FastVPINNs: An Efficient tensor based library for solving PDEs using hp-Variational Physics informed Neural Networks

02 May 2023

Introduction

Partial differential equations (PDEs) are essential in modeling physical phenomenon such as Heat transfer, Electromagnetics, Fluid Dynamics etc. The current progress in the field of scientific machine learning has resulted in incorporating deep learning and data-driven methods to solve PDEs. Physics informed neural networks introduced by [@raissi2019physics] works by incorporating the governing equations of the physical systems into the Neural Network architecture. Lets consider a two-dimensional Poisson equation as an example which reads as follows

$$-\nabla^2 u(x) = f(x), \quad \Omega \in \mathbb{R}^2, \tag{1}$$

$$u(x) = g(x), \quad x \in \partial\Omega \tag{2}$$

Here $x \in \Omega$ is the spatial coordinate, $u(x)$ is the solution of the PDE, $f(x)$ is the source term and $g(x)$ is the value of the solution on the boundary $\partial\Omega$.

Physics informed neural networks (PINNs) introduced by [@raissi2019physics] works by incorporating the governing equations of the physical systems and the boundary conditions as physics-based constrains to train the neural networks. The core idea of PINNs lies in the ability of obtaining the gradients of the solutions with respect to the input using the automatic differention routines available within deep-learning frameworks such as tensorflow[@tensorflow]. The loss function for the PINNs can be written as follows

$$
\begin{aligned}
L_p(W,b) &= \frac{1}{N_T}\sum_{t=1}^{N_T} \left|(-\Delta u(x;W,b) - f(x))\right|^2 ,\\
L_b(W,b) &= \frac{1}{N_D}\sum_{d=1}^{N_D} \left|u_{\mathrm{NN}}(x;W,b) - g(x)\right|^2 ,\\
L_{\mathrm{PINN}}(W,b) &= L_p + \tau L_b,
\end{aligned}
$$

where $u_{\mathrm{NN}}(x;W,b)$ is the output of the neural network with weights W and biases b. The $u(x;W,b)$ is the solution of the PDE obtained by the neural network. where N_T is the total number of training points in the interior of the domain Ω, N_D is the total number of training points on the boundary $\partial\Omega$ and τ is a scaling factor applied to control the penalty on the boundary term and $L_{\mathrm{PINN}}(W,b)$ is the loss function of the PINNs.

Variational Physics informed neural networks(VPINNs)[@kharazmi2019variational] are an extension to the PINNs, where the weak form of the PDE is used to train the neural network. The weak form of the PDE is obtained by multiplying the PDE with a test function and integrating over the domain. hp-VPINNs[@kharazmi2021hp] was developed to increase the accuracy using domain decomposition using h-refinement(increasing number of elements) and p-refinement(increasing the number of test functions).The loss function of hp-VPINNs with `N_elem` elements in the domain can be written as follows

$$
\begin{aligned}
L_v(W,b) &= \frac{1}{\texttt{N\_elem}}\sum_{k=1}^{\texttt{N\_elem}} \left|\int_{K_k} \left(\nabla u_{\mathrm{NN}}(x;W,b)\cdot\nabla v_k \;-\; f\,v_k\right)\,dK\right|^2 ,\\
L_b(W,b) &= \frac{1}{N_D}\sum_{d=1}^{N_D} \left|u_{\mathrm{NN}}(x;W,b) - g(x)\right|^2 ,\\
L_{\mathrm{VPINN}}(W,b) &= L_v + \tau L_b,
\end{aligned}
$$

Where K_k is the k^{th} element in the domain, v_k is the test function in the corresponding element. $L_v(W,b)$ is the loss function for the weak form of the PDE and $L_{\mathrm{VPINN}}(W,b)$ is the loss function of the hp-VPINNs. For more information on the derivation of the weak form of the PDE and the loss function of the hp-VPINNs, refer to [@anandh2024fastvpinn].

Statement of need

The existing implementation of hp-VPINNs framework suffers from two major challenges. One being the inabilty of the framework to handle complex geometries and the other being the increased training times associated with the increase in number of elements within the domain. In the work[anandh2024fastvpinns], we presented FastVPINNs, which addresses both these the challenges of hp-VPINNs by using bilinear transformation to handle complex geometries and uses a tensor based loss computation to reduce the dependency of training time on number of elements. Current Implementation of FastVPINNs can acheive an speedup of upto 100 times when compared with the existing implementation of hp-VPINNs. We have also shown that with proper hyperparameter selection, FastVPINNs can outperform PINNs both in terms of accuracy and training time, especially for problems with high frequency solutions.

Our FastVPINNs framework is built using tensorflowv2.0[@tensorflow2015-whitepaper], and provides an elegent API for users to solve both forward and inverse problems for PDEs like Poisson, Helmholtz and Convection-Diffusion equations.The framework is well documented with examples and API documentation, which can enable users to get started with the framework with ease. The framework also provides an API of all the modules for users to write their custom functionalities to suit their needs. With the current level of API abstraction, users should be able to solve PDE with less than 5 API calls as shown in [@sec:minimal-working-example].

This ability of the framework to allow users to train a hp-VPINNs to solve a PDE both faster and with minimal code, can result in wide spread application of this method on lot of real-world problems, which often requires complex geometries with large number of elements within the domain.

Modules

The FastVPINNs framework consists of 5 core modules, They are

- Geometry - Handles mesh generation
- FE2D - Handles the finite element test functions and their gradients
- Data - Handles tensor assembly
- Physics - Handles the tensor based loss computation for the weak form of the PDE
- Model - Handles the dense NN for training the PDE

Geometry Module:

This module provides the functionality to define the geometry of the domain. The module provides the functionalities to either generate a quadrilateral mesh internally or to read an external `.mesh` file. The module also provides the functionality to obtain boundary points for complex geometries.

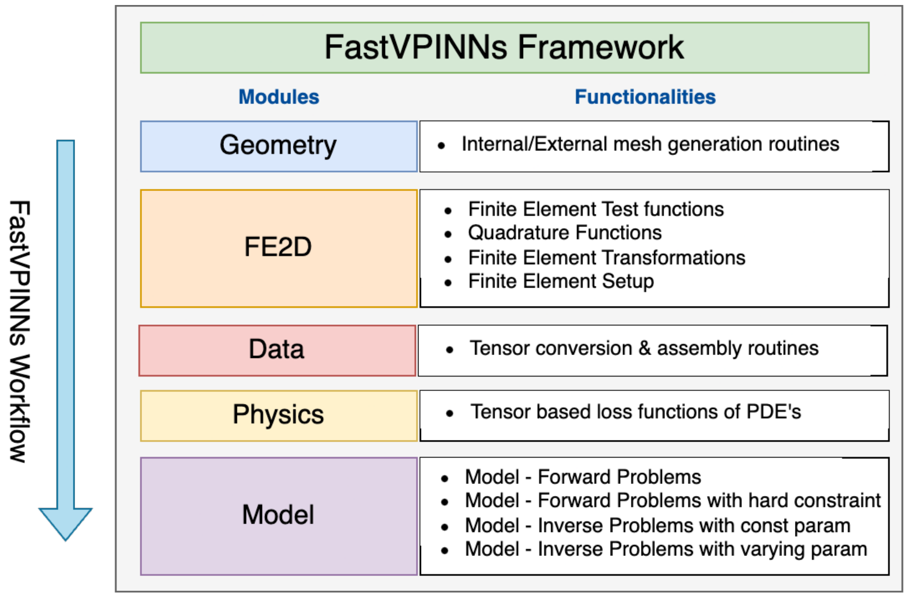
FastVPINNs Framework
Modules
Functionalities
Geometry
Internal/External mesh generation routines
FE2D
Finite Element Test functions
Quadrature Functions
Finite Element Transformations
Finite Element Setup
Data
Tensor conversion & assembly routines
Physics
Tensor based loss functions of PDE's
Model
Model - Forward Problems
Model - Forward Problems with hard constraint
Model - Inverse Problems with const param
Model - Inverse Problems with varying param
FastVPINNs Workflow

Figure 1: FastVPINNs Modules

FE2D Module:

The FE2D module is responsible for handling the finite element test functions and their gradients along with the quadrature and transformation routines. The module's functionality can be broadly classified into into four categories,

FE2D Module:

The FE2D module is responsible for handling the finite element test functions and their gradients. The module's functionality can be broadly classified into into four categories,

- **Finite Element Test functions**: provides the test function values and its gradients for a given element. This has been implemented for basis functions like Lagrange, Jacobi polynomials etc.
- **Quadrature functions**: Provides the quadrature points and weights for a given element based on the quadrature order selected by the user. These values will be used for numerical integration of the weak form of the PDE.
- **Transformation functions**: Provides the implementation of transformation routines such as `Affine` transformation and `Bilinear` transformation. This can be used to transform the basis function values and gradients from the reference element to the actual element.
- **Finite Element Setup**: Provides functionality to setup the basis functions, quadrature rules and the transformation for every element and save them in a common wrapper to access them. Further, it also hosts functions

to plot the mesh with quadrature points, assign boundary values based on the boundary points obtained from the geometry module, calculation of the forcing term etc.

Data Module:

The Data module is responsible for two main functionalities, it collects data from all modules which are required for training to be converted into a tensor data type with required precision and it assembles the basis function gradient and values to form a 3D tensor, which will be used during the loss computation.

Physics Module:

The physics of the framework contains functions, which are used to compute the loss function for the weak form of different PDEs in our novel tensor based approach. The functions accept the basis function tensors and the predicted gradients from the Neural Network along with the forcing matrix and computes the PDE residual.

Model Module:

This contains custom subclasses of the tensorflow.keras.Model class, which are used to train the neural network. The module provides the functionality to train the neural network using the tensor based loss computation and also provides the functionality to predict the solution of the PDE for a given input. It can also be used to save and load the trained model.

Minimal Working Example

With the higher level of Abstraction and API provided by the FastVPINNs framework, users can solve a PDE with less than 5 API calls. Below is a minimal working example to solve a Poisson equation using the FastVPINNs framework.

```
#load the geometry
domain = Geometry_2D("quadrilateral", "internal",
                     100, 100, "./")
cells, boundary_points =
                domain.generate_quad_mesh_internal(\
                x_limits=[0, 1], y_limits=[0, 1],\
                n_cells_x=4, n_cells_y=4,\
                num_boundary_points=400)

# Note: Users need to provide the necessary boundary values and the forcing function

# load the FEspace
fespace = Fespace2D(domain.mesh,cells,boundary_points,\
    domain.mesh_type,fe_order=5,\
    fe_type="jacobi",quad_order=5,\
    quad_type="legendre", \
    fe_transformation_type="bilinear"\
    bound_function_dict=bound_function_dict,\
    bound_condition_dict=bound_condition_dict,\
    forcing_function=rhs,\
    output_path=i_output_path,\
    generate_mesh_plot=True)

# Instantiate Data handler
dh = datahandler2D(fespace, domain, dtype=tf.float32)

# Instantiate the model with the loss function for the model
model = DenseModel(layer_dims=[2, 30, 30, 30, 1],\
            learning_rate_dict=0.01,params_dict=params_dict,
            ## Loss function of poisson2D
            loss_function=pde_loss_poisson,
            input_tensors_list=\
                [in_tensor,
                dir_in,
                dir_out],
            orig_factor_matrices=\
                [dh.shape_val_mat_list,
                dh.grad_x_mat_list,
                dh.grad_y_mat_list],
            force_function_list=dh.forcing_function_list,\
            tensor_dtype=tf.float32,
            use_attention=i_use_attention,
            activation=i_activation,
            hessian=False)

# Train the model
for epoch in range(1000):
    model.train_step()
```

Testing

The FastVPINNs framework has a strong testing suite, which tests the core functionalities of the framework. The testing suite is built using the pytest framework and is integrated with the continuous integration pipeline provided by Github Actions. The testing functionalites can be broadly classified into the three categories,

- **Unit Testing**: Covers the testing of individual modules and their functionalities.

- **Integration Testing**: Covers the overall flow of the framework. Different PDE's are solved with different parameters to check if the accuracy after training is within the acceptable limits. This ensures that the collection of modules work together as expected.
- **Compatibility Testing**: The framework is tested with different versions of python such as 3.8, 3.9, 3.10, 3.11 and on different versions of OS such as Ubuntu, MacOS and Windows to ensure the compatibility of the framework across different platforms.

Acknowledgements

We thank Shell Research, India, for providing partial funding for this project. We are thankful to the MHRD Grant No.STARS-1/388 (SPADE) for partial support.

References