Deep learning
The basics

Mathieu Lefort

October 11 & 25, 2023

Content

Outline

- General principles of deep learning (neurons, layers, chain rule, etc.)
- (Multilayer) Perceptron
- Convolutional neural network
- General overview of deep learning architectures and "Tips & Tricks"
- Limits of deep learning

Objectives

- To understand the principles, properties and limits of the models
- To practice deep learning (tuning, framework, ...) (graded practical work)

Learning (theory)

Question

What to expect from a learning system ? Why ? How ?

Learning (theory)

Question

What to expect from a learning system ? Why ? How ?

Objectives

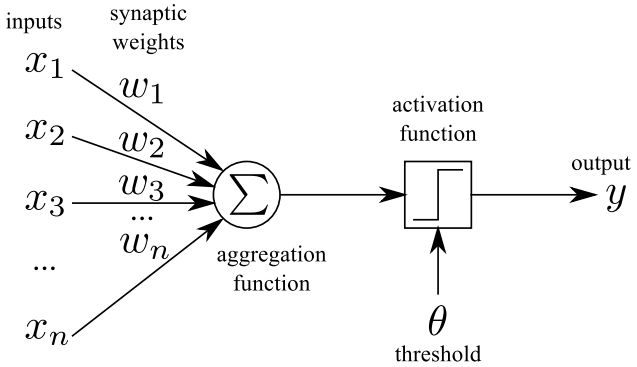
- memorization : finding (back) the answer to a previously seen input
- generalization : finding the answer to an unknown input
- hypothesis : locally continuous function, Occam razor
- methods : train/validation/test datasets, regularisation, ...

Types

- supervised : $y = f(x)$ with $x \in \mathcal{X}$ et $y \in \mathcal{Y}$
  - regression : when $\mathcal{Y}$ is continuous, usually $\mathbb{R}^n$
  - classification : when $\mathcal{Y}$ is discrete
  - in pratice $f(x) = \sum_{i=1}^{n} (a_i^T.x + b_i)\phi(x, \theta_i)$ with ϕ a non linear function parameterized by θ_i, $b_i \in \mathbb{R}^{card(\mathcal{Y})}$ and $a_i \in \mathbb{R}^{card(\mathcal{Y}) \times card(\mathcal{X})}$
- unsupervised : $f(x)$ with $x \in \mathcal{X}$
  - clustering : clusters of inputs
  - generative : learning of the input distribution $p(x)$ (e.g. $x = f(\tilde{x})$)
- reinforcement learning,, ...

Artificial neuron - McCulloch et Pitts (1943)

Biological inspiration :

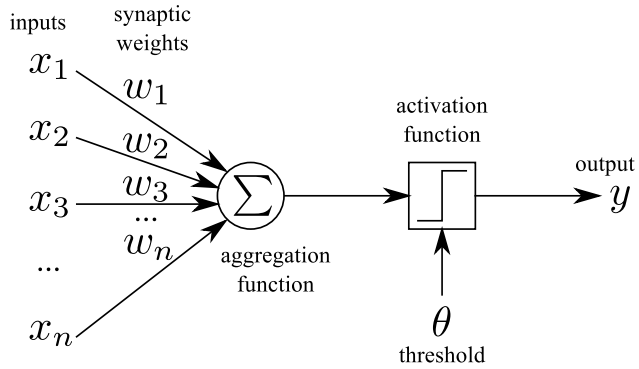


Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i - \theta > 0 \\ 0 & \text{else} \end{cases}$

Artificial neuron - McCulloch et Pitts (1943)

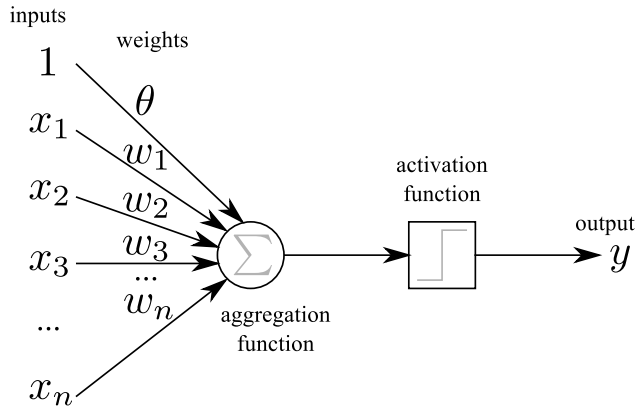
Biological inspiration :

$$\underline{\textbf{Neuron output}} : y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i - \theta > 0 \\ 0 & \text{else} \end{cases}$$



Model in machine learning :

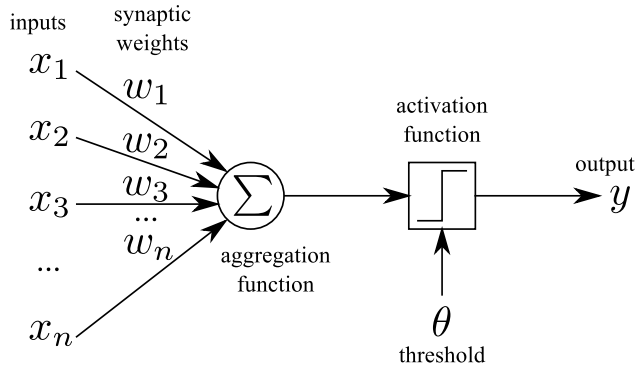
$$\underline{\textbf{Neuron output}} : y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$$



Artificial neuron - McCulloch et Pitts (1943)

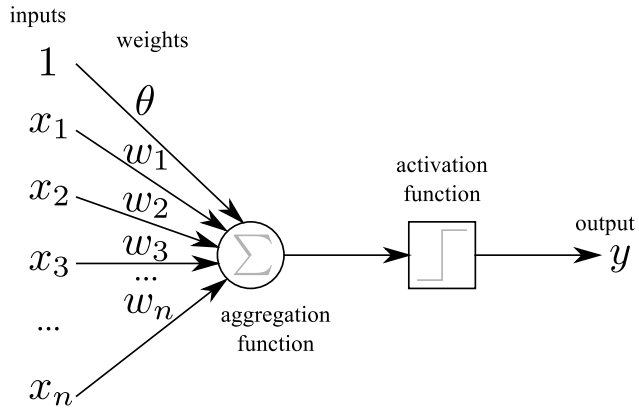
Biological inspiration :

$$\underline{\textbf{Neuron output}} : y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i - \theta > 0 \\ 0 & \text{else} \end{cases}$$



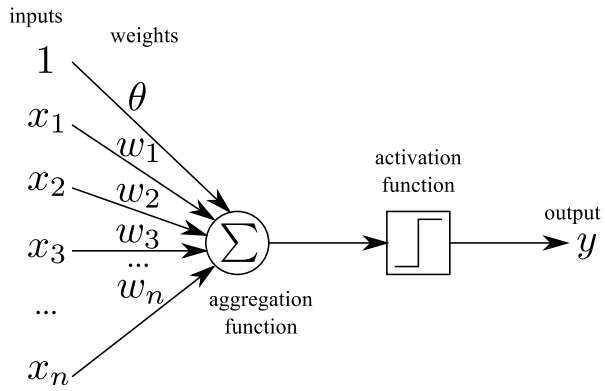
Model in machine learning :

$$\underline{\textbf{Neuron output}} : y = \begin{cases} 1 & \text{if } \mathrm{w}^T \mathrm{x} > 0 \\ 0 & \text{else} \end{cases}$$



Perceptron - Rosenblatt (1957)

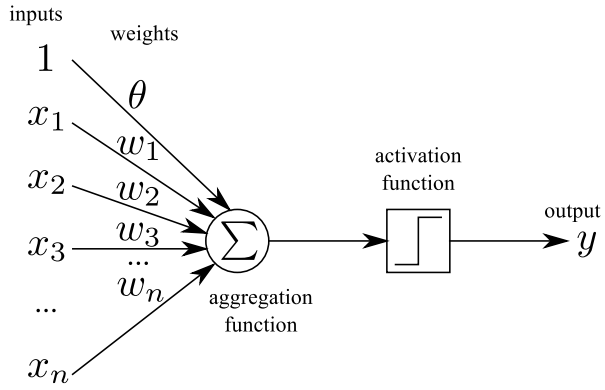
$$\underline{\textbf{Neuron output}} : y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$$



Perceptron - Rosenblatt (1957)

Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

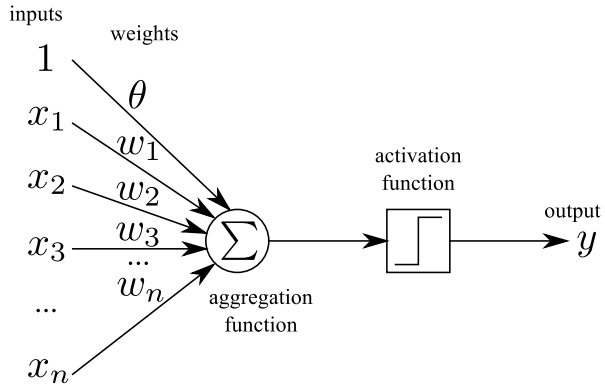
Learning rule : $\Delta W = \eta X(t - y)$



Perceptron - Rosenblatt (1957)

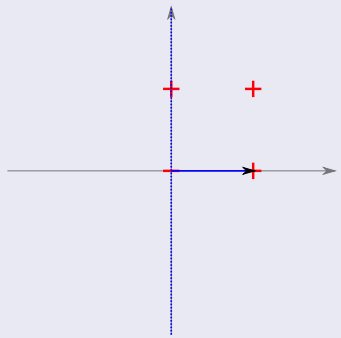
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

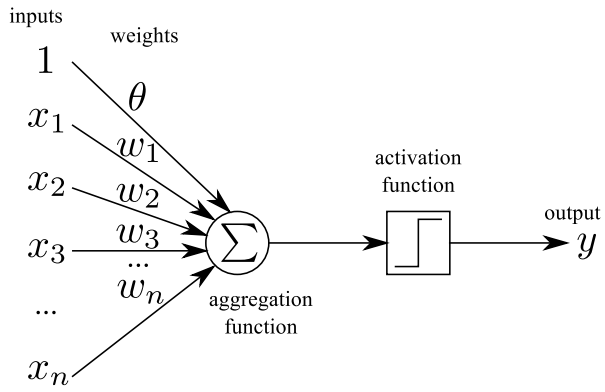
Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	0	0	0	1	0	0	0



Perceptron - Rosenblatt (1957)

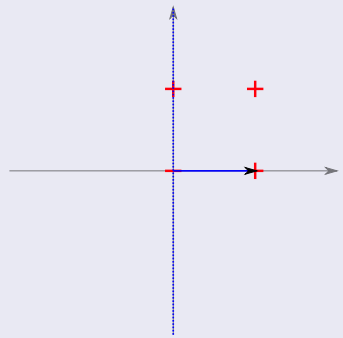
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

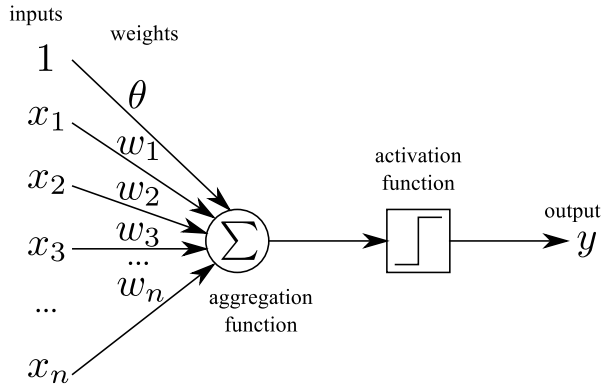
Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	0	1	0	1	0	0	1



Perceptron - Rosenblatt (1957)

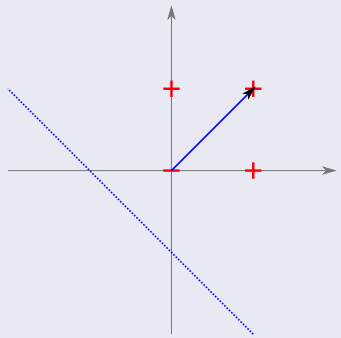
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

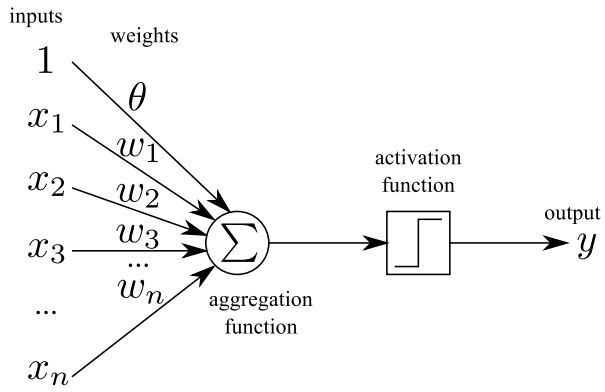
Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	0	1	1	1	1	0	1



Perceptron - Rosenblatt (1957)

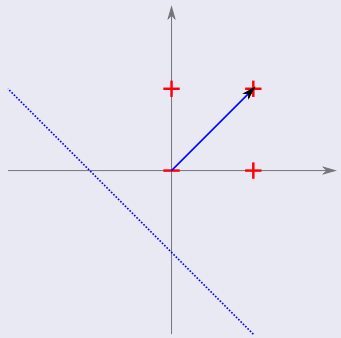
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

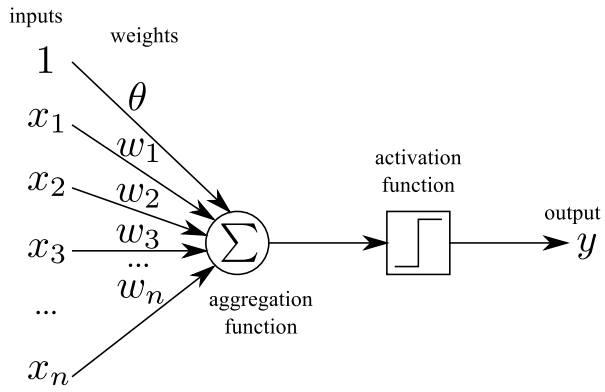
Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	1	0	1	1	1	1	1



Perceptron - Rosenblatt (1957)

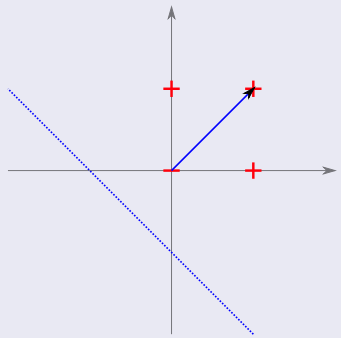
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

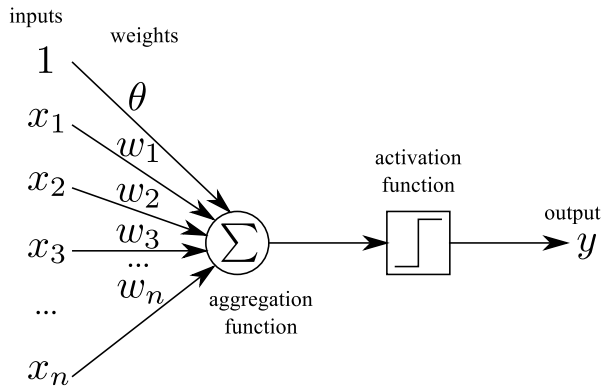
Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	1	1	1	1	1	1	1



Perceptron - Rosenblatt (1957)

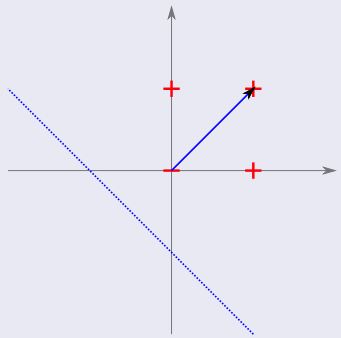
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

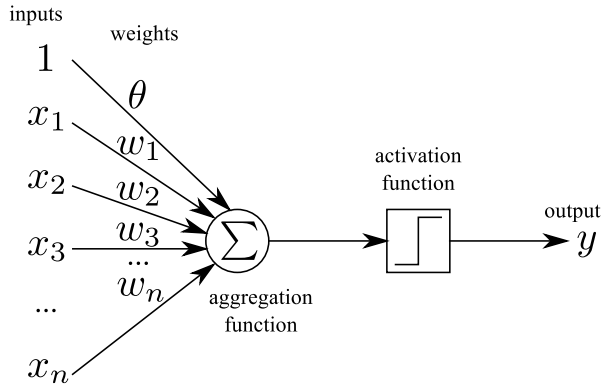
Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	0	0	1	1	1	1	0



Perceptron - Rosenblatt (1957)

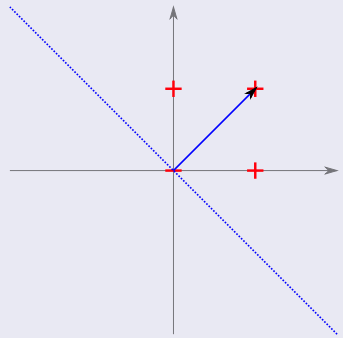
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

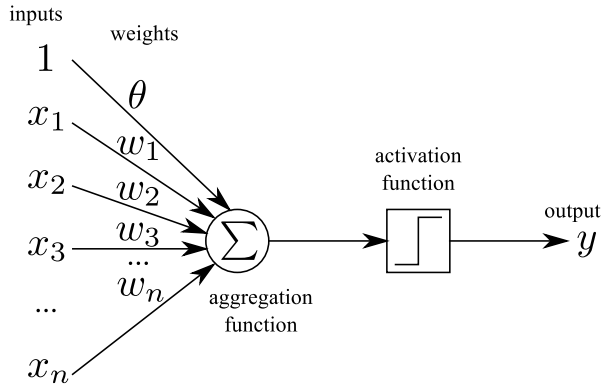
Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	0	0	0	1	1	1	0



Perceptron - Rosenblatt (1957)

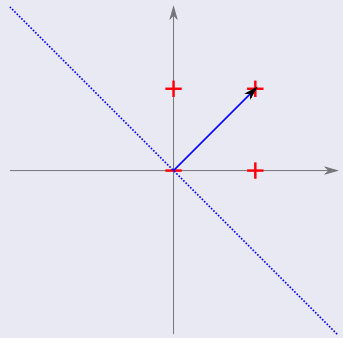
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

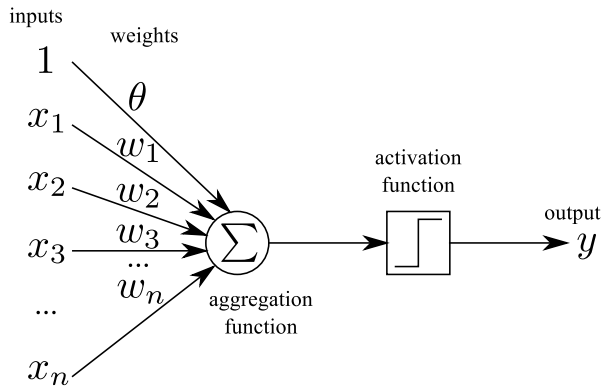
Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	0	1	0	1	1	1	1



Perceptron - Rosenblatt (1957)

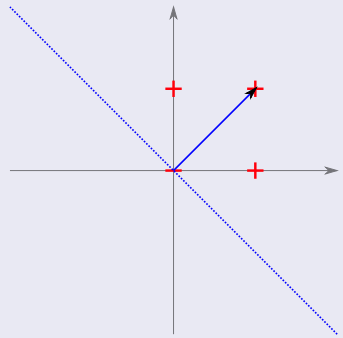
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

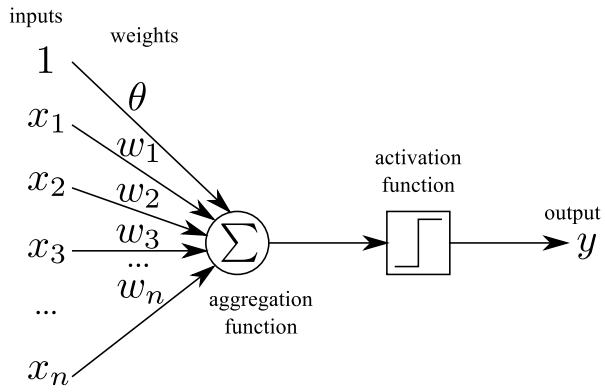
Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	1	0	0	1	1	1	1



Perceptron - Rosenblatt (1957)

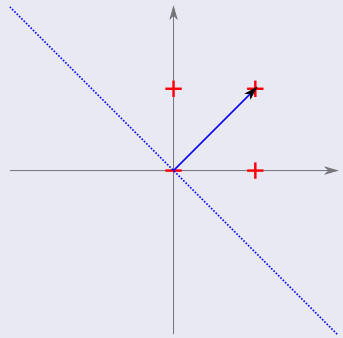
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

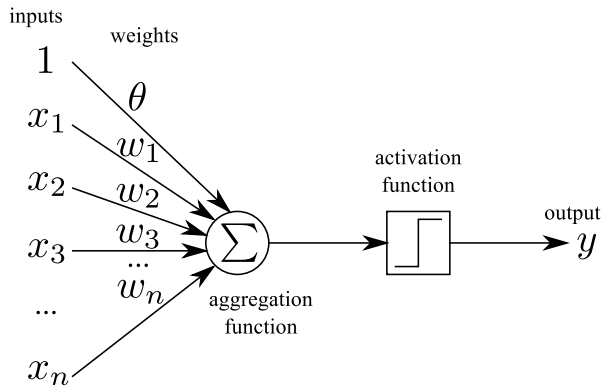
Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	1	1	0	1	1	1	1



Perceptron - Rosenblatt (1957)

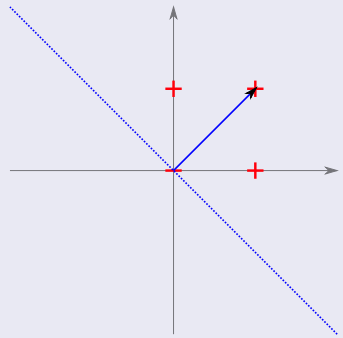
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

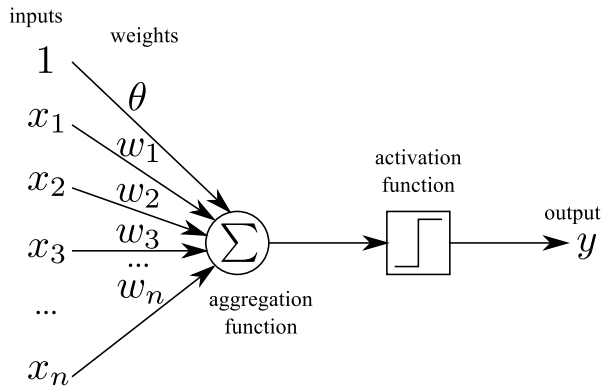
Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	0	0	0	1	1	0	0



Perceptron - Rosenblatt (1957)

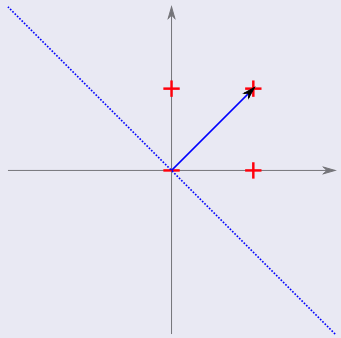
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	0	0	0	1	1	0	0



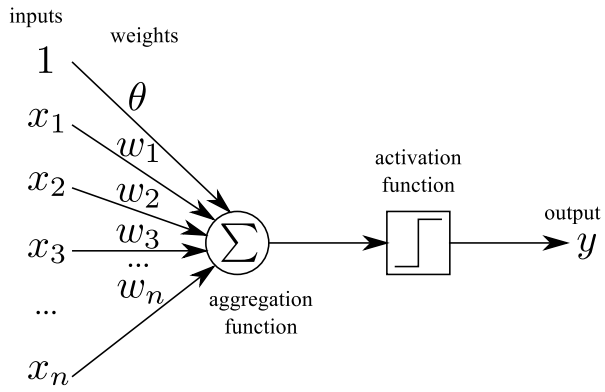
Properties

- Linear classifier (separates the space with an hyperplan)
- Weight = vector orthogonal to the hyperplan, Bias = Y-intercept
- Converges if η is (infinitesimally) small and if data are linearly separable

Perceptron - Rosenblatt (1957)

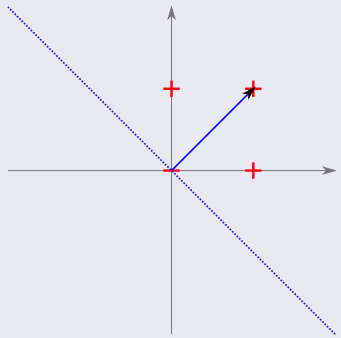
Neuron output : $y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 & \text{else} \end{cases}$

Learning rule : $\Delta W = \eta X(t - y)$



One example : OR

Input X			Weights W			Output	Target
x_0	x_1	x_2	w_0	w_1	w_2	y	t
1	0	0	0	1	1	0	0

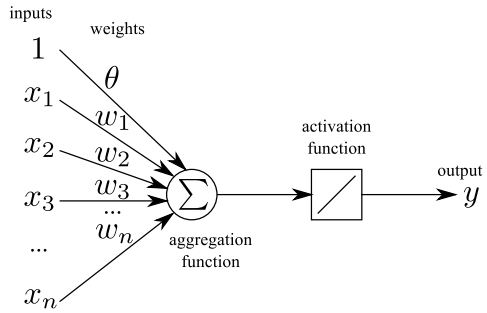


Question

What happens when there is noise / outliers (i.e. data are no more linearly separable) ?

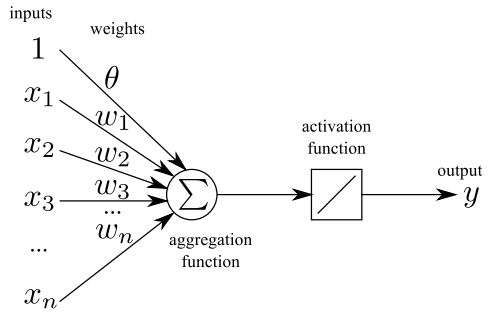
Perceptron - Stochastic gradient descent

Neuron output : $y = \sum_{i=0}^{n} w_i x_i$

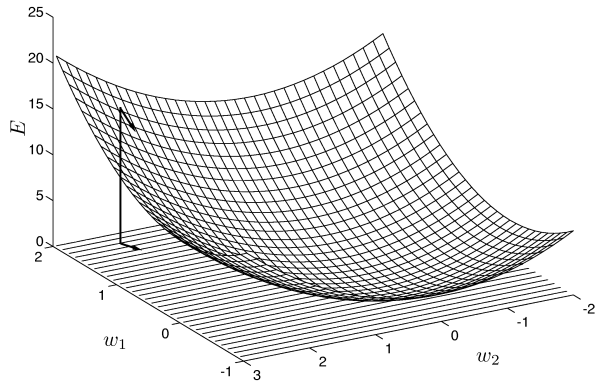


Perceptron - Stochastic gradient descent

Neuron output : $y = \sum_{i=0}^{n} w_i x_i$

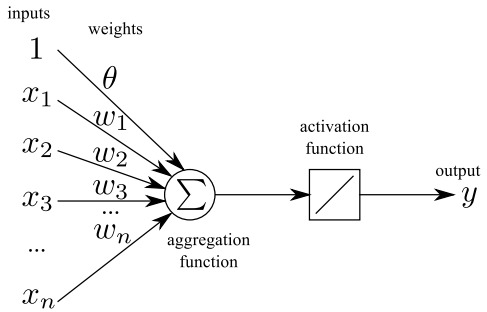


Learning rule : We want to minimise the Mean Squared Error $E = \frac{1}{2} \sum_{x} (t - y)^2$

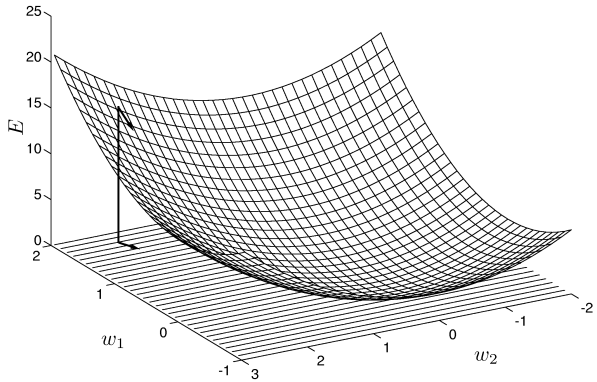


Perceptron - Stochastic gradient descent

Neuron output : $y = \sum_{i=0}^{n} w_i x_i$



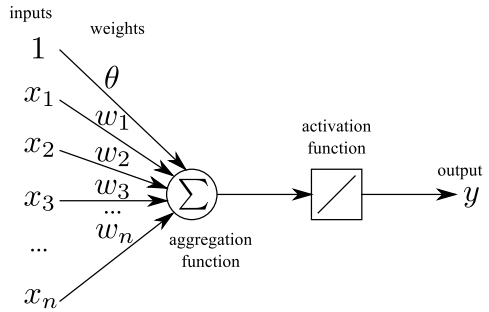
Learning rule : We want to minimise the Mean Squared Error $E = \frac{1}{2}\sum_{x}(t-y)^2$



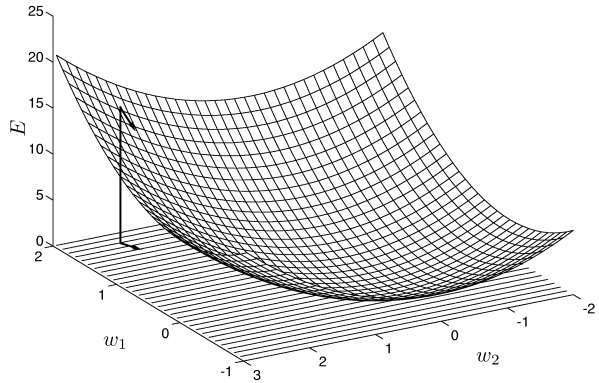
$$
\begin{aligned}
\Delta w_i &= -\eta \frac{\partial E}{\partial w_i} \\
&= -\frac{\eta}{2} \sum_{x} \frac{\partial (t-y)^2}{\partial w_i} \\
&= -\frac{\eta}{2} \sum_{x} -2 \frac{\partial y}{\partial w_i} (t-y) \\
&= \sum_{x} \eta x_i (t-y)
\end{aligned}
$$

Perceptron - Stochastic gradient descent

Neuron output : $y = \sum_{i=0}^{n} w_i x_i$



Learning rule : We want to minimise the Mean Squared Error $E = \frac{1}{2}\sum_{x}(t-y)^2$ by stochastic gradient descent : $\Delta w = \sum_{x} \eta x (t-y)$

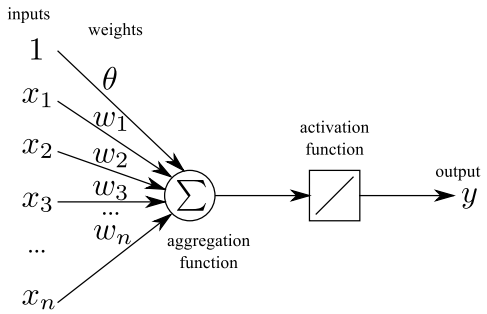


Properties

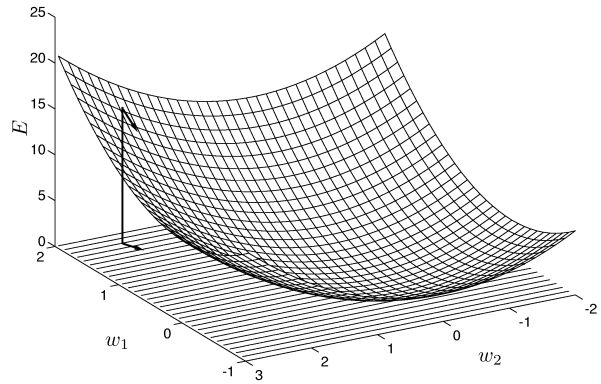
- Linear classifier
- Convergence is guaranteed if η is small (even if the data are not linearly separable)
- Optimization is efficient (as the function to optimize is quadratic)
- In practice often converges quicker with online learning but theoretically only guaranteed in batch learning

Perceptron - Stochastic gradient descent

Neuron output : $y = \sum_{i=0}^{n} w_i x_i$



Learning rule : We want to minimise the Mean Squared Error $E = \frac{1}{2}\sum_{x}(t-y)^2$ by stochastic gradient descent : $\Delta w = \sum_{x} \eta x (t-y)$



Question

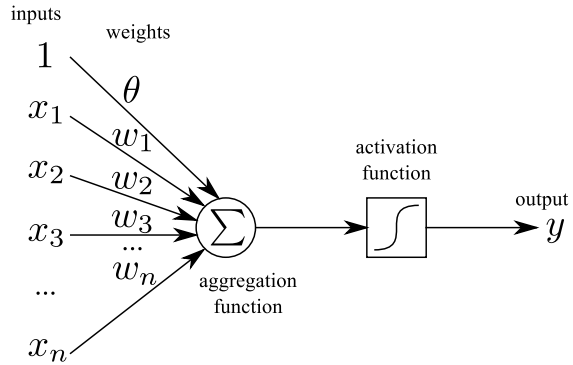
How to learn non linear functions ?

MultiLayer Perceptron - Werbos & Rumelhard (1984-1986)

Neuron output :

$$s = \sum_{i=0}^{n} w_i x_i$$

$$y = \frac{1}{1 + e^{-s}}$$

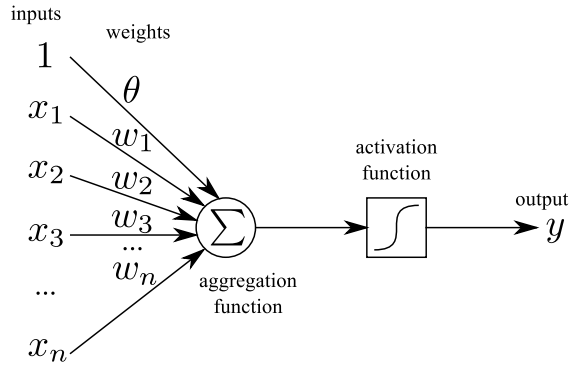


MultiLayer Perceptron - Werbos & Rumelhard (1984-1986)

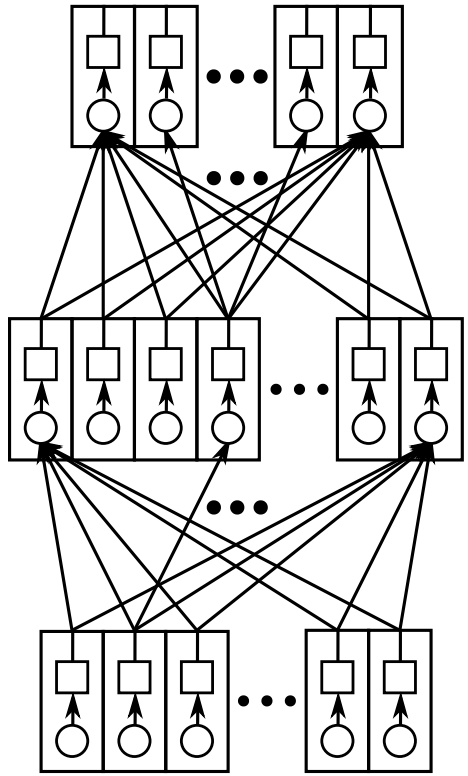
Neuron output :

$$s = \sum_{i=0}^{n} w_i x_i$$

$$y = \frac{1}{1+e^{-s}}$$



Network : full connections between input layer, hidden layer(s) and output layer



MultiLayer Perceptron - Werbos & Rumelhard (1984-1986)

Learning rule : We want to minimize the mean squared error $E = \frac{1}{2}\sum_{x}(t-y)^2$ by stochastic gradient descent.

For each neuron :

$$s = \sum_{i=0}^{n} w_i x_i$$

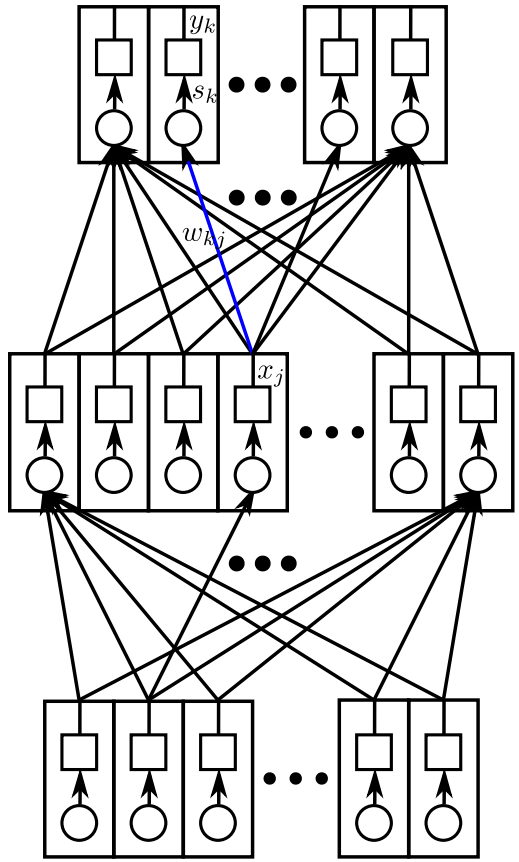
$$y = \frac{1}{1+e^{-s}}$$

Learning rule : We want to minimize the mean squared error $E = \frac{1}{2}\sum_{x}(t-y)^2$ by stochastic gradient descent.

$$\begin{aligned}\Delta w_{kj} &= -\eta \frac{\partial E}{\partial w_{kj}} \\ &= -\eta \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial w_{kj}} \\ &= \eta \delta_k x_j\end{aligned}$$

For each neuron :

$$\begin{aligned}s &= \sum_{i=0}^{n} w_i x_i \\ y &= \frac{1}{1+e^{-s}}\end{aligned}$$

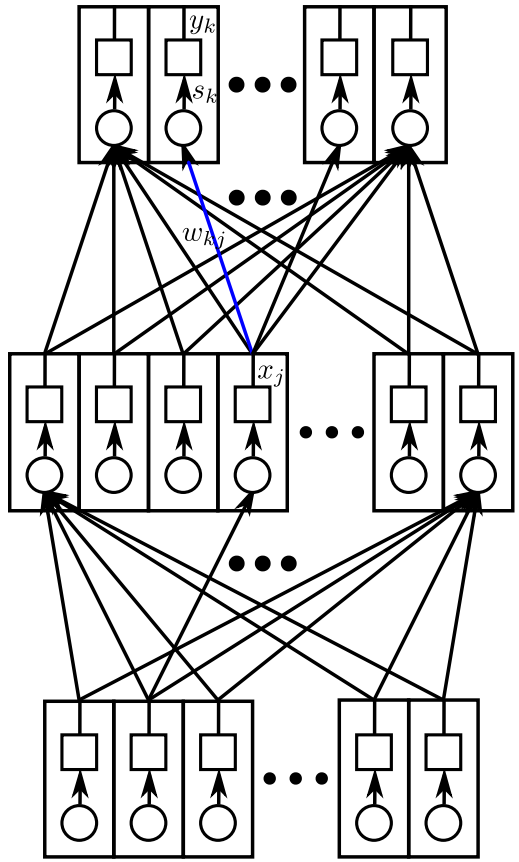


Learning rule : We want to minimize the mean squared error $E = \frac{1}{2}\sum_x (t-y)^2$ by stochastic gradient descent.

$$\begin{aligned}
\Delta w_{kj} &= -\eta \frac{\partial E}{\partial w_{kj}} \\
&= -\eta \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial w_{kj}} \\
&= \eta \delta_k x_j \\
\delta_k &= -\frac{\partial E}{\partial s_k} \\
&= \sum_x \frac{\partial y_k}{\partial s_k}(t_k - y_k) \\
&= \sum_x y_k(1-y_k)(t_k - y_k)
\end{aligned}$$

For each neuron :

$$\begin{aligned}
s &= \sum_{i=0}^{n} w_i x_i \\
y &= \frac{1}{1+e^{-s}}
\end{aligned}$$

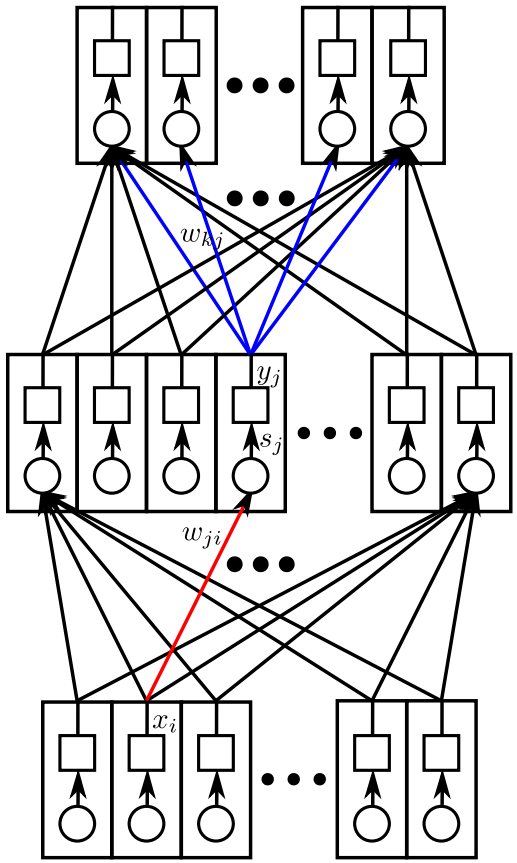


Learning rule : We want to minimize the mean squared error $E = \frac{1}{2}\sum_{x}(t-y)^2$ by stochastic gradient descent.

$$\begin{aligned}\Delta w_{ji} &= -\eta \frac{\partial E}{\partial w_{ji}} \\ &= -\eta \frac{\partial E}{\partial s_j}\frac{\partial s_j}{\partial w_{ji}} \\ &= \eta \delta_j x_i\end{aligned}$$

For each neuron :

$$\begin{aligned}s &= \sum_{i=0}^{n} w_i x_i \\ y &= \frac{1}{1+e^{-s}}\end{aligned}$$

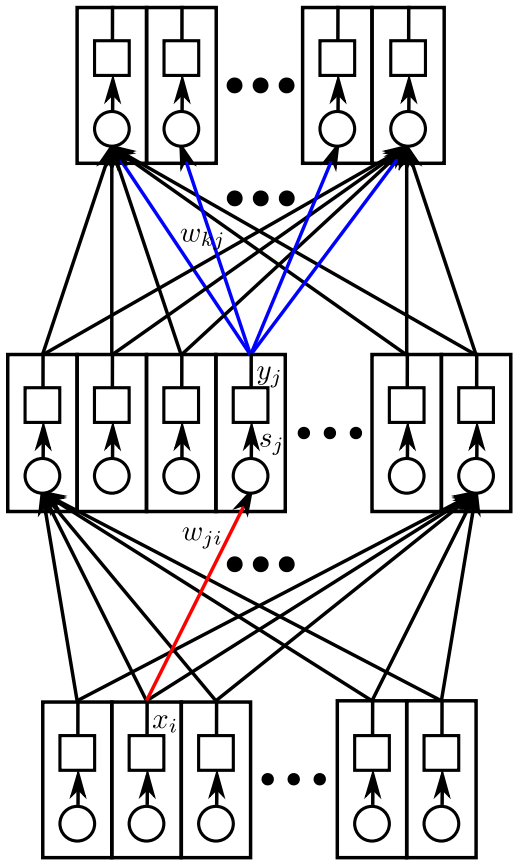


Learning rule : We want to minimize the mean squared error $E = \frac{1}{2}\sum_x (t-y)^2$ by stochastic gradient descent.

$$
\begin{aligned}
\Delta w_{ji} &= -\eta \frac{\partial E}{\partial w_{ji}} \\
&= -\eta \frac{\partial E}{\partial s_j}\frac{\partial s_j}{\partial w_{ji}} \\
&= \eta \delta_j x_i \\
\delta_j &= -\frac{\partial E}{\partial s_j} \\
&= -\sum_k \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial y_j}\frac{\partial y_j}{\partial s_j} \\
&= \sum_k \delta_k w_{kj} y_j (1-y_j) \\
&= y_j(1-y_j)\sum_k \delta_k w_{jk}
\end{aligned}
$$

For each neuron :

$$
\begin{aligned}
s &= \sum_{i=0}^{n} w_i x_i \\
y &= \frac{1}{1+e^{-s}}
\end{aligned}
$$



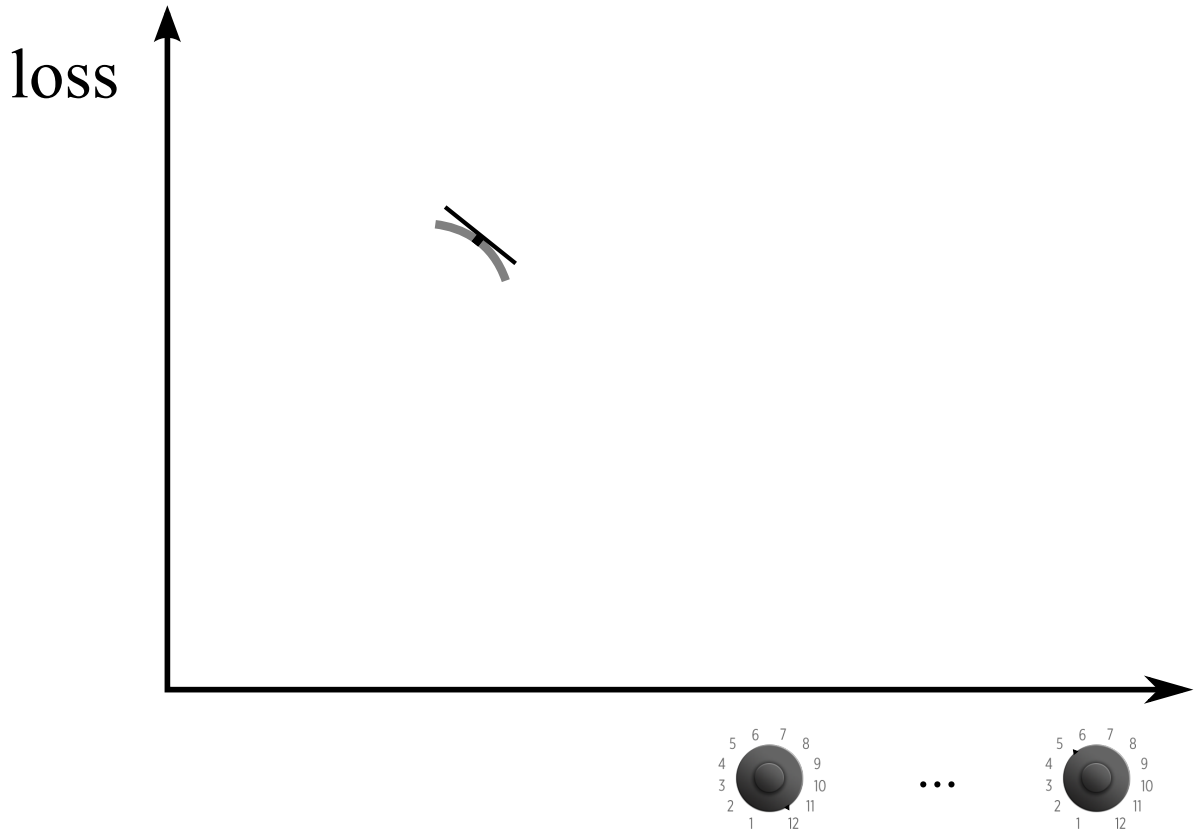
MultiLayer Perceptron - Werbos & Rumelhard (1984-1986)

Learning rule : We want to minimize the mean squared error $E = \frac{1}{2}\sum_x (t-y)^2$ by stochastic gradient descent.

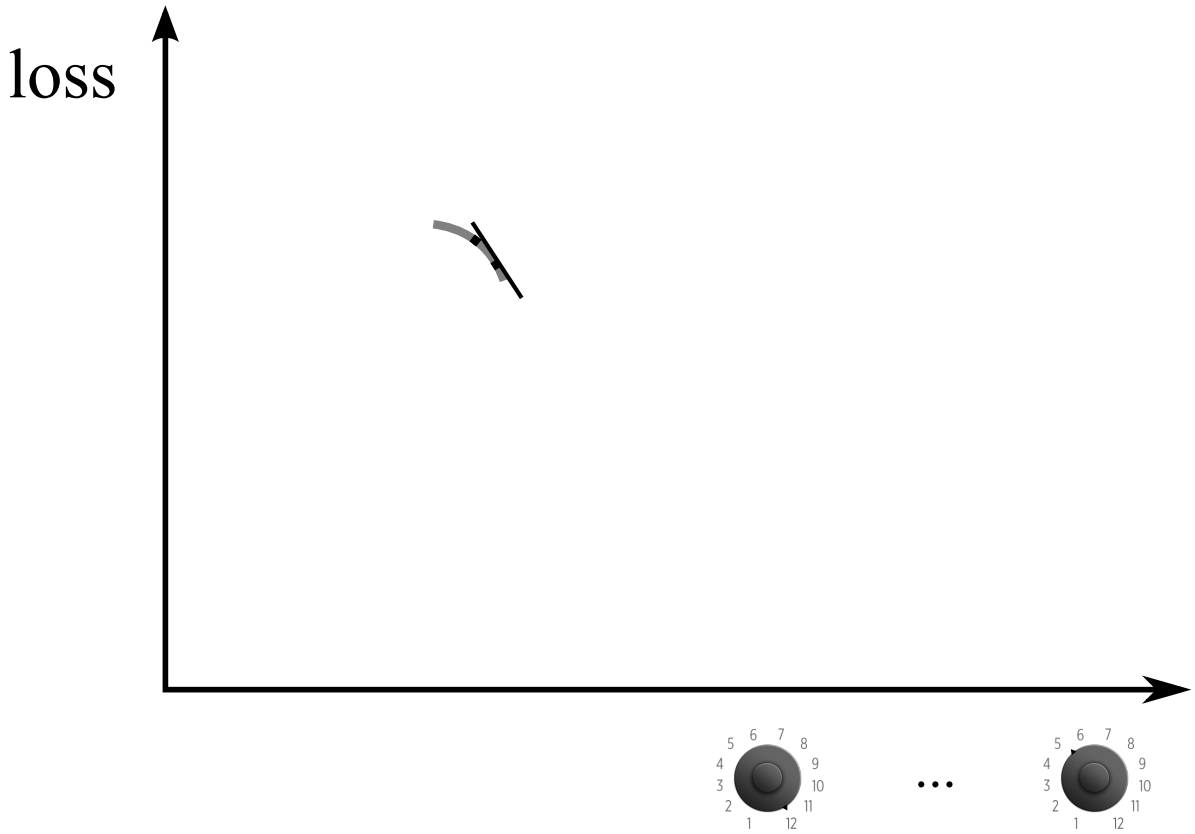
For each input :

1. Compute the network output y by propagating activity (layer by layer)
2. Compute the error in the output layer : $\delta_k = y_k(1-y_k)(t_k - y_k)$
3. Retropropagate the error through each layer j of the network $\delta_j = y_j(1-y_j)\sum_k \delta_k w_{jk}$
4. Modify each weight $\Delta w_{ij} = \sum_{x_i} \eta \delta_j x_i$

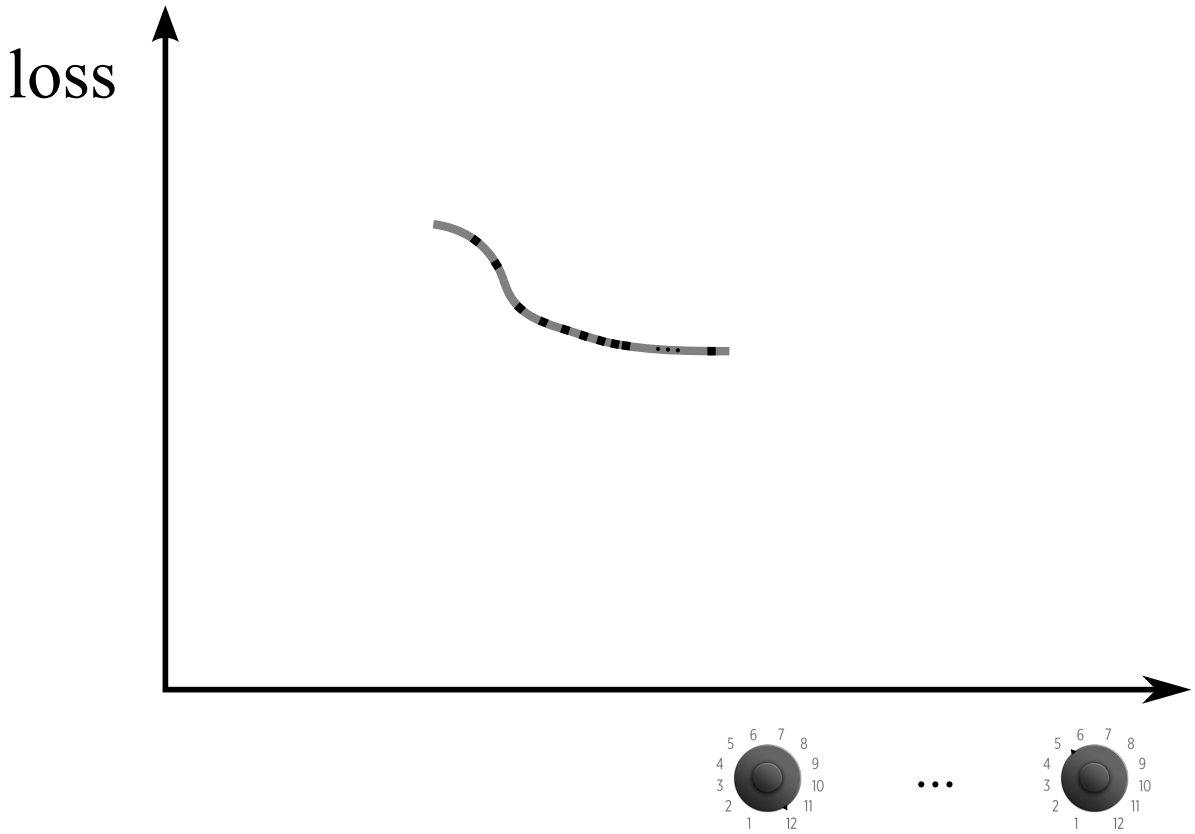
MultiLayer Perceptron - Werbos & Rumelhard (1984-1986)



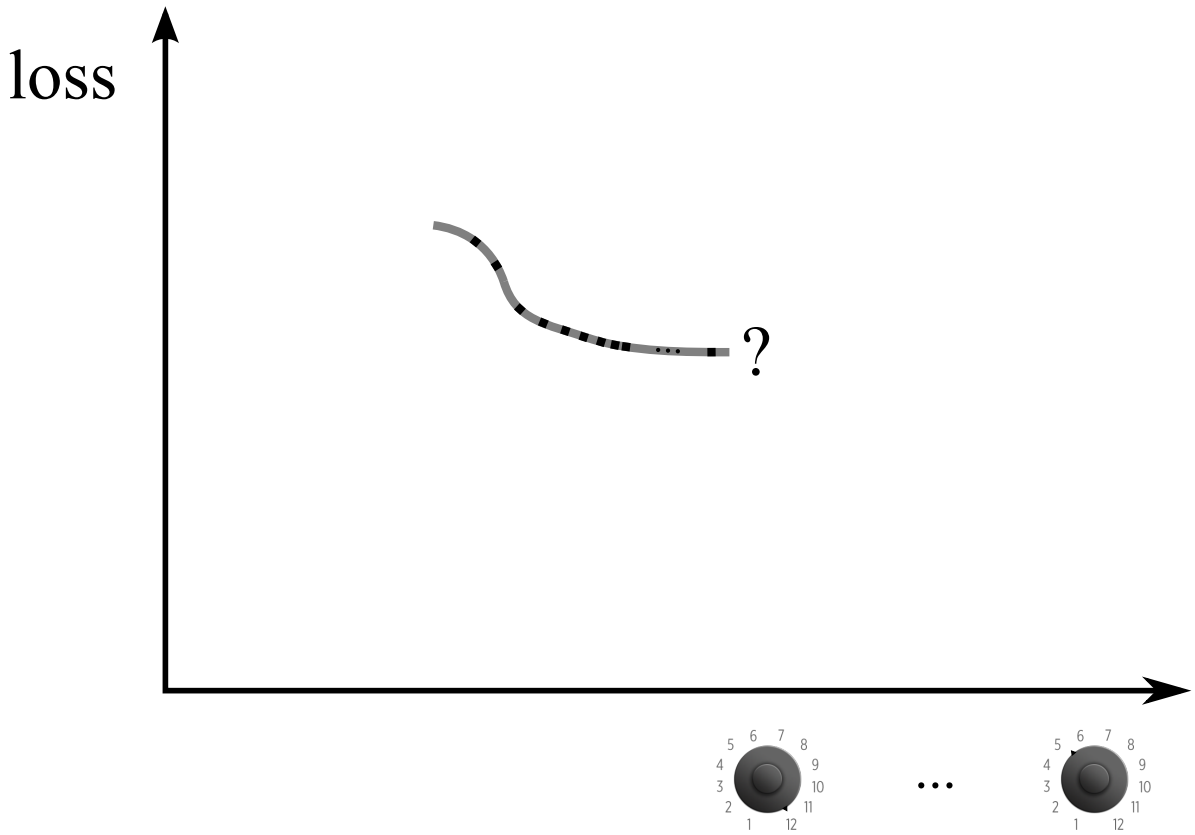
MultiLayer Perceptron - Werbos & Rumelhard (1984-1986)



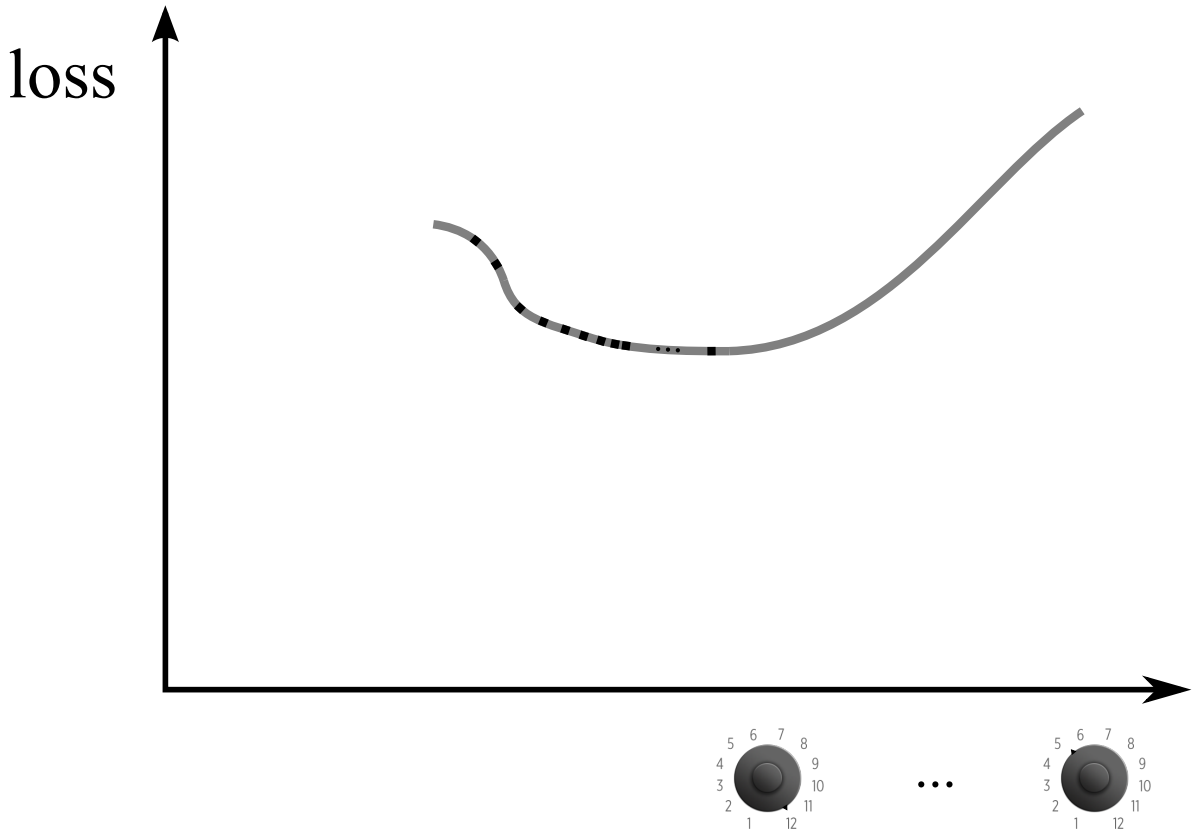
MultiLayer Perceptron - Werbos & Rumelhard (1984-1986)



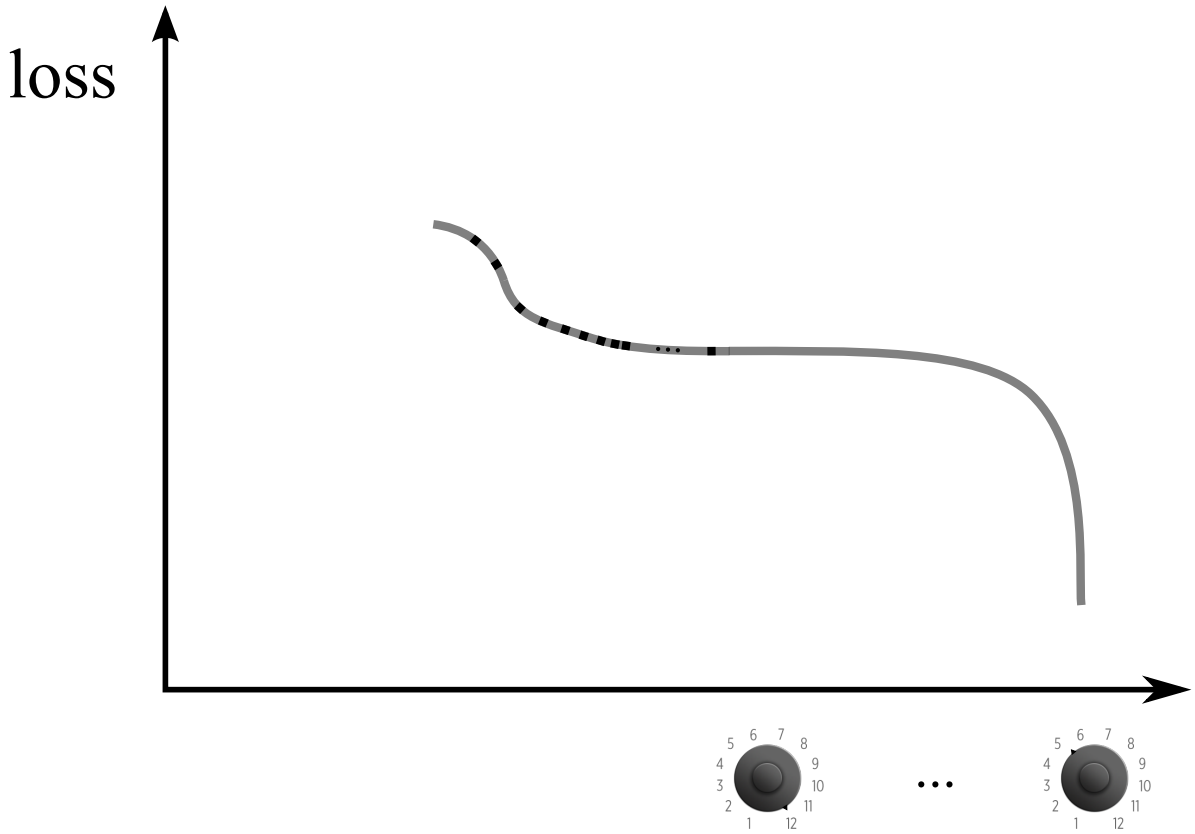
MultiLayer Perceptron - Werbos & Rumelhard (1984-1986)



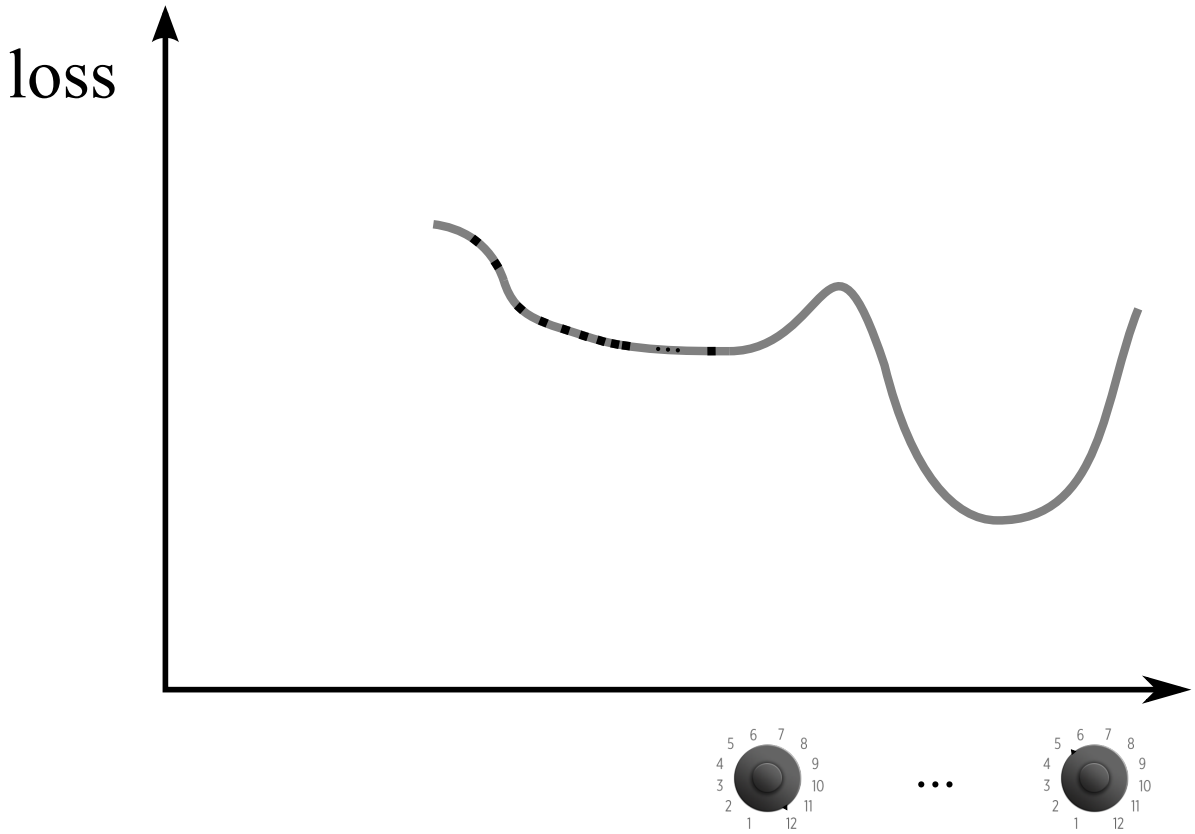
MultiLayer Perceptron - Werbos & Rumelhard (1984-1986)



MultiLayer Perceptron - Werbos & Rumelhard (1984-1986)



MultiLayer Perceptron - Werbos & Rumelhard (1984-1986)



MLP summary

Properties

- Universal approximator (with sigmoid, gausssian or many others activation function)
- Potentially uneffective convergence (non convex optimization, high dimensional space, vanishing or unbounded gradient, ...)

MLP summary

Properties

- Universal approximator (with sigmoid, gausssian or many others activation function)
- Potentially uneffective convergence (non convex optimization, high dimensional space, vanishing or unbounded gradient, ...)

Demo

`http://playground.tensorflow.org/`

Convolutional Neural Network - LeCun (1989)

Question

How to learn from images as inputs ?

Convolutional Neural Network - LeCun (1989)

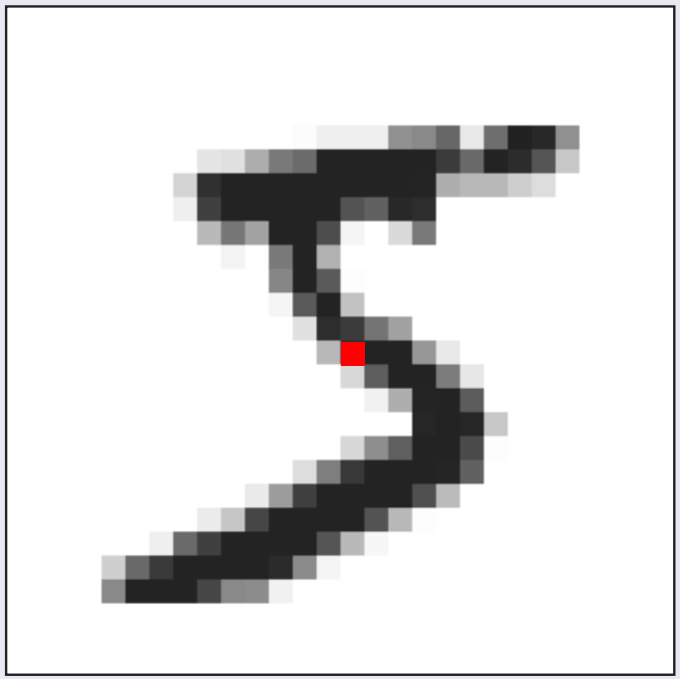
Question

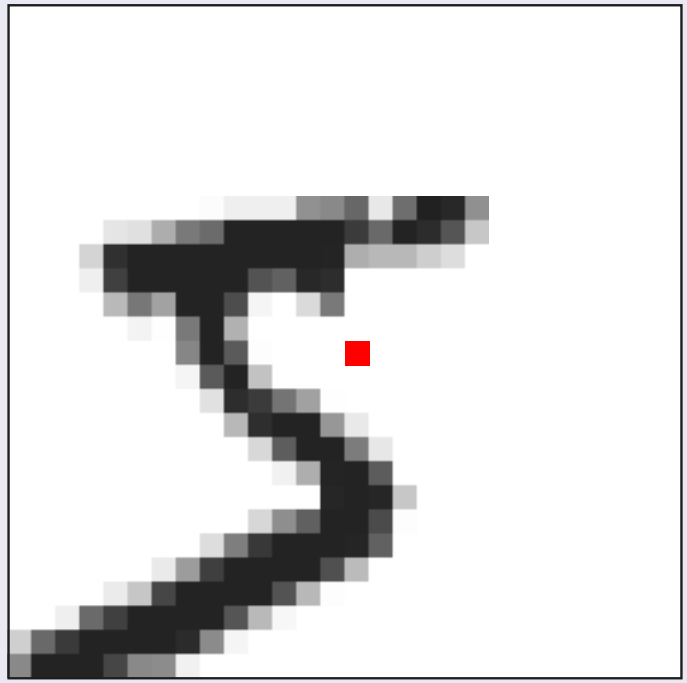
How to learn from images as inputs ?

(Bad) solution

To use a MLP but there is

- a huge number of weights to learn (an image has at least 1000 dimensions)
- the problem of translation





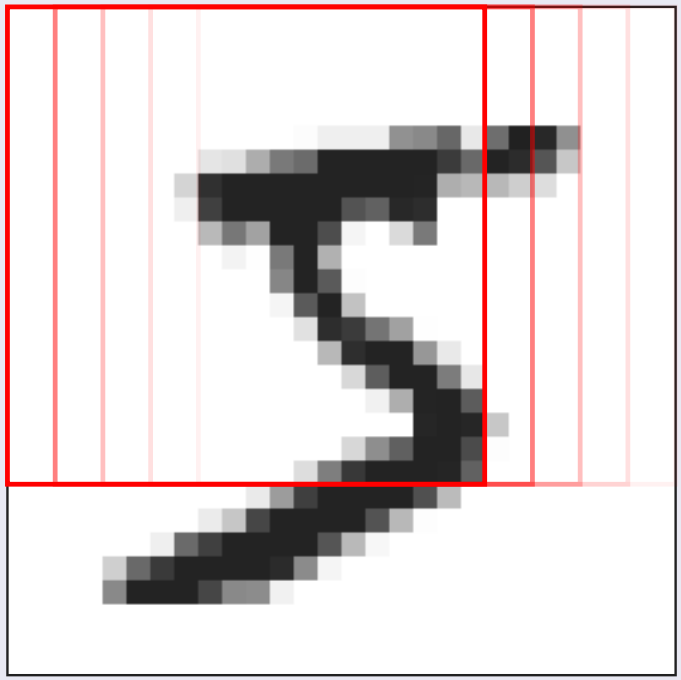
Convolutional Neural Network - LeCun (1989)

Question

How to learn from images as inputs ?

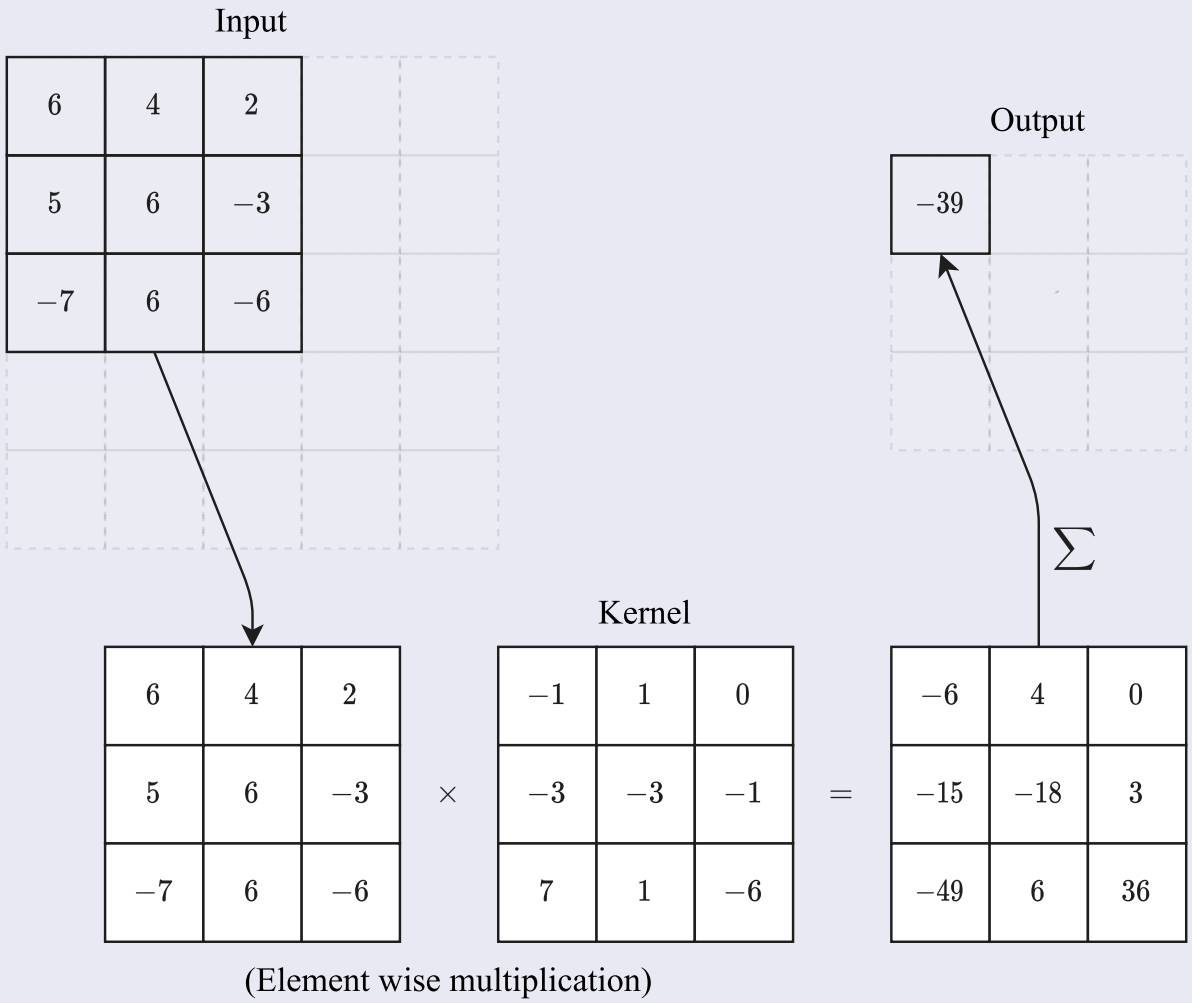
Good solution

To adapt the neurons and network to perform convolution.



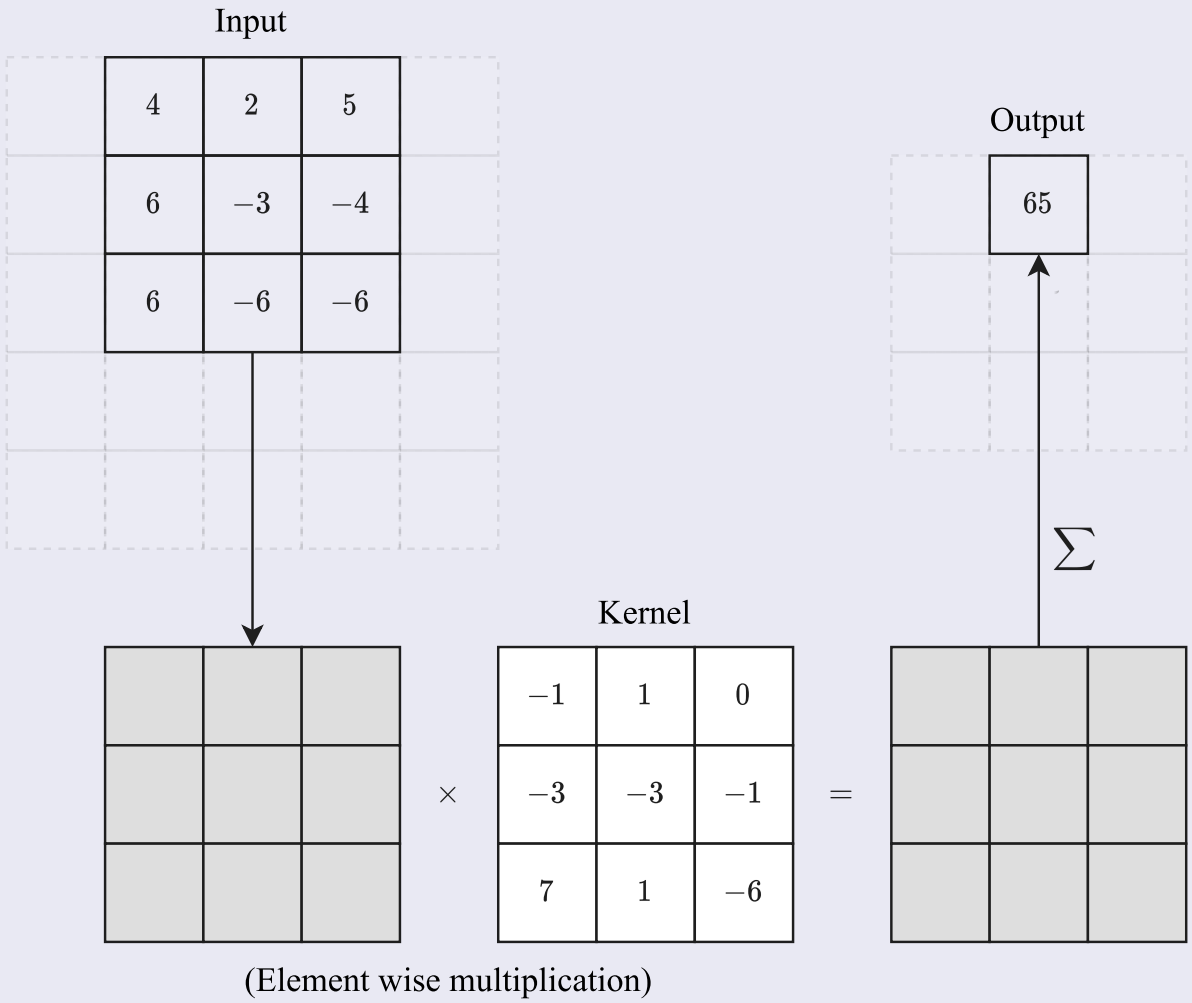
Convolutional Neural Network - LeCun (1989)

Convolution operation



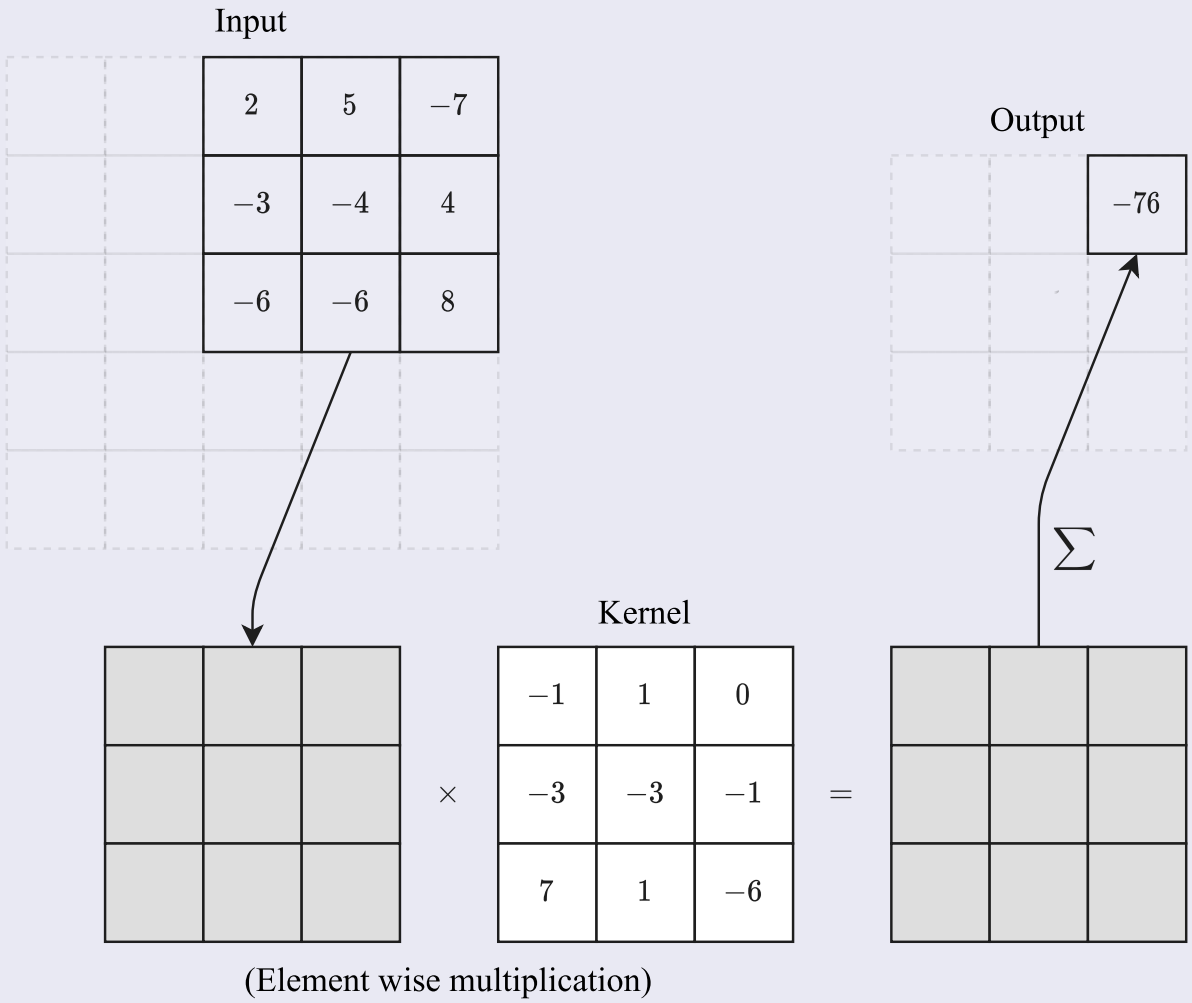
Convolutional Neural Network - LeCun (1989)

Convolution operation



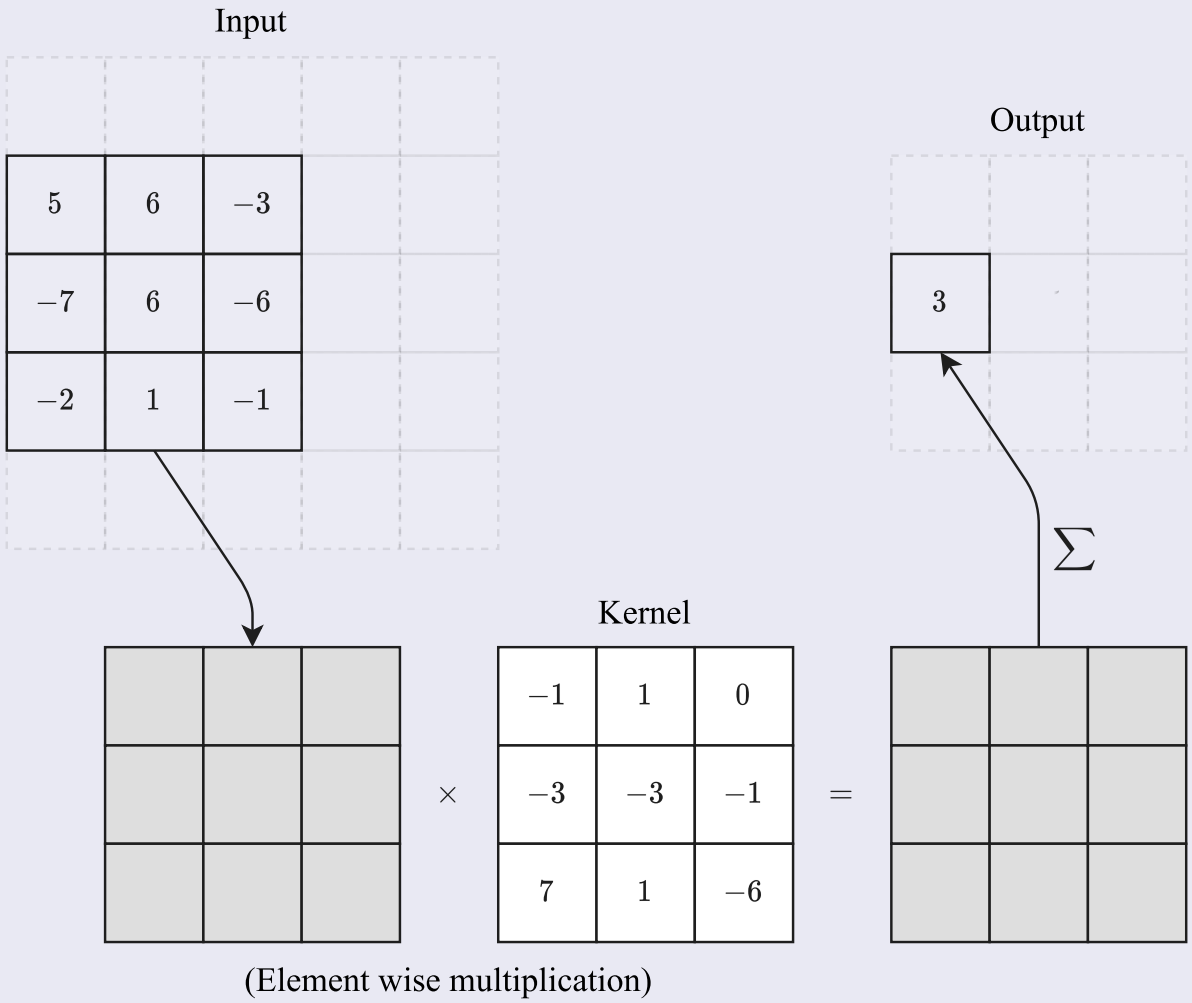
Convolutional Neural Network - LeCun (1989)

Convolution operation

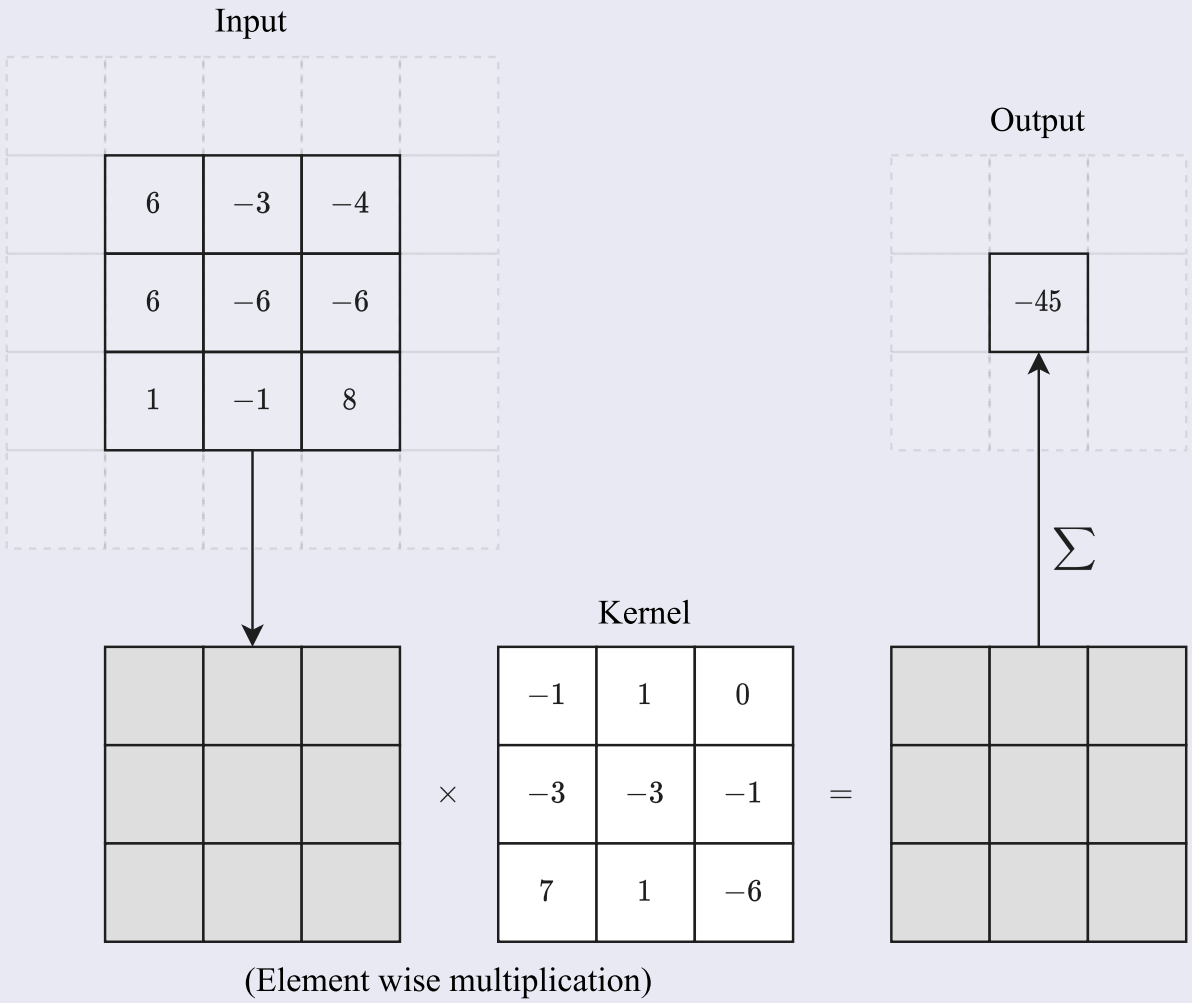


Convolutional Neural Network - LeCun (1989)

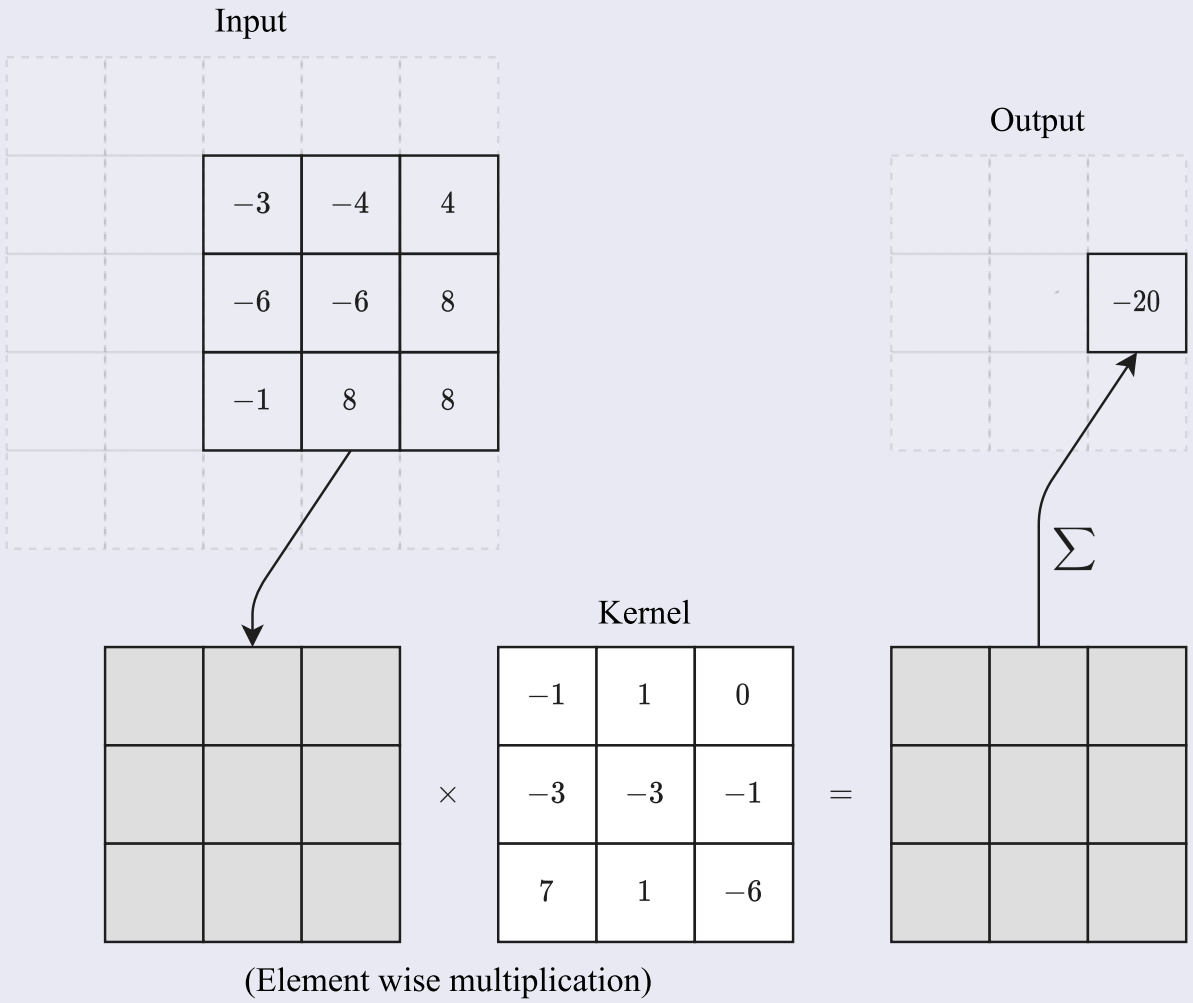
Convolution operation



Convolution operation

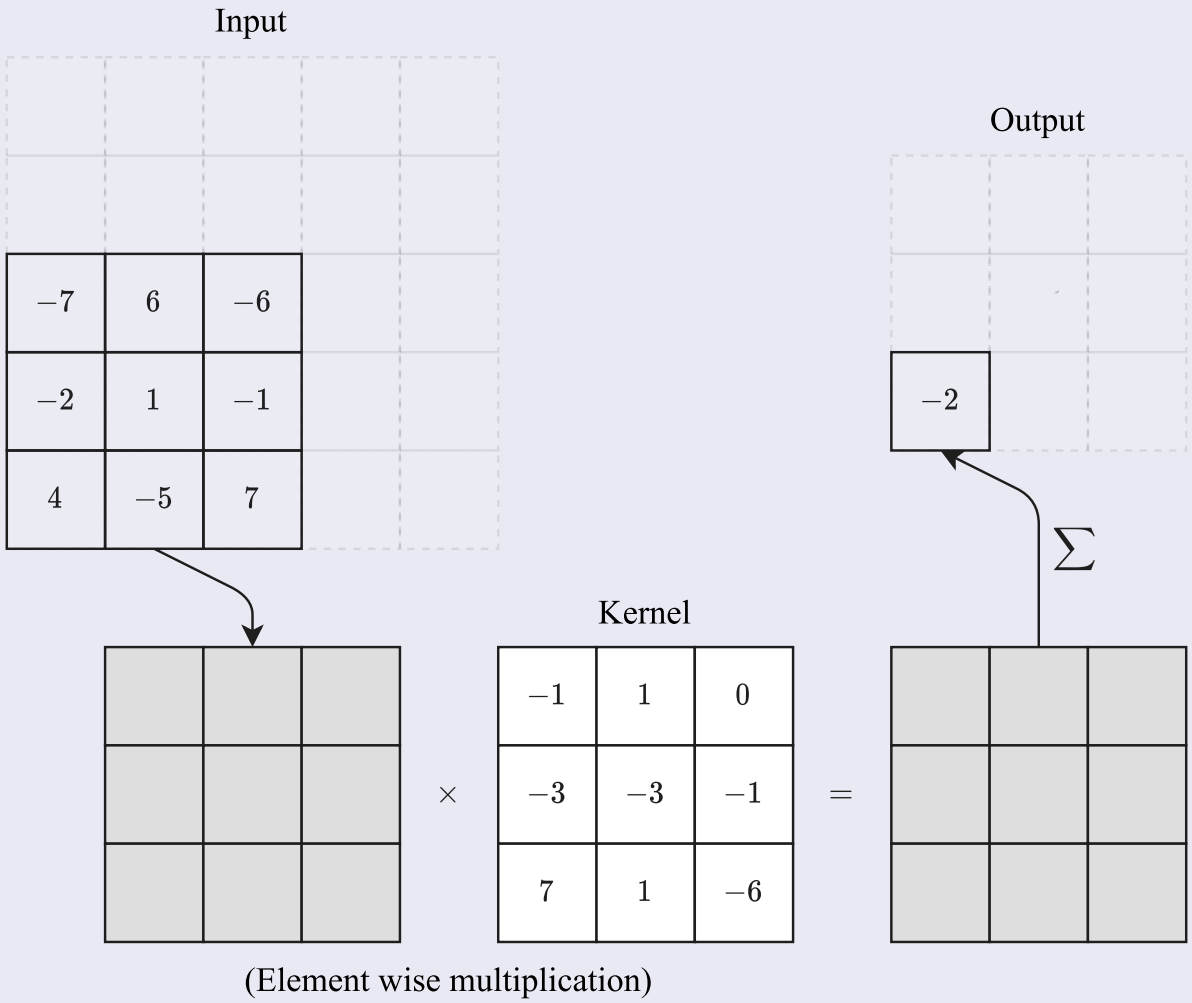


Convolution operation

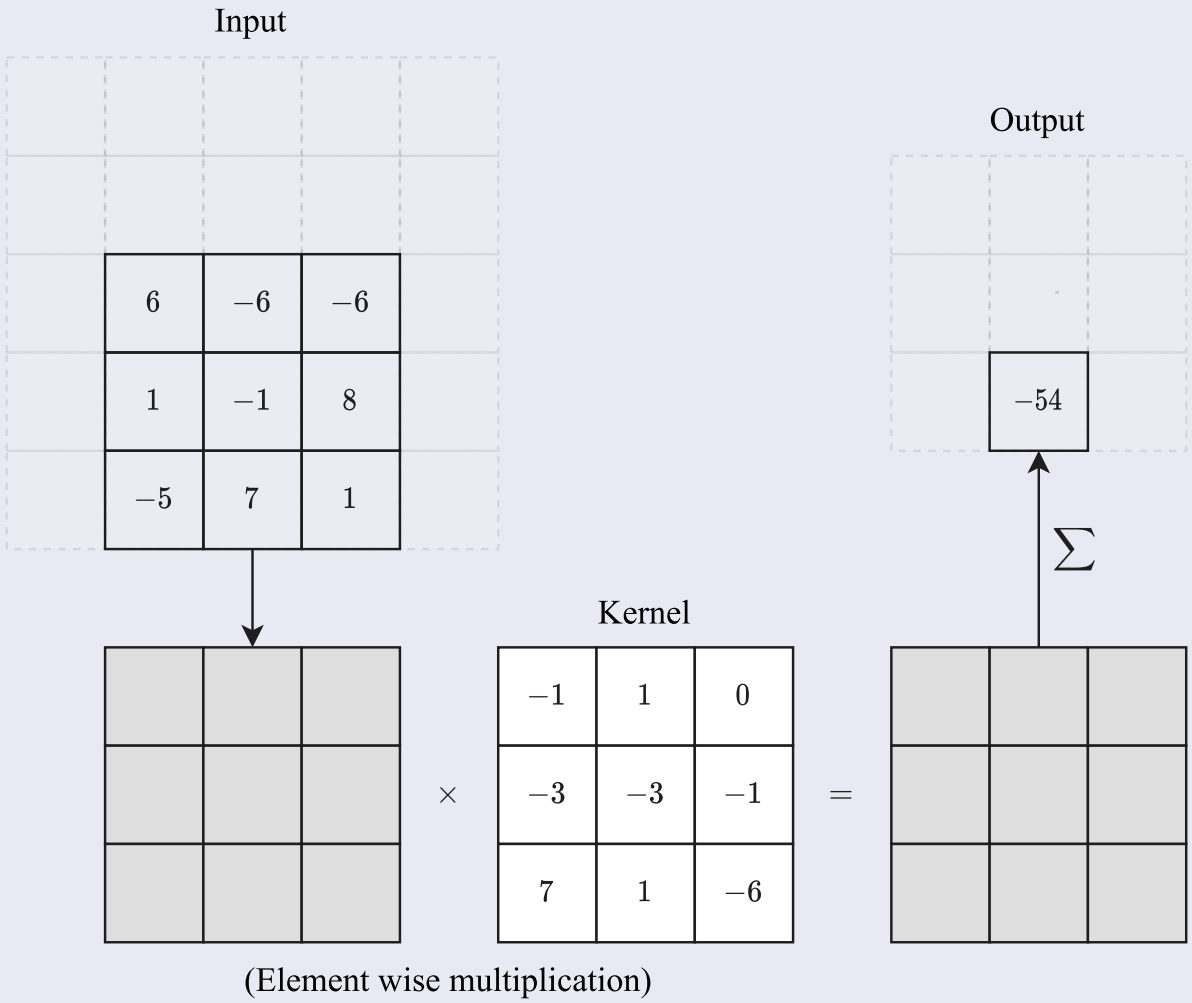


Convolutional Neural Network - LeCun (1989)

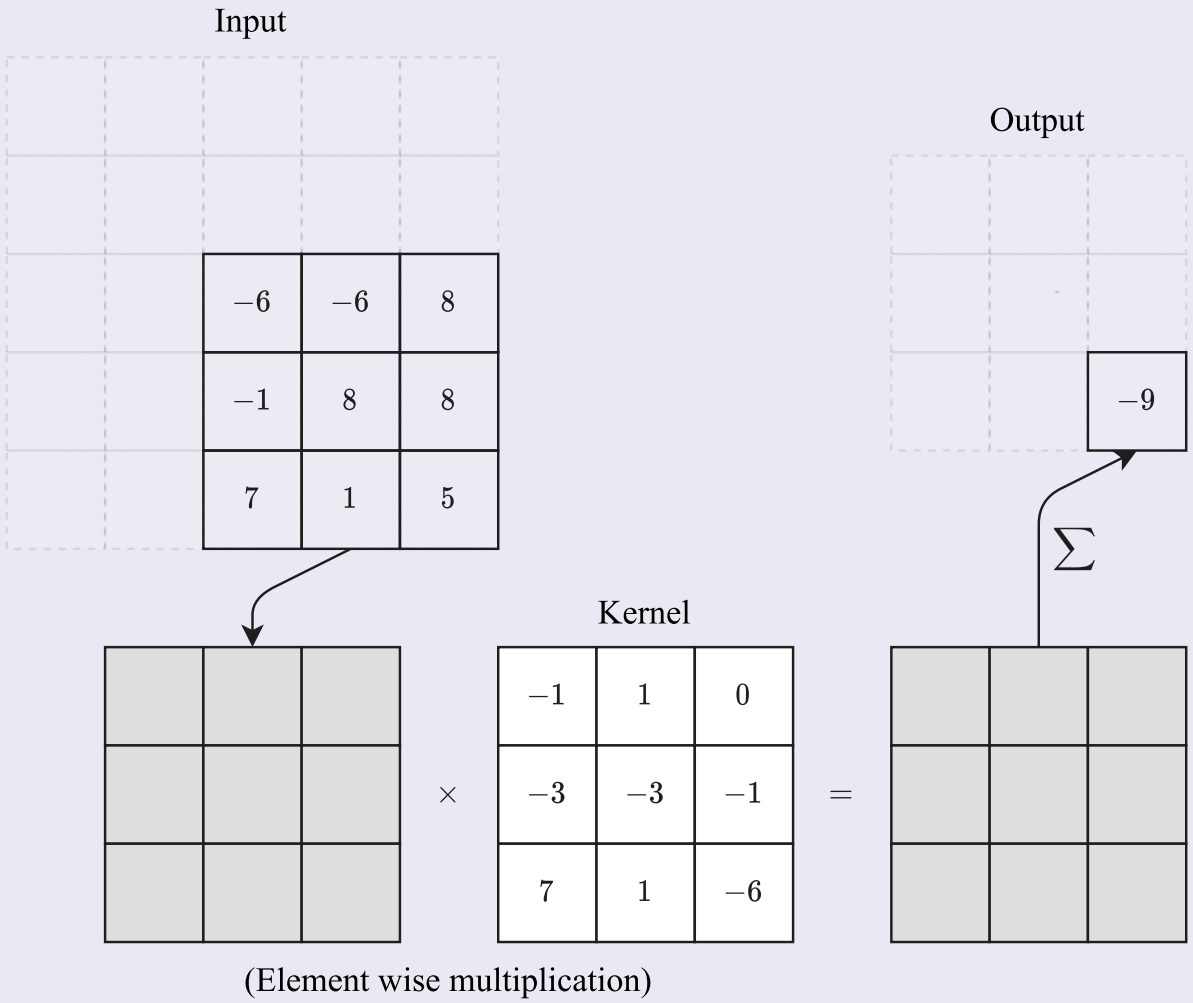
Convolution operation



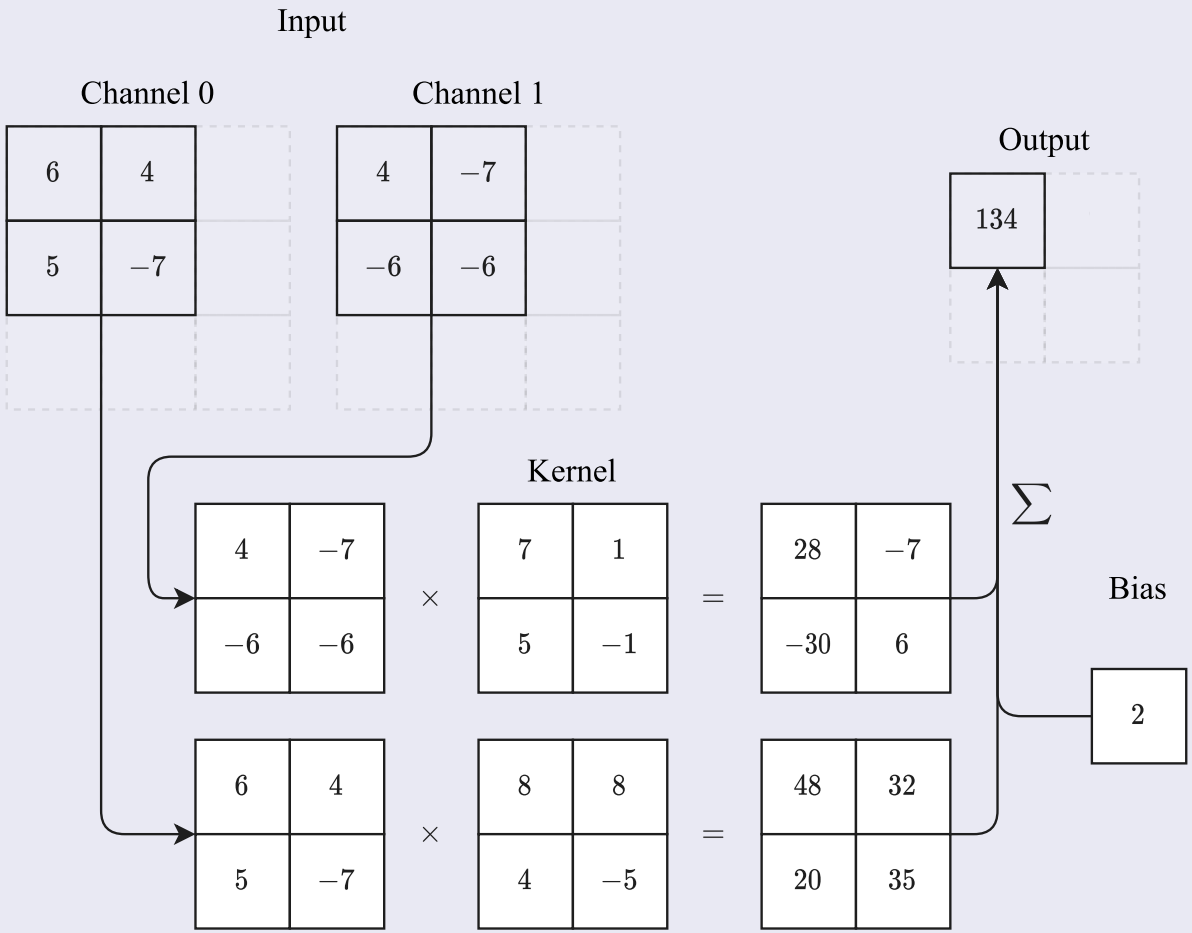
Convolution operation



Convolution operation

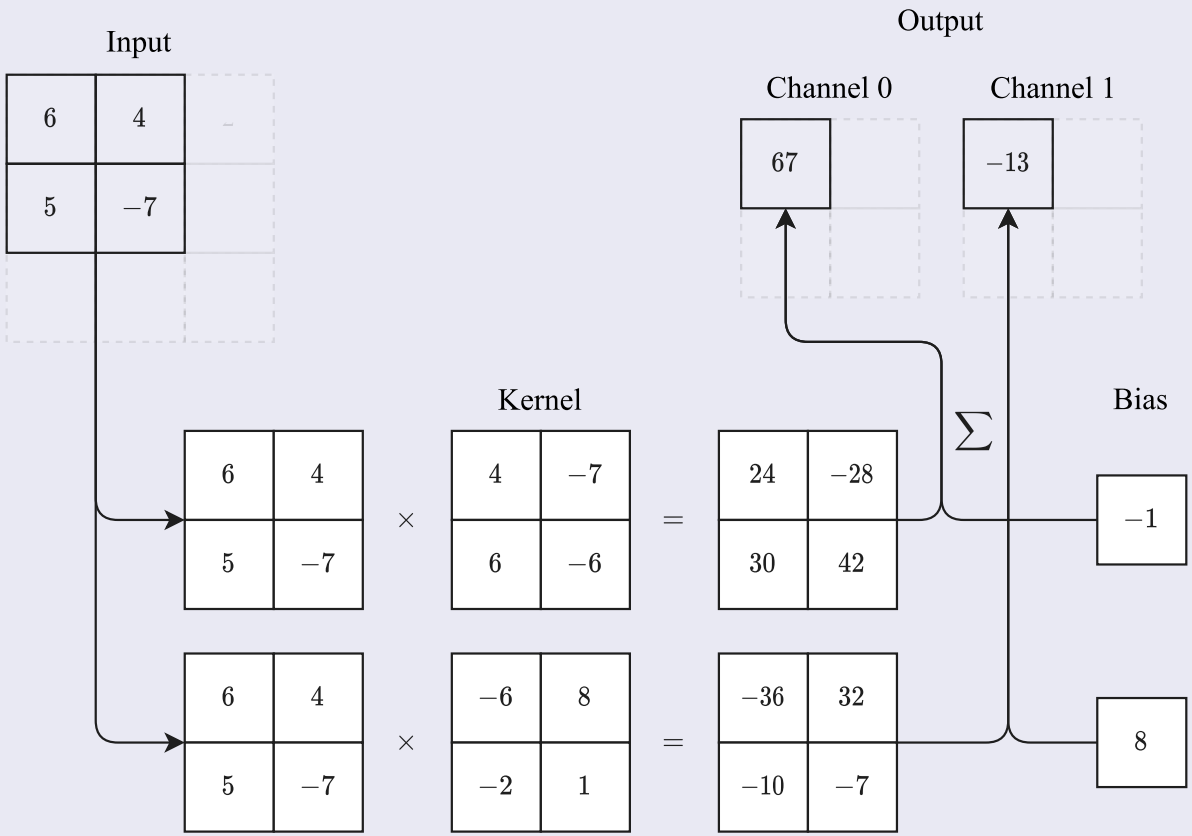


Convolutional neuron

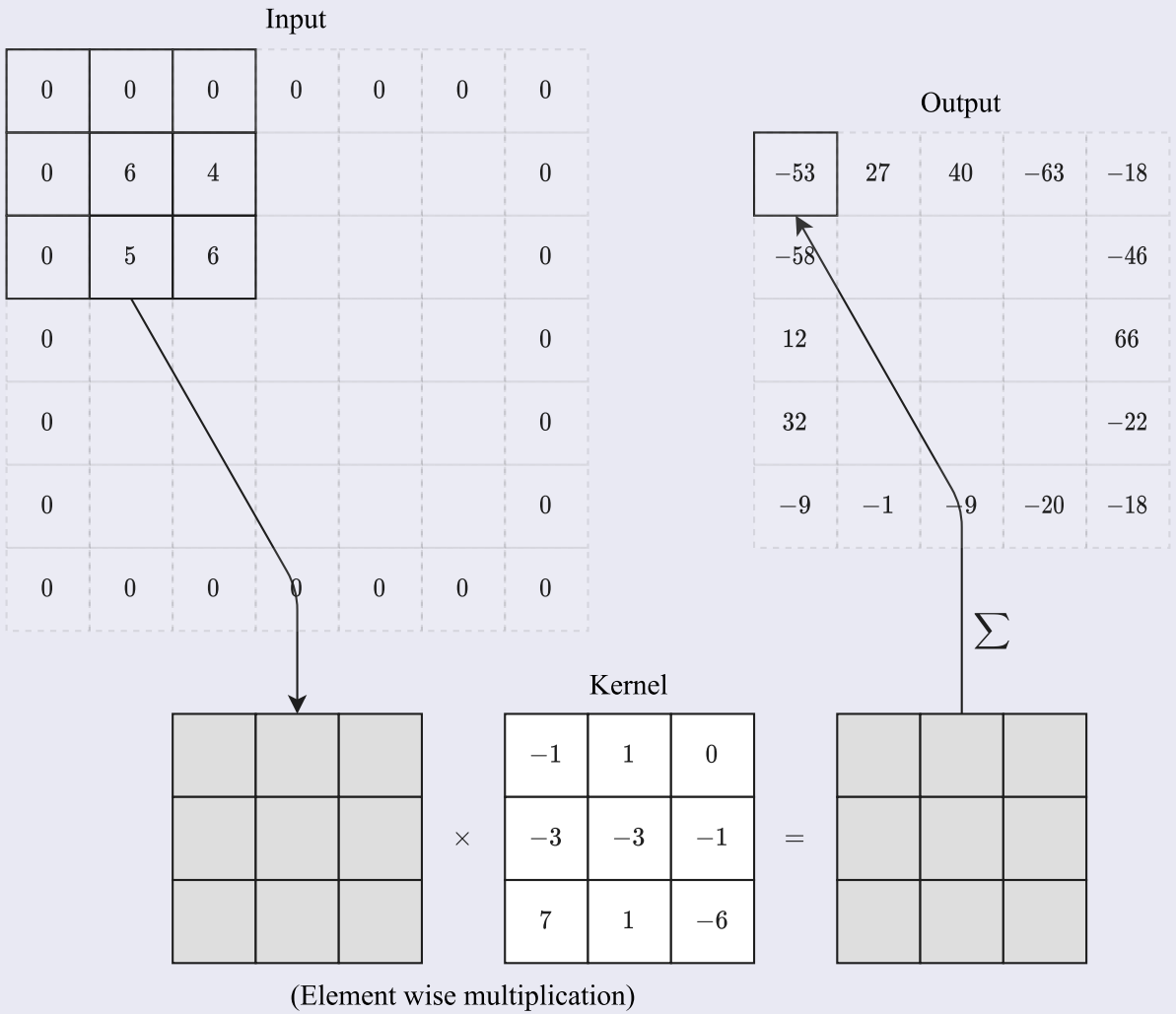


Convolutional Neural Network - LeCun (1989)

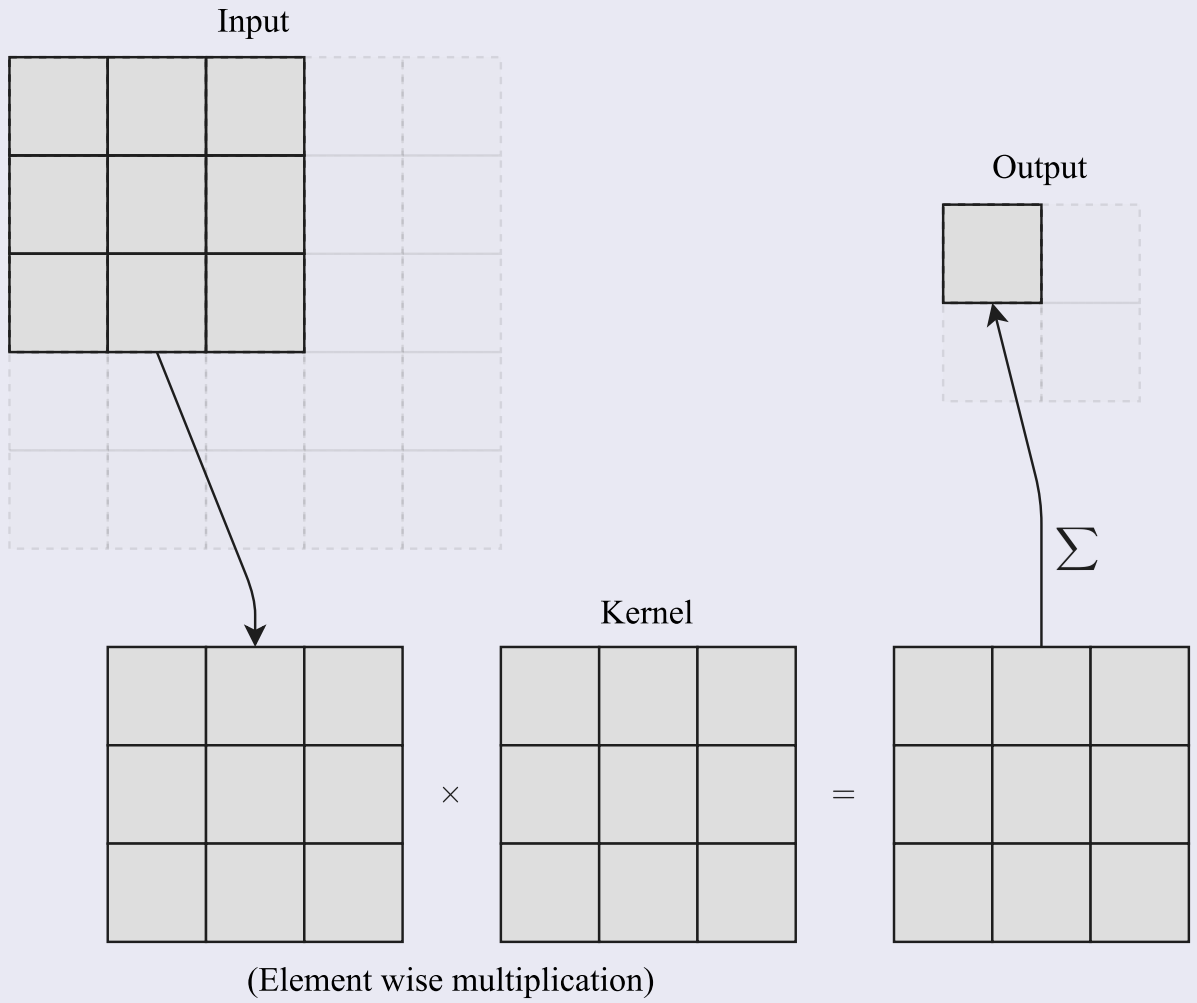
Convolutional layer



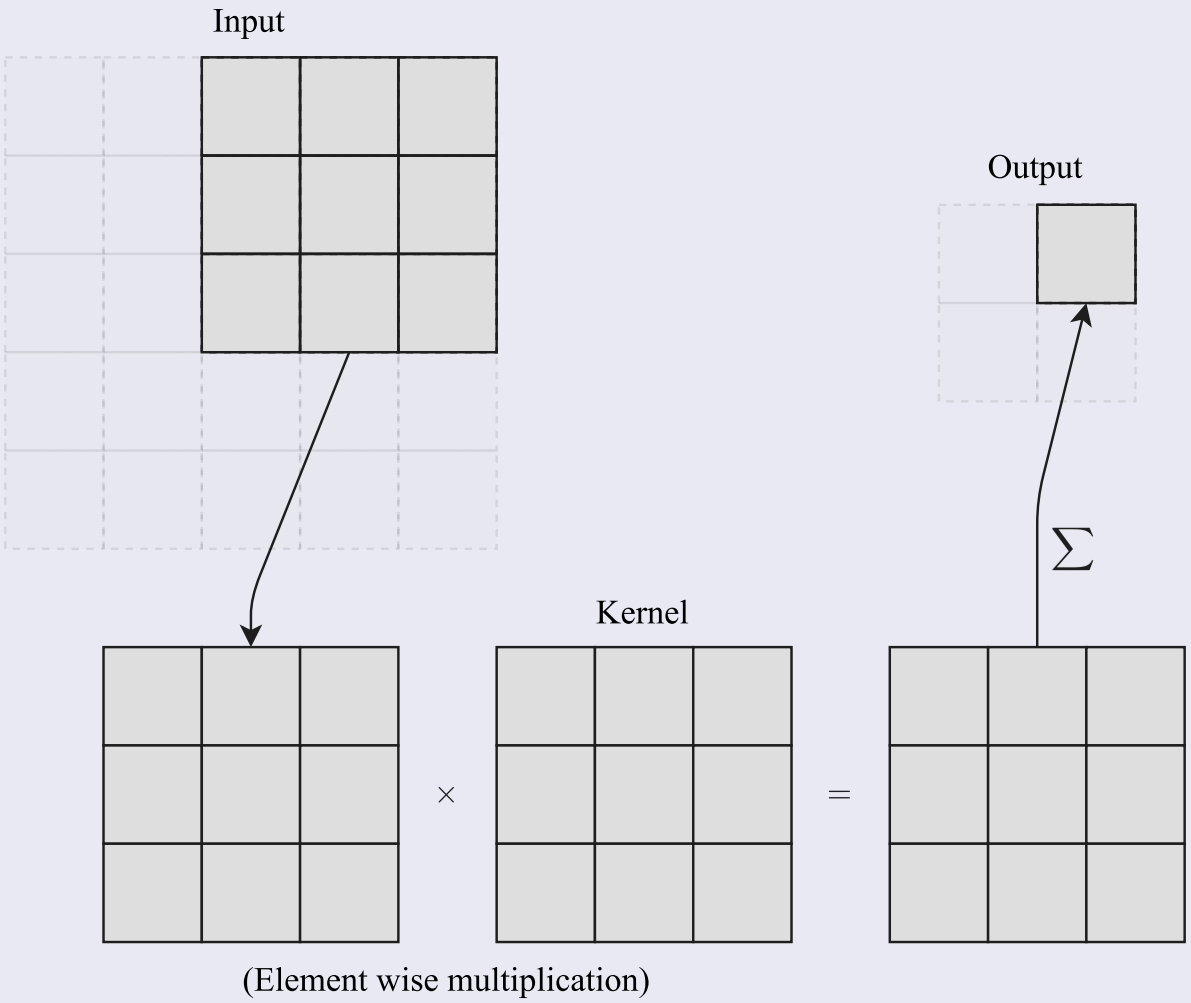
Convolutional layer - Padding



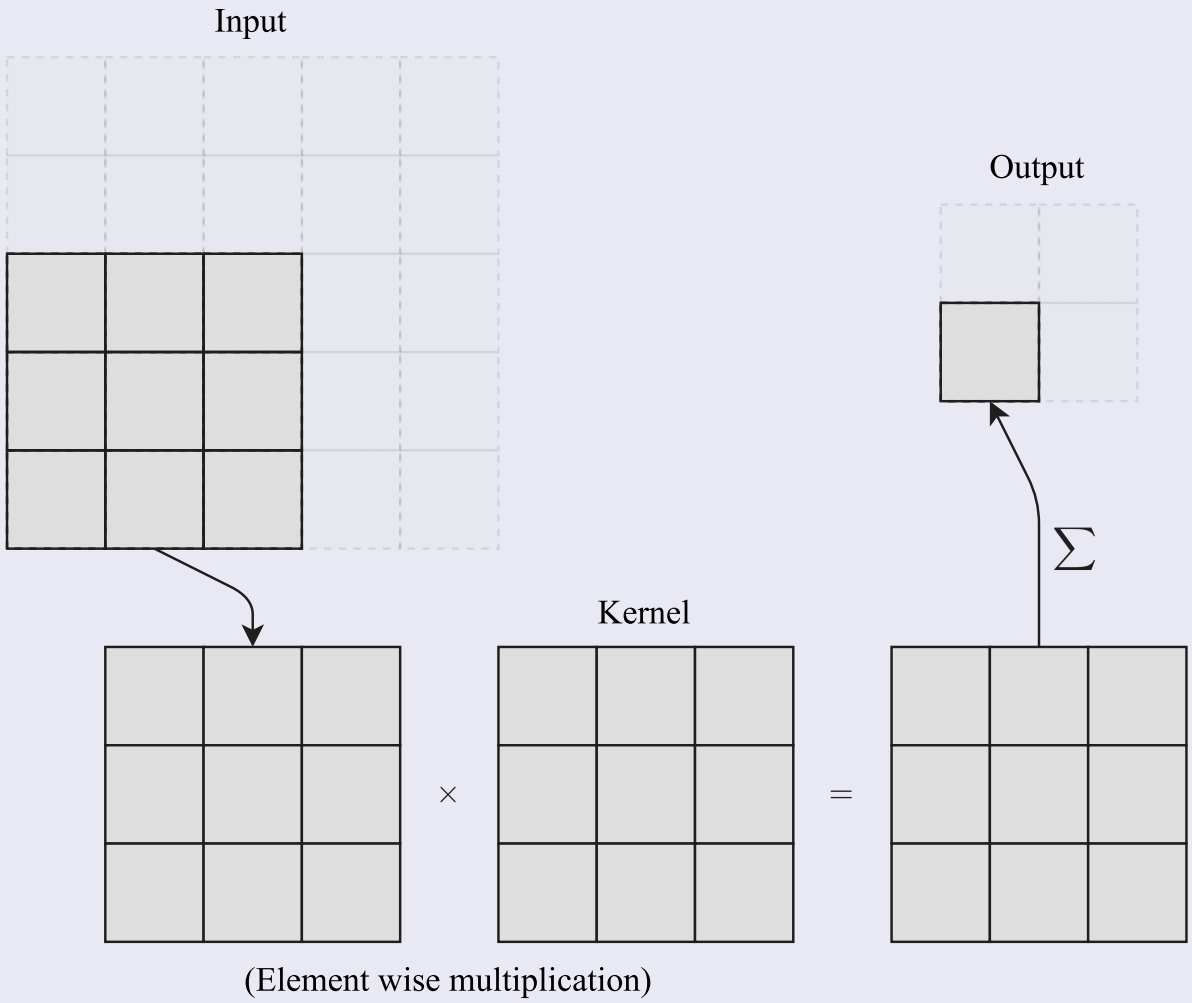
Convolutional layer - Stride



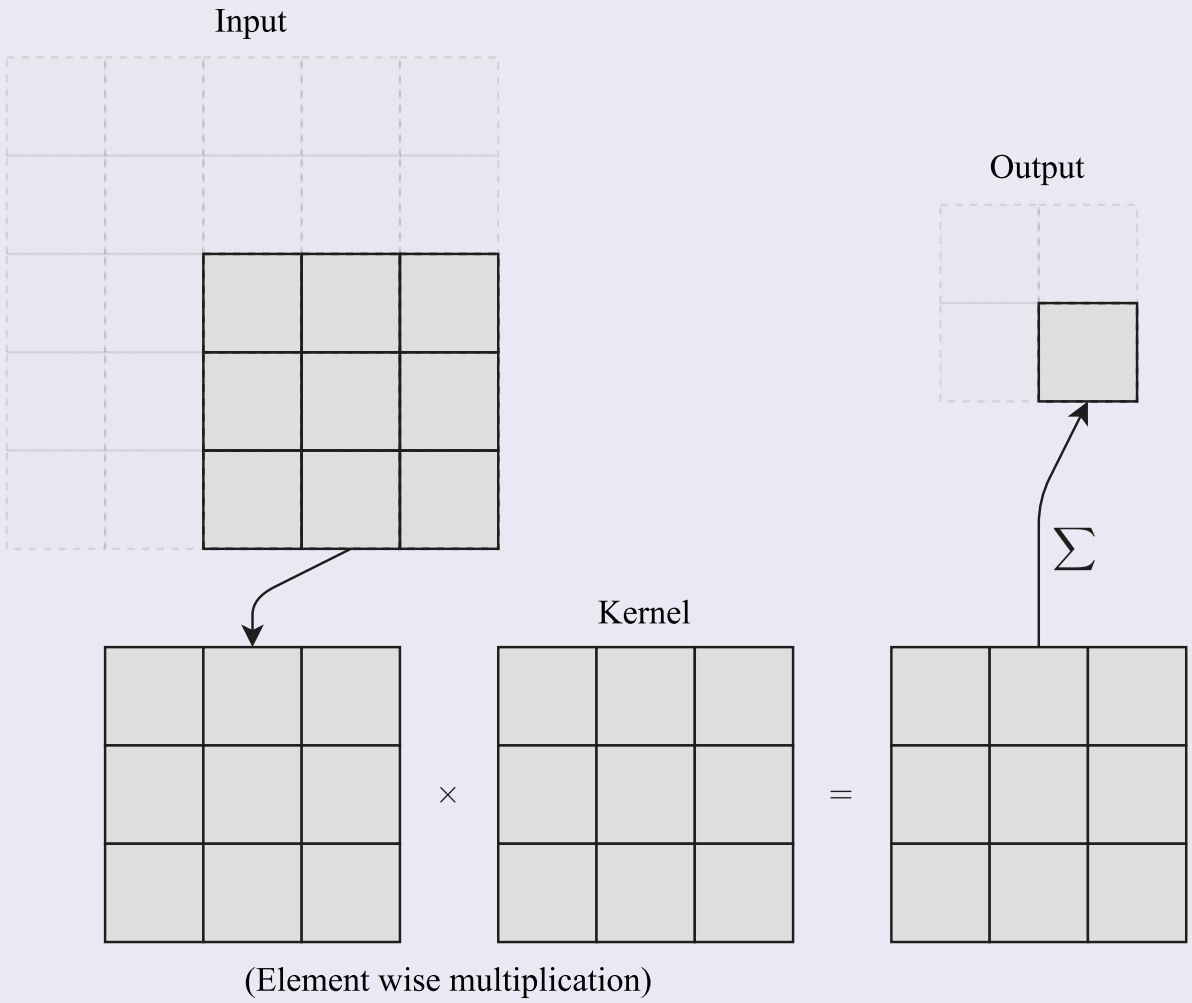
Convolutional layer - Stride



Convolutional layer - Stride



Convolutional layer - Stride

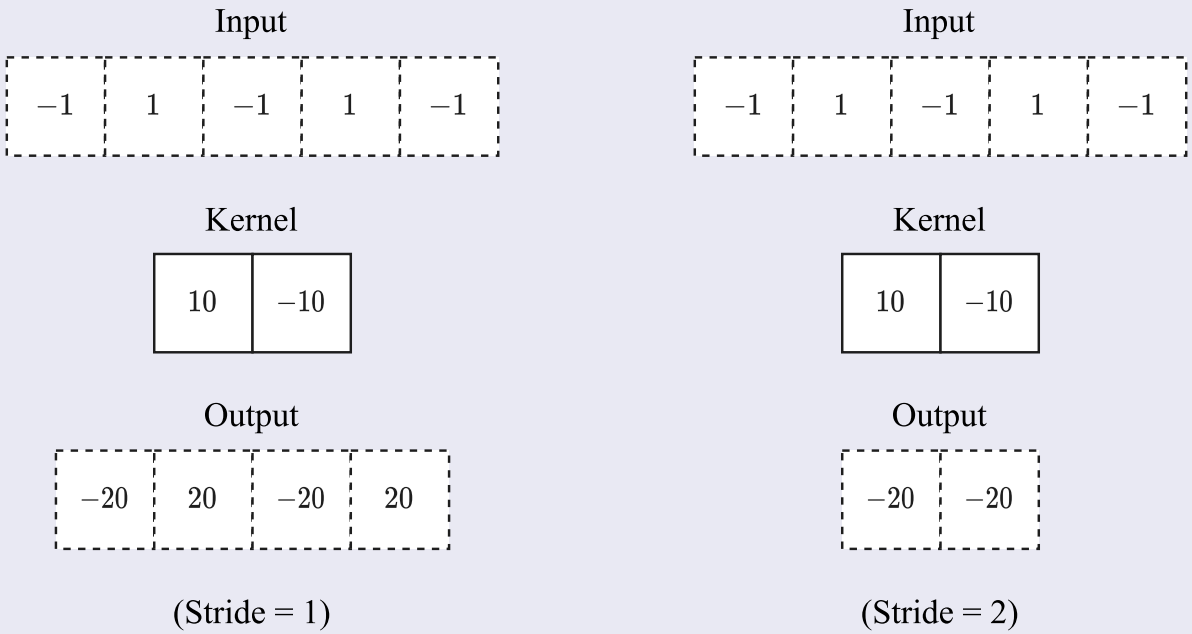


Pooling layer

No need of high resolution / dimensionnality to recognize the content ...

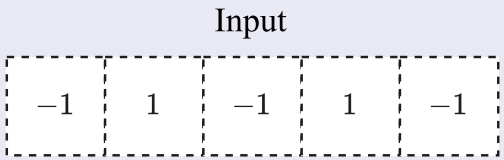
Pooling layer

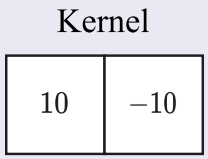
No need of high resolution / dimensionnality to recognize the content ...
... but increasing the stride can be risky ...

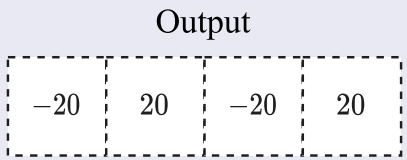


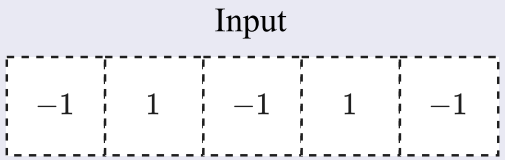
Pooling layer

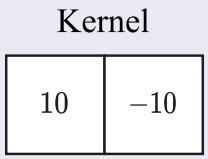
No need of high resolution / dimensionnality to recognize the content ...
... but increasing the stride can be risky ...
... so performing pooling with a max (or an average) operation

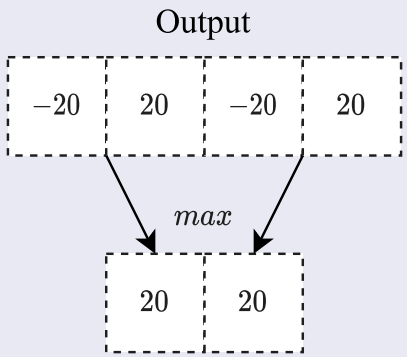




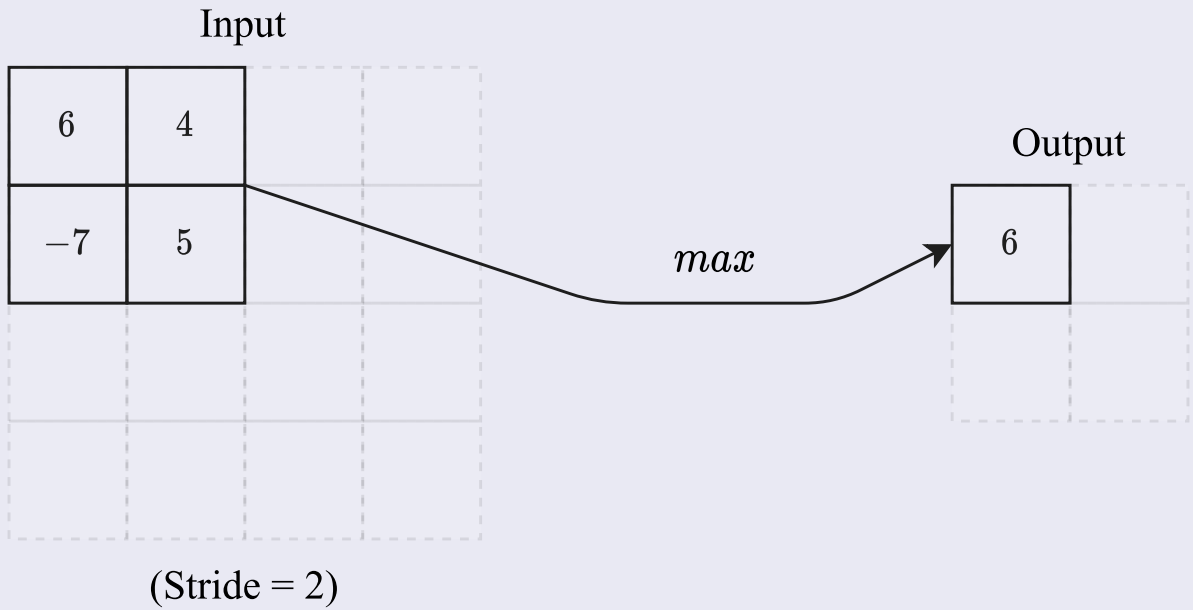




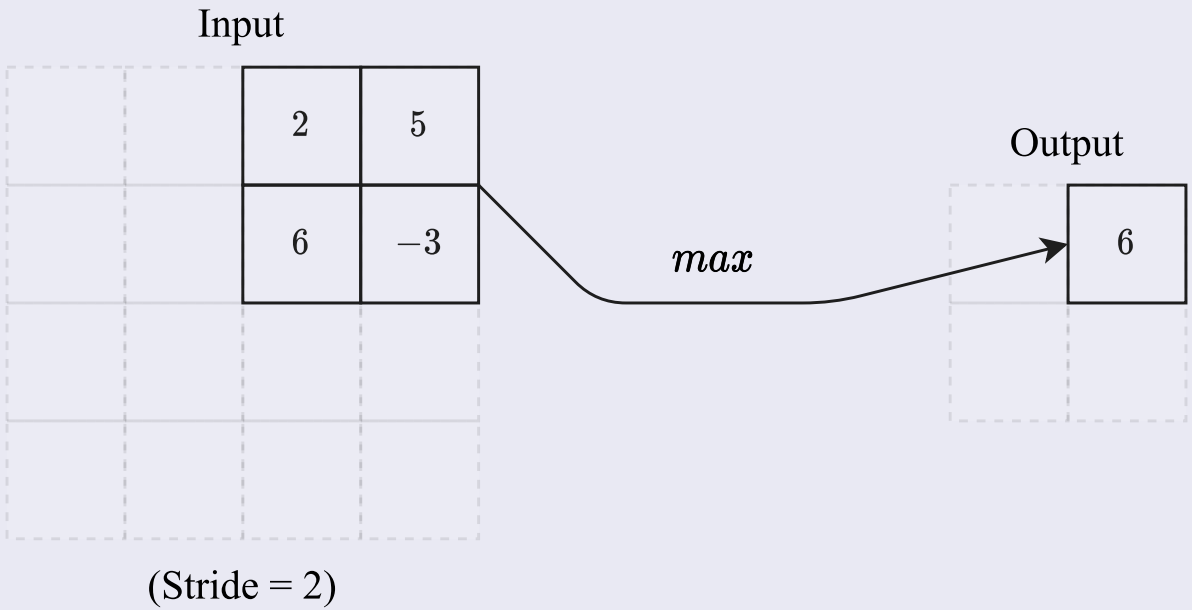




Pooling layer

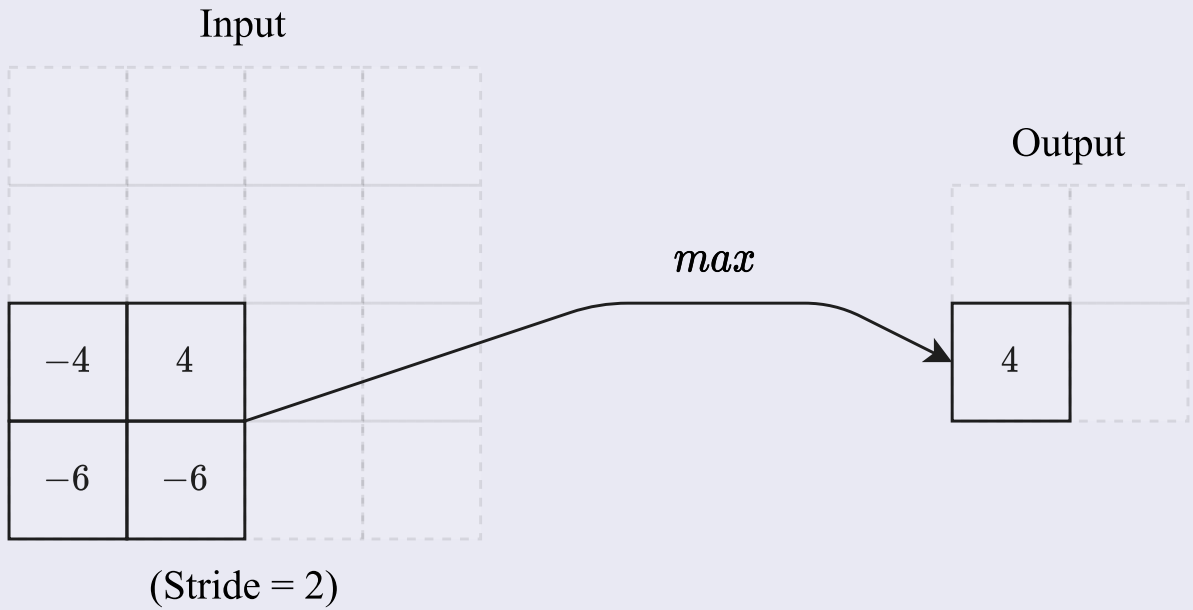


Pooling layer



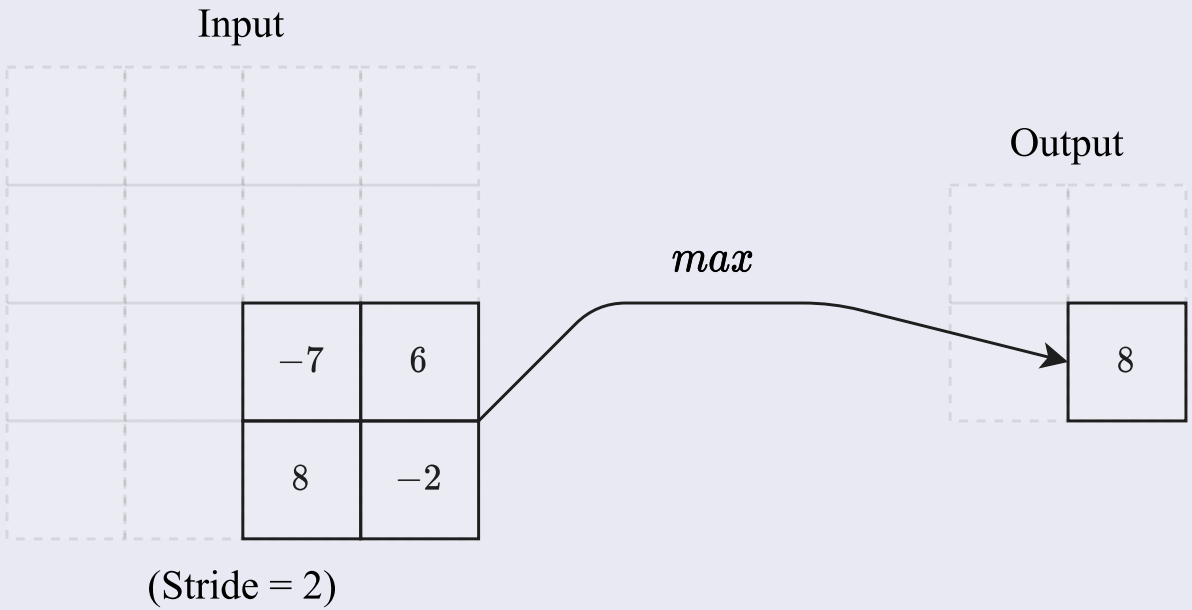
Convolutional Neural Network - LeCun (1989)

Pooling layer



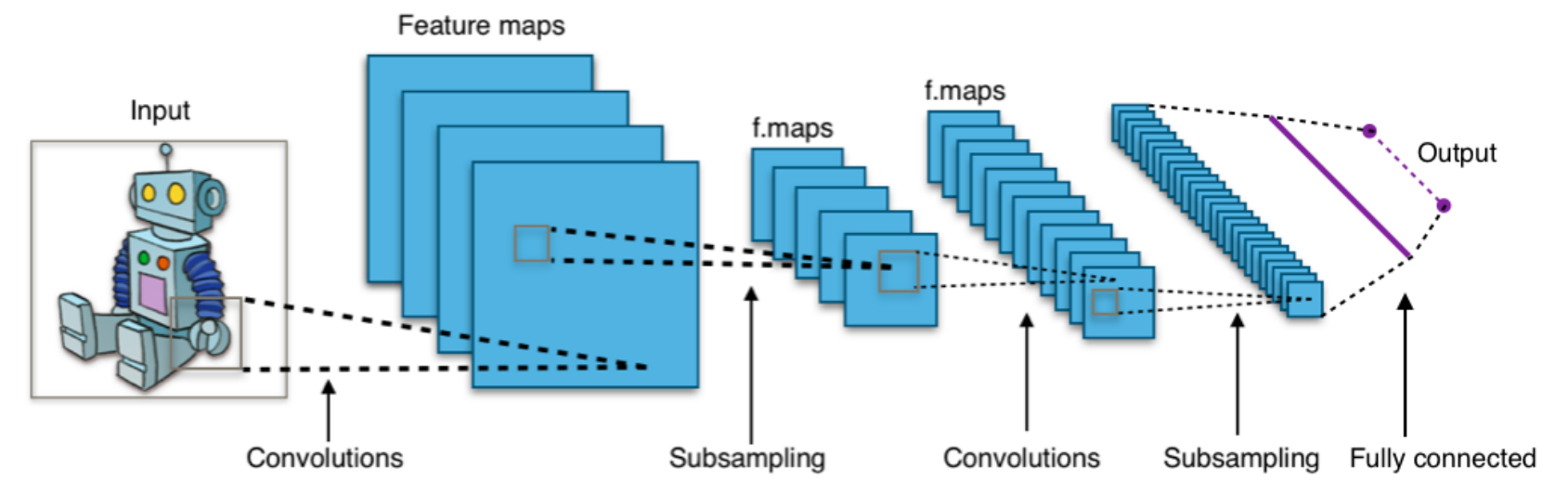
Convolutional Neural Network - LeCun (1989)

Pooling layer



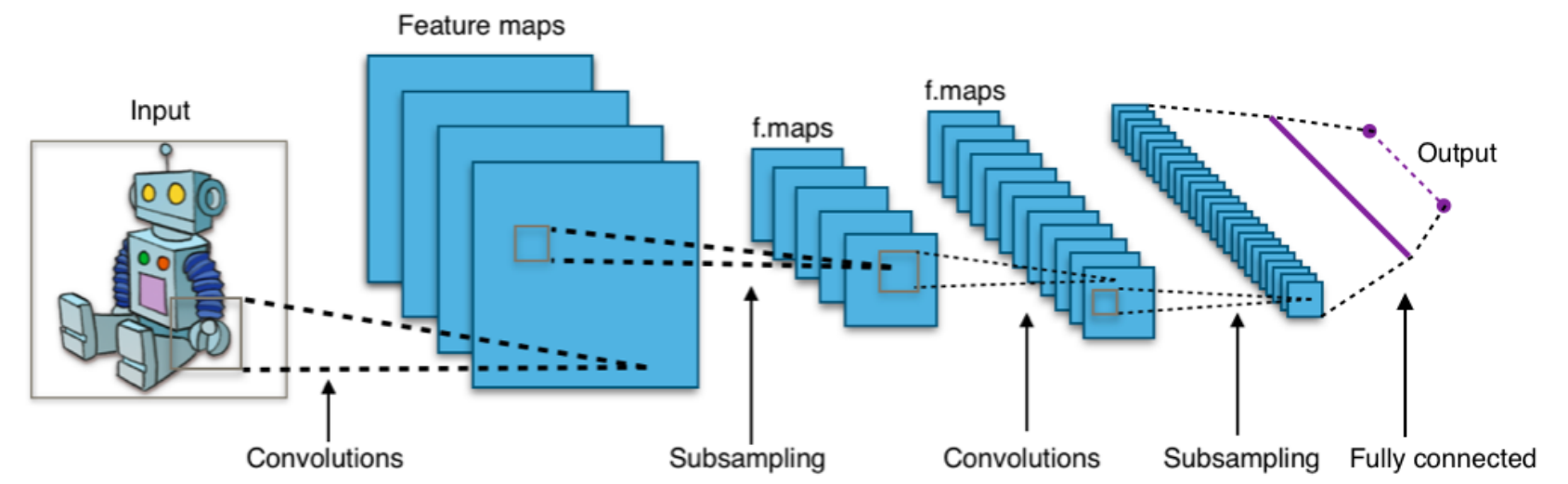
Convolutional Neural Network - LeCun (1989)

Architecture



Convolutional Neural Network - LeCun (1989)

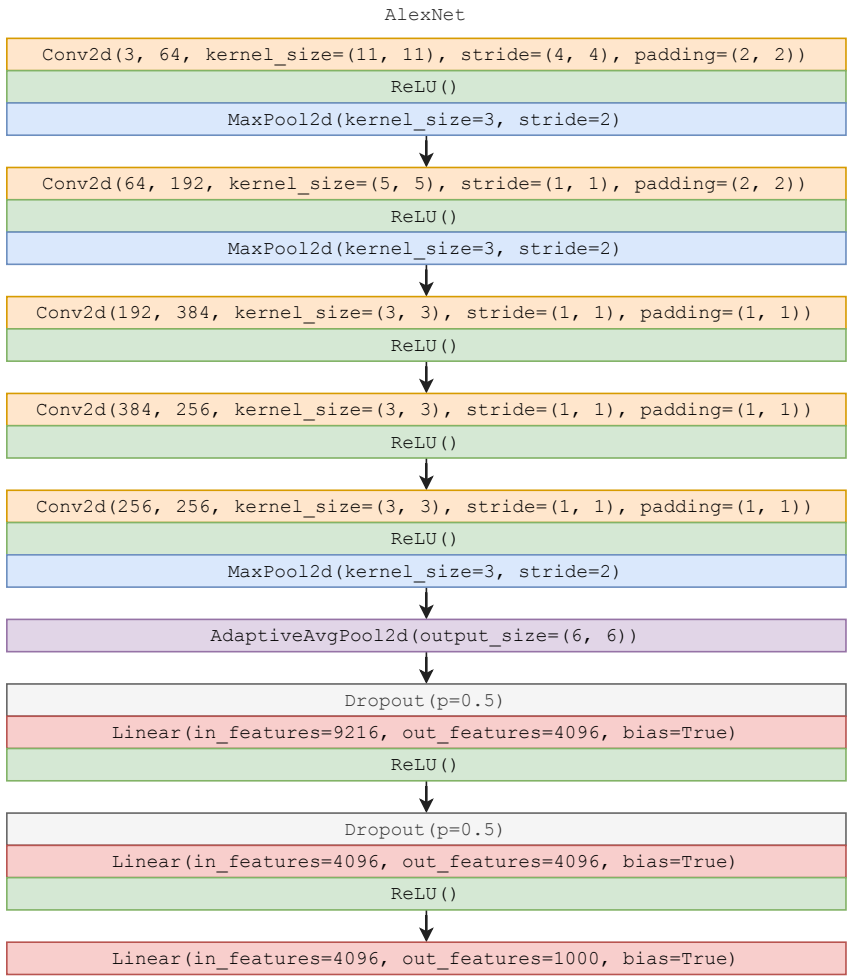
Architecture



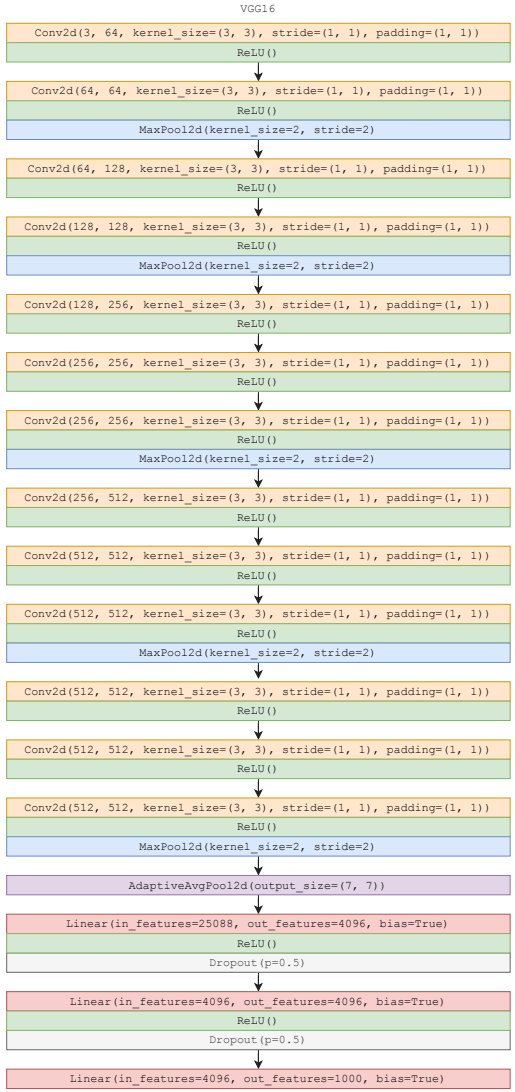
Learning rule

- Each neuron still performs a weighted sum, so as before apply a stochatic gradient descent on the loss function.
- The differences :
  - the weights (kernel) is shared between the neurons of a convolutional layer, so the gradient is aggredated (sum or average)
  - for the pooling layer, either retropropagate where the data come from (when using a max operation), or do as for any other weighted sum (when using an average operation)

Convolutional Neural Network - AlexNet (2012)



Convolutional Neural Network - VGG16 (2014)

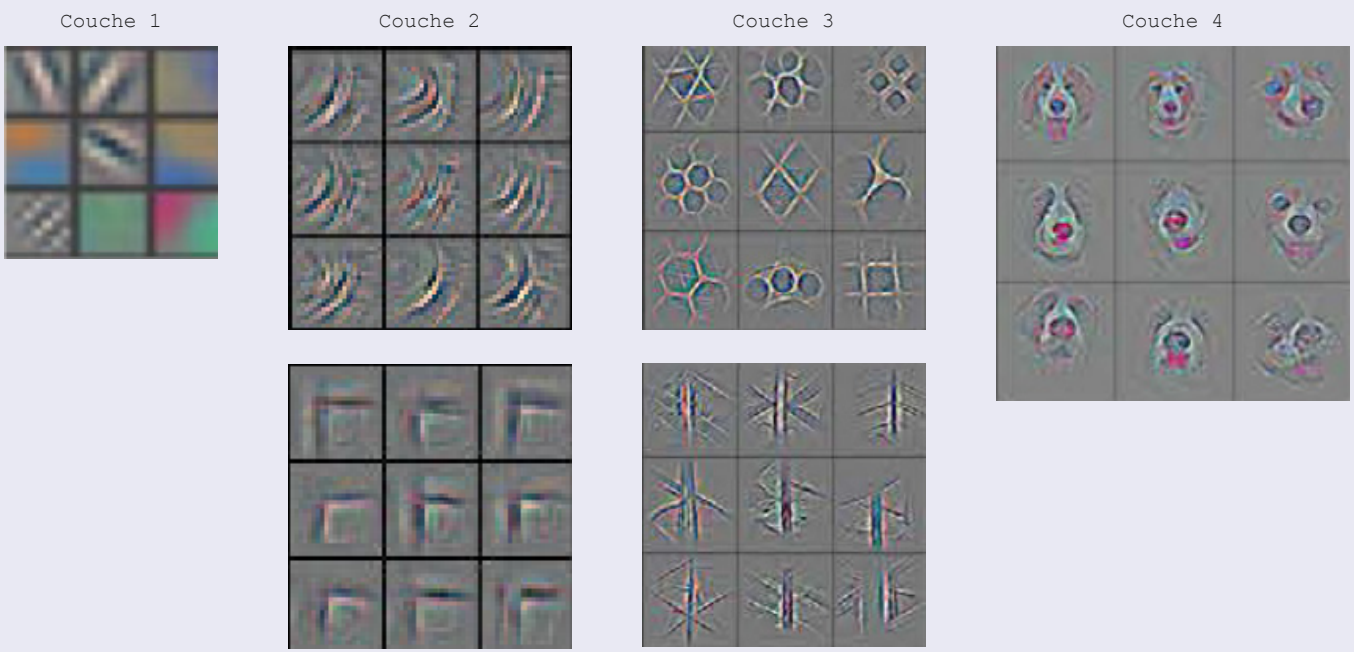


CNN summary

Principles

- Use of convolution for translation invariance and weights sharing
- Pooling to reduce dimensionnality
- A MLP on the top of the architecture as there is no more spatial structure
- Architecture adaptable to 1D (audio), 3D (video), or graphs, ...

Learned features



If you want to know more : https://youtu.be/AgkfIQ4IGaM

Deep learning summary

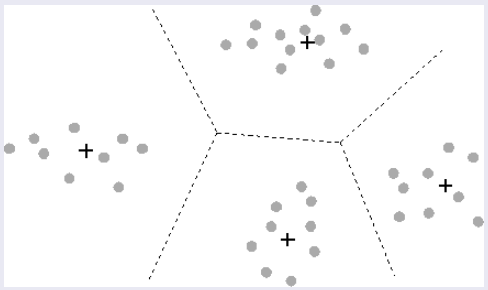
Why does it work ?

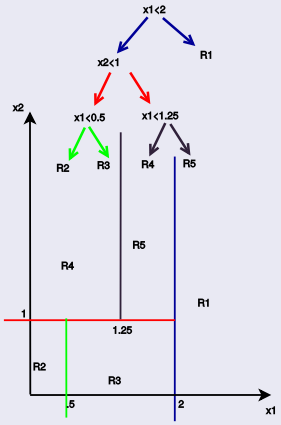
- Universal approximator + a huge amount of data / GPUs ... only part of the story
- *Features* (vs prototypes) in machine learning
- (Exponential) abstraction power in theory
- Local minima seems quite good
- Deals more easily with data/task in practice

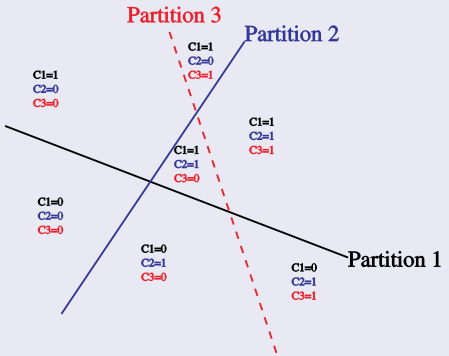
Deep learning summary

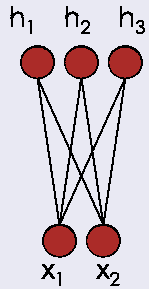
Why does it work ?

- Universal approximator + a huge amount of data / GPUs ... only part of the story
- *Features* (vs prototypes) in machine learning









- (Exponential) abstraction power in theory
- Local minima seems quite good
- Deals more easily with data/task in practice

Deep learning summary

Why does it work ?

- Universal approximator + a huge amount of data / GPUs ... only part of the story
- *Features* (vs prototypes) in machine learning
- (Exponential) abstraction power in theory
- Local minima seems quite good
- Deals more easily with data/task in practice

Deep learning summary

Why does it work ?

- Universal approximator + a huge amount of data / GPUs ... only part of the story
- *Features* (vs prototypes) in machine learning
- (Exponential) abstraction power in theory
- Local minima seems quite good
  Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., LeCun, Y. (2015). The loss surfaces of multilayer networks. In Artificial intelligence and statistics.
- Deals more easily with data/task in practice

Deep learning summary

Why does it work ?

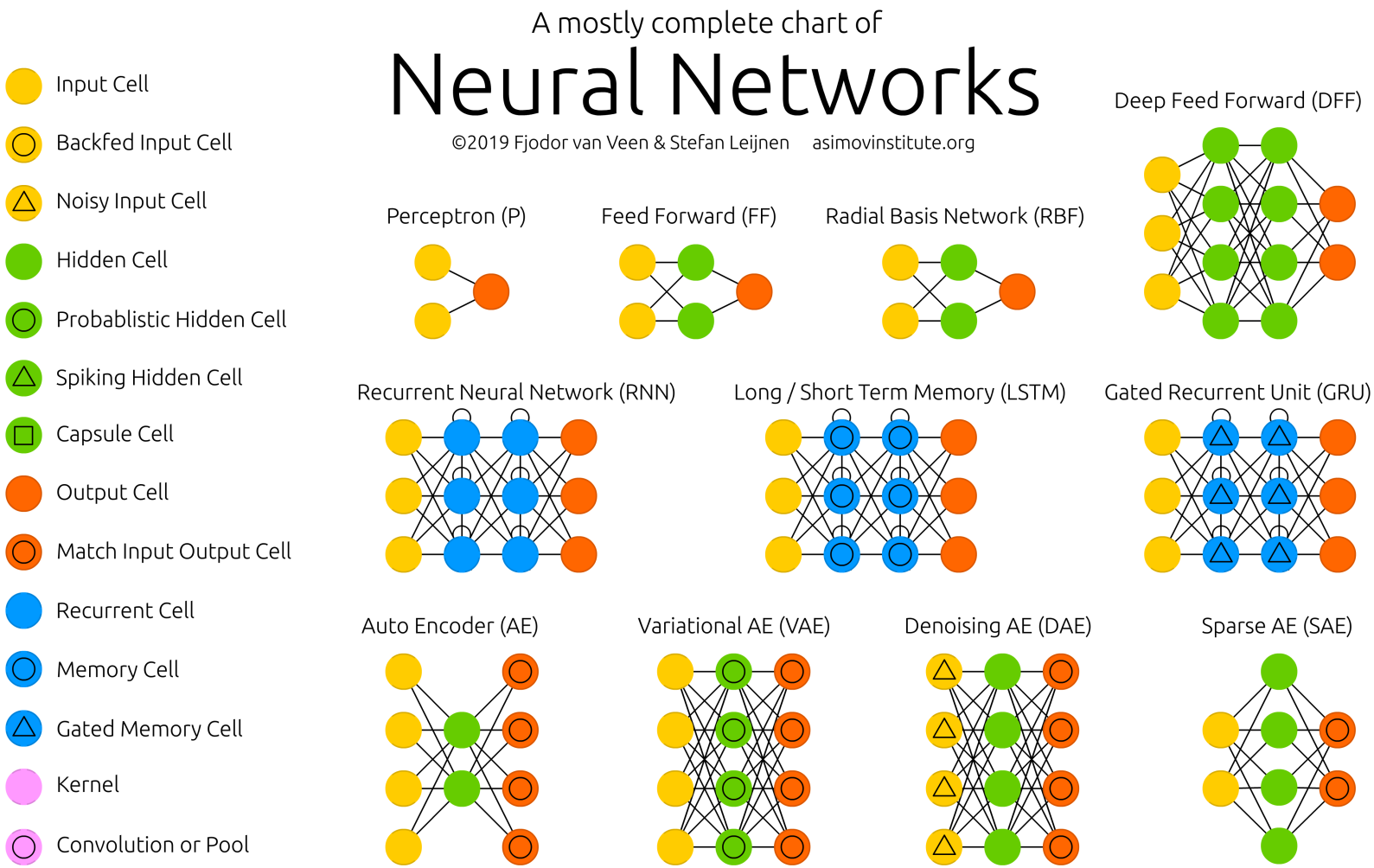
- Universal approximator + a huge amount of data / GPUs ... only part of the story
- *Features* (vs prototypes) in machine learning
- (Exponential) abstraction power in theory
- Local minima seems quite good
- Deals more easily with data/task in practice

	Architecture	# Param.	# Hidden units	Err.
DNN	2000-2000 + dropout	∼10M	4k	57.8%
SNN-30k	128c-p-1200L-30k + dropout input&hidden	∼70M	∼190k	21.8%
single-layer feature extraction	4000c-p followed by SVM	∼125M	∼3.7B	18.4%
CNN[11] (no augmentation)	64c-p-64c-p-64c-p-16lc + dropout on lc	∼10k	∼110k	15.6%
CNN[21] (no augmentation)	64c-p-64c-p-128c-p-fc + dropout on fc and stochastic pooling	∼56k	∼120k	15.13%
teacher CNN (no augmentation)	128c-p-128c-p-128c-p-1kfc + dropout on fc and stochastic pooling	∼35k	∼210k	**12.0%**
ECNN (no augmentation)	ensemble of 4 CNNs	∼140k	∼840k	**11.0%**
SNN-CNN-MIMIC-30k trained on a single CNN	64c-p-1200L-30k with no regularization	∼54M	∼110k	**15.4%**
SNN-CNN-MIMIC-30k trained on a single CNN	128c-p-1200L-30k with no regularization	∼70M	∼190k	**15.1%**
SNN-ECNN-MIMIC-30k trained on ensemble	128c-p-1200L-30k with no regularization	∼70M	∼190k	**14.2%**

Table 2: Comparison of shallow and deep models: classification error rate on CIFAR-10. Key: c, convolution layer; p, pooling layer; lc, locally connected layer; fc, fully connected layer

Ba, J., & Caruana, R. (2014). Do deep nets really need to be deep ?. In Advances in neural information processing systems .

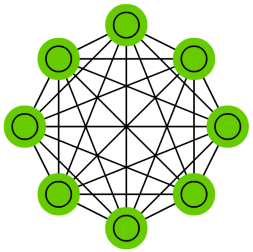
Deep learning zoo



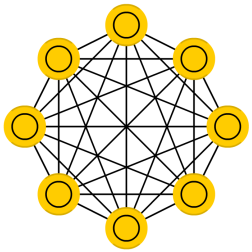
https ://www.asimovinstitute.org/neural-network-zoo/

Deep learning zoo

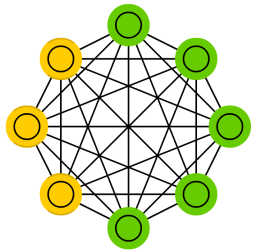
Markov Chain (MC)



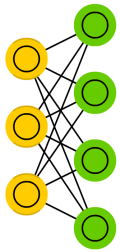
Hopfield Network (HN)



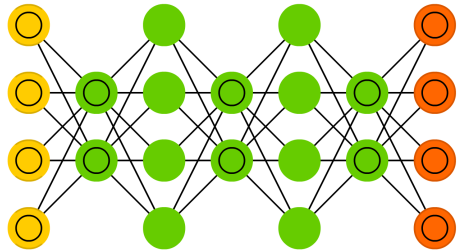
Boltzmann Machine (BM)



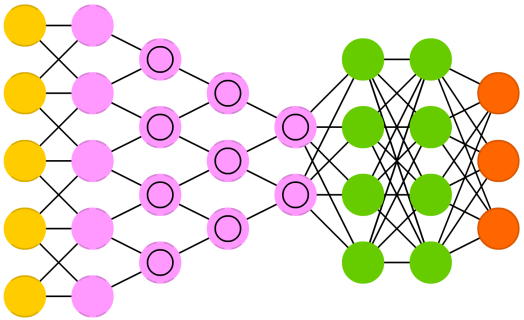
Restricted BM (RBM)



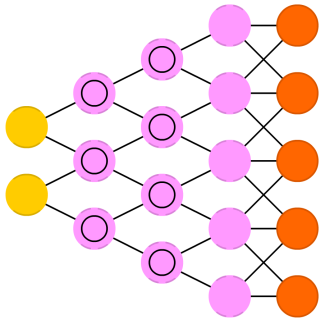
Deep Belief Network (DBN)



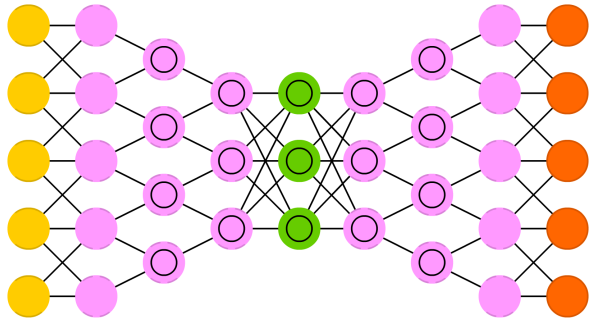
Deep Convolutional Network (DCN)



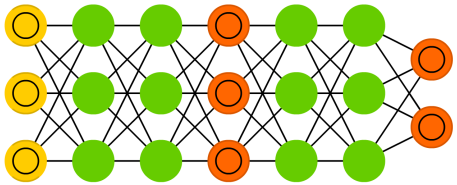
Deconvolutional Network (DN)



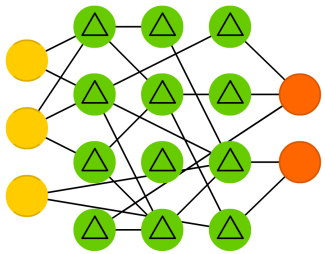
Deep Convolutional Inverse Graphics Network (DCIGN)



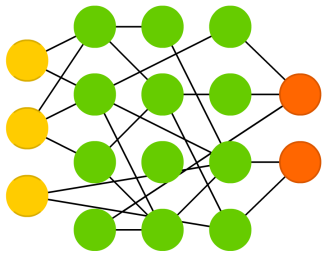
Generative Adversarial Network (GAN)



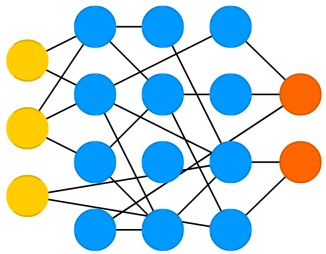
Liquid State Machine (LSM)



Extreme Learning Machine (ELM)



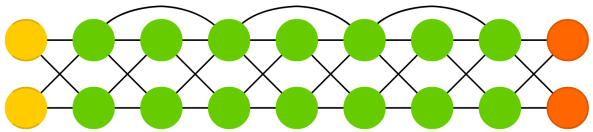
Echo State Network (ESN)



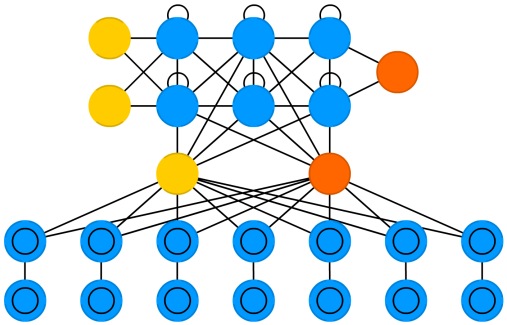
https ://www.asimovinstitute.org/neural-network-zoo/

Deep learning zoo

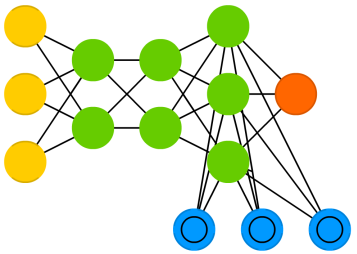
Deep Residual Network (DRN)



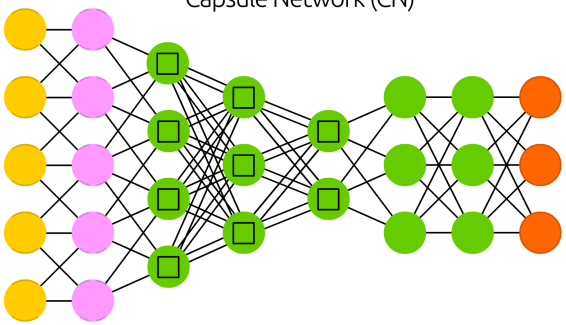
Differentiable Neural Computer (DNC)



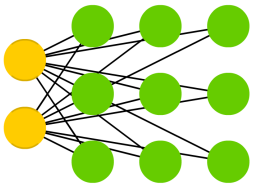
Neural Turing Machine (NTM)



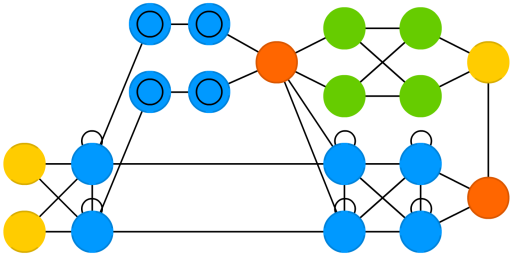
Capsule Network (CN)



Kohonen Network (KN)



Attention Network (AN)



https ://www.asimovinstitute.org/neural-network-zoo/

How to make it work in practice ?

- Data (pre) processing
- Choice of the model
- Training
- Getting the better performances

How to make it work in practice ?

- Data (pre) processing
  - Balanced/representative data
  - Data augmentations : for images : change color, zoom, orientation, ...
  - Standardised data : $\frac{x-\bar{x}}{\sigma(x)}$
  - Use of mini batchs
- Choice of the model
- Training
- Getting the better performances

How to make it work in practice ?

- Data (pre) processing
  - Balanced/representative data
  - Data augmentations : for images : change color, zoom, orientation, ...
  - Standardised data : $\frac{x-\bar{x}}{\sigma(x)}$
  - Use of mini batchs
- Choice of the model
- Training
- Getting the better performances

How to make it work in practice ?

- Data (pre) processing
  - Balanced/representative data
  - Data augmentations : for images : change color, zoom, orientation, ...
  - Standardised data : $\frac{x-\bar{x}}{\sigma(x)}$
  - Use of mini batchs
- Choice of the model
- Training
- Getting the better performances

How to make it work in practice ?

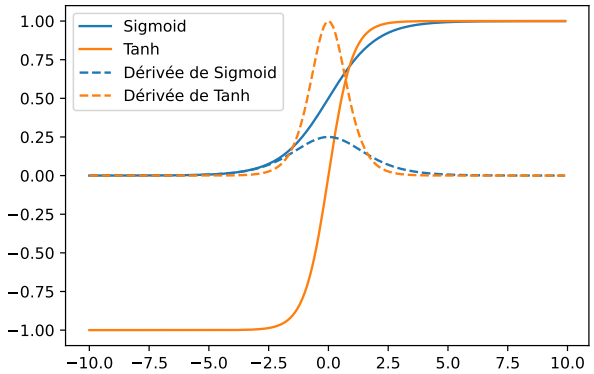
- Data (pre) processing
  - Balanced/representative data
  - Data augmentations : for images : change color, zoom, orientation, ...
  - Standardised data : $\frac{x-\bar{x}}{\sigma(x)}$
  - Use of mini batchs
- Choice of the model
- Training
- Getting the better performances

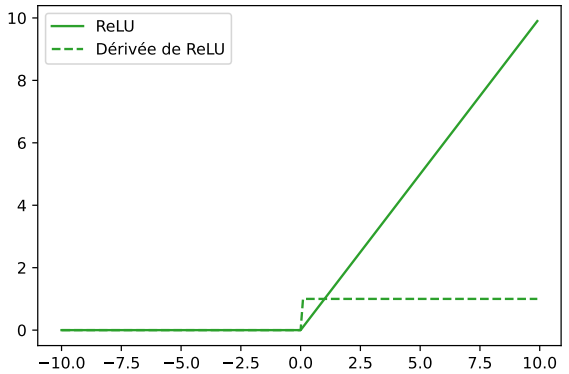
How to make it work in practice ?

- Data (pre) processing
- Choice of the model
  - Type of architecture w.r.t. the data / problem
  - Choice of the activation function
- Training
- Getting the better performances

How to make it work in practice ?

- Data (pre) processing
- Choice of the model
  - Type of architecture w.r.t. the data / problem
  - Choice of the activation function





- Training
- Getting the better performances

How to make it work in practice ?

- Data (pre) processing
- Choice of the model
- Training
  - Loss function :
    - regression : (Mean) squared error : $\frac{1}{2}\sum_{x,i}(t_i - y_i)^2$
    - classification : Cross entropy : $\sum_x -log\frac{e^{y_t}}{\sum_i e^{y_i}}$
  - Regularisation : $+||w||$ in the loss function
  - Noise / *drop out* : some random values are set to 0
  - Choice of the optimizer
    - SGD : the basic
    - momentum : adds some inertia to the gradient
    - Adagrad : adapts the learning rate (depending on the gradient)
    - Adam : combines both previous points
- Getting the better performances

How to make it work in practice ?

- Data (pre) processing
- Choice of the model
- Training
  - Loss function :
    - regression : (Mean) squared error : $\frac{1}{2}\sum_{x,i}(t_i - y_i)^2$
    - classification : Cross entropy : $\sum_x -log\frac{e^{y_t}}{\sum_i e^{y_i}}$
  - Regularisation : $+||w||$ in the loss function
  - Noise / *drop out* : some random values are set to 0
  - Choice of the optimizer
    - SGD : the basic
    - momentum : adds some inertia to the gradient
    - Adagrad : adapts the learning rate (depending on the gradient)
    - Adam : combines both previous points
- Getting the better performances

How to make it work in practice ?

- Data (pre) processing
- Choice of the model
- Training
  - Loss function :
    - regression : (Mean) squared error : $\frac{1}{2}\sum_{x,i}(t_i - y_i)^2$
    - classification : Cross entropy : $\sum_x -log\frac{e^{y_t}}{\sum_i e^{y_i}}$
  - Regularisation : $+||w||$ in the loss function
  - Noise / *drop out* : some random values are set to 0
  - Choice of the optimizer
    - SGD : the basic
    - momentum : adds some inertia to the gradient
    - Adagrad : adapts the learning rate (depending on the gradient)
    - Adam : combines both previous points
- Getting the better performances

How to make it work in practice ?

- Data (pre) processing
- Choice of the model
- Training
  - Loss function :
    - regression : (Mean) squared error : $\frac{1}{2}\sum_{x,i}(t_i - y_i)^2$
    - classification : Cross entropy : $\sum_x -log\frac{e^{y_t}}{\sum_i e^{y_i}}$
  - Regularisation : $+||w||$ in the loss function
  - Noise / *drop out* : some random values are set to 0
  - Choice of the optimizer
    - SGD : the basic
    - momentum : adds some inertia to the gradient
    - Adagrad : adapts the learning rate (depending on the gradient)
    - Adam : combines both previous points
- Getting the better performances

How to make it work in practice ?

- Data (pre) processing
- Choice of the model
- Training
- Getting the better performances
  - Hyperparameters : define default values ...
    - for the MLP, usually a big 1st layer, then decreasing size
    - for the CNN, usually the number of channels increases at each layer (to "compensate" the decreasing size of the feature maps)
  - ... then empirally (typically with a *grid search*) find what works on the validation set
  - *Fine tuning* : use of a ("general") pre trained model that is locally adapted to the data

How to make it work in practice ?

- Data (pre) processing
- Choice of the model
- Training
- Getting the better performances
  - Hyperparameters : define default values ...
    - for the MLP, usually a big 1^{st} layer, then decreasing size
    - for the CNN, usually the number of channels increases at each layer (to "compensate" the decreasing size of the feature maps)
  - ... then empirally (typically with a *grid search*) find what works on the validation set
  - *Fine tuning* : use of a ("general") pre trained model that is locally adapted to the data

How to make it work in practice ?

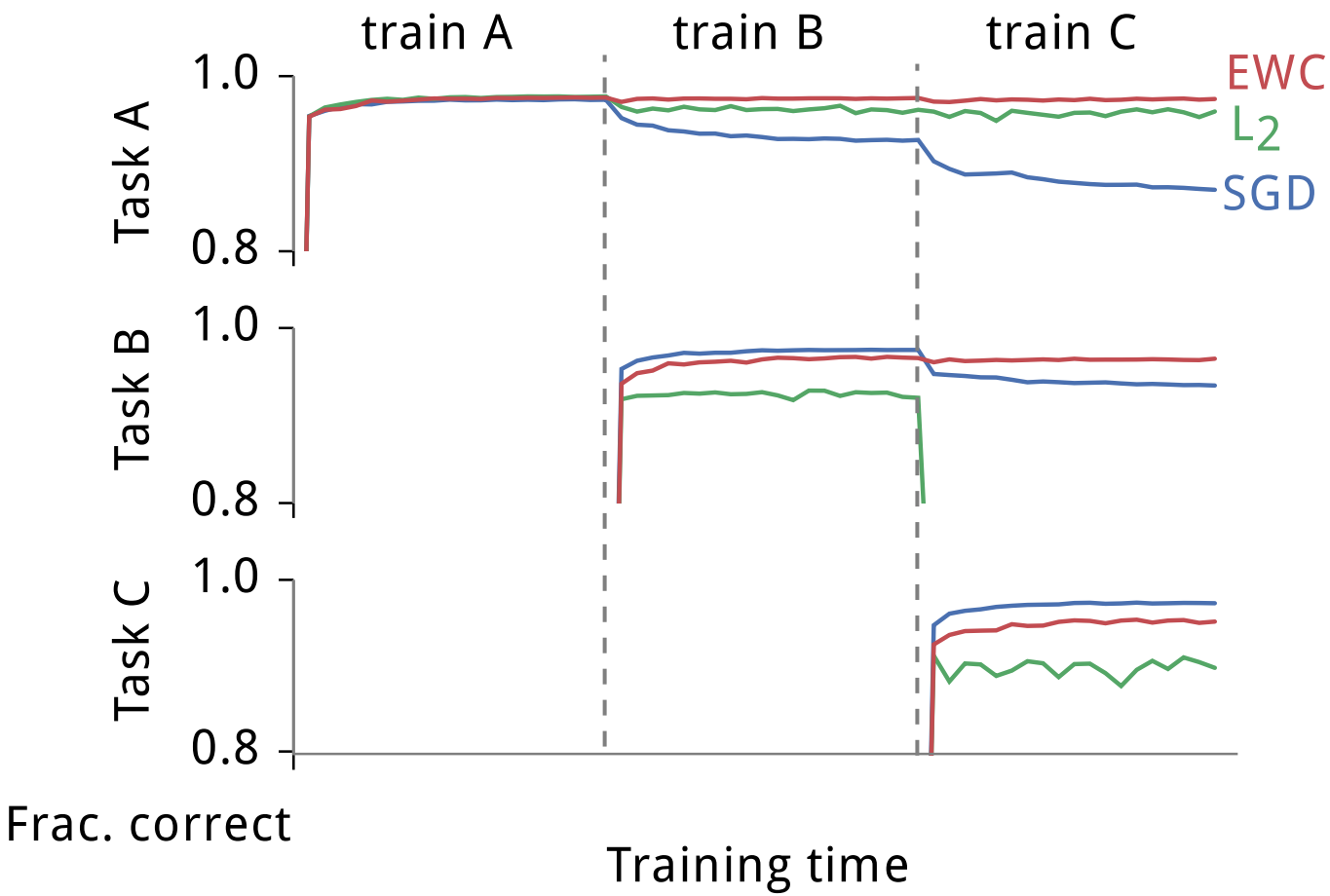
- Data (pre) processing
- Choice of the model
- Training
- Getting the better performances
  - Hyperparameters : define default values ...
    - for the MLP, usually a big 1st layer, then decreasing size
    - for the CNN, usually the number of channels increases at each layer (to "compensate" the decreasing size of the feature maps)
  - ... then empirally (typically with a *grid search*) find what works on the validation set
  - *Fine tuning* : use of a ("general") pre trained model that is locally adapted to the data

Limitations - Over training ?

data aug	dropout	weight decay	top-1 train	top-5 train	top-1 test	top-5 test
ImageNet 1000 classes with the original labels						
yes	yes	yes	92.18	99.21	77.84	93.92
yes	no	no	92.33	99.17	72.95	90.43
no	no	yes	90.60	100.0	67.18 (72.57)	86.44 (91.31)
no	no	no	99.53	100.0	59.80 (63.16)	80.38 (84.49)
Alexnet (Krizhevsky et al., 2012)			-	-	-	83.6
ImageNet 1000 classes with random labels						
no	yes	yes	91.18	97.95	0.09	0.49
no	no	yes	87.81	96.15	0.12	0.50
no	no	no	95.20	99.14	0.11	0.56

Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. arXiv preprint arXiv :1611.03530.

Limitations - Catastrophic forgetting



Kirkpatrick, James, et al. "Overcoming catastrophic forgetting in neural networks." Proceedings of the national academy of sciences (2017) : 201611835.

Limitations - Noise

Limitations - Bias

Classifier	Metric	All	F	M	Darker	Lighter	DF	DM	LF	LM
MSFT	PPV(%)	93.7	89.3	97.4	87.1	99.3	79.2	94.0	98.3	**100**
	Error Rate(%)	6.3	10.7	2.6	12.9	0.7	**20.8**	6.0	1.7	0.0
	TPR (%)	93.7	96.5	91.7	87.1	99.3	92.1	83.7	**100**	98.7
	FPR (%)	6.3	8.3	3.5	12.9	0.7	**16.3**	7.9	1.3	0.0
Face++	PPV(%)	90.0	78.7	99.3	83.5	95.3	65.5	**99.3**	94.0	99.2
	Error Rate(%)	10.0	21.3	0.7	16.5	4.7	**34.5**	0.7	6.0	0.8
	TPR (%)	90.0	98.9	85.1	83.5	95.3	98.8	76.6	**98.9**	92.9
	FPR (%)	10.0	14.9	1.1	16.5	4.7	**23.4**	1.2	7.1	1.1
IBM	PPV(%)	87.9	79.7	94.4	77.6	96.8	65.3	88.0	92.9	**99.7**
	Error Rate(%)	12.1	20.3	5.6	22.4	3.2	**34.7**	12.0	7.1	0.3
	TPR (%)	87.9	92.1	85.2	77.6	96.8	82.3	74.8	**99.6**	94.8
	FPR (%)	12.1	14.8	7.9	22.4	3.2	**25.2**	17.7	5.20	0.4

Buolamwini, Joy, and Timnit Gebru. "Gender shades : Intersectional accuracy disparities in commercial gender classification." Conference on Fairness, Accountability and Transparency. 2018