8-2020

Numerical Approximations of Phase Field Equations with Physics Informed Neural Networks

Colby Wight
Utah State University

NUMERICAL APPROXIMATIONS OF PHASE FIELD EQUATIONS WITH PHYSICS INFORMED NEURAL NETWORKS

by

Colby L. Wight

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Mathematics

Approved:

Jia Zhao, Ph.D. Major Professor	Joseph V. Koebbe, Ph.D. Committee Member

Heng Da Cheng, Ph.D.
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2020

ABSTRACT

Numerical Approximations of Phase Field Equations with Physics Informed Neural Networks

by

Colby L. Wight, Master of Science

Utah State University, 2020

Major Professor: Jia Zhao, Ph.D.
Department: Mathematics and Statistics

Designing numerical algorithms for solving partial differential equations (PDEs) is one of the major research branches in applied and computational mathematics. Recently there has been some seminal work on solving PDEs using the deep neural networks. In particular, the Physics Informed Neural Network (PINN) has been shown to be effective in solving some classical partial differential equations. However, we find that this method is not sufficient in solving all types of equations and falls short in solving phase-field equations. In this thesis, we propose various techniques that add to the power of these networks. Mainly, we propose to embrace the adaptive idea in both space and time and introduce various sampling strategies to improve the efficiency and accuracy of the PINN. The improved PINN can solve a broader set of PDEs, and in particular, the phase-field equations. The improved PINN sheds light on numerical approximations of other PDEs in general.

(34 pages)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

RESEARCH BACKGROUND

This major goal of this thesis is to investigate strategies to improve the capabilities of deep neural networks on solving differential equations. In this section, We will give a brief introduction of several concepts and research background.

1.1 Differential equations

A differential equation (DE) is an equation that relates the derivatives of a (scalar) function depending on one or more variables. A differential equation is called partial differential equation (PDE) if it depends on more than one variable [12]. A "solution" to a PDE is a function of the independent variables that satisfy the equation at every point in the domain of definition. An example is the heat equation, $u_t = u_{xx}$, on the domain $D \subset \mathbb{R}^2$. The function $u(t, x) = t + \frac{1}{2}x^2$ is a solution in this case. This type of solution is called an analytic solution. In this case there can be many solutions to this equation. Initial conditions and boundary conditions can be defined to restrict the possible solutions of a PDE.

1.2 Artificial neural networks

The artificial neural network is named after the fundamental unit of computation inside the mammalian brain [9]. Many neurons inside the brain work together to carry out complex tasks. Similarly, an artificial neural network is composed of multiple connected neurons that work to solve complex tasks.

A single neuron in a neural network can take input from multiple neurons (or nodes). Each input has a parameter called a weight associated with it. There is also typically a bias term that doesn't have an input associated with it. The neuron receives the sum of these inputs multiplied by their weights and bias multiplied by 1. This weighted sum then

goes through an activation function that gives the final output for this neuron. In the brain, a neuron usually doesn't fire unless the total of its input reaches a certain threshold. The output is either on or off. In the field of deep learning, continuous functions are more commonly used [11]. The *sigmoid* function can be used as a smoother version of the step function. There are benefits in using differentiable functions like this to help in "learning" good weights. Other useful activation functions used in deep learning include *relu*, tanh, and leaky *relu* [17].

A typical feedforward, fully connected artificial neural network has input going to and from multiple neurons. The input to the network makes up the input layer. The value of each input get propagated to the first hidden layer. The output of each neuron in this hidden layer then becomes the input to other neurons in the next hidden layer. See the Figure 1.1 for a visual representation of a simple architecture.

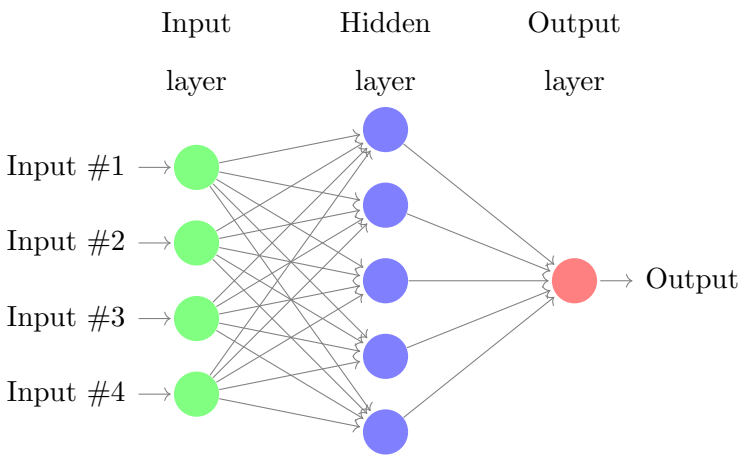

Fig. 1.1: A diagram of a neural network with an input layer (with 4 inputs), a hidden layer, and an output layer (with 1 output). Each input gets sent to each neuron in the hidden layer. The arrows between the neurons all have a weight associated with them. The bias for each hidden neuron and output neuron are not shown.

Essentially neural networks are non-linear mappings with many parameters. Due to the large number of parameters they are referred to as a "black-box" [4]. The parameters can be 'learned' by optimizing the loss function. For a supervised learning, the loss function compares the networks output with the known output, and tries to tune the parameters

to minimize this loss. Typically a gradient based optimization scheme is applied. The backpropagation algorithm is an efficient way of find the gradient of this highly dimensional loss function [6]. Stochastic gradient descent is a popular method which uses smaller subsets of the training data called batches for each step to achieve better results [2].

Training a network refers to using one of these optimization methods to change or tune the weights in the network so that the network performs better at the task. The loss function is used as the objective function.

1.3 Solving differential equations with deep neural networks

There are various studies that deal with problems pertaining to data-driven modeling. Some of these methods use data to find the differential equations themselves in order to model real world phenomenon [3,15]. In the past few years, there has been intensive research on understanding how deep neural networks can be adopted to solve and discover differential equations, though some early seminal work can be traced back to [5,13].

In particular, Raissi et al. propose the Physics Informed Neural Networks (PINNs) to aid in both the solution of differential equations as well as their discovery [13,14]. The PINNs were shown to solve Burgers' equation and the Schrodinger equation with certain initial conditions accurately. Since this initial discovery, work has been done to further develop the PINNs. Some focus has been on convergence of the PINNs [7,16]. Misyris et al. showed the ability PINNs in solving power systems faster than conventional methods [10].

Though the PINN has been widely appreciated in the community, we found a direct application of the PINN on solving the phase field equations (such as the Allen-Cahn equations and Cahn-Hilliard equations) can't find the solutions accurately. We also observe that the solutions of phase field equations in some parts of the domain might be harder to learn than others, as the solutions usually have a sharp transient layer. These difficult areas could even change over the course of learning the solution. The original method with fixed training data points, though randomly chosen across the domain, won't work any more. This motivates us to investigate strategies to improve the accuracy and efficiency of PINNs on solving phase field equations.

CHAPTER 2

NUMERICAL METHODS

In this section, we will give a brief review of the physics informed neural network (PINN). Then, we propose several strategies to improve the accuracy and approximating capability of the PINN.

Along with simple strategies for improving performance in general, our major contribution is adopting the concepts of adaptivity in numerical PDEs for training the deep neural network. Mainly, instead of randomly picking data points at the beginning of training and fixing them, we find it useful to resample the data points throughout the training process. This idea is akin to those of more classical adaptive methods for solving differential equations such as adaptive mesh refinement [1]. We explore the benefits of both adaptive sampling in space and time.

2.1 Physics informed neural networks

In [13, 14], the authors dealt with differential equations of the form:

$$u_t + \mathcal{N}[u; \lambda] = 0,$$

where the solution to the equation is some function $u(x, t)$. u_t denotes the partial derivative of u with respect to t and $\mathcal{N}[u; \lambda]$ is the nonlinear part of the differential equation parameterized by λ. A neural network, acting as a surrogate of the solution $u(t, x)$, is used to approximate the solution across the problem domain. This network is a multilayer perceptron that has two inputs and one output. The inputs are time t and spatial location in one dimension x. The output is the approximate value for the solution u at that x and t point in the domain. This is the u-network and can be trained in the supervised learning sense by passing in points across the domain of the problem where the solution is know along with the initial condition.

In addition, a f-network is also defined:

$$f := u_t + \mathcal{N}[u; \lambda].$$

Using automatic differentiation of the network u the appropriate derivatives can be taken to form $f(t, x)$, which shares the same weights and biases as the u-network. Any x and t point chosen from the domain of the equation that is passed into the f-network will ideally have an output of 0. This means we don't need to know the value of the solution to pass points into this network for training. The points passed into this function when training the neural network are called collocation points. Finally, the loss function used to train the shared weights of the u and f-network is made using the mean squared error of their outputs.

To better elucidate the idea, we use the Burgers equations as an example, which is given as below.

$$\begin{aligned} &u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1], \\ &u(0, x) = -sin(\pi x), \\ &u(t, -1) = u(t, 1) = 0. \end{aligned} \tag{2.1}$$

For instance, we can propose the architecture for the u-network has 2 input neurons, 20 neurons in each of 8 hidden layers, and 1 neuron in the final output layer. The hyperbolic tangent function can be used as the activation function in all of the layers. The f-network is then created by taking the appropriate partial derivatives of the u-network and adding them together with the parameters to mirror the left hand side of the equation.

The loss function for this problem is the mean squared error of the u-network plus the mean squared error of the f-network. The combined mean squared error looks like this:

$$MSE = MSE_u + MSE_f, \tag{2.2}$$

where MSE_u, and MSE_f are defined as

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2, \quad MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2. \tag{2.3}$$

Here t_u^i and x_u^i are the initial condition points that are the inputs to the u-network and u^i is the actual value of u at those points obtained by evaluating the initial condition. t_f^i and x_f^i are the collocation points passed into the f-network. We see that an output of zero for the f-network results in an error of zero. N_u and N_f are the number of initial training data and number of collocation points respectively.

For this trial 100 randomly selected initial points are used and 10,000 collocation points are selected using a Latin hypercube sampling (lhs) strategy across the domain of the problem. The network is trained using the L-BFGS-B optimizer with default settings. In later problem Adam optimizer will be used as well.

A prediction method gives the output of the trained network for both the u-network and the f-network for any input. Recall the u-network gives the approximated value of the solution to the differential equation, and the f-network gives the approximated value of the left hand side of the equation at the given point which should ideally be zero. To test the accuracy of the solution, the actual solution is obtained using MATLAB where the value of the solution is obtained at points on a fine grid across the domain. The accuracy of the trained model is assessed by taking the relative l_2-norm of the difference between the actual value at those points and the u-network output at those points.

2.2 Some strategies to improve the physics informed neural networks

In the rest of this section, we introduce some strategies to improve the approximation power of the PINN.

2.2.1 Adding weights in the loss function

One simple technique to improve the accuracy of learnt solutions from the PINN approach is to add weights in the loss function. For example, the Allen-Cahn equation as

with other reactive diffusion equations can only be solved in the forward time direction. In other words, if we do not know the solution of an equation at time t_1 well, there is little hope of learning the solution at a latter time t_2 (with $t_2 > t_1$). In order to put an emphasis on the importance of first solving the solution near $t = 0$ we put more weight on the part of the loss function that enforces the initial condition by multiplying it by a big positive constant, saying 100 for our trial. We use the same setting as in the benchmark trial for solving Allen-Cahn and changed the loss function as follows:

$$MSE = 100 * MSE_u + MSE_f + MSE_b. \tag{2.4}$$

2.2.2 Adaptive sampling in space

The goal of this method is to choose collocation points across the domain, where they are needed most. Instead of only sampling points evenly across the domain we periodically stop training and re-evaluate where points are needed most. We notice there was correlation between the area that had a larger error in the u-network (the solution) and the points that had large error in the f-network. If we intermittently stop training and evaluated the f-network using test points, we can see where the error is highest and choose more collocation points from that area and then resume training of the network with these updated collocation points. In order to do this, we first train the network using the randomly selected points across the domain. We then choose a different set of sample test points across the domain using the same Latin hypercube sampling technique and pick a portion of the points that give the highest error in predicting f. We add this set of points to the set of random collocation points and train the network again. We do this as to not lose the accuracy of the solution across the entire domain and to focus more points to learn the trickier parts better. This process can be iterated as many times as necessary adding a different set of sampled collocation points to the original set and training again. See Figure 2.1 for a diagram of this process.

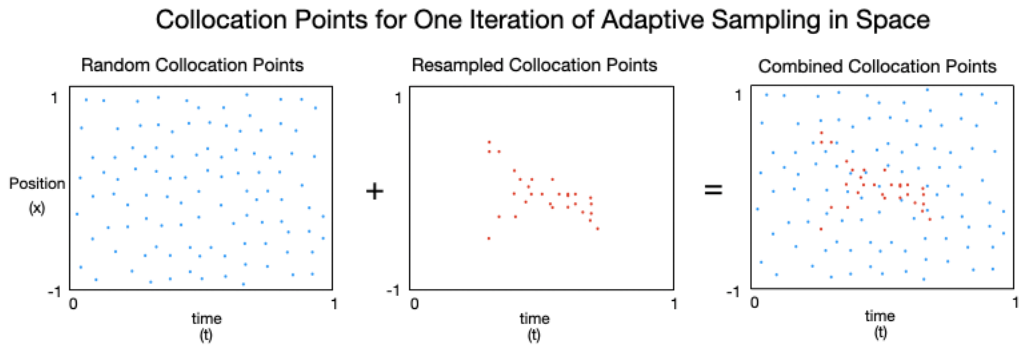

Fig. 2.1: This is what the collocation points for one iteration of re-sampled training might look like for a problem with domain $x \in [-1, 1]$ and $t \in [0, 1]$. The blue points show the set of randomly sampled collocation points using Latin hypercube sampling. Training on these points keeps the solution of the equation accurate across the whole domain. The red points show an example of a set of re-sampled collocation points sampled after evaluating the f-prediction network for the highest areas of error. These points help the solution to be more accurate in areas of higher difficulty. The combined set of points are the collocation points used to train the network. The network can repeat this process for multiple re-sampling iterations. The blue points will stay the same but the red points may change to focus on other parts of the domain that are not being learned well.

We first develop this approach while testing it on Burgers' equation. Here we are able to test the algorithm quickly on this simpler equation. It took some tries before we were able to match the accuracy obtained by the base PINN. We had to play around with the re-sampling strategy we used to evaluate the f-network such as: the number of points to test, the number of points of highest error to select, and the best way to choose the test points. We found what worked well was to use the LHS with the same number of original collocation points and then to take 10 to 20 percent of those points to add to the original.

For Burgers' equation we first train the network on the original 2,000 collocation points alone. We then generate another 2,000 points across the domain to act as test points. We test these points using the f-predict function and find the top 200 points where the error was the highest. We add these 200 points back to the original 2,000 collocation points and train again. For each re-sampling iteration we do the same thing by finding the points of highest error and adding those to the original collocation points. There will be 2,200 collocation points for each re-sampling iteration. For the optimization scheme, we train the

network using the Adam optimizer for up to 20,000 iterations, and then the same L-BFGS-B optimizer after that.

2.2.3 Mini-batching strategy to improve convergence

Mini-batching is a technique that has been used in deep learning to improve performance. Instead of using the entire data set to calculate the exact direction of the gradient, a subset of the data, called a batch or mini-batch, is used to evaluate the direction. The direction of the gradient for each batch may not be in the direction, but it has actually been shown to help avoid less desirable local minimum better than full-batch gradient descent [8].

We employ a mini-batching approach to the Allen-Cahn equation to see if we can get improved convergence.

2.2.4 Adaptive strategies in time

When the difficulty of the PDEs to be solved is increased further, even the PINN with adaptive sampling in space fails to converge to the actual solution. The focus here is to introduce adaptivity in time and space together. Mainly we introduce two approaches. The first time-adaptive approach is similar to the space adaptive method in that collocation points in time are strategically chosen to improve learning. The second time-adaptive method takes a different approach where we actually create separate networks on smaller (subsequent) time domains of fixed or adaptive length.

Time-adaptive approach I: adaptive sampling in time

At each time step of this approach, we require the data points, (initial, boundary, and general collocation both original and re-sampled) to come from within a specified time interval. For instance, if we are approximating the solution in the time domain $[0\ \ 1]$, we start with small time intervals $[0\ \ t_1]$, $t_1 > 0$, where t_1 is close to zero such as $t_1 = 0.1$. Then we gradually increase the span, i.e., $[0\ \ t_i]$, $i = 1, 2, \cdots, N$, with $0 < t_1 < t_2 < \cdots < t_N = 1$, when each time span is learned well until the solution is learned well on the whole domain. This idea is illustrated in Figure 2.2.

The first implementation allows the user to designate a list of time steps. For each time step collocation points will be sampled from only this restricted domain. Adaptive space sampling is used with the time restriction to improve learning. The network is then trained for each time interval using adaptive sampling in space. At each new iteration for the time step, the f-predicted error on that time domain is calculated. It was observed that for the Allen-Cahn equation this error stops improving when the network has learned the solution well on that time domain with a two norm of $9e{-}5$. Once the error is smaller than this, we move on to the next time step and re-sample the original collocations from the new extended domain. This is repeated at each time step until the whole domain is being trained on.

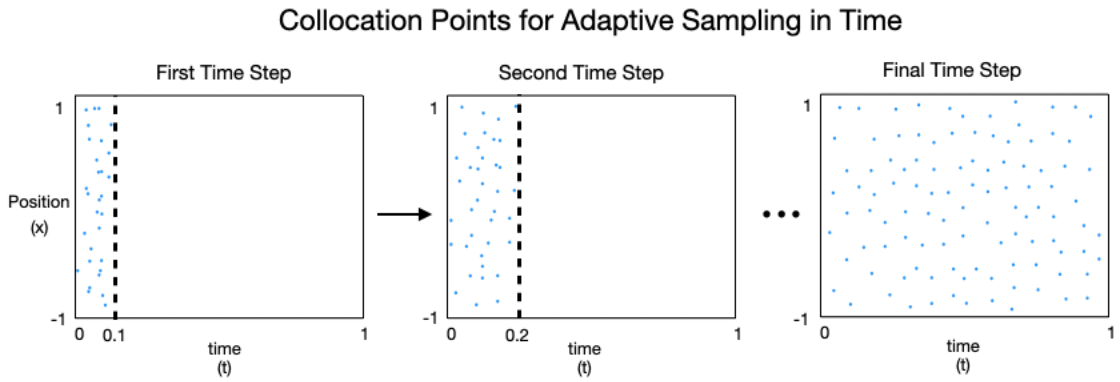

Fig. 2.2: This figure illustrates the concept behind adaptive time sampling method 1. For this problem the time domain is from 0 to 1. The first time step only allows data points to be taken between $t \in [0, 0.1]$. Once the f-predicted error on the interval is sufficiently small, collocation points are then chosen on the larger $[0, 0.2]$ time interval including adaptive space sampling. collocation points are still chosen from the earlier time domains to keep what has been learned there learned well. This is continued until the time interval is as large as the domain for the entire problem. Note this is all done on one PINN (in the next time method, multiple networks are created for each time interval.

We first test this method on the most difficult Allen-Cahn equation with a γ_2 parameter of 5 to compare the results with adaptive sampling in space alone.

Time-adaptive approach II: adaptive time marching strategy

In the second time-adaptive approach, we propose to split up the domain of interest into smaller problems. Notice that in the first time-adaptive approach, we only have a single network that focuses the collocation points adaptively in time. In the second approach, we actually create separate networks for each time step (interval). For example, if our domain of interest is $[0\ \ 1]$, we train one network to learn the solution on the interval from $[0, 0.1]$. Once the solution is learned well on this time interval, we train another network on the interval $[0.1, 0.2]$. A caveat here is we cannot use the initial condition for the later network as the given initial value is only valid for $t = 0$. We can use the solution from the previous time step's network for when $t = 0.1$ as the initial condition for the this time partition. We continue doing this until we have covered the whole time domain of the original problem. The individual networks can be combined to obtain solution at any point in the domain of the original problem. The idea is illustrated in Figure 2.3.

Fixed interval length. We started by testing this on the most difficult parameters for the Allen-Cahn equation where $\gamma_2 = 5$. This implementation uses fixed time interval lengths. We first wrote an algorithm that split the time domain into 10 evenly sized time partitions and train a network to learn each section. We created a network for each time interval where the time interval was of length 0.1. For each time interval we use the similar spatial re-sampling method where we do an initial training with the L-BFGS-B, and Adam optimizers with spatial re-sampling. Adam optimizer and L-BFGS-B optimizer max iterations were both set to 6000 with 4 re-sampling in space iterations per time interval.

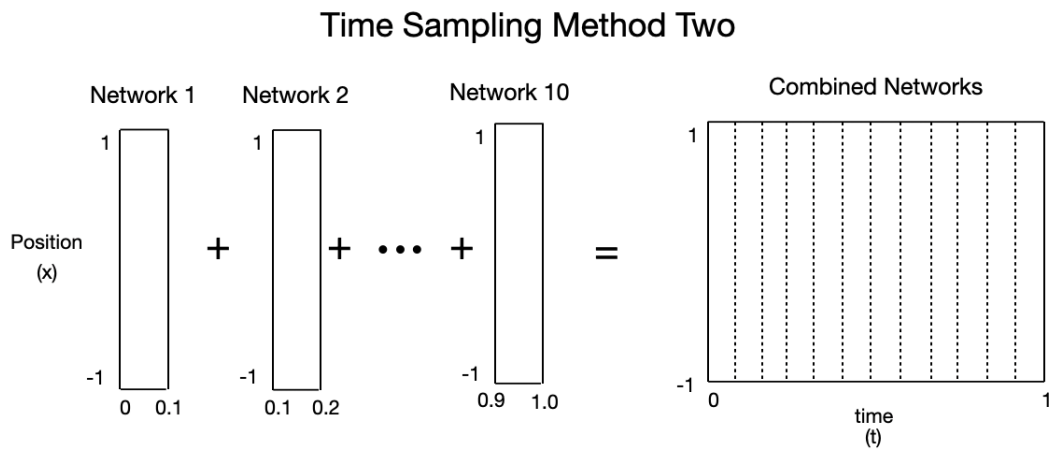

Fig. 2.3: This figure illustrates the concept behind adaptive time-marching strategy. Here individual networks are trained for each time step. Network 2 shares the same initial time as the last time in Network 1. Once Network 1 has been learned well its values at $t = 0.1$ can be used as the initial condition for Network 2. The solution for each individual network can be combined at the end into one continuous solution that covers the entire domain of the problem. Each network has the same time length, but can be bigger or smaller for parts of the domain that are easier or harder to learn.

Adaptive interval length. Instead of requiring the length of the time intervals to be set before hand, we allow for them to change as needed during the learning process. This should allow for more difficult intervals to be focused on using smaller intervals and less difficult intervals to be trained faster. A max interval length of 0.1 is chosen, but now we allow for the interval to become smaller by cutting it in half. We use the value of the loss function to evaluate whether or not the solution has been learned well. If the loss function is sufficiently small after training on the time partition then we move on to the next time interval using an interval length of 0.1 and repeat the process. We test this on the Allen-Cahn equation. We also try different max interval lengths to compare the accuracy and the loss function relationship.

CHAPTER 3

NUMERICAL RESULTS

In this section, we will provide several numerical tests on solving the Allen-Cahn equation and the Cahn-Hilliard equation with the improved the PINN.

3.1 Solving the Allen-Cahn equation

We first tested the Allen-Cahn equation as follows:

$$\begin{aligned} &u_t - 0.0001u_{xx} + 5u^3 - 5u = 0, \quad x \in [-1, 1], \quad t \in [0, 1], \\ &u(0, x) = x^2 cos(\pi x), \\ &u(t, -1) = u(t, 1), \\ &u_x(t, -1) = u_x(t, 1). \end{aligned} \tag{3.1}$$

Note that the original PINN paper did not test the solution of this equation using the same continuous technique as above, but we will use it as a baseline. The general set-up for this network architecture is the same as in Burgers' with the same number of neurons and same tanh activation function. However, the loss function must be changed to accommodate a different equation. We still have the same u-network and that portion of the loss function. We still have the f-network potion as well but now we formulate the terms to match the left hand side of this new equation. In contrast with Burgers' equation, we also add the different periodic boundary condition constraints to the loss function. The new loss function becomes:

$$MSE = MSE_u + MSE_f + MSE_b, \tag{3.2}$$

Where MSE_u and MSE_f are the same as before and

$$MSE_b = \frac{1}{N_b}\sum_{i=1}^{N_b} |u(t_b, x_u) - u(t_b, x_l)| + |u_x(t_b, x_u) - u_x(t_b, x_l)|. \tag{3.3}$$

Here N_b is the number of collocation points used on the boundary, t_b are the time values for those points, x_u is the upper bound for x, x_l is the lower bound. For this problem we have $x_u = 1$ and $x_l = -1$ from our domain. We see that both the expressions inside the absolute values are ideally zero if they follow the boundary conditions set by the equation.

For this trial we will again use 10,000 collocation points, 100 initial points, and now 100 boundary points. We will change the optimization procedure a little and use the method Raissi used for solving Schrodinger where the Adam optimizer is first used and then the L-BFGS-B optimizer.

Using the baseline approach outlined in Section 3.1, we are not able to solve the Allen-Cahn equation. The relative l_2 error is 0.99 or almost one. Looking at Figure 3.1, we see that the predicted solution at different time steps is not close to the actual solution.

The weighted loss function preforms slightly better with an error of 0.52, but the algorithm still fails to converge as seen in Figure 3.2. The network learns the solution better at points near the initial condition and near the boundary points, but the curves are not learned well in the middle of the domain where the collocation points are sampled randomly.

With adaptive sampling in space we obtain a much better solution. This performs 10 re-sampling iterations. For this case, after about 6 re-sampling iterations the solution does not improve as much. As seen in Figure 3.4, while the solution is much better, at the latter time steps the solution doesn't quite match the real solution. The error is improved from the others at 2.33e-02. See Table 3.1 for a table of other errors. It is important to highlight also that this improved accuracy was obtained with a fraction of the collocation points that the non-adaptive tests used. The adaptive test used only 2,000 collocation points while the others were tested with their typical 10,000. Note that this same iteration scheme was tried without adding re-sampled points and the solution does not converge as

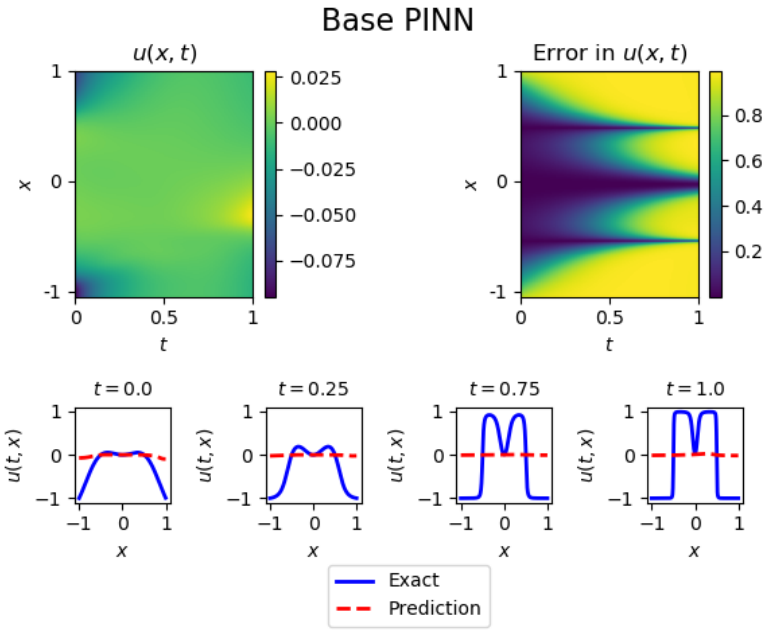

Fig. 3.1: Original AC equation as solved using the base PINN method. Here we use a network with 10,000 collocation points. The four plots on the bottom of the predicted solution vs the actual solution at different times. The predicted solution fails to model the solution well.

expected. The mini-batching trial with a mini-batch size of 32 performs almost as well as space sampling. Notice how for both these solutions the points near $t = 1$ and $x = 0$ are not exactly matching the actual solution. By using time sampling method 2, this is fixed completely where both end up matching.

Allen-Cahn	Original	Weighted Loss	Mini-batching	Space Sampling
Relative L^2	9.90e-1	5.22e-1	3.25e-2	2.33e-2
Relative L^1	9.90e-1	3.25e-1	8.80e-3	6.20e-3
Infinity-norm	9.96e-1	1.37	3.37e-1	2.64e-1

Table 3.1: The comparison of errors in the solution of the Allen-Cahn equation using the baseline approach vs the weighting the loss function and finally using adaptive sampling in space. This method produces the best results.

Next, we changed the initial condition for the Allen-Cahn equation and use various

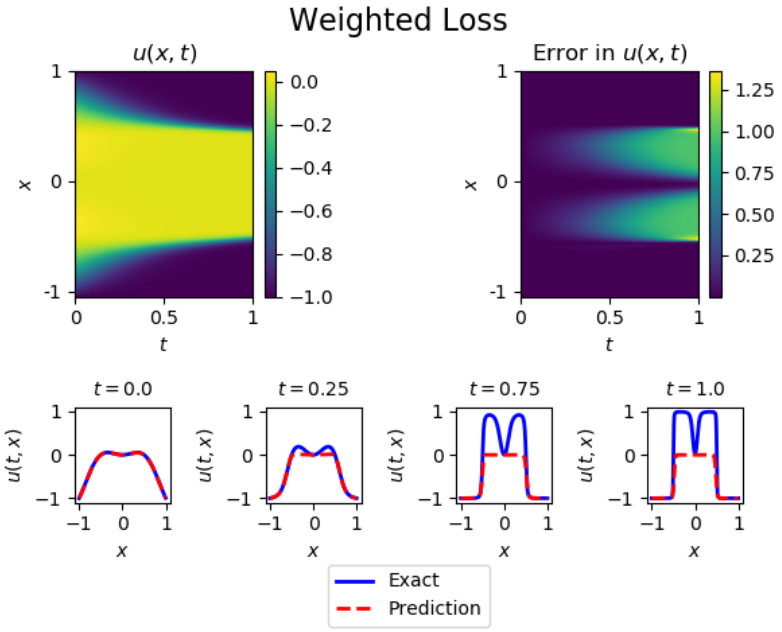

Fig. 3.2: AC weighted loss function. Here we are still using the base approach without changing the collocation point, but we use a weighted loss function. We put more weight on the initial condition vs the collocation and boundary conditions. The results in a slight improvement especially at times near t=0. The solution still fails to converge well.

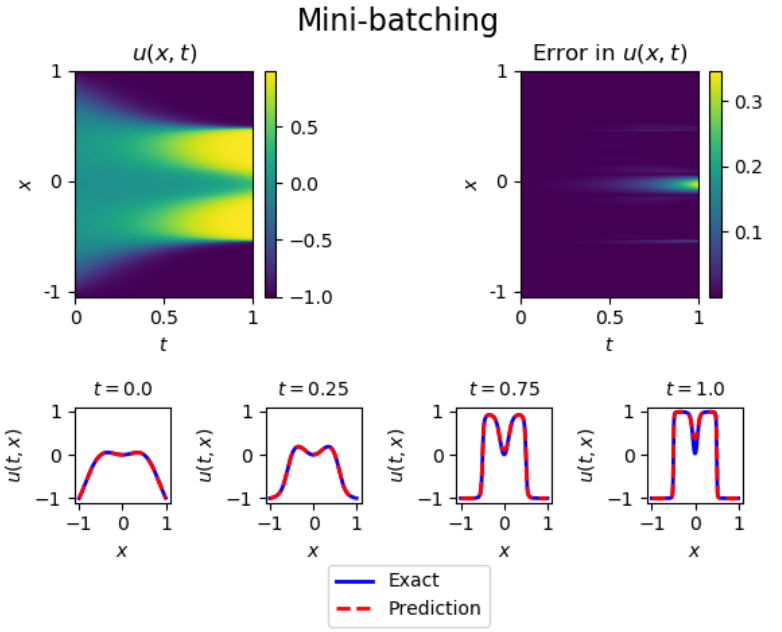

Fig. 3.3: AC with mini-batching helps improve the solution more than weighting alone.

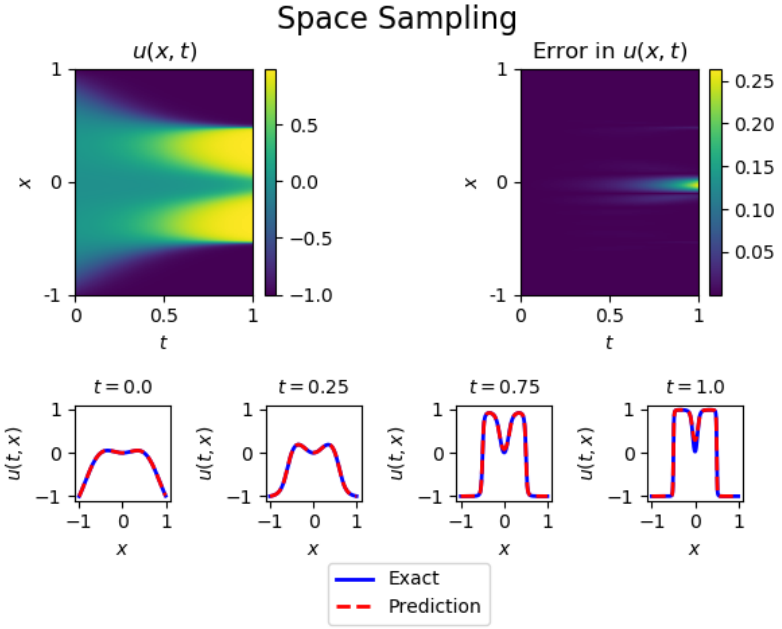

Fig. 3.4: AC weighted loss function with adaptive sampling in space. This implementation solves the equations much better than the base method. The focusing of collocation points in the areas of highest difficulty have allowed the network to learn the solution function.

values for the parameters to see how the methods work on different problems. In the following series of tests we use the following form of the equation:

$$\begin{aligned} &u_t - \gamma_1 u_{xx} + \gamma_2 u^3 - \gamma_2 u = 0, \quad x \in [-1, 1], \quad t \in [0, 1], \\ &u(0, x) = x^2 sin(2\pi x), \\ &u(t, -1) = u(t, 1), \\ &u_x(t, -1) = u_x(t, 1). \end{aligned} \tag{3.4}$$

This equation is different from the one previously tested in the initial condition and that the gamma parameters are not set and the initial condition has nearly double the periodicity. Instead of testing one set of parameters we will vary them, in particular γ_2, to see how our method works on parameters of increasing difficulty. We will keep γ_1 set to 0.0001 as in the previous problem.

We tested the adaptive re-sampling method using parameters $\gamma_2 = 1, 2, 3, 4$. We observed graphically that for smaller values of γ_2 such as 1, and 2 the proposed method converges in a reasonable number of re-sampling iterations. However, for $\gamma_2 = 3$ the approximated solution does not seem to converge within a reasonable number of re-sampling iterations. As seen in Figure 3.5, the inner sharp curves cannot be learned well. Note that this happens in fairly early time steps, before $t = .35$ and the error gets propagated and enlarged later on at time $t = 1$. It could be that a certain combination of the number of original collocation points with a number of re-sampled collocation points would help this solution to converge, we did not find one. This issue gets even worse for $\gamma_2 = 4$. The solution is unable to learn the larger curves in this case. See Figure 3.6. A summary of the errors with various γ_2 is shown in Table 3.2, which confirms that the difficulty of the equation does make the network perform worse.

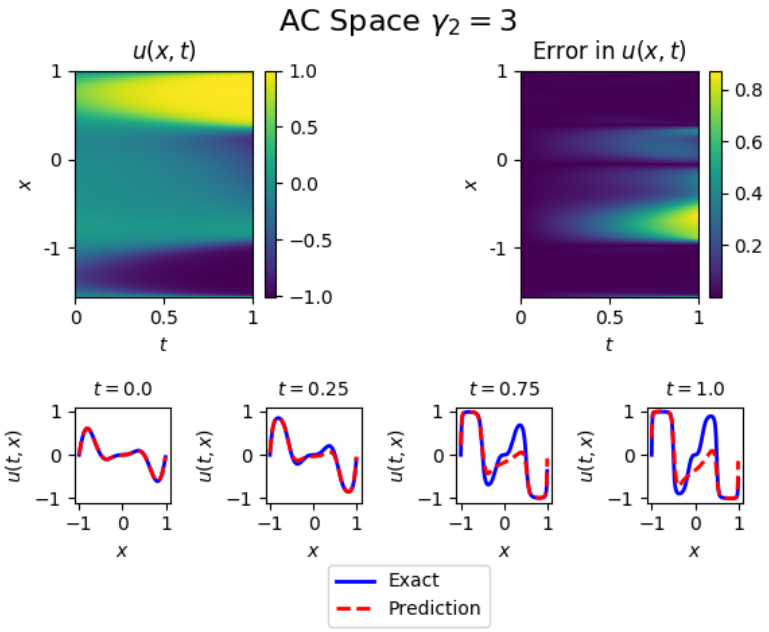

Fig. 3.5: Solution results of the AC ($\gamma_2 = 3$). For this value of gamma we see that the solution is not learned all the way even with adaptive sampling.

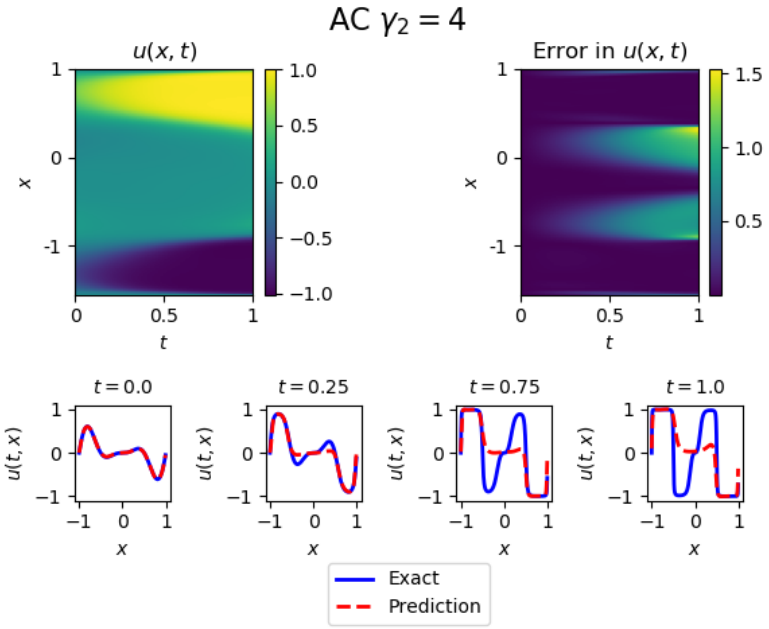

Fig. 3.6: AC with $\gamma_2 = 4$ with adaptive space re-sampling. The network does even worse at predicting the solution with this higher parameter.

γ_2 value	relative l_2 error
1.0	2.11e-2
2.0	4.40e-2
3.0	3.20e-1
4.0	5.14e-1

Table 3.2: Results for the comparison of the second Allen-Cahn Equation. As the γ_2 parameter increases the network performs worse.

Finally, we see the results for the trial using adaptive in time sampling method 1. Using this approach the solution for the more difficult problem with $\gamma_2 = 4$ can be learned much better with a relative l_2 error of 0.04. See Figure 3.7. The solution approximated by the network follows the actual solution well across the domain. As has been observed in other tests, the sharp curve when time is close to 1 is not learned perfectly. With time sampling method 2 we actually observe that difference to be smaller. For the time sampling method

2 on Allen-Cahn we saw that at times near $t = 1$ the solution is learned better than it was in the past.

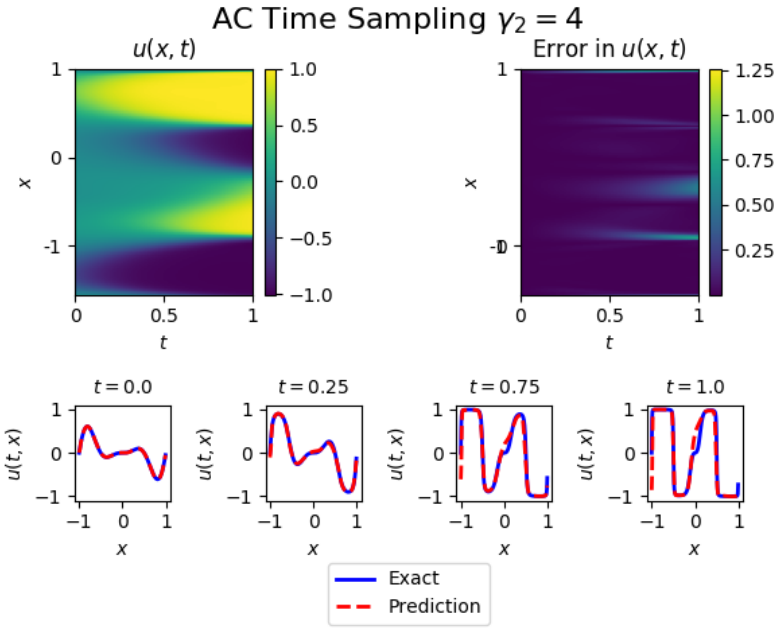

Fig. 3.7: Solution results of the AC ($\gamma_2 = 4$) solved with adaptive time and space. Previously this equation could not be solved using only adaptive in space re-sampling. With fixed time steps of length .1 this approach focuses on earlier times and then keeps expanding the interval to encompass the whole domain.

3.2 Solving the Cahn-Hilliard equation

Next we move onto the more difficult Cahn-Hilliard Equation:

$$\begin{aligned} &u_t - D(u^3 - u - \gamma u_{xx})_{xx} = 0, \quad x \in [-1,1], \quad t \in [0,1], \\ &u(0,x) = -cos(2\pi x), \\ &u(t,-1) = u(t,1), \\ &u_x(t,-1) = u_x(t,1). \end{aligned} \tag{3.5}$$

This is a phase field equation like the Allen-Cahn equation, but it has a higher order. For the following trials we use this equation with parameters $D = 0.01$ and $\gamma = 0.0001$ We

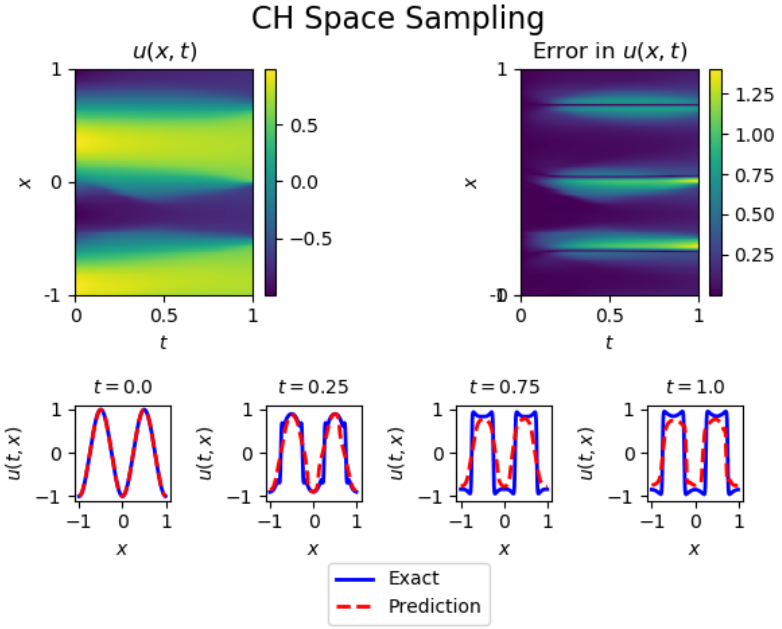

Fig. 3.8: Solution of the Cahn-Hillard equation solution using space sampling alone. Parameters: $D = 0.01$ and $\gamma = 0.0001$. Space sampling alone is not enough to achieve convergence even after many re-sampling iterations.

test the same adaptive time method here as with Allen-Cahn with the regular adjustments to the f-network. We also introduce an intermediate variable in the creation of the f-network and adapt the loss function accordingly. This helps in reducing the need to take the highest order derivatives and speeds up computation and accuracy.

This equation was tested using space sampling and time sampling. We see in the space sampling trial the solution does not converge. Even at times near $t = 0$ the error begins and continues throughout the entire domain. When we use time sampling we will see the difference it makes. It is able to learn the solution well near the $t = 0$, and by gradually allowing collocation points to be sampled at later times it maintains this accuracy across the whole domain.

Finally, we show the results for the trials run on the most difficult to learn of the equations thus far. Only methods that involve adaptive sampling in space and time are able to solve this equation well. The first trial used the Time Sampling Method 1 on the Cahn-Hilliard equation we obtain our best error rate yet of 9.51e-3. See Figure 3.9.

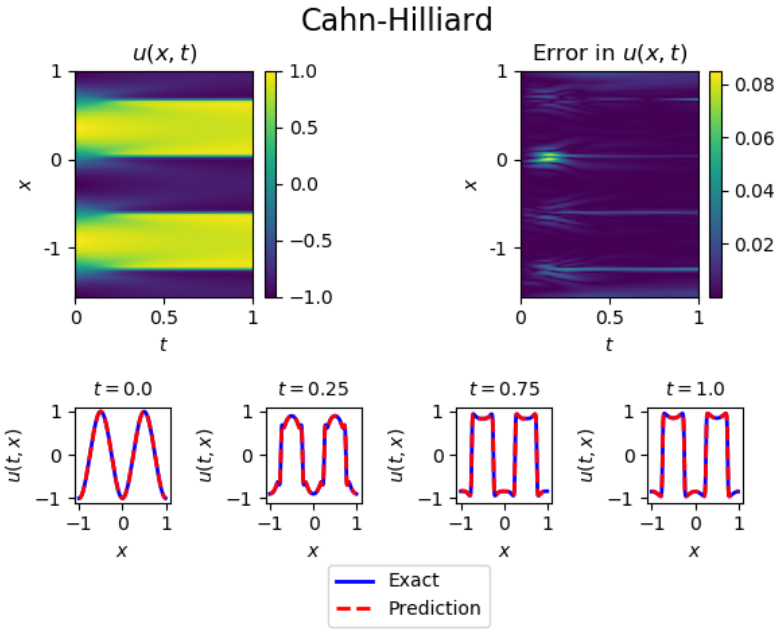

Fig. 3.9: Solution of the Cahn-Hillard with an l_2 relative error of 9.51e-3. The previous adaptive sampling in space method was unable to solve this equation, but with the addition of adaptive time sampling the best error rate yet is achieved on the most difficult problem.

The adaptive length interval did give better results on Cahn-Hilliard, but we noticed that with more individual time steps, the initial condition for the subsequent time step network is slightly off. With more networks up to a certain point we saw decrease accuracy.

At this point it is important to point out that as the complexity of the differential equation increases, so does the complexity of the PINN. The training times went from minutes to train on burgers equation to a few hours on average for training Allen-Cahn to multiple hours with Cahn-Hilliard. This shows the importance of focusing on efficient algorithms all the more important.

CHAPTER 4

DISCUSSION

In this thesis, we have introduced several strategies to improve the approximating capability of the Physics Informed Neural Networks (PINNs). And we have used to improved PINN to solve the phase field equations of increased complexity. Even though we focused on the problem of solving phase field equations, these techniques could prove useful in solving other difficult classes of partial differential equations as well. We have seen how adapting mathematical techniques and principles used on more classical methods can help us find useful approaches for PINNs as well. The best performance is obtained by using a combination of all of the techniques presented. More simple methods such as mini-batching, and adding weights in loss function are relevant, especially when they are combined with more powerful adaptive sampling methods.

Space sampling opened the door to other ideas of adaptive sampling. Space sampling uses the f-network predictions to pinpoint areas to focus data points on. Time sampling uses knowledge of differential equations to chose areas to focus on. We saw how both the value of the loss function and the values of the f-network predictions could help in determining if a network has learned a solution well. Using this information can help the network in making a decision such as whether to use more collocation points or whether focus on a smaller time domain.

We also saw merit in both of the time sampling methods. Time sampling method 1 proved better than just space sampling alone. Time sampling method 2 took a different approach that has the potential for even higher accuracy and learning the solution faster. It uses individual networks that can focus on a smaller problem domain which can potentially use more simple networks. A potential downside with more difficult equations is that if the solution gets off on a time interval, all the intervals after that will propagate that error. It is important to learn the solution well on a time interval before moving on to the next one.

This method may also be helpful when working on problems with larger domains.

This research has focused mainly on the problem of *solution* of differential equations. The next step of our future work is to test these approaches on the *discovery* of differential equations. The same neural network architectures can be used with the addition of a few more learnable/trainable parameters.

REFERENCES

[1] Marsha J Berger, Phillip Colella, et al. Local adaptive mesh refinement for shock hydrodynamics. *Journal of computational Physics*, 82(1):64–84, 1989.

[2] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[3] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[4] Vanessa Buhrmester, David Münch, and Michael Arens. Analysis of explainers of black box deep neural networks for computer vision: A survey, 2019.

[5] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning, 2017.

[6] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[7] Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.

[8] Robert Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does sgd escape local minima?, 2018.

[9] B. Mehlig. Artificial neural networks, 2019.

[10] George S. Misyris, Andreas Venzke, and Spyros Chatzivasileiadis. Physics-informed neural networks for power systems, 2019.

[11] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

[12] Peter J. Olver. *Introduction to Partial Diferential Equations*. Springer, Switzerland, 1993.

[13] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017.

[14] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations, 2017.

[15] Samuel Rudy, Steven Brunton, Joshua Proctor, and J. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3, 09 2016.

[16] Yeonjong Shin, Jerome Darbon, and George Karniadakis. On the convergence and generalization of physics informed neural networks, 04 2020.

[17] G. Wang, G. B. Giannakis, and J. Chen. Learning relu networks on linearly separable data: Algorithm, optimality, and generalization. *IEEE Transactions on Signal Processing*, 67(9):2357–2370, 2019.

APPENDICES