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 differentiable functions which solve differential equations has vast application, and neural networks in particular may provide a range of benefits over traditional numerical methods, ranging from high-dimensional performance to computational efficiency.

This thesis develops a new method for solving differential equations with unsupervised neural networks that leverages 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 outperform 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 accuracy. Finally, we include a discussion of various iterations of our method to provide insights into our approach.[*]

[*]All code, models, and hyperparameters will be made available.

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 set of functions, and thereby define a relationship between a quantity and its rate 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 over the past century. Recently, differential equations have been combined with sophisticated numerical algorithms and powerful computers to study challenging problems such as fluid flow.

Concurrently, recent advances in so-called "deep learning"* have led to major successes in areas of computer vision [17,14], natural language processing [36,2,37], and control tasks such as video games, robotics, and even the ancient game of Go [30,4,6,32]. This has led many researchers to apply neural networks to problems outside the domain of traditional computer science and machine learning, including in medicine [22].

Inspired jointly by these successes, and the provable distinction that neural networks are universal function approximators [12], 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, learning a *differentiable* function which solves a differential equation can in principle allow us to tackle inverse problems by differentiating backwards from solutions to initial conditions. Furthermore, forward passes of neural networks are embarrassingly data-parallel and can easily leverage parallel computing architectures for efficient speedups, enabling faster computation of solutions. Moreover, Mattheakis et al. [27] show that neural networks can outperform traditional numerical methods in terms of obeying certain constraints, such as conservation of energy. Finally, 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 [33,31,8].

The initial idea for solving differential equations with neural networks was proposed by Lagaris et al. [19]. They showed that a neural network method could outperform the finite element method

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

in terms of interpolation accuracy while still maintaining equal solution accuracy on a fixed mesh. Others have further developed their 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[21], and applied neural networks to quantum mechanics[20]. Recent work by Sirignano & Spiliopoulos[34] 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.[26] embedded physical symmetries into the structure of neural networks to improve training convergence and solution accuracy. Kumar & Yadav[18] 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 considerable impact on solution quality. Second, we present a new method which trains Generative Adversarial Networks[5] (GANs) in a fully unsupervised manner to solve differential equations. We argue that *learning the loss function* in this manner 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

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. Figure 1.1 shows a variety of possible fully-

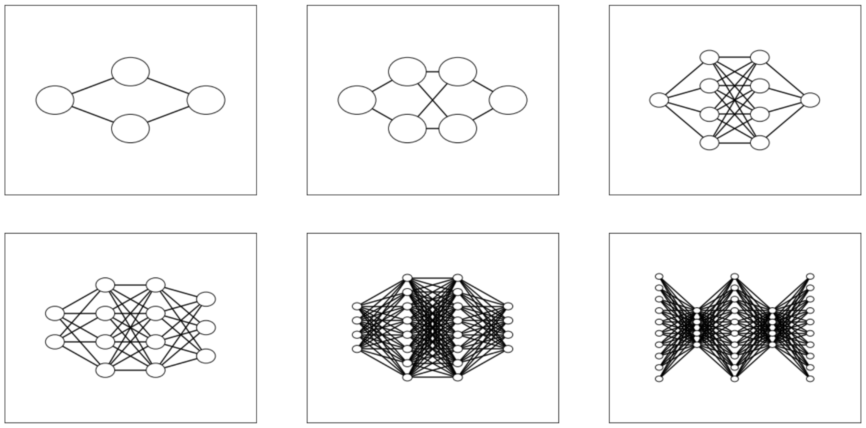


Figure 1.1: Examples of fully-connected neural network architectures. Neural networks can be composed of arbitrary numbers of layers and neurons per layer. The first (left-most) layer is called the input layer while the last (right-most) layer is called the output layer.

connected* neural network architectures.

Neural nets have been shown, in the limit of infinitely many neurons, to be capable of approximating any reasonable function [12]. This property has led researchers to doggedly pursue neural networks for many years, developing efficient techniques, most notably "backpropagation" [10], for training these often high-dimensional models.

Early work by Lagaris et al. [19] proposed solving differential equations in an unsupervised manner with neural networks. The paper considers general differential equations of the form

$$F(x, \Psi(x), \Delta\Psi(x), \Delta^2\Psi(x)) = 0 \tag{1.1}$$

where Δ and Δ^2 represent first and second derivatives, and the system is subject to certain boundary or initial conditions. The learning problem is then formulated as minimizing the squared error

*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 $i + 1$.

of the above equation

$$\min_{\theta} \sum_{x \in D} F(x, \Psi_\theta(x), \Delta\Psi_\theta(x), \Delta^2\Psi_\theta(x))^2 \tag{1.2}$$

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

$$\ddot{x}(t) + x(t) = 0 \tag{1.3}$$

with initial conditions $x_0 = 0$ and $\dot{x}_0 = 0.5$. This equation has an exact analytical solution of the form $x(t) = sin(t)$.

We can solve this equation without access to training data in the form of ground truth solutions using a neural network which minimizes the loss function

$$\min_{\theta} \sum_{t \in T} \left(\ddot{\hat{x}}(t) + \hat{x}(t)\right)^2 \tag{1.4}$$

where $f_\theta(t) = \hat{x}(t)$ is the output of a neural network parameterized by θ, T specifies the time domain over which to solve 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 $\ddot{\hat{x}} = \frac{\partial^2 \hat{x}}{\partial t^2}$.

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

[†]Hidden layers are the layers which are in between the input and output layers.

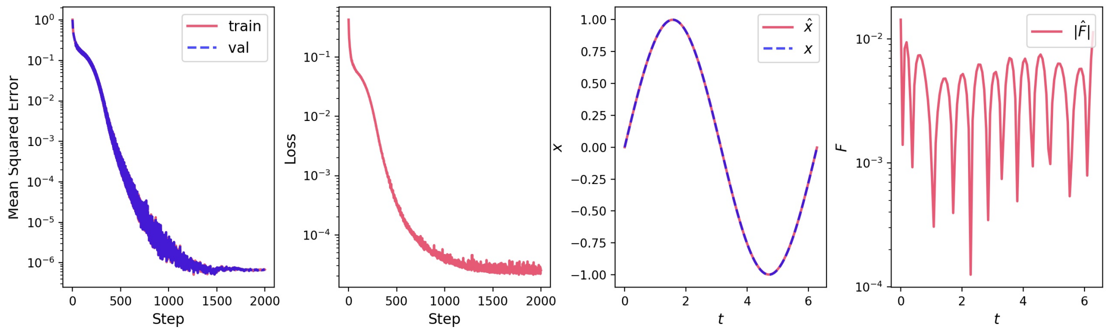


Figure 1.2: Visualization of a fully-connected neural network trained to solve the simple oscillator differential equation. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the unsupervised loss for each step. Right of this we plot the prediction 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 $\hat{F}$.

the form of solution data. The neural network learns to solve the equation simply by minimizing the residuals as shown in Equation 1.4.

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 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.[‡]

1.1 Adjusting for Physical Constraints

Mattheakis et al.[26] show that it is possible to embed physical constraints into neural networks to directly optimize for solutions which are physically realizable. They demonstrate that their method improves convergence of neural networks trained to solve differential equations. Throughout this

[‡]For more information, see e.g. Lagaris et al.[19], Mattheakis et al.[26].

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 $\tilde{N}(t)$ to satisfy the initial condition $\tilde{N}(t_0) = x_0$. This is achieved by applying the transformation

$$\hat{x} = x_0 + (1 - e^{-(t-t_0)})\tilde{N}(t) \tag{1.5}$$

Intuitively, equation 1.5 adjusts the output of the neural network $\tilde{N}(t)$ 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

When moving beyond very simple differential equations, it is helpful to think of ways to improve neural network training convergence. One method we have developed leverages the fact that, with neural networks for solving differential equations, we are free to flexibly choose points at which to train the model.

The vanilla approach to training a neural network to solve a differential equation simply considers a fixed mesh of points over which to minimize the unsupervised los (the residuals of the equation). However, we find that it is often beneficial to instead perform sampling of the points to improve training and solution accuracy.

Specifically, the main approach which we leverage throughout the remainder of this work is to "perturb" the points of our initial mesh with Gaussian noise. Concretely, for each point in the mesh, we add a small term ε sampled from a Normal distribution:

$$\varepsilon \sim \mathcal{N}(0, (\Delta x/3)^2) \tag{1.6}$$

where Δx is the inter-point spacing of the original mesh. This has the effect of randomly shifting the training points, reducing overfitting to the fixed mesh, while ensuring with $\sim$ 99% probability that points do not cross into neighboring regions, reducing the overall variance of the sampling procedure. We find that this significantly improves convergence for more complex differential equations, such as the one presented below.

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

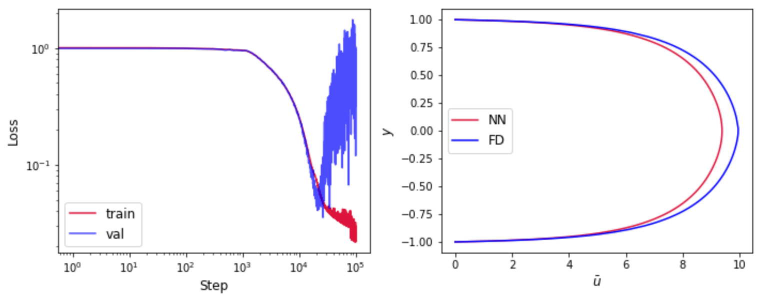
$$\nu \frac{d^2u}{dy^2} - \frac{d}{dy}\left((\kappa y)^2 \left|\frac{du}{dy}\right| \frac{du}{dy}\right) - \frac{1}{\rho}\frac{dp}{dx} = 0 \tag{1.7}$$

where ν, κ, ρ are constants and $\frac{dp}{dx}$ is a given pressure gradient.

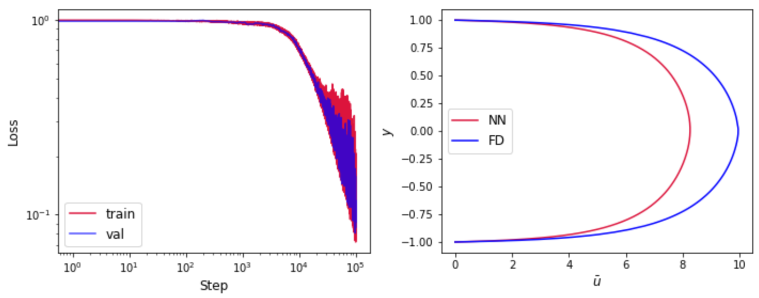
Figure 1.3 shows the results of training a three layer neural network with 30 units per layer to solve this problem. We compare three strategies for choosing points: a fixed mesh where the points are always the same, uniform random sampling where every point is sampled i.i.d. from a uniform distribution whose support is 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 in the captions.

We observe that, for fixed grid sampling, the network overfits to the training points as the training loss decreases but the validation loss increases. With uniform sampling, the problem of overfitting is decreased, yet the solution accuracy is lower (the neural network solution is further from the ground truth finite difference solution). With our perturbation approach, overfitting is decreased–both training and validation losses decrease together–and solution accuracy is higher than either of the fixed or uniform sampling methods.

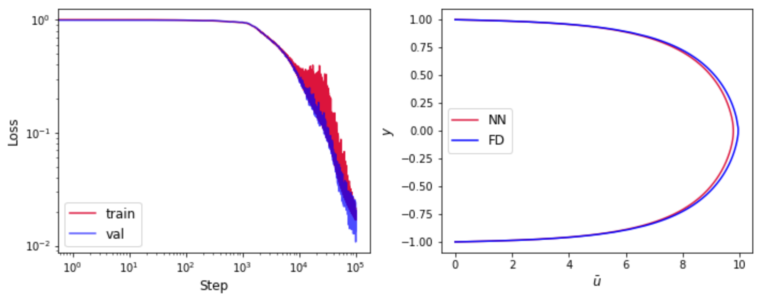
Based on these results, we apply the perturbed sampling method for all problems considered



(a) Fixed mesh. Solution mean squared error = 0.224.



(b) Uniform sampling. Solution mean squared error = 2.15.



(c) Perturbed sampling. Solution mean squared error = 0.0173.

Figure 1.3: Comparison of three sampling strategies: fixed, uniform, and perturbed. The left panel plots the (unsupervised) training and validation 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 most accurate results while controlling overfitting.

throughout the remainder of this work.

2

Generative Adversarial Networks: Learning The Loss Function

HERE WE PRESENT A NEW METHOD for solving differential equations with neural networks in an unsupervised manner, based on Generative Adversarial Networks (GANs)[5]. Given the plethora of potential loss functions, and the lack of–to the best of our knowledge–any formal justification for a particular choice, we propose *learning the loss function* with GANs.

2.1 Introduction

In the classic setting of data that is assumed to follow a Gaussian noise model describing the variability in the data, it is straightforward to show that the maximum likelihood estimate for the model parameters is that which minimizes the mean squared error between the predicted solution and actual ground truth. However, in the case of differential equations there is no noise model, and we lack formal justification to ensure that our choice of loss function is a principled one.

To circumvent this problem, we propose GANs for solving differential equations in an unsupervised manner. We think of the discriminator model in a GAN as learning the appropriate loss function for the generator and the problem at hand. We present empirical results demonstrating that our method can outperform, sometimes dramatically, the results obtained by classical methods for solving differential equations with neural networks in an unsupervised manner.

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[23,24,15]. We hypothesize that using the rich function space of learned neural network loss functions could yield more accurate solutions to differential equations due to the potentially complex spatio-temporal dependencies present in actual solutions to some differential equations.

Our main contribution is a method, which we call Differential Equation GAN (DEQGAN), 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 orders of magnitude lower mean squared errors (computed from known analytic or numerical solutions) than comparable classical unsupervised methods which use mean squared error (squared L_2), L_1, and Huber loss functions.

2.2 Related Work

Goodfellow et al.[5] 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[28] proposed Conditional GANs which introduce auxiliary conditioning information (e.g. class labels) to enable generative models of conditional distributions and reduce the problem of "mode collapse". Arjovsky et al.[1] introduced WGAN, a formulation of GANs based on the Wasserstein distance as loss function and showed that this led to improved training stability and output quality. Gulrajani et al.[7] 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.[29] proposed a spectral normalization technique to enforce to the Lipschitz constraint that outperforms WGAN on some problems.

Our work distinguishes itself from other GAN-based approaches to 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 physical data gathered by simulation or experiment. Yang et al.[40] apply GANs to stochastic differential equations but include "snapshots" of ground-truth data for training. A project by students at Stanford[35] employed GANs to perform "turbulence enrichment" in a manner akin to that of super-resolution for images proposed by Ledig et al.[24]. However, their method uses solution data when training the GAN.

2.3 Background

2.3.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs)[5] 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[24,15,41].

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

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[1 - \log D(G(z))] \tag{2.1}$$

where $x \sim p_{\text{data}}(x)$ represents samples from the empirical data distribution and $p_z \sim \mathcal{N}(0,1)$ samples from the standard normal distribution.

2.3.2 Conditional GAN

Conditional GANs[28] 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

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

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.[11] proposed a two time-scale update rule (TTUR) for training GANs, a method in which the discriminator and generator use separate learning rates. They showed that their method leads to improved performance and proved that, in some cases, TTUR ensures convergence to a stable local Nash equilibrium. We make use of TTUR throughout this paper as an instrumental lever to tune our GANs to reach a desired performance.

2.3.4 Wasserstein Loss with Gradient Penalty

Motivated by the instability of optimizing (2.1), Arjovsky et al.[1] propose the Wasserstein GAN (WGAN), 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[38] to obtain

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim p_{\text{real}}(x)} \left[D(x)\right] - \mathbb{E}_{z \sim p(z)} [D(G(z))] \tag{2.3}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions. To enforce the Lipschitz constraint, Gulrajani et al.[7] 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 the Lipschitz constraint. The updated discriminator loss becomes

$$L_D = \mathbb{E}_{z\sim p(z)}[D(G(z))] - \mathbb{E}_{x\sim p_{\text{real}}(x)}[D(x)] + \lambda \mathbb{E}_{\hat{x}\sim \mathbb{P}_{\hat{x}}}\left[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2\right] \tag{2.4}$$

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 $z \sim p(z)$.

2.3.5 Spectral Normalization

To combat the instability of training GANs, Miyato et al.[29] proposed a novel weight normalization technique, Spectrally Normalized GANs (SN-GANs), which they apply to the discriminator. Their method is light and simple to apply, and in our experiments leads to even better performance than WGAN-GP.

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

$$W_{SN} = \frac{W}{\sigma(W)}, \tag{2.5}$$

where

$$\sigma(W) = \max_{||h||_2 \leq 1} ||Wh||_2 \tag{2.6}$$

The authors prove that this normalization technique leads to the Lipschitz constant of the discriminator being bounded above by 1.

2.3.6 Residual Connections

He et al.[9] 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 throughout the models and improve performance. Residual connections augment a typical activation with the identity operation:

$$y = \mathcal{F}(x, W_i) + x \tag{2.7}$$

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 whatsoever 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 $x \sim \mathcal{D}$ and use them as input to a generator $G(x)$, which produces candidate solutions $\hat{\psi}$. We adjust $\hat{\psi}$ for initial or boundary conditions according to the analytic formula in section 1.1. Then we construct the *LHS* from the differential equation F using automatic differentiation

$$LHS = F\left(x, \hat{\psi}(x), \Delta\hat{\psi}(x), \Delta^2\hat{\psi}(x)\right) \tag{2.8}$$

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 generator

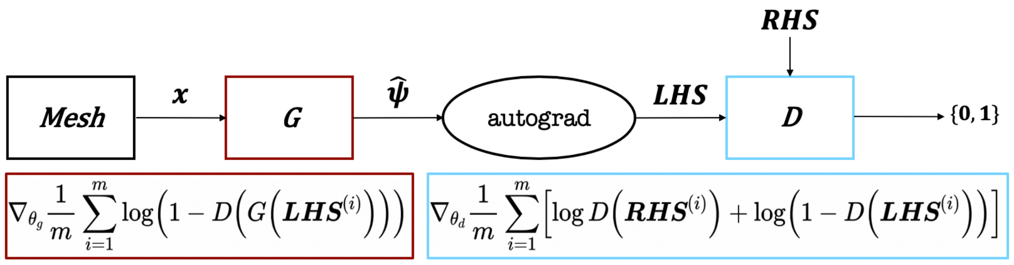

RHS
Mesh
x
G
$\hat{\psi}$
autograd
LHS
D
{0, 1}
$\nabla_{\theta_g} \frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(G\left(\boldsymbol{LHS}^{(i)}\right)\right)\right)$
$\nabla_{\theta_d} \frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(\boldsymbol{RHS}^{(i)}\right)+\log\left(1-D\left(\boldsymbol{LHS}^{(i)}\right)\right)\right]$

Figure 2.1: Schematic representation of DEQGAN.

Algorithm 1 DEQGAN

```
**Input:** Differential equation F, generator $G(.;\theta_g)$, discriminator $D(.;\theta_d)$, mesh $x$ of $m$ elements with spacing $d$, initial/boundary condition adjustment $\varphi$, learning rates $\alpha_G, \alpha_D$, Adam moment coefficients $\beta_{G1}, \beta_{G2}, \beta_{D1}, \beta_{D2}$
**for** $i = 1$ **to** $N$ **do**
    Sample $m$ points $x_s \sim x + \mathcal{N}(0, \frac{d}{3})$
    Forward pass $\hat{\psi} = G(x_s)$
    Adjust for conditions $\hat{\psi}' = \varphi(\hat{\psi})$
    Set $LHS = F(x, \hat{\psi}', \nabla\hat{\psi}', \nabla^2\hat{\psi}')$, $RHS = 0$
    Update generator $\theta_g \leftarrow Adam(\theta_g, \alpha_G, \eta_G, \beta_{G1}, \beta_{G2})$ ($\eta_G$ from Eqn 2.9)
    Update discriminator $\theta_d \leftarrow Adam(\theta_d, \alpha_D, \eta_D, \beta_{D1}, \beta_{D2})$ ($\eta_D$ from Eqn 2.10)
**end for**
Return $G$
```

G and discriminator D using the following gradients (for traditional GANs):

$$\eta_G = \nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(G\left(LHS^{(i)}\right)\right)\right) \tag{2.9}$$

$$\eta_D = \nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(RHS^{(i)}\right)+\log\left(1-D\left(LHS^{(i)}\right)\right)\right] \tag{2.10}$$

We provide a pictorial representation of the setup in Figure 2.1, and detail training 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 with $LHS = RHS$.

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 will be constantly changing as the generator is updated throughout training. Instead, we can fix this by setting $LHS = \ddot{x} + x$ and $RHS = 0$. For the examples in this paper, we move all terms of the differential equation to the *LHS* and set $RHS = 0$.[*]

2.5 Experiments

We run experiments on several differential equations of increasing complexity, comparing DEQ-GAN to the classical unsupervised neural network method using squared L_2 (i.e. mean squared error)[†], L_1, and Huber[13] loss functions. We compute the mean squared errors of the solutions produced by each method by using known solutions obtained either analytically or through standard numerical methods. We do not compare to classic numerical methods as others have substantively performed this comparison[19,21,8]. We note that deterministic differential equations do not have any aleatoric uncertainty, and given a high-enough capacity neural network, we could in principle obtain arbitrarily low errors. The reason we do not observe this is because we use stochastic gradient descent and randomly perturb a finite set of points on the domain which we find, in practice, improves the training convergence rate and solution accuracy.

[*] 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 vs. the a priori known ground truth solution.

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[16], 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 results of non-GAN training and the hyperparameters we use are provided in Appendix A.

2.5.1 Exponential Decay

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

$$\dot{x} + x = 0, \tag{2.11}$$

with initial condition $x(0) = 0$. The ground truth solution $x(t) = e^{-t}$ can be obtained analytically through integration.

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 2.2 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 around ~ 0.7. The mean squared error fluctuates for both the training and validation batches throughout training, yet converges to a highly accurate solution around $\sim 10^{-11}$. Visually the prediction of the generator $\hat{x}$ accurately mimics the analytic solution, and the residuals of the equation $\hat{F}$ are small throughout the domain.

To compare against the classical method, in Figure 2.3 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

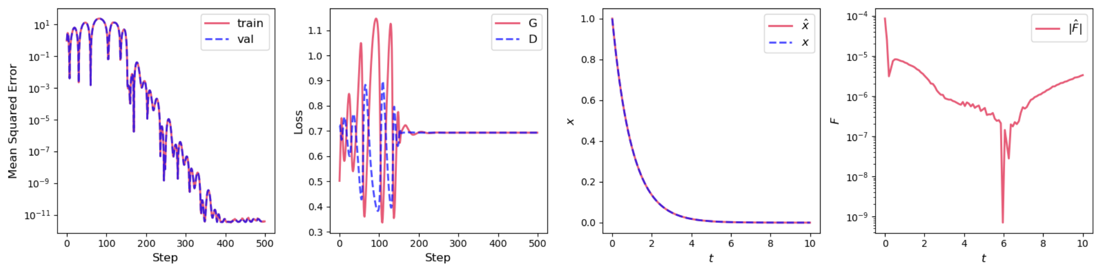


Figure 2.2: Visualization of DEQGAN training for the exponential decay problem. The left-most figure plots the mean squared error vs. iteration. To the right, we plot the value of the generator (G) and discriminator (D) losses at each iteration. Right of this we plot the prediction of the generator $\hat{x}$ 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 $\hat{F}$.

$(2.5, 97.5)$ percentile interval of the mean squared error of the solution at each iteration. We see that for the given choice of hyperparameters[‡], DEQGAN reaches orders of magnitude lower mean squared error. We observe that the variance of the solution accuracy is, however, notably larger for DEQGAN than the classical method across all loss functions. Nevertheless, this figure demonstrates that it is possible to 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

$$\ddot{x} + x = 0, \tag{2.12}$$

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.

[‡]Note that the results are trained using the best hyperparameters found for DEQGAN using a random search procedure.

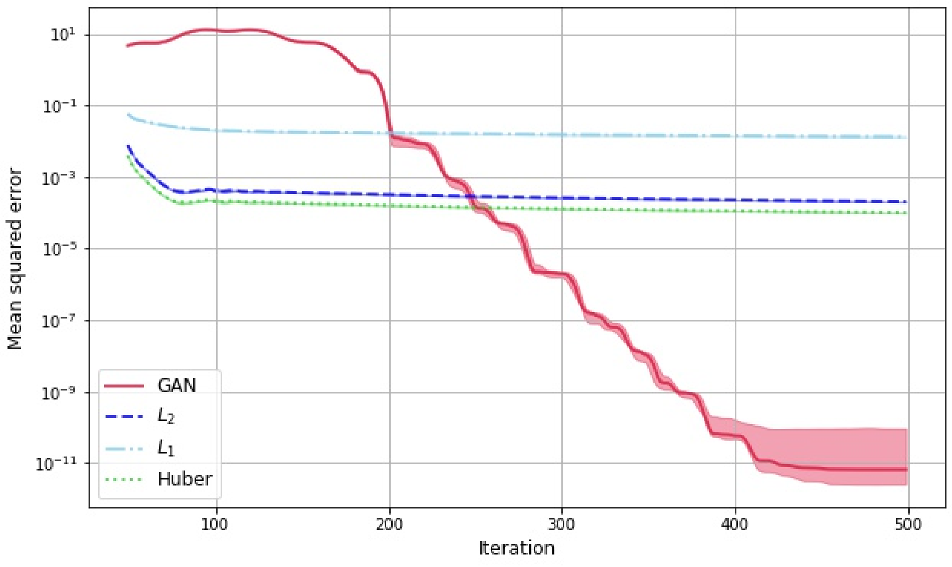


Figure 2.3: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the exponential decay equation. We perform ten randomized trials and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

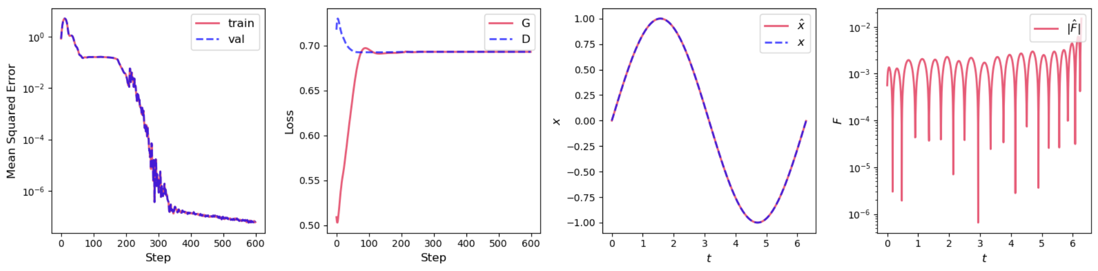


Figure 2.4: Visualization of DEQGAN training for the simple harmonic oscillator problem. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the prediction 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 $\hat{F}$.

Figure 2.4 plots the results of training DEQGAN on this problem. In this case, we observe that the generator and discriminator losses quickly and (almost) monotonically converge to a stable equilibrium. We see the mean squared error decreasing rapidly then plateauing and approaching $\sim 10^{-7}$. The predicted solution $\hat{x}$ is indistinguishable from the true solution, and the residuals of the predicted solution are small across the domain. Interestingly, it appears that the residuals increase slightly near the right edge of the domain, $t \sim 2\pi$.

Comparing to the classical method, in Figure 2.5 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 $\sim 10^{-7}$, while the best of the classical methods reaches only $\sim 10^{-3}$. We note that our DEQGAN exhibits higher variance in solution accuracy, but nevertheless handily outperforms the competitors.

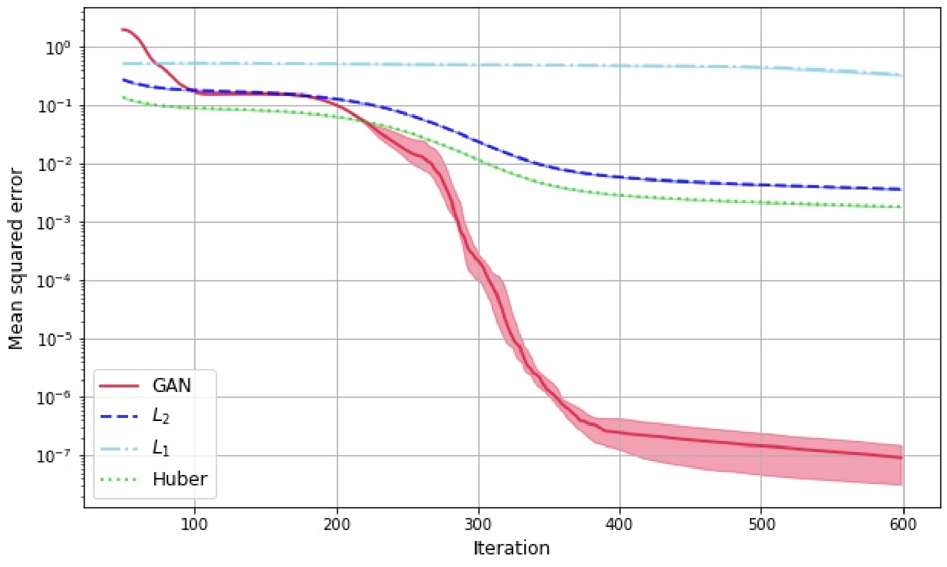


Figure 2.5: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the simple harmonic oscillator equation. We perform ten randomized trials and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size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

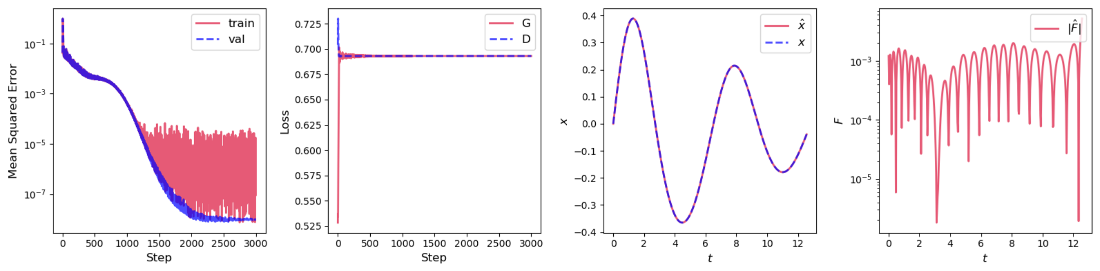


Figure 2.6: Visualization of DEQGAN training for the nonlinear oscillator problem. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the prediction 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

$$\ddot{x} + 2\beta\dot{x} + \omega^2 x + \phi x^2 + \varepsilon x^3 = 0, \tag{2.13}$$

with $\beta = 0.1$, $\omega = 1$, $\phi = 1$, $\varepsilon = 0.1$ 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 our ground truth solutions.

As before, we proceed by setting $LHS = \ddot{x} + 2\beta\dot{x} + \omega^2 x + \phi x^2 + \varepsilon x^3 = 0$ 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 2.6 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's mean squared error decreases to $\sim 10^{-8}$. We note that the mean squared

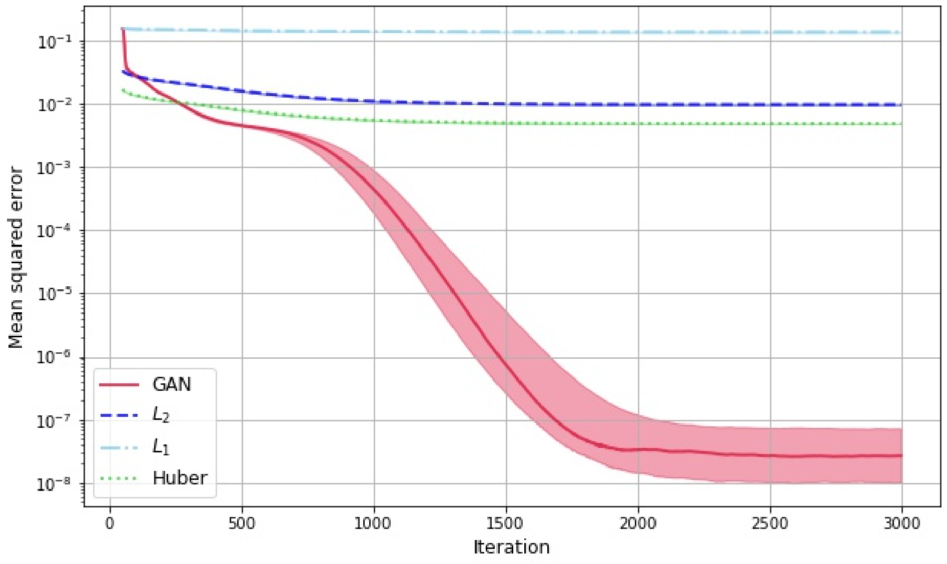


Figure 2.7: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the nonlinear oscillator equation. We perform ten randomized trials and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

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 low throughout the domain (again, increasing slightly near the right edge).

To compare against the classical unsupervised neural network method, Figure 2.7 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 method with L_1, L_2, and Huber loss functions. DEQGAN again outperforms the classical method by orders of magnitude in terms of accuracy, 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.

2.5.4 SIR Epidemiological Model

Given the recent outbreak and pandemic of novel coronavirus (COVID-19)[39], we turn our consideration to 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

$$\frac{dS}{dt} = -\beta\frac{IS}{N} \tag{2.14}$$

$$\frac{dI}{dt} = \beta\frac{IS}{N} - \gamma I \tag{2.15}$$

$$\frac{dR}{dt} = \gamma I \tag{2.16}$$

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 we impose the initial conditions $S_0 = 0.99, R_0 = 0, I_0 = 0.01$.

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

$$LHS = \left[\frac{dS}{dt} + \beta\frac{IS}{N}, \frac{dI}{dt} - \beta\frac{IS}{N} + \gamma I, \frac{dR}{dt} - \gamma I\right]^T \tag{2.17}$$

and $RHS = [0, 0, 0]^T$. Then instead of a scalar the generator outputs the 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.

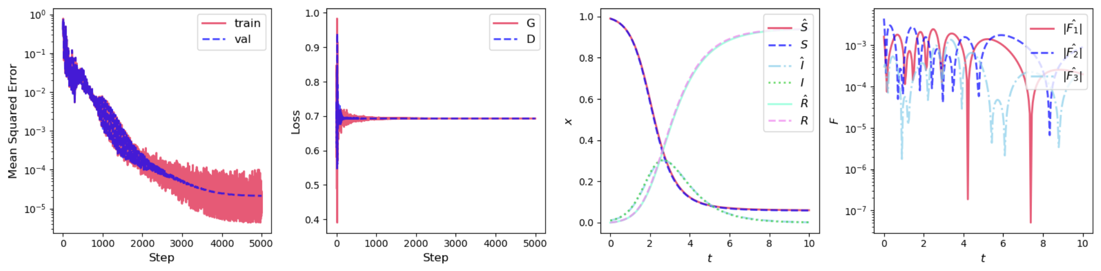


Figure 2.8: Visualization of DEQGAN training for the SIR system of equations. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the predictions of 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 for each equation j, $\hat{F}_j$.

We present the results of training DEQGAN to solve this system of differential equations in Figure 2.8. We observe the generator and discriminator losses initially fluctuating substantially, but quickly reaching a stable equilibrium around ~ 0.7. We note 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 methods, we perform ten randomized experiments where we change the random seed used for sampling mini-batches. Figure 2.9 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

[§]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.

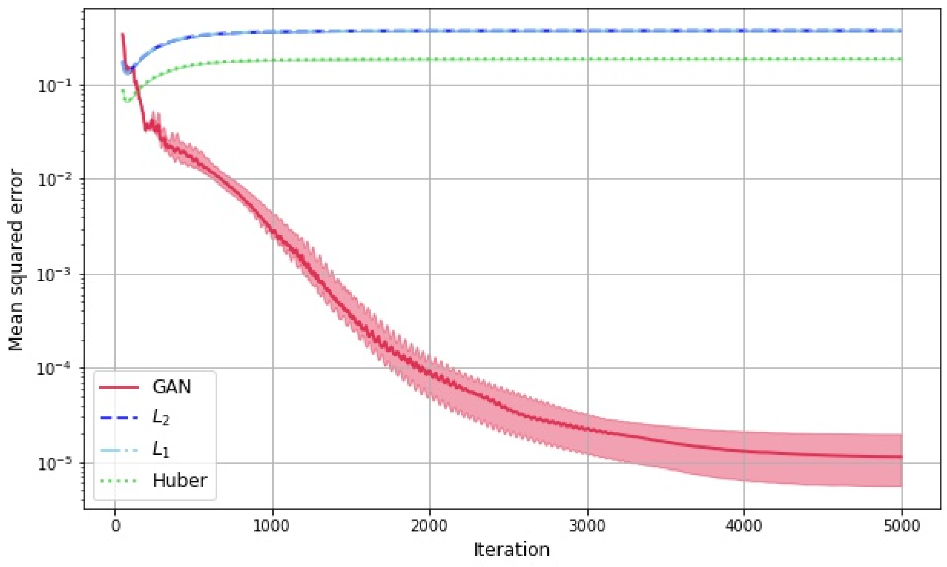


Figure 2.9: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the SIR system of equations. We perform ten randomized trials and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

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.

3 Discussion

3.1 Instability

A point that we have not yet explicitly addressed is the apparent instability of the GAN 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 towards improving the stability and convergence of GANs [1,7,3,28]. In our experiments, we find that the specific initialization of the generator and

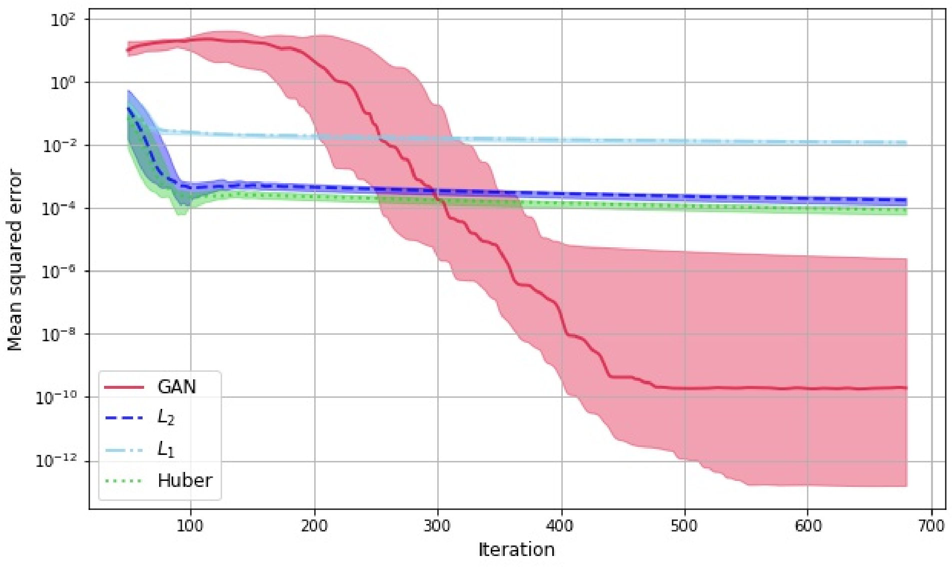


Figure 3.1: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the exponential decay equation. We perform ten randomized trials, with model initialization included in the randomization, and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

discriminator 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 initialization of the models as was done in Figure 2.3. We observe a large increase in the variability of solution accuracy, $\sim$6 orders of magnitude in the $(2.5, 97.5)$ percentile interval, at iteration 500 vs. 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, as in Figure 3.2 for the simple oscillator equation where we plot the mean squared error for DEQGAN and the classical methods, as in Figure 2.5, but including model weight initialization in the randomization of each trial. We observe a significant increase in variability of solution accuracy for all methods, but here a $\sim$3 order of magnitude increase in the solution accuracy range for DEQGAN diminishes its relative performance further compared

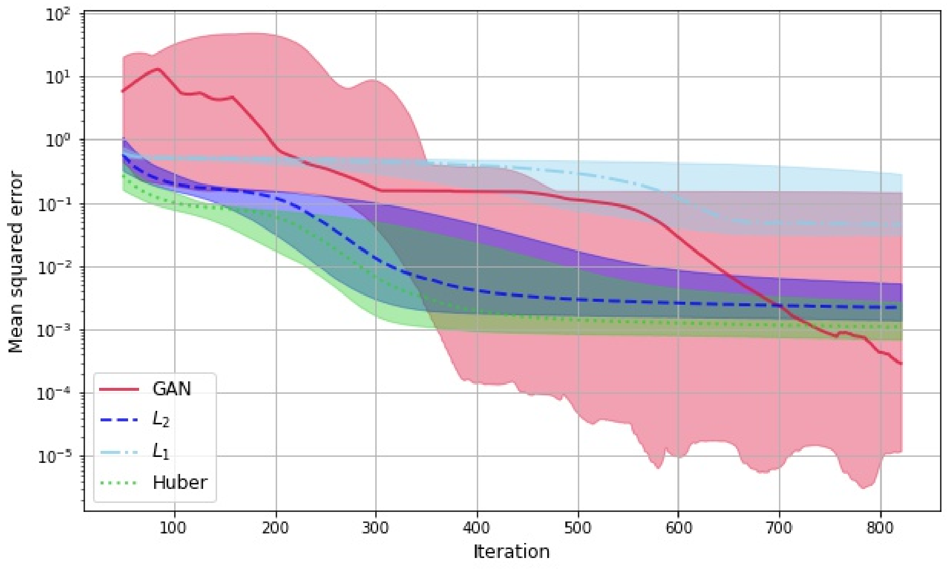


Figure 3.2: Mean squared errors vs. iteration for DEQGAN, L_1, L_2, and Huber loss for the simple oscillator equation. We perform ten randomized trials, with model initialization included in the randomization, and plot the median (bold) and $(2.5, 97.5)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

to the classical neural network methods.

Our solution to this problem is to fix the model weight initializations when tuning hyperparamters for DEQGAN, and to keep these same initializations going forward. If we think of the generator and discriminator initializations as the starting points in the non-convex, min-max optimization problem, then informally we conjecture that the set of hyperparameters which perform well in this starting configuration may not perform as well for other configurations. Maintaining a consistent starting point relative to the hyperparameters, even with randomization of mini-batches, appears to significantly reduce this problem of instability.

Given that non-convex, min-max optimization problems largely lack formal guarantees of convergence in terms of reaching both local and global Nash equilibria, we settle with this solution for now and look forward to further developments in min-max optimization that enable GANs to be trained with greater stability and 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 that everything else is fake. When *RHS* varies with the generator, the real data are no longer fixed, and the GAN training becomes exceedingly unstable, beyond the point of being capable of convergence beyond the exponential decay equation. The solution we adopt is to simply move all terms to the *LHS* and set $RHS = 0$.

3.2.2 Semi-Supervised

In an attempt to counteract the apparent 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. On top of 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 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.

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 discriminator 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 to 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[7] (WGAN-GP, see Section 2.3) to combat optimization difficulties of the vanilla GAN. We found that this clearly enhanced our results and improved the stability of training. 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 DE-QGAN training.

4 Conclusion

Inspired by the line of research developing methods for solving differential equations with unsupervised neural networks, we have presented a new method based on GANs which leverages adversarial training to learn a loss function. We have empirically shown that our method can, often dramatically, outperform the classical unsupervised neural network method with (squared) L_2, L_1 and Huber loss functions. Additionally, we have presented our simple pertubation-based point sampling approach which reduces overfitting and increases solution accuracy.

We are excited by the many promising benefits of learning differentiable solutions to differential equations with unsupervised neural networks. Among the various promising directions being taken in the literature, from deep learning's advantage in high-dimensional settings[33,31,8] to its superiority in obeying physical laws such as energy conservation[27], we hope that our work may help advance the understanding of the importance of loss functions and sampling in tackling this problem, and may benefit future research and applications leveraging neural networks to solve differential equations.

A

A.1 Hyperparameters

We performed random search to tune the hyperparameters of the DEQGAN method for each differential equation. We are grateful for the creators of the Ray-Tune package[25] 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$(B_1 = 0.9, B_2 = 0.999)$
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.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

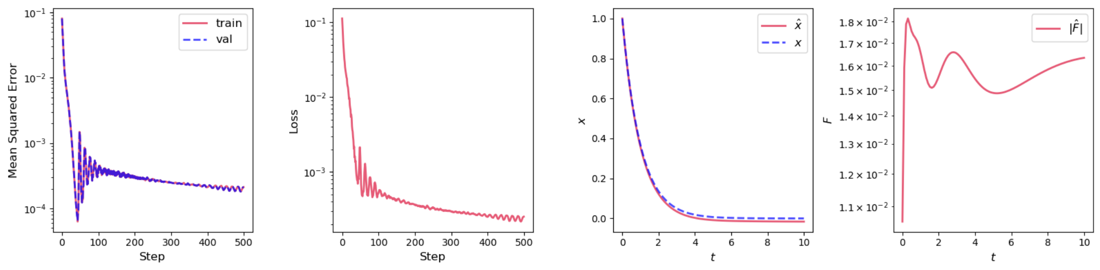
Hyperparameter	Value
Num. Iterations	3000
Num. Grid Points	400
Sampling Method	Perturb
Grid Boundary	$(0, 4\pi)$
G Units	20
G Layers	2
D Units	20
D Layers	2
G Learning Rate	0.005801628839417561
D Learning Rate	0.0007291873762250555
G Optimizer	Adam($B_1 = 0.10244627, B_2 = 0.76328835$)
D Optimizer	Adam($B_1 = 0.54142685, B_2 = 0.67750577$)
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

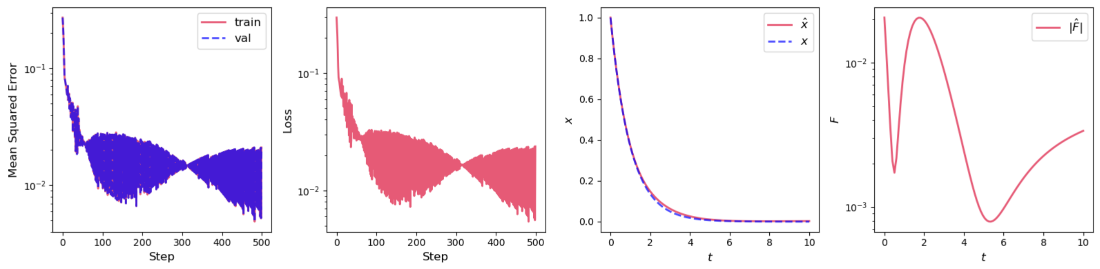
Hyperparameter	Value
Num. Iterations	5000
Num. Grid Points	200
Sampling Method	Perturb
Grid Boundary	$(0, 10)$
G Units	20
G Layers	2
D Units	20
D Layers	2
G Learning Rate	0.006429803531841584
D Learning Rate	0.009647103812410594
G Optimizer	Adam($B_1 = 0.14666949$, $B_2 = 0.93261048$)
D Optimizer	Adam($B_1 = 0.51750004$, $B_2 = 0.85405624$)
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 Results

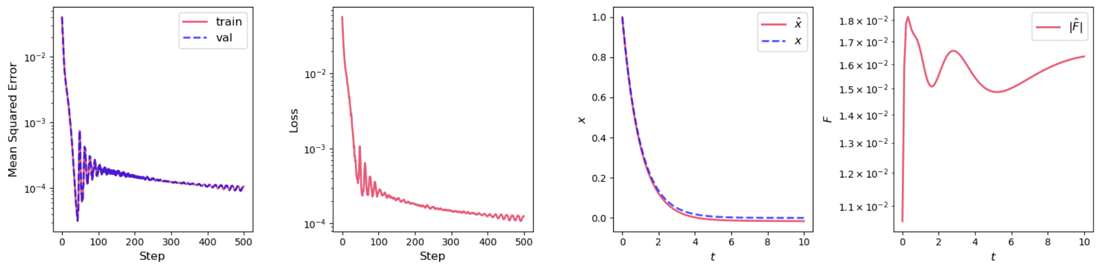
Here we present the training visualizations for each of the non-GAN-based methods (L_2, L_1, Huber) which we compared with in Chapter 2.



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

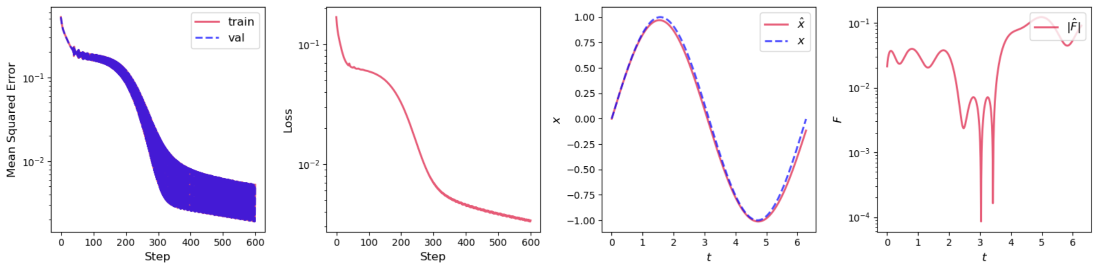


(b) L_1 loss.

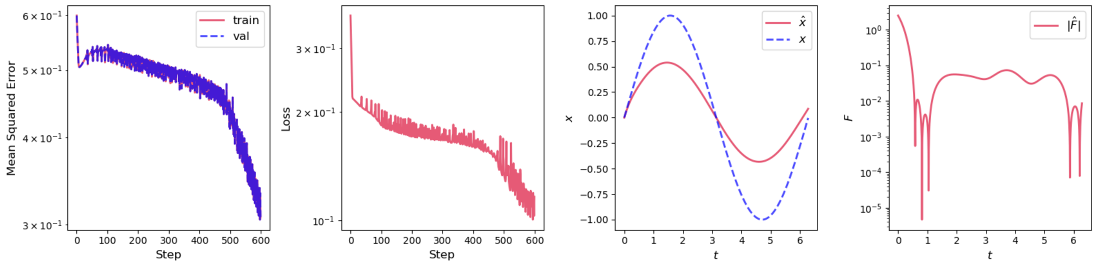


(c) Huber loss.

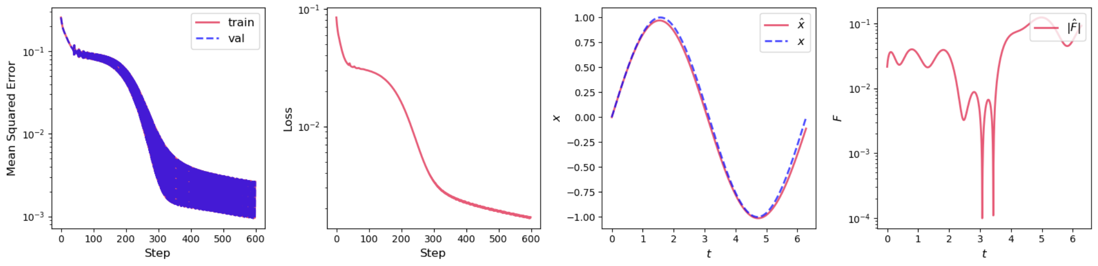
Figure A.1: Visualization of fully-connected neural networks trained with various losses to solve the exponential decay differential equation. The left-most figures plot the mean squared error vs. step (iteration) count. To the right of this, we plot the unsupervised loss for each step. Right of this 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.

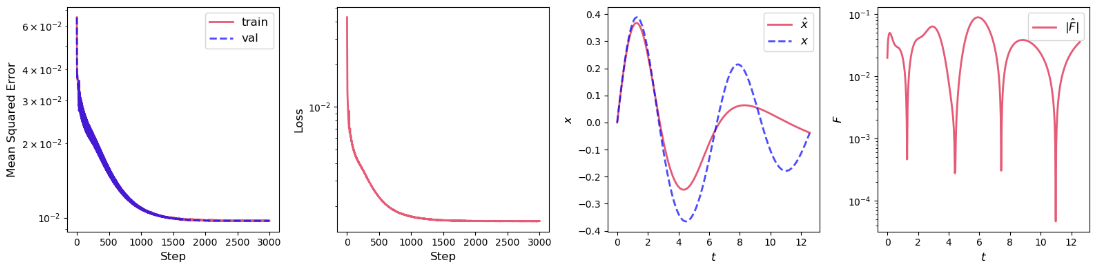


(b) L_1 loss.

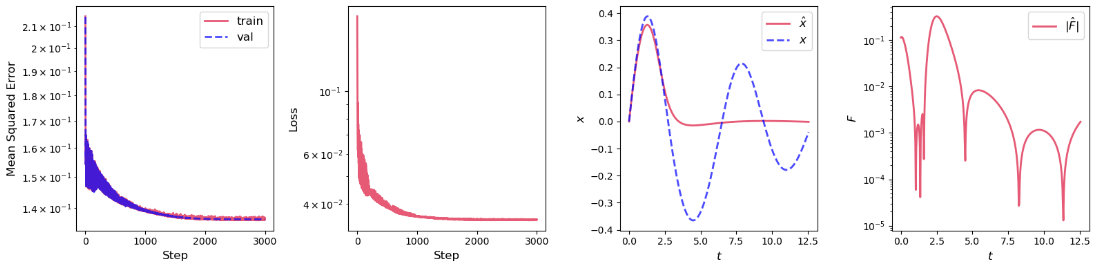


(c) Huber loss.

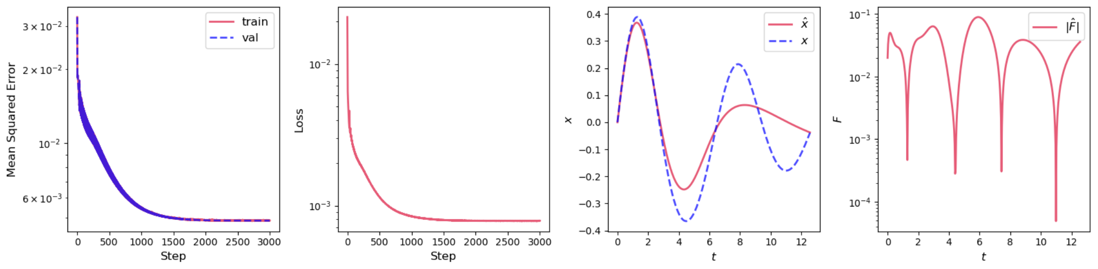
Figure A.2: Visualization of fully-connected neural networks trained with various losses to solve the simple oscillator differential equation. The left-most figures plot the mean squared error vs. step (iteration) count. To the right of this, we plot the unsupervised loss for each step. Right of this 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.

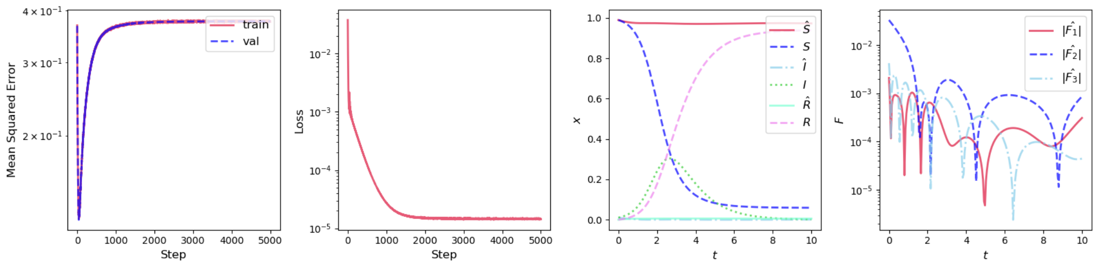


(b) L_1 loss.

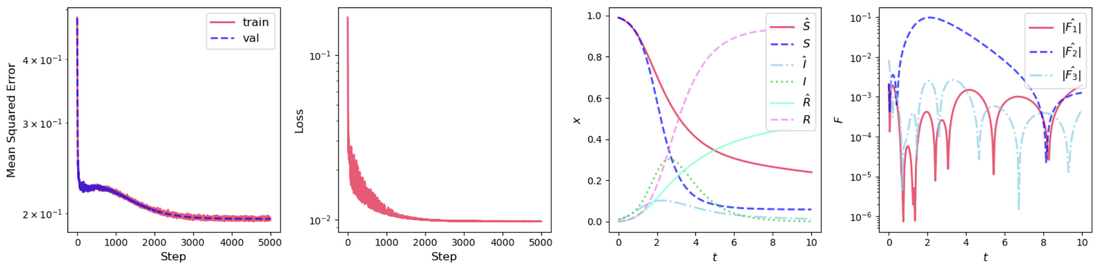


(c) Huber loss.

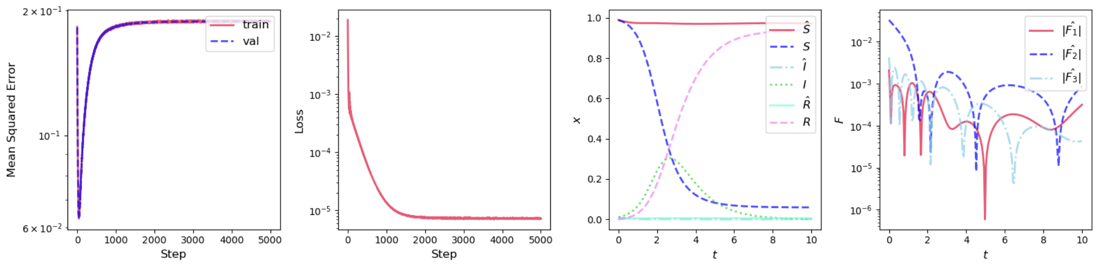
Figure A.3: Visualization of fully-connected neural networks trained with various losses to solve the nonlinear oscillator differential equation. The left-most figures plot the mean squared error vs. step (iteration) count. To the right of this, we plot the unsupervised loss for each step. Right of this 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 model 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 $\hat{F}$.

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

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

[6] 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.

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

[8] 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.

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

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

[11] 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.

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

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

[14] 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).

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

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

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

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

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

[30] 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*.

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

[32] 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.

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

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

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

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

[37] 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.

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

[39] 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.

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

[41] 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.