Neural Network approaches for Physical Problems(with application to SDOF system)

Aditya Garg

SRF student, NMCAD lab, Dept. of Aerospace Engineering, IISc

Neural Networks

- A neural network is a deep learning method that is used for value prediction, pattern recognition and other applications
- A neural network is characterized by the structure of its layers, associated activation functions and the pattern of connection between its layers(weights and biases).
- Each neuron acts on the net input through an activation function. Commonly used activation functions are Hyperbolic tangent, Rectified linear unit, bipolar sigmois.

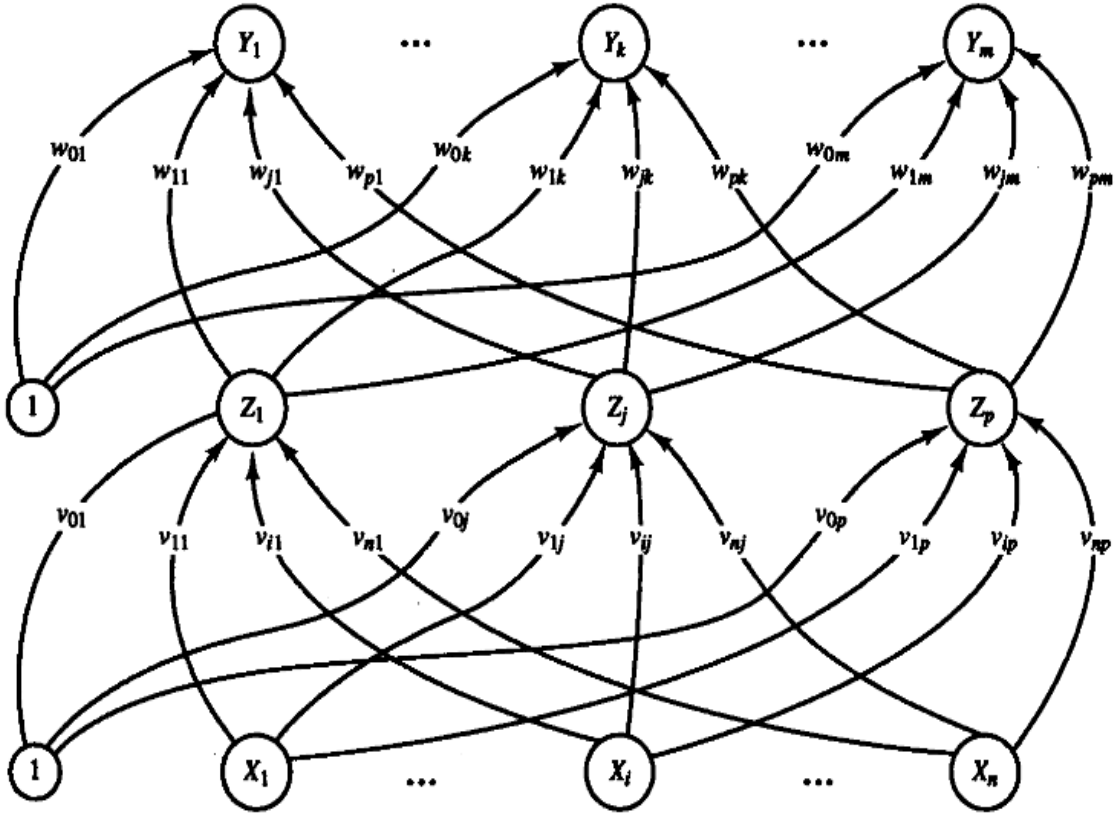


Fig 1: An example of neural network architecture (from [4])

Working of Neural Networks

The example assumes a network with 1 hidden layer similar to fig. in slide 2

Step 0:Initialize weights

Step 1:For each training pair:

— Step 2 :Each input unit receives an input signal and, broadcasts this signal to all units in the next layer.

— Step 3:Each hidden unit sums its weighted input signals, and applies activation function

$$z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i v_{ij} \qquad z_j = f(z\_in_j) \tag{1}$$

— Step 4: Output unit calculates output signal

$$y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j w_{jk} \qquad y_k = f(y\_in_k) \tag{2}$$

Working of Neural Networks(contd.)

Backpropagation of error:

Step 5:For each output unit calculate the error as

$$\delta_k = (t_k - y_k) f'(y\_in_k) \tag{1}$$

similarly weight correction $\Delta\omega_j k = \alpha\delta_k z_j$ and bias correction $\Delta\omega_{0k} = \alpha\delta_k$

Step 6:For each hidden unit calculate its reverse input as :$\delta\_in_j = \sum_{k=1}^{m} \delta_k \omega_{jk}$

Hence the back propagated error is:$\delta_j = \delta\_in_j f'(z\_in_j)$,

similarly weight correction $\Delta\nu_j k = \alpha\delta_j x_i$ and bias correction $\Delta\nu_{0j} = \alpha\delta_j$

Step 7: Change the weights and repeat for all training points:

— $\omega_{jk}(new) = \omega_{jk}(old) + \Delta\omega_{jk}$ $\nu_{ij}(new) = \nu_{ij}(old) + \Delta\nu_{ij}$

Need for Physics Informed Neural Networks(PINNs)

Neural networks have been developed to predict various functions accurately. But when applied to the problem of predicting the solution of a physical problem, they are not enough. A neural network fits a function to given labelled data, however it doesn't ensure whether the fitted function is a valid solution or not. A solution to a given physical problem should satisfy all the initial and boundary conditions. While neural networks don't take into account the information about boundary and initial conditions, Physics Informed Neural Networks use this information while at the same time attempting to fit a function to the labelled data.

Physics Informed Neural Networks(PINNs)

Consider solving the differential equation

$$\mathcal{N}_{\mathbf{x}}(u) = f(\mathbf{x}), x \in \mathcal{D}_x \qquad \mathcal{B}_{\mathbf{x}}(u) = b, \mathbf{x} \in \Gamma \tag{1}$$

where $\mathcal{N}$ is a differential Operator acting on u; u is the vector repressenting the state of the system x is the input to the system; f is the forcing term B is boundary condition operator

The goal of PINNs is to build a model to approximate the state of the system,i.e., to approximate $\mathbf{x}$. That is we have $g(t;\Theta) = \hat{u}(t;\Theta) \approx u(x;u;\mathcal{B}_x)$ where x_0 are the initial conditions the system is subjected to. A standard neural network solver with rms error as following aims to fit a function to the dataset $\mathcal{S}$ without caring whether the fit wil be a valid.

$$\underset{\Theta}{min} \quad \frac{1}{N}\sum_{i=1}^{N}\sqrt{\hat{u}^{(i)}(\Theta) - (u_{true}^{(i)} + \eta^{(i)})^2} \tag{2}$$

Physics Informed Neural Networks(PINNs) contd.

The basic difference between a normal neural network and a PINN is in the error term.The PINNs account for the physical conditions by introducing an error term derived from the initial conditions.Thus, making the approximation by a PINNs a more accurate and a valid solution of the physical system,i.e,PINNs consider the error term given by

$$h(u,x) = \mathcal{N}(\hat{u}) - f(\hat{u};x) \qquad h1(u,x) = \mathcal{B}(\hat{u}) - b(\hat{u};x) \tag{1}$$

along with the rms error term and thus,ensuring that physical conditions are not violated.Thus, the total error to consider while training is

$$w_1 \frac{1}{N} \sum_{i=1}^{N} \sqrt{\hat{u}^{(i)}(\Theta) - (u_{true}^{(i)} + \eta^{(i)})^2} + w_2 h(u,x) + w_3 h(u,x) \tag{2}$$

where w_1,w_2 and w_3 are training weights.

Need for Uncertainty quantification

Although Physics Informed Neural Networks(PINNs) can estimate physical functions to good enough accuracies, it is always desirable to have information of the variability of the prediction. For example , In a problem of predicting the displacement of one end of a rod clamped at the other end, we always want to ensure that the rod doesn't break in the process, hence knowing information about the maximum variance in the predicted displacement can be used to ensure that the wing doesn't break. PINNs can be fused with another variation of neural networks called "Bayesian Neural Networks" to get a method known as "Bayesian-Physics Informed Neural Networks(B-PINNs)" which enables us to predict the uncertainty in our prediction.

Bayesian Neural Networks

Bayesian Neural networks are a probabilistic approach to neural networks where weights and biases are assumed from a stochastic process. A commonly used special case is where the assumed stochastic process is a Gaussian Process. The above can be stated as

$$\theta \sim p(\theta) \tag{1}$$

$$y = \Phi_\theta(x) + \epsilon \tag{2}$$

where θ is the vector of model parameteres(weights and biases) and is from a random process p. y is the function we want to approximate.Φ represents the prediction of the neural network which depends on θ.ϵ represents the error in the fit.

Bayesian Neural Networks

The likelihood can be calculated as

$$p(D|\theta) = P(\mathcal{D}_y|\mathcal{D}_x) \tag{1}$$

where $\mathcal{D}_y$ represents the response training data and $\mathcal{D}_x$ represents the input training data The posterior then can be calculated from the Bayes' Theorem as

Bayes Theorem:
$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

$$p(\theta|D) = \frac{P(\mathcal{D}_y|\mathcal{D}_x)p(\theta)}{\int_\theta P(\mathcal{D}_y|\mathcal{D}_x)p(\theta')d\theta'} \tag{2}$$

The resultant probability distribution after training with the data can then be used to calculate uncertainities releted to the model. For large datasets calculating the posterior as above is not always feasible.Thus, We make use of techniques like Markov Chain Monte Carlo and Variational Inference

Bayesian PINNs

Bayesian PINNs emerge as a fusion of PINNs and Bayesian neural networks.Again the physics error term is the difference between BNNs and B-PINNs. The calculation for likelihood changes for B-PINNs and is given by

$$p(D|\theta) = P(\mathcal{D}_y|\mathcal{D}_x) \prod_{j=1}^{2} \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma_{h_j^{(i)2}}}} exp(-\frac{(h_j^{(i)}(u,x))^2}{2\sigma_{h_j^{(i)2}}}) \tag{1}$$

and hence the posterior is now calculated as:

$$p(\theta|D) = \frac{P(\mathcal{D}_y|\mathcal{D}_x)p(\theta) \prod_{j=1}^{2} \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma_{h_j^{(i)2}}}} exp(-\frac{(h_j^{(i)}(u,x))^2}{2\sigma_{h_j^{(i)2}}})}{\int_\theta P(\mathcal{D}_y|\mathcal{D}_x)p(\theta')d\theta'} \tag{2}$$

An assumption of Gaussian Process for errors has been made to write the likelihoods

Bayesian PINNs

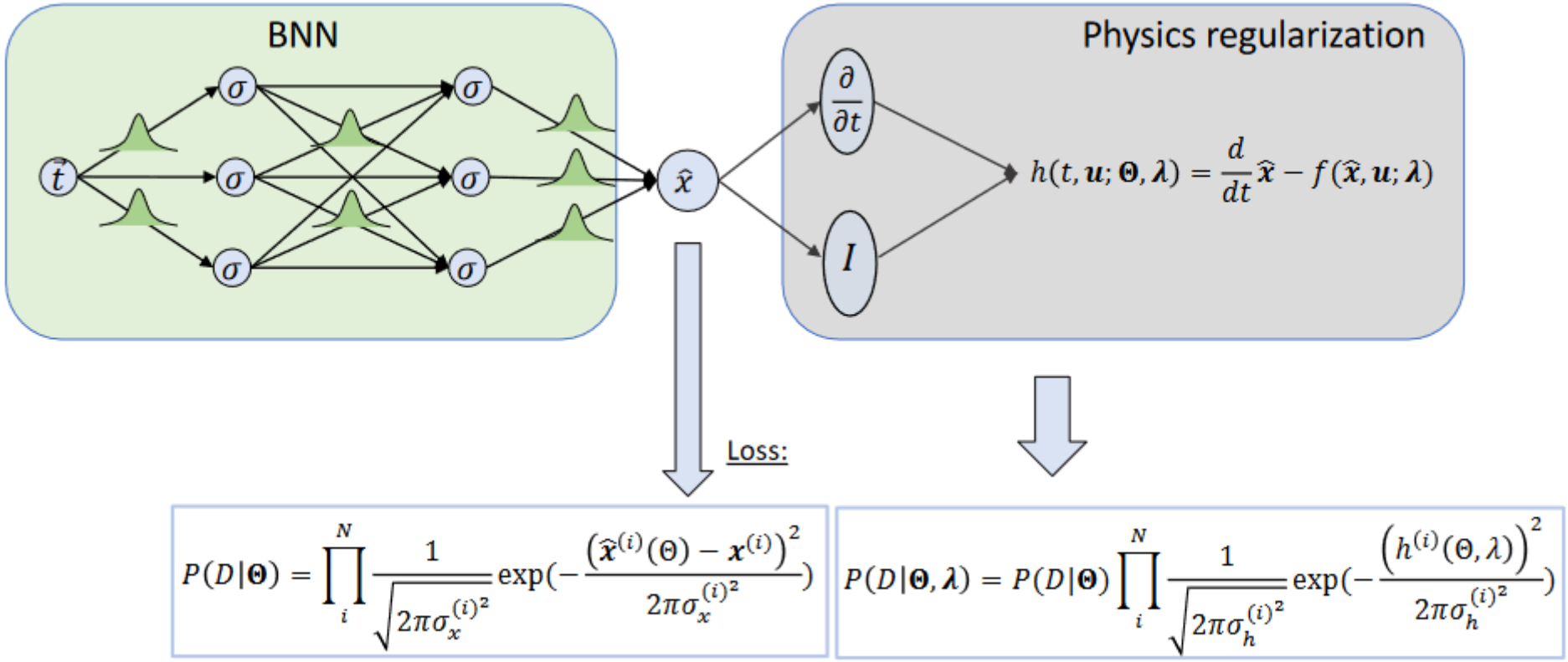


Fig 2: A depiction of working of Bayesian PINNs(from [1])

Differences in the three methods

Neural Networks

- Aims to fit a function to the given data without accounting for physical conditions.
- The fit might not be an appropriate solution for the physical problem.
- Weights and biases are discrete.
- Does not account for the uncertainty of the model

Physics Informed Neural Network

- Aims to fit a function to the given data while accounting for physical conditions as well.
- The fit is an appropriate solution for the physical problem.
- Weights and biases are discrete.
- Does not account for the uncertainty of the model

Bayesian PINNs

- Aims to fit a function to the given data while accounting for physical conditions as well.
- The fit is an appropriate solution for the physical problem.
- Weights and biases are assumed to follow a probability distribution
- Does account for the uncertainty of the model

Single degree of freedom(SDOF) systems(Theory)

A single degree of freedom system can be modelled as a mass attached to a spring undergoing free vibrations with damping as shown in the figure. Looking at the free body diagram, one can write the governing equations as:

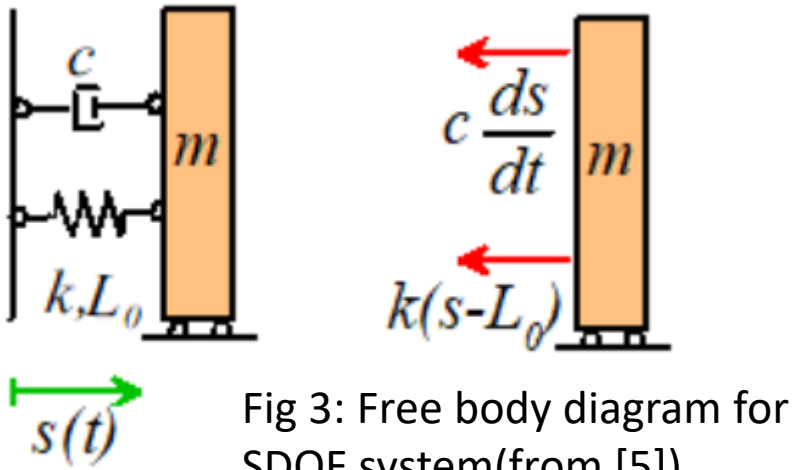


Fig 3: Free body diagram for SDOF system(from [5])

$$k(s - L_0) + c\frac{ds}{dt} = ma = m\frac{d^2s}{dt^2} \tag{1}$$

$$\Rightarrow \frac{m}{k}\frac{d^2s}{dt^2} + \frac{c}{k}\frac{ds}{dt} + s = L_0 \tag{2}$$

Single degree of freedom(sdof) systems(Solution)

The equation can be rewritten as:

$$\frac{1}{w_n^2}\frac{d^2x}{dt^2} + \frac{2\zeta}{\omega_n}\frac{dx}{dt} + x = 0 \tag{1}$$

where

$$s = x \qquad \omega_n = \sqrt{\frac{k}{m}} \qquad \zeta = \frac{c}{2\sqrt{(km}} \tag{2}$$

The general solution of the system is given as:

$$x(t) = Ce^{\zeta\omega_n t}cos(\omega_d t - \phi) \tag{3}$$

where

$$C = \sqrt{x_0^2 + (\frac{\zeta\omega_n x_0 + v_0}{\omega_d})^2} \qquad \phi = tan^{-1}\frac{\zeta\omega_n x_0 + v_0}{\omega_d x_0} \qquad \omega_d = \sqrt{1-\zeta^2}\omega_n \tag{4}$$

Ex: Single degree of freedom(SDOF) systems PINN(Code:DeepXde)

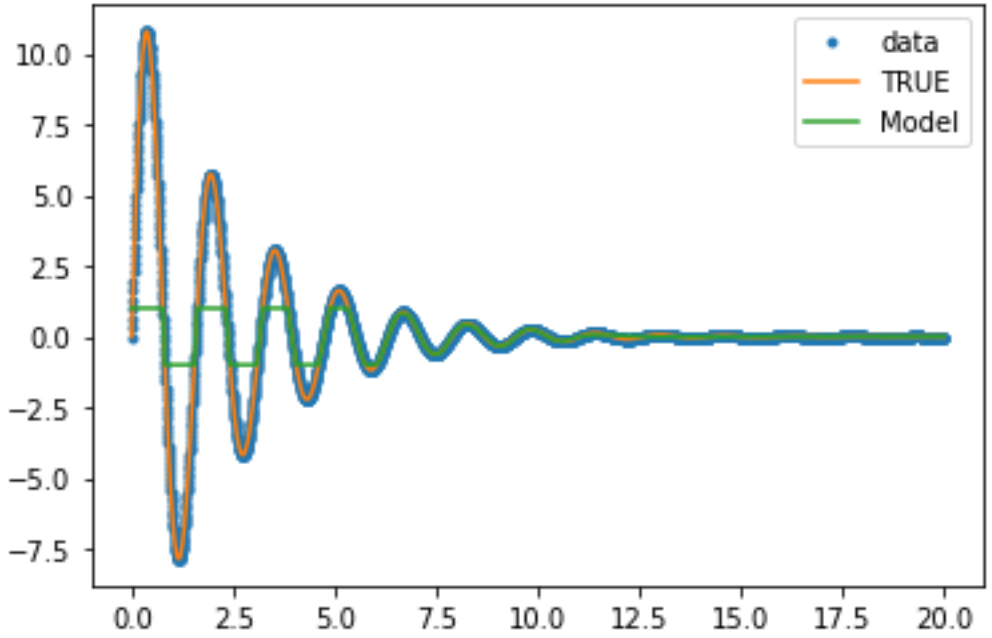
```
import deepxde as xd
import numpy as np
import math
from matplotlib import pyplot as plt
#Physical Cnstants
w_n=4
zeta=0.1
#w_n=float(input("W_n:"))
#zeta=float(input("Zeta:"))
#Defining the differential Equation
def ode(x,y):
    y_der=xd.grad.jacobian(y,x)
    y_dder=xd.grad.hessian(y,x)
    return y_dder + y_der*2*zeta*w_n+y*w_n**2
#Initial Conditions
x0=0
v0=50
#Analytical Solution
w_d=np.sqrt(1-zeta**2)*w_n
C=math.sqrt(x0**2+((zeta*w_n*x0+v0)/w_d)**2)
if w_d*x0==0:
    phi=math.pi/2
else:
    phi=math.atan(((zeta*w_n*x0)+v0)/w_d/x0)
def soln(t):
    exponent=np.exp(-zeta*w_n*t)
    return C*exponent*(np.cos((w_d*t)-phi))
#Geometry for the system
geom=xd.geometry.TimeDomain(0,20)
#Initial Conditions
def boundary(x,on_initial):
    return xd.utils.isclose(x[0],0)and on_initial
ic1=xd.icbc.IC(geom,lambda x:x0,boundary)
def error(inputs,outputs,X):
    return xd.grad.jacobian(outputs,inputs,i=0)-v0
ic2=xd.icbc.OperatorBC(geom,error, boundary)
#Generating Training and Testing Data
data=xd.data.TimePDE(geom,ode,[ic1,ic2],3000,2,solution=soln,num_test=5000)
#Hyper Parameters
layer=[1]+3*[50]+[1]
activation="tanh"
initializer="Glorot normal"
#Defining Neural Network
net=xd.nn.FNN(layer,activation,initializer)
#Compiling model,and Training
model=xd.Model(data,net)
model.compile("adam",lr=0.001,metrics=["l2 relative error"])
#Plotting Results
losshistory,train_state=model.train(iterations=60000)
xd.saveplot(losshistory, train_state)
T=np.linspace(0,20,1001).reshape(-1,1)
y_pred=model.predict(T)
plt.plot(T,y_pred,"--",label="Model")
plt.plot(T,soln(T),"-",label="True")
plt.legend()
plt.xlabel("Time")
plt.ylabel("Displaacement")
plt.title("Single Degree of Freedom system")
plt.show()
```

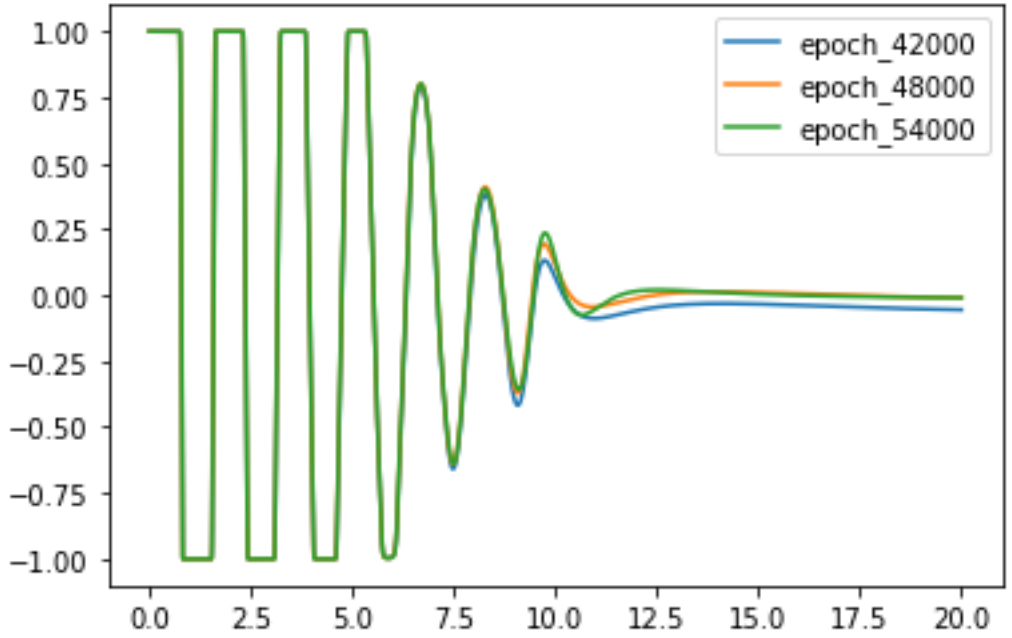
Ex: Single degree of freedom(SDOF) systems NN(Code:Tensorflow)

```python
import tensorflow as tf
import numpy as np
import keras
import matplotlib.pyplot as plt
import math
x=tf.linspace(0.0,20,3001)
#Physical Cnstants
w_n=4
zeta=0.1
#Initial Conditions
x0=0
v0=50
#Analytical Solution
w_d=np.sqrt(1-zeta**2)*w_n
C=math.sqrt(x0**2+((zeta*w_n*x0+v0)/w_d)**2)
if w_d*x0==0:
    phi=math.pi/2
else:
    phi=math.atan(((zeta*w_n*x0)+v0)/w_d/x0)
def soln(t):
    exponent=np.exp(-zeta*w_n*t)
    return C*exponent*(np.cos((w_d*t)-phi))

y=soln(x)
#NOISE=tf.random.normal([101],stddev=0.2)
y1=y#+NOISE
plt.plot(x,y1,".",label="data")
class NNModel(tf.keras.Model):
    def __init__(self,name=None):
        super().__init__(name=name)
        self.layer0=tf.keras.layers.Flatten()
        self.layer1=tf.keras.layers.Dense(1,input_shape=(5001,),activation="tanh")
        self.layer2=tf.keras.layers.Dense(50,activation="tanh")
        self.layer3=tf.keras.layers.Dense(50,activation="tanh")
        self.layer4=tf.keras.layers.Dense(50,activation="tanh")
        self.layer5=tf.keras.layers.Dense(1,activation="tanh")

    def __call__(self,x):
        x=self.layer0(x)
        x=self.layer1(x)
        x=self.layer2(x)
        x=self.layer3(x)
        x=self.layer4(x)
        return self.layer5(x)
```

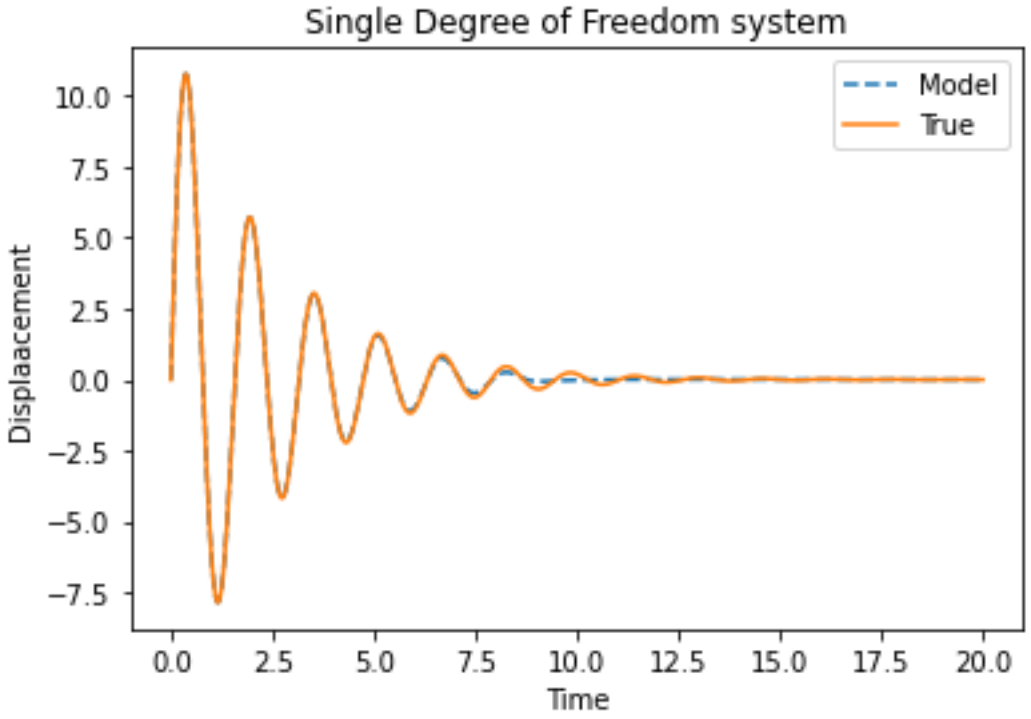
```python
model=NNModel()
loss_fn=keras.losses.MeanSquaredError()
optimizer=keras.optimizers.Adam(learning_rate=0.001)
train_loss=tf.keras.metrics.Mean(name="train_loss")
train_accuracy=tf.keras.metrics.Accuracy(name="train_accuracy")
predictions={}
losses={}
def train(x,y):
    with tf.GradientTape() as tape:
        prediction=model(x)
        predictions[epoch]=prediction
        loss=loss_fn(y,prediction)

    gradient=tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradient,model.trainable_variables))
    losses[epoch]=train_loss(loss).numpy()

    if epoch%6000==0:
        print("TRAIN loss:",train_loss(loss))
        print("TRAIN accuracy:",train_accuracy(y,prediction))
        plt.plot(x,prediction,"-",label=f"epoch_{epoch}")
        if epoch%18000==0:
            plt.legend()
            plt.show()
epochs=60000
for epoch in range(epochs):
    global brk_val
    brk_val=epoch
    if epoch%5000==0:
        print("epoch$",epoch,"of",epochs)
    train(x,y1)
    if float(losses[epoch])<=0.02:
        brk_val=epoch
        break


plt.plot(x,y1,".",label="data")
plt.plot(x,y,"-",label="TRUE")
plt.plot(x,predictions[brk_val],"-",label="Model")
plt.legend()
plt.show()
```

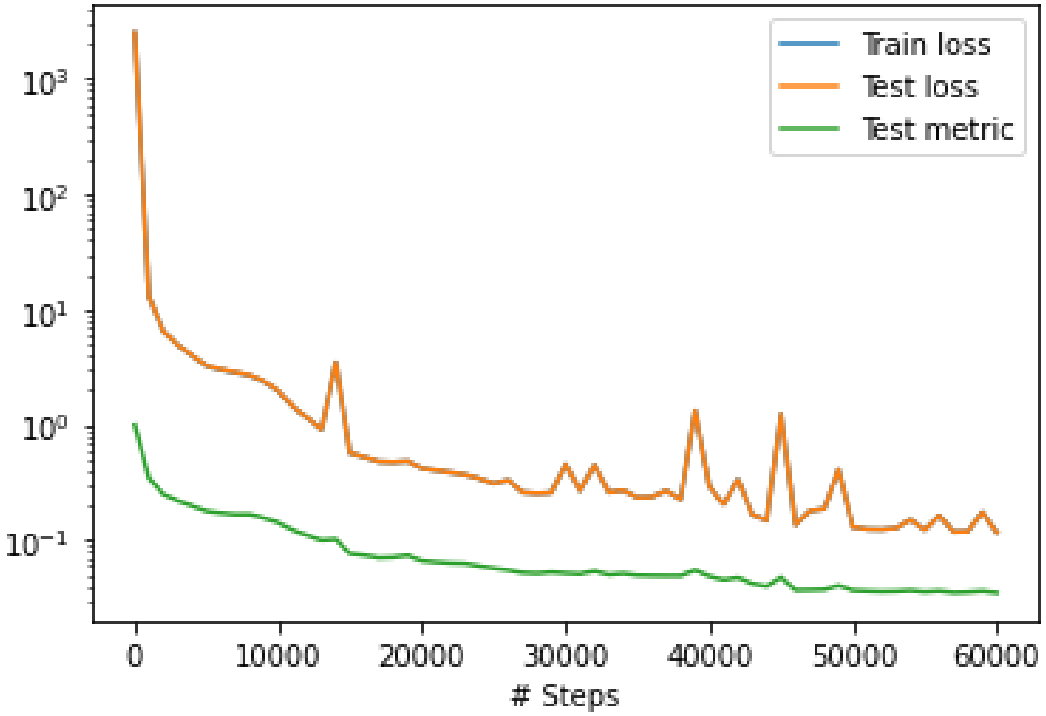
Plots for NN


data
TRUE
Model
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


epoch_42000
epoch_48000
epoch_54000
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Plots for PINN





Piston Vibration(ODE Problem)

Consider oil(incompressible in nature) entering a cylinder through a constriction such that the flow rate is given as $Q = \alpha(p_i - p_0)$ where p_i is the supply pressure and p_0 is the pressure in the cylinder.The cylinder contains a piston of mass m and area A backed by a spring of stiffness k.If $p_i$is of the form $p_i(t) = P_0 + P_1 sin\Omega t$ where P_1, Ω, P_0 are constants We wish to determine $x_p(t)$ which represents the displacement of the piston at time t

Newton's second law yeilds:

— $m\ddot{x} = p_0 A - kx$

Incompressibility of oil forces the condition:

— $p_0 = p_i - \frac{A}{\alpha}\frac{dx}{dt}$

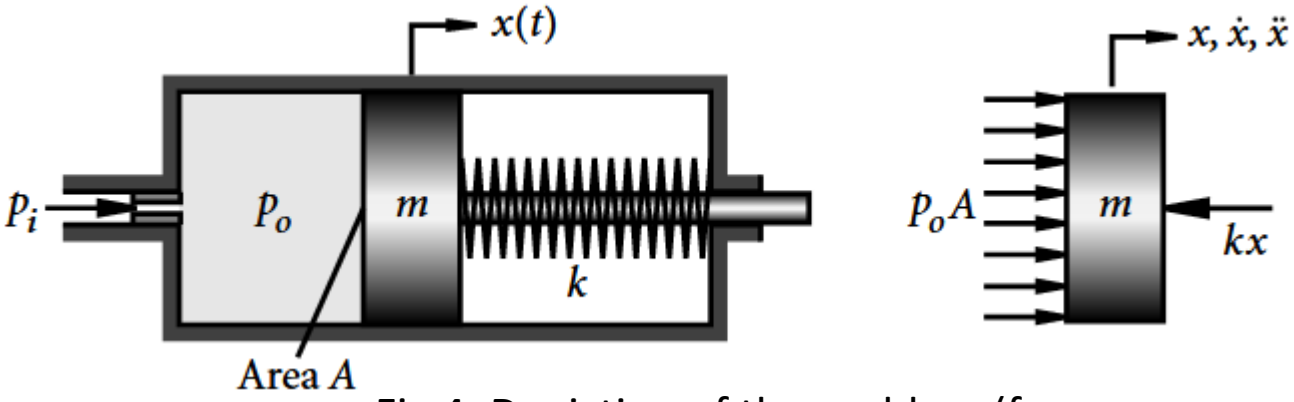


Fig 4: Depiction of the problem (from [6])

Piston Vibration(ODE Problem)(Contd.)

The Governing differential equation can be then written using the conditions as:

$$\ddot{x} + \frac{A}{\alpha m}\dot{x} + \frac{k}{m}x = \frac{A}{m}p_i \tag{1}$$

Substituting p_i and rearranging,We get:

$$\ddot{x} + c\dot{x} + \omega_0^2 x = \frac{A}{m}(P_0 + P_1 sin\Omega t) \tag{2}$$

where $c = \frac{A^2}{\alpha m}$ and $\omega_0^2 = \frac{k}{m}$ The solution of the system can then be written as :

$$x_p(t) = \frac{AP_0}{m\omega_0^2} + \frac{AP_0}{m}\frac{sin(\Omega t - \phi)}{\sqrt{(\omega_0^2 - \Omega^2)^2 + c^2\Omega^2}} \tag{3}$$

where $\phi = tan^{-1}\frac{c\Omega}{\omega_0^2 - \Omega^2}$

Transverse Vibration Problem(PDE Problem)

Consider the cable with constant tension F as shown in the figure.The transverse vibration of the wire $\nu(x,t)$ is given as

$$\frac{\delta^2\nu}{\delta t^2} = \frac{F}{m}\frac{\delta^2\nu}{\delta x^2} \tag{1}$$

We consider the problem with initial conditions as :

$$\nu(x,p) = asin\frac{\pi x}{L}$$
$$\frac{\delta\nu(x,t)}{\delta t}|_{t=0} = 0$$

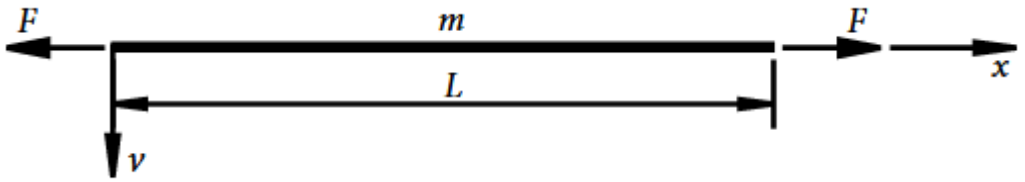


Fig 5: Depiction of problem [6]

The solution of the system is given as:

$$\nu(x,t) = asin(\frac{\pi x}{L})cos(\frac{\pi}{L})\sqrt{\frac{F}{m}}t \tag{2}$$

References Used:

- [1] Yang, Liu, Xuhui Meng, and George Em Karniadakis. "B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data." *Journal of Computational Physics* 425 (2021)
- [2] Jospin, Laurent Valentin, et al. "Hands-on Bayesian neural networks—A tutorial for deep learning users." *IEEE Computational Intelligence Magazine* 17.2 (2022): 29-48.
- [3] Stock, Simon, et al. "Bayesian physics-informed neural networks for robust system identification of power systems." 2023 IEEE Belgrade PowerTech. IEEE, 2023
- [4] Fausett, Laurene V. *Fundamentals of neural networks: architectures, algorithms and applications*. Pearson Education India, 2006.
- [5]https://www.brown.edu/Departments/Engineering/Courses/En4/Notes/Vibrations/Vibrations.htm#Sect52
- [6] Xie, Wei-Chau. *Differential equations for engineers*. cambridge university press, 2010
- [7] Meirovitch, Leonard. *Fundamentals of vibrations*. Waveland Press, 2010.
- Codes:
  - (i) Neural Network code https://github.com/adi0ga/SRFP2024/blob/main/Spring%20mass%20System/Springmass_tf_NN.py
  - (ii) PINN code https://github.com/adi0ga/SRFP2024/blob/main/Spring%20mass%20System/Springmass_alternate.py