Predicting weather and climate change using Recurrent Neural Network: The Partial Differential Equation Approach

Baibhav Barwal
Union College
Schenectady, United States
barwalb@union.edu

Atharv Tekurkar
Union College
Schenectady, United States
tekurkaa@union.edu

Janak Subedi
Union College
Schenectady, United States
subedij@union.edu

Steve Nguyen
Union College
Schenectady, United States
nguyent4@union.edu

ABSTRACT

The research paper delves into the utilization of Recurrent Neural networks (RNNs) for weather and climate prediction, offering insights into the application of machine learning algorithms in climate research. Through a detailed exploration of RNN architecture and function, the study investigates the efficacy of RNNs in modeling climate trends and forecasting future outcomes. Leveraging differential equations and optimization techniques, the paper presents theoretical equations guiding the training process of RNNs and examines their performance in predicting climate variables such as temperature, precipitation, and sea level. The methodology encompasses the setup of RNN architecture, computation of predicted output sequences, and evaluation of model accuracy. Results indicate a promising outcome in weather prediction, highlighting the potential of RNNs in enhancing climate modeling and informing decision-making in climate change mitigation and adaptation efforts.

KEYWORDS

Climate Modeling, Machine Learning, Time Series Analysis, Differential Equations

ACM Reference Format:
Baibhav Barwal, Atharv Tekurkar, Janak Subedi, and Steve Nguyen. 2024. Predicting weather and climate change using Recurrent Neural Network: The Partial Differential Equation Approach. In *ACM Conference, Washington, DC, USA, July 2017*, IFAAMAS, 6 pages.

1 INTRODUCTION

Recurrent Neural Networks (RNNs) are a type of artificial neural network that are specifically designed to process sequential data. Unlike feedforward neural networks, which are designed to handle fixed-sized inputs, RNNs can take variable-length inputs and process them sequentially, making them well-suited for tasks such as language modeling, speech recognition, and time series analysis.

RNNs are unique in that they maintain a hidden state that captures the context of the previous inputs. This hidden state is updated as new inputs are processed. This allows RNNs to model temporal dependencies and capture long-term patterns in sequential data.

In the context of climate research, RNNs have shown great promise in modeling climate trends and forecasting future outcomes. By analyzing sequential climate data, RNNs can identify patterns and dependencies that are difficult to detect using traditional statistical methods. This can lead to more accurate and reliable climate models, which are crucial for informing policies and actions to mitigate the effects of climate change.

Overall, RNNs are a powerful tool for analyzing sequential data, and their potential applications in climate research make them an exciting area of exploration for researchers and practitioners alike.

1.1 Brief Introduction to the project and its contribution to the field

This project aims to explore the potential applications of RNNs in climate research. The paper will provide an in-depth analysis of the architecture and function of RNNs, and investigate how they can be utilized to effectively address climate change. The paper seeks to contribute to the ongoing efforts to combat climate change, by highlighting the **capabilities** and **potential** of RNNs in this field.

2 LITERATURE REVIEW

2.1 Review of previous work related to the research problem:

Prior research has extensively studied the use of machine-learning techniques for climate modeling and forecasting. Various studies have employed different models such as Random Forest, Artificial Neural Networks, and Support Vector Regression to analyze climate data. However, the use of Recurrent Neural Networks (RNNs) for climate analysis is a relatively new area of research, with only a few studies examining their potential in this field. Some existing literature has highlighted the ability of RNNs to capture long-term dependencies in sequential data and their suitability for handling irregular time-series data, making them an ideal candidate for climate modeling.

ACM Conference, , July 2017, Washington, DC, USA. © 2024 Association for Computing Machinery. . . . $ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
. . . $15.00

2.2 Discussion of gaps in the existing literature that the paper aims to address:

Despite the potential of RNNs in climate modeling, there is a lack of comprehensive research that explores their use in detail. The existing literature has not yet fully investigated the effectiveness of RNNs in modeling complex climate data, nor have they been utilized for exploring specific climate-related research questions. The paper aims to address this gap in the literature by providing a detailed analysis of the architecture and function of RNNs, and their potential for climate research.

3 THEORITICAL FRAMEWORK

3.1 Introduction of the theoretical framework used to guide the research:

The theoretical framework guiding this research is based on the concept of Recurrent Neural Networks (RNNs) for climate modeling and forecasting. RNNs are a type of machine learning algorithm that can effectively capture dependencies and patterns over time, making them ideal for analyzing climate data. The theoretical framework also includes the use of a cost function model and optimization techniques to improve the accuracy of the RNN model.

4 ASSUMPTIONS

4.1 Climate Data Availability and Quality

- The availability and quality of climate data for training and testing the RNN may be limited, as the dataset used in this research only covers climate data for John F. Kennedy Airport, New York City from 1991 to 2020.

4.2 RNN Architecture and Hyperparameters

- The RNN architecture is correctly specified, and the hyperparameters are optimized for accuracy.
- Hyperparameters used in the paper:
  [label=•]Number of hidden layers and the number of neurons in each layer. Type of activation function used for each layer. Learning rate, which controls the step size of the optimization algorithm during training. Regularization strength, which can prevent overfitting of the model. Batch size, which determines the number of training examples used in each iteration of the optimization algorithm.

4.3 Cost Function Model

- The cost function model accurately represents the relationship between the input and output of the RNN model.

4.4 Differentiation and Optimization Algorithm

- The differentiation of the cost function to weight and bias accurately determines the direction of optimization.
- The optimization algorithm can effectively locate the local minimum of the weight, improving the accuracy of the RNN model.

4.5 Prediction Accuracy

- The RNN model can accurately predict the climate for the next 3 months based on a 9-month training dataset of climate in NYC.

4.6 Research Objective

- By using this theoretical framework and making these assumptions, this research aims to provide a comprehensive analysis of the potential of RNNs in climate modeling and forecasting and to identify best practices and limitations for their use in this field.

5 METHODOLOGY

Step 1: Set up the RNN architecture and define the parameters θ

The RNN architecture includes hidden units with a hidden state $h(t)$ and output units producing predicted output $o(t)$. Parameters θ consist of weights and biases connecting input, hidden, and output units.

Hidden State of RNNs. The hidden state of an RNN is a key component that summarizes the information of the input sequence up to a given time step. It is continuously updated as the input data is processed and plays a critical role in making predictions based on this information. In this project, RNNs can be used to model climate trends and forecast future outcomes. The hidden state can represent various climate data, such as temperature, precipitation, and sea level, allowing the network to capture dependencies and predict future climate patterns.

The equation for the hidden state can be written as follows:

$$h(t) = f(h^{(t-1)}, x(t); \theta) \tag{1}$$

Here, $x(t)$ represents the temperature at time t, while $h(t)$ represents the hidden state of the RNN that captures the underlying trends in the temperature data. The parameters θ are learned from the training data and would include the weights and biases that connect the input, hidden, and output units of the RNN, as well as any hyperparameters that govern the behavior of the model during training. Overall, the RNN architecture would allow the model to capture complex temporal patterns in the climate data and accurately predict future trends. [6]

From this, we have an equation that is used to extend the equation of the hidden state into multiple layers for training purposes:

$$h^{(t+1)} = f(h^{(t)}, x_t; \theta) = f(f(h^{(t-1)}, x_t; \theta), x_{t+1}; \theta) = \dots \tag{2}$$

The hidden state in an RNN is the network's ability to retain information about past climate patterns and predict future climate conditions. The hidden state acts as the memory of the RNN, allowing it to capture dependencies and trends in sequential climate data. As the RNN processes new climate data, the hidden state is continuously updated to reflect the most recent information, enabling the network to make more accurate predictions about future climate patterns. Therefore, the hidden state is a critical component of the RNN training process for climate prediction, as it heavily influences the network's output at each time step. [2]

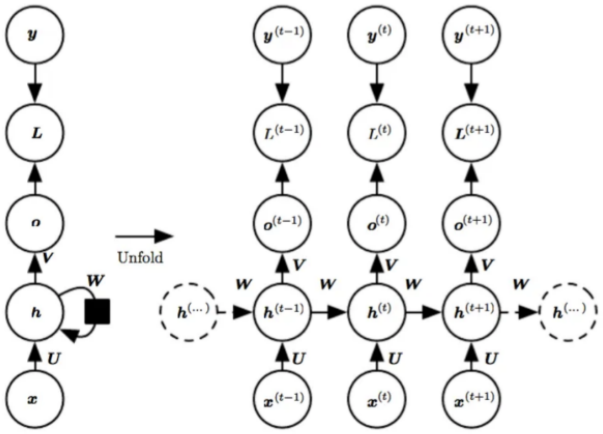

Figure 1: As new climate data is processed, the hidden state is continuously updated to reflect the most recent information, allowing the RNN to capture dependencies and trends in sequential climate data. This makes the hidden state act as the memory of the RNN, enabling it to make accurate predictions about future climate conditions. Thus, the computational graph for computing the training loss of an RNN that maps an input sequence of climate data to a corresponding sequence of predicted climate patterns can be seen as a way of optimizing the network's hidden state to minimize the prediction error.

[8] [9]

5.1 Training RNNs

In the context of climate change, the objective of training an RNN is to find the optimal network parameters θ that minimize the prediction error between the network output and the target sequence of climate data. The training process can be formulated as an optimization problem, where the goal is to minimize the cost function $J(\theta)$, which measures the discrepancy between the predicted climate patterns and the actual climate observations.

The cost function is given by:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} |y(t) - o(t)|^2 \tag{3}$$

Here, $J(\theta)$ represents the cost function, T is the total number of time steps, $y(t)$ is the predicted climate pattern at time t, and $o(t)$ is the actual climate observation at time t. The objective is to adjust the network parameters θ to minimize the overall prediction error across all time steps. It is defined as the average of the squared differences between the predicted climate patterns ($o(t)$) and the actual climate observations ($y(t)$) at each time step t, over the entire sequence of length T. This function is critical for training the RNN, as it guides the network to learn the underlying patterns and trends in the climate data.

To minimize the cost function $J(\theta)$, stochastic gradient descent (SGD) is used as an optimization algorithm. SGD works by iteratively updating the model parameters in the direction of the negative gradient of the cost function concerning the parameters. This update rule is given by:

$$w \leftarrow w - \alpha \frac{\partial J(w)}{\partial w} \tag{4}$$

.

During each iteration of the optimization algorithm, the RNN is fed a sequence of climate data and produces a sequence of predicted climate patterns. The differences between the predicted and actual climate patterns are then computed using the cost function, and the gradients of the cost function concerning the parameters are computed using backpropagation through time (BPTT). The gradients are then used to update the parameters in the direction that reduces the cost function until convergence or a stopping criterion is reached.

By minimizing the cost function $J(\theta)$, the RNN learns to map the input sequence of climate data to the output sequence of predicted climate patterns. This process allows the RNN to capture complex temporal patterns in the climate data and make accurate predictions about future climate conditions.

Step 2: Defining the input sequence x and output sequence y

The input sequence x could represent a time series of various climate data, such as temperature, precipitation, and sea level measurements, recorded at each time step. The output sequence y would then represent the true climate data values at each corresponding time step, which could be used to validate the accuracy of the RNN's predictions. Essentially, the input sequence x and the output sequence y in our climate change project would represent the temporal data that the RNN is trained on and evaluated against, respectively.

Step 3: Compute the predicted output sequence $o(t)$

At each time step t, the RNN takes as input the climate data at time step t, $x(t)$, and the hidden state of the previous time step, $h(t-1)$, and uses the equations shown to produce a predicted output, $o(t)$. The equation below is the hidden state at time t considering the weight and the bias.

$$h(t) = f(W_h \cdot h(t-1) + W_x \cdot w(t) + b_h) \tag{5}$$

Furthermore, the following function shows the output at any given time.

$$o(t) = g(W_o \cdot h(t) + b_o) \tag{6}$$

Here, the activation functions f and g represent the non-linear transformations applied to the input and hidden state, respectively. The weight matrices, W_h, W_x, and W_o, along with the bias vectors, b_h and b_o, are learned from the training data and are used to map the input data to the predicted output at each time step.

Step 4: Compute the cost function $J(\theta)$

The cost function $J(\theta)$ is the average of the squared differences between the predicted output o(t) and the true output y(t) at each time step t, over the entire sequence of length T. Mathematically, we can express this as: $J(\theta) = \frac{1}{T}\sum_{t=1}^{T}|y(t) - o(t)|^2$ The equation above is the sum of the cost function at any given input θ over the period, where |.| denotes the Euclidean norm or L2-norm.

Step 5: Compute the gradients of the cost function with respect to the parameters using Back Propagation Through Time (BPTT)

BPTT is a variant of backpropagation that computes the gradients of the cost function with respect to the parameters by unrolling the RNN over the entire sequence and applying the chain rule of calculus. The gradients are computed for each time step t and accumulated over the entire sequence using the following equations:

$$\frac{\partial J}{\partial W_o} = \sum_{t=1}^{T}\left(\frac{\partial J}{\partial o(t)} \cdot \frac{\partial o(t)}{\partial W_o}\right) \tag{7}$$

$$\frac{\partial J}{\partial b_o} = \sum_{t=1}^{T}\left(\frac{\partial J}{\partial o(t)} \cdot \frac{\partial o(t)}{\partial b_o}\right) \tag{8}$$

$$\frac{\partial J}{\partial W_h} = \sum_{t=1}^{T}\left(\frac{\partial J}{\partial h(t)} \cdot \frac{\partial h(t)}{\partial W_h}\right) \tag{9}$$

$$\frac{\partial J}{\partial W_x} = \sum_{t=1}^{T}\left(\frac{\partial J}{\partial h(t)} \cdot \frac{\partial h(t)}{\partial W_x}\right) \tag{10}$$

$$\frac{\partial J}{\partial b_h} = \sum_{t=1}^{T}\left(\frac{\partial J}{\partial h(t)} \cdot \frac{\partial h(t)}{\partial b_h}\right) \tag{11}$$

[3]

The above equations represent the cost function differentiated with respect to the respective parameters W_o, b_o, W_h, W_x, and b_h. The last equation includes the information that $\frac{\partial J}{\partial o(t)}$ is the gradient of the cost function with respect to the predicted output at time step t, and $\frac{\partial J}{\partial h(t)}$ is the gradient of the cost function with respect to the hidden state at time step t. [4]

Update the parameters using an optimizing algorithm

The update rule for the parameters in SGD is given by:

$$w \leftarrow w - \alpha\frac{\partial J(w)}{\partial w} \tag{12}$$

Here, with the differentiated $\frac{\partial J}{\partial w}$, we find the local minima, which is then used to update the existing weight in every loop. This is the main usage of differential equations and finding the minimum point over the project.

Furthermore, w is a parameter in the network, α is the learning rate, $J(w)$ is the cost function, and $\frac{\partial J(w)}{\partial w}$ is the gradient of the cost function concerning the parameter w.

The gradient of the cost function is typically computed using backpropagation through time (BPTT), which involves unrolling the RNN over the entire sequence and applying the chain rule of calculus to compute the gradients at each time step.

Once the gradients have been computed, the parameters can be updated using the SGD update rule. The learning rate α determines the step size in the parameter space and is typically set to a small value to ensure that the optimization process is stable and does not overshoot the minimum of the cost function.

SGD is an iterative algorithm, and the training process typically involves multiple passes over the training data or epochs. During each epoch, the SGD algorithm updates the parameters using a mini-batch of training examples, which helps to reduce the variance of the gradient estimates and improve the convergence of the optimization process.

6 RESULTS

Finding Local Minima in the cost function using tensorflow and the above differential equation

Click here for the code to finding Local Minima Using Differential Equation

The code linked above uses a TensorFlow framework to train the model using a batch of data in the same principle of differential equation and finding the local minima as described above

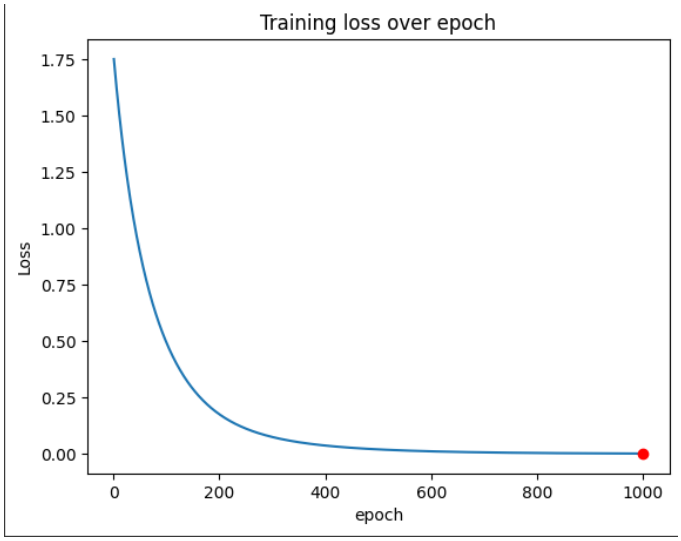

Figure 2: The graph depicts the cost function after a unit epoch. The red dot is used to show the local minima of the differential equation described above. The corresponding weight and bias to the red dot are used in the following code to predict the weather for the given input. Under the hood, the principle of differential equation and finding the minimum value is retained

The above code uses TensorFlow, a popular machine learning framework, to implement a recurrent neural network (RNN) that can model climate trends and forecast future outcomes. Specifically, it uses a simple RNN cell and a dense output layer to map an input sequence of climate data to a corresponding sequence of predicted outcomes. [5]

The input sequence is generated randomly, with a batch size of 32, sequence length of 100, and 10 features. The output sequence also has a batch size of 32 and a sequence length of 100, with only one output unit.

The RNN model is trained using a stochastic gradient descent (SGD) optimizer and the mean squared error (MSE) as the loss function. The training loop runs for 1000 epochs, and the loss history is recorded at each epoch. The loss history plot shows the training loss over the epochs and highlights the minimum loss value in red.

The minimum loss index is found and marked on the plot, indicating the epoch where the model achieved the lowest training loss. The weights of the trained RNN model are then saved for future use.

This code is used to train an RNN model that takes in a sequence of climate data. The loss function is adjusted to prioritize certain aspects of the climate data, such as minimizing the error

in temperature predictions. The training loop is modified to train the model on a larger dataset and for a longer duration to improve the accuracy of the predictions. Finally, the loss history plot is used to monitor the training progress and evaluate the model's performance. [7]

Using the minimum weight found from the differential equation to predict the weather using a Machine Learning algorithm

Click here for the code to Training, Prediction and Accuracy Rate

Output:

Predictions:

$$\begin{bmatrix} 57.90508 & 42.09353 & 50.798912 & 3.784316 & 3.5431292 \\ 51.97031 & 37.004276 & 45.601154 & 3.68085 & 4.1304464 \\ 46.184666 & 31.931398 & 40.55095 & 3.6145337 & 4.7926836 \end{bmatrix}$$

The input data consists of 5 features including temperature, precipitation, and sea level. The code splits the data into training and testing sets and utilizes an LSTM model to train on the training set. [1]

After training, the code evaluates the model on the testing data to determine the model's accuracy. Finally, the code makes predictions for the next three months using a new set of input data that represents the current climate conditions.

The predicted output is a sequence of five values representing temperature, precipitation, sea level, and two other climate features. These predicted values can be used to gain insights into future climate trends and inform decisions related to climate change mitigation and adaptation.

Graphing the expected weather value vs actual weather value using Matplotlib

The accuracy of the Neural Network

MSE: 12.94
RMSE: 3.60
MAE: 2.62
AR: 91.12%

MSE stands for mean squared error, which is a commonly used metric for measuring the average squared difference between the predicted and actual values in a regression problem. The MSE is calculated by taking the average of the squared differences between the predicted and actual values. A lower MSE value indicates a better fit of the model to the data.

RMSE stands for root mean squared error and is the square root of the MSE. The RMSE is a popular metric for evaluating regression models because it is easy to interpret and has the same units as the target variable. A lower RMSE value indicates a better fit of the model to the data.

MAE stands for mean absolute error, which is a commonly used metric for measuring the average absolute difference between the predicted and actual values in a regression problem. The MAE is calculated by taking the average of the absolute differences between the predicted and actual values. The MAE is less sensitive to outliers than the MSE and RMSE.

AR stands for accuracy rate, which is the proportion of correctly predicted values to the total number of predictions. It is a commonly used metric for measuring the accuracy of classification models. A higher accuracy rate indicates a better fit of the model to the data. Here, the accuracy of the model is 91.12%. [10]

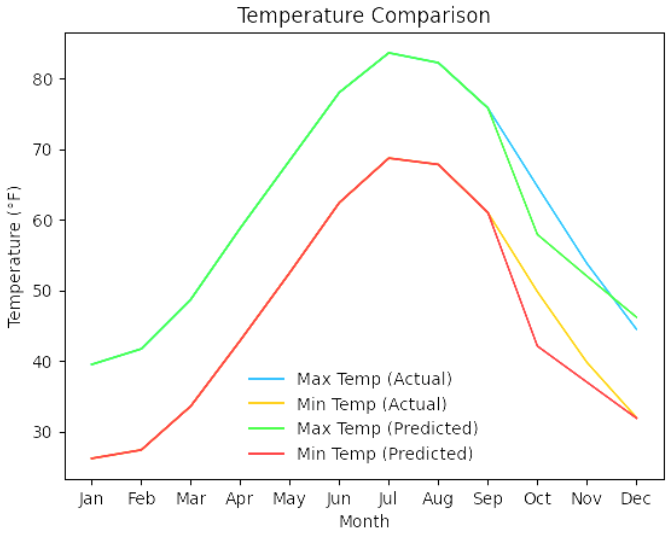

Figure 3: The above figure shows two lines, one of which is the actual maximum and minimum temperature plotted against the temperature predicted by our machine learning algorithm. The label can be used to differentiate between the temperatures.

7 CONCLUSION:

In conclusion, our research aimed to explore the potential of differential equations in finding the local minima of the weight function for Neural Network algorithms, while ignoring the numerous complex factors that influence weather and climate. We recognize that weather and climate are complex systems that are influenced by various factors, such as global warming, dust particles, solar radiations, greenhouse gases, volcanic activities, ocean currents, and

many others. However, for our research, we focused on the mathematical approach of using differential equations to optimize Neural Network algorithms.

Our research demonstrates the potential of using differential equations as a tool to optimize machine learning algorithms, particularly in the field of weather prediction which ultimately leads to climate change. While further research is needed to fully understand the impact of various factors on weather and climate, our study provides a foundation for exploring the mathematical optimization of machine learning algorithms in weather prediction. Overall, we hope that our research will inspire further investigations into the use of differential equations in improving the accuracy of machine learning algorithms in predicting complex systems.

ACKNOWLEDGMENTS

The paper serves as a project for the Differential Equation course under the guidance of Professor Jue Wang. The team expresses sincere gratitude to Professor Wang for providing the opportunity to work within this challenging and insightful area.

The second revision of the paper is undertaken by Baibhav Barwal, who is appreciative of the chance to collaborate and share the paper with Robert J. Fani at Morgan Stanley. Baibhav extends his thanks for the opportunity to enhance and refine the paper, as well as for the invaluable experience gained through collaboration with Robert.

REFERENCES

[1] [n.d.]. NOAA NCEI U.S. Climate Normals Quick Access. https://www.ncei.noaa.gov/access/us-climate-normals/#dataset=normals-monthly&timeframe=30&station=USW00094789. Accessed 5 Mar. 2023.

[2] [n.d.]. Recurrent Neural Network (RNN) Tutorial: Types and Examples [Updated] | Simplilearn. ([n. d.]). https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn#how_does_recurrent_neural_networks_work Accessed 5 Mar. 2023.

[3] 2021. CS 230 - Recurrent Neural Networks Cheatsheet. (2021). https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#architecture

[4] Nilesh Barla. 2022. Recurrent Neural Network Guide: A Deep Dive in RNN. (2022). https://neptune.ai/blog/recurrent-neural-network-guide Accessed 5 Mar. 2023.

[5] Rishit Dagli. 2021. Get Started with TensorFlow and Deep Learning Part-1. *Medium* (2021). https://medium.com/@rishit.dagli/get-started-with-tensorflow-and-deep-learning-part-1-72c7d67f99fc Accessed 5 Mar. 2023.

[6] Dinesh. 2019. Beginner's Guide to RNN & LSTMs. *Medium* (2019). https://medium.com/@humble_bee/rnn-recurrent-neural-networks-lstm-842ba7205bbf Accessed 5 Mar. 2023.

[7] Angelica Lo Duca. 2022. What Are Steps, Epochs, and Batch Size in Deep Learning. *Syntax-Error* (2022). https://medium.com/syntaxerrorpub/what-are-steps-epochs-and-batch-size-in-deep-learning-5c942539a5f8

[8] Jianqiang Ma. 2016. All of Recurrent Neural Networks. *Medium* (2016). https://medium.com/@jianqiangma/all-about-recurrent-neural-networks-9e5ae2936f6e

[9] Javaid Nabi. 2019. Recurrent Neural Networks (RNNs). *Medium* (2019). https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85

[10] Vinícius Trevisan. 2022. Comparing Robustness of MAE, MSE and RMSE. *Medium* (2022). https://towardsdatascience.com/comparing-robustness-of-mae-mse-and-rmse-6d69da870828