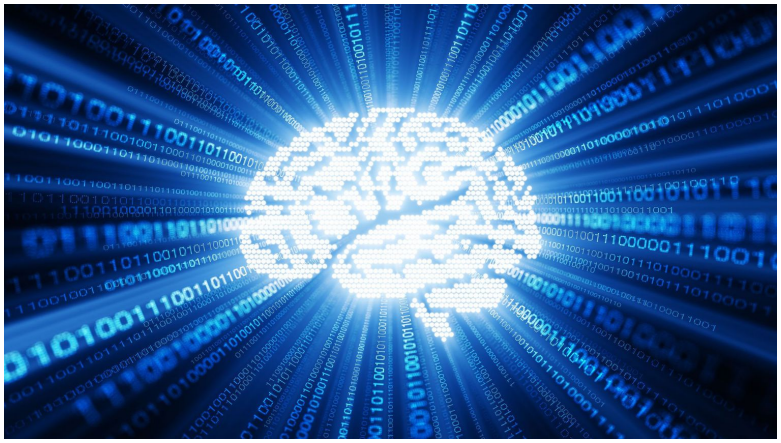
Introduction to Deep Learning

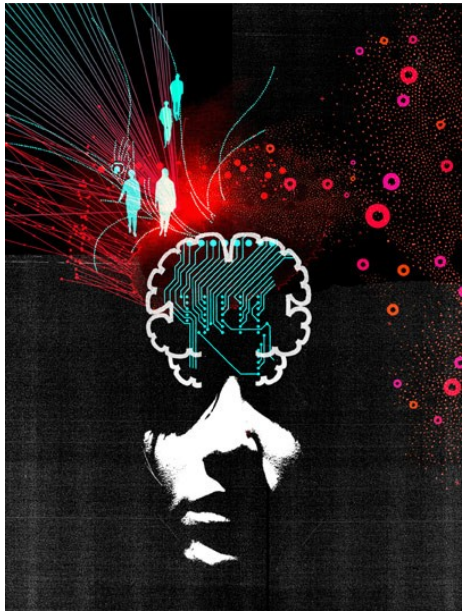
Vincent Lepetit

Deep Learning

[Vierra / Shutterstock / Deep Dream Generator / Le Monde 2017]



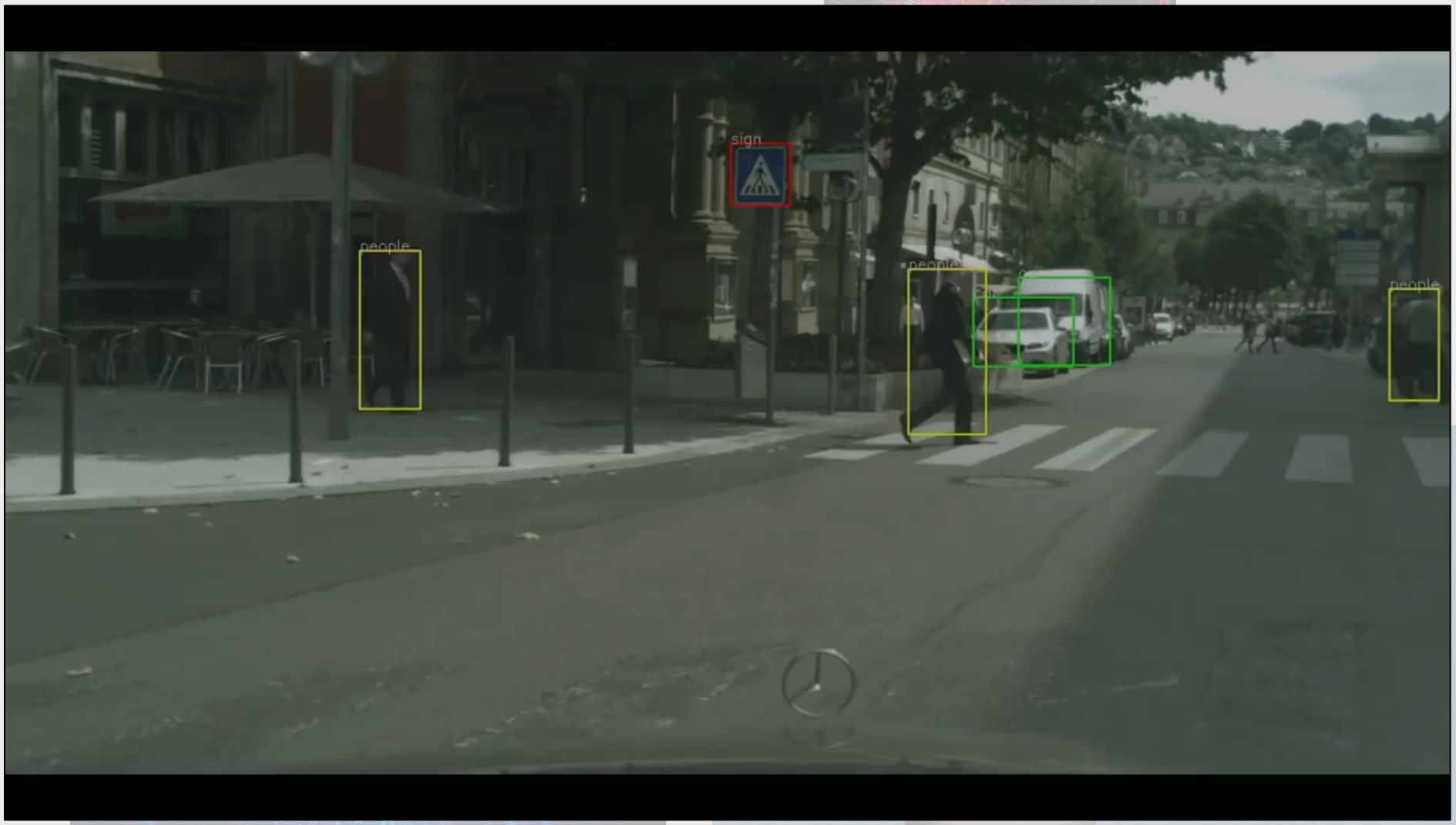
[L'Express, 2017]



[MIT review technology, 2017]

[New Economist, 2015]

Deep Learning

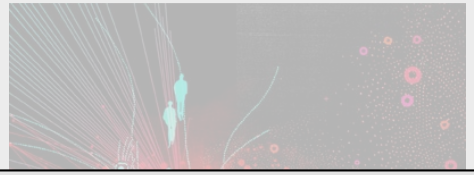


NVIDIA CES 2016

[L'Express, 2017]

[New Economist, 2015]

Deep Learning



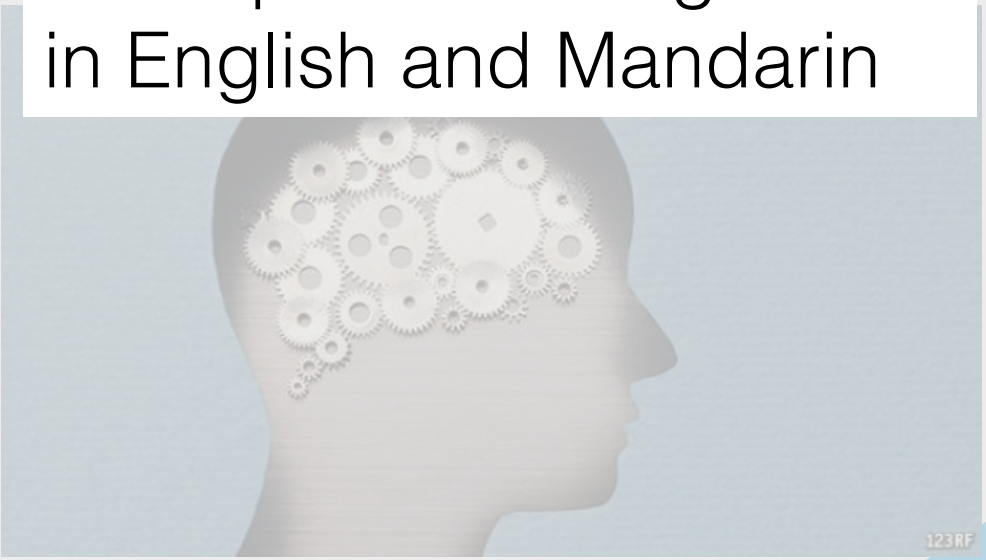
[L'Express, 2017]

[New Economist, 2015]

Deep Speech 2

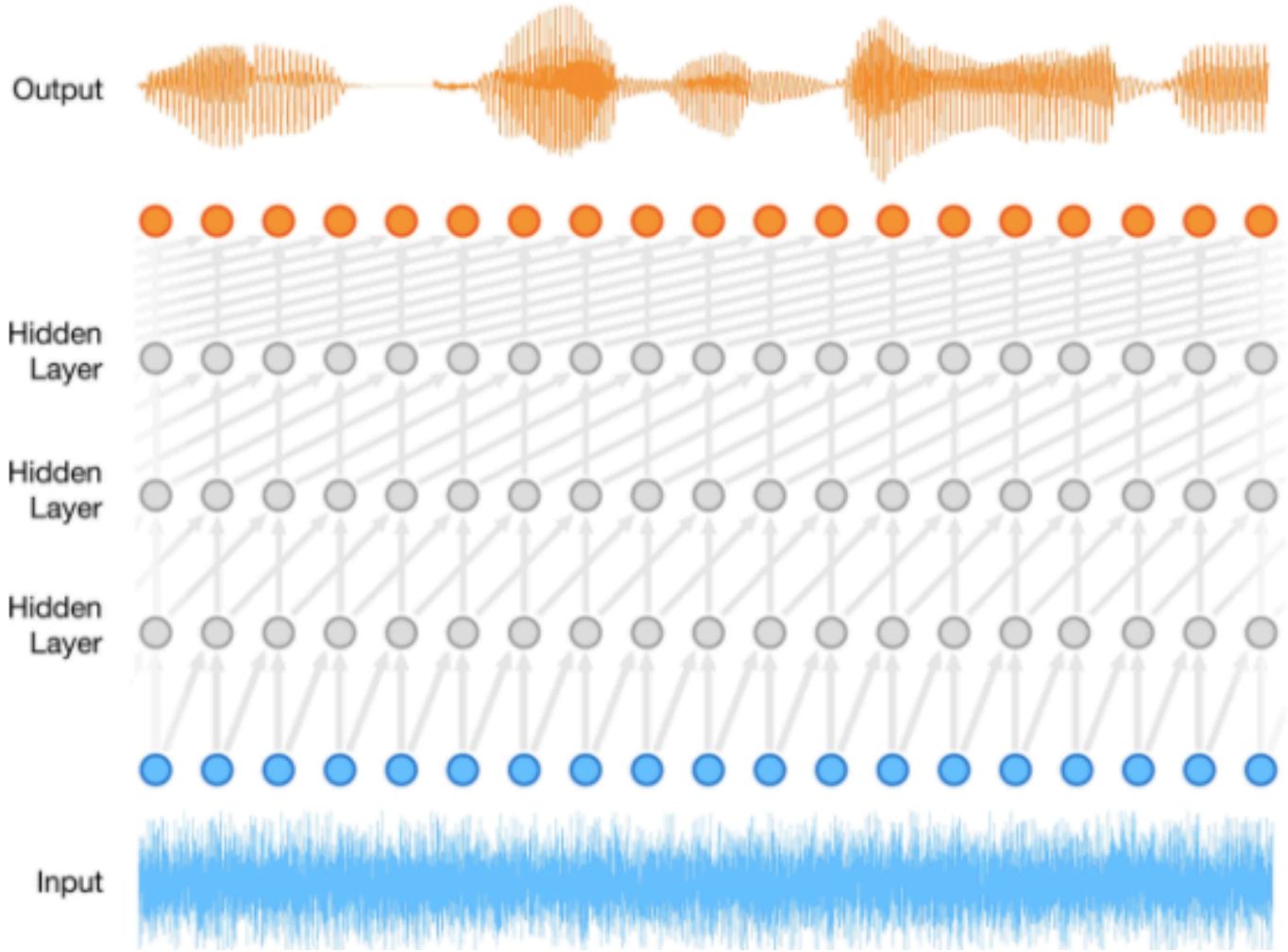
[L'Express, 2017]

Deep Speech 2: End-to-End Speech Recognition in English and Mandarin

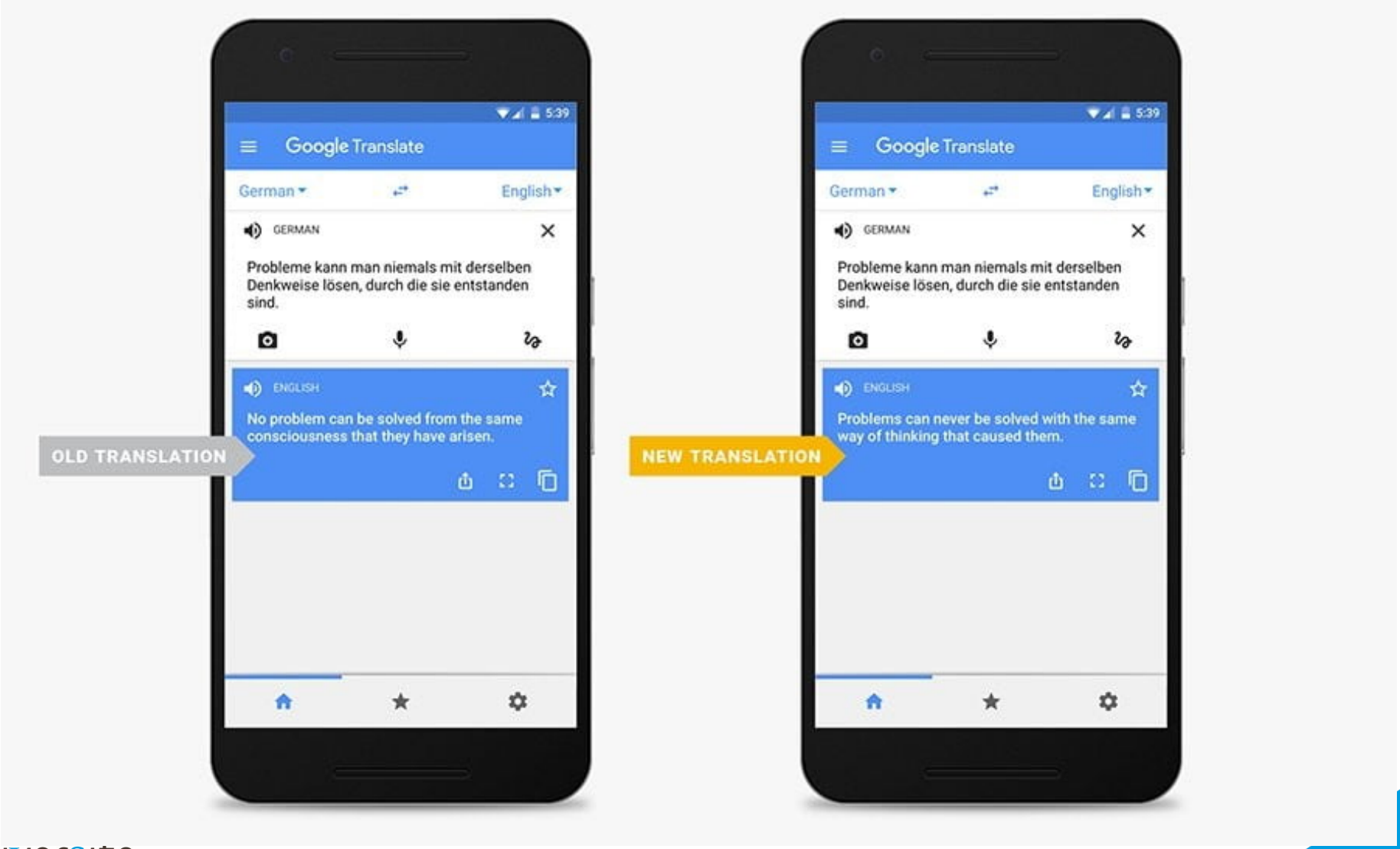


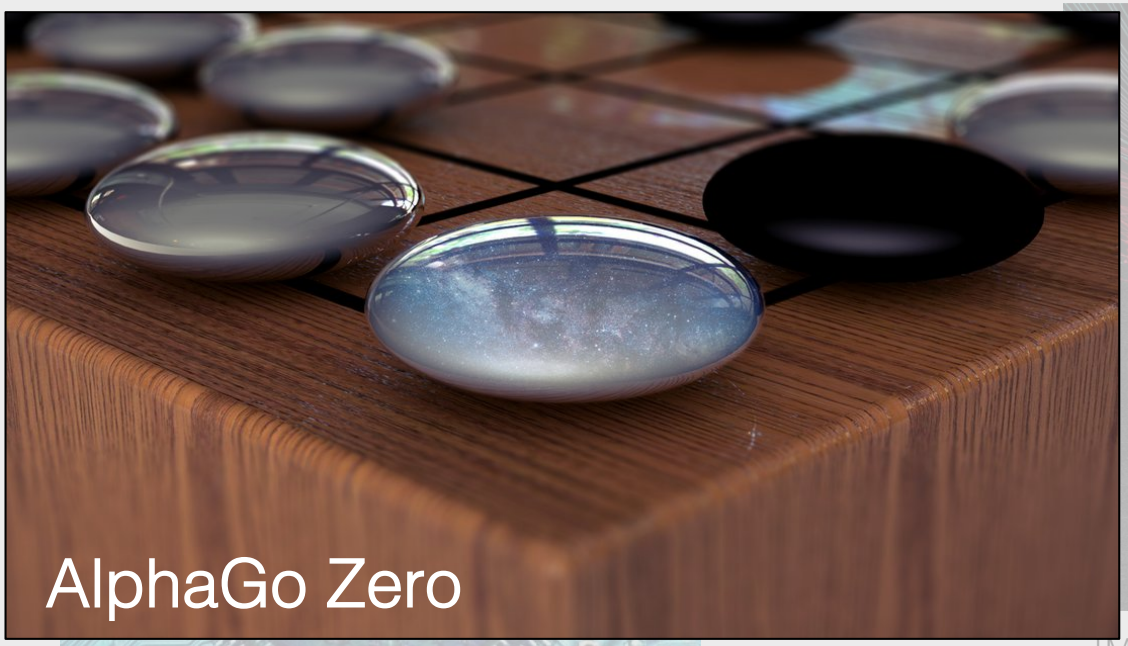
[New Economist, 2015]

WaveNet: Speech Synthesis



Machine Translation

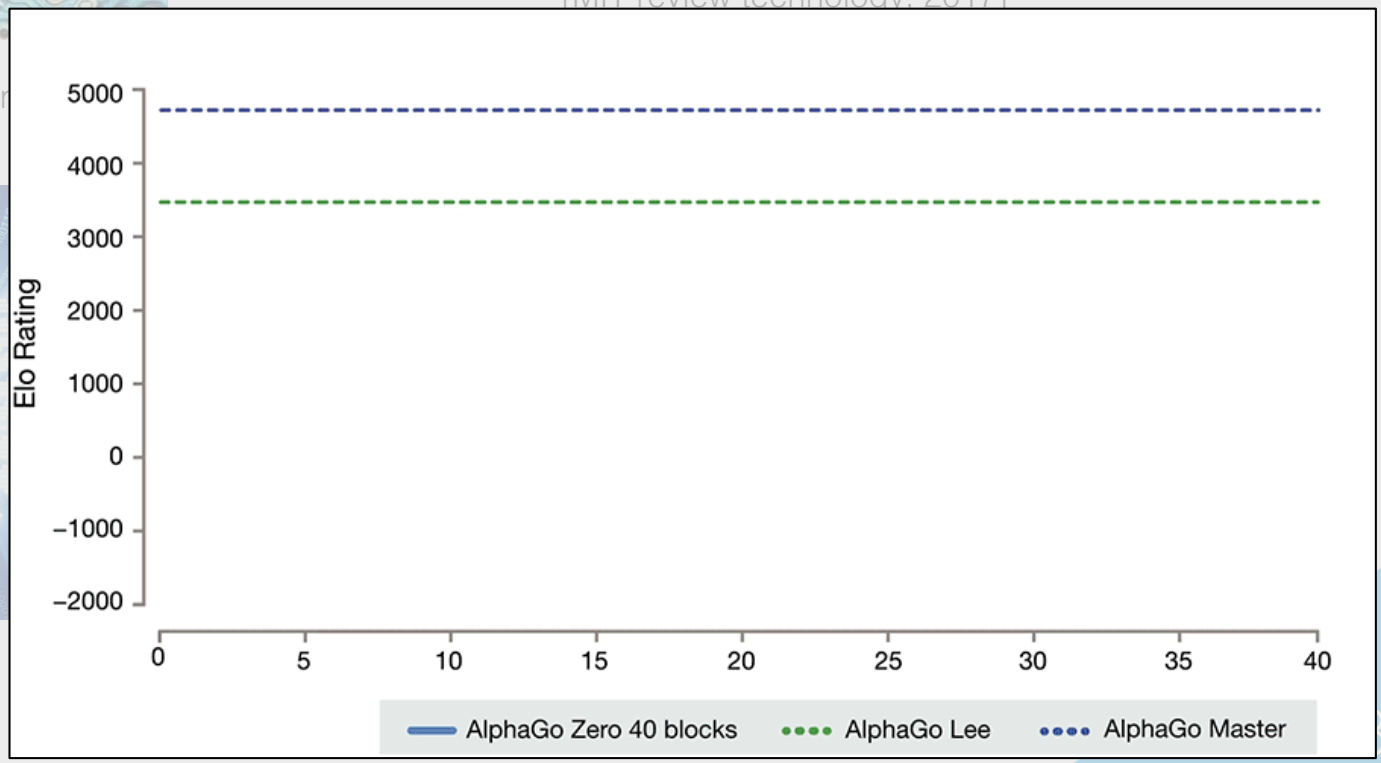




[MIT review technology, 2017]

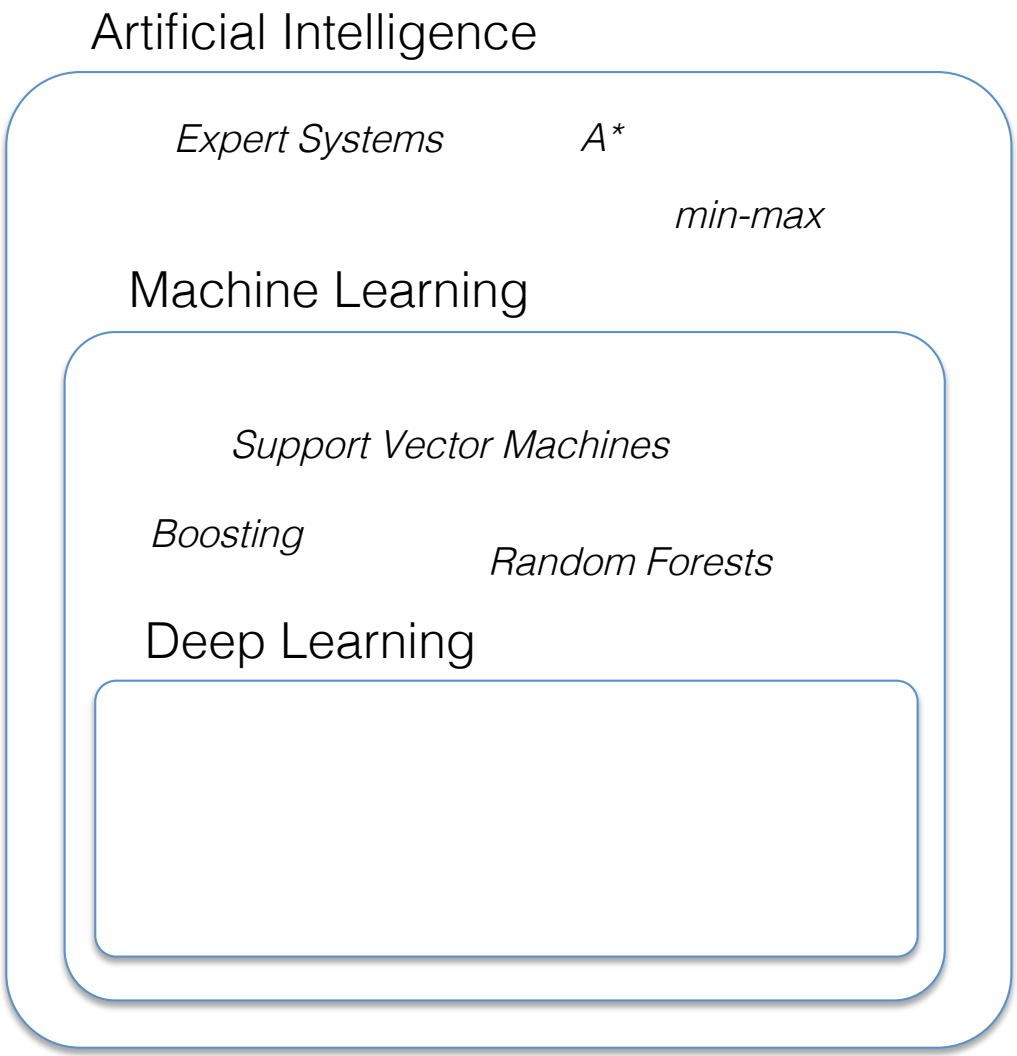
[Vierra / Shutterstock / Deep Dr
Generator / Le Monde 2017]

[L'Express, 2017]

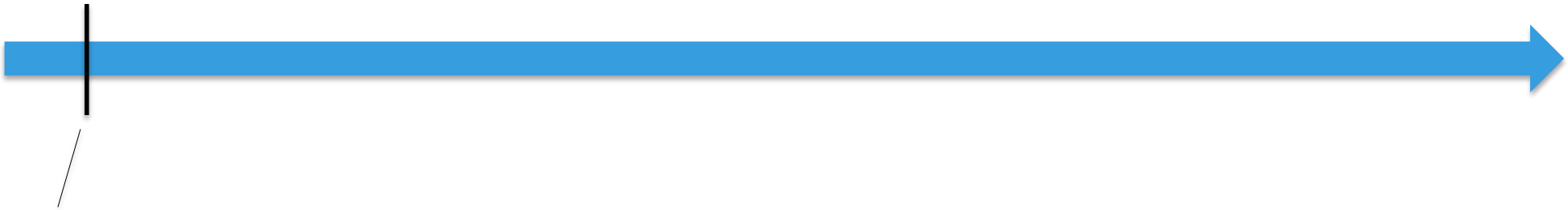


université de BORDEAUX

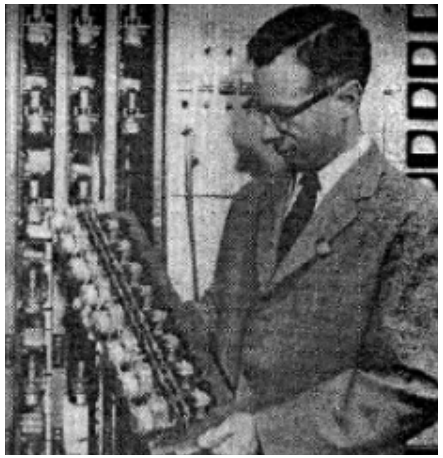
Artificial Intelligence/Machine Learning/Deep Learning



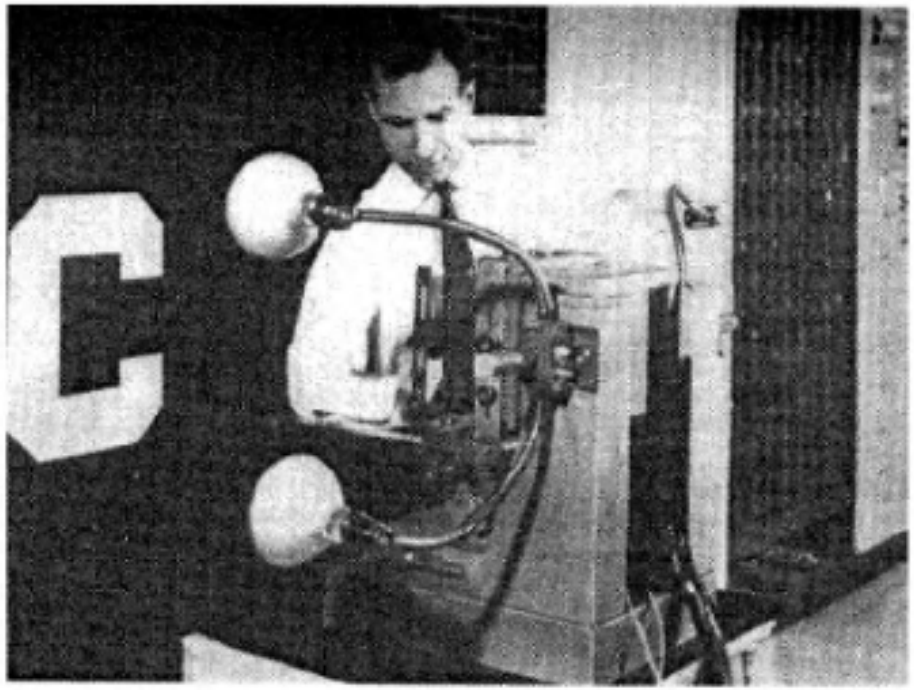
History of [Deep] Learning

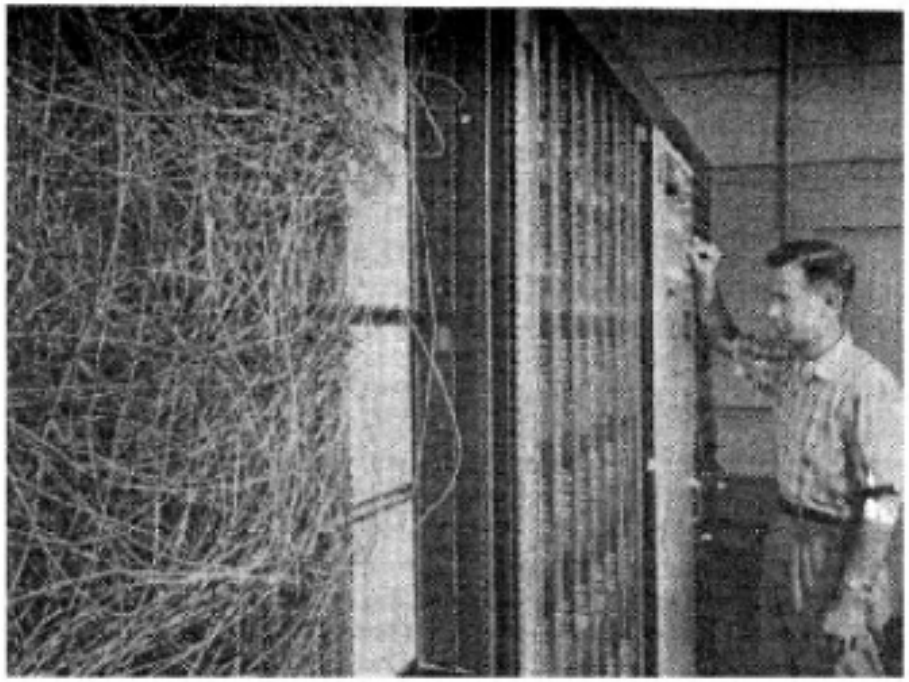


1958, Cornell:
Perceptron



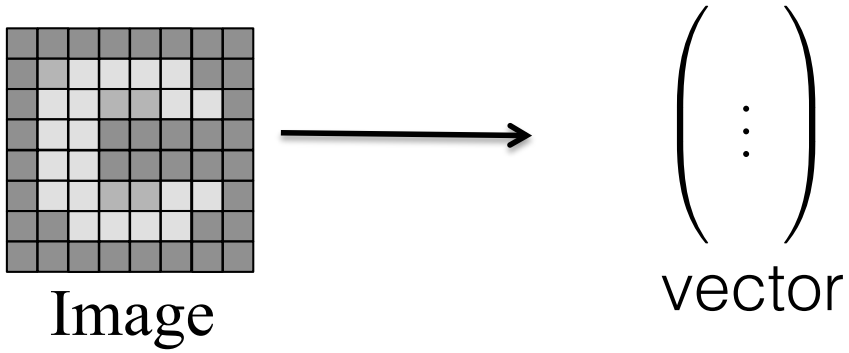
Perceptron (1958, Frank Rosenblatt)





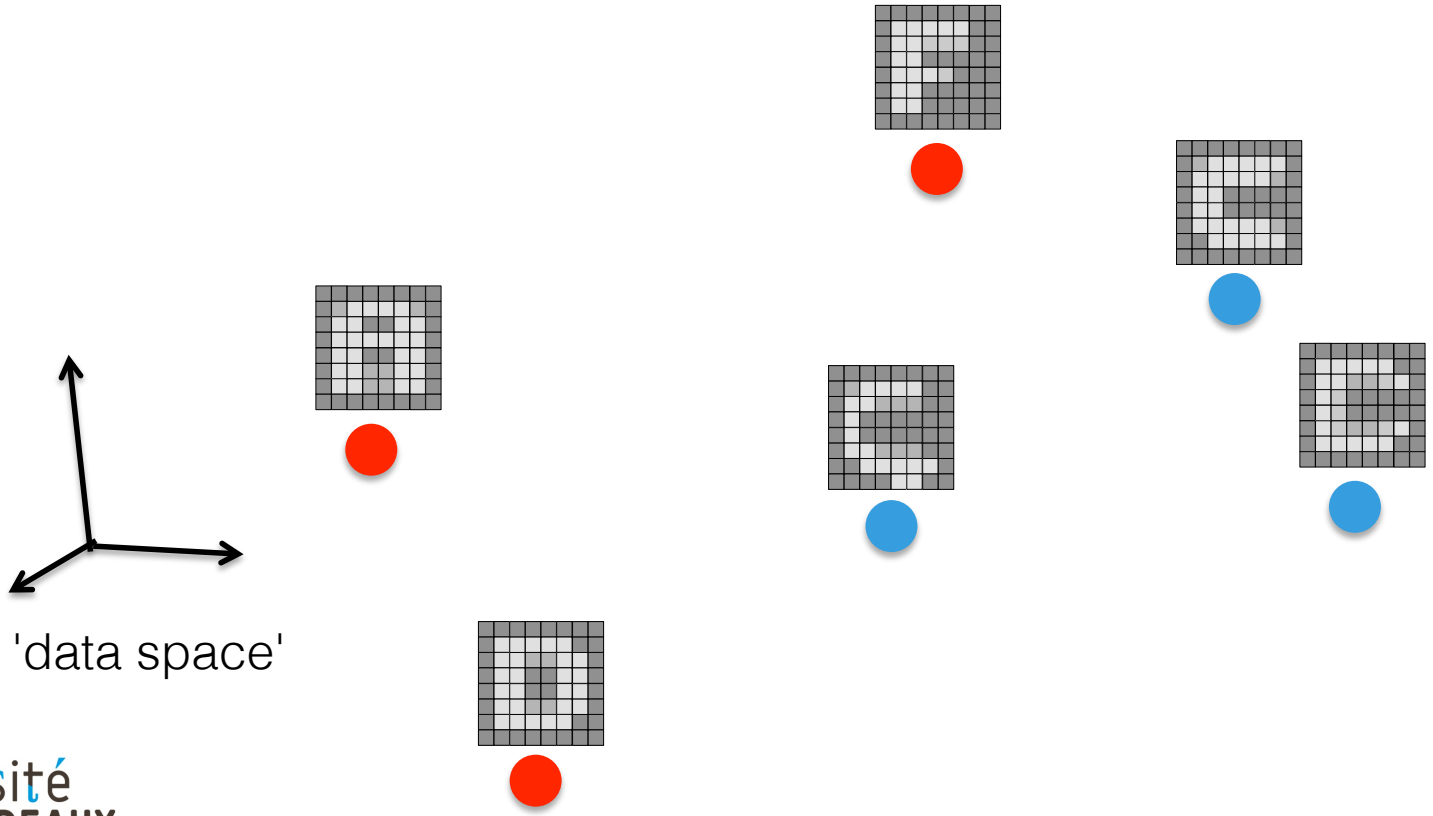
Perceptron: Idea

1. Data are represented as vectors:



Perceptron: Idea

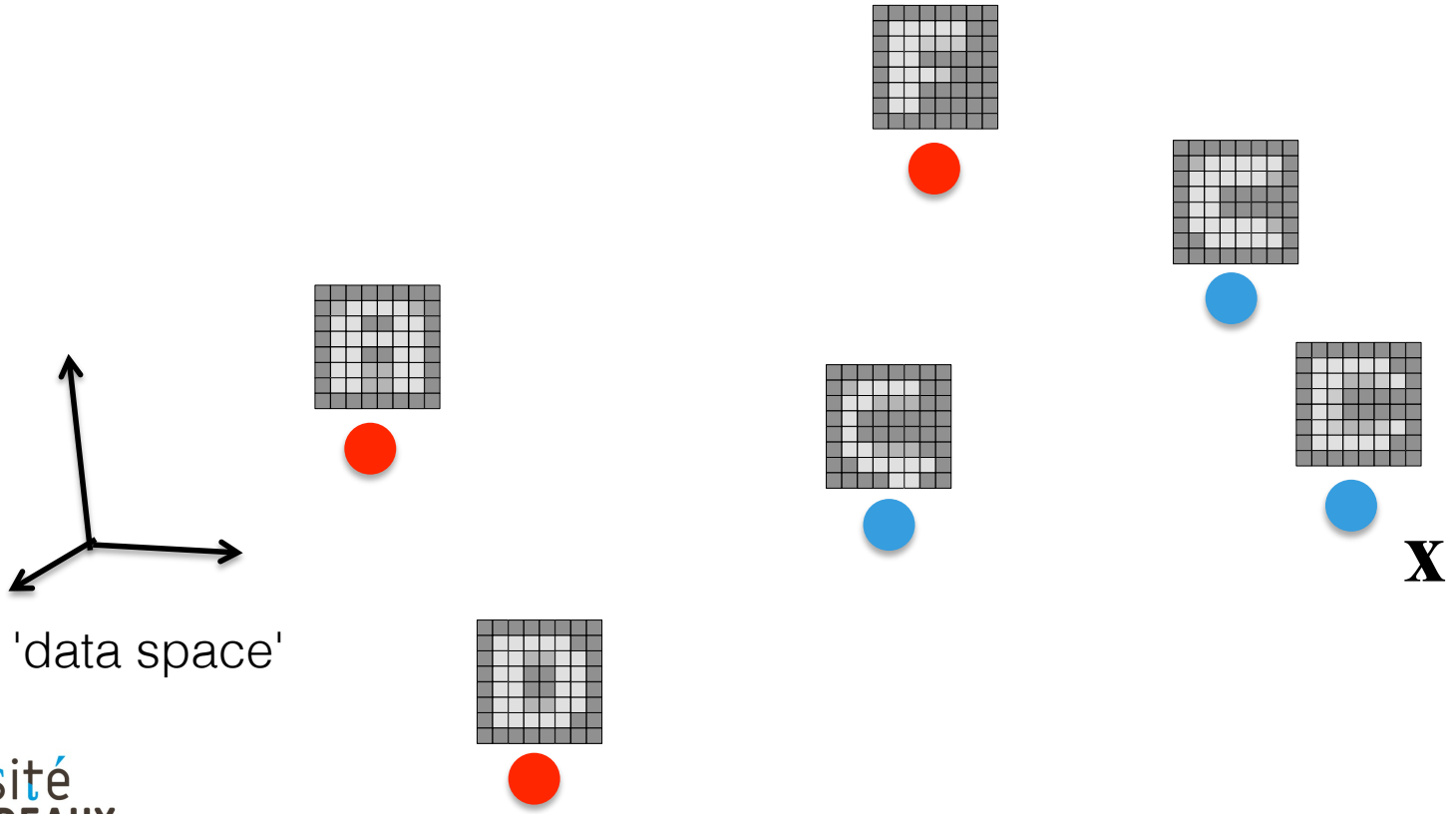
2. Collect training data: some are *positive* examples, some are *negative* examples



Perceptron: Idea

3. Training: find $\mathbf{a}$ and b so that

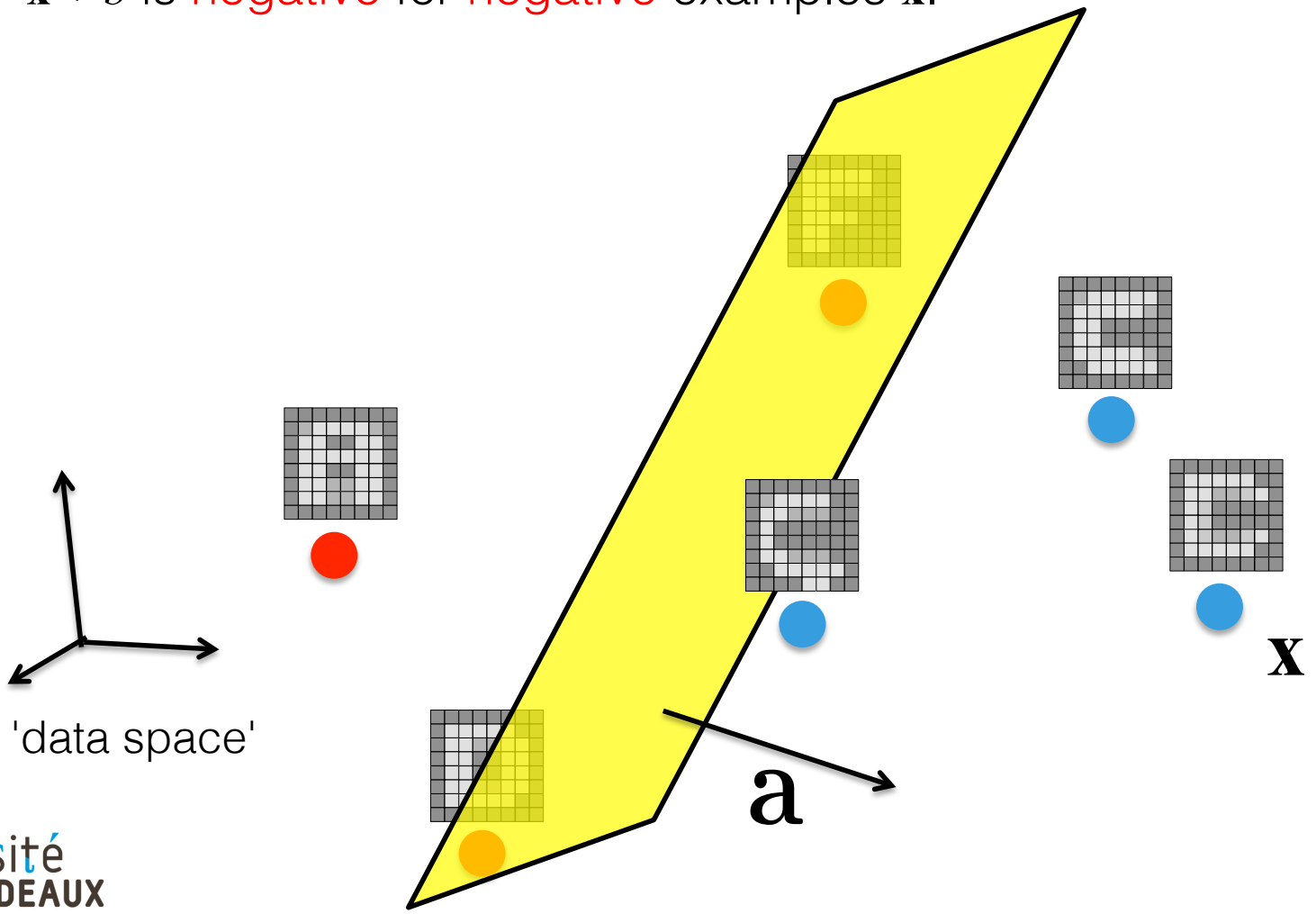
- $\mathbf{a}^\top \mathbf{x} + b$ is positive for positive examples $\mathbf{x}$;
- $\mathbf{a}^\top \mathbf{x} + b$ is negative for negative examples $\mathbf{x}$.



Perceptron: Idea

3. Training: find $\mathbf{a}$ and b so that

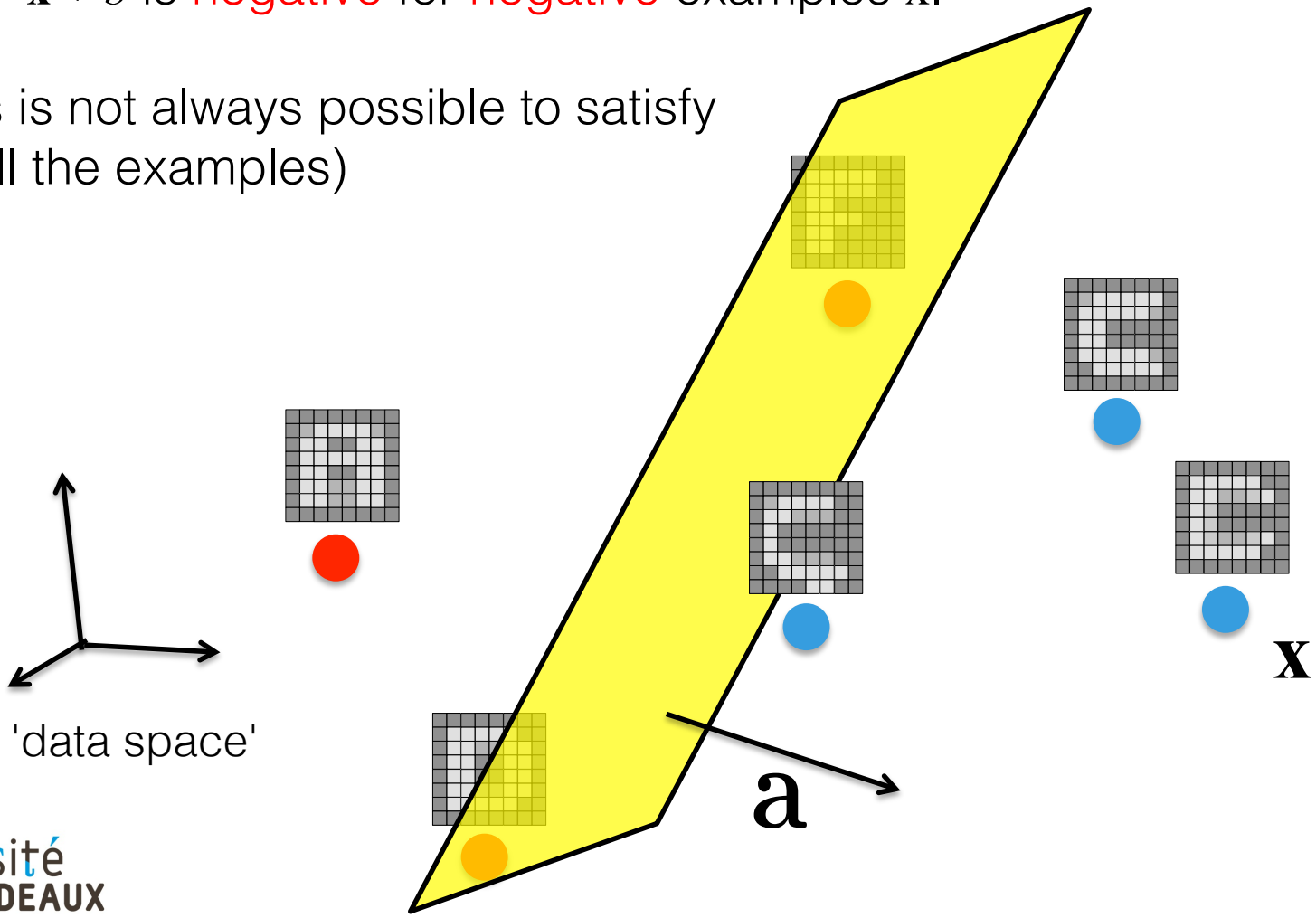
- $\mathbf{a}^\top \mathbf{x} + b$ is positive for positive examples $\mathbf{x}$;
- $\mathbf{a}^\top \mathbf{x} + b$ is negative for negative examples $\mathbf{x}$.



Perceptron: Idea

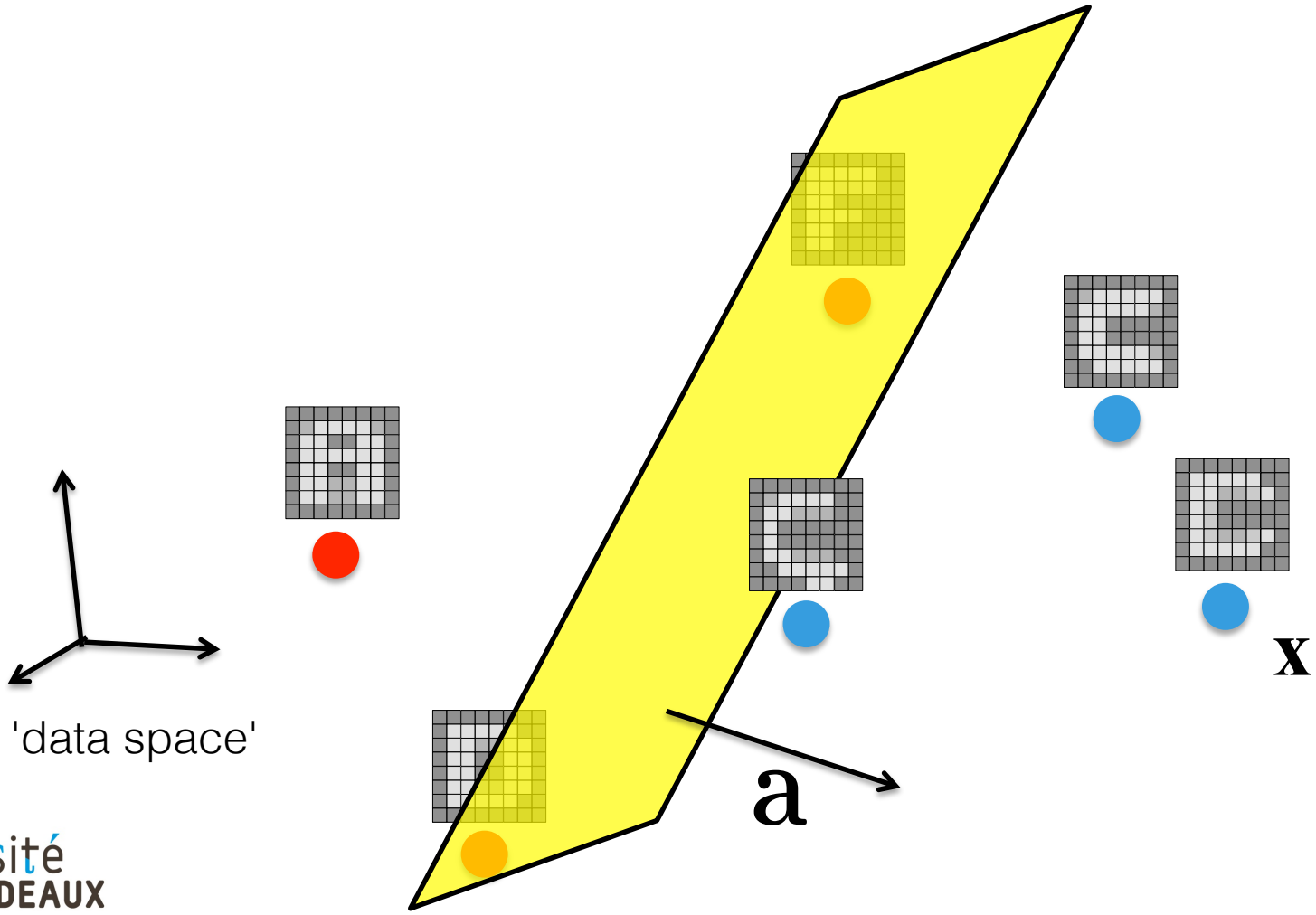
3. **Training**: find $\mathbf{a}$ and b so that
- $\mathbf{a}^\top \mathbf{x} + b$ is positive for positive examples $\mathbf{x}$;
- $\mathbf{a}^\top \mathbf{x} + b$ is negative for negative examples $\mathbf{x}$.

(This is not always possible to satisfy for all the examples)



Perceptron: Idea

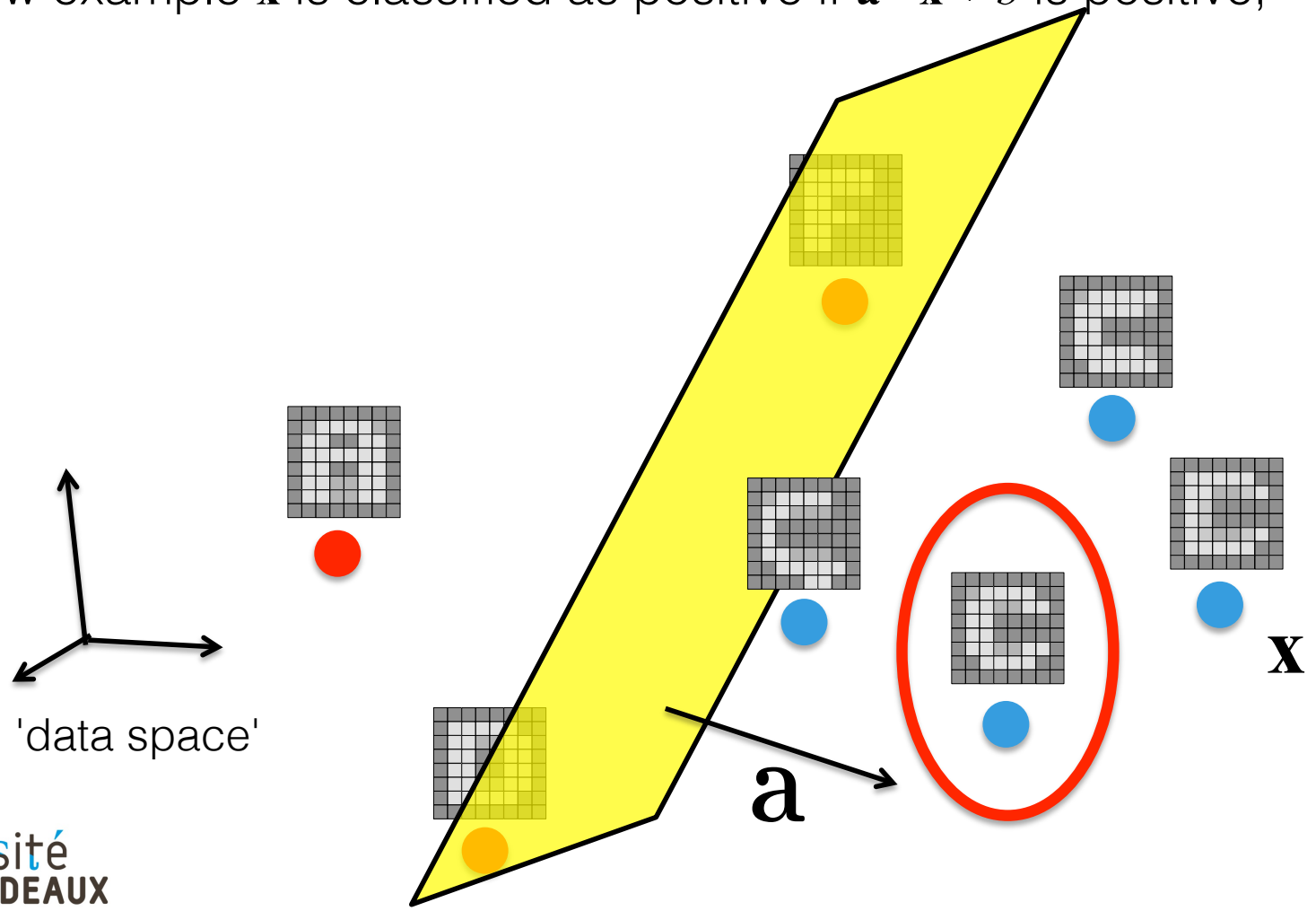
4. **Testing**: the perceptron can now classify new examples.



Perceptron: Idea

4. **Testing**: the perceptron can now classify new examples.

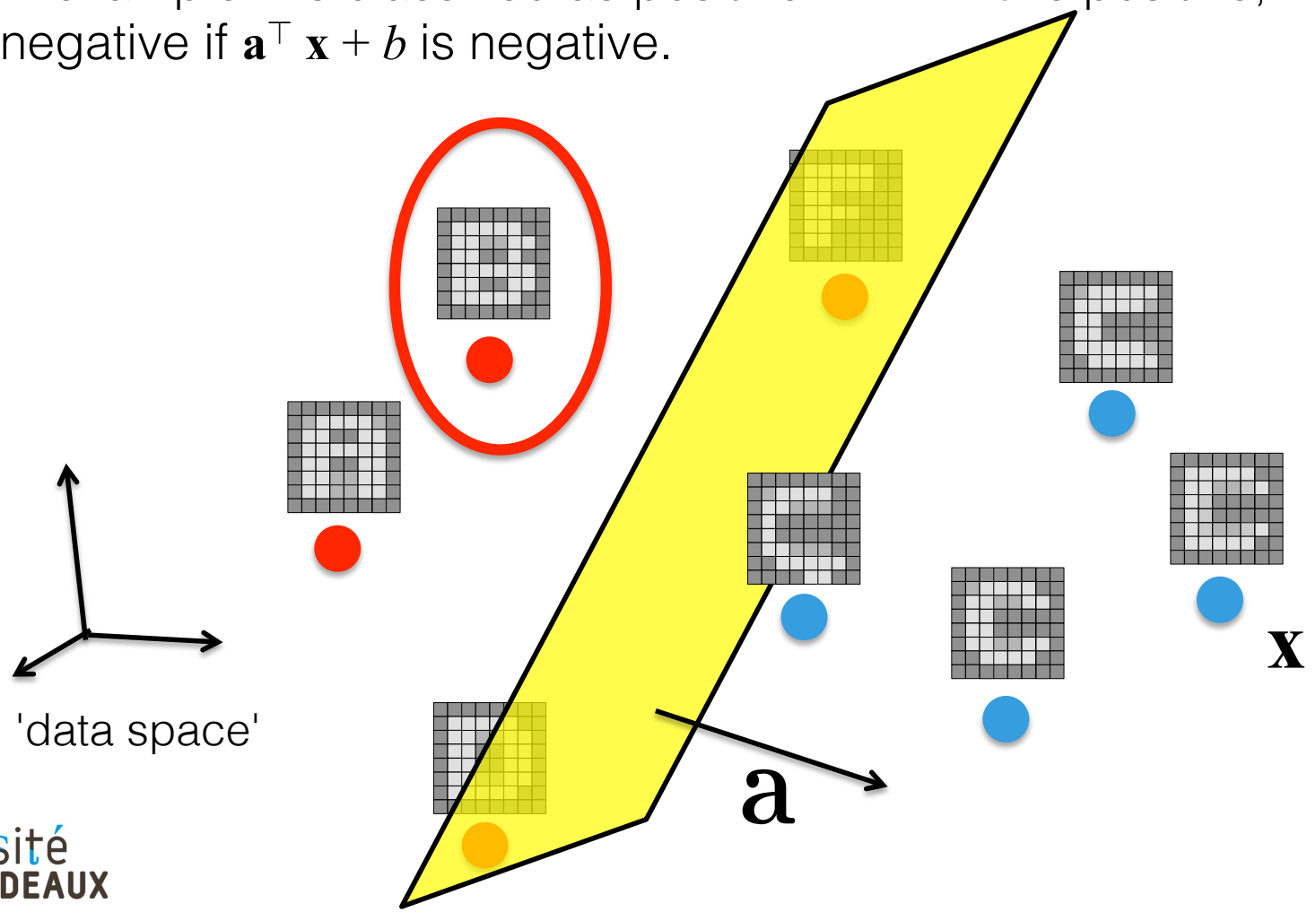
A new example $\mathbf{x}$ is classified as positive if $\mathbf{a}^\top \mathbf{x} + b$ is positive,



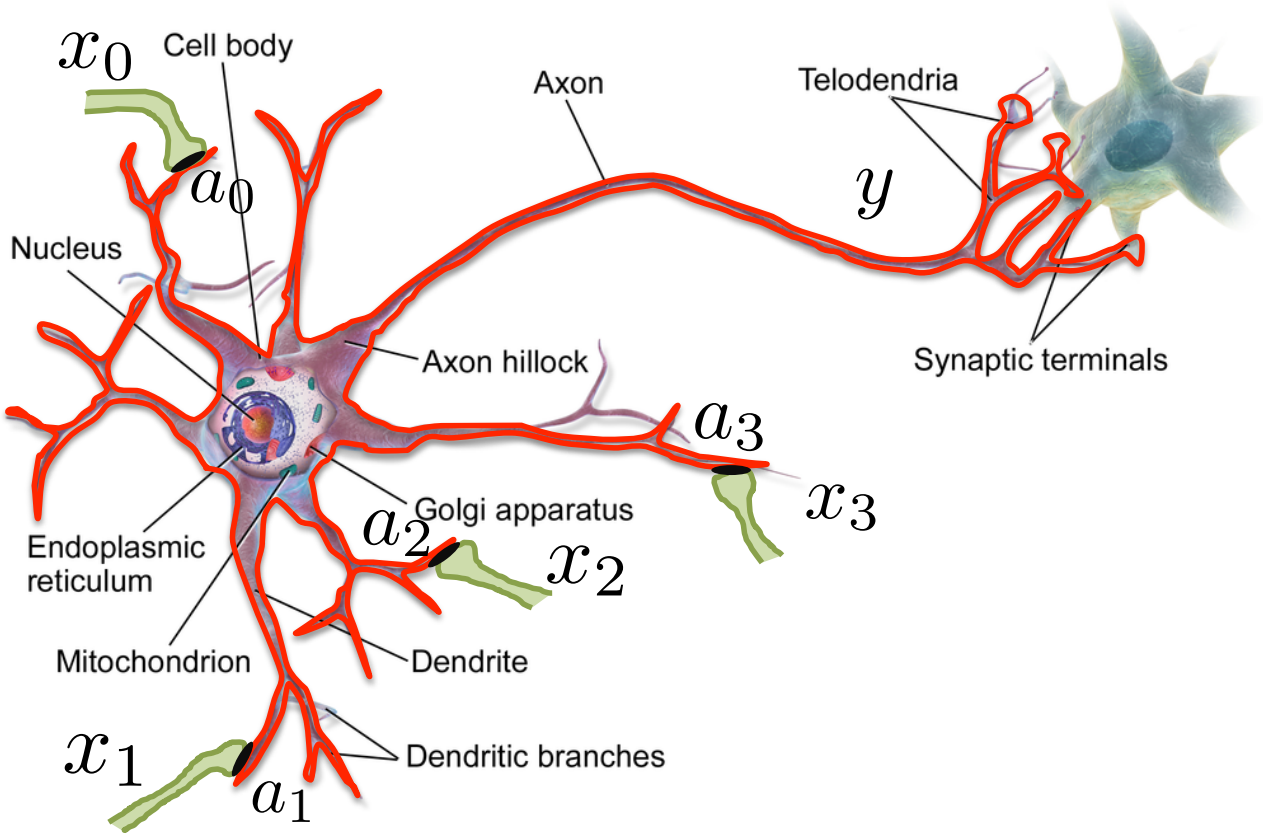
Perceptron: Idea

4. **Testing**: the perceptron can now classify new examples.

A new example $\mathbf{x}$ is classified as positive if $\mathbf{a}^\top \mathbf{x} + b$ is positive, and negative if $\mathbf{a}^\top \mathbf{x} + b$ is negative.



The Inspiration for the Perceptron

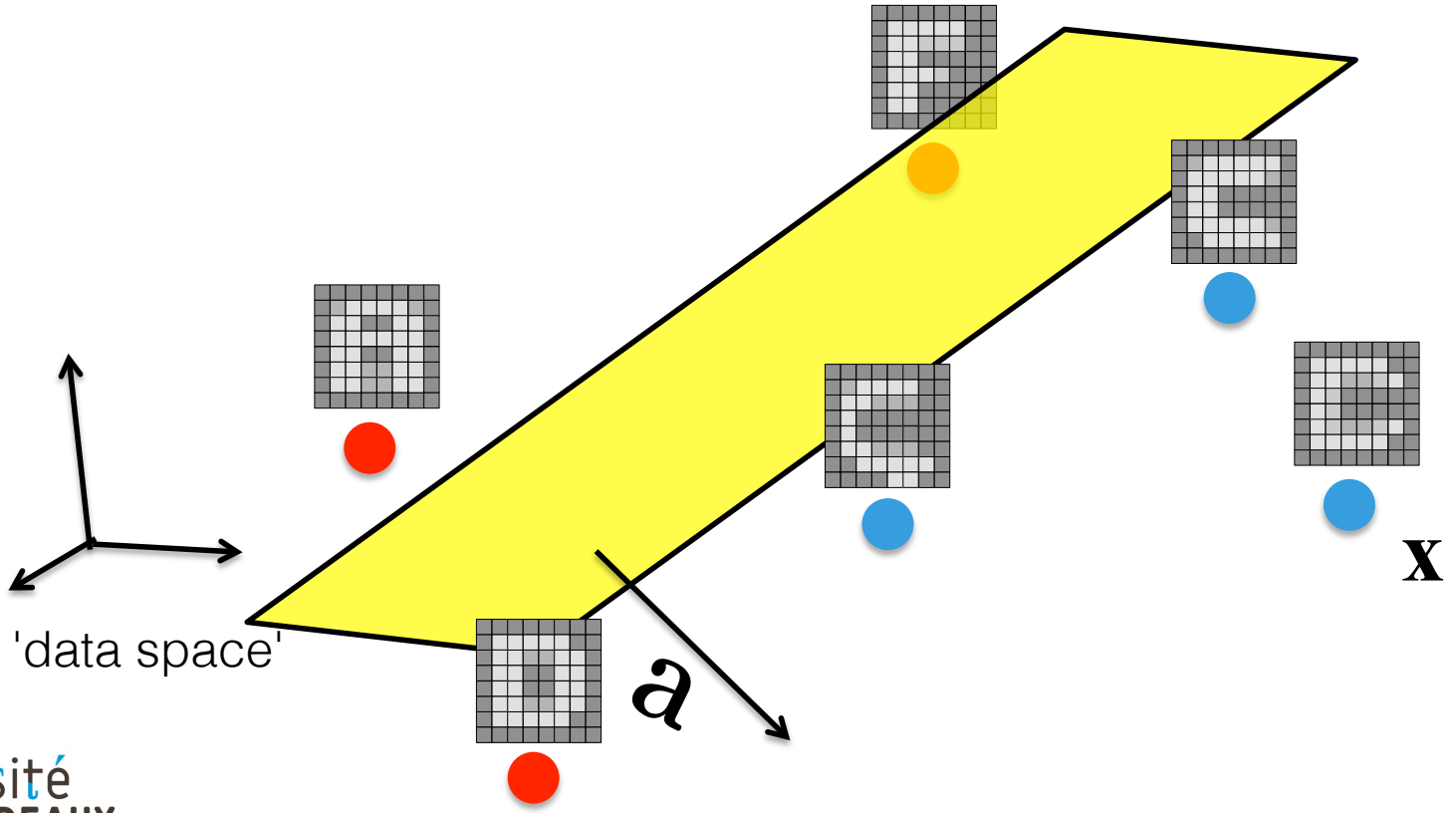


$$
\begin{aligned}
y &= \text{sign}(a_0 x_0 + a_1 x_1 + a_2 x_2 + a_3 x_3) \\
&= \text{sign}(\mathbf{a}^\top \mathbf{x})
\end{aligned}
$$

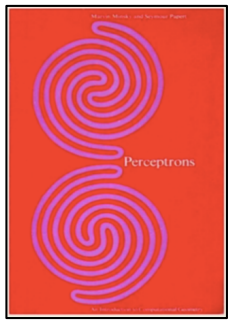
Perceptron: How Can We Find $\mathbf{a}$ and b?

At the time, *ad hoc* algorithm:

1. Start from a random initialization;
2. For each training example $\mathbf{x}$:
   - compare the value of $\mathbf{a}^\top \mathbf{x} + b$ and its expected sign,
   - adapt $\mathbf{a}$ and b to get a better value for $\mathbf{a}^\top \mathbf{x} + b$.

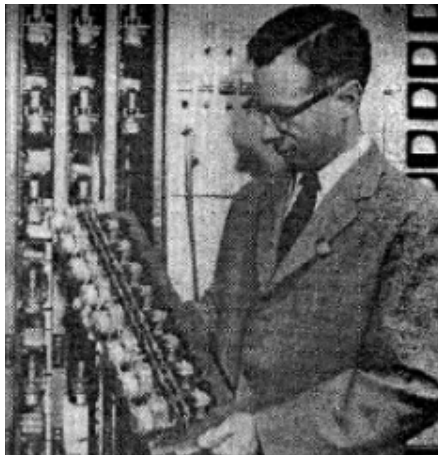


History of [Deep] Learning



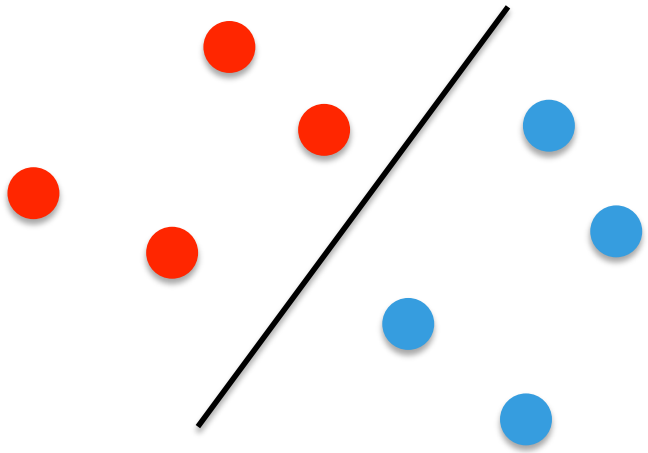
1969: *Perceptrons* book,
Minsky and Papert

1958, Cornell:
Perceptron

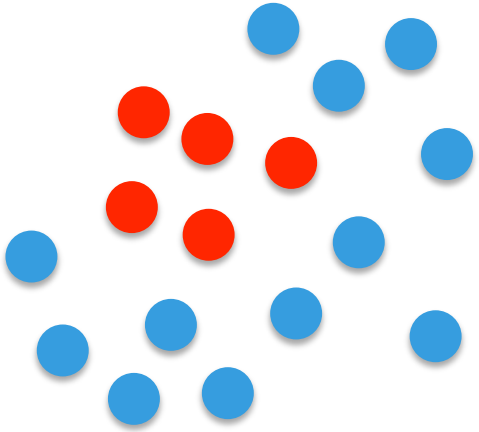


Perceptrons book (1969)

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



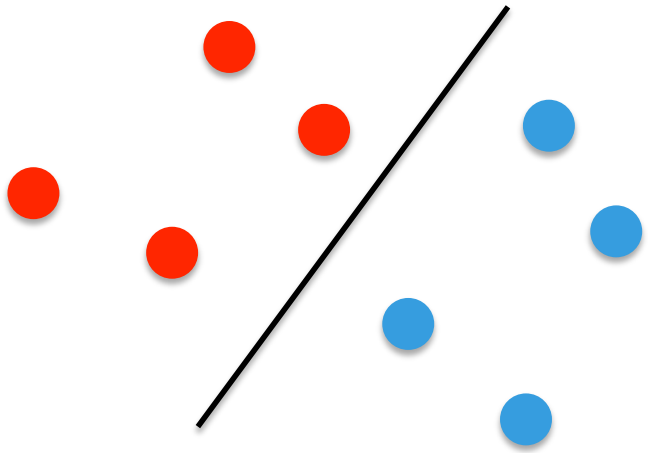
linearly separable



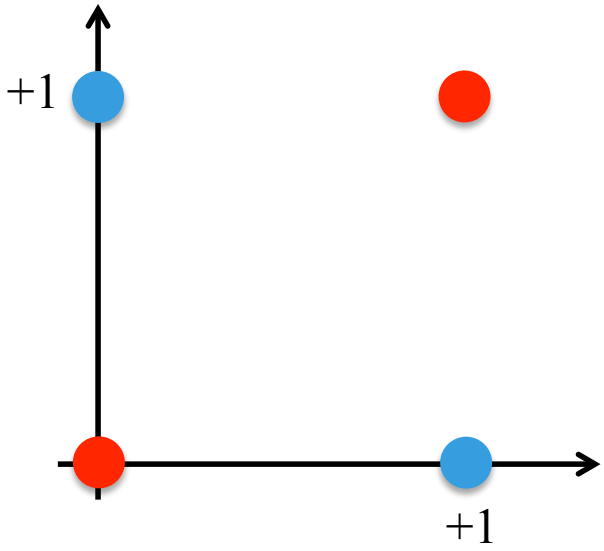
nonlinearly separable

Perceptrons book (1969)

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



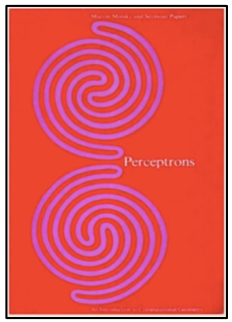
linearly separable



nonlinearly separable:
the case of the x-or function

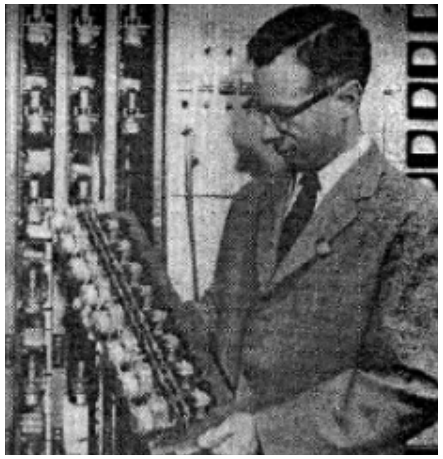
Seen by many as a justification to stop research on perceptrons.

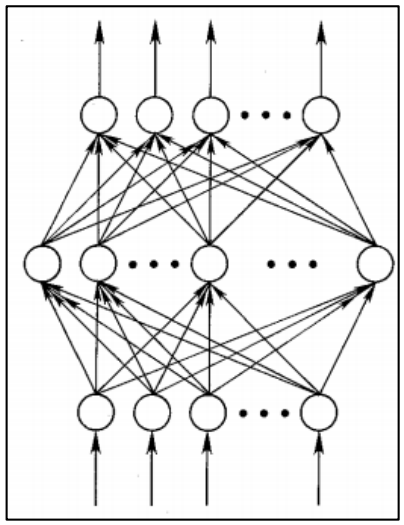
History of [Deep] Learning



1969: *Perceptrons* book, Minsky and Papert

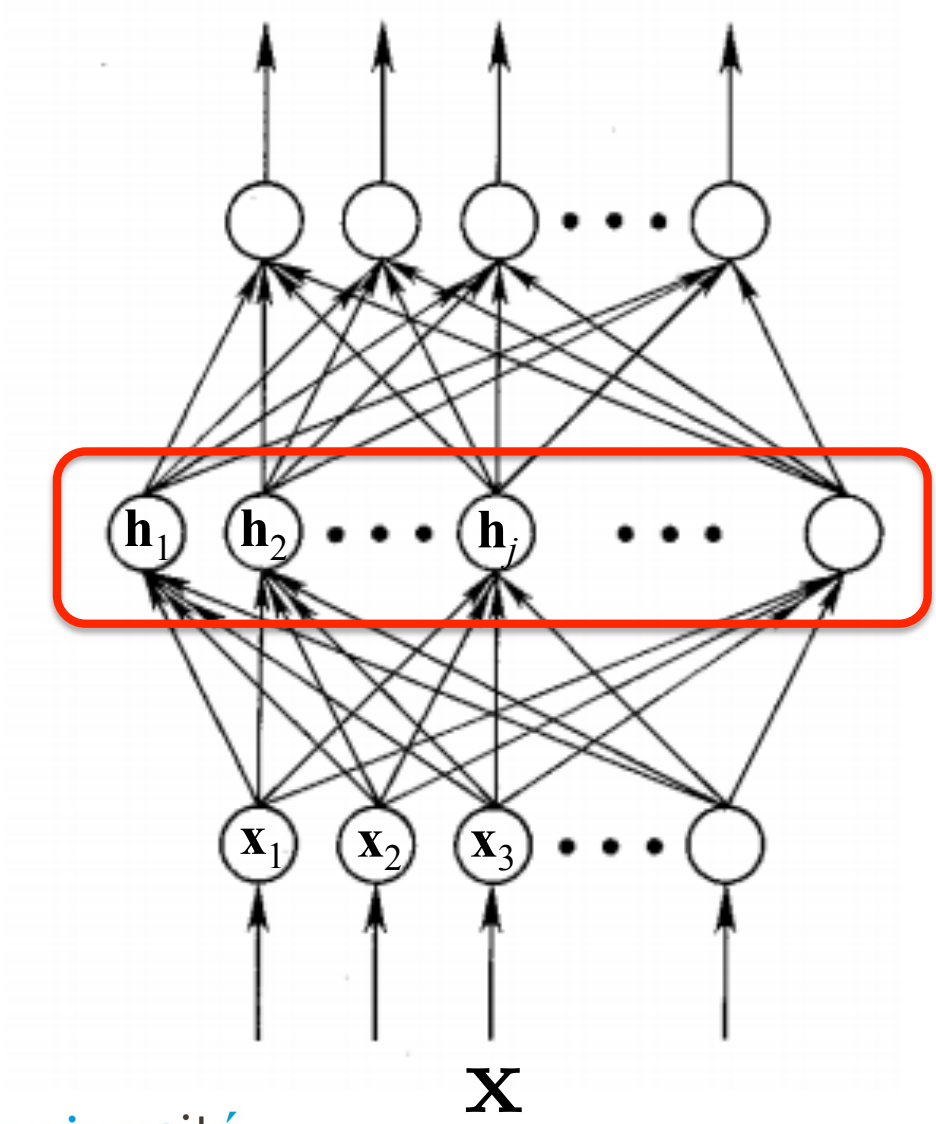
1958, Cornell: Perceptron



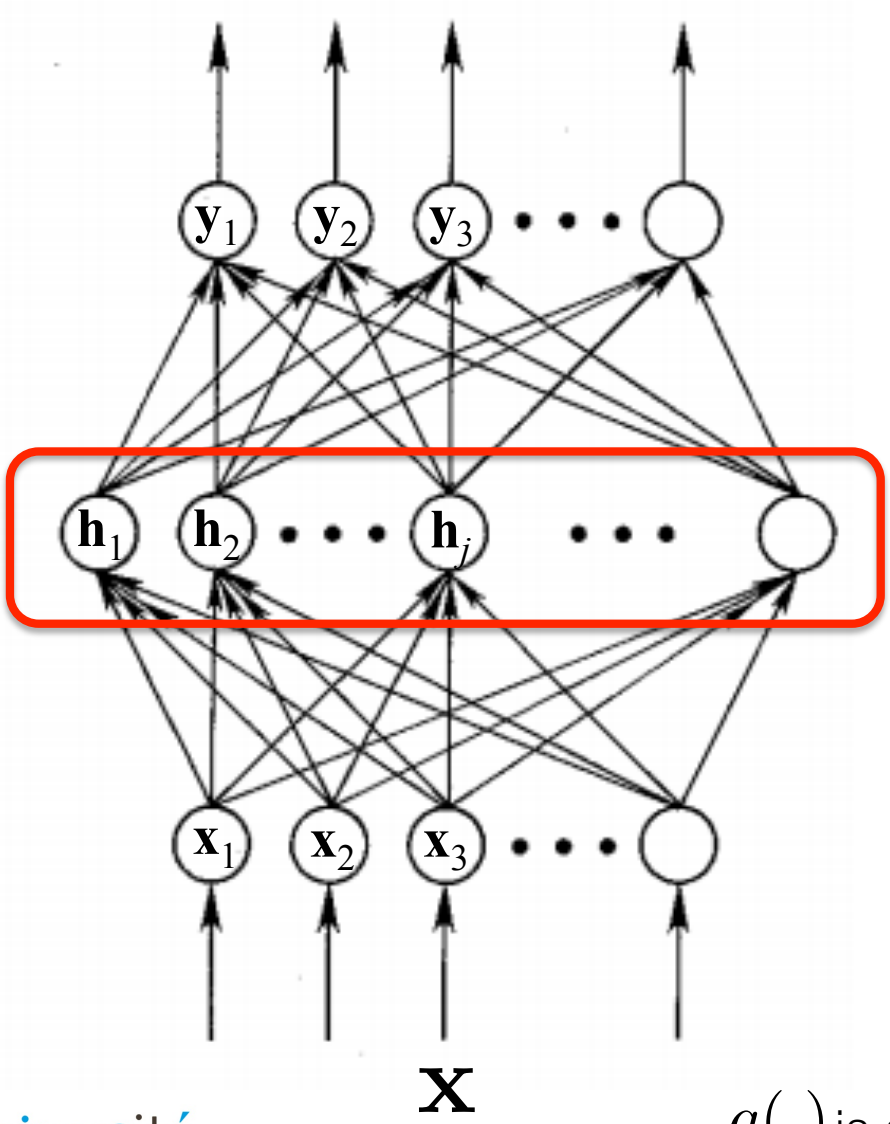


~1980: Multi-layer networks (Rumelhart, Hinton, Williams)

Multilayer Perceptron/Neural Network

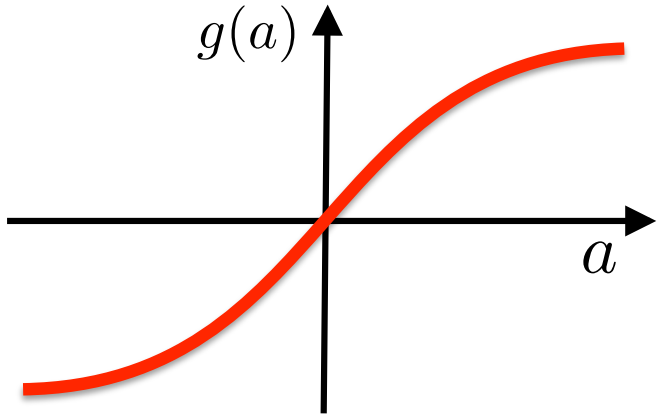


Multilayer Perceptron/Neural Network



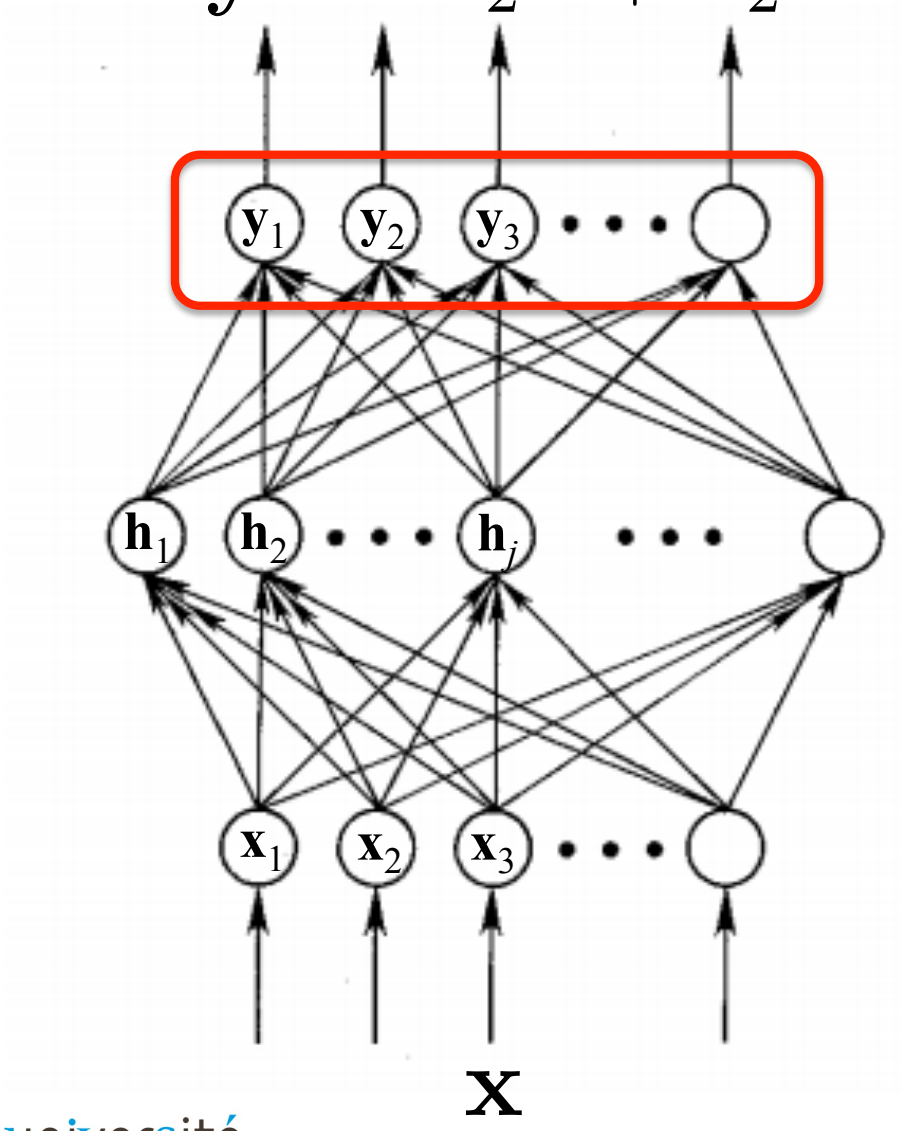
$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$g(.)$: any 'squashing' function, for example:



$g(.)$ is applied to each coefficient of $\mathbf{W}\mathbf{x} + \mathbf{b}$

$$\mathbf{y} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$



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

Multilayer Perceptron

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y}(\mathbf{x}) = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

How can we find $\mathbf{W}$, $\mathbf{b}$, $\mathbf{W}_2$, and $\mathbf{b}_2$?

Multilayer Perceptron: Optimization

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y}(\mathbf{x}) = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

How can we find $\mathbf{W}$, $\mathbf{b}$, $\mathbf{W}_2$, and $\mathbf{b}_2$?

[Rumelhart, Hinton, Williams] introduces an objective (or loss) function:

$$\mathcal{L}(\mathcal{T}) = \sum_{(\mathbf{x},\mathbf{d})\in\mathcal{T}} \|\mathbf{y}(\mathbf{x}) - \mathbf{d}\|^2$$

Multilayer Perceptron: Optimization

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y}(\mathbf{x}) = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

How can we find $\mathbf{W}$, $\mathbf{b}$, $\mathbf{W}_2$, and $\mathbf{b}_2$?

[Rumelhart, Hinton, Williams] introduces an objective (or loss) function:

$$\mathcal{L}(\mathcal{T}) = \sum_{(\mathbf{x},\mathbf{d})\in\mathcal{T}} \|\mathbf{y}(\mathbf{x}) - \mathbf{d}\|^2$$

$\mathcal{T}$: training set

Multilayer Perceptron: Optimization

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y}(\mathbf{x}) = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

How can we find $\mathbf{W}$, $\mathbf{b}$, $\mathbf{W}_2$, and $\mathbf{b}_2$?

[Rumelhart, Hinton, Williams] introduces an objective (or loss) function:

$$\mathcal{L}(\mathcal{T}) = \sum_{(\mathbf{x},\mathbf{d})\in\mathcal{T}} \|\mathbf{y}(\mathbf{x}) - \mathbf{d}\|^2$$

$\mathcal{T}$: training set

$\mathbf{x}$: 1 training example;
$\mathbf{d}$: the desired output for $\mathbf{x}$.

Multilayer Perceptron: Optimization

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y}(\mathbf{x}) = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

How can we find $\mathbf{W}$, $\mathbf{b}$, $\mathbf{W}_2$, and $\mathbf{b}_2$?

[Rumelhart, Hinton, Williams] introduces an objective (or loss) function:

$$\mathcal{L}(\mathcal{T}) = \sum_{(\mathbf{x},\mathbf{d})\in\mathcal{T}} \|\mathbf{y}(\mathbf{x}) - \mathbf{d}\|^2$$

$\mathcal{T}$: training set

$\mathbf{x}$: 1 training example;
$\mathbf{d}$: the desired output for $\mathbf{x}$.

$\mathbf{y}(\mathbf{x})$: network output for $\mathbf{x}$

Multilayer Perceptron: Optimization

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y}(\mathbf{x}) = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

How can we find $\mathbf{W}$, $\mathbf{b}$, $\mathbf{W}_2$, and $\mathbf{b}_2$?

[Rumelhart, Hinton, Williams] introduces an objective (or loss) function:

$$\mathcal{L}(\mathcal{T}) = \sum_{(\mathbf{x},\mathbf{d})\in\mathcal{T}} \|\mathbf{y}(\mathbf{x}) - \mathbf{d}\|^2$$

Looks for $\mathbf{W}$, $\mathbf{b}$, $\mathbf{W}_2$, and $\mathbf{b}_2$ that minimize the

$$(\hat{\mathbf{W}}, \hat{\mathbf{b}}, \hat{\mathbf{W}}_2, \hat{\mathbf{b}_2}) = \underset{(\mathbf{W},\mathbf{b},\mathbf{W}_2,\mathbf{b}_2)}{\arg\min} \sum_{(\mathbf{x},\mathbf{d})\in\mathcal{T}} \|\mathbf{y}(\mathbf{x}) - \mathbf{d}\|^2$$

Multilayer Perceptron: Optimization

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y}(\mathbf{x}) = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

How can we find $\mathbf{W}$, $\mathbf{b}$, $\mathbf{W}_2$, and $\mathbf{b}_2$?

[Rumelhart, Hinton, Williams] introduces an objective (or loss) function:

Looks for $\mathbf{W}$, $\mathbf{b}$, $\mathbf{W}_2$, and $\mathbf{b}_2$ that minimize the

$$(\hat{\mathbf{W}}, \hat{\mathbf{b}}, \hat{\mathbf{W}}_2, \hat{\mathbf{b}_2}) = \underset{(\mathbf{W},\mathbf{b},\mathbf{W}_2,\mathbf{b}_2)}{\arg\min} \sum_{(\mathbf{x},\mathbf{d})\in\mathcal{T}} \|\mathbf{y}(\mathbf{x}) - \mathbf{d}\|^2$$

Supervised Machine Learning becomes regression!

(In practice, do NOT use a least-squares loss function for classification problems!)

Multilayer Perceptron: Optimization

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y}(\mathbf{x}) = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

$$(\hat{\mathbf{W}}, \hat{\mathbf{b}}, \hat{\mathbf{W}}_2, \hat{\mathbf{b}_2}) = \underset{(\mathbf{W},\mathbf{b},\mathbf{W}_2,\mathbf{b}_2)}{\arg\min} \sum_{(\mathbf{x},\mathbf{d})\in\mathcal{T}} \|\mathbf{y}(\mathbf{x}) - \mathbf{d}\|^2$$

How can we solve this optimization problem?

Multilayer Perceptron: Optimization

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y}(\mathbf{x}) = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

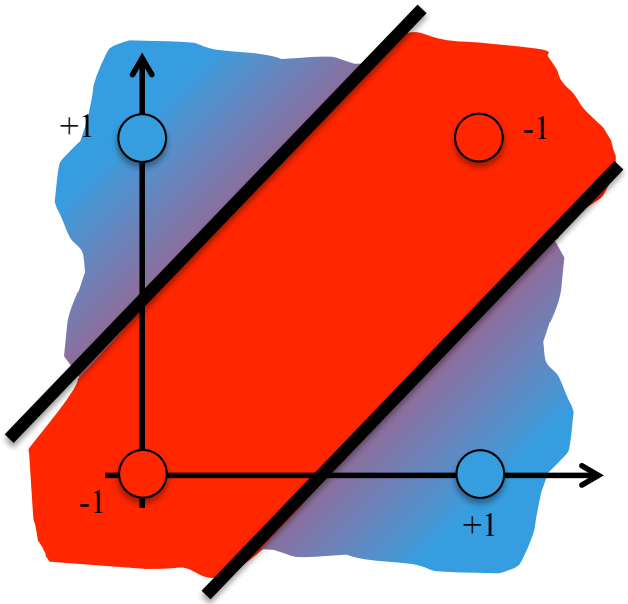
$$(\hat{\mathbf{W}}, \hat{\mathbf{b}}, \hat{\mathbf{W}}_2, \hat{\mathbf{b}_2}) = \underset{(\mathbf{W},\mathbf{b},\mathbf{W}_2,\mathbf{b}_2)}{\arg\min} \sum_{(\mathbf{x},\mathbf{d})\in\mathcal{T}} \|\mathbf{y}(\mathbf{x}) - \mathbf{d}\|^2$$

How can we solve this optimization problem?

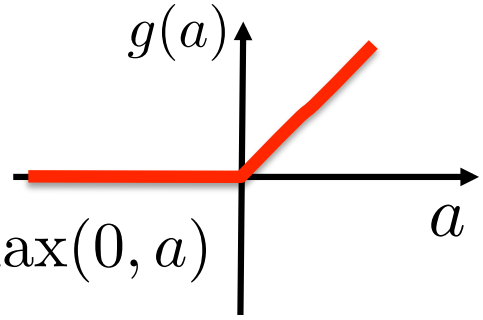
No closed-form method → gradient descent.

A Multilayer Perceptron Solves Non-Linearly Separable Problems

the x-or function



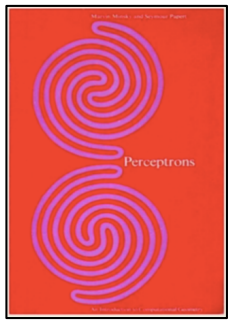
$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$y(\mathbf{x}) = \mathbf{w}_2^\top \mathbf{h} + b_2$$



$$g(a) = \max(0, a)$$

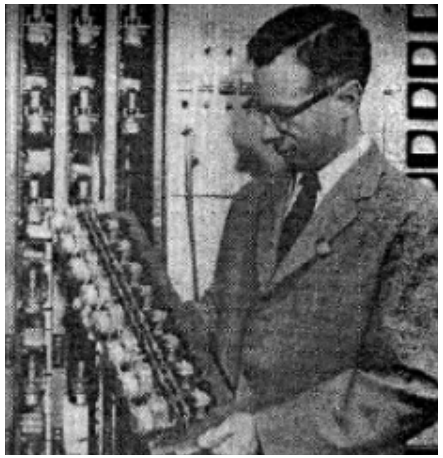
$$\mathbf{W} = \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} -1/2 \\ -1/2 \end{bmatrix}, \mathbf{w}_2 = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, b_2 = -1$$

History of [Deep] Learning

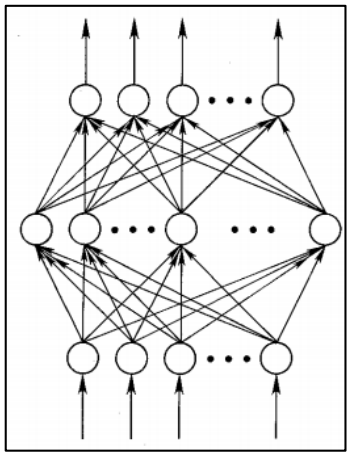


1969: *Perceptrons* book, Minsky and Papert

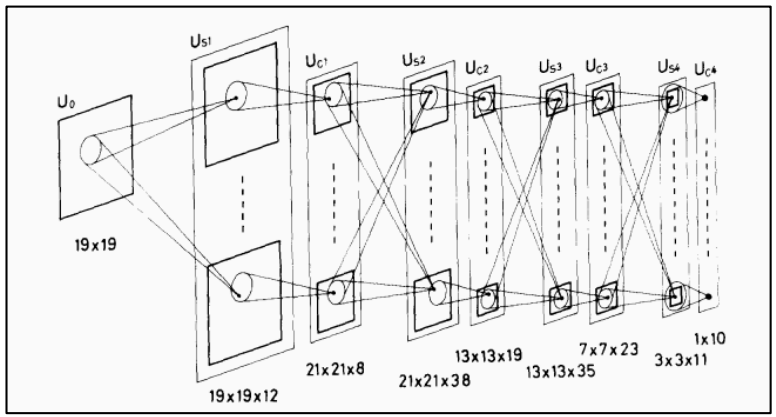
1958, Cornell: Perceptron



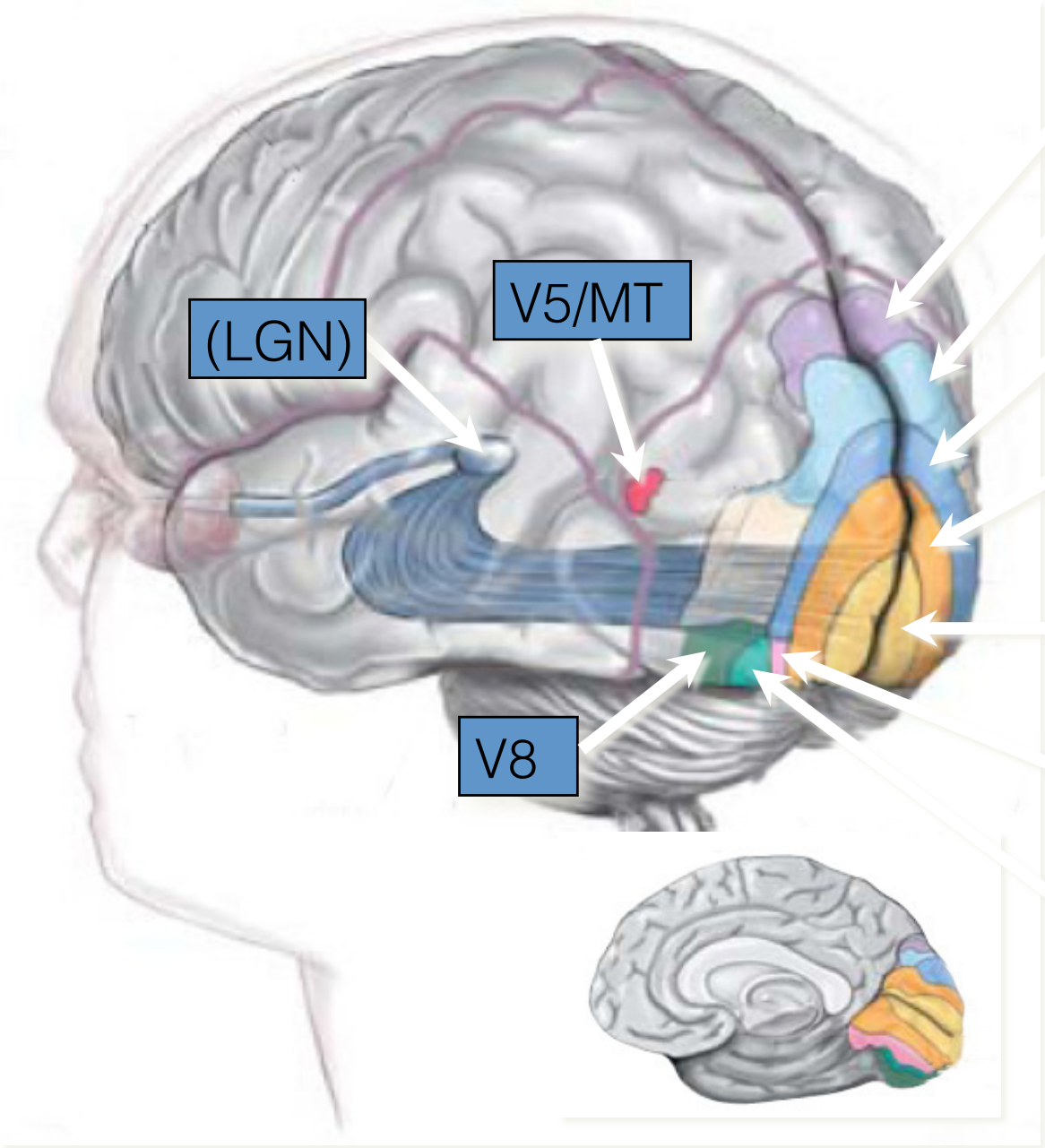
~1980: Multi-layer networks



1987: Neocognitron, Fukushima



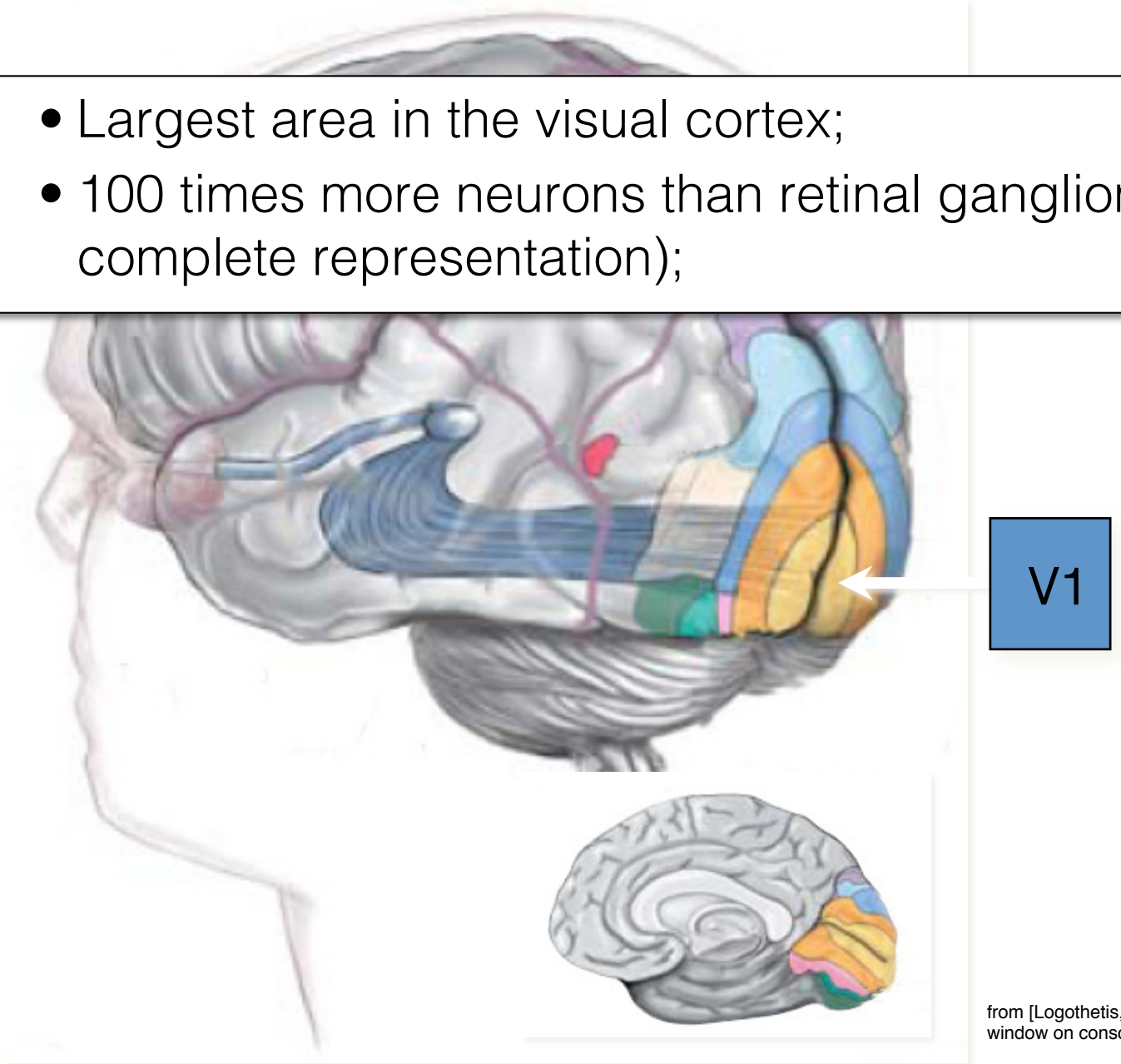
The Visual Cortex



from [Logothetis, N., November 1999. Vision: A window on consciousness. Scientific American]

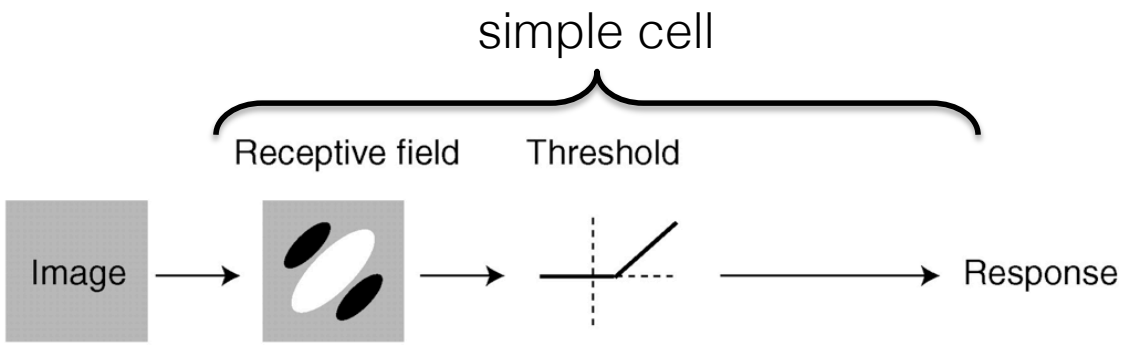
V1, the Primary Visual Cortex

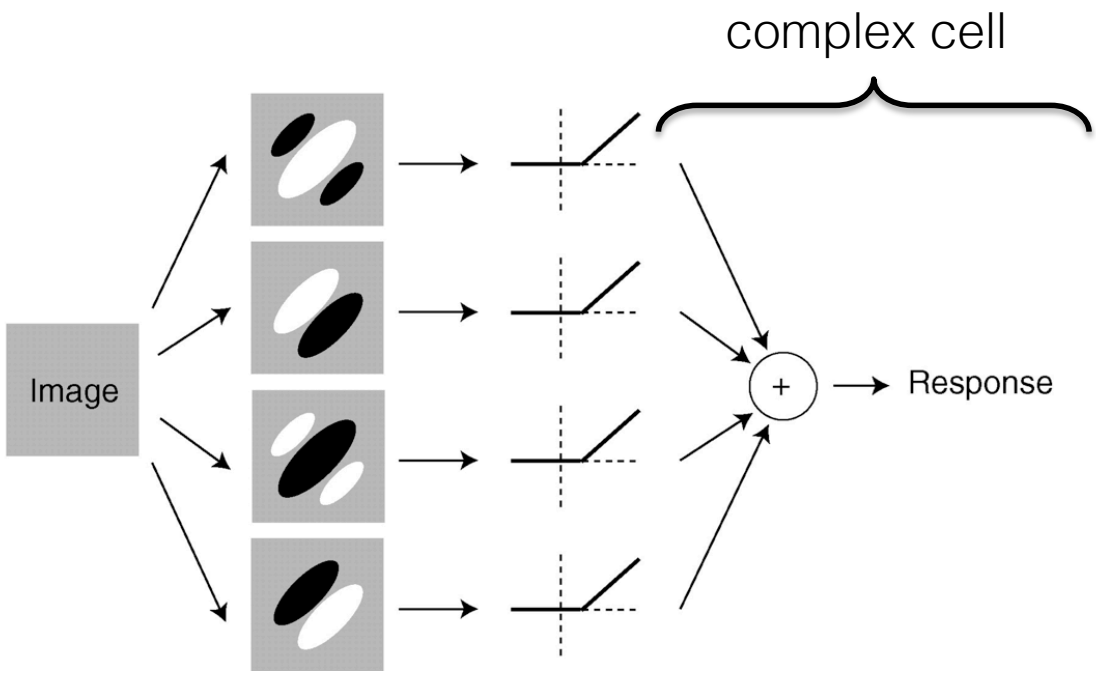
- Largest area in the visual cortex;
- 100 times more neurons than retinal ganglion cells (over-complete representation);



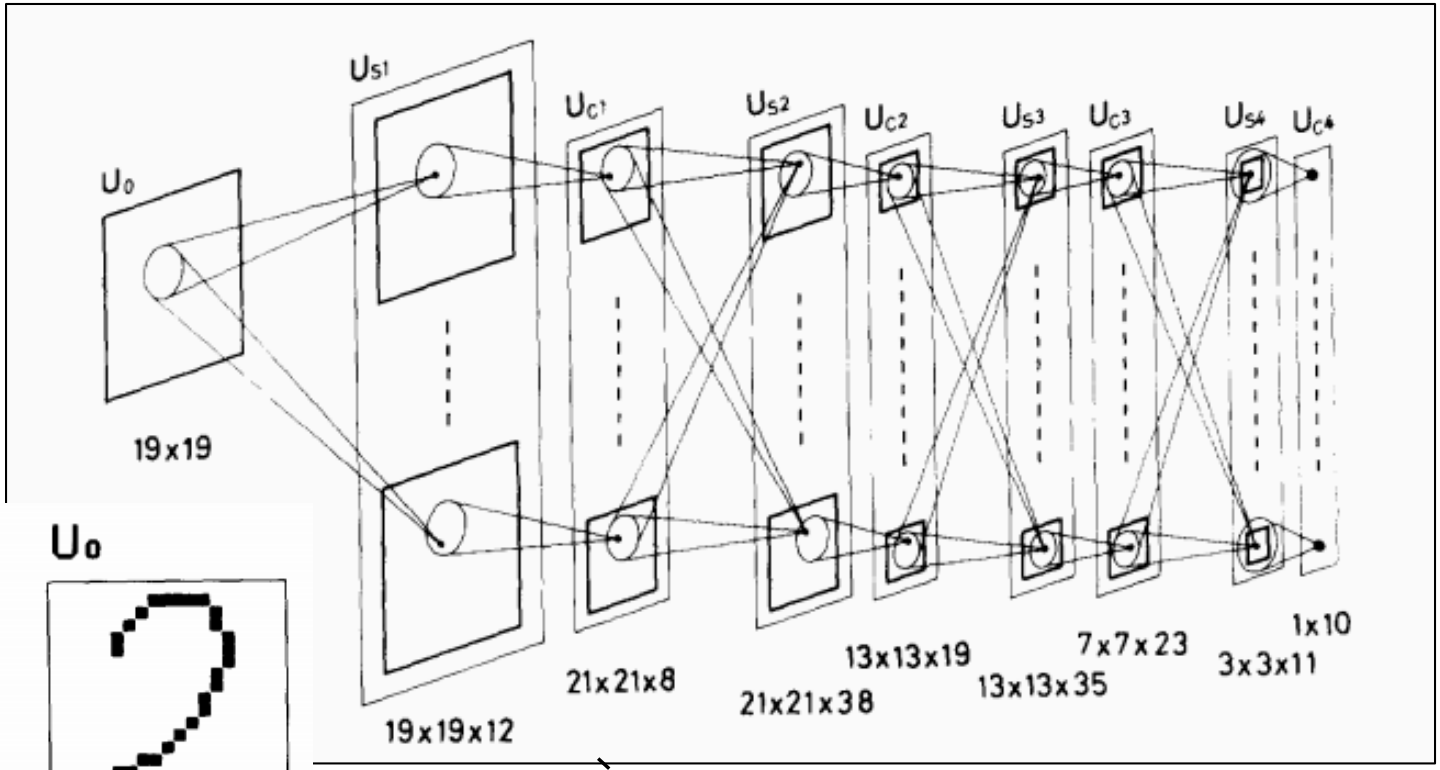
from [Logothetis, N., November 1999. Vision: A window on consciousness. Scientific American]

Hubel and Wiesel (1959)





Neocognitron (1987)



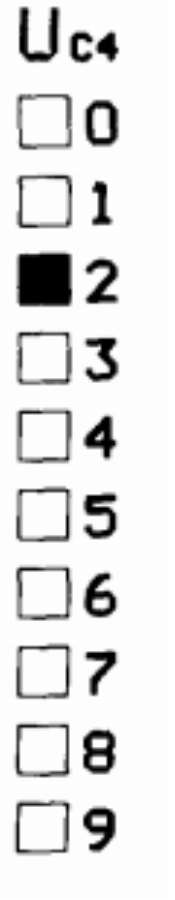
'simple cells'

'complex cells'

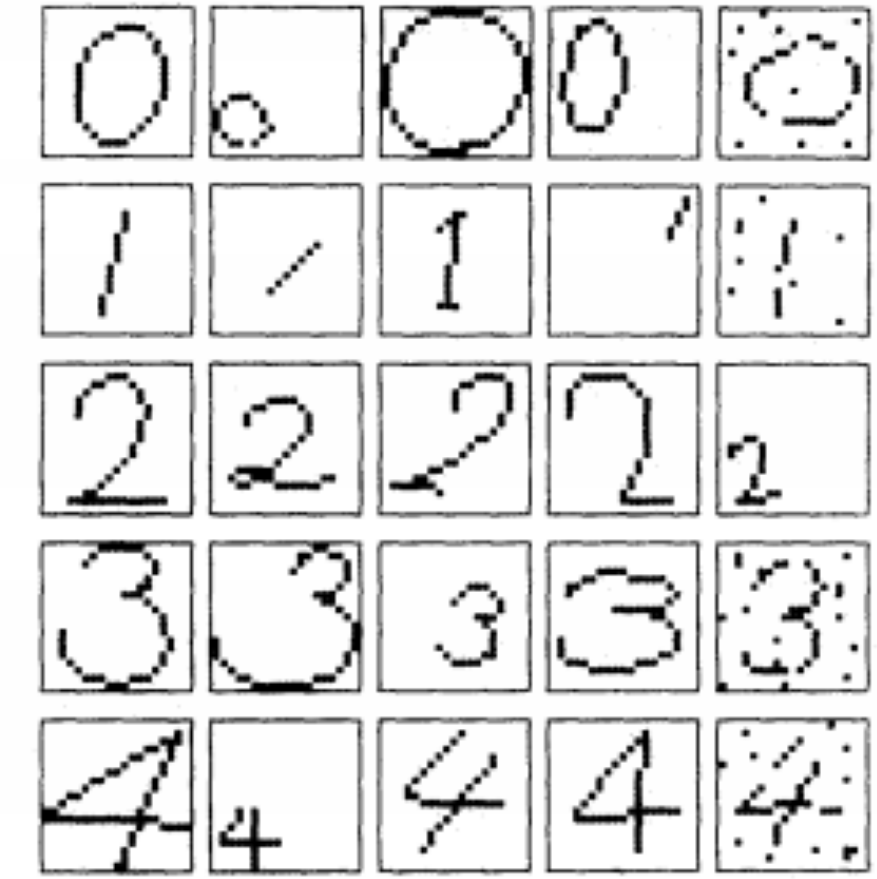
'simple cells'

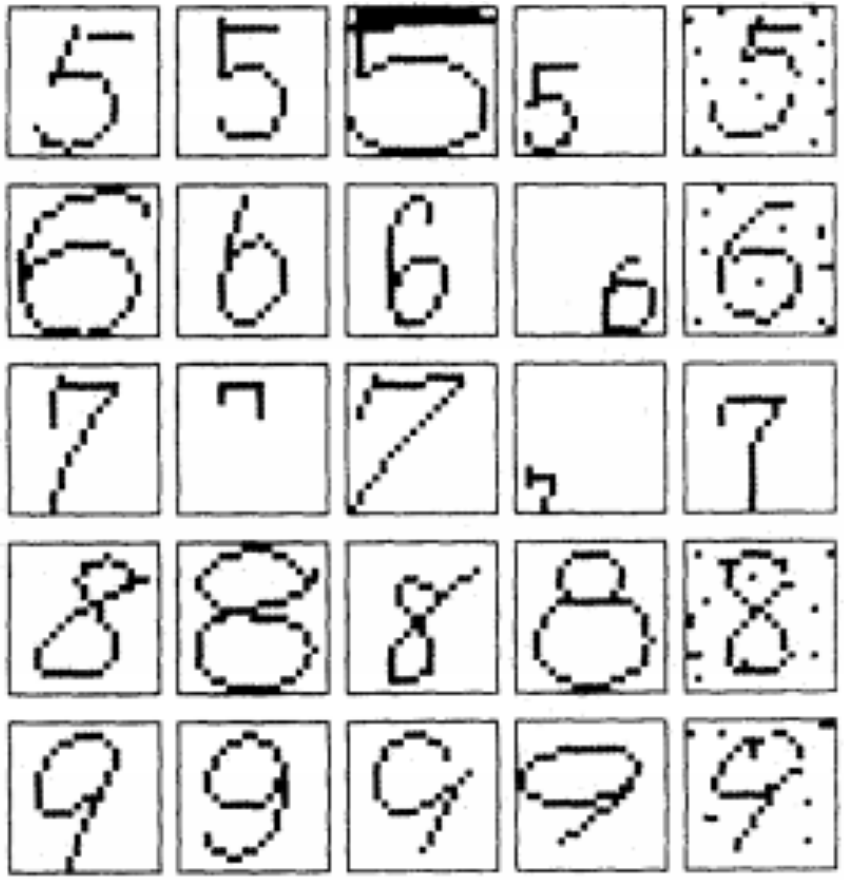
'complex cells'

10 outputs



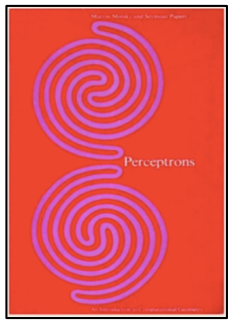
Neocognitron (1987)





Some digits correctly recognized

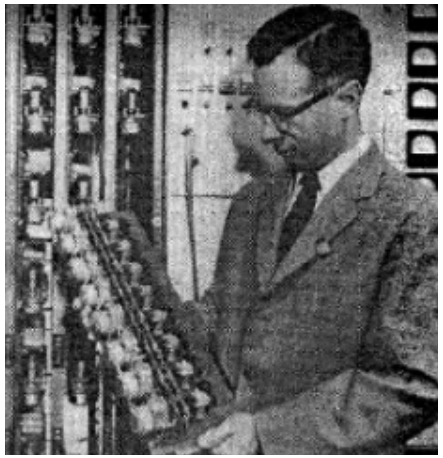
History of [Deep] Learning



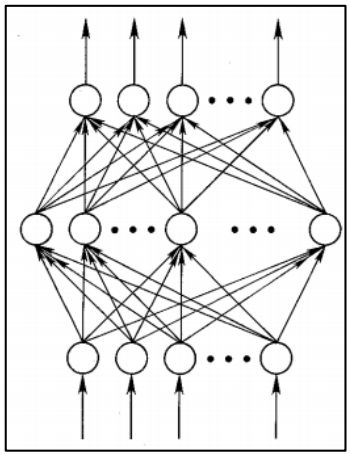
1969: *Perceptrons* book, Minsky and Papert

1989: Universal Approximation Theorem (Hornik, Cybenko)

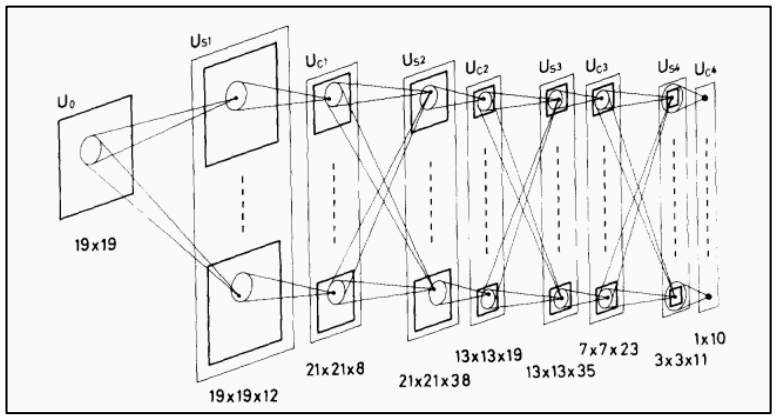
1958, Cornell: Perceptron



~1980: Multi-layer networks

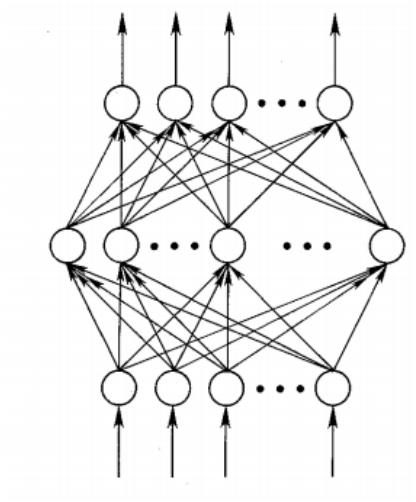


1987: Neocognitron, Fukushima



Universal Approximation Theorem [Hornik et al, 1989; Cybenko, 1989]

Proves that any continuous function can be approximated by a two-layer network:



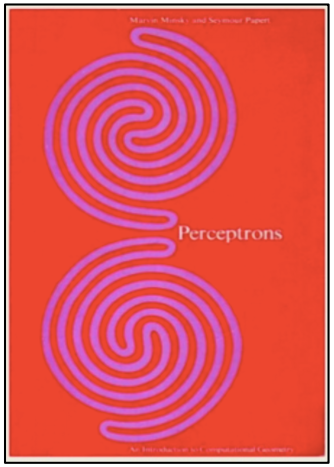
$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y} = \mathbf{W}_2^\top \mathbf{h} + \mathbf{b}_2$$

Universal Approximation: Actually…

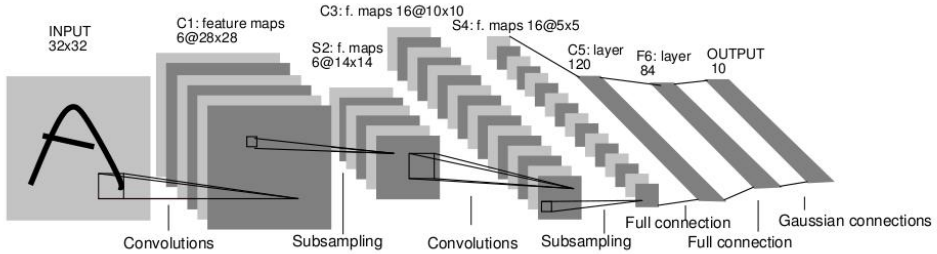
The theorem does not say how large the network needs to be;

No guarantee that the training algorithm will be able to train the network.

History of [Deep] Learning

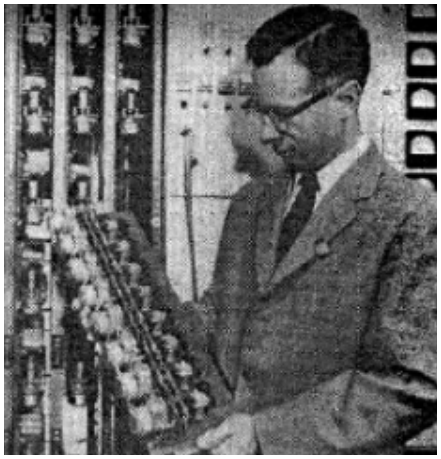


1969: *Perceptrons* book, Minsky and Papert

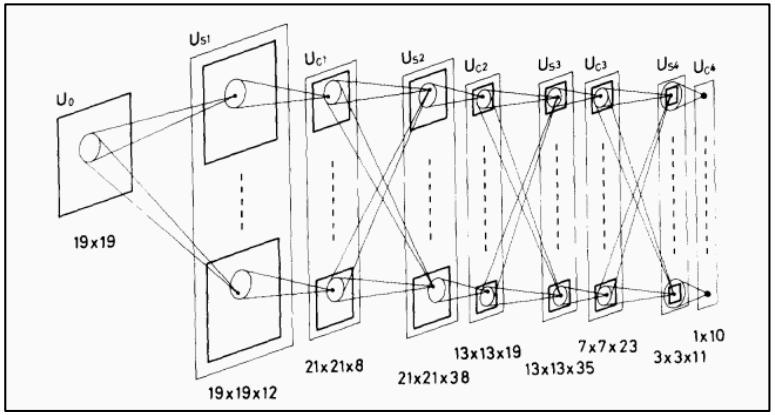


1992: LeNet, LeCun

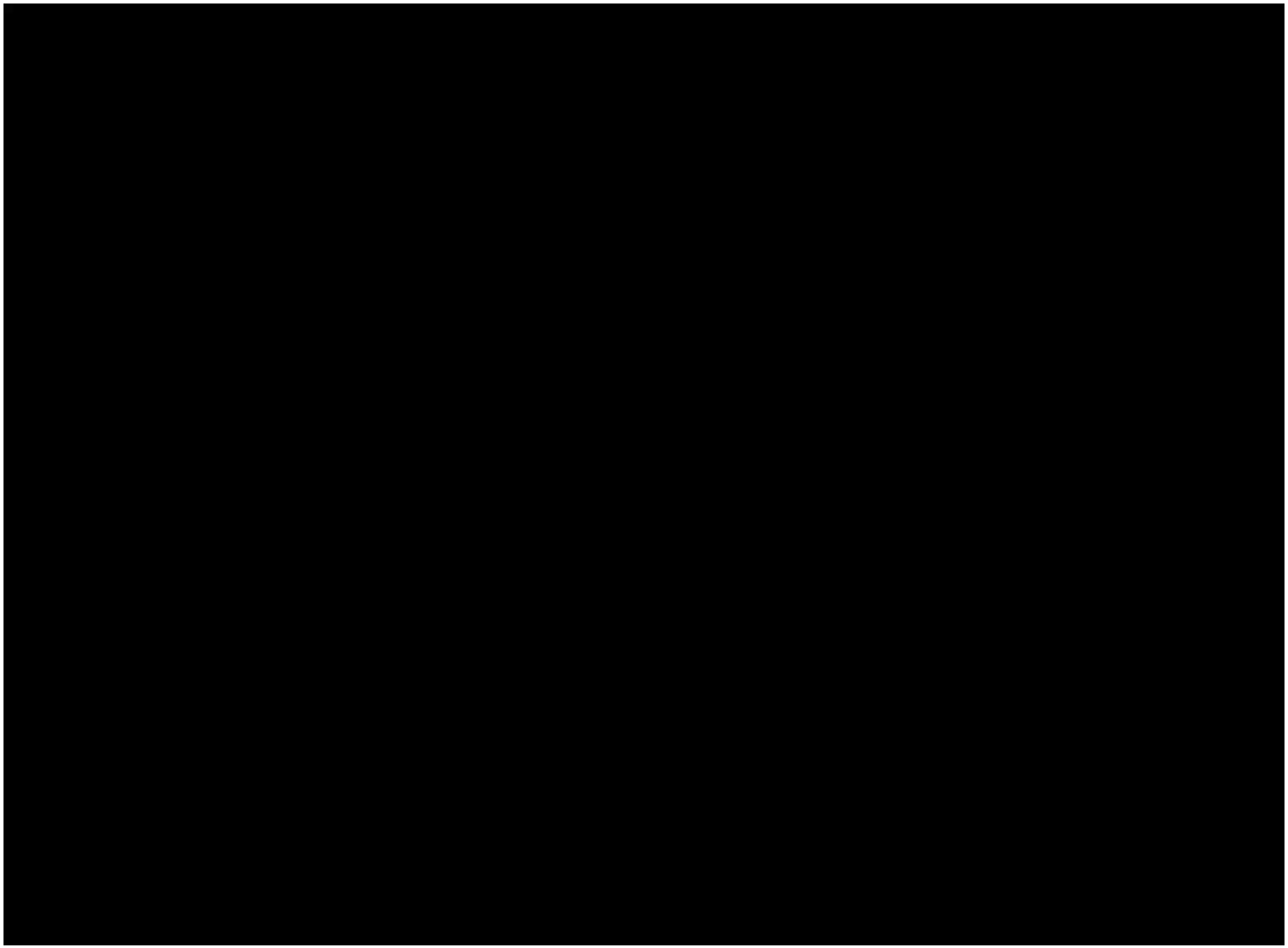
1958, Cornell: Perceptron



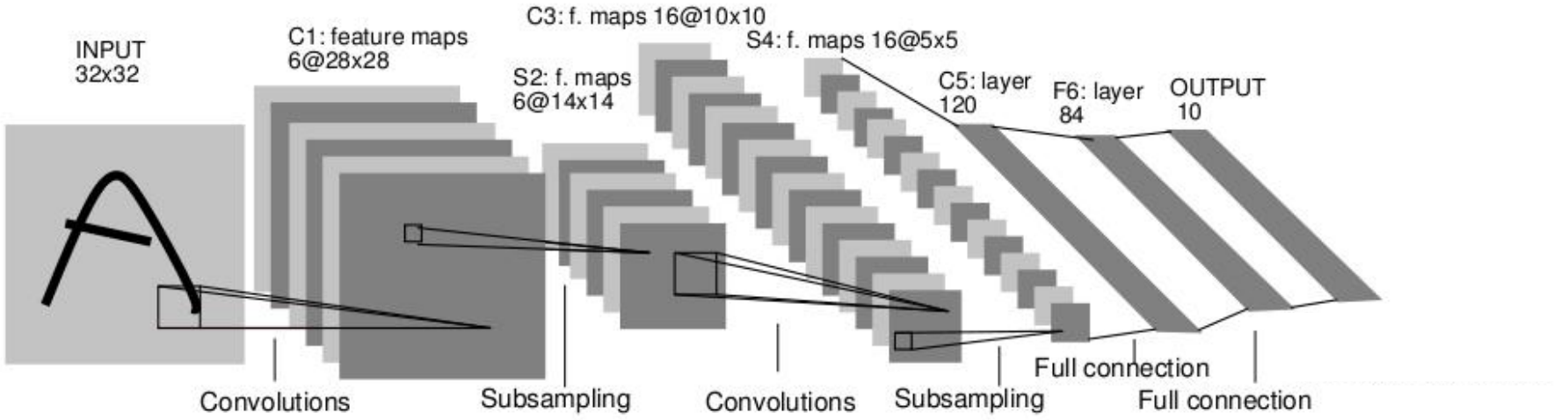
1987: Neocognitron, Fukushima



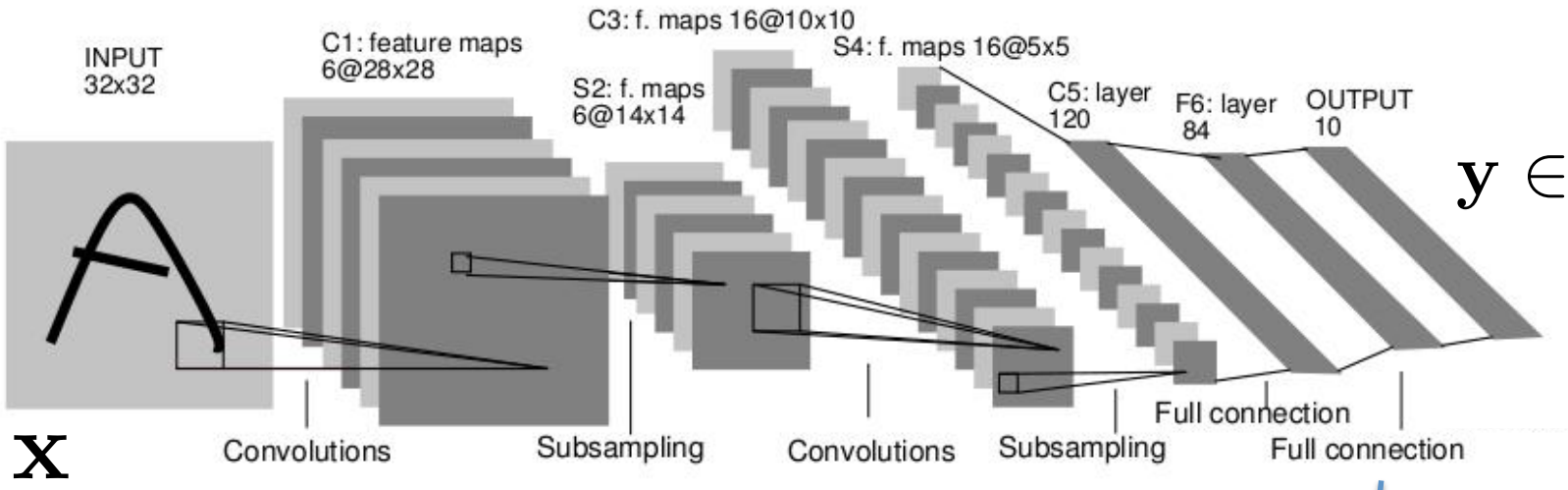
LeNet5 (LeCun, 1992)



Deep Learning *a la* LeCun



Deep Learning *a la* LeCun



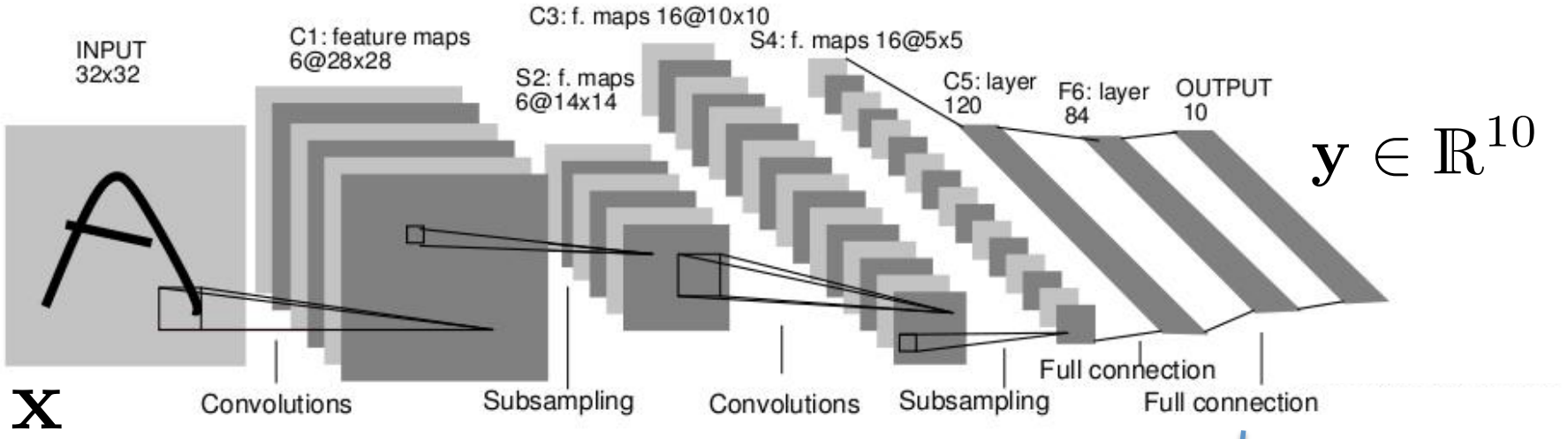
$\mathbf{x}$

$\mathbf{y} \in \mathbb{R}^{10}$

Fully connected layer:

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

Deep Learning *a la* LeCun



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

Fully connected layer:

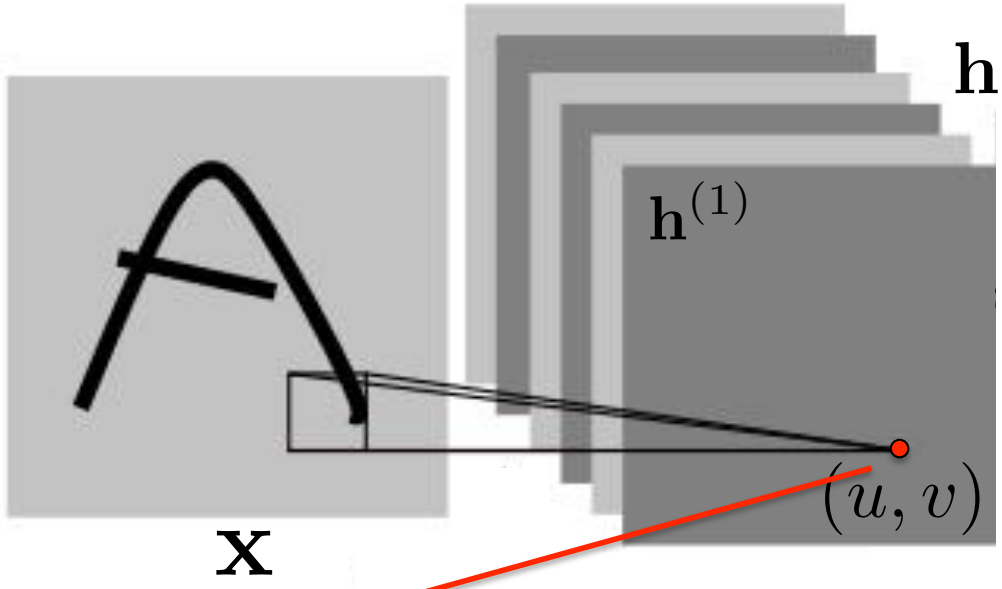
$$\mathbf{h}_6 = g(\mathbf{W}_5\mathbf{h}_5 + \mathbf{b}_5)$$

Convolutional layer:

$$\mathbf{h}^{(i)} = g(\mathbf{f}^{(i)} * \mathbf{x})$$

More efficient for images

Convolutional Layer

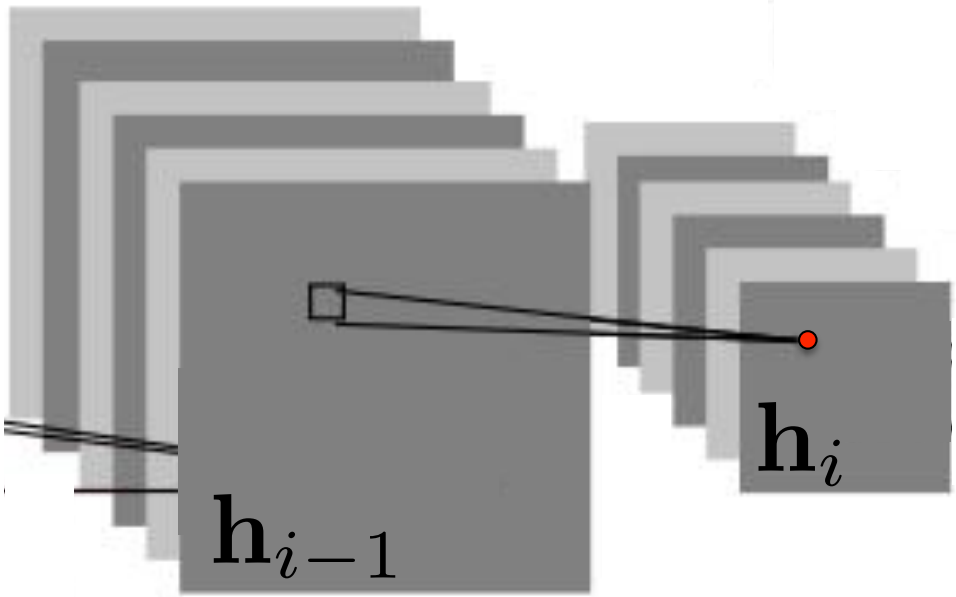


$$\mathbf{h}^{(i)} = g(\mathbf{f}^{(i)} * \mathbf{x})$$

$$\mathbf{h}^{(i)}[u, v] = g\left(\sum_{du=-m}^{+m} \sum_{dv=-n}^{+n} \mathbf{f}^{(i)}[du, dv]\mathbf{x}[u + du, v + dv]\right)$$

$$g(a) = \tanh(a)$$

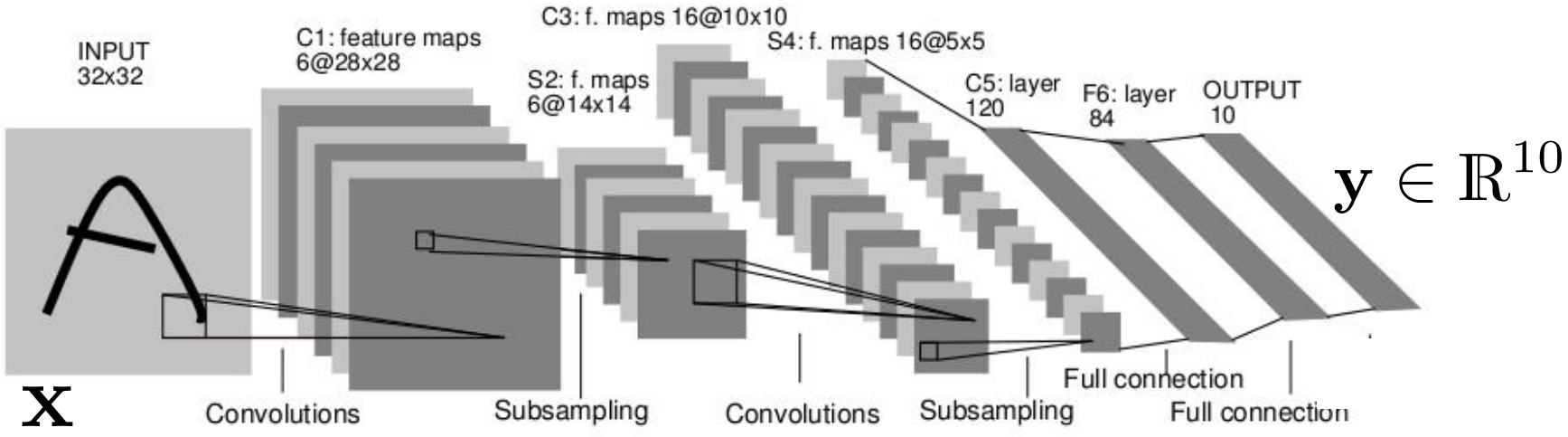
Pooling Layer



For example, max-pooling:

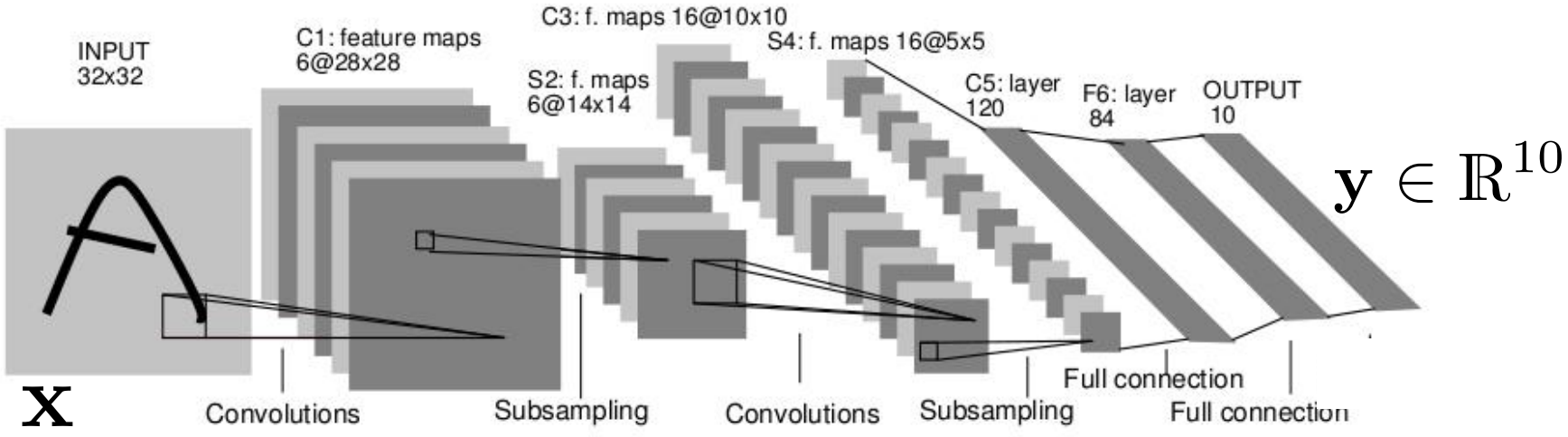
$$\mathbf{h}_i[u,v] = \max\{ \begin{array}{ll} \mathbf{h}_{i-1}[2u, & 2v], \\ \mathbf{h}_{i-1}[2u, & 2v+1], \\ \mathbf{h}_{i-1}[2u+1, & 2v], \\ \mathbf{h}_{i-1}[2u+1, & 2v+1] \end{array} \}$$

A Complete Deep Network



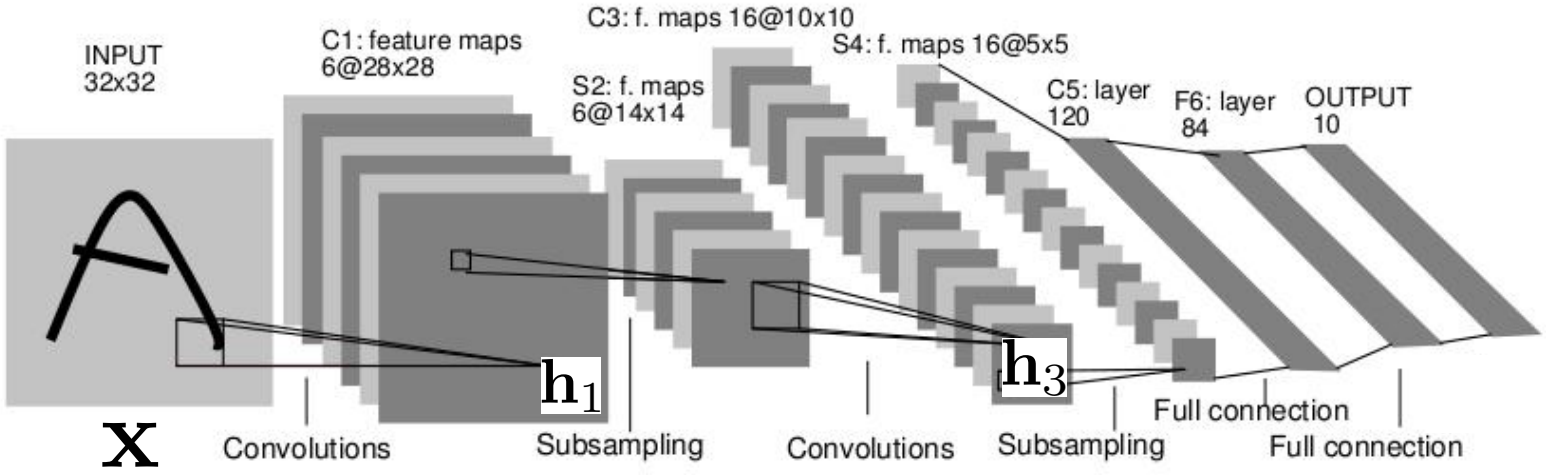
$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{1}_{,\mathbf{n}} * \mathbf{x})]$$
$$\mathbf{h}_2 = [\text{pool}(\mathbf{h}_{1,1}), \ldots, \text{pool}(\mathbf{h}_{1,n})]$$
$$\mathbf{h}_3 = [g(\mathbf{f}_{3,1} * \mathbf{h}_{2,1}), \ldots, g(\mathbf{f}_{3,n} * \mathbf{h}_{2,n})]$$
$$\mathbf{h}_4 = [\text{pool}(\mathbf{h}_{3,1}), \ldots, \text{pool}(\mathbf{h}_{3,n})]$$
$$\mathbf{h}'_4 = \text{Vec}(\mathbf{h}4)$$
$$\mathbf{h}_5 = g(\mathbf{W}_5\mathbf{h}'_4 + \mathbf{b}_5)$$
$$\mathbf{h}_6 = g(\mathbf{W}_6\mathbf{h}_5 + \mathbf{b}_6)$$
$$\mathbf{y} = \mathbf{W}_7\mathbf{h}_6 + \mathbf{b}_7$$

A Complete Deep Network

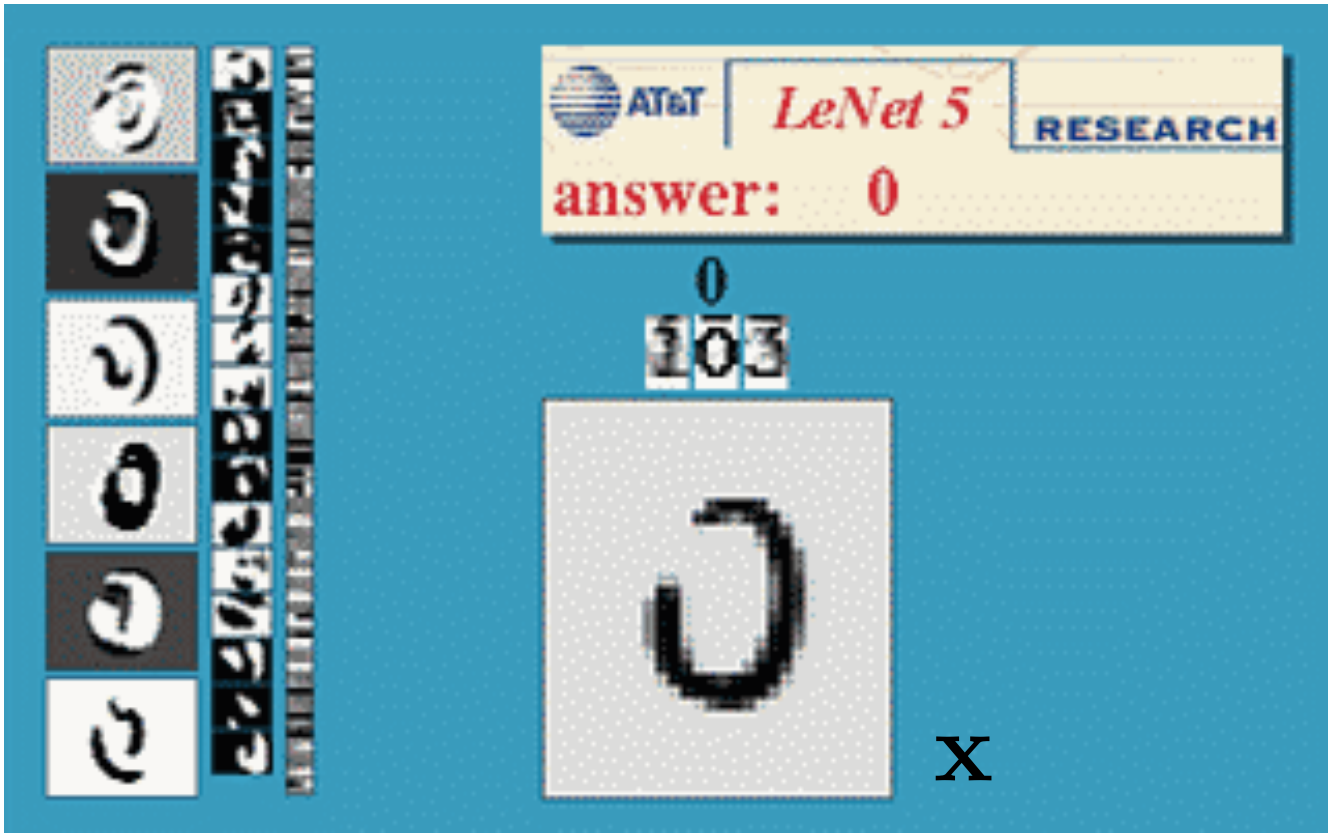


$$\mathbf{h}_1 = [g(\mathbf{f}_{1,1} * \mathbf{x}), \ldots, g(\mathbf{1}_{,\mathbf{n}} * \mathbf{x})]$$
$$\mathbf{h}_2 = [\text{pool}(\mathbf{h}_{1,1}), \ldots, \text{pool}(\mathbf{h}_{1,n})]$$
$$\mathbf{h}_3 = [g(\mathbf{f}_{3,1} * \mathbf{h}_{2,1}), \ldots, g(\mathbf{f}_{3,n} * \mathbf{h}_{2,n})]$$
$$\mathbf{h}_4 = [\text{pool}(\mathbf{h}_{3,1}), \ldots, \text{pool}(\mathbf{h}_{3,n})]$$
$$\mathbf{h}'_4 = \text{Vec}(\mathbf{h}4)$$
$$\mathbf{h}_5 = g(\mathbf{W}_5\mathbf{h}'_4 + \mathbf{b}_5)$$
$$\mathbf{h}_6 = g(\mathbf{W}_6\mathbf{h}_5 + \mathbf{b}_6)$$
$$\mathbf{y} = \mathbf{W}_7\mathbf{h}_6 + \mathbf{b}_7$$

$$p(\mathbf{x} \text{ contains digit } \#i) = \frac{\exp(\mathbf{y}[i])}{\sum_j \exp(\mathbf{y}[j])}$$


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
h1
h3
x
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection

h_1 h_3


AT&T
LeNet 5
RESEARCH
answer: 0
0
103
x

Loss Function

Do NOT a least-squares loss function for a classification problem!

Can be (for example):

$$\mathcal{L}(\mathcal{T}) = -\sum_{(\mathbf{x},d)\in\mathcal{T}} \log\left(\frac{\exp(\mathbf{y}(\mathbf{x})_d)}{\sum_j \exp(\mathbf{y}(\mathbf{x})_i)}\right)$$

Training sample

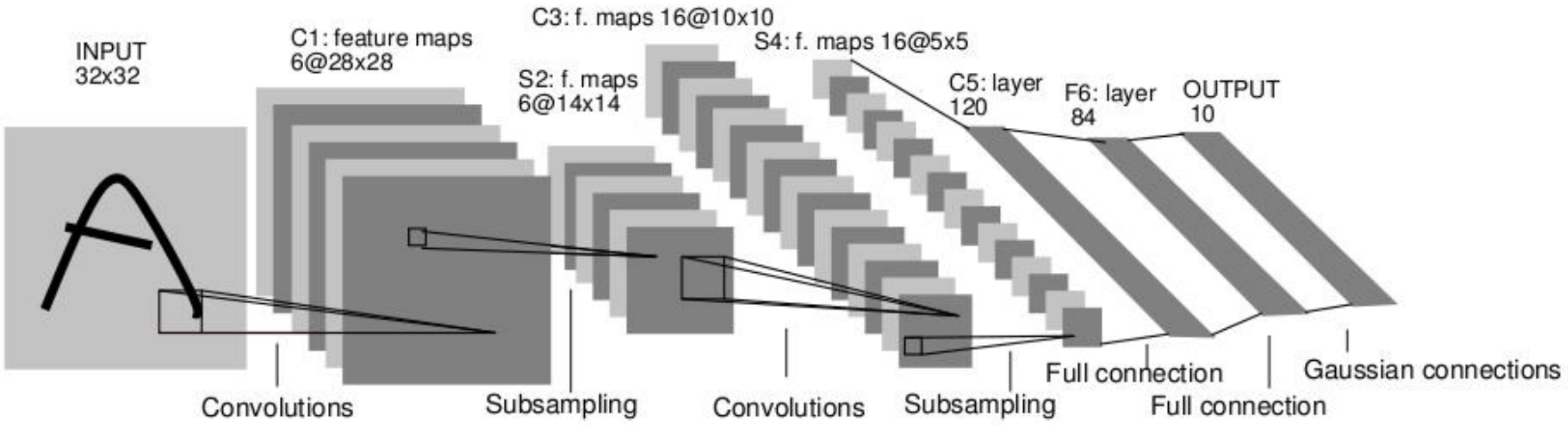
Desired output for the training sample

Predicted output for the training sample

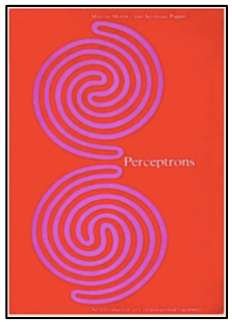
$\mathcal{T}$

Optimization:
still gradient descent

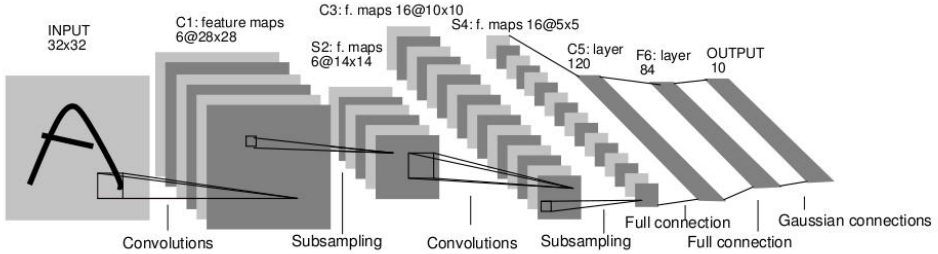
How to Choose the Number of Layers? The Number of Filters per Layer?



History of [Deep] Learning

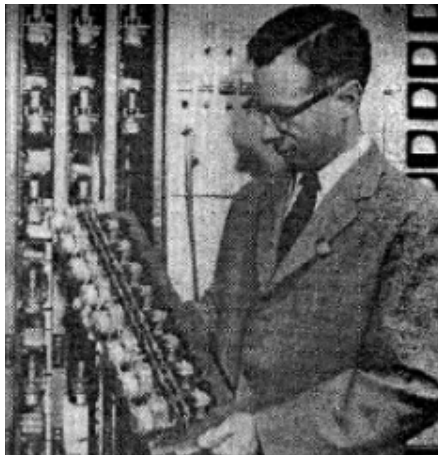


1969: *Perceptrons* book, Minsky and Papert

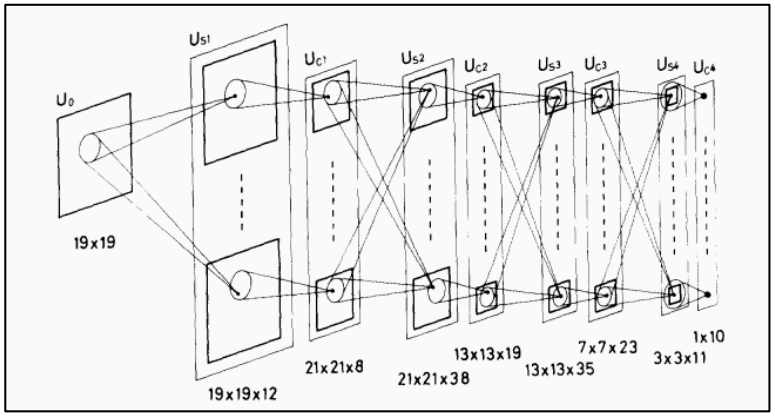


1992: LeNet, LeCun

1958, Cornell: Perceptron



1987: Neocognitron, Fukushima



circa 2006: Unsupervised Pretraining

Unsupervised Pretraining

Greedy layer-wise training of deep networks:

- Deep Belief Networks;
- Stacked Denoising Autoencoders;
- Energy-Based Models.

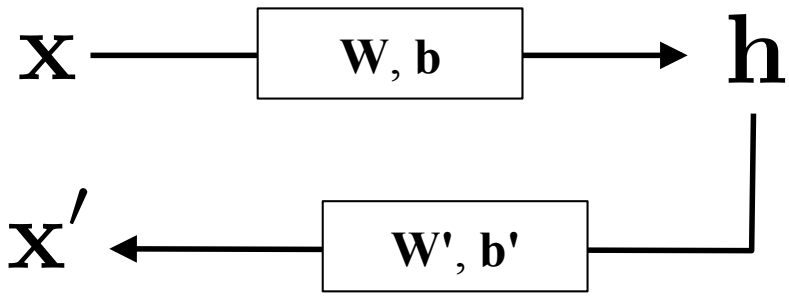
Example: Simple autoencoder

$$\min_{\mathbf{W},\mathbf{b},\mathbf{W}',\mathbf{b}'} \|\mathbf{x} - \mathbf{x}'\|^2$$

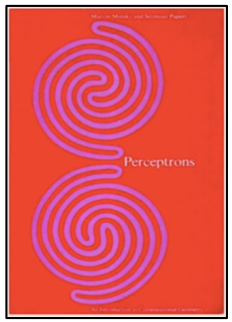
with $\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$

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

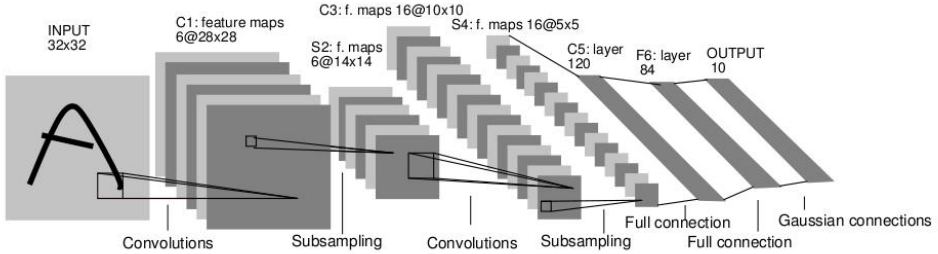
$\dim(\mathbf{h}) \neq \dim(\mathbf{x})$



History of [Deep] Learning

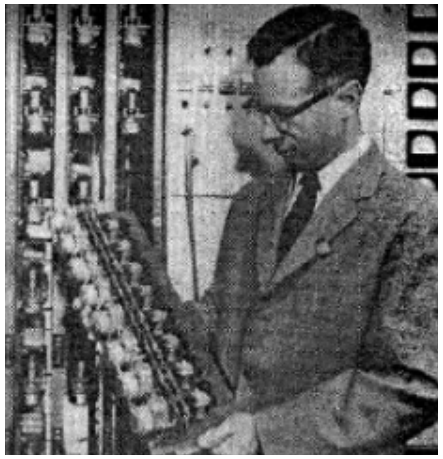


1969: *Perceptrons* book, Minsky and Papert

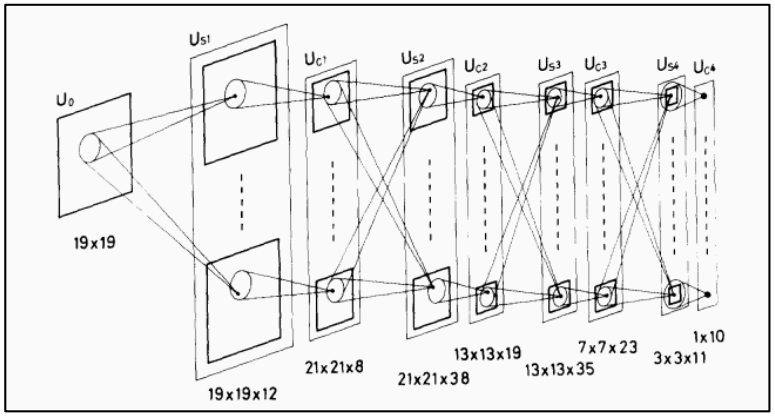


1992: LeNet, LeCun

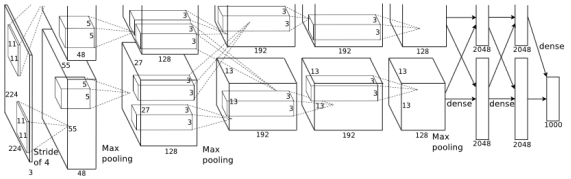
1958, Cornell: Perceptron



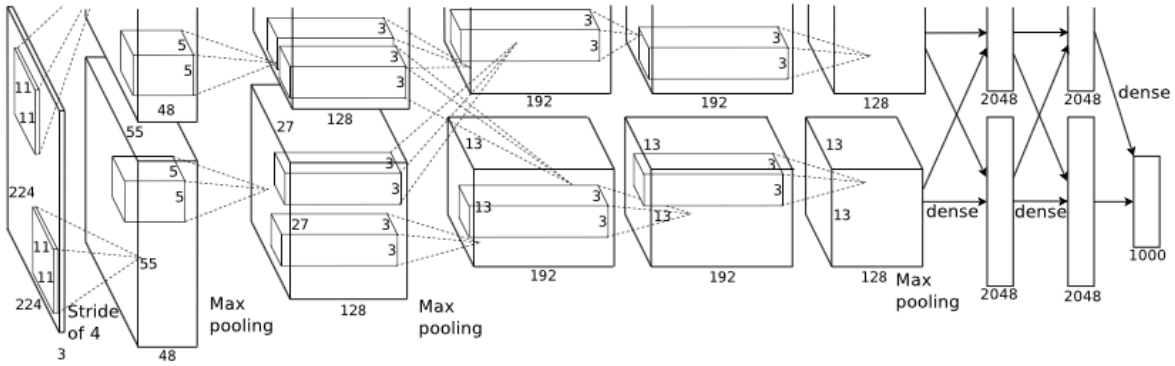
1987: Neocognitron, Fukushima



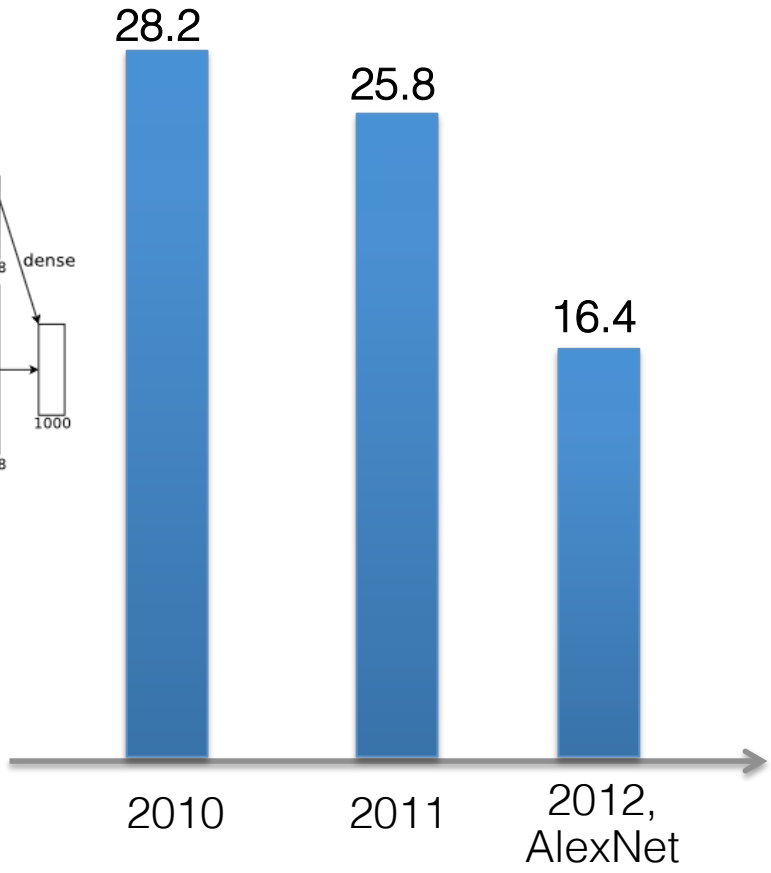
2010: AlexNet



AlexNet (2010)

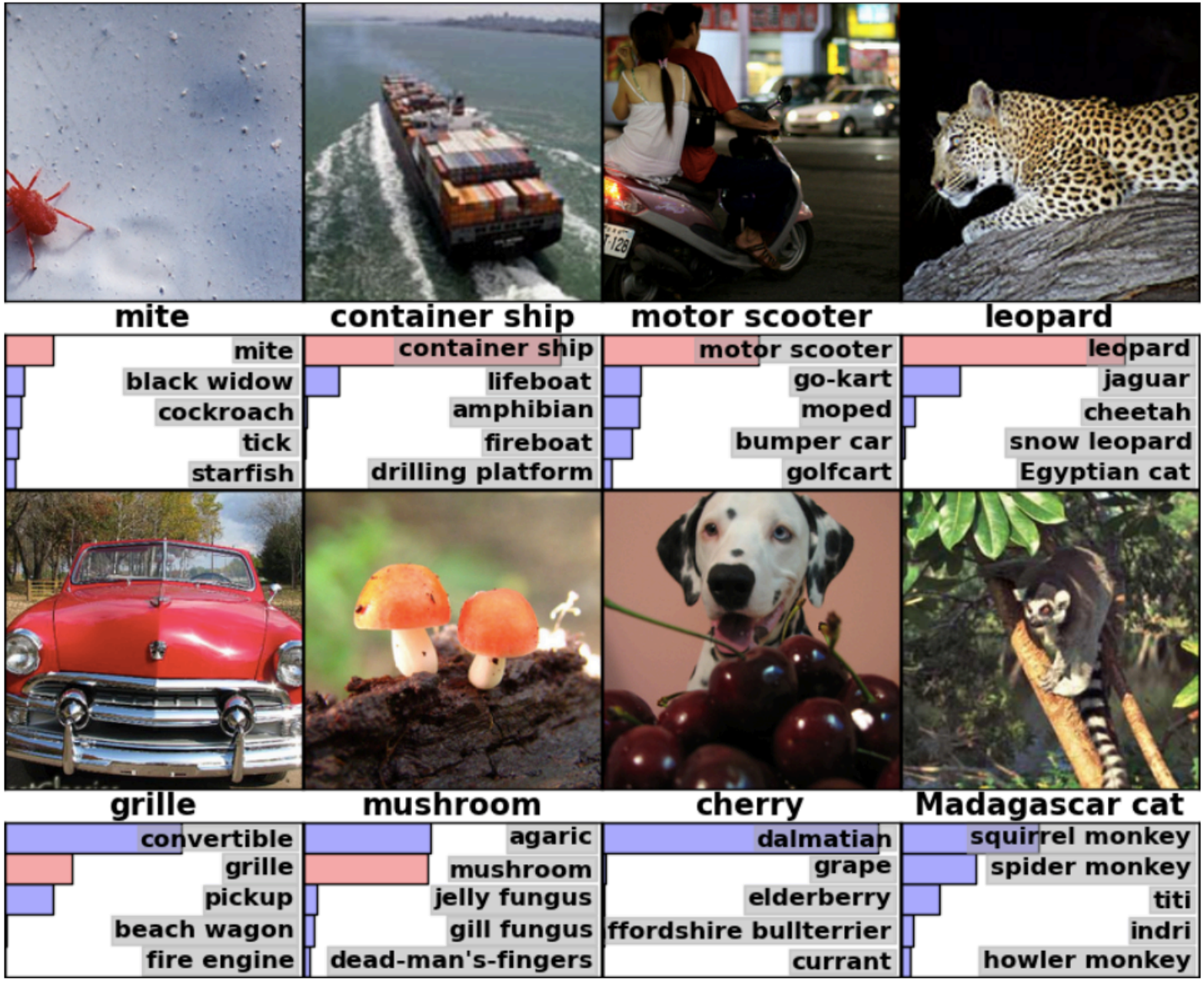


Training time: 2 weeks on 2 GPUs



Classification task on ImageNet challenge top-5 error.

AlexNet: Some results on ILSVRC-2010

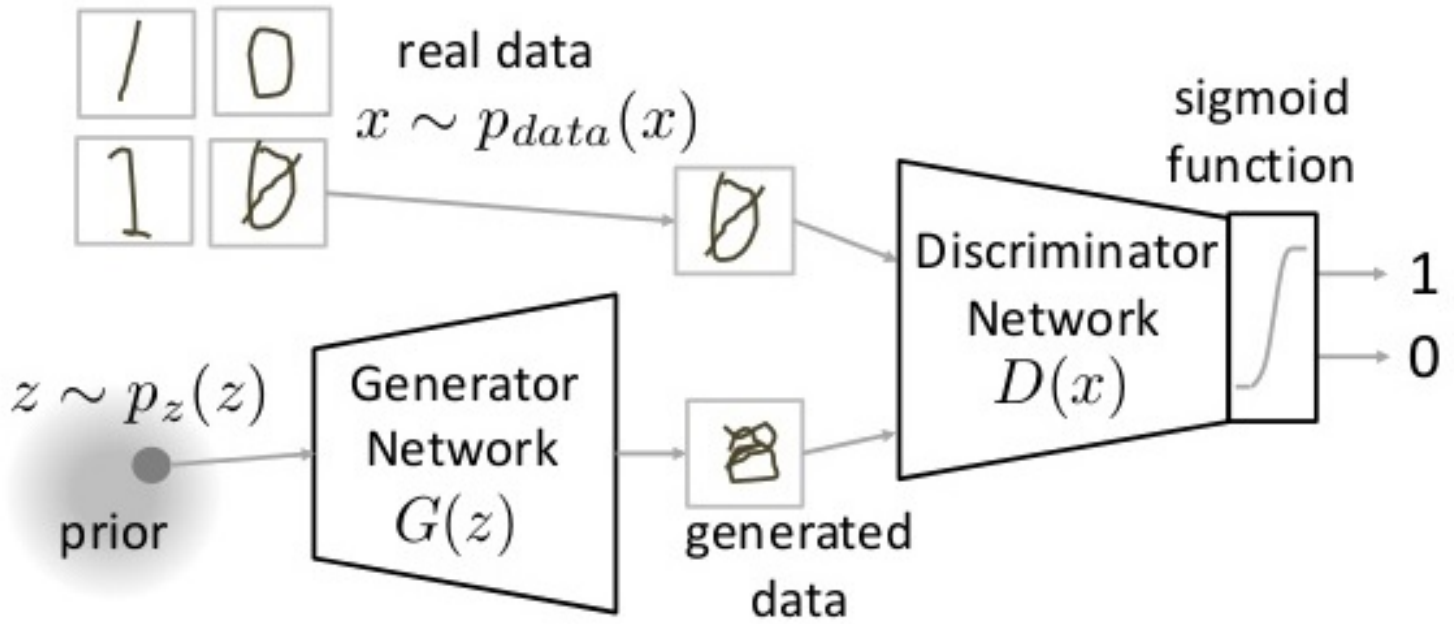


History of Deep Learning

2013: Deep Q-Learning

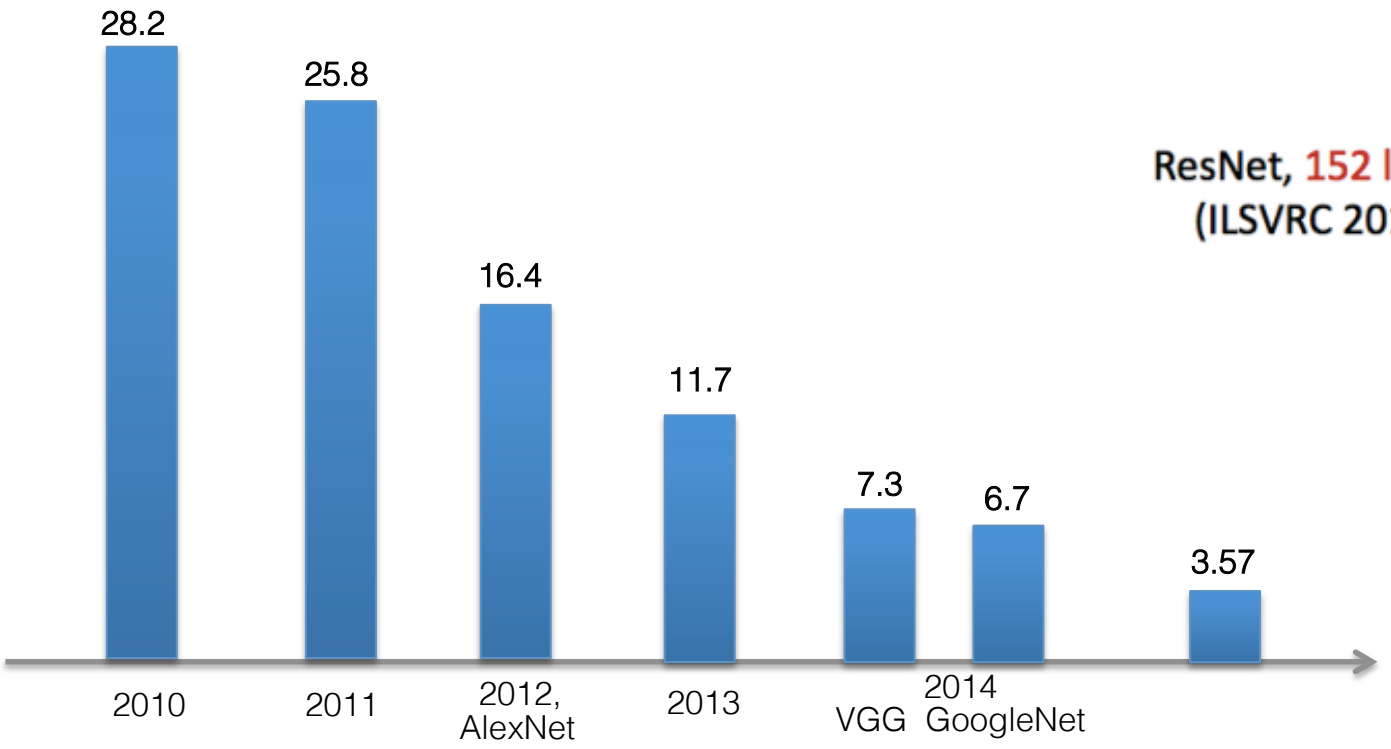
History of Deep Learning

2014: Generative Adversarial Networks



History of Deep Learning

2015: ResNet



ResNet, 152 layers
(ILSVRC 2015)

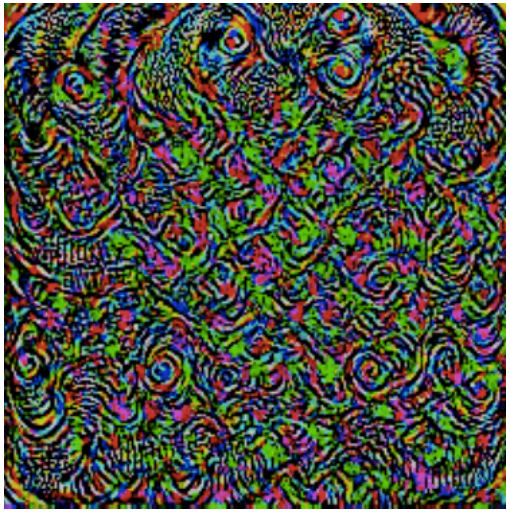
Classification task on ImageNet challenge top-5 error.

Why Now?

- Faster computers (with GPUs);
- More training data;
- Better optimization algorithms;
- Easy to use and powerful libraries in Python;
- Before AlexNet, people were not convinced that Deep Learning worked.

DeepFool

$$\Delta(\boldsymbol{x}; \hat{k}) := \min_{\boldsymbol{r}} \|\boldsymbol{r}\|_2 \text{ subject to } \hat{k}(\boldsymbol{x} + \boldsymbol{r}) \neq \hat{k}(\boldsymbol{x}),$$



'Universal' perturbation $\boldsymbol{r}$ for GoogLeNet

'Indian elephant'

'Macaw'