Convolutional neural networks in learning Fokker-Planck equations

Andrew Gracyk

For completion of MA

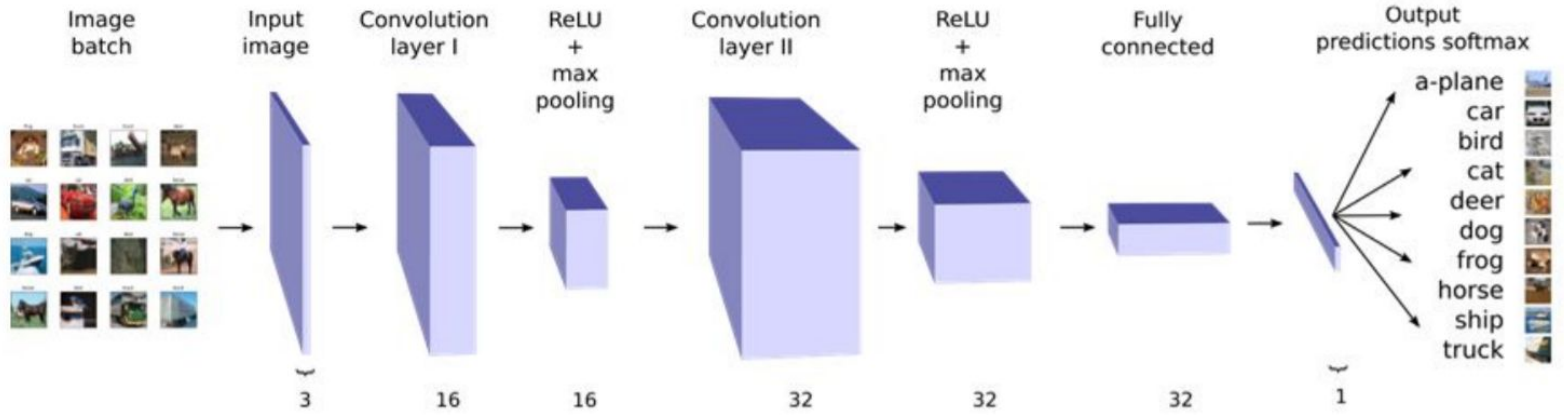
Advisor: Paul Atzberger. Thesis committee: Katy Craig, Xu Yang.

Objective

It is our overarching goal to deduce Fokker-Planck drift and diffusion coefficients given data of solutions.

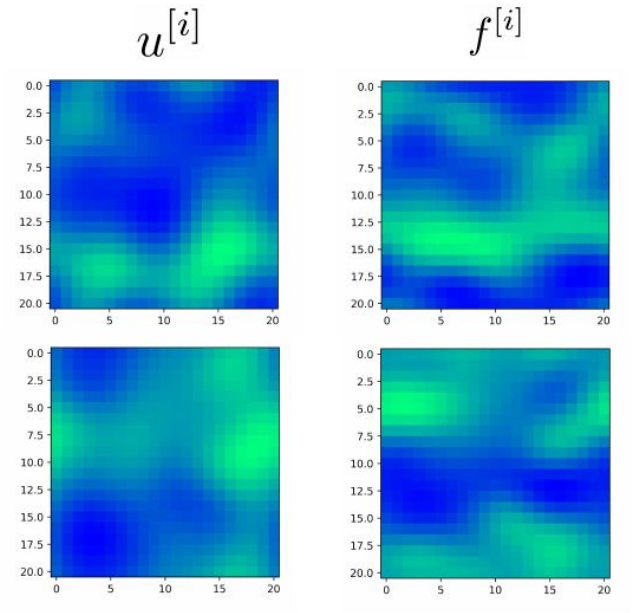
We may employ machine learning techniques in order to do so.

Convolutional neural network (CNN) background



Andrew Gracyk - UCSB Department of Mathematics

CNN background



CNNs do not only classify. Backpropagation can be performed between one image and another. Here, our CNN can learn the Laplacian.

Andrew Gracyk - UCSB Department of Mathematics

Turning Fokker-Planck solutions into images

We can turn Fokker-Planck solutions into "images" by discretizing a domain and evaluating solutions over this.

In particular, we can evaluate Fokker-Planck solutions over numerical domain

$$\Omega = \left\{(-mh, nh) \in \mathbb{R}^2 : h \in \mathbb{R}, m, n \in \{-\lambda, \ldots, -1, 0, 1, \ldots, \lambda\}\right\}$$

and define the continuous PDFs over

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

Turning Fokker-Planck solutions into images

Define the boundary of this discretized domain

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

This boundary will prove valuable for our numerical methods.

Integration condition

We provide the integration condition, which is important for initial conditions.

Our general PDFs cohere to

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

but in the context of our problem we consider

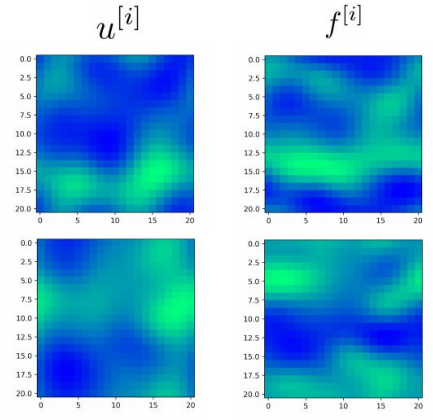
$$\iint_{\overline{\Omega}} \rho(\boldsymbol{X}, t_i | \rho_0) d\boldsymbol{X} \leq 1$$

Here, ρ is a multivariate PDF that solves the Fokker-Planck, $\boldsymbol{X}$ is a random vector, t_i is time, and ρ_0 is an initial condition.

What will CNNs do for us?

Our CNNs can do two things:

1) Learn the differential operator (a mapping from a solution to some new lattice)
2) A direct mapping from one solution to another



How will CNNs attain our goal?

Our CNN kernels should be dependent on drift and diffusion coefficients.

We can introduce a second CNN that maps the kernels of the first to the predicted coefficients.

The Fokker-Planck equation

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

We will hold drift and diffusion terms as constant functions. It is our goal to learn these constants given data with these values unknown.

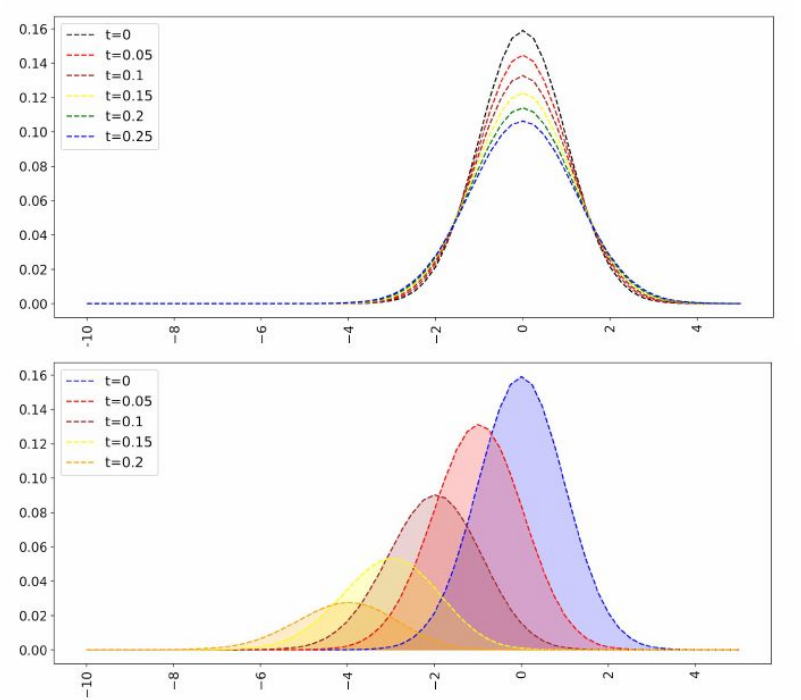
Our data: numerical solutions

We may evaluate Fokker-Planck solutions over our discretized domain Ω using a numerical method. Our numerical solutions will be denoted

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

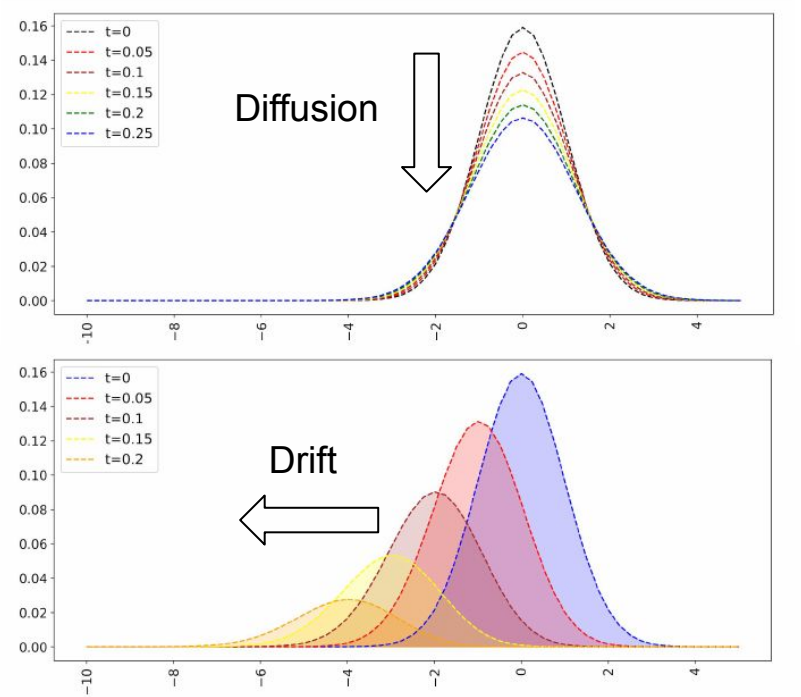
These form lattices which act as images, perfect for our CNNs. We will discuss our numerical method later.

Fokker-Planck solutions example



Andrew Gracyk - UCSB Department of Mathematics

Fokker-Planck solutions example



Andrew Gracyk - UCSB Department of Mathematics

The Fokker-Planck differential operator

We extract the Fokker-Planck differential operator. We can rewrite our the Fokker-Planck equation as

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

A large portion of what we'll do is attempting to deduce the right-hand side with CNNs. The differential operator given by

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

With a discretization, our PDE formulation becomes

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

Initial conditions

We establish two initial conditions, which are as follows:

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

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

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

where coefficient ν is predetermined. We only consider distributions with the desired support.

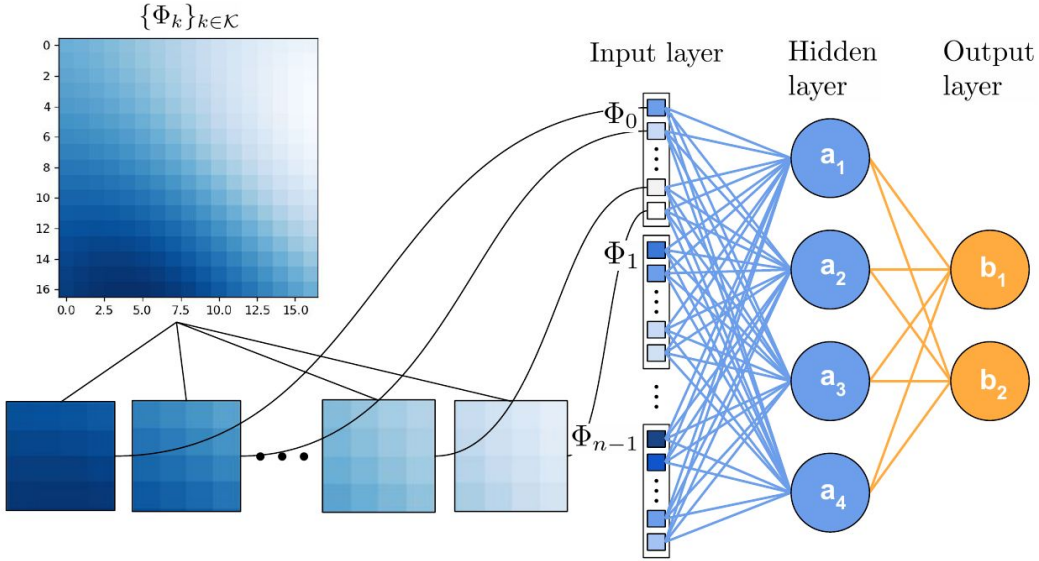
Designing our CNNs

We consider two types of CNNs, each with a different purpose:

1) The first has the goal of learning either the differential operator or a direct mapping between solutions.

2) The second has the goal of predicting drift and diffusion values from the kernels of the first.

CNN Illustration



Andrew Gracyk - UCSB Department of Mathematics

Our first type of CNN

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

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

where $\boldsymbol{T}_0$ is a PyTorch tensor of dimension $m \times 1 \times (2\lambda+1) \times (2\lambda+1)$ that is mapped to a new PyTorch tensor $\boldsymbol{T}_1$ with lattice elements. $\boldsymbol{\theta}_1^1, \ldots, \boldsymbol{\theta}_{k_1}^1$ are trainable kernel hyperparameters.

The input tensor will be a Fokker-Planck solution. The output tensor is the differential operator lattice or the next Fokker-Planck solution.

Our second type of CNN

Our second type of CNN learns the function

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

Where input tensor $\boldsymbol{\Theta}_0$ is a concatenation of kernels from the first CNN $\mathcal{C}^1$.
$\boldsymbol{\Sigma}_0$ is some prediction tensor to any collection of drift or diffusion values.

Our strategy

1) We begin by training $\mathcal{C}^1$ by generating data with drift and diffusion values within sequences with ranges $\sigma_{j,\min} \leq \sigma_{j,i} \leq \sigma_{j,\max}$, $\mu_{k,\min} \leq \mu_{k,i} \leq \mu_{k,\max}$.

2) With the kernels created from this, we can train $\mathcal{C}^2$ to map the kernels to drift or diffusion values.

3) Now, we can again construct Fokker-Planck solution data with new coefficients. This is the data expected to have unknown values.

4) We can feed this data into $\mathcal{C}^2$ to predict the drift and diffusion values.

A simplified case: the diffusion equation

We start our investigation by considering the PDE

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

with objective of recovering diffusion constant $D(\rho, \boldsymbol{X}) = D$ given snapshots of data $\{\Phi_k\}_{k\in\mathcal{K}}$.

This is a Fokker-Planck equation given $\nabla_{\boldsymbol{X}} D(\rho, \boldsymbol{X}) = 0$. Ψ_0 is an initial condition.

A simplified case: the diffusion equation

Here, we map solutions $\{\Phi_k\}_{k\in\mathcal{K}}$ to the numerical Laplacian lattices $\{\Delta\Phi_k\}_{k\in\mathcal{K}}$.

To do this, CNN $\mathcal{C}_D^1$ learns an operator $\boldsymbol{R}$ represented with one kernel. A second CNN that takes kernel negative transpose $-\boldsymbol{R}^T$ is applied, and so $\boldsymbol{L} = -\boldsymbol{R}^T\boldsymbol{R}$ is learned.

Our hypothesis space for $\boldsymbol{R}$ is constrained to

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

Operator $\boldsymbol{R}$ may have a non-unique representation given

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

The kernel weights from $\boldsymbol{L}$ are formed into our dataset for $\mathcal{C}_D^2$.

A simplified case: the diffusion equation

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

The advection-diffusion equation

Next, we investigate the advection-diffusion equation in an attempt to learn its coefficients. Our PDE formulation is below:

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

This is a Fokker-Planck equation if we hold $\nabla_{\boldsymbol{X}} \boldsymbol{v}(\boldsymbol{X},t) = 0$ with $\boldsymbol{v} = (v_1, v_2)$.

The advection-diffusion equation

Our numerical PDE formulation becomes

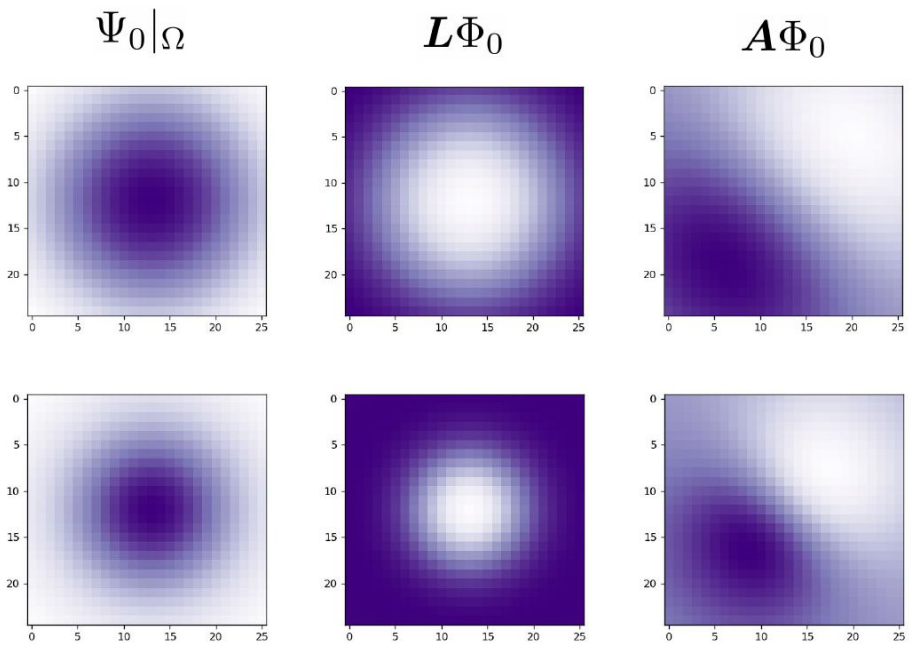
$$\frac{\Phi_{i+1} - \Phi_i}{\Delta t} = \boldsymbol{L}\Phi_i + \boldsymbol{A}\Phi_i = (\boldsymbol{L} + \boldsymbol{A})\Phi_i$$

where the terms are separated with the Laplacian and advection operators.

Here, the Laplacian operator is learned just as before, but we also learn skew-symmetric operator $\boldsymbol{A} = (1/2)(\boldsymbol{M} - \boldsymbol{M}^T)$.

Our CNN stencil learns $\boldsymbol{M}$. A new CNN takes this stencil transpose, and we minimize loss with convolution with $\boldsymbol{A}$ and the numerical advection lattice.

The advection-diffusion equation



The advection-diffusion equation

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

The table represents the ability to recover drift coefficients. Our first CNN learns the advection operator only.

The advection-diffusion equation

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**

Long-time integration

Here, it is our aim for our CNN $\mathcal{C}^1_{LTI}$ to learn an operator $\mathbf{\Gamma}$ such that

$$\Phi_{i+1} = \mathbf{\Gamma}\Phi_i$$

Lattice Φ_i is mapped directly to Φ_{i+1} .

Our numerical method

Allow us to provide our full numerical method for generating data. We provide it here since this is the most general case.

$$\frac{\Phi_{m+1}^{k,l} - \Phi_m^{k,l}}{\Delta t} = -\mu_1 \frac{\Phi_m^{k+1,l} - \Phi_m^{k-1,l}}{2h} - \mu_2 \frac{\Phi_m^{k,l+1} - \Phi_m^{k,l-1}}{2h} + \sigma_1 \frac{\Phi_m^{k+1,l} - 2\Phi_m^{k,l} + \Phi_m^{k-1,l}}{h^2} + \sigma_2 \frac{\Phi_m^{k,l+1} - 2\Phi_m^{k,l} + \Phi_m^{k,l-1}}{h^2}$$

We perform a backward scheme with this method. $\Phi_i^{k,l}$ are the lattice elements.

Long-time integration continued

Our optimization problem for our LTI method is as follows:

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

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

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

Long-time integration continued

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

Long-time integration continued

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**

Finite difference methods

With finite difference methods (FDMs), we learn the unconstrained differential operator, which is the right-hand side of

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

which can be reformulated as

$$\Phi_{i+1} = \left(1 + \Delta t \cdot \mathcal{D}_{FDM}\right)\Phi_i \, .$$

We are learning the lattice that is the numerical approximation of the operator applied to Φ_i which is

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

Finite difference methods

We can learn this differential operator with the loss minimization problem

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

$$\boldsymbol{\beta}_i = \left(1 + \Delta t \cdot \mathcal{C}^1_{FDM}\right)\Phi_i$$

where our CNN $\mathcal{C}^1_{FDM}$ takes place of our differential operator.

Finite volume methods

Our next problem is learning an operator such that the following equation is satisfied:

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

The divergence theorem is employed here, meaning we take surface integrals. The flux passses through the cells

$$\mathbb{C} = \{[X_{k-\frac{1}{2}}, X_{k+\frac{1}{2}}] \times [X_{l-\frac{1}{2}}, X_{l+\frac{1}{2}}] \times [0, \ell] \mid k, l \in \{1, \ldots, 2\lambda + 1\}, \ell \in \mathbb{R}^+\}$$

created around the discretized points in Ω. $\mu(c_{k,l})$ is cell Lebesgue measure and $F_{k,l}$ are the cell faces.

Finite volume methods

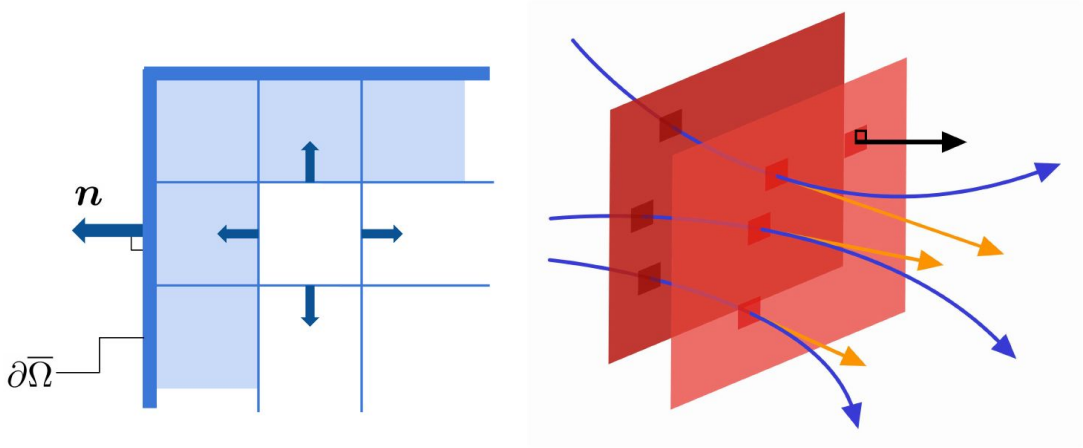


Illustration of cell construction and flux through cells.

Finite volume methods

Our loss minimization problem to learn this operator is

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

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

Finite volume methods

Here, we are producing a vector field from a lattice. We require a CNN with two output channels, one learning the $\pm x$ direction, the other the $\pm y$.

Hence, we can write

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

We require vector field values along the cell faces and not cell centers. Such means we can solve our problem with a staggered mesh setup.

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**

FDM and FVM results

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**

FDM and FVM results

In general, we did not find data set size for $\mathcal{C}^2$ significantly affected results. There are notable computational expenses for greater quantities of data, namely generating Fokker-Planck data and training $\mathcal{C}^1$ for 3,000 epochs.

We train on higher amounts of data to see results. We use FVMs.

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**

There is minimal discernible impact.

Conclusion

We can turn Fokker-Planck solutions into images. We may train CNNs to learn differential operators or direct mappings. We can learn the coefficients given data with these values unknown. Relative error is consistently under 5% when one coefficient is unknown and under 15% when all four are unknown.

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

[9] Atzberger, P.J. Convolutional Neural Networks (CNNs) Basics. http://web.math.ucsb.edu/~atzberg/pmwiki_intranet/uploads/AtzbergerHomePage/ml_lecture_NeuralNetworks_Basic_ConvNets_Atzberger.pdf

[10] Maschen. (Sept. 11 2012) General Flux Diagram.

Andrew Gracyk - UCSB Department of Mathematics

Questions?

Thanks for listening!