

Supplemental Information for Self-supervised learning method to solve the inverse problem of Fokker-Planck Equation

Wei Liu¹, Connie Khor Li Kou¹, Kun Hee Park², and Hwee Kuan Lee^{1,3,4,5,6}

¹Bioinformatics Institute, Agency for Science, Technology and Research (A*STAR), 30 Biopolis Street, #07-01 Matrix, 138671, Singapore, Singapore

²Centre for Quantum Technologies, National University of Singapore, 3 Science Drive 2, 117543, Singapore, Singapore

³School of Computing, National University of Singapore, 13 Computing Drive, 117417, Singapore, Singapore

⁴Singapore Eye Research Institute (SERI), 11 Third Hospital Ave, 168751, Singapore, Singapore

⁵Image and Pervasive Access Laboratory (IPAL), 1 Fusionopolis Way, #21-01 Connexis (South Tower), 138632, Singapore, Singapore

⁶Rehabilitation Research Institute of Singapore, 11 Mandalay Road #14-03, Clinical Sciences Building, 308232, Singapore, Singapore

*corresponding.author@email.example

Initializing $\hat{g}(x)$ and $\hat{h}(x)$ based on $\hat{P}^0(x, t)$ by linear least squares

In this section, $P(x, t)$, $g(x)$ and $h(x)$ are not discretized over variable x , and they are abbreviated as P , g and h for simplicity sake. We expand the right hand side of FPE:

$$\begin{aligned}\partial_t P &= \partial_x(gP) + \partial_{xx}(hP) \\ &= P\partial_x g + g\partial_x P + P\partial_{xx}h + 2\partial_x h\partial_x P + h\partial_{xx}P \\ &= (\partial_x g + \partial_{xx}h)P + (g + 2\partial_x h)\partial_x P + h\partial_{xx}P\end{aligned}\tag{1}$$

where P , $\partial_x P$, $\partial_{xx}P$ and $\partial_t P$ can be derived from P . Hence, for a fixed spatial point x and different time points $\{t_0, t_1, \dots, t_n\}$, we have a matrix equation:

$$\begin{bmatrix} P_{t0} & \partial_x P_{t0} & \partial_{xx} P_{t0} \\ P_{t1} & \partial_x P_{t1} & \partial_{xx} P_{t1} \\ \vdots & \vdots & \vdots \\ P_{tn} & \partial_x P_{tn} & \partial_{xx} P_{tn} \end{bmatrix} \begin{bmatrix} \partial_x g + \partial_{xx} h \\ g + 2\partial_x h \\ h \end{bmatrix} = \begin{bmatrix} \partial_t P_{t0} \\ \partial_t P_{t1} \\ \vdots \\ \partial_t P_{tn} \end{bmatrix}\tag{2}$$

where only the three terms $\partial_x g + \partial_{xx} h$, $g + 2\partial_x h$ and h are unknown and could be easily solved by linear least squares (LLS). We then repeat the process to solve the corresponding h at all the spatial points x to derive $h(x)$ and subsequently to calculate the derivative $\partial_x h$. Finally, we can solve g from the second term $g + 2\partial_x h$. The first term is not used in this method.

As describe in the main body, we smooth P_{noisy} to derive the initialized input $\hat{P}^0$. Then we can calculate the corresponding g and h base on $\hat{P}^0$, and use them as the initialized parameters g^0 and h^0 . In fact, the result of the LLS method is accurate enough only when P_{noisy} has minimal noise. Otherwise, the noise will be amplified by the derivative calculations and cause significant errors in the calculated g and h . Nevertheless, as the initialized parameters, the calculated g^0 and h^0 are still more accurate than the values from the conventional initialization methods.

Fully-connected layers to calculate derivatives with uniform grid

This method is generally a 1-dimensional version filter of the 2-dimensional filter in PDE-Net¹. Let f be a derivable function over x , $f(x) : \mathbb{R} \rightarrow \mathbb{R}$. The variable x is discretized as a vector $\mathbf{X} := [x_0, x_1, \dots, x_m]$ with uniform gap Δx . Consequently $f(x)$ becomes a vector $\mathbf{F} := [f(x_0), f(x_1), \dots, f(x_m)]$. To calculate the derivative of f at position x_i , we could use the Taylor series:

$$f(x_k) = f(x_i) + \frac{(k-i)\Delta x}{1!} \frac{df(x_i)}{dx} + \dots + \frac{(k-i)^n \Delta x^n}{n!} \frac{d^n f(x_i)}{dx^n} + \mathcal{O}(\Delta x^n) \quad (3)$$

We substitute Eq. 3 into the dot product between $\mathbf{F}$ and a coefficient vector $\mathbf{C} := [c_0, c_1, \dots, c_m]$:

$$\begin{aligned} \mathbf{C} \cdot \mathbf{F} &= \sum_{k=0}^m c_k f(x_k) \\ &= \sum_{k=0}^m c_k \left[f(x_i) + \frac{(k-i)\Delta x}{1!} \frac{df(x_i)}{dx} + \dots + \frac{(k-i)^n \Delta x^n}{n!} \frac{d^n f(x_i)}{dx^n} + \mathcal{O}(\Delta x^n) \right] \\ &= A_0 f(x_i) + A_1 \Delta x \frac{df(x_i)}{dx} + \dots + A_n \Delta x^n \frac{d^n f(x_i)}{dx^n} + \mathcal{O}(\Delta x^n) \end{aligned} \quad (4)$$

where:

$$A_n = \sum_{k=0}^m c_k \frac{(k-i)^n}{n!} \quad (5)$$

For a integer $s \leq n$, if we could find a vector $\mathbf{C}$ forcing A_j becomes:

$$A_j = \begin{cases} 0 & \text{if } j \neq s \text{ and } j \leq n, \\ 1 & \text{if } j = s. \end{cases} \quad (6)$$

then Eq. 4 becomes:

$$\mathbf{C} \cdot \mathbf{F} = \Delta x^s \frac{d^s f(x_i)}{dx^s} + \mathcal{O}(\Delta x^n) \quad (7)$$

therefore:

$$\frac{d^s f(x_i)}{dx^s} = \frac{\mathbf{C} \cdot \mathbf{F}}{\Delta x^s} + \mathcal{O}(\Delta x^{n-s}) \quad (8)$$

In such a way, we could approximate the s -order derivatives of $f(x_i)$ with a high order $(n-s)$ infinitesimal error. The vector $\mathbf{C}$ can be solved by substituting Eq. 5 into Eq. 6. In fact, we could always find a solution of $\mathbf{C}$ which only has $n+1$ non-zero entries. Because the Taylor series require $(k-i)\Delta x$ to be small enough, we choose the $n+1$ entries which are closest to the i -th entry to be non-zero entries. Finally, all the coefficient vector $\mathbf{C}$ could be concatenated together, to generate a $m \times m$ matrix $\mathbf{D}$ to approximate the derivative of $\mathbf{F}$:

$$\frac{d^s \mathbf{F}}{dx^s} \approx \frac{\mathbf{D} \mathbf{F}}{\Delta x^s} = \frac{1}{\Delta x^s} \begin{bmatrix} \mathbf{C}_{0s} \\ \mathbf{C}_{1s} \\ \vdots \\ \mathbf{C}_{ms} \end{bmatrix} \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_m) \end{bmatrix} \quad (9)$$

where $\mathbf{C}_{is}$ is the coefficient vector to calculate the s -th-order derivative at position x_i . For example, if $m = 8$ then the matrix $\mathbf{D}$ to calculate the 2nd-order derivative is:

$$\begin{bmatrix} -25/12 & 4 & -3 & 4/3 & -1/4 & 0 & 0 & 0 \\ -1/4 & -5/6 & 3/2 & -1/2 & 1/12 & 0 & 0 & 0 \\ 1/12 & -2/3 & 0 & 2/3 & -1/12 & 0 & 0 & 0 \\ 0 & 1/12 & -2/3 & 0 & 2/3 & -1/12 & 0 & 0 \\ 0 & 0 & 1/12 & -2/3 & 0 & 2/3 & -1/12 & 0 \\ 0 & 0 & 0 & 1/12 & -2/3 & 0 & 2/3 & -1/12 \\ 0 & 0 & 0 & -1/12 & 1/2 & -3/2 & 5/6 & 1/4 \\ 0 & 0 & 0 & 1/4 & -4/3 & 3 & -4 & 25/12 \end{bmatrix}$$

to ensure a 3rd-order infinitesimal error for all x points. Matrix $\mathbf{D}$ is used as the weights of a fully-connected layer, which takes $\mathbf{F}$ as input and output the corresponding derivative of $\mathbf{F}$.

Fully-connected layers to calculate derivatives with non-uniform grid

The section shows how to generate the weights of fully-connected layers which perform derivative calculation, when there is no uniform gap Δx for discretized variable x . The whole procedure is very similar to the case with uniform grid as described previously. The major difference is that the distance between x_k and x_i cannot be represented by $(k-i)\Delta x$ and we have to use $x_k - x_i$ instead. First, we expand the function f over x by Taylor series:

$$f(x_k) = f(x_i) + \frac{x_k - x_i}{1!} \frac{df(x_i)}{dx} + \dots + \frac{(x_k - x_i)^n}{n!} \frac{d^n f(x_i)}{dx^n} + \mathcal{O}((x_k - x_i)^n) \quad (10)$$

Then we multiply $\mathbf{F}$ which is the discretized form of $f(x)$ element-wisely to a coefficient vector $\mathbf{C}$:

$$\begin{aligned} \mathbf{C} \cdot \mathbf{F} &= \sum_k c_k f(x_k) \\ &= \sum_k c_k \left[f(x_i) + \frac{x_k - x_i}{1!} \frac{df(x_i)}{dx} + \dots + \frac{(x_k - x_i)^n}{n!} \frac{d^n f(x_i)}{dx^n} + \mathcal{O}((x_k - x_i)^n) \right] \\ &= A_{0i} f(x_i) + A_{1i} \frac{df(x_i)}{dx} + \dots + A_{ni} \frac{d^n f(x_i)}{dx^n} + \sum_k \mathcal{O}((x_k - x_i)^n) \end{aligned} \quad (11)$$

where:

$$A_{ni} = \sum_k c_k \frac{(x_k - x_i)^n}{n!} \quad (12)$$

Similarly, we always can find a vector $\mathbf{C}_i$ forcing A_j become:

$$A_{ji} = \begin{cases} 0 & \text{if } j \neq s \text{ and } j \leq n, \\ 1 & \text{if } j = s. \end{cases} \quad (13)$$

The subscript i of $\mathbf{C}_i$ indicates that the vector is specific to variable point x_i . We can repeat the process to find the corresponding vector to all the other variable points, and concatenate them to form a matrix $\mathbf{D}$. Hence the derivative of $\mathbf{F}$ can be approximated by:

$$\begin{aligned} \frac{d^s \mathbf{F}}{dx^s} &\approx \mathbf{D} \mathbf{F} \\ &\approx \begin{bmatrix} \mathbf{C}_{0s} \\ \mathbf{C}_{1s} \\ \vdots \\ \mathbf{C}_{ms} \end{bmatrix} \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_m) \end{bmatrix} \end{aligned} \quad (14)$$

Effect of the time point range in prediction when training FPE-Net with noisy target distributions

For a given x and a given time point t , Fig. S1A shows the $\mathbf{P}$ at current time point and nearby time points. The black triangles are $\mathbf{P}_{clean}$ which have a nearly linear relationship:

$$\mathbf{P}_{clean}(t + i\Delta t) = \mathbf{P}_{clean}(t) + i\Delta t \partial_t \mathbf{P}_{true}(t) + \mathcal{O}(i^2 \Delta t^2) \quad (15)$$

The red dots are $\mathbf{P}_{noisy}$ which contain random noise:

$$\mathbf{P}_{noisy}(t + i\Delta t) = \mathbf{P}_{clean}(t) + i\Delta t \partial_t \mathbf{P}_{clean}(t) + \mathcal{O}(i^2 \Delta t^2) + \mathbf{R}(t + i\Delta t) \quad (16)$$

where $\mathbf{R}(t + i\Delta t)$ is the random noise which usually is much larger than $\mathcal{O}(i^2 \Delta t^2)$.

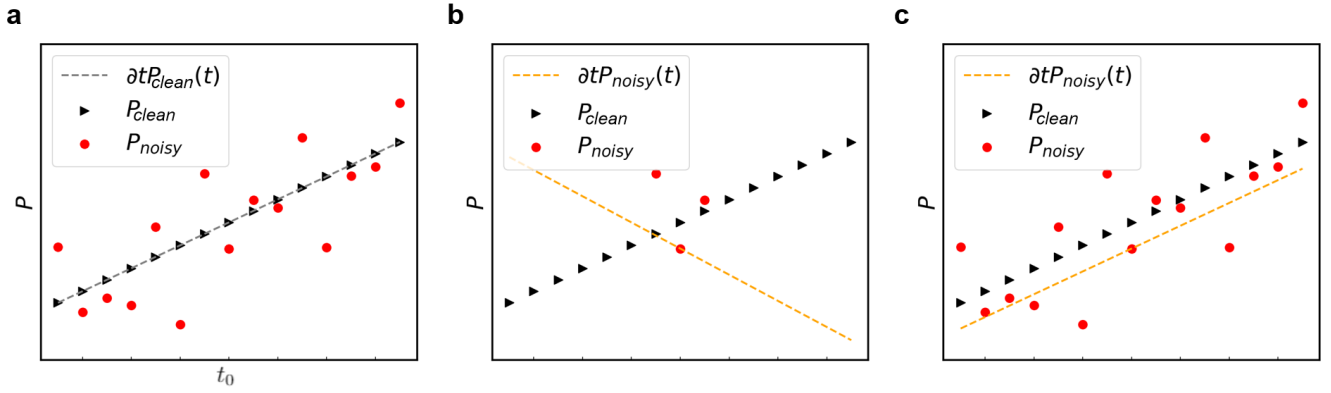


Figure S1. Plot of $\mathbf{P}$ and $\partial_t \mathbf{P}$ from true or noisy data. The black triangles $\blacktriangle$ are $\mathbf{P}_{clean}$, the red circles $\bullet$ are $\mathbf{P}_{noisy}$, the gray dash line -- is $\partial_t \mathbf{P}_{clean}(t)$ and the orange dash lines - - are the calculated $\partial_t \mathbf{P}_{noisy}(t)$.

If $\mathbf{P}_{clean}$ is unknown then we have to use $\partial_t \mathbf{P}_{noisy}(t)$ to approximate $\partial_t \mathbf{P}_{clean}(t)$. The $\partial_t \mathbf{P}_{noisy}(t)$ is derived by finding its optimal value which is used to predict:

$$\hat{\mathbf{P}}(t + i\Delta t) = \mathbf{P}_{noisy}(t) + i\Delta t \partial_t \mathbf{P}_{noisy}(t) \quad (17)$$

where $\hat{\mathbf{P}}(t + i\Delta t)$ have minimum square error with the target distribution $\mathbf{P}_{noisy}(t + i\Delta t)$:

$$L = \sum_{i=-n}^n \| \hat{\mathbf{P}}(t + i\Delta t) - \mathbf{P}_{noisy}(t + i\Delta t) \|_2 \quad (18)$$

Figure S1B shows if we set $n = 1$, the optimal $\partial_t \mathbf{P}_{noisy}(t)$ (orange dash line) is significantly different from $\partial_t \mathbf{P}_{clean}(t)$ (gray dash line) in Figure S1A. In contrast, $\partial_t \mathbf{P}_{noisy}(t)$ becomes much similar to $\partial_t \mathbf{P}_{clean}(t)$ when we set a larger n . For example, we set $n = 7$ in Fig. S1C. Hence when we train FPE NN with $\mathbf{P}_{noisy}$, adding more time points in prediction could help to suppress the noise effect.

The training results of the simulated data with higher initial noisy ratio (E_P)

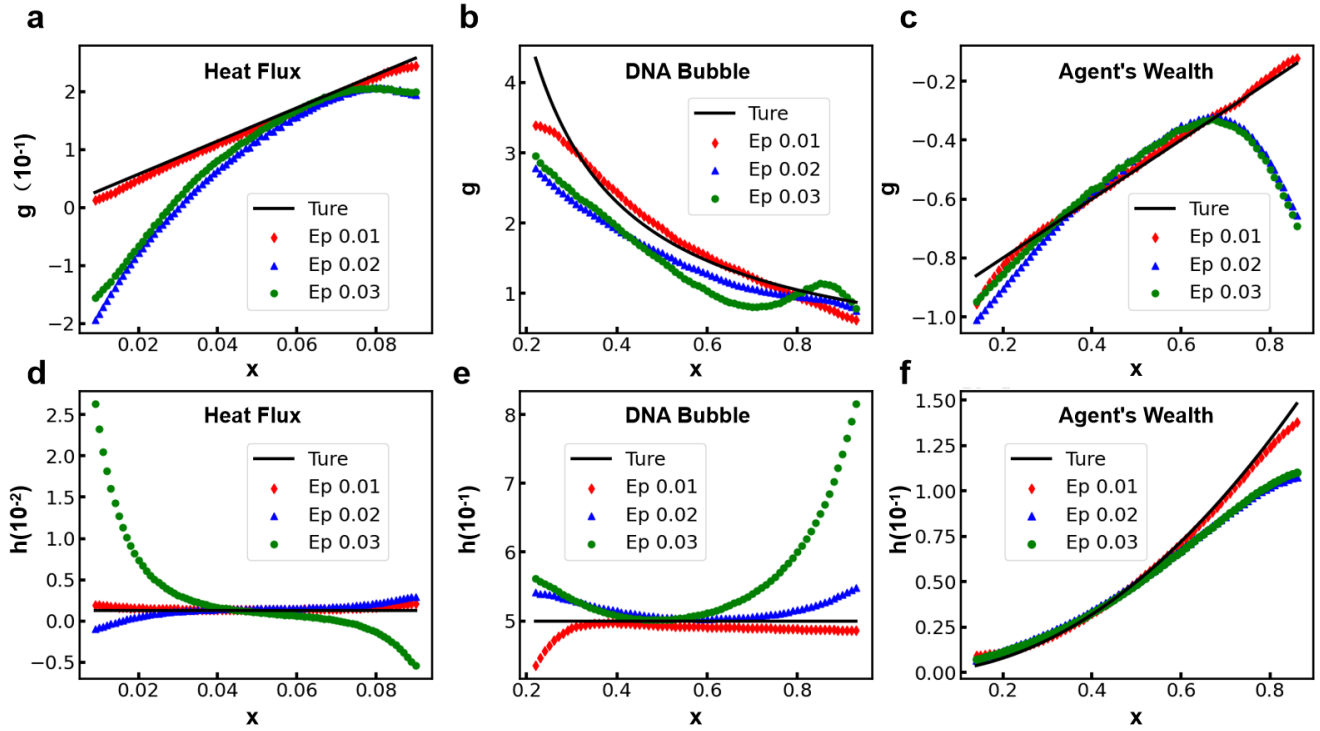


Figure S2. The final training result of the simulated data with initial noise ratio (E_P) 0.01 (red diamond $\blacklozenge$), 0.02 (blue triangle $\blacktriangle$), and 0.03 (green circle $\bullet$). The boundary points are excluded. All the training processes use the same hyperparameter $n = 8$. The true g and h are indicated in black line $—$. The result that the calculated $\hat{g}$ and $\hat{h}$ fit their true values very well, but a higher noise ratio could cause the training result less accurate.

References

1. Long, Z., Lu, Y., Ma, X. & Dong, B. Pde-net: Learning pdes from data (2018). [1710.09668](https://arxiv.org/abs/1710.09668).