FAST, CONVEX AND CONDITIONED SINGLE-LAYER NETWORK FOR LEARNING MULTI-FIDELITY UNIVARIATE DATA AND LINEAR DIFFERENTIAL EQUATIONS

Siddharth Rout
Institute of Applied Mathematics
University of British Columbia
Vancouver, BC, Canada
siddharth.rout@ubc.ca

ABSTRACT

Accuracy in neural PDE solvers often breaks down not because of limited expressivity, but due to poor optimisation caused by ill-conditioning- especially in multi-fidelity and stiff problems. We study this issue in Physics-Informed Extreme Learning Machines (PIELMs), a convex variant of neural PDE solvers, and show that asymptotic components in governing equations can produce highly ill-conditioned activation matrices, severely limiting convergence. We introduce Shifted Gaussian Encoding, a simple yet effective activation filtering step that increases matrix rank and expressivity while preserving convexity. Our method extends the solvable range of Peclet numbers in steady advection-diffusion equations by over two orders of magnitude, achieves up to six orders lower error on multi-frequency function learning, and fits high-fidelity image vectors more accurately and faster than deep networks with over a million parameters. This work highlights that conditioning, not depth, is often the bottleneck in scientific neural solvers and that simple architectural changes can unlock substantial gains.

1 INTRODUCTION

Neural networks have demonstrated remarkable versatility and effectiveness across a broad range of applications, including images and audios (39; 35; 42), natural language processing (42; 10; 52), and complex scientific modelling (53; 12; 41). Despite these advancements, these powerful networks fail to impress the scientific community. For an instance, some simple mathematical or physical symbolic functions cannot be learned by deep neural networks (36). Key issues arise while fitting to intricate data patterns, chaotic dynamics (11), addressing differential equations (DEs) with varying orders of terms (46; 2), ensuring stable training (18), and overcoming optimisation stiffness (46). When it comes to accurate prediction for large-scale scientific applications such as weather prediction and particularly for prediction of rare events such as heat waves, droughts, tornadoes, and cyclones, the inherent complex and chaotic high-dimensional nonlinear dynamics become very difficult to learn from.

1.1 MOTIVATION

1.1.1 NEURAL DIFFERENTIAL EQUATION SOLVERS

Solving differential equations is fundamental to many scientific and engineering fields, underpinning the modelling of phenomena in fluid dynamics, structural analysis, climate science, and finance. Traditional numerical methods—such as finite element, finite difference, and spectral methods—are well-established for their robustness and accuracy. However, they can be computationally intensive and often struggle with high-dimensional or complex problems. The emergence of automatic differentiation (7) and high-performance computing frameworks (17; 1; 40) has enabled deep learning

(32) to offer alternative approaches. These methods harness the function approximation capabilities of neural networks to address some of the limitations of classical techniques.

Physics-Informed Neural Networks (PINNs)—based on techniques developed decades ago (14; 31; 43)—have recently emerged as a powerful framework for solving differential equations. By embedding physical laws directly into the neural network's loss function, PINNs try to ensure that their predictions are consistent with the governing equations of the system. Unlike conventional machine learning methods, PINNs do not require training data and eliminate the need for specialised numerical schemes, meshing, or discretisation. This effectively recasts the problem of solving partial differential equations (PDEs) as a data-free optimisation task, where the solution is represented by a neural network that approximates the true function. Essentially, PINNs pose solving PDEs as optimisation problems. Hence, the exactness of the solution is dependent on how easy the optimisation problem is, and hence, proof of convergence is not guaranteed.

The use of deep neural networks in this context offers several key advantages. Their universal approximation capability allows them to represent complex, nonlinear functions with high accuracy (13; 23; 22). Once trained, these networks can produce solutions in real time, significantly reducing computational costs compared to traditional iterative solvers. Moreover, their flexibility enables them to handle noisy inputs and complex boundary conditions more effectively. Unlike traditional numerical methods, which solve discretised versions of the original equations and whose accuracy depends heavily on the number of collocation points, PINNs solve the exact differential equation. As a result, the accuracy of the solution is less dependent on the density of collocation points, especially when the target function is smooth.

PINNs have demonstrated their utility across various domains, including biomedics (3; 48), fluid mechanics (44; 9), uncertainty quantification (58; 57), and many more. Most of these successful applications are inverse problems. The method for forward problems is a bit tricky. The generalizability of the method is not questionable. However, when it comes to competing with the robustness of traditional methods, PINNs are lagging far behind (19). Rout (46) shows that though neural networks are boasted for their strong and universal approximation capabilities which strengthen with increasing depth and width of a network, a small and shallow network could fit to the data points denoting the solution of an extremely advection-dominated ODE, but could not train itself to the loss function generating exact gradients from the governing ODE to update the weights. The depth and width of the network had a minimal role in the convergence of the loss function. Training PINNs is in fact an optimisation problem and hence, can be challenging, especially for applied problems which often involve sharp gradients, multiple constraints, and high frequencies. Rout (46) emphasizes on advection-dominated PDEs while highlighting some of these issues and demonstrating ways to tackle them. They essentially adopt various optimisation tricks to reduce stiffness and optimise better. Some of the tricks, like piecewise approximation, weighted loss terms, and step/gradual optimisation, can be seen as effective. Stiff optimisation is a special characteristic evident in PINNs, unlike typical deep learning inverse problems. Wang et al. (56) also show that the use of weighted loss terms is very effective in reducing the stiffness of the problem. Also for the same reason, most of the work using PINNs for forward problems (43; 19; 50; 29; 49) prefer a second-order optimiser specifically, L-BFGS. McClenny and Braga-Neto (37) use an attention mechanism to relax the stiffness. As the field of scientific machine learning continues to evolve, addressing these challenges will be crucial for further advancing the capabilities and applications of neural networks in solving complex PDEs across diverse scientific and engineering disciplines.

1.1.2 Convexity of Networks

Neural networks can be very non-convex. LeCun et al. (33) discussed the series of problems they faced while training a network, practically arising due to the non-convexity. Networks can come in many shapes and sizes, for which the use of higher-order optimisation methods may not look reasonable. This sparked interest among the computing community to explore convex surrogates or relaxations. Bengio et al. (8) characterised the optimisation problem for NNs as a convex program if the output loss function is convex in the NN output and if the output layer weights are regularised by a convex penalty. Bach (6) studied connections between neural networks and convex optimisation by relating infinitely wide single-layer networks to kernel methods, which are convex in nature. Some later breakthroughs are primarily based on the ideas of shallow network and/or kernel method. Hazan et al. (20) demonstrated that certain classes of shallow neural networks could be trained using

convex optimisation frameworks in online settings, motivating new algorithms. The recent breakthroughs have been linking convex optimisation tools to network learning. Jacot et al. (28) proposed the neural tangent kernel, where the gradients of weights are mapped to those of the kernel method, which is known to be convex. Bach (5) discussed how neural networks can exploit structured convex sets to simplify high-dimensional problems, further linking convex optimisation to modern deep learning. While Ergen and Pilanci (16) connected two-layer neural networks to convex optimisation through convex duality. Essentially, the aim has been to bridge neural networks with classic convex optimisation tools.

In terms of approximation strength, a point to note is that the approximation ability of a single-layer network is also universal (13; 6). It seems sensible to simplify the network to understand the formulation of the cost function and not always think about a deep network. If the number of weights is big enough to provide a sufficient number of superpositions, as per (13), then we are done. And, quite often, the required number of weights to express a function or set is much less than what we use, yet still the network does not fit. Rout (46) in chapter 2 provides an example of such a case with some interesting results, which is also discussed in section 5.2. The exact solution to the ODE problem mentioned there can be represented by a single-layered network with only two nodes. Essentially, we know the values the weights should converge to. However, on optimising the network, it can be noticed that it is not that easy to get the exact weights, especially in cases where advection dominates the diffusion more. Rout (46) in chapter 3 demonstrated how if the loss function is a differential expression of the network output, the optimisation can get even tougher. In both the problem setups, the network is supposed to have learned the same function. With the advent of automatic differentiation (7), many algorithms have gained popularity under the name umbrella name physics-informed neural networks (12). So, it is just to state that training a neural network can pose cumbersome optimisation problems because of how we pose the problem. Specifically, the training task should not always be about what the expressivity of a network is but also rather about how well the loss function can be optimised. A significant number of serious reasons are mentioned in the subsection section 5.1 to push us to study the optimisation aspect of PDE problems.

For the reason described in the previous section, we choose a particular kind of neural networks called extreme learning machines (ELM) (26). The speciality of this algorithm is its simplicity. To convert a network to ELM, we just fix all the weights and biases except those in the last layer. With this assumption, the network output becomes linear in terms of weights and biases. Hence, the training of this algorithm is solving a system of linear equations, which can be easily solved in a single step by Moore-Penrose generalised inverse (45). The good part is the convexity of this formulation, which is proved in theorem 3.1.

1.1.3 Ill-conditioning of Networks

A network architecture can be called better than others if it can learn the task more, as in the approximation is closer to the exact solution, and/or it can learn faster. Training neural networks is a game of function approximation capability and optimisation. Neural networks, by default, are non-convex formulations (16). Various studies have highlighted various obstacles and ill-conditionings present in the training process of deep networks (18). Hence, several attempts have been made over the decades to address such issues. Saarinen et al. (47) showed that feedforward neural networks can easily have ill-conditioned Hessians and concluded that many network training problems are ill-conditioned and may not be solved more efficiently by higher-order optimisation methods. Smagt and Hirzinger (55) showed that the ill-conditioning is because of the structure of the network, hence presented a less popular modified structure of the network. In comparatively recent advancements along the line, popular have been by various normalizations procedures (27; 4; 54), like batch norm, layer norm and instance norm, to allow inputs to the active region of activation functions in each layer. Some typical options to tackle such problems are the use of higher-order optimisers, regularisations, adaptive hyperparameterization, etc. Li et al. (34) relied upon gradient descent powered by Langevin dynamics to act as a stochastic preconditioner. Ill-conditioning can severely impact the efficiency and effectiveness of training, leading to slow convergence and sub-optimal performance. No certain way to completely eradicate ill-conditioning in optimisation problems exists. Hence, newer methods for conditioning are always encouraged and looked up to.

2 CONTRIBUTIONS

With these motivations, from optimisation and conditioning point of view, it makes sense to look into the training of ELM based differential equation solvers (15; 46), called PIELM. The fundamental problem of possible ill-conditioning of such a solver is pointed out. Specifically, it is highlighted that asymptotic terms in an equation can be a reason for ill-conditioning. This issue can lead to significant stiffness in the training process, complicating the optimisation of neural models. Based on the understanding a novel neural architecture is proposed which gives well conditioned activation matrix. We propose a novel encoding/filtering procedure using shifted Gaussian functions and shifted ReLU to filter the activations from the previous layer in a particular fashion to generate a well-conditioned activation matrix.

3 PRELIMINARIES

Let us say for any differential equation defined by

$$\mathcal{N}[u](\mathbf{x}) = 0, \tag{1}$$

with constraints at $\mathbf{X}_u$ we have

$$u(\mathbf{X}_u) = \mathbf{u}_o. \tag{2}$$

The algorithms for solving the PDE defined by eq. (1) and eq. (2) using PINN and PIELM are mentioned in the subsequent subsections.

3.1 PHYSICS-INFORMED NEURAL NETWORKS

The algorithm 1 shows the pseudocode for PINNs as per the vanilla settings in (43). The method uses training data points $(\mathbf{X}_u, \mathbf{u})$ to fit the neural network with trainable parameters θ denoted by $u_\theta(\mathbf{x})$, and minimize the residual of $\mathcal{N}[u](\mathbf{x})$ at collocation points $\mathbf{X}_f$.

3.2 PHYSICS-INFORMED EXTREME LEARNING MACHINE (PI-ELM) ALGORITHM

The PI-ELM algorithm integrates the principles of extreme learning machines (ELMs) with PINNs to solve differential equations efficiently. The vanilla ELMs can be considered as simplified neural networks since they typically have only a single layer of weights to learn and all other layers are fixed with random constants(26). Amazingly, despite such simplifications, the theory of universal approximation remains valid(24; 25). The beauty of this method, which makes it special, is that the loss function could be represented as a set of equations that are nonlinear in terms of input variables and, however, linear in terms of trainable weights. Hence, the weights can be easily calculated by solving the system of linear equations, which is also the reason for the extreme speed of training single-layer networks.

The algorithm 2 shows the pseudocode for PI-ELM. Similar to the problem definition in PINNs, this method also uses training data points $(\mathbf{X}_u, \mathbf{u})$ to fit the neural network with trainable parameters β denoted by $u_\beta(\mathbf{x})$ and minimise the residual of $\mathcal{N}[u](\mathbf{x})$ at collocation points $\mathbf{X}_f$. Trained output weights β are obtained by solving the system of linear equations.

Theorem 3.1. *Let $\mathcal{L}$ be a loss function of a single-layered neural network as per the definition of an extreme learning machine, whose trainable weights are β, then $\mathcal{L}$ is convex in β.*

Proof. Check section A □

4 PROPOSED ARCHITECTURE

The architecture we propose is a layer of encoding after the layer of linear combination of weights to the inputs. Following the procedure of algorithm algorithm 2, we make a change to the equation eq. (27) following the same set of definitions and notations. The replacement equation is stated as:

$$u_\beta(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x} \odot E(\mathbf{x} - \mu) + \mathbf{b})\beta, \tag{3}$$

where E is the encoding function, $\odot$ is a Hadamard product, μ is a set $\mu_i = \frac{i}{L} | i \in [0, 1, .., L]$ and L is one less than the number of hidden nodes.

4.1 SHIFTED GAUSSIAN ENCODING

In this type, the encoding function E is defined as:

$$E(x) = e^{-\frac{x^2}{d}}, \tag{4}$$

where d is called the filter width.

5 EXPERIMENTS AND OBSERVATIONS

5.1 SCALES OF GRADIENTS AND 5 POSSIBLE SETBACKS

Let us assume an ordinary differential equation as eq. (5)

$$a_0 u + a_1 \frac{du}{dx} + a_2 \frac{d^2u}{dx^2} + .. + a_n \frac{d^n u}{dx^n} = b. \tag{5}$$

Like the concept we have been discussing, in order to solve the above equation by minimising the L2 norm of residuals by fitting neural networks $N(\mathbf{w}, x)$ at random collocation points in the domain, we define the loss function L as:

$$L(\mathbf{w}, x) = ||a_0 N(\mathbf{w}, x) + a_1 \frac{dN(\mathbf{w}, x)}{dx} + .. + a_n \frac{d^n N(\mathbf{w}, x)}{dx^n} - b||_2. \tag{6}$$

Since we opt for gradient descent to find the least square fit, we can update each trainable weight ($\mathbf{w}$) by the following update eq. (7).

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \nabla L(\mathbf{x}_k, x) \tag{7}$$

where ∇ is the gradient of the loss function and η is the learning rate. Where,

$$\nabla L(\mathbf{x}_k, x) = \sqrt{L}(a_0 \frac{\partial N}{\partial \mathbf{w}} + a_1 \frac{\partial^2 N}{\partial \mathbf{w} \partial x} + .. + a_n \frac{\partial^{n+1} N}{\partial \mathbf{w} \partial x^n}). \tag{8}$$

Looking at eq. (8), we can make a couple of serious observations. Firstly, the gradient descent depends on a set of derivatives of different orders of the network output and in nonlinear cases, it can depend on derivatives of different degrees as well. This should be leading to competing gradients due to each term suggesting a clear difficulty in finding a gradient step for reaching the global or Pareto optimum of the residual. Secondly, more is the number of different terms with different orders, the tougher it gets for the optimiser as it has to deal with more terms of different orders of magnitudes. It might require multiscale tactics. Thirdly, the approximating test function has to be flexible and capable enough to learn a function whose derivatives of different orders and are of different scales. The more varying the scales of each order of differentials are, which is a definite thing in perturbation or asymptotic problems, the more robust the test function might be needed. Fourthly, for the same conceptual reason, with reference to eq. (21), the loss function has two kinds of loss terms as described in eq. (19) for data fitting for the variable and eq. (20) for fitting to the differential equations. As such, this is a multi-objective optimisation problem, which brings in the chances of non-Pareto solutions. Of course when the scales of the data fitting error and the PDE residuals do not match. It demands opting for weighted residual instead of averaged residuals as in vanilla PINN. The same observation is described in a couple of works (46; 56), where the relative scaling of terms and variables helps us in getting better solutions. Additionally, from the perspective of machine precision and numerical computing, at perturbed or asymptotic regions in the domain, some of the derivatives can jump off the machine limit and hence crash the training or poorly approximate crucial gradients. Knowing all these factors, now when we checkout the work by Grossman et al. (19) on why neural differential equation solvers cannot beat finite element methods, we would not be surprised. It would not be wrong to say that using neural differential equation solvers is primarily solving optimisation problems.

5.2 SOLVING STEADY 1-DIMENSIONAL LINEAR STIFF ADVECTION DIFFUSION EQUATION

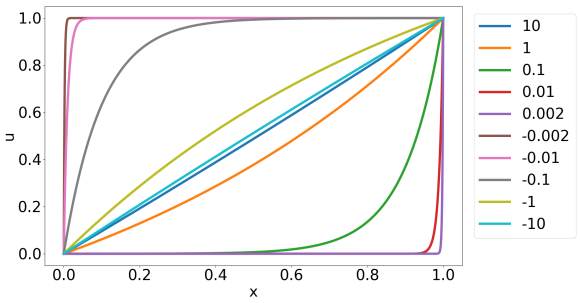

Figure 1: Exact solutions of steady 1D ADE for different ϵ.

Let us consider the case of solving one-dimensional steady advection diffusion equation (ADE) for u where $\epsilon \in \mathbb{R}$ is the diffusivity (a constant) and $x \in [0, 1]$ as in eq. (9)

$$\frac{\partial u}{\partial x} = \epsilon \frac{\partial^2 u}{\partial x^2}, \tag{9}$$

$$u(x = 0) = 0, \text{ and } u(x = 1) = 1. \tag{10}$$

Luckily, we know the exact solution of this problem. We can derive it by integrating the equation twice and substituting the boundary values. It is given by

$$u(x) = \frac{e^{x/\epsilon} - 1}{e^{1/\epsilon} - 1}. \tag{11}$$

The good part is that the solution is a function of a single exponential activation function, which means the function is expressible with very few nodes (2+). The fig. 1 shows the solution for different diffusivities. The solution is very sensitive to perturbations around zero diffusivity and gives a very different solution for the left and the right limits. The smaller is the diffusivity, the sharper is the gradient. Numerically, solving this problem for smaller diffusivity is tougher, as it requires much finer discretisation, at least around the region of sharper gradients. A realistic scale of diffusivity can be much smaller than what we see in the figure, typically found in sonic waves and fluid dynamics.

The fig. 6 shows that a typical neural differential equation solver (PIELM (46)) can solve for $\epsilon \in (\approx 0.06, \infty)$. The finite element approach of piecewise approximation with neural networks, works better, but is expensive and looses the charm of using a powerful approximator. The fig. 7 shows how such a solver, PIDELM (15), can solve for $\epsilon \in (\approx 0.02, \infty)$. The fig. 8 shows how such a solver with asymptotic scaling based transformation, PINDELM (46), can solve for $\epsilon \in (\approx 0.0025, \infty)$.

So, the motif of exploring how well we can solve using a single neural network for a side ODE remains unfulfilled. This motivates us to study the functional composition of the network and the loss function.

5.2.1 Statistics of Activation Matrix

We start looking at the steps in the algorithm of PIELM, refered in algorithm 2. The formulation of the loss function is convex as proved in theorem 3.1. So, ill-conditioning of inner function might be an obvious possibility. eq. (28) is the step we focus on. In the equation, we look at the statistics of the activation matrix H.

For a case, where we take an ELM of 1000 nodes to solve for $\epsilon = 1.0$ using 1000 collocation points, which generates a square activation matrix H, of 1002 rows, and hence the maximum possible rank is 1002. The case is solvable with appreciable accuracy. However, on generating the matrix, the rank it has is 14, the determinant is ≈ 0.0, and the condition number is $1.11e21$. The rank is too small and the condition number is too large. The fig. 10 shows how poor the condition is. The fig. 9 shows how dense the matrix is. It is clear evidence that the linear system we are solving is extremely ill-conditioned. Accordingly, the fig. 13 shows how, depending on ϵ, the magnitudes of residual (mean absolute error) as well as the weights increase sharply as it reduces down from ≈ 0.1. This suggests why a penalty loss term using L1 or L2 regularisation on weights, might help.

With PIELM on our architecture with shifted Gaussian encoding, we look upon the statistics of the activation matrix H. For a case, where we take an ELM of 1000 nodes to solve for $\epsilon = 1.0$ using 1000 collocation points, which generates a square activation matrix H, of 1002 rows, and hence the maximum possible rank is 1002. The case is solvable with appreciable accuracy. On generating the activation matrix with filter width, d, is 0.0001, the rank gets to 702, the determinant is ≈ 0.0, and the condition number is $7.68e20$. Though the condition number has not improved much, the rank has improved significantly. The fig. 12 shows better information in the activation matrix. The fig. 11 shows how the activation matrix is now sparse and diagonal.

Architecture	$Domain(\epsilon)$	Rank(H)	$\mathcal{O}(MAE)$ for $\epsilon = 0.01$
Typical Network	[≈1e-1, ∞)	14	1e-1
Our Network	**[≈1e-3, ∞)**	**702**	**1e-8**

Table 1: Comparison of results from using a typical single-layered neural network and our network with a layer of filtering. $\mathcal{O}(MAE)$ is the order-of-magnitude mean absolute error

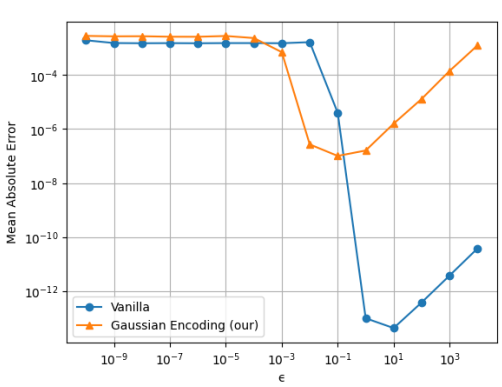

Figure 2: Comparison of mean absolute error for different ϵ.

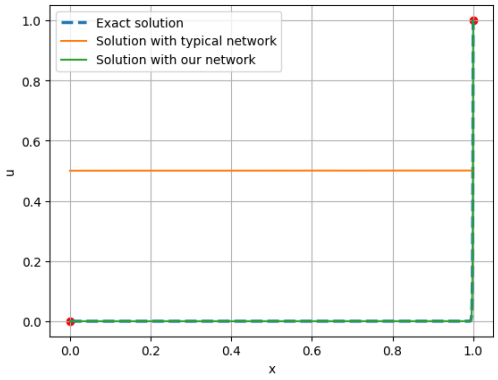

Figure 3: Solution to an extremely small diffusivity $\epsilon = 1e-3$ with red dots depicting the exact boundaries.

5.3 Learning Multi-fidelity Trends in Data

The goal of this study is to learn stiff and complex patterns in data. So, here the aim is designed as to train a neural network to approximate a complex target function that exhibits non-linear, oscillatory, and rapidly varying behaviour using the data points generated by the function. The function is defined by:

$$f(x) = \sin(10x) + 0.2\cos(20x + 50x^2) + e^{-100x^2}\sin(200x), \tag{12}$$

where:

- The term $\sin(10x)$ introduces a smooth oscillatory component with a moderate frequency.
- The term $0.2\cos(20x + 50x^2)$ adds a high-frequency oscillatory component modulated by a quadratic phase term.
- The term $e^{-100x^2}\sin(200x)$ contributes a highly localised oscillatory feature, with rapid damping due to the Gaussian factor e^{-100x^2}.

The generated data points can be seen in the fig. 15. The problem we propose here is not that easy. Learning this function poses several challenges that make this problem very exciting. The presence of oscillatory terms with varying high frequencies, such as $\sin(10x)$ and $\sin(200x)$, demands that the network accurately captures and represents these stiff variations. The Gaussian term e^{-100x^2} introduces sharp, localised features that require the network to effectively resolve fine details in the input space. The quadratic phase term $50x^2$ in $\cos(20x + 50x^2)$ creates a non-linear dependency of frequency that complicates the learning task.

The results to compare the learnability with different methods can be observed in fig. 15. A deep neural network (DNN) with four hidden layers with 100, 1000, 1000, and 100 nodes, respectively, with ReLU activation is used to demonstrate that even a deep network with more than a million parameters is not able to learn the high-frequency patterns. After that, as per reducing approximation strength, a single-layered network (SLN) with 1000 nodes and an ELM with 1000 nodes are used, and as expected, the mean squared error (MSE) increases as per the sequence of approximation strength. The table 2 shows the error values for different methods. It is clearly evident that our

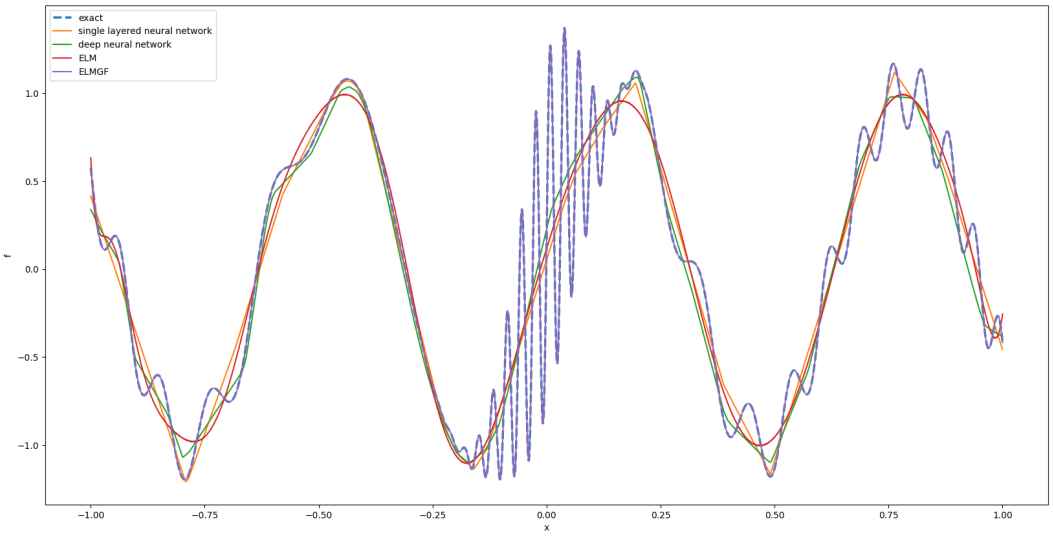

Figure 4: Trends learned with various methods.

ELM with Gaussian Encoding, which is conditioned for better expressivity of activations having the same 1000 hidden nodes, has a much smaller MSE. This suggests that it is not the approximation capability we need to look for, but how well the model can be trained is an important factor. Also our conditioned ELM takes a lesser runtime to find the pseudoinverse and hence learns the weights faster. It can also be observed in table 2. The Gaussian filter width (d) in our model controls the condition of the activation matrix. So, with a smaller d, we make the activation matrix of ELM get sparser and a better (higher) rank. So, if d is 1e-2 it shows a little better approximation than typical methods, while if d is 1e-3, it shows much more better results as it learns almost all visible trends.

An ELM with the first set of weights randomly assigned and the activation matrix we see has a rank of 16 out of a maximum of 1000. The fig. 16 shows the density distribution of the gram activation matrix of a typical ELM with our data points from the complex functional. The ELM takes in 10000 distinct data points to express the information with 1000 hidden nodes, that is, our activation matrix of 10000 rows and 1000 columns. Its rank is 16, and the network could have expressed 1000 different eigenvectors, while we do not recover the information 16 out of 16 might be a low-level expression of the available information in the data points. whereas when the activation matrix of the same size in the case of our method is compared, it can express the information from the same 10000 data points through 1000 hidden nodes as 356 different eigenvectors, that is, the rank we get. Our model is in fact able to learn the last set of weights easily. Our model could perform better as it could not only learn dominant low frequency eigenmodes but also a sufficient number of imapctful high-frequency eigenmodes. Our gram matrix can be seen in fig. 17, it is much more structured. So, a great learning is to not just take a big network but also understand if the activation from the previous layer of a network is expressing the information to the next layer sufficiently. Ideally, we would wish to create a layer that sufficiently expresses itself. In case the input data is expressible with 1000 vectors and our model layer has 100 nodes for a lower-order representation so, the rank of an ideal activation matrix should be 100. If not we should try to keep the rank as high as possible. Having 100 nodes yet the rank is let's say 10, would mean only 10-20 nodes would have been sufficient to perform what it is performing at the moment.

5.4 Fitting VanGogh's Eye

A single-channel image of Van Gogh's eye is vectorised to pose a complex multiscale vector to be fitted. The table 3 shows the comparison of performance and fig. 5 shows the images obtained by different methods.

Method	# Trainable Parameters	MSE	Training Time (s)
DNN	1.2M+	4.2e-3	179.968
SLN	3001	4.5e-3	31.448
ELM	1000	4.5e-3	6.686
Our ELM (d=1e-2)	**1000**	**1.1e-3**	**5.548**
Our ELM (d=1e-3)	**1000**	**4.9e-09**	**4.694**

Table 2: Comparison of results from using a typical single-layered neural network and our network with a layer of filtering.

Method	# Trainable Parameters	MSE	Training Time (s)
DNN	1.2M+	1.39e-2	118.67
SLN	3001	1.41e-2	5.35
ELM	1000	1.36e-2	11.76
Our ELM (d=1e-2): ELMGF	**1000**	**1.32e-2**	**18.62**
Our ELM (d=1e-4): ELMGF2	**1000**	**3.29e-3**	**7.22**
Our ELM (d=1e-5): ELMGF3	**5000**	**1.23e-3**	**220.87**

Table 3: Comparison of results for fitting the image of Van Gogh's Eye.

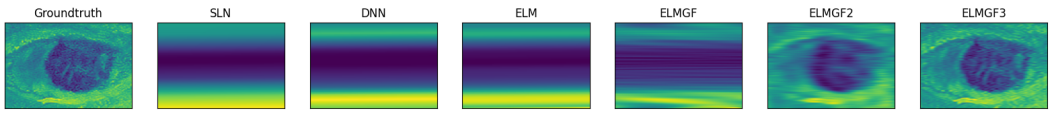

Figure 5: Learned images with various methods.

6 Results and Discussions

Consequently, fig. 2 shows how, depending on ϵ, the magnitudes of the residual (mean absolute error) are different for different architectures. Shifted Gaussian encoding improves the result from vanilla architecture over a comparatively wider domain. The results with shifted Gaussian encoding are great achievements for computational scientists working with hyperbolic PDEs, as no neural network could ever approximate such a sharp gradient in advection-diffusion problems. The other best solutions for singularly perturbed problems either involve high fidelity higher order discretised finite approximation that are expensive and might require manual modelling (51; 38), otherwise solve for asymptotic approximations (2; 30) which poorly match the whole domain and fail at different scales (21). Similarly, the method is brilliant at fitting to complex one-dimensional equations and vectors.

7 Conclusion

A conditioning-focused architecture for scientific neural solvers by combining the convexity of Extreme Learning Machines with *Shifted Gaussian Encoding* is presented. This approach substantially improves the rank and sparsity of activation matrices, enabling PIELMs to solve stiff DEs and capture multi-scale patterns far beyond the reach of conventional shallow or deep baselines.

The experiments demonstrate broader solvability, including advection–diffusion equations down to $\epsilon \approx 10^{-3}$ without domain decomposition or asymptotic preprocessing; higher accuracy at lower cost, achieving orders-of-magnitude improvement in error for oscillatory functions while reducing pseudoinverse computation time; and strong generality, with robust performance across synthetic functions, DEs, and high-dimensional vectors such as image fitting.

These results suggest that, for many scientific learning problems, improving *conditioning* can be more impactful than increasing network depth or width. Future work will focus on formalising the link between encoding width and matrix spectrum, and exploring adaptive encodings for dynamic conditioning across training.

ACKNOWLEDGEMENTS

Special thanks to Prof. Eldad Haber, Prof. Michael Friedlander, Prof. Balaji Srinivasan, Dr. Vikas Dwivedi, and Benjamin Liu.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] A. Arzani, K. W. Cassel, and R. M. D'Souza. Theory-guided physics-informed neural networks for boundary layer problems with singular perturbation. *Journal of Computational Physics*, 473:111768, 2023.

[3] A. Arzani and S. T. M. Dawson. Data-driven cardiovascular flow modelling: examples and opportunities. *Journal of The Royal Society Interface*, 18(175):20200802, 2021.

[4] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] F. Bach. Breaking the curse of dimensionality with convex neural networks. *Journal of Machine Learning Research*, 18(19):1–53, 2017.

[6] F. R. Bach. Consistency of the group lasso and multiple kernel learning. *Journal of Machine Learning Research*, 9(6), 2008.

[7] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153):1–43, 2018.

[8] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, and U. Montreal. Greedy layer-wise training of deep networks. volume 19, 01 2007.

[9] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: A review. *Acta Mechanica Sinica*, 37(12):1727–1738, 2021.

[10] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, et al. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45, 2024.

[11] A. Chattopadhyay, P. Hassanzadeh, and D. Subramanian. Data-driven predictions of a multiscale lorenz 96 chaotic system using machine-learning methods: reservoir computing, artificial neural network, and long short-term memory network. *Nonlinear Processes in Geophysics*, 27(3):373–389, 2020.

[12] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022.

[13] G. V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

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

[15] V. Dwivedi and B. Srinivasan. Physics informed extreme learning machine (pielm)–a rapid method for the numerical solution of partial differential equations. *Neurocomputing*, 391:96–118, 2020.

[16] T. Ergen and M. Pilanci. Revealing the structure of deep neural networks via convex duality. In *International Conference on Machine Learning*, pages 3004–3014. PMLR, 2021.

[17] M. Fatica. Cuda toolkit and libraries. In *2008 IEEE Hot Chips 20 Symposium (HCS)*, pages 1–22, 2008.

[18] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[19] T. G. Grossmann, U. J. Komorowska, J. Latz, and C.-B. Schönlieb. Can physics-informed neural networks beat the finite element method? *IMA Journal of Applied Mathematics*, page hxae011, 05 2024.

[20] E. Hazan, K. Levy, and S. Shalev-Shwartz. Beyond convexity: Stochastic quasi-convex optimization. *Advances in neural information processing systems*, 28, 2015.

[21] E. J. Hinch. *Perturbation Methods*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 1991.

[22] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

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

[24] G. Huang, L. Chen, and C. K. Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17:879–892, 2006.

[25] G.-B. Huang and L. Chen. Convex incremental extreme learning machine. *Neurocomputing*, 70(16):3056–3062, 2007.

[26] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.

[27] S. Ioffe and C. Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[28] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[29] A. D. Jagtap and G. E. Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5), 2020.

[30] M. K. Kadalbajoo and V. Gupta. A brief survey on numerical methods for solving singularly perturbed problems. *Applied Mathematics and Computation*, 217(8):3641–3716, 2010.

[31] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[32] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[33] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.

[34] C. Li, C. Chen, D. Carlson, and L. Carin. Preconditioned stochastic gradient langevin dynamics for deep neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[35] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019, 2022.

[36] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024.

[37] L. D. McClenny and U. M. Braga-Neto. Self-adaptive physics-informed neural networks. *Journal of Computational Physics*, 474:111722, 2023.

[38] A. Mohebbi and M. Dehghan. High-order compact solution of the one-dimensional heat and advection–diffusion equations. *Applied Mathematical Modelling*, 34(10):3071–3084, 2010.

[39] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan. Speech recognition using deep neural networks: A systematic review. *IEEE Access*, 7:19143–19165, 2019.

[40] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[41] J. Pathak, S. Subramanian, P. Harrington, S. Raja, A. Chattopadhyay, M. Mardani, T. Kurth, D. Hall, Z. Li, K. Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

[42] W. Peebles and S. Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

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

[44] M. Raissi, A. Yazdani, and G. E. Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[45] C. R. Rao and S. K. Mitra. Generalized inverse of a matrix and its applications. In *Proceedings of the sixth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 601–620. University of California Press Oakland, CA, USA, 1972.

[46] S. Rout. Numerical approximation in cfd problems using physics informed machine learning. *Indian Institute of Technology Madras*, 2019.

[47] S. Saarinen, R. Bramley, and G. Cybenko. Ill-conditioning in neural network training problems. *SIAM Journal on Scientific Computing*, 14(3):693–714, 1993.

[48] F. Sahli Costabal, Y. Yang, P. Perdikaris, D. E. Hurtado, and E. Kuhl. Physics-informed neural networks for cardiac activation mapping. *Frontiers in Physics*, 8, 2020.

[49] Y. Shin, J. Darbon, and G. E. Karniadakis. On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes. *Communications in Computational Physics*, 2020.

[50] K. Shukla, A. D. Jagtap, and G. E. Karniadakis. Parallel physics-informed neural networks via domain decomposition. *Journal of Computational Physics*, 447:110683, 2021.

[51] M. Stynes. Steady-state convection-diffusion problems. *Acta Numerica*, 14:445–508, 2005.

[52] A. J. Thirunavukarasu, D. S. J. Ting, K. Elangovan, L. Gutierrez, T. F. Tan, and D. S. W. Ting. Large language models in medicine. *Nature medicine*, 29(8):1930–1940, 2023.

[53] J. Thiyagalingam, M. Shankar, G. Fox, and T. Hey. Scientific machine learning benchmarks. *Nature Reviews Physics*, 4(6):413–420, 2022.

[54] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[55] P. van der Smagt and G. Hirzinger. *Solving the Ill-Conditioning in Neural Network Learning*, pages 193–206. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[56] S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[57] Y. Yang and P. Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 394:136–152, 2019.

[58] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, Oct. 2019.

A THEOREMS AND PROOFS

Theorem A.1. *Let $\mathcal{L}$ be a loss function of a single-layered neural network as per the definition of an extreme learning machine whose trainable weights are β, then $\mathcal{L}$ is convex in β.*

Proof. Given, $\mathcal{L}$ is the loss function of an extreme learning machine where let the single-layered neural network be $(\mathcal{N})$ of L hidden nodes. Let $(\mathcal{N})$ be a function of input $x \in \mathbb{R}^{C_1 \times n}$, fixed weight $W \in \mathbb{R}^{L \times C_1}$, fixed bias $b \in \mathbb{R}^L$ and trainable weight $\beta \in \mathbb{R}^{C_2 \times n}$ to predict as defined by

$$\mathcal{N}(X, \beta, W, b) = \beta\phi(Wx + b), \tag{13}$$

where ϕ is a nonlinear activation function, $L \in \mathbb{N}$ is the number of nodes in the hidden layer, $C_1 \in \mathbb{N}$ is the number of input nodes, $C_2 \in \mathbb{N}$ is the number of output nodes and n is the number of samples.

If the output is $Y \in \mathbb{R}^{C_2 \times n}$,

$$\mathcal{L}(\beta) = ||Y - \beta\phi(Wx + b)||_2. \tag{14}$$

So, the gradient with respect to $beta$ is given by

$$\nabla\mathcal{L}(\beta) = 2(Y - \beta\phi(Wx + b))(-\phi(Wx + b))^T. \tag{15}$$

So, the Hessian with respect to $beta$ is given by

$$\nabla^2\mathcal{L}(\beta) = 2(\phi(Wx + b))(\phi(Wx + b))^T. \tag{16}$$

If $A = \phi(Wx + b)$ then the Hessian $\mathcal{H}$ can be written as,

$$\mathcal{H} = 2AA^T. \tag{17}$$

Since AA^T is positive semidefinite, $AA^T \geq 0$. So,

$$\mathcal{H} = \nabla^2\mathcal{L}(\beta) \geq 0. \tag{18}$$

Hence, $\mathcal{L}$ is convex in terms of the weight β.

□

B PSEUDOCODES

The algorithm 1 shows the algorithm for training physics-informed neural networks. The algorithm 2 shows the algorithm for training physics-informed extreme learning machine.

C PINNS 1D STEADY ADVECTION-DIFFUSION EQUATION

C.1 COMPARISON OF EXISTING METHODS

See figs. 6 to 8.

C.2 COMPARISON OF ACTIVATION MATRICES

See figs. 9 to 12.

C.3 COMPARISON OF ORDERS OF WEIGHTS AND RESIDUALS

See figs. 13 and 14.

D COMPLEX MULTISCALE FUNCTION

See fig. 15 for the plot of the exact multiscale function.

Algorithm 1 PINN

1: **Initialization** Initialize neural network parameters θ (weights and biases)
2: **Define the Neural Network** Construct the neural network $u_\theta(\mathbf{x})$ with parameters θ
3: **Formulate the Loss Functions** Define data fitting loss:

$$\mathcal{L}_{\text{data}} = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u_\theta(\mathbf{x}_u^i) - u^i \right|^2. \tag{19}$$

Define physics-informed loss using the governing differential equation $\mathcal{N}[u] = 0$:

$$\mathcal{L}_{\text{phys}} = \frac{1}{N_f} \sum_{j=1}^{N_f} \left| \mathcal{N}[u_\theta](\mathbf{x}_f^j) \right|^2. \tag{20}$$

Combine the losses:

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{phys}}. \tag{21}$$

4: **Training** Minimize the total loss $\mathcal{L}$ with respect to the network parameters θ:

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta). \tag{22}$$

5: **Inference** Use the trained network u_{θ^*} to make predictions for new inputs $\mathbf{X}_{\text{new}}$:

$$\hat{\mathbf{u}} = u_{\theta^*}(\mathbf{X}_{\text{new}}). \tag{23}$$

Algorithm 2 PI-ELM

1: **Initialization** Randomly initialize input weights $\mathbf{W} \in \mathbb{R}^{L \times n}$ and biases $\mathbf{b} \in \mathbb{R}^L$. Define activation function $\phi(\cdot)$
2: **Formulate Composite Loss Function** Define data fitting loss:

$$\mathcal{L}_{\text{data}} = \sum_{i=1}^{N_u} \left| u(\mathbf{x}_u^i) - u^i \right|^2. \tag{24}$$

Define physics-informed loss:

$$\mathcal{L}_{\text{phys}} = \sum_{i=1}^{N_f} \left| \mathcal{N}[u(\mathbf{x}_f^i)] \right|^2. \tag{25}$$

Combine the losses:

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{phys}}. \tag{26}$$

3: **Generate the System of Equations** The equation for the neural network $u_\beta(\mathbf{x})$ is

$$u_\beta(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})\beta. \tag{27}$$

The effective system of linear equations in $\mathcal{L}$ could be rewritten in the form:

$$\mathbf{H}\beta = \mathbf{T}. \tag{28}$$

4: **Training** Solve for the output weights using the Moore-Penrose pseudoinverse of $\mathbf{H}$ denoted by $\mathbf{H}^+$ which is essentially the least square solution:

$$\beta = \mathbf{H}^+\mathbf{T}. \tag{29}$$

5: **Inference** Use the trained PI-ELM model to predict outputs for new inputs $\mathbf{X}_{\text{new}}$:

$$\hat{\mathbf{Y}} = \phi(\mathbf{W}\mathbf{X}_{\text{new}} + \mathbf{b})\beta. \tag{30}$$

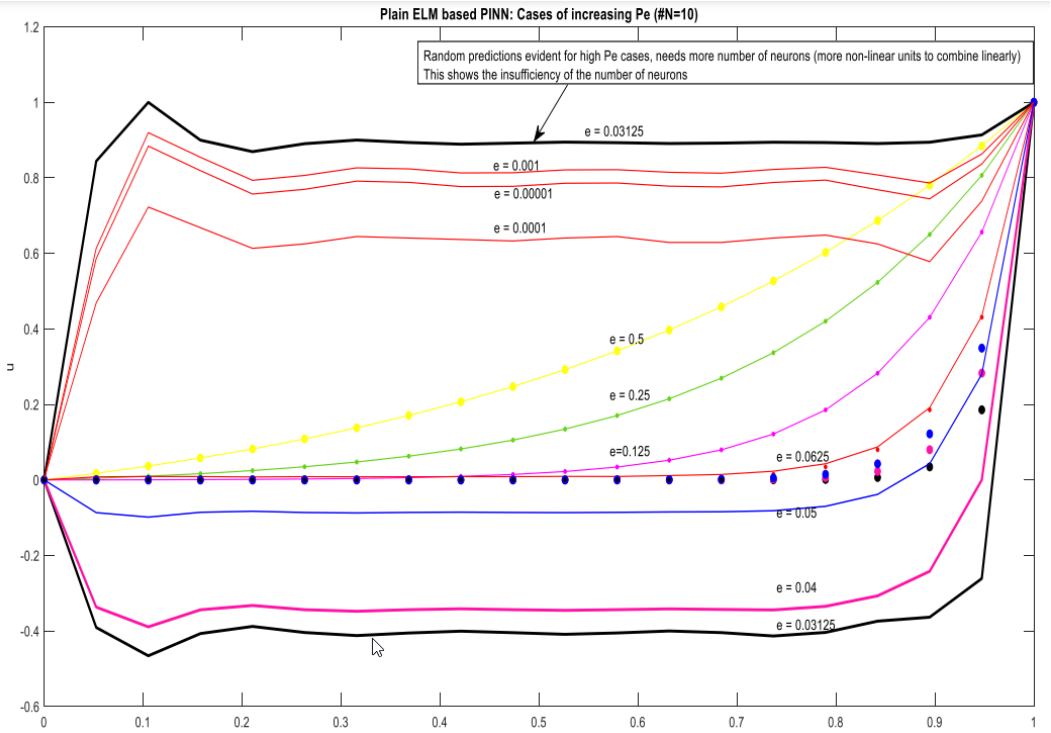

Figure 6: Solutions of steady 1D ADE for different ϵ values with PIELM in continuous lines and exact solutions in dots.

D.1 Comparison of activation matrices

See figs. 16 and 17 for the comparison of Gram activation matrices formed in different approaches.

E Multiscale Image

E.1 Comparison of activation Gram matrices

See fig. 21 for comparison of the Gram matrix formed in various methods.

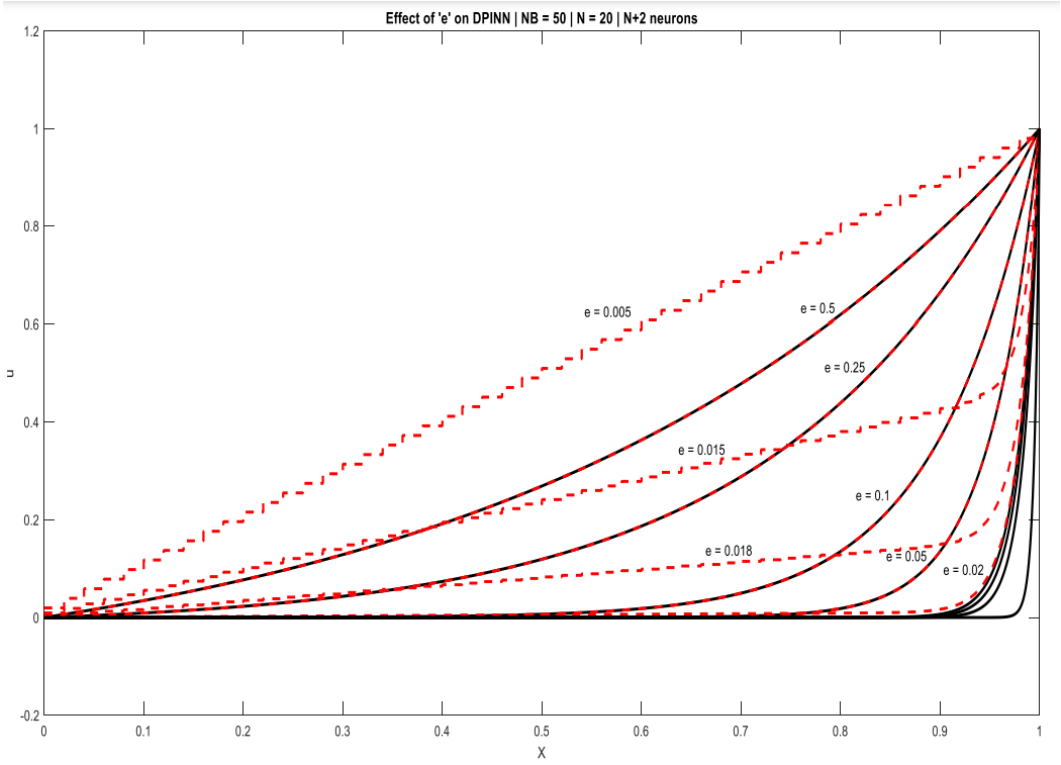

Figure 7: Solutions of steady 1D ADE for different ϵ values with PIDELM in red dashed lines and exact solutions in black lines.

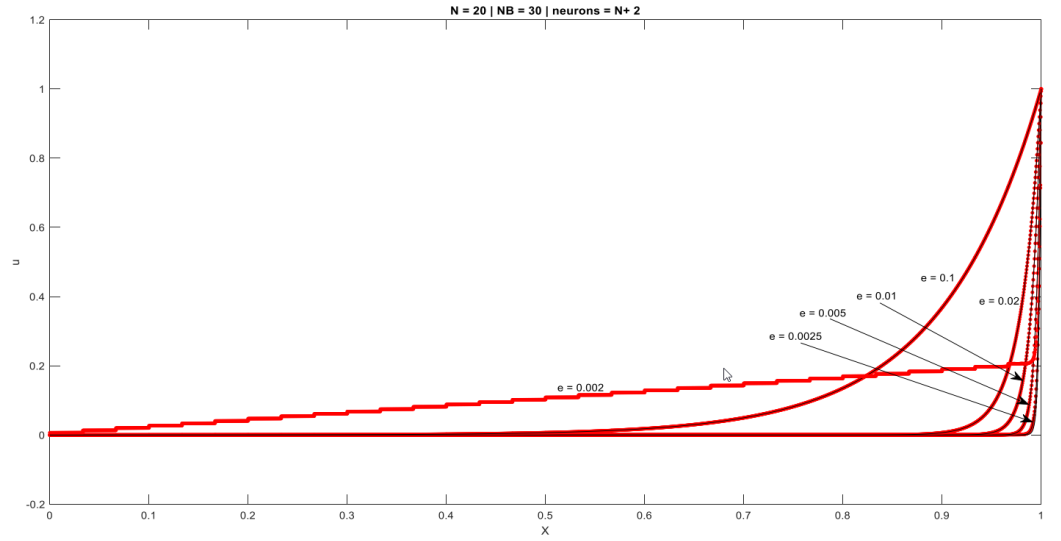

Figure 8: Solutions of steady 1D ADE for different ϵ values with PINDELM in red dotted lines and exact solutions in black lines.

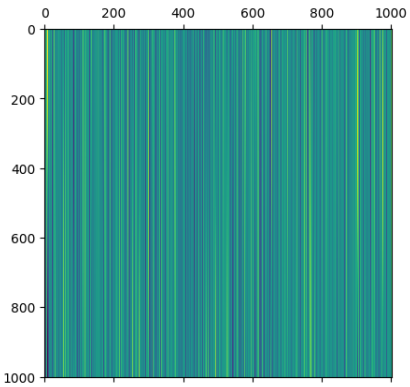

Figure 9: Visualisation of H of PIELM for steady 1D ADE with $\epsilon = 1.0$.

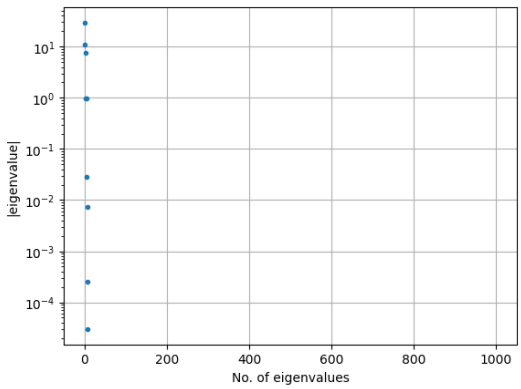

Figure 10: Eigenvalues of H of PIELM for steady 1D ADE with $\epsilon = 1.0$.

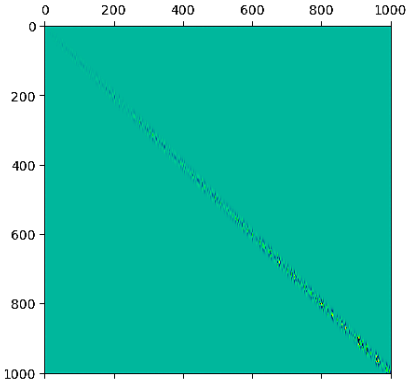

Figure 11: Visualisation of H of PIELM with our architecture for steady 1D ADE with $\epsilon = 1.0$ when shifted Gaussian encoding is used.

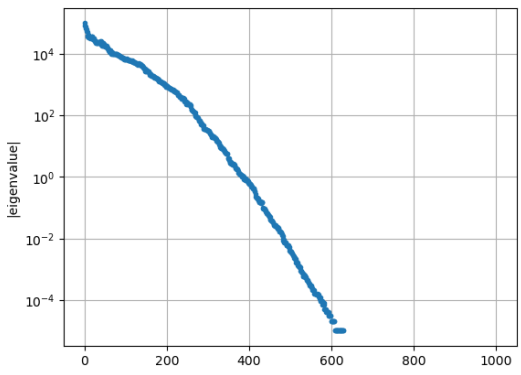

Figure 12: Eigenvalues of H of PIELM with our architecture for steady 1D ADE with $\epsilon = 1.0$ when shifted Gaussian encoding is used.

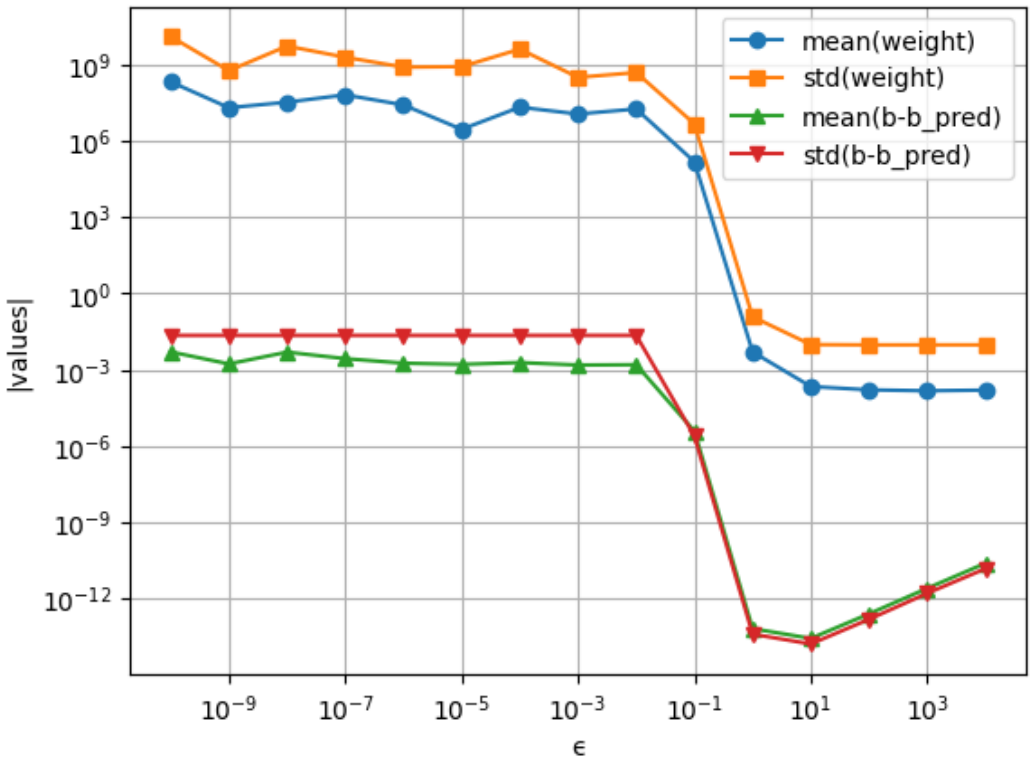

Figure 13: Mean and standard deviation of weights and residual (b - b_pred) of PIELM solution of steady 1D ADE for various ϵ values.

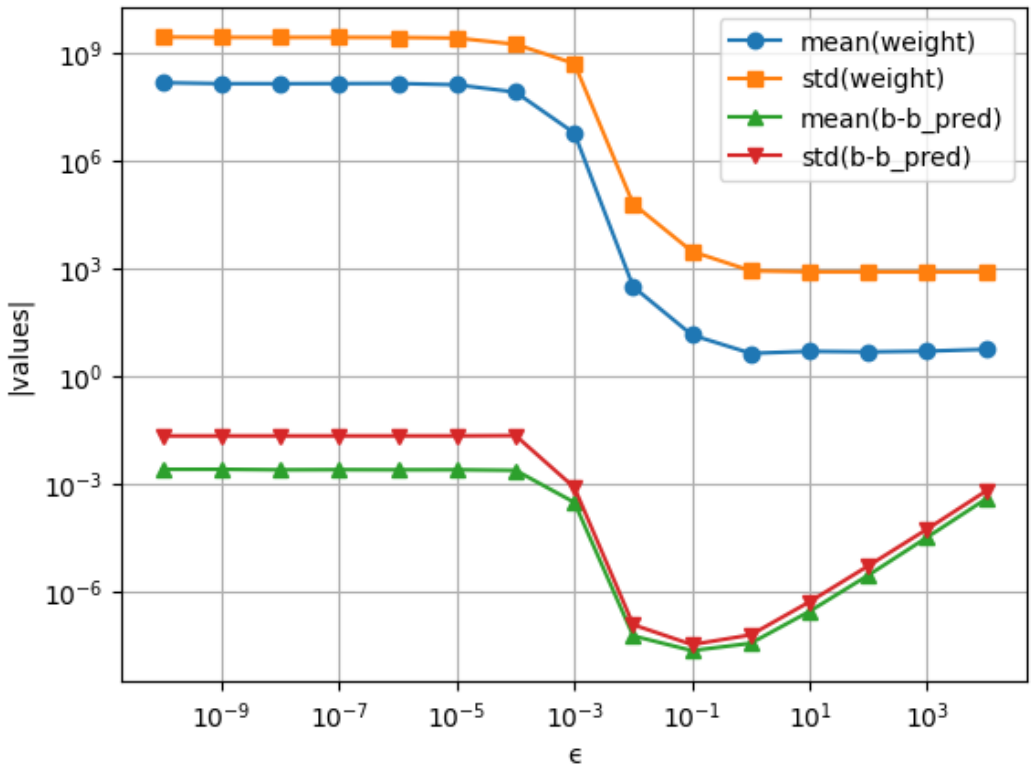

Figure 14: Mean and standard deviation of weights and residual (b - b_pred) of PIELM solution of steady 1D ADE for various ϵ values when shifted Gaussian encoding is used.

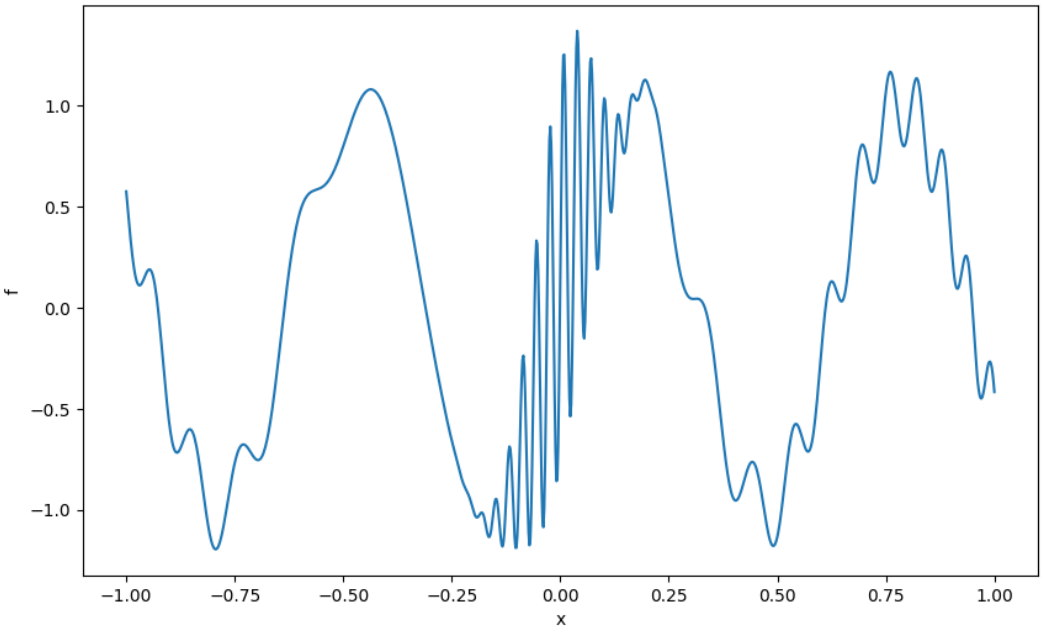

Figure 15: Points generated from the target function.

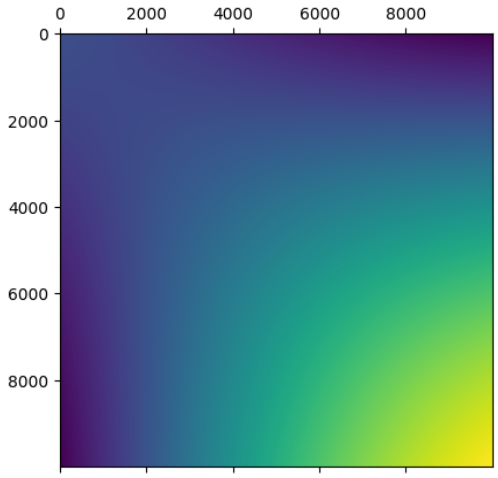

Figure 16: Visualisation of Activation matrix for the ELM.

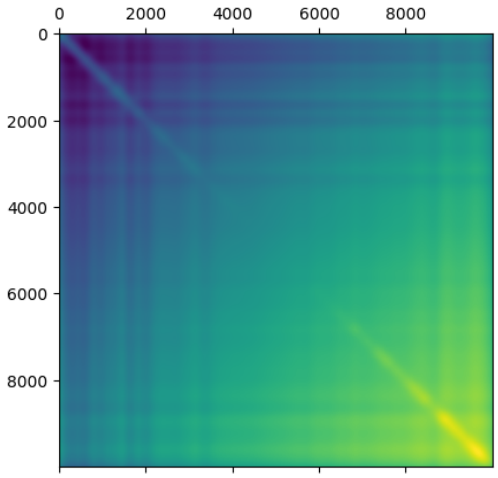

Figure 17: Visualisation of Activation matrix for our ELM after Gaussian Encoding.

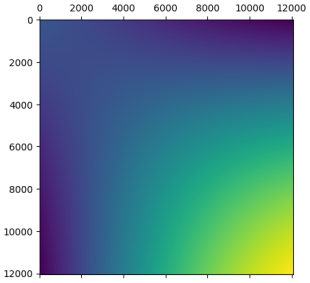

Figure 18: Visualisation of Activation matrix for the ELM.

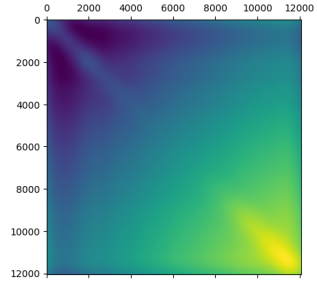

Figure 19: Visualisation of Activation matrix for our ELM (ELMGF).

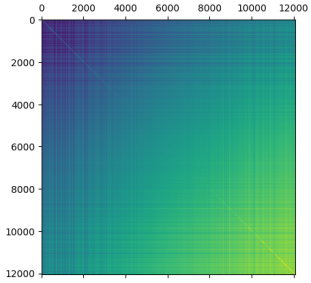

Figure 20: Visualisation of Activation matrix for our ELM (ELMGF3).

Fitting Van Gogh's Eye.