Isaac Pérez Anguiano - u214402
Marc Rodríguez Rodríguez - u214013

FINAL PROJECT:

TIME SERIES FORECASTING USING LSTM IN PYTORCH

Introduction

Accurate daily weather prediction is crucial for sectors such as agriculture, urban planning, and resource management, where decisions depend heavily on reliable forecasts. Traditional methods often struggle with capturing the nonlinear dynamics and long-term dependencies present in weather data.

In this project, we explore the use of Long Short-Term Memory (LSTM) networks for daily weather forecasting, taking advantage of their ability to model sequential data and learn temporal patterns across multiple variables. Our LSTM models, Implemented in PyTorch, are trained on multivariate time series data to predict key weather parameters.

We use a publicly available Kaggle dataset focused on a single location, which is Madrid (1997–2015), that includes weather measurements. Unlike classical approaches that explicitly model seasonality (like ARIMA or SARIMA), our approach relies on the LSTM's internal memory to learn cyclic patterns directly from the multivariate input data.

The project addresses challenges such as preparing complex multivariate datasets and designing effective LSTM architectures. Our objective is to evaluate whether LSTMs can offer strong accuracy in the context of daily weather forecasting.

State of the Art

Recent advances in time-series forecasting and weather prediction have demonstrated the benefits of combining stochastic seasonal modelling and deep-learning approaches:

- Stochastic Multi-Seasonal ARIMA, Ávila, Alonso & Peña (2025) [1]. The purpose of the project is to propose a flexible time series model using stochastic seasonal ARIMA to capture multiple seasonalities (daily, weekly, and yearly) in hourly NO_2 pollution data in Madrid. The approach aims to demonstrate its forecasting accuracy compared to other methods such as deep neural networks.

- Spatio-Temporal Stacked LSTM, Karevan & Suykens (2018) [2]. The objective of the project is to propose a two-layer spatiotemporal stacked LSTM model for weather forecasting that captures spatial relationships between locations. In the first layer, independent LSTM "streams" process each station's time series and their hidden states are concatenated to feed a second LSTM. The project is developed with data from stations in Belgium and the Netherlands.

Methodology

Data Analysis

The dataset used in this weather forecast project originates from the Kaggle file "weather_madrid_lemd_1997_2015.csv" [3], which contains daily meteorological records for Madrid between 1997 and 2015. Each entry includes some atmospheric variables such as maximum temperature, dew point, humidity, sea level pressure, minimum visibility, wind speed and direction, precipitation, cloud cover, and a free-text "Events" field indicating weather phenomena.

We performed the following preprocessing steps:

- Variable Selection: After computing the correlation of each meteorological variable with storm occurrences, we retained the six most predictive features: maximum temperature, mean dew point, mean humidity, mean sea level pressure, minimum visibility, and cloud cover. These were selected to capture the key atmospheric conditions associated with storm development.
- Event Label Simplification: Missing entries in the "Events" field were filled with "None." Any record whose description contained "Rain", "Thunderstorm" or "Snow" was relabeled simply as "Storm". The dataset was then filtered to keep only observations labeled "Storm" or "None." For binary classification, the "None" observations were relabeled as 0 and the "Storm" ones were labeled as 1.
- Chronological Train/Test Split: To prevent data leakage, the processed dataset was split chronologically. The first 80 % of the ordered samples were used for training, and the remaining 20 % for testing. This approach ensures that the LSTM model is trained on past observations and evaluated on truly future data, reflecting a realistic forecasting scenario.
- Class Balancing: To address the imbalance between storm and non-storm days, we oversampled the minority "Storm" class with replacement until it reached a similar number of "None" instances in the training set.
- Feature Scaling: The minimum visibility feature, originally capped at 10 km prior to 2012, was normalized using min-max scaling to correct measurement limits while preserving relative differences. This scaled visibility column was then recombined with the other five raw features to form the final input matrix.

Models and Optimization

This section outlines the four core components of our storm prediction system: (1) Model Architecture, describing the LSTM design; (2) Training Procedure, detailing the optimizer, loss function, and convergence monitoring; (3) Evaluation Protocol and Metrics, explaining how we assess model performance; and (4) Key Hyperparameters, summarizing the configuration used to control model capacity.

1. Model Architecture: Our sequence classifier is implemented in PyTorch as a LSTM network tailored for binary classification of weather events. Each daily input vector, containing six features (maximum temperature, minimum dew point, mean humidity, mean sea-level pressure, minimum visibility and cloud cover), is passed into the LSTM, which maintains internal memory states to capture temporal dependencies over time. The final hidden state is fed into a fully connected layer and activated through a sigmoid function to output a probability estimate of a storm occurrence.
2. Training Procedure: Supervised training is conducted in full-batch mode over 5 000 epochs. We employ the Adam optimizer with a learning rate of 0.001, chosen for its adaptive step size and effectiveness in recurrent architectures. To ensure reproducibility, both CPU and GPU random seeds are fixed to seed 42. At every epoch, a forward pass computes the binary cross-entropy loss, and backpropagation updates network parameters accordingly. The loss trajectory is recorded throughout to monitor convergence and detect potential signs of overfitting.
3. Evaluation and Metrics: After training, the model is switched to evaluation mode and tested on the 20% of the data, which preserves chronological ordering to simulate true forecasting conditions. No further weight updates occur during testing. We compute not only the final test loss but also classification metrics such as accuracy, precision, recall and F1 score by thresholding the sigmoid output at 0.5. To capture the model's discrimination power across all thresholds, we report the AUC-ROC. A confusion matrix summarizes the counts of true positives, false positives, true negatives and false negatives, offering insight into error types such as missed storms and false alarms.
4. Key Hyperparameters:
    - Input size: 6 features per time step
    - Hidden size: variable units in each of the LSTM layers
    - Number of layers: variable stacked LSTM layers
    - Optimizer: Adam with learning rate 0.001
    - Loss function: Binary cross-entropy (BCELoss)

These architectural and training decisions find a balance between model expressiveness and generalization, allowing the LSTM to learn complex temporal patterns from multivariate weather data and deliver strong performance in storm prediction.

Experiments

Initially, the model was trained using categorical cross-entropy. Later, we tried binary cross-entropy and we observed that, while overall accuracy remained similar, the recall significantly improved. Then, our decision was to use the binary approach since we give importance to the recall metric. That is because it measures the proportion of actual storm events correctly identified by the model or how often the model correctly predicts a storm when one truly occurs. From a practical standpoint, we consider it more important to issue a warning for a potentially dangerous weather event, even at the cost of occasionally generating false alarms.

In addition to recall, we also monitored other key performance metrics including loss, accuracy, precision, and Area Under the ROC Curve (AUC-ROC) to gain a more complete understanding of the model's behavior.

Results

In figure 1 below we show the results obtained in the testing set from modifying different hyperparameters of the model.

Hidden size	Number of layers	Test loss	Test accuracy	Test recall
32	1	0.4381	80.56%	68.51%
32	2	0.4265	81.32%	69.61%
32	4	0.4092	82.09%	70.72%
64	1	0.4203	81.99%	70.72%
64	2	0.4306	81.80%	73.48%
64	4	0.4315	82.18%	65.19%
128	1	0.4142	81.32%	72.38%
128	2	0.4287	82.18%	67.13%
128	4	0.4554	80.65%	78.73%

Figure 1: Table showing the test results of the model for different hyperparameters

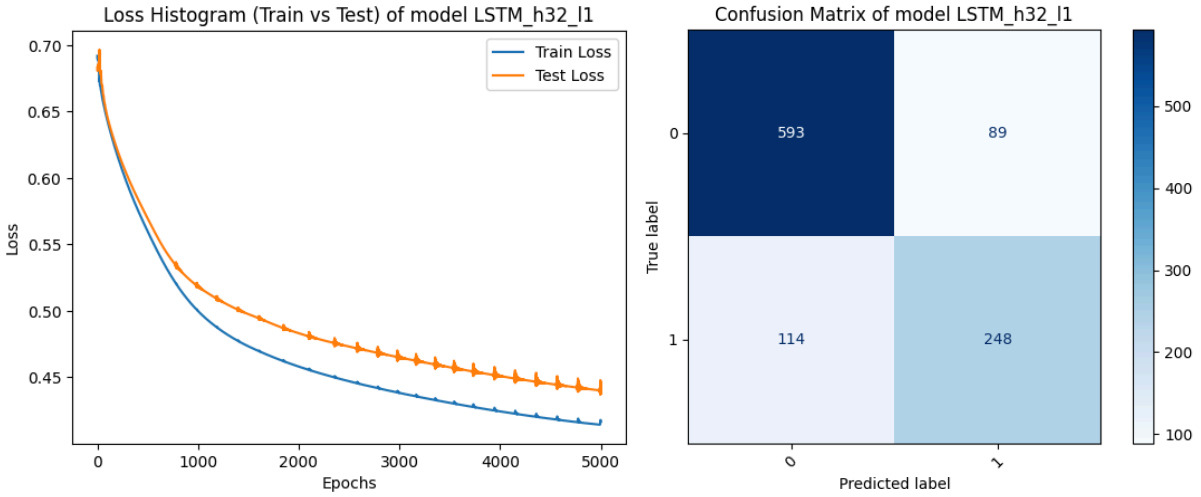

Figure 2: Training and test loss curves and confusion matrix of the LSTM_h32_l1

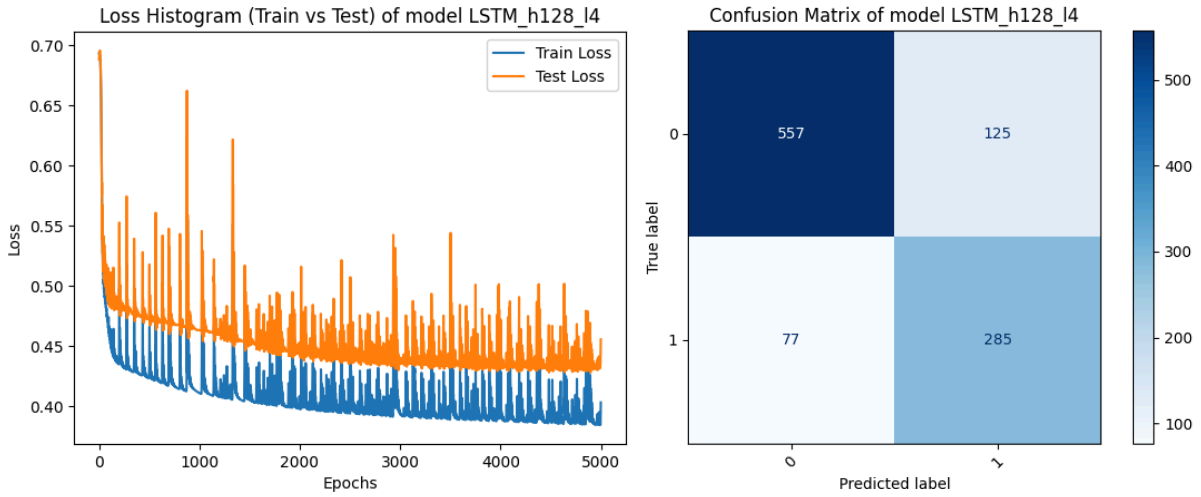

Figure 3: Training and test loss curves and confusion matrix of the LSTM_h32_l1

Discussion

Analyzing the results shown in the table, several conclusions can be drawn about the model's behavior based on the hidden layer size and the number of layers. In general, it is observed that a larger number of layers tends to improve the model's accuracy to a certain extent, although this does not always translate into a consistent improvement in loss or recall. For example, with a hidden layer size of 32, increasing the number of layers from 1 to 4 gradually improves both precision and recall, indicating that a deeper architecture can better capture the relevant features of the problem in this case.

However, this trend does not fully hold at larger hidden layer sizes. With a hidden layer size of 64, although precision also improves slightly with more layers, recall exhibits more erratic behavior. In particular, using two layers with a hidden layer size of 64 achieves one of the highest recall values (73.48%), but with four layers, this value drops abruptly to 65.19%. This suggests that greater depth is not always beneficial and may even impair the model's ability to correctly retrieve positive instances, perhaps due to overfitting or training difficulty.

With hidden size 128, an even more interesting behavior is observed: although the 4-layer configuration produces the worst loss (0.4554), it also generates the best recall of all experiments (78.73%). This could indicate that this model is very sensitive to the trade-off between precision and recall, prioritizing recall at the expense of overall precision. The lower precision in this case (80.65%) compared to other models with lower recall reinforces this idea.

Overall, the results suggest that there is no perfect balance of layer size and number of layers. Rather, performance depends on the prioritized evaluation criterion. Since our goal is to optimize recall, a model with a hidden size of 128 and 4 layers would be the best choice, although at the expense of greater test loss. It's also important to note that more complex models do not always perform better and may face limitations associated with overfitting or computational complexity. This highlights the importance of considering both the overall performance of the model and specific metrics relevant to the problem.

The loss history graph for the LSTM model with a hidden size of 128 and 4 layers (LSTM_h128_l4) reinforces some of the observations previously obtained from the table. This graph compares the training and test losses over 5,000 epochs, and we can find important trends that provide more information about the model's final performance.

First, the training loss decreases relatively steadily and consistently, indicating that the model is able to learn the patterns present in the training data well. However, the curve exhibits much unstable behavior, with noticeable fluctuations throughout training. The test curve follows a similar pattern compared with the training one, but with a higher mean value. This phenomenon suggests that the model is experiencing some overfitting and struggles to generalize.

The presence of pronounced spikes could indicate the presence of sequences that are particularly difficult to process or could be caused by numerical instabilities in prolonged training on deep architectures. This behavior helps explain why, despite having the highest recall capacity in the results table, this model has one of the worst performances in terms of

test loss. The model is able to capture a large number of true positives (hence the high recall), but at the cost of a large number of false positives that increase the overall loss and reduce accuracy.

When comparing the loss curve of the LSTM_h32_l1 model with that of the LSTM_h128_l4, we can see a high contrast in behavior. The LSTM_h32_l1 model shows a much smoother and more stable convergence for both training and test loss, with minimal fluctuations and a relatively narrow gap between the two curves. This indicates a well generalized model that learns consistently without much less overfitting, even after 5,000 epochs. On the other hand, while the LSTM_h128_l4 model achieves higher recall, its loss curve is far more erratic, with frequent spikes and a larger difference between training and test loss, which is a symptom of overfitting and potential instability during training. This comparison reinforces the idea that simpler architectures, like LSTM_h32_l1, offer more reliable generalization, while deeper models may prioritize recall at the cost of higher test loss and reduced stability.

Conclusions

This project demonstrated the effectiveness of LSTM networks for multivariate time series prediction in the context of daily weather forecasting. The models were able to capture complex temporal dependencies and provide robust results, particularly in storm identification. One of the most important findings is that properly configured and trained LSTM architectures are capable of learning cyclical and nonlinear patterns directly from weather data, without the need for explicit seasonal modeling.

We observed that increasing model complexity (through larger hidden sizes and deeper architectures) tended to improve recall. This reinforces the idea that in weather forecasting, especially for binary classification tasks such as storm forecasting, recall can be more important than overall accuracy. However, this improvement in recall often came at the cost of lost testing and generalization. The most complex model (hidden size 128, 4 layers) achieved the highest recall (78.73%), but also exhibited instability in the loss curve and a considerable difference between the training and testing loss, indicating overfitting.

This trade-off between model complexity and generalization ability emerged as a central challenge. While deeper models captured richer patterns, they were also more sensitive to noise and prone to overfitting. The loss curve of the most complex model clearly illustrated this problem, with high variance and spikes in both the training and the test loss.

Another key discovery was the crucial importance of data preparation. Steps such as feature selection, event relabeling, class balancing, and scaling were critical for the model to learn effectively. In particular, balancing the storm and non-storm classes helped the network focus on the minority class, significantly improving recall without modifying the model architecture.

For future work, several improvements could be made. First, exploring different activation functions such as ReLU or tanh could improve gradient flow and reduce training instability. Regularization techniques such as data dropout, weight decay, or early stopping should also be applied to mitigate overfitting in deeper models. Furthermore, evaluating the model with additional and more diverse datasets would help verify its robustness and ensure its generalization beyond the specific conditions of Madrid between 1997 and 2015.

In summary, LSTMs provide a solid foundation for weather event prediction, especially when prioritizing recall. However, achieving a balance between performance and generalization remains a key task, and future improvements should aim to reduce overfitting while maintaining the model's ability to detect relevant events in complex multivariate sequences.

References

[1] Avila, M., Alonso, A., & Peña, D. (2023). *Modelling multiple seasonalities with ARIMA: Forecasting Madrid NO_2 hourly pollution levels*. Stochastic Environmental Research and Risk Assessment. https://doi.org/10.1007/s00477-025-02958-6

[2] Karevan, Z., & Suykens, J. A. K. (2018). *Spatio-temporal stacked LSTM for temperature prediction in weather forecasting* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.1811.06341

[3] Simon, J. (2017). *Weather Madrid LEMD 1997–2015* [Dataset]. Kaggle. https://www.kaggle.com/datasets/juliansimon/weather_madrid_lemd_1997_2015.csv