Learning the elastic wave equation with Fourier Neural Operators

Tianze Zhang, Kris Innanen and Daniel Trad

ABSTRACT

Neural operators are extensions of neural networks which in supervised training learn how to map complex relationships, such as classes of PDE. Recent literature reports efforts to develop one type of these, the Fourier Neural Operator (FNO) such that it learns to create relatively general solutions to PDEs such as the Navier-Stokes equation. In this study, we seek what we believe is the first numerical instance of a Fourier Neural Operator (FNO) be trained to learn the elastic wave equation from a synthetic training data set. FNO attempts to find a manifold for elastic wave propagation. On that manifold, wave fields are represented in lower numbers of dimensions than those needed for standard solutions, and the calculations for wave propagation are correspondingly simpler. The FNO combines a linear transform, the Fourier transform, and a non-linear local activation, to produce a network with sufficient freedom to map from a general parameterization of a forward wave problem to its solution. Post-training, the FNO is observed to generate accurate elastic wave fields at approximately 10 times the speed of traditional finite difference methods on a CPU, and about a hundred times faster on a GPU.

INTRODUCTION

Solving the elastic wave equation, a time- or frequency-domain multidimensional partial differential equation for one or more components of a wave field, is an essential part of seismic inversion and imaging. Standard methods like those based on finite differences and finite elements are based on discretization in space, on a grid or mesh. Derivatives are computed at or around each grid or mesh point, and so from a computational point of view, resolution and computational expense form a trade-off. A coarse grid is fast but captures lower-frequency and wavenumber behaviour only; fine grids accommodate higher frequency and wavenumber wave phenomena but are computationally slow and expensive. These costs play a dominant role in the practical formulation (and use of) processes such as seismic waveform inversion, where for computational reasons we are often forced to choose between accuracy / completeness over computational expense and time. The expense of inversion caused by the expense of simulation grows with the maximum frequency desired and the maximum resolution needed, but also depth of investigation and model extent in general, and any other aspect of the problem that increases the length of the data trace being simulated or the number of source points being used. In all modern seismic imaging and inversion problems, including and especially waveform inversion, we are motivated to seek out accurate techniques for solution of elastic wave equations.

Data-driven machine learning methods are very rapidly developing tools for producing maps between very general classes of input and output data. It has become understood recently that in the area of PDEs, learning-based approaches, if properly designed, can be orders of magnitude faster than conventional solvers. Since as recently as 2019, various networks types, often based on very different architectures, have been introduced to solve

complex PDEs, with application across many disciplines. Raissi et al. (2019) proposed the Thermodynamics-based Artificial Neural Network (TANN) for the constitutive modeling of materials with inelastic and complex microstructure. Esmaeilzadeh et al. (2020) developed a novel deep learning-based super-resolution framework to generate continuous (grid-free) spatio-temporal solutions from the low-resolution inputs. Kochkov et al. (2021) use end-to-end deep learning to improve approximations inside computational fluid dynamics for modeling two-dimensional turbulent flows.

A *manifold* in mathematics is a topological space that locally resembles Euclidean space near each point. The term is also used in machine learning, and is conceptually similar but has some of its own features. The basic assumption in what is called manifold learning is that high-dimensional data is accurately representable as low-dimensional data that is embedded in the high dimension space with some level of redundancy. The low dimension space is referred to as the manifold of that data set. For instance, suppose that our task is to given the position of a city on the Earth. In general any position requires three coordinate values, but, knowing that the city is on the surface of the Earth, we could also use the latitude and longitude to represent the same location. The dimension we use to represent the same point is in this way reduced, through an agreement a priori about possible places the city can be in the more general space. The Earth's surface is here playing the approximate role of a manifold on the 3D space. Manifolds can act as a stepping stone from relating a complex space to a simpler, smoother subset space. Many tasks in machine learning are concerned with learning manifold representations for data and then utilizing this representation to make predictions about the remaining space. In the problem of solving an elastic wave PDE in a seismic appoication, our task is to seek a manifold on which to model the propagation of a wave field, whose dimension is significantly lower than the space needed for a standard discretized solution. If this can be done, it may be possible to generate wavefields using fewer and simpler operations.

Li et al. (2020) introduced the Fourier Neural Operator (FNO) as a tool learn a number of the PDE equations, for instance, Burg's equation, the Darcy Flow equation, and the Navier-stokes equation. The structure of the network consists of three parts: two networks to calculate the dimensional reduction and dimensional extension, and a set of Fourier layers, which calculate any requisite spatial derivatives. Simple numerical tests are suggestive that the predictions of the FNO are in principle accurate and about a thousand times faster to create than those of the conventional finite difference solver. The heart of that work is the attempt to find a low-dimensional manifold for the propagation of those fields; in those cases it was possible, and this explains the rapid calculation.

Inspired by this work, we modify Li et al. (2020) and train the modified FNO to learn the elastic wave equation in the time domain. As part of the modification, the FNO is made more suitable for calculating derivatives in multiple directions. We develop and describe this elastic wave FNO in this paper, and discuss the influence of network hyper-parameters on the convergence of training. To our knowledge this is the first discussion of the role of hyper-parameters to appear. After our modification, an elastic FNO network generates elastic wavefields accurately, and at a computational cost about 10 times less than a standard finite difference solver when computed on a CPU, and about a thousand times faster on a GPU.

THEORY

Let $\mathcal{M}^{\dagger}$: $\mathcal{A} \rightarrow \mathcal{W}$ represent a non-linear map, where the $\mathcal{A}$ and $\mathcal{W}$ are subspace of $\mathbb{R}$ taking the real values. If $\{a_j\}_{j=1}^{N} \in \mathcal{A}$ and $\{w_j\}_{j=1}^{N} \in \mathcal{W}$ are N points of realizations in $\mathcal{A}$ and $\mathcal{W}$, the the non-linear mapping between the $\mathcal{A}$ and $\mathcal{W}$ is $w_j = \mathcal{M}^{\dagger}(a_j)$. In this study we regard the $\mathcal{M}^{\dagger}$ as the operator that can solve the elastic partial differential equation, and we aim to build the parametric mapping:

$$\mathcal{M}_{\theta} : \mathcal{A} \rightarrow \mathcal{W}, \theta \in \mathbf{\Theta}, \tag{1}$$

which the operator $\mathcal{M}_{\theta}$ is parameterized with θ, θ belonging to the finite-dimensional space $\mathbf{\Theta}$ such that $M_{\theta} \approx M^{\dagger}$. Suppose that $\mathcal{J}$ is a cost functional $\mathcal{J}(\mathcal{W}, \mathcal{W}) \rightarrow \mathbb{R}$. Thus we are seeking to minimize the problem of:

$$\min_{\theta \in \mathbf{\Theta}} \mathcal{J}[\mathcal{M}(a, \theta), \mathcal{M}^{\dagger}(a)], \tag{2}$$

where $a \in \mathcal{A}$, which representing the measure on space $\mathcal{A}$. Minimizing the cost function and testing the accuracy of the approximation operator can be realized with data driven supervised learning train-testing setting, which is the empirical approximation in machine learning.

The neural operator

According to Li et al. (2020) the full structure of the Fourier Neural Operator has been shown in Figure 1. We made several changes in the network to make the it more suitable for learning the elastic wave equation. The input wave fields are first projected to a compressed dimension with a linear transform, $f_1 = Dp_1\big(u(x, z, t_{in}), m_1, m_2, m_3\big)$, where the Dp_1 a linear operator that takes in the wave fields, coordinates x, y and environments coefficients, m_1, m_2, m_3. The environments coefficients in this case are Vp, Vs and density models, and for other cases, this environments coefficients could be a certain kind of physics properties like the conductivity or viscosity that describe the physical environments in spatial domain. In the Dp1 layer, the dimension projection is achieved with a shallow fully connected neural network parameterized with θ, $Dp_{1_\theta} : \mathbb{R}^{T_{in}, \times d_x \times d_z + 5 \times d_x \times d_z} \rightarrow \mathbb{R}^{d_w \times d_x \times d_z}$. T_{in} is the number of the times steps for the wavefields that we input to the network. d_w is the dimension projection width which is the hyper-parameter of the network. As the data is transformed into a compressed dimension, thus this means the d_w is much smaller than T_{in}. Then dimension projection output f_1 would follow several updates in the Fourier layer (the purple square layer). In Figure 1, we use 5 updates for the Fourier layer. $Dp_{2_\theta} : \mathbb{R}^{d_w \times d_x \times d_z} \rightarrow \mathbb{R}^{T_{out}, \times d_x \times d_z}$, is the second dimension projection layer parameterized with θ. It is also obtained with a fully connected layer that transforms the data in the higher dimension back into the regular time domain.

The update in the Fourier layer update is defined as:

$$f_{i+1} = \sigma\left[W f_i + \mathcal{K}_x(f_i; \phi) + \mathcal{K}_z(f_i; \phi)\right], \tag{3}$$

where $\mathcal{K}_x$ and $\mathcal{K}_z$ are an operation parameterized with $\theta \in \mathbf{\Theta}$ and performs the transform with spatial Fourier transform and element point wise product. The definition of $\mathcal{K}$ as

$$\mathcal{K}_c(f_i; \phi) = \mathcal{F}_c^{-1}\big(R_{\phi_c} \cdot \mathcal{F}_c(f_i)\big), c \in x, y, \tag{4}$$

where the $\mathcal{F}_c$ and $\mathcal{F}_c^{-1}$ are the forward and inverse spatial Fourier transform in coordinate c, $(c \in x, y)$. For instance, the Fourier transform with x direction as an example, if $f_i \in \mathbb{R}^{d_w \times d_x \times d_y}$, then $\mathcal{F}_x(f_i) \in \mathbb{C}^{d_w \times k_x \times d_y}$. In the work presented with Li et al. (2020), the $R_{\phi_x} \in \mathbb{C}^{d_c \times d_w \times k_j \times d_j}$, is the weight tensor that only does the element point wise product with only part of $\mathcal{F}_x(f_i)$,where $|k_j| \leq k_{max}$ and $d_j \leq d_y$.

$$\left(R_{\phi_x} \cdot \mathcal{F}_x(f_i)\right)_{w,k,j} = \sum_{c}^{d_c} (\mathcal{F}_x f_i)_{w,k,j} R_{c,w,k,j}, \tag{5}$$
$$c = 1, \ldots, d_c, w = 1, \ldots, d_w, k = 1, \ldots, k_j, j = 1, \ldots, d_j$$

The same operations are applied on the z coordinate as well. The updating the weight kernel R_{ϕ_c} in operator $\mathcal{K}$ is actually approximating the partial differential calculation. We can understand the operator of $\mathcal{K}$ with the convolution theorem. The convolution theorem shows that the product in the spatial Fourier domain is actually the convolution in the regular spatial domain. Thus, transforming the fields in one spatial direction with the Fourier transform and calculating the product with a kernel is equivalent to performing the convolution with the corresponding kernel on the original spatial domain. The work of Sun et al. (2020), Zhang et al. (2020), Zhang et al. (2021), have demonstrated that the forward modeling of the wave fields could be achieved by the spatial convolution operation with a filter that is designed according to the finite difference stencil for partial differential calculation. $W : \mathbb{R}^{d_x \times d_z} \to \mathbb{R}^{d_x \times d_z}$ is a linear transform which achieved with 2D convolution in the regular space domain, and $\sigma : \mathbb{R} \to \mathbb{R}$ is the non-linear activation function which defined element point-wise.

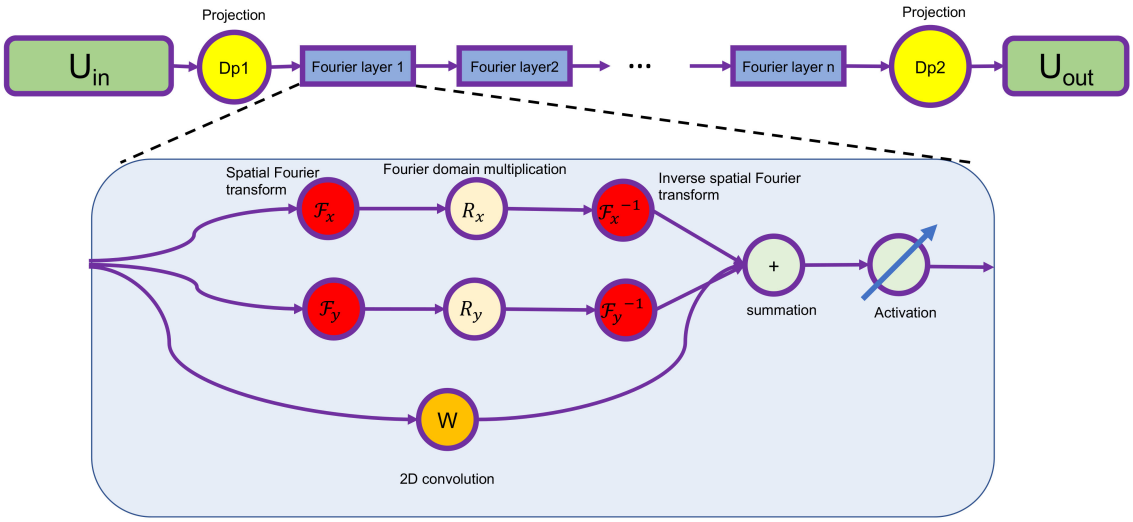

FIG. 1. The structure of the Fourier neural operator, aiming to train a network, which can give the rest steps of the wavefields given by the first steps of the wavefields. U_{in} and U_{out} are the input steps of the fields and the output predictions of the wavefields. Dp_1 and Dp_2 are the dimension projection layer at operates on time axis. In the Fourier layer, $\mathcal{F}_c, c \in x, y$ and $\mathcal{F}_c^{-1}, c \in x, y$ stands the forward and inverse Fourier transform with respect to c direction. $R_c, c \in x, y$ stands for the kernel for multiplication in the Fourier domain. W stands a 2D convolution operation.

According to the previous introduction, we have seen that there are mainly two hyperparameters for the network. The first parameter is the dimension projection width d_w. d_w

determines how much the input layer would be compressed with respect to time. If the $d_w = 1$, such transform is similar to the Fourier transform with respect to time, which changes a series of wavefields propagation time step slices into a single frequency domain slice. The difference, in this case, is that we will always assign a small value for d_w rather than the value of 1. The second hyper-parameter is wavenumbers k_j, which defines the kernel shape to perform the product in the Fourier domain. In the paper of Li et al. (2020), they only use $k_j = 12$ to perform the dot product, which means that only a number of the wavenumbers of k_x would be updated during each operation, and for seismic waves forward modeling, this is somehow not that convincing or reasonable. For instance, in the traditional Pseudo-spectral forward method, the wavenumber coefficients should apply to all the components of the fields after the spatial Fourier transform. The value of the network hyper-parameter d_w and k_j can influence the accuracy of the learning, which will be discussed in the following section.

The effectiveness of the Fourier Neural Operator comes from the combination of the linear operation, operators that resemble the partial differential calculation (via the Fourier transform), and the non-linear local activation. The parameters in such a network are located in the linear transform operator Dp_1 and Dp_2, the Fourier kernels R_c and the convolution filters in W. The operators Dp_1 and Dp_2 stand for the operation with respect to time, and operators $\mathcal{K}$ could represent the operations with respect to space that generate the partial derivatives. The various kinds of partial differential equations can also be seen as a polynomial of the partial derivatives of the space, time with the combination of different physical environments. Thus, the construction of such a network is like that we are building the skeleton of the network for the partial differential equation, and through training and updating the weights in the network, the network would gradually have the ability to learn the physics that is embedding in the data set.

NUMERICAL TESTS

Learning the elastic wave equation

We use the wavefields generated with the stress velocity isotropic elastic wave equation, using the staggered grid finite difference method with 4-orders of accuracy in space as the training data. We generate 100 random models, and the random objects with the shape of rectangle and circle are located in the simulation area. The shape of the rectangle and the diameter of the circle are all randomly generated, and the locations of the objects are also randomly located. Figure 2 shows two of these velocity models in the model data set. The VS and ρ models are obtained through scaling the VP model, with the relationship of $VS = 0.7VP$, and $\rho = 1.74VP^{0.25}$. The sizes of these models are 84×84, and the grid lengths of the models are $dx = dz = 10$.The sources of the wavelet are also randomly located though out the velocity model. We use Ricker's wavelet as our source. When we are generating the training wavefields, the main frequency of the sources are also randomly generated, ranging from $5Hz \sim 15Hz$. Since the wave fields are generated with the finite difference method, we do not tend to use the sources with very high main frequencies to avoid numerical dispersion during the forward modeling. We use 20 layers of the PML boundary condition to absorb the boundary reflections.

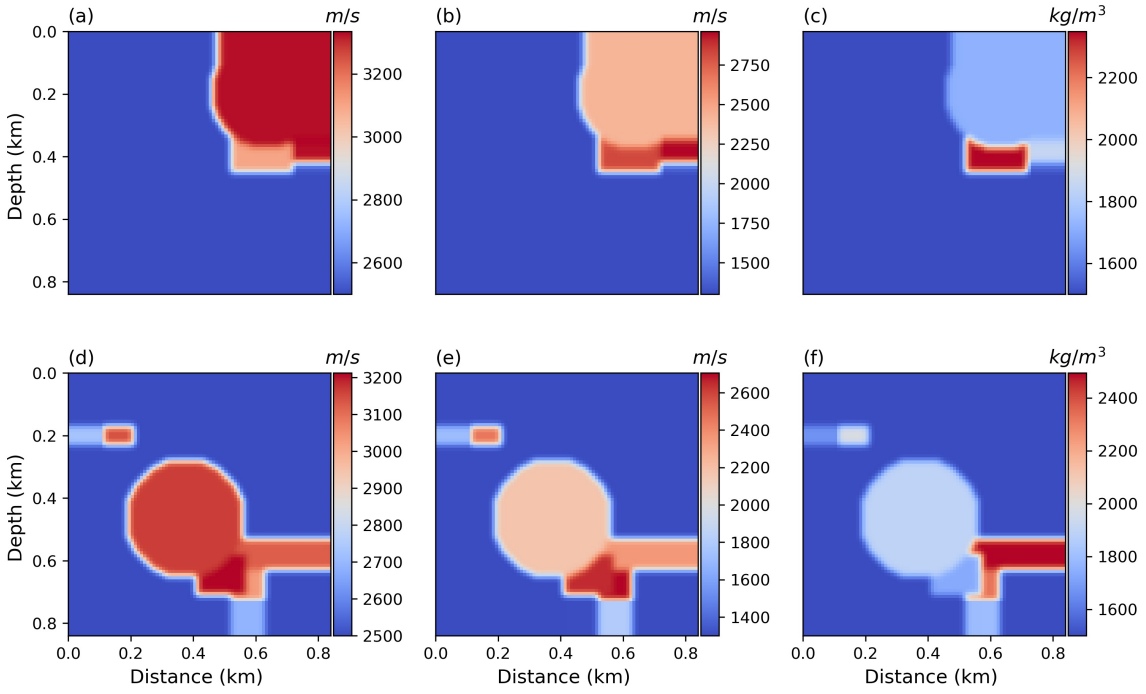

FIG. 2. The random velocity models with random rectangle and circles randomly located inside. The shape of the rectangle and the circles are all randomly located within the models.

Figure 3 shows the nine time slices of the true wavefields for one shot located in the right corner of the model. We will feed the network 50 time steps of the wavefields and train the network to generate the rest 300 time steps for the wavefields. We use 90 models out of the 100 model data set as the training models and the rest 10 models as the validation tests. The maximum epoch is 1500, and the training results are plotted in Figure 4. In this test, we will use 33 Fourier modes, and the dimension projection width is 60. The comparisons show that the FNO could correctly generate the phase position for different components of the wavefields, especially from Figure 4 (a)-(l). The prediction alliances well with the ground truth in Figure 3. However, with the increase of the propagation time, the accuracy of the prediction decreases, especially in Figure (m)-(p). In Figure (m) and (n), though we have the correct phase of the waveforms, their amplitudes are distinct from the ground truth. In Figure (o) and Figure (p), we can also observe the incorrect source distortion influence that does not exist in Figure 3. A proper combination of the dimension projection width and the number of the Fourier model could help to release the problem. The Dp layers are related to the dimension projection width and are responsible for dimension reduction in time. The very small value of the Dp width represents a strong effort for dimension reduction, which may cause the network harder to converge.

Figure 6 shows the influence of the Dp width on the convergence property of the network. In Figure 6, we show the training with Dp width 10, 20, 40 and 60. Figure 5 shows that as the increase of the Dp width, the training convergence shows better convergence property. Also, the convergence line for Dp width 60, showing more fluctuation compared with the other lines. This can be due to the over-fitting of the network. Figure 5 (b) shows the validation test loss. We can see that the validation loss of Dp width 60 has slow con-

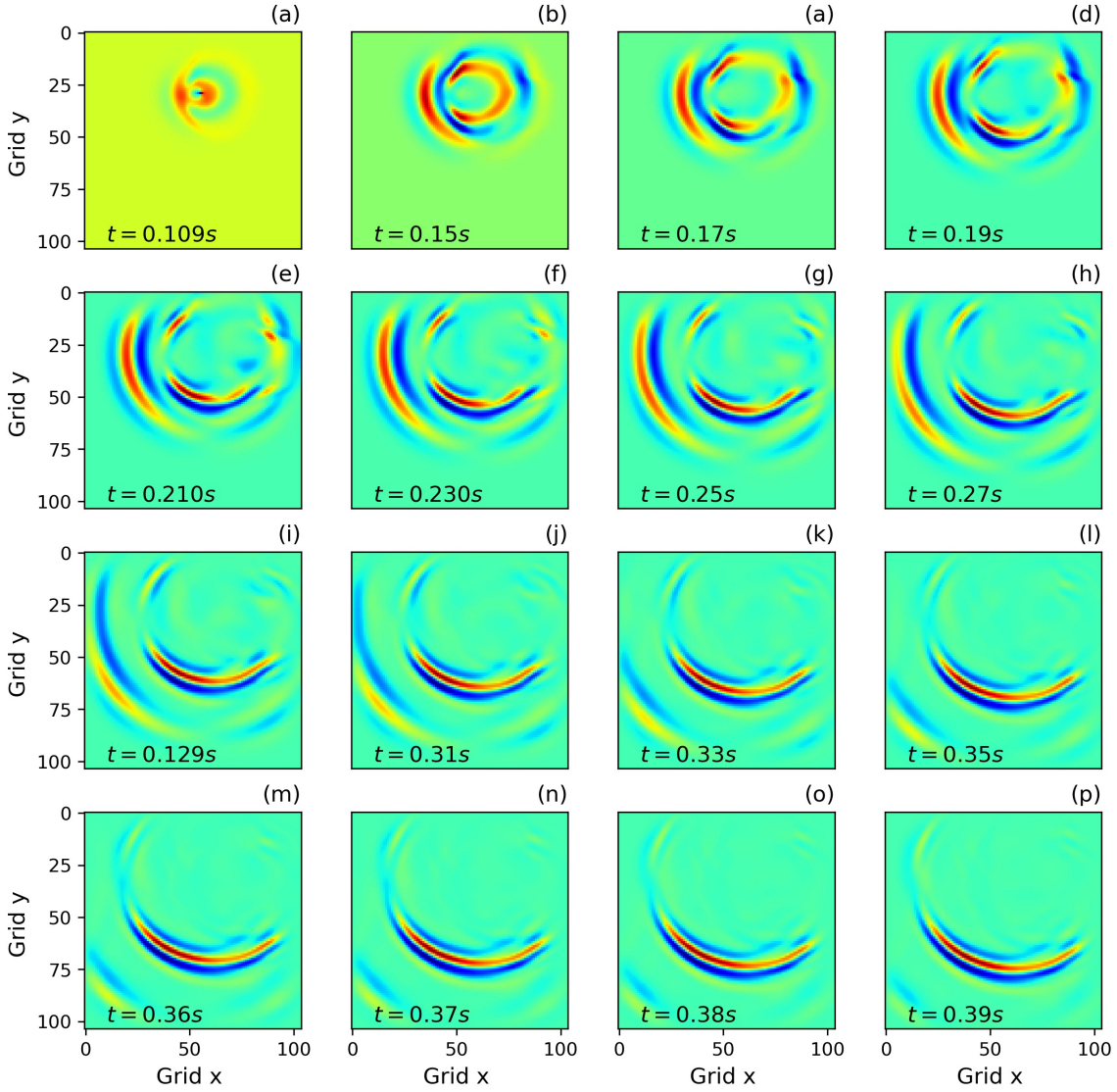

FIG. 3. Nine snap shots of the x component of the wavefields generated with staggered grid finite difference method of on random model, regarded as the ground truth for training.

vergence compared with other lines. Thus, there should be over-fitting in the Dp width 60 test. Figure 6 shows the predictions of the FNO at different time steps for a shot. In Figure 6, in columns, from left to right, are the predictions given by Dp width 10, 20 ,40 and 60. At $t = 0.15s$, we can see all the tests gives the right phase of the wavefields, though with slight blurs in Figure 5 (a). At $t = 0.25s$, the training with Dp width 10 shows results with strong noise in the wavefields, while the other results perform well. At time step $t = 0.35$ and $t = 0.39$, both the prediction given by Dp width 10 and 20 shows the noise in the center of the wave field, and in Figure 6 (n), we saw the incorrect source in the wavefields. The Dp width 40 and 60 shows a good ability to generate the correct phase of the fields.

The other network hyper-parameter, the Fourier modes, could also influence the training. The number of the Fourier mode is also the width of the Fourier domain multiplication

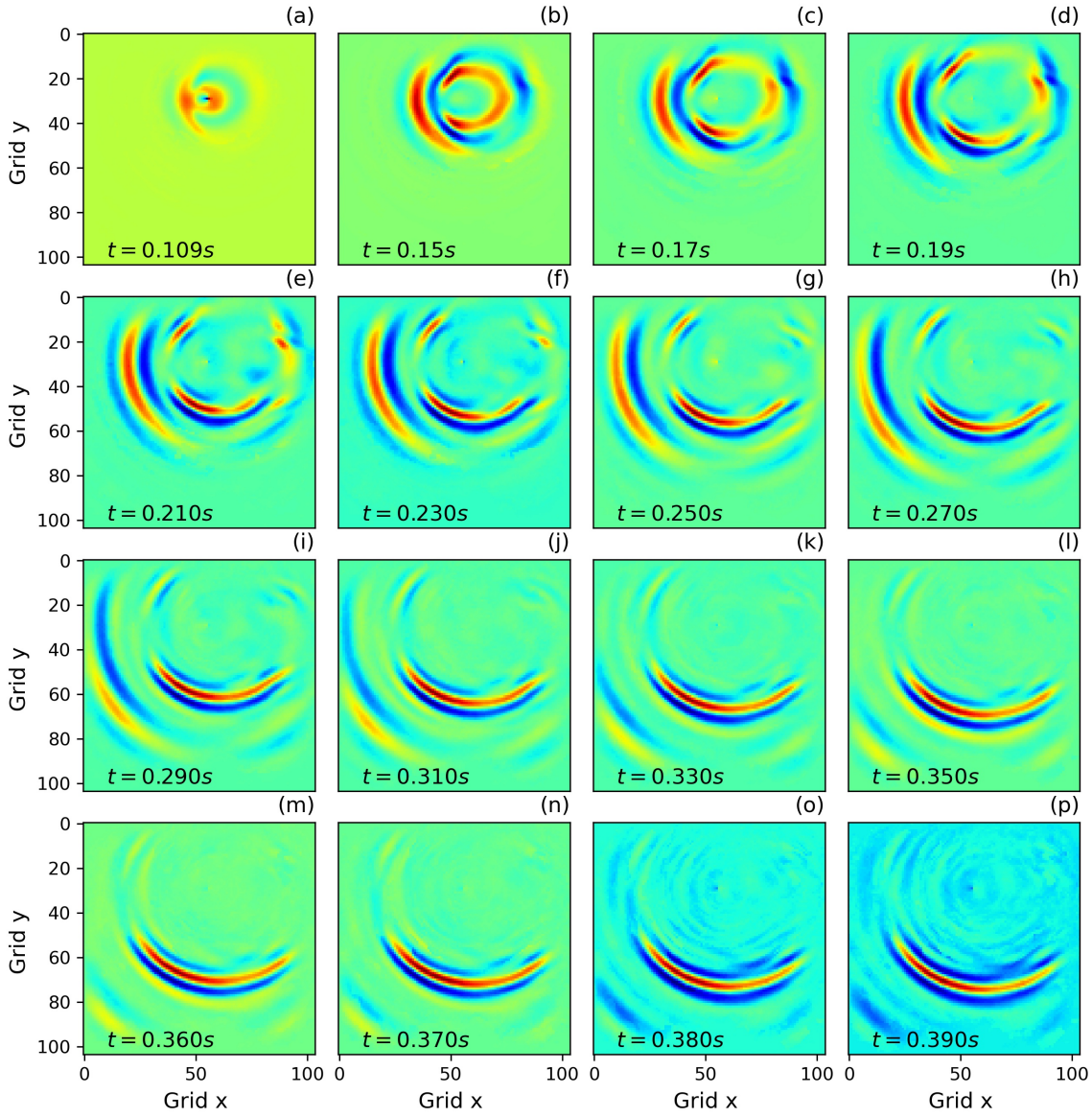

FIG. 4. Nine snap shots of the x component of the wavefields generated with the FNO. With dimension projection width 60 and 33 Fourier models.

kernel, which is critical for learning the spatial derivatives. Figure 7 shows the influence of the Fourier mode influence on the training convergence. We use four Fourier modes in this test which are 10, 20, 33, and 40. Figure 7 illustrates that the loss does have a better convergence rate when we are using a larger Fourier mode. The test in the validation test shows that the Fourier mode 10 and 40 shows bad generalization ability of the network. We can see that the convergence does improve too much among Fourier modes 20, 33, and 40. This could indicate that we may do not need to use a very large Fourier mode for training since the increase of the Fourier model would increase the number of the training parameters. Figure 8 shows the prediction results of these testing. From left to right, in columns, they are the predictions with Fourier mode 10, 20, 33, and 40, with Dp with 40. We choose Dp width 40, since Dp width 40 shows good ability of generalization ability in Figure 5.

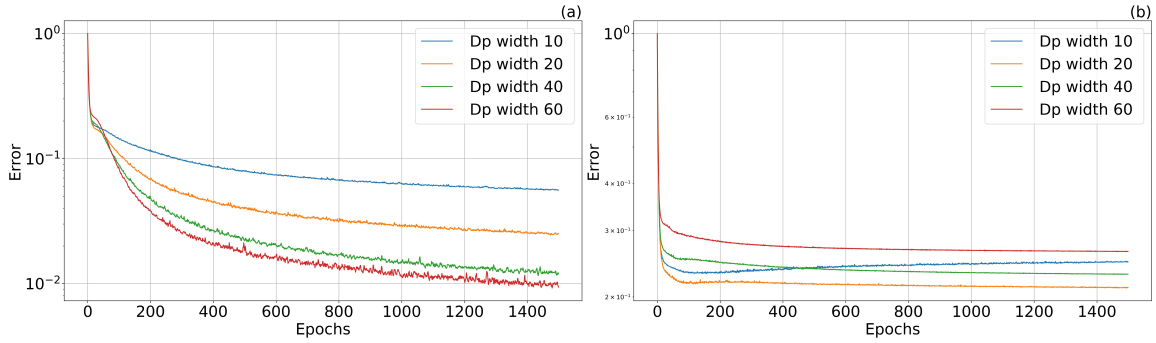

FIG. 5. The training loss and testing loss of training with different Dp width. As the increase with the Dp width the training loss decreases. But the higher value of the Dp width does not guarantee a better convergence of the validation data.

In Figure 8 we can see that at time $t = 0.15s$,$t = 0.25s$,$t = 0.35s$ all the prediction gives promising wavefields, however the at time $t = 0.39$, they show different orders of noise in the fields. The FNO with Fourier mode 10 almost failed to give the prediction at this time. Figure 8 (n) and (p) also generate fields with noise, and in Figure 8 we could see the incorrect source point influence in the field. This should be one of the consequences of the over-fitting we discussed in Figure 7.

We observed several issues with this method, which needed to be improved in the future. Firstly, is that FNO shows a weak ability to generate fields at a long prediction time. In Figure 8 and Figure 6, we all observed that before $t = 0.25$, the predictions are promising. However, as time propagates, more errors could be observed. More numbers of Fourier layers would solve this problem. However, it would result in a much more number of the parameters needed to be trained. The second is that the wavefields we use in training are the time domain wavefields. If we want to train a network that has a good generalization ability, we need to train a huge amount of velocity models with sufficient long times of wavefields. It means that the loading of the data for the wavefields, and the training processes are all hard burdens for the device. For instance, in this study, the training data set for VX wavefields is around 8Gb, and thus loading the fields onto the GPU takes nearly 7 minutes. In these tests, we only have 16GB of the RAM on GPU, and training the network to predict 300 steps with 100 training models is the maximum time steps that we could predict. Third, from figure 7 (b) and 5 (b), we could see that convergence of validation loss is bad. This also means the model has a bad ability for generalization of the training. This could also be due to that we only train 100 models, and training on such a small data set is not enough for generalization for the elastic wave equation.

Computational Performance comparison

One of the most important attractive features of the Fourier Neural Operator has a much faster speed of generating forward modeling wavefields. Table 1 shows computational cost by using the staggered grid finite difference method and the Fourier Neural Operator with different Dp widths and the Fourier modes. The Finite difference method code is based on Python and is a Recurrent Neural Network based Forward modeling, which uses the 2D convolution operation to calculate the spatial partial derivative Zhang et al. (2020). The

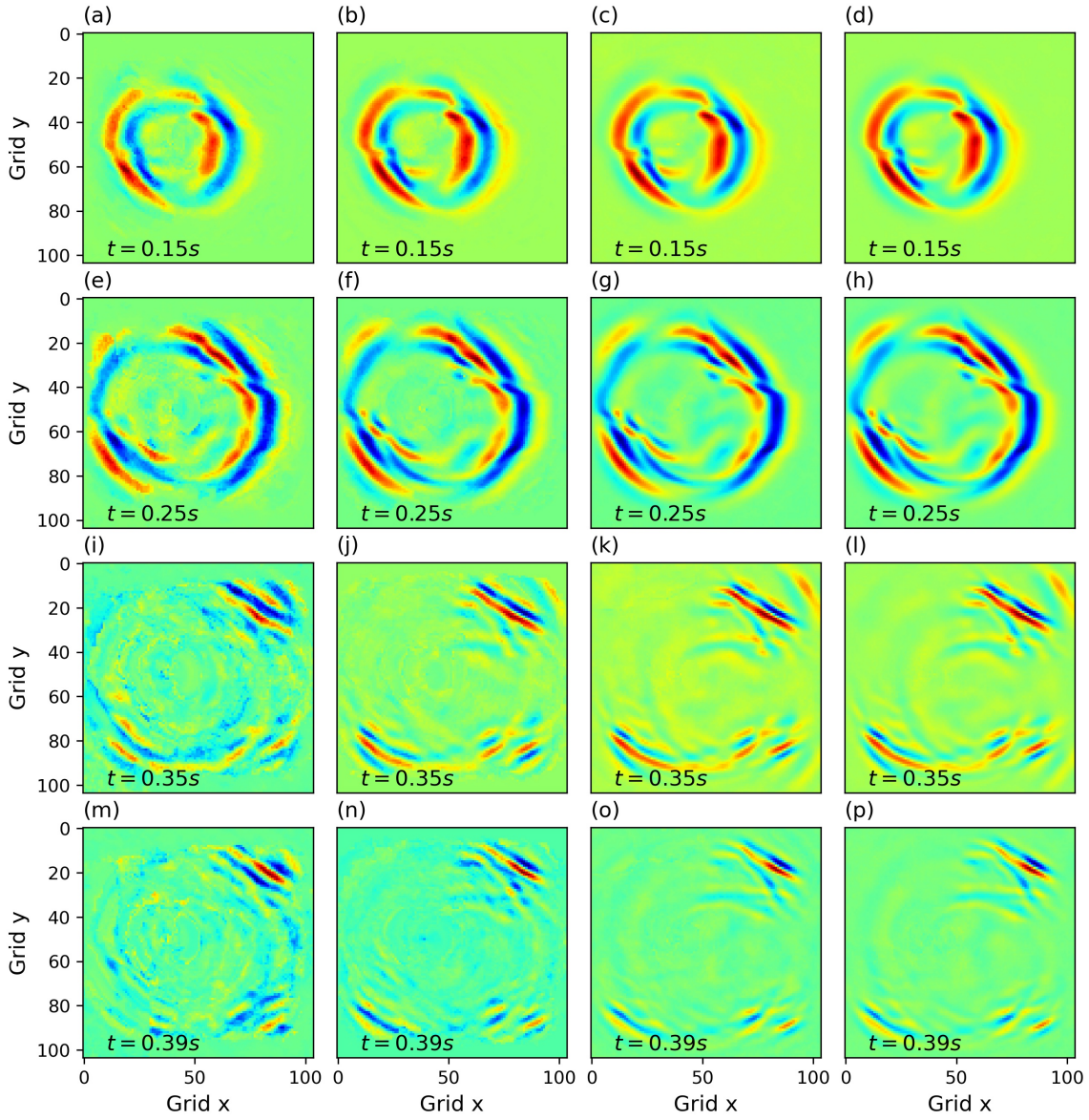

FIG. 6. The comparison of the training with dimension projection width at different time steps. From left to right, in columns, are the wavefields generated with different dimension projection widths: 10, 20, 40, and 60. The time of the shots is labelled on the left corner of each figure. The FNO could generate the fields with the correct phases for the wavefields. However, if we do not have enough width, as time propagates, the accuracy of the phase position would decrease.

maximum prediction time step is 500. We perform the calculation on both the CPU and GPU. The CPU we are using is the 2.3 GHz 8-Core Intel Core i9, and GPU is Nividia V100 in ARC cluster provided by the University of Calgary. First, we can see that on CPU, the FNO algorithm is about 10 times faster than the finite difference method, even with a relatively large Dp width and Fourier mode. The computational speed improvements with GPU are very significant. FNO algorithm is about approximately a thousand times faster than the Finite difference method. This is because the major mathematical operations in the FNO are matrix multiplication, element point-wise multiplication, and the Fast Fourier

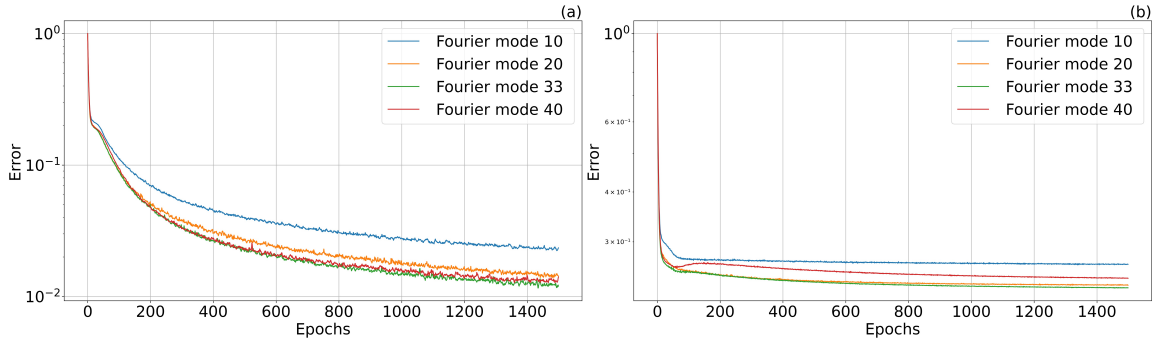

FIG. 7. The loss for training with different Fourier mode. (a) The loss for the training data set with Fourier mode 10, 20, 33 and 40. (b) The validation loss with Fourier mode 10, 20, 33 and 40.

Transform. These operations are well suited for GPU acceleration. While, the finite difference method, according to Zhang et al. (2020), needs to take the finite difference stencil to create a kernel to perform the spatial convolution, which needs to loop through space, and if something needs to do in loops, then usually it would take times. We can also see that, with the increase of the Dp width and the Fourier modes, the computational cost also increases, which means that not only the higher value of these two network hyper-parameter would cause over-fitting, it also increase the computational cost. Thus the hyper-parameter choosing plays an essential role in training.

Table 1. Forward modeling computational performance comparison

Modeling method	CPU	GPU	CPU/GPU Ratio
Finite Difference method (PY)	1.345670445s	0.95749302s	1.4
FNO(Dp width=10, Model=33)	0.10359570s	0.00122648s	84
FNO(Dp width=20, Model=33)	0.13362584s	0.00125602s	106
FNO(Dp width=30, Model=33)	0.29434162s	0.00130555s	225
FNO(Dp width=60, Model=33)	0.48886882s	0.00244271s	200
FNO(Dp width=40, Model=10)	0.11607934s	0.00357925s	32
FNO(Dp width=40, Model=20)	0.15244401s	0.00177402s	85
FNO(Dp width=40, Model=40)	0.26186507s	0.00196767s	133
FNO(Dp width=40, Model=60)	0.33643302s	0.00242882s	138

CONCLUSIONS

In this study, we train a fast forward modeling method with the Fourier Neural Operator(FNO). If the network is well trained, the computational speed for generating the wavefields is about 1000 times faster than the traditional finite difference method. The network consists of three parts, which are two dimensions projection layers that operate on time, and several Fourier layer that learns the spatial partial derivatives. The power of the Fourier Neural operator comes from the combination of the linear operation, operators that resemble partial differential calculation (via the Fourier transform), and the non-linear local activation. The numerical tests suggest that FNO could generate promising wavefields within certain prediction steps, however, with the decreasing of accuracy as time propagates.

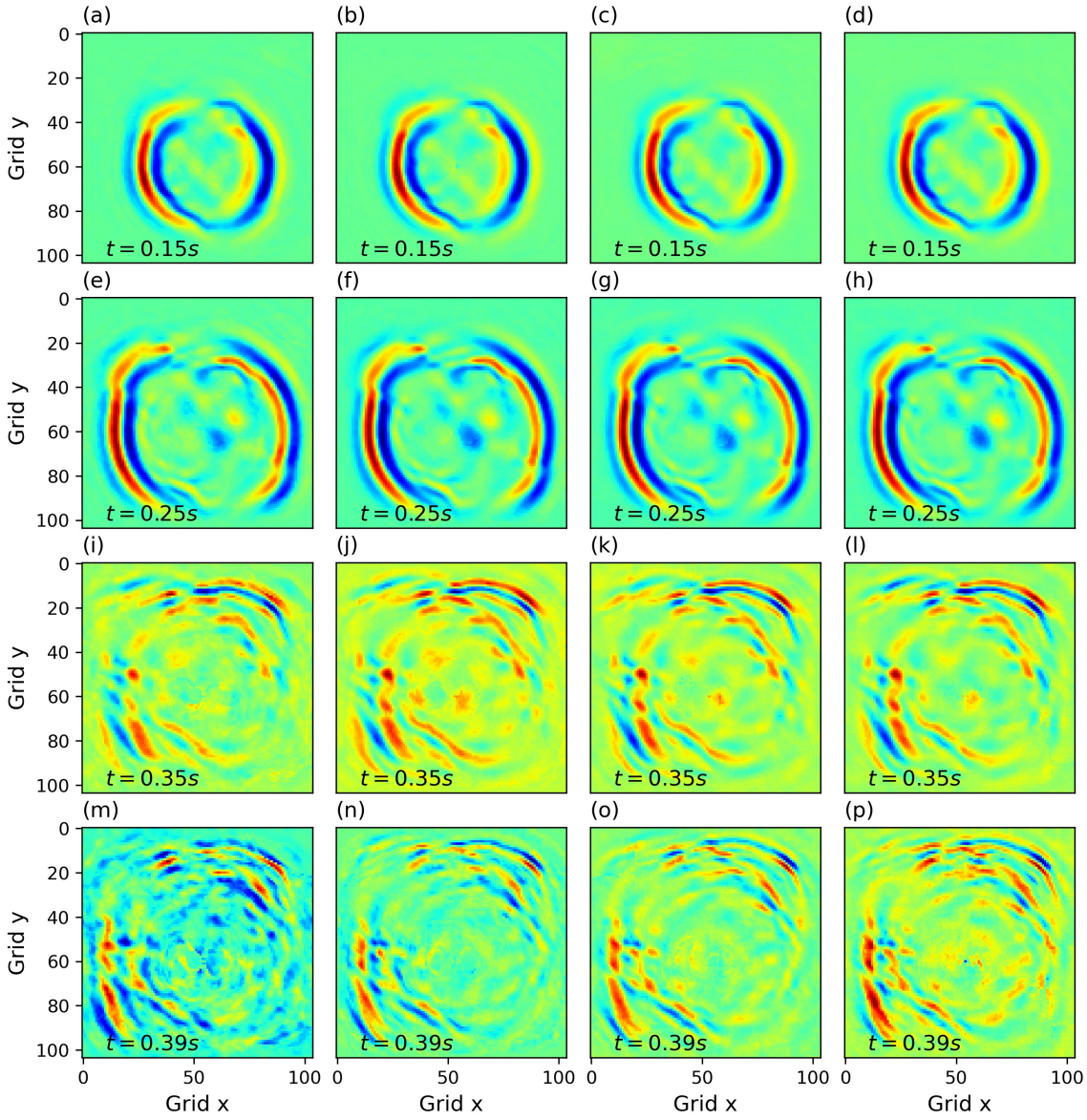

FIG. 8. The wavefields generated with FNO by using different Fourier mode and different time step. From left to right, in column are predictions with Fourier mode 10, 20, 33 and 40 respectively. The time of the wavefield snap shots are labeled on the left corner of each picture.

ACKNOWLEDGMENT

We thank the sponsors of CREWES for continued support. This work was funded by CREWES industrial sponsors and NSERC (Natural Science and Engineering Research Council of Canada) through the grant CRDPJ 543578-19. The first author is also supported by the Chine Scholarship Council (CSC).

REFERENCES

Esmaeilzadeh, S., Azizzadenesheli, K., Kashinath, K., Mustafa, M., Tchelepi, H. A., Marcus, P., Prabhat, M., Anandkumar, A. et al., 2020, Meshfreeflownet: a physics-constrained deep continuous space-time super-

resolution framework, *in* SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 1–15.

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S., 2021, Machine learning–accelerated computational fluid dynamics: Proceedings of the National Academy of Sciences, **118**, No. 21.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A., 2020, Fourier neural operator for parametric partial differential equations: arXiv preprint arXiv:2010.08895.

Raissi, M., Perdikaris, P., and Karniadakis, G. E., 2019, 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.

Sun, J., Niu, Z., Innanen, K. A., Li, J., and Trad, D. O., 2020, A theory-guided deep-learning formulation and optimization of seismic waveform inversion: Geophysics, **85**, No. 2, R87–R99.

Zhang, T., Innanen, K. A., Sun, J., and Trad, D. O., 2020, Numerical analysis of a deep learning formulation of multi-parameter elastic full waveform inversion, *in* SEG International Exposition and Annual Meeting, OnePetro.

Zhang, T., Sun, J., Innanen, K. A., and Trad, D. O., 2021, A recurrent neural network for l1 anisotropic viscoelastic full-waveform inversion with high-order total variation regularization, *in* First International Meeting for Applied Geoscience & Energy, Society of Exploration Geophysicists, 1374–1378.