Unsupervised Neural Network Methods for Solving Differential Equations

A DISSERTATION PRESENTED
BY
DYLAN L. RANDLE
TO
THE DEPARTMENT OF APPLIED COMPUTATION

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE
IN THE SUBJECT OF
DATA SCIENCE

HARVARD UNIVERSITY
CAMBRIDGE, MASSACHUSETTS
MAY 2020

Thesis advisor: Dr. Pavlos Protopapas Dylan L. Randle

Unsupervised Neural Network Methods for Solving Differential Equations

Abstract

Solving differential equations with neural networks in an unsupervised manner has become an exciting area of research. Learning closed-form, differentiable functions that solve differential equations has vast applications, and leveraging neural networks in particular may provide a range of benefits over traditional numerical methods, from high-dimensional performance to computational efficiency.

This thesis develops a new method for solving differential equations with unsupervised neural networks that applies Generative Adversarial Networks (GANs) to *learn the loss function* for optimizing the neural network. We present empirical results on a variety of problems showing that our method, which we call Differential Equation GAN (DEQGAN), can obtain multiple orders of magnitude lower mean squared errors than the classical unsupervised neural network method based on (squared) L_2, L_1, and Huber loss functions. Additionally, we present a simple perturbation-based sampling approach that reduces model overfitting and increases solution quality. Finally, we provide a discussion of training stability, hyperparameter tuning, and prior formulations of DEQGAN to provide insights into our approach.*

*All code will be made available at https://github.com/dylanrandle/denn.

Contents

Listing of figures

To the pursuit of knowledge.

Acknowledgments

The author would like to thank Dr. Pavlos Protopapas, Dr. David Sondak, Dr. Marios Mattheakis, and Dr. Cengiz Pehlevan for their gracious guidance, help, and support throughout the development and writing of this thesis. The author would like to thank Harvard FAS Research Computing for providing computational resources that made this work possible. Finally, the author would like to thank family and friends for their unconditional love and unwavering support.

0

Introduction

Differential equations involve the derivatives of a function, or a set of functions, and thereby define relationships between quantities and their rates of change. This initially simple and elegant idea, however, belies a vast utility and rich mathematical complexity.

Solutions to differential equations are of significant interest to a broad range of disciplines. In fields such as physics, chemistry, biology, engineering, and economics, differential equations are

applied to the modeling of important and complex phenomena.

Solving differential equations has been the topic of much work for centuries and, coupled with powerful computers and sophisticated numerical algorithms, they have been applied to the study of challenging problems such as fluid flow[4].

Concurrently, recent advances in so-called "deep learning"* have led to major successes in areas of computer vision [18,15], natural language processing [37,2,38], and control tasks such as video games, robotics, and even the ancient game of Go [31,5,7,33]. This has led researchers to apply neural networks to problems outside the domain of traditional computer science and machine learning, such as in radiology [23].

Inspired jointly by these successes, and the provable distinction that neural networks are universal function approximators [13], this work leverages neural networks to solve differential equations in a fully unsupervised manner.

We are motivated to pursue this research through a variety of factors. First, by removing a reliance on finely crafted grids which suffer from the "curse of dimensionality" in high dimensions, neural networks may be more effective than traditional methods in high-dimensional settings [34,32,9]. Furthermore, learning a closed-form, differentiable function which solves differential equations can in principle allow us to tackle inverse problems by differentiating backwards from solutions to initial conditions. By the same token, directly learning a functional solution provides a principled, and potentially more accurate, interpolation scheme [20]. Moreover, forward passes of neural networks are embarrassingly data-parallel, even in temporal dimensions, and can readily leverage parallel computing architectures for computational speedups. Finally, Mattheakis et al. [28] show that neural network solutions may obey certain physical constraints, such as conservation of energy, more accurately.

The initial idea for solving differential equations with neural networks was proposed by Lagaris

*Deep learning is a term coined to describe a subset of machine learning methods which train many-layer ("deep") neural networks.

et al. [20]. They showed that a neural network method could outperform the finite element method in terms of interpolation accuracy while still maintaining equal solution accuracy on a fixed mesh. Others have further developed this idea, providing evidence that neural networks can outperform traditional methods in a variety of ways.

Lagaris et al. expanded their work to consider arbitrarily-shaped domains in higher dimensions [22], and applied neural networks to quantum mechanics [21]. Recent work by Sirignano & Spiliopoulos [35] has used neural networks in place of basis functions to solve high-dimensional partial differential equations. To reduce the need to re-learn known physical laws, Mattheakis et al. [27] embedded physical symmetries into the structure of neural networks to improve training convergence and solution accuracy. Kumar & Yadav [19] presents a survey of neural network and radial basis function methods for solving differential equations.

Interest in research on solving differential equations with neural networks is growing. This thesis develops techniques for this line of research, focusing on two aspects of the problem. First, we show that different strategies for sampling training points can have a considerable impact on solution quality, and that by perturbing grid points with Gaussian noise we can reduce overfitting while improving accuracy. Second, we present a new method which trains Generative Adversarial Networks (GANs) [6] to solve differential equations in a fully unsupervised manner. We argue that *"learning" the loss function* in this way is worthwhile based on the lack of theoretical justification supporting classical loss functions and empirical results which we present in Chapter 2.

1

Unsupervised Neural Networks for Differential Equations

Artificial neural networks are powerful parametric models loosely based on the human brain. They are hierarchical in nature, composed of layers of "neurons" (or units) which transform their inputs through (often nonlinear) activation functions. An artificial neural network can be

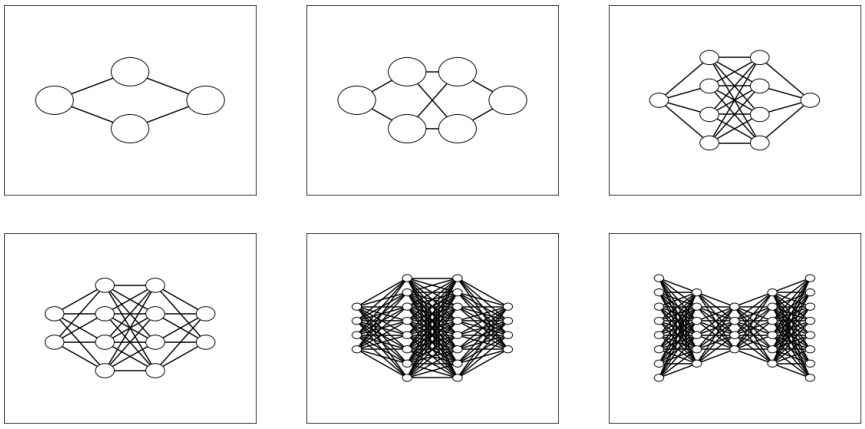


Figure 1.1: Examples of fully-connected neural network architectures. Neural networks can be composed of arbitrary numbers of layers and neurons per layer. The first (left-most) layer is called the input layer while the last (right-most) layer is called the output layer. The layers in between the input and output layers are called "hidden" layers.

represented as a sequence of affine transformations followed by activation functions

$$y = f_{\text{layer}_n}\left(f_{\text{layer}_{n-1}}\left(\ldots\left(f_{\text{layer}_1}(x)\right)\ldots\right)\right) \tag{1.1}$$

where

$$f_{\text{layer}_i}(x) = \sigma\left(W_i^T x + b_i\right) \forall i \in [1, \ldots n] \tag{1.2}$$

with $\sigma(\cdot) = \tanh(\cdot)$, for example. Figure 1.1 plots a variety of possible fully-connected* neural network architectures.

Neural networks have been shown, in the limit of infinitely many neurons, to be capable of approximating any reasonable function [13]. This property has led researchers to doggedly pursue neural networks for many years, developing efficient techniques, most notably "backpropagation" [11], for

*Fully-connected networks are a type of neural network architecture in which each neuron in layer i is connected to all neurons in the next layer $i+1$.

training these often high-dimensional models.

Early work by Lagaris et al. [20] proposed solving differential equations in an unsupervised manner with neural networks. The paper considers general differential equations of the form

$$F(t, \Psi(t), \Delta\Psi(t), \Delta^2\Psi(t)) = 0 \tag{1.3}$$

where $\Psi(t)$ is the solution of the equation, Δ and Δ^2 represent the first and second derivatives, and the system is subject to certain boundary or initial conditions. The learning problem is then formulated as minimizing the sum of squared errors of the above equation

$$\min_{\theta} \sum_{t \in D} F(t, \Psi_\theta(t), \Delta\Psi_\theta(t), \Delta^2\Psi_\theta(t))^2 \tag{1.4}$$

where Ψ_θ is a neural network parameterized by θ. This allows one to use backpropagation to train the parameters of the neural network to satisfy the differential equation, thus training the model to solve the equation.

For example, consider the motion $x(t)$ of an oscillating body (e.g. a mass on a frictionless spring) given by the simple harmonic oscillator differential equation

$$\ddot{x}(t) + x(t) = 0 \tag{1.5}$$

with initial conditions $x_0 = 0$ and $\dot{x}_0 = 1$. This equation has an exact analytical solution of the form $x(t) = sin(t)$.

We can solve this equation without access to training data (i.e. ground truth solutions) using a neural network which minimizes the loss function

$$\min_{\theta} \sum_{t \in D} \left(\hat{\ddot{x}}(t) + \hat{x}(t) \right)^2 \tag{1.6}$$

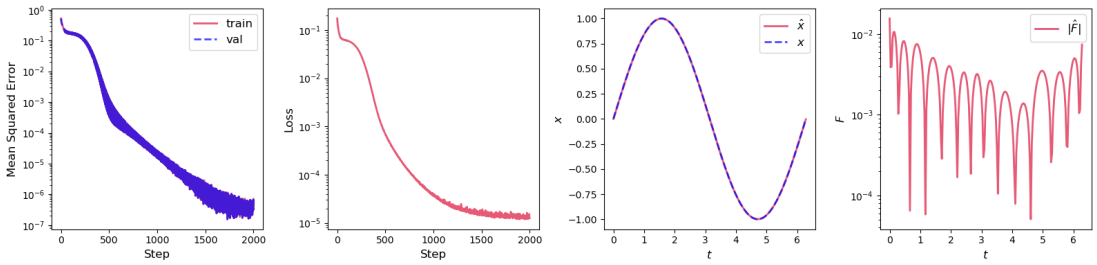


Figure 1.2: Visualization of a fully-connected neural network trained to solve the simple oscillator differential equation. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the unsupervised loss for each step. Right of this we plot the prediction of the model $\hat{x}$ and the true analytic solution x as functions of time t. The right-most figure plots the absolute value of the residual of the predicted solution $\hat{F}$.

where $\Psi_\theta(t) = \hat{x}(t)$ is the output of a neural network parameterized by θ, D specifies the time domain of the problem, and the minimization is over the parameters of the neural network. We use backpropagation to iteratively optimize the objective, and automatic differentiation to compute the derivatives $\hat{\ddot{x}} = \frac{\partial^2 \hat{x}}{\partial t^2}$.

A two hidden layer network composed of 30 units per layer can solve this problem to a high degree of accuracy. Figure 1.2 presents the result of training the model without any supervision in the form of solution data. The neural network learns to solve the equation simply by minimizing the residuals as shown in Equation 1.6.

In Figure 1.2, we observe that the mean squared error of the predicted solution, computed against the known ground truth, decreases steadily over the course of training and converges to $\sim 10^{-6}$. We note that the unsupervised training loss (which is the mean squared error of the residuals) is similarly decreasing and converges to $\sim 10^{-5}$. We see that the predicted solution $\hat{x}$ is indistinguishable from the ground truth solution x, and that the absolute value of the residuals of the candidate solution $\hat{F}$ are small across the domain. This example nicely illustrates the classical method for training unsupervised neural networks to solve differential equations.[†]

[†]For more information, see Lagaris et al. [20] and Mattheakis et al. [27].

1.1 Adjusting for Constraints

Mattheakis et al. [27] show that it is possible to embed physical constraints into neural networks to directly optimize for solutions that are physically realizable. They demonstrate that their method improves convergence of neural networks trained to solve differential equations. Throughout this work, we employ the adjustments proposed by their paper to the candidate solutions as a way to satisfy initial and boundary conditions for differential equations.

For example, consider adjusting the neural network solution $N(t)$ to satisfy the initial condition $N(t_0) = x_0$. This is achieved by applying the transformation

$$\tilde{N}(t) = \Phi\left(N(t)\right) = x_0 + \left(1 - e^{-(t-t_0)}\right) N(t) \tag{1.7}$$

Intuitively, Equation 1.7 adjusts the output of the neural network $N(t)$ to be exactly x_0 when $t = t_0$, and decays this constraint exponentially in t. We apply this transformation to each prediction from the neural network as a way of exactly satisfying initial and boundary conditions.

1.2 Sampling Strategies

The traditional approach for training a neural network to solve a differential equation considers a fixed mesh of points over which to minimize the unsupervised loss (the residuals of the equation). When moving beyond simple differential equations, however, it is helpful to think of ways to improve training convergence. One technique we have developed leverages the fact that, with the unsupervised neural network method for solving differential equations, we are not constrained to using a fixed grid of points and are thus free to sample them.

Our motivation for performing sampling stems from the fact that non-convex optimization procedures often benefit from introducing stochasticity (e.g. in *stochastic* gradient descent), and the

observation that sampling is a mechanism through which one can induce this useful randomness. Moreover, our empirical results, presented below, show that the choice of sampling procedure has a significant impact on training convergence and solution quality.

The sampling approach which we leverage throughout the remainder of this work is to individually "perturb" the points of an initial fixed mesh with Gaussian noise. For each point in the mesh, we add a small term ε sampled i.i.d. from the Normal distribution

$$\varepsilon \sim \mathcal{N}\left(\mu = 0, \sigma = \frac{\Delta t}{\tau}\right) \tag{1.8}$$

where Δt is the inter-point spacing of the original mesh, and τ is a hyperparameter that controls the probability of sampling points in overlapping regions. This has the effect of randomly jittering the training points, thereby reducing overfitting to the fixed mesh, while simultaneously enabling the controlling of sample variance through the hyperparmater τ.

Figure 1.3 visualizes the effect of changing τ on the sampling distribution of grid points. We see that as we increase τ the sampling distribution of the grid points becomes concentrated at their original fixed centers. Conversely, as we reduce τ, the sampling distribution becomes more spread across the grid and total sample variance increases. In our experiments, we set $\tau = 3$ as we find this adequately eliminates overfitting while significantly reducing the variability of the training loss, leading to improved solution quality.

Reynolds-Averaged Navier Stokes Consider the Reynolds-Averaged Navier Stokes equation for the average velocity profile u of an incompressible fluid at position y in a one-dimensional channel given by

$$\nu \frac{d^2u}{dy^2} - \frac{d}{dy}\left((\kappa y)^2 \left|\frac{du}{dy}\right| \frac{du}{dy}\right) - \frac{1}{\rho}\frac{dp}{dx} = 0 \tag{1.9}$$

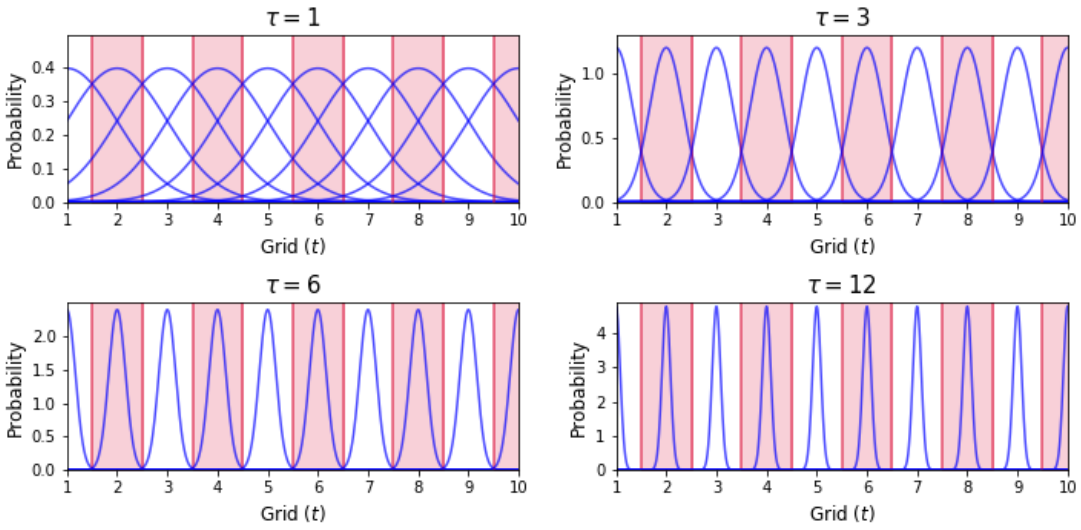
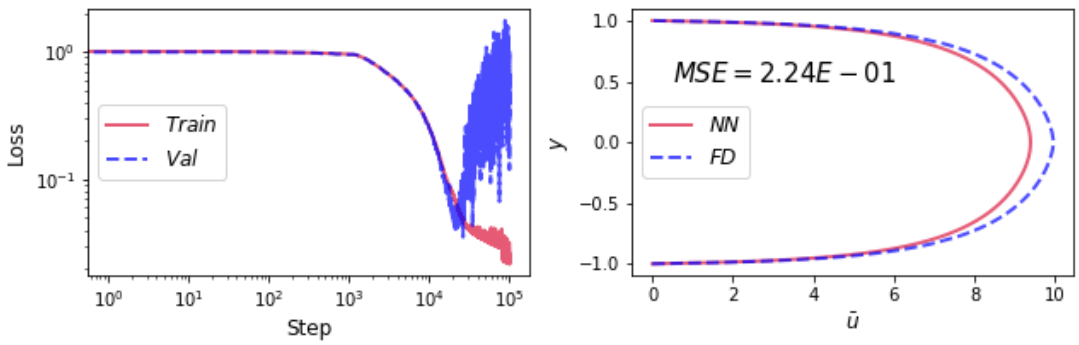


Figure 1.3: Effect of τ on the sampling distribution of grid points for the "perturbed" sampling method. Each figure plots the probability distribution of ε centered at each point on a grid of integers from 1 to 10, for different values of τ.

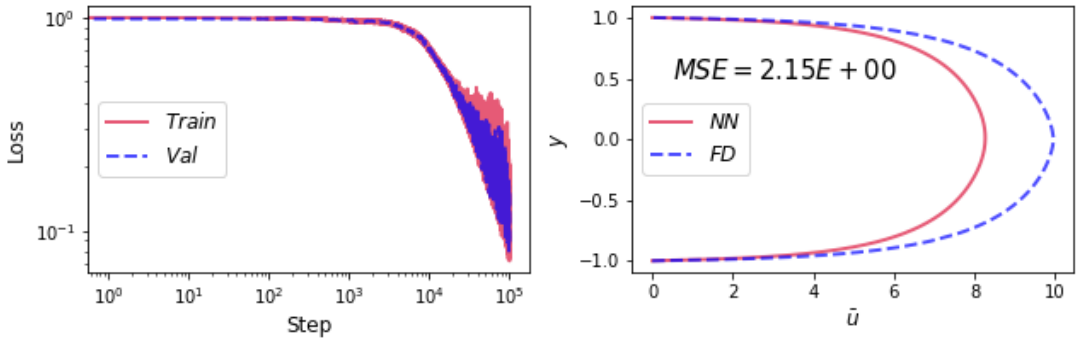
where $\nu = 0.0055, \kappa = 0.41, \rho = 1$ are given constants and $\frac{dp}{dx} = -1$ is a given pressure gradient.

Figure 1.4 shows the results of training a three layer neural network with 30 units per layer to solve this problem. We compare three strategies for choosing points: a fixed mesh where the points are always the same, uniform random sampling where every point is sampled i.i.d. from a uniform distribution with support over the domain of the problem, and our perturbation method described above which jitters the points from the grid according to a small Gaussian noise. We plot the training and validation losses as well as the predicted and ground truth solutions, providing the final solution accuracy alongside the prediction.

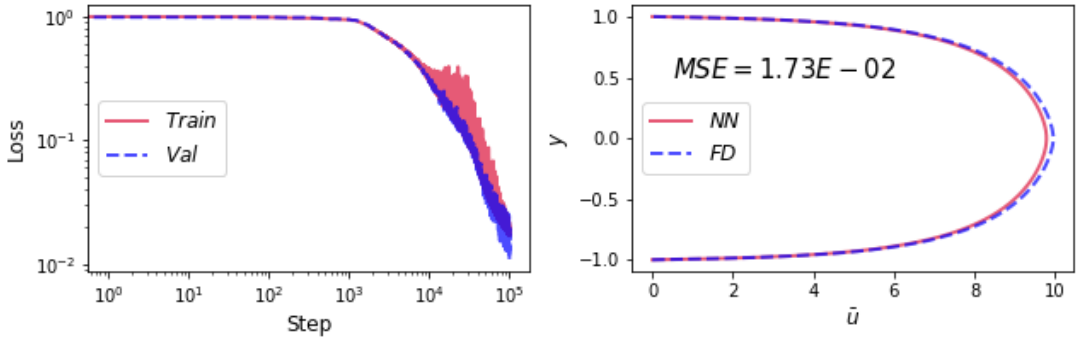
With a fixed grid we see that the network clearly suffers from overfitting as the validation loss diverges after step $\sim 10^4$. With uniform sampling the problem of overfitting is reduced, but the variability of the training and validation losses is greatly increased and the solution mean squared error is higher than with a fixed grid. Using our perturbation-based approach with $\tau = 3$, we simultaneously eliminate overfitting while significantly reducing the variability of the loss and increasing



(a) Fixed mesh.



(b) Uniform sampling.



(c) Perturbed sampling.

Figure 1.4: Comparison of three sampling strategies: fixed, uniform, and perturbed. The left panel plots the (unsupervised) training ($Train$) and validation (Val) losses, while the right panel shows the predicted neural network solution (NN) compared to the ground truth solution obtained via a finite differences method (FD). We find that our perturbed sampling yields the lowest mean squared error (MSE) while controlling for overfitting.

solution accuracy over either a fixed grid or uniform sampling. We observe this effect across a variety of experiments, and apply this perturbed sampling method for all problems considered throughout the remainder of this work.

2

Generative Adversarial Networks: Learning The Loss Function

We present a new method for solving differential equations with unsupervised neural networks that is based on Generative Adversarial Networks (GANs)[6]. Given the plethora of potential loss functions for the task of solving differential equations with neural networks, and the lack–to the

best of our knowledge–of any theoretical justification for a particular choice, we propose *learning the loss function* with GANs. We show that this method, which we call Differential Equation GAN (DEQGAN), can obtain multiple orders of magnitude lower mean squared errors than the classical unsupervised neural network method based on L_1, L_2, and Huber loss functions.

2.1 Introduction

In the classic setting of data following a Gaussian noise model

$$y = x + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2) \tag{2.1}$$

there is clear theoretical justification, based on the maximum likelihood principle, for fitting models with the squared error (L_2 norm) loss function. In the case of deterministic differential equations, however, there is no noise model and we lack formal justification for a particular choice of loss function among the multitude of options.

To circumvent this problem, we propose GANs for solving differential equations in a fully unsupervised manner. We think of the discriminator model of the GAN as learning an appropriate loss function for a given equation. Moreover, GANs have been shown to excel in scenarios where classic loss functions, such as the mean squared error, struggle due to their inability to capture complex spatio-temporal dependencies[24,25,16]. We present empirical results demonstrating that our method can dramatically outperform the results obtained by classical methods for solving differential equations with unsupervised neural networks.

Our main contribution is a method for formulating the task of solving differential equations as a GAN training problem. DEQGAN works by separating the differential equation into left-hand side (*LHS*) and right-hand side (*RHS*), then training the generator to produce a *LHS* that is indistinguishable to the discriminator from the *RHS*. Experimental results show that our method produces

solutions which obtain multiple orders of magnitude lower mean squared errors (computed from known analytic or numerical solutions) than comparable classical unsupervised methods which use (squared) L_2, L_1, and Huber loss functions.

2.2 Related Work

Goodfellow et al. [6] introduced the idea of learning generative models with neural networks and an adversarial training algorithm, called Generative Adversarial Networks (GANs). Since their seminal paper, a plethora of authors have further developed this idea. Mirza & Osindero [29] proposed Conditional GANs which introduce auxiliary conditioning information (e.g. class labels) to enable generative models of conditional distributions. Arjovsky et al. [1] introduced WGAN, a formulation of GANs based on the Wasserstein distance loss function, and showed that this improves training stability and output quality. Gulrajani et al. [8] introduced WGAN-GP, an extension to WGAN which approximately enforces a Lipschitz constraint on the discriminator (or "critic") through a gradient penalty instead of ad-hoc weight clipping. As an alternative to WGAN, Miyato et al. [30] proposed a spectral normalization technique to enforce the Lipschitz constraint that outperforms WGAN on some problems.

Our work distinguishes itself from other GAN-based approaches for solving differential equations by removing the dependence on using supervised training data (i.e. solutions of the equation). Others have applied GANs to differential equations, but invariably use some solution data gathered by simulation or experiment. Yang et al. [41] apply GANs to stochastic differential equations but include "snapshots" of ground-truth data for training. A project by students at Stanford [36] employed GANs to perform "turbulence enrichment" in a manner akin to that of super-resolution for images proposed by Ledig et al. [25]. However, their method uses solution data when training the GAN. Our method is the first to our knowledge to apply GANs to differential equations in fully unsupervised

settings.

2.3 Background

2.3.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs)[6] are a type generative model that use two neural networks to induce a generative distribution $p(x)$ of the data by formulating the inference problem as a two-player, zero-sum game.

The generative model first samples a latent variable $z \sim \mathcal{N}(0, 1)$, which is used as input into the generator G (e.g. a neural network). A discriminator D is trained to classify whether its input was sampled from the generator (i.e. "fake") or from a reference data set (i.e. "real"). In practice, GANs have performed exceptionally well at generating realistic samples from complex, high-dimensional data[25,16,42].

Informally, the process of training GANs proceeds by optimizing a minimax objective over the generator and discriminator such that the generator attempts to trick the discriminator to classify "fake" samples as "real". Formally, one optimizes

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[1 - \log D(G(z))] \tag{2.2}$$

where $x \sim p_{\text{data}}(x)$ denotes samples from the empirical data distribution and $p_z \sim \mathcal{N}(0, 1)$ samples in latent space. In practice, the optimization alternates between gradient ascent and descent steps for D and G respectively.

2.3.2 Conditional GAN

Conditional GANs[29] are a simple extension to the above formulation, whereby the discriminator and/or generator are conditioned on an extra variable y. This could be any kind of auxiliary information, such as class labels or data from other modalities. Formally, a conditional GAN optimizes

$$\min_G \max_D V(D, G) = \mathbb{E}_{x\sim\, p_{\text{data}}(x)} \left[\log D(x|y)\right] + \mathbb{E}_{z\sim p_z(z)} \left[1 - \log D(G(z|y))\right]. \tag{2.3}$$

For this paper, we found that conditioning the discriminator on points in the domain of the system (e.g. the time variable t) can lead to improved training stability in some cases.

2.3.3 Two Time-Scale Update Rule

Heusel et al. [12] proposed the two time-scale update rule (TTUR) for training GANs, a method in which the discriminator and generator are trained with separate learning rates. They showed that their method led to improved performance and proved that, in some cases, TTUR ensures convergence to a stable local Nash equilibrium. One intuition for TTUR comes from the potentially different loss surface curvatures of the discriminator and generator. Allowing learning rates to be tuned to a particular loss surface can enable more efficient gradient-based optimization. In practice make use of TTUR throughout this paper as an instrumental lever when tuning GANs to reach desired performance.

2.3.4 Wasserstein Loss with Gradient Penalty

Motivated by the tendency of discriminator gradients to explode when optimizing Equation 2.2, Arjovsky et al. [1] propose the Wasserstein GAN (WGAN), a formulation of GANs in which the Wasserstein-1 (or "Earth Mover") distance is used as the loss function instead of cross-entropy. The

WGAN value function is constructed using the Kantorovich-Rubinstein duality [39] to obtain

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim p_{\text{real}}(x)} [D(x)] - \mathbb{E}_{z \sim p(z)} [D(G(z))] \tag{2.4}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions. The Lipschitz constraint is imposed to satisfy a strong form of continuity, known as Lipschitz continuity, which bounds the derivatives of the discriminator and prevents gradient explosion. To enforce the Lipschitz constraint, Gulrajani et al. [8] propose a gradient penalty (WGAN-GP) which is added to the loss function of the discriminator. This can be interpreted as a soft constraint, since the term added to the loss does not strictly enforce that the discriminator be 1-Lipschitz. The updated discriminator loss becomes

$$L_D = \mathbb{E}_{z \sim p(z)} [D(G(z))] - \mathbb{E}_{x \sim p_{\text{real}}(x)} [D(x)] + \lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} \left[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2 \right] \tag{2.5}$$

where $\hat{x} \sim \mathbb{P}_{\hat{x}}$ is the distribution of points uniformly sampled along straight lines between the data distribution $x \sim p_{\text{real}}(x)$ and the generator distribution $G(z)$ with $z \sim p(z)$, and λ is a hyperparameter controlling the relative influence of the gradient penalty on the loss function.

2.3.5 Spectral Normalization

Proposed by Miyato et al. [30], Spectrally Normalized GAN (SN-GAN) is an alternative method for controlling exploding discriminator gradients when optimizing Equation 2.2 that leverages a novel weight normalization technique.

The key idea is to control the Lipschitz constant of the discriminator by constraining the spectral norm of each layer in the discriminator. Specifically, the authors propose dividing the weight matrices W_i of each layer i by their spectral norm $\sigma(W_i)$

$$W_{SN,i} = \frac{W_i}{\sigma(W_i)} \tag{2.6}$$

where

$$\sigma(W_i) = \max_{\|h_i\|_2 \leq 1} \|W_i h_i\|_2 \tag{2.7}$$

and h_i denotes the input to layer i. The authors prove that this normalization technique bounds the Lipschitz constant of the discriminator above by 1, thus strictly enforcing the 1-Lipshcitz constraint on the discriminator. In our experiments, adopting the SN-GAN formulation leads to even better performance than WGAN-GP.

2.3.6 Residual Connections

He et al. [10] showed that adding residual connections improved training of deep neural networks. We employ residual connections to our deep networks as they allow gradients to more easily flow through the models and thereby reduce numerical instability. Residual connections augment a typical activation with the identity operation

$$y = \mathcal{F}(x, W_i) + x \tag{2.8}$$

where $\mathcal{F}$ is the activation function, x is the input to the unit, W_i are the weights, and y is the output of the unit. This acts as a "skip connection", allowing inputs and gradients to forego the nonlinear component.

2.4 Differential Equation GAN

Here we present our method, Differential Equation GAN (DEQGAN), which trains a GAN in a fully unsupervised manner (without access to ground-truth solutions during training) to solve differential equations. To do this, we rearrange the differential equation such that the left-hand side *LHS* contains all of the terms which depend on the generator (e.g. $\hat{\Psi}$, $\Delta\hat{\Psi}$, $\Delta^2\hat{\Psi}$, etc.), and the right-hand side *RHS* contains only constants (e.g. zero).

Then we sample points from the domain $t \sim \mathcal{D}$ and use them as input to a generator $G(t)$, which produces candidate solutions $\hat{\Psi}$. We adjust $\hat{\Psi}$ for initial or boundary conditions according to the analytic adjustment function Φ presented in Section 1.1. Then we construct the *LHS* from the differential equation F using automatic differentiation

$$LHS = F\left(t, \hat{\Psi}(t), \Delta\hat{\Psi}(t), \Delta^2\hat{\Psi}(t)\right) \tag{2.9}$$

and set *RHS* to its appropriate value (in our examples, $RHS = 0$).

From here, training proceeds in a manner similar to traditional GANs. We update the weights of the generator G and discriminator D according to the gradients

$$\eta_G = \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(LHS^{(i)}\right)\right) \tag{2.10}$$

$$\eta_D = \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(RHS^{(i)}\right) + \log\left(1 - D\left(LHS^{(i)}\right)\right)\right] \tag{2.11}$$

where $LHS^{(i)}$ is the output of $G\left(t^{(i)}\right)$ after adjusting for initial or boundary conditions and constructing the *LHS* from F. Note that we perform stochastic gradient *descent* for G (gradient steps $\propto -\eta_G$) and stochastic gradient *ascent* for D (gradient steps $\propto \eta_D$). We provide a schematic repre-

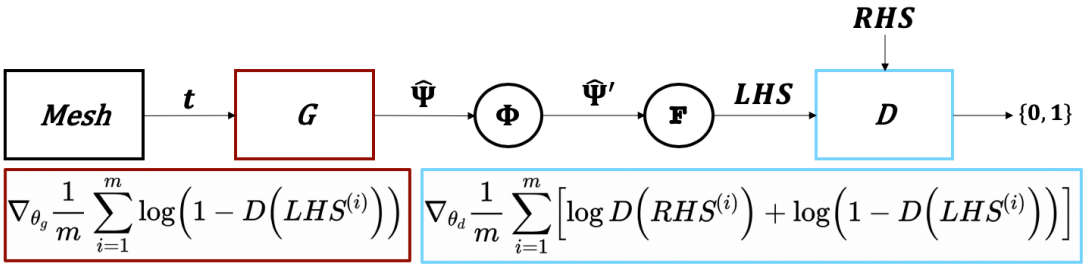


Figure 2.1: Schematic representation of DEQGAN. We perturb points t from the mesh and input them to a generator G, which produces candidate solutions $\hat{\Psi}$. Then we analytically adjust these solutions according to Φ and apply automatic differentiation to construct LHS from the differential equation F. RHS and LHS are passed to a discriminator D, which is trained to classify them as "real" and "fake" respectively.

sentation of DEQGAN in Figure 2.1 and detail the training steps in Algorithm 1.

Informally, our algorithm trains a GAN by setting the "fake" component to be the LHS (in our formulation, the residuals of the equation), and the "real" component to be the RHS of the equation. This results in a GAN that learns to produce solutions that make LHS indistinguishable from RHS, thereby approximately solving the differential equation.

A subtle yet important note is that training can be unstable if LHS and RHS are not chosen properly. Specifically, we find that training fails if RHS is a function of the generator. For example, consider the equation $\ddot{x} + x = 0$. If we set $LHS = \ddot{x}$ and $RHS = -x$, then RHS is a function of the generator and will be constantly changing as the generator is updated throughout training, and DEQGAN will become exceedingly unstable. We can fix this, however, by simply setting $LHS = \ddot{x} + x$ and taking $RHS = 0$.

Our intuition for why this happens is that if RHS depends on the outputs of the generator, the "real" data distribution $p_{\text{data}}(x)$ (from Equation 2.2) changes as the generator weights are updated throughout training. If the distribution $p_{\text{data}}(x)$ is constantly changing, the discriminator will not have a reliable signal for learning to classify "real" from "fake", which violates a core assumption of traditional GANs. By setting $RHS = 0$, we resolve the problem by effectively setting the "real" distribution to be the fixed Dirac delta function $p_{\text{data}}(x) = \delta(0)$. For the examples in this paper, we

Algorithm 1 DEQGAN

Input: Differential equation F, generator $G(\cdot;\theta_g)$, discriminator $D(\cdot;\theta_d)$, mesh t of m points with spacing Δt, perturbation precision τ, analytic adjustment function Φ, total steps N, learning rates α_G, α_D, Adam [17] optimizer parameters $\beta_{G1}, \beta_{G2}, \beta_{D1}, \beta_{D2}$
for $i = 1$ **to** N **do**
  for $j = 1$ **to** m **do**
    Perturb j-th point in mesh $t_s^{(j)} = t^{(j)} + \varepsilon, \varepsilon \sim \mathcal{N}(0, \frac{\Delta t}{\tau})$
    Forward pass $\hat{\Psi} = G(t_s^{(j)})$
    Analytic adjustment $\hat{\Psi}' = \Phi(\hat{\Psi})$ (Equation 1.7)
    Compute $LHS^{(j)} = F(t, \hat{\Psi}', \nabla\hat{\Psi}', \nabla^2\hat{\Psi}')$, set $RHS^{(j)} = 0$
  end for
  Compute gradients η_G, η_D (Equation 2.10 & 2.11)
  Update generator $\theta_g \leftarrow \texttt{Adam}(\theta_g, -\eta_G, \alpha_G, \beta_{G1}, \beta_{G2})$
  Update discriminator $\theta_d \leftarrow \texttt{Adam}(\theta_d, \eta_D, \alpha_D, \beta_{D1}, \beta_{D2})$
end for
Output: G

move all terms of the differential equation to LHS and set $RHS = 0$.*

2.5 Experiments

We run experiments on several differential equations of increasing complexity, comparing DEQGAN to the classical unsupervised neural network method using squared L_2 (i.e. mean squared error)†, L_1, and Huber [14] loss functions. Figure 2.2 plots the L_2, L_1, and Huber loss functions for reference. We report the mean squared errors of the solutions produced by each method by using known solutions obtained either analytically or through standard numerical methods, but do not use these solution data for training. We do not compare to traditional numerical methods as others have substantively performed this comparison [20,22,9,34,32,28].

*Further discussion of this point is provided in Section 3.2.1.

†We use the term L_2 to avoid conflating the *loss function* being used, which is the mean squared error on the unsupervised problem of minimizing the differential equation residuals, with the final *evaluation metric*, which is the mean squared error of the predicted solution computed against the known ground truth.

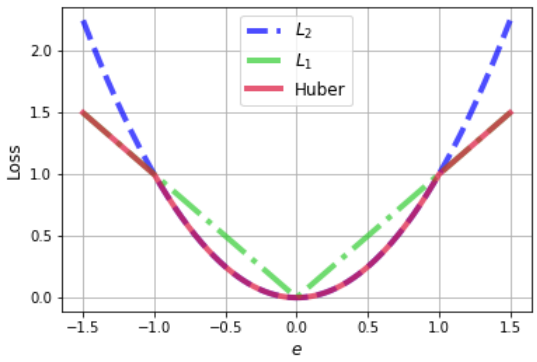


Figure 2.2: Comparison of L_2, L_1, and Huber loss functions. The Huber loss is equal to L_2 for $e \leq 1$ and to L_1 for $e > 1$.

We use feed-forward neural networks with residual connections throughout our experiments for both the generator G and discriminator D, and train them with the Adam optimizer [17], alternating equally between G and D updates. For all problems, we sample points from the domain (a grid or mesh of points) with a small Gaussian noise perturbation, as in Algorithm 1. The exact problem specifications and hyperparameter settings used, as well as the results obtained from non-GAN training, are provided in Appendix A.

We note that deterministic differential equations do not exhibit aleatoric uncertainty and that all errors we observe are therefore epistemic. Given that neural networks are universal function approximators, one may initially expect to obtain arbitrarily low error. The reason we do not observe this is two-fold. First, the universal approximation theorem holds only in the limit of infinite-width networks. Since we are constrained to implementing finite-width models, it may be that our neural networks lacks representational capacity and the problem requires a wider or deeper network. Second, our neural network loss surfaces are non-convex, and stochastic gradient descent is unlikely to reach globally optimal solutions in this scenario. Our training likely reaches local optima, or oscillates about them, and the gap between local and global optimality may further introduce epistemic error.

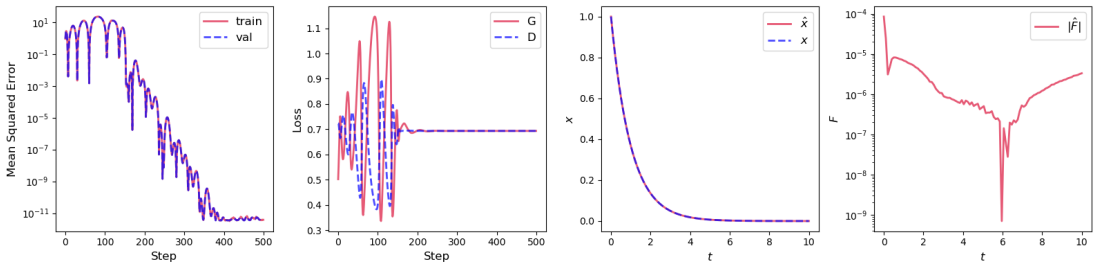


Figure 2.3: Visualization of DEQGAN training for the exponential decay problem. The left-most figure plots the mean squared error vs. iteration. To the right, we plot the value of the generator (G) and discriminator (D) losses at each iteration. Right of this we plot the prediction of the generator $\hat{x}$ and the true analytic solution x as functions of time t. The right-most figure plots the absolute value of the residual of the predicted solution $\hat{F}$.

2.5.1 Exponential Decay

Consider a model for population decay $x(t)$ given by the exponential differential equation

$$\dot{x}(t) + x(t) = 0, \tag{2.12}$$

with initial condition $x(0) = 1$. The ground truth solution $x(t) = e^{-t}$ can be obtained analytically through integration. We reiterate, however, that our method is fully unsupervised and does not make use of this solution data during training. We simply use the ground truth solutions to report mean squared errors of predicted solutions.

To set up the problem for DEQGAN, we define $LHS = \dot{x} + x$ and $RHS = 0$. We iteratively sample a collection of points from the grid $t_s \sim T$ and perform a gradient step separately for the generator and discriminator on each sample (i.e. mini-batch).

In Figure 2.3 we present the results of training DEQGAN on this problem. We observe that the generator and discriminator losses fluctuate significantly to start training, but eventually reach a stable equilibrium. The mean squared error fluctuates for both the training and validation batches, yet converges to a highly accurate solution of $\sim 10^{-11}$. Visually, the prediction of the generator $\hat{x}$

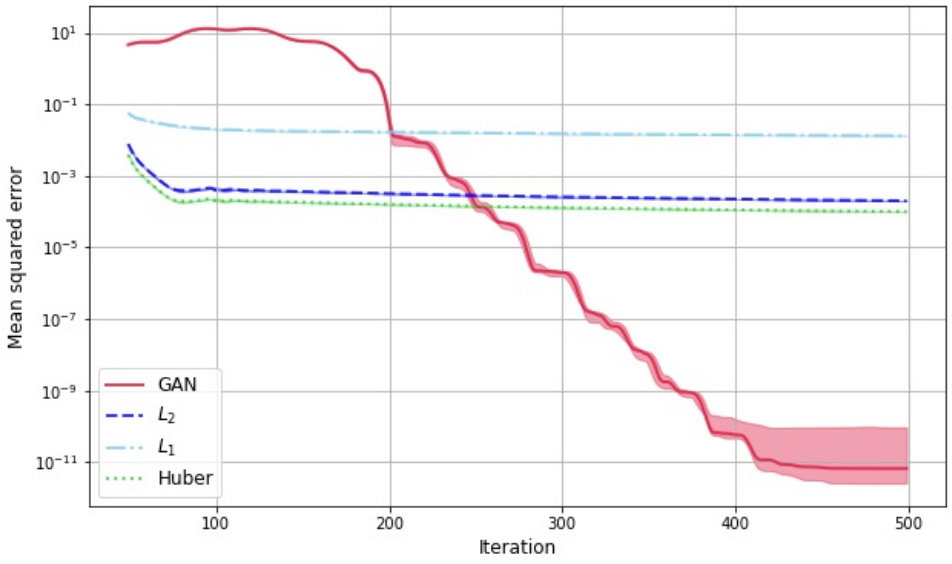


Figure 2.4: Mean squared errors vs. iteration (step) for DEQGAN, L_1, L_2, and Huber loss for the exponential decay equation. We perform ten randomized trials and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

accurately mimics the analytical solution, and the residuals of the equation $\hat{F}$ are small throughout the domain.

To compare against the classical neural network method, in Figure 2.4 we run ten trials varying the random seed used to sample batches and train both DEQGAN and the classical method with L_1, L_2 and Huber loss functions. We use the same hyperparameters across all experiments and plot the median and $(2.5, 97.5)$ percentile interval of the mean squared error of the solution at each iteration. We see that DEQGAN reaches orders of magnitude lower mean squared errors than the classical non-GAN methods[‡]. We observe that the variance of the solution accuracy is, however,

[‡]Note that the results are obtained using the best hyperparameters found for DEQGAN through a random search procedure. In Appendix A.3, we present results obtained after tuning the classical non-GAN methods for comparison. We do not observe a meaningful difference as DEQGAN still obtains multiple orders of magnitude lower mean squared errors than classical methods.

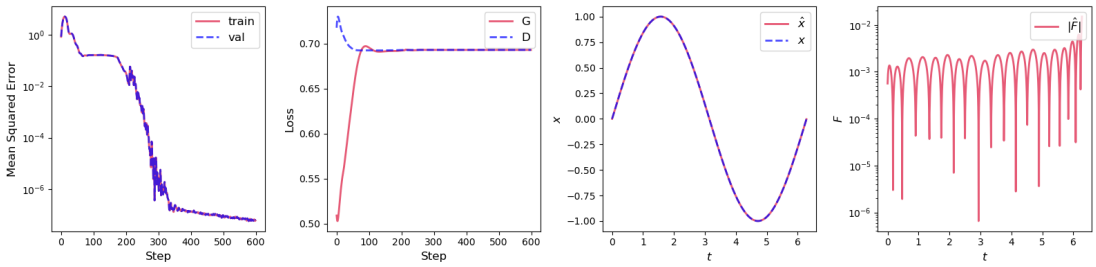


Figure 2.5: Visualization of DEQGAN training for the simple harmonic oscillator problem. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the prediction of the generator $\hat{x}$ and the true analytic solution x as functions of time t. The right-most figure plots the absolute value of the residual of the predicted solution $\hat{F}$.

notably larger for DEQGAN than the classical method across all loss functions. Nevertheless, this figure demonstrates that it is possible to significantly outperform the classical method with DEQGAN.

2.5.2 Simple Harmonic Oscillator

Now consider the motion of an oscillating body $x(t)$, which can be modeled by the simple harmonic oscillator differential equation

$$\ddot{x}(t) + x(t) = 0, \tag{2.13}$$

with initial conditions $x(0) = 0$, and $\dot{x}(0) = 1$. This differential equation can be solved analytically and has an exact solution $x(t) = \sin t$.

Here we set $LHS = \ddot{x} + x$ and $RHS = 0$ and proceed as before. We sample points from a grid, compute gradients, and perform optimization steps for the generator and discriminator separately on each batch.

Figure 2.5 plots the results of training DEQGAN on this problem. In this case, we observe that the generator and discriminator losses quickly converge to a stable equilibrium. We see that the

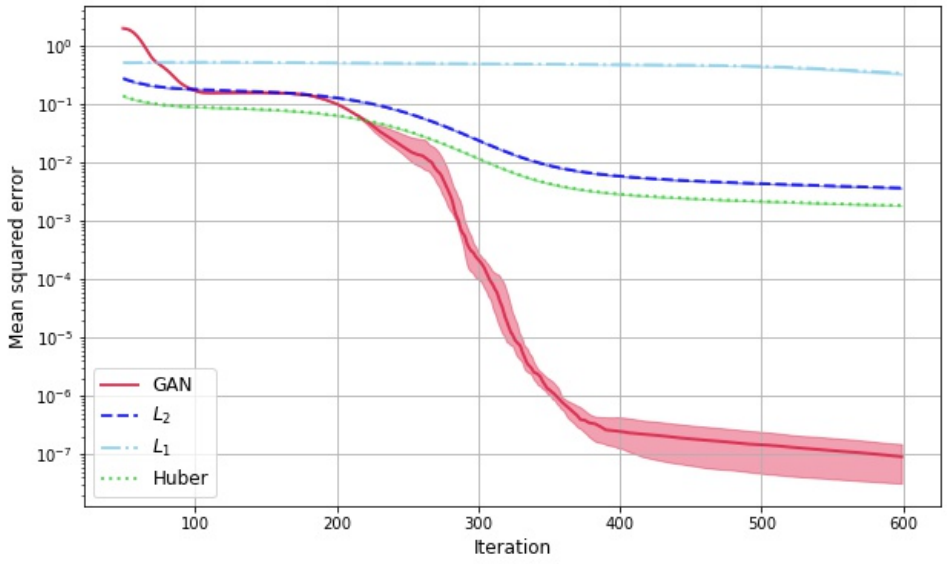


Figure 2.6: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the simple harmonic oscillator equation. We perform ten randomized trials and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

mean squared error decreases rapidly then plateaus and approaches $\sim 10^{-7}$. The predicted solution $\hat{x}$ is indistinguishable from the true solution, and the residuals of the predicted solution are small across the domain, albeit increasing slightly near the right edge of the domain ($t \approx 2\pi$).

Comparing to the classical neural network method, in Figure 2.6 we run ten randomized trials, with different random seeds for mini-batch sampling, and compute the mean squared error of the solution from each of the four methods (L_1, L_2, Huber, DEQGAN) at each iteration. We observe that our GAN method matches the performance of the classical methods initially, but by step ~ 300 outperforms the non-GAN methods by orders of magnitude in terms of solution accuracy. DEQGAN eventually converges to an accuracy of $\sim 10^{-7}$, while the best of the classical methods reaches only $\sim 10^{-3}$. We note that DEQGAN exhibits higher variance in solution accuracy, but nevertheless handily outperforms the competitors.

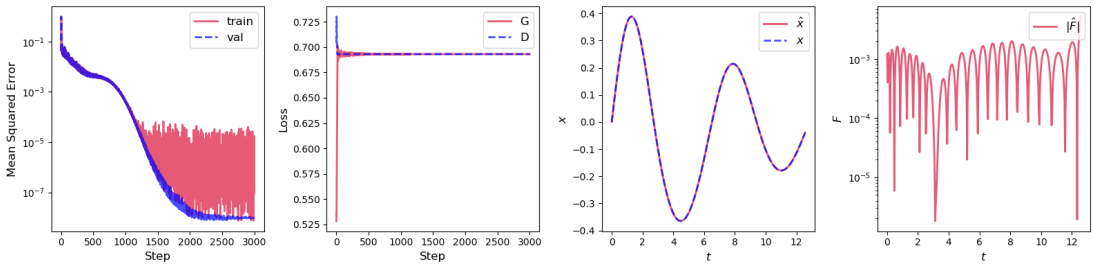


Figure 2.7: Visualization of DEQGAN training for the nonlinear oscillator problem. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the prediction of the generator $\hat{x}$ and the true analytic solution x as functions of time t. The right-most figure plots the absolute value of the residual of the predicted solution $\hat{F}$.

2.5.3 Nonlinear Oscillator

Further increasing the complexity of the differential equations being considered, let us focus on a less idealized oscillating body subject to additional forces, whose motion $x(t)$ we can described by the nonlinear oscillator differential equation

$$\ddot{x}(t) + 2\beta\dot{x}(t) + \omega^2 x(t) + \varphi x(t)^2 + \varepsilon x(t)^3 = 0, \tag{2.14}$$

with $\beta = 0.1, \omega = 1, \varphi = 1, \varepsilon = 0.1$ and initial conditions $x(0) = 0$ and $\dot{x}(0) = 0.5$. This equation does not admit an analytical solution; instead, we use the fourth-order Runge–Kutta numerical method to obtain ground truth solutions.

As before, we proceed by setting $LHS = \ddot{x} + 2\beta\dot{x} + \omega^2 x + \varphi x^2 + \varepsilon x^3 = 0$ and $RHS = 0$, and iteratively perform gradient steps to optimize the generator and discriminator separately on each mini-batch of time points sampled from a grid, as in Algorithm 1.

Figure 2.7 plots the results obtained from training DEQGAN on this nonlinear oscillator differential equation. We see that the generator and discriminator very quickly reach a stable equilibrium, and that the prediction mean squared error decreases to $\sim 10^{-8}$. We note that the mean squared

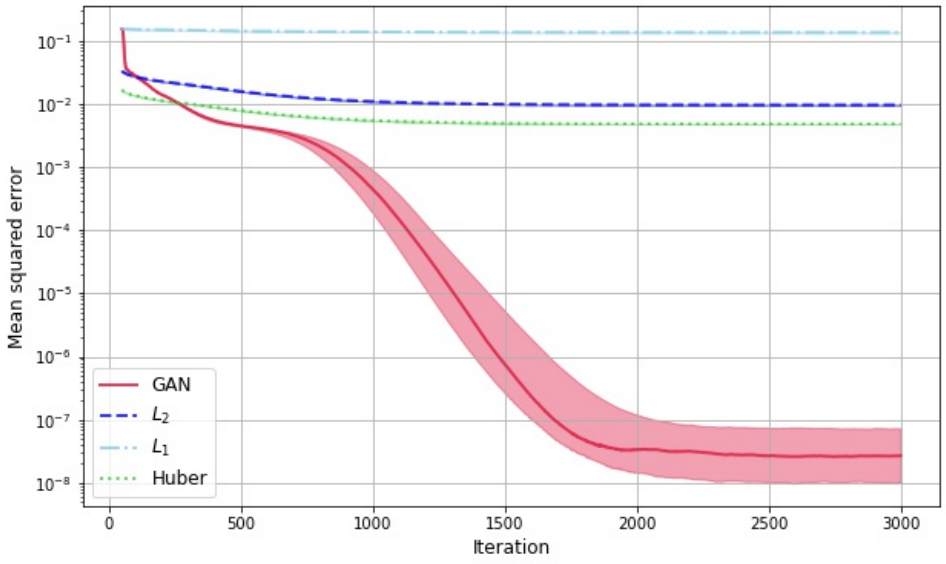


Figure 2.8: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the nonlinear oscillator equation. We perform ten randomized trials and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

error on the training set (which is the set of randomly sampled points) fluctuates sharply after step $\sim$1000, but the validation set (which is the set of fixed points) remains smooth. Nevertheless, the predicted solution is again indistinguishable from the ground truth, and the residuals of the equation from the predicted solution are small throughout the domain (increasing slightly near the right edge $t \approx 4\pi$).

To compare against the classical unsupervised neural network method, Figure 2.8 shows the results of ten trials where the grid sampling randomization is altered, and plots the mean squared error as a function of iteration for our DEQGAN method and the classical methods with L_1, L_2, and Huber loss functions. DEQGAN again outperforms the classical methods by orders of magnitude, reaching a median mean squared error $\sim 10^{-8}$, with a modest variability of $\sim 10^{-1}$ measured by the $(2.5, 97.5)$ percentile interval.

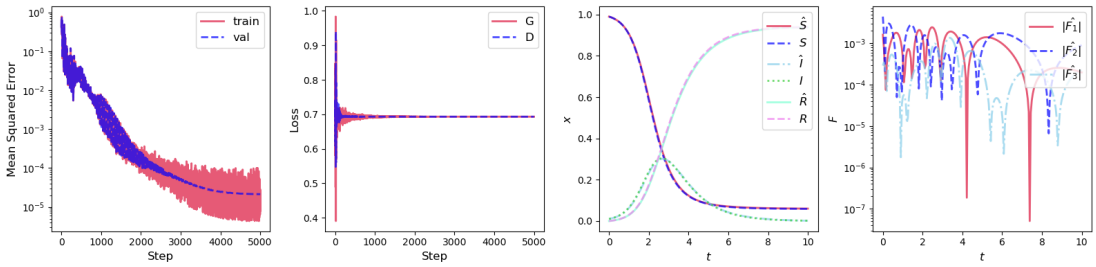


Figure 2.9: Visualization of DEQGAN training for the SIR system of equations. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the predictions of the generator $\hat{S}, \hat{I}, \hat{R}$ and the true analytic solutions S, I, R as functions of time t. The right-most figure plots the absolute value of the residuals of the predicted solution $\hat{F}_j$ for each equation j.

2.5.4 SIR Epidemiological Model

Given the recent outbreak and pandemic of novel coronavirus (COVID-19)[40], we consider an epidemiological model of infectious disease spread given by a system of ordinary differential equations. Specifically, consider the Susceptible $S(t)$, Infected $I(t)$, Recovered $R(t)$ model for the spread of an infectious disease over time t. The model is given by a system of three ordinary differential equations

$$\frac{dS}{dt} = -\beta \frac{IS}{N} \tag{2.15}$$

$$\frac{dI}{dt} = \beta \frac{IS}{N} - \gamma I \tag{2.16}$$

$$\frac{dR}{dt} = \gamma I \tag{2.17}$$

where $\beta = 3, \gamma = 1$ are given constants related to the infectiousness of the disease, $N = S + I + R$ is the (constant) total population, and the system is subject to initial conditions $S_0 = 0.99, I_0 = 0.01, R_0 = 0$.

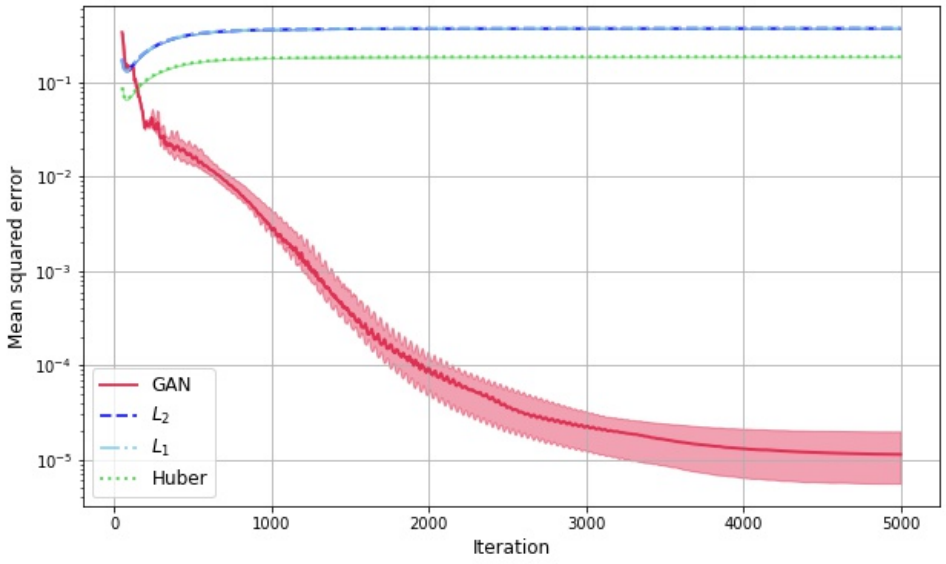


Figure 2.10: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the SIR system of equations. We perform ten randomized trials and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

Here we solve the problem in a similar manner as above, but with an important difference. Because the SIR model is given as a *system* of equations, we set LHS to be the *vector*

$$LHS = \left[\frac{dS}{dt} + \beta\frac{IS}{N}, \frac{dI}{dt} - \beta\frac{IS}{N} + \gamma I, \frac{dR}{dt} - \gamma I\right]^T \tag{2.18}$$

and $RHS = [0, 0, 0]^T$. Then instead of a scalar, the generator outputs a vector $\hat{LHS}$, and the discriminator receives this vector as input. As before, we sample points from a grid and perform gradient steps on each mini-batch for the generator and discriminator separately.

We present the results of training DEQGAN to solve this system of differential equations in Figure 2.9. We observe that the generator and discriminator losses initially fluctuate substantially, but quickly reach a stable equilibrium. We note that the mean squared error of the solutions on both

the training and validation data decreases steadily, albeit with some fluctuation on the (randomized) training data as we approach the performance plateau. The generator is able to produce accurate solutions for all three equations simultaneously and the residuals of the predicted solution are small.

To compare to the classical neural network methods, we perform ten randomized experiments where we change the random seed used for sampling mini-batches. Figure 2.10 plots the mean squared errors for each iteration of DEQGAN as well as the classical methods with L_1, L_2, and Huber loss functions. We see that DEQGAN significantly outperforms the classical methods. DEQGAN's mean squared error decreases steadily until plateauing around $\sim 10^{-5}$. The solution variance is low, suggesting that the GAN reaches stable equilibria. The classical loss functions unanimously perform poorly for this system as they collapse to the trivial solution.[§] This is a big win for DEQGAN since being able to accurately solve the system at the given initial conditions (starting with a small proportion of infected people I_0) is crucial for modeling infectious diseases in a real-world setting.

[§]See Figure A.4 in the Appendix. It is due to the extremeness of the initial conditions (S_0 being close to 1 and I_0 close to 0) that the classical methods fail and collapse to the trivial solution. If we set S_0 and I_0 to less extreme values (e.g. $S_0 = 0.7$ and $I_0 = 0.3$) the classical methods can solve the system.

3

Discussion

3.1 Instability

A point that we have not yet explicitly addressed is the apparent instability of the DEQGAN training algorithm and how it compares to the classical methods. In fact, the instability of GANs is not a new problem and much work has been dedicated to improving the stability and convergence of GANs [1,8,3,29]. In our experiments, we find that the specific initialization of the generator and dis-

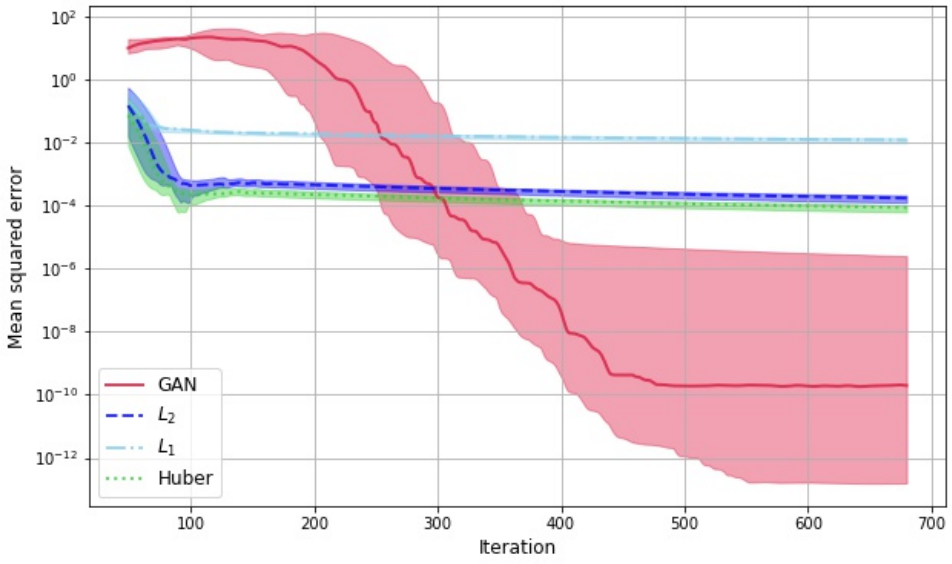


Figure 3.1: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the exponential decay equation. We perform ten randomized trials, without fixing the initial model weights, and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

criminator weights can have a substantial effect on the final performance of DEQGAN.

For example, consider Figure 3.1 which shows the results of the ten randomized trials of the exponential decay experiment, but this time without fixing the random initialization of the model weights, as was done in Figure 2.4. We observe a large increase in the variability of solution accuracy, approximately $\sim$6 orders of magnitude in the $(2.5, 97.5)$ percentile interval, compared to the results with fixed model initializations. While the classical methods also exhibit higher variability, their increase is much smaller.

We observe this effect across our experiments. The solution that we adopt is to fix the initial model weights when tuning hyperparameters for DEQGAN, and to keep the same weight initialization thereafter. Maintaining the same initial weights, even after randomization of mini-batches, appears to significantly reduce the problem of instability.

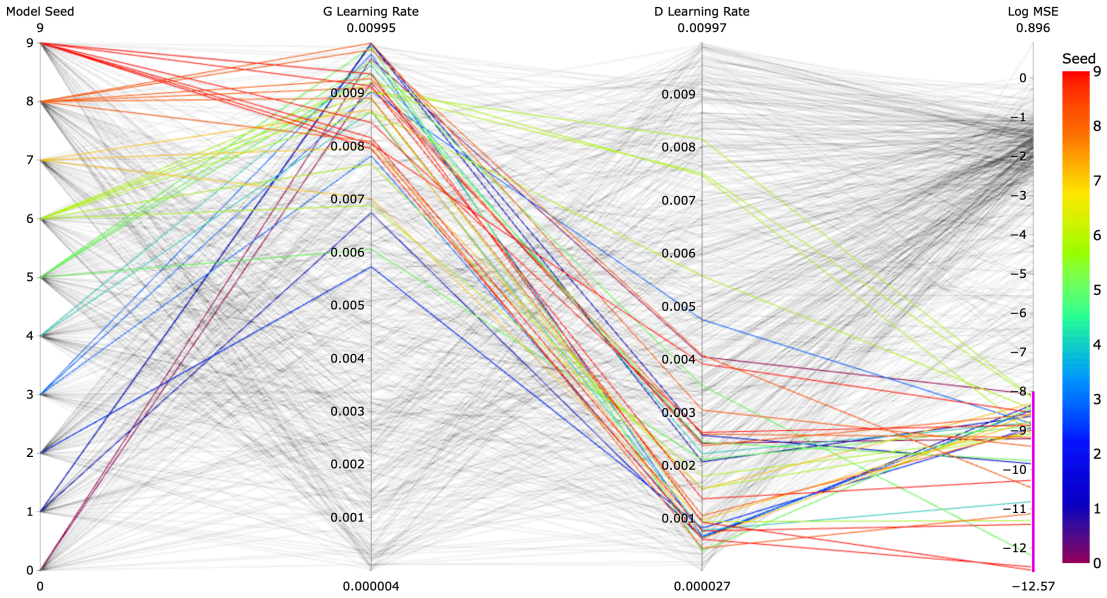


Figure 3.2: Parallel plot showing the results of 500 DEQGAN experiments on the exponential decay equation. The colors represent the different random seeds used to initialize model weights. We filter the results (non-selected lines appear gray) to highlight experiments achieving mean squared error $\leq 10^{-8}$. The mean squared error is plotted on a log_{10} scale.

To further illustrate the relationship between performance, hyperparameters, and initial model weights, Figure 3.2 plots the results of 500 DEQGAN experiments for the exponential decay equation. For each experiment, we uniformly at random select model weight initialization random seeds as integers from the range $[0, 9]$, as well as separate learning rates* for the discriminator and generator in the range $[10^{-6}, 10^{-2}]$, and record the final mean squared error on the validation set after running DEQGAN training for 500 steps.

In Figure 3.2, each line represents a combination of model weight initialization random seed, learning rate hyperparameters, and final (log) mean squared error of a single experiment. We note that the results as a whole exhibit considerable variation in final mean squared error. Interestingly, however, filtering on experiments achieving low mean squared errors ($\leq 10^{-8}$) demonstrates that

*The only tuning of DEQGAN hyperparameters on the exponential decay equation was for the discriminator and generator learning rates.

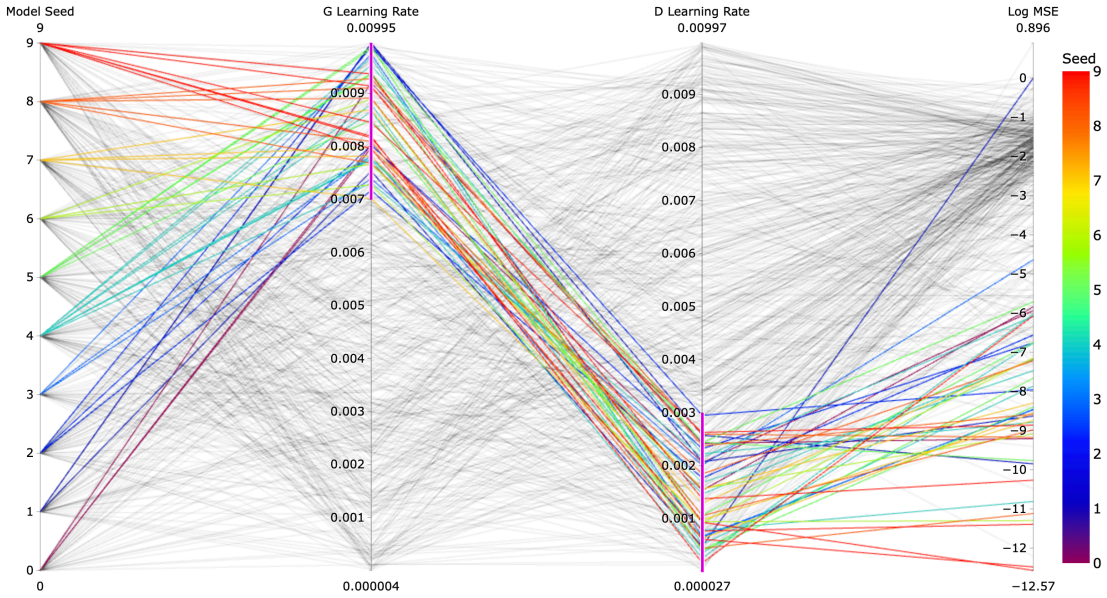


Figure 3.3: Parallel plot showing the results of 500 DEQGAN experiments on the exponential decay equation. The colors represent the different random seeds used to initialize model weights. We filter the results (non-selected lines appear gray) to highlight the experiments with relatively high generator learning rates ($\sim 10^{-2}$ to $\sim 7^{-3}$) and low discriminator learning rates ($\sim 3^{-3}$ to $\sim 3^{-5}$). The mean squared error is plotted on a log_{10} scale.

hyperparameter settings exist, for each of the model weight initialization seeds, that lead to highly accurate DEQGAN solutions.

We also observe a pattern in the hyperparameters which produce the low mean squared error experiments. We note that relatively high generator learning rates and low discriminator learning rates lead to the best DEQGAN performance, across different model initialization seeds. In Figure 3.3, we filter on high generator ($\sim 10^{-2}$ to $\sim 7^{-3}$) and low discriminator ($\sim 3^{-3}$ to $\sim 3^{-5}$) learning rates to show this pattern. We see that, while the mean squared error of a few experiments does not fall in the $\leq 10^{-8}$ range, the majority of trials with high generator and low discriminator learning rates achieve relatively low mean squared errors, and this pattern holds across the choices of random weight initialization seed.

While DEQGAN is capable of performing well across all of the initial model weights shown here,

our results confirm that optimizing DEQGAN is rather sensitive to the selection of hyperparameters. Interestingly, we observe a pattern in the discriminator and generator learning rates which seems to lead to better performance. In general, GANs lack formal guarantees of convergence and we look forward to further developments that enable GANs to be trained with greater stability and, in particular, DEQGAN to achieve even better performance.

3.2 Prior Formulations

The formulation of DEQGAN as presented in Chapter 2 is the result of many trials and errors. Here we detail some of the methods we tried which we have variously found to be either unsuitable or sub-optimal for training GANs to solve differential equations in an unsupervised manner.

3.2.1 Balancing

Initially we imagined that any reasonable partitioning of the differential equation into *LHS* and *RHS* would suffice for training DEQGAN. This showed promise, as initially we were able to significantly outperform the classical neural network method on the exponential decay equation $\dot{x}+x=0$.

We set $LHS = \dot{x}$ and $RHS = -x$, and proceeded as detailed in Algorithm 1 to train DEQGAN. As mentioned, we were able to obtain results which outperformed the classical methods we used as baselines. However, upon experimenting with another equation, the simple oscillator $\ddot{x}+x=0$, we were flummoxed to find that it simply did not work, however many sets of hyperparameters we tried.

We believe that this is because when we allow the *RHS* (which, for DEQGAN, is considered the "real" data in the GAN terminology) to vary with the generator, then the discriminator model has great difficulty in classifying real from fake. In classic GAN training, the real data come from a *fixed* distribution, allowing the discriminator to learn to classify what is real and what is not. When *RHS* varies with the generator, the real data are no longer fixed, and the GAN training becomes

exceedingly unstable, virtually incapable of convergence beyond the exponential decay equation. The solution we adopt is to move all terms to the *LHS* and set $RHS = 0$.

3.2.2 Semi-Supervised

In an attempt to counteract the difficulty we found in training DEQGAN for the simple oscillator, noted above, we tried adding some supervision in the form of "observer" solution data points. In addition to the unsupervised training signal from the discriminator, we added another loss term to the generator optimization which included actual solution values, and were compared to predictions either through a second discriminator or a point-wise loss function (we tried both).

This had the effect of constraining the space of possible generator candidate solutions by imposing a soft constaint that the solutions be close to ground truth. While this indeed improved the results, we made a serendipitous discovery, as noted above, that simply moving all terms to the *LHS* and setting $RHS = 0$ greatly improved the convergence and performance of DEQGAN, and were able to train in a fully unsupervised manner. We experimented with semi-supervised training even after this discovery and found that, to our surprise, the fully unsupervised method led to greater solution accuracy than semi-supervised. This is possibly due to the fact that unsupervised solutions require adhering to the differential equation, while supervised ones do not[†]. However, this observation may only hold for the relatively simple examples considered in our experiments, and semi-supervised training may be useful for more difficult equations.

3.2.3 Conditional GAN

In tandem with the attempts to improve training convergence of our original "balancing" formulation of DEQGAN with semi-supervised training, we hypothesized that conditioning the discrimina-

[†]Supervised solutions simply fit the given data, while unsupervised solutions must obey the differential equation by satisfying the relationship between the solution, its derivatives, and possibly other terms.

tor on the grid points (e.g. t) which produced a given real or fake solution could help the discriminator classify real from fake and thus provide a superior training signal for the generator and improve DEQGAN performance.

While we did observe this to be the case for the balancing formulation, upon re-formulating the problem to, in essence, minimize the residuals with $LHS = F\left(x, \hat{\Psi}(x), \Delta\hat{\Psi}(x), \Delta^2\hat{\Psi}(x)\right)$ and $RHS = 0$, we were able to remove this additional complexity from DEQGAN.

3.2.4 Wasserstein GAN

Prior to implementing spectral normalization in the discriminator, we followed the common practice of adopting the Wasserstein GAN [1] formulation with gradient penalty [8] (WGAN-GP, see Section 2.3) to combat optimization difficulties of the vanilla GAN. We found that this clearly enhanced our results and improved training stability. However, we discovered that spectral normalization led to even greater performance and required less tuning and fewer iterations to reach convergence, so we dispatched with WGAN-GP and used spectral normalization instead for all DEQGAN training.

4

Conclusion

Inspired by the line of research studying approaches for solving differential equations with unsupervised neural networks, and the lack of theoretical justification for a particular choice of loss function, we have presented a new method which leverages GAN-based adversarial training to *learn the loss function* in a fully unsupervised manner. We have shown empirically that our our method, which we call Differential Equation GAN (DEQGAN), can obtain multiple orders of magnitude lower mean squared errors than the classical unsupervised neural network method with (squared)

L_2, L_1, and Huber loss functions. We analyzed the stability of our approach and found it to be sensitive to hyperparameters, requiring careful tuning to reach desired performance, and discussed prior formulations to provide further insight into the development of our method.

In addition, we presented a simple point sampling technique which individually "perturbs" points from a fixed mesh according to i.i.d. Gaussian noise. We showed that this method can reduce overfitting while rendering explicit control over sample variance, enabling more stable optimization of the unsupervised loss and lower solution mean squared errors.

To further develop these ideas, future work could run experiments with more complex, potentially stochastic, differential equations. Additional robustness studies of training stability across varying initial conditions and experiments could also be useful. And investigations into more sophisticated sampling techniques, such as "active learning" approaches, in which points are sampled in proportion to their contribution to the loss function, may prove fruitful.

We are motivated by the many potential benefits of learning solutions to differential equations with unsupervised neural networks. Among the various promising developments in the literature, from neural network's advantage in high-dimensional settings to their potential superiority in obeying physical constraints, we hope that this work helps advance our understanding of the importance of loss functions and point sampling in obtaining highly accurate solutions to differential equations with unsupervised neural networks.

The deep learning revolution is upon us, and it seems that an increasing number of problems can be approached with the help of neural networks. We look to the future with great excitement, anticipating many innovative approaches that leverage neural networks to solve problems in science, engineering, and our world at large.

A

A.1 Hyperparameters

We performed random search to tune the hyperparameters of the DEQGAN method for each differential equation. We thank the creators of the Ray-Tune package [26] for the ease with which we were able to perform hyperparameter tuning using their software.

Table A.1: Hyperparameter Settings for DEQGAN: Exponential Decay

Hyperparameter	Value
Num. Iterations	500
Num. Grid Points	100
Sampling Method	Perturb
Grid Boundary	$(0, 10)$
G Units	20
G Layers	2
D Units	20
D Layers	2
G Learning Rate	0.01785332956321333
D Learning Rate	0.0025312451764215923
G Optimizer	Adam($\mathrm{B}_1 = 0.9, \mathrm{B}_2 = 0.999$)
D Optimizer	Adam($\mathrm{B}_1 = 0.9, \mathrm{B}_2 = 0.999$)
G Activations	Tanh
D Activations	Tanh
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True
Learning Rate Decay	None

Table A.2: Hyperparameter Settings for DEQGAN: Simple Oscillator

Hyperparameter	Value
Num. Iterations	600
Num. Grid Points	1000
Sampling Method	Perturb
Grid Boundary	$(0, 2\pi)$
G Units	20
G Layers	2
D Units	20
D Layers	2
G Learning Rate	0.006397509258273433
D Learning Rate	0.0001715952321721463
G Optimizer	Adam($B_1 = 0.9, B_2 = 0.999$)
D Optimizer	Adam($B_1 = 0.9, B_2 = 0.999$)
G Activations	Tanh
D Activations	Tanh
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True
Learning Rate Decay	None

Table A.3: Hyperparameter Settings for DEQGAN: Nonlinear Oscillator

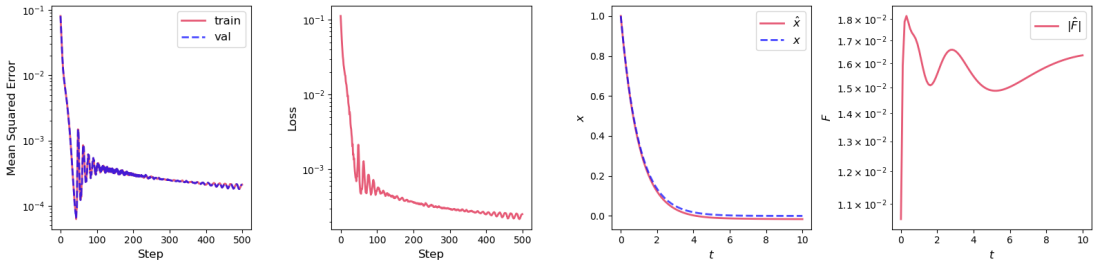
Hyperparameter	Value
Num. Iterations	3000
Num. Grid Points	400
Sampling Method	Perturb
Grid Boundary	$(0, 4\pi)$
G Units	20
G Layers	2
D Units	20
D Layers	2
G Learning Rate	0.005801628839417561
D Learning Rate	0.0007291873762250555
G Optimizer	Adam($B_1 = 0.10244627, B_2 = 0.76328835$)
D Optimizer	Adam($B_1 = 0.54142685, B_2 = 0.67750577$)
G Activations	Tanh
D Activations	Tanh
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True
Learning Rate Decay	Exp($\gamma = 0.9962712909742575$)

Table A.4: Hyperparameter Settings for DEQGAN: SIR Model

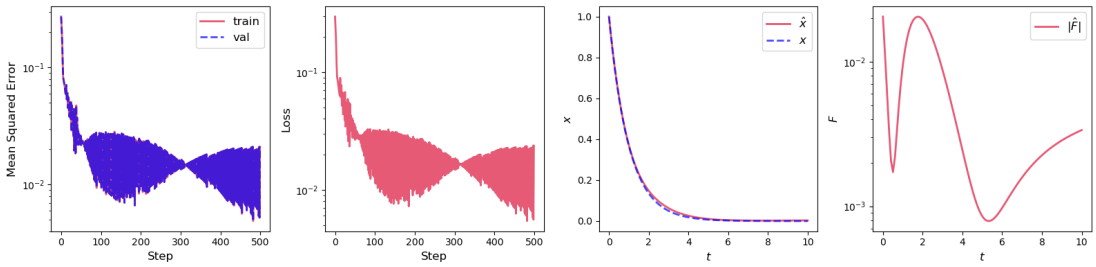
Hyperparameter	Value
Num. Iterations	5000
Num. Grid Points	200
Sampling Method	Perturb
Grid Boundary	$(0, 10)$
G Units	20
G Layers	2
D Units	20
D Layers	2
G Learning Rate	0.006429803531841584
D Learning Rate	0.0096471038124105949
G Optimizer	Adam($\mathrm{B}_1 = 0.14666949, \mathrm{B}_2 = 0.93261048$)
D Optimizer	Adam($\mathrm{B}_1 = 0.51750004, \mathrm{B}_2 = 0.85405624$)
G Activations	Tanh
D Activations	Tanh
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True
Learning Rate Decay	Exp($\gamma = 0.9976839429505154$)

A.2 Non-GAN Training

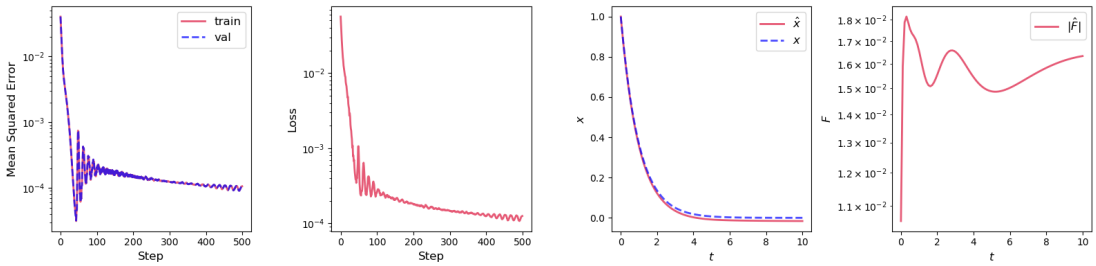
Here we present training visualizations for each of the non-GAN methods (L_2, L_1, Huber) which we compared with in Chapter 2.



(a) Squared L_2 (mean squared error) loss.

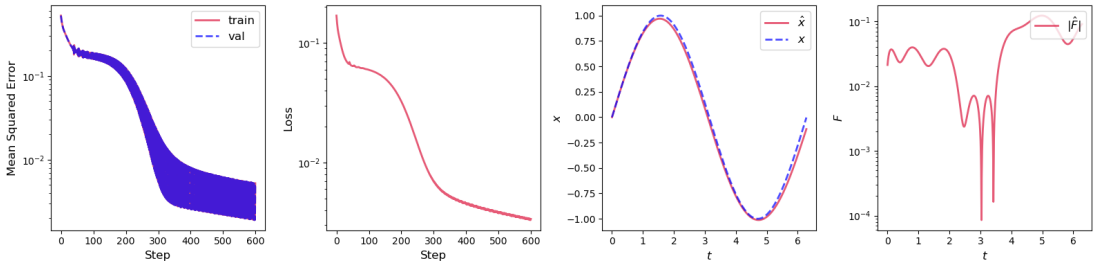


(b) L_1 loss.

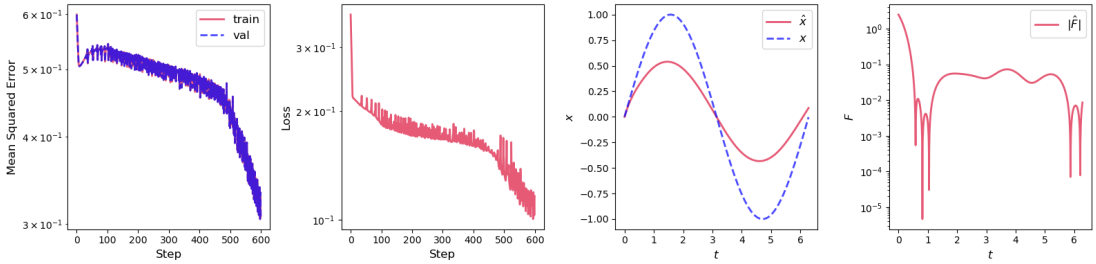


(c) Huber loss.

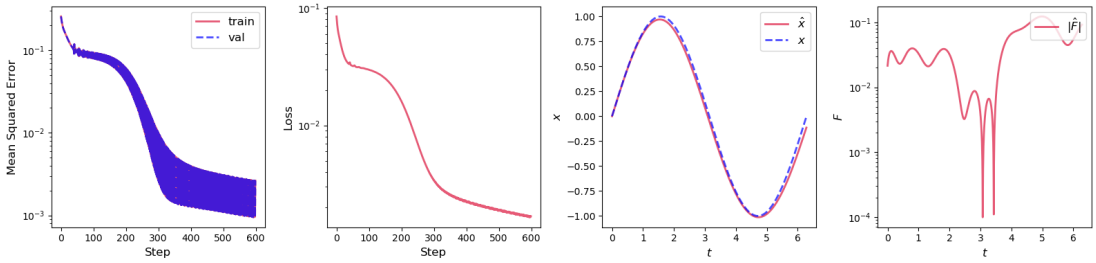
Figure A.1: Visualization of fully-connected neural networks trained with various losses to solve the exponential decay differential equation. The left-most figures plot the mean squared error vs. step (iteration) count. To the right of this, we plot the unsupervised loss for each step. Right of this we plot the prediction of the model $\hat{x}$ and the true analytic solution x as functions of time t. The right-most figures plot the absolute value of the residual of the predicted solution $\hat{F}$.



(a) Squared L_2 (mean squared error) loss.

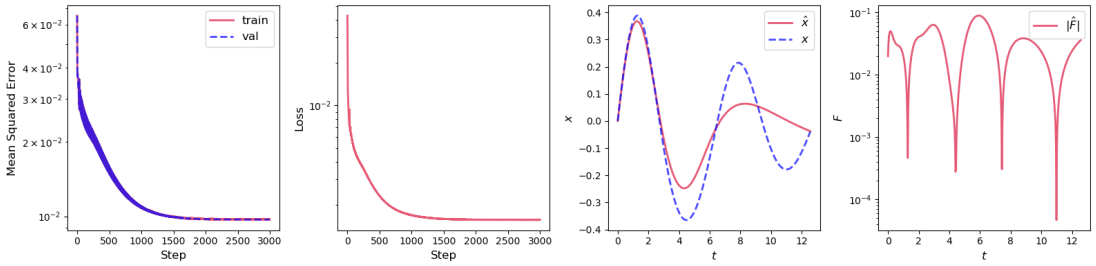


(b) L_1 loss.

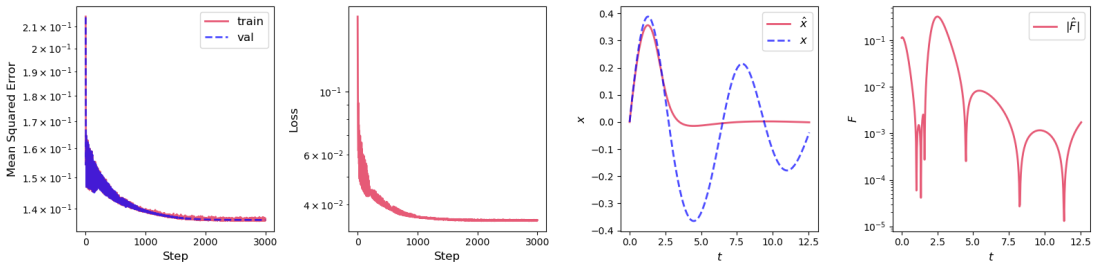


(c) Huber loss.

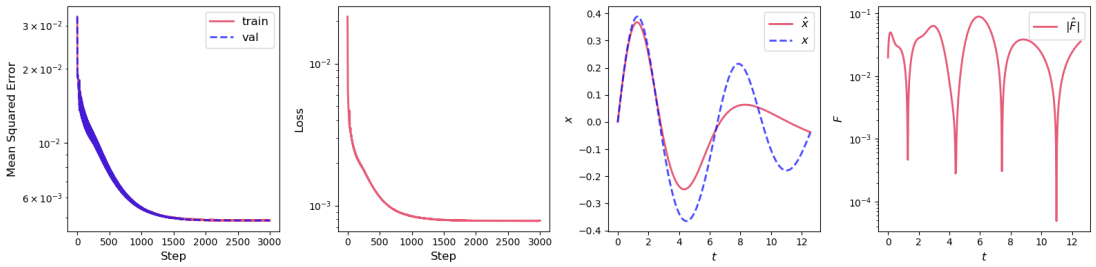
Figure A.2: Visualization of fully-connected neural networks trained with various losses to solve the simple oscillator differential equation. The left-most figures plot the mean squared error vs. step (iteration) count. To the right of this, we plot the unsupervised loss for each step. Right of this we plot the prediction of the model $\hat{x}$ and the true analytic solution x as functions of time t. The right-most figures plot the absolute value of the residual of the predicted solution $\hat{F}$.



(a) Squared L_2 (mean squared error) loss.

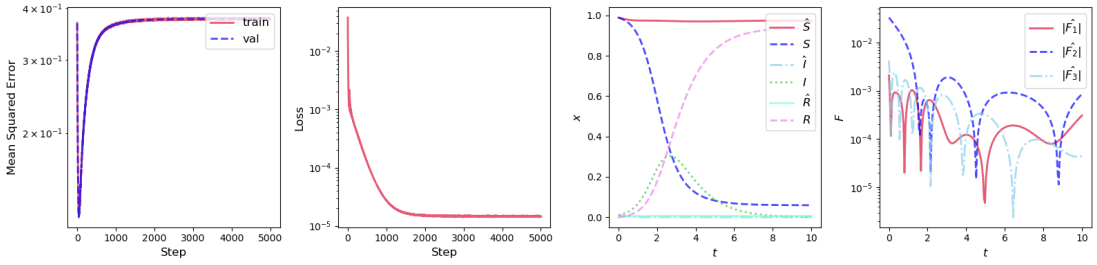


(b) L_1 loss.

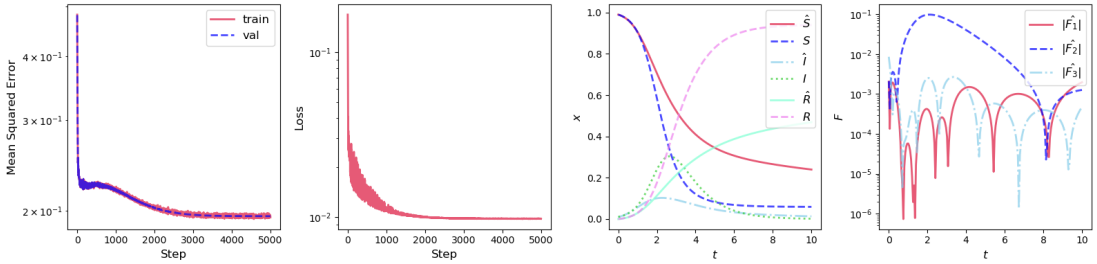


(c) Huber loss.

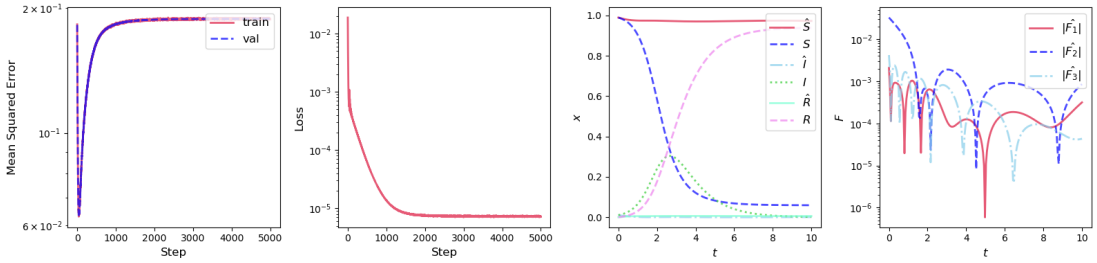
Figure A.3: Visualization of fully-connected neural networks trained with various losses to solve the nonlinear oscillator differential equation. The left-most figures plot the mean squared error vs. step (iteration) count. To the right of this, we plot the unsupervised loss for each step. Right of this we plot the prediction of the model $\hat{x}$ and the true analytic solution x as functions of time t. The right-most figures plot the absolute value of the residual of the predicted solution $\hat{F}$.



(a) Squared L_2 (mean squared error) loss.



(b) L_1 loss.



(c) Huber loss.

Figure A.4: Visualization of fully-connected neural networks trained with various losses to solve the SIR system of differential equations. The left-most figures plot the mean squared error vs. step (iteration) count. To the right of this, we plot the unsupervised loss for each step. Right of this we plot the predictions of the model $\hat{S}, \hat{I}, \hat{R}$ and the true analytic solutions S, I, R as functions of time t. The right-most figures plot the absolute value of the residual of the predicted solution $\hat{F}_j$ for each equation j.

A.3 Non-GAN Tuning

Here we present DEQGAN vs. non-GAN performance comparisons obtained from tuning the non-GAN methods with a random search procedure similar to the one used for DEQGAN.

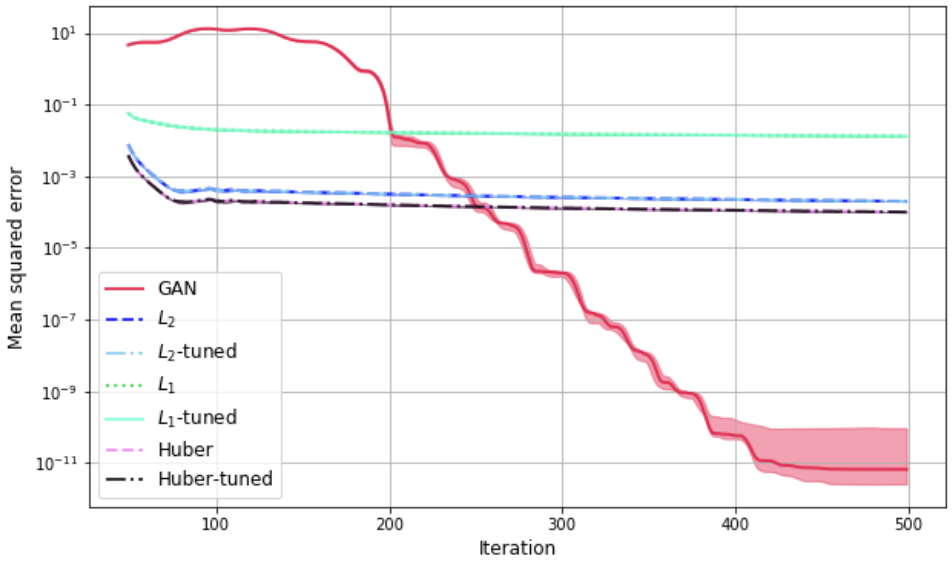


Figure A.5: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the exponential decay equation, with separate hyperparameters tuned for the non-GAN methods. We perform ten randomized trials and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

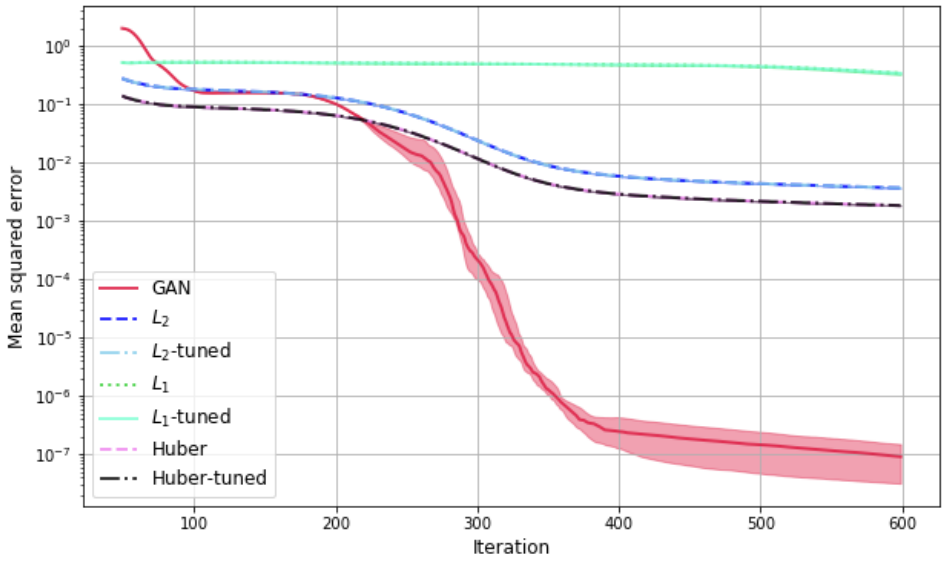


Figure A.6: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the simple harmonic oscillator equation, with separate hyperparameters tuned for the non-GAN methods. We perform ten randomized trials and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

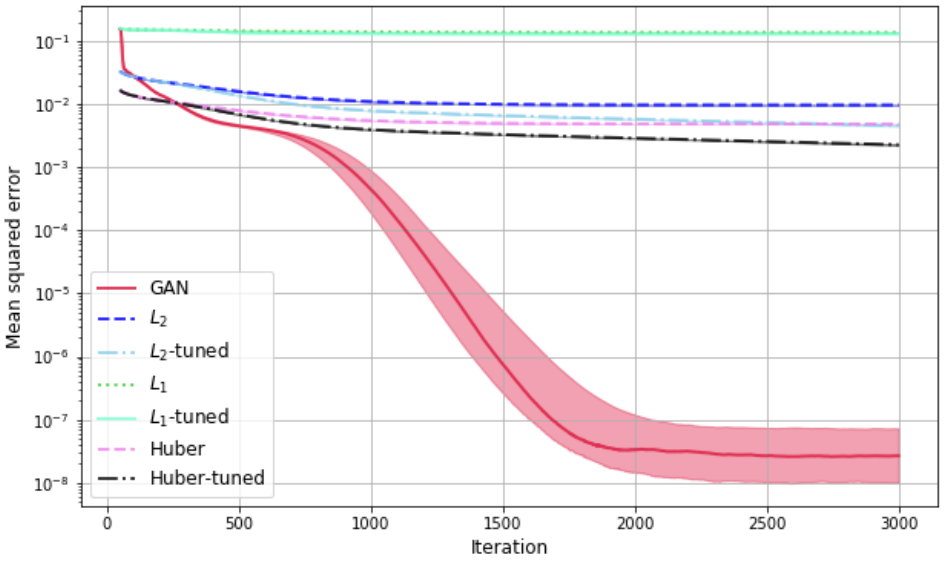


Figure A.7: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the nonlinear oscillator equation, with separate hyperparameters tuned for the non-GAN methods. We perform ten randomized trials and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

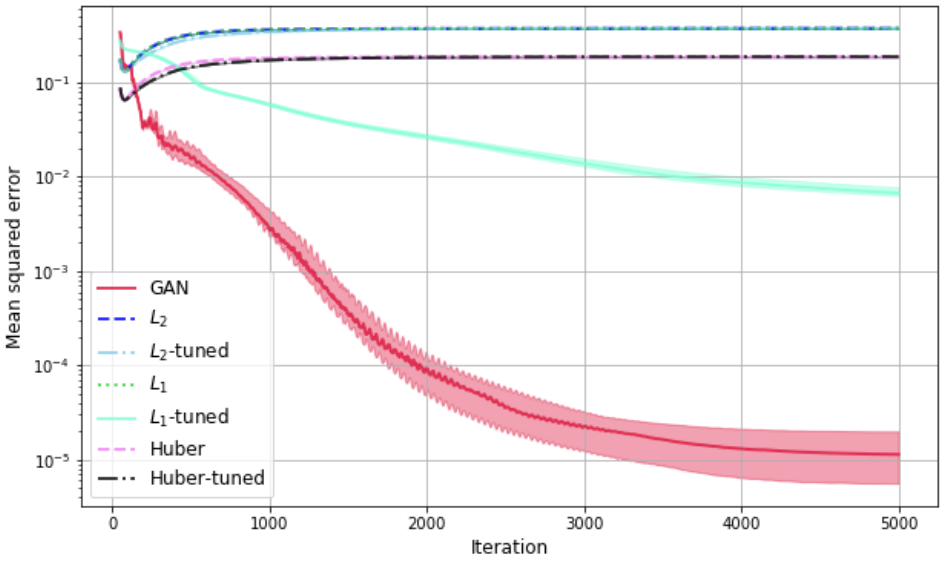


Figure A.8: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the SIR system of equations, with separate hyperparameters tuned for the non-GAN methods. We perform ten randomized trials and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

References

[1] Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan.

[2] Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[3] Berthelot, D., Schumm, T., & Metz, L. (2017). BEGAN: boundary equilibrium generative adversarial networks. *CoRR*, abs/1703.10717.

[4] Blazek, J. (2015). *Computational fluid dynamics: principles and applications*. Butterworth-Heinemann.

[5] Dabney, W., Rowland, M., Bellemare, M. G., & Munos, R. (2018). Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[6] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks.

[7] Gu, S., Holly, E., Lillicrap, T., & Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)* (pp. 3389–3396).: IEEE.

[8] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. (2017). Improved training of wasserstein gans.

[9] Han, J., Jentzen, A., & Weinan, E. (2017). Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning. *ArXiv*, abs/1707.02568.

[10] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[11] Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception* (pp. 65–93). Elsevier.

[12] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Klambauer, G., & Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500.

[13] Hornik, K., Stinchcombe, M., White, H., et al. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5), 359–366.

[14] Huber, P. J. (1964). Robust estimation of a location parameter. *Ann. Math. Statist.*, 35(1), 73–101.

[15] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (pp. 1725–1732).

[16] Karras, T., Laine, S., & Aila, T. (2018). A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948.

[17] Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[18] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 25* (pp. 1097–1105). Curran Associates, Inc.

[19] Kumar, M. & Yadav, N. (2011). Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: a survey. *Computers & Mathematics with Applications*, 62(10), 3796–3811.

[20] Lagaris, I., Likas, A., & Fotiadis, D. (1998a). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5), 987–1000.

[21] Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1997). Artificial neural network methods in quantum mechanics.

[22] Lagaris, I. E., Likas, A., & Papageorgiou, D. G. (1998b). Neural network methods for boundary value problems defined in arbitrarily shaped domains. *CoRR*, cs.NE/9812003.

[23] Lakhani, P. & Sundaram, B. (2017). Deep learning at chest radiography: automated classification of pulmonary tuberculosis by using convolutional neural networks. *Radiology*, 284(2), 574–582.

[24] Larsen, A. B. L., Sønderby, S. K., & Winther, O. (2015). Autoencoding beyond pixels using a learned similarity metric. *CoRR*, abs/1512.09300.

[25] Ledig, C., Theis, L., Huszar, F., Caballero, J., Aitken, A. P., Tejani, A., Totz, J., Wang, Z., & Shi, W. (2016). Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802.

[26] Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., & Stoica, I. (2018). Tune: A research platform for distributed model selection and training. *CoRR*, abs/1807.05118.

[27] Mattheakis, M., Protopapas, P., Sondak, D., Giovanni, M. D., & Kaxiras, E. (2019). Physical symmetries embedded in neural networks.

[28] Mattheakis, M., Sondak, D., Dogra, A. S., & Protopapas, P. (2020). Hamiltonian neural networks for solving differential equations.

[29] Mirza, M. & Osindero, S. (2014). Conditional generative adversarial nets.

[30] Miyato, T., Kataoka, T., Koyama, M., & Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957.

[31] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[32] Raissi, M. (2018). Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv preprint arXiv:1804.07010*.

[33] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419), 1140–1144.

[34] Sirignano, J. & Spiliopoulos, K. (2018a). Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375, 1339–1364.

[35] Sirignano, J. A. & Spiliopoulos, K. (2018b). Dgm: A deep learning algorithm for solving partial differential equations.

[36] Subramanian, A., Wong, M.-L., Borker, R., & Nimmagadda, S. (2018). Turbulence enrichment using generative adversarial networks.

[37] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

[38] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

[39] Villani, C. (2008). *Optimal transport: old and new*, volume 338. Springer Science & Business Media.

[40] Wang, C., Horby, P. W., Hayden, F. G., & Gao, G. F. (2020). A novel coronavirus outbreak of global health concern. *The Lancet*, 395(10223), 470–473.

[41] Yang, L., Zhang, D., & Karniadakis, G. E. (2018). Physics-informed generative adversarial networks for stochastic differential equations.

[42] Zhu, J., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593.

This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. The above illustration, "Science Experiment 02", was created by Ben Schlitter and released under CC BY-NC-ND 3.0. A template that can be used to format a PhD thesis with this look and feel has been released under the permissive MIT (X11) license, and can be found online at github.com/suchow/Dissertate or from its author, Jordan Suchow, at suchow@post.harvard.edu.