Time Series Forecasting for Epidemic Prediction using PINN and LSTM

Kunal Ghosh
Indian Institute of Science
Bengaluru
kunalghosh@iisc.ac.in

Abstract

This study compares the performance of LSTM and PINN networks on a time-series dataset. The experiments explore the impact of various factors on the performance of these networks, such as learning rate, training data amount, sequence length, number of epochs, loss function, and number of LSTM layers

1 SIR model

The SIR model is one of the simplest compartmental models. Compartmental models are mathematical models used to study the spread and control of infectious diseases in populations. They divide a population into different compartments based on their disease status and model the transitions between these compartments over time. Also, people may progress between different compartments. The most commonly used compartmental model is the SIR model, which divides the population into three compartments: susceptible (S), infected (I), and recovered (R). The model assumes that individuals move from the susceptible compartment to the infected compartment at a rate proportional to the number of susceptible individuals and the number of infected individuals. Once infected, individuals move to the recovered compartment at a rate proportional to the number of infected individuals. The dynamics of the model is described using the following set of ordinary differential equations [Ref. 1]:

$$\frac{dS(t)}{dt} = \frac{-aS(t)I(t)}{N} \tag{1}$$

$$\frac{dI(t)}{dt} = \frac{aS(t)I(t)}{N} - bI(t) \tag{2}$$

$$\frac{dR(t)}{dt} = bI(t) \tag{3}$$

Given a time series dataset of daily new cases of some infectious disease (COVID-19) in a certain region, the time series forecasting problem is to predict the number of new cases in the next time period (such as the next day, week, or month) based on historical data. This can be formulated as a supervised learning problem, where the input variables are the previous values of the time series (e.g., the number of cases in the previous day or previous week), and the output variable is the number of cases in the next time period. The goal is to train an LSTM network and PINN that can accurately predict the future trend of the time series, which can help public health officials to make informed decisions about disease control measures and resource allocation. In addition to that, historical data can be used to estimate the parameters of the model, such as the rate at which people move from the susceptible group to the infected group (the infection rate, a), and the rate at which people move from the infected group to the recovered group (the recovery rate, b).

2 Recurrent Neural Networks (RNN)

RNNs are a type of neural network that can process sequences of inputs. Unlike feedforward neural networks, which process each input independently, RNNs have a hidden state that captures information about previous inputs in the sequence. This allows RNNs to model temporal dependencies in the data and make predictions based on the context of previous inputs. So, we can try to solve the given problem using RNN.

2.1 Long Short-Term Memory (LSTM) network

LSTM network is a type of RNN that is designed to address the vanishing gradient problem that can occur in traditional RNNs. The vanishing gradient problem occurs when the gradients used to update the network weights become very small, making it difficult for the network to learn long-term dependencies. LSTM networks achieve this by using a special cell state that can selectively add or remove information from the state over time. The cell state can be thought of as a conveyor belt that runs through the LSTM networks, with gates controlling the flow of information into and out of the cell state. The three gates in an LSTM network are:

- Forget gate: Controls how much of the previous cell state to forget.
- Input gate: Controls how much of the current input to add to the cell state.
- Output gate: Controls how much of the current cell state to output as the hidden state.

Each of these gates is usually implemented using a sigmoid activation function, which outputs values between 0 and 1. The gates can be thought of as switches that can be turned on or off, depending on the input and the previous state of the network. We will be using LSTM networks for our problem.

2.2 Gated Recurrent Unit (GRU) network

GRU network is a type of RNN that is similar to the LSTM network, but with fewer parameters. Like LSTM networks, GRU networks were designed to address the vanishing gradient problem that can occur in traditional RNNs. The GRU network also uses a gating mechanism to selectively allow information to flow through the network. However, unlike LSTMs, the GRU has only two gates:

- Update gate: Controls how much of the previous hidden state to keep and how much of the current input to add.
- Reset gate: Controls how much of the previous hidden state to forget.

Both of these gates are usually implemented using a sigmoid activation function, which outputs values between 0 and 1. The update gate can be thought of as a switch that controls how much of the current input to add to the previous hidden state, while the reset gate controls how much of the previous hidden state to forget.

Like LSTMs, the GRU also has a set of learning parameters that control how the gates operate. These parameters are updated during training using backpropagation through time.

GRU network is a simpler and more computationally efficient alternative to the LSTM, but it may not perform as well on tasks that require capturing longer-term dependencies in the data. So, it can be a good choice if we have limited computational resources and do not need to model very long-term dependencies. But in our problem, we have long-term dependencies. So, we cannot use GRU networks for our problem.

3 Physics-Informed Neural Network (PINN)

PINNs are a class of neural networks that incorporate physical laws or principles into the learning process of the neural network. This is achieved by enforcing the physical laws as constraints on the neural network during the training process. The idea behind PINNs is to enable the neural network to learn from data while respecting the underlying physical laws that govern the system being studied. This is usually incorporated into the loss function of the neural network. We will be incorporating the governing equations of the SIR model into the loss function of the PINN.

3.1 Data-driven solution for PDE

A data-driven solution for partial differential equations (PDEs) involves training a machine learning model to approximate the solution of a given PDE using data instead of traditional numerical methods that require explicit equations. This approach is particularly useful when explicit equations for a given PDE are difficult or impossible to obtain. In a data-driven solution for PDEs, the model is trained on a set of input-output pairs, where the inputs represent different parameter values of the PDE and the outputs represent the corresponding PDE solution values. The model then learns to generalize from the training data to make predictions for new input values. One common data-driven approach for solving PDEs is physics-informed neural networks (PINNs), which combine traditional numerical methods with deep neural networks to learn an approximation of the PDE solution. In PINNs, the neural network is trained to satisfy the PDE and the boundary conditions simultaneously, which ensures that the learned solution is physically meaningful and accurate. In our work, we will be trying to implement the data-driven solution for PDE, for the SIR model.

3.2 Data-driven discovery of PDEs

Data-driven discovery of PDEs is the process of discovering unknown or partially-known PDEs from observational data using machine learning techniques. The idea is to use data to infer the underlying physical laws that govern the observed phenomena, rather than starting with a known set of equations.

The process of data-driven discovery of PDEs involves several steps, including data collection, feature engineering, model selection, and model evaluation. The first step is to collect observational data that is representative of the phenomenon of interest. The data can be collected from experiments or simulations, or from natural systems such as the atmosphere or ocean.

The second step is feature engineering, which involves extracting relevant features from the data that can be used as inputs to a machine-learning model. These features may include time series data, spatial data, or other relevant measurements.

Once the features have been engineered, a suitable machine learning model is selected, which can be a neural network, support vector machine, or any other suitable model. The model is then trained on the data to learn the underlying patterns in the data, and to infer the unknown PDEs.

The final step is to evaluate the performance of the model, which involves assessing how well the inferred PDEs match the observed data. This can be done using various metrics such as mean squared error or correlation coefficients.

Data-driven discovery of PDEs has the potential to revolutionize the field of physics by enabling the discovery of new physical laws that govern observed phenomena. This approach is particularly useful when the PDEs are unknown or only partially known, and when traditional analytical methods are not sufficient to uncover the underlying physical laws.

4 Literature Survey

W. Hethcote (2000) provides an overview of mathematical models used to study the spread and control of infectious diseases. The article covers several types of models, including compartmental models, network models, and stochastic models, and discusses their applications to various diseases and interventions. The article emphasizes the role of mathematical modeling in informing public health policies and disease control strategies and highlights some of the challenges and limitations of these models. [Ref. 1]

Raissi et al (2018) propose a PINN as a framework that combines deep learning with partial differential equations (PDEs) to solve forward and inverse problems. PINNs incorporate the physical laws described by PDEs into the neural network architecture, enabling them to learn from limited data and generalize to new situations. The effectiveness of the PINN framework was demonstrated by applying it to several forward problems and inverse problems involving nonlinear PDEs, including the Schrodinger equation, the Burgers' equation and the Navier-Stokes equations. Both data-driven solutions of PDEs as well as the data-driven discovery of PDEs were performed and the results showed agreement with traditional methods for solving PDEs, as well as the data-driven discovery of PDEs accurately modeled the established equations. [Ref. 2]

Ouyang et al (2022) proposed a method for reconstructing the hydrofoil cavitation flow using chain-

style PINN. In a chain-style PINN, the output of one neural network is used as the input to the next neural network in the chain, and each neural network is responsible for modeling a specific physical property or aspect of the system. The chain-style PINN in the study consisted of three sub-networks. The validation was carried out on a 3D NACA66 hydrofoil and compared to Direct Numerical Simulation (DNS) results, computational efficiency was greatly improved accompanied by a small reduction in accuracy. [Ref. 3]

5 Methodology

We have used a conventional LSTM network to predict the data. We have trained the model for the first 122 days $(2/3rd)$ of data. Then we use the next 60 days $(1/3rd)$ of data to test the models. We have used mean square error loss and L1 loss as our loss functions for two separate cases. Also, we are using the Adam optimizer to optimize the weights and biases of the neural network.

5.1 Training

We have used a sliding time window approach to train our LSTM model. In the training, we trained our model with initial k days of data and then tried to predict the data for the $k + 1^{th}$ day. Here we have done experimentation for $k = 4$. After, predicting the data for the $k + 1^{th}$ day we have used that data to predict the data for the subsequent 10 days. (**Data:** For the purpose of experimentation we have used COVID-19 data for Canada.)

5.2 Architecture of Neural Network

We are using two LSTM layers with 20 neurons in each layer followed by a linear layer in our neural network. In addition to that we have three neurons both in the input layer and output layer. As we will be training our neural network on S, I and R and then try to predict their future values.

5.3 Loss Function for PINN

In PINN we have used the same LSTM network but with a different loss function. We are using the following loss functions to incorporate the governing equations of the SIR model. Two different loss functions are tried. (**NOTE:** $m(t)$ and $\bar{m}(t)$ are the actual quantity from the dataset and the predicted value, respectively.)

- MSE Loss: Mean squared error (MSE) loss is a commonly used loss function in regression problems, where the goal is to predict a continuous value. The MSE measures the average squared difference between the predicted value and the actual value for each example in the dataset.

$$MSE_{Loss} = \frac{1}{N}\left(\sum_{m=S,I,R}\left(\left(\frac{dm(t)}{dt} - \frac{d\bar{m}(t)}{dt}\right)^2 + (m(t) - \bar{m}(t))^2\right)\right) \tag{4}$$

- L1 Loss: L1 loss, or mean absolute error (MAE) loss, is a widely used loss function in regression problems. The L1 loss measures the average absolute difference between the predicted value and the actual value for each example in the dataset.

$$L1_{Loss} = \frac{1}{N}\left(\sum_{m=S,I,R}\left(\left|\frac{dm(t)}{dt} - \frac{d\bar{m}(t)}{dt}\right| + |m(t) - \bar{m}(t)|\right)\right) \tag{5}$$

6 Optimizer

6.1 Adam

We are using Adam (Adaptive Moment Estimation) as the optimizer during the training of the neural network. Adam is an optimization algorithm that is commonly used to update the parameters of a deep learning model during training. It is an extension of the stochastic gradient descent (SGD)

optimization algorithm, which updates the parameters based on the average of the gradients over a mini-batch of training examples.

Adam combines the advantages of two other optimization techniques: momentum and RMSprop. Momentum helps the optimization algorithm to move quickly along the steep directions of the cost function, while RMSprop helps the algorithm to scale the step size according to the gradients of the parameters.

The Adam algorithm maintains two moving averages of the gradients: the first moment (the mean) and the second moment (the uncentered variance). The first moment is similar to the momentum term in other optimization algorithms, while the second moment scales the step size based on the past gradients of the parameters. The update rule for the Adam optimization algorithm can be written as follows:

Calculate the first and second moments of the gradients:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{6}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{7}$$

Here, g_t is the gradient with respect to the model parameter at the current iteration, m_t and v_t are the first and second moments respectively, β_1 and β_2 are the decay rates for the first and second moments (usually set to 0.9 and 0.999 respectively), and m_{t-1} and v_{t-1} are the first and second moments from the previous iteration.

Compute the bias-corrected first and second moments:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{8}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{9}$$

Here, t is the current iteration, and $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected estimates of the first and second moments.

Update the model parameters:

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{10}$$

Here, θ_t is the model parameter at the current iteration, α is the learning rate, and ϵ is a small constant added for numerical stability (usually set to 10^{-8}).

7 Results

7.1 Learning Rate:

On increasing the learning rate LSTM is performing better than PINN. This is evident from the adjoining graphs. Actually, PINN is overfitting the data, in this case. If we reduce the number of epochs for the PINN then PINN is performing better than the LSTM. We have found from experimentation, PINN usually takes a lesser number of epochs to fit the model. (NOTE: We are keeping everything constant during experimentation, only changing the learning rate)

7.2 Training data amount:

On increasing the training data amount of the entire dataset. We observed that PINN is performing slightly better than the LSTM network. Also, there is a significant improvement in the performance of both networks. However, on decreasing the training data amount of the entire dataset, the performance of both networks deteriorates significantly. In this case, also we have found from experimentation that, PINN usually takes a lesser number of epochs to fit the model. (NOTE: We are keeping everything constant during experimentation, only changing the training data amount)

7.3 Sequence Length:

On increasing the number of days used to predict future data, the performance of both models improves significantly. Interestingly, PINN does not predict any unrealistic data. (Actually, LSTM is predicting a negative value of S, which is NOT possible.) However, on decreasing the number of days used to predict future data the performance of both models deteriorates.

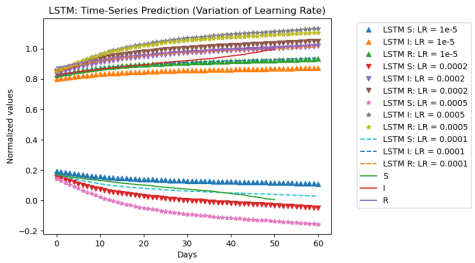

Figure 1:
Variation of Learning Rate for LSTM

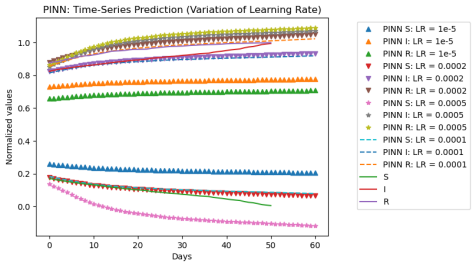

Figure 2:
Variation of Learning Rate for PINN

7.4 Number of epochs:

In general, we have found that PINN requires a much lesser number of epochs than LSTM.

7.5 Variation of the number of LSTM layers:

On increasing the number of LSTM layers, we have found that PINN is performing slightly better than the LSTM network. But as we increase the number of LSTM layers, both the network requires more epochs to train.

7.6 Variation of the number of neurons in LSTM layers:

On increasing the number of neurons in LSTM layers, we have found that both networks are performing slightly better. But again in this case PINN requires much fewer training epochs as compared to the LSTM Network.

7.7 Comparision of MSE Loss and L1 Loss:

In general, we have found from experimentation that MSE loss is giving much better results as compared to the L1 loss.

8 Conclusion

Based on the experiments and observations, it can be concluded that the performance of LSTM and PINN networks is dependent on various factors such as learning rate, training data amount, sequence length, number of epochs, loss function, and the number of LSTM layers. PINN generally requires a lesser number of epochs to fit the model compared to LSTM. Additionally, PINN performs better than LSTM in certain scenarios, such as when the learning rate is increased or the number of LSTM layers is increased. In addition to that, PINN also performs marginally better than LSTM in scenarios, such as when the training data amount is increased. Moreover, increasing the number of days used to predict future data improves the performance of both models. When we increased the number of neurons in the LSTM layers, we observed a slight improvement in the performance of both networks. However, it is worth noting that PINN still required fewer training epochs than the LSTM network in this scenario. Finally, the MSE loss function gives better results than the L1 loss function.
(**NOTE:** We do NOT expect our PINN model to generalize well for other datasets because a and b both will change for different datasets.)

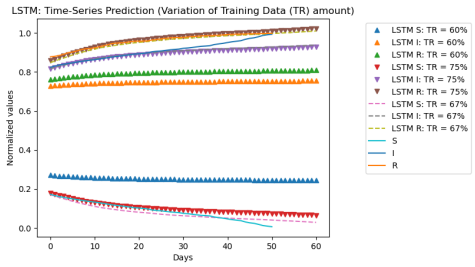

Figure 3:
Variation of training data amount for LSTM

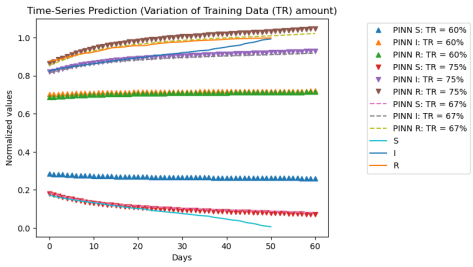

Figure 4:
Variation of training data amount for PINN

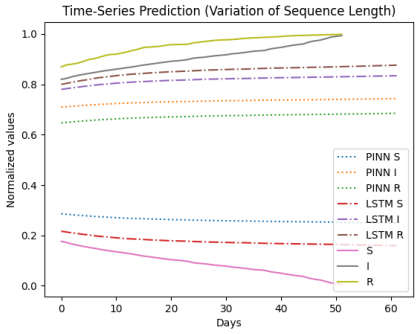

Figure 5:
Sequence Length: 2
Prediction Length: 10

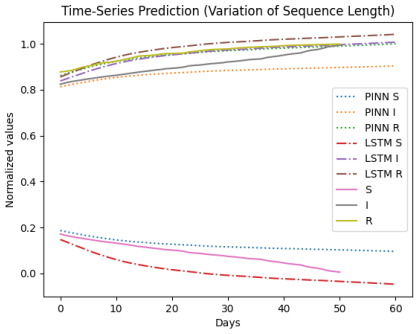

Figure 6:
Sequence Length: 5
Prediction Length: 10

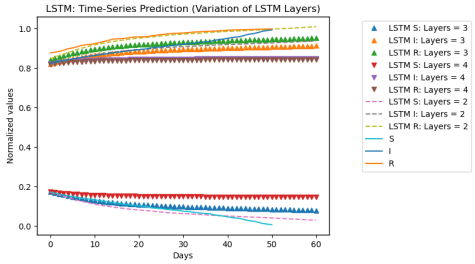

Figure 7:
Variation of the performance of LSTM on changing the number of LSTM layers

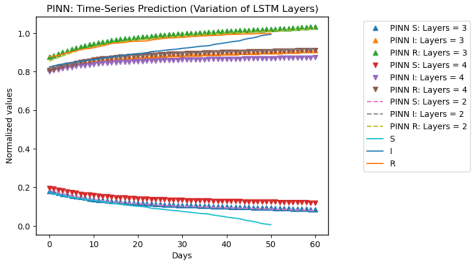

Figure 8:
Variation of the performance of PINN on changing the number of LSTM layers

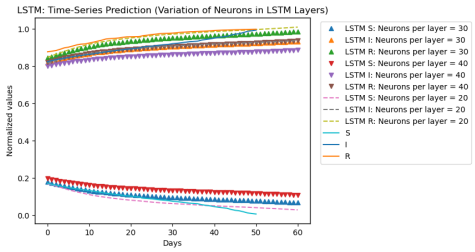

Figure 9:
LSTM: Variation of the number of neurons in LSTM layers

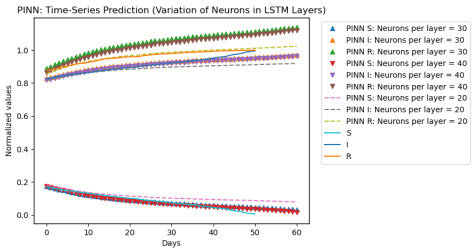

Figure 10:
PINN: Variation of the number of neurons in LSTM layers

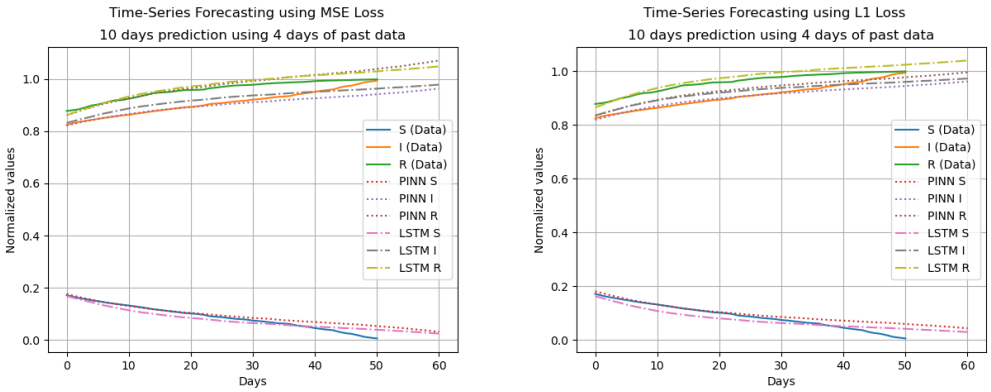

Figure 11: Comparision of MSE Loss and L1 Loss for PINN and LSTM

References

[1] Hethcote, Herbert W. (2000) The Mathematics of Infectious Diseases. *SIAM Review, Volume 42*, pp. 599-653. https://doi.org/10.1137/S0036144500371907

[2] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics, *Volume 378*, 2019, Pages 686-707, ISSN 0021-9991. https://doi.org/10.1016/j.jcp.2018.10.045.

[3] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, Journal of Computational Physics, *Volume 375*, 2019, Pages 1339-1364, ISSN 0021-9991. https://doi.org/10.1016/j.jcp.2018.08.029

[4] M. J. Keeling, P. Rohani, Modeling Infectious Diseases in Humans and Animals, Princeton University Press, 2008, https://doi.org/10.2307/j.ctvcm4gk0

[5] I. Cooper, A. Mondal, C. G. Antonopoulos, A SIR model assumption for the spread of COVID-19 in different communities, Chaos, Solitons and Fractals, 2020, https://doi.org/10.1016/j.chaos.2020.110057

[6] https://www.kaggle.com/datasets/tanuprabhu/population-by-country-2020

[7] https://www.kaggle.com/datasets/imdevskp/corona-virus-report?resource=download

[8] https://towardsdatascience.com/time-series-forecasting-with-deep-learning-in-pytorch-lstm- rnn-1ba339885f0c

[9] https://github.com/maziarraissi/PINNs

[10] https://towardsdatascience.com/physics-informed-neural-networks-pinns-an-intuitive-guide-fff138069563