Machine learning for graphs and with graphs

Graph neural networks

Titouan Vayer & Pierre Borgnat
email: titouan.vayer@inria.fr, pierre.borgnat@ens-lyon.fr

October 15, 2024

ENS DE LYON

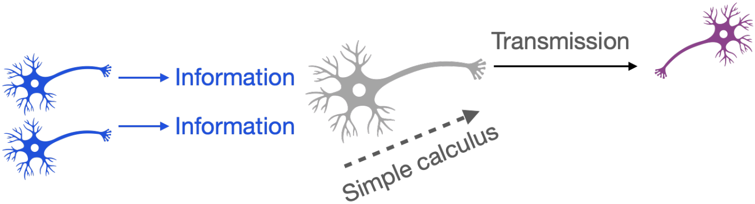
Table of contents

Table of contents

What is a neural network ?

Neural network is a certain family of functions **parametrized by weights**.

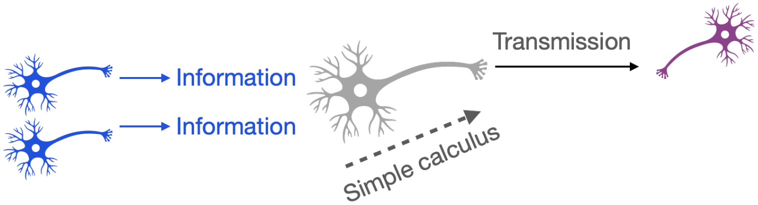
Built upon a biological analogy Rosenblatt 1958



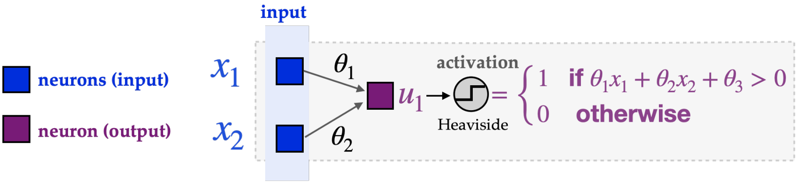
What is a neural network ?

Neural network is a certain family of functions **parametrized by weights**.

Built upon a biological analogy Rosenblatt 1958



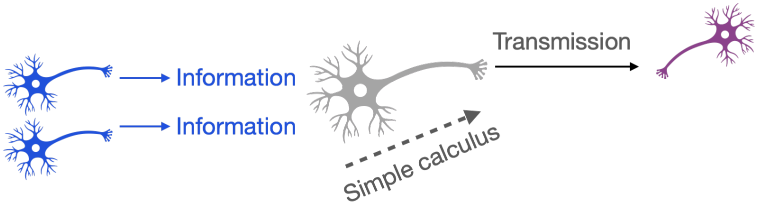
- First example $f(\mathbf{x} = (x_1, x_2)) = \text{activation}(\theta_1 x_1 + \theta_2 x_2 + \theta_3)$:



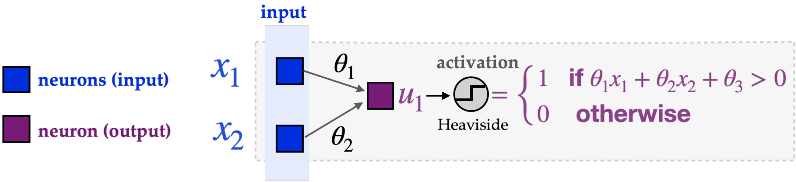
What is a neural network ?

Neural network is a certain family of functions **parametrized by weights**.

Built upon a biological analogy Rosenblatt 1958



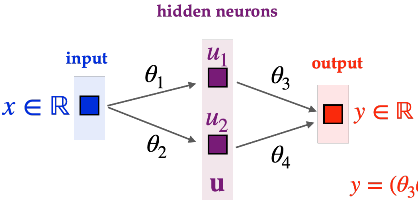
- Second example $f(\mathbf{x} = (x_1, x_2)) = \text{activation}(\theta_1 x_1 + \theta_2 x_2 + \theta_3)$:



What is a neural network ?

Feed-forward neural networks

- Linear neural network:



bias

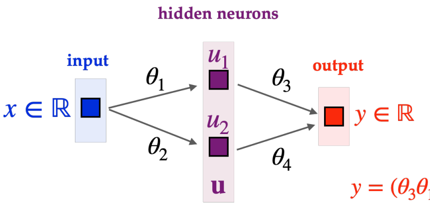
$$u_1 = \theta_1 x + b_1$$
$$u_2 = \theta_2 x + b_2$$
$$y = \theta_3 u_1 + \theta_4 u_2 + b_3$$
$$y = \theta_3(\theta_1 x + b_1) + \theta_4(\theta_2 x + b_2) + b_3$$

$$y = (\theta_3\theta_1 + \theta_4\theta_2)x + \theta_3 b_1 + \theta_4 b_2 + b_3$$

What is a neural network ?

Feed-forward neural networks

- Linear neural network:

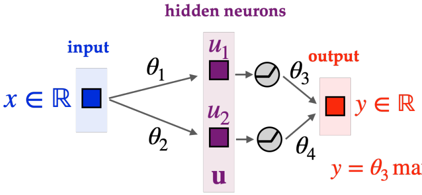


bias

$$u_1 = \theta_1 x + b_1$$
$$u_2 = \theta_2 x + b_2$$
$$y = \theta_3 u_1 + \theta_4 u_2 + b_3$$
$$y = \theta_3(\theta_1 x + b_1) + \theta_4(\theta_2 x + b_2) + b_3$$

$$y = (\theta_3\theta_1 + \theta_4\theta_2)x + \theta_3 b_1 + \theta_4 b_2 + b_3$$

- Non-linearity:



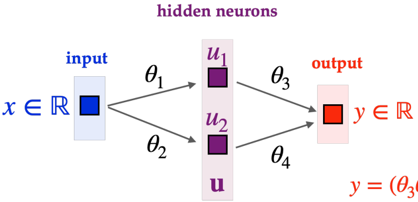
$$u_1 = \theta_1 x + b_1$$
$$u_2 = \theta_2 x + b_2$$
$$y = \theta_3 \max\{u_1, 0\} + \theta_4 \max\{u_2, 0\} + b_3$$

$$y = \theta_3 \max\{\theta_1 x + b_1, 0\} + \theta_4 \max\{\theta_2 x + b_2, 0\} + b_3$$

What is a neural network ?

Feed-forward neural networks

- Linear neural network:

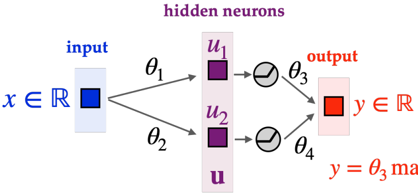


bias

$$u_1 = \theta_1 x + b_1$$
$$u_2 = \theta_2 x + b_2$$
$$y = \theta_3 u_1 + \theta_4 u_2 + b_3$$
$$y = \theta_3(\theta_1 x + b_1) + \theta_4(\theta_2 x + b_2) + b_3$$

$$y = (\theta_3\theta_1 + \theta_4\theta_2)x + \theta_3 b_1 + \theta_4 b_2 + b_3$$

- Non-linearity:



$$u_1 = \theta_1 x + b_1$$
$$u_2 = \theta_2 x + b_2$$
$$y = \theta_3 \max\{u_1, 0\} + \theta_4 \max\{u_2, 0\} + b_3$$

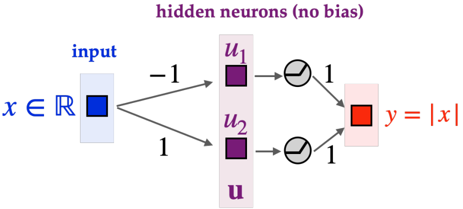
$$y = \theta_3 \max\{\theta_1 x + b_1, 0\} + \theta_4 \max\{\theta_2 x + b_2, 0\} + b_3$$

- Find a neural network that implements the function $f(x) = |x|$.

What is a neural network ?

Feed-forward neural networks

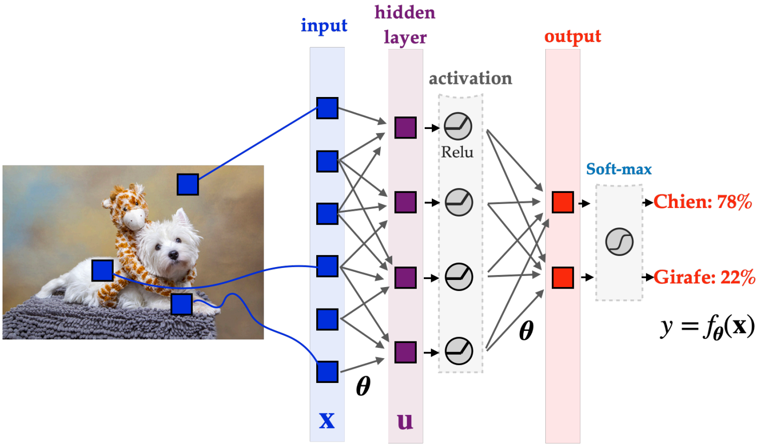
- Find a neural network that implements the function $f(x) = |x|$



$$u_1 = -x$$
$$u_2 = x$$
$$y = \max\{u_1, 0\} + \max\{u_2, 0\}$$
$$y = \max\{x, 0\} + \max\{-x, 0\}$$

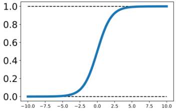
What is a neural network ?

Feed-forward neural networks



Soft-max:

$$\text{sotfmax}[u]_i = \frac{\exp(u_i)}{\sum_j \exp(u_j)}$$



What is a neural network ?

Feed-forward neural networks

- Feed-forward NN are function of the form

$$f(\mathbf{x}) = T_K \circ \sigma_{K-1} \circ \cdots \circ \sigma_1 \circ T_1(\mathbf{x})$$

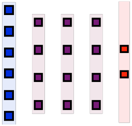
where $T_k(\mathbf{z}) = \mathbf{W}^{(k)}\mathbf{z} + \mathbf{b}^{(k)}$

and σ_k pointwise activation function.

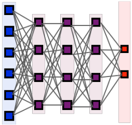
- All the weights: $\boldsymbol{\theta} = (\mathbf{W}^{(1)}, \cdots, \mathbf{W}^{(K)}, \mathbf{b}^{(1)}, \cdots \mathbf{b}^{(K)})$.
- Depending on the task the output of a NN is also transformed $g(\mathbf{x}) = \mathrm{norm}(f(\mathbf{x}))$.
- E.g. $f : \mathbb{R}^d \to \mathbb{R}$ and $g : \mathbb{R}^d \to (0, 1)$ for binary classification with $\mathrm{norm}(u) = 1/(1 + \exp(-u))$ (logistic/sigmoid function).

What is a neural network ?

A zoo of architectures

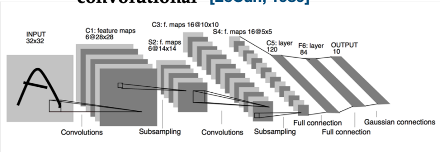


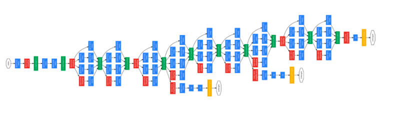
multi-layers



fully-connected

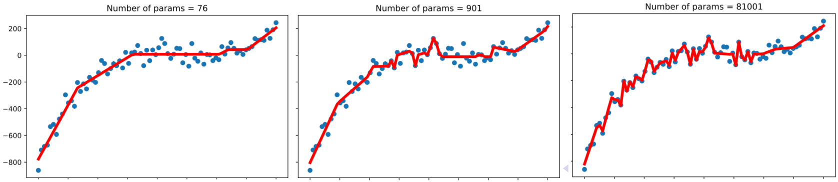
convolutional **[LeCun, 1989]**





deep-learning **also: generative, recurrent, transformers, attention layer transformers...**

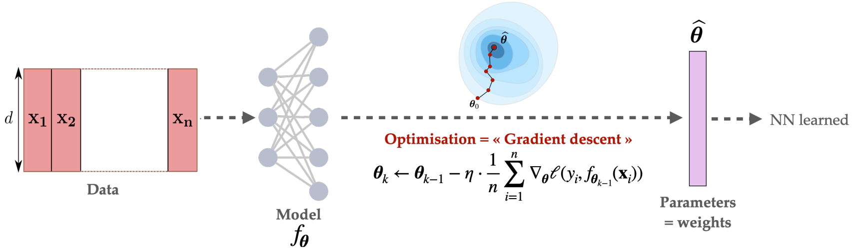
Richness of neural network



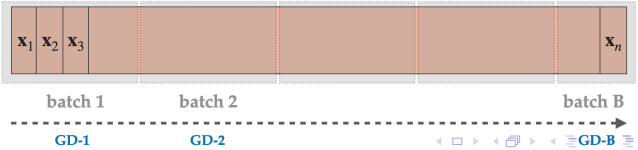
Neural network in practice

The (very) big picture

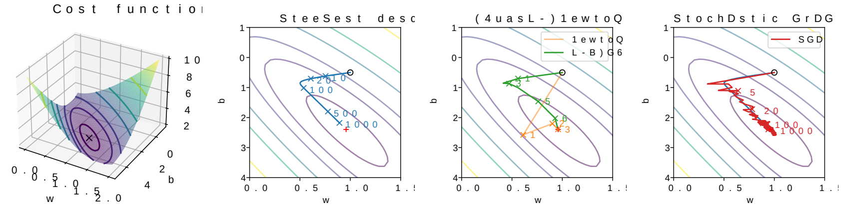
Find the weights that minimizes the empirical minimization loss.



- In practice gradient descent very slow.
- We use stochastic gradient descents (and variations) on batches of the data.



(almost) All optimization in one slide



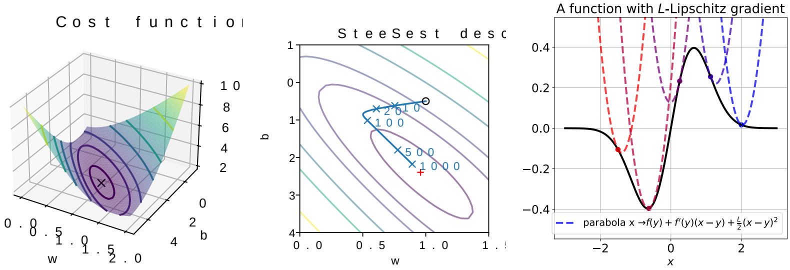
Principle

- Minimize a smooth function $J(\boldsymbol{\theta})$ using its gradient (or $\approx$).
- Initialize a vector $\boldsymbol{\theta}^{(0)}$ and update it at each iteration k as:

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \mu_k \mathbf{d}_k$$

  where μ_k is a step and $\mathbf{d}_k$ is a descent direction $\mathbf{d}_k^\top \nabla J(\boldsymbol{\theta}^{(k)}) < 0$.
- Classical descent directions are :
  - **Steepest descent**: $\mathbf{d}_k = -\nabla J(\boldsymbol{\theta}^{(k)})$ (a.k.a. Gradient descent).
  - (Quasi) **Newton**: $\mathbf{d}_k = -(\nabla^2 J(\boldsymbol{\theta}^{(k)}))^{-1} \nabla J(\boldsymbol{\theta}^{(k)})$, $\nabla^2 J$ is the Hessian.
  - **Stochastic Gradient Descent** : $\mathbf{d}_k = -\tilde{\nabla} J(\boldsymbol{\theta}^{(k)})$ with approx. gradient.
- For NN: gradient computed with **automatic differentiation** (TD).

(almost) All optimization in two slides...



Why is this a good idea ? (on the board)

Let $J : \mathbb{R}^D \to \mathbb{R}$ with L-Lipschitz gradient[1] and $J^\star := \min_{\boldsymbol{\theta}} \; J(\boldsymbol{\theta}) > -\infty$. Then, provided that $0 < \mu_k < \frac{2}{L}$, the iterations $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \mu_k \nabla J(\boldsymbol{\theta}^{(k)})$ satisfy

$$J(\boldsymbol{\theta}^{(k+1)}) < J(\boldsymbol{\theta}^{(k)}) \text{ (decrease the objective function)}$$

$$\lim_{k \to +\infty} \nabla J(\boldsymbol{\theta}^{(k)}) = \mathbf{0} \text{ (critical point)}$$

[1] it means that $\forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathbb{R}^d, \; \|\nabla J(\boldsymbol{\theta}_1) - \nabla J(\boldsymbol{\theta}_2)\|_2 \leq L \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_2.$

(almost) All optimization in three slides...

Be aware of local minima

- When the functions are not convex, GD and its variants can fall into bad local minima.
- **Neural networks are not convex w.r.t. the optimized parameters !**

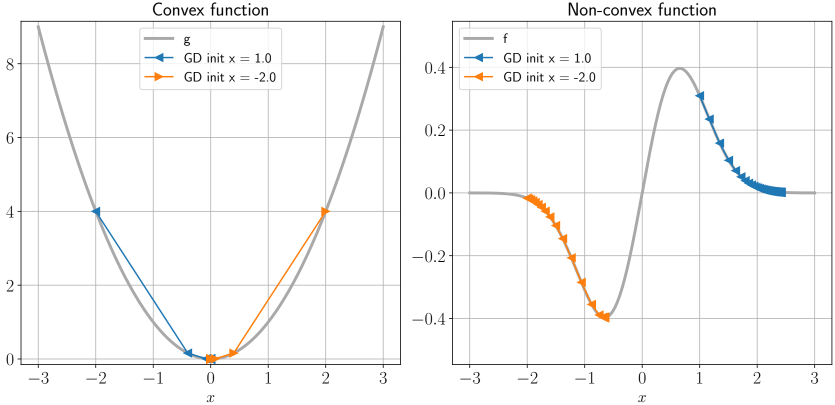


Table of contents

First simple neural network: logistic regression

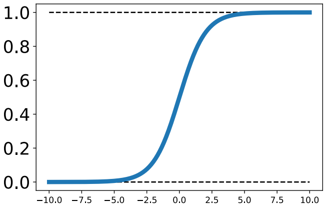
- It is a **classification method**: input $(\mathbf{x}_i)_i \in \mathbb{R}^d$ and $(y_i)_i \in \{+1, -1\}$.
- **Probabilistic model**: find a model $h_{\boldsymbol{\theta}}$ s.t. $\mathbb{P}(y = +1|\mathbf{x}) \approx h_{\boldsymbol{\theta}}(\mathbf{x})$.
- Bayes decision: $f(\mathbf{x}) = \text{sign}(\mathbb{P}(y = +1|\mathbf{x}) - \mathbb{P}(y = -1|\mathbf{x})) \in \{-1, +1\}$.

First simple neural network: logistic regression

- It is a **classification method**: input $(\mathbf{x}_i)_i \in \mathbb{R}^d$ and $(y_i)_i \in \{+1, -1\}$.
- **Probabilistic model**: find a model $h_{\boldsymbol{\theta}}$ s.t. $\mathbb{P}(y = +1|\mathbf{x}) \approx h_{\boldsymbol{\theta}}(\mathbf{x})$.
- Bayes decision: $f(\mathbf{x}) = \text{sign}(\mathbb{P}(y = +1|\mathbf{x}) - \mathbb{P}(y = -1|\mathbf{x})) \in \{-1, +1\}$.

The sigmoid function

$\sigma(z) = 1/(1 + \exp(-z))$.



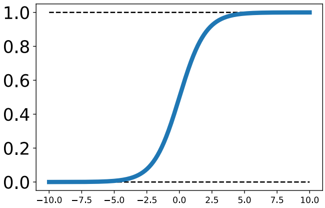
- Usually used to model probabilities.

First simple neural network: logistic regression

- It is a **classification method**: input $(\mathbf{x}_i)_i \in \mathbb{R}^d$ and $(y_i)_i \in \{+1, -1\}$.
- **Probabilistic model**: find a model $h_{\boldsymbol{\theta}}$ s.t. $\mathbb{P}(y = +1|\mathbf{x}) \approx h_{\boldsymbol{\theta}}(\mathbf{x})$.
- Bayes decision: $f(\mathbf{x}) = \text{sign}(\mathbb{P}(y = +1|\mathbf{x}) - \mathbb{P}(y = -1|\mathbf{x})) \in \{-1, +1\}$.

The sigmoid function

$\sigma(z) = 1/(1 + \exp(-z))$.

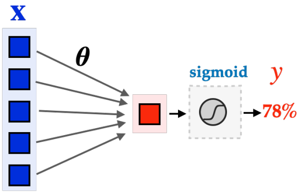


- Usually used to model probabilities.

The logistic regression model

The model is $\mathbb{P}(y = +1|\mathbf{x}) = \sigma(\boldsymbol{\theta}^\top \mathbf{x} + b)$.

- $\boldsymbol{\theta} \in \mathbb{R}^d$ are weights, $b \in \mathbb{R}$ is a bias that are to be optimized.
- It is a generalized linear model.
- Is is also a one layer neural-network (no hidden layer).



First simple neural network: logistic regression

One property

$\mathbb{P}(y=-1|\mathbf{x}) = 1 - \mathbb{P}(y=1|\mathbf{x}) = 1 - \sigma(\boldsymbol{\theta}^\top \mathbf{x} + b) = \sigma(-(\boldsymbol{\theta}^\top \mathbf{x} + b))$

First simple neural network: logistic regression

One property

$\mathbb{P}(y = -1|\mathbf{x}) = 1 - \mathbb{P}(y = 1|\mathbf{x}) = 1 - \sigma(\boldsymbol{\theta}^\top \mathbf{x} + b) = \sigma(-(\boldsymbol{\theta}^\top \mathbf{x} + b))$

Maximum likelihood estimation

Find $\boldsymbol{\theta} \in \mathbb{R}^d, b \in \mathbb{R}$ that maximize the (conditional) log-likelihood (board)

$$\begin{aligned}
&\sum_{i:y_i=1} \log \mathbb{P}(y_i = 1|\mathbf{x}_i) + \sum_{i:y_i=-1} \log \mathbb{P}(y_i = -1|\mathbf{x}_i) \\
&= \sum_{i:y_i=1} \log \sigma(\boldsymbol{\theta}^\top \mathbf{x}_i + b) + \sum_{i:y_i=-1} \log \sigma(-(\boldsymbol{\theta}^\top \mathbf{x} + b)) \\
&= \sum_{i=1}^{n} \log \sigma(y_i(\boldsymbol{\theta}^\top \mathbf{x}_i + b)) \,.
\end{aligned}$$

First simple neural network: logistic regression

One property

$\mathbb{P}(y=-1|\mathbf{x}) = 1 - \mathbb{P}(y=1|\mathbf{x}) = 1 - \sigma(\boldsymbol{\theta}^\top \mathbf{x} + b) = \sigma(-(\boldsymbol{\theta}^\top \mathbf{x} + b))$

Maximum likelihood estimation

Find $\boldsymbol{\theta} \in \mathbb{R}^d, b \in \mathbb{R}$ that maximize the (conditional) log-likelihood (board)

$$
\begin{aligned}
&\sum_{i:y_i=1} \log \mathbb{P}(y_i = 1|\mathbf{x}_i) + \sum_{i:y_i=-1} \log \mathbb{P}(y_i = -1|\mathbf{x}_i) \\
&= \sum_{i:y_i=1} \log \sigma(\boldsymbol{\theta}^\top \mathbf{x}_i + b) + \sum_{i:y_i=-1} \log \sigma(-(\boldsymbol{\theta}^\top \mathbf{x} + b)) \\
&= \sum_{i=1}^{n} \log \sigma(y_i(\boldsymbol{\theta}^\top \mathbf{x}_i + b)) \,.
\end{aligned}
$$

Minimizing the logistic loss

$$
\min_{\boldsymbol{\theta}, b} \sum_{i=1}^{n} \log \left[1 + \exp\left(-y_i(\boldsymbol{\theta}^\top \mathbf{x}_i + b) \right) \right] \,.
$$

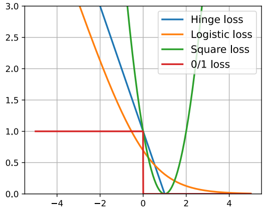
- **Convex problem**, can be solved with (Quasi) Newton's method.

First simple neural network: logistic regression

Remember your losses

With $f : \mathbb{R}^d \to \mathbb{R}$, many losses can be written as $\ell(y_i, f(\mathbf{x}_i)) = \Phi(y_i f(\mathbf{x}_i))$ with $\Phi \downarrow$.

- $\ell(y_i, f(\mathbf{x}_i)) = \mathbf{1}_{y_i f(\mathbf{x}_i) \leq 0}$.
- $\ell(y_i, f(\mathbf{x}_i)) = \max\{0, 1 - y_i f(\mathbf{x}_i)\}$.
- $\ell(y_i, f(\mathbf{x}_i)) = \log(1 + e^{-y_i f(\mathbf{x}_i)})$.

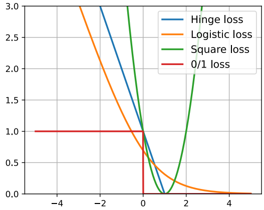


First simple neural network: logistic regression

Remember your losses

With $f : \mathbb{R}^d \to \mathbb{R}$, many losses can be written as $\ell(y_i, f(\mathbf{x}_i)) = \Phi(y_i f(\mathbf{x}_i))$ with $\Phi \downarrow$.

- $\ell(y_i, f(\mathbf{x}_i)) = \mathbf{1}_{y_i f(\mathbf{x}_i) \leq 0}$.
- $\ell(y_i, f(\mathbf{x}_i)) = \max\{0, 1 - y_i f(\mathbf{x}_i)\}$.
- $\ell(y_i, f(\mathbf{x}_i)) = \log(1 + e^{-y_i f(\mathbf{x}_i)})$.



And so ?

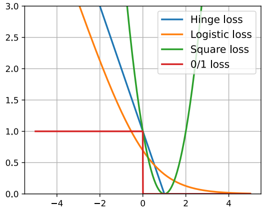
- Logistic regression = fitting $f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} + b$ with the logistic loss.
- The decision/prediction of the label is $\mathrm{sign}(f(\mathbf{x}))$.
- So it is a **linear decision boundary** (linear classification).

First simple neural network: logistic regression

Remember your losses

With $f : \mathbb{R}^d \to \mathbb{R}$, many losses can be written as $\ell(y_i, f(\mathbf{x}_i)) = \Phi(y_i f(\mathbf{x}_i))$ with $\Phi \downarrow$.

- $\ell(y_i, f(\mathbf{x}_i)) = \mathbf{1}_{y_i f(\mathbf{x}_i) \leq 0}$.
- $\ell(y_i, f(\mathbf{x}_i)) = \max\{0, 1 - y_i f(\mathbf{x}_i)\}$.
- $\ell(y_i, f(\mathbf{x}_i)) = \log(1 + e^{-y_i f(\mathbf{x}_i)})$.



And so ?

- Logistic regression = fitting $f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} + b$ with the logistic loss.
- The decision/prediction of the label is $\text{sign}(f(\mathbf{x}))$.
- So it is a **linear decision boundary** (linear classification).

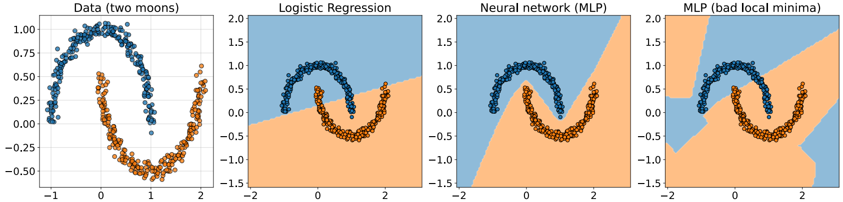
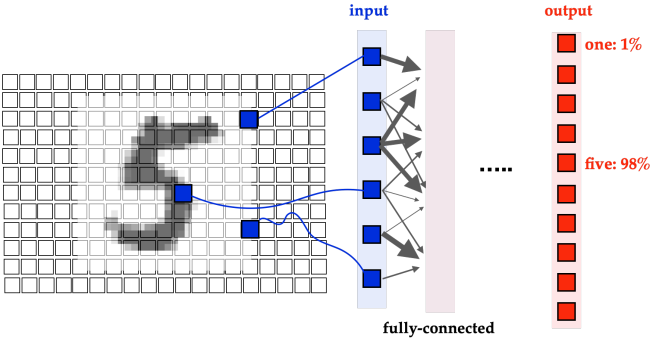


Table of contents

Convolutional neural networks

- The core block for deep learning on images.
- Induces an **implicit bias** on the architecture.

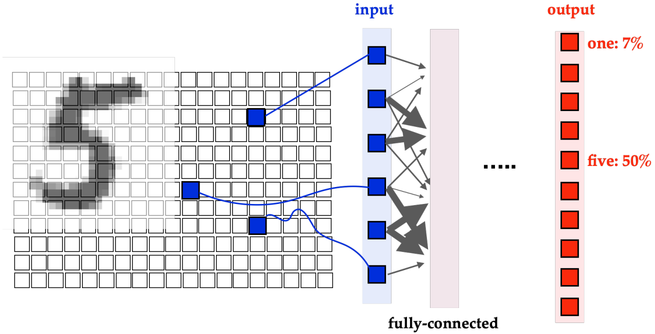
What could happen with a fully-connected architecture?



Convolutional neural networks

- The core block for deep learning on images.
- Induces an **implicit bias** on the architecture.

What could happen with a fully-connected architecture?



- We want a function that doesn't change if we only translate the image. We want a **translation invariant function**.
- Convolution: particular structure on the weights that induce **translation equivariance**.

Convolutional neural networks

Convolution/correlation of functions

Let $f, h \in L_2(\mathbb{R})$. The convolution $f * h \in L_2(\mathbb{R})$ is defined as

$$f * h(x) = \int_{-\infty}^{+\infty} f(t)h(x-t)\mathrm{d}t \text{ and } f \star h(x) = \int_{-\infty}^{+\infty} f(t)h(t+x)\mathrm{d}t$$

- **Translate a filter** h and then take the inner product with[2] f:

$$f \star h(x) = \langle \tau_{-x} h, f \rangle_{L_2(\mathbb{R})} .$$

- It weights the local contributions of f by a filter.

[2] $\tau_x f = t \to f(t-x)$

Convolutional neural networks

Convolution/correlation of functions

Let $f, h \in L_2(\mathbb{R})$. The convolution $f * h \in L_2(\mathbb{R})$ is defined as

$$f * h(x) = \int_{-\infty}^{+\infty} f(t)h(x-t)\mathrm{d}t \text{ and } f \star h(x) = \int_{-\infty}^{+\infty} f(t)h(t+x)\mathrm{d}t$$

- **Translate a filter** h and then take the inner product with[2] f:

$$f \star h(x) = \langle \tau_{-x} h, f \rangle_{L_2(\mathbb{R})} .$$

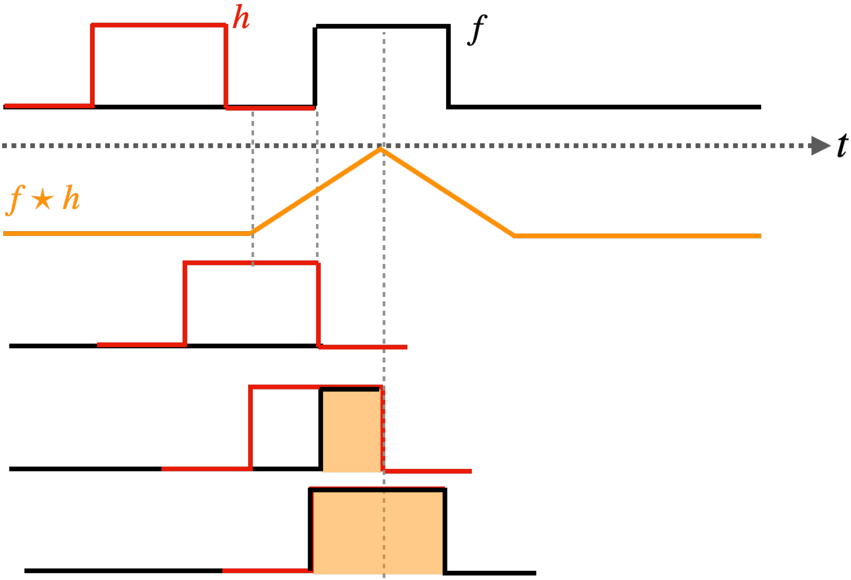
- It weights the local contributions of f by a filter.
- It is **translation equivariant**.

$$(\tau_x f) * h = \tau_x (f * h)$$

- If we translate the input, the output will be equally translated.

[2] $\tau_x f = t \to f(t-x)$

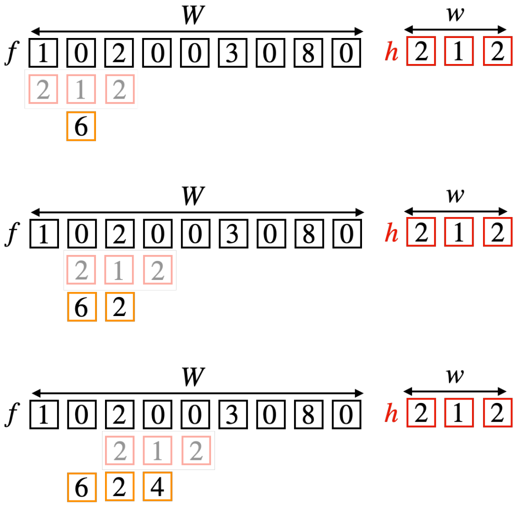
Convolutional neural networks



Convolutional neural networks

In practice convolutions are applied on discrete signals.

Discrete convolutions in 1D

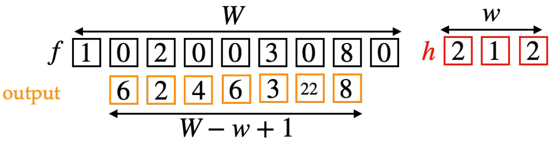


Question: size of the output ?

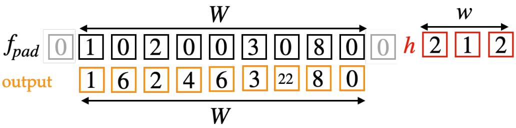
Convolutional neural networks

In practice convolutions are applied on discrete signals.

Discrete convolutions in 1D



- Padding strategies can be used to have output of the same size.



- Also stride can be used to move the filter from more than one pixel.

Convolution as matrix multiplication

Let $f = (f_1, \cdots, f_W), h = (h_1, \cdots, h_{w-1})$. In practice

$$\forall i \in [\![W - w + 1]\!], (f * h)_i = \sum_{j=1}^{w} f_{i-1+j} h_j = \sum_{n=i}^{i-1+w} f_n h_{n-i+1} \qquad (1)$$

Convolution as matrix multiplication

Let $f = (f_1, \cdots, f_W), h = (h_1, \cdots, h_{w-1})$. In practice

$$\forall i \in [\![W - w + 1]\!], (f * h)_i = \sum_{j=1}^{w} f_{i-1+j} h_j = \sum_{n=i}^{i-1+w} f_n h_{n-i+1} \tag{1}$$

Same as a matrix multiplication with a $(W - w + 1) \times W$ **Toeplitz matrix**

$$f * h = \begin{pmatrix} h_1 & h_2 & \cdots & h_w & 0 & 0 & \cdots & 0 \\ 0 & h_1 & h_2 & \cdots & h_w & 0 & \cdots & 0 \\ 0 & 0 & h_1 & h_2 & \cdots & h_w & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & h_1 & h_2 & \cdots & h_w \end{pmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_W \end{bmatrix} \tag{2}$$

Convolution as matrix multiplication

Let $f = (f_1, \cdots, f_W), h = (h_1, \cdots, h_{w-1})$. In practice

$$\forall i \in [\![W - w + 1]\!], (f * h)_i = \sum_{j=1}^{w} f_{i-1+j} h_j = \sum_{n=i}^{i-1+w} f_n h_{n-i+1} \tag{1}$$

Same as a matrix multiplication with a $(W - w + 1) \times W$ **Toeplitz matrix**

$$f * h = \begin{pmatrix} h_1 & h_2 & \cdots & h_w & 0 & 0 & \cdots & 0 \\ 0 & h_1 & h_2 & \cdots & h_w & 0 & \cdots & 0 \\ 0 & 0 & h_1 & h_2 & \cdots & h_w & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & h_1 & h_2 & \cdots & h_w \end{pmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_W \end{bmatrix} \tag{2}$$

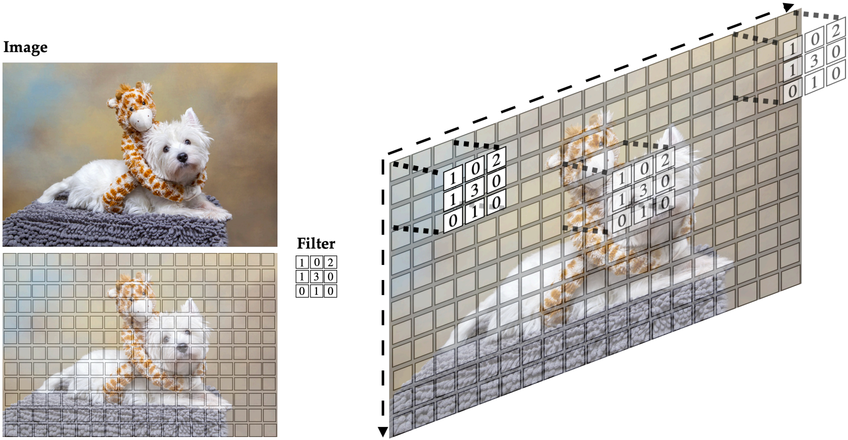
Convolution is just a specific linear layer

$\mathrm{Conv}(\mathbf{x}) = \mathbf{W}\mathbf{x}$ with **shared weights** in the $\mathbf{W}$ matrix.

Convolutional neural networks

Discrete convolutions **not** in 1D

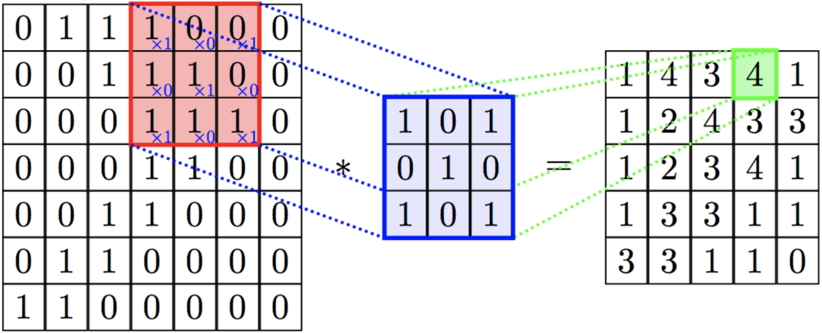
See also `https://github.com/vdumoulin/conv_arithmetic`.



Convolutional neural networks

Discrete convolutions **not** in 1D

See also `https://github.com/vdumoulin/conv_arithmetic`.



Convolutional neural networks

Discrete convolutions **not** in 1D

See also `https://github.com/vdumoulin/conv_arithmetic`.

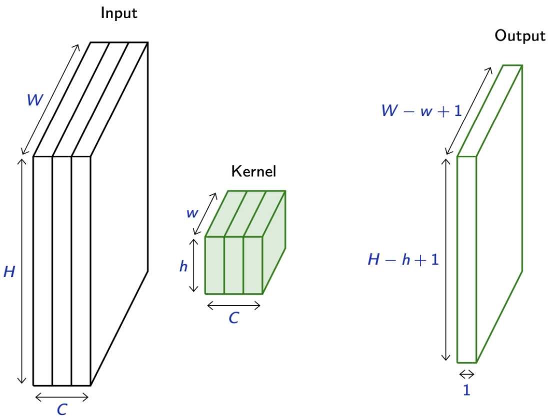


Figure: From Franccois Fleuret `https://fleuret.org/dlc/`

Convolutional neural networks

Discrete convolutions **not** in 1D

See also `https://github.com/vdumoulin/conv_arithmetic`.

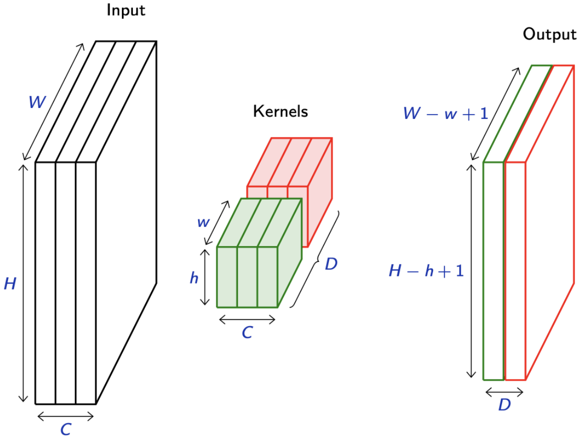


Figure: From Franccois Fleuret `https://fleuret.org/dlc/`

Convolutional neural networks

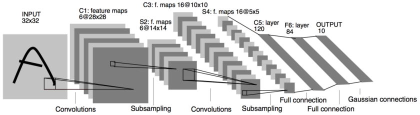


Figure: LeNet from LeCun et al. 1998

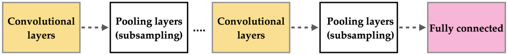


Figure: Schematic view

Principle and intuition (Zeiler and Fergus 2014)

- Define multiple convolutions, **learn the corresponding filter weights**.
- Recognize local patterns in images.
- Find intermediate features that are "general" and "adaptive" due to the translation equivariance bias `https://fabianfuchsml.github.io/equivariance1of2/`.
- Revealing local features that are shared across the data domain.

Conclusion

- Deep learning: in almost everything when there are images.
- Very versatile: learn complex functions.
- Prior also helps ! (translation equivariance).
- Side note: still struggles on tabular data (Grinsztajn, Oyallon, and Varoquaux 2022).

Conclusion

- Deep learning: in almost everything when there are images.
- Very versatile: learn complex functions.
- Prior also helps ! (translation equivariance).
- Side note: still struggles on tabular data (Grinsztajn, Oyallon, and Varoquaux 2022).

Graph neural networks ?

- How do we extend neural networks to graphs?
- Careful to node ordering: must be invariant to relabelling of the nodes (graph isomorphism).