Unsupervised Learning of Solutions to Differential Equations with Generative Adversarial Networks

Dylan Randle
IACS
Harvard University
Cambridge, MA 02138

Pavlos Protopapas
IACS
Harvard University
Cambridge, MA 02138

David Sondak
IACS
Harvard University
Cambridge, MA 02138

Abstract

Solutions to differential equations are of significant scientific and engineering relevance. Recently, there has been a growing interest in solving differential equations with neural networks. This work develops a novel method for solving differential equations with unsupervised neural networks that applies Generative Adversarial Networks (GANs) to *learn the loss function* for optimizing the neural network. We present empirical results showing that our method, which we call Differential Equation GAN (DEQGAN), can obtain multiple orders of magnitude lower mean squared errors than an alternative unsupervised neural network method based on (squared) L_2, L_1, and Huber loss functions. Moreover, we show that DEQGAN achieves solution accuracy that is competitive with traditional numerical methods. Finally, we analyze the stability of our approach and find it to be sensitive to the selection of hyperparameters, which we provide in the Appendix.[1]

1 Introduction

In fields such as physics, chemistry, biology, engineering, and economics, differential equations are applied to the modeling of important and complex phenomena. While traditional methods for solving differential equations perform well, and the theory for their stability and convergence are well established, the recent success of deep learning techniques [19, 38, 2, 39, 30, 5, 8, 34] has inspired researchers to apply neural networks to solving differential equations [33, 10, 31, 26, 36, 27, 11, 32, 35].

Applying neural networks to solving differential equations can provide a range of benefits over traditional methods. By removing a reliance on finely crafted grids which suffer from the "curse of dimensionality", neural networks can be more effective than traditional solvers in high-dimensional settings [11, 32, 35]. Furthermore, recent work has shown that neural network solutions can be more accurate in obeying certain physical constraints, such as conservation of energy [27, 33]. Neural networks can also provide provide a more accurate interpolation scheme [21]. Finally, forward passes of neural networks are embarrassingly data-parallel, even in difficult-to-parallelize temporal dimensions, and can readily leverage parallel computing architectures.

Interest in research on solving differential equations with unsupervised neural networks is growing. However, due to a lack (to the best of our knowledge) of theoretical justification for a particular choice of loss function, we propose "learning" the loss function with a Generative Adversarial Network (GAN) [7]. For data following a known noise model, there is clear theoretical justification, based on the maximum likelihood principle, for fitting models with particular loss functions. For example, in

[1]Code available at `https://github.com/dylanrandle/denn`. Please address any electronic correspondence to `dylanrandle@alumni.harvard.edu`.

the case of a Gaussian noise model

$$y = x + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2), \tag{1}$$

the maximum likelihood estimate of the model parameters minimizes the squared error (L_2 norm) loss function. In the case of deterministic differential equations, however, there is no noise model and we lack formal justification for a particular choice of loss function among multiple options.

To circumvent this problem, we propose GANs for solving differential equations in a fully unsupervised manner. The discriminator model of a GAN can be thought of as learning the loss function used for optimizing the generator. Moreover, GANs have been shown to excel in scenarios where classic loss functions, such as the mean squared error, struggle due to their inability to capture complex spatio-temporal dependencies [24, 25, 17].

Our main contribution is a novel method, which we call Differential Equation GAN (DEQGAN), for formulating the task of solving differential equations in a *fully unsupervised* manner as a GAN training problem. DEQGAN works by separating the differential equation into left-hand side (LHS) and right-hand side (RHS), then training the generator to produce a LHS that is indistinguishable to the discriminator from the RHS. Experimental results show that our method produces solutions which obtain multiple orders of magnitude lower mean squared errors (computed from known analytic or numerical solutions) than a comparable unsupervised neural network method with (squared) L_2, L_1, and Huber loss functions. Moreover, DEQGAN achieves solution accuracy that is competitive with traditional fourth-order Runge-Kutta and second-order finite difference methods.

2 Related Work

Dissanayake & Phan-Thien [6] were one of the first to develop a method for solving differential equations with neural networks. They showed that a neural network-based method could solve differential equations when transformed to unconstrained optimization problems. Lagaris et al. [21] extended this work by introducing analytical adjustments to the neural network output that exactly satisfied initial and boundary conditions. They showed that their method achieved lower interpolation error than the finite element method, while maintaining equal error on a fixed mesh. The authors expanded this work to consider arbitrarily-shaped domains in higher dimensions [23], and applied neural networks to quantum mechanics [22].

Recent work has solved high-dimensional partial differential equations (PDEs) with neural networks in place of basis functions [35] and by reformulating PDEs as backward stochastic differential equations [11]. To reduce the need to re-learn known physics, Mattheakis et al. [26] embedded physical symmetries into the structure of neural networks, and Raissi et al. [33] regularized neural networks according to physical models described by nonlinear PDEs, leading to improved solution accuracy and training convergence over physics-agnostic counterparts. Leveraging recent advances in deep learning, Hagge et al. [10] developed a supervised recurrent neural network method that uses measurement data to solve ordinary differential equations with unknown functional forms, and Stevens & Colonius [36] presented a fully convolutional LSTM network that augments traditional finite difference/finite volume methods used to solve PDEs. Kumar & Yadav [20] presented a survey of neural network and radial basis function methods for solving differential equations.

In parallel, Goodfellow et al. [7] introduced the idea of learning generative models with neural networks and an adversarial training algorithm, called Generative Adversarial Networks (GANs). To solve issues of GAN training instability, Arjovsky et al. [1] introduced a formulation of GANs based on the Wasserstein distance, and Gulrajani et al. [9] added a gradient penalty to approximately enforce a Lipschitz constraint on the discriminator. Miyato et al. [29] introduced an alternative method for enforcing the Lipschitz constraint with a spectral normalization technique that outperforms the former method on some problems.

Further work has applied GANs to differential equations with solution data used for supervision. Yang et al. [42] apply GANs to stochastic differential equations by using "snapshots" of ground-truth data for semi-supervised training. A project by students at Stanford [37] employed GANs to perform "turbulence enrichment" of solution data in a manner akin to that of super-resolution for images proposed by Ledig et al. [25]. Our work distinguishes itself from other GAN-based approaches for solving differential equations by being *fully unsupervised*, and removing the dependence on using supervised training data (i.e. solutions of the equation).

3 Background

3.1 Unsupervised Neural Networks for Differential Equations

Early work by Dissanayake & Phan-Thien [6] proposed solving differential equations in an unsupervised manner with neural networks. Their paper considers general differential equations of the form

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

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

$$\min_{\theta} \sum_{t \in \mathcal{D}} F(t, \Psi_\theta(t), \Delta\Psi_\theta(t), \Delta^2\Psi_\theta(t), \ldots)^2 \tag{3}$$

where Ψ_θ is a neural network parameterized by θ, $\mathcal{D}$ is the domain of the problem, and we compute derivatives with automatic differentiation. This allows us to use backpropagation [13] to train the parameters of the neural network to satisfy the differential equation. Note that this formalism can be trivially extended to handle spatial domains and multidimensional problems, which we do in our experiments and describe in the Appendix.

3.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [7] are a type generative model that use two neural networks to induce a generative distribution $p(x)$ of the data by formulating the inference problem as a two-player, zero-sum game.

The generative model first samples a latent variable $z \sim \mathcal{N}(0, 1)$, which is used as input into the generator G (e.g. a neural network). A discriminator D is trained to classify whether its input was sampled from the generator (i.e. “fake") or from a reference data set (i.e. “real").

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

$$\min_{G} \max_{D} V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[1 - \log D(G(z))] \tag{4}$$

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

3.2.1 Two Time-Scale Update Rule

Heusel et al. [14] proposed the two time-scale update rule (TTUR) for training GANs, a method in which the discriminator and generator are trained with separate learning rates. They showed that their method led to improved performance and proved that, in some cases, TTUR ensures convergence to a stable local Nash equilibrium. One intuition for TTUR comes from the potentially different loss surface curvatures of the discriminator and generator. Allowing learning rates to be tuned to a particular loss surface can enable more efficient gradient-based optimization. We make use of TTUR throughout this paper as an instrumental lever when tuning GANs to reach desired performance.

3.2.2 Spectral Normalization

Proposed by Miyato et al. [29], Spectrally Normalized GAN (SN-GAN) is a method for controlling exploding discriminator gradients when optimizing Equation 4 that leverages a novel weight normalization technique.

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

$$W_{SN,i} = \frac{W_i}{\sigma(W_i)}, \tag{5}$$

where

$$\sigma(W_i) = \max_{\|h_i\|_2 \leq 1} \|W_i h_i\|_2 \tag{6}$$

and h_i denotes the input to layer i. The authors prove that this normalization technique bounds the Lipschitz constant of the discriminator above by 1, thus strictly enforcing the 1-Lipshcitz constraint on the discriminator. In our experiments, adopting the SN-GAN formulation leads to even better performance than WGAN-GP [1, 9].

3.3 Guaranteeing Initial & Boundary Conditions

Lagaris et al. [21] showed that it is possible to exactly satisfy initial and boundary conditions by adjusting the output of the neural network. For example, consider adjusting the neural network solution $\Psi(t)$ to satisfy the initial condition $\Psi(0) = x_0$. We can apply the transformation

$$\Psi(t)' = x_0 + t\Psi(t) \tag{7}$$

which exactly satisfies the condition. Mattheakis et al. [26] proposed an augmented transformation

$$\Psi(t)' = \Phi\left(\Psi(t)\right) = x_0 + \left(1 - e^{-(t-t_0)}\right)\Psi(t) \tag{8}$$

that further improved training convergence. Intuitively, Equation 8 adjusts the output of the neural network $\Psi(t)$ to be exactly x_0 when $t = t_0$, and decays this constraint exponentially in t.

3.4 Residual Connections

He et al. [12] showed that adding residual connections improved training of deep neural networks. We employ residual connections to our deep networks as they allow gradients to more easily flow through the models and thereby reduce numerical instability. Residual connections augment a typical activation with the identity operation

$$y = \mathcal{F}(x, W_i) + x \tag{9}$$

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

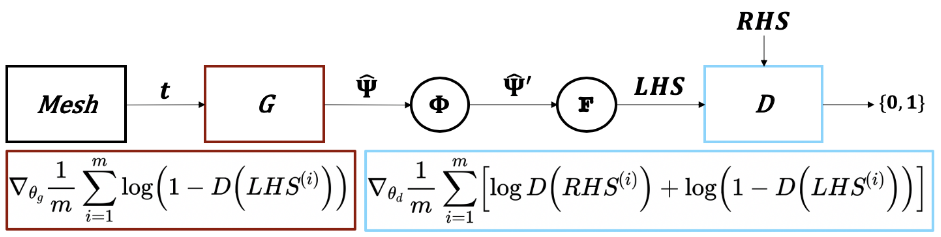
4 Differential Equation GAN

Here we present our method, Differential Equation GAN (DEQGAN), which trains a GAN to solve differential equations in a *fully unsupervised* manner. To do this, we rearrange the differential equation such that the left-hand side (LHS) contains all of the terms which depend on the generator (e.g. $\hat{\Psi}$, $\Delta\hat{\Psi}$, $\Delta^2\hat{\Psi}$, etc.), and the right-hand side (RHS) contains only constants (e.g. zero).

From here we sample points from the domain $t \sim \mathcal{D}$ and use them as input to a generator $G(t)$, which produces candidate solutions $\hat{\Psi}$. We adjust $\hat{\Psi}$ for initial or boundary conditions according to Equation 8. Then we construct the LHS from the differential equation F using automatic differentiation

$$LHS = F\left(t, \hat{\Psi}(t), \Delta\hat{\Psi}(t), \Delta^2\hat{\Psi}(t)\right) \tag{10}$$

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



Mesh
t
G
$\hat{\Psi}$
Φ
$\hat{\Psi}'$
F
LHS
RHS
D
{0, 1}
$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(LHS^{(i)}\right)\right)$
$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(RHS^{(i)}\right) + \log\left(1 - D\left(LHS^{(i)}\right)\right)\right]$

Figure 1: Schematic representation of DEQGAN. We perturb points t from the mesh and input them to a generator G, which produces candidate solutions $\hat{\Psi}$. Then we analytically adjust these solutions according to Φ and apply automatic differentiation to construct LHS from the differential equation F. RHS and LHS are passed to a discriminator D, which is trained to classify them as “real" and “fake" respectively.

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

$$\eta_G = \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(LHS^{(i)}\right)\right), \tag{11}$$

$$\eta_D = \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(RHS^{(i)}\right) + \log\left(1 - D\left(LHS^{(i)}\right)\right)\right] \tag{12}$$

where $LHS^{(i)}$ is the output of $G\left(t^{(i)}\right)$ after adjusting for initial or boundary conditions and constructing the LHS from F. Note that we perform stochastic gradient *descent* for G (gradient steps $\propto -\eta_G$), and stochastic gradient *ascent* for D (gradient steps $\propto \eta_D$). We provide a schematic representation of DEQGAN in Figure 1 and detail the training steps in Algorithm 1.

Algorithm 1 DEQGAN

```
**Input:** Differential equation $F$, generator $G(\cdot;\theta_g)$, discriminator $D(\cdot;\theta_d)$, mesh $t$ of $m$ points with spacing $\Delta t$, perturbation precision $\tau$, analytic adjustment function $\Phi$, total steps $N$, learning rates $\alpha_G, \alpha_D$, Adam optimizer [18] parameters $\beta_{G1}, \beta_{G2}, \beta_{D1}, \beta_{D2}$
for $i = 1$ to $N$ do
    for $j = 1$ to $m$ do
        Perturb $j$-th point in mesh $t_s^{(j)} = t^{(j)} + \epsilon, \epsilon \sim \mathcal{N}(0, \frac{\Delta t}{\tau})$
        Forward pass $\hat{\Psi} = G(t_s^{(j)})$
        Analytic adjustment $\hat{\Psi}' = \Phi(\hat{\Psi})$ (Equation 8)
        Compute $LHS^{(j)} = F(t, \hat{\Psi}', \nabla\hat{\Psi}', \nabla^2\hat{\Psi}')$, set $RHS^{(j)} = 0$
    end for
    Compute gradients $\eta_G, \eta_D$ (Equation 11 & 12)
    Update generator $\theta_g \leftarrow \texttt{Adam}(\theta_g, -\eta_G, \alpha_G, \beta_{G1}, \beta_{G2})$
    Update discriminator $\theta_d \leftarrow \texttt{Adam}(\theta_d, \eta_D, \alpha_D, \beta_{D1}, \beta_{D2})$
end for
**Output:** $G$
```

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

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

Table 1: Summary of Experiments

Key	Equation	Class	Order	Linear	System
EXP	$\dot{x}(t) + x(t) = 0$	ODE	1st	Yes	No
SHO	$\ddot{x}(t) + x(t) = 0$	ODE	2nd	Yes	No
NLO	$\ddot{x}(t) + 2\beta\dot{x}(t) + \omega^2 x(t) + \phi x(t)^2 + \epsilon x(t)^3 = 0$	ODE	2nd	No	No
NAS	$\begin{cases} \dot{x}(t) = -ty \\ \dot{y}(t) = tx \end{cases}$	ODE	1st	Yes	Yes
SIR	$\begin{cases} \dot{S}(t) & = -\beta I(t)S(t)/N \\ \dot{I}(t) & = \beta I(t)S(t)/N - \gamma I(t) \\ \dot{R}(t) & = \gamma I(t) \end{cases}$	ODE	1st	No	Yes
POS	$u_{xx} + u_{yy} = 2x(y-1)(y-2x+xy+2)e^{x-y}$	PDE	2nd	Yes	No

Table 2: Experimental Results

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

Our intuition for this is that if RHS depends on the outputs of the generator, the “real" data distribution $p_{\text{data}}(x)$ (from Equation 4) changes as the generator weights are updated throughout training. If the distribution $p_{\text{data}}(x)$ is constantly changing, the discriminator will not have a reliable signal for learning to classify “real" from “fake", which violates a core assumption of traditional GANs. By setting $RHS = 0$, we resolve the problem by effectively setting the “real" distribution to be the fixed Dirac delta function $p_{\text{data}}(x) = \delta(0)$. For the examples in this paper, we move all terms of the differential equation to LHS and set $RHS = 0$.

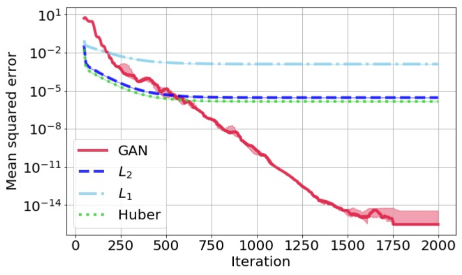
5 Experiments

We conducted experiments on several differential equations of increasing complexity, comparing DEQGAN to an alternative unsupervised neural network method using squared L_2 (i.e. mean squared error[2]), L_1, and Huber [16] loss functions. We also report results obtained by the traditional fourth-order Runge-Kutta (RK4) and second-order finite differences (FD) methods for initial and boundary value problems, respectively. A detailed description of each experiment including exact problem specifications, hyperparameters, and a comparison of loss functions used is provided in the Appendix.

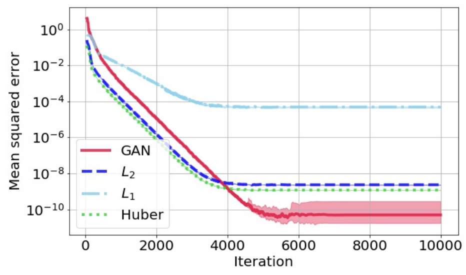
We report the mean squared errors of the solutions produced by each method, computed from known solutions obtained either analytically or with high-quality numerical solvers [40], but we do not use these solution data for training. We add residual connections between neighboring layers of all models, and apply spectral normalization to the discriminator in all experiments. Results are obtained with hyperparameters tuned for DEQGAN. In the Appendix, we tune each alternative method for comparison but do not observe a meaningful difference.

We note that deterministic differential equations do not exhibit aleatoric uncertainty and that all errors we observe are therefore epistemic. Given that neural networks are, theoretically, universal function approximators [15], one may initially expect to obtain arbitrarily low error. The reason we do not observe this is, first, that our models are finite-width and may lack representational capacity and,

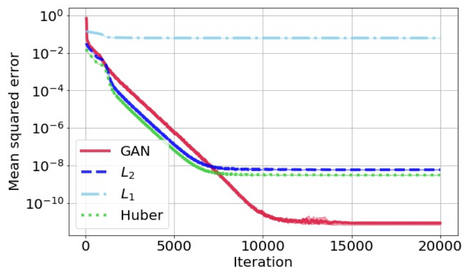
[2]We use the term L_2 to avoid conflating the *loss function* being used, which is the mean squared error on the unsupervised problem of minimizing the differential equation residuals, with the final *evaluation metric*, which is the mean squared error of the predicted solution computed against the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



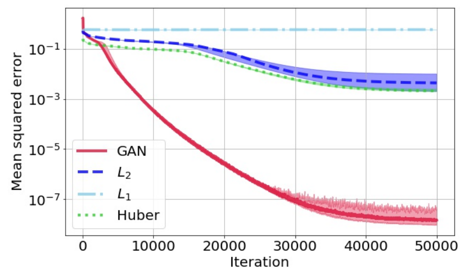
(a) Exponential Decay (EXP)



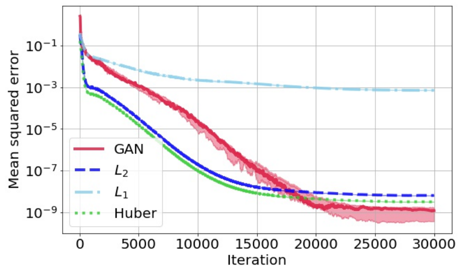
(b) Simple Harmonic Oscillator (SHO)



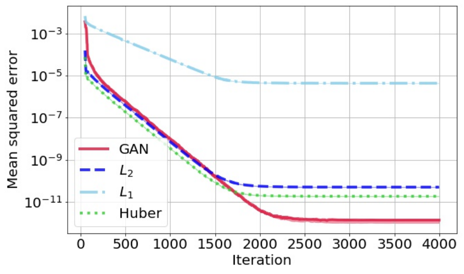
(c) Nonlinear Oscillator (NLO)



(d) Non-Autonomous System (NAS)



(e) SIR Disease Model (SIR)



(f) Poisson Equation (POS)

Figure 2: Mean squared errors vs. iteration for DEQGAN, L_2, L_1, and Huber loss for various equations. We perform five 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.

second, that optimizing the objective in Equation 4 with stochastic gradient descent is unlikely to reach globally optimal solutions and is more likely to converge to local optima.

Table 1 summarizes the equations we study in our experiments, and Table 2 reports the lowest mean squared errors obtained across five trials for each method. We see that DEQGAN obtains multiple orders of magnitude lower mean squared errors than the alternative unsupervised neural network method with L_1, L_2, and Huber loss functions across the differential equations studied. Moreover, DEQGAN achieves accuracy that is competitive with the traditional RK4 and FD numerical methods.

Figure 2 plots mean squared error against training iteration for DEQGAN and the alternative neural network method with L_1, L_2 and Huber loss functions. We observe that DEQGAN converges to lower mean squared errors than the alternative unsupervised neural network method, often by multiple orders of magnitude, across the equations studied. We note, however, that the mean squared error curves of DEQGAN are less smooth and exhibit greater variability than the normal unsupervised neural network method, an issue we study in the following section.

6 Stability of DEQGAN Training

A point that we have not addressed is the instability of the DEQGAN training algorithm. The instability of GANs is not a new problem and much work has been dedicated to improving the stability and convergence of GANs [1, 9, 3, 28, 29]. In our experiments we find that the initial weights of the generator and discriminator can have a substantial impact on the final performance of DEQGAN. The solution that we adopt is to fix the initial model weights when tuning hyperparameters for DEQGAN and to keep the same weight initialization thereafter, which appears to significantly reduce the problem of instability.

To illustrate the relationship between performance, hyperparameters, and initial model weights, Figure 3 plots the results of 500 DEQGAN experiments for the exponential decay equation. For each experiment, we uniformly at random select model weight initialization random seeds as integers from the range $[0, 9]$, as well as separate learning rates for the discriminator and generator in the range $[10^{-6}, 10^{-2}]$. We then record the final mean squared error on the validation set after running DEQGAN training for 500 steps. Each line represents a combination of model weight initialization random seed, learning rate hyperparameters, and final (log) mean squared error of a single experiment.

Notably, the results as a whole exhibit considerable variation in final mean squared error. However, when we filter on experiments achieving low mean squared errors ($\leq 10^{-8}$), we see that hyperparameter settings exist, for each of the model weight initialization seeds, that provide highly accurate DEQGAN solutions. We also observe a pattern in the hyperparameters which produce the low mean squared error experiments. We note that relatively high generator learning rates and low discriminator learning rates lead to the best DEQGAN performance across different model initialization seeds.

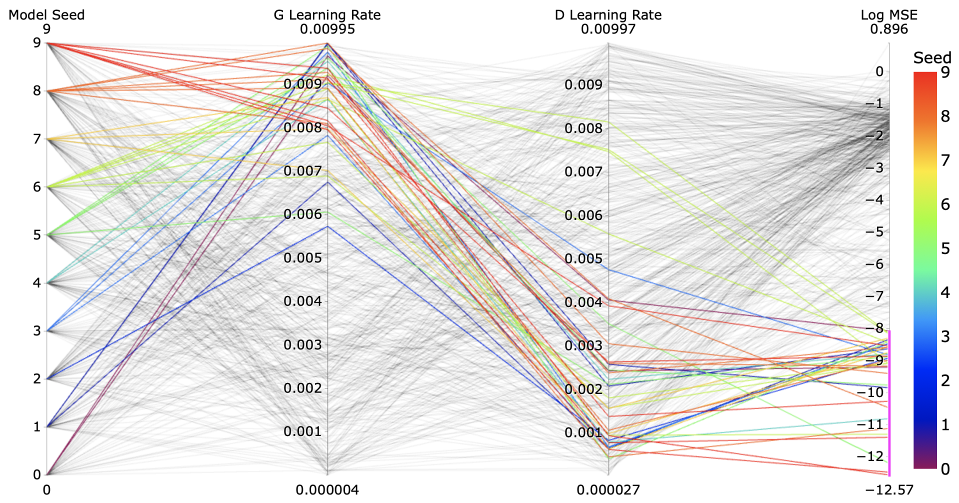


Figure 3: Parallel plot showing the results of 500 DEQGAN experiments on the exponential decay equation. The colors represent the different random seeds used to initialize model weights. We filter the results (non-selected lines appear gray) to highlight experiments achieving mean squared error $\leq 10^{-8}$. The mean squared error is plotted on a log_{10} scale.

7 Conclusion

We have presented a novel method which leverages GAN-based adversarial training to “learn" the loss function for solving differential equations with unsupervised neural networks. We have shown empirically that our our method, which we call Differential Equation GAN (DEQGAN), can obtain multiple orders of magnitude lower mean squared errors than an alternative unsupervised neural network method with L_2, L_1, and Huber loss functions. Moreover, we show that DEQGAN achieves solution accuracy that is competitive with the traditional fourth-order Runge-Kutta and second-order finite difference methods. While our approach is sensitive to hyperparmaters, we have shown that it is possible to train a GAN in a *fully unsupervised* manner to achieve highly accurate solutions to differential equations.

Broader Impact

We hope that the broader impact of this work will be to advance the study of unsupervised neural network methods for solving differential equations. We do not believe that our work holds particularly poignant ethical or societal consequences. We note that our method does not provide theoretical guarantees of solution accuracy, and any critical implementations relying on this should exercise caution.

Acknowledgments and Disclosure of Funding

The authors would like to acknowledge helpful discussions with Marios Mattheakis, Cengiz Pehlevan, and Feiyu Chen.

References

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

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

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

[4] 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.

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

[6] 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.

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

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

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

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

[11] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[31] 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.

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

[33] 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.

[34] 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.

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

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

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

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

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

[40] 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.

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

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

Appendix

Description of Experiments

A plot of the various classical loss functions is provided in Figure 4.

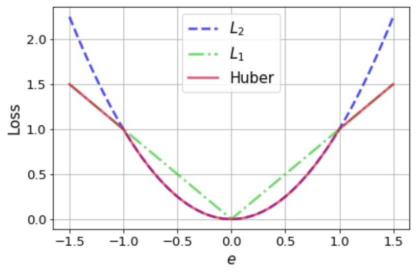


Figure 4: Comparison of L_2, L_1, and Huber loss functions. The Huber loss is equal to L_2 for $e \leq 1$ and to L_1 for $e > 1$.

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

$$\dot{x}(t) + x(t) = 0, \tag{13}$$

with $x(0) = 1$ and $t \in (0, 10)$. The ground truth solution $x(t) = e^{-t}$ can be obtained analytically. We reiterate, however, that our method is fully unsupervised and does not make use of this solution data during training. We simply use the ground truth solutions to report mean squared errors of predicted solutions.

To set up the problem for DEQGAN, we define $LHS = \dot{x} + x$ and $RHS = 0$. Figure 5 presents the results from training DEQGAN on this equation.

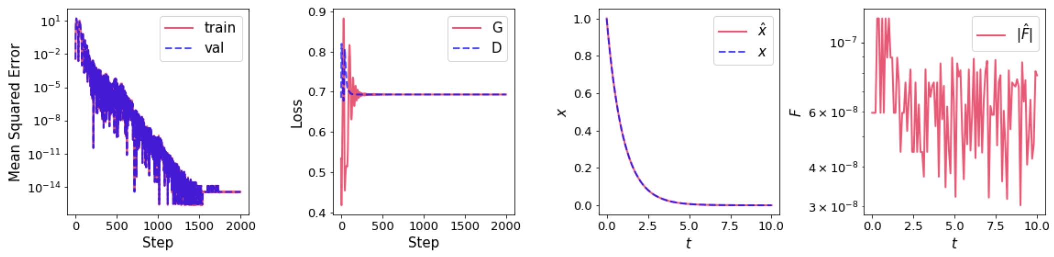


Figure 5: Visualization of DEQGAN training for the exponential decay problem. The left-most figure plots the mean squared error vs. iteration. To the right, we plot the value of the generator (G) and discriminator (D) losses at each iteration. Right of this we plot the prediction of the generator $\hat{x}$ and the true analytic solution x as functions of time t. The right-most figure plots the absolute value of the residual of the predicted solution $\hat{F}$.

Simple Harmonic Oscillator (SHO) Consider the motion of an oscillating body $x(t)$, which can be modeled by the simple harmonic oscillator differential equation

$$\ddot{x}(t) + x(t) = 0, \tag{14}$$

with $x(0) = 0$, $\dot{x}(0) = 1$, and $t \in (0, 2\pi)$. This differential equation can be solved analytically and has an exact solution $x(t) = \sin t$.

Here we set $LHS = \ddot{x} + x$ and $RHS = 0$. Figure 6 plots the results of training DEQGAN on this problem.

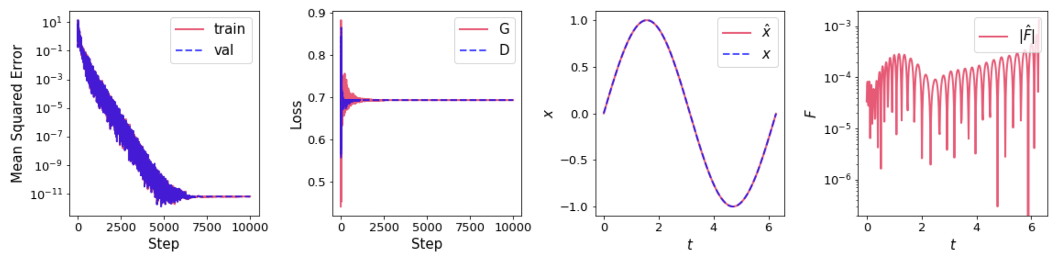


Figure 6: Visualization of DEQGAN training for the simple harmonic oscillator problem. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the prediction of the generator $\hat{x}$ and the true analytic solution x as functions of time t. The right-most figure plots the absolute value of the residual of the predicted solution $\hat{F}$.

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

$$\ddot{x}(t) + 2\beta\dot{x}(t) + \omega^2 x(t) + \phi x(t)^2 + \epsilon x(t)^3 = 0, \tag{15}$$

with $\beta = 0.1, \omega = 1, \phi = 1, \epsilon = 0.1$, $x(0) = 0$, $\dot{x}(0) = 0.5$, and $t \in (0, 4\pi)$. This equation does not admit an analytical solution. Instead, we use the high-quality solver provided by SciPy's `solve_ivp` [40].

We set $LHS = \ddot{x} + 2\beta\dot{x} + \omega^2 x + \phi x^2 + \epsilon x^3 = 0$ and $RHS = 0$. Figure 7 plots the results obtained from training DEQGAN on this equation.

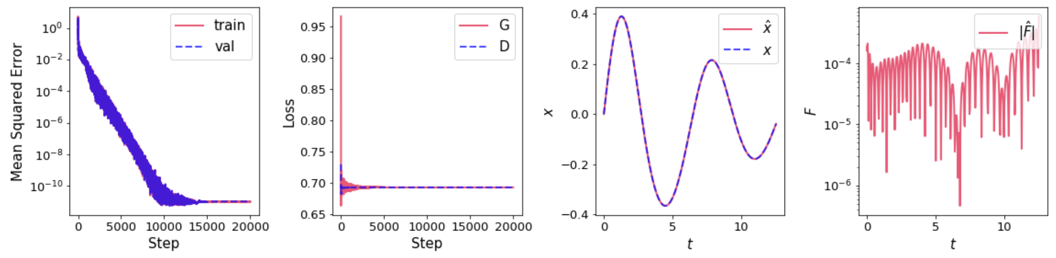


Figure 7: Visualization of DEQGAN training for the nonlinear oscillator problem. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the prediction of the generator $\hat{x}$ and the ground truth solution x as functions of time t. The right-most figure plots the absolute value of the residual of the predicted solution $\hat{F}$.

Non-Autonomous System (NAS) Consider the system of ordinary differential equations given by

$$\dot{x}(t) = -ty \tag{16}$$

$$\dot{y}(t) = tx \tag{17}$$

with $x(0) = 1$, $y(0) = 0$, and $t \in (0, 2\pi)$. This equation has an exact analytical solution given by

$$x = \cos\left(\frac{t^2}{2}\right) \tag{18}$$

$$y = \sin\left(\frac{t^2}{2}\right). \tag{19}$$

Here we set

$$LHS = \left[\frac{dx}{dt} + ty, \frac{dy}{dt} - xy\right]^T \tag{20}$$

and $RHS = [0, 0]^T$. Figure 8 plots the result of training DEQGAN on this problem.

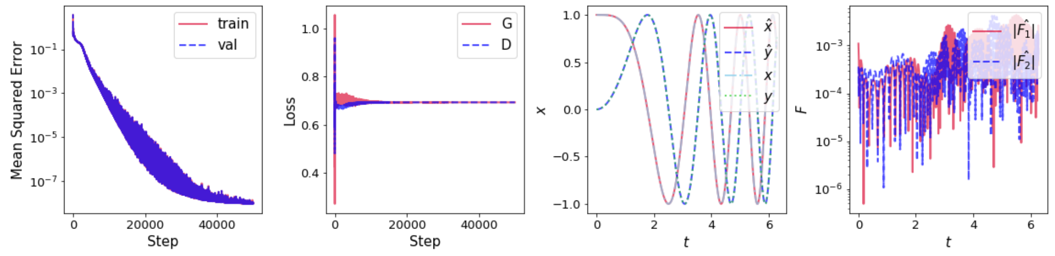


Figure 8: Visualization of DEQGAN training for the non-autonomous system of equations. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the predictions of the generator $\hat{x}, \hat{y}$ and the true analytic solutions x, y as functions of time t. The right-most figure plots the absolute value of the residuals of the predicted solution $\hat{F_j}$ for each equation j.

SIR Epidemiological Model (SIR) Given the recent outbreak and pandemic of novel coronavirus (COVID-19) [41], we consider an epidemiological model of infectious disease spread given by a system of ordinary differential equations. Specifically, consider the Susceptible $S(t)$, Infected $I(t)$, Recovered $R(t)$ model for the spread of an infectious disease over time t. The model is defined by a system of three ordinary differential equations

$$\frac{dS}{dt} = -\beta \frac{IS}{N} \tag{21}$$

$$\frac{dI}{dt} = \beta \frac{IS}{N} - \gamma I \tag{22}$$

$$\frac{dR}{dt} = \gamma I \tag{23}$$

where $\beta = 3, \gamma = 1$ are given constants related to the infectiousness of the disease, $N = S + I + R$ is the (constant) total population, $S(0) = 0.99, I(0) = 0.01, R(0) = 0$, and $t \in (0, 10)$. Again we use SciPy's `solve_ivp` solver [40] to obtain ground truth solutions.

We set LHS to be the vector

$$LHS = \left[\frac{dS}{dt} + \beta \frac{IS}{N}, \frac{dI}{dt} - \beta \frac{IS}{N} + \gamma I, \frac{dR}{dt} - \gamma I\right]^T \tag{24}$$

and $RHS = [0, 0, 0]^T$. We present the results of training DEQGAN to solve this system of differential equations in Figure 9.

Poisson Equation (POS) Consider the Poisson partial differential equation (PDE) given by

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 2x(y-1)(y-2x+xy+2)e^{x-y} \tag{25}$$

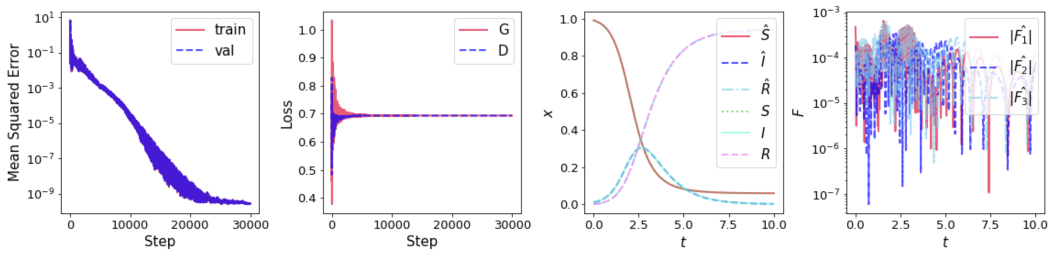


Figure 9: Visualization of DEQGAN training for the SIR system of equations. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the predictions of the generator $\hat{S}$, $\hat{I}$, $\hat{R}$ and the ground truth solutions S, I, R as functions of time t. The right-most figure plots the absolute value of the residuals of the predicted solution $\hat{F}_j$ for each equation j.

where $(x, y) \in [0, 1] \times [0, 1]$. The equation is subject to Dirichlet boundary conditions on the edges of the unit square

$$\begin{aligned} u(x,y)\Big|_{x=0} &= 0 \\ u(x,y)\Big|_{x=1} &= 0 \\ u(x,y)\Big|_{y=0} &= 0 \\ u(x,y)\Big|_{y=1} &= 0. \end{aligned} \tag{26}$$

The analytical solution is

$$u(x, y) = x(1 - x)y(1 - y)e^{x-y}. \tag{27}$$

We use the two-dimensional Dirichlet boundary adjustment formulae provided in Chen et al. [4]. To set up the problem for DEQGAN we let

$$LHS = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - 2x(y - 1)(y - 2x + xy + 2)e^{x-y} \tag{28}$$

and $RHS = 0$. We present the results of training DEQGAN on this problem in Figure 10.

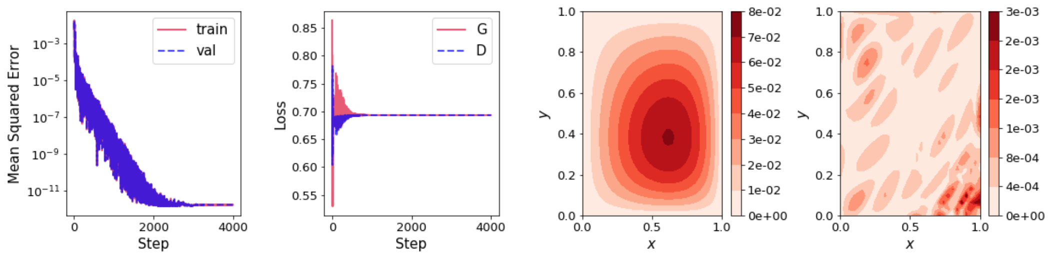


Figure 10: Visualization of DEQGAN training for the Poisson equation. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the prediction of the generator $\hat{u}$ as a function of position (x, y). The right-most figure plots the absolute value of the residual $\hat{F}$, as a function of (x, y).

DEQGAN Hyperparameters

We performed 1000 iterations of random search to tune the hyperparameters of DEQGAN for each differential equation. Table 3 summarizes the final hyperparameter settings used for DEQGAN.

Table 3: Hyperparameter Settings for DEQGAN

Hyperparameter	EXP	SHO	NLO	NAS	SIR	POS
Num. Iterations	$2 \cdot 10^3$	$1 \cdot 10^4$	$2 \cdot 10^4$	$5 \cdot 10^4$	$3 \cdot 10^4$	$4 \cdot 10^3$
Num. Grid Points	100	400	400	800	800	32×32
G Units/Layer	30	40	40	30	40	40
G Num. Layers	2	4	4	3	2	4
D Units/Layer	20	40	30	50	20	20
D Num. Layers	4	2	3	2	3	4
Activations	tanh	tanh	tanh	tanh	tanh	tanh
G Learning Rate	0.008	0.009	0.006	0.006	0.010	0.008
D Learning Rate	0.0005	0.002	0.0007	0.001	0.002	0.002
G β_1 (Adam)	0.671	0.444	0.102	0.706	0.207	0.410
G β_2 (Adam)	0.143	0.633	0.763	0.861	0.169	0.447
D β_1 (Adam)	0.866	0.271	0.541	0.538	0.193	0.593
D $\beta 2$ (Adam)	0.165	0.142	0.677	0.615	0.617	0.915
Exponential LR Decay (γ)	0.991	0.998	0.999	0.9998	0.9996	0.996

Non-GAN Hyperparameter Tuning

Table 4 presents results after tuning the hyperparameters of the alternative unsupervised neural network method with L_1, L_2, and Huber loss functions. We ran 1000 random search iterations for each differential equation.

Table 4: Experimental Results With Non-GAN Hyperparameter Tuning

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