1. Feedforward Neural Networks

(FFNN)

($\hookrightarrow$ NN)

$\hookrightarrow$ MLP = Multi-layer Perceptron.

1.A Intro

Goal of a NN

approximate a fonction f^*

$\hookrightarrow$ ex: classifier $y = f^*(x)$

category y

input data x

FFNN $y = f(x, \theta)$ learns the θ that results in the best f° approximation of f^{*}.

Network

hidden layer

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

3rd layer → output layer.

2nd layer

1st layer.

Training of NN: drive $f(x)$ towards $f^*(x)$

$\boxed{\text{training data } \mathcal{D}}$

↳ provides noisy approximation de $f^*(x)$ evaluated at training points.

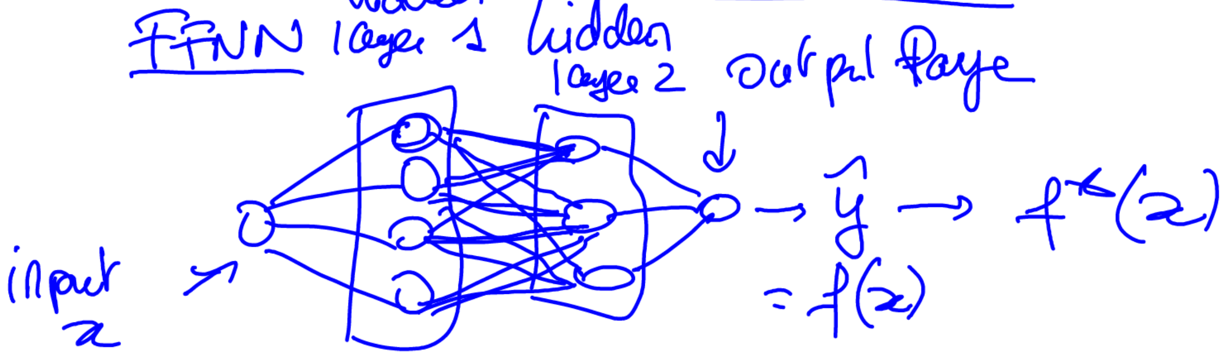
label y : $y \approx f^*(x)$

↳ specify the behavior of the output layer at each point $x_i \in \mathcal{D}$

behavior of the other layers is not specified by the training samples.

↳ The learning algo should 'learn' it.

HIDDEN LAYERS



hidden layer

outputs a $\boxed{\text{vector}}$: dimensionality of this vector specifies the <u>WIDTH</u> of the NN.

each element of the vector plays a role analogical to a vector.

<u>DEPTH of a NN</u> : # of layers of the NN.

B. Example: learning XOR fonction

$(x_1;x_2) \in \{0,1\}^2 \longrightarrow X = \{[0,0], [1,1], [1,0], [0,1]\}$

$$\left\| \begin{array}{l} f^*(1,1) = f^*(0,0) = 0 \\ f^*(0,1) = f^*(1,0) = 1 \end{array} \right.$$

model: $y = f(x,\theta) \xrightarrow[\text{learn } \theta \text{ so that}]{} f^*$

$\hookrightarrow$ fit to the learning set X.

treat this as a regression pb:

↳ loss function: Mean Square Error (MSE).

$$\boxed{J(\theta) = \frac{1}{4} \sum_{x \in X} \left(f^*(x) - f(x, \theta) \right)^2}$$

↳ form of the model: $\boxed{f_{LIN}(x, w, b) = x^T w + b.}$

$\boxed{w = 0, \ b = 1/2}$ → outputs 1/2 partout!!

FFWN

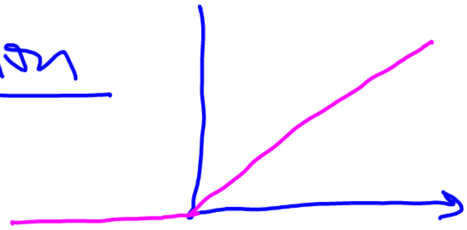
$h = f^{(1)}(x, W, c)$ → hidden layer

$y = f^{(2)}(h, w, b)$ → output layer

$h = g(w^T x + b)$

fonction d'activation

ReLU $\quad g(z) = \max\{0; z\}$



$$f(x, W, c, w, b) \qquad x \in \{0,1\}^2$$

$$\to f^{(1)} = g(W^T x + c)$$

$$f^{(2)}(h) = w^T h + b.$$

$$= w^T g(W^T x + c) + b.$$

$$\boxed{= w^T \max\{0, W^T x + c\} + b}.$$

$$\boxed{W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad c = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad w = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad b = 0}$$

II/ Gradient based learning

↳ Billions of Parameters!

↳ Billions of Training Data!

  ↳ <u>gradient-based algo</u> to learn θ.

⚠ Loss f° associated to NNs are not <u>convex</u>.

↳ iterative methods that drive the loss function $J(\theta)$ towards a very low value.

↳ Sensibility ~~of~~ to parameters Initializa°.

$W^T x + b$

- W: weights → small random values.
- b: biais → zero or very small positive values

Training algo

↳ loss function $J(\theta)$ → to choose (depending on the pb)

↳ $f(x, \theta) = y$ → choice of the NN architecture

NN architecture

- # of layers : DEPTH
- dim of each layer : WIDTH
- f^o d'activa° pour chaque couche.

II.A Cost function

Maximum likelihood principle

↳ cost f^o = negative log likelihood.

$$\boxed{J(\theta) = - \mathbb{E}_{\underbrace{x,y \sim \hat{p}_{data}}_{\text{training data.}}} \log p_{model}(y|x)}$$

type of SL pbs
supervised learning

- regression problem → $y \in \mathbb{R}$.
- classifica° problem → category y (discrete)
  - → classifica° 2 classes
  - → multiclass classifica° $(c > 2)$

Regression case

target   identity matrix

$$p_{model}(y|x) = \mathcal{N}(y, f(x,\theta), I)$$

output of NN

MSE

$$\boxed{J(\theta) = \frac{1}{2} \mathbb{E}_{x,y \sim \hat{p}_{data}} \|y - f(x,\theta)\|^2 + \text{cste}}$$

$$\boxed{x \in \mathbb{R}^k \to f(x \in \mathbb{R}^k, \mu, \Sigma) = \frac{\exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)}{\sqrt{(2\pi)^k |\Sigma|}}}$$

Advantage of ML principle

↳ undo the 'exp' : f° d'activation have exponential terms

↳ No saturation

↳ better for Gradient learning.

ML Principle → Pmodel ⇒ we do not learn all $p_\theta(y|x)$

Conditional statistics

⟶ mean of Pmodel.

II B / Output units

$J(\theta)$ dep $f(x,\theta) \rightarrow$ dependant of the output layer

$h = f'(x, \theta)$

output of the last ~~out~~ hidden layer

$\hookrightarrow$ y? $\rightarrow$ shape of the output layer?

1. linear output layer

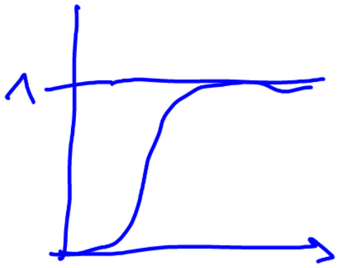
$$\boxed{\hat{y} = \underbrace{W^T \times h}_{\text{hidden representatio}} + b.} \rightarrow \boxed{\text{mean of Gaussian output Distributio}}$$

↳ the output layer for a regression problem

2. sigmoïd output layer

output layer ⟺ Pmodel = Bernoulli output Distributio

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



$\hat{y} = \sigma(w^T h + b)$

↳ For tasks requiring a binary variable $\hat{y} \in \{0, 1\}$

↳ $p(y=1 / x)$ → si $p(y=1 / x) \leq 0.5$ $y = 0$

$p(y=1/x) > 0.5$ $y = 1$

⇓

output layer used for BINARY CLASSIFICAº PB

3. softmax output layer

$$(\text{softmax}(z))_i = \frac{\exp(z_i)}{\sum_{j \in \{1..d\}} \exp(z_j)} \in [0,1].$$

$z \in \mathbb{R}^d$

→ p_{model} → distribution multinomiale (Multinoulli Probability Distribution)

$$\boxed{\hat{y} = \text{softmax}(W^T h + b)} \in [0,1]^d$$

output layer for multi-class classification

N class classification pb:

$\hat{y} \in [0,1]^N \longrightarrow \underbrace{\hat{c} = \text{argmax}\ \hat{y}}_{\text{prediction de la classe}}$

II c/ activation functions for hidden layers

which activation f°?

(d, h)

parameters of the layer

hidden layer: $\boxed{g}(z = W^T x + b) \rightarrow \in \mathbb{R}^h$

$z \in \mathbb{R}^h$ weights $x \in \mathbb{R}^d$ biais

standardiser: $\dfrac{x-\mu}{\sigma}$ μ = mean, σ = ecart-type.

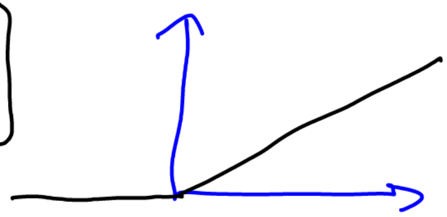
normalisen → avec le max et le min.

$$f(x,\theta) = \underbrace{f^{(3)}}_{\text{output layer}}\left(f^{(2)}\left(f^{(1)}(x)\right)\right)$$

$h_2 = g_2\left(W_2^T h_1 + b_2\right)$ $\quad h_1 = g_1\left(W_1^T x + b_1\right)$

fonction ReLU

$$\boxed{g(z) = \max\{0, z\}}$$



↳ easy to optimize ($\sim$ linear)

↳ drawback: cannot learn via gradient-based methods for example equal to 0.

→ | Leaky ReLU
maxout units

if sigmoidal f° should be used prefer tanh

logistic sigmoid & hyperbolic tangent

} → 2 fonctions d'activation avant découverte de ReLU.

sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\boxed{\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}}$$

↳ DRAWBACKS: when z is very positive, saturate to high values
when z is very negative, saturate to very low values.

II/D) Architecture design

type du réseau

↳ type de donnée d'entrée

(structure de la donnée)

- $\boxed{\text{FFNN}}$ → donnée tabulaire
- CNN : $\boxed{\text{Convolutional}}$ Neural Network → Images
- RNN : $\boxed{\text{Recurrent}}$ Neural Network → texte, séries temporelles.

on fixe type du réseau

- depth = nbre de couches
- width = dimension de chaque couche = nbre d'unités / de neurons de la couche

ième couche cachée

$$h^{(i)} = g^{(i)}\left(W^{(i)T} h^{(i-1)} + b^{(i)}\right)$$

size of the matrix.

Universal Approximation Properties / Theorem

States that a FFNN with a linear output layer and at least one hidden layer with any "squashing" activation function can approximate any Borel $f(D_i \subset \mathbb{R}^n$ fermé et borné$)$ measurable function from one dimensional space $D_i \subset \mathbb{R}^n \to D_o \subset \mathbb{R}^n$ to another with any desired non-zero amount of error provided that the network has enough units

In theory, this theorem means that we can consider 'one-layer' FFNN
↳ (A) one hidden layer
(B) one output layer.

BUT in practice we cannot guarantee that the training algo will be learn that function
↳ An exponential # of units may be required!
↳ use DEEPER NETWORKS instead.

III/ The Backpropagation algorithm

Input $x \xrightarrow[\text{FFNN}]{} \hat{y}$ Output : input x provides initial information and then propagates up to the hidden layers until finally producing $\hat{y}$ ⊗ FORWARD PROPAGAº

⊗ Pass Forward

Backpropagation: allow the information to flow backward through the network, to compute the gradient

Gradient-based learning → loss function $J(\theta)$

↳ θ^* that minimizes $J(\theta)$

$$\boxed{\theta_k = \theta_{k-1} - \alpha \frac{\nabla J(\theta)}{\partial \theta}}$$

how to compute efficiently this gradient?

learning rate / tx d'apprentissage.

FNN : $f(x,\theta) = f^{(n)}(f^{(n-1)} \dots f^{(1)}(x)))$

Based on chain rule of calculus :

$z = f(g(x))$
$\;\; = f(y)$

$$\boxed{\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \times \frac{\partial y}{\partial x}}$$ scalar case

Generalizaᵒ to $\mathbb{R}^n$

$x \in \mathbb{R}^n = (x_1 \dots x_n)$

$$\boxed{\frac{\partial z}{\partial x_i} = \sum_j \left(\frac{\partial z}{\partial y_j} \times \frac{\partial y_j}{\partial x_i}\right)}$$

Vector notation : $\boxed{\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T \nabla_y z}$

$\in \mathbb{R}^n$

$\in \mathbb{R}^m$

$z = f(\underbrace{g(x)}_{y})$

Matrice Jacobienne

n×m