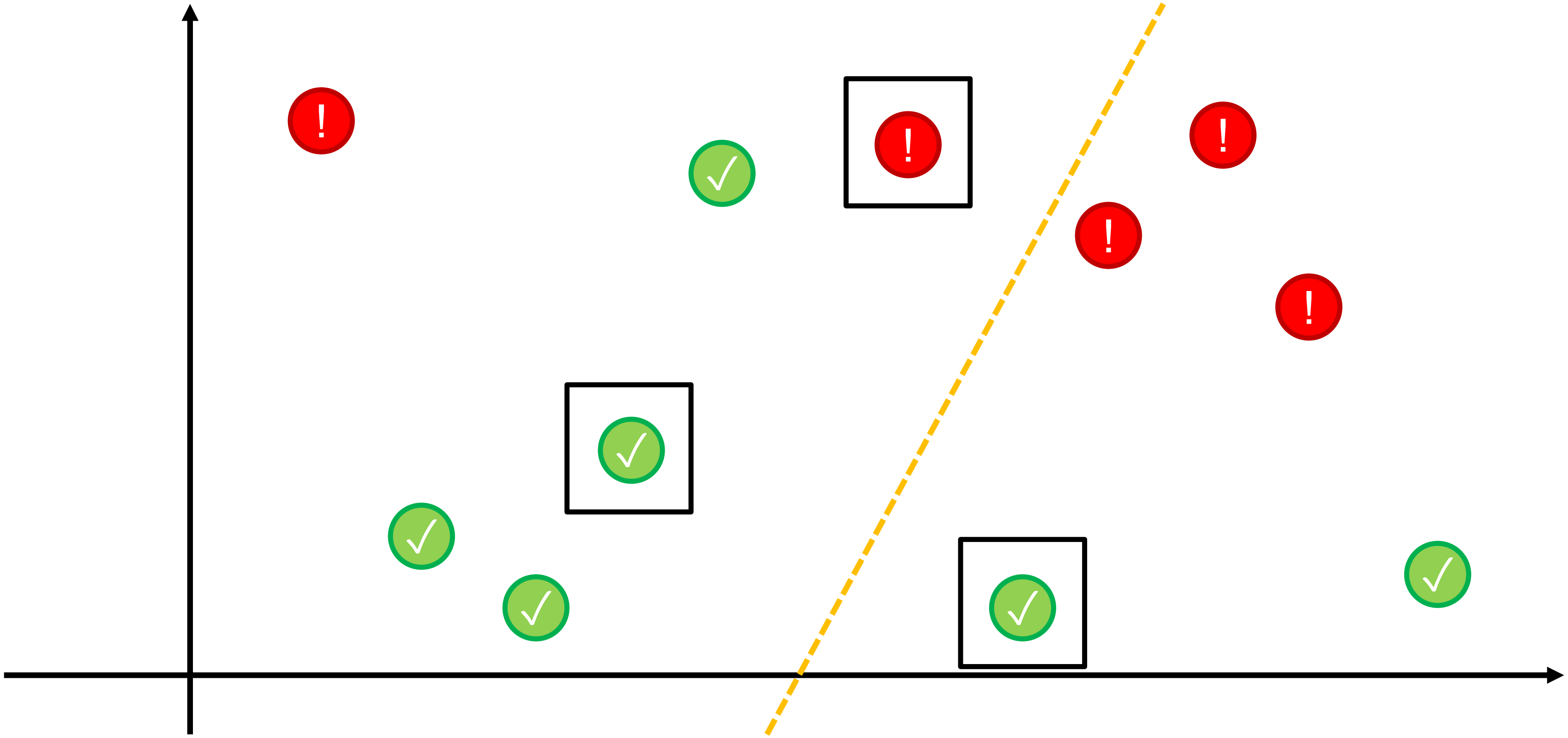
finding the parameters of a perceptron



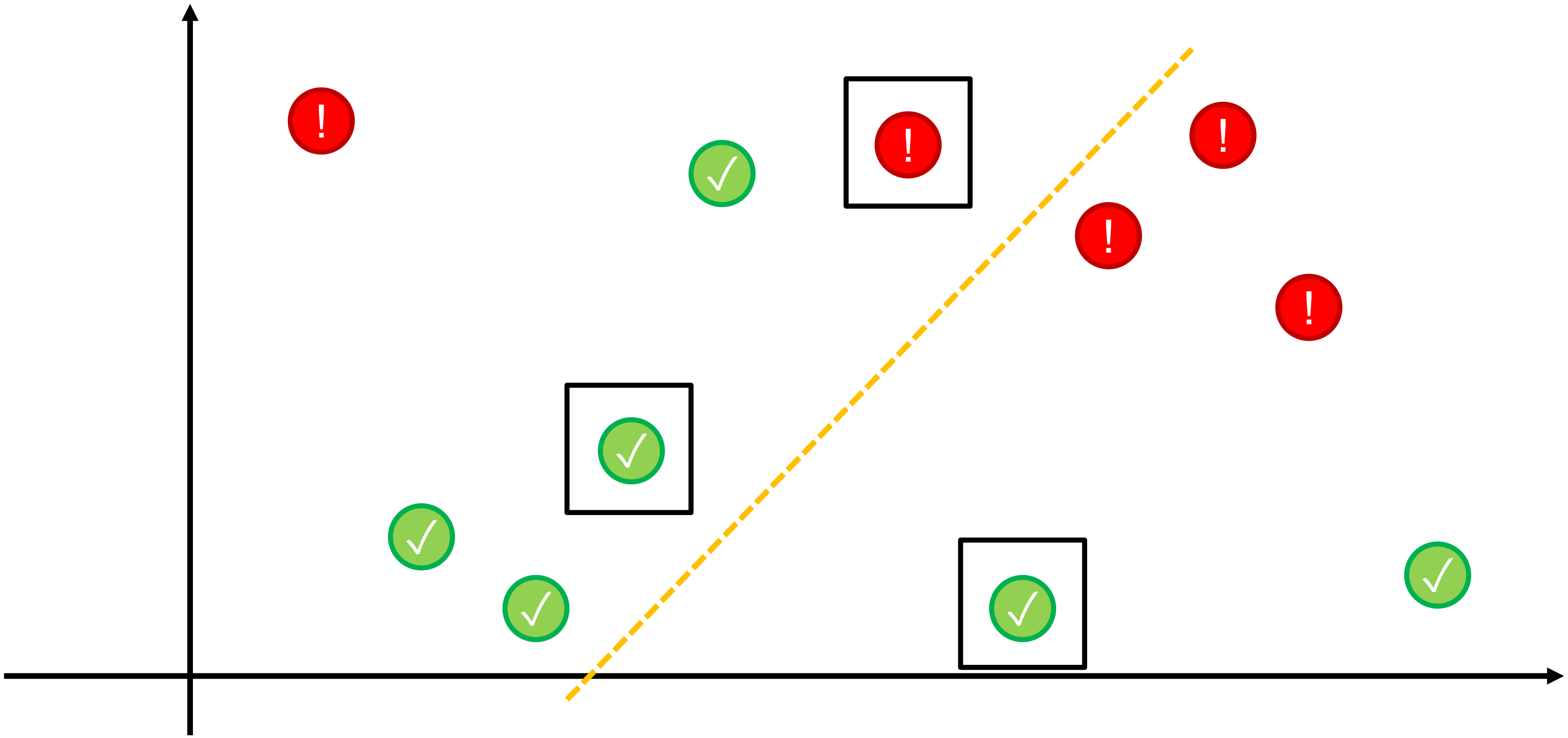
random initialization



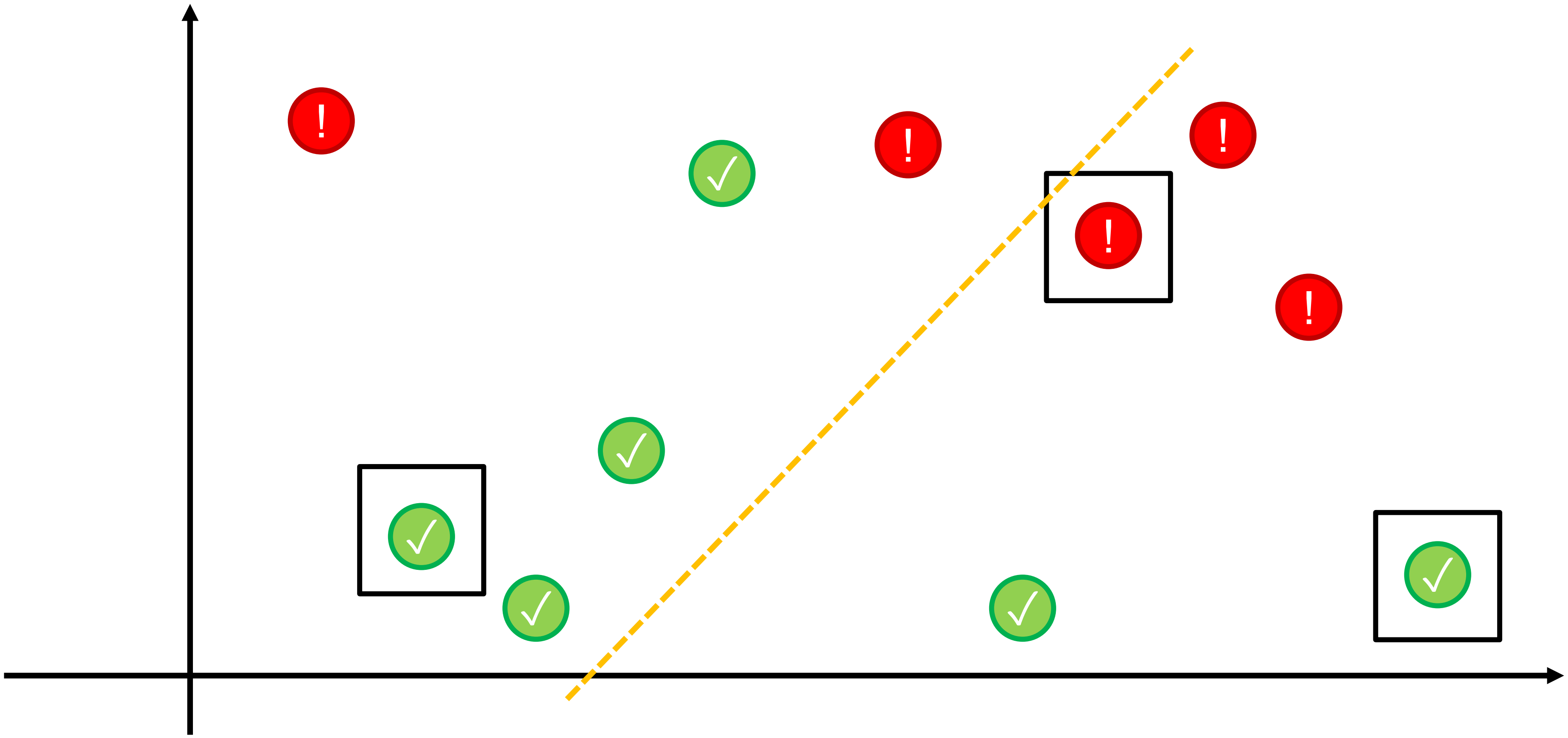
pick training samples randomly



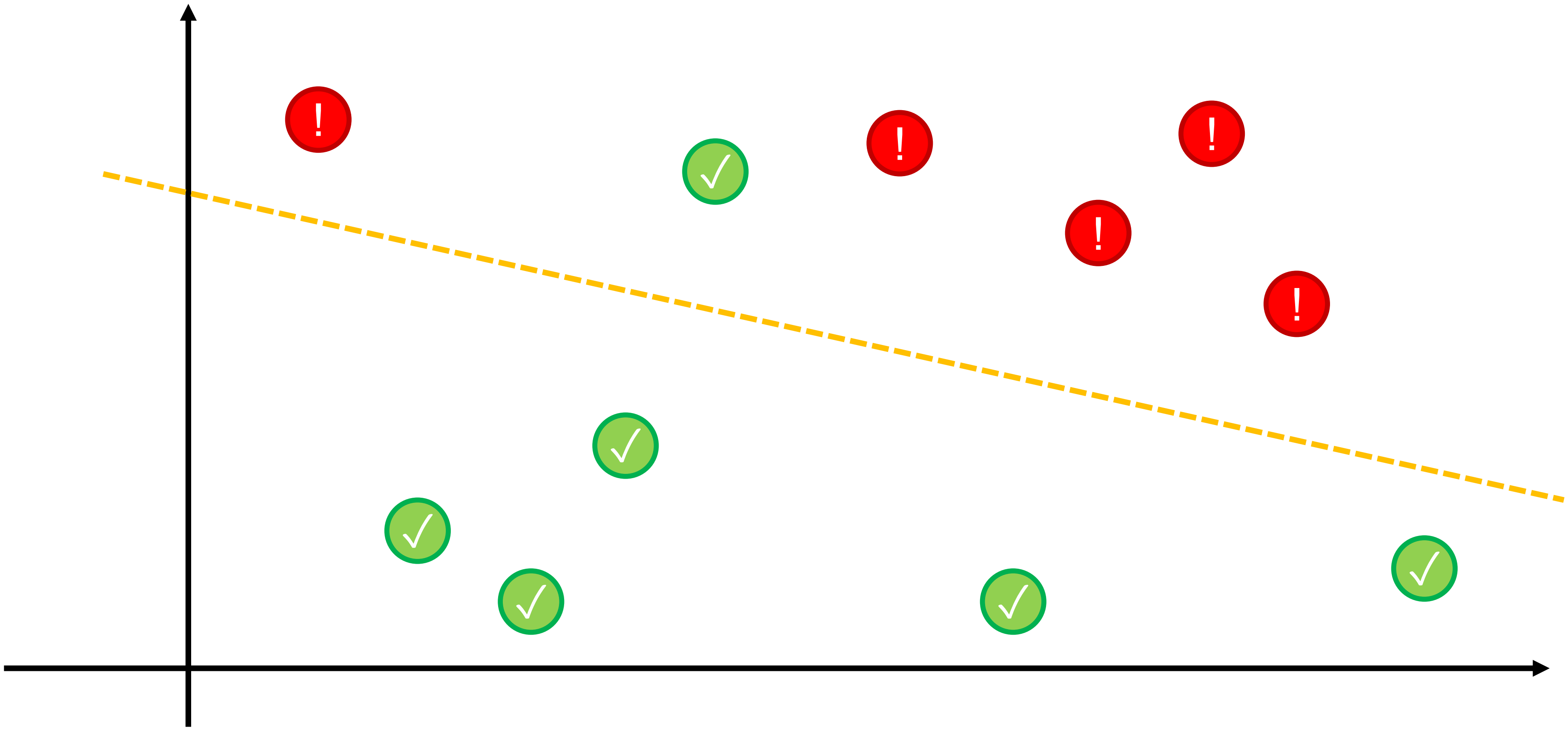
improve the parameters for these samples



iterate



until we get a good solution



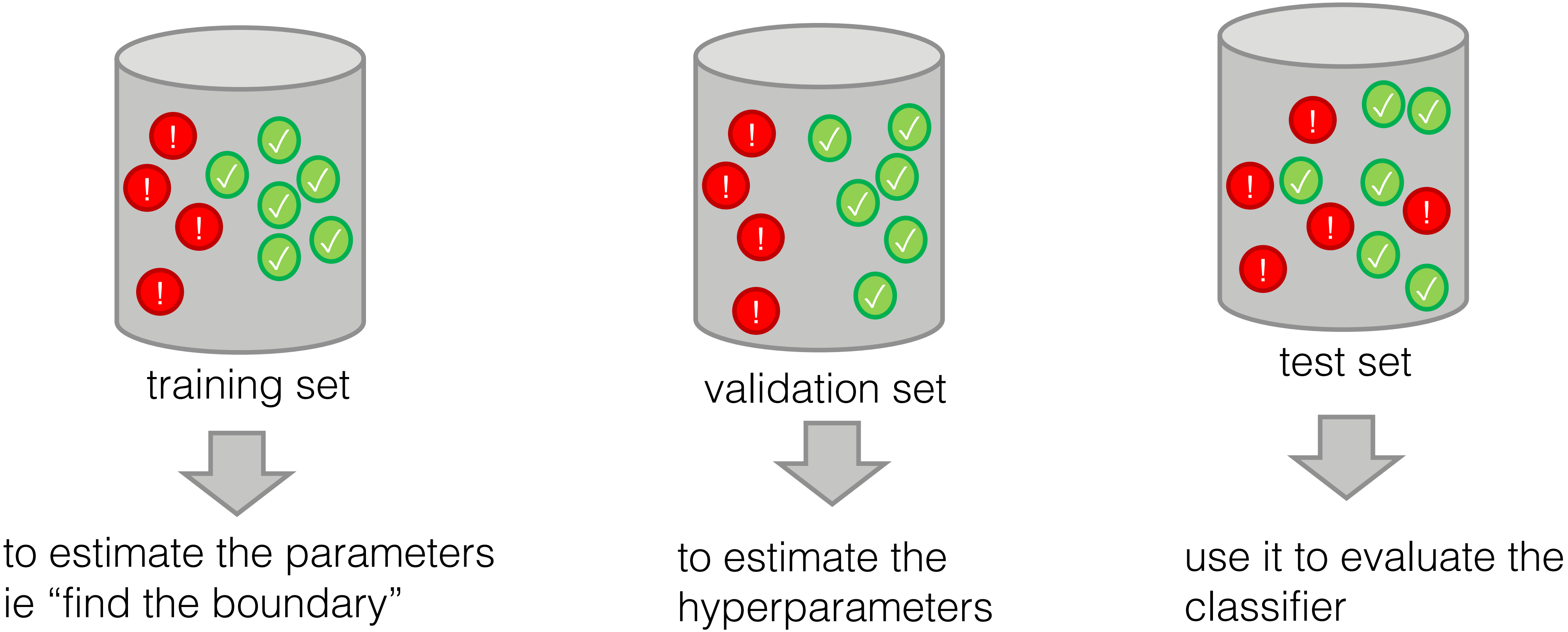
It can happen that we have

- good performance on the training set;
- poor performance on the validation set;
- poor performance on the test set.

This is called ‘overfitting’ (*surapprentissage*). “The method does not generalize well”

This may be due to many reasons *e.g.*

- the training and validation sets are not representative of the test set, or
- when the method has too many parameters.

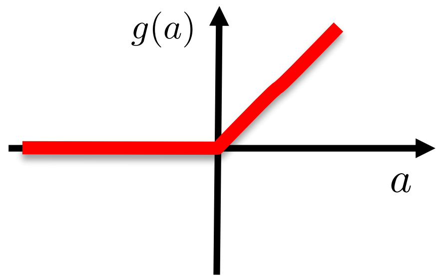


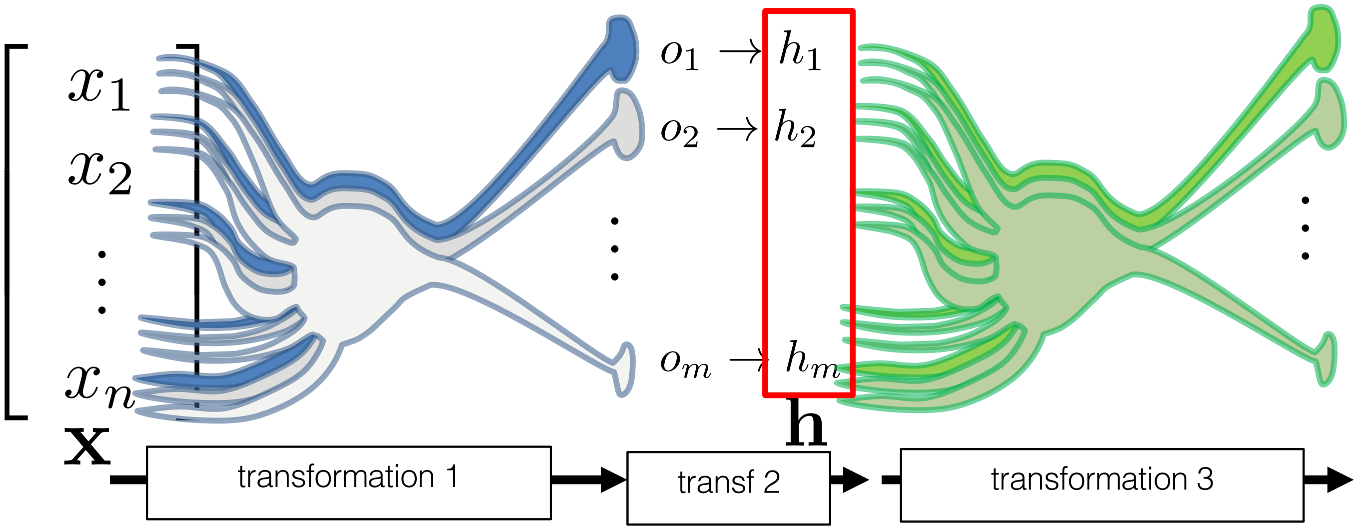
GOING BACK TO THE TWO-LAYER NETWORK

transformations

transformation 1: $\mathbf{o} = \underline{\mathbf{W}_1}\mathbf{x} + \underline{\mathbf{b}_1}$

transformation 2: $\mathbf{h} = g(\mathbf{o}) = g(\mathbf{W}_1\mathbf{x} + \mathbf{b1})$





transformation 3:

$\mathbf{O} = \underline{\mathbf{W}_2}\mathbf{h} + \underline{\mathbf{b}_2}$

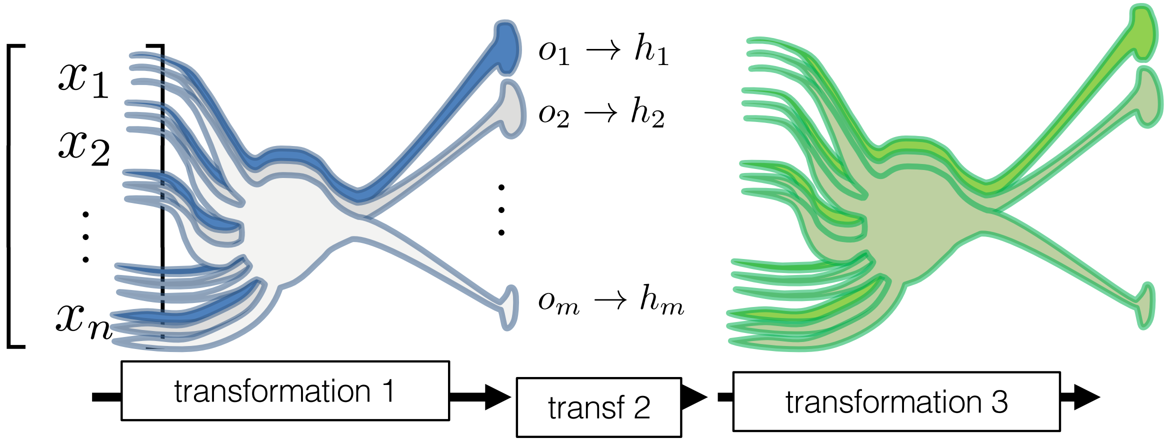
(linear/affine) transformation

$$\Theta = (w_{1,1}, w_{1,2}, \ldots, w_{1,n}, b_1, \ldots, b_m, W_{1,1}, W_{1,2}, \ldots)$$

A FIRST ”DEEP NETWORK”

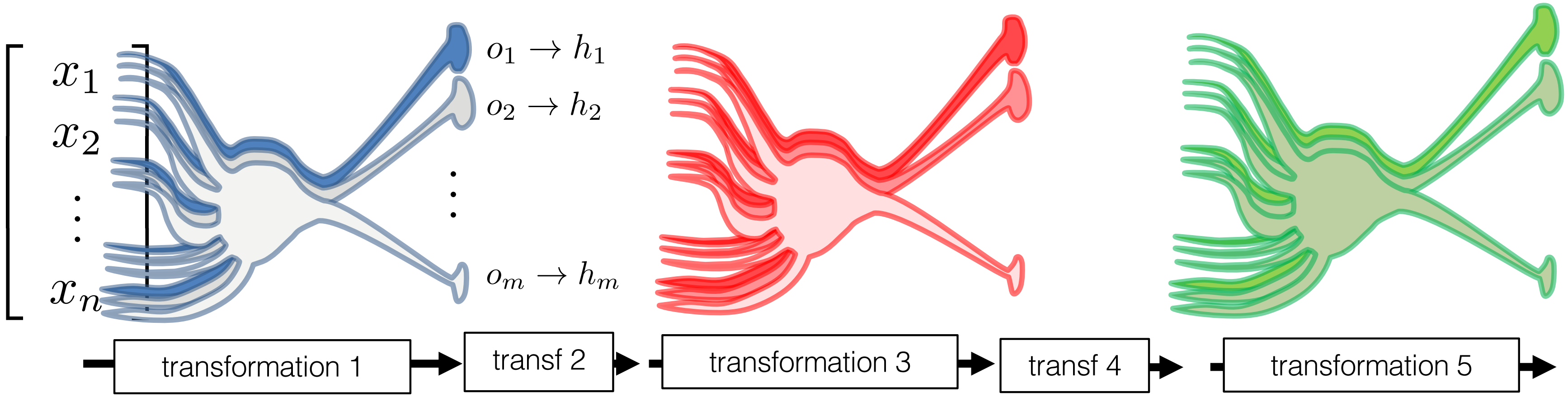
Multi-Layer Networks/
Multi-Layer Perceptrons

Two layers:



Multi-Layer Networks/ Multi-Layer Perceptrons: a first „deep network“

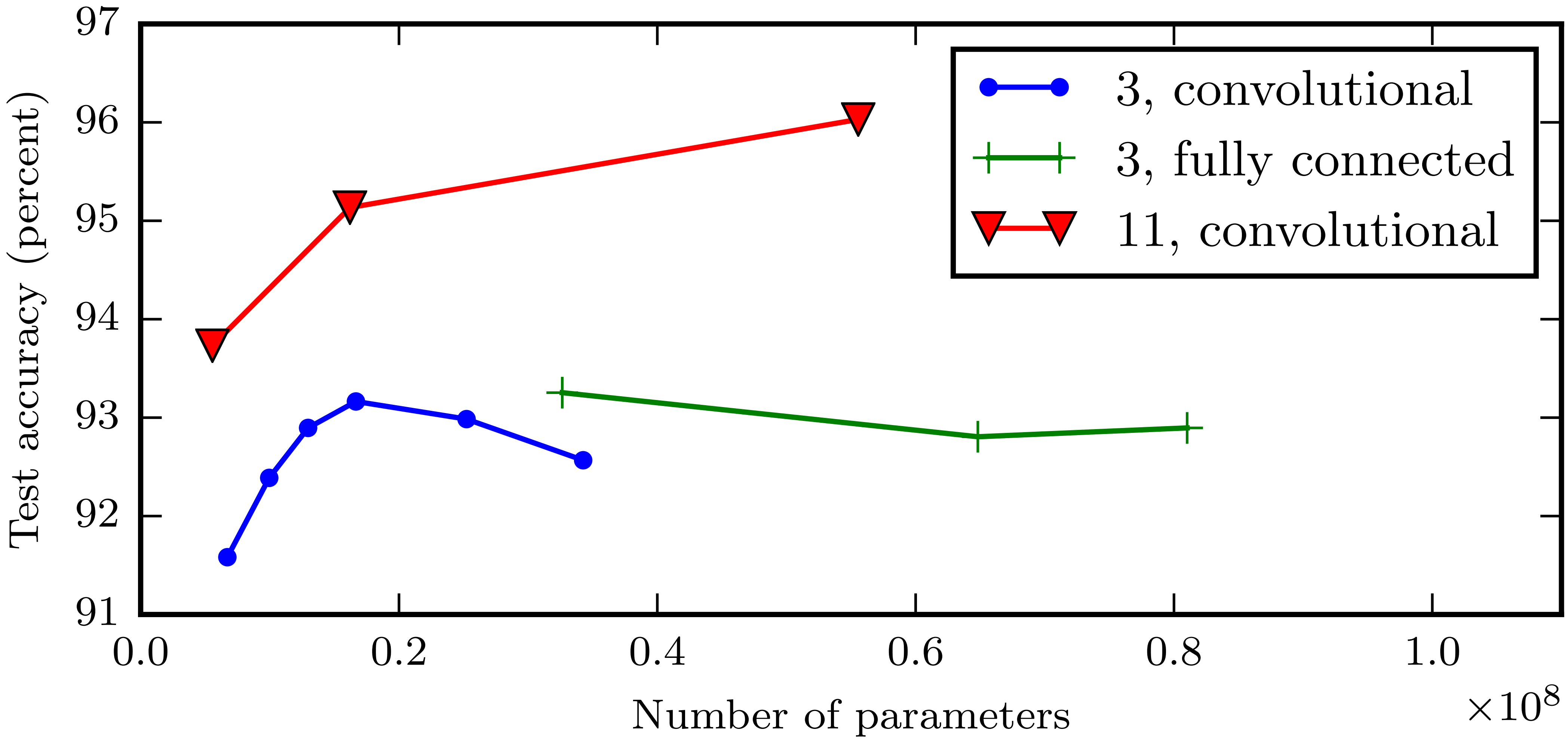
Three layers:



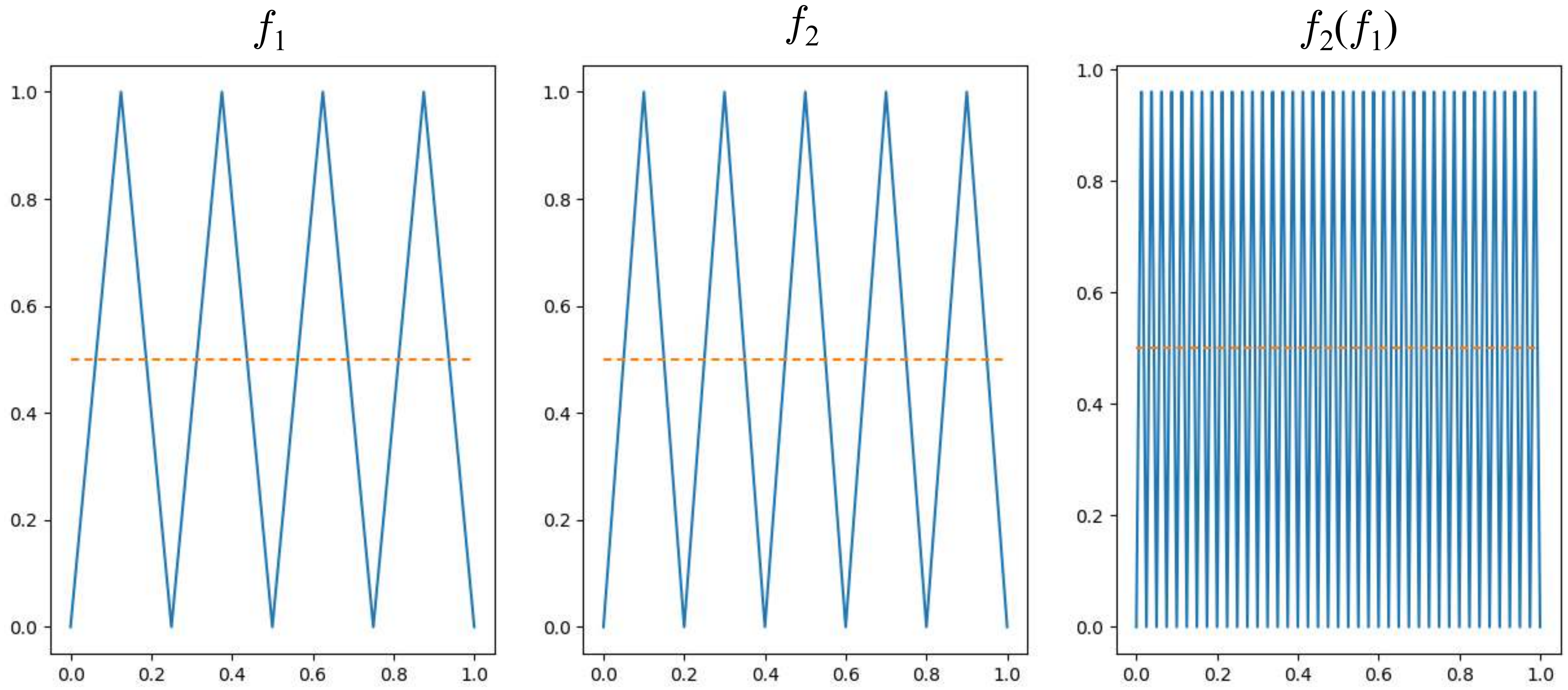
Can be used exactly the same way as a two-layer network;

Advantage: Can work better than a two-layer network.

Shallow Networks Overfit More

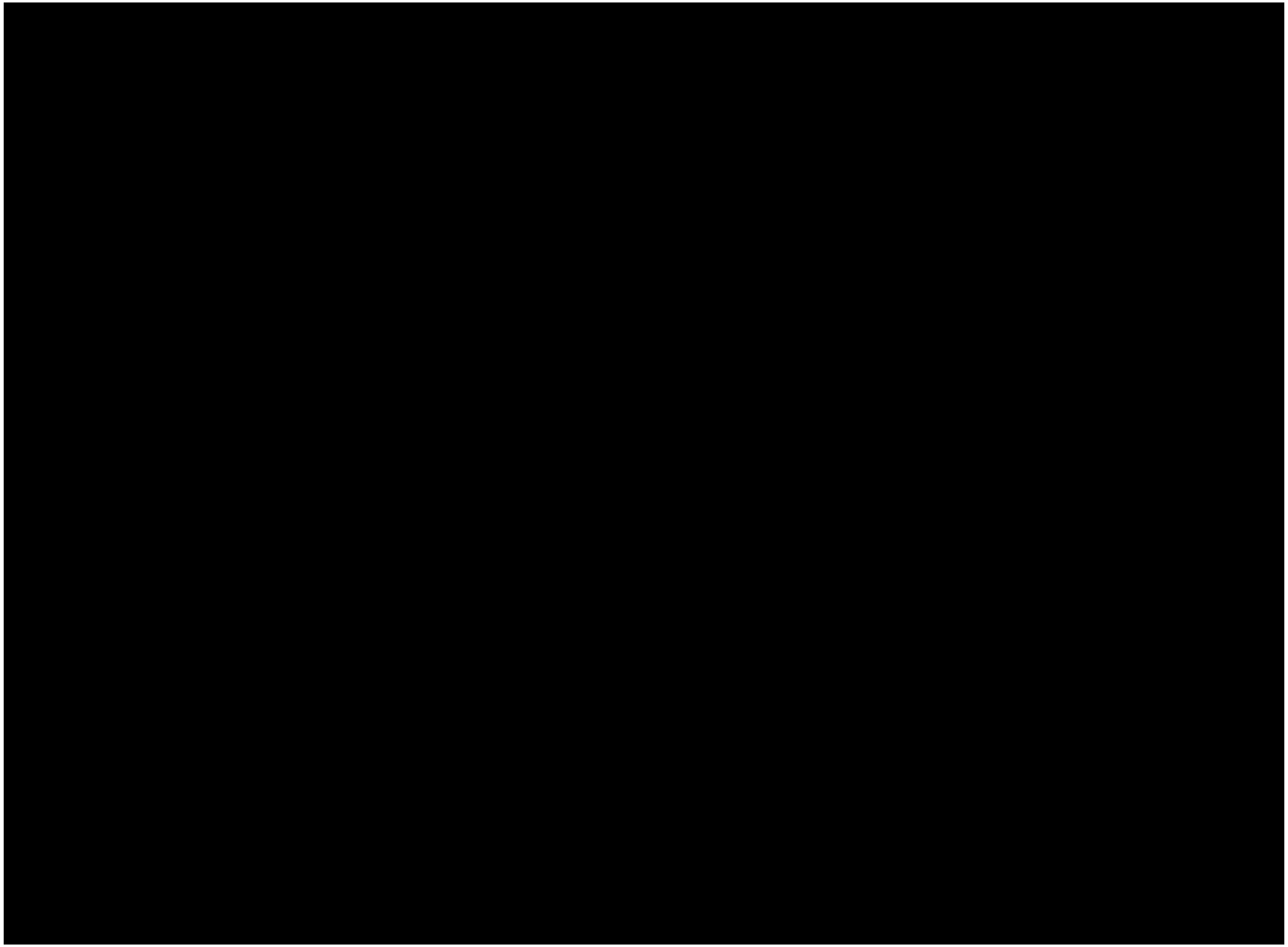


the power of compositions

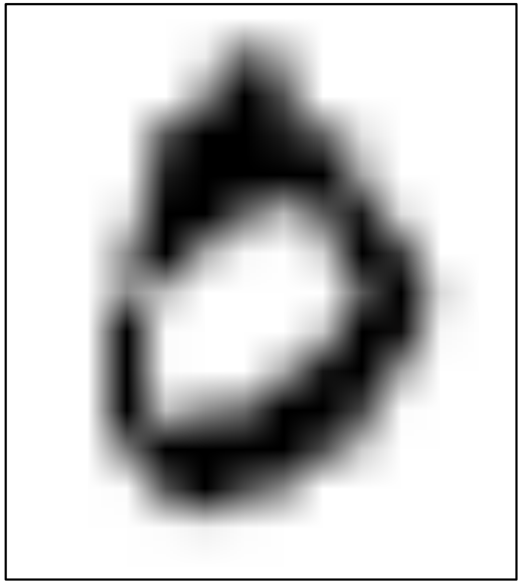


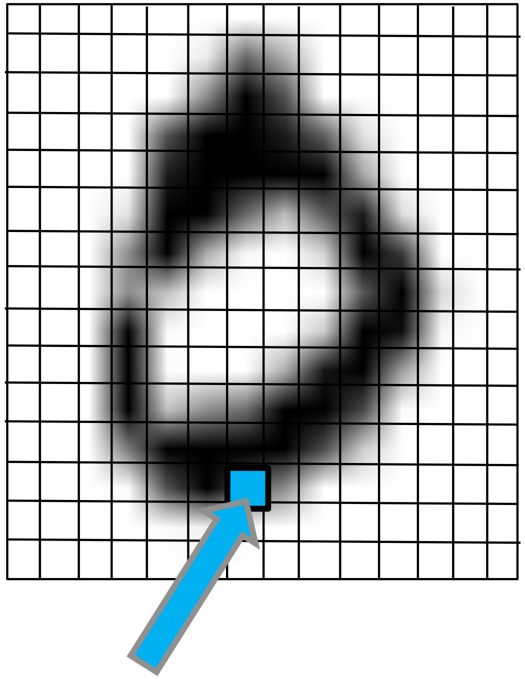
A DEEPER NETWORK

LeNet5 (LeCun, 1992)



An Image is a Set of Values





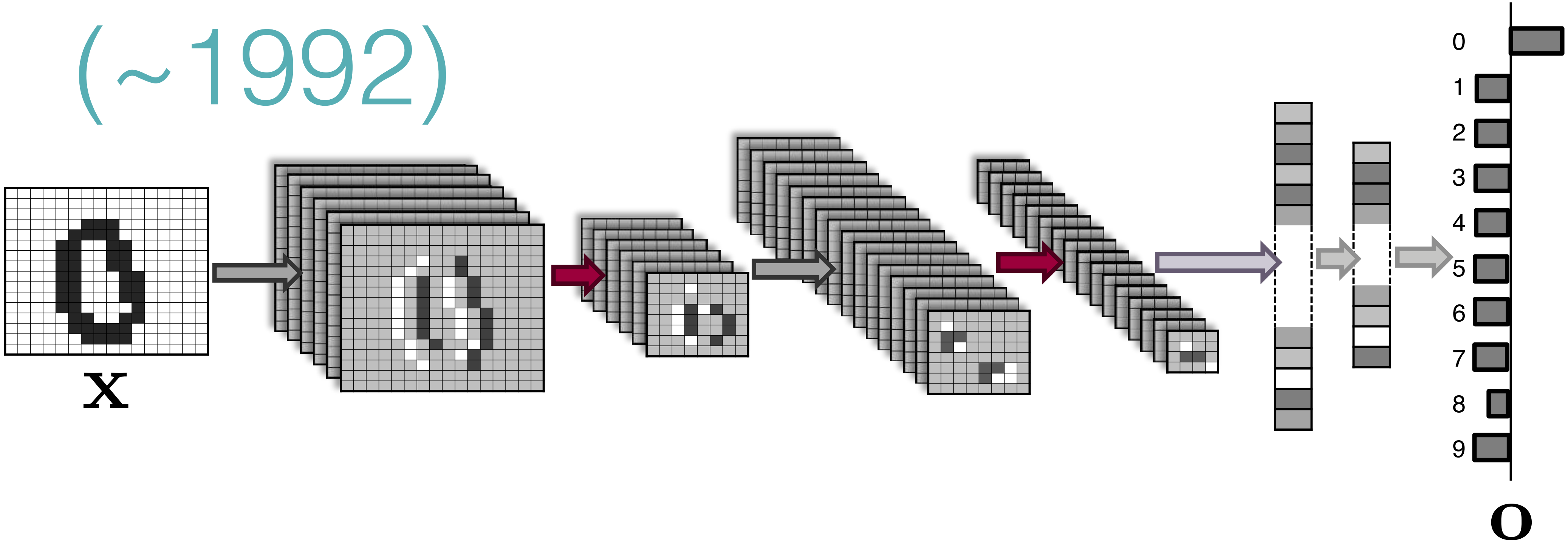
To each pixel correspond 1 value (3 for color images).

(for grayscale images: 0 for black, 255 for white)

210 values for an image of size 14 x 15 pixels.

262’000+ values for an image of size 512 x 512 pixels.

Deep Learning *a la* LeCun (~1992)

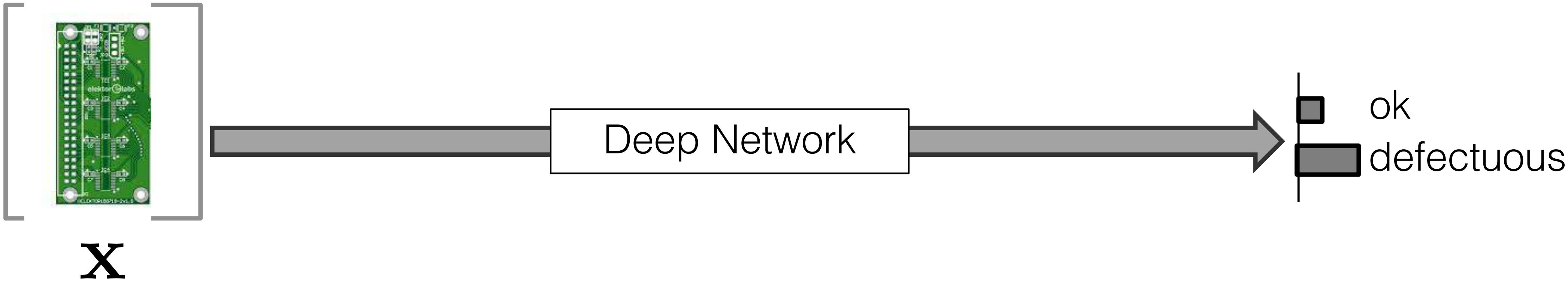


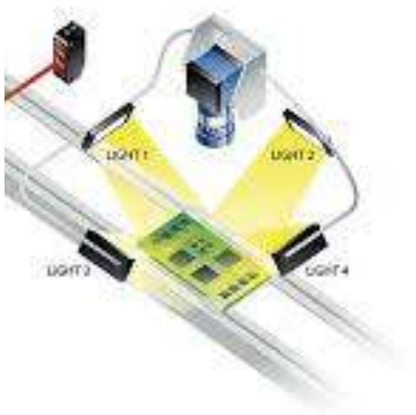
application example

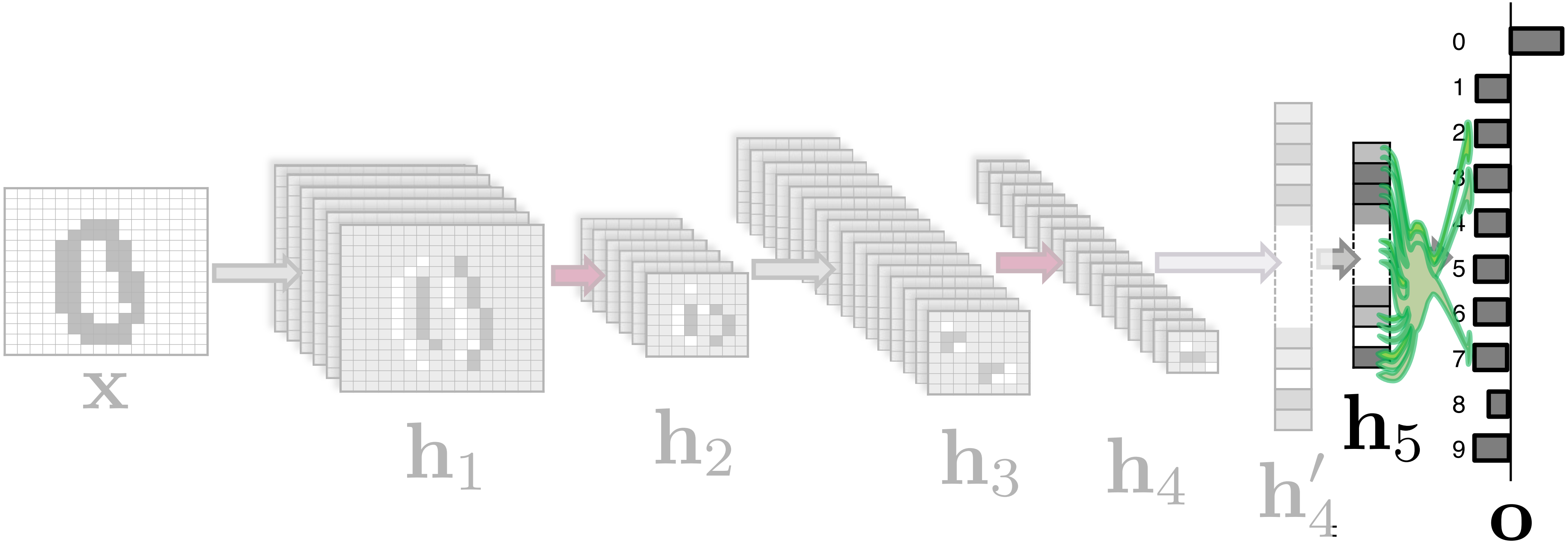
$\mathbf{x}$

lung ultrasounds

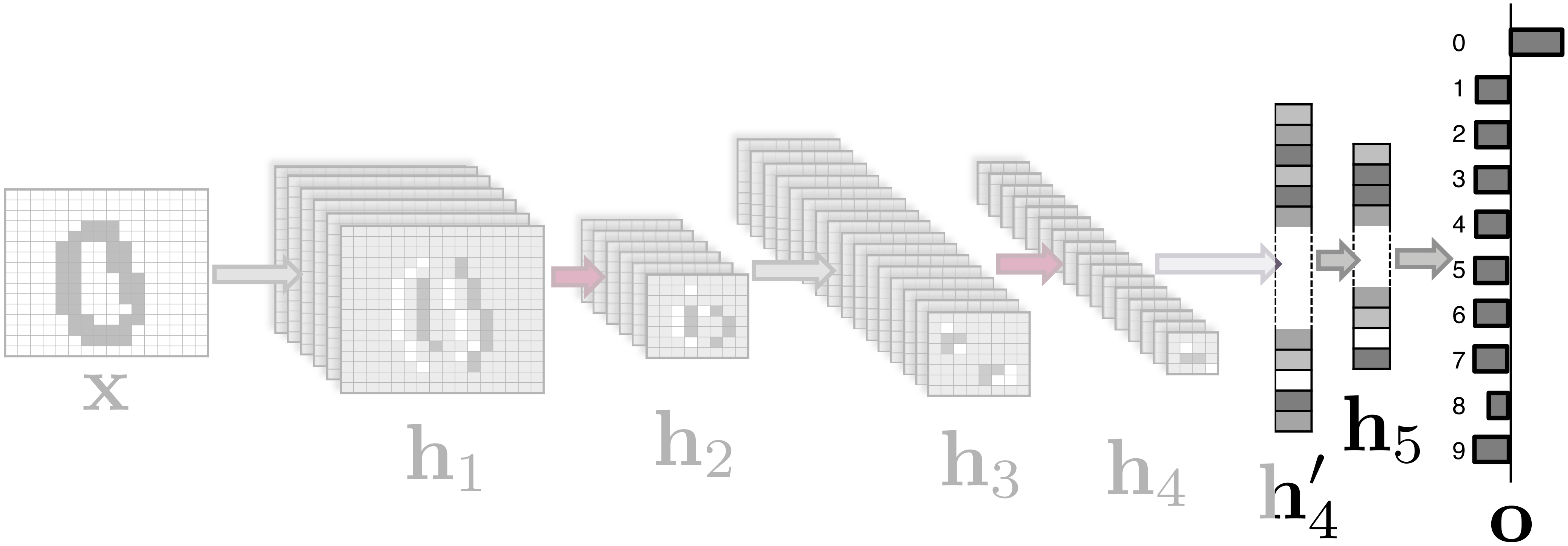
application example (2)



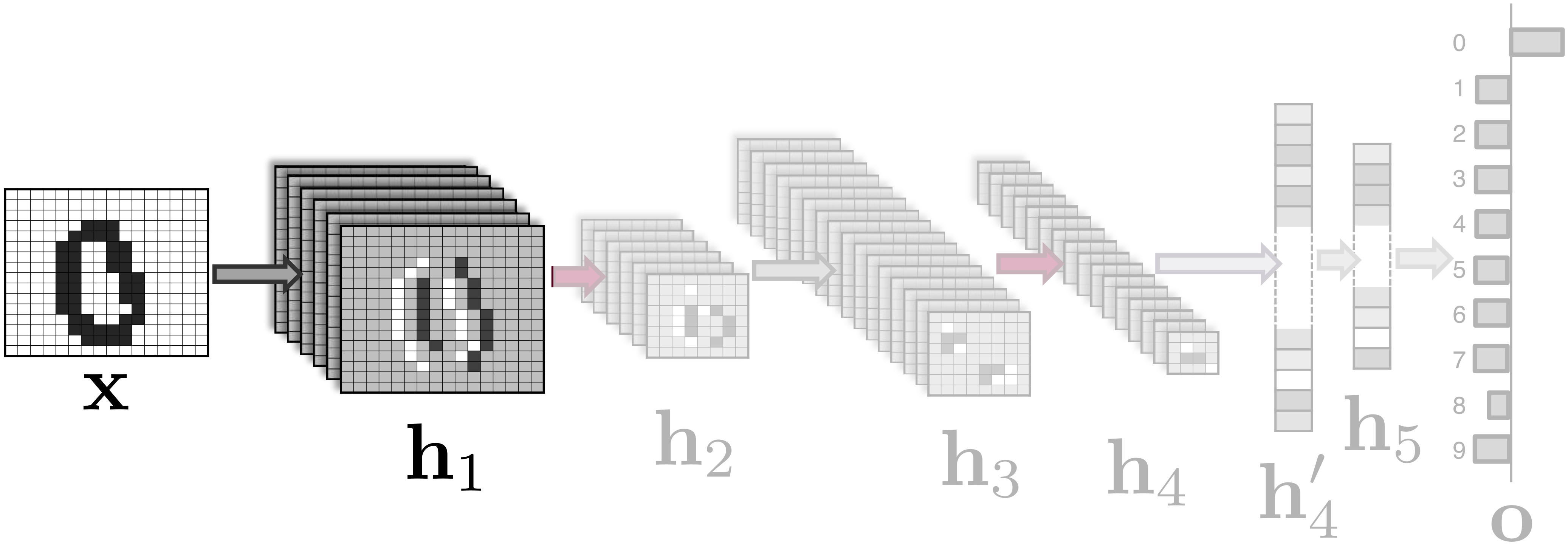


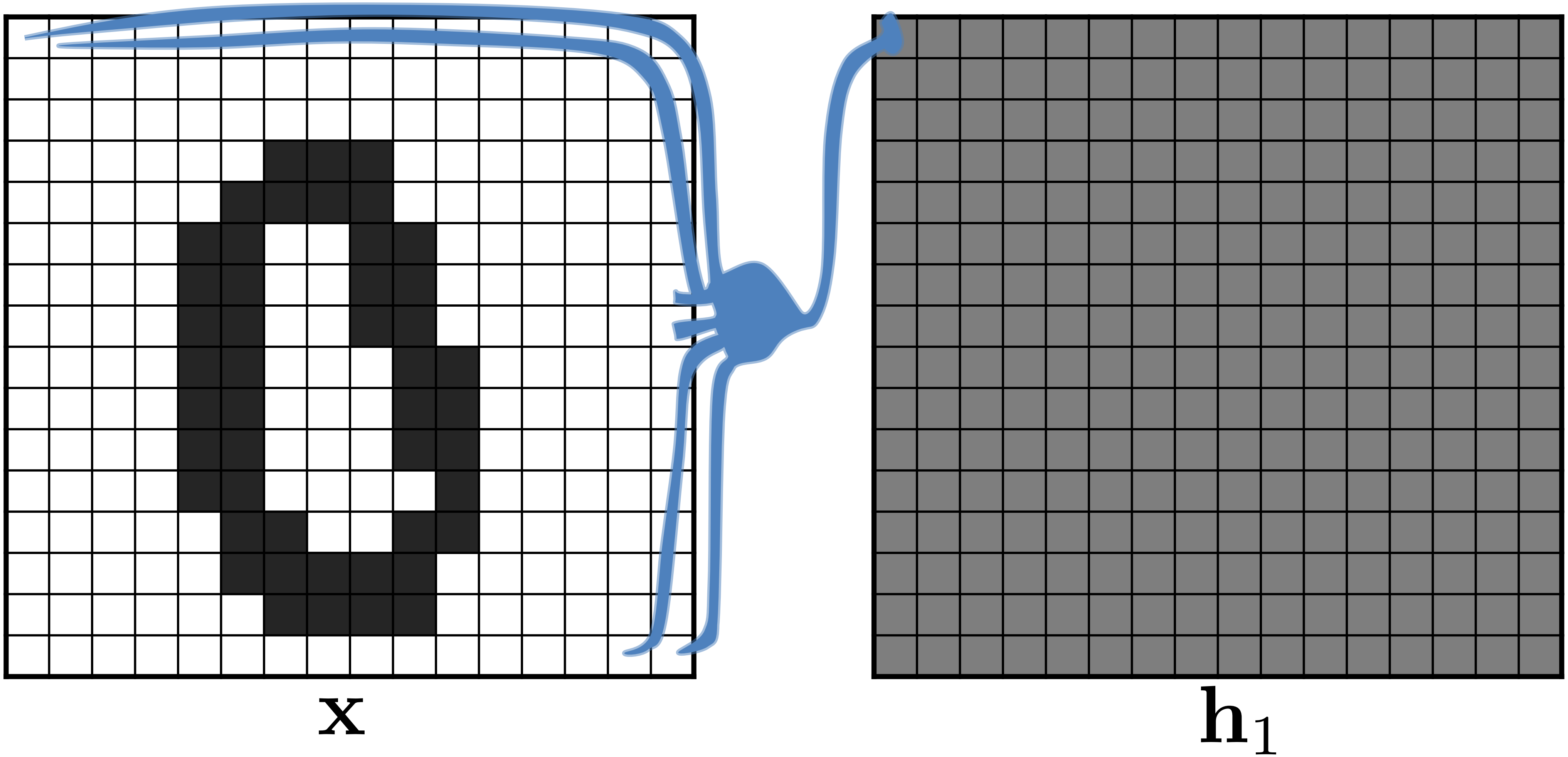


$$\mathbf{o} = \mathbf{W}_6 \mathbf{h}_5 + \mathbf{b}_6$$

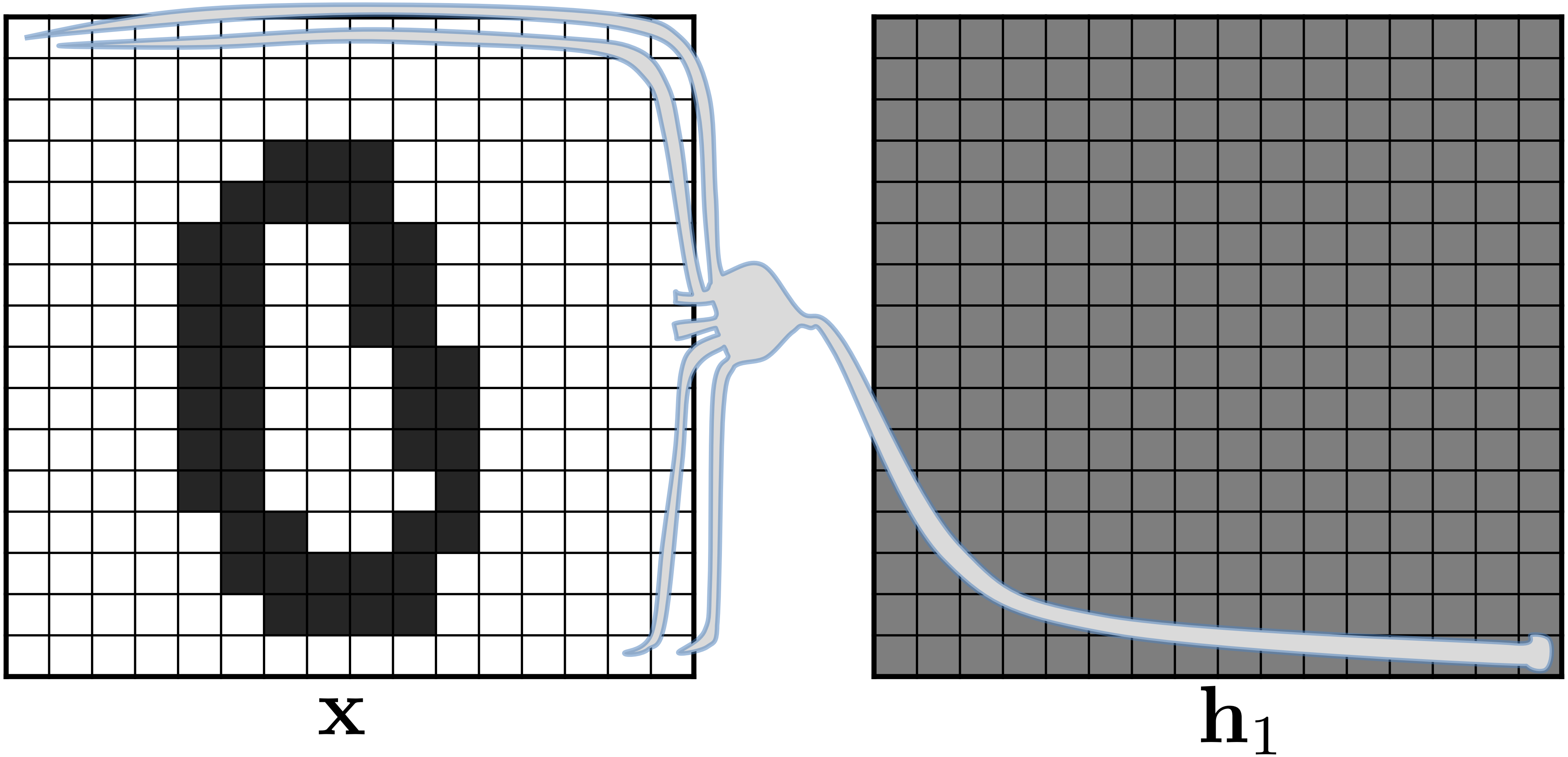


$$\mathbf{h}_5 = g(\mathbf{W}_5 \mathbf{h}'_4 + \mathbf{b}_5)$$
$$\mathbf{o} = \mathbf{W}_6 \mathbf{h}_5 + \mathbf{b}_6$$

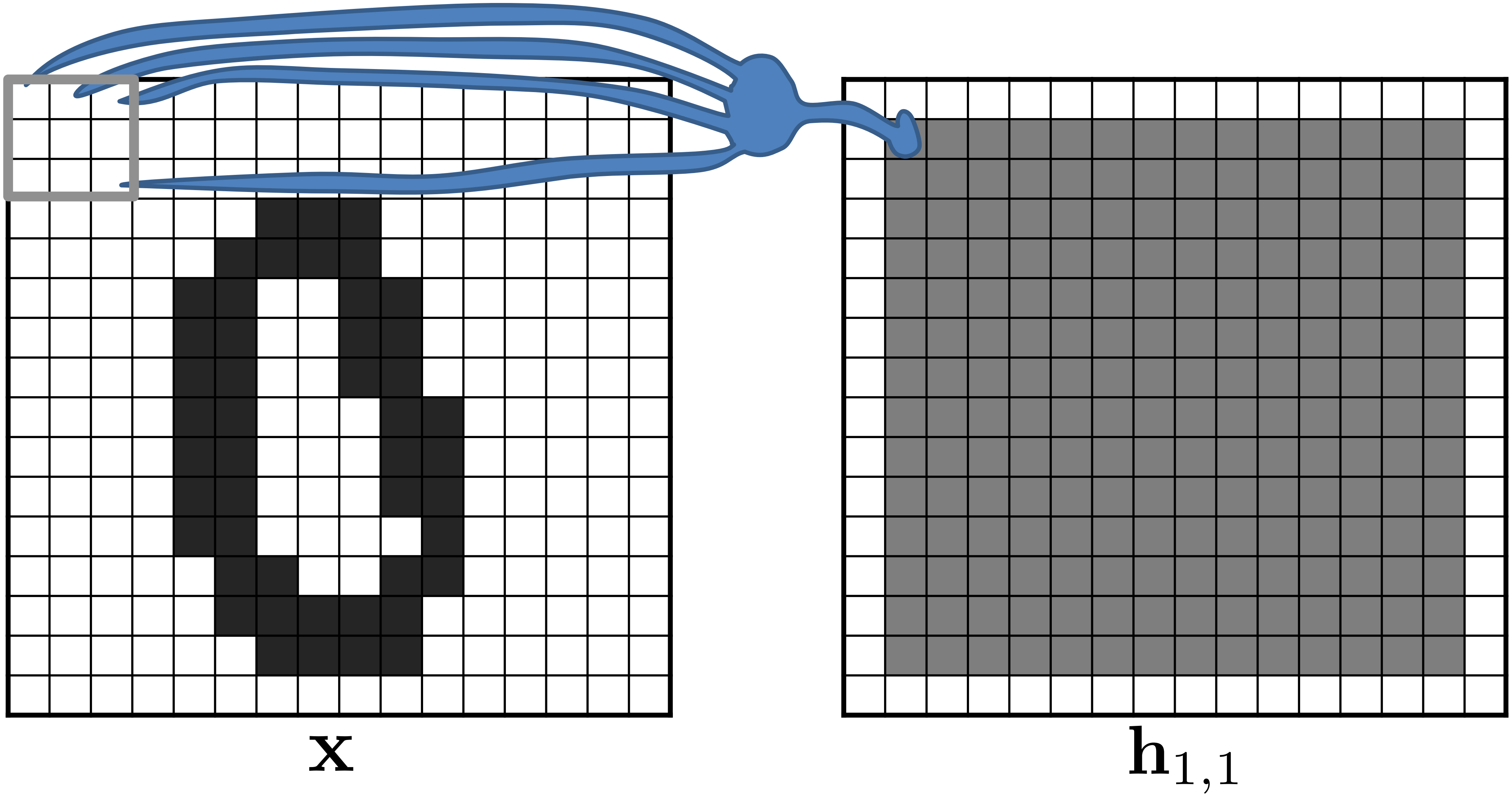

x
h1
h2
h3
h4
h'4
h5
0
1
2
3
4
5
6
7
8
9
o

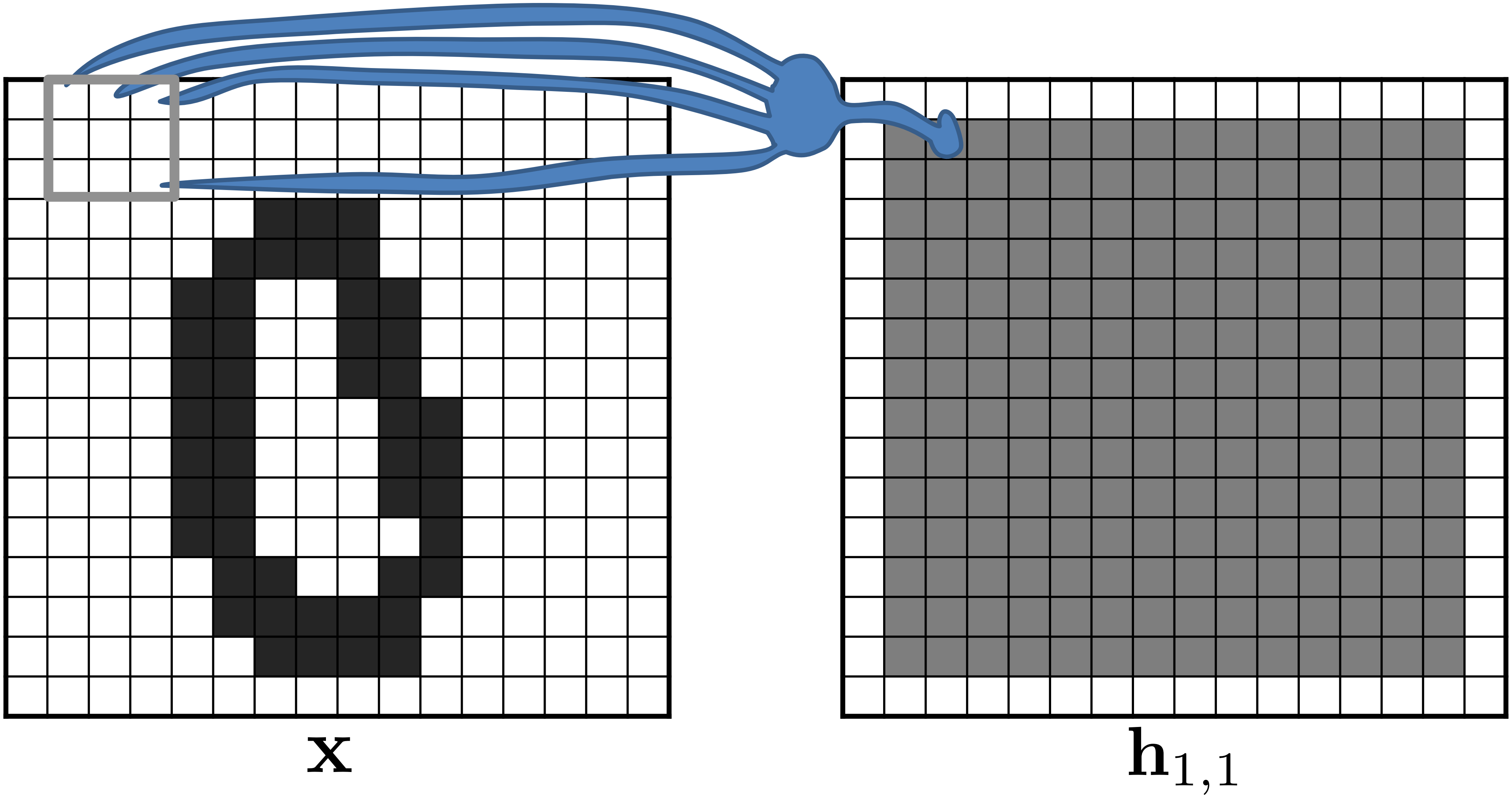


$$\mathbf{h}_1 = g(\mathbf{W}\mathbf{x} + \mathbf{b}) \; ?$$

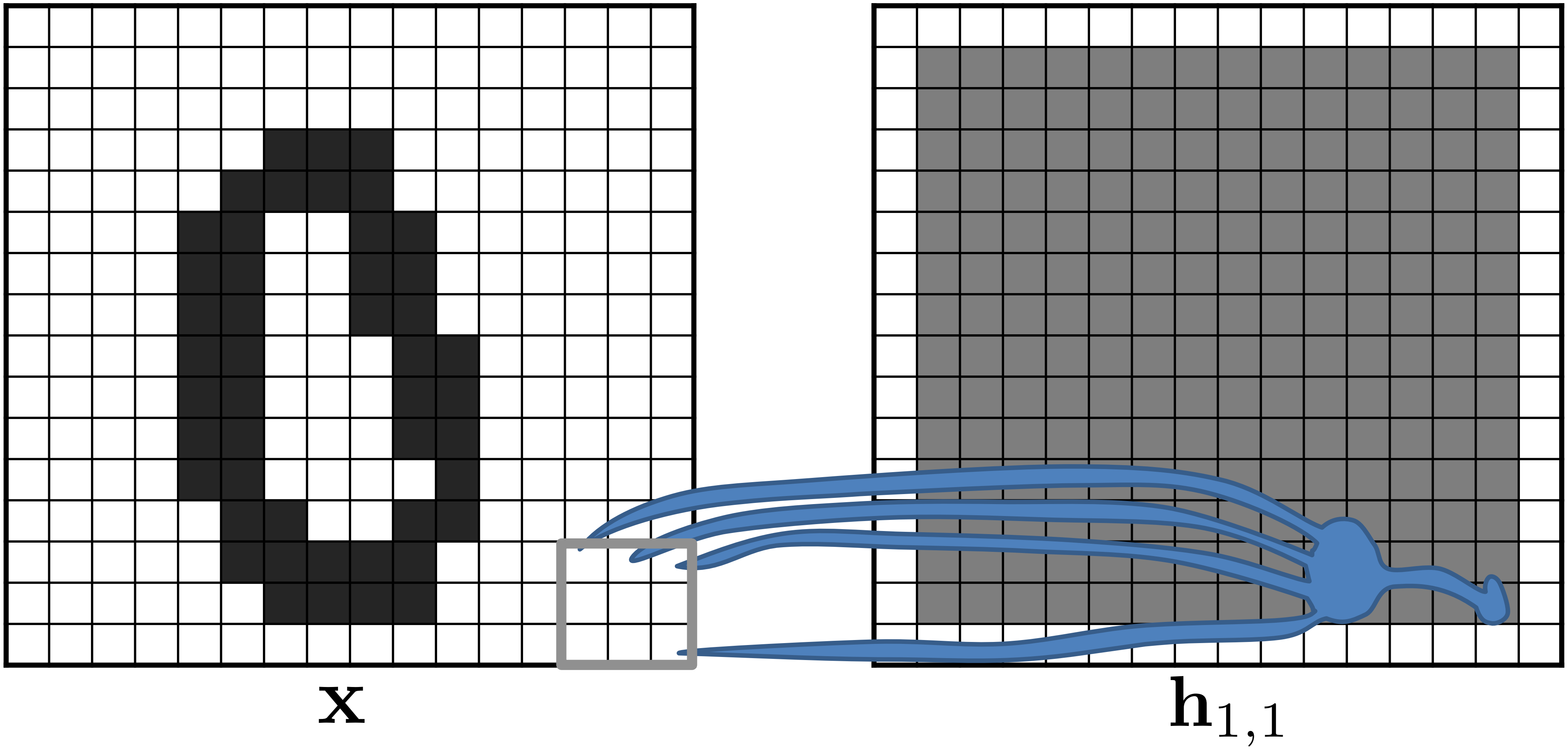


$$\mathbf{h}_1 = g(\mathbf{W}\mathbf{x} + \mathbf{b}) \ ?$$


$\mathbf{x}$
$\mathbf{h}_{1,1}$

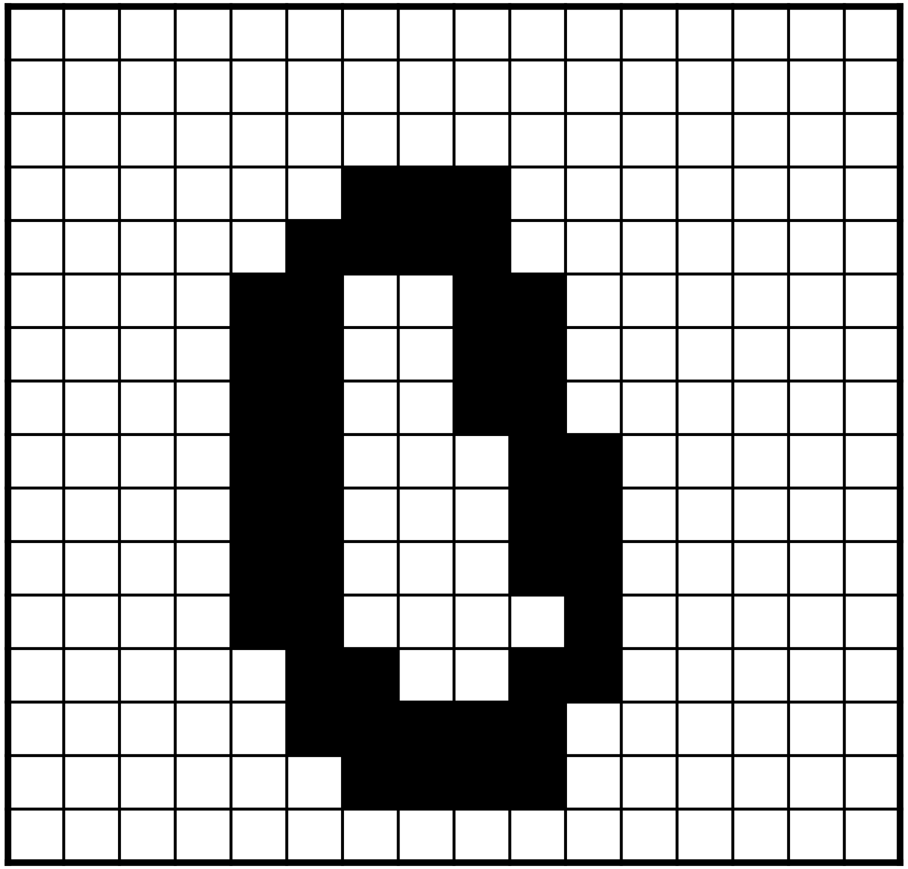


$\mathbf{x}$
$\mathbf{h}_{1,1}$

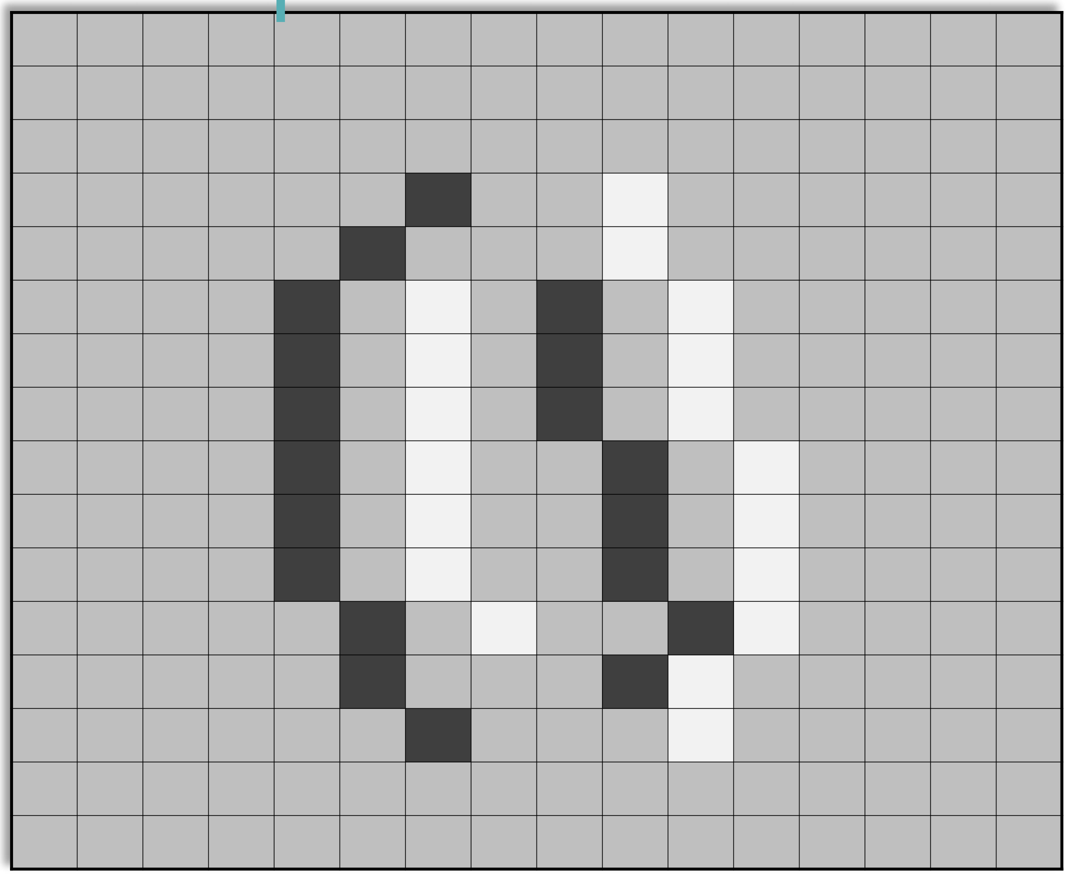


Product of convolution: $\mathbf{h}_{1,1} = g(\mathbf{f}_{1,1} * \mathbf{x} + \mathbf{b}_{1,1})$

Convolution: Example



$\mathbf{x}$



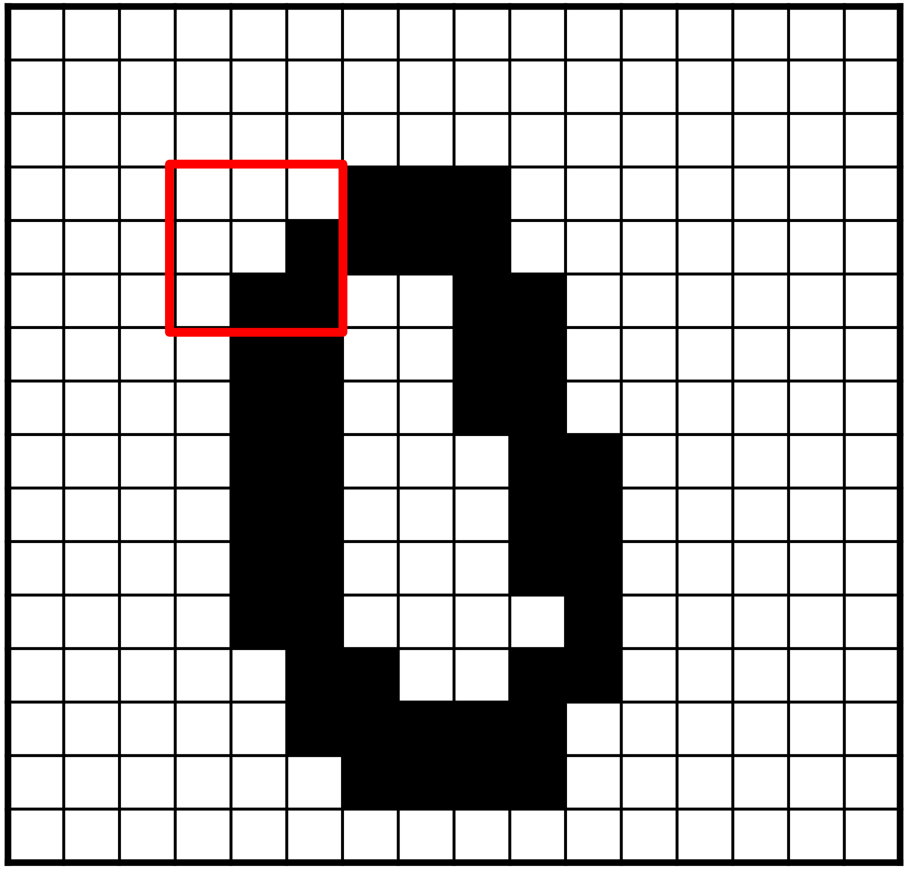
$\mathbf{h}_{1,1}$

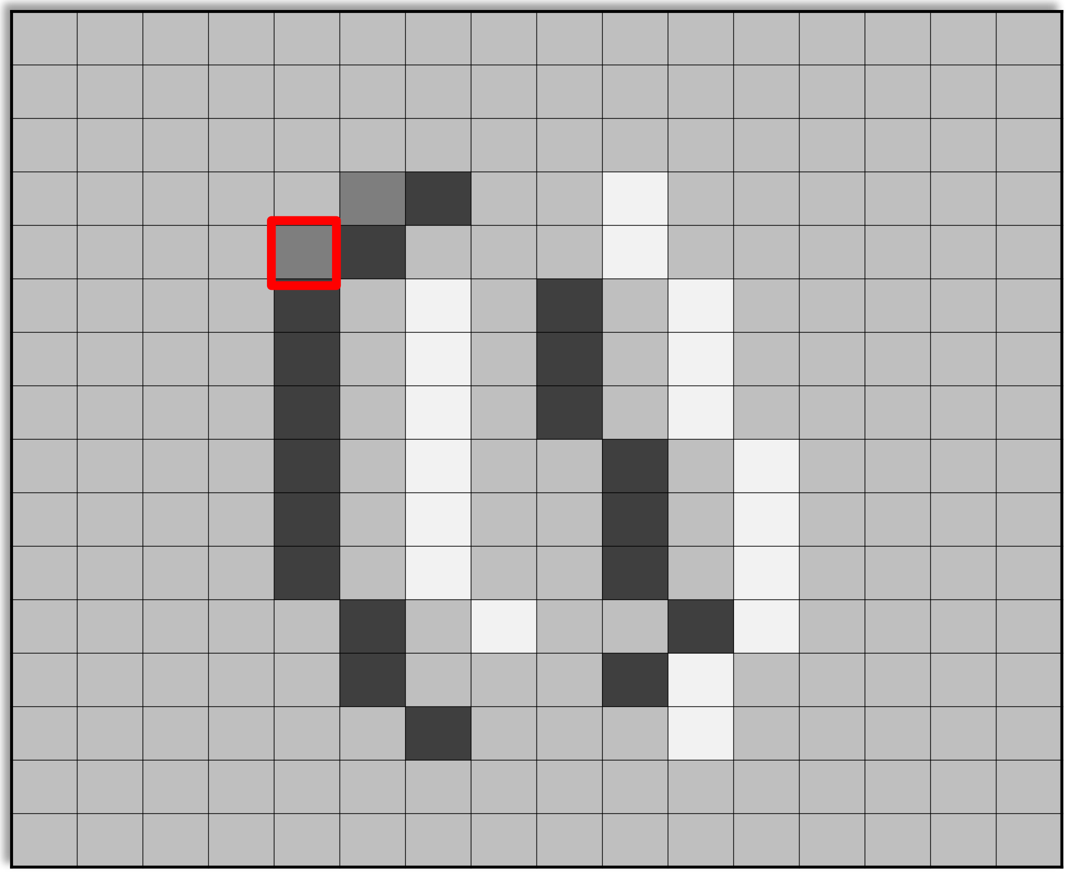
$\mathbf{f}_{1,1} =$

-1	0	+1
-1	0	+1
-1	0	+1

$$\mathbf{h}_{1,1} = g(\mathbf{f}_{1,1} * \mathbf{x} + \mathbf{b}_{1,1})$$

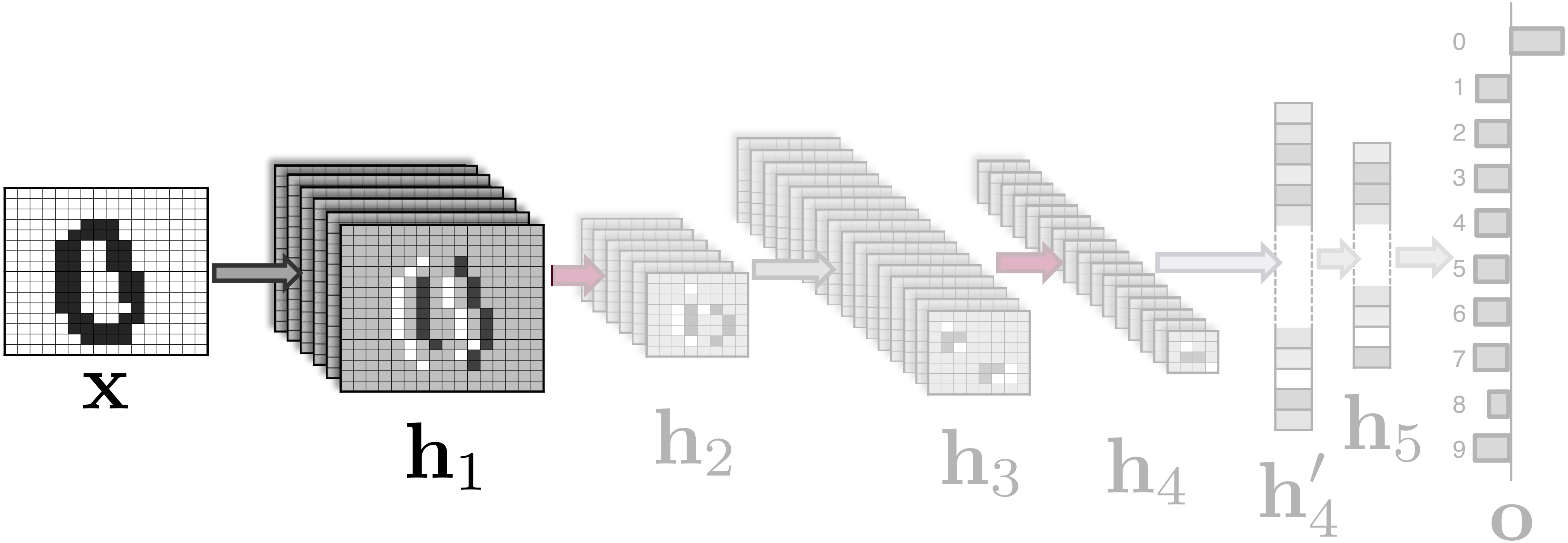
Numerical Example



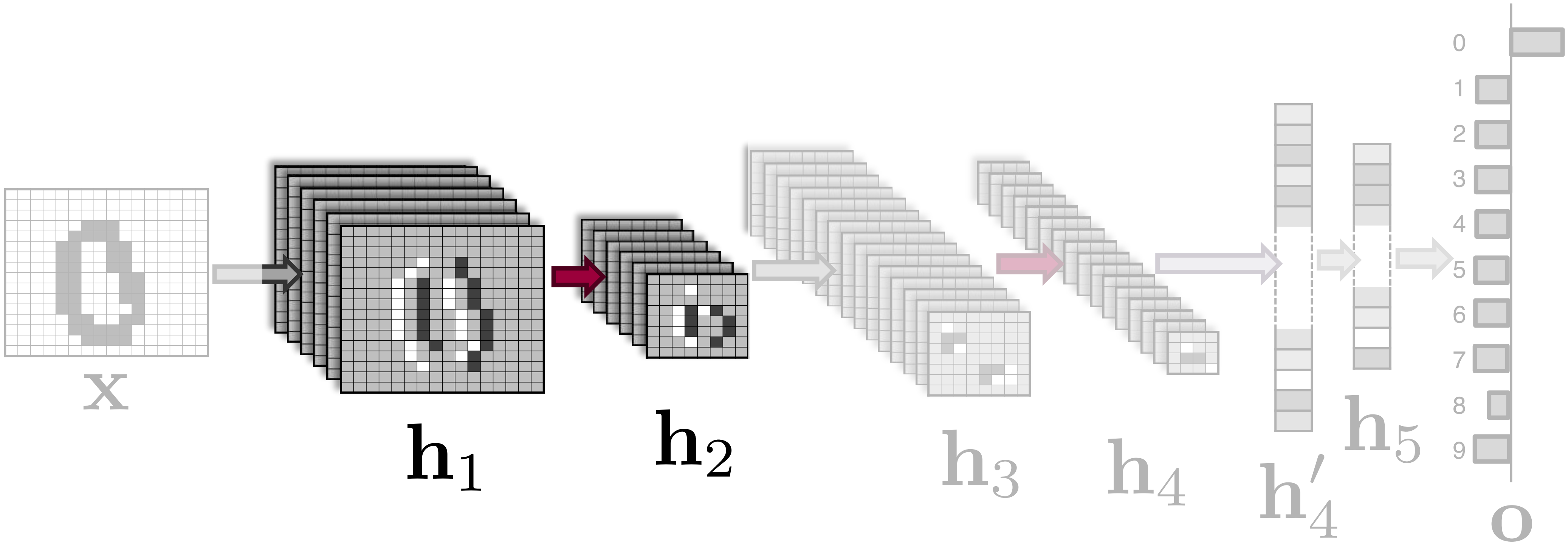


$$\mathbf{f}_{1,1} = \begin{bmatrix} \mathbf{-1} & \mathbf{0} & \mathbf{+1} \\ \mathbf{-1} & \mathbf{0} & \mathbf{+1} \\ \mathbf{-1} & \mathbf{0} & \mathbf{+1} \end{bmatrix}$$

$$\begin{aligned} h &= (-1) \times 255 + 0 \times 255 + (+1) \times 255 + \\ &\quad (-1) \times 255 + 0 \times 255 + (+1) \times 0 + \\ &\quad (-1) \times 255 + 0 \times 0 + (+1) \times 0 \\ &= -255 + 0 + 255 \\ &\quad -255 + 0 + 0 \\ &\quad -255 + 0 + 0 \\ &= -510 \end{aligned}$$

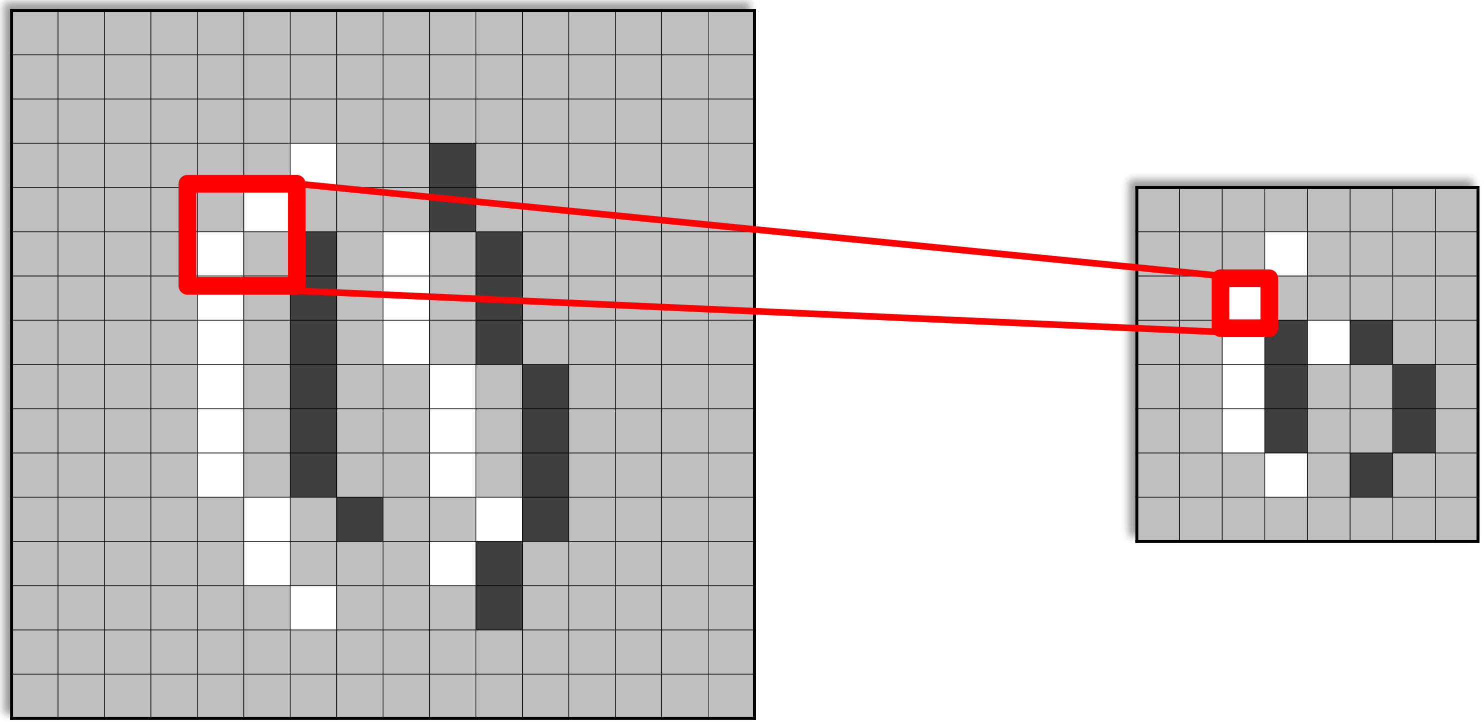


$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})]$$



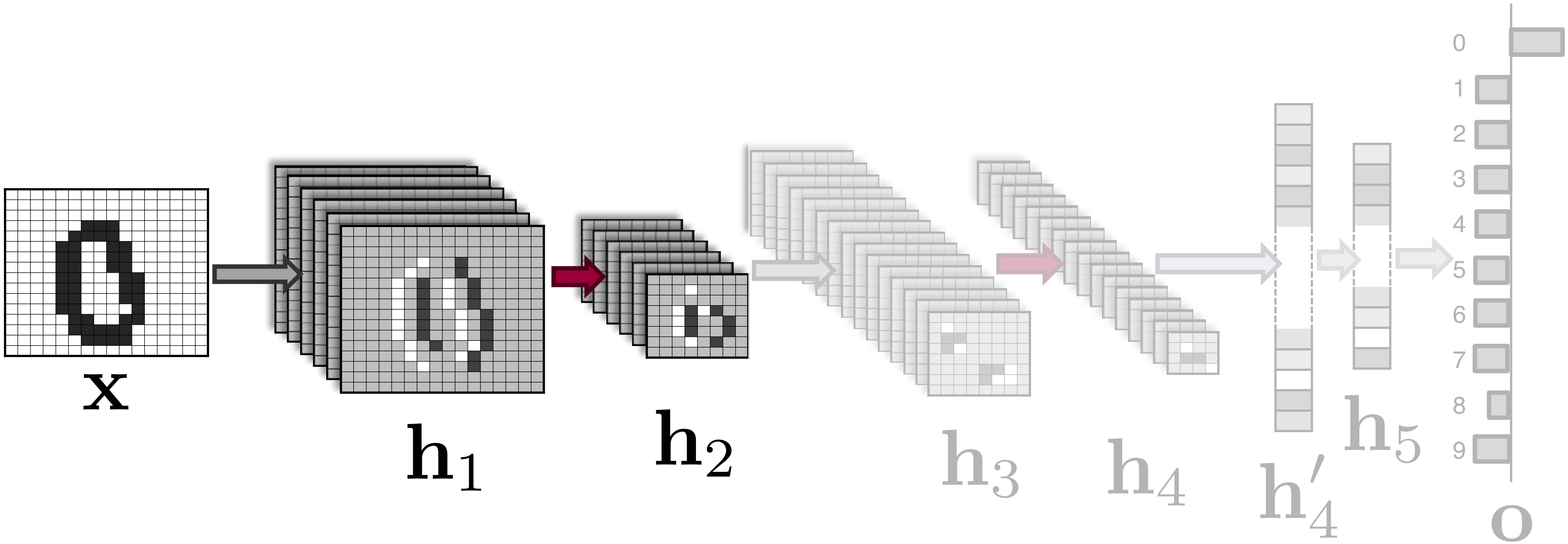
x
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{h}_4$
$\mathbf{h}'_4$
$\mathbf{h}_5$
0
1
2
3
4
5
6
7
8
9
o

Subsampling / Pooling



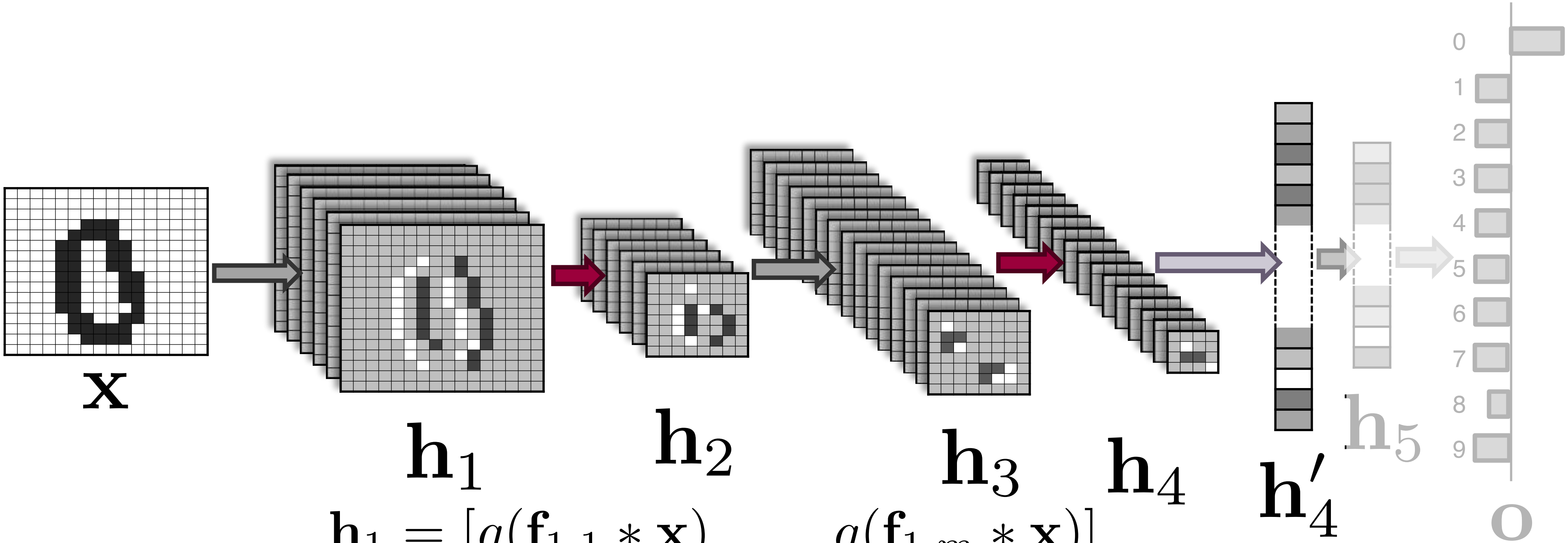
For example, max-pooling:

$$\mathbf{h}_i[u, v] = \max\{ \begin{array}{ll} \mathbf{h}_{i-1}[2u, & 2v], \\ \mathbf{h}_{i-1}[2u, & 2v+1], \\ \mathbf{h}_{i-1}[2u+1, & 2v], \\ \mathbf{h}_{i-1}[2u+1, & 2v+1] \end{array} \}$$

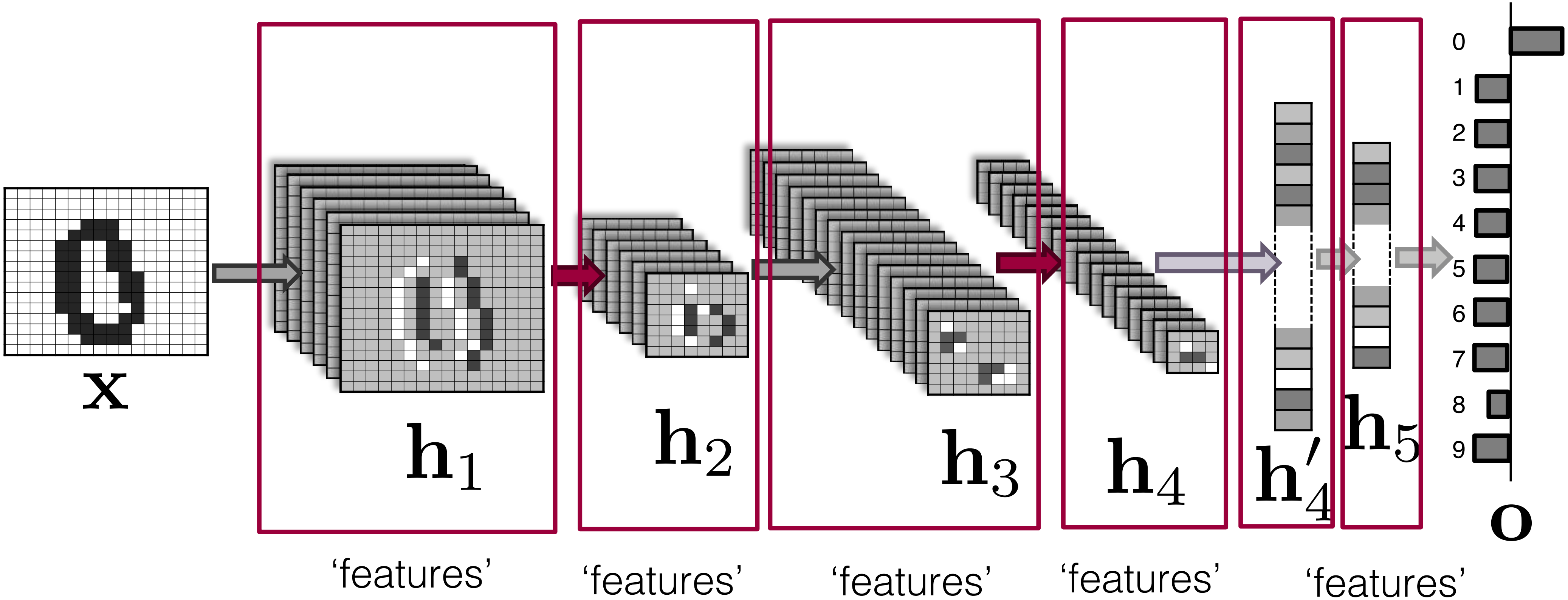


$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})]$$
$$\mathbf{h}_2 = \text{pooling}(\mathbf{h}_1)$$

Inspired by the theory of Hubel and Wiesel on V1
(Nobel prize in 1981)

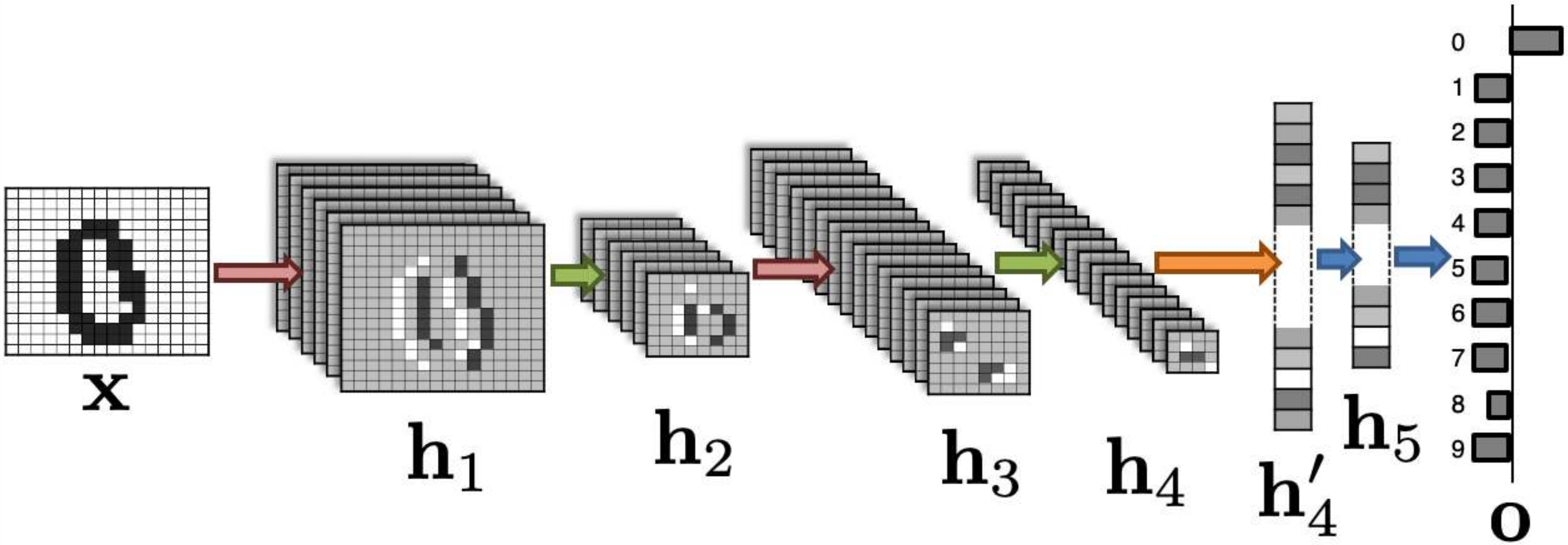


$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})]$$
$$\mathbf{h}_2 = \text{pooling}(\mathbf{h}_1)$$
$$\mathbf{h}_3 = [g(\mathbf{f}_{3,1} * \mathbf{h}_2), \ldots, g(\mathbf{f}_{3,n} * \mathbf{h}_2)]$$
$$\mathbf{h}_4 = \text{pooling}(\mathbf{h}_3)$$
$$\mathbf{h}'_4 = \text{Vec}(\mathbf{h}_4)$$



First features may be 'interpretated', but the rest is usually more difficult.

How Can We Find the Network's Parameters?

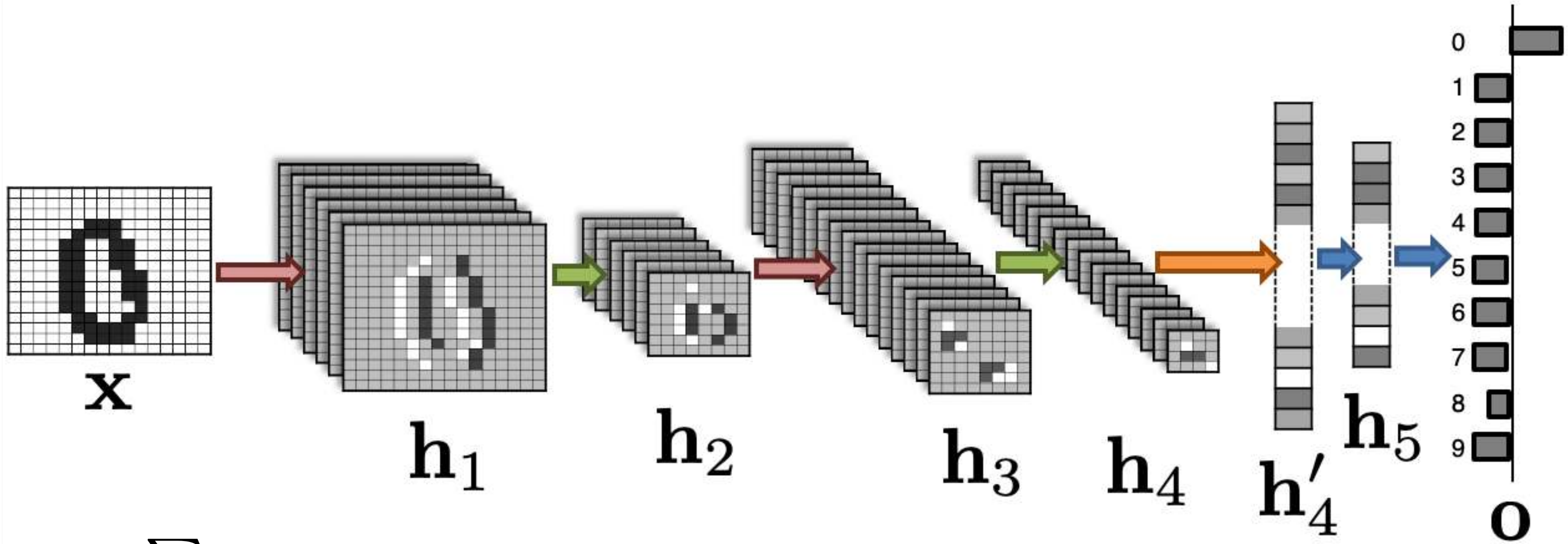


$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})]$$
$$\mathbf{h}_2 = \text{pooling}(\mathbf{h}_1)$$
$$\mathbf{h}_3 = [g(\mathbf{f}_{3,1} * \mathbf{h}_2), \ldots, g(\mathbf{f}_{3,n} * \mathbf{h}_2)]$$
$$\mathbf{h}_4 = \text{pooling}(\mathbf{h}_3)$$
$$\mathbf{h}'_4 = \text{Vec}(\mathbf{h}_4)$$
$$\mathbf{h}_5 = g(\mathbf{W}_5\mathbf{h}'_4 + \mathbf{b}_5)$$
$$\mathbf{o} = \mathbf{W}_6\mathbf{h}_5 + \mathbf{b}_6$$

$$\mathcal{L}(\Theta) = \sum_i L(c_i, f(\mathbf{x}_i;\ \Theta))$$
$$\Theta = (\mathbf{f}_{1,1}, \ldots, \mathbf{f}_{1,m}, \ldots, \mathbf{W}_5, \mathbf{b}_5, \mathbf{W}_6, \mathbf{b}6)$$

OPTIMIZING / TRAINING A DEEP NETWORK

How Can We Find the Network's Parameters?



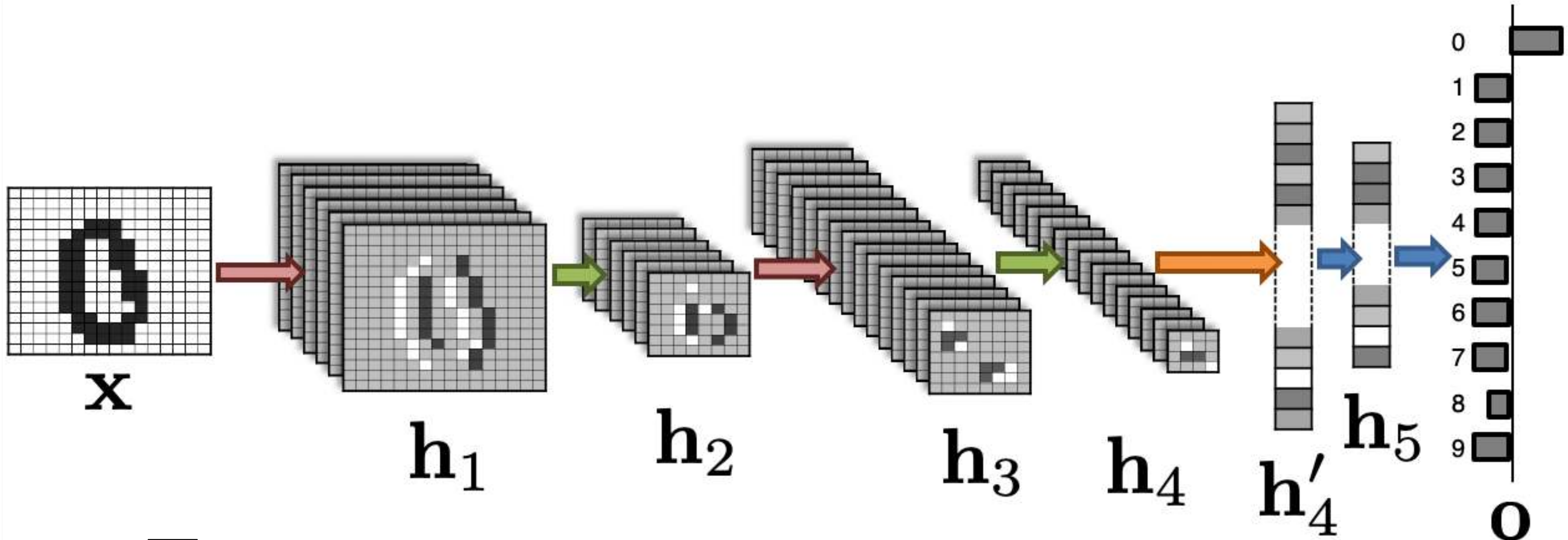
$$\mathcal{L}(\Theta) = \sum_i L(c_i, f(\mathbf{x}_i;\ \Theta))$$

$$\Theta = (\mathbf{f}_{1,1}, \ldots, \mathbf{f}_{1,m}, \ldots, \mathbf{W}_5, \mathbf{b}_5, \mathbf{W}_6, \mathbf{b}6)$$

As before:

- Initialize Θ randomly;
- Optimize Θ using gradient descent.

backpropagation

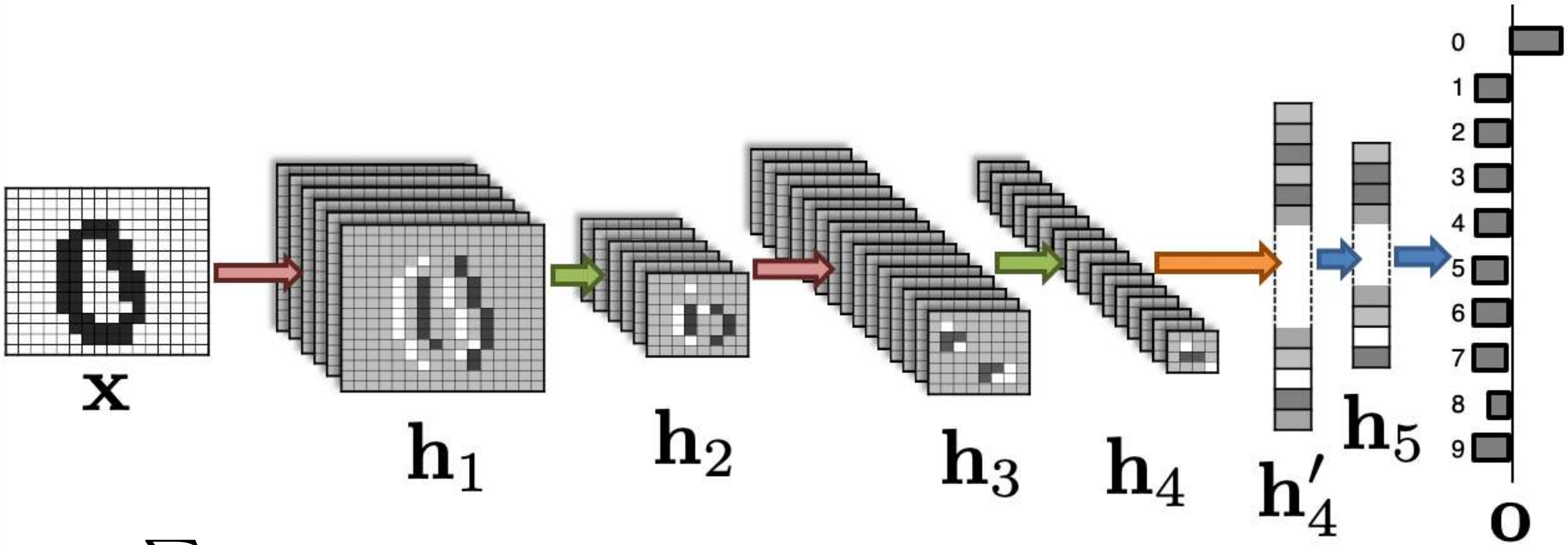


$$\mathcal{L}(\Theta) = \sum_i L(c_i, f(\mathbf{x}_i;\ \Theta))$$

$$\Theta = (\mathbf{f}_{1,1}, \ldots, \mathbf{f}_{1,m}, \ldots, \mathbf{W}_5, \mathbf{b}_5, \mathbf{W}_6, \mathbf{b6})$$

Back-propagation: An efficient method to compute the gradient of the objective function;

backpropagation



$$\mathcal{L}(\Theta) = \sum_i L(c_i, f(\mathbf{x}_i;\ \Theta))$$

$$\Theta = (\mathbf{f}_{1,1}, \ldots, \mathbf{f}_{1,m}, \ldots, \mathbf{W}_5, \mathbf{b}_5, \mathbf{W}_6, \mathbf{b6})$$

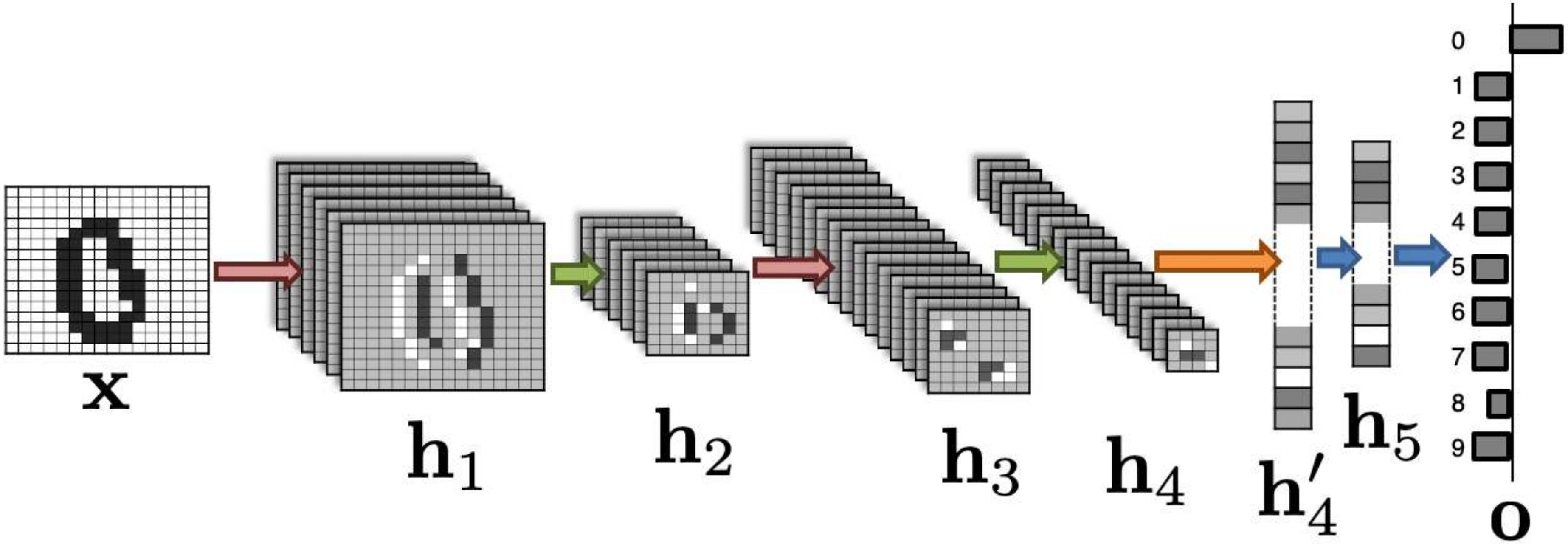
Several variants of gradient descent have been developed recently (Adam, RMSProp).

Python Libraries

TensorFlow (Google), Keras (Google – higher level), PyTorch (Facebook), ..

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
from keras.layers import Flatten
model.add(Flatten())

from keras.layers import Dense
model.add(Dense(128, activation='relu'))

model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X_train, Y_train, batch_size=32, epochs=10, verbose=1)
```

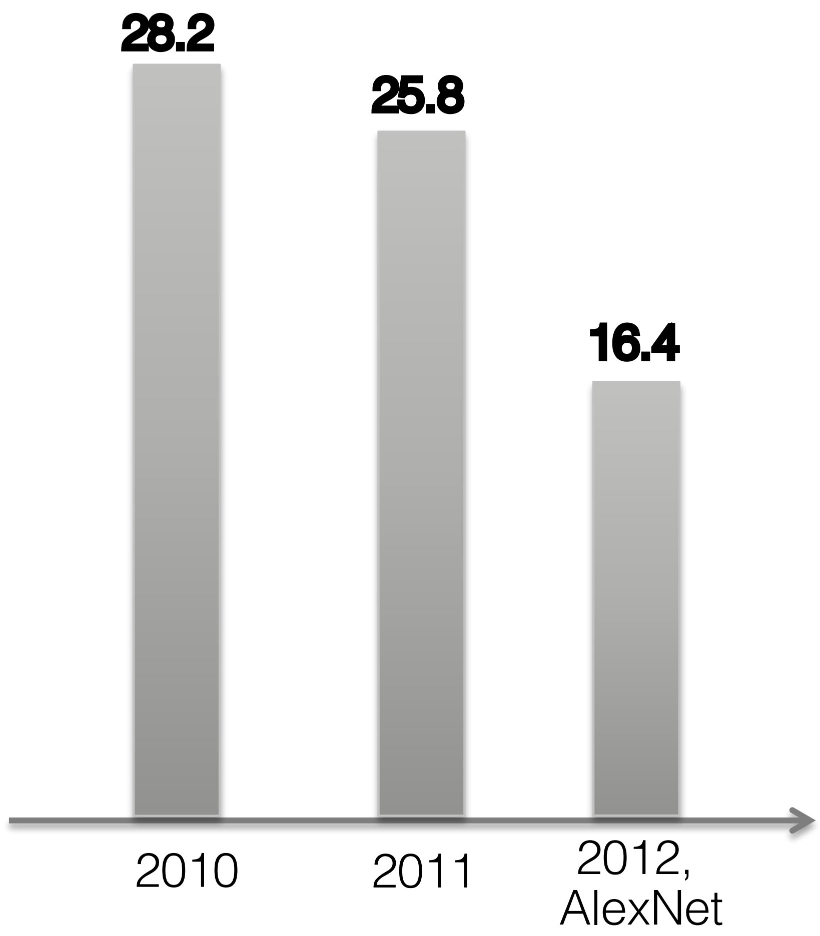
How to Choose the Number of Layers?
The Number of Filters per Layer?
The Sizes of the Filters?



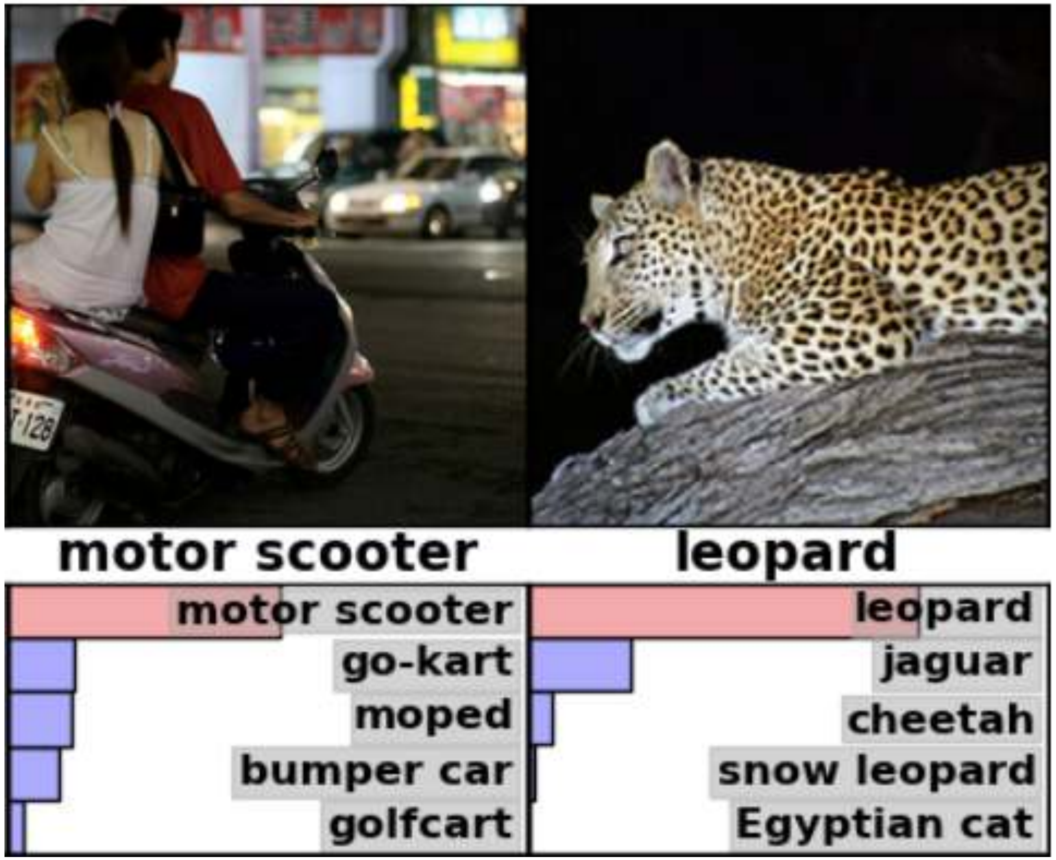
- experience;
- trial-and-error;
- Auto-ML: automated methods to find a good architecture.

MORE RECENT RESULTS

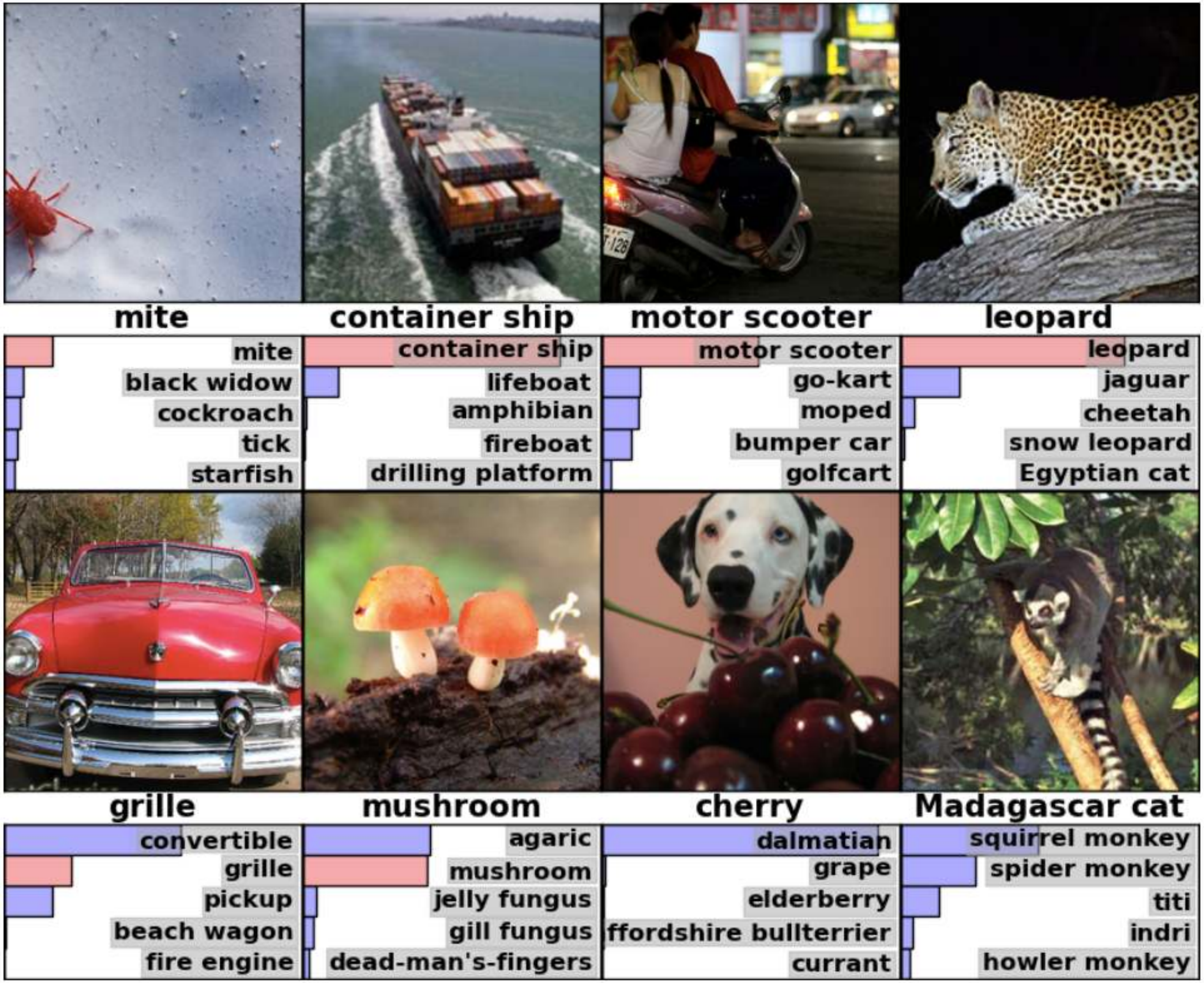
AlexNet (2010)



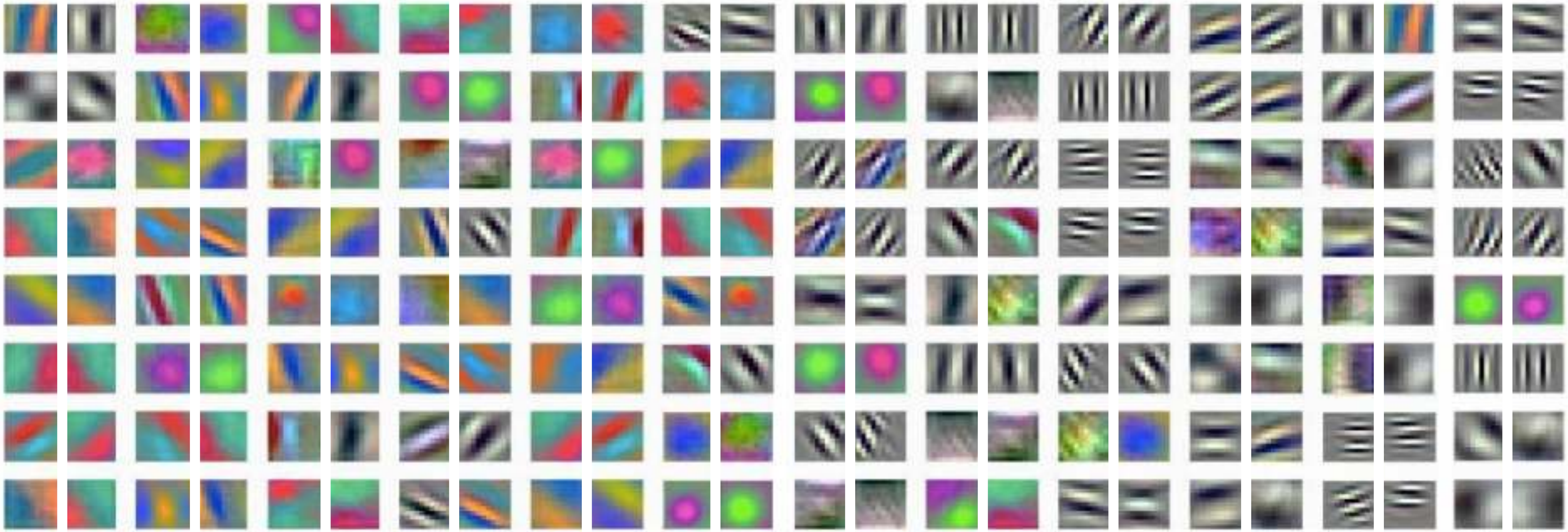
Classification task on ImageNet challenge top-5 error.



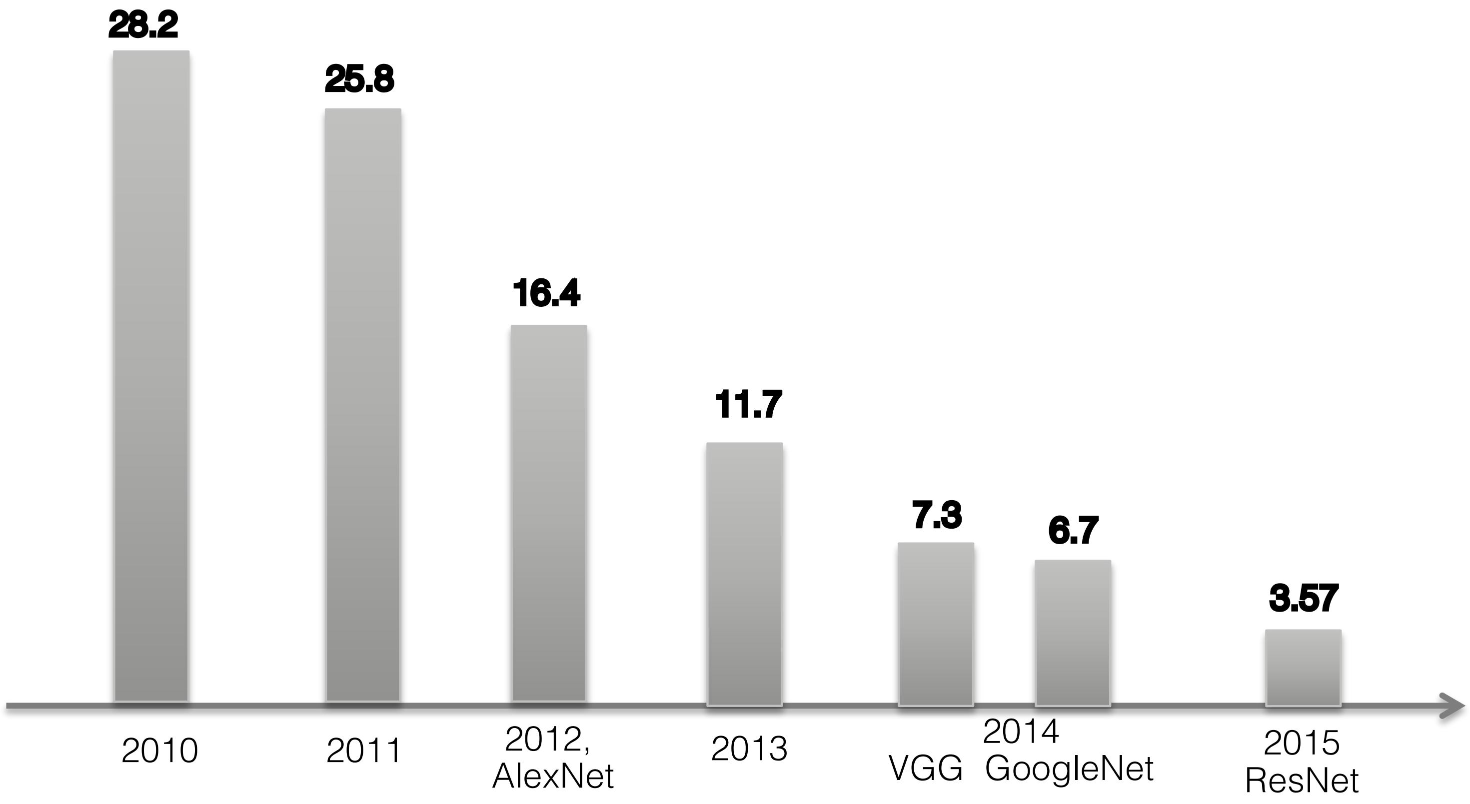
AlexNet: Some results on Imagenet



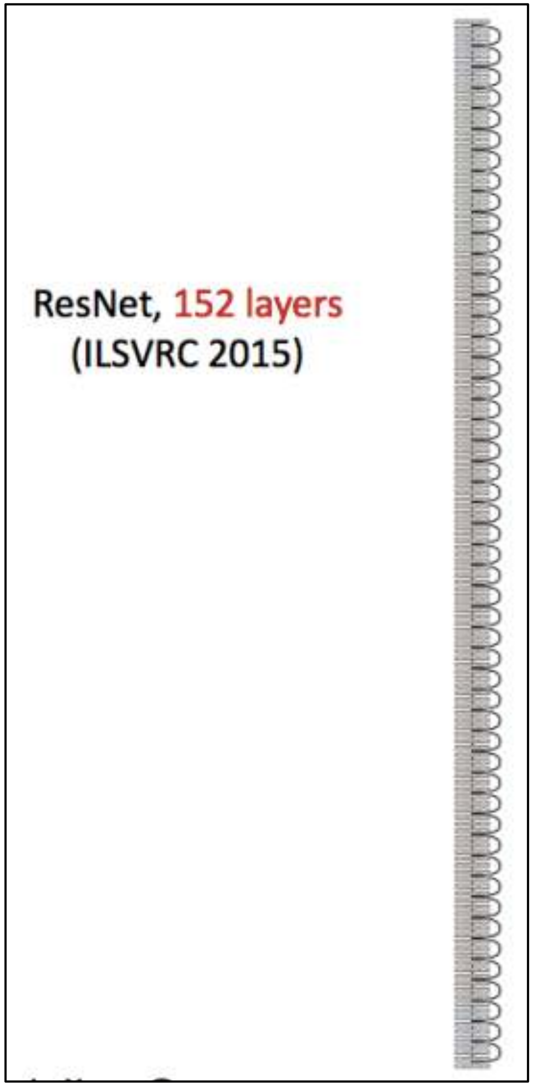
AlexNet: Learned Filters for the First Layer



Classification task on ImageNet challenge top-5 error

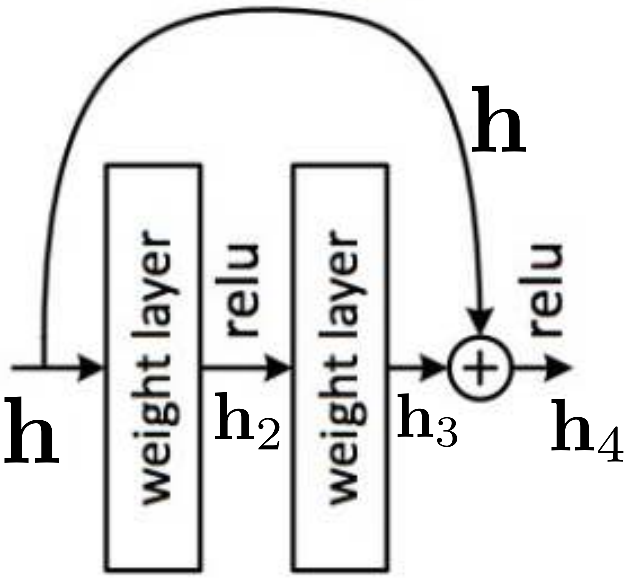


ResNet [He et al, CVPR 2016] ILSVRC 2015 Winner



The Residual Module (resnet block)

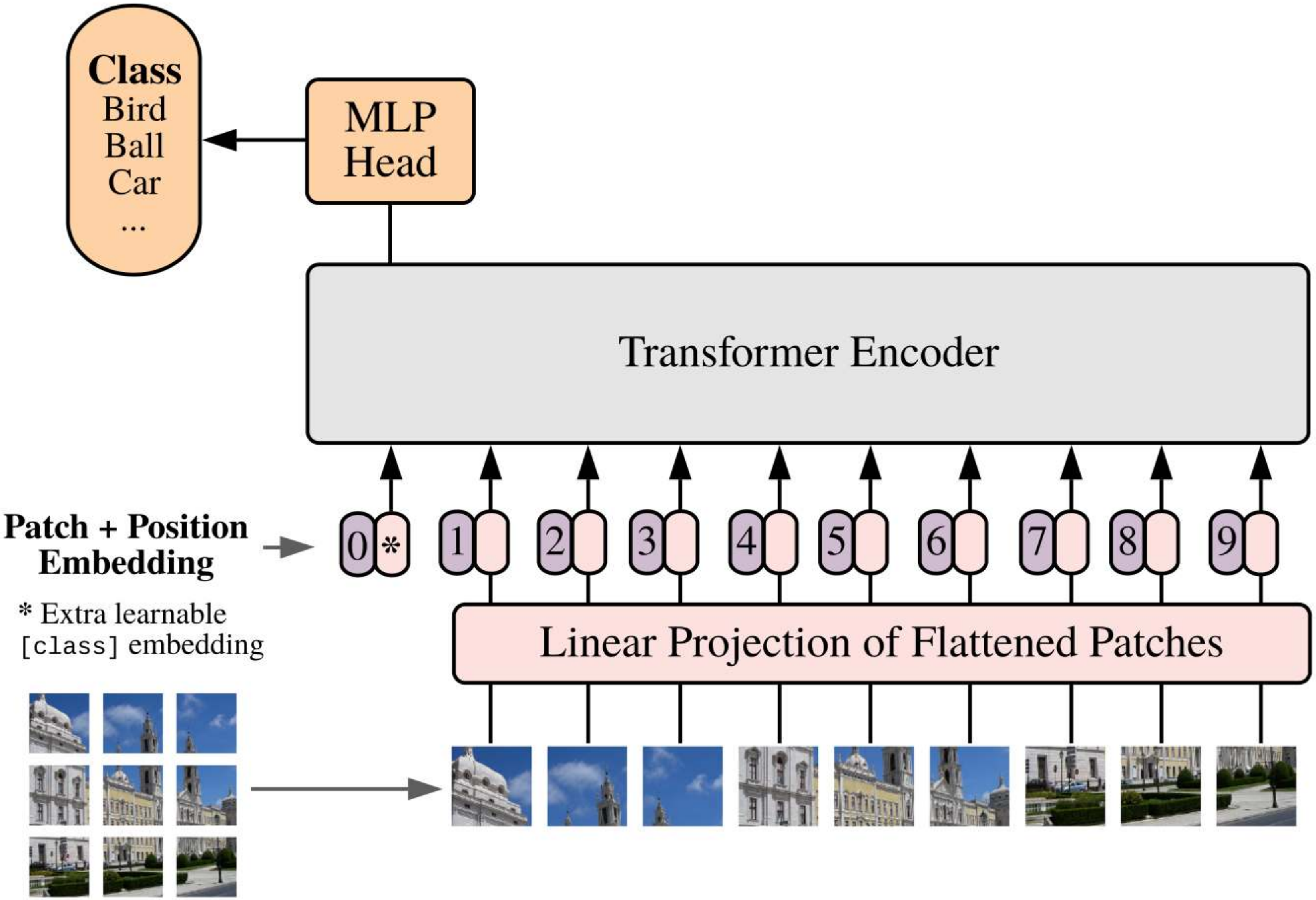
Uses 'skip' or 'shortcut' connections:



$$\mathbf{h}_2 = g(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$
$$\mathbf{h}_3 = \mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3$$
$$\mathbf{h}_4 = g(\mathbf{h}_3 + \mathbf{h})$$

- Makes it easy for network layers to represent the identity mapping;
- Limits vanishing and exploding gradients.

Transformers



BEING CAREFUL ABOUT WHAT A DEEP NETWORK REALLY LEARNS

recognizing objects

recognizing objects

(a) Output prediction on original images.

(b) Prediction when foreground is whitened.

recognizing objects

(a) Output prediction on original images.

(b) Prediction when foreground is whitened.

some failures

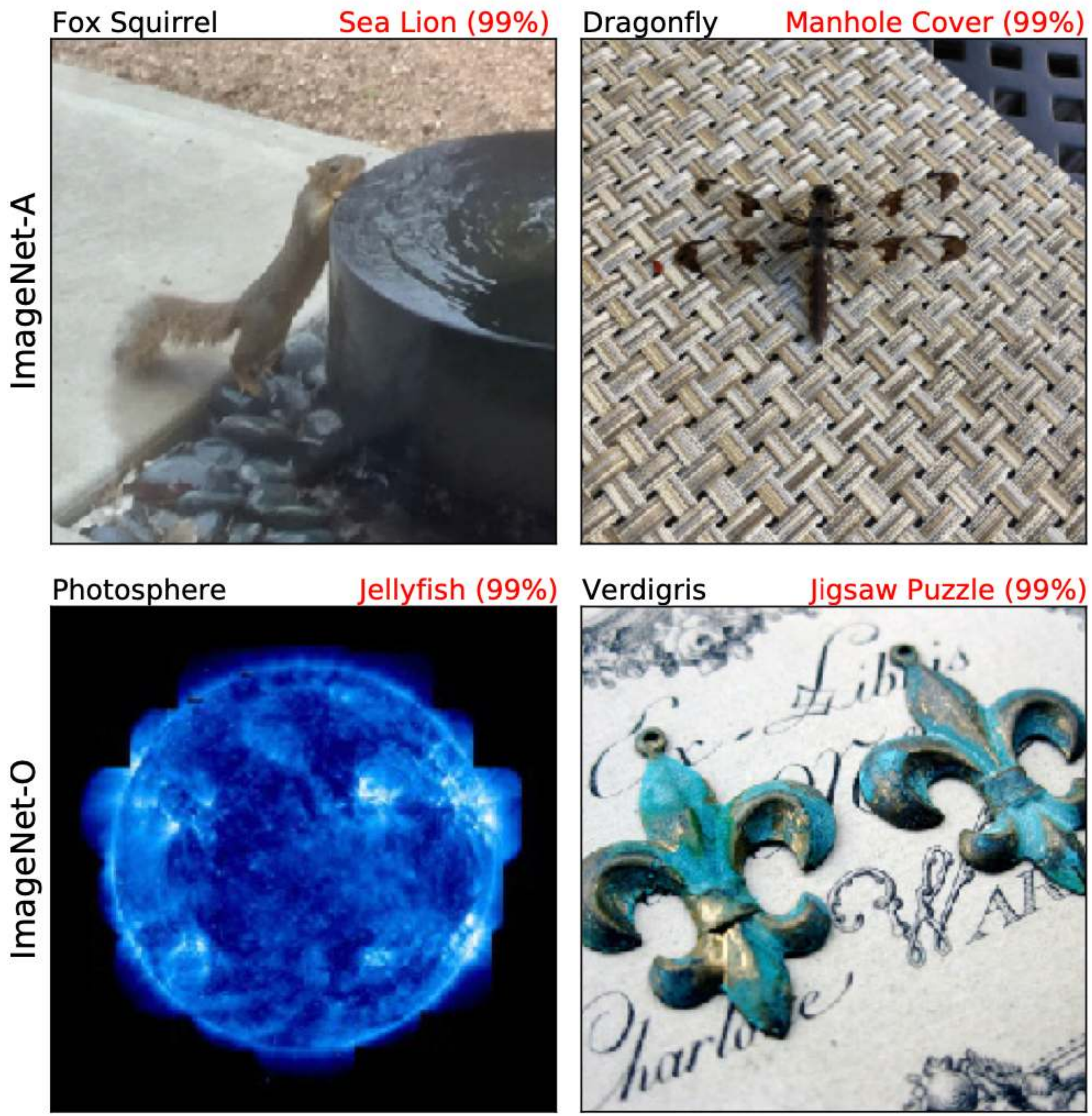
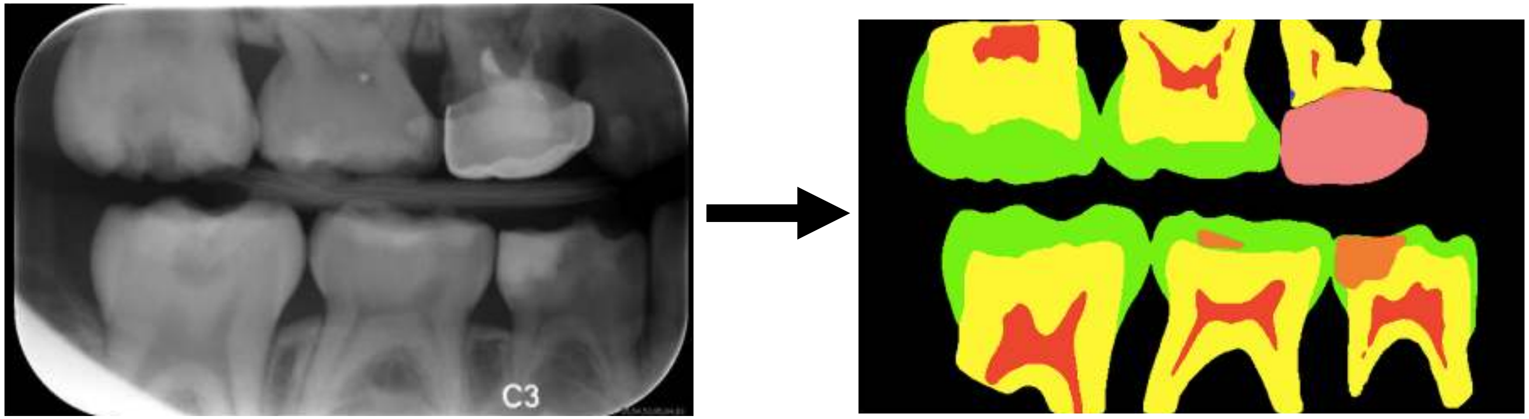
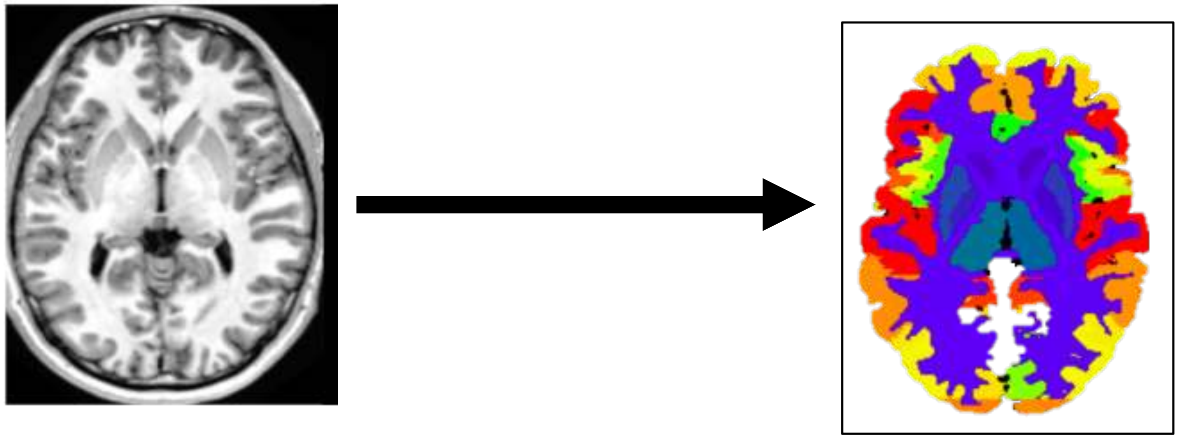


IMAGE SEGMENTATION

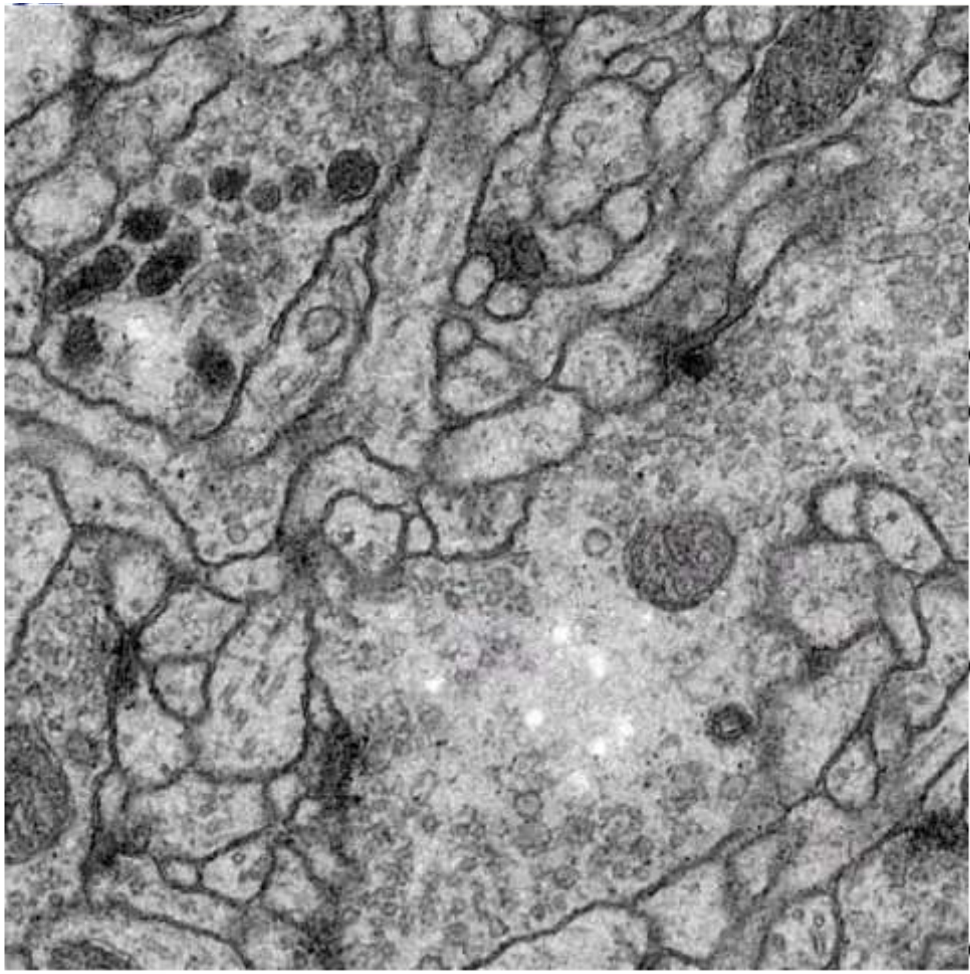
image segmentation

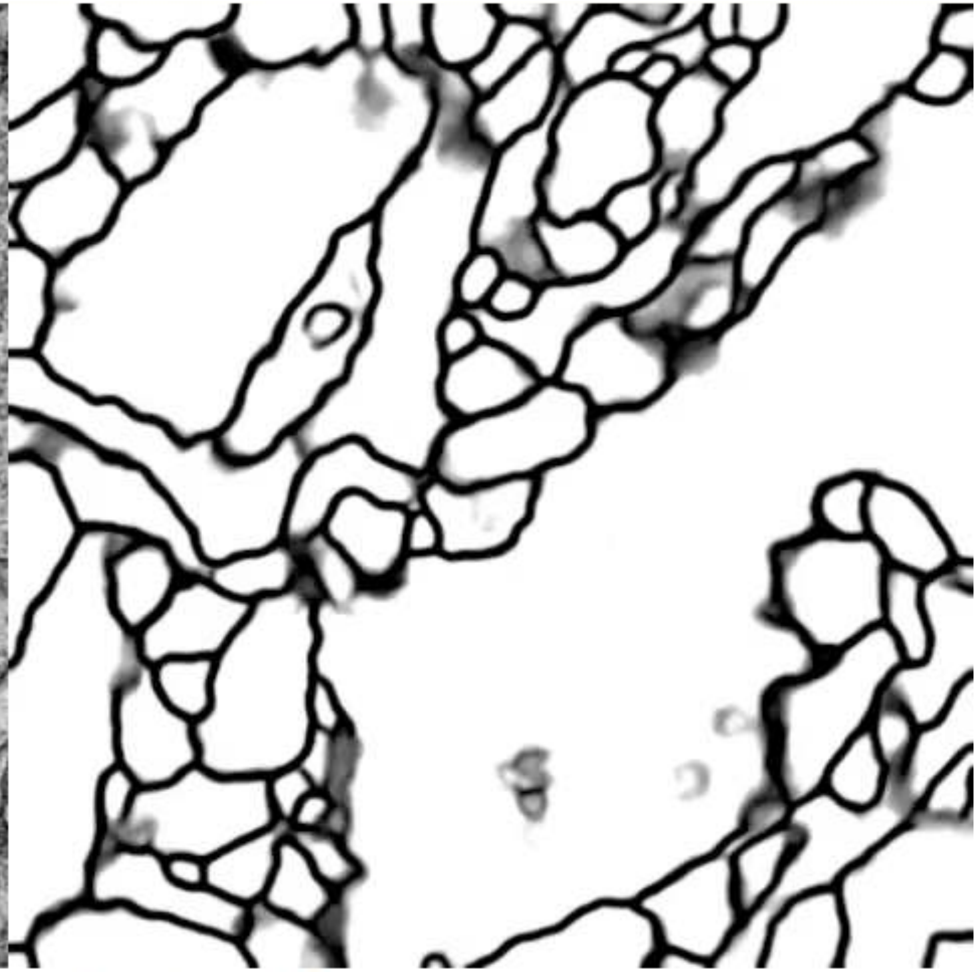
these can be seen as 1 classification problem for each pixel





U-Net: Results

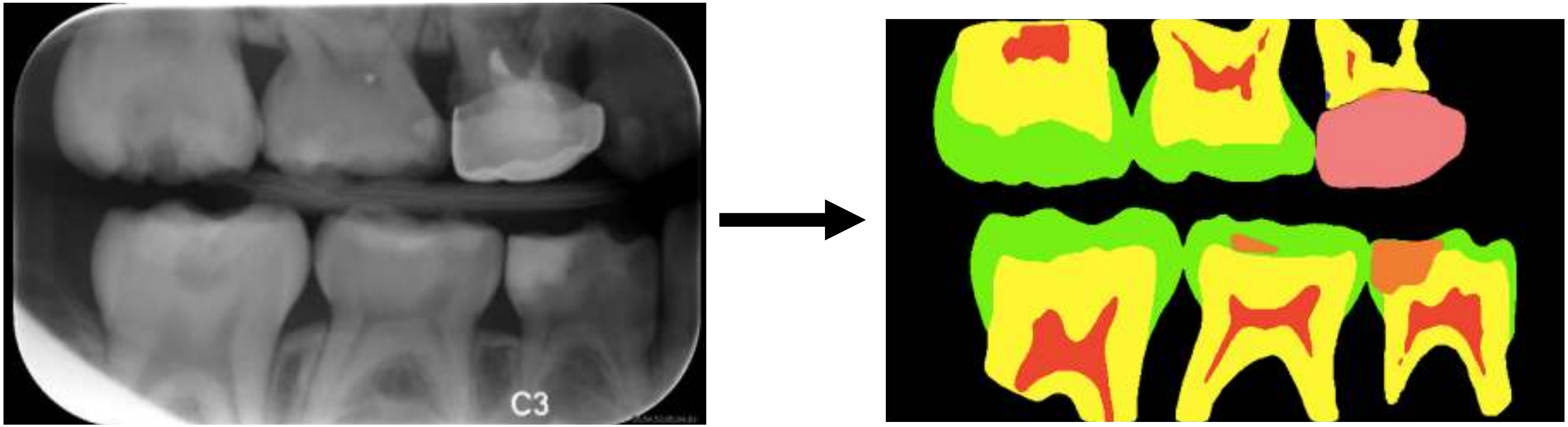




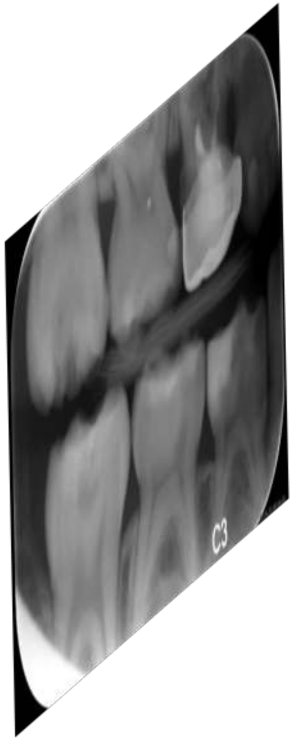
Input image

Our result: 0.000353 warping error
(**New best score** at submission march 6th, 2015)
Sliding-window CNN: 0.000420
Training time: 10h, Application: 1s per image

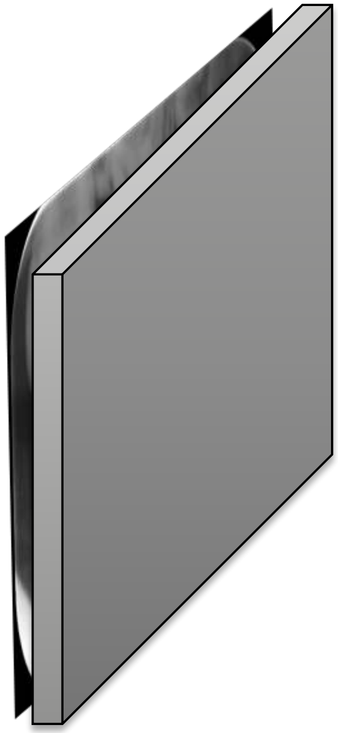
image segmentation



U-Net: Architecture

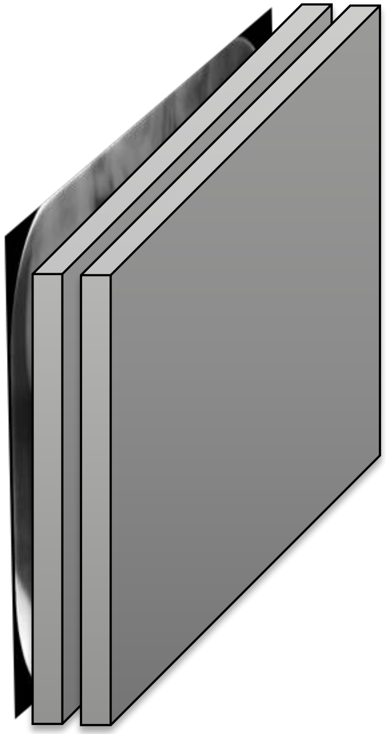


U-Net: Architecture



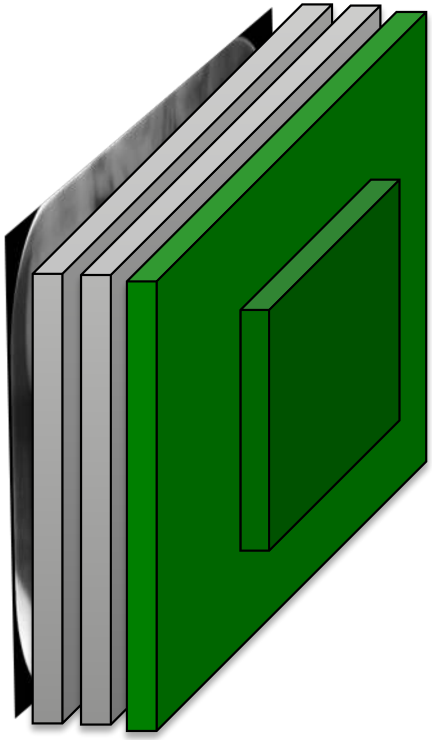
$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})]$$

U-Net: Architecture



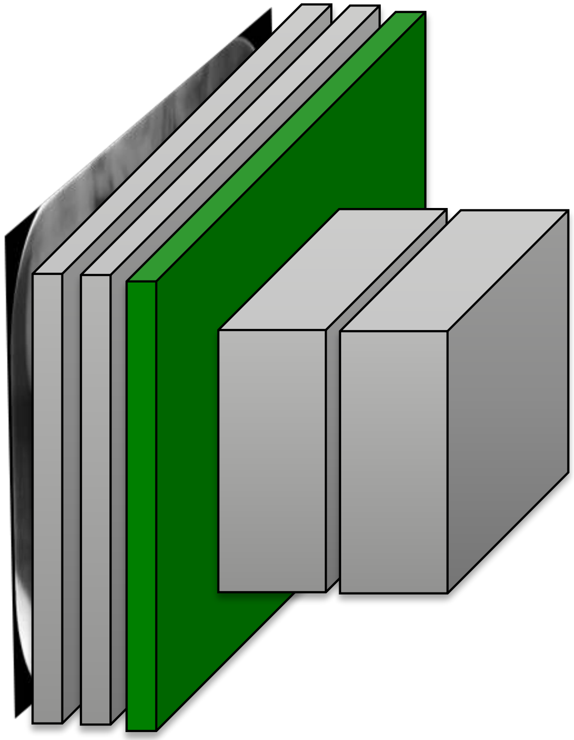
$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})]$$
$$\mathbf{h}_2 = [g(\mathbf{f}_{2,1} * \mathbf{h}_1), \ldots, g(\mathbf{f}_{2,m_2} * \mathbf{h}_1)]$$

U-Net: Architecture



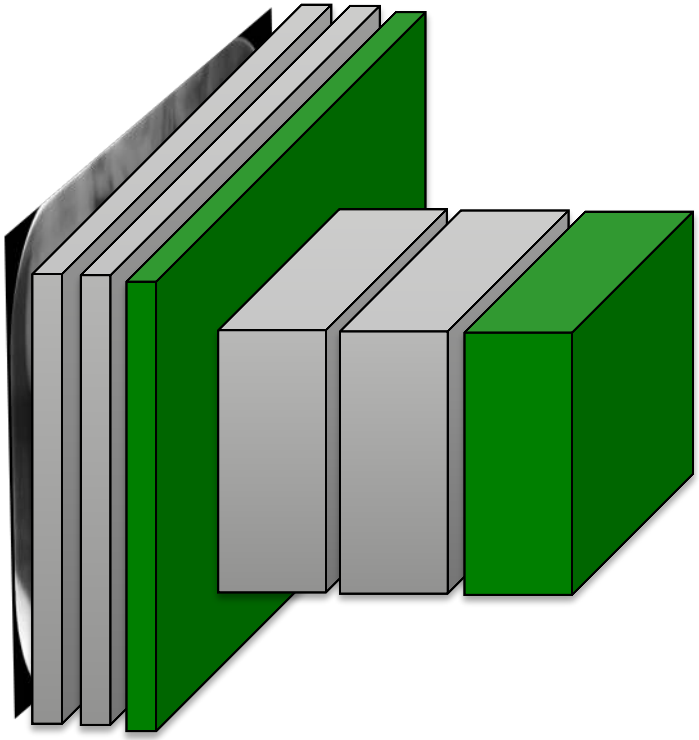
$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})]$$
$$\mathbf{h}_2 = [g(\mathbf{f}_{2,1} * \mathbf{h}_1), \ldots, g(\mathbf{f}_{2,m_2} * \mathbf{h}_1)]$$
$$\mathbf{h}_3 = \text{pooling}(\mathbf{h}_2)$$

U-Net: Architecture



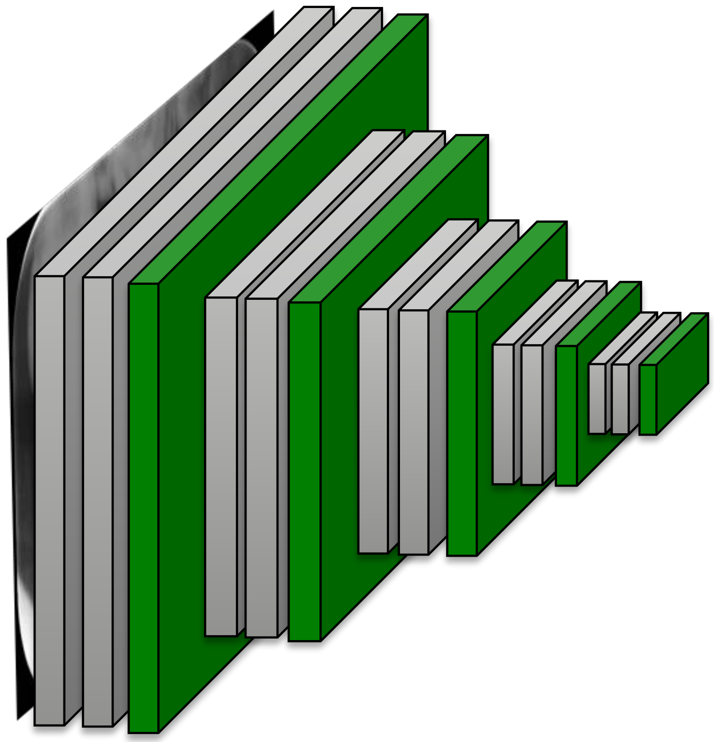
$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})]$$
$$\mathbf{h}_2 = [g(\mathbf{f}_{2,1} * \mathbf{h}_1), \ldots, g(\mathbf{f}_{2,m_2} * \mathbf{h}_1)]$$
$$\mathbf{h}_3 = \text{pooling}(\mathbf{h}_2)$$
$$\mathbf{h}_4 = [g(\mathbf{f}_{4,1} * \mathbf{h}_3), \ldots, g(\mathbf{f}_{4,m_4} * \mathbf{h}_3)]$$

U-Net: Architecture



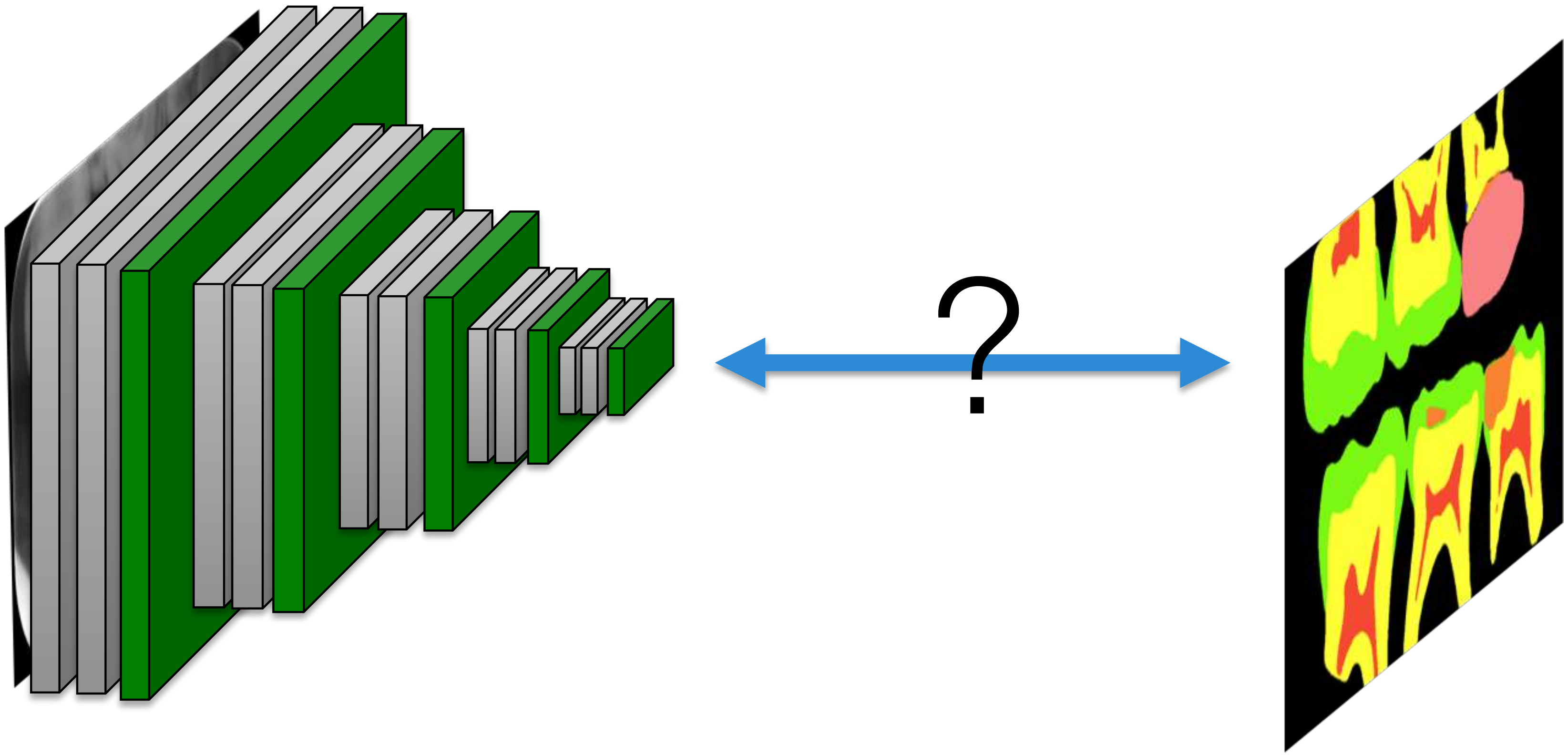
$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{f}_{1,m} * \mathbf{x})]$$
$$\mathbf{h}_2 = [g(\mathbf{f}_{2,1} * \mathbf{h}_1), \ldots, g(\mathbf{f}_{2,m_2} * \mathbf{h}_1)]$$
$$\mathbf{h}_3 = \text{pooling}(\mathbf{h}_2)$$
$$\mathbf{h}_4 = [g(\mathbf{f}_{4,1} * \mathbf{h}_3), \ldots, g(\mathbf{f}_{4,m_4} * \mathbf{h}_3)]$$
$$\mathbf{h}_5 = [g(\mathbf{f}_{5,1} * \mathbf{h}_4), \ldots, g(\mathbf{f}_{5,m_5} * \mathbf{h}_4)]$$
$$\mathbf{h}_6 = \text{pooling}(\mathbf{h}_5)$$

U-Net: Architecture

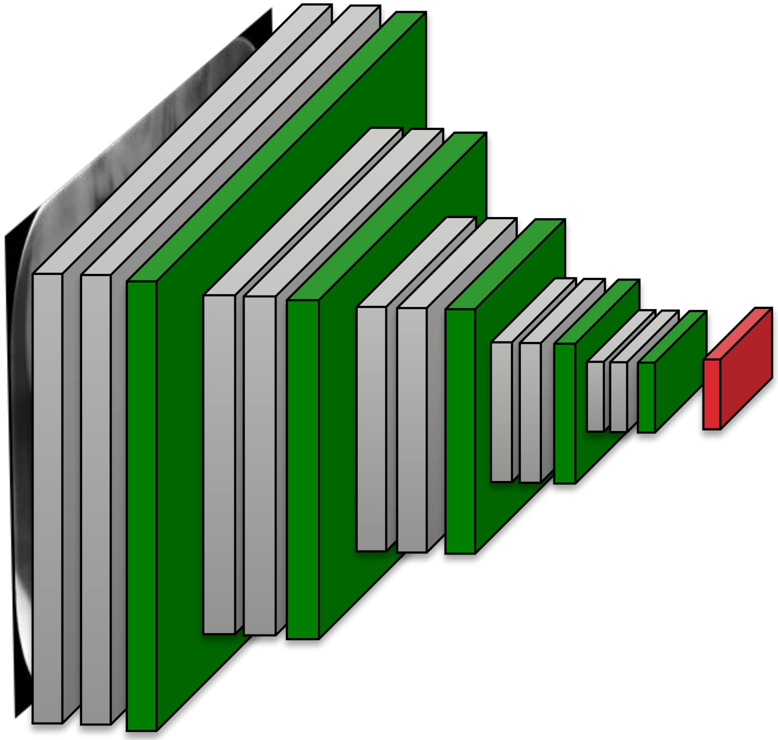


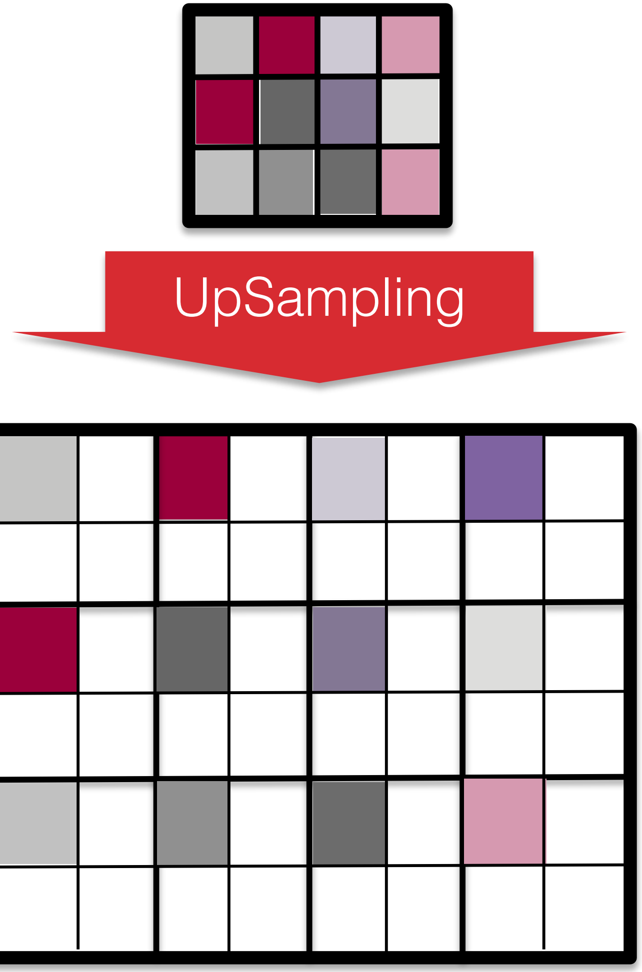
$$\mathbf{h}_{13} = [g(\mathbf{f}_{13,1} * \mathbf{h}_{12}), \ldots, g(\mathbf{f}_{13,m_{13}} * \mathbf{h}_{12})]$$
$$\mathbf{h}_{14} = [g(\mathbf{f}_{14,1} * \mathbf{h}_{13}), \ldots, g(\mathbf{f}_{14,m_{14}} * \mathbf{h}_{13})]$$
$$\mathbf{h}_{15} = \text{pooling}(\mathbf{h}_{14})$$

U-Net: Architecture



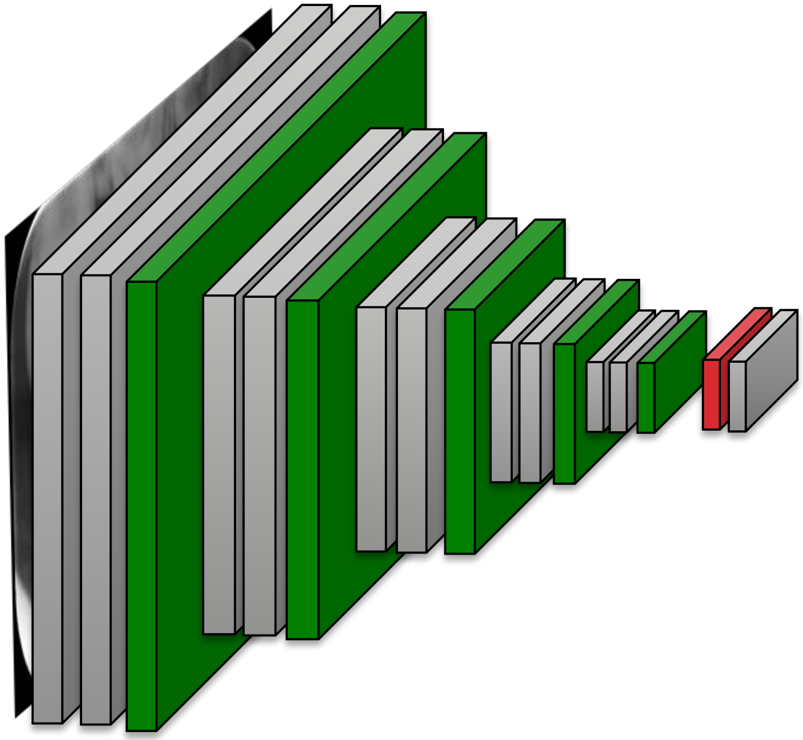
U-Net: Architecture

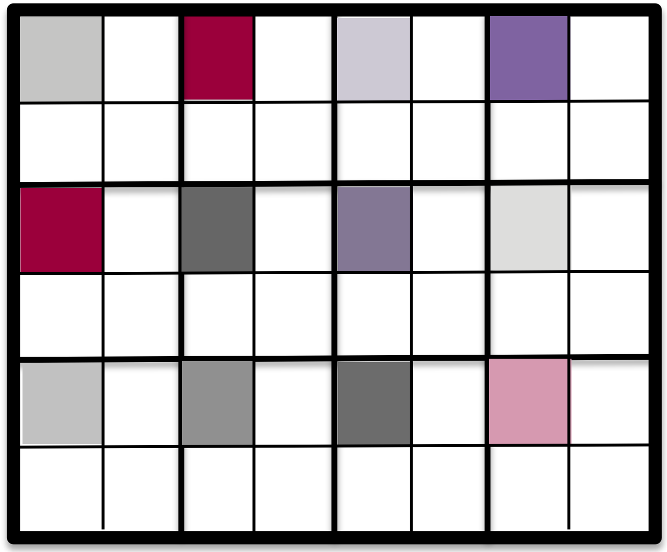




$$\mathbf{h}_{16} = \text{UpSampling}(\mathbf{h}_{15})$$

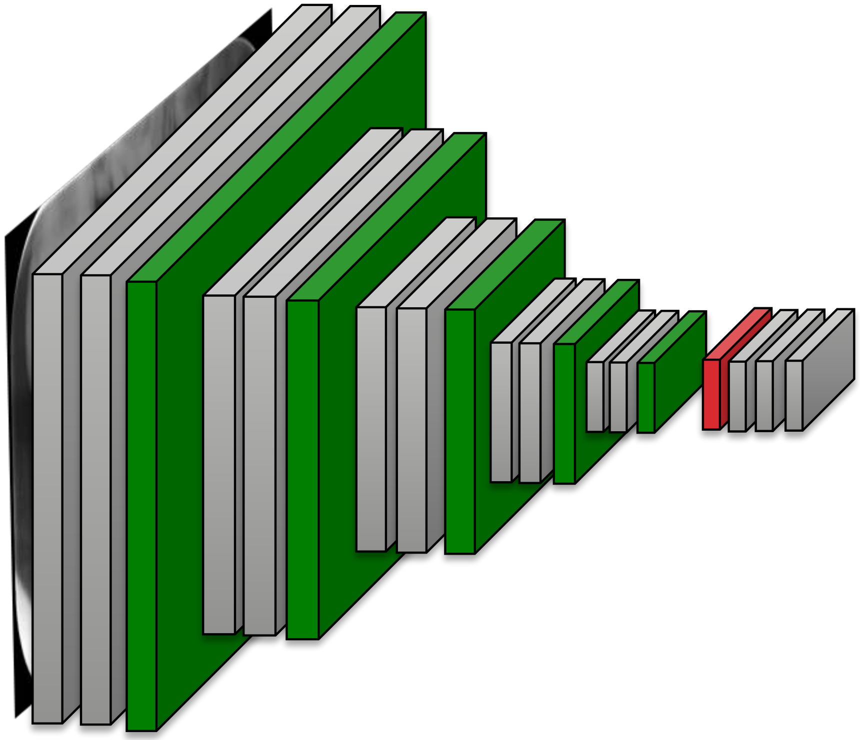
U-Net: Architecture





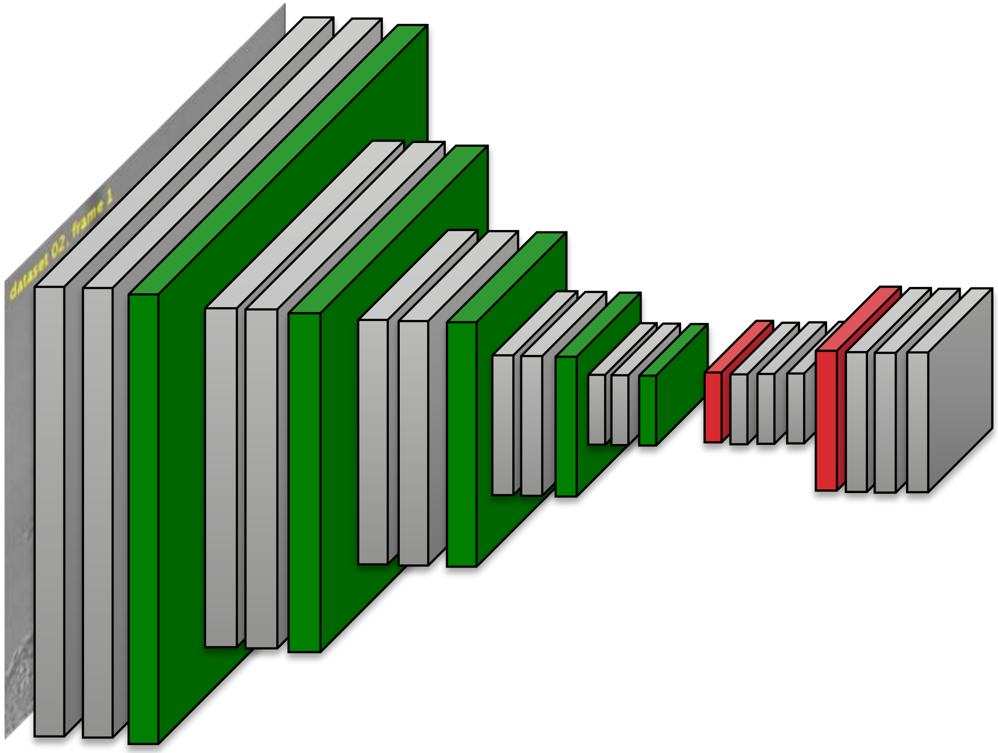
$$\mathbf{h}_{16} = \text{UpSampling}(\mathbf{h}_{15})$$
$$\mathbf{h}_{17} = [g(\mathbf{f}_{17,1} * \mathbf{h}_{16}), \ldots, g(\mathbf{f}_{17,m_{17}} * \mathbf{h}_{16})]$$

U-Net: Architecture

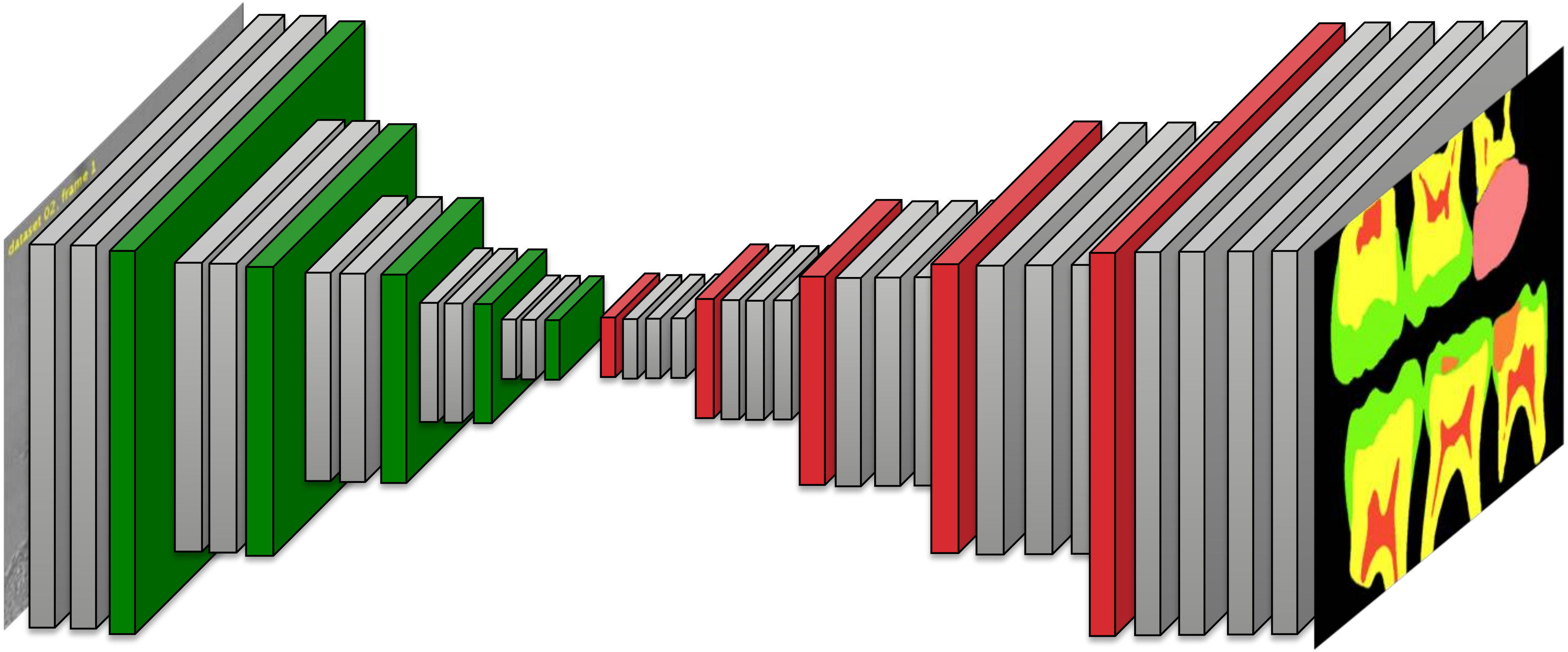


$$\mathbf{h}_{16} = \mathrm{UpSampling}(\mathbf{h}_{15})$$
$$\mathbf{h}_{17} = [g(\mathbf{f}_{17,1} * \mathbf{h}_{16}), \ldots, g(\mathbf{f}_{17,m_{17}} * \mathbf{h}_{16})]$$
$$\mathbf{h}_{18} = [g(\mathbf{f}_{18,1} * \mathbf{h}_{17}), \ldots, g(\mathbf{f}_{18,m_{18}} * \mathbf{h}_{17})]$$

U-Net: Architecture

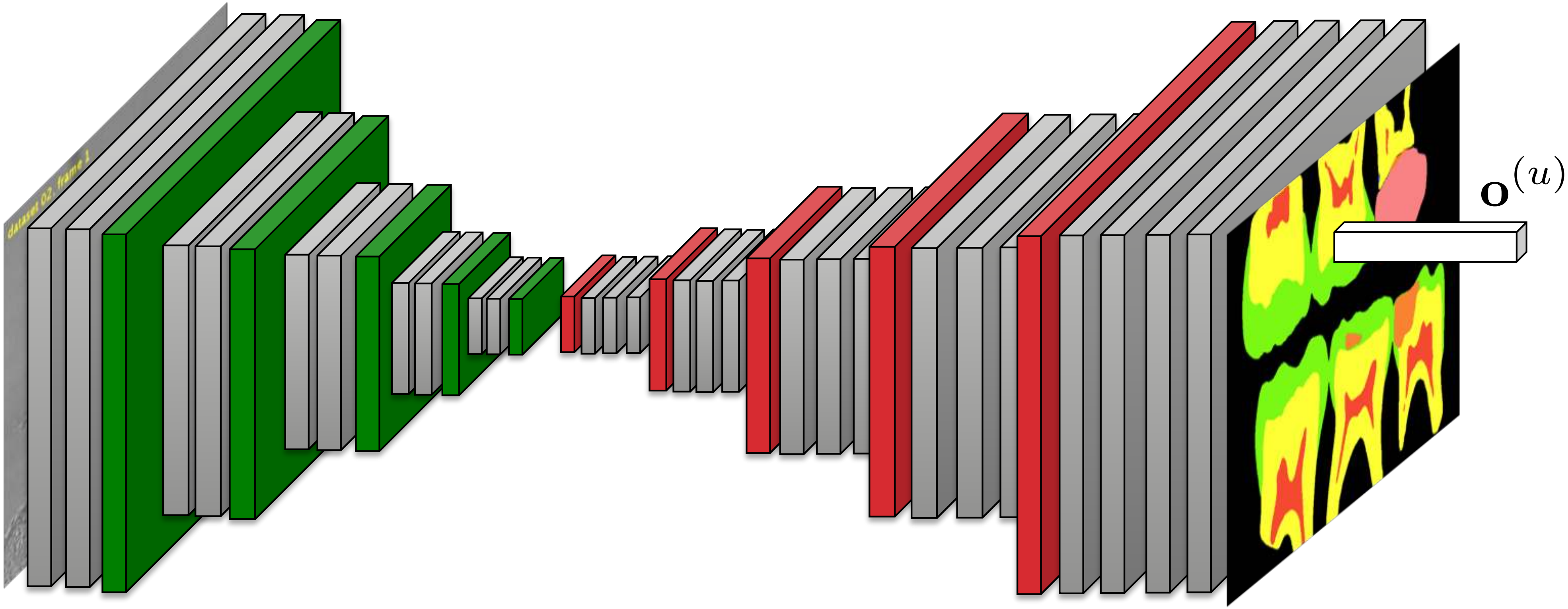


U-Net: Architecture



What is the output of the network exactly?

U-Net: Architecture

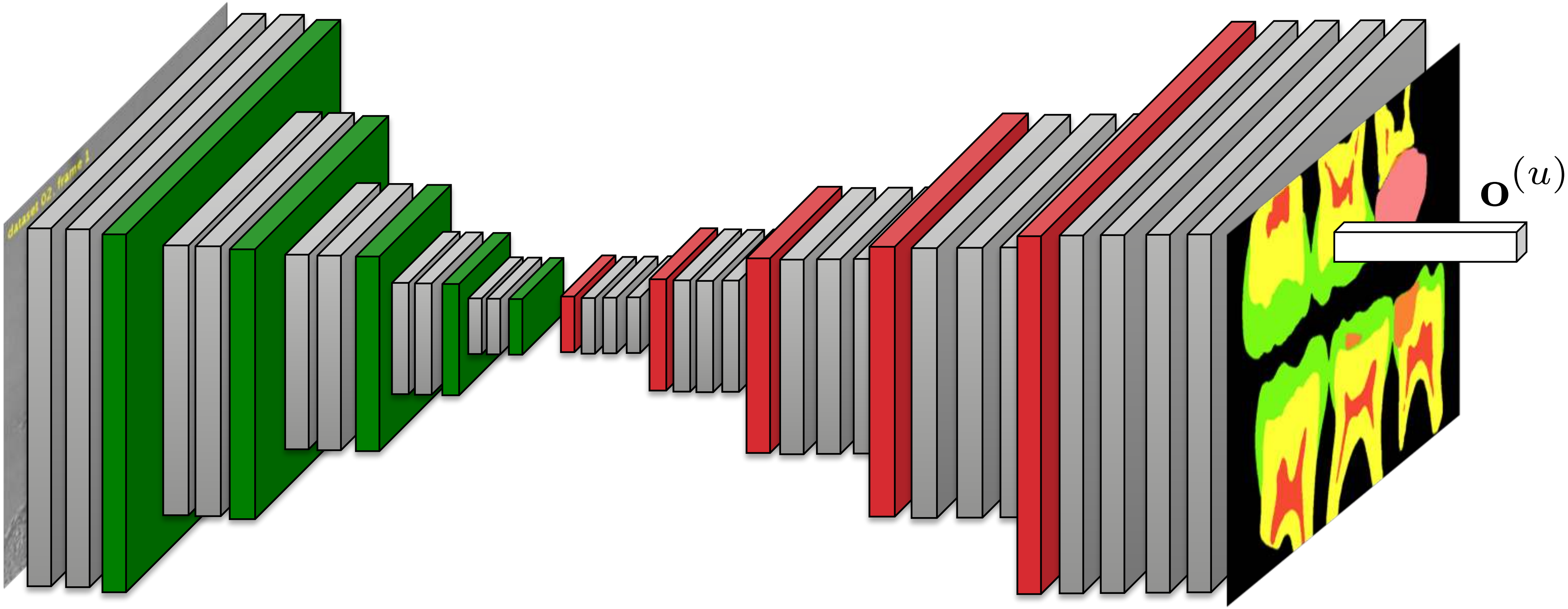


What is the output of the network exactly?

For each pixel, we have 6 possible classes.

$\mathbf{o}^{(u)}$, where u is a pixel, is a vector of 6 values.

U-Net: Loss Function



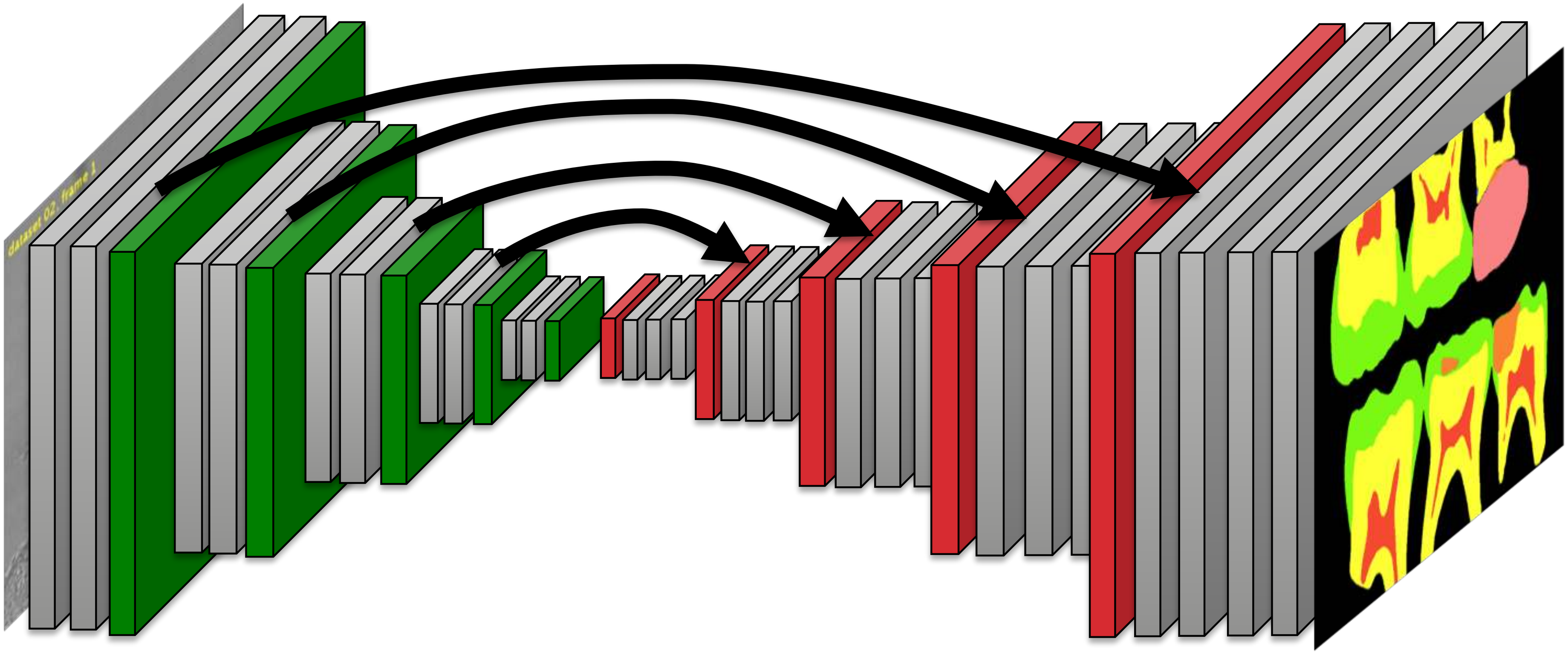
$\mathcal{L} = -\sum_{(\mathbf{x},\mathbf{d})\in\mathcal{T}} \sum_u \log y(\mathbf{x}, \mathbf{d}, u)$

with $y(\mathbf{x}, \mathbf{d}, u) = \text{softmax}(\mathbf{o}^{(u)})_{\mathbf{d}^{(u)}}$

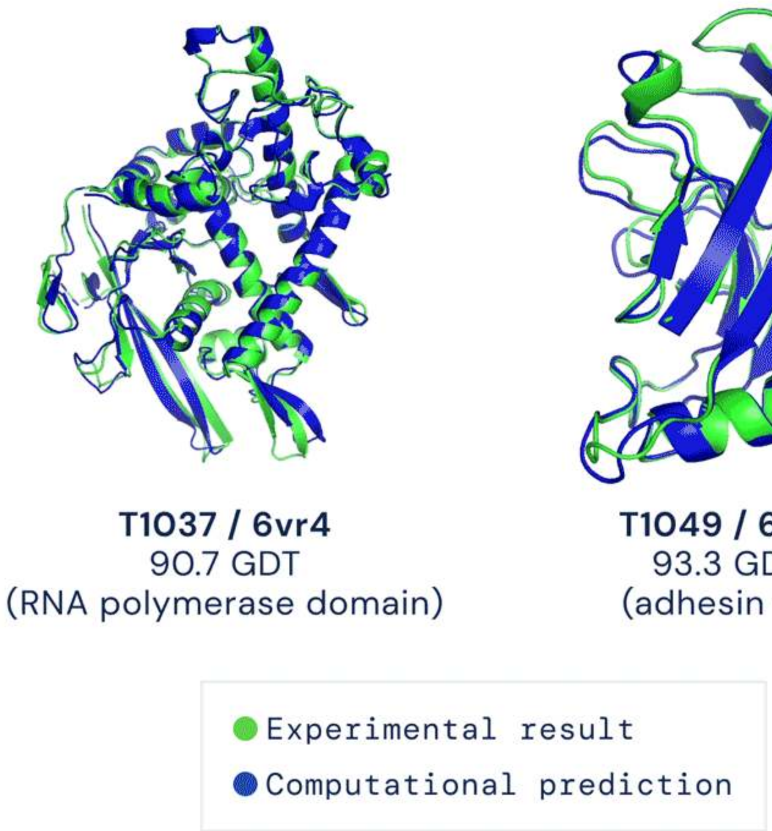
where

- ▷ $\mathbf{o}^{(u)}$ is the network output for image $\mathbf{x}$ for pixel u, and
- ▷ $\mathbf{d}^{(u)}$ is the desired class for pixel u in image $\mathbf{x}$ for pixel u

U-Net: Skip Connections



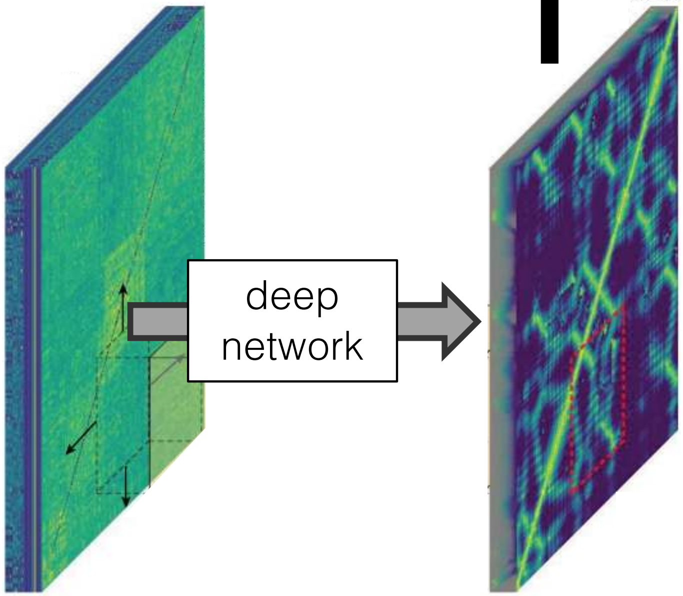
ALPHA FOLD


T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)
T1049 / 6y4f
93.3 GDT
(adhesin tip)
Experimental result
Computational prediction

Protein sequence and MSA features

covariance matrix of the MSA features

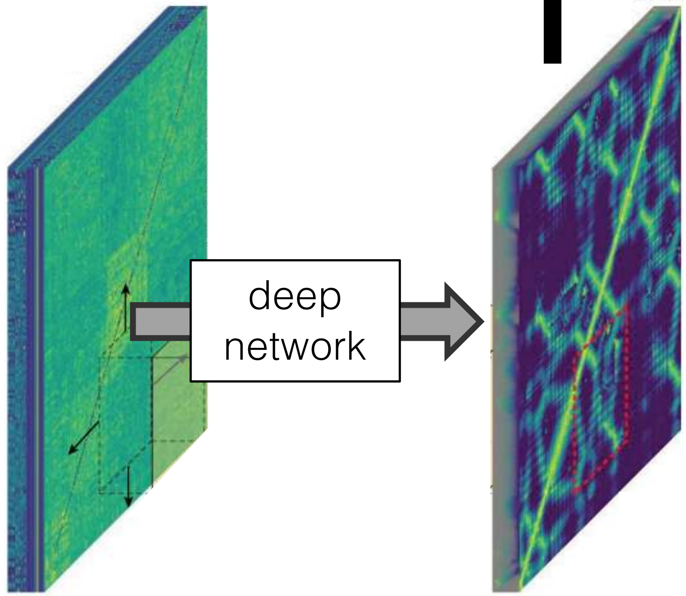
the predicted distagram is used to define a potential: a protein structure with a similar distagram has a low potential



‘distagram’: distribution over the distances and torsions between every pair of residues

Protein sequence and MSA features

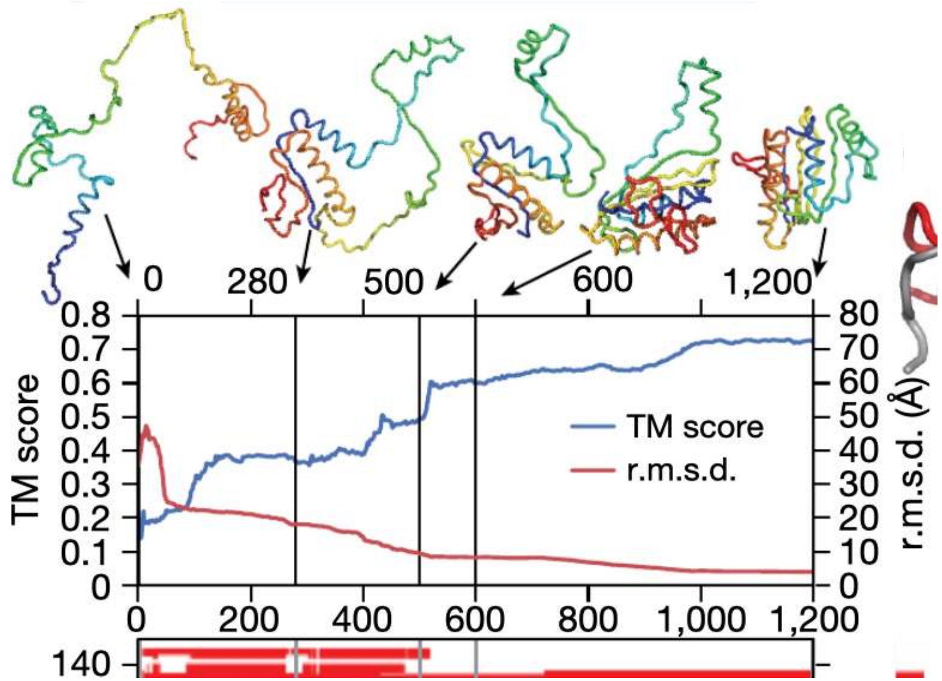
covariance matrix of the MSA features

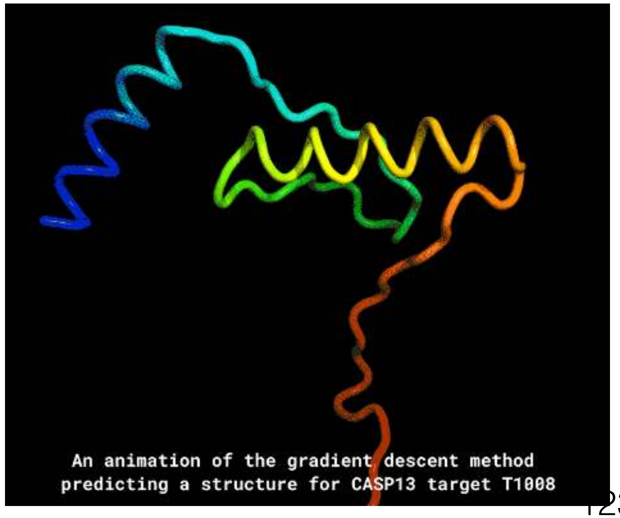


‘distagram’: distribution over the distances and torsions between every pair of residues

the predicted distagram is used to define a potential: a protein structure with a similar distagram has a low potential

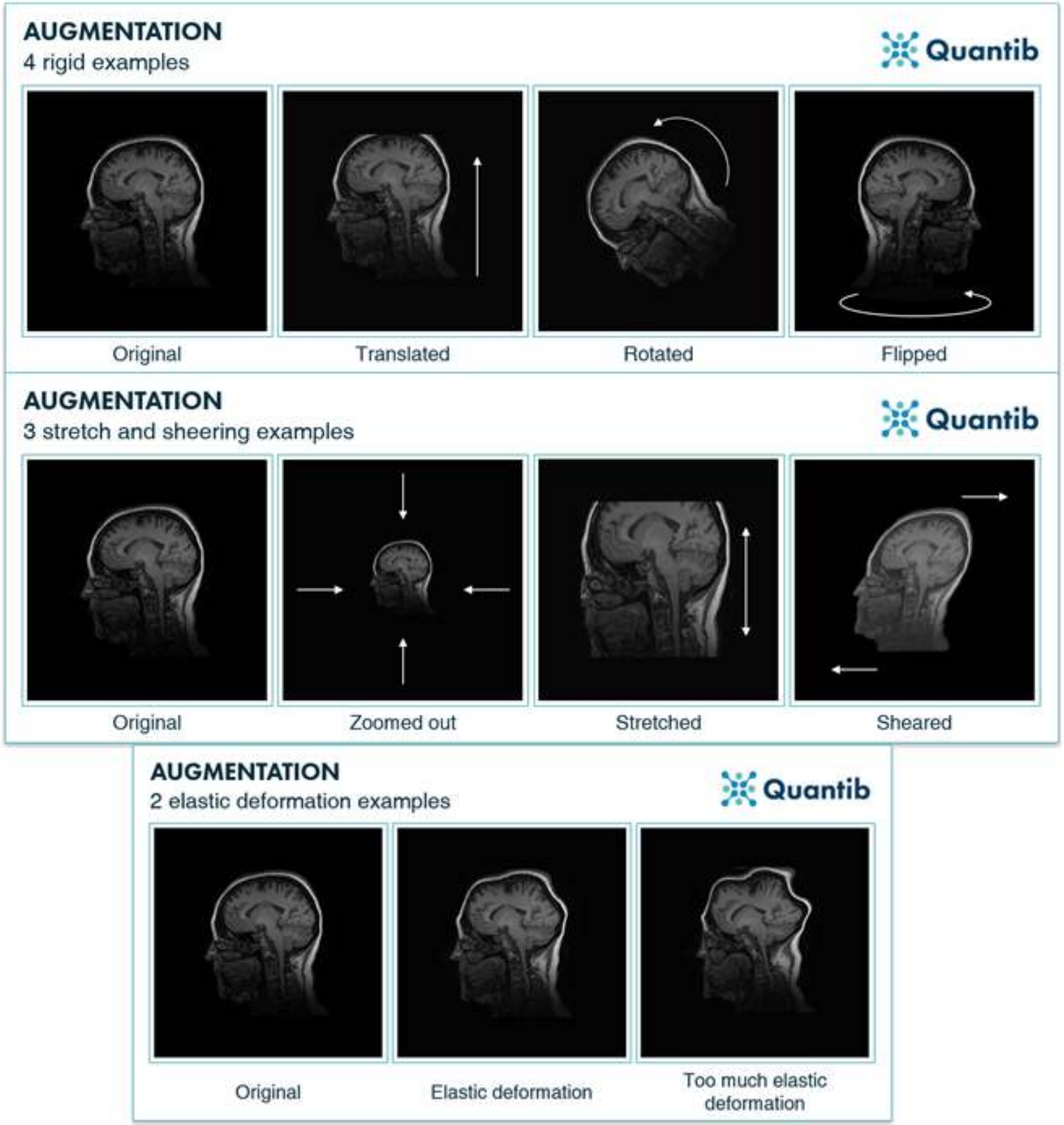
start with a random 3d structure for the protein sequence and deform it to minimize its potential using gradient descent





WHAT TO DO WHEN ONLY FEW TRAINING DATA IS AVAILABLE

Data Augmentation



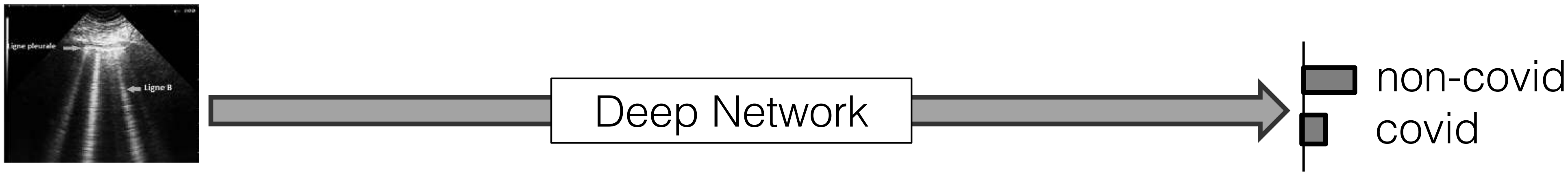
transfer learning / domain transfer

Transfer learning:

- we have few training data on our problem, but
- we have a lot of training data for a similar problem.

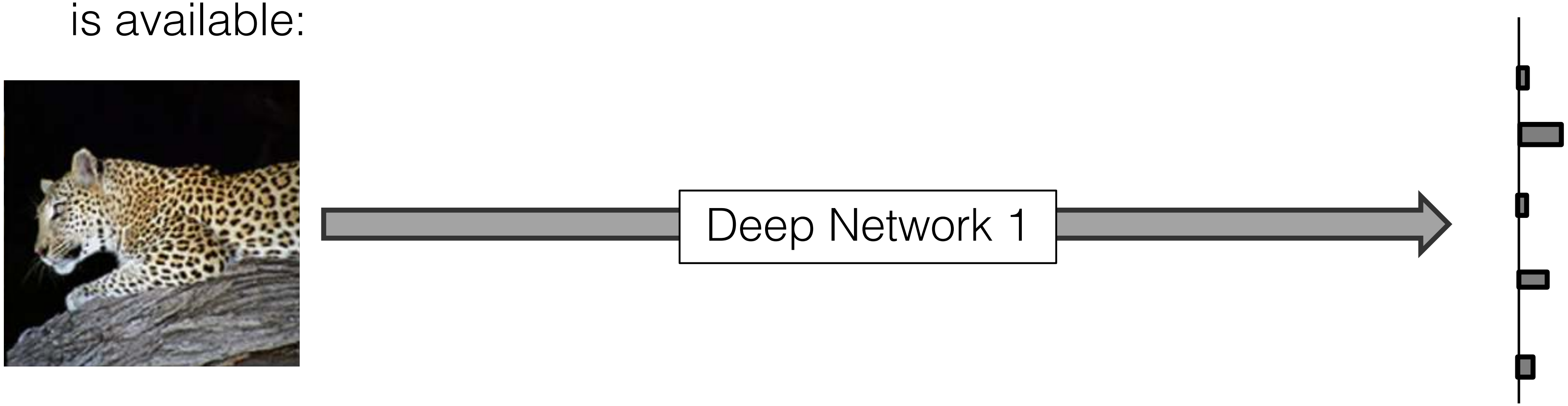
transfer learning / domain transfer

A simple method for transfer learning:



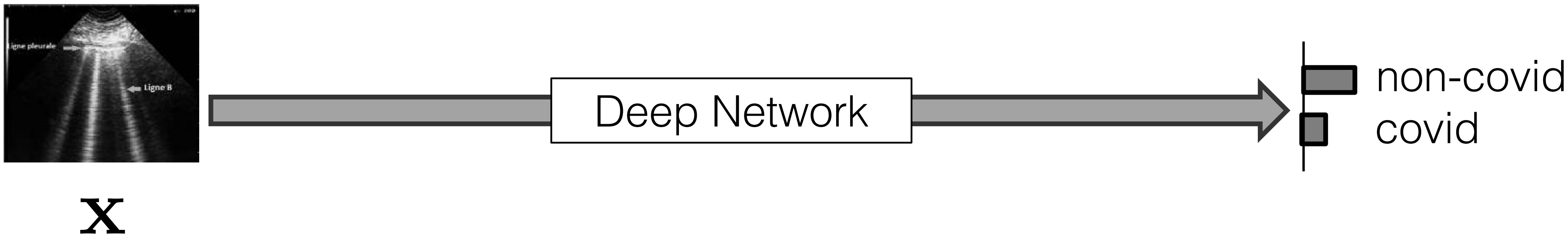
$\mathbf{x}$

1. Training a deep network on a problem where a large amount of training data is available:

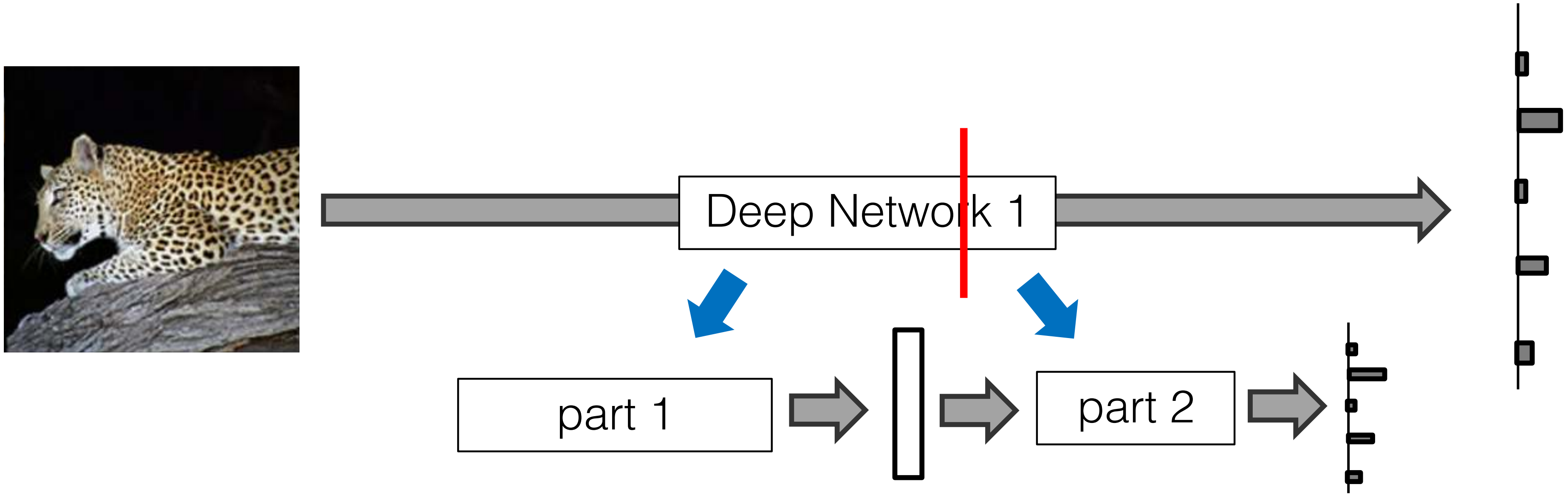


transfer learning / domain transfer

A simple method for transfer learning:



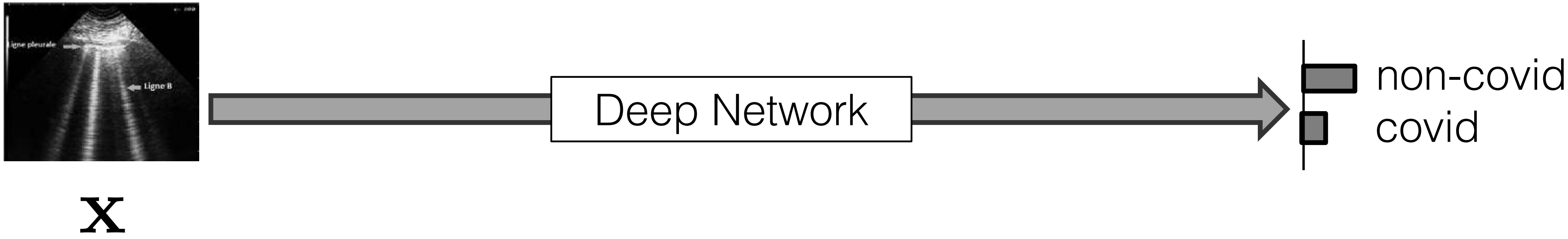
2. Cut this network into two parts (after training):



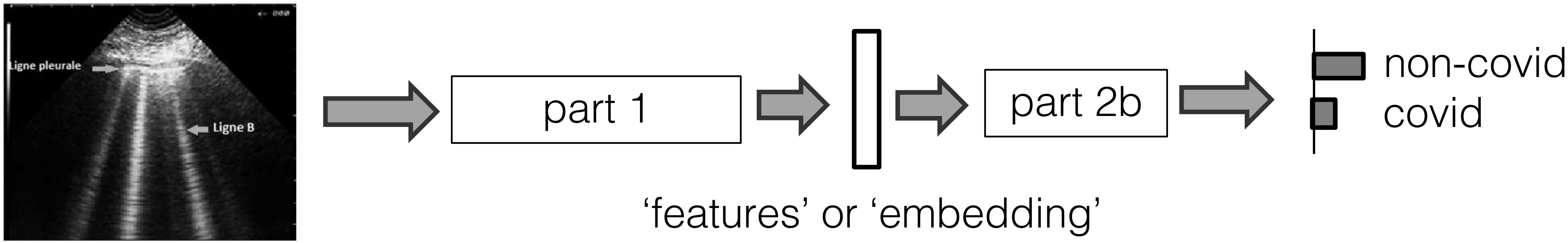
'features' or 'embedding'

transfer learning / domain transfer

A simple method for transfer learning:

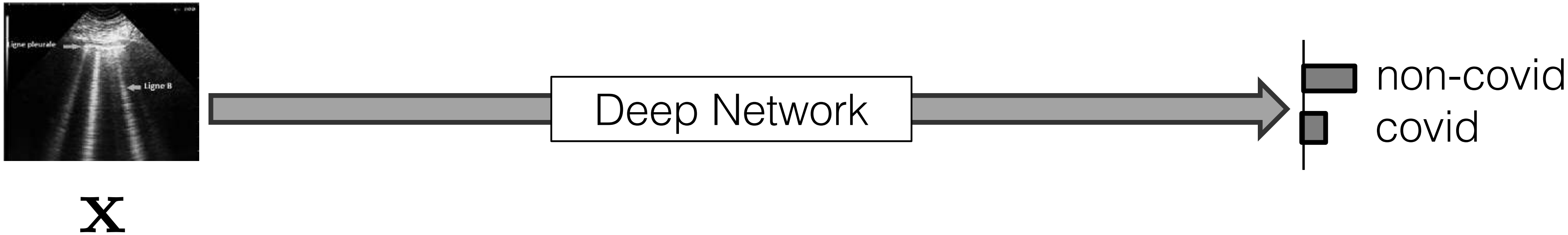


3. Keep the parameters of Part 1, initialize randomly Part 2b with the new number of classes

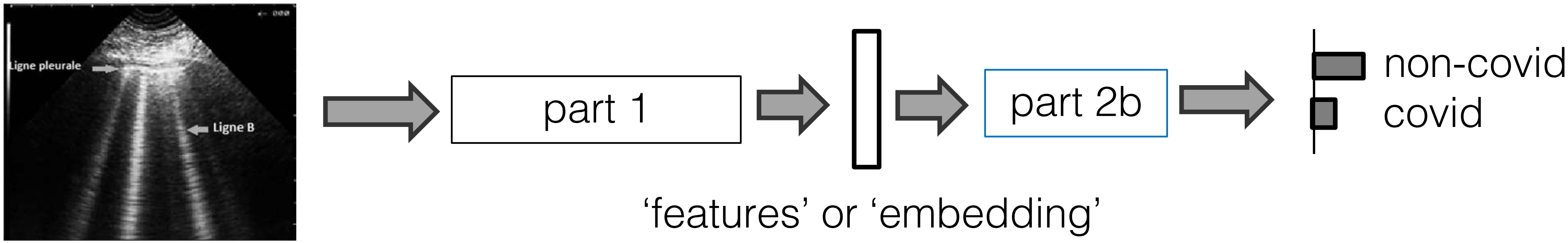


transfer learning / domain transfer

A simple method for transfer learning:

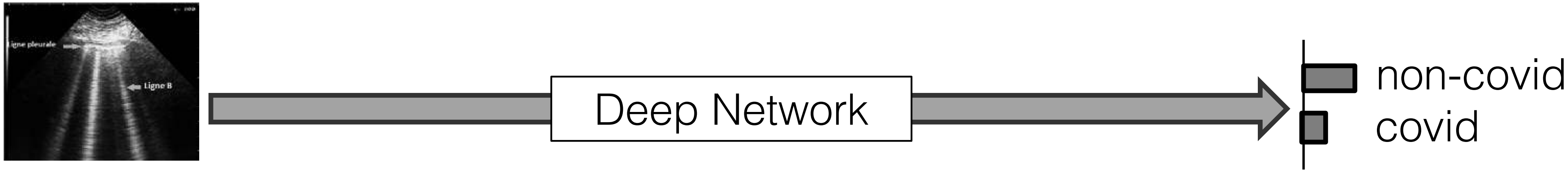


4. Keep the parameters of Part 1, optimize only the parameters of Part 2b on the available data
Alternatively, we can ‘fine-tune‘ the parameters of Part 1.



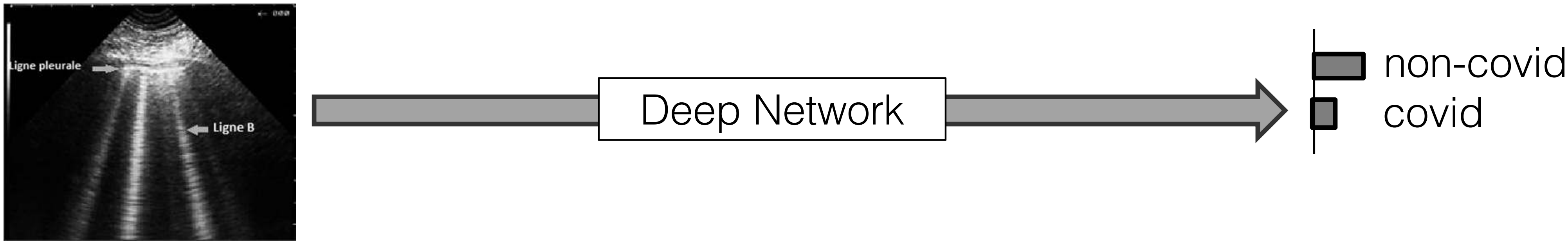
transfer learning / domain transfer

A simple method for transfer learning:



$\mathbf{x}$

Part 1 and Part 2b form a deep network:

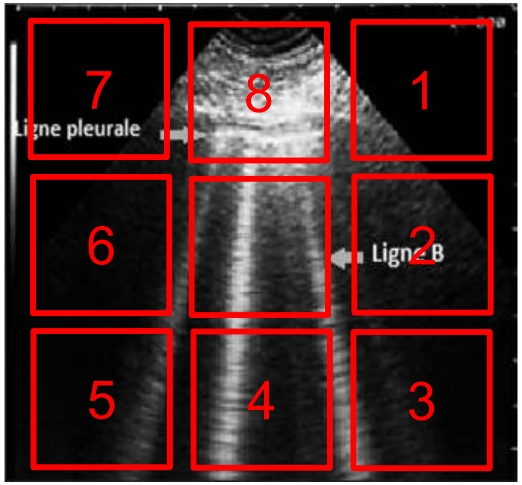


self-learning

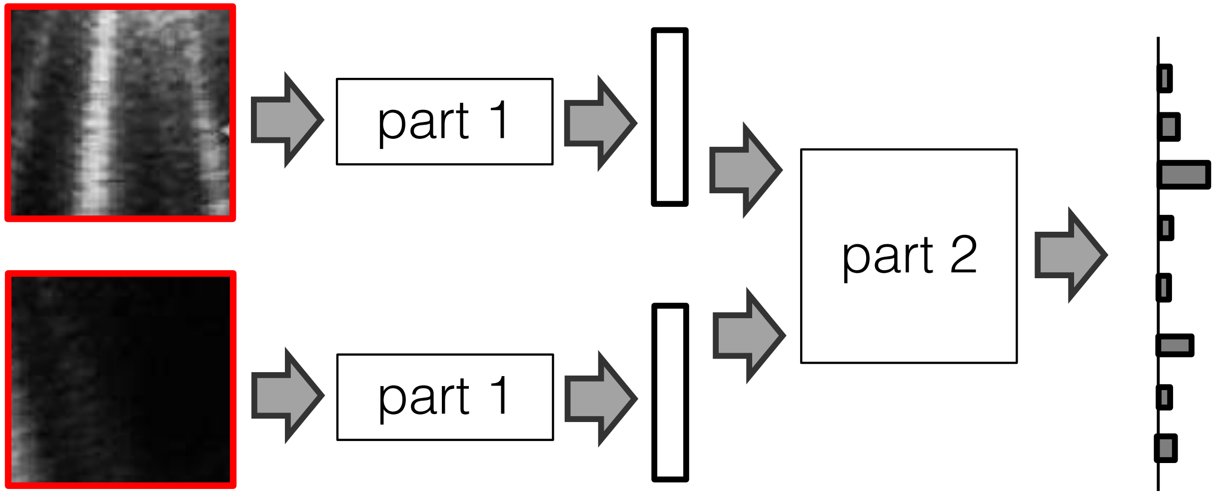
Self-learning for learning features.

Related to transfer learning but the first problem is 'artificial'.

For example:



Given the center image, and one the 8 images, predict from where is taken this second image:



TEXT PROCESSING

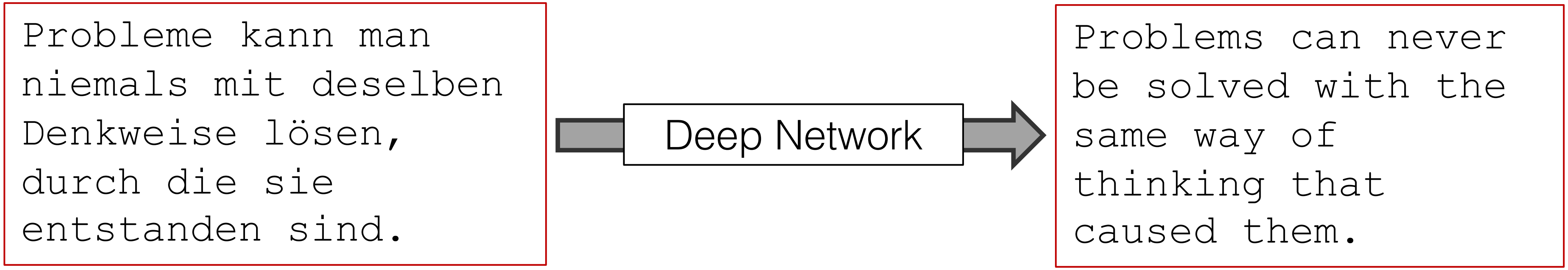

Probleme kann man niemals mit deselben Denkweise lösen, durch die sie entstanden sind.
Deep Network
Problems can never be solved with the same way of thinking that caused them.

IMAGE CAPTIONING

The man at bat readies to swing at the pitch while the umpire looks on.

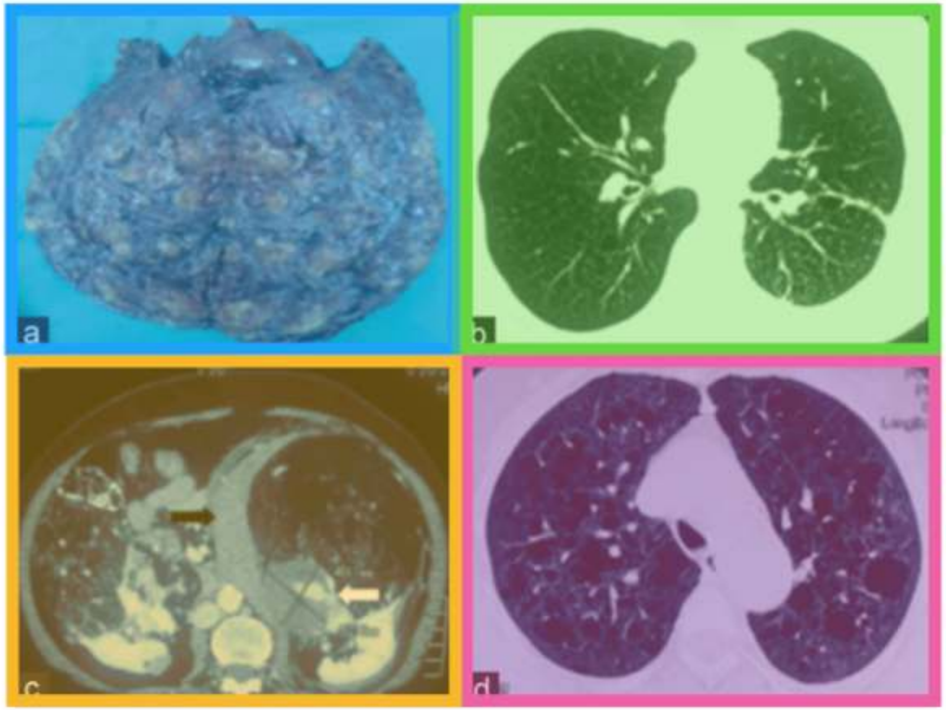


Figure 1: **(a)** Right renal angiomyolipoma (gross specimen postexcision). **(b)** High-resolution computed tomography chest images of Case 1 showing multiple variable sized cysts uniformly scattered in both lungs. **(c)** Computed tomography abdomen showing bilateral renal angiomyolipomas with fat densities, tortuous vessels, and pseudoaneurysm (white arrow). There is also the presence of perinephric hematoma (black arrow). **(d)** High-resolution computed tomography image of Case 2 showing bilateral lung cysts.

GENERATIVE ADVERSARIAL NETWORKS (GANS)

some applications of gans

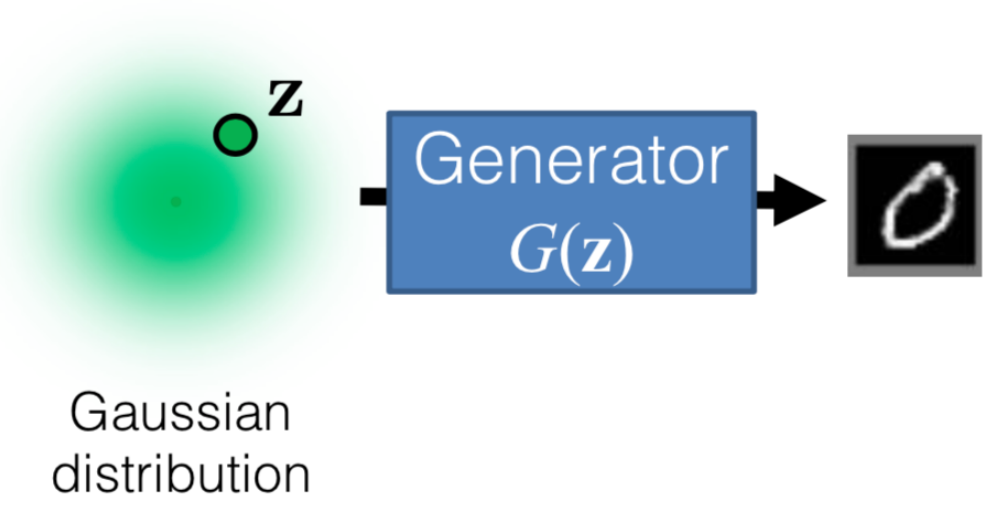
Generating new images/data:

Style-transfer and Deep Fakes:

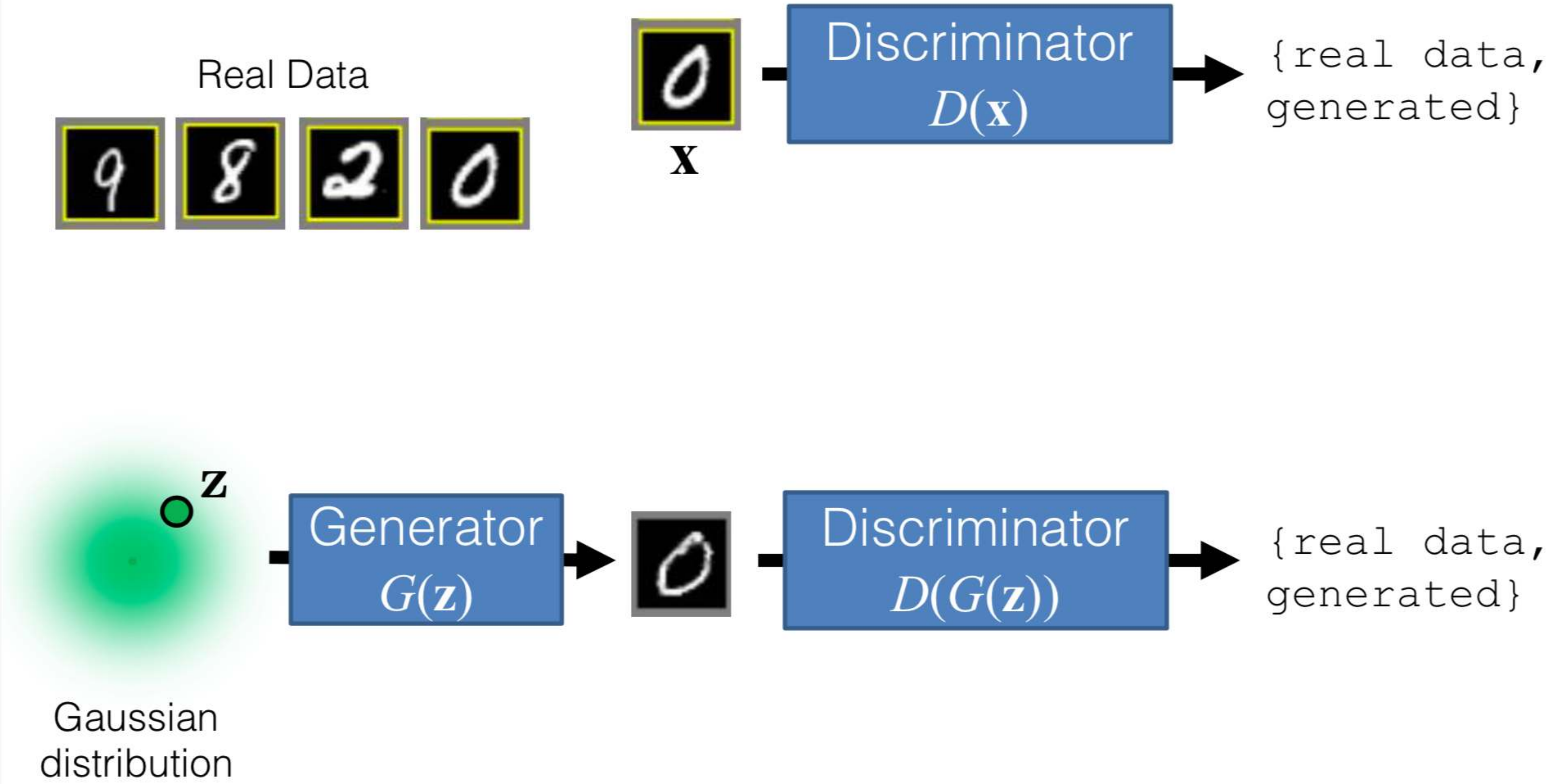
Generating potential drugs

GANs

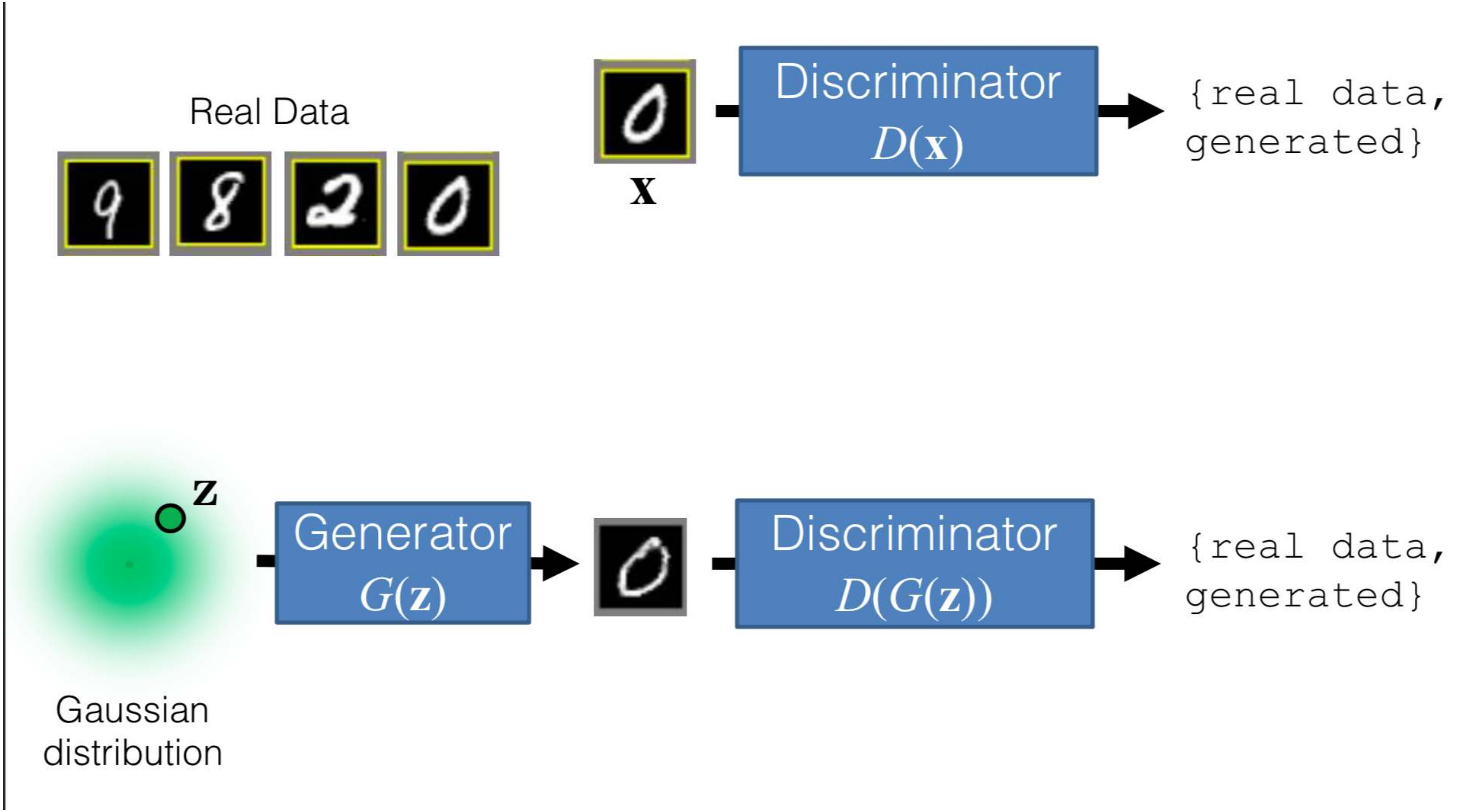
We would like to train a network G to generate images of digits from noise vectors $\mathbf{z}$:



GANs


Real Data
x
Discriminator
$D(\mathbf{x})$
{real data, generated}
z
Gaussian distribution
Generator
$G(\mathbf{z})$
Discriminator
$D(G(\mathbf{z}))$
{real data, generated}

GANs

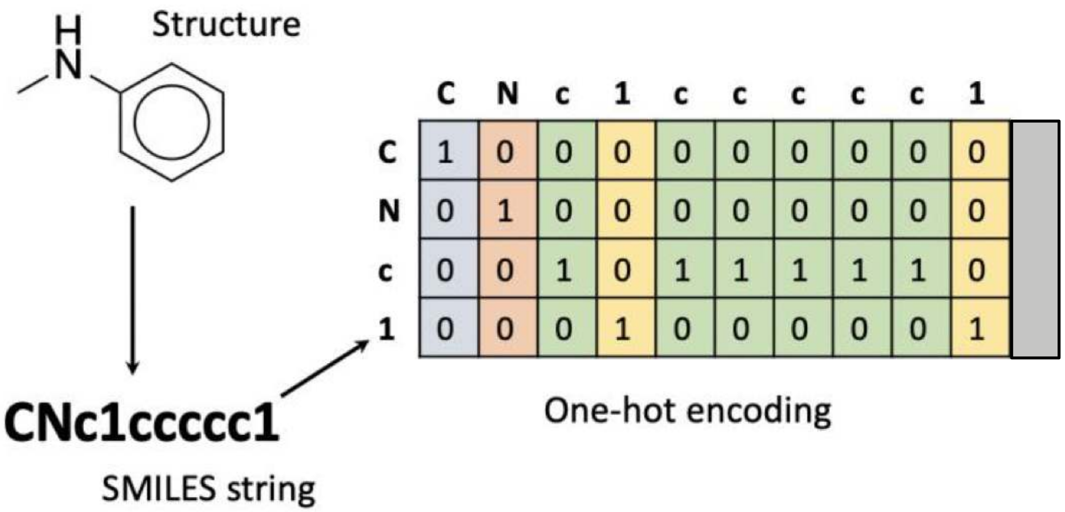


$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

generating molecules

Generative chemistry: drug discovery with deep learning generative models. Yuemin Bian and Xiang-Qun Xie. arXiv 2020.

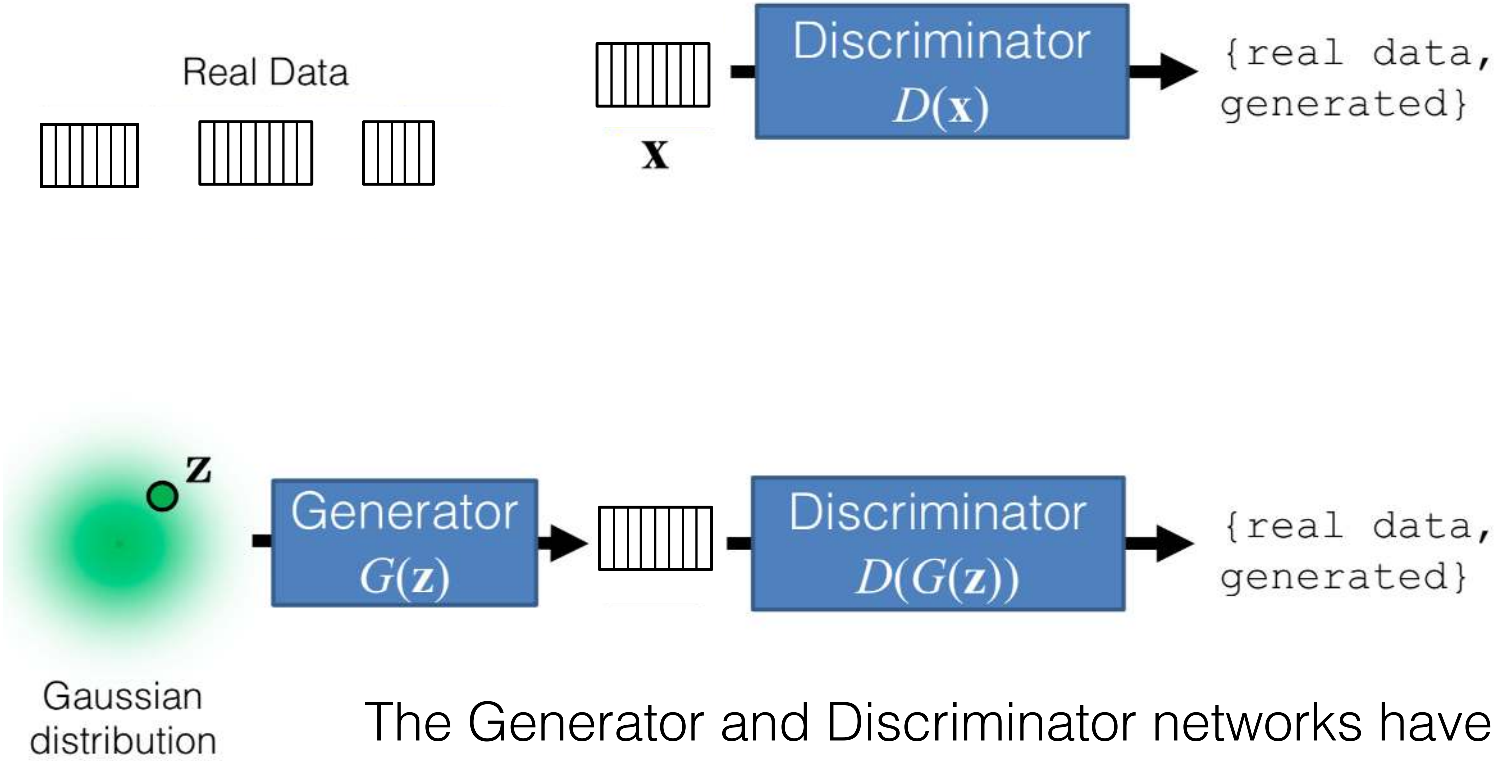
1) We need a representation for molecules:



this is a text-like representation.

generating molecules

2) → we can train a GAN to learn generating new representations of molecules



The Generator and Discriminator networks have the same form as networks for text generation and text analysis.

perspectives

- Black box models and explicability;
- Learning with less training data;
- Good practices for ‘AI engineering’ as for ‘software engineering’.