Research Academy for Young Scientists

Finding Numerical Solutions to Partial Differential Equations with Deep Learning

Holger Molin
holger.molin@icloud.com

under the direction of
Björn Wehlin
KTH Royal Institute of Technology
Department of Mathematics

July 12, 2023

*

Abstract

Quasi-stationary distributions underlie a great number of fundamental phenomena and accurate modeling of the ones studied in this article will positively impact fields such as molecular biology, neuroscience, and population studies. Methods for obtaining the density of quasi-stationary distributions belonging to a class of Ornstein-Uhlenbeck processes on a bounded domain are outlined and evaluated in this paper. The goal is to find approaches that both imitate the distributions closely and minimize the number of calculations necessary, thereby making accurate approximations in higher dimensions feasible. Existing methods for this scale very poorly into higher dimensions as the computational complexity of the meshes used becomes impractical. The investigated solution attempts to, through the use of neural networks, replace the mesh with random point sampling, thereby decreasing the number of calculations needed. This problem was approached through a physics-informed neural network where several architectures, hyper-parameters, and algorithms were experimented with to optimize performance. The findings show that changing the hyper-parameter betas could improve convergence five-fold and that methods like learning rate annealing are not as successful as previously thought, at least with simpler differential equations.

Acknowledgements

I'd like to extend a sincere thank you to the organizers and participants of Rays who have given me this unique opportunity for growth and incredibly memorable summer. I'd also like to thank KPS and Beijerstiftelsen for making such a wonderful research academy possible. Most of all, I'd like to thank my mentor Björn Wehlin. Without his patience, dedication, and guidance, none of this project would have been possible and I wouldn't have gotten to engage with an as interesting topic as the one studied below.

Contents

1 Introduction

Partial differential equations (PDEs) are useful instruments for modelling complex relationships with applications in nearly all scientific fields. Often, however, they are very difficult, if not impossible, to solve analytically which leaves numerical approaches as the only viable option [1]. One such approach is the Physics-Informed Neural Network (PINN): a machine learning model that stands in for the solution of the PDE [2].

1.1 Notation

Let $\Omega \subset \mathbb{R}^d$ denote an open, connected subset of $\mathbb{R}^d$ with boundary $\partial\Omega$. Further, define $\mathcal{C}_0^2(\mathbb{R}^d)$ as the set of all twice continuously differentiable functions on $\mathbb{R}^d$ where $\{x : f(x) \neq 0\}$ is closed and bounded. The Hessian matrix of f will be represented by $\nabla^2 f$ and $\text{tr}[A]$ will denote the trace of the matrix A. Lastly θ will be used in place of the set of network parameters.

1.2 Artificial Neural Networks

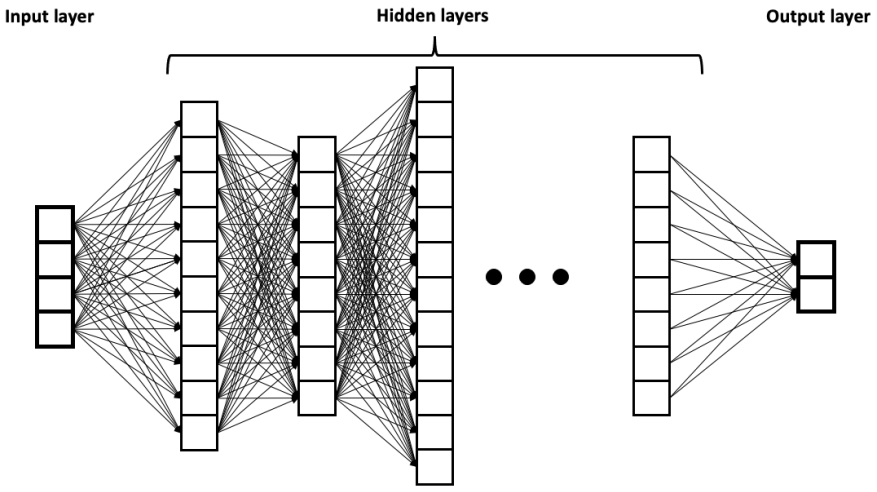

Figure 1: The Structure of a Multilayer Feed-Forward Network [3].

Inspired by biological neural networks, an artificial neural network (ANN) is a system of layers of nodes and connections. Typical for an ANN is having an input layer where a value is assigned to each input node and then sent to the second, so called "hidden," layer where the value of the new signal is given as

$$\text{signal} = b + \sum_{i=1}^{n} w_i x_i. \tag{1}$$

Here, n is the number of input nodes connected to the node in question, b is a constant known as the bias, x_i is the input signal from the ith node, and w_i is the corresponding coefficient, known as the weight, of that signal. Before this signal is passed onto the next node, it is generally put through an activation function. This is chosen as part of the neural network architecture and is often a non-linear function that prevents the ANN from collapsing into a linear regression. If the activation function is omitted, the system can be simplified to a linear sum with a bias similar to Equation (1) which limits the set of relationships that can be approximated to the linear ones. When introducing non-linearity into the signal, however, this simplification is no longer possible which allows the ANN to model a greater number of systems. This process is then repeated for the remaining hidden layers. When the output layer is reached, a final signal is generated for each output node which serves as the ANN's prediction. As all functions within the network can be chosen to be differentiable, so too can the network as a whole be differentiated which has some important consequences [4].

For the ANN to yield interesting outputs, it has to be fitted to the problem. This is done through back-propagation followed by gradient descent [5]. Back-propagation is where the loss function, a measure of how much the ANN's output differs from the correct one, is differentiated with respect to the network parameters. Then, in the optimization phase, the parameters are shifted to progress towards the minimum of the loss function. Repeatedly doing this across a large set of data is what adapts the weight and bias parameters of the network to the problem in question and allows the system to make accurate predictions on previously unseen inputs.

1.3 Physics-Informed Neural Networks

Due to the adaptability of machine learning models, almost any function can be approximated with a neural network [6]. A Physics-Informed Neural Network (PINN) [2] is a framework for approximating the solutions to PDEs with deep learning. The idea is that,

though the underlying function is unknown, adequate information to train a model can be extracted from the relationship between the function and its derivatives as well as from the boundary conditions and other analytically derived properties of the solution like probability mass. A function that fulfills these properties sufficiently closely, will also represent the solution of the PDE [**?**].

For instance, if the differential operator N acts on the function f such that

$$Nf(x) = 0. \tag{2}$$

If it is additionally known that the output of f should be zero on the boundary, one can construct a set of loss functions

$$\text{Operator loss} = \|Nf(x)\|, x \in \Omega \tag{3}$$

$$\text{Boundary loss} = \|f(x)\|, x \in \partial\Omega. \tag{4}$$

The neural network can then try to minimize these losses, thereby emulating the behaviour of the sough-after function f without having to know it explicitly as long as $Nf(x)$ is known.

A major advantage of the PINN over other methods, is that it does not require the construction of a mesh which can be both difficult to design and computationally impractical for higher dimensional PDEs. Instead it can take samples from the function's input space in a process which requires fewer calculations [7].

1.4 Quasi-Stationary Distributions

A quasi-stationary distribution (QSD) is a property of a certain type of stochastic process; namely one that has at least one condition, referred to as an absorbing state, that terminates the process and is reached almost surely[1]. This definition makes no assertions

[1]The technical definition of "almost surely" permits the existence of non-conforming cases but demands that the probability of these be 0.

as to when the absorbing state occurs, only that it does which means that one can study the behaviour of the process as time tends towards infinity, given that the absorbing state has not yet occurred. Through this, a distribution describing the likelihood of the process taking a certain range of values within the domain can be found, which is what defines a QSD.

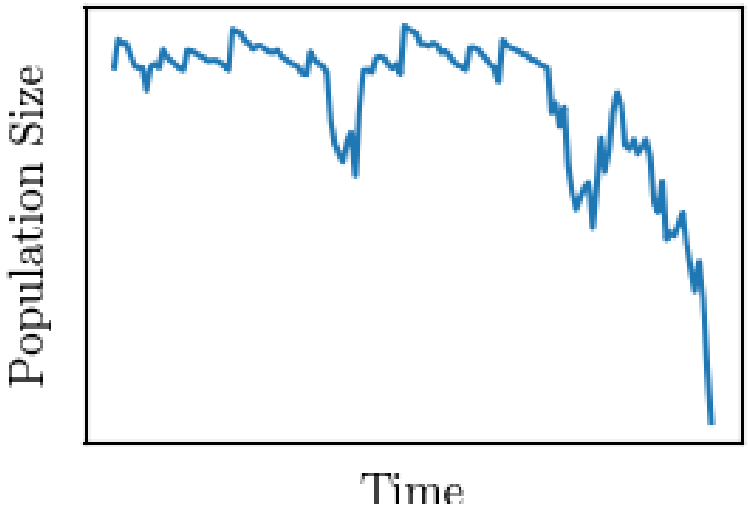

Figure 2: Population as a stochastic process that could be described with a QSD where the absorbing state is population extinction.

A typical example of such a stochastic process is a population in an environment with limited resources; there can be long periods of stability but, as there is a non-zero chance of extinction at any given time, eventually all individuals will die off, terminating the population process. The QSD of this process would then be the function describing the likelihood of the population consisting of certain range of individuals.

1.5 Presentation of the SDE

The class of equations studied in this paper is the set of stochastic differential equations (SDEs) described as

$$\mathrm{d}X_t = -b(X_t)\mathrm{d}t + \mathrm{d}W_t, \tag{5}$$

where $X_t \in \mathbb{R}^n$ is a Markov process, $b(x)$ is an arbitrary vector-valued function, and W_t is

a stochastic process. Here, X_t is most easily conceptualized as a particle moving through a confined space and which is terminated if it collides with the boundary of said space as illustrated in Figure 3. The equation can then be interpreted as saying that the change in the particle's position is dependent on some position-specific function and the change in some stochastic process. The sought-after function describes the QSD density of the particle being within a given region of the domain.

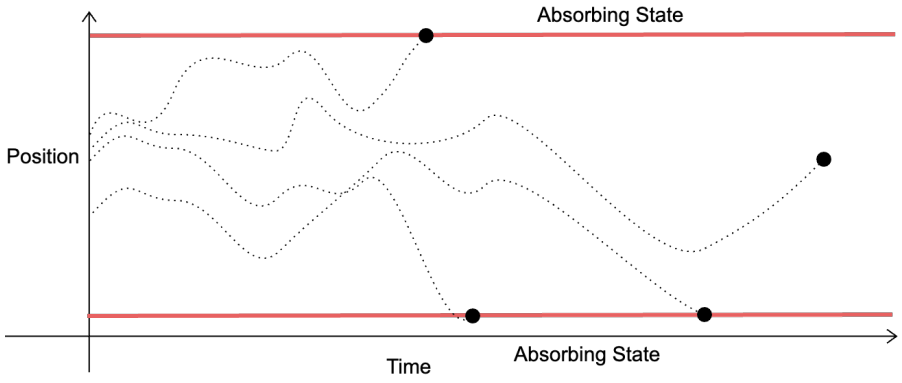

Figure 3: Particles moving stochastically through a one dimensional room with absorbing states on the room boundaries.

1.6 Aim of Study

The aim of this study is to find a general solution method for the QSDs associated with equations on the form of Equation (5) with sufficient accuracy and with adequate speed so that the solution can feasibly be found even for more complicated cases.

Accurate modelling of quasi-stationary distributions would contribute greatly to fields such as population modeling [8], molecular dynamics [9], and neuroscience [10].

1.7 The Loss Function

As has been shown by Budhiraja et al. [11], the probability density solution function of equation (5) fulfills the properties

$$-\mathcal{L}^* f(x) = \lambda f(x), x \in \Omega \tag{6}$$

$$f(x) = 0, x \in \partial\Omega \tag{7}$$

where $f \in \mathcal{C}_0^2(\mathbb{R}^d)$, λ is an unknown constant that the network can optimize and $\mathcal{L}^*$ is a differential operator described as [11]:

$$\mathcal{L}^* f(x) = \beta(x) \cdot \nabla f(x) + \frac{1}{2}\text{tr}[a(x)\nabla^2 f(x)] - c(x)f(x), \tag{8}$$

with

$$\beta_i(x) = -b_i(x) + \sum_{j=1}^{d}(a_{ij}(x))_{x_{ij}},$$

$$c(x) = -\frac{1}{2}\sum_{i=1}^{d}(a_{ij}(x))_{x_i x_j} + \sum_{i=1}^{d}(b_i(x)_{x_i}.$$

For this project, $b_i(x)$ is the ith element of the arbitrary vector-valued function $b(x)$ and $a(x)$ is the identity matrix. This means that Equation (8) can be simplified to

$$\mathcal{L}^* f(x) = -b(x) \cdot \nabla f(x) + \frac{1}{2}\text{tr}[\nabla^2 f(x)] - \sum_{i=1}^{d}(b_i(x))_{x_i} f(x). \tag{9}$$

This is very useful as (6) allows for the construction of a deterministic PDE-based loss function involving only known terms and constants as well as optimizable parameters. Namely,

$$\text{PDE loss} = \|\mathcal{L}^* f(x) + \lambda f(x)\|. \tag{10}$$

By training a PINN to minimize this loss, a good approximation of f can be generated.

1.8 Multi-Objective Optimization

Neural networks are effective optimizers which means that, in the instance of Equation (10), they will converge towards the perfect solution $f(x) = 0$. As this isn't a useful result, standard practice is to introduce another loss function that punishes the ANN if, for example, the total probability mass strays too far from one. This gives rise to another

problem as these loss functions need to be combined into one so that gradient descent can be performed. Typically, this is done by defining a total loss as the weighted sum of all loss functions. Balancing these weights for optimal performance can then be done in several ways. Manual adjustments work but are often tedious and ineffective.

An alternative method suggested by Bischof and Kraus [12] is learning rate annealing. A loss weight $w_i(t)$ is updated every iteration by:

$$w_i(t) = \alpha \cdot w_i(t-1) + (1-\alpha) \cdot \hat{w}_i(t) \tag{11}$$

where

$$\hat{w}_i(t) = \frac{\max\{|\nabla_\theta \mathcal{L}_\Omega(t)|\}}{\overline{|\nabla_\theta \mathcal{L}_{\Gamma_i}(t)|}}, i \in \{1, 2\} \tag{12}$$

and $\alpha \in [0, 1]$ is a constant affecting the speed of weight change, $\mathcal{L}_\Omega$ is PDE Loss, and $\mathcal{L}_{\Gamma_i} \in \{\text{Boundary Loss}, \text{Mass Loss}\}$. This means that the weight is updated based on how the maximum component of PDE loss gradient vector relates to the mean component of the loss that the weight belongs to. A large such relationship implies that gradient descent is going to disproportionately focus on decreasing the PDE loss which learning rate annealing corrects for by increasing the weight of the loss function in question, thereby ensuring that all losses are considered equally in gradient descent.

1.9 Fleming-Viot Simulation

An alternative method of modelling SDEs to PINNs is a so called Fleming-Viot simulation. Instead of learning an analytically derived loss function, this leverages the particle-interpretation of the quasi-stationary distribution. A number of particles is distributed uniformly through the domain and an iterative process much like the Euler–Maruyama step method is initiated. The new position of each particle after n iterations is given recursively as

$$X_n = X_{n-1} - b(X_{n-1})\Delta t + \Delta W_n,$$

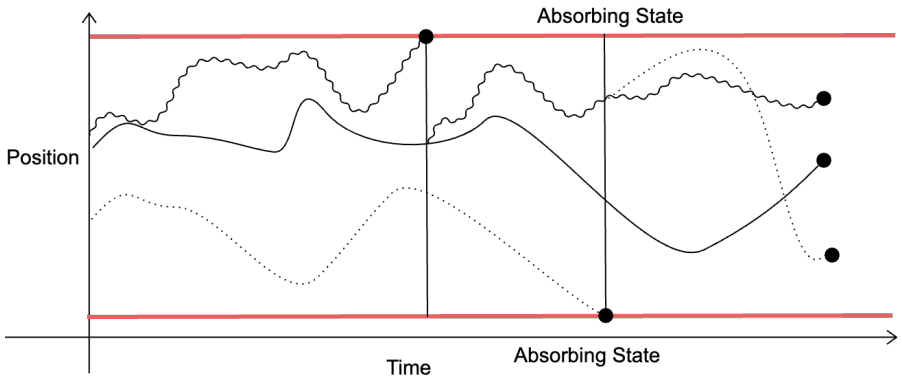

Figure 4: A Fleming-Viot simulation of three particles that are restarted in the position of another particle when absorbed.

where X_n is the position of a particle moving according to Equation (5) at time $= n$ $\Delta W_n = W_n - W_{n-1}$ is a normal random variable with mean 0 and variance Δt. If a particle reaches the boundary, it is killed and a new one is initiated in the same place as one of the surviving ones, chosen with uniform probability among the remaining points. If the position of all particles is recorded at each time interval, a histogram can be developed that can then be converted into a function, for example through Gaussian Kernel density estimation [13], which describes the probability density of the QSD [14].

1.10 Long Short-Term Memory Network

Aside from the standard feed-forward neural network (FFNN) described above, there are many other network structures that have been effectively used to solve differential equations numerically. One such architecture is the long short-term network (LSTM) used by Sirignano and Spiliopoulos due to its discontinuous derivatives in certain points which allows the network to make sharper turns [7]. This can, for instance, improve the ANN's ability to meet boundary conditions.

1.11 Some Interesting Hyper-Parameters

In the following section, some particularly useful hyper-parameters are defined.

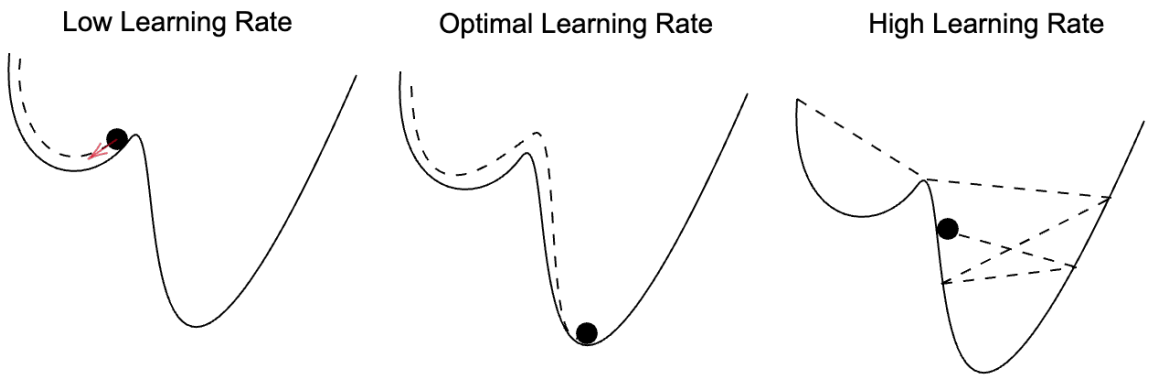

Figure 5: Effects of varyingly well-defined learning rates

Learning Rate The learning rate of an ANN is the speed with which it follows a gradient. This means that a network with a high learning rate will adjust its parameters more after every iteration. This, as shown by Figure 5, is a balancing act as a too high learning rate can lead to unstable behavior and hinder convergence whereas a slow learning rate leads to a longer training process and risks trapping the network in a sub-optimal local minimum [15].

Learning Rate Schedulers Broadly, a learning rate scheduler is a protocol for dynamically adapting the learning rate of a neural network during the training process. Most relevant to this investigation is the reduce on plateau method wherein the learning rate decreases if the loss function hasn't decreased in a set number of training epochs, known as the patience. This allows the system to train quickly in the beginning and to find the approximate location of the global minimum and then fine-adjust when the major improvements have been made.

Betas The used optimizer for this project is AdamW [16] which, aside from loss function, step size, and initial parameters, takes two values: β_1 and β_2 [17]. The exact interpretation of these varies depending on the problem but, in general, they can be thought of as keeping a moving average of momentum and velocity over a number of generations determined by

their size. A higher value corresponds to a longer moving average.

Weight Decay Weight decay is a method used to prevent the network parameters from growing too large. This can help avoid unnecessary complexity as well as overfitting [18].

2 Method

For the experiments conducted in this investigation, the variation

$$\mathrm{d}X_t = -X_t \mathrm{d}t + \mathrm{d}W_t \tag{13}$$

of equation (5) is used where X_t moves through a two-dimensional space and W is standard Brownian motion.

2.1 The Loss Function

Three main loss functions were used to inform the PINN's development. Namely the eigenvalue-based PDE loss, mass loss, and boundary loss [19].

Mass loss was defined as follows:

$$\text{Mass loss} = \left\| \int_\Omega f(x)dx - 1 \right\| \tag{14}$$

and used to ensure that the solution met the mass requirement for a probability density function and the trivial solution $f(x) = 0$ was avoided.

Boundary loss was defined as:

$$\text{Boundary loss } = \left\| \int_{\partial\Omega} f(x)dx \right\|. \tag{15}$$

QSDs are conditioned on the process terminating at the border meaning that the probability of the particle being there must be zero which this loss function requires. Practically, these integrals were calculated as the area of the domain multiplied by the average of a set of uniformly distributed sample points of the domain.

They were then combined into a holistic loss function on the form:

$$\text{Loss} = \text{PDE Loss} + w_1 \cdot \text{Density Loss} + w_2 \cdot \text{Boundary Loss} \tag{16}$$

with $w_1, w_2 \in \mathbb{R}$. This holistic function was the one used for back-propagation but, for the sake of consistency, all losses were looked at independently and without weight when evaluating the models.

2.2 Design Variations

Through the process of approximating the PDE, several hyper-parameters, algorithms, and network architectures were experimented with. In regards to network architecture both the typical feed-forward network and a LSTM network were tested. Additionally hyperparameters like Betas, weight decay, and the learning rate were adjusted. Finally learning rate annealing was tested.

When testing the features above, all other testing parameters were set to their default values in PyTorch or disabled if that was an option.

2.3 Network Specifications

Unless it is the aspect being experimented with during training, this section describes specifications of all networks trained for this investigation. All networks, including LSTM and feed-forward, used 3 hidden layers with 64 nodes each and $\tanh(x)$ as the activation function. The total number of points used was $2^{14} = 16,384$ both on the domain and boundary. The selected optimizer was AdamW with a learning rate of 10^{-2} and an on-plateau learning rate scheduler with a patience of 200 epochs and a factor of 0.9. For the specifics of implementation `https://github.com/HolgerMolin/Rays_QSD_Approximation.git` contains the code used for this project.

2.4 Benchmarking

When evaluating the models against one another, the unweighted loss functions were considered and optimized against. Then, to verify the effectiveness of the proposed model, a Fleming-Viot simulation of the system was created. The model's predicted probability distribution was then compared to the result of Fleming-Viot simulation to ensure that the loss functions' outputs actually actually correspond to real errors and to determine how precisely how accurate the model was.

3 Results

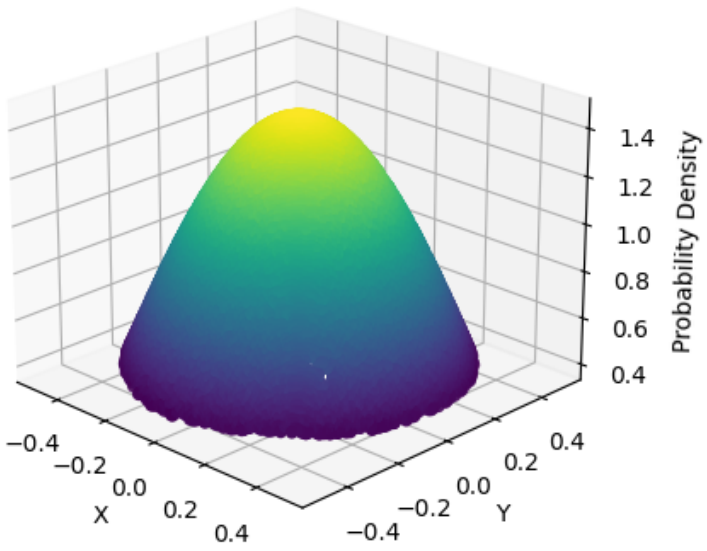

Figure 6: The PINN output. Here, x and y are spacial coordinates z is the probability density associated with each point.

Network Output Figure 6 shows an approximate solution of the probability density function associated with Equation (13) and Figure 7 shows its corresponding Fleming-Viot simulation.

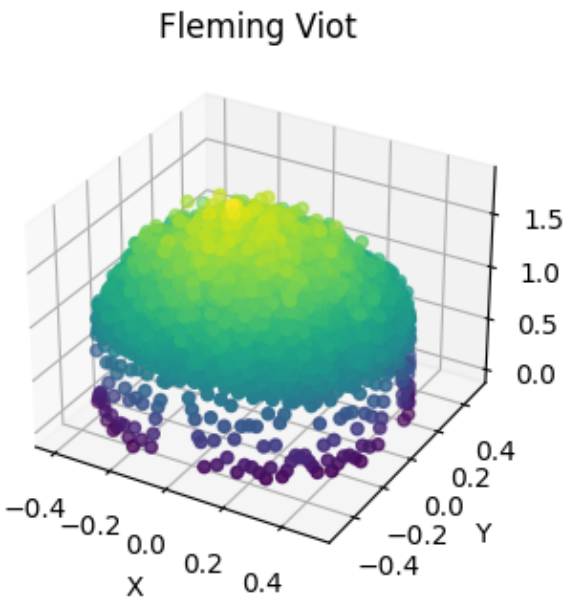

Figure 7: Fleming-Viot simulation output. Once again, x and y represent coordinates in a two-dimensional space and z is the probability density of each point.

Learning Rate Annealing In the two dimensional case, learning rate annealing positively impacts the speed of convergence significantly but this comes at the cost of accuracy as the converged error is 80% higher. It should, however, be noted that this error is significantly less than in the one dimensional case where the NN with learning rate annealing enabled had an error approximately 208% higher than the one without.

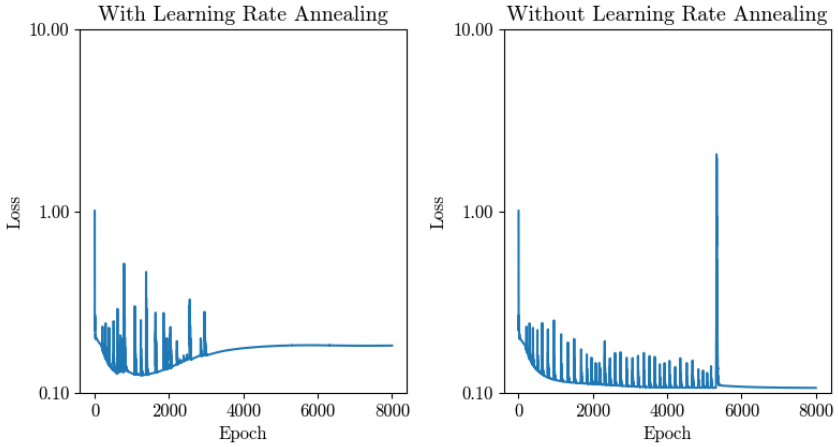

Figure 8: Holistic loss over 8000 epochs of learning with and without learning rate annealing enabled. $\alpha = 0.9$ was used when using learning rate annealing.

Weight Decay The introduction of weight decay had a negligible impact on accuracy but reduced the number of iterations required to converge by about 1000.

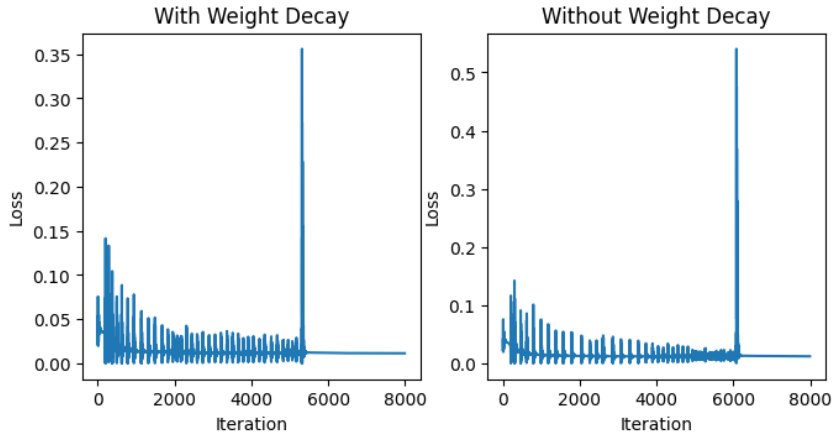

Figure 9: Holistic loss over 8000 iterations of learning with and without weight decay enabled.

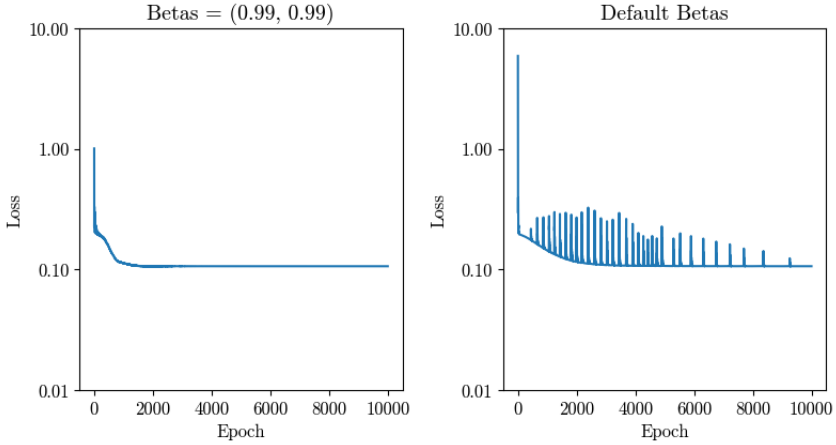

Figure 10: Loss sum over 10000 epochs of learning with and without betas enabled.

Betas Specifying betas to (0.99. 0.99) as recommended by [20] resulted in the PINN converging after around 1500 iterations instead of 10000 and an 23% improvement in loss.

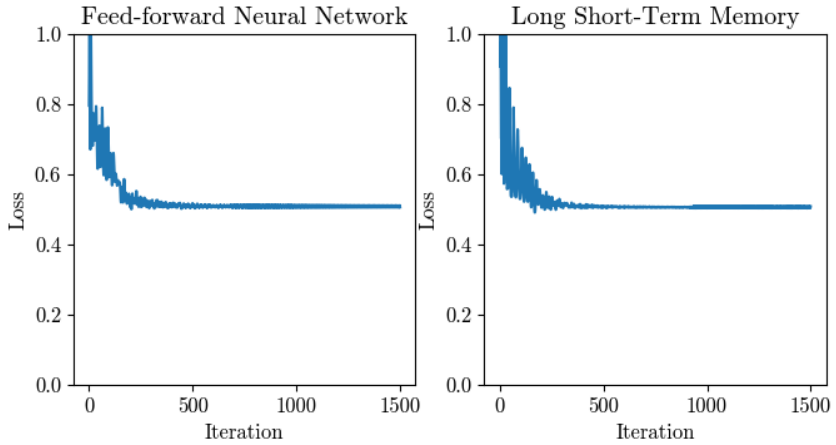

Figure 11: Comparison of summed losses between an FFNN and an LSTM architecture o.

Network Architectures The FFNN was slightly more stable during training than the LSTM but the final loss only differed by 3% in favor of the FFNN and the number of epochs for convergence differed marginally.

4 Discussion

When compared to the Fleming-Viot simulation, the average output difference at the evaluated points was 0.0526 which, while far from optimal, is enough that the output can be considered significant given the range is between 0 and 1.5 and, thus, has a margin of error around 3%.

When compared to the Fleming-Viot simulation, the average output difference at the center of each histogram bucket was 0.0526 which, while not perfect, represents an error of only about 3% as the range varies between 0 and 1.5. It should be noted that this error may very well be lower but the Fleming-Viot simulation is quite slow and obtaining fully converged model without outliers that bring the average error up has been difficult, pointing to machine learning methods as a viable alternative to Fleming-Viot. Similarly, the difference in graph appearance can be explained by the fact that modeling with high resolution, especially around the border where the gradient is higher and the points are more spread out, is outside of the scope of this article.

4.1 Learning Rate Annealing

On the tested QSD, learning rate annealing had a clear, negative impact on the loss function. Though this has yet to be tested, it could be worthwhile investigating further how learning rate annealing functions with higher dimensional SDEs and if the increased complexity makes annealing more useful. This is supported by the fact that when implementing learning rate annealing in the one dimensional case, the error is greater by a factor of 2.5

4.2 Network Architecture

The convergence of the FFNN was marginally better than that of the LSTM and the difference in converged loss was small enough that it can not be considered significant. Therefore, it can be concluded that, for the purpose of modeling relatively simple QSDs, the FFNN and LSTM are equally effective for loss optimization.

4.3 Betas

By changing betas from its default of $\beta_1 = 0.9, \beta_2 = 0.999$ to $\beta_1 = 0.99, \beta_2 = 0.99$, convergence was reached significantly sooner. As stated in the introduction, these values determine what number of previous iterations to keep a moving average of. The resulting loss change was somewhat small but the number of iterations needed for convergence decreased by a factor of five. For the two-dimensional case, this is not major but for higher dimensional cases where computational complexity becomes a limiting factor, this could be the difference between feasibly being able to approximate a QSD and not.

4.4 Conclusion and Further Studies

Based on the results from this study, some recommendations for future PINN modelling can be made. Regarding β_1 and β_2, it is clear that small alterations can have major impacts on performance. As default parameters, $\beta_1 = 0.99, \beta_2 = 0.99$ are suggested but further optimization should be made on a project-specific basis. Weight decay is also suggested for improved convergence. 0.001 worked quite well for this project but experimentation is advised.

Learning rate annealing is not recommended for simple problems such as the one investigated here but a more thorough investigation into its effectiveness in higher dimensional PDEs with more complex behaviour could yield useful results. Similarly, it is unclear whether LSTM is better or worse than the standard feed-forward method meaning that further investigations on a wider array of SDEs should be conducted to give clearer results.

References

[1] Z. Hao, J. Yao, C. Su, H. Su, Z. Wang, F. Lu, Z. Xia, Y. Zhang, S. Liu, L. Lu, and J. Zhu, "Pinnacle: A comprehensive benchmark of physics-informed neural networks for solving pdes," 2023.

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

[3] W. Commons, "Example of a deep neural network," 2021. [Online; accessed 12 July, 2023].

[4] G. P. Leonardi and M. Spallanzani, "Analytical aspects of non-differentiable neural networks," 2020.

[5] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[6] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[7] J. Sirignano and K. Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, pp. 1339–1364, dec 2018.

[8] S. Mé léard and D. Villemonais, "Quasi-stationary distributions and population processes," *Probability Surveys*, vol. 9, jan 2012.

[9] T. Lelièvre, M. Ramil, and J. Reygner, "Quasi-stationary distribution for the langevin process in cylindrical domains, part i: existence, uniqueness and long-time convergence," 2021.

[10] N. Berglund and D. Landon, "Mixed-mode oscillations and interspike interval statistics in the stochastic FitzHugh–nagumo model," *Nonlinearity*, vol. 25, pp. 2303–2335, jul 2012.

[11] A. Budhiraja, P. Dupuis, P. Nyquist, and G.-J. Wu, "Quasistationary distributions and ergodic control problems," 2021.

[12] R. Bischof and M. Kraus, "Multi-objective loss balancing for physics-informed deep learning," 2021.

[13] Weglarczyk, Stanislaw, "Kernel density estimation and its application," *ITM Web Conf.*, vol. 23, p. 00037, 2018.

[14] S. Mé léard and D. Villemonais, "Quasi-stationary distributions and population processes," *Probability Surveys*, vol. 9, jan 2012.

[15] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay," 2018.

[16] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[18] A. Krogh and J. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems* (J. Moody, S. Hanson, and R. Lippmann, eds.), vol. 4, Morgan-Kaufmann, 1991.

[19] B. Wehlin, "Approximating quasistationary distributions using deep learning (master's thesis)," 2022.

[20] J. Yao, C. Su, Z. Hao, S. Liu, H. Su, and J. Zhu, "Multiadam: Parameter-wise scale-invariant optimizer for multiscale training of physics-informed neural networks," *arXiv preprint arXiv:2306.02816*, 2023.