Introduction to Physics Informed Neural Networks

Application in Solving Differential Equations

Agney K Rajeev

2111010

References:

[1] M. Dagrada, "Introduction to physics-informed neural networks," *Towards Data Science*, 2022.

[2] B. Moseley, "So, what is a physics-informed neural network?" 2021.

References

The following Python modules were used in this project

- PyTorch : Machine Learning Framework
- Numpy : Scientific Computing
- Matplotlib : Data Visualization
- PIL : Animations

All code used in this project can be found in the following GitHub link : https://github.com/AKR211/PINN

1. Neural Networks
2. PINN
3. Solved Examples
4. Pros & Cons
5. Questions

1 Neural Networks

2 PINN

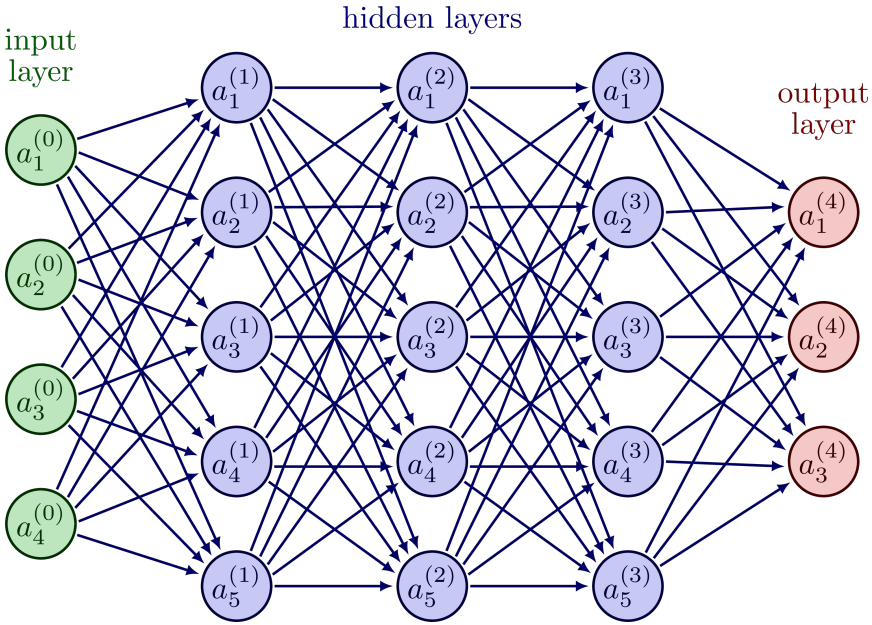
3 Solved Examples

4 Pros & Cons

5 Questions

Introduction

- **Neural Network** : A neural network is a collection of layered nodes or **neurons** which hold some value. Every neuron of a layer is connected to each neuron of the next layer by connections called edges or **weights**



Universal Function Approximator

- Value of the ith neuron in the (j+1)th layer is given as $a_i^{(j+1)} = R((\sum_k a_k^{(j)} w_{ki}^{(j)}) + b_i^{(j)})$
- Value of the (j+1)th layer is given as $\mathbf{a}^{(\mathbf{j+1})} = R(\mathbf{a}^{(\mathbf{j})}.\mathbf{W}^{(\mathbf{j})} + \mathbf{b}^{(\mathbf{j})})$
- **W**:Weight matrix **b**:bias vector R:non-linear activation function
- Neural Networks are **Universal Function Approximators**
- Any function can be approximated up to a tolerance within a finite range by a neural network with a finite amount of layers and neurons

Loss Function

- **Loss function** : The loss function is a function which quantifies the difference between the approximated function and the true function
- In this project, we use the **Mean Squared Error** loss function which is defined as
  $L = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$
- f is the approximated function and the points (x_i, y_i) are called **sample points** which belong to the solution function
- The process of training the neural network is **minimizing** the loss function using **optimizers** like Gradient Descent with respect to the weights and biases.

Code

```
class NeuralNet(nn.Module):
        def __init__(self,numinputs=1,numlayers=3,numoutputs=1,numneurons=8):
                super().__init__()
                self.ni = numinputs
                self.no = numoutputs
                self.nl = numlayers
                self.nneu = numneurons

                layers=[]
                layers.append(nn.Linear(self.ni,self.nneu))

                for _ in range(self.nl):
                        layers.append(nn.Linear(self.nneu,self.nneu))   #create each layer
                        layers.append(nn.Tanh())  #tanh activation function is used

                layers.append(nn.Linear(self.nneu,self.no))

                self.Network = nn.Sequential(*layers)    #chain together layers

        def forward(self,input):
                return self.Network(input.view(-1,1))    #forward propogator
```

Figure 1: Neural Network

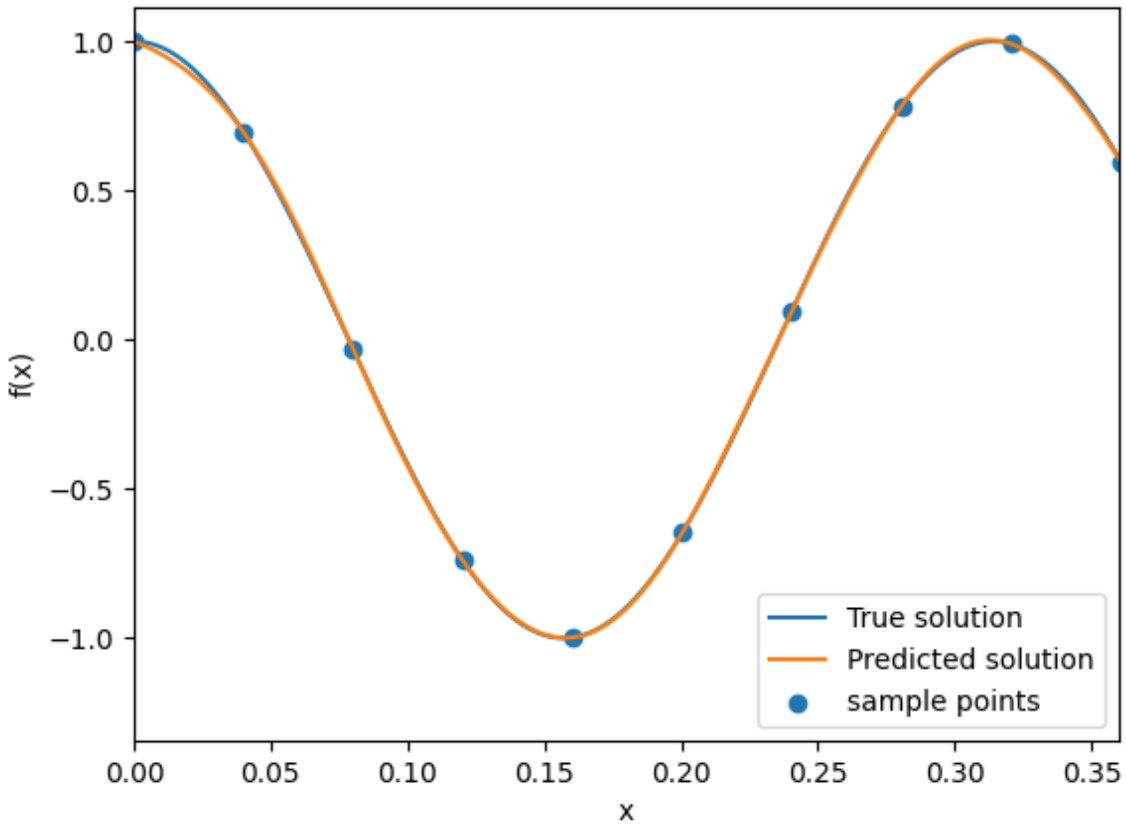
Code

```
def loss_func(x):

        f_value = f(x)

        x0 = x_data
        f0 = y_data
        Bdryloss = f(x0) - f0 #loss due to the sample points

        loss = nn.MSELoss()
        loss_val = loss(Bdryloss,zeros_like(Bdryloss))

        return loss_val
```

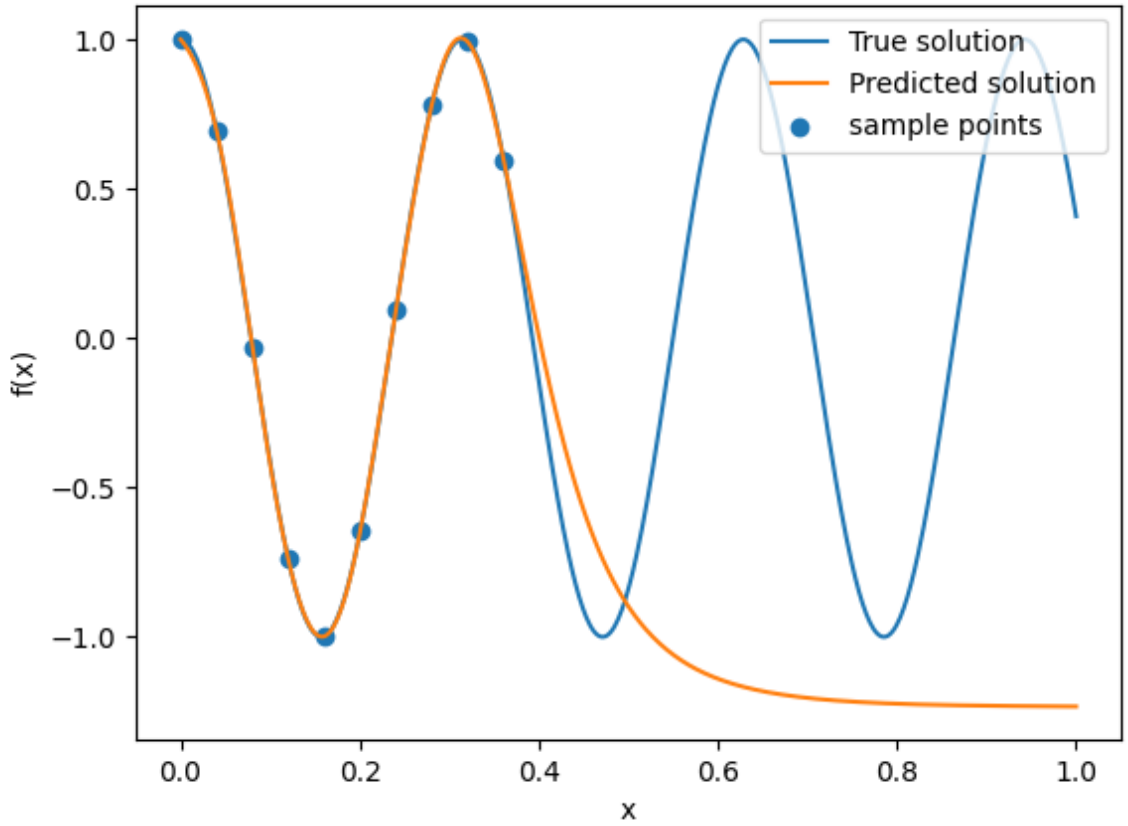
Figure 2: Loss function

Differential Equations

- Can neural networks be used for solving differential equations?



Differential Equations



- Sample points must be distributed across the entire range
- Large number of sample points are required

1 Neural Networks

2 PINN

3 Solved Examples

4 Pros & Cons

5 Questions

Introduction

- **Physics-Informed Neural Networks (PINNs)** are a type of Neural Networks that can embed the knowledge of any physical laws that govern a given data-set in the learning process
- They can be described by Differential Equations of the form $(Df)(x) = 0$ where D is a **differential operator**
- We incorporate the physics information in the Loss function

Loss Function Revisited

- Previously we described our Loss function as $L = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$
- Here we add an additional **differential loss** term $L = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2 + k\frac{1}{m}\sum_{j=1}^{m}((Df)(x_j) - 0)^2$
- The points x_j are called collocation points and can be sampled throughout the domain since knowledge of y_j is not required. k is the weight of the differential loss
- This incentivises the Neural network to adhere to the differential equation while training

Code

```
#the derivative calculator for nth degree
def dnfdxn(n,model,x_values):
        out = model(x_values)
        for i in range(n):
                out = grad(out, x_values, ones_like(out), create_graph=True)[0]
        return out.view(-1,1)
```

```
def loss_func(x):

        f_value = f(x)
        DEloss = dnfdxn(1,f,x) - R*f_value*(1-f_value)   #loss due to differetnial equation constraint

        x0 = x_data
        f0 = y_data
        Bdryloss = f(x0) - f0 #loss due to the sample points

        loss = nn.MSELoss()
        loss_val = (1e-2)*loss(DEloss,zeros_like(DEloss)) + loss(Bdryloss,zeros_like(Bdryloss))
        return loss_val

return loss_func
```

Figure 3: New loss function

1 Neural Networks

2 PINN

3 Solved Examples

4 Pros & Cons

5 Questions

Example 1

Logistic Differential Equation

$$\frac{df}{dx} - Rf(x)(1 - f(x)) = 0$$

Describes the statistical distribution of fermions over the energy states of a system in thermal equilibrium and phase diagrams in diffusion bonding

$$f(x) = \frac{P_0 e^{Rx}}{1 + P_0(e^{Rx} - 1)}$$

Applying the following boundary conditions : $f(0) = 0.5$

$$f(x) = \frac{e^{Rx}}{1 + e^{Rx}}$$

Code

```
#define all specifications/hyperparameters
R = 1.0
x_data = tensor([0.0]) #sample points (x)
y_data = tensor([0.5]) #sample points (y)
inputs=1   # number of inputs of the fucntion
outputs=1  #number of outputs of the function
layers=1   # number of hidden layers in the NN
neurons=16 # number of edges per neuron
learning_rate=1e-4  #Step size of optimization
num_epochs=10000  #Number of training steps
batch_size=30  #Number of collocation points
domain=(-5.0,5.0) #Domain of approximation
```

Figure 4: Specifications for the PINN

Example 1

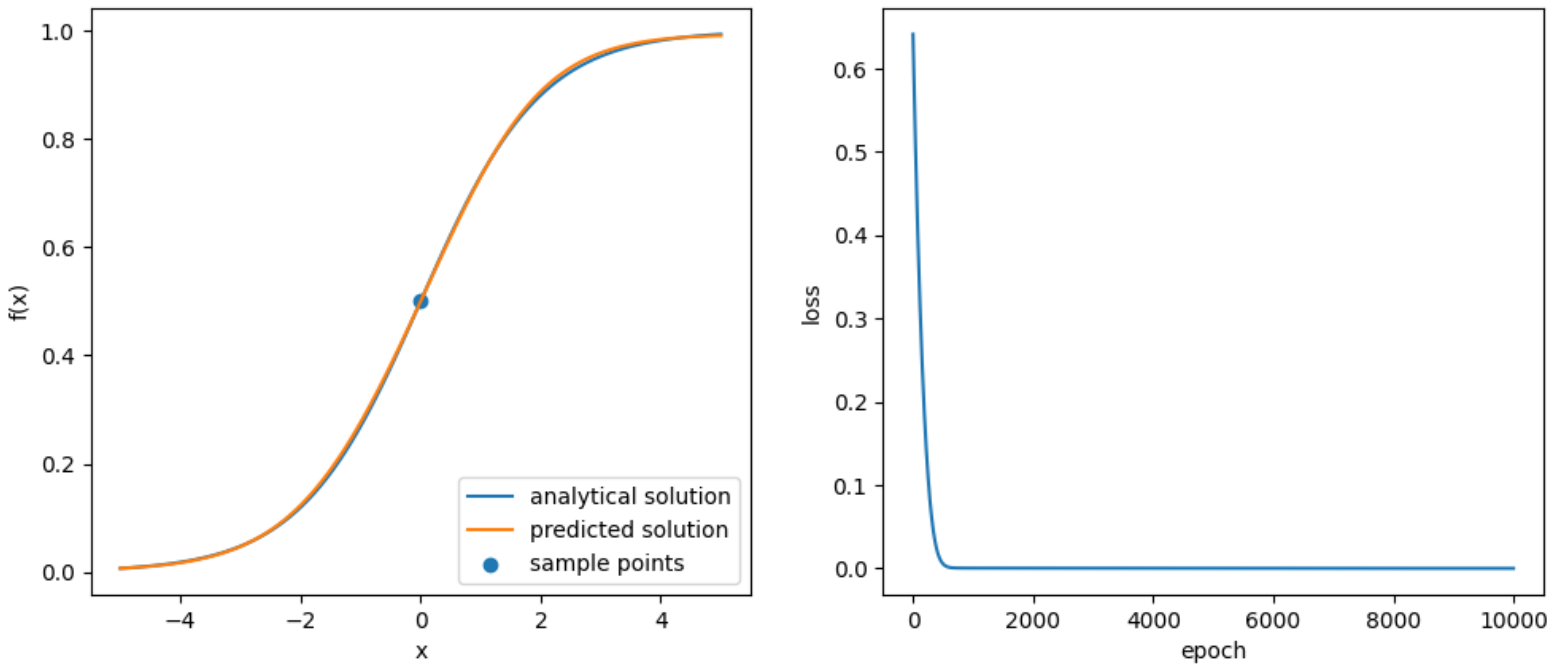
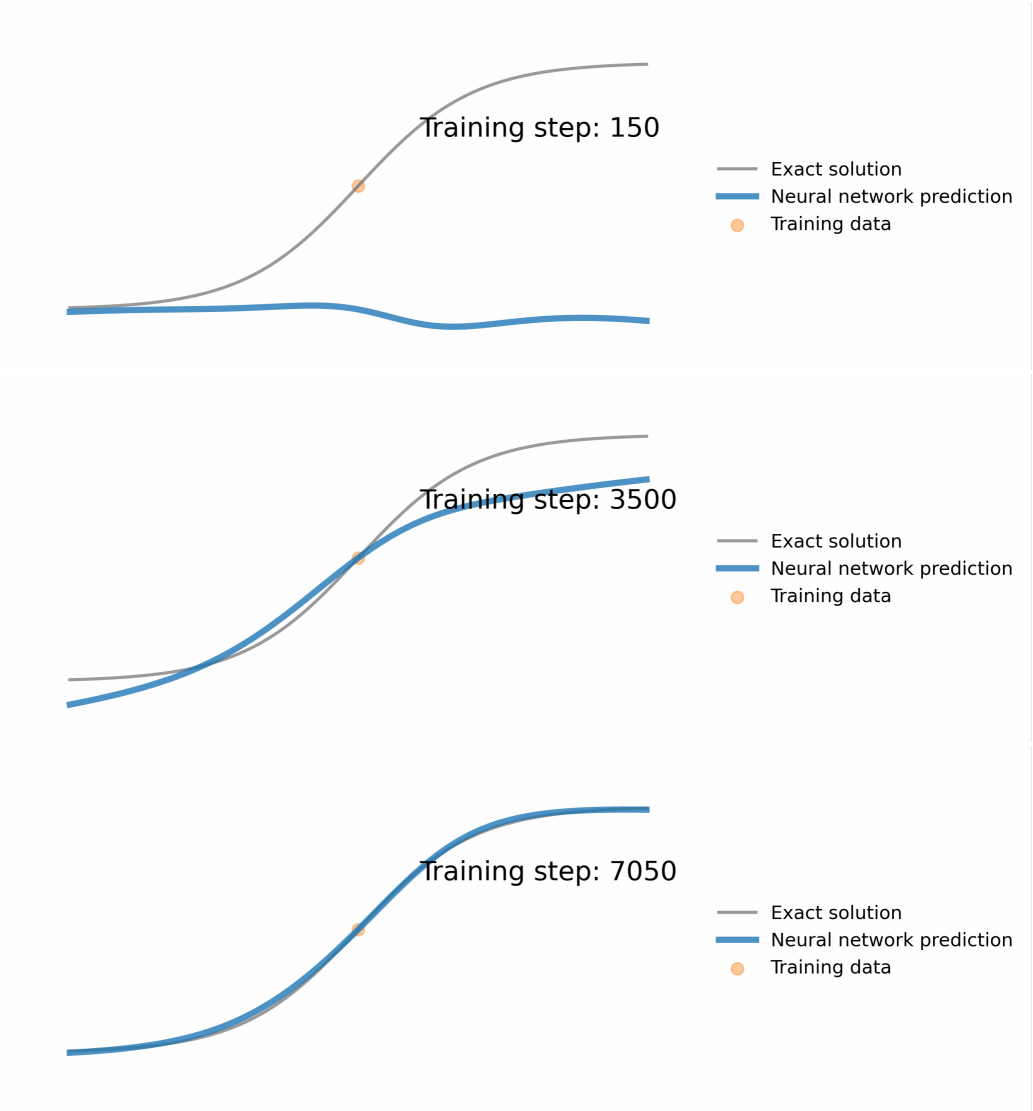


Figure 5: Plots of the Solution Function (left) and Loss with training step (right) Final loss : 3×10^{-7}

Training Montage



Example 2

Damped Harmonic Oscillator Equation

$$\frac{d^2f}{dx^2} + 2\gamma\frac{df}{dx} + \omega^2 f(x) = 0$$

Used to approximate oscillatory behaviours for small perturbation

$$f(x) = Ae^{-\gamma x} sin\left[\left(\sqrt{\omega^2 - \gamma^2}\right)x + \phi\right]$$

Applying appropriate boundary conditions we make $A = 1,\ \phi = 0$

$$f(x) = e^{-\gamma x} sin\left[\left(\sqrt{\omega^2 - \gamma^2}\right)x\right]$$

Code

```
#define all specifications/hyperparameters
Omega = 20.0
Gamma = 2.0
x_data = tensor([0.,0.1,0.2]).view(-1,1) #sample points (x)
y_data = tensor([0.,0.7478,-0.4984]).view(-1,1) #sample points (y)
inputs=1   # number of inputs of the fucntion
outputs=1  #number of outputs of the function
layers=1   # number of hidden layers in the NN
neurons=16 # number of edges per neuron
learning_rate=1e-4  #Step size of optimization
num_epochs=30000  #Number of training steps
batch_size=30  #Number of collocation points
domain=(0.,1.0) #Domain of approximation
```

Figure 7: Specifications for the PINN

Example 2

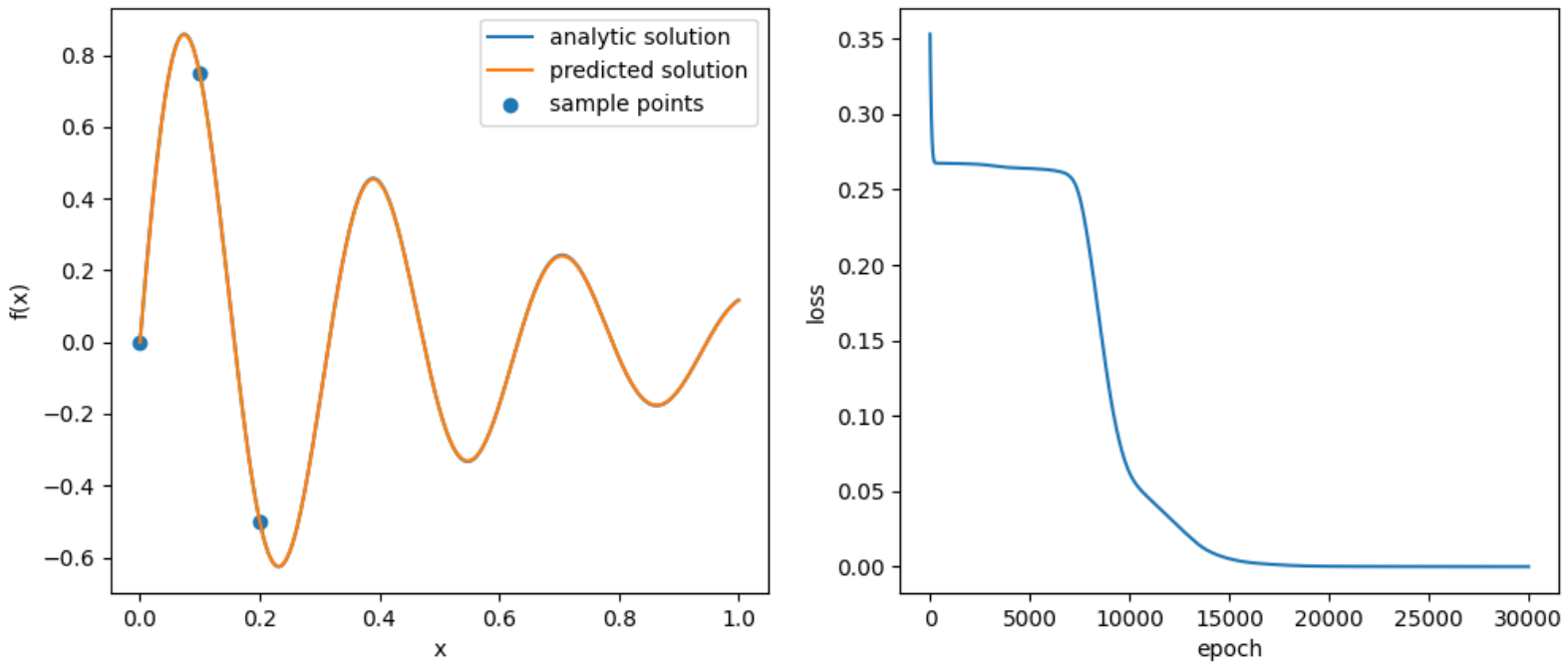
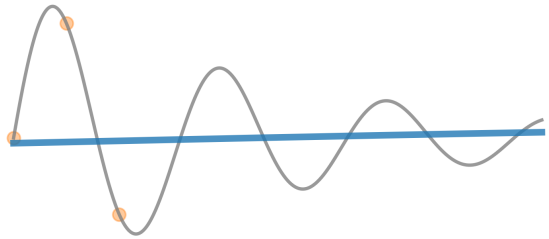


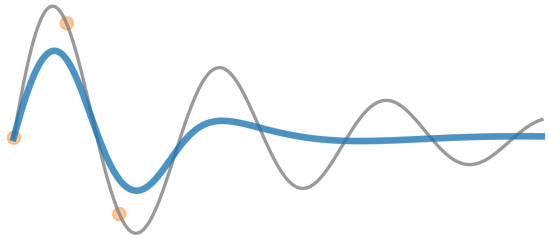
Figure 8: Plots of the Solution Function (left) and Loss with training step (right). Final Loss : 8.44×10^{-5}

Training Montage



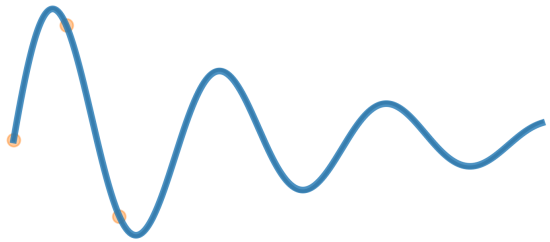
Training step: 100

Exact solution
Neural network prediction
Training data



Training step: 10000

Exact solution
Neural network prediction
Training data



Training step: 20000

Exact solution
Neural network prediction
Training data

Example 3

Wave Equation

$$\frac{\partial^2 f}{\partial t^2} - c^2 \frac{\partial^2 f}{\partial x^2} = 0$$

Describes mechanical and electromagnetic waves as well as standing wave fields

$$f(x,t) = F(x - ct) + G(x + ct)$$

Boundary conditions: $f(x,0) = sin[20x]$, $f(x,1) = sin[20(x-1)]$, $f(0,t) = sin[-20t]$, $f(0,t) = sin[20(1-t)]$

$$f(x,t) = sin\left[20(x - ct)\right]$$

Code

```
#define all specifications/hyperparameters
c = 1.0
truef = lambda x,t: np.sin(20.0*x - 20.0*t) #analytic solution
n=30 #number of collocation points
x_data = hstack((linspace(0,1,n),linspace(0,1,n),zeros(n),ones(n))).view(-1,1) #sample points (x)
t_data = hstack((zeros(n),ones(n),linspace(0,1,n),linspace(0,1,n))).view(-1,1) #sample points (t)
y_data = truef(x_data,t_data).view(-1,1) #sample points (y)
inputs=2 #number of inputs
outputs=1 #number of outputs
layers=2 #number of hidden layers
neurons=32 #number of edges per neuron
learning_rate=1e-4 #step size of optimization
num_epochs=30000 #number of training steps
domainx=(0.,1.0) #domain of x
domaint=(0.,1.0) #domain of t
```

Figure 10: Specifications for the PINN

Example 3

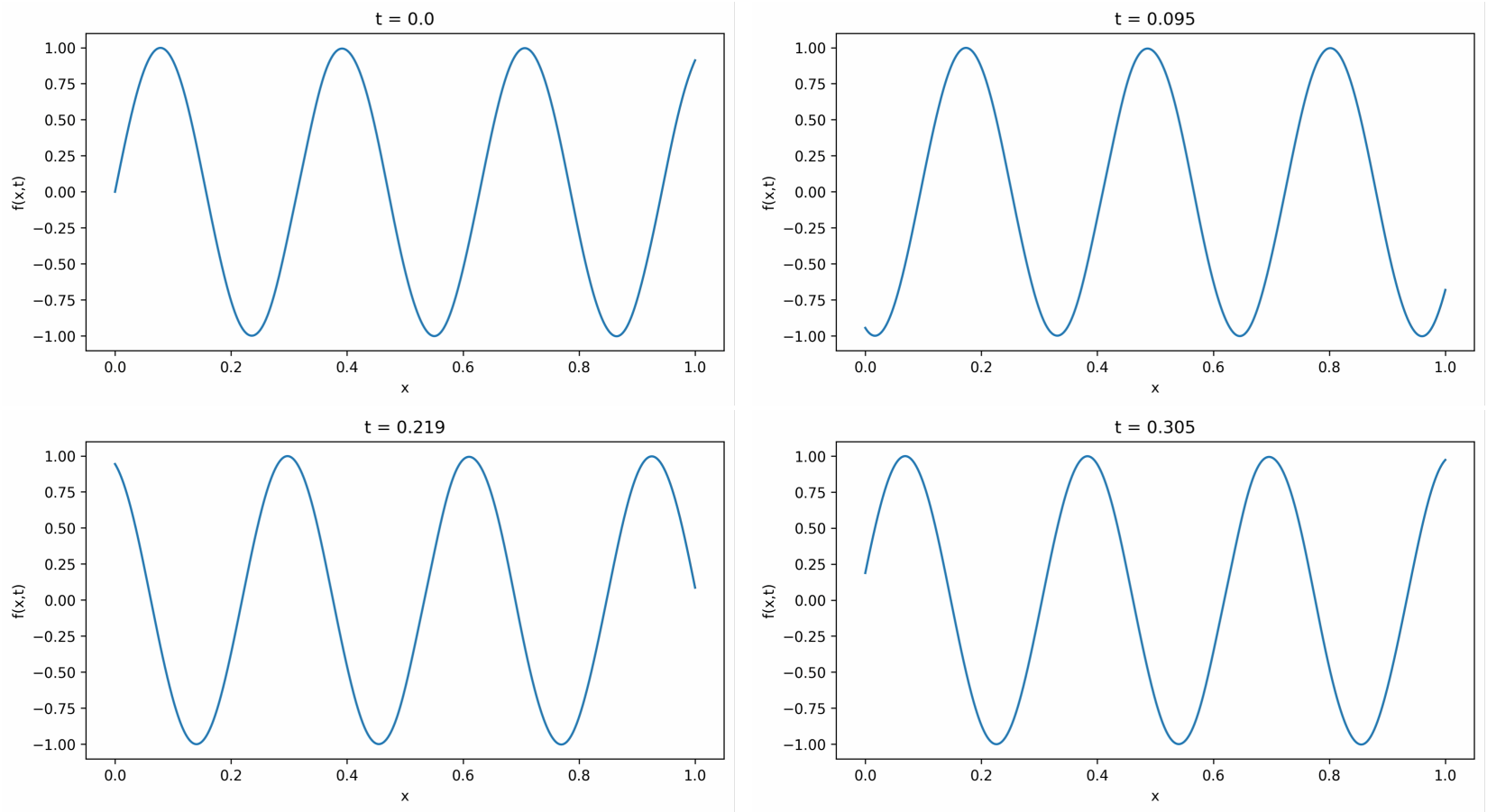


Figure 11: Solution Function (x,f(x,t)) for different t

Example 3

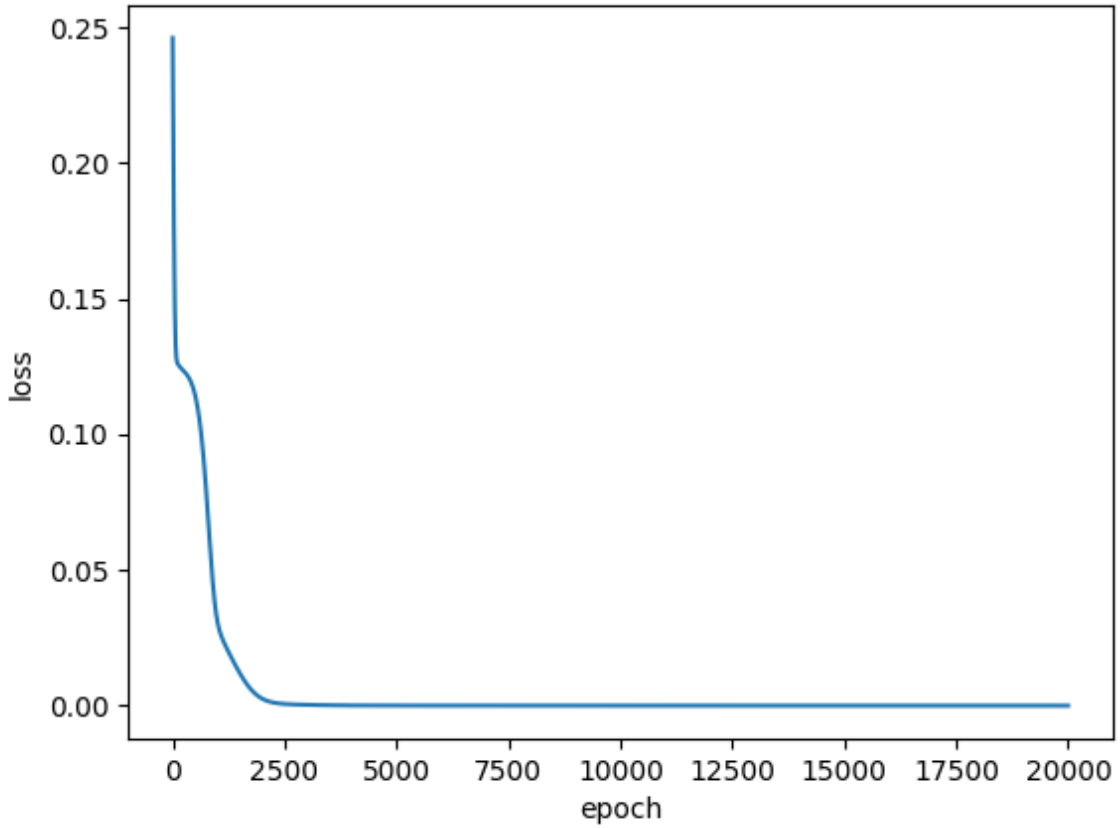


Figure 12: Loss plot for the wave equation. Final loss : 1.2×10^{-6}

1 Neural Networks

2 PINN

3 Solved Examples

4 Pros & Cons

5 Questions

Advantages

- **PINNs are self-learning**
  There is no need to derive a systematic solution method. PINNs can converge to a solution by itself with appropriate supervision
- **PINNs are data-driven**
  The performance of a PINN relies majorly on the training data and not the complexity of the underlying dynamics. It can also improve upon itself with the addition of new data
- **PINNs are independent of the intermediate training steps**
  Only the final output needs to be an accurate approximation. Errors in intermediate steps do not affect the final output.
- **PINNs are not discretized**
  A PINN is a continous function. It can provide the function value for any input without interpolation.

Disadvantages

- **PINNs are self-learning**
  PINNs are a black-boxes. We do not have a solid proof as to whether the PINN is trained to be the correct function
- **PINNs are data-driven**
  PINNs can perform poorly if there is a lack of sample points. Also the computation cost for using a large number of points can be quite high
- **PINNs are independent of the intermediate training steps**
  We cannot access a partial solution. Extending the domain for approximation would require retraining the whole model
- **PINNs are not discretized**
  The computation cost or accuracy cannot be adjusted with the step size

WARNING : This Image is AI Generated

1 Neural Networks

2 PINN

3 Solved Examples

4 Pros & Cons

5 Questions

Question 1

What is the adjacency matrix of a Neural network?

The adjacency matrix of a neural network is not usually explicitly defined as there are a large number of nodes in the network (most of which are not connected to each other.)

E.g. If a network with two hidden layers of 32 neurons each would require a 64x64 matrix out of which 75% are just zeros. It is better to just represent the network with the size of the 25% non-zero weights and specifying between which two layers they belong.

Question 2

Try and solve the Schrodinger equation for quantum tunneling

The Time Independent Schrodinger equation can be written as

$$\frac{d^2\psi}{dx^2} + \frac{2m(E - V(x))}{\hbar^2}\psi = 0$$

For quantum tunneling to occur, we require a potential barrier such that

$$V(x) = \begin{cases} 0 & x \leq a \\ V_0 & a \leq x \leq b \\ 0 & b \leq x \end{cases}$$

where $V_0 > E$

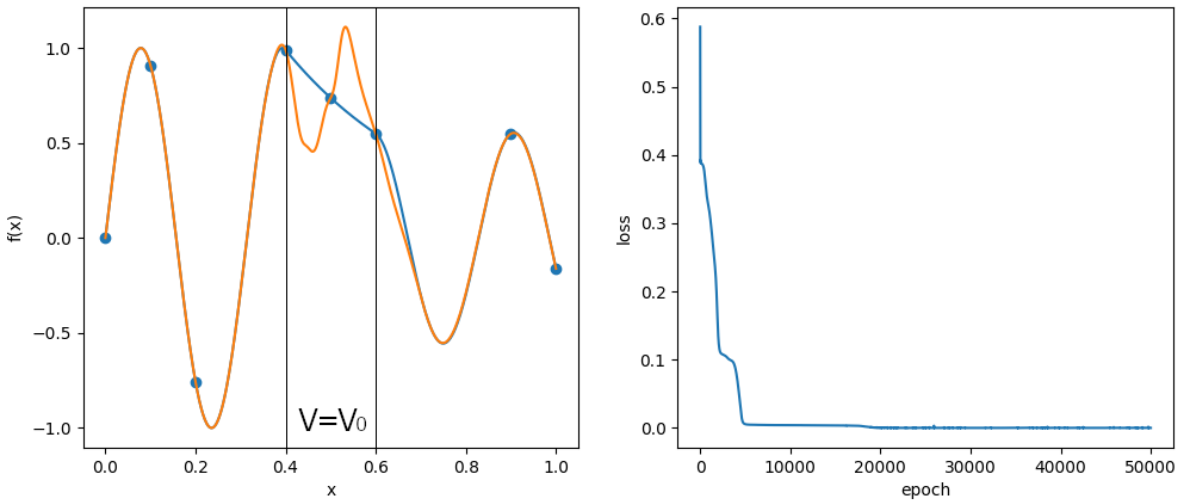


Figure 13: Plot for quantum tunneling and the loss evolution

We can observe that the approximation behaves nicely before and after crossing the barrier with the amplitude of the wave decreasing (but still being non-zero) after crossing the barrier. However, inside the barrier, an exponential decay was expected and not an oscillatory solution.

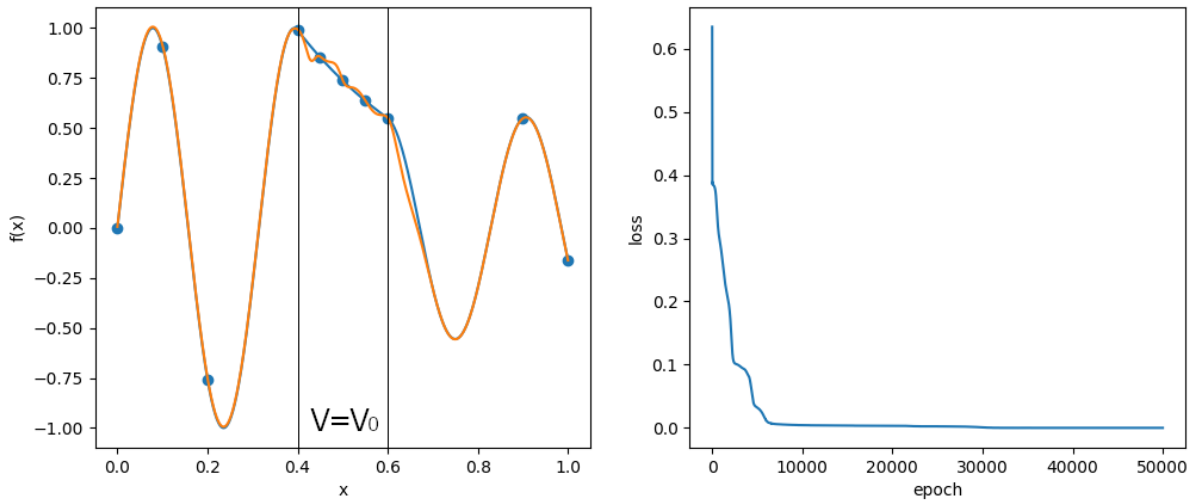


Figure 14: Plot for quantum tunneling (more points) and the loss evolution

This can be overcome to an extent by increasing the number of sample points in the region of the barrier but the oscillatory behaviour is still present and feasibility of obtaining so many sample points is questionable

This likely occurs because of the piece-wise definition of the potential (and hence the differential equation and hence the solution function). the oscillatory behaviour which the Neural Network learnt from outside the barrier carries over to its behaviour inside the barrier. While a Neural Network is theoretically capable of approximating a piece-wise function, it is very difficult to train it do so as we have to train it to behave as different functions at different regions. This can be overcome by training two neural networks separately for both regions with appropriate boundary points to ensure that they are in mutual agreement. However, this would require a larger amount of computational cost and time

```python
def V(x):
        L=[]
        for i in x:
                p = float(i.squeeze())
                if p<0.5:
                        L.append(0.0)
                        continue
                if p<0.7:
                        #L.append(R*(np.cot(np.sqrt(R)*0.4))**2)
                        L.append(400.0 + (2.94130127899**2))
                        continue
                L.append(0.0)
        return tensor(L).view(-1,1)
```

```python
#define all specifications
inputs=1
outputs=1
layers=2
neurons=64
learning_rate=1e-4
num_epochs=50000
batch_size=30
domain=(0.,1.0)
```

Figure 15: Code for the potential function and the specifications (full code can be found in the GitHub page mentioned in slide 2)

Question 3

Why are Physics-Informed Neural Networks named so?
PINNs are DE solvers. They have inherently very little to do with physics in terms of its working principle. They are named so because they combine principles from physics with neural networks to solve scientific and engineering problems. They are designed to incorporate the laws of physics as constraints into the training process of neural networks, making them informed by the underlying physical principles.

The concept of "Domain-Informed Neural Networks" can be executed to a variety of fields where you would like to ensure that neural network solutions fit with domain-specific information or limitations and not just physics. The term "Physics-Informed Neural Networks" is well-known because it was one of the first uses of this technology to earn academic interest. Similar strategies, however, can and have been applied to problems in Biology[1], Chemistry[2], and other fields.

[1] Biologically-informed neural networks guide mechanistic modeling from sparse experimental data doi: 10.1371/journal.pcbi.1008462

[2] Chemistry-Informed Neural Networks modelling of lignocellulosic biomass pyrolysis https://doi.org/10.1016/j.biortech.2022.127275

Question 4

How are PINNs continous functions?

A discrete function is a function in which the domain and range are each a discrete set of values. A non-discrete (continuous) function is one that is continuous either on its entire domain, or on intervals within its domain. A discrete structure can produce a continuous output if the inputs are continuous and the operations executed on the input ensure that the output is continuous as well. E.g. A matrix is a discrete structure. Consider the following function composed of a matrix multiplication

$$f(x_1, x_2) = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{bmatrix}$$

if $x_1, x_2 \in R$ then $f(x_1, x_2) \in R^2$. Both domain and range are continuous and hence the function is continuous

The neural network consists only of operations described in slide 6, all of which (matrix multiplication, matrix addition and application of the activation function) are continuous. Hence, the Neural Network is able to produce a continuous output if provided with a continuous input.