

Locally Convex Neural Lyapunov Functions and Region of Attraction Maximization for Stability of Nonlinear Systems

Lucas Hugo¹, Philippe Feyel² and David Saussié¹ ^a

¹Robotics and Autonomous Systems Laboratory, Polytechnique Montréal, Montréal, Québec H3T1J4, Canada

²Safran Electronics & Defense Canada, Montréal, Canada

Keywords: Lyapunov Functions, Stability, Neural Network Optimization, Convexity, Region of Attraction.

Abstract: The Lyapunov principle involves to find a positive Lyapunov function with a local minimum at the equilibrium point, whose time derivative is negative with a local maximum at that point. As a validation, it is usual to check the sign of the Hessian eigenvalues which can be complex: it requires to know a formal expression of the system dynamics, and especially a differentiable one. In order to circumvent this, we propose in this paper a scheme allowing to validate these functions without computing the Hessian. Two methods are proposed to force the convexity of the function near the equilibrium; one uses a neural single network to model the Lyapunov function, the other uses an additional one to approximate its time derivative. The training process is designed to maximize the region of attraction of the locally convex neural Lyapunov function trained. The use of examples allows us to validate the efficiency of this approach, by comparing it with the Hessian-based approach.


1 INTRODUCTION

Lyapunov theory is a commonly used tool in control theory to analyse stability of dynamical systems. A Lyapunov function can be used to determine the equilibrium stability of nonlinear systems, as well as to estimate the Region of Attraction (RoA) or establish a stabilizing control (Mawhin, 2005; Khalil, 2001). This type of potential function keeps track of the energy dissipated by a system. In addition to modelling physical energy, a Lyapunov function can also represent abstract quantities provided it fulfills the following properties: (1) it must be a positive definite function locally, (2) it must have continuous partial derivatives, and (3) its time derivative along any state trajectory must be negative semi-definite.

There are many ways of finding a Lyapunov function in the literature, each one of them with its drawbacks and advantages (Giesl and Hafstein, 2015), but recently optimization methods based on neural networks training have increased significantly. Indeed, the ability of neural networks to represent a function has been well established since the early 90s (Cybenko, 1989; Hornik et al., 1989; Hornik, 1991; Liang and Srikant, 2017). Neural approximations

were first used to determine Lyapunov candidates for nonlinear autonomous systems in (Prokhorov, 1994) and various methods have since been proposed to guide network training.

(Petridis and Petridis, 2006) develop an Hessian-based approach to ensure a strict validation of Lyapunov properties near an equilibrium and (Bocquillon et al., 2022) extend it to prove asymptotic, exponential or ISS stability of continuous or discrete time systems using a genetic algorithm. The network structure in these works is restricted to a single hidden layer network, and they require to know a differentiable expression of the system dynamics. (Richards et al., 2018) propose a training framework based on an iterative gradient-descent to estimate the largest possible RoA for general nonlinear dynamical systems with structural properties on activation functions and weighting matrices. To completely avoid structural limitations on the trained network, (Chang et al., 2020) repeats multiple gradient-based optimization problems to identify states that violate Lyapunov conditions and (Gaby et al., 2022) propose a versatile neural network architecture called *Lyapunov-Net* that ensure positive definiteness of the Lyapunov candidate while simplifying the optimization problem formulation. Nevertheless a concession is made in terms of training efficiency

^a  <https://orcid.org/0000-0002-4228-1109>

as the cost functions used needs to converge to 0 so that the Lyapunov candidate obtained strictly validate Lyapunov properties, especially near the equilibrium. To circumvent this issue (Chang and Gao, 2021) introduce the concept of *Almost Lyapunov functions*, theoretically relevant but difficult to validate in practice.

To provide a tool with the same validity guarantee as the Hessian-based approach, but applicable to general non-linear systems that are not necessarily differentiable, we propose two methods to train locally convex neural Lyapunov functions with the flexibility of an optimization-based approach. The framework is designed so that time derivative function local concavity is guided by the cost function while local convexity of the Lyapunov function is structurally enforced, allowing a great liberty of network structure, while maximizing the RoA.

This paper is structured as follows. Preliminaries related to Lyapunov theory are introduced in section 2. In section 3, we present the two methods to train locally convex neural Lyapunov function, one uses a single network to model the Lyapunov function, the other uses an additional network to approximate its time derivative. Finally, we illustrate in section 4 the efficiency of our approaches with examples, by comparing it with the Hessian-based approach.

2 PRELIMINARIES

Notations and definitions used in this paper are introduced in this section. Let \mathbb{R} denote the set of real numbers, $\|\cdot\|$ denote the euclidean norm on \mathbb{R}^n , \sqcup the disjoint union between sets, and $\mathbb{X} \subset \mathbb{R}^n$, a set containing $\mathbf{x} = 0$.

This paper deals with following autonomous systems:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \quad (1)$$

where $\mathbf{f} : \mathbb{X} \rightarrow \mathbb{R}^n$ is a locally Lipschitz map with at least one equilibrium point \mathbf{x}_e , that is $\mathbf{f}(\mathbf{x}_e) = 0$. Throughout this paper, the equilibrium point is commonly set at the origin, $\mathbf{x}_e = 0$ without loss of generality.

Theorem 2.1 (Lyapunov Theory). (Khalil, 2001) Let $V : \mathbb{X} \rightarrow \mathbb{R}$ be a continuously differentiable function,

$$V(\mathbf{x}_e) = 0 \text{ and } V(\mathbf{x}) > 0 \text{ in } \mathbb{X} - \{\mathbf{x}_e\} \quad (2)$$

$$\dot{V}(\mathbf{x}) \leq 0 \text{ in } \mathbb{X} \quad (3)$$

then, $\mathbf{x}_e = 0$ is stable. Moreover, if

$$\dot{V}(\mathbf{x}) < 0 \text{ in } \mathbb{X} - \{\mathbf{x}_e\} \quad (4)$$

then, $\mathbf{x}_e = 0$ is asymptotically stable.

For some $c > 0$, the surface $V(\mathbf{x}) = c$ is called a level set of V , and the (possibly conservative) subset $\Omega_c = \{\mathbf{x} \in \mathbb{X} | V(\mathbf{x}) \leq c\} \subset \mathbb{X} \subset \mathbb{R}^n$ is an estimate of the Region of Attraction (RoA). The system will converge to 0 from every initial point \mathbf{x}_0 belonging to Ω_c .

A neural network Lyapunov function takes any state vector of the system as an input, and gives a scalar value at output. In this paper, the parameter vector for a Lyapunov function candidate is noted as $\boldsymbol{\theta}$, the candidate itself is noted as $V_{\boldsymbol{\theta}}$, and its time derivative function is noted as $\dot{V}_{\boldsymbol{\theta}}$.

3 TRAINING PROCESS

We describe how to train a locally convex neural Lyapunov function, so that the Lyapunov conditions can be verified to ensure that the equilibrium point of the system (1) is locally asymptotically stable.

3.1 Single Network Approach

To define the structure of the function to be trained, we use the Lyapunov-Net formulation proposed in (Gaby et al., 2022). The neural network used is noted $\boldsymbol{\phi}_{\boldsymbol{\theta}}$, its output size is $m \in \mathbb{N}^*$, and we fix:

$$V_{\boldsymbol{\theta}}(\mathbf{x}) = \|\boldsymbol{\phi}_{\boldsymbol{\theta}}(\mathbf{x}) - \boldsymbol{\phi}_{\boldsymbol{\theta}}(0)\|^2 + \alpha \log(1 + \|\mathbf{x}\|^2) \quad (5)$$

where the parameter $\alpha > 0$ can be fixed by the user. The neural network $\boldsymbol{\phi}_{\boldsymbol{\theta}}$ can be arbitrarily constructed as discussed in (Gaby et al., 2022). Any other function $\gamma : \mathbb{R}^m \rightarrow \mathbb{R}^+$ can be used to replace the term $\|\boldsymbol{\phi}_{\boldsymbol{\theta}}(\mathbf{x}) - \boldsymbol{\phi}_{\boldsymbol{\theta}}(0)\|^2$ by $\gamma(\boldsymbol{\phi}_{\boldsymbol{\theta}}(\mathbf{x}) - \boldsymbol{\phi}_{\boldsymbol{\theta}}(0))$, as long as $\gamma(0) = 0$ and γ is locally convex twice differentiable at 0.

Proposition. The function defined in (5) is locally convex in 0 if network activation functions are twice differentiable in 0.

Proof. It is sufficient to prove the positivity of the Hessian at 0. We have:

$$\nabla V_{\boldsymbol{\theta}}(\mathbf{x}) = 2(\boldsymbol{\phi}_{\boldsymbol{\theta}}(\mathbf{x}) - \boldsymbol{\phi}_{\boldsymbol{\theta}}(0))^{\top} \nabla \boldsymbol{\phi}_{\boldsymbol{\theta}}(\mathbf{x}) + \frac{2\alpha \mathbf{x}^{\top}}{1 + \|\mathbf{x}\|^2}$$

$$\begin{aligned} \nabla^2 V_{\boldsymbol{\theta}}(\mathbf{x}) &= 2 \left((\nabla \boldsymbol{\phi}_{\boldsymbol{\theta}}(\mathbf{x}))^{\top} \nabla \boldsymbol{\phi}_{\boldsymbol{\theta}}(\mathbf{x}) \right. \\ &\quad \left. + (\boldsymbol{\phi}_{\boldsymbol{\theta}}(\mathbf{x}) - \boldsymbol{\phi}_{\boldsymbol{\theta}}(0))^{\top} \otimes \nabla^2 \boldsymbol{\phi}_{\boldsymbol{\theta}}(\mathbf{x}) \right) \\ &\quad + \frac{2\alpha \mathbf{I}_n}{1 + \|\mathbf{x}\|^2} - \frac{4\alpha \mathbf{x} \mathbf{x}^{\top}}{(1 + \|\mathbf{x}\|^2)^2} \end{aligned}$$

with \otimes a tensor contraction. Assuming $\mathbf{x} = 0$, we have:

$$\nabla^2 V_{\theta}(0) = 2(\nabla \phi_{\theta}(0))^{\top} \nabla \phi_{\theta}(0) + 2\alpha \mathbf{I}_n \succ 0$$

Regardless of the activation functions used and the network architecture in ϕ_{θ} , the Hessian of V_{θ} is positive definite for any $\alpha > 0$. In other words, V_{θ} is convex near equilibrium when differentiable activation functions are used.

This proof can be extended to any γ function introduced previously.

The cost function used for the training contains three different terms:

$$\mathcal{L}_{\theta} = c_1 \mathcal{L}_1 + c_2 \mathcal{L}_2 + c_3 \mathcal{L}_3 \quad (6)$$

with $c_1, c_2, c_3 > 0$ user-fixed hyperparameters once and for all.

The first term \mathcal{L}_1 is used to force \dot{V}_{θ} to be strictly negative in the largest possible set within the domain \mathbb{X} . It is similar to the formulation commonly proposed in literature about neural Lyapunov functions:

$$\mathcal{L}_1 = \frac{1}{N_1} \sum_{i \in \mathcal{X}} \max(0, \dot{V}_{\theta}(\mathbf{x}_i)) \quad (7)$$

where $\mathcal{X} = \{1 \leq i \leq N_1, \mathbf{x}_i \in \mathbb{X}\}$ is a subset of state vectors picked in \mathbb{X} and $N_1 = |\mathcal{X}|$.

This formulation benefits samples with large amplitudes rather than those close to the equilibrium. Since we may not know the true RoA of the system before training, some of the term $\dot{V}_{\theta}(\mathbf{x}_i)$ may prevent convergence to 0, so the term \mathcal{L}_1 is not necessarily sufficient to have a good accuracy of the result near the equilibrium.

To ensure \dot{V}_{θ} to be strictly negative near 0, the term \mathcal{L}_2 is introduced. It consists of a Manhattan distance between \dot{V}_{θ} and a negative definite locally concave function $h: \mathbb{R}^n \rightarrow \mathbb{R}^-$:

$$\mathcal{L}_2 = \frac{1}{N_1} \sum_{i \in \mathcal{X}} |\dot{V}_{\theta}(\mathbf{x}_i) - h(\mathbf{x}_i)| \quad (8)$$

In this paper, the function h chosen is:

$$h(\mathbf{x}) = -\beta \log(1 + \|\mathbf{x}\|^2) \quad (9)$$

where the parameter $\beta > 0$ can be fixed by the user. Any other function can be used, as long as it is null at 0, negative definite near 0 and locally concave. However $\log(1 + \|\mathbf{x}\|^2) \approx \|\mathbf{x}\|^2$ as $\mathbf{x} \rightarrow 0$ and grows very slowly as $\|\mathbf{x}\|$ increases, which makes it really suitable for the task.

For a candidate V_{θ} estimated for all $\mathbf{x}_i \in \mathcal{X}$, one can compute an approximate level set of V_{θ} :

$$\tilde{c} = \min_{i \in \mathcal{X}} V_{\theta}(\mathbf{x}_i) \text{ s.t. } \dot{V}_{\theta}(\mathbf{x}_i) > 0 \quad (10)$$

If any of the \mathbf{x}_i verify the condition on the time derivative, we take $\tilde{c} = \infty$.

Finally, in order to find the largest RoA estimate at the end of the training, we add the term \mathcal{L}_3 defined as follow:

$$\mathcal{L}_3 = 1 - \frac{N_2}{N_1} \quad (11)$$

where $N_2 = |\mathcal{A}|$ and $\mathcal{A} = \{\mathbf{x}_i \in \mathcal{X}, V(\mathbf{x}_i) < \tilde{c}\}$. (11) tends towards 0 the more points are included in the RoA estimate.

The best case is obtained when the system considered is stable for all $\mathbf{x} \in \mathbb{X}$, i.e. the cost function \mathcal{L}_{θ} (6) can converge to 0. Generally speaking, we arrange the terms of the cost functions using constant terms c_1, c_2, c_3 . As \mathcal{L}_1 (7) is the only term that depends on the system dynamics through \dot{V}_{θ} , while \mathcal{L}_3 (11) is of interest only if the other two terms impact the training enough to exploit an estimate of the RoA, we usually set $c_1 > c_2 > c_3$ during training. We can then get rid of c_1 as an hyperparameter by fixing it to 1 and then fix c_2, c_3 such that $c_3 < c_2 < 1$.

Any optimization process can be used for training as long as it can minimize \mathcal{L}_{θ} . The pseudocode of an algorithm iteration is provided in Algorithm 1.

Algorithm 1: Single Network Approach Iteration.

-
- 1: **Input:** network ϕ_{θ} , dynamical system f , parameters α, β , hyperparameters c_1, c_2, c_3 , set $\mathcal{X} \subset \mathbb{X}$
 - 2: Compute candidates $V_{\theta}(\mathcal{X})$ and $\dot{V}_{\theta}(\mathcal{X})$
 - 3: Compute approximate level set \tilde{c} (10)
 - 4: Compute cost function \mathcal{L}_{θ} (6)
 - 5: Update weights of ϕ_{θ} to minimize \mathcal{L}_{θ}
 - 6: **Return:** Updated network ϕ_{θ}
-

In section 4, we describe a gradient descent training algorithm 3 implemented in Matlab to train the network, however, other algorithms can be picked to find the optimal vector of parameters θ .

3.2 Two-Network Approach

In order to make the method independent of the choice of h (9) and thus give ourselves every chance of maximizing the RoA, we propose to use a secondary neural network as an optimal function h to approximate the Lyapunov candidate time derivative.

We write θ_1 the parameter vector and ϕ_{θ_1} the neural network of the Lyapunov candidate V_{θ_1} . The parameter vector of the second network is noted θ_2 , this network is then noted ϕ_{θ_2} , and the time derivative approximator is noted H_{θ_2} . The structure of this approximator is negative definite by design:

$$H_{\theta_2}(\mathbf{x}) = -\|\phi_{\theta_2}(\mathbf{x}) - \phi_{\theta_2}(0)\|^2 - \beta \log(1 + \|\mathbf{x}\|^2) \quad (12)$$

where the parameter $\beta > 0$ is a once and for all user-fixed parameter. H_{θ_2} is locally concave near the equilibrium if the activation functions used for the network ϕ_{θ_2} are differentiable, as demonstrated in subsection 3.1. β is deliberately noted in the same way as in (9), as we fix a unique value for it in order to consider H_{θ_2} as an optimal estimator of h .

The cost term \mathcal{L}_2 (8) is then replaced by \mathcal{L}_2' :

$$\mathcal{L}_2' = \frac{1}{N_2} \sum_{i \in \mathcal{X}} |\dot{V}_{\theta_1}(\mathbf{x}_i) - H_{\theta_2}(\mathbf{x}_i)| \quad (13)$$

The cost function used for the training is noted \mathcal{L}_{Θ} with $\Theta = [\theta_1; \theta_2]$. It is equal to:

$$\mathcal{L}_{\Theta} = c_1 \mathcal{L}_1 + c_2 \mathcal{L}_2' + c_3 \mathcal{L}_3 \quad (14)$$

The pseudocode of an algorithm iteration is provided in Algorithm 2.

Algorithm 2: Two-Network Approach Iteration.

- 1: **Input:** networks $\phi_{\theta_1}, \phi_{\theta_2}$, dynamical system f , parameters α, β , hyperparameters c_1, c_2, c_3 , set $\mathcal{X} \subset \mathbb{X}$
 - 2: Compute candidates $V_{\theta_1}(\mathcal{X})$, $\dot{V}_{\theta_1}(\mathcal{X})$ and $H_{\theta_2}(\mathcal{X})$
 - 3: Compute approximate level set \tilde{c} (10)
 - 4: Compute cost function \mathcal{L}_{Θ} (14)
 - 5: Update weights of ϕ_{θ_1} and ϕ_{θ_2} to minimize \mathcal{L}_{Θ}
 - 6: **Return:** Updated networks $\phi_{\theta_1}, \phi_{\theta_2}$
-

Although this formulation has some obvious shortcomings, including the increase in parameters involved, and thus a longer computation time, these parameters are not necessarily prohibitive if a suitable Lyapunov function is to be found for the system under consideration.

4 EXAMPLES

In the following section, we demonstrate that the proposed methods are efficient through numerical experiments.

4.1 Training Method

The neural network function is trained using Matlab Automatic Differentiation and the standard ADAM network training algorithm (Kingma and Ba, 2017), from the R2023A Matlab Deep Learning Toolbox. The documentation for the `adamupdate` Matlab function is available at (Matlab, 2019).

In order to compute a gradient that can effectively influence the training direction, an equivalence

of $N_2 = |\mathcal{A}|$ directly related to θ (resp. θ_1) is used in (11). It is noted \tilde{N}_2 and is equal to:

$$\tilde{N}_2 = \sum_{i \in \mathcal{X}} \max(0, \sigma(\tilde{c} - V_{\theta}(\mathbf{x}_i))) \quad (15)$$

with σ the sigmoid function.

In general, the number of neurons per layer is determined by the dimension of the system, the number of hidden layers is first set at 1, then increased if necessary, and the networks parameters are bounded between -1 and 1 . Lyapunov candidates are judged according to the size of their largest RoA estimate Ω_c as defined in section 2.

The pseudocode of the algorithm proposed is provided in Algorithm 3. In order to have sufficient training data, we use a grid of initial conditions $\mathbf{X}_0 \subset \mathbb{X}$. For each \mathbf{x}_0 in \mathbf{X}_0 , the system studied is simulated with Simulink, and the trajectory is collected into \mathbf{X} . We define n_E as the number of epochs and n_I as the number of iterations per epoch. Since there is no guarantee that the cost function will converge, we fix the total number of iterations $N_I = n_E \times n_I$. For each epoch the table \mathbf{X} is shuffled and n_I is fixed so that:

$$\mathbf{X} := \bigsqcup_{j=1}^{n_I} \mathbf{X}_{b,j} \quad (16)$$

where $\mathbf{X}_{b,j}$ are training data batches of size N_I .

At each iteration, update is driven by the cost function gradient:

- for the single network approach we compute $\nabla_{\theta} \mathcal{L}_{\Theta}$ to update ϕ_{θ} ;
- for the two-network approach we compute $\nabla_{\theta_1} \mathcal{L}_{\Theta}$ and $\nabla_{\theta_2} \mathcal{L}_{\Theta}$ to update ϕ_{θ_1} and ϕ_{θ_2} respectively.

4.2 Parameter Setting

In all the examples, a learning rate of 0.01 is chosen for the training of the network modeling V_{θ} . The learning rate of the second neural network is fixed to 0.02 when used to match \dot{V}_{θ} quickly, especially during the first iterations, as empirically observed during the algorithm implementation.

Each network has the same structure with tanh activation functions and a single hidden layer. The hidden layer neurons number is fix to $4n$ and the output dimension is fix to $2n$ as this allows a suitable optimization time and sufficiently significant results for the study of the examples considered. For a two-dimensional system, i.e. $n = 2$, we then have a total of 60 parameters to train by network.

As c_1 is commonly fixed to 1, hyperparameters c_2, c_3 can be fixed by trial and error and their common values are $c_2 = 0.1$ and $c_3 = 0.01$. Settings α and β

Algorithm 3: Network Training Implementation.

1: **Input:** dynamical system f , parameters α, β , hyperparameters c_1, c_2, c_3 , initial set \mathbf{X}_0 of sampled states in \mathbb{X} , number of epochs n_E , number of iterations per epoch n_I , batch size N_1

2: Initialize Φ_{θ} (resp. $\Phi_{\theta_1}, \Phi_{\theta_2}$)

3: **for all** $\mathbf{x}_0 \in \mathbf{X}_0$ **do**

4: Simulate the trajectory initialized at \mathbf{x}_0

5: Update data table \mathbf{X} with measures $\mathbf{x}(t)$

6: **end for**

7: **for** $i = 0, 1, \dots, n_E$ **do**

8: Shuffle \mathbf{X}

9: **for** $j = 0, 1, \dots, n_I$ **do**

10: Get a batch $\mathbf{X}_{b,j}$ from \mathbf{X} of size N_1

11: Compute algorithm 1 or 2 with $\mathcal{X} = \mathbf{X}_{b,j}$

12: **end for**

13: **end for**

14: **Return:** Φ_{θ} (resp. $\Phi_{\theta_1}, \Phi_{\theta_2}$)

are also fixed by trial and error and then kept constant for each of the systems considered. Table 1 lists the selected values.

 Table 1: (α, β) setting.

Benchmarks	Example 1	Example 2
α	0.1	0.1
β	0.1	1

Table 2 includes information about the data set used for training and the fixed number of iteration.

Table 2: $|\mathbf{X}_0|$ — $|\mathbf{X}_0|$ — $|\mathbf{X}_0|$ is the number of initial conditions considered, $|\mathbf{X}_0|$ — $|\mathbf{X}|$ — $|\mathbf{X}_0|$ is the number of training data obtained with Simulink. $|\mathbf{X}_0|N_I$ — $|\mathbf{X}_0|$ is the number of training iterations, and $|\mathbf{X}_0|N_1$ — $|\mathbf{X}_0|$ is the batch size fixed for each of the examples considered.

Benchmarks	Example 1	Example 2
$ \mathbf{X}_0 $	1681	961
$ \mathbf{X} $	9.9×10^5	4.7×10^5
N_I	270	270
N_1	1×10^5	5×10^4

4.3 Experiments

The two systems considered aims to illustrate the ability of our methods to treat general nonlinear systems.

The first example is a differentiable two-dimensional nonlinear system. We compare our results with another training method based on the Hessian-based approach described in (Bocquillon et al., 2022) and, to properly compare methods, the cost term \mathcal{L}_3 (11) is added to the training framework suggested:

Evolution of the Training Process. Let Q and $\bar{\lambda}$ be as defined in (Bocquillon et al., 2022). The cost function Q is rewritten as follow:

$$Q = \bar{\lambda} + \mathcal{L}_3 \quad (17)$$

The best case is obtained when the system considered is stable for all $\mathbf{x} \in \mathbb{X}$, i.e. $Q < 0$ after training. Generally speaking, the Lyapunov candidate obtained is valid if $\bar{\lambda} < 0$ at the end of the training.

The second example is a 2d-system defined to demonstrate briefly the ability of our methods to find a suitable Lyapunov function even if the system is not differentiable.

Both examples are defined so that a large part of their domain of definition is unstable. Our goal is to test the ability of our methods to find a suitable Lyapunov function even if part of the data considered during training are part of unstable trajectories.

Example 1.

$$\begin{cases} \dot{x}_1 = -x_1 + x_2 + \frac{1}{2}(e^{x_1} - 1) \\ \dot{x}_2 = -x_1 - x_2 + x_1 x_2 + x_1 \cos x_1 \end{cases} \quad (18)$$

The domain considered is $\mathbb{X} = [-6; 6]^2$ and 0 is a stable equilibrium.

In Figure 1, Regions of Attraction (RoA) of Lyapunov functions obtained by the single network method and the two-network method, as long as Hessian-based approach, are compared. The regions obtained are not included in the display window, which is limited to the size of \mathbb{X}_0 , as trajectories considered during training may leave the window before converging to 0. In view of this result, our training methods can be considered efficient, since they allow us to find Lyapunov functions with regions of attraction close to the true region of attraction. As we have not precisely optimized the parameter settings, we can hope that with an hyperparameter optimization method we can obtain an even more accurate estimate of the true RoA.

Lyapunov functions obtained for the single-network method and the two-network method, and their time derivative functions, are respectively displayed in Figures 2a and 2b. As can be seen, 0 is a local minimum for V_{θ} , and a local maximum for \dot{V}_{θ} , on the points grid considered for display.

In order to compare single network and two-network approaches, Figure 3 shows functions h (10) and H_{θ_2} (12). As described in section 3, the second is defined by training, whereas the first is not. As we obtained a similar RoA for the two methods, it is difficult to conclude here about a possible advantage of

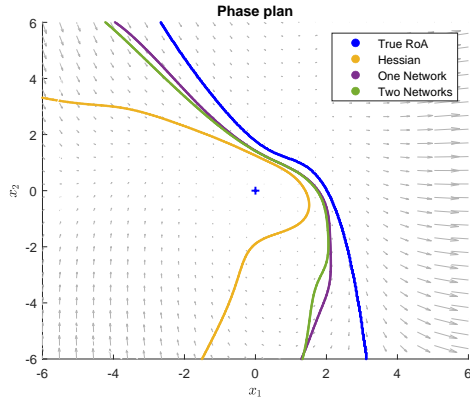


Figure 1: RoA comparison - Example 1.

using the two-network approach instead of the single one. However, we can emphasize the supposed advantage of using it, since H_{θ_2} is visually closer to \hat{V}_{θ_1} in Figure 2b than h to \hat{V}_{θ} in Figure 2a. Final values of cost terms \mathcal{L}_2 (8) and \mathcal{L}'_2 (13), estimated for all $\mathbf{x} \in \mathbf{X}$, provide a numerical overview of the difference between the two approaches. We have $\mathcal{L}_2 = 7.7 \times 10^{-4}$ and $\mathcal{L}'_2 = 4.4 \times 10^{-3}$, which confirm the visual observation. The performance measurement study detailed in section 4.4 would allow us to conclude.

Example 2.

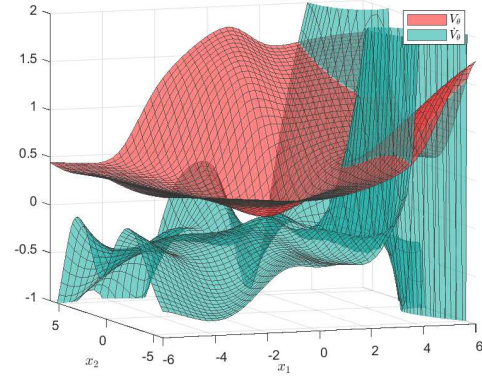
$$\begin{cases} \dot{x}_1 = -\left(x_1 - \frac{\sqrt{2}}{2}\right) \left|x_1 - \frac{\sqrt{2}}{2}\right| + e^{x_2 - \frac{\sqrt{2}}{2}} - 1 \\ \dot{x}_2 = -\left(x_2 - \frac{\sqrt{2}}{2}\right) \left|x_2 - \frac{\sqrt{2}}{2}\right| + e^{x_1 - \frac{\sqrt{2}}{2}} - 1 \end{cases} \quad (19)$$

The domain considered is $\mathbb{X} = [-4; 4]^2$ and $x_0 = [-7.4 \cdot 10^{-3}; -7.4 \cdot 10^{-3}]$ is a stable equilibrium.

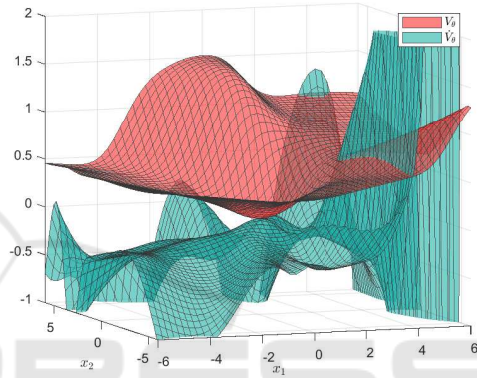
In Figure 4, Regions of Attraction (RoA) of Lyapunov functions obtained by the single network method and the two-network method are compared. The second method seems to be more efficient but we will see in section 4.4 if it comes from a better random seed at initialization or a more efficient approach.

Lyapunov functions obtained for the single-network method and the two-network method, and their time derivative functions, are respectively displayed in Figures 5a and 5b, and the same conclusion as before can be made about their values at the equilibrium.

Figure 6 shows functions h (10) and H_{θ_2} (12), and we apply the same argumentation as for example 1. H_{θ_2} is visually closer to \hat{V}_{θ_1} in Figure 5b than h to \hat{V}_{θ} in Figure 5a. For all $\mathbf{x} \in \mathbf{X}$, we have $\mathcal{L}_2 = 0.1531$ and $\mathcal{L}'_2 = 0.0478$, which confirm the visual observation. We rely on section 4.4 to compare the effectiveness of the two approaches.



(a) Single Network Approach.



(b) Double Network Approach.

Figure 2: Lyapunov functions comparison - Example 1.

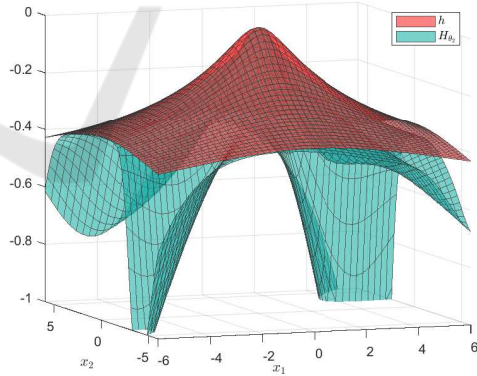


Figure 3: Estimator comparison - Example 1.

4.4 Performance Measurement

Due to the stochastic nature of our methods, we subject each system to a series of 50 successive runs using the same training data set and different seeds in order to test whether their results are parameter-dependent.

We list the minimum value t_{min} , the average value t_{mean} and the standard deviation t_{std} of the training

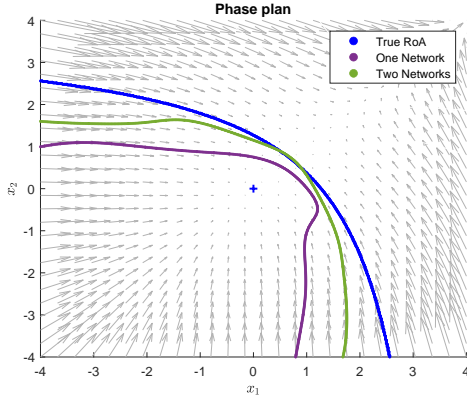


Figure 4: RoA comparison - Example 2.

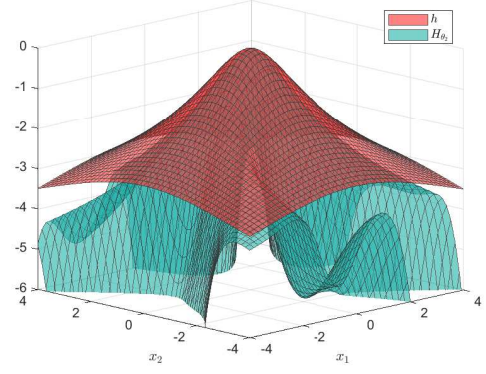
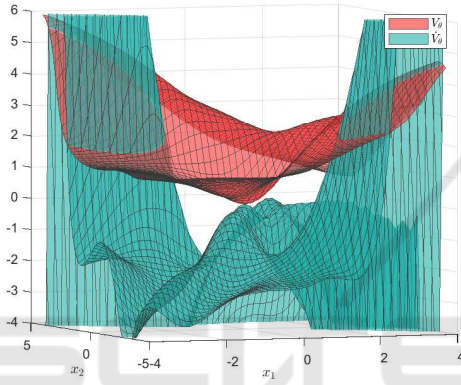
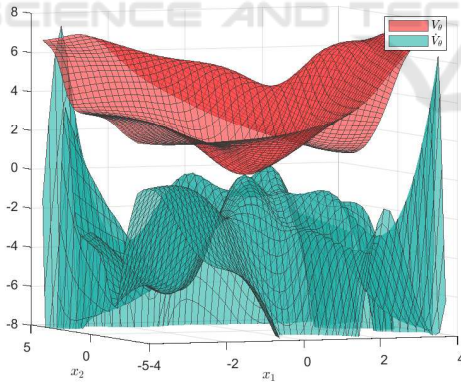


Figure 6: Estimator comparison - Example 2.



(a) Single Network Approach.



(b) Double Network Approach.

Figure 5: Lyapunov functions comparison - Example 2.

time obtained, the minimum value \mathcal{L}_{min} , the average value \mathcal{L}_{mean} and the standard deviation \mathcal{L}_{std} of the final cost function obtained, as long as the maximum value D_{max} , the average value D_{mean} and the standard deviation D_{std} of the size of the RoA obtained as a percentage compared to the size of the initial set \mathbf{X}_0 in Table 3 for the single network method. The same results are presented in Table 4 for the two-network

Table 3: Single Network Approach Performance Measurement.

Benchmarks	Example 1	Example 2
t_{min} (s)	21.59	9.49
t_{mean} (s)	22.35	10.77
t_{std} (s)	1.34	2.24
\mathcal{L}_{min}	6.1×10^{-3}	2.64×10^{-2}
\mathcal{L}_{mean}	6.6×10^{-3}	3.05×10^{-2}
\mathcal{L}_{std}	6.8×10^{-4}	1.9×10^{-3}
D_{max} (%)	82.7	71.6
D_{mean} (%)	19.2	32.4
D_{std} (%)	13.35	21.9

method. The results presented in section 4.3 correspond to the D_{max} values.

The difference in execution times between the two examples can be linked to the number of training data items considered, as shown in Table 2. As the number of iterations is equally fixed for each run, it is consistent to obtain low standard deviations for \mathcal{L} and t .

The evolution of D_{mean} between the two approaches is significant, with an increase of 5.5% for example 1 and 11.9% for example 2, which tends to validate the advantages of the two-network approach. If the differentiability of the system considered is unknown or uncertain, the second method is therefore recommended. Since this approach requires twice the parameters to be trained, the average calculation time is, as expected, the shortcoming.

5 CONCLUSIONS

In conclusion, this paper addresses the challenge of validating Lyapunov properties without explicitly computing the Hessian eigenvalues and proposes two approaches to so. By applying them to examples, the paper demonstrates the effectiveness of the process in validating the Lyapunov principle. A compar-

Table 4: Two-Network Approach Performance Measurement.

Benchmarks	Example 1	Example 2
t_{min} (s)	32.43	16.77
t_{mean} (s)	33.90	18.33
t_{std} (s)	2.33	1.74
\mathcal{L}_{min}	5.4×10^{-3}	2.53×10^{-2}
\mathcal{L}_{mean}	5.8×10^{-3}	2.91×10^{-2}
\mathcal{L}_{std}	3.6×10^{-4}	2.1×10^{-3}
D_{max} (%)	77.1	86.2
D_{mean} (%)	24.7	44.3
D_{std} (%)	15.8	24.5

ison with a Hessian-based approach confirms the efficiency of the proposed training scheme. The ability to validate Lyapunov functions without explicitly computing the Hessian provides a valuable alternative, especially in cases where a differentiable expression of the system dynamics is not readily available.

This research opens up possibilities for broader application of the Lyapunov principle, making it more accessible and applicable in various domains. Further investigations can explore the extension of those approaches to more complex systems and practical scenarios, ensuring their robustness and reliability, or can focus on hyperparameters optimization to increase the probability to find the largest Region of Attraction for the system considered.

REFERENCES

- Bocquillon, B., Feyel, P., Sandou, G., and Rodriguez-Ayerbe, P. (2022). A comprehensive framework to determine lyapunov functions for a set of continuous time stability problems. In *IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–6.
- Chang, Y.-C. and Gao, S. (2021). Stabilizing Neural Control Using Self-Learned Almost Lyapunov Critics. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1803–1809.
- Chang, Y.-C., Roohi, N., and Gao, S. (2020). Neural Lyapunov Control. *arXiv:2005.00611 [cs, eess, stat]*.
- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.
- Gaby, N., Zhang, F., and Ye, X. (2022). Lyapunov-net: A deep neural network architecture for lyapunov function approximation. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 2091–2096.
- Giesl, P. and Hafstein, S. (2015). Review on computational methods for Lyapunov functions. *Discrete and Continuous Dynamical Systems - B*, 20(8):2291.
- Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257.
- Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366.
- Khalil, H. K. (2001). *Nonlinear Systems*. Pearson, Upper Saddle River, NJ, 3 edition.
- Kingma, D. P. and Ba, J. (2017). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*.
- Liang, S. and Srikant, R. (2017). Why deep neural networks for function approximation? *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.
- Matlab (2019). Update parameters using adaptive moment estimation (Adam) - MATLAB adamupdate.
- Mawhin, J. (2005). *Alexandr Mikhailovich Liapunov, The general problem of the stability of motion (1892)*, pages 664–676. Elsevier.
- Petridis, V. and Petridis, S. (2006). Construction of neural network based lyapunov functions. *IEEE International Conference on Neural Networks - Conference Proceedings*, pages 5059 – 5065.
- Prokhorov, D. (1994). A Lyapunov machine for stability analysis of nonlinear systems. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 2, pages 1028–1031 vol.2.
- Richards, S. M., Berkenkamp, F., and Krause, A. (2018). The Lyapunov Neural Network: Adaptive Stability Certification for Safe Learning of Dynamical Systems. *arXiv:1808.00924 [cs]*.