Deep learning driven self-adaptive hp finite element method

Maciej Paszyński[1], Rafał Grzeszczuk[1],
David Pardo[2,3,4], and Leszek Demkowicz[5]

[1] AGH University of Science and Technology, Poland
{paszynsk,grzeszcz}@agh.edu.pl
[2] The University of the Basque Country, Bilbao, Spain
dzubiaur@gmail.com
[3] Basque Center for Applied Mathematics, Bilbao, Spain
[4] IKERBASQUE
[5] Oden Institute, The University of Texas at Austin, USA
leszek@oden.utexas.edu

Abstract. The finite element method (FEM) is a popular tool for solving engineering problems governed by Partial Differential Equations (PDEs). The accuracy of the numerical solution depends on the quality of the computational mesh. We consider the self-adaptive *hp*-FEM, which generates optimal mesh refinements and delivers exponential convergence of the numerical error with respect to the mesh size. Thus, it enables solving difficult engineering problems with the highest possible numerical accuracy. We replace the computationally expensive kernel of the refinement algorithm with a deep neural network in this work. The network learns how to optimally refine the elements and modify the orders of the polynomials. In this way, the deterministic algorithm is replaced by a neural network that selects similar quality refinements in a fraction of the time needed by the original algorithm.

Keywords: Partial Differential Equations, Finite Element Method, Adaptive algorithms, Neural networks

1 Introduction

The self-adaptive hp-Finite Element Method (FEM) has been developed for many years by the community of applied mathematicians working in the field of numerical analysis [5, 6, 3, 4, 9]. They require extremely high numerical accuracy, which is difficult to obtain by other numerical methods. In this paper, we refer to the iterative Algorithm 1 proposed by [3], and we introduce the simplified, one-step, Algorithm 2 as a kernel for the selection of the optimal refinements for the interiors of elements. The edge refinements are adjusted by taking the minimum of the corresponding orders of interiors. We further propose how to replace Algorithm 2 with a Deep Neural Network (DNN).

The DNN can make similar quality decisions about mesh refinements as Algorithm 2, while the online computational time is reduced. The main motivation for

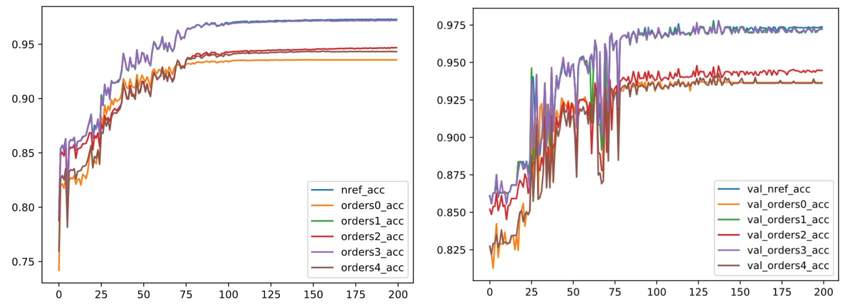


Fig. 1. The convergence of accuracy of training (left) and validation (right) datasets.

this work is the following observation. We have noticed that making random of 10 percent of the decision about element refinements made by the self-adaptive *hp*-FEM algorithm does not disturb the algorithm's exponential convergence. Thus, the possibility of teaching the deep neural network making decisions optimal up to 90 percent is enough to keep the exponential convergence.

Algorithm 1: Self-adaptive *hp*-FEM algorithm

```
   Input: Initial mesh, PDE, boundary conditions, error
   Output: Optimal mesh
 1 coarse mesh = initial mesh
 2 Solve the coarse mesh problem
 3 Generate fine mesh
 4 Solve the fine mesh problem
 5 if maximum relative error > accuracy then
 6     return fine mesh solution
 7 end
 8 Select optimal refinements for every hp finite element from the coarse
     mesh (Call algorithm 2)
 9 Perform all required h refinements
10 Perform all required p refinements
11 coarse mesh = actual mesh
12 goto 2
```

2 Self-adaptive *hp*-FEM with neural network

We focus on the L-shape domain model problem [5, 6] to illustrate the self-adaptive applicability hp-FEM algorithm for the solution of a model problem with a singular point. The gradient of the solution tends to infinity, and intensive mesh refinements are needed to approximate this behavior properly.

We describe in Algorithm 1 the self-adaptive *hp*-algorithm, initially introduced by [3]. It utilizes Algorithm 2 for the selection of the optimal refinements over element K. This algorithm delivers exponential convergence of the numerical error with respect to the mesh size, which has been verified experimentally by multiple numerical examples [3, 4].

Algorithm 2: Selection of optimal refinements over K

```
Input: Element $K$, coarse mesh solution $u_{hp} \in V_{hp}$, fine mesh solution
       $u_{\frac{h}{2},p+1} \in V_{\frac{h}{2},p+1}$
Output: Optimal refinement $V^K_{opt}$ for element $K$
1  for coarse mesh elements $K \in T_{hp}$ do
2      for approximation space $V_{opt} \in K$ do
3          $rate_{min} = \infty$
4          Compute the projection based interpolant $w|_K$ of $u_{\frac{h}{2},p+1}|_K$
5          Compute the error decrease rate
           $rate(w) = \frac{\left|u_{\frac{h}{2},p+1} - u_{hp}\right|_{H^1(K)} - \left|u_{\frac{h}{2},p+1} - w\right|_{H^1(K)}}{\Delta \mathrm{nrdof}(V_{hp}, V^K_{opt}, K)}$
6          if $rate(w) < rate_{min}$ then
7              $rate_{min} = rate(w)$
8              Select $V^K_{opt}$ corresponding to $rate_{min}$ as the optimal
                 refinement for element $K$
9          end
10     end
11 end
12 Select orders of approximation on edges as minimum of corresponding
     orders from neighboring interiors
```

Our goal is to replace Algorithm 2 with a deep neural network. The left column in Figure 2 presents the optimal distribution of refinements, as provided by the deterministic algorithm. We can see that all the h refinements (breaking of elements) are performed towards the point singularity. We also see that the p refinements are surrounding the singularity as layers with a different color. They change from red, light and dark pink ($p = 6, 7, 8$), through brown ($p = 5$), yellow ($p = 4$), green ($p = 3$), blue ($p = 2$) and dark blue ($p = 1$) close to the singularity.

The refinements performed by the iterative Algorithm 1 are executed first closer to the singularity. With the iterations, the differences between the coarse and fine mesh solution tend to zero [3].

Dataset We propose the following samples to train the DNN:

Input variables: coarse mesh solution $u_{hp} \in V_{hp}$ for element K, the element sizes and coordinates, the norm of the fine mesh solution over element K, the maximum norm of the fine mesh solution over elements
Output variables: Optimal refinement V^K_{opt} for element K

We construct the dataset by executing the deterministic Algorithm 1 for the model L-shape domain problem. We perform 50 iterations of the hp-adaptivity,

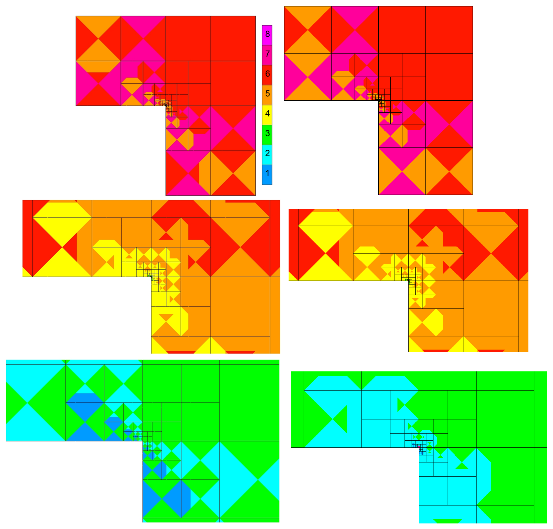


Fig. 2. The mesh provided by the deterministic *hp*-FEM algorithm and by the deep learning-driven *hp*-FEM algorithm. Different colors denote different polynomial orders of approximation on element edges and interiors. The original L-shape domain. Zoom 1x, 1000x, 100000x towards the center. The sequence of *hp* refined meshes generated by deterministic algorithm (left panel) and DNN driven algorithm (right panel).

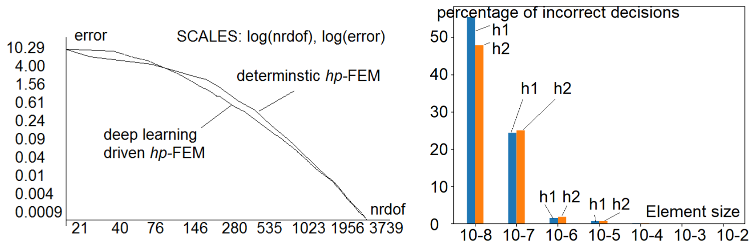


Fig. 3. Let panel: The comparison of deterministic and DNN *hp*-FEM on original L-shape domain. **Right panel:** The sizes (horizontal h1 / vertical h2 directions) from 10^{-2} (right) down to 10^{-8} (left) of the elements where MPL network made incorrect decisions during verification.

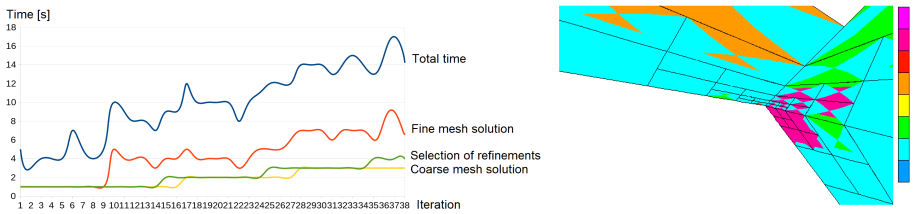


Fig. 4. Left panel: The execution times of the parts of the self-adaptive *hp*-FEM algorithm. **Right panel:** The refinements generated by DNN for a distorted mesh.

generating over 10,000 deterministic element refinements, resulting in 10,000 samples. We repeat this operation for rotated boundary conditions (4) by the following angles: 10, 20, 30, 40, 50, 60, 70, 80, and 90 degrees. Each rotation changes the solution and the samples. We obtain a total of 100,000 samples. We randomly select 90% of the samples for training and use the remaining 10% as a test set. We further sample the training data and use 10% of training data as a validation set. After one-hot encoding the categorical variables, each sample is represented by a 136-dimensional vector. Since it is much more common for the deterministic algorithm to make specific h refinement decisions ($nref$) for the L-shape domain problem, the dataset is imbalanced. To mitigate this, we apply supersampling of underrepresented $nref$ classes.

DNN architecture. We use a feed-forward DNN[1] with 12 fully-connected layers. After 8 layers, the network splits into 6 branches, 4 layers each: the first branch decides about the optimal $nref$ parameter - h refinement, the remaining branches decide about modifying the polynomial orders - p refinement. Experiments have shown that further expanding of the network makes it prone to overfitting[8]. Splitting the network into branches assures sufficient parameter freedom for each variable. This approach also simplifies the model: there is no need to train a DNN for each variable. Since all possible decisions are encoded as categorical variables, we use cross-entropy as the loss function. We encoded the input data as a 136-dimensional normalized vector, as detailed in Table 1. We assume

Feature name	data dimensionality
Polynomial degree	1
Element coordinates	2
H1 norms	2
Polynomial orders (one-hot)	10
Polynomial coefficients	121

Table 1. Dimensionality of specific input features to the DNN, encoded in a single 136-dimensional vector. Polynomial coefficients that do not exist in a given polynomial order are always 0.

that the polynomial degree will not exceed $n = 11$. We train the network for up to 200 epochs with validation loss-based early stopping on an Nvidia Tesla v100 GPGPU with 650 tensor cores available in ACK Cyfronet PROMETHEUS cluster [2]. To minimize the loss function, we use the Adam optimizer [7], with the learning rate set to $10e^{-3}$. We apply kernel L2-penalty throughout the training as a means of regularization and dropout [10] with probability 0.5. The network converges after approximately 110 epochs.

DNN performance. The network achieved over 92% accuracy on the test set. We run three tests to assess whether such a network can be used in the hp-FEM.

First numerical experiment is to reproduce the deterministic Algorithm 2 for the original L-shape domain problem, presented in Figures 1, 2 and left panel in Figure 3. Both deterministic and DNN-driven algorithms provide exponential convergence. The verification phase shows that the DNN makes up to 50% of incorrect decisions when the element sizes go down to 10^{-7} and less, see the right panel in Figure 3. Thus, at the zoom of 100,000 times, we see some differences in Figure 2. Despite that, the algorithm still converges exponentially.

Second numerical experiment. We run the self-adaptive *hp*-FEM algorithm, and we provide zeros as the coarse mesh solution degrees of freedom. We get the same convergence. This second test shows that the DNN is not sensitive with respect to the coarse mesh solution and that the norm of the fine mesh solution, the maximum norm, and the coordinates and dimensions of the elements are enough to make proper decisions. The DNN looks at the fine mesh solution's norms at the given and neighboring elements and, based on these data in decides whether the element is to be broken and how it should be broken. Thus, we can replace Algorithm 2 and the coarse mesh solution phase with the DNN. Left panel in Figure 4 presents the execution times of particular parts of the *hp*-FEM algorithm. The removal of the coarse mesh solution phase and replacing Algorithm 2 by the DNN saves up to 50 percent of the execution times.

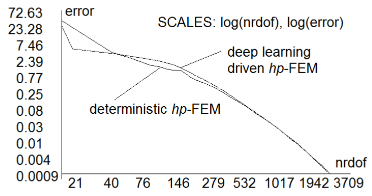
Third numerical experiment. The third test, illustrated in Figure 5, concerns the L-shape domain algorithm with boundary conditions rotated 45 degrees (no samples for this case were provided in the training set). The DNN *hp*-FEM also provides exponential convergence in this case.

Fourth numerical experiment. The last test, illustrated in the right panel in Figure 4 concerns randomly disturbed mesh, different from the training set. The DNN captures both top and bottom singularities. It produces *hp* refinements towards the bottom singularity and *p* refinements towards the top singularity. The resulting accuracy was 1 percent of the relative error after ten iterations.

3 Conclusions

We replaced the algorithm selecting optimal refinements in the self-adaptive *hp*-FEM by a deep neural network. We obtained over 92% of correct answers, the same accuracy of the final mesh, and exponential convergence of the mesh refinement algorithm. A very interesting observation is that DNN requires coordinates of elements (to recognize the adjacency between elements), the dimensions of elements (to recognize the refinement level), the H^1 norm of the solution over the element, and the maximum norm of the solutions over elements. The DNN by "looking" at the norms over adjacent elements, recognizes with 92 percent accuracy the proper *p*-refinement of the element. The replacement of the coarse mesh solver (line 2 in Algorithm 1) and the optimal refinements selection (Algorithm 2) by the DNN allows for a 50 % reduction of the computational time.

The DNN used is available at
home.agh.edu.pl/paszynsk/dnn_hp2d/dnn_hp2d.tar.gz



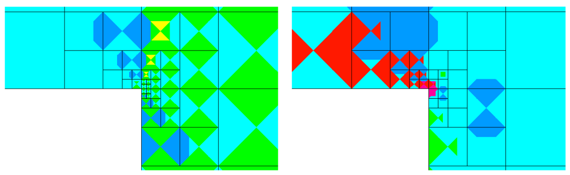


Fig. 5. The convergence for deterministic and DNN hp-FEM algorithms for the L-shape with b.c. rotated by 45 deg. The meshes of the deterministic *hp*-FEM algorithm and by DNN driven hp-FEM algorithm for the L-shape domain with b.c. rotated by 45 deg. Zoom 10^5x times.

Acknowledgement

This work was partially supported by National Science Centre grant no. 2016/21/B/ST6/01539 and 2019/35/O/ST6/ 00571. The visit of Maciej Paszyński at the Odens Institute has been supported by J. T. Oden Research Faculty Fellowship.

References

1. Bebis, G., Georgiopoulos, M.: Feed-forward neural networks. IEEE Potentials **13**(4), 27–31 (1994)
2. Bubak, M., Kitowski, J., Wiatr, K.: eScience on Distributed Computing Infrastructure: Achievements of PLGrid Plus Domain-Specific Services and Tools, vol. 8500. Springer (2014)
3. Demkowicz, L.: Computing with hp-Adaptive Finite Elements, vol. 1. Chapman & Hall / CRC Applied Mathematics & Non-linear Science (2006)
4. Demkowicz, L., Kurtz, J., Pardo, D., Paszyński, M., Rachowicz, W., Zdunek, A.: Computing with hp-Adaptive Finite Elements., vol. 2. Chapman & Hall / CRC Applied Mathematics & Non-linear Science (2007)
5. Guo, B., Babuška, I.: The hp version of the finite element method, part i: The basic approximation results. Computational Mechanics **1**(1), 21–41 (1986)
6. Guo, B., Babuška, I.: The hp version of the finite element method, part ii: General results and applications. Computational Mechanics **1**(1), 203–220 (1986)
7. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
8. Salman, S., Liu, X.: Overfitting mechanism and avoidance in deep neural networks. arXiv preprint arXiv:1901.06566 (2019)
9. Schwab, C.: p-and hp-finite element methods. The Clarendon Press, Oxford University Press, New York (1998)
10. Srivastava, N.: Improving neural networks with dropout. University of Toronto **182**(566), 7 (2013)