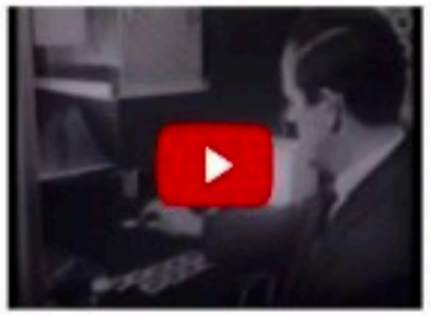
# Deep Learning
# Perceptron, Gradient descent, Multilayer Perceptron, Backprop, CNNs

Sergi Pujades

IntroIA (ENSIMAG - 2A)
Intelligent Systems (MOSIG - M1)



**1958** Perceptron

**1974** Backpropagation

Convolution Neural Networks for Handwritten Recognition

**1998**

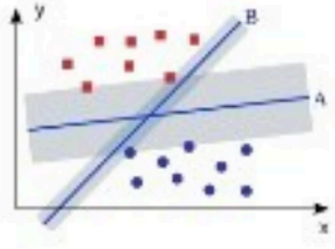
Google Brain Project on 16k Cores

**2012**

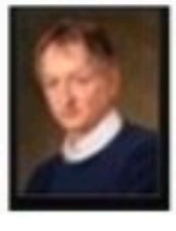
awkward silence (AI Winter)

**1969**
Perceptron criticized

**1995**
SVM reigns




**2006**
Restricted Boltzmann Machine



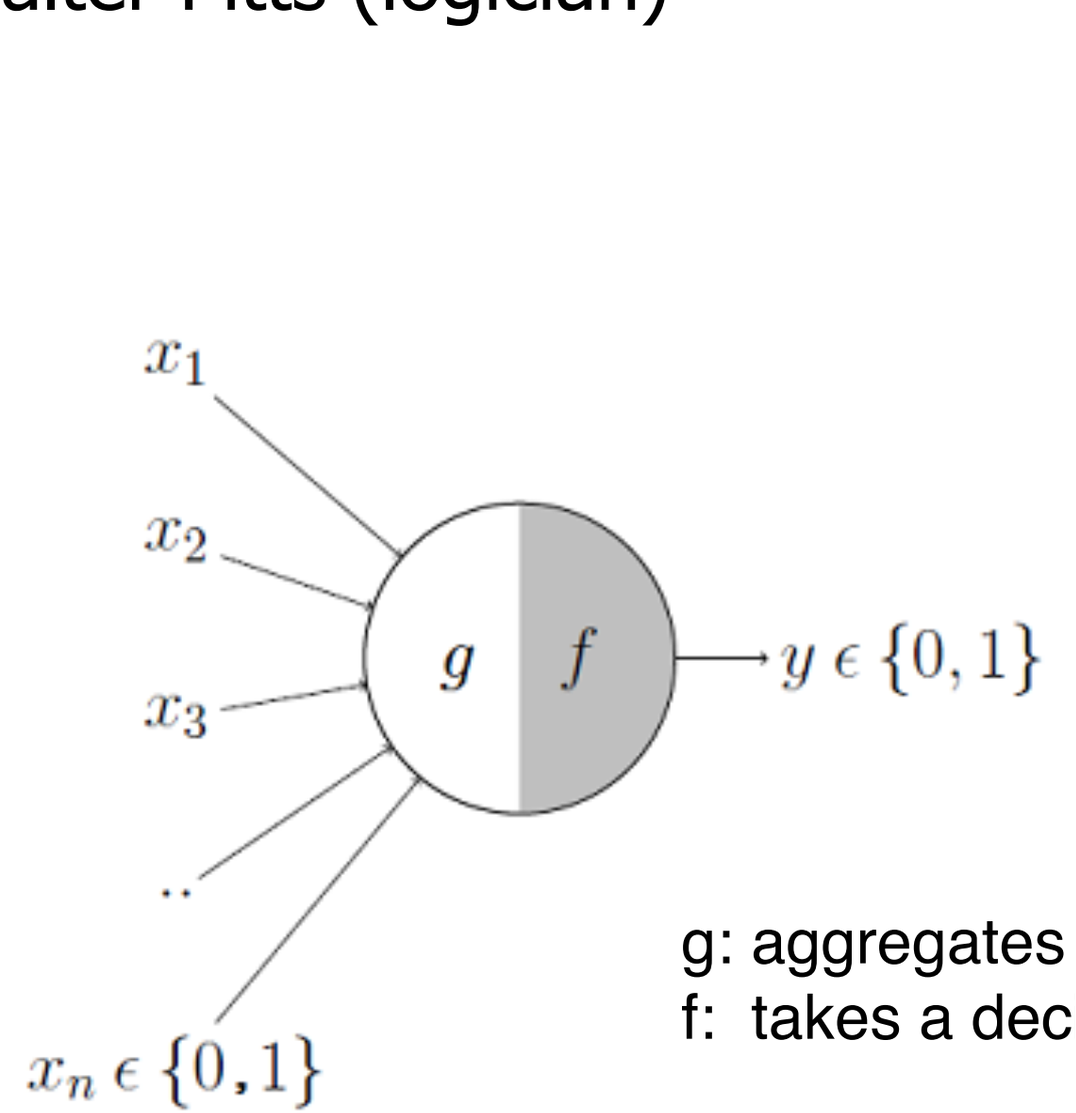
**2012**
AlexNet wins ImageNet

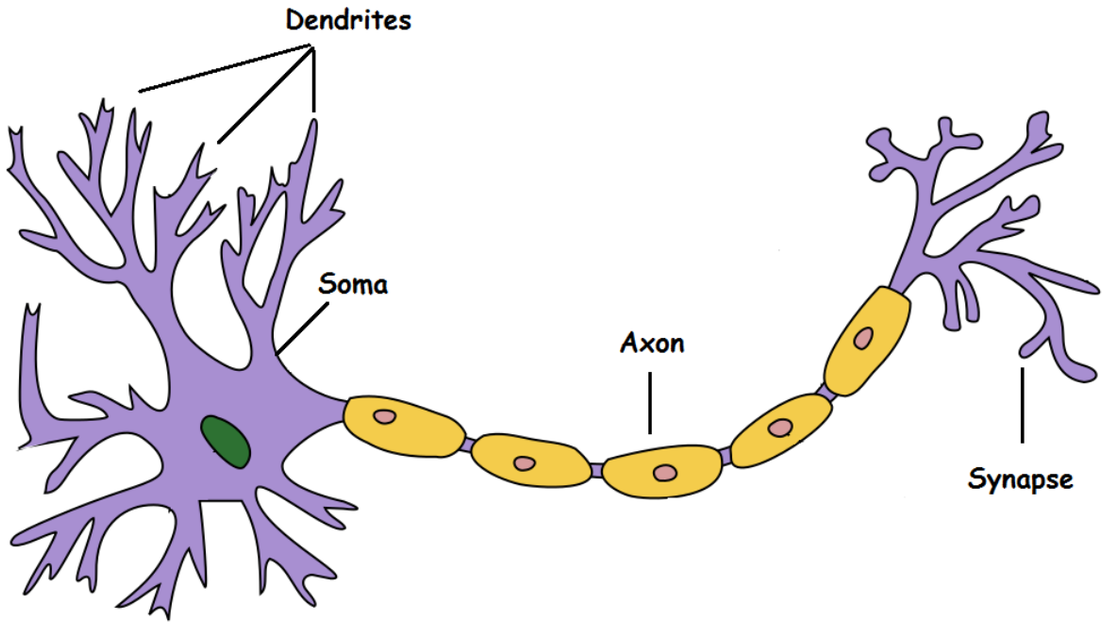
IMAGENET

Image from `https://www.slideshare.net/LuMa921/deep-learning-a-visual-introduction`.

# McCulloch-Pitts neuron (1943)

Warren McCulloch (neuroscientist)
Walter Pitts (logician)



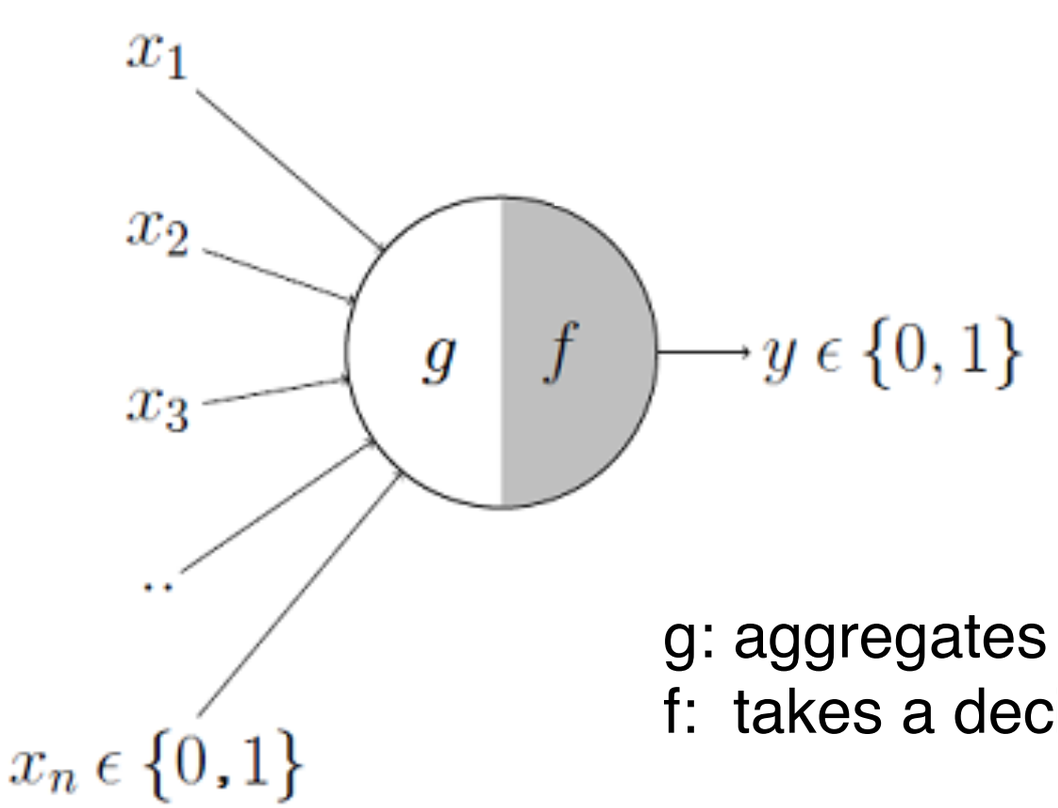

g: aggregates information
f: takes a decision

Images from
https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

# McCulloch-Pitts neuron (1943)

$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$y = f(g(\mathbf{x})) = 1 \quad if \quad g(\mathbf{x}) \geq \theta$$
$$= 0 \quad if \quad g(\mathbf{x}) < \theta$$




g: aggregates information
f: takes a decision

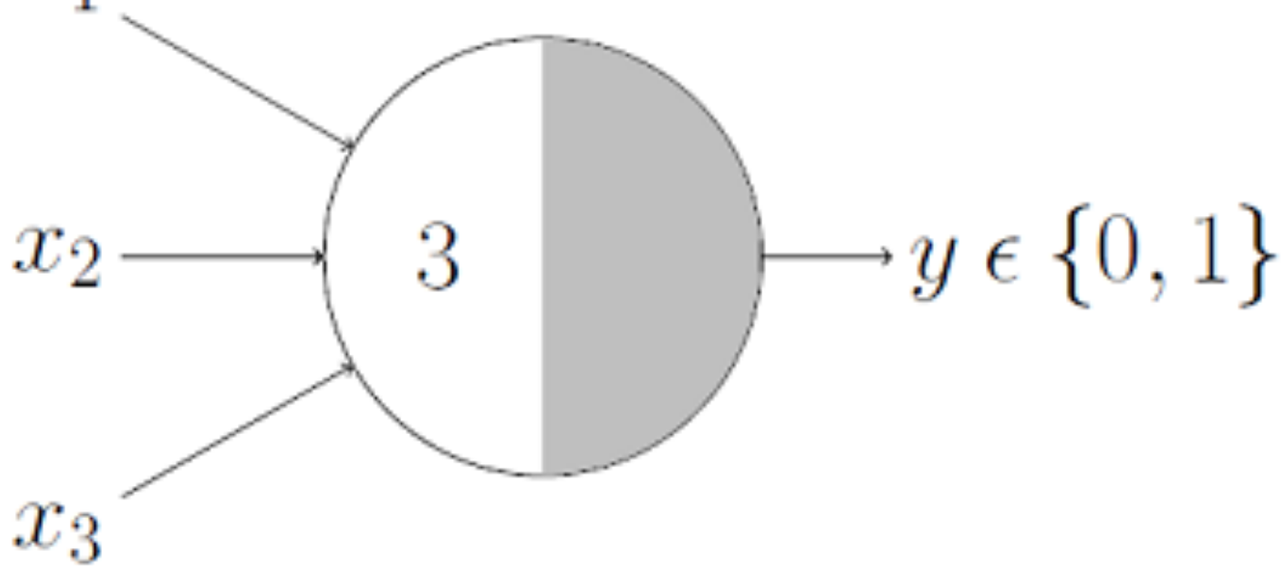
Images from
https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

# McCulloch-Pitts neuron (1943)

AND function

$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$f(g(\mathbf{x})) = 1 \quad if \quad g(\mathbf{x}) \geq \theta$$
$$= 0 \quad if \quad g(\mathbf{x}) < \theta$$




$$\theta = 3$$

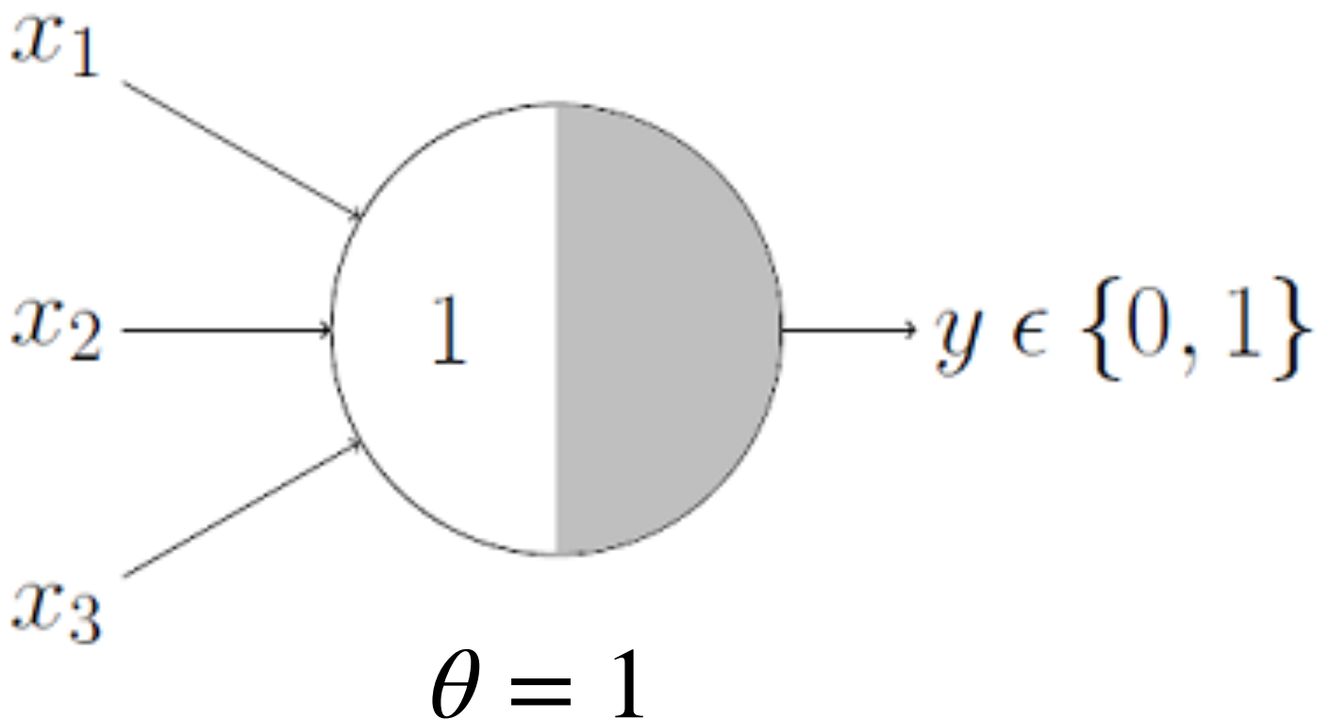
Images from
https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

# McCulloch-Pitts neuron (1943)

OR function

$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$\begin{aligned} f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \theta \\ &= 0 \quad if \quad g(\mathbf{x}) < \theta \end{aligned}$$




Images from
https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

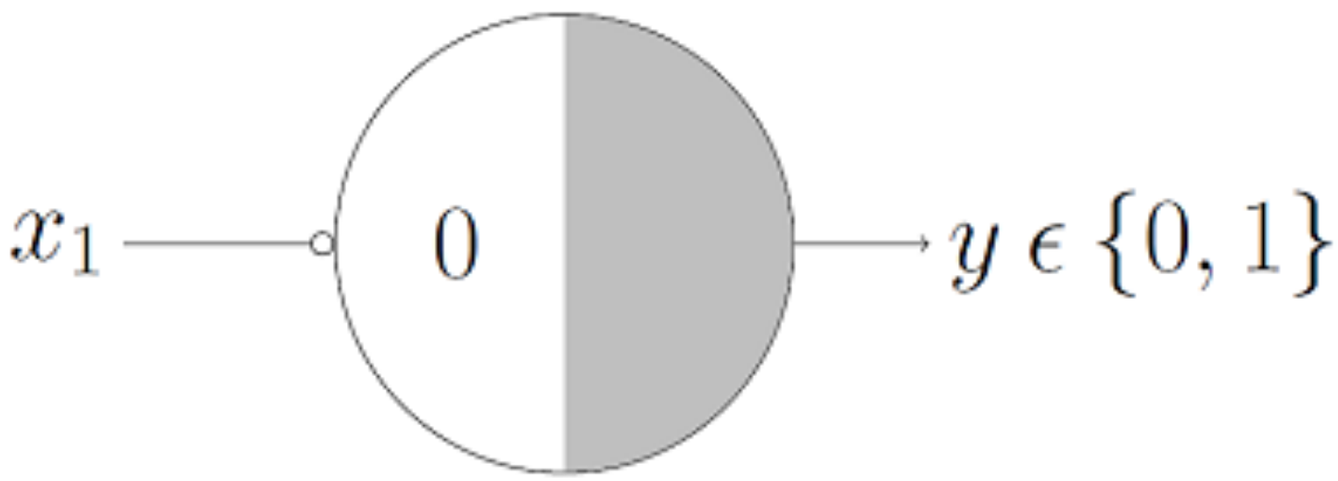
# McCulloch-Pitts neuron (1943)

Inhibitory inputs:
if present, output is zero

NOT example

$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \theta \\ &= 0 \quad if \quad g(\mathbf{x}) < \theta \\ &= 0 \quad if \quad \mathbf{x}_i > 0 \end{aligned}$$




Images from
https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

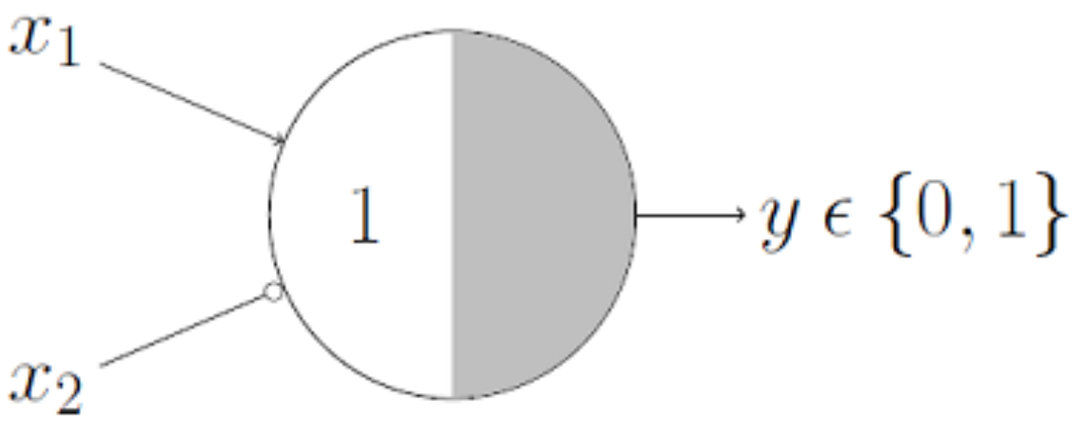
# McCulloch-Pitts neuron (1943)

Inhibitory inputs:
if present, output is zero

Other combinations

$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \theta \\ &= 0 \quad if \quad g(\mathbf{x}) < \theta \\ &= 0 \quad if \quad \mathbf{x}_i > 0 \end{aligned}$$

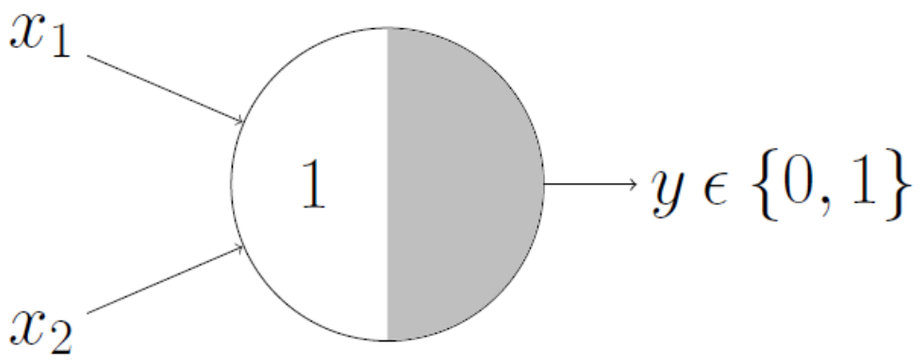



$x_1\ AND\ !x_2$*

Images from
https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

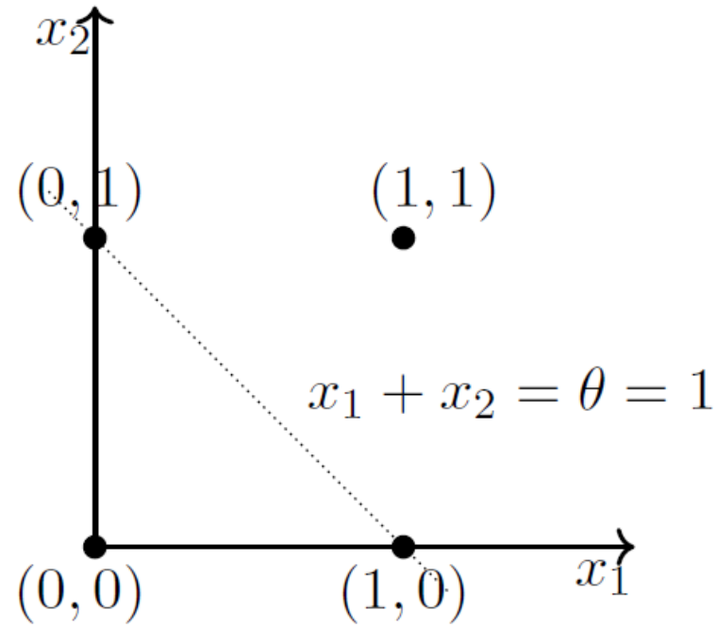
# McCulloch-Pitts neuron (1943)

Geometric interpretation




*OR function*

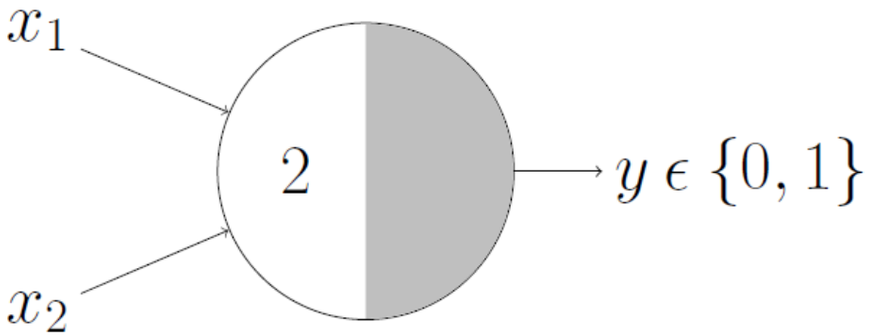
$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$




Images from
https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

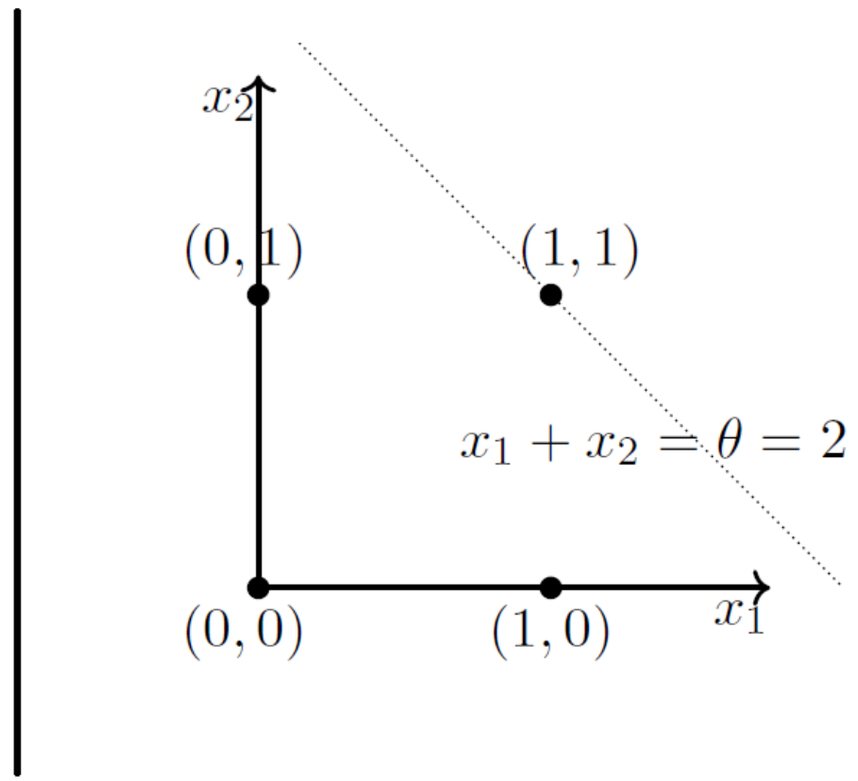
# McCulloch-Pitts neuron (1943)

Geometric interpretation




*AND function*

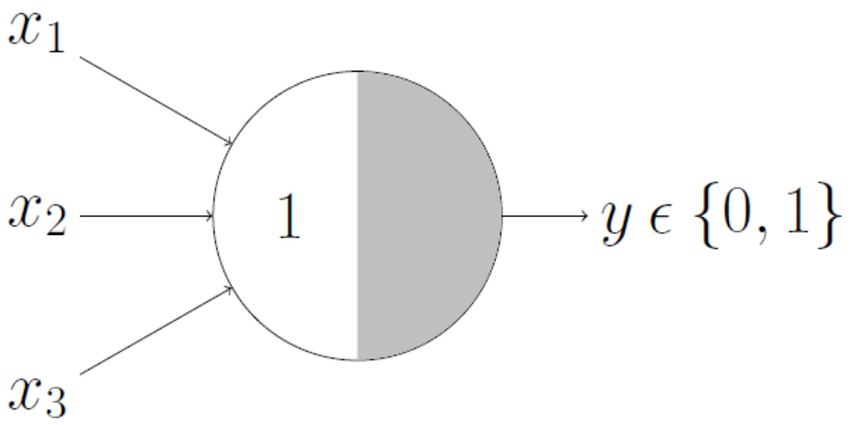
$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$$




Images from
https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

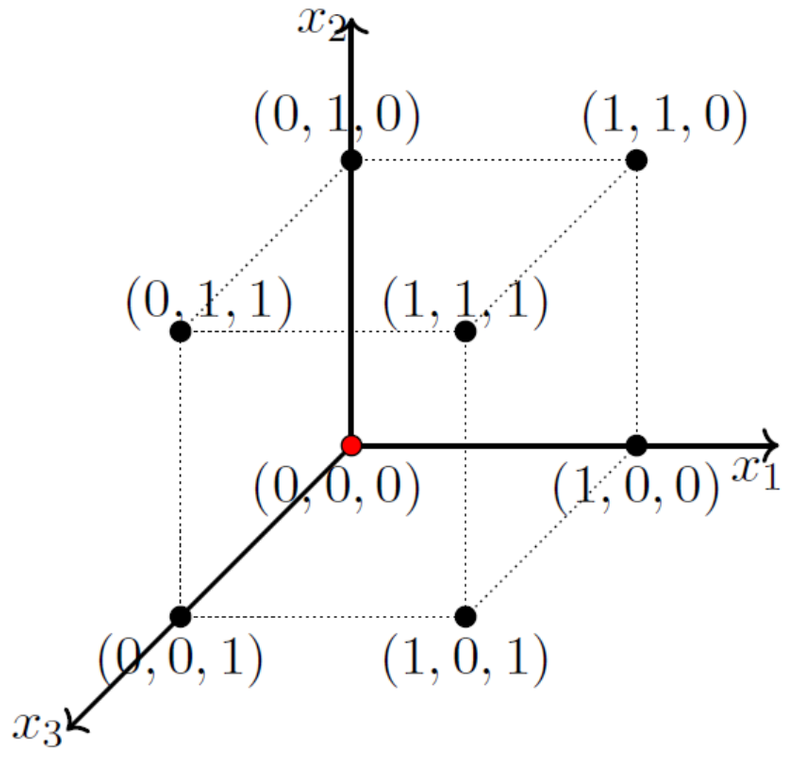
# McCulloch-Pitts neuron (1943)

Geometric interpretation




*OR function*

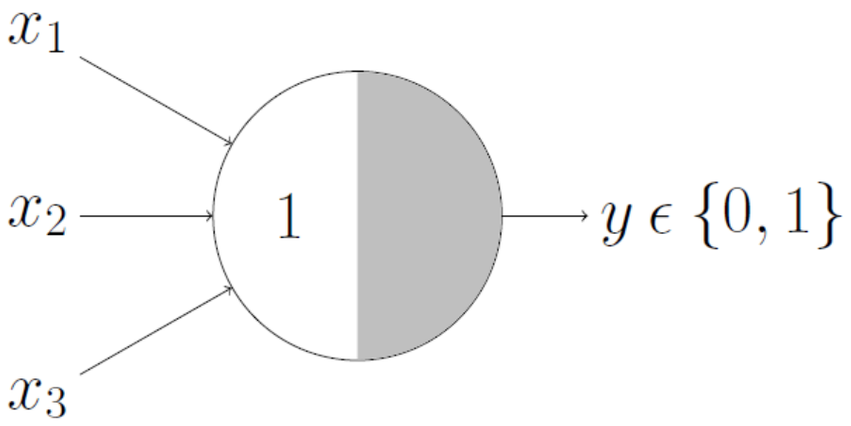
$$x_1 + x_2 + x_3 = \sum_{i=1}^{3} x_i \geq 1$$




Images from
https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

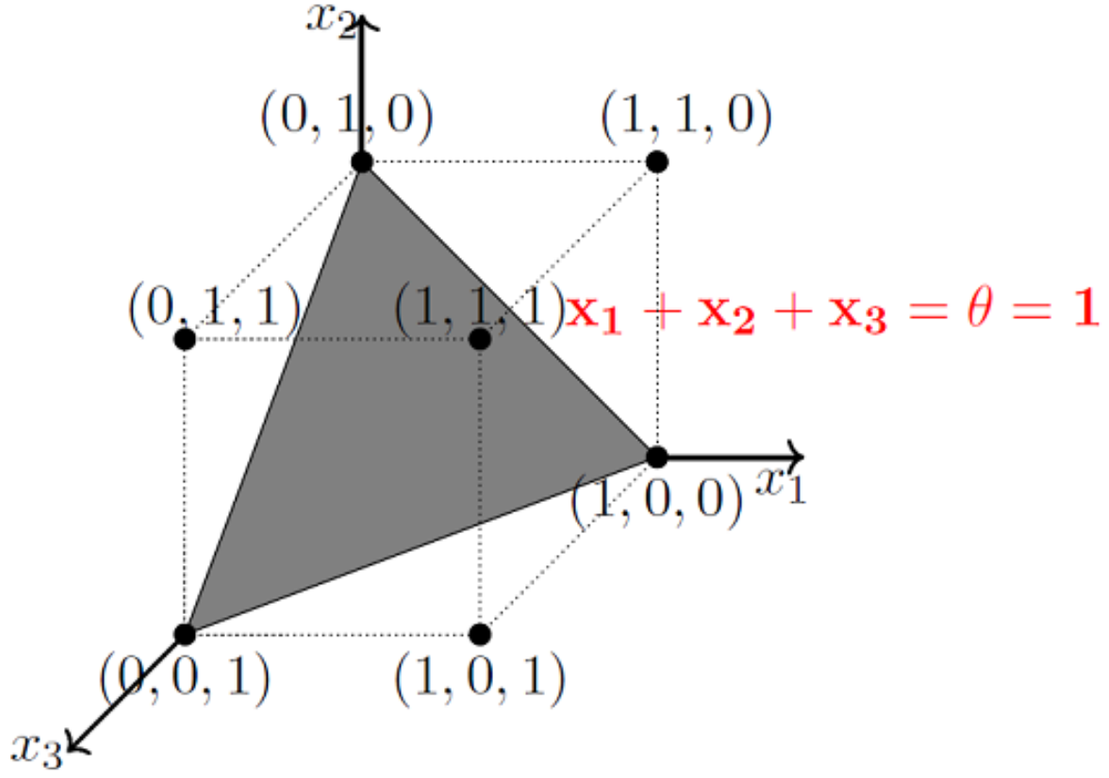
# McCulloch-Pitts neuron (1943)

Geometric interpretation




*OR function*

$$x_1 + x_2 + x_3 = \sum_{i=1}^{3} x_i \geq 1$$




Images from
https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

# Frank Rosenblatt Perceptron (1956)

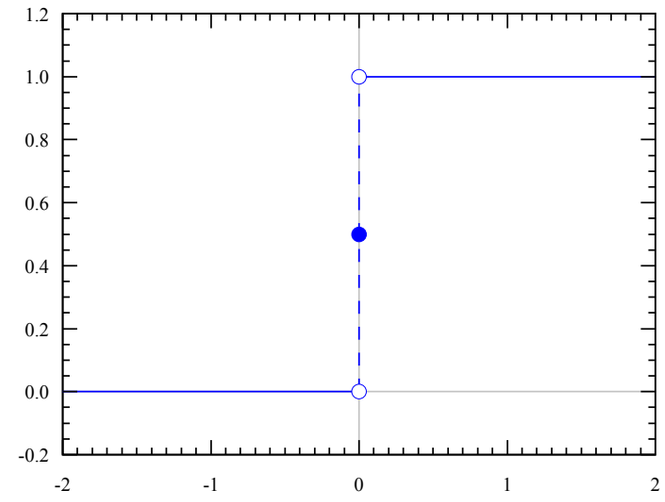
- Use data to learn the coefficients

$$f(g(\mathbf{x})) = y \qquad f(g(x)) = \begin{cases} 1 & \text{if } \mathbf{w} \,.\, \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Observations - training set $D$

$f$ : activation function

$$\begin{cases} \mathbf{x} \in \mathbb{R}^N & \text{input} \\ y \in \mathbb{B} & \text{output binary} \end{cases}$$




By Omegatron - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=801382

- Learnable parameters

$$\begin{cases} \mathbf{w} \in \mathbb{R}^N & \text{weights} \\ b \in \mathbb{R} & \text{bias} \end{cases}$$

# Frank Rosenblatt Perceptron (1956)

Optimization

- Initialize weights $\mathbf{w}$ to zero (or small random)

- For each sample $j$ in the training set $D = \{\mathbf{x_j}, d_j\}$:
  - Compute prediction

$$y_j(t) = f(\mathbf{w}(t) \,.\, \mathbf{x}_j)$$
$$= f(w_0(t)\ x_{j,0} + w_1(t)\ x_{j,1} + w_2(t)\ x_{j,2} + \ldots + w_{N-1}(t)\ x_{j,N-1})$$

- Update all weights $0 \leq i \leq N-1$

$$w_i(t+1) = w_i(t) + r \cdot (d_j - y_j(t))\ x_{j,i}$$

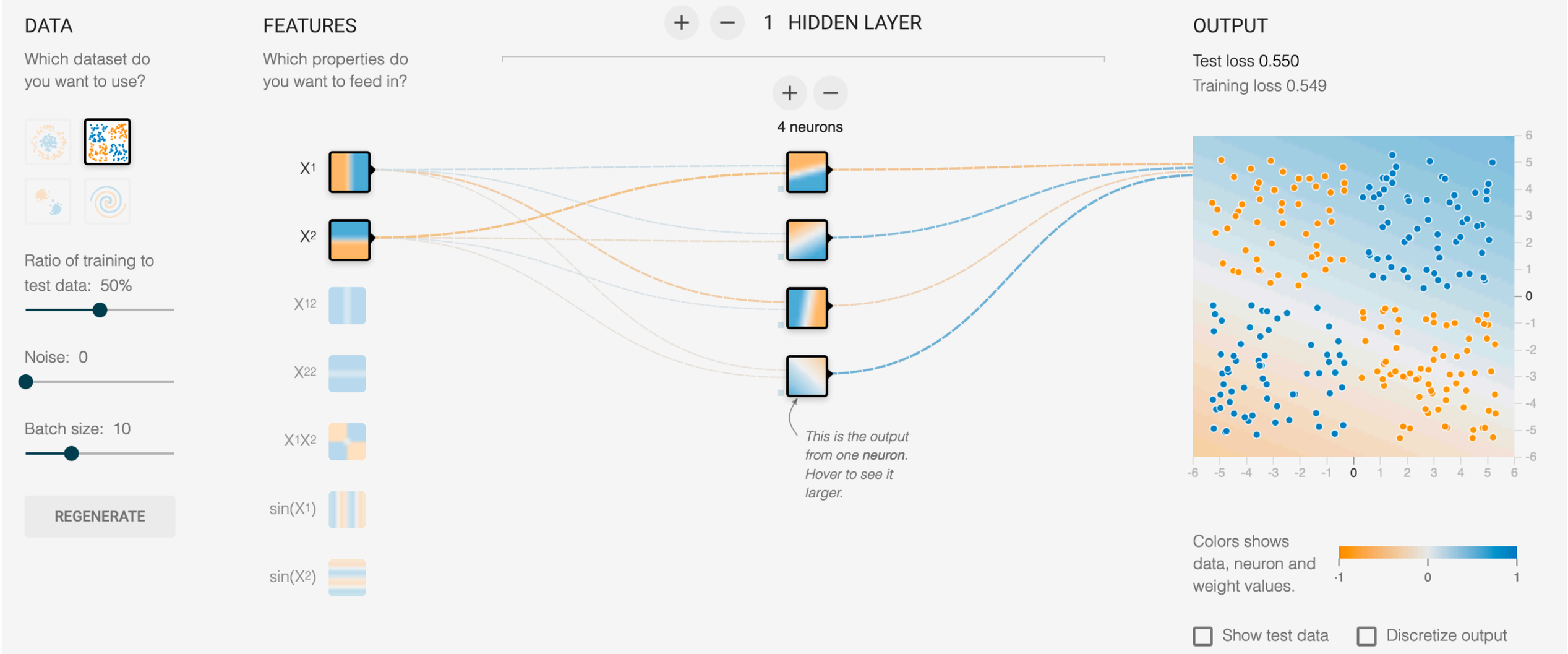
$r$ learning rate (usually small)

# Frank Rosenblatt Perceptron (1956)

Limitations:

- The perceptron learning algorithm does not terminate if the learning set is not linearly separable
  - Needs a limit on the number of iterations!
- Example: exclusive OR (XOR)
  - Algorithm fails completely! (no approx. solution)

# Frank Rosenblatt Perceptron (1956)

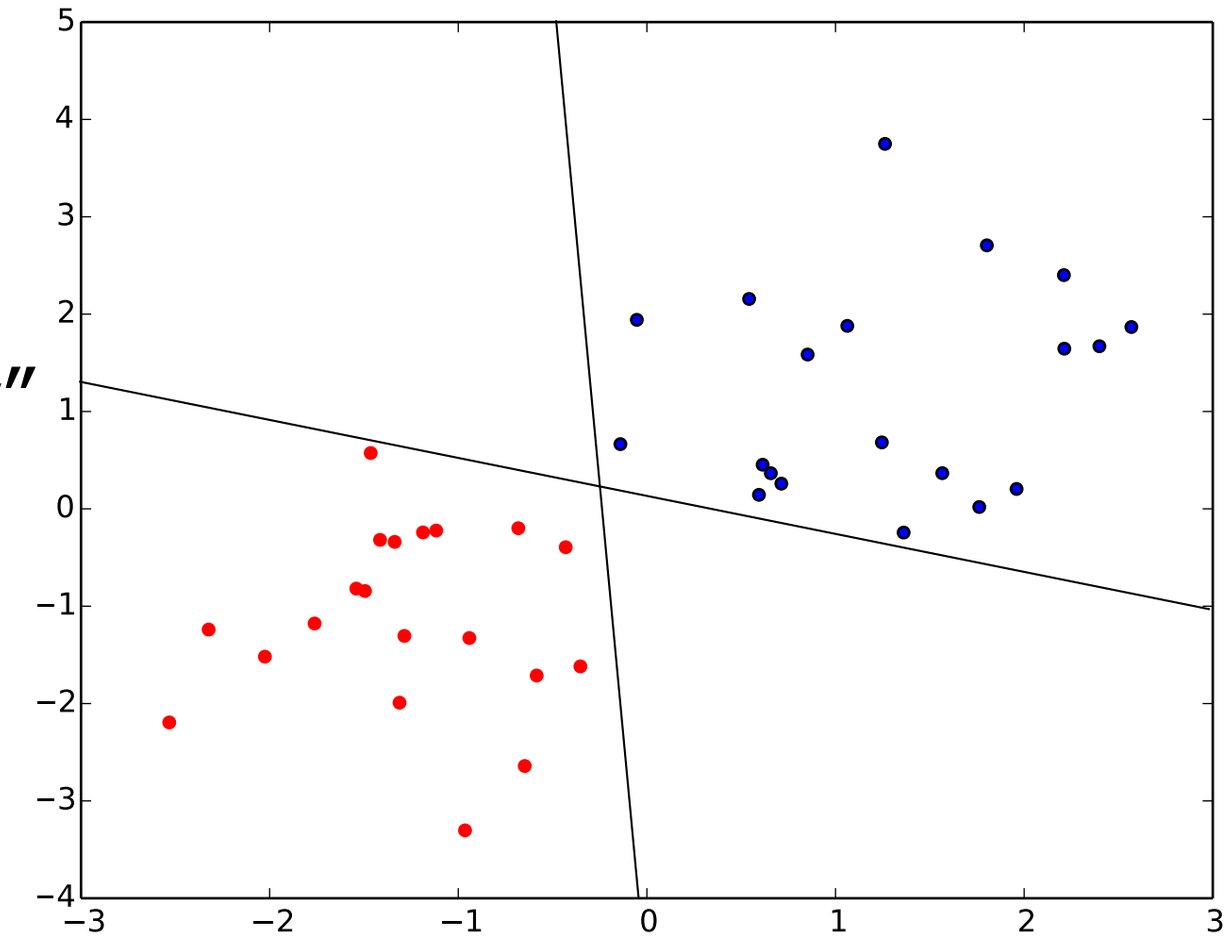



https://playground.tensorflow.org

# Frank Rosenblatt Perceptron (1956)

Limitations:

- Perceptron can’t choose “The best solution”
- Motivation for the linear SVM “perceptron of optimal stability” (Krauth and Mezard, 1987)




By Qwertyus - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=41351528

# Frank Rosenblatt Perceptron (1956)

- Controversy

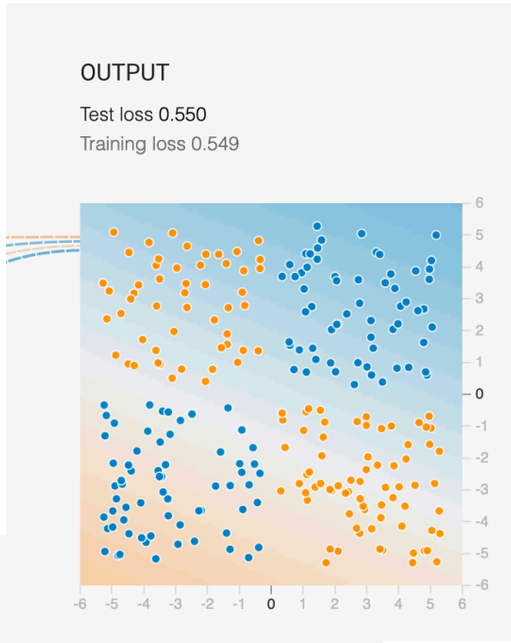
> The perceptron in … "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

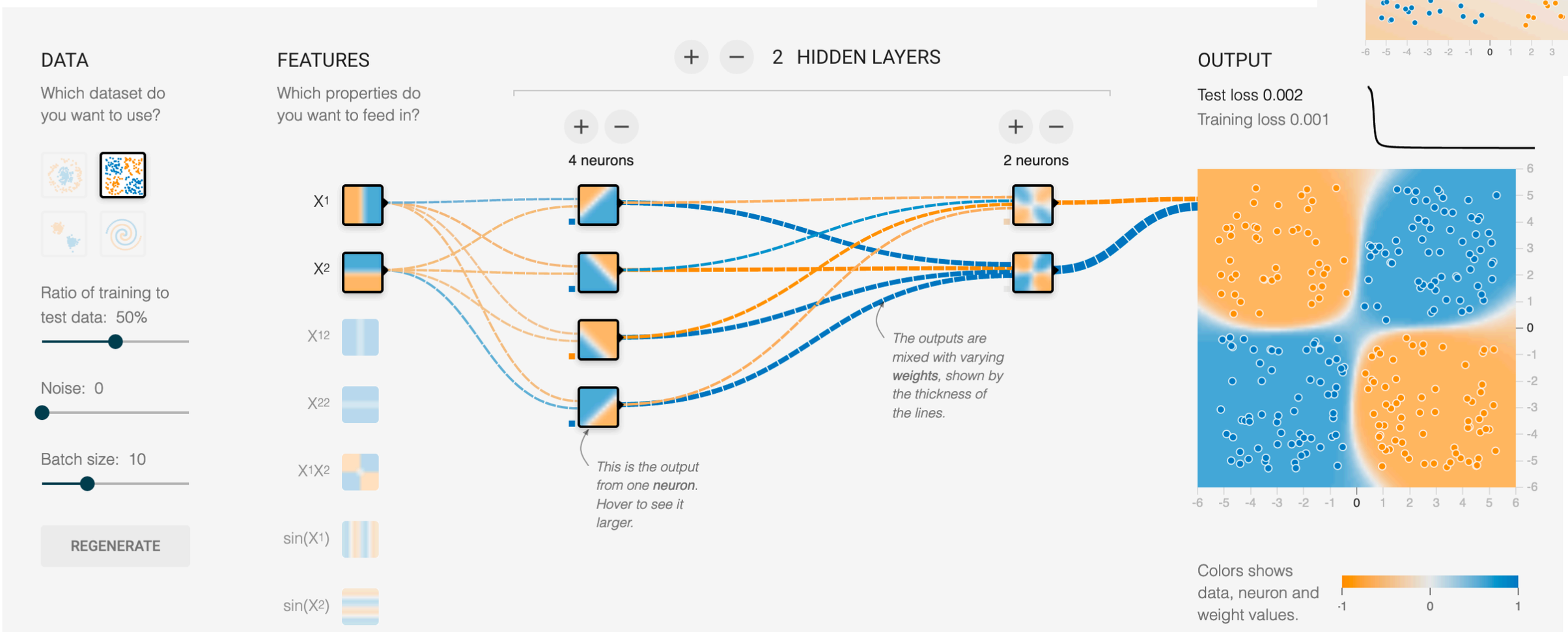
The New York Times, 1958

*Olazaran, Mikel (1996). "A Sociological Study of the Official History of the Perceptrons Controversy". Social Studies of Science. 26 (3): 611–659*

# Multi-Layer Perceptron

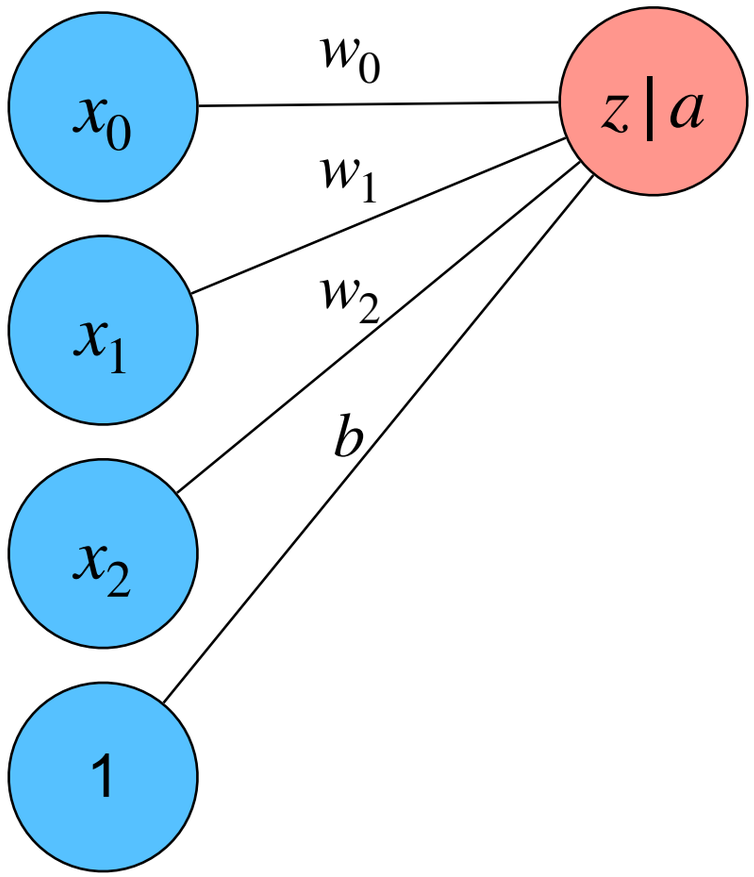
Two layers can handle the “non linear separable set” limitation




https://playground.tensorflow.org

# Multi-Layer Perceptron



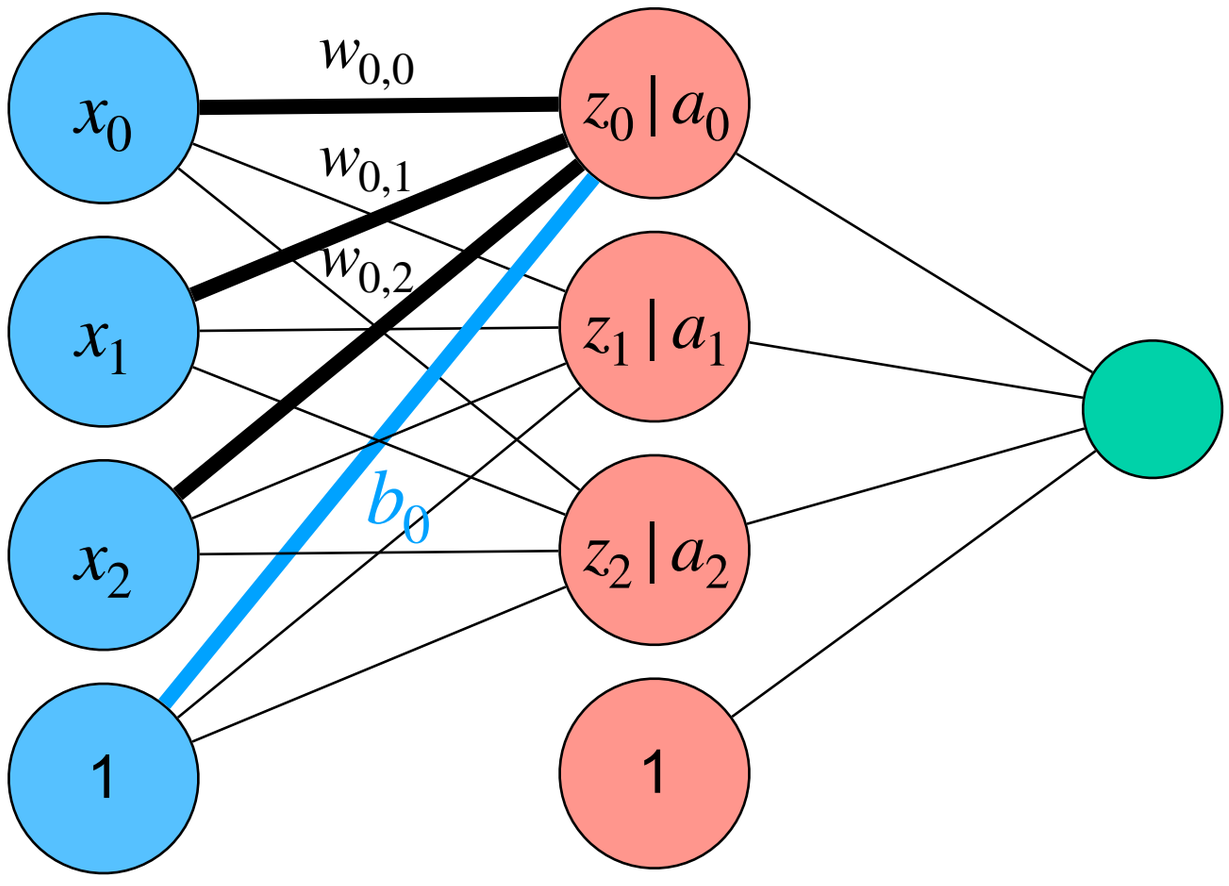

$$z = \mathbf{w} \,.\, \mathbf{x} + b$$

$$a = f(\mathbf{w} \,.\, \mathbf{x} + b)$$

$$= f(w_0\ x_0 + w_1\ x_1 + w_2\ x_2 + \ldots + w_{N-1}\ x_{N-1} + b)$$

# Multi-Layer Perceptron

$$\mathbf{w_0} = (w_{0,0}, w_{0,1}, w_{0,2})$$

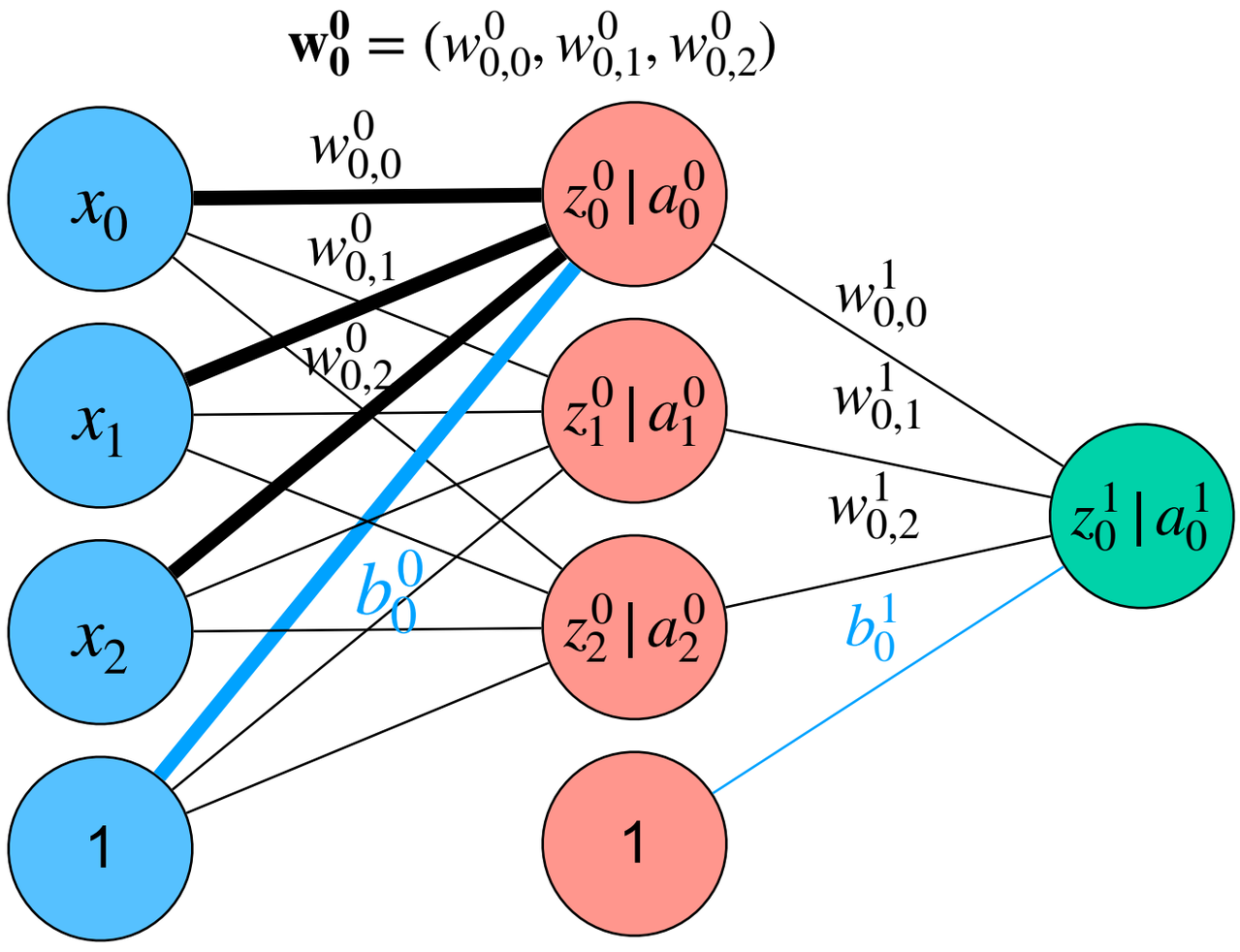



$$z_i = \mathbf{w_i} \cdot \mathbf{x} + b_i$$

$$a_i = f(\mathbf{w_i} \cdot \mathbf{x} + b_i)$$

$$= f(w_{i,0}\ x_0 + w_{i,1}\ x_1 + w_{i,2}\ x_2 + \ldots + w_{i,N-1}\ x_{N-1} + b_i)$$

# Multi-Layer Perceptron

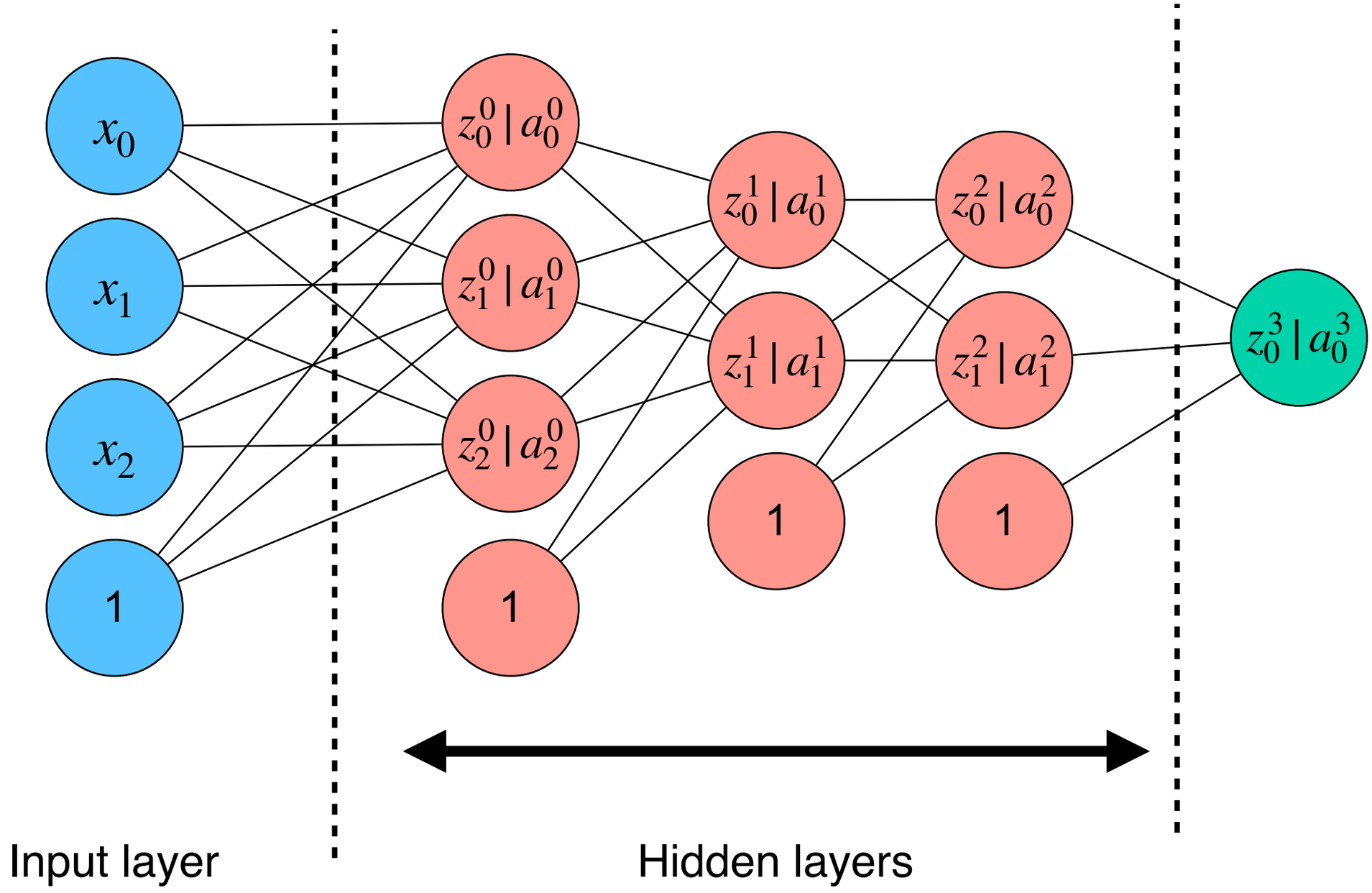



$$z_i^l = \mathbf{w_i^l} . \mathbf{x} + b_i^l$$

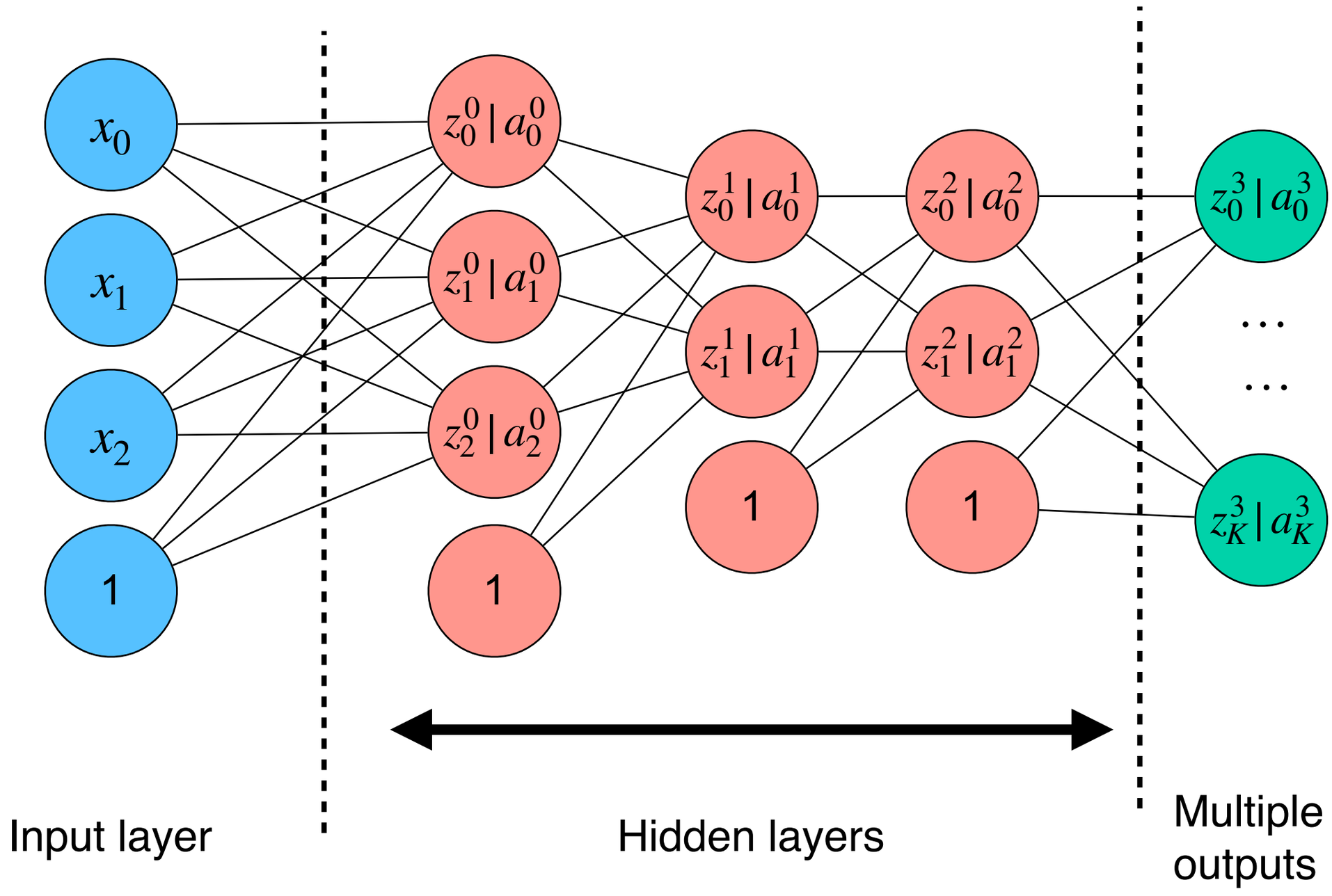
$$a_i^l = f(\mathbf{w_i^l} . \mathbf{x} + b_i^l)$$

$$= f(w_{i,0}^l \; x_0 + w_{i,1}^l \; x_1 + w_{i,2}^l \; x_2 + \ldots + w_{i,N-1}^l \; x_{N-1} + b_i^l)$$

# Multi-Layer Perceptron

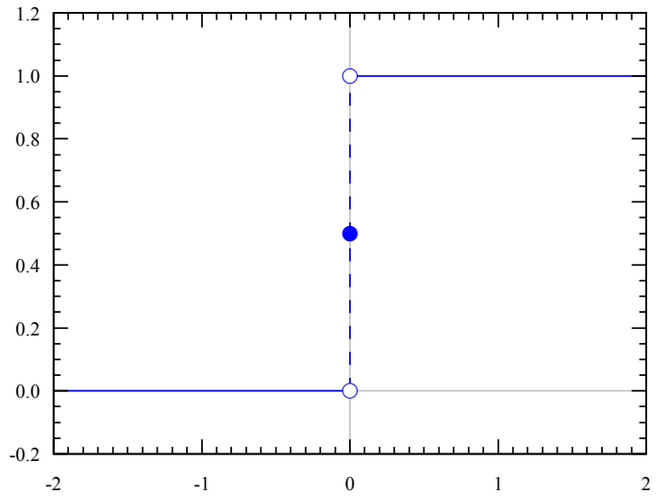


# Multi-Layer Perceptron



# Activation functions

- Play a major role
- Original perceptron: heavy side step-function
- Many variants:
  - Linear
  - Logistic
  - tanh
  - ReLU - rectified linear unit
  - GeLU - Gaussian error linear unit
  - …



# Activation functions

- Limitations of linear functions:
  - A N-layer network can be written as a 2-layer network

# Linear activations

Consider a network with a single hidden layer and two activation functions $\zeta$ and $\sigma$:

$$y_k(x) = \sigma\left(\sum_j w_{kj}^{(2)} \zeta\left(\sum_i w_{ji}^{(1)} x_i\right)\right)$$

If the activation functions are linear, then we can rewrite the equations in the following vectorial notation:

$$\begin{aligned} y &= (\sigma.W^{(2)})(\zeta.W^{(1)}x) \\ &= W^{(2)'}(W^{(1)'}x) \\ &= W^{''}x \end{aligned}$$

Linear activation functions result in linear models!
$\rightarrow$ The activation functions between layers should be non-linear.

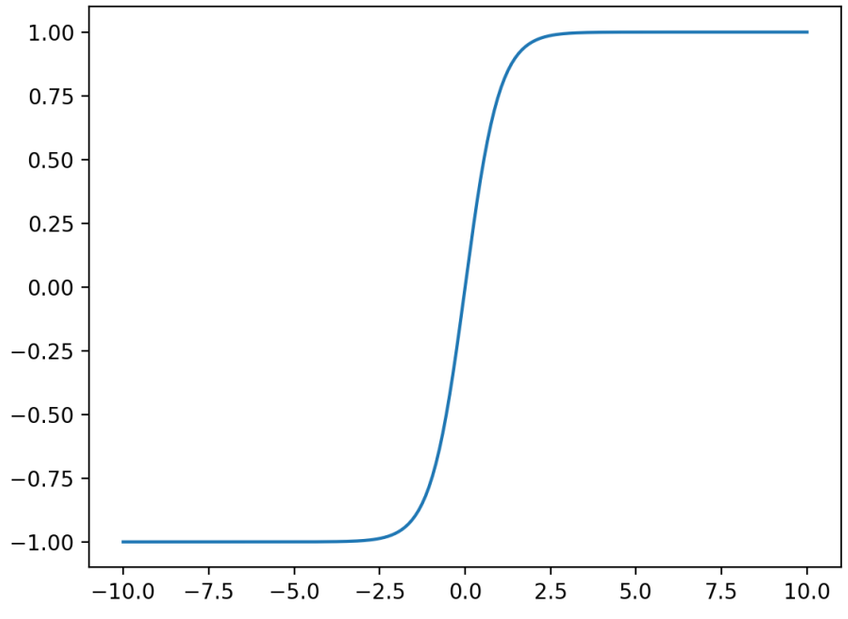
Original Slide from Christian Wolf
https://perso.liris.cnrs.fr/christian.wolf/teaching/

# Activation functions

Examples

Hyperbolic tangent

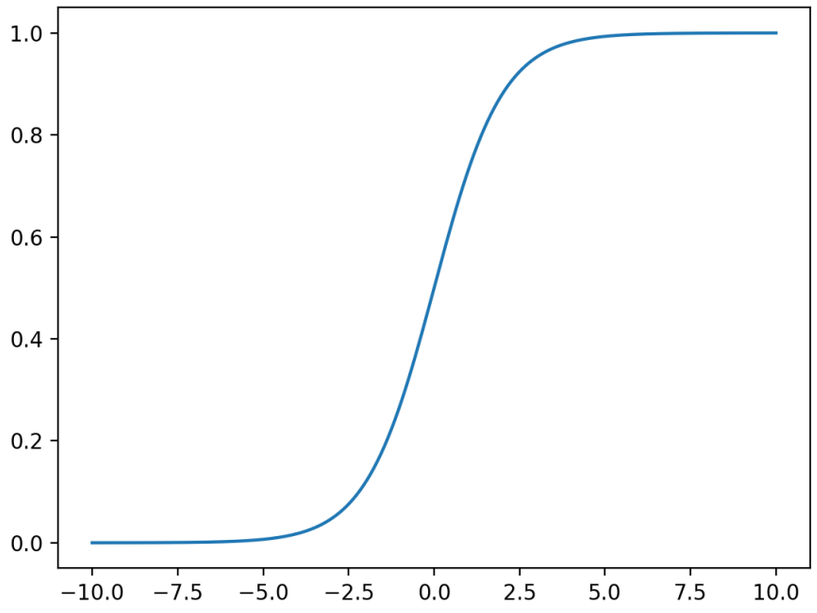
$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1 = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$




Logistic function

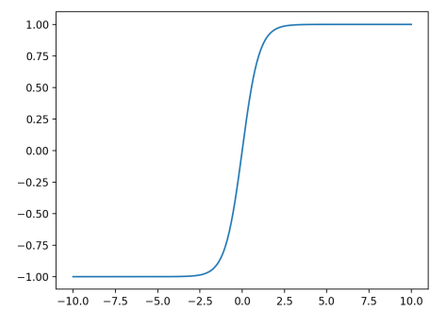
$$\sigma(x) = \frac{1}{1 + e^{-kx}}, \text{ with } k = 1$$

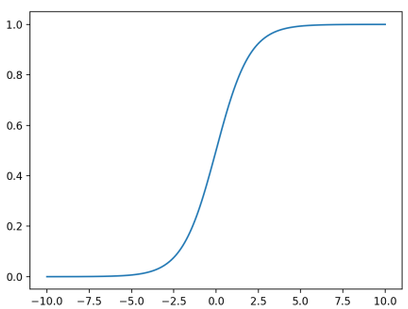
$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^{x}}{e^{x} + 1}$$



# Activation functions

Limitations of “sigmoid” functions: tanh and logistic




… the hyperbolic tangent activation function typically performs better than the logistic sigmoid. - Page 195

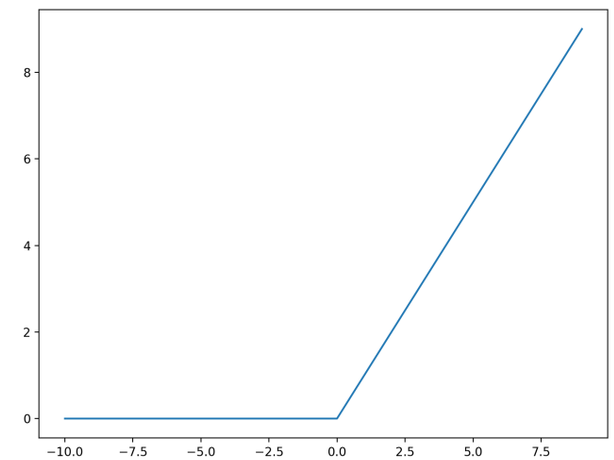
… sigmoidal units saturate across most of their domain—they saturate to a high value when z is very positive, saturate to a low value when z is very negative, and are only strongly sensitive to their input when z is near 0. - Page 195

Vanishing gradients make it difficult to know which direction the parameters should move to improve the cost function - Page 290

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

# Activation functions

$$ReLU(x) = \max(0,x)$$




- Motivation of ReLU
  - Computational Simplicity
  - Representation Sparsity - can output true zero values
  - Linear Behavior

… [another] major algorithmic change that has greatly improved the performance of feedforward networks was the replacement of sigmoid hidden units with piecewise linear hidden units, such as rectified linear units. - Page 226

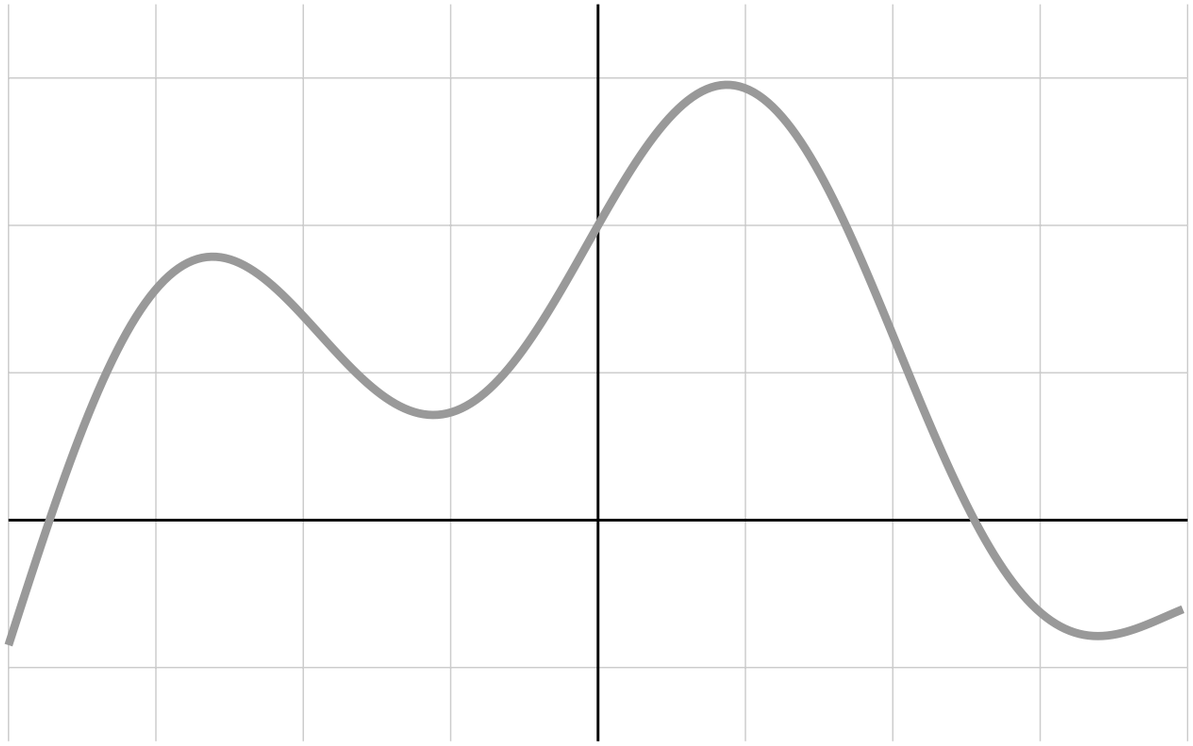
Computations are also cheaper: there is no need for computing the exponential function in activations. - Page 226

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

# Universal approximation

A continuous function can be approximated with a linear combination of translated/scaled ReLU functions

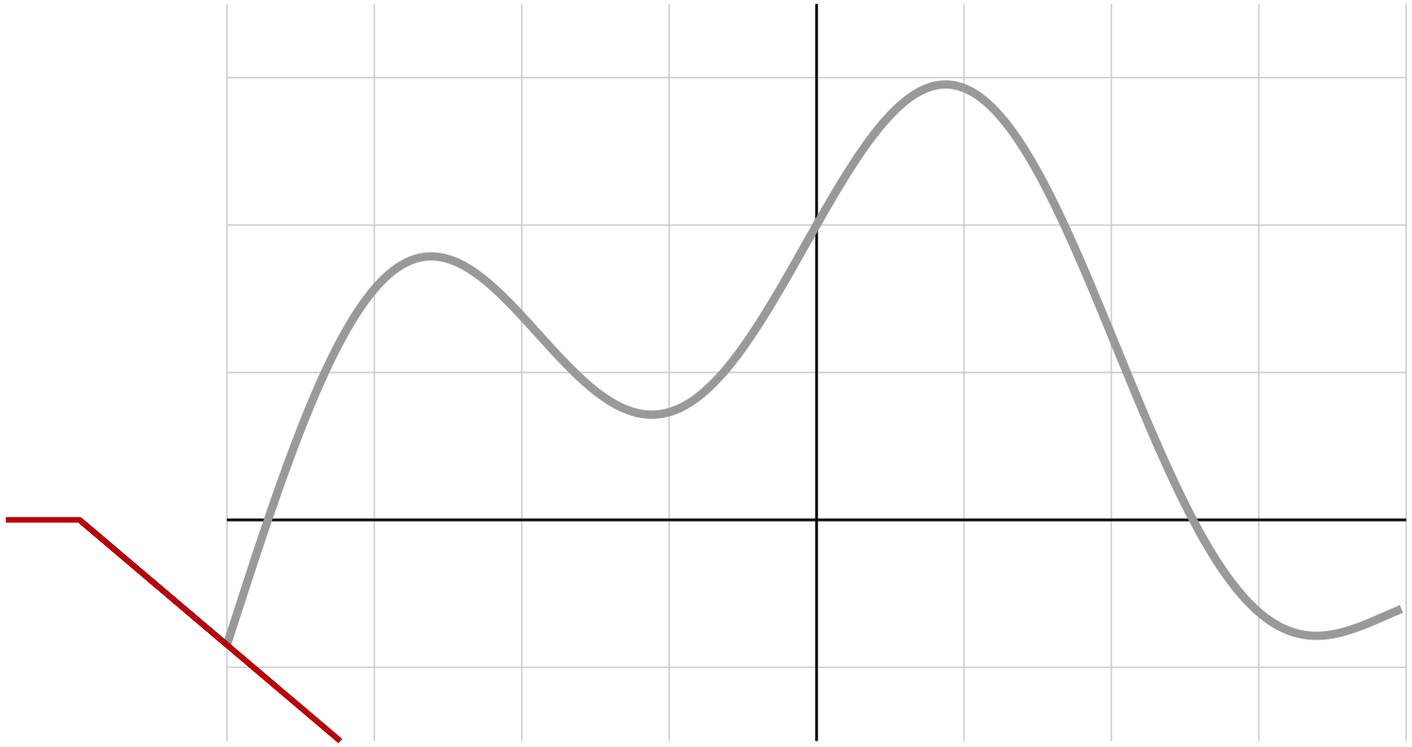
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

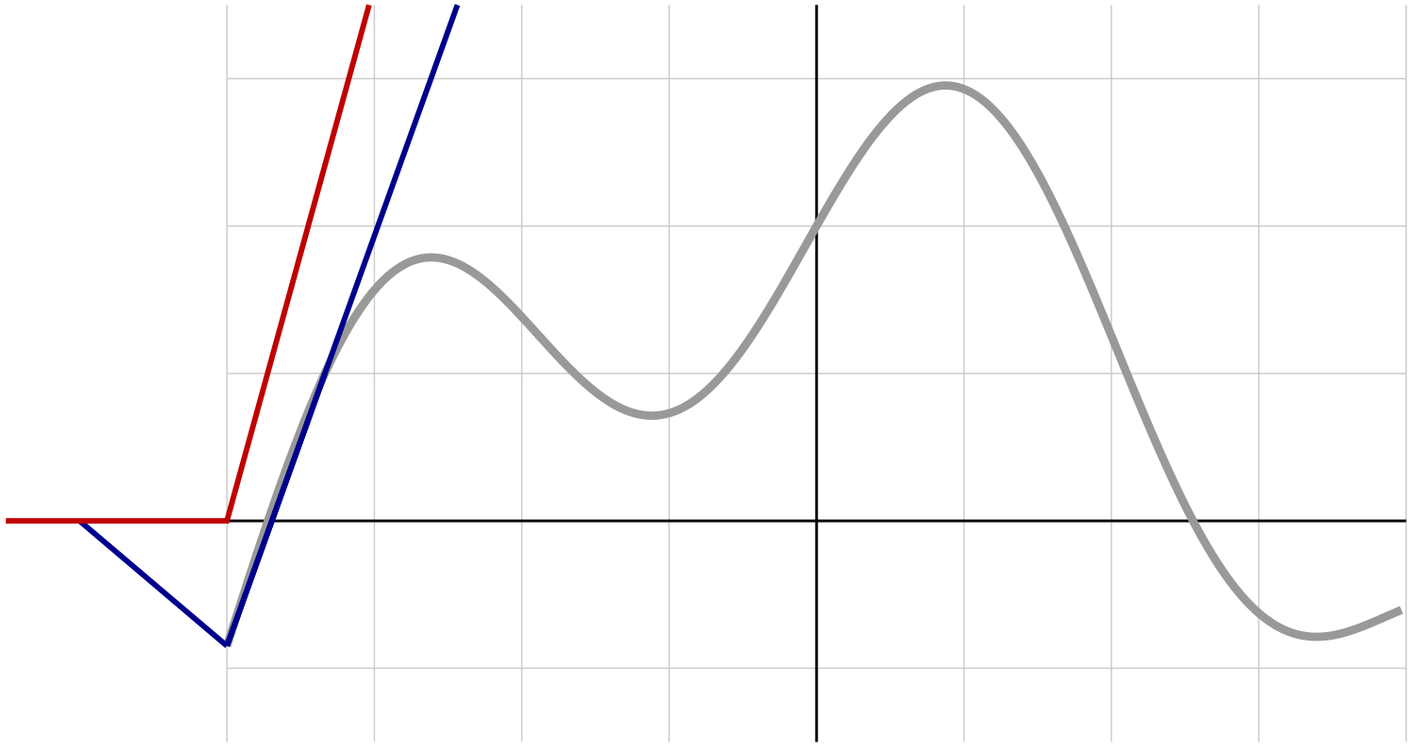
$$f(x) = \sigma(w_1 x + b_1)$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

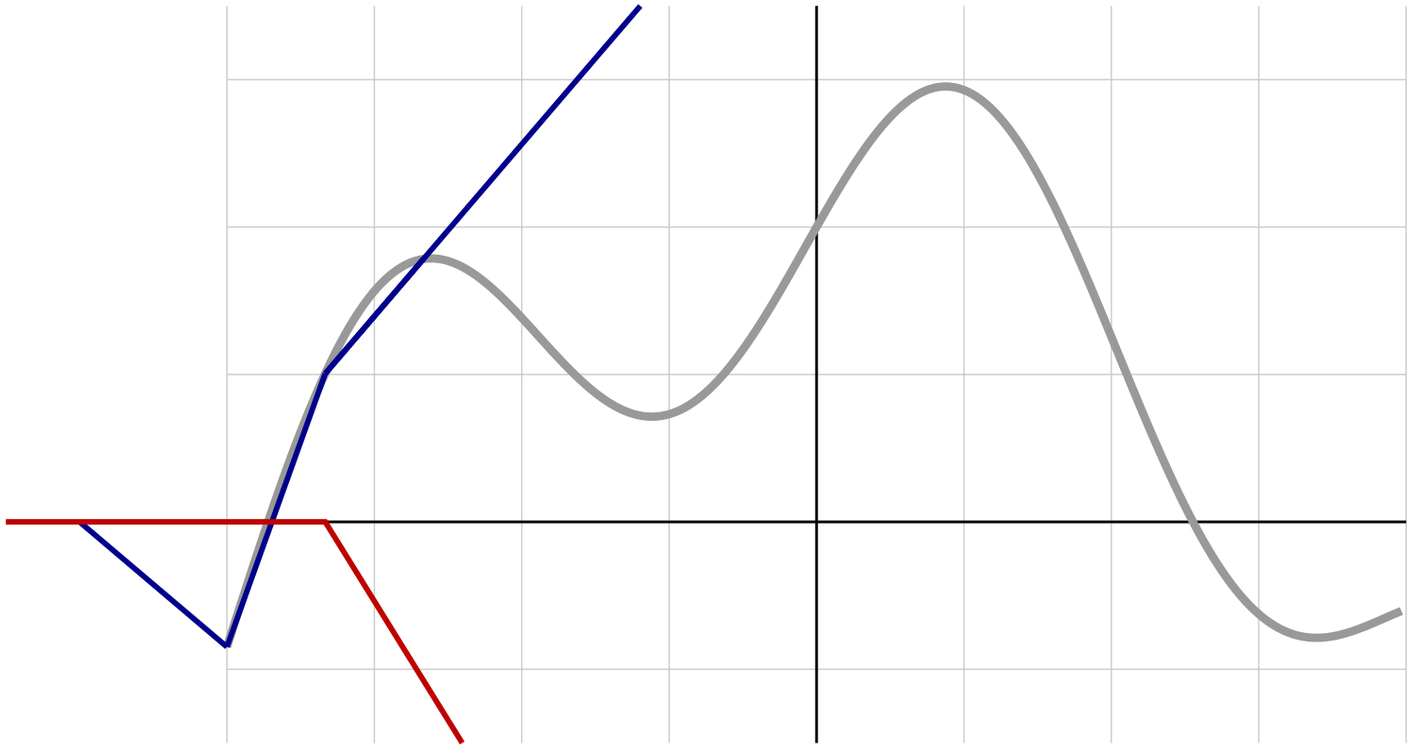
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2)$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

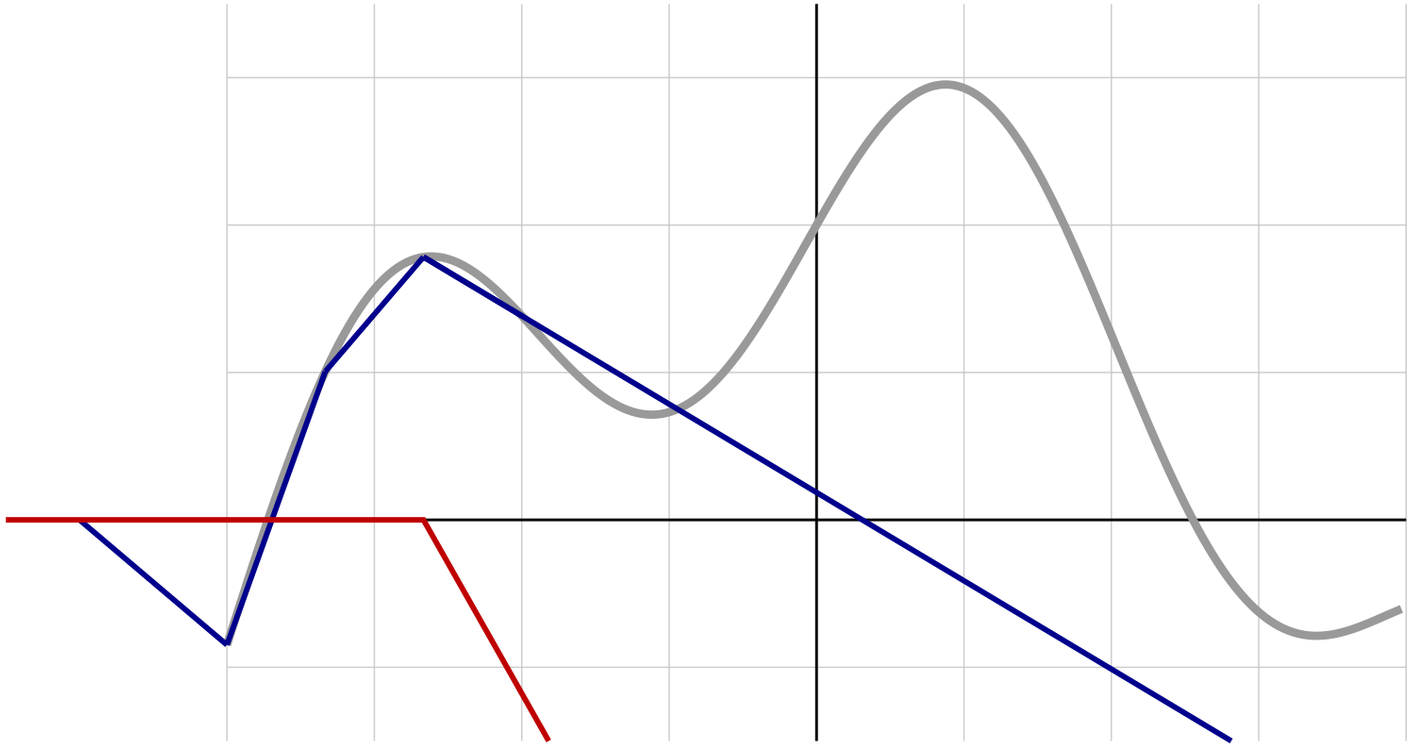
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3)$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

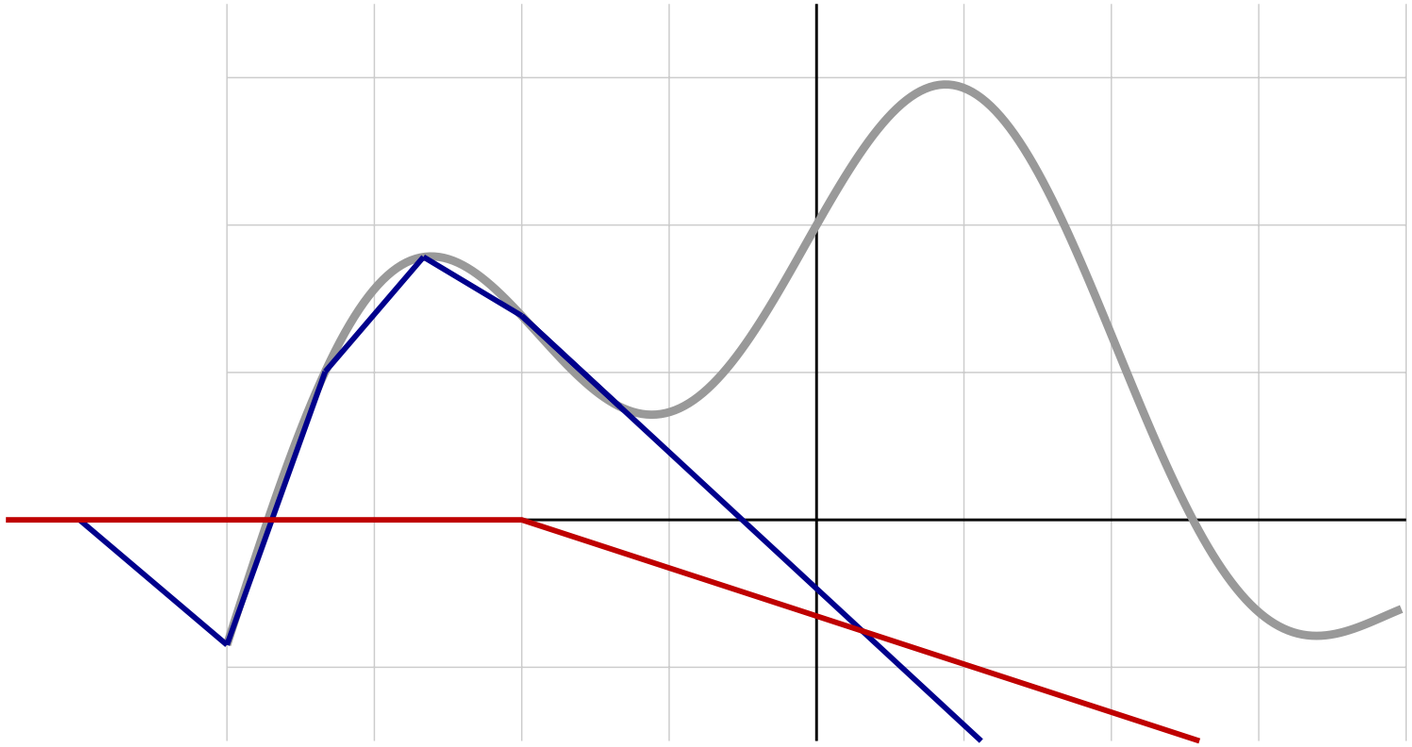
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

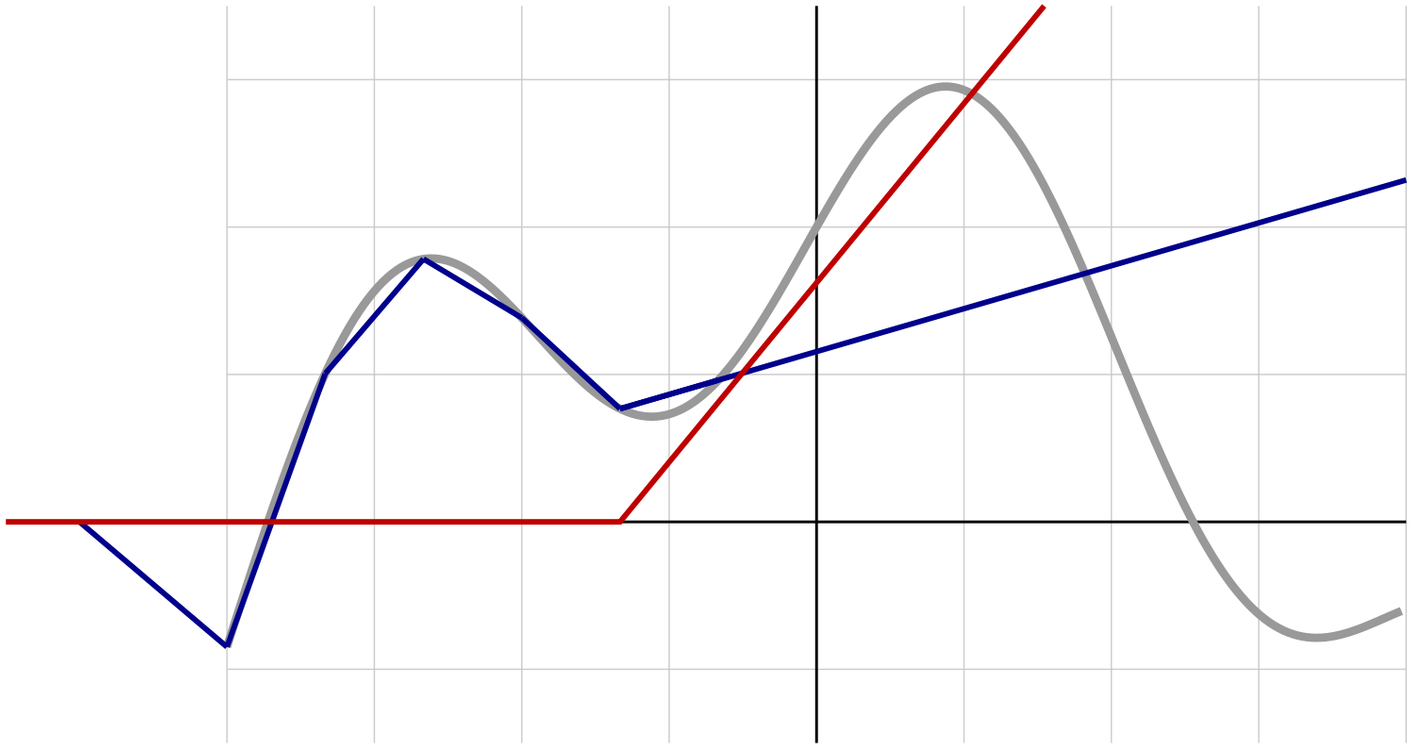
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

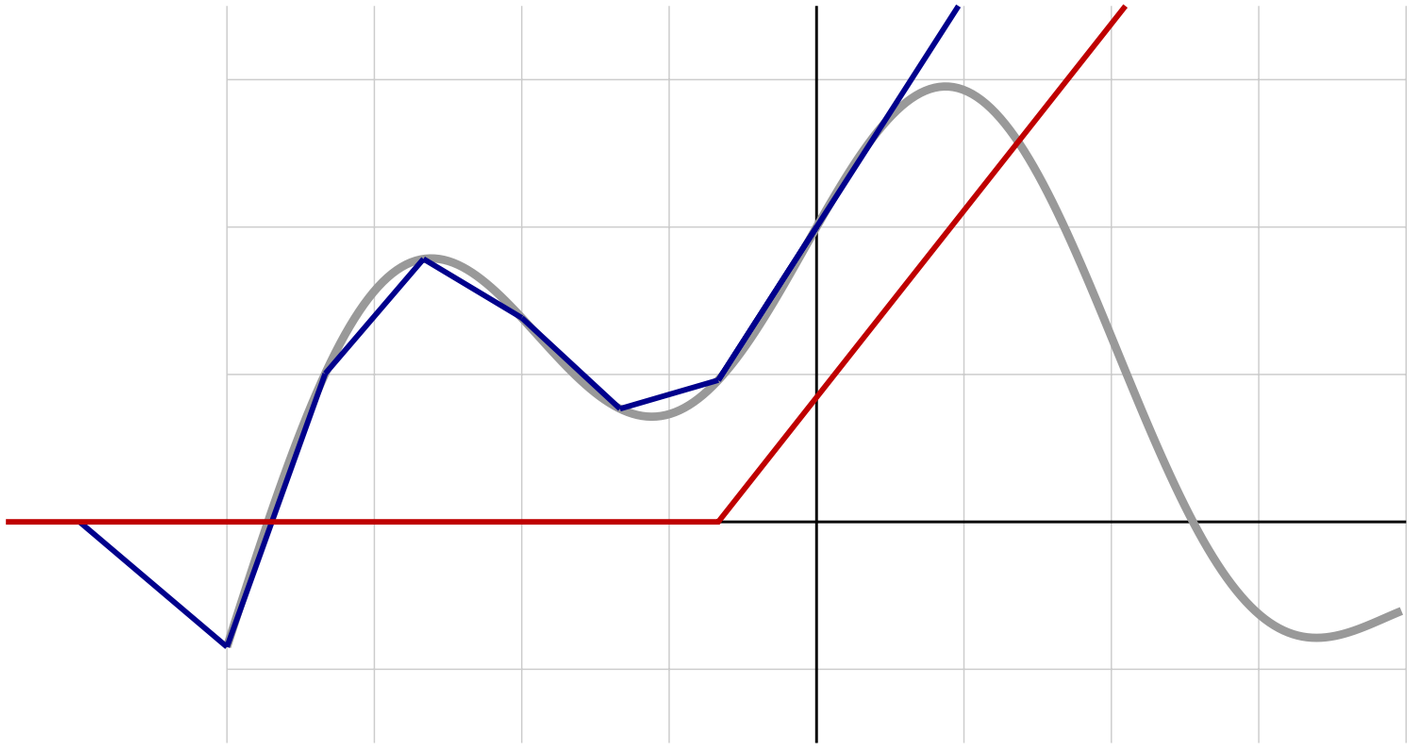
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

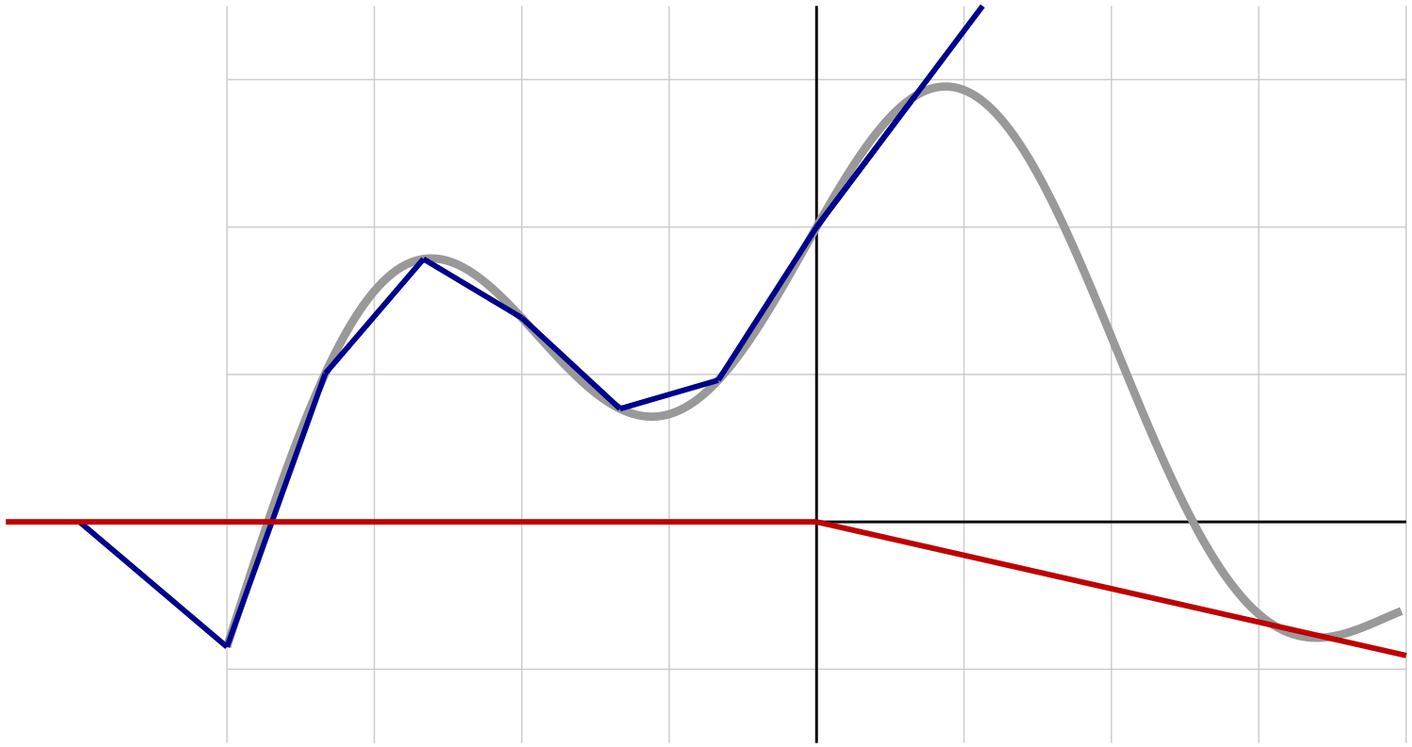
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

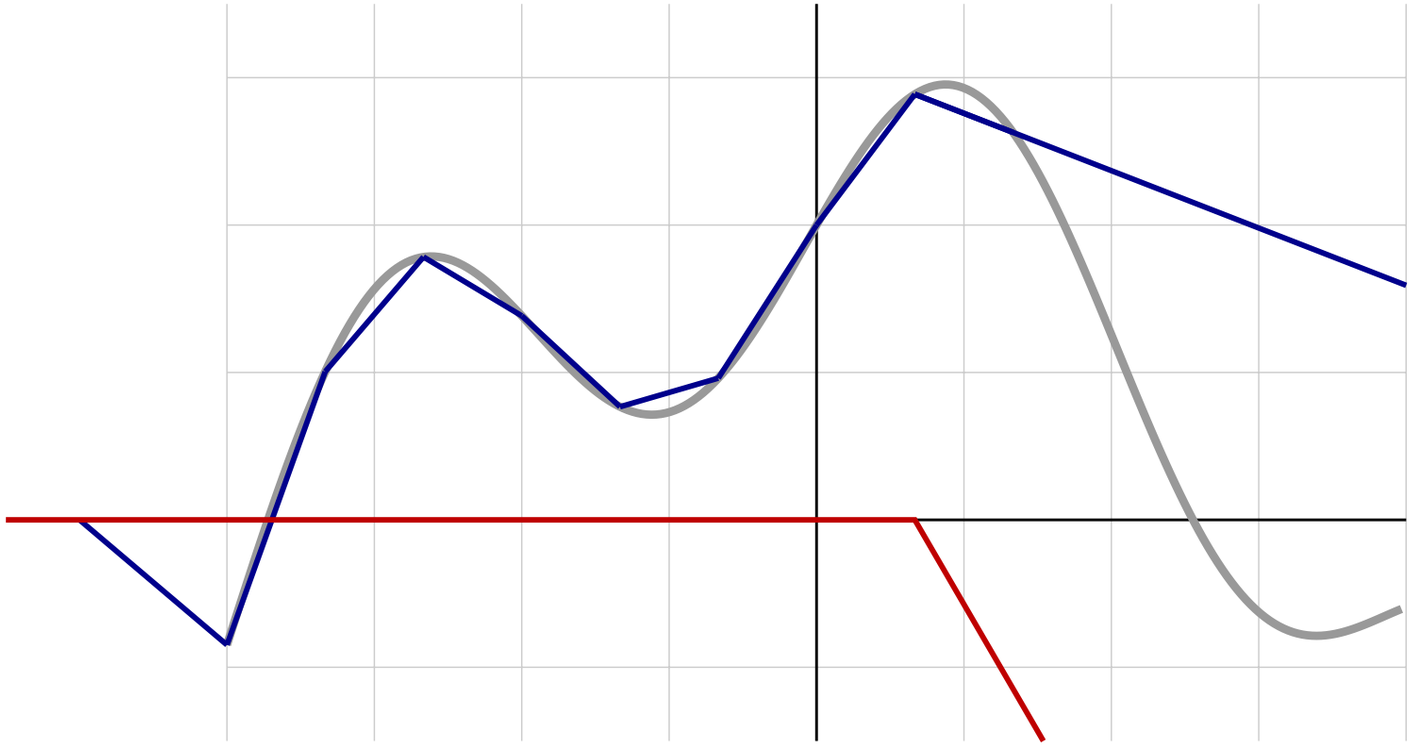
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

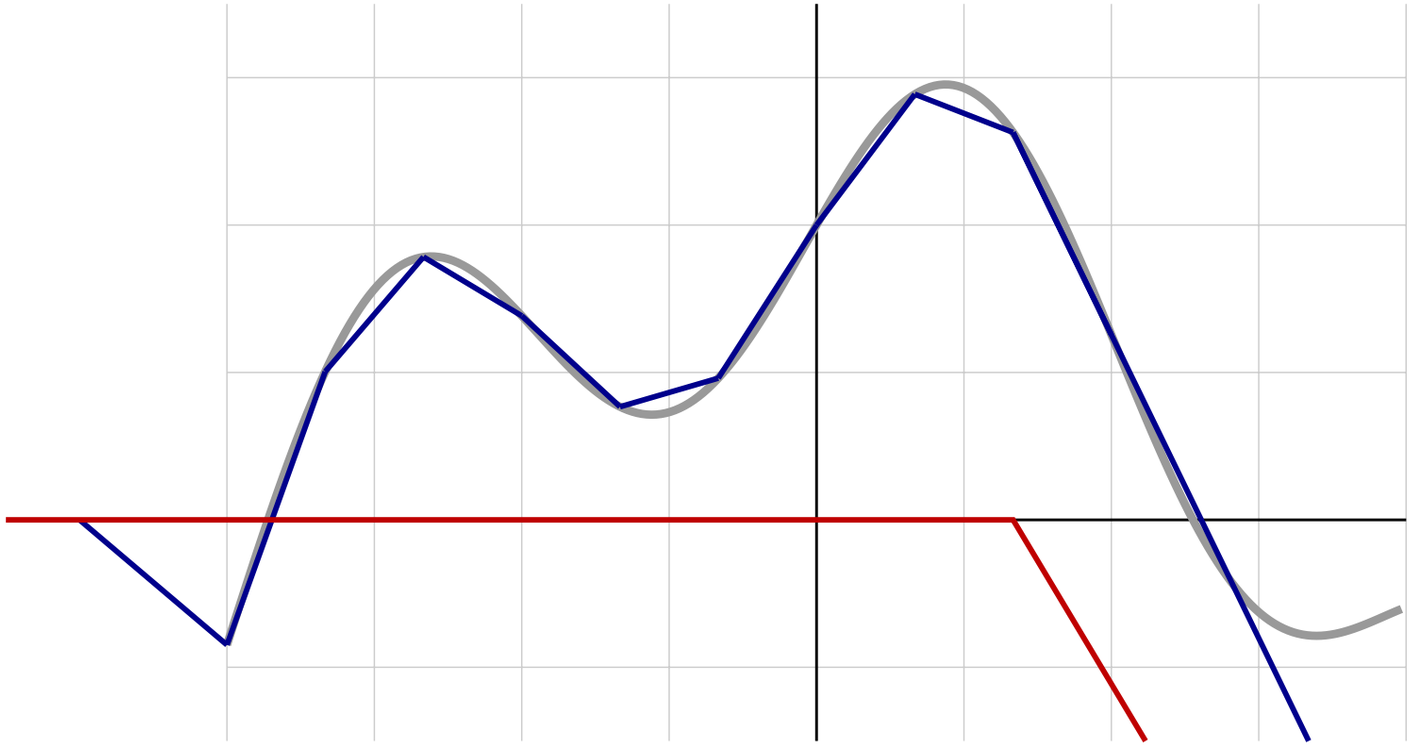
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

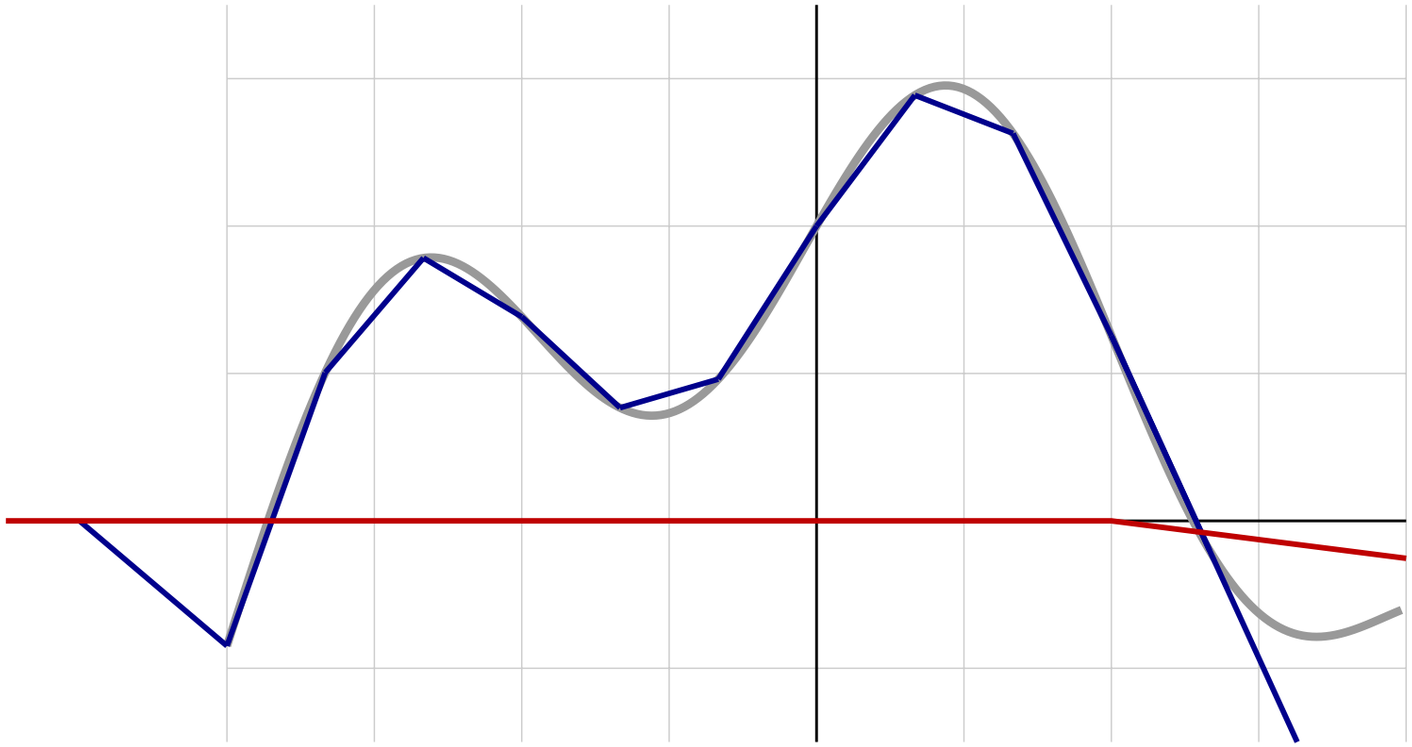
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

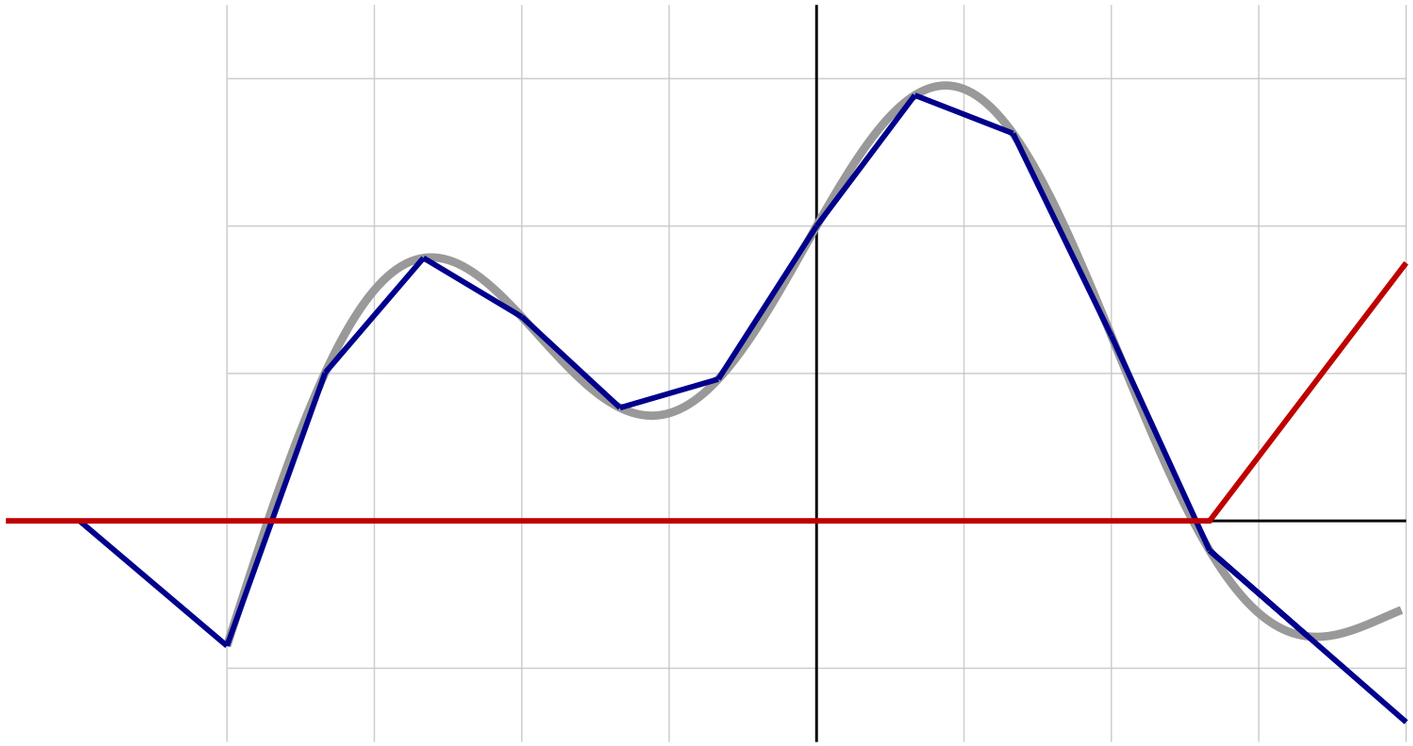
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

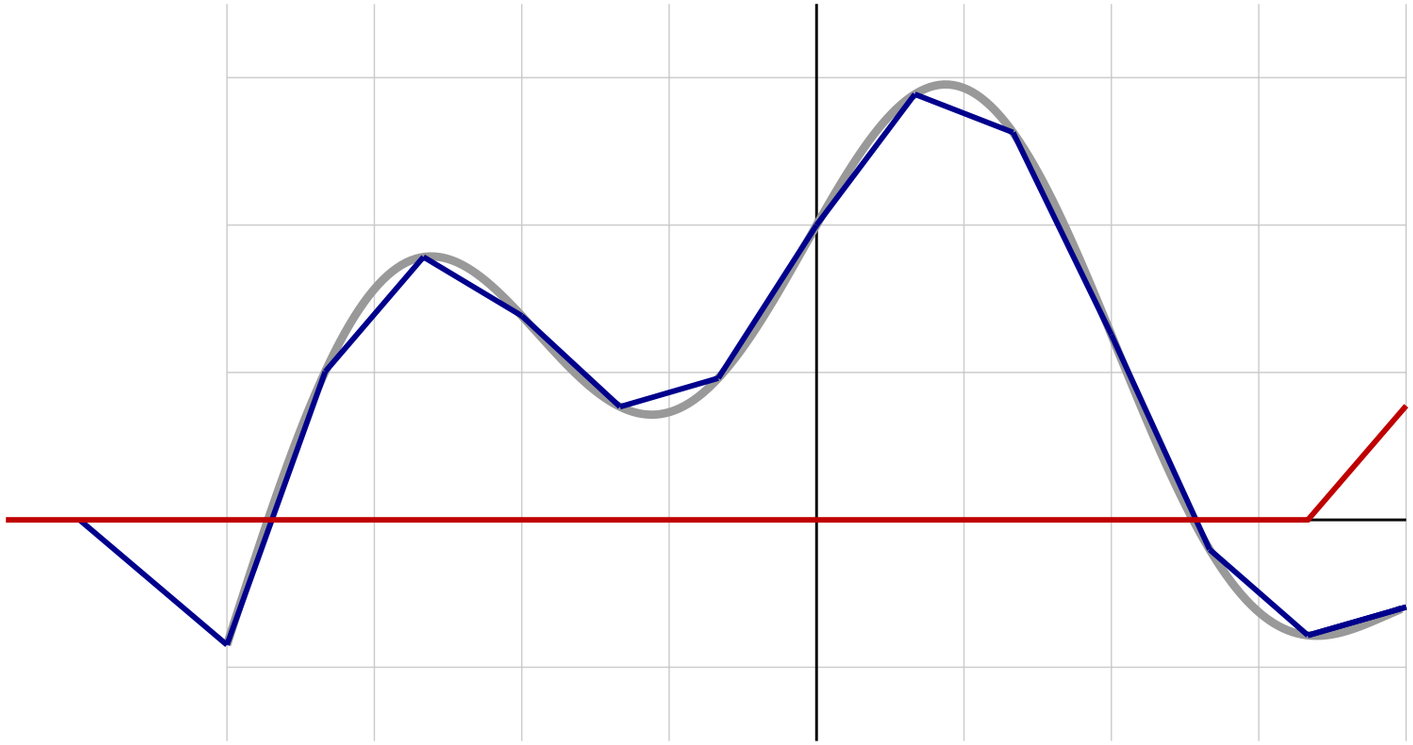
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

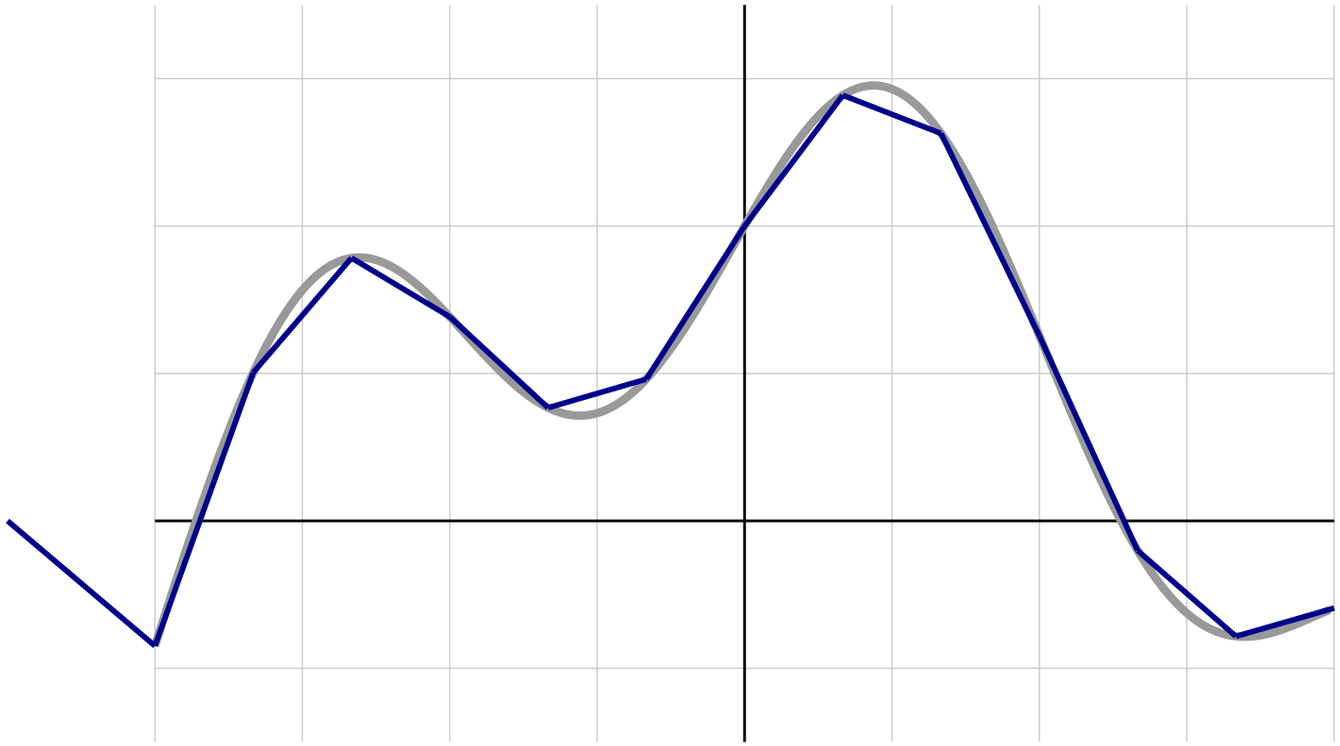
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

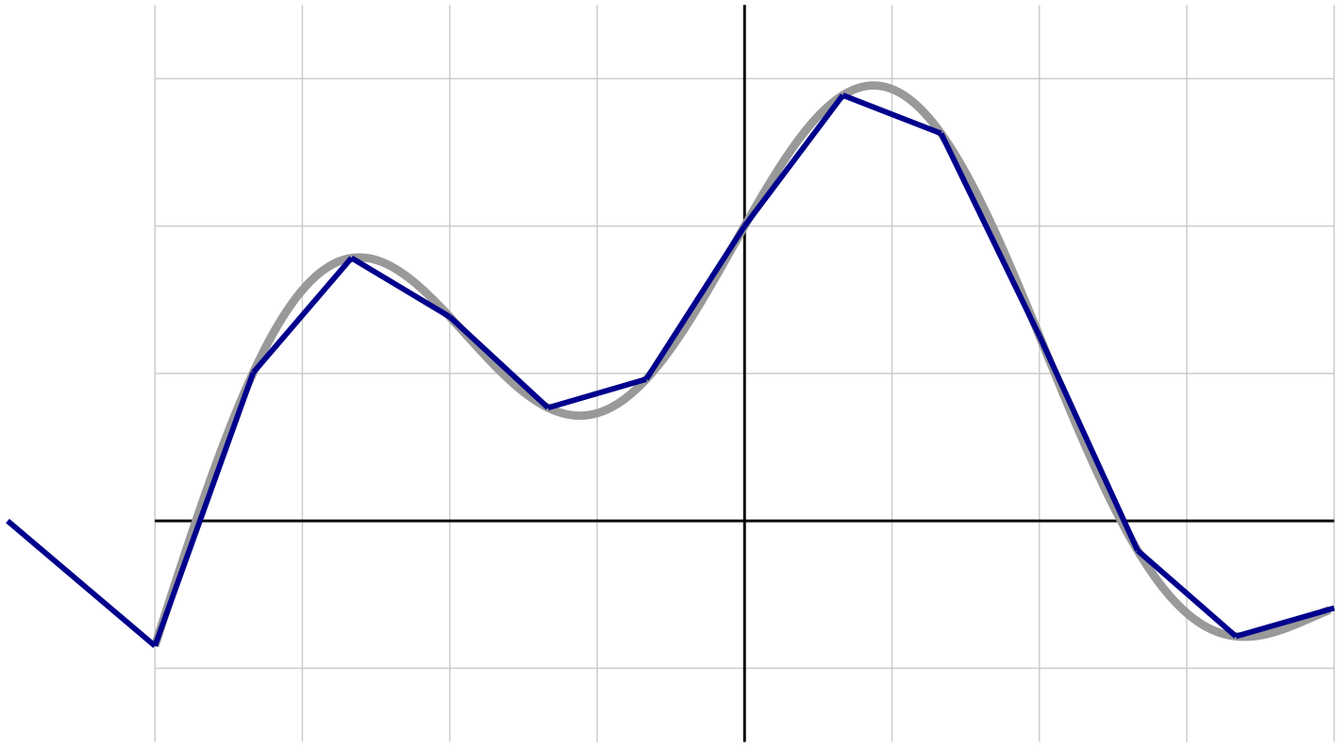
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$

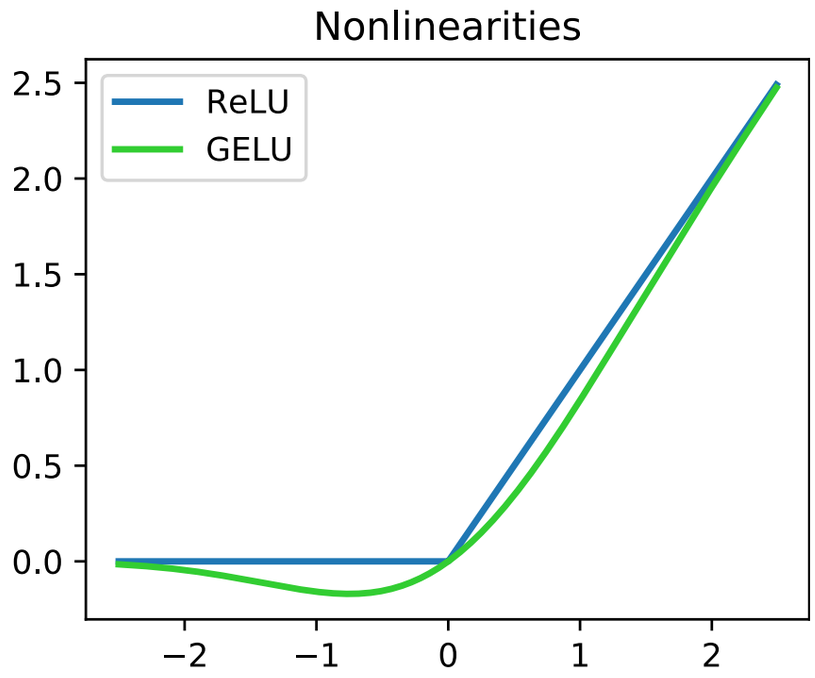


This is true for other activation functions under mild assumptions.

Original Slides from François Fleuret
https://www.idiap.ch/~fleuret/

# Activation functions

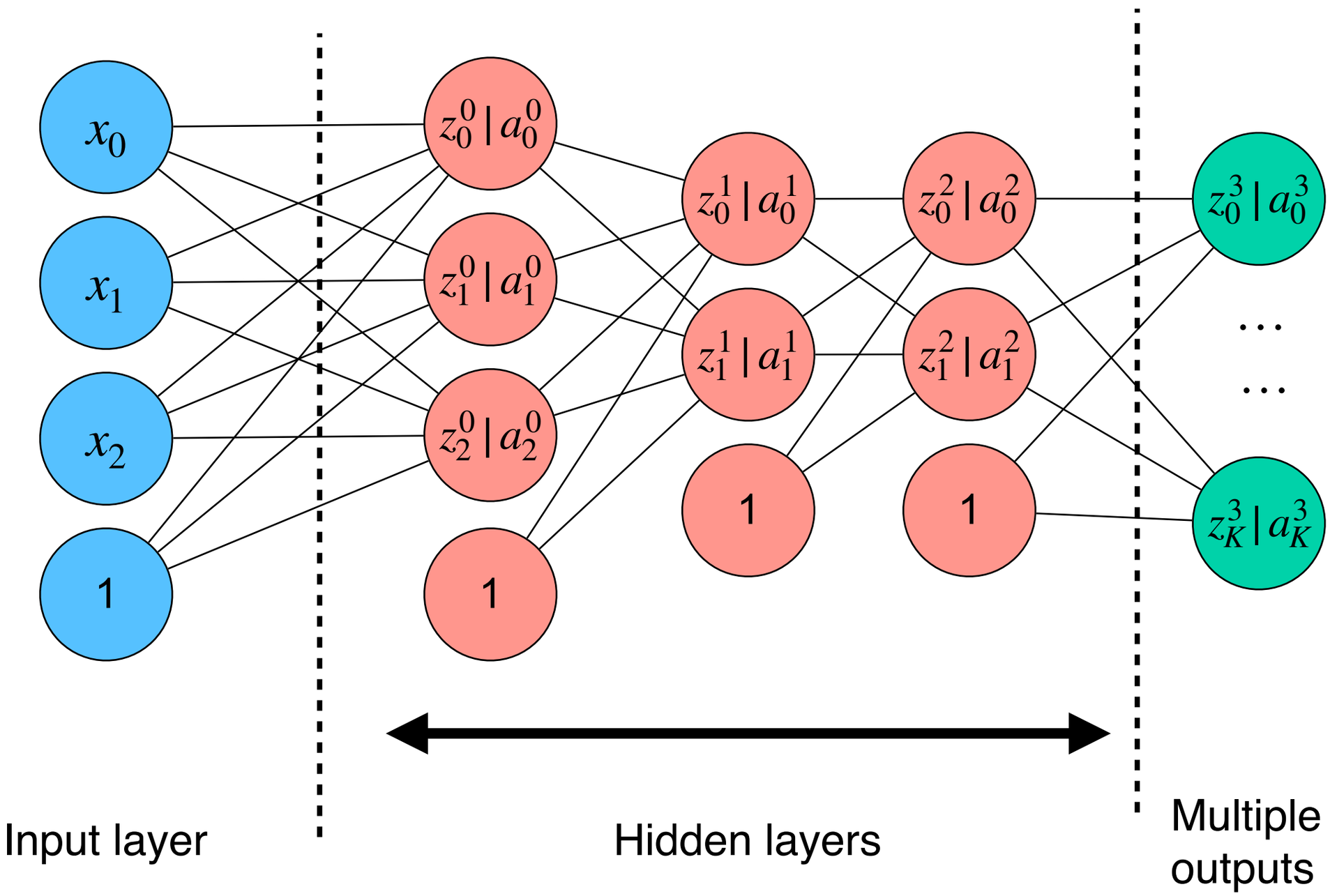
- Variant of ReLU:
  - GeLU - Gaussian Error Linear Unit




By Ringdongdang - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=95947821

Hendrycks, Dan, and Kevin Gimpel. "Gaussian error linear units (gelus)." *arXiv preprint arXiv:1606.08415* (2016).

# How to learn the weights?



# Frank Rosenblatt Perceptron (1956)

Optimization

- Initialize weights $\mathbf{w}$ to zero (or small random)

- For each sample $j$ in the training set $D = \{\mathbf{x_j}, d_j\}$:
  - Compute prediction

$$y_j(t) = f(\mathbf{w}(t) \,.\, \mathbf{x}_j)$$

$$= f(w_0(t)\ x_{j,0} + w_1(t)\ x_{j,1} + w_2(t)\ x_{j,2} + \ldots + w_{N-1}(t)\ x_{j,N-1})$$

- Update all weights $0 \leq i \leq N-1$

$$w_i(t+1) = w_i(t) + r \cdot (d_j - y_j(t))\ x_{j,i}$$

$r$ learning rate (usually small)

# Frank Rosenblatt Perceptron (1956)

Optimization

- Initialize weights $\mathbf{w}$ to zero (or small random)

- For each sample $j$ in the training set $D = \{\mathbf{x_j}, d_j\}$:
  - Compute prediction

$$y_j(t) = f(\mathbf{w}(t) \,.\, \mathbf{x}_j)$$
$$= f(w_0(t)\ x_{j,0} + w_1(t)\ x_{j,1} + w_2(t)\ x_{j,2} + \ldots + w_{N-1}(t)\ x_{j,N-1})$$

  - Update all weights $0 \leq i \leq N-1$

$$w_i(t+1) = w_i(t) + r \cdot (d_j - y_j(t))\ x_{j,i}$$

$r$ learning rate (usually small)

Update done for each observed sample - *stochastic gradient descent*

# Gradient descent

**First-order** iterative optimization algorithm
to find a **local minimum**
of a **differentiable function**

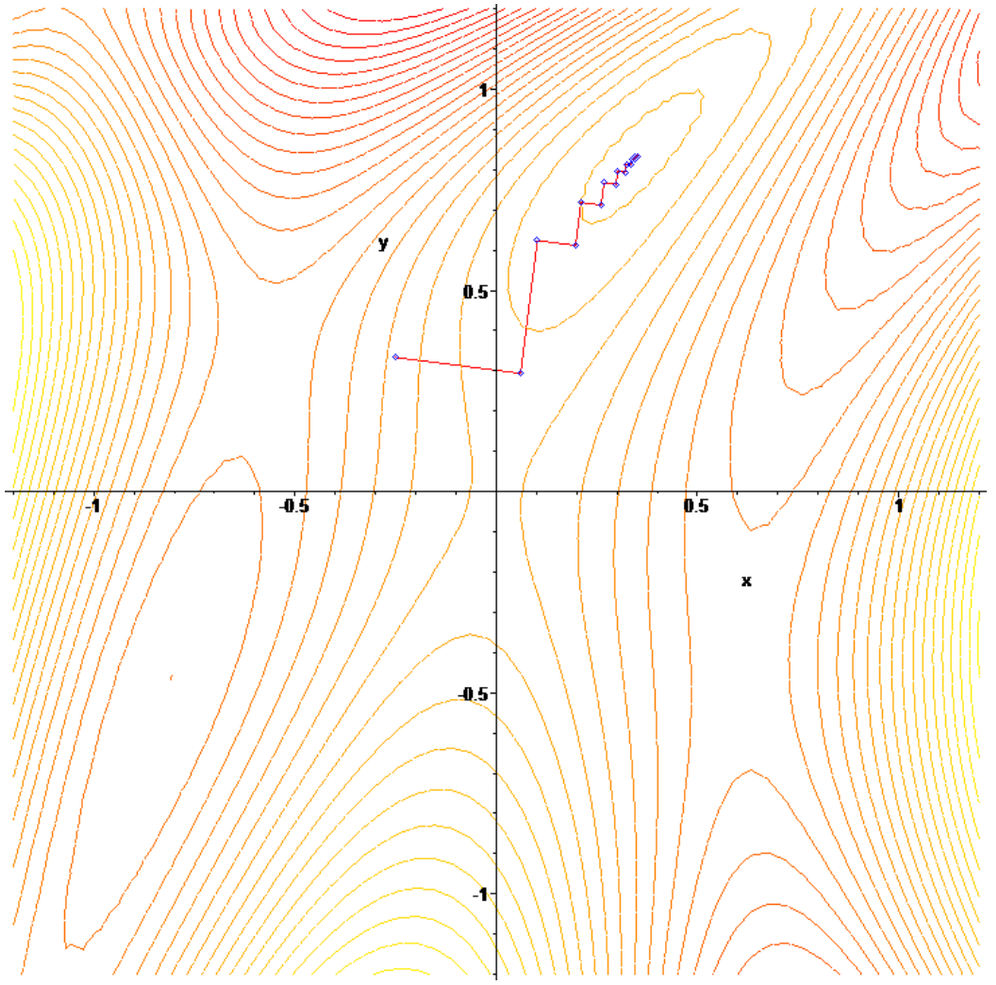
$$f : \mathbb{R}^D \to \mathbb{R}$$
$$x \to f(x_1, \ldots, x_D)$$

$$\nabla f : \mathbb{R}^D \to \mathbb{R}^D$$
$$x \to \left( \frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_D}(x) \right)$$

# Gradient descent

Fog in the mountain analogy

By U.S. Forest Service- Pacific Northwest Region - Okanogan-Wenatchee National Forest, morning fog shrouds trees.jpg, Public Domain, https://commons.wikimedia.org/w/index.php?curid=72715391




By user:Joris Gillis - Created with Maple 10, Public Domain, https://commons.wikimedia.org/w/index.php?curid=521419

# Gradient descent

Starting with a guess $x_0$ consider the sequence $x_0, x_1, x_2, \ldots$

$$x_{n+1} = x_n - \gamma_n \nabla f(x_n)$$

For a small $\gamma_n$ we obtain that

$$f(x_0) \geq f(x_1) \geq f(x_2) \geq \ldots$$

$\gamma_n$ can change at every iteration

Example: Barzilai-Borweing method

$$\gamma_n = \frac{|\,(x_n - x_{n-1})^T[\,\nabla f(x_n) - \nabla f(x_{n-1})]\,|}{||\,\nabla f(x_n) - \nabla f(x_{n-1}\,||^2}$$

# Gradient descent

Main limitation of gradient descent for learning:

- Evaluation of the gradient is computationally expensive

$$\nabla f(\omega, d_i), \ \omega \in \mathbb{R}^N, d_i \in \mathbb{R}^M, i \in [1,K]$$

- $\omega$ network parameters
- dataset $\{d_i, y_i\}$ with $i \in [1,\ K]$ with K big

To evaluate the function (gradient) one needs to use the entire dataset (K samples)

Solution: stochastic approximation of gradient descent

Estimate the function (gradient) from a randomly selected subset $\{d_i, y_i\}, i \in [1,\ L], L \ll K$

# Stochastic gradient descent

**First-order** iterative optimization algorithm
to find a **local minimum**
of a **differentiable function**

**Stochastic approximation** of **gradient descent**

Reduces the computational cost:

- Faster iterations
- Lower convergence rate

# Stochastic gradient descent

Minimize

$$f(\omega) = \frac{1}{M}\sum_{i=1}^{M} f_i(\omega)$$

M number of data points

$$\omega_{n+1} = \omega - r\nabla f(w)$$

$$= \omega - \frac{r}{M}\sum_{i=1}^{M} \nabla f_i(\omega)$$

In stochastic (“on-line”) gradient descent $\nabla f(w) \approx \nabla f_i(w)$ or $\nabla f(w) \approx \sum_{i=1}^{L} \nabla f_i(w)$ (mini-batch)

$$\omega_{n+1} = \omega - r\nabla f_i(\omega)$$

# How to learn the weights?

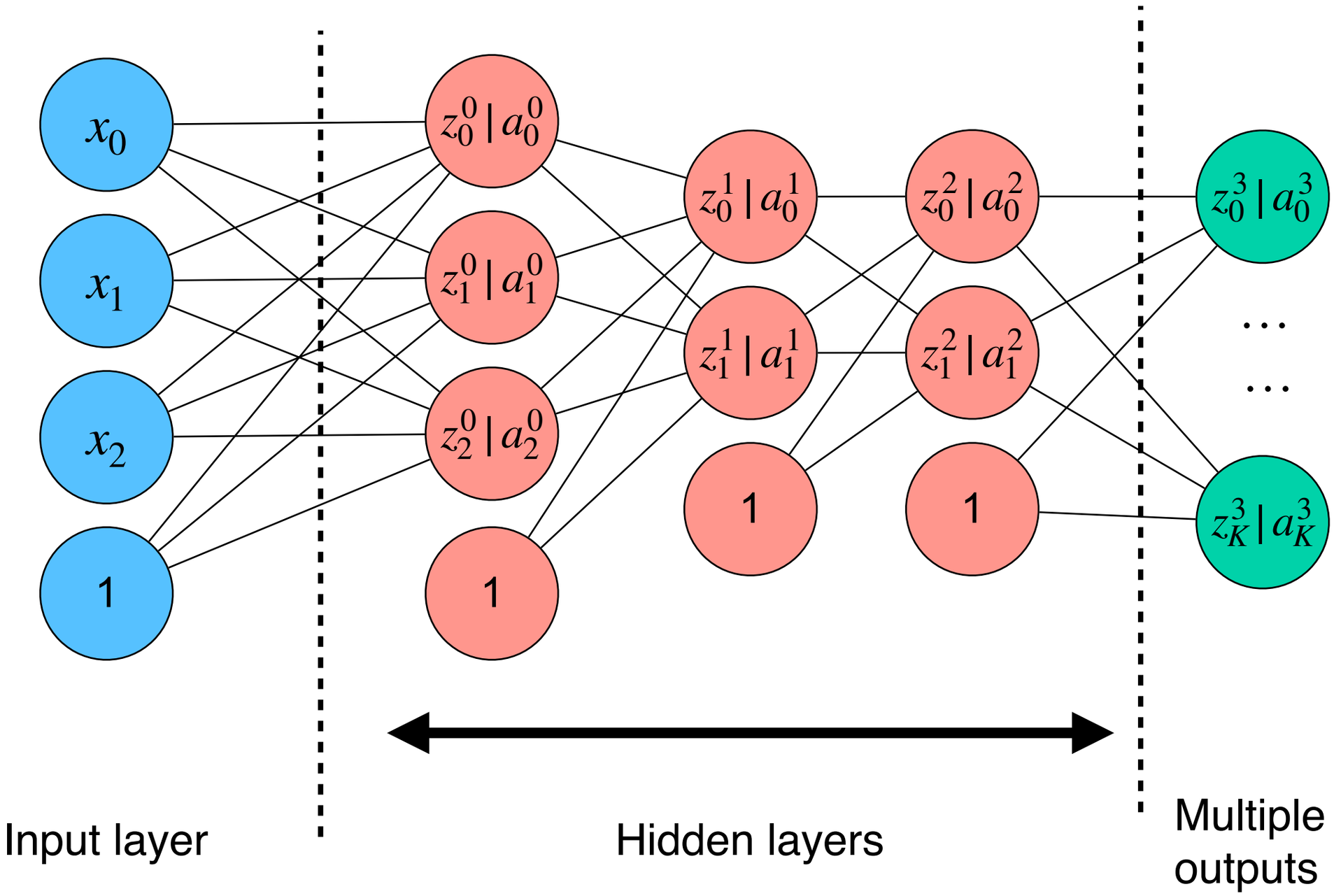
Which optimization technique to use?

- Gradient Descent
- Stochastic Gradient descent (SGD)
- SGD variants:
  - implicit updated (ISGD)
  - averaging
  - momentum
  - RMSProp
  - Adam
- BFGS (second order)
- …

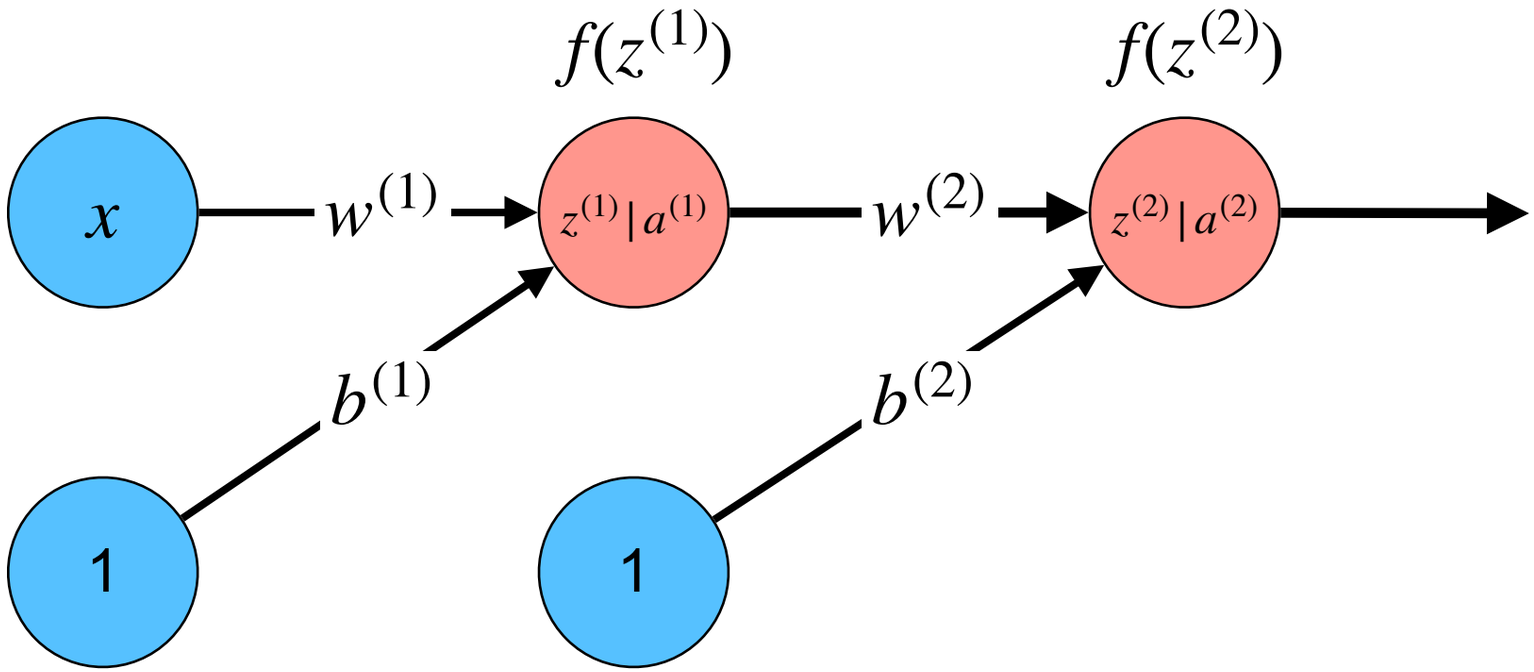
For the methods which are first order or higher:

- How to compute the gradients?

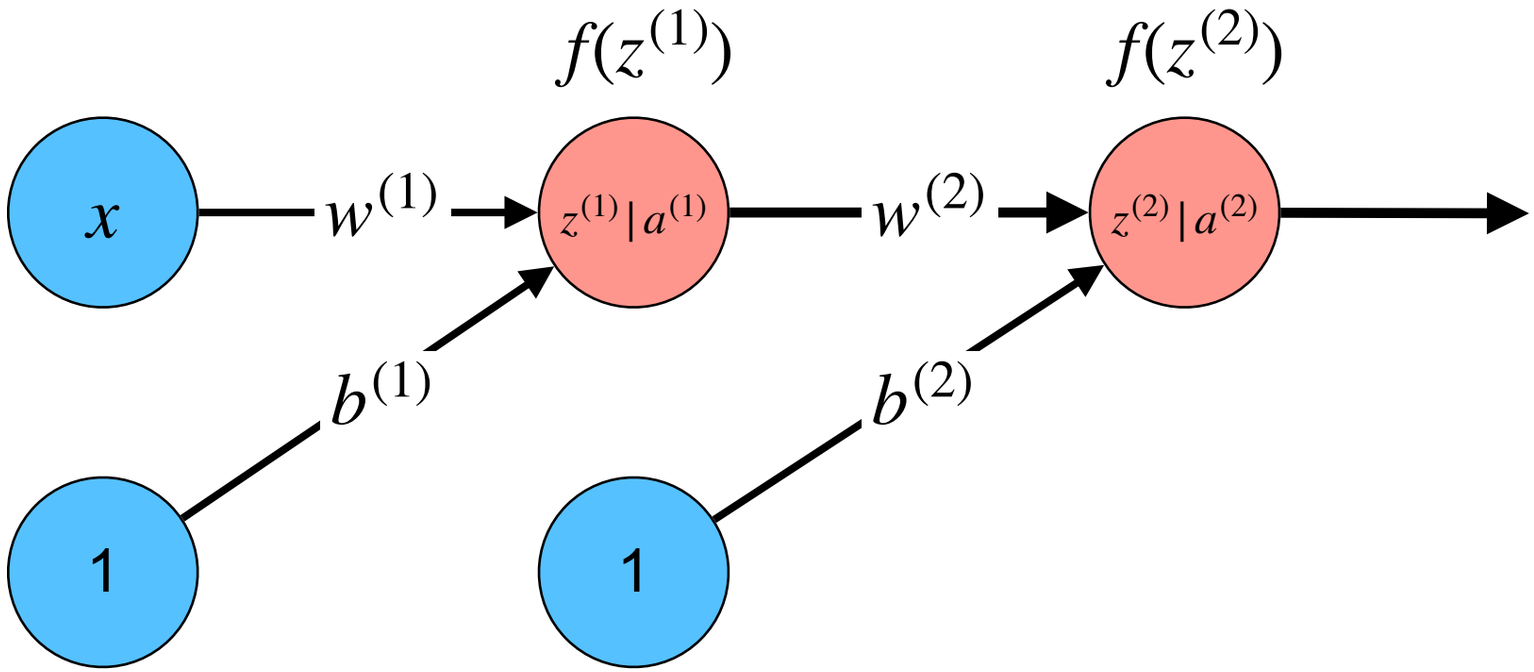
# How to compute the gradients? Backpropagation



# Backpropagation



# Backpropagation




Equations

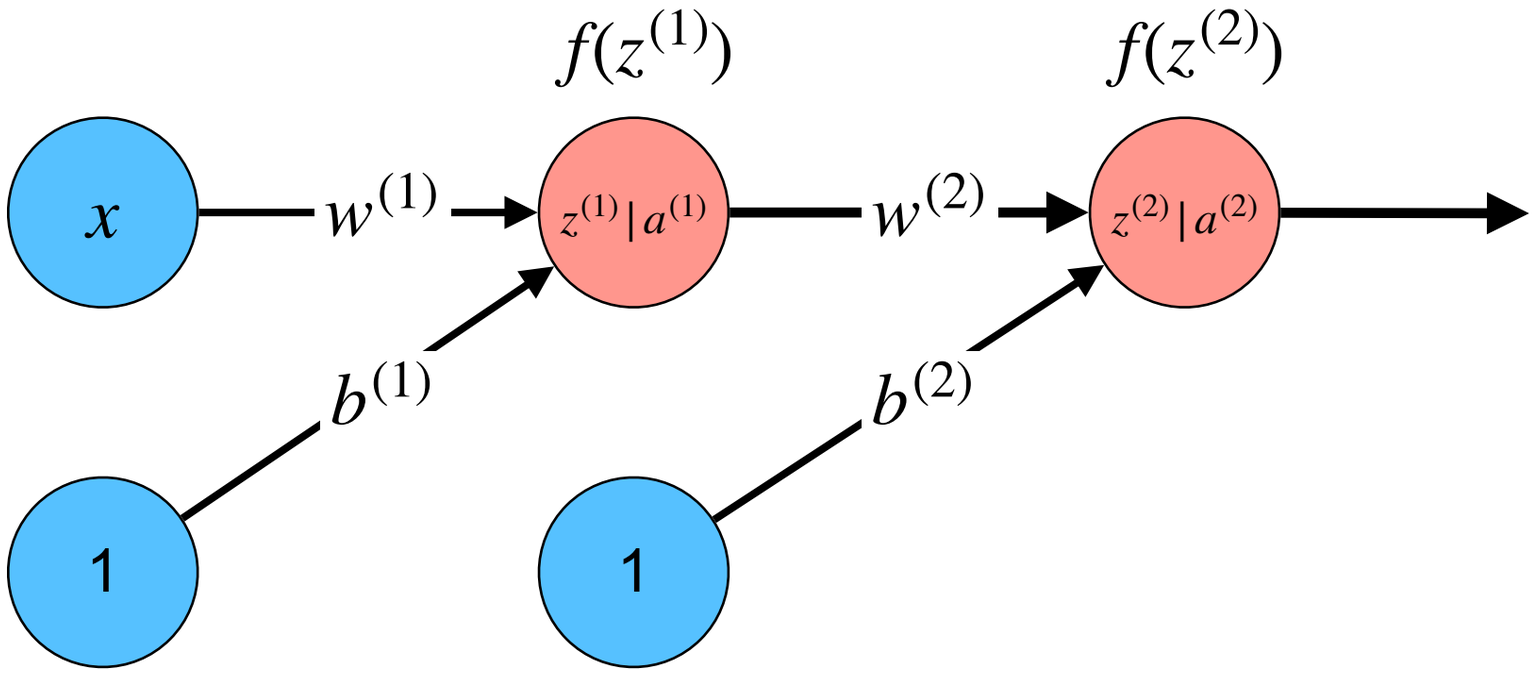
$$z^{(1)} = w^{(1)}x + b^{(1)}$$

$$a^{(1)} = f(z^{(1)}) = f(w^{(1)}x + b^{(1)})$$

$$z^{(2)} = w^{(2)}a^{(1)} + b^{(2)}$$

$$a^{(2)} = f(z^{(2)}) = f(w^{(2)}a^{(1)} + b^{(2)})$$

# Backpropagation



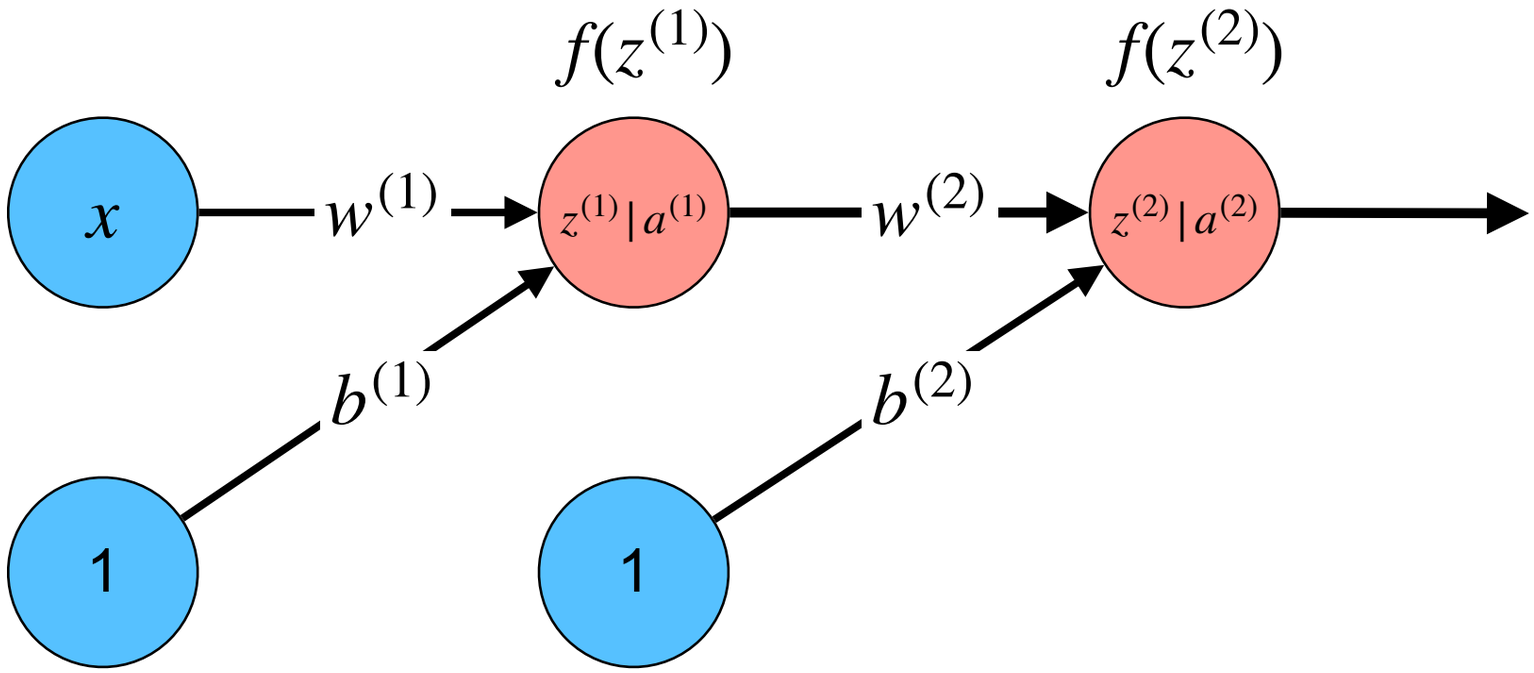

Equations

$$z^{(1)} = w^{(1)}x + b^{(1)}$$
$$a^{(1)} = f(z^{(1)}) = f(w^{(1)}x + b^{(1)})$$
$$z^{(2)} = w^{(2)}a^{(1)} + b^{(2)}$$
$$a^{(2)} = f(z^{(2)}) = f(w^{(2)}a^{(1)} + b^{(2)})$$

Learnable parameters

$$\mathbf{w} = \begin{pmatrix} w^{(1)} \\ b^{(1)} \\ w^{(2)} \\ b^{(2)} \end{pmatrix}$$

# Backpropagation




Equations

$$z^{(1)} = w^{(1)}x + b^{(1)}$$
$$a^{(1)} = f(z^{(1)}) = f(w^{(1)}x + b^{(1)})$$
$$z^{(2)} = w^{(2)}a^{(1)} + b^{(2)}$$
$$a^{(2)} = f(z^{(2)}) = f(w^{(2)}a^{(1)} + b^{(2)})$$

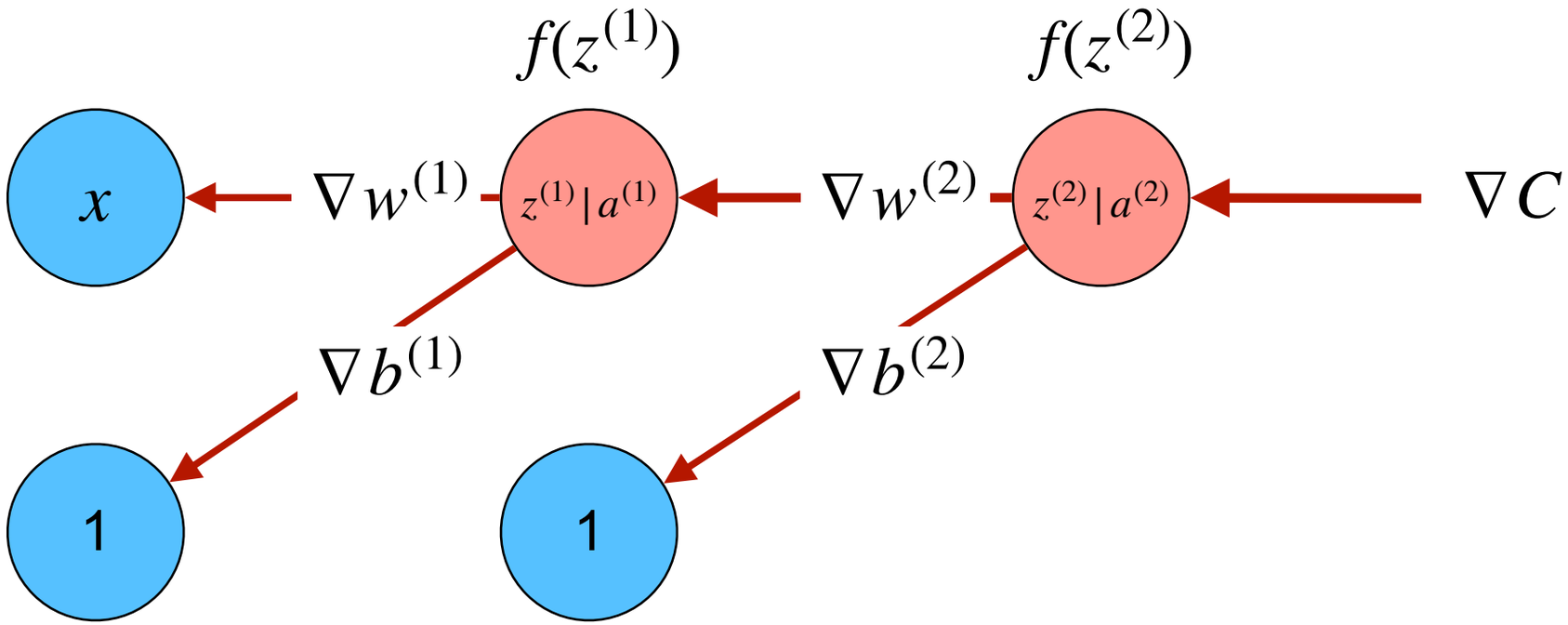
Learnable parameters

$$\mathbf{w} = \begin{pmatrix} w^{(1)} \\ b^{(1)} \\ w^{(2)} \\ b^{(2)} \end{pmatrix}$$

Cost (loss)

$$C = C(x)$$
$$= \frac{1}{2}(a^{(2)} - y)^2$$

# Backpropagation



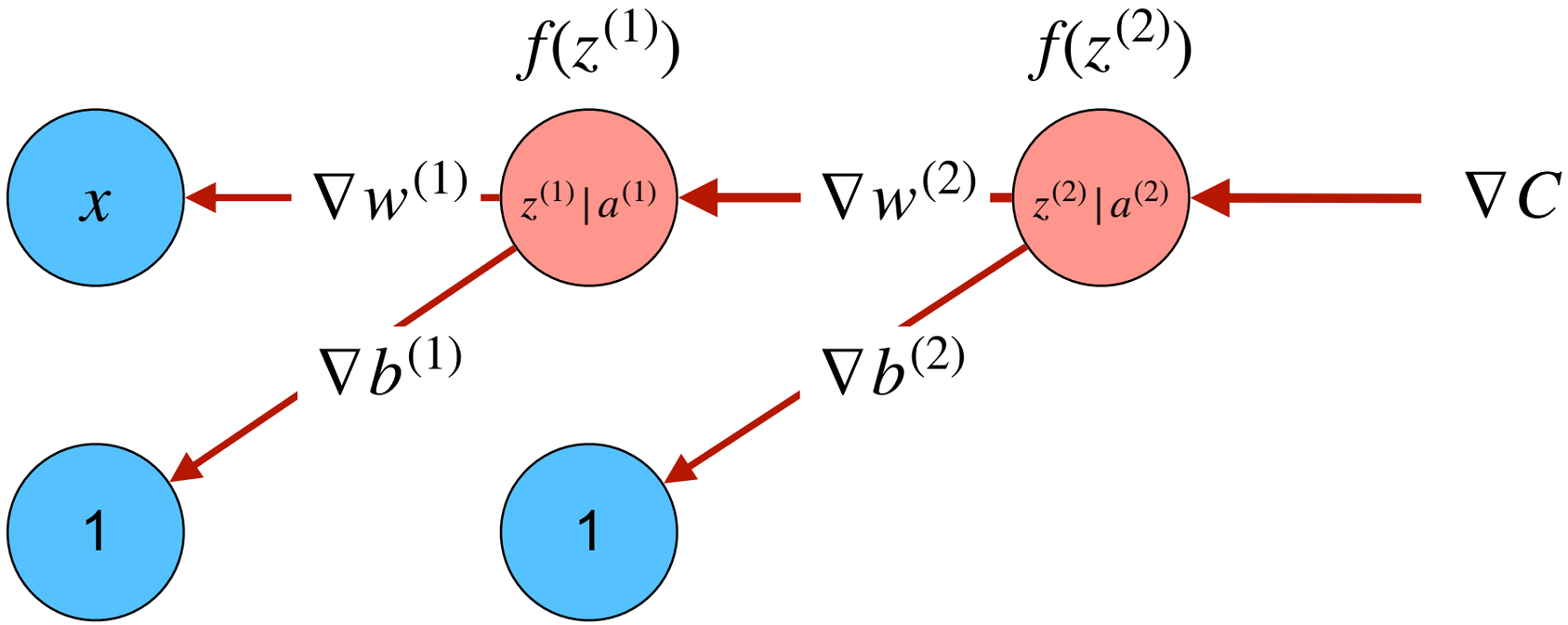

Compute the derivative of the cost wrt to the learnable parameters and update them

$$\nabla C = \frac{\partial C}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \frac{\partial C}{\partial w^{(2)}} \\ \frac{\partial C}{\partial b^{(2)}} \end{pmatrix} = \begin{pmatrix} \nabla w^{(1)} \\ \nabla b^{(1)} \\ \nabla w^{(2)} \\ \nabla b^{(2)} \end{pmatrix}$$

Example: $\frac{\partial C}{\partial w^{(2)}} = \frac{\partial C}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial w^{(2)}}$

Update : $w^{(i)} \leftarrow w_i - r \frac{\partial C}{\partial w_i}(a_i)$

# Backpropagation




Summary:

- Forward pass computes $a^{(i)}$ where derivatives are evaluated
- Chain-rule factorizes the derivatives
- Derivatives of activation functions are “closed form”
- Computing “backwards” allows to reuse the previous computations
- Can be massively parallelized:
  - All individual partials can be parallelized
  - Product is performed per-layer

# How to learn the weights?

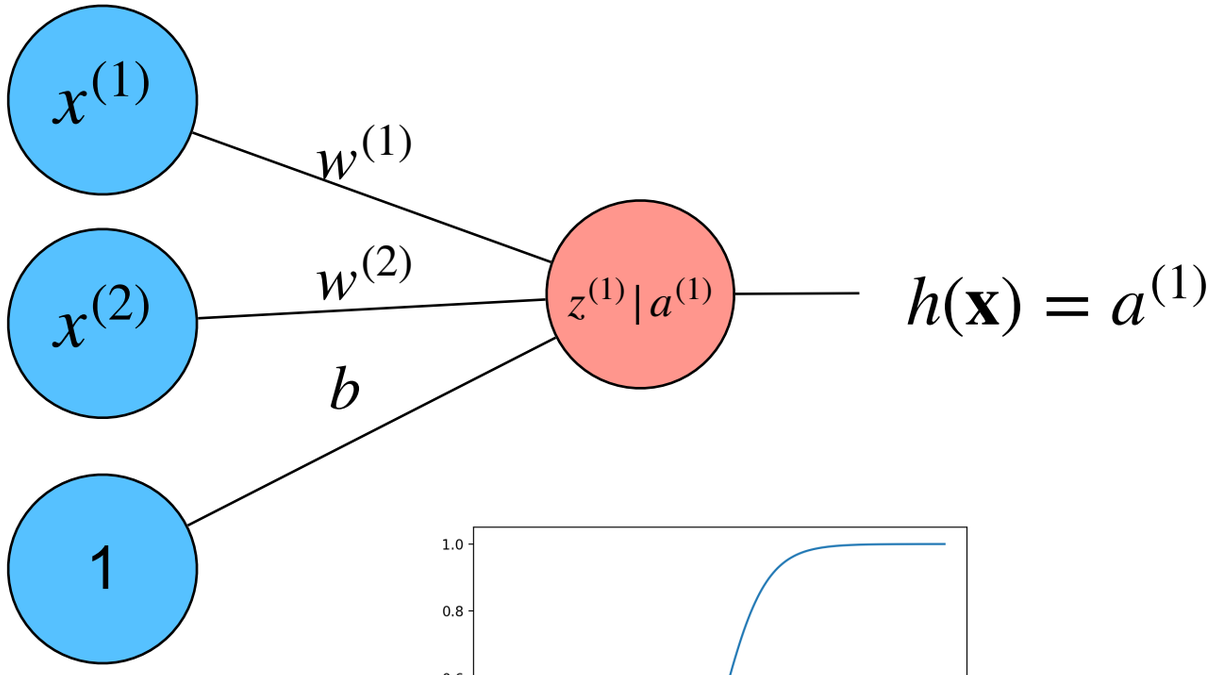
Two layers of abstraction:

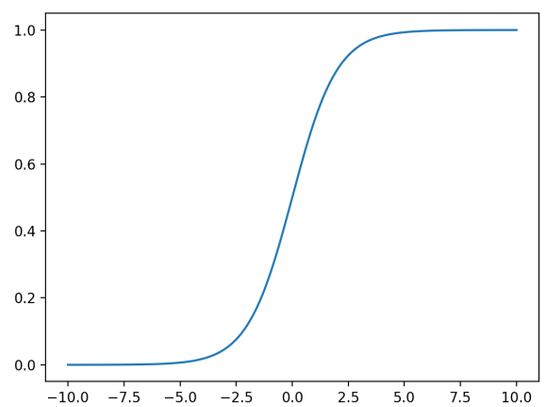
1. Gradient computation:
   - where backpropagation comes to play
2. Optimization level:
   - techniques like GD, SGD, Adam, Rprop, BFGS etc. come into play
     - use the computed gradient (or parts of it)

# Exercice

You are presented with one neuron with two inputs $\mathbf{x} = (x^{(1)}, x^{(2)})$ and a single output $y$.
Your network initial weights are $w^{(1)} = 0.1,\ w^{(2)} = -0.2$ and $b = 0.2$
Your activation function is a logistic function with k=1 : $a(z) = \frac{1}{1+e^{z}}$
We assume a learning rate $r = 0.1$
Your network should be optimized to recognize the following training data $(\mathbf{x_i}, y_i)$ with $i \in \{1,2,3,4\}$
The cost to be optimized is $C(\mathbf{x}) = \frac{1}{2}(h(\mathbf{x}) - y)^2$

Q1: Compute the updated parameters for the observation $\mathbf{x_1}, y_1$ .
Q2: Will the neuron converge for this training data?




| m | x1 | x2 | y |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 1 |
| 3 | 1 | 1 | 1 |
| 4 | 0 | 0 | 0 |

# A note of weights initialization

How to initialize the weights?

Starting with all weights to 0 causes problems (weak signals)
Starting with all weights high causes problems (overflow)

Initialize with small random values:

- sampling from a normal distribution
- Xavier method - optimized for sigmoids
- He method - variant optimized for ReLU activation

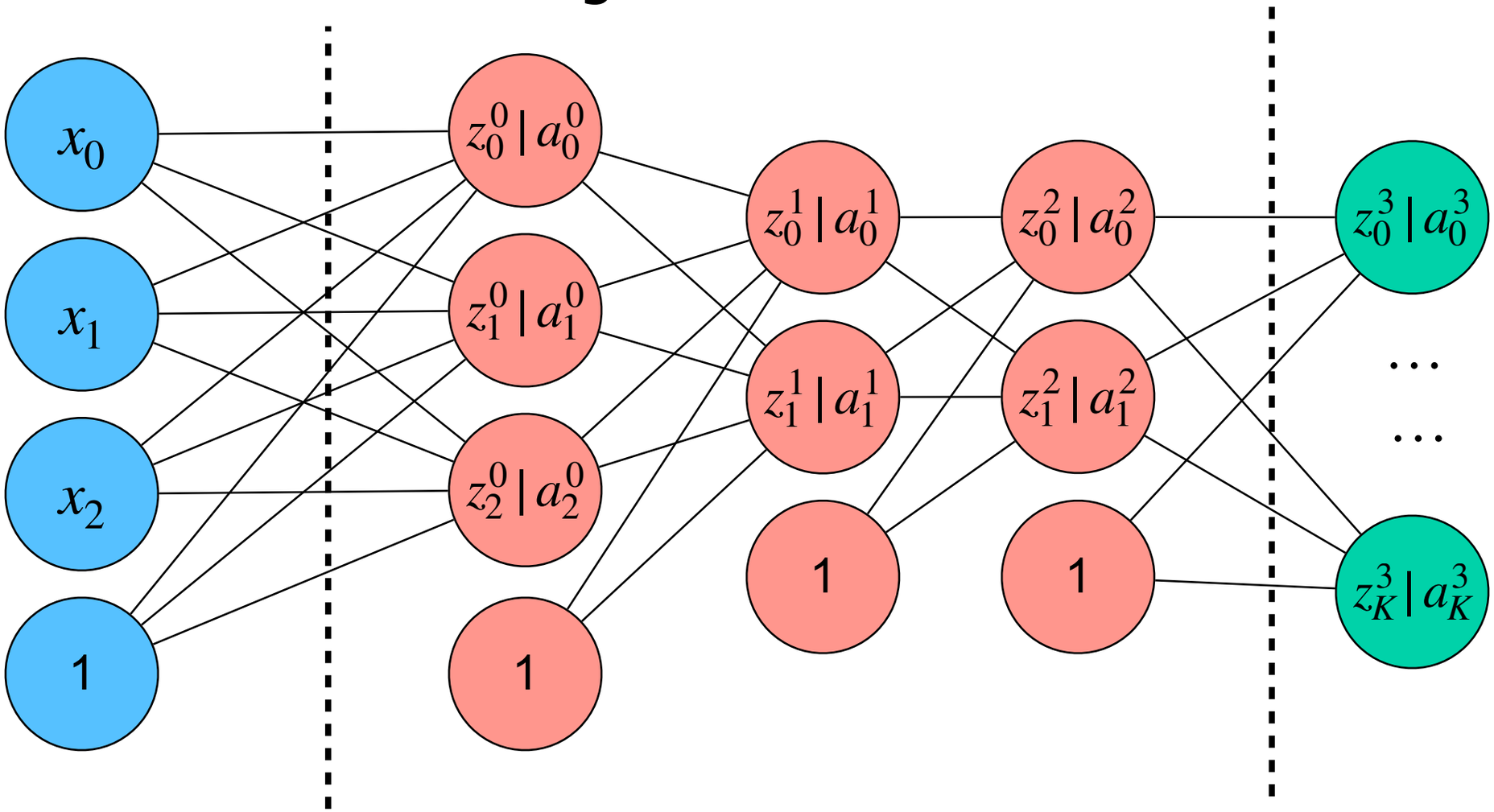
Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249-256. JMLR Workshop and Conference Proceedings, 2010.

# Big Families of Neural Networks

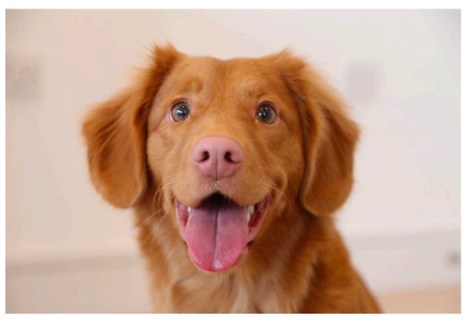
- Multi Layer Perceptron (MLP)
- Convolutional Neural Networks (CNNs)
- Autoencoders (AEs)
  - Variational Autoencoders (VAEs)
- Generative Adversarial Networks (GANs)
- Recurrent Neural Networks (RNNs)
  - Long Short Temporal Memory (LSTMs)
- Graph Neural Networks (GNNs)
- Transformers

# Convolution Neural Networks

- Are MLPs suitable for images?

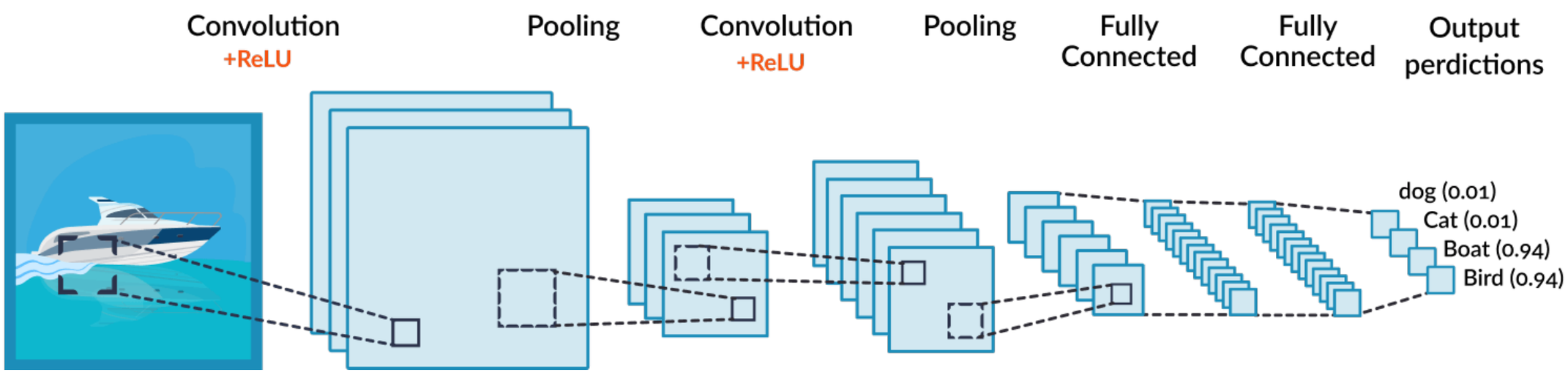






# CNN: Overview

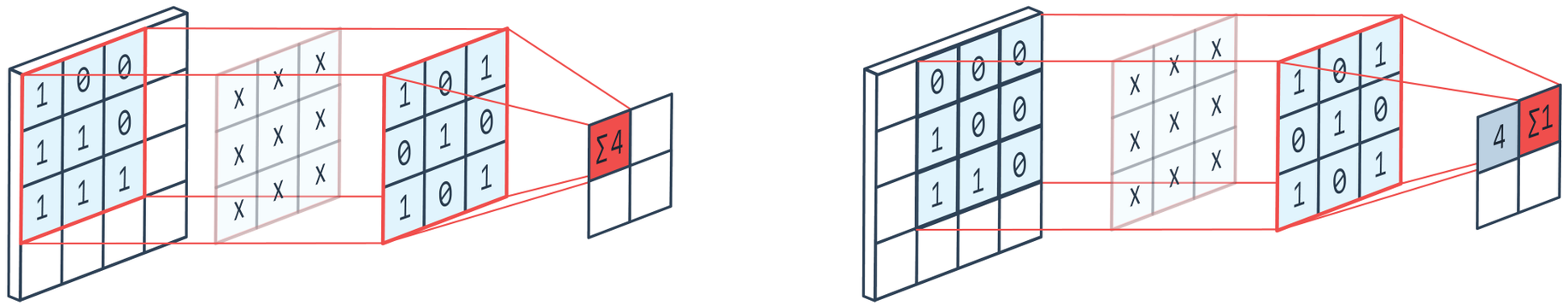
- A convolutional neural network is a special feedforward network
- Hidden units are organized into grid, as is the input
- Linear mapping from layer to layer takes form of convolution
  - Translation invariant processing
  - Local processing




Slide from Xavier Alameda-Pineda

# 2D convolutions

2D convolutions are linear operations:

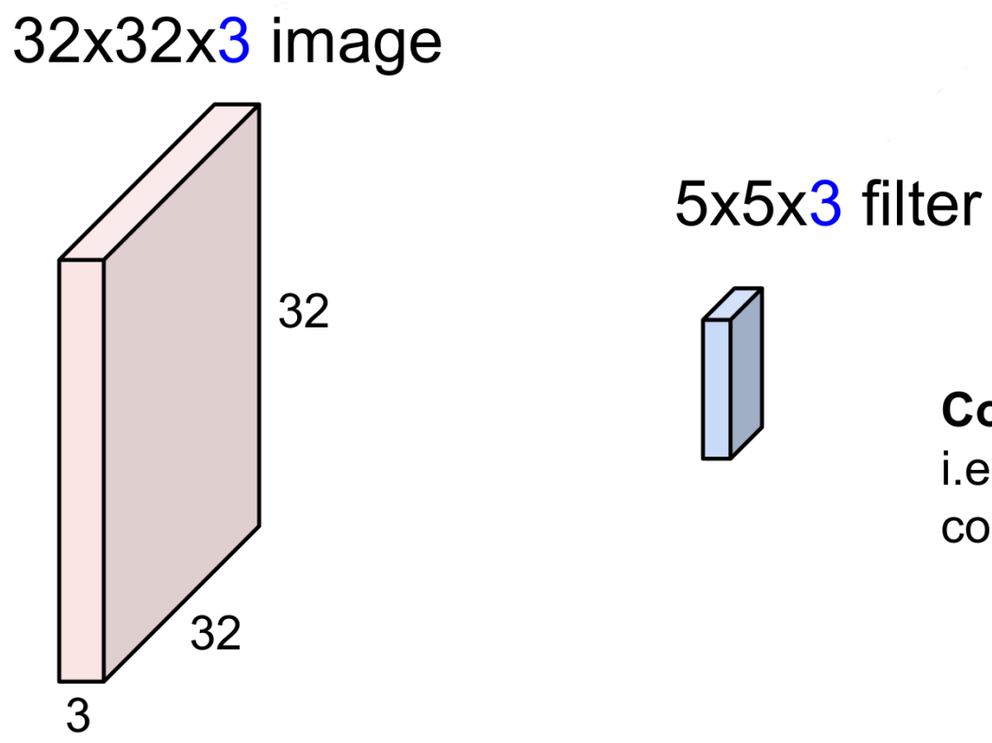



- Start from the top-left position.
- Multiply-and-add.
- Store in the output image.
- Go to next pixel position.

Slide from Xavier Alameda-Pineda

# Convolutions in ConvNets

In ConvNets (or CNNs), the convolution filters or kernels have depth:

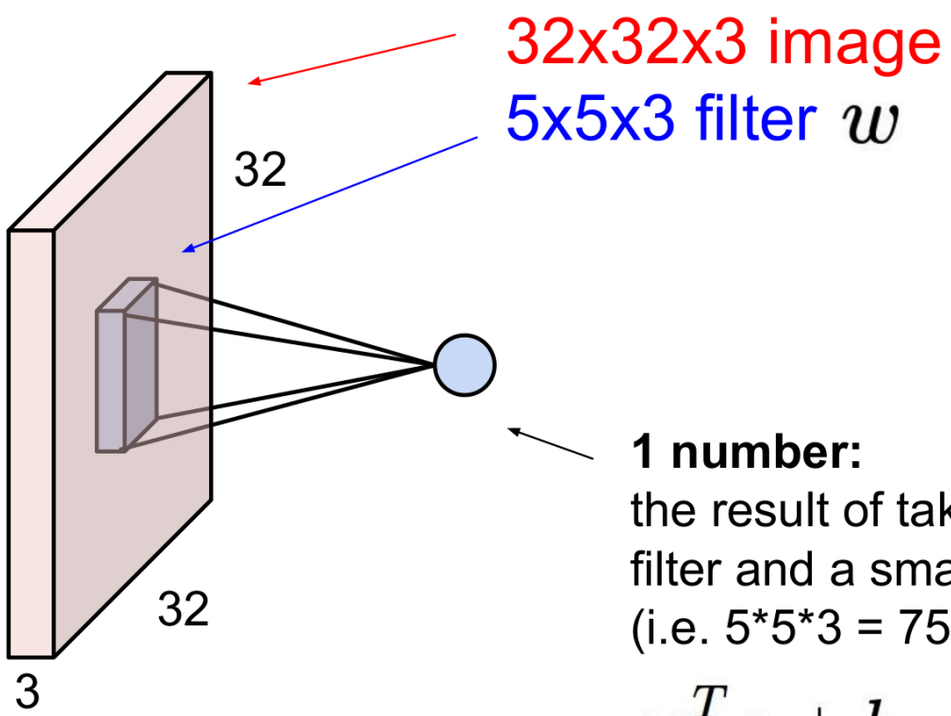



Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Slide from Xavier Alameda-Pineda

# Convolutions in ConvNets (II)

What is the width and height of the output image?



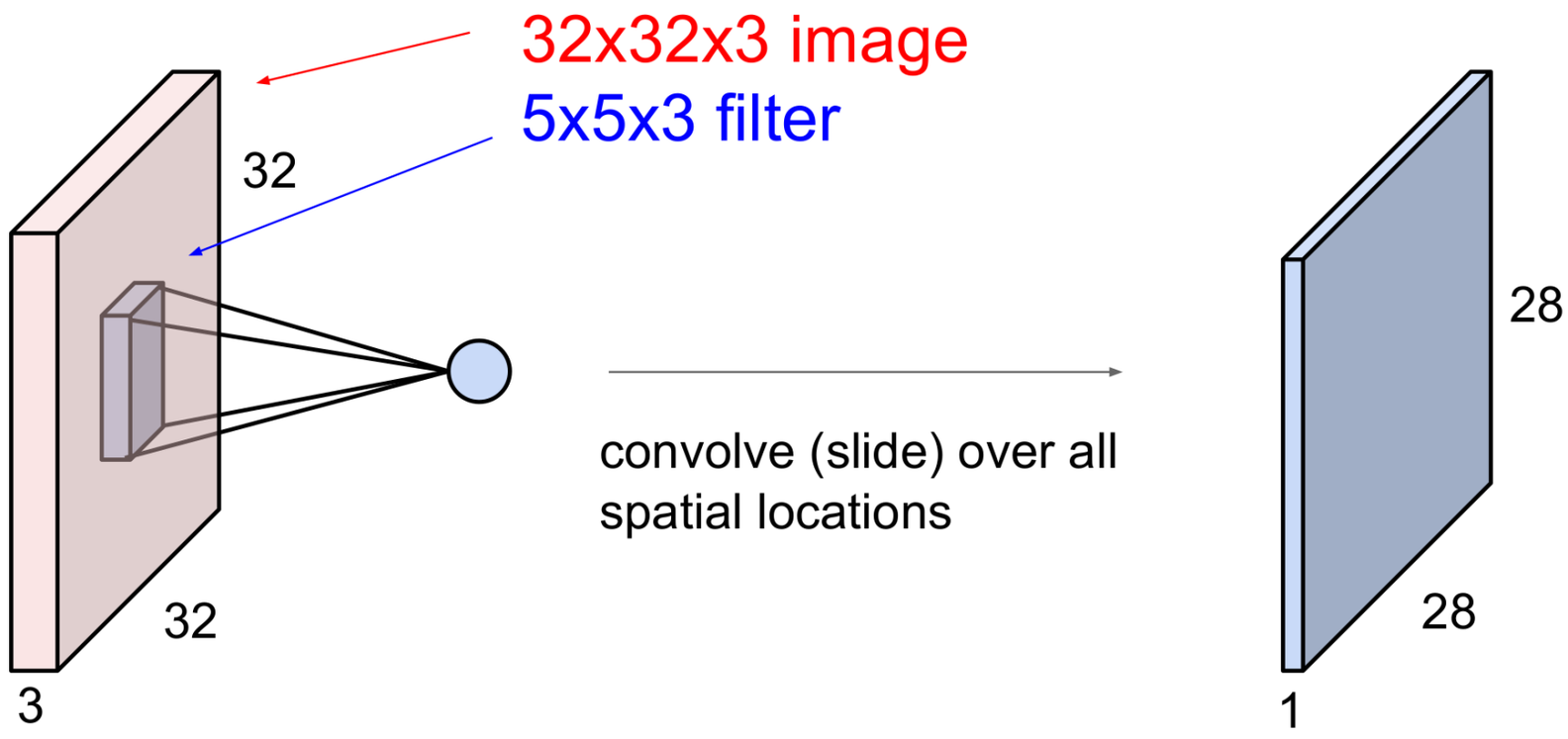

**1 number:**
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Slide from Xavier Alameda-Pineda

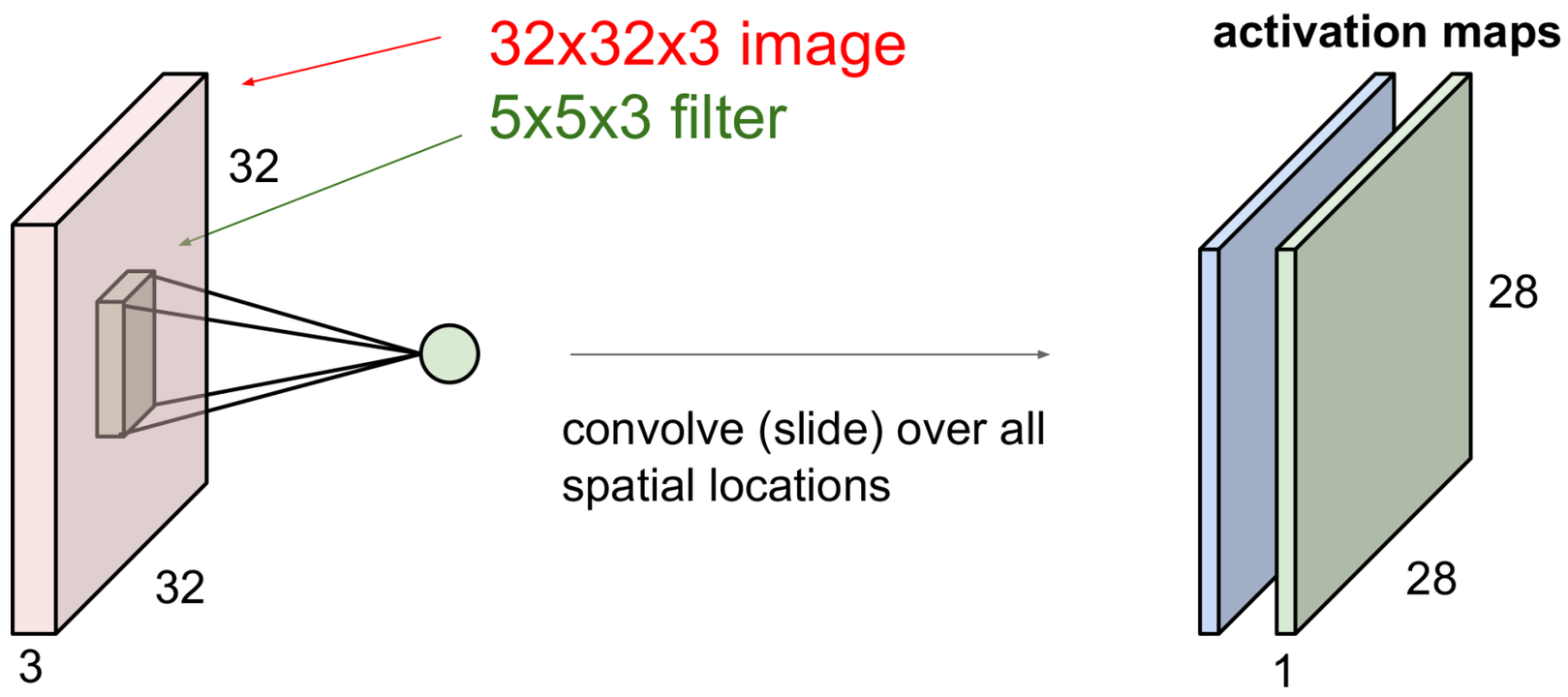
# Convolutions in ConvNets (III)




Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Slide from Xavier Alameda-Pineda

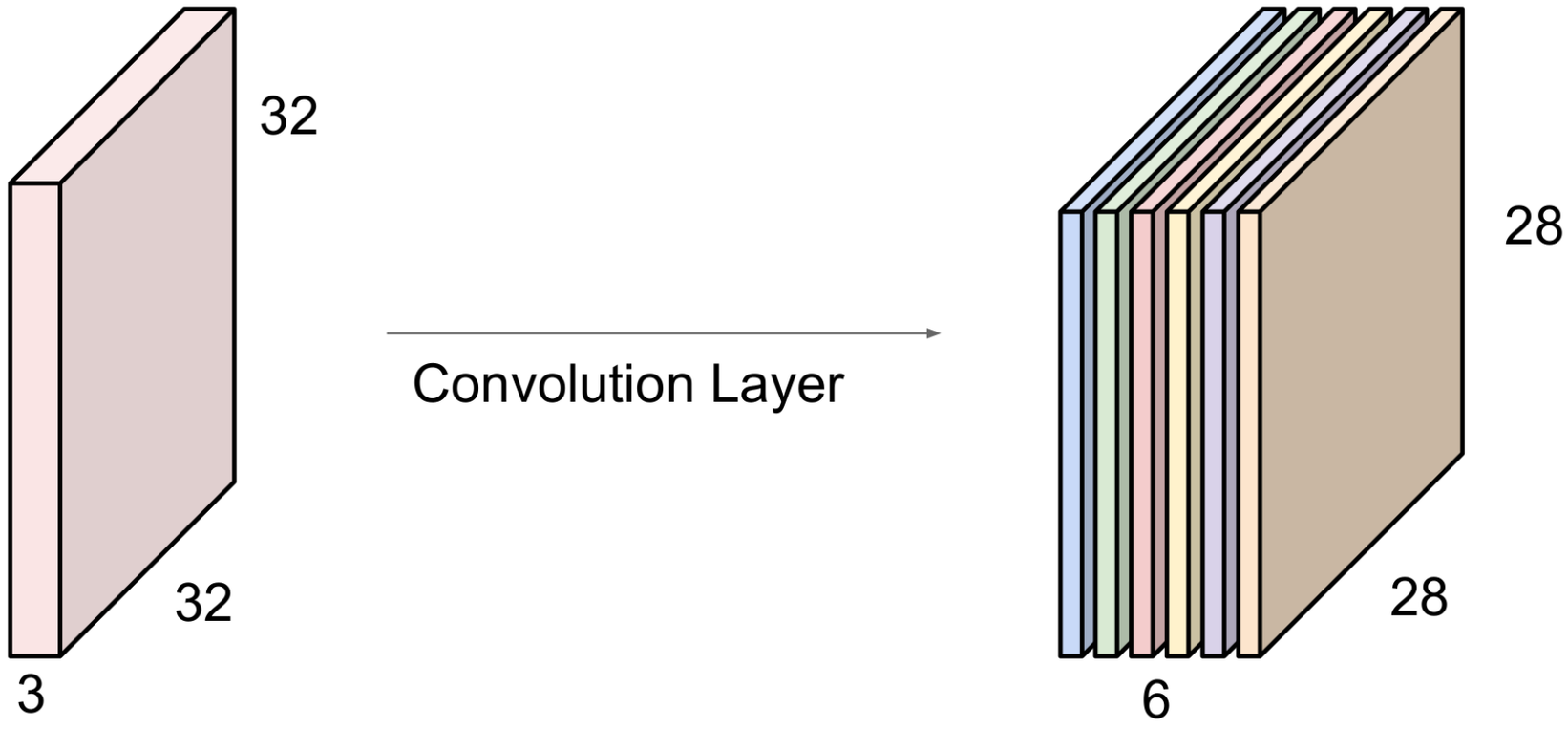
# Convolutions in ConvNets (IV)




Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Slide from Xavier Alameda-Pineda

# Convolutions in ConvNets (V)

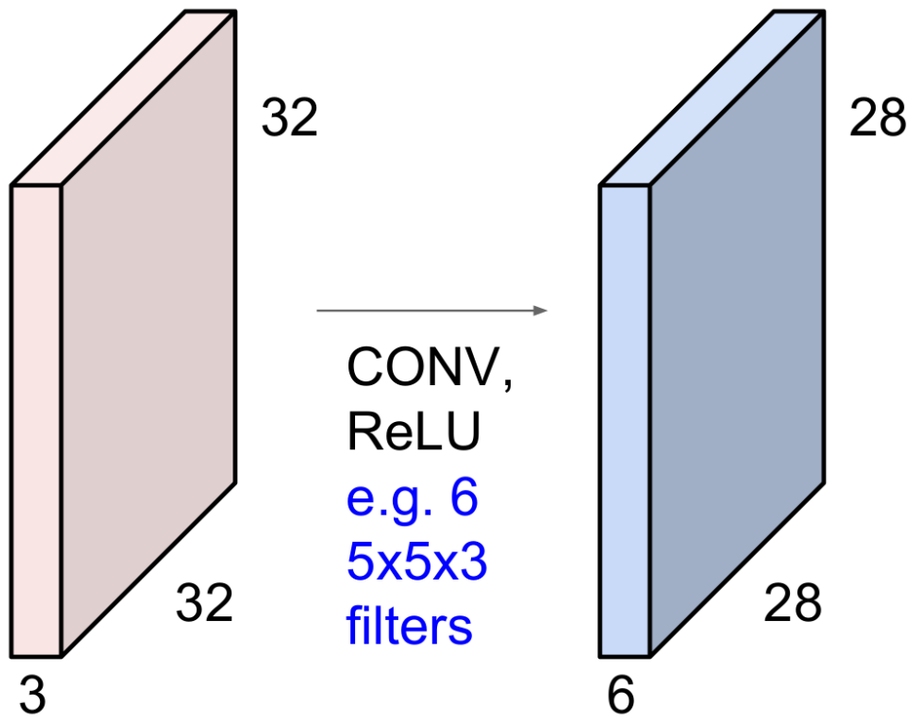



Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Slide from Xavier Alameda-Pineda

# Convolutions in ConvNets (VI)

The filters **must** have the same depth as the input. The output depth corresponds to the number of filters.

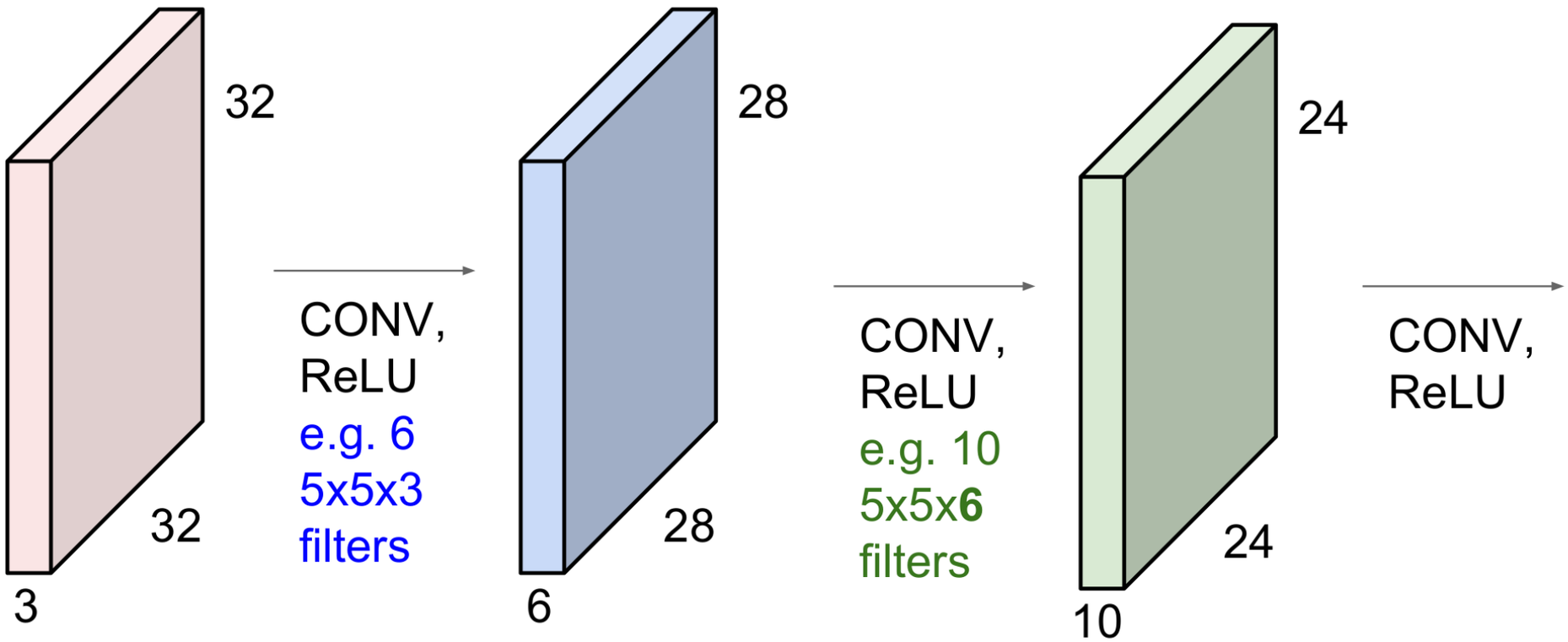



Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Slide from Xavier Alameda-Pineda

# Convolutions in ConvNets (VII)

We can repeat the operation with a second convolutional layer.




Slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Slide from Xavier Alameda-Pineda

# Different types of convolutions

- Standard
- Padded (full, half, etc)
- Strided
- Dilated
- etc.

More details in

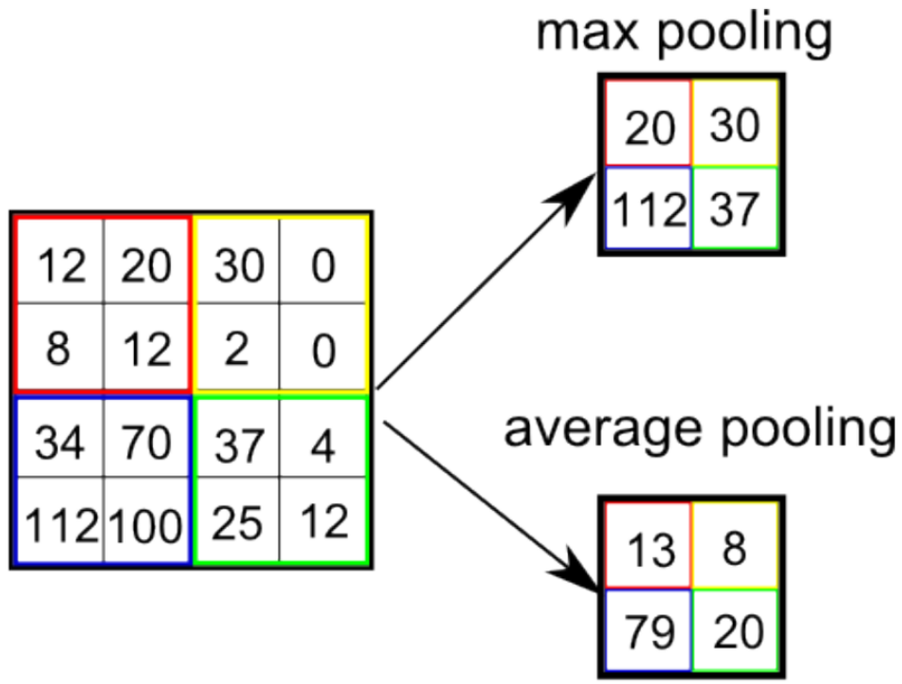
https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).
Chapters 1 & 2

Slide from Xavier Alameda-Pineda

# Pooling

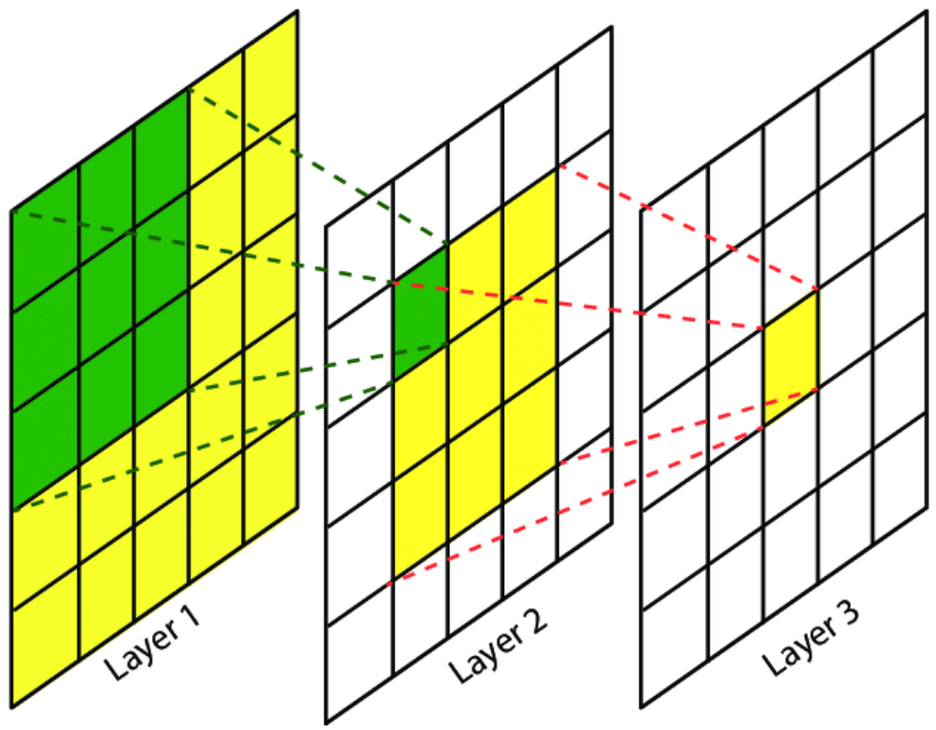
- Applied separately per feature channel
- Effect: invariance to small translations of the input
- Max and average pooling most common, other things possible
- Parameter free layer
- Similar to strided convolution with special non-trainable filter




Slide from Xavier Alameda-Pineda

# Receptive field

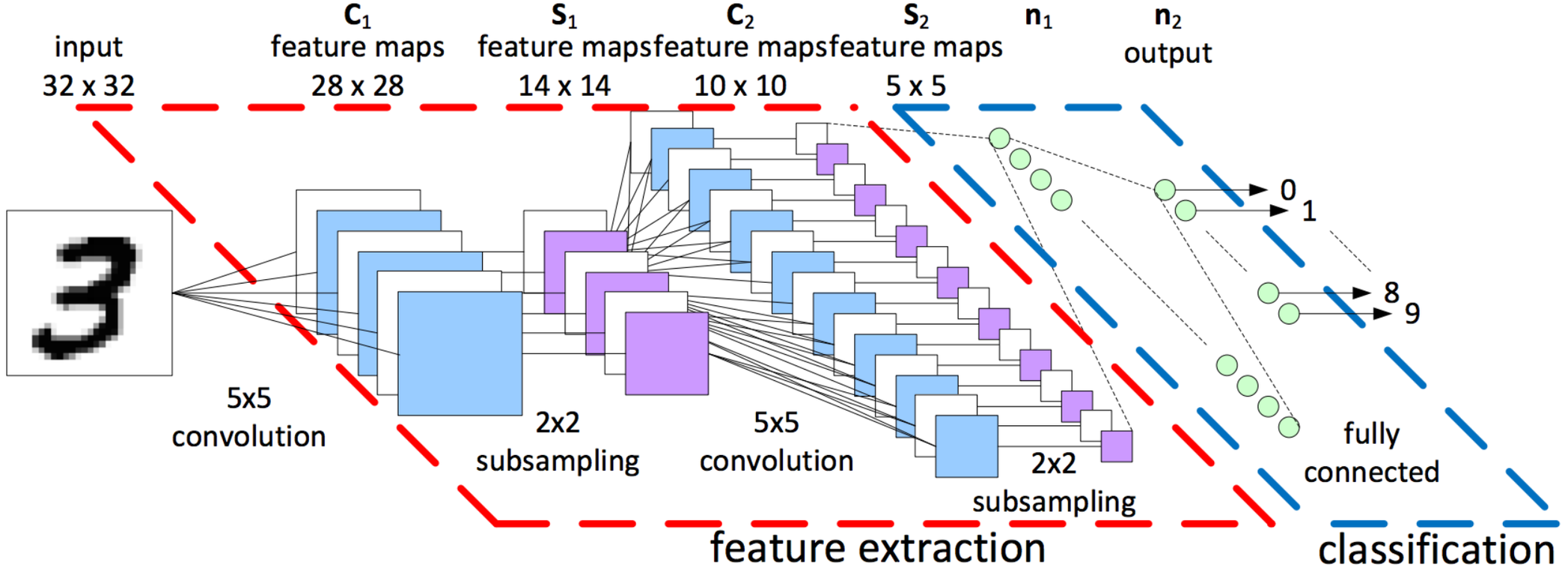
- Receptive field is area in original image impacting a certain unit.
  Later layers can capture more complex patterns over larger areas.
- Receptive field size grows linearly over convolutional layers.
  If we use a convolutional filter of size $w \times w$, then each layer the receptive field increases by $w - 1$.




Slide from Xavier Alameda-Pineda

# Fully connected layers

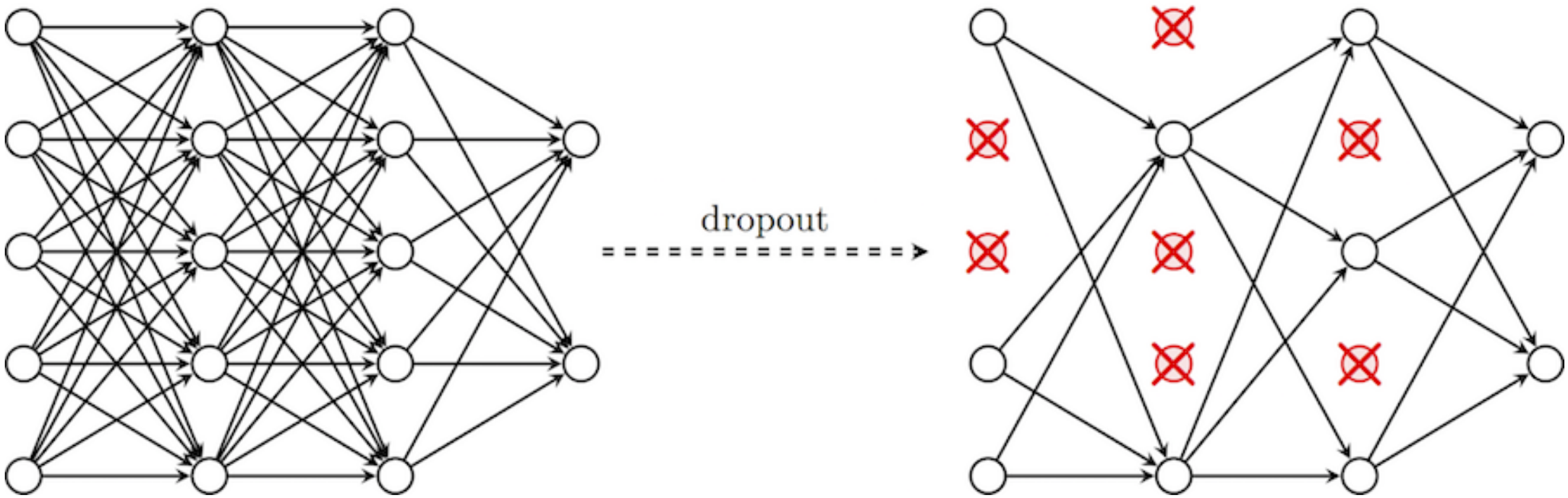
- Convolutional and pooling layers typically followed by several "fully connected" (FC) layers, i.e. a standard MLP
- FC layer connects all units in previous layer to all units in next layer
- Assembles all local information into global vectorial representation




Slide from Xavier Alameda-Pineda

# Drop-out regularisation

- Main idea: deactivate a subset of neurons, randomly selected at every batch during training.
- If forces the network to be redundant, hence robust.
- Different paths to recognise the same pattern.




Slide from Xavier Alameda-Pineda

# Batch normalisation

- Motivation: for very deep networks with ReLU, we quickly overflow (very large values).
- Main idea: the output of the linear combinations must follow a standard Gaussian distribution (zero mean, unit variance).
- Rationale: then all layers work at a similar regime.
- Usually after the fully connected or convolutional layers and before the non-linear activation.
- Compute the mean and variance for each neuron and:

$$x^{\text{new}} = \frac{x^{\text{old}} - \mu}{\sqrt{\nu}}$$

.

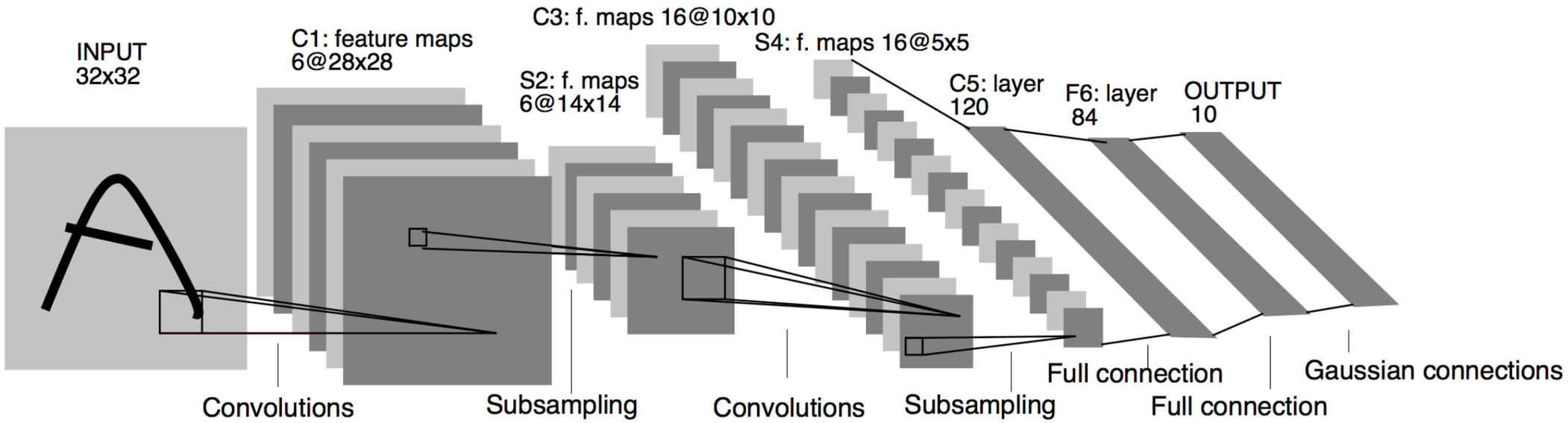
- Improves gradient flow and allows for larger learning rates.
- It is an unsupervised process that can take place at test time as well.

Slide from Xavier Alameda-Pineda

# CNN Architectures

Slide from Xavier Alameda-Pineda

# Lenet (1998)

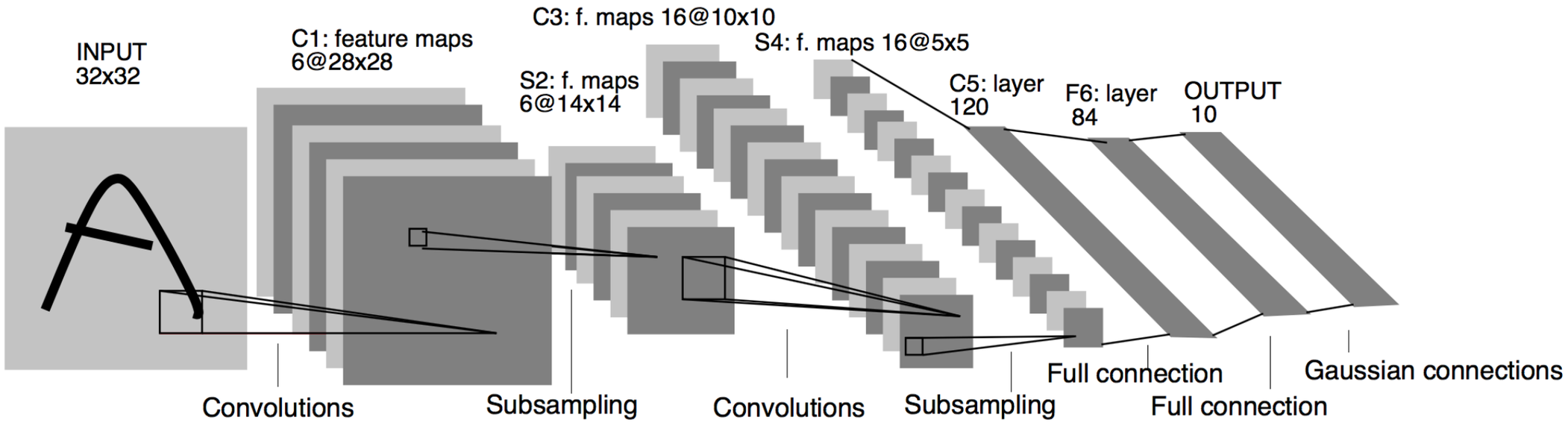



Let's compute the # parameters and activations (input image is $32 \times 32$):

- C1 (6 filters of $5 \times 5$):

Slide from Xavier Alameda-Pineda

# Lenet (1998)

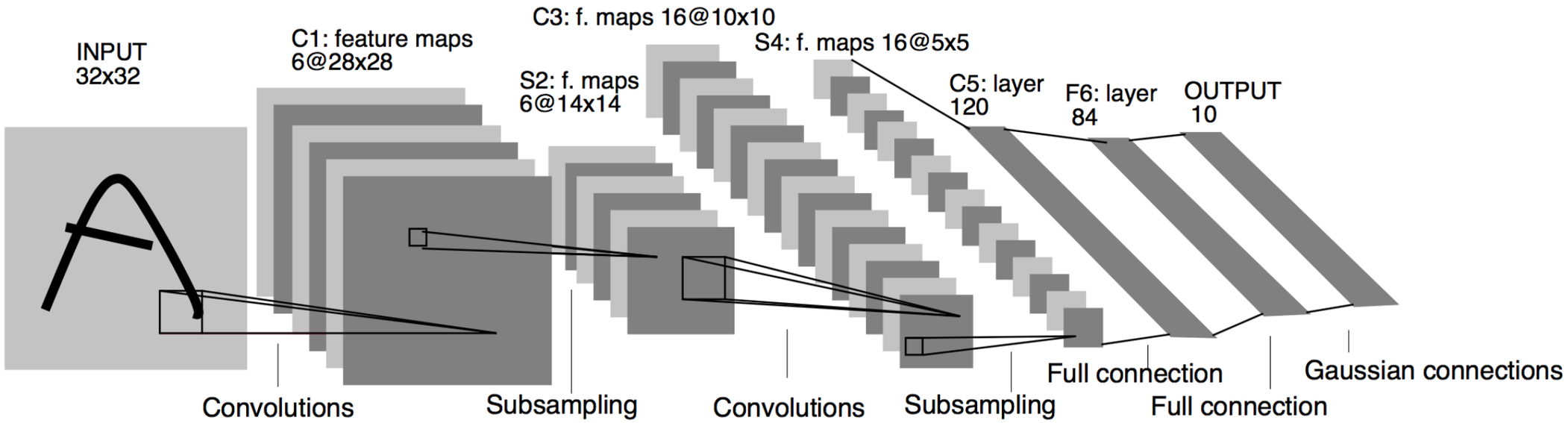



Let's compute the # parameters and activations (input image is $32 \times 32$):

- C1 (6 filters of $5 \times 5$): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling):

Slide from Xavier Alameda-Pineda

# Lenet (1998)

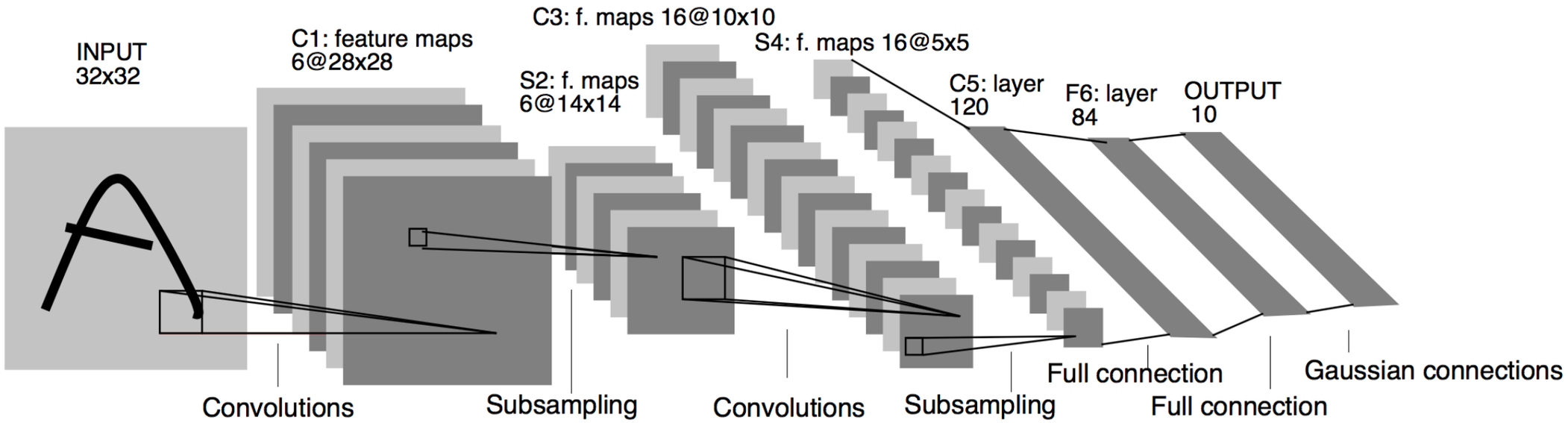



Let's compute the # parameters and activations (input image is $32 \times 32$):

- C1 (6 filters of $5 \times 5$): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling): #par : 0 #act: $14 \times 14 \times 6 = 1176$.
- C3 (16 filters of $5 \times 5$):

Slide from Xavier Alameda-Pineda

# Lenet (1998)

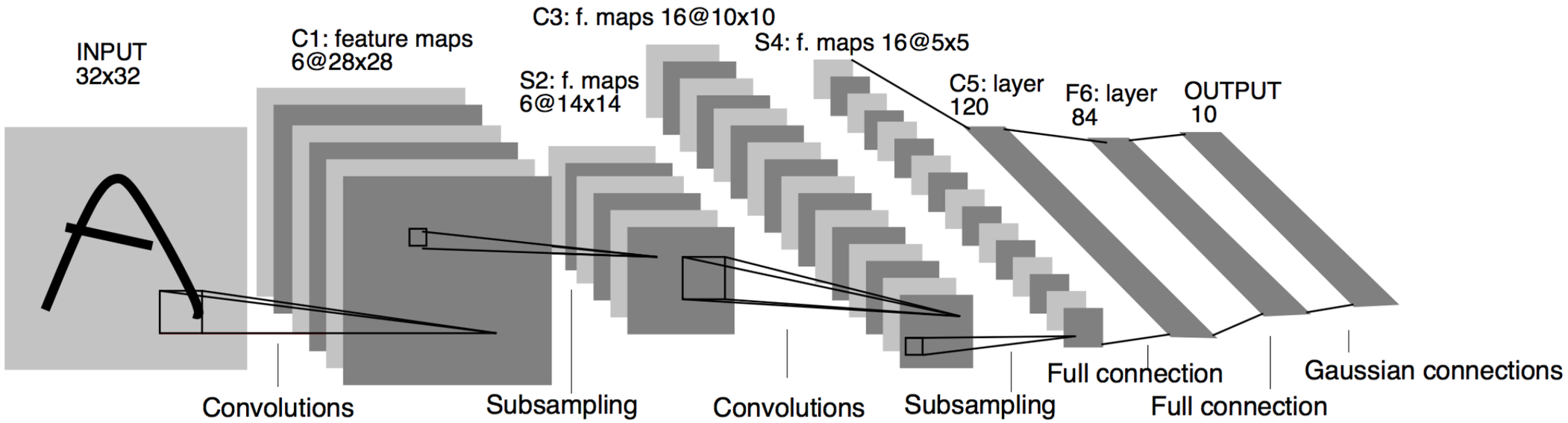



Let's compute the # parameters and activations (input image is $32 \times 32$):

- C1 (6 filters of $5 \times 5$): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling): #par : 0 #act: $14 \times 14 \times 6 = 1176$.
- C3 (16 filters of $5 \times 5$): #par: $(5 \times 5 + 1) \times 16 \times 6 = 2496$ #act: $10 \times 10 \times 16 = 1600$.
- S4 (subsampling):

Slide from Xavier Alameda-Pineda

# Lenet (1998)

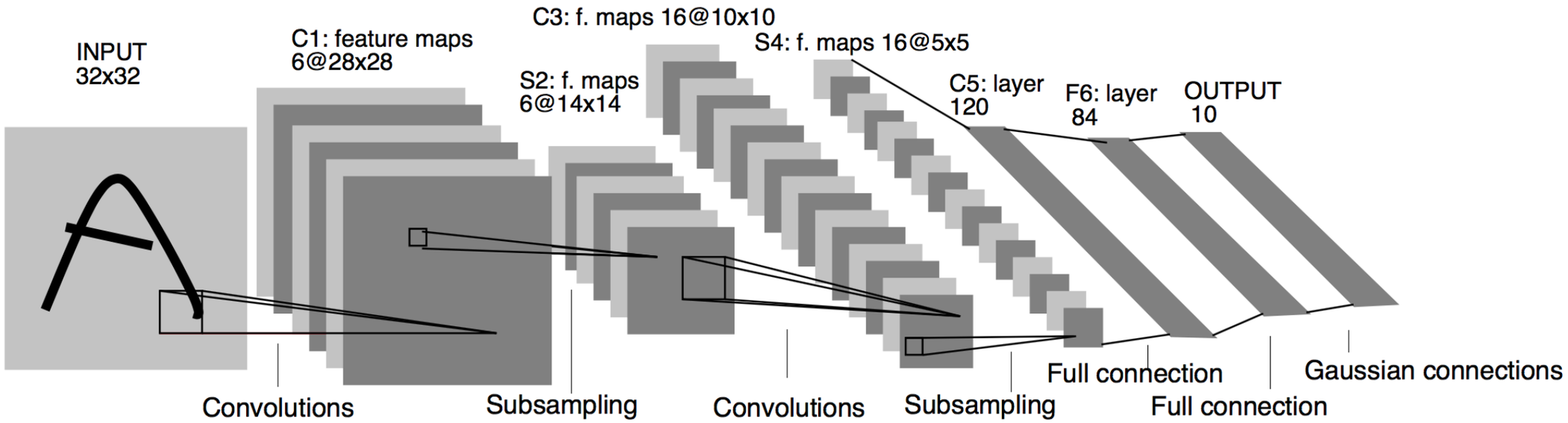



Let's compute the # parameters and activations (input image is $32 \times 32$):

- C1 (6 filters of $5 \times 5$): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling): #par : 0 #act: $14 \times 14 \times 6 = 1176$.
- C3 (16 filters of $5 \times 5$): #par: $(5 \times 5 + 1) \times 16 \times 6 = 2496$ #act: $10 \times 10 \times 16 = 1600$.
- S4 (subsampling): #par: 0 #act: $5 \times 5 \times 16 = 400$.
- C5 (FC output 120):

Slide from Xavier Alameda-Pineda

# Lenet (1998)

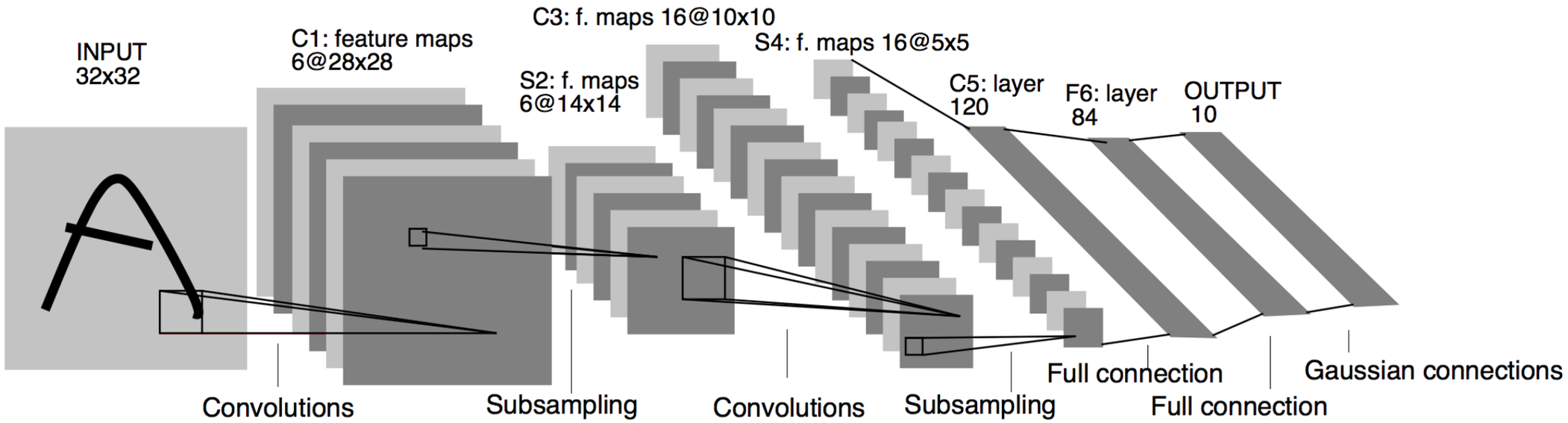



Let's compute the # parameters and activations (input image is $32 \times 32$):

- C1 (6 filters of $5 \times 5$): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling): #par : 0 #act: $14 \times 14 \times 6 = 1176$.
- C3 (16 filters of $5 \times 5$): #par: $(5 \times 5 + 1) \times 16 \times 6 = 2496$ #act: $10 \times 10 \times 16 = 1600$.
- S4 (subsampling): #par: 0 #act: $5 \times 5 \times 16 = 400$.
- C5 (FC output 120): #par: $(400 + 1) \times 120 = 48120$ #act: 120.
- F6 (FC output 84):

Slide from Xavier Alameda-Pineda

# Lenet (1998)

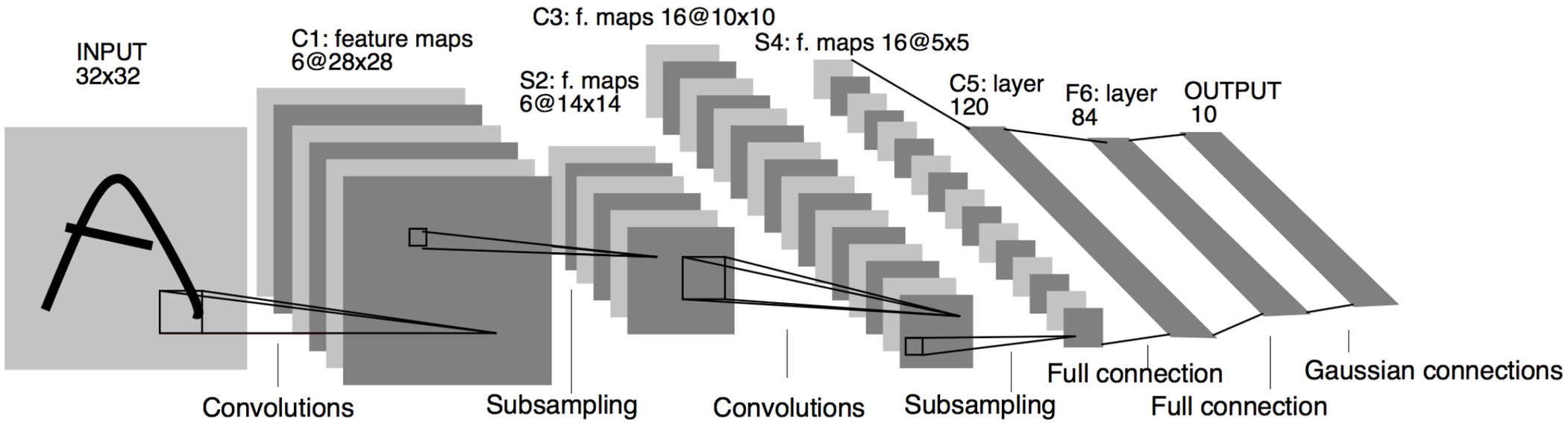



Let's compute the # parameters and activations (input image is $32 \times 32$):

- C1 (6 filters of $5 \times 5$): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling): #par : 0 #act: $14 \times 14 \times 6 = 1176$.
- C3 (16 filters of $5 \times 5$): #par: $(5 \times 5 + 1) \times 16 \times 6 = 2496$ #act: $10 \times 10 \times 16 = 1600$.
- S4 (subsampling): #par: 0 #act: $5 \times 5 \times 16 = 400$.
- C5 (FC output 120): #par: $(400 + 1) \times 120 = 48120$ #act: 120.
- F6 (FC output 84): #par: $120 * 84 = 10080$ #act: 84.
- Class (FC output 10):

Slide from Xavier Alameda-Pineda

# Lenet (1998)




Let's compute the # parameters and activations (input image is $32 \times 32$):

- C1 (6 filters of $5 \times 5$): #par: $(5 \times 5 + 1) \times 6 = 156$ #act: $28 \times 28 \times 6 = 4704$.
- S2 (subsampling): #par : 0 #act: $14 \times 14 \times 6 = 1176$.
- C3 (16 filters of $5 \times 5$): #par: $(5 \times 5 + 1) \times 16 \times 6 = 2496$ #act: $10 \times 10 \times 16 = 1600$.
- S4 (subsampling): #par: 0 #act: $5 \times 5 \times 16 = 400$.
- C5 (FC output 120): #par: $(400 + 1) \times 120 = 48120$ #act: 120.
- F6 (FC output 84): #par: $120 * 84 = 10080$ #act: 84.
- Class (FC output 10): #par: $84 * 10 = 840$ #act: 10.

Slide from Xavier Alameda-Pineda

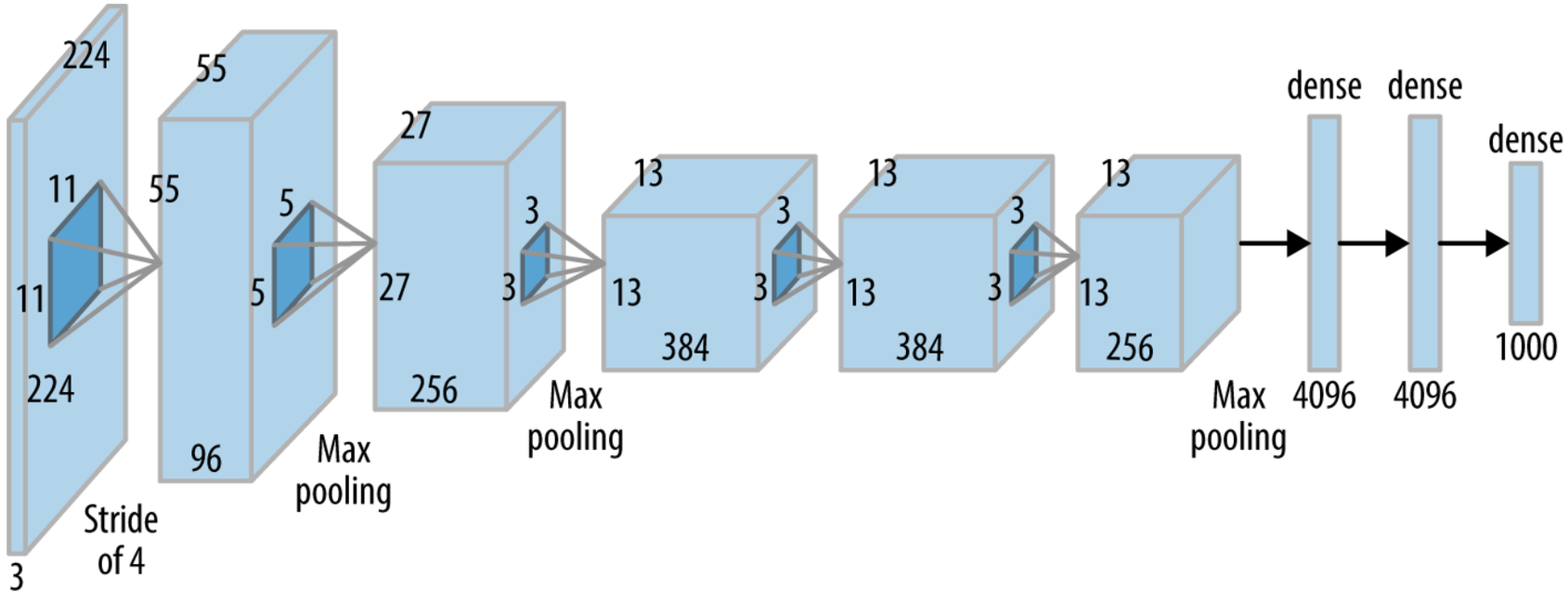
# What changed?

- Large training **data**sets for computer vision
  - 1.2 million images of 1000 classes in ImageNet challenge (2012)
  - 200 million faces to train face recognition nets (2015)
- **GPU**-based implementation: much faster than CPU
  - Parallel computation for matrix products
  - Krizhevsky & Hinton, 2012: six days on two GPUs (see next slide)
  - Rapid progress in GPU compute performance
- Network **architectures**
- Industrially backed **open-source** software (Pytorch, TensorFlow, etc)

Slide from Xavier Alameda-Pineda

# AlexNet CNN (2012)

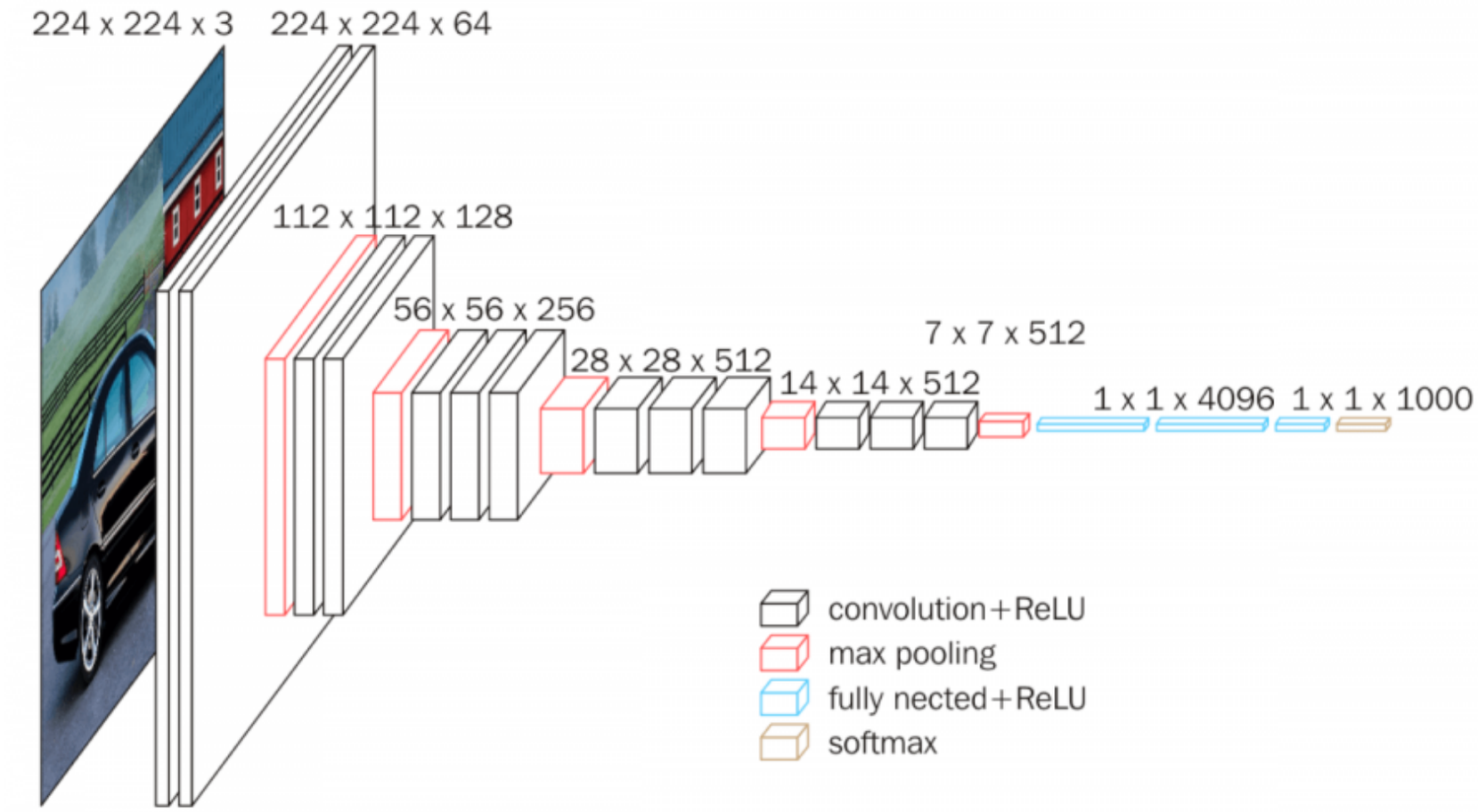
- Winner ImageNet 2012 image classification challenge, huge impact (+50k citations). CNNs improving "traditional" computer vision techniques.
- Compared to LeNet
  - Inputs at 224x224 rather than 32x32
  - 5 rather than 3 conv layers (more feature channels in each layer)
  - $\sim$ 60 million parameters
  - ReLU non-linearity



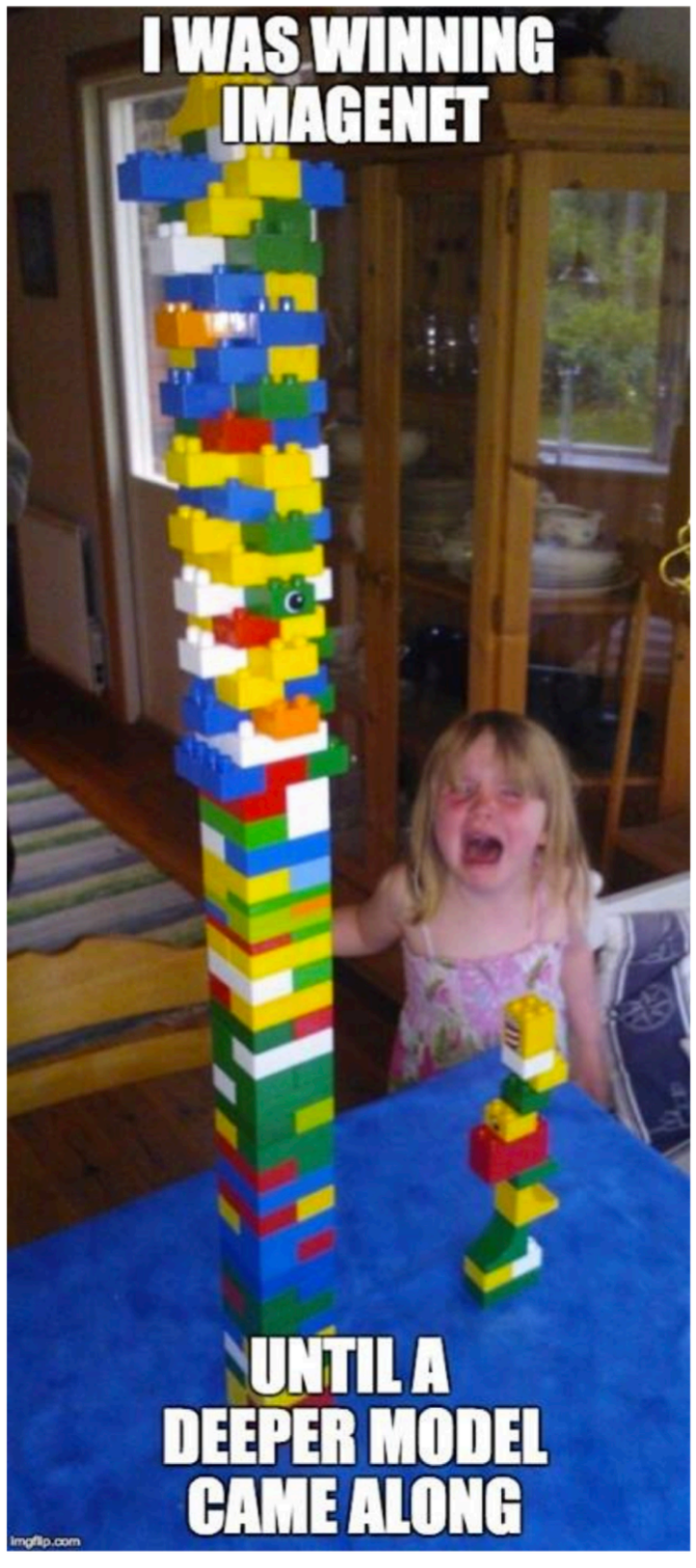

Slide from Xavier Alameda-Pineda

# VGG CNN (2015)

- Double the number of layers (up to 16/19).
- Only small $3 \times 3$ filters (rather than 11 in AlexNet). Same receptive field, less parameters per layer, better learned. About 140 million parameters.



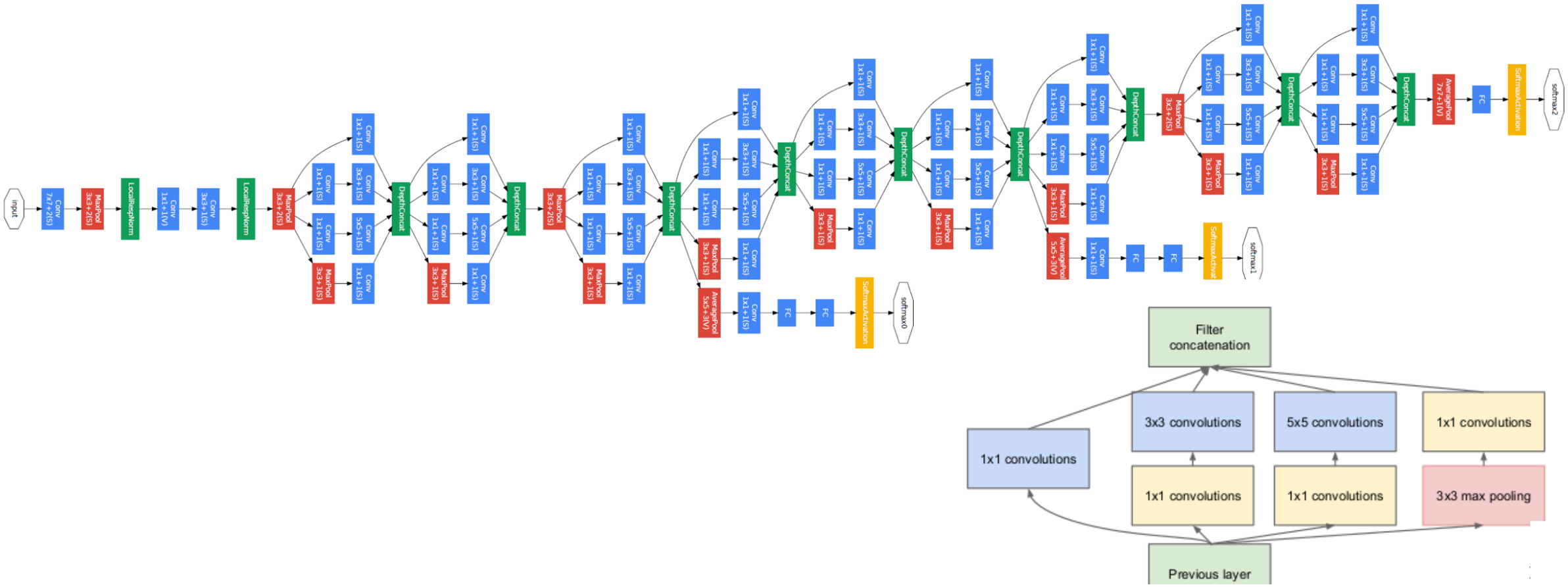

Slide from Xavier Alameda-Pineda


I WAS WINNING
IMAGENET
UNTIL A
DEEPER MODEL
CAME ALONG
imgflip.com


Inception, Christopher Nolan, 2010

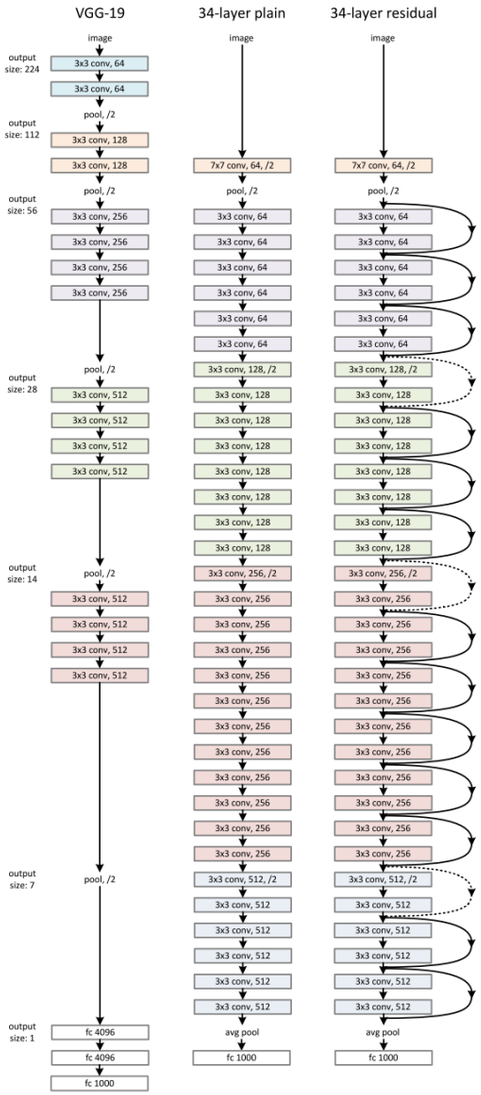
# GoogleNet Inception CNN (2015)

- Reduced number of parameters (5 million) but more layers (27 or 48).
- Inception module to compress features before convolution.
- Replaces fully-connected with average pooling.
- Intermediate loss functions to improve training.

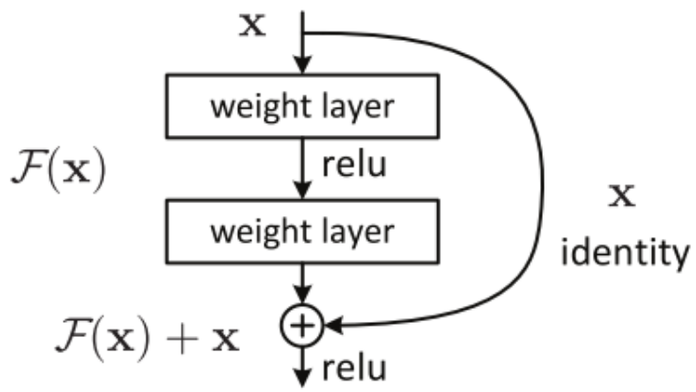



Slide from Xavier Alameda-Pineda

# Res(idual)Net CNN (2015)



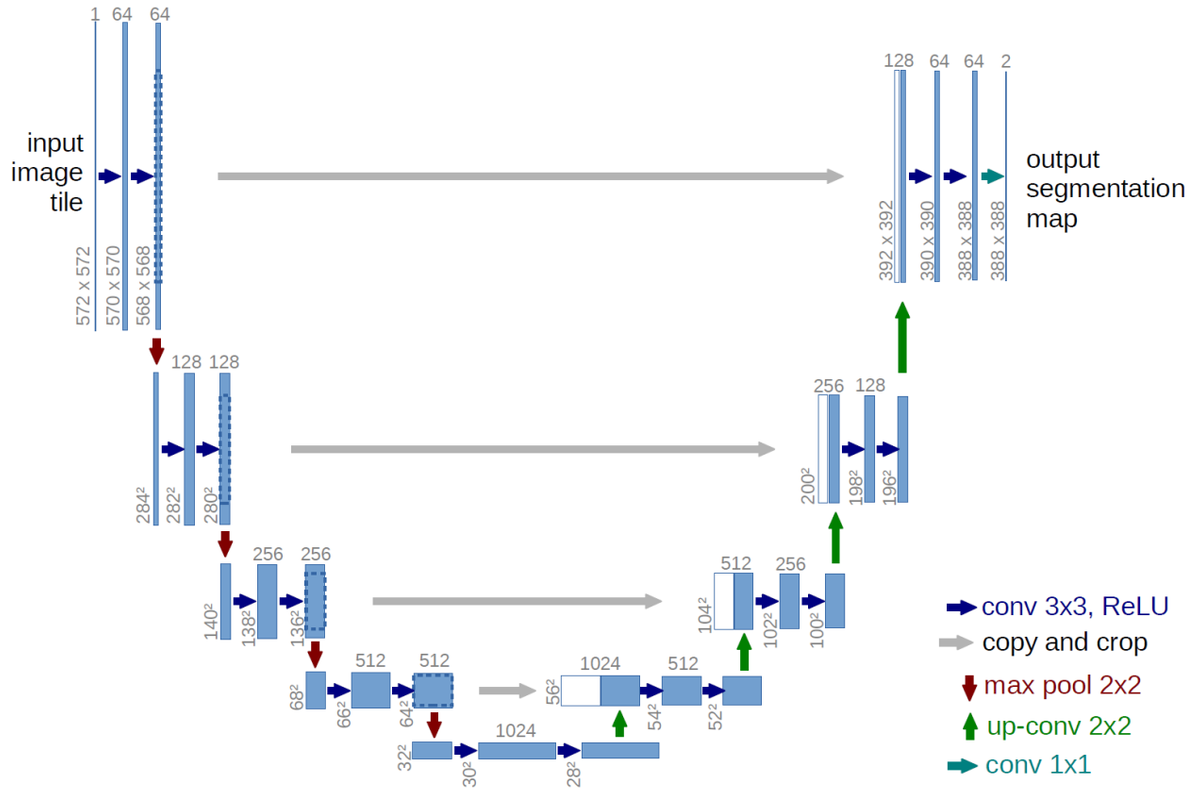

- Many more layers (34, 50, 110, 1200), multi-GPU training is required.
- Residual module to ensure gradient flow.
- Residual block does not require intermediate losses.




Slide from Xavier Alameda-Pineda

# U-Net (2015)

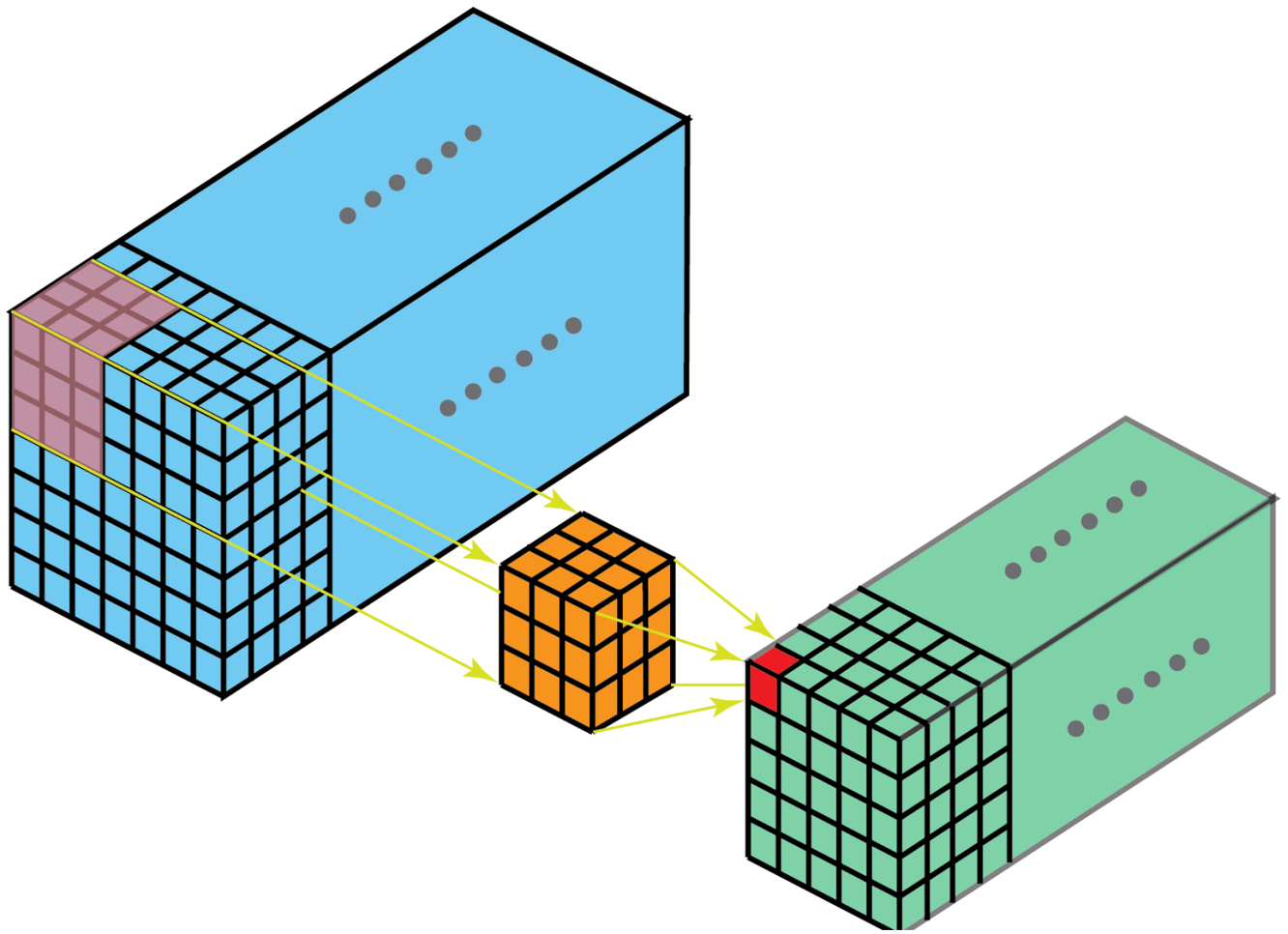
- Convolution/deconvolution architecture for tissue segmentation.
- Convolutions downsample the feature maps (and increase the # channels).
- Deconvolutions restore high-resolution image.
- Skip connections allow to transfer information from intermediate representations to the deconvolutions.




Slide from Xavier Alameda-Pineda

# C3D or 3D ConvNets (2015)

- Consider convolutional filters with less channels than the input.
- The kernel can move also along channels, and convolve the input.
- Can be used to process video frames (contactenated in a cube).



Slide from Xavier Alameda-Pineda

# Finetunning pre-trained CNNs

# Finetunning pre-trained CNNs

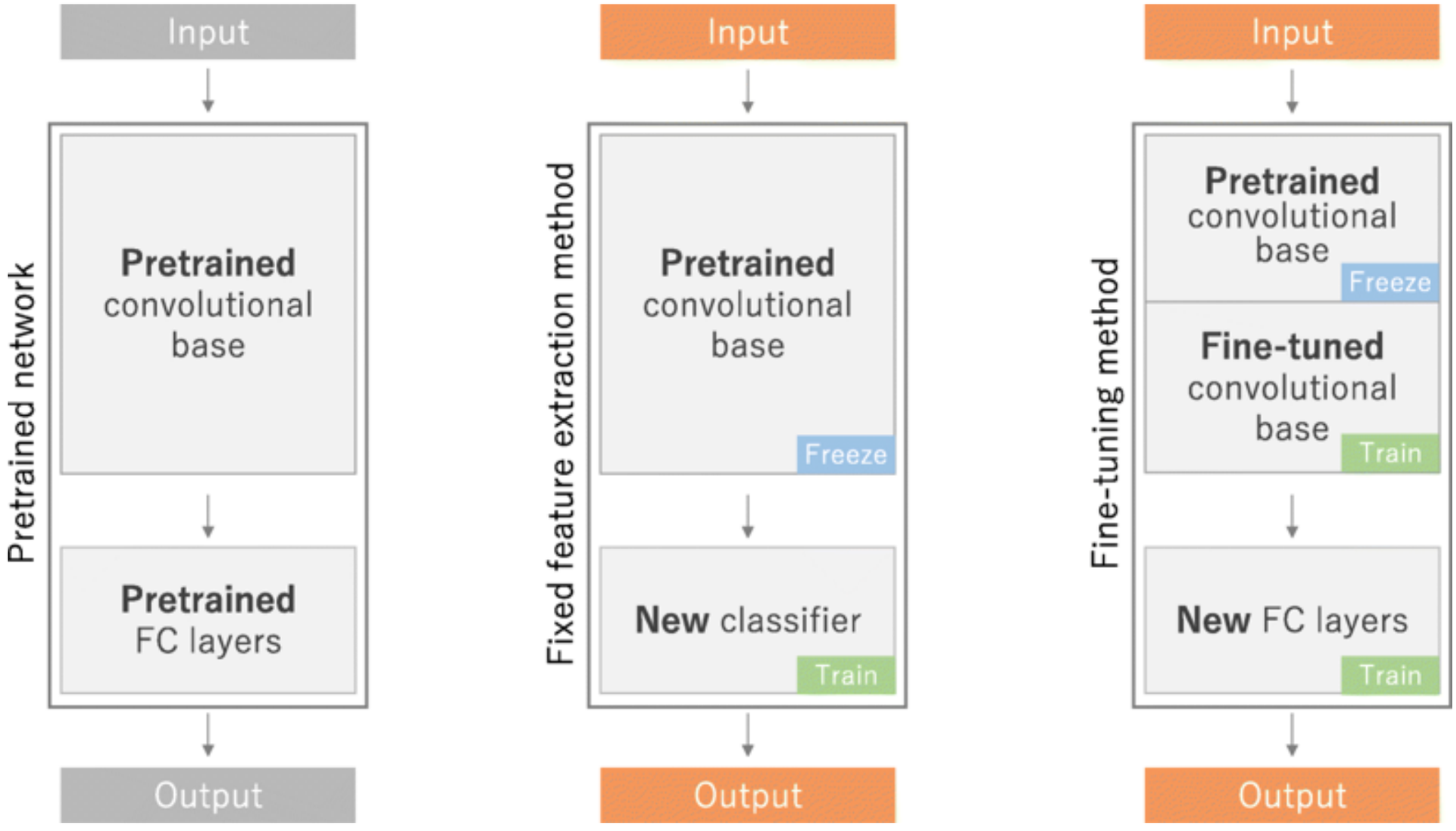
- Early CNN layers extract generic features that seem useful for different tasks.
  Object localization, semantic segmentation, action recognition, etc.

- On some datasets too little training data to learn CNN from scratch.
  For example, only few hundred objects bounding box to learn from.

- Pre-train AlexNet/VGGnet/ResNet/DenseNet on large scale dataset.
  In practice mostly ImageNet classification: millions of labeled images.
- Fine-tune CNN weights for task at hand, perhaps modifying the architecture.
  - Replace classification layer, add bounding box regression, ...
  - Reduced learning rate and possibly freezing early network layers

Slide from Xavier Alameda-Pineda

# Finetunning pre-trained CNNs (II)




From Yamashita et al "Convolutional neural networks: an overview and application in radiology" Insights into Imaging, 2018.

Slide from Xavier Alameda-Pineda

# Fine-tunning example

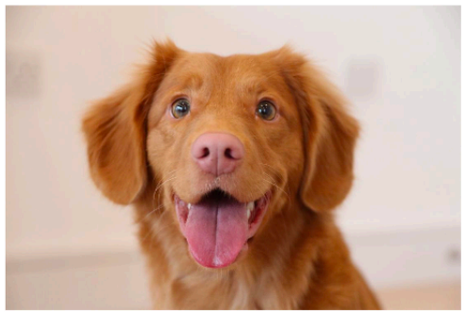
Human body pose regression

- Remove the last layers of the network.
- Add new layers regression the limbs position.
- Change the loss e.g. Euclidean distance.
- Fine-tune previously trained layers and train the new ones from scratch.

Slide from Xavier Alameda-Pineda

# Overfit Problem

Plane



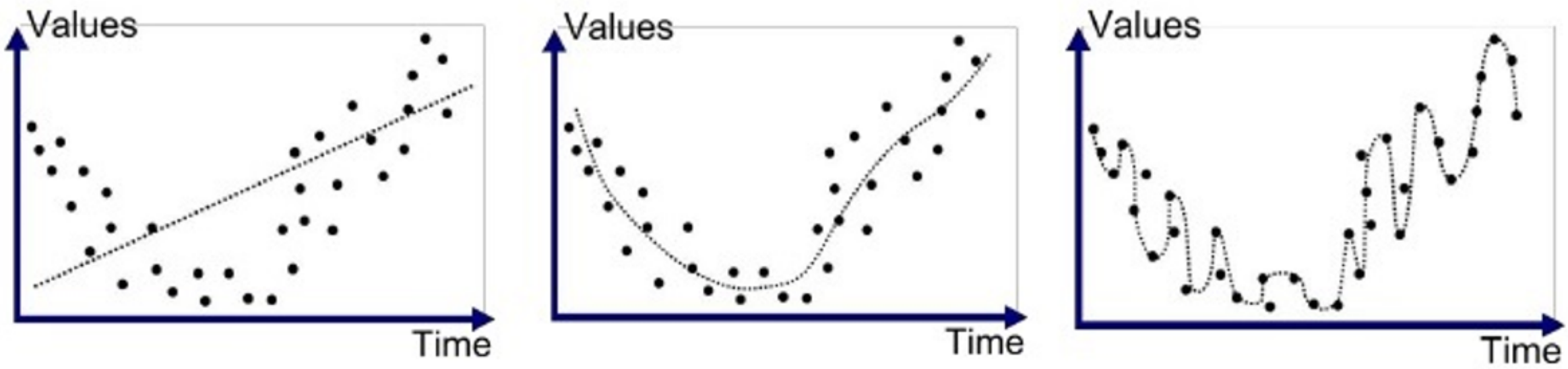
Dog

Table

Cat

Plane!

# Overfit Problem




Underfitted

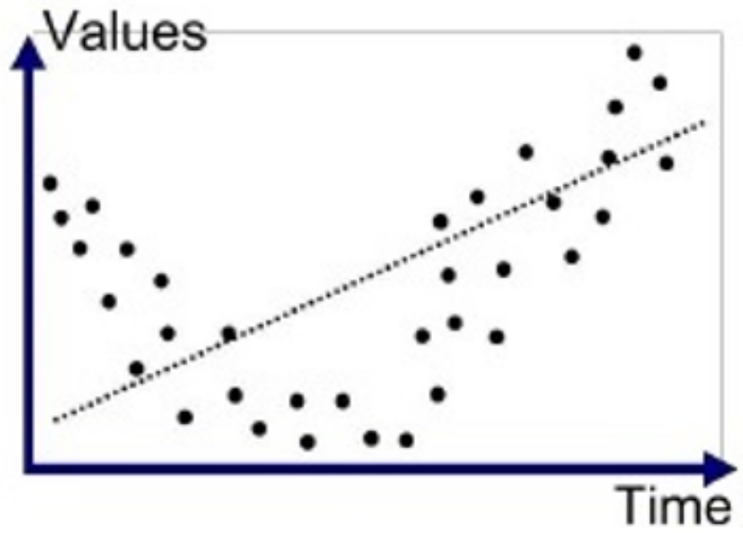
Good Fit/Robust

Overfitted

Images from “Machine Learning: How to Prevent Overfitting”
https://medium.com/swlh/machine-learning-how-to-prevent-overfitting-fdf759cc00a9

# Overfit Problem

The bias - variance tradeoff




Underfitted

Model is too simple, not “Expressive” enough

Accuracy can still be > 0

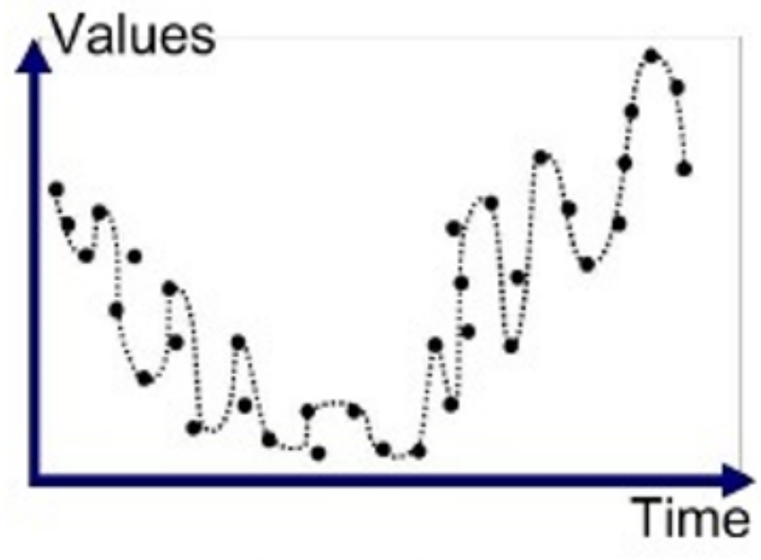
“A broken clock is right twice a day”

High bias - low variance

Images from “Machine Learning: How to Prevent Overfitting”
https://medium.com/swlh/machine-learning-how-to-prevent-overfitting-fdf759cc00a9

# Overfit Problem

The bias - variance tradeoff




Overfitted

Model is too complex, too “Expressive”

Very “accurate”

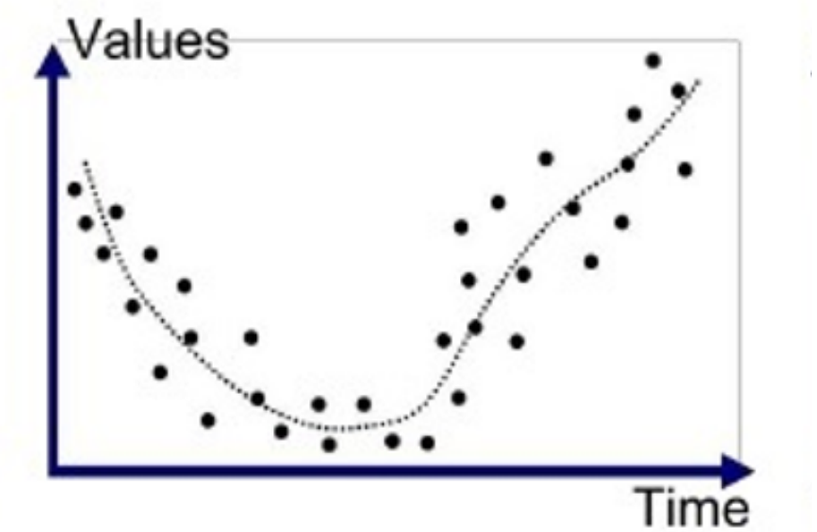
Does not generalize - useless!

Low bias - high variance

Images from “Machine Learning: How to Prevent Overfitting”
https://medium.com/swlh/machine-learning-how-to-prevent-overfitting-fdf759cc00a9

# Overfit Problem

The bias - variance tradeoff




Good Fit/Robust

Tradeoff between bias and variance?

Relatively accurate

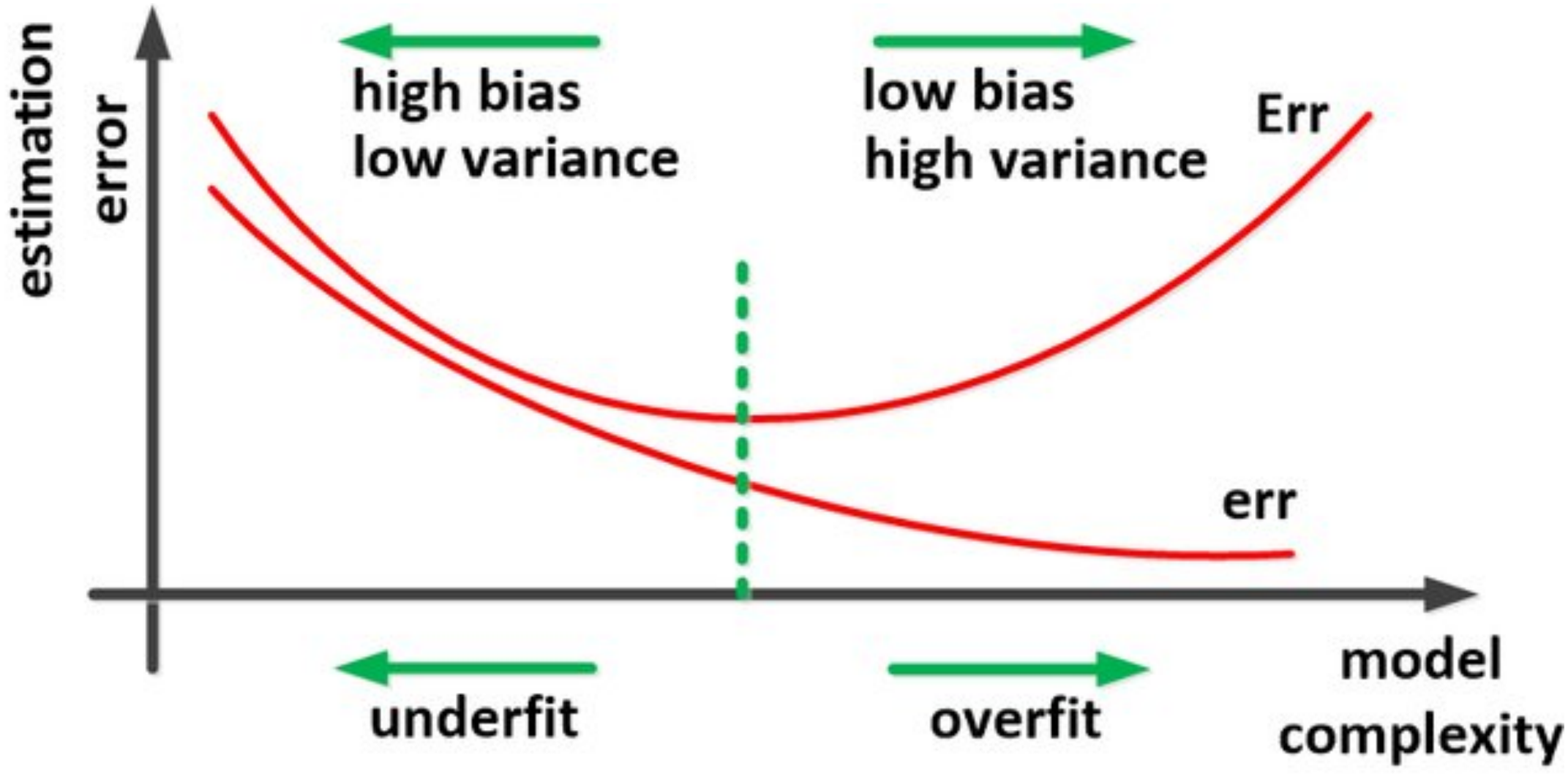
Relatively good generalization properties

moderate bias - moderate variance

Images from “Machine Learning: How to Prevent Overfitting”
https://medium.com/swlh/machine-learning-how-to-prevent-overfitting-fdf759cc00a9

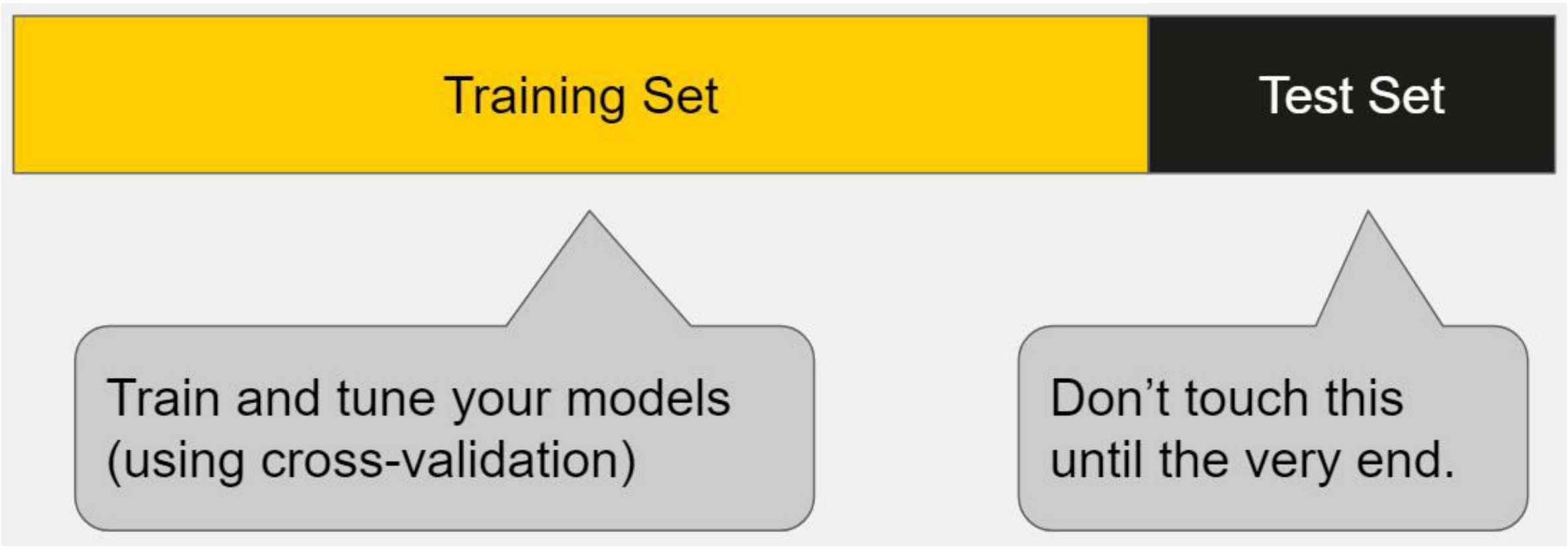
# Overfit Problem

The bias - variance tradeoff




Ghojogh, Benyamin, and Mark Crowley. "The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial." *arXiv preprint arXiv:1905.12787* (2019).

# Avoiding Overfit

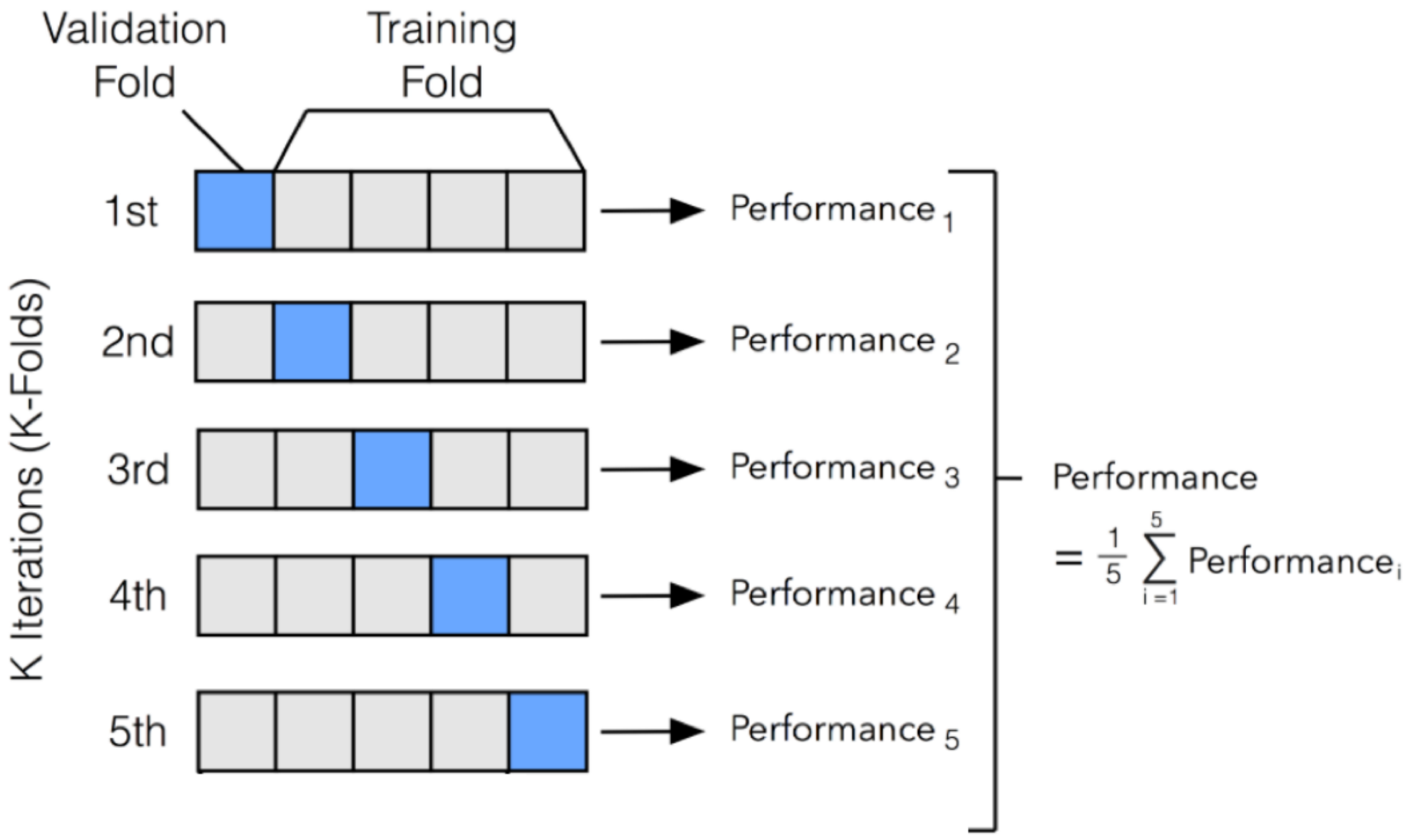



Images from “Machine Learning: How to Prevent Overfitting”
https://medium.com/swlh/machine-learning-how-to-prevent-overfitting-fdf759cc00a9

# Avoiding Overfit

Validation set - super important!

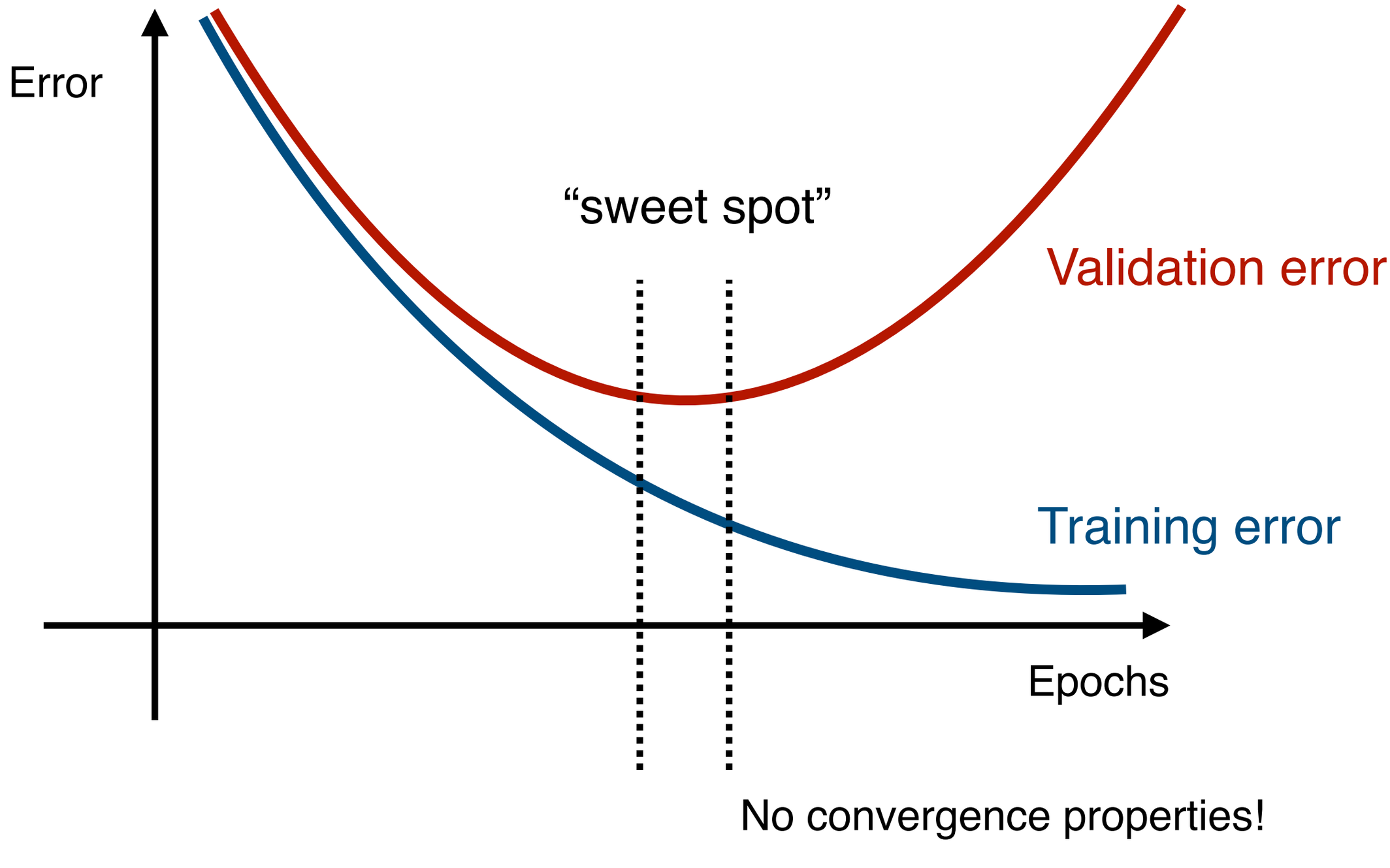
Cross-validation (if possible)




Images from “Machine Learning: How to Prevent Overfitting”
https://medium.com/swlh/machine-learning-how-to-prevent-overfitting-fdf759cc00a9

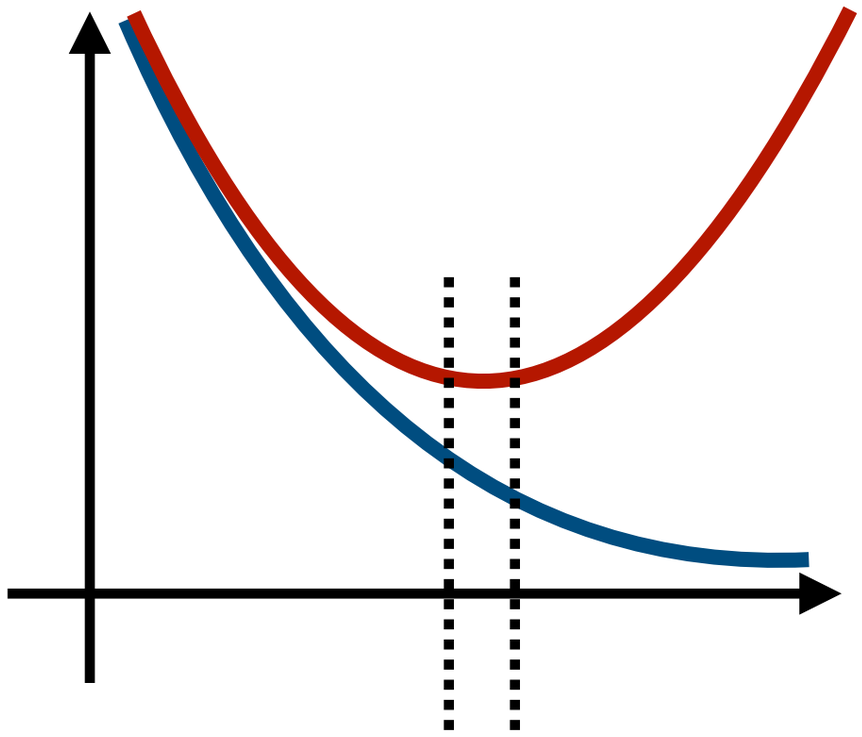
# Avoiding Overfit

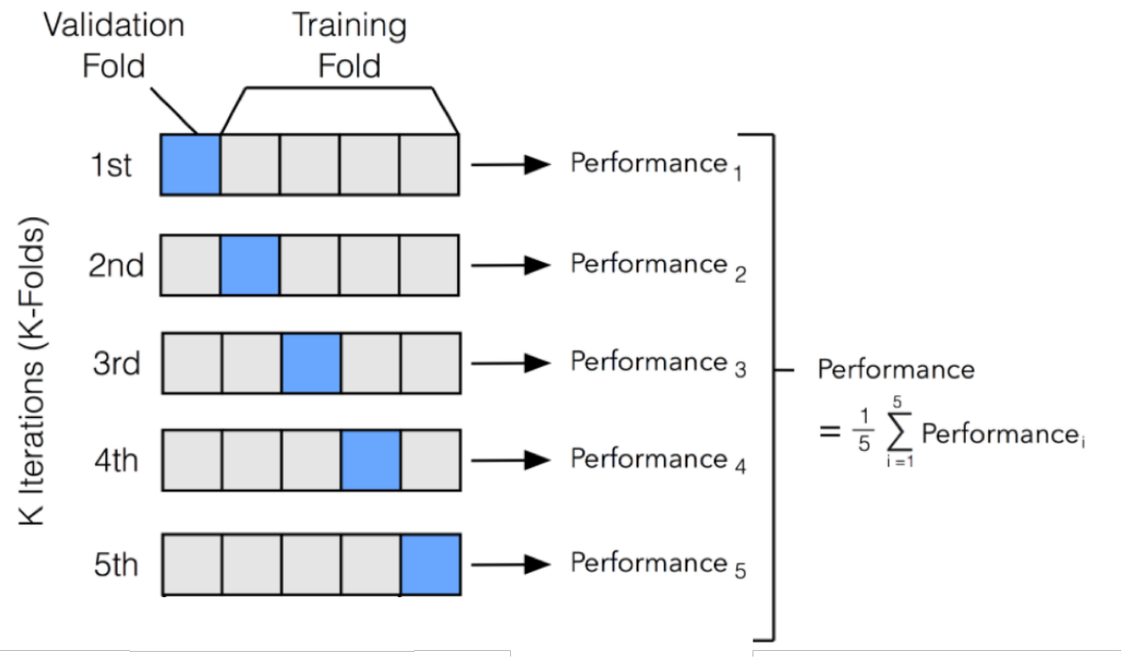
Early stop




No convergence properties!

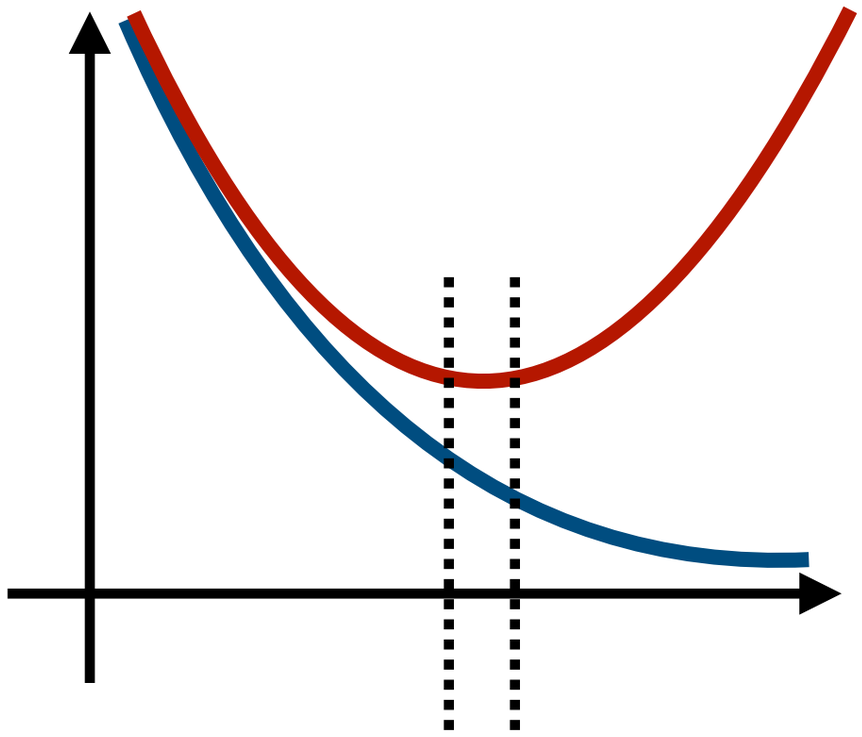
# A note on accuracy

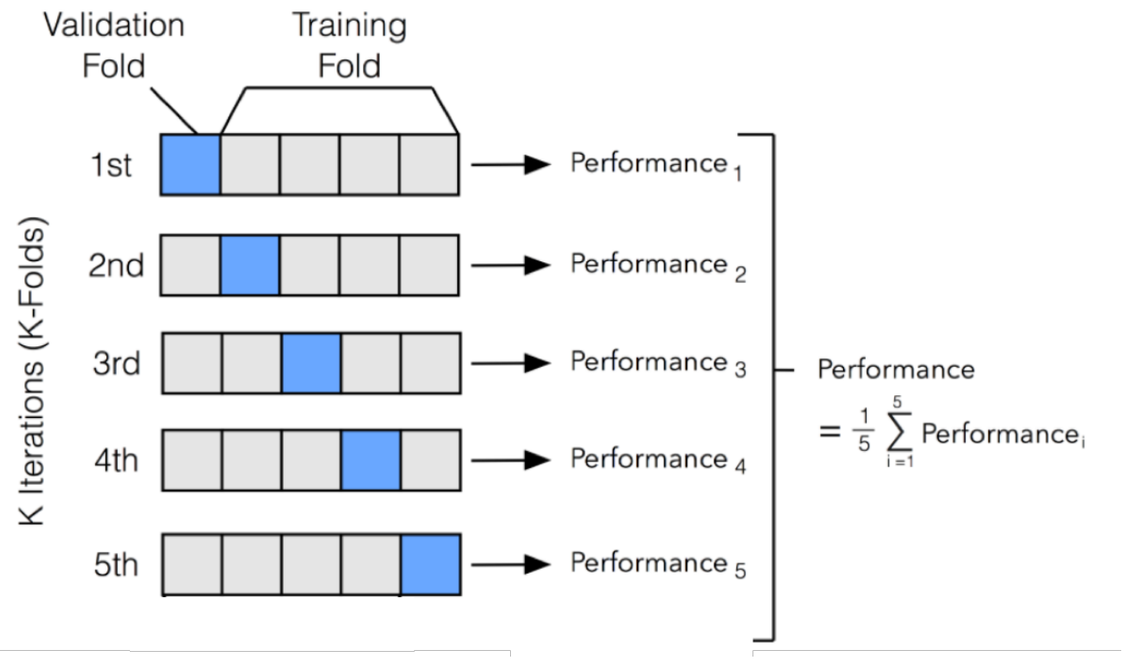





The same method will have different results in different folds
-> what do you think is happening?

# A note on accuracy

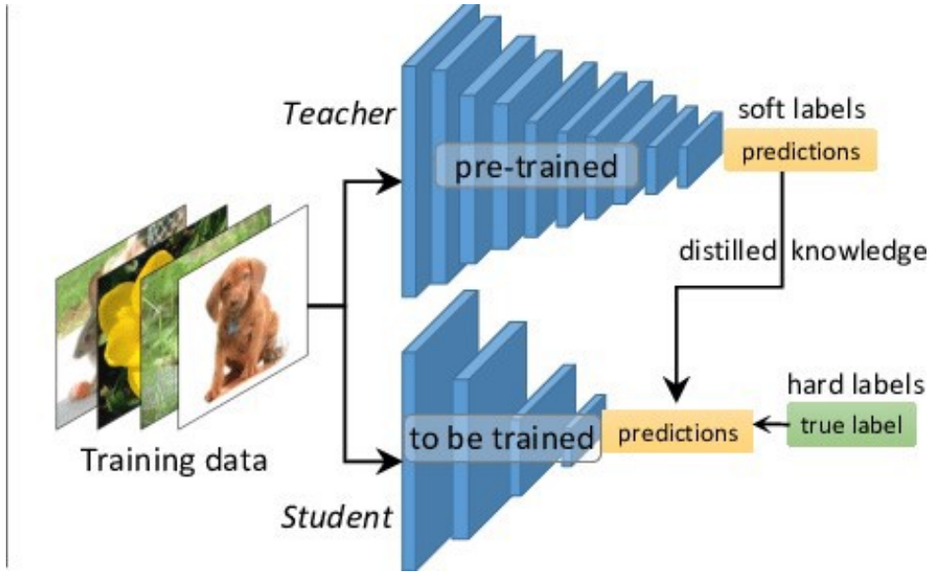


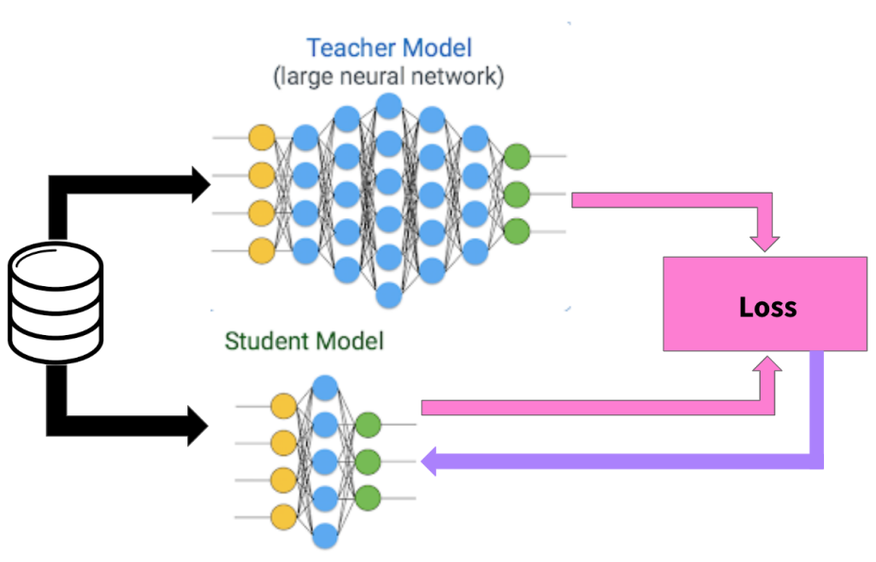



Evaluate on easy “naive” baselines (underfit)

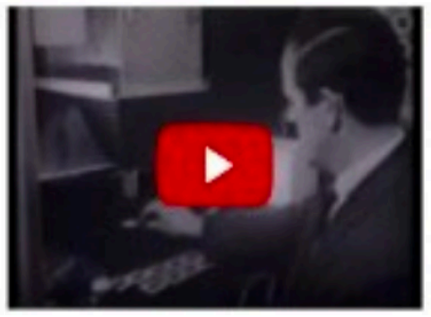
# Distillation [Hinton et al., 2015]

- A teacher (large) and a student (small) network.
- Teacher is pre-trained.
- The student is trained to imitate the output of the teacher network (before soft-max).
- Training the student directly does not work!




Slide from Xavier Alameda-Pineda



**1958** Perceptron

**1974** Backpropagation

Convolution Neural Networks for Handwritten Recognition

**1998**

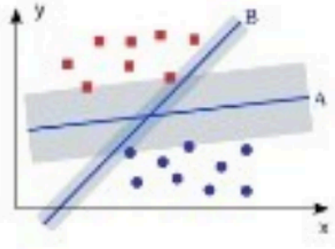
Google Brain Project on 16k Cores

**2012**

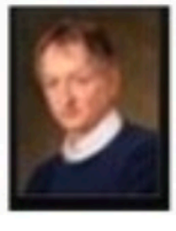
awkward silence (AI Winter)

**1969**
Perceptron criticized

**1995**
SVM reigns




**2006**
Restricted Boltzmann Machine



**2012**
AlexNet wins ImageNet

IMAGENET

Image from `https://www.slideshare.net/LuMa921/deep-learning-a-visual-introduction`.

# References


- A Gentle Introduction to the Rectified Linear Unit (ReLU)
  https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
  (last accessed 1 Oct 2021)

- Unit on Deep Learning and Differentiable Programming
  https://perso.liris.cnrs.fr/christian.wolf/teaching/index.html
  (last accessed 1 Oct 2021)

- Unit on Deep Learning UNIGE/EPFL
  https://fleuret.org/dlc/
  (last accessed 1 Oct 2021)

- Unit on Deep Learning - Xavier Alameda-Pineda
- http://thoth.inrialpes.fr/people/alahari/teaching/clor19-20/8-DeepLearning.pdf