Weather Prediction in Oslo using an RNN

Richard Fang
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, TX, USA
rxf200009@utdallas.edu

Long Vu
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, TX, USA
lqv210000@utdallas.edu

Devanshu Kumar
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, TX, USA
dxk210062@utdallas.edu

Aditya Katariya
Erik Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, TX, USA
axk220164@utdallas.edu

***Abstract*—Deep learning and recurrent neural networks (RNNs) are a promising new technique in machine learning to make predictions from time series data. This project investigates the use of recurrent neural networks (RNN) for temperature prediction in Oslo, Norway. A dataset with daily mean temperature, humidity, and wind speed for 10 years was split into independent continuous 5-day series, and these series were used to train an RNN model. The RNN model was able to reach a maximum R^2 of 0.9221. The simplicity of the custom model holds it back, and further development is necessary for a practical forecast.**

Keywords— RNN, backpropagation through time, weather, temperature, forecasting

I. Introduction

The human brain possesses a very unique ability to learn, which allows adaptation to a wide variety of situations and the discovery of solutions to a wide variety of problems. For instance, humans have discovered physical laws, exploited regularities in the stock market, projected future events like the weather, and developed optimizations to everyday processes. On the other hand, the machines created by humans lack the same capacity, requiring a human's thinking to instruct them through every step. At the same time, because human cognition is a limited resource, it is in the interest of development and progress to simulate the same ability in machines, which are faster and more scalable.

A. Artificial Neural Networks

Inspired by the human brain, Rumelhart, Hinton, and Williams created the backpropagation algorithm to train an artificial neural network [1]. The initial networks mapped input to output through various neurons, although there were no cycles in the flow of input to output. Despite the simplicity, these "one-pass" artificial neural networks found powerful applications, especially when the number of neurons increased in a hidden layer. For example, one hidden layer neural networks were able to successfully steer a self-driving car, recognize vowel sounds in speech, and do a four-class classification of human faces [2]. However, these artificial neural networks were still relatively simple, and there was potential for further accuracy gain and better effectiveness on particular types of problems.

B. Deep Learning

After the discovery of backpropagation, the algorithm naturally generalized to any number of hidden layers, as it only required uniform applications of the calculus multivariate chain rule [2]. General research curiosity about extending neural networks into increasingly deeper layers emerged. As computational technology like central processing units and graphics processing units improved, these deep learning models also became more feasible for practical tasks. For example, deep learning found significant applications in image recognition, including facial, medical, and satellite [3]. These models included those trained for cancer detection and land-type classification and surveying [3]. However, "one-pass" deep learning still had a challenge: it only handled data collected at one moment.

C. Recurrent Neural Networks

Recurrent neural networks (RNNs) were designed with time in mind. These "multiple-pass" networks fed the output back into the input to incorporate new information at multiple time points. For longer sequences of time points or data, the initial input traveled through long paths of neurons, thus requiring deep learning considerations. Because of its compatibility with sequential data, these networks found diverse applications in sentence generation, language translation, and time series tasks [4]. This project, in particular, focuses on the time series capabilities that RNNs provide.

II. Background Work

Weather prediction is a particular application of RNNs for time series data. An initial source for this project involved a more volatile form of time series data, stocks. Zhu constructs a (50, 100) RNN and applies it to 5-day and 10-day time series of Apple stocks [5]. The resulting predictions mirror the actual data points closely, with the predicted value within ±6 of the true value (on average) for the 5-day series. However, the model yielded a larger mean absolute error and mean squared error on the 10-day series.

Xu et al. develops a simple RNN-based model to predict rainfall in Chinese cities with 21 days of time series data [6]. The study found that more inputs per time improved predictive accuracy, as their optimal feature configuration included all available inputs. They also achieved >60% precision and 90% recall with the simple RNN.

Srivastava and S. create a temperature prediction model for a single city in Australia [7]. They take 10 days of temperature data and predict the 11th day. Using an LSTM, they were able to achieve a mean squared error of less than 0.06 for the regression outcome. Their R^2 value was 0.998.

III. Theoretical and Conceptual Study

In the briefest possible form, the RNN is a form of deep neural network where time groups some of its hidden layers. The simplest abstract representation is a neural net that sends some of its output back to its input [8]. However, that form does not reveal implementation insights, and Fig. 1 shows a more useful representation where an individual neural net is chained.

Fig. 1 shows a sequence of units, each of which contains its own neural network (shown in blue). Each unit has two high level interfaces: input (green) and output (red). A unit's neural network takes a state from the previous unit and inputs from the data. It passes the data through its hidden layers and can create an output value/prediction or a state as a result of its computation. Each unit of the RNN can be considered as a different point in time. Then, each input block is the input at a different time step. Finally, to tie this to the abstract representation, the neural nets inside each unit share the same weights, so they can be imagined as the same neural net.

A. Loss Function Design

Because of the multiple units, there are many more choices for loss in an RNN. The two main candidates for this project were a "many-to-many" architecture and a "many-to-one" architecture. [8] describes the two options. The "many-to-many" architecture involves collecting the output of each RNN unit and creating a loss function that minimizes the error of each individual output. The "many-to-one" architecture is more similar to a deep neural net, where the loss is computed after passing through all the hidden layers at all time steps.

Fig. 2 illustrates the two types of networks. The arrows show a loose flow of the forward pass. Starting from loss, the "many-to-many" architecture requires visiting a given RNN unit multiple times: once for its direct output, and once for each output after it. In this design, a given weight is updated $O(t^2)$ times. A given weight appears in t RNN units, and a weight in an RNN unit is updated for each subsequent output. On the other hand, the "many-to-one" architecture has a linear flow of values. A weight is updated each time it appears in an RNN unit. The path from the loss only visits each RNN unit once.

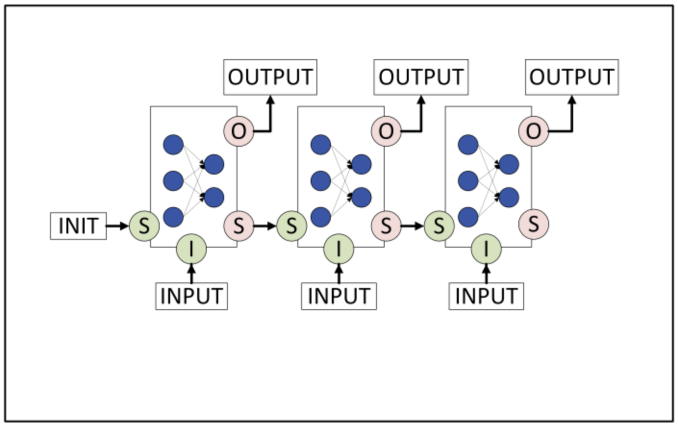


Fig. 1. RNN Schematic. Original diagram, based on [8].

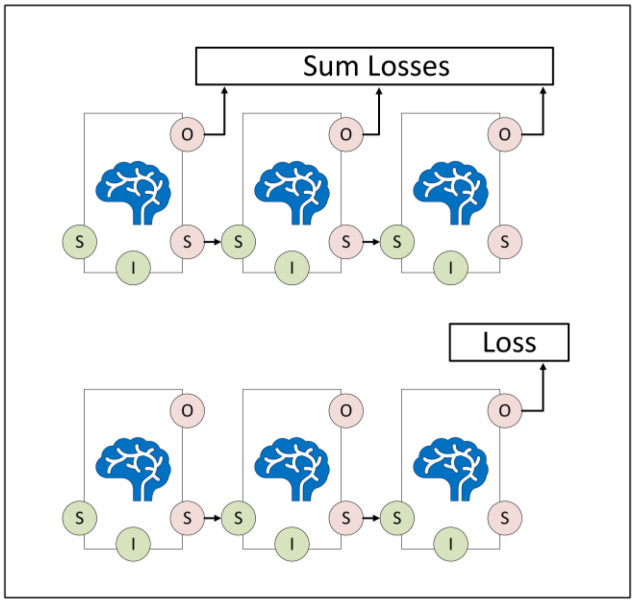


Fig. 2. "Many-to-many" (top) and "many-to-one" (bottom). Original diagram, based on [8].

Immediately, the "many-to-many" architecture requires more computation, leading to longer training times. However, another problem is that the "many-to-many" architecture inherently demands more complexity. By minimizing loss with each unit's output, this option forces the same weights to produce different outputs at different times. Since this project's dataset was limited, the additional complexity was troublesome, and "many-to-one" was more suitable.

B. Backpropagation Through Time

Backpropagation through time (BPT) is a more complex version of the standard backpropagation for simple neural networks. If the network of Fig. 1 were expanded to show all connections, the algorithm would just be gradient descent on the weights of the resulting "simple" neural network [8]. However, computing the derivatives with respect to each weight is not a simple task due to the number of dependencies. Luckily, a simple intuition of backpropagation involves a small weight change rippling through all subsequent neurons [9]. This brings to mind a mnemonic of summing a derivative term for each possible path through the neural net graph, which suffices for implementation.

C. Connection Design

The two major architectures of "simple" RNNs include the Elman network and Jordan network. [10] gives a description of each. The distinguishing feature of an Elman network is that the hidden layer output is fed back into the network as a state. On the other hand, the Jordan network computes the whole output before feeding it back as state. (The Jordan network additionally computes the new state from previous output and old state.)

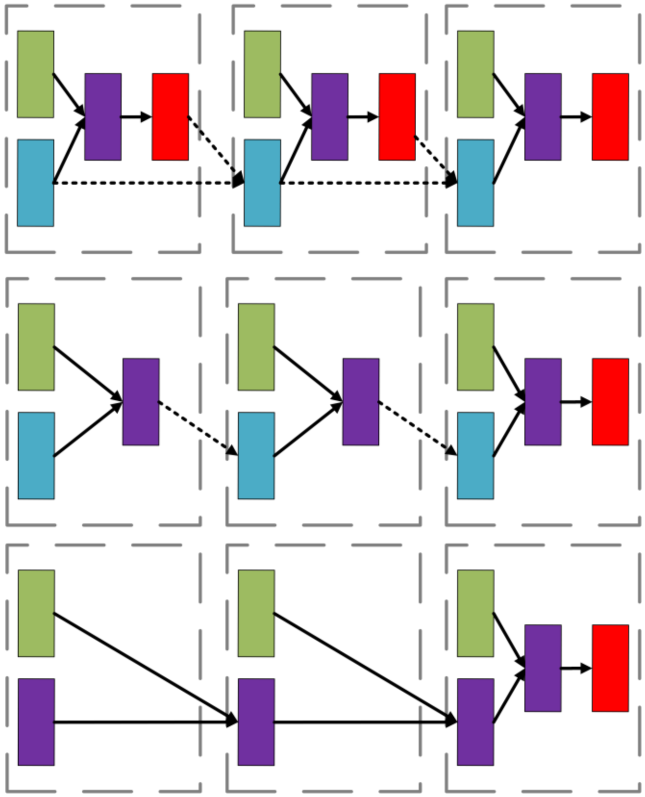


Fig. 3. Different “simple” RNNs. Original diagram, based on [10].

Since there was no need to compute intermediate outputs, the Jordan network added an unnecessary step. However, the primary advantage of the Elman network was that its gradient was simple to analyze. Removing the context block (see Fig. 3) and unrolling, the Elman network looks very similar to a deep “one-pass” neural net. Each hidden layer fully connects to the next hidden layer, until the final output produces a prediction. On the other hand, the Jordan network requires two parallel paths between RNN units when computing gradient.

Fig. 3 shows a Jordan network (top), an Elman network (middle), and the adapted Elman network (bottom) for this project. The green box is the input block, the blue box is the context block, the purple box are the hidden layers, and the red box is the output block. The solid arrows indicate full connections; the dotted arrows indicate element-wise connections with fixed weights [10]. The adapted Elman network and the standard Elman network have the same number of learnable weights (and thus learning potential), but the adapted Elman network is easier to program. Ignoring the intermediate inputs, the adapted Elman network allows a BPT algorithm that directly parallels standard backpropagation. Note that the hidden layer of unit u is drawn in unit u+1, as the adaptation combines context and hidden layer into one entity.

IV. Data and Task Design

The base dataset for this project came from Zenodo [11]. It consists of weather measures from 2000 to 2010 in multiple European cities. For this project, Oslo data was extracted, and then narrowed to temperature, humidity, and wind speed. Then, five consecutive day series of temperature, humidity, and wind speed were created. The temperature of the sixth day became the label for regression. The final dataset contained 609 datapoints. The task is predicting a temperature based on a 5-day time series with 3 inputs.

Originally, there was an additional classification task for the neural net that determined whether the temperature of the 6th day went up or down relative to the 5th day. Attempts at making the neural net simultaneously learn a classification and regression task, however, did not yield good results. Similarly, computing the classification value by comparing the regression value against the 5th day’s temperature also did not yield high accuracy. The classification task was scrapped, but the label remains in the dataset.

V. Results and Evaluation

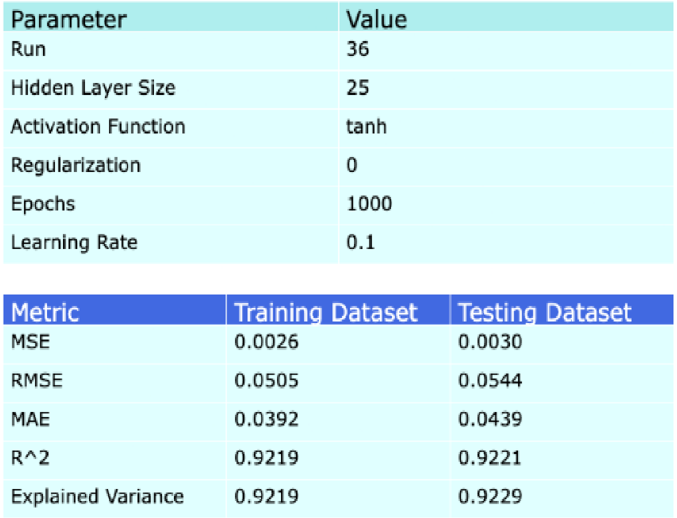
A. *Methodology*

The following analysis focuses on the R^2 value. R^2 is the percentage of variance in the dependent variable the independent variables can explain in a regression model. A higher score indicates a better fit since the train and test data were constant. MSE, RMSE, MAE, and explained variance were also measured. The five tunable hyperparameters were the activation function, hidden layer size, epochs, learning rate, and regularization. The performance data was collected from 70 experimental trials. Each trial differed by one hyperparameter relative to one or multiple sets of trials.

B. *Analysis*

1) *Overall Best Model*

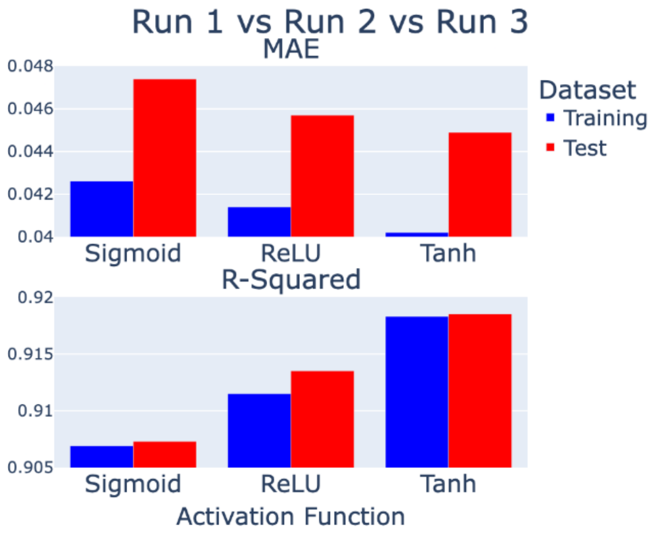
The results of the repeated assessments show that model performance is highly sensitive to hyperparameter choices. The best model (see Fig. 4) achieved a testing dataset $R^2 = 0.9221$ and a training dataset $R^2 = 0.9219$. The RMSE average deviation from the scaled temperature was 0.0544 (2.4° from true). The MAE average deviation was 0.0439 (1.9°). Determining overfitting is ambiguous; while MSE, RMSE, and MAE errors are less optimal, the variance-based measures that adjust to each dataset are more optimal. The model has decent predictive power for simple weather patterns, although more exactness is desirable.



Parameter	Value
Run	36
Hidden Layer Size	25
Activation Function	tanh
Regularization	0
Epochs	1000
Learning Rate	0.1

Metric	Training Dataset	Testing Dataset
MSE	0.0026	0.0030
RMSE	0.0505	0.0544
MAE	0.0392	0.0439
R^2	0.9219	0.9221
Explained Variance	0.9219	0.9229

Fig. 4. Hyperparameters and metrics for the optimal run, run 36.



2) *Activation Function*

Activation function was the initial optimization. In Fig. 5, tanh (R^2 = 0.9185), performed better than sigmoid (R^2 = 0.9073) and RELU (R^2 = 0.9135). Tanh also had the lowest MAE. Even with further specifics like regularization for RELU (see Fig. 7 later), tanh remained the clear best fit for activation functions.

3) *Hidden Layer Size*

The results demonstrate an optimal range for the number of neurons in the hidden layer. Table I fixes the activation at tanh, regularization at 0, epochs at 1000, and learning rate at 0.01. Overall performance was high at 5, 20, and 25. However, because 5 tended to perform poorly for other configurations, 20-25 is the likely true peak. Between 5-20 was the size used initially and then higher sizes of around 30-50 were used to settle on this value. A model with too little neurons underfits the data, while a model with too many neurons overfits and takes longer to converge.

4) *Epochs and Learning Rate*

The combination of using 1000 epochs and a learning rate of 0.1 showcased the best results, avoiding the overfitting from too many epochs, the lack of convergence from too few epochs, the fluctuation and instability of larger learning rates, or the slow process of lower learning rates. The trials explored epochs from 100 to 1500. Similarly, the trials explored learning rates from 0.005 to 0.25.

TABLE I. VARYING HIDDEN LAYER SIZES

Hidden Layer Sizes	**R^2 values**	
	Training R^2	*Test R^2*
5	0.9191	0.9211
10	0.9188	0.9204
20	0.9191	0.9207
25	0.9192	0.9207
30	0.9185	0.9197
40	0.9183	0.9195
50	0.9183	0.9195

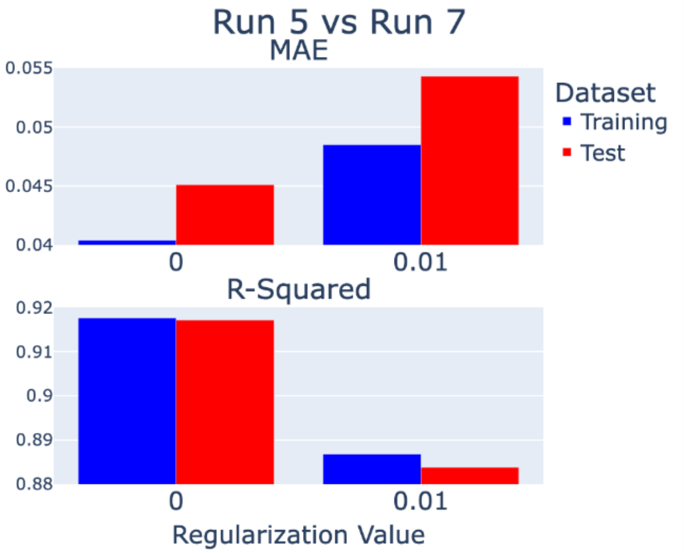


Fig. 6. Varying regularization with hidden layer size 20, tanh activation, epochs 100, learning rate 0.05.

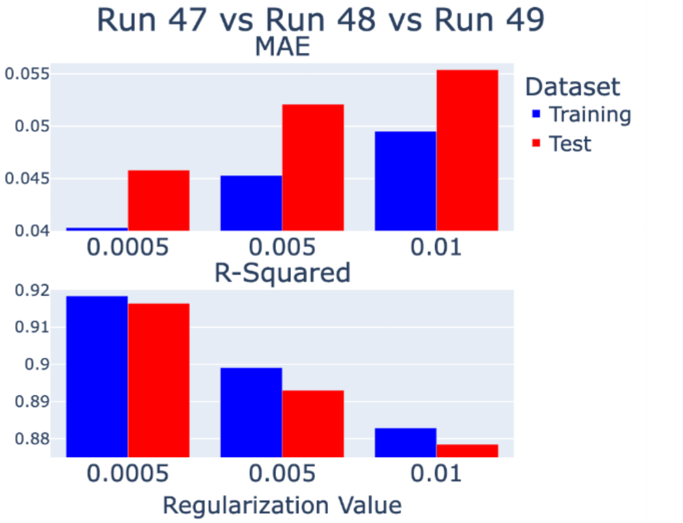


Fig. 7. Varying regularization with hidden layer size 50, RELU activation, epochs 1500, learning rate 0.05.

5) *Regularization*

All relevant comparisons (Fig. 6 and Fig. 7) showed that L_2 regularization generally shifted the train and test R^2 scores lower and made MAE worse. Therefore, it was likely unnecessary for this model and this dataset, although in a few circumstances, relatively smaller values (i.e., 0.0005) did help slightly.

C. Discussion

The results give insight into advantages and limitations of the simple RNN used here. Primary advantages include simplicity and understandability. With fewer hyperparameters, the effect of each can be studied without either neglecting many combinations or facing combinatoric explosion. An estimate of typical performance and performance limit can be discovered quickly.

Primary weaknesses include training stability and the data itself. Although simple RNNs often struggle with the vanishing gradient problem, this is a minor concern with very short 5-day series. The complex error surface is more impactful. This implementation, without momentum, can be prone to trapping in local minima and missing the optimal set of weights. Additionally, it is very dependent and sensitive to the random initialization; the training can yield high or low levels of performance from the initial weights alone. Further, the overall randomness of weather data creates the risk that a minimum may not be a good solution; in that case, the model can only give a decent approximation based on minima.

For 5-day patterns, a simple RNN is suitable. The model selected a hidden layer size of 25, which expresses many possible functions yet limits overfitting potential. The tanh activation is better than the sigmoid and RELU activations because of its zero-centered output (from –1 to 1), which allows flexible gradient updates in both the positive and negative direction. Sigmoid and RELU have a positive range, so weight updates can be pushed into one direction.

0,

VI. Future Work

The main limitations of the current approach are its simple design and lack of diverse representations for many variables. The following modifications can overcome these limitations:

- *Deeper hidden layers*: If each hidden layer block (see purple boxes of Fig. 3) contained more hidden layers, the RNN would have more hypotheses to choose from. There may be better models among them.
- *Multiple output optimization*: The architecture could be changed from "many-to-one" to "many-to-many," where each intermediate output is optimized as well. By encouraging models that match each day's temperature, the algorithm can explore a region of possible models that is currently hard to reach.
- *Complex RNN architecture*: Using a Long Short-Term Memory network can allow an accurate forecast across a longer time series, which may provide more stability to the prediction instead of 5-day series.

Additionally, the following adaptations can increase real-world utility:

- *Longer forecasts*: The model could be applied to longer time series (e.g., 7 days or 14 days) by rebuilding the dataset with longer input series and/or output series.
- *Multiple variable prediction*: The model could predict temperature, humidity, and wind speed for a more detailed forecast.

VII. Conclusion

This report described the study, design, and tuning of a simple RNN for daily temperature forecasting. The best discovered hyperparameters gave an R^2 of 0.9221 on the test set and an average deviation of 1.9° from true temperatures. Despite RELU being a choice function for vanishing gradients, tanh was the most suitable function in a simple RNN. Regularization, although an overfitting avoidance tool, can hurt performance if the model is inherently simple. Since the implementation was simple, future work for a more expressive RNN may improve results and utility.

VIII. References

[1] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533-536, Oct. 1986. [Online]. Available: https://www.nature.com/articles/323533a0

[2] T. Mitchell, "Artificial neural networks," in *Machine Learning*, 1st ed. USA: McGraw-Hill, 1997, ch. 4, pp. 81-124.

[3] Y. Li. (2022). Research and application of deep learning in image recognition. Presented at ICPECA 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9718847

[4] I. Mienye, T. Swart, and G. Obaido, "Recurrent neural networks: A comprehensive review of architectures, variants, and applications," *Information*, vol. 15, no. 9, art. 517, Aug. 2024. [Online]. Available: https://www.mdpi.com/2078-2489/15/9/517

[5] Y. Zhu, "Stock price prediction using the RNN model," *Journal of Physics: Conference Series*, vol. 1650, no. 2, Oct. 2020. [Online]. Available: https://iopscience.iop.org/volume/1742-6596/1650

[6] Y. Xu, Z. Yang, F. Zhang, X. Chen, and H. Zhou, "A rainfall prediction model based on ERA5 and Elman neural network," *Advances in Space Research*, vol. 75, no. 2, pp. 1732-1746, Jan. 2025. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0273117724010214

[7] A. Srivastava, A. S. (2022). Weather prediction using LSTM neural networks. Presented at I2CT 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9824268

[8] Stanford University. (2017). Recurrent neural networks. [Online]. Available: https://www.youtube.com/watch?v=6niqTuYFZLQ

[9] M. Nielsen. "How the backpropagation algorithm works." Neural Networks and Deep Learning. Accessed: Aug. 6, 2025. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap2.html

[10] J. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179-211, 1990. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1207/s15516709cog1402_1

[11] F. Huber, D. van Kuppevelt, P. Steinbach, C. Sauze, Y. Liu, B. Weel, Sept. 2022, "Weather prediction dataset," Zenodo. [Online]. Available: https://zenodo.org/records/7525955