Physics Informed Neural Network for Phase-field Modeling of Brittle Fracture

by

Yuchen Xin

B.A., Tsinghua University, 2018 China

Thesis

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in the School of Engineering at Brown University.

Providence, Rhode Island

October 2022

This thesis by Yuchen Xin is accepted in its present form by the School of Engineering as satisfying the thesis requirement for the degree of Master of Science.

Date	**George Em. Karniadakis** **Advisor**

Approved by the Graduate Council

Date	**Thomas A. Lewis,** **Dean of the Graduate School**

Abstract

This thesis presents the applications of phase field modeling in fracture analysis. In this method, a diffusive crack zone controlled by a scalar auxiliary variable approximates the sharp crack surface topology in a solid. Because no interface tracking is necessary, the crack trajectories are automatically calculated as part of the solution in phase field modeling. Over the domain, the damage parameter changes continually. However, this flexibility comes with challenges because strong local gradients must be accurately represented by a very precise spatial discretization. The practical usefulness of phase field models is hence severely constrained.

In this work, I have developed an attention mechanism integrated physics informed deep neural networks to solve brittle fracture. The network is trained using a method that minimizes the variational energy of a system that is characterized by general non-linear partial differential equations while adhering to any specified physical law. For effective network optimization, self-adaptive trainable weight parameters are linked to the loss terms (boundary loss and the variational energy loss). In contrast to traditional mesh-based discretization techniques, the suggested solution just requires a set of points to specify the geometry. An adaptive h-refinement scheme is integrated with this framework, where the refinement is based on a critical threshold value of ϕ. In this work, we have considered displacement controlled loading conditions.

Contents

List of Figures

1 Background and Motivation

Fracture is defined as the whole or partial separation of a previously connected solid body caused by applying stress loads substantial enough to overcome the material's cohesive strength. Brittle fracture, which usually leads to sudden failure of the structure without warning, has been a major curiosity in structure designs. For engineers and scientists developing new materials with greater strength and toughness, or strong designs against failure, or for those concerned with an accurate estimate of a component's design life, an understanding of the microscopic mechanisms underlying material failure is essential. Therefore, from the perspective of safety, forecasting the initiation and propagation of the fracture is crucial. In this research field, a better understanding of the fracture process might enable the development and improvement of early warning systems.

The primary impetus for the original studies of fracture was indeed the failure of technical structures due to fracture which could not be explained by the available mechanical theories at the time. The discipline that is currently known as *fracture mechanics* was founded, in particular, by the theoretical writings of Griffith (1921) [4] and Irwin (1956) [5]. These early discoveries allowed the engineer to predict the likelihood of propagation of an existing break in brittle materials like hardened steel and ceramics. Griffith also laid the theoretical groundwork for the concept of fracture as an energy-driven process. According to him, the body must provide mechanical energy in order to separate the substance and create the new surfaces. Griffith and Irwin did not take the material's inertia into account; instead, they solely analyzed quasi-static circumstances. As long as the stress is applied slowly compared to a time scale given by the ratio of a characteristic length of the problem and a characteristic wave speed, the proposed theory provided accurate estimations.

Experiments are still the most reliable way to validate hypotheses and designs in science and engineering. However, experiments are frequently time consuming and expensive, leading

to a search for alternate methods like numerical modeling to analyze physical processes like fracture. Over the years, various numerical methods have been proposed to study the failure mechanism with some trade-offs related to the degree of accuracy and relative ease of modeling. The methods available in the literature to study the growth of fracture can be broadly classified into two categories: discrete approaches and continuum approaches. In discrete approaches, there is a need to track the discontinuities and the growth of fracture, which lead to significant challenges to predict crack behavior. Complications associated with the discrete approaches are overcome in the continuous modeling approaches. One popular continuous fracture modeling approach is the phase field modeling which incorporate the effects associated with the crack formation (like stress release) into the constitutive equation. Hence, tracking of the growth of the discontinuity is not required. In this dissertation, I have put forward a deep learning approach, commonly called the physics informed neural network [10], to solve the phase field based fracture. The approach incorporates attention mechanism to overcome the optimization challenges of a non-convex objective function.

2 Phase Field Modeling of Fracture

The phase field approach aims to simultaneously solve for the displacement field and fracture region by minimizing the variational energy of the system [3], thereby generalizing Griffith's theory for brittle fracture. A continuous scalar-valued phase field parameter, $\phi \in [0, 1]$ is used to indicate the phase of the material, with 0 representing the undamaged state of the material and 1 a fully damaged state. In this method, the diffusion of the discrete crack zone is controlled by a length scale parameter, l_0. In a limiting case of sharp crack topology, $l_0 \to 0$. Using the variational energy minimization approach, the brittle crack propagation problem is reformulated in terms of a coupled system of partial differential equations (PDEs). In this section, we present the theoretical formulation of the governing phase field model.

2.1 Phase Field Modeling for Brittle Fracture

Griffith [4] postulated that for brittle materials, crack propagation necessitates the formation of new surfaces with their corresponding surface energies, Ψ_c and came to the conclusion that this energy must be generated by the body as mechanical energy, Ψ_e for a fracture to grow. The energy release rate, which may be understood as a generalized force that is work-conjugated to the magnitude of crack advance, is the negative rate of change of a body's potential energy with regard to a crack advance when inertial effects are ignored. In a quasi-static loading condition, the total energy of the system, $\mathcal{E}$ can be written as

$$\begin{aligned} \mathcal{E} &= \Psi_e + \Psi_c - \mathcal{P}_{ext}, \\ \text{where } \Psi_e &= \int_\Omega \Psi_0 \left(\varepsilon\left(\mathbf{u}\right), \phi\left(\boldsymbol{x}\right)\right) d\Omega, \\ \Psi_c &= \int_{\Gamma_0} G_c \Gamma_{\phi,n} \, d\Omega_0 \, d\Gamma_0, \\ \mathcal{P}_{ext} &= \int_\Omega \mathbf{f} \cdot \mathbf{u} \, d\Omega \; + \; \int_{\partial\Omega_N} \mathbf{t}_N \cdot \mathbf{u} \, d\Gamma. \end{aligned} \tag{2.1}$$

In Eq. 2.1, Ψ_e is the stored elastic strain energy, Γ is the crack region, and Ψ_0 is the initial strain energy density functional expressed in terms of the small deformation strain tensor, $\varepsilon\left(\mathbf{u}\right)$, where $\mathbf{u}$ denotes the displacement field and ϕ is the phase field. The physical domain, $\Omega \subset \mathcal{R}^d$ is defined with the external boundary, $\partial\Omega \subset \mathcal{R}^{d-1}$, where $d \in \{1, 2, 3\}$ denotes the number of spatial dimensions. The surface energy, is defined in terms of the material property called *critical energy release rate*, G_c, and the crack density functional, $\Gamma_{\phi,n}$. $\Gamma_{\phi,n}$ is dependent on the phase field ϕ, its derivative up to order n, and the scale parameter $l_0 \in \mathbb{R}^+$ controlling the width of the continuous approximation to the discrete crack. For the second-order phase field model, $n = 2$. The external work $\mathcal{P}_{ext}$ is calculated using the boundary force $\mathbf{t_N}$ applied over the boundary, $\partial\Omega_N$ and the distributed body force, $\mathbf{f}$ over the whole domain.

In the concept of phase field approximation, the coefficients of $\phi(x)$ and its derivatives upto

order $n = 2$ are chosen such that:

$$\int_{-\infty}^{\infty} G_c \Gamma_n(\phi) dx \approx \int_{\Gamma} G_c \, d\Gamma. \tag{2.2}$$

Since the crack is modeled as a continuous region controlled by a scale parameter l_0, an exponentially decaying function is introduced to approximate the damage region. For 2D problems, a particular form of the second order phase field, $n = 2$ is:

$$\Gamma_{\phi,2} = \frac{1}{2l_0} \int_{\Omega} \left(\phi^2 + l_0^2 |\nabla \phi|^2\right) d\Omega. \tag{2.3}$$

The coefficients of $\phi(x)$ and its derivatives in Eq.2.3 are chosen such that it satisfies Eq.2.2. Hence, for the second order phase field model, Ψ_c is approximated as:

$$\Psi_c = \int_{\Gamma} G_c d\Gamma = \frac{G_c}{2l_0} \int_{\Omega} \left(\phi^2 + l_0^2 \left|\nabla \phi\right|^2\right) d\Omega. \tag{2.4}$$

2.2 Energy Decomposition

The total elastic energy is the sum of tensile and compressive energy components. The compressive strain energy does not participate in the propagation of the crack. To prevent cracking in regions under compression, a tension-compression split of $\Psi_0(\epsilon)$ is considered as:

$$\Psi_e(\epsilon) = g(\phi) \Psi_e^+(\boldsymbol{\epsilon}) + \Psi_e^-(\boldsymbol{\epsilon}), \tag{2.5}$$

where $g(\phi)$ represents the monotonically decreasing stress-degradation function, Ψ_e^+ and Ψ_e^- are tensile and compressive components of the strain energy, respectively. The stress degradation function,$g(\phi)$, reduces the stiffness of the bulk material in the region of the crack, and hence is applied to tensile component of elastic strain energy. Ψ_e^+ and Ψ_e^- are defined through a spectral decomposition of strain tensor as:

$$\boldsymbol{\varepsilon} = \mathbf{P} \boldsymbol{\Lambda} \mathbf{P}^T, \tag{2.6}$$

where $\mathbf{P}$ consists of the orthonormal eigenvectors of strain tensor and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \lambda_3)$ is a diagonal matrix of the principal strains. Then we define the decomposed components as:

$$\varepsilon^+ = \mathbf{P}\mathbf{\Lambda}^+\mathbf{P}^T, \tag{2.7}$$

$$\varepsilon^- = \mathbf{P}\mathbf{\Lambda}^-\mathbf{P}^T, \tag{2.8}$$

where $\mathbf{\Lambda}^+ = \text{diag}(\langle\lambda_1\rangle, \langle\lambda_2\rangle, \langle\lambda_3\rangle)$, $\mathbf{\Lambda}^- = \mathbf{\Lambda} - \mathbf{\Lambda}^+$, and $\langle\bullet\rangle_\pm = \frac{1}{2}(\bullet \pm |\bullet|)$.

For isotropic linear elastic material, the undamaged elastic energy density is given by:

$$\Psi_e(\epsilon) = \frac{1}{2}\lambda \text{tr}(\epsilon)^2 + \mu \text{tr}\left(\epsilon^2\right), \tag{2.9}$$

where λ and μ are Lam*é* constants. So the tensile and the compressive strain energy density are defined as:

$$\psi_e^\pm(\epsilon) = \frac{1}{2}\lambda \langle \text{tr}(\epsilon)\rangle_\pm^2 + \mu \text{tr}\left(\epsilon_\pm^2\right). \tag{2.10}$$

The stress degradation function, $g(\phi)$ should satisfy following properties [11]:

$$g(0) = 0, \;\; g(1) = 1, \;\; g(\phi) > 0 \text{ for } \phi \neq 0, \;\; g^{'}(0) = 0, \;\; g^{'}(1) > 0. \tag{2.11}$$

A common form of the stress-degradation function commonly used in the literature is [9]:

$$g(\phi) = (1-\phi)^2. \tag{2.12}$$

2.3 Irreversibility conditions on the crack

To enforce the irreversibility conditions on the growth of the crack, Miehe et al. [8] introduced the strain-history functional:

$$H(x,t) = \max_{s\in[0,t]} \Psi_0^+(\epsilon(\boldsymbol{x}, s)), \tag{2.13}$$

which contains the maxiimum tensile energy in the history of deformation and is updated in every load/displacement step. In Eq. 2.13, $\boldsymbol{x}$ is the co-ordinate of any material point in the domain. The definition of crack energy presented in Eq. 2.4 is modified as:

$$\Psi_c = \int_\Gamma G_c d\Gamma = \int_\Omega \frac{G_c}{2l_0}\left(\phi^2 + l_0^2 |\nabla\phi|^2\right) + g(\phi) H(x,t) d\Omega \tag{2.14}$$

The strain energy functional can also be used to introduce the initial defects into the simulation domain. For pre-existing cracks, the initial strain history function $H(\boldsymbol{x}, 0)$ could be defined as a function of the closest distance from $\boldsymbol{x}$ to the line l, which represents the discrete crack. In particular, we set

$$H(x,0) = \begin{cases} \frac{BG_c}{2l_0}(1 - \frac{2d(x,l)}{l_0}) & d(x,l) \leqslant \frac{l_0}{2} \\ 0 & d(x,l) > \frac{l_0}{2} \end{cases}, \tag{2.15}$$

where B is a scalar parameter that controls the magnitude of the scalar history field and is calculated as:

$$B = \frac{1}{1-\phi} \quad \text{for } \phi < 1. \tag{2.16}$$

3 Physics-informed Neural Networks

In this section, we present the components of physics informed neural network approach, which is employed as a tool to solve brittle fracture problems. Artificial neural networks (deep and shallow) are inspired by the functional units, neurons inside the human brain. Deep neural networks are distinguished from the conventional shallow neural networks by the number of hidden layers present in the network. In conventional shallow networks, we have one input layer, one output layer, and a hidden layer. On the other hand, in a deep neural network, there is more than one hidden layer. The intuition being, more hidden layers will make the network to be more expressive and hence, will provide more accurate results [2]. In this work, we have used fully connected feed-forward deep neural networks. The first layer is also called input layer, while the last is output layer. Other layers of the network are called hidden layers.

In PINNs approach, the parameters of the network are trained based on the physics of a problem (defined in terms of a PDE). The primary goal of this work is to obtain the crack path. This is generally achieved by applying a displacement/load increment until failure. In this work, we have considered a displacement-controlled failure. We have also assumed a constant displacement step, Δu. With this setup, the neural network is trained at each displacement

increment and the strain-history function is updated at each increment. Before we start to train the network, we initialize the weights of the network randomly from a Gaussian distribution using the Xavier initialization technique [1]. Once the weights are initialized, we begin training the neural network. To that end, we represent the displacement field, $\boldsymbol{u}$, and the phase field, ϕ by using a deep neural network.

$$(\boldsymbol{u}, \phi) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\theta}). \tag{3.1}$$

In order to compute the parameters of the neural network at the i-th displacement step, we generate collocation points over the desired domain and simulate a forward pass to obtain the field variables. In the next step, we use the automatic differentiation technique to compute the displacement gradients, $\nabla \boldsymbol{u}$ and the eigenvalues of the strain, $\lambda_1, \ldots, \lambda_d$. The computed eigenvalues are then used to obtain Ψ^+ and Ψ^-.

$$\Psi^+ = \frac{\lambda}{8}(\lambda_s + |\lambda_s|)^2 + \frac{\mu}{4}\sum_{i=1}^{d}(\lambda_i + |\lambda_i|)^2, \tag{3.2a}$$

$$\Psi^- = \frac{\lambda}{8}(\lambda_s - |\lambda_s|)^2 + \frac{\mu}{4}\sum_{i=1}^{d}(\lambda_i - |\lambda_i|)^2, \tag{3.2b}$$

where $\lambda_s = \sum_{i=i}^{d} \lambda_i$. In the fourth step, we utilize Ψ^+ and Ψ^- to compute the elastic energy and the history function, $H(\boldsymbol{x}^g, i)$ as:

$$H(\boldsymbol{x}, i) = \max\left\{\Psi^+, H(\boldsymbol{x}, i-1)\right\}, \quad \text{where } i > 0. \tag{3.3}$$

The crack is initialized at $i = 0$ using Eq. (2.15). In the next step, we use the automatic differentiation technique to obtain the phase field gradients, $\nabla\phi$ and $\Delta\phi$ and then use the gradients and $H(\boldsymbol{x}, i)$ to compute the elastic energy and the fracture energy, depending on the desired phase field model.

Next, we compute Ψ_e and Ψ_c and then we approximate the total energy. Finally, we minimize the total variational energy to compute the parameters of the network, $\boldsymbol{\theta} = [\mathbf{W}, \mathbf{b}]$, where $\mathbf{W}$ are the weights and $\mathbf{b}$ are the biases. For optimization, we use the Adam (adaptive momentum)

optimizer followed by a quasi-Newton method (L-BFGS). We note that the above mentioned steps are to be repeated for each displacement step. A loss function is a measurable sign of the accuracy of a neural network. We define the loss function as:

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_{var} + \mathcal{L}_{bound}, \\ \mathcal{L}_{var} &= \Psi_e + \Psi_c, \\ \mathcal{L}_{bound} &= \sum_{i=1}^{n} \left(\boldsymbol{u}_i - \mathcal{N}(x_i; \boldsymbol{\theta}) \right)^2, \end{aligned} \tag{3.4}$$

where $\mathcal{L}_{var}$ is the loss term from the energy loss and $\mathcal{L}_{bound}$ is the loss term to satisfy the boundary conditions. Fig. 3.1 shows how a typical deep neural network is trained. The commonly used

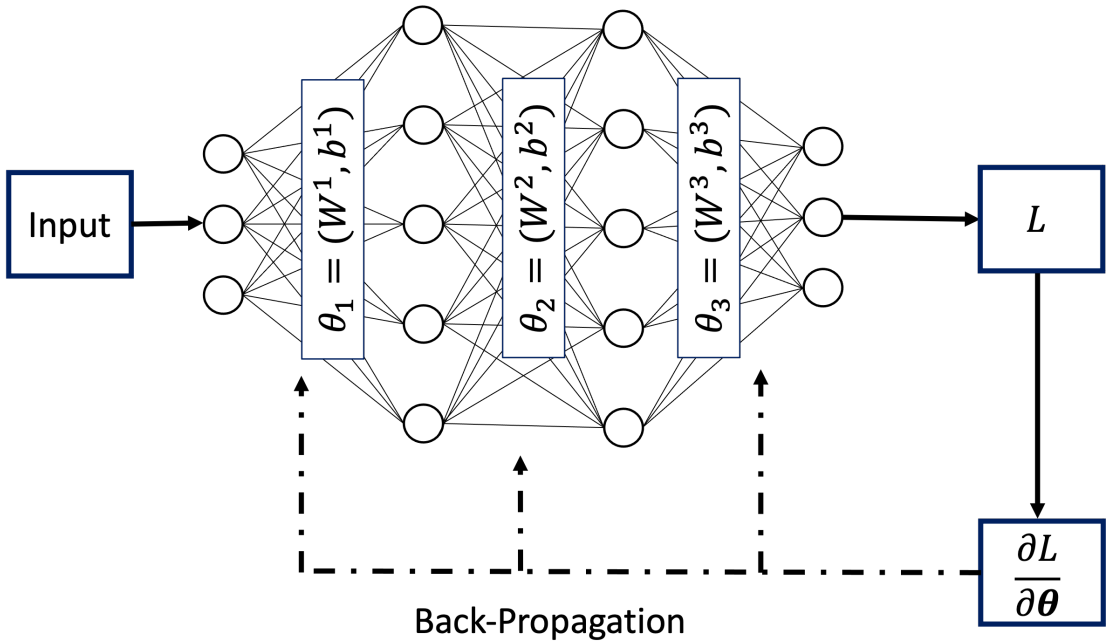


Figure 3.1: Schematic representation of a physics informed neural network.

activation functions include *sigmoid* functions, *ReLU* or *Leaky-ReLU* functions. The training of a neural network is based on a selected *training set* and later facilitates the prediction on a *testing set*. The trainable parameters are the weights of each neuron in the network. Through the backpropagation algorithm, the network is able to minimize a *loss function*.

3.1 Adaptive Mesh for Phase Field PINNs

The analysis of fracture usually requires a very fine mesh along the crack propagation path. However, a very fine mesh for the whole domain results in extremely expensive Computational cost. So in this thesis, we developed an adaptive h-refinement scheme to generate mesh for the PINN method.

The geometry is modeled with NURBS patches. The `get_E_r` method in `pf_pinn` is provided as residual elastic loss. In this chapter a h-refinement scheme is developed and integrated with an efficient data transfer algorithm to bridge between the coarse and the fine meshes. In the, h-refinement scheme, the polynomial order of the shape function remains constant during refinement while the number of elements in the mesh increases. We start from a tensor product mesh (marked as level 0) as the base mesh. For each following refinement step, selected elements from level k mesh are subdivided into 2^d sub-cells as the level $k+1$ mesh. The refined elements are then marked and will become inactive in following refinement procedure. While the subdivided elements (marked as children) will be set as active. Supposing the crack propagates along the region Γ_l, refinement will only occur in elements inside Γ_l. This will reduce the analysis and storage of data at each analysis step.

3.2 Attention Mechanism in Physics-Informed Neural Networks

One problem with the PINNs based phase field is that the loss function of the network consists two parts: one for elastic and crack energy density and another for boundary loss term. The two terms are of significantly different magnitude so penalizing factors are associated with each loss term for an efficient optimization. Usually, these weights are manually chosen but requires several trials to find out the optimized value. In this section, an adaptive algorithm is introduced so the optimized weight are adaptively obtained as a part of the training process.

The attention mechanism was first used in recurrent neural networks. In the thesis we took similar ideas in phase field problem. The boundary loss term << energy loss term, and hence

an adaptive weight is applied to the boundary term to match the magnitude of energy term. This results in a faster convergence. While previously proposed weighting methods produce improvements in stability and accuracy over the vanilla PINN, they are neither adaptive nor introduce inflexible adaption. In this section, fully-trainable weights produce a multiplicative soft attention mask, in a manner that is reminiscent of attention mechanisms used in computer vision. Instead of hard-coding weights at a particular region of the solution, the newly proposed method is in agreement with the neural network philosophy of self-adaptation [6], where its weights in the loss function are updated by gradient descent along with the network weights. The proposed self-adaptive PINNs utilizes the following loss function:

$$\mathcal{L}\left(w,\lambda\right)=\mathcal{L}_{var}\left(w\right)+\mathcal{L}_{bound}\left(w,\lambda_b\right), \tag{3.5}$$

where λ are trainable, non-negative *self-adaptation weights* for the boundary points, and

$$\mathcal{L}_{bound}\left(w,\lambda_b\right)=\frac{1}{2}\sum_{N_b}^{i=1} m\left(\lambda_b^i\right)\left|\mathcal{B}_x\left[u\left(\mathbf{x}_b^i;\mathbf{w}\right)\right]-g\left(\mathbf{x}_b^i\right)\right|^2, \tag{3.6}$$

where the *self-adaptive mask function* $m\left(\lambda_b\right)$ defined on (λ) is a non-negative, differentiable, strictly increasing function on λ. A key feature of self-adaptive PINNs is that the loss $\mathcal{L}$ is minimized with respect to the network weights w as usual, but is maximized with respect to the self-adaptation weights.

$$\min_{w}\max_{\lambda_b}\mathcal{L}\left(w,\lambda_b\right), \tag{3.7}$$

Considering the updates of a gradient descent/ascent approach to this problem:

$$w^{k+1}=w^k-\eta_k\nabla_w\mathcal{L}\left(w^k,\lambda_b^k\right), \tag{3.8}$$

$$\lambda_b^{k+1}=\lambda_b^k+\rho_b^k\nabla_{\lambda_b}\mathcal{L}\left(w^k,\lambda_b^k\right), \tag{3.9}$$

where $\eta^k>0$ is the learning rate for the neural network weights at step k, $\rho_b^k>0$ is a separate learning rate for the self-adaptive scheme. The mask function, $m^{'}\left(\lambda\right)$ is strictly increasing by assumption and $\nabla_{\lambda_b}\mathcal{L}\geq 0$ and any gradient component is zero if and only if the corresponding

unmasked loss is zero. This shows that the sequences of weights are monotonically increasing, provided that the corresponding unmasked losses are non-zero. Furthermore, the magnitude of the gradients $\nabla_{\lambda_b}\mathcal{L}$, and therefore of the updates, are larger if the corresponding unmasked losses are larger. In addition, the magnitude of updates can be controlled by specifying a schedule for the learning rates ρ_p^k, for $p = b$, adding extra flexibility. This progressively penalizes the network more for not fitting the residual, boundary, and initial points closely. We remark that any of the weights can be set to fixed, non-trainable values, if desired.

Notice that the self-adaptive weights need to be initialized at the beginning of the training. They could be set a fixed value or randomly generated over an interval, similarly as to how neural network weights did. Here, prior knowledge plays an important role; e.g., if it is known that the initial conditions in a problem are hard to fit, then the weights of initial condition could be set to a larger value than others. The shape of function g affects mask sharpness and training of the PINNs. Examples include polynomial masks $m(\lambda) = c\lambda^q$, for $c, q > 0$, and sigmoidal masks. In practice, the polynomial mask function has to be kept below a suitable value, in order to avoid numerical overflow. The sigmoidal function, on the other hand, sigmoidal masks do not have this issue, and can be used to produce sharp masks. It starts small for small starting values of the self-adaptive weight λ, and later exceed a certain threshold. The mask value will quickly take on the upper saturation value. Similarly to neural network non-linearities, a sigmoid mask function also suffers from vanishing gradients during training. This is particularly a problem at the lower starting value. Therefore, excessively sharp sigmoidal mask functions should be avoid. In this thesis, a typical *polynomial* function is chosen as the mask function for the network. It's controlled by a trainable value λ in the `pf_pinn` class.

Formally, SA-PINN training corresponds to a *penalty method* to solve the previous optimization problem. The gradient ascent/descent step can be implemented easily using off-the-self neural network software, by simply flipping the sign of $\nabla_{\lambda_b}\mathcal{L}$. In the implementation of SA-PINNs, we use Pytorch 1.10 with a fixed number of iterations of Adam. In some cases, these are

followed by another fixed number of iterations of the L-BFGS quasi-Newton method to reach a better convergence. This is consistent with the baseline PINN formulation, as well as follow-up literature. However, due to the algorithm in Pytorch, per-parameter tuning is disabled in L-BFGS method. The adaptive weights only updated in Adam training routine.

4 Numerical Examples

In this section we present the numerical problem solved as a part of the work to validate the proposed approach. In this example, we consider a unit square plate with a horizontal notch at the bottom quarter height from left edge to the center of the plate. The geometrical setup and boundary conditions as shown in Fig 4.1

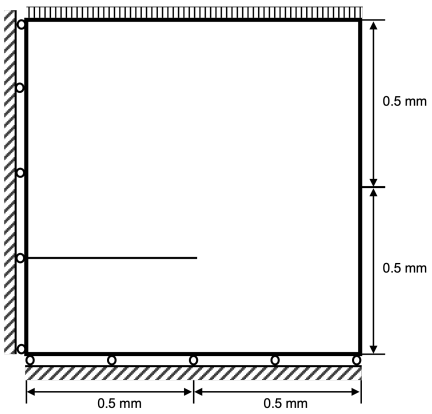


Figure 4.1: Geometrical setup and boundary conditions for the single-edge notched plate subjected to tensile loading.

4.1 Code Structure

The main structure of neural network is maintained in `pf_pinn` class which is a subclass of `torch.nn.Module`. In the `__init__` method, all geometric and material property parameters are defined, as well as all trainable parameters and degradation function. The trainable parameters from `self.net` which indicates the weights and bias of each layer in the feed-forward network, and an additional λ as the trainable weight. The main structure of the code is in `main.py`.

The training process comes in two parts: the first part with Adam optimizer with weight decay and the second part with L-BFGS. Loshchilov and Hutter states that although L2-regularization and weight decay is the same in vanilla SGD algorithm, they are different in gradient decent optimizers with momentum [7]. While optimizer with weight decay in addition to L2-regularization usually provides better convergence. Limited-memory BFGS is an optimization algorithm in the family of quasi-Newton methods while approximating the BFGS algorithm with a limited computer memory. As a quasi-Newton method, LBFGS provides better result.

The class `pf_pinn` inherited from `torch.nn.Module` is the basic of the neural network. The class contains necessary material parameters and methods for energy calculation. It takes two arguments, `args` for basic parameters and `df` for the type of degradation function (default as 'quadratic'). Only quadratic and cubic degradation function are previously defined. The tensile-plate code for PINN offers two way for energy computation, with or without stress decomposition. Stress decomposition aims to use an isotopic model for the elastic field and an anisotropic model for teh phase field. In tensile loading problem, the eigenvalue decomposition in Pytorch `torch.linalg.eigh` is not optimized with batched input (torch.1.10.0)[1] and therefore is expensive for large tensors. Although the decomposition operation can be replaced by `torch.linalg.svd` which supports batched input, but it may lead to unstable gradient in backward computation. In the problem of notched plate under tensile loading, no stress decomposition is needed. But in the `pf_pinn`, methods for computing loss energy as well as history functions using stress decomposition are provided. (class `get_energy_dcp` and class `get_hist_dcp`).

Degradation function: The choice of degradation function plays an import role in obtained an accurate crack path. The script for the degradation function is:

[1] https://github.com/pytorch/pytorch/issues/69528

```
def deg_func(self,x):
        '''
        Degradation function - default: quadratic
        '''
        if self.df == 'quadratic':
            n = 2
            g = (1-x)**n
        if self.df == 'cubic':
            n = 3
            g = (1-x)**n
        else:
            sys.exit('Degradation function not defined')

        return g
```

```
class pf_pinn(nn.Module):
    '''
    Class for phase field model
    '''
    def __init__(self,args,df='quadratic'):
        super(pf_pinn,self).__init__()

        if df == 'quadratic':
            self.deg_func = lambda p: (1-p)**2
        elif df == 'cubic':
            self.deg_func = lambda p: (1-p)**3
        else:
            sys.exit('Degradation function not defined')

        self.length = args.L
        self.height = args.W
        self.dist = nn.MSELoss(reduction='sum')
        self.vdelta = 0.0
        self.k = args.k

        self.gc = 3
        self.l = args.l0
        self.crackTip = [args.L/2.0, args.W/4.0]
        self.B = 90

        self.E = 210*1e3
        self.nu = 0.3 # Poission's ratio
        self.mu = 0.5 * self.E/(1 + self.nu)
        self.lmd = self.nu * self.E /((1 - 2*self.nu)*(1 + self.nu))

        self.c11 = self.E*(1-self.nu)/((1+self.nu)*(1-2*self.nu))
        self.c22 = self.E*(1-self.nu)/((1+self.nu)*(1-2*self.nu))
        self.c33 = self.E*(1-self.nu)/((1+self.nu)*(1-2*self.nu))
        self.c12 = self.E*self.nu/((1+self.nu)*(1-2*self.nu))
        self.c21 = self.E*self.nu/((1+self.nu)*(1-2*self.nu))
```

```
        self.c31 = 0.0
        self.c32 = 0.0
        self.c13 = 0.0
        self.c23 = 0.0
        self.c44 = self.E/(2*(1+self.nu))

        self.lb = torch.tensor([0.0,0.0],dtype = torch.float32,device=device)
        self.ub = torch.tensor([self.length,self.height],dtype = torch.float32,device=device)

        # Trainable parameters
        self.lambda_b = nn.Parameter(torch.randn((1,1)))
        self.net = nn.ModuleList([])
        for i, h_dim in enumerate(args.dim):
            if i == 0:
                in_channels = 2
            self.net.append(nn.Linear(in_channels, h_dim))
            self.net.append(nn.Tanh())
            in_channels = h_dim
        self.net.append(nn.Linear(args.dim[-1], 3, bias=True))

    def get_energy_n(self,x,w,hist_prev): # Calculate energy without stress decomposition
        torch.cuda.empty_cache()

        y = self.forward(x)
        u,v,phi = y[:,0:1],y[:,1:2],y[:,2:3]

        phi_g = self.grads(phi, x)
        u_g = self.grads(u, x)
        v_g = self.grads(v,x)
        self.u_g = u_g.detach()
        self.v_g = v_g.detach()

        # degradation function
        g = (1-phi)**2

        hist = self.get_hist_n(hist_prev)

        sigx = self.c11 * u_g[:,0:1] + self.c12 * v_g[:,1:2]
        sigy = self.c21 * u_g[:,0:1] + self.c22 * v_g[:,1:2]
        txy = self.c44 * (u_g[:,1:2] + v_g[:,0:1])
        E_e = 0.5 * g * (sigx*u_g[:,0:1] + sigy*v_g[:,1:2] + txy*(u_g[:,1:2] + v_g[:,0:1]))
        E_e = E_e * w

        nabla = phi_g[:,0:1]**2 + phi_g[:,1:2]**2
        E_c = 0.5 * self.gc * (phi**2/self.l + self.l*nabla) + g * hist
        E_c = E_c * w

        return E_e.sum(),E_c.sum()
```

```
    def init_hist(self,x): # Initialize history function
        init_hist = torch.zeros(x[:,0].size(),device=device)
        a = torch.sqrt((x[:,0]-self.crackTip[0])**2 + (x[:,1]-self.crackTip[1])**2)
        b = torch.abs(x[:,1]-self.crackTip[1])
        dist = torch.where(x[:,0] > self.crackTip[0], a, b)
        init_h = torch.where(dist < 0.5*self.l, self.B*self.gc*0.5*(1-(2*dist/self.l))/self.l,
        ↪  init_hist)
        self.init_h = init_h[:,None]
        return init_h[:,None]
```

For obtaining the boundary loss and the self adaptive weights:

```
    def get_weight(self):
        l_b = torch.exp(self.lambda_b)
        return l_b**2 * 1000

    def get_bc(self,x):
        k = int(self.k/4)
        y = self.forward(x)[:,0:2]
        target = y.detach().clone()
        # upper bound: v = vdelta
        target[0:k,1:2] = self.vdelta
        # bottom:  v=0
        target[k:(k*2),1:2] = 0.0
        # left u = 0
        target[(k*2):(k*3),0:1] = 0.0
        bc = self.dist(y[:,0:1],target[:,0:1]) + self.dist(y[:,1:2],target[:,1:2])
        return bc
```

The training function for Adam and L-BFGS is listed as follows:

```
    def train_adap(model,epoch,x,w,h,opt_n,opt_w): # With mixed precision
    torch.cuda.empty_cache()
    model.train()
    with autocast():
        E_e,E_c = model.get_energy_n(x,w,h)
        d = model.get_bc(x)
        l_b = model.get_weight()
        loss = E_e +E_c + l_b * d  #loss = E_e + 1.0 * E_c + 1e3 * bc for step 0

    train_loss[epoch,0]=float(loss)
    train_E_e[epoch,0]=float(E_e)
    train_E_c[epoch,0]=float(E_c)
    train_bc[epoch,0]=float(d)
    train_lb[epoch,0]=float(l_b)

    opt_n.zero_grad()
    opt_w.zero_grad()
```

```
        scaler.scale(loss).backward()
        # set gradient of adaptive weights for boundary loss as negative
        model.lambda_b.grad.data = -model.lambda_b.grad.data
        scaler.step(opt_n)
        scaler.step(opt_w)
        scaler.update()
```

```
    def train_lbfgs(model,epoch,x,w,h,opt):
    torch.cuda.empty_cache()
    model.train()
    if epoch == (ep1 + 1000):
        lr = 0.005
        adjust_learning_rate(opt,lr)
    def closure():
        E_e,E_c = model.get_energy_n(x,w,h)
        d = model.get_bc(x)
        l_b = model.get_weight()
        loss = E_e +E_c + l_b * d

        opt.zero_grad()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(),max_norm) # gradient clipping
        train_loss[epoch,0]=float(loss)
        train_E_e[epoch,0]=float(E_e)
        train_E_c[epoch,0]=float(E_c)
        train_bc[epoch,0]=float(d)
        train_lb[epoch,0]=float(l_b)
        return loss

    opt.step(closure)
```

In neural networks, attention is a technique that mimics cognitive attention. The effect enhances some parts of the input data while diminishing the other parts. Learning which part of the data is more important than others depends on the context and is trained by gradient descent. The optimizer used in this code Adam with weight decay and L-BGFS algorithm [7]. A gradient clipping scheme is applied to avoid exploding of gradients. During the gradient clipping, when the gradient norm in backward propagation exceeds a pre-determined threshold, it is re scaled down to match the norm as:

$$\mathbf{g} \longleftarrow c \cdot \frac{\mathbf{g}}{\|\mathbf{g}\|} \quad \text{if } \|\mathbf{g}\| \geq c$$

In this work we have employed mixed precision with the Pytorch extension called “Apex”.

4.2 Result

In this section we discuss the results obtained using the proposed approach. Figure 4.2 presents the initial discretization of the collocation points with which we start the simulation. The adaptively refined meshes are shown in Figure 4.4. To refine the meshes, we have chosen a critical threshold value of $\phi = 0.2$ to identify the zones of refinement. The scatter plots for the growth of the crack is shown in Figure 4.5. In our proposed approach, we have adopted the idea of self-adaptive weights in our loss function. The adaptive weights are employed with the boundary loss term and are adaptively updated along with the network parameters. The adaptive update of the weights for representative steps is shown in Figure 4.3.

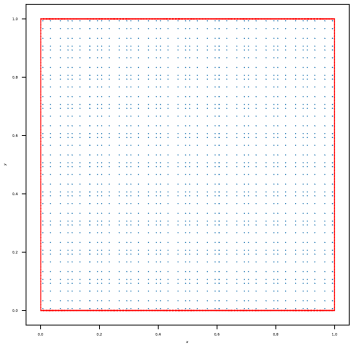
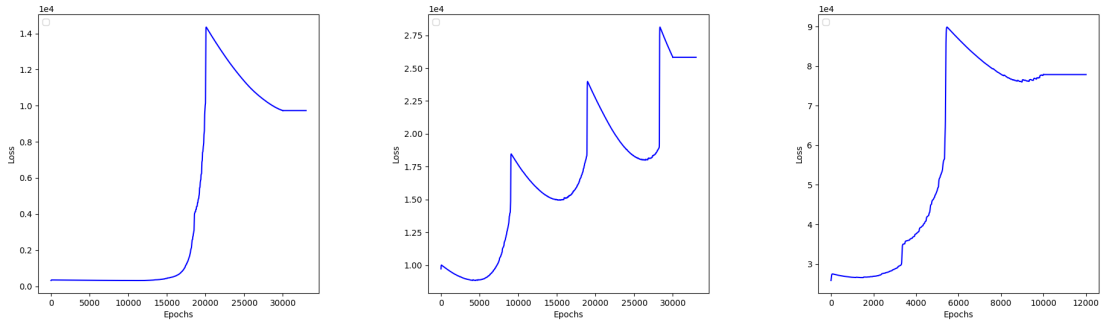
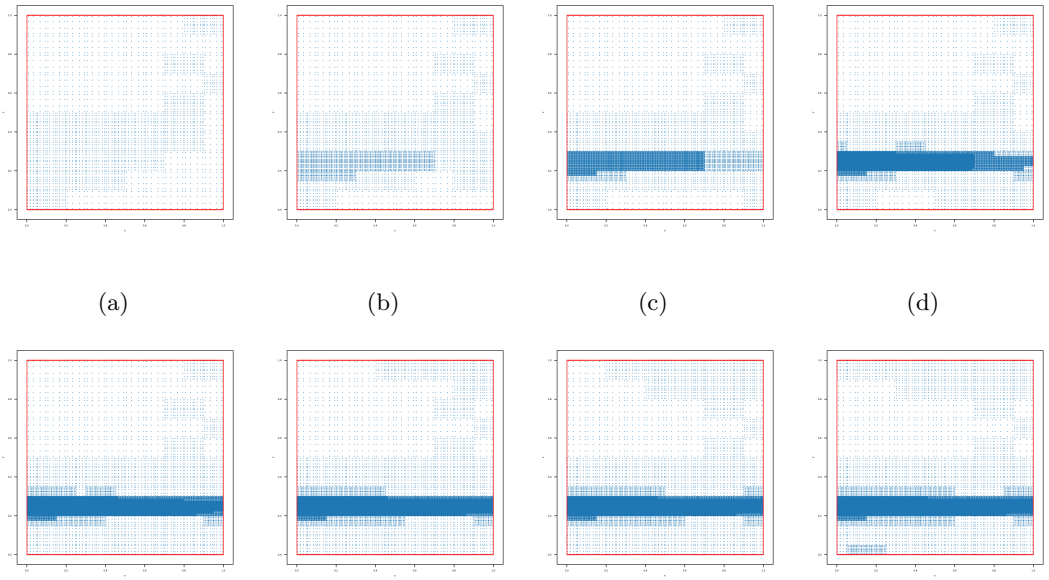


Figure 4.2: Initial Training discretization



(a) Step 1 (b) Step 4 (c) Step 5

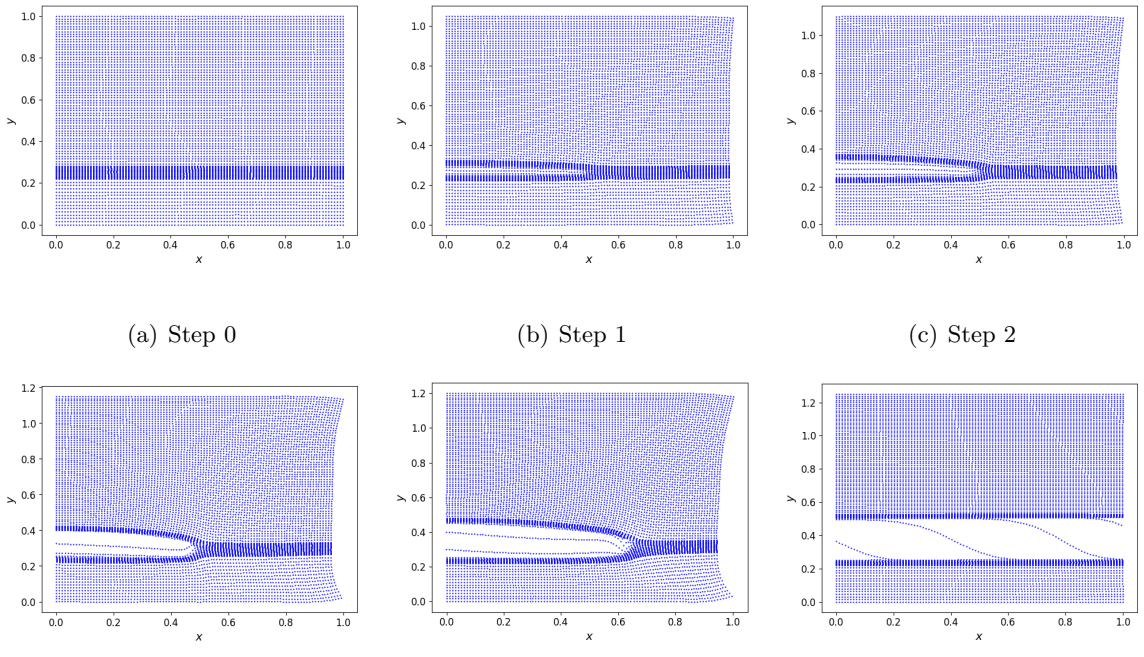
Figure 4.3: Representative plots for adaptive weights for the loss function for steps 1, 4, and 5.



(a) (b) (c) (d)

(e) (f) (g) (h)

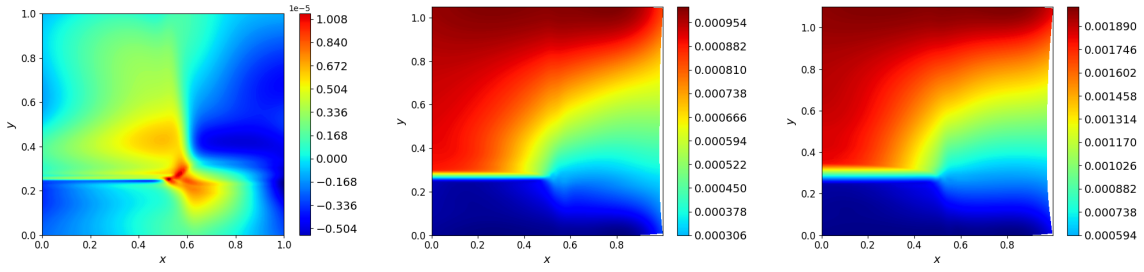
Figure 4.4: Adaptive mesh refinement of the tensile plate problem. The critical threshold value of $\phi = 0.2$, which allows to choose the refinement zone.



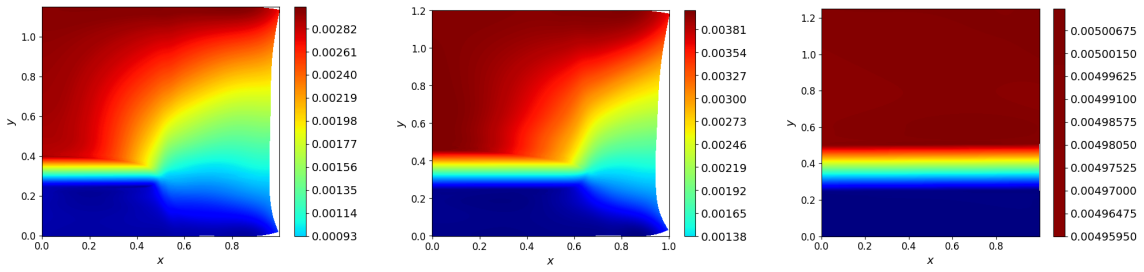
(a) Step 0 (b) Step 1 (c) Step 2

(d) Step 3 (e) Step 4 (f) Step 5

Figure 4.5: Scatter plots for the growth of the tensile crack displacement-controlled loading conditions.



(a) Step 0 (b) Step 1 (c) Step 2

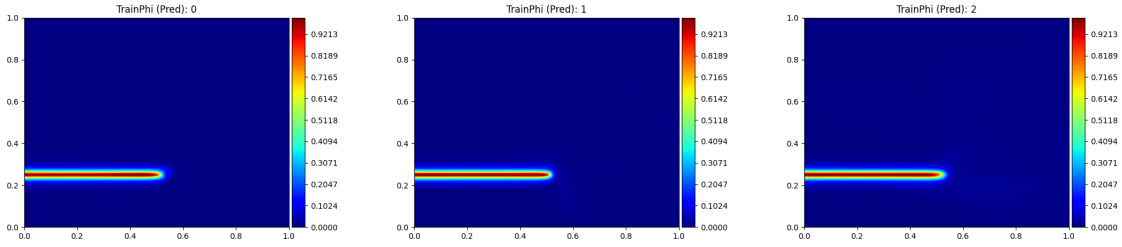


(d) Step 3 (e) Step 4 (f) Step 5

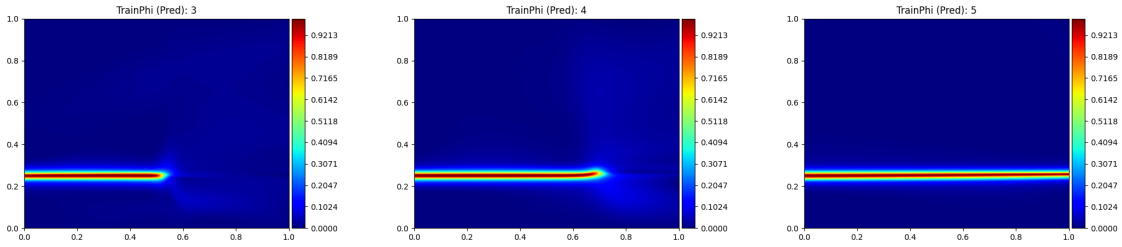
Figure 4.6: Plots showing the predicted y-displacement for prescribed displacement for prescribed displacement steps. The crack is initialized at Step 0, no displacement is applied.

5 Conclusion

In this work, we have proposed an attention mechanism integrated physics informed neural network (PINN) algorithm for predicting the crack path using the phase-field approach. The zone of the crack is adaptively refined with more collocation points as the crack grows. The phase field parameters is used to track the region for refinement. In order to compute the total variational energy of the system in an efficient manner, we utilize NURBS to model for building the problem domain/geometry. For efficient numerical integration of the fracture zone, the problem domain is discretized into a number of elements and then collocation points are generated each element. Due to the non-convex nature of the loss function, self-adaptive penalty parameters are employed with the loss terms which are updated during the training of the network. For an efficient training, an adaptive refinement scheme is considered to generate more collocation points in the region of the crack growth. The proposed approach is implemented for



(a) Step 0 (b) Step 1 (c) Step 2



(d) Step 3 (e) Step 4 (f) Step 5

Figure 4.7: Plots showing the predicted phase-field for prescribed displacement steps. Step 0 is the crack initialization.

a tensile plate problem and is presented in this work. As an extension of this work, we plan to integrate transfer learning with this model.

References

[1] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[3] Somdatta Goswami, Cosmin Anitescu, Souvik Chakraborty, and Timon Rabczuk. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theoretical and Applied Fracture Mechanics*, 106:102447, 2020.

[4] Alan Arnold Griffith. Vi. the phenomena of rupture and flow in solids. *Philosophical transactions of the royal society of london. Series A, containing papers of a mathematical or physical character*, 221(582-593):163–198, 1921.

[5] G.R. Irwin. Onset of fast crack propagation in high strength steel and aluminum alloys. Technical report, Naval Research Lab, Washington DC, 1956.

[6] Katiana Kontolati, Somdatta Goswami, Michael D Shields, and George Em Karniadakis. On the influence of over-parameterization in manifold based surrogates and deep neural operators. *arXiv preprint arXiv:2203.05071*, 2022.

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[8] Christian Miehe, Martina Hofacker, and Fabian Welschinger. A phase field model for rate-independent crack propagation: Robust algorithmic implementation based on operator splits. *Computer Methods in Applied Mechanics and Engineering*, 199(45-48):2765–2778, 2010.

[9] Christian Miehe, Fabian Welschinger, and Martina Hofacker. Thermodynamically consistent phase-field models of fracture: Variational principles and multi-field fe implementations. *International journal for numerical methods in engineering*, 83(10):1273–1311, 2010.

[10] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[11] Juan Michael Sargado, Eirik Keilegavlen, Inga Berre, and Jan Martin Nordbotten. High-accuracy phase-field models for brittle fracture based on a new family of degradation functions. *Journal of the Mechanics and Physics of Solids*, 111:458–489, 2018.