A Neural Network for Fluid Structure Interaction

January 11, 2022

1 Description

This document describes some results for the fluid structure interaction of a neural network. This work is a part of a class project for APMA 922 with Mahdi Salehzadeh. This section covers a brief description of the problem, and the next section gives some results for the neural network. The entire project is not included in this report and some sections are missing. The results here are part of my contributions to the project.

Fluid structure interaction (FSI) problems are a coupling between a structure and the fluid it is immersed in. These problems are found in a wide range of applications including the modelling of blood flow and heart valves [11], muscle mechanics [4], as well as other areas of biomechanics (eg. jellyfish [6] or squid [5]). One of the first numerical methods to solve the FSI problems were developed by Peskin [11]. These are the immersed boundary (IB) methods, and the techniques developed in these works [11] are still used in the current IB solvers. The IB method is an elegant way to solve the fully coupled motion of a fluid and an immersed elastic structure.

There is an open source code IB2d [2, 3, 4] which uses a finite difference method to solve IB problems. This code has been developed with a variety of test cases built into the existing code including a rubber-band, beam, and muscle mechanics [2]. All of these methods are based on the same formulation of the FSI problem as the incompressible Navier-Stokes equation coupled with a given structure through an additional forcing term in the Navier-Stokes equation.

$$\rho\frac{D\mathbf{u}(\mathbf{x},t)}{Dt} = -\nabla p(\mathbf{x},t) + \mu\nabla^2\mathbf{u}(x,t) + \mathbf{f}(\mathbf{x},t), \qquad \nabla\cdot\mathbf{u}(\mathbf{x},t) = 0, \tag{1}$$

where ρ is the fluid density, $p(\mathbf{x},t)$ is the fluid pressure, μ is the dynamic viscosity of the fluid, $\mathbf{u}(\mathbf{x},t)$ is the fluid velocity, $\mathbf{f}(\mathbf{x},t)$ is the Eulerian force density acting on fluid. Here $\frac{D\mathbf{u}}{Dt} = \frac{\partial\mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}$ is the material derivative. The force density is given by

$$\mathbf{f}(\mathbf{x},t) = \int_\Gamma \mathbf{F}(q,t)\delta(\mathbf{x}-\mathbf{X}(q,t))dq$$

and

$$\mathbf{U}(q,t) = \frac{\partial\mathbf{X}}{\partial t} = \int_\Omega \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x}-\mathbf{X}(q,t))d\mathbf{x},$$

where $\delta(x)$ is a 2D delta function kernel, $\mathbf{X}(q,t)$ is the mapping of the Lagrangian material point q to the Cartesian grid, and $\mathbf{U}(q,t)$ is the velocity of the Lagrangian material coordinate.

Another method to solve these immersed boundary methods would be through a deep learning approach. These approaches are a more recent advancement in the solution of partial differential equations (PDEs) (see eg. [13, 9]). These methods work to fit a given neural network (NN) (see Section 2) to a set of data for the PDE by minimizing a given loss function. The formulation of the loss function will depend on the problem at hand, in its simplest form the loss function could look to minimize the difference between some unknowns and the given data, eg. the velocities and pressures in the Navier-Stokes equation. Another additional term in the loss function that is also typically used [9, 13] is the requirement that the PDE also be satisfied. When the NN is required to enforce this condition, this is sometimes called a *physics informed neural network* [13]. One benefit of utilizing these methods are that there are no mesh requirements, which was noted above to be a drawback in the IB2d method (the computational time gets large for refined meshes). Another benefit is that this method can be extremely fast ($< 1s$) once the NN has been trained, although training the NN can take very long (eg. > 12 hours for complex problems) to achieve a very accurate NN. There are some other downsides to the NN approach as well. This is relatively new approach to solving PDEs compared to the finite difference approach, and so there is relatively little theory for these methods. In particular, this means there is no theory available for deriving a predicted error in the NN or the ability of a NN to converge to the exact solution. In contrast, there is a large amount of theory available for finite difference methods including convergence and guaranteed errors bounds. Nevertheless, this method of solution to PDEs has been a very popular area of research as of late, and will be investigated in this project as an alternative to the IB2d method. To do this, we will utilize existing code from two sources. The first is Raissi et al. [13], where there is existing code for the Navier-Stokes equation in 2d. The other source is Balsam et al. [1], where there is existing code for an immersed boundary problem for elliptic equations. These two codes can be combined to generate a NN code to solve the FSI problem.

The test problem that we will use is the elastic band immersed in a fluid, because it is a simple example to work with (it is one of the examples from [4]); however, it still produces an interesting interaction between the band and the surrounding fluid. Periodic boundary conditions are used and the initial condition will be the rubberband pulled to an ellipsoid. For more details on the test problem see [4].

2 Neural network

The deep neural network approach will involve modifying existing code developed in [1] and [13] to the Navier-Stokes equation with an immersed boundary.

We can define a function f_{NS} given by

$$f_{NS}(\mathbf{x}, t) = \rho \frac{D\mathbf{u}(\mathbf{x}, t)}{Dt} + \nabla p(\mathbf{x}, t) - \mu \nabla^2 \mathbf{u}(\mathbf{x}, t).$$

The goal is then to approximate the solution to Equation 1. This is done by determining the L layer neural network, $g : \mathbb{R}^k \to \mathbb{R}^l$, given by

$$\mathbf{g} = \sigma^{[L]} \circ \mathbf{A}^{[L]} \circ \sigma^{[L-1]} \circ \mathbf{A}^{[L-1]} \circ \cdots \sigma^{[0]} \circ \mathbf{A}^{[0]},$$

where $\mathbf{g}$ will act on some given initial and boundary data to predict the solution at a later time. Here the $\sigma^{[i]}$ correspond to the activation functions on the ith layer, which will be the hyperbolic tangent function. $\mathbf{A}^i$ will be that affine transformation on the ith layer. The exact form of the affine map will at each layer by minimizing a loss function given some training data. In particular the loss function for the immersed boundary method will be a mean square error function given by

$$L(x, t) = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(x_u^i, t_u^i) - u^i|^2 + \frac{1}{N_f} \sum_{i=1}^{N_f} |f_{NS}(x_f^i, t_f^i) - f(x_f^i, t_f^i)|^2, \qquad (2)$$

where $\{x_u^i, t_u^i, u_u^i\}_{i=1}^{N_u}$ are the training points given on the boundary and at initial data and $\{x_f^i, t_f^i\}_{i=1}^{N_f}$ are the training points throughout the domain. The training points for the neural network solutions will be generated using IB2D on a refined mesh.

3 Numerical Results

3.1 Neural Network

To test the behaviour of the NN on the FSI problem, data was obtained using the finite difference code as described above. However, for simplicity, the density, ρ, was set to 1, and the dynamic viscosity, μ, was set to 0.1. The data for all the tests in this section were constructed using a 64×64 mesh and $dt = 0.001$ up to a time of $t = 0.1$, with data taken every 20 timesteps. This was done so that there was not a large number of time points to train the NN. This keeps the training time at reasonable level (≈ 1 hour). The NN works by optimizing the parameters as described in Section 2 using two optimizers; first, a Adam optimizer [8] and, secondly, an L-BFGS-B optimizer [10]. The stopping criterion for these two optimizers for convergence was set to be machine epsilon. Additionally, the maximum number of iterations for each optimizer was varied depending on the test (as will be seen below). It should be noted as well that this method uses a random weight and bias initialization, a random choice of training data, and stochastic gradient method (Adams optimizer) to minimize the loss function. Therefore, the results will not be the same for every run. However, after running a a few sample tests, we find there is not much difference in the final results for different runs.

3.1.1 Determining the hyperparameters

One difficulty in using a physics informed neural network to solve PDE's is choosing the correct hyperparameters. The hyperparameters in the neural network are the parameters that are not determined through the training of the network. This includes the depth, width, initializations, etc. These parameters are likely to have an influence on the speed of the convergence of the neural network. This neural network is adapted from the code by [13], which was designed to solve the Navier-Stokes equations in 2D with 50000 L-BFGS-B iterations and 200000 Adams iterations. This code took at least 2-3 hours to run and only got to an loss on the order of 1×10^{-2} (when running this code, we terminated well before the maximum number of iterations was reached). Running for the full amount of time to fully decrease the loss would take many more hours. Therefore, to determine the hyperparameters in this project, the maximum number of iterations will be set to 4000 for the Adams optimizer and 5000 for the L-BFGS-B optimizer. This results in a run time of a 20 minutes to an hour, depending on the exact width and depth of the neural network. This will allow for a determination of the best neural network parameters, and then the number of iterations can be increased.

First, the width and depth of the neural network can be investigated. To do this, we will use a *grid search* method, which involves keeping one parameter fixed, eg. the width, while varying the length, and then keeping the length fixed while varying the width. This results from this procedure are shown in Table 1, for varying the width at fixed length, and Table 2, for varying the depth at fixed width. We see here that the errors continue to decrease for larger widths at the expense of longer computational time. Since we see the largest change in the loss for the 20 to 40 change in width (Table 1), we can fix the width now to be 40. The reason that we do not choose a wider NN is that this would result in longer computational times, but not much change in the loss of the NN. We then can vary the lengths; however, we see here that increasing the NN depth only resulted in a decrease in loss up to a depth of 8. For depths greater than 8, there was an increase in the training time. Therefore, we can fix the depth of the NN to be 8. This procedure resulted the best width and length being 40 and 8, respectively. It seems for the given data and loss function that wider networks appear to work better than deeper ones.

3.1.2 Results for optimal hyper parameters

Here we outline the results of our tests for the optimal hyperparameters. In order to investigate the L^2 error and the run time of the neural networks, we increased the number of iterations of the optimizers so a smaller loss could be achieved (10000 for the Adams optimizer and 10000 for the L-BFGS optimizer). Here we should expect to see a smaller loss and the errors should decrease. However, even after increasing the number of iterations, we see that the loss only decreases to a value of 4.539×10^{-1}, which is not much difference than the result from the previous tests (see Table 1 and 2). These results took 3624.7343s

Width	Loss	Training time(s)
20	1.285×10^{1}	385.86437
40	2.946	1031.90429
60	1.370	1750.308311
80	9.539×10^{-1}	2204.3696
100	6.075×10^{-1}	3004.864877

Table 1: Neural network losses and training time for 4000 Adams optimizer iteration and 5000 L-BFGS-B optimization iterations. The depth of the neural network was fixed at 4 hidden layers, while the width was varied.

Depth	Loss	Training time(s)
4	2.946	1031.90429
8	1.119	2128.8550
12	1.889	3023.9176

Table 2: Neural network and training time for 4000 Adams optimizer iteration and 5000 L-BFGS-B optimization iterations. The width of the neural network was fixed at **40** hidden layers, while the depth was varied.

for the NN training and 1.99833s test the NN. The errors are shown in Table 4 for the resulting testing error. While the training of the NN took a long time to run, running the NN with the learned parameters is very quick. The error was so large in these simulations that the resulting plots were not comparable to the exact solutions from the finite difference methods, so they were not included. One thing that should also be noted is that there was very little change in the loss at larger iterations. This could be due to the fact that the gradients are getting very small or there is a bug in the code; however, we were not able to find any errors in the code.

3.1.3 Computational time of the neural network

As can be seen in the previous sections (eg. Table 1 and 2), the neural network training time has a dependence on the size of the network (width and depth). The wider and deeper the NN, the more variables, and so the optimization procedure will take more and longer iterations. To further investigate the run time of the NN, we can look at the effects of the amount of training data. An increase in the amount of training data makes the problem more difficult to solve since the NN needs to be fit to more points; however, the benefit of increasing the amount of training data is an improved accuracy of the NN.

To investigate the effect of the number of training points, we ran simulations for the optimal parameters determined above (NN width of 80 and depth of 8) of 10000 iterations of the Adams and L-BFGS-B optimizers. We see that there is a decrease in the training time for a decrease in the number of training data as expected (Table 3). One benefit observed in decreasing the number of training

Number of training data	Loss	Training time	Testing time
1000	0.341787	9823.830113	6.004370
500	0.169702	5644.99634	2.0874991
250	0.161730	4067.65595	6.45895
125	0.073587	2890.32950	9.5273988

Table 3: Results for varying the number of training data for fixed NN width and depth of 10 and 8, respectively. Each test was run for 10000 iterations of both the Adam's and L-BFGS-B optimizers.

Variable	L^2 error
u	1.414249
v	1.556452
p	1.016281

Table 4: L^2 errors for the optimal neural network. The loss value at the final iteration for this run was 4.539×10^{-1}.

data was that the loss was able to reach a smaller value (Table 3). Unfortunately, the errors observed in the velocities were still too large to compare with each other (errors were on the order of 10^0). It is likely that if it was feasible to run the test for a longer period of time, then we would see that the errors would be lower for less data points if the loss was lower.

3.2 Comparison of the numerical methods

We have looked at two methods to solve the immersed boundary problem on the elastic band test case. The tests for the finite difference method have shown a second order convergence in both space and time in the magnitudes and velocities and were able to obtain very accurate solutions. Meanwhile the NN was unable to produce accurate approximations to the numerical data. Raissi et al. [13] note that these physics informed neural networks are able to solve PDE's based on any physical laws. In their work they develop a code for the Navier-Stokes code that we have adapted in our study. Given the development in Raissi [13] we should be able to generate a neural network that will be able solve the FSI problem, since this only involves the adding a forcing term to the equations. We found in our project that the run times were significantly longer for the NN compared to the finite difference method. The simulations using the NN took approximately 10^3s, while the FD method took on the order of 10^2s. This is a significant downside of the NN, as it was also not able to able to achieve the same level of error in this given time frame. It should be noted that if the NN parameters could be learned fast, then the actual time that the NN took to get the value of the solution at a time t was much faster (< 10s).

Theoretically, the NN should have a number of benefits over the classical methods such as finite difference methods. For example, one main factor that should be beneficial for the FSI problem is the lack of mesh requirements. While we did not test this explicitly in our project, we might expect that the method is easier to implement (and possible performs better) since we do not have to construct two meshes as in the IB method [12]. One aspect of the data driven approach is the requirement of data. This can be either a benefit or a downside depending on the specific problem. For the problem implemented in this study we used data obtained using the finite difference method. In this way, we need to use both of the methods, IB and NN, to use a data driven approach to solve the IB problem. However, in some physical problems there may be data available, and so this can make implementation easier and possibly more so that the finite difference methods were you may need to interpolate data onto a mesh to compute the solution.

References

[1] Reymundo Itzá Balam, Francisco Hernandez-Lopez, Joel Trejo-Sánchez, Miguel Uh Zapata. An immersed boundary neural network for solving elliptic equations with singular forces on arbitrary domains[J]. Mathematical Biosciences and Engineering, 2021, 18(1): 22-56. doi: 10.3934/mbe.2021002

[2] Battista, Strickland, W. C., and Miller, L. A. (2017). IB2d: a Python and MATLAB implementation of the immersed boundary method. Bioinspiration & Biomimetics, 12(3), 036003–036003. https://doi.org/10.1088/1748-3190/aa5e08

[3] Battista, Strickland, W. C., Barrett, A., and Miller, L. A. (2018). IB2d Reloaded: A more powerful Python and MATLAB implementation of the immersed boundary method. Mathematical Methods in the Applied Sciences, 41(18), 8455–8480. https://doi.org/10.1002/mma.4708

[4] Nicholas A. Battista, Austin J. Baird, Laura A. Miller, A Mathematical Model and MATLAB Code for Muscle–Fluid–Structure Simulations, Integrative and Comparative Biology, Volume 55, Issue 5, November 2015, Pages 901–911

[5] Bi, and Zhu, Q. (2019). Dynamics of a squid-inspired swimmer in free swimming. Bioinspiration & Biomimetics, 15(1), 016005–016005. https://doi.org/10.1088/1748-3190/ab57e4

[6] Dawoodian, and Sau, A. (2021). Kinetics and prey capture by a paddling jellyfish: three-dimensional simulation and Lagrangian coherent structure analysis. Journal of Fluid Mechanics, 912. https://doi.org/10.1017/jfm.2020.1069

[7] IB2d Written by Nicholas A. Battista, Ph.D. Department of Mathematics & Statistics, College of New Jersey Source : url github.com/nickabattista/IB2d

[8] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, 2015.

[9] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (1998) 987–1000.

[10] D.C. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, Math. Program. 45 (1989) 503–528.

[11] Peskin CS, McQueen DM. 1996. Fluid dynamics of the heart and its valves. In: Adler FR, Lewis MA, Dalton JC, Othmer HG, editors. Case studies in mathematical modeling—ecology, physiology, and cell biology. New Jersey: Prentice Hall. p. 309–37.

[12] C. S. Peskin, The immersed boundary method, Acta Numerica 11 (2002) 479–517.

[13] M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learn-ing framework for solving forward and inverse problems involving nonlinear partial differential equations.JournalofComputationalPhysics, 378:686–707, 2019. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2018.10.045. URL-https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[14] Richter. (2017). Fluid-structure Interactions Models, Analysis and Finite Elements / by Thomas Richter. (1st ed. 2017.). Springer International Publishing: Imprint: Springer.