

# Neural Network - Foundations

CORS 2021 Workshop

June 8, 2021

Maxime Gasse, Polytechnique Montréal



CORS · SCRO

## In this talk

A bit of theory

- ▶ decision theory
- ▶ empirical risk minimization
- ▶ statistical learning (capacity, overfitting, underfitting)

Neural networks, with a bit of history

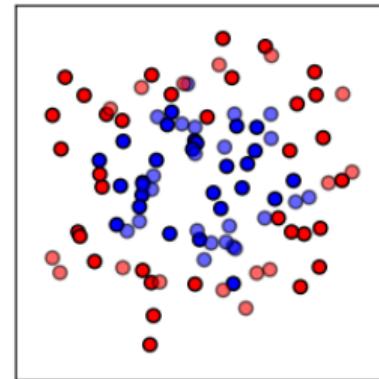
- ▶ Perceptron Era (1958-60s)
- ▶ Backpropagation Era (1986-90s)
- ▶ Deep Learning Era (2006-)

## Supervised learning

We have a data set  $\mathcal{D} \sim p$  of previously observed  $(x, y)$  pairs.

### Example

- ▶ customer bank records  $\rightarrow$  solvency
- ▶ patient symptoms  $\rightarrow$  disease
- ▶ apartment address, surface, year  $\rightarrow$  price
- ▶ mushroom picture  $\rightarrow$  variety

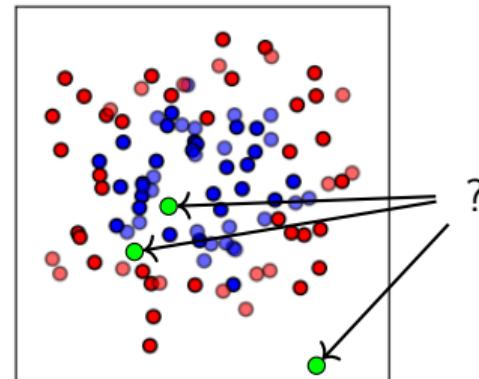


## Supervised learning

We have a data set  $\mathcal{D} \sim p$  of previously observed  $(x, y)$  pairs.

### Example

- ▶ customer bank records  $\rightarrow$  solvency
- ▶ patient symptoms  $\rightarrow$  disease
- ▶ apartment address, surface, year  $\rightarrow$  price
- ▶ mushroom picture  $\rightarrow$  variety



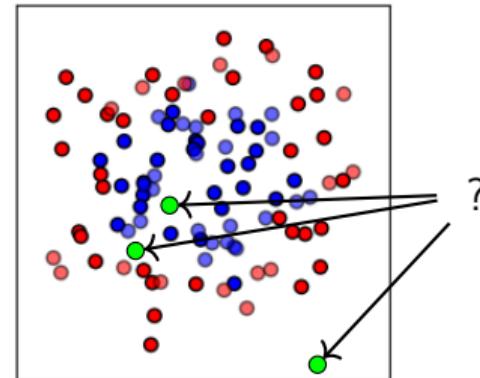
Objective: for any new observation  $x$ , make a prediction  $\hat{y}$ .

## Supervised learning

We have a data set  $\mathcal{D} \sim p$  of previously observed  $(x, y)$  pairs.

### Example

- ▶ customer bank records  $\rightarrow$  solvency
- ▶ patient symptoms  $\rightarrow$  disease
- ▶ apartment address, surface, year  $\rightarrow$  price
- ▶ mushroom picture  $\rightarrow$  variety



Objective: for any new observation  $x$ , make a prediction  $\hat{y}$ .

Find a **mapping**  $h$  from an **input space**  $\mathcal{X}$  to an **output space**  $\mathcal{Y}$ ,

$$h : \mathcal{X} \rightarrow \mathcal{Y}.$$

## Decision theory

Cost of  $\hat{y}$  instead of  $y$  ? **Loss function**

$$L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}.$$

## Decision theory

Cost of  $\hat{y}$  instead of  $y$  ? **Loss function**

$$L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}.$$

Risk of  $\hat{y}$  ? **Expected cost**

$$\mathbb{E}_{y|x}[L(\hat{y}, y)] = \sum_y p(y | x) \times L(\hat{y}, y) \quad (\text{classification}).$$

## Decision theory

Cost of  $\hat{y}$  instead of  $y$  ? **Loss function**

$$L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}.$$

Risk of  $\hat{y}$  ? **Expected cost**

$$\mathbb{E}_{y|x}[L(\hat{y}, y)] = \sum_y p(y | x) \times L(\hat{y}, y) \quad (\text{classification}).$$

Optimal mapping ? Risk-minimizing prediction

$$h_L^*(x) = \arg \min_{\hat{y}} \mathbb{E}_{y|x} L(\hat{y}, y).$$

## Example: binary classification

Loss function:

		$\hat{y}$
		0    1
$y$	0	0    2
	1	5    0

Expected loss:

$$\mathbb{E}_{y|x}[L(1, y)] = p(y = 0|x) \times 2$$

$$\mathbb{E}_{y|x}[L(0, y)] = p(y = 1|x) \times 5$$

Prediction rule:

$$h_L^*(x) = [p(y = 1|x) > 28.6\%].$$

All we need is to estimate  $p(y|x)$ .

## Plug-in risk minimization

$\mathcal{D}$  a set of i.i.d. data samples  $\{(x, y)^{(i)}\}_{i=1}^N$  drawn from  $p(x, y)$ ,  
 $\mathcal{Q}$  a restricted **probabilistic model** (e.g. parametric distribution).

## Plug-in risk minimization

$\mathcal{D}$  a set of i.i.d. data samples  $\{(x, y)^{(i)}\}_{i=1}^N$  drawn from  $p(x, y)$ ,

$\mathcal{Q}$  a restricted **probabilistic model** (e.g. parametric distribution).

Learning: empirical likelihood maximization given  $\mathcal{D}$

$$q^* = \arg \max_{q \in \mathcal{Q}} \sum_{(x,y) \in \mathcal{D}} \log q(y|x).$$

## Plug-in risk minimization

$\mathcal{D}$  a set of i.i.d. data samples  $\{(x, y)^{(i)}\}_{i=1}^N$  drawn from  $p(x, y)$ ,  
 $\mathcal{Q}$  a restricted **probabilistic model** (e.g. parametric distribution).

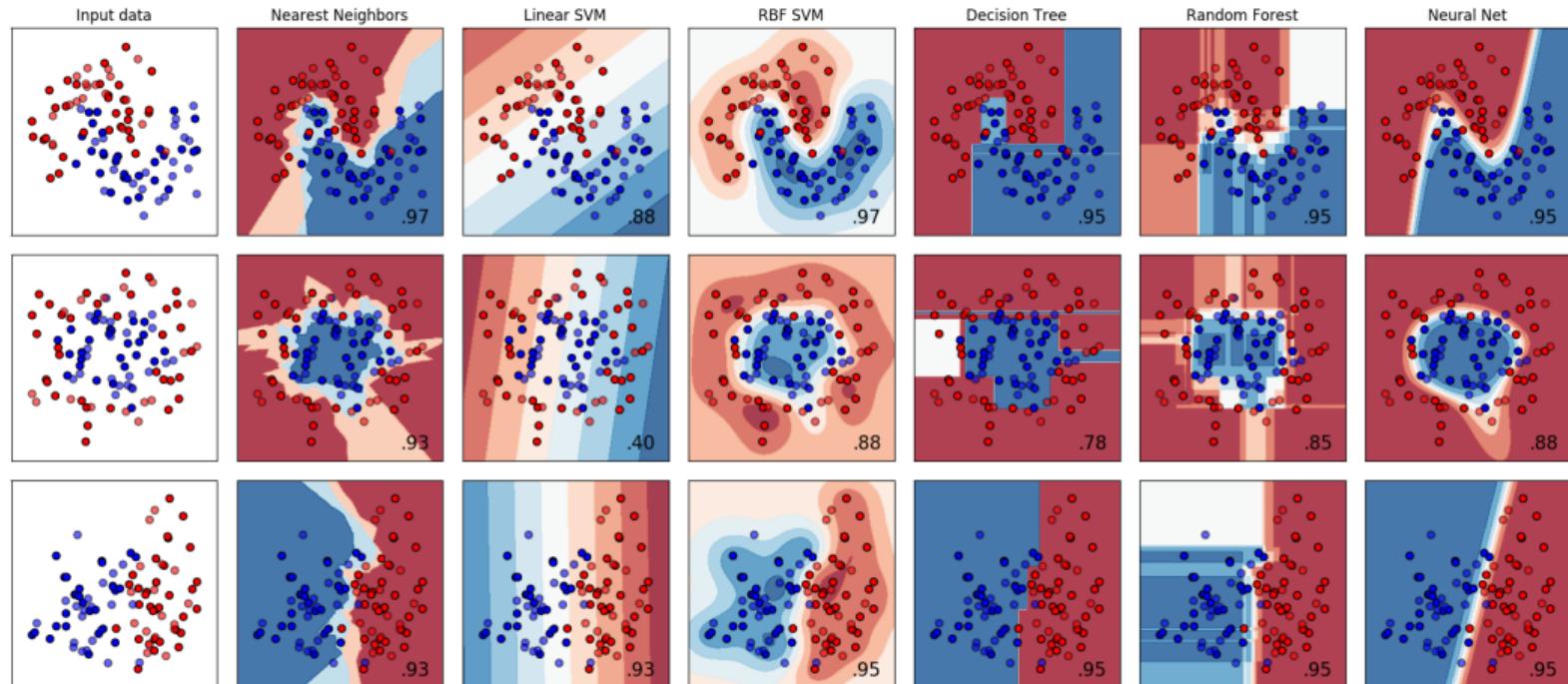
Learning: empirical likelihood maximization given  $\mathcal{D}$

$$q^* = \arg \max_{q \in \mathcal{Q}} \sum_{(x,y) \in \mathcal{D}} \log q(y|x).$$

Inference: risk minimization given  $q^*$

$$h_L^*(x) = \arg \min_{\hat{y}} \sum_y q^*(y|x) L(\hat{y}, y)$$

## Decision boundaries

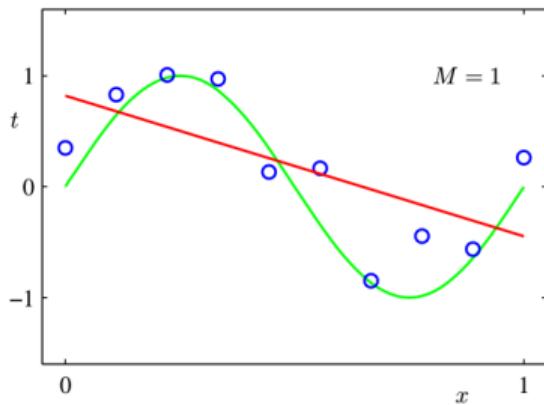
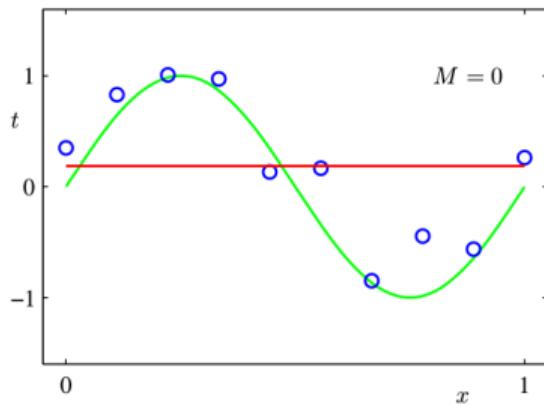


# Capacity, overfitting, underfitting

The higher the capacity of  $\mathcal{Q}$ , the more powerful the model.

Example (polynomial regression with degree  $M$ )

$M$  increases  
 $N = 10$  (fixed)

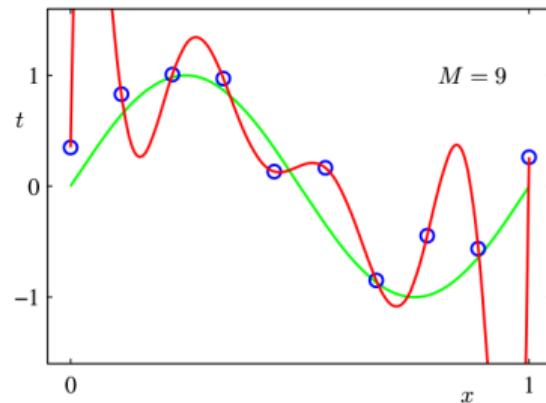
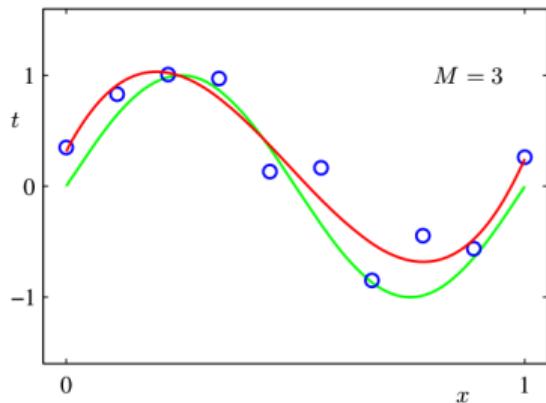


# Capacity, overfitting, underfitting

The higher the capacity of  $\mathcal{Q}$ , the more powerful the model.

Example (polynomial regression with degree  $M$ )

$M$  increases  
 $N = 10$  (fixed)

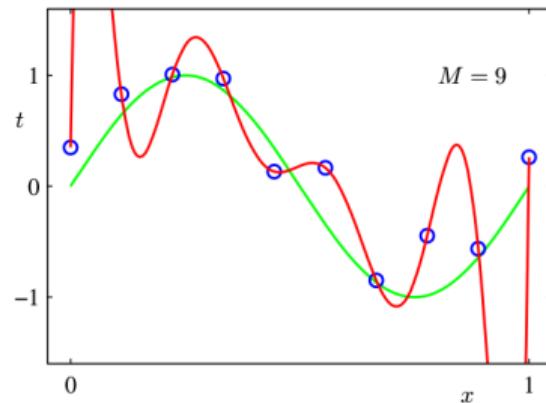
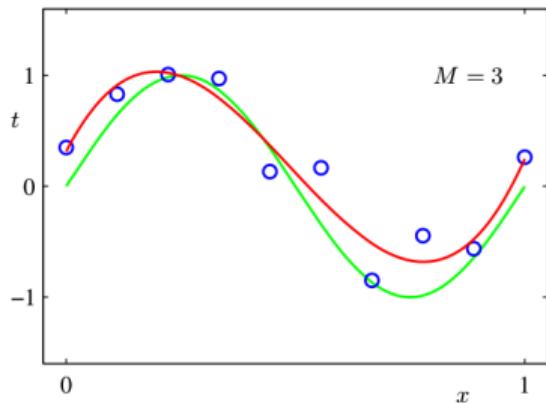


# Capacity, overfitting, underfitting

The higher the capacity of  $\mathcal{Q}$ , the more powerful the model.

Example (polynomial regression with degree  $M$ )

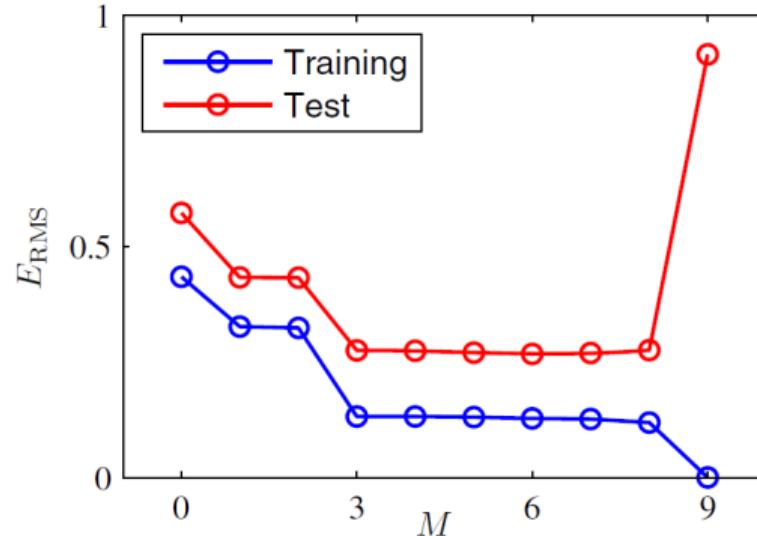
$M = 9$  (fixed)  
 $N$  increases



## Measuring under/over-fitting

Split  $\mathcal{D}$  into a training set  $\mathcal{D}_{train}$  and a test set  $\mathcal{D}_{test}$ .

Test error increases  $\implies$  overfitting...



Always evaluate your model on a separate test set !



# Artificial Neural Network (ANN)

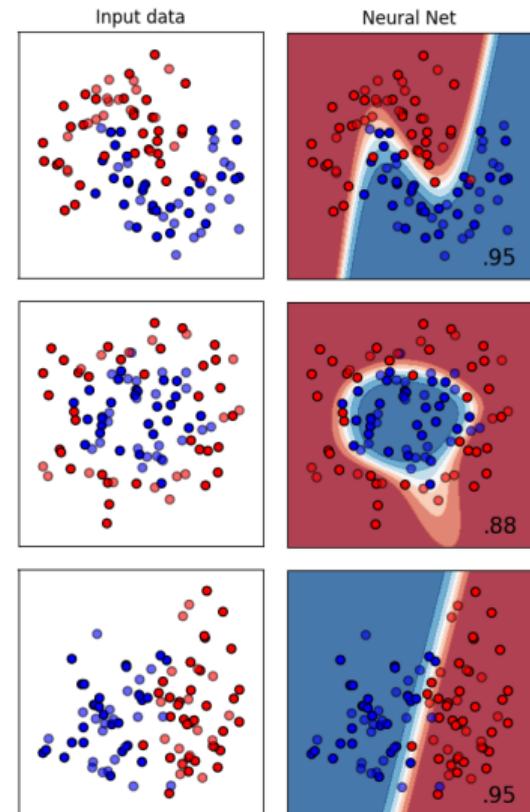
Pros:

- + state-of-the-art on many hard problems
- + scale well to large data sets

Cons:

- requires large data sets
- no theoretical guarantee
- more an art than a science

Covered in this lecture !



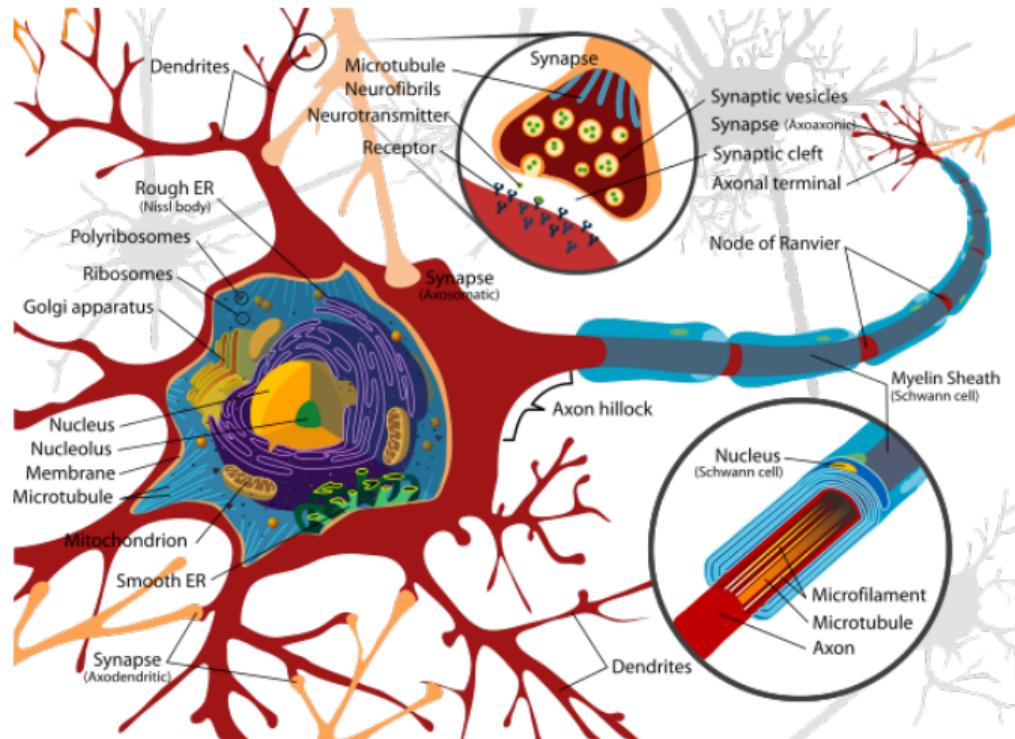
Perceptron Era (1958-60s)

# Biological neurons

Extremely complicated cells.

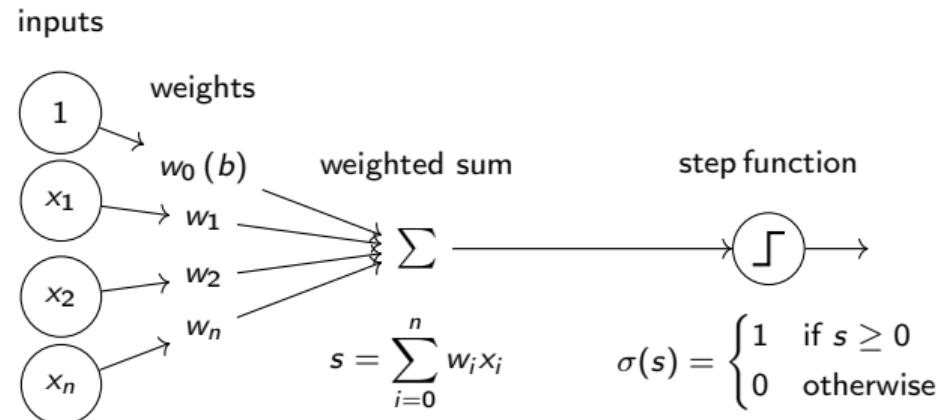
In a nutshell:

- ▶ Receives electrical pulses from dendrites (input).
- ▶ Accumulates electrical potential.
- ▶ Above a certain threshold, fires off along its axon (output).



# McCulloch-Pitts artificial neuron

A simplified mathematical model of how neurons operate.

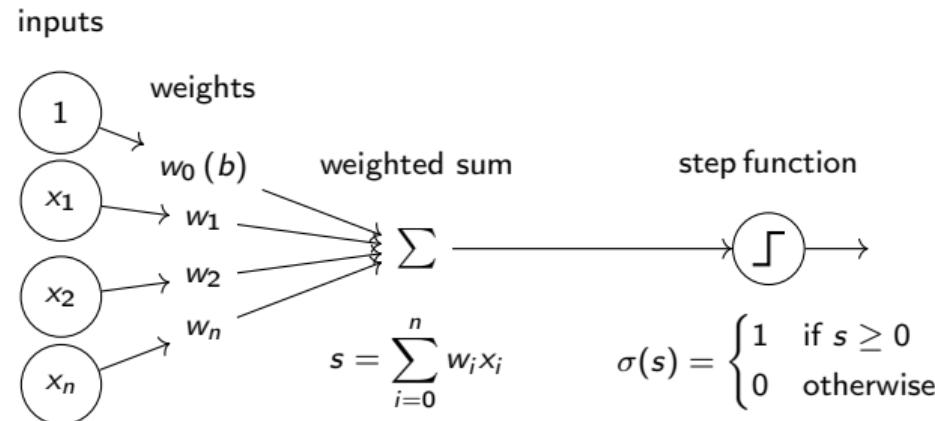


---

W. S. McCulloch and W. Pitts (1943). A logical calculus of the ideas immanent in nervous activity.

# McCulloch-Pitts artificial neuron

A simplified mathematical model of how neurons operate.



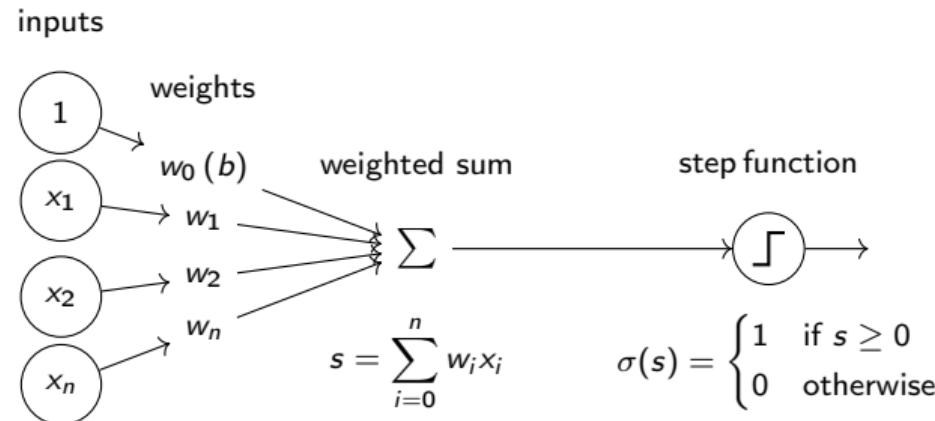
Can model logical gates: AND, OR, NOT.

---

W. S. McCulloch and W. Pitts (1943). A logical calculus of the ideas immanent in nervous activity.

## McCulloch-Pitts artificial neuron

A simplified mathematical model of how neurons operate.



Can model logical gates: AND, OR, NOT.

In NN jargon, that's a linear model:  $h(x) = \sigma(w^T x + b)$ .

---

W. S. McCulloch and W. Pitts (1943). A logical calculus of the ideas immanent in nervous activity.

## Rosenblatt's perceptron

$\mathcal{D}$  a training set of input-output pairs  $(x, y)$ , with  $x \in \mathbb{R}^n$  and  $y \in \{0, 1\}$ .

---

F. Rosenblatt (1957). The perceptron, a perceiving and recognizing automaton. Tech. rep. Project Para., Cornell Aeronautical Laboratory

## Rosenblatt's perceptron

$\mathcal{D}$  a training set of input-output pairs  $(x, y)$ , with  $x \in \mathbb{R}^n$  and  $y \in \{0, 1\}$ .

Start with random weights, and iterate over the training examples

- ▶ if target is 1 and output is 0:
  - ▶ increase weights  $w_i$  where  $x_i$  is positive
  - ▶ decrease weights  $w_i$  where  $x_i$  is negative
- ▶ if target is 0 and output is 1:
  - ▶ decrease weights  $w_i$  where  $x_i$  is positive
  - ▶ increase weights  $w_i$  where  $x_i$  is negative

---

F. Rosenblatt (1957). The perceptron, a perceiving and recognizing automaton. Tech. rep. Project Para., Cornell Aeronautical Laboratory

## Rosenblatt's perceptron

$\mathcal{D}$  a training set of input-output pairs  $(x, y)$ , with  $x \in \mathbb{R}^n$  and  $y \in \{0, 1\}$ .

Start with random weights, and iterate over the training examples

- ▶ if target is 1 and output is 0:
  - ▶ increase weights  $w_i$  where  $x_i$  is positive
  - ▶ decrease weights  $w_i$  where  $x_i$  is negative
- ▶ if target is 0 and output is 1:
  - ▶ decrease weights  $w_i$  where  $x_i$  is positive
  - ▶ increase weights  $w_i$  where  $x_i$  is negative

Formally:  $w \leftarrow w + \alpha(y - \sigma(w^T x))x$ , with  $\alpha$  the learning rate. Gradient descent ?

---

F. Rosenblatt (1957). The perceptron, a perceiving and recognizing automaton. Tech. rep. Project Para., Cornell Aeronautical Laboratory

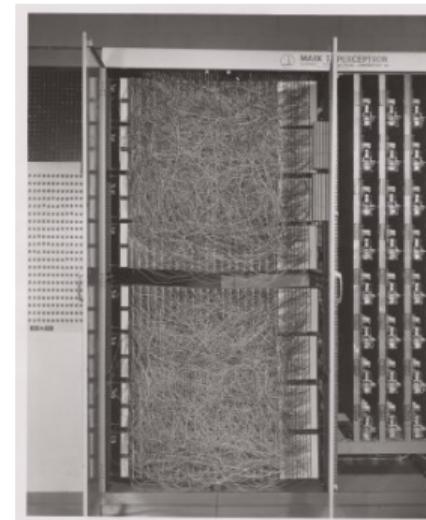
## Rosenblatt's perceptron

Custom hardware implementation: Mark I Perceptron

- ▶ input:  $20 \times 20$  photocells
- ▶ weights: potentiometers
- ▶ learning: electric motors

Classifies simple shapes correctly.

Machine learning is born!



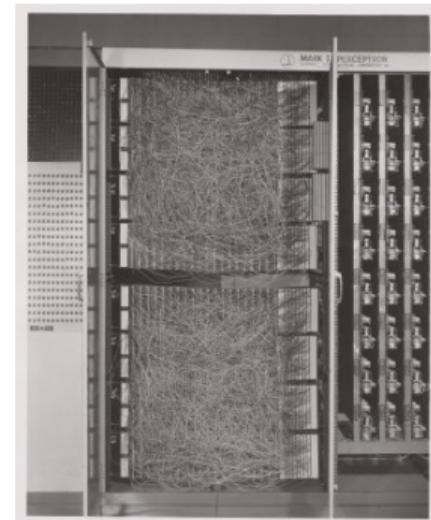
## Rosenblatt's perceptron

Custom hardware implementation: Mark I Perceptron

- ▶ input:  $20 \times 20$  photocells
- ▶ weights: potentiometers
- ▶ learning: electric motors

Classifies simple shapes correctly.

Machine learning is born!



« *The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence [...] Dr. Frank Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.* »

*New-York Times, 1958*

# 1969: First NN winter



---

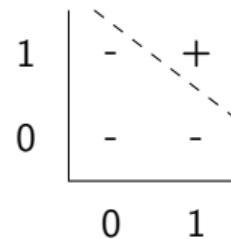
M. Minsky and S. Papert (1969). Perceptrons An Introduction to Computational Geometry.

## 1969: First NN winter

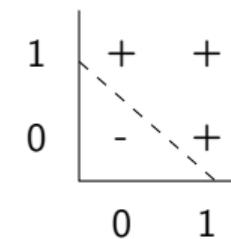
Critical analysis by Minsky and Papert (MIT lab).

Linearity limits, can not learn the simple XOR function.

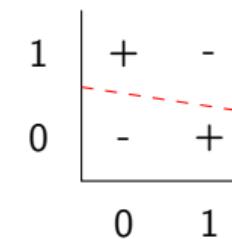
AND



OR



XOR

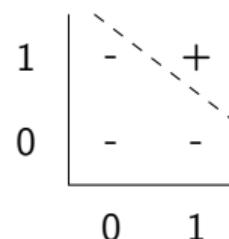


## 1969: First NN winter

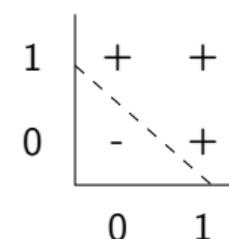
Critical analysis by Minsky and Papert (MIT lab).

Linearity limits, can not learn the simple XOR function.

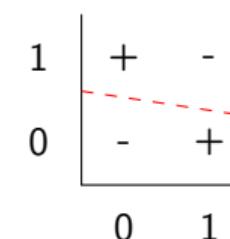
AND



OR



XOR



Non-linearity requires a multilayer perceptron (MLP).

Rosenblatt's algorithm does not work for multiple layers.

⇒ Massive disillusionment, freeze to funding and publications.

Backpropagation Era (1986-90s)

## Gradient descent

Gradient descent (GD):  $\theta \leftarrow \theta - \alpha \Delta \theta$ , with  $\alpha$  the learning rate.

## Gradient descent

Gradient descent (GD):  $\theta \leftarrow \theta - \alpha \Delta\theta$ , with  $\alpha$  the learning rate.

Batch GD: compute gradients over the whole training set  $\mathcal{D}$

$$\Delta\theta = \frac{\partial \sum_{(x,y) \in \mathcal{D}} L(h(x), y)}{\partial \theta} = \sum_{(x,y) \in \mathcal{D}} \frac{\partial L(h(x), y)}{\partial \theta}$$

## Gradient descent

Gradient descent (GD):  $\theta \leftarrow \theta - \alpha \Delta\theta$ , with  $\alpha$  the learning rate.

Batch GD: compute gradients over the whole training set  $\mathcal{D}$

$$\Delta\theta = \frac{\partial \sum_{(x,y) \in \mathcal{D}} L(h(x), y)}{\partial \theta} = \sum_{(x,y) \in \mathcal{D}} \frac{\partial L(h(x), y)}{\partial \theta}$$

Mini-batch GD: compute gradients over a small subset  $\mathcal{D}_{mb} \subset \mathcal{D}$

$$\Delta\theta = \sum_{(x,y) \in \mathcal{D}_{mb}} \frac{\partial L(h(x), y)}{\partial \theta}$$

## Gradient descent

Gradient descent (GD):  $\theta \leftarrow \theta - \alpha \Delta\theta$ , with  $\alpha$  the learning rate.

Batch GD: compute gradients over the whole training set  $\mathcal{D}$

$$\Delta\theta = \frac{\partial \sum_{(x,y) \in \mathcal{D}} L(h(x), y)}{\partial \theta} = \sum_{(x,y) \in \mathcal{D}} \frac{\partial L(h(x), y)}{\partial \theta}$$

Mini-batch GD: compute gradients over a small subset  $\mathcal{D}_{mb} \subset \mathcal{D}$

$$\Delta\theta = \sum_{(x,y) \in \mathcal{D}_{mb}} \frac{\partial L(h(x), y)}{\partial \theta}$$

Stochastic GD: compute gradients over a single example  $(x, y) \in \mathcal{D}$

$$\Delta\theta = \frac{\partial L(h(x), y)}{\partial \theta}$$

$\implies$  all of these will reach a local minima ( $\alpha \rightarrow 0$ ).

## Gradient descent

Gradient descent (GD):  $\theta \leftarrow \theta - \alpha \Delta\theta$ , with  $\alpha$  the learning rate.

Batch GD: compute gradients over the whole training set  $\mathcal{D}$

$$\Delta\theta = \frac{\partial \sum_{(x,y) \in \mathcal{D}} L(h(x), y)}{\partial \theta} = \sum_{(x,y) \in \mathcal{D}} \frac{\partial L(h(x), y)}{\partial \theta}$$

Mini-batch GD: compute gradients over a small subset  $\mathcal{D}_{mb} \subset \mathcal{D}$

$$\Delta\theta = \sum_{(x,y) \in \mathcal{D}_{mb}} \frac{\partial L(h(x), y)}{\partial \theta}$$

Stochastic GD: compute gradients over a single example  $(x, y) \in \mathcal{D}$

$$\Delta\theta = \frac{\partial L(h(x), y)}{\partial \theta}$$

$\implies$  all of these will reach a local minima ( $\alpha \rightarrow 0$ ).

Noisy but fast updates often more efficient. Also, can escape local minima !

## Back-propagation

Proposed in the 70s, rediscovered in the 80s by several groups.

---

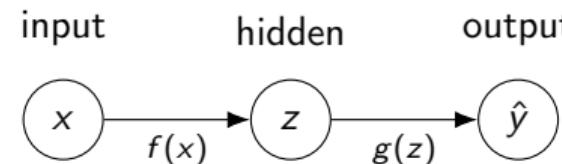
P. Werbos (1974). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams (1986). Learning representations by back-propagating errors.

# Back-propagation

Proposed in the 70s, rediscovered in the 80s by several groups.

Multi-layer network: composition of functions  $h = g \circ f$



---

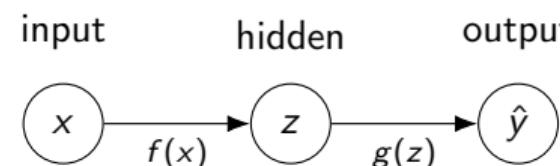
P. Werbos (1974). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams (1986). Learning representations by back-propagating errors.

# Back-propagation

Proposed in the 70s, rediscovered in the 80s by several groups.

Multi-layer network: composition of functions  $h = g \circ f$



Gradient: chain rule of calculus

$$\frac{\partial \hat{y}}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial \hat{y}}{\partial z}.$$

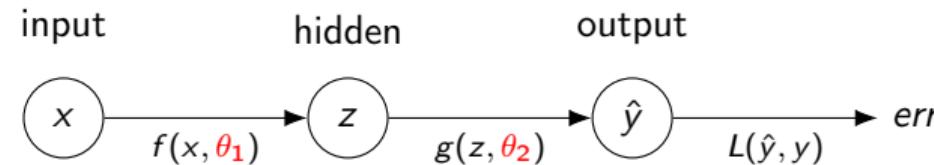
---

P. Werbos (1974). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams (1986). Learning representations by back-propagating errors.

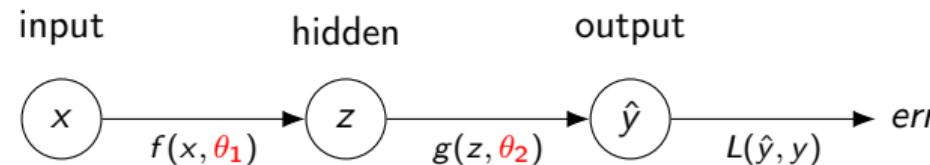
# Back-Propagation

Forward pass: composition of functions

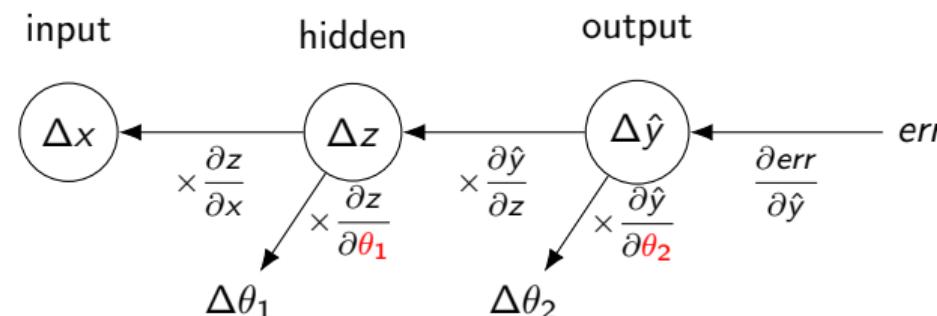


# Back-Propagation

Forward pass: composition of functions

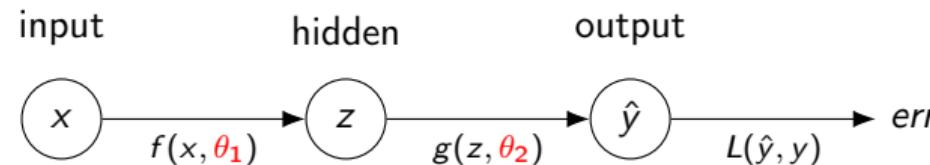


Backward pass: chain rule of calculus

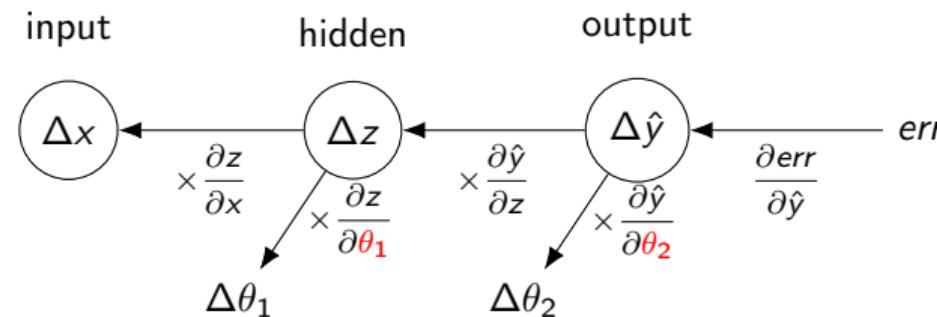


# Back-Propagation

Forward pass: composition of functions



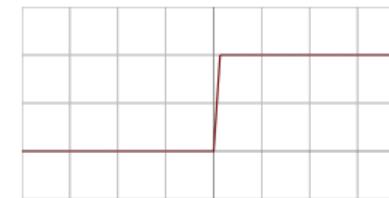
Backward pass: chain rule of calculus



$\implies$  all we need is  $f$ ,  $g$  and  $L$  differentiable.

## Activation functions

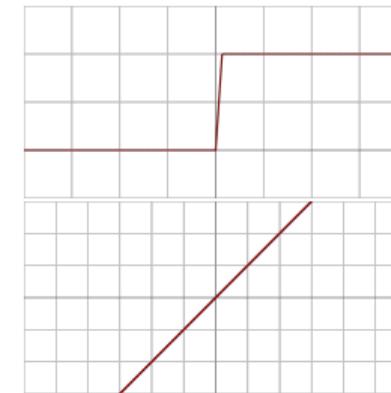
Step function: non-differentiable...



## Activation functions

Step function: non-differentiable...

Identity: linear model



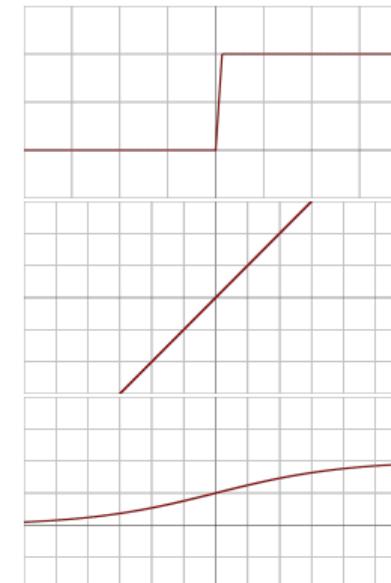
# Activation functions

Step function: non-differentiable...

Identity: linear model

Logistic (sigmoid)

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



# Activation functions

Step function: non-differentiable...

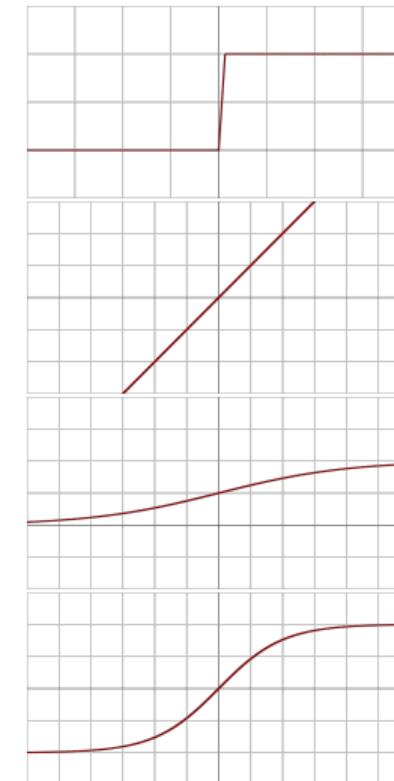
Identity: linear model

Logistic (sigmoid)

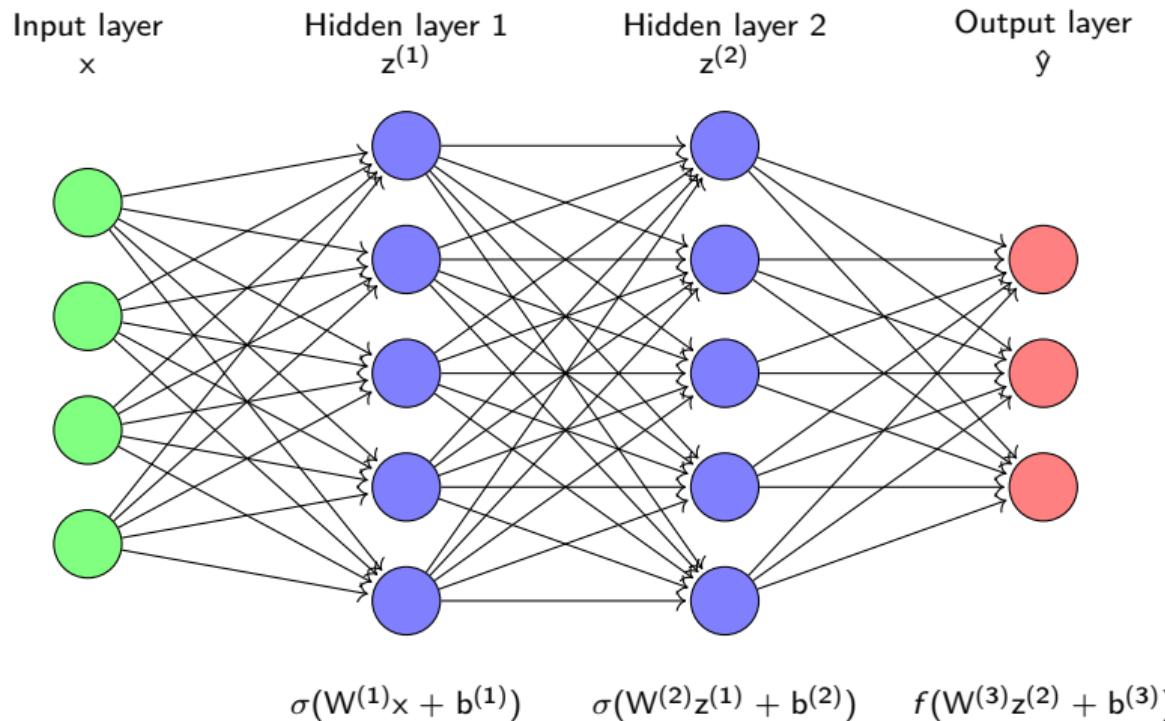
$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$

Hyperbolic tangent (tanh)

$$\sigma(x) = \frac{2}{1 + \exp^{-2x}} - 1$$

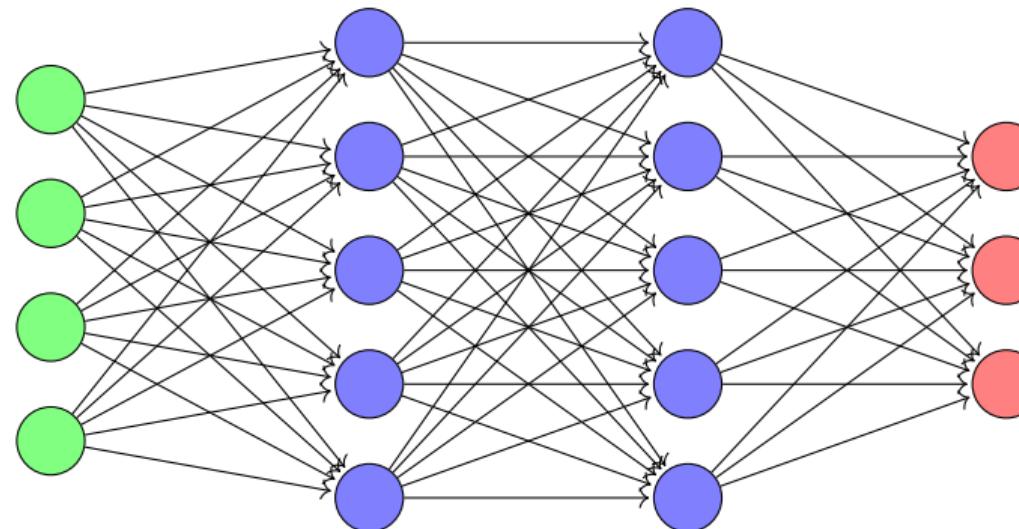


# Multilayer feed-forward neural network



# Multilayer feed-forward neural network

Input layer      Hidden layer 1      Hidden layer 2      Output layer  
 $x$                    $z^{(1)}$                    $z^{(2)}$                    $\hat{y}$

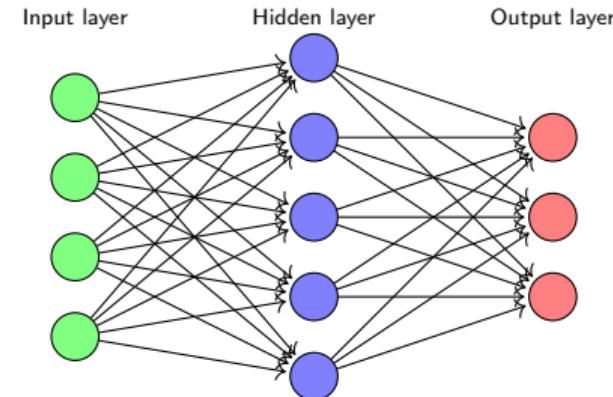


$$\sigma(W^{(1)}x + b^{(1)}) \quad \sigma(W^{(2)}z^{(1)} + b^{(2)}) \quad f(W^{(3)}z^{(2)} + b^{(3)})$$

# MLPs are universal approximators

A single hidden layer with sufficiently many hidden units can approximate any  $\mathcal{X} \mapsto \mathcal{Y}$  mapping to any desired degree of accuracy.

In other words: infinite capacity.



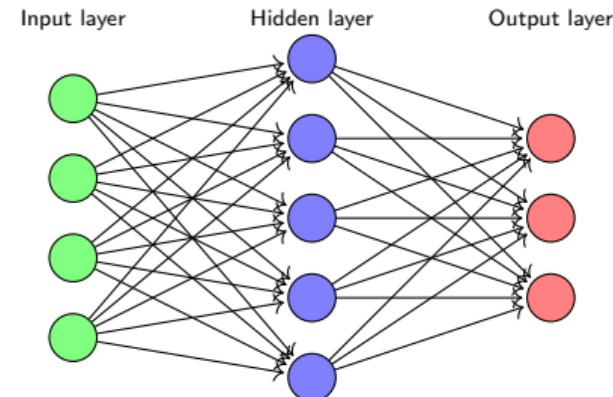
---

K. Hornik, M. B. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators.

## MLPs are universal approximators

A single hidden layer with sufficiently many hidden units can approximate any  $\mathcal{X} \mapsto \mathcal{Y}$  mapping to any desired degree of accuracy.

In other words: infinite capacity.



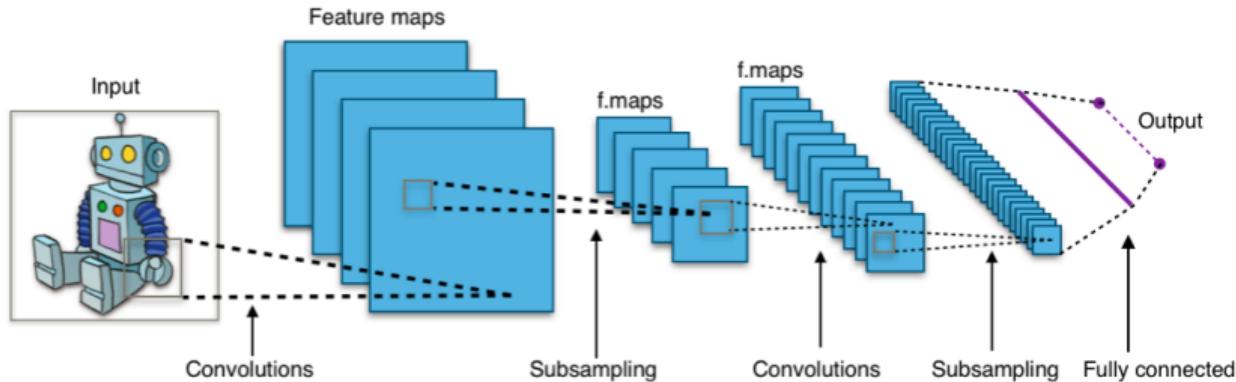
Strong theoretical argument at the time. Still, are NNs useful ?

- ▶ k-NN has infinite capacity;
- ▶ decision trees have infinite capacity;
- ▶ requires an exponential number of hidden neurons.

---

K. Hornik, M. B. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators.

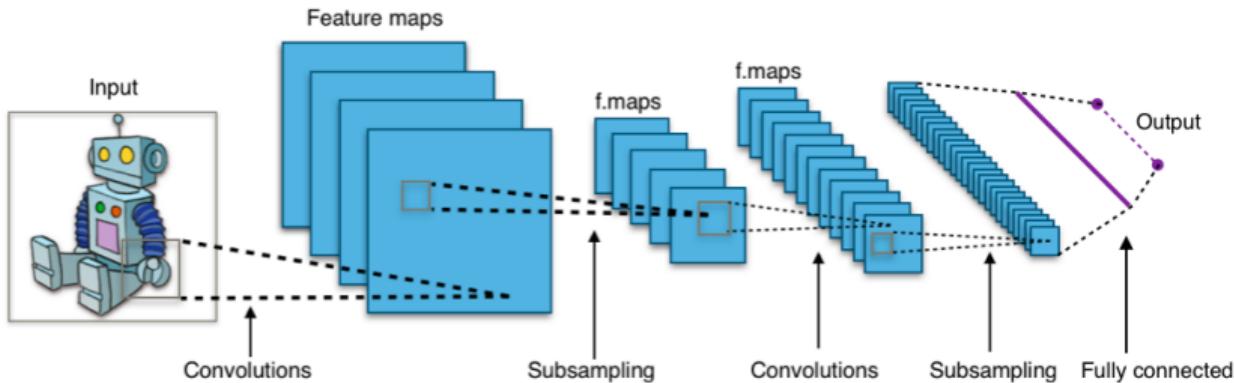
# Convolutional neural networks



Typical CNN layer: convolution + max pooling

- ▶ convolutions → weight sharing, less free parameters
- ▶ max pooling → translation invariance, dimensionality reduction

# Convolutional neural networks



Typical CNN layer: convolution + max pooling

- ▶ convolutions → weight sharing, less free parameters
- ▶ max pooling → translation invariance, dimensionality reduction

Leverages spatial / temporal structures

- ▶ image processing (2D spatial convolutions)
- ▶ signal processing (1D temporal convolution)

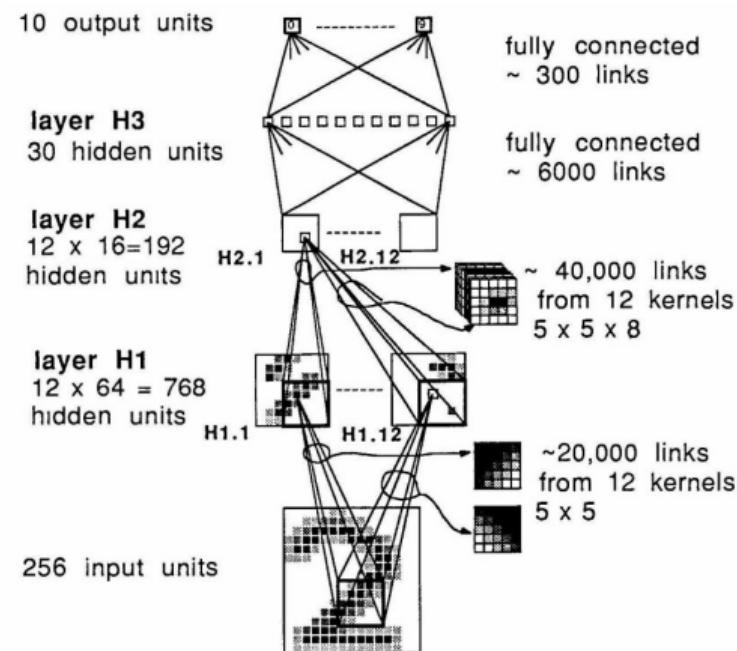
# Handwritten digit recognition: LeNet-5

SUN 4/260 16.67 MHz, 128 MB

- ▶  $\tanh$  activations
- ▶  $L_2$  loss
- ▶ Stochastic GD

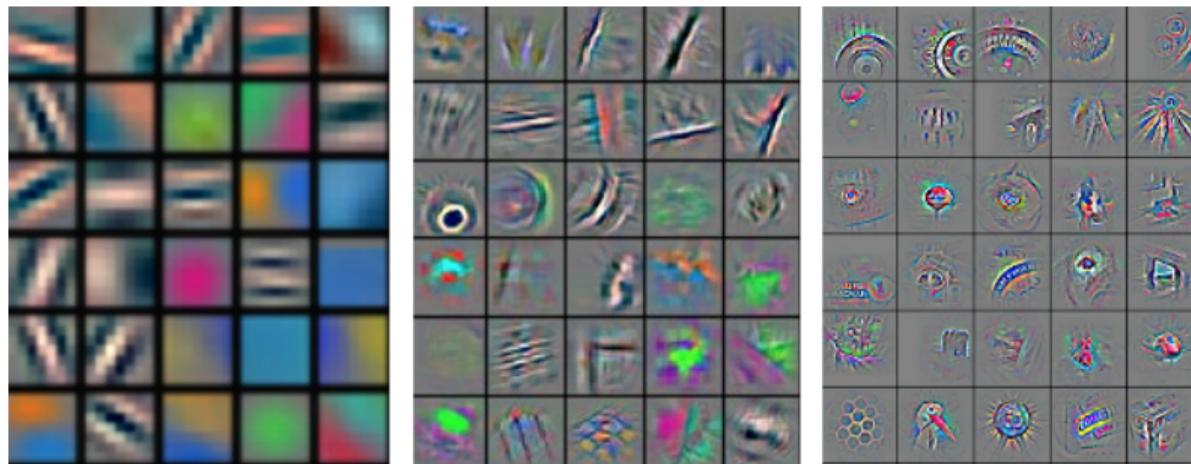
[https://youtu.be/FwFduRA\\_L6Q](https://youtu.be/FwFduRA_L6Q)

At some point in the 90s, read about 10-20% of all the checks in the US.



Y. LeCun et al. (1989). Backpropagation Applied to Handwritten Zip Code Recognition.

## CNN feature maps



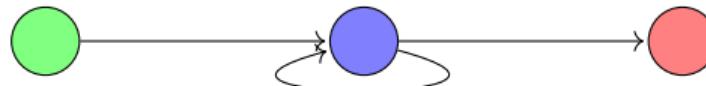
<https://distill.pub/2017/feature-visualization/>

Intuition:

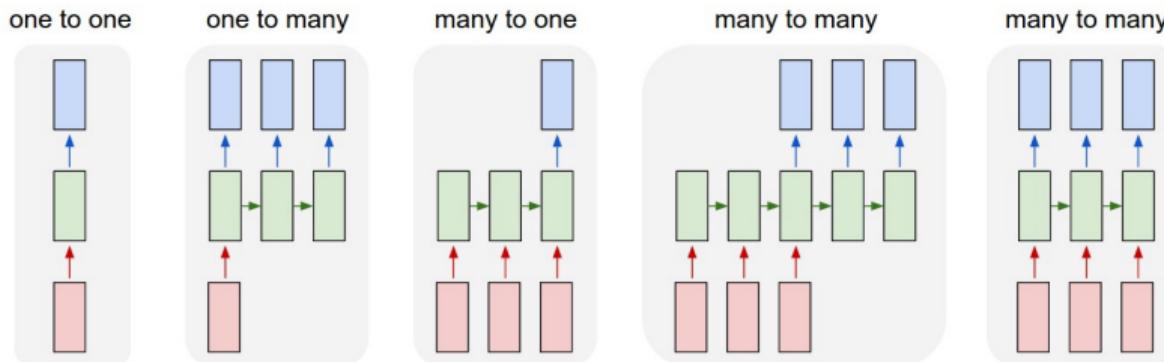
- ▶ lower layers = low-level features (edges, blobs)
- ▶ higher layers = high-level features

## Recurrent neural networks

Time-delayed connections  $\implies$  memory.



Can process non i.i.d. data (sequences):



Back-propagation through time (BPTT)  $\implies$  vanishing / exploding gradients...

## The vanishing gradient problem

Chain rule  $\approx$  geometric sequence

- ▶ derivatives  $\frac{\partial \sigma(x)}{x} < 1 \implies$  vanishing gradient
- ▶ derivatives  $\frac{\partial \sigma(x)}{x} > 1 \implies$  exploding gradient

Related to:

- ▶ activation function  $\sigma$
- ▶ initial weight initialization

Affects MLPs with many layers, and recurrent neural nets.

- ▶ hard to train deep networks;
- ▶ hard to learn long-term dependencies.

## Limitations

Gradient descent: no theoretical guarantee

- ▶ non-convex optimization problem (hidden layers)
- ▶ quality of the local minimum ?

## Limitations

Gradient descent: no theoretical guarantee

- ▶ non-convex optimization problem (hidden layers)
- ▶ quality of the local minimum ?

Backpropagation: vanishing / exploding gradients

- ▶ training of deep / recurrent networks is problematic

## Limitations

Gradient descent: no theoretical guarantee

- ▶ non-convex optimization problem (hidden layers)
- ▶ quality of the local minimum ?

Backpropagation: vanishing / exploding gradients

- ▶ training of deep / recurrent networks is problematic

Computer resources: limited at the time

- ▶ convergence is slow

## Limitations

Gradient descent: no theoretical guarantee

- ▶ non-convex optimization problem (hidden layers)
- ▶ quality of the local minimum ?

Backpropagation: vanishing / exploding gradients

- ▶ training of deep / recurrent networks is problematic

Computer resources: limited at the time

- ▶ convergence is slow

Outperformed by other models in the 2000s (random forest, SVM).

## Limitations

Gradient descent: no theoretical guarantee

- ▶ non-convex optimization problem (hidden layers)
- ▶ quality of the local minimum ?

Backpropagation: vanishing / exploding gradients

- ▶ training of deep / recurrent networks is problematic

Computer resources: limited at the time

- ▶ convergence is slow

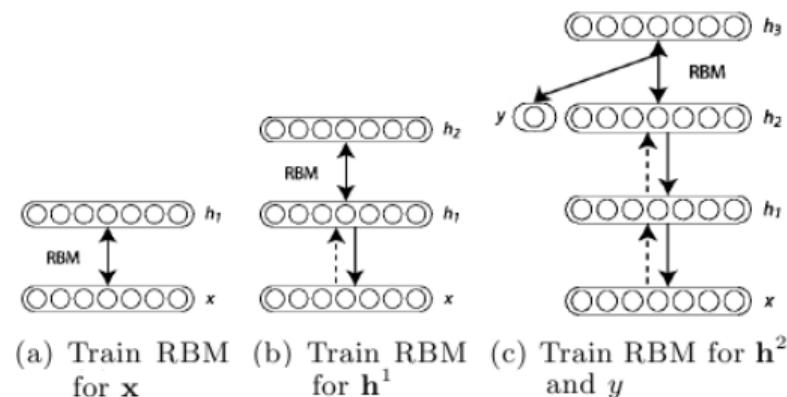
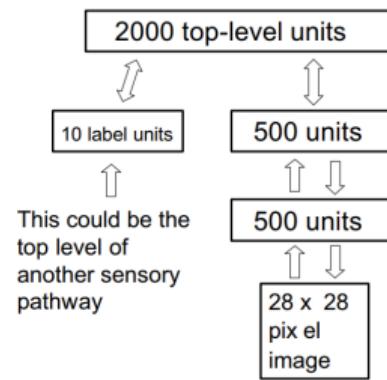
Outperformed by other models in the 2000s (random forest, SVM).

Despite some irrefutable successes, neural networks lose popularity.

⇒ Second NN winter.

Deep Learning Era (2006-)

# 2006 Deep belief network (DBN)



Unsupervised layer-by-layer pre-training (weight initialization).

⇒ successful training of deep neural nets is possible.

1.25% error on MNIST without convolutions ! (0.95% with CNNs)

Generative model of digit images (MNIST).

G. E. Hinton, S. Osindero, and Y. W. Teh (2006). A Fast Learning Algorithm for Deep Belief Nets.

## 2006 Resurgence: Deep belief network (DBN)

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

Generative model: "Looking into  
the mind of a neural network".

## Computational power

Forward / backward passes highly parallel by nature.

Use GPUs instead of CPUs for training ?  $\implies$  70x speed-up.

---

R. Raina, A. Madhavan, and A. Y. Ng (2009). Large-scale deep unsupervised learning using graphics processors.

Q. V. Le et al. (2012). Building high-level features using large scale unsupervised learning.

## Computational power

Forward / backward passes highly parallel by nature.

Use GPUs instead of CPUs for training ?  $\implies$  70x speed-up.

DBN training on Youtube images by Google

- ▶ 16 000 parallel CPUs
- ▶  $\sim$  1 billion weights (Hinton's 2006 DBN  $\sim$  1M)

«We never told it during the training, 'This is a cat'. It basically invented the concept of a cat.» *Jeff Dean*



One of the neurons.

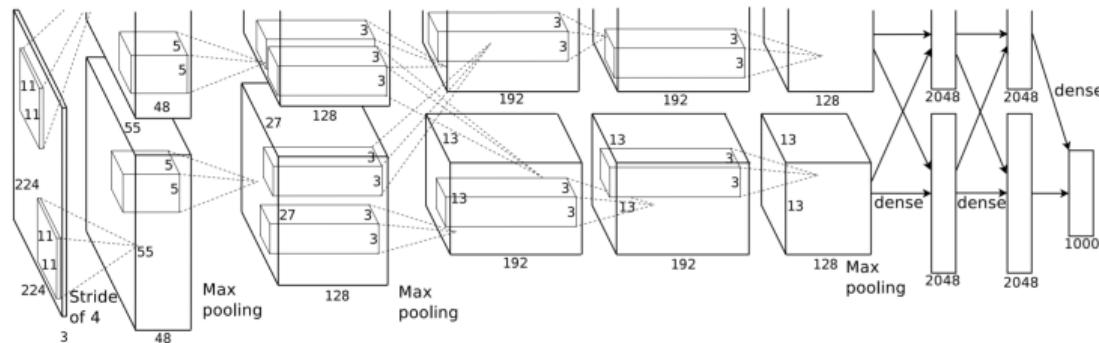
---

R. Raina, A. Madhavan, and A. Y. Ng (2009). Large-scale deep unsupervised learning using graphics processors.

Q. V. Le et al. (2012). Building high-level features using large scale unsupervised learning.

## 2012 Tsunami: AlexNet

Successful supervised training of a deep convolutional network on GPUs.



- ▶ one week training on two gaming GPUs (Nvidia GTX 580, 3GB)
- ▶ 5 convolutional layers + 3 fully-connected layers
- ▶ relu activation functions (non-saturating neurons)
- ▶ dropout (regularization)
- ▶ data augmentation (cropping + translations)

---

A. Krizhevsky, I. Sutskever, and G. E. Hinton (2012). ImageNet Classification with Deep Convolutional Neural Networks.

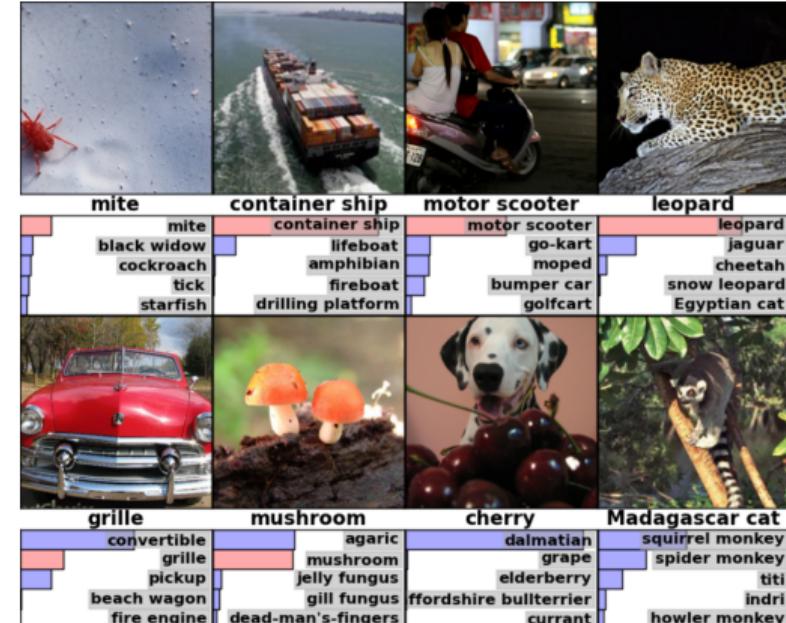
## 2012 Tsunami: AlexNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012

- ▶ crowd-sourced image annotation
- ▶ 1000 classes
- ▶ 1.3 million training examples

The only NN competitor, AlexNet, wins by a large margin

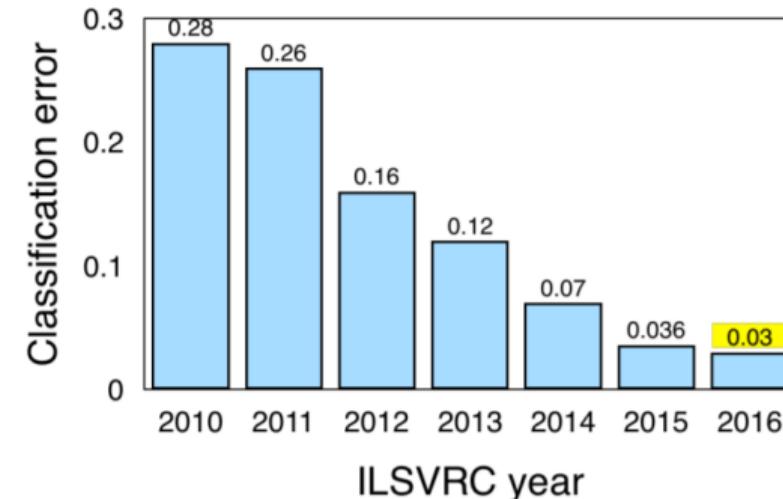
- ▶ 15.3% top-5 error rate, compared to 26.2% for second entry



A. Krizhevsky, I. Sutskever, and G. E. Hinton (2012). ImageNet Classification with Deep Convolutional Neural Networks.

## 2012 Tsunami: AlexNet

Since 2012: overflow of CNN competitors, test error continually decreases



## Modern activation functions

Rectified linear unit (relu)

$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



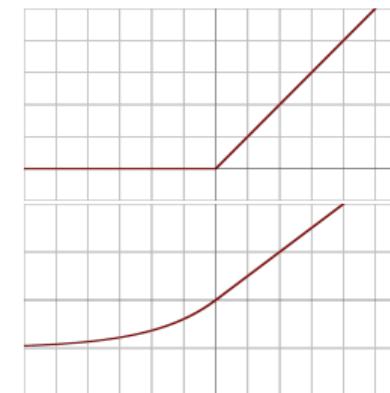
## Modern activation functions

Rectified linear unit (relu)

$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Exponential linear unit (elu)

$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ \exp(x) - 1 & \text{otherwise} \end{cases}$$



## Modern activation functions

Rectified linear unit (relu)

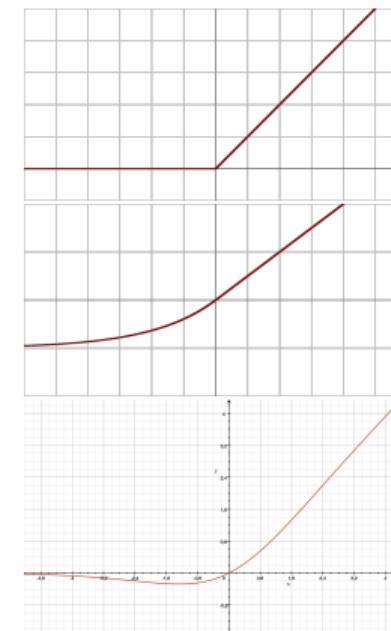
$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Exponential linear unit (elu)

$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ \exp(x) - 1 & \text{otherwise} \end{cases}$$

Google's activation (swish)

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



---

P. Ramachandran, B. Zoph, and Q. Le (2017). Swish: a Self-Gated Activation Function.

## Modern activation functions

Rectified linear unit (relu)

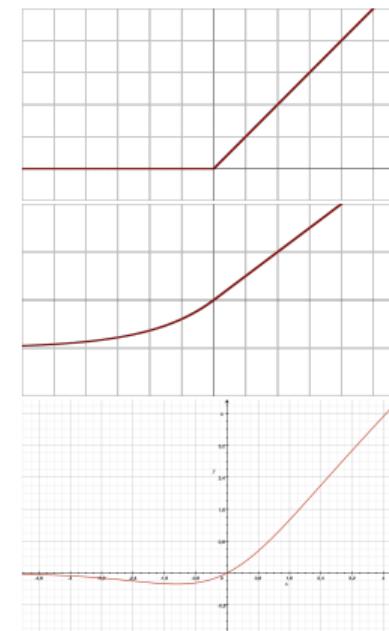
$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Exponential linear unit (elu)

$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ \exp(x) - 1 & \text{otherwise} \end{cases}$$

Google's activation (swish)

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



⇒ non-saturating neurons.

---

P. Ramachandran, B. Zoph, and Q. Le (2017). Swish: a Self-Gated Activation Function.

## Modern gradient descent algorithms

Momentum:

$$\begin{aligned}\Delta\theta &\leftarrow \beta\Delta\theta + (1 - \beta)\frac{\partial L}{\partial\theta} \quad \text{with } \beta \in ]0, 1[, \\ \theta &\leftarrow \theta - \alpha\Delta\theta.\end{aligned}$$



No momentum



Momentum

⇒ faster convergence, especially with SGD.

## Modern gradient descent algorithms

Adaptive moment estimation (adam): first and second order statistics

$$m \leftarrow \beta_1 m + (1 - \beta_1) \frac{\partial L}{\partial \theta} \quad (\text{mean}),$$

$$v \leftarrow \beta_2 v + (1 - \beta_2) \left( \frac{\partial L}{\partial \theta} \right)^2 \quad (\text{uncentered variance}),$$

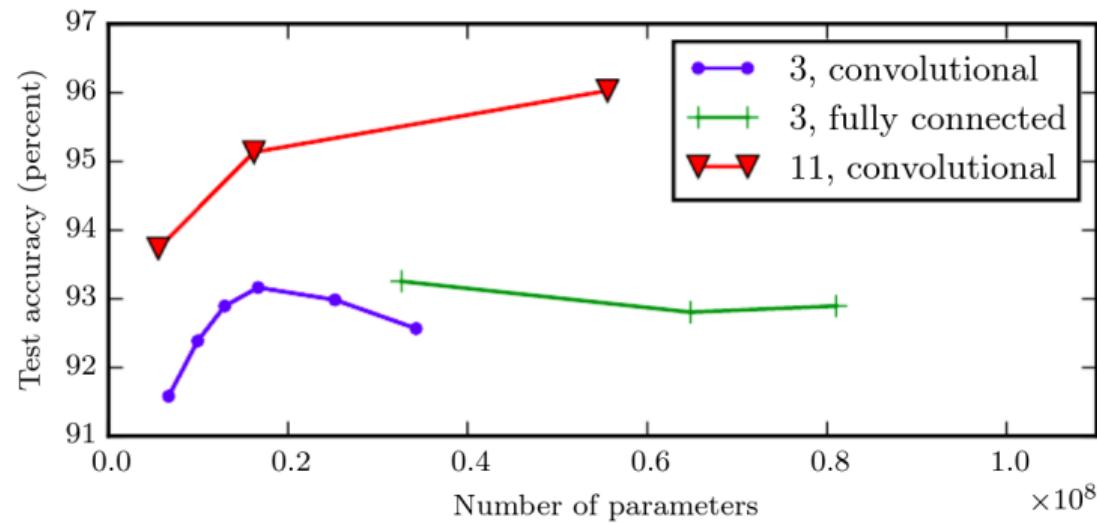
$$\hat{m} = \frac{m}{1 - \beta_1} \quad (\text{bias correction}),$$

$$\hat{v} = \frac{v}{1 - \beta_2} \quad (\text{bias correction}),$$

$$\theta \leftarrow \theta - \alpha \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}.$$

⇒ really fast, often used by default.

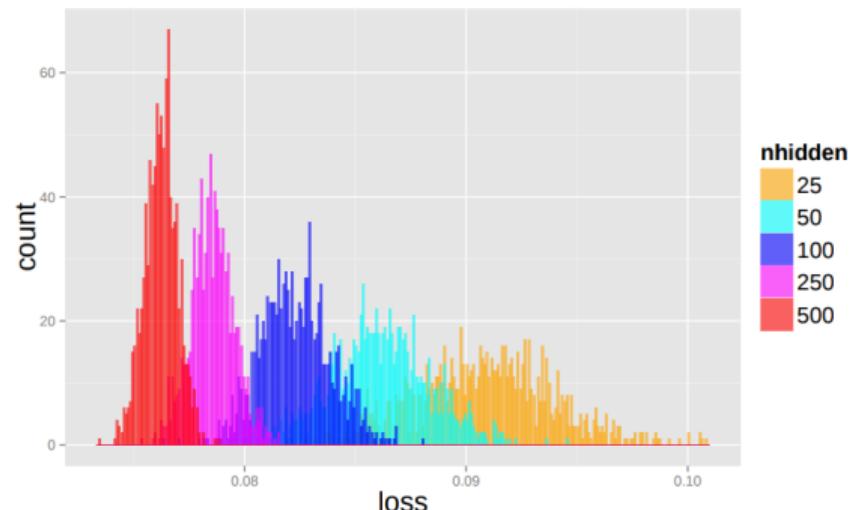
## Deeper is better than wider



# Local minima is not an issue with large networks

Analogy to spin glasses

«For large-size decoupled networks the lowest critical values [...] are located in a well-defined band lower-bounded by the global minimum. The number of local minima outside that band diminishes exponentially with the size of the network.»



---

A. Choromanska et al. (2015). The Loss Surfaces of Multilayer Networks.

## Recent developments

### Architectures

- ▶ Residual networks, 1000+ layers
- ▶ Transformer networks (text)
- ▶ Graph neural networks (proteins, networks)

### Normalization techniques (stabilizes gradients)

- ▶ Batch norm
- ▶ Layer norm, instance norm, group norm
- ▶ Weight norm

Core ingredients: **compositional** architectures + **differentiable** operations + backpropagation.

---

See also <https://bmk.sh/2019/12/31/The-Decade-of-Deep-Learning/>

# Neural Network - Foundations

CORS 2021 Workshop

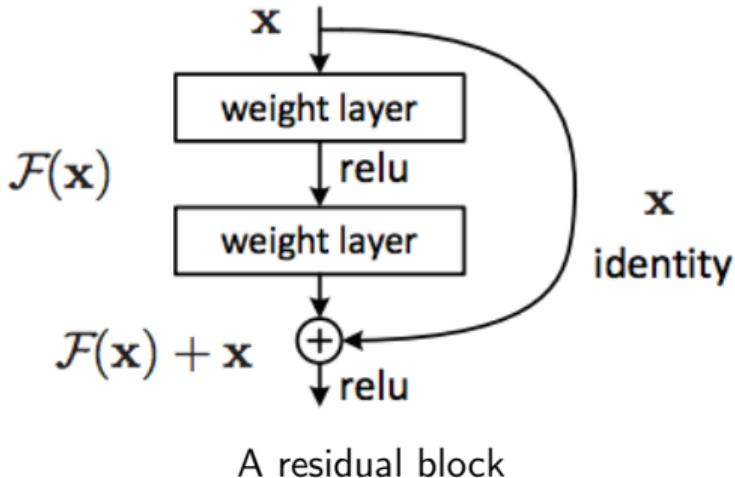
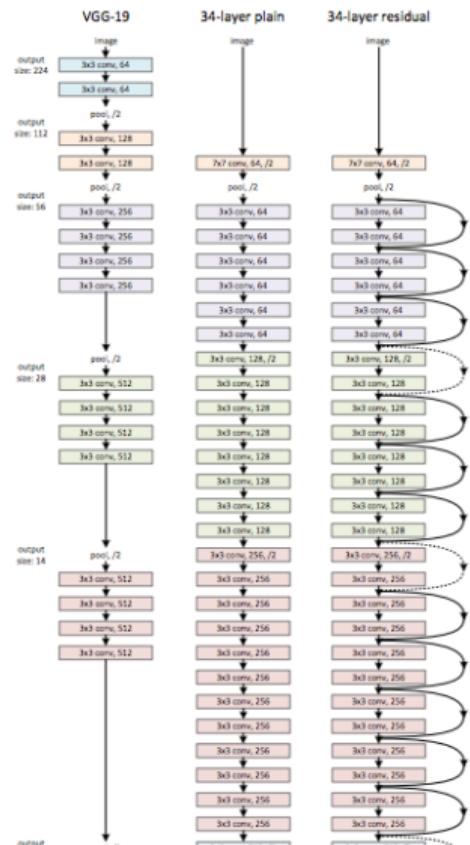
Thank you!

Maxime Gasse, Polytechnique Montréal



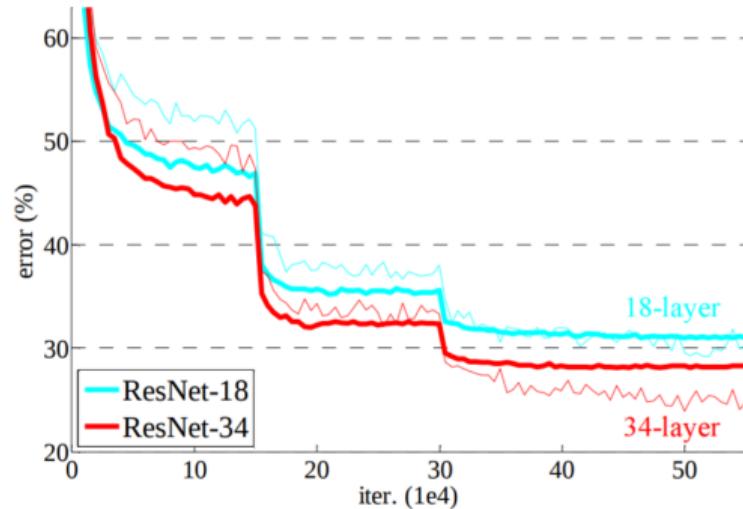
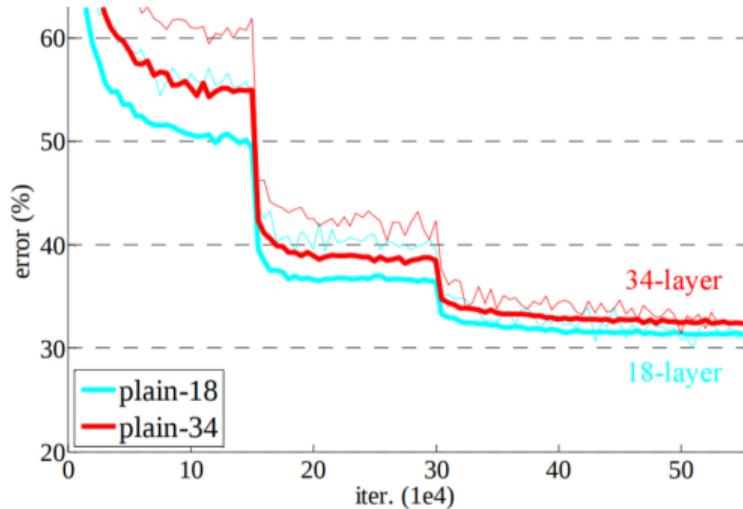
CORS · SCRO

## Residual networks



State-of-the-art on many tasks (AlphaGo Zero)

## Residual networks



Scales up to a thousand layers  
(but actually exploits paths of < 30 layers).

K. He et al. (2015). Deep Residual Learning for Image Recognition.

A. Veit, M. J. Wilber, and S. J. Belongie (2016). Residual Networks Behave Like Ensembles of Relatively Shallow Networks.

# Generative Adversarial Networks (GANs)

«Adversarial training is the coolest thing since sliced bread » Yann LeCun



---

T. Karras et al. (2017). Progressive Growing of GANs for Improved Quality, Stability, and Variation.

## Generative Adversarial Networks (GANs)

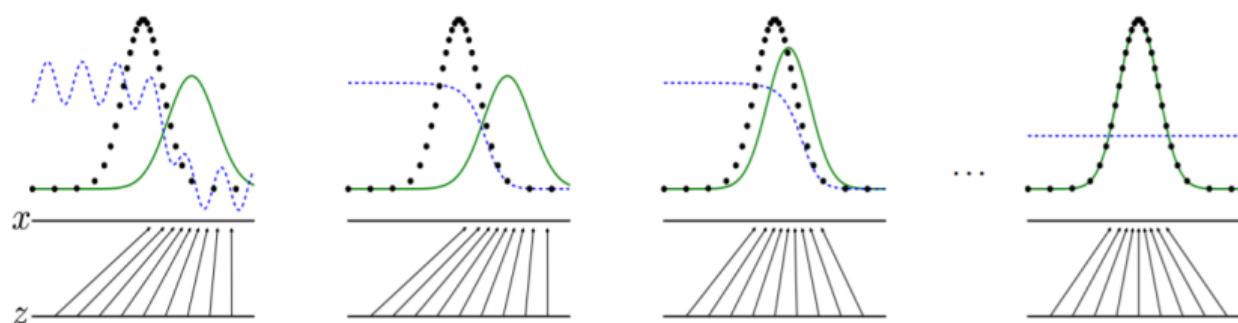
Real samples  $x \sim p(x)$ , fake samples  $G(z) \sim q(x)$  where  $z \sim U(0, 1)^M$ .

Generator network  $G : \mathcal{Z} \rightarrow \mathcal{X}$ .

Discriminator network  $D : \mathcal{X} \rightarrow \mathbb{R}$  (bounded).

Two-player min-max game:

$$G^*, D^* = \arg \min_G \max_D \mathbb{E}_{x,z} [D(x) - D(G(z))]$$

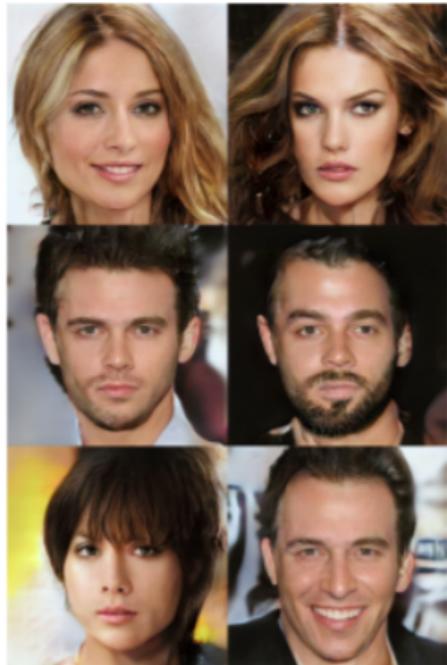
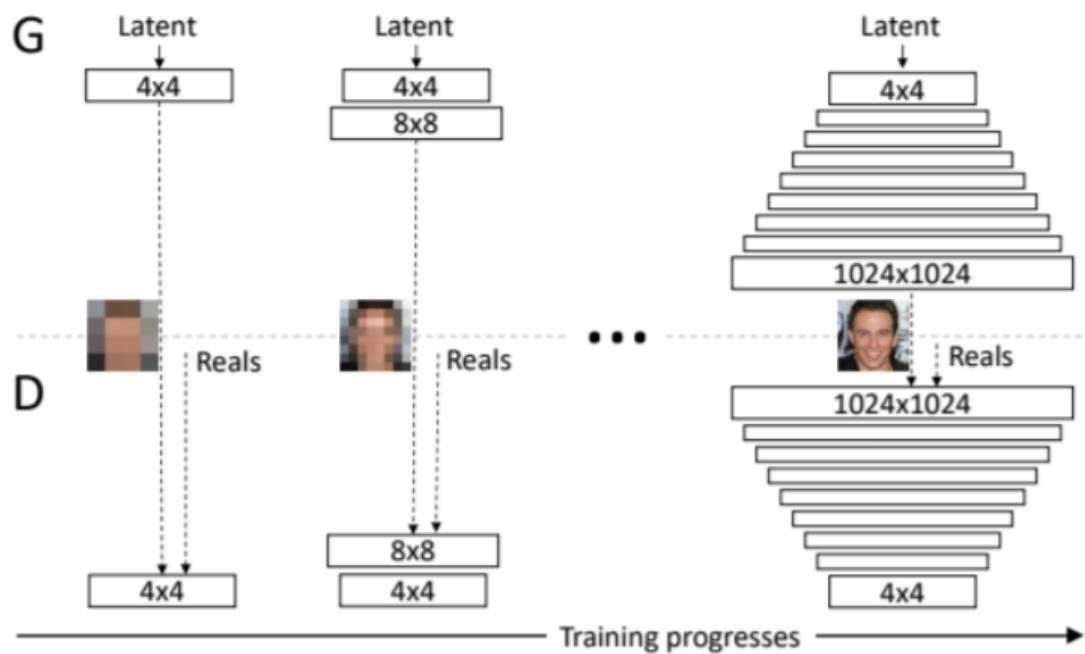


Optimal point: Nash equilibrium.

---

I. J. Goodfellow et al. (2014). Generative Adversarial Nets.

## Realistic face image generation



<https://youtu.be/X0xxPcy5Gr4>

T. Karras et al. (2017). Progressive Growing of GANs for Improved Quality, Stability, and Variation.