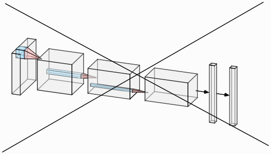
Deep Learning from a statistical viewpoint

Session 0: How, why and what is deep learning?

Tanguy Lefort
IMAG, Univ Montpellier, CNRS
LIRMM, Inria, Univ Montpellier, CNRS

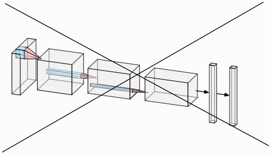
Joseph Salmon
IMAG, Univ Montpellier, CNRS
Institut Universitaire de France (IUF)



What **we are not going** to talk about:

- Zoology of neural networks
  (CNN, Transformers, etc.) : too many, more every other day!
- Very big neural networks (some models need thousands TPUv3-days to train [1] $\Rightarrow$ neither practical nor theoretically well understood).

[1] A. Dosovitskiy et al. (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *arXiv:2010.11929 [cs]*.



What **we are not going** to talk about:

- Zoology of neural networks
  (CNN, Transformers, etc.) : too many, more every other day!
- Very big neural networks (some models need thousands TPUv3-days to train [1] $\Rightarrow$ neither practical nor theoretically well understood).

What **we are going** to talk about:

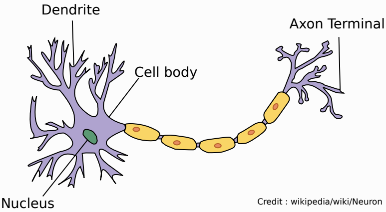
- What is a feed-forward NN?
- Why do we use activation functions?
- Optimization with gradient methods and vanishing gradient

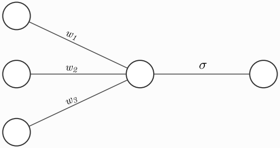
(1) A. Dosovitskiy et al. (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *arXiv:2010.11929 [cs]*.

Outline

The brain $\simeq$ 86 billions neurons each connected up to 10K others.

The core element: a neuron





- Input: $x \in \mathbb{R}^d$
- Hidden layer: $\langle w, x \rangle + b$
- Output: $\theta = (w, b)$
  $\hat{f}(x, \theta) = \sigma(\langle w, x \rangle) \in \mathbb{R}^{out}$
- Training:
  observations: $X = [x_1^\top, \ldots, x_n^\top]^\top$
  binary labels (± 1): $y = (y_1, \ldots, y_n)^\top$

Outline

```python
def perceptron(X, y, w, b, n_iter, rho=1):
    for k in range(n_iter):
        for i in range(len(y)):
            if (X[i].T @ w + b) * y[i] <= 0:  # error made
                w += rho * X[i] * y[i]
                b += rho * y[i]
    return w, b
```

Theorem: convergence [3]

The algorithm stops in a finite number of steps if the dataset (X, y) is separable, *i.e.*, if $\exists \gamma > 0, w^{\text{sep}} \in \mathbb{R}^d$ such that $\langle w^{\text{sep}}, x_i \rangle + b > \gamma$.

Rem: Roughly # steps $\propto \max\left(\|x_i\|\right)^2 \cdot \left\|w^{\text{sep}}\right\|^2 / \gamma^2$

[2] F. Rosenblatt (1958). *The perceptron: a theory of statistical separability in cognitive systems (Project Para)*. Cornell Aeronautical Laboratory

[3] A. B. Novikoff (1963). *On convergence proofs for perceptrons*. Tech. rep. STANFORD RESEARCH INST MENLO PARK CA

What is the perceptron?

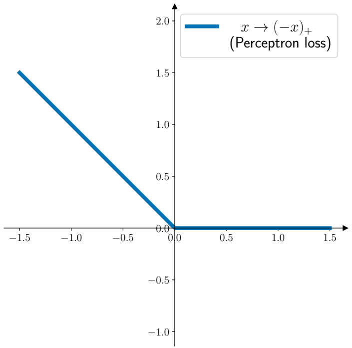
Modern answer: it is a linear binary classifier learnt with stochastic gradient descent (SGD) for the (perceptron) hinge loss and a fixed step size

(Perceptron) hinge loss:

$$\mathcal{L}(w, b) = \frac{1}{n}\sum_{i=1}^{n}\Big(- y_i \cdot (\langle w, x_i \rangle + b)\Big)_+$$

with

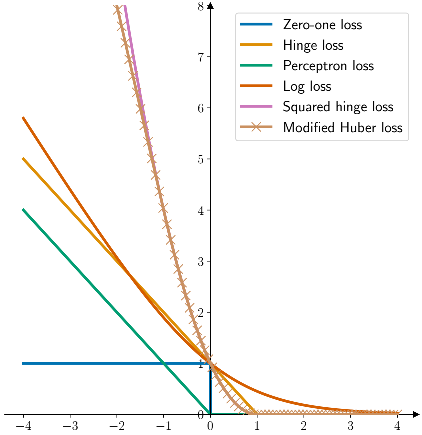
$$(x)_+ = ReLU(x) := \max(0, x)$$



Perceptron issues

- the Perceptron might never stop (*cf.* XOR problem)
- minimization of an 1-homogeneous function whose optimal value is trivial $(w = 0, b = 0)$
- SGD requires decreasing step size and does not converge for non-smooth case with fixed step size

Various standard loss



Credit:

`https://scikit-learn.org/dev/auto_examples/linear_model/plot_sgd_loss_functions.html`

Many linear/affine predictors (classifiers) can leverage an expression: $\sigma\left(\langle w, x\rangle + b\right)$

- σ the activation function
- w the weight (of the neuron)
- b the bias (of the neuron)

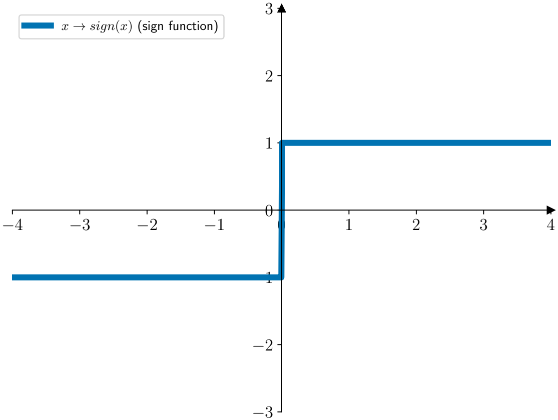
Many linear/affine predictors (classifiers) can leverage an expression: $\sigma\left(\langle w, x\rangle + b\right)$

- σ the activation function
- w the weight (of the neuron)
- b the bias (of the neuron)

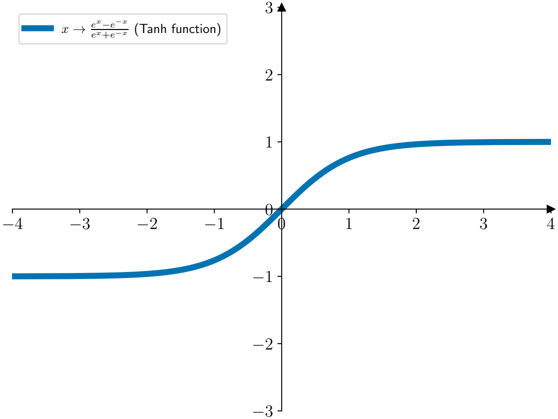
For binary classification, we can use as **activation** function σ

- the sign function: $x \mapsto \mathrm{sign}(x)$,
- the ReLU function: $x \mapsto (x)_+$,
- the tanh function: $x \mapsto \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- the sigmoïd: $x \mapsto \frac{1}{1+e^{-x}}$ (with in mind $\mathbb{P}(y_i = 1 | x_i) = \frac{1}{1+e^{-(\langle w, x_i\rangle + b)}}$)

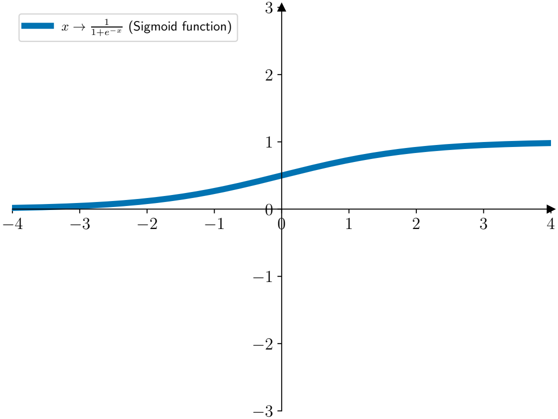
Activation function



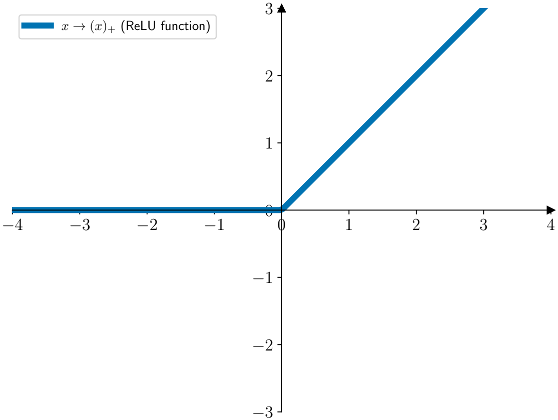
Activation function



Activation function



Activation function



$$\widehat{L}(\theta) = \frac{1}{n}\sum_i (y_i - f(x_i,\theta))^2 = \frac{1}{n}\sum_i \Big(y_i - \big(w^\top x_i + b\big)_+\Big)^2$$

Denoting $u = w^\top x_i + b, v = (y_i - u)$:

$$\frac{\partial \widehat{L}}{\partial w} = \frac{\partial \widehat{L}}{\partial v}\frac{\partial v}{\partial u}\frac{\partial u}{\partial w}$$
$$= -\frac{2}{n}\sum_i (y_i - \hat{y}_i) x_i^\top \mathbb{1}(u > 0)$$

$$\frac{\partial \hat{L}}{\partial b} = \frac{\partial \hat{L}}{\partial v}\frac{\partial v}{\partial u}\frac{\partial u}{\partial b}$$
$$= -\frac{2}{n}\sum_i (y_i - \hat{y}_i)\mathbb{1}(u > 0)$$

Using AutoGrad [4] :

```
model = nn.Sequential(
    nn.Linear(2, 1), # input 2D, output 1D
    nn.ReLU()  # ReLU activation
)
loss = nn.MSELoss()  # cost function
y_pred = model(x)  # prediction
val_loss = loss(y, y_pred)
val_loss.backward()  # backpropagation
model[0].weight.grad  # grad wrt w
model[0].bias.grad  # grad wrt b
```

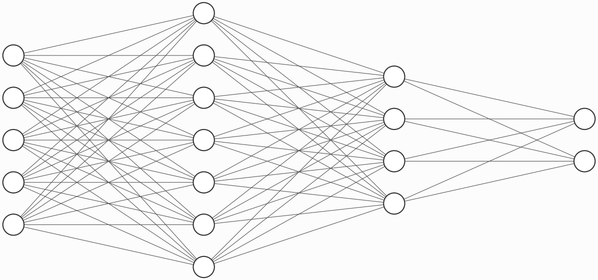
<u>Rem:</u>
Gradient tree can take a lot of memory

[4] A. Paszke et al. (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *arXiv:1912.01703 [cs, stat]*

Outline

Multi layer perceptron

- Feed-forward NN $\supset$ MLP (Multi-Layer Perceptron)



Idea: "There is strength in numbers"

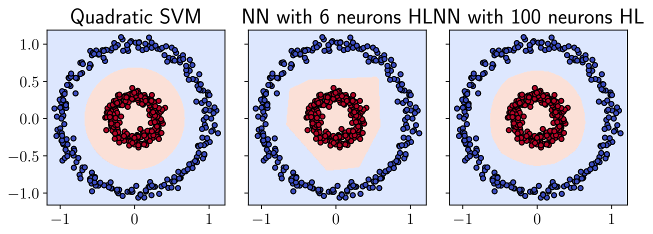
- each hidden layer : perceptrons = linear transformations + activations
- can handle multiclass cases (*e.g.*, with softmax output activation)

SVM and MLP classification

1 hidden layer in $\mathbb{R}^2$

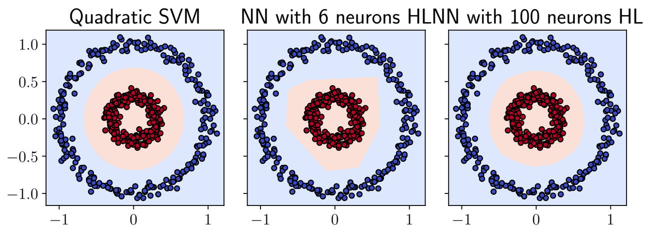
Decision boundary with quadratic kernel SVM in $\mathbb{R}^2$:

$$\left\{x \middle| \sum_i \alpha_i y_i k(x_i, x) + b = 0\right\}, \quad k(x, x') = (x^\top x' + c)^2$$



Decision boundary with quadratic kernel SVM in $\mathbb{R}^2$:

$$\left\{ x \middle| \sum_i \alpha_i y_i k(x_i, x) + b = 0 \right\}, \quad k(x, x') = (x^\top x' + c)^2$$



Decision boundary for MLP

- MLP compute probabilities and is scalable (for labels & features)
- # linear separations = # neurons in hidden layer
- Warning: overfitting with MLP is common

Outline

Outline

For K layers: matrices (weights) $W_1, \ldots, W_K$ and vectors (bias) $b_1, \ldots, b_K$

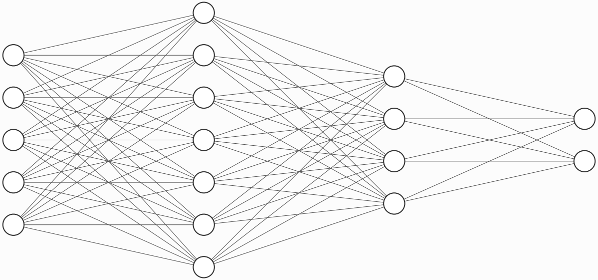
- **without** activations:

$$\begin{aligned} y &= W_K x_{K-1} + b_K = W_K(W_{K-1}x_{K-2} + b_{K-1}) + b_K \\ &= W_K W_{K-1} x_{K-2} + W_K b_{K-1} + b_K \\ &= \ldots \\ &= W_K W_{K-1} \ldots W_1 x_0 + \mathbf{c} = \mathbf{W}x + \mathbf{c} \quad \text{(affine!)} \end{aligned}$$

- **with** activations σ_K at layer k:

$$\begin{aligned} y &= \sigma_K(W_K z_{K-1} + b_K) = \sigma_K(z_K) \\ &= \sigma_K(W_K \sigma_{K-1}(W_{K-1} z_{K-2} + b_{K-1}) + b_K) \\ &= \sigma_K(W_K \sigma_{K-1}(W_{K-1} \sigma_{K-2}(\ldots) + b_{K-1}) + b_K) \quad \text{(not affine!)} \end{aligned}$$

Example



Here: $\hat{f} : \mathbb{R}^5 \to \mathbb{R}^2$

$$x \mapsto \sigma_3(W_3\sigma_2(W_2\sigma_1(W_1 x + b_1) + b_2) + b_3)$$

with:

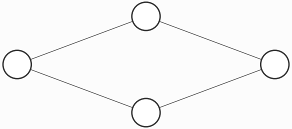
$$W_1 \in \mathbb{R}^{7\times 5}, \quad b_1 \in \mathbb{R}^7$$

$$W_2 \in \mathbb{R}^{4\times 7}, \quad b_2 \in \mathbb{R}^4$$

$$W_3 \in \mathbb{R}^{2\times 4}, \quad b_3 \in \mathbb{R}^2$$

Outline

Example of Code for pytorch

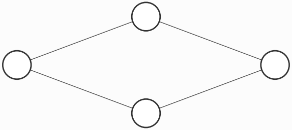


```python
class Two_Perceptron(torch.nn.Module):
    def __init__(self):
        super(Two_Perceptron, self).__init__()
        self.in_fc = nn.Linear(1, 2)
        self.fc = nn.Linear(2, 1)
        self.sig = nn.Sigmoid()

    def forward(self, x):
        out = self.in_fc(x)
        out = self.sig(out)
        out = self.fc(out)
        return out
```

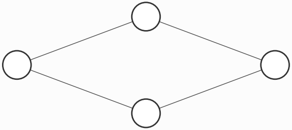
Simple 2 neurons-1 hidden layer and sigmoid activation:

$$x \mapsto w_3\sigma(w_1 x + b_1) + w_4\sigma(w_2 x + b_2)$$

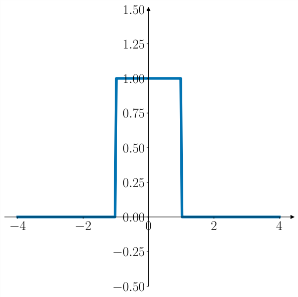


Simple 2 neurons-1 hidden layer and sigmoid activation:

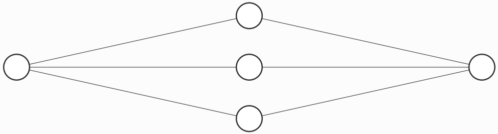
$$x \mapsto w_3\sigma(w_1x + b_1) + w_4\sigma(w_2x + b_2)$$



```
model = Two_Perceptron()
model.in_fc.weight.data =
            torch.tensor([[1000.], [1000.]])
model.in_fc.bias.data =
            torch.tensor([0., -1000.])
model.fc.weight.data =
            torch.tensor([[1., -1.]])
model.fc.bias.data = torch.tensor([0.])
```



Example of Code for pytorch

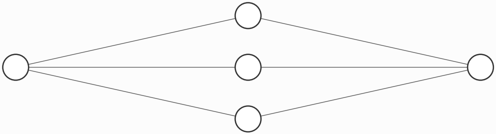


```python
class Three_Perceptron(torch.nn.Module):
    def __init__(self):
        super(Three_Perceptron, self).__init__()
        self.in_fc = nn.Linear(1, 3)
        self.fc = nn.Linear(3, 1)
        self.sig = nn.ReLU()

    def forward(self, x):
        out = self.in_fc(x)
        out = self.sig(out)
        out = self.fc(out)
        return out
```

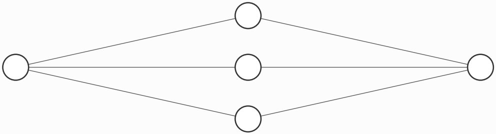
Simple 3 neurons-1 hidden layer and ReLu activation:

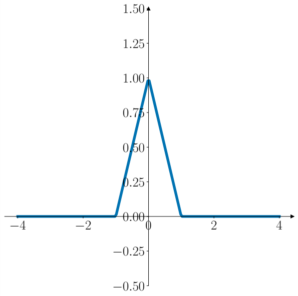
$$y = f(x, \theta) = \sigma(x + 1) - 2\sigma(x) + \sigma(x - 1)$$



Simple 3 neurons-1 hidden layer and ReLu activation:

$$y = f(x, \theta) = \sigma(x+1) - 2\sigma(x) + \sigma(x-1)$$






```
model = Three_Perceptron()
model.in_fc.weight.data =
            torch.tensor([[1.], [1.], [1.]])
model.in_fc.bias.data =
            torch.tensor([1, 0.,-1])
model.fc.weight.data =
            torch.tensor([[1., -2, 1.]])
model.fc.bias.data = torch.tensor([0.])
```

Outline

Notation: $\mathcal{C}([0,1]^d) := \{f : [0,1]^d \to \mathbb{R}, f \text{ continuous}\}$

Under the condition that σ is continuous with $\begin{cases} \lim_{t\to-\infty} \sigma(t) = 0, \\ \lim_{t\to+\infty} \sigma(t) = 1 \end{cases}$ then

Theorem (5)

For every function $g \in \mathcal{C}([0,1]^d)$ and $\epsilon > 0$ there exists a feedforward neural network f such that $\|g - f\|_\infty < \epsilon$, *i.e.*,

$$f(x) = \sum_{j=1}^{N} \alpha_j \sigma(\langle w_j, x\rangle + b_j)$$

for some integer N, some $w_j \in \mathbb{R}^d$ and $b_j \in \mathbb{R}$.

(5) G. Cybenko (1989). "Approximation by superpositions of a sigmoidal function". en. In: *Mathematics of Control, Signals and Systems* 2.4, pp. 303–314

(6) L. Devroye, L. Györfi, and G. Lugosi (1996). *A probabilistic theory of pattern recognition*. Vol. 31. Applications of Mathematics (New York). Springer-Verlag

(7) Z. Lu et al. (2017). *The Expressive Power of Neural Networks: A View from the Width*.

Notation: $\mathcal{C}([0,1]^d) := \{f : [0,1]^d \to \mathbb{R}, f \text{ continuous}\}$

Under the condition that σ is continuous with $\begin{cases} \lim_{t\to-\infty} \sigma(t) = 0, \\ \lim_{t\to+\infty} \sigma(t) = 1 \end{cases}$ then

Theorem [(5)]

For every function $g \in \mathcal{C}([0,1]^d)$ and $\epsilon > 0$ there exists a feedforward neural network f such that $\|g - f\|_\infty < \epsilon$, *i.e.*,

$$f(x) = \sum_{j=1}^{N} \alpha_j \sigma(\langle w_j, x\rangle + b_j)$$

for some integer N, some $w_j \in \mathbb{R}^d$ and $b_j \in \mathbb{R}$.

Rem: one can adapt the proof[(6)] to ReLU with the "triangle" approximation.
Rem: refined control on width/depth are possible[(7)]

(5) G. Cybenko (1989). "Approximation by superpositions of a sigmoidal function". en. In: *Mathematics of Control, Signals and Systems* 2.4, pp. 303–314

(6) L. Devroye, L. Györfi, and G. Lugosi (1996). *A probabilistic theory of pattern recognition*. Vol. 31. Applications of Mathematics (New York). Springer-Verlag

(7) Z. Lu et al. (2017). *The Expressive Power of Neural Networks: A View from the Width*.

What the theorem says and does not say:

- a MLP will approximate the function not learn (overfitting for example)

What the theorem says and does not say:

- a MLP will approximate the function not learn (overfitting for example)
- there always exists a large NN but might be exponentially large.

What the theorem says and does not say:

- a MLP will approximate the function not learn (overfitting for example)
- there always exists a large NN but might be exponentially large.

- a single hidden layer feedforward NN can represent any function BUT in practice, require a large single layer (extremelly time and memory consuming). Depth is a key element in NN.

- Widgets Runge and Gibbs phenomenon
- Show: universal approx video

Kolmogorov-Arnold theorem[3]**:** For any $g : [0,1]^d \to \mathbb{R}$ continuous there exist univariate continuous functions such that:

$$g(x_1, \ldots, x_d) = \sum_{q=}^{2d} h_q \left(\sum_{p=1}^{d} \psi_{p,q}(x_p) \right) \quad .$$

- *Kolmogorov's theorem is irrelevant*[4]: smoothness issue (learning) and h_q highly dependent of g (no parametric form).

[3] A. N. Kolmogorov (1956). "The representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables". In: *Doklady Akademii Nauk SSSR* 108.2, pp. 179–182.

[4] F. Girosi and T. Poggio (1989). "Representation properties of networks: Kolmogorov's theorem is irrelevant". In: *Neural Computation* 1.4, pp. 465–469.

[5] V. Kůrková (1991). "Kolmogorov's Theorem Is Relevant". In: *Neural Computation* 3.4, pp. 617–622.

[6] J. Schmidt-Hieber (2021). "The Kolmogorov–Arnold representation theorem revisited". In: *Neural Networks* 137, pp. 119–126.

Kolmogorov-Arnold theorem[3]**:** For any $g : [0,1]^d \to \mathbb{R}$ continuous there exist univariate continuous functions such that:

$$g(x_1, \ldots, x_d) = \sum_{q=}^{2d} h_q \left(\sum_{p=1}^{d} \psi_{p,q}(x_p) \right) \quad .$$

- *Kolmogorov's theorem is irrelevant*[4]: smoothness issue (learning) and h_q highly dependent of g (no parametric form).
- *Kolmogorov's theorem is relevant*[5]: construction of smoother functions and reduction of the problem.

---

(3) A. N. Kolmogorov (1956). “The representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables”. In: *Doklady Akademii Nauk SSSR* 108.2, pp. 179–182.

(4) F. Girosi and T. Poggio (1989). “Representation properties of networks: Kolmogorov's theorem is irrelevant”. In: *Neural Computation* 1.4, pp. 465–469.

(5) V. Kůrková (1991). “Kolmogorov's Theorem Is Relevant”. In: *Neural Computation* 3.4, pp. 617–622.

(6) J. Schmidt-Hieber (2021). “The Kolmogorov–Arnold representation theorem revisited”. In: *Neural Networks* 137, pp. 119–126.

UNIVERSAL THEOREM OF APPROXIMATION

IS IT SIMPLY KOLMOGOROV-ARNOLD THEOREM?

Kolmogorov-Arnold theorem [3]**:** For any $g : [0,1]^d \to \mathbb{R}$ continuous there exist univariate continuous functions such that:

$$g(x_1, \ldots, x_d) = \sum_{q=}^{2d} h_q \left(\sum_{p=1}^{d} \psi_{p,q}(x_p) \right) \quad .$$

- *Kolmogorov's theorem is irrelevant* [4]: smoothness issue (learning) and h_q highly dependent of g (no parametric form).
- *Kolmogorov's theorem is relevant* [5]: construction of smoother functions and reduction of the problem.
- *The Kolmogorov-Arnold representation theorem revisited* [6]: simplify hypothesis and approximate the inner function with deep ReLU NN.

---

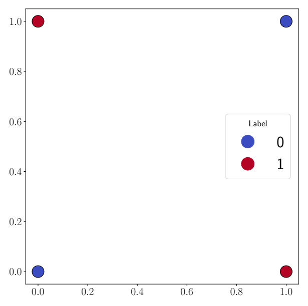
[3] A. N. Kolmogorov (1956). "The representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables". In: *Doklady Akademii Nauk SSSR* 108.2, pp. 179–182.

[4] F. Girosi and T. Poggio (1989). "Representation properties of networks: Kolmogorov's theorem is irrelevant". In: *Neural Computation* 1.4, pp. 465–469.

[5] V. Kůrková (1991). "Kolmogorov's Theorem Is Relevant". In: *Neural Computation* 3.4, pp. 617–622.

[6] J. Schmidt-Hieber (2021). "The Kolmogorov–Arnold representation theorem revisited". In: *Neural Networks* 137, pp. 119–126.

Outline



$$X = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \quad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$\widehat{L}(\theta) = \frac{1}{4} \sum_i (y_i - f(x_i, \theta))^2$$

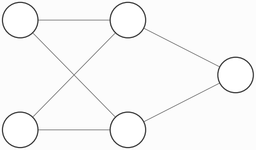
$$f(x_i, \theta) = \mathbb{1}(f(x) > 0.5)$$

Non separable case: the XOR problem

The XOR problem with this configuration can not be solved with a linear separator $x \mapsto w^\top x + b$. (To show: video xor perceptron)

1^{st} order conditions: $\hat{w} = (0,0)^\top$, and $\hat{b} = \frac{1}{2} \Longrightarrow f(x_i, \theta) = \frac{1}{2}$.

(7) M. L. Minsky and S. A. Papert (1969). *Perceptrons. An Introduction to Computational Geometry. 1969, Expanded.* Cambridge, MA: MIT Press

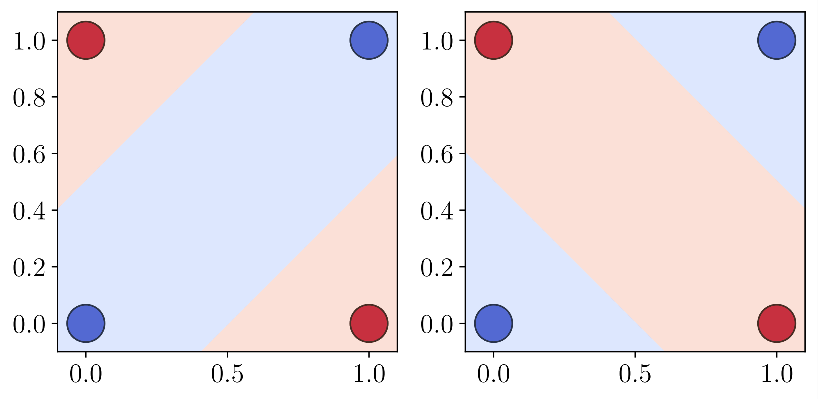


$$\widehat{L}(\theta) = \frac{1}{4}\sum_i (y_i - f(x_i,\theta))^2,$$

$$f(x_i,\theta) = \langle w_2, h\rangle + b_2,\ h = (W_1 x_i + b_1)_+$$

Multiple solutions give $\widehat{L}(\theta) = 0$

- if $b_1 = (0,0)$, and $b_2 = 0$:
  - $w_2 = (1,1)$ and $W_1 = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$
  - $w_2 = (1,1)$ and $W_1 = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$
  - for $D = \mathrm{diag}(d_1, d_2)$, $d_1, d_2 > 0$ taking $\widetilde{w}_2 = D^{-1}w_2$, $\widetilde{W}_1 = DW_1$ with w_2 and W_1 leading to a global minimum.
- if $b_1 = (1,-1)$ and $b_2 = 1$ and $W_1 = \begin{pmatrix} -1 & -1 \\ 1 & 1 \end{pmatrix}$ and $w_2 = (-1,-1)$
- ...



To show: video xor 1 HL 2 neurons

Outline

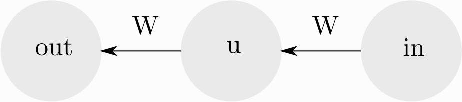
Learn a neural network by miniming your (favorite) loss $\mathcal{L}$ on the training set:

$$\begin{aligned} \min_{\substack{W_1,\ldots,W_K \\ b_1,\ldots,b_K}} \mathcal{L}\left(\hat{f}(x_i), y_i\right) \\ \text{s.t. } \hat{f} = \sigma_K(W_K\sigma_{K-1}(W_{K-1}\sigma_{K-2}(\ldots) + b_{K-1}) + b_K) \end{aligned}$$

Algorithm choice: use SGD and/or variants (with mini-batch) if possible with a GPU (tailored for fast matrix/vector operations)

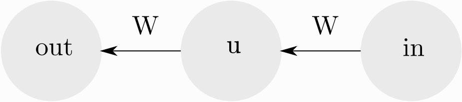
Outline

Chain rule: $\frac{\partial out}{\partial in} = \frac{\partial out}{\partial u}\frac{\partial u}{\partial in}$



[7] Ian Goofellow, Yoshua Yoshua, and Aaron Courville (2016). *Deep Learning*. MIT Press

Chain rule: $\frac{\partial out}{\partial in} = \frac{\partial out}{\partial u}\frac{\partial u}{\partial in}$



Risk with too deep NN: if $W = P\,\mathrm{diag}(\lambda_i)_i P^{-1} \implies W^k = P\,\mathrm{diag}(\lambda_i^k) P^{-1}$.

- $\lambda_i > 1 \implies$ exploding gradient,
- $\lambda_i < 1 \implies$ vanishing gradient.

Diagnose the vanishing / exploding gradient

- necessary condition (vanishing): weights distribution barely changing (not sufficient!)
- learning curve very unstable or not decreasing (also not sufficient)

(7) Ian Goofellow, Yoshua Yoshua, and Aaron Courville (2016). *Deep Learning*. MIT Press

Outline

Sigmoïd activation induce gradient issues

Very low variations in the tails $\Longrightarrow$ flat gradient!

Sigmoïd activation induce gradient issues

Very low variations in the tails $\Longrightarrow$ flat gradient!

RELU activations $=$ our rescue?

Sigmoïd activation induce gradient issues

Very low variations in the tails $\implies$ flat gradient!

RELU activations = our rescue? NO (even if the internet often says so).

- too much dying RELUs $\implies$ zero-out layers
- same for ℓ_2 and ℓ_1 penalties (but great to prevent exploding gradients instead of gradient clipping)

Currently there is no savior, just band-aids

Batch normalizations, weight initializations, leaky RELUs, … = help

Leave MLPs and go to Residuals networks.

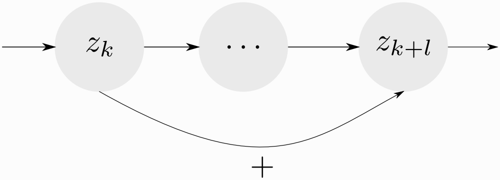
Outline

ResNets

We need to go deeper (8)

Idea: add "skip connections" to avoid flatlining gradients

$$z_{k+l} = \sigma_{k+l-1}(z_{k+l-1}) + z_k \Rightarrow \frac{\partial z_{k+l}}{z_k} = \frac{\partial \sigma_{k+l-1}(z_{k+l-1})}{\partial z_k} + 1$$



Let's adress the dimensionality

- dimension is an issue in CNNs (and any NN using pooling)
- authors suggested using a linear projection of z_k if needed.
- l is kept fairly small (2 or 3).

(7) K. He et al. (2015). "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385 [cs]*

Conclusion

- New techniques come up very often,
- Most of the time, understanding the basics is enough to scrap the ideas and use them in your situation.
- Lots of empirical results and strategies adopted, the *why does it (not) work* is sometimes left behind (lots of research needed),
- Time vs Cost $\Longrightarrow$ essential factor in (re)using lots of NN.

We are now ready to tackle both theoretically and practically the next sessions

References I

- Cybenko, G. (Dec. 1989). "Approximation by superpositions of a sigmoidal function". en. In: *Mathematics of Control, Signals and Systems* 2.4, pp. 303–314.
- Devroye, L., L. Györfi, and G. Lugosi (1996). *A probabilistic theory of pattern recognition*. Vol. 31. Applications of Mathematics (New York). New York: Springer-Verlag.
- Dosovitskiy, A. et al. (June 2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *arXiv:2010.11929 [cs]*. arXiv: 2010.11929.
- Girosi, F. and T. Poggio (1989). "Representation properties of networks: Kolmogorov's theorem is irrelevant". In: *Neural Computation* 1.4, pp. 465–469.
- He, K. et al. (Dec. 2015). "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385 [cs]*. arXiv: 1512.03385.
- Ian Goofellow, Yoshua Yoshua, and Aaron Courville (2016). *Deep Learning*. MIT Press.

References II

- Kolmogorov, A. N. (1956). "The representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables". In: *Doklady Akademii Nauk SSSR* 108.2, pp. 179–182.
- Kůrková, V. (Dec. 1991). "Kolmogorov's Theorem Is Relevant". In: *Neural Computation* 3.4, pp. 617–622.
- Lu, Z. et al. (2017). *The Expressive Power of Neural Networks: A View from the Width*.
- Minsky, M. L. and S. A. Papert (1969). *Perceptrons. An Introduction to Computational Geometry. 1969, Expanded*. Cambridge, MA: MIT Press.
- Novikoff, A. B. (1963). *On convergence proofs for perceptrons*. Tech. rep. STANFORD RESEARCH INST MENLO PARK CA.
- Paszke, A. et al. (Dec. 2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *arXiv:1912.01703 [cs, stat]*. arXiv: 1912.01703.
- Rosenblatt, F. (1958). *The perceptron: a theory of statistical separability in cognitive systems (Project Para)*. Cornell Aeronautical Laboratory.
- Schmidt-Hieber, J. (2021). "The Kolmogorov–Arnold representation theorem revisited". In: *Neural Networks* 137, pp. 119–126.