Mathematics of Deep Learning
Session 1 - Monday January 25th

Practical information

- **Lecturers**
  - Rémi Gribonval
  - Aurélien Garivier
  - Nelly Pustelnik

- **Online course**
  - Monday 13:30-16:40
- **Language** : english
- **Attendance** : online

Practical information

- **Web page maintained by A. Garivier**
  https://perso.ens-lyon.fr/aurelien.garivier/www.math.univ-toulouse.fr/_agarivie/deepLearning.html
  - course specific information
  - links, bibliographical references …

- **Evaluation**
  - Principle
    - Homework, in-class exercises & final exam: 50%
    - Project (review and oral presentation of an article): 50%
  - More details in due time

Mathematics of deep learning

- **Approximation theory:** « expressivity »
  - Given f, how small can be $\inf_\theta \|f_\theta - f\|$
    - Universal Approximation Theorem and more (Rémi Gribonval)
    - Power of Depth (Aurélien Garivier)
- **Learning from samples** $y_i \approx f(x_i)$

$$\texttt{Landscape}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \texttt{Loss}(y_i, f_\theta(x_i))$$

  - Optimization algorithms ?
    - Smooth & nonsmooth optimization (Nelly Pustelnik)
    - SGD, Adam (Nelly Pustelnik)
  - Convergence ?
    - Landscape of cost functions, convexity (Nelly Pustelnik, Rémi Gribonval)
  - Statistical relevance ?
    - VC dimension, double descent (A. Garivier)

Agenda

- **Generalities on feedforward neural networks**
- Focus on ReLU networks
- Calculus with networks
- Role of depth
- Expressivity of (sparsely connected) deep networks

Feedforward neural networks

- **Neuron**
  - input vector $x \in \mathbb{R}^N$
  - parameters
    - weight vector $a \in \mathbb{R}^N$
    - bias $b \in \mathbb{R}$
    - nonlinearity $\varrho : \mathbb{R} \to \mathbb{R}$
      - aka activation function
  - scalar output

$$y = \varrho(\langle x, a \rangle + b) \in \mathbb{R}$$

Feedforward neural networks

Neuron

- input vector $x \in \mathbb{R}^N$
- parameters
  - weight vector $a \in \mathbb{R}^N$
  - bias $b \in \mathbb{R}$
  - nonlinearity $\varrho : \mathbb{R} \to \mathbb{R}$
    - aka activation function

- scalar output

$$y = \varrho(\langle x, a \rangle + b) \in \mathbb{R}$$

Layer

- weight matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$
- bias vector $\mathbf{b} \in \mathbb{R}^M$
- entrywise activation $\varrho : \mathbb{R}^M \to \mathbb{R}^M$

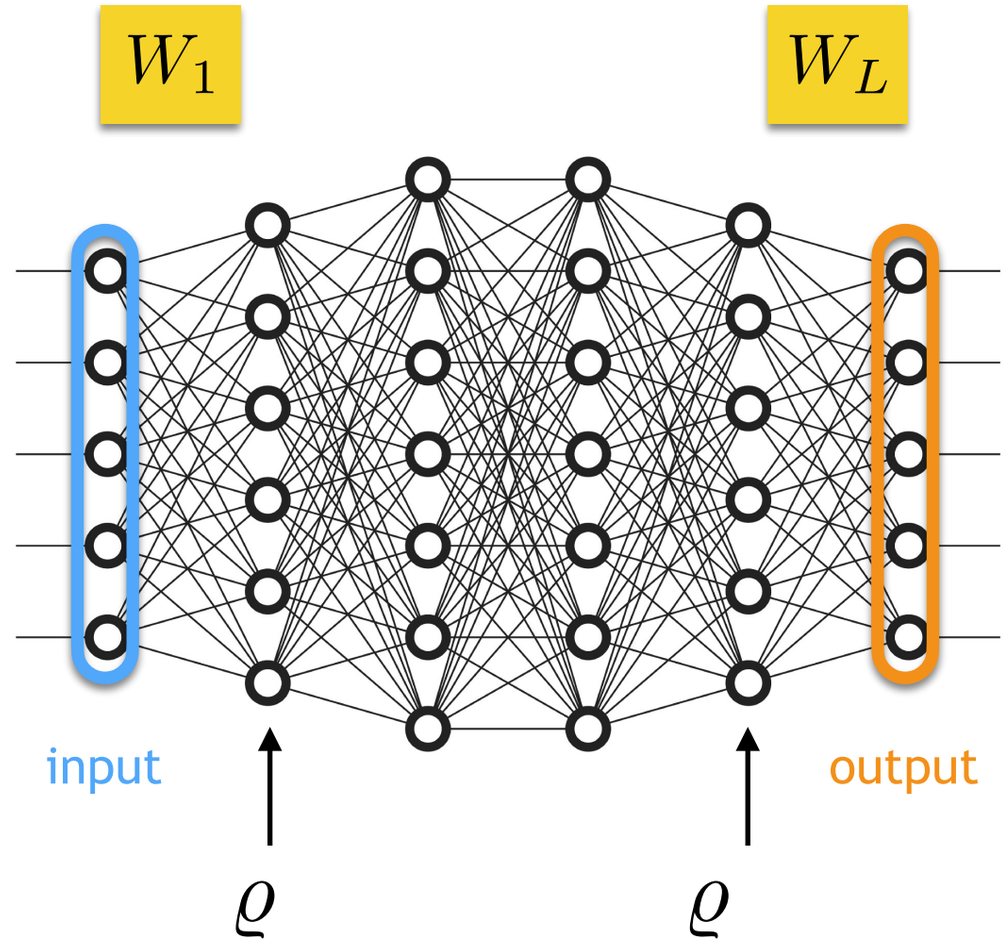
- vector output

$$y = \varrho(\underbrace{\mathbf{A}x + \mathbf{b}}_{W(x)}) \in \mathbb{R}^M$$

$W(x)$ affine transform

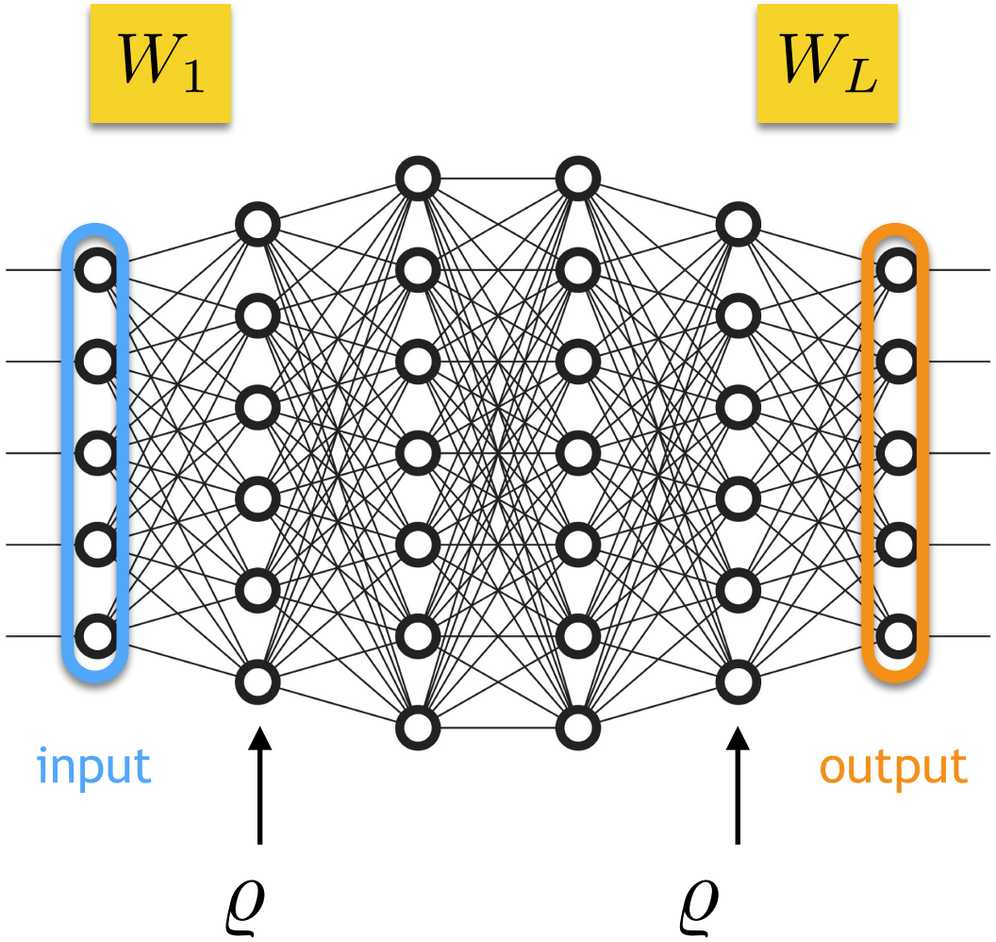
Feedforward neural networks

- **Feedforward network**
  - vector input $x \in \mathbb{R}^d$
  - parameters
    - L affine (“linear”) layers W_ℓ
    - L-1 (hidden) nonlinear layers
  - vector output $y \in \mathbb{R}^k$



Feedforward neural networks

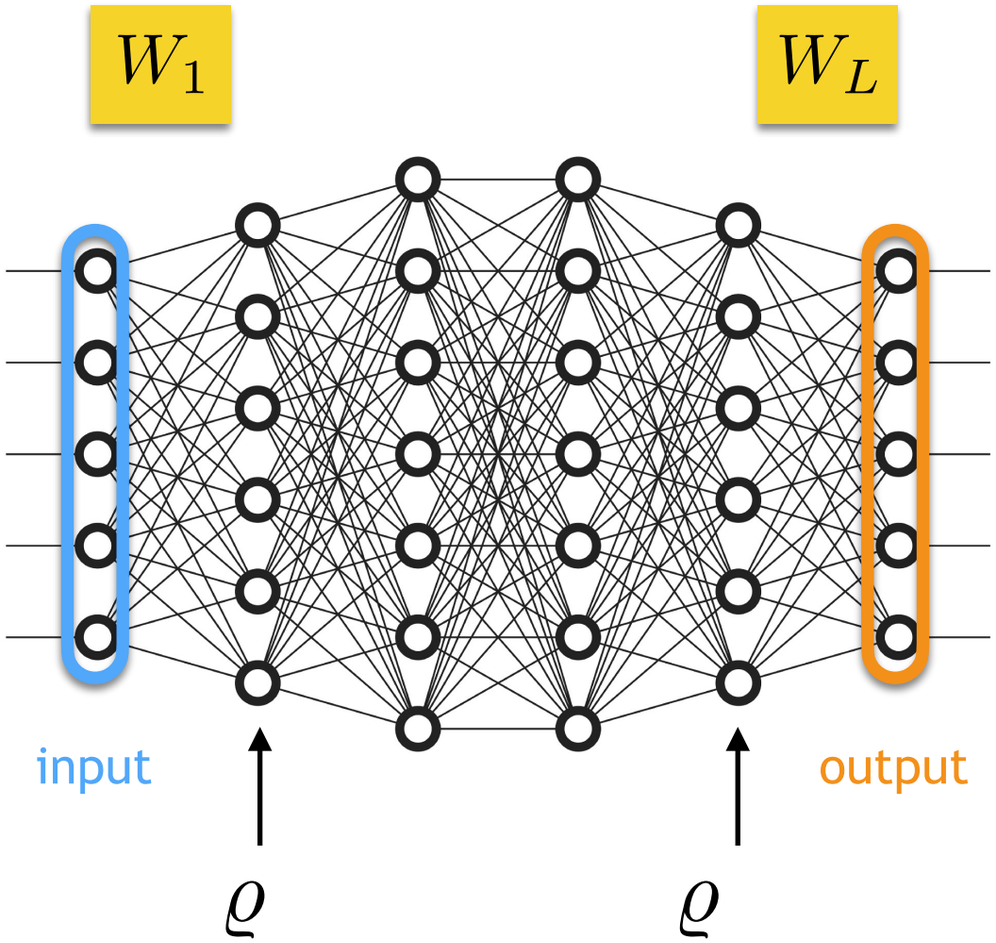
- **Feedforward network**
  - vector input $x \in \mathbb{R}^d$
  - parameters
    - L affine (“linear”) layers W_ℓ
    - L-1 (hidden) nonlinear layers
  - vector output $y \in \mathbb{R}^k$
  - description $\theta = (W_\ell)_{\ell=1}^L$
  - ***realization*** $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$



$$f_\theta = W_L \circ \varrho \circ W_{L-1} \circ \cdots \circ \varrho \circ W_1$$

Feedforward neural networks

- **Feedforward network**
  - vector input $x \in \mathbb{R}^d$
  - parameters
    - L affine ("linear") layers W_ℓ
    - L-1 (hidden) nonlinear layers
  - vector output $y \in \mathbb{R}^k$
  - description $\theta = (W_\ell)_{\ell=1}^L$
  - ***realization*** $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$



$$f_\theta = W_L \circ \varrho \circ W_{L-1} \circ \cdots \circ \varrho \circ W_1$$

- other ingredients: max-pooling, skip connections, conv … NOT IN THIS COURSE

Studying the « expressivity » of DNNs

DNN = rich architecture to implement functions

- $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ parameterized by θ (weights & biases)

Trained networks

- e.g. goal = regression

$$f_{\hat{\theta}}(x) \approx \mathbb{E}(Z|X=x)$$

- $\hat{\theta}$ typically found using stochastic gradient descent:
  See course of Nelly Pustelnik

Designed networks

- e.g. goal = solve LASSO

$$f_{\hat{\theta}}(x) \approx \arg\min_\alpha \tfrac{1}{2}\|x - \mathbf{A}\alpha\|^2 + \lambda\|\alpha\|_1$$

- typically proximal iterations
- learned variant LISTA

Studying the « expressivity » of DNNs

DNN = rich architecture to implement functions

- $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ parameterized by θ (weights & biases)

Trained networks

- e.g. goal = regression

$$f_{\hat{\theta}}(x) \approx \mathbb{E}(Z|X=x)$$

- $\hat{\theta}$ typically found using stochastic gradient descent:
  See course of Nelly Pustelnik

Designed networks

- e.g. goal = solve LASSO

$$f_{\hat{\theta}}(x) \approx \arg\min_{\alpha} \tfrac{1}{2}\|x - \mathbf{A}\alpha\|^2 + \lambda\|\alpha\|_1$$

- typically proximal iterations
- learned variant LISTA

Best achievable error given a budget ?

- typical budget = #neurons or #connections

Role of “architecture” ?

- activation function(s), *aka* nonlinearity, e.g. ReLU
- depth, skip-connections …

Universal approximation property (UAP)

A celebrated result

- *One hidden layer* enough to approximate arbitrarily well any continuous function ***on any compact subset*** of $\mathbb{R}^d$, with any "sigmoid-like" activation

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, 1989.

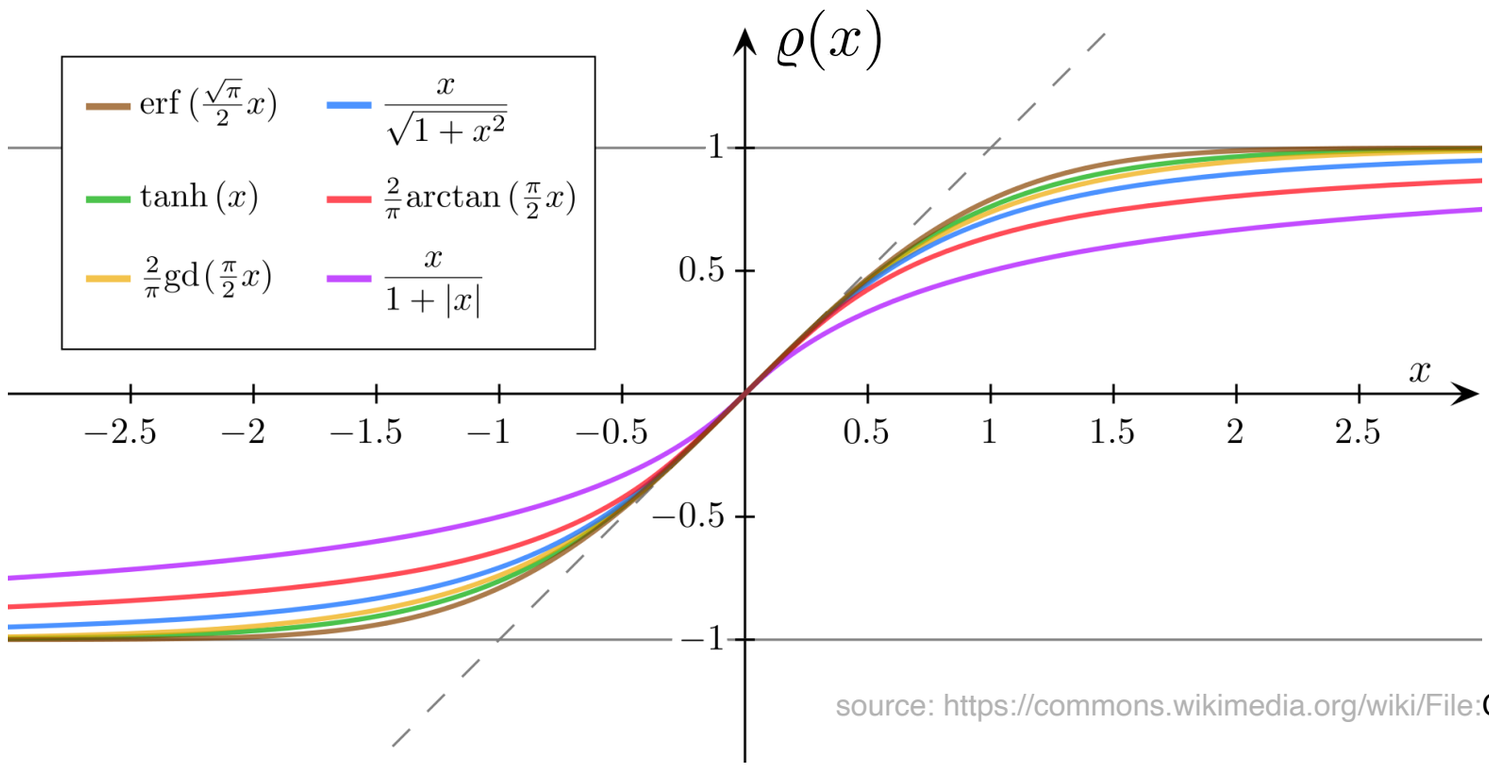
G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

Tradeoffs / Limitations?

- One hidden layer sufficient … with « enough » neurons
- Approximation *rates* wrt #neurons for "smooth" function
  - `Barron, DeVore, Mhaskar, and many more since the 1990s`
- Two hidden layers or more needed *on non-compact domains in dimension d>1*

Which activation function ?

- Validity of the UAP
  - **Initially**: for any « sigmoid-like » activation function



source: https://commons.wikimedia.org/wiki/File:Gjl-t(x).svg

Degenerate activation functions

- *Affine* activation function :
  - with any depth L, can only implement affine transforms
- *Polynomial* activation function of degree r
  - with bounded depth L, can only implement polynomials of bounded degree

Universal Approximation Theorem

- **Theorem**
  - Consider a continuous activation function $\varrho : \mathbb{R} \to \mathbb{R}$ and a compact subset $K \subset \mathbb{R}^d$
  - Let $\Sigma(K, \varrho)$ be the set of functions implemented by (finite) one-hidden layer networks with this activation function, i.e. which write

$$f(x) = \sum_{i=1}^{n} c_i \varrho(\langle a_i, x \rangle + b_i) \ \forall x \in K\} \qquad n \geq 1, a_1, \ldots, a_n \in \mathbb{R}^d, b, c \in \mathbb{R}^n$$

  - The following properties are equivalent
    - $\Sigma(K, \varrho)$ is dense in the set of continuous function $(C(K), \| \cdot \|_\infty)$
    - the activation function ϱ is not a polynomial

Proof sketch

- **Step 1**: reduce to univariate case
- **Step 2**: reduce to (univariate) case with $\varrho \in C^\infty(\mathbb{R})$
- **Step 3**: prove univariate case with $\varrho \in C^\infty(\mathbb{R})$

- Steps 1 and 3 rely on **Stone-Weierstrass**

Reminder

- **Stone-Weierstrass Theorem**
  - Let *K* be compact set and *C(K)* be the Banach algebra of continuous real-valued functions on *K*
  - A sub-algebra $\mathcal{A}$ of *C(K)* is **dense** in *C(K)* iff
    - it separates points, i.e., for each $x, y \in K$ there exists $f \in \mathcal{A}$ such that $f(x) \neq f(y)$
    - for each point $x \in K$, there exists $f \in \mathcal{A}$ such that $f(x) \neq 0$

- **Main example**: sub-algebra of polynomials

Proof

- **Step 1: reduction to univariate case**
  - The set $\mathcal{A} := \mathtt{span}\{g : x \mapsto \exp(\langle v, x \rangle), v \in \mathbb{R}^d\}$ is a sub-algebra of *C(K)* that vanishes nowhere and separates points
  - By Stone-Weierstrass $\mathtt{closure}(\mathcal{A}) = C(K)$
  - [details filled in on the board]

Proof

- **Step 2: reduce to (univariate) case with $\varrho \in C^\infty(\mathbb{R})$**
  - Consider $\psi \in C^\infty(\mathbb{R})$ with compact support
  - Then $\sigma := \varrho \star \psi$ belongs to $C^\infty(\mathbb{R})$
  - For any $a, b \in \mathbb{R}$ the function

$$f : x \mapsto \sigma(ax+b) = \int_{-\infty}^{+\infty} \varrho(ax+b-y)\psi(y)dy$$

belongs to the closure of $\Sigma(K, \varrho)$

  - As a result

$$\texttt{closure}(\Sigma(K,\sigma)) \subset \texttt{closure}(\Sigma(K,\varrho))$$

Proof

- **Step 3: prove univariate case with** $\varrho \in C^\infty(\mathbb{R})$
  - Fix an integer $k \geq 0$
  - Since $\varrho \in C^\infty(\mathbb{R})$ is not a polynomial there exists a scalar $b_k \in \mathbb{R}$ such that $\phi^{(k)}(b_k) \neq 0$
  - *[admitted] Using « Peano's representation » it can be shown that*

$$x \mapsto (\phi^{(k)}(b_k)/k!)x^k \in \texttt{closure}(\Sigma(K, \varrho))$$

  - As a result $\texttt{closure}(\Sigma(K, \varrho))$ contains all polynomials
  - Conclude by Stone-Weierstrass (again!)

« Ultimate » version of the UAP

- **Definition:** is *non-degenerate* if
  - it is Borel measurable
  - it is locally bounded
  - it is continuous except possibly on a closed set of measure zero
  - it does not coincide a.e. with a polynomial
  - ***Remark***: a *continuous* is non-degenerate iff it is not a polynomial
- **Theorem:**
  - if $\varrho : \mathbb{R} \to \mathbb{R}$ is non-degenerate then the UAP holds

M. Leshno, V. Ya. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Netw.*, 6(6):861–867, 1993.

How many neurons are needed ?

Theorem

V. Maiorov and A. Pinkus. Lower bounds for approximation by MLP neural networks. *Neurocomputing*, 25(1):81–91, 1999.

- there is a real *analytic* function $\varrho : \mathbb{R} \to \mathbb{R}$ s.t.
  - it is strictly increasing and sigmoidal

$$\lim_{x \to -\infty} \varrho(x) = 0; \quad \lim_{x \to +\infty} \varrho(x) = 1$$

  - with L=3 (two hidden layers), *any continuous function f* on the unit cube can be approximated *arbitrarily well* in the uniform norm with $3d^2(6d+3)$ neurons and $21d^2 + 15d + 3$ connections.

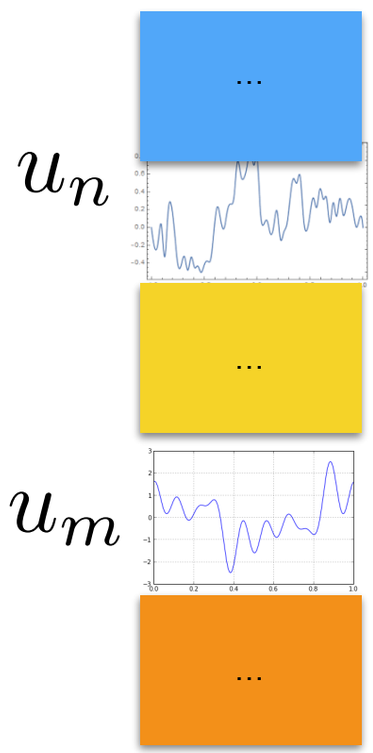
Pathological case: finitely many neurons !!!

- `optimizing the network weights is not tractable`

Main idea (scalar case)

- Without loss of generality, f is bounded by one
- The unit ball of C([0,1]) in the sup norm is separable
  - there exists a dense sequence of functions

$\{u_n(x)\}_{n\in\mathbb{Z}}$

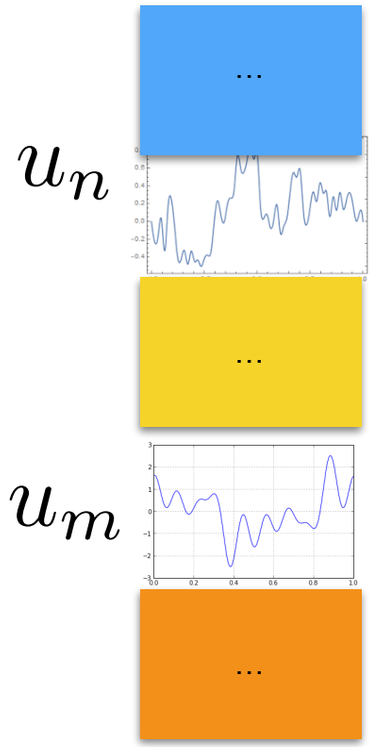


- Massaging this construction to get an analytic sigmoidal function, and extend this to higher dimension is more involved (uses Kolmogorov's superposition thm)

Main idea (scalar case)

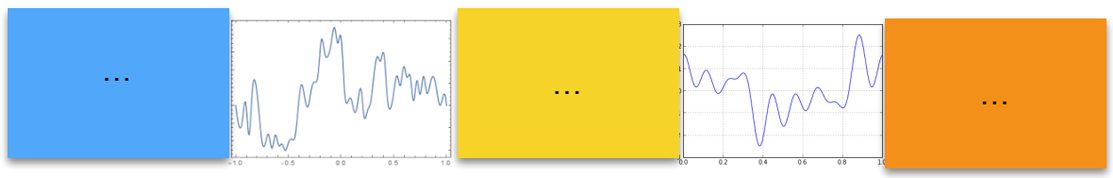
- Without loss of generality, f is bounded by one
- The unit ball of C([0,1]) in the sup norm is separable
  - there exists a dense sequence of functions

$\{u_n(x)\}_{n\in\mathbb{Z}}$



  - activation function built by concatenation

$\varrho(x) =$

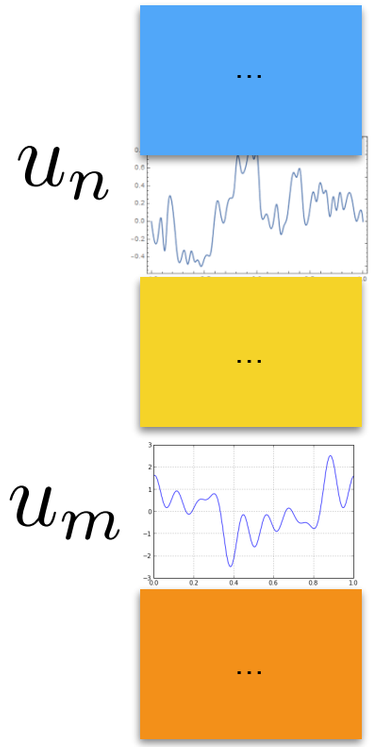


  - Massaging this construction to get an analytic sigmoidal function, and extend this to higher dimension is more involved (uses Kolmogorov's superposition thm)

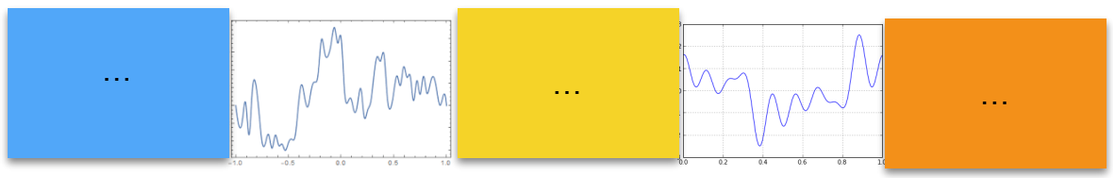
Main idea (scalar case)

- Without loss of generality, f is bounded by one
- The unit ball of C([0,1]) in the sup norm is separable
  - there exists a dense sequence of functions

$\{u_n(x)\}_{n\in\mathbb{Z}}$



  - activation function built by concatenation

$\varrho(x) =$ 

  - by construction there is a sequence of integers n_j such that

$$\lim_{j\to\infty} \|f - \varrho(\cdot - n_j)\|_\infty = \lim_{j\to\infty} \|f - u_{n_j}\|_\infty = 0$$

- Massaging this construction to get an analytic sigmoidal function, and extend this to higher dimension is more involved (uses Kolmogorov's superposition thm)

Agenda

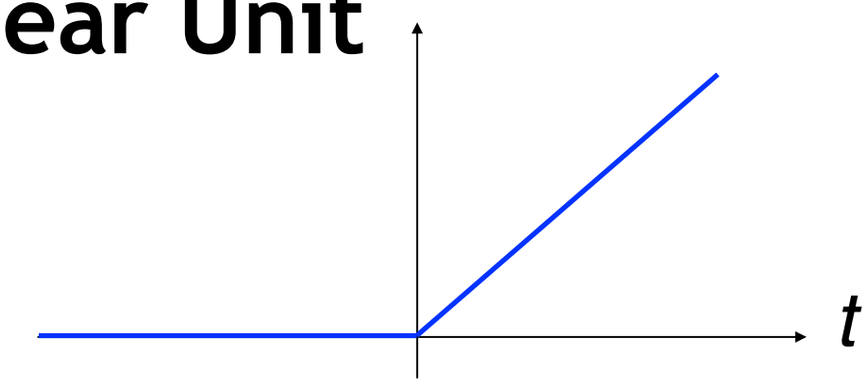
- Generalities on feedforward neural networks
- **Focus on ReLU networks**
- Calculus with networks
- Role of depth
- Expressivity of (sparsely connected) deep networks

The ReLU activation function

Definition: the Rectified Linear Unit

- **Definition**

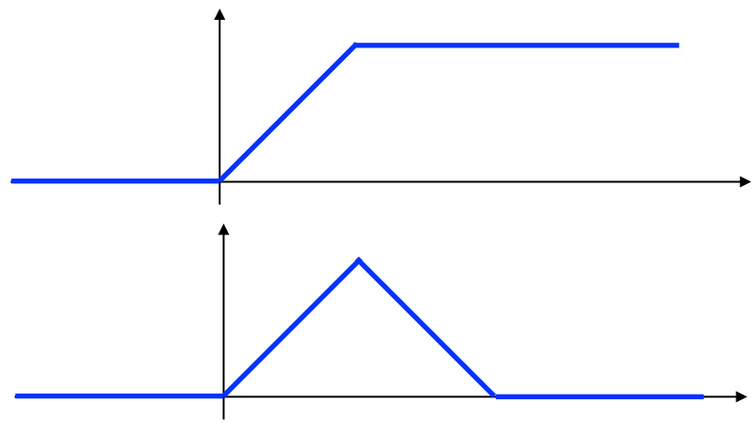
$$\varrho(t) = \texttt{ReLU}(t) = \max(t, 0) = t_+$$



- **Why** ? popular in practice for computational reasons (~ avoids vanishing / exploding gradients)

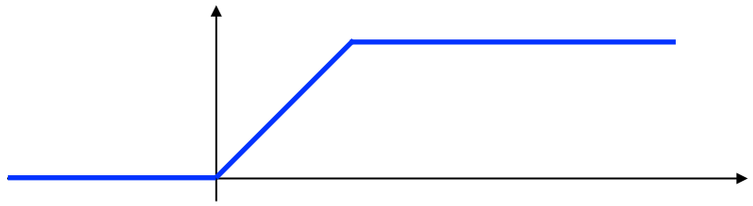
See course of Nelly Pustelnik

- **EXERCISE**: how to implement
  - « sigmoid-like » function (L=2, one hidden layer)
  - hat function (L=2, one hidden layer)
  - hat function (L=3, two hidden layers)
  - absolute value, soft-thresholding …

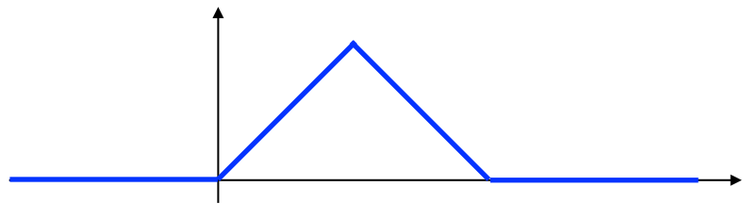


EXERCICE: univariate functions …

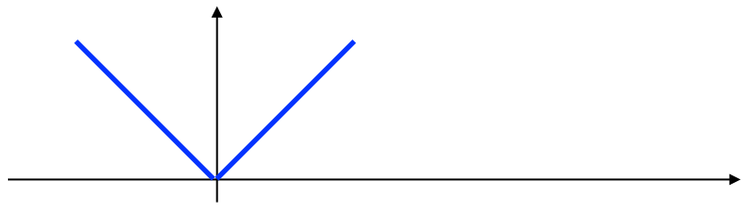
- « sigmoid-like » function
  - L=2, one hidden layer



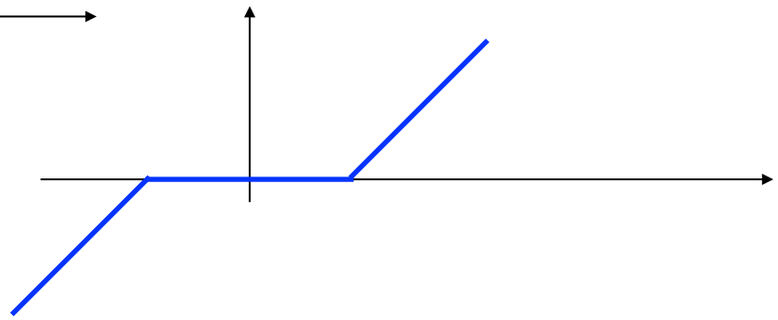
- hat function
  - L=2, one hidden layer
  - L=3, two hidden layers



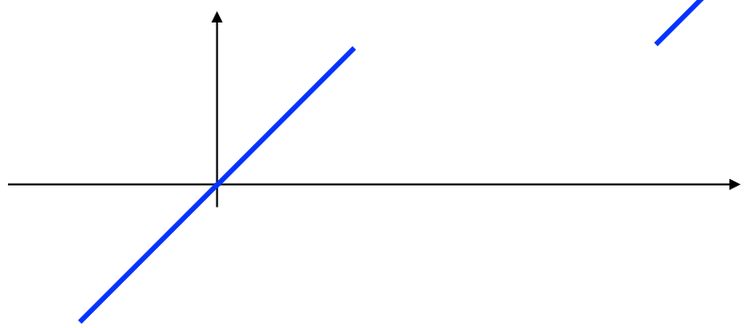
- absolute value
  - L=2, one hidden layer



- soft-thresholding
  - L=2, one hidden layer



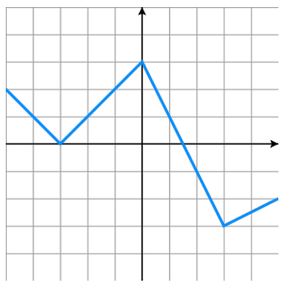
- identity



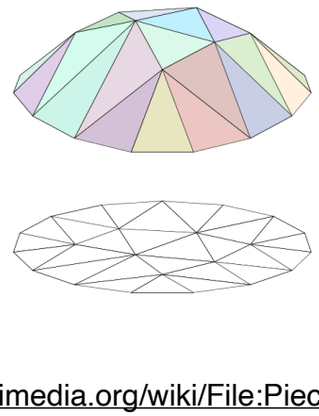
Realizations of ReLU-networks

Property 1

- any realization of a ReLU-network is continuous and piecewise (affine) linear
  - **d=1**



  - **d>1**

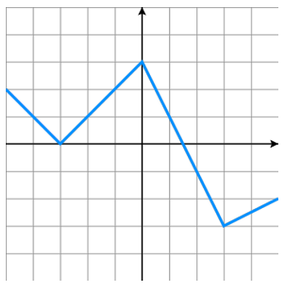


source: https://commons.wikimedia.org/wiki/File:Piecewise_linear_function2D.svg
(domaine public)

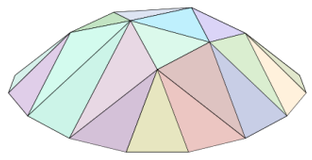
Realizations of ReLU-networks

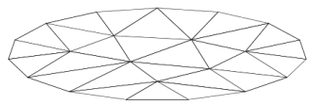
Property 1

- any realization of a ReLU-network is continuous and piecewise (affine) linear
  - **d=1**



  - **d>1**





source: https://commons.wikimedia.org/wiki/File:Piecewise_linear_function2D.svg (domaine public)

Converse?

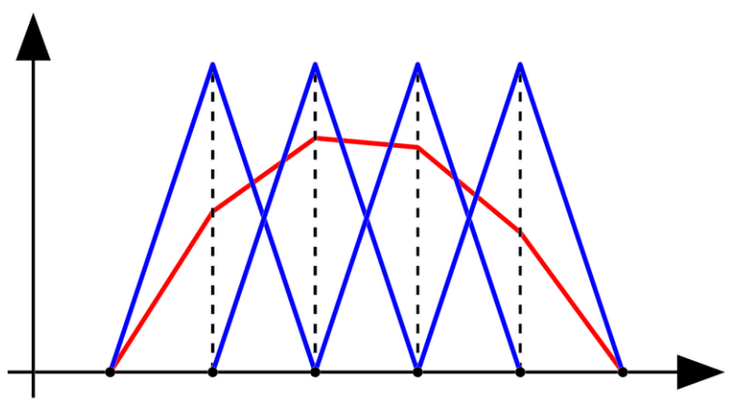
✓ **d=1:** ***any*** piecewise linear function is a realization of a ReLU-network **with one hidden layer**

★ **d>1:** no longer true

- One hidden layer: realization not compactly supported, ***not even integrable*** (unless it is zero)
- Need **at least two hidden layers** to be integrable

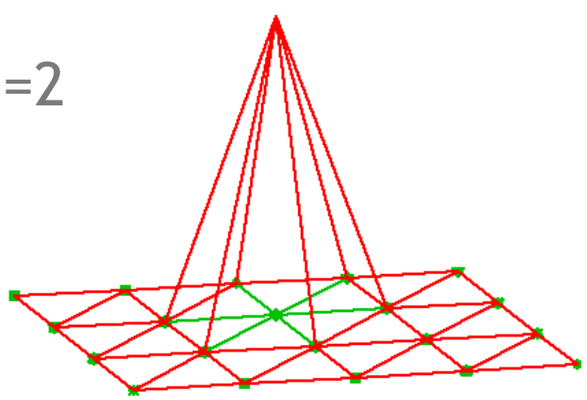
ReLU-networks and finite elements

- **FEM basis functions**
  - are continuous & piecewise affine
    - d=1



source: https://fr.wikipedia.org/wiki/Fichier:Finite_element_method_1D_illustration2.png
(domaine public)

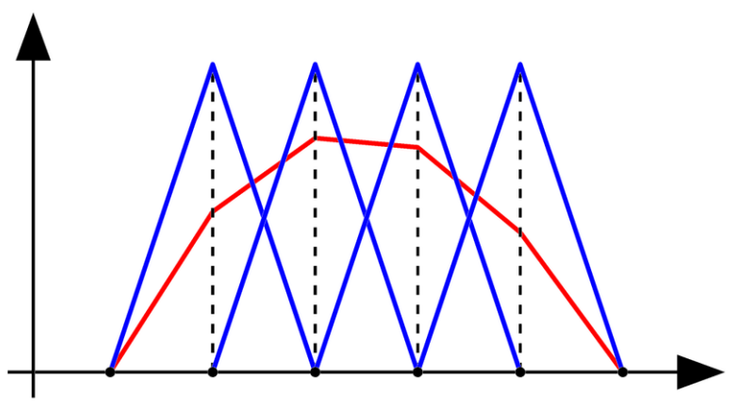
    - d=2



source: http://hplgit.github.io/INF5620/doc/pub/sphinx-fem/._main_fem009.html

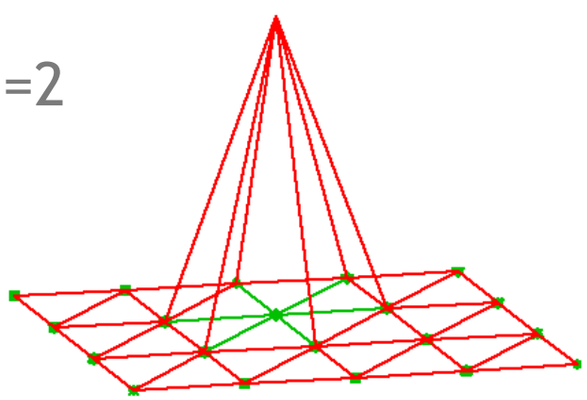
ReLU-networks and finite elements

- **FEM basis functions**
  - are continuous & piecewise affine
    - d=1



source: https://fr.wikipedia.org/wiki/Fichier:Finite_element_method_1D_illustration2.png (domaine public)

    - d=2



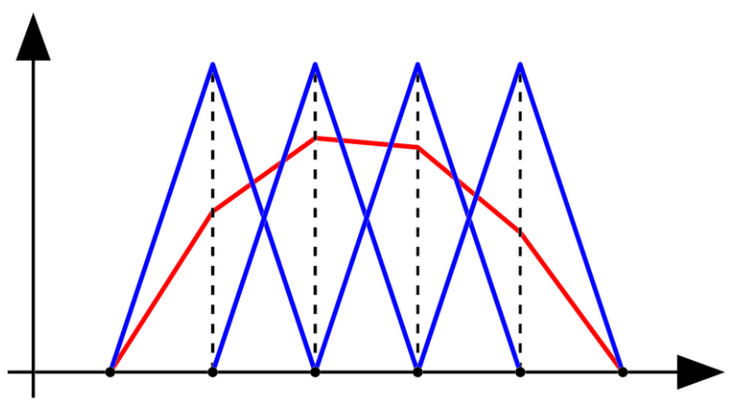
source: http://hplgit.github.io/INF5620/doc/pub/sphinx-fem/._main_fem009.html

  - can be implemented as a ReLU-network ..

R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, “Understanding Deep Neural Networks with Rectified Linear Units.,” ICLR 2018.
J. He, L. Li, J. Xu, C. Zheng, “Relu Deep Neural Networks and Linear Finite Elements,” *JCM*, vol. 38, no. 3, pp. 502–527, Feb. 2020.

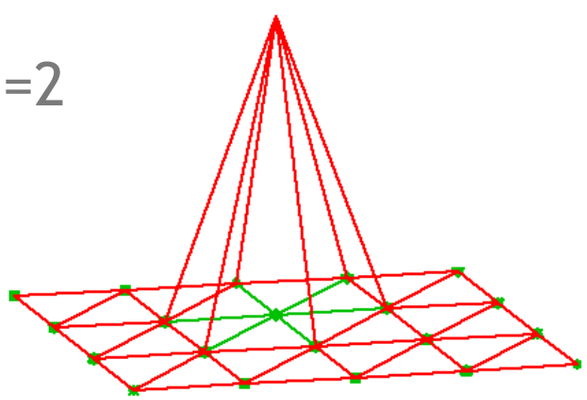
ReLU-networks and finite elements

- **FEM basis functions**
  - are continuous & piecewise affine
    - d=1



source: https://fr.wikipedia.org/wiki/Fichier:Finite_element_method_1D_illustration2.png (domaine public)

    - d=2



source: http://hplgit.github.io/INF5620/doc/pub/sphinx-fem/._main_fem009.html

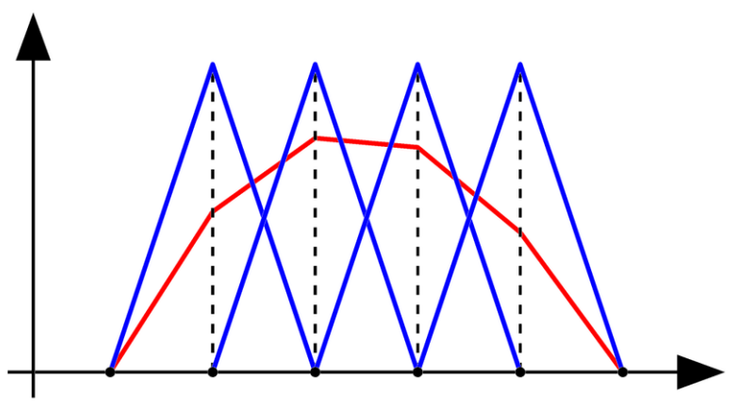
  - can be implemented as a ReLU-network ..

R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, “Understanding Deep Neural Networks with Rectified Linear Units.,” ICLR 2018.
J. He, L. Li, J. Xu, C. Zheng, “Relu Deep Neural Networks and Linear Finite Elements,” *JCM*, vol. 38, no. 3, pp. 502–527, Feb. 2020.

    - In dimension *d=1*:
      - $L \geq 2$ (one hidden layer) **necessary & sufficient** (L=2, n=3 neurons are enough)

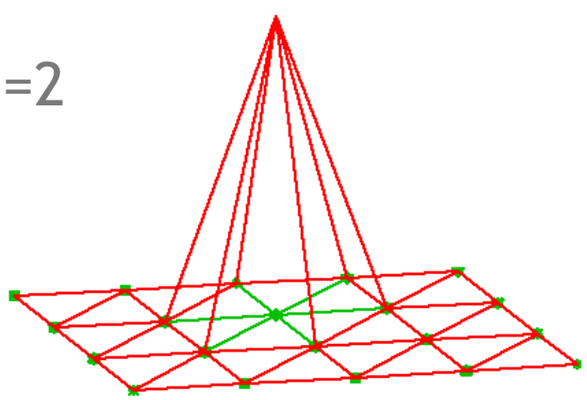
ReLU-networks and finite elements

- **FEM basis functions**
  - are continuous & piecewise affine
    - d=1



source: https://fr.wikipedia.org/wiki/Fichier:Finite_element_method_1D_illustration2.png (domaine public)

    - d=2



source: http://hplgit.github.io/INF5620/doc/pub/sphinx-fem/._main_fem009.html

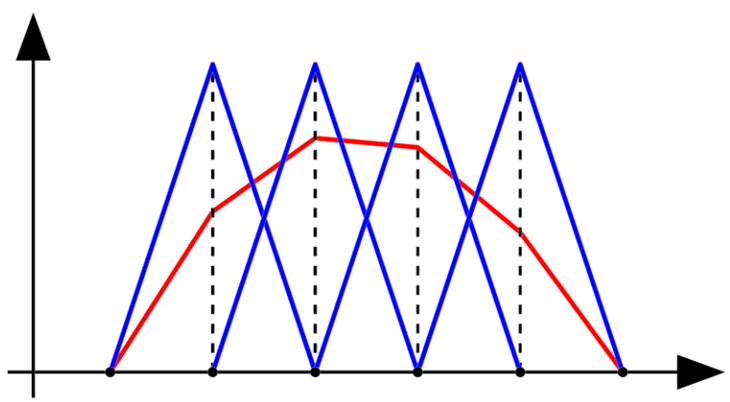
  - can be implemented as a ReLU-network ..

R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, “Understanding Deep Neural Networks with Rectified Linear Units.,” ICLR 2018.
J. He, L. Li, J. Xu, C. Zheng, “Relu Deep Neural Networks and Linear Finite Elements,” *JCM*, vol. 38, no. 3, pp. 502–527, Feb. 2020.

    - In dimension *d=1*:
      - $L \geq 2$ (one hidden layer) **necessary & sufficient** (L=2, n=3 neurons are enough)
    - In dimension *d=2 or d=3*:
      - $L \geq 3$ (two hidden layers) **necessary & sufficient**

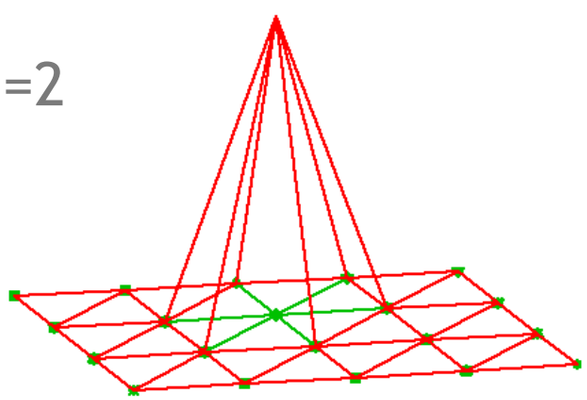
ReLU-networks and finite elements

- **FEM basis functions**
  - are continuous & piecewise affine
    - d=1



source: https://fr.wikipedia.org/wiki/Fichier:Finite_element_method_1D_illustration2.png
(domaine public)

    - d=2



source: http://hplgit.github.io/INF5620/doc/pub/sphinx-fem/._main_fem009.html

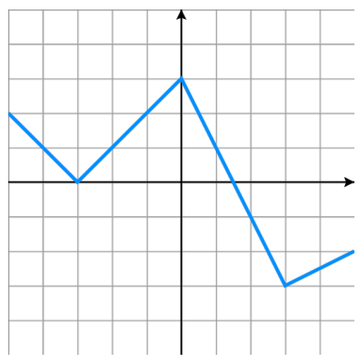
  - can be implemented as a ReLU-network ..

R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, “Understanding Deep Neural Networks with Rectified Linear Units.,” ICLR 2018.
J. He, L. Li, J. Xu, C. Zheng, “Relu Deep Neural Networks and Linear Finite Elements,” *JCM*, vol. 38, no. 3, pp. 502–527, Feb. 2020.

    - In dimension *d=1*:
      - $L \geq 2$ (one hidden layer) **necessary & sufficient** (L=2, n=3 neurons are enough)
    - In dimension *d=2 or d=3*:
      - $L \geq 3$ (two hidden layers) **necessary & sufficient**
    - In arbitrary dimension
      - $L \geq \lceil \log_2(d+1) \rceil + 1$ layers are **sufficient**

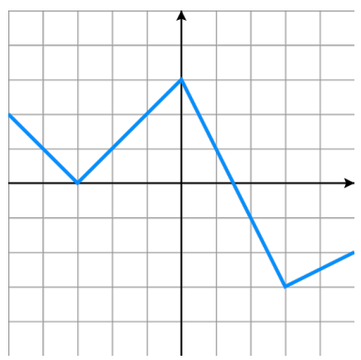
Number of pieces

- For ReLU networks of depth L
  - in dimension $d=1$
    - If #neurons = n then $\sharp \text{pieces} = \mathcal{O}(n^{L-1})$
    - If #connections = n then $\sharp \text{pieces} = \mathcal{O}(n^{\lfloor L/2 \rfloor})$
      - *even when counting only connections with nonzero weights*
    - These bounds are sharp



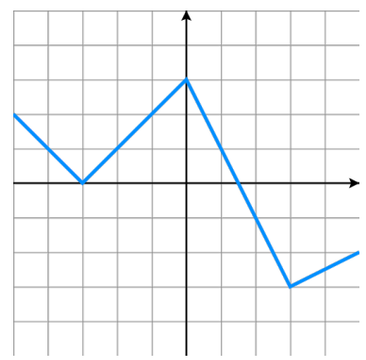
Number of pieces

- For ReLU networks of depth L
  - in dimension $d=1$
    - If #neurons = n then $\sharp \text{pieces} = \mathcal{O}(n^{L-1})$
    - If #connections = n then $\sharp \text{pieces} = \mathcal{O}(n^{\lfloor L/2 \rfloor})$
      - *even when counting only connections with nonzero weights*
    - These bounds are sharp
    - Proof: R. Gribonval, G. Kutyniok, M. Nielsen, and F. Voigtlaender, *Approximation Spaces of Deep Neural Networks*, arXiv e-prints, (2019), arXiv:1905.01208, p. arXiv:1905.01208.
      - using lemmas from Matus Telgarsky. Benefits of depth in neural networks. *Journal of Machine Learning Research*, 49(June):1517–1539, June 2016. 29th Conference on Learning Theory, COLT 2016 - New York, United States.



Number of pieces

- For ReLU networks of depth *L*
  - in dimension *d=1*
    - If #neurons = *n* then $\sharp \texttt{pieces} = \mathcal{O}(n^{L-1})$
    - If #connections = *n* then $\sharp \texttt{pieces} = \mathcal{O}(n^{\lfloor L/2 \rfloor})$
      - *even when counting only connections with nonzero weights*



    - These bounds are sharp
    - Proof: R. Gribonval, G. Kutyniok, M. Nielsen, and F. Voigtlaender, *Approximation Spaces of Deep Neural Networks*, arXiv e-prints, (2019), arXiv:1905.01208, p. arXiv:1905.01208.
      - using lemmas from Matus Telgarsky. Benefits of depth in neural networks. *Journal of Machine Learning Research*, 49(June):1517–1539, June 2016. 29th Conference on Learning Theory, COLT 2016 - New York, United States.
  - in dimension d>1: see

Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, 2014.