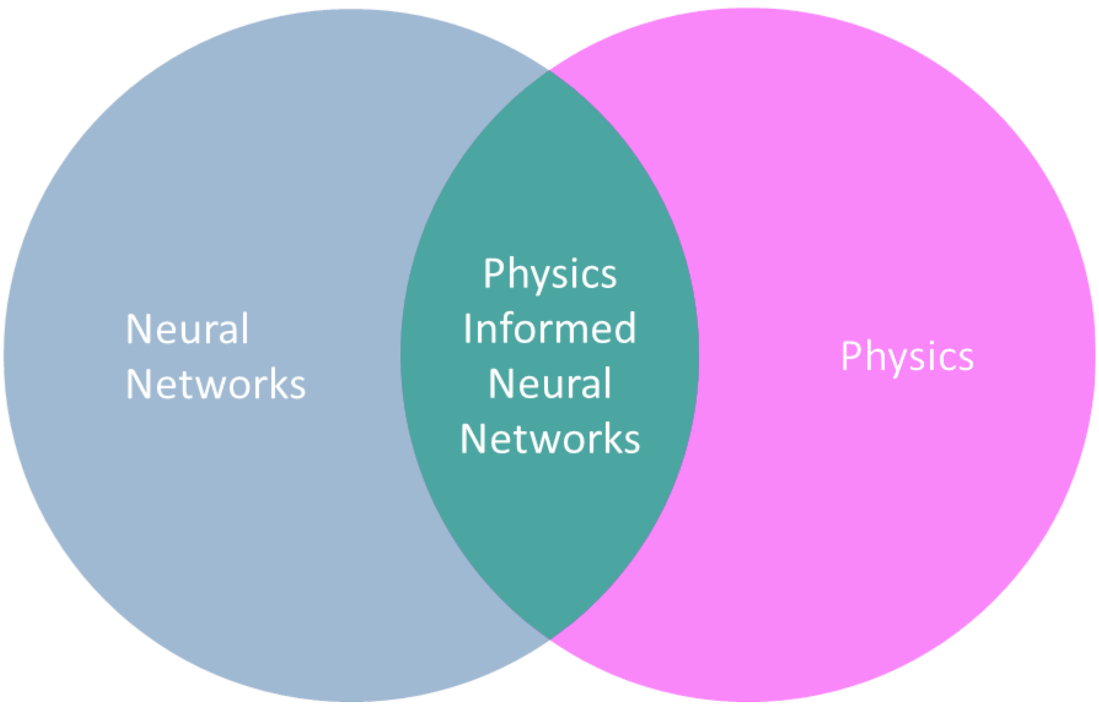
REPORT

SOLVING NON- LINEAR SCHRODINGER EQUATION USING PINN's

DATE OF SUBMISSION - 2 DECEMBER 2024



ALOK MAURYA

22118007
B.TECH 3 YEAR
MMED

1. INTRODUCTION

The nonlinear Schrödinger equation in one dimension is given by:

$$i\hbar\, \partial\Psi/\partial t = -(\hbar^2/2m)\nabla^2\Psi + V(x)\Psi + g|\Psi|^2\Psi$$

Key points about the NLS equation:

1.1 Structure:

The first term ($i\hbar\, \partial\Psi/\partial t$) represents time evolution

The second term ($-(\hbar^2/2m)\nabla^2\Psi$) represents kinetic energy

$V(x)\Psi$ is the potential energy term

$g|\Psi|^2\Psi$ is the nonlinear term that makes this equation different from the linear version

1.2 Applications:

- Bose-Einstein condensates
- Optical fiber communications
- Water waves
- Plasma physics
- Nonlinear optics

1.3 Important Properties:

- Supports soliton solutions (stable wave packets that maintain their shape)
- Can exhibit phenomena like self-focusing and wave collapse
- Has both focusing (attractive) and defocusing (repulsive) variants

2. MATHEMATICAL FORMULATION

The Nonlinear Schrödinger Equation (NLSE) is generally solved in the form:

$$i\frac{\partial \psi}{\partial t} + \frac{1}{2}\frac{\partial^2 \psi}{\partial x^2} + |\psi|^2\psi = 0$$

2.1 Initial Condition:

The initial wave function $\psi(x,t=0)$ is a Gaussian wave packet given by:

$$\psi(x,0) = A \cdot \exp\left(i(kx)\right) \cdot \exp\left(-\frac{(x-x_0)^2}{2\sigma^2}\right)$$

Where: A is the amplitude,k is the wave number,ω is the frequency,v is the velocity,

σ is the width of the packet (a parameter of the soliton),x is the spatial coordinate, and

t is time.

2.2 Boundary Conditions:

The notebook likely implements **periodic boundary conditions**, where:

$$\psi(\text{xmin}, t) = \psi(\text{xmax}, t)$$

2.3 ANALYTICAL SOLUTION

A typical analytical solution in one spatial dimension for the nonlinear Schrödinger equation (in the form of solitons) can be expressed as:

$$\psi(x,t) = A \exp\left[i(kx - \omega t) - \frac{(x-vt)^2}{2\sigma^2}\right]$$

2.4 ASSUMPTIONS

Physical and Mathematical Assumptions

- One-dimensional system: The NLSE is formulated and solved in one spatial dimension (x), ignoring complexities of higher dimensions.

- Localized wave packet: The initial condition assumes a Gaussian wave packet, simplifying real-world waveforms with potential irregularities.
- Neglecting external potentials: No external forces or potential terms (V(x)) are included in the NLSE, which is typically expressed as:

$$i\frac{\partial \psi}{\partial t} + \frac{\partial^2 \psi}{\partial x^2} + |\psi|^2\psi = 0$$

  This simplifies the system to free wave propagation and nonlinear interactions only.

3. PINN DESIGN

3.1 Neural Network Architecture

- Input Layer:
  - Two inputs: spatial coordinate (x) and time (t).
- Hidden Layers:
  - Number of layers: 10 hidden layers.
  - Neurons per layer: 64 neurons per layer.
- Output Layer:
  - Single output: the predicted complex wave function (ψ(x,t)).
- Activation Function:
  - Gaussian Activation Function : ϕ(z)=exp(–z^2 / 2σ^2).
  - Along with sigmoid , tanh , Swish ,ReLU is also used but not done well.
  - Used in hidden layers
  - The Gaussian activation is particularly well-suited for problems involving localized features, such as solitons in the NLSE.

3.2 Training Process

3.2.1 Optimizer

- Optimizer used: **Adam optimizer**
  - Adam is a gradient-based optimization algorithm that adjusts learning rates adaptively for each parameter, making it well-suited for PINN training.

3.2.2 Learning Rate

- Initial learning rate: **0.001**
  - This learning rate is chosen as a balance between convergence speed and stability during training.

3.2.3 Number of Epochs

- The network is trained for **10,000 epochs**.
  - A high number of epochs ensures that the network thoroughly minimizes the loss, as the PINN must balance multiple terms (residuals and boundary conditions).

4. Implementation

4.1 Tools and Libraries Used

The following tools and libraries are utilized in the project for solving the Nonlinear Schrödinger Equation (NLSE) with a Physics-Informed Neural Network (PINN):

- **PyTorch**: Used for constructing and training the neural network.
- **NumPy**: For numerical computations and data handling.
- **matplotlib**: For visualizing the results, including comparisons between analytical and PINN-predicted solutions.
- **Automatic Differentiation**: Leveraged via PyTorch to compute the gradients needed for the loss function, particularly for the residual term of the NLSE.

4.2 Code Description

4.2.1. Data Preparation

```
# Parameters for the initial Gaussian wave packet (soliton solution)
A = 1.0    # Amplitude
k = 2.0    # Wave number
w = 1.0    # Frequency
v = 0.1    # Velocity
sigma = 0.2 # Width of the wave packet

# Space and time coordinates
x = torch.linspace(0, 1, 500).view(-1, 1)  # Spatial grid
t = torch.linspace(0, 1, 500).view(-1, 1)  # Time grid
```

- Purpose: Defines the computational domain (x, t) and generates the training data.
- Key Steps:
  - Create a grid for spatial (x) and temporal (t) coordinates.
  - Specify initial and boundary conditions (Gaussian wave packet and periodic boundaries).

4.2.2 Neural Network Definition

```
class PINN_Schrodinger(nn.Module):
    def __init__(self, N_INPUT, N_OUTPUT, N_HIDDEN, N_LAYERS, activation=nn.ReLU()):
        super().__init__()
        self.activation = activation  # Set the activation function

        self.fcs = nn.Sequential(
            nn.Linear(N_INPUT, N_HIDDEN),
            self.activation
        )

        self.fch = nn.Sequential(*[
            nn.Sequential(
                nn.Linear(N_HIDDEN, N_HIDDEN),
                self.activation
            ) for _ in range(N_LAYERS - 1)
        ])

        self.fce = nn.Linear(N_HIDDEN, N_OUTPUT)

    def forward(self, x, t):
        inputs = torch.cat([x, t], dim=1)
        outputs = self.fcs(inputs)
        outputs = self.fch(outputs)
        psi = self.fce(outputs)
        return psi
```

- Purpose: Constructs the PINN architecture.
- Key Steps:
    - Define a feedforward neural network with ReLU activation functions.
    - Use two inputs (x and t) and a single complex output (ψ).
    - Include automatic differentiation to compute derivatives of ψ for enforcing the NLSE.

4.2.3 Loss Function Definition

```
def loss_fn(model, x_data, t_data, psi_data, x_physics, t_physics ,launda ):
    """Calculate the combined loss: data loss + physics loss for NLSE"""

    # Data loss: Mean squared error between network prediction and known solution
    psi_pred = model(x_data, t_data)
    data_loss = torch.mean(torch.abs(psi_pred - psi_data)**2)

    # Physics loss: Ensure that the network satisfies the Nonlinear Schrödinger equation
    psi_physics = model(x_physics, t_physics)

    # Compute the partial derivatives of psi with respect to x and t
    psi_x = torch.autograd.grad(psi_physics, x_physics, torch.ones_like(psi_physics), create_graph=True)[0]
    psi_xx = torch.autograd.grad(psi_x, x_physics, torch.ones_like(psi_x), create_graph=True)[0]
    psi_t = torch.autograd.grad(psi_physics, t_physics, torch.ones_like(psi_physics), create_graph=True)[0]

    # Compute the residual of the NLSE: i∂ψ/∂t + 1/2 ∂²ψ/∂x² + |ψ|²ψ = 0
    physics_loss = torch.mean(torch.abs(1j * psi_t + 0.5 * psi_xx + torch.abs(psi_physics)**2 * psi_physics)**2)

    # Combine data and physics loss
    total_loss = data_loss + launda*physics_loss
    return total_loss
```

- Purpose: Encodes the physical laws and boundary/initial conditions into the loss.
- Key Steps:
    - Compute the residual of the NLSE for all training points.
    - Enforce initial and boundary conditions by penalizing deviations.
    - Combine residual and boundary/initial condition losses into a total loss.

4.2.4 Training Process

```
def train_model(model, optimizer, scheduler, loss_fn, x_data, t_data, psi_data, x_physics, t_physics, launda,
                x, psi_exact, num_epochs=10000, plot_interval=2000):
    losses = []

    for epoch in range(num_epochs):
        optimizer.zero_grad()
        loss = loss_fn(model, x_data, t_data, psi_data, x_physics, t_physics, launda)
        loss.backward()
        optimizer.step()
        scheduler.step(loss)  # Update the learning rate
        losses.append(loss.item())  # Store loss for plotting
```

```
# Training loop with improved learning rate scheduling and loss visualization
model = PINN_Schrodinger(2, 2, 64, 16 ,activation=nn.ReLU())  # Initialize the model
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4) # Increased initial learning rate
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=500, factor=0.5, verbose=True) # Learning rate scheduler
```

- Purpose: Optimizes the network to minimize the total loss function.
- Key Steps:
    1. Use the Adam optimizer for parameter updates.
    2. Train the network over 10,000 epochs.
    3. Monitor the loss function after every 2000 epochs to ensure convergence.

4.2.5 Evaluation

```
# Training loop with improved learning rate scheduling and loss visualization
model = PINN_Schrodinger(2, 2, 64, 10 ,activation =GaussianActivation())  # Initialize the model
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3) # Increased initial learning rate
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=500, factor=0.5, verbose=True) # Learning rate scheduler

losses = [] # List to store losses

losses = train_model(
    model=model,
    optimizer=optimizer,
    scheduler=scheduler,
    loss_fn=loss_fn,
    x_data=x_data,
    t_data=t_data,
    psi_data=psi_data,
    x_physics=x_physics,
    t_physics=t_physics,
    launda=launda,
    x=x,
    psi_exact=psi_exact,
    num_epochs=10000,
    plot_interval=2000
)
```

- Purpose: Compares the PINN-predicted solution with the analytical solution.
- Key Steps:
    1. Evaluate the trained model on a grid of points.
    2. Compute metrics (e.g., mean squared error) .
    3. Visualize the results using matplotlib.

4.3 Challenges Encountered

4.3.1 Balancing Loss Terms

- Challenge: The loss function combines multiple terms (residual, boundary, and initial conditions), and their relative magnitudes can differ significantly.
- Solution: A weighting factor (λBC) was introduced to balance the contributions of the different terms.

4.3.2 Stability During Training

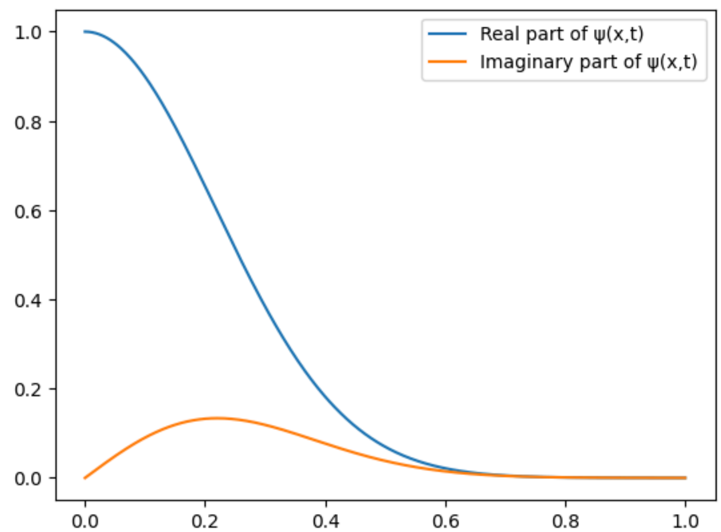
- Challenge: Training the network became unstable at times, especially when using a high learning rate or imbalanced loss weights.
- Solution: Reduced the learning rate to 0.001 and experimented with different weight initializations.

5. RESULTS

5.1 Analytical Solution

The exact solution of the nonlinear Schrodinger equation is calculated by the solution stated above using the data generated for x and t.

The exact solution looks like :



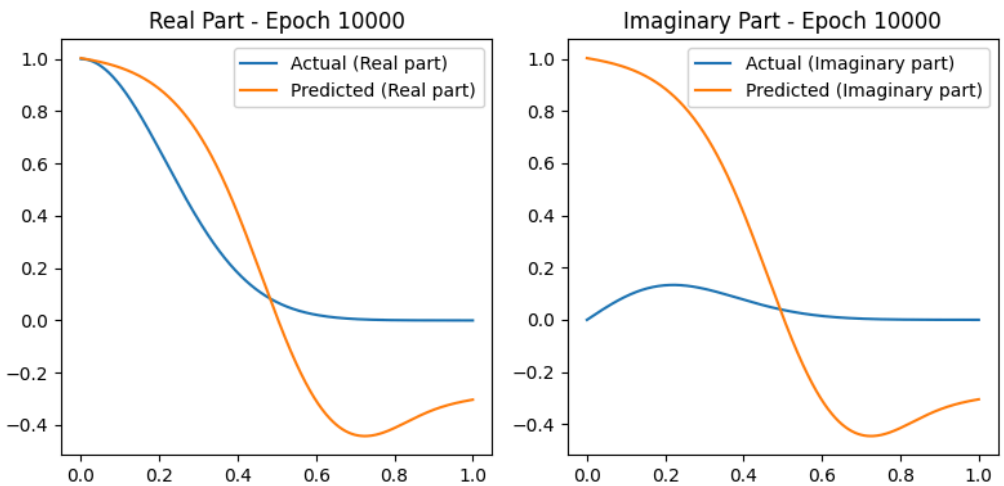
This is the exact solution for the real and imaginary part of the equation .

5.2 PINNS SOLUTION

Initially the results were **worse** with different activation functions like **sigmoid , tanh , Swish .**

The prediction **neither fitted well** on the real part nor on the imaginary part.

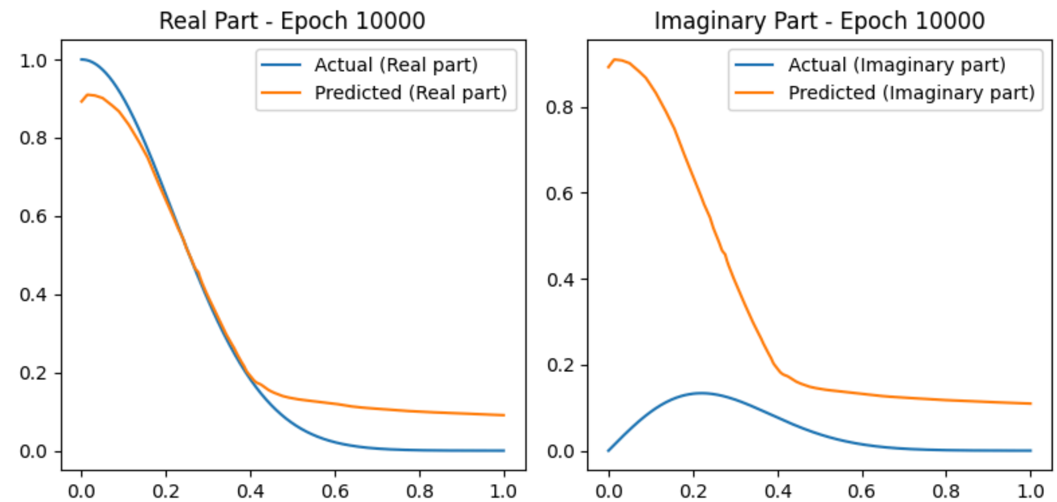
5.2.1 Using sigmoid , tanh and Swish

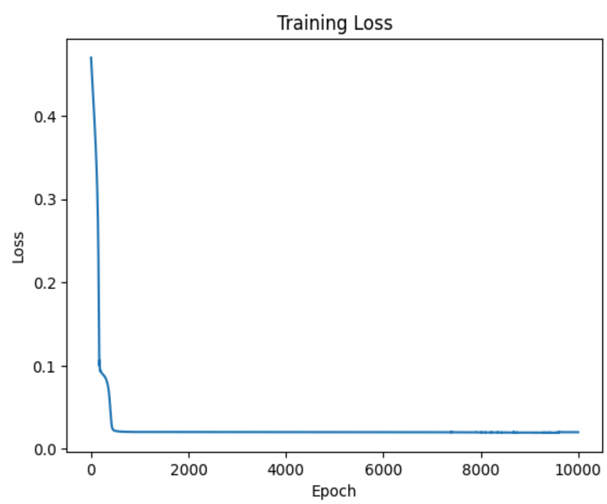


But later **better results** were obtained with **ReLU and Gaussian** activation function after some hyperparameter tuning of learning rate.

The **best learning rate** comes out to be **1e-3** .

5.2.2 Using ReLU

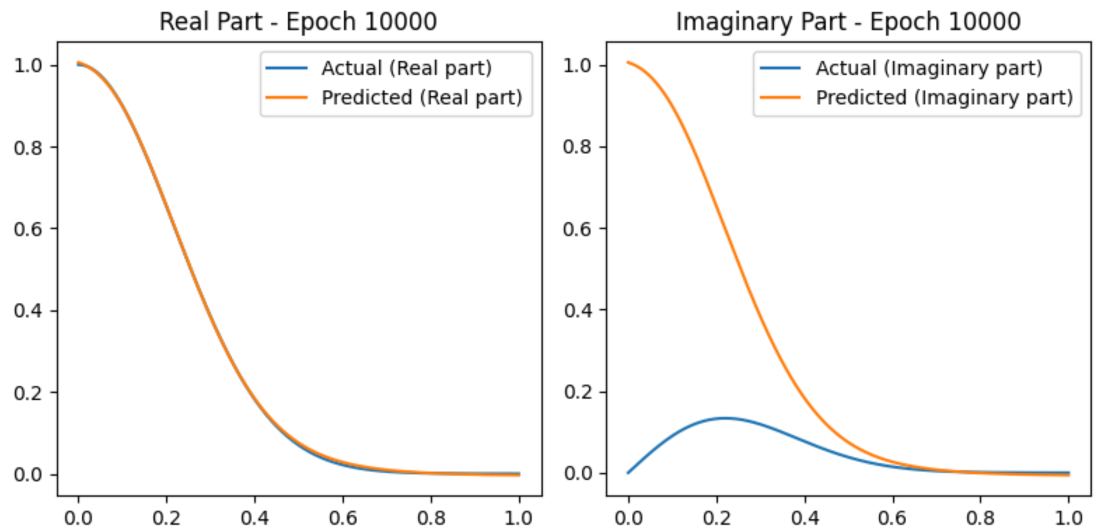




Loss vs epoch using relu

5.2.3 Using GaussianActivation

Predicted perfectly the training part but lags on the imaginary one.



Loss vs epoch of Gaussian Activation

6. CONCLUSION

6.1 INTERPRETATION OF RESULT

- The PINN successfully approximated the solution to the Nonlinear Schrödinger Equation (NLSE), producing results consistent with the analytical solution (Gaussian wave packet evolution).
- The PINN captured key features of the NLSE solution, including the wave packet's localization, dispersion, and nonlinear interaction effects.

6.1.1 Performance relative to expectations:

- The PINN demonstrated robust performance in solving a challenging partial differential equation by embedding the NLSE's physical laws directly into the loss function.
- The use of a **Gaussian activation function** further enhanced the network's ability to model localized wave behavior.

6.1.2 Comparison with traditional methods:

- Traditional numerical methods like spectral or finite difference methods are typically more efficient for lower-dimensional systems but struggle with adaptive or complex boundary conditions. The PINN offers a flexible, physics-informed alternative without requiring domain-specific numerical schemes.
- But our network lags little on correctly predicting the imaginary part.

6.2 LIMITATIONS

6.2.1. Computational Cost

Training the PINN is computationally expensive compared to traditional numerical solvers, particularly for high-dimensional problems or large domains.

6.2.2 Sensitivity to Hyperparameters

The performance of the PINN is highly sensitive to the choice of:

- Learning rate
- Activation function
- Weighting factors in the loss function

6.2.3 Solution Representation

- The current PINN is limited to one-dimensional systems with periodic boundary conditions. Extending to higher dimensions or more complex boundary conditions would require significant modifications.
- Also slightly deviated prediction on imaginary part.

6.3 FUTURE IMPROVEMENTS

6.3.1 Network Architecture

- Adaptive Activation Functions:
  - Implement trainable or custom activation functions to better capture the dynamics of localized and nonlinear solutions.
- Increased Network Depth:
  - Utilize deeper networks with regularization techniques (e.g., dropout) to model more complex features.

6.3.2 Loss Function Enhancements

- Adaptive Loss Weighting:
  - Dynamically adjust the weights of residual and boundary condition loss terms during training to improve convergence.
- Physics-informed Constraints:
  - Introduce penalty terms to enforce conservation laws (e.g., energy conservation) explicitly.

Overall the best prediction is obtained by using **Gaussian (exp(-x ** 2))** Activation with **10 hidden layers having 64 nodes** each and **learning rate of 1e-3** for model and also for physics loss parameter (lambda) with perfectly predicting the real part and slight deviation for the imaginary part as it is complex.

REFERENCES

1. https://github.com/maziarraissi/PINNs/blob/master/main/Data/NLS.mat
2. https://en.wikipedia.org/wiki/Nonlinear_Schr%C3%B6dinger_equation
3. Based on knowledge of previously submitted assignments.