NEURAL INTEGRAL FUNCTIONALS

Zheyuan Hu[a, b], **Tianbo Li**[a], **Zekun Shi**[a, b], **Kunhao Zheng**[a], **Giovanni Vignale**[c], **Kenji Kawaguchi**[b], **Shuicheng Yan**[a], **Min Lin**[a]
[a]SEA AI Lab, [b]School of Computing, National University of Singapore,
[c]Institute for Functional Intelligent Materials, National University of Singapore
zyhu2001@gmail.com, linmin@sea.com

ABSTRACT

Functionals map input functions to output scalars, which are ubiquitous in various scientific fields. In this work, we propose neural integral functional (NIF), which is a general functional approximator that suits a large number of scientific problems including the brachistochrone curve problem in classical physics and density functional theory in quantum physics. One key ingredient that enables NIF on these problems is the functional's explicit dependence on the derivative of the input function. We demonstrate that this is crucial for NIF to outperform neural operators (NOs) despite the fact that NOs are theoretically universal. With NIF, we further propose to jointly train the functional and its functional derivation (FD) to improve generalization and to enable applications that require accurate FD. We validate these claims with experiments on functional fitting and functional minimization.

1 INTRODUCTION

Functionals (Engel & Dreizler, 2011) are mappings from an input function to an output scalar. In machine learning, the losses (e.g., cross-entropy, mean square error) mapping a classifier/predictor function to the scalar loss value to be minimized by gradient descent are functionals. Functional derivative (FD) is in turn, the analog of "gradient" for functionals, which can be used for finding the minima of the functionals. FD is ubiquitous in physics, whose applications involve solving partial differential equations (PDEs), quantum and analytical mechanics, and density functional theory (DFT). Specifically, all of these problems can be formulated as functional minimization problems.

There have been some efforts in learning functionals with ML. One line of work is the neural operators (NOs) (Lu et al., 2019; Li et al., 2020), which can be used as functionals by integrating their output function into a scalar. NOs are theoretically universal as they are essentially multilayer perception (MLP) for functions. However, the difficulty of applying NOs lies in the integrations in their intermediate layers. In practice, these integrals are computed numerically using grids, which can lead to a bias that diminishes its generalization ability. Due to the same reason, it is difficult for NOs to capture the derivative of the input function if not given explicitly.

In this paper, we introduce a novel method, named **Neural Integral Functional** (NIF), as a general approximator for the classical integral-type of functionals defined by

$$J[f] = \int I(\boldsymbol{x}, f(\boldsymbol{x}), \nabla f(\boldsymbol{x})) d\boldsymbol{x}. \tag{1}$$

In some quantum physics literature, this type of functional is also called "semilocal". This form of the functional is prevalent. Many scientific problems can be written in such forms, such as the brachistochrone curve problem in classical physics and density functional theory in quantum physics. Unlike NOs that rely solely on the input function, NIF takes the derivative of the input function as an extra input. We demonstrate that the explicit dependence on the FD is critical for fitting an accurate functional. Our experiments show that NIF excels on tasks where NOs fail. NIF can be used to target a wide range of applications for learning functionals in physics.

Furthermore, the FD of integral-type functionals (Equation 1) has the following analytical form derived from the well-known Euler-Lagrange equation (Fox, 1987)

$$\frac{\delta J}{\delta f}(\boldsymbol{x}) = \frac{\partial I}{\partial f(\boldsymbol{x})} - \nabla \cdot \left(\frac{\partial I}{\partial \left(\nabla f(\boldsymbol{x})\right)} \right), \tag{2}$$

by auto-differentiation software. The FD of NIF has several important uses, including accurate FD prediction, Sobolev training (Czarnecki et al., 2017), and solving variational problems.

We conduct extensive experiments to demonstrate the capability of NIF on functional fitting and functional minimization, from which we validate the importance of the input function derivative and the effectiveness of FD training.

2 Preliminaries

In this work, we consider integral-type functionals given below.

Definition 2.1 (Definite Integral Functional)**.** Given a function $f \in L^2(\mathbb{R}^d, \mathbb{R})$ with input $\boldsymbol{x} \in \mathbb{R}^d$, a functional $J : L^2(\mathbb{R}^d, \mathbb{R}) \to \mathbb{R}$ maps a function to a scalar via the following transformation:

$$J[f] = \int_\Omega I(\boldsymbol{x}, f(\boldsymbol{x}), \nabla f(\boldsymbol{x}))\, d\boldsymbol{x}, \tag{3}$$

where Ω is a subset of $\mathbb{R}^d$ and the integrand I is a real-valued function on three vector inputs.

In real-world applications, the minimum of functionals is of great interest, e.g., minimizing the CE loss for classification in machine learning. Analogous to gradient descent for the minimum of a function, to find the minimum of functionals, we require the concept of *functional derivative*.

For the integral-type functionals, we can derive its FD using the Euler-Lagrange equation (Fox, 1987) given in Equation (2). The detailed definition of FD and the derivation of the above FD formula are presented in Appendix B, in which we also provide examples of FD for several functionals.

3 Method

3.1 Neural Integral Functional

We propose Neural Integral Functional (NIF) to approximate functional and its FD with neural networks. NIF takes function f as input, applies transformation recursively and constructs a complex nonlinear functional. NIF contains: (1) the function transformation layers and (2) the integral layer.

Function Transformation Layers The first part of NIF is a transformation of the input functions. The flexibility of NIF allows I to be arbitrary types of neural networks. Here we consider MLP:

$$I_l^{\boldsymbol{\theta}} = \sigma\Big(\boldsymbol{W}_l I_{l-1}^{\boldsymbol{\theta}}\big(\boldsymbol{x}, f(\boldsymbol{x}), \nabla f(\boldsymbol{x})\big) + \boldsymbol{b}_l\Big), \boldsymbol{W}_l \in \mathbb{R}^{h_l \times h_{l-1}}, \boldsymbol{b}_l \in \mathbb{R}^{h_l}, 1 \le l \le L, \tag{4}$$

where $\boldsymbol{W}_l, \boldsymbol{b}_l$ are the trainable parameters, and σ is the elementwise activation.

Integral Layer Differently, at the $(L+1)$-th layer, we compute the integral to transform the output function from the previous layer to a scalar, such that the entire model $J^{\boldsymbol{\theta}}[f]$ is a functional. Specifically, we design the integral at the last layer to be a numerical integral scheme, i.e.,

$$J^{\boldsymbol{\theta}}[f] = \int I_L^{\boldsymbol{\theta}}\big(\boldsymbol{x}, f(\boldsymbol{x}), \nabla f(\boldsymbol{x})\big)\, d\boldsymbol{x} \approx \sum_{i=1}^{N_{\text{grid}}} w_i\, I_L^{\boldsymbol{\theta}}(\boldsymbol{x}_i, f(\boldsymbol{x}_i), \nabla f(\boldsymbol{x}_i)), \tag{5}$$

where w_i are the corresponding weights. It is noteworthy that the grid points in NIF can be arbitrary.

Functional Derivative (FD) The FD of the NIF $J^{\boldsymbol{\theta}}[f]$ w.r.t. the input function f can be computed analytically by Equation (2), where the FD is reduced to function differentiation w.r.t. $x, f, \nabla f$.

As an emerging tool, NIF is capable of performing various tasks.

3.2 Fitting Functional, Functional Derivative & Sobolev Training

Here, we formulate functional fitting problem and the FD loss for FD fitting and Sobolev training.

We consider an arbitrary grid for integral $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^{N_{\text{grid}}}$, a set of train functions $\mathcal{F} = \{f_1, f_2, \cdots, f_{N_{\text{train}}}\}$, and the functional to be learned $J[f]$. In the cases where the FDs $\frac{\delta J}{\delta f}$ are

Table 1: Relative L_2 error results for fitting two synthetic functionals with polynomial and sinusoid input functions, with ($\lambda_f = 1$) and without ($\lambda_f = 0$) the FD loss.

Component			Error of Functional I_0 (Polynomial)			Error of Functional I_0 (Sinusoid)		
Model	FD	∇f	Integrand	Integral	FD	Integrand	Integral	FD
FNO	×	NA	1.270E-03	4.896E-03	2.451E-02	5.197E-04	4.946E-04	6.062E-02
FNO	✓	NA	6.943E-04	4.563E-03	7.775E-04	1.498E-04	5.620E-04	1.582E-03
NIF	×	×	6.605E-04	4.773E-03	8.169E-04	1.948E-04	5.123E-04	9.270E-04
NIF	✓	×	**3.845E-04**	**4.455E-03**	**2.963E-04**	**1.473E-04**	**4.896E-04**	**3.060E-04**
NIF	×	✓	1.141E-03	5.266E-03	7.852E-02	7.151E-04	6.052E-04	3.728E-02
NIF	✓	✓	1.117E-03	4.910E-03	6.333E-03	4.135E-04	5.757E-04	1.939E-03
Component			Error of Functional I_1 (Polynomial)			Error of Functional I_1 (Sinusoid)		
Model	FD	∇f	Integrand	Integral	FD	Integrand	Integral	FD
FNO	×	NA	9.974E-03	2.092E-02	1.223E+01	5.231E-03	1.965E-03	3.868E+00
FNO	✓	NA	2.555E-01	5.754E-01	6.587E+00	2.698E-01	2.283E-01	3.289E+00
NIF	×	×	6.182E-01	6.030E-01	3.536E+00	9.407E-01	8.675E-01	1.893E+00
NIF	✓	×	6.759E-01	6.361E-01	2.081E+00	9.118E-01	7.980E-01	6.153E-01
NIF	×	✓	7.667E-03	2.146E-02	8.071E-01	3.494E-03	3.439E-03	7.760E-02
NIF	✓	✓	**7.361E-03**	**2.091E-02**	**3.704E-02**	**7.841E-04**	**1.040E-03**	**5.713E-03**

available, we can add a loss term for the FD to improve the performance. The overall training loss for fitting the functional is $\min_{\boldsymbol{\theta}} \mathcal{L}_{\text{func}}(\boldsymbol{\theta}) + \lambda_f \mathcal{L}_{\text{FD}}(\boldsymbol{\theta})$, where

$$\mathcal{L}_{\text{func}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{F}|}\sum_{f\in\mathcal{F}}\sum_{i=1}^{N_{\text{grid}}} w_i\left(I^{\boldsymbol{\theta}}_{L,i} - I_i\right)^2, \quad \mathcal{L}_{\text{FD}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{F}|}\sum_{f\in\mathcal{F}}\sum_{i=1}^{N_{\text{grid}}} w_i\left(\frac{\delta J^{\boldsymbol{\theta}}}{\delta f}(\boldsymbol{x}_i) - \frac{\delta J}{\delta f}(\boldsymbol{x}_i)\right)^2.$$

The first functional loss aims to fit the functional itself on the pointwise values, where $I^{\boldsymbol{\theta}}_{L,i} := I^{\boldsymbol{\theta}}_L(\boldsymbol{x}_i, f(\boldsymbol{x}_i), \nabla f(\boldsymbol{x}_i))$ is the output at the L-th layer given a sample point $\boldsymbol{x}_i$ and w_i is the integral weight in Equation (5). The second term, FD loss, regularizes the similarity between the FD of the model and the ground truth. $\mathcal{L}_{\text{FD}}$ is crucial when downstream tasks require explicitly the FD, or for Sobolev training that improves generalization (Czarnecki et al., 2017).

3.3 Solving Variation Problems

When the functional is known, searching for a function that minimizes the functional is called the variation problem, i.e., to find f^* such that $f^* = \arg\min_{f\in\mathcal{F}} J[f]$.

To solve the variational problem, we first train a NIF $J^{\boldsymbol{\theta}}$ to approximate J on both functional and FD. Once we obtain the trained NIF J^{θ}, the variation problem searches for the $f^* = \arg\min_f J^{\theta}[f]$. In practice, we need to use a parameterized function f^{ϕ}. Then it is converted into a parameter space problem $\phi^* = \arg\min_{\phi} J^{\theta}[f^{\phi}]$. One common choice for f^{ϕ} is the linear combination of basis functions, where ϕ denotes the combination coefficients.

4 Experiments

In this section, we present experiments. More details are listed in Appendix C.

4.1 Functional Fitting: Synthetic Functional

We use NIF to fit the following synthetic functionals, which are used to illustrate the effectiveness of the training scheme of our NIF method: $J_0 : f(x) \mapsto \int_0^1 f(x)dx$, $J_1 : f(x) \mapsto \int_0^1 xf'(x)dx$. The FDs of J_0 and J_1 w.r.t. f are 1 and -1, respectively.

Data Generation To fit the functional and its FD, we first generate a synthetic dataset of input functions f and the corresponding ground-truth functional output $J[f]$. This is analogous to the data and labels in machine learning. The input function is sampled from linear combinations of

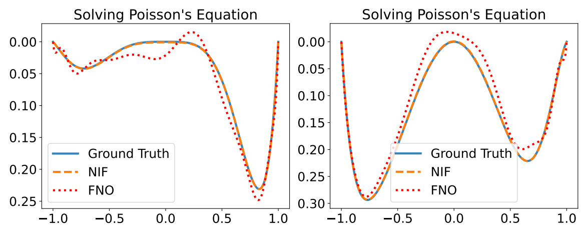

Figure 1: Comaprison between ground truth and the solutions obtained from NIF and FNO in Poisson's equation with two different ground truths u_1 (left) and u_2 (right).

sinusoids and polynomials, i.e. $f_{\text{poly}} = \sum_{i=0}^{N-1} a_i x^i$ and $f_{\sin} = \sum_{i=0}^{M-1} b_i \sin(c_i x)$. Both are universal approximators to ensure good coverage of the input space by tuning the parameters. We choose $N = M = 20$, $a_i, b_i \sim$ Uniform$(-1, 1)$ and $c_i \sim$ Uniform$(0.1, 2.1)$. 1100 data points are randomly generated with 90% for training and the rest 10% for testing.

Compared Models (1) *Fourier neural operator (FNO)* (Li et al., 2020): We append an integral layer at the end of FNO, and its input only includes $\boldsymbol{x}, f(\boldsymbol{x})$. (2) *NIF with/without FD loss*: to investigate if the FD loss is able to reduce test FD error and the regression error at the functional level, i.e., the effectiveness of FD-based Sobolev training on improving generalization. (3) *NIF with/without* ∇f *as input*: to validate the necessity of ∇f as a direct input to NIF.

Evaluation metrics We evaluate the fitting performance with relative L_2 errors for the functional J and functional derivative $\delta J/\delta f$, as well as pointwise error on the integrated I. They are shown in Table 1 under column "integral", "FD" and "integrand" respectively.

The relative L_2 errors at test time are shown in Table 1, where we have several observations.

∇f **is necessary for fitting functionals**. For the functional I_0, since the ground truth doesn't include ∇f, FNO and NIF work well in every metric. However, NIF with ∇f greatly outperforms others when fitting I_1. NIF without ∇f works even better for I_0, since the ground truth doesn't include ∇f, which prevents the noisy feature for this specific functional. But, ∇f does show up in most real-world functionals. This justifies the effectiveness of inputting ∇f in NIF.

The use of FD regularization is beneficial for the fitting of functionals. The use of FD loss leads to a significant reduction in all three errors, bringing the model closer to the ground-truth functional. Both FNO and NIF have even lower errors when FD regularization is applied.

4.2 Solving Poisson's Equation

In this section, we solve Poisson's problem $-\Delta u = f(\boldsymbol{x})$, in $\boldsymbol{x} \in \Omega = [-1, 1]$ by NIFs, whose corresponding functional is $\int -\frac{1}{2}|\nabla u|^2 + uf$, i.e., the condition that its FD equals zero is just the original Poisson's equation. We consider the zero boundary condition, i.e., the ground truth solution $u(\boldsymbol{x}) = 0$ on $\partial\Omega$, while f is generated by the true solutions $u_1(x) = (x^2 - 1)(0.8x^5 + 0.9x^4)$ and $u_2(x) = (x^2 - 1)(-0.8x^7 + x^2)$. The input functions generating the training set for functional and FD learning are polynomials of the form $(x^2 - 1)P(x)$ where $(x^2 - 1)$ to ensure the zero boundary condition, and $P(x) = \sum_{i=0}^{19} a_i x^i$ and $a_i \sim$ Unif$(-1, 1)$.

The ground truth solution and solutions obtained from NIF and FNO are compared in Figure 1. The ground truth and the solution obtained from NIF are almost indistinguishable, while the error of FNO is much larger, i.e., NIF significantly outperforms FNO.

5 Conclusion

In this paper, we propose the NIF, adopting MLP to approximate functionals and enabling FD computation by the definition of FD. It is capable of fitting unknown functionals & their FD and minimizing functionals. Extensive experiments on functional fitting and functional minimization validate the design of NIF that inputs the derivatives of the input function, and its FD computation.

REFERENCES

Yixiao Chen, Linfeng Zhang, Han Wang, and Weinan E. Deepks: A comprehensive data-driven approach toward chemically accurate density functional theory. *Journal of Chemical Theory and Computation*, 17(1):170–181, 2020.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630*, 2020.

Wojciech M Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. *Advances in Neural Information Processing Systems*, 30, 2017.

Sebastian Dick and Marivi Fernandez-Serra. Machine learning accurate exchange and correlation functionals of the electronic density. *Nature communications*, 11(1):1–10, 2020.

Emilien Dupont, Yee Whye Teh, and Arnaud Doucet. Generative models as distributions of functions. *arXiv preprint arXiv:2102.04776*, 2021.

Emilien Dupont, Hyunjik Kim, SM Eslami, Danilo Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you should treat it like one. *arXiv preprint arXiv:2201.12204*, 2022.

E. Engel and R.M. Dreizler. *Density Functional Theory: An Advanced Course*. Theoretical and Mathematical Physics. Springer Berlin Heidelberg, 2011. ISBN 9783642140907.

Charles Fox. *An introduction to the calculus of variations*. Courier Corporation, 1987.

S Alireza Ghasemi and Thomas D Kühne. Artificial neural networks for the kinetic energy functional of non-interacting fermions. *The Journal of Chemical Physics*, 154(7):074107, 2021.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

James Kirkpatrick, Brendan McMorrow, David HP Turban, Alexander L Gaunt, James S Spencer, Alexander GDG Matthews, Annette Obika, Louis Thiry, Meire Fortunato, David Pfau, et al. Pushing the frontiers of density functionals by solving the fractional electron problem. *Science*, 374(6573):1385–1389, 2021.

Li Li, Thomas E Baker, Steven R White, Kieron Burke, et al. Pure density functional for strong correlation and the thermodynamic limit from machine learning. *Physical Review B*, 94(24): 245129, 2016.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Chao Ma and José Miguel Hernández-Lobato. Functional variational inference based on stochastic process generators. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 21795–21807. Curran Associates, Inc., 2021.

Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent. In S. Solla, T. Leen, and K. Müller (eds.), *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

Yuxiang Mo, Guocai Tian, and Jianmin Tao. Comparative study of semilocal density functionals on solids and surfaces. *Chemical Physics Letters*, 682:38–42, 2017.

John P Perdew and Karla Schmidt. Jacob's ladder of density functional approximations for the exchange-correlation energy. In *AIP Conference Proceedings*, volume 577, pp. 1–20. American Institute of Physics, 2001.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 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, 2019.

Kevin Ryczko, David A Strubbe, and Isaac Tamblyn. Deep learning and density-functional theory. *Physical Review A*, 100(2):022512, 2019.

John C Snyder, Matthias Rupp, Katja Hansen, Klaus-Robert Müller, and Kieron Burke. Finding density functionals with machine learning. *Physical review letters*, 108(25):253002, 2012.

Shengyang Sun, Guodong Zhang, Jiaxin Shi, and Roger Grosse. Functional variational bayesian neural networks. *arXiv preprint arXiv:1903.05779*, 2019.

Kevin Vu, John C Snyder, Li Li, Matthias Rupp, Brandon F Chen, Tarek Khelif, Klaus-Robert Müller, and Kieron Burke. Understanding kernel ridge regression: Common behaviors from simple functions to density functionals. *International Journal of Quantum Chemistry*, 115(16):1115–1128, 2015.

A Related Work

Machine Learning and Functionals In machine learning, Dupont et al. (2022; 2021) proposed to view images, 3D shapes, and neural radiance fields as continuous function inputs and tested the function data on meta-learning, generative modeling, and classification. However, physics-related functionals, where the definition of *functional* comes from, and FD are not considered. Functionals are also adopted to solve specific applications including functional variational inference (Ma & Hernández-Lobato, 2021) and variational Bayes (Sun et al., 2019). Mason et al. (1999) provided a functional derivative perspective for optimization.

Another major field that learns functionals with ML is density functional theory (DFT). Before the era of deep learning, early work mainly focused on kernel regression to learn density functionals. Vu et al. (2015) investigated the importance of hyperparameter setting in kernel ridge regression for DFT. However, Vu et al. (2015) did not consider the FD, which is critical to the DFT problem. Snyder et al. (2012) computed FD by direct differentiation to the input discretized density. Their computed self-consistent density suffered from inaccuracy: the error is an order of magnitude larger than the true self-consistent density. Li et al. (2016) utilized a density matrix renormalization group for self-consistency computation. Ghasemi & Kühne (2021); Ryczko et al. (2019) fit every density functional in DFT computation and achieved chemical accuracy, but their model did not compute the FD. Our framework enables FD for all neural network-based functionals, so that the model in Ryczko et al. (2019) can be flexibly embedded.

DFT has seen the emerging applications of functional learning. Examples include learning kinetic functionals or exchange-correlation (XC) functionals (Snyder et al., 2012; Li et al., 2016; Vu et al., 2015; Dick & Fernandez-Serra, 2020; Chen et al., 2020; Kirkpatrick et al., 2021). Unlike NOs, the functionals studied in DFT are usually *semilocal* (Mo et al., 2017; Engel & Dreizler, 2011), i.e., only depending on local values of the input function and its derivatives. They can be written as an integral over the pointwise mapping of the input function. In contrast with neural operators, the integral in this type of functional can be amortized using stochastic integration and thus can be asymptotically unbiased. The limitation of the functionals in DFT is that it is not universal enough to model non-local functionals. It is alleviated by introducing the derivative of the input function or even the second-order derivative of the input function, i.e., GGA and meta-GGA functionals (Perdew & Schmidt, 2001).

Physics-Informed Machine Learning Although functionals have not drawn much attention from the machine learning community, machine learning has become popular in other physics, and science-related fields, e.g., physics-informed neural networks (PINNs) (Raissi et al., 2019) for solving PDEs, DeepONet (Lu et al., 2019) and Fourier neural operator (FNO) (Li et al., 2020) for learning operators for solving the Kohn-Sham DFT, Hamiltonian neural networks (HNNs) (Greydanus et al., 2019) and Lagrangian neural network (Cranmer et al., 2020) for discovering the physical law governs by Hamiltonians Euler-Lagrangian equation in analytical mechanics.

B Functional Derivative

Definition B.1 (Variation of a Functional)**.** The variation of a functional $J[\cdot]$ is defined as $\delta J := J[f+\epsilon\eta] - J[f], \epsilon \in \mathbb{R}$, where ϵ is an infinitesimal number, and η is an arbitrary function called the **test function**.

Definition B.2 (Functional Derivative)**.** In this section, we provide detailed definitions of functional derivatives. We follow the definition provided in (Engel & Dreizler, 2011) The functional derivative of a functional $J[\cdot]$ is defined as a **function** $\frac{\delta J}{\delta f}(x)$ whose inner product with any test function yields the first order expansion of functional variation δJ, treated as an ordinary function of the infinitesimal ϵ:

$$\begin{aligned}\langle \frac{\delta J}{\delta f}, \eta\rangle = \int dx\, \frac{\delta J}{\delta f}(x)\eta(x) =& \frac{d}{d\epsilon}\delta J\Big|_{\epsilon=0} \\ =& \frac{d}{d\epsilon}(J[f+\epsilon\eta]-J[f])\Big|_{\epsilon=0} = \frac{d}{d\epsilon}J[f+\epsilon\eta]\Big|_{\epsilon=0}.\end{aligned} \tag{6}$$

Thus we can rewrite the Taylor expansion of ϵ around 0 as

$$\begin{aligned}J[f+\epsilon\eta] =& J[f] + \frac{d}{d\epsilon}J[f+\epsilon\eta]\Big|_{\epsilon=0}\epsilon + \mathscr{O}(\epsilon^2) \\ =& J[f] + \langle \frac{\delta J}{\delta f}, \eta\rangle\epsilon + \mathscr{O}(\epsilon^2).\end{aligned}$$

Consider the definite integral-based functional:

$$J = \int I(\boldsymbol{x}, f(\boldsymbol{x}), \nabla f(\boldsymbol{x}), \cdots, \nabla^n f(\boldsymbol{x}))d\boldsymbol{x}.$$

Based on the definition of FD,

$$\begin{aligned}J[f+\epsilon\eta]-J[f] =& \int \left(I(\boldsymbol{x}, f(\boldsymbol{x})+\epsilon\eta(\boldsymbol{x}), \cdots, \nabla^n f(\boldsymbol{x})+\epsilon\nabla^n\eta(\boldsymbol{x})) - I(\boldsymbol{x}, f(\boldsymbol{x}), \cdots, \nabla^n f(\boldsymbol{x}))\right)d\boldsymbol{x} \\ =& \epsilon\sum_{i=0}^{n}\int \frac{\partial I}{\partial\left(\nabla^i f(\boldsymbol{x})\right)}\nabla^i\left(\eta(\boldsymbol{x})\right)d\boldsymbol{x} + \mathscr{O}(\epsilon^2) \\ =& \epsilon\sum_{i=0}^{n}(-1)^i\int \nabla^i\left(\frac{\partial I}{\partial\left(\nabla^i f(\boldsymbol{x})\right)}\right)\eta(\boldsymbol{x})d\boldsymbol{x} + \mathscr{O}(\epsilon^2),\end{aligned} \tag{7}$$

where in the last equation we consider integral by parts.

For instance, we consider the following basic functionals:

$$\begin{aligned}J_0 : f(x) \mapsto \int f(x)dx, &\quad \frac{\delta I_0}{\delta f}(x) = 1. \\ J_1 : f(x) \mapsto \int xf'(x)dx, &\quad \frac{\delta I_1}{\delta f}(x) = -1. \\ J_2 : f(x) \mapsto \int f^n(x)dx, &\quad \frac{\delta I_2}{\delta f}(x) = nf^{n-1}(x),\ n \in \mathbb{N}^*.\end{aligned}$$

For J_0, we have $I_0(x, f(x)) = f(x)$, so $\frac{\partial I_0}{\partial f(x)} = 1$ and $\frac{\delta J_0}{\delta f}(x) = \frac{\partial I_0}{\partial f(x)} = 1$. For J_1, we have $I_1[x, \nabla f(x)] = x\nabla f(x)$, so $\frac{\partial I_1}{\partial f(x)} = x$, $\frac{d}{dx}\left(\frac{\partial I_1}{\partial f(x)}\right) = 1$ and $\frac{\delta J_1}{\delta f}(x) = -\frac{d}{dx}\left(\frac{\partial I_1}{\partial f(x)}\right) = -1$. For J_2, we have $I_2[x, f(x)] = f^n(x)$, so $\frac{\partial I_2}{\partial f(x)} = nf^{n-1}(x)$ and $\frac{\delta J_2}{\delta f}(x) = \frac{\partial I_2}{\partial f(x)} = nf^{n-1}(x)$.

C Experimental Detail

The results are the average of 5 independent runs. We omit its standard deviation since it is 1 to 2 orders of magnitude smaller than the mean.

C.1 Fitting Synthetic Functionals

We generate 1100 data, the number of uniform grid points is 256, and the model is trained for 10001 epochs with Adam of 1e-3 learning rate. $\lambda_f = 1$ for FD training. The network is a 4-layer net with 64 units at hidden layers, activated by GeLU (Hendrycks & Gimpel, 2016). For Fourier neural operator (FNO) (Li et al., 2020), we use a 4-layer FNO with 64 hidden units activated by GeLU.

C.2 Solving Poisson's Equation with NIF

We generate 1100 data, the number of grid points is 256, and the model is trained for 100001 epochs with Adam + lr exponential decay (decay_epoch=5k). $\lambda_f = 0.1$. The network is a 4-layer net with 64 units at hidden layers, activated by GeLU (Hendrycks & Gimpel, 2016). For Fourier neural operator (FNO) (Li et al., 2020), we use a 4-layer FNO with 32 hidden units activated by GeLU. The functional minimization process after obtained the trained functional by NIF/FNO is conducted for 100001 epochs with Adam + lr exponential decay (decay_epoch=5k).