Generative Adversarial Network Methods for Solving Differential Equations

A DISSERTATION PRESENTED
BY
J. BLAKE BULLWINKEL
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 2022

Thesis advisor: Professor Pavlos Protopapas J. Blake Bullwinkel

Generative Adversarial Network Methods for Solving Differential Equations

Abstract

Differential equations have wide-ranging applications in science and engineering but can rarely be solved exactly. There is growing interest in approximating the solutions to these equations using unsupervised neural networks, which provide closed-form, differentiable solutions. While most neural network solvers use simple feedforward architectures, a method called Differential Equation GAN (DEQGAN), which uses generative adversarial networks to solve differential equations in a fully unsupervised manner, has been shown to outperform classical neural networks in terms of predictive accuracy. However, DEQGAN is comparatively unstable to train and sensitive to hyperparameters.

In this thesis, we explore a variety of approaches to improve the robustness and utility of DEQGAN. First, we adaptively improve its training convergence by adding instance noise based on the generator and discriminator losses. Second, we utilize the equation residuals to detect and terminate poor-performing runs early. Third, we show how these techniques enable DEQGAN to obtain reliable results that outperform those obtained by classical unsupervised neural networks on a range of challenging PDEs and systems of ODEs, including the non-linear Burgers', Allen-Cahn, Hamilton, and modified Einstein's gravity equations. Finally, we propose a multi-head DEQGAN architecture and demonstrate that transfer learning can be applied to discover solutions to multiple initial conditions at once or to different parameterizations of a given system more efficiently. We find that by leveraging a pre-trained base generator, transfer learning greatly stabilizes DEQGAN training and can improve solution accuracy.*

*All code is publicly available at https://github.com/dylanrandle/denn.

Contents

Listing of figures

To my parents.

Acknowledgments

First, I would like to thank Dr. Pavlos Protopapas, who took me under his wing when I was still learning the fundamentals of machine learning and has been a patient and supportive advisor ever since. Over the past three semesters, I have also received much guidance from Dr. David Sondak and Dylan Randle, whose own research at IACS inspired mine and forms the basis of the work presented in this thesis.

I would also like to thank Dr. Marios Mattheakis and Raphael Pellegrin, who welcomed me as a research collaborator and introduced me to branched flows and multi-head neural network architectures. Within StellarDNN more broadly, I have found a group of exceptionally motivated and kind people who offer a wealth of ideas and expertise. In particular, I am grateful to Elaine Cunha for many fruitful conversations in the fall semester, Shuheng Liu for answering my questions about re-parameterizations and PyTorch, Augusto Chantada for teaching me about modified gravity theories, and Olga Graf for sharing code on Burgers' equation. Outside of the research lab, I have also benefitted a great deal from Kevin Howarth, who taught me about numerical methods for solving PDEs, and from Harvard's FAS Research Computing team. I would also like to thank Dr. Weiwei Pan, who has imparted much wisdom about the research process and helped me understand how the things I have learned over the past two years fit into society at large.

Finally, I would like to thank my family – Jeff Bullwinkel, Kimberly Latham, and Avery Bullwinkel – and Lauren Heuer for their endless love and support.

Carbon Impact

The recent success of deep learning can largely be attributed to the advent of big data and big compute. The combination of these two resources has given rise to powerful AI models like OpenAI's GPT-3, which can, for example, generate incredibly convincing human-like text.[6] Unfortunately, most of these models demand significant computational resources for training, drawing massive amounts of electricity and creating a large carbon impact.

While the models trained in this thesis come nowhere close to the scale of GPT-3, our experiments required expensive hyperparameter tuning jobs and were run on Harvard's FAS Research Computing cluster. More specifically, this research utilized a total of 13,272 hours of compute performed primarily on Intel Cascade Lake CPU cores.

Lacoste et al.[28] created an ML Emissions Calculator that shows how this information can be combined with other factors to estimate carbon footprint in kilograms of CO_2e, or carbon dioxide equivalent, which is a standardized unit of global warming potential. According to ElectricityMap, the carbon efficiency of the electrical grid in the New England region is $0.275 kgCO_2eq/kWh$. Further, the CPUs utilized have TDP ranging between 85W and 150W.

Plugging this information into the ML Emissions Calculator, we estimate the total carbon footprint of this thesis to be **$310–547 kgCO_2e$**, which is equivalent to driving around 1,000 miles in a car or eating about 100 cheeseburgers.

0

Introduction

Over the past decade, deep learning has achieved major successes on a wide range of tasks. From explaining jokes, to writing code, [11] to beating the world champion in Go, [49] deep neural networks have equipped computers with abilities that were previously thought impossible and are even associated with human-like intelligence.

Aside from these exciting capabilities, deep learning can also be an incredibly useful tool in fields such as scientific computing and mathematical physics. In recent years, a wave of research has applied neural networks to solving differential equations, which describe the dynamics of natural phenomena that occur on practically every scale in the universe and have many applications in science and engineering. Fields such as physics, chemistry, biology, and economics use differential equations to model complex phenomena and rely on computational methods to approximate their solutions efficiently and accurately.

Although traditional numerical methods for solving differential equations perform well and the theory of their stability and convergence is well established, neural networks offer an array of compelling advantages, including that they provide solutions that are closed-form, [29] suffer less from the "curse of dimensionality," [21,43,50,17] do not propagate numerical errors, [38] provide a more accurate interpolation scheme, [29] and can leverage transfer learning for the fast discovery of new solutions. [14,12] Further, neural networks do not require an underlying grid and offer a meshless approach to solving differential equations. This makes it possible to use trained neural networks, which typically have small memory footprints, to generate solutions over arbitrary grids in a single forward pass.

A variety of neural network methods for solving differential equations has emerged. Some of these are supervised and learn the dynamics of real-world systems from data. [44,10,16,4] Others are semi-supervised, learning general solutions to a differential equation and extracting a best fit solution based on observational data. [41] The third category, which we explore in this thesis, is *unsupervised* neural networks, which are trained in a data-free manner that depends solely on the network's predictions and the equation itself. [44,20,42,37,52,38,21,43,50]

Most unsupervised neural networks use simple feed-forward architectures and require the specification of a loss function over the equation residuals. Motivated by the lack of justification for a particular choice of loss function, Randle et al. [45] proposed Differential Equation GAN (DEQGAN), which uses generative adversarial networks to "learn the loss function" for optimizing the network. While DEQGAN can obtain solution accuracies that outperform classical neural networks by multiple orders of magnitude, it is comparatively unstable to train and sensitive to hyperparameters. These issues are common among GAN training algorithms, and we draw upon recent developments in the GAN literature to address them in the context of DEQGAN.

In this thesis, we explore several approaches for improving the robustness and utility of DEQGAN. In Chapter 1, we elaborate on how differential equations can be solved using unsupervised neural networks that use feed-forward and GAN-based architectures. In Chapter 2, we introduce techniques to improve the convergence of DEQGAN and detect poor-performing runs early. Chapter 3 presents results on a range of ordinary differential equations (ODEs) and partial differential equations (PDEs), comparing DEQGAN with popular numerical methods and classical neural networks that use L_2, L_1 and Huber loss functions. In Chapter 4, we propose a multi-head DEQGAN architecture and demonstrate how transfer learning can be leveraged to more efficiently obtain solutions to multiple initial conditions or different parameterizations of a given problem. We end with a discussion of the limitations of our approach and suggestions for future work.

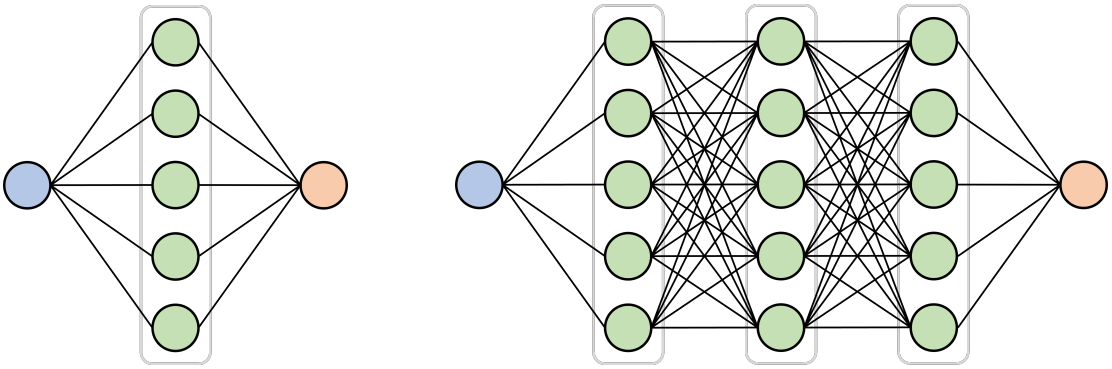
Nearly everything is interesting if you look into it deeply enough.

Richard Feynman

1

Unsupervised Learning of Solutions to Differential Equations

Neural networks are highly expressive models that have attracted a great deal of attention in recent years. Loosely based on the structure of the brain, they input data through a network



(a) Single layer network **(b)** Network with three layers

Figure 1.1: Two different neural network architectures. The blue and orange dots represent inputs and outputs to the network, respectively, while the green represent "hidden nodes" that form layers. Both of these networks are universal function approximators.

of interconnected nodes to obtain a desired output. A single layer of a neural network corresponds to a subset of these nodes and performs a simple series of operations:

1. Linear transformation by a weight matrix W
2. Translation by a vector b
3. Application of a non-linear activation f

Therefore, a neural network with only one layer, as pictured in Figure 1.1a, transforms an input vector x to produce the output y as follows.

$$y = f(Wx + b) \tag{1.1}$$

Hornik et al. [22] showed that even this single-layer neural network is a universal approximator, given a sufficient number of nodes in the layer. Figure 1.1b shows an example of a neural network that uses three layers, which is also a universal approximator. In general, so-called "deep" neural

networks combine many layers in this hierarchical fashion, giving rise to both increased expressivity and the capacity to learn latent representations of the inputs.

These properties are very useful for predicting and modeling complex physical phenomena and have inspired researchers to apply deep learning to a vast array of problems in mathematical physics. However, neural networks also have major drawbacks. For example, they learn purely from correlations in data and may violate known laws of physics if those laws are not fully represented in the data. Further, neural networks usually consist of so many parameters and apply such complex transformations of the input data that it can be very difficult to determine how they actually make predictions. For these reasons, neural networks are often referred to as uninterpretable "black-box" models.

These limitations have given rise to a growing interest in developing neural networks that incorporate known physics. Because much of physics is expressed in the language of differential equations, one approach is to train neural networks to learn solutions to ordinary (ODEs) and partial differential equations (PDEs). In the following sections, we elaborate on how this can be achieved without access to ground-truth training data – that is, in an unsupervised fashion.

1.1 Feed-Forward Neural Networks

Solving differential equations with unsupervised neural networks was originally proposed by Dissanayake & Phan-Thien [13], who considered equations of the form

$$F(t, u(t), u'(t), u''(t), \dots) = 0 \tag{1.2}$$

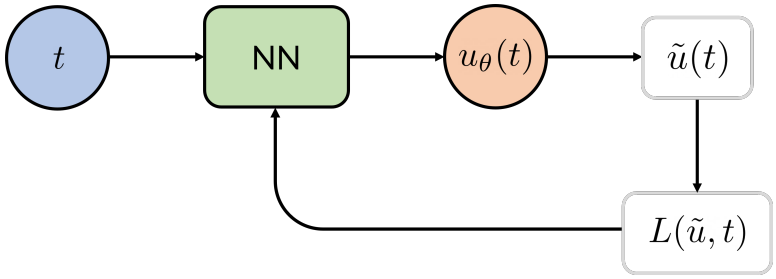


Figure 1.2: Classical method for solving differential equations with unsupervised neural networks. We provide an input t to the network and obtain $u_\theta(t)$, which we re-parameterize so that it exactly satisfies any initial or boundary conditions. This gives our proposed solution $\tilde{u}(t)$, with which we calculate a pre-specified loss $L(\tilde{u}, t)$ that the network is trained to minimize.

where $u(t)$ is the solution of the equation and $u'(t), u''(t), \ldots$ are the first, second, and higher order derivatives of the solution. As illustrated in Figure 1.2, we denote t as the input to the neural network and $u_\theta(t)$ as the corresponding output. Plugging $u_\theta(t)$ into the left-hand side of Equation 1.2 gives the equation *residuals*, which will be equal to zero when $u_\theta(t) = u(t)$. Therefore, we train the network to minimize the loss

$$L(u_\theta, t) = \sum_{t \in \mathcal{D}} F(t, u_\theta(t), u'_\theta(t), u''_\theta(t), \ldots)^2 \tag{1.3}$$

where $\mathcal{D}$ specifies the domain of the equation. This formalism can be generalized to handle spatial dimensions and multidimensional problems simply be passing multiple inputs to the network, which we do in our experiments and describe further in Chapter 3.

During training, it is common to sample $t \in \mathcal{D}$ from a fixed or noisy grid. In our experiments, we primarily adopt the latter approach, which Randle et al. [45] found reduces overfitting. More specifically, we perturb each point in an evenly spaced grid by adding zero-centered Gaussian noise with

standard deviation $\Delta t/3$, where Δt is the spacing between points.

Equation 1.3 gives a good indication of how close the output of the neural network is to the true solution, but ignores the initial condition required to specify the problem. Therefore, it is common to re-parameterize the neural network outputs so that they exactly satisfy this condition, as suggested by Mattheakis et al. [38]. For initial value problems with $u(t)\big|_{t=t_0} = u_0$, we can use

$$\tilde{u}(t) = u_0 + \left(1 - e^{-(t-t_0)}\right) u_\theta(t) \tag{1.4}$$

which forces the predicted solution to exactly satisfy the initial condition at $t = t_0$ and exponentially decays this constraint for $t > t_0$. Chen et al. [9] have implemented re-parameterizations to handle a range of other conditions required to specify different types of ODEs and PDEs, such as Neumann and Dirichlet boundary conditions. In Chapter 3, we elaborate on several of these re-parameterizations in relation to the particular equations we study in this thesis.

Plugging Equation 1.4 into Equation 1.3, therefore, the loss function becomes

$$L(\tilde{u}, t) = \sum_{t \in \mathcal{D}} F(t, \tilde{u}(t), \tilde{u}'(t), \tilde{u}''(t), \dots)^2 \tag{1.5}$$

Equation 1.5 is also known as the sum of squared residuals and can be iteratively optimized using backpropagation. Because neural networks provide closed-form, differentiable outputs, we can use automatic differentiation to compute $\tilde{u}'(t)$, $\tilde{u}''(t)$, and any higher order derivatives in the differential equation.

1.2 Generative Adversarial Networks

The basic approach described above has been shown to work well on a wide array of ODEs[29,14,38,36] and PDEs.[21,50,43,52] However, there is no theoretical justification for the use of any particular loss function. Equation 1.5 penalizes the squared L_2 of the equation residuals, but it is not clear why this would be preferred over L_1, Huber or any other loss function.

The maximum likelihood principle provides a clear reason to fit data that follow Gaussian and Laplace noise models with L_2 and L_1 loss functions, respectively. Given that differential equations are deterministic and our approach to solving them is entirely unsupervised, we lack an equivalent justification.

In light of this, Randle et al.[45] proposed Differential Equation GAN (DEQGAN) for solving differential equations in a fully unsupervised manner using generative adversarial networks.[15] The discriminator network of a GAN can be thought of as "learning the loss function" to train the generator, thereby eliminating the need for a pre-specified loss function. Recently, Zeng et al.[55] proposed a piecewise continuous loss function that generalizes several common losses, including L_2 and L_1. By introducing a trainable parameter, this method adaptively modifies the loss throughout training. DEQGAN, however, uses a dedicated neural network (the discriminator) to learn the loss function and is even less restrictive than an adaptive loss. Beyond the context of differential equations, it has also been shown that where classical loss functions struggle to capture complex spatio-temporal dependencies, GANs may be an effective alternative.[30,31,26]

GANs are a type of generative model that uses two neural networks, called the generator and the

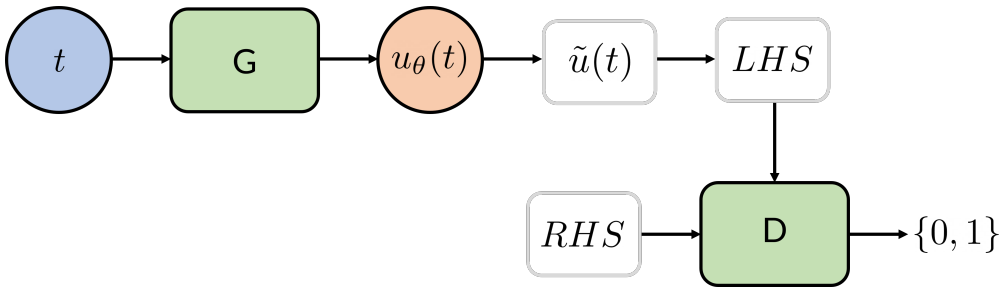


Figure 1.3: Differential Equation GAN (DEQGAN) method for solving differential equations in an unsupervised fashion. We provide an input t to the network and obtain $u_\theta(t)$, which we re-parameterize so that it exactly satisfies any initial or boundary conditions. This gives our proposed solution $\tilde{u}(t)$, which we use to construct the LHS vector of equation residuals. We input "fake" LHS data and "real" RHS data into a discriminator, which is trained to differentiate between the two.

discriminator, to induce a generative distribution $p_{\text{fake}}(x)$ that mimics a target data distribution $p_{\text{data}}(x)$. Informally, this is achieved by training the discriminator to distinguish between "real" data samples $x \sim p_{\text{data}}(x)$ and "fake" samples produced by the generator. In particular, the training procedure optimizes the generator G and discriminator D according to

$$\min_G \max_D V(D, G) = \min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[1 - \log D(G(z))] \tag{1.6}$$

where $z \sim \mathcal{N}(0, 1)$ denote samples from the latent space, which serve as input to the generator. Ideally, optimizing Equation 1.6 results in a discriminator that becomes increasingly accurate at classifying "real" and "fake" samples, and a generator that can produce increasingly realistic "fake" data.

The classical neural network method for solving differential equations can be extended quite naturally to the GAN framework. As depicted in Figure 1.3, the DEQGAN training procedure also

involves passing points from the domain $t \sim \mathcal{D}$ to a network and re-parameterizing its output to obtain a proposed solution $\tilde{u}(t)$. But rather than calculate a loss $L(\tilde{u}, t)$, we use $\tilde{u}(t)$ to construct the "fake" data vector, which we denote LHS, by plugging $\tilde{u}(t)$ into the differential equation.

$$LHS = F\left(t, \tilde{u}(t), \tilde{u}'(t), \tilde{u}''(t), \dots\right) \tag{1.7}$$

Notice that the LHS is simply the residuals of the proposed solution. Because this expression will be equal to zero when $\tilde{u}(t)$ exactly satisfies the differential equation, DEQGAN sets the "real" data vector $RHS = 0$. The effectively fixes the target distribution p_{real} to the Dirac delta function $\delta(0)$.

Letting $RHS^{(i)}$ and $LHS^{(i)}$ be the real and fake samples, respectively, at iteration i, the gradients used to update the weights of the generator and discriminator take the same form as those derived in the original GAN paper. [15]

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(LHS^{(i)}\right)\right), \tag{1.8}$$

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(RHS^{(i)}\right) + \log\left(1 - D\left(LHS^{(i)}\right)\right)\right] \tag{1.9}$$

By training a GAN in this way, DEQGAN learns to generate solutions $\tilde{u}(t)$ such that Equation 1.7 (the LHS vector of equation residuals) is indistinguishable from a vector of zeros, which would imply that $\tilde{u}(t)$ is a good approximate solution to the differential equation.

The key difference between classical unsupervised neural networks and DEQGAN is that the latter frees the network optimization from a pre-specified loss function by leveraging a generator to *automatically* learn one that minimizes the equation residuals. In the sections that follow, we build upon this method and show that it offers major advantages over the classical approach.

Above all, don't fear difficult moments. The best comes from them.

Rita Levi-Montalcini

2

Training Stability

While GANs have achieved state of the art results on a wide range of generative modeling tasks, they are often difficult to train. As a result, much recent work on GANs has been dedicated to improving their sensitivity to hyperparameters, training stability, and convergence. [47,18,51,2,25,27,3,5,39,40]

In our experiments, we found that DEQGAN can be sensitive to the generator and discriminator

weight initializations and learning rates. We recommend two additional techniques for improving the robustness of DEQGAN to different hyperparameter values.

2.1 Instance Noise

Sønderby et al. [51] note that the convergence of GANs relies on the existence of a unique optimal discriminator separating the distribution of "fake" samples p_{fake} produced by the generator, and the distribution of "real" data p_{real}. In practice, however, there may be many near-optimal discriminators that pass very different gradients to the generator, depending on their initialization. Arjovsky & Bottou [2] proved that this problem will arise when there is insufficient overlap between the supports of p_{fake} and p_{real}. As illustrated by Figure 2.1, the original DEQGAN training algorithm fixes $p_{\text{real}} = \delta(0)$, implying that the distribution of "real" data lies in a zero-dimensional manifold. This makes it very unlikely that p_{fake} and p_{real} will share support in a high-dimensional space at the beginning of training.

Recent works have proposed adding "instance noise" to p_{fake} and p_{real} to encourage their overlap, [51,2] which amounts to adding noise to the LHS and RHS, respectively, at each iteration of the DEQGAN training algorithm. Because this makes the discriminator's job more difficult, we add Gaussian noise ε with standard deviation equal to the difference between the generator and discriminator losses, i.e.,

$$\varepsilon = \mathcal{N}(0, \sigma^2), \quad \sigma = \text{ReLU}(L_g - L_d) \tag{2.1}$$

where L_g and L_d are the generator and discriminator losses, respectively. As the generator and dis-

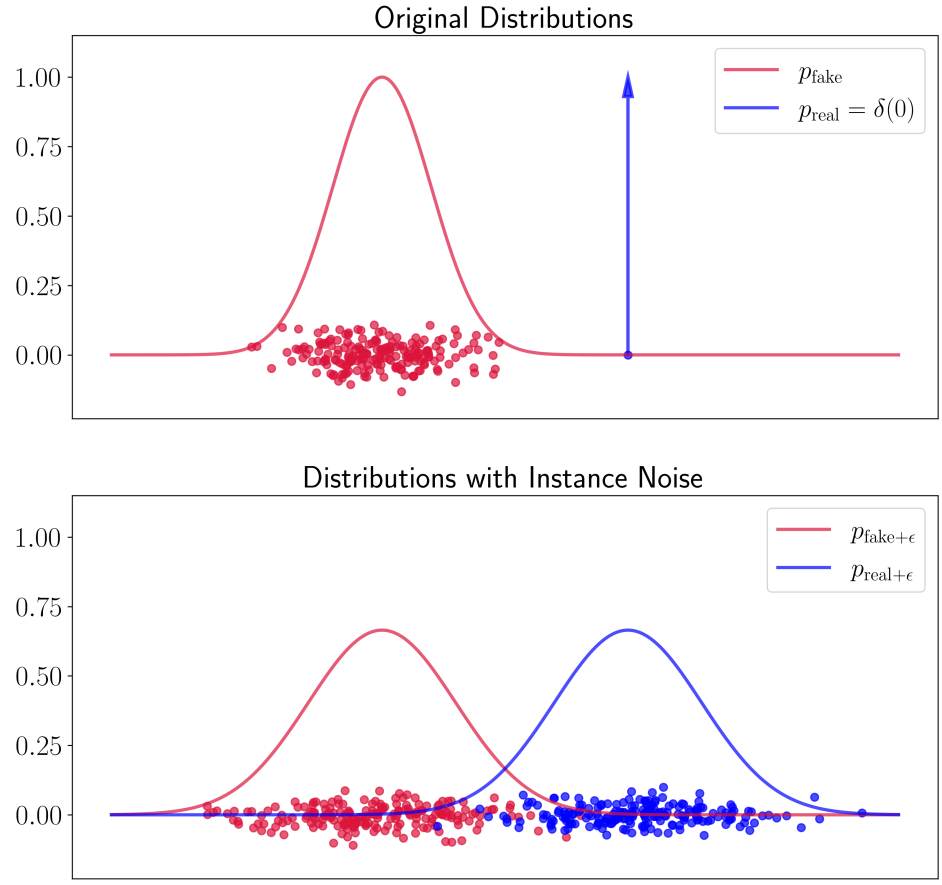


Figure 2.1: Comparison of "real" (blue) and "fake" (red) data distributions in the original DEQGAN formulation (top) and with instance noise added to the LHS and RHS samples (bottom).

criminator reach equilibrium, Equation 2.1 will naturally converge to zero, annealing the amount of noise added. We use the ReLU function because when $L_d > L_g$, the generator is already able to fool the discriminator, suggesting that additional noise is not required. In Section 2.3, we conduct an ablation study and find that this improves the ability of DEQGAN to produce accurate solutions across a range of hyperparameter settings.

2.2 Residual Monitoring

One of the attractive properties of the DEQGAN setup is that the LHS vector of equation residuals gives a direct measure of the quality of the solution at each training iteration. We observe that the LHS tends to oscillate wildly when DEQGAN training becomes unstable but decreases steadily throughout successful training runs. By monitoring the L_1 norm of the LHS in the first 25% of training iterations, we are able to easily detect and terminate bad runs if the variance of these values exceeds some threshold.

Figure 2.2 shows what the L_1 norm of the LHS looks like for runs that ended with high (red) and low (blue) mean squared errors (calculated against the ground truth solution) for three example problems. Because the LHS may oscillate initially even for successful runs, we use a patience window in the first 15% of iterations. In all three equations below, we terminate a run if the variance of the residual L_1 norm calculated over the previous 20 iterations exceeds 0.01.

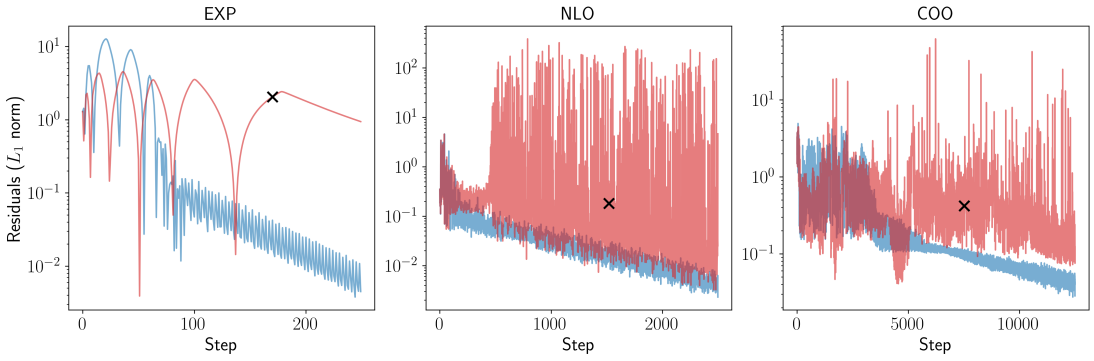


Figure 2.2: Equation residuals in the first 25% of training runs that ended with high (red) and low (blue) mean squared error for the exponential decay (EXP), non-linear oscillator (NLO) and coupled oscillator (COO) problems. The black crosses show the point at which the high MSE runs were terminated early.

2.3 Ablation study

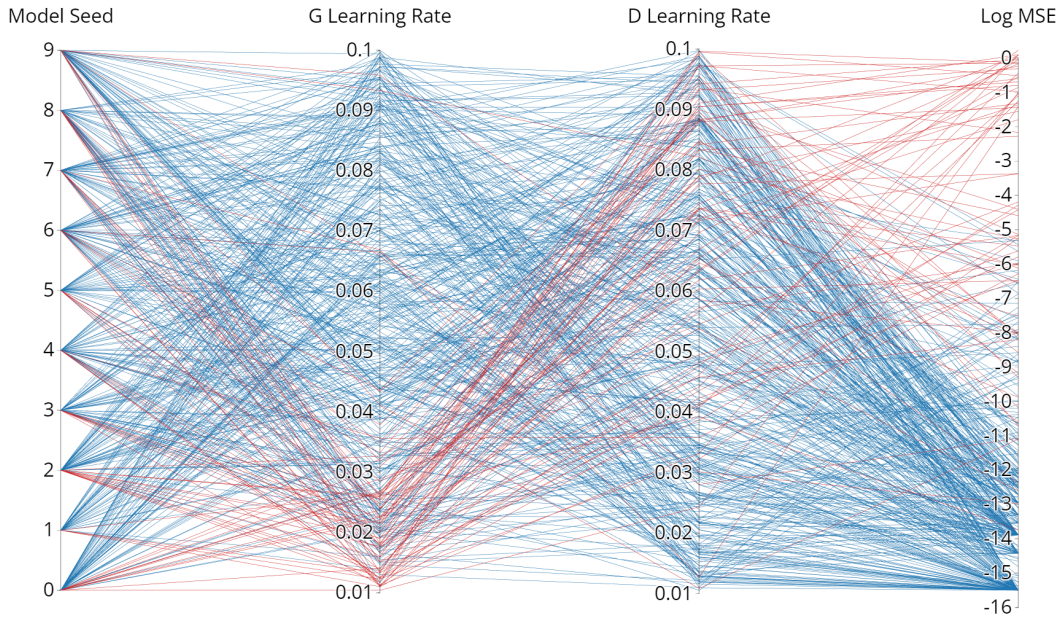


Figure 2.3: Parallel plot showing the results of 500 DEQGAN experiments on the exponential decay equation with instance noise. The red lines represent runs which would be terminated early by monitoring the variance of the equation residuals in the first 25% of training iterations. The mean squared error is plotted on a log_{10} scale.

To quantify the increased robustness offered by instance noise and residual monitoring, we performed an ablation study comparing the percentage of high MSE ($\geq 10^{-5}$) runs obtained by 500 randomized DEQGAN runs for the exponential decay equation with and without using these techniques.

Figure 2.3 plots the results of these 500 DEQGAN experiments with instance noise added. For each experiment, we uniformly selected a random seed controlling model weight initialization as an integer from the range $[0, 9]$, as well as separate learning rates for the discriminator and generator

in the range $[0.01, 0.1]$. We then recorded the final mean squared error after running DEQGAN training for 1000 iterations. The red lines represent runs which would be terminated early by our residual monitoring method, while the blue lines represent those which would be run to completion.

Figure 2.3 shows that the large majority of hyperparameter settings tested with the addition of instance noise resulted in low mean squared errors. Further, residual monitoring was able to detect all runs with MSE $\geq 10^{-5}$ within the first 25% of training iterations. Approximately half of the MSE runs in $[10^{-8}, 10^{-5}]$ would be terminated, while 96% of runs with MSE $\leq 10^{-8}$ would be run to completion.

Table 2.1 summarizes the results of our ablation study, comparing the percentage of high MSE runs with and without instance noise and residual monitoring. We see that adding instance noise decreased the percentage of runs with high MSE and that residual monitoring is highly effective at filtering out poor performing runs. When used together, these techniques eliminated all runs with MSE $\geq 10^{-5}$.

	% Runs with High MSE ($\geq 10^{-5}$)	
	No Residual Monitoring	With Residual Monitoring
No Instance Noise	12.4	0.4
With Instance Noise	8.0	0.0

Table 2.1: Summary of the ablation study comparing the results of 500 DEQGAN experiments on the exponential decay equation (EXP) with and without instance noise. We report the proportion of runs that ended in high MSE solutions and the proportion of runs that would be terminated early using residual monitoring.

The results of our ablation study agree with previous works, which have found that instance

noise can improve the training convergence of other GAN training algorithms[51,2]. Further, residual monitoring could readily be applied to other unsupervised neural network methods for solving differential equations and may be useful beyond the context of DEQGAN.

The miracle of the appropriateness of the language of mathematics for the formulation of the laws of physics is a wonderful gift which we neither understand nor deserve.

Eugene Wigner

3

Differential Equations

In this section, we present experimental results on a range of ordinary and partial differential equations. Because we are interested in how our method differs from classical unsupervised neural networks, for each equation, we compare the performance of DEQGAN to that of networks which use L_2, L_1 and Huber loss functions. In addition, we provide results obtained by popular numeri-

cal approaches, including the fourth-order Runge Kutta (RK4) and finite difference (FD) methods.

To evaluate each method, we calculate the mean squared error of the predicted solution against ground-truth solutions obtained either analytically or using state of the art numerical solvers. Where analytical solutions do not exist, we use SciPy's `solve_ivp`[53] for ODEs and the fast Fourier transform (FFT)[7] method for PDEs. We use the original DEQGAN setup but add instance noise to p_{fake} and p_{real} and use residual monitoring to terminate poor-performing runs in the first 25% of training iterations. Results on all neural network methods are obtained with hyperparameters tuned for DEQGAN. In Appendix A.2, we tune hyperparameters for each classical loss function for comparison but do not observe a significant difference.

In their original paper, Randle et al. [45] demonstrated that DEQGAN can obtain solutions with multiple orders of magnitude lower mean squared errors than classical unsupervised neural networks on six example equations. In this work, we leverage the added stability offered by instance noise and residual monitoring to tackle more challenging problems. More specifically, we add six new equations, including highly non-linear PDEs and systems of ODEs. In Sections 3.1 and 3.2, we describe these differential equations and compare the performance of DEQGAN to that of classical neural network methods. In Section 3.3, we provide a summary of the results obtained on the full suite of twelve equations by all methods, including traditional numerical solvers.

3.1 Ordinary Differential Equations

An ordinary differential equation (ODE) is an equation containing a single independent variable and derivatives of a dependent variable with respect to that independent variable. Randle et al. [45] previously applied DEQGAN to solve a variety of ODEs and systems of ODEs, including the exponential decay equation, the simple harmonic oscillator, a non-linear oscillator, coupled oscillators, and the SIR disease model. Results on these problems are provided in Appendix A.3.

Encouraged by the promising results achieved by DEQGAN on these problems, we test our method on two additional systems of ODEs that exhibit higher degrees of non-linearity and more complex dynamics.

3.1.1 Hamilton Equations (HAM)

Consider a particle moving through a potential V, the trajectory of which is described by the system of ordinary differential equations

$$\begin{cases} \dot{x}(t) = p_x \\ \dot{y}(t) = p_y \\ \dot{p_x}(t) = -V_x \\ \dot{p_y}(t) = -V_y \end{cases} \tag{3.1}$$

We solve the system for $x(t)\big|_{t=0} = 0, y(t)\big|_{t=0} = 0.3, p_x(t)\big|_{t=0} = 1, p_y(t)\big|_{t=0} = 0$, and $t \in [0, 1]$. The functions $x(t)$ and $y(x)$ represent the horizontal and vertical positions of the particle

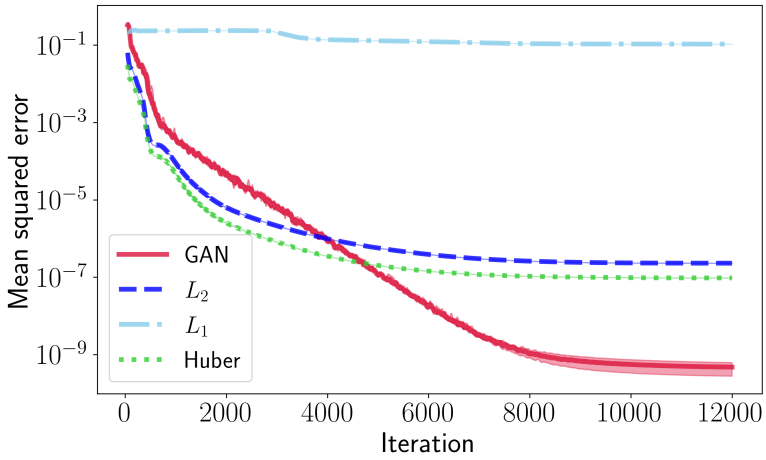


Figure 3.1: Mean squared errors vs. iteration for DEQGAN, L_2, L_1 and Huber loss for the Hamilton system of ODEs. We perform ten randomized trials and plot the median (bold) and $(25, 75)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

in time, while $p_x(t)$ and $p_y(t)$ represent the velocity functions. V_x and V_y are the derivatives of the potential with respect to x and y. We construct the potential V by summing ten random bivariate Gaussians

$$V = -\frac{A}{2\pi\sigma^2}\sum_{i=1}^{10}\exp\left(-\frac{1}{2\sigma^2}||\mathbf{x}(t) - \boldsymbol{\mu}_i||_2^2\right) \tag{3.2}$$

where $\mathbf{x}(t) = [x(t), y(t)]^T$. We use $A = 0.1, \sigma = 0.1$, and sample each $\boldsymbol{\mu}_i = [\mu_x, \mu_y]^T$ from $[0, 1] \times [0, 1]$ uniformly at random. As these equations lack an analytical solution, we use SciPy's `solve_ivp`[53] to obtain ground-truth solutions.

After re-parameterizing the generator outputs to satisfy the initial conditions using Equation 1.4, we set LHS to be the vector

$$LHS = \left[\frac{dx}{dt} - p_x, \frac{dy}{dt} - p_y, \frac{dp_x}{dt} + V_x, \frac{dp_y}{dt} + V_y\right]^T \tag{3.3}$$

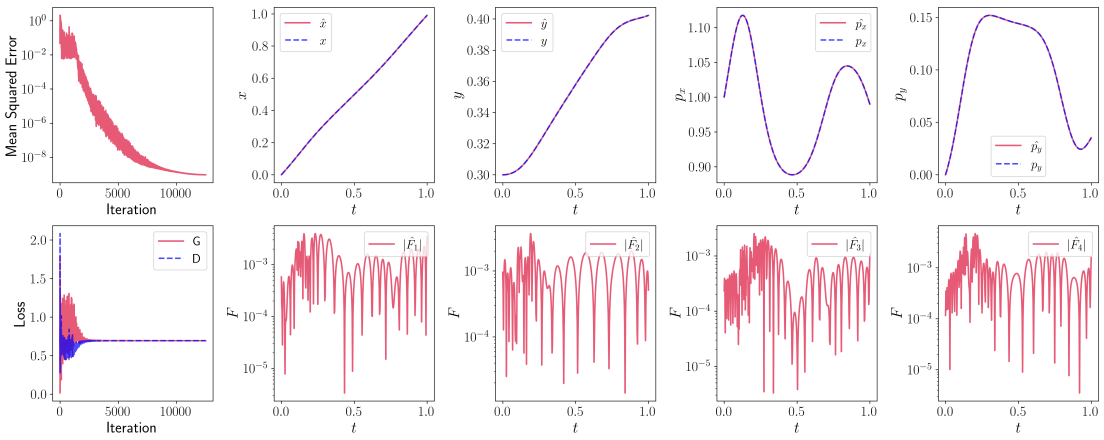


Figure 3.2: Visualization of DEQGAN training for the Hamiltonian system of equations. The top left figure plots the mean squared error vs. iteration. To the right, we plot the predictions of the generator $\hat{x}, \hat{y}, \hat{p_x}, \hat{p_y}$ and the numerical solutions x, y, p_x, p_y as functions of time t. On the bottom left, we plot the value of the generator (G) and discriminator (D) losses at each iteration. To the right, we show the absolute value of the residuals of the predicted solution $\hat{F_j}$ for each equation j.

and $RHS = [0, 0, 0, 0]^T + \varepsilon$, where ε represents added instance noise. Note that the LHS has an additional dimension over the time points sampled that is not explicitly depicted in Equation 3.3. We present the results of training DEQGAN to solve this system of differential equations in Figure 3.2. We see that the generator and discriminator losses converge to equilibrium relatively quickly, and the mean squared error of the DEQGAN solution against the numerical solution decreases steadily throughout training.

In Figure 3.1, we compare the performance of DEQGAN to that of unsupervised neural networks that use classical loss functions by performing ten randomized trials of each method. Note that for DEQGAN, we use residual monitoring to terminate poor-performing runs within the first 25% of training iterations. While L_1 and L_2 attain rapidly decreasing mean squared errors initially, they are ultimately outperformed by DEQGAN by several orders of magnitude.

3.1.2 Modified Einstein's Gravity Equations (EIN)

The most challenging system of ODEs we consider comes from cosmology. In 1929, Edwin Hubble observed that light emitted by distant galaxies becomes redshifted as it travels through space, implying the existence of an expanding universe.[24] Until the end of the 20th century, cosmologists believed that gravity would slow this expansion over time, but observations from type Ia supernovae in 1998[46] revealed that the opposite is true: the expansion of the universe is *accelerating*.

Since then, several cosmological models have been proposed to explain this phenomenon. Some of these, such as the ΛCDM model, assume that general relativity is the correct theory of gravity and rely on the existence of dark energy. Others explain the accelerated expansion of the universe by directly modifying Einstein's theory.

Hu-Sawicky $f(R)$ gravity is one model that falls into the latter category. In their detailed manuscript, Chantada et al.[8] explain how $f(R)$ gravity results in a set of modified Friedmann equations, which can be manipulated to derive the system of five ODEs in Equation 3.4 by introducing the dimensionless variables x, y, v, Ω, and r.

$$\begin{cases} \dfrac{dx}{dz} = \dfrac{1}{z+1}(-\Omega - 2v + x + 4y + xv + x^2) \\ \dfrac{dy}{dz} = \dfrac{-1}{z+1}(vx\Gamma(r) - xy + 4y - 2yv) \\ \dfrac{dv}{dz} = \dfrac{-v}{z+1}(x\Gamma(r) + 4 - 2v) \\ \dfrac{d\Omega}{dz} = \dfrac{\Omega}{z+1}(-1 + 2v + x) \\ \dfrac{dr}{dz} = \dfrac{-r\Gamma(r)x}{z+1} \end{cases} \tag{3.4}$$

where

$$\Gamma(r) = \frac{(r+b)\left[(r+b)^2 - 2b\right]}{4br}. \tag{3.5}$$

Unlike the other ODEs we have solved using DEQGAN, this system uses z, which represents redshift, as the independent variable. Because larger values of z correspond to earlier points in the history of the universe, our "initial conditions" are defined at the maximum value of the independent variable, which we denote z_0 (the earliest point in time). More specifically, we solve the equations for $z \in [0, z_0]$. To accommodate the initial value re-parameterization performed by Equation 1.4, Chantada et al. [8] proposed the variable change $z' = 1 - z/z_0$, which we employ in our experiments as well.

One additional issue with this system is that r tends to grow rapidly with z. As neural networks often struggle with unscaled data, we follow Chantada et al. [8] and use the variable change $r' = \ln(r)$, which prevents the dependent variable from becoming too large.

Putting these two variable changes together, we obtain the more neural network-friendly system

$$
\begin{cases}
\dfrac{dx}{dz'} = \dfrac{-z_0}{-z_0(z'-1)+1}(-\Omega - 2v + x + 4y + xv + x^2) \\
\dfrac{dy}{dz'} = \dfrac{-z_0}{-z_0(z'-1)+1}(vx\Gamma(r') - xy + 4y - 2yv) \\
\dfrac{dv}{dz'} = \dfrac{z_0 v}{-z_0(z'-1)+1}(x\Gamma(r') + 4 - 2v) \\
\dfrac{d\Omega}{dz'} = \dfrac{-z_0\Omega}{-z_0(z'-1)+1}(-1 + 2v + x) \\
\dfrac{dr'}{dz'} = \dfrac{z_0\Gamma(r')x}{-z_0(z'-1)+1}
\end{cases}
\tag{3.6}
$$

where

$$
\Gamma(r') = \frac{(e^{r'} + b)\left[(e^{r'} + b)^2 - 2b\right]}{4be^{r'}}. \tag{3.7}
$$

Finally, Chantada et al.[8] also modify the standard initial value re-parameterization in Equation 1.4 to incorporate additional information about the relationship between $f(R)$ gravity and the ΛCDM model, which is based on the standard theory of general relativity. In particular, the authors note that as $b \to 0$, $f(R)$ gravity approaches ΛCDM. Therefore, b represents the amount of deviation from Einstein's general relativity. When $b = 0$, the solution to Equation 3.4 is given exactly by

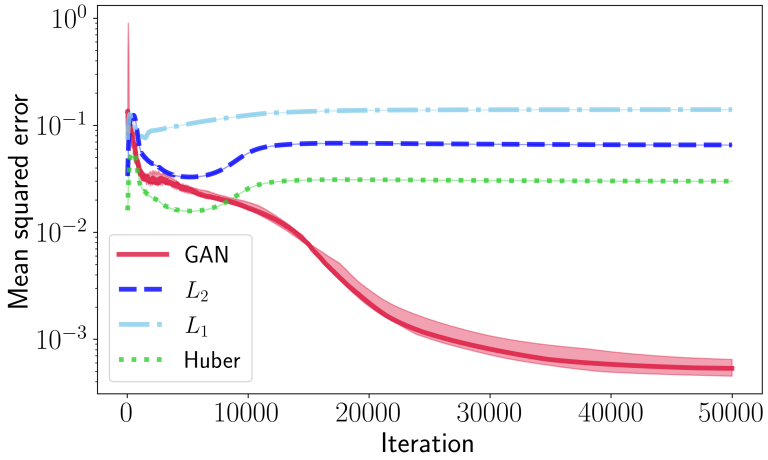


Figure 3.3: Mean squared errors vs. iteration for DEQGAN, L_2, L_1 and Huber loss for the modified Einstein's gravity system of ODEs. We perform ten randomized trials and plot the median (bold) and $(25, 75)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

$$\hat{\mathbf{u}}(z) = \begin{bmatrix} x(z) \\ y(z) \\ v(z) \\ \Omega(z) \\ r(z) \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{\Omega_{m,0}(1+z)^3+2(1-\Omega_{m,0})}{2[\Omega_{m,0}(1+z)^3+(1-\Omega_{m,0})]} \\ \frac{\Omega_{m,0}(1+z)^3+4(1-\Omega_{m,0})}{2[\Omega_{m,0}(1+z)^3+(1-\Omega_{m,0})]} \\ \frac{\Omega_{m,0}(1+z)^3}{\Omega_{m,0}(1+z)^3+(1-\Omega_{m,0})} \\ \frac{\Omega_{m,0}(1+z)^3+4(1-\Omega_{m,0})}{(1-\Omega_{m,0})} \end{bmatrix} \tag{3.8}$$

We note that $\hat{\mathbf{u}}(z_0)$ also gives the final values for Equation 3.4 for any value of b. Therefore, the final re-parameterization for this problem is

$$\tilde{\mathbf{u}}(z') = \hat{\mathbf{u}}(z') + \left(1 - e^{-(z'-z'_0)}\right)\left(1 - e^{-\alpha b}\right)\mathbf{u}_\theta(z') \tag{3.9}$$

where $\mathbf{u}_\theta = [x_\theta, y_\theta, v_\theta, \Omega_\theta, r'_\theta]$ is the vector of generator outputs for each dependent variable and α is a hyperparameter that controls how closely the ΛCDM model approximates the solution to the $f(R)$ system. The first exponential term in Equation 3.9 ensures that the initial values at z'_0 are satisfied and decays this constraint in z', while the second exponential term shrinks the neural network output relative to $\mathbf{u}_\theta$ depending on b and α.

In our experiments, we set $z_0 = 10, \Omega_{m,0} = 0.15, b = 5, \alpha = 1/30$ and generate solutions for $z \in [0, z_0]$, which corresponds to solving Equation 3.6 for $z' \in [0, 1]$. While the physical interpretation of these parameters is beyond the scope of this thesis, we note that Equations 3.6 and 3.7 exhibit a high degree of non-linearity, making this a very challenging problem. As the system has no analytical solution, ground truth solutions are again obtained using SciPy.

The LHS and RHS for this problem are constructed in the same fashion as described in previous sections, and the results obtained by DEQGAN are shown in Figure 3.4, where we have returned z' and r' to their original scales. We see that the generator and discriminator losses converge relatively quickly, and the mean squared error decreases steadily to an accurate solution. Towards larger values of z, however, we observe that the error of the DEQGAN solution tends to increase, particularly for $y(z)$ and $v(z)$.

Figure 3.3 plots the results of running ten randomized runs of DEQGAN and unsupervised neural networks that use L_2, L_1 and Huber loss functions. While the classical loss functions appear to achieve decreasing mean squared errors in the early training iterations, all three ultimately fail to solve the system. While these equations have been solved by Chantada et al. [8] using unsupervised neural networks, their training procedure leveraged a custom loss function enforcing conservation

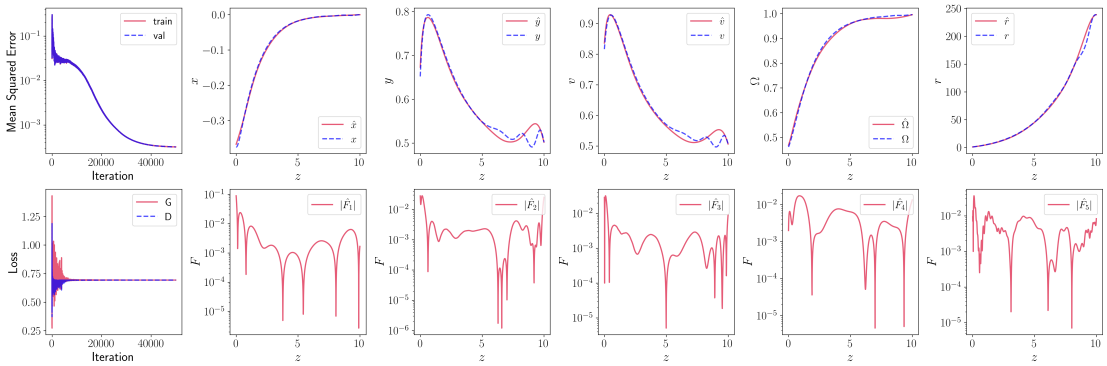


Figure 3.4: Visualization of DEQGAN training for the modified Einstein's gravity system of equations. The top left figure plots the mean squared error vs. iteration. To the right, we plot the predictions of the generator $\hat{x}, \hat{y}, \hat{v}, \hat{\Omega}, \hat{r}$ and the numerical solutions x, y, v, Ω, r as functions of redshift z. On the bottom left, we plot the value of the generator (G) and discriminator (D) losses at each iteration. To the right, we show the absolute value of the residuals of the predicted solution $\hat{F}_j$ for each equation j.

of mass relationships among the dependent variables.

This highlights a major advantage of DEQGAN over classical unsupervised neural networks: while the L_2, L_1 and Huber loss functions fail to capture the complex optimization landscape of this problem, DEQGAN is able to *automatically* learn a loss function that optimizes the generator to produce accurate solutions.

3.2 Partial Differential Equations

Naturally extending ODEs into higher dimensions, partial differential equations (PDEs) contain multiple independent variables and partial derivatives of a dependent variable with respect to those variables. In their original work, Randle et al. [45] used DEQGAN to solve the Poisson equation, a fundamental linear PDE that we provide results for in Appendix A.3.

Given the importance of PDEs for modeling a wide range of phenomena in science and engineering, including chemical reactions and fluid dynamics, we add four more to our suite of differential equations.

3.2.1 Heat Equation (HEA)

The heat equation is a second-order linear partial differential equation that models the diffusion of heat through a given region. It is one of the most fundamental PDEs and is given by

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} \tag{3.10}$$

where u represents temperature, x is a spatial dimension, and t is a time dimension. With a single spatial dimension, this equation could be used to model the dissipation of heat along a one-dimensional rod over time. By the second law of thermodynamics, the rate at which heat flows from hotter to cooler regions is proportional to the difference in temperature between them. Equation 3.10 captures this with the 1-dimensional Laplacian $\frac{\partial^2 u}{\partial x^2}$ which, for each point in space, gives the

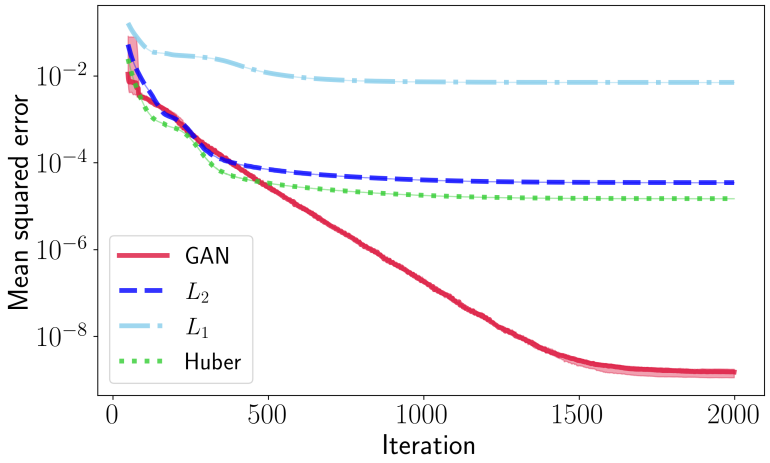


Figure 3.5: Mean squared errors vs. iteration for DEQGAN, L_2, L_1 and Huber loss for the heat equation. We perform ten randomized trials and plot the median (bold) and $(25, 75)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

average temperature difference between that point and the surrounding region. The model assumes that heat is transferred evenly and in all directions at a rate controlled by the thermal diffusivity κ.

In our experiments, we solve the equation for $\kappa = 1$ and $(x, t) \in [0, 1] \times [0, 0.2]$. We subject the equation to an initial condition and Dirichlet boundary conditions given by

$$\begin{aligned} u(x,t)\Big|_{t=0} &= \sin(\pi x) \\ u(x,t)\Big|_{x=0} &= 0 \\ u(x,t)\Big|_{x=1} &= 0 \end{aligned} \tag{3.11}$$

which has an analytical solution

$$u(x,t) = e^{-\kappa\pi^2 t}\sin(\pi x). \tag{3.12}$$

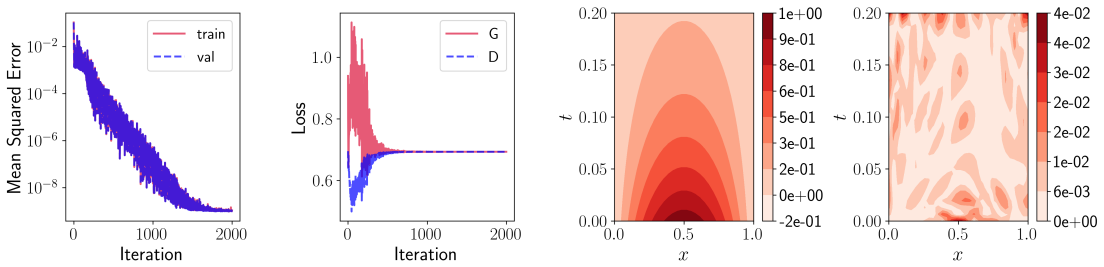


Figure 3.6: Visualization of DEQGAN training for the heat equation. The left-most figure plots the mean squared error vs. iteration. To the right, we plot the value of the generator (G) and discriminator (D) losses at each iteration. In the third figure, we plot the prediction of the generator $\hat{u}$ as a function of position (x, t). The right-most figure plots the absolute value of the residual $\hat{F}$ as a function of (x, t).

To ensure that the DEQGAN solution exactly satisfies the initial and boundary conditions listed in Equation 3.11, we use the following re-parameterization from Chen et al. [9].

$$\tilde{u}(x,t) = u(x,t)\big|_{t=0} + \tilde{x}(1-\tilde{x})\left(1 - e^{-\tilde{t}}\right) u_\theta(x,t) \tag{3.13}$$

where $\tilde{x} = \frac{x - x_0}{x_1 - x_0}$ and $\tilde{t} = \frac{t - t_0}{t_1 - t_0}$ are normalized versions of x and t. Note that x_0 and x_1 represent minimum and maximum values, respectively. We see that when $t = t_0, \tilde{t} = 0$, which satisfies the initial condition. At the boundaries, we have $x = x_0, x_1$, giving $\tilde{x} = 0, 1$, leaving only the values of the initial condition at the boundaries.

As usual, we construct the "fake" data vector by moving all terms to the left-hand side of the equation to obtain

$$LHS = \frac{\partial u}{\partial t} - c^2 \frac{\partial^2 u}{\partial x^2} \tag{3.14}$$

and set $RHS = \epsilon$, where ϵ represents instance noise. Figure 3.6 shows that while the generator and

discriminator losses converge initially, they eventually reach equilibrium, allowing DEQGAN to obtain highly accurate solutions with residuals distributed evenly across the (x, t) domain.

In Figure 3.5, we see that DEQGAN outperforms the classical unsupervised neural networks by multiple orders of magnitude across ten randomized trials, suggesting that the discriminator network is able to learn a loss function that is much more effective than the L_2, L_1 and Huber losses.

With both a time and a spatial dimension, our PDE solution can also be plotted in 3D, as illustrated by Figure 3.7. As expected, the temperature decreases evenly along the spatial domain as time progresses.

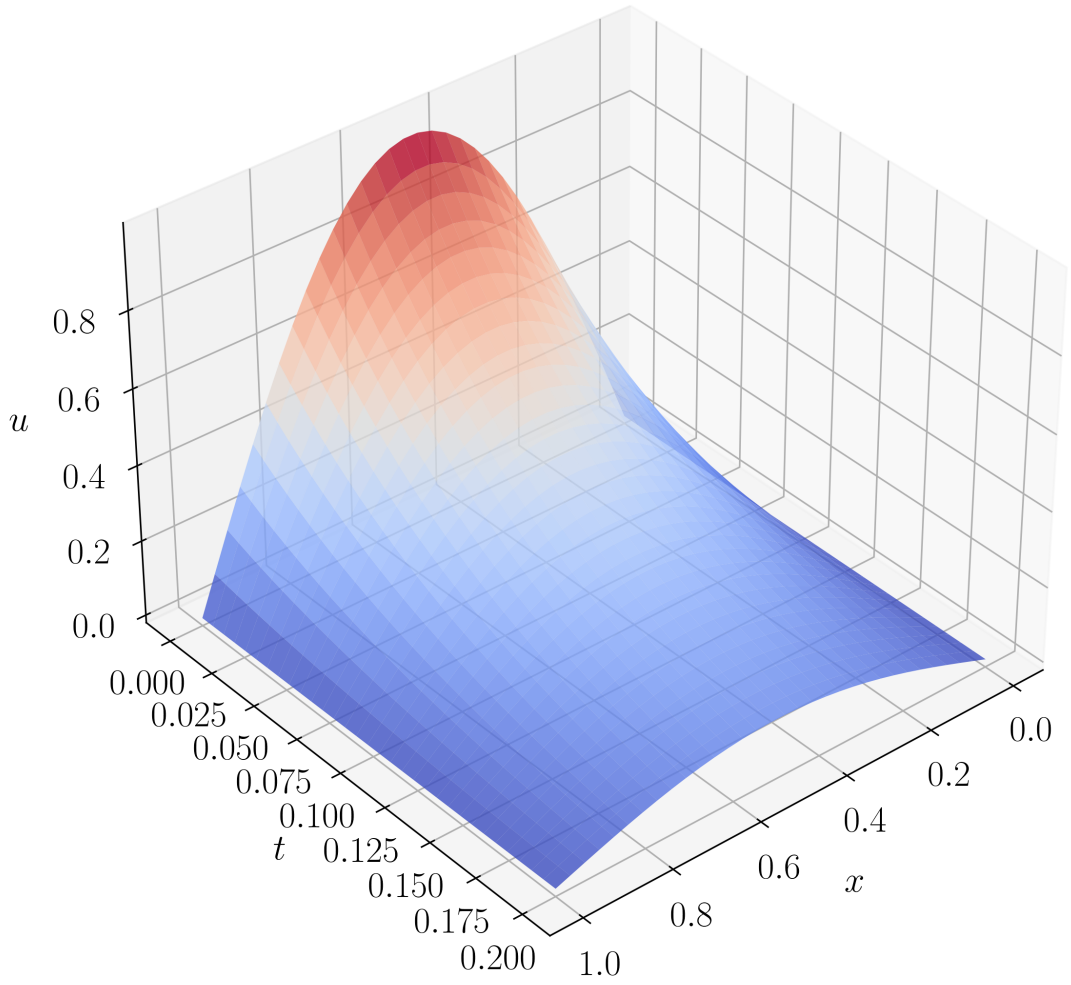


Figure 3.7: DEQGAN solution to the heat equation plotted in 3D. We use perturbed sampling to train on points from a noisy 32×32 grid and plot the final solution on an evenly spaced 32×32 grid.

3.2.2 Wave Equation (WAV)

The wave equation is a second-order linear PDE that models the motion of classical waves, such as those found on strings, in fluids, or in light. The equation is given by

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \tag{3.15}$$

where u is the vertical displacement of the wave, x is a spatial dimension, and t is a time dimension. Given a single spatial dimension, this equation could be used to model the motion of a vibrating wave on a string. The constant c specifies the speed of the wave.

We set $c = 1$ and solve the equation for $(x, t) \in [0, 1] \times [0, 1]$. To specify the equation, we use the same initial condition and boundary conditions as in 3.11 but require an added Neumann condition due to the second time derivative.

$$\begin{aligned} u(x,t)\Big|_{t=0} &= \sin(\pi x) \\ u_t(x,t)\Big|_{t=0} &= 0 \\ u(x,t)\Big|_{x=0} &= 0 \\ u(x,t)\Big|_{x=1} &= 0 \end{aligned} \tag{3.16}$$

This yields the analytical solution

$$u(x,t) = \cos(c\pi t)\sin(\pi x). \tag{3.17}$$

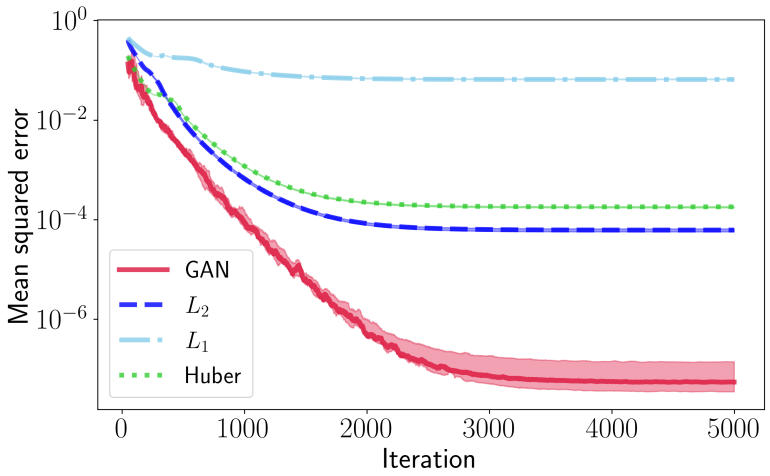


Figure 3.8: Mean squared errors vs. iteration for DEQGAN, L_2, L_1 and Huber loss for the wave equation. We perform ten randomized trials and plot the median (bold) and $(25, 75)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

To handle the added Neumann condition, we re-parameterize the generator output according to the following modification of Equation 3.13.

$$\tilde{u}(x,t) = u(x,t)\big|_{t=0} + \tilde{x}(1-\tilde{x})\left(1 - e^{-\tilde{t}^2}\right) u_\theta(x,t) \tag{3.18}$$

The results of DEQGAN training on this problem are shown in Figure 3.9. We see that the generator and discriminator losses reach equilibrium relatively quickly. The generator predictions converge to a very accurate solution, as measured by the mean squared error computed against the analytical solution, and the final residuals are distributed fairly evenly across the problem domain.

Figure 3.8 illustrates how the performance of DEQGAN compares to that of classical unsupervised neural networks that use pre-specified loss functions. As we have observed in other problems, our method obtains multiple orders of magnitude lower mean squared error than the L_1, L_2 and

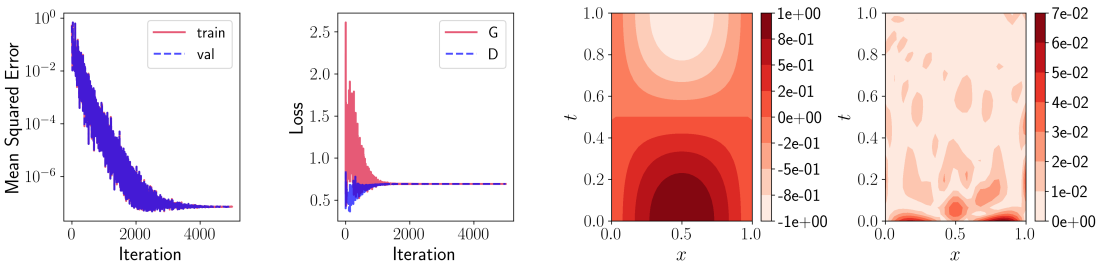


Figure 3.9: Visualization of DEQGAN training for the wave equation. The left-most figure plots the mean squared error vs. iteration. To the right, we plot the value of the generator (G) and discriminator (D) losses at each iteration. In the third figure, we plot the prediction of the generator $\hat{u}$ as a function of position (x, t). The right-most figure plots the absolute value of the residual $\hat{F}$ as a function of (x, t).

Huber loss functions.

We also notice that DEQGAN shows greater variability in performance across the ten randomized trials than the classical methods. This reflects the relative instability of DEQGAN training. While instance noise and residual monitoring are effective strategies for maximizing the proportion of stable runs of DEQGAN, Figure 3.8 indicates that the runs we retain still tend to have greater variability than the runs of classical unsupervised neural networks.

In Figure 3.10, we visualize the DEQGAN solution in three dimensions. As time progresses, we observe that the initial sinusoidal wave oscillates in the negative direction until it has been completely reflected about the x-axis.

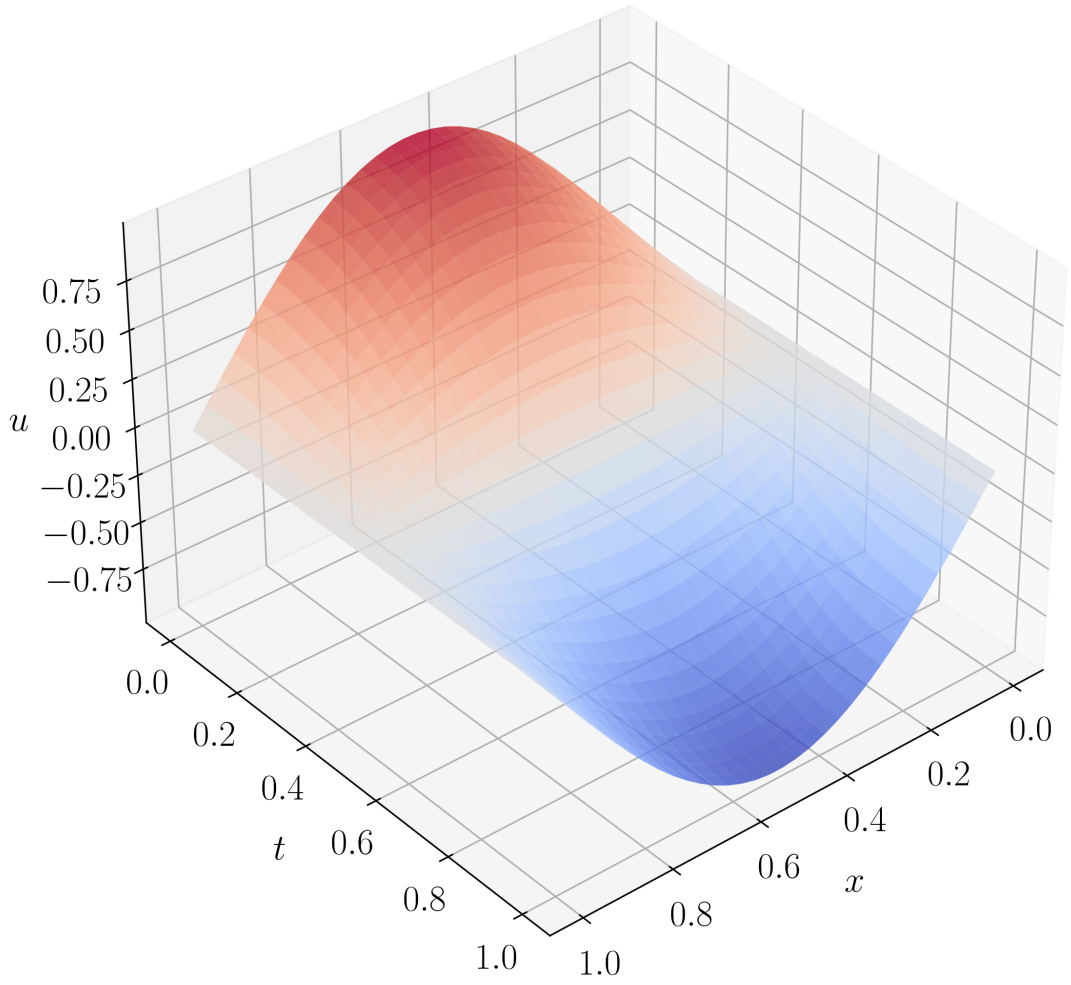


Figure 3.10: Solution to wave equation plotted in 3D. We use perturbed sampling to train on points from a noisy 32×32 grid and plot the final solution on an evenly spaced 32×32 grid.

3.2.3 Burgers' Equation (BUR)

Next, we consider Burgers' viscous equation, an important non-linear PDE that frequently arises in the modeling of fluids. The equation is given by

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{3.19}$$

where x is a spatial dimension, t is a time dimension, and $u(x,t)$ represents the speed of the fluid at x and t. This equation could be used to describe the speed of a fluid running through a thin ideal pipe over time. The viscosity of the fluid is controlled by the constant ν.

In our experiments, we set $\nu = 0.001$ and solve the equation for $(x,t) \in [-5,5] \times [0,2.5]$. To specify the equation, we use the following initial condition and Dirichlet boundary conditions:

$$\begin{aligned} u(x,t)\Big|_{t=0} &= \frac{1}{\cosh(x)} \\ u(x,t)\Big|_{x=-5} &= 0 \\ u(x,t)\Big|_{x=5} &= 0 \end{aligned} \tag{3.20}$$

Given these conditions, we can use the re-parameterization in Equation 3.13, which we introduced for the heat equation. As this equation has no analytical solution, we use the fast Fourier transform (FFT) method[7] to obtain accurate ground truth solutions.

The results obtained by DEQGAN are summarized in Figure 3.12. While the generator loss oscillates significantly in the first half of training iterations, DEQGAN ultimately converges to an

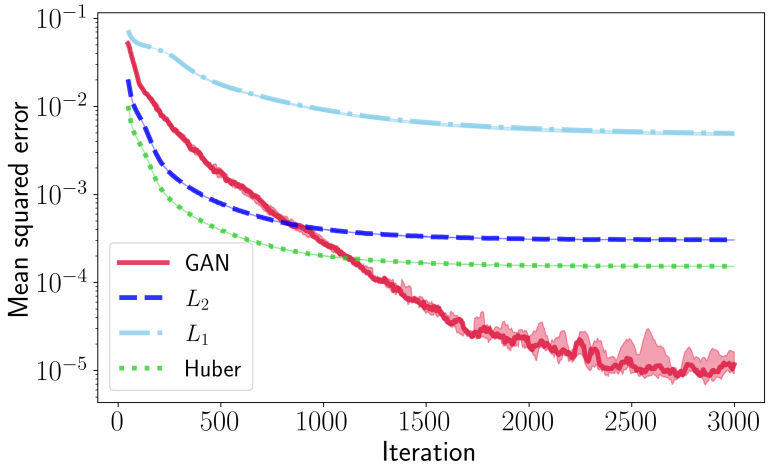


Figure 3.11: Mean squared errors vs. iteration for DEQGAN, L_2, L_1 and Huber loss for Burgers' equation. We perform ten randomized trials and plot the median (bold) and $(25, 75)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

accurate solution, as reflected in the mean squared error plot (note that here we plot only the "validation MSE," which is computed over a fixed grid, to avoid having to compute the ground truth solution using the FFT over a different noisy grid at each training iteration).

In the third plot in the top row, we see a "top-down" view of the DEQGAN solution over x and t, which reveals that the initial condition grows increasingly steep in the positive direction as time progresses. Note, however, that the solution does not become infinitely steep due to the regularizing diffusive term νu_{xx}. This behavior is known as *shock wave* formation and is frequently observed in solutions to Burgers' equation. The shock is illustrated more clearly by the second row of plots in Figure 3.12, which shows snapshots of the solution along x at different points in time. Finally, the top right plot indicates that nearly all residual error comes from the top of the shock wave. Because the "ground-truth" FFT solution is, itself, an approximation, however, it is difficult to know whether this error is truly associated with DEQGAN.

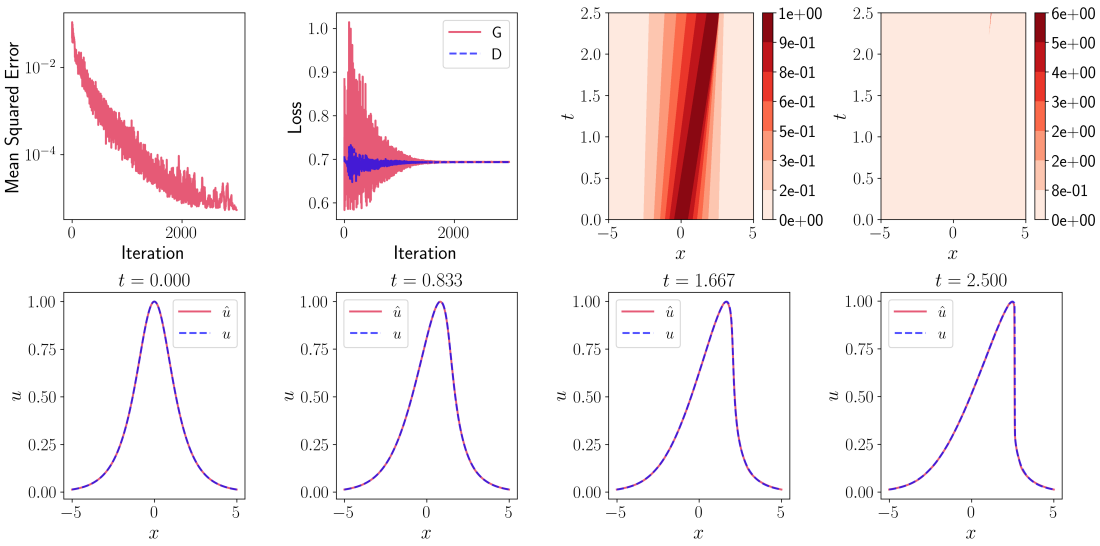


Figure 3.12: Visualization of DEQGAN training for Bugers' equation. The top left figure plots the mean squared error vs. iteration. To the right, we plot the value of the generator (G) and discriminator (D) losses at each iteration. To the right of that, we plot the prediction of the generator $\hat{u}$ as a function of position (x, t). The top right figure plots the absolute value of the residual $\hat{F}$ as a function of (x, t). The plots in the second row show snapshots of the 1D wave at different points along the time domain.

In Figure 3.11, we see that once again, DEQGAN outperforms classical unsupervised neural networks that use L_2, L_1 and Huber loss functions. As observed previously, however, DEQGAN also shows greater variability in mean squared error across randomized runs.

We plot the DEQGAN solution to Burgers' equation in 3D in Figure 3.13, providing another view of the shock wave formation. While we sample training points from a noisy 64×64 grid, we make the final prediction over a 1000×100 evenly-spaced grid to obtain a smoother 3D plot. In fact, this highlights one of the advantages of using neural networks to solve differential equations over numerical methods: because a neural network provides a closed-form solution, we can generate predictions over any grid we like after training is complete. Numerical methods, on the other hand, provide an approximate solution over a pre-specified mesh of points.

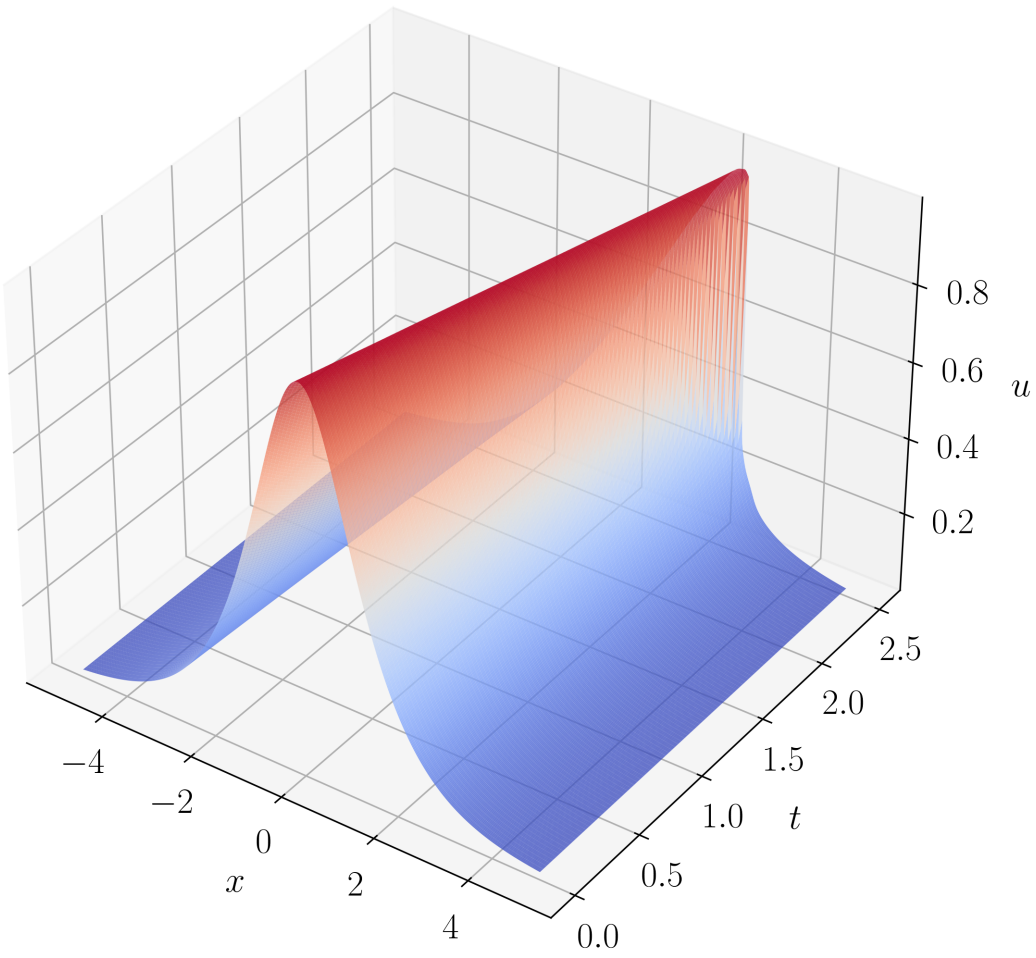


Figure 3.13: Solution to Burgers' equation plotted in 3D. We use perturbed sampling to train on points from a noisy 64×64 grid and plot the final solution on an evenly spaced 1000×100 grid.

3.2.4 Allen-Cahn Equation (ACA)

Finally, we consider the Allen-Cahn PDE, a non-linear reaction-diffusion equation given by

$$\frac{\partial u}{\partial t} - \epsilon \frac{\partial^2 u}{\partial x^2} - u + u^3 = 0 \tag{3.21}$$

This equation is frequently used to describe phase separation, such as the separation observed between oil and water, or phase transition, such as that between water vapor and liquid. As in previous problems, x and t represent spatial and time dimensions, respectively, while $u(x, t)$ is the phase function between the two substances at each point in space and time. The constant ϵ is related to the width of the phase boundary between the substances and controls diffusivity.

In our experiments, we use $\epsilon = 0.001$ and generate solutions for $(x, t) \in [0, 2\pi] \times [0, 5]$. We subject the equation to an initial condition and Dirichlet boundary conditions given by

$$\begin{aligned} u(x,t)\Big|_{t=0} &= \frac{1}{4}\sin(x) \\ u(x,t)\Big|_{x=0} &= 0 \\ u(x,t)\Big|_{x=2\pi} &= 0 \end{aligned} \tag{3.22}$$

which can again be enforced using the re-parameterization in Equation 3.13. Like Burgers' equation, the Allen-Cahn equation has no analytical solutions, so we use the fast Fourier transform to obtain ground truth solutions for comparison.

Figure 3.15 shows the results of training DEQGAN on this problem. Although the generator

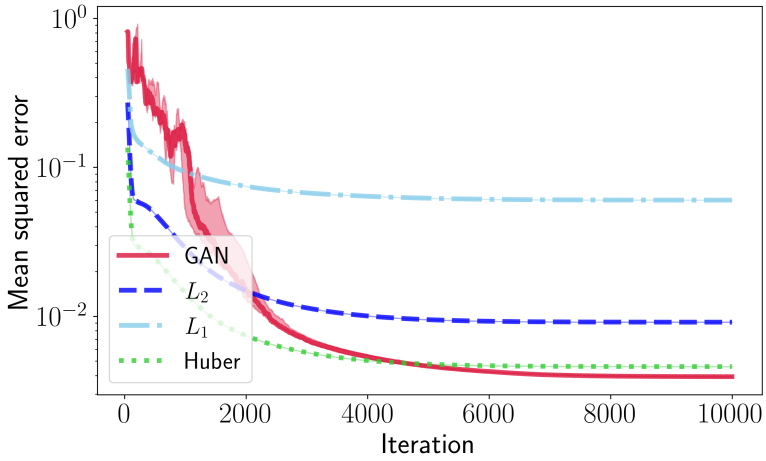


Figure 3.14: Mean squared errors vs. iteration for DEQGAN, L_2, L_1 and Huber loss for the Allen-Cahn equation. We perform ten randomized trials and plot the median (bold) and $(25, 75)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

and discriminator losses reach equilibrium, the second row of plots clearly illustrates that the DEQ-GAN solution is accurate almost everywhere except along the boundaries. This is confirmed by the residual plot in the top right of Figure 3.15, which shows that errors are high along $x = x_0, x_1$ for all $t > t_0$. Further, the plot of mean squared error vs. training iteration suggests that DEQGAN has converged to a locally optimal solution.

Seeing as the high residual regions appear to be restricted to the boundaries, we suspect that DE-QGAN is struggling to produce accurate solutions in these regions due to the re-parameterization in Equation 3.13, which decays the initial condition exponentially in time, but perhaps not quickly enough. In particular, for small $t > t_0$, the re-parameterized solution may still be down-weighting the neural network output in comparison to the initial condition, causing errors to arise at small t and propagate across the time domain.

To address this, we propose re-weighting the LHS by shrinking the residuals associated with

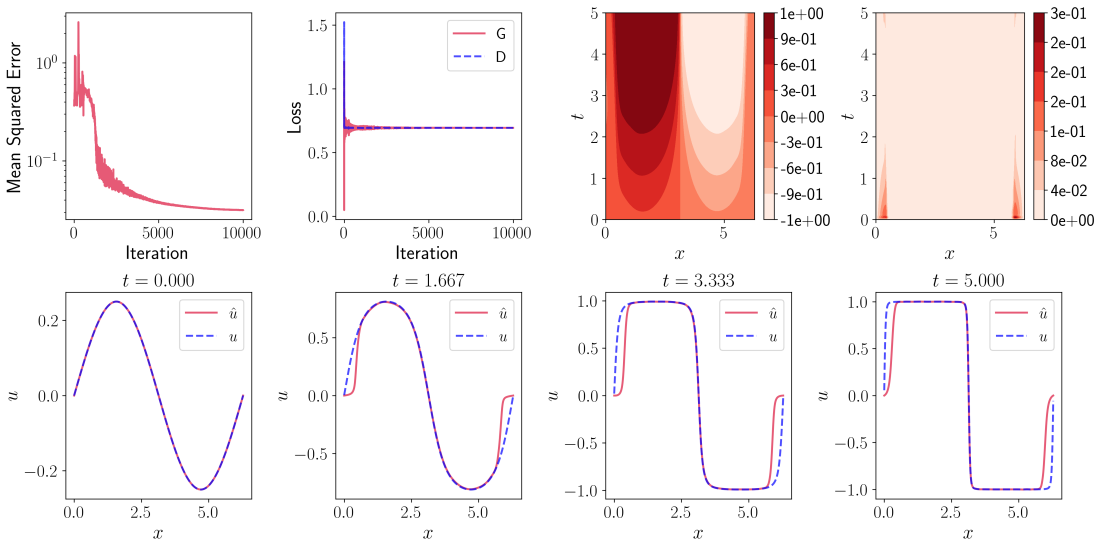


Figure 3.15: Visualization of DEQGAN training for the Allen-Cahn equation. The top left figure plots the mean squared error vs. iteration. To the right, we plot the value of the generator (G) and discriminator (D) losses at each iteration. To the right of that, we plot the prediction of the generator $\hat{u}$ as a function of position (x, t). The top right figure plots the absolute value of the residual $\hat{F}$ as a function of (x, t). The plots in the second row show snapshots of the 1D wave at different points along the time domain.

points further along the time domain. Essentially, this makes the residuals associated with points early in the time domain appear larger, encouraging the discriminator network to pay more attention to them. More formally, we construct the LHS as follows.

$$LHS = \left(\frac{\partial \tilde{u}}{\partial t} - \epsilon \frac{\partial^2 \tilde{u}}{\partial x^2} - \tilde{u} + \tilde{u}^3 \right) e^{-t/\lambda} \tag{3.23}$$

where λ is a hyperparameter that controls the degree of re-weighting. After training DEQGAN with a new set of tuned hyperparameters that includes λ, we obtained the results shown in Figure 3.16. The second row of plots reveals that the solution is much more accurate along the boundary. This difference is particularly pronounced between the snapshots at $t = 1.667$ in Figures 3.15 and

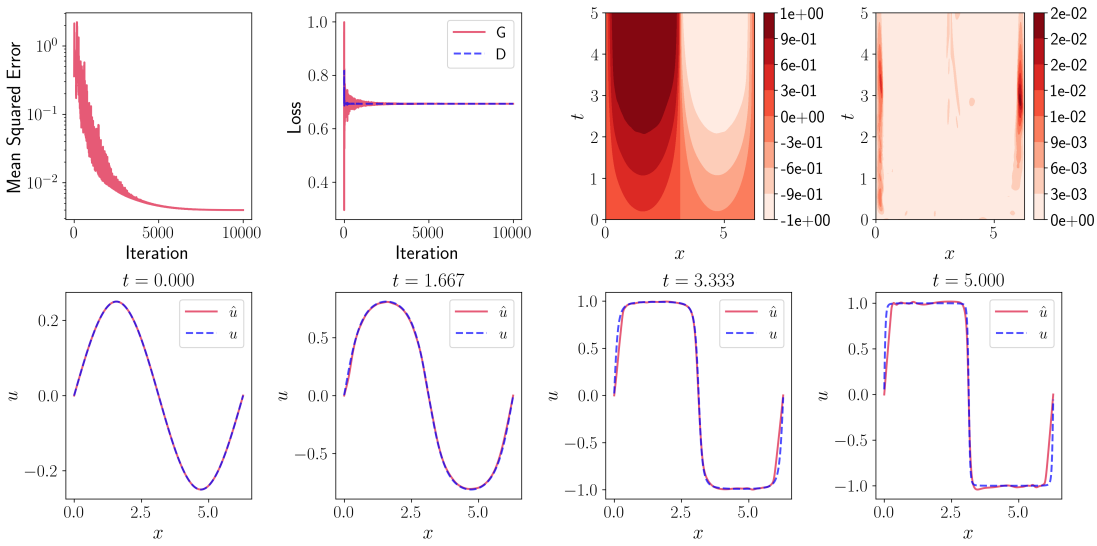


Figure 3.16: Visualization of DEQGAN training for the Allen-Cahn equation with the LHS reweighed according to Equation 3.23.

3.16. In the snapshots at larger t, however, we see that the solution quality begins to degrade, which makes sense given our re-weighting of the LHS.

In Figure 3.14, we compare the performance of DEQGAN to that of the classical unsupervised neural networks. Although our method still performs the best, it only narrowly outperforms the Huber loss function, suggesting that the boundary errors may remain a performance bottleneck.

In Figure 3.17, we plot the final DEQGAN solution obtained with the reweighed LHS in three dimensions. As we did for Burgers' equation, we sampled training points from a noisy 64×64 grid but generated a solution for plotting over an evenly-spaced 1000×100 grid. Despite the "wrinkles" in the solution towards the end of the time domain, we still observe the expected transformation of the sinusoidal initial condition into a square wave over time.

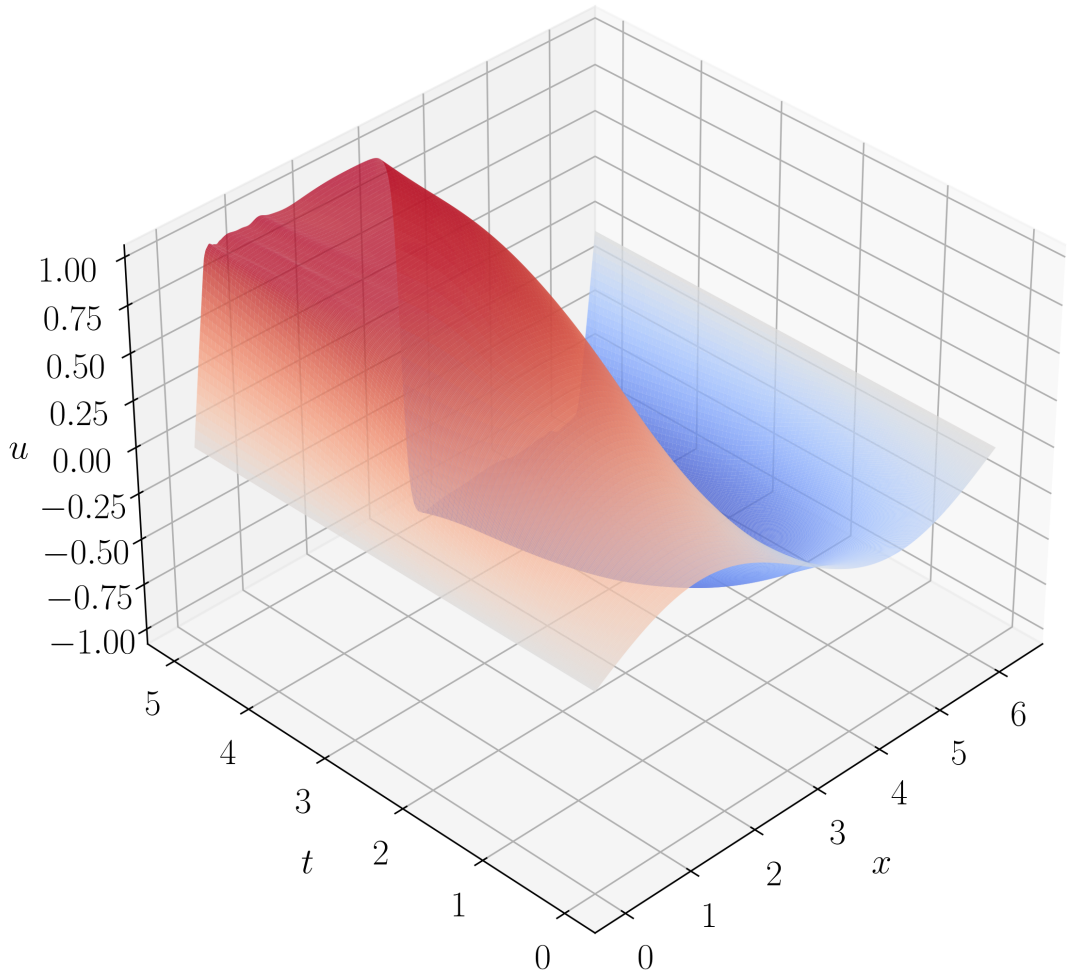


Figure 3.17: Solution to Allen-Cahn equation with the reweighed LHS plotted in 3D. We use perturbed sampling to train on points from a noisy 64×64 grid and plot the final solution on an evenly spaced 1000×100 grid.

Problem (Key)	Equation
Exponential Decay (EXP)	$\dot{x}(t) + x(t) = 0$
Simple Harmonic Oscillator (SHO)	$\ddot{x}(t) + x(t) = 0$
Non-Linear Oscillator (NLO)	$\ddot{x}(t) + 2\beta\dot{x}(t) + \omega^2 x(t) + \phi x(t)^2 + \epsilon x(t)^3 = 0$
Coupled Oscillators (COO)	$\begin{cases} \dot{x}(t) = -ty \\ \dot{y}(t) = tx \end{cases}$
SIR Disease Model (SIR)	$\begin{cases} \dot{S}(t) &= -\beta I(t)S(t)/N \\ \dot{I}(t) &= \beta I(t)S(t)/N - \gamma I(t) \\ \dot{R}(t) &= \gamma I(t) \end{cases}$
Poisson Equation (POS)	$u_{xx} + u_{yy} = 2x(y-1)(y-2x+xy+2)e^{x-y}$

Table 3.1: Differential equations previously solved by DEQGAN.

3.3 Summary of Results

In addition to the six ODEs and PDEs discussed above, we applied DEQGAN with instance noise and residual monitoring to the six differential equations solved by Randle et al. [45] in the original DEQGAN paper. These equations are listed in Table 3.1. Note that in Appendix A.1, we also specify the problem domains and hyperparameters used to train DEQGAN for all twelve equations.

In Table 3.2, we report the minimum mean squared error obtained by each method over ten randomized runs for all twelve problems. The mean squared error is computed against analytial solutions when they exist. Otherwise, we use high quality numerical solvers (SciPy's `solve_ivp` [53] and the fast Fourier transform [7] for ODEs and PDEs, respectively) to obtain ground truth solutions. The hyperparameters used for all neural networks were tuned for DEQGAN. In Appendix A.2, we report the mean squared errors obtained by tuning each unsupervised neural network method separately, but we generally do not observe a significant difference.

	Mean Squared Error				
Key	L_1	L_2	Huber	DEQGAN	Traditional
EXP	$3 \cdot 10^{-3}$	$2 \cdot 10^{-5}$	$1 \cdot 10^{-5}$	$3 \cdot 10^{-16}$	$2 \cdot 10^{-14}$ (RK4)
SHO	$9 \cdot 10^{-6}$	$1 \cdot 10^{-10}$	$6 \cdot 10^{-11}$	$4 \cdot 10^{-13}$	$1 \cdot 10^{-11}$ (RK4)
NLO	$6 \cdot 10^{-2}$	$1 \cdot 10^{-9}$	$9 \cdot 10^{-10}$	$1 \cdot 10^{-12}$	$4 \cdot 10^{-11}$ (RK4)
COO	$5 \cdot 10^{-1}$	$1 \cdot 10^{-7}$	$1 \cdot 10^{-7}$	$1 \cdot 10^{-8}$	$2 \cdot 10^{-9}$ (RK4)
SIR	$7 \cdot 10^{-5}$	$3 \cdot 10^{-9}$	$1 \cdot 10^{-9}$	$1 \cdot 10^{-10}$	$5 \cdot 10^{-13}$ (RK4)
HAM	$1 \cdot 10^{-1}$	$2 \cdot 10^{-7}$	$9 \cdot 10^{-8}$	$1 \cdot 10^{-10}$	$7 \cdot 10^{-14}$ (RK4)
EIN	$6 \cdot 10^{-2}$	$2 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	$3 \cdot 10^{-4}$	$4 \cdot 10^{-7}$ (RK4)
POS	$4 \cdot 10^{-6}$	$1 \cdot 10^{-10}$	$6 \cdot 10^{-11}$	$4 \cdot 10^{-13}$	$3 \cdot 10^{-10}$ (FD)
HEA	$6 \cdot 10^{-3}$	$3 \cdot 10^{-5}$	$1 \cdot 10^{-5}$	$6 \cdot 10^{-10}$	$4 \cdot 10^{-7}$ (FD)
WAV	$6 \cdot 10^{-2}$	$4 \cdot 10^{-5}$	$6 \cdot 10^{-4}$	$1 \cdot 10^{-8}$	$7 \cdot 10^{-5}$ (FD)
BUR	$4 \cdot 10^{-3}$	$2 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$4 \cdot 10^{-6}$	$1 \cdot 10^{-3}$ (FD)
ACA	$6 \cdot 10^{-2}$	$9 \cdot 10^{-3}$	$4 \cdot 10^{-3}$	$3 \cdot 10^{-3}$	$2 \cdot 10^{-4}$ (FD)

Table 3.2: Minimum mean squared errors achieved by DEQGAN and classical unsupervised neural networks that use L_2, L_2 and Huber loss functions across ten randomized trials, as well as the mean squared error achieved by traditional numerical methods on all twelve problems.

In addition, we calculate the accuracy of the fourth-order Runge Kutta (RK4) and finite difference (FD) methods against the ground truth solutions for initial and boundary value problems, respectively, in order to see how the neural network methods compare to traditional numerical solvers. For a fair comparison, we produced numerical solutions using the same number of grid points as were used to train the neural network methods. These grid sizes are also provided in Appendix A.1.

We observe that DEQGAN outperforms the classical unsupervised neural network methods on every differential equation tested, often by multiple orders of magnitude. Further, DEQGAN achieves solution accuracies that are competitive with those attained using the RK4 and FD numerical methods.

The results summarized in Table 3.2, along with the array of advantages offered by neural net-

works methods in general – including having closed-form solutions that enable prediction over arbitrary grids and offer more accurate interpolation – provide compelling reasons to use DEQGAN for solving differential equations over traditional numerical methods. One common criticism of neural networks, however, is that they often require significant computational resources for training, making them generally slower than numerical methods. In the next section, we explore how transfer learning can be leveraged to address this weakness.

The more clearly we can focus our attention on the wonders and realities of the universe about us, the less taste we shall have for destruction.

Rachel Carson

4

Transfer Learning

Unsupervised neural networks offer an array of advantages over traditional numerical methods for solving differential equations. One common criticism, however, is that they are usually slow to train. Further, the classical unsupervised approach trains a neural network to generate solutions to an equation for a particular initial condition, making it computationally expensive

to obtain solutions for multiple initial conditions. This limits the utility of neural network solvers in real-world applications such as large-scale physics simulations, which often require many solutions corresponding to different initial conditions. Numerical methods, by contrast, are generally much faster.

In addition, and as discussed in Chapter 2, DEQGAN can be sensitive to hyperparameter settings. While techniques such as instance noise and residual monitoring increase the robustness of DEQGAN to different model weight initializations and learning rates, we found that our method required hyperparameter tuning for each new equation in order to achieve the best results.

In this chapter, we explore how transfer learning can be leveraged to address both of these concerns. More specifically, we show how to learn a more expressive generator network that is able to solve a given differential equation or system of equations for multiple initial conditions at once. In doing so, we are able to 1) drastically reduce the time required for DEQGAN to solve differential equations for a set of initial conditions, and 2) significantly improve the ability of DEQGAN to generate accurate solutions to new initial conditions given a single set of hyperparameters.

4.1 Multi-Head DEQGAN

Transfer learning encompasses a broad range of techniques that aim to store the information gained on one (source) task and adapt it to a different but related (target) task. Much recent work has focused on developing transfer learning methods for neural networks, in part because they are often expensive to train, but also because they are able to learn highly expressive and *generalizable* induc-

tive biases within their many hidden layers. [32,34] Therefore, neural networks not only have much to gain from transfer learning, but they are also likely to be very good at it.

Transfer learning with neural networks is often performed by training the network on a source task, "freezing" the majority of the weights in the network (remove them from gradient computations), and then "fine-tuning" the network by training only the last few layers on the target task. In this way, it is possible to obtain a model that performs well on the target task without having to train it from scratch, cutting computational costs significantly. This approach has been successfully applied to a wide range of problems, especially in computer vision [19,32,48,1] and natural language processing. [35,23,54]

Recent work has also applied transfer learning to unsupervised neural networks for solving differential equations. Flamant et al. [14] proposed neural network solution bundles, which provide solutions over a range of initial conditions by incorporating equation parameters into the loss function. More recently, Desai et al. [12] showed how a trained network can be used to perform one-shot inference for linear ODEs and PDEs. In our work, we apply these ideas to DEQGAN with the goal of reducing the computational cost of our method while improving its robustness.

Let's say that we would like to solve a differential equation $F(t, u(t), u'(t), u''(t), ...)$ for a set of n initial conditions $I = \{t_0^{(1)}, t_0^{(2)}, \ldots, t_0^{(n)}\}$. To do this, we propose a slightly modified DEQGAN architecture. As illustrated in Figure 4.1, we first pass an input t into a "base" generator network. Rather than re-parameterize a single output of this network to satisfy one initial condition, however, we pass the outputs of the last hidden layer to n output heads $h_1, h_2, \ldots, h_n$, where h_i consists of a single linear layer and is responsible for generating solutions corresponding to initial

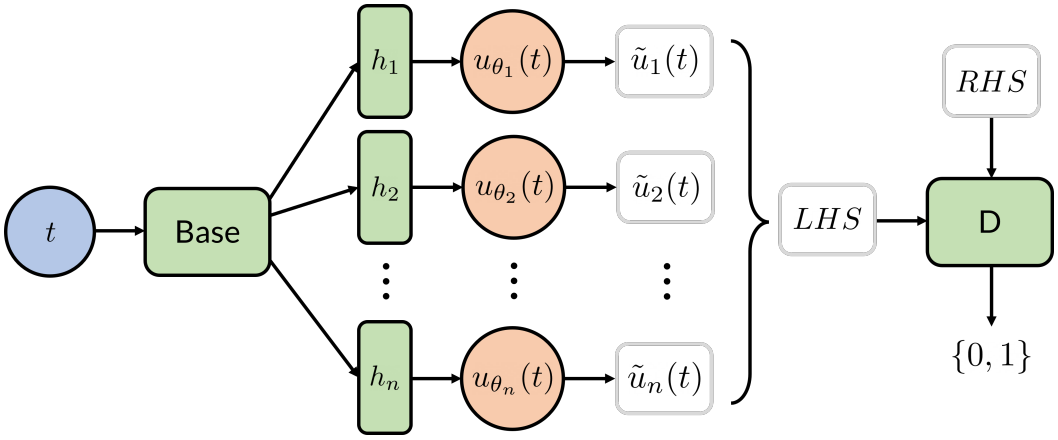


Figure 4.1: Multi-head DEQGAN architecture for solving a differential equation for n initial conditions at once. An input t is passed to a base generator network, the last hidden layer of which is connected to n output heads $h_1, \ldots, h_n$, each of which consists of a single linear layer. The output $u_{\theta_i}(t)$ of each h_i is re-parameterized to satisfy the i^{th} initial condition, giving the predicted solution $\tilde{u}_i(t)$. We use the predicted solutions for all n initial conditions to construct the "fake" LHS vector, which are passed to the discriminator, along with the "real" RHS data.

condition $t_0^{(i)}$.

We then adjust the output $u_{\theta_i}(t)$ of each head according to the same initial value re-parameterization used previously.

$$\tilde{u}_i(t) = u_0^{(i)} + \left(1 - e^{-\left(t - t_0^{(i)}\right)}\right) u_{\theta_i}(t) \tag{4.1}$$

where $u_0^{(i)} = u(t)\big|_{t=t_0^{(i)}}$. This gives us predicted solutions $\tilde{u}_1(t), \tilde{u}_2(t), \ldots, \tilde{u}_n(t)$. The original DEQGAN architecture yielded just a single solution, which we plugged into the differential equation F to construct the "fake" LHS vector. To handle n solutions, we simply concatenate the LHS vectors over all solutions, as shown in Equation 4.2.

$$
\begin{aligned}
LHS = \big[& F\left(t, \tilde{u}_1(t), \tilde{u}_1'(t), \tilde{u}_1''(t), \dots\right), \\
& F\left(t, \tilde{u}_2(t), \tilde{u}_2'(t), \tilde{u}_2''(t), \dots\right), \\
& \dots \\
& F\left(t, \tilde{u}_n(t), \tilde{u}_n'(t), \tilde{u}_n''(t), \dots\right) \big]
\end{aligned}
\tag{4.2}
$$

Without instance noise, the RHS will then be a vector of zeros with the same dimension as Equation 4.2. From here, training proceeds in the same way as the original DEQGAN setup, with the discriminator being trained to distinguish between the LHS and RHS and the generator aiming to fool the discriminator.

By training the generator on multiple initial conditions at once, the "base" network is forced to learn a representative basis for the differential equation, while each head learns a linear transformation from the output of the base to the solution corresponding to a particular initial condition. This approach enables us to transfer the basis learned from the set of initial conditions I_b used for base training to a second set of initial conditions I_t. The training procedures used to train the base and perform transfer learning are defined more precisely in Algorithms 1 and 2, respectively.

The critical difference between these two algorithms is that while Algorithm 1 optimizes an entire generator network, Algorithm 2 "freezes" the weights of the pre-trained generator (turns off their gradient computations) and fine-tunes only the linear output heads. This makes transfer learning much less computationally expensive than base training per iteration.

Algorithm 1 DEQGAN Base Training

Input: Differential equation F, randomly-initialized generator G and discriminator D, total iterations N, number of base initial conditions n, domain $\mathcal{I}$ of initial conditions
Set I_b to be a list of n evenly spaced initial conditions in $\mathcal{I}$
Add linear output heads $h_1, h_2, \ldots, h_n$ to the last hidden layer of G
for $i = 1$ **to** N **do**
  Run DEQGAN training algorithm
end for
Remove $h_1, h_2, \ldots, h_n$ from G
Output: G

Algorithm 2 DEQGAN Transfer Learning

Input: Differential equation F, pre-trained generator G, randomly-initialized discriminator D, total iterations N, list of m arbitrary initial conditions for transfer $I_t = \{t_0^{(1)}, t_0^{(2)}, \ldots, t_0^{(m)}\}$
Turn off gradient computation for the weights of G
Add linear output heads $h_1, h_2, \ldots, h_m$ to the last hidden layer of G
for $i = 1$ **to** N **do**
  Run DEQGAN training algorithm
end for
Output: G

In order to learn a representative basis over the domain $\mathcal{I}$ of initial conditions that we would like to solve for, Algorithm 1 trains DEQGAN on a set of evenly-spaced initial conditions I_b spanning this domain. Algorithm 2, however, allows us to transfer to an arbitrary set of initial conditions $I_t \in \mathcal{I}$. By using the pre-trained generator output of Algorithm 1 as an input to Algorithm 2, we are able to generate solutions for I_t more efficiently than for I_b. Finally, Algorithm 2 outputs a fine-tuned multi-head generator that we can use to produce solutions for all initial conditions I_t in a single forward pass.

4.2 Branched Flows

In the experiments that follow, we use the Hamilton system (HAM) of ODEs, which we introduced in Chapter 3 (see Section 3.1.1 for details) and include below for reference.

$$\begin{cases} \dot{x}(t) = p_x \\ \dot{y}(t) = p_y \\ \dot{p_x}(t) = -V_x \\ \dot{p_y}(t) = -V_y \end{cases} \tag{4.3}$$

In our experiments, we set $x(t)\big|_{t=0} = 0, p_x(t)\big|_{t=0} = 1, p_y(t)\big|_{t=0} = 0$, and solve the system for $t \in [0, 1]$. V_x and V_y are the derivatives of the potential V with respect to x and y, and we construct V by summing ten bivariate Gaussians with means randomly sampled $[\mu_x, \mu_y] \in [0, 1] \times [0, 1]$, as described in Section 3.1.1. To demonstrate transfer learning, we vary $y(t)\big|_{t=0} = y_0$, the initial vertical position of the particle, in $[0, 1]$. Therefore, this interval defines our domain of initial conditions $\mathcal{I}$.

These equations can be used to model a phenomenon in wave mechanics known as "branched flow," whereby waves passing through a potential are scattered in different directions, forming tree-like patterns. Consider a series of particles moving through a potential V from different vertical starting positions. In Figure 4.2, we illustrate this effect by generating solutions to the system at 100 equally-spaced vertical starting positions y_0 using SciPy's `solve_ivp` numerical solver. At around $(x, y) \in [0.5, 0.4]$, we see that the rays self-intersect, forming what is known as a "caustic."

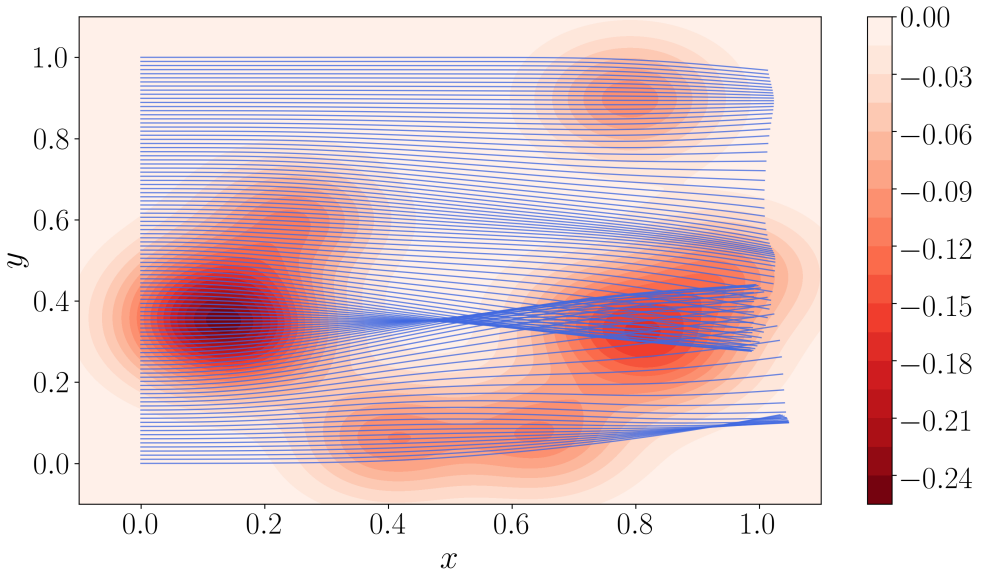


Figure 4.2: Solutions to the Hamilton system obtained by a numerical solver. The red background indicates the potential obtained by summing ten random bivariate Gaussians. The blue lines plot $x(t)$ vs. $y(t)$ for 100 evenly-spaced initial conditions in $[0, 1]$, which correspond to ray trajectories through the potential starting from 100 different vertical positions.

4.3 Experimental Results

In this section, we present the results obtained by DEQGAN on a variety of transfer learning tasks involving the Hamilton equations. First, we discuss the training of the base generator. Then, we utilize the pre-trained base for three different types of transfer: single initial condition transfer, multiple initial condition transfer, and potential transfer.

4.3.1 Base Training

Effective transfer learning necessitates an accurate and expressive base generator. Therefore, it is important to choose a large enough number of initial conditions n to train the base in Algorithm 1.

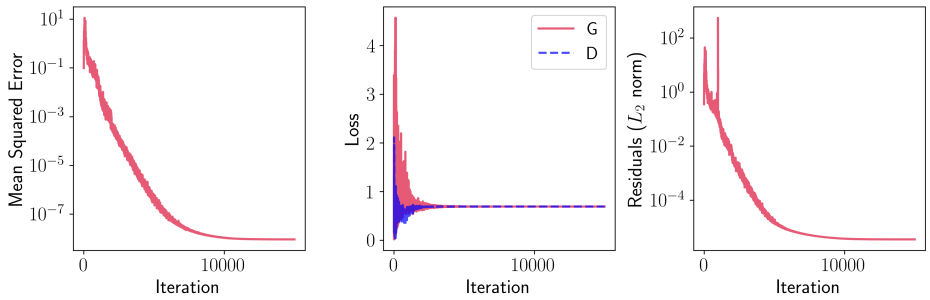
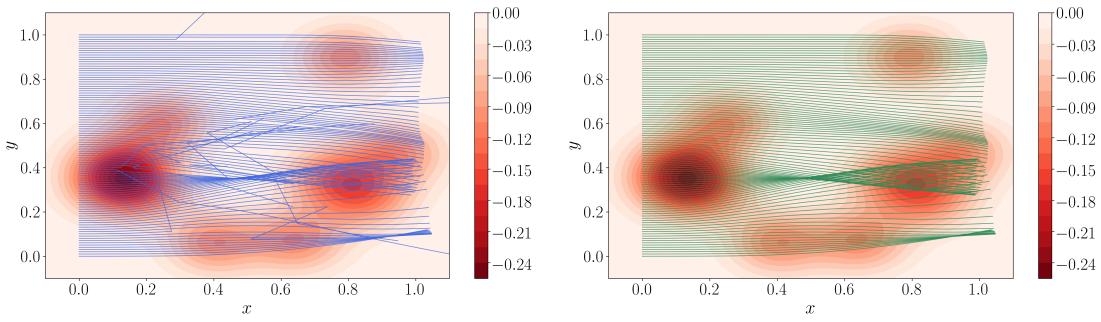


Figure 4.3: Visualization of multi-head DEQGAN base training for the Hamilton system over 11 evenly-spaced initial conditions $I_b = [0, 0.1, \ldots, 1.0]$. The left-most figure plots the mean squared error (averaged over all I_b) vs. iteration. To the right, we plot the value of the generator (G) and discriminator (D) losses at each iteration. The right-most figure plots the L_2 of the equation residuals (the LHS) vs. iteration.

In our experiments, we found that while DEQGAN was able to generate accurate predictions up to around $n = 11$, solution quality deteriorated beyond this point. We believe that this is because the dimensionality of the "fake" data vector (Equation 4.2) increases with n, presenting an increasingly difficult classification task for the discriminator. To balance accuracy with expressivity, we chose $n = 11$ and trained the the multi-head DEQGAN model on evenly-spaced initial conditions $I_b = \{0, 0.1, \ldots, 1.0\}$.

Figure 4.3 visualizes the results of this training procedure. In the leftmost plot, the mean squared error, which is calculated as an average over all I_b, converges to $\sim 10^{-9}$ after around 1000 iterations. To the right, we see that the generator and discriminator losses reach convergence relatively quickly, which is indicative of a good training run. These results suggest successful training of the base generator. In the sections that follow, we investigate how this base can be transferred to other problems.



(a) Original DEQGAN training

(b) Single IC transfer

Figure 4.4: Predicted solutions obtained by the original DEQGAN model trained from scratch (left) and by leveraging transfer learning with the pre-trained base generator (right). In both procedures, we solve one initial condition at a time to obtain solutions for 100 evenly-spaced initial conditions in $[0, 1]$.

4.3.2 Single Initial Condition Transfer

Having completed base training, we are now ready for transfer learning. In this section, we transfer to one new initial condition during each training run, i.e., $m = 1$ and $I_t = \{y_0\}$. As indicated in Algorithm 2, this requires replacing the 11 heads used to train the base with a single randomly initialized head, which will be fine-tuned to solve the equation for $y(t)\big|_{t=0} = y_0$.

We repeat this procedure for the same 100 evenly-spaced initial conditions in $[0, 1]$ shown in Figure 4.2. For comparison, we also generate solutions for these 100 initial conditions using the original (single-headed) DEQGAN architecture, which does not leverage a pre-trained base. In Figure 4.4, we plot the ray trajectories corresponding to the solutions generated by each method.

Figure 4.4a plots the solutions obtained by training the original version of DEQGAN from scratch. While the majority of trajectories match the numerical solutions shown in Figure 4.2, we see that several veer significantly off course. This is an indication of DEQGAN's sensitivity to hy-

perparameters. In particular, the model was trained using the same hyperparameters for all initial conditions but was not always able to produce accurate solutions.

In Figure 4.4b, however, we see that that transferring the pre-trained base to the same initial conditions eliminates this issue entirely. This comparison reveals that transfer learning is able to improve the robustness of our method significantly. Further, because transfer learning requires only fine-tuning linear heads, it is also more computationally efficient than training DEQGAN from scratch. In Table 4.1, we compare the average time (in seconds) taken per iteration of training DEQGAN from scratch against DEQGAN with transfer learning.

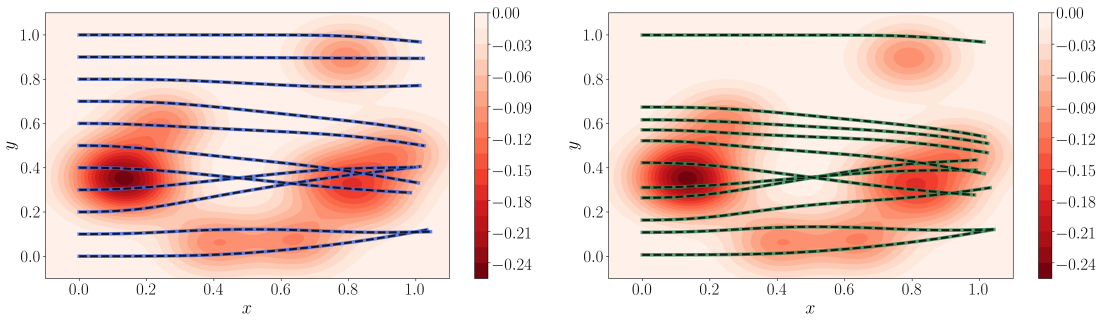
	Time per Iteration (seconds)
Training from scratch	0.0211
Transfer learning	0.0172

Table 4.1: Average time taken per iteration by training DEQGAN from scratch vs. DEQGAN with transfer learning. Each training procedure involves solving the Hamilton system for one initial condition at a time.

4.3.3 Multiple Initial Condition Transfer

In the previous section, we transferred a base trained on 11 initial conditions to one new condition at a time. In this section, we leverage the multi-head DEQGAN architecture to transfer to multiple initial conditions at once, i.e., $m > 1$. As we found for base training, however, adding too many heads to the model caused solution accuracy to deteriorate.

In Figure 4.5, we plot the ray trajectories for the solutions produced by DEQGAN with $n = 11$ heads during base training and $m = 11$ heads during transfer. As shown in Figure 4.5a, the base



(a) Base training

(b) Multiple IC transfer

Figure 4.5: Ray trajectories learned by DEQGAN after base training and multiple initial condition transfer. The base was trained on 11 evenly spaced initial conditions in $[0, 1]$, while transfer learning was used to obtain solutions to 11 new initial conditions sampled uniformly at random in the same interval. The dashed black lines represent the ground-truth solutions obtained by SciPy's numerical solver, while the blue and green lines show the solutions obtained during base training and transfer learning, respectively.

is trained on the same evenly-spaced initial conditions $I_b = \{0, 0.1, \ldots 1.0\}$ used previously. For transfer learning, we uniformly at random sample new initial conditions $I_t \in [0, 1]$. For both training procedures, we see that the predicted solutions are indistinguishable from the "ground-truth" numerical solutions, which are indicated in dashed black lines.

For a clearer comparison between base training and transfer learning, Figure 4.6 plots the mean squared error averaged over all initial conditions vs. iteration for each training procedure. Notably, transfer learning not only appears to converge slightly more rapidly than base training, but also reaches a lower final MSE of $\sim 10^{-10}$. This suggests that the linear transformations added to the pre-trained base enable DEQGAN to converge to more accurate solutions.

As discussed in the previous section, fine-tuning the linear heads is also less computationally expensive than training the base per iteration. Therefore, transfer learning enables DEQGAN to obtain solutions to multiple initial conditions at once both more efficiently *and* more accurately in

comparison to base training.

It should be noted, however, that there is an apparent limit to the number of initial conditions for which the multi-head DEQGAN model can generate accurate solutions. In our experiments, we found that the mean squared error converged to around $\sim 10^{-5}$ for 15 initial conditions and $\sim 10^{-4}$ for 20 initial conditions. Transferring to 11 new initial conditions, however, consistently yielded accurate results.

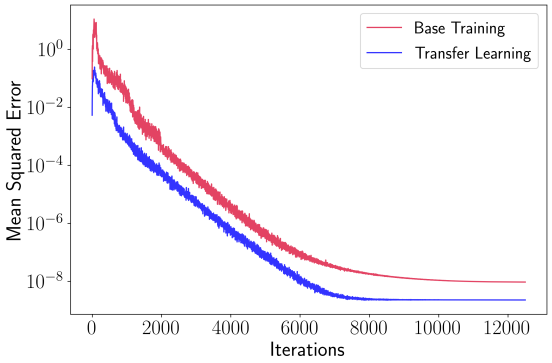
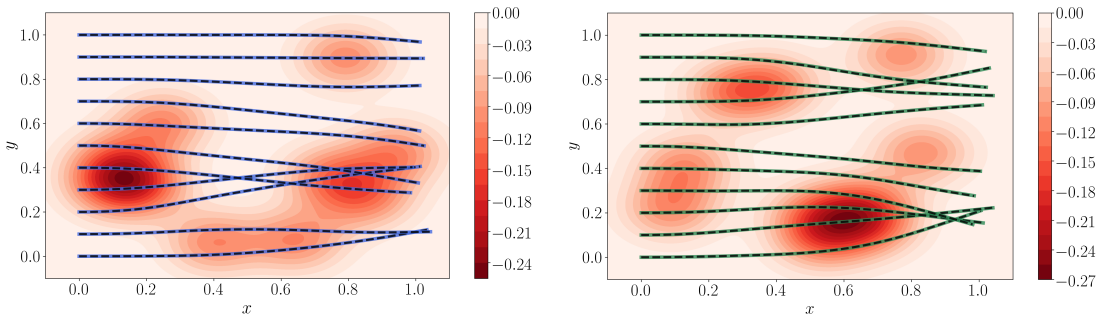


Figure 4.6: Mean squared errors vs. iteration during base training (red) and multiple initial condition transfer (blue). For each training procedure, the mean squared errors are calculated against the ground-truth numerical solutions and averaged over all initial conditions.

4.3.4 Potential Transfer

So far, we have demonstrated how transfer learning can be used to more efficiently compute solutions for a new set of initial conditions with the same potential V. In this section, we show that this procedure can be used to transfer from one potential to another while keeping the initial conditions unchanged.

More specifically, we randomly re-sample ten bivariate Gaussian means $[\mu_x, \mu_y] \in [0, 1] \times [0, 1]$



(a) Base training **(b)** Potential transfer

Figure 4.7: Ray trajectories learned by DEQGAN after base training and potential transfer. The base was trained on 11 evenly spaced initial conditions in $[0, 1]$ with potential V_b, while transfer learning was used to obtain solutions to the same initial conditions with potential V_t. The dashed black lines represent the ground-truth solutions obtained by SciPy's numerical solver, while the blue and green lines show the solutions obtained during base training and transfer learning, respectively.

to construct a new potential

$$V_t = -\frac{A}{2\pi\sigma^2}\sum_{i=1}^{10}\exp\left(-\frac{1}{2\sigma^2}||\mathbf{x}(t)-\boldsymbol{\mu}_i||_2^2\right) \tag{4.4}$$

where $\boldsymbol{\mu}_i = [\mu_x, \mu_y]^T$. The potential used during base training, which we denote V_b, was constructed with scaling factor $A = 0.1$ and standard deviation $\sigma = 0.1$, which define the statistical properties of the potential. Therefore, the basis learned by our pre-trained base is also defined by these values of A and σ, implying that we should build the transfer potential V_t using the same values. If we changed these values, we would be attempting to transfer the basis learned by the base generator to a system with different statistical properties.

To prevent this transfer learning task from becoming too difficult, we solve the system for the same initial conditions used to train the base. The training procedure is equivalent to that described in Algorithm 2, but uses the new potential V_t to define the Hamilton system and sets $I_t = I_b$.

The ray trajectories obtained over V_t by DEQGAN are illustrated in Figure 4.7b. For comparison, we include the trajectories discovered over V_b during base training in Figure 4.7a. While the initial conditions in these two plots are the same, the new potential deflects the rays in different ways, producing different trajectories. Nonetheless, the multi-head DEQGAN model is able to discover solutions that are indistinguishable from those obtained using a numerical solver, which are indicated by dashed black lines.

In Figure 4.8, we visualize the mean squared errors averaged over all initial conditions for base training and transfer learning. Although transfer learning converges more quickly, we notice that it reaches a slightly less accurate local optimum, with mean squared error $\sim 10^{-8}$. This indicates that potential transfer may be a more challenging task than initial condition transfer. Nonetheless, these results indicate that we are able to perform transfer learning from one potential to another and that our pre-trained base was, indeed, able to learn the underlying statistics of the system.

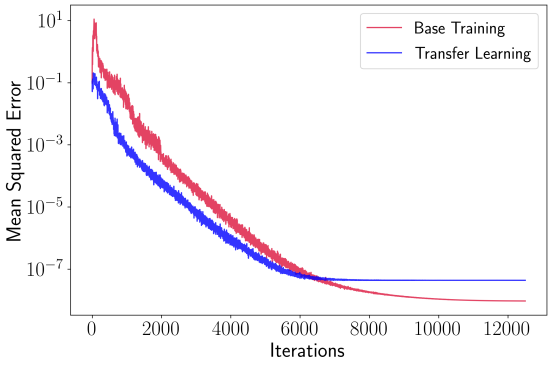


Figure 4.8: Mean squared errors vs. iteration during base training (red) and potential transfer (blue). For each training procedure, the mean squared error is calculated against the ground-truth numerical solutions and averaged over all initial conditions.

People will forget what you said, people will forget what you did, but people will never forget how you made them feel.

Maya Angelou

5

Discussion

The primary objective of this work was to improve the robustness and utility of DEQGAN, a method proposed by Randle et al. [45] that uses a generative adversarial network to solve differential equations in a fully unsupervised manner, thereby removing the need for a pre-specified loss function. While the original version of DEQGAN showed promising results and outperformed classical

neural networks in terms of predictive accuracy on a range of differential equations, is was comparatively unstable to train and sensitive to hyperparameters. For example, changing the model weight initializations could produce highly variable results.

We have addressed this issue using a variety of approaches. First, we proposed adding instance noise to the "real" and "fake" data samples, which encourages their distributions to overlap and improved convergence. Second, we employed a residual monitoring technique, which leverages the information contained within the LHS vector of residuals to terminate poor-performing runs early. Finally, we proposed a multi-head DEQGAN architecture, which allows us to train a base generator on multiple initial conditions at once. By leveraging a pre-trained base for transfer learning, we found that DEQGAN training not only became more stable and less computationally expensive, but often also improved accuracy in comparison to training DEQGAN from scratch.

Transfer learning enabled DEQGAN to learn solutions to new initial conditions and even different parameterizations of a given system using a single set of hyperparameters. In particular, the hyperparameters used to train the base generator also worked well on our transfer learning tasks. Nonetheless, those hyperparameters still required tuning to obtain, and using default hyperparameter values would likely have yielded significantly worse results.

The sensitivity of generative adversarial networks to hyperparameter settings is an open research problem and remains the biggest bottleneck of our approach. While DEQGAN consistently outperforms classical unsupervised neural networks that use L_2, L_1 and Huber loss functions, these classical networks are usually able to obtain accurate solutions with little to no hyperparameter tuning. Indeed, as we found in Chapter 3, the solution accuracies obtained by classical neural networks

showed significantly less variability across randomized trials than those obtained by DEQGAN. Further, we found that hyperparameter tuning the classical methods did not significantly improve their results.

Despite these limitations, we believe that the results presented in this thesis indicate that DEQGAN could offer advantages over other approaches to solving differential equations in real world scenarios. In particular, if the computation required to tune hyperparameters is available, DEQGAN is likely to yield solution accuracies that beat classical neural networks and that are competitive with popular numerical solvers. But unlike numerical methods, DEQGAN produces solutions that are in closed-form, meaning that we can obtain solutions over arbitrary grids with a single forward pass of a trained generator network.

At the same time, we note that solving differential equations with unsupervised neural networks is a relatively new line of research and lacks the well established theory enjoyed by numerical approaches. Importantly, neural networks, especially generative adversarial networks, provide no guarantees of training stability or convergence. In a real world scenario, therefore, they should always be used alongside numerical solvers, particularly if they are being deployed in safety critical applications in science and engineering.

6

Conclusion

In this thesis, we have explored various methods for improving the robustness of DEQGAN, a GAN-based method for solving differential equations in a fully unsupervised fashion. We found that adding instance noise to the "real" and "fake" data samples in a manner that depends on the generator and discriminator losses is an effective method for adaptively improving training conver-

gence. Further, we showed how the information contained within the equation residuals can be leveraged to detect and terminate poor-performing runs early. By conducting an ablation study, we demonstrated the added robustness offered by these approaches.

Presenting results on a suite of ODEs and PDEs, including the non-linear Burgers', Allen-Cahn, Hamilton, and modified Eintesin's gravity equations, we showed that these improvements enable DEQGAN to obtain reliable results on challenging problems and outperform classical unsupervised neural networks that use L_2, L_1 and Huber loss functions, often by multiple order of magnitude. These experiments also revealed that the loss function used to optimize classical neural networks can have a significant impact on the quality of the solution and highlighted the advantage of using DEQGAN to simply "learn the loss function" based on the equation residuals.

Despite these impressive results, they still required extensive hyperparameter tuning to attain, making DEQGAN more computationally espensive than alternative unsupervised neural networks that use feed-forward architectures. In light of this, we proposed a multi-head DEQGAN model that solves a given system for multiple initial conditions at once. This allowed us to obtain a pre-trained base generator that can be transferred to a new set of initial conditions or a different parameterization of the system. We found that transfer learning improved not only computational efficiency but also training stability and, in some cases, solution accuracy.

While neural networks generally remain more computationally expensive than numerical methods, our results point to the efficacy of transferring the inductive biases learned on one differential equation to a different parameterization of that equation, or potentially to new systems entirely. To make unsupervised neural networks more competitive with traditional methods, future work could

explore how transfer learning can be harnessed to efficiently compute solutions for entire classes of differential equations.

In addition, it may be fruitful to explore alternative methods for sampling points during training. While the experiments conducted in this thesis utilized only fixed-grid and noisy-grid sampling schemes, it may be more efficient to concentrate sampling in high residual areas of the problem domain. Some of these methods might be based on the gradient of the residuals, while others could adopt a probabilistic re-sampling approach.

More broadly, we advocate for future work on solving differential equations with deep learning methods to focus on efficiency. We hope that the ideas presented in this thesis inspire other researchers to develop neural network solvers that are more intelligent and place a smaller strain on computational resources.

A

Appendix

A.1 Training Details

In Tables A.1 – A.12, we specify the problem domains and hyperparameters used for the full suite of twelve differential equations that we have solved with DEQGAN. We used optimization algorithms from Ray-Tune [33] to tune hyperparameters.

Hyperparameter	Value
Num. Iterations	1200
Num. Grid Points	100
Sampling Method	Perturb
Grid Boundary	$[0, 10]$
Instance Noise	True
G Units	40
G Layers	2
D Units	20
D Layers	4
G Learning Rate	0.09465492
D Learning Rate	0.01221551
Learning Rate Decay	`StepLR`(`step_size` $= 3, \gamma = 0.95003024$)
G Optimizer	`Adam`($\beta_1 = 0.49137950, \beta_2 = 0.31989269$)
D Optimizer	`Adam`($\beta_1 = 0.54223041, \beta_2 = 0.26406118$)
G Activations	`tanh`
D Activations	`tanh`
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True

Table A.1: DEQGAN Training Details: Exponential Decay (EXP)

Hyperparameter	Value
Num. Iterations	12000
Num. Grid Points	400
Sampling Method	Perturb
Grid Boundary	$[0, 2\pi]$
Instance Noise	True
G Units	40
G Layers	3
D Units	50
D Layers	3
G Learning Rate	0.00596461
D Learning Rate	0.00048861
Learning Rate Decay	`StepLR`(`step_size` $= 19, \gamma = 0.97862221$)
G Optimizer	`Adam`($\beta_1 = 0.36317042, \beta_2 = 0.75224866$)
D Optimizer	`Adam`($\beta_1 = 0.58413382, \beta_2 = 0.45382877$)
G Activations	`tanh`
D Activations	`tanh`
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True

Table A.2: DEQGAN Training Details: Simple Harmonic Oscillator (SHO)

Hyperparameter	Value
Num. Iterations	12000
Num. Grid Points	400
Sampling Method	Perturb
Grid Boundary	$[0, 4\pi]$
Instance Noise	True
G Units	40
G Layers	4
D Units	20
D Layers	2
G Learning Rate	0.01064872
D Learning Rate	0.02104329
Learning Rate Decay	`StepLR`(`step_size` $= 15, \gamma = 0.98037812$)
G Optimizer	`Adam`($\beta_1 = 0.22544921, \beta_2 = 0.33148318$)
D Optimizer	`Adam`($\beta_1 = 0.36273349, \beta_2 = 0.55130697$)
G Activations	`tanh`
D Activations	`tanh`
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True

Table A.3: DEQGAN Training Details: Non-Linear Oscillator (NLO)

Hyperparameter	Value
Num. Iterations	70000
Num. Grid Points	800
Sampling Method	Perturb
Grid Boundary	$[0, 2\pi]$
Instance Noise	True
G Units	40
G Layers	5
D Units	40
D Layers	2
G Learning Rate	0.00421514
D Learning Rate	0.08228618
Learning Rate Decay	`StepLR`(`step_size` $= 16, \gamma = 0.99756482$)
G Optimizer	`Adam`($\beta_1 = 0.60325852, \beta_2 = 0.61447363$)
D Optimizer	`Adam`($\beta_1 = 0.41218949, \beta_2 = 0.11003992$)
G Activations	`tanh`
D Activations	`tanh`
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True

Table A.4: DEQGAN Training Details: Coupled Oscillators (COO)

Hyperparameter	Value
Num. Iterations	20000
Num. Grid Points	800
Sampling Method	Perturb
Grid Boundary	$[0, 10]$
Instance Noise	True
G Units	50
G Layers	4
D Units	50
D Layers	4
G Learning Rate	0.00629315
D Learning Rate	0.01239617
Learning Rate Decay	`StepLR`(`step_size` $= 11, \gamma = 0.99275622$)
G Optimizer	`Adam`($\beta_1 = 0.27854976, \beta_2 = 0.77766866$)
D Optimizer	`Adam`($\beta_1 = 0.01864329, \beta_2 = 0.908877$)
G Activations	`tanh`
D Activations	`tanh`
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True

Table A.5: DEQGAN Training Details: SIR Disease Model (SIR)

Hyperparameter	Value
Num. Iterations	12500
Num. Grid Points	400
Sampling Method	Perturb
Grid Boundary	$[0, 1]$
Instance Noise	True
G Units	40
G Layers	5
D Units	50
D Layers	2
G Learning Rate	0.01791762
D Learning Rate	0.01959963
Learning Rate Decay	`StepLR`(`step_size` $= 13, \gamma = 0.98551429$)
G Optimizer	`Adam`($\beta_1 = 0.25259308, \beta_2 = 0.93144190$)
D Optimizer	`Adam`($\beta_1 = 0.10584021, \beta_2 = 0.86945859$)
G Activations	`tanh`
D Activations	`tanh`
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True

Table A.6: DEQGAN Training Details: Hamilton Equations (HAM)

Hyperparameter	Value
Num. Iterations	50000
Num. Grid Points	1000
Sampling Method	Perturb
Grid Boundary	$[0, 10]$
Instance Noise	True
G Units	40
G Layers	4
D Units	30
D Layers	2
G Learning Rate	0.01177194
D Learning Rate	0.00662024
Learning Rate Decay	`StepLR`(`step_size` $= 17, \gamma = 0.99636960$)
G Optimizer	`Adam`($\beta_1 = 0.20296515, \beta_2 = 0.97508494$)
D Optimizer	`Adam`($\beta_1 = 0.15440781, \beta_2 = 0.79705852$)
G Activations	`tanh`
D Activations	`tanh`
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True

Table A.7: DEQGAN Training Details: Modified Einstein's Gravity Equations (EIN)

Hyperparameter	Value
Num. Iterations	3000
Num. Grid Points	32×32
Sampling Method	Perturb
Grid Boundary	$[0, 1] \times [0, 1]$
Instance Noise	True
G Units	50
G Layers	4
D Units	30
D Layers	2
G Learning Rate	0.01908663
D Learning Rate	0.02154824
Learning Rate Decay	`StepLR`(`step_size` $= 3, \gamma = 0.95727502$)
G Optimizer	`Adam`($\beta_1 = 0.13931808, \beta_2 = 0.36966975$)
D Optimizer	`Adam`($\beta_1 = 0.74574617, \beta_2 = 0.7590883$)
G Activations	`tanh`
D Activations	`tanh`
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True

Table A.8: DEQGAN Training Details: Poisson Equation (POS)

Hyperparameter	Value
Num. Iterations	2000
Num. Grid Points	32×32
Sampling Method	Perturb
Grid Boundary	$[0, 1] \times [0, 0.2]$
Instance Noise	True
G Units	40
G Layers	4
D Units	30
D Layers	2
G Learning Rate	0.01086166
D Learning Rate	0.00195769
Learning Rate Decay	`StepLR`(`step_size` $= 10, \gamma = 0.95003910$)
G Optimizer	`Adam`($\beta_1 = 0.23067650, \beta_2 = 0.65763986$)
D Optimizer	`Adam`($\beta_1 = 0.12032453, \beta_2 = 0.25102168$)
G Activations	`tanh`
D Activations	`tanh`
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True

Table A.9: DEQGAN Training Details: Heat Equation (HEA)

Hyperparameter	Value
Num. Iterations	5000
Num. Grid Points	32×32
Sampling Method	Perturb
Grid Boundary	$[0, 1] \times [0, 1]$
Instance Noise	True
G Units	50
G Layers	4
D Units	50
D Layers	2
G Learning Rate	0.01212438
D Learning Rate	0.08844106
Learning Rate Decay	`StepLR`(`step_size` $= 18, \gamma = 0.95336880$)
G Optimizer	`Adam`($\beta_1 = 0.29551987, \beta_2 = 0.35822444$)
D Optimizer	`Adam`($\beta_1 = 0.57517950, \beta_2 = 0.13306347$)
G Activations	`tanh`
D Activations	`tanh`
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True

Table A.10: DEQGAN Training Details: Wave Equation (WAV)

Hyperparameter	Value
Num. Iterations	3000
Num. Grid Points	64×64
Sampling Method	Perturb
Grid Boundary	$[-5, 5] \times [0, 2.5]$
Instance Noise	True
G Units	50
G Layers	3
D Units	20
D Layers	5
G Learning Rate	0.01274294
D Learning Rate	0.00542646
Learning Rate Decay	`StepLR`(`step_size` $= 20, \gamma = 0.95484413$)
G Optimizer	`Adam`($\beta_1 = 0.18525066, \beta_2 = 0.59412178$)
D Optimizer	`Adam`($\beta_1 = 0.09374523, \beta_2 = 0.18467156$)
G Activations	`tanh`
D Activations	`tanh`
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True

Table A.11: DEQGAN Training Details: Burgers' Equation (BUR)

Hyperparameter	Value
Num. Iterations	10000
Num. Grid Points	64×64
Sampling Method	Perturb
Grid Boundary	$[0, 2\pi] \times [0, 5]$
Instance Noise	True
G Units	50
G Layers	2
D Units	30
D Layers	2
G Learning Rate	0.02075779
D Learning Rate	0.01349295
Learning Rate Decay	`StepLR(step_size` $= 15, \gamma = 0.98398755$)
G Optimizer	`Adam`$(\beta_1 = 0.43650750, \beta_2 = 0.91051010)$
D Optimizer	`Adam`$(\beta_1 = 0.48427734, \beta_2 = 0.29753110)$
LHS Reweighing	$\lambda = 1.20162002$
G Activations	`tanh`
D Activations	`tanh`
GAN Formulation	Cross-Entropy
GAN Regularization	Spectral Normalization
G Skip Connections	True
D Skip Connections	True

Table A.12: DEQGAN Training Details: Allen-Cahn Equation (ACA)

A.2 Classical Neural Network Hyperparameter Tuning

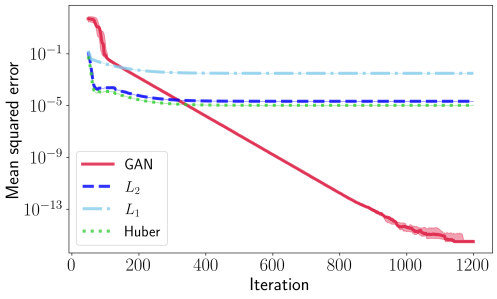
Table A.13 presents the results obtained by tuning hyperparameters for the classical unsupervised neural network method with L_1, L_2 and Huber loss functions.

	Mean Squared Error				
Key	L_1	L_2	Huber	DEQGAN	Traditional
EXP	$1 \cdot 10^{-4}$	$4 \cdot 10^{-8}$	$2 \cdot 10^{-8}$	$3 \cdot 10^{-16}$	$2 \cdot 10^{-14}$ (RK4)
SHO	$1 \cdot 10^{-5}$	$1 \cdot 10^{-9}$	$5 \cdot 10^{-10}$	$4 \cdot 10^{-13}$	$1 \cdot 10^{-11}$ (RK4)
NLO	$1 \cdot 10^{-4}$	$3 \cdot 10^{-10}$	$1 \cdot 10^{-10}$	$1 \cdot 10^{-12}$	$4 \cdot 10^{-11}$ (RK4)
COO	$5 \cdot 10^{-1}$	$2 \cdot 10^{-7}$	$3 \cdot 10^{-7}$	$1 \cdot 10^{-8}$	$2 \cdot 10^{-9}$ (RK4)
SIR	$9 \cdot 10^{-6}$	$1 \cdot 10^{-10}$	$1 \cdot 10^{-10}$	$1 \cdot 10^{-10}$	$5 \cdot 10^{-13}$ (RK4)
HAM	$4 \cdot 10^{-5}$	$1 \cdot 10^{-8}$	$6 \cdot 10^{-9}$	$1 \cdot 10^{-10}$	$7 \cdot 10^{-14}$ (RK4)
EIN	$5 \cdot 10^{-2}$	$2 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	$4 \cdot 10^{-4}$	$4 \cdot 10^{-7}$ (RK4)
POS	$9 \cdot 10^{-6}$	$1 \cdot 10^{-10}$	$1 \cdot 10^{-10}$	$4 \cdot 10^{-13}$	$3 \cdot 10^{-10}$ (FD)
HEA	$1 \cdot 10^{-4}$	$4 \cdot 10^{-8}$	$2 \cdot 10^{-8}$	$6 \cdot 10^{-10}$	$4 \cdot 10^{-7}$ (FD)
WAV	$4 \cdot 10^{-4}$	$6 \cdot 10^{-7}$	$2 \cdot 10^{-7}$	$1 \cdot 10^{-8}$	$7 \cdot 10^{-5}$ (FD)
BUR	$1 \cdot 10^{-3}$	$1 \cdot 10^{-4}$	$9 \cdot 10^{-5}$	$4 \cdot 10^{-6}$	$1 \cdot 10^{-3}$ (FD)
ACA	$5 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	$3 \cdot 10^{-3}$	$5 \cdot 10^{-3}$	$2 \cdot 10^{-4}$ (FD)

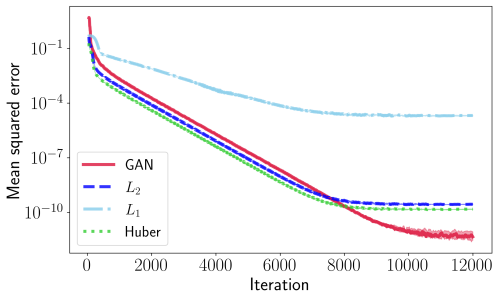
Table A.13: Experimental Results With Classical Hyperparameter Tuning

A.3 DEQGAN vs. Classical Neural Networks: Previous Experiments

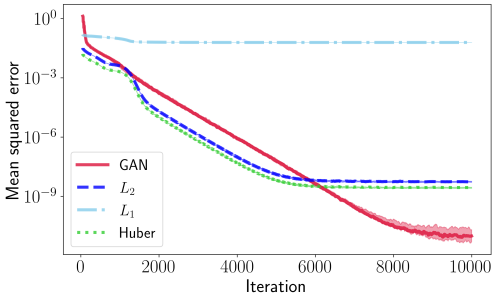
Figure A.1 compares the DEQGAN training results with those obtained by the classical neural network methods on the six differential equations previously studied by Randle et al. [45].



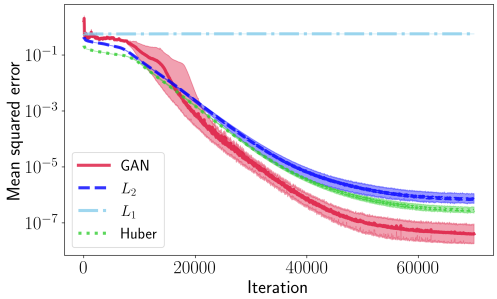
(a) Exponential Decay (EXP)



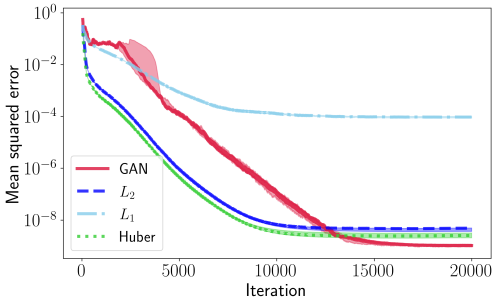
(b) Simple Harmonic Oscillator (SHO)



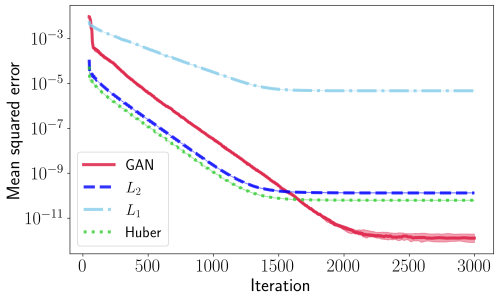
(c) Non-Linear Oscillator (NLO)



(d) Coupled Oscillators (COO)



(e) SIR Disease Model (SIR)



(f) Poisson Equation (POS)

Figure A.1: Mean squared errors vs. iteration for DEQGAN, L_2, L_1 and Huber loss for various equations. We perform ten randomized trials and plot the median (bold) and $(25, 75)$ percentile range (shaded). We smooth the values using a simple moving average with window size 50.

References

[1] Alencastre-Miranda, M., Johnson, R. M., & Krebs, H. I. (2021). Convolutional neural networks and transfer learning for quality inspection of different sugarcane varieties. *IEEE Transactions on Industrial Informatics*, 17(2), 787–794.

[2] Arjovsky, M. & Bottou, L. (2017). Towards principled methods for training generative adversarial networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*: OpenReview.net.

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

[4] Bertalan, T., Dietrich, F., Mezić , I., & Kevrekidis, I. G. (2019). On learning hamiltonian systems from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(12), 121107.

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

[6] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., & Amodei, D. (2020). Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems*, volume 33 (pp. 1877–1901).: Curran Associates, Inc.

[7] Brunton, S. L. & Kutz, J. N. (2019). *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press.

[8] Chantada, A. T., Landau, S. J., Protopapas, P., Scóccola, C. G., & Garraffo, C. (2022). Cosmological informed neural networks to solve the background dynamics of the universe.

[9] Chen, F., Sondak, D., Protopapas, P., Mattheakis, M., Liu, S., Agarwal, D., & Di Giovanni, M. (2020). Neurodiffeq: A python package for solving differential equations with neural networks. *Journal of Open Source Software*, 5(46), 1931.

[10] Choudhary, A., Lindner, J., Holliday, E., Miller, S., Sinha, S., & Ditto, W. (2020). Physics-enhanced neural networks learn order and chaos. *Physical Review E*, 101.

[11] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., & Fiedel, N. (2022). Palm: Scaling language modeling with pathways.

[12] Desai, S., Mattheakis, M., Joy, H., Protopapas, P., & Roberts, S. (2021). One-shot transfer learning of physics-informed neural networks.

[13] Dissanayake, M. & Phan-Thien, N. (1994). Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3), 195–201.

[14] Flamant, C., Protopapas, P., & Sondak, D. (2020). Solving differential equations using neural network solution bundles.

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

[16] Greydanus, S., Dzamba, M., & Yosinski, J. (2019). Hamiltonian neural networks.

[17] Grohs, P., Hornung, F., Jentzen, A., & von Wurstemberger, P. (2018). A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations.

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

[19] Guo, Y., Shi, H., Kumar, A., Grauman, K., Rosing, T., & Feris, R. (2019). Spottune: Transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[20] Hagge, T., Stinis, P., Yeung, E., & Tartakovsky, A. M. (2017). Solving differential equations with unknown constitutive relations as recurrent neural networks.

[21] Han, J., Jentzen, A., & E, W. (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34), 8505–8510.

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

[23] Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., & Gelly, S. (2019). Parameter-efficient transfer learning for NLP. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research* (pp. 2790–2799).: PMLR.

[24] Hubble, E. (1929). A relation between distance and radial velocity among extra-galactic nebulae. *Proceedings of the National Academy of Sciences*, 15(3), 168–173.

[25] Karnewar, A., Wang, O., & Iyengar, R. S. (2019). MSG-GAN: multi-scale gradient GAN for stable image synthesis. *CoRR*, abs/1903.06048.

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

[27] Kodali, N., Abernethy, J. D., Hays, J., & Kira, Z. (2017). How to train your DRAGAN. *CoRR*, abs/1705.07215.

[28] Lacoste, A., Luccioni, A., Schmidt, V., & Dandres, T. (2019). Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*.

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

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

[31] 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.

[32] Li, X., Grandvalet, Y., & Davoine, F. (2018). Explicit inductive bias for transfer learning with convolutional networks.

[33] 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.

[34] Maddox, W. J., Tang, S., Moreno, P. G., Wilson, A. G., & Damianou, A. (2021). Fast adaptation with linearized neural networks.

[35] Malte, A. & Ratadiya, P. (2019). Evolution of transfer learning in natural language processing.

[36] Mattheakis, M., Joy, H., & Protopapas, P. (2021). Unsupervised reservoir computing for solving ordinary differential equations.

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

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

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

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

[41] Paticchio, A., Scarlatti, T., Mattheakis, M., Protopapas, P., & Brambilla, M. (2020). Semi-supervised neural networks solve an inverse problem for modeling covid-19 spread.

[42] Piscopo, M. L., Spannowsky, M., & Waite, P. (2019). Solving differential equations with neural networks: Applications to the calculation of cosmological phase transitions. *Phys. Rev. D*, 100, 016002.

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

[44] Raissi, M., Perdikaris, P., & Karniadakis, G. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378, 686 – 707.

[45] Randle, D., Protopapas, P., & Sondak, D. (2020). Unsupervised learning of solutions to differential equations with generative adversarial networks.

[46] Riess, A. G., Filippenko, A. V., Challis, P., Clocchiatti, A., Diercks, A., Garnavich, P. M., Gilliland, R. L., Hogan, C. J., Jha, S., Kirshner, R. P., Leibundgut, B., Phillips, M. M., Reiss, D., Schmidt, B. P., Schommer, R. A., Smith, R. C., Spyromilio, J., Stubbs, C., Suntzeff, N. B., & Tonry, J. (1998). Observational evidence from supernovae for an accelerating universe and a cosmological constant. *The Astronomical Journal*, 116(3), 1009–1038.

[47] Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training gans. *CoRR*, abs/1606.03498.

[48] Shin, H.-C., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., & Summers, R. M. (2016). Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE Transactions on Medical Imaging*, 35(5), 1285–1298.

[49] 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.

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

[51] Sønderby, C. K., Caballero, J., Theis, L., Shi, W., & Huszár, F. (2016). Amortised MAP inference for image super-resolution. *CoRR*, abs/1610.04490.

[52] Stevens, B. & Colonius, T. (2020). Finitenet: A fully convolutional lstm network architecture for time-dependent partial differential equations.

[53] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, İ., Feng, Y., Moore, E. W., Vand erPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., & Contributors, S. . . (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17, 261–272.

[54] You, K., Kou, Z., Long, M., & Wang, J. (2020). Co-tuning for transfer learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems*, volume 33 (pp. 17236–17246).: Curran Associates, Inc.

[55] Zeng, S., Zhang, Z., & Zou, Q. (2022). Adaptive deep neural networks methods for high-dimensional partial differential equations. *Journal of Computational Physics*, (pp. 111232).