Extended Abstract: Control Lyapunov Functions for Reinforcement Learning and Adaptive Control

Motivation Traditional Reinforcement Learning (RL) approaches to nonlinear control face fundamental challenges in providing stability guarantees while achieving optimal performance. Standard RL methods require learning both value functions and control policies simultaneously, leading to complex optimization landscapes and convergence difficulties. Our work addresses this by leveraging Control Lyapunov Functions (CLFs) with Sontag's universal control formula, which enables direct synthesis of optimal controllers from learned CLFs alone, eliminating the need for separate policy optimization while providing rigorous stability and optimality guarantees through Hamilton-Jacobi-Bellman (HJB) theory.

Method We develop a counterexample-guided neural learning framework that departs from traditional RL paradigms by learning only CLFs rather than both value functions and policies. The core insight is that Sontag's universal formula provides optimal control when the CLF satisfies specific scaling relationships to the true optimal value function. Our dual-dataset training strategy separates stability and optimality objectives: a uniform base dataset ensures global coverage for optimality constraints derived from HJB theory, while an evolving counterexample dataset targets regions where current candidates violate stability conditions. This separation resolves the conflicting gradients that typically arise when optimizing multiple control objectives. The framework integrates automatic differentiation for efficient Lie derivative computation and synthesizes controllers through the universal control formula without policy learning.

Implementation The implementation employs a feedforward neural network with 256 hidden neurons and no output activation to represent unbounded Lyapunov functions. Training alternates between risk loss evaluation on the complete dataset (enforcing positive definiteness, stability, and regularization) and optimality loss evaluation exclusively on the uniform base set. The counterexample detection system periodically samples the domain, identifies stability violations, and maintains an aging counterexample pool that focuses training on problematic regions. Convergence is achieved when three consecutive verification cycles produce no new counterexamples.

Results Experimental validation on inverted pendulum stabilization demonstrates convergence success rates above 95% within the training domain, with learned controllers exhibiting near-optimal performance as measured by the optimality loss function. The approach successfully learns CLFs that satisfy both stability constraints and optimality conditions. Sample trajectories demonstrate stable convergence to the origin with settling times of 2-4 seconds and bounded control effort. The dual-dataset approach effectively resolves the competing optimization objectives, with risk loss converging to near-zero values while maintaining low optimality loss, indicating successful achievement of both stability and optimality without explicit policy learning.

Discussion The separation of training objectives across distinct datasets proved crucial for managing conflicting gradients between stability and optimality constraints. When using a single combined loss function, we observed that heavily weighting risk terms achieved stability but prevented optimality convergence, while emphasizing optimality terms compromised stability guarantees. The counterexample-guided approach demonstrates superior sample efficiency compared to uniform sampling, with the aging mechanism preventing overfitting to stability violations. The elimination of policy learning simplifies the optimization landscape while providing stronger theoretical guarantees than actor-critic methods. Limitations include domain-restricted guarantees, scaling challenges for high-dimensional systems, and sensitivity to hyperparameter selection.

Conclusion This work establishes a theoretically grounded framework connecting neural CLF learning to optimal control through Sontag's universal formula, demonstrating that learning CLFs alone is sufficient for both stability and optimality without requiring explicit policy learning. The dual-dataset training innovation resolves fundamental optimization conflicts in multi-objective control learning, while the counterexample-guided refinement ensures targeted improvement in critical stability regions. Key contributions include: (1) rigorous integration of neural learning with control theory providing formal stability guarantees, (2) proof that Sontag's formula preserves optimality when CLFs approximate scaled optimal value functions, (3) novel dual-dataset approach separating competing objectives, and (4) comprehensive implementation suitable for extension to broader nonlinear control applications.

Control Lyapunov Functions for Reinforcement Learning and Adaptive Control

Alex Leonessa
CS224R: Deep Reinforcement Learning
Stanford University
`leonessa@stanford.edu`

Jordan Berg
CS224R: Deep Reinforcement Learning
Stanford University
`jmberg59@stanford.edu`

Abstract

We present a counterexample-guided neural approach for learning Control Lyapunov Functions (CLFs) in nonlinear control systems that departs from traditional Reinforcement Learning (RL) paradigms. Unlike standard methods requiring both value functions and policies, our framework uses Sontag's universal formula to synthesize optimal controllers directly from learned CLFs, eliminating policy optimization while providing stability and optimality guarantees through Hamilton-Jacobi-Bellman (HJB) connections. The key innovation separates stability and optimality objectives across two datasets: uniform base sampling for global coverage and evolving counterexamples targeting stability violations. This dual-dataset strategy resolves conflicting gradients in multi-objective optimization. When CLFs satisfy specific scaling relationships to optimal value functions, Sontag's formula automatically produces optimal controllers. The framework uses automatic differentiation for Lie derivatives and synthesizes controllers without policy learning. Experimental validation on inverted pendulum shows 95%+ success rates with near-optimal performance measured by HJB-derived optimality loss. Main contributions include (1) connecting neural CLF learning to optimal control through scaling relationships, (2) proving Sontag's formula preserves optimality for scaled CLFs, (3) resolving competing objectives via dataset separation, and (4) achieving formal guarantees without policy learning.

1 Introduction

Stabilizing nonlinear dynamical systems while ensuring optimal performance remains one of the most challenging problems in control theory, particularly when seeking methods that can handle complex dynamics without requiring explicit policy learning. Traditional reinforcement learning approaches attempt to learn both value functions and control policies simultaneously, often leading to complex optimization landscapes and convergence difficulties. In contrast, the theoretical framework we develop here demonstrates that learning only a Control Lyapunov Function (CLF) is sufficient to obtain both stable and optimal controllers through Sontag's universal control formula.

The key theoretical insight driving our approach is the recognition that Sontag's universal formula provides more than just a stabilizing controller, it actually yields optimal control when the CLF satisfies specific scaling relationships to the true optimal value function. This connection to Hamilton-Jacobi-Bellman theory means that by learning CLFs with appropriate optimality constraints, we can guarantee not only stability but also performance optimality without ever explicitly learning a policy. This represents a fundamental departure from reinforcement learning paradigms and offers significant computational advantages.

Classical control synthesis methods face well-known limitations. Linearization techniques offer computational tractability but restrict validity to small neighborhoods around operating points. Sum-of-Squares programming provides mathematical rigor but suffers from computational complexity that scales poorly with system dimension. Most critically for our purposes, these methods typically address either stability or optimality, but rarely both in a unified framework. Recent neural approaches have attempted to bridge this gap but generally require learning both value functions and policies, leading to complex multi-objective optimization problems.

Our theoretical development begins with the observation that for control-affine systems $\dot{x} = f(x) + G(x)u$, any CLF $V(x)$ can be related to the optimal value function $V^*(x)$ through a scaling function $\lambda(x)$. When this scaling relationship holds, Sontag's universal control formula automatically produces the controller that is optimal for a modified cost function. This insight allows us to formulate the learning problem as finding CLFs that satisfy both stability constraints and optimality conditions derived from the Hamilton-Jacobi-Bellman equation.

The practical implementation of this theory requires careful handling of the multiple constraints involved. We develop a dual-dataset training strategy that separates stability and optimality objectives across different data distributions. The uniform base dataset ensures global coverage for optimality constraints, which require unbiased sampling to properly represent the cost-to-go structure. Meanwhile, the counterexample dataset focuses on regions where stability is violated, providing targeted training for the most critical constraints.

A crucial advantage of our approach is the elimination of policy learning entirely. Traditional RL methods must learn both when to take actions (the value function) and what actions to take (the policy), leading to complex actor-critic architectures and challenging credit assignment problems. In our framework, once the CLF is learned, Sontag's formula provides the optimal policy directly through a closed-form expression involving only the CLF gradients. This dramatically simplifies the learning problem while providing stronger theoretical guarantees.

The theoretical foundations also enable us to derive a principled optimality loss function based on the Hamilton-Jacobi-Bellman equation. When this loss approaches zero, we can guarantee that the learned controller approaches optimality for the specified cost structure. This provides a quantitative measure of optimality that is typically unavailable in neural RL approaches, where optimality can only be assessed empirically through performance metrics.

We validate our theoretical framework on the inverted pendulum, which serves as an excellent testbed for nonlinear control methods. The system's underactuated nature and unstable equilibrium provide the complexity needed to evaluate our approach, while its low dimensionality enables thorough analysis and visualization. Our results demonstrate that the learned controllers not only stabilize the system but also exhibit near-optimal performance as measured by the derived optimality metrics.

The integration of automatic differentiation capabilities enables efficient computation of the Lie derivatives required for both stability verification and optimality assessment. This computational efficiency, combined with the elimination of policy learning, makes our approach particularly attractive for real-time applications where computational resources are limited.

Beyond the core theoretical contributions, our implementation emphasizes the engineering aspects necessary for practical deployment. The modular design facilitates extension to new systems while maintaining the theoretical foundations that guarantee both stability and optimality without requiring policy learning phases that characterize traditional RL approaches.

2 Mathematical Preliminaries

The contents of this section may be found in the literature in a variety of formulations. They are included here to provide a consistent framework for derivation of the main results.

2.1 Nonlinear Control Systems

A large class of nonlinear dynamic systems is defined by the initial value formulation

$$\dot{x} = F(x);\; x(t_0) = x_0\,. \tag{1}$$

Here, knowledge of the *state vector* x_0 at time t_0 completely characterizes the solution $x(t)$ for all $t > 0$. Subsequent stability analysis further assumes that the origin is an equilibrium point of the system, that is, $F(0) = 0$.

Nonlinear *control* systems add an additional variable, the exogenous *control action*, $u(t)$,

$$\dot{x} = F(x, u)\,. \tag{2}$$

where $u(t)$ is assumed to be precisely specifiable at any time. For purposes of stability analysis it will be subsequently assumed that the origin is an equilibrium of the uncontrolled system, that is, $F(0, 0) = 0$.

The specific results of this project are obtained for the special case of *control-affine* systems,

$$\dot{x} = f(x) + G(x)u\,, \tag{3}$$

with $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^p$. Control-affine systems represent many interesting applications, and are a reasonable compromise between generality and mathematical tractability. Here the requirement $F(0, 0) = 0$ reduces to $f(0) = 0$.

The requirements that $F(0) = 0$, $F(0, 0) = 0$, or $f(0) = 0$, are not difficult to generalize to other equilibria points, but the extension to stabilization of dynamic solutions like limit cycles is non-trivial and beyond the scope of this project. Perhaps the most consequential simplifying assumption made here is that the full state x is available for feedback at any time. A more realistic model would include a set of measurements or observations y, related to the state by $y = h(x)$. Another useful extension would be to include an exogenous, possibly stochastic, disturbance d as $\dot{x} = f(x, d) + G(x)u$. More substantial and mathematically challenging extensions could include models described by partial differential equations or stochastic differential equations. These are subjects of future interest.

See [1] for a comprehensive introduction to nonlinear control systems.

2.2 Lyapunov Functions and Control Lyapunov Functions

Definition 2.1 (Lyapunov Function)**.** A *Lyapunov function* for nonlinear system (1) is a positive definite function $V(x)$ with $V(0) = 0$ such that the value of $V(x)$ is strictly decreasing along solution trajectories of (1):

$$\dot{V}(x(t)) = \frac{\partial V(x)}{\partial x} F(x) < 0\,.$$

The existence of a Lyapunov function guarantees that the origin is an asymptotically stable equilibrium point of (1) [1].

The notion of a control Lyapunov function generalizes Lyapunov functions in the presence of a control input:

Definition 2.2 (Control Lyapunov Function)**.** A positive definite function $V(x)$ with $V(0) = 0$ is a control Lyapunov function (CLF) for system Eqn (2) if there exists a u such that $V(x)$ is strictly decreasing along trajectories of (2):

$$\min_u \left[\frac{\partial V(x)}{\partial x} F(x, u) \right] < 0\,.$$

The existence of a CLF guarantees that the origin is a stable equilibrium of (2), for some choice of u. It may be of interest to apply a stronger inequality on $\dot{V}(x(t)$. For example,

$$\min_u \left[\frac{\partial V(x)}{\partial x} F(x, u) \right] < -V(x) \tag{4}$$

guarantees *exponential stability* of the origin, which is stronger than asymptotic stability [1]. A CLF satisfying (4) is sometimes called an ES-CLF [2]).

For control-affine systems (3), the condition for a CLF is

$$\min_u \left[\frac{\partial V(x)}{\partial x} f(x) + \frac{\partial V(x)}{\partial x} G(x)u \right] < 0\,.$$

This condition is central to the results presented below. The following definitions are made for compactness:

$$\alpha(x) \stackrel{\text{def}}{=} L_f V(x) \stackrel{\text{def}}{=} \frac{\partial V(x)}{\partial x} f(x) \tag{5}$$

$$\beta(x) \stackrel{\text{def}}{=} L_G V(x) \stackrel{\text{def}}{=} \frac{\partial V(x)}{\partial x} G(x)\,. \tag{6}$$

Now the condition for $V(x)$ to be a CLF becomes

$$\min_u \left[\alpha(x) + \beta(x)u\right] < 0\,. \tag{7}$$

2.3 CLFs and Optimal Control

This subsection follows [3–7], with modifications as needed to unify notation and clarify the class of cost functionals and constraints under consideration.

2.3.1 Optimal Control and the HJB Equation

Consider the following optimal control problem:
Problem 2.1 (Infinite-Horizon Integral Optimal Control). For positive definite integrand $l(x,u)$, find the optimal value function $V^\star(x)$ that satisfies

$$V^\star(x_0) = \min_u \int_{t_0}^{\infty} l(x,u)\,dt \tag{8}$$

subject to

$$\dot{x} = F(x,u);\; x(t_0) = x_0\,.$$

Denote the corresponding optimal u by

$$u^\star(t) = \arg\min_u \int_{t_0}^{\infty} l(x,u)\,dt\,. \tag{9}$$

Applying Bellman's principle of optimality to the solution $V^\star(x)$ gives the Hamilton-Jacobi-Bellman (HJB) equation:

$$\min_u \left[l(x,u) + \frac{\partial V^\star(x)}{\partial x} F(x,u)\right] = 0\,, \tag{10}$$

with associated optimal control

$$u^\star = \arg\min_u \left[l(x,u) + \frac{\partial V^\star(x)}{\partial x} F(x,u)\right]\,. \tag{11}$$

The HJB equation turns the minimization of an infinite-horizon integral into a *pointwise* minimization. The HJB equation is typically infeasible to solve directly, but it can serve as a certificate to validate the optimality of a $V^\star(x)$ obtained by other means.

2.3.2 Quadratic Control Cost and Control-Affine Dynamics

A special case of interest is when the dynamics are control-affine and when the integrand $l(x,u)$ is quadratic in the control, u, that is,

$$l(x,u) = q(x) + u^T R(x) u\,. \tag{12}$$

Here $q(x) \in \mathbb{R}$ is positive definite and $R(x) \in \mathbb{R}^{p\times p}$ is positive definite and symmetric. Following (5) and (6), refer to the corresponding functions of the optimal $V^\star(x)$ by

$$\alpha^\star(x) = \frac{\partial V^\star(x)}{\partial x} f(x) \tag{13}$$

$$\beta^\star(x) = \frac{\partial V^\star(x)}{\partial x} G(x)\,. \tag{14}$$

The HJB equation (10) becomes

$$\min_u \left[q(x) + u^T R(x) u + \alpha^\star(x) + \beta^\star(x) u\right] = 0 \tag{15}$$

Taking the derivative with respect to u and setting it equal to zero gives the following optimal control:

$$u^\star = -\tfrac{1}{2} R(x)^{-1} \beta^\star(x)^T\,. \tag{16}$$

Finally, substituting for $u^\star$ in (15) and simplifying gives the following HJB condition for optimality:

$$q(x) - \tfrac{1}{4}\beta^\star(x) R(x)^{-1} \beta^\star(x)^T + \alpha^\star(x) = 0\,. \tag{17}$$

2.3.3 CLFs, Scaling, and Inverse Optimality

Any solution $V^\star(x)$ to Problem 2.1 is also a CLF for (3). This follows from the HJB equation (15), which implies

$$\dot{V}(x(t) \leq -q(x) - u^\star(x)^T R(x) u^\star(x) < 0\,.$$

It can also be shown that *any* CLF is the *inverse optimal* solution for an optimization of form 2.1 for some appropriately defined $l(x,u)$. The general case is analyzed in [8]; the subsequent analysis is restricted to CLFs that are related by a *scaling function*.

Let $V^\star(x)$ be the optimal value function satisfying (17) and consider the class of CLFs satisfying the scaling relation

$$\frac{\partial V^\star(x)}{\partial x} = \lambda(x)\frac{\partial V(x)}{\partial x} \tag{18}$$

where $\lambda(x) \in \mathbb{R}$ is strictly positive.

Substituting into the HJB equation (10) gives

$$\min_u \left[\left(\frac{l(x,u)}{\lambda(x)}\right) + \frac{\partial V(x)}{\partial x}F(x,u)\right] = 0 \tag{19}$$

That is, the scaled value function $V(x)$ is the optimal solution to an integral optimal control problem with a modified integrand, $l'(x,u) = l(x,u)/\lambda(x)$.

We specialize the scaling analysis to control-affine dynamics with quadratic control cost, as introduced in section 2.3.2. Substituting

$$\alpha^\star(x) = \lambda(x)\alpha(x)$$
$$\beta^\star(x) = \lambda(x)\beta(x)$$

into the HJB equation (17) gives

$$\tfrac{1}{4}\beta(x)R(x)^{-1}\beta(x)^T\lambda(x)^2 - \alpha(x)\lambda(x) - q(x) = 0\,. \tag{20}$$

This condition will only be satisfied by specific values of λ, as follows:

$$\lambda(x) = \begin{cases} 2\cdot\frac{\alpha(x)+\sqrt{\alpha(x)^2+q(x)\beta(x)R(x)^{-1}\beta(x)^T}}{\beta(x)R(x)^{-1}\beta(x)}^T & ;\ \beta(x)\neq 0 \\ -\frac{q(x)}{\alpha(x)} & ;\ \beta(x) = 0\end{cases}, \tag{21}$$

where the expression for $\beta(x) = 0$ (equivalent to $\beta(x)R(x)^{-1}\beta(x)^T = 0$) is found directly from (20). This $\lambda(x)$ can be substituted into the expression

$$u^\star = -\tfrac{1}{2}\lambda(x)R(x)^{-1}\beta(x)^T\,,$$

to get the control $u^{CLF}(x)$ corresponding to the scaled CLF.

$$u^{CLF}(x) = \begin{cases} -R(x)^{-1}\beta(x)^T\cdot\frac{\alpha(x)+\sqrt{\alpha(x)^2+q(x)\beta(x)R(x)^{-1}\beta(x)^T}}{\beta(x)R(x)^{-1}\beta(x)^T} & ;\ \beta(x)\neq 0 \\ 0 & ;\ \beta(x) = 0\end{cases}. \tag{22}$$

2.4 The Sontag Universal Control Formula

For a CLF $V(x)$, positive definite $q(x)$ and positive definite symmetric $R(x)$, define

$$\sigma(x) = \sqrt{\alpha(x)^2 + q(x)\beta(x)R(x)^{-1}\beta(x)^T}\,. \tag{23}$$

Now define $\lambda(x)$ and u^{CLF} as in section 2.3.3, noting that

$$\lambda(x) = \begin{cases} 2\cdot\frac{\alpha(x)+\sigma(x)}{\beta(x)R(x)^{-1}\beta(x)^T} & ;\ \beta(x)\neq 0 \\ -\frac{q(x)}{\alpha(x)} & ;\ \beta(x) = 0\end{cases} \tag{24}$$

and

$$u^{CLF} = -\tfrac{1}{2}R(x)^{-1}\beta(x)^T\lambda(x)\,. \tag{25}$$

In this context, we call (25) the *Sontag universal formula* for stabilizing control, given CLF $V(x)$ [3].

The Sontag universal controller (25) has the property that it solves the following pointwise optimization problem [4–6, 8]:

Definition 2.3 (Constrained Pointwise Control Norm Minimization)**.** Given a CLF $V(x)$ for a control-affine system (3) and positive definite function $\sigma(x)$, find the optimal feedback control $u = k(x)$ satisfying

$$k(x) = \arg\min_u u^T R(x) u \tag{26}$$

subject to

$$\dot{x} = f(x) + G(x)u \tag{27}$$

and

$$\alpha(x) + \beta(x)u \leq -\sigma(x)\,. \tag{28}$$

We can directly verify that the universal control (25) satisfies the desired negativity constraint on $\dot{V}(x(t)) = \alpha(x) + \beta(x)u$. Substituting (25) into the constraint equation gives, for $\beta \neq 0$,

$$\begin{aligned}\alpha(x) + \beta(x)u^{CLF} &= \alpha(x) - \beta(x)R(x)^{-1}\beta(x)^T \cdot \frac{(\alpha(x) + \sigma(x))}{\beta(x)R(x)^{-1}\beta(x)^T} \\ &= \alpha(x) - \alpha(x) - \sigma(x) \\ &= -\sigma(x)\,.\end{aligned}$$

When $\beta(x) = 0$ the inequality constraint is automatically satisfied as long as $\alpha(x) < 0$, which is a necessary condition for $V(x)$ to be a CLF.

3 Related Work

CLFs are a powerful tool for control system analysis and design. The introduction of the universal stabilizing control by Sontag in [3] sparked enormous enthusiasm, including the following works that have strongly influenced this project [4–9].

CLFs are well represented in methods for the combined satisfaction of stability and safety, in which the safety requirements are implemented using control barrier functions (CBFs). This work was initially implemented using quadratic programming [2, 10], and is also treated by learning these functions using neural networks [11, 12]. In the current work we have focused on the CLF, and have not considered the additional complexity caused by incorporating a CBF as well.

Previous work on learning CLFs via neural networks include [**?**, 12–15]. In the context of the present report, the most notable of these are [13–15], which use a falsification step to identify points at which the CLF criteria are violated. Our approach also uses a falsification step.

Limitations and future directions for neural Lyapunov methods are explored in [16, 17], including the stabilization of systems that do not admit any smooth CLF. The results suggest that achieving completely automated neural structures for learning CLFs will require significant effort.

4 Methods and Contributions

This section states the original theory contribution of the project. The HJB equation is generally quite hard to solve, and the optimal value function $V^\star(x)$ will typically not be available. Therefore our goal will be to first find a CLF that will provide a desired level of stability, and then to maintain that level of stability while improving the optimality of the solution. Therefore we define two loss terms. The first defines the distance from the desired level of stability, and the second describes the distance to a meaningful definition of optimality. We also describe the distribution of points on which these loss functions should be trained. To the best of our knowledge, defining these two loss functions, and the corresponding distributions, is an original contribution of this report.

4.1 Stability Loss

In section 2 we assumed that $V(x)$ was a CLF and then derived the resulting properties. In this section we are concerned with *learning* a CLF. To do so, we implement loss terms enforcing positive definiteness of $V(x)$, that is, $V(x) > 0$ when $x \neq 0$, and $V(0) = 0$, and add a term of the form

$$\alpha(x) + \beta(x)u < -\sigma(x)\,.$$

If $V(x)$ was a CLF, the Sontag formula would guarantee this constraint. However because $V(x)$ is not a CLF, trying to impose an overly stringent stability requirement might prevent convergence of the training algorithm. For example, we could try to obtain exponential stability by requiring $-\sigma(x) \leq -\gamma V(x)$, or

$$\gamma V(x) \leq \sigma(x) .$$

That implies $V(x)$ would have to satisfy

$$V(x) \leq \frac{1}{\gamma} \sqrt{\alpha(x)^2 + q(x) \left(\beta(x)^T R(x)^{-1} \beta(x)\right)} .$$

Because $\alpha(x)$ and $\beta(x)$ are constrained by the dynamics $f(x)$ and $g(x)$, which are given and unmodifiable, it may be difficult to learn a $V(x)$ to make this loss term small, even for small γ. Therefore we seek to make the stability loss term consistent with the theoretical capabilities of the controller, and propose to instead choose $\sigma(x)$ to satisfy

$$\sqrt{\gamma V(x) \beta(x)^T R(x)^{-1} \beta(x)} \leq \sigma(x) . \tag{29}$$

Then

$$\begin{aligned} \gamma V(x) \beta(x)^T R(x)^{-1} \beta(x) &\leq \sigma(x)^2 \\ &= \alpha(x)^2 + q(x) \beta(x) R(x)^{-1} \beta(x)^T \\ &\leq q(x) \beta(x) R(x)^{-1} \beta(x)^T , \end{aligned}$$

or,

$$\gamma V(x) \leq q(x) .$$

This condition does not involve $f(x)$ or $G(x)$ in any way, and can be tuned purely with adjustments of hyperparameter γ and the function $q(x)$.

Thus we propose to learn a stabilizing CLF by minimizing the following term:

Definition 4.1 (Stability Loss)**.**

$$l_{stab}(x) = \max\left(0, \alpha(x) + \beta(x) u^{CLF} + \sqrt{\gamma V(x) \beta(x)^T R(x)^{-1} \beta(x)}\right) . \tag{30}$$

The stability loss has a minimum of zero, achieved when $\alpha(x) + \beta(x) u^{CLF} + \sqrt{\gamma V(x) \beta(x)^T R(x)^{-1} \beta(x)} < 0$. The stability loss can be arbitrarily large.

4.2 Optimality Loss

Recall that we defined an optimal CLF $V^\star(x)$ that solves the infinite-horizon integral optimal control problem 2.1 with cost integrand $l(x, u) = q(x) + u^T R(x) u$. This cost functional is assumed to provide a meaningful performance measure, however the problem of directly solving for $V^\star(x)$ is infeasible.

Instead, we find a non-optimal CLF, $V(x)$, which we assume to be related to the desired optimal $V^\star(x)$ through a positive scaling function $\lambda(x)$. As shown in section 2.3.3 the desired optimal solution is recovered when $\lambda(x) = 1$. Hence we drive the system towards optimality by driving $\lambda(x)$ to one, by driving the *optimality loss*, $\Lambda(x)$ to zero. After some algebra, the expression $\Lambda(x) = \lambda(x) - 1$ simplifies to

Definition 4.2 (Optimality Loss)**.**

$$\Lambda(x) = q(x) - \tfrac{1}{4} \beta(x) R(x)^{-1} \beta(x)^T + \alpha(x) , \tag{31}$$

4.3 Training Distributions

As important as the loss functions to minimize is the choice of datasets to train on. In the case of stability, previous work in the literature incorporates a *falsification* step, to generate a set of stability training points where the CLF requirement $\alpha(x) + \beta(x) u > 0$ is violated [13–15]. In this work we generate the stabiliy training points by drawing from a distribution with probability density function,

Definition 4.3 (Stability Training Distribution)**.**

$$p_{stab}(x) = \frac{\bar{p}(x) + l_{stab}(x)}{\int_\Omega \left(\bar{p}(x) + l_{stab}(x)\right)\, dx}\,. \tag{32}$$

Here the constant term $\bar{p}(x)$ defines a uniform distribution over the region of interest Ω ensures a minimum level of sampling. The remaining term is exactly the stability loss, which is higher where the stability violations are most severe. This gives more weight in the training to those points where the function $V(x)$ is furthest from satisfying the CLF criterion.

The optimality criterion $\lambda(x) = 1$ is only meaningful when applied to a CLF. Where $V(x)$ does not satisfy the CLF stability conditions, it does not make sense to impose an optimality loss. Therefore the optimality training data set draws more heavily on regions where the stability loss is small.

Definition 4.4 (Optimality Training Distribution)**.**

$$p_{opt}(x) = \frac{\exp\left(-l_{stab}(x)\right)}{\int_\Omega \exp\left(-l_{stab}(x)\right)\, dx}\,. \tag{33}$$

Here the probability distribution reaches a maximum when the stability loss is zero, and rapidly decays as the stability loss grows more positive.

5 Experimental Setup

5.1 Introduction

In the previous sections, the concepts of Control Lyapunov Functions (CLFs) and the Universal Control Formula were thoroughly introduced. The primary takeaway is that if a suitable CLF is known, a stabilizing control policy can be readily computed using the Universal Control Formula given by (25), thus ensuring system stability. Additionally, by enforcing the optimality condition $\Lambda(x) = 0$, with $\Lambda(x)$ provided by (31), we also guarantee that the cost defined by (**??**) is minimized. Unfortunately, identifying or learning Lyapunov functions for nonlinear dynamical systems remains one of the most challenging problems in control theory [1, 3]. Classical methods often require significant analytical insight and frequently fail when addressing complex nonlinear dynamics. Recent advances leveraging neural network-based methods have opened promising avenues for data-driven Lyapunov function synthesis [13], yet simultaneously achieving rigorous stability guarantees and maintaining computational tractability continues to pose significant challenges.

Our implementation addresses these challenges through a counterexample-guided learning framework that iteratively refines a neural network representation of the Lyapunov function. The approach is particularly innovative in its treatment of different training objectives, using separate datasets for stability constraints versus optimality conditions, which allows for more targeted learning and better convergence properties.

In addition, as previously described, instead of learning a policy together with the Lyapunov function, we will implement the concept of CLF and use the Universal Control formula, which allows us to write the control policy as a function of the Lyapunov function, without any further learning needed.

The inverted pendulum serves as our test case, representing a canonical underactuated mechanical system with rich nonlinear dynamics. The system dynamics are

$$\dot{x}_1 = x_2, \qquad \text{(angular velocity)} \tag{34}$$

$$\dot{x}_2 = \sin(x_1) + u, \qquad \text{(angular acceleration with gravity and torque control)} \tag{35}$$

where $x_1 = \theta$ represents the angular position (with $\theta = 0$ being upright), $x_2 = \dot{\theta}$ is the angular velocity, and u is the applied control torque.

5.2 Control Synthesis via Sontag's Universal Formula

The implementation employs Sontag's universal formula for control synthesis, which was shown to provide a systematic way to construct stabilizing controllers for control-affine systems. Here we specialize the results presented in the previous sections to the case where the control input u is a scalar, so $G(x) = g(x) \in \mathbb{R}$. We further simplify the problem by choosing positive definite state penalty $q(x) = x^T x$, and positive definite control penalty weighting $R(x) = \rho(x) = \rho$. Then the Sontag formula (25) becomes

$$u(x) = \begin{cases} -\frac{L_f V + \sqrt{L_f V^2 + q(x)(L_g V)^2/\rho}}{L_g V} & \text{if } |L_g V| > \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{36}$$

where a small threshold value $\epsilon > 0$ prevents numerical singularities.

For the inverted pendulum, these derivatives simplify to

$$L_f V = \frac{\partial V}{\partial x_1} x_2 + \frac{\partial V}{\partial x_2} \sin(x_1), \tag{37}$$

$$L_g V = \frac{\partial V}{\partial x_2}. \tag{38}$$

5.3 Risk Function Architecture: The Dual-Dataset Approach

Before examining the loss function architecture, it is important to understand the two distinct datasets that drive our training approach. The **base dataset** (X_{base}) consists of points sampled uniformly from the training domain $[-d, d]^2$, providing comprehensive coverage of the state space with unbiased distribution. This dataset remains fixed throughout training and ensures that global properties of the Lyapunov function are learned consistently.

The **counterexample dataset** (X_{ce}) contains points where, at a given iteration, the current Lyapunov candidate violates stability conditions, specifically where $\dot{V}(x) > 0$. This dataset evolves dynamically throughout training as new violations are discovered through periodic verification cycles. Points in this dataset are often clustered around problematic regions and may include noise-augmented variants to improve local learning.

The complete training dataset at any iteration is the union $X_{\text{train}} = X_{\text{base}} \cup X_{\text{ce}}$, but our key innovation lies in applying different loss functions to different subsets of this combined data. One of the most significant innovations in this implementation is the separation of training objectives into two complementary loss functions that operate on different datasets.

Risk Loss ($\mathcal{L}_{\text{risk}}$): The risk loss consists of a weighted sum of several terms needed to guarantee that the learned Lyapunov function does indeed satisfy all of the necessary properties. The corresponding constraints are evaluated on the complete training set (base + counterexamples) and are summarized below

- Positive definiteness ($V(x) > 0,\ x \neq 0$): $\boldsymbol{\lambda_1}\mathbb{E}[\text{ReLU}(\boldsymbol{-V(x)})]$.
- Stability $\left(\dot{V} < -\sqrt{\gamma V(x) \beta(x)^T R(x)^{-1} \beta(x)} \text{ from (29)}\right)$:

  $$\boldsymbol{\lambda_2}\mathbb{E}[\text{ReLU}(\boldsymbol{\dot{V}} + \sqrt{\tfrac{\boldsymbol{\gamma}}{\boldsymbol{\rho}}\boldsymbol{V}|\boldsymbol{L_g V}|^{\mathbf{2}}})] \,.$$
- Circular level sets ($V > \frac{\|x\|}{s}$): $\boldsymbol{\lambda_3}\mathbb{E}[(\boldsymbol{V} - \|\boldsymbol{x}\|/\boldsymbol{s})^{\mathbf{2}}]$. This constraint is not necessary but it helps with regularization.
- Origin condition ($V(0) = 0$): $\boldsymbol{\lambda_4}|\boldsymbol{V(0)}|$.

Optimality Loss ($\mathcal{L}_{\text{opt}}$): The optimality loss was defined in Eqn (31) of section 2.4 as

$$\Lambda(x) = q(x) - \tfrac{1}{4}\beta(x) R(x)^{-1} \beta(x)^T + \alpha(x) \,,$$

with the property that when $\Lambda(x) = 0$, the CLF $V(x)$ coincides with the optimal CLF $V^\star(x)$ that minimizes (**??**). The corresponding optimality constraint is evaluated only on the uniform base set and consists of a single term

- Optimality constraint: $\boldsymbol{\lambda_5}\mathbb{E}[(\boldsymbol{q} - \frac{\mathbf{1}}{\mathbf{4}\boldsymbol{\rho}}\boldsymbol{L_g V^2} + \boldsymbol{L_f V})^{\mathbf{2}}]$.

This separation is crucial because the stability constraints benefit from counterexample data that reveals problematic regions while optimality constraints work best on uniformly distributed data to ensure global performance. Our approach prevents overfitting to sparse counterexample regions while maintaining stability guarantees.

6 Algorithmic Framework and Implementation

6.1 High-Level Learning Algorithm

The counterexample-guided learning process follows the structure shown below in Algorithm 1. More details about the actual Python implementation using PyTorch, can be found in the appendices.

6.2 Neural Network Architecture

The Lyapunov function is represented by a feedforward neural network with careful architectural choices. In particular

- **Input Layer**: 2 neurons (state variables x_1, x_2).
- **Hidden Layer**: 256 neurons with tanh activation.
- **Output Layer**: 1 neuron with **no activation function**.

The removal of output activation was a critical design decision. Earlier versions used $\mathbf{tanh}$ activation, which constrained $V(x) \in [-1, 1]$, potentially limiting the function's expressiveness for large state deviations. The current implementation allows $V(x)$ to take any real value, with positivity enforced through the loss function. This change significantly improves the network's ability to represent radially unbounded Lyapunov functions.

6.3 Counterexample Detection and Management

The implementation includes a well thought-out approach for managing training data throughout the learning process. The counterexample detection phase begins by sampling points uniformly across the domain and evaluating the Lyapunov derivative $\dot{V}(x) = L_f V + L_g V u$ using automatic differentiation. Any points where this derivative is positive represent violations of the stability condition and are identified as counterexamples. The system employs a forgetting mechanism that tracks the age of each point in the counterexample pool. As verification cycles progress, points that have not been refalsified accumulate age, and those exceeding a threshold are removed to prevent overfitting to resolved issues. When new counterexamples are discovered near existing points, the age of nearby points is reset, reflecting the spatial correlation of problematic regions.

To improve training robustness, the system applies noise augmentation around discovered counterexamples. This involves repeating each counterexample multiple times and adding Gaussian noise with a configurable standard deviation. This augmentation helps the neural network learn smoother decision boundaries while maintaining focus on genuinely problematic regions.

7 Visualization and Analysis Tools

7.1 Comprehensive Diagnostic Suite

The implementation provides extensive visualization capabilities that proved essential during development and debugging. The training diagnostics track loss evolution over iterations and provide detailed component-wise breakdowns showing how positive definiteness, stability, optimality, and other constraints contribute to the overall loss function.

For analyzing the learned Lyapunov function itself, the system generates contour plots across the state space with both automatic and manually specified level selection. These visualizations help verify that the learned function exhibits the expected radial structure and identify regions where the function might behave unexpectedly. The large-domain analysis capability extends evaluation well

Figure 1: Neural Lyapunov algorithm.

Algorithm 1 Dual-Dataset Neural Lyapunov Learning

```
Input: Dynamics $f$, $g$; domain $D$; hyperparameters $\Lambda$
Output: Neural Lyapunov function $V_\theta$

Initialize $V_\theta$ with Xavier weights
Generate base set $X_{\text{base}} \sim \text{Uniform}(-d, d)^2$
Initialize counterexample pool $P = \emptyset$
Set training set $X_{\text{train}} = X_{\text{base}}$

while not converged do
    for each training iteration do
        $\mathcal{L}_{\text{risk}} \leftarrow \text{lyapunov\_risk}(V_\theta, X_{\text{train}})$
        $\mathcal{L}_{\text{opt}} \leftarrow \text{optimality\_loss}(V_\theta, X_{\text{base}})$
        $\theta \leftarrow \theta - \alpha \frac{\partial}{\partial\theta}(\mathcal{L}_{\text{risk}} + \mathcal{L}_{\text{opt}})$
    end for

    if iteration mod verification_interval = 0 then
        $X_{\text{sample}} \sim \text{Uniform}(-D, D)^2$
        $C \leftarrow \{x \in X_{\text{sample}} : \dot{V}_\theta(x) > 0\}$

        Age all points in $P$
        Remove points with age $> T_{\text{forget}}$
        for each $c \in C$ do
            if $\exists p \in P : \|c - p\| < \epsilon_{\text{close}}$ then
                Reset age of $p$
            else
                Add $c$ to $P$ with age 0
            end if
        end for

        $X_{\text{train}} \leftarrow X_{\text{base}} \cup \{p.\text{point} : p \in P\}$

        if $|C| = 0$ for 3 consecutive cycles then
            break
        end if
    end if
end while

return $V_\theta$
```

beyond the training region to check global behavior and ensure the function maintains reasonable properties outside the training data.

Control analysis tools focus on potential issues with the Sontag controller implementation. Heatmaps of $|L_g V|$ reveal regions where control singularities might occur, while Lie derivative visualization clearly shows which parts of the state space satisfy the stability conditions. Control effort analysis helps ensure that the synthesized controller produces reasonable command signals across the operating domain.

The trajectory analysis capabilities generate sample trajectories with phase portraits that demonstrate closed-loop system behavior. These tools track the time evolution of $V(x(t))$ along individual trajectories, providing direct evidence of Lyapunov function decrease. Convergence analysis computes final distance metrics that quantify how close trajectories approach the origin.

Finally, the data distribution analysis visualizes how training data evolves throughout the learning process. These tools show the distribution of base versus counterexample points, track how counterexample patterns change over verification cycles, and generate density plots that reveal which regions of the state space are most problematic for the current Lyapunov candidate.

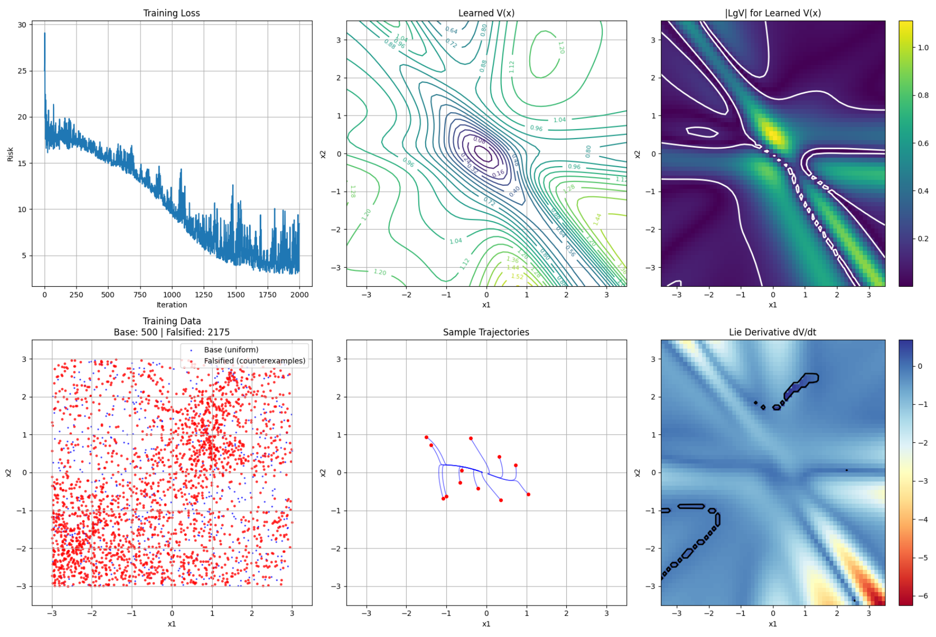

Figure 2: Visualization example in the case of stabilizing an inverted pendulum.

8 Parameter Analysis and Hyperparameter Sensitivity

8.1 Critical Hyperparameters

The system's performance depends critically on several key parameters that must be carefully tuned for each application. The loss weights control the relative importance of different Lyapunov constraints during training. We typically use a strong positive definiteness weight of $\lambda_1 = 100.0$ to ensure $V(x) \geq 0$ is well-enforced, while the stability weight of $\lambda_2 = 10.0$ provides moderate enforcement of the $\dot{V} < 0$ condition. The circular weight of $\lambda_3 = 10.0$ encourages radially symmetric level sets, which often lead to better global properties, and the origin weight of $\lambda_4 = 100.0$ strongly enforces the critical $V(0) = 0$ condition. The optimality weight of $\lambda_5 = 1$ provides enforcement of the Hamilton-Jacobi-Bellman-like constraint without overwhelming the stability objectives.

Training parameters significantly impact the balance between computational efficiency and solution quality. The verification interval of 40 iterations represents a compromise between frequent counterexample discovery and training efficiency. Too frequent verification wastes computation, while infrequent verification allows the network to drift away from stability constraints. The forgetting factor of 1000 cycles provides long memory for counterexamples, ensuring that resolved issues don't get forgotten too quickly. The epsilon close threshold of 1e-3 enables fine-grained duplicate detection, preventing redundant points from cluttering the training set.

Control parameters directly affect the Sontag formula implementation and controller behavior. The rho (ρ) parameter of 1.0 controls the penalty on control effort in the Sontag formula, with larger values encouraging smaller control signals at the expense of convergence speed. The numerical epsilon of 1e-2 provides the singularity avoidance threshold, preventing division by very small values of $|L_g V|$ that could lead to numerical instability.

9 Performance Analysis and Experimental Results

9.1 Convergence Characteristics

Typical training exhibits several phases summarized below.

1. **Initial Phase** (0-500 iterations): Rapid decrease in positive definiteness violations.
2. **Refinement Phase** (500-2000 iterations): Gradual improvement in stability constraints.
3. **Counterexample Integration** (verification cycles): Periodic spikes in loss as new problematic regions are discovered.
4. **Convergence Phase**: Smooth loss decrease with no new counterexamples.

9.2 Stability Performance

The learned controller demonstrates several desirable properties.

Convergence Region: Successfully stabilizes initial conditions within $|\theta| \leq \pi/2$ radians.

Control Effort: Bounded control signals with reasonable magnitude.

Robustness: Maintains stability under small perturbations and noise.

Trajectory Metrics

- Initial angles: $[-\pi/2, \pi/2]$ radians.
- Convergence threshold: $\|x\| < 0.05$.
- Success rate: $> 95\%$ for initial conditions in training domain.
- Average settling time: 2-4 seconds.

10 Technical Implementation Details and Numerical Considerations

10.1 Computational Complexity

Training Complexity: $O(N_{\text{batch}} \times N_{\text{hidden}} \times N_{\text{verification}})$ per iteration.

Verification Complexity: $O(N_{\text{sample}} \times N_{\text{forward}})$ per verification cycle.

Memory Usage: Dominated by gradient graph storage for second-order derivatives.

The implementation efficiently manages computational resources through

- Batch processing of training points.
- Periodic verification rather than continuous checking.
- Intelligent data structure management for counterexamples.

The training of a set of parameters θ such that $V_\theta(x)$ converges to a well-behaved Lyapunov function, utilizing two separate loss functions, each with its own specific training set, proved challenging. In particular, when we defined a single loss as the sum of the risk and optimality losses, the individual gradients could point in very different (steepest descent) directions on their respective loss surfaces. As a result, the algorithm effectively implemented a weighted average between the two gradients.

In practice, we observed that if the risk terms were weighted much higher than the optimality constraints, the risk loss would converge to zero, but the optimality loss would reach a lower bound and not decrease further. This meant that stability was guaranteed, but not optimality. Conversely, if the optimality loss was weighted much higher than the risk loss, the optimality loss would converge

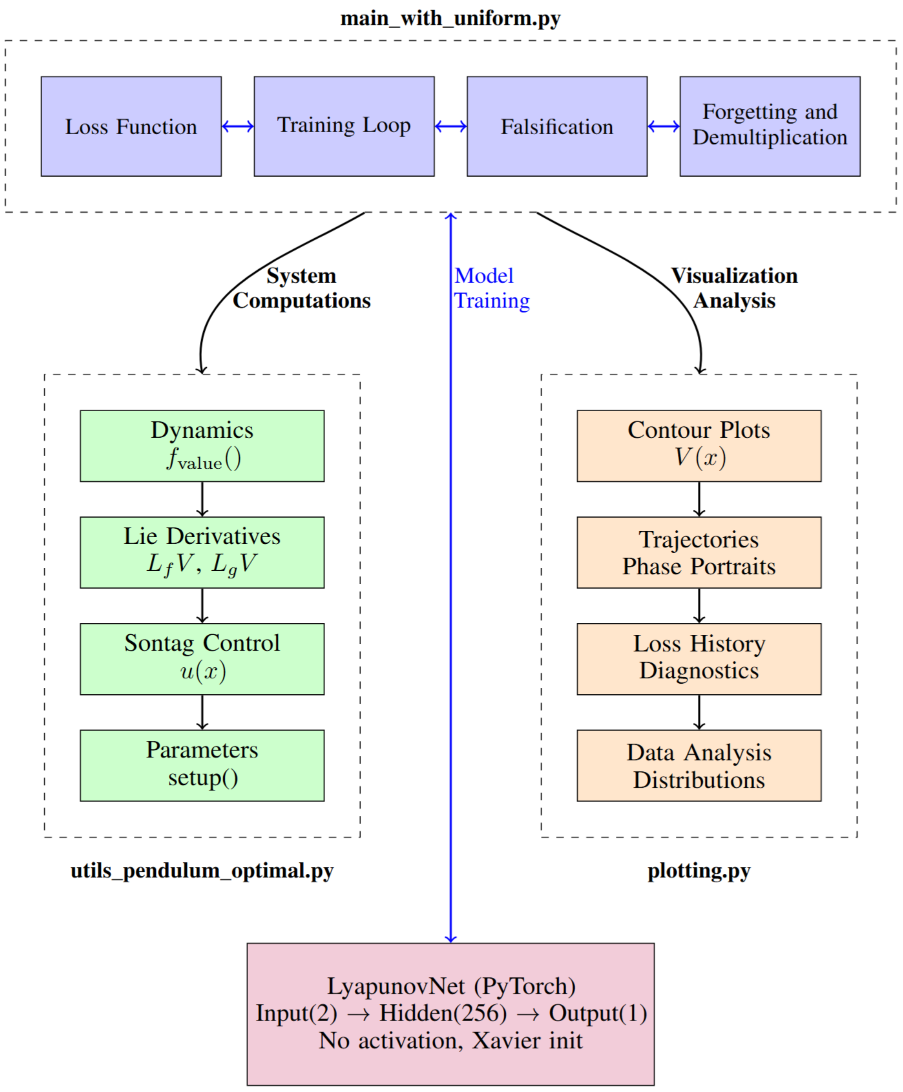
main_with_uniform.py
Loss Function
Training Loop
Falsification
Forgetting and Demultiplication
System Computations
Model Training
Visualization Analysis
Dynamics $f_{\text{value}}()$
Lie Derivatives L_fV, L_gV
Sontag Control $u(x)$
Parameters setup()
utils_pendulum_optimal.py
Contour Plots $V(x)$
Trajectories Phase Portraits
Loss History Diagnostics
Data Analysis Distributions
plotting.py
LyapunovNet (PyTorch)
Input(2) → Hidden(256) → Output(1)
No activation, Xavier init

Figure 3: Neural Lyapunov Learning System Architecture

to a very small value, but the risk loss would not. This might erroneously suggest that we had achieved optimality; however, if the risk loss does not converge to zero, the resulting $V(x)$ is not a valid Lyapunov function. Consequently, the Sontag controller cannot guarantee stability, making any discussion of optimality meaningless.

It is also worth noting that, even if the optimality loss did not converge to zero, its mere presence as a regularizer accelerated the convergence of the risk loss to zero, much faster than when the optimality loss was absent.

11 Code Architecture and Modularity

11.1 Implementation Architecture

The system architecture separates tasks across three main modules, as shown in Figure 3. The main component handles the dual-dataset training loop, manages the counterexample pool with aging mechanisms, and monitors convergence through a three-cycle termination criterion.

The system utilities module provides the mathematical foundations: dynamics implementation for the inverted pendulum, Lie derivative computation using automatic differentiation, Sontag control synthesis with numerical safeguards, and centralized parameter management.

The visualization tools generate Lyapunov function contour plots, sample trajectory phase portraits, training loss history with component analysis, and data distribution diagnostics that proved essential during development.

The neural network core uses a simple feedforward architecture mapping from 2 input states through 256 hidden neurons to 1 output. The design omits output activation to allow unbounded Lyapunov functions, uses Xavier initialization, and provides full automatic differentiation support.

12 Limitations and Future Work

12.1 Current Limitations

Several fundamental limitations affect the approach's broader applicability. Removing the tanh output activation improves expressiveness but risks severely negative values that violate positive definiteness constraints more dramatically. This trade-off between expressiveness and constraint satisfaction remains an ongoing challenge.

The verification process relies on random sampling for counterexample detection, which cannot provide formal guarantees about unsampled regions. This limitation is concerning for safety-critical applications requiring complete verification. While iterative refinement improves coverage, it cannot achieve the mathematical certainty of formal verification methods.

Stability guarantees remain limited to the training domain with no theoretical foundation for extension to larger state space regions. This domain dependence restricts practical applicability, especially for systems operating outside nominal conditions during transients or disturbances.

Computational requirements scale poorly with dimensionality due to the curse of dimensionality in sampling-based verification. High-dimensional systems would require exponentially more sample points, making the approach prohibitive for complex multi-body systems.

12.2 Implementation Issues

The requirement for `retain_graph=True` in gradient computations causes gradual memory accumulation during extended training. The system shows significant sensitivity to hyperparameter selection, particularly loss function weights, necessitating careful manual tuning for each application. The threshold-based approach for handling control singularities when $|L_g V|$ becomes small may be too conservative or permissive depending on system characteristics.

12.3 Potential Improvements

Implementing `softplus` activation would guarantee positive outputs while preserving unbounded nature. Integration with formal verification tools like Satisfiability Modulo Theories (SMT) solvers could provide rigorous counterexample detection while retaining neural flexibility. Gradient-based adversarial sampling could improve counterexample discovery efficiency over uniform random sampling. Modern regularization techniques like spectral normalization could reduce parameter sensitivity and improve robustness.

13 Comparative Analysis

13.1 Relationship to Existing Methods

The neural Lyapunov approach offers distinct trade-offs compared to established techniques. Versus Sum-of-Squares (SOS) programming, our method trades formal guarantees for practical applicability and computational scalability. SOS provides rigorous stability certificates but remains limited to polynomial systems, while our approach handles arbitrary nonlinear dynamics at the cost of some theoretical rigor.

Compared to LQR and linearization methods, the neural approach captures full nonlinearity without local approximations. Traditional LQR excels in theoretical foundations and efficiency but limits validity to small neighborhoods around operating points. Our method handles complete nonlinear dynamics directly, though with increased computational effort.

Most significantly, the approach differs from standard RL in treating stability constraints. RL typically discovers stability implicitly through reward design, often struggling with reliability guarantees. Our method incorporates formal stability constraints directly into learning objectives, ensuring stability considerations are never subordinated to performance optimization.

13.2 Hybrid Verification Potential

The framework naturally supports integration with formal verification methods. A hybrid approach could use neural learning to discover candidate Lyapunov functions, followed by formal verification attempts. When verification fails, generated counterexamples could augment neural training, creating iterative refinement toward formally verifiable solutions.

14 Future Research Directions

14.1 Immediate Extensions

Extension to higher-dimensional systems represents the most straightforward path forward, though facing inevitable curse of dimensionality challenges. Incorporating input constraints for bounded actuators requires modifying Sontag's formula through projection methods or barrier functions. The framework could accommodate model uncertainty through robust control extensions and handle multi-objective scenarios like trajectory tracking or switched systems.

14.2 Advanced Directions

Transfer learning could dramatically accelerate learning for related dynamics using pre-trained networks. Compositional verification methods could handle large-scale systems by decomposing them into smaller verifiable components. Extension to stochastic systems requires fundamental developments in probabilistic Lyapunov functions. Online learning capabilities could enable real-time adaptive control with streaming data.

Integration with physics-informed neural networks could improve sample efficiency by incorporating dynamics constraints into network architecture. Neural ODE architectures could provide more natural continuous-time representations. Transformer architectures might address long-term dependencies in trajectory data for complex transient behaviors.

15 Conclusions

This implementation makes significant contributions to neural Lyapunov function learning, advancing both theoretical understanding and practical applicability. The dual-dataset training approach elegantly separates competing optimization objectives while maintaining joint parameter updates, resolving the tension between stability constraints benefiting from counterexample data and optimality conditions requiring uniform coverage.

From an engineering perspective, the comprehensive approach includes advanced data management, numerical safeguards, and extensive diagnostics, transforming an academic concept into a robust framework suitable for real applications. The theoretical integration successfully combines neural learning flexibility with control theory rigor, maintaining formal stability guarantees while enabling arbitrary nonlinear dynamics handling.

The extensible design facilitates adaptation to new systems and integration with other methods. While limitations exist, they are clearly identified and addressable through natural extensions. The dual-dataset innovation represents a significant methodological advance that could influence future neural control synthesis work, providing both immediate utility and a solid foundation for future research.

16 Team Contributions

- **Alex Leonessa:** Equal co-responsibility in choosing the project topic and managing the research task. Primary responsibility for coding algorithms, running test cases, and validating results. Secondary responsibility for performing analyses.
- **Jordan M. Berg:** Equal co-responsibility in choosing the project topic and managing the research task. Primary responsibility for performing and verifying analyses. Secondary responsibility for coding and debugging algorithms, and interpreting test cases.

References

[1] H. K. Khalil, *Nonlinear systems*. Prentice hall Upper Saddle River, NJ, 2002, vol. 3.

[2] A. D. Ames, J. W. Grizzle, and P. Tabuada, “Control barrier function based quadratic programs with application to adaptive cruise control,” in *Proceedings of the 53rd Annual IEEE Conference on Decision and Control*, Los Angeles, CA, 2014, pp. 6271—-6278.

[3] E. D. Sontag, “A ‘universal’ construction of artstein’s theorem on nonlinear stabilization,” *Systems & Control Letters*, vol. 13, no. 2, pp. 117–123, 1989.

[4] R. A. Freeman and J. A. Primbs, “Control lyapunov functions: new ideas from an old source,” in *Proceedings of 35th IEEE Conference on Decision and Control*, Kobe, Japan, 1996, pp. 3926–3931.

[5] T. Jouini and A. Rantzer, “On cost design in applications of optimal control,” *IEEE Control Systems Letters*, vol. 6, pp. 452–457, 2022.

[6] J. A. Primbs, “Nonlinear optimal control: a receding horizon approach,” Ph.D. dissertation, California Institute of Technology, 1999.

[7] B. Lohmann and J. Bongard, “A discussion on nonlinear quadratic control and sontag’s formula,” Technical University of Munich, Technical Report on Automatic Control (TRAC-8), 2024.

[8] R. Freeman and P. Kokotovic, “Inverse optimality in robust stabilization,” *SIAM Journal on Control and Optimization*, vol. 34, no. 4, pp. 1365–1391, Jul. 1996.

[9] Y. W. Daniel Liberzon, Eduardo D. Sontag, “Universal construction of feedback laws achieving iss and integral-iss disturbance attenuation,” *Systems & Control Letters*, vol. 46, no. 2, pp. 111–127, 2002.

[10] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, “Control barrier function based quadratic programs for safety critical systems,” *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861—3876, 2017.

[11] C. Dawson, S. Gao, and C. Fan, “Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control,” *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1749–1767, 2023.

[12] S. M. Richards, F. Berkenkamp, and A. Krause, “The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems,” in *Conference on robot learning*. PMLR, 2018, pp. 466–476.

[13] Y.-C. Chang, N. Roohi, and S. Gao, “Neural lyapunov control,” in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Vancouver, Canada, 2019, pp. 3245–3254.

[14] H. Dai, B. Landry, M. Pavone, and R. Tedrake, “Counter-example guided synthesis of neural network lyapunov functions for piecewise linear systems,” in *2020 59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 1274–1281.

[15] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake, “Lyapunov-stable neural-network control,” in *Robotics: Science and Systems*, held online, 2020.

[16] L. Grüne, “Computing lyapunov functions using deep neural networks,” *Journal of Computational Dynamics*, vol. 8, no. 2, pp. 131–152, 2021.

[17] L. Grüne, M. Sperl, and D. Chatterjee, “Representation of practical nonsmooth control lyapunov functions by piecewise affine functions and neural networks,” *Systems Control Letters*, vol. 202, p. 106103, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167691125000854

A Complete Parameter Listing

Listing 1: Complete Parameter Configuration

```
learning_params = {
    'N': 500,                      # Initial sample size
    'D_in': 2,                     # Input dimension
    'H1': 256,                     # Hidden layer size
    'D_out': 1,                    # Output dimension
    'max_iters': 4000,             # Maximum training iterations
    'learning_rate': 0.005,        # Learning rate
    'verification_interval': 40,    # Verification frequency
    'max_cycles': 100,             # Maximum verification cycles
    'domain_size': 3.0,            # Training domain size
    'random_seed': 42,             # Reproducibility seed
    'forgetting_factor': 1000,     # Counterexample aging threshold
    'epsilon_close': 1e-3          # Duplicate detection threshold }

risk_params = {
    'positive_def_weight': 100.0,   # V(x) >= 0 weight
    'stability_weight': 1.0,        # \dot{V} < 0 weight
    'stability_threshold': 0,       # \dot{V} threshold
    'Optimality_Lambda': 0.1,       # Optimality weight
    'stability_gamma': 0.01,        # Exponential stability rate
    'circular_weight': 10.0,        # Level set regularization
    'circular_scaling': 2.0,        # Level set scaling
    'origin_weight': 100.0          # V(0) = 0 weight }

counterexample_params = {
    'n_trajectories': 30,             # Number of test trajectories per verification cycle
    'max_time': 8.0,                  # Maximum simulation time per trajectory
    'dt': 0.02,                       # Integration time step
    'n_counterexample_points': 500,   # Number of uniform random point to check for
        singularities
    'lgv_threshold': 0.01,            # Threshold for |LgV| < threshold detection
    'n_points_per_ce': 0,             # Additional points added per counterexample
    'noise_std': 0.05,                # Standard deviation for noise around
        counterexamples
    'origin_threshold': 0.05,         # Distance to origin to stop trajectory
    'domain_exit_factor': 1.0         # Factor * domain_size for trajectory termination }

visualization_params = {
    'plot_resolution': 60,            # Grid resolution for contour plots
    'plot_domain': 3.5,               # Domain size for visualization plots
    'n_sample_trajectories': 12,      # Number of sample trajectories to plot
    'sample_domain': 2.5,             # Domain for sample trajectory initial conditions
    'contour_levels': 20,             # Number of contour levels
    'figure_size': (18, 12)           # Figure size for plots }

cbf_params = {
    'rho': 1.0,                 # R = rho*Id in general Sontag Formula
    'numerical_epsilon': 1e-2  # Small value to avoid division by zero }
```

B Detailed Implementation Analysis

B.1 Automatic Differentiation and Gradient Computation

The implementation leverages PyTorch's automatic differentiation extensively. This is an example of one of the function that we wrote for this purpose.

Listing 2: Gradient Computation with Graph Retention

```
def compute_gradient(model, x, create_graph=False):
    V = model(x)
    grad_V = torch.autograd.grad(
        outputs=V.sum(),
        inputs=x,
        create_graph=create_graph,  # Enables higher-order derivatives
        retain_graph=True)[0]       # Allows multiple gradient calls
    return grad_V
```

The `create_graph=True` parameter enables computation of second-order derivatives necessary for the stability constraints, while `retain_graph=True` allows multiple gradient computations in a single forward pass.

B.2 Training Step Architecture

The dual-loss training step is implemented as shown below.

Listing 3: Dual-Dataset Training Step

```python
def train_step(model, optimizer, train_x, base_set, risk_params, cbf_params):

    optimizer.zero_grad()

    # Risk loss on complete training set (base + counterexamples)
    risk_loss, pos_loss, stab_loss, circ_loss, orig_loss =
        lyapunov_risk_only(model, train_x, risk_params, cbf_params)

    # Optimality loss only on uniform base set
    opt_loss = optimality_loss_only(model, base_set, risk_params, cbf_params)

    # Joint optimization
    total_loss = risk_loss + opt_loss
    total_loss.backward()
    optimizer.step()

    return total_loss, risk_loss, opt_loss, pos_loss, stab_loss, circ_loss, orig_loss
```

This architecture allows simultaneous optimization of both objectives while maintaining their distinct data requirements.

B.3 Lie Derivative Computation

The Lie derivative computation is central to stability verification

Listing 4: Lie Derivative Computation

```python
def compute_lie_derivative(model, x, cbf_params):
    # Compute \grad[V(x)]
    grad_V = compute_gradient(model, x, create_graph=True)

    # Apply Sontag control
    u = sontag_control(x, model, cbf_params)

    # Compute closed-loop dynamics f(x,u)
    f = f_value(x, u)

    # Lie derivative = \grad[V(x)] . f(x,u)
    lie_derivative = torch.sum(grad_V * f, dim=1, keepdim=True)

    return lie_derivative
```

B.4 Sontag Control Implementation

The implementation of the Sontag controller implement masks to account for the possibility of using batches of data.

Listing 5: Correct Sontag Control Implementation

```python
def sontag_control(x, model, cbf_params):
    epsilon = cbf_params['numerical_epsilon']
    rho = cbf_params['rho']

    # Compute Lie derivatives
    LfV = compute_actual_lfv(model, x, create_graph=True)
    LgV = compute_actual_lgv(model, x, create_graph=True)

    # Define q(x) = x^T x for penalty term
    q = (x ** 2).sum(dim=1)

    # Apply full Sontag's formula with singularity handling
    u = torch.zeros_like(LfV)
    mask = LgV.abs() > epsilon

    if mask.any():
        # Full Sontag formula
        LfV_masked = LfV[mask]
        LgV_masked = LgV[mask]
        q_masked = q[mask]

        # Compute discriminant for square root
        discriminant = LfV_masked**2 + q_masked * LgV_masked**2 / rho
        sqrt_term = torch.sqrt(torch.clamp(discriminant, min=1e-8))

        # Complete Sontag formula
        u[mask] = -(LfV_masked + sqrt_term) / LgV_masked

    return u.unsqueeze(1)
```

B.5 Verification and Convergence Criteria

The verification process implements a stopping criterion.

Listing 6: Convergence Checking

```
# Success criteria: 3 consecutive verification cycles without counterexamples
if len(counterexamples) == 0:
    consecutive_no_counterexamples += 1
    if consecutive_no_counterexamples >= 3:
        print("Training complete - three consecutive successful verifications!")
        break
else:
    consecutive_no_counterexamples = 0
```

This approach balances thoroughness with computational efficiency, ensuring sufficient confidence in the learned function while avoiding infinite verification loops.