Deep learning for System Identification

Introduction

Marco Forgione, Dalle Molle Institute for Artificial Intelligence, USI-SUPSI, Lugano, Switzerland

Milano, June 30, 2025

Slides originally prepared by:

Dario Piga, Dalle Molle Institute for Artificial Intelligence, Lugano, Switzerland

Introduction to System Identification

Models

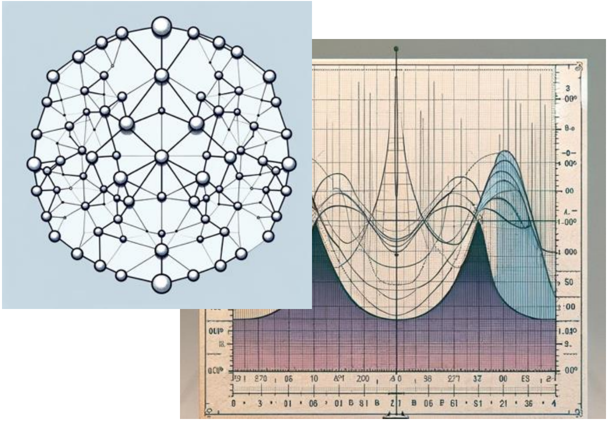
- Abstractions of reality

- Models can describe mechanical systems, social interactions, markets, etc.

- Different types of models:
  - Intuitive/mental models
  - Graphical models (Bode diagrams, flowcharts)
  - Mathematical models describing relations between variables/signals of the system/process

$$\dot{\mathbf{x}}(t) = f(x(t), u(t))$$

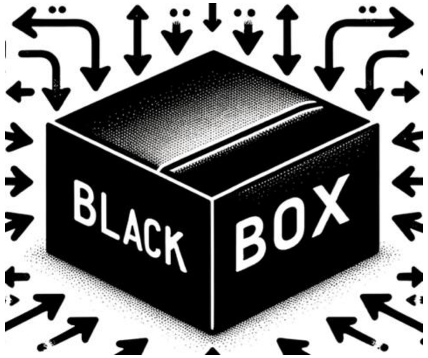
$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$



Modelling paradigms

- First principles modelling (from first principles laws of physics/chemistry/biology, etc.)
  - Pure physical modelling: **white-box approach**
  - Knowledge of the physics behind the system/process
  - Models are interpretable
  - Sometimes, physical modelling can be complex or even impossible

- Data-driven modelling (from measurements of variables gathered from experiments)
  - Pure data-driven modelling: **black-box approach**
  - Limited knowledge of the system/process
  - Models are usually not explainable
  - Need for (informative) data
  - Generalization properties might be limited



- Combination of first principles and data-driven modelling is also possible
  - E.g., model structure derived from physics, and some parameters estimated from data
  - In any case, even pure physical models needs often to be validated with experiments

Dynamical systems

- Systems with memory (present also depends on the past)

- Described by equations evolving in time

Continuous time: ODE

$$\begin{cases} \dot{x}(t) = f(x(t), u(t); \theta) \\ y(t) = g(x(t); \theta) \end{cases}$$

$$y^{(n)}(t) = h\left(y^{(n-1)}, \dots, y, u^{(n)}, \dots, u; \theta\right)$$

Discrete time: Recurrence equation

$$\begin{cases} x(k+1) & = f(x(k), u(k); \theta) \\ y(k) & = g(x(k); \theta) \end{cases}$$

$$y(k) = h\left(y(k-1), \dots, y(k-n_a), u(k), \dots, u(k-n_b); \theta\right)$$

Model structures

- Model: Given past inputs and outputs, provides an estimate of the current output
  - Example: AutoRegressive eXogenous (ARX) model

$$M(\theta): \hat{y}(k) = a_1 y(k-1) + \ldots + a_{na} y(k-na) + b_0 u(k) + \ldots + b_{nb} u(k-nb)$$
$$= \underbrace{[y(k-1) \ \ldots y(k-na) \ u(k) \ u(k-1) \ \ldots \ u(k-nb)]}_{\phi^\top(k)}\theta$$

$$\theta = [a_1 \quad a_2 \quad \ldots \quad a_{na} \quad b_0 \quad b_1 \quad \ldots \quad b_{nb}]^\top$$

- Residual signal: $\epsilon(k,\theta) = y(k) - \hat{y}(k)$
  - Example: $\epsilon(k,\theta) = y(k) - \phi^\top(k)\theta$

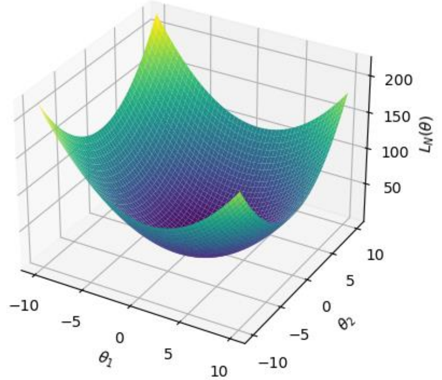
- Loss: $\mathcal{L}_N(\theta) = \frac{1}{N}\sum_{k=1}^{N} \ell(y(k), \hat{y}(k,\theta))$
  - Example: least-squares loss:

$$\mathcal{L}_N(\theta) = \frac{1}{N}\sum_{k=1}^{N} \|\epsilon(k,\theta)\|^2$$

Least squares estimate: ARX model structure

$$M(\theta): \hat{y}(k) = a_1 y(k-1) + \ldots + a_{na} y(k-na) + b_0 u(k) + \ldots + b_{nb} u(k-nb)$$
$$= \underbrace{[y(k-1) \ \ldots y(k-na) \ u(k) \ u(k-1) \ \ldots \ u(k-nb)]}_{\phi^\top(k)} \theta$$

$$\mathcal{L}_N(\theta) = \frac{1}{N} \sum_{k=1}^{N} \|\epsilon(k, \theta)\|^2 = \frac{1}{N} \sum_{k=1}^{N} \left(\phi^\top(k)\theta - y(k)\right)^2$$

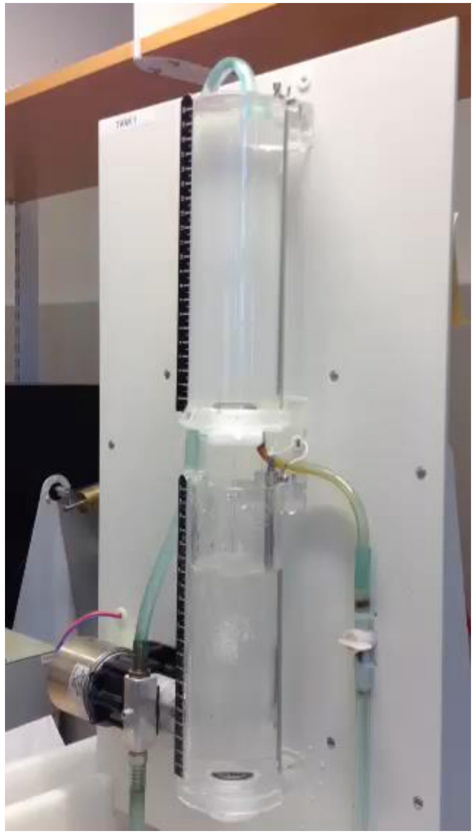


$$\theta^* = \arg\min_{\theta} \mathcal{L}_N(\theta) \rightarrow \theta^* : \frac{\partial L_N(\theta)}{\partial \theta} = 0$$

$$\theta^* : \frac{1}{N} \sum_{k=1}^{N} \phi(k) \left(\phi^\top(k)\theta - y(k)\right) = 0$$

$$\theta^* = \left(\frac{1}{N} \sum_{k=1}^{N} \phi(k)\phi^\top(k) \right)^{-1} \frac{1}{N} \sum_{k=1}^{N} \phi(k) y(k)$$

Example: Cascaded Tanks System



- Fluid level control system with two tanks
- Upper tank fed by a pump
- Water flows from upper tank to lower tank through a small opening
- Water flows from lower tank to reservoir through a small opening
- Overflow in the upper tank may happen

- Input: pump voltage
- Output: water level of lower tank
- Length of training set: 1024 samples
- Length of test set: 1024 samples
- Sampling time: 4 s

Jupyter Notebook: cascaded_tanks (Part 1)

- Metrics (over test data) both for 1-step and open-loop simulation:

$$\mathrm{MSE} = \frac{1}{N} \sum_{k=1}^{N} \|y(k) - \hat{y}(k)\|^2 \qquad R^2 = 1 - \frac{\sum_{k=1}^{N} \|y(k) - \hat{y}(k)\|^2}{\sum_{k=1}^{N} \|y(k) - \bar{y}\|^2}$$

Schoukens, M. et al. "Cascaded tanks benchmark combining soft and hard nonlinearities." Workshop on nonlinear system identification benchmarks, 2016.
https://www.nonlinearbenchmark.org/

Including nonlinearities

- Linear-in-the-parameter NARX model:

$$M(\theta) : \hat{y}(k) = F^{\top}\Bigg(\overbrace{y(k-1),\dots,y(k-n_a),u(k-1),\dots,u(k-n_b)}^{=\phi(k)}\Bigg)\,\theta$$

- F is a vector function of ϕ containing (known) nonlinear elements, e.g.,

$$F(\phi(k)) = \begin{bmatrix} y(k-1) \\ \vdots \\ y^2(k-na) \\ u^3(k-1) \\ \vdots \\ u(k-1)\sin(u(k-nb)) \end{bmatrix}$$

How to choose the non-linearities?

- General NARX model:

- $\mathcal{F}$ is a nonlinear function (e.g., neural network) of past observations
- Linearity in θ is lost

$$M(\theta) : \hat{y}(k) = \mathcal{F}\big(y(k-1),\dots,y(k-na),u(k),u(k-1),\dots,u(k-nb);\theta\big)$$

Limitations of (N)ARX models

$$M(\theta) : \hat{y}(k) = \mathcal{F}\big(y(k-1), \ldots, y(k-na), u(k), u(k-1), \ldots, u(k-nb); \theta\big)$$

$$\epsilon(k, \theta) = y(k) - \hat{y}(k)$$

- Prediction error only looks one-step ahead

- In case of data sampled at high-frequency, a good one-step ahead predictor could be:

$$M(\theta) : \hat{y}(k) = y(k-1)$$

- We would like to have predictive models that provide good prediction in a long-horizon

$$M(\theta) : \hat{y}(k) = \mathcal{F}\big(\hat{y}(k-1), \ldots, \hat{y}(k-na), u(k), u(k-1), \ldots, u(k-nb); \theta\big)$$

$$k = 1, 2, \ldots$$

Simulation error minimization

- Linear models
  - Output Error (OE) model:

$$M(\theta): \hat{y}(k) = a_1 \hat{y}(k-1) + \ldots + a_{na}\hat{y}(k-na) + b_0 u(k) + \ldots + b_{nb} u(k-nb)$$
$$= \underbrace{\left[\hat{y}(k-1;\theta) \ \ldots \ \hat{y}(k-na;\theta) \ u(k) \ u(k-1) \ \ldots \ u(k-nb)\right]}_{\phi^\top(k;\theta)}\theta$$

- Residual signal: $\epsilon(k,\theta) = y(k) - \hat{y}(k) = y(k) - \phi^\top(k;\theta)\theta$

- Loss: sum of squared residuals is no longer convex!

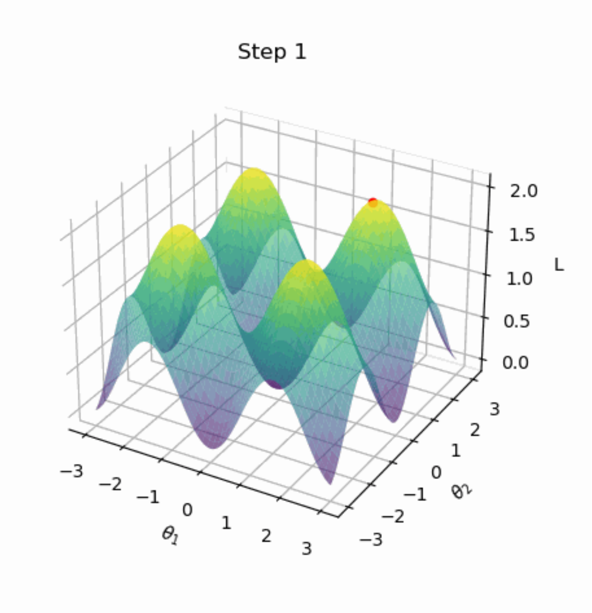
$$L_N(\theta) = \frac{1}{N}\sum_{k=1}^{N} \|\epsilon(k,\theta)\|^2$$

First-order methods for non-convex optimization

Vanilla Gradient Descent

1. **Initialize Parameters:** $\theta^{(0)}$
2. **for** $k = 0, 1, \ldots$: **until** maximum number of iterations or convergence **do**:
   (a) **Compute Gradient:** $\nabla_\theta \mathcal{L}(\theta^{(k)})$
   (b) **Update Parameters:**

$$\theta^{(k+1)} = \theta^{(k)} - \gamma \cdot \nabla_\theta \mathcal{L}(\theta^{(k)})$$



[Jupyter Notebook: cascaded_tanks.ipynb (Part II)]

Other updates are possible, e.g., gradient-descent with momentum:

$$v^{(k+1)} = \mu v^{(k)} + (1 - \tau) \cdot \nabla_\theta \mathcal{L}(\theta^{(k)})$$
$$\theta^{(k+1)} = \theta^{(k)} - \gamma v^{(k+1)}$$

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
optimizer = optim.Adam([var1, var2], lr=0.0001)
```

Convergence of gradient-descent

Assumptions:

- $L(\theta^*) = 0$ (just to keep notation simple)
- Lipschitz continuity of ∇L: $\nabla^2 L(\theta) \preceq \beta I \quad \forall \theta \in B$
- μ-PL condition: $\frac{1}{2} \|\nabla L(\theta)\|^2 \geq \mu L(\theta) \quad \forall \theta \in B$
- $\theta^{(k+1)} = \theta^{(k)} - \gamma \nabla L(\theta^{(k)})$ (gradient update)

Convergence:

$$
\begin{aligned}
L(\theta^{(k+1)}) &= L(\theta^{(k)}) + \left(\theta^{(k+1)} - \theta^{(k)}\right)^\top \nabla L(\theta^{(k)}) + \frac{1}{2}\left(\theta^{(k+1)} - \theta^{(k)}\right)^\top \nabla^2 L(\bar{\theta}) \left(\theta^{(k+1)} - \theta^{(k)}\right) \\
&\leq L(\theta^{(k)}) - \gamma \left\|\nabla L(\theta^{(k)})\right\|^2 + \frac{1}{2}\gamma^2 \left\|\nabla L(\theta^{(k)})\right\|^2 \beta \\
&= L(\theta^{(k)}) - \gamma \left\|\nabla L(\theta^{(k)})\right\|^2 \left(1 - \frac{1}{2}\gamma\beta\right) \\
&= L(\theta^{(k)}) - \gamma \frac{1}{2} \left\|\nabla L(\theta^{(k)})\right\|^2 \\
&\leq L(\theta^{(k)})(1 - \gamma\mu)
\end{aligned}
$$

Take $\gamma = \frac{1}{\beta}$

$$\boxed{L(\theta^{(k)}) \leq (1 - \gamma\mu)^k L(\theta^{(0)})}$$

Karimi, H., Nutini, J., & Schmidt, M. (2016, September). Linear convergence of gradient and proximal-gradient methods under the Polyak-Lojasiewicz condition. In *Joint European conference on machine learning and knowledge discovery in databases* (pp. 795-811). Cham: Springer International Publishing.

Stochastic Gradient Descent

$$\theta^{(k+1)} = \theta^{(k)} - \gamma \nabla \ell_{i_k}(\theta^{(k)})$$

Uniformly randomly sampled from {1,2,…N}

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell_i(\theta)$$

$$\nabla L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla \ell_i(\theta)$$

$$\mathbb{E}[\nabla \ell_{i_k}(\theta^{(k)})] = \sum_{i=1}^{N} \nabla \ell_i(\theta^{(k)}) P(i_k = i) = \frac{1}{N}\sum_{i=1}^{N} \nabla \ell_i(\theta^{(k)}) = \nabla L(\theta^{(k)})$$

Mini-batch gradient descent (B<<N):

$$\theta^{(k+1)} = \theta^{(k)} - \gamma \frac{1}{B}\sum_{j=1}^{B} \nabla \ell_{i_j}(\theta^{(k)})$$

1 iteration = 1 parameter update

1 epoch = $\frac{N}{B}$ parameter updates

Do not process the whole dataset to update the model parameters, just process a smaller batch of samples and then make the update

Bottou, L., Curtis, F. E., & Nocedal, J. (2018). Optimization methods for large-scale machine learning. SIAM review, 60(2), 223-311.

Enforcing sparsity: L1-regularization

$$\min_{\theta} \frac{1}{N} \sum_{k=1}^{N} \|\epsilon(k,\theta)\|^2 + \lambda \|\theta\|_1$$

- Linear-in-the-parameter NARX model:

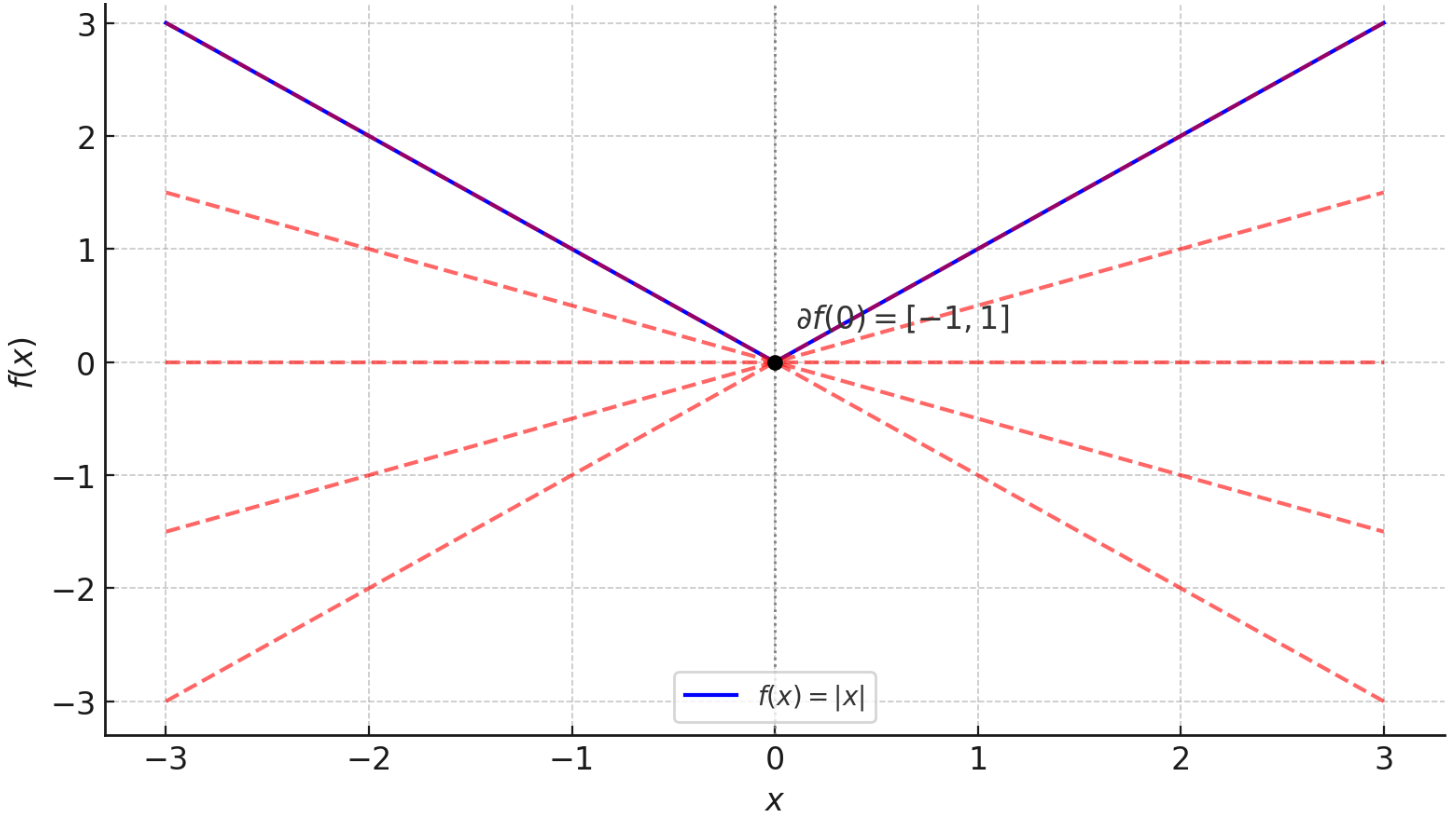
$$M(\theta): \hat{y}(k) = F^\top(y(k-1), \ldots, y(k-na), u(k), u(k-1), \ldots, u(k-nb))\theta$$

- L1 regularization (LASSO-like): penalize model complexity during training

$$\min_{\theta} \frac{1}{N} \sum_{k=1}^{N} \|y(k) - F^\top(\phi(k))\theta\|^2 + \lambda \|\theta\|_1$$

Non smooth functions

$$\min_{\theta} \mathcal{L}(\theta) + \lambda \|\theta\|_1 \longrightarrow 0 \in \nabla \mathcal{L}(\theta^\star) + \partial \|\lambda \cdot \theta^\star\|_1$$



Parikh, N., & Boyd, S. (2014). Proximal algorithms. Foundations and Trends® in Optimization, 1(3), 127-239.

Proximal gradient methods

non-differentiable convex funtion

$$\min_{\theta} \; \mathcal{L}(\theta) + R(\theta)$$

Proximal operator of function $R(\cdot)$ with parameter γ

$$\operatorname{prox}_{\gamma R}(v) := \arg\min_{\theta \in \Theta} \left\{ \frac{1}{2} \|\theta - v\|^2 + \gamma R(\theta) \right\}$$

$\theta^\star$ minimizer of $\mathcal{L}(\theta) + R(\theta)$ $\longrightarrow$ $0 \in \nabla \mathcal{L}(\theta^\star) + \partial R(\theta^\star)$

$$\Updownarrow$$

Fixed point equation

$$\boxed{\theta^{(k+1)} = \operatorname{prox}_{\gamma R}\left(\theta^{(k)} - \gamma \nabla \mathcal{L}(\theta^{(k)})\right)}$$

$$\theta^\star = \operatorname{prox}_{\gamma R}\left(\theta^\star - \gamma \nabla \mathcal{L}(\theta^\star)\right)$$

Fixed point equation: proof of stationarity

We only prove in one direction:

$$\theta^* = \mathrm{prox}_{\gamma R}(\theta^* - \gamma \nabla \mathcal{L}(\theta^*)) \to 0 \in \nabla \mathcal{L}(\theta^*) + \partial R(\theta^*)$$

- By definition of the proximal operator:

$$\theta^* = \mathrm{prox}_{\gamma R}(\theta^* - \gamma \nabla \mathcal{L}(\theta^*)) = \arg\min_{\theta} \frac{1}{2\gamma} \left\| \theta - \left(\theta^* - \gamma \nabla \mathcal{L}(\theta^*)\right) \right\|^2 + R(\theta)$$

- The minimized function is convex: smooth quadratic + non-smooth yet convex term. Its optimality condition is:

$$0 \in \frac{1}{\gamma} \left(\theta - \theta^* + \gamma \nabla \mathcal{L}(\theta^*)\right)\Big|_{\theta = \theta^*} + \partial R(\theta^*)$$

$$0 \in \nabla \mathcal{L}(\theta^*) + \partial R(\theta^*)$$

Proximal gradient methods: constrained optimization

$$\min_{\theta \in C \subseteq \mathbb{R}^d} \mathcal{L}(\theta) \longrightarrow \min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta) + \mathbb{I}_C(\theta) \qquad \mathbb{I}_C(\theta) = \begin{cases} 0 & \text{if } \theta \in C \\ +\infty & \text{if } \theta \notin C \end{cases}$$

Example: $R(\theta)$ indicator function $\mathbb{I}_C$ of a convex $C \subseteq \mathbb{R}^d$:

$$\begin{aligned} \operatorname{prox}_{\mathbb{I}_C}(v) &= \arg\min_{\theta \in \mathbb{R}^d} \left\{ \frac{1}{2}\|\theta - v\|^2 + \mathbb{I}_C(\theta) \right\} \\ &= \arg\min_{\theta \in C} \|\theta - v\|^2 \\ &= \Pi_C(v) \end{aligned}$$

orthogonal projection

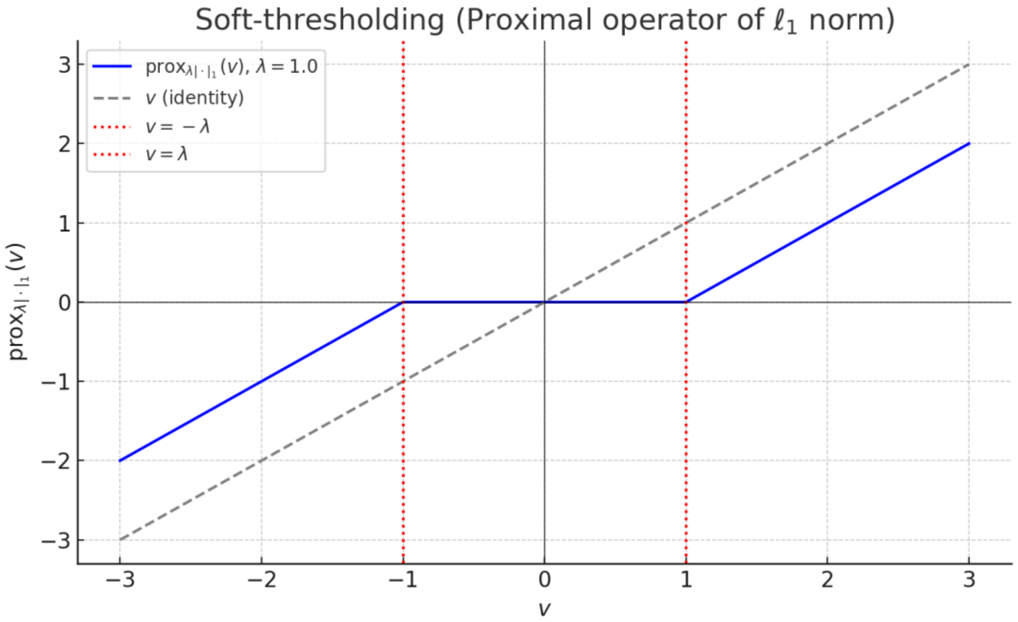
This leads to the *projected* gradient descent algorithm:

$$\theta^{(k+1)} = \Pi_C(\theta^{(k)} - \lambda \nabla \mathcal{L}(\theta^{(k)}))$$

Proximal gradient methods: L1 regularization

$$\min_{\theta} \mathcal{L}(\theta) + \lambda \|\theta\|_1 \qquad \operatorname{prox}_{\lambda\|\cdot\|_1}(v) = \arg\min_{\theta\in\mathbb{R}^d} \left\{ \frac{1}{2}\|\theta - v\|^2 + \lambda\|\theta\|_1 \right\}$$

$$= \left[\operatorname{prox}_{\lambda\|\cdot\|_1}(v)\right]_i = \operatorname{sign}(v_i)\cdot\max\left(|v_i| - \lambda, 0\right)$$



$$\theta^{(k+1)} = \operatorname{SoftThreshold}(\theta^{(k)} - \lambda\nabla\mathcal{L}(\theta^{(k)}))$$

Proximal gradient methods: algorithm L1-regularization

$$\min_{\theta} \; \mathcal{L}(\theta) + \lambda \|\theta\|_1$$

$$\operatorname{prox}_{\lambda\|\cdot\|_1}(v) = \arg\min_{\theta \in \mathbb{R}^d} \left\{ \frac{1}{2}\|\theta - v\|^2 + \lambda \|\theta\|_1 \right\}$$

$$= \left[\operatorname{prox}_{\lambda\|\cdot\|_1}(v)\right]_i = \operatorname{sign}(v_i) \cdot \max\left(|v_i| - \lambda, 0\right)$$

$$\theta^{(k+1)} = \operatorname{prox}_{\gamma R}\left(\theta^{(k)} - \gamma \nabla \mathcal{L}(\theta^{(k)})\right)$$

Algorithm 1 Proximal Gradient Method (LASSO)

1: **Input:** differentiable loss $\mathcal{L}(\theta)$, regularization parameter $\lambda > 0$, step size γ, initial $\theta^{(0)}$
2: **for** $k = 0$ to N_{iter} **do**
3: $\quad g^{(k)} \leftarrow \nabla \mathcal{L}(\theta^{(k)})$
4: $\quad z^{(k)} \leftarrow \theta^{(k)} - \gamma g^{(k)}$
5: $\quad \theta^{(k+1)} \leftarrow \text{SoftThreshold}_{\gamma\lambda}(z^{(k)})$
6: **end for**
7: **Return:** $\theta^{(N_{\text{iter}})}$

Jupyter Notebook: cascaded_tanks.ipynb (Part III)