UNIVERSITY OF CALIFORNIA

Santa Barbara

Convolutional Neural Networks in Learning Fokker-Planck Equations

A Thesis submitted in partial satisfaction of the
requirements for the degree of Master of Arts
in Applied Mathematics

by

Andrew Gracyk

June 2021

The thesis of Andrew Gracyk is approved.

Katy Craig

Xu Yang

Paul Atzberger, Committee Chair

May 2021

ABSTRACT

Convolutional Neural Networks in Learning Fokker-Planck Equations

by

Andrew Gracyk

We discretize spatial domains into lattices. We provide the multivariate Fokker-Planck partial differential equation and its numerical solutions. We establish our convolutional neural network details, aiming to train these networks on our Fokker-Planck data with the goal of recovering Fokker-Planck coefficients. We break up our objective into different cases. For each case, we discuss results. First, we consider the simplified diffusion equation, then the advection-diffusion equation, where our networks learn the differential operators. We consider long-time integration methods. Finally, we consider finite difference and finite volume methods.

1 Introduction

The Fokker-Planck equation is a partial differential equation that describes the time evolution of a multivariate probability density function (PDF) subject to an initial condition as well as drift and diffusion influences. It is our aim to extract unknown drift and diffusion coefficients given data of solutions to this equation. In order to do this, there are machine learning techniques we may employ.

Convolutional neural networks (CNNs) are a machine learning mechanism frequently used in classification. A lattice full of values represents an image, where a value typically corresponds to a pixel. The goal of such a CNN is to learn a mapping from this lattice to a single value, and this single value is representative of some classification. For example, if we are training on data to learn classification between two types of images, a 0 may represent one class of images and 1 the other; however, CNNs go beyond classification. Instead of learning mappings between images to the set $\mathbb{N}$, CNNs can be used to learn mappings from one image to another. Backpropagation can be performed in this mapping, and trainable kernel parameters $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k$, with its elements the weights, can be learned in this mapping to minimize a loss function. This loss function is a penalization of how incorrect a classification, or in this case a predicted image, is. Naturally, this is something we would like to minimize, as we strive to make predictions as accurately as can be.

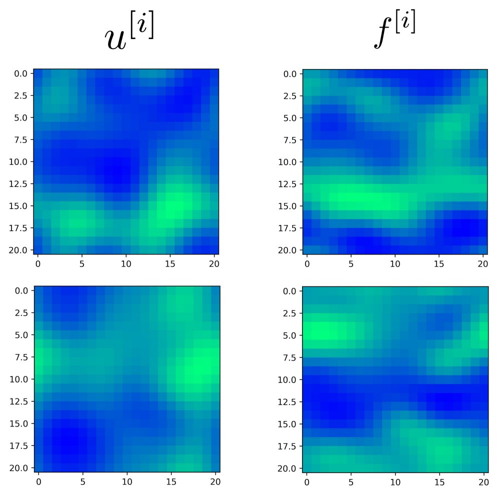

Figure 1. Above are examples of discretized functions turned into images. Training data for our CNNs will be similar. In this particular case, the figures on the right-hand side $f^{[i]}$ are the Laplacian Δ of the figures on the left $u^{[i]}$.

We can turn solutions of the Fokker-Planck equation into images by discretizing a spatial domain in $\mathbb{R}^2$, and evaluating Fokker-Planck equations at times $t_i = i\Delta t, i \in \mathbb{N} \cup \{0\}$ along this domain.

Such discretized solutions can be generated with a variety of numerical methods, which we will elaborate later. With this data, we can use CNNs to learn time-evolution of the PDE, learning a direct mapping between one discretized solution to the other, or to learn the differential operator. In particular, we consider discretized domain

$$\Omega = \Big\{(-mh, nh) \in \mathbb{R}^2 : h \in \mathbb{R}, m, n \in \{-\lambda, \dots, -1, 0, 1, \dots, \lambda\}\Big\}. \tag{1}$$

We can define the continuous PDFs that solve the Fokker-Planck equation over the "closure" of this domain, being the square in $\mathbb{R}^2$ that occupies the space covered by Ω, by

$$\overline{\Omega} = [-\lambda h, \lambda h] \times [-\lambda h, \lambda h]. \tag{2}$$

Similarly, define the boundary $\partial\Omega$ of Ω as the discretized points along the edges, being

$$\partial\Omega = \bigcup_{\ell \in \{-\lambda, \lambda\}} \bigcup_{n \in \{-\lambda, \dots, -1, 0, 1, \dots, \lambda\}} \Big((\ell h, nh) \cup (nh, \ell h)\Big) \tag{3}$$

for point $\boldsymbol{X} = (a, b) \in \mathbb{R}^2$. The mesh Ω is equispaced, meaning $\Delta X_1 = \Delta X_2 = \Delta X = h$. These discretized Fokker-Planck solutions will serve as valuable training data for our CNNs.

We are dealing solely with multivariate PDFs here, as only these are solutions to the Fokker-Planck equation. Such means our solutions ρ will be subject to the constraint

$$\iint \dots \int_{\mathbb{R}^n} \rho(\boldsymbol{X}, t_i | \rho_0) d\boldsymbol{X} = 1 \tag{4}$$

where $\rho(\boldsymbol{X}, t_i | \rho_0)$ is a Fokker-Planck solution evaluated at spatial location $\boldsymbol{X}$ and juncture t_i, subject to initial condition ρ_0. In particular, in the context of our problem, as we restrict the domain of our continuous PDFs to $\overline{\Omega}$, we consider

$$\iint_{\overline{\Omega}} \rho(\boldsymbol{X}, t_i | \rho_0) d\boldsymbol{X} \leq 1. \tag{5}$$

where equality is nearly attained, as the PDFs approach 0 along the $\partial\overline{\Omega}$. These conditions will not prove to be imperative in learning drift and diffusion values, but they do provide a means to provide correct initial conditions.

Our CNNs that learn either the time-evolution of the solutions or the differential operator will not tell us everything we need to know explicitly. There is still a matter of deducing drift and diffusion coefficients. We may introduce a second CNN, where this CNN does not train on Fokker-Planck data, but instead, trains on the kernels of the first CNN. Fokker-Planck solutions can be generated by iterating drift and diffusion coefficient sequences, and we may train our first CNN in some manner we prescribe. By turning these first CNN kernels into a new data set, our second CNN can predict drift and diffusion values by mapping the weights to these values. The kernels of the first CNN should be contingent on choice of drift and diffusion values, meaning these kernels are characteristic of the coefficient choices.

After training the pair of CNNs on predetermined data, we can again train our first CNN on new Fokker-Planck data with unknown drift and diffusion coefficients to generate new CNN kernels. We can feed these kernels into the second CNN to predict these coefficients.

2 The Fokker-Planck equation

In this section, we provide the Fokker-Planck equation and its differential operator. This is the key equation we will be considering. It provides the description on how a multivariate probability density function behaves as it evolves in times. The drift and diffusion coefficients are what we ultimately hope to deduce given data that conforms to this equation, where these coefficients are intended to be omitted from the information that is given.

The Fokker-Planck equation coheres to the stochastic differential equation

$$d\boldsymbol{X}_t = \boldsymbol{\mu}(\boldsymbol{X}_t, t)dt + \boldsymbol{\sigma}(\boldsymbol{X}_t, t)d\boldsymbol{W}_t \tag{6}$$

where $\boldsymbol{X}_t$ is an N-dimensional vector of random variables in a sequence in time. $\boldsymbol{\mu}(\boldsymbol{X}_t, t)$ is also an N-dimensional random vector known as the drift vector. $\boldsymbol{\sigma}(\boldsymbol{X}_t, t)$ is an $N \times M$ matrix of diffusion values, and $\boldsymbol{W}_t$ is an M-dimensional Wiener process.

In particular, the general N-dimensional Fokker-Planck equation [2] is given by

$$\frac{\partial \rho(\mathbf{X}, t|\rho_0)}{\partial t} = -\sum_{i=1}^{N} \frac{\partial}{\partial X_i}[\mu_i(\mathbf{X}, t)\rho(\mathbf{X}, t|\rho_0)] + \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{\partial^2}{\partial X_i \partial X_j}[D_{ij}(\mathbf{X}, t)\rho(\mathbf{X}, t|\rho_0)] \tag{7}$$

and the diffusion tensor is an $N \times N$ matrix.

When we discretize solutions to equation (7), we do so over domain Ω. We use the following notation to denote Fokker-Planck numerical solutions evaluated at the lattice points in this domain at time t_i:

$$\Phi_i \approx \rho(\Omega, t_i|\rho_0) = \rho(\boldsymbol{X}, t_i|\rho_0)\Big|_{\boldsymbol{X}\in\Omega}. \tag{8}$$

These form new lattices that are discretized solutions to equation (7).

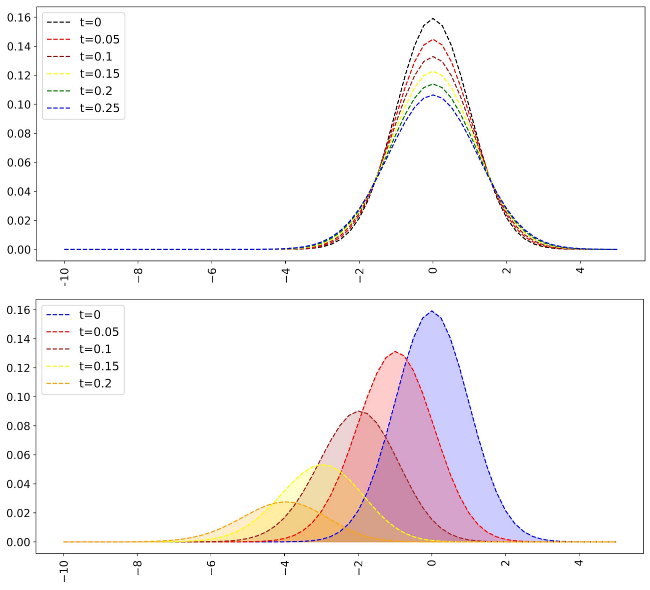

Figure 2. Above we have cross-sections of discretized Fokker-Planck solutions over Ω. It appears that the integration property is not satisfied in the second image, but this is because the drift values are forcing the local maxima of the solutions to shift.

We will begin our investigation by reducing equation (7) to the diffusion and advection-diffusion cases, which are Fokker-Planck equations under the right circumstances. We can also attempt to learn a direct mapping of this equation, using a CNN to map Φ_i to Φ_{i+1}. We will hold $N = 2$ to simplify our investigation. In general, with constant coefficients, the drift and diffusion matrices are given by

$$\boldsymbol{\mu} = (\mu_1, \mu_2)^T, \qquad \boldsymbol{D} = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix}. \tag{9}$$

We will hold $\sigma_{12} = \sigma_{21} = 0$ throughout the remainder of our investigation, but one can extend the ideas of what is done to cases when these coefficients are nonzero. A more sophisticated numerical method would be required to generate data, and ability to determine coefficients from data may become more difficult.

We also extract the corresponding differential operator to equation (7), which we will call $\mathcal{D}$. Such an operator means that the Fokker-Planck equation can be rewritten as

$$\frac{\partial \rho(\boldsymbol{X}, t|\rho_0)}{\partial t} = \mathcal{D}\big[\rho(\boldsymbol{X}, t|\rho_0)\big]. \tag{10}$$

A large portion of what we do will be attempting to deduce the right-hand side of equation (10). We will call a CNN that learns this right-hand side one that learns the differential operator. When we discretize time, combined with the spatial discretization of Ω, our PDE formulation becomes

$$\frac{\Phi_{i+1} - \Phi_i}{\Delta t} = \mathcal{D}\big[\{\Phi_i\}_{i \in \mathcal{K}}\big], \tag{11}$$

where the right-hand side is a numerical approximation of the differential operator applied to the lattices. Note that the corresponding differential operator $\mathcal{D}$ is given by

$$\mathcal{D} = -\sum_{i=1}^{2} \mu_i \frac{\partial}{\partial X_i} + \sum_{j=1}^{2} \sigma_j \frac{\partial^2}{\partial X_j^2}. \tag{12}$$

It is also necessary to establish initial conditions. We consider two, being the multivariate normal density function, and the multivariate t-distribution. We only consider distributions with $\text{supp}(\rho) = \mathbb{R}^2 \times \mathbb{R}$, so we can appropriately create square lattices $\{\Phi_k\}_{k \in \mathcal{K}}$ over Ω. For example, the Dirichlet distribution $\rho(\boldsymbol{X}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} X_i^{\alpha_i - 1}$ lacks desired support and cannot necessarily be discretized over the entirety of Ω.

We denote the two initial functions as follows:

$$\phi_0 = \phi(\boldsymbol{X}, t_0 = 0; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) = \frac{1}{2\pi \cdot |\boldsymbol{\Sigma_1}|^{1/2}} \exp\Big\{ -\frac{1}{2}(\boldsymbol{X} - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma_1}^{-1}(\boldsymbol{X} - \boldsymbol{\mu}_1)\Big\}, \tag{13}$$

where $\boldsymbol{\mu}_1$ is a vector of means, and $\boldsymbol{\Sigma}_1$ is a variance-covariance matrix; and

$$\psi_0 = \psi(\boldsymbol{X}, t_0 = 0; \nu, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) = \frac{\Gamma[(\nu+2)/2]}{\Gamma(\nu/2)\nu\pi|\boldsymbol{\Sigma}_2|^{1/2}} \Big[1 + \frac{1}{\nu}(\boldsymbol{X} - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}_2^{-1}(\boldsymbol{X} - \boldsymbol{\mu}_2)\Big]^{-(\nu+2)/2}. \tag{14}$$

Coefficient ν is predetermined, as well are $\boldsymbol{\mu}_2$ and $\boldsymbol{\Sigma}_2$.

3 Designing our convolutional neural networks

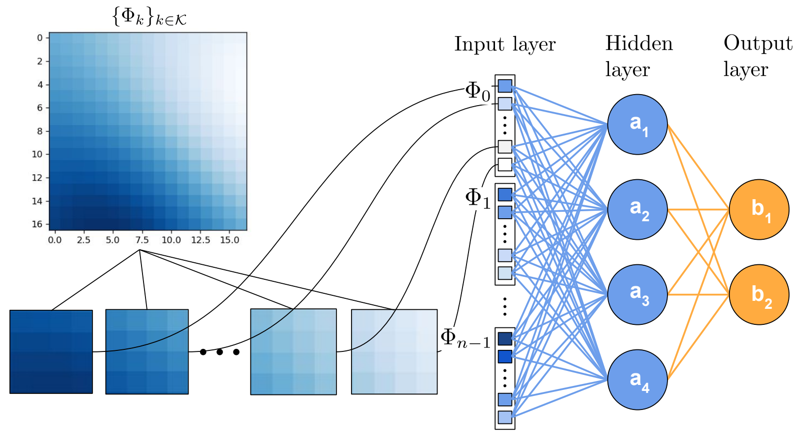

Figure 3. We have a depiction of the CNN $\mathcal{C}^1$. A Fokker-Planck lattice is divided into subgrids that are ultimately mapped to a new lattice, which can be either the differential operator lattice or the next discretized Fokker-Planck solution in time.

We will consider two different CNNs, each with a different purpose. The first CNN we consider has the aim of learning a function [8]

$$\mathcal{C}^1 : \boldsymbol{T}_0 \times \boldsymbol{\theta}_1^1 \times \ldots \times \boldsymbol{\theta}_{k_1}^1 \to \boldsymbol{T}_1, \tag{15}$$

that takes arguments of a PyTorch tensor $\boldsymbol{T}_0$ and trainable kernel hyperparameters $\boldsymbol{\theta}_1^1, \ldots, \boldsymbol{\theta}_{k_1}^1$. PyTorch tensor $\boldsymbol{T}_0$, which has dimension $m \times 1 \times (2\lambda + 1) \times (2\lambda + 1)$, is mapped to a new lattice $\boldsymbol{T}_1$ of the same dimension. An additional tensor will need to be taken as argument in our loss function, however that function may be defined. This tensor will typically be the lattices in $\boldsymbol{T}_0$ but indexed one higher.

This particular type of CNN has the aim of either learning a Fokker-Planck differential operator or a direct mapping from one Fokker-Planck solution to the next in time. The kernel parameters are crucial. The weights that belong to these kernels are contingent on choice of drift and diffusion choices, and they will be determined by training our CNN. These kernels can be concatenated into a new dataset, which will form the data we use for our second type of CNN $\mathcal{C}^2$. As we will later see, we restrict the number of kernels of $\mathcal{C}^1$ to be either one or two, depending on our strategy of recovering the coefficients.

The coefficients of our training data are from predetermined sequences that we choose and held within ranges $\sigma_{j,\min} \leq \sigma_{j,i} \leq \sigma_{j,\max}$, $\mu_{k,\min} \leq \mu_{k,i} \leq \mu_{k,\max}$. The first subscript in $\sigma_{j,i}, \mu_{k,i}$ refers to any of $\sigma_{11}, \sigma_{22}, \mu_1, \mu_2$ from the drift and diffusion matrices in the previous section. For example, $\sigma_{1,i}$ refers to coefficient σ_{11}. The second index is the index within the sequence in which it was generated. We must also produce a test data set, whose coefficients differ from those in the training set. We train upon this "test" data set in the same manner as the training set in order produce the appropriate kernels.

Our second type of CNN has the aim of learning a function

$$\mathcal{C}^2 : \mathbf{\Theta}_0 \times \boldsymbol{\theta}_1^2 \times \ldots \times \boldsymbol{\theta}_{k_2}^2 \to \mathbf{\Sigma}_0 \tag{16}$$

where $\mathbf{\Theta}_0$ is the dataset concatenation of the kernels created from $\mathcal{C}^1$, and $\mathbf{\Sigma}_0$ is some tensor that is a prediction to any collection of drift or diffusion values.

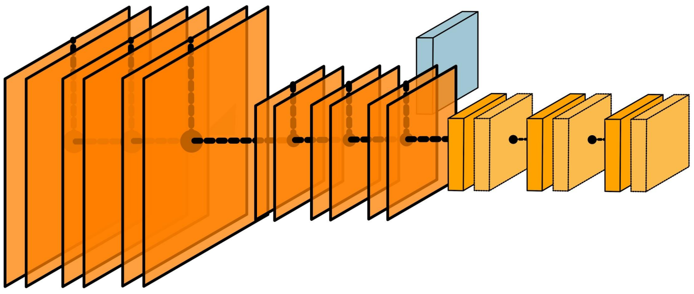

Figure 4. Above is a generalized illustration of CNN architecture. Image size is typically compressed as the data is processed through convolution layers, which is emphasized in this image. The number of images produced per layer is dependent on architecture choices. More channels mean more images produced.

This CNN will have a more sophisticated architecture than $\mathcal{C}^1$, and we will discuss this in future sections. We train this CNN upon the kernels produced from $\mathcal{C}^1$ and the predetermined test drift and diffusion coefficients. $\mathcal{C}^2$ is then applied to the test data set to predict the coefficients. This test data set acts as a true test set, unlike for $\mathcal{C}^1$, because there will be no training on this. Instead the CNN will be directly applied to the input data in order to produce the expected coefficients.

We use the following notation of a CNN mapping a lattice element of an input tensor to the respective lattice element of the output tensor: $\mathcal{C}[\Phi_i]$.

4 Learning the diffusion equation: a special case of the Fokker-Planck

We start by considering the simplified diffusion equation [7]

$$\begin{cases} \dfrac{\partial \rho(\boldsymbol{X},t)}{\partial t} = \nabla \cdot [D(\rho,\boldsymbol{X})\nabla\rho(\boldsymbol{X},t)] = D\Delta\rho(\boldsymbol{X},t), \qquad \boldsymbol{X} \in \overline{\Omega}, t \in [0,T] \\ \rho(\boldsymbol{X}, t_0 = 0) = \Psi_0 \end{cases} \tag{17}$$

where diffusion function $D(\rho, \boldsymbol{X}) = D$ is constant here. Ψ_0 is an initial condition, either ϕ_0 or ψ_0. The diffusion equation is a special case of the Fokker-Planck equation under the circumstance $\nabla_{\boldsymbol{X}} D(\rho, \boldsymbol{X}) = 0$, which is certainly satisfied here. It is our aim to deduce diffusion coefficient D using CNNs given snapshots of data $\{\Phi_k\}_{k\in\mathcal{K}}$. Time is also discretized by $t_i = i\Delta t$.

Our CNN $\mathcal{C}_D^1$ can be represented by the operator $\boldsymbol{R}$ that is equivalent to the kernel $\boldsymbol{\theta} = \boldsymbol{\theta}^1$, where only one kernel is used in our CNN. This is certainly sufficient to learn the Laplacian Δ, as it is known the Laplacian can be computed using a finite difference stencil encapsulated by the weights of the kernel. The Laplacian can be represented by the operator $\boldsymbol{L} = -\boldsymbol{R}^T\boldsymbol{R}$, which is negative semi-definite and constrains the hypothesis space to

$$\mathscr{H} = \left\{\boldsymbol{\theta} \;\middle|\; \boldsymbol{\theta} \in \mathbb{R}^{n\times n},\ \boldsymbol{x}^T(-\boldsymbol{\theta}^T\boldsymbol{\theta})\boldsymbol{x} \le 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^n \setminus \boldsymbol{0}\right\}. \tag{18}$$

The operator $\boldsymbol{R}$ may have a non-unique representation given the possibility of a unitary matrix $\boldsymbol{U}$ such that

$$\boldsymbol{L} = -(\boldsymbol{U}\boldsymbol{R})^T(\boldsymbol{U}\boldsymbol{R}) = -\boldsymbol{R}^T\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{R} = -\boldsymbol{R}^T\boldsymbol{I}\boldsymbol{R} = -\boldsymbol{R}^T\boldsymbol{R}. \tag{19}$$

To learn such an operator $\boldsymbol{L}$, we have a CNN $\mathcal{C}_D^1$ learn operator $\boldsymbol{R}$, in which case an additional CNN applies the same operator in the loss function but negative transpose, $\boldsymbol{R}^T$. The loss function trains $\mathcal{C}_D^1$. The weights of this new operator $\boldsymbol{L} = -\boldsymbol{R}^T\boldsymbol{R}$ are what forms our dataset for $\mathcal{C}_D^2$. Input data is $\{\Phi_k\}_{k\in\mathcal{K}}$, and the estimated Laplacian, $\{\Delta\Phi_k\}_{k\in\mathcal{K}}$, is our overall output goal. The data that is trained upon is produced using a predictor-corrector implicit Backward Euler analog for the Laplacian. Kernels of dimension 3×3 are generally sufficient in learning $\boldsymbol{R}$, hence $\boldsymbol{L}$, but a wider kernel can be under consideration and achieve equivalent, or potentially better, results. Allow us to demonstrate results with the following table.

Table 1: Diffusion equation

$\mathcal{C}_D^1$ **stencil size**	Δt	$\mathcal{C}_D^1$ ℓ_2**-loss**	$\mathcal{C}_D^2$ ℓ_2**-loss**	$\lvert\widehat{D} - D_{\text{test}}\rvert/D_{\text{test}}$
3×3	1e-4	0.9682	0.0009	0.0709
3×3	1e-3	0.2208	3.9173e-5	0.0082
3×3	5e-3	0.0344	5.5073e-5	0.0245
5×5	1e-4	0.9397	3.0797e-6	0.0181
5×5	1e-3	0.2328	9.5620e-5	0.0078
5×5	5e-3	0.0379	4.0656e-5	0.0386

Table 1. We train our first CNN $\mathcal{C}_D^1$ on Laplacian data where the diffusion coefficient D_i is iterated within a sequence $\{D_i\}_{i\in I}$. The kernels produced are merged into a new data set, on which we train CNN $\mathcal{C}_D^2$ mapping such data to the diffusion values. We provide a test data set with test diffusion coefficient D_{test}. The given $\mathcal{C}_D^1$ loss is on such a test set. Relative error is provided for the $\mathcal{C}_D^2$ estimation of this diffusion coefficient. No activation is used in either CNN. Initial condition $\Psi_0 = \phi_0$ is used.

$\mathcal{C}_D^1$ generally reached a convergence condition $|L_{i+2} - L_i| = |L_{i+3} - L_{i+1}| = 0$ for loss values $L_i, (E_{\max} - 3) \geq i \geq I$ in approximately $I = 3,000$ epochs, where $E_{\max}$ is the final epoch. This oscillating loss condition suggests the need for a lower learning rate, which we did apply, but this did not yield better loss and was not needed. $\mathcal{C}_D^2$ reached a sense of convergence in a higher number of epochs, upwards of $10,000$.

The loss functions considered in the CNNs $\mathcal{C}_D^1$ and $\mathcal{C}_D^2$ are ℓ_2 loss, which we will elaborate in section (6). The loss of $\mathcal{C}_D^1$ is the difference between $(\boldsymbol{\theta}^1)^T, (\boldsymbol{\theta}^1)$ repeated convolution with data and the data's estimated Laplacian. The loss of $\mathcal{C}_D^2$ is the difference between the predicted and true diffusion coefficients $\sum_i(\widehat{D}_i - D_i)^2$.

As demonstrated in the table, diffusion coefficient estimation is consistently under 8% error, down to less than 1% error. The architecture for $\mathcal{C}_D^2$ that achieved the best results was moderate in depth and moderate in node quantity. In particular, the above table was constructed with two convolution layers and 100 intermediary nodes. One could consider adding an overfitting parameter $\sum_{i=1}^{k_2} \operatorname{Tr}((\boldsymbol{\theta}_i^2)^T(\boldsymbol{\theta}_i^2))$ within the loss function when more data is considered. Lower amounts of data $\{\boldsymbol{\theta}^{1,m}\}_{m\in\mathcal{M}}$ were also found to be sufficient in learning diffusion coefficient approximations with $\mathcal{C}_D^2$, but more data was found to provide similar results. We constructed the above table using up to 50 kernels from $\mathcal{C}_D^1$. We will discuss data set size in more depth later in section (7).

5 Learning the advection-diffusion equation

The second Fokker-Planck equation we consider is the advection-diffusion equation

$$\begin{cases} \dfrac{\partial \rho(\boldsymbol{X},t)}{\partial t} = \nabla \cdot (D\nabla\rho(\boldsymbol{X},t)) - \nabla \cdot (\boldsymbol{v}(\boldsymbol{X},t)\rho(\boldsymbol{X},t)), \qquad \boldsymbol{X} \in \overline{\Omega}, t \in [0,T] \\ \rho(\boldsymbol{X}, t_0 = 0) = \Psi_0 \end{cases} \tag{20}$$

which is a Fokker-Planck equation when $\nabla_{\boldsymbol{X}}\boldsymbol{v}(\boldsymbol{X},t) = 0$, similar to the diffusion function condition we previously saw. To satisfy this condition, we hold vector field $\boldsymbol{v}(\boldsymbol{X},t)$ constant such that $\boldsymbol{v} = (v_1, v_2)$.

Our procedure for learning the coefficients is similar to the case with just the Laplacian term. We can represent the Laplacian by learning an operator $\boldsymbol{L} = -\boldsymbol{R}^T\boldsymbol{R}$ as we did in the previous case, and we may also learn an operator $\boldsymbol{A}$ for the advection term. Our numerical scheme becomes

$$\frac{\Phi_{i+1} - \Phi_i}{\Delta t} = \boldsymbol{L}\Phi_i + \boldsymbol{A}\Phi_i = (\boldsymbol{L} + \boldsymbol{A})\Phi_i. \tag{21}$$

Again, we gather data using a predictor-corrector Backward Euler scheme for the diffusion and advection terms to obtain $\{\Phi_k\}_{k\in\mathcal{K}}$. The diffusion and advection lattices can be stored independently, in which we will train to learn the operators.

Just as before, we constrain the regression to learn a skew-symmetric operator $\boldsymbol{A} = -\boldsymbol{A}^T$, which possesses nice properties and similarly constrains the hypothesis space like the previous case with the Laplacian. To learn a skew-symmetric operator, we may represent operator $\boldsymbol{A} = (1/2)(\boldsymbol{M} - \boldsymbol{M}^T)$ for arbitrary matrix $\boldsymbol{M}$, and our CNN learns $\boldsymbol{M}$. We may assign a new CNN with the same architecture, and we fix the stencil weights as the transpose $(\boldsymbol{\theta}^1)^T$ of the previous CNN stencil $\boldsymbol{\theta}^1$, hence effectively learning $\boldsymbol{M}^T$. The argument $(1/2)(\boldsymbol{M} - \boldsymbol{M}^T)$ is taken in the loss function and the operator $\boldsymbol{A}$ is obtained.

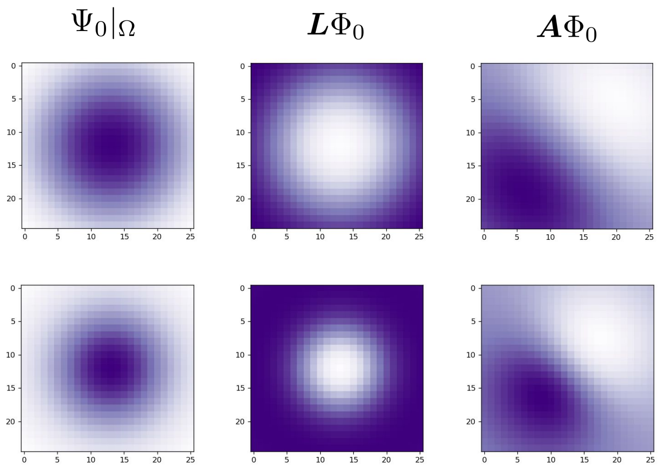

Figure 5. Above are sample lattices. Initial conditions ϕ_0 and ψ_0 are on the very left, with Laplacian and advection lattices on the center and right.

Allow us to demonstrate the capacity of learning just the advection data. We vary the advection coefficients v_1, v_2 in sequences to obtain operator data $\boldsymbol{A}$. We concatenate these operator weights into new data sets, and we train CNN $\mathcal{C}^2_{AD}$ to learn a mapping from the operator stencil weights to the coefficients. Diffusion coefficient D, hence operator $\boldsymbol{L}$, remains constant, and is irrelevant from the data of $\boldsymbol{A}$ in the sense that it is not directly used; however, it is indirectly used, as it is involved in the computation of $\{\Phi_k\}_{k\in\mathcal{K}}$. We list average relative error in the following table for certain cases.

Table 2: Advection-Diffusion equation, advection term

Parameter(s) varied	$\frac{1}{n}\sum_{i=1}^{n} \lvert\widehat{v}_i - v_{\mathbf{test},i}\rvert / v_{\mathbf{test},i}$
v_1	0.0134
v_2	0.0529
v_1, v_2	0.0109

Table 2. We vary either coefficient v_1, v_2 or both, and record the average relative error on a test data set that produces estimated coefficient values $\widehat{v}_i$. Initial condition ϕ_0 and 5×5 kernels are used.

As the table shows, our CNNs demonstrate high learning capacity for the advection coefficients, as the average relative error is around $1\% - 5\%$. Similar to the case of the Laplacian, if we are able to determine operator $\boldsymbol{A}$, we have the ability to recover $\boldsymbol{v}$ coefficients. Now, suppose we cannot determine operators $\boldsymbol{L}$ and $\boldsymbol{A}$ independently from each other, and the only data we possess is information cohering to $(\Phi_{i+1} - \Phi_i)/\Delta t = \boldsymbol{P}\Phi_i$, where $\boldsymbol{P}$ represents the differential operator

$\boldsymbol{P} = \boldsymbol{L} + \boldsymbol{A}$. Is it still possible to recover both drift (advection) and diffusion (Laplacian) coefficients? The following table illustrates our results.

Table 3. Advection-diffusion equation, both advection and diffusion terms

Coefficient(s) varied	**Avg. relative test error for** $\widehat{D}_i$	**Avg. relative test error for** $\widehat{v}_{1,i}$	**Avg. relative test error for** $\widehat{v}_{2,i}$
$D_i < 1$	**0.0145**	0.0070	0.0067
$v_{1,i} \leq 1$	0.0062	**0.0376**	0.0150
$-1 \leq v_{2,i} < 0$	0.0142	0.0277	**0.0581**
$v_{1,i} \leq 10$	0.0023	**0.0150**	0.0237
$-10 \leq v_{2,i} < 0$	0.0625	0.0663	**0.0337**
$D_i, v_{1,i} \leq 10$	**0.0372**	**0.0066**	0.0530
$-10 \leq D_i, v_{2,i} \leq 1$	**0.0572**	0.0857	**0.0276**
$-10 \leq v_{1,i}, v_{2,i} \leq 10$	0.0612	**0.0714**	**0.0648**
$\lvert D_i \rvert, \lvert v_{1,i} \rvert, \lvert v_{2,i} \rvert \leq 1$	**0.0618**	**0.0663**	**0.1196**
$\lvert D_i \rvert \leq 1, \lvert v_{1,i} \rvert, \lvert v_{2,i} \rvert \leq 5$	**0.0761**	**0.0153**	**0.0274**
$\lvert D_i \rvert \leq 1, \lvert v_{1,i} \rvert, \lvert v_{2,i} \rvert \leq 10$	**0.1043**	**0.0622**	**0.0701**

Table 3. Above is a table demonstrating the ability to recover the drift and diffusion coefficients from CNN kernels. The first two portions of the table rely on preexisting knowledge of one or two coefficients, as coefficients are fixed. Only some are variable. The third portion is when all three coefficients are iterated, suggesting the ability to recover all three coefficients from Fokker-Planck data. Relative error is recorded for test data sets in which the coefficients differ from taining data.

Training coefficients are iterated within sequences $\{D_i\}_{i \in I_k}, \{v_{1,i}\}_{i \in I_k}, \{v_{2,i}\}_{i \in I_k}$, with sequences changing on how many coefficients are iterated and the ranges. The values within these ranges are contained within the table. We vary the ranges to ensure that one operator, $\boldsymbol{L}$ or $\boldsymbol{A}$, does not dominate the other in terms of stencil weights when constructing $\boldsymbol{P}$. In certain cases, coefficients are held constant and only certain coefficients vary. Such illustrates the first and second parts of the table. The third case is the most interesting as it best illustrates the ability to recover all three coefficients given no information beforehand. Relative error is the greatest in this scenario ($< 12\%$), and it is the best in the case when only one coefficient is undetermined ($< 7\%$). This is still not terrible relative error and suggests that recovering all three coefficients is manageable. It is suspected that coefficient approximation is best determined if a particular range in which the coefficients may lie is used to construct the training data. For example, if the true coefficient $v_1 = 10$, and CNN data is trained upon $v_1 < 1$, relative error may be high.

Just as with learning the Laplacian, data set sizes in the above table range up to 50 kernels from $\mathcal{C}^1_{AD}$. Results using different data set sizes were similar, except more data was needed when more than one coefficient was varied to get equivalent results among all cases. Again, we will do an in-depth investigation on how data set size of the kernels from $\mathcal{C}^1_{AD}$ impacts accuracy when we discuss finite volume methods in section (7).

One problem with this method is that training data depends on $\boldsymbol{P} = \boldsymbol{L} + \boldsymbol{A}$, where $\boldsymbol{L}$ is negative semi-definite and $\boldsymbol{A}$ is skew-symmetric. If these exact operators cannot be determined, and only $\boldsymbol{P}$ can be determined without $\boldsymbol{L}$ and $\boldsymbol{A}$, we may not be able to recover the coefficients as effectively. We will later on discuss this case when the differential operator is unconstrained.

6 Long-time integration

In this section, our CNNs now learn a mapping directly from one Fokker-Planck discretization Φ_i to the next Φ_{i+1}, and we are no longer learning a differential operator. Our aim is for our CNN $\mathcal{C}^1_{LTI}$ to learn an operator $\boldsymbol{\Gamma}$ such that

$$\Phi_{i+1} = \boldsymbol{\Gamma}\Phi_i \tag{22}$$

where the stencil weights of $\boldsymbol{\Gamma}$ directly map lattice values at time t_i to t_{i+1} when convolution is taken over lattice subgrids.

Again, we use a predictor-corrector Backward Euler scheme for generating data. The full numerical method for finding data [2] [7] in our predictor step is given by

$$\begin{aligned}\frac{\Phi^{k,l}_{m+1} - \Phi^{k,l}_{m}}{\Delta t} = &-\mu_1 \frac{\Phi^{k+1,l}_{m} - \Phi^{k-1,l}_{m}}{2h} - \mu_2 \frac{\Phi^{k,l+1}_{m} - \Phi^{k,l-1}_{m}}{2h} &(23)\\ &+\sigma_1 \frac{\Phi^{k+1,l}_{m} - 2\Phi^{k,l}_{m} + \Phi^{k-1,l}_{m}}{h^2} + \sigma_2 \frac{\Phi^{k,l+1}_{m} - 2\Phi^{k,l}_{m} + \Phi^{k,l-1}_{m}}{h^2}, &(24)\end{aligned}$$

where $\Phi^{k,l}_i$ are the elements of lattice Φ_i at position (k,l) along Ω. We use the same numerical method for the diffusion and advection-diffusion equations but in simplified forms. Note when $\sigma_1 = \sigma_2$, we have the stencil for the five-point Laplacian Δ. The first two terms correspond to the advection terms, holding vector field coefficients $(v_1, v_2) = (\mu_1, \mu_2)$ in the previous section. Boundary conditions along $\partial\Omega$ are 0.

The individual loss function along with the overall optimization goal for $\mathcal{C}^1_{LTI}$ are the MSE-analog

$$\mathcal{L}^i_{LTI}(\Phi_i, \Phi_{i+1}|\boldsymbol{\theta}^1_1, \ldots, \boldsymbol{\theta}^1_k) = \gamma_{LTI} \sum_k \sum_l \left[(\boldsymbol{\Gamma}\Phi_i)^{k,l} - \Phi^{k,l}_{i+1}\right]^2 \tag{25}$$

$$= \gamma_{LTI} \left|\left|\mathcal{C}^1_{LTI}[\Phi_i] - \Phi_{i+1}\right|\right|^2_2 \tag{26}$$

$$(\boldsymbol{\theta}^1_1)^*, \ldots, (\boldsymbol{\theta}^1_k)^* = \underset{\boldsymbol{\theta}^1_1,\ldots,\boldsymbol{\theta}^1_k}{\arg\min} \left\{ \sum_{i=1}^{m-1} \mathcal{L}^i_{LTI}(\Phi_i, \Phi_{i+1}) \right\}. \tag{27}$$

Similarly, $(\boldsymbol{\Gamma}\Phi_i)^{k,l}$ are the elements of lattice $\boldsymbol{\Gamma}\Phi_i$. $||\cdot||_2$ is the matrix norm equivalent to the Frobenius norm $||\cdot||_F$. A matrix norm is applicable because our lattices are equivalent to matrices in the sense of how they are stored as data. $0 < \gamma_{LTI} \leq 1$ is a scaling constant to ensure minimal loss is achieved for desirable choice of learning rate λ_1 in a reasonable number of epochs. Only one convolution kernel $\boldsymbol{\theta}^1$ is needed to learn the mapping $\boldsymbol{\Gamma}$.

We proceed in a similar manner as with the diffusion and advection-diffusion cases: we train a CNN $\mathcal{C}^1_{LTI}$ to learn the operator $\boldsymbol{\Gamma}$. We can construct training data $\{\Phi_k\}_{k\in\mathcal{K}}$ using the numerical method previously outlined. With the kernels from $\mathcal{C}^1_{LTI}$, we can train a new CNN $\mathcal{C}^2_{LTI}$ mapping the weights of the $\mathcal{C}^1_{LTI}$ kernel to the drift and diffusion values, which are known as part of our training data. The architecture for $\mathcal{C}^1_{LTI}$ is simple, with only one 5×5 kernel and padding of 2. An architecture for $\mathcal{C}^2_{LTI}$ identical to that in the diffusion and advection-diffusion sections is chosen.

Allow us to demonstrate some results with the following tables:

Table 4. Learning coefficients with long-time integration method

Δt	**Coef. varied**	**Error for** $\widehat{\mu}_{1,i}$	**Error for** $\widehat{\mu}_{2,i}$	**Error for** $\widehat{\sigma}_{1,i}$	**Error for** $\widehat{\sigma}_{2,i}$
1e-3	$\lvert\mu_{1,i}\rvert \leq 0.25$	**1.1509**	0.1683	0.0102	0.0157
1e-3	$\lvert\mu_{2,i}\rvert \leq 0.25$	0.0197	**0.5881**	0.1165	0.0868
1e-3	$\lvert\sigma_{1,i}\rvert \leq 0.25$	0.0132	0.0155	**0.1273**	0.0087
1e-3	$\lvert\sigma_{2,i}\rvert \leq 0.25$	0.0150	0.0113	0.0062	**0.1130**
1e-3	$\lvert\mu_{1,i}\rvert \leq 10$	**0.1282**	0.0022	0.0880	0.0382
1e-3	$\lvert\mu_{2,i}\rvert \leq 10$	0.0218	**0.0586**	0.1132	0.0862
1e-2	$\lvert\mu_{1,i}\rvert \leq 0.25$	**0.8962**	0.0033	0.0113	0.0142
1e-2	$\lvert\mu_{2,i}\rvert \leq 0.25$	0.0110	**0.7830**	0.0110	0.0118

Table 4. Above we have average relative error in predicting drift and diffusion coefficients using the long-time integration method on a test data set in which the test coefficients differ from the training coefficients. Only one coefficient is varied at a time. The others are expected to remain constant.

The first observation we make is that there is a severe inability to recover drift values when they remain relatively close in value to diffusion values (between 0 and 0.25 for all coefficients). Relative error is as high as 1.1509, which is highly inaccurate; however, when we increase drift values to as high as 10 but maintain diffusion values under 0.25, our accuracy in predicting coefficients on a test data set increases dramatically. It is possible diffusion coefficients dominate the weights in the kernels, or the weights cannot measure differences in drift when sufficiently small. We also notice a change in Δt does not make a dramatic change in ability to recover drift values. Lastly, we note ability to recover diffusion values is decent ($\sim$ 10% relative error) for all cases, even when sufficiently small.

Table 5. Learning coefficients with LTI continued

Coef. varied	**Error for** $\widehat{\mu}_{1,i}$	**Error for** $\widehat{\mu}_{2,i}$	**Error for** $\widehat{\sigma}_{1,i}$	**Error for** $\widehat{\sigma}_{2,i}$
$\lvert\mu_{1,i}\rvert, \lvert\mu_{2,i}\rvert \leq 0.25$	**0.7406**	**0.9219**	0.0085	0.0120
$\lvert\mu_{1,i}\rvert, \lvert\mu_{2,i}\rvert \leq 10$	**0.2362**	**0.2114**	0.0052	0.0113
$\lvert\sigma_{1,i}\rvert, \lvert\sigma_{2,i}\rvert \leq 0.25$	0.0083	0.0217	**0.2312**	**0.1638**

Table 5. Above we have average relative error in coefficient predictions when more than one coefficient at a time is varied. This table is more interesting as it better demonstrates ability to predict coefficients when they are unknown.

Relative error increases ($\sim 20\%$) when more than one coefficient at a time is varied, but this is to be expected. Relative error is equally high as before in cases with low drift values, but decreases dramatically as these values are increased and made further from the diffusion values.

Overall, we see that our long-time integration method of determining drift and diffusion coefficients does not perform as well as in the previous cases when $\mathcal{C}^1$ learns a differential operator. In the next section, we will return to learning differential operators but in a more generalized setting.

7 Finite difference and finite volume methods

The diffusion and advection-diffusion equations provide good benchmarks for cases to check and understand when learning the differential operator. Now we aim to deduce drift and diffusion coefficients in a more generalized setting, one in which we can leave the Fokker-Planck differential operator in its form $\mathcal{D}$. Our CNNs are now motivated by two problems. The first is finite difference method (FDM) [4] [5]

$$\frac{\Phi_{i+1} - \Phi_i}{\Delta t} = \mathcal{D}_{FDM}\big[\Phi_i\big] \tag{28}$$

which can be reformulated as

$$\Phi_{i+1} = \big(1 + \Delta t \cdot \mathcal{D}_{FDM}\big)\Phi_i. \tag{29}$$

$\mathcal{D}_{FDM}$ is the Fokker-Planck differential operator applied to lattice Φ_i computed with finite difference methods, providing the subscript notation. Φ_i is distributive in the traditional sense, hence

$$\mathcal{D}_{FDM} \cdot \Phi_i = \mathcal{D}_{FDM}\big[\Phi_i\big] \approx \Big[\Big(-\sum_{i=1}^{2} \mu_i \frac{\partial}{\partial X_i} + \sum_{j=1}^{2} \sigma_j \frac{\partial^2}{\partial X_j^2}\Big)\rho(\boldsymbol{X}, t_i|\rho_0)\Big]\Big|_{\boldsymbol{X}\in\Omega} \tag{30}$$

is the numerical approximation of the differential operator applied to Φ_i. Our CNN attempts to learn the differential operator $\mathcal{D}_{FDM}$ which we can do with the individual loss minimization problem

$$\mathcal{L}^{i}_{FDM}(\Phi_i, \Phi_{i+1}|\boldsymbol{\theta}^1_1, \dots, \boldsymbol{\theta}^1_k) = \gamma_{FDM}\left|\left|\frac{\Phi_{i+1} - \boldsymbol{\beta}_i}{\Delta t}\right|\right|^2_2 \tag{31}$$

$$\boldsymbol{\beta}_i = \Big(1 + \Delta t \cdot \mathcal{C}^1_{FDM}\Big)\Phi_i. \tag{32}$$

where our CNN $\mathcal{C}^1_{FDM}$ takes place of $\mathcal{D}_{FDM}$ and is applied to Φ_i.

Our FDM is similar to the advection-diffusion case with combined operators but with some alterations. The first is that there are now two diffusion coefficients instead of one $D = \sigma_1 = \sigma_2$. The second is that the differential operator we are learning is unconstrained, and is no longer constricted to the form $\boldsymbol{P} = \boldsymbol{L} + \boldsymbol{A}$ for negative semi-definite $\boldsymbol{L}$ and skew-symmetric $\boldsymbol{A}$.

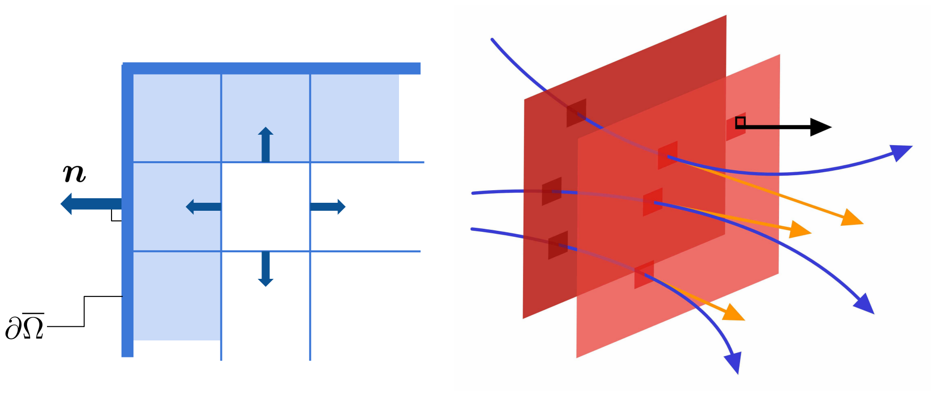

Figure 6. Above we have an illustration of cell construction for our finite volume methods. The left depicts the layout of the lattice. The right depicts the flux through the cell walls, where the the wall of one cell is an additional wall of a different cell.

The second problem motivating our CNN design is a finite volume model (FVM) [4] [5]. A differential operator $\mathcal{D}_{FVM}$ containing a divergence term can be learned such that the time discretization in the left-hand side of equation (26) holds as it did in the previous FDM case. This new differential operator corresponds to the Fokker-Planck differential operator. A FVM employs the divergence theorem

$$\int_V (\nabla \cdot \boldsymbol{u})dV = \oint_S \boldsymbol{u} \cdot d\boldsymbol{A}. \tag{33}$$

for arbitrary vector field $\boldsymbol{u}$ and domains V, $S = \partial V$, meaning surface integrals are under consideration. Additionally, FVMs are conservative. Mass is preserved, meaning the multivariate probability density condition $\iint_{\mathbb{R}^2} \rho d\boldsymbol{X} = 1$ is satisfied for our Fokker-Planck solutions. The FVM under consideration is

$$\frac{\Phi_{i+1}^{k,l} - \Phi_i^{k,l}}{\Delta t} = \frac{1}{\mu(c_{k,l})} \sum_{f \in F_{k,l}} \int_f \mathcal{D}_{FVM}[\Phi_i] \cdot d\boldsymbol{A}. \tag{34}$$

We use the notation outlined in the previous section, where $\Phi_i^{k,l}$ denote the elements of lattice Φ_i at location (k,l) along Ω. The cells $\mathbb{C} = \{[X_{k-\frac{1}{2}}, X_{k+\frac{1}{2}}] \times [X_{l-\frac{1}{2}}, X_{l+\frac{1}{2}}] \times [0, \ell] \mid k,l \in \{1, \ldots, 2\lambda+1\}, \ell \in \mathbb{R}^+\}$ are constructed around discretized points in Ω. $\mu(c_{k,l})$ is cell Lebesgue measure, being measure of the Cartesian product of intervals. $\mu(c_{k,l}) = \ell(\Delta X)^2$ given equispaced construction of Ω. Cell faces $F_{k,l} = \partial c_{k,l}$ are the rectangular surfaces in which the flux is passing through. We will attempt to learn the operator in the integrand by solving the loss minimization problem

$$\mathcal{L}_{FVM}^i(\Phi_i, \Phi_{i+1} | \boldsymbol{\theta}_1^1, \ldots, \boldsymbol{\theta}_k^1) = \gamma_{FVM} \left|\left| \frac{\Phi_{i+1} - \Phi_i}{\Delta t} - \boldsymbol{\Lambda}_i \right|\right|_2^2 \tag{35}$$

$$\Lambda_i^{k,l} = \frac{1}{\mu(c_{k,l})} \sum_{f \in F_{k,l}} \int_f \mathcal{C}_{FVM}^1[\Phi_i] \cdot d\boldsymbol{A}. \tag{36}$$

Here $\Lambda_i^{k,l}$ is the surface integral for particular cell $c_{k,l}$ scaled by $1/\mu(c_{k,l})$. $\boldsymbol{\Lambda}_i$ is the lattice containing the values $\Lambda_i^{k,l}$. Again, the kernels are the tuned hyperparameter to minimize total loss, and γ_{FVM} is a suitable scalar.

It is notable that the operator here, represented by $\mathcal{C}_{FVM}$, is producing a vector field from a lattice. We require a CNN with two distinct ouput channels, one learning the vector field with respect to one direction, say the faces with normal vectors in the $\pm x$-direction in the $\mathbb{R}^2$ plane, and another with normals facing the $\pm y$-direction. Hence, we can write

$$\sum_{f \in F_{k,l}} \int_f \mathcal{C}_{FVM}^1[\Phi_i] \cdot d\boldsymbol{A} = \sum_{f \in F_{k,l}} \int_f \left(\mathcal{C}_{FVM}^x, \mathcal{C}_{FVM}^y\right)^T \Phi_i \cdot d\boldsymbol{A} \tag{37}$$

where $\mathcal{C}_{FVM}^x$ and $\mathcal{C}_{FVM}^y$ are the CNN output channels computing this discretized vector field with respect to either the $\pm x$ or $\pm y$ directions. $(\mathcal{C}_{FVM}^x, \mathcal{C}_{FVM}^y)^T \Phi_i$ is the vector concatenation of the elements in $\mathcal{C}_{FVM}^x[\Phi_i]$ and $\mathcal{C}_{FVM}^y[\Phi_i]$, producing the discretized vector field. Based on cell construction, we require vector field values situated not along Ω but along boundary $\partial([X_{k-\frac{1}{2}}, X_{k+\frac{1}{2}}] \times [X_{l-\frac{1}{2}}, X_{l+\frac{1}{2}}])$ instead. Such a setup is a staggered mesh. We can form our staggered mesh by translating the vector field produced by the CNN.

Allow us to demonstrate our FDM and FVM results.

Table 6. Learning coefficients with FDMs

Coefficient(s) varied	**Error for** $\widehat{\sigma}_{1,i}$	**Error for** $\widehat{\sigma}_{2,i}$	**Error for** $\widehat{\mu}_{1,i}$	**Error for** $\widehat{\mu}_{2,i}$
$0 < \sigma_{1,i} \leq 1$	**0.0304**	0.0129	0.0108	0.0473
$0 < \sigma_{2,i} \leq 1$	0.0245	**0.0195**	0.0194	0.0333
$0 < \mu_{1,i} \leq 1$	0.0322	0.0095	**0.0672**	0.0211
$-1 \leq \mu_{2,i} < 0$	0.0173	0.0213	0.0287	**0.0399**
$0 < \mu_{1,i} \leq 10$	0.0170	0.0126	**0.0869**	0.0175
$-10 \leq \mu_{2,i} < 0$	0.0449	0.0441	0.0174	**0.0728**
$\lvert\sigma_{1,i}\rvert, \lvert\sigma_{2,i}\rvert \leq 1$	**0.0592**	**0.0987**	0.0231	0.0979
$\lvert\mu_{1,i}\rvert, \lvert\mu_{2,i}\rvert \leq 1$	0.0205	0.0093	**0.0440**	**0.0235**
$\lvert\mu_{1,i}\rvert, \lvert\mu_{2,i}\rvert \leq 10$	0.0131	0.0139	**0.0655**	**0.0652**
$\lvert\sigma_{1,i}\rvert, \lvert\sigma_{1,i}\rvert, \lvert\mu_{1,i}\rvert, \lvert\mu_{2,i}\rvert \leq 1$	**0.0712**	**0.0916**	**0.0285**	**0.0100**
$\lvert\sigma_{1,i}\rvert, \lvert\sigma_{1,i}\rvert \leq 1, \lvert\mu_{1,i}\rvert, \lvert\mu_{2,i}\rvert \leq 10$	**0.1738**	**0.1366**	**0.0376**	**0.0123**

Table 6. Above are our results for coefficient estimation using our FDM scheme. Just as before, a 5×5 kernel is used for $\mathcal{C}^1_{FDM}$ and initial condition ϕ_0 is used. The results are for average relative error on a test dataset.

Results with this approach are comparable to the advection-diffusion case when the constrained differential operator is learned. Results in approximation for $\sigma_{1,i}, \sigma_{2,i}$ when all coefficients are varied are better when values lie closer to drift values $\mu_{1,i}, \mu_{2,i}$. Drift approximations do not change as much when the coefficients are varied within different ranges. In general, average relative error is about under 10%. Loss for $\mathcal{C}^1_{FDM}$ typically terminated with a value of about 0.1 after 3,000 epochs. Loss for $\mathcal{C}^2_{FDM}$ varied on dataset size, as this was changed based on the coefficients to be estimated.

Table 7. Learning coefficients with FVMs

Coefficient(s) varied	**Error for** $\widehat{\sigma}_{1,i}$	**Error for** $\widehat{\sigma}_{2,i}$	**Error for** $\widehat{\mu}_{1,i}$	**Error for** $\widehat{\mu}_{2,i}$
$0 < \sigma_{1,i} \leq 1$	**0.0390**	0.0121	0.0255	0.0185
$0 < \sigma_{2,i} \leq 1$	0.1040	**0.0503**	0.1039	0.1037
$0 < \mu_{1,i} \leq 1$	0.0174	0.0191	**0.0136**	0.0227
$-1 \leq \mu_{2,i} < 0$	0.0231	0.0287	0.0230	**0.0302**
$0 < \mu_{1,i} \leq 10$	0.0613	0.0608	**0.0788**	0.0172
$-10 \leq \mu_{2,i} < 0$	0.0363	0.0369	0.0092	**0.0599**
$\lvert\sigma_{1,i}\rvert, \lvert\sigma_{2,i}\rvert \leq 1$	**0.0731**	**0.0369**	0.0131	0.0105
$\lvert\mu_{1,i}\rvert, \lvert\mu_{2,i}\rvert \leq 1$	0.0192	0.0275	**0.0082**	**0.0205**
$\lvert\mu_{1,i}\rvert, \lvert\mu_{2,i}\rvert \leq 10$	0.0515	0.0526	**0.0541**	**0.0748**
$\lvert\sigma_{1,i}\rvert, \lvert\sigma_{1,i}\rvert, \lvert\mu_{1,i}\rvert, \lvert\mu_{2,i}\rvert \leq 1$	**0.1186**	**0.1238**	**0.1586**	**0.1669**
$\lvert\sigma_{1,i}\rvert, \lvert\sigma_{1,i}\rvert \leq 1, \lvert\mu_{1,i}\rvert, \lvert\mu_{2,i}\rvert \leq 10$	**0.1597**	**0.0822**	**0.0124**	**0.0180**

Table 7. Above are our results when using FVMs. Average relative error is generally consistent with FDMs.

Results with FVMs do not differ significantly when learning a differential operator in a different setting, such as with FDMs. Average relative error is low (typically $< 6\%$) when one coefficient is varied, and is generally under 15% when all coefficients are intended to be learned. Varying the ranges of our coefficients in the final setting did not result in a significant impact in ability to recover diffusion values, where diffusion values remain between 0 and 1, and drift values ascend as high as 10.

There are computational considerations when constructing a dataset for $\mathcal{C}^2$. Typically, we used under 50 kernels from $\mathcal{C}^1$ to generate our results, as we found that dataset size has insignificant impact. 50 kernels means that will we need to generate 50 Fokker-Planck datasets, as well as train 50 CNNs to generate the kernels, each with 3,000 epochs. This becomes computationally expensive when the number of kernels we wish to train $\mathcal{C}^2$ upon becomes large; however, for the sake of confirming the suspicion that kernel size does not affect coefficient prediction accuracy to a notable extent, we perform coefficient estimation with $\mathcal{C}^2_{FVM}$ with 256 kernels. Our results are illustrated below in the following table.

Table 8. Learning coefficients with FVMs, higher quantity of $\mathcal{C}^2_{FVM}$ data

Coefficient(s) varied	**Error for** $\widehat{\sigma}_{1,i}$	**Error for** $\widehat{\sigma}_{2,i}$	**Error for** $\widehat{\mu}_{1,i}$	**Error for** $\widehat{\mu}_{2,i}$
$\lvert\sigma_{1,i}\rvert, \lvert\sigma_{2,i}\rvert \leq 1$	**0.2959**	**0.2947**	0.0253	0.0653
$\lvert\mu_{1,i}\rvert, \lvert\mu_{2,i}\rvert \leq 1$	0.0263	0.0328	**0.0173**	**0.0346**
$\lvert\sigma_{1,i}\rvert, \lvert\sigma_{1,i}\rvert, \lvert\mu_{1,i}\rvert, \lvert\mu_{2,i}\rvert \leq 1$	**0.0388**	**0.0535**	**0.0538**	**0.1059**

Table 8. The above results were collected by training $\mathcal{C}^2_{FVM}$ on 256 kernels generated from $\mathcal{C}^1_{FVM}$. We perform such an investigation in attempt to deduce the effect the amount of training data has on accuracy.

There is minimal discernible difference by training on more data for the last two cases of the table. Average relative error is generally consistent with the FVMs when training on lower amounts of data. Regarding the first case on the table, average relative error is significantly higher when deducing diffusion coefficients. This may be caused by overfitting. In general, lower amounts of data are sufficient or preferred.

8 Conclusion

We may compute numerical solutions of the Fokker-Planck equation over a discretized domain in space, which are suitable for convolutional neural networks. These CNNs may be employed to learn either some notion of a differential operator, or to map Fokker-Planck solutions at a particular juncture to those at a greater juncture. We may train a second CNN to map the kernels of the first to the drift and diffusion coefficients, allowing us to deduce these unknown values given Fokker-Planck data. We break up our objective into cases. Relative error when predicting one coefficient is generally below 5%, and when predicting all four coefficients, is generally below 15%.

References

[1] Moehlis. Appendix: Derivation of the Fokker-Planck Equation. https://sites.me.ucsb.edu/~moehlis/moehlis_papers/appendix.pdf

[2] Pichler L., Masud A., Bergman L.A. (2013) Numerical Solution of the Fokker–Planck Equation by Finite Difference and Finite Element Methods—A Comparative Study. In: Papadrakakis M., Stefanou G., Papadopoulos V. (eds) Computational Methods in Stochastic Dynamics. Computational Methods in Applied Sciences, vol 26. Springer, Dordrecht. http://congress.cimne.com/eccomas/proceedings/compdyn2011/compdyn2011_full/059.pdf

[3] Caughey T.K. (1963) Derivation and Application of the Fokker-Planck Equation to Discrete Nonlinear Dynamic Systems Subjected to White Random Excitation. In: The Journal of the Acoustical Society of America, vol. 35, no. 11. https://authors.library.caltech.edu/4087/1/CAUjasa63a.pdf

[4] Trask, N., Patel, R.G., Gross, B.J., Atzberger, P.J. (2019) GMLS-Nets: A Framework for Learning from Unstructured Data. https://arxiv.org/pdf/1909.05371.pdf

[5] Trask, N., Patel, R.G., Gross, B.J., Atzberger, P.J. (2019) GMLS-Nets: Scientific Machine Learning Methods for Unstructured Data. http://web.math.ucsb.edu/~atzberg/pmwiki_intranet/uploads/AtzbergerHomePage/preprint_GMLS_Nets__NeurIPs.pdf

[6] Qi, J., Du, J., Siniscalchi, S.M., Ma, X., Lee, C. (2020) On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression. https://arxiv.org/pdf/2008.07281.pdf

[7] Linge, S., Langtangen, H. (2017) Diffusion Equations. https://www.researchgate.net/publication/318168235_Diffusion_Equations

[8] Ruthotto, L., Haber, E. (2018) Deep Neural Networks Motivated by Partial Differential Equations. CoRR, vol. abs/1804.04272. https://arxiv.org/pdf/1804.04272.pdf

A Appendix: A proof of the Chapman-Kolmogorov theorem

The Chapman-Kolmogorov theorem is one that is used in the derivation of the Fokker-Planck equation. We provide a proof of this theorem for the continuous case for three n-dimensional random variables. The proof of the discrete case is significantly more renowned than the continuous case, which is why we include it here. We omit the derivation of the Fokker-Planck equation, as one can be found in references [1], [3]. Denote $\rho(\cdot,\cdot)$ as a joint probability density function and $\rho(\cdot|\cdot)$ as a conditional one.

Theorem 1 (Chapman-Kolmogorov): *Suppose random variables $\boldsymbol{X}_1, \boldsymbol{X}_2, \boldsymbol{X}_3$ in $\mathbb{R}^n$ are part of a Markov process, $t_1 < t_2 < t_3$, all of which with PDF support $\mathbb{R}^n \times \mathbb{R}$. Then*

$$\rho(\boldsymbol{X}_3, t_3|\boldsymbol{X}_1, t_1) = \iint \cdots \int_{\mathbb{R}^n} \rho(\boldsymbol{X}_3, t_3|\boldsymbol{X}_2, t_2)\rho(\boldsymbol{X}_2, t_2|\boldsymbol{X}_1, t_1) d\boldsymbol{X}_2.$$

Proof.

Consider

$$\rho(\boldsymbol{X}_3, t_3|\boldsymbol{X}_1, t_1) = \iint \cdots \int_{\mathbb{R}^n} \rho(\boldsymbol{X}_3, t_3, \boldsymbol{X}_2, t_2|\boldsymbol{X}_1, t_1) d\boldsymbol{X}_2.$$

By the conditional probability property for continuous random variables, if $\rho(\boldsymbol{X}_1, t_1) > 0$,

$$= \iint \cdots \int_{\mathbb{R}^n} \frac{\rho(\boldsymbol{X}_3, t_3, \boldsymbol{X}_2, t_2, \boldsymbol{X}_1, t_1)}{\rho(\boldsymbol{X}_1, t_1)} d\boldsymbol{X}_2.$$

Now, the conditional probability property in conjunction with the Markov property gives a general recursive formula

$$\rho(\boldsymbol{X}_n, t_n, \ldots, \boldsymbol{X}_2, t_2, \boldsymbol{X}_1, t_1) = \rho(\boldsymbol{X}_1, t_1) \prod_{i=1}^{n-1} \rho(\boldsymbol{X}_{i+1}, t_{i+1}|\boldsymbol{X}_i, t_i)$$

where the Markov property is given by

$$\rho(\boldsymbol{X}_i, t_i|\boldsymbol{X}_{i-1}, t_{i-1}, \ldots, \boldsymbol{X}_1, t_1) = \rho(\boldsymbol{X}_i, t_i|\boldsymbol{X}_{i-1}, t_{i-1}),$$

i.e., the probability of moving to a next state is dependent only the current state and not the previous states. Hence,

$$= \iint \cdots \int_{\mathbb{R}^n} \frac{\rho(\boldsymbol{X}_3, t_3|\boldsymbol{X}_2, t_2)\rho(\boldsymbol{X}_2, t_2|\boldsymbol{X}_1, t_1)\rho(\boldsymbol{X}_1, t_1)}{\rho(\boldsymbol{X}_1, t_1)} d\boldsymbol{X}_2$$

and by canceling,

$$= \iint \cdots \int_{\mathbb{R}^n} \rho(\boldsymbol{X}_3, t_3|\boldsymbol{X}_2, t_2)\rho(\boldsymbol{X}_2, t_2|\boldsymbol{X}_1, t_1) d\boldsymbol{X}_2.$$

□