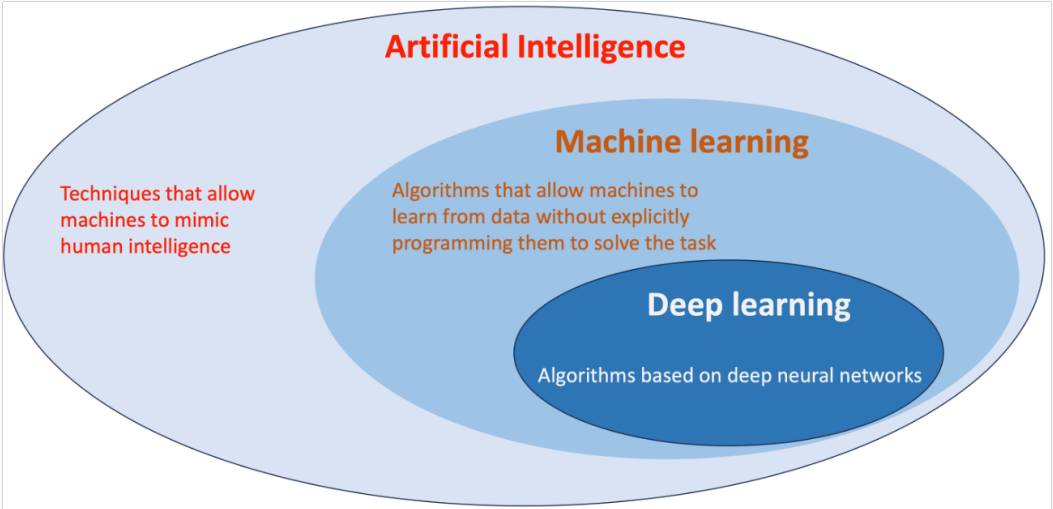
Deep Learning for System Identification

Introduction

Marco Forgione

IDSIA USI-SUPSI, Lugano, Switzerland

Artificial Intelligence, Machine Learning, Deep Learning

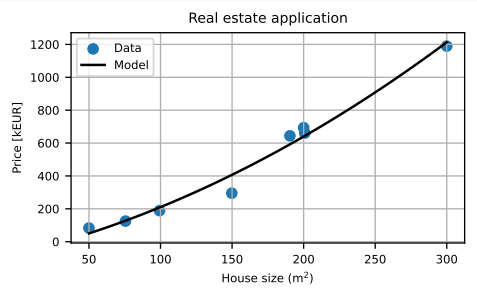


Machine learning: Regression

Nothing really special...

- Dataset: $D = \{(x_i, y_i)\}_{i=1}^{N}$
- Model structure: $\hat{y} = M(x; \theta)$
- Loss: $\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - \hat{y}_i(\theta)\|^2$

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta)$$

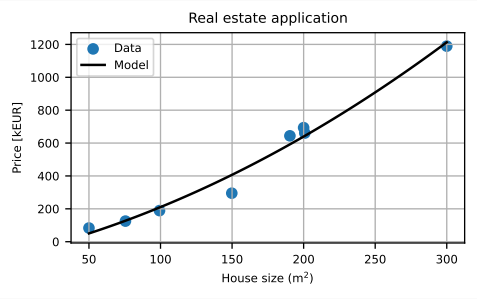


Machine learning: Regression

Nothing really special...

- Dataset: $D = \{(x_i, y_i)\}_{i=1}^{N}$
- Model structure: $\hat{y} = M(x; \theta)$
- Loss: $\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - \hat{y}_i(\theta)\|^2$

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta)$$

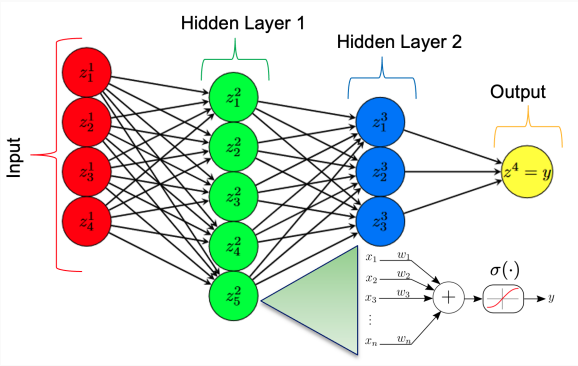


In deep learning:

- Expressive model structures like **neural networks**
- Iterative optimization, often gradient-based
- Efficient software with support for **automatic differentiation**

Is it just a form of *glorified curve fitting*?

Feedforward Neural Networks

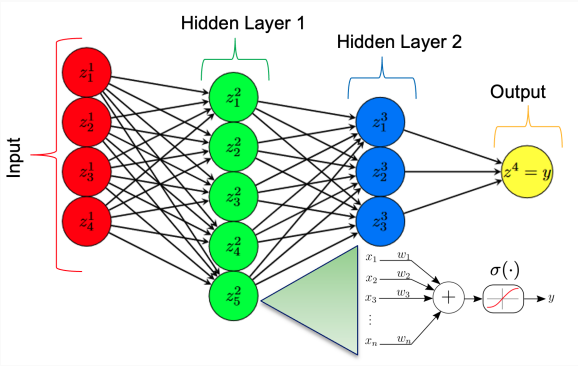


$$z_1^2 = \sigma\left(\sum_{j=1}^{4} w_{1,j}^1 z_j^1 + b_1^1\right)$$

$$z_2^3 = \sigma\left(\sum_{j=1}^{5} w_{2,j}^2 z_j^2 + b_2^2\right)$$

$$y = z^4 = \sum_{j=1}^{3} w_{1,j}^3 z_j^3 + b^3$$

Feedforward Neural Networks



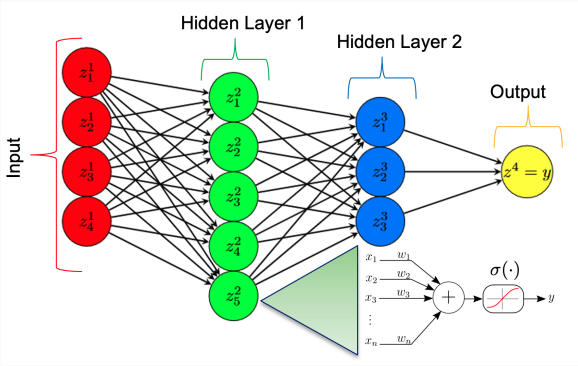
$$z_1^2 = \sigma\left(\sum_{j=1}^{4} w_{1,j}^1 z_j^1 + b_1^1\right)$$

$$z_2^3 = \sigma\left(\sum_{j=1}^{5} w_{2,j}^2 z_j^2 + b_2^2\right)$$

$$y = z^4 = \sum_{j=1}^{3} w_{1,j}^3 z_j^3 + b^3$$

$$y = W_3\sigma(W_2\sigma(W_1 x + b_1) + b_2) + b_3 = \mathrm{FF}(x;\theta)$$

Feedforward Neural Networks



$$z_1^2 = \sigma\left(\sum_{j=1}^{4} w_{1,j}^1 z_j^1 + b_1^1\right)$$

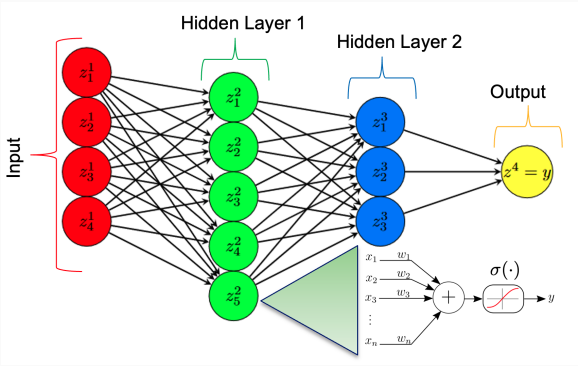
$$z_2^3 = \sigma\left(\sum_{j=1}^{5} w_{2,j}^2 z_j^2 + b_2^2\right)$$

$$y = z^4 = \sum_{j=1}^{3} w_{1,j}^3 z_j^3 + b^3$$

$$y = W_3\sigma(W_2\sigma(W_1 x + b_1) + b_2) + b_3 = \mathrm{FF}(x;\theta)$$

- Linear blocks interleaved by non-linear **activation functions** σ, e.g. $\tanh$

Feedforward Neural Networks



$$z_1^2 = \sigma\left(\sum_{j=1}^{4} w_{1,j}^1 z_j^1 + b_1^1\right)$$

$$z_2^3 = \sigma\left(\sum_{j=1}^{5} w_{2,j}^2 z_j^2 + b_2^2\right)$$

$$y = z^4 = \sum_{j=1}^{3} w_{1,j}^3 z_j^3 + b^3$$

$$y = W_3\sigma(W_2\sigma(W_1 x + b_1) + b_2) + b_3 = \mathrm{FF}(x;\theta)$$

- Linear blocks interleaved by non-linear **activation functions** σ, e.g. $\tanh$
- Non-linearities essential for **expressiveness**

Gradient Descent

1. Initialize: $\hat{\theta}$
2. **for** $k = 1, \dots, N$:
   - Compute Gradient: $g = \nabla_\theta \mathcal{L}(\theta^k)$
   - Update Parameters:

$$\hat{\theta} = \hat{\theta} - \gamma g$$

3. **end**

Gradient Descent

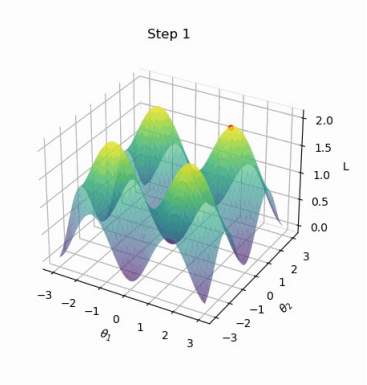
1. Initialize: $\hat{\theta}$
2. **for** $k = 1, \ldots, N$:
   - Compute Gradient: $g = \nabla_\theta \mathcal{L}(\theta^k)$
   - Update Parameters:

$$\hat{\theta} = \hat{\theta} - \gamma g$$

3. **end**

Local convergence to one of the **several minima** is OK!

Loss landscape



Gradient Descent

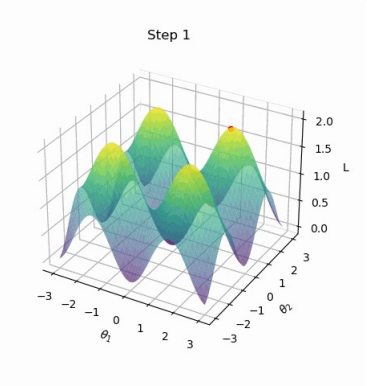
1. Initialize: $\hat{\theta}$
2. **for** $k = 1, \ldots, N$:
   - Compute Gradient: $g = \nabla_\theta \mathcal{L}(\theta^k)$
   - Update Parameters:

$$\hat{\theta} = \hat{\theta} - \gamma g$$

3. **end**

Local convergence to one of the **several minima** is OK!

Loss landscape



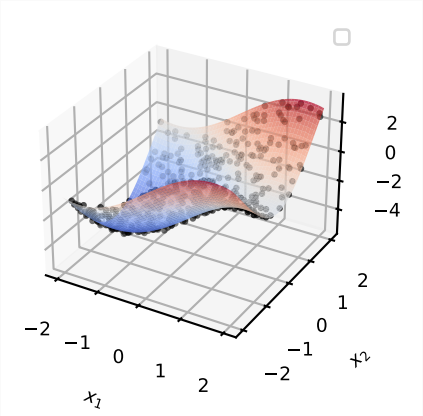
- Mini-batching: at each iteration, compute loss on a subset of the dataset
- Variants of plain gradient descent like `Adam` are more common and effective
- Second-order methods like (L)-BFGS also suitable for small/medium-scale problems

Example: synthetic toy dataset

Consider the 2D function $f : \mathbb{R}^2 \mapsto \mathbb{R}$

$$f(x) = 2\sin(x_1) - 3\cos(x_2)$$
$$x \in [-2, 2]^2 \subset \mathbb{R}^2$$

- Training and test datasets: 500 random points in the domain
- Additive noise with standard deviation 0.1



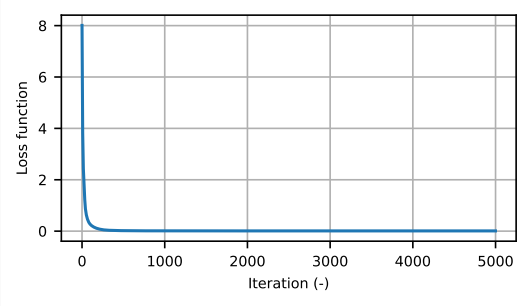
Feed-forward neural network

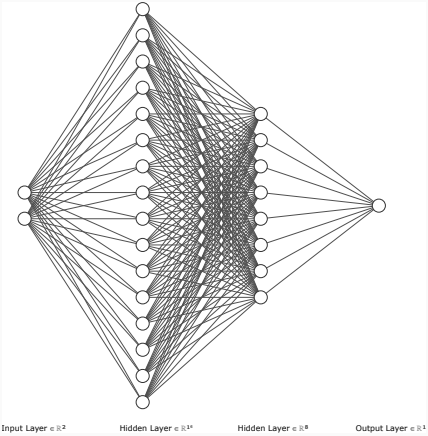
Model with:

- 2 inputs, 1 output - just like $f(x)$
- 2 hidden layers with $[16, 8]$ neurons
- $\tanh(\cdot)$ activation functions

$$\hat{y} = W_3 \tanh\left(W_2 \tanh(W_1 x + b_1) + b_2\right) + b_3$$

Training: 5000 iterations of `Adam`...



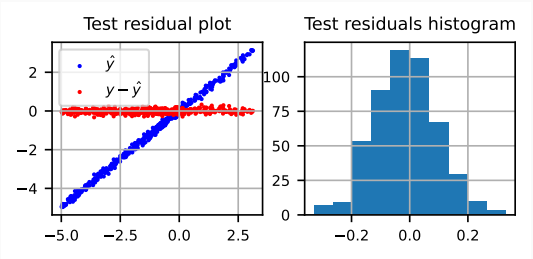


Model evaluation

Common to inspect on the test:

- Predictions and residuals vs measured output (left)
- Histogram of residuals (right)

Keep on following the good practices!



Model evaluation

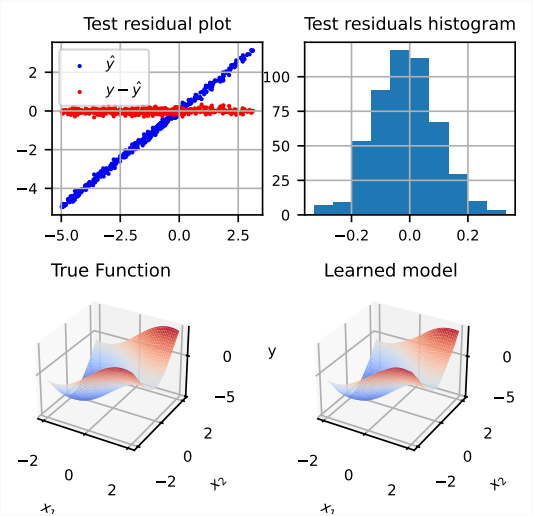
Common to inspect on the test:

- Predictions and residuals vs measured output (left)
- Histogram of residuals (right)

Keep on following the good practices!

2D toy example. We can also visualize:

- The known true function (right)
- The learned model over a grid (left)



Deep Learning For System Identification

- Feedforward nets fed by lagged input/outputs for NARX identification:

$$\hat{y}(k) = \mathrm{FF}(\overbrace{y(k-1), \ldots, y(k-n_a), u(k), u(k-1), \ldots, u(k-n_b-1)}^{=x(k)}; \theta)$$

Deep Learning For System Identification

- Feedforward nets fed by lagged input/outputs for NARX identification:

$$\hat{y}(k) = \mathrm{FF}(\overbrace{y(k-1), \ldots, y(k-n_a), u(k), u(k-1), \ldots, u(k-n_b-1)}^{=x(k)}; \theta)$$

- State-space models with neural-network updata/output maps:
  - Also known as Recurrent Neural Networks

$$x(k+1) = \mathrm{FF}_x(x(k), u(k); \theta)$$
$$y(k) = \mathrm{FF}_y(x(k); \theta)$$

What we won't cover

- Advanced architectures
  - physics-informed
  - grounded on system theory (e.g., stable by design)
  - transformers, diffusion, foundation models,...

What we won't cover

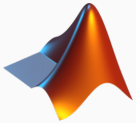
- Advanced architectures
  - physics-informed
  - grounded on system theory (e.g., stable by design)
  - transformers, diffusion, foundation models,...
- Advanced optimization
  - mini-batching (to handle large datasets)
  - second-order, non-smooth, constrained...
  - automatic differentiation details

What we won't cover

- Advanced architectures
  - physics-informed
  - grounded on system theory (e.g., stable by design)
  - transformers, diffusion, foundation models,...
- Advanced optimization
  - mini-batching (to handle large datasets)
  - second-order, non-smooth, constrained...
  - automatic differentiation details
- Theoretical aspects
  - learning theory
  - uncertainty quantification
  - ...

Software for Deep Learning

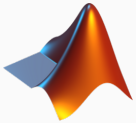
An essential aspect is the software implementation. A few options:

- PyTorch and JAX: libraries on top of Python
- Julia: a programming language focused on numerical computations
- MATLAB: you should know it already...

Software for Deep Learning

An essential aspect is the software implementation. A few options:

- PyTorch and JAX: libraries on top of Python
- Julia: a programming language focused on numerical computations
- MATLAB: you should know it already...

We will try out Google's **JAX**!

Some Literature

Machine learning reference books

- Murphy, Kevin P. *Probabilistic machine learning: An introduction.* MIT press, 2022.
- Murphy, Kevin P. *Probabilistic machine learning: Advanced topics.* MIT press, 2023

Deep learning in system identification (my biased perspective)

- Forgione, Marco, and Piga, Dario. "Continuous-time system identification with neural networks: Model structures and fitting criteria." *European Journal of Control* 59 (2021).
- Beintema, Gerben I., Maarten Schoukens, and Roland Tóth. "Deep subspace encoders for nonlinear system identification." *Automatica* (2023).
- Bemporad, Alberto. "An L-BFGS-B Approach for Linear and Nonlinear System Identification Under ℓ_1 and Group-Lasso Regularization." *IEEE Transactions on Automatic Control* (2025).
- Pillonetto, Gianluigi, Aleksandr Aravkin, Daniel Gedon, Lennart Ljung, Antonio H. Ribeiro, and Thomas B. Schön. "Deep networks for system identification: a survey." *Automatica* (2025).