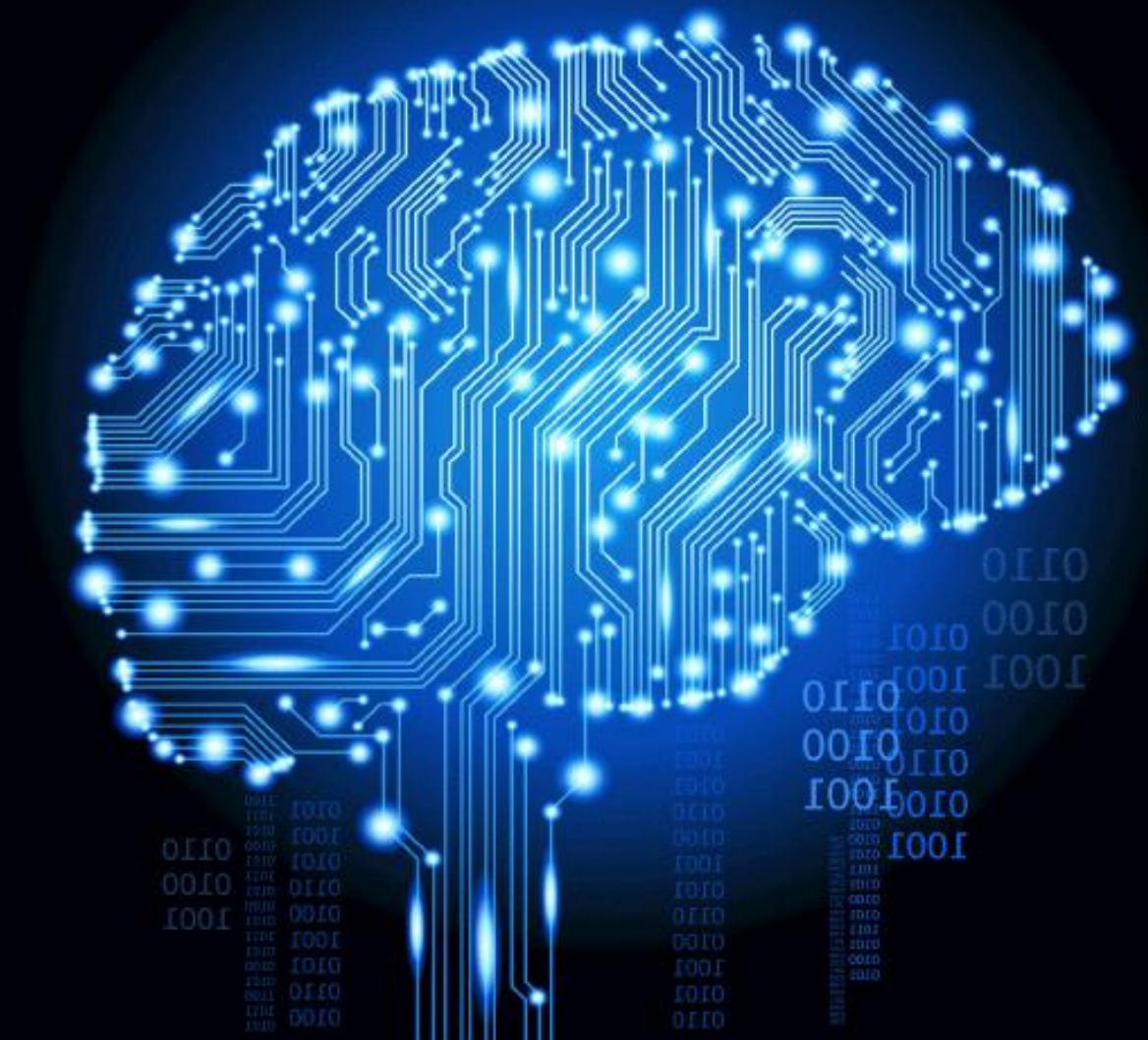
Theoretical Foundations of Physics-informed Machine Learning and and Deep Learning Models

by

Dr. Bivas Bhaumik
Department of Mathematics
National Institute of Technology, Rourkela



Assistant professor (contract), Department of Mathematics,
National Institute of Technology, Rourkela, India

Assistant professor, Department of Computer Science,
Institute of Engineering & Management, Kolkata, India

Ph.D. in Applied Mathematics
University of Calcutta, India

M.Sc. in Applied Mathematics
University of Calcutta

Important Links:

Google Scholar: FfG_kKQAAAAJ&hl=en&oi=ao
ORCID ID: 0000-0002-3168-2687
Primary Scopus ID: 57224404057
h-Index: 10
i10-Index: 10

What is Artificial Neural Network (ANN)?

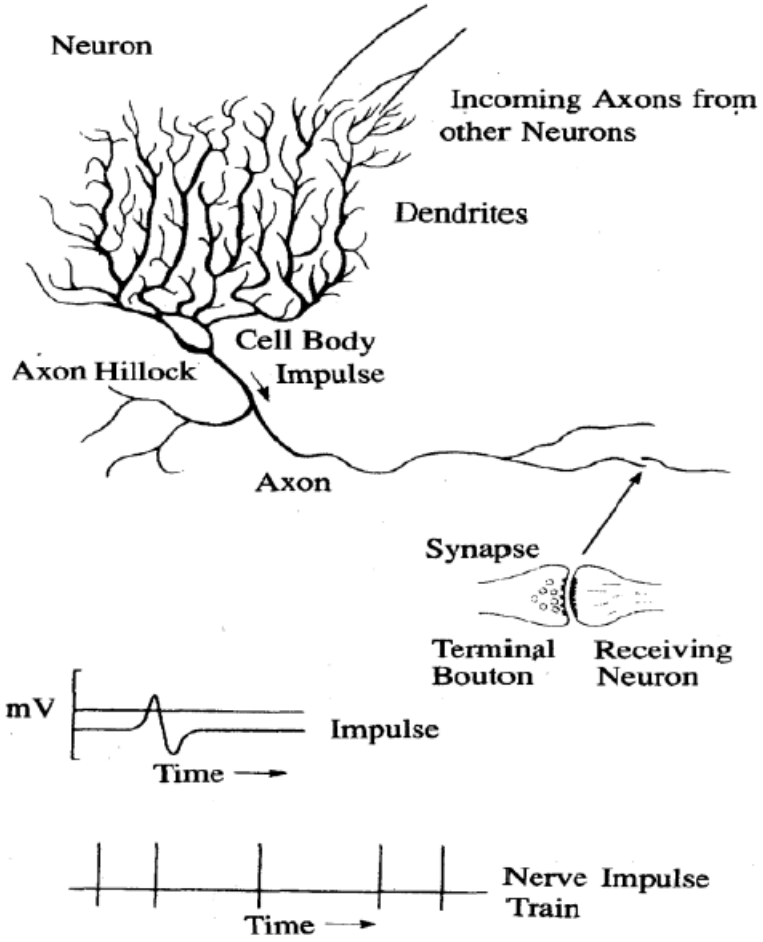


Fig. 1: Structure of the Biological Neural Network

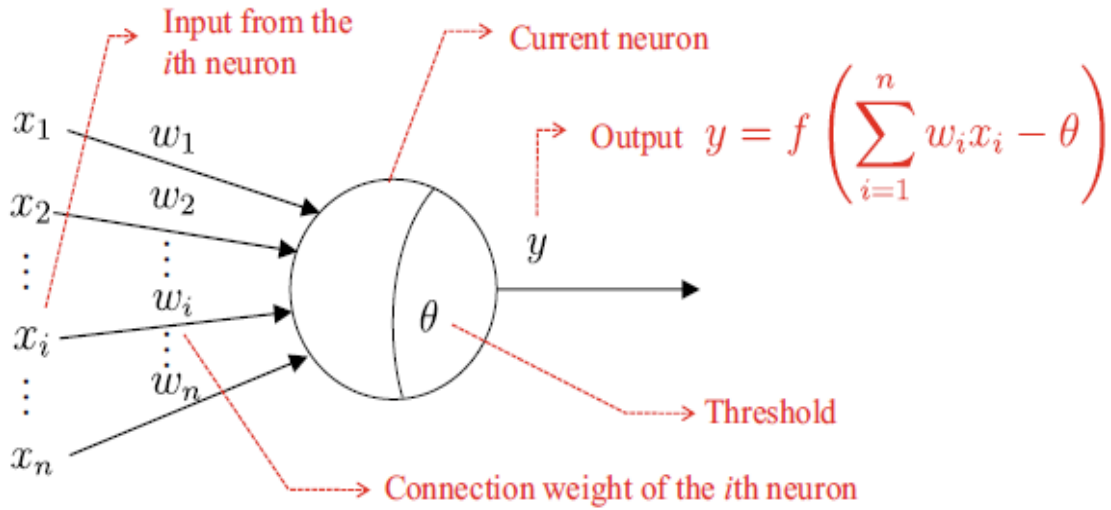


Fig. 2: Structure of the Artificial Neural Network

- The neurons when 'excited' send neurotransmitters to interconnected neurons.
- When electrical potential exceeds a threshold the neuron is active.
- The activated neuron will send neurotransmitters to the other neurons.

Universal Approximation Theorem and Neural Networks:

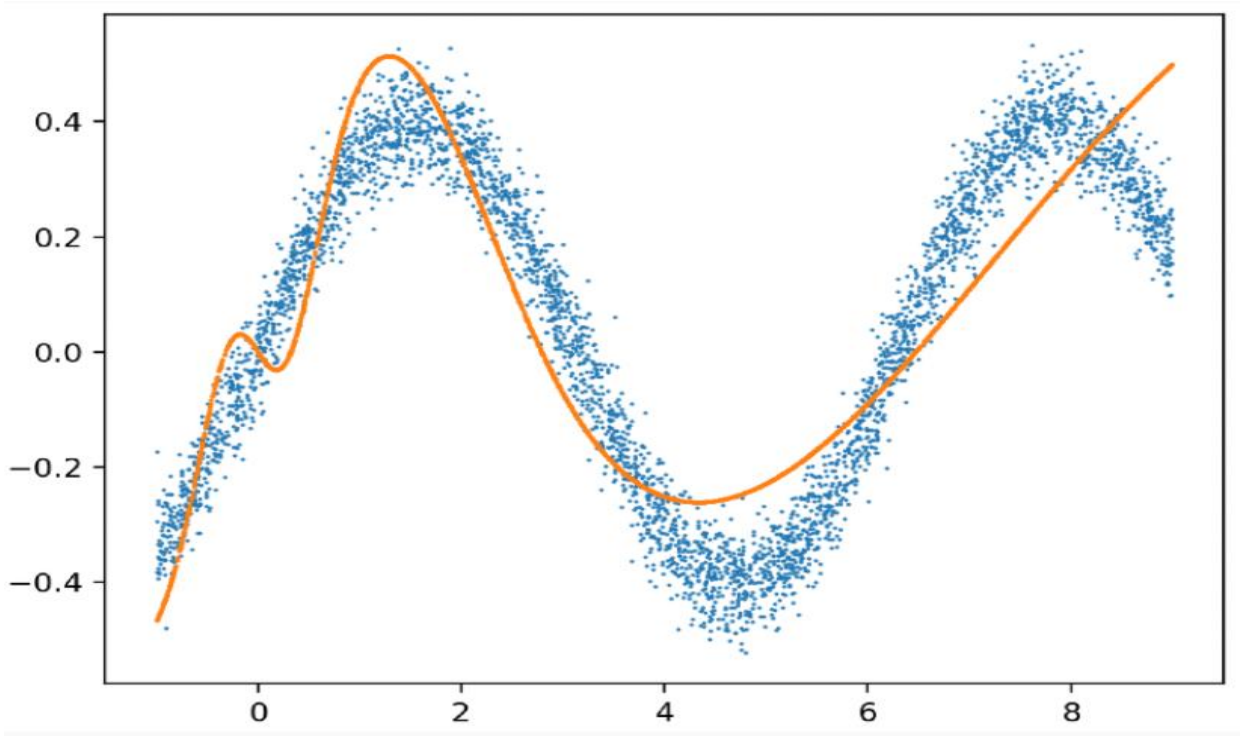
This theorem states that "a feedforward neural network with a single hidden layer and a suitable activation function can approximate any continuous function (linear or non-linear) from one finite-dimensional space to another with arbitrary precision, given enough hidden neurons".

- The idea behind both views is function approximation, and we want to emphasize more in function approximation to give you a more accurate description of supervised learning.

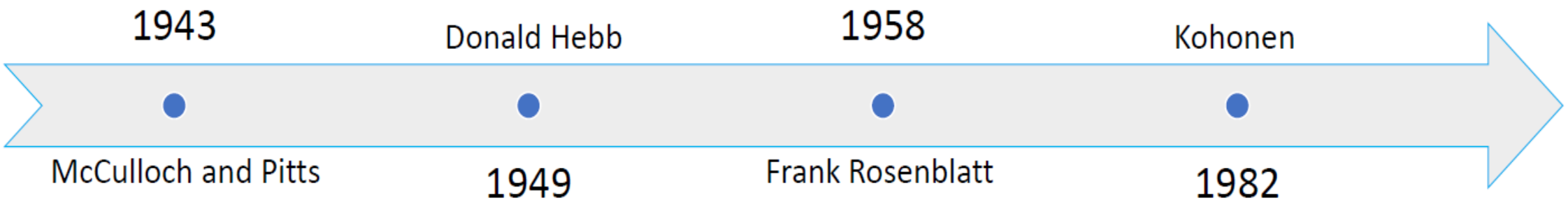
NN view:

- Gather training data $(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)$
- Choose a NN with k neurons, so that there is a group of weights, e.g., $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k, b_1, \ldots, b_k)$, to ensure $\boldsymbol{y}_i \approx \{\mathrm{NN}(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k, b_1, \ldots, b_k)\}(\boldsymbol{x}_i) \quad \forall i$
- Set up a loss function ℓ
- Find weights $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k, b_1, \ldots, b_k)$ to minimize the average loss

$$\min_{\boldsymbol{w}'s, b's} \frac{1}{n} \sum_{i=1}^{n} \ell\left[\boldsymbol{y}_i, \{\mathrm{NN}(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k, b_1, \ldots, b_k)\}(\boldsymbol{x}_i)\right]$$

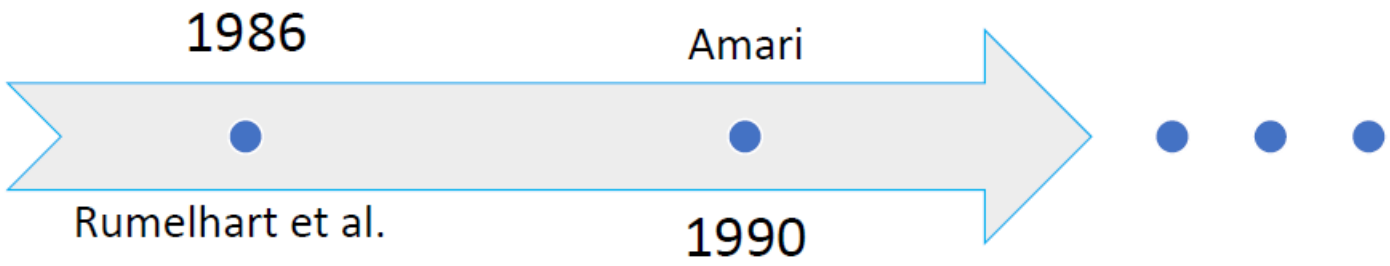


History of ANNs:



- McCulloch and Pitts (1943) outlined the first formal model of an elementary computing neuron. Therefore, year 1943 is often considered the initial year in the development of artificial neural systems.
- Donald Hebb (1949) first proposed a learning scheme for updating neuron's connections that we now refer to as the Hebbian learning rule. He stated that the information can be stored in connections and postulated the learning technique. Hebb's learning rule made primary contributions to neural networks theory.
- Frank Rosenblatt (1958) invented neuron-like element called perceptron. The idea caught the imagination of engineers and scientists and laid the groundwork for the basic machine learning algorithms that we still use today.
- Kohonen (1982) developed unsupervised learning networks for feature mapping into regular arrays of neurons.

Continue…



- Rumelhart et al. (1986) popularized and refined the backpropagation algorithm, leading to its widespread adoption.
- Amari (1990) defined the general learning rule which is adopted in neural network studies.

Importance of Activation Functions in ANNs

- The main aim of using activation functions is to introduce **non-linearity** in the neural network.
- Without nonlinearity, an ANN becomes a cascade of linear functions and effectively behaves as a linear model.

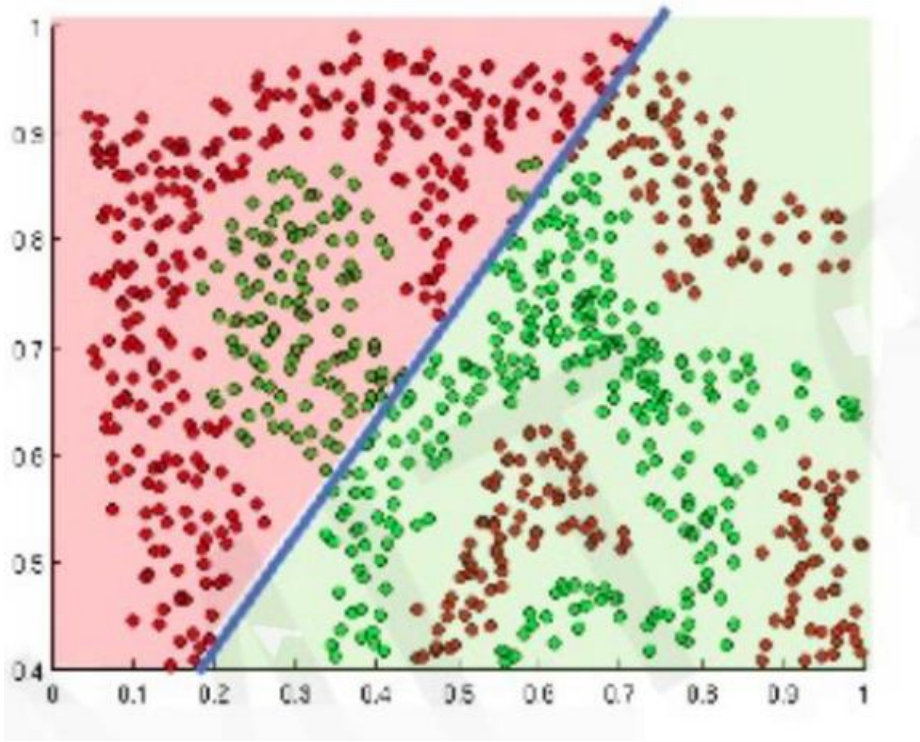


Fig. 3: Linear activation functions always produce linear decision boundaries, regardless of how large or deep the network is.

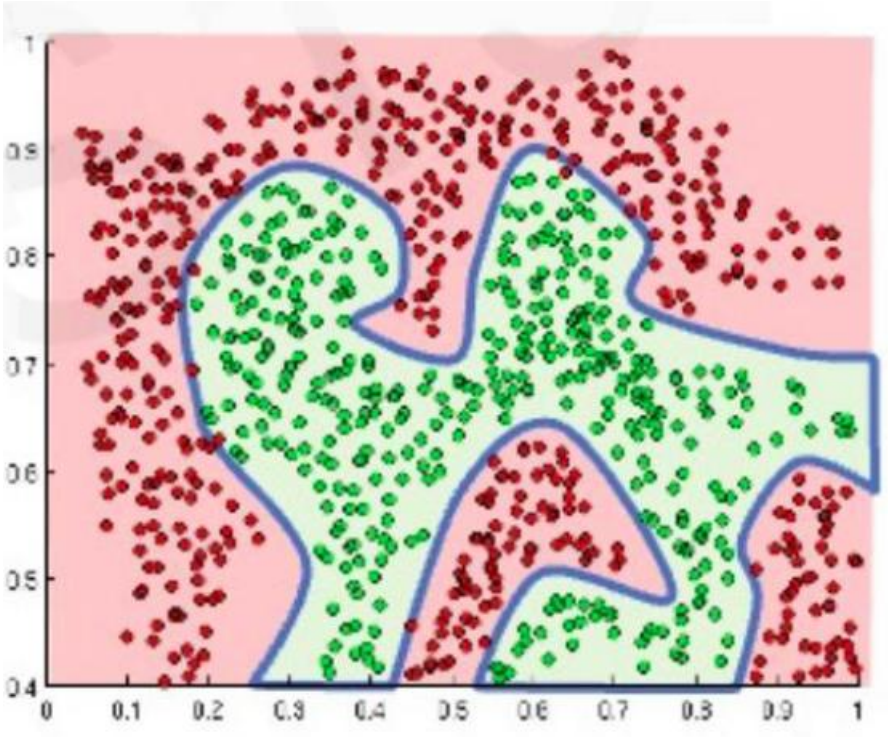
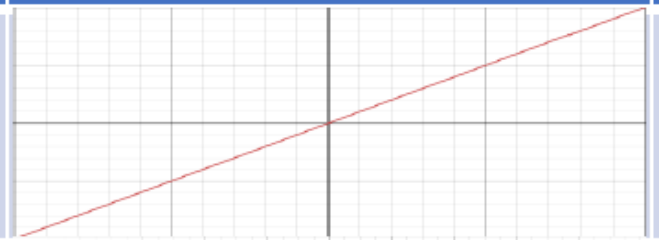
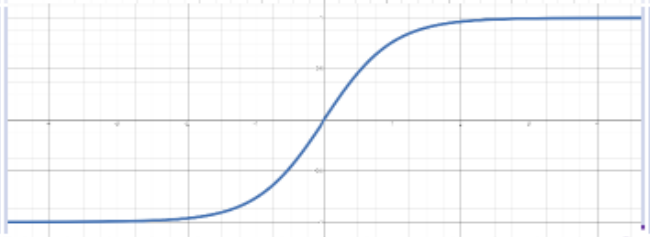


Fig. 4: Introducing non-linear activation functions enables the network to approximate arbitrary and highly complex functions.

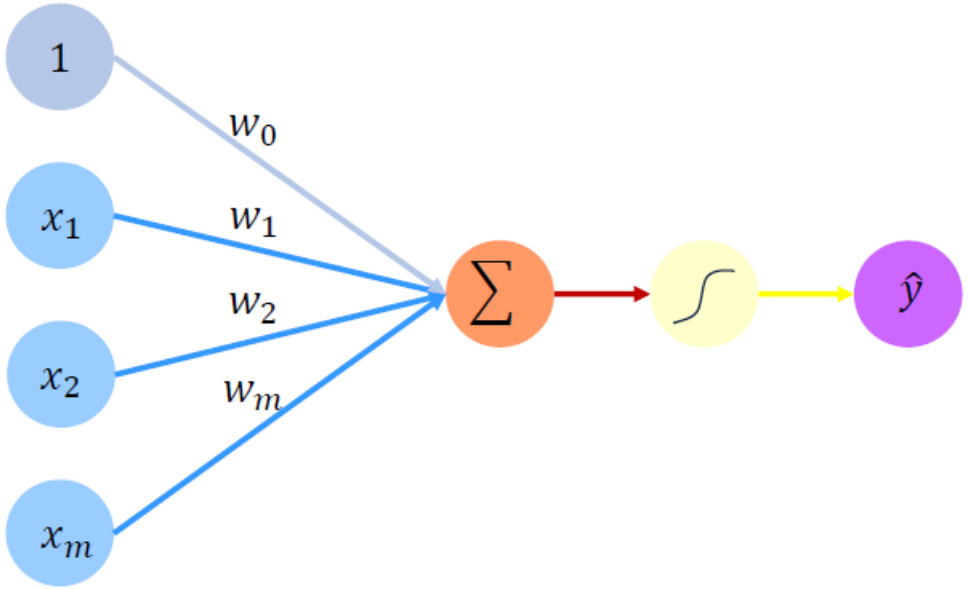
Continue…

Table 1. Some of the common Activation Functions

Name	Plot	f(x)	f'(x)
Linear		$f(x) = x$	$f'(x) = 1$
Sigmoid		$f(x) = \frac{1}{1 + e^{-\lambda x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$	$f'(x) = 1 - f(x)^2$
ReLU		$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$

Perceptron:

- The perceptron means the "structural building block" of a neural network.



Inputs Weights Sum Non-linearity Output

Linear combination of inputs

Output

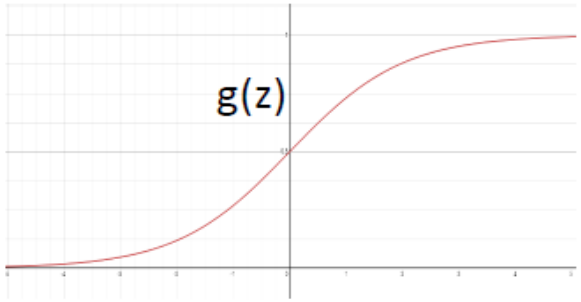
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} w_i\, x_i\right)$$

Non-linear activation function

Bias

- Example: Sigmoid Function

$$g(z) = \sigma(z) = \frac{1}{1 - e^{-z}}$$



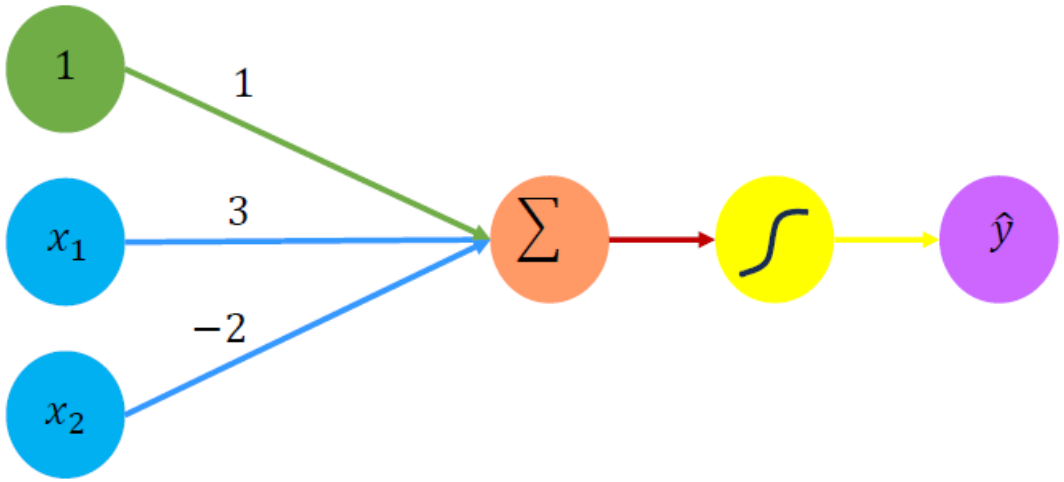
Forward propagation:

$$\hat{y} = g\,(w_0 + \boldsymbol{W}^T \boldsymbol{X})$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$

Continue…

Example 1:



Suppose, We have: $w_0 = 1$ and $W = [3, -2]$

$$\hat{y} = g(w_0 + W^T X) = g\left(1 + [3 \quad -2].\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right)$$

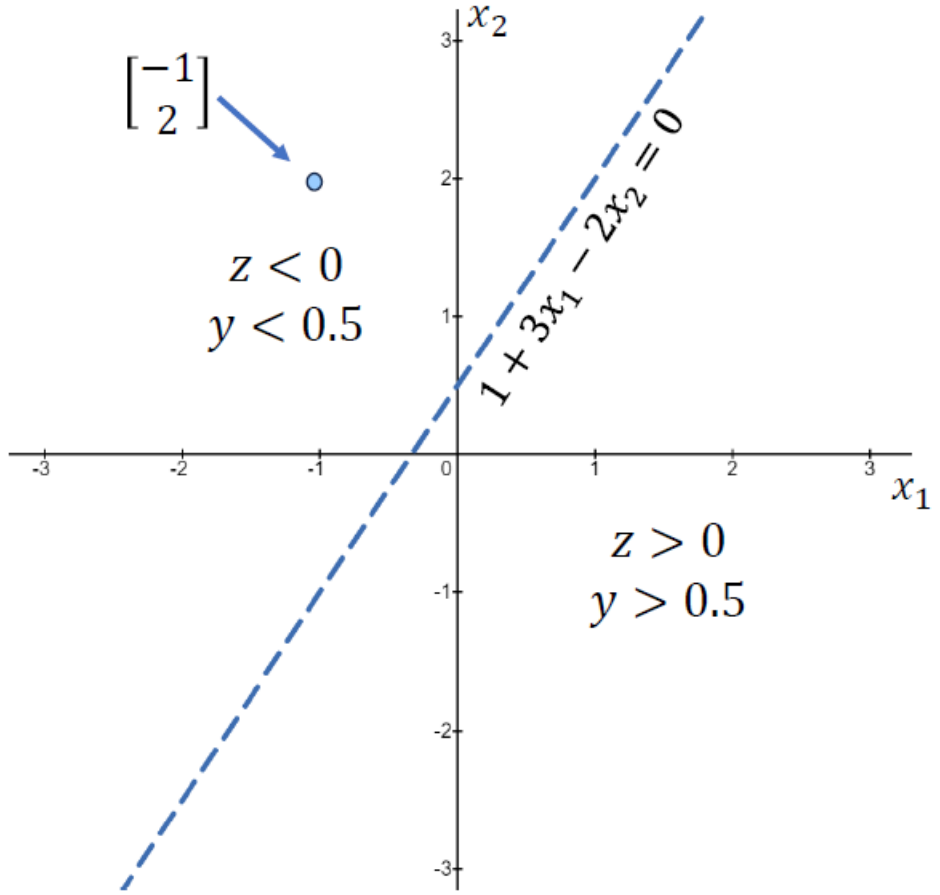
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

This is just a line in 2D!

Assume we have input: $X = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

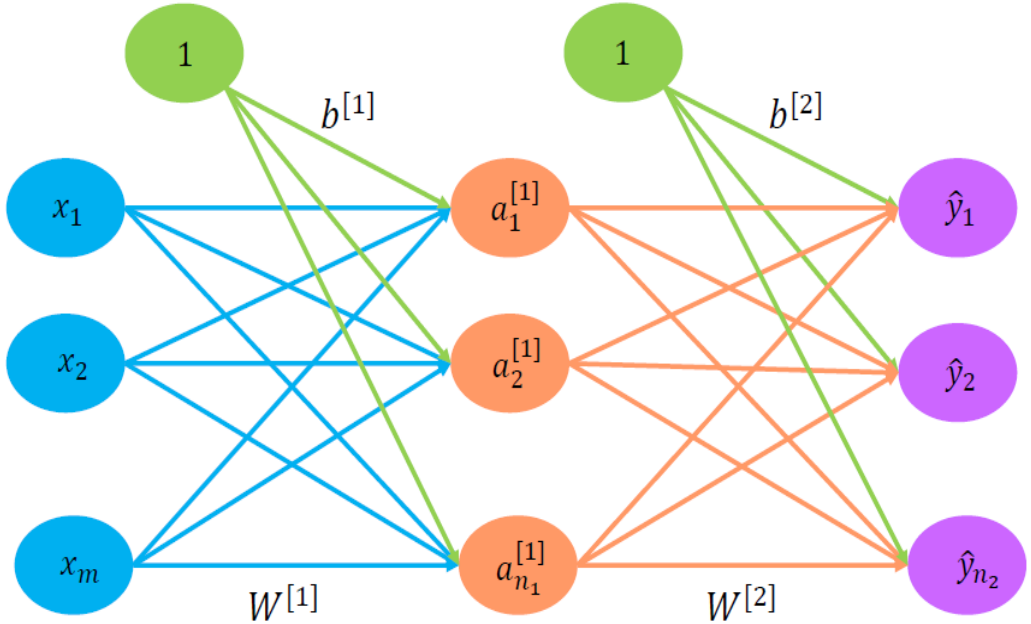
$$\hat{y} = g\big(1 + (3 * -1) - (2 * 2)\big) = g(-6) \approx 0.002$$

$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



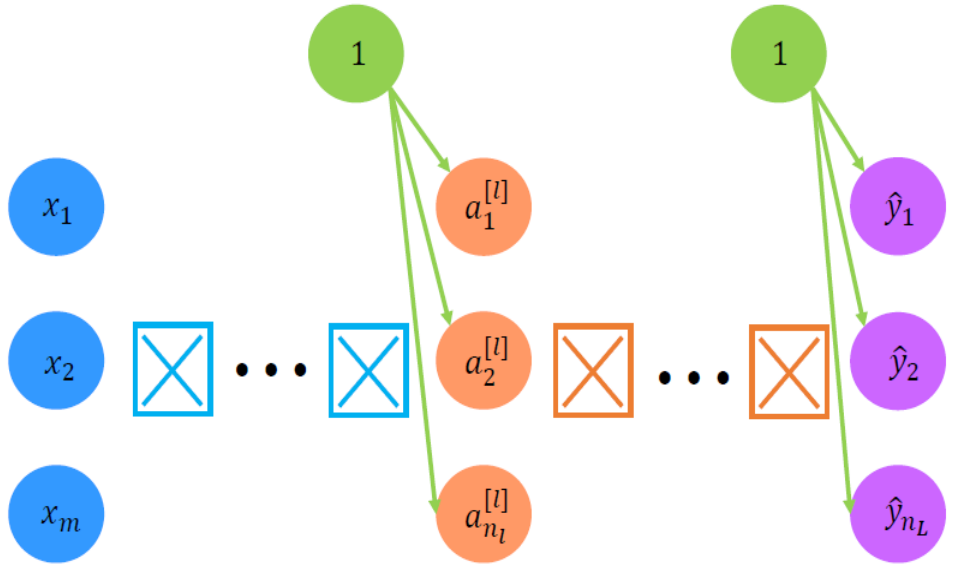
ANN Vs DNN:

Single Layer Neural Network



$$z_j^{[1]} = b_j^{[1]} + \sum_{i=1}^{m} w_{j,i}^{[1]} x_i \qquad a_j^{[1]} = g\left(z_j^{[1]}\right) \qquad \hat{y}_k = g\left(b_k^{[2]} + \sum_{j=1}^{n_1} w_{k,j}^{[2]} a_j^{[1]}\right)$$

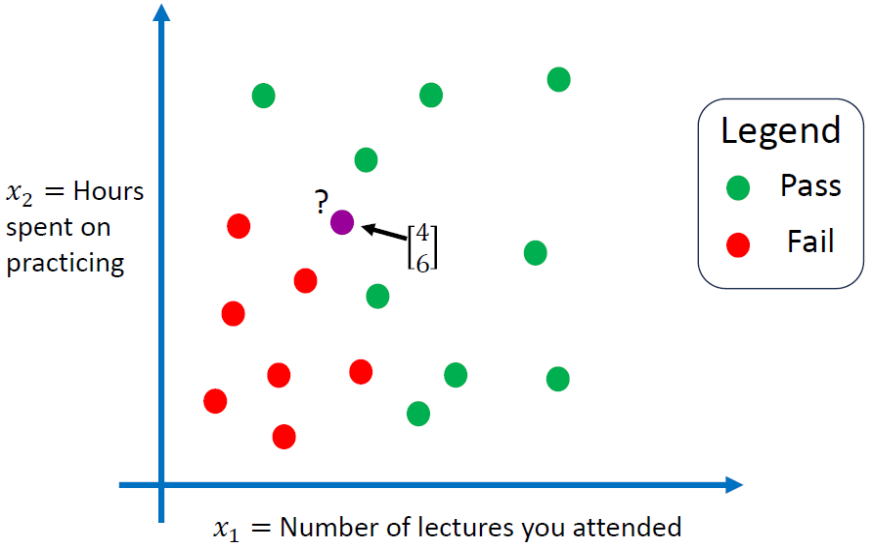
Deep Neural Network

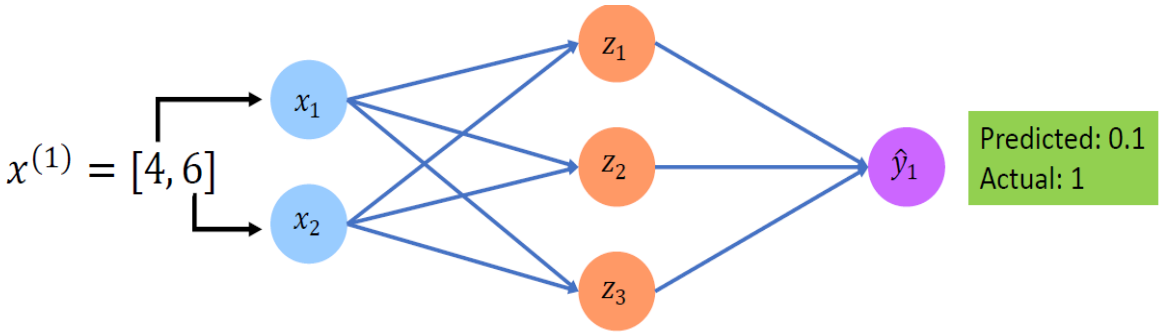


$$z_j^{[l]} = b_j^{[l]} + \sum_{i=1}^{n_{l-1}} w_{j,i}^{[l]} a_i^{[l-1]} \qquad \hat{y}_k = g\left(w_{0,k}^{[L]} + \sum_{j=1}^{n_{L-1}} w_{j,k}^{[L]} g(z_j^{[L-1]})\right)$$

Loss and Cost Functions

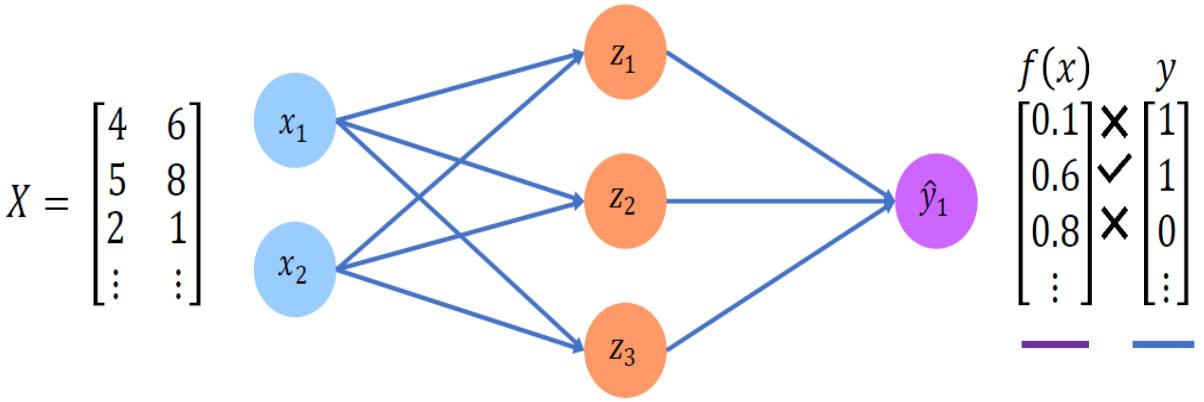
Example 2: Will I pass the test?





Empirical Loss or Cost Function:

- The measure of total loss over the entire training dataset.



Also known as:
- Objective function
- Cost function
- Empirical Risk

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left((x^{(i)}; \boldsymbol{W}), y^{(i)}\right)$$

Predicted Actual

- The ***loss*** of our network measures the cost incurred from incorrect predictions: $\mathcal{L}\left((x^{(i)}; \boldsymbol{W}), y^{(i)}\right)$

Predicted Actual

Continue…

Loss Functions for Regression:

- ***Mean Squared Loss (MSE)*** function can be used for regression problem.

$$J(W) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f(x^{(i)}; \boldsymbol{W})\right)^2$$

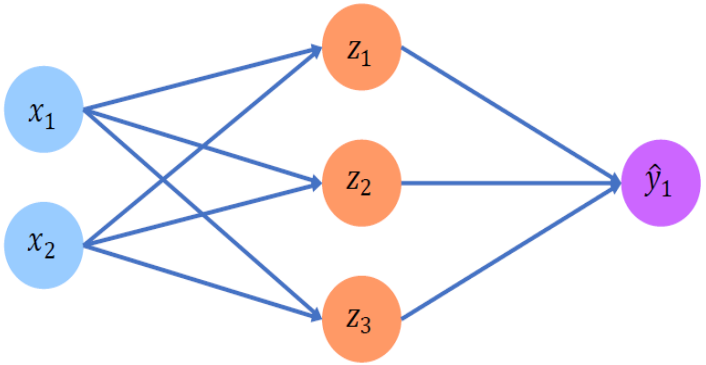
Actual Predicted

Loss Functions for Classification:

- ***Binary Cross Entropy Loss function*** can be used for binary classification problem.
- This provides us an output with probability values between 0 & 1.
- $J(W) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log\left(f(x^{(i)}; \boldsymbol{W})\right) + \left(1 - y^{(i)}\right) \log\left(1 - f(x^{(i)}; \boldsymbol{W})\right)$

Actual Predicted Actual Predicted

$$X = \begin{bmatrix} 4 & 6 \\ 5 & 8 \\ 2 & 1 \\ \vdots & \vdots \end{bmatrix}$$



$f(x)$ y

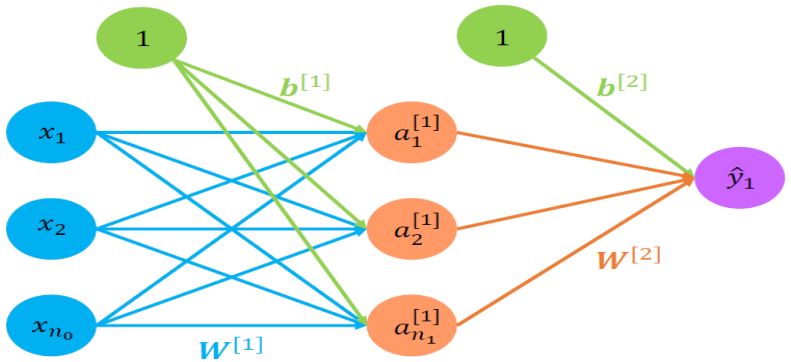
$$\begin{bmatrix} 0.1 \\ 0.6 \\ 0.8 \\ \vdots \end{bmatrix} \begin{matrix} \times \\ \checkmark \\ \times \\ \end{matrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ \vdots \end{bmatrix}$$

- The ***multi-class cross-entropy loss*** *J* for each training example being:

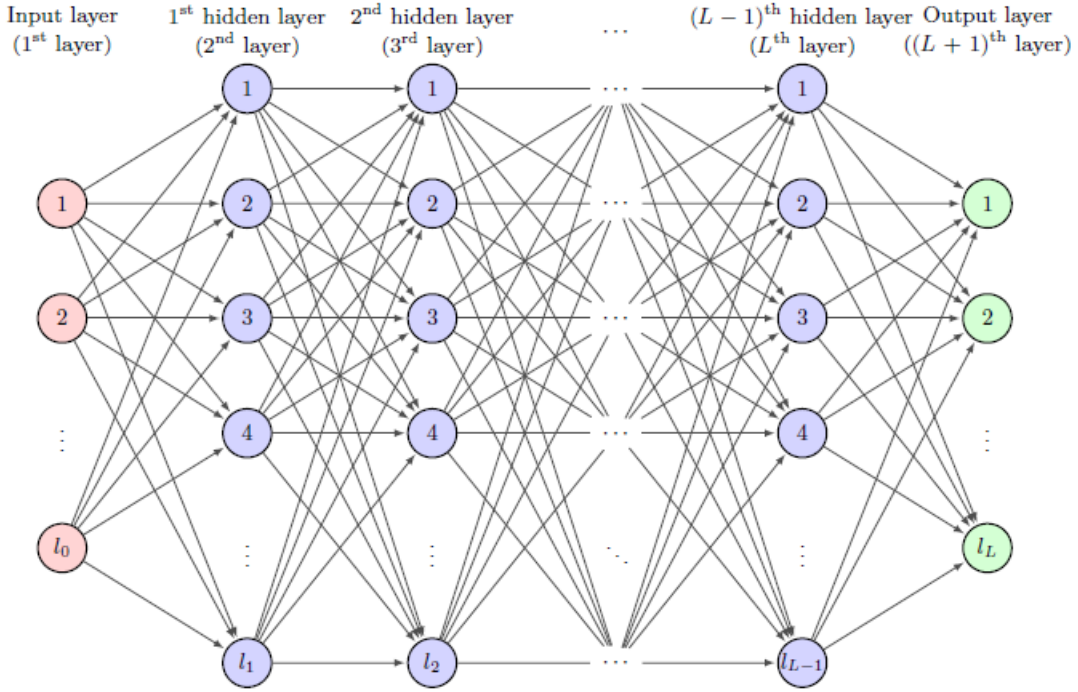
$$J(W) = -\sum_{k}^{K} y^{(k)} \log \hat{y}^{(k)}$$

Matrix Representation of Neural Networks

Artificial Neural Network:



Multi-layered Neural Network



$$a_i^{[0]} = x_i$$

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ \vdots \\ z_{n_1}^{[1]} \end{bmatrix} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} & \cdots & w_{1n_0}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} & \cdots & w_{2n_0}^{[1]} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_1 1}^{[1]} & w_{n_1 2}^{[1]} & \cdots & w_{n_1 n_0}^{[1]} \end{bmatrix} \begin{bmatrix} a_1^{[0]} \\ a_2^{[0]} \\ \vdots \\ a_{n_0}^{[0]} \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_{n_1}^{[1]} \end{bmatrix}$$

$$\boldsymbol{z}^{[1]} = \boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}$$

$$\boldsymbol{a}^{[1]} = g\left(\boldsymbol{z}^{[1]}\right) \quad \text{OR} \quad \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_{n_0}^{[1]} \end{bmatrix} = g\left(\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ \vdots \\ z_{n_1}^{[1]} \end{bmatrix}\right)$$

Similarly

$$\boldsymbol{z}^{[2]} = \boldsymbol{W}^{[2]}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$$
$$\hat{y} = \boldsymbol{a}^{[2]} = g\left(\boldsymbol{z}^{[2]}\right)$$

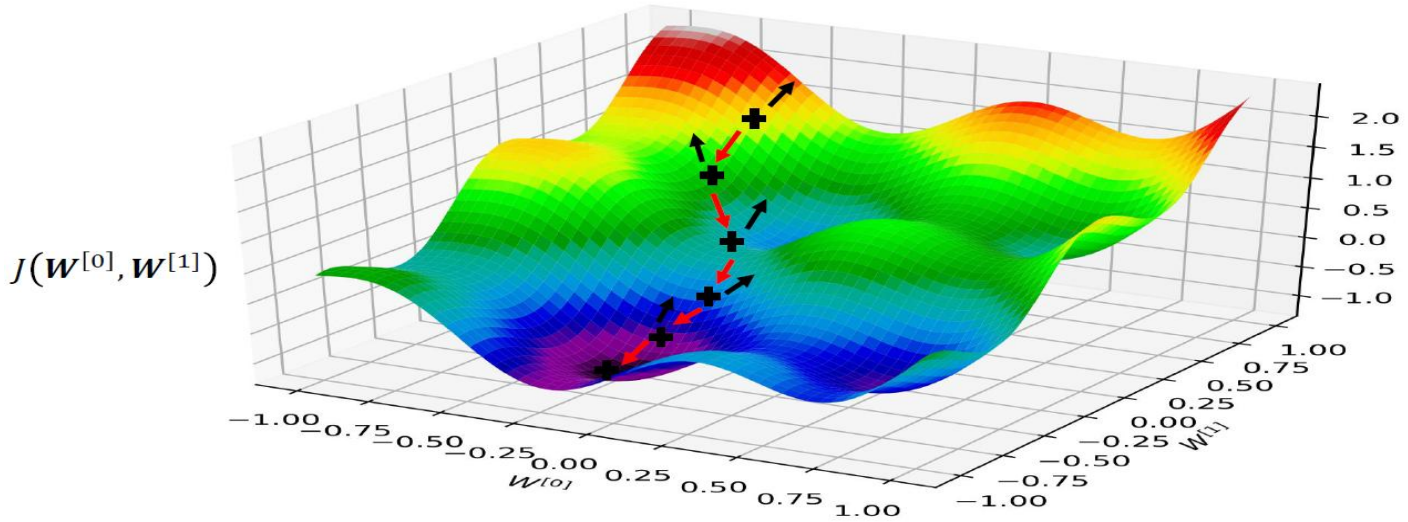
- ***Multi-layered Neural Network*** is a cascade of composition of multiple activation functions.

Gradient Descent in ANN

Take small step in opposite direction of gradient and repeat until convergence

Algorithm:

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
   a. Compute gradient, $\frac{\partial J(W)}{\partial W}$
   b. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
3. Return weights



Forward and Backpropagation in ANN:

Forward Propagation

$$J = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

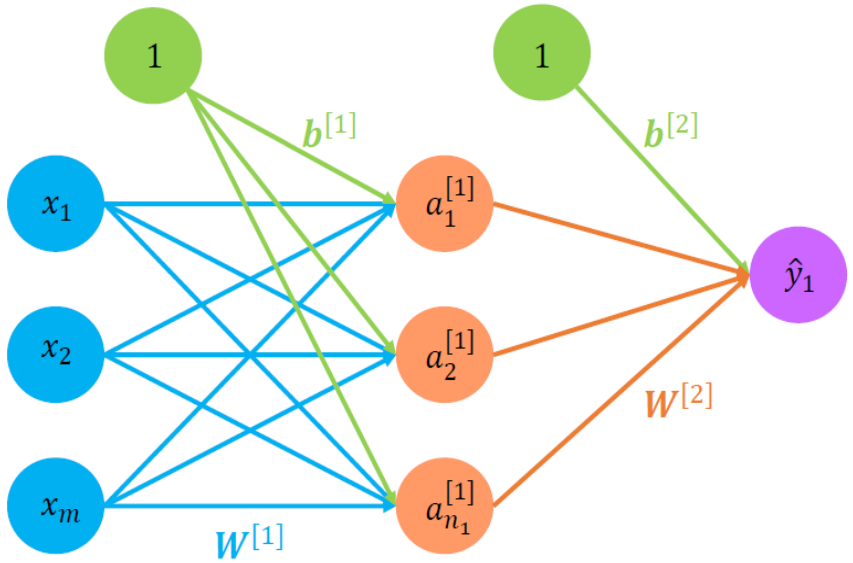
$$Z^{[1]} = W^{[1]}X + b^{[1]} \rightarrow A^{[1]} = g(Z^{[1]}) \rightarrow Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]} \rightarrow \hat{Y} = A^{[2]} = g(Z^{[2]}) \rightarrow J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]})$$

Backpropagation

$$\frac{\partial J}{\partial \mathcal{L}} \leftarrow \frac{\partial \mathcal{L}}{A^{[2]}} \leftarrow \frac{\partial A^{[2]}}{\partial Z^{[2]}} \leftarrow \frac{\partial Z^{[2]}}{\partial A^{[1]}} \leftarrow \frac{\partial A^{[1]}}{\partial Z^{[1]}} \leftarrow \frac{\partial Z^{[1]}}{\partial W^{[1]}}$$

$$\frac{\partial Z^{[2]}}{\partial W^{[2]}}$$

Continue…



$$J\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(y^{(i)}, \hat{y}^{(i)}\right)$$

$a^{[2](i)}$

Forward Propagation

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = g(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g(Z^{[2]})$$

$$Z^{[1]}_{n_1\times m} = W^{[1]}_{n_1\times n_0} X_{n_0\times m} + b^{[1]}_{n_1\times 1}$$
$$A^{[1]}_{n_1\times m} = g\left(Z^{[1]}_{n_1\times m}\right)$$
$$\vdots$$
$$Z^{[l]}_{n_l\times m} = W^{[l]}_{n_l\times n_{l-1}} A^{[l-1]}_{n_{l-1}\times m} + b^{[l]}_{n_l\times 1}$$
$$A^{[l]}_{n_l\times m} = g\left(Z^{[l]}_{n_l\times m}\right)$$
$$\vdots$$
$$Z^{[L]}_{n_L\times m} = W^{[L]}_{n_L\times n_{L-1}} A^{[L-1]}_{n_{L-1}\times m} + b^{[L]}_{n_L\times 1}$$
$$\hat{Y}_{n_L\times m} = A^{[L]}_{n_L\times m} = g\left(Z^{[L]}_{n_L\times m}\right)$$

⟵ 1st Layer

⟵ lth Layer

⟵ Lth Layer

Computing gradients in Backpropagation:

$$\frac{\partial J}{W^{[L]}_{n_L \times n_{L-1}}} = \left(\frac{\partial J}{\partial \mathcal{L}} \cdot \frac{\partial \mathcal{L}}{A^{[L]}_{n_L \times m}} * \frac{\partial A^{[L]}}{\partial Z^{[L]}_{n_L \times m}} \right) \cdot \frac{\partial Z^{[L]}_{n_L \times m}}{\partial W^{[L]}_{n_L \times n_{L-1}}}$$

$$dW^{[L]}_{n_L \times n_{L-1}} = dZ^{[L]}_{n_L \times m} \cdot A^{[L-1]^T}_{m \times n_{L-1}}$$

BP of layer L

$$\frac{\partial J}{W^{[L-1]}_{n_{L-1} \times n_{L-2}}} = \left(\frac{\partial J}{\partial \mathcal{L}} \cdot \frac{\partial \mathcal{L}}{A^{[L]}_{n_L \times m}} * \frac{\partial A^{[L]}}{\partial Z^{[L]}_{n_L \times m}} \right) \cdot \frac{\partial Z^{[L]}_{n_L \times m}}{\partial A^{[L-1]}_{n_{L-1} \times m}} \cdot \frac{\partial A^{[L-1]}}{\partial Z^{[L-1]}_{n_{L-1} \times m}} \cdot \frac{\partial Z^{[L-1]}_{n_L \times m}}{\partial W^{[L-1]}_{n_{L-1} \times n_{L-2}}}$$

$$dW^{[L-1]}_{n_{L-1} \times n_{L-2}} = \left(\left(W^{[L]^T}_{n_{L-1} \times n_L} \cdot dZ^{[L]}_{n_L \times m} \right) * g' \left(Z^{[L-1]}_{n_{L-1} \times m} \right) \right) \cdot A^{[L-2]^T}_{m \times n_{L-2}}$$

$$dW^{[L-1]}_{n_{L-1} \times n_{L-2}} = dZ^{[L-1]}_{n_{L-1} \times m} \cdot A^{[L-2]^T}_{m \times n_{L-2}}$$

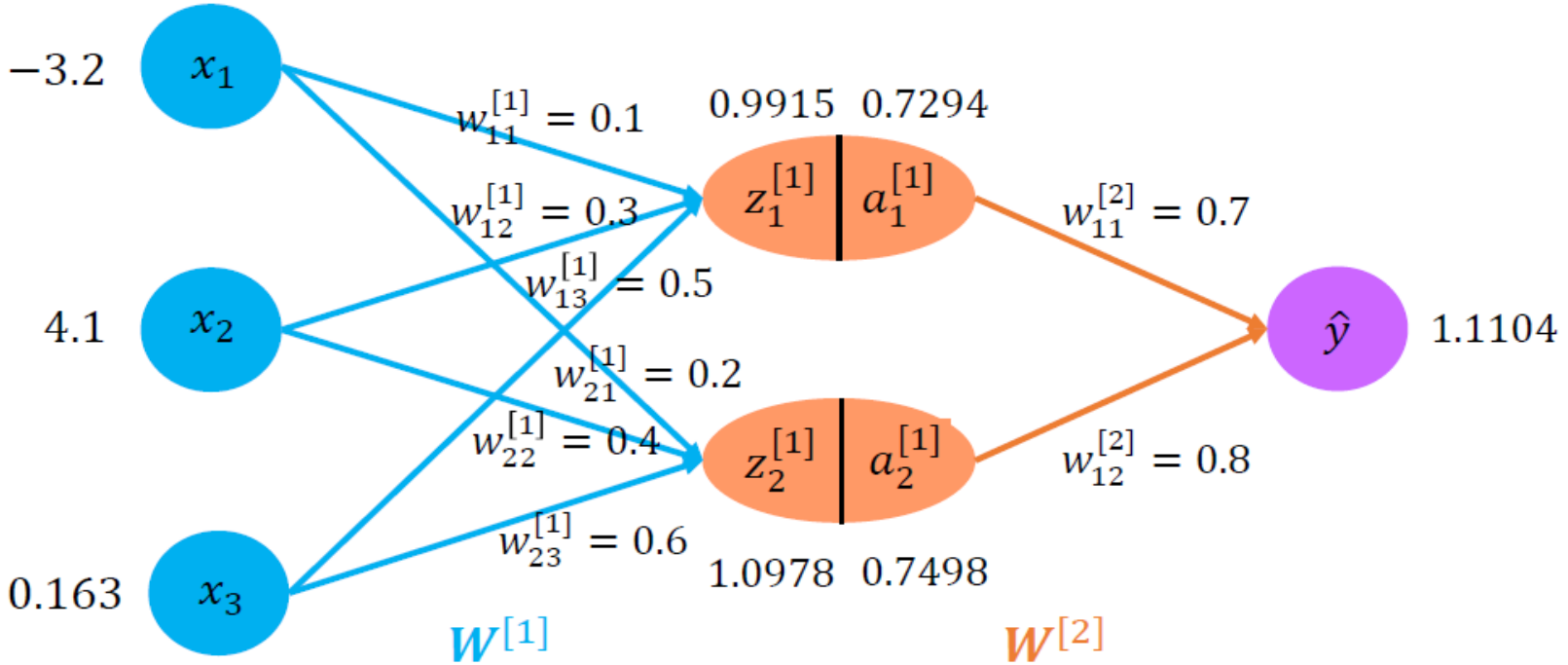
BP of layer $L-1$

$$dW^{[l]}_{n_l \times n_{l-1}} = \left(\left(W^{[l+1]^T}_{n_l \times n_{l+1}} \cdot dZ^{[l+1]}_{n_{l+1} \times m} \right) * g' \left(Z^{[l]}_{n_l \times m} \right) \right) \cdot A^{[l-1]^T}_{m \times n_{l-1}}$$

$$dW^{[l]}_{n_l \times n_{l-1}} = dZ^{[l]}_{n_l \times m} \cdot A^{[l-1]^T}_{m \times n_{l-1}}$$

BP of layer l, $l = 1,2, \dots, L-1$

Example



$$W^{[1]}x = z^{[1]} \Rightarrow \begin{bmatrix} 0.1 & 0.3 & 0.5 \\ 0.2 & 0.4 & 0.6 \end{bmatrix} \begin{bmatrix} -3.2 \\ 4.1 \\ 0.163 \end{bmatrix} = \begin{bmatrix} 0.9915 \\ 1.0978 \end{bmatrix}$$

$$a^{[1]} = \sigma(z^{[1]}) \Rightarrow \sigma\left(\begin{bmatrix} 0.9915 \\ 1.0978 \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{1+e^{-0.9915}} \\ \frac{1}{1+e^{-1.0978}} \end{bmatrix} = \begin{bmatrix} 0.7294 \\ 0.7498 \end{bmatrix}$$

$$z^{[2]} = W^{[2]}a^{[1]} = \hat{y} \Rightarrow \begin{bmatrix} 0.7 & 0.8 \end{bmatrix} \begin{bmatrix} 0.7294 \\ 0.7498 \end{bmatrix} = 1.1104$$

Continue…

$$MSE = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(1.1104 - 2)^2 = 0.3957 \ (Actual\ value = 2, say)$$

Task: To minimise MSE

For that we must find gradient of cost function with respect to weights of layers

Here, cost function $J = \frac{1}{2}(\hat{y} - y)^2$

$$\frac{\partial J}{\partial w^{[1]}} = \begin{bmatrix} \frac{\partial J}{\partial w_{11}^{[1]}} & \frac{\partial J}{\partial w_{12}^{[1]}} & \frac{\partial J}{\partial w_{13}^{[1]}} \\ \frac{\partial J}{\partial w_{21}^{[1]}} & \frac{\partial J}{\partial w_{22}^{[1]}} & \frac{\partial J}{\partial w_{23}^{[1]}} \end{bmatrix}$$

$$\frac{\partial J}{\partial w^{[2]}} = \begin{bmatrix} \frac{\partial J}{\partial w_{11}^{[2]}} & \frac{\partial J}{\partial w_{12}^{[2]}} \end{bmatrix}$$

Continue…

$$\frac{\partial J}{W^{[2]}_{1\times 2}} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z^{[2]}}.\frac{\partial z^{[2]}}{\partial W^{[2]}_{1\times 2}} = (\hat{y}-y).\,a^{[1]^T} = -0.8896\begin{bmatrix}0.7294\\0.7498\end{bmatrix}^T = [-0.6489 \quad -0.667]$$

$$\frac{\partial J}{W^{[1]}_{2\times 3}} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z^{[2]}}.\frac{\partial z^{[2]}}{\partial a^{[1]}}.\frac{\partial a^{[1]}}{\partial z^{[1]}_{2\times 1}}.\frac{\partial z^{[1]}_{2\times 1}}{\partial W^{[1]}_{2\times 3}} = (\hat{y}-y)\left(W^{[2]^T} * \sigma'(z^{[1]})\right)_{2\times 1}.x^T_{1\times 3}$$

$$= -0.8896\begin{bmatrix}0.7\\0.8\end{bmatrix} * \begin{bmatrix}a_1^{[1]}\left(1-a_1^{[1]}\right)\\a_2^{[1]}\left(1-a_2^{[1]}\right)\end{bmatrix}x^T = -0.8896\begin{bmatrix}0.7\\0.8\end{bmatrix} * \begin{bmatrix}0.7294(1-0.7294)\\0.7498(1-0.7498)\end{bmatrix}x^T$$

$$= -0.8896\begin{bmatrix}0.7\\0.8\end{bmatrix} * \begin{bmatrix}0.1974\\0.1876\end{bmatrix}x^T = -0.8896\begin{bmatrix}0.1382\\0.1501\end{bmatrix}[-3.2 \quad 4.1 \quad 0.163] = \begin{bmatrix}0.3934 & -0.504 & -0.02\\0.4273 & -0.5475 & -0.022\end{bmatrix}$$

Continue…

Now we will update the weights using gradient decent algorithm

$$W^{[1]} = W^{[1]} - 0.5\frac{\partial J}{\partial w^{[1]}}; \text{Learning rate} = 0.5, \text{say}$$

$$W^{[2]} = W^{[2]} - 0.5\frac{\partial J}{\partial W^{[2]}}$$

$$W^{[1]} = \begin{bmatrix} 0.1 & 0.3 & 0.5 \\ 0.2 & 0.4 & 0.6 \end{bmatrix} - 0.5 \begin{bmatrix} 0.3934 & -0.504 & -0.02 \\ 0.4273 & -0.5475 & -0.022 \end{bmatrix} = \begin{bmatrix} -0.0967 & 0.552 & 0.51 \\ -0.0137 & 0.6738 & 0.611 \end{bmatrix}$$

$$W^{[2]} = [0.7 \quad 0.8] - 0.5[-0.6489 \quad -0.667] = [1.0245 \quad 1.1335]$$

Continue…

$$W^{[1]}x = z^{[1]} \Rightarrow \begin{bmatrix} -0.0967 & 0.552 & 0.51 \\ -0.0137 & 0.6738 & 0.611 \end{bmatrix} \begin{bmatrix} -3.2 \\ 4.1 \\ 0.163 \end{bmatrix} = \begin{bmatrix} 2.6558 \\ 2.9056 \end{bmatrix}$$

$$a^{[1]} = \sigma(z^{[1]}) \Rightarrow \sigma\left(\begin{bmatrix} 2.6558 \\ 2.9056 \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{1+e^{-2.6558}} \\ \frac{1}{1+e^{-2.9056}} \end{bmatrix} = \begin{bmatrix} 0.9344 \\ 0.9481 \end{bmatrix}$$

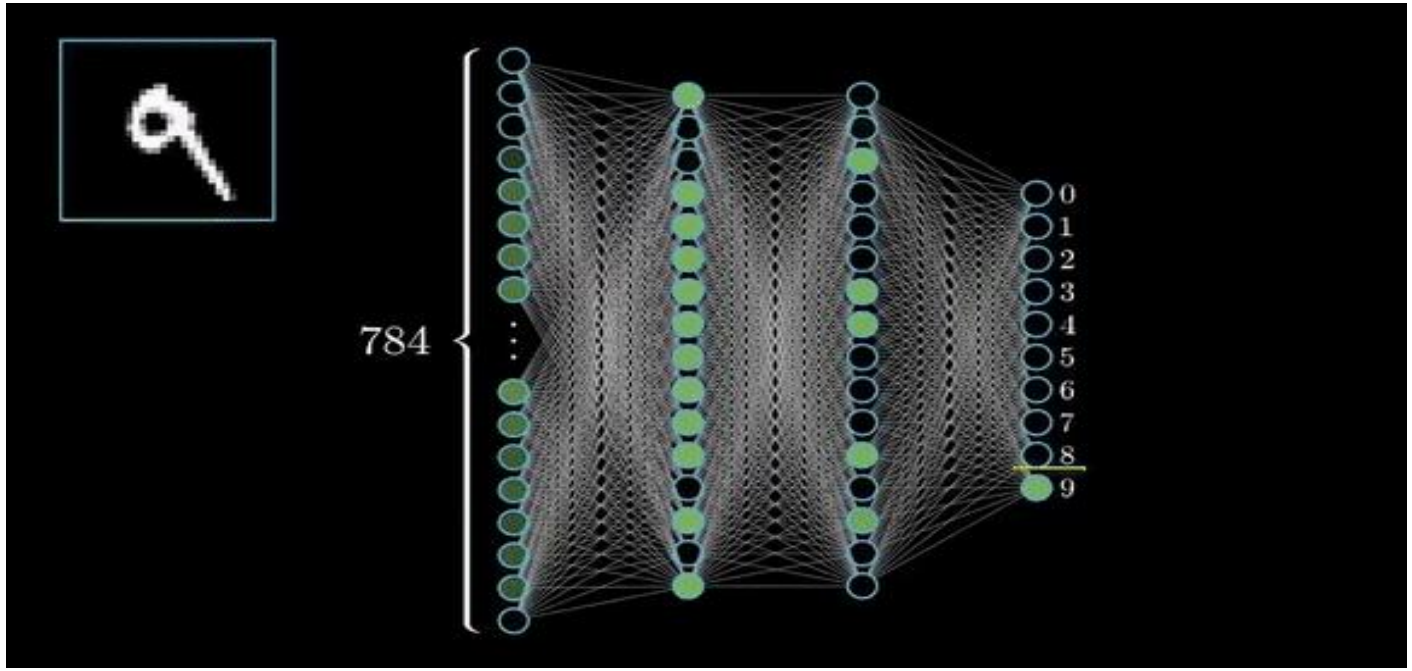
$$W^{[2]}a^{[1]} = \hat{y} \Rightarrow \begin{bmatrix} 1.0245 & 1.1335 \end{bmatrix} \begin{bmatrix} 0.9344 \\ 0.9481 \end{bmatrix} = 2.032$$

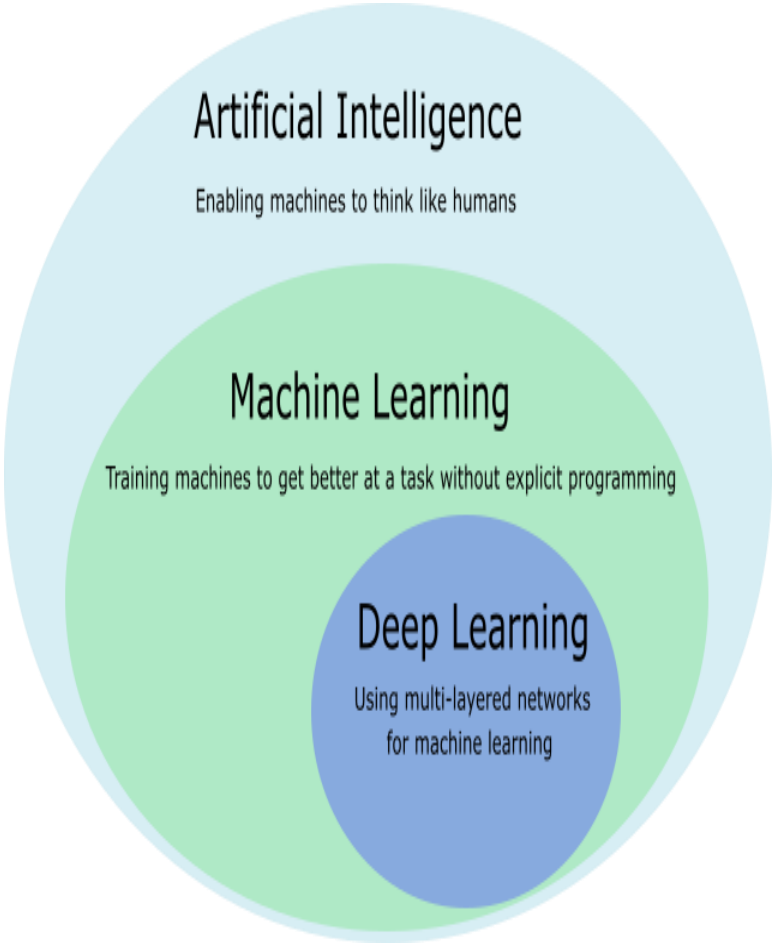
$$MSE = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(2.032 - 2)^2 = 0.000512$$

What is Deep Learning?

Deep learning has significantly improved the state-of-the-art for many problems that machine learning and AI community faced for a lot of years.

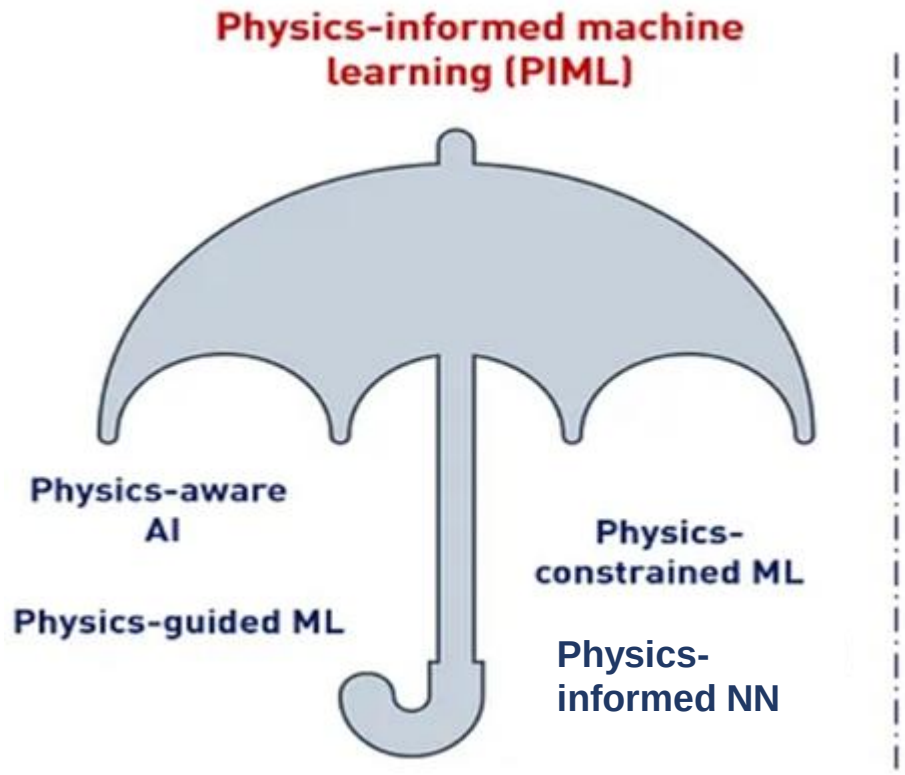
Deep Learning is representation learning using deep neural network (DNN) .

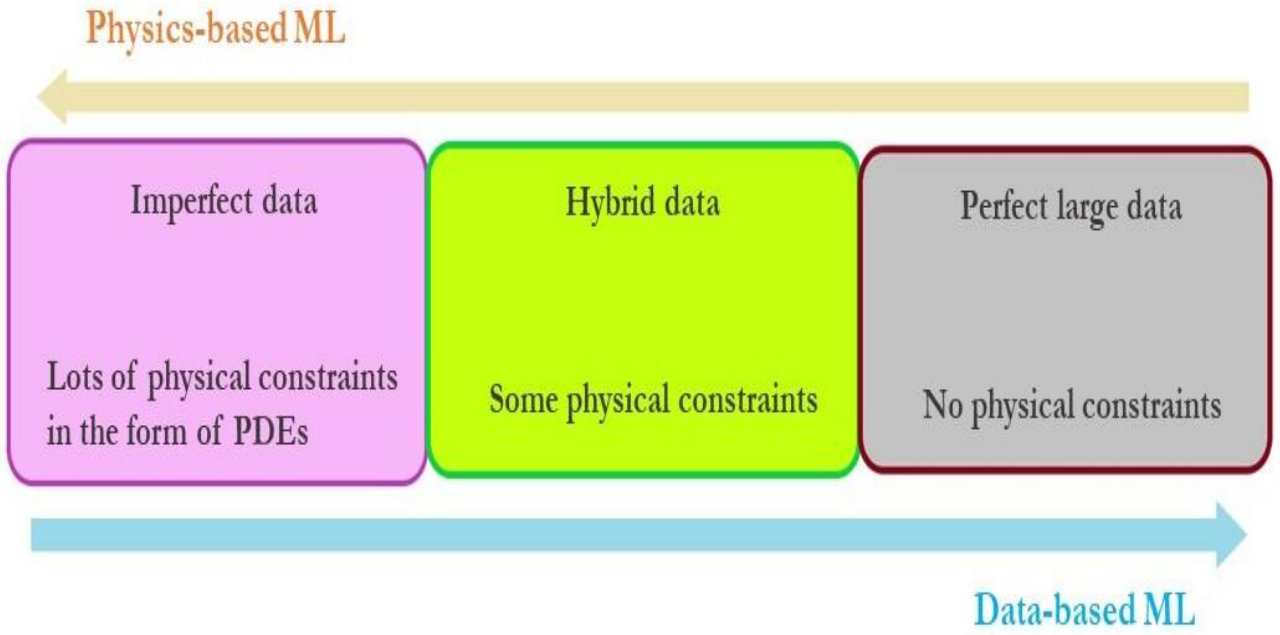




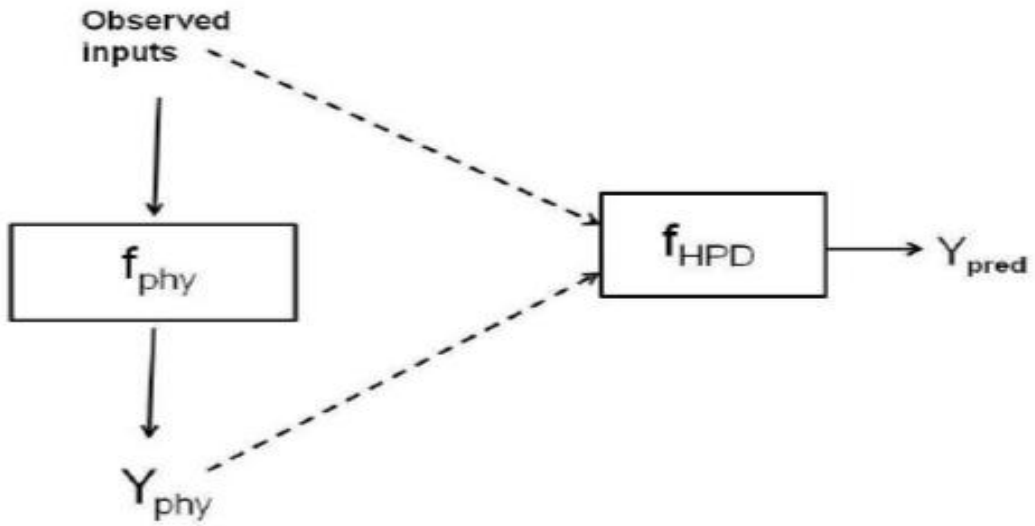
A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship.

Physics-informed Machine Learning: Utilization of Physical Knowledge





Formation of hybrid-physics-data model:



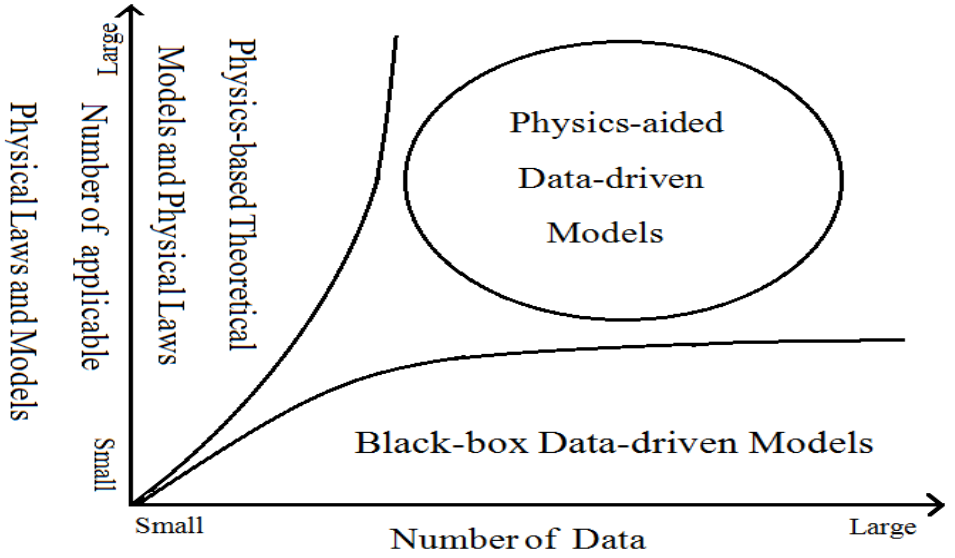
$$f_{HPD}: D = [I, Y_{phy}] \rightarrow Y_{true}$$

Then why do we need HPD model ?

In this approach, if Yphy = Ytrue then HPD model can accelerate to predict Ytrue.

However, if Yphy (not equal to Ytrue) has some discrepancies, then such models can extract complex features from the domain of input variables and minimizes our knowledge gaps.

Using physical constraints into the loss function:



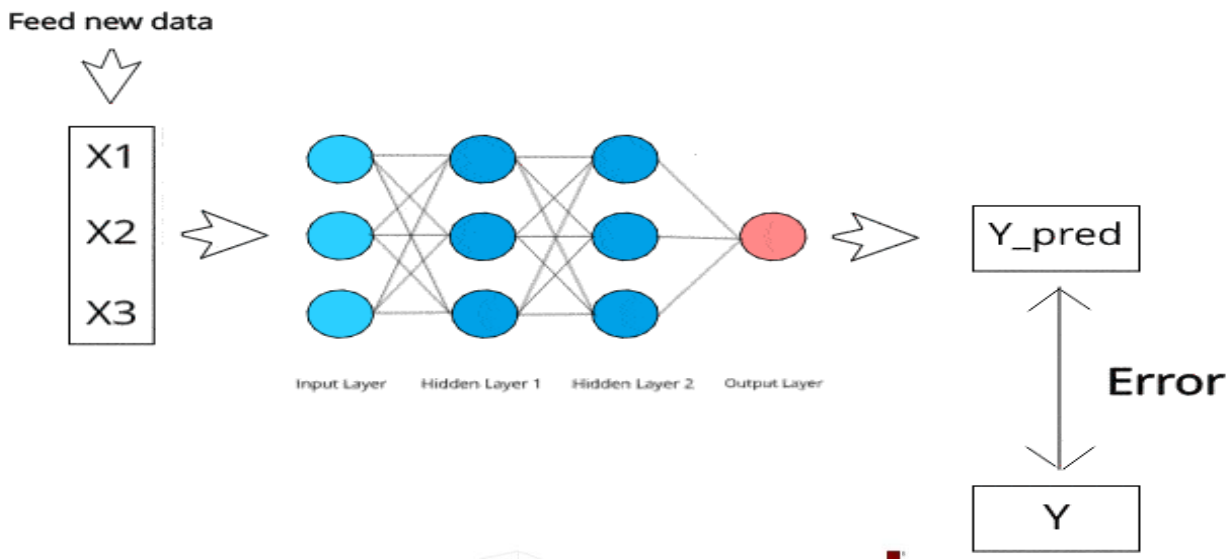
$$\text{Final Loss}_{\text{Train}} = \text{Loss}_{\text{Empirical}}\left(Y_{\text{exp}}, Y_{\text{pred}}\right) + \lambda\, \text{Loss}_{\text{Delta}}\left(Y_{\text{pred}}\right)$$

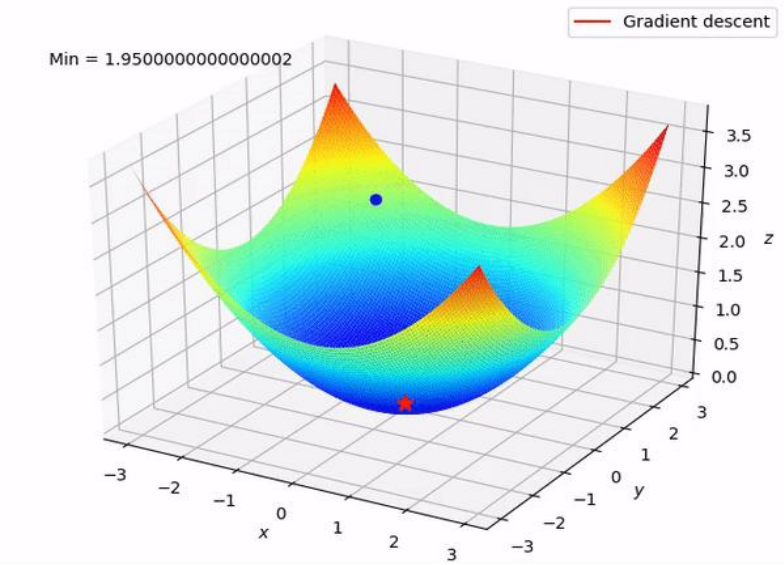
Then why do we need to modify loss function?

The processes of minimizing the final loss is considered as a condition to update the network weights and biases.

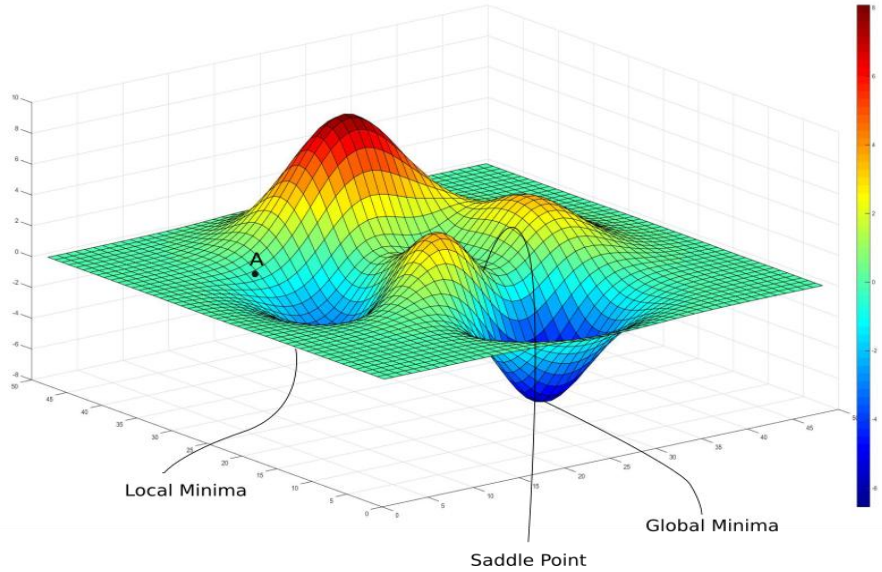
This mechanism carries on unless the final error converges to an optimal value.

Loss Functions in Deep Learning

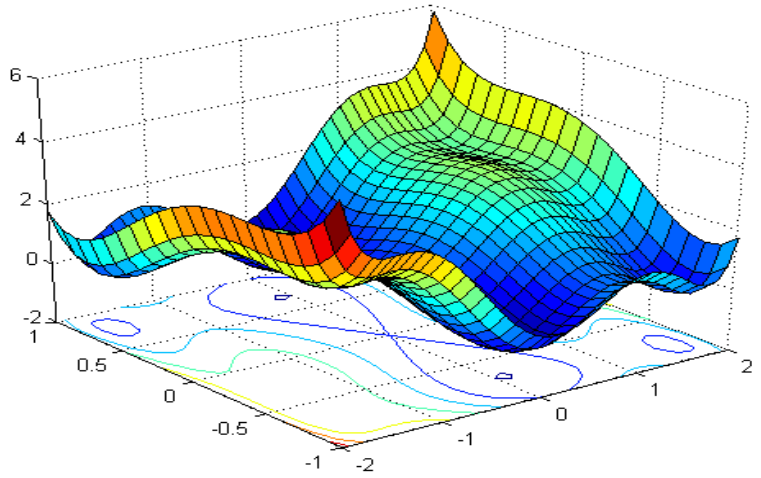




Convex Loss Function

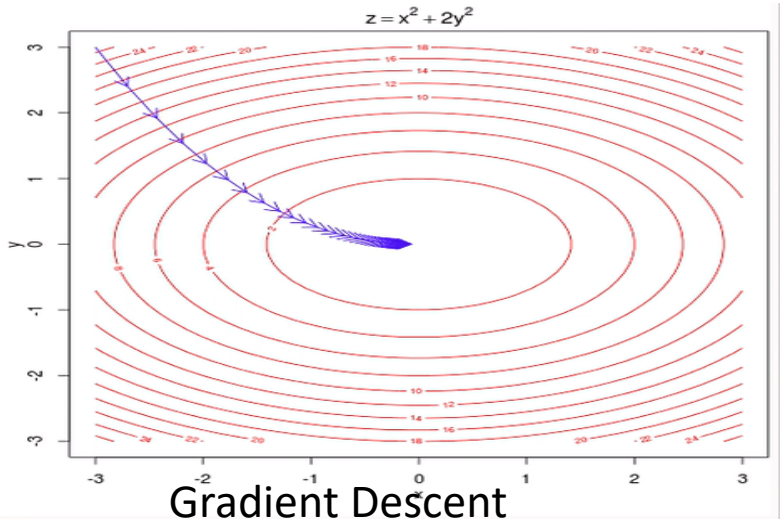


Non-Convex Loss Function

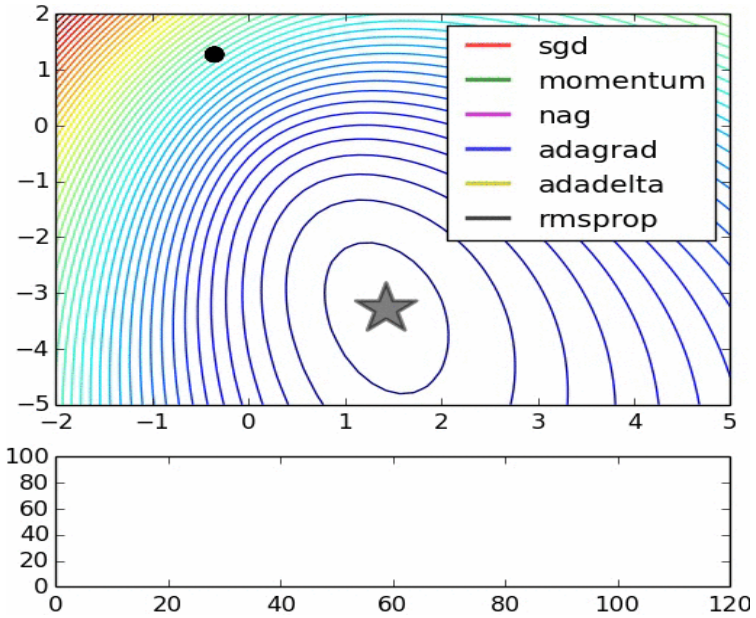


Non-Convex Loss Function

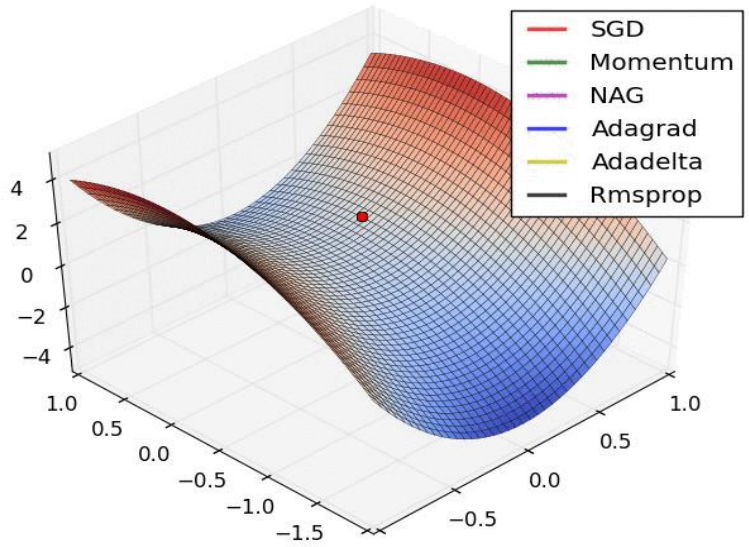
Visualization of optimization algorithms used in Deep Learning

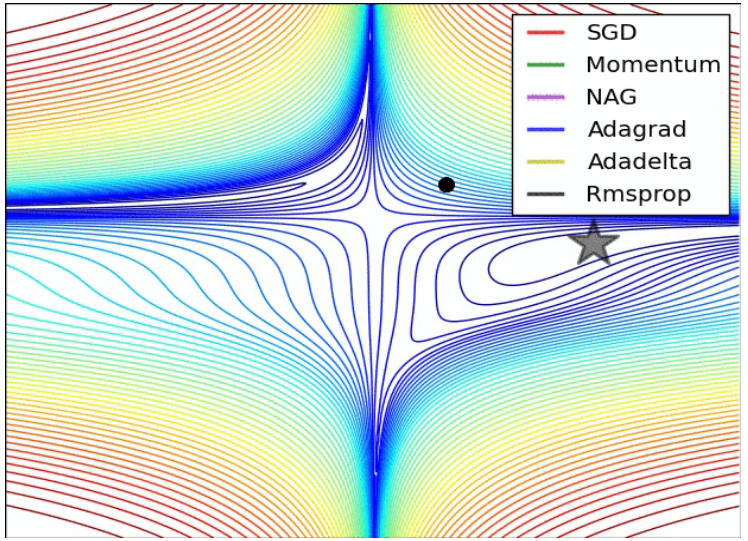


Gradient Descent



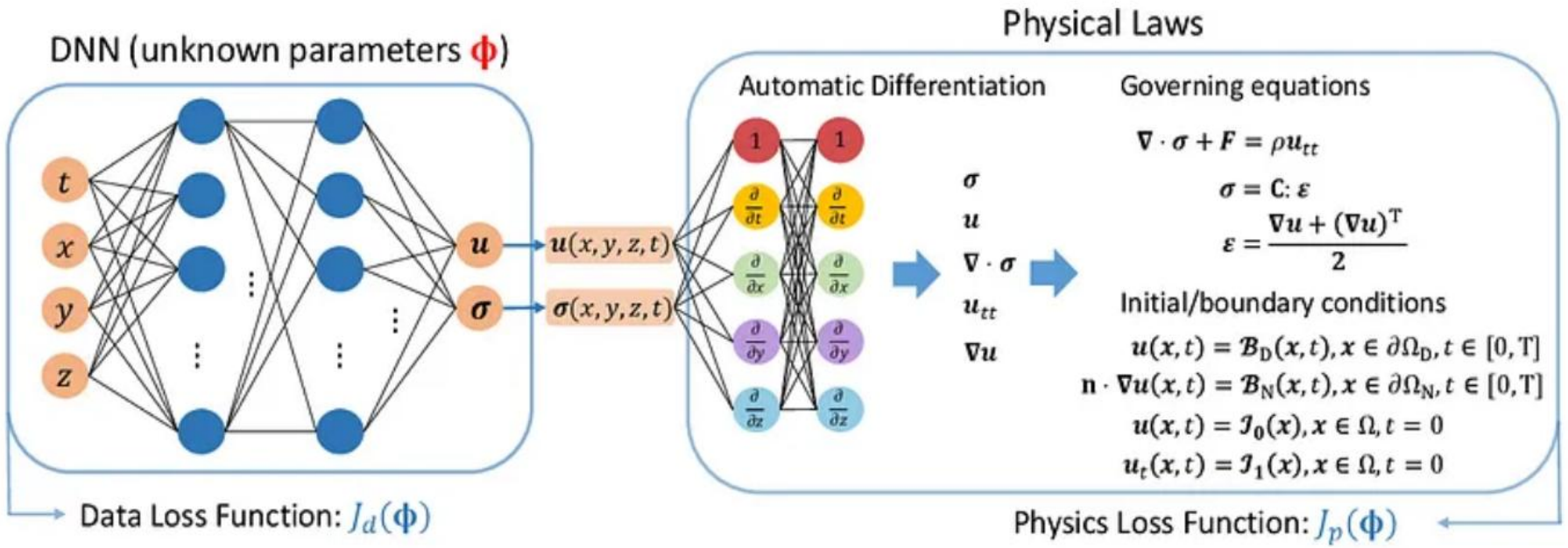
Comparison of different optimizers





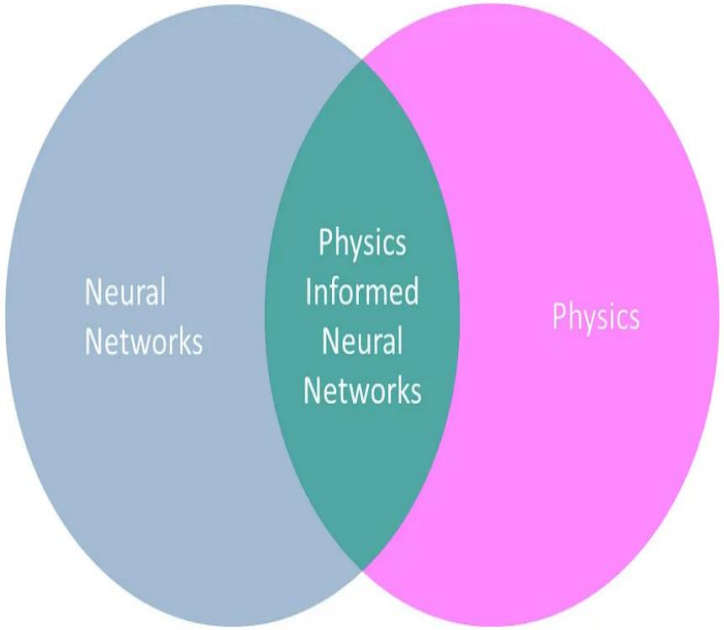
Physics-informed Neural Networks (PINNs): Solving PDEs

Methodology for solving non-linear PDEs (Forward Problems)



$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \{ J_d(\phi) + J_p(\phi) \}$$

PINNs:



Adapted from Lu et al. (2021)

Methods for Incorporating PDEs: Penalty-Based Methods

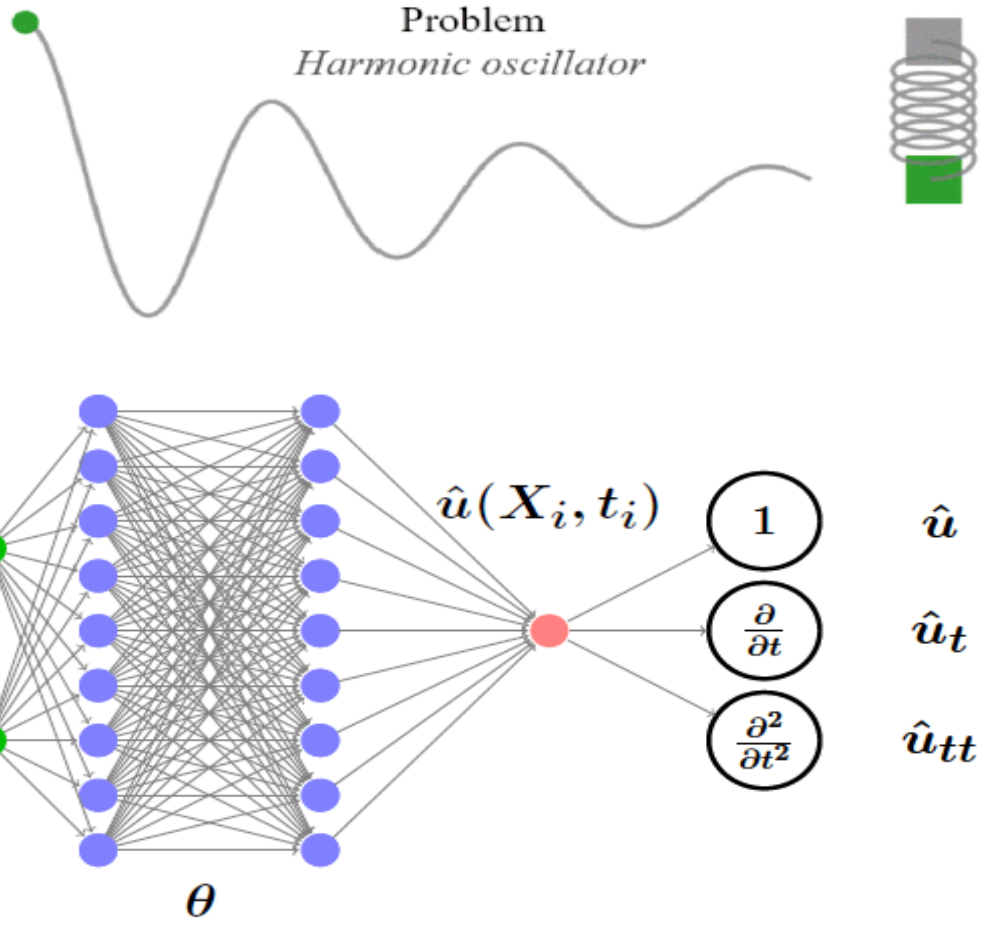
Method: Enforce physical laws as hard constraints either in:

- NN Architecture: This is an open problem.
- Optimization: Very difficult to train the NN with such constraints.

$$m\frac{\partial^2 u}{\partial t^2} + \mu\frac{\partial u}{\partial t} + ku = 0$$

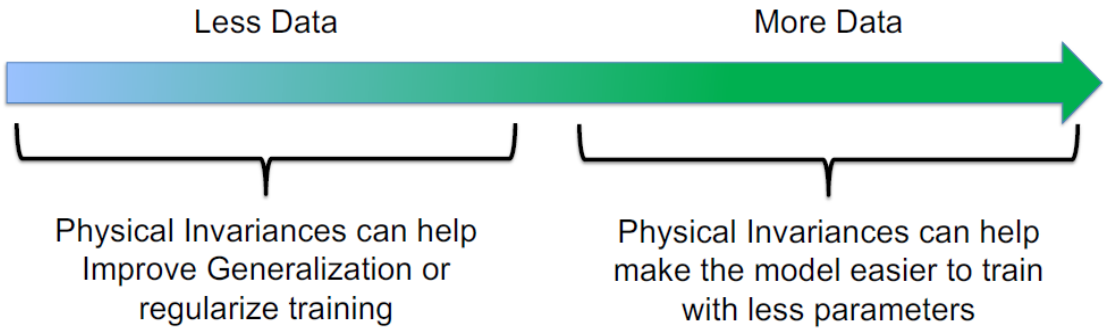
$$\min \frac{1}{N}\sum_{i}^{N}(\hat{u} - u_{true})^2 +$$

$$\frac{1}{M}\sum_{j}^{M}(m\frac{\partial^2 u_j}{\partial t^2} + \mu\frac{\partial u_j}{\partial t} + ku_j)^2$$

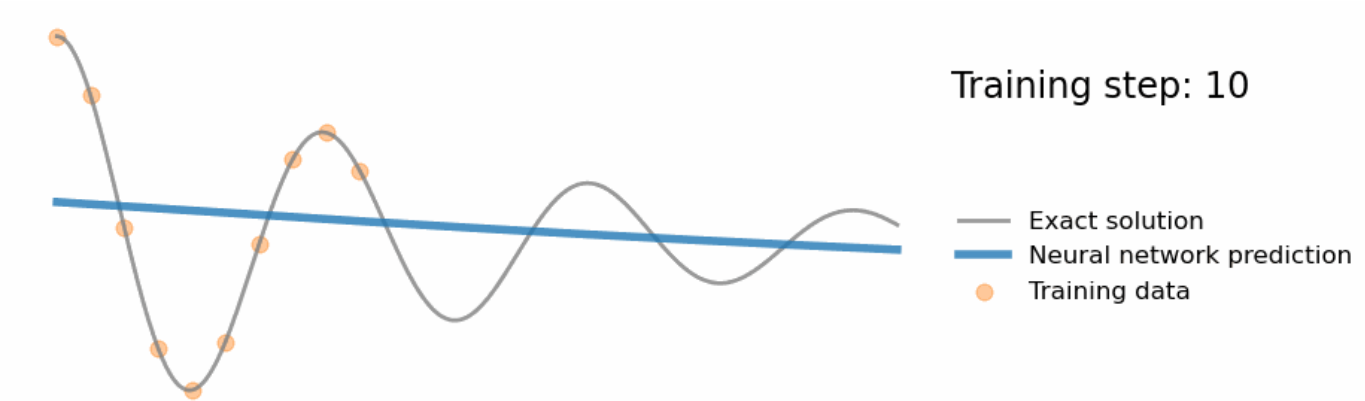


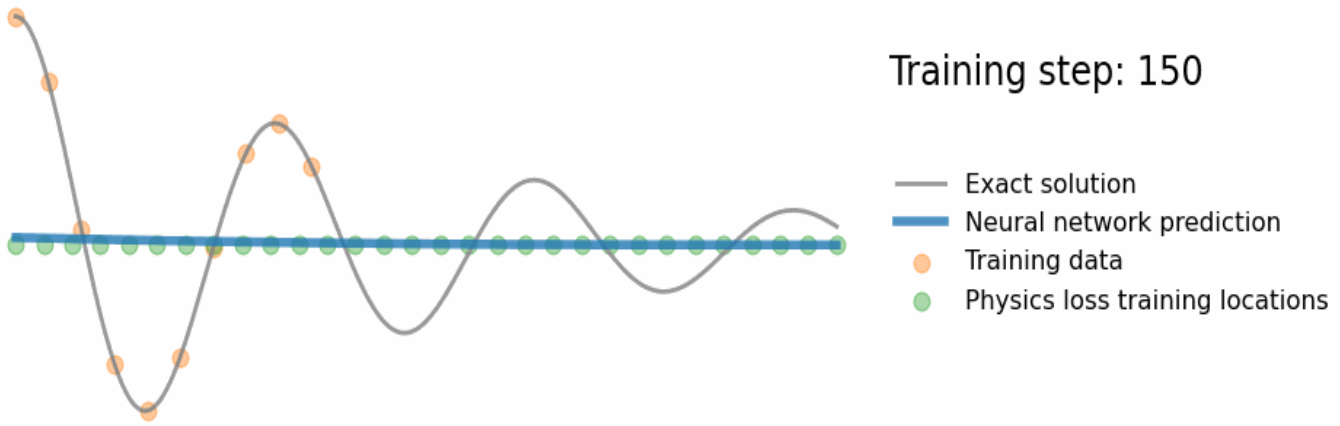
- Neural Networks **require a lot of data** to train.
- Collecting large scale data is not always possible.

Continue…



- Other than observed data, we know the invariances that govern physical phenomena.
- An important source of data are the **Physical Laws** that govern our world which have been largely ignored in exchange for observed data.
- Physical Laws include: Conservation of Mass, Momentum, Energy, etc. in the form of PDEs.





Incorporation of Initial and boundary conditions

Method: Use penalty methods and add the PDE residual to the loss.
– Very easy to implement, and works with any NN architecture
– Does not require a mesh or a numerical solver for the PDE
– Can (in theory) work for high dimensional problems, and complex PDEs

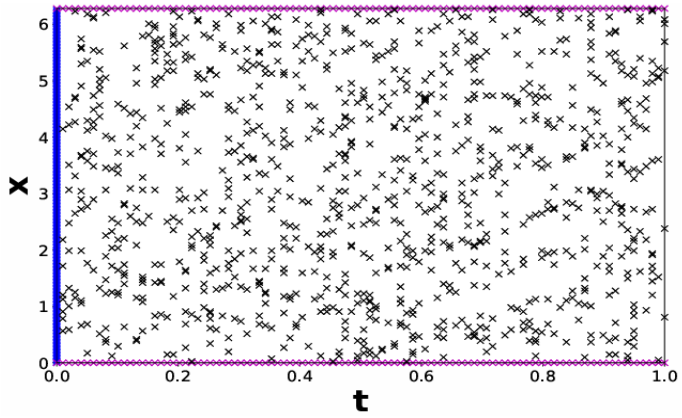
• For example, Consider a one-dimensional convection problem, a hyperbolic PDE which is commonly used to model transport phenomena:

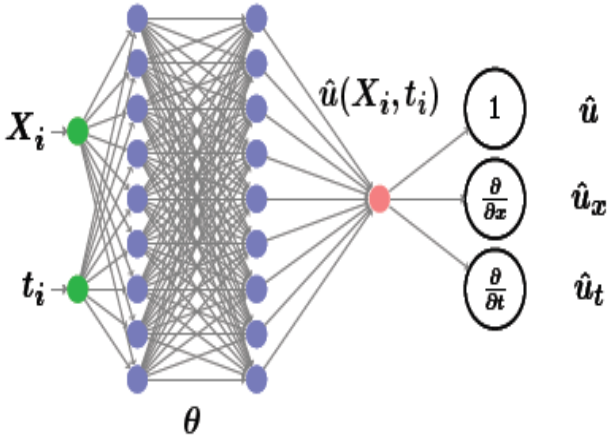
$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0, \quad x \in \Omega,\ t \in [0, T],$$
$$u(x, 0) = h(x), \quad x \in \Omega.$$

The simple initial and periodic boundary condition:

$$u(x, 0) = sin(x),$$
$$u(0, t) = u(2\pi, t).$$

PINN Dataset





$\hat{u}$ $\hat{u}_x$ $\hat{u}_t$

$$\min_{\theta} \mathcal{L} = \lambda_{\mathcal{F}} \|\hat{u}_t + \beta \hat{u}_x\|_2^2 \quad \text{PDE Residual}$$
$$+ \|\hat{u}(x, 0) - sin(x)\|_2^2 \quad \text{Initial Condition}$$
$$+ \|\hat{u}(x = 2\pi) - \hat{u}(x = 0)\|_2^2 \quad \text{Boundary Condition}$$

× Initial condition points × Boundary points × Collocation points

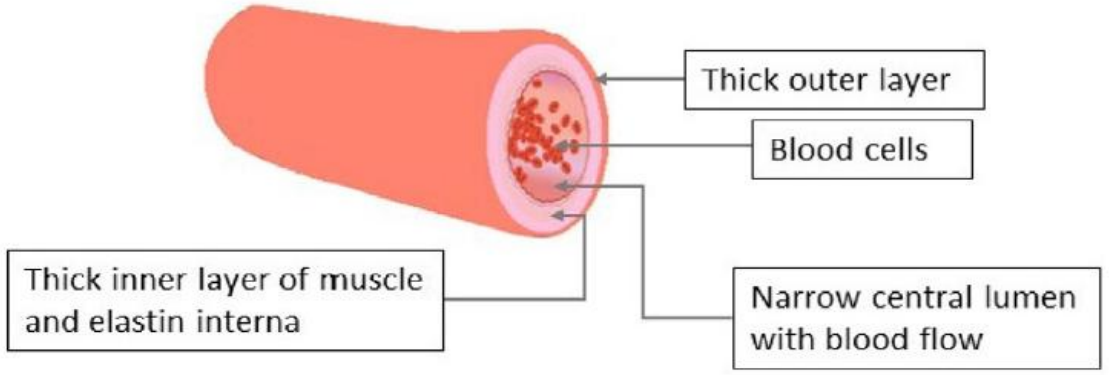
My collaborative Research Work with University of Copenhagen

The physics informed neural networks (PINNs) model for

Solving Nonlinear evolutionary equations [Published in the Journal "Mathematics Computers in Simulation", Elsevier, 2024]

- **Effects in the process of arterial blood flow:**
  - The properties of the fluid medium
  - The elastic properties of the tube wall.

- **Utilization of Deep Learning:**
  - Introduce physics-informed deep neural networks
  - Problem Setup Using PINN Approach
  - Dataset construction
  - Utilization of Bayesian Hyper-parameter Optimization
  - Validation of deep learning based solutions



Viscoelastic arterial tube

Blood flow Model

Basic Equations of Blood Flow Through an Arterial Tube:

One-dimensional Equation of Fluid mass:

$$\frac{\partial A^*}{\partial t} + \frac{\partial (A^* \hat{u})}{\partial x} = 0.$$

One-dimensional Momentum Equation:

$$\frac{\partial \hat{u}}{\partial t} + \hat{u}\frac{\partial \hat{u}}{\partial x} + \frac{1}{\rho_f}\frac{\partial P^*}{\partial x} = \nu^0 \frac{\partial^2 \hat{u}}{\partial x^2} - 2\hat{u}(\gamma + 2)\frac{\hat{u}}{R^{*2}}.$$

Where, v_x^* is the radial profile of the axial flow velocity component, $\hat{u}(x,t)$ is the mean velocity over the cross section, $R^* = R^*(x,t)$ is the radius of the arterial tube and γ is the velocity profile sharpness.

Here, $A^*(x,t) = \pi(R^*(x,t))^2$ is the cross-sectional area of the arterial tube.

Blood flow Model (Contd.)

Equation of Motion of The Viscoelastic Tube:

$$P^* - P_0 = \rho_v h_0 \eta^*_{tt} - k h_0 \eta^*_{xx} - \chi h_0 \eta^*_{txx} + \mu \eta^*_t + \frac{\kappa h_0}{R_0}\eta^* + \frac{\kappa_2 h_0}{R_0^2}\eta^{*2}.$$

Where,

$$\kappa_2 \equiv \kappa_1 R_0 - 2\kappa,$$

$$R^*(x,t) = R_0 + \eta^*(x,t), \quad R_0 = \text{equilibrium radius}, \quad \|\eta^*\| << R_0$$

P_0 is the pressure on the outer surface of the tube be constant, ρ_v is the volume density of the tube, h_0 the thickness of the wall in stable position, μ represents the coefficient proportionality between the resistance of the medium and the motion of the viscoelastic wall, χ is the viscosity coefficient of the tube material, and κ and κ_1 are respectively the linear and nonlinear elasticity coefficient of the tube.

Geometry of problem

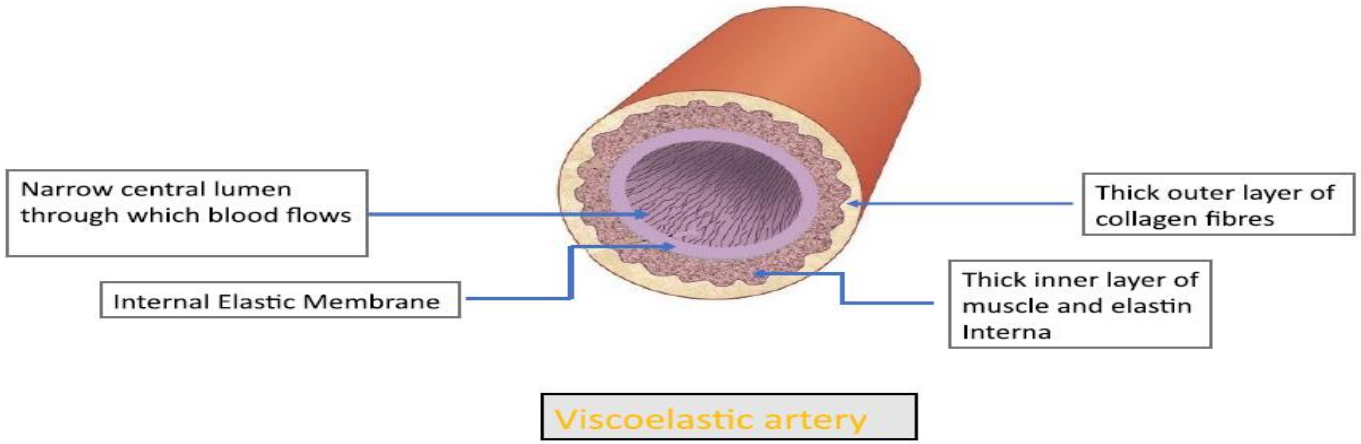


Figure 1: A visualization of viscoelastic arterial tube

Dimensionless Flow Equations:

$$\eta_t + u_x + \frac{1}{2}\eta u_x + u\eta_x = 0$$

$$u_t + uu_x + \frac{1}{\alpha}P_x = 0$$

$$P = \gamma\eta_{tt} - \beta\eta_{xx} + \lambda\eta_t - \delta\eta_{txx} + \alpha\eta + \alpha_1\eta^2 + 1.$$

$$\text{Where, } \alpha = \frac{\rho c_0^2}{P_0}, \quad \beta = \frac{kh_0R_0}{2P_0l^2}, \quad \gamma = \frac{\rho_w h_0 R_0 c_0^2}{2P_0l^2}$$

$$\delta = \frac{\chi h_0 R_0 c_0}{2P_0 l^3}, \quad \lambda = \frac{\mu R_0 c_0}{2P_0 l}, \quad \alpha_1 = \frac{\kappa_1 h_0 R_0}{4P_0} - \alpha$$

Simplifications of Model

In order to construct approximate solutions for the system of equations, we used the reductive perturbation method. In the next step, we could find the solution to the system of equations by utilizing the following variables:

$$\eta^* = \varepsilon^p \eta', \quad u^* = \varepsilon^p u', \quad P^* = 1 + \varepsilon^p P', \quad p \in N$$
$$\xi = \varepsilon^m (x - t), \quad \tau = \varepsilon^n t, \quad m, n \in N, \quad n > m$$
$$\frac{\partial}{\partial x} = \varepsilon^m \frac{\partial}{\partial \xi}, \quad \frac{\partial}{\partial t} = \varepsilon^n \frac{\partial}{\partial \tau} - \varepsilon^m \frac{\partial}{\partial \xi}$$

By assuming $m = 1$ and using the relation $n - m = p = q$, we are going to consider three different cases regarding p, q, and n as follows: (1) $p = q = 1$ and $n = 2$; (2) $p = q = 2$ and $n = 3$; and (3) $p = q = 3$ and $n = 4$.

Continue….

Simplifications of Model

$$u_t + uu_x = \frac{1}{2}u_{xx}, \tag{23}$$

$$u_t + uu_x + \frac{\gamma - \beta}{2}u_{xxx} = 0, \tag{24}$$

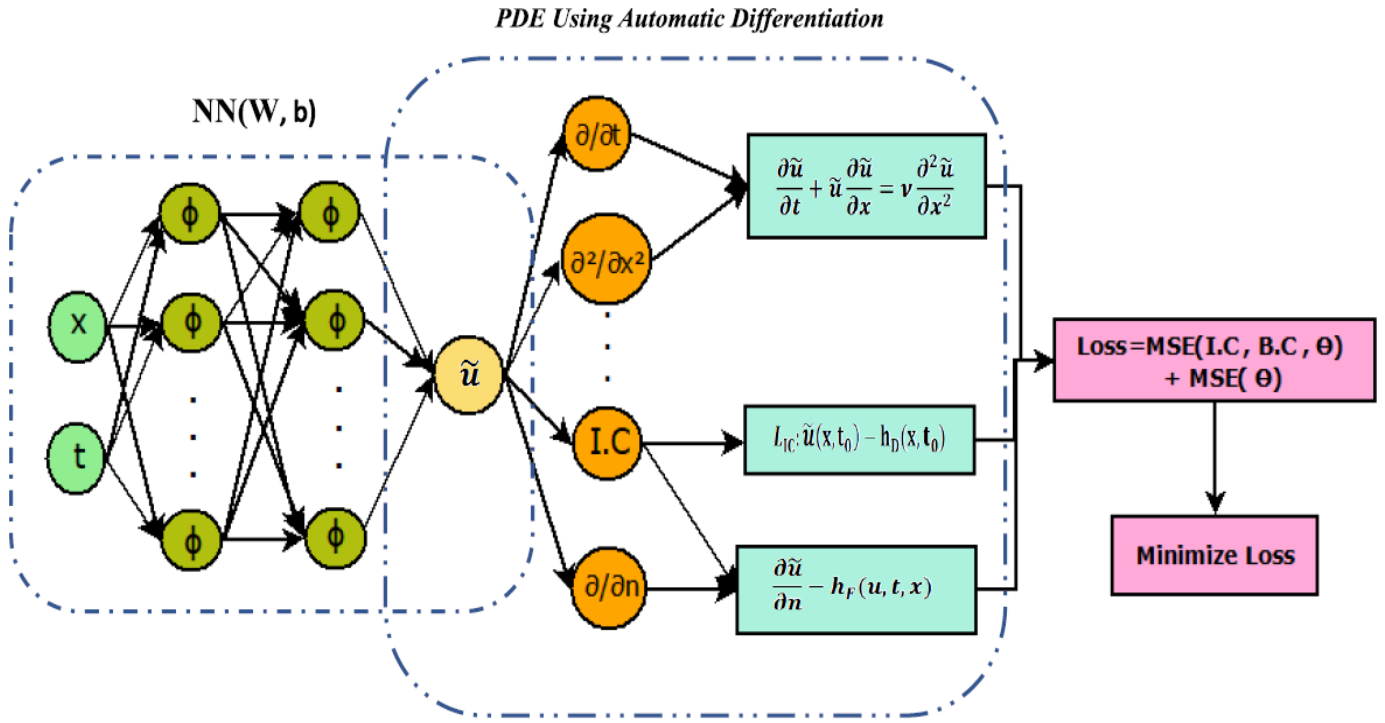
$$u_t + uu_x + \frac{1}{2}u_{xxxx} = 0. \tag{25}$$

As noted above, during the initial phase, the perturbations are damped in accordance with the Burgers equation (Eq. (23)). The leading factor affecting the wave dissipation at this stage is the resistance of the medium to the movement of the wall. In the next phase, a KdV equation (Eq. (24)) can be used to explain the propagation of waves. The main determining factor in that case is the elastic nature of the wall. In the third stage, the wave propagation can be described by the fourth-order nonlinear evolutionary equation (Eq. (25)). In this case, the viscous properties of the wall are the primary factor.

Problem Setup Using PINN Approach

$$u_t + F\left(x, t; u_x, u_{xx}, u_{xxx}, \ldots\right) = 0, \quad \mathrm{x} \in D, \quad \mathrm{t} \in (0, T), \qquad \mathcal{B}(u; x, t) = 0 \quad \text{on } \partial D.$$

In which, u is the solution of the differential equation, F is the nonlinear function and B(u; x, t) represents the boundary condition.



$$u(x,t) \approx \tilde{u}(x,t) = \mathbf{NN}\left(x, t; \mathcal{T}_B, \mathcal{T}_C; \theta\right).$$

Where,

- $\mathcal{T}_B = \left\{\left(x_i, t_i, \mathcal{B}(u(x_i, t_i))\right)_{i=1}^{N_b}\right\}$
- $\mathcal{T}_C = \left\{\left(x_i, t_i, g\left(\tilde{u}(x_i, t_i)\right)\right)_{i=1}^{N_c}\right\}$
- $\theta = \left\{W_i^T, b^i\right\}_{i=1}^{L-1}$

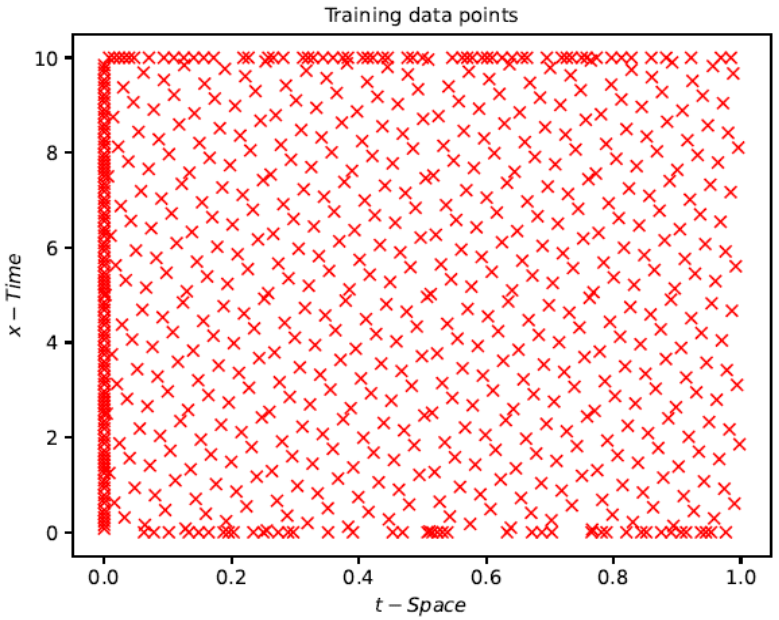
$$\mathrm{Loss}_{\mathrm{Trn}}(\theta; \mathcal{T}_B, \mathcal{T}_C) = \lambda_1 \mathrm{L}_1\left(\theta; \mathcal{T}_B\right) + \lambda_2 \mathrm{L}_2\left(\theta; \mathcal{T}_C\right)$$

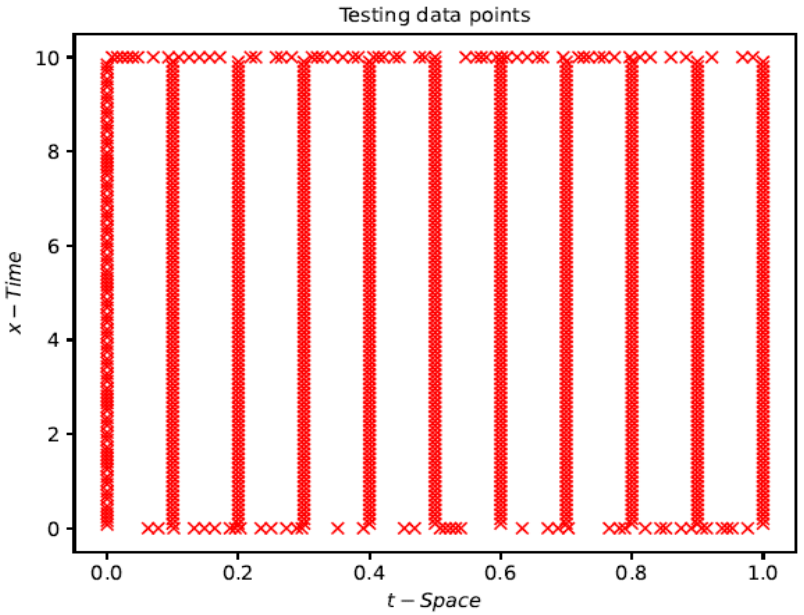
where

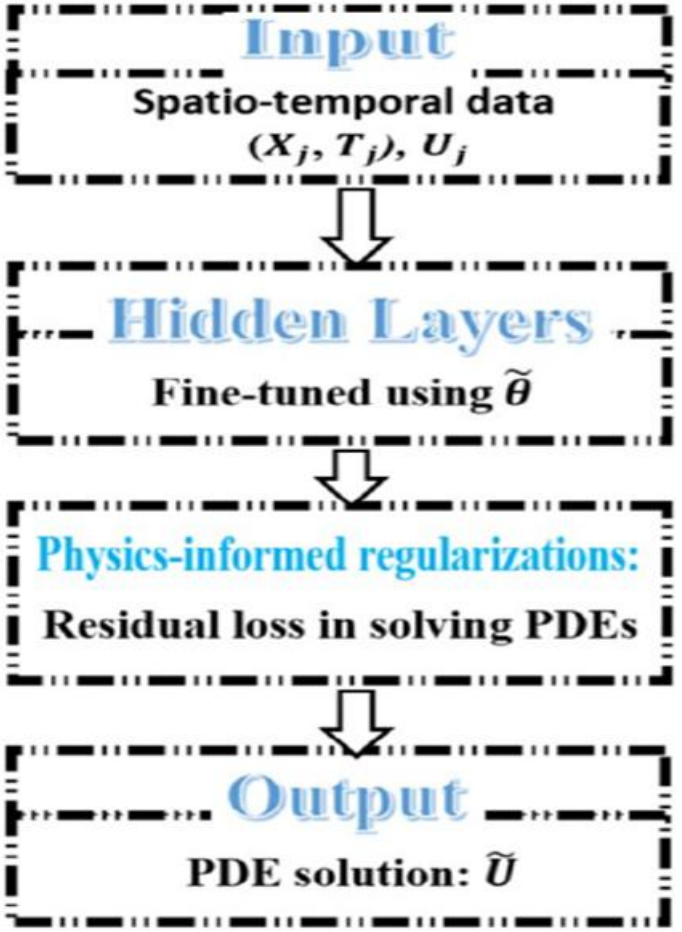
$$\mathrm{L}_1\left(\theta; \mathcal{T}_B\right) = \frac{1}{|\mathcal{T}_f|} \sum_{(x,t)\in\mathcal{T}_B} \left\|u_t + F\left(x, t; u_x, u_{xx}, u_{xxx}, \ldots\right)\right\|_2^2,$$

$$\mathrm{L}_2\left(\boldsymbol{\theta}; \mathcal{T}_C\right) = \frac{1}{|\mathcal{T}_C|} \sum_{(x,t)\in\mathcal{T}_C} \left\|\mathcal{B}(\tilde{u}; x, t)\right\|_2^2,$$

Dataset Construction For Solving PDEs







- Approximately 700 residual points, including 500 randomly selected data points within the domain, 100 on the domain, and 100 as initial residuals.
- We chose 1000 testing data points from the interior and on the boundary of the domain. Among those points, 800 data points were uniformly selected inside the domain.
- The 200 boundary points were the same as those taken in the training set to validate the accuracy and generalizability of the model on unseen data.

Bayesian Hypermeter Optimization

In Bayesian optimization, the aim is to determine the global maximum or minimum of an unknown objective function,

$$x^* = \arg\min_{x^*} f(x), \quad x \in X,$$

where, X represents the space of hyperparameters, and x^* represents the hyperparameters to be optimized.

Hyper-parameters used in Bayesian optimization algorithm.

Hyperparameters	Values		Optimal values	Optimal values
	Default values	Range of parameters	Burger	4th order
Learning rate	$1e^{-3}$	$1e^{-6}$–$1e^{-1}$	$1.416e^{-4}$	$1.416e^{-4}$
Dense layers	4	1–10	4	4
Nodes	50	5–500	16	16
Initializer	Glorot normal	Glorot normal	Glorot normal	Glorot normal
Activation functions	sin	Sigmoid sin, tanh	sin	sin
Optimizer	Adam	Adam	Adam	Adam

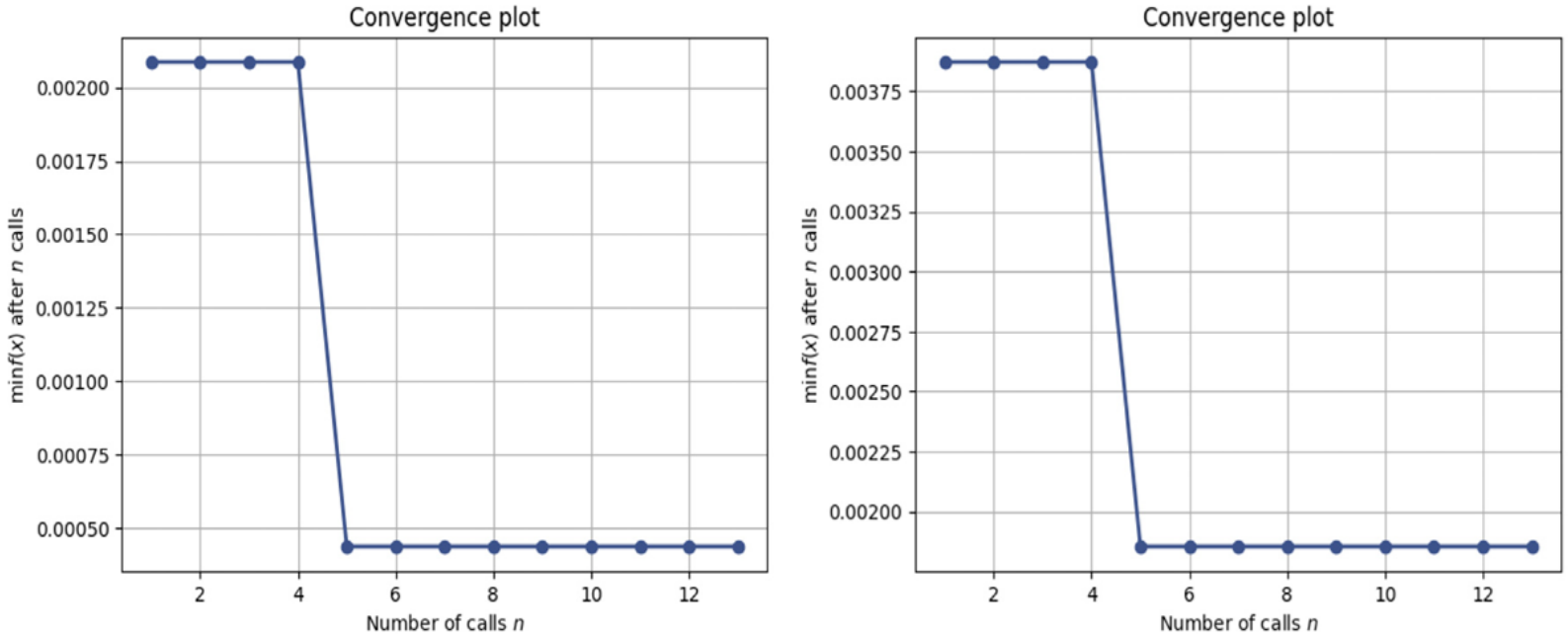
- To attain a high level of accuracy of using the PINN model the process of tuning network parameters and selecting hyper-parameters, such as network structure (i.e., number of layers and number of nodes in each layer), learning rate, and activation function, needs to be more compact.

Steps in Bayesian Optimization:

- Create a surrogate model to predict objective function
- Use Acquisition function i.e., Expected improvement method or upper confidence bound method
- Evaluate the model score with previous known configuration
- Repeat the above procedure

Continue….

- Simplification of the task can be achieved through Bayesian hyperparameter optimization algorithm, which develop a probability distribution over the unknown function to reduce the number of evaluations required.



Convergence for the neural network solutions of (a) Burger and (b) fourth-order evolutionary equations with the number of calls by using Bayesian optimization.

- Here, the Y -axis denotes the minimum value of the black box objective function $f(x)$ after n calls/trials and the X-axis shows the number of calls to the objective function. These figures reveal that the convergence of the Burger and fourth-order evolutionary PDEs demonstrates a decreasing value of $f(x)$, and the minimum value of $f(x)$ is attained after 5 calls.

- The training process in this approach included early stopping criteria implemented through the use of the callback concept. An interesting feature of this technique is that it compels the training process to terminate at the minimum testing error value.

Analysis of prediction performance

- Figs. depict the loss history for the solution of the Burger and fourth-order evolutionary equations for the same initial condition with respect to the number of epochs.

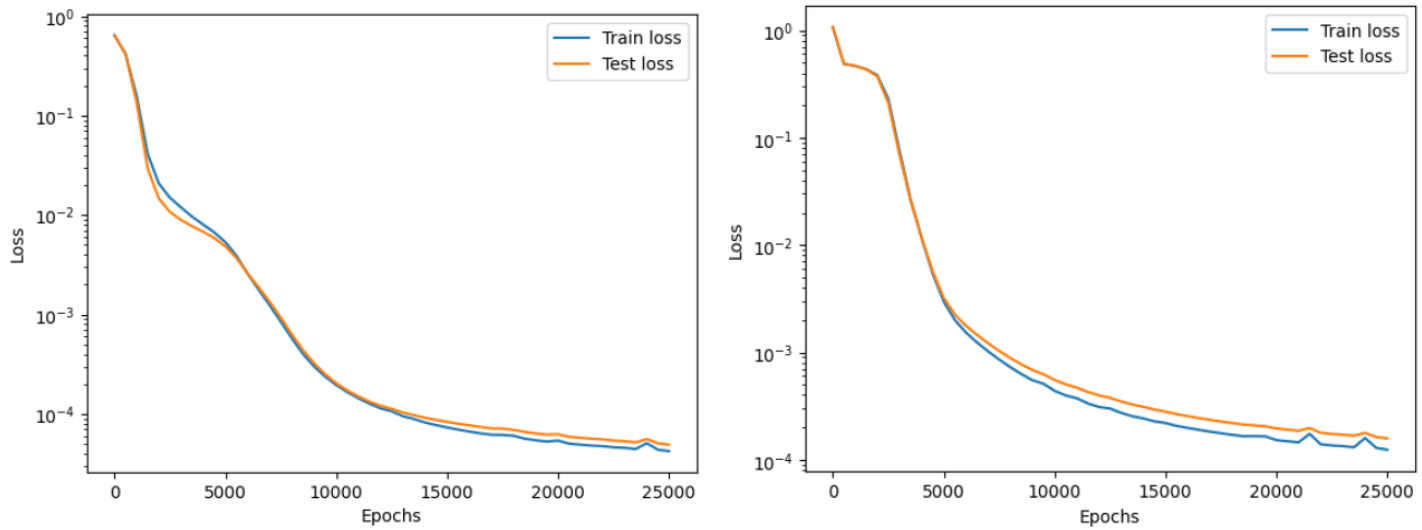


Figure : Losses (MSE) for the solution of (a) Burger and (b) fourth-order evolutionary equations with initial condition $u(x,0) = \sin\left(\frac{3\pi}{5}x\right)$.

- Figs. represent the PDE residual losses for both the Burger and fourth-order evolutionary equations.

$$\frac{1}{N_c}\sum_i \left\|\tilde{u}_t + F\left(x_i, t_i; \tilde{u}_x, \tilde{u}_{xx}, \tilde{u}_{xxx}, \ldots\right)\right\|_2^2$$

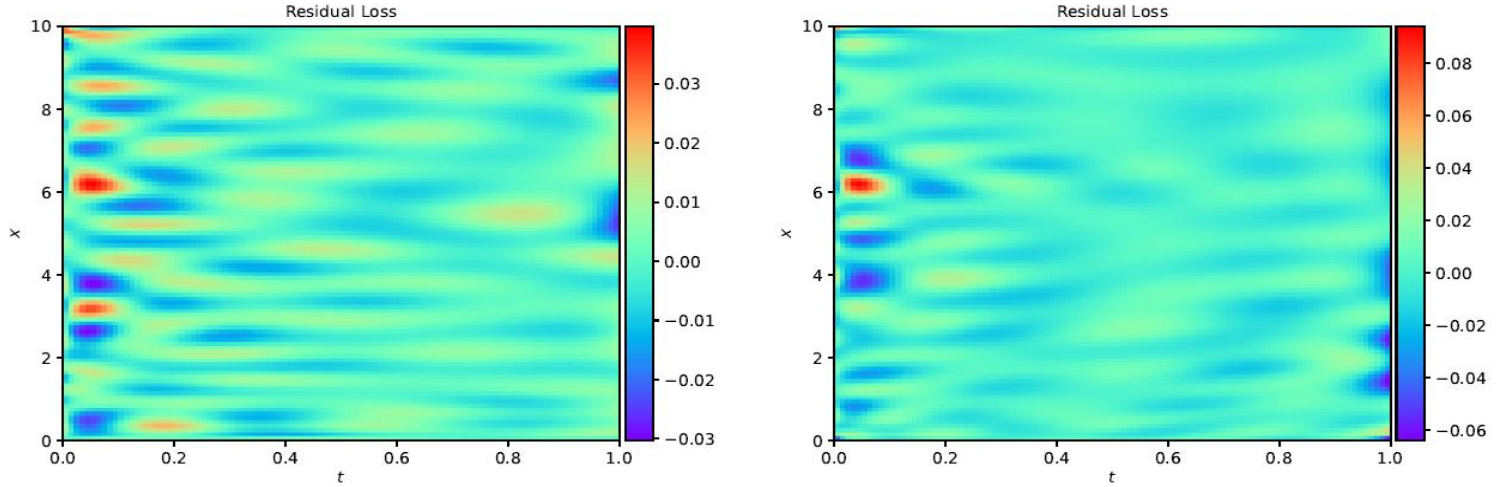


Figure : Residual loss of the solutions of (a) Burger and (b) fourth-order evolutionary equations on a rectangular region with periodic initial conditions $u(x,0) = \sin\left(\frac{3\pi}{5}x\right)$

Solutions of the PDEs in the form of Periodic Pressure Pulses and Solitary waves:

- These figures demonstrate that the process is dissipative as indicated by the higher order derivatives. The dissipation of these pulse waves occurs in a different way for different nonlinear equations.

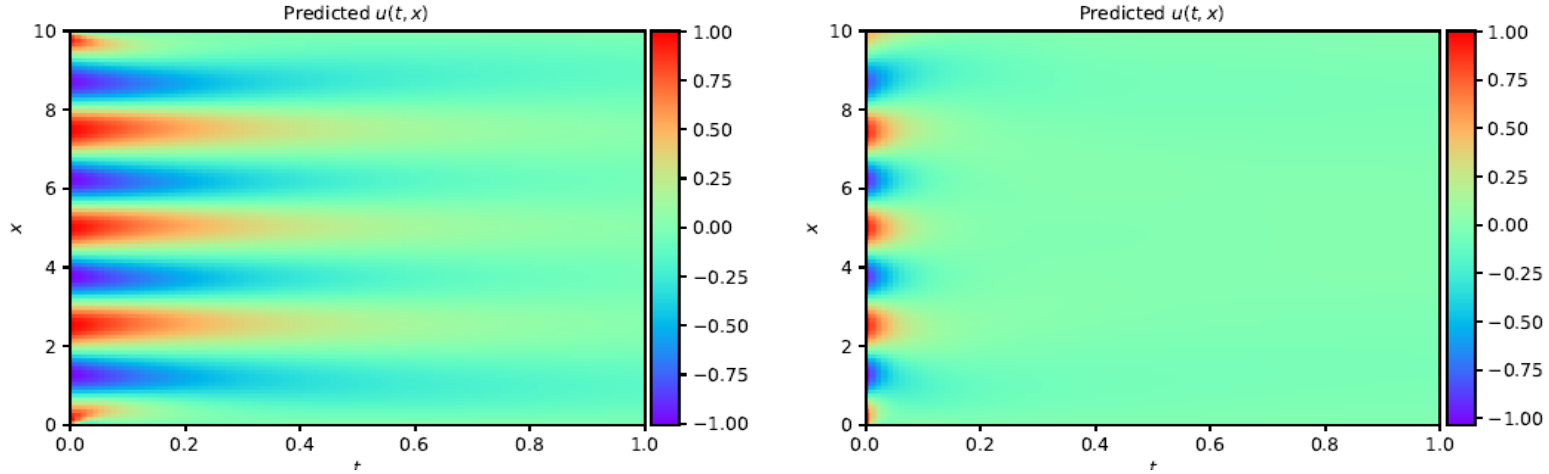


Figure : Solutions of (a) Burger and (b) fourth-order evolutionary PDEs using PINN approach with periodic initial condition $u(x,0) = \cos\left(\frac{4\pi}{5}x\right)$.

- Burger's equation decreases in amplitude over time and its shape also changes simultaneously. The fourth-order evolutionary equation, pressure pulses are more rapidly smoothed and damped than in Burger's equation.

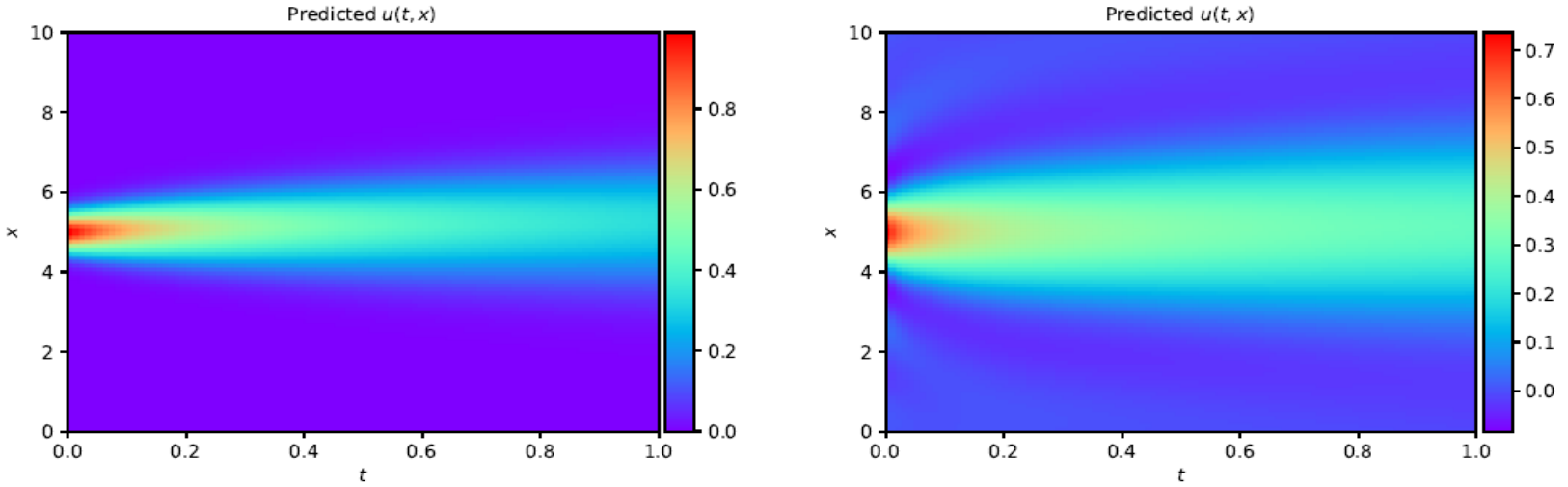


Figure : Solutions of (a) Burger and (b) fourth-order evolutionary PDEs using PINN approach on a rectangular region with solitary waves $u(x,0) = \exp(-4(x-5)^2)$ as initial condition.

Validation of deep learning based solutions with numerical solutions:

- According to these Figs., blue dotted lines represent numerical solutions, while red dotted lines represent PINN-based predicted solutions.

- As shown in these figures, the graphical comparison at different time instants t = 0, 0.25, 0.50, 0.75 are found to be good in agreement when we set the initial profiles same as those in Kudryashov and Chernyavskii (2005).

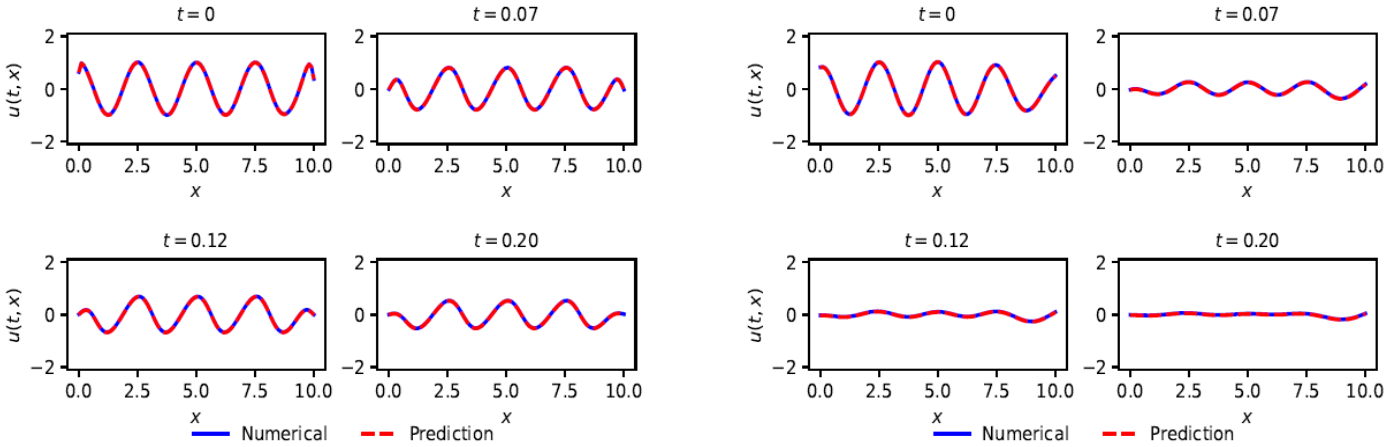


Figure : Solution comparison between numerical and PINN Algorithm for (a) Burger and (b) fourth-order evolutionary PDEs at instants $t = 0, t = 0.07, t = 0.12, t = 0.20$.

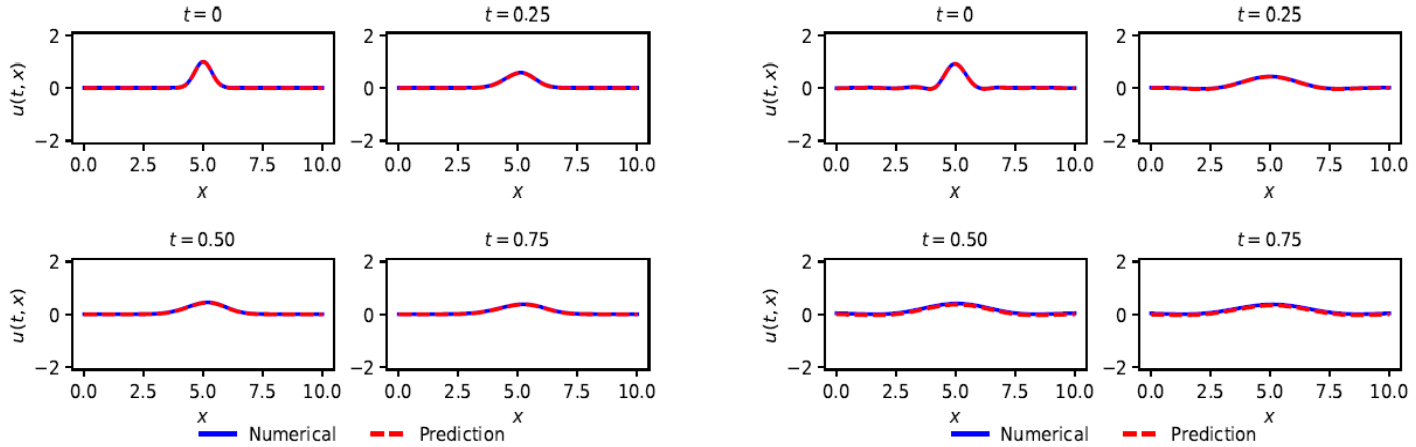


Figure : Solution comparison between numerical and PINN approach for (a) Burger and (b) fourth-order evolutionary PDEs at instants $t = 0, t = 0.25, t = 0.50, t = 0.75$.

Validation of deep learning based solutions with numerical solutions (Continue…):

- Additionally, we compared the PINN based solutions of burger equation with the numerical solution using finite difference scheme in these figures graphically by taking the periodic pressure pulse and solitary waves as an initial conditions.

- These figures confirm that the predicted solutions are closely aligned with the numerical solutions.

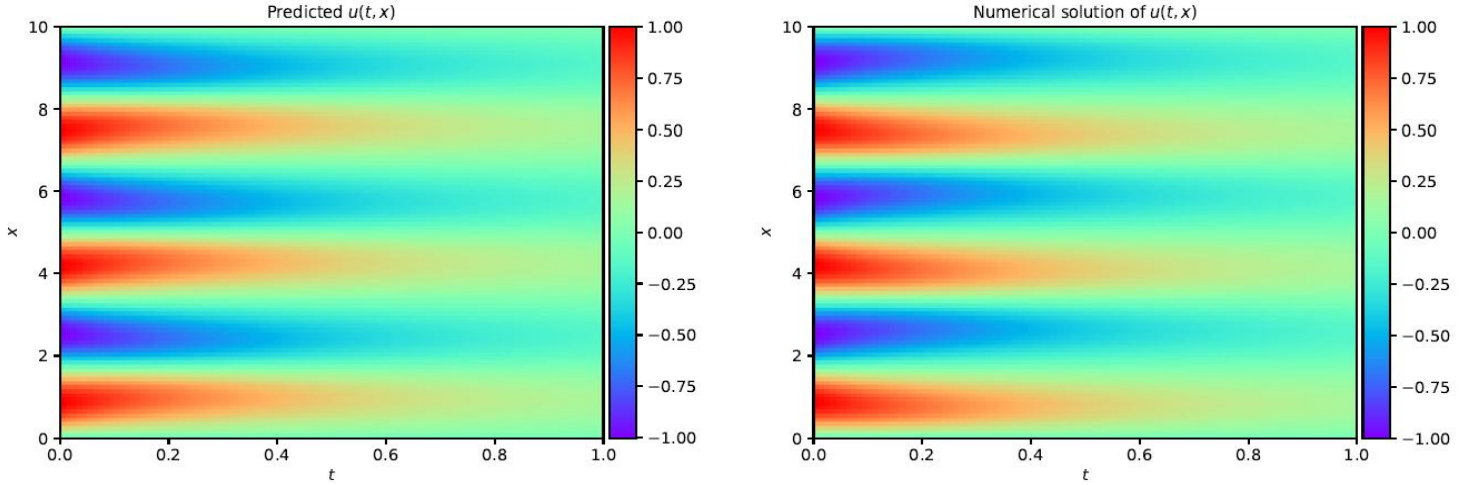


Figure : Solutions of Burger equation on a rectangular region by using (b) PINN and (a) Numerical (finite difference) methods with periodic initial condition $u(x,0) = \sin\left(\frac{3\pi}{5}x\right)$

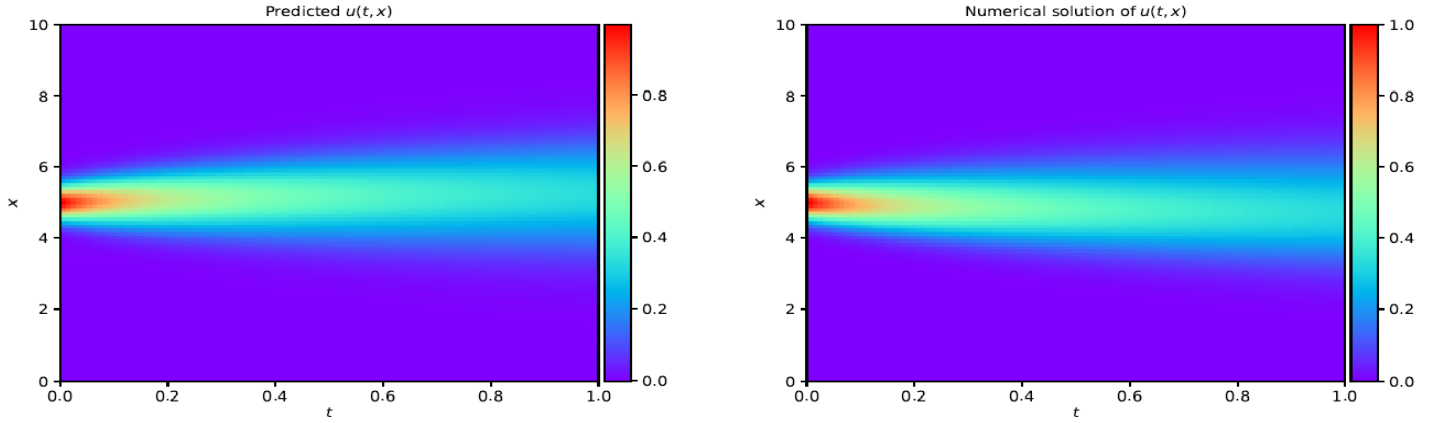
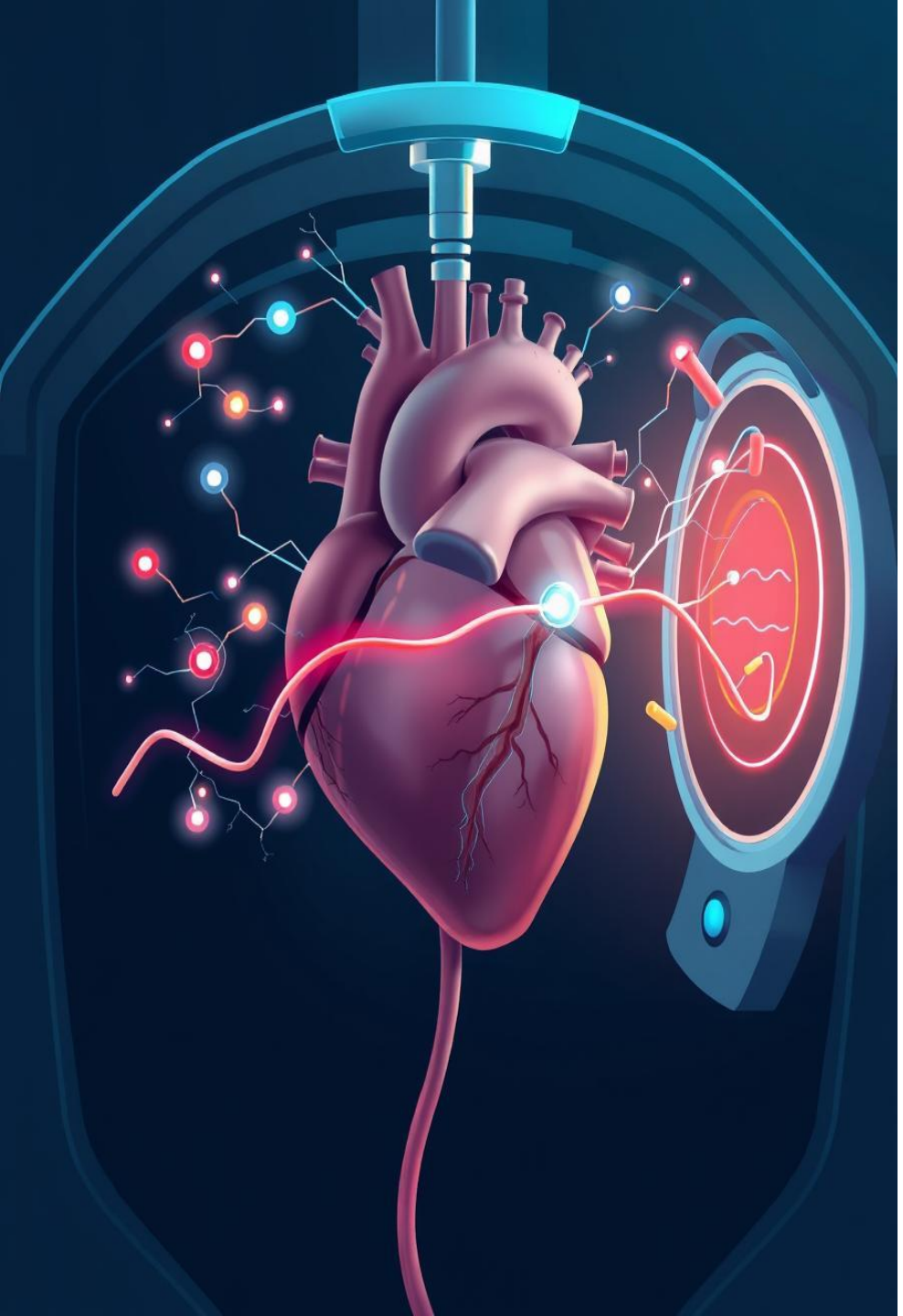
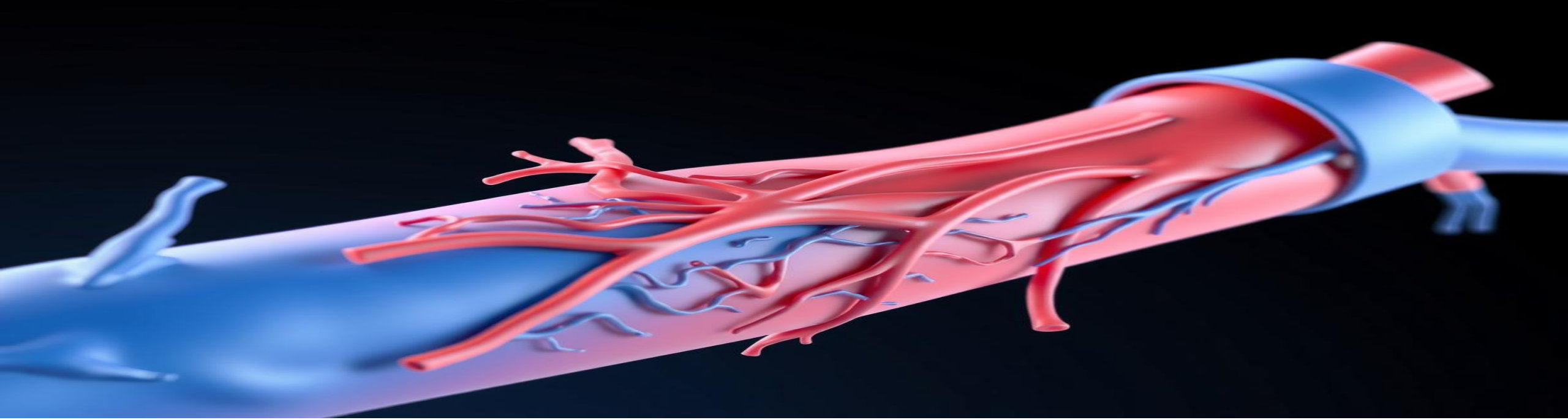


Figure : The solutions of Burger equation on a rectangular region by using (b) PINN and (a) Numerical (finite difference) methods with solitary waves $u(x,0) = \exp(-4(x-5)^2)$ as initial condition.



A Refined Physics-Informed Neural Network Framework for Solving Nonlinear Partial Differential Equations in Arterial Blood Flow Under Magnetic Field [Communicated]

This presentation explores a novel approach for solving nonlinear partial differential equations (PDEs) governing arterial blood flow in the presence of a magnetic field. We leverage the power of Physics Informed Neural Networks (PINNs) and introduce a Residual-based Adaptive Residual Enhancement (ARE) technique for enhanced accuracy and efficiency.



Introduction: Arterial Blood Flow and Magnetic Fields

Arterial Blood Flow

- Delivering oxygen and nutrients.
- Its dynamics are complex.
- Influenced by factors like vessel geometry and blood viscosity.

Magnetic Fields properties in blood flow

- Conductive properties.
- Flow Resistance.
- Changing Blood Pressures
- Applications in medical diagnostics and therapies.

Nonlinear Partial Differential Equations (PDEs)

1 Governing Equations

- Nonlinear PDEs
- Navier-Stokes equations coupled with Maxwell's equations.

2 Complexity

- Involve multiple variables
- Nonlinear terms
- Difficult to solve analytically.

Challenges in Solving Nonlinear PDEs

1

Nonlinearities

- More nonlinear terms in the equations.
- Multiple variables exist.
- Few closed-form solutions

2

Geometry

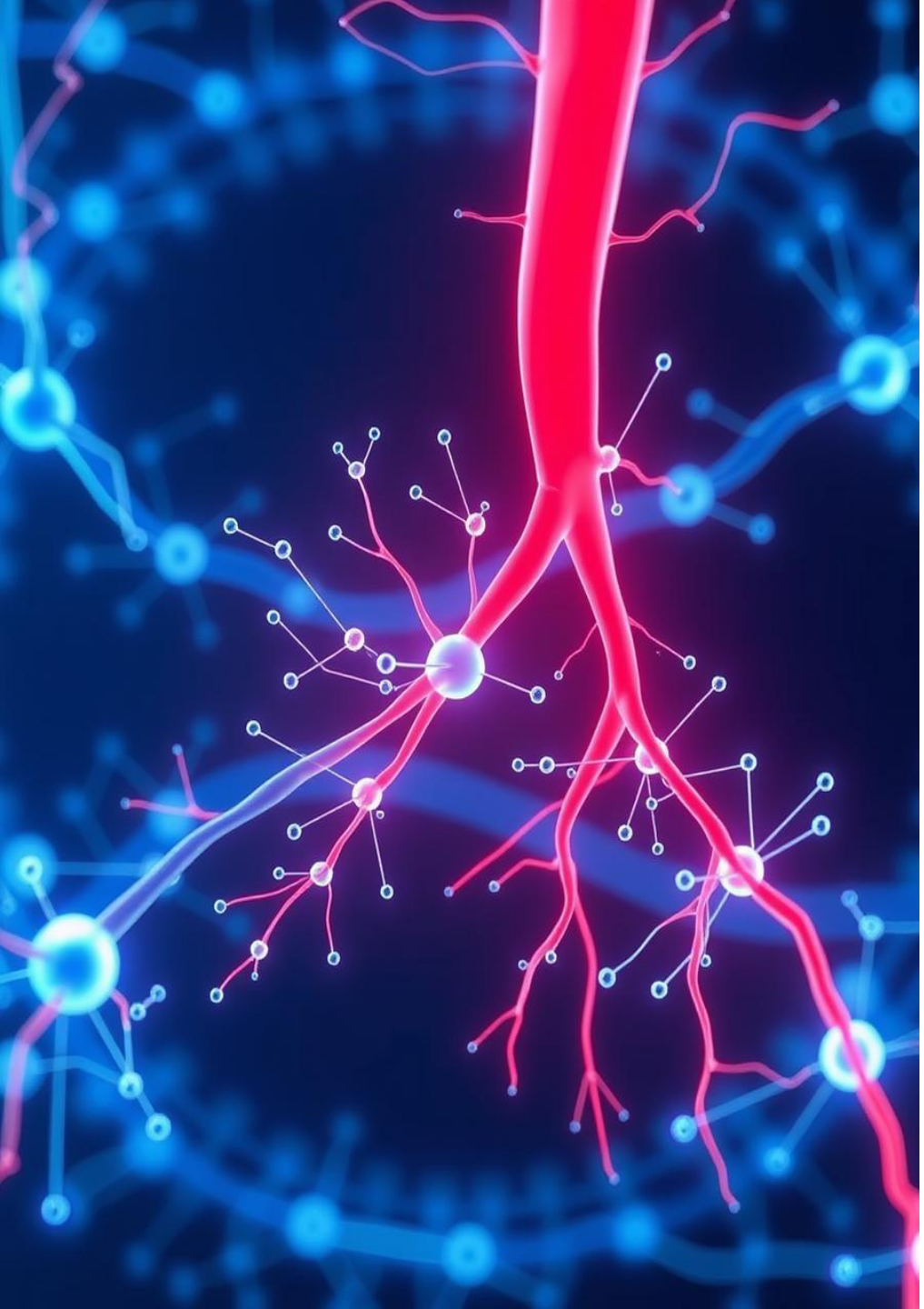
- Irregular or complex domains.
- Nonlinear boundary conditions.
- Requires fine grids or adaptive meshing.

3

Boundary Conditions

- Depends on specific Initial condition.
- Rely on suitable boundary conditions.

The boundary conditions at the vessel walls, valves, and other structures can be challenging to define accurately.



PINN Formulation for Arterial Blood Flow

1

- **Network Architecture**
- A neural network with multiple layers is constructed to approximate the solution of the PDE.

2

- **Loss Function**
- A loss function is defined that combines the error in satisfying the PDEs, boundary conditions, and data fitting.

3

- **Training**
- The network is trained using Adam optimizer, minimizing the loss function and learning the parameters that best solve the problem.

Designing PINN Architecture

Input Layer

This layer takes the space-time coordinates as input data and feeds the physical constraints into the network.

Hidden Layer

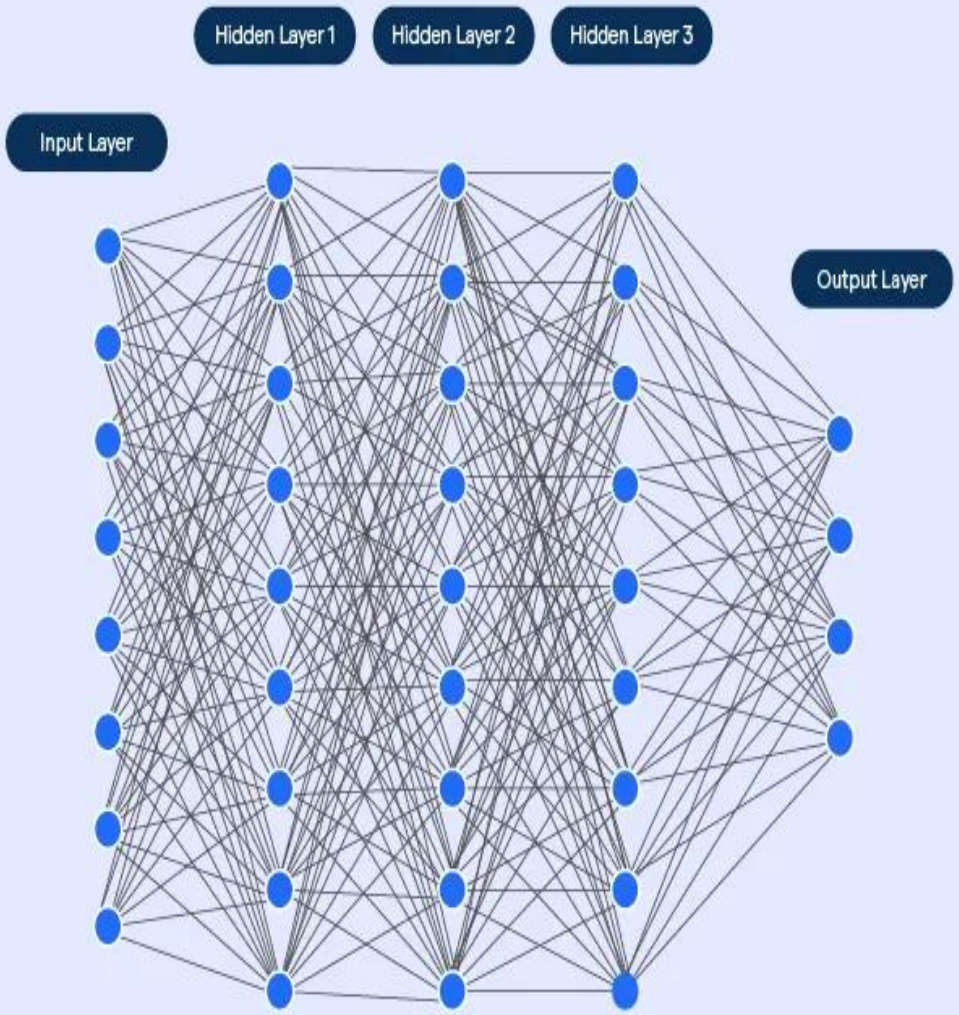
Multiple hidden layers with a sufficient number of neurons are employed. Each layer uses an activation function (e.g., sin, Sigmoid, tanh) to introduce non-linearity.

Output Layer

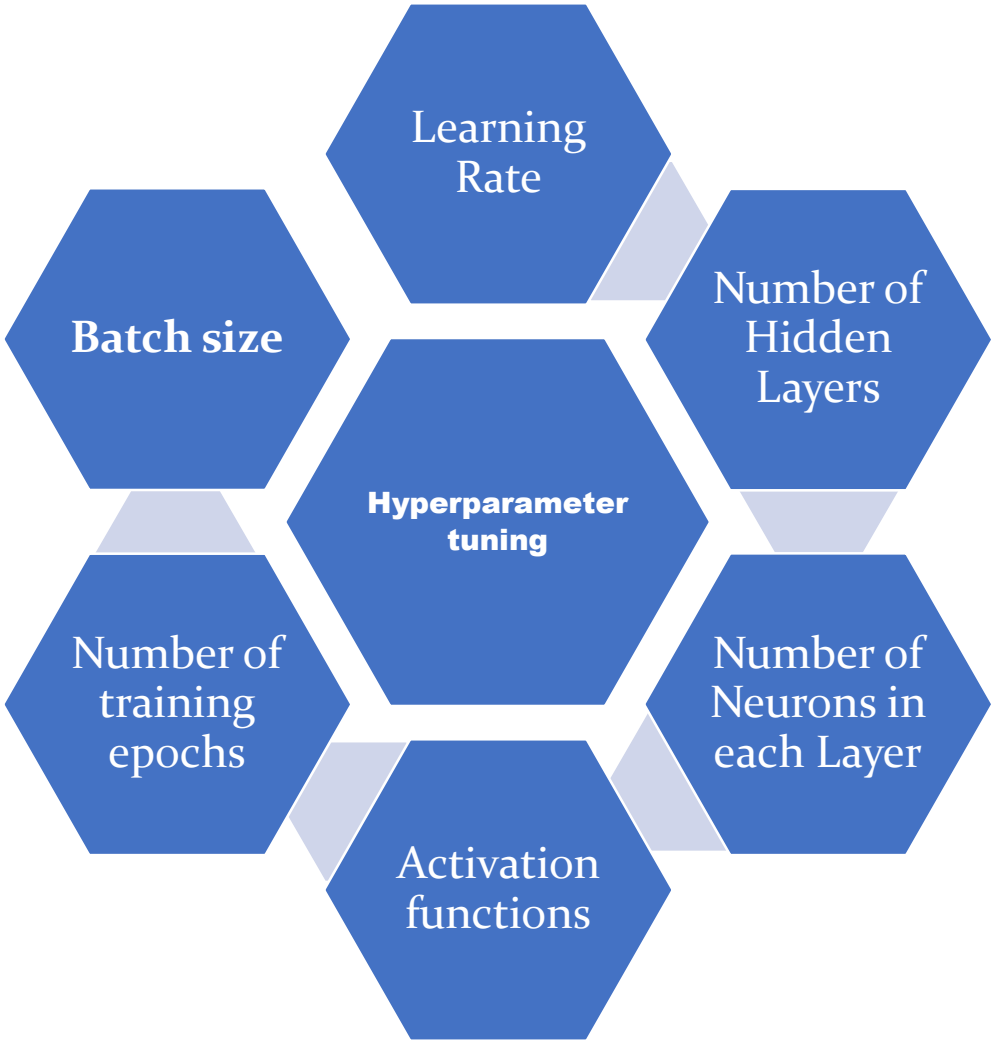
This layer predicts the solution of the PDEs. The loss function guides the training to minimize the error between the network's predictions and known physical solutions or data.

Activation Functions

A suitable activation function (e.g., sin, tanh, sigmoid) is chosen for the hidden layers.



Hyperparameter Optimization



Advantages

1. Better Tuning of hyperparameters creates the best model for training.
2. Computational cost is reduced with this hyperparameter optimizing technique.
3. Provides best results with minimum errors.
4. Requires fewer iterations to train the model.
5. Model efficiency is enhanced with high accuracy.

Some Important Equations

Equation of Continuity:

$$\frac{\partial \tilde{v_x}}{\partial x} + \frac{1}{r}\frac{\partial (r\tilde{v_r})}{\partial r} = 0$$

Equation of Momentum:

Axial component:

$$\frac{\partial \tilde{v_x}}{\partial t} + \tilde{v_r}\frac{\partial \tilde{v_x}}{\partial r} + \tilde{v_x}\frac{\partial \tilde{v_x}}{\partial x} + \frac{1}{\rho}\frac{\partial \tilde{P}}{\partial x} = v^0\left[\frac{\partial^2 \tilde{v_x}}{\partial r^2} + \frac{1}{r}\frac{\partial \tilde{v_x}}{\partial r} + \frac{\partial^2 \tilde{v_x}}{\partial x^2}\right] - \sigma B_0^2 \tilde{v_x}$$

Considering the small perturbations of the tube wall, the function $\tilde{\zeta}(x,t)$ is introduced as follows:

$$R'(x,t) = \bar{R} + \tilde{\zeta}(x,t), \quad \bar{R} = \text{equilibrium radius} \quad \|\bar{\zeta}\| << \bar{R},$$

Final Flow Equations under a small Magnetic Field

The equations of fluid flow take the following form:

$$\nu_t + \nu\nu_x = \frac{1}{2}\nu_{xx} + \frac{M}{2}\nu,$$

Burger Equation

$$\nu_t + \nu\nu_x + \frac{\gamma - \beta}{2}\nu_{xxx} = \frac{M}{2}\nu,$$

KdV: Korteweg-D Vries Equation

$$\nu_t + \nu\nu_x + \frac{1}{2}\nu_{xxxx} = \frac{M}{2}\nu.$$

Evolutionary Equation

$$\nu_t + \nu\nu_x + \frac{\gamma - \beta}{2}\nu_{xxx} + \frac{1}{2}\nu_{xxxxx} = \frac{M}{2}\nu$$

Kawahara Equation

As the initial profile, we took a periodic pulse in the form $v(x,0)$= -sin(πx).
Here, M is the magnetic field term. In the calculation, we assumed that γ-β=1

PDE Implementation in PINN

General PDE structure:

$$\mathcal{G} \equiv V_t + \mathcal{F}_V\left(x, t; V_x, V_{xx}, V_{xxx}, V_{xxxx}, V_{xxxxx} \ldots\right) = 0, \quad \mathrm{x} \in S, \quad \mathrm{t} \in (0, \hat{T})$$

PDE Residual

S is the space domain and $(0, \hat{T})$ is the time domain

Loss Function:

$$\mathrm{L}_{loss_{Trn}}\left(\boldsymbol{\Theta}; \mathcal{P}_C, \mathcal{P}_o, \mathcal{P}_B\right) = \mu_1 \mathrm{L}_1\left(\boldsymbol{\Theta}; \mathcal{P}_C\right) + \mu_2 \mathrm{L}_2\left(\boldsymbol{\Theta}; \mathcal{P}_o\right) + \mu_2 \mathrm{L}_3\left(\boldsymbol{\Theta}; \mathcal{P}_B\right)$$

Here, $\Theta = \{\mathcal{W}_j^T, \mathcal{C}^j\}_{j=1}^{l-1}$

Network Parameters

Where,

$$\mathrm{L}_1\left(\boldsymbol{\Theta}; \mathcal{P}_C\right) = \frac{1}{\mathcal{N}_c} \sum_j \left\| \tilde{V}_t + \mathcal{F}_V\left(x_j, t_j; \tilde{V}_x, \tilde{V}_{xx}, \tilde{V}_{xxx}, \ldots\right) \right\|_2^2$$

Residual Loss

$$\mathrm{L}_2\left(\boldsymbol{\Theta}; \mathcal{P}_o\right) = \frac{1}{\mathcal{N}_o} \sum_j \left\| \mathcal{I}\left(\tilde{V}(x_j, t_j); x_j, t_j\right) \right\|_2^2$$

Initial Loss

$$\mathrm{L}_3\left(\boldsymbol{\Theta}; \mathcal{P}_B\right) = \frac{1}{\mathcal{N}_b} \sum_j \left\| \mathcal{B}\left(\tilde{V}(x_j, t_j); x_j, t_j\right) \right\|_2^2$$

Boundary Loss

Predicted Solution: $V(x,t) \approx \tilde{V}(x,t) = \mathrm{NN}^{\Gamma}\left(x, t; \mathcal{P}_o, \mathcal{P}_B, \mathcal{P}_C; \Theta\right)$

Some graphical results and analysis

- **Fifth-order evolutionary equation:**

$$u_t + uu_x + \frac{\gamma - \beta}{2} u_{xxx} + \frac{1}{2} u_{xxxxx} = \frac{M}{2} u,$$

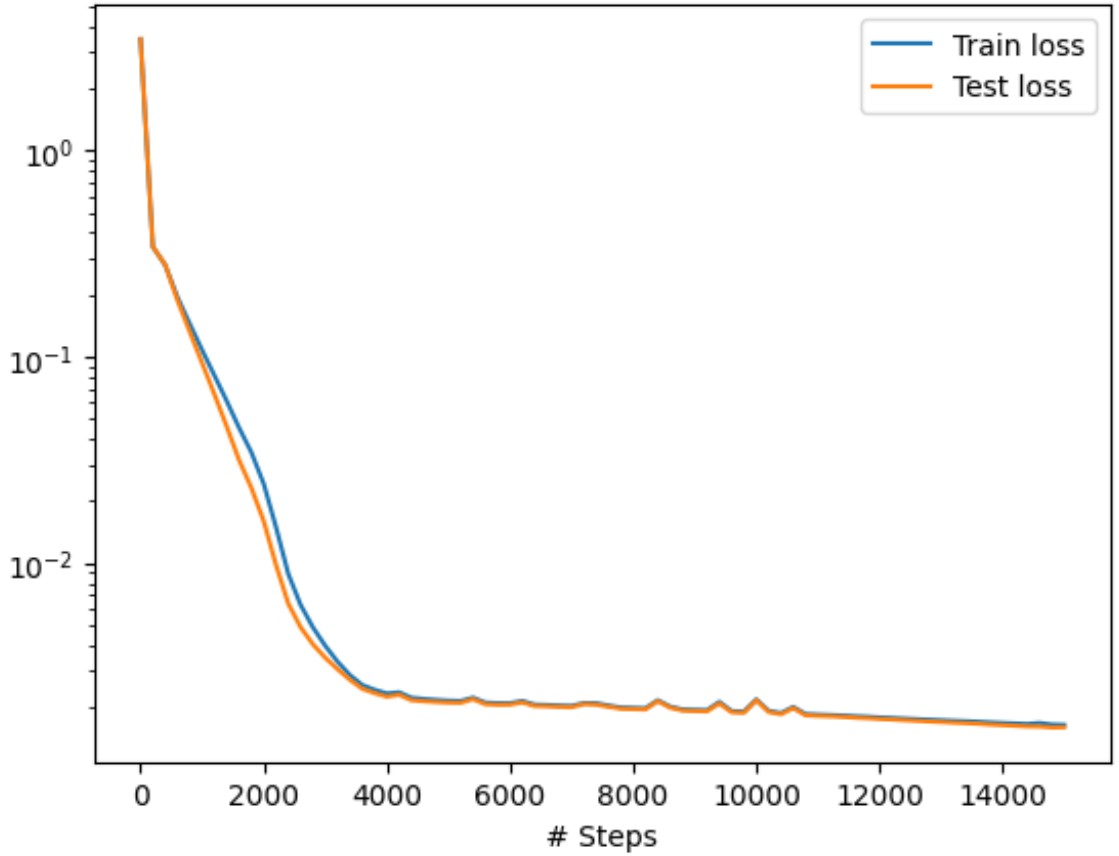
- **Initial and Boundary conditions:**

$$u(0, x) = -\sin(\pi x),$$
$$u(t, -1) = u(t, 1) = 0.$$

PINN Architecture:

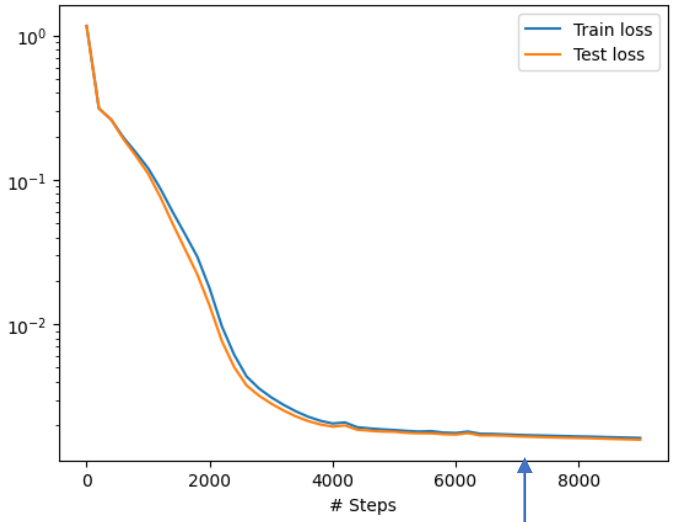
Input Layer- 2
Hidden Layers- 3
No. of Neurons in hidden layers- 20
Output Layer- 1
Activation function: sin

M = 0.02



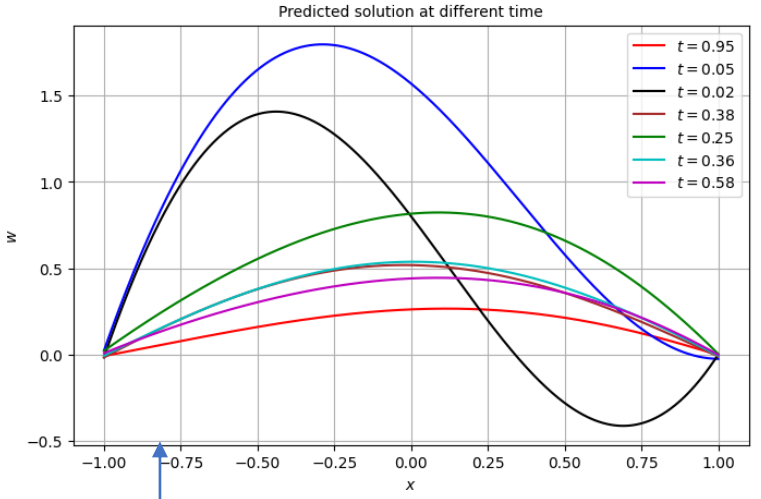
Train Loss: 1.26e-03 Test Loss: 1.22e-03

Importance of Boundary Conditions:

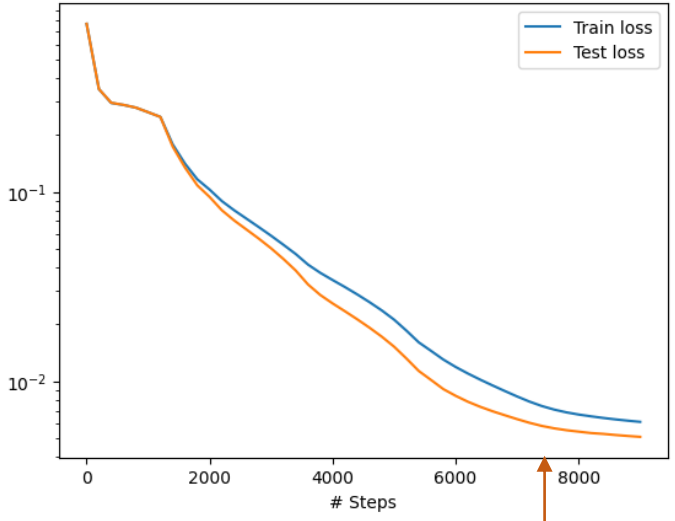


Train Loss: 1.65e-03
Test Loss: 1.61e-03

No. of Hidden Layers: 3
No. of Neurons in each Layer: 20
Activation Function: sin

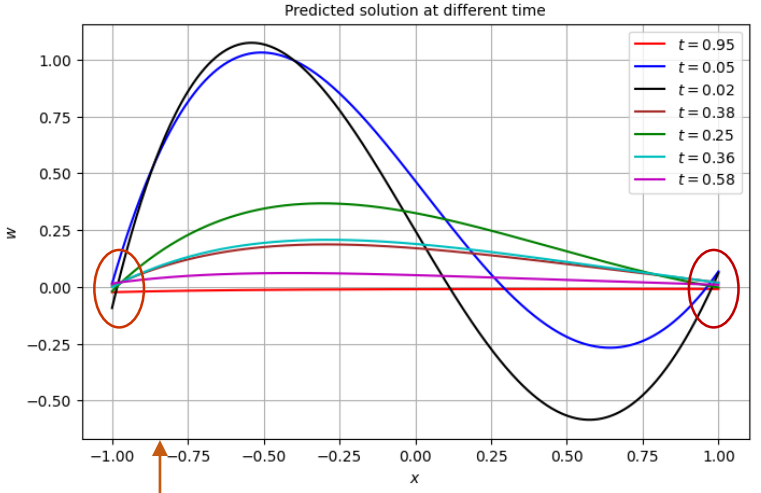


Better Predictions

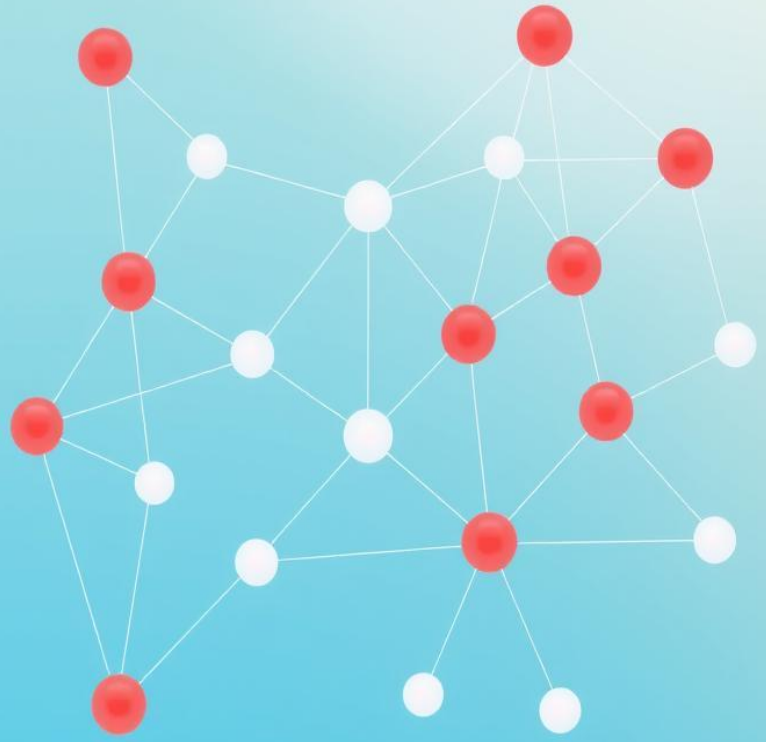


Train Loss: 6.11e-03
Test Loss: 5.09e-03

No. of Hidden Layers: 2
No. of Neurons in each Layer: 10
Activation Function: sin



Not Satisfactory



Residual-based Adaptive Refinement

1

- **Residual Analysis**
- Residuals, the difference between predicted and actual values, are analyzed to identify areas where the model needs improvement.

2

- **Adaptive Refinement**
- Neural network architecture is adaptively refined in high residual regions, focusing computational resources where needed.

3

- **Improved Accuracy**
- This technique leads to a more accurate and efficient solution by concentrating computational effort where it's most beneficial.

ARE Implementation in PINN:

Identifies high-error regions:
Identifies high-error regions: ARE evaluates the residuals across the domain and detects points where the PDE residual is significantly large.

Adaptive sampling:
It increases the sampling density around high-residual points so that the PINN focuses more on difficult regions of the PDE.

Dynamic weight adjustment:
Points with larger residuals are assigned higher weights during training, making the network prioritize reducing large errors.

Iterative refinement:
During training, the residual distribution is recalculated, and sampling weights are continuously updated to follow shifting error hotspots.

Improves convergence and accuracy:
By repeatedly concentrating learning effort on complex regions, ARE helps the PINN converge faster and achieve higher accuracy, especially for higher-order PDEs.

Burger Equation

$$\nu_t + \nu\nu_x = \frac{1}{2}\nu_{xx} + \frac{M}{2}\nu$$

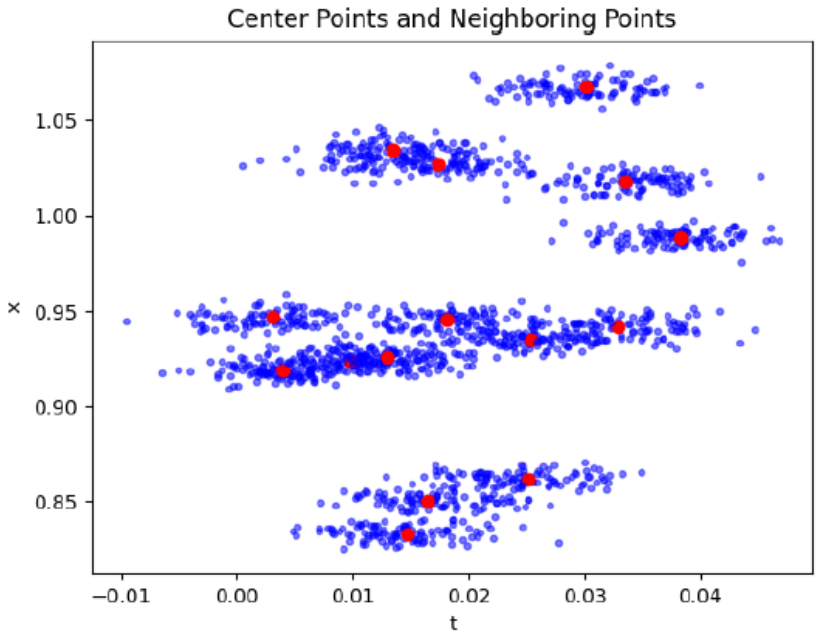


Fig: Using ARE method in forced Burgers' equation for M = 0 (a) With 15 high residual points in domain.

Implementation of ARE in Burger Equation

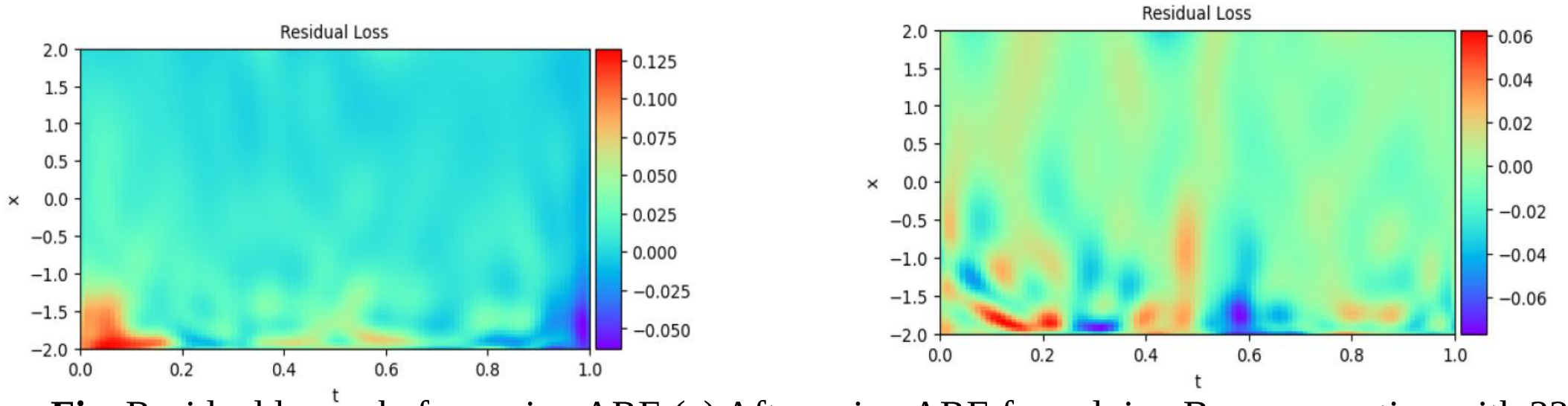


Fig. Residual losses before using ARE (c) After using ARE for solving Burger equation with 3200 points in domain.

- For the case M = 0, the PINN model was trained using 1700 collocation points within the domain. **After 14000 iterations**, the training and testing losses were 3.44×10^{-4} and 2.77×10^{-4}, respectively, with a mean residual error of 0.012.
- It was seen after utilizing the ARE method that 3200 points were sampled in the domain, and after training with these points, the mean residual became 0.007.
- This result is excellent, as we needed more than **20000 iterations** to achieve the same mean residual error by using conventional PINN algorithms.

Some graphical results and analysis

- **Fifth-order evolutionary equation:**

$$u_t + uu_x + \frac{\gamma - \beta}{2} u_{xxx} + \frac{1}{2} u_{xxxxx} = \frac{M}{2} u,$$

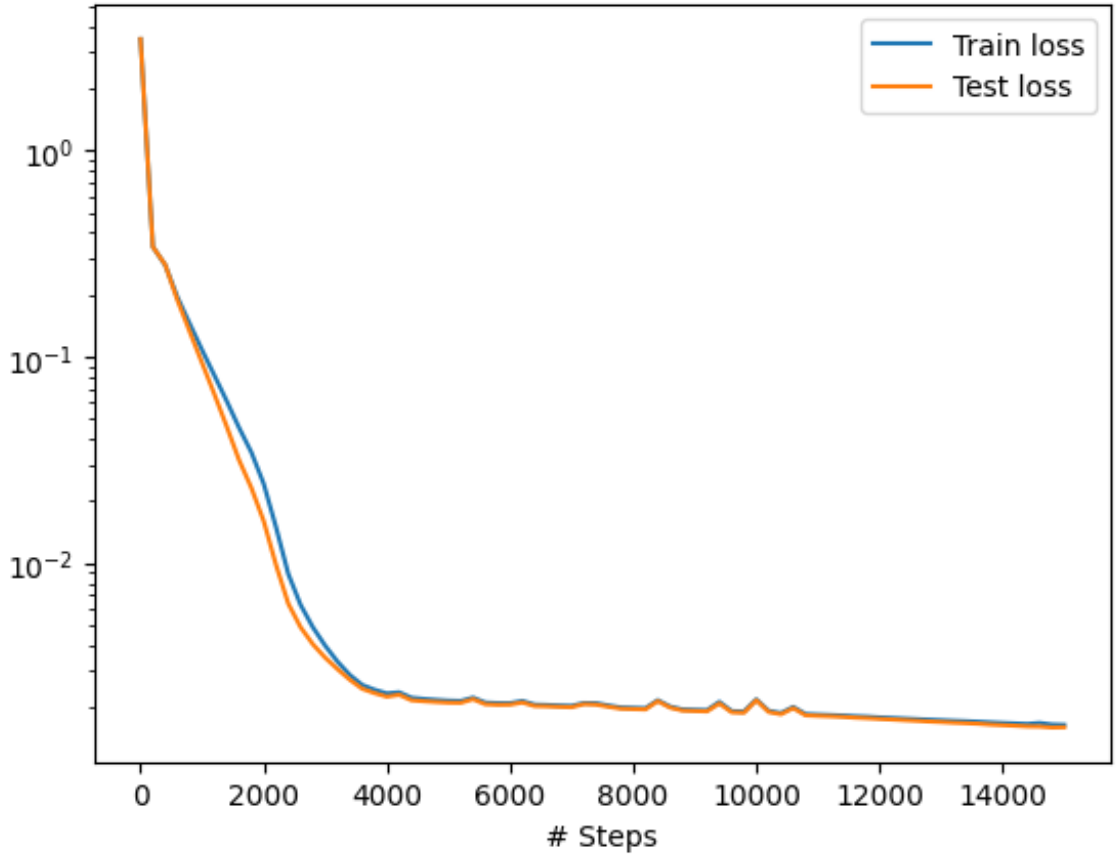
- **Initial and Boundary conditions:**

$$u(0, x) = -\sin(\pi x),$$
$$u(t, -1) = u(t, 1) = 0.$$

PINN Architecture:

Input Layer- 2
Hidden Layers- 3
No. of Neurons in hidden layers- 20
Output Layer- 1
Activation function: sin

M = 0.02



Train Loss: 1.26e-03 Test Loss: 1.22e-03

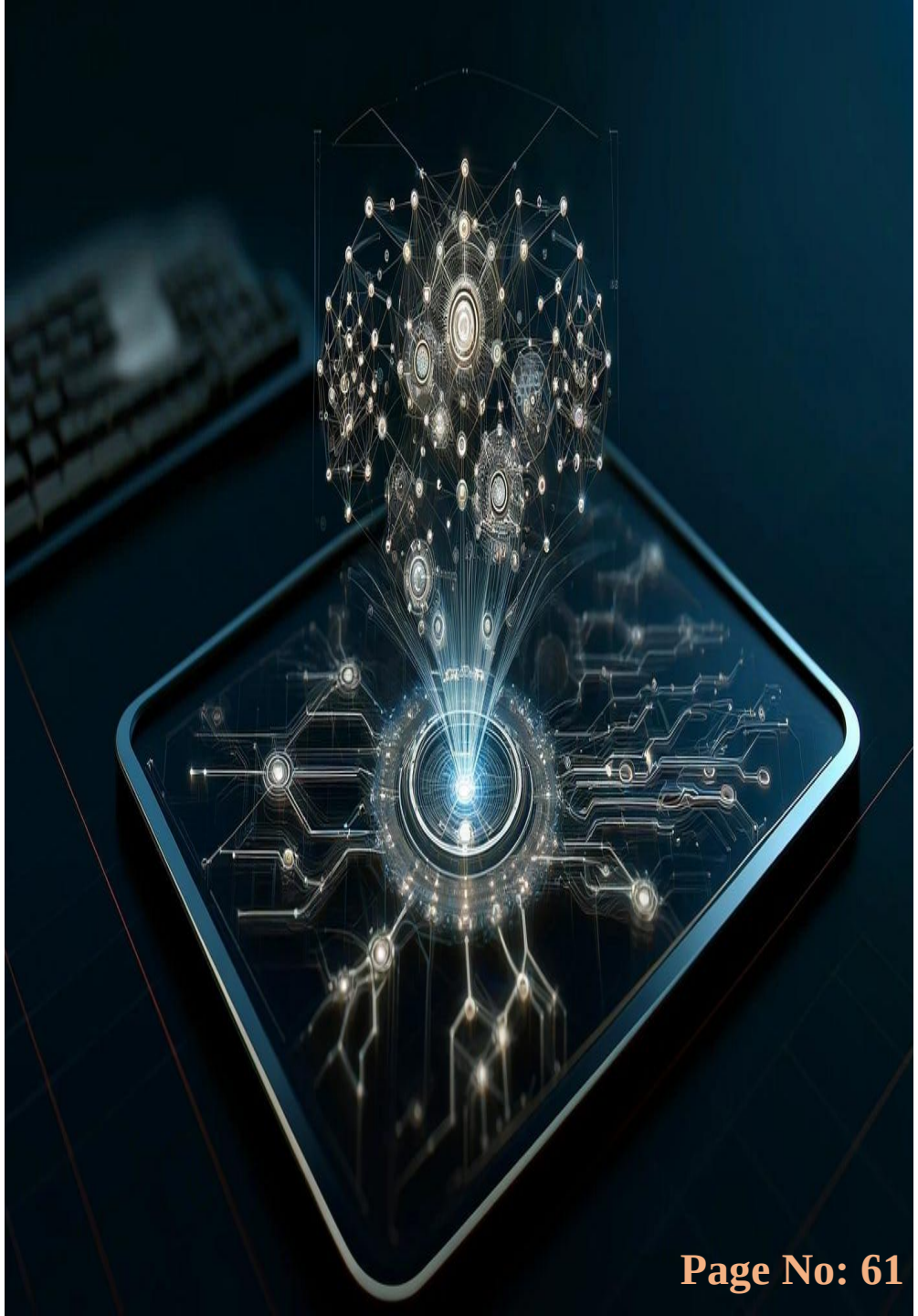
Conclusion

- A fifth-order evolutionary equation that includes the tube wall bending factor is found. These equations are calculated throughout specific periods by investigating various beginning conditions.

- The hyperparameter values are demonstrated for various PINN models to determine the PDE solution using Bayesian Optimization techniques.

- The validation of the PINN model is identified by analysing the residual error and high accuracy of the predicted solutions.

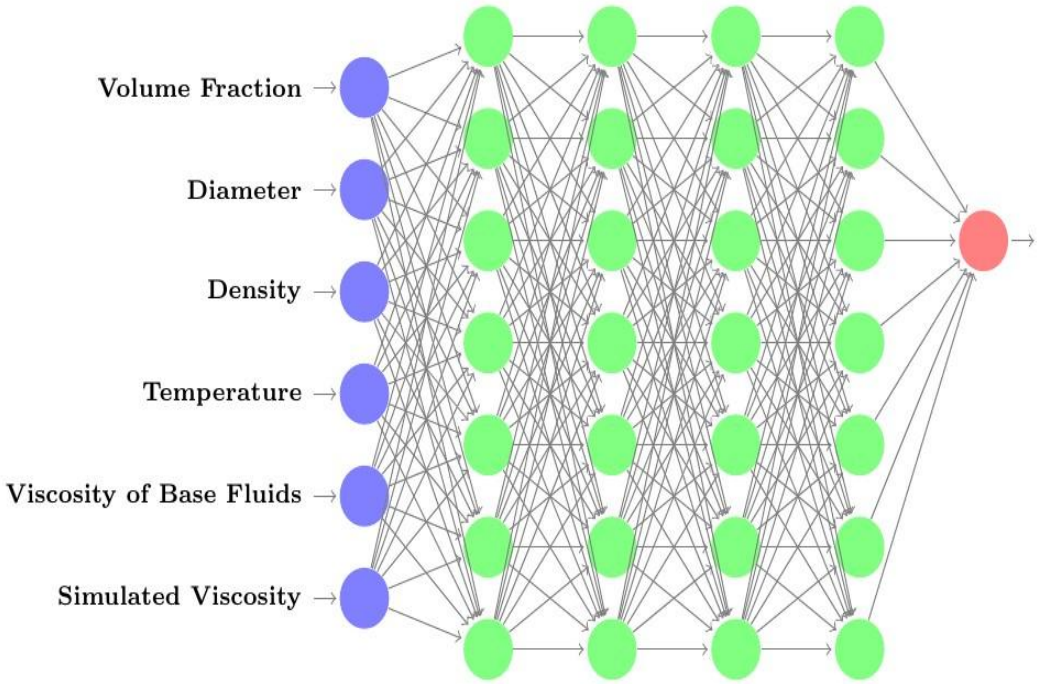
- An analytical form of the predicted solution is estimated with the help of Symbolic regression technique, interpreting the contribution of the PINN-based solution.



Published Research Work on Data-Driven Modeling

Physics-based smart model for prediction of viscosity of nanofluids containing nanoparticles using deep learning.

[Published in Journal of Computational Design and Engineering, Oxford University Press, 2021]



$$\mathrm{MSE} = \frac{\sum_{k=1}^{N} \left(\mu_{\mathrm{pre},k} - \mu_{\mathrm{exp},k}\right)^2}{N},$$

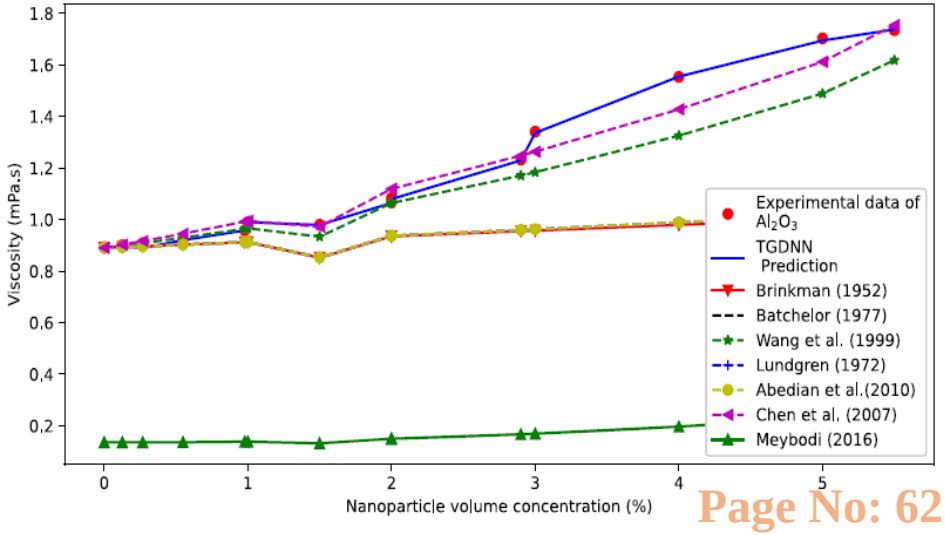
$$R_W = \lambda_1 \|W\|_1 + \lambda_2 \|W\|_2,$$

Model	RMSE
Brinkman (1952)	0.065743
Batchelor (1977)	0.065348
Lundgren (1972)	0.065348
Wang *et al.* (1999)	0.034401
Chen *et al.* (2007)	0.032188
Abedian and Kachanov (2010)	0.065338
Meybodi *et al.* (2016) (equation 14)	0.651798
This study (TGDNN)	0.001143

Masoumi et al. (2009)

$$\mu_{\mathrm{nf}} = \mu_{\mathrm{bf}} + \frac{\rho V_{\mathrm{b}} {d_{\mathrm{p}}}^2}{72 C \delta},$$

- We proposed a novel approach for predicting viscosity of water-based nanofluids using physics-guided Hybrid data model from dataset containing both experimental and simulated data of spherical oxide nanoparticles Al2O3, CuO, SiO2, and TiO2.

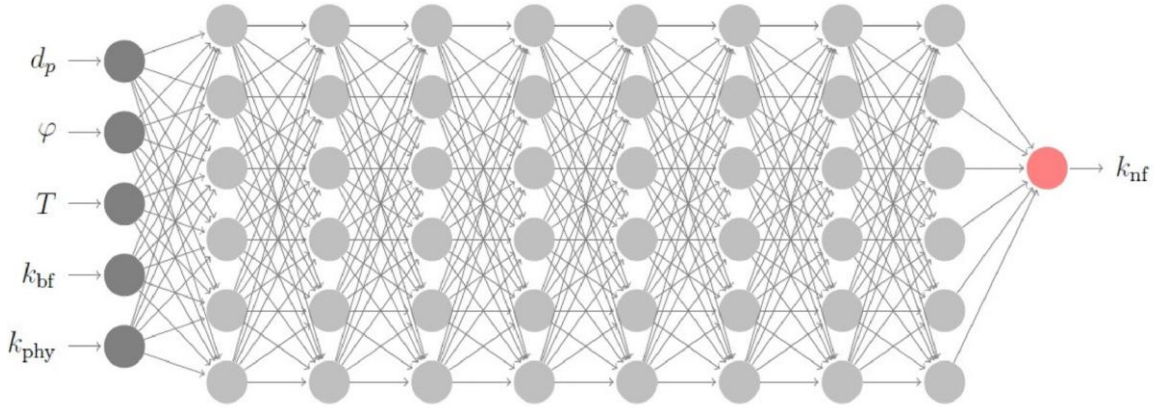


My Another Research Work

Physics-Aware Neural Network for the Prediction of Thermal Conductivity of Nanofluids

Bivas Bhaumik, Satyasaran Changdar, Soumen De

Journal of Heat Transfer American Society of Mechanical Engineers Digital Collection, 2023



Chon et al. (2005)

$$\frac{k_{nf}}{k_{bf}} = 1 + 64.7\varphi^{0.746}\left(\frac{d_{bf}}{d_p}\right)^{0.369}\left(\frac{k_p}{k_{bf}}\right)^{0.7476}\mathrm{Pr}^{0.9955}\mathrm{Re}^{1.2321}$$

Physical knowledge:

$$1 - \frac{k_{nf}[\varphi, t]}{k_{bf}} \leq 0 \quad \text{for} \quad \text{all}\ \varphi > 0 \qquad \Delta[i,t] = 1 - \frac{k_{nf}[\varphi_i, t]}{k_{nf}[\varphi_o, t]}$$

$$\text{Final Loss}_{\text{Train}} = \text{Loss}_{\text{Empirical}}(Y_{\text{exp}}, Y_{\text{pred}}) + \lambda \text{Loss}_{\text{Delta}}(Y_{\text{pred}})$$

Methods	Training R^2	Testing R^2	RMSE (W/mK)	Loss
Ridge regression	0.692974	0.675841	0.028186	0.011524
Lasso regression	0.684517	0.681612	0.026842	0.010434
Random forest (RF)	0.847051	0.796687	0.022322	0.010795
XGBoost (XGB)	0.918174	0.833770	0.020184	0.010133
MLPNN (Without physics)	0.9061	0.8846	0.0056	0.002270
Present study (Physics-aware MLP)	**0.9525**	**0.8962**	**0.0042**	**0.001143**
Buongiorno [33]	0.2807	0.3095	0.043553	0.0054
Chon et al. [34]	0.4553	0.5090	0.0768	0.0072
Corcione [35]	0.0586	0.0612	0.7767	0.5726
Garoosi [37]	0.1410	0.1163	0.1994	0.0398

SUMMARY OF DATASET

Data used in TGDNN model:

Table : Detail specifications of experimental and simulated datasets.

Experimental dataset Nanoparticle	Al_2O_3	TiO_2	SiO_2	CuO	All
Data points	450	150	100	200	900
Volume fraction (%)	0.03–13	0.20–11	0.45–4	0.15–7	0.03–13
Base fluid viscosity (mPa.s)	0.393–1.306	0.404–1.306	0.393–0.995	0.438–1.306	0.393–1.306
Size (nm)	13–100	21–76	12–76	11–33	11–100
Temperature (K)	283.15–345.15	283.15–343.15	293.15–345.15	283.15–337.15	283.15–345.15
Density (gm/cm^3)	3.7	4.26	2	6.31	2–6.31
Nanofluid viscosity (mPa.s)	0.431–4.864	0.412–3.091	0.593–4.215	0.477–4.189	0.412–4.864
Simulated dataset					
Nanoparticle	Al_2O_3	TiO_2	SiO_2	CuO	All
Data points	3990	1900	1700	1510	9100
Volume fraction (%)	0.01–10	0.01–7	0.01–7	0.01–7	0.01–7
Size (nm)	10–100	20–80	12–90	11–70	10–100
Temperature (K)	283.6–341.6	283.12–344.92	283.10–345.15	283.12–340.10	283.12–345.15
Density (gm/cm^3)	3.7	4.26	2	6.31	2–6.31
Basefluid viscosity (mPa.s) Jamshidi *et al.* (2012) $\left[\mu_w = 2.414 \times 10^{-2} \times 10^{\left(\frac{247.8}{(T-140)}\right)}\right]$	0.415–1.283	0.390–1.301	0.390–1.328	0.418–1.301	0.39–1.328
Nanofluid viscosity (mPa.s) Wang *et al.* (1999) $[\mu_{nf} = \mu_{bf}(1 + 10.6\varphi + (10.6\varphi)^2)]$	0.456–4.1578	0.436–4.1578	0.436–4.2278	0.467–4.1578	0.435–4.2278

Data used in Physics-Aware Neural Network model:

Experimental dataset				
Nanoparticle	Al_2O_3	TiO_2	CuO	All
Number of data points	420	120	130	670
Volume fraction (%)	0.0001–0.180	0.006–0.127	0.0001–0.169	0.0001–0.180
Thermal conductivity of base fluid ($Wm^{-1}K^{-1}$)	0.5581–0.6709	0.5581–0.6587	0.5777–0.6527	0.5581–0.6709
Size (nm)	10–285	21–220	10–33	10–285
Temperature (K)	274.15–362.98	274.15–340	283.15–333.25	274.15–357.00
Thermal conductivity	0.5455–1.5388	0.5462–0.8400	0.5990–0.9412	0.5455–1.5388
Simulated dataset				
Nanoparticle	Al_2O_3	TiO_2	CuO	All
Number of data points	4580	1380	1370	7330
Volume fraction (%)	0.0013–0.10	0.0001–0.1201	0.0001–0.10	0.0001–0.1201
Size (nm)	11–150	20–200	10–35	10–200
Temperature (K)	283.16–344.16	283.12–340.12	288–333	283.12–344.16
Thermal conductivity of base fluid ($Wm^{-1}K^{-1}$) [54]	0.5777–0.6617	0.5778–0.6610	0.5877–0.6527	0.5777–0.6610
Thermal conductivity [32]	0.5796–0.7760	0.5836–0.7995	0.5942–0.7650	0.5796–0.7995

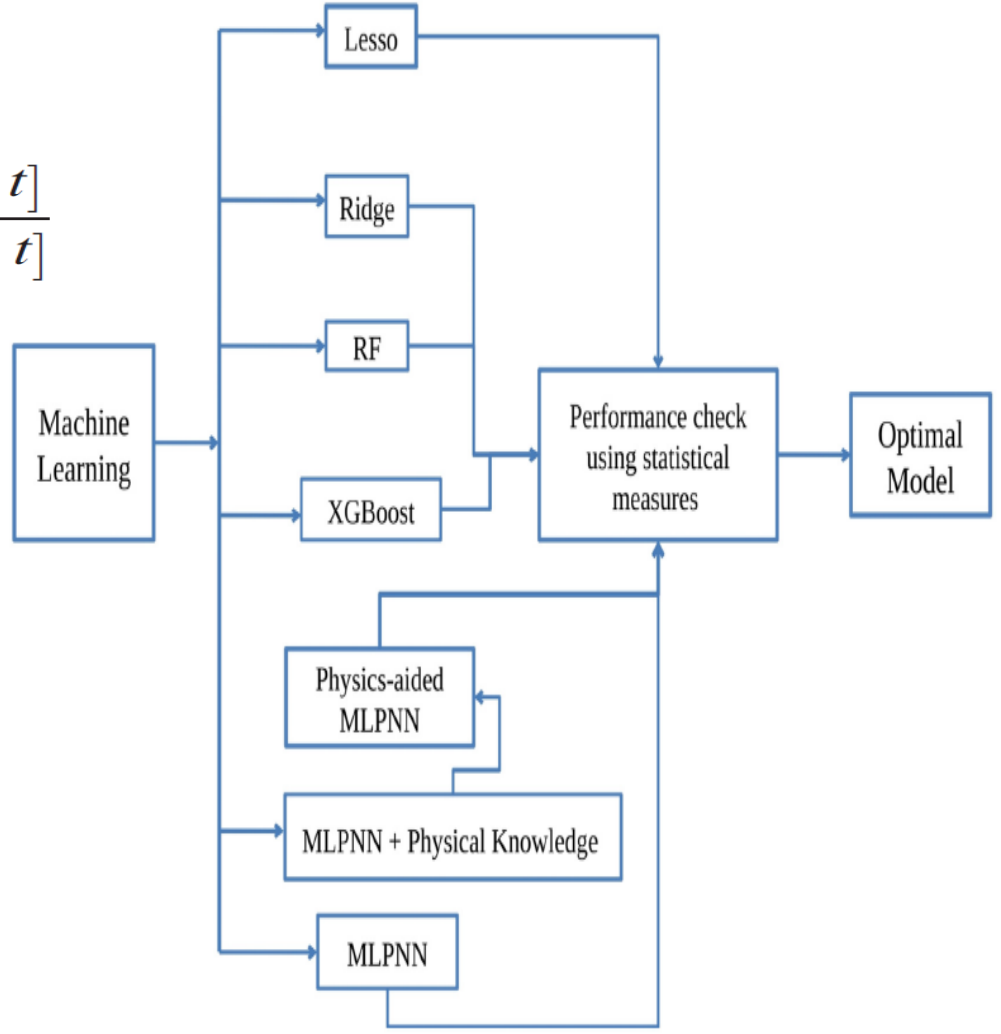
ARCHITECTURES Of LOSS FUNCTIONS & COMPARISON

Loss function in Physics-Aware Neural Network model:

$$\text{Final Loss}_{\text{Train}} = \text{Loss}_{\text{Empirical}}(Y_{\text{exp}}, Y_{\text{pred}}) + \lambda \text{Loss}_{\text{Delta}}(Y_{\text{pred}})$$

$$\text{MSE} = \frac{\sum_{k=1}^{N} (Y_{\text{pred},k} - Y_{\text{exp},k})^2}{N} \quad \text{and} \quad \Delta[i,t] = 1 - \frac{k_{\text{nf}}[\varphi_i, t]}{k_{\text{nf}}[\varphi_o, t]}$$

$$\text{Loss}_{\text{Delta}}(Y_{\text{pred}}) = \frac{1}{n_d(n_t - 1)} \sum_{t=1}^{n_t - 1} \sum_{i=1}^{n_d} \text{ReLU}(\Delta[i,t])$$



Loss used in TGDNN model:

$$\text{MSE} = \frac{\sum_{k=1}^{N} (\mu_{\text{pre},k} - \mu_{\text{exp},k})^2}{N}, \quad \text{and} \quad R_W = \lambda_1 \|W\|_1 + \lambda_2 \|W\|_2,$$

$\mu_{\text{exp},k}$ and $\mu_{\text{pre},k}$ are the true and predicted values of the k-th input data, respectively, and λ_1 and λ_2 are hyper-parameters to control the network weights W of model complexity loss.

STATISTICAL MEASURE

Statistical criteria applied for model validation.

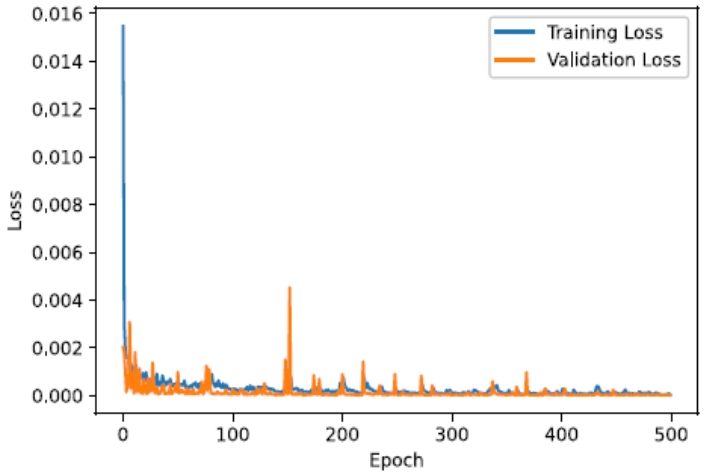
Statistical criterion	Expression
RMSE	$\sqrt{\frac{\sum_{k=1}^{N}\left(\mu_{\text{pre},k}-\mu_{\text{exp},k}\right)^2}{N}}$
AARD(%)	$\frac{100}{N}\sum_{k=1}^{N}\frac{\left\|\mu_{\text{pre},k}-\mu_{\text{exp},k}\right\|}{\mu_{\text{exp},k}}$
R^2	$1-\frac{\sum_{k=1}^{N}\left(\mu_{\text{pred},k}-\mu_{\text{exp},k}\right)^2}{\sum_{k=1}^{N}\left(\mu_{\text{pred},k}-\text{average}(\mu_{\text{exp},k})\right)^2}$

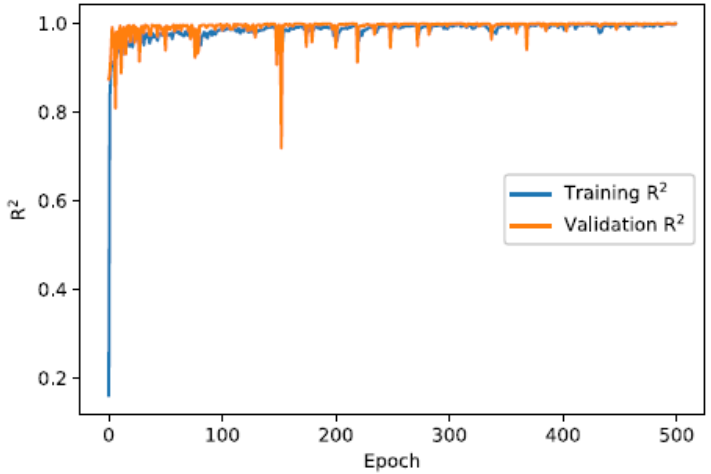
IMPORTANT FINDINGS

- The concept of integrating physics-aware learning objectives at the training stage not only generates physically relevant predictions but also provides a robust model that has better generalization capabilities.
- A comparative analysis is done between the physics-aware MLPNN model and several black-box ML models Ridge regression, Lasso regression, Random Forest, XGBOOST, and multilayer perceptron. The outcomes of these comparisons reveal that the proposed model provides better predicted results.
- The results of our studies (TGDNN) reveal that the values of R^2 , RMSE, and AARD% are 0.999868, 0.001143, and 2.198887, respectively.

IMPLEMENTATION

- The model was trained using Keras package and TensorFlow backend on Google Colab using python programming language.
- Adam optimizer was used to minimize the loss function of the model parameters of the neural network and batch size of 256 with maximum number of epochs equal to 500.
- The architecture is formed by fully-connected neural network using Keras Dense layer.





- Finally, we save both the neural network model architecture and its learned weights in the .h5 format. The trained model can then be deployed for viscosity prediction of nanofluids.

References

- **Bhaumik, B.**, De, S. and Changdar, S., 2024. Deep learning based solution of nonlinear partial differential equations arising in the process of arterial blood flow. **Mathematics and Computers in Simulation,** 217, pp.21-36. **[SCI, IF-4.4]**

- **Bhaumik B.**, Changdar, S., Chakraverty, S. and Soumen De (2024) Effects of viscosity and induced magnetic fields on weakly nonlinear wave transmission in a viscoelastic tube using physics-informed neural networks, ***Physics of Fluids*** *(American Institute of Physics-*), **36(12),** 121902. **[SCI, IF-4.1]**

- Changdar, S., **Bhaumik, B.**, Sadhukhan, N., Pandey, S., Mukhopadhyay, S., De, S. and Bakalis, S. **(2024)** Integrating Symbolic Regression with Physics-Informed Neural Networks for Simulating Nonlinear Wave Dynamics in Arterial Blood Flow, ***Physics of Fluids*** *(American Institute of Physics-*)**36(12). [SCI, IF-4.1]**

- Wang, Y. and Zhong, L., 2024. NAS-PINN: neural architecture search-guided physics informed neural network for solving PDEs. Journal of Computational Physics, 496, p.112603.

- **Bhaumik, B.**, Chaturvedi, S., Changdar, S. and De, S., 2023. A unique physics-aided deep learning model for predicting viscosity of nanofluids. **Int. J. Comput. Methods Eng. Sci. Mech.**, **24(3)**, pp.167-181. **[SCI, IF-1.4]**

- **Bhaumik, B.**, Changdar, S., and De, S., 2022. An expert model based on physics-aware neural network for the prediction of thermal conductivity of nanofluids. ***ASME Journal of Heat and Mass Transfer***, **144(10)**, pp.103501. **[SCI, IF-2.8]**

- Mekheimer, K.S. and El Kot, M.A., 2008. Influence of magnetic field and Hall currents on blood flow through a stenotic artery. Applied Mathematics and Mechanics, 29, pp.1093-1104.

- Raissi, M., Perdikaris, P. and Karniadakis, G.E., 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys., 378, pp.686-707.

Thank You For Listening

Any Query ?

THE END