Pre-course, part II:
black-box machine learning for image denoising

Luca Ratti

Department of Mathematics, Università degli studi di Bologna,

luca.ratti5@unibo.it

ERASMUS+ International PhD Summer School 2025
Mathematics and Machine Learning for image analysis

University of Bologna
3rd June 2025

A statistical learning perspective on image denoising

Statistical learning - a supervised regression problem

On a completely different note...

[Cucker, Smale, *On the mathematical foundations of learning*, 2002]

A supervised regression problem

Given two random variables x on X and y on Y

find a function $f\colon Y \to X$ s.t. $f(y) \approx x$

- (ideal setting) knowing the joint distribution $\pi_{y,x} \sim (y, x)$;
- (real setting) knowing a sample $\{(\mathbf{y}_i, \mathbf{x}_i)\}_{i=1}^{N} \sim_{i.i.d} \pi_{y,x}$;

Statistical learning - a supervised regression problem

On a completely different note...
[Cucker, Smale, *On the mathematical foundations of learning*, 2002]

A supervised regression problem

Given two random variables x on X and y on Y
find a function $f\colon Y \to X$ s.t. $f(y) \approx x$

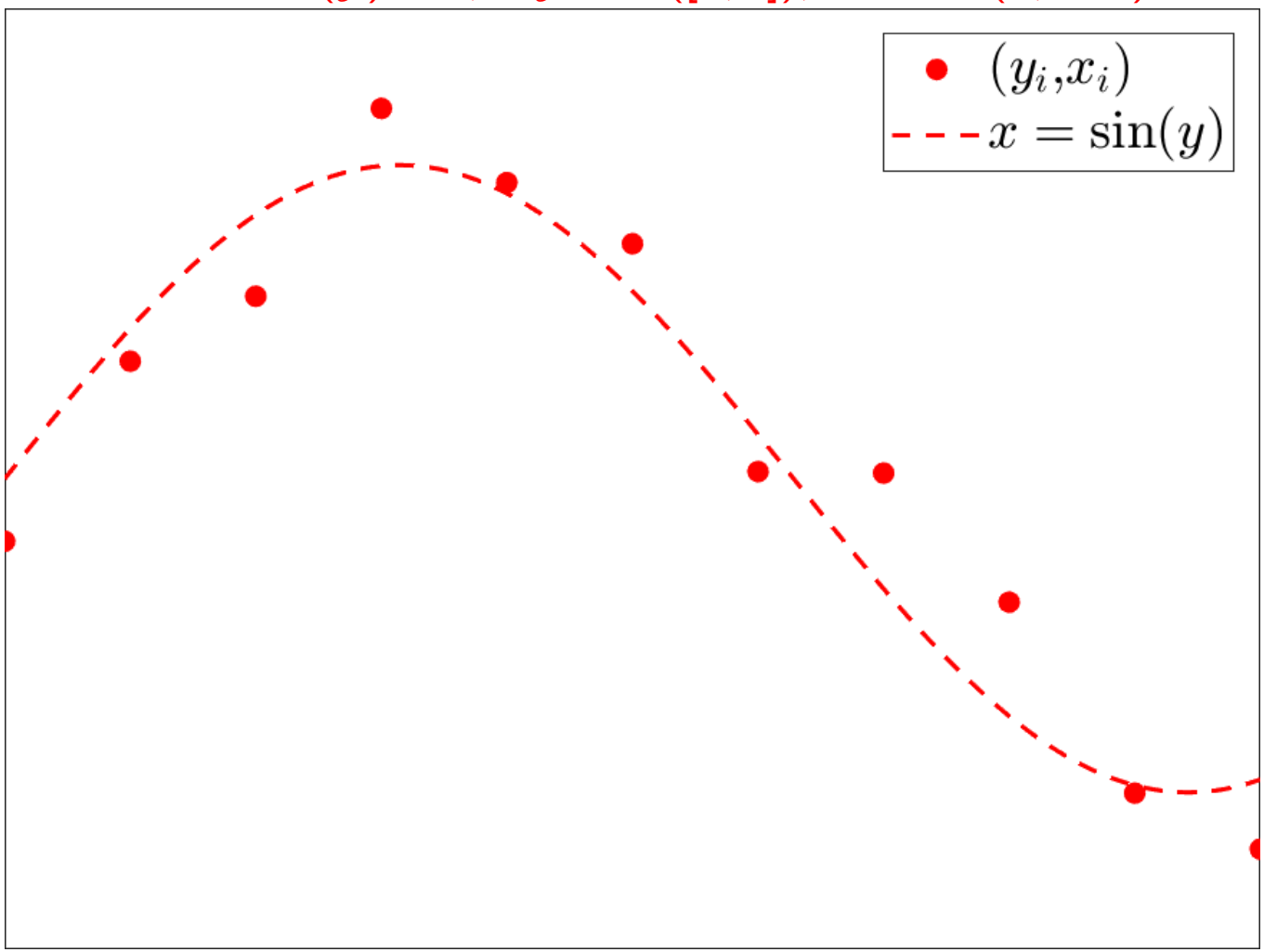
- (ideal setting) knowing the joint distribution $\pi_{y,x} \sim (y, x)$;
- (real setting) knowing a sample $\{(\mathbf{y}_i, \mathbf{x}_i)\}_{i=1}^N \sim_{i.i.d} \pi_{y,x}$;

1) Consider a loss function, e.g. $\ell(\mathbf{x}', \mathbf{x}) = \|\mathbf{x}' - \mathbf{x}\|_X^2$.
2) Introduce $\mathcal{F}$: hypothesis space, a set of functions from Y to X.
3.id) Minimize the **Expected Loss**: $\rightsquigarrow \min_{f \in \mathcal{F}} \mathbb{E}_{(x,y)\sim\pi_{xy}} \Big[\ell(f(y), x)\Big]$
3.re) Minimize the **Empirical Risk**: $\rightsquigarrow \min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \ell(f(\mathbf{y}_i), \mathbf{x}_i)$

How we usually see regression...

$$X = \mathbb{R}, \quad Y = \mathbb{R},$$

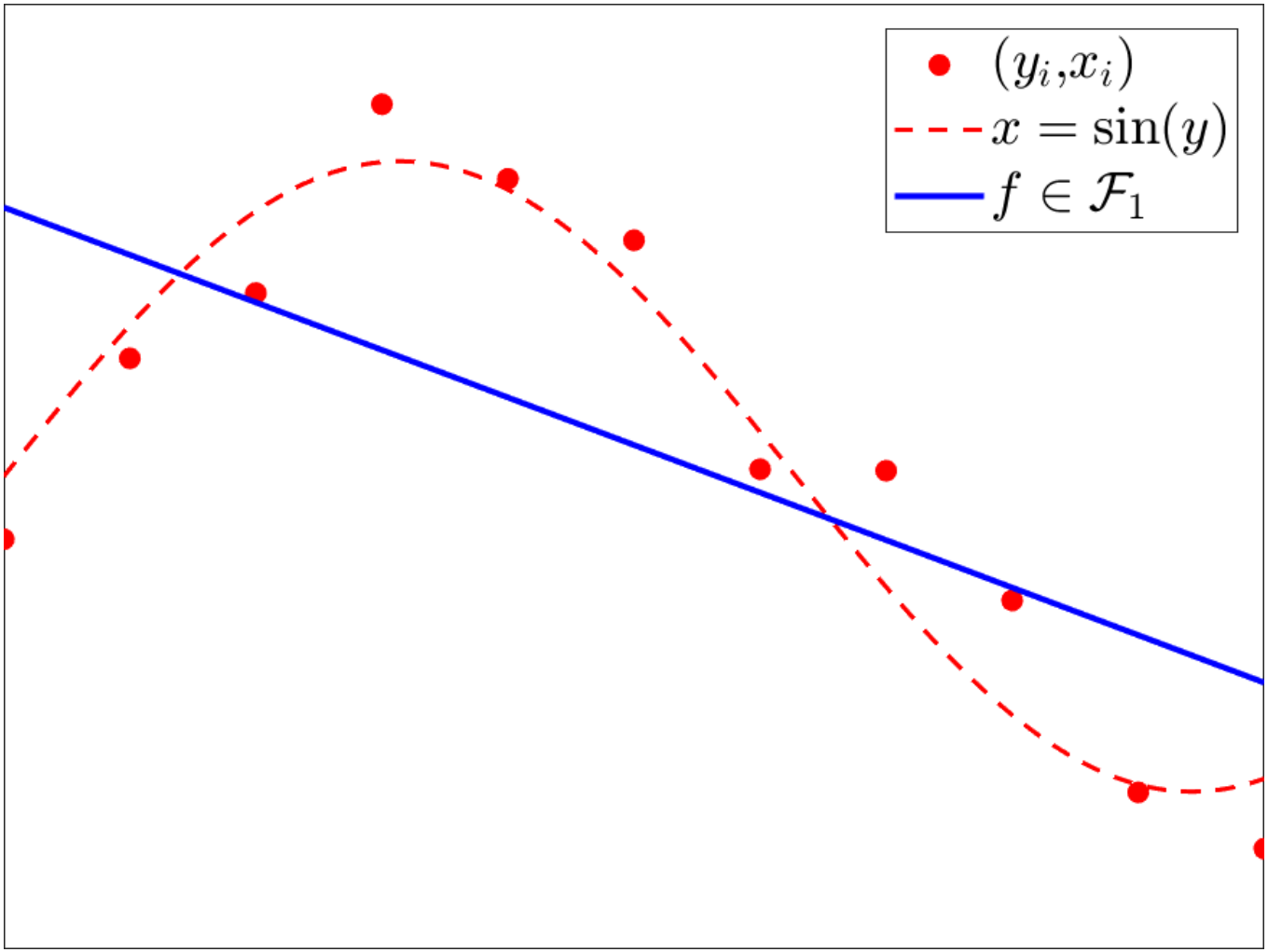
$$x = \sin(y) + \varepsilon, \quad y \sim \mathcal{U}([0, 5]), \quad \varepsilon \sim \mathcal{N}(0, 0.2^2)$$



How we usually see regression...

$X = \mathbb{R}, \quad Y = \mathbb{R},$

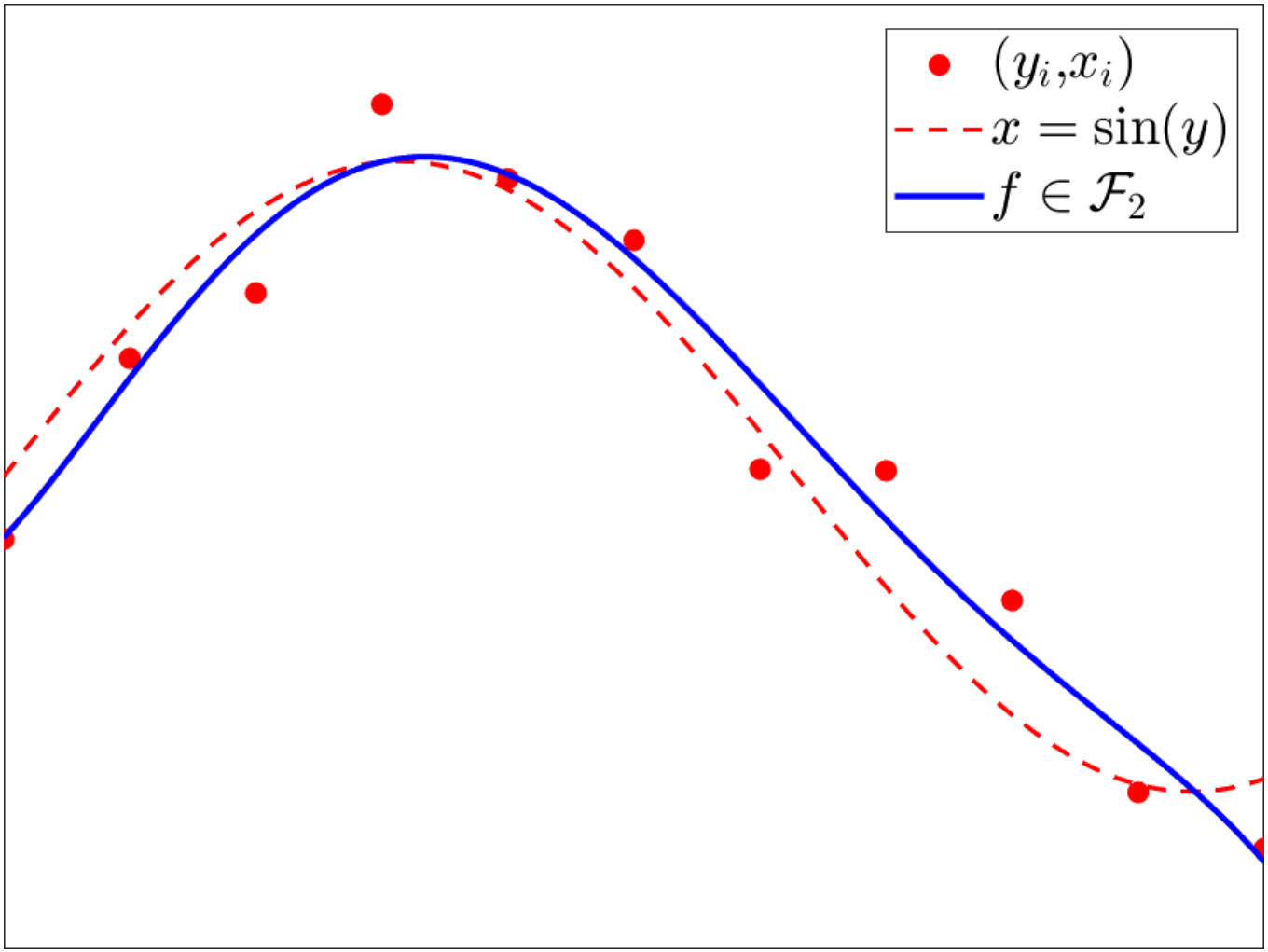
$\mathcal{F}$: polynomials of degree 1



How we usually see regression...

$X = \mathbb{R}, \quad Y = \mathbb{R},$

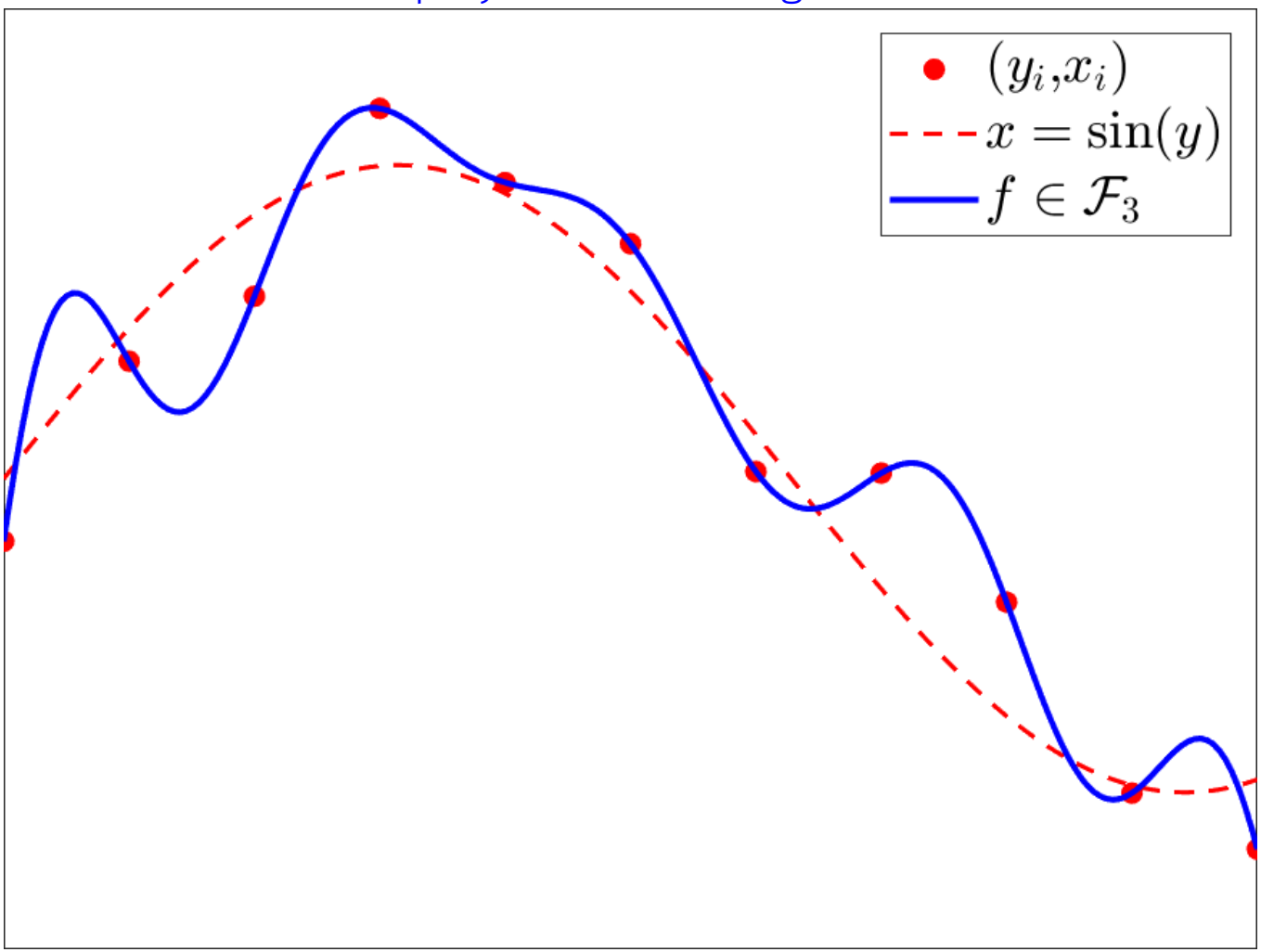
$\mathcal{F}$: polynomials of degree 5



How we usually see regression...

$$X = \mathbb{R}, \quad Y = \mathbb{R},$$

$\mathcal{F}$: polynomials of degree 10



... but denoising can be seen as a regression task!

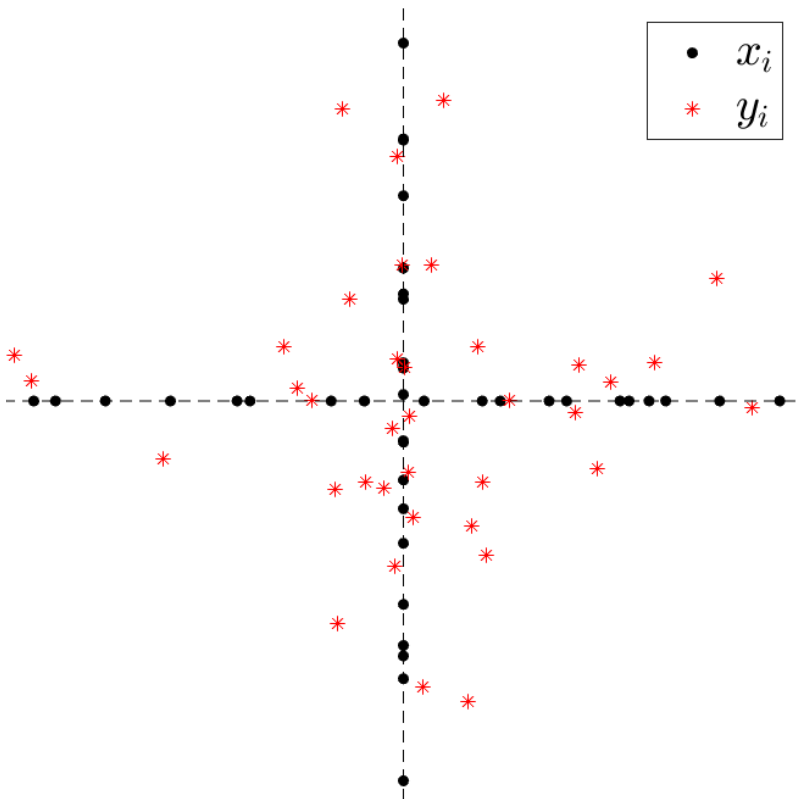
Learned reconstruction for inverse problems:
suppose that $y = \mathsf{A}x + \boldsymbol{\varepsilon}$ (forward model): $f(y) \approx x \rightsquigarrow$ find $f \approx \mathsf{A}^{-1}$.

Denoising: let $X = Y$ and $\mathsf{A} = \mathsf{Id}$, i.e.: $y = x + \boldsymbol{\varepsilon}$.
Why not using simply $f = \mathsf{A}^{-1} = \mathsf{Id}$?

... but denoising can be seen as a regression task!

Learned reconstruction for inverse problems:
suppose that $y = \mathsf{A}x + \varepsilon$ (forward model): $f(y) \approx x \rightsquigarrow$ find $f \approx \mathsf{A}^{-1}$.

Denoising: let $X = Y$ and $\mathsf{A} = \mathsf{Id}$, i.e.: $y = x + \varepsilon$.
Why not using simply $f = \mathsf{A}^{-1} = \mathsf{Id}$?



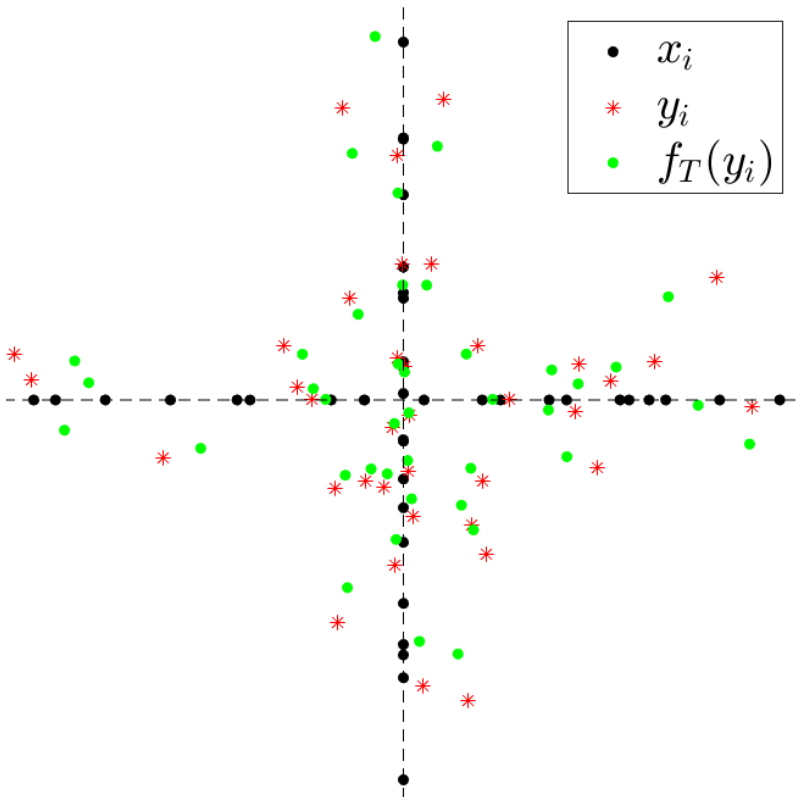
The simplest example:

- $X = \mathbb{R}^2$;
- $\varepsilon \sim \mathcal{N}(0, 0.1^2\,\mathsf{Id})$;
- x : uniformly distributed on $\{0\} \times [-1, 1] \cup [-1, 1] \times \{0\}$;
- $y = x + \varepsilon$.
- $f = \mathsf{Id}$
  relative error: 20.70%

... but denoising can be seen as a regression task!

Learned reconstruction for inverse problems:
suppose that $y = \mathsf{A}x + \varepsilon$ (forward model): $f(y) \approx x \rightsquigarrow$ find $f \approx \mathsf{A}^{-1}$.

Denoising: let $X = Y$ and $\mathsf{A} = \mathsf{Id}$, i.e.: $y = x + \varepsilon$.
Why not using simply $f = \mathsf{A}^{-1} = \mathsf{Id}$?



The simplest example:

- $X = \mathbb{R}^2$;
- $\varepsilon \sim \mathcal{N}(0, 0.1^2\,\mathsf{Id})$;
- x : uniformly distributed on $\{0\} \times [-1, 1] \cup [-1, 1] \times \{0\}$;
- $y = x + \varepsilon$.
- Variational denoiser: Tikhonov

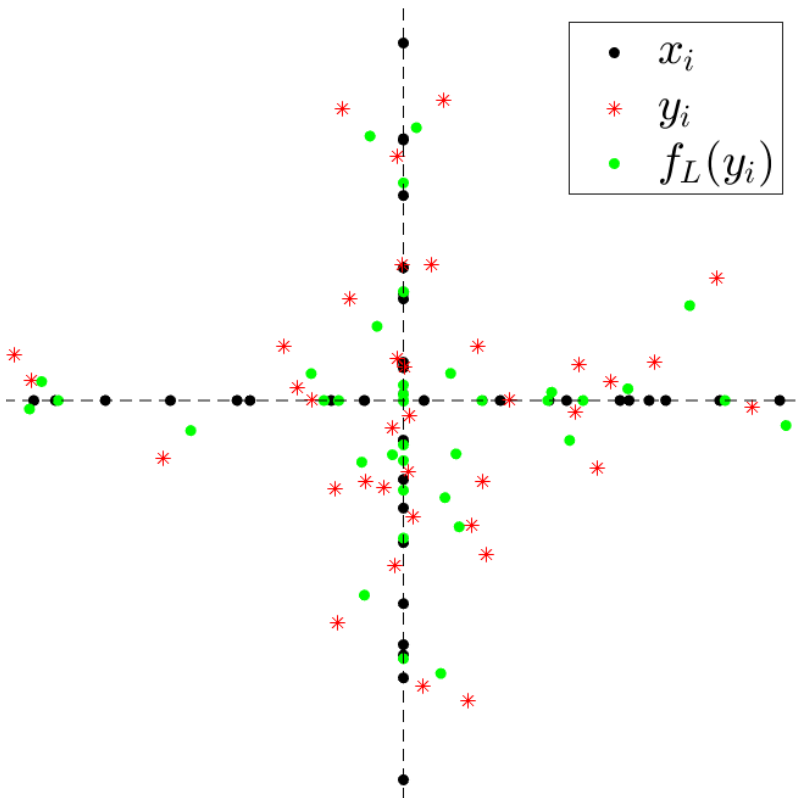
$$f_T(\mathsf{y}) = \arg\min_{\mathsf{x}} \left\{ \frac{1}{2}\|\mathsf{x} - \mathsf{y}\|_2^2 + \frac{\lambda}{2}\|\mathsf{x}\|_2^2 \right\}$$
$$= \frac{1}{1+\lambda}\mathsf{y}$$

relative error: 19.21%

... but denoising can be seen as a regression task!

Learned reconstruction for inverse problems:
suppose that $y = \mathsf{A}x + \varepsilon$ (forward model): $f(y) \approx x \rightsquigarrow$ find $f \approx \mathsf{A}^{-1}$.

Denoising: let $X = Y$ and $\mathsf{A} = \mathsf{Id}$, i.e.: $y = x + \varepsilon$.
Why not using simply $f = \mathsf{A}^{-1} = \mathsf{Id}$?



The simplest example:

- $X = \mathbb{R}^2$;
- $\varepsilon \sim \mathcal{N}(0, 0.1^2\,\mathsf{Id})$;
- x : uniformly distributed on $\{0\} \times [-1, 1] \cup [-1, 1] \times \{0\}$;
- $y = x + \varepsilon$.
- Variational denoiser: Lasso

$$f_L(y) = \mathsf{prox}_{\lambda\|\cdot\|_1}(\mathsf{y}) = \mathsf{ST}(\mathsf{y}; \lambda)$$

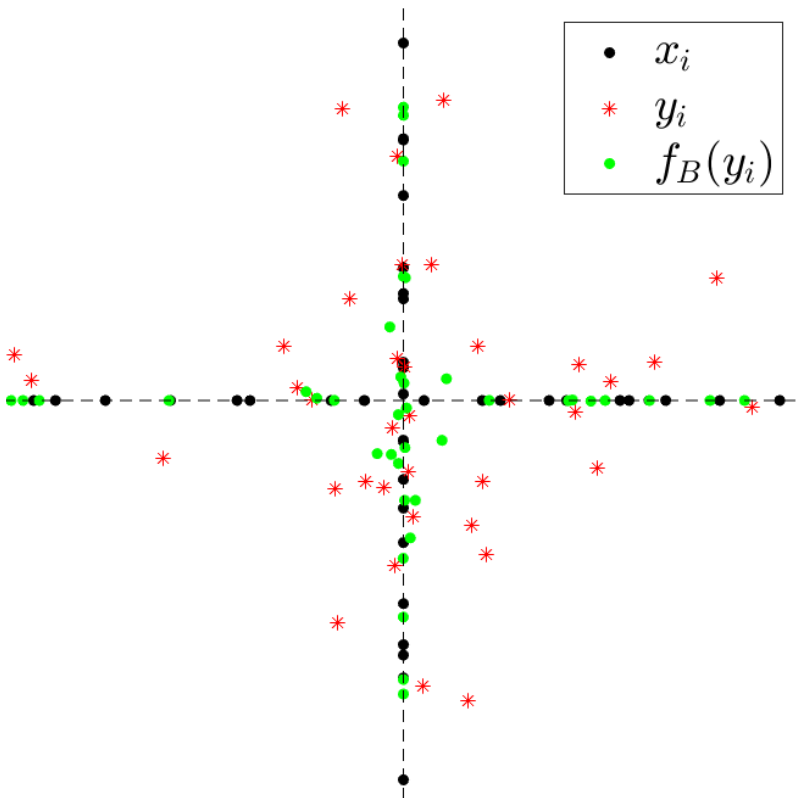
$$[f_L(\mathsf{y})]_j = \begin{cases} \mathsf{sign}(y_j)(|y_j| - \lambda) \text{ if } |y_j| > \lambda \\ 0 \text{ if } |y_j| \leq \lambda \end{cases}$$

relative error: 18.41%

... but denoising can be seen as a regression task!

Learned reconstruction for inverse problems:
suppose that $y = \mathsf{A}x + \varepsilon$ (forward model): $f(y) \approx x \rightsquigarrow$ find $f \approx \mathsf{A}^{-1}$.

Denoising: let $X = Y$ and $\mathsf{A} = \mathsf{Id}$, i.e.: $y = x + \varepsilon$.
Why not using simply $f = \mathsf{A}^{-1} = \mathsf{Id}$?



The simplest example:

- $X = \mathbb{R}^2$;
- $\varepsilon \sim \mathcal{N}(0, 0.1^2\,\mathsf{Id})$;
- x : uniformly distributed on $\{0\} \times [-1, 1] \cup [-1, 1] \times \{0\}$;
- $y = x + \varepsilon$.
- Learned denoiser (ideal case)
  $f_B = \ldots$ - Bayes denoiser
  relative error: 16.56%

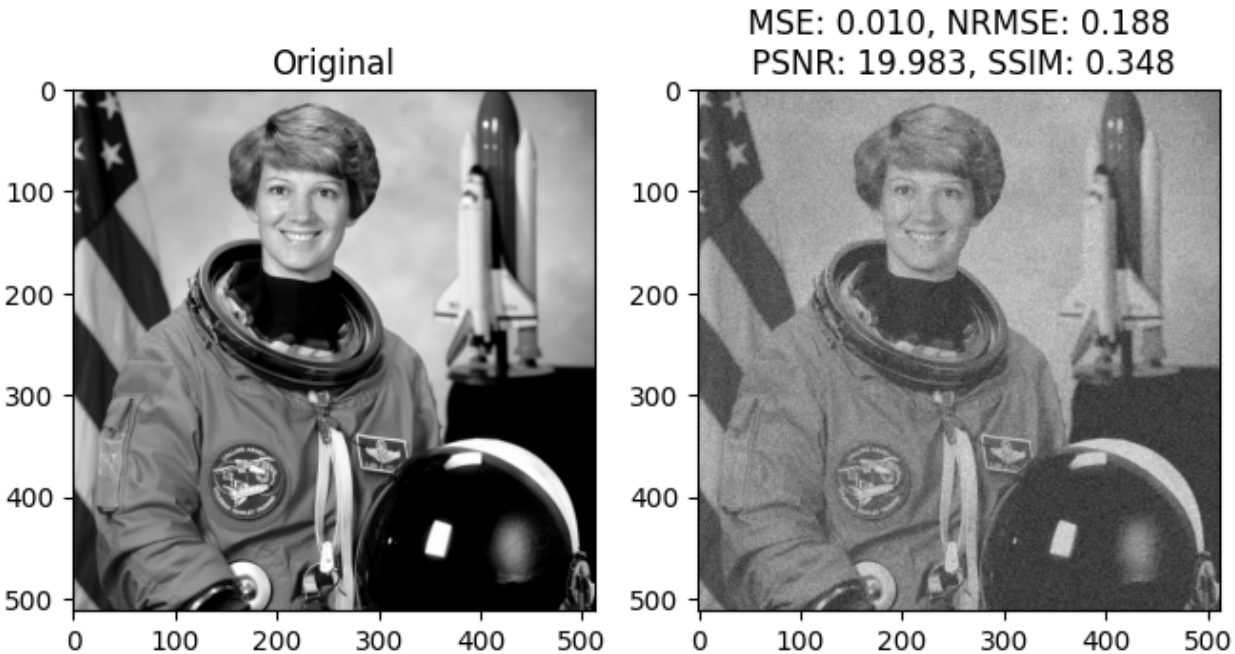
Image Denoising as a regression problem

Setup:

- Each image is a vector $\mathbf{x} \in \mathbb{R}^n$, where $n = \text{height} \times \text{width} \times \text{channels}$
- Noisy image: $\mathbf{y} = \mathbf{x} + \boldsymbol{\varepsilon}$, Clean image: $\mathbf{x}$, Noise: $\boldsymbol{\varepsilon}$
- **Goal:** learn $f : \mathbb{R}^n \to \mathbb{R}^n$ such that $f(\mathbf{y}) \approx \mathbf{x}$

Loss Function:

$$\ell(f(\mathbf{y}), \mathbf{x}) = \text{MSE}(f(\mathbf{y}), \mathbf{x}) = \frac{1}{n}\sum_{j=1}^{n}([f(\mathbf{y})]_j - [\mathbf{x}]_j)^2 = \frac{1}{n}\|f(\mathbf{y}) - \mathbf{x}\|_2^2$$



Audience-driven slide

In your experience,
what are key ingredients of statistical learning?

Audience-driven slide

In your experience,
what are key ingredients of statistical learning?

My personal list

A. a training dataset;

B. a loss function;

C. a (parametric) hypothesis class;

D. an optimization algorithm.

A statistical learning perspective on image denoising

A. Training datasets

Supervised datasets and beyond

1) Supervised setting

A dataset of paired noisy and clean images $\{(\mathbf{y}_i, \mathbf{x}_i)\}_{i=1}^{N}$ is available.

- empirical loss minimization.

Supervised datasets and beyond

1) Supervised setting

A dataset of paired noisy and clean images $\{(\mathbf{y}_i, \mathbf{x}_i)\}_{i=1}^N$ is available.

- empirical loss minimization.

2) Self-supervised setting

A dataset of clean images $\{(\mathbf{x}_i)\}_{i=1}^N$ is available $+$ the noise model is known (e.g., $y = x + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$)

- sample $\{\varepsilon_i\}_{i=1}^N \sim_{i.i.d.} \pi_\varepsilon$, define $\mathbf{y}_i = \mathbf{x}_i + \varepsilon_i$;
- empirical loss minimization.

Supervised datasets and beyond

1) Supervised setting

A dataset of paired noisy and clean images $\{(\mathbf{y}_i, \mathbf{x}_i)\}_{i=1}^N$ is available.

- empirical loss minimization.

2) Self-supervised setting

A dataset of clean images $\{(\mathbf{x}_i)\}_{i=1}^N$ is available $+$ the noise model is known (e.g., $y = x + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \sigma^2 \mathsf{I})$)

- sample $\{\varepsilon_i\}_{i=1}^N \sim_{i.i.d.} \pi_\varepsilon$, define $\mathbf{y}_i = \mathbf{x}_i + \varepsilon_i$;
- empirical loss minimization.

3) Unsupervised setting - case x

A dataset of clean images $\{(\mathbf{x}_i)\}_{i=1}^N$ is available.

- learn the prior distribution π_x (e.g. via Tweedie's formula);
- use a model-adaptive algorithm to identify the noise model.

Supervised datasets and beyond

1) Supervised setting

A dataset of paired noisy and clean images $\{(\mathbf{y}_i, \mathbf{x}_i)\}_{i=1}^N$ is available.

- empirical loss minimization.

2) Self-supervised setting

A dataset of clean images $\{(\mathbf{x}_i)\}_{i=1}^N$ is available $+$ the noise model is known (e.g., $y = x + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$)

- sample $\{\varepsilon_i\}_{i=1}^N \sim_{i.i.d.} \pi_\varepsilon$, define $\mathbf{y}_i = \mathbf{x}_i + \varepsilon_i$;
- empirical loss minimization.

3) Unsupervised setting - case x

A dataset of clean images $\{(\mathbf{x}_i)\}_{i=1}^N$ is available.

- learn the prior distribution π_x (e.g. via Tweedie's formula);
- use a model-adaptive algorithm to identify the noise model.

4) Unsupervised setting - case y

A dataset of noisy images $\{(\mathbf{y}_i)\}_{i=1}^N$ is available.

- ad-hoc techniques (when available).

A statistical learning perspective on image denoising

B. Loss function

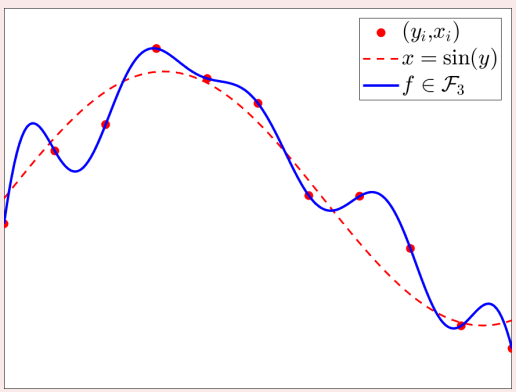
Loss minimization and regularization

- **Loss function, ℓ:** measures the quality of a single denoised image.
  Es.: $\ell(\mathbf{x}', \mathbf{x}) = \|\mathbf{x}' - \mathbf{x}\|_2^2, \quad \ell(\mathbf{x}', \mathbf{x}) = \mathrm{PSNR}(\mathbf{x}', \mathbf{x})$.
- **Expected loss, L:** measures the quality of a denoiser f, using the full knowledge of $\pi_{y,x}$ (ideal). Es. $L(f) = \mathbb{E}_{(y,x)\sim\pi_{(y,x)}}[\ell(f(y), x)]$.
- **Empirical risk, $\hat{L}$:** measures the quality of a denoiser f, using a supervised dataset. Es. $\hat{L}(f) = \frac{1}{m}\sum_{i=1}^{N} \ell(f(\mathbf{y}_i), \mathbf{x}_i)$.

Loss minimization and regularization

- **Loss function, ℓ:** measures the quality of a single denoised image.
  Es.: $\ell(\mathbf{x}', \mathbf{x}) = \|\mathbf{x}' - \mathbf{x}\|_2^2$, $\quad \ell(\mathbf{x}', \mathbf{x}) = \mathrm{PSNR}(\mathbf{x}', \mathbf{x})$.
- **Expected loss, L:** measures the quality of a denoiser f, using the full knowledge of $\pi_{y,x}$ (ideal). Es. $L(f) = \mathbb{E}_{(y,x)\sim\pi_{(y,x)}}[\ell(f(y), x)]$.
- **Empirical risk, $\hat{L}$:** measures the quality of a denoiser f, using a supervised dataset. Es. $\hat{L}(f) = \frac{1}{m}\sum_{i=1}^{N} \ell(f(\mathbf{y}_i), \mathbf{x}_i)$.

The risk of overfitting



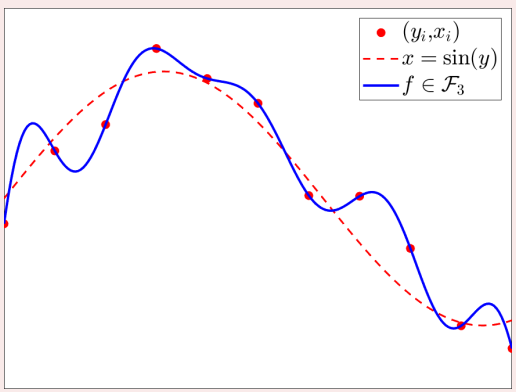
Minimizing $\hat{L}$ among all possible $f\colon Y \to X$ is not a good idea:

- the result f might work poorly on new data $y_{m+1}, y_{m+2}, \ldots$
- the result f might be unstable ($\|f\| \gg 1$)

Loss minimization and regularization

- **Loss function, ℓ:** measures the quality of a single denoised image.
  Es.: $\ell(\mathbf{x}', \mathbf{x}) = \|\mathbf{x}' - \mathbf{x}\|_2^2$, $\quad \ell(\mathbf{x}', \mathbf{x}) = \mathrm{PSNR}(\mathbf{x}', \mathbf{x})$.
- **Expected loss, L:** measures the quality of a denoiser f, using the full knowledge of $\pi_{y,x}$ (ideal). Es. $L(f) = \mathbb{E}_{(y,x)\sim\pi_{(y,x)}}[\ell(f(y), x)]$.
- **Empirical risk, $\hat{L}$:** measures the quality of a denoiser f, using a supervised dataset. Es. $\hat{L}(f) = \frac{1}{m}\sum_{i=1}^{N} \ell(f(\mathbf{y}_i), \mathbf{x}_i)$.

The risk of overfitting



Minimizing $\hat{L}$ among all possible $f\colon Y \to X$ is not a good idea:

- the result f might work poorly on new data $y_{m+1}, y_{m+2}, \ldots$
- the result f might be unstable ($\|f\| \gg 1$)

Explicit regularization

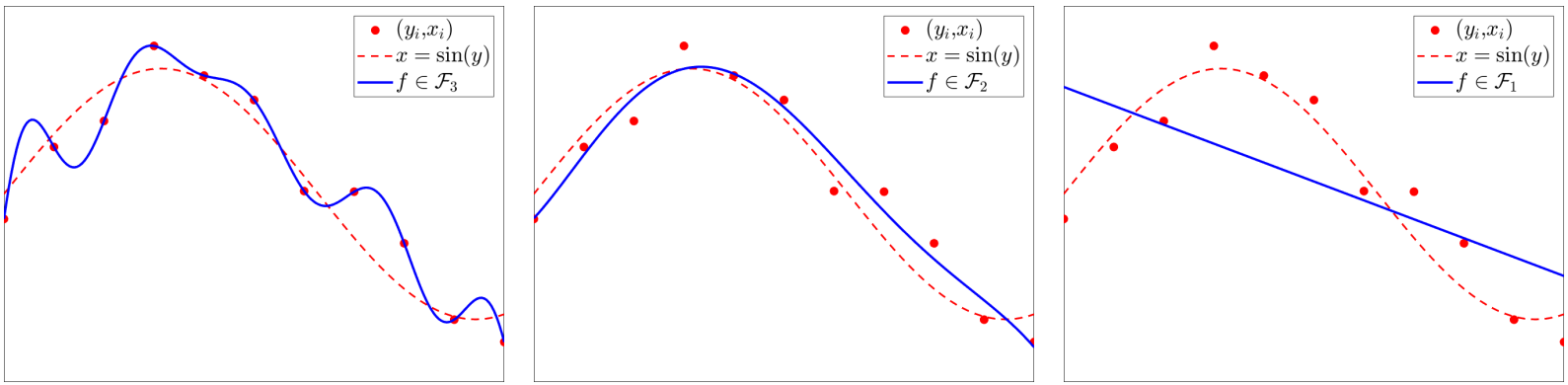
One possible solution: minimize $\hat{L}(f) + R(f)$.
Ex. $R(f) = \|f\|_{\mathcal{H}}$, $\mathcal{H}$ a suitable function space (e.g. a RKHS $\rightsquigarrow$ kernel methods).

A statistical learning perspective on image denoising

C. Parametric hypothesis classes

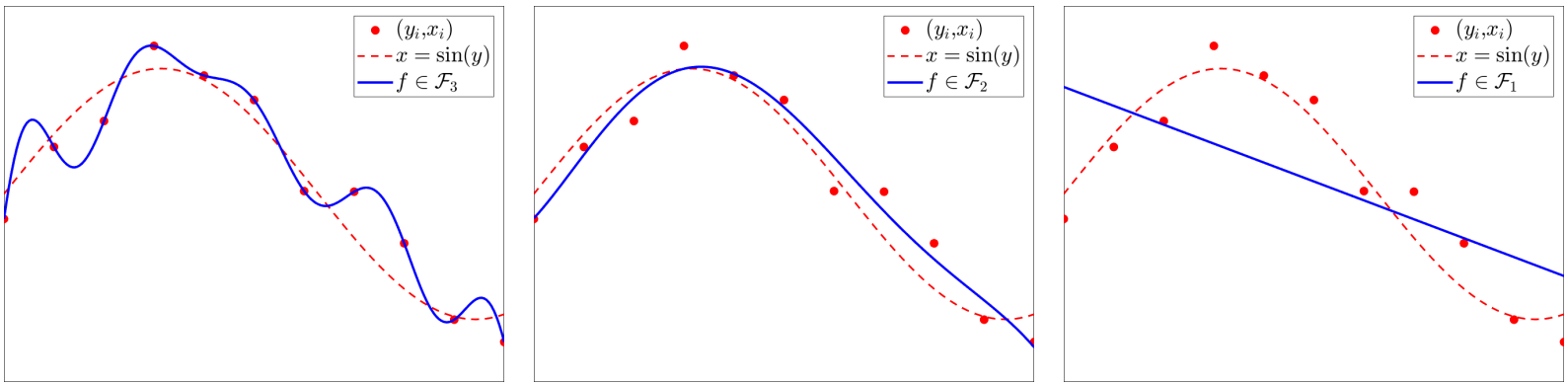
Implicit regularization via parametric hypothesis spaces

Idea: restricting the space $\mathcal{F}$ in which optimizing $\hat{L}$ induces an implicit regularization. Ex: $\mathcal{F} =$ polynomials of degree $10,\ 5,\ 1$.



Implicit regularization via parametric hypothesis spaces

Idea: restricting the space $\mathcal{F}$ in which optimizing $\hat{L}$ induces an implicit regularization. Ex: $\mathcal{F} =$ polynomials of degree 10, 5, 1.



Parametric hypothesis spaces

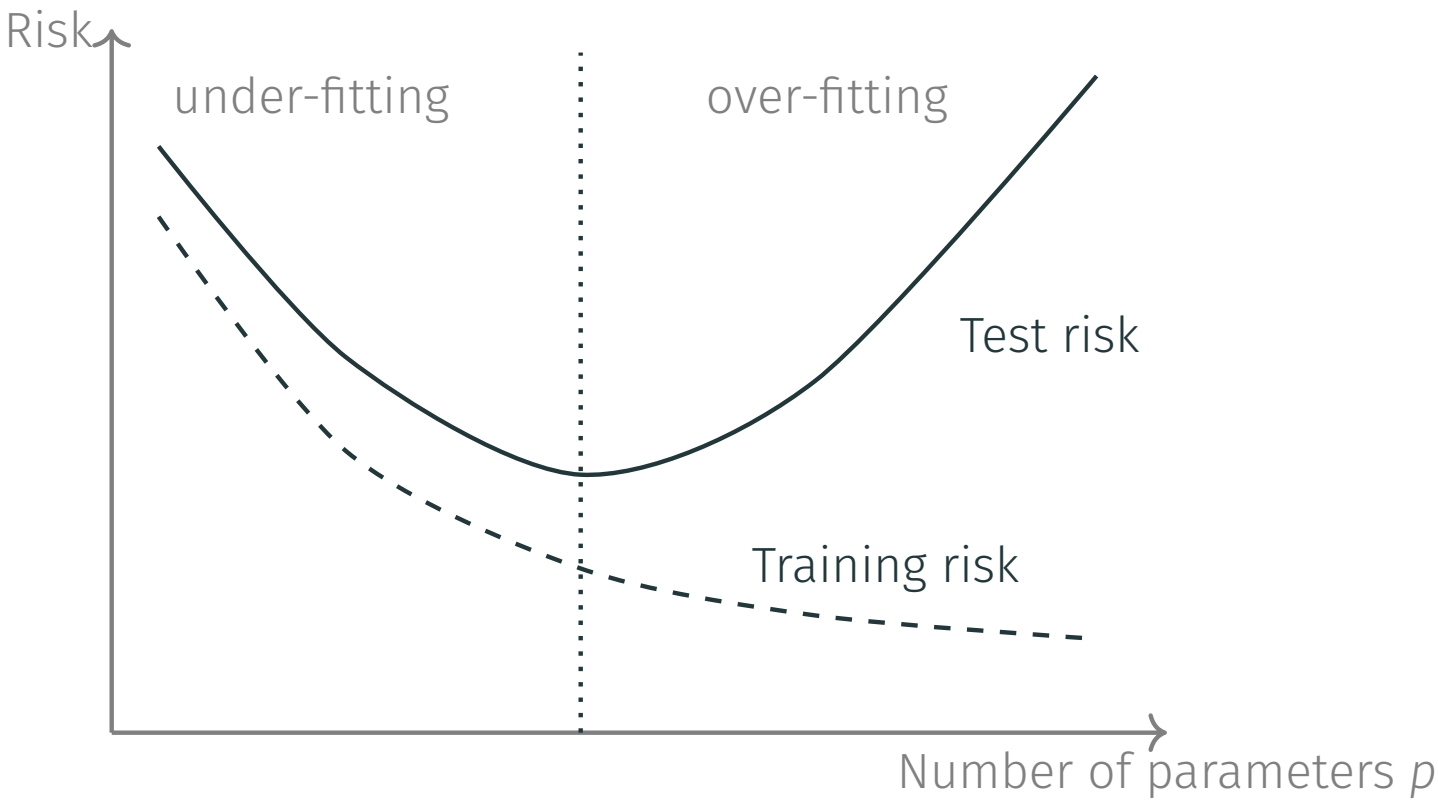
Spaces of functions depending on (p) parameters:

$\mathcal{F} = \{f_\theta : Y \to X, \quad \theta \in \Theta \cong \mathbb{R}^p\}$. Examples $(X = Y = \mathbb{R})$:

- polynomials: $f_\theta(y) = \theta_1 + \theta_2 y + \ldots + \theta_p y^{p-1}$;
- linear splines: $f_\theta(y) = \theta_1 + \theta_2 \chi_{[\theta_3, +\infty)}(y) + \ldots \theta_{p-1} \chi_{[\theta_p, +\infty)}(y)$;
- Neural Networks, e.g. $f_\theta(y) = \theta_1 \sigma(\theta_2 y) + \ldots + \theta_{p-1} \sigma(\theta_p y)$.

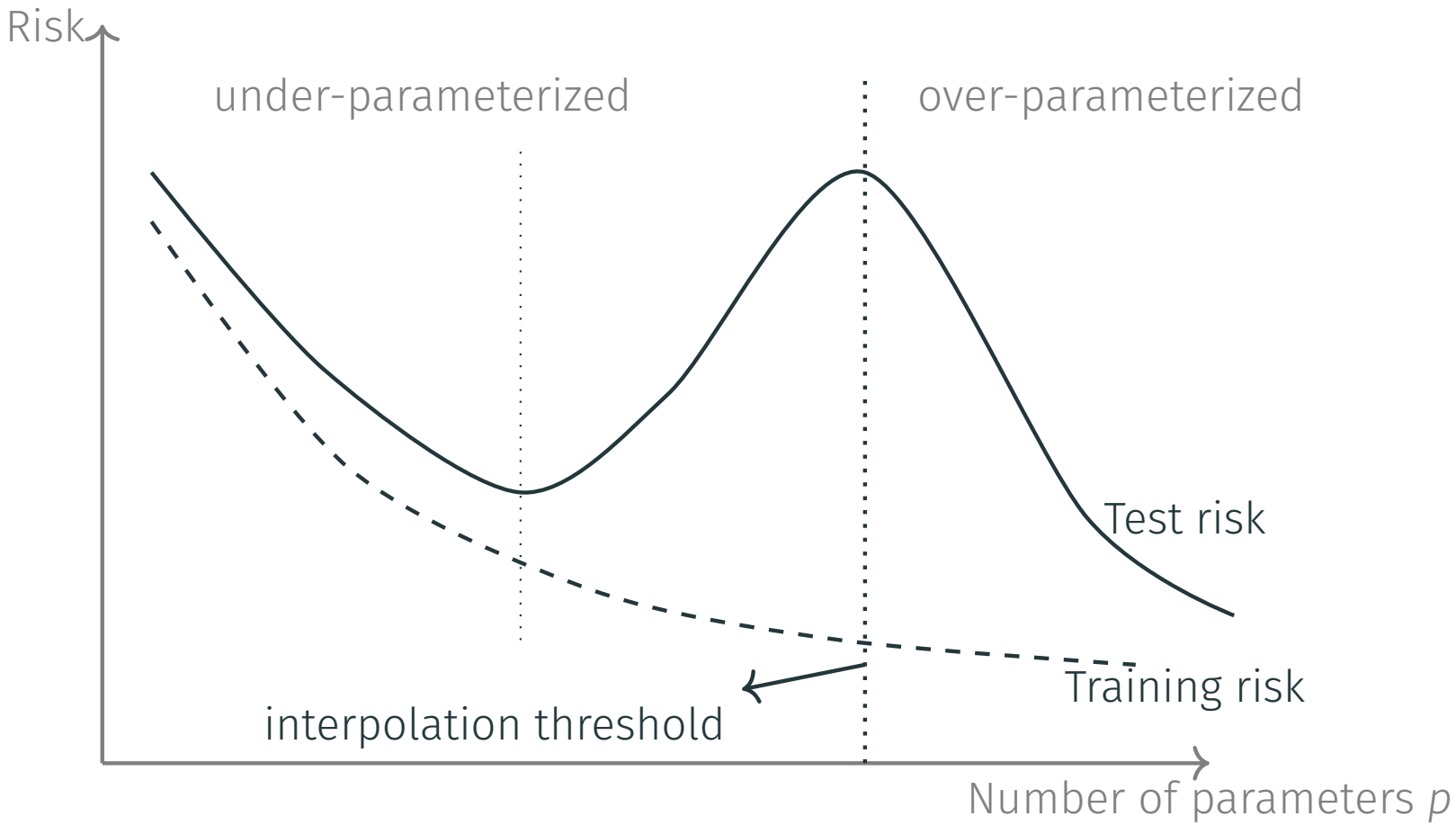
A bias-variance tradeoff

A variation of a picture from [M. Belkina, D. Hsuc, S. Maa, S. Mandal, *Reconciling modern machine-learning practice and the classical bias–variance trade-off*, 2019] - part 1



A bias-variance tradeoff

A variation of a picture from [M. Belkina, D. Hsuc, S. Maa, S. Mandal, *Reconciling modern machine-learning practice and the classical bias–variance trade-off*, 2019] - part 2



Neural Networks: a working expression

Given a nonlinear function $\sigma : \mathbb{R} \to \mathbb{R}$ and its element-wise version s.t. $[\boldsymbol{\sigma}(\mathbf{x})]_j = \sigma([\mathbf{x}]_j)$, given $W_l \in \mathbb{R}^{n_{l-1} \times n_l}$ and $\mathbf{b}_l \in \mathbb{R}^{n_l}$, $l = 1, \dots, L$, let

$$f_\theta(x) = W_{L+1}\boldsymbol{\sigma}(W_L \cdots \boldsymbol{\sigma}(W_1\mathbf{x} + \mathbf{b}_1) \cdots + \mathbf{b}_L) + \mathbf{b}_{L+1}.$$

The learnable parameters are $\theta = \{W_1, \dots, W_L, \mathbf{b}_1, \dots, \mathbf{b}_L\}$.

Neural Networks: very expressive parametric functions from $\mathbb{R}^n \to \mathbb{R}^n$

Neural Networks: a working expression

Given a nonlinear function $\sigma : \mathbb{R} \to \mathbb{R}$ and its element-wise version s.t. $[\boldsymbol{\sigma}(\mathbf{x})]_j = \sigma([\mathbf{x}]_j)$, given $W_l \in \mathbb{R}^{n_{l-1} \times n_l}$ and $\mathbf{b}_l \in \mathbb{R}^{n_l}$, $l = 1, \dots, L$, let

$$f_\theta(x) = W_{L+1}\boldsymbol{\sigma}(W_L \cdots \boldsymbol{\sigma}(W_1\mathbf{x} + \mathbf{b}_1) \cdots + \mathbf{b}_L) + \mathbf{b}_{L+1}.$$

The learnable parameters are $\theta = \{W_1, \dots, W_L, \mathbf{b}_1, \dots, \mathbf{b}_L\}$.

Examples:

- **Multilayer Perceptron (MLP)**: fully connected layers (full matrices W_l), $\sigma =$ ReLU, tanh - target: vector data.
- **Convolutional Neural Network (CNN)**: convolutional layers ($W_l\mathbf{x} = \mathbf{K}_l * \mathbf{X}$) - target: image data.

Neural Networks: very expressive parametric functions from $\mathbb{R}^n \to \mathbb{R}^n$

Neural Networks: a working expression

Given a nonlinear function $\sigma : \mathbb{R} \to \mathbb{R}$ and its element-wise version s.t. $[\boldsymbol{\sigma}(\mathbf{x})]_j = \sigma([\mathbf{x}]_j)$, given $W_l \in \mathbb{R}^{n_{l-1} \times n_l}$ and $\mathbf{b}_l \in \mathbb{R}^{n_l}$, $l = 1, \dots, L$, let

$$f_\theta(x) = W_{L+1}\boldsymbol{\sigma}(W_L \cdots \boldsymbol{\sigma}(W_1\mathbf{x} + \mathbf{b}_1) \cdots + \mathbf{b}_L) + \mathbf{b}_{L+1}.$$

The learnable parameters are $\theta = \{W_1, \dots, W_L, \mathbf{b}_1, \dots, \mathbf{b}_L\}$.

Examples:

- **Multilayer Perceptron (MLP)**: fully connected layers (full matrices W_l), $\sigma =$ ReLU, tanh - target: vector data.
- **Convolutional Neural Network (CNN)**: convolutional layers ($W_l\mathbf{x} = \mathbf{K}_l * \mathbf{X}$) - target: image data.

Theorem (Universal Approximation):

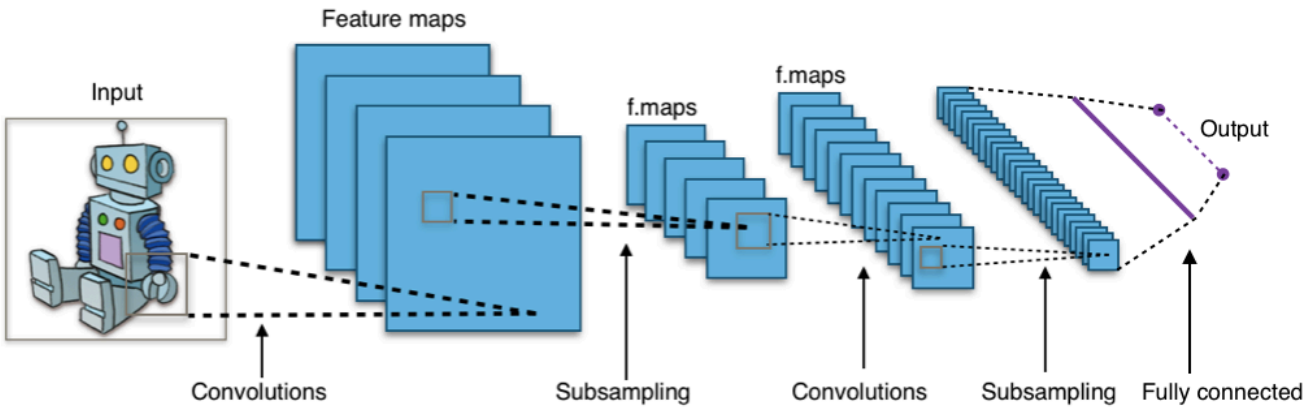
Under reasonable assumptions on σ, any continuous function $f : [0,1]^n \to \mathbb{R}^n$ can be approximated arbitrarily well (in the L^∞ norm) by a NN with $L = 1$, provided that $W_1 \in \mathbb{R}^{n_1 \times n}$ and $W_2 \in \mathbb{R}^{n \times n_1}$, and n_1 is sufficiently large. *(Cybenko, 1989; Hornik, 1991)*

Variants: arbitrary width, higher regularity, avoid the curse of dimensionality.

Network architectures for images: Convolutional Neural Networks (CNNs)

Key Idea: exploit spatial locality and translational invariance using local linear operations.

- Layers apply convolutions: $X \mapsto \sigma(K * X + b)$;
- Common operations: ReLU, pooling, normalization;
- Weight sharing between layers (parameter reduction).

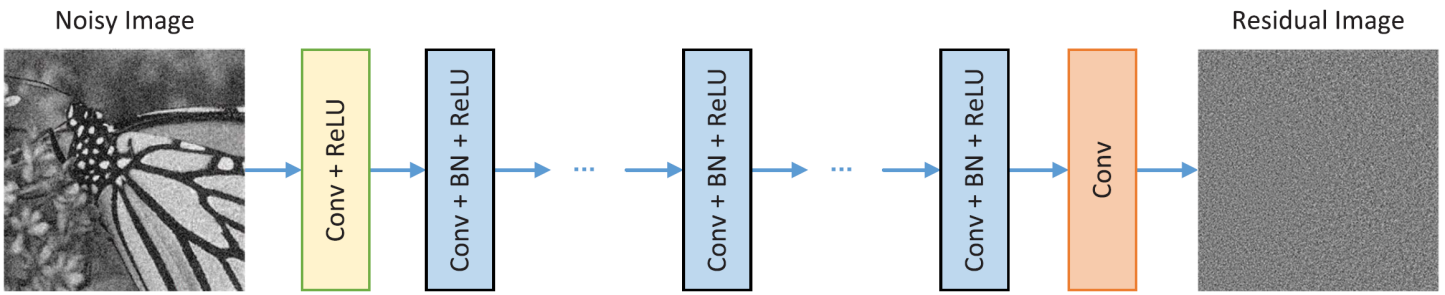


Credits: Wikipedia, by Aphex34

Network architectures for images: Convolutional Neural Networks (CNNs)

Key Idea: exploit spatial locality and translational invariance using local linear operations.

- Layers apply convolutions: $\mathbf{X} \mapsto \sigma(\mathbf{K} * \mathbf{X} + \mathbf{b})$;
- Common operations: ReLU, pooling, normalization;
- Weight sharing between layers (parameter reduction).



Credits: [Zhang et al., 2017]

Image denoising example:

- **DnCNN** (Zhang et al., 2017): deep CNN trained to remove additive Gaussian noise.

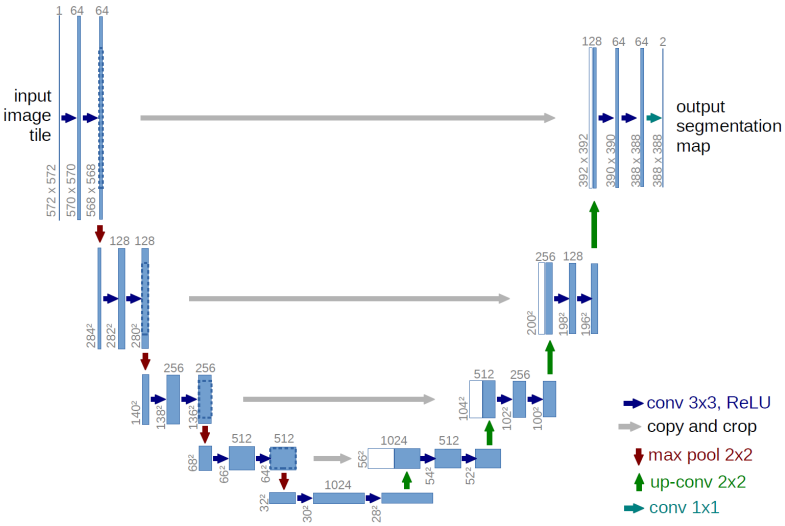
Warning: input dimensions!

In these examples, the input of the network is not a vectorized image $\mathbf{x} \in \mathbb{R}^n$ but a tensor $\mathbf{X}$ of size: height $\times$ width $\times$ channels.

Network architectures for images: U-Nets

Key Idea: extract and process local features with symmetric skip connections.

- An encoder branch extracts coarse features (convolution + downsampling)
- A decoder branch reconstructs full-resolution output (upsampling)
- Skip connections copy feature maps to enhance spatial details

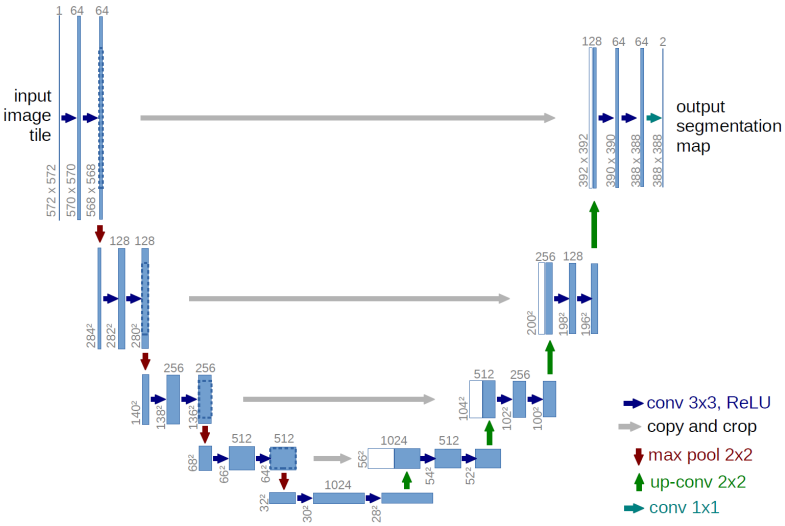


Credits: [Ronneberger, Fischer, Brox, 15]

Network architectures for images: U-Nets

Key Idea: extract and process local features with symmetric skip connections.

- An encoder branch extracts coarse features (convolution + downsampling)
- A decoder branch reconstructs full-resolution output (upsampling)
- Skip connections copy feature maps to enhance spatial details



Credits: [Ronneberger, Fischer, Brox, 15]

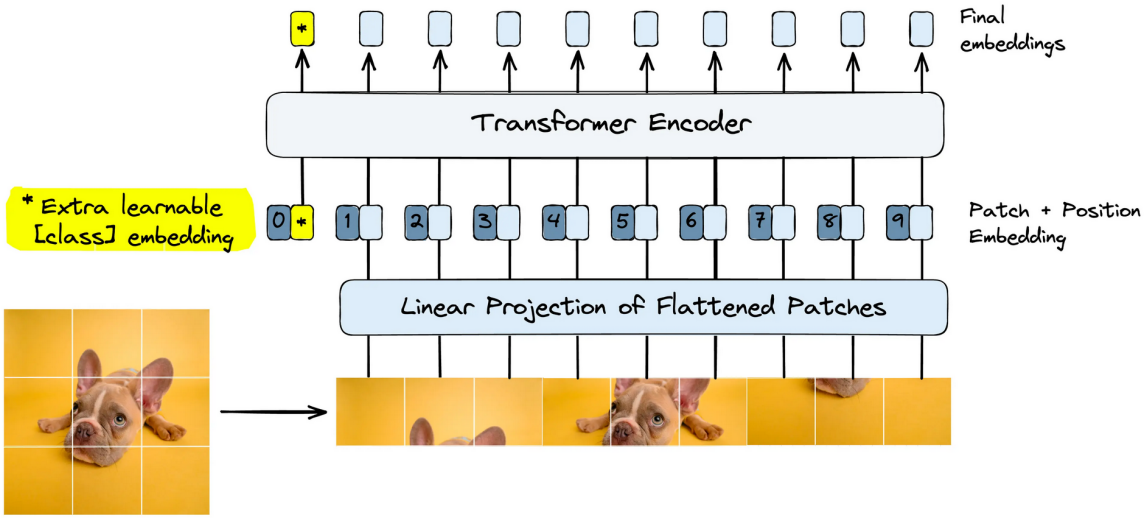
Image denoising example:

- U-Net variants are widely used in medical image denoising, see e.g. **Noise2Noise** (Lehtinen et al., 2018), **DRUNET** (Devalla et al., 2018), ...

Network architectures for images: Vision Transformers

Key Idea: Process images as sequences of patches with self-attention, removing convolutional inductive bias.

- Image is split into fixed-size patches (e.g., 16×16)
- Each patch is embedded into a vector (via linear projection)
- Transformer encoder applies global attention across all patches.



https://www.pinecone.io/learn/series/image-search/vision-transformers/

Image denoising example:

- **Restormer** (Zamir et al., 2022): uses self-attention over multi-resolution image representations; can be applied to unsupervised settings.

A statistical learning perspective on image denoising

D. Optimization algorithms

How to train your network - part I

Training a network: the process of finding θ that minimizes $\hat{\mathcal{L}}(\theta) = \hat{L}(f_\theta)$.

How to train your network - part I

Training a network: the process of finding θ that minimizes $\hat{\mathcal{L}}(\theta) = \hat{L}(f_\theta)$.
An **optimizer** is a numerical algorithm to do so. Key features:

1. first-order schemes: leverages derivatives of $\hat{\mathcal{L}}$ in θ;
2. stochastic optimization: it exploits random batches to reduce computations.

How to train your network - part I

Training a network: the process of finding θ that minimizes $\hat{\mathcal{L}}(\theta) = \hat{L}(f_\theta)$.
An **optimizer** is a numerical algorithm to do so. Key features:

1. first-order schemes: leverages derivatives of $\hat{\mathcal{L}}$ in θ;
2. stochastic optimization: it exploits random batches to reduce computations.

Backpropagation: just an intuition

- Exploit the compositional expression of f_θ: repeated chain rule.
- Use automated differentiation and zero-order methods.
- Efficiency: proportional to the forward pass (computing f_θ)

How to train your network - part I

Training a network: the process of finding θ that minimizes $\hat{\mathcal{L}}(\theta) = \hat{L}(f_\theta)$.
An **optimizer** is a numerical algorithm to do so. Key features:

1. first-order schemes: leverages derivatives of $\hat{\mathcal{L}}$ in θ;
2. stochastic optimization: it exploits random batches to reduce computations.

Backpropagation: just an intuition

- Exploit the compositional expression of f_θ: repeated chain rule.
- Use automated differentiation and zero-order methods.
- Efficiency: proportional to the forward pass (computing f_θ)

Stochastic Gradient Descent: just an intuition

- Exploit that $\hat{\mathcal{L}}(\theta) = \sum_{i=1}^{N} \ell(f_\theta(\mathbf{y}_i), \mathbf{x}_i)$: replace the full sum with the one on a randomly subsampled batch.
- Combined with momentum, acceleration, and adaptive step sizes, it provides efficient versions (Adam, AdaGrad, RMSProp)

How to train your network - part II

Other keywords involved in the training process of a network:

How to train your network - part II

Other keywords involved in the training process of a network:

Initialization:

[What?] Weights are initialized randomly (some methods: Xavier, He).

[Why?] Avoid symmetries, preserve variance across layers.

How to train your network - part II

Other keywords involved in the training process of a network:

Initialization:

[What?] Weights are initialized randomly (some methods: Xavier, He).

[Why?] Avoid symmetries, preserve variance across layers.

Scheduling:

[What?] Dynamically adjusts the learning rate during training.

[Why?] Crucial for stable convergence and escaping local minima.

How to train your network - part II

Other keywords involved in the training process of a network:

Initialization:

[What?] Weights are initialized randomly (some methods: Xavier, He).

[Why?] Avoid symmetries, preserve variance across layers.

Scheduling:

[What?] Dynamically adjusts the learning rate during training.

[Why?] Crucial for stable convergence and escaping local minima.

Early Stopping:

[What?] Interrupt training when the performance no longer improves.

[Why?] It prevents overfitting and reduces unnecessary computation.

How to train your network - part II

Other keywords involved in the training process of a network:

Initialization:

[What?] Weights are initialized randomly (some methods: Xavier, He).

[Why?] Avoid symmetries, preserve variance across layers.

Scheduling:

[What?] Dynamically adjusts the learning rate during training.

[Why?] Crucial for stable convergence and escaping local minima.

Early Stopping:

[What?] Interrupt training when the performance no longer improves.

[Why?] It prevents overfitting and reduces unnecessary computation.

Validation set:

[What?] A separate dataset used to evaluate generalization.

[Why?] Hyperparameter tuning (parameters of the network - number of layers, channels - or of the optimizer - learning rate, batch size).

Learning, from the errors

Training outcome: $\widetilde{\theta}$ $(\rightsquigarrow f_{\widetilde{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ
through a numerical method

Learning, from the errors

Training outcome: $\widetilde{\theta}$ $(\rightsquigarrow f_{\widetilde{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ
through a numerical method

Empirical target: $\widehat{\theta}$ $(\rightsquigarrow f_{\widehat{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ

Learning, from the errors

Training outcome: $\widetilde{\theta}\ (\rightsquigarrow f_{\widetilde{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ
through a numerical method

Empirical target: $\widehat{\theta}\ (\rightsquigarrow f_{\widehat{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ

Optimal target: $\theta^{\star}(\rightsquigarrow f_{\theta^{\star}})$
obtained by minimizing $\mathcal{L}(\theta) = L(f_{\theta})$ over Θ

Learning, from the errors

Training outcome: $\widetilde{\theta}\ (\rightsquigarrow f_{\widetilde{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ
through a numerical method

Empirical target: $\widehat{\theta}\ (\rightsquigarrow f_{\widehat{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ

Optimal target: $\theta^\star (\rightsquigarrow f_{\theta^\star})$
obtained by minimizing $\mathcal{L}(\theta) = L(f_\theta)$ over Θ

Bayes estimator: $f^\star = \mathbb{E}_\pi[x|y = \cdot]$
obtained by minimizing L on all
measurable functions $Y \to X$

Learning, from the errors

Training outcome: $\widetilde{\theta}\ (\rightsquigarrow f_{\widetilde{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ
through a numerical method

Empirical target: $\widehat{\theta}\ (\rightsquigarrow f_{\widehat{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ

Optimal target: $\theta^\star (\rightsquigarrow f_{\theta^\star})$
obtained by minimizing $\mathcal{L}(\theta) = L(f_\theta)$ over Θ

Bayes estimator: $f^\star = \mathbb{E}_\pi[x|y = \cdot]$
obtained by minimizing L on all
measurable functions $Y \to X$

True solution: if there existed a way
to connect $y \rightsquigarrow x$

Learning, from the errors

Training outcome: $\widetilde{\theta}\ (\rightsquigarrow f_{\widetilde{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ
through a numerical method

Empirical target: $\widehat{\theta}\ (\rightsquigarrow f_{\widehat{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ

Optimal target: $\theta^\star (\rightsquigarrow f_{\theta^\star})$
obtained by minimizing $\mathcal{L}(\theta) = L(f_\theta)$ over Θ

Bayes estimator: $f^\star = \mathbb{E}_\pi[x|y = \cdot]$
obtained by minimizing L on all
measurable functions $Y \to X$

True solution: if there existed a way
to connect $y \rightsquigarrow x$

Optimization error
(how good is my optimizer?)

Learning, from the errors

Training outcome: $\widetilde{\theta}\ (\rightsquigarrow f_{\widetilde{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ
through a numerical method

Empirical target: $\widehat{\theta}\ (\rightsquigarrow f_{\widehat{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ

Optimal target: $\theta^\star (\rightsquigarrow f_{\theta^\star})$
obtained by minimizing $\mathcal{L}(\theta) = L(f_\theta)$ over Θ

Bayes estimator: $f^\star = \mathbb{E}_\pi[x|y = \cdot]$
obtained by minimizing L on all
measurable functions $Y \to X$

True solution: if there existed a way
to connect $y \rightsquigarrow x$

Optimization error
(how good is my optimizer?)

Sample error
(how much does my result
depend on the training sample?)

Learning, from the errors

Training outcome: $\widetilde{\theta}\ (\rightsquigarrow f_{\widetilde{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ
through a numerical method

Empirical target: $\widehat{\theta}\ (\rightsquigarrow f_{\widehat{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ

Optimal target: $\theta^\star (\rightsquigarrow f_{\theta^\star})$
obtained by minimizing $\mathcal{L}(\theta) = L(f_\theta)$ over Θ

Bayes estimator: $f^\star = \mathbb{E}_\pi[x|y = \cdot]$
obtained by minimizing L on all
measurable functions $Y \to X$

True solution: if there existed a way
to connect $y \rightsquigarrow x$

Optimization error
(how good is my optimizer?)

Sample error
(how much does my result
depend on the training sample?)

Approximation error
(how expressive is my parametric
space of regularizers?)

Learning, from the errors

Training outcome: $\widetilde{\theta}\ (\rightsquigarrow f_{\widetilde{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ
through a numerical method

Empirical target: $\widehat{\theta}\ (\rightsquigarrow f_{\widehat{\theta}})$
obtained by minimizing $\widehat{\mathcal{L}}(\theta)$ over Θ

Optimal target: $\theta^\star (\rightsquigarrow f_{\theta^\star})$
obtained by minimizing $\mathcal{L}(\theta) = L(f_\theta)$ over Θ

Bayes estimator: $f^\star = \mathbb{E}_\pi[x|y = \cdot]$
obtained by minimizing L on all
measurable functions $Y \to X$

True solution: if there existed a way
to connect $y \rightsquigarrow x$

Optimization error
(how good is my optimizer?)

Sample error
(how much does my result
depend on the training sample?)

Approximation error
(how expressive is my parametric
space of regularizers?)

Irreducible error
(if data are noisy,
I can't reduce this!)

Implementation aspects

Image processing in Python

The most important libraries for image processing:

- **numpy**: process tensors representing images - can handle arithmetic operations, cropping...
- **skimage**: basic image processing operations (thresholding, blurring,...) and metrics (MSE, PSNR, SSIM,...)
- **matplotlib**: image visualization, color adjustment,...
- **pytorch**: neural network - definition, training, testing.

A small tutorial: tomorrow!