Computer Modeling in Engineering & Sciences

Tech Science Press

https://doi.org/10.32604/cmes.2026.077726

ARTICLE

# Gradient Descent with Time-Decaying Regularization for Training Linear Neural Networks

**Sergio Isai Palomino-Resendiz[1,2], César Ulises Solís-Cervantes[1,*], Luis Alberto Cantera-Cantera[1,3], Jorge de Jesús Morales-Mercado[1] and Diego Alonso Flores-Hernández[4]**

[1]Departamento de Ingeniería en Control y Automatización, Escuela Superior de Ingeniería Mecánica y Eléctrica (ESIME), Unidad Zacatenco, Instituto Politécnico Nacional, Unidad Profesional Adolfo López Mateos. Av. Luis Enrique Erro S/N, Gustavo A. Madero, Zacatenco, Ciudad de México, México

[2]Departamento de Control Automático, Centro de Investigación y de Estudios Avanzados (CINVESTAV) del Instituto Politécnico Nacional, Unidad Zacatenco, Av. Instituto Politécnico Nacional No. 2508, Col. San Pedro Zacatenco, Ciudad de México, México

[3]Facultad de Ingeniería, Universidad Anáhuac México, Campus Norte, Huixquilucan, Estado de México, México

[4]Sección de Estudios de Posgrado e Investigación, Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas (UPIITA), Instituto Politécnico Nacional, Av IPN 2580, La Laguna Ticoman, G. A. M., Ciudad de México, México

*Corresponding Author: César Ulises Solís-Cervantes. Email: csolisc@ipn.mx

Received: 16 December 2025; Accepted: 25 February 2026; Published: 27 April 2026

**ABSTRACT:** Many linear-in-parameters models arising in identification and control can be expressed as single-layer artificial neural networks (ANNs) with linear activation, enabling online learning via first-order optimization. In practice, however, standard gradient descent often exhibits slow convergence, large intermediate weights, and stagnation when the regressor data are ill-conditioned or computations are performed under finite precision. This paper proposes *Gradient Descent with Time-Decaying Regularization* (GD-TDR), a training algorithm that augments the quadratic loss with a regularization term whose weight decays exponentially in time. The proposed schedule enforces uniform strong convexity during early iterations, effectively mitigating neural-paralysis-like behavior associated with flat directions, while asymptotically vanishing so that the unregularized least-squares solution is recovered. A convergence theorem for GD-TDR is established and a concise pseudocode implementation is provided. Numerical and embedded experiments on an online identification problem of a Chua-type chaotic oscillator demonstrate that GD-TDR converges faster and avoids stagnation compared to standard gradient descent, without introducing the steady-state bias characteristic of fixed quadratic regularization.

**KEYWORDS:** Time-decaying regularization; gradient descent; single-layer linear neural network; online system identification; chaotic oscillator; embedded implementation

## 1 Introduction

### *1.1 State of the Art*

Artificial neural networks (ANNs) have been extensively studied as flexible function approximators and parametric models across a wide range of scientific and engineering tasks. General surveys and application-driven reviews document the breadth of ANN deployments and motivate their use in identification and control, particularly when explicit first-principles modeling is difficult or when data-driven adaptation is required [1–3]. Foundational treatments established the core modeling paradigms and training principles, including linear and nonlinear network structures and their algorithmic implementations [4–6].

In this classical view, many practical learning rules can be interpreted as iterative optimization procedures acting on a quadratic or near-quadratic objective, a perspective that remains central to modern online learning formulations.

A large portion of the ANN training literature is rooted in incremental (first-order) updates. Recent theoretical analyses of learning dynamics in deep linear networks provide explicit characterizations of transient behavior and the interaction between initialization, regularization, and optimization geometry under gradient-based training [7,8]. Within the quadratic-loss setting, basic gradient-descent rules and their stochastic variants remain canonical examples of first-order adaptation mechanisms and highlight how data statistics, conditioning, and step-size choices shape stability and speed of learning [9,10]. In parallel, modern energy-based learning continues to emphasize the role of objective shaping and conditioning in learnability [11]. These works collectively support the view that optimization geometry—not only model expressiveness—plays a decisive role in whether training proceeds smoothly or becomes trapped in slow transient regimes.

Beyond standard multilayer architectures, several specialized ANN families have been developed for robustness, interpretability, and control-oriented deployment. Radial basis function networks and their robust variants provide a well-established pathway to stable approximation under uncertainty and noise [12,13]. In adaptive and self-learning control, ANN-based schemes have been reported for real-time compensation and online tuning, where training must remain stable under streaming data and limited numerical precision [14]. Related approaches in intelligent control also include fuzzy and cerebellar-model architectures that stress adaptation under nonlinearities and disturbances [15,16]. Complementary lines of research continue to refine computationally efficient first-order training, including recent advances in stochastic gradient descent variants [17].

A particularly demanding class of identification problems arises in nonlinear and chaotic dynamics, where sensitivity to initial conditions and measurement noise can degrade learning reliability. Recent studies illustrate both the feasibility of parameter identification in chaotic systems and the numerical challenges of learning from chaotic trajectories [18–20]. These challenges are amplified in embedded or resource-constrained implementations, where finite-precision arithmetic and strict real-time requirements can exacerbate ill-conditioning and lead to slow or stagnant learning. Recent reviews on hardware realizations of neural methods, including FPGA-oriented implementations and embedded control applications, highlight the practical importance of training rules that remain stable and well-conditioned under limited precision. Consistent with these trends, widely used embedded platforms and rapid-prototyping toolchains have enabled end-to-end experimental validation of online learning strategies on microcontrollers [21,22].

Despite the breadth of architectures and applications, an enduring challenge in first-order online training is the susceptibility to slow plateaus and weight growth when the regressor is ill-conditioned or when optimization directions become nearly flat. Classical quadratic (L2/Tikhonov) regularization is a standard remedy to improve conditioning, yet fixed regularization may introduce steady-state bias when the target objective is the unregularized least-squares criterion. This motivates strategies that improve early-stage conditioning while preserving asymptotic fidelity to the original objective, which is the central perspective adopted in this work.

### *1.2 Description and Main Contributions*

Single-layer ANNs with a linear activation function are equivalent to linear regression models and are widely used to represent linear-in-parameters structures in system identification, adaptive filtering, and control. In these applications, the model output can be written as a linear combination of known regressors and unknown parameters, so that training reduces to minimizing a least-squares functional. Although a

closed-form solution exists for batch least squares, embedded and real-time settings often require iterative, lightweight, and online algorithms. For this reason, first-order methods based on gradient descent remain attractive due to their low computational complexity and ease of implementation [2,4,17].

In practice, however, standard gradient descent may perform poorly when the regressor data are ill-conditioned or nearly rank-deficient. These situations are frequent in online identification problems with delayed signals, correlated regressors, or limited excitation. The resulting cost surface can contain nearly flat directions, which leads to slow progress, long plateaus, and very large intermediate weights. On finite-precision hardware, such dynamics can manifest as training stagnation and numerical instabilities that are commonly described as neural-paralysis-like plateau behavior in the neural-network literature [23]. In the linear setting considered here, the effect is not caused by saturation of nonlinear activation functions, but rather by loss of curvature and poor conditioning of the quadratic objective.

A standard remedy is to augment the least-squares loss with a quadratic (Tikhonov) regularizer, which enforces strong convexity and penalizes large weights. Fixed regularization, however, introduces a bias: the minimizer of the regularized problem does not generally coincide with the minimizer of the original least-squares cost. This trade-off is particularly undesirable in identification tasks, where asymptotic accuracy is essential.

This work proposes GD-TDR (Gradient Descent Algorithm with Regularizer—Time Decay), a first-order scheme that interpolates between these two extremes. The algorithm employs a quadratic regularizer whose coefficient decays exponentially over time. As a result, the early iterations benefit from improved curvature and bounded weights, while the regularization vanishes asymptotically and the algorithm recovers the minimizer of the unregularized least-squares functional.

The main contributions of this work are summarized as follows:

- a unified analytical framework that explicitly connects classical gradient descent, gradient descent with fixed quadratic (L2) regularization, and the proposed Gradient Descent with Time-Decaying Regularization (GD-TDR) through a single decay parameter, thereby clarifying their structural similarities and fundamental differences;
- a rigorous convergence theorem establishing that the time-decaying regularization enforces uniform strong convexity during the transient phase while asymptotically recovering the minimizer of the unregularized least-squares problem;
- a concise and self-contained pseudocode implementation of GD-TDR that directly reflects the theoretical development and facilitates reproducible implementation;
- a comprehensive numerical and embedded validation, including online parameter identification of a Chua-type chaotic oscillator and a real-time implementation on an ® STM32F4-Nucleo microcontroller, demonstrating accelerated convergence and mitigation of stagnation without steady-state bias.

The paper is organized as follows. Section 2 introduces the linear ANN model and the least-squares training objective. Section 3 compares gradient-descent training schemes and presents GD-TDR. Section 4 states and proves the convergence theorem and provides the GD-TDR pseudocode. Section 5 shows how common identification models can be written in linear ANN form. Sections 6 and 7 report the numerical and embedded validation, respectively. Section 8 concludes the paper and outlines future work.

## 2 Single-Layer Linear ANN and Least-Squares Training

### 2.1 Model Representation

Consider a single-layer ANN with linear activation (*purelin*) and no bias. Its output is

$$\hat{y} := \mathbf{w}^{\top}\mathbf{x}, \tag{1}$$

where the input vector is $\mathbf{x} := [x_1\ x_2\ \cdots\ x_n]^{\top}$ and the weight vector is $\mathbf{w} := [w_1\ w_2\ \cdots\ w_n]^{\top}$. Fig. 1 illustrates the architecture.

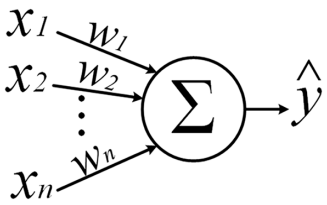


**Figure 1:** Single-layer ANN with linear activation.

### 2.2 Batch Least-Squares Objective

Given a finite data set $\{(\mathbf{x}_k, y_{t,k})\}_{k=1}^{N}$, define the prediction errors $e_k := \hat{y}_k - y_{t,k}$ and the quadratic cost

$$J(\mathbf{w}) := \frac{1}{N}\sum_{k=1}^{N} e_k^2. \tag{2}$$

Let $\mathbf{X} \in \mathcal{M}_{n\times N}(\mathbb{R})$ be the regressor matrix and $\mathbf{y}_t \in \mathbb{R}^N$ the target vector,

$$\mathbf{X} := [\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_N], \qquad \mathbf{y}_t := [y_{t,1}\ y_{t,2}\ \cdots\ y_{t,N}]^{\top}. \tag{3}$$

Then

$$J(\mathbf{w}) = \frac{1}{N}\left\|\mathbf{X}^{\top}\mathbf{w} - \mathbf{y}_t\right\|_2^2. \tag{4}$$

The gradient and Hessian of $J$ with respect to $\mathbf{w}$ are

$$\nabla J(\mathbf{w}) = \frac{2}{N}\mathbf{X}\left(\mathbf{X}^{\top}\mathbf{w} - \mathbf{y}_t\right), \qquad \nabla^2 J(\mathbf{w}) = \frac{2}{N}\mathbf{X}\mathbf{X}^{\top}. \tag{5}$$

The Hessian is positive semidefinite. It is positive definite (and hence $J$ is strongly convex) if and only if $\mathbf{X}\mathbf{X}^{\top}$ is nonsingular.

## 3 Training Schemes and Algorithmic Comparisons

This section summarizes three closely related first-order schemes: classical gradient descent (GD), gradient descent with a *fixed* quadratic regularizer (GD-QR), and the proposed time-decaying regularized scheme (GD-TDR). Only GD-TDR is presented in pseudocode form (Section 4).

### 3.1 Classical Gradient Descent (GD)

Using (2), the GD update with step size $\eta > 0$ is

$$\mathbf{w}(j+1) = \mathbf{w}(j) - \eta \nabla J\big(\mathbf{w}(j)\big) = \mathbf{w}(j) - \frac{2\eta}{N}\mathbf{X}\,\mathbf{e}(j), \tag{6}$$

where $\mathbf{e}(j) := \mathbf{X}^\top \mathbf{w}(j) - \mathbf{y}_t$. When $J$ is strongly convex and $\eta$ is chosen appropriately, GD converges linearly to the unique minimizer. If $\mathbf{X}\mathbf{X}^\top$ is singular or ill-conditioned, $J$ is not strongly convex and GD may exhibit slow progress and large intermediate weights.

### 3.2 Gradient Descent with Fixed Quadratic Regularization (GD-QR)

A standard approach to improve conditioning is to add a quadratic regularizer

$$J_\gamma(\mathbf{w}) := J(\mathbf{w}) + \frac{\gamma}{2}\mathbf{w}^\top \mathbf{P}\mathbf{w}, \tag{7}$$

where $\gamma > 0$ and $\mathbf{P} \in \mathcal{M}_{n\times n}(\mathbb{R})$ is symmetric positive definite. The Hessian becomes

$$\nabla^2 J_\gamma(\mathbf{w}) = \frac{2}{N}\mathbf{X}\mathbf{X}^\top + \gamma\mathbf{P} > 0, \tag{8}$$

so $J_\gamma$ is strongly convex for any $\gamma > 0$. The GD-QR update reads

$$\mathbf{w}(j+1) = \mathbf{w}(j) - \eta\left(\frac{2}{N}\mathbf{X}\mathbf{e}(j) + \gamma\mathbf{P}\mathbf{w}(j)\right). \tag{9}$$

Fixed regularization controls weight growth and improves curvature, but the minimizer of $J_\gamma$ generally differs from the minimizer of $J$. In identification problems, this bias can be detrimental.

### 3.3 Proposed GD-TDR (Time-Decaying Quadratic Regularization)

GD-TDR replaces the constant regularization weight $\gamma$ in (9) by a time-decaying sequence

$$\gamma(j) := \gamma_0 \lambda^j, \qquad \gamma_0 > 0, \quad 0 < \lambda < 1. \tag{10}$$

The weight update becomes

$$\mathbf{w}(j+1) = \mathbf{w}(j) - \eta\left(\frac{2}{N}\mathbf{X}\mathbf{e}(j) + \gamma(j)\mathbf{P}\mathbf{w}(j)\right). \tag{11}$$

Two limiting cases highlight the relationship among the three schemes:

- *Classical GD:* setting $\gamma_0 = 0$ yields (6).
- *Fixed regularization:* setting $\lambda = 1$ yields GD-QR in (9).

Therefore, GD-TDR provides a continuous mechanism to improve conditioning early in training while asymptotically removing the regularization bias. The theoretical properties of this scheme are stated in Theorem 1.

## 4 Theoretical Properties and GD-TDR Pseudocode

We restate the objective in compact form. Define

$$J(\mathbf{w}) := \frac{1}{N}\left\|\mathbf{X}^\top \mathbf{w} - \mathbf{y}_t\right\|_2^2, \qquad \mathbf{e}(\mathbf{w}) := \mathbf{X}^\top \mathbf{w} - \mathbf{y}_t, \tag{12}$$

and let $\mathbf{P} > 0$ be symmetric. For each iteration $j$ consider the regularized functional

$$J_j(\mathbf{w}) := J(\mathbf{w}) + \frac{\gamma(j)}{2}\,\mathbf{w}^\top \mathbf{P}\mathbf{w}, \tag{13}$$

with $\gamma(j)$ defined by (10).

**Theorem 1 (Online GD-TDR):** *Let $\boldsymbol{X} \in \mathbb{M}_{n\times N}(\mathbb{R})$ and $\boldsymbol{y}_t \in \mathbb{R}^N$ be given and assume that $\boldsymbol{X}\boldsymbol{X}^\top$ is positive semidefinite. Let $\boldsymbol{P} \in \mathbb{M}_{n\times n}(\mathbb{R})$ be symmetric and positive definite and define*

$$\mathbf{H} := \frac{2}{N}\,\mathbf{X}\mathbf{X}^\top. \tag{14}$$

*For $\gamma_0 > 0$ and $0 < \lambda < 1$ define*

$$\gamma(j) := \gamma_0 \lambda^j, \qquad j = 0, 1, 2, \ldots \tag{15}$$

*and*

$$J_j(\mathbf{w}) := J(\mathbf{w}) + \frac{\gamma(j)}{2}\,\mathbf{w}^\top \mathbf{P}\mathbf{w}. \tag{16}$$

(a) *For every $j \geq 0$ the functional $J_j$ is strongly convex. More precisely, its Hessian*

$$\nabla^2_{\mathbf{w}} J_j(\mathbf{w}) = \mathbf{H} + \gamma(j)\mathbf{P} \tag{17}$$

*satisfies*

$$\lambda_{\min}\left(\nabla^2_{\mathbf{w}} J_j(\mathbf{w})\right) \geq \gamma(j)\,\lambda_{\min}(\mathbf{P}) > 0 \tag{18}$$

*for all $\boldsymbol{w} \in \mathbb{R}^n$. Consequently, each $J_j$ has a unique global minimizer $\boldsymbol{w}_j^\star$.*

(b) *Suppose that $\boldsymbol{X}\boldsymbol{X}^\top$ is positive definite, so that $J$ has a unique minimizer $\boldsymbol{w}^\star$. Let $\mu := \lambda_{\min}(\boldsymbol{H})$ and $L := \lambda_{\max}(\boldsymbol{H})$, and denote by $\lambda_{\min}(\boldsymbol{P})$ and $\lambda_{\max}(\boldsymbol{P})$ the extremal eigenvalues of $\boldsymbol{P}$. Choose $\eta > 0$ such that*

$$0 < \eta < \frac{2}{L + \gamma_0 \lambda_{\max}(\mathbf{P})}. \tag{19}$$

*Consider the GD-TDR iteration*

$$\mathbf{e}(j) = \mathbf{X}^\top \mathbf{w}(j) - \mathbf{y}_t, \tag{20}$$

$$\mathbf{w}(j+1) = \mathbf{w}(j) - \eta\left(\frac{2}{N}\,\mathbf{X}\mathbf{e}(j) + \gamma(j)\mathbf{P}\mathbf{w}(j)\right), \qquad j = 0, 1, 2, \ldots \tag{21}$$

*Then $\{\boldsymbol{w}(j)\}_{j\geq 0}$ is bounded and converges to $\boldsymbol{w}^\star$.*

(c) *For each $j$ and every $\boldsymbol{w} \in \mathbb{R}^n$,*

$$\left\|\nabla J_j(\mathbf{w})\right\|_2 \geq \gamma(j)\,\lambda_{\min}(\mathbf{P})\,\left\|\mathbf{w} - \mathbf{w}_j^\star\right\|_2. \tag{22}$$

*In particular, when $\gamma(j)$ is large (early iterations), the curvature of $J_j$ is uniformly bounded away from zero and the gradient cannot vanish far from the minimizer. This reduces extended flat regions of the cost surface and mitigates neural-paralysis-like stagnation associated with weight growth.*

**Proof:** (a) Since $\mathbf{H}$ is symmetric and positive semidefinite and $\mathbf{P} > 0$, the matrix $\mathbf{H} + \gamma(j)\mathbf{P}$ is symmetric positive definite for every $\gamma(j) > 0$. Moreover,

$$\lambda_{\min}(\mathbf{H} + \gamma(j)\mathbf{P}) \geq \lambda_{\min}(\mathbf{H}) + \gamma(j)\lambda_{\min}(\mathbf{P}) \geq \gamma(j)\lambda_{\min}(\mathbf{P}) > 0. \tag{23}$$

Hence $J_j$ is strongly convex and has a unique minimizer $\mathbf{w}_j^\star$.

(b) When $\mathbf{H}$ is positive definite, $J$ is $\mu$-strongly convex and $L$-smooth. For each $j$, $J_j$ is $\mu_j$-strongly convex and $L_j$-smooth with

$$\mu_j = \lambda_{\min}(\mathbf{H} + \gamma(j)\mathbf{P}) \geq \mu, \qquad L_j = \lambda_{\max}(\mathbf{H} + \gamma(j)\mathbf{P}) \leq L + \gamma_0\lambda_{\max}(\mathbf{P}). \tag{24}$$

Thus $0 < \eta < 2/L_j$ holds uniformly for all $j$. The update can be written as

$$\mathbf{w}(j+1) = \mathbf{w}(j) - \eta\,\nabla J_j\big(\mathbf{w}(j)\big). \tag{25}$$

Gradient descent on a strongly convex, smooth function with a step size in $(0, 2/L_j)$ is a contraction mapping, hence $\{\mathbf{w}(j)\}$ is bounded. Since $\gamma(j) \to 0$, $\nabla J_j(\mathbf{w}) \to \nabla J(\mathbf{w})$ uniformly on bounded sets. Taking limits in the iteration yields $\nabla J(\bar{\mathbf{w}}) = 0$ for the limit point $\bar{\mathbf{w}}$, which must equal the unique minimizer $\mathbf{w}^\star$.

(c) By strong convexity of $J_j$ with modulus at least $\gamma(j)\lambda_{\min}(\mathbf{P})$ and the standard gradient characterization of strong convexity,

$$\left\|\nabla J_j(\mathbf{w})\right\|_2 \geq \gamma(j)\lambda_{\min}(\mathbf{P})\left\|\mathbf{w} - \mathbf{w}_j^\star\right\|_2. \tag{26}$$

□

**Remark 1:** *The decay rule $\gamma(j+1) = \lambda\gamma(j)$ implements an exponentially vanishing regularization weight. Early in training, $\gamma(0)$ can be chosen sufficiently large so that the quadratic term dominates the curvature and penalizes large weights. As $\gamma(j)$ decreases, the regularization bias disappears asymptotically and the algorithm recovers the geometry of the original least-squares cost. An alternative and simplified way to visualize all of the above is through the pseudocode of the algorithm contained in Algorithm 1.*

**Algorithm 1:** GD-TDR update (time-decaying regularization), cf. Theorem 1

**Require:** $\mathbf{X} \in \mathbb{R}^{n\times N}$, $\mathbf{y}_t \in \mathbb{R}^N$, step size $\eta > 0$, matrix $\mathbf{P} > 0$, $\gamma_0 > 0$, decay factor $0 < \lambda < 1$
**Require:** stopping tolerance $\varepsilon > 0$, maximum iterations $J_{\max}$
Initialize $\mathbf{w}(0)$ (e.g., zeros or small random values)
Set $\gamma(0) \leftarrow \gamma_0$
**for** $j = 0, 1, \ldots, J_{\max} - 1$ **do**
  $\mathbf{e}(j) \leftarrow \mathbf{X}^\top\mathbf{w}(j) - \mathbf{y}_t$
  $\mathbf{g}(j) \leftarrow \frac{2}{N}\mathbf{X}\mathbf{e}(j) + \gamma(j)\mathbf{P}\mathbf{w}(j)$ ▷ gradient of $J_j$
  $\mathbf{w}(j+1) \leftarrow \mathbf{w}(j) - \eta\,\mathbf{g}(j)$
  $\gamma(j+1) \leftarrow \lambda\,\gamma(j)$
  **if** $\|\mathbf{w}(j+1) - \mathbf{w}(j)\|_2 \leq \varepsilon$ **then**
    **break**
  **end if**
**end for**
**return** $\mathbf{w}(j+1)$

To make the above easier to visualize, Table 1 presents a comparison of key aspects.

**Table 1:** Comparison of GD, GD–QR, and GD–TDR for quadratic objectives.

| Aspect | GD | GD–QR (fixed) | GD–TDR (time-decaying) |
|---|---|---|---|
| Objective optimized | $J(\mathbf{w}) = \frac{1}{N}\|\mathbf{X}^\top\mathbf{w} - \mathbf{y}\|_2^2$ | $J_\lambda(\mathbf{w}) = J(\mathbf{w}) + \lambda\|\mathbf{w}\|_2^2$ | $J_{\lambda(k)}(\mathbf{w}) = J(\mathbf{w}) + \lambda(k)\|\mathbf{w}\|_2^2$ |
| Update rule | $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta\nabla J(\mathbf{w}_k)$ | $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta\big(\nabla J(\mathbf{w}_k) + 2\lambda\,\mathbf{w}_k\big)$ | $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta\big(\nabla J(\mathbf{w}_k) + 2\lambda(k)\,\mathbf{w}_k\big)$ |
| Early strong convexity | Not guaranteed (data dependent) | Guaranteed for $\lambda > 0$ | Guaranteed if $\lambda(0)$ is sufficiently large |
| Hessian conditioning | $\nabla^2 J = \frac{2}{N}\mathbf{X}\mathbf{X}^\top$ (may be ill-conditioned) | $\nabla^2 J_\lambda = \frac{2}{N}\mathbf{X}\mathbf{X}^\top + 2\lambda\mathbf{I}$ | $\nabla^2 J_{\lambda(k)} = \frac{2}{N}\mathbf{X}\mathbf{X}^\top + 2\lambda(k)\mathbf{I}$ |
| Asymptotic objective recovered | Yes (minimizes $J$) | No (minimizes $J_\lambda$) | Yes, if $\lambda(k) \to 0$ |
| Steady-state bias (w.r.t. LS) | No | Yes (increases with $\lambda$) | No (vanishing regularization) |
| Stagnation/NP mitigation | May exhibit plateaus | Plateaus reduced, but biased optimum | Plateaus reduced without biasing the optimum |
| Recommended use case | Well-conditioned regressors | Permanent conditioning / shrinkage | Early conditioning with unbiased asymptote |

## 5 Application to Online Parameter Identification

This section shows how common linear identification models can be written as single-layer linear ANNs, which allows applying GD-TDR directly.

### *5.1 Discrete Transfer Functions*

Consider a discrete transfer function

$$\frac{Y(z)}{U(z)} := \frac{a_n z^{-s} + a_{n-1}z^{-s+1} + \cdots + a_1 z^{-1} + a_0}{b_m z^{-m} + b_{m-1}z^{-m+1} + \cdots + b_1 z^{-1} + 1}, \tag{27}$$

with $s \le m$. After inverse $z$-transform (zero initial conditions), one obtains an ARX-like representation

$$y[n] = \sum_{k=0}^{s} a_k\, u[n-k] - \sum_{r=1}^{m} b_r\, y[n-r]. \tag{28}$$

Defining

$$\begin{aligned} \mathbf{w} &:= \begin{bmatrix} a_0 & a_1 & \cdots & a_s & b_1 & \cdots & b_m \end{bmatrix}^\top, \\ \mathbf{x}[n] &:= \begin{bmatrix} u[n] & u[n-1] & \cdots & u[n-s] & -y[n-1] & \cdots & -y[n-m] \end{bmatrix}^\top. \end{aligned} \tag{29}$$

The model becomes $y[n] = \mathbf{w}^{\top}\mathbf{x}[n]$, which is exactly the output of a single-layer linear ANN. Fig. 2 sketches this identification setup.

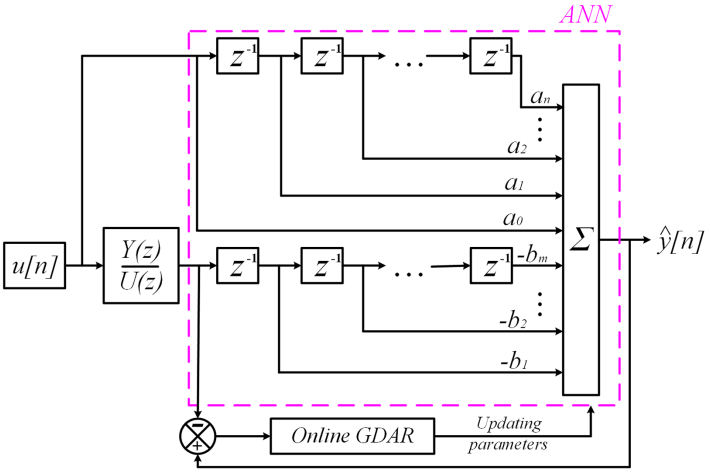


**Figure 2:** Single-layer ANN representation for parameter identification.

### *5.2 Discrete State-Space Models*

For a discrete state-space (DSS) system

$$\mathbf{x}[n+1] = \mathbf{A}\mathbf{x}[n] + \mathbf{B}\mathbf{u}[n], \tag{30}$$

with appropriate dimensions, the right-hand side is linear in the unknown entries of $\mathbf{A}$ and $\mathbf{B}$. By stacking the parameters into a single vector (or matrix) and defining a regressor vector that contains state and input components, the DSS update can also be written in the form $\mathbf{x}[n+1] = \mathbf{W}\boldsymbol{\phi}[n]$, where $\mathbf{W}$ collects the unknown parameters. This is compatible with GD-TDR, which can be applied entrywise.

A practical issue is causality: to update parameters at time $n$, the target $\mathbf{x}[n+1]$ is needed. A simple remedy is to insert a one-step delay in the learning loop, which preserves the identification objective while keeping the update implementable in real time. Fig. 3 illustrates the ANN view of the DSS model, while Fig. 4 shows a delay-based identification for DSS parameter identification.

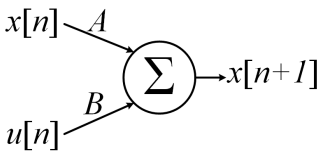


**Figure 3:** DSS model viewed as a linear ANN mapping.

**Remark 2:** *Although the focus is on linear-in-parameters models, the same idea can be used to identify local linearizations of nonlinear systems around operating points, provided the regressors are constructed accordingly.*

## 6 Numerical Validation

### *6.1 Experimental Setup*

The numerical validation considers online parameter identification of a chaotic system whose dynamics are equivalent to those of a Chua-type oscillator. Chaotic trajectories provide a demanding excitation pattern

for adaptive algorithms and are known to expose slow transients and stagnation effects in gradient-based learning [19,20].

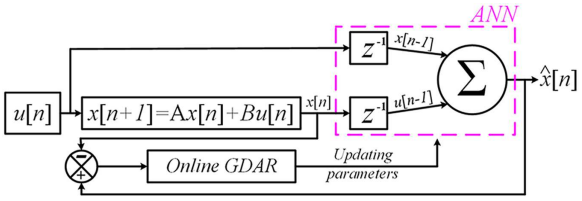


**Figure 4:** Delay-based scheme for DSS parameter identification.

The Chua oscillator is described by

$$\begin{cases} \dot{x} = \alpha\big(y - x - \varphi(x)\big), \\ \dot{y} = x - y + z, \\ \dot{z} = -\beta y, \end{cases} \tag{31}$$

where

$$\varphi(x) := m_1 x + \frac{1}{2}\,(m_0 - m_1)\,\big(|x+1| - |x-1|\big). \tag{32}$$

For the parameter values $\alpha = 15.6$, $\beta = 25$, $m_0 = -1.1429$, and $m_1 = -0.7143$, the corresponding attractor is shown in Fig. 5.

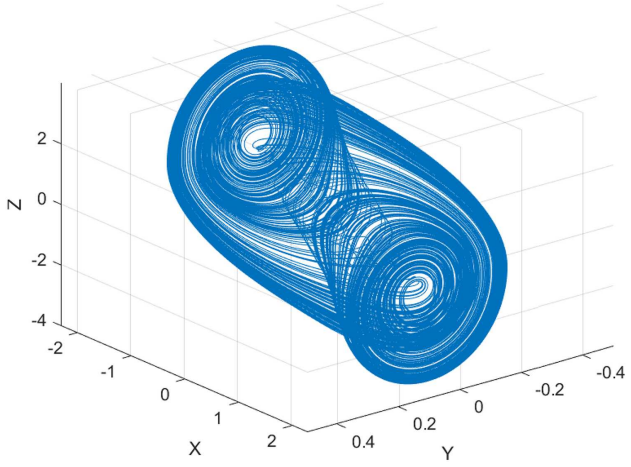


**Figure 5:** Trajectory of Chua dynamics (chaotic attractor).

To pose an identification problem that is linear in the unknown parameters, the dynamics are rewritten as

$$\underbrace{\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}}_{\mathbf{y}_t} = \underbrace{\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} x \\ y \\ z \\ d \end{bmatrix}}_{\phi}, \qquad d := |x+1| - |x-1|. \tag{33}$$

In this form, **W** is an unknown parameter matrix to be estimated online from the measured signals. The validation compares GD-TDR against classical GD. In addition, Section 3 provides an explicit comparison with fixed quadratic regularization (GD-QR), which is obtained as the special case $\lambda = 1$.

Fig. 6 shows the main program used in the numerical simulations.

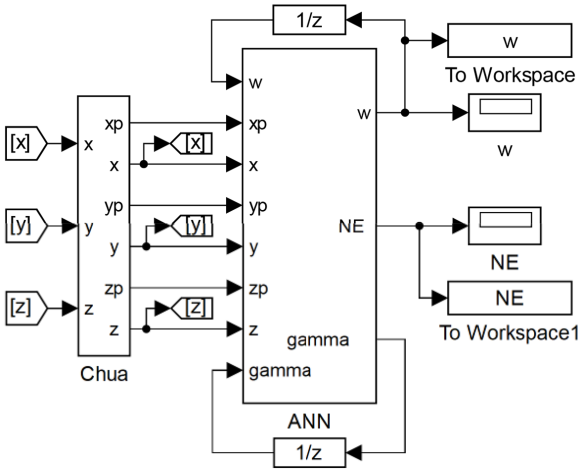


**Figure 6:** Main program in ® Matlab-Simulink environment.

In the reported tests, the learning factor was set to $\eta = 0.02$. For GD-TDR, the regularization parameters were initialized at $\gamma_0 = 0.99$ and decayed according to (10) with $\lambda = 0.9998$. The simulations used the `ode8` solver with fixed step size and sampling time 0.001 s.

### *6.2 Results*

Fig. 7 shows the convergence of the estimated parameters towards the target values, while Fig. 8 reports the norm of the estimation error. The final identified parameters for GD-TDR, GD and GD-QR are listed in (34) to (36), respectively.

$$\mathbf{W}_{\text{GD-TDR}} = \begin{bmatrix} -4.457 & 15.600 & 0.000 & 3.343 \\ 1.000 & -1.000 & 1.000 & 0.000 \\ 0.000 & -25.000 & 0.000 & 0.000 \end{bmatrix}, \qquad \text{NE} = 0.0001. \tag{34}$$

$$\mathbf{W}_{\text{GD}} = \begin{bmatrix} -4.437 & 15.580 & 0.001 & 3.327 \\ 0.997 & -0.997 & 1.000 & 0.002 \\ 0.0202 & -24.987 & 0.000 & 0.0151 \end{bmatrix}, \qquad \text{NE} = 0.0013. \tag{35}$$

$$\mathbf{W}_{\text{GD-QR}} = \begin{bmatrix} -3.3124 & 14.3232 & 0.0969 & 2.5913 \\ 0.8175 & -0.8712 & 0.9908 & 0.1169 \\ -1.0477 & -23.4959 & -0.3202 & 0.5065 \end{bmatrix}, \qquad \text{NE} = 0.0419. \tag{36}$$

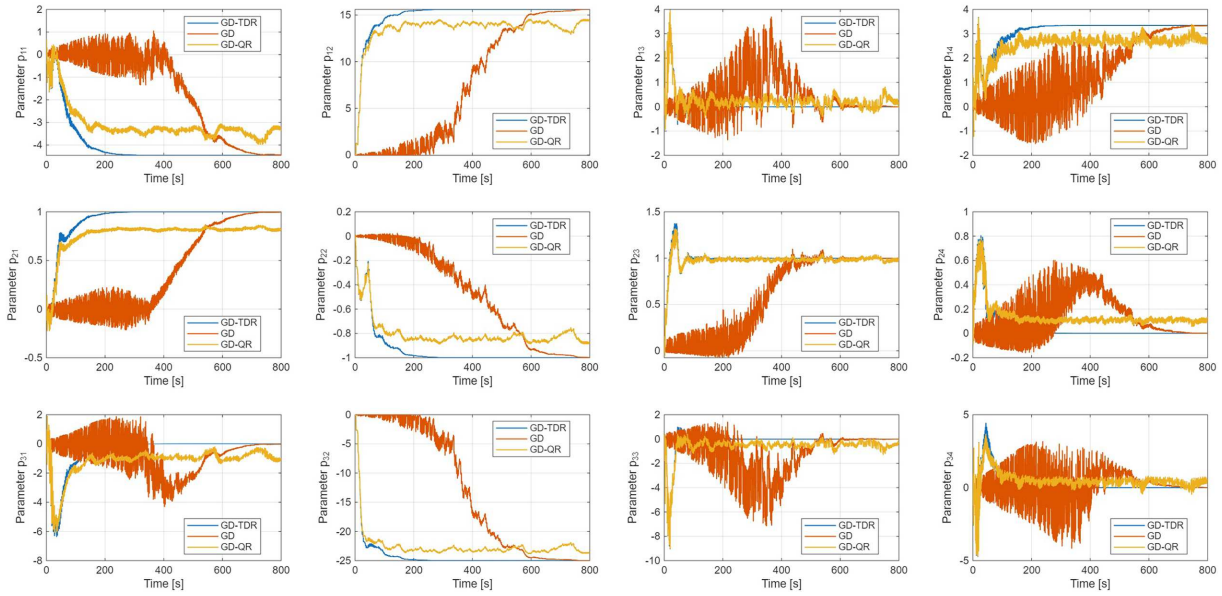


**Figure 7:** Convergence of GD, GD-QR and GD-TDR parameters.

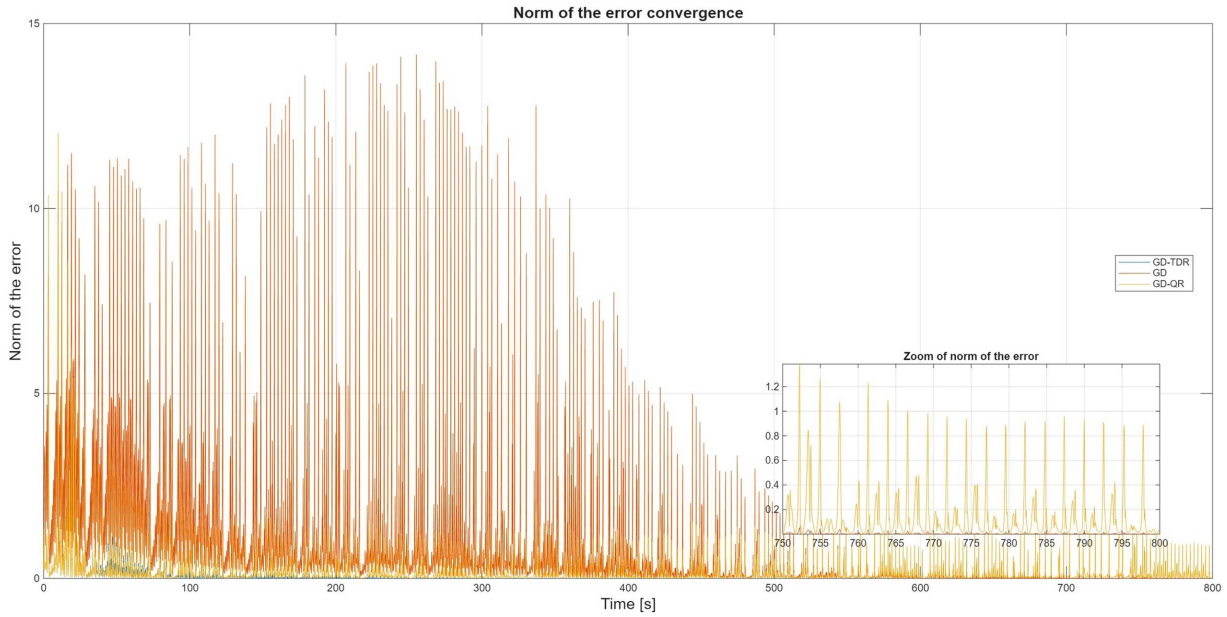


**Figure 8:** Norm of the parameter-estimation error.

### *6.3 Discussion*

GD and GD-TDR algorithms converge to high-accuracy estimates; however, GD-TDR exhibits markedly faster convergence and substantially shorter stagnation transients. In the considered experiment, GD requires approximately 800 s to reach steady convergence, whereas GD-TDR attains comparable accuracy in about 200 s. The error norm (NE) trajectories further indicate that GD spends a significant portion of the runtime in plateau-like regions before converging, a behavior consistent with ill-conditioning and the presence of flat directions in the quadratic objective function.

GD-QR mitigates oscillations during the convergence process but yields the poorest parameter convergence (NE = 0.0419). This behavior is expected, since the QR-based transformation effectively shifts the original optimal point to an alternative one in order to increase the convexity of the cost functional, thereby smoothing the convergence dynamics at the expense of final parameter accuracy.

From an algorithmic viewpoint, these improvements can be interpreted through Theorem 1: the decaying regularization increases the curvature of $J_j$ in the early iterations, preventing the gradient from vanishing far from the minimizer and discouraging weight growth in nearly-flat directions. As $\gamma(j)$ decreases, the method smoothly transitions toward the unregularized least-squares objective, thereby avoiding the steady-state bias that would occur if a fixed $\gamma$ were used (GD-QR).

## 7 Embedded Implementation on a Microcontroller

To evaluate suitability for low-processing-capacity platforms, GD-TDR was implemented on an ® STM32F4-Nucleo microcontroller. The implementation was developed in the ® Matlab-Simulink environment and deployed using the ® Waijung toolkit. Figs. 9 and 10 show the board-level program configuration, which follows the same signal flow as the numerical setup and adds serial communication blocks for monitoring. In particular, in Fig. 9, the content of the block called chua can be located in Fig. A1 as well as its programming, which are contained in the Appendix A.

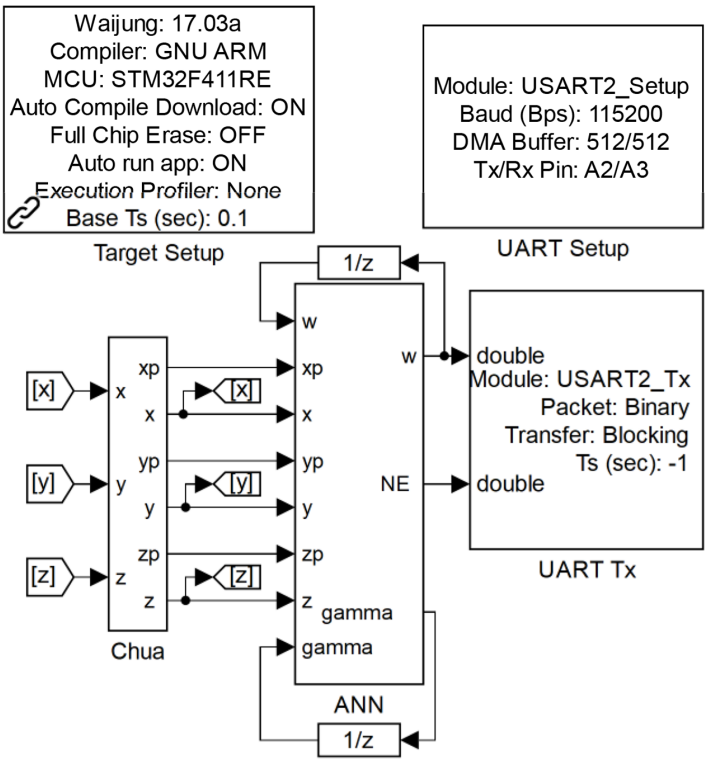


**Figure 9:** Program configuration for the ® STM32F4-Nucleo board.

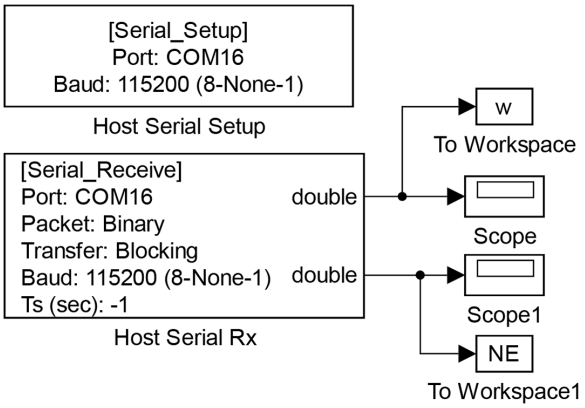


**Figure 10:** CPU monitoring program.

## Experimental Results

In this work, neuron-paralysis-like behavior is quantitatively assessed through a combination of indicators, including prolonged plateaus in the loss evolution, persistent attenuation of the effective gradient norm, and excessive transient growth of the parameter vector prior to convergence.

Fig. 11 reports the convergence behavior observed on the microcontroller. The results are qualitatively consistent with the numerical simulations, indicating that the proposed method preserves its robustness against stagnation even under limited precision and memory.

$$\mathbf{W}_{\mathrm{STM32}} = \begin{bmatrix} -4.439 & 15.585 & -0.001 & 3.333 \\ 0.997 & -0.987 & 1.000 & 0.001 \\ 0.014 & -24.988 & 0.001 & 0.007 \end{bmatrix}, \qquad \mathrm{NE} = 0.001. \tag{37}$$

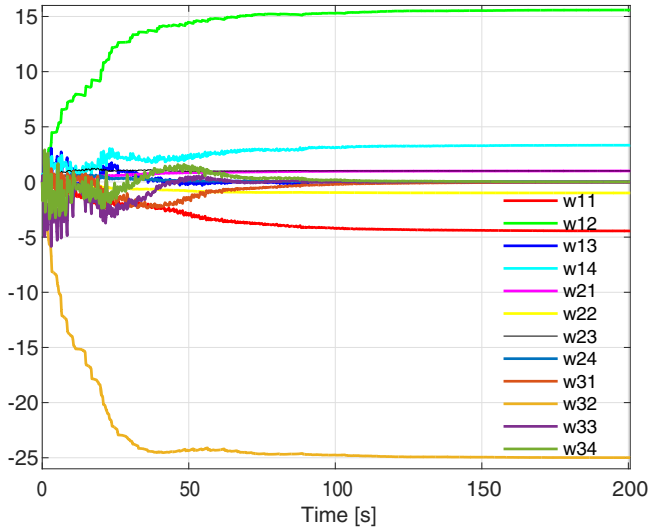


**Figure 11:** Dynamics of convergence of GD-TDR weights on the ® STM32F4-Nucleo board.

## 8 Conclusions and Future Work

This paper introduced GD-TDR, a time-decaying quadratically regularized gradient-descent algorithm for training single-layer linear ANNs. The method is motivated by online identification problems in which the regressor data can be ill-conditioned and standard gradient descent may suffer from long plateaus, large intermediate weights, and neural-paralysis-like stagnation on finite-precision hardware. GD-TDR addresses this issue by enforcing strong convexity early in training through a quadratic penalty and then removing the penalty asymptotically via an exponential decay schedule.

A convergence theorem was provided that formalizes the key mechanism: for every iteration index the regularized objective remains strongly convex, so flat directions are eliminated, and under standard step-size conditions the iterates converge to the minimizer of the original (unregularized) least-squares cost as the regularization vanishes. The numerical validation on online identification of a Chua-type chaotic oscillator and the implementation on an ® STM32F4-Nucleo microcontroller confirm that the proposed scheme converges faster than conventional gradient descent and significantly reduces stagnation transients, while preserving high identification accuracy.

Future work will focus on three directions. First, the decay schedule $\gamma(j)$ can be adapted online using measurable indicators of conditioning or excitation, rather than being fixed *a priori*. Second, systematic and low-cost design rules for selecting $\mathbf{P}$ (e.g., diagonal or structured choices compatible with embedded computation) will be investigated, including robustness to noise and time-varying parameters. Third, extending the approach beyond linear activation to shallow nonlinear networks and to constrained identification problems is of interest, where time-decaying regularization may provide similar benefits without sacrificing asymptotic accuracy.

**Acknowledgement:** The authors would like to thank Professor Alexander Poznyak for his valuable review of the work, as well as Bruce Dickinson for his motivation throughout the development of this research. They also acknowledge and are grateful for the funding provided by the IPN-SIP (SIP 20250023, 20250424, 20251300, 20251721, and 20253411), SECIHTI (CF-2023-I-1635), and the Sistema Nacional de Investigadores e Investigadoras (SNII) of Mexico.

**Funding Statement:** Funding was provided by the IPN-SIP (SIP 20250023, 20250424, 20251300, 20251721, 20253411 and MULTI-2026-0035), SECIHTI (CF-2023-I-1635), and the Sistema Nacional de Investigadores e Investigadoras (SNII) of Mexico.

**Author Contributions:** Conceptualization, Sergio Isai Palomino-Resendiz and César Ulises Solís-Cervantes; methodology, Diego Alonso Flores-Hernández and Sergio Isai Palomino-Resendiz; software, César Ulises Solís-Cervantes, Sergio Isai Palomino-Resendiz and Luis Alberto Cantera-Cantera; validation, Luis Alberto Cantera-Cantera and Jorge de Jesús Morales-Mercado; formal analysis, César Ulises Solís-Cervantes, Sergio Isai Palomino-Resendiz and Diego Alonso Flores-Hernández; investigation, Sergio Isai Palomino-Resendiz; resources, César Ulises Solís-Cervantes, Sergio Isai Palomino-Resendiz and Diego Alonso Flores-Hernández; data curation, Luis Alberto Cantera-Cantera and Jorge de Jesús Morales-Mercado; writing—original draft preparation, Sergio Isai Palomino-Resendiz; writing—review and editing, Sergio Isai Palomino-Resendiz and César Ulises Solís-Cervantes; visualization, Sergio Isai Palomino-Resendiz; supervision, Sergio Isai Palomino-Resendiz and César Ulises Solís-Cervantes; project administration, Sergio Isai Palomino-Resendiz; funding acquisition, Diego Alonso Flores-Hernández and Sergio Isai Palomino-Resendiz. All authors reviewed and approved the final version of the manuscript.

**Availability of Data and Materials:** The data that support the findings of this study are available from the Corresponding Author, César Ulises Solís-Cervantes, upon reasonable request.

**Ethics Approval:** Not applicable.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| ANN | Artificial Neural Network |
| GD | Gradient Descent |
| GD-QR | Gradient Descent with Quadratic Regularization |
| GD-TDR | Gradient Descent with Time-Decaying Regularization |
| NP | Neural Paralysis |
| SLM | Stagnation in Local Minima |

## Appendix A Chua Model Block

The following MATLAB function block implements (31).

```
function [xp, yp, zp] = fcn(x,y,z)
alpha = 15.6; beta = 25; m0 = -8/7; m1 = -5/7;
phi = m1*x + 0.5*(m0-m1)*(abs(x + 1)-abs(x - 1));
d1 = y - x - phi; d2 = x - y + z; d3 = -y;
xp = alpha*d1; yp = d2; zp = beta*d3;
end
```

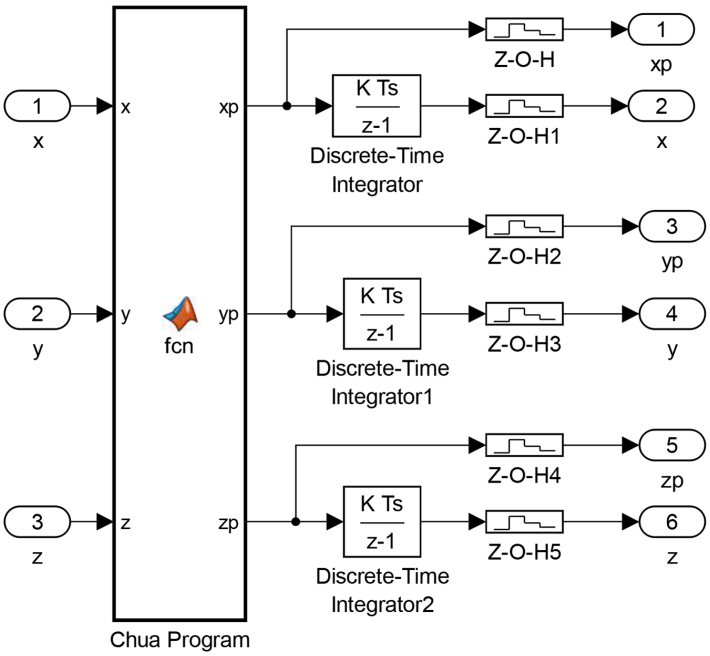


**Figure A1:** Contents of the block called `Chua` of the main program.

## References

1. Pillonetto G, Aravkin A, Gedon D, Ljung L, Ribeiro AH, Schön TB. Deep networks for system identification: a survey. Automatica. 2025;171(7):111907. doi:10.1016/j.automatica.2024.111907.
2. Dong Q, Liu L, Wang P, Zhang L, Zhang J. Neural network-based parametric system identification: a comprehensive review. Int J Syst Sci. 2023;54(13):2676–88. doi:10.1080/00207721.2023.2241957.

3. Yu P, Wan H, Zhang B, Wu Q, Zhao B, Xu C, et al. Review on system identification, control, and optimization based on artificial intelligence. Mathematics. 2025;13(6):952. doi:10.3390/math13060952.
4. Prince SJD. Understanding deep learning. Cambridge, MA, USA: MIT Press; 2023.
5. Bishop CM, Bishop H. Deep learning: foundations and concepts. Cham, Switzerland: Springer; 2024.
6. Murphy KP. Probabilistic machine learning: an introduction. Cambridge, MA, USA: MIT Press; 2022.
7. Braun L, Dominé CCJ, Fitzgerald JE, Saxe AM. Exact learning dynamics of deep linear networks with prior knowledge. In: Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Red Hook, NY, USA: Curran Associates, Inc.; 2022. p. 6615–29.
8. Ziyin L, Li B, Meng X. Exact solutions of a deep linear network. In: Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Red Hook, NY, USA: Curran Associates, Inc.; 2022. p. 24446–58.
9. Gambella C, Ghaddar B, Naoum-Sawaya J. Optimization problems for machine learning: a survey. Eur J Oper Res. 2021;290(3):807–28. doi:10.1016/j.ejor.2020.08.045.
10. Ahn K, Zhang J, Sra S. Understanding the unstable convergence of gradient descent. In: Proceedings of the 39th International Conference on Machine Learning (ICML). London, UK: PMLR; 2022. p. 247–57.
11. Xie J, Gao R, Nijkamp E, Zhu S-C, Wu YN. Cooperative training of fast thinking initializer and slow thinking solver for conditional learning. IEEE Trans Pattern Anal Mach Intell. 2022;44(8):3957–73. doi:10.1109/TPAMI.2021.3069023.
12. Wurzberger J, Schwenker F. Learning in deep radial basis function networks. Entropy. 2024;26(5):368. doi:10.3390/e26050368.
13. Kalina J, Vidnerová P, Janáček P. Highly robust training of regularized radial basis function networks. Kybernetika. 2024;60(1):38–59. doi:10.14736/kyb-2024-1-0038.
14. Wang D, Gao N, Liu D, Li J, Lewis F. Recent progress in reinforcement learning and adaptive dynamic programming for advanced control applications. IEEE/CAA J Autom Sin. 2024;11(1):18–36. doi:10.1109/JAS.2023.123843.
15. Le T-L, Huynh T-T, Hong S-K, Lin C-M. Hybrid neural network cerebellar model articulation controller design for non-linear dynamic time-varying plants. Front Neurosci. 2020;14:695. doi:10.3389/fnins.2020.00695.
16. Razmi M, Macnab CJB. Near-optimal neural-network robot control with adaptive gravity compensation. Neurocomputing. 2020;389(6):83–92. doi:10.1016/j.neucom.2020.01.026.
17. Tian Y, Zhang Y, Zhang H. Recent advances in stochastic gradient descent in deep learning. Mathematics. 2023;11(3):682. doi:10.3390/math11030682.
18. Liang C, Ma W, Ma C, Guo L. Harnessing machine learning for identifying parameters in fractional chaotic systems. Appl Math Comput. 2025;500(2):129454. doi:10.1016/j.amc.2025.129454.
19. Chakraborty S, Größmann AH, Benner P. Divide and conquer: learning chaotic dynamical systems using deep neural networks. Comput Methods Appl Mech Eng. 2024;430(8):117442. doi:10.1016/j.cma.2024.117442.
20. Mikhaeil JM, Monfared Z, Durstewitz D. On the difficulty of learning chaotic dynamics with RNNs. In: NIPS'22: Proceedings of the 36th International Conference on Neural Information Processing Systems. Red Hook, NY, USA: Curran Associates, Inc.; 2022. p. 11297–312.
21. Mohan N, Hosni A, Atef M. Neural networks implementations on fpga for biomedical applications: a review. SN Computer Sci. 2024;5(8):1004. doi:10.1007/s42979-024-03381-4.
22. Majumdar P. Spiking neural networks: a comprehensive review of diverse applications, research progress, challenges and future research directions. Evol Syst. 2025;16(4):125. doi:10.1007/s12530-025-09755-0.
23. Achour EM, Malgouyres F, Gerchinovitz S. The loss landscape of deep linear neural networks: a second-order analysis. J Mach Learn Res. 2024;25:242.