CS230

# Learning Euler Equation Discontinuities with a Fourier Neural Operator

**Taylor R. Brodine**
Department of Aeronautics & Astronautics
Stanford University
tbrodine@stanford.edu

**Ian M. Hokaj**
Department of Aeronautics & Astroautics
Stanford University
ianhokaj@stanford.edu

## 1 Introduction

Solving partial differential equations (PDEs) numerically is computationally expensive and necessitates using very fine meshes to achieve accurate results. Alternatively, using deep learning to learn solution operators for families of PDE's that can map the solution on a fine grid is a faster alternative to traditional PDE solvers and maintains a relatively high accuracy. We are implementing the newly introduced Fourier Neural Operator (FNO) to solve the Euler Equations (a family of Partial Differential Equations in fluid mechanics) using a deep neural network. The FNO has previously been employed to learn solution operators for families of fluid PDE's in [3]. Notably, these explorations all revolve around a single PDE governing a single scalar variable's evolution through space and time. The Euler Equations consist of three coupled PDE's (governing mass, momentum, and energy conservation) with spatial and temporal dependence. This poses a challenge, as it increases the dimension of the input and output data and adds complexity through interdependence of the three variables. For context, the conservation form of the Euler equations in one dimension is given below. Parameters $\rho, u, P$ and $E$ represent density, velocity, pressure, and extensive energy.

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho u\\ E\end{pmatrix}+\frac{\partial}{\partial x}\begin{pmatrix}\rho u\\ \rho u^2+P\\ (E+P)u\end{pmatrix}=0 \tag{1}$$

In this project we solve the Euler equations applied to a Riemann Problem, which is a 1D initial value problem with piecewise-constant initial conditions with a discontinuity at the center. The Riemann Problem is very common for examining the Euler Equations, as discontinuities (shocks) and expansion waves propagate from the initial discontinuity. We thus expand our investigation by attempting to capture shock conditions and propagation in our operator learning. This pushes the limits of the FNO to learn the discontinuous and highly non-intuitive evolution of the target functions. We adapt the FNO architecture to capture the inter-dependencies between the three equations, and experiment with network modifications and alternative loss functions to improve shock-capturing ability.

## 2 Related Work

The neural network developed in this project builds upon the FNO framework laid out in *Li* [3], using the architecture depicted in Figure 1.

FNOs are mathematically motivated by the notion of mapping between the input and solution spaces of PDEs. Take an operator $\mathcal{G} : \mathcal{A} \to \mathcal{U}$ that performs the nonlinear mapping between two spaces: the input space $\mathcal{A}$ (which are the boundary/initial conditions required to parametrize the PDE of interest), and the solution space $\mathcal{U}$ (the solutions resulting from constraining the PDE with $\mathcal{A}$). In the same manner that a deep neural network learns a nonlinear function mapping between input/output data values $(x, y)$, the FNO aims to learn the nonlinear operator mapping between input/solution function data $(a(x), u(x))$. Continuing this analogy, where a vanilla neural network multiplies weights and adds biases to incoming activations at a given layer, the FNO performs similar operations over functions. Weight

CS230: Deep Learning, Fall 2022, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

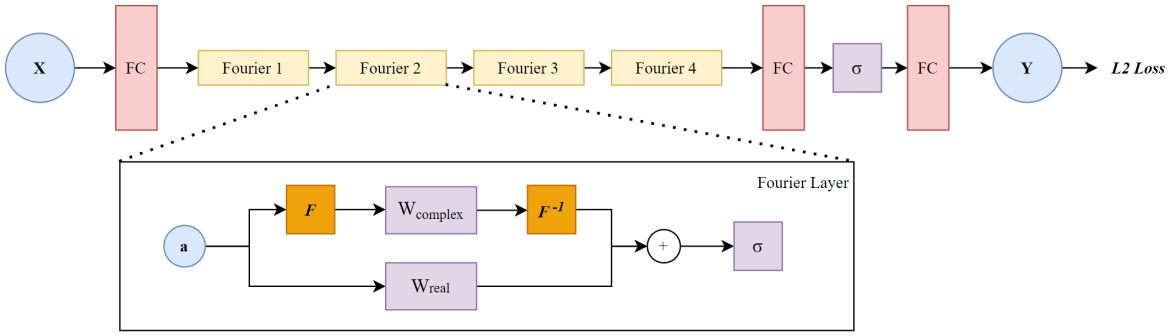



Figure 1: Original FNO architecture

multiplication is replaced by convolution, so the Fourier Transform of the input function activation $(a(x))$ is taken, then the dominant modes are weighted in the frequency domain (equivalent to convolusion in time-domain), after which an inverse Fourier transform is applied. In parallel, the bias is represented by a scalar weighting over the entire input function in the time domain.

Figure 1 shows the deep network achieving of this concept. The input passes through a fully-connected layer to increase the channels of the dataset, creating a mixture of spatial coordinates and the associated function evaluations at the locations. Then, the Fourier layers are applied element-wise on the channels, weighting the dominant spectral modes in the frequency domain and adding scalar multiplicative bias. In the diagram, $F$ and $F^{-1}$ denote these Fourier and inverse Fourier transforms. A Gaussian Error Linear Unit (GELU) nonlinear activation function is then applied via $\sigma$, and the process is repeated over $N$ sequential *Fourier Layers*. The fully connected layers at the end decode the activations back to the desired channel dimension of the solution, at which point the loss (L2 in [3]) is computed.

After constructing data generation scripts, developing pre-processing framework, and performing necessary adaptations for learning system of PDEs, we used the PyTorch FNO implementation from [3] for early testing and to generate one of the baselines. Thereafter, we adapted the code and architecture to better capture discontinuities as laid out in Section 4.

## 3 Dataset and Features

The input to our model consists of a discretized function represented by a vector of initial condition values at each grid point, with the outputs consiting of the vector of solution values at the same points at some specified later time. In the case of the Euler Equations, each grid point "value" is actually the three primitive (independent) quantities: density, velocity, and pressure. Thus, for a single training example in 1D, a input or output vector is of size `(grid_size,3)`.

In order to generate the inputs and outputs for our data set, we implemented MATLAB script that generates random (piecewise-constant) initial conditions for density, velocity, and Pressure for a Riemann Problem (also known as a shock tube). We elected to use this problem because the solution to Euler equations applied to the Riemann Problem can be solved for analytically in 1D by leveraging self-similarity. The random initial conditions are solved on various grid sizes using the exact solution to the Euler Equations following the method presented in [2], thus providing the reference solutions for training the FNO. Training/test sets contain 1000/100 samples, with each sample consisting of the functions $(\rho, u, p)$ at time $t = 0$ seconds and $t = 0.01$ seconds (for input and solution data, respectively), discretized on a 1D grid of resolution 8192 in $\mathbb{R} \in [-10, 10]$ meters. Generating the 1100 samples with the above resolution parameters takes about 60 seconds on one processor core, storing close to 1 gigabyte of data.

A single data example, including the initial condition and exact solution, can be seen in section 5.

## 4 Method

The data are fed into the adapted 1D FNO script from [3], modified to take three function inputs and outputs (rather than one) such that the learned operator maps all three functions to their time-propagated quantities according to the *system* of governing PDEs, rather than a single governing PDE. This minor modification to the implementation in [3] is intended to serve as a baseline for comparison with our future FNO architecture adaptations, and is handled by upping the input channel depth of the first layer from 2 (being $[x, f(x)]$) to 4 (being $[x, \rho(x), u(x), p(x)]$).

### 4.1 Normalization

In [3], the input data (being a scalar quantity) was standardized to have zero mean and unity variance over the domain. Since the solution to the Euler equations, being density, velocity, and pressure, differ in several orders of magnitude (and pressure and density must remain strictly positive to remain physical), we implemented a range normalization such that all three input vectors of $X$ vary between zero and one. The output, $Y$, is unnormalized using the inverse scaling by which the input was normalized (the upper- and lower-bounds for each of the flow parameters).

### 4.2 Choice of Loss Function

In initial testing of the baseline, the general shape of the solutions was tracked, but the solutions perform a smooth transition at the discontinuity (rather than a barrier function-like jump). We attribute this to the L2's indifference of pushing the relatively close-fitted discontinuity to of even smaller magnitude. To improve this, L1, L2, and Sobolev loss functions were tested. Sobolev Loss takes into account the error in derivatives of the predicted versus true solution via adding a weighted 2-norm term to a standard L2 loss function, as is seen in equation 2 [1].

$$L_{Sobolev}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{M} ||\hat{y}_i - y_i||_2^2 + \lambda||\frac{\partial \hat{y}_i}{\partial x} - \frac{\partial y_i}{\partial x}||_2^2 \tag{2}$$

A loss function that takes into account the derivative should motivate the solution to be steeper at the these discontinuities by penalizing the gradient deviations. Tuning of the gradient penalty weighting parameter $\lambda$ and associated results are discussed in Section 5.

### 4.3 Architecture Modification

We also modify the network architecture by adding a bias term parameter to the Fourier layers with the hope that adding element-wise constants will help sharpen the corners of the discontinuities and resolve errors that the linear weighting term misses. Furthmore, the first layer for upscaling the activations channels was changed from a fully connected layer to a 1D convolutional layer with kernel size of 5. This minor change reduces the number of trainable parameters on the order of 10,000 and may have a minor regularization effect by convolving a data point with data near it rather than the full domain from full connectivity. The modified model architecture is depicted in Figure 2.

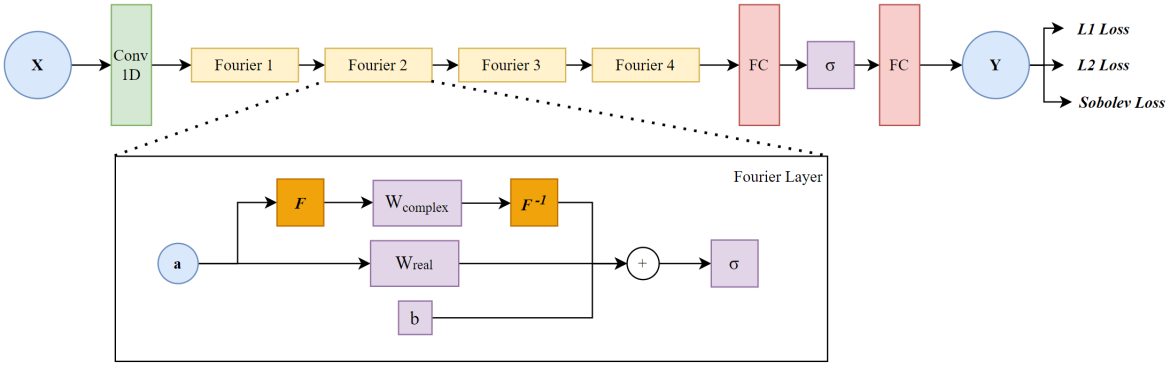



Figure 2: Modified FNO architecture with initial 1D Conv layer, pure bias term, and alternative loss functions

### 4.4 Hyperparameter Tuning

Hyperparameters tuned in the modified neural network consist of learning rate, weight decay, minibatch size, epochs, weight of derivative error in Sobolev Loss (equation 2), number of Fourier modes, and number of channels in the upscaled Fourier layers. All models were trained on an NVIDIA GeForce GTX 1650 Ti GPU.

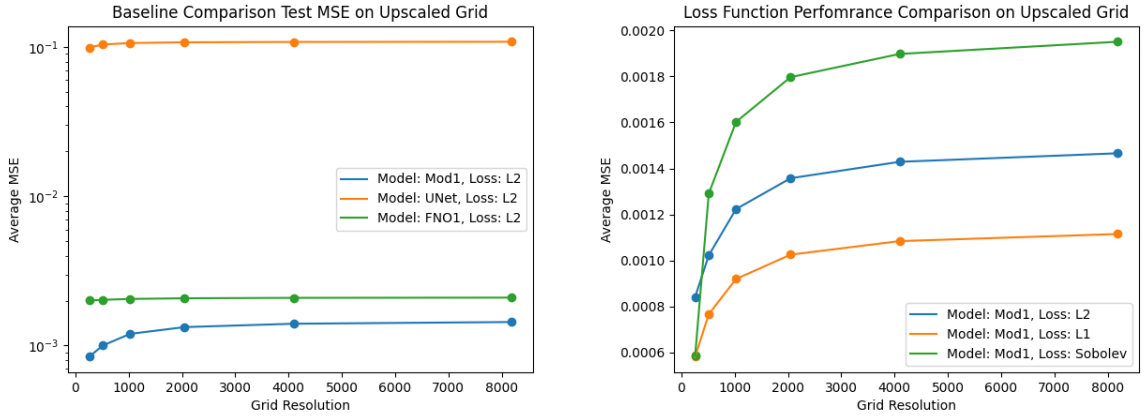
Hyperparameters were initialized based on what was used in [3], then the test MSE versus epochs was assessed from varying each hyperparameter independently. The most impactful hyperparameters were determined to be learning rate, batch size, and derivative error weight in the Sobolev loss. The others had little effect on the training and test accuracy accuracy. Table 1 depicts the test MSE for various learning rate and minibatch size, which guided our selection of a learning rate of 0.005 and minibatch of 20. This hyperparameter tuning was executed on our modified architecture with an L2 loss function. The Sobolev loss error weight was tested independently over $\lambda \in [0.005, 0.01, 0.05, 0.1 and 0.2]$. The weighting $\lambda = 0.01$ was chosen as it yielded the lowest test MSE.

Table 1: Learning rate and batch size tuning, with L2 loss on our modified FNO architecture. Optimal values chosen by lowest Test MSE and reasonable runtime

| Learning Rate | Batch Size | Test MSE | Total / Per Epoch Runtime (sec) |
|---|---|---|---|
| 0.0001 | 20 | 0.002900 | 377.97 / 0.7559 |
| 0.001 | 20 | 0.000912 | 376.77 / 0.7536 |
| **0.005** | **20** | **0.000874** | **389.24 / 0.7785** |
| 0.01 | 20 | 0.000895 | 385.77 / 0.7785 |
| 0.1 | 20 | 0.146408 | 389.69 / 0.7794 |
| 0.005 | 10 | 0.000922 | 757.94 / 1.5159 |
| 0.005 | 30 | 0.000999 | 264.05 / 0.5281 |
| 0.005 | 40 | 0.000912 | 201.13 / 0.4023 |
| 0.005 | 50 | 0.001054 | 166.78 / 0.3336 |

## 5 Results/Discussion

Accuracy at discontinuities are highly dependent on the grid resolution; furthermore, an important figure of merit of the FNO is its ability to predict solutions on a finer resolution grid than the model was trained on (upscaling). Thus, accuracy with respect to the upscaled resolution is an important metric to examine. The test MSE of the modified FNO is compared with the FNO from [3] (adapted for a three dimensional input/output) and a U-Net [4] for various upscaled resolutions in the Figure 3a. The same training hyperparameters enumerated in Section 4.4 were used to train both baselines and the modified FNO, with an L2 loss on all for consistency. All models are coarse grid data obtained bysubsampling the original 8196-resolution dataset. The models are then tested on increasing higher resolution subsamples of the same dataset.




(a) Performance of modified FNO, baseline FNO, and U-Net (b) Performance of L1-, L2-, and Sobolev-norm loss functions

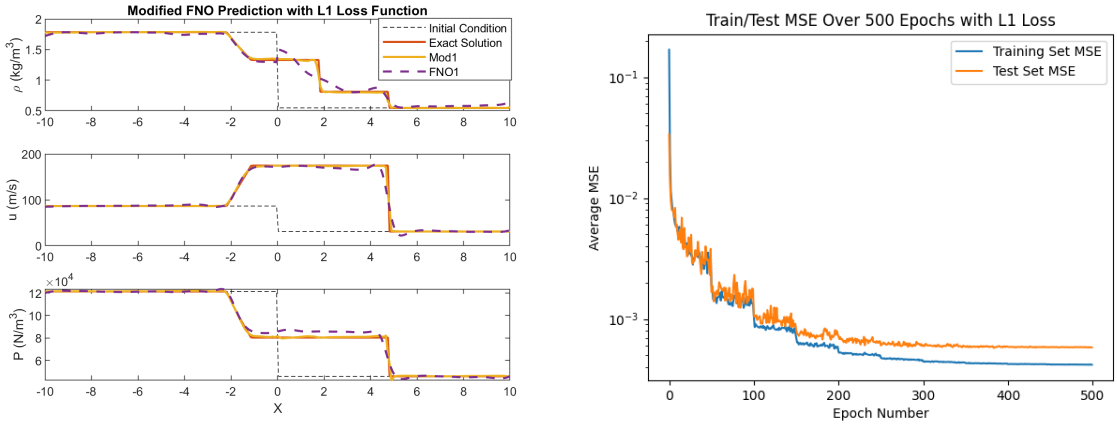
Figure 3: Upscaling performance of models trained on 256-resolution data, evaluated on up to 8196-resolution data

In terms of average MSE on the test set, we observe that the modified FNO (denoted Mod1 in the plots) achieves the lowest MSE on the test set not only at the trained resolution, but also throughout upscaling. Although it shows some performance drop during early upscaling, it outperforms the standard FNO (denoted FNO1 in the plots) by about 10% and the U-Net by far more (which is not unsprising, given a standard CNN seems ill-equipped for this task).

We also include the performance across resolutions for the modified FNO with the L1, L2, and Sobolev (with $\lambda = 0.001$) loss functions in Figure 3b. Using an L1 loss function in the modified FNO yields the highest accuracy across all upscaled resolutions. The Sobolev loss performs similarly well to the L1 at the baseline (training) resolution, but it does not scale well at higher resolutions. We hypothesize that since the derivative is calculated via finite differencing, dividing by small grid spacing values leads to noise in the numerical derivatives. From Figure 3 we therefore conclude that the best candidate for the task is our modified FNO using L1 loss.

To qualitatively assess the FNO's discontinuity-capturing ability, the predicted solution of one example from the test set is shown in Figure 4a for the Modified FNO (with L1 loss function) and the baseline FNO from [3], along with the exact solution and initial conditions. The modified FNO tracks the discontinuities significantly better than the baseline

FNO. The L2 and Sobolev losses also perform well from a qualitative standpoint, but they display slightly more noise (see Appendix). Figure 4b shows the smooth convergence of the L1 modified FNO, with the learning rate scheduler effectively mitigating stalling and a resulting low variance in the test set. Similar plots for the L2 and Sobolev training are included in the Appendix.




(a) (b)

Figure 4

Finally, Table 2 compares the average test MSE for the Modified FNO and the baselines. The modified FNO is significantly more accurate than both baselines and requires similar training time to that of [3].

Table 2: Test MSE comparison with baselines

| Model | Test MSE | Train Time (sec) |
|---|---|---|
| U-Net | 0.0994 | 1024 |
| *Li* FNO | 0.0020 | 340 |
| Modified FNO (L1 Loss) | 0.00058 | 368 |

## 6 Conclusions/Future Work

The novelty in this project lies in the fact that the Euler equations and shock capturing have yet to be attempted using operator learning. This application differs from previous uses of the FNO because the Euler Equations are a system of PDE's rather than a single PDE. Modifications made to the baseline FNO architecture in *Li* [3] include range normalization, bias term parameter, L1 and Sobolev Loss function, and a convolution layer for channel upscaling. These changes led to significant improvement in solution accuracy and ability to resolve discontinuities in the solution. The modified FNO also proved to predict the solution more accurately on resolutions higher than the model was trained on, as compared to the baselines.

## 7 Contributions and Source Code

The project GitHub is here: `https://github.com/ian-hokaj/cs230_project`. Relevant datasets: `https://drive.google.com/drive/folders/13pPSY2vLNwXmbB4IbxtWHWgwSej-lyWS?usp=sharing`

Taylor created the MATLAB data generation code and scripts to pre-process and visualize data. He tuned the hyperparameters and carried out high-resolution testing. Ian adapted the existing FNO architecture to the system-of-PDEs problem setup, implemented the changes to the FNO architecture and losses, and wrote scripts to train and evaluate models. The group worked together to ideate necessary modifications to the architecture and loss function, write the report, and produce the presentation. We would also like to thank our TA Manasi Sharma for her guidance throughout the lifecycle of our project.

## References

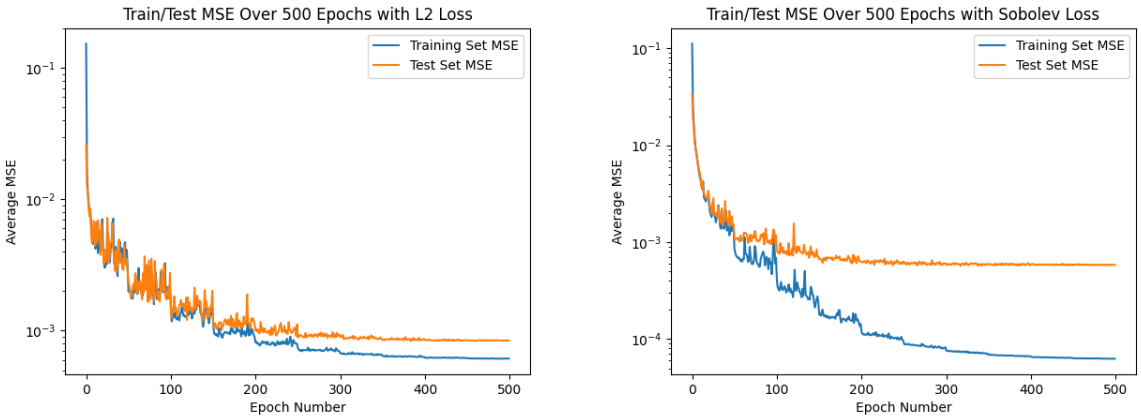

[1] Wojciech Marian Czarnecki et al. "Sobolev Training for Neural Networks". In: *CoRR* abs/1706.04859 (2017). arXiv: 1706.04859. URL: http://arxiv.org/abs/1706.04859.

[2] Charbel Farhat. "Representative Model Problems". In: *AA214: Numerical Methods for Compressible Flows, Course Notes, Ch 5* (2022).

[3] Zongyi Li et al. *Fourier Neural Operator for Parametric Partial Differential Equations*. 2020. DOI: 10.48550/ARXIV.2010.08895. URL: https://arxiv.org/abs/2010.08895.

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer assisted intervention* (2015), pp. 234–241.

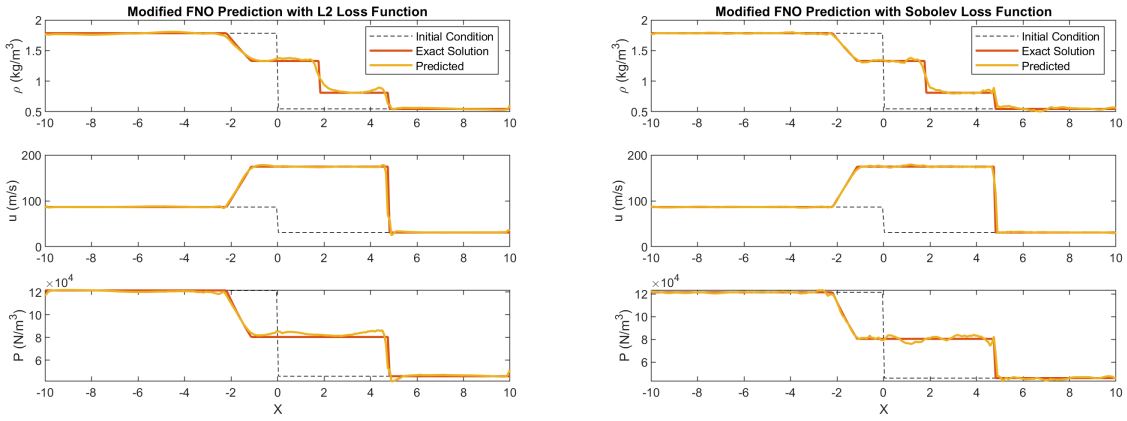

# 8 Appendix




(a) L2 train/test loss

(b) Sobolev train/test loss

Figure 5: Training and test loss at each epoch throughout training with different loss functions




(a) Modified FNO test solution with L2 loss

(b) Modified FNO test solution with Sobolev loss

Figure 6: Learned Euler solutions to test data from FNOs trained on different loss functions