



Deep learning for System Identification

Introduction

Marco Forgone, Dalle Molle Institute for Artificial Intelligence, Lugano, Switzerland

Milano, June 30, 2025

Slides originally prepared by:

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

Introduction to System Identification

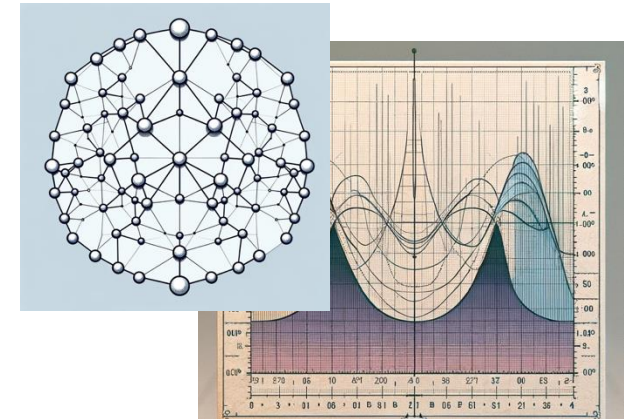
Models

- 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



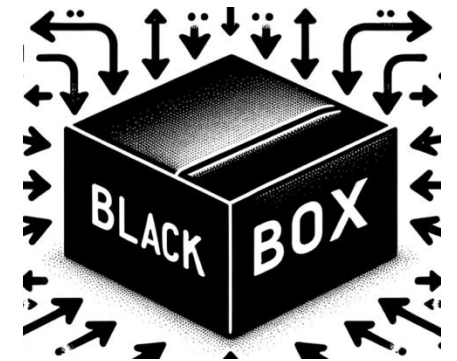
$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), u(t))$$

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



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

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

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

Discrete time: Recurrence equation

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

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

Model structures

- Model: Given past inputs and outputs, provides an estimate of the current output

- Example: ARX model

$$\begin{aligned} M(\boldsymbol{\theta}) : \hat{y}(k) &= \mathbf{a}_1 y(k-1) + \dots + \mathbf{a}_{n_a} y(k-n_a) + \mathbf{b}_0 u(k) + \dots + \mathbf{b}_{n_b} u(k-n_b) \\ &= \underbrace{\begin{bmatrix} y(k-1) & \dots & y(k-n_a) & u(k) & u(k-1) & \dots & u(k-n_b) \end{bmatrix}}_{\boldsymbol{\phi}^T(k)} \boldsymbol{\theta} \end{aligned}$$

$$\boldsymbol{\theta} = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \dots \quad \mathbf{a}_{n_a} \quad \mathbf{b}_0 \quad \mathbf{b}_1 \quad \dots \quad \mathbf{b}_{n_b}]^T$$

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

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

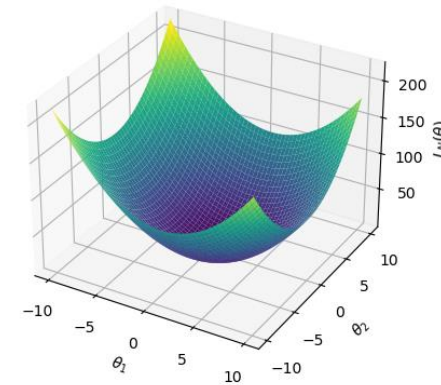
- Example: least-squares loss:

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

Least squares estimate: ARX model structure

$$\begin{aligned} M(\theta) : \hat{y}(k) &= a_1 y(k-1) + \dots + a_{na} y(k-na) + b_0 u(k) + \dots + b_{nb} u(k-nb) \\ &= \underbrace{[y(k-1) \ \dots \ y(k-na) \ u(k) \ u(k-1) \ \dots \ u(k-nb)]}_{\phi^T(k)} \theta \end{aligned}$$

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

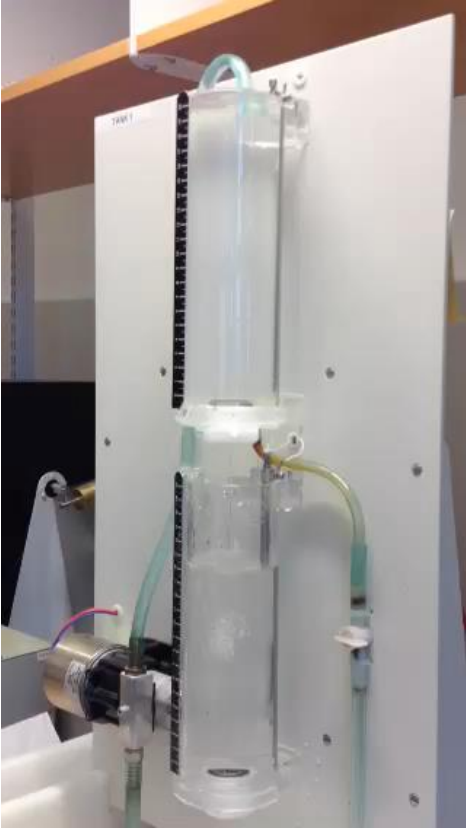


$$\theta^* = \arg \min_{\theta} \mathcal{L}_N(\theta) \rightarrow \theta^* : \frac{\partial \mathcal{L}_N(\theta)}{\partial \theta} = 0$$

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

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

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:**

$$\text{MSE} = \frac{1}{N} \sum_{k=1}^N \|y(k) - \hat{y}(k)\|^2$$

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

Including nonlinearities

- Linear-in-the-parameter NARX model:

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

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

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

How to choose the non-linearities?

- General NARX model:

- N is a nonlinear function (e.g., neural network) of past observations
- Linearity in θ is lost

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

Limitations of (N)ARX models

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

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

- 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

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

$$k = 1, 2, \dots$$

Simulation error minimization

- Linear models

- OE model:

$$\begin{aligned}M(\boldsymbol{\theta}) : \hat{y}(k) &= \boldsymbol{a}_1 \hat{y}(k-1) + \dots + \boldsymbol{a}_{na} \hat{y}(k-na) + \boldsymbol{b}_0 u(k) + \dots + \boldsymbol{b}_{nb} u(k-nb) \\&= \underbrace{[\hat{y}(k-1; \boldsymbol{\theta}) \dots \hat{y}(k-na; \boldsymbol{\theta}) \ u(k) \ u(k-1) \dots u(k-nb)]}_{\boldsymbol{\phi}^T(k; \boldsymbol{\theta})} \boldsymbol{\theta}\end{aligned}$$

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

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

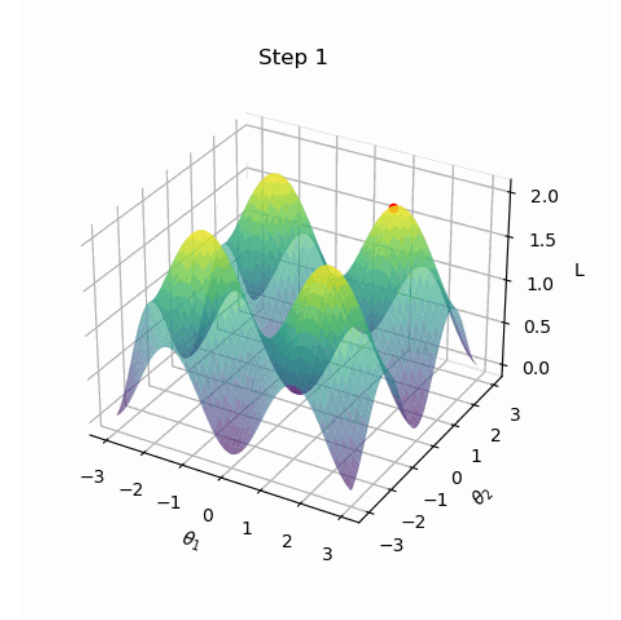
$$L_N(\boldsymbol{\theta}) = \frac{1}{N} \sum_{k=1}^N \|\epsilon(k, \boldsymbol{\theta})\|^2$$

First-order methods for non-convex optimization

Vanilla Gradient Descent

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

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



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

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

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

$$\theta^{(k+1)} = \theta^{(k)} - \gamma v^{(k+1)}$$

```
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:

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

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

Stochastic Gradient Descent

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

Uniformly randomly sampled from $\{1, 2, \dots, N\}$

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

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

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

Mini-batch gradient descent ($B \ll N$):

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

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

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:

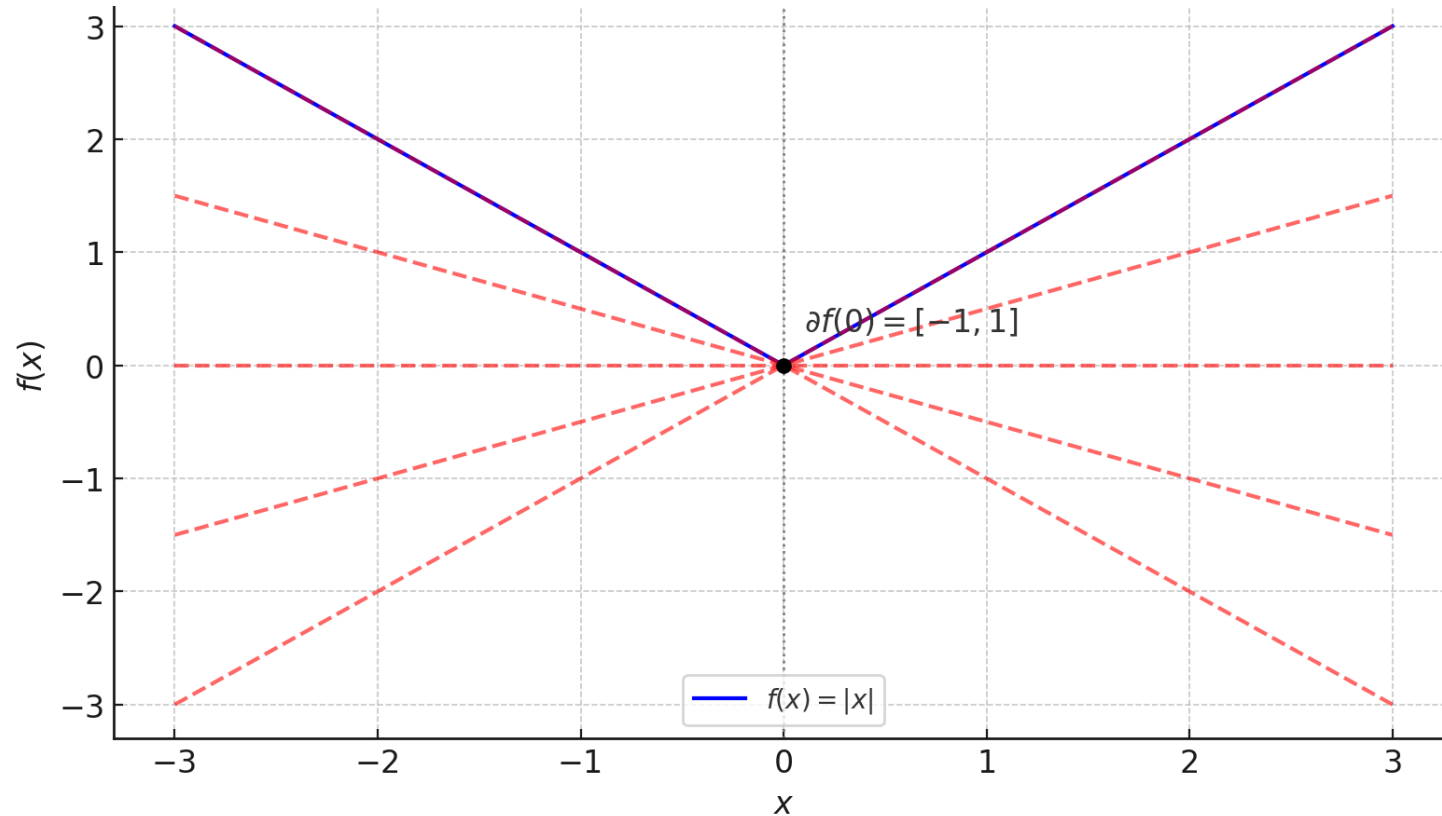
$$M(\theta) : \hat{y}(k) = F^T(y(k-1), \dots, y(k-na), u(k), u(k-1), \dots, u(k-nb))\theta$$

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

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

Non smooth functions

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



Proximal gradient methods

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

non-differentiable convex function 

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

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

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

Fixed point equation



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

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

Fixed point equation: proof of stationarity

We only prove in one direction:

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

- By definition of the proximal operator:

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

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

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

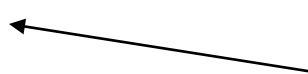
$$0 \in \nabla \mathcal{L}(\theta^*) + \partial R(\theta^*)$$

Proximal gradient methods: constrained optimization

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

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

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

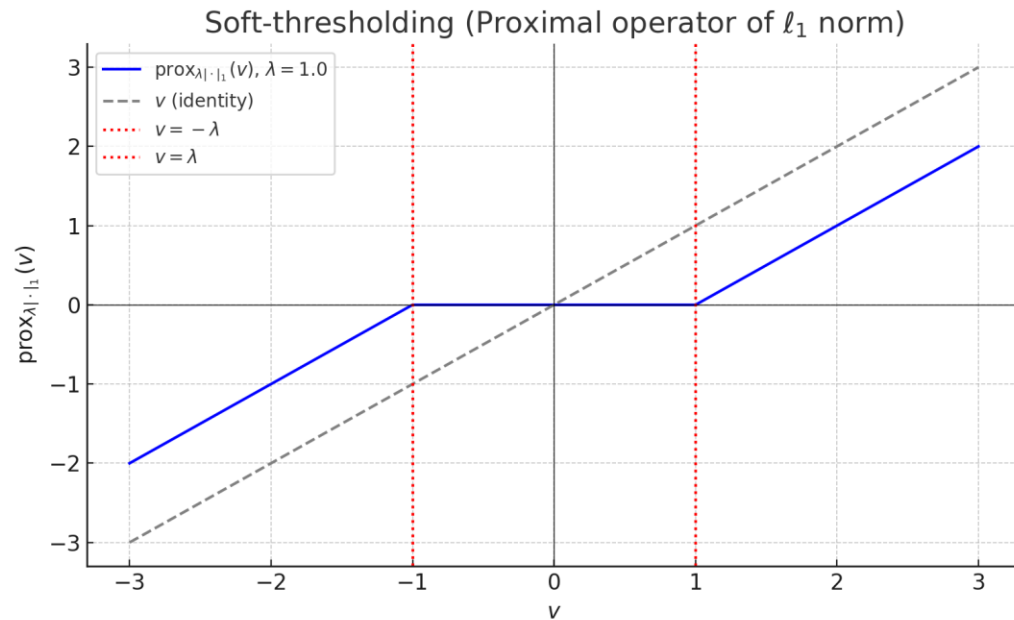
orthogonal projection 

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

$$\min_{\theta} \mathcal{L}(\theta) + \lambda \|\theta\|_1 \quad \text{prox}_{\lambda \|\cdot\|_1}(v) = \arg \min_{\theta \in \mathbb{R}^d} \left\{ \frac{1}{2} \|\theta - v\|^2 + \lambda \|\theta\|_1 \right\}$$
$$= \left[\text{prox}_{\lambda \|\cdot\|_1}(v) \right]_i = \text{sign}(v_i) \cdot \max(|v_i| - \lambda, 0)$$



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

Proximal gradient methods: algorithm L1-regularization

$$\min_{\theta} \mathcal{L}(\theta) + \lambda \|\theta\|_1$$
$$\text{prox}_{\lambda \|\cdot\|_1}(v) = \arg \min_{\theta \in \mathbb{R}^d} \left\{ \frac{1}{2} \|\theta - v\|^2 + \lambda \|\theta\|_1 \right\}$$
$$= \left[\text{prox}_{\lambda \|\cdot\|_1}(v) \right]_i = \text{sign}(v_i) \cdot \max(|v_i| - \lambda, 0)$$
$$\theta^{(k+1)} = \text{prox}_{\gamma R} \left(\theta^{(k)} - \gamma \nabla \mathcal{L}(\theta^{(k)}) \right)$$

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: $g^{(k)} \leftarrow \nabla \mathcal{L}(\theta^{(k)})$
- 4: $z^{(k)} \leftarrow \theta^{(k)} - \gamma g^{(k)}$
- 5: $\theta^{(k+1)} \leftarrow \text{SoftThreshold}_{\gamma \lambda}(z^{(k)})$
- 6: **end for**
- 7: **Return:** $\theta^{(N_{\text{iter}})}$