Change Point - Physics Informed Neural Networks (CP-PINNs)

GitHub: https://github.com/yamansan/CP-PINNS

Yaman Sanghavi

December 13, 2024

Recap

- Using neural networks, we can approximate almost all practically useful functions. (Universal Approximation Theorem)

Recap

- Using neural networks, we can approximate almost all practically useful functions. (Universal Approximation Theorem)
- Using neural networks, we can also approximate the solutions to partial differential equations.

Goal

- Consider the Advection-Diffusion equation

$$\frac{\partial u(x,t)}{\partial t} + \frac{\partial u(x,t)}{\partial x} = \lambda(t)\frac{\partial^2 u(x,t)}{\partial x^2}$$

with

$$\lambda(t) = \begin{cases} 0.5 & \text{for } t \in [0, \frac{1}{3}), \\ 0.05 & \text{for } t \in [\frac{1}{3}, \frac{2}{3}), \\ 1.0 & \text{for } t \in [\frac{2}{3}, 1). \end{cases}$$

Goal

- Consider the Advection-Diffusion equation

$$\frac{\partial u(x,t)}{\partial t} + \frac{\partial u(x,t)}{\partial x} = \lambda(t)\frac{\partial^2 u(x,t)}{\partial x^2}$$

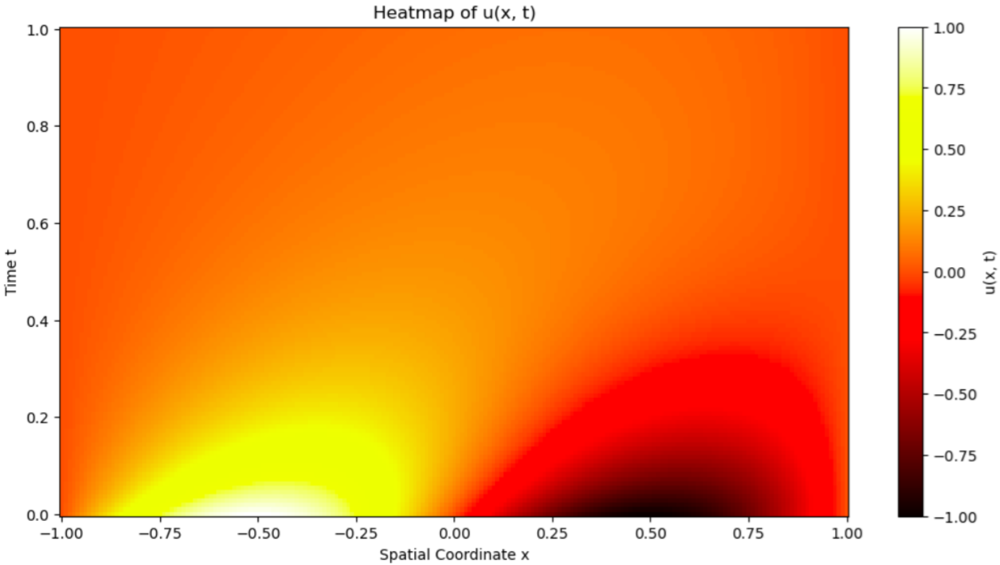
  with

$$\lambda(t) = \begin{cases} 0.5 & \text{for } t \in [0, \frac{1}{3}), \\ 0.05 & \text{for } t \in [\frac{1}{3}, \frac{2}{3}), \\ 1.0 & \text{for } t \in [\frac{2}{3}, 1). \end{cases}$$

- Given actual data points $u(x,t)$, we want to estimate $\lambda(t)$, hence the change-points too.

Training Data

For training data, we first solve the equation numerically.



Numerical Solution

Figure. Numerical Solution $\frac{\partial u}{\partial x} + \frac{\partial u}{\partial t} = \lambda(t)\frac{\partial^2 u}{\partial x^2}$ where $\lambda = 0.5$ with initial conditions $u(x,0) = -sin(\pi x)$ and $u(-1,t) = u(1,t) = 0$

Training Data

We choose data points from the numerical solution as our training data.

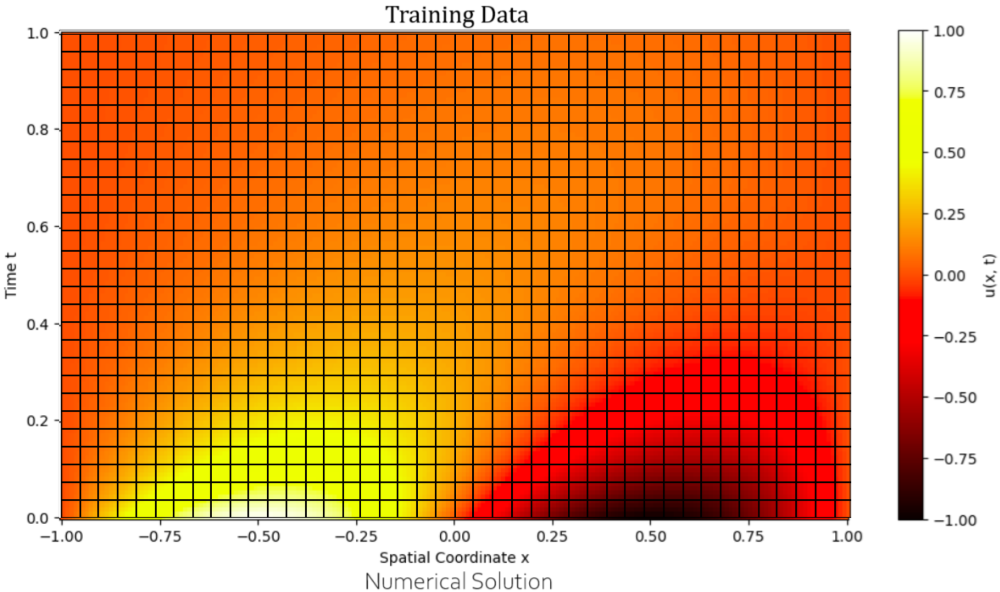


Figure. Training data taken from the numerical solution.

Training Data

We choose data points from the numerical solution as our training data.

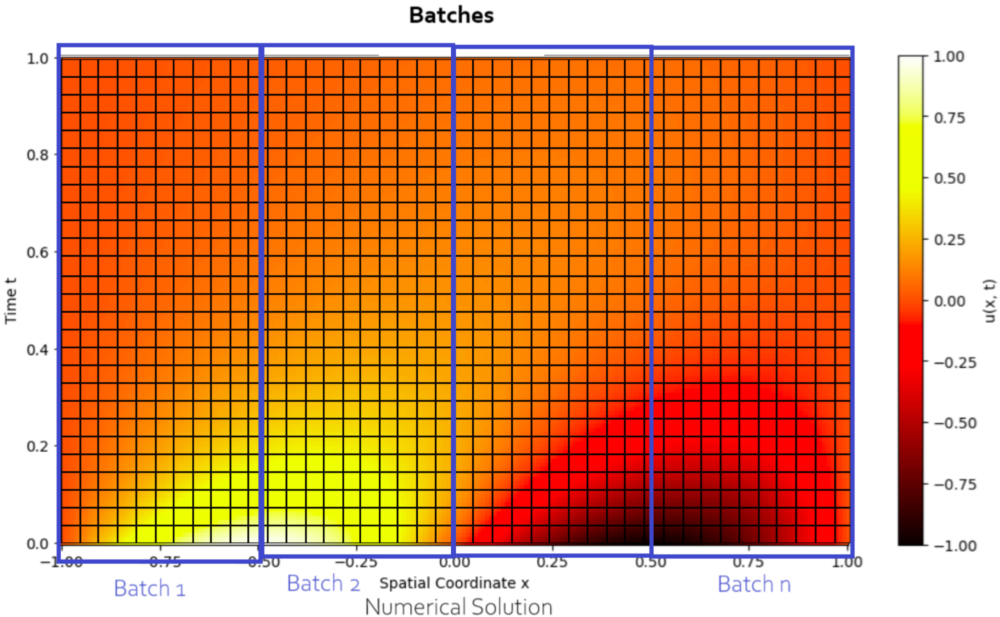


Figure. Training data taken from the numerical solution.

Loss Function for CP-PINN

- The first loss function term comes from the residual of the PDE

$$L^{NN} = \sum_{i,j} \left(\frac{\partial u_{NN}(x_i, t_j)}{\partial t_j} + \frac{\partial u_{NN}(x_i, t_j)}{\partial x_i} - \lambda_{NN}(t_j) \frac{\partial^2 u_{NN}(x_i, t_j)}{\partial x_i^2} \right)^2$$

Loss Function for CP-PINN

- The first loss function term comes from the residual of the PDE

$$L^{NN} = \sum_{i,j} \left(\frac{\partial u_{NN}(x_i, t_j)}{\partial t_j} + \frac{\partial u_{NN}(x_i, t_j)}{\partial x_i} - \lambda_{NN}(t_j) \frac{\partial^2 u_{NN}(x_i, t_j)}{\partial x_i^2} \right)^2$$

- The second loss function is from the training data and the boundary condition

$$L^{Training} = \sum_{i,j} \left(u_{NN}(x_i, t_j) - u_{train}(x_i, t_j) \right)^2$$

Loss Function for CP-PINN

- The first loss function term comes from the residual of the PDE

$$L^{NN} = \sum_{i,j} \left(\frac{\partial u_{NN}(x_i, t_j)}{\partial t_j} + \frac{\partial u_{NN}(x_i, t_j)}{\partial x_i} - \lambda_{NN}(t_j) \frac{\partial^2 u_{NN}(x_i, t_j)}{\partial x_i^2} \right)^2$$

- The second loss function is from the training data and the boundary condition

$$L^{Training} = \sum_{i,j} \left(u_{NN}(x_i, t_j) - u_{train}(x_i, t_j) \right)^2$$

- Third, we define a regularization term for $\lambda(t)$

$$V^{\lambda} = \sum_{i=1}^{T-1} \delta(t^i) \left| \Delta \lambda(t^i) \right|, \tag{1}$$

where $\delta(t)$ is a U-shaped function around $t = 0$.

Training Algorithm for CP-PINNs

- Then we write the total cost function as

$$L(\mathbf{w}; \mathbf{\Theta}, \lambda(t)) = w_1 L^{NN} + w_2 L^{Training} + w_3 V^{\lambda} \tag{2}$$

where $w_1 + w_2 + w_3 = 1$.

Training Algorithm for CP-PINNs

- Then we write the total cost function as

$$L(\mathbf{w}; \boldsymbol{\Theta}, \lambda(t)) = w_1 L^{NN} + w_2 L^{Training} + w_3 V^{\lambda} \tag{2}$$

  where $w_1 + w_2 + w_3 = 1$.
- For $(k-1)^{th}$ batch, minimize the cost function above w.r.t. the neural network weights $\boldsymbol{\Theta}$ and then use the to update the weights $\mathbf{w}$ for k^{th} batch

$$\begin{bmatrix} w_1^{(k)} \\ w_2^{(k)} \\ w_3^{(k)} \end{bmatrix} = \begin{bmatrix} \exp\left[-\eta L_{(k-1)}^{NN} - (1-\eta\gamma)\right] \\ \exp\left[-\eta L_{(k-1)}^{Training} - (1-\eta\gamma)\right] \\ \exp\left[-\eta V_{(k-1)}^{\hat{\lambda}} - (1-\eta\gamma)\right] \end{bmatrix}, \tag{3}$$

Using ordinary PINNs for Pre-Training The Model

- To detect change points better, we first train the neural network just over the training data.

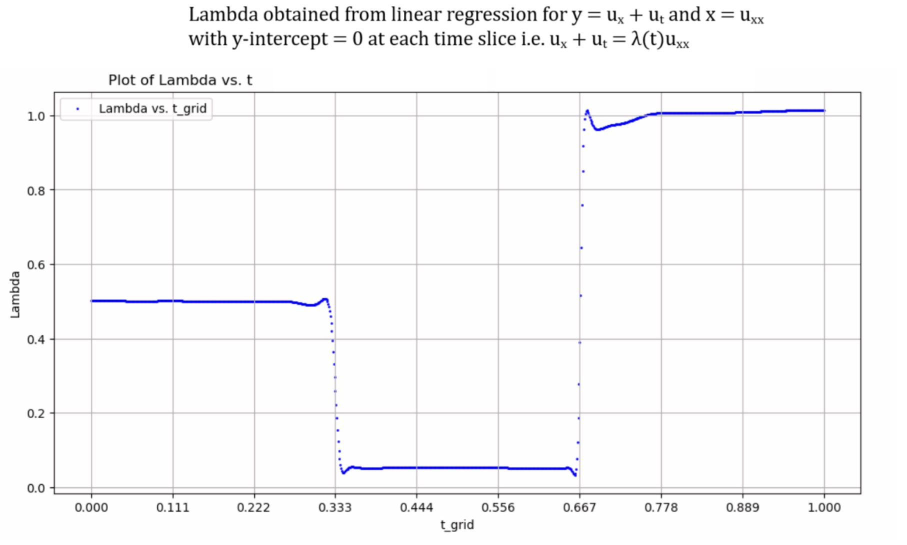


Figure. Lambda obtained from training data without the regularization term V^{λ}

Using ordinary PINNs for Pre-Training The Model

- To detect change points better, we first train the neural network just over the training data.
- Let $y = u_x + u_t$ and $x = u_{xx}$ and do a linear regression on $y = \lambda(t)x$ for each t with intercept 0.

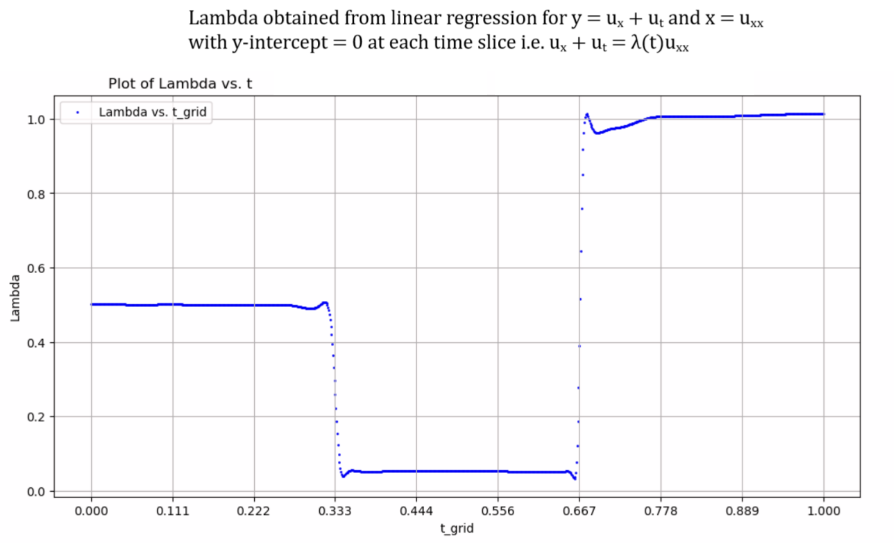


Figure. Lambda obtained from training data without the regularization term V^λ

Using ordinary PINNS for Pre-Training The Model

- To detect change points better, we first train the neural network just over the training data.
- Let $y = u_x + u_t$ and $x = u_{xx}$ and do a linear regression on $y = \lambda(t)x$ for each t with intercept 0.
- This gives a good estimate of $\lambda(t)$ and the change points.

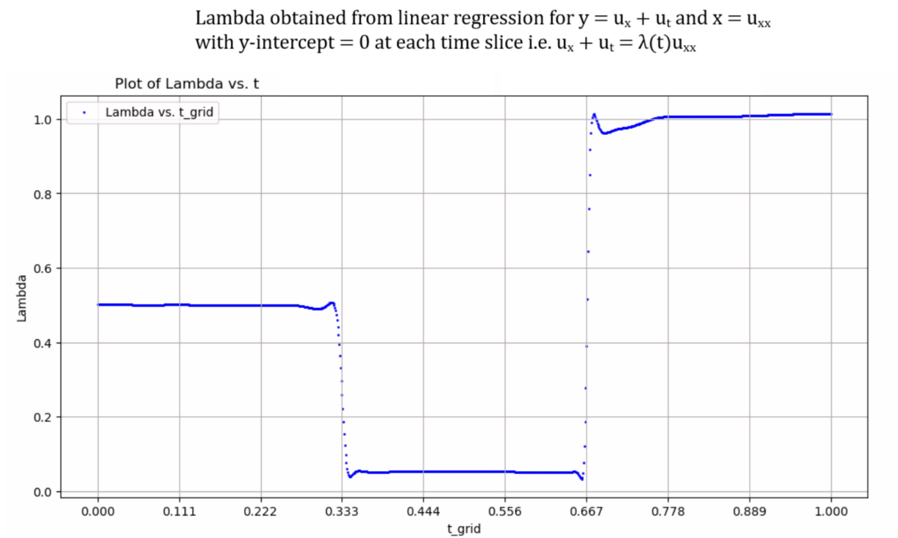


Figure. Lambda obtained from training data without the regularization term V^{λ}

Training The Model with Pre-Trained Parameters

- Using the pre-trained parameters as initial parameters in our CP-PINN algorithm, we obtain the following $\lambda(t)$

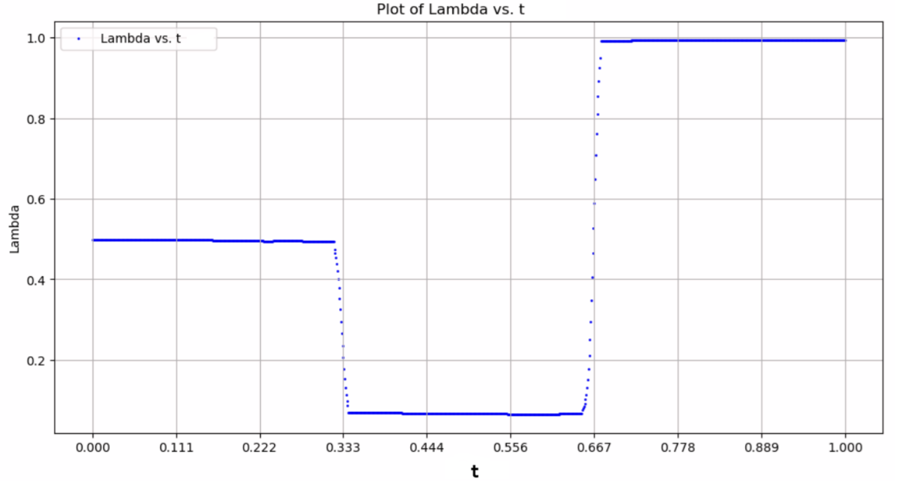


Figure. Lambda obtained from CP-PINN training.

Training the Model with Pre-Trained Parameters

- The absolute error between the training data and the neural network solution

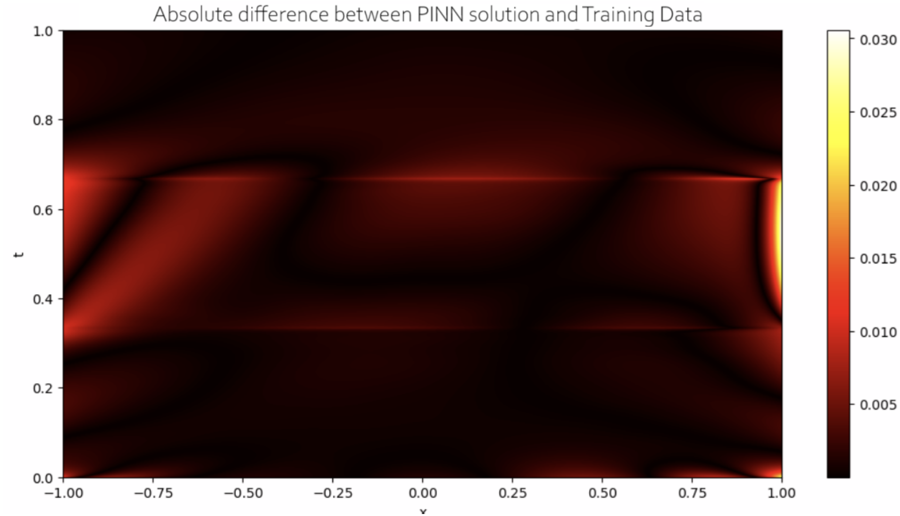


Figure. Lambda obtained from CP-PINN training.

Final Results

- Change points obtained are $t_1 = 0.33 \pm 0.01$ and $t_2 = 0.66 \pm 0.01$.

Final Results

- Change points obtained are $t_1 = 0.33 \pm 0.01$ and $t_2 = 0.66 \pm 0.01$.
- Values of $\lambda_1 = 0.5 \pm 0.0017$, $\lambda_2 = 0.05 \pm 0.0011$ and $\lambda_3 = 1.0 \pm 0.0007$

CP-PINNs: Summary and Next Steps

- Physics-informed neural networks can be used when we need to solve a physical problem like solving the advection equation with very little training data.

CP-PINNs: Summary and Next Steps

- Physics-informed neural networks can be used when we need to solve a physical problem like solving the advection equation with very little training data.
- When the parameters of the equation change drastically, CP-PINNs perform much better than ordinary PINNs in detecting the change points.

CP-PINNs: Summary and Next Steps

- Physics-informed neural networks can be used when we need to solve a physical problem like solving the advection equation with very little training data.
- When the parameters of the equation change drastically, CP-PINNs perform much better than ordinary PINNs in detecting the change points.
- Another training algorithm could be to use quadratic programming for optimizing the parameter $\lambda(t)$ for each batch and use Adam for the neural network parameters.

CP-PINNs: Summary and Next Steps

- Physics-informed neural networks can be used when we need to solve a physical problem like solving the advection equation with very little training data.
- When the parameters of the equation change drastically, CP-PINNs perform much better than ordinary PINNs in detecting the change points.
- Another training algorithm could be to use quadratic programming for optimizing the parameter $\lambda(t)$ for each batch and use Adam for the neural network parameters.
- One may use CP-PINNs in quantitative finance, for instance, to estimate points of high volatility if we think of high-volatility as change points.