Deep learning

Alexandre Boulch

www.boulch.eu

IOGS - ATSI

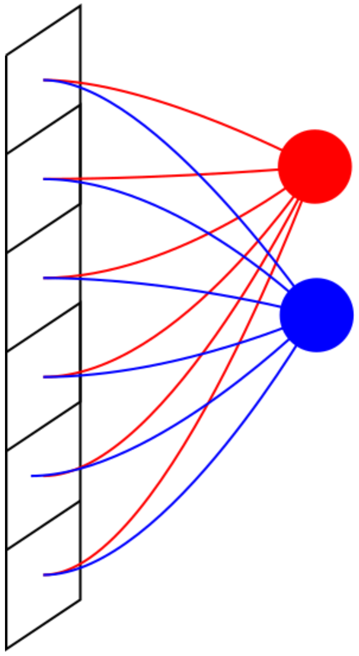
Outline

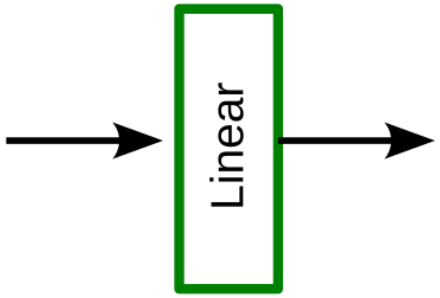
- Back on last course
- Concept of deep learning
- Convolutional Neural Networks
- Attention and transformers

Back on neural networks

The linear layer

- Also called *fully connected*
    - a neuron is connected to all the inputs
- High number of parameters ($\mathbf{W}$ matrix): $|inputs| * |outputs|$

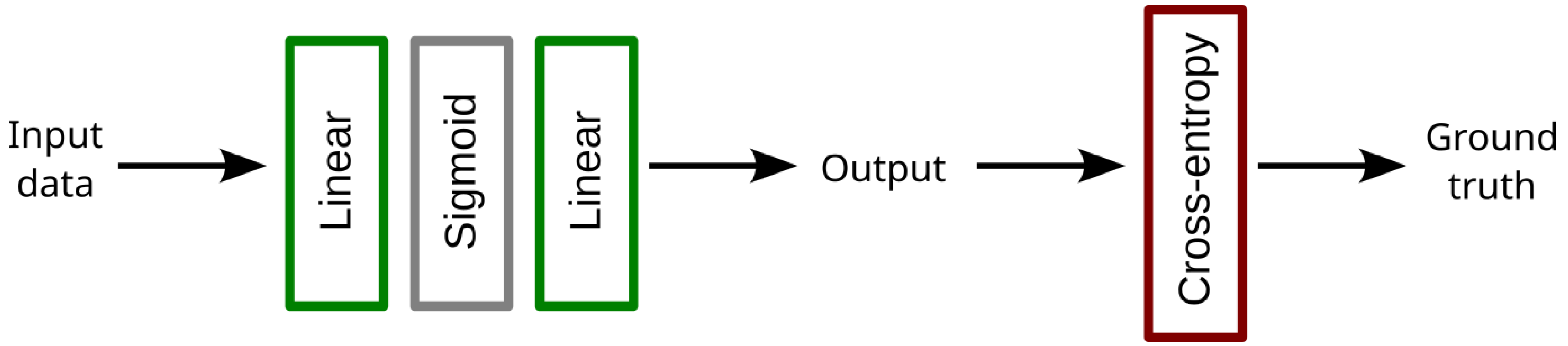




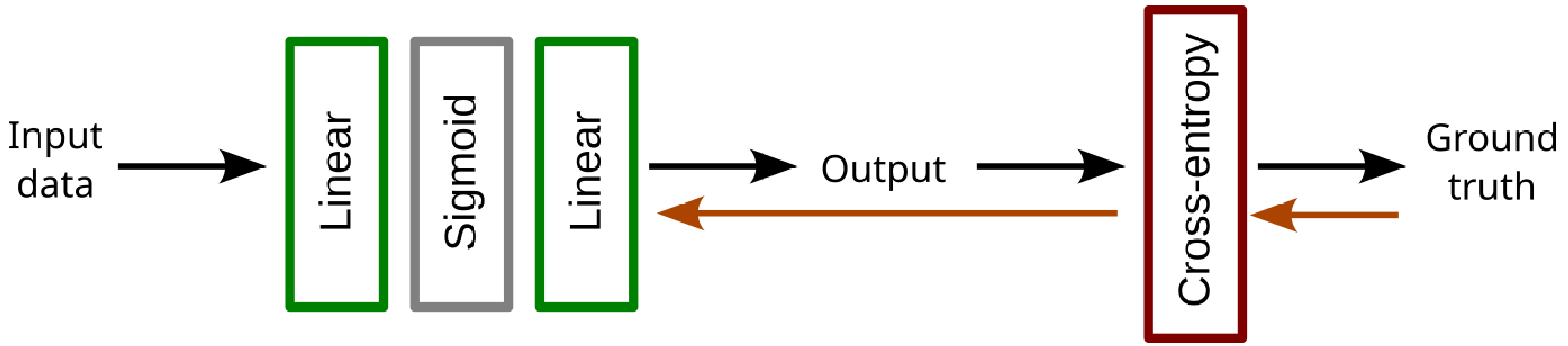
Multi-layer perceptron

A stack of linear layers with activation functions (e.g., sigmoids)

Optimization with **gradient descent**.



Optimization: forward-backward algorithm



Chain rule applied to neural networks

Forward

$$\boldsymbol{y}_1 = f_1(\boldsymbol{x}_n, \theta_1) \qquad \boldsymbol{y}_2 = f_2(\boldsymbol{y}_1) \qquad y_3 = f_3(\boldsymbol{y}_2, \theta_2) \qquad \hat{y} = L(y_3, y_n)$$

Chain rule applied to neural networks

Forward

$$\boldsymbol{y}_1 = f_1(\boldsymbol{x}_n, \theta_1) \qquad \boldsymbol{y}_2 = f_2(\boldsymbol{y}_1) \qquad y_3 = f_3(\boldsymbol{y}_2, \theta_2) \qquad \hat{y} = L(y_3, y_n)$$

Backward

$$\frac{\partial L}{\partial y_3} = \ldots$$

$$\frac{\partial L}{\partial y_2} = \boxed{\frac{\partial f_3}{\partial y_2}} \boxed{\frac{\partial L}{\partial y_3}}$$

$$\frac{\partial L}{\partial \theta_2} = \boxed{\frac{\partial f_3}{\partial \theta_2}} \boxed{\frac{\partial L}{\partial y_3}}$$

$$\frac{\partial L}{\partial y_1} = \boxed{\frac{\partial f_2}{\partial y_1}} \boxed{\frac{\partial L}{\partial y_2}}$$

$$\frac{\partial L}{\partial \theta_1} = \boxed{\frac{\partial f_1}{\partial \theta_1}} \boxed{\frac{\partial L}{\partial y_1}}$$

Chain rule applied to neural networks

Forward

$$\boldsymbol{y}_1 = f_1(\boldsymbol{x}_n, \theta_1) \qquad \boldsymbol{y}_2 = f_2(\boldsymbol{y}_1) \qquad y_3 = f_3(\boldsymbol{y}_2, \theta_2) \qquad \hat{y} = L(y_3, y_n)$$

Backward

$$\frac{\partial L}{\partial \theta_1} = \boxed{\frac{\partial f_1}{\partial \theta_1}} \boxed{\frac{\partial L}{\partial y_1}} \qquad \frac{\partial L}{\partial y_1} = \boxed{\frac{\partial f_2}{\partial y_1}} \boxed{\frac{\partial L}{\partial y_2}} \qquad \frac{\partial L}{\partial y_2} = \boxed{\frac{\partial f_3}{\partial y_2}} \boxed{\frac{\partial L}{\partial y_3}} \qquad \frac{\partial L}{\partial y_3} = \dots$$

$$\frac{\partial L}{\partial \theta_2} = \boxed{\frac{\partial f_3}{\partial \theta_2}} \boxed{\frac{\partial L}{\partial y_3}}$$

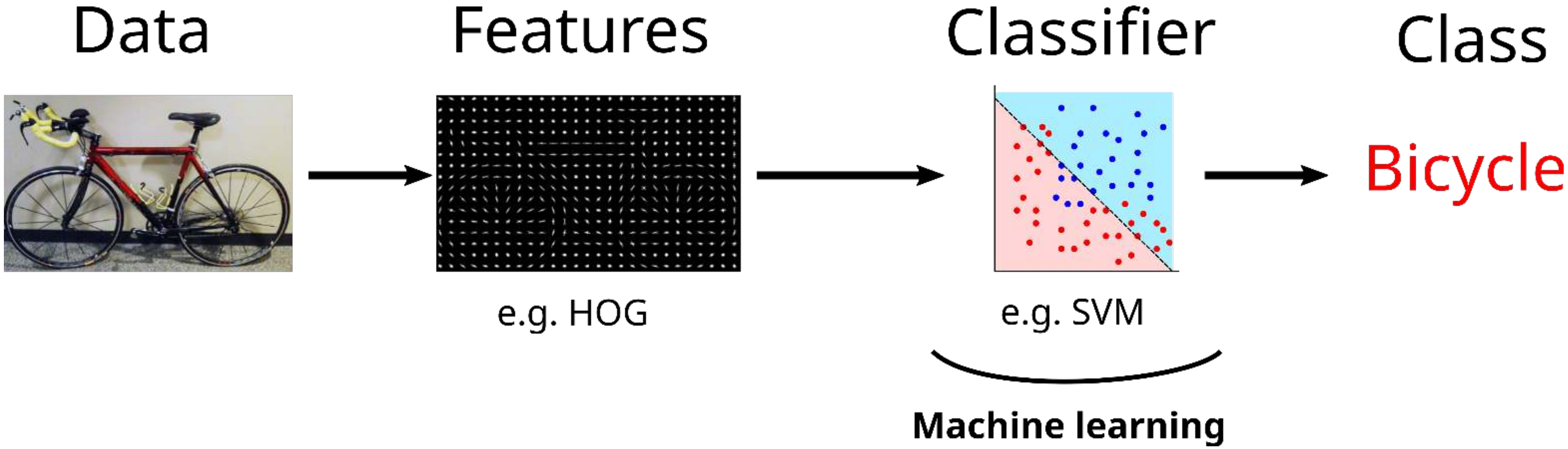
$$\theta_1 \leftarrow \theta_1 - r \frac{\partial L}{\partial \theta_1}$$

Weight update

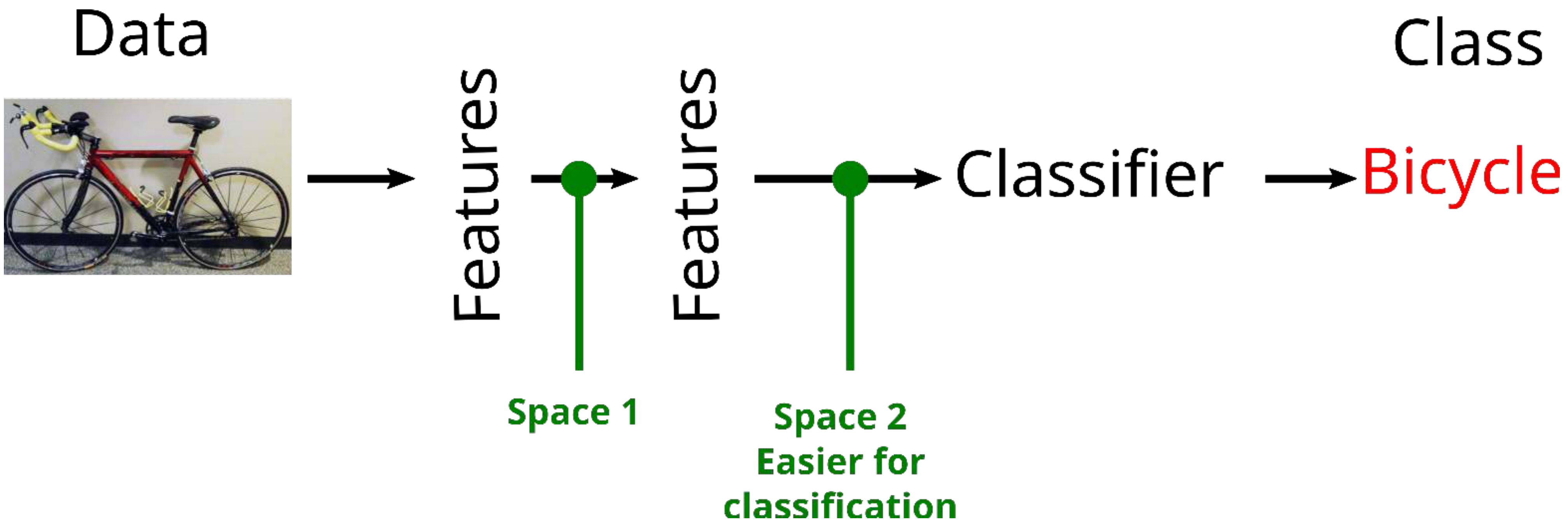
$$\theta_2 \leftarrow \theta_2 - r \frac{\partial L}{\partial \theta_2}$$

Deep learning concept

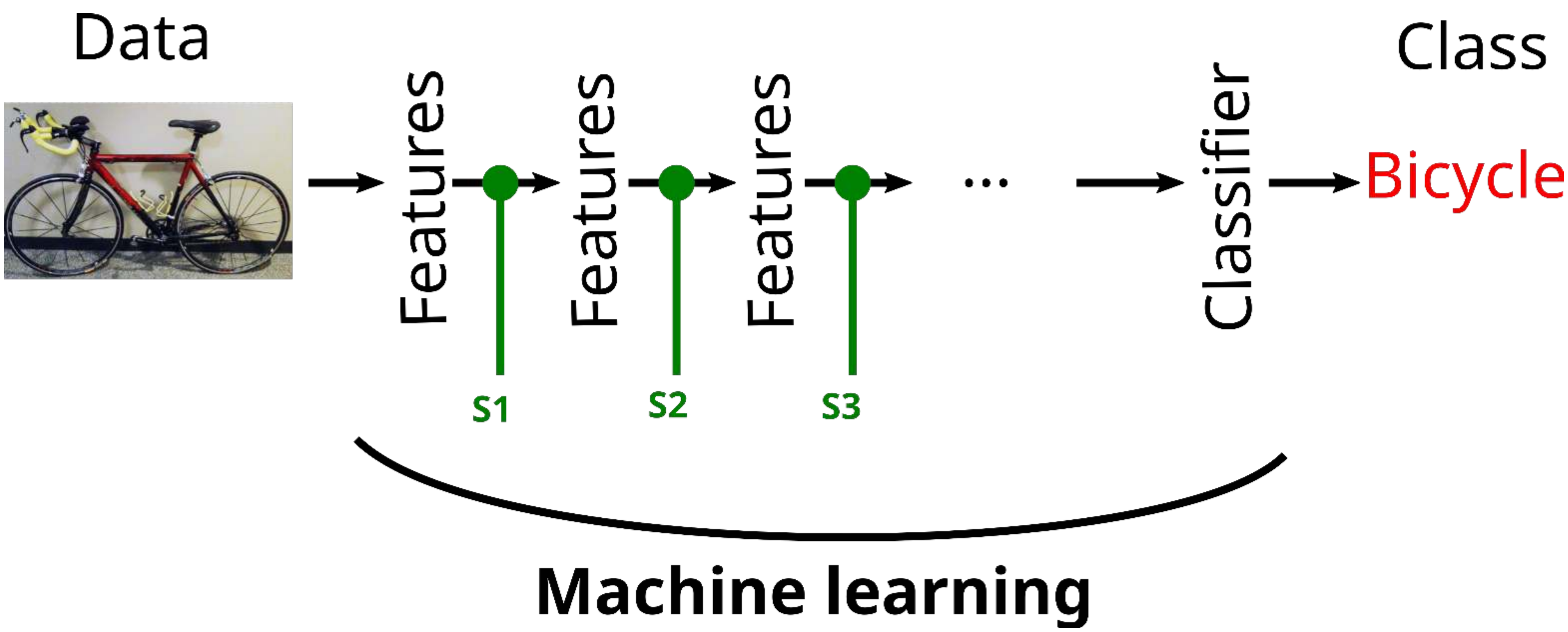
Deep learning concept



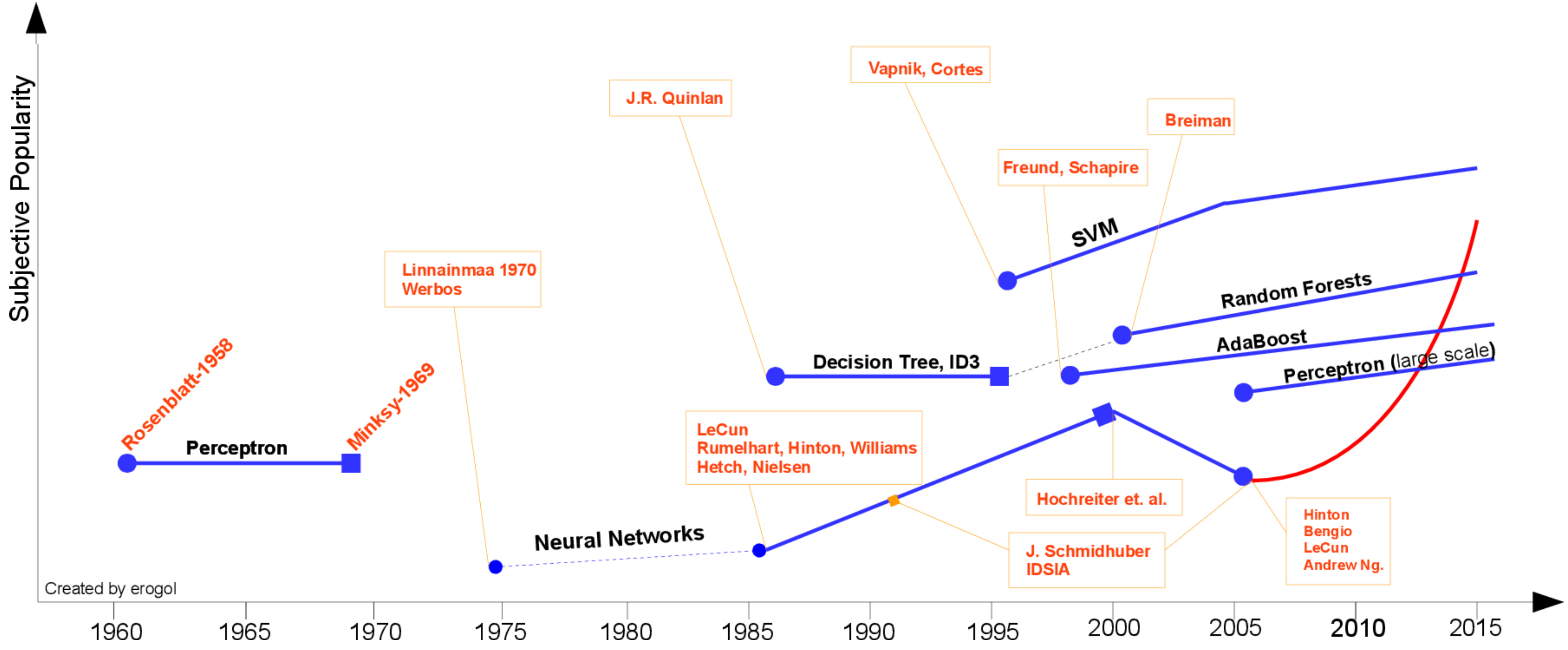
Deep learning concept



Deep learning concept



Massively data driven approaches



Convolutions andimage processing

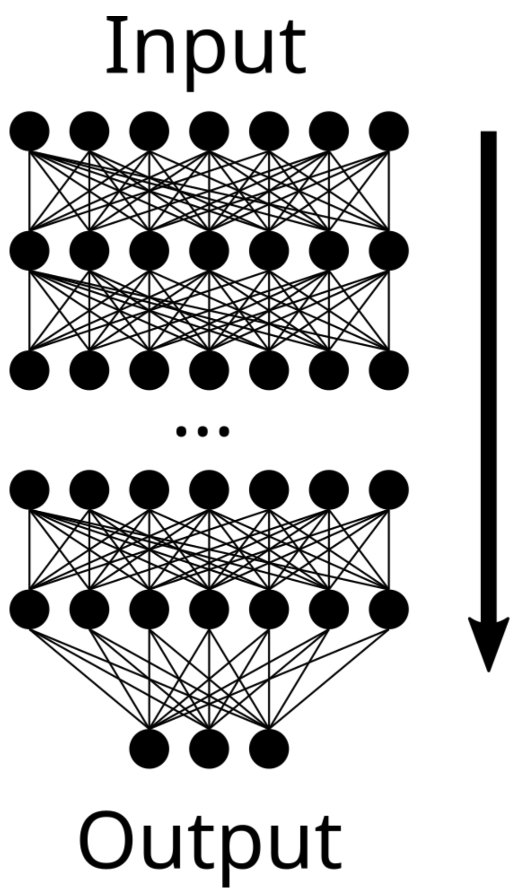
Multi-layer perceptron (before 1990)

MLP becomes larger and deeper

- difficult convergence
- few data
- very long training
- progessive loss of interest

SVM

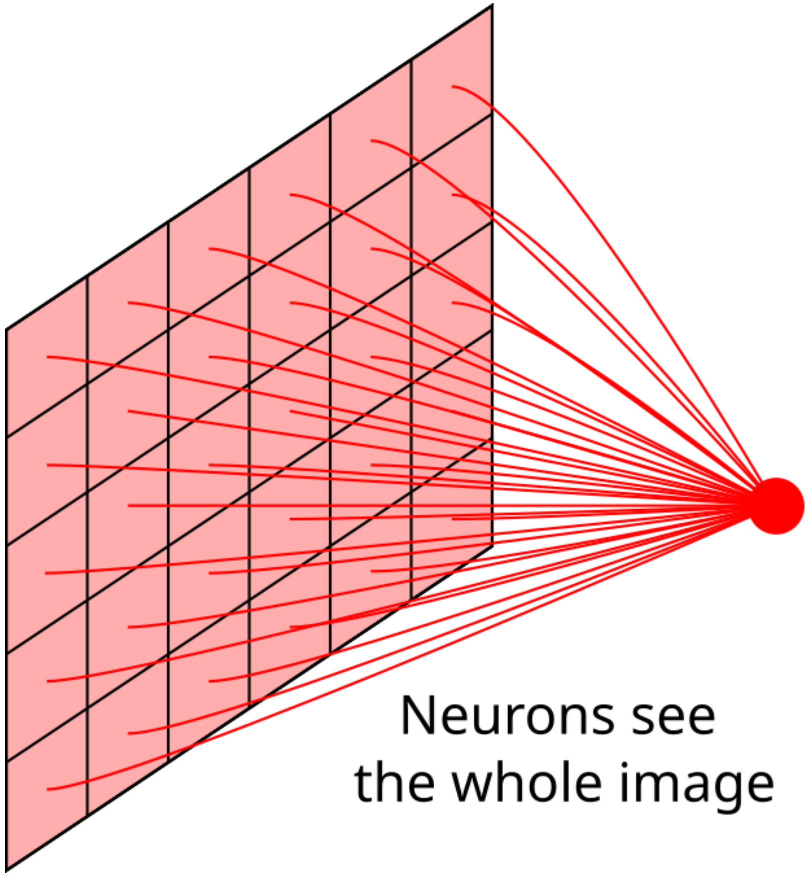
- simpe to use
- convergence proof
- fast



Multi-layer perceptron for images

Using a linear layer ?

Lots of weights! (at least one per pixel!)

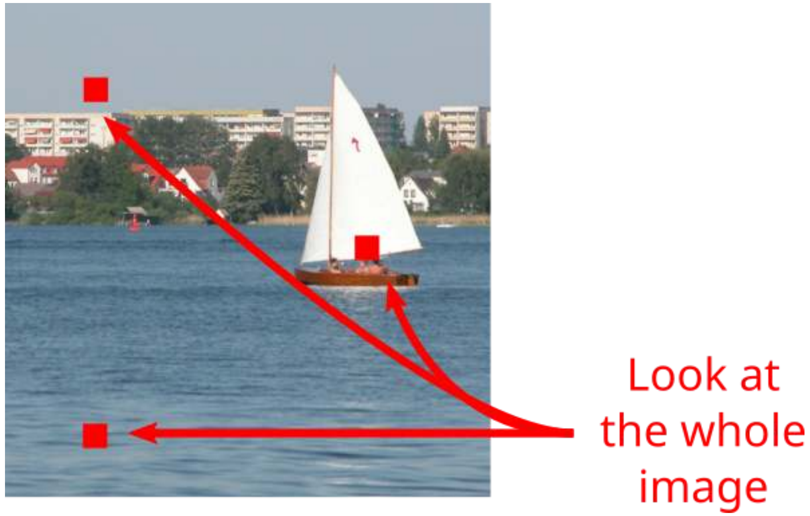


Multi-layer perceptron for images

Using a linear layer ?

Lots of weights! (at least one per pixel!)

Is it interesting to look at relations in the whole image ?



https://www.flickr.com/photos/nathelbiya/17050330

Convolution

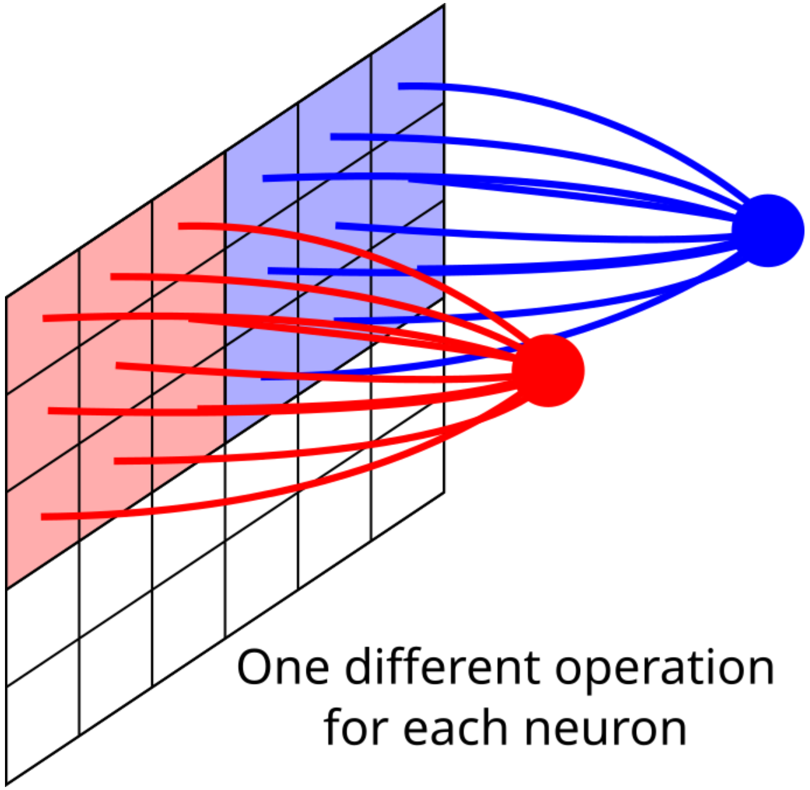
Look at small neighborhoods (where the objects are)

https://www.flickr.com/photos/nathelbiya/17050330

Convolution

Look at small neighborhoods (where the objects are)

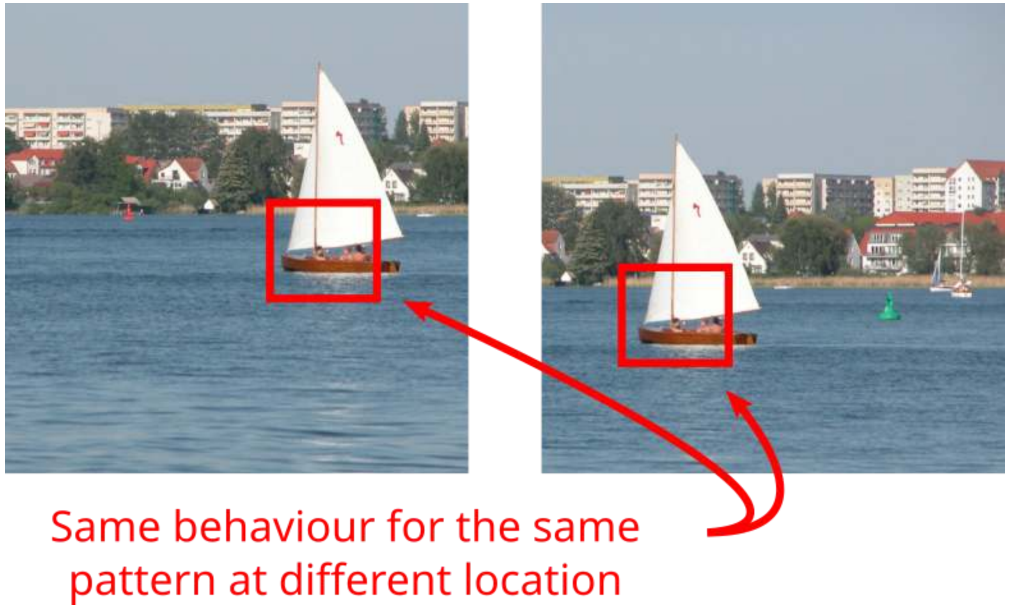
Create neurons that take a patch input



Convolution

Problem

Translation of the object must lead to same behaviour of the neurons



https://www.flickr.com/photos/nathelbiya/17050330

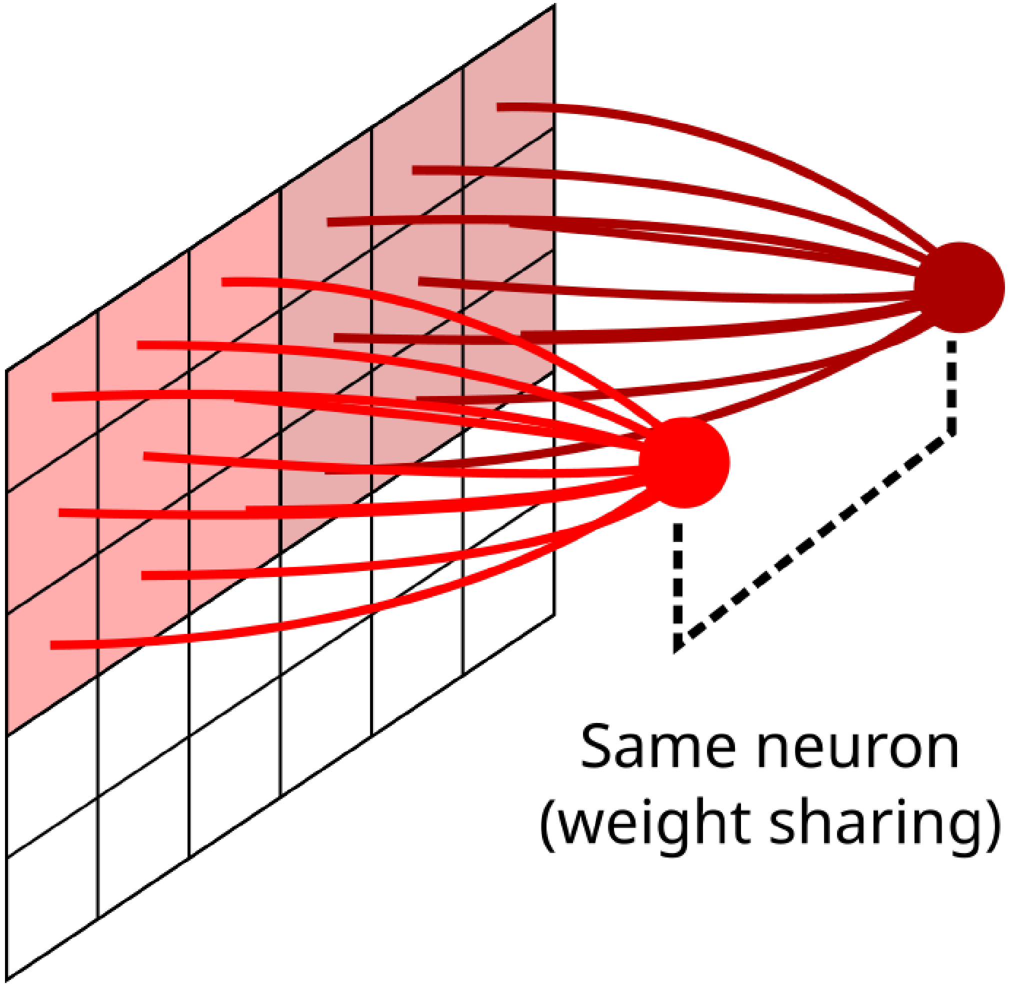
Convolution

Problem

Translation of the object must lead to same behaviour of the neurons

Solution

Use the same neuron (i.e. all the neurons of the layer share weights)



Convolution

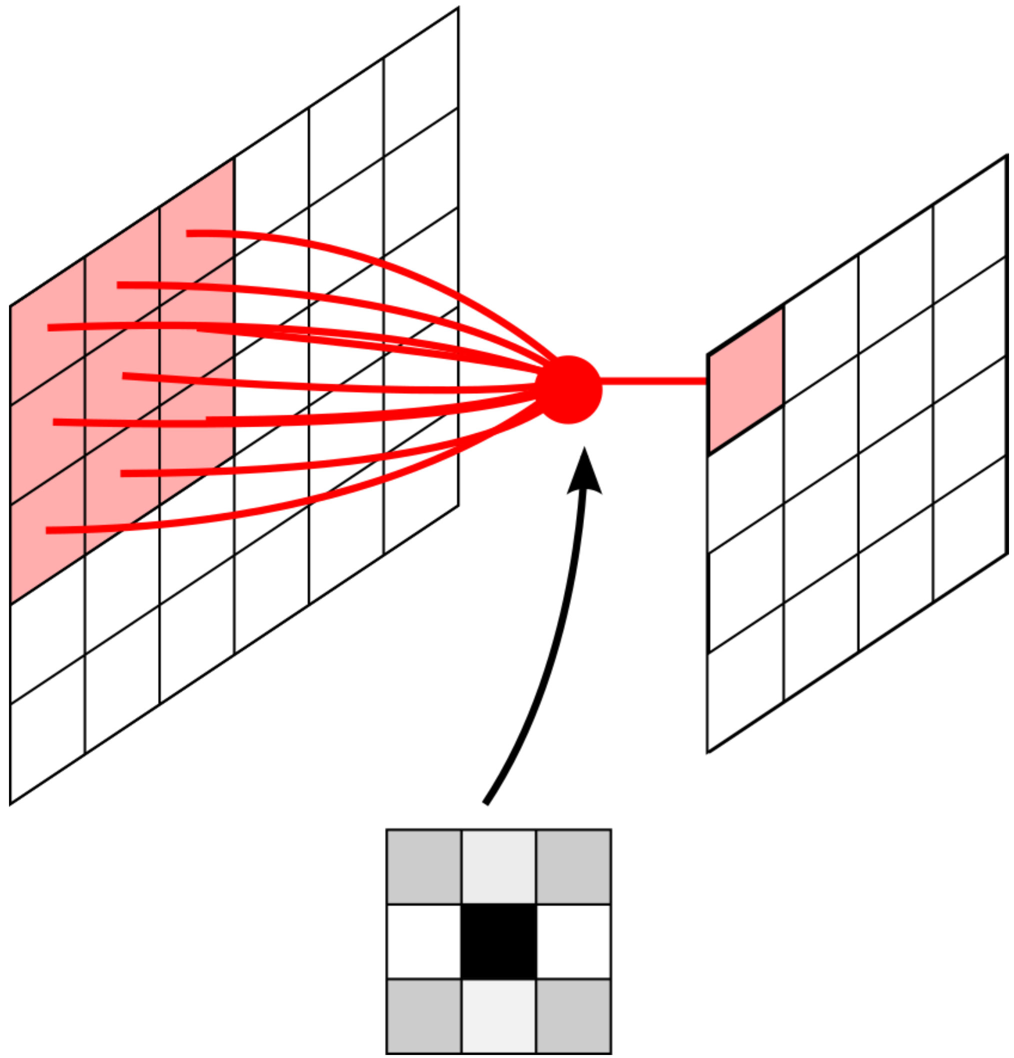
Forward

Let (i, j) be the coordinates in the input map.

(k, l) be the size of the patch (size of the kernel, usually $k = l$)

Then:

$$y_{i,j} = \sum_k \sum_l w_{k,l} x_{i+k,j+l} + b$$



Convolution

Backward weight update

$$\frac{\partial y_{i,j}}{\partial w_{k,l}} = x_{i+k,j+l}$$

Let y be the output map and Δy be the gradient coming back:

$$\frac{\partial y}{\partial w_{k,l}} = \sum_i \sum_j x_{i+k,j+l}$$

Finally, the update rule:

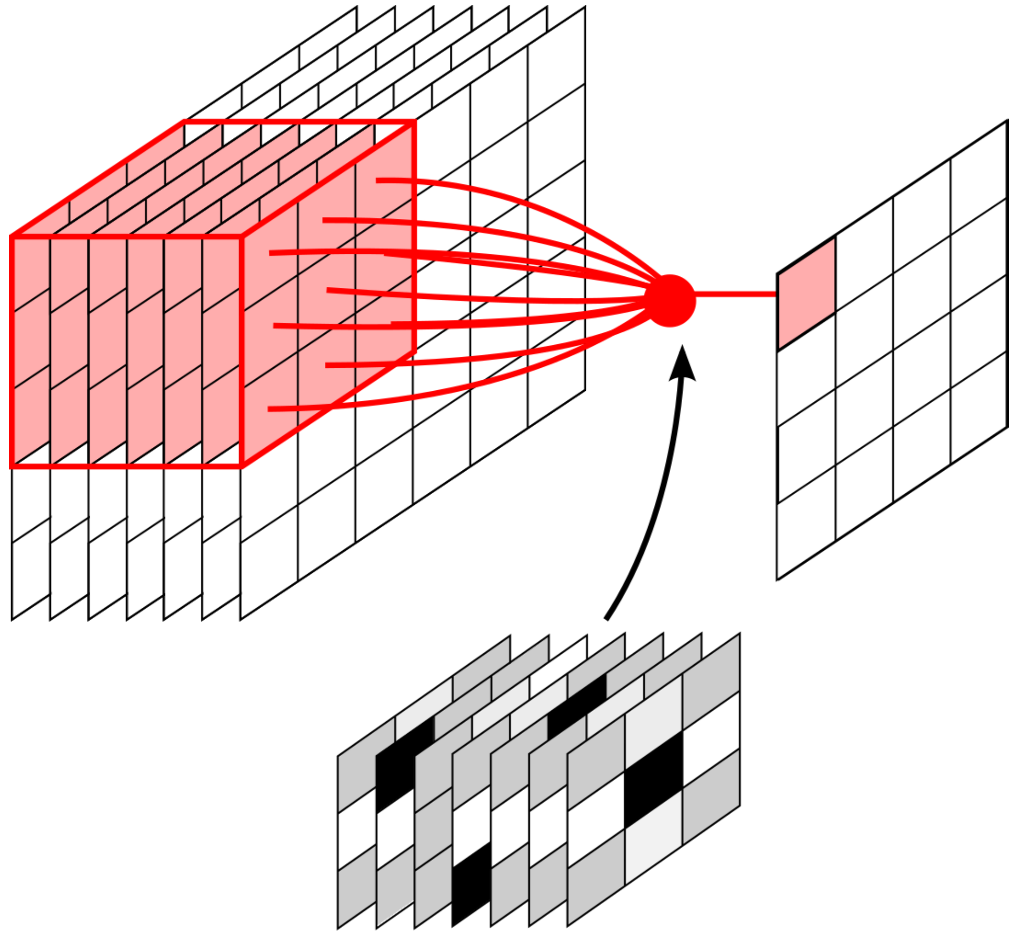
$$\begin{aligned} w_{k,l} &\leftarrow w_{k,l} - \alpha \frac{\partial y}{\partial w_{k,l}} \Delta y \\ &\leftarrow \sum_i \sum_j x_{i+k,j+l} \Delta y_{i,j} \end{aligned}$$

Convolutions

Forward

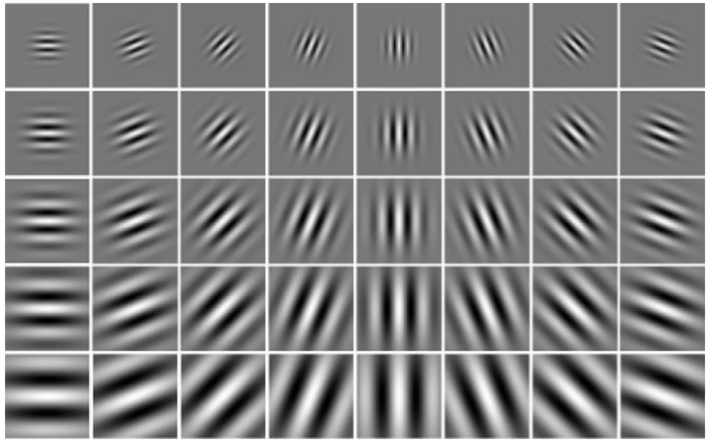
Same with term to term multiplication:

$$y_{i,j} = \sum_a \sum_b \mathbf{w}_{a,b} \mathbf{x}_{i+a,j+b}$$

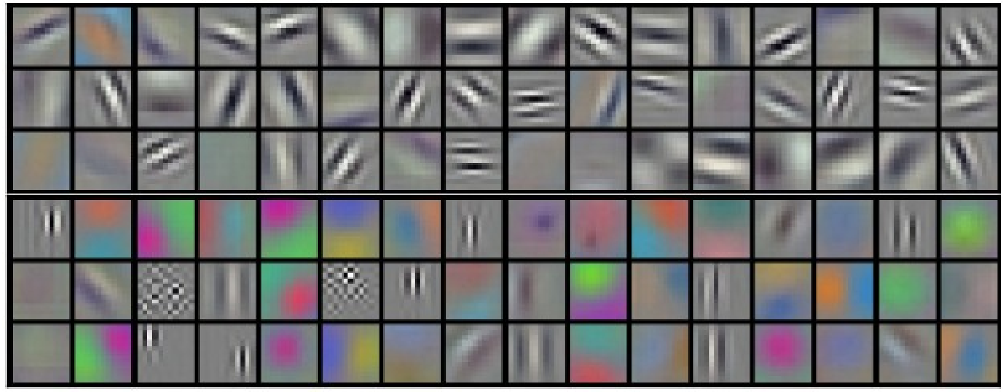


valeo.ai

Convolution: what do convolutions learn?



Gabor filters.



First layer of AlexNet.

Dimension reduction

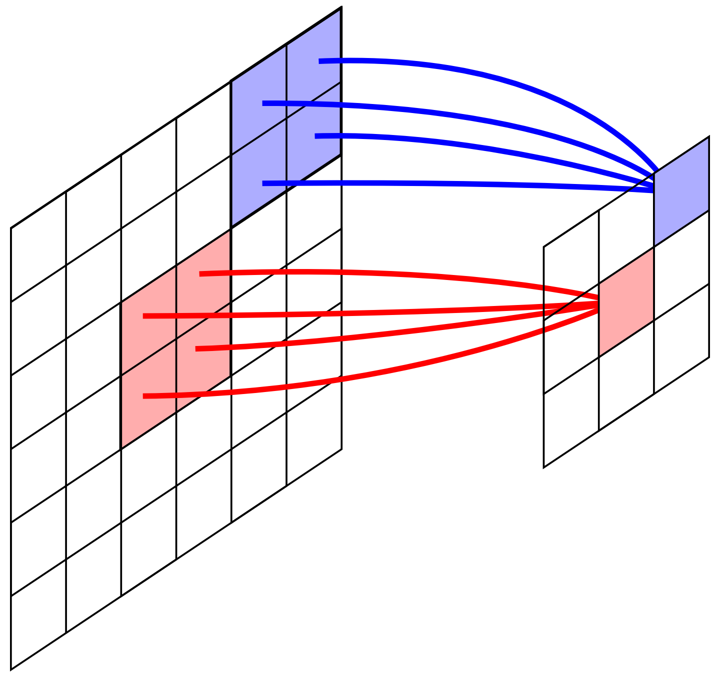
With the previous convolution, the output dimension is the same as the output dimension.

For classification: only one label, need for \textbf{dimension reduction}.

- **convolution stride**: do not look at all the pixels of the input (one every two, one every three...)
- **Max Pooling**

Max Pooling

- dimension reduction
- relative translation invariability



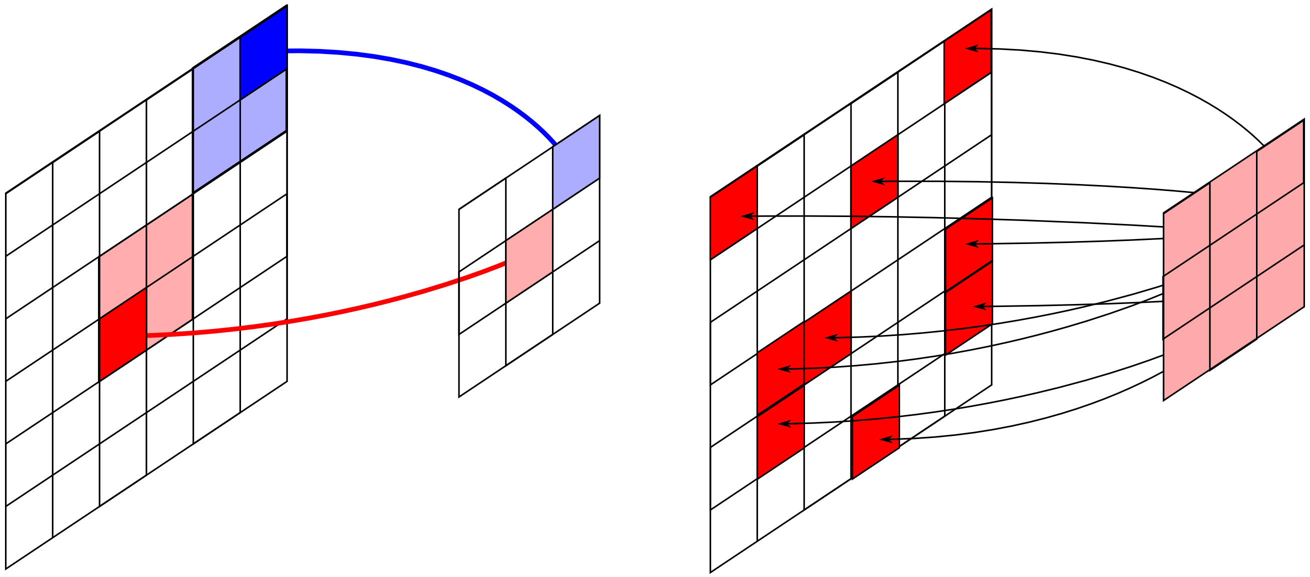
Max Pooling

Forward

Max signal

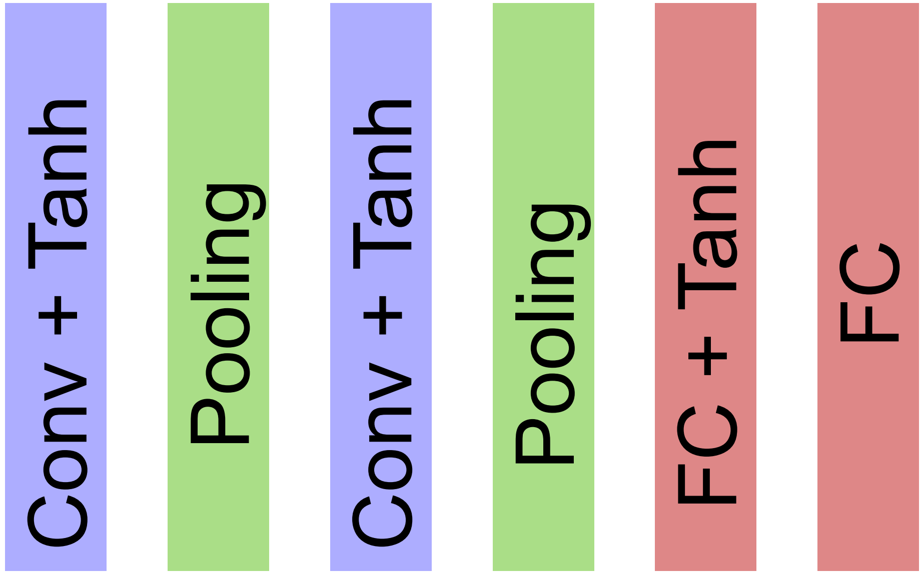
Backward

Gradient transmission to max signal origin, zero otherwise



Convolutional Neural Netowkrs - LeNet (1990

LeNet (1990)
Images 28x28



Very good results on digits recognition !

Isues

Issues

- Learning speed
- Exploding or vanishing gradients
- Overfitting
- Local minima

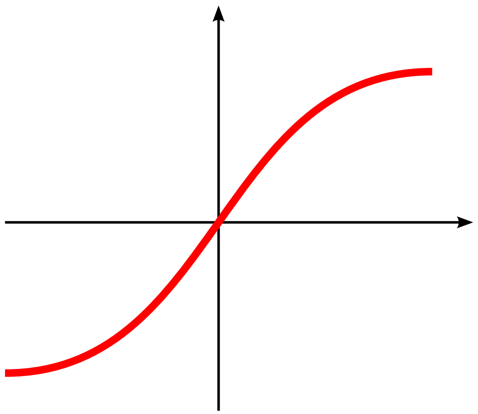
Limitations

- Architecture
- Initialization
- Computing power
- Data
- Optimization

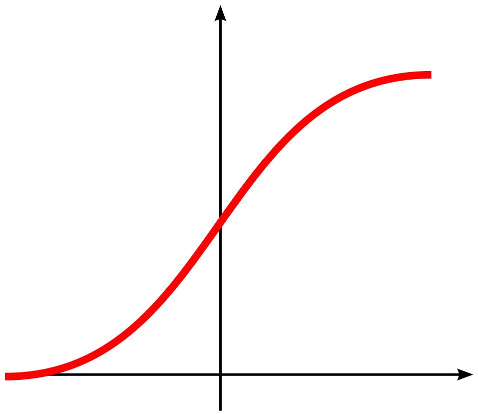
Solutions

- activations
- mini-batches
- batch norm
- good weight initialization
- better optimization

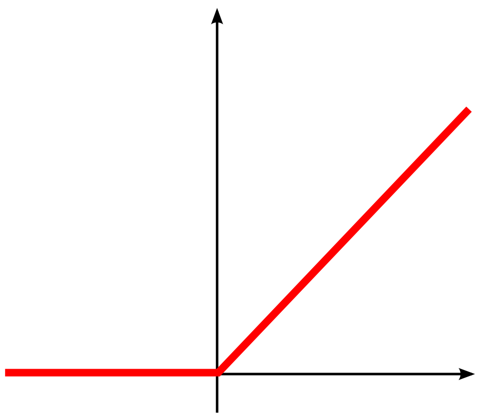
Activations



Tangente
hypebolique



Sigmoïde

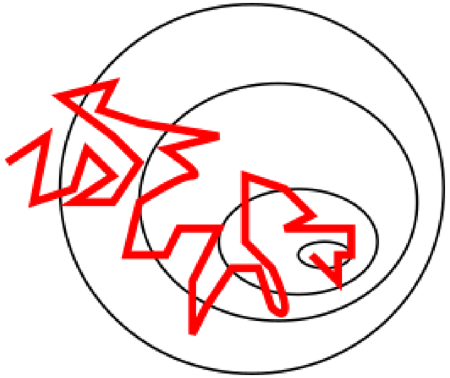


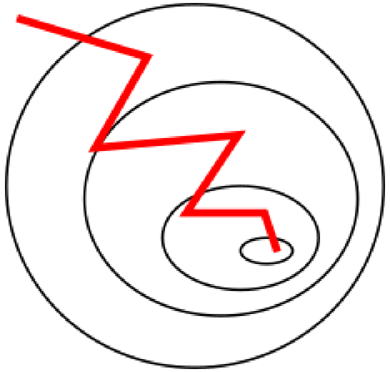
Rectified Linear
Unit (ReLU)

Rectified linear unit}

- Faster gradient computation
- Similar convergence

Mini-batches



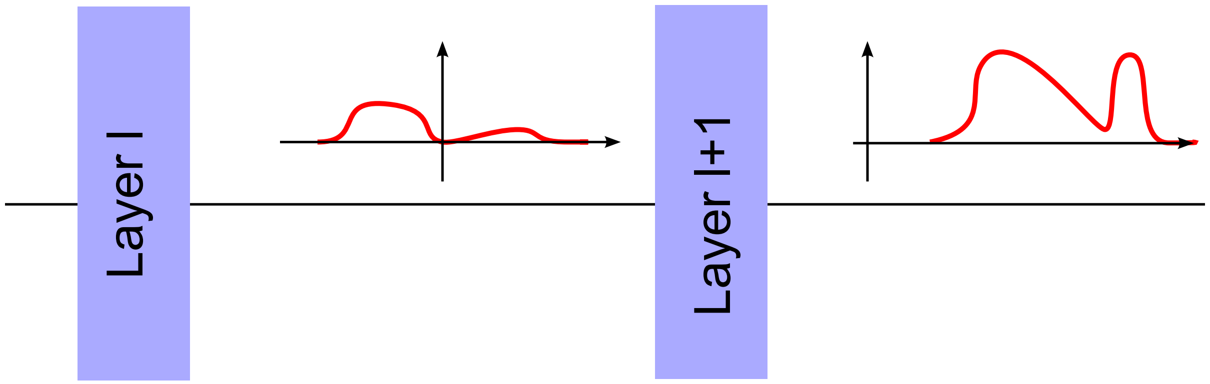


Gradient smoothing

Smoother gradient converges faster.

valeo.ai

BatchNorm



Changes in the signal dynamic make the model more difficult to optimize: exponential or vanishing gradients.

Objective: control the signal distribution:

$$y^{l*} = \frac{y^l - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta$$

γ and β are learnt, μ and σ are computed (mean and standard deviation).

Learning is faster (iteration number) but slower (statistics computation).

Weight initialization}

Weights have great influence on convergence speed.

They are randomly initialized.

- too small weigths: vanishing signal
- to high: exploding signal

Conservation of signal properties.

$$Var(Y) = Var(X)$$

Weight initialization

Xavier initialization

$X \in \mathbb{R}^n$, weights W and output $Y \in \mathbb{R}$

$$Y = W_1 X_1 + W_2 X_2 + \cdots + W_n X_n$$

X_i and W_i independent:

$$Var(W_i X_i) = E[X_i]^2 Var(W_i) + Var(X_i) Var(W_i) + Var(X_i) E[W_i]^2$$

$E[X_i] = 0$ and $E[W_i] = 0$:

$$Var(W_i X_i) = Var(X_i) Var(W_i)$$

finally $ Var(Y) = n Var(X_i) Var(W_i)$ and we chose

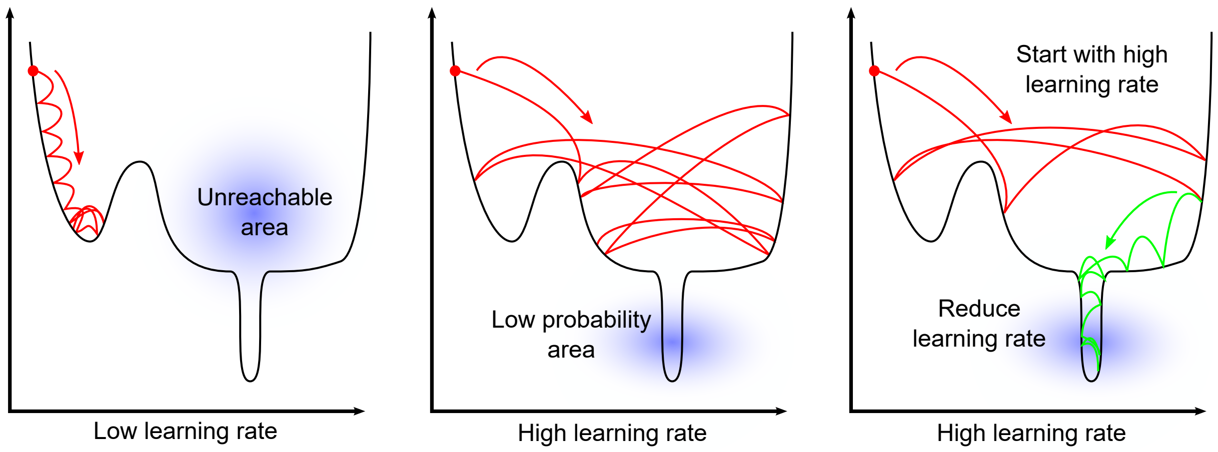
$$Var(W_i) = \frac{1}{n}$$

valeo.ai

Optimization - Stochastic Gradient Descent

See neural network class

Update rule: $w_{t+1} = w_t + \alpha \Delta w$ (learning rate α)



- Step decrease
- Exponential decrease
- Cosine annealing …

Optimization - SGD with Momentum

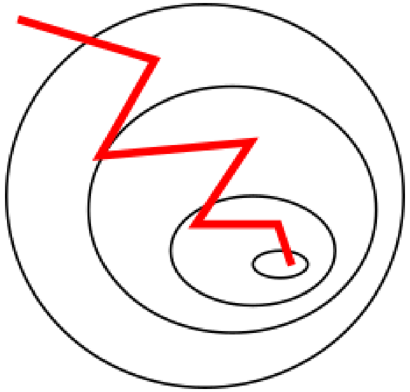
Same idea as mini batch: smooth gradient in the good direction

Momentum

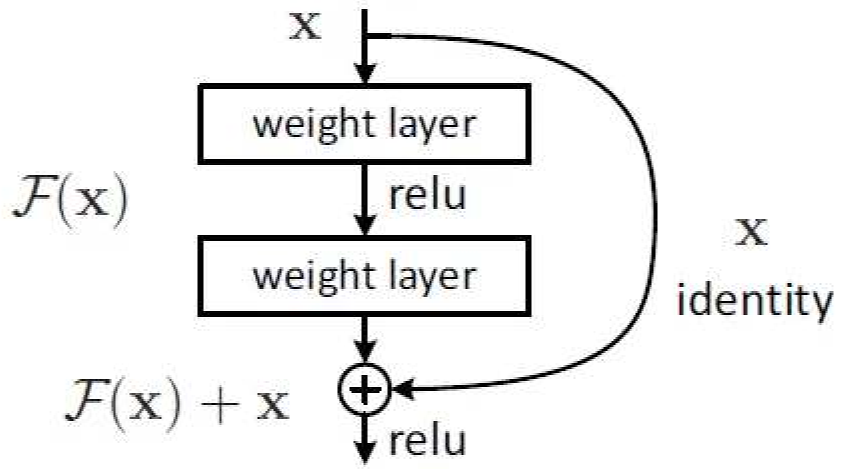
Use previous gradient to ponderate the direction of the new gradient.

$$
\begin{aligned}
v_t &= \gamma v_{t-1} + \alpha \Delta w \\
w_t &= w_{t-1} - v_t
\end{aligned}
$$

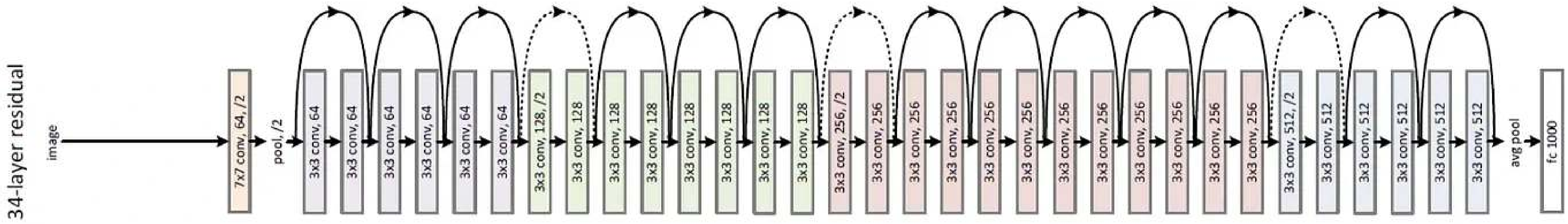
γ is the momentum.



ResNet



x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu



34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000

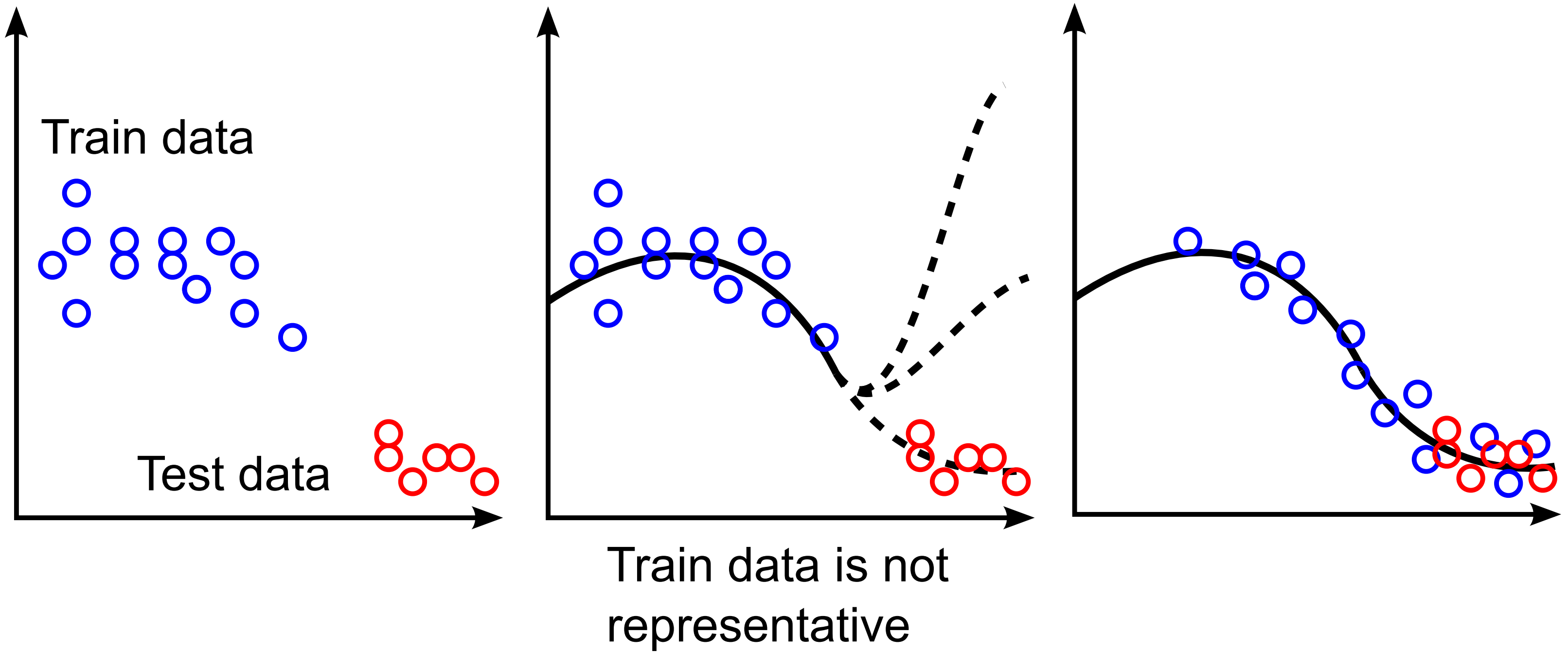
Do not forget the classics

Data

- Representative
- Data augmentation
- Data normalization

Data

Train data must be representative of the problem}



Data

Data normalization

-Compute mean μ and standard deviation σ on the train set.

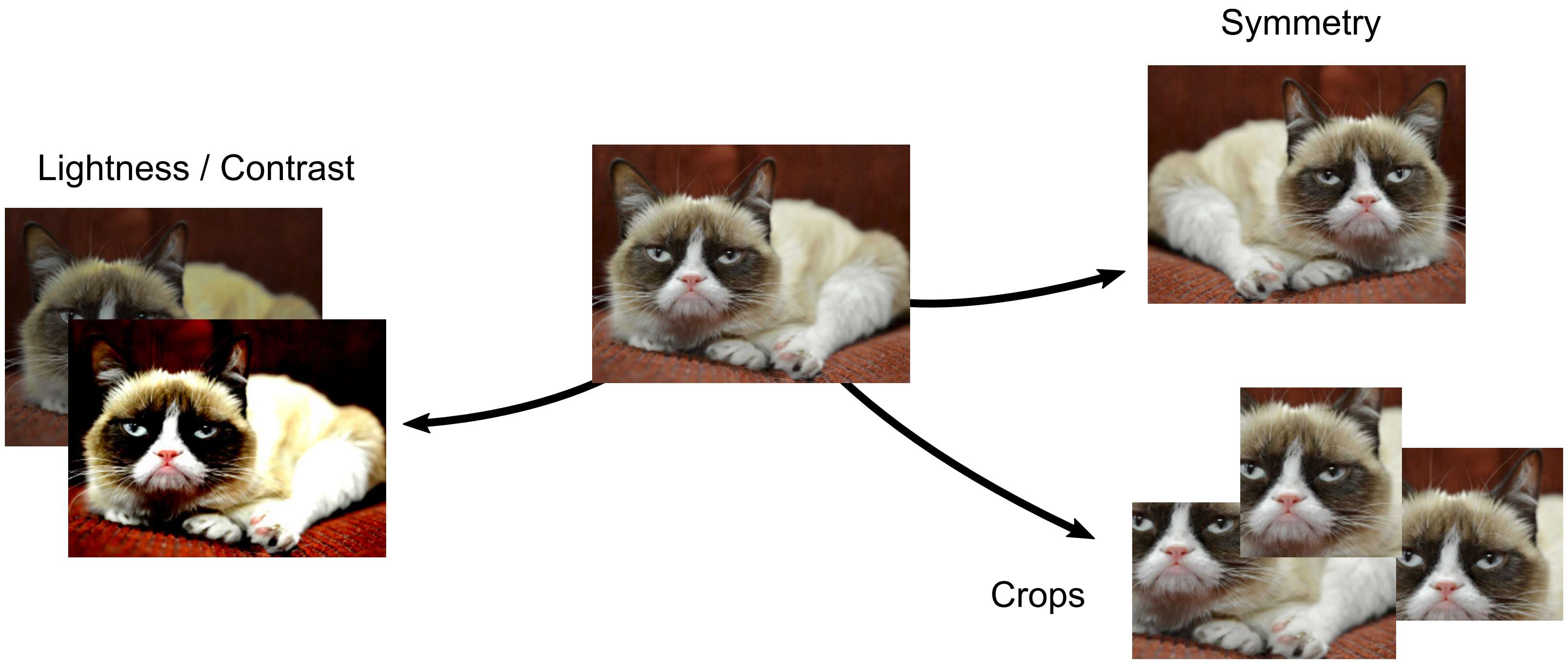
-Normalize input I (train and test):

$$\hat{I} = \frac{I - \mu}{\sigma}$$

Data - data augmentation

Random variations of input parameters (images: lightness, contrast \dots)

- train on a more representative set
- avoid learning on unwanted features



Problems and partial solutions

Problems

- Small amount of data
- Low computational power

Solutions ?

- Use classical approaches (Perceptron, SVM, …)