KATHOLISCHE UNIVERSITÄT
EICHSTÄTT-INGOLSTADT

MIDS

Uncertainty Quantification in Shallow Water Model

(as part of the Data Lab Project in SS 2024)

Author:	Hiyabu Ghebresslaise
	Pavitra Chandarana
Matriculation Number:	290813
	202679
Email Address:	hiyabu.ghebresslasie@stud.ku.de
	pavitra.chandarana@stud.ku.de
Semester:	SS 2024
Place:	Ingolstadt
Date:	July 18, 2024
Examiner:	Prof. Janjic

Abstract

Weather forecasting is vital for many industries, such as disaster relief, transportation, and agriculture. With advancements in weather forecasting, such as Google's DeepMind model and the integration of AI, a significant improvement in accuracy has been noticed in recent years. However, with improved accuracy, efficiency still remains an issue. To bridge this gap, this paper investigates how machine learning methods can improve computational efficiency and be realistic for simultaneously testing data assimilation methods. We are precisely studying the ensemble method and its effectiveness in terms of statistical properties. Our research heavily focuses on the behavior of the clouds, specifically on the 3 parameters: height, wind, and rain. Our main findings demonstrate that the algorithms are good at predicting the mean of the parameters but perform poorly for standard deviation. The results reveal how effective machine learning algorithms are for specific statistical properties, even with constraints like limited data and resources.

Contents

List of Figures

List of Tables

List of Symbols

X	Train Data X
Y	Ground Truth Y
X'	Scaled Train Data
i, j	Indices for spatial dimensions
$\mathbf{X} * \mathbf{K}$	Convolution of input $\mathbf{X}$ with filter $\mathbf{K}$
$\text{ReLU}(z)$	Rectified Linear Unit activation function
z	Value in the feature map
$\text{MaxPool}(\mathbf{X})(i, j)$	Max pooling operation on feature map $\mathbf{X}$
$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$	Output from fully connected layer
Loss	Cross-entropy loss for classification tasks
y_c, $\hat{y}_c$	True label and predicted probability for class c
θ	Model parameters
η	Learning rate
$\frac{\partial \text{Loss}}{\partial \theta}$	Gradient of loss with respect to parameter θ
$\mathbf{h}_t$	Hidden state of RNN at time step t
$\mathbf{y}_t$	Output of RNN at time step t
$\mathbf{c}_t$, $\tilde{\mathbf{c}}_t$	Memory cell states in LSTM
$\mathbf{f}_t$, $\mathbf{i}_t$, $\mathbf{o}_t$	Forget, input, and output gates in LSTM
Loss	Mean squared error (MSE) for regression tasks
$\hat{\mathbf{y}}_t$	Predicted output at time step t
$\frac{\partial \text{Loss}}{\partial \theta}$	Gradient of loss with respect to parameter θ in BPTT
$\hat{Y}_i$	Predicted Value
$\bar{Y}_i$	Mean of Ground Truth

1 Introduction

The weather has evolved a lot in the past decade. The global average temperature has significantly increased by 25% from 0.88°C in 2010 to 1.1°C in 2019. Also, the weather extremes have become more frequent. Events like heat weaves, traditional levels of rain and floods, and drought are becoming more severe [1]. Due to these reasons, weather forecasting has become more vital as well as challenging. The main setbacks in forecasting weather are the quality of the observed data collected, amount of data, timeliness, etc [2]. Intending to use machine learning, this paper examines how well machine learning algorithms can be used to make future predictions when limited to reduced data and inferior hardware support. In particular, we are using the ensemble method to predict properties for the near future. Traditionally, the ensemble methods were implemented manually by computing properties for distinct ensembles. However, computing them manually for each ensemble can be tedious in terms of time and resources. This is when machine learning comes into the picture. We are trying to improve the runtime efficiency by training a machine learning algorithm with limited sequential data and trying to predict statistical metrics for the future. In our case, the data will be generated using the shallow water model, a simplified atmospheric model used to study weather patterns without the need for complex computations and based on shallow water equations. We are using this toy model to generate the data for quantitative analysis later.[3]

2 Overview of the data

Utilizing the Shallow water model, we ran multiple simulations of different ensemble members. Initially, we planned to extract an ensemble of 10,000 members and assign it ground truth. But we ran into hardware capability issues and had to compromise using an ensemble of 1000 as our ground truth. For analysis, we fixed the range for ensemble members from 6 to 1000 and selected intermediate values, including 10, 15, 100, 250, 500, and 750. Each ensemble consists of 60 time steps across 1000 grid points and includes 3 variables, as follows:

- **u** - Velocity of wind
- **h** - Height of clouds
- **r** - Rain

Reading all the ensembles for each variable across 60-time steps turned out to be a very memory-intensive task, and conducting mathematical computations would take an unreasonable amount of time on typical hardware. To resolve this issue, we tried fixing the problem using two different methods. i) working in chunks and ii) converting data into pickle format, but neither of the methods achieved our goal. Hence, we were forced to consider only 6 time steps rather than 60. This was a huge data loss, but given the issue at hand and the time constraint we had, this seemed to be the best option.

3 Main Work

3.1 Statistical analysis (on ensembles)

Following data extraction, our goal is to study how mean and standard deviation behaves across different ensembles.

Variables u, h, and r exhibit vast differences in magnitude scales, so it is vital to scale the data before performing any analysis. There are many scalers available, which is why it is important to choose a scaler cautiously, as choosing the wrong one can lead to incorrect interpretation and, consequently, a wrong conclusion. In our case, we used MinMaxScaler to standardize the data and brought each variable to a common range between 0 and 1, which increased the interpretability of the data without much loss of information. The formula of MinMaxScaler is as follows:

$$X' = \frac{X - \min(X)}{\max(X) - \min(X)} \tag{3.1}$$

On the contrary, if we had used StandardScaler, we would have lost crucial information from our data, as the mean and standard deviation would have been set to 0 and 1, respectively. Maintaining the originality of the mean and standard deviation was crucial for our study, as they were the variables on which our analysis primarily focused. Firstly, we wanted to see how the overall mean of all ensembles across time looked.

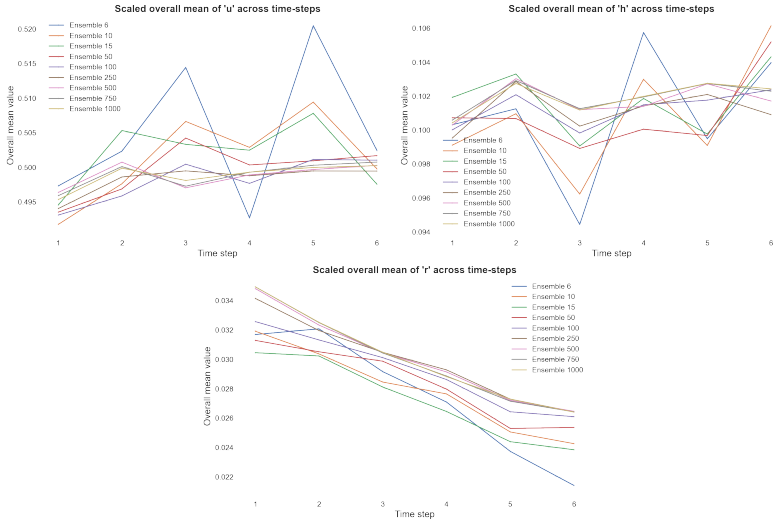

Figure 3.1: Overall Mean across time for all ensembles

As expected, we can observe from fig 3.1 that as the number of ensembles increases, the mean is more stable across time. i.e., ensemble 6 has the most fluctuation, while on the other hand, ensemble 1000 is the most linear (stable). This happens with u and h variables while less noticeable in r since the variability in r is considerably low. Next up we will be examining how standard deviation varies across time for each ensemble.

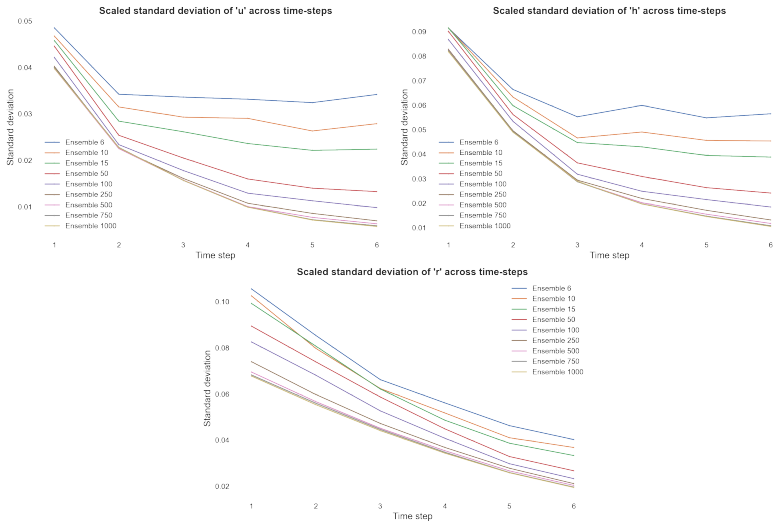

Figure 3.2: Standard Deviation across time for all ensembles

Interestingly, figure 3.2 shows that the higher the number of ensembles, the lower the standard deviation at time step 6. Another noticeable observation is that the difference between the highest and lowest ensemble increases over the time steps for the variables

u and h; conversely, the difference slightly decreases for variable r. Furthermore, let's investigate how the error in the mean behaves across different ensembles.

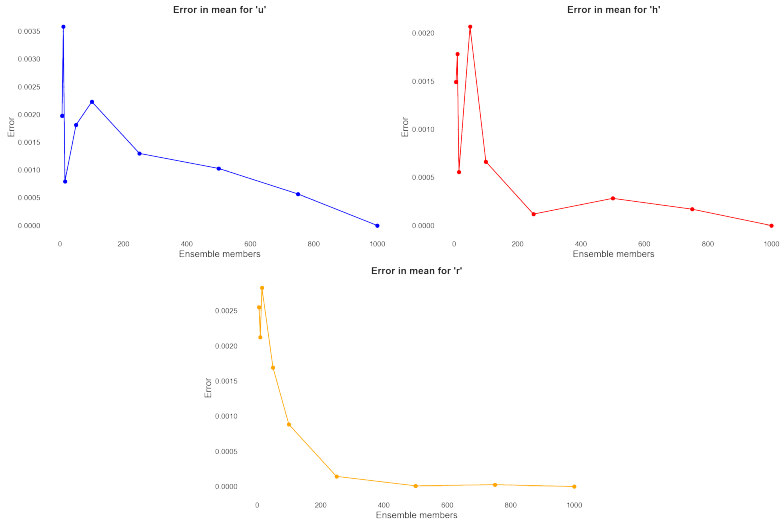

Figure 3.3: Error in means w.r.t ground truth across different ensembles

Figure 3.3 illustrates that smaller ensembles, i.e., from 6 to 100, exhibit more fluctuation, gradually stabilizing with an increase in ensemble members. Variables u and h are comparatively more unstable than r, which is consistent with the outcome of the plots above. Moving forward, we study the error in standard deviation across different ensembles.

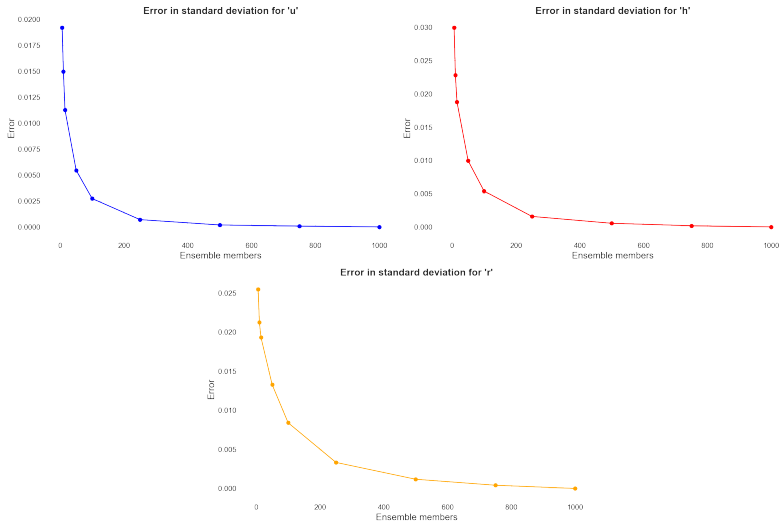

Figure 3.4: Error in standard deviation w.r.t ground truth

From figure 3.4, it is clear that there are no fluctuations in any of the variables. However, it is noteworthy that variables u and h have a higher convergence rate than r.

Statistical analysis concludes that increasing the number of ensemble members reduces the error for both the parameters, i.e., mean and standard deviation.

3.2 Machine learning

Following statistical analysis, we want to answer our main question: Can machine learning models predict the mean and standard deviation of the next time step? Our data is sequential concerning time (time steps) and space (grid points). Given the current time step t_0, we want to predict t_1 and t_2, which we refer to as $t+\Delta$ and $t+2\Delta$, respectively. Our machine learning algorithms are Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). CNNs and RNNs are types of deep learning architectures that define machine learning.

3.2.1 Convolutional Neural Networks (CNN)

1. Objective: CNNs learn spatial hierarchies of features from input images or grid-like data.

2. Convolution Operation: The convolutional layer filters the input data to produce a feature map. For a 2D input $\mathbf{X}$ and a 2D filter $\mathbf{K}$, the convolution is:

$$(\mathbf{X} * \mathbf{K})(i, j) = \sum_m \sum_n \mathbf{X}(i+m, j+n) \cdot \mathbf{K}(m, n)$$

where i and j index the spatial dimensions.

3. Activation Function: An activation function (such as ReLU) is applied to the feature map to introduce non-linearity:

$$\text{ReLU}(z) = \max(0, z)$$

where z is a value in the feature map.

4. Pooling Layer: Pooling layers reduce the spatial dimensions of feature maps. Max pooling takes the maximum value in a window (e.g., 3×3):

$$\text{MaxPool}(\mathbf{X})(i,j) = \max_{m,n} \mathbf{X}(i+m, j+n)$$

where (i, j) indexes the pooled feature map.

5. Fully Connected Layer: After convolutional and pooling layers, the output is flattened and fed into a fully connected layer:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

where $\mathbf{x}$ is the flattened feature map.

6. Loss Function: For classification tasks, the cross-entropy loss is used:

$$\text{Loss} = -\sum_{c} y_c \log(\hat{y}_c)$$

where y_c is the true label and $\hat{y}_c$ is the predicted probability for class c.

7. Training the Model: Model parameters are optimized by minimizing the loss function using gradient descent:

$$\theta := \theta - \eta \frac{\partial \text{Loss}}{\partial \theta}$$

where η is the learning rate. (In our case we used Adam optimizer as that performed better).

8. Backpropagation: Gradients are computed using backpropagation. For a parameter θ:

$$\frac{\partial \text{Loss}}{\partial \theta} = \frac{\partial \text{Loss}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \theta}$$

[4]

3.2.2 Recurrent Neural Networks (RNN)

1. Objective: RNNs model sequential data by capturing temporal dependencies through recurrent connections.

2. Recurrent Layer: An RNN processes data by maintaining a hidden state $\mathbf{h}_t$:

$$\mathbf{h}_t = \sigma(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

where σ is an activation function, $\mathbf{W}_{hx}$ and $\mathbf{W}_{hh}$ are weight matrices.

3. Output Layer: The output $\mathbf{y}_t$ at time step t is:

$$\mathbf{y}_t = \mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y$$

where $\mathbf{W}_{hy}$ is the weight matrix.

4. Long Short-Term Memory (LSTM): LSTMs address the limitations of vanilla RNNs by introducing memory cells and gates to better capture long-term dependencies:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

where $\mathbf{f}_t$, $\mathbf{i}_t$, and $\mathbf{o}_t$ are forget, input, and output gates, respectively.

5. Loss Function: For sequence prediction tasks, the loss is the sum of losses at each time step. Using mean squared error (MSE) for regression:

$$\text{Loss} = \frac{1}{T}\sum_{t=1}^{T}(\mathbf{y}_t - \hat{\mathbf{y}}_t)^2$$

6. Training the Model: Parameters are optimized by minimizing the loss function using gradient descent:

$$\theta := \theta - \eta\frac{\partial \text{Loss}}{\partial \theta}$$

(In our case we used Adam optimizer as that performed better).

7. Backpropagation Through Time (BPTT): Gradients are computed using BPTT,

unrolling the RNN through time. For a parameter θ:

$$\frac{\partial \text{Loss}}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial \text{Loss}_t}{\partial \theta}$$

[5]

3.3 Data Pre-Processing

After gaining a good understanding of the models. Let's delve into data preparation for the machine learning part. Here, we considered all 60 time steps and an ensemble of 1000 members. To improve runtime efficiency, we randomly created subsets of only 20 ensemble members from the 1000-member ensemble file, sampling every 6^{th} time step, i.e., 0,6,12,18,24,30,36,42,48,54 using python indexing. This resulted in a 3-dimensional ndarray. As mentioned, we then created subsets of 20 members randomly and fixed a random seed for reproducibility. We scaled our data using MinMaxScaler and split our train and test data into an 80-20% ratio.

As discussed earlier, our data is sequential in terms of both space and time, so splitting it randomly would be very problematic. Therefore, we split the data sequentially. For $t + \Delta$, we considered the first 9 time steps (i.e., 1-48) for X, our training data, and the last 9 time steps (i.e., 6-54) for Y, which serves as our ground truth. For $t + 2\Delta$, we considered the first 8 time steps (i.e., 1-42) for X and the last 8 time steps (i.e., 6-54) for Y. As we want to predict $t + \Delta$ and $t + 2\Delta$, this method ensures that we have corresponding ground truth values for each instance in the training data, which is important for validation purposes.

While training the models, our input is (20,3000) (i.e., the dimension of a single subset), and our output would be the same. This is how we predict future values for variable u, h, and r. Subsequently, we use a custom function to extract the mean and standard deviation from the input and output. Finally, we evaluate the results.

3.4 Results

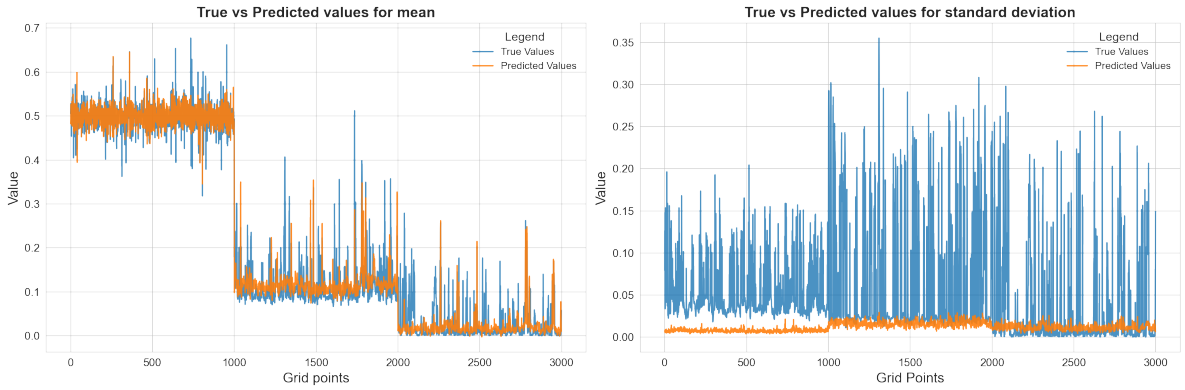

Figure 3.5: CNN Mean and Standard deviation Predictions for $t + \Delta$

Based on Figure 3.5, the CNN model performs considerably better for mean predictions than standard deviation predictions. To gain more insights, let's examine the true vs predicted values for both the mean and standard deviation.

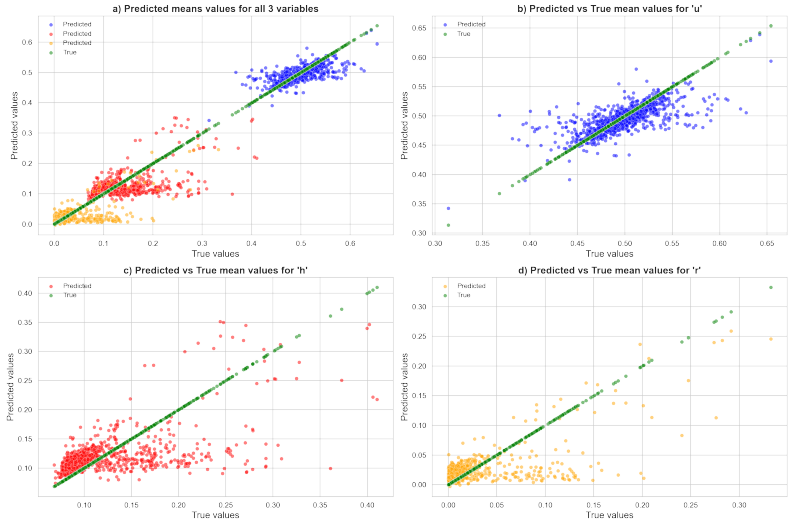

Figure 3.6: CNN Mean Predictions for each variable $t + \Delta$

Plot a) in Figure 3.6 displays predicted mean values for all 3 variables. The data points are color-coded as blue, red, and yellow for the variables u, h, and r respectively. By taking a closer look at u in plot b), we can see that predicted data points are concentrated around the green line indicating high accuracy. On the contrary, data points in plots c) and d) suggest poor accuracy for h and r as the predicted data points are more sparse around the green line (true values).

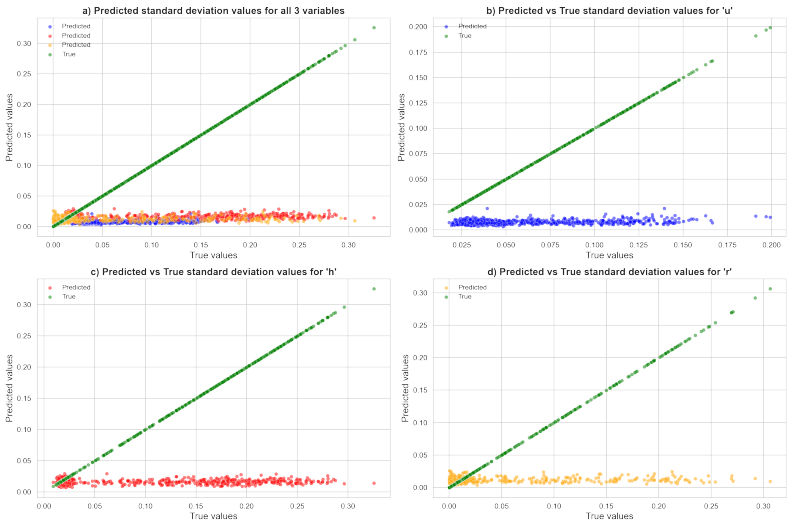

Figure 3.7: CNN Standard Deviation Predictions for each variable $t + \Delta$

Figure 3.7 proclaims that the CNN model performs terribly for predicting standard deviation as the predicted data points in all the subplots are not closely aligned with the true values. Now, let us study RNNs' performance for mean and standard deviation predictions.

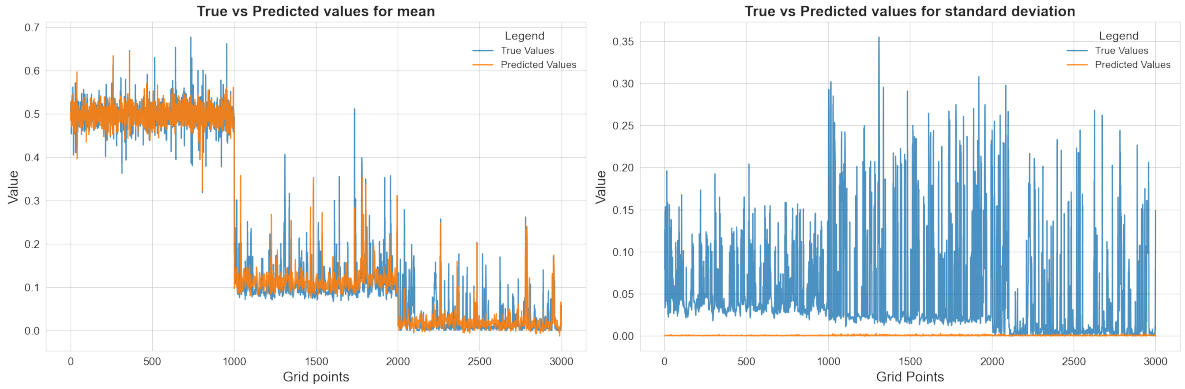

Figure 3.8: RNN Mean and Standard deviation Predictions for $t + \Delta$

RNN performs similarly to CNN in predicting mean but performs worse in predicting standard deviation, as shown in Figure 3.8. Looking at the standard deviation plot, it is clear that both models are performing poorly in predicting standard deviation. Let's look at predicted vs true values for each variable.

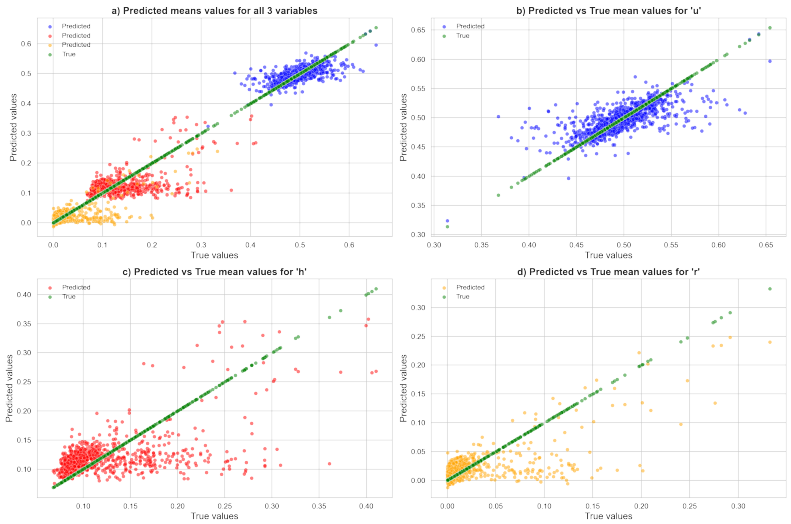

Figure 3.9: RNN Mean Predictions for each variable $t + \Delta$

Figure3.9 supports the outcome of figure 3.6, as the predicted values of mean for variable u are highly concentrated around the true values (green scatter plot). On the other hand, data points for variables h and r deviate significantly from the green line.

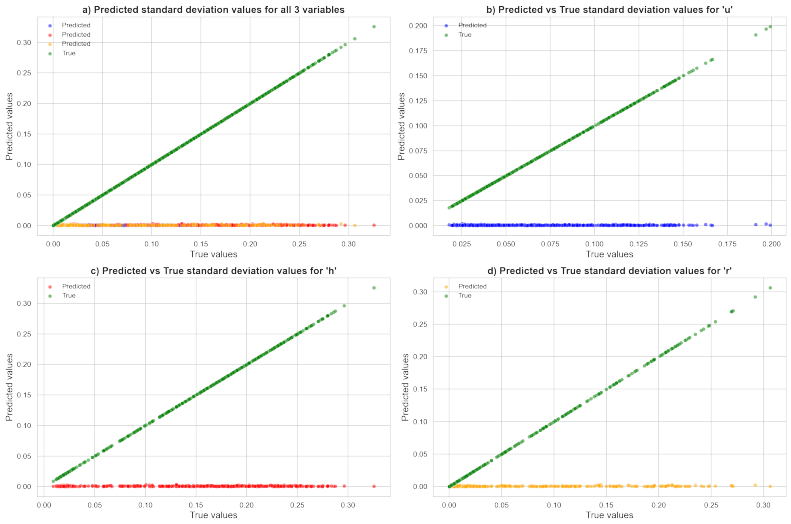

Figure 3.10: CNN Standard Deviation Predictions for each variable $t + \Delta$

Figure 3.10 is very similar to figure 3.7, and conclusions can be drawn that both CNN and RNN are worse in predicting standard deviation for $t + \Delta$. Now, let's investigate how the models perform for $t + 2\Delta$.

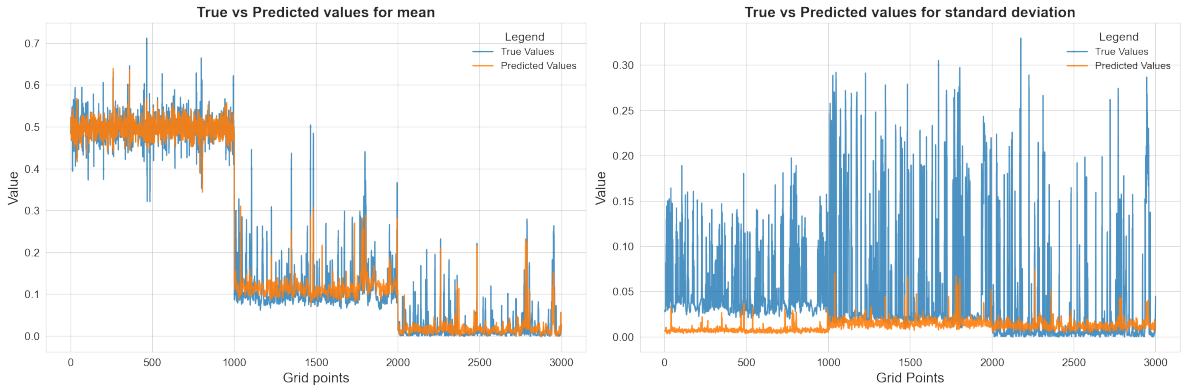

Figure 3.11: CNN Mean and Standard deviation Predictions for $t + 2\Delta$

Figure 3.11 represents the performance of CNN for both the mean and standard deviation. Evidently, CNN performs better for the mean predictions compared to the standard deviation for $t + 2\Delta$, which is also the case for $t + \Delta$. Let's look into detail how each variable behaves for mean and standard deviation.

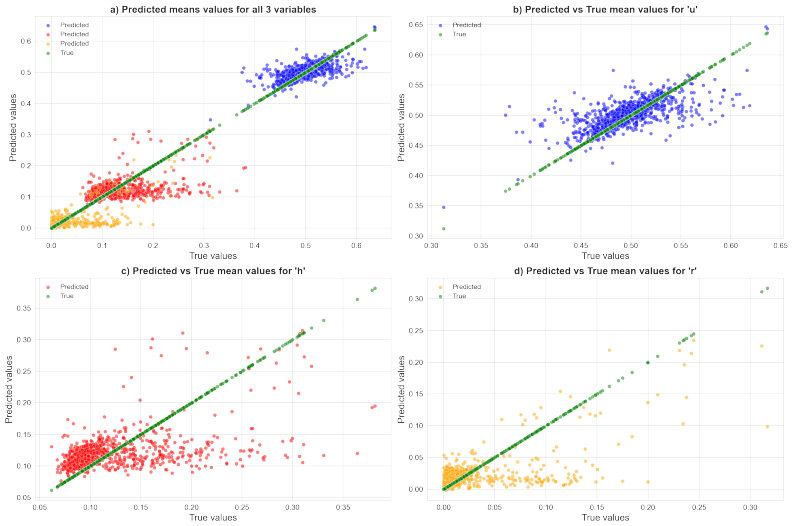

Figure 3.12: CNN Mean Predictions for each variable for $t + 2\Delta$

All the subplots in Figure 3.12 are very similar to those in Figure 3.6, supporting the conclusion that the mean predictions for variable u are more accurate than for variables h and r.

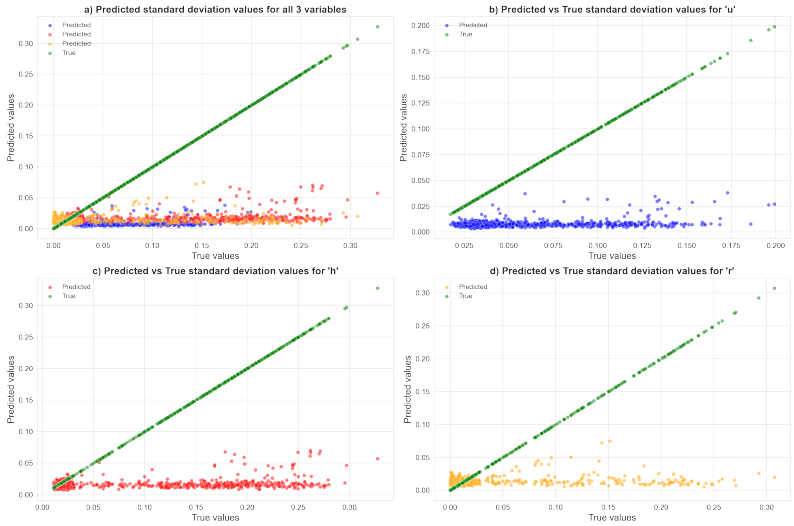

Figure 3.13: CNN Standard Deviation Predictions for each variable $t + 2\Delta$

The subplots in Figure 3.13 draw the same conclusion: CNN is not good at predicting standard deviation. However, a slight improvement in accuracy for standard deviation can be noticed compared to $t + \Delta$. Now, let's investigate how RNN responds for $t + 2\Delta$.

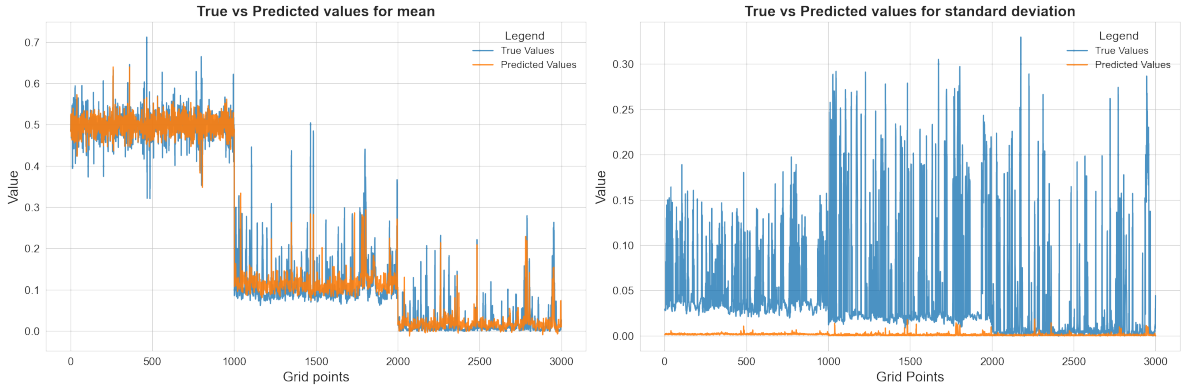

Figure 3.14: RNN Mean and Standard deviation Predictions for $t + 2\Delta$

Figure 3.14 suggests no significant difference in the performance of RNN between $t + \Delta$ and $t + 2\Delta$. Let's delve into specific details for each variable.

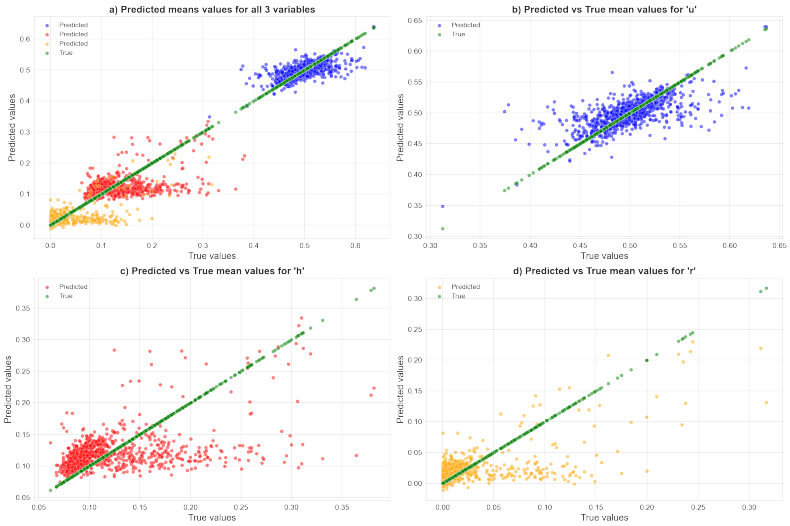

Figure 3.15: RNN Mean Predictions for each variable for $t + 2\Delta$

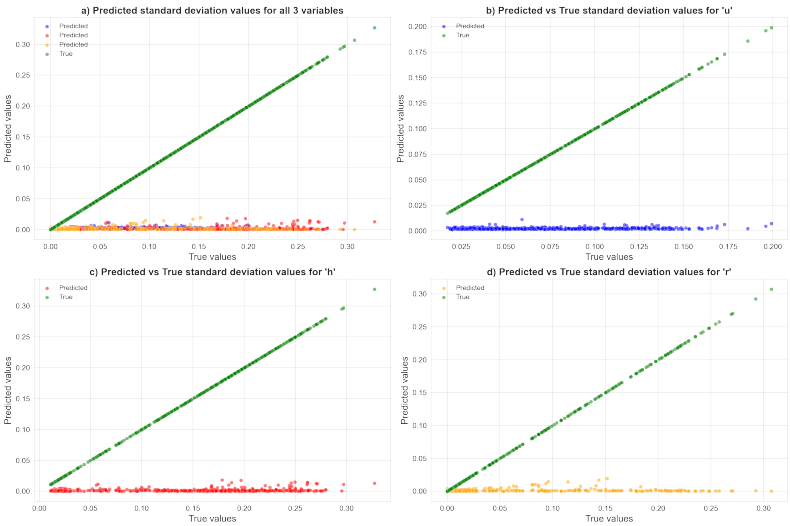

Figure 3.16: RNN Standard Deviation Predictions for each variable $t + 2\Delta$

Figures 3.15 and 3.16 also support the conclusion that mean predictions are more accurate than standard deviation predictions.

To evaluate both models, we have used Relative Absolute Mean Error (RAME) and Relative Absolute Standard Deviation Error (RASDE) to compare their performances by focusing on the errors of mean and standard deviation from the observations.

The formulas for RAME and RASDE are the same. The only difference is that while

evaluating for mean, we consider the mean observations and vice versa for standard deviation. The formula for both of them is as follows:

$$\text{RAME} = \sum_{i=1}^{n} \frac{|Y_i - \hat{Y}_i|}{|Y_i - \bar{Y}_i|} = \text{RASDE}$$

	CNN $t + \Delta$	CNN $t + 2\Delta$	RNN $t + \Delta$	RNN $t + \Delta$
RAME for u	0.8332	0.8148	0.6688	0.6907
RAME for h	0.8759	0.9395	0.9007	0.9494
RAME for r	0.8538	0.9077	0.9569	0.9711
RASDE for u	1.7815	1.7662	2.0641	1.9750
RASDE for h	0.8686	0.8486	1.0785	1.0553
RASDE for r	0.7969	0.8004	0.7877	0.7612

Table 3.1: Comparison of Models Based on Error Metrics

It is clear from the Table 3.1 that RNN performs better for variable u in predicting mean, while it struggles for variables h and r, for both $t + \Delta$ and $t + 2\Delta$. On the contrary, CNN performs slightly better for variables u and h in predicting standard deviation while struggling with variable r for both $t + \Delta$ and $t + 2\Delta$. Table 3.1 also supports the outcome of Figure 3.11 as the error for standard deviation decreases slightly for CNN in $t + 2\Delta$ compared to $t + \Delta$. It is important to take away from 3.1 that variable u is the hardest parameter to correctly predict for standard deviation.

4 Conclusion/Summary

This paper investigates and closes the gap in predicting statistical metrics on future time steps using machine learning models. Specifically, it examines how CNN and RNN can predict the mean and standard deviation of weather parameters such as wind velocity, cloud height, and rain in the shallow water model. This study shows that while RNN and CNN perform well in predicting mean values for both time steps($t+\Delta$ and $t+2\Delta$), they struggle to predict standard deviation, with CNN having slightly better accuracy. RNN struggles more since the algorithm works best with temporal sequential data. However, our data includes information on space (grid points) as well. This is one of the reasons it was under performing from its well-known ability to detect patterns. Additionally, we did not have a big dataset to leverage the specialty of RNN, which is LSTM, which requires a lot more data than we had.

5 Future Possible Work

Future work should focus on detecting why standard deviation is easier to predict further in the future and also on seeing what impact a custom loss function would have on the models. We used MSE as our loss function in our models, and this method focuses on the centrality of the data, which could be one of the reasons why standard deviation was hard to predict. Therefore, further studies should be done on these aspects to improve the accuracy of predicting both statistical metrics at future time steps.

Declaration

We hereby declare that we have composed this paper by ourselves and without any assistance other than the sources in our list of works cited. This paper has not been submitted previously or is currently being submitted to any other examination institution. It has not been published. All direct quotes and indirect quotes that in phrasing or original ideas have been taken from a different text (written or otherwise) have been marked as such clearly and in every single instance under a precise specification of the source. We understand that any false claim made here results in the failure of the examination.

Bibliography

[1] https://www.wri.org/insights/6-ways-climate-changed-over-past-decade, .

[2] https://link.springer.com/chapter/10.1007/978-1-935704-20-1_3, .

[3] https://github.com/wavestoweather/SWM_largeensemble, .

[4] https://link.springer.com/article/10.1007/s10710-017-9314-z, .

[5] https://link.springer.com/book/10.1007/978-3-030-89929-5, .