Differentiable Optimisation layers

Koushiki Dasgupta Chaudhuri, Suryanarayana Maddu

June 2021

1 Abstract

In this study, we investigate the utility of differentiable layers in neural networks for learning consistent PDE differential operators. Deep learning is usually based on end-to-end training with layers composed of explicit expressions with parameters that can be learned from data. However, recent advances in differentiable programming has enabled us to formulate deep learning architectures with *implicit* layers. In contrary to traditional layers with explicit computable rules, *implicit* layers are designed to satisfy a set of user-defined condition/constraints.

2 Introduction

Given $x \in X, z \in Z$ explicit layers involve the explicit function $f : X \to Z$, i.e.

$$z = f(x) \tag{1}$$

An implicit layer is instead defined by a function $g : X \times Z \to R^n$, which is a joint function of both the input x and the output z, and where the output of the layer z is required to satisfy some constraint, i.e., finding the root of the equation, find z such that

$$g(x, z) = 0 \tag{2}$$

Implicit layers separate the output of the layer from the way in which the output is computed. This separation of a layer's objective from its solution method is useful in a wide variety of applications - such as comparing optimization solvers or differential equation solvers among each other based upon how well they satisfy the constraints imposed upon the layer. The main advantage of implicit layers, however, is in the context of deep learning and automatic differentiation, in the building of deep equilibrium models [3], neural ODEs [5], differentiable optimization layers [2] etc. The implicit function theorem can be used to directly compute gradients for these layers at the solution point of their constraint equations, thus removing the need for storing intermediate variables/temporary iterates created along the way of computing the solution. This highly improves memory consumption and numerical accuracy of these methods, and is a massive improvement over traditional backpropagation in explicit layers [8]. A differentiable optimization layer is a type of implicit layer which solves an optimization problem as the layer's function, i.e. equation 2 is an optimization problem.
A generic optimization problem can be defined as :

$$\begin{aligned} &\underset{z}{\text{minimize}} f(z, x) \\ &\text{subject to } z \in C(x) \end{aligned} \tag{3}$$

where $z \in R^n$ is the optimization variable, $x \in R^m$ is additional problem data that defines the optimization problem, $f : R^n \times R^m \to R$ is the objective function and $C(x) \subseteq R^n$ denotes a constrain set.
The output of an optimization layer is the optimal solution(z^*) to equation 3 :

$$z^* = \underset{z \in C(x)}{\text{argmin}} f(z, x) \tag{4}$$

where argmin denotes the value of the variable z at which minimum of f(z,x) is obtained.
We used differentiable optimization layers in our work to learn "hard constraints" from data, i.e. constraints which are directly embedded in the layers of the neural network.

3 Methodology and Results

3.1 De-noising

We used differentiable optimization layers in a toy denoising problem - to denoise a noisy 1D signal given training data consistency of noisy and clean signals generated from the same distribution. Two types of 1D signals were used in this work -
(i) cosine signals(range -1 to 1) with noise drawn from a multivariate normal/Gaussian distribution

Table 1: Finding optimum order of norm and differencing for denoising cosine signals with 0.1 noise

Norm used	Type of differencing	Best λ	Least Train Loss	Least Test Loss	Search Time
l_1 norm	1st order	2.6316	0.1338	0.1354	1 min 17s
l_1 norm	2nd order	7.8947	0.0849	0.0833	1 min 24s
l_1 norm	3rd order	34.2105	0.0819	0.0796	2 mins
l_2 norm	1st order	7.8947	0.0915	0.0934	1 min 4s
l_2 norm	2nd order	50	0.0928	0.0903	1 min 5s
l_2 norm	3rd order	50	0.141	0.1355	1 min 6s

Table 2: Finding optimum order of norm and differencing for denoising piecewise constant signals with 5 noise

Norm used	Type of differencing	Best λ	Least Train Loss	Least Test Loss	Search Time
l_1 norm	1st order	10.5263	4.3777	4.7051	3mins 21s
l_1 norm	2nd order	2.6316	10.2687	10.0671	4mins 10s
l_1 norm	3rd order	2.6316	13.3525	13.8286	4mins 29s
l_2 norm	1st order	0	25.068	24.8718	2mins 49s
l_2 norm	2nd order	2.6316	22.9727	25.117	3mins 2s
l_2 norm	3rd order	2.6316	21.8773	23.8416	3mins 6s

with zero mean and a randomly generated covariance matrix with noise levels 0.01,0.05 and 0.1.
(ii) piecewise constant signals(range 10 to 100) with noise drawn from a random normal distribution with zero mean and standard deviation (noise levels) 1,5 and 10.
1D total variation denoising [7] is a popular approach to solve such problems. It attempts to smoothen some noisy observed signal x by solving the following optimization problem :

$$\underset{y}{\text{argmin}} \frac{1}{2}\|x-y\|_2^2 + \lambda\|Dy\|_1 \tag{5}$$

where x is the noisy input signal, y is the denoised output signal, D is the differential operator which might be of first,second or third order. The value of λ determines the extent of smoothening of the output signal. The Python library for constructing differentiable convex optimization layers - cvxpylayers [1], was used in our work to solve the convex optimization problem given in equation 5. CvxpyLayer was used to solve equation 5 with first, second and third order differencing operations and l_1 and l_2 norms of the output signal difference to find optimum parameters for the two datasets used.
600 noisy signals were generated for each dataset(cosine signals with 0.1 noise, piecewise constant signals with 5 noise) and were split up into 540 training samples and 60 test samples. Each signal had 100 features or data points. Grid search was performed for λ between 0-50 to find optimum λ which gave least mean squared error (MSE) loss on the train set. Results are summarized in **Table** 1 and **Table** 2 which includes best values of λ found for each case (which performed best denoising of the training noisy signals) and the time it took to find the best λ by grid search.

From **Table** 1 and **Table** 2, it is evident that l_1 norm with 3rd order differencing yields best results for cosine signals while l_1 norm with 1st order differencing yields best results for the piecewise constant signals. The first order differencing operation D can be expressed in matrix form with rows as $D_i = e_i - e_{i+1}$. Penalizing the l_1 norm of the signal difference encourages this difference to be sparse, i.e. the number of changepoints of the signal to be small, hence approximating the input signal by a roughly piecewise constant function.

Next, we tried incorporating this differentiable optimization layer as the second layer of a neural network with the first layer given by a fully connected layer(without any activation) with 100 input features and 100 output features. An optimum value of λ which gave least train MSE loss was chosen by performing grid search between 0-50. l_1 norm with 3rd order differencing was used for cosine signals while l_1 norm with 1st order differencing was used for piecewise constant signals. There were 540 training samples and 60 test samples. Training and testing was done in batches with batch size taken as 10 for each. The learning rate was taken as 1e-4 and the weights and the biases of the FC layer were optimised with Adam optimiser. Training was performed for 100 epochs. Experiments were repeated for all noise levels. Mean squared error (MSE) was chosen as the performance metric. Results are given in **Table** 3 and **Table** 4.
It is evident from **Figure** 1 and **Figure** 2 that adding a CvxpyLayer after the fully connected layer greatly improves the accuracy of the predictions/denoised signal outputs, though it adds a significant bottleneck in training time.

The FC layer alone performs significant denoising in case of the cosine signals, but fails to acheive the required smoothness in case of the piecewise constant signals. It is interesting to note that the weights and biases of the FC layer get optimised after training such that the weights become a diagonally dominant matrix and the biases become a vector centred around 0.

Table 3: Performance assessment of FC layer + CvxpyLayer for denoising cosine signals

Signal	Model	Train Loss	Test Loss	Training Time
Cosine signals with 0.1 noise	FC layer	0.0138	0.0187	16s
Cosine signals with 0.1 noise	FC layer+CvxpyLayer	0.0095	0.0127	30mins 51s
Cosine signals with 0.05 noise	FC layer	0.0065	0.0080	12s
Cosine signals with 0.05 noise	FC layer+CvxpyLayer	0.0038	0.0048	29mins 31s
Cosine signals with 0.01 noise	FC layer	0.0023	0.0024	11s
Cosine signals with 0.01 noise	FC layer+CvxpyLayer	0.0009	0.0009	25mins 16s

Table 4: Performance assessment of FC layer + CvxpyLayer for denoising piecewise constant signals

Signal	Model	Train Loss	Test Loss	Training Time
Piecewise signals with 10 noise	FC layer	73.7327	92.0652	12s
Piecewise signals with 10 noise	FC layer+CvxpyLayer	55.6828	75.6486	38mins 37s
Piecewise signals with 5 noise	FC layer	53.8459	69.4389	12s
Piecewise signals with 5 noise	FC layer+CvxpyLayer	44.3047	58.3109	37mins 48s
Piecewise signals with 1 noise	FC layer	47.00489	60.7361	11s
Piecewise signals with 1 noise	FC layer+CvxpyLayer	41.2010	53.7937	20mins 48s

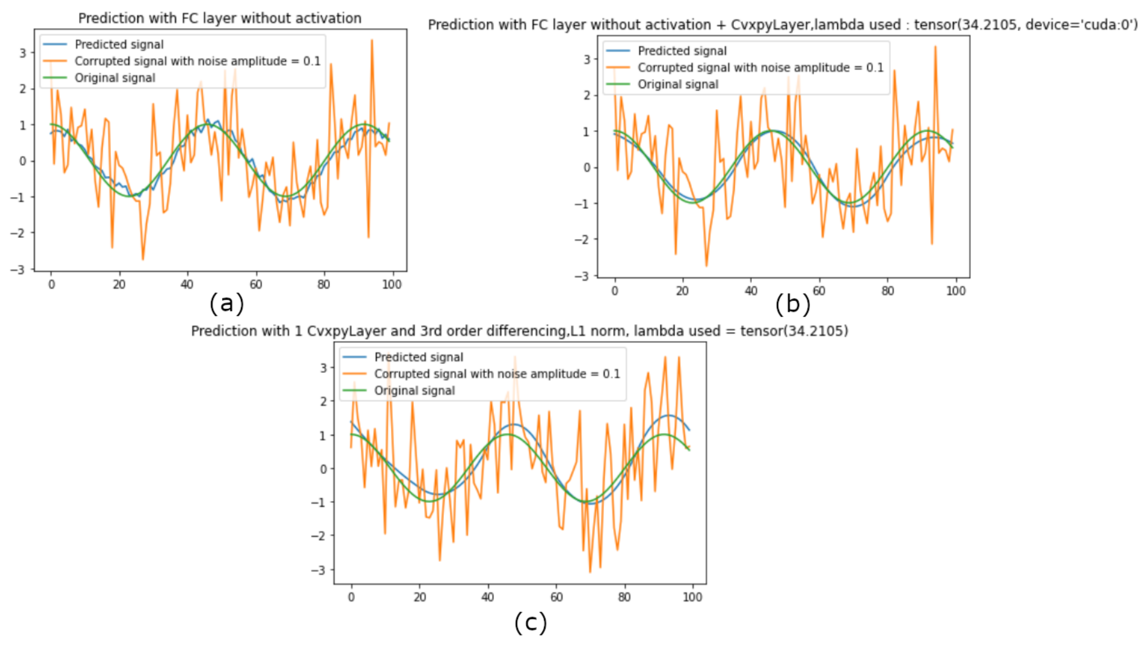

Figure 1: **Predictions for a cosine signal with 0.1 noise with (a)FC layer (b)FC layer + CvxpyLayer (c)CvxpyLayer**

3.2 Learning consistent differential operators

In this section, we are interested in learning consistent differential operators given the function values on a grid. In specific we are interested are computing the differential operators given the function values on the stencil $\mathcal{S}(x_p) = \{x_{p-m}, ..., x_p, ..., x_{p+m}\}$, i.e.

$$\left.\frac{\partial^l f}{\partial x^l}\right|_{x_p} = \sum_{x_q \in \mathcal{S}(x_p)} \xi_q f(x_q) + \mathcal{O}(\Delta x^r), \tag{6}$$

where l is the derivative order, and r is the approximation accuracy. For instance, for $l = 0$ and $r = 2$, the above Eq. 6 can be used to compute the function value at the midpoint for $m = 1$, i.e.

$$f(x_p) = \frac{f(x_p + \Delta x) + f(x_p - \Delta x)}{2} + \mathcal{O}(\Delta x^2),$$

where $\xi_{x_p+\Delta x} = \xi_{x_p-\Delta x} = 0.5$. Similarly, for $l = 2$ and $r = 2$, the weighted rule is given as,

$$\left.\frac{\partial^2 f}{\partial x^2}\right|_{x_p} = \frac{f(x_p + \Delta x) - 2f(x_p) + f(x_p - \Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2).$$

where $\xi_{x_p+\Delta x} = \xi_{x_p-\Delta x} = \frac{1}{\Delta x^2}$ and $\xi_{x_p} = \frac{-2}{\Delta x^2}$. It is important to note that derivative-order l and accuracy order r control the size of the stencil $|\mathcal{S}(x_p)|$. For more details check [4], [6].

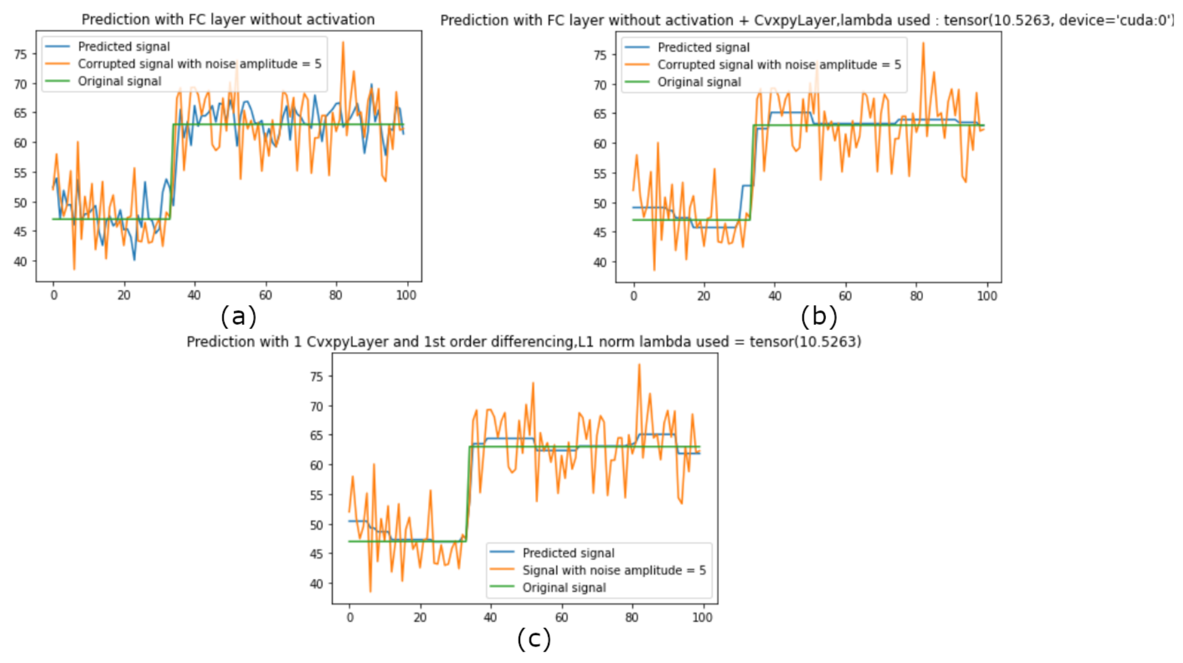

Figure 2: **Predictions for a piecewise constant signal with 5 noise with (a)FC layer (b)FC layer + CvxpyLayer (c)CvxpyLayer**

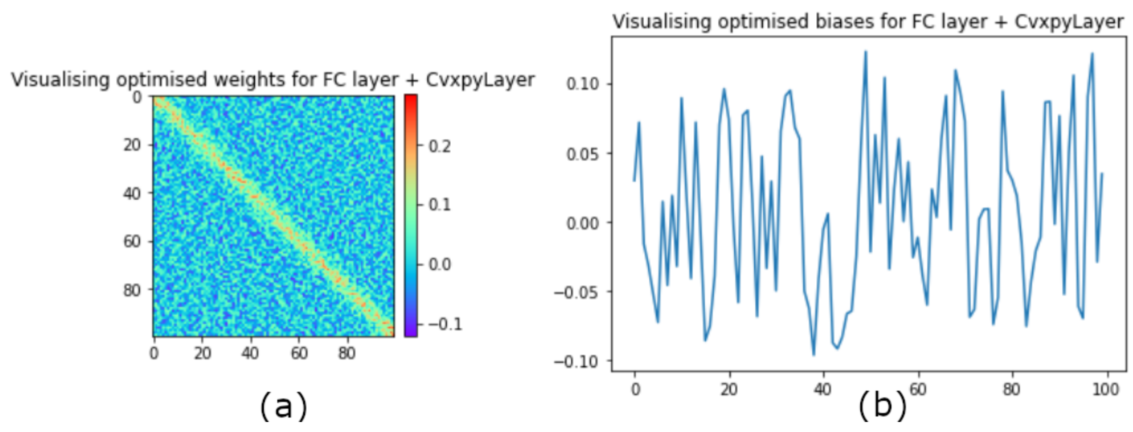

Figure 3: **Visualising optimised (a)weights and (b)biases of FC layer + CvxpyLayer for piecewise constant signals with 5 noise**

For the function values $f(x_i), i = 1, ..., N$ given on the grid, we are interested in approximating the derivative of order m, i.e.

$$\arg\min_{\boldsymbol{\xi}} \| \sum_{x_q \in \mathcal{N}(x_p)} \xi_q f(x_q) - \partial_x^m f(x_p) \|_2^2, \text{ subject to } \boldsymbol{D\xi} = \boldsymbol{\beta}. \tag{7}$$

The discrete moment vector $\boldsymbol{D} \in \boldsymbol{R}^{(k)\times(2m+1)}$ can be constructed to satisfy to the moment constraints as described in proposition 3.2.1. The coefficients $\boldsymbol{\xi} \in \boldsymbol{R}^{2m+1}$ can also be part of the differential physics program used for solving partial differential equations.

Proposition 3.2.1 (Discrete moment conditions) *For an r^{th} order approximation of the l^{th} derivative at point x_i, the stencil needs to satisfy the following moment conditions:*

$$\sum_{x_j \in S_m(x_i)} \xi_j \left(\Delta x_j\right)^k = \begin{cases} 0 & \alpha_{min} \le k \le l + r - 1, k \neq l \\ (-1)^k k! & k = l \\ < \infty & \textit{otherwise}, \end{cases},$$

where $k, l, r \in \boldsymbol{Z}_0^+$ and $\Delta x_j = (x_i - x_j)$. A stencil fulfilling these conditions must have size $|S_m| \ge |l + r - 1|$. The value of α_{min} is chosen to be one for even values of l, and zero for odd values of l.

The vandemonde matrix $\boldsymbol{D}$ is usually given as,

$$\boldsymbol{D} = \begin{pmatrix} 1 & \vdots & 1 & 1 & 1 & \vdots & 1 \\ -m\Delta x & \vdots & -\Delta x & 0 & \Delta x & \vdots & m\Delta x \\ (-m\Delta x)^2 & \vdots & (-\Delta x)^2 & 0 & (\Delta x)^2 & \vdots & (m\Delta x)^2 \\ \dots & \vdots & \dots & \vdots & \dots & \vdots & \dots \\ (-m\Delta x)^{k-1} & \vdots & (-\Delta x)^{k-1} & 0 & (\Delta x)^{k-1} & \vdots & (m\Delta x)^{k-1} \end{pmatrix}. \tag{8}$$

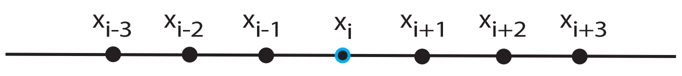

Figure 4: **Illustration of the stencil:** Schematic representation of the stencil with $m = 3$ and size $|\mathcal{S}_m| = 7$.

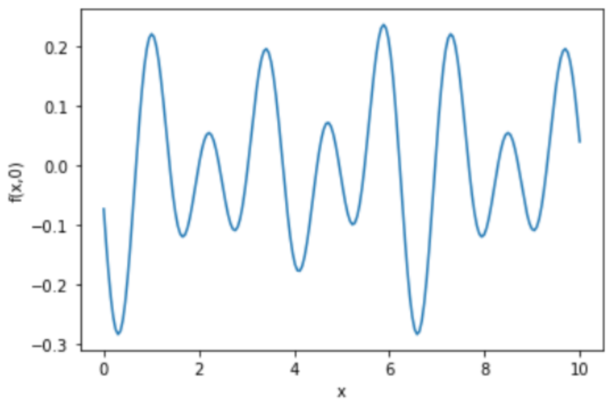

Figure 5: **Visualisation of the forcing term from the forced Burgers' equation in 1D**

For example, for $l = 0, r = 2$, we need $k = 2$ moment conditions to be fulfilled according to proposition 3.2.1 given in the matrix form $\boldsymbol{D\xi} = \boldsymbol{\beta}$ as follows,

$$\underbrace{\begin{pmatrix} 1 & 1 \\ -\Delta x & \Delta x \end{pmatrix}}_{\boldsymbol{D}} \underbrace{\begin{pmatrix} \xi_{+1} \\ \xi_{-1} \end{pmatrix}}_{\boldsymbol{\xi}} = \underbrace{\begin{pmatrix} 1 \\ 0 \end{pmatrix}}_{\boldsymbol{\beta}} \tag{9}$$

For $l = 1, r = 2$, we need $k = 3$ moment conditions to be fulfilled according to proposition 3.2.1 given in the matrix form $\boldsymbol{D\xi} = \boldsymbol{\beta}$ as follows,

$$\underbrace{\begin{pmatrix} 1 & 0 & 1 \\ -\Delta x & 0 & \Delta x \\ (-\Delta x)^2 & 0 & (\Delta x)^2 \end{pmatrix}}_{\boldsymbol{D}} \underbrace{\begin{pmatrix} \xi_{+1} \\ \xi_0 \\ \xi_{-1} \end{pmatrix}}_{\boldsymbol{\xi}} = \underbrace{\begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}}_{\boldsymbol{\beta}} \tag{10}$$

For $l = 2, r = 4$, we need $k = 3$ moment conditions to be fulfilled according to proposition 3.2.1:

$$\underbrace{\begin{pmatrix} -2\Delta x & -\Delta x & 0 & \Delta x & 2\Delta x \\ (-2\Delta x)^2 & (-\Delta x)^2 & 0 & (\Delta x)^2 & (2\Delta x)^2 \\ (-2\Delta x)^3 & (-\Delta x)^3 & 0 & (\Delta x)^3 & (2\Delta x)^3 \\ (-2\Delta x)^4 & (-\Delta x)^4 & 0 & (\Delta x)^4 & (2\Delta x)^4 \\ (-2\Delta x)^5 & (-\Delta x)^5 & 0 & (\Delta x)^5 & (2\Delta x)^5 \end{pmatrix}}_{\boldsymbol{D}} \underbrace{\begin{pmatrix} \xi_{+2} \\ \xi_{+1} \\ \xi_0 \\ \xi_{-1} \\ \xi_{-2} \end{pmatrix}}_{\boldsymbol{\xi}} = \underbrace{\begin{pmatrix} 0 \\ 2 \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{\boldsymbol{\beta}} \tag{11}$$

The discrete moment constraint for the implicit layer involves the optimization problem of the form,

$$\hat{\boldsymbol{z}} = \arg\min_{\boldsymbol{z}} \|\boldsymbol{Dz} - \boldsymbol{\beta}\|_2^2, \quad \text{subject to } \boldsymbol{z} = \boldsymbol{\xi} \tag{12}$$

where $\boldsymbol{\xi} \in R^{2m}$ is the output of the layer before the implicit layer, and $\hat{\boldsymbol{z}} \in R^{2m}$.

Experiments were performed to learn differential operators of order 0, which is essentially a function reconstruction problem :

$$f(x_p) = \sum_{x_q \in \mathcal{S}(x_p)} \xi_q f(x_q) + \mathcal{O}(\Delta x^r),$$

where r is 2 and 4 for stencil sizes 2 and 4 respectively.

The function used in these experiments was the forcing term from the forced Burgers' equation in 1D :

$$f(x,t) = \sum_{i=1}^{N} A_i sin(\omega_i t + 2\pi l_i x/L + \phi_i) \tag{13}$$

with each parameter drawn independently and uniformly at random from its respective range [6] : A = [-0,1,0.1], ω = [-0.4,0.4], ϕ = [0,2π] and N = 20. L is set to 2π and l = 2,3,4,5 .f(x,0) at t=0 was simulated to obtain a 1D function (**Figure** 5). f(x,0) was simulated at 200 points from x = 0-10 to form the dataset.

Stencil sizes of 2 and 4 respectively, were used for reconstructing the function or learning the differential operator of order 0. For stencil size 2, (l = 0, r = 2 case), the discrete moment conditions

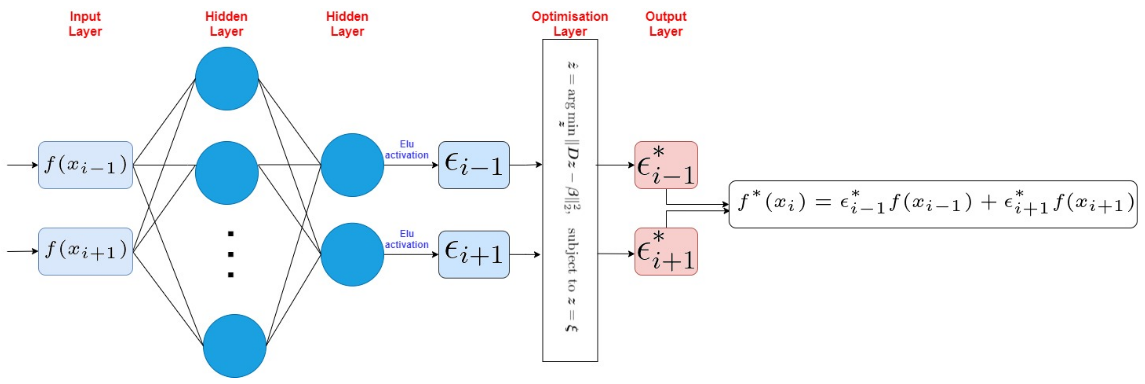

Figure 6: **Network used for function reconstruction with stencil size 2**

Table 5: Performance assessment of function reconstruction with stencil size 2 and 50-50 train-test split for different network architectures

FC network architecture	Model	Train Loss	Test Loss	Training Time
(2,8)->(8,2)->elu	Without moment constraints	0.0021768	0.0029827	1.3s
	With moment constraints	0.0009303	0.0013417	156.5s
(2,16)->(16,2)->elu	Without moment constraints	0.0011914	0.0015768	1.27s
	With moment constraints	0.0004647	0.0006396	153s

are outlined above in Eq. 9. For stencil size 4 (l = 0, r = 4), the discrete moment conditions are as follows :

$$\underbrace{\begin{pmatrix} 1 & 1 & 1 & 1 \\ -2\Delta x & -\Delta x & \Delta x & 2\Delta x \\ (-2\Delta x)^2 & (-\Delta x)^2 & (\Delta x)^2 & (2\Delta x)^2 \\ (-2\Delta x)^3 & (-\Delta x)^3 & (\Delta x)^3 & (2\Delta x)^3 \end{pmatrix}}_{\boldsymbol{D}} \underbrace{\begin{pmatrix} \xi_{+2} \\ \xi_{+1} \\ \xi_{-1} \\ \xi_{-2} \end{pmatrix}}_{\boldsymbol{\xi}} = \underbrace{\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{\boldsymbol{\beta}} \tag{14}$$

A MLP network with two fully connected linear layers activated by elu function, was used to learn the coefficients for the 0th order differential operator, with and without a Cvxpy optimization layer to satisfy the moment constraints (**Figure** 6).

The dataset of 200 points was divided into train and test parts. The learning rate was chosen as 1e-3 and the weights and biases of the fully connected layers were optimised by Adam optimiser. The network was trained for 100 epochs. The experiments were repeated for different network architectures and different train-test splits for stencil sizes 2 and 4 respectively. The results are summarized in **Table** 5 and **Table** 6.

It is evident from the tables that in all cases, adding an optimisation layer to satisfy the discrete moment conditions of the coefficients increases the accuracy of function reconstruction. **Figure** 7 shows the reconstructed function for stencil size 4 and 20-80 train-test split with (4,16)->(16,4)->elu architecture. The predicted coefficients for the same are shown in **Figure** 8. It is clear from the figure that adding the optimisation layer helps predict coefficients that obey the discrete moment conditions such as their sum adding up to 1, which results in a more accurate function reconstruction. The figure also indicates that it is harder to predict the coefficients at the peaks of the sinosoidal function, thus making function reconstruction difficult in those regions.

References

[1] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, 2019.

[2] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural

Table 6: Performance assessment of function reconstruction with stencil size 4, (4,16)->(16,4)->elu architecture for different train-test splits

Train-test split	Model	Train Loss	Test Loss	Training Time
50-50	Without moment constraints	0.0006184	0.0008860	1.33s
	With moment constraints	0.0001178	0.0001417	153s
20-80	Without moment constraints	0.0015863	0.0012400	1.35s
	With moment constraints	0.0003289	0.0002201	133.8

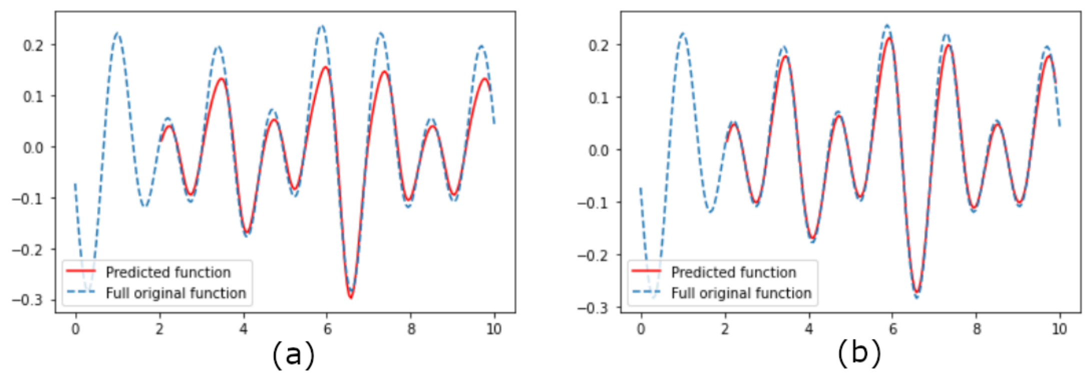

Figure 7: **Function reconstruction with stencil size 4, 20-80 train-test split and (4,16)->(16,4)->elu architecture (a) without moment constraints (b) with moment constraints**

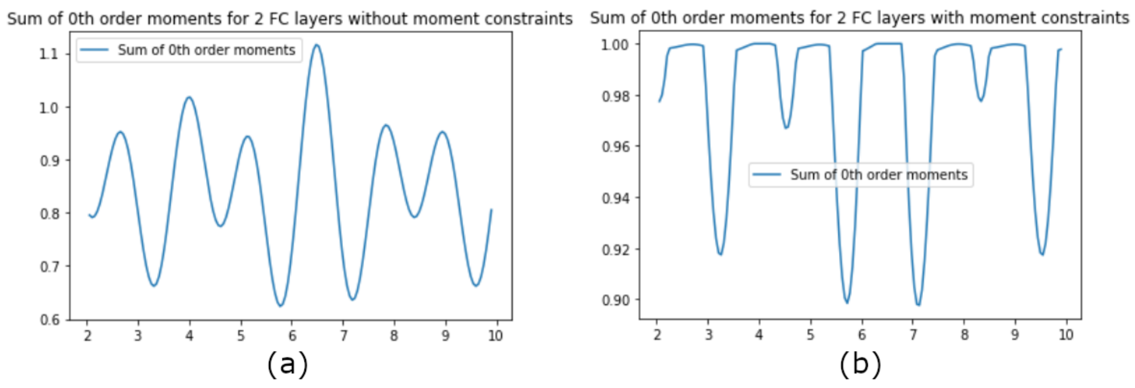

Figure 8: **Sum of predicted coefficients for function reconstruction with stencil size 4, 20-80 train-test split and (4,16)->(16,4)->elu architecture (a) without moment constraints (b) with moment constraints**

networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR, 2017.

[3] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[4] George C Bourantas, Bevan L Cheeseman, Rajesh Ramaswamy, and Ivo F Sbalzarini. Using dc pse operator discretization in eulerian meshless collocation methods improves their robustness in complex geometries. *Computers & Fluids*, 136:285–300, 2016.

[5] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

[6] Suryanarayana Maddu, Dominik Sturm, Bevan L Cheeseman, Christian L Müller, and Ivo F Sbalzarini. Stencil-net: Data-driven solution-adaptive discretization of partial differential equations. *arXiv preprint arXiv:2101.06182*, 2021.

[7] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.

[8] David Duvenaud Zico Kolter and Matt Johnson. Deep implicit layers - neural odes, deep equilibirum models, and beyond. `http://implicit-layers-tutorial.org`, 2020.