Unsupervised Neural Network Methods for Solving Differential Equations

Learning the Loss Function & Sampling Strategies

Dylan L. Randle

School of Engineering & Applied Sciences
Harvard University
Cambridge, MA 02138

May 4, 2020

Table of Contents

Table of Contents

Differential Equations

Differential equations are of significant scientific and engineering interest.

- They relate quantities to rates of change (i.e. derivatives)

Differential Equations

Differential equations are of significant scientific and engineering interest.

- They relate quantities to rates of change (i.e. derivatives)
- Applied to physics, chemistry, biology, engineering, economics

Differential Equations

Differential equations are of significant scientific and engineering interest.

- They relate quantities to rates of change (i.e. derivatives)
- Applied to physics, chemistry, biology, engineering, economics
- However, equations of practical interest are generally not analytically solvable

Differential Equations

Differential equations are of significant scientific and engineering interest.

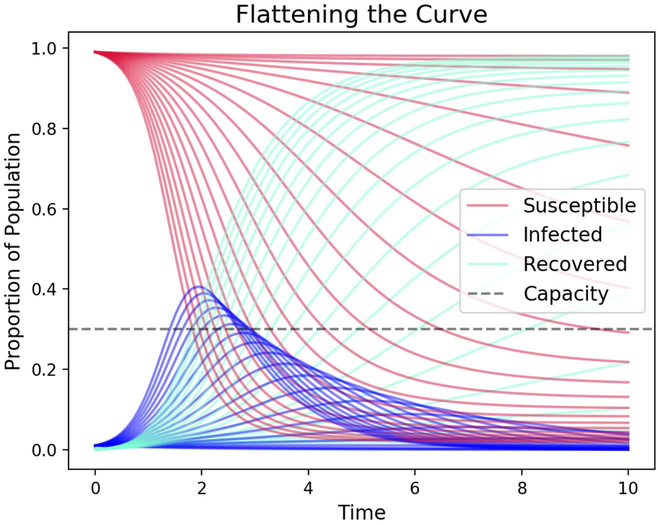
- They relate quantities to rates of change (i.e. derivatives)
- Applied to physics, chemistry, biology, engineering, economics
- However, equations of practical interest are generally not analytically solvable
- Instead, numerical methods compute approximate solutions over a discrete mesh or grid

Example: Fluid Flow

Credit: Pavel Dobryakov

https://paveldogreat.github.io/WebGL-Fluid-Simulation/

Example: Infectious Disease



Why Neural Networks?

Traditional numerical methods perform well and the theory for stability and convergence is well-established. Why use neural networks? Some potential advantages:

- Remove reliance on finely-crafted grids which suffer the "curse of dimensionality"; can be more tractable in high-dimensional settings (Sirignano & Spiliopoulos, 2018; Raissi, 2018; Han et al., 2017)

Why Neural Networks?

Traditional numerical methods perform well and the theory for stability and convergence is well-established. Why use neural networks? Some potential advantages:

- Remove reliance on finely-crafted grids which suffer the "curse of dimensionality"; can be more tractable in high-dimensional settings (Sirignano & Spiliopoulos, 2018; Raissi, 2018; Han et al., 2017)
- Theoretically, neural networks can approximate any reasonable function (Hornik et al., 1989); closed-form, differentiable functions could solve inverse problems, provide more principled & accurate interpolation scheme

Why Neural Networks?

Traditional numerical methods perform well and the theory for stability and convergence is well-established. Why use neural networks? Some potential advantages:

- Remove reliance on finely-crafted grids which suffer the "curse of dimensionality"; can be more tractable in high-dimensional settings (Sirignano & Spiliopoulos, 2018; Raissi, 2018; Han et al., 2017)
- Theoretically, neural networks can approximate any reasonable function (Hornik et al., 1989); closed-form, differentiable functions could solve inverse problems, provide more principled & accurate interpolation scheme
- Can more precisely obey certain constraints, such as conservation of energy (Mattheakis et al., 2020)

Why Neural Networks?

Traditional numerical methods perform well and the theory for stability and convergence is well-established. Why use neural networks? Some potential advantages:

- Remove reliance on finely-crafted grids which suffer the "curse of dimensionality"; can be more tractable in high-dimensional settings (Sirignano & Spiliopoulos, 2018; Raissi, 2018; Han et al., 2017)
- Theoretically, neural networks can approximate any reasonable function (Hornik et al., 1989); closed-form, differentiable functions could solve inverse problems, provide more principled & accurate interpolation scheme
- Can more precisely obey certain constraints, such as conservation of energy (Mattheakis et al., 2020)
- Embarassingly data-parallel, even in temporal dimensions; more readily parallelizable for computational speedup

Table of Contents

Artificial Neural Networks

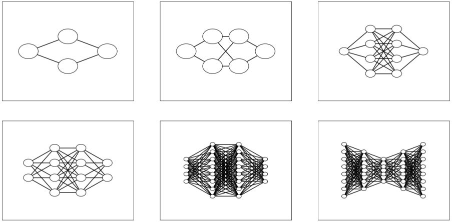
Parametric models loosely based on the human brain. Sequence of affine transformations followed by activation functions:

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

where

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

with $\sigma(\cdot) = \tanh(\cdot)$, for example.



Unsupervised Neural Networks for Differential Equations

Lagaris et al. (1998) proposed solving differential equations in an unsupervised manner with neural networks. Consider differential equations of the form

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

The learning problem is formulated as minimizing the sum of squared errors (i.e. residuals) 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 parameterized by θ, and $\Psi_\theta(x)$ yields predicted solutions.

Adjusting for Constraints

Mattheakis et al. (2019) consider adjusting the neural network solution $N(t)$ to satisfy the initial condition $N(t_0) = x_0$. This is achieved by applying the transformation

$$\tilde{N}(t) = x_0 + \left(1 - e^{-(t - t_0)}\right) N(t) \tag{3}$$

Intuitively, this adjusts the output of the neural network $N(t)$ to be exactly x_0 when $t = t_0$, and decays this constraint exponentially in t. We apply this adjustment throughout to satisfy initial and boundary conditions.

Example: Simple Harmonic Oscillator

Consider the motion $x(t)$ of an oscillating body (e.g. a mass on a frictionless spring) given by

$$\ddot{x}(t) + x(t) = 0 \tag{4}$$

with initial conditions $x_0 = 0$ and $\dot{x}_0 = 1$.[1] We optimize

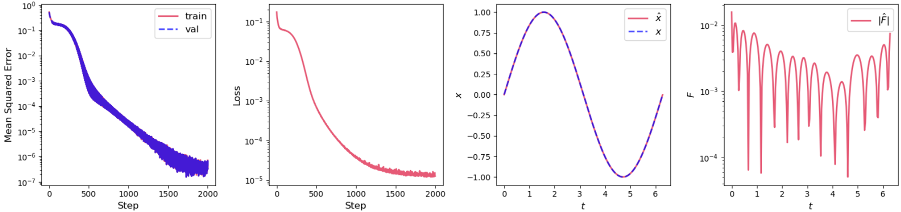
$$\min_{\theta} \sum_{t \in T} \left(\hat{\ddot{x}}_\theta(t) + \hat{x}_\theta(t) \right)^2 \tag{5}$$

to train the model, where $\hat{x}_\theta(t)$ is the output of the neural network.

[1] Exact analytical solution $x(t) = sin(t)$

Example: Simple Harmonic Oscillator

A two hidden layer network composed of 30 units per layer solves this problem to a high degree of accuracy (low mean squared error).



For more detail on this classical unsupervised neural network approach, see e.g. Lagaris et al. (1998); Mattheakis et al. (2019).

Table of Contents

Motivation

- Classical setting of data following a Gaussian noise model

$$y = x + \epsilon, \quad \epsilon \sim N(0, \sigma^2) \tag{6}$$

has clear justification for the squared error loss function (L_2 norm) from the maximum likelihood principle

Motivation

- Classical setting of data following a Gaussian noise model

$$y = x + \epsilon, \quad \epsilon \sim N(0, \sigma^2) \tag{6}$$

  has clear justification for the squared error loss function (L_2 norm) from the maximum likelihood principle
- Deterministic differential equations, with no noise model, have no such justification. To circumvent this we propose *learning the loss function* with Generative Adversarial Networks (GANs)

Motivation

- Classical setting of data following a Gaussian noise model

$$y = x + \epsilon, \quad \epsilon \sim N(0, \sigma^2) \tag{6}$$

  has clear justification for the squared error loss function (L_2 norm) from the maximum likelihood principle
- Deterministic differential equations, with no noise model, have no such justification. To circumvent this we propose *learning the loss function* with Generative Adversarial Networks (GANs)
- Moreover, GANs have been shown to excel in scenarios where classic loss functions struggle (Larsen et al., 2015; Ledig et al., 2016; Karras et al., 2018)

Generative Adversarial Networks (GANs)

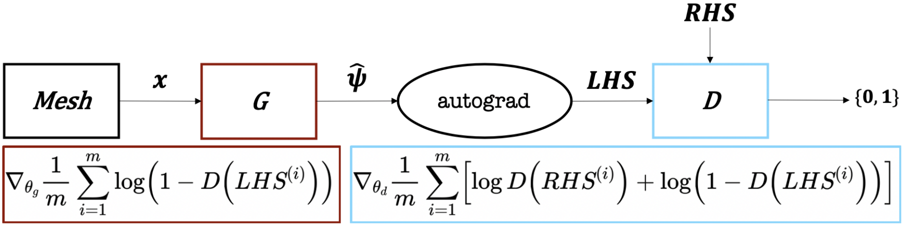
Goodfellow et al. (2014) introduced GANs as a two player game between a generator G and discriminator D such that the generator attempts to trick the discriminator to classify "fake" samples as "real". Formally, one optimizes the minimax objective

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[1 - \log D(G(z))] \quad (7)$$

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.

Differential Equation GAN (DEQGAN)

Separate equation into left-hand side *LHS* and right-hand side *RHS*, and set *LHS* as the “fake” component and *RHS* as “real”. DEQGAN learns to approximately solve the equation by setting $LHS = RHS$.



DEQGAN Algorithm

Algorithm 1 DEQGAN

Input: Differential equation F, generator $G(\cdot;\theta_g)$, discriminator $D(\cdot;\theta_d)$, mesh x of m elements with spacing d, initial/boundary condition adjustment ϕ, learning rates α_G, α_D, Adam moment coefficients $\beta_{G1}, \beta_{G2}, \beta_{D1}, \beta_{D2}$

for $i = 1$ **to** N **do**

    Sample m points $x_s \sim x + \mathcal{N}(0, \frac{d}{3})$

    Forward pass $\hat{\psi} = G(x_s)$

    Adjust for conditions $\hat{\psi}' = \phi(\hat{\psi})$

    Set $LHS = F(x, \hat{\psi}', \nabla\hat{\psi}', \nabla^2\hat{\psi}')$, $RHS = \mathbf{0}$

    Update generator $\theta_g \leftarrow Adam(\theta_g, \alpha_G, -\eta_G, \beta_{G1}, \beta_{G2})$

    Update discriminator $\theta_d \leftarrow Adam(\theta_d, \alpha_D, \eta_D, \beta_{D1}, \beta_{D2})$

end for

Return G

Extensions to Traditional GANs

- Two Time-Scale Update Rule (TTUR): discriminator and generator trained with separate learning rates; in some cases, TTUR ensures convergence to a stable local Nash equilibrium (Heusel et al., 2017)

Extensions to Traditional GANs

- Two Time-Scale Update Rule (TTUR): discriminator and generator trained with separate learning rates; in some cases, TTUR ensures convergence to a stable local Nash equilibrium (Heusel et al., 2017)
- Spectral Normalization (Miyato et al., 2018):

$$W_{SN} = \frac{W}{\sigma(W)}, \tag{8}$$

where

$$\sigma(W) = \max_{\|h\|_2 \leq 1} \|Wh\|_2, \tag{9}$$

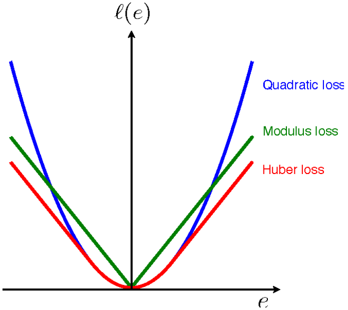
which bounds the Lipschitz constant of the discriminator ≤ 1.

Experiments

- Perform experiments on 4 differential equations of increasing complexity

Experiments

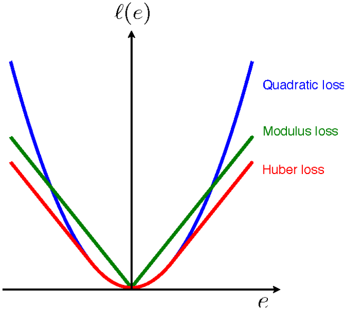
- Perform experiments on 4 differential equations of increasing complexity
- Compare DEQGAN to the classical unsupervised neural network method with L_1, L_2, and Huber loss functions



Credit: Pediredla & Seelamantula (2011)

Experiments

- Perform experiments on 4 differential equations of increasing complexity
- Compare DEQGAN to the classical unsupervised neural network method with L_1, L_2, and Huber loss functions



Credit: Pediredla & Seelamantula (2011)

- Show that DEQGAN obtains multiple orders of magnitude lower mean squared errors than classical neural network methods

Experiment: Exponential Decay

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

$$\dot{x}(t) + x(t) = 0, \tag{10}$$

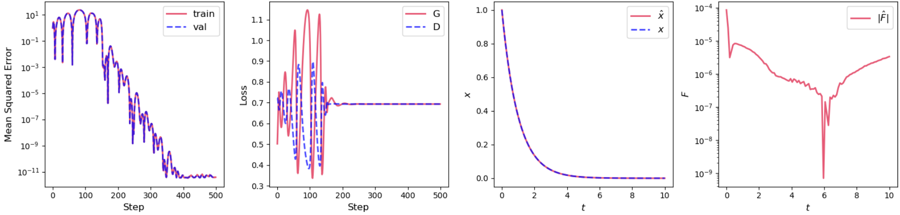
with initial condition $x(0) = 1$.[2] We set

$$LHS = \dot{x}(t) + x(t),$$

$$RHS = 0.$$

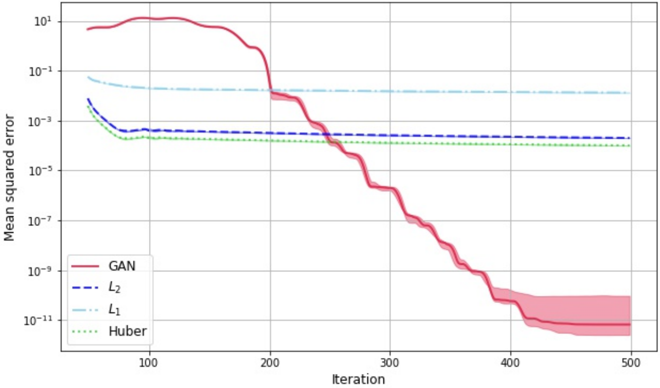
[2] The ground truth solution $x(t) = e^{-t}$ can be obtained analytically.

Experiment: Exponential Decay



- G and D losses initially exhibit high variability but reach equilibrium
- Mean squared error decreases to 10^{-11} by step $\sim$400

Experiment: Exponential Decay



- DEQGAN achieves $\sim 10^{-6}$ times lower mean squared error than classic loss functions (see video)

Experiment: Simple Oscillator

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

$$\ddot{x}(t) + x(t) = 0, \tag{11}$$

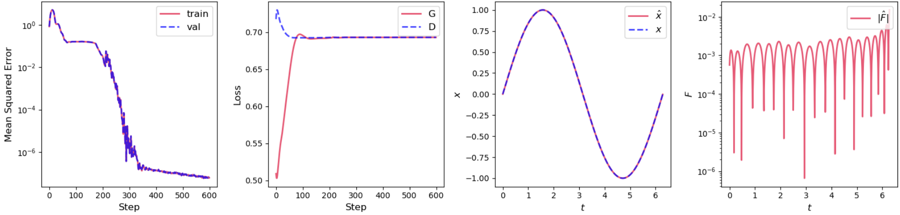
with initial conditions $x(0) = 0$, and $\dot{x}(0) = 1$.[3] We set

$$LHS = \ddot{x}(t) + x(t),$$

$$RHS = 0.$$

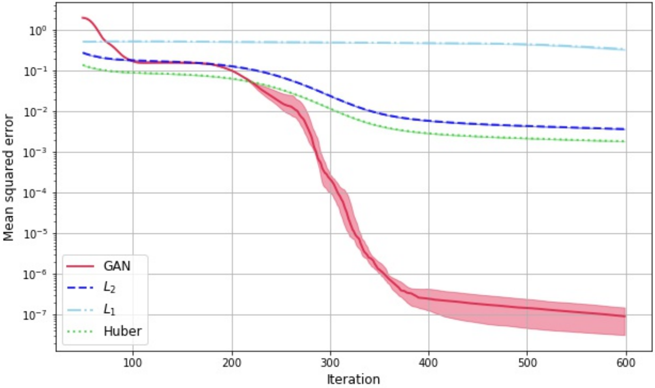
[3] This differential equation has an exact solution $x(t) = \sin t$.

Experiment: Simple Oscillator



- G and D losses reach equilibrium almost monotonically
- Mean squared error decreases to $\sim 10^{-7}$

Experiment: Simple Oscillator



- DEQGAN achieves $\sim 10^{-4}$ times lower mean squared error than classical loss functions (see video)

Experiment: Nonlinear Oscillator

Consider the less idealized motion $x(t)$ of an oscillating body subject to additional forces, given by the nonlinear oscillator differential equation

$$\ddot{x}(t) + 2\beta\dot{x}(t) + \omega^2 x(t) + \phi x(t)^2 + \epsilon x(t)^3 = 0, \tag{12}$$

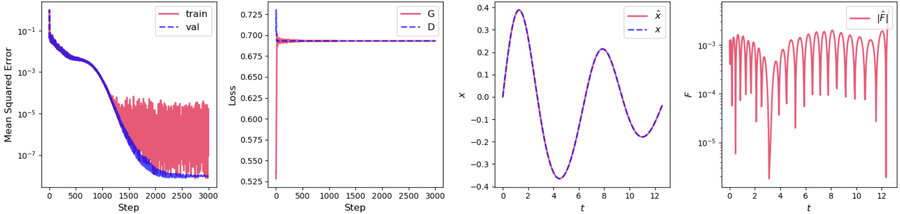
with $\beta = 0.1, \omega = 1, \phi = 1, \epsilon = 0.1$ and initial conditions $x(0) = 0$ and $\dot{x}(0) = 0.5$.[4] We set

$$LHS = \ddot{x} + 2\beta\dot{x} + \omega^2 x + \phi x^2 + \epsilon x^3,$$

$$RHS = 0.$$

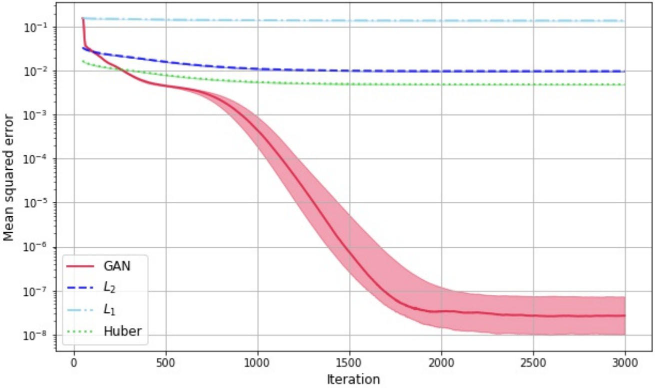
[4] The equation does not have an analytical solution. We use the fourth-order Runge-Kutta method to obtain "ground truth" solutions.

Experiment: Nonlinear Oscillator



- Fast convergence of G and D losses
- Validation mean squared error reaches $\sim 10^{-7}$

Experiment: Nonlinear Oscillator



- DEQGAN reaches $\sim 10^{-5}$ times lower error than classical loss functions (see video)

Experiment: SIR System of Equations

Consider the Susceptible $S(t)$, Infected $I(t)$, Recovered $R(t)$ model for the spread of an infectious disease over time t:

$$\frac{dS}{dt} = -\beta \frac{IS}{N} \tag{13}$$

$$\frac{dI}{dt} = \beta \frac{IS}{N} - \gamma I \tag{14}$$

$$\frac{dR}{dt} = \gamma I \tag{15}$$

with $\beta = 3, \gamma = 1$, constant population $N = S + I + R$, and initial conditions $S_0 = 0.99, I_0 = 0.01, R_0 = 0$.[5]

[5] We obtain ground truth solutions through numerical integration.

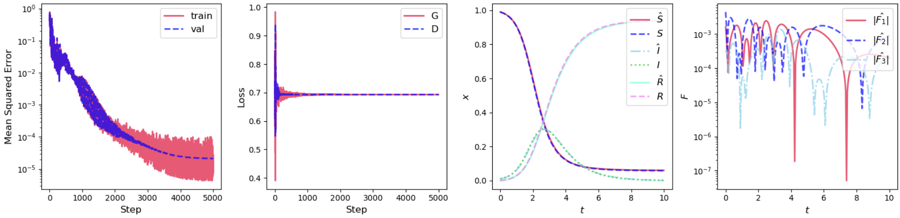
Experiment: SIR System of Equations

We set

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

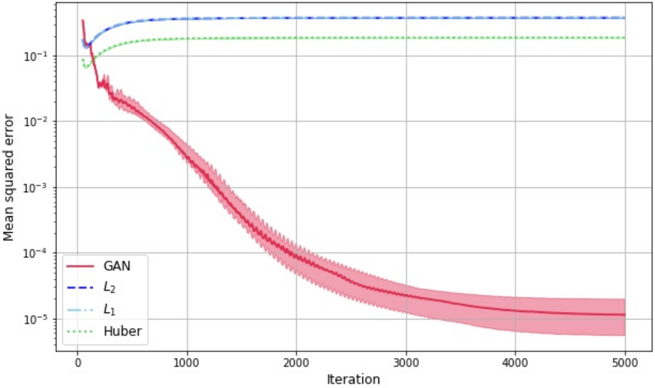
$$RHS = [0, 0, 0]^T.$$

Experiment: SIR System of Equations



- Fast convergence of G and D losses to equilibrium
- Validation mean squared error reaches $\sim 10^{-5}$
- Residuals are small for each equation

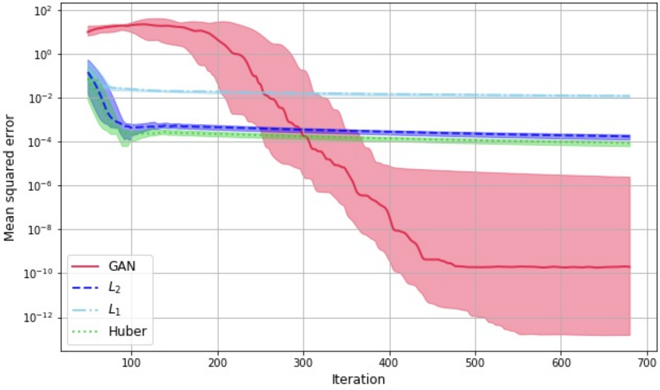
Experiment: SIR System of Equations



- DEQGAN obtains $\sim 10^{-4}$ times lower mean squared error; classic methods collapse to trivial solution (see video)

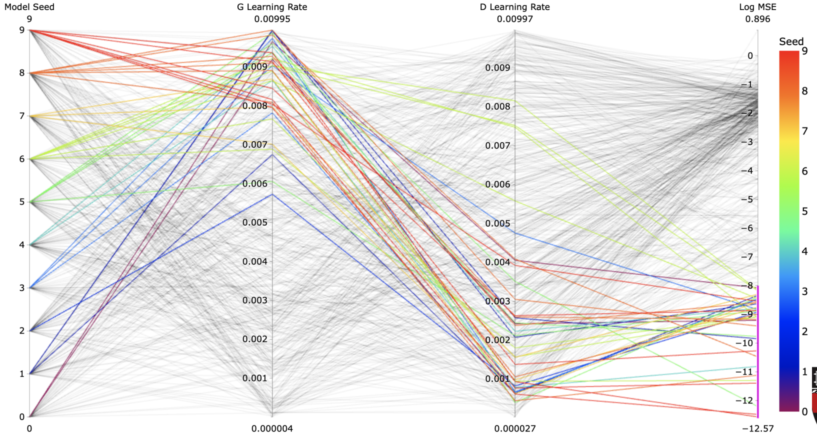
Discussion: Instability to Model Initialization

- High variability in solution accuracy when model weight initialization (either D or G, or both) not fixed (via random seed)



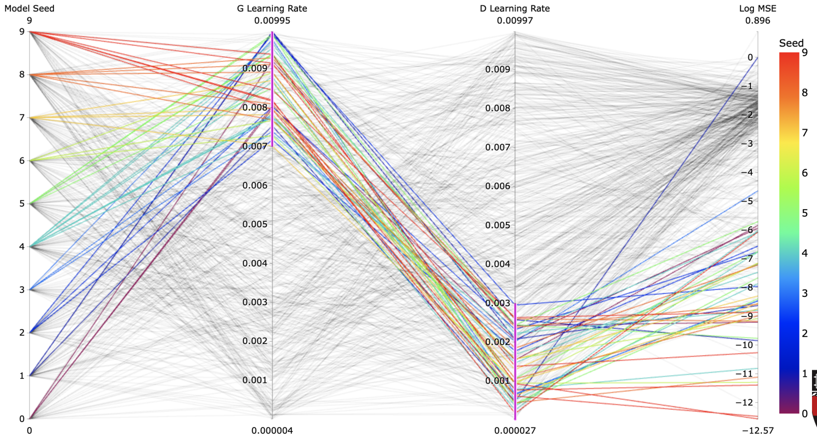
Instability: Varying Model Initialization

- Random search shows settings exist for each model weight initialization seed that perform well (filtering on MSE $\leq 10^{-8}$)



Instability: Pattern of Hyperparameters

- High generator and low discriminator learning rates mostly lead to best performance; still requires hyperparameter search



Instability: Solution

- Perform hyperparameter tuning (e.g. random search) with fixed model initialization

[6] Ray-Tune: `https://docs.ray.io/en/latest/tune.html`

Instability: Solution

- Perform hyperparameter tuning (e.g. random search) with fixed model initialization
- Leverage hyperparameter tuning schedulers (e.g. asynchronous Hyperband) to quickly and reliably find good hyperparameter settings[6]

[6] Ray-Tune: `https://docs.ray.io/en/latest/tune.html`

Discussion: Prior Formulations

For completeness, briefly mention negative results:

- Balancing: e.g. setting $LHS = \dot{x}$ and $RHS = -x$ for exponential. Fails possibly because "real" data distribution $p_{\text{data}}(x)$ changing as generator updated

Discussion: Prior Formulations

For completeness, briefly mention negative results:

- Balancing: e.g. setting $LHS = \dot{x}$ and $RHS = -x$ for exponential. Fails possibly because "real" data distribution $p_{\text{data}}(x)$ changing as generator updated
- Semi-Supervised: worse than fully unsupervised; perhaps because unsupervised solutions require adhering to equation, while supervised do not

Discussion: Prior Formulations

For completeness, briefly mention negative results:

- Balancing: e.g. setting $LHS = \dot{x}$ and $RHS = -x$ for exponential. Fails possibly because "real" data distribution $p_{\text{data}}(x)$ changing as generator updated
- Semi-Supervised: worse than fully unsupervised; perhaps because unsupervised solutions require adhering to equation, while supervised do not
- Other GAN Extensions: conditional GAN & Wasserstein GAN with gradient penalty (WGAN-GP); both sub-optimal upon reformulation and implementation of spectral normalization

Table of Contents

Motivation

- Unsupervised neural network method for differential equations is not constrained to a fixed grid of points

Motivation

- Unsupervised neural network method for differential equations is not constrained to a fixed grid of points
- Non-convex optimization procedures often benefit from introducing stochasticity (e.g. *stochastic* gradient descent); sampling can induce useful stochasticity

Motivation

- Unsupervised neural network method for differential equations is not constrained to a fixed grid of points
- Non-convex optimization procedures often benefit from introducing stochasticity (e.g. *stochastic* gradient descent); sampling can induce useful stochasticity
- Our empirical results show that the choice of sampling procedure has significant impact on convergence and accuracy

Methods

- Fixed grid: no sampling, use the same fixed set of points at each gradient step

Methods

- Fixed grid: no sampling, use the same fixed set of points at each gradient step
- Uniformly sampling: each point is sampled i.i.d. uniform with support over the domain of the problem $x \sim U(D)$

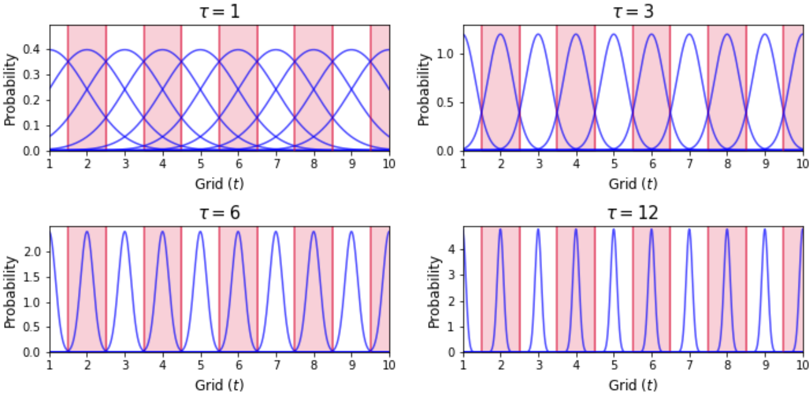
Methods

- Fixed grid: no sampling, use the same fixed set of points at each gradient step
- Uniformly sampling: each point is sampled i.i.d. uniform with support over the domain of the problem $x \sim U(D)$
- "Perturbed" sampling: "jitter" points from a fixed grid with i.i.d. Gaussian noise. For each point in the mesh, add

$$\epsilon \sim \mathcal{N}\left(\mu = 0, \sigma = \frac{\Delta x}{\tau}\right) \tag{16}$$

where Δx is the inter-point spacing, and τ is a hyperparameter that controls sample variance

Effect of Tau



Example: Reynolds-Averaged Navier Stokes

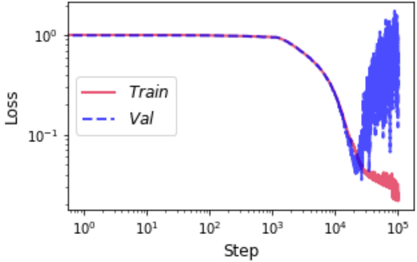
Consider the Reynolds-Averaged Navier Stokes (RANS) equation for the average velocity profile u of an incompressible fluid at position y in a one-dimensional channel given by

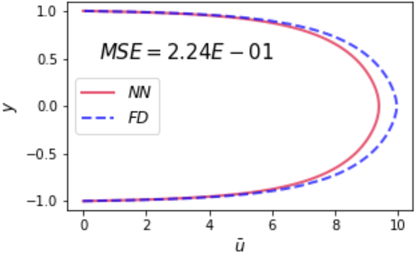
$$\nu \frac{d^2 u}{dy^2} - \frac{d}{dy}\left((\kappa y)^2 \left|\frac{du}{dy}\right| \frac{du}{dy}\right) - \frac{1}{\rho}\frac{dp}{dx} = 0 \tag{17}$$

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

Example: RANS with Fixed Grid

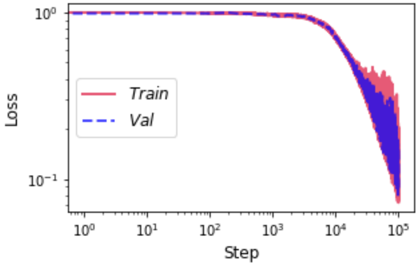
- Overfitting: validation loss diverges by step $\sim 10^4$

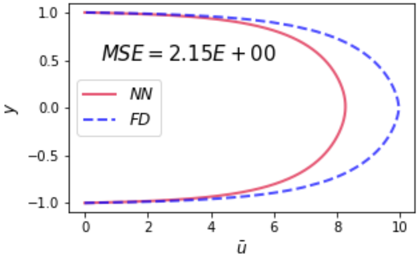




Example: RANS with Uniform Sampling

- Overfitting reduced but loss exhibits higher variance; mean squared error is higher (solution is worse)





Example: RANS with Perturbed Sampling

- Overfitting eliminated, loss variance reduced, and lowest mean squared error (best solution)

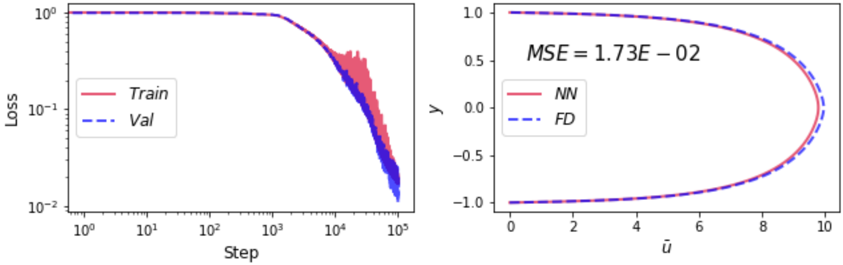


Table of Contents

Conclusion

- Introduced a new method (DEQGAN) that leverages adversarial training to *learn the loss function* for solving differential equations with unsupervised neural networks

Conclusion

- Introduced a new method (DEQGAN) that leverages adversarial training to *learn the loss function* for solving differential equations with unsupervised neural networks
- Showed that DEQGAN obtains orders of magnitude lower mean squared errors than classical unsupervised neural network methods with L_1, L_2, and Huber loss functions

Conclusion

- Introduced a new method (DEQGAN) that leverages adversarial training to *learn the loss function* for solving differential equations with unsupervised neural networks
- Showed that DEQGAN obtains orders of magnitude lower mean squared errors than classical unsupervised neural network methods with L_1, L_2, and Huber loss functions
- Provided a foundation for future work on learning the loss function for differential equations with unsupervised neural networks

Conclusion

- Introduced a new method (DEQGAN) that leverages adversarial training to *learn the loss function* for solving differential equations with unsupervised neural networks
- Showed that DEQGAN obtains orders of magnitude lower mean squared errors than classical unsupervised neural network methods with L_1, L_2, and Huber loss functions
- Provided a foundation for future work on learning the loss function for differential equations with unsupervised neural networks
- Introduced a sampling technique that yields robustness to overfitting while improving solution quality

Future Work

- Experiment with more complex, potentially stochastic, differential equations

Future Work

- Experiment with more complex, potentially stochastic, differential equations
- Conduct further robustness studies, e.g. across initial conditions and experiments

Future Work

- Experiment with more complex, potentially stochastic, differential equations
- Conduct further robustness studies, e.g. across initial conditions and experiments
- Investigate more sophisticated sampling techniques, e.g. active learning

Acknowledgements

- Dr. Pavlos Protopapas, Dr. David Sondak, Dr. Marios Mattheakis, and Dr. Cengiz Pehlevan for their guidance and support

Acknowledgements

- Dr. Pavlos Protopapas, Dr. David Sondak, Dr. Marios Mattheakis, and Dr. Cengiz Pehlevan for their guidance and support
- Harvard FAS Research Computing for computational resources

Acknowledgements

- Dr. Pavlos Protopapas, Dr. David Sondak, Dr. Marios Mattheakis, and Dr. Cengiz Pehlevan for their guidance and support
- Harvard FAS Research Computing for computational resources
- Family and friends for unconditional love and support

References

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

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

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.

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

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

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.

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

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.

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

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

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

Pediredla, A. K. & Seelamantula, C. S. (2011). A huber-loss-driven clustering technique and its application to robust cell detection in confocal microscopy images. *2011 7th International Symposium on Image and Signal Processing and Analysis (ISPA)*, (pp. 501–506).

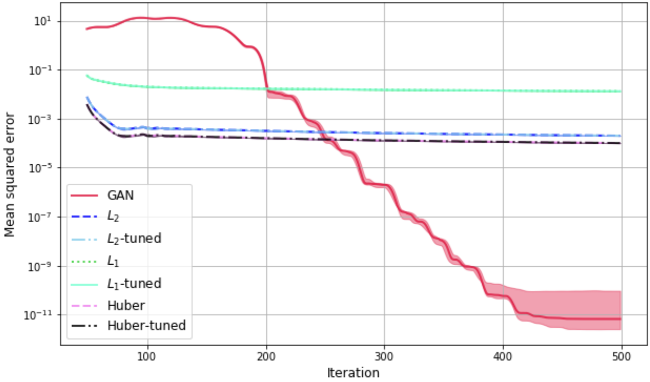
Raissi, M. (2018). Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv preprint arXiv:1804.07010*.

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

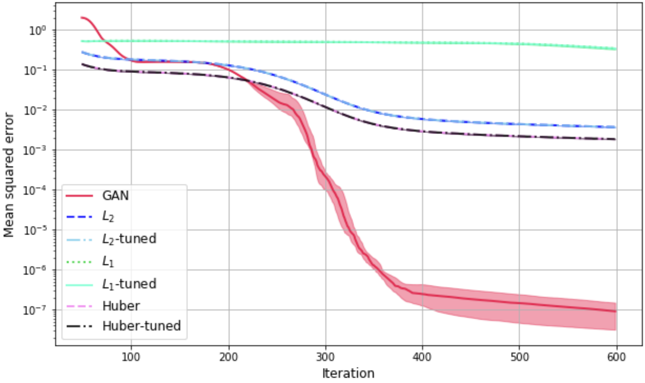
Fin

Questions?

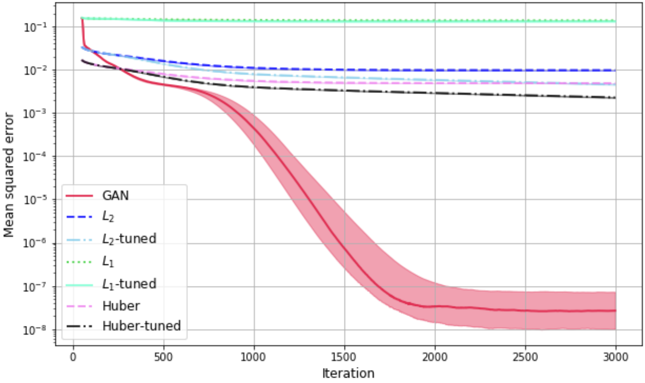
Additional Material: Exponential with Classical Tuning



Additional Material: Simple Oscillator with Classical Tuning



Additional Material: Nonlinear Oscillator with Classical Tuning



Additional Material: SIR System with Classical Tuning