

Introduction to Deep Learning

Vincent Lepetit

Deep Learning



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



[L'Express, 2017]

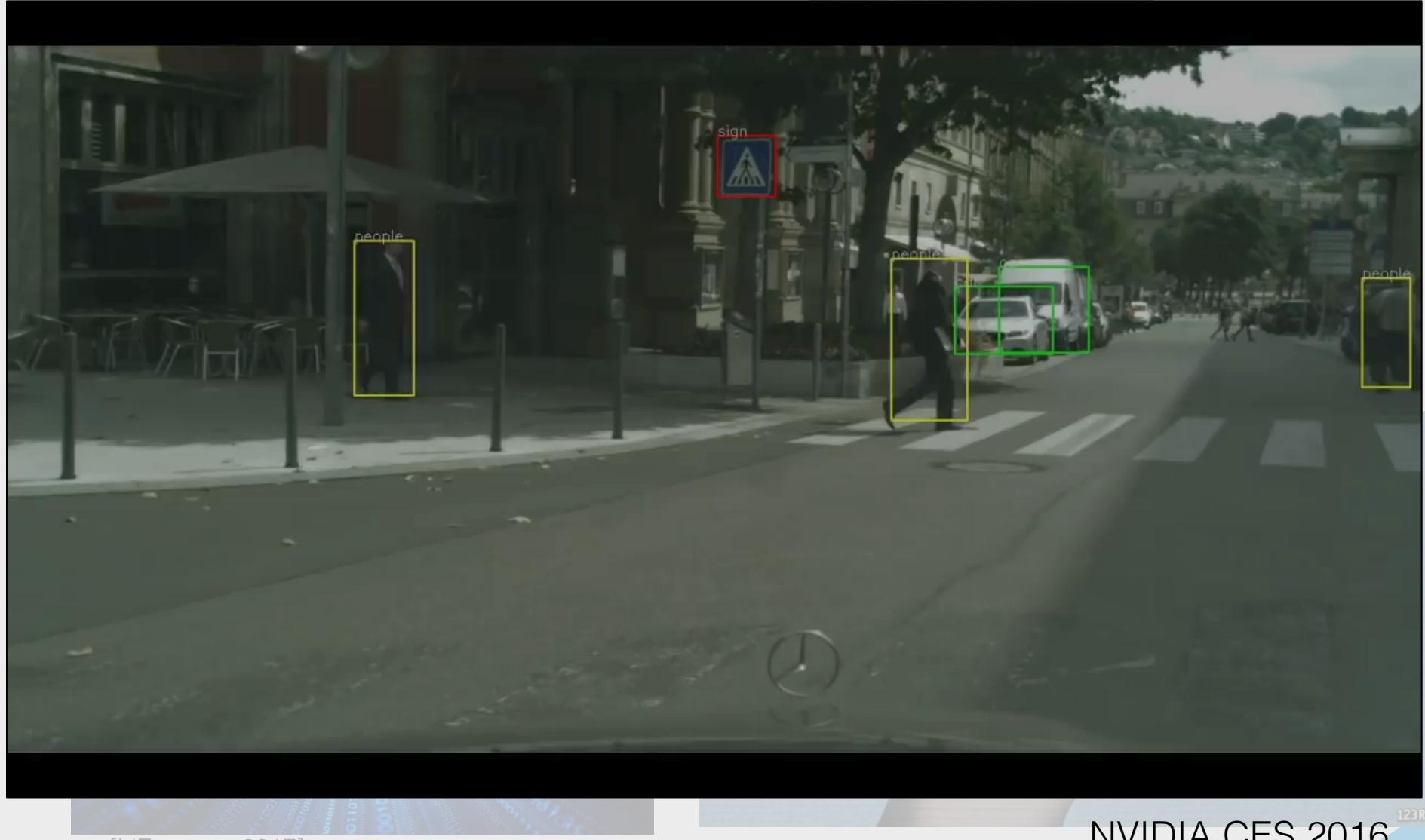


[MIT review technology, 2017]



[New Economist, 2015]

Deep Learning



[L'Express, 2017]

[New Economist, 2015]

NVIDIA CES 2016

Deep Learning

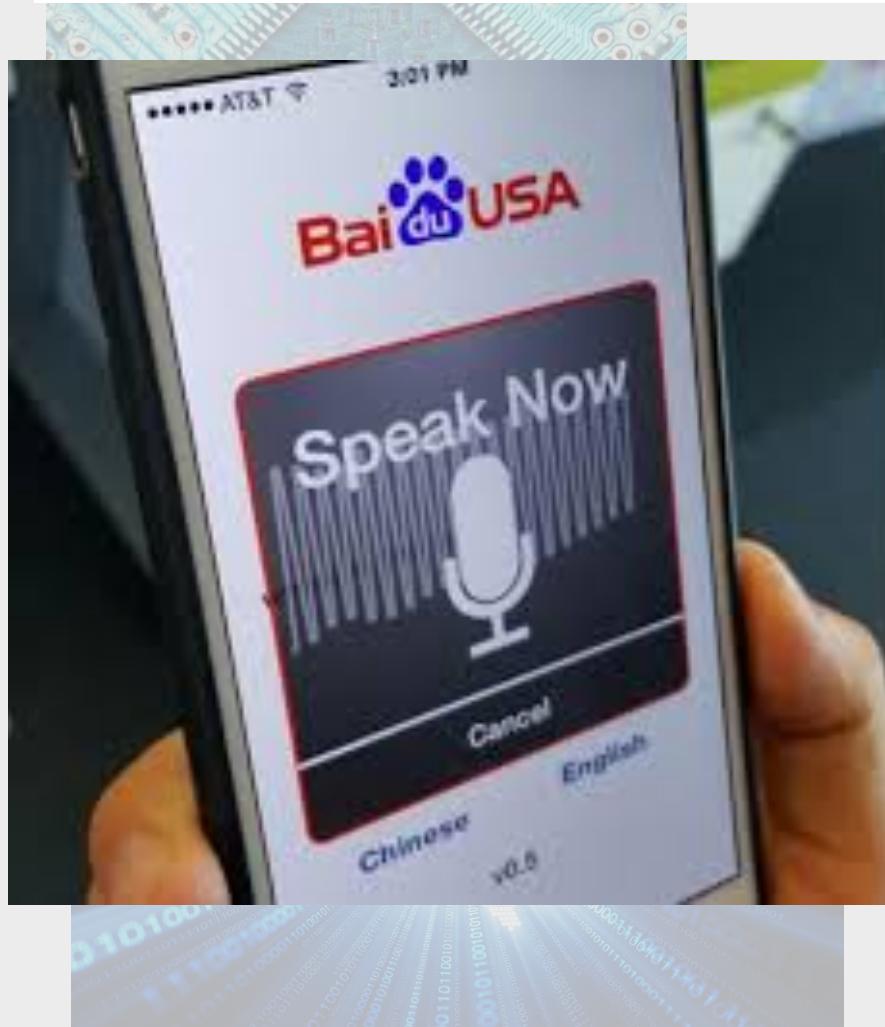


10.3 fps



[L'Express, 2017]

Deep Speech 2



[L'Express, 2017]

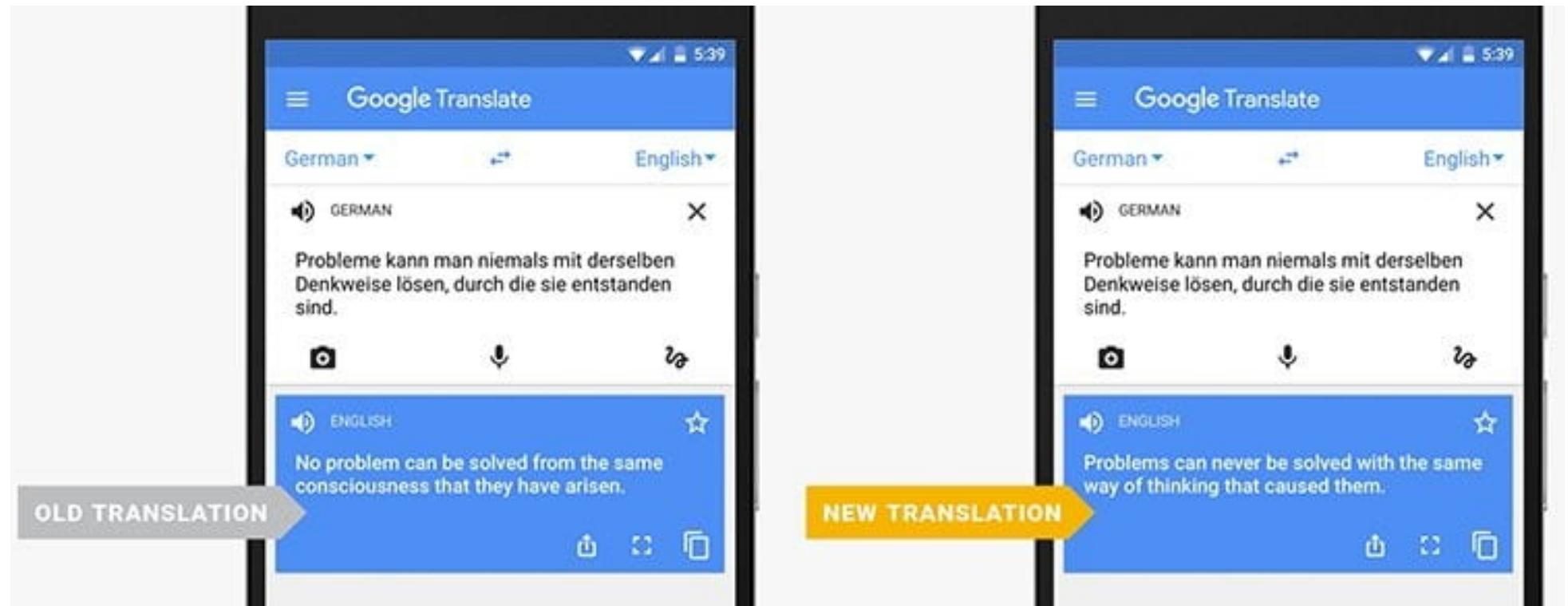


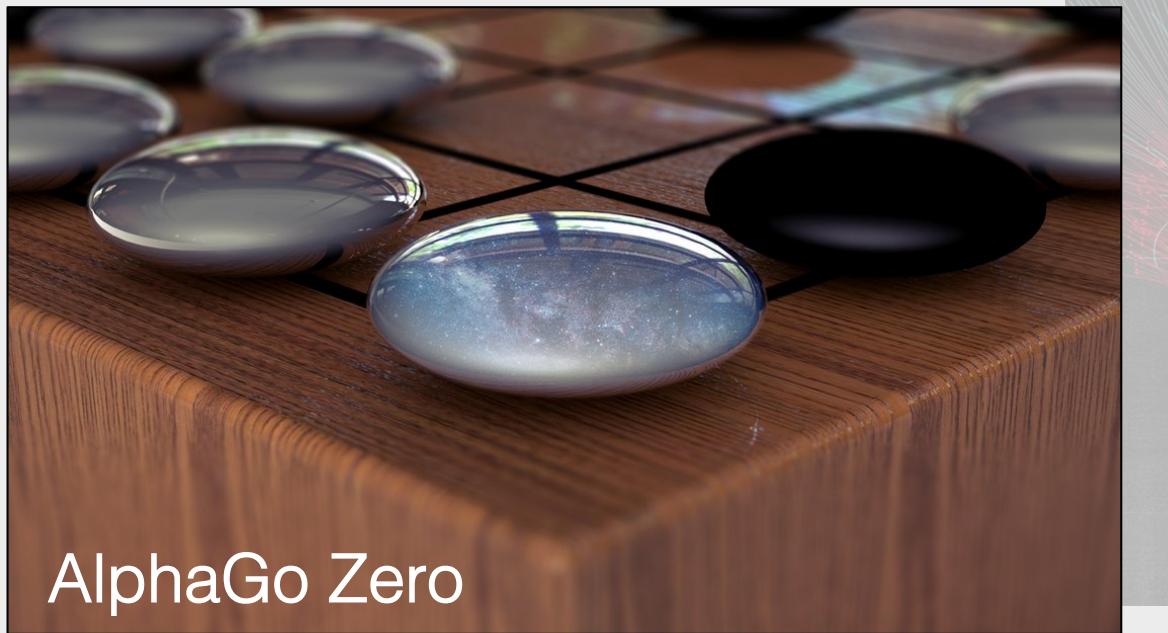
Deep Speech 2: End-to-End Speech Recognition in English and Mandarin



[New Economist, 2015]

Machine Translation

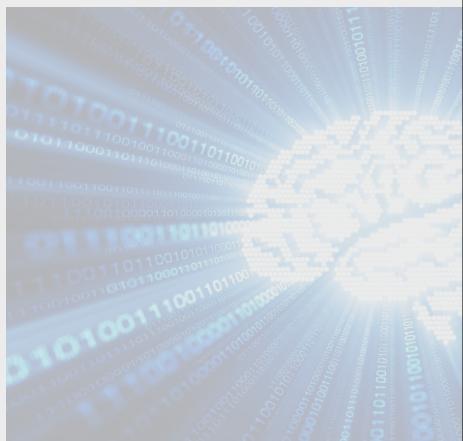




AlphaGo Zero

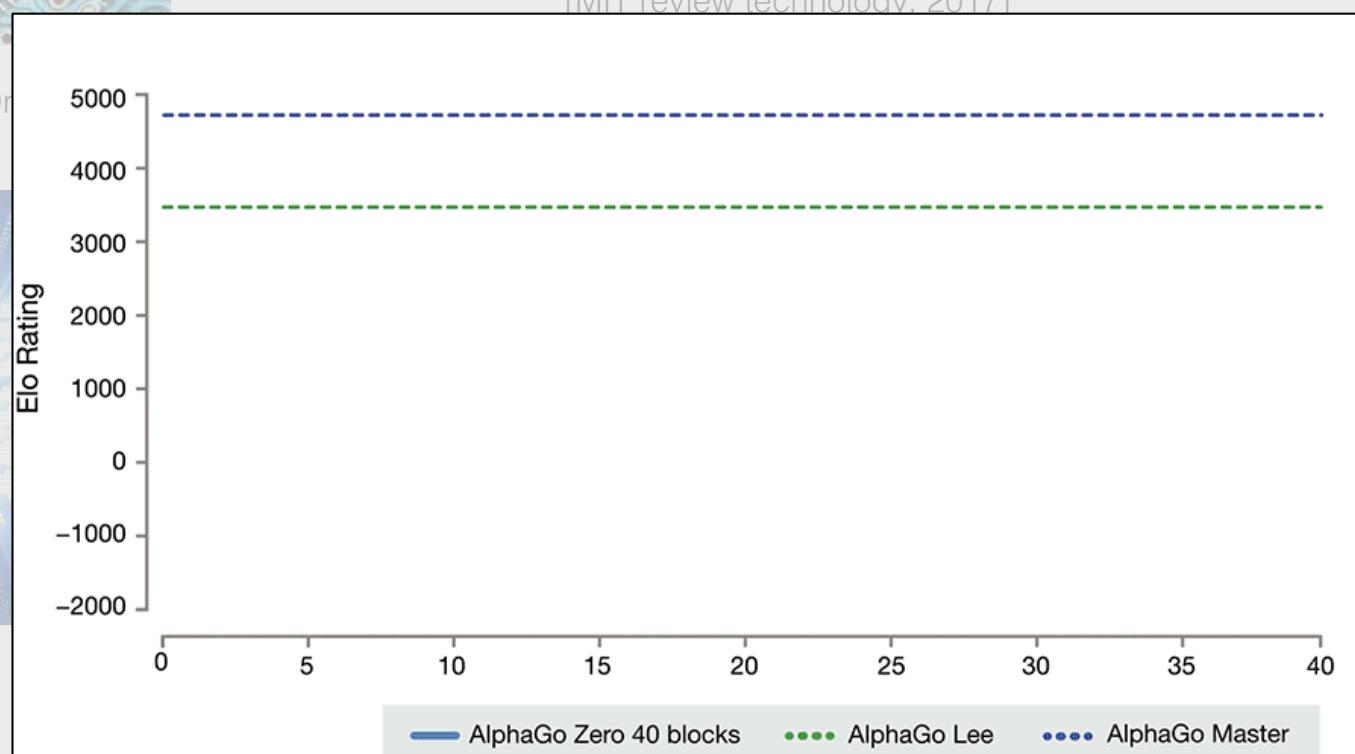


[Vierra / Shutterstock / Deep Dr
Generator / Le Monde 2017]

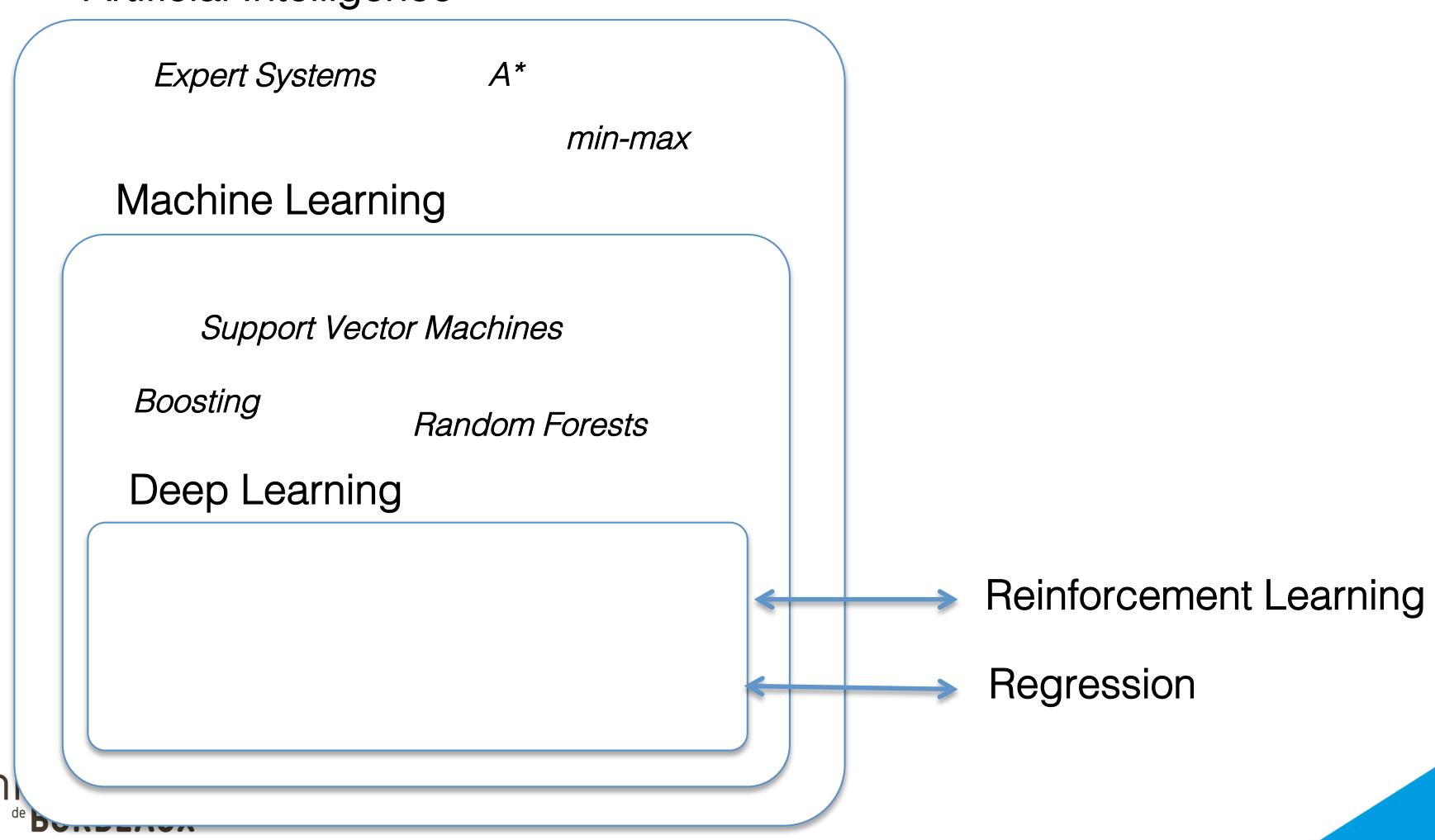


[L'Express, 2017]

[MIT review technology, 2017]



Artificial Intelligence/Machine Learning/Deep Learning



Schedule

Jan 19, 13h50-15h20, 15h30-17h: Introduction to Deep Learning and to Keras

Jan 26, 13h50-15h20: lecture, Optimization
15h30-17h00: TP, mini-project 1

Feb 2, 13h50-15h20: lecture, Specific architectures
15h30-17h00: TP, mini-project 1

Feb 9, 13h50-15h20: lecture, Natural Language Processing
15h30-17h00: TP, mini-project 2

Mar 2, 13h50-15h20: lecture, Natural Language Processing
15h30-17h00: TP, mini-project 2

Mar 9, 13h50-15h20: lecture, Reinforcement Learning
15h30-17h00: TP, mini-project 3

Mar 16, 13h50-15h20: lecture, Reinforcement Learning, Alpha Go
15h30-17h00: TP, mini-project 3

Schedule

Jan 19, 13h50-15h20, 15h30-17h: Introduction to Deep Learning and to Keras

Jan 26, 13h50-15h20: lecture, Optimization
15h30-17h00: TP, mini-project 1

Feb 2, 13h50-15h20: lecture, Specific architectures
15h30-17h00: TP, mini-project 1



Feb 9, 13h50-15h20: lecture, Natural Language Processing
15h30-17h00: TP, mini-project 2

Mar 2, 13h50-15h20: lecture, Natural Language Processing
15h30-17h00: TP, mini-project 2

Mar 9, 13h50-15h20: lecture, Reinforcement Learning
15h30-17h00: TP, mini-project 3

Mar 16, 13h50-15h20: lecture, Reinforcement Learning, Alpha Go
15h30-17h00: TP, mini-project 3

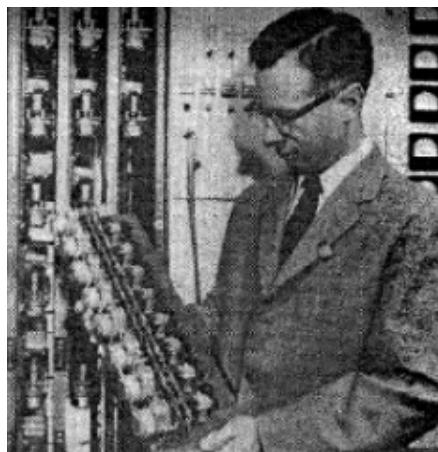
Teachers

- Edouard Oyallon
- Alexis Conneau
- Maria Vakalopoulou
- Vincent Lepetit

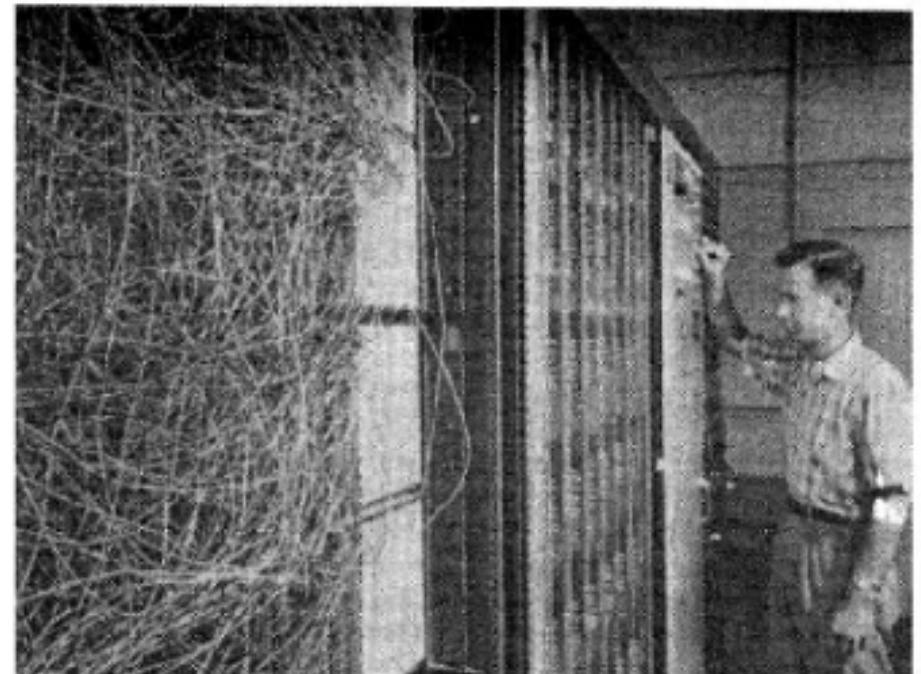
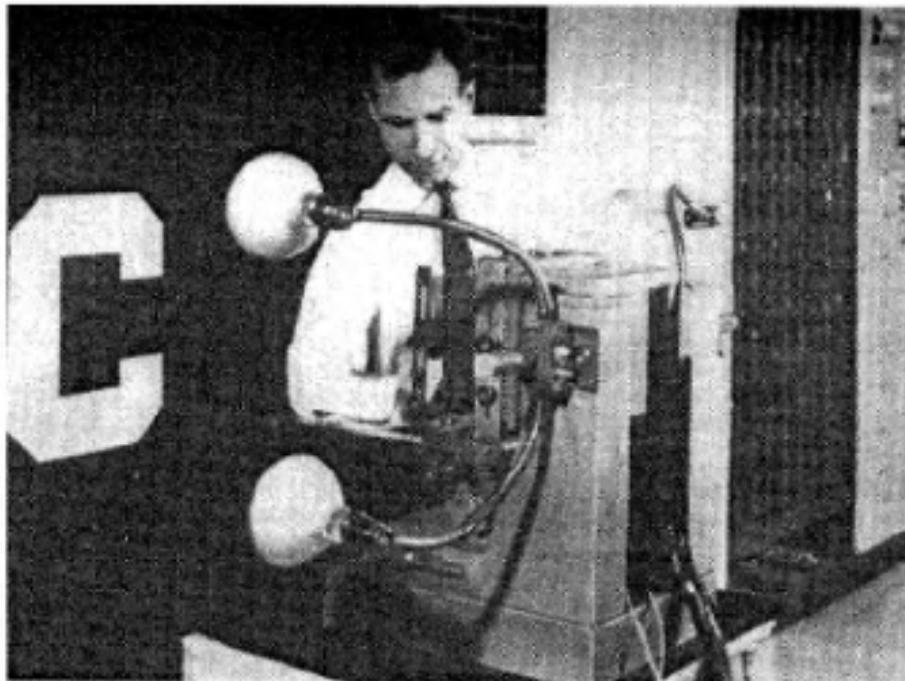
History of [Deep] Learning



1958, Cornell:
Perceptron

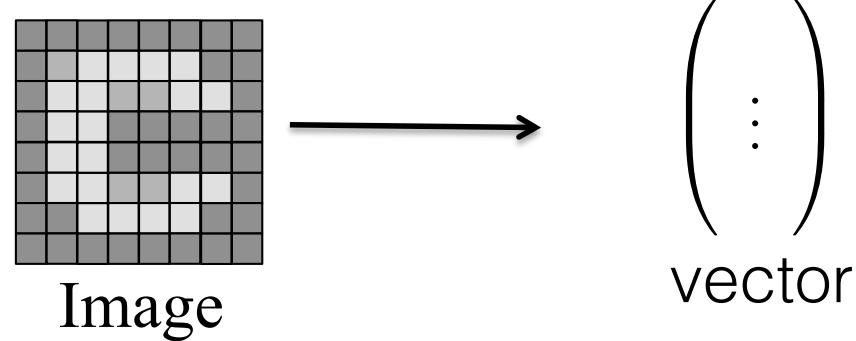


Perceptron (1958, Frank Rosenblatt)



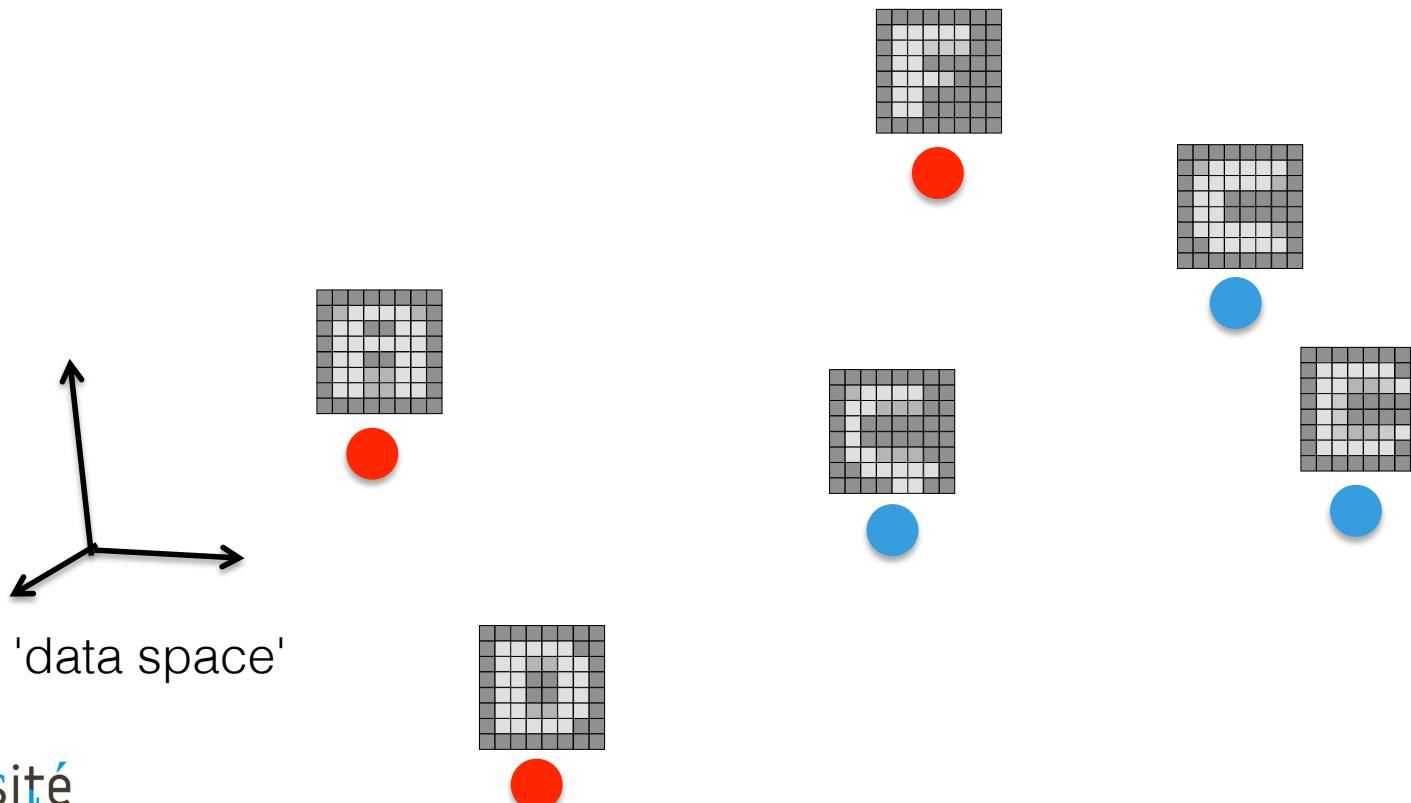
Perceptron: Idea

1. Data are represented as vectors:



Perceptron: Idea

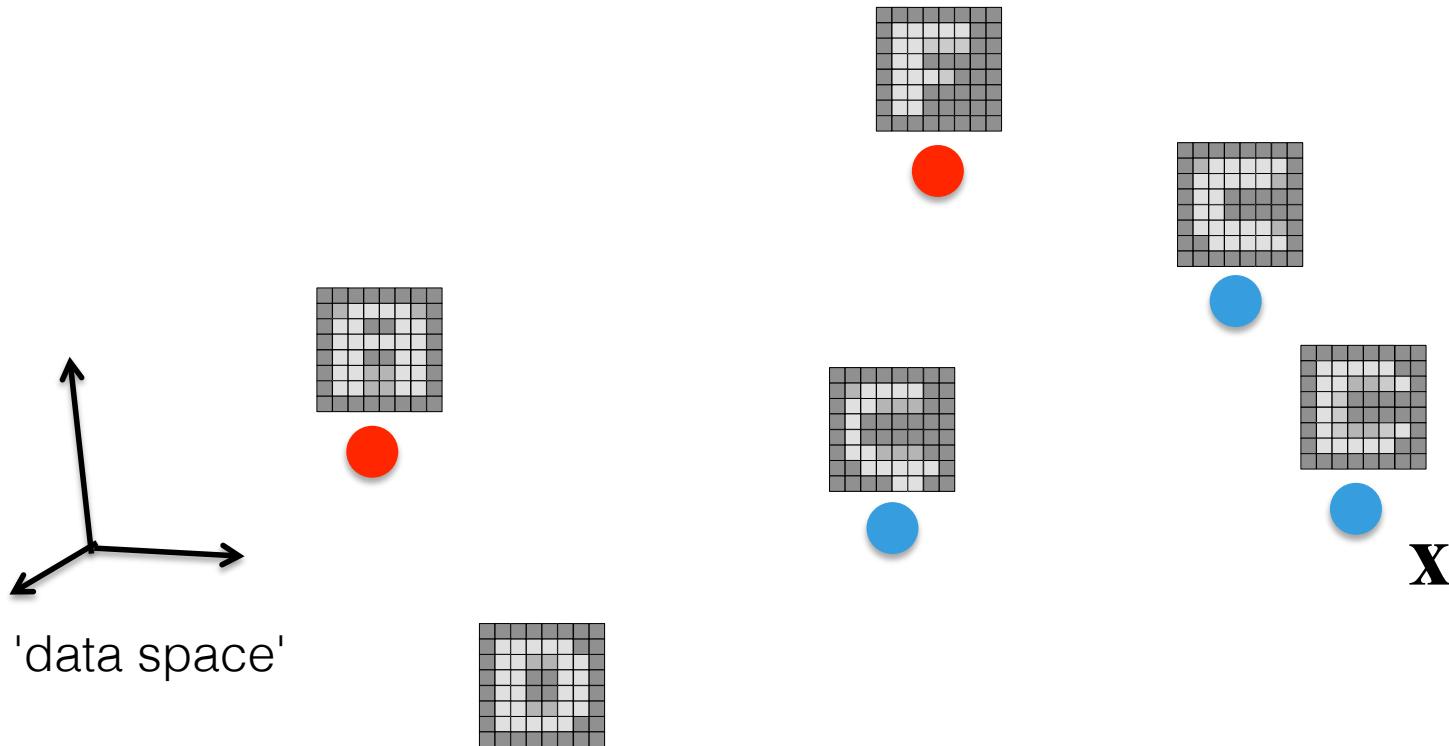
2. Collect training data: some are *positive examples*, some are *negative examples*



Perceptron: Idea

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

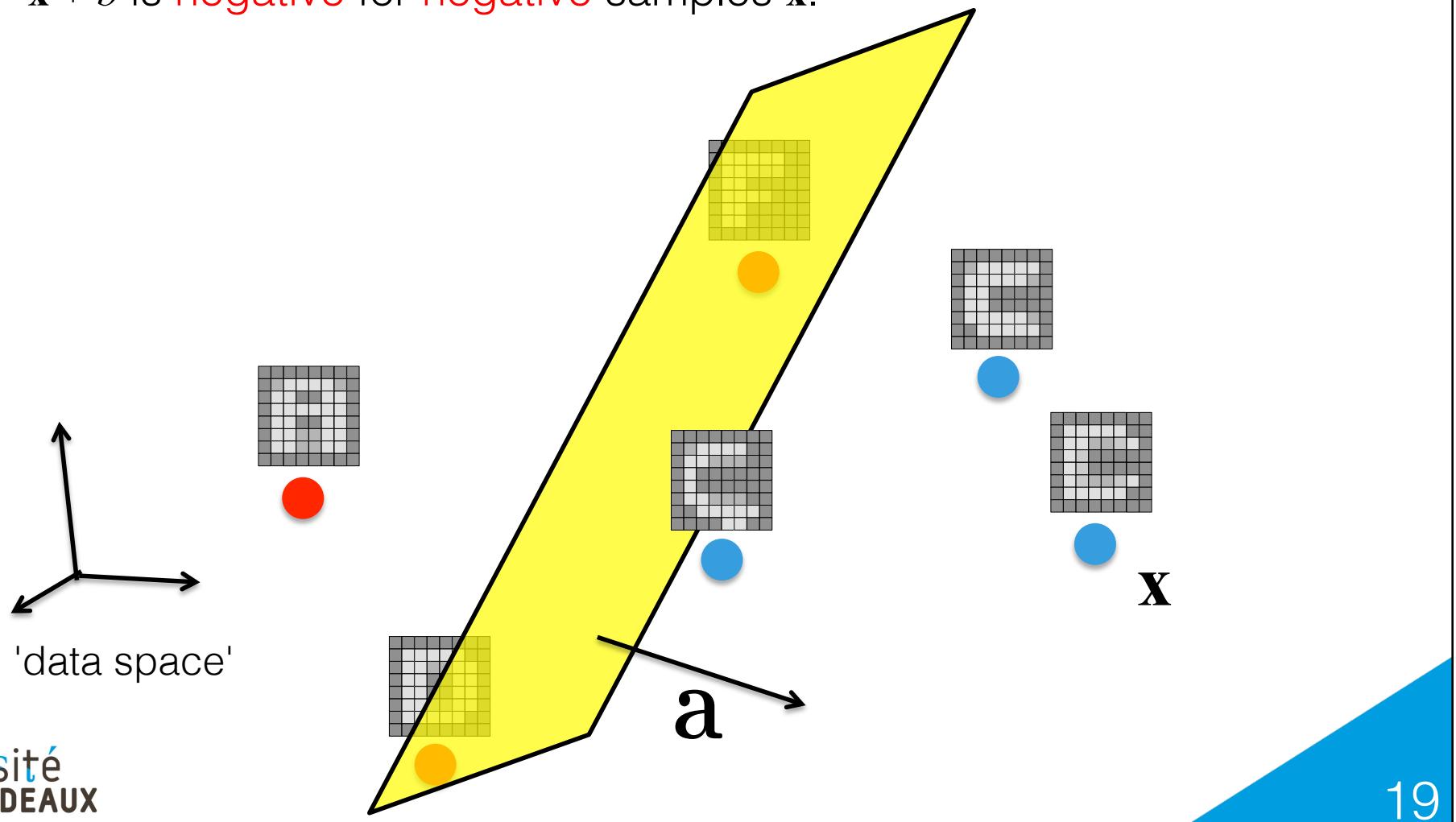
- $\mathbf{a}^\top \mathbf{x} + b$ is **positive** for **positive** samples \mathbf{x} ;
- $\mathbf{a}^\top \mathbf{x} + b$ is **negative** for **negative** samples \mathbf{x} .



Perceptron: Idea

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

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

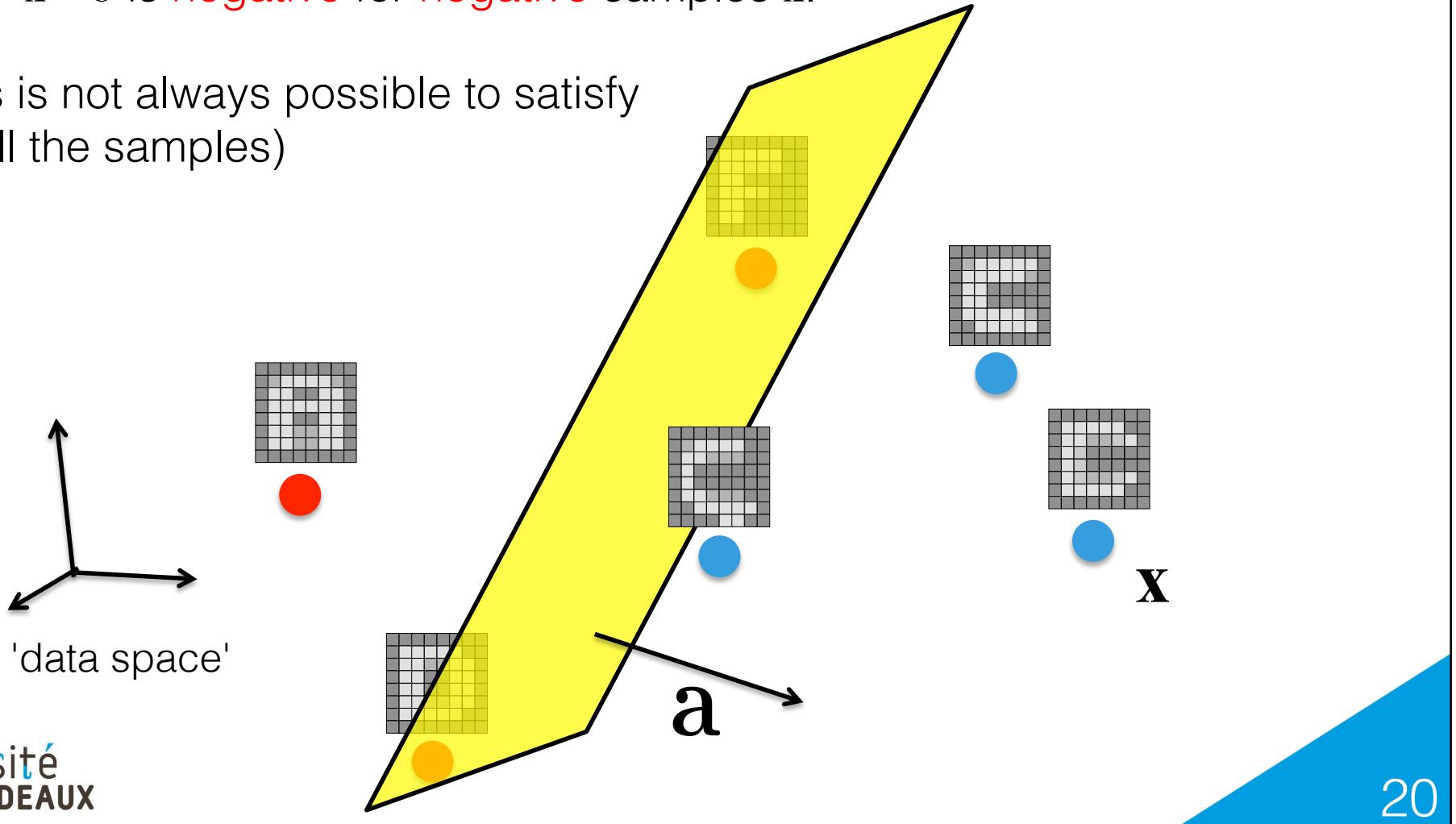


Perceptron: Idea

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

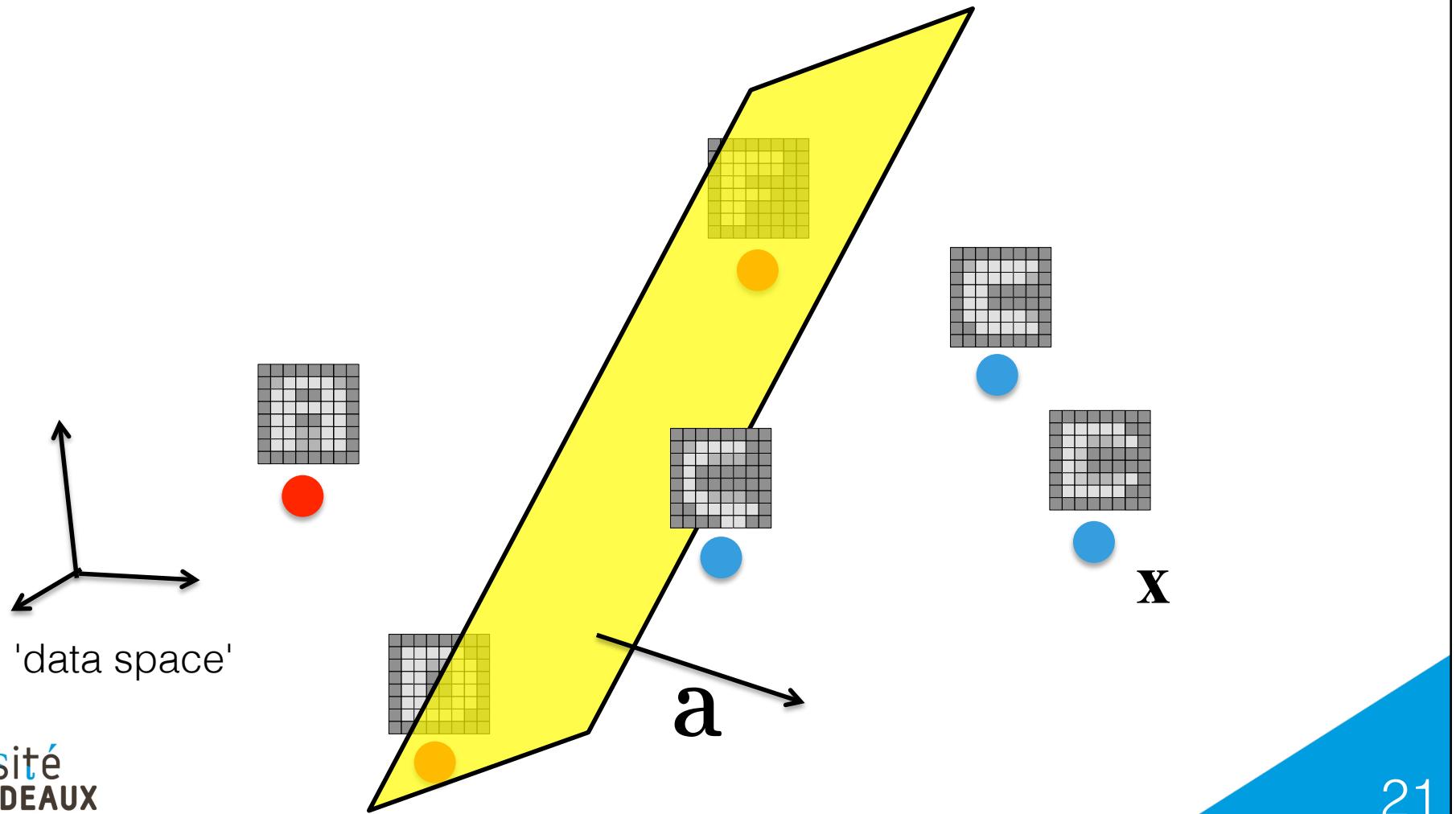
- $\mathbf{a}^\top \mathbf{x} + b$ is **positive** for **positive** samples \mathbf{x} ;
- $\mathbf{a}^\top \mathbf{x} + b$ is **negative** for **negative** samples \mathbf{x} .

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



Perceptron: Idea

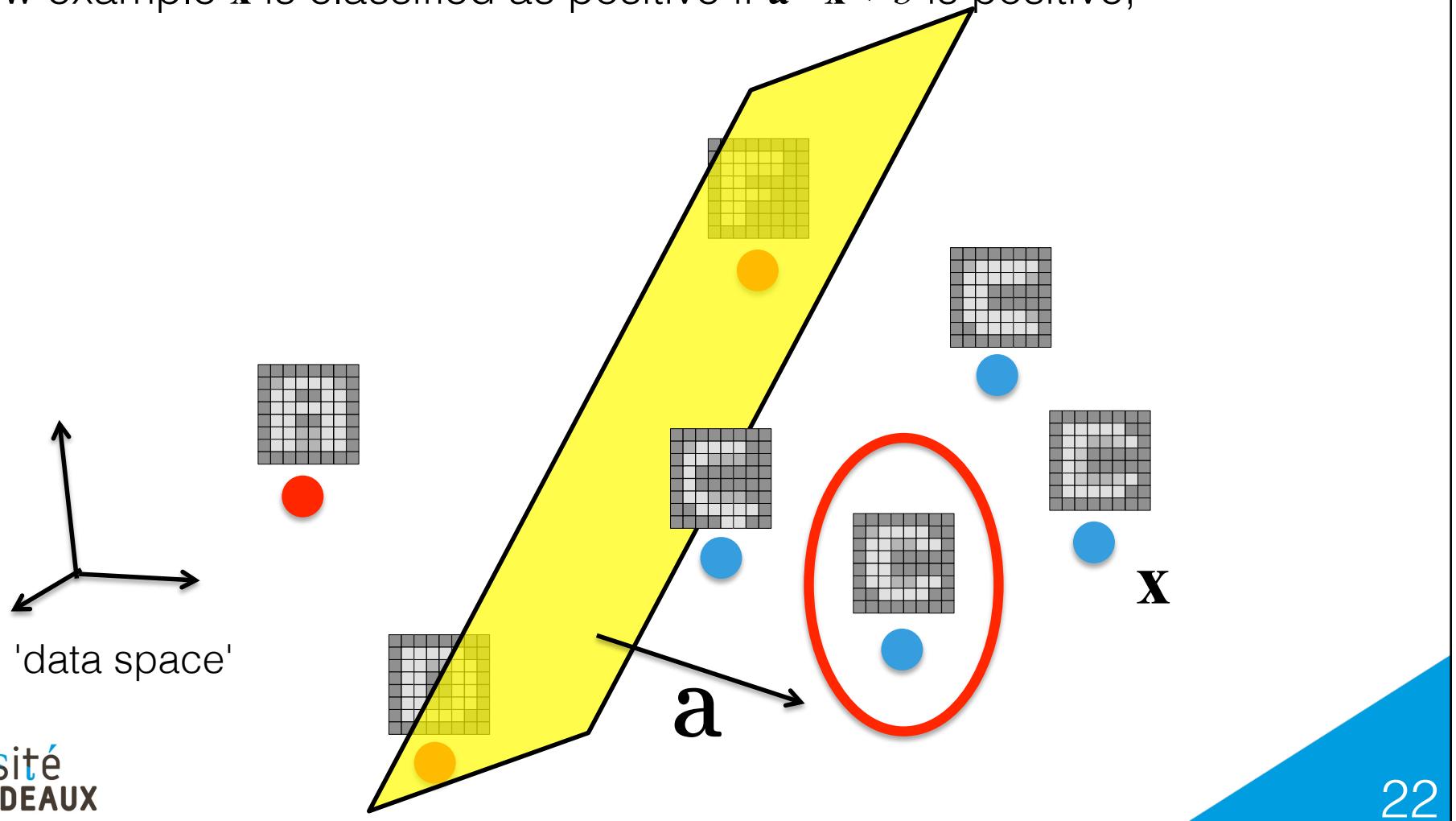
4. Testing: the perceptron can now classify new examples.



Perceptron: Idea

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

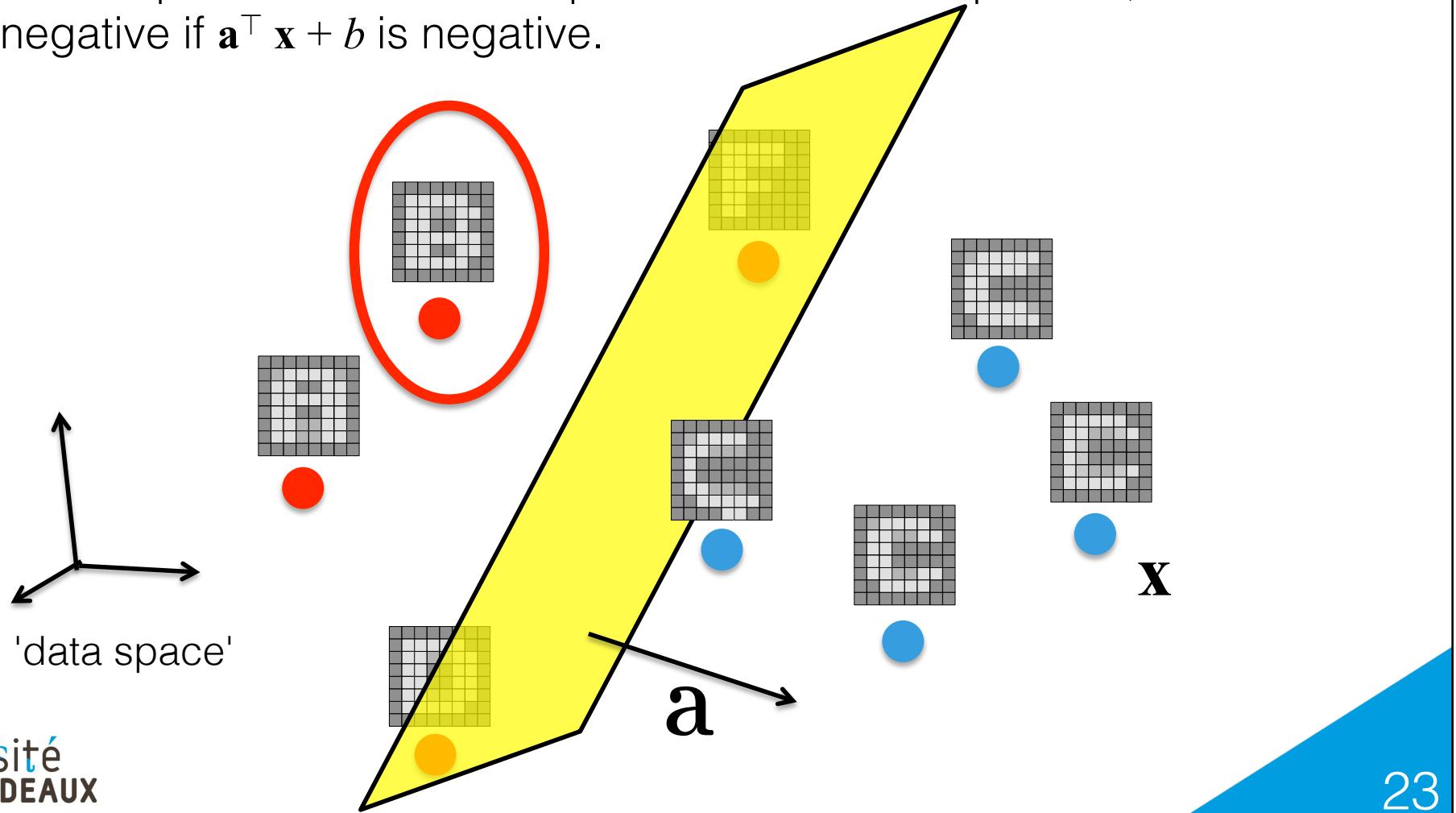
A new example \mathbf{x} is classified as positive if $\mathbf{a}^\top \mathbf{x} + b$ is positive,



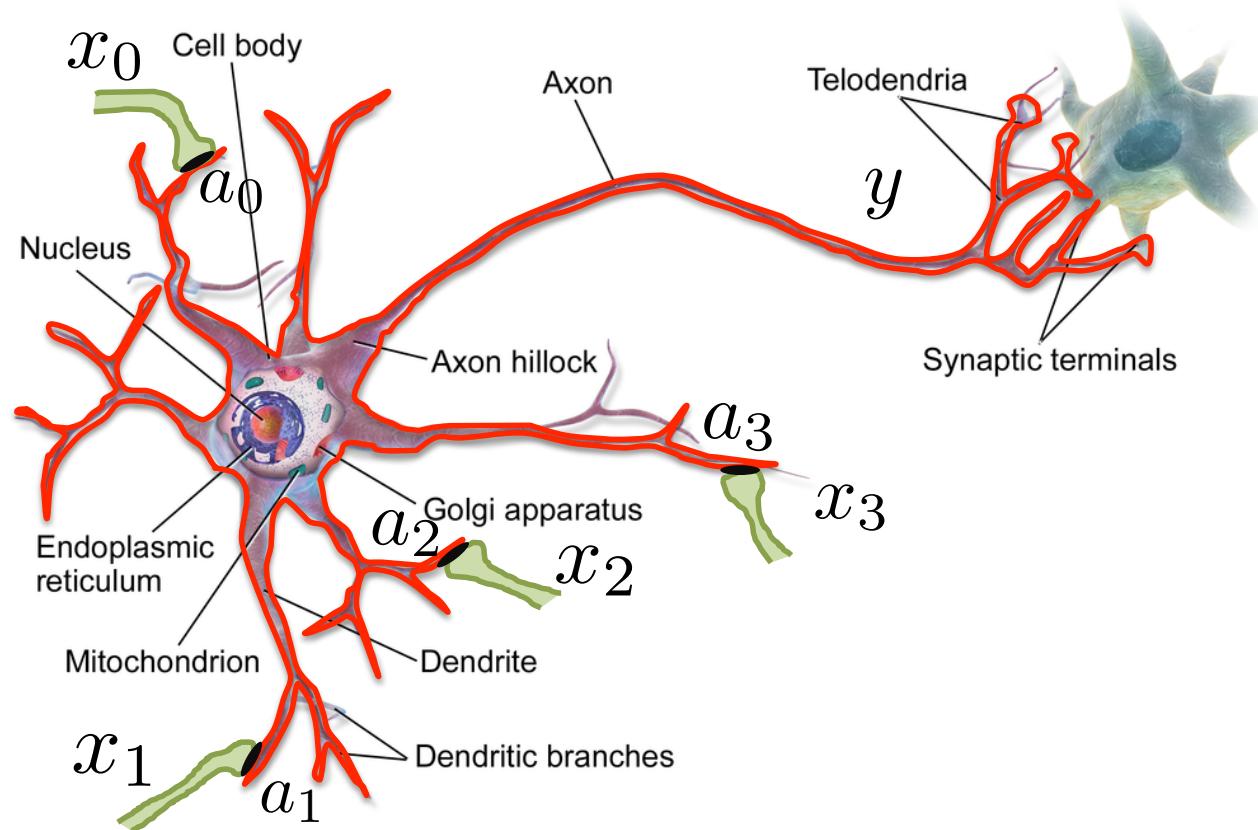
Perceptron: Idea

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

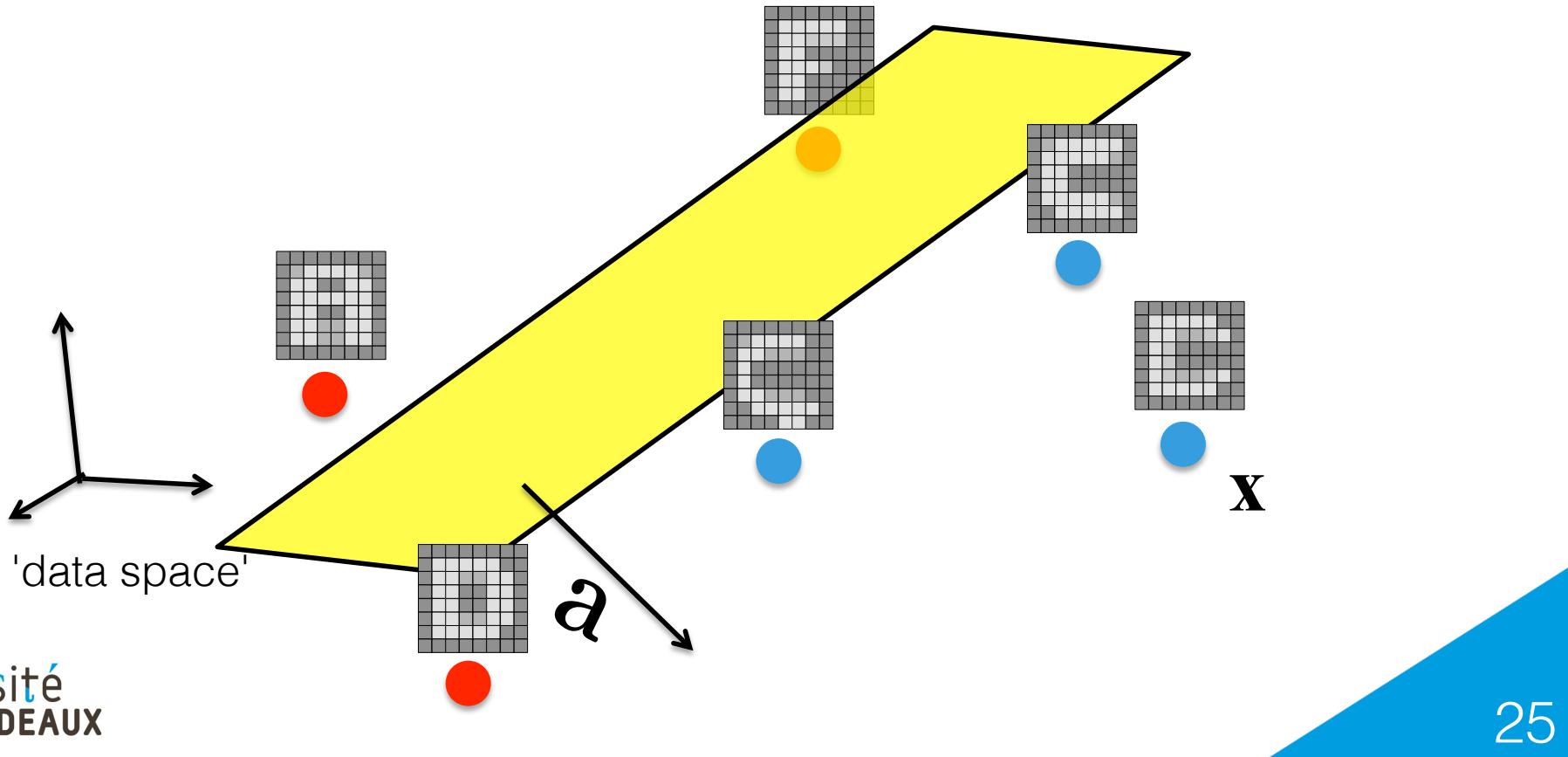


$$\begin{aligned}y &= \text{sign}(a_0x_0 + a_1x_1 + a_2x_2 + a_3x_3) \\&= \text{sign}(\mathbf{a}^\top \mathbf{x})\end{aligned}$$

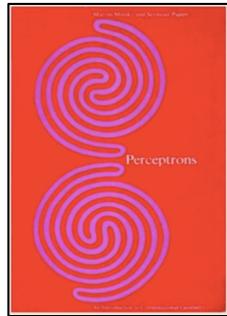
Perceptron: How Can We Find \mathbf{a} and b ?

At the time, *ad hoc* algorithm:

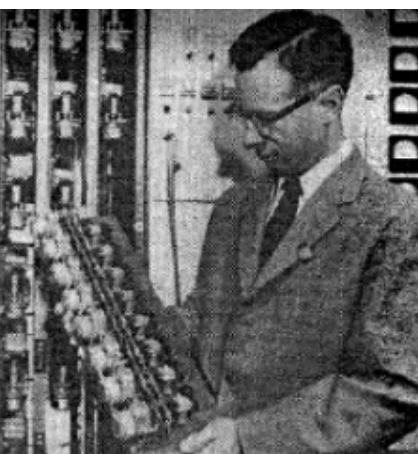
1. Start from a random initialization;
2. For each training sample \mathbf{x} :
 - compare the value of $\mathbf{a}^\top \mathbf{x} + b$ and its expected sign,
 - adapt \mathbf{a} and b to get a better value for $\mathbf{a}^\top \mathbf{x} + b$.



History of [Deep] Learning

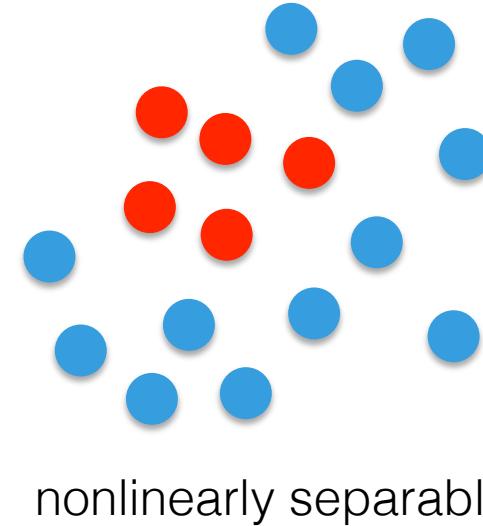
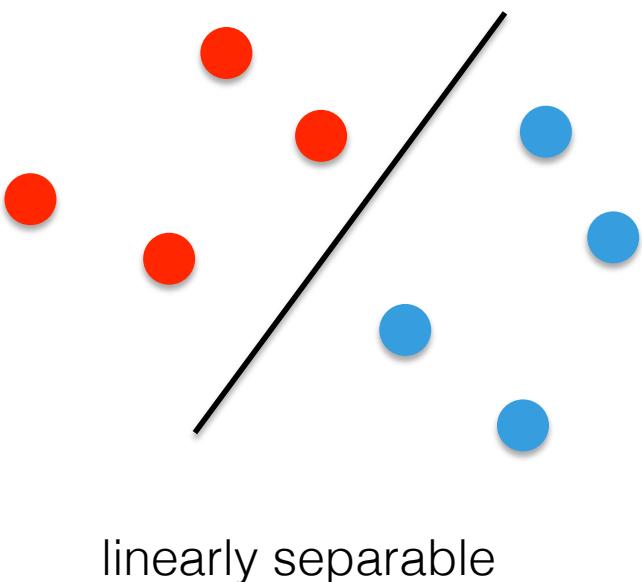


1969: *Perceptrons* book,
Minsky and Papert



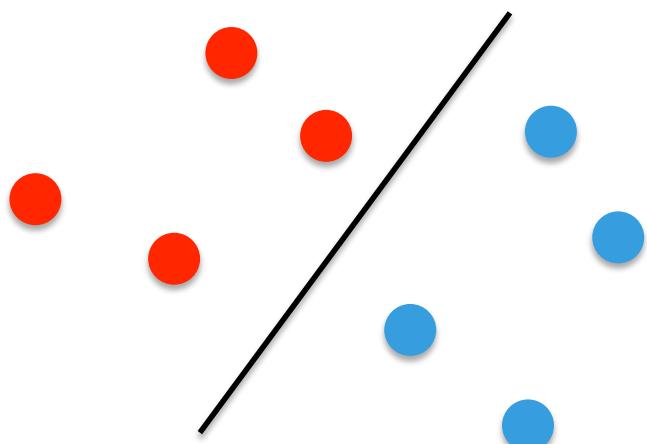
Perceptrons book (1969)

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

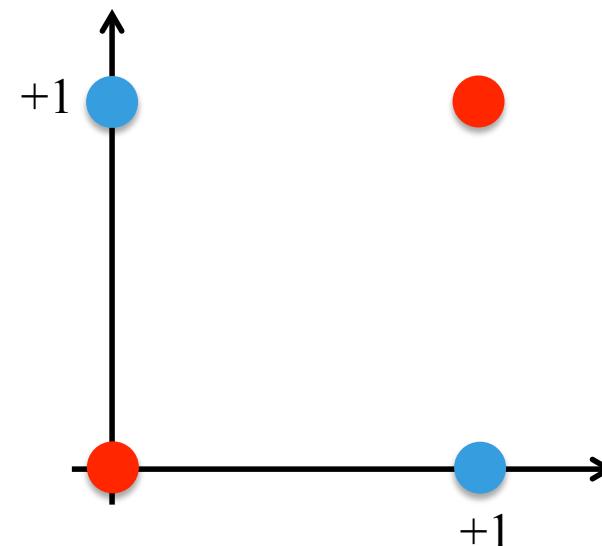


Perceptrons book (1969)

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



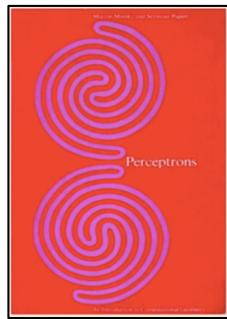
linearly separable



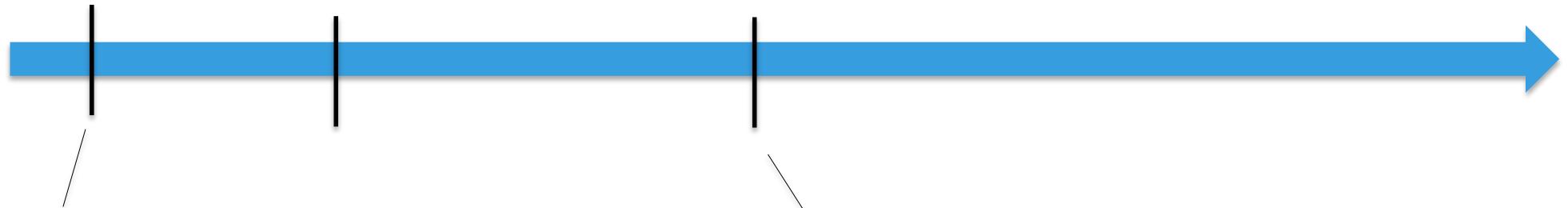
nonlinearly separable:
the case of the x-or function

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

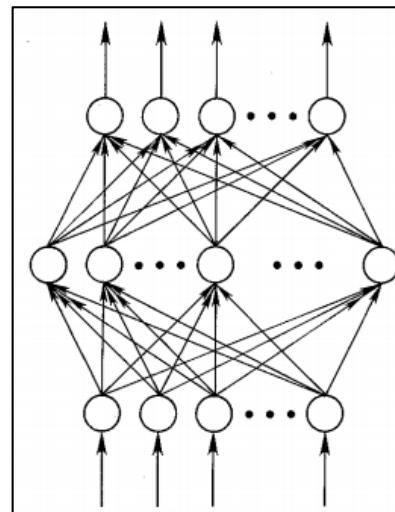
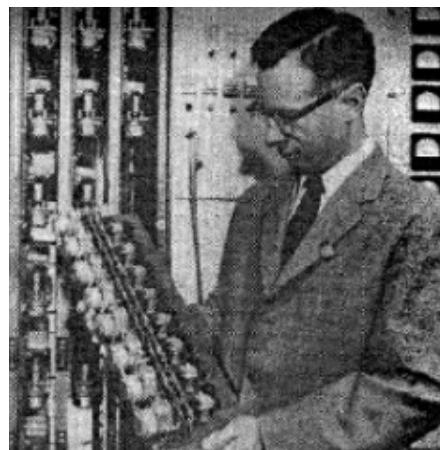
History of [Deep] Learning



1969: *Perceptrons* book,
Minsky and Papert

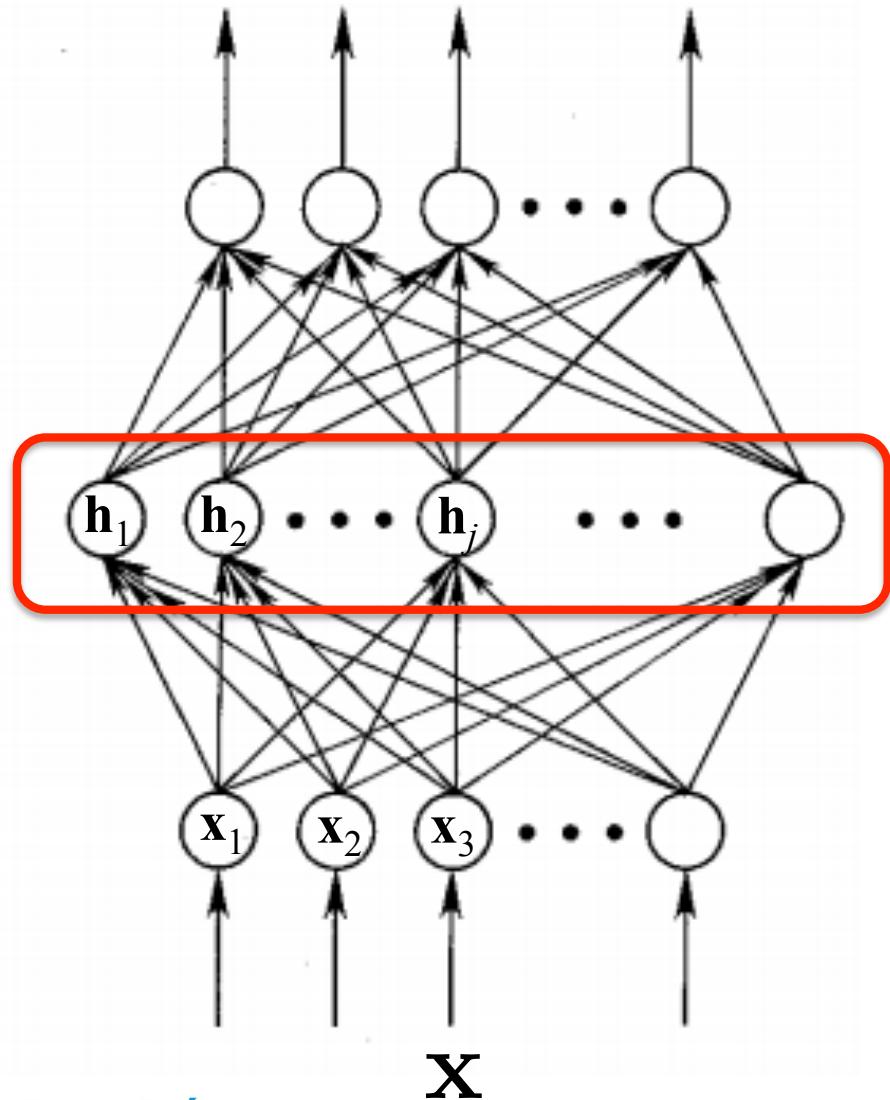


1958, Cornell:
Perceptron

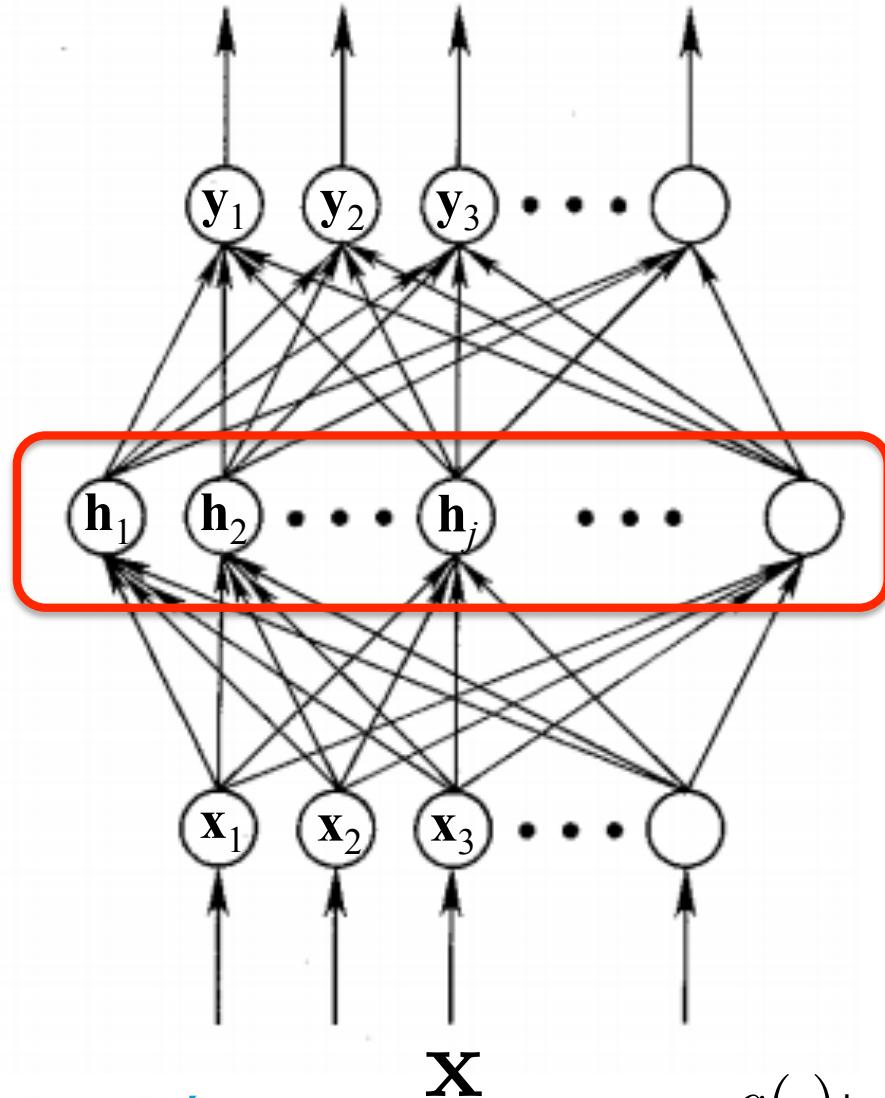


~1980: Multi-layer
networks (Rumelhart,
Hinton, Williams)

Multilayer Perceptron/Neural Network

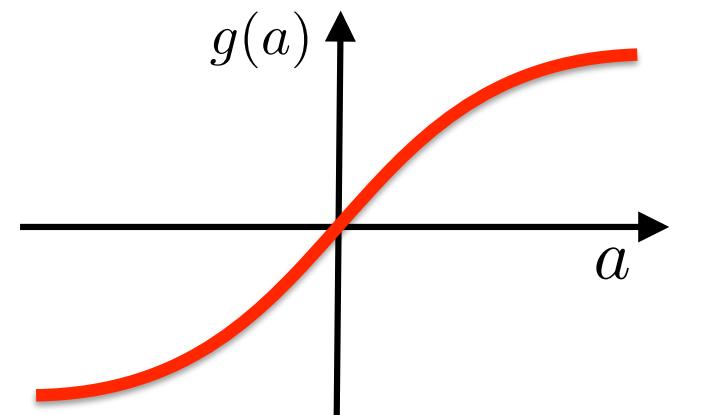


Multilayer Perceptron/Neural Network



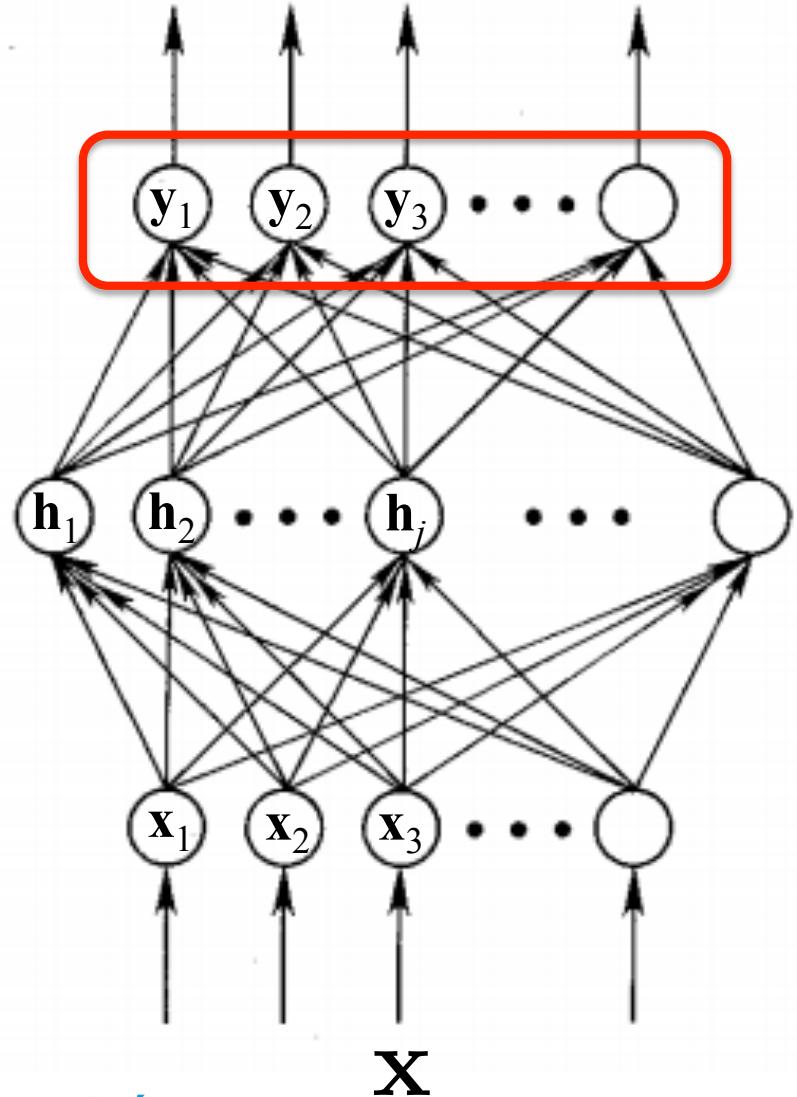
$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

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



$g(\cdot)$ 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

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

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

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

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

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

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

Supervised Machine Learning becomes regression!

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

Multilayer Perceptron: Optimization

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

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

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

How can we solve this optimization problem?

Multilayer Perceptron: Optimization

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

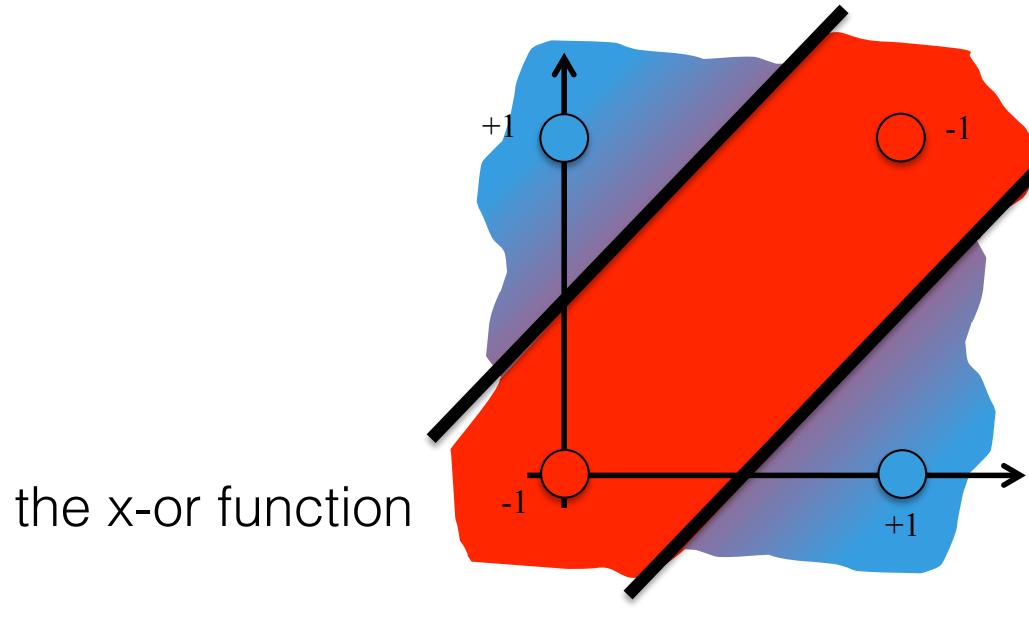
$$\mathbf{y}(\mathbf{x}) = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

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

How can we solve this optimization problem?

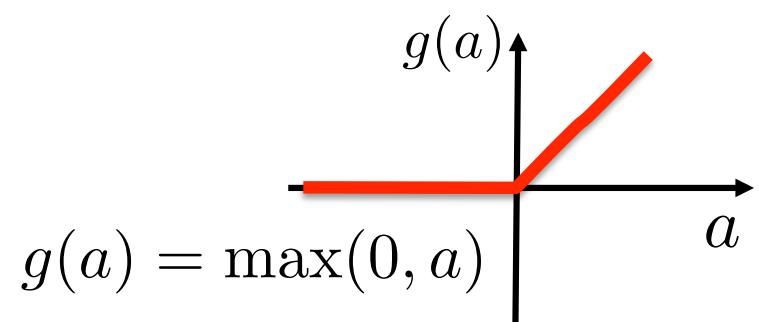
No closed-form method → gradient descent.

A Multilayer Perceptron Solves Non-Linearly Separable Problems



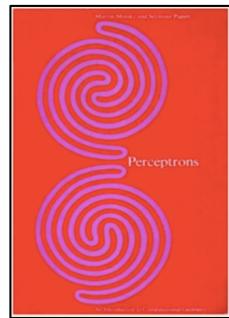
$$\mathbf{h} = g(\mathbf{Wx} + \mathbf{b})$$

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

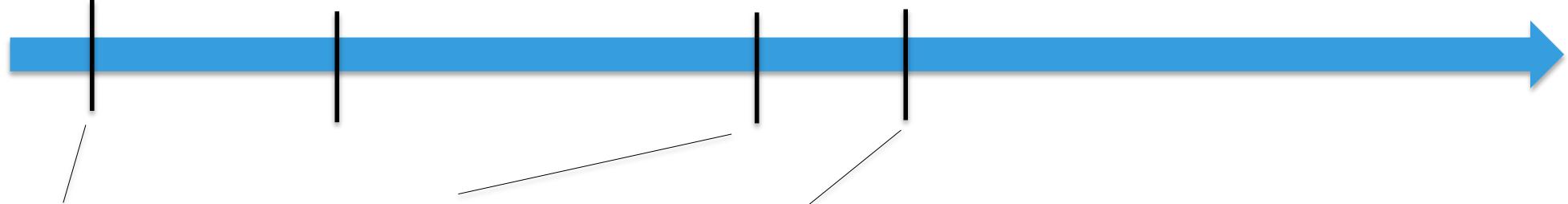


$$\mathbf{W} = \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} -1/2 \\ -1/2 \end{bmatrix}, \mathbf{w}_2 = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, b_2 = -1$$

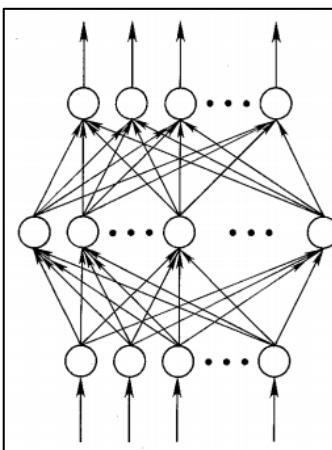
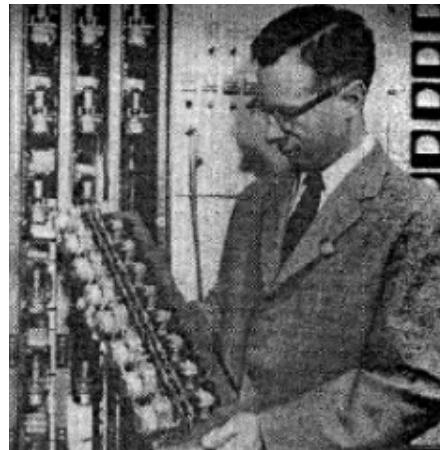
History of [Deep] Learning



1969: *Perceptrons* book,
Minsky and Papert

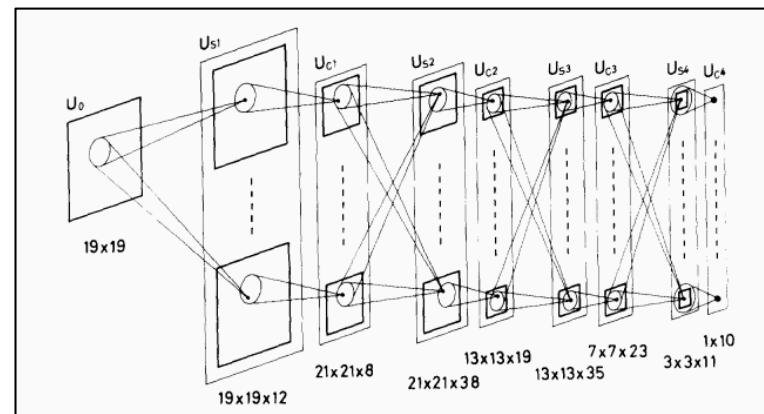


1958, Cornell:
Perceptron

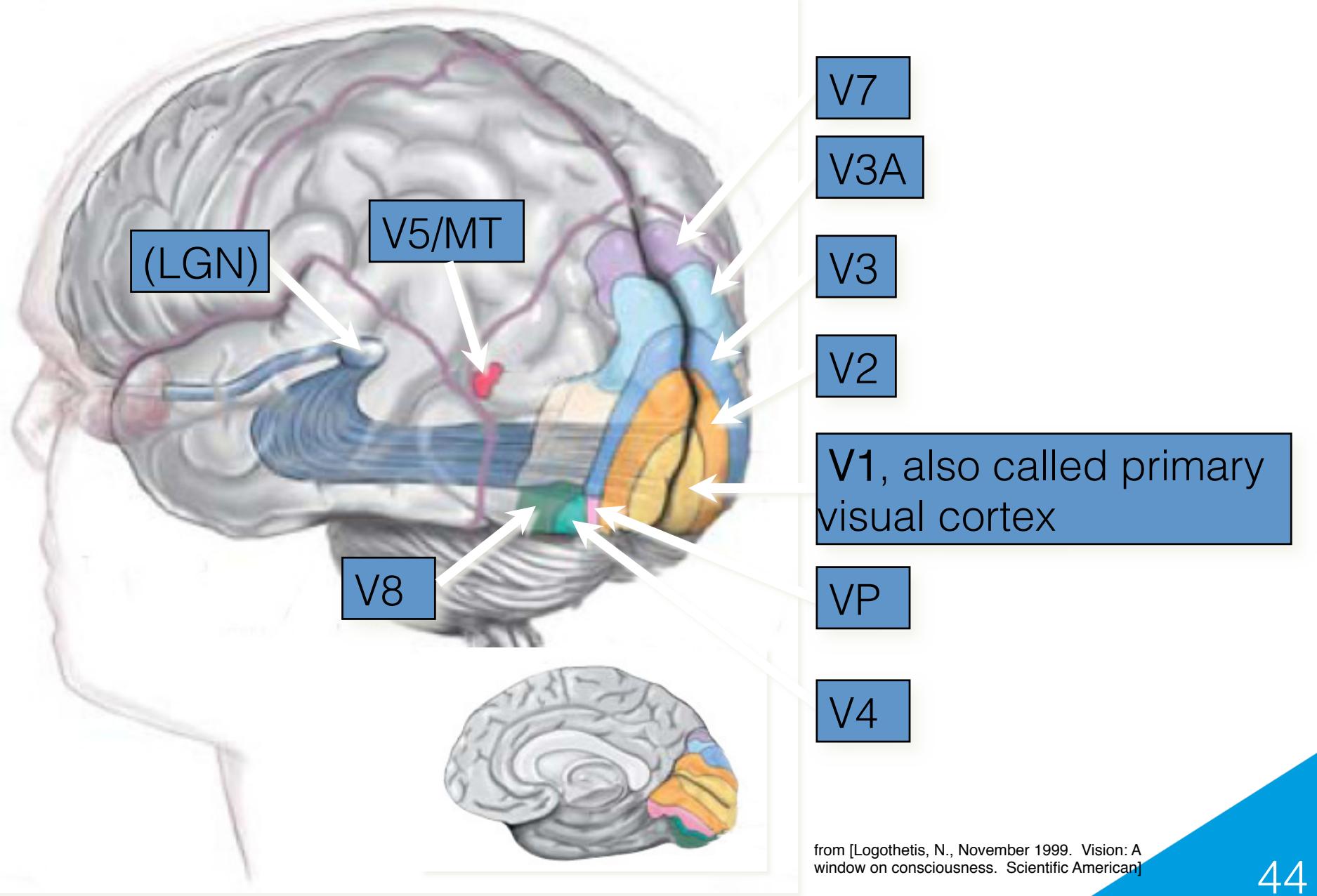


~1980: Multi-
layer networks

1987: Neocognitron,
Fukushima

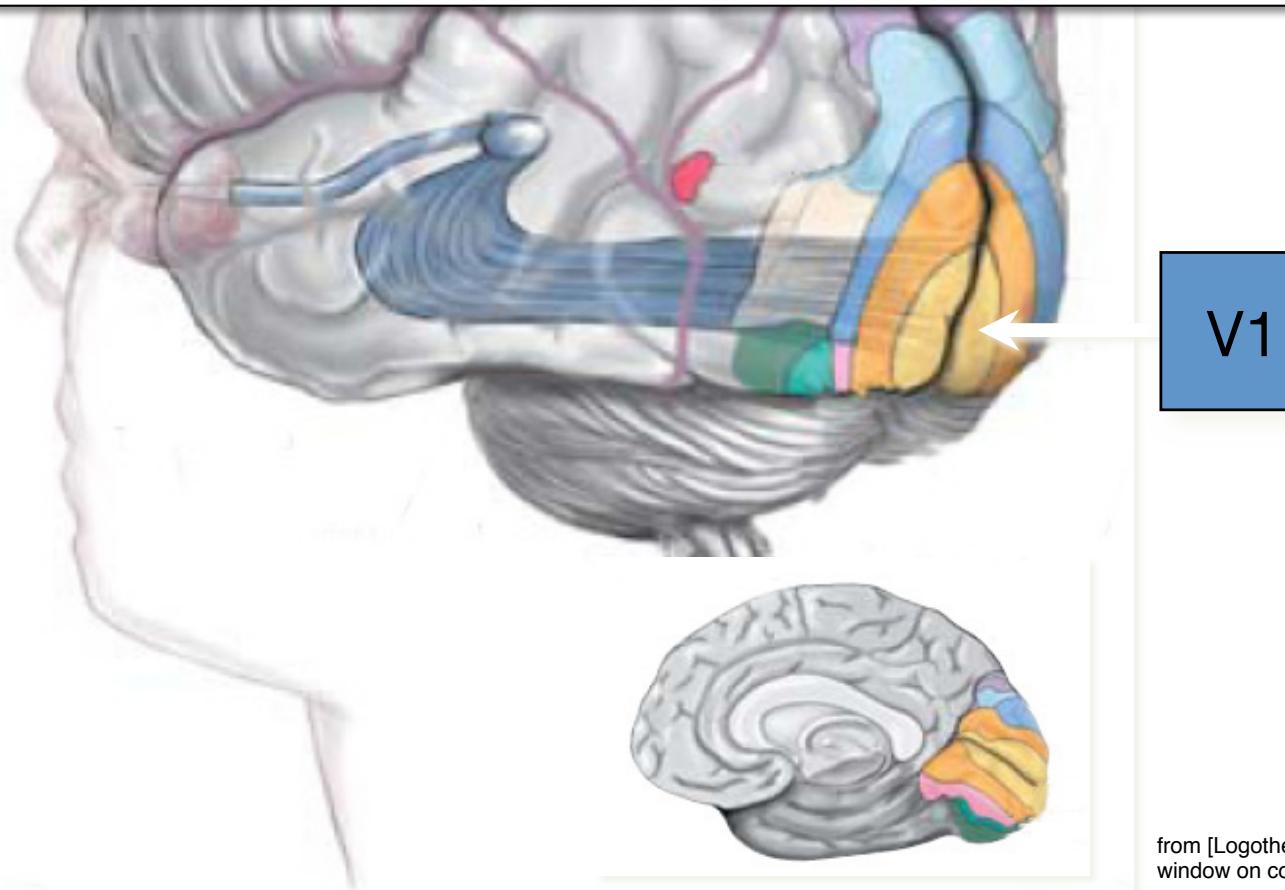


The Visual Cortex

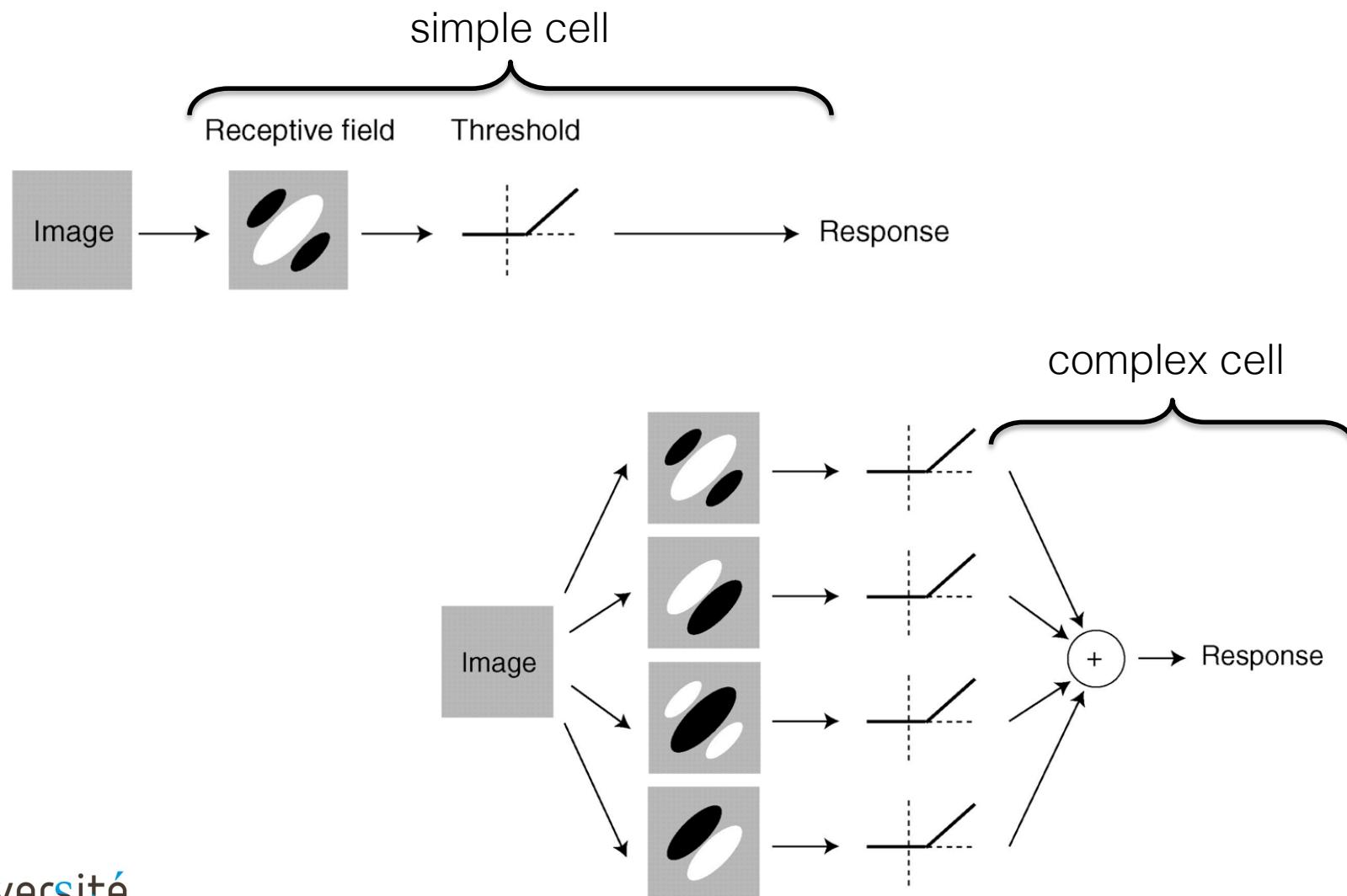


V1, the Primary Visual Cortex

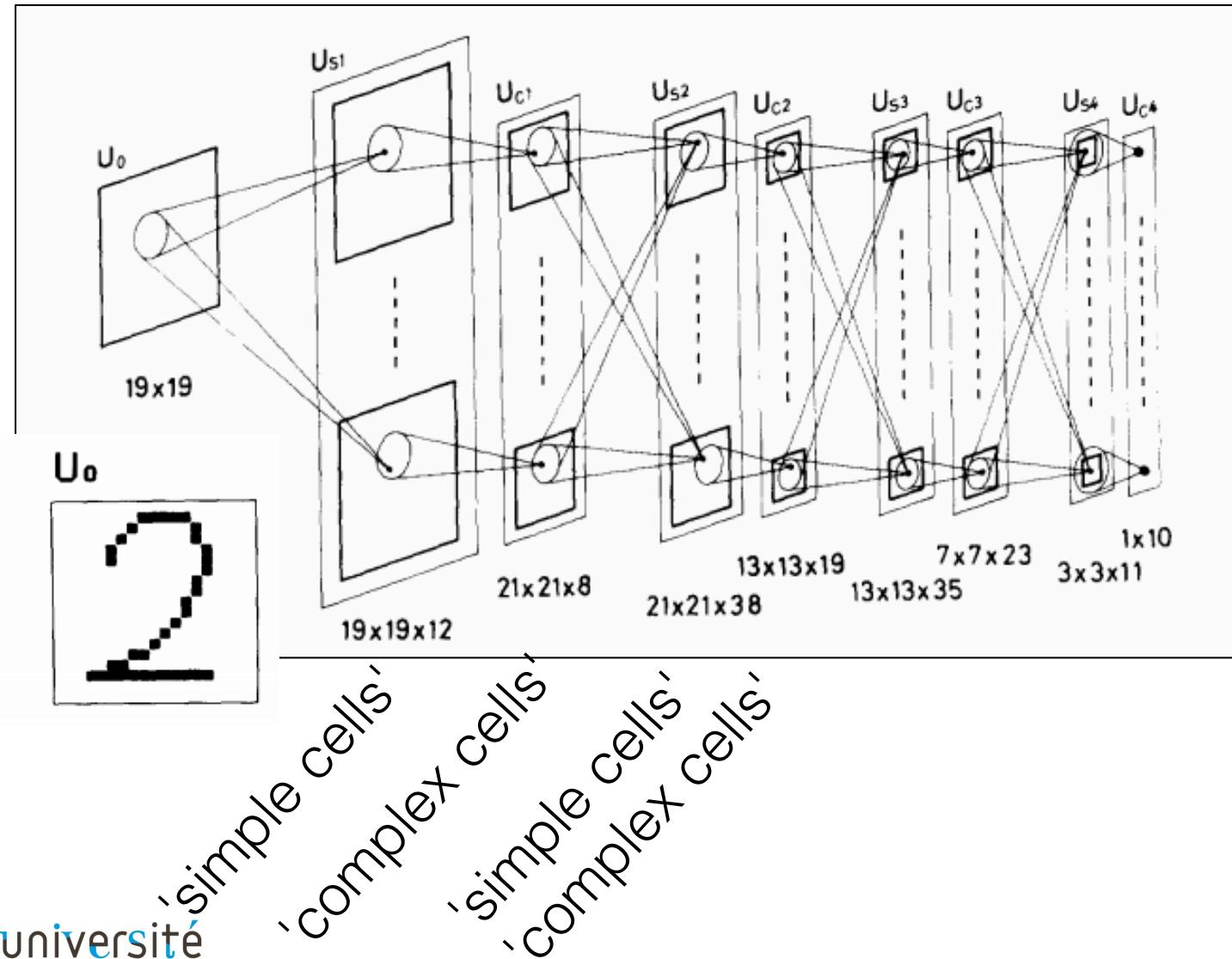
- Largest area in the visual cortex;
- 100 times more neurons than retinal ganglion cells (over-complete representation);



Hubel and Wiesel (1959)



Neocognitron (1987)

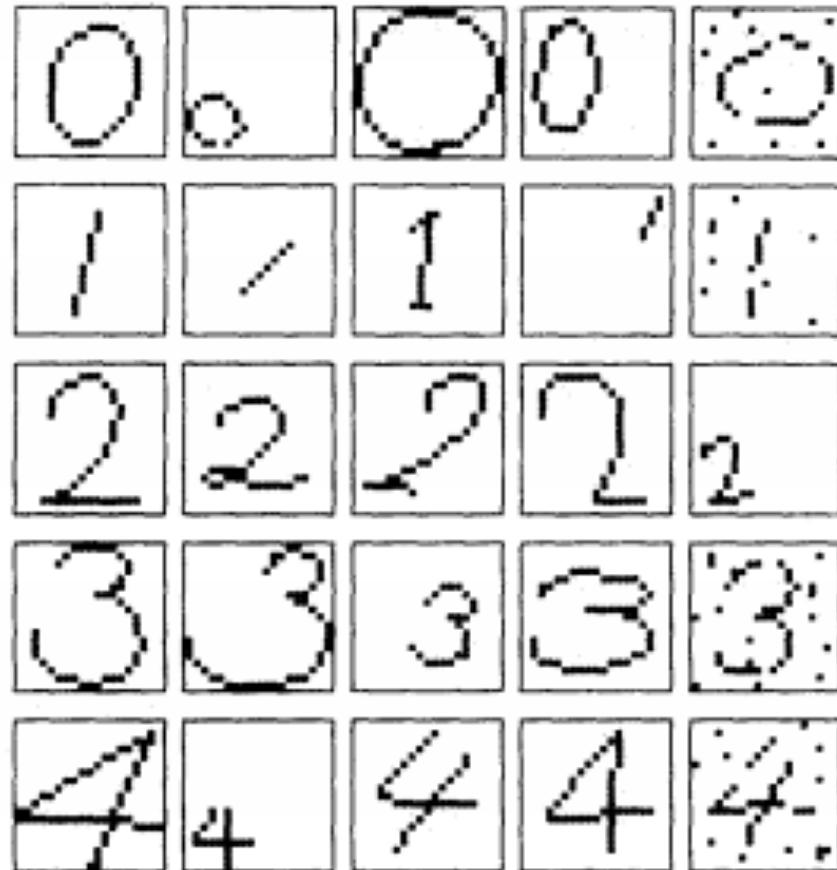


10 outputs

U_{C4}

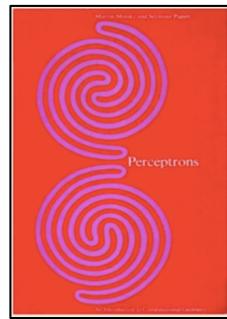
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

Neocognitron (1987)



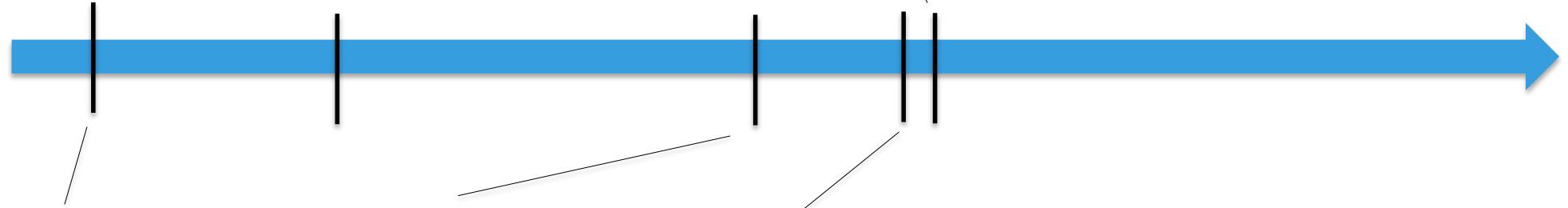
Some digits correctly recognized

History of [Deep] Learning

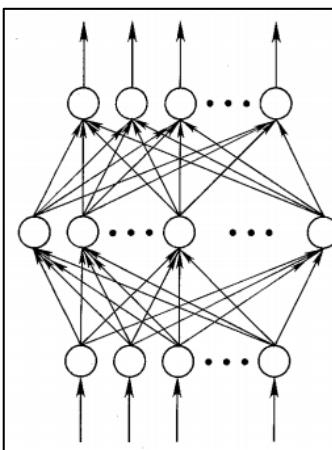
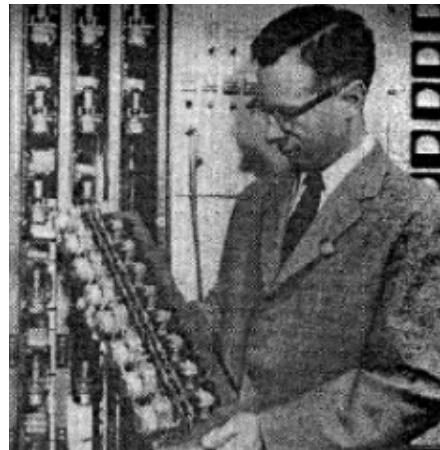


1969: *Perceptrons* book,
Minsky and Papert

1989: Universal
Approximation Theorem
(Hornik, Cybenko)

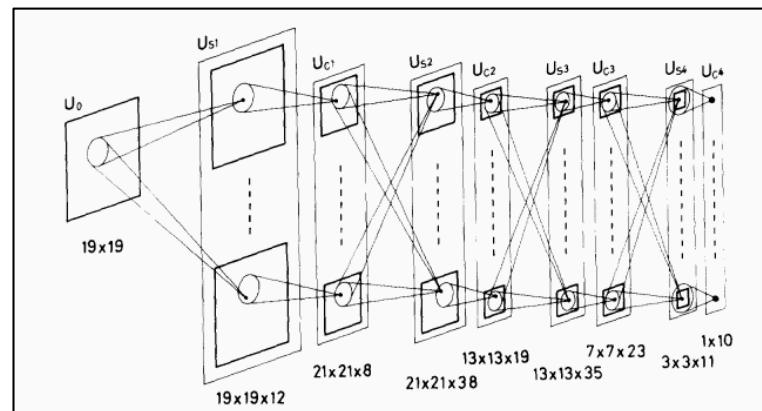


1958, Cornell:
Perceptron



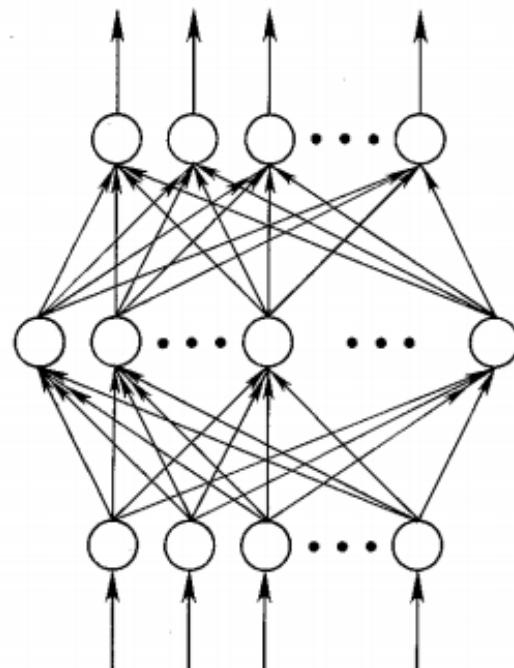
~1980: Multi-
layer networks

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:

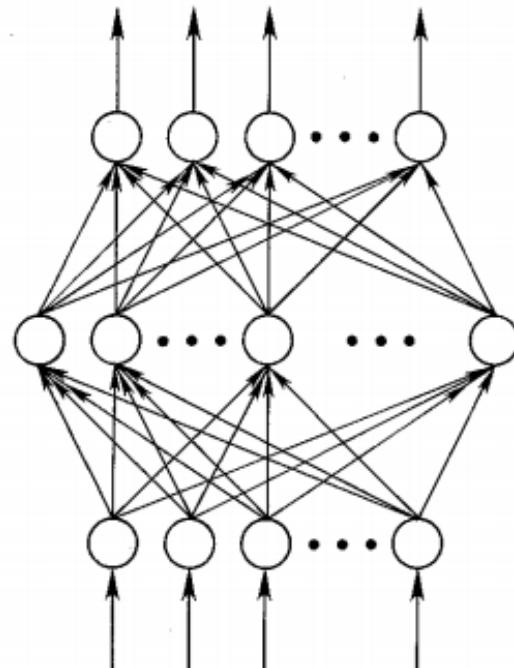


$$\begin{aligned} \mathbf{h} &= g(\mathbf{W}\mathbf{x} + \mathbf{b}) \\ \mathbf{y} &= \mathbf{W}_2^\top \mathbf{h} + \mathbf{b}_2 \end{aligned}$$

Universal Approximation: Actually...

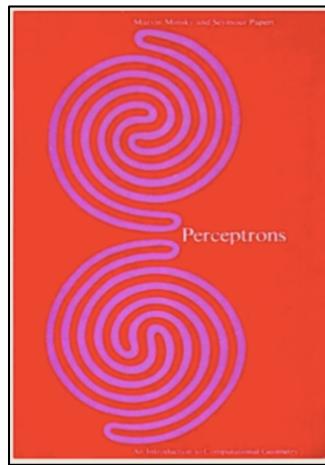
The theorem does not say how large the network needs to be;

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

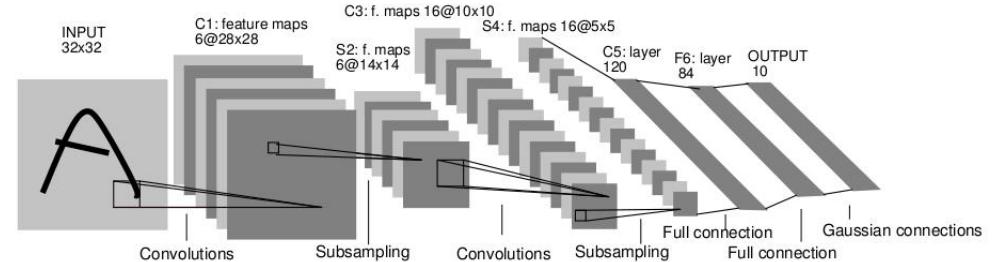


$$\begin{aligned} \mathbf{h} &= g(\mathbf{Wx} + \mathbf{b}) \\ \mathbf{y} &= \mathbf{W}_2^\top \mathbf{h} + \mathbf{b}_2 \end{aligned}$$

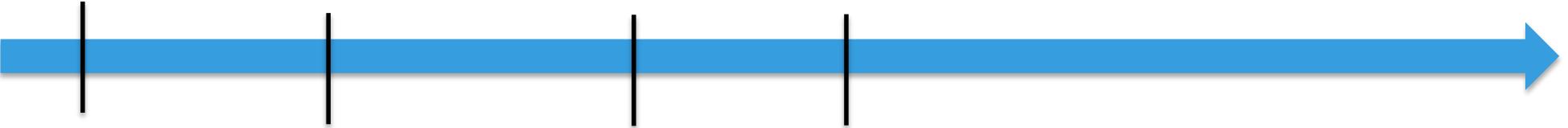
History of [Deep] Learning



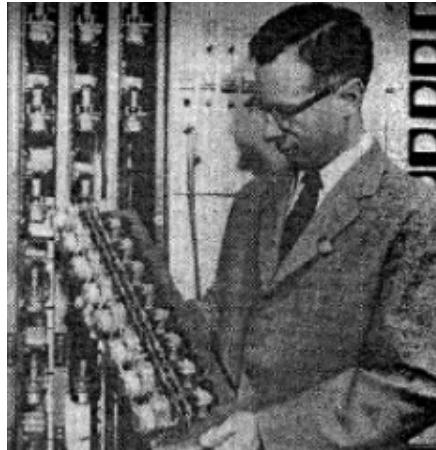
1969: *Perceptrons* book,
Minsky and Papert



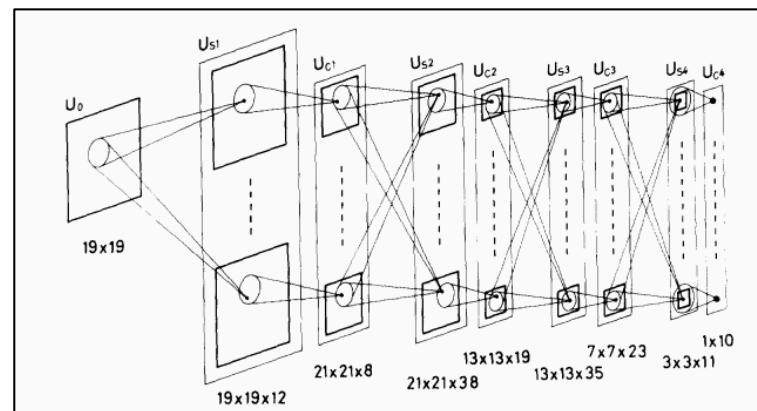
1992: LeNet, LeCun



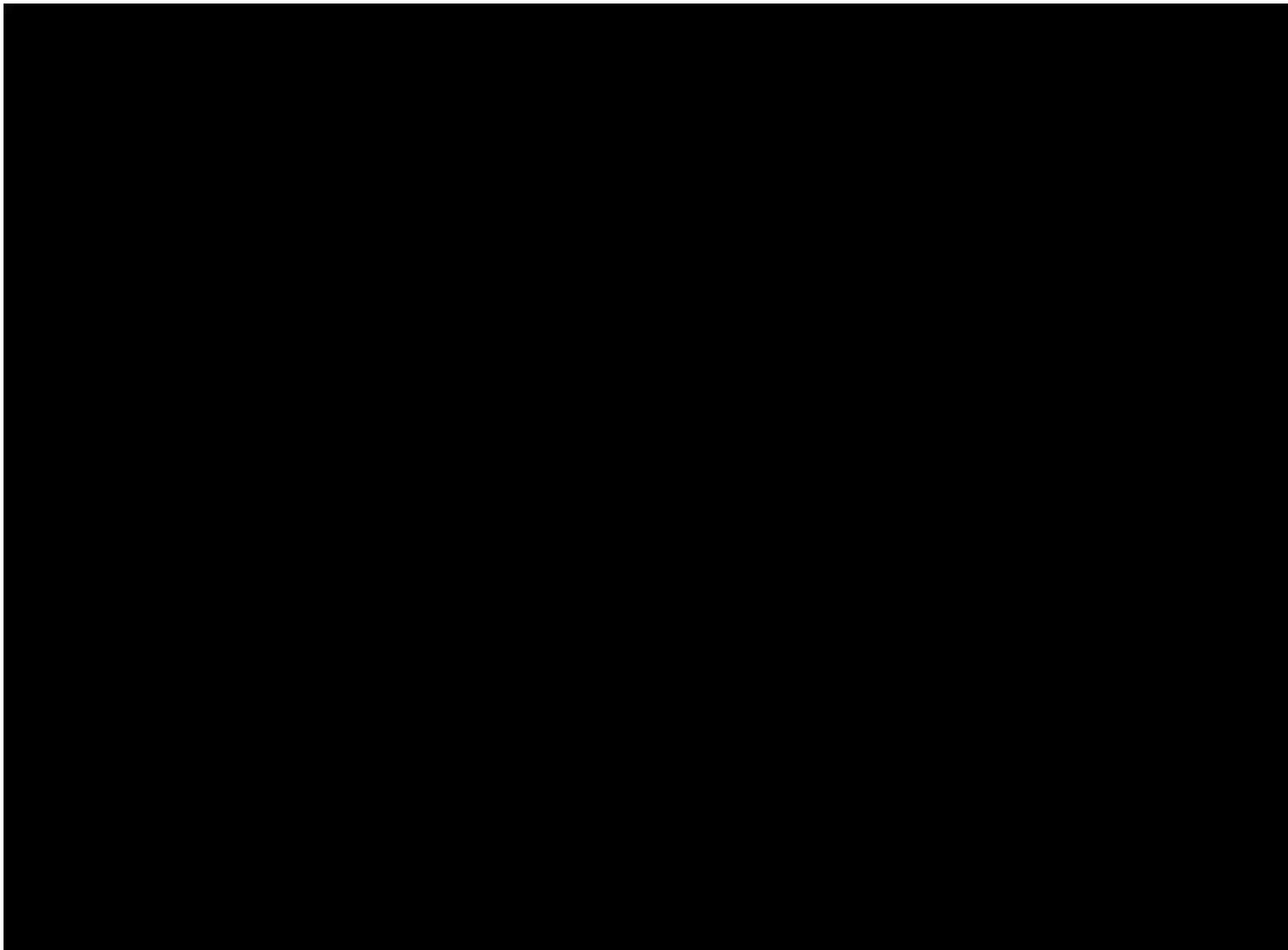
1958, Cornell: Perceptron



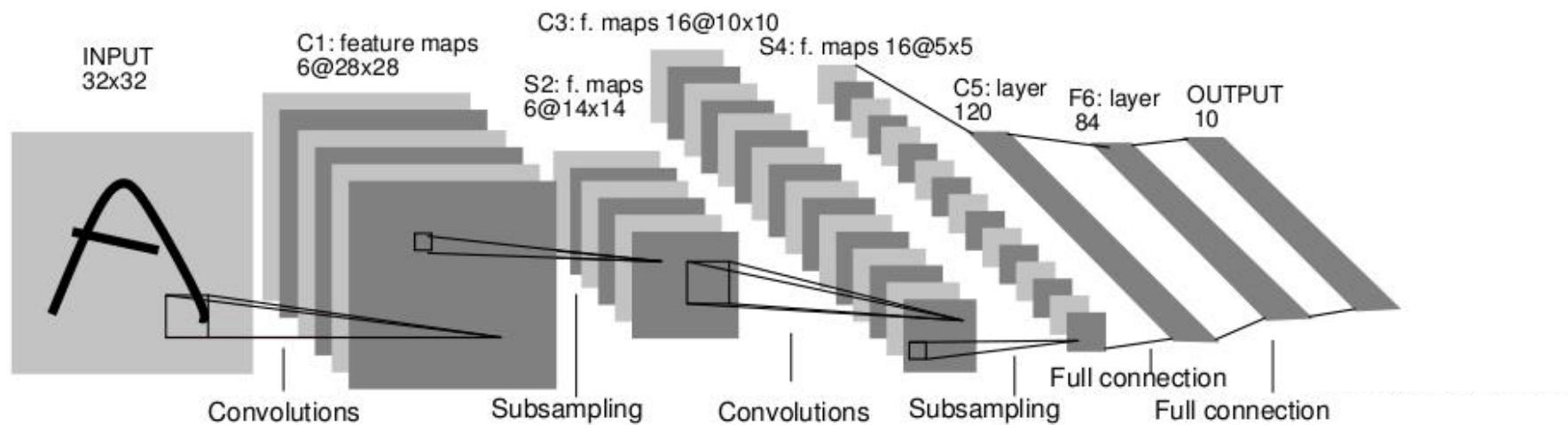
1987: Neocognitron,
Fukushima



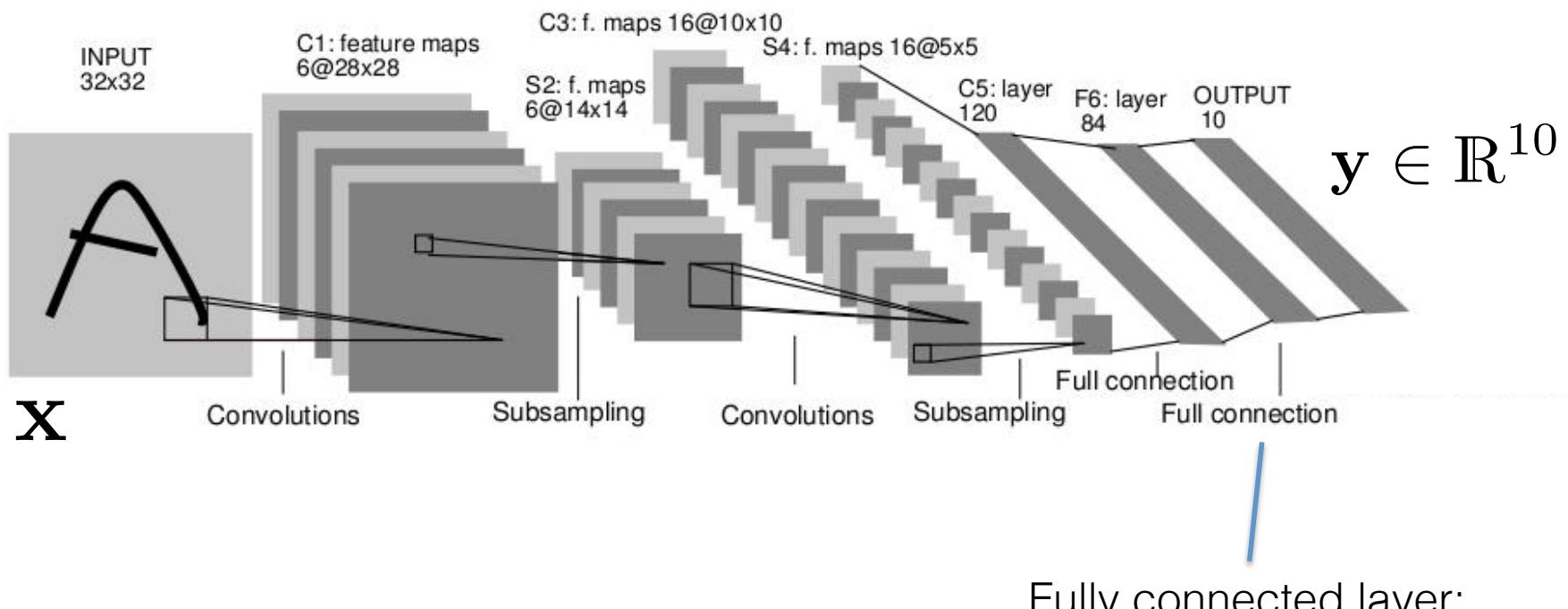
LeNet5 (LeCun, 1992)



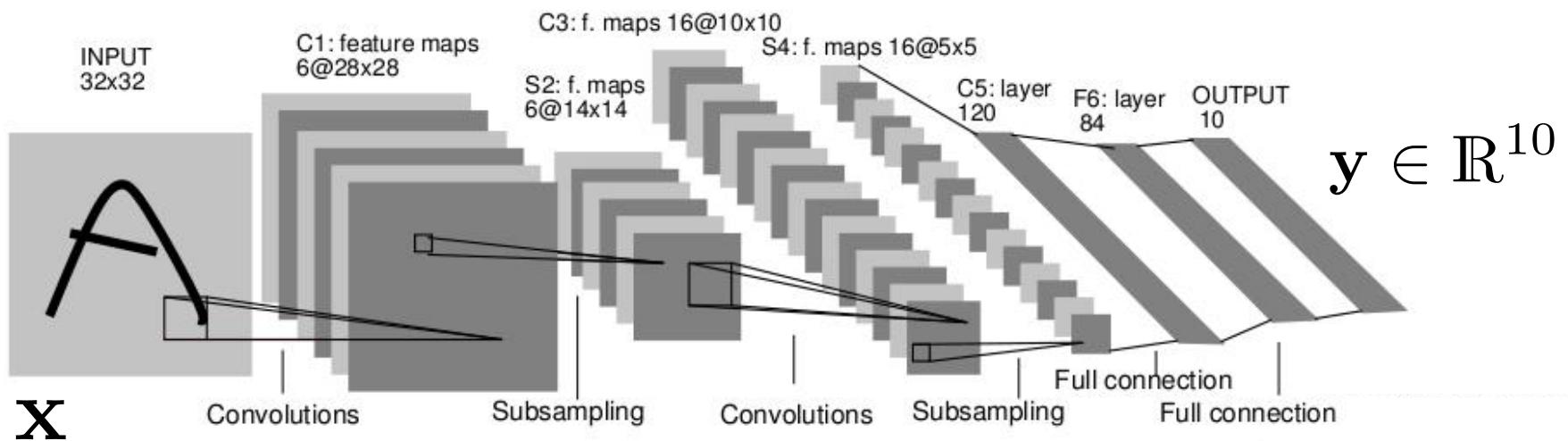
Deep Learning *a la* LeCun



Deep Learning *a la* LeCun



Deep Learning *a la* LeCun



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

Convolutional layer:

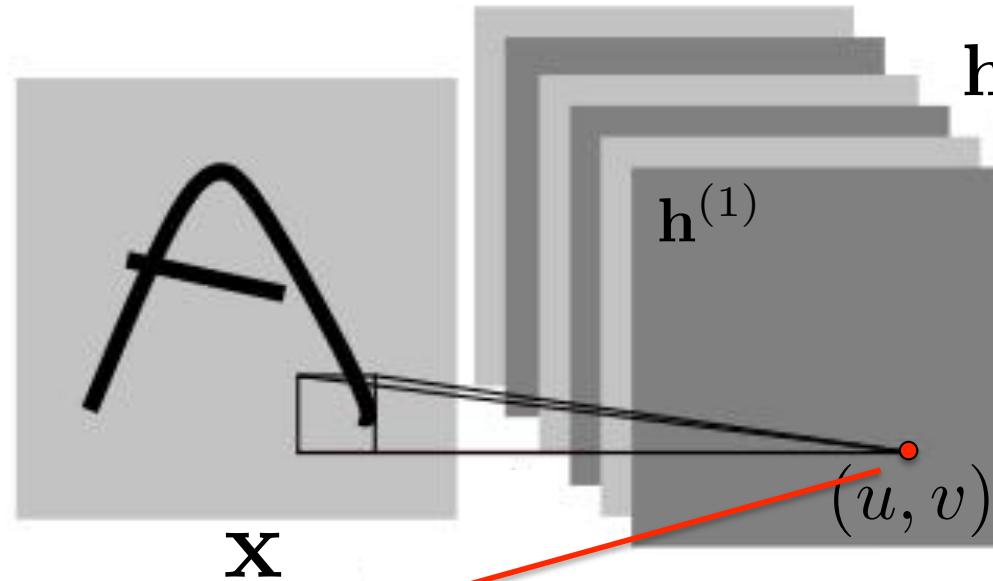
$$\mathbf{h}^{(i)} = g(\mathbf{f}^{(i)} * \mathbf{x})$$

Fully connected layer:

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

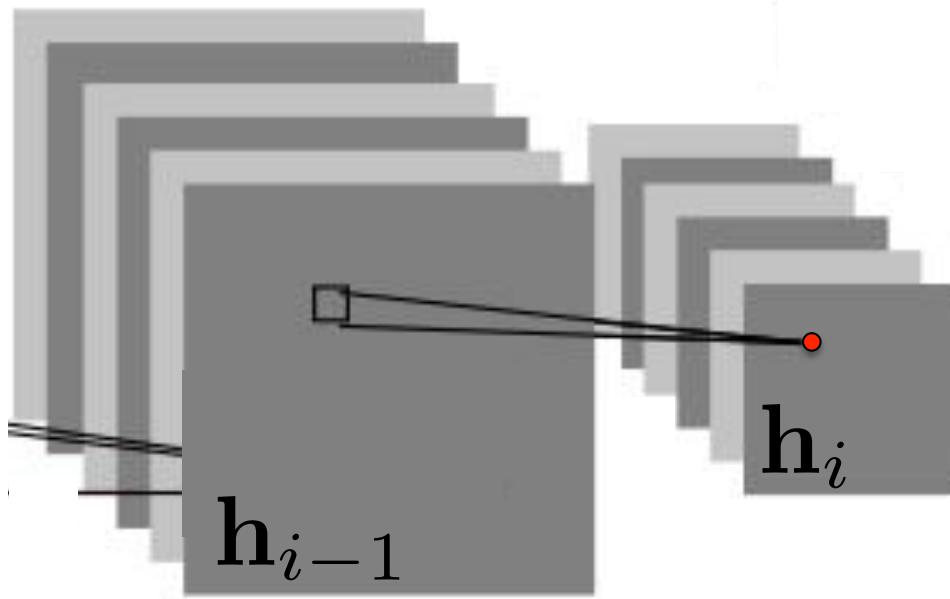
More efficient for images

Convolutional Layer



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

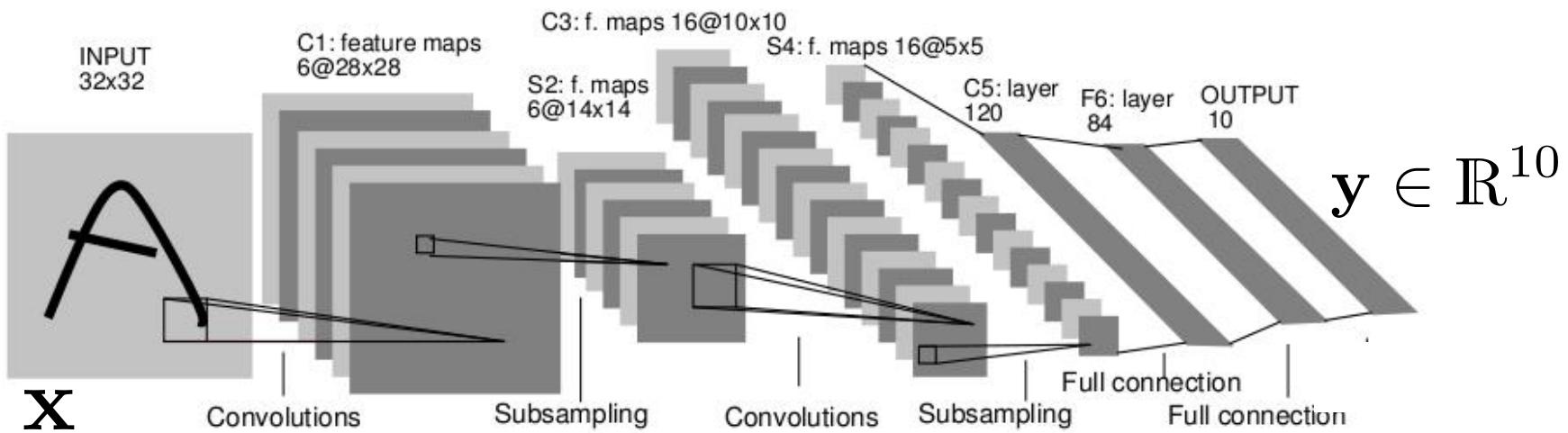
Pooling Layer



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



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

$$\mathbf{h}_2 = [\text{pool}(\mathbf{h}_{1,1}), \dots, \text{pool}(\mathbf{h}_{1,n})]$$

$$\mathbf{h}_3 = [g(\mathbf{f}_{3,1} * \mathbf{h}_{2,1}), \dots, g(\mathbf{f}_{3,n} * \mathbf{h}_{2,n})]$$

$$\mathbf{h}_4 = [\text{pool}(\mathbf{h}_{3,1}), \dots, \text{pool}(\mathbf{h}_{3,n})]$$

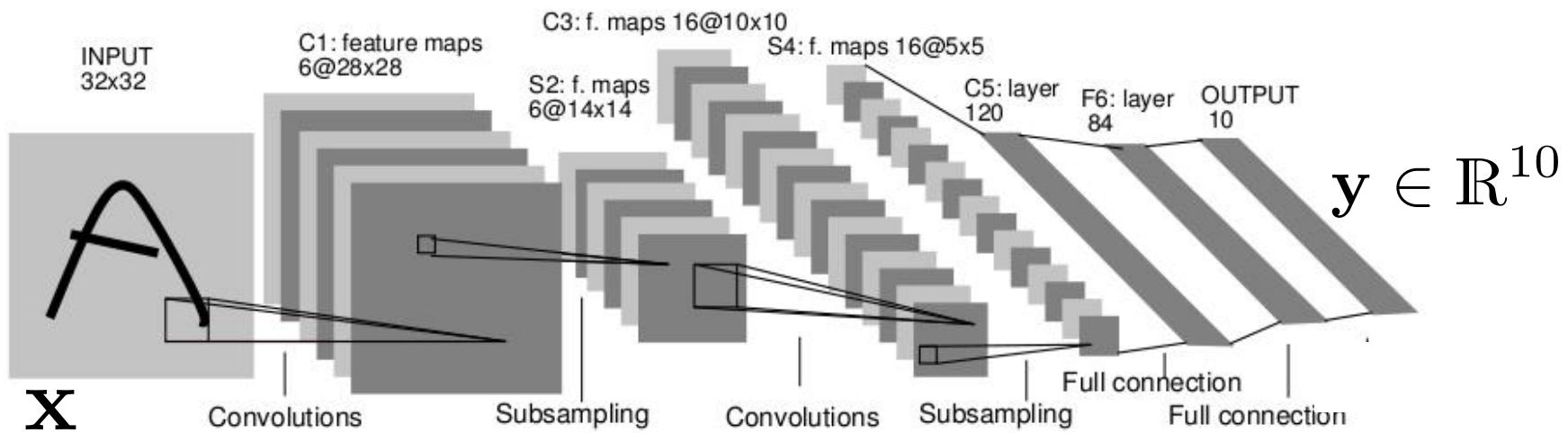
$$\mathbf{h}'_4 = \text{Vec}(\mathbf{h}_4)$$

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

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

$$\mathbf{y} = \mathbf{W}_7 \mathbf{h}_6 + \mathbf{b}_7$$

A Complete Deep Network



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

$$\mathbf{h}_2 = [\text{pool}(\mathbf{h}_{1,1}), \dots, \text{pool}(\mathbf{h}_{1,n})]$$

$$\mathbf{h}_3 = [g(\mathbf{f}_{3,1} * \mathbf{h}_{2,1}), \dots, g(\mathbf{f}_{3,n} * \mathbf{h}_{2,n})]$$

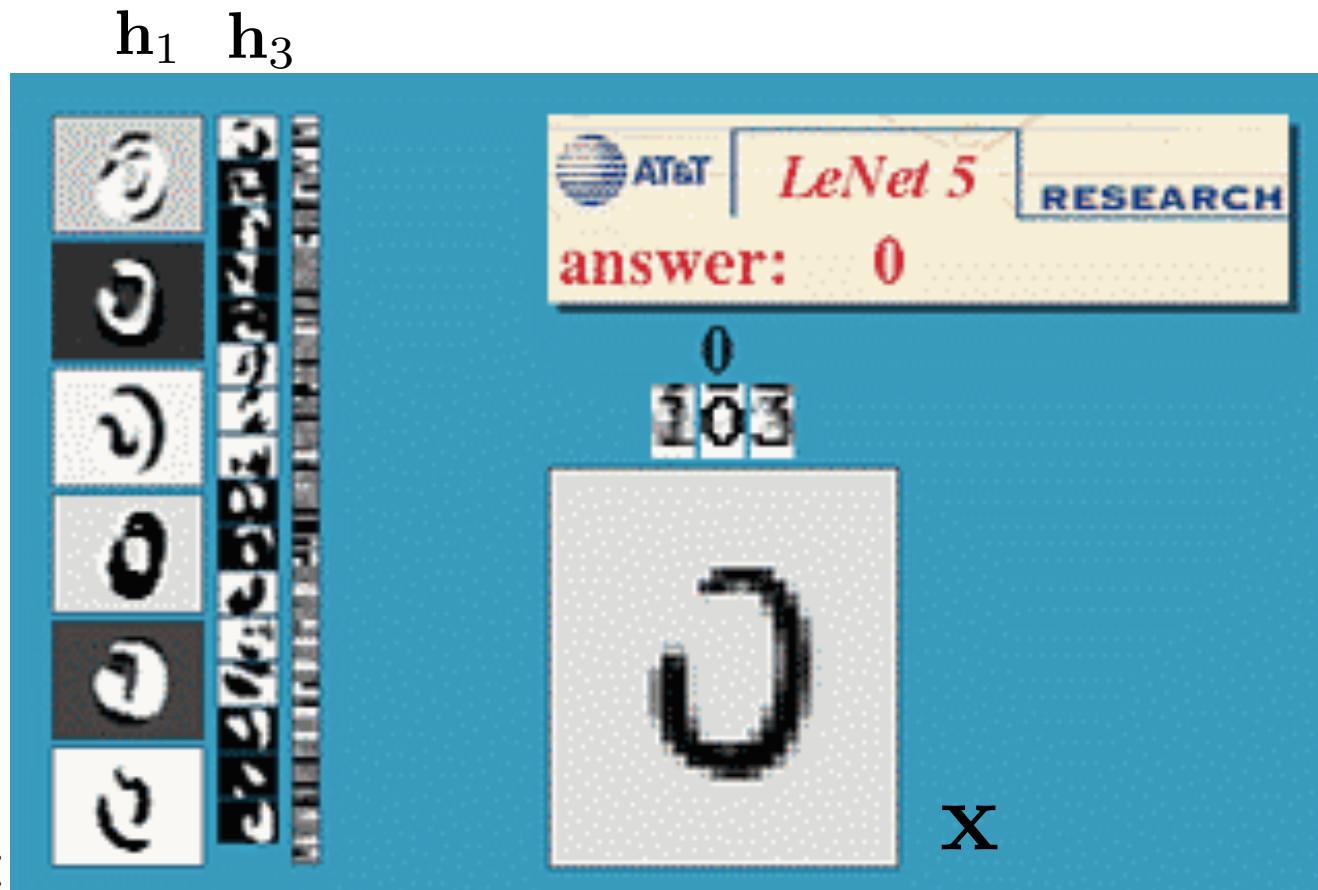
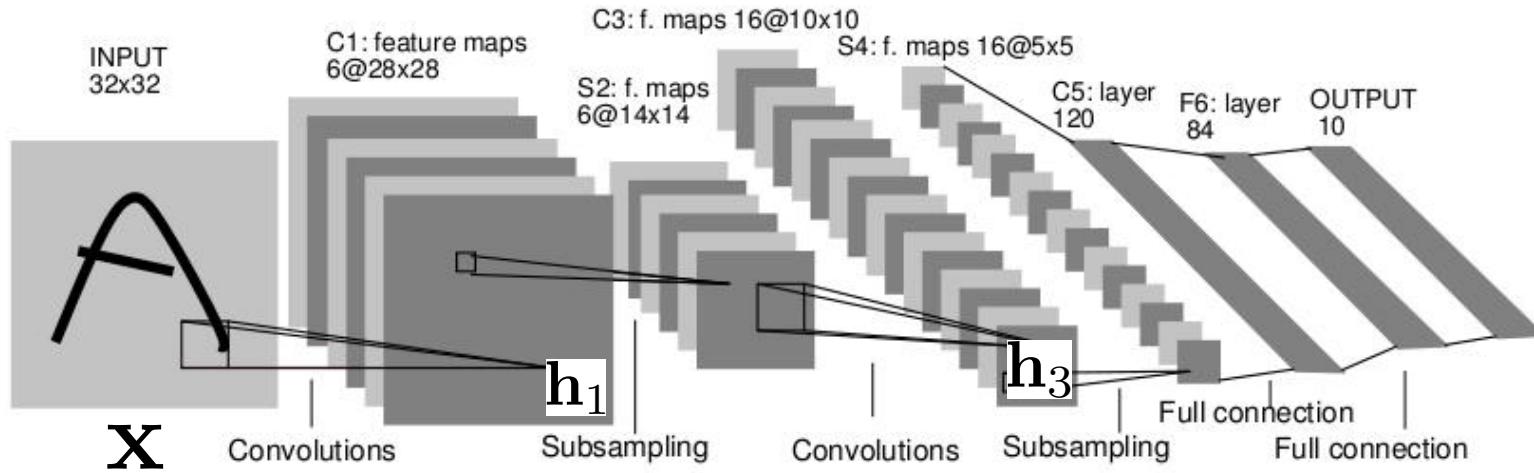
$$\mathbf{h}'_4 = \text{Vec}(\mathbf{h}_4)$$

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

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

$$\text{un}\mathbf{y} = \mathbf{W}_7 \mathbf{h}_6 + \mathbf{b}_7$$

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



Loss Function

Do NOT use 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

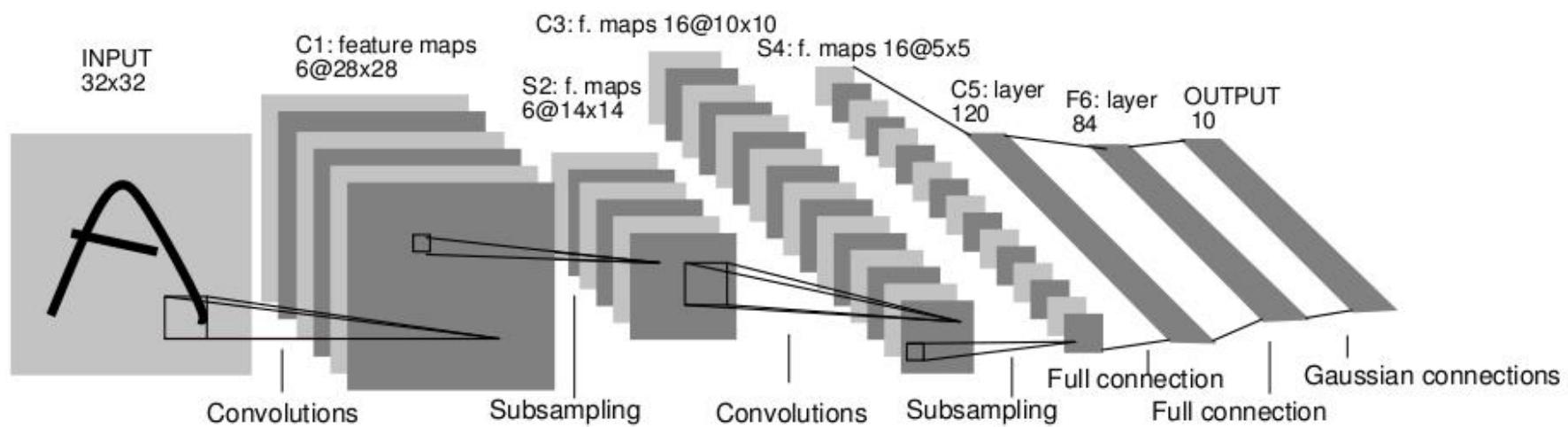
Desired output for the training sample

Predicted output for the training sample

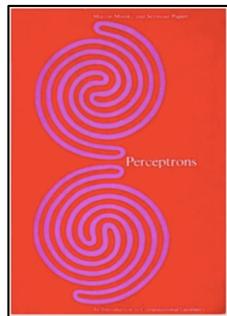
Optimization: stochastic gradient descent

00000000000000000000
11111111111111111111
22222222222222222222
33333333333333333333
44444444444444444444
55555555555555555555
66666666666666666666
77777777777777777777
88888888888888888888
99999999999999999999

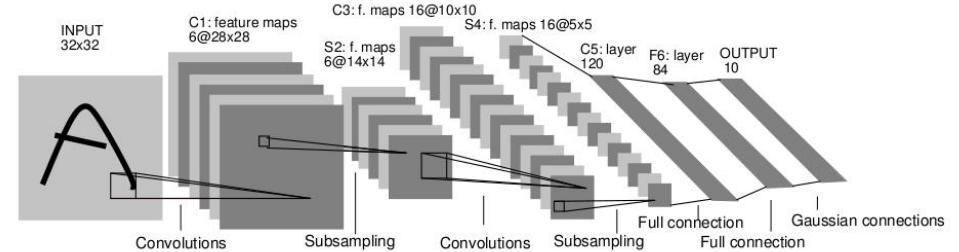
How to Choose the Number of Layers? The Number of Filters per Layer?



History of [Deep] Learning

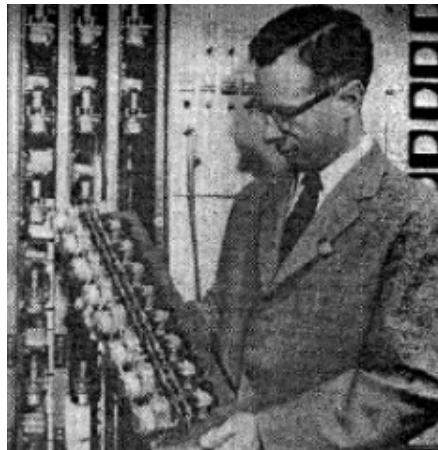


1969: *Perceptrons* book,
Minsky and Papert

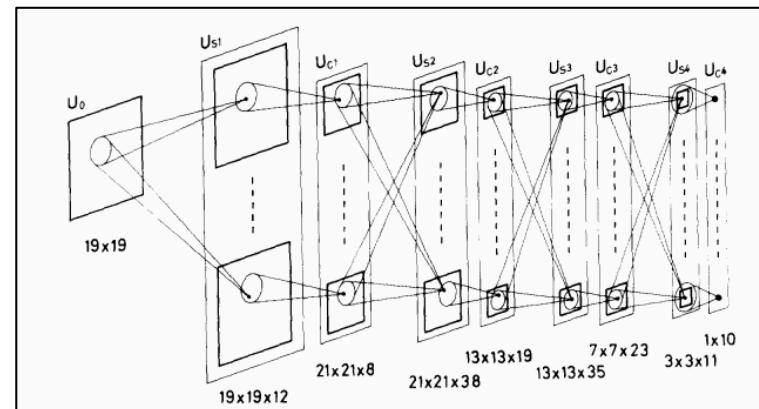


1992: LeNet, LeCun

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.

Unsupervised Pretraining

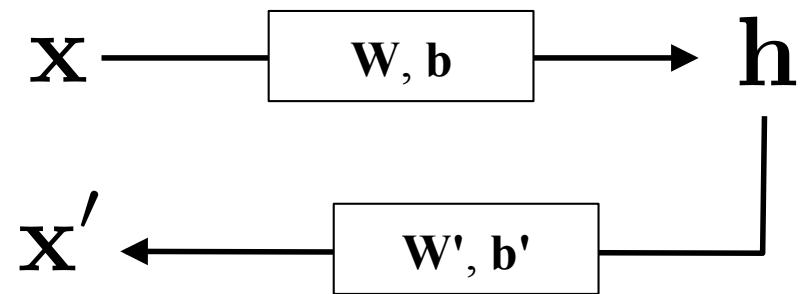
Example: Simple autoencoder:

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

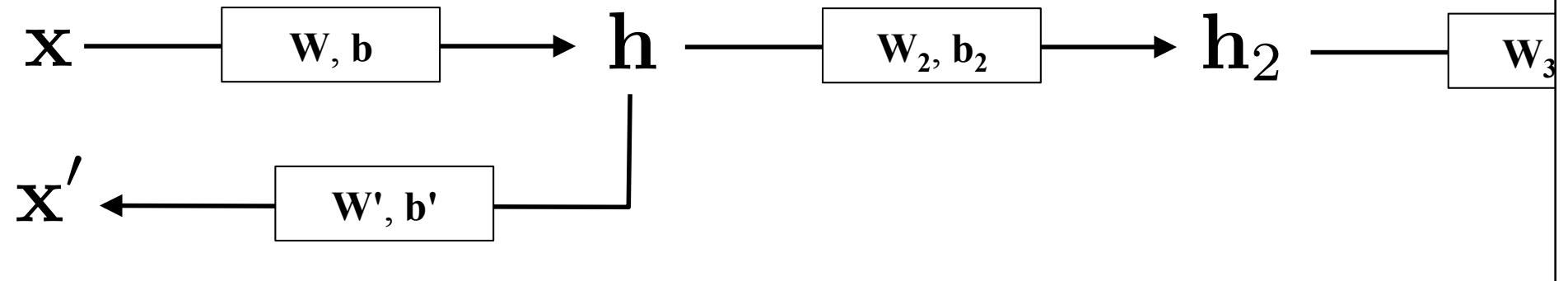
$$\text{with } \mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

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

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

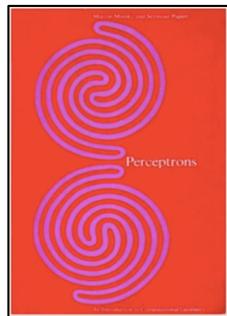


Unsupervised Pretraining

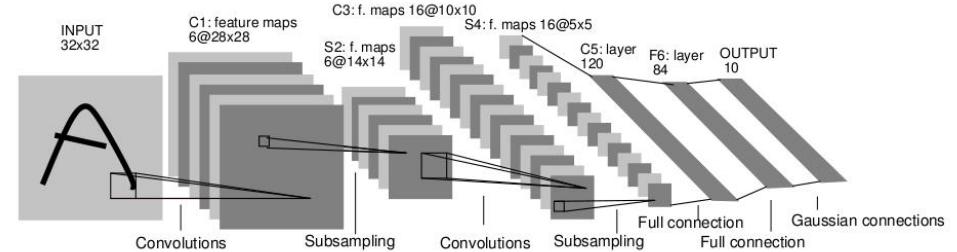


+ finetuning

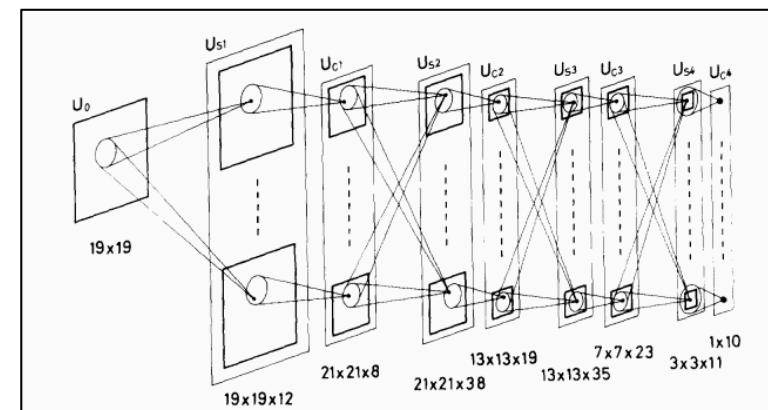
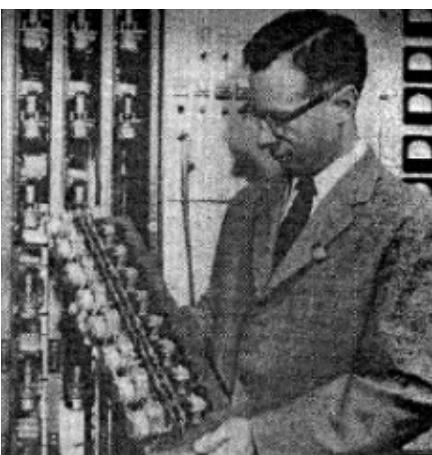
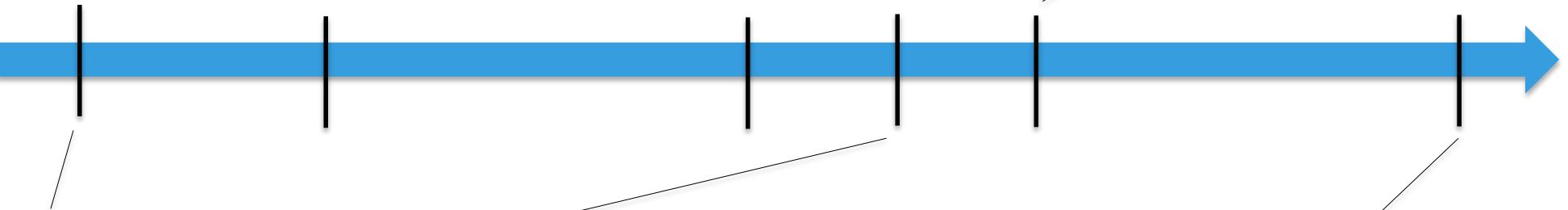
History of [Deep] Learning



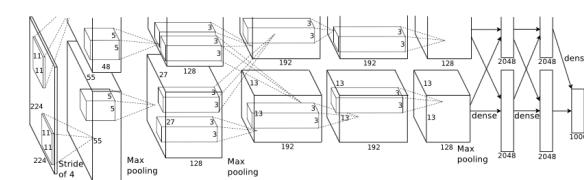
1969: *Perceptrons* book,
Minsky and Papert



1992: LeNet, LeCun

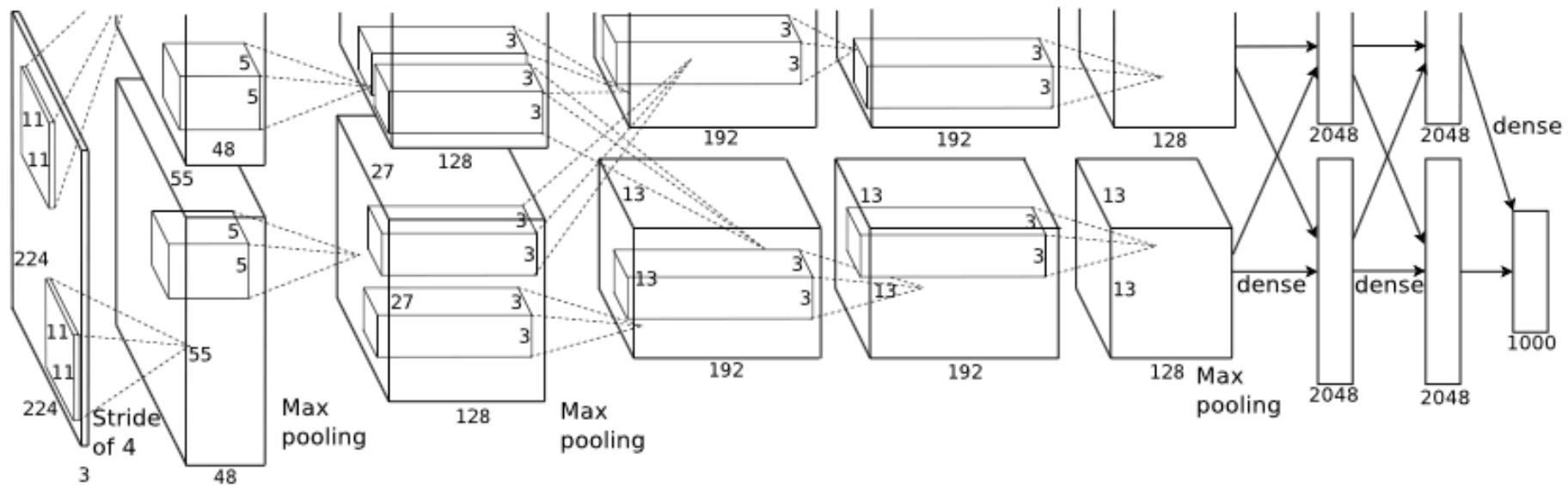


1987: Neocognitron,
Fukushima



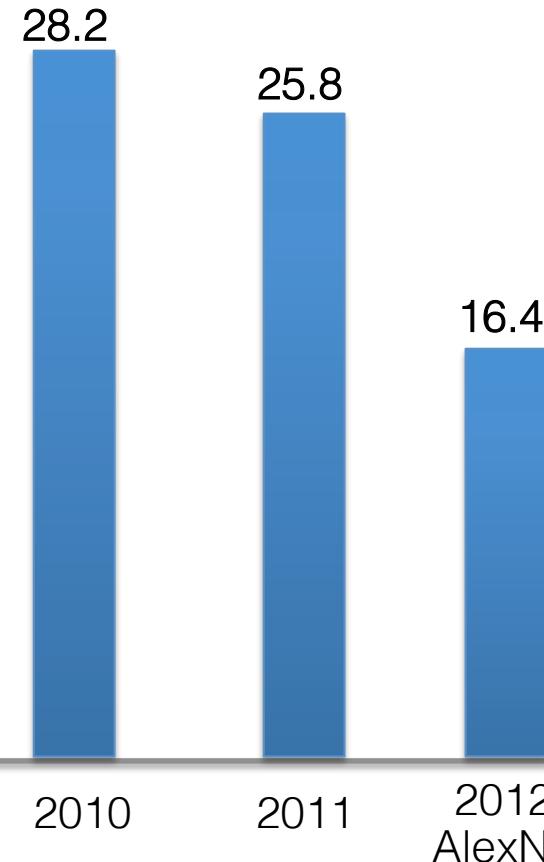
2010: AlexNet

AlexNet (2010)



Training time: 2 weeks on 2 GPUs

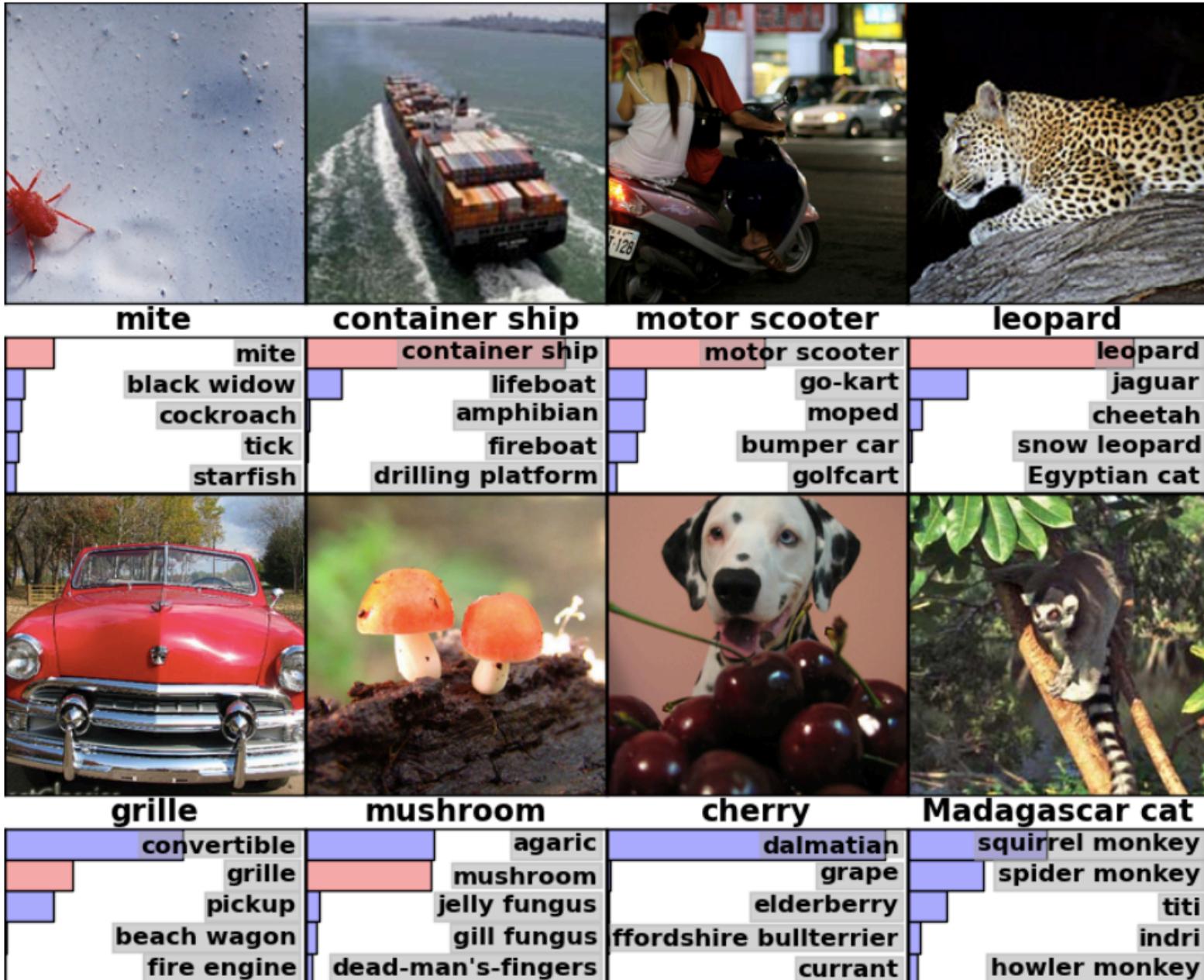
AlexNet (2010)



Classification task on ImageNet
challenge top-5 error.



AlexNet: Some results on ILSVRC-2010

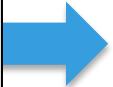


History of Deep Learning

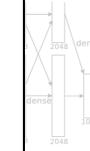
2013: Deep Q-Learning



connections



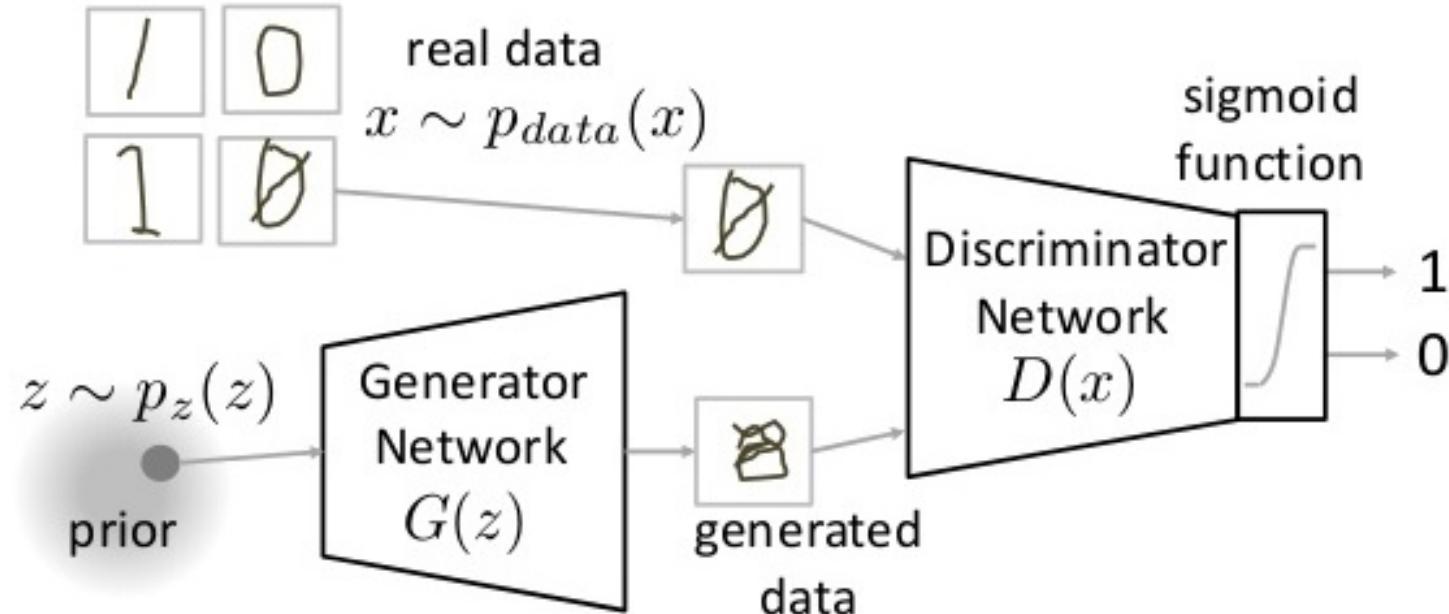
et



72

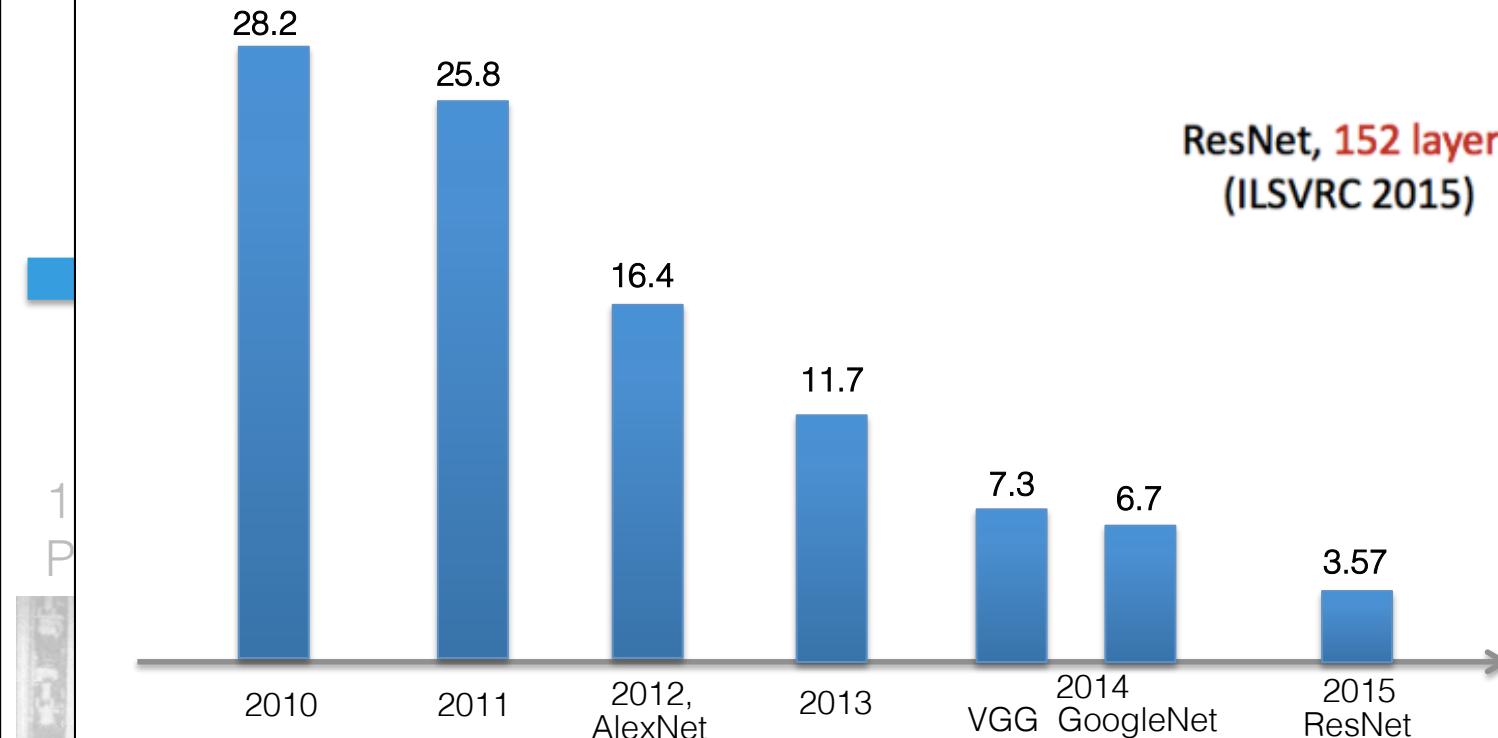
History of Deep Learning

2014: Generative Adversarial Networks



History of Deep Learning

2015: ResNet

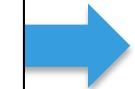


Classification task on ImageNet challenge top-5 error.

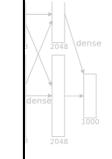
ResNet, 152 layers
(ILSVRC 2015)



connections



et



Why Now?

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

DeepFool

$$\arg \min_{\mathbf{r}} \|\mathbf{r}\|_2 \text{ subject to: } \forall (\mathbf{x}, d) \in \mathcal{T} \quad \mathbf{y}(\mathbf{x} + \mathbf{r}) \neq d$$



'Universal' perturbation \mathbf{r} for
GoogLeNet

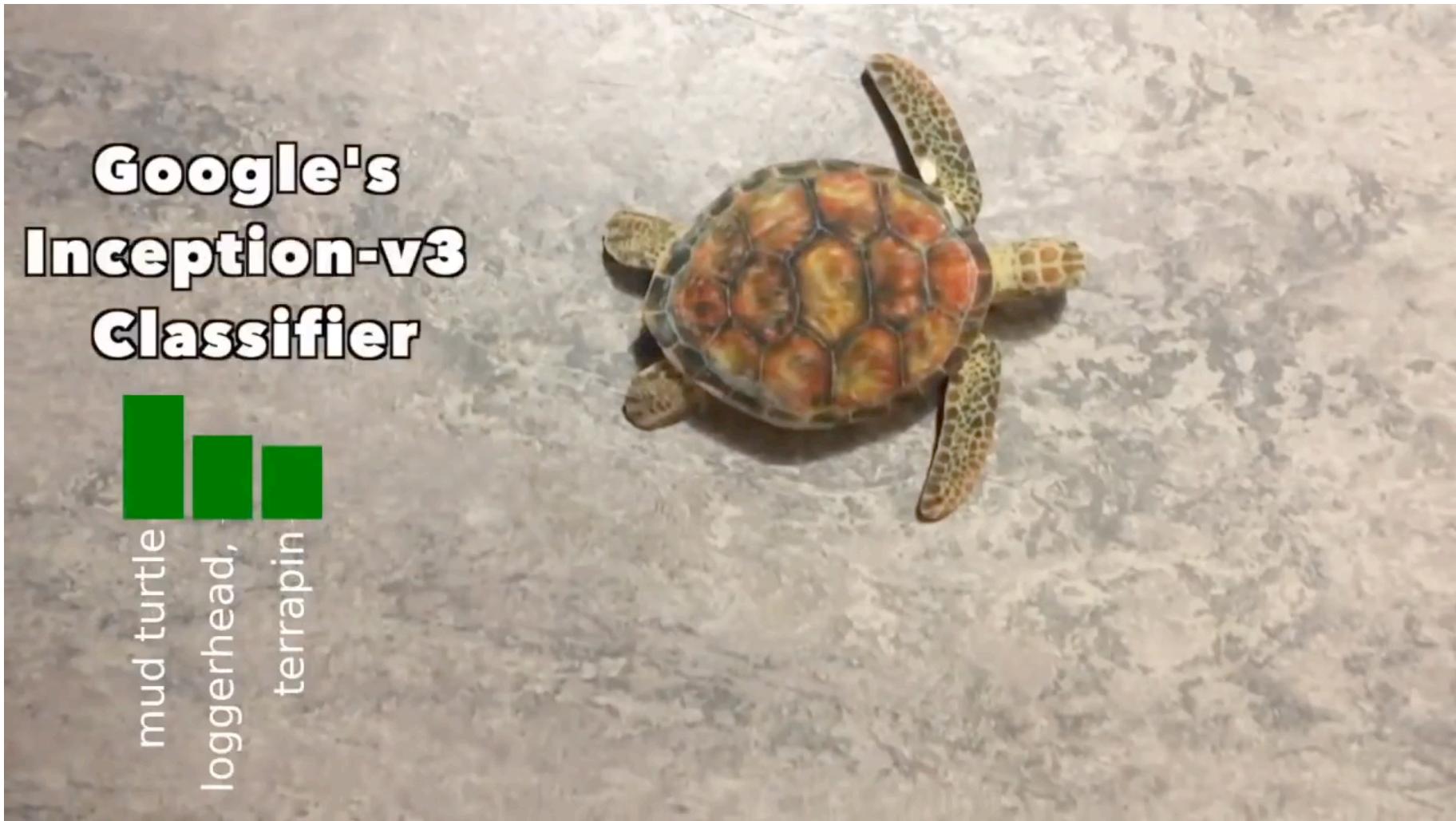


'Indian elephant'



'Macaw'

DeepFool



**Google's
Inception-v3
Classifier**

