DEQGAN: Solving Differential Equations with Generative Adversarial Networks

Anonymous Authors[1]

Abstract

We present a new method for solving differential equations in an unsupervised manner based on Generative Adversarial Networks (GANs). Given the plethora of potential loss functions available for the problem of solving differential equations with neural networks, we argue that learning the loss function with GAN-based adversarial training is a more principled approach to generating functional solutions to differential equations. We present empirical results on problems of varying complexity which demonstrate that our method returns solutions that achieve orders of magnitude lower mean squared error (computed from known analytic or numerical solutions) than an unsupervised method trained with the L_2-norm loss function[1].

1. Introduction

Solutions to differential equations are of significant scientific and engineering interest. However, many equations of practical relevance can't be solved analytically and must be approximated numerically. Popular algorithms for solving these systems, such as finite differences and finite elements, return solutions at *fixed* sets of points. While these algorithms perform well, we believe that directly *learning a function* (e.g. a neural network) which exactly satisfies the governing equations is desirable as it can generate more accurate interpolations (Lagaris et al., 1998a), enables faster simulation due to parallelization[2], and scales more favorably in terms of both accuracy and computational cost in higher dimensions (Han et al., 2017).

Neural networks have been shown to perform extremely well on a variety of supervised machine learning problems (Krizhevsky et al., 2012; Sutskever et al., 2014; He et al., 2015; Bahdanau et al., 2015; Vaswani et al., 2017). This success, and their distinction as "universal" function approximators (Hornik et al., 1989), has inspired researchers from a broad range of disciplines to apply neural networks to many problems. Early work on training neural networks to solve differential equations (Lagaris et al., 1998a) demonstrated that a neural network trained with a L_2-norm loss function could provide solutions to differential equations which reduce interpolation errors by orders of magnitude compared to cubic interpolation of solutions from the finite element method.

Inspired by this line of research, but unsatisfied by the arbitrary choice of the L_2 loss function, we set out to *learn the loss function*[3] through an adversarial training algorithm inspired by GANs (Goodfellow et al., 2014). We believe this is a more principled approach for the problem of solving differential equations because we lack any theoretical justification for a particular loss function. Furthermore, we believe that using a rich function space of learned neural network loss functions can yield more accurate solutions due to the potentially complex spatio-temporal dependencies present in solutions to some differential equations. Indeed, GANs have been shown to excel in scenarios where pointwise loss functions, such as the L_2-norm, struggle precisely due to their inability to capture such dependencies (Larsen et al., 2015; Ledig et al., 2016; Karras et al., 2018).

This paper presents a new method for training GANs to solve differential equations in a fully unsupervised manner, without leveraging any prior solution data (e.g. from simulation or experiment) during training[4]. Our main contribution is a method, which we call "Differential EQuation GAN" (DEQGAN), for formulating the task of solving differential equations as a GAN training problem. DEQGAN works by separating the differential equation into left-hand side (LHS) and right-hand side (RHS), then training the generator to

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

[1]Code will be made available.

[2]Neural networks are much easier to parallelize across time than classical numerical algorithms.

[3]One can think of the *discriminator* in a GAN as a parametric loss function which we learn during training.

[4]We use known solution values after training to compute the mean squared error of the solutions produced by the methods we are evaluating.

produce a LHS that is indistinguishable by the discriminator from the RHS. Experimental results show that our method produces solutions which obtain orders of magnitude lower mean squared errors (computed from known analytic or numerical solutions) than an unsupervised method based on the L_2-norm loss function.

The rest of this paper is structured as follows: section 2 outlines related work; section 3 summarizes key background material for understanding our method and experiments; section 4 details the DEQGAN training algorithm; section 5 presents quantitative experimental results; sections 6 and 7 present some discussion, conclusions, and future work.

2. Related Work

(Lagaris et al., 1998a) proposed solving differential equations in an unsupervised manner with neural networks. They showed that neural network approaches could obtain competitive solution accuracy on a fixed mesh while reducing interpolation errors by orders of magnitude compared to classical finite element methods on various ordinary and partial differential equations. They expanded this work to consider arbitrarily-shaped domains in higher dimensions (Lagaris et al., 1998b), and applied it to quantum mechanics (Lagaris et al., 1997). (Kumar & Yadav, 2011) present a survey of neural network and radial basis function methods for solving differential equations, covering published articles from 2001 to 2011. Recent work by (Sirignano & Spiliopoulos, 2018) uses neural networks in place of basis functions to solve high-dimensional partial differential equations. To reduce the need to re-learn known physical laws, (Mattheakis et al., 2019) embed physical symmetries into the structure of a neural network and show that this improves convergence time and solution accuracy over non-symplectic models.

(Goodfellow et al., 2014) introduced the idea of learning generative models with neural networks and an adversarial training algorithm, called Generative Adversarial Networks (GANs). Since their seminal paper, a plethora of authors have further developed this idea. (Mirza & Osindero, 2014) proposed Conditional GANs which introduce auxiliary conditioning information (e.g. class labels) to enable generative models of conditional distributions and reduce the problem of "mode collapse". (Arjovsky et al., 2017) introduced WGAN, a formulation of GANs based on the Wasserstein distance as loss function and showed that this led to improved training stability and output quality. (Gulrajani et al., 2017) introduced WGAN-GP, an extension to WGAN which approximately enforces a Lipschitz constraint on the discriminator (or "critic") through a gradient penalty instead of ad-hoc weight clipping.

Our work distinguishes itself from other GAN-based approaches to solving differential equations by removing the dependence on using supervised training data (i.e. solutions of the equation). Others have applied GANs to differential equations, but invariably use some physical data gathered by simulation or experiment. (Yang et al., 2018) apply GANs to stochastic differential equations but include "snapshots" of ground-truth data for training. A project by students at Stanford[5] employed GANs to perform "turbulence enrichment" in a manner akin to that of super-resolution for images proposed by (Ledig et al., 2016). However, their method uses data on the solutions of the system when training the GAN.

3. Background

3.1. Unsupervised Neural Networks for Differential Equations

Early work by (Lagaris et al., 1998a) proposed solving differential equations in an unsupervised manner with neural networks. The paper considers general differential equations of the form:

$$F(x, \Psi(x), \Delta\Psi(x), \Delta^2\Psi(x)) = 0 \tag{1}$$

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

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

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

3.2. Generative Adversarial Networks

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) are a type generative model that use two neural networks to induce a generative distribution $p(x)$ of the data by formulating the inference problem as a two-player, zero-sum game.

The generative model first samples a latent variable $z \sim \mathcal{N}(0, 1)$, which is used as input into the generator G (e.g. a neural network). A discriminator D is trained to classify whether its input was sampled from the generator (i.e. "fake") or from a reference data set (i.e. "real"). In practice, GANs have performed exceptionally well at generating realistic samples from complex, high-dimensional data (Ledig et al., 2016; Karras et al., 2018; Zhu et al., 2017).

[5]Link obscured for anonymity.

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:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[1-\log D(G(z))] \quad (3)$$

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

3.3. Conditional GAN

Conditional GANs (Mirza & Osindero, 2014) are a simple extension to the above formulation, whereby the discriminator and/or generator are conditioned on an extra variable y. This could be any kind of auxiliary information, such as class labels or data from other modalities. Formally, a conditional GAN optimizes:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim\ p_{\text{data}}(x)}\left[\log D(x|y)\right] + \mathbb{E}_{z\sim p_z(z)}\left[1-\log D(G(z|y))\right] \quad (4)$$

For this paper, conditioning the discriminator on points in the domain of the system (e.g. the time variable t) leads to improved learning stability.

3.4. Wasserstein Loss with Gradient Penalty (WGAN-GP)

Motivated by the instability of optimizing (3), (Arjovsky et al., 2017) propose the Wasserstein-1 (or "Earth Mover") distance as an alternative loss function to cross-entropy. The WGAN value function is constructed using the Kantorovich-Rubinstein duality (Villani, 2008) to obtain:

$$\min_G \max_{D\in\mathcal{D}} \mathbb{E}_{x\sim p_{\text{real}}(x)}\left[D(x)\right] - \mathbb{E}_{z\sim p(z)}[D(G(z))] \quad (5)$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions. To enforce this constraint, (Gulrajani et al., 2017) propose a gradient penalty which is added to the loss function of the discriminator. This can be interpreted as a soft constraint, since the term added to the loss does not strictly enforce the Lipschitz constraint. The updated discriminator loss becomes:

$$L_D = \mathbb{E}_{z\sim p(z)}[D(G(z))] - \mathbb{E}_{x\sim p_{\text{real}}(x)}\left[D(x)\right] + \lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}\left[(||\nabla_{\hat{x}}\, D(\hat{x})||_2 - 1)^2\right] \quad (6)$$

where $\hat{x} \sim \mathbb{P}_{\hat{x}}$ is the distribution of points uniformly sampled along straight lines between the data distribution $x \sim p_{\text{real}}(x)$ and the generator distribution $G(z)$ with $z \sim p(z)$.

3.5. Adjusting for Physical Constraints

(Mattheakis et al., 2019) show that it is possible to embed physical constraints into neural networks to directly optimize for solutions which are physically realizable. They demonstrate that doing so improves convergence of neural networks trained to solve differential equations. In this paper we employ the adjustments proposed by the paper to the candidate solutions as a way to satisfy initial and boundary conditions. Specifically, for initial values the paper proposes adjusting the output of the neural network following:

$$\hat{x} = x_0 + (1 - e^{-(t-t_0)})\tilde{N}(t) \quad (7)$$

Intuitively, the above equation adjusts the output of the neural network $\tilde{N}$ to be x_0 when $t = t_0$, and decays this constraint exponentially in t.

3.6. Residual Connections

(He et al., 2015) showed that adding residual connections improved training of deep neural networks. We employ residual connections to our deep networks as they allow gradients to more easily flow throughout the models and improve performance. Residual connections augment a typical activation with the identity operation:

$$y = \mathcal{F}(x, W_i) + x \quad (8)$$

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

4. DEQGAN

We train DEQGAN in a fully unsupervised way, without access to ground-truth solutions during training. We rearrange the equation such that the left-hand side LHS contains all of the terms which depend on the generator, and the right-hand side RHS contains only constants (e.g. zero).

Then we sample points from the domain $x \sim \mathcal{D}$ and use them as input to a generator $G(x)$, which produces candidate

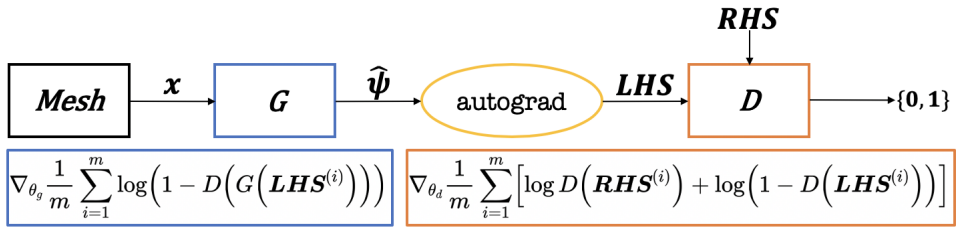

Figure 1. Schematic representation of DEQGAN.

solutions $\hat{\psi}$. We adjust $\hat{\psi}$ for initial or boundary conditions according to the analytic formula in section 3.5. Then we construct the LHS from the differential equation F using automatic differentiation:

$$LHS = F\left(x, \hat{\psi}(x), \Delta\hat{\psi}(x), \Delta^2\hat{\psi}(x)\right) \tag{9}$$

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

From here, training proceeds in a manner similar to traditional GANs. We update the generator G and discriminator D using the following gradients (for vanilla GANs):

$$\eta_G = \nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log\left(1 - D\left(G\left(\boldsymbol{LHS}^{(i)}\right)\right)\right) \tag{10}$$

$$\eta_D = \nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(\boldsymbol{RHS}^{(i)}\right) + \log\left(1 - D\left(\boldsymbol{LHS}^{(i)}\right)\right)\right] \tag{11}$$

We provide a pictorial representation of the setup in Figure 1, and detail training in Algorithm 1.

Our algorithm trains the 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 which try to make LHS indistinguishable from RHS, thereby approximately solving the equation such that $LHS = RHS$.

A subtle yet important point is that training can be unstable if LHS and RHS are not chosen properly. Specifically, we find that training fails if RHS is a function of the generator. For example, consider the equation $\ddot{x} + x = 0$. If we set $LHS = \ddot{x}$ and $RHS = -x$, then RHS will be constantly changing as the generator is updated throughout training. Instead, we can fix this by setting $LHS = \ddot{x} + x$ and $RHS = 0$. For the examples in this paper, we move all terms of the differential equation to the LHS and set $RHS = 0$.

Algorithm 1 DEQGAN

Input: Differential equation F, generator $G(.;\theta_g)$, discriminator $D(.;\theta_d)$, mesh x of m elements with spacing d, initial/boundary condition adjustment ϕ, learning rates α_G, α_D
for $i = 1$ **to** N **do**
  Sample m points $x_s \sim x + \mathcal{N}(0, \frac{1}{3})/d$
  Forward pass $\hat{\psi} = G(x_s)$
  Adjust for conditions $\hat{\psi}' = \phi(\hat{\psi})$
  Set $LHS = F(x, \hat{\psi}', \nabla\hat{\psi}', \nabla^2\hat{\psi}')$, $RHS = \mathbf{0}$
  Update generator $\theta_g \leftarrow \theta_g + \alpha_G\eta_G$ (Eqn 10)
  Update discriminator $\theta_d \leftarrow \theta_d + \alpha_D\eta_D$ (Eqn 11)
end for
Return G

5. Experiments

We run experiments on several problems, comparing DEQGAN to a L_2-based method. We compute the mean squared errors of the solutions produced by each method by using known solutions obtained either analytically or through standard numerical methods. We do not compare to classic numerical methods as others have substantively performed this comparison (Lagaris et al., 1998a;b; Han et al., 2017). We note that deterministic differential equations do not have any aleatoric uncertainty, and given a high-enough capacity neural network, we could in principle obtain arbitrarily low errors. The reason we do not observe this is because we use stochastic gradient descent and randomly perturb a finite set of points on the domain (which we find, in practice, improves the training convergence rate).

We use feed-forward neural networks with residual connections throughout our experiments for both the generator G and discriminator D, and train them with the Adam optimizer (Kingma & Ba, 2014), alternating equally between G and D updates. For all problems, we sample points from the domain (a grid or mesh of points) with a small Gaussian

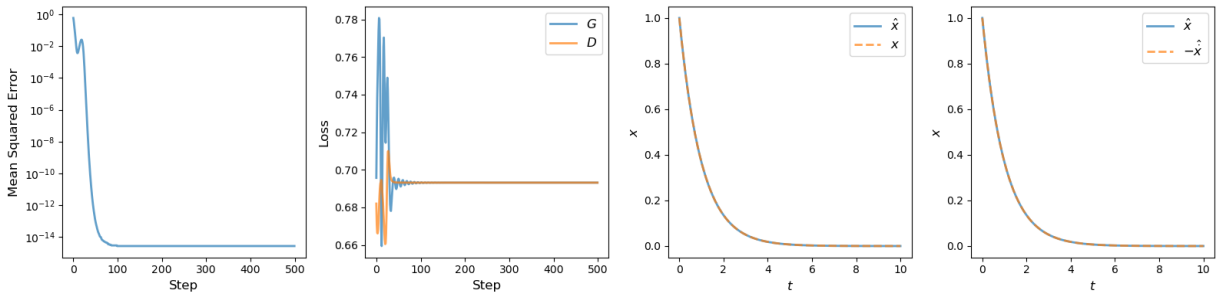

Figure 2. Visualization of DEQGAN training for the exponential decay problem. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the prediction of the generator $\hat{x}$ and the true analytic solution x as functions of time t. The right-most figure plots the prediction $\hat{x}$ and negative first derivative $-\hat{\dot{x}}$.

noise perturbation, as in Algorithm 1.

5.1. Exponential Decay

Consider the exponential decay problem, defined by:

$$\dot{x} + x = 0 \tag{12}$$

with initial condition $x(0) = x_0$. This equation can be solved analytically and its exact solution is $x(t) = x_0 e^{-t}$.

We solve this problem on the range $t \in [0, 10]$ using $n = 100$ points for the grid. We train with the cross-entropy loss function, as in vanilla GANs. Both D and G have 2 layers with 20 units per layer, use LeakyReLU activation functions, and are optimized with learning rates $\alpha_G, \alpha_D = 10^{-2}$. We run training for $N = 500$ iterations.

Figure 2 shows the result of training DEQGAN on the exponential decay problem. Initially the solution produced by G has a high mean squared error, but this quickly decreases and plateaus around step 100. We observe both G and D losses fluctuating to begin training, but that they quickly reach an equilibrium. Importantly, we see that the derivative of the solution produced by the generator $\hat{\dot{x}}$ properly satisfies the actual equation $\dot{x} + x = 0$.

To compare to the L_2-based method, we run 20 replications with different random seeds and plot the results of the 10 best runs[6] in Figure 3. This figure demonstrates the crux of our motivation for DEQGAN: we observe that the GAN-based method which learns the loss function for solving the equation achieves orders of magnitude lower mean squared error than the method using an L_2-norm loss function. We believe this is evidence that DEQGAN is useful for the development of neural network-based solutions to differential equations.

[6]We take the top k (here $k = 10$) best runs, ordered by lowest mean squared error on ground-truth solutions, because GAN training has no guarantee of convergence, and diverges for some initializations (choice of seed). We do the same for the L_2 method for comparability.

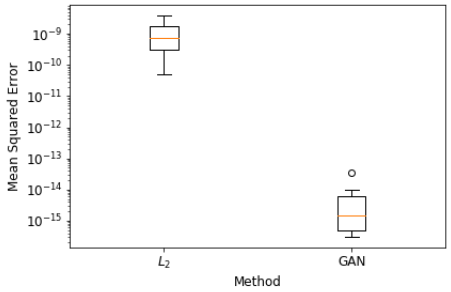

Figure 3. Boxplot of mean squared errors of L_2-based and DEQGAN (GAN) methods on the exponential decay problem. The results shown are the top 10 out of 20 trials with different random seeds (initializations). Each method is run for 500 iterations, with hyperparameters defined in Section 5.1.

5.2. Simple Harmonic Oscillator

Consider a simple harmonic oscillator, given by the differential equation:

$$\ddot{x} + x = 0 \tag{13}$$

with initial conditions $x(0) = x_0$, and $\dot{x}(0) = v_0$. This equation has an exact analytic solution $x(t) = \sin t$.

We solve this equation on the range $t \in [0, 4\pi]$ with $n = 100$ points. We use the Wasserstein loss function with gradient penalty (WGAN-GP) with $\lambda = 0.1$ and train for $N = 10,000$ steps. G uses 2 layers with 64 units per layer, while D has 10 layers with 64 units per layer. Both models use the twice-differentiable $\tanh$ activation function since this is a second-order equation. The models are optimized with Adam with $\alpha_G = \alpha_D = 10^{-3}$ and $(\beta_1, \beta_2) = (0, 0.9)$, and exponentially decaying learning rate schedule with $\gamma = 0.999$.

Figure 4 shows the outcome of training DEQGAN on this

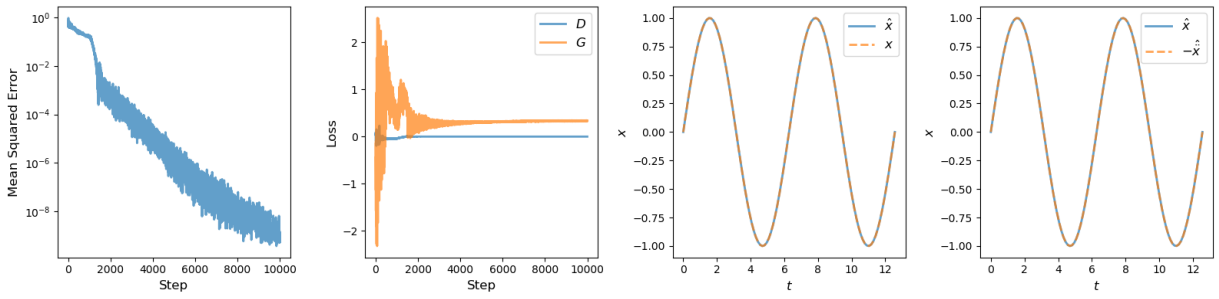

Figure 4. Visualization of DEQGAN training for the simple harmonic oscillator problem. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. Right of this we plot the prediction of the generator $\hat{x}$ and the true analytic solution x as functions of time t. The right-most figure plots the prediction $\hat{x}$ and negative second derivative $-\ddot{\hat{x}}$.

problem. We observe the mean squared error decreasing, albeit less dramatically than on the exponential problem, throughout training. We note that the loss of G fluctuates significantly with respect to the loss of D, but reaches an equilibrium around step 4000. We see that the prediction by G is highly accurate and indistinguishable from the analytic solution. Crucially, we see that the second derivative of the solution produced by the generator is obeying the differential equation $\ddot{x} + x = 0$.

In Figure 5, we compare DEQGAN to the L_2-based method by plotting the mean squared error vs. the number of steps for both methods. We plot the mean and standard deviation of the best 10 runs taken from 20 replications with different random seeds. We see that the DEQGAN method dominates L_2 in terms of mean squared error after about 7,500 steps, and that both methods reach a plateau around 10,000 steps. Again, this figure supports our argument that learning the loss function with DEQGAN is a more principled approach to the problem of solving differential equations, as it leads to more accurate functional solutions than the standard method based on the L_2-norm loss function.

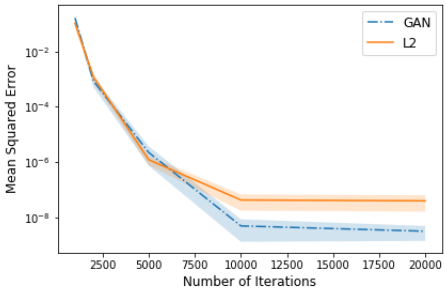

Figure 5. Mean squared error of DEQGAN (GAN) and the L_2-based method vs. number of iterations for the simple harmonic oscillator problem. Results are the top 10 runs from 20 replications with different random seeds (initializations). The line represents the mean while the shaded area indicates one standard deviation on either side of the mean.

5.3. Nonlinear Oscillator

Consider a nonlinear oscillator, defined by the differential equation:

$$\ddot{x} + 2\beta\dot{x} + \omega^2 x + \phi x^2 + \epsilon x^3 = 0 \tag{14}$$

with $\beta = 0.1, \omega = 1, \phi = 1, \epsilon = 0.1$. This equation does not have an exact analytic solution; instead, we use the fourth order Runge–Kutta (RK4) method to obtain "ground truth" solutions.

We solve the equation on the range $t \in [0, 8\pi]$ with $n = 1000$ points for the grid. We use a Wasserstein loss function and gradient penalty (WGAN-GP) with $\lambda = 0.1$, and train for $N = 50,000$ iterations. G uses 12 layers with 64 units per layer; D has 16 layers with 64 units per layer. We use the twice-differentiable $\tanh$ activation function. The models are optimized with learning rates $\alpha_G = \alpha_D = 10^{-3}$ and $(\beta_1, \beta_2) = (0, 0.9)$, and exponentially decaying learning rate schedule with $\gamma = 0.9999$.

Figure 6 visualizes the outcome of training DEQGAN for 50,000 iterations on the non-linear oscillator problem. We observe the mean squared error decreasing throughout training, although with a few jumps at the beginning and end. We note that the G loss fluctuates relative to the D loss, but equilibrates around step 40,000. We see that the solution produced by DEQGAN is indistinguishable from the ground-truth solution.

Figure 7 plots the mean squared error vs. number of iterations for both the DEQGAN and L_2-based method. We plot the mean and standard deviation of the top 3 runs from 20 replications with different random seeds. We see that DEQGAN achieves about 2 orders of magnitude lower mean squared error by 100,000 iterations, and both methods plateau around this point while maintaining an accuracy sep-

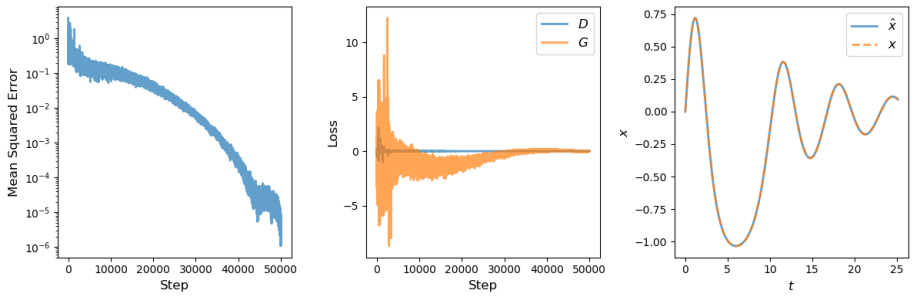

Figure 6. Visualization of DEQGAN training for the non-linear oscillator problem. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. The right-most figure plots the prediction of the generator $\hat{x}$ and the ground-truth solution x as functions of time t.

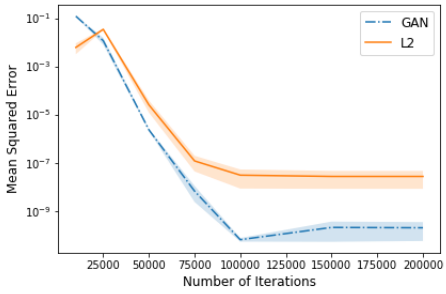

Figure 7. Mean squared error of DEQGAN (GAN) and the L_2-based method vs. number of iterations for the non-linear oscillator problem. Results are the top 3 runs from 20 replications with different random seeds (initializations). The line represents the mean while the shaded area indicates one standard deviation on either side of the mean.

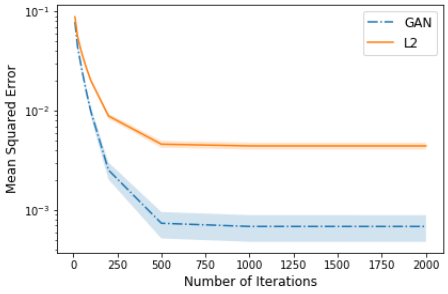

Figure 8. Mean squared error of DEQGAN (GAN) and the L_2-based method vs. number of iterations for the 2-D Poisson problem. Results are the top 10 runs from 20 trials with different initializations. The line represents the mean while the shaded area indicates one standard deviation on either side of the mean.

aration of about 2 orders of magnitude between DEQGAN and the L_2 method.

We note that DEQGAN, at least with the current hyperparameters and architecture, is more sensitive to convergence failure on the non-linear oscillator than the other differential equations studied. Indeed, we must take the top 3 of 20 replications with different random seeds because the majority of runs do not converge. Nevertheless, the best solutions produced by DEQGAN maintain superior accuracy over the best solutions produced by the standard L_2 method.

5.4. Two-Dimensional Poisson Equation

Consider the two-dimensional (2-D) Poisson equation, given by the partial differential equation:

$$-\nabla^2 u = f \tag{15}$$

with boundary condition $u_D = 0$ on the unit square, constant $f = 10$, and where ∇^2 represents the Laplace operator. We solve this equation with $x, y \in [0, 1]$ with $n = 10,000$ points. We use WGAN-GP with $\lambda = 0.1$. We use 4 layers and 64 units per layer for both G and D, with $\tanh$ activations. Both models are optimized with equal learning rates $\alpha_G = \alpha_D = 10^{-3}$, and $(\beta_1, \beta_2) = (0, 0.9)$, and exponentially decaying learning rate schedule with $\gamma = 0.99$

Figure 9 visualizes DEQGAN training for 500 iterations on the 2-D Poisson equation. We see that mean squared error quickly decreases and reaches $\sim 10^{-3}$. We observe that the G and D losses initially diverge away from each other, but move towards an equilibrium by step 400. The solution $\hat{u}$ produced by DEQGAN is visually accurate and matches closely with the expected result.

In Figure 8, we compare DEQGAN to L_2-based training by plotting the mean squared error vs. number of training iterations for each method. We plot the mean and one standard deviation of the top 10 trials taken from 20 runs with different initializations. We observe that DEQGAN immediately dominates the L_2-based method, obtaining about an order of magnitude lower mean squared error by 500 iterations, and maintaining this quality gap thereafter.

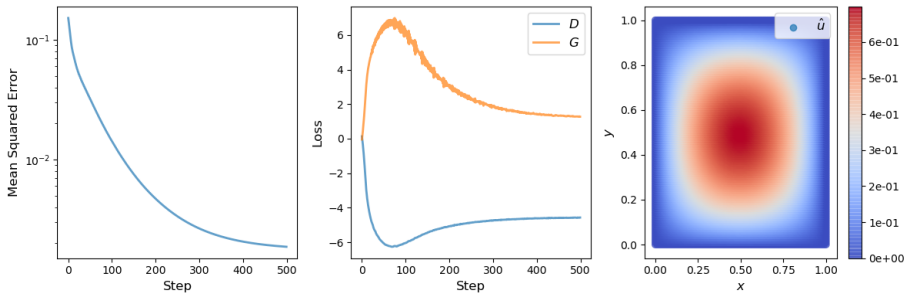

Figure 9. Visualization of DEQGAN training for the 2-D Poisson equation. The left-most figure plots the mean squared error vs. step (iteration) count. To the right of this, we plot the value of the generator (G) and discriminator (D) losses for each step. The right-most figure plots the prediction of the generator $\hat{u}$ as a function of x and y.

6. Discussion

6.1. Training Stability

A point that we have not explicitly addressed is the instability of the GAN training algorithm. In fact, this is not new, and much work has been dedicated towards improving the stability and convergence properties of GANs (Arjovsky et al., 2017; Gulrajani et al., 2017; Berthelot et al., 2017; Mirza & Osindero, 2014).

Here we have found that the choice of random seed, and thus the initialization of the generator and discriminator weights, affects convergence. This is why, in our experiments, we take the best k (e.g. $k = 10$) out of n runs, where best is defined in terms of lowest mean squared error on ground-truth solutions, and discard the bottom $n - k$ to eliminate divergences[7]. This is a consequence of the instability of the minimax objective optimized during GAN training, and something we hope will be ameliorated with the ongoing development of GANs.

6.2. Hyperparameter Tuning

Training GANs often requires setting hyperparameters carefully to obtain the best possible performance. We performed grid searches over hyperparameters (detailed in the supplementary material) to reach the results presented here for DEQGAN. However, we believe that it is often necessary to tune hyperparameters for a given problem, whether or not a GAN is being used, and so this is an acceptable requirement.

6.3. Future Work

Here we trained GANs on deterministic differential equations, but we believe DEQGAN holds promise for *stochastic* differential equations since the GAN framework is naturally, and originally, suited to learning from distributions.

[7]This is similar to how some algorithms, such as Expectation Maximization (EM), which finds local optima, should be run for multiple random seeds with only the best value(s) retained.

7. Conclusion

We have presented a novel technique for formulating the task of solving differential equations as a GAN training problem. Our method, DEQGAN, is fully unsupervised as it doesn't require any ground-truth solution data during training. We have shown empirically that DEQGAN, which we think of as learning the loss function through adversarial training, can lead to orders of magnitude improvements in accuracy across a range of problems of varying complexity. We hope that this work contributes meaningful insights to the growing community of researchers developing neural network methods for differential equations.

References

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein gan, 2017.

Bahdanau, D., Cho, K., and Bengio, Y. 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*, 2015. URL http://arxiv.org/abs/1409.0473.

Berthelot, D., Schumm, T., and Metz, L. BEGAN: boundary equilibrium generative adversarial networks. *CoRR*, abs/1703.10717, 2017. URL http://arxiv.org/abs/1703.10717.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks, 2014.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. Improved training of wasserstein gans, 2017.

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

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

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

Karras, T., Laine, S., and Aila, T. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018. URL http://arxiv.org/abs/1812.04948.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

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

Kumar, M. and Yadav, N. 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, 2011.

Lagaris, I., Likas, A., and Fotiadis, D. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998a. ISSN 1045-9227. doi: 10.1109/72.712178. URL http://dx.doi.org/10.1109/72.712178.

Lagaris, I. E., Likas, A., and Fotiadis, D. I. Artificial neural network methods in quantum mechanics. 1997.

Lagaris, I. E., Likas, A., and Papageorgiou, D. G. Neural network methods for boundary value problems defined in arbitrarily shaped domains. *CoRR*, cs.NE/9812003, 1998b. URL https://arxiv.org/abs/cs/9812003.

Larsen, A. B. L., Sønderby, S. K., and Winther, O. Autoencoding beyond pixels using a learned similarity metric. *CoRR*, abs/1512.09300, 2015. URL http://arxiv.org/abs/1512.09300.

Ledig, C., Theis, L., Huszar, F., Caballero, J., Aitken, A. P., Tejani, A., Totz, J., Wang, Z., and Shi, W. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016. URL http://arxiv.org/abs/1609.04802.

Mattheakis, M., Protopapas, P., Sondak, D., Giovanni, M. D., and Kaxiras, E. Physical symmetries embedded in neural networks, 2019.

Mirza, M. and Osindero, S. Conditional generative adversarial nets, 2014.

Sirignano, J. A. and Spiliopoulos, K. Dgm: A deep learning algorithm for solving partial differential equations. 2018.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL http://arxiv.org/abs/1409.3215.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Villani, C. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

Yang, L., Zhang, D., and Karniadakis, G. E. Physics-informed generative adversarial networks for stochastic differential equations, 2018.

Zhu, J., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017. URL http://arxiv.org/abs/1703.10593.