Reliable Energy Grid Forecasting

Group 1 - Greg Reckenwald, Destinee Sheung, Andrew Golembeski, Matt Draugelis,
Nathan Adrian, Mohit Sharma

CS 613

September 1, 2025

Abstract

Accurate energy demand forecasting is essential for maintaining grid stability and operational efficiency, particularly as renewable energy adoption and climate variability introduce greater uncertainty into power systems. This study evaluates multiple machine learning approaches for forecasting energy grid metrics using four years of hourly data from Spain's five largest cities. We implemented and compared Linear Regression (LR), Autoregressive Integrated Moving Average (ARIMA), Feedforward Neural Network (FFNN), Recurrent Neural Network (RNN), and an ensemble model combining LR and RNN, evaluated using Root Mean Square Error (RMSE) and, for some, Symmetric Mean Absolute Percentage Error (SMAPE) metrics. Results demonstrate that no single model dominates across all forecasting targets: Linear Regression proved effective for long-term load forecasting but struggled with short-term fluctuations, ARIMA performed best on stable load data but poorly on variable renewable series, RNN excelled at solar generation forecasting by capturing diurnal cycles, and FFNN achieved superior wind generation predictions through non-linear feature modeling. The ensemble approach offers the most robust performance, but further tuning is required for our implementation. These findings suggest that complementary modeling approaches yield superior operational forecasting capabilities, demonstrating that there is no single model evaluated here which best forecasts over both short- and long-terms.

Contents

1 Introduction

In an era of dynamic energy markets and a growing emphasis on sustainability, the ability to accurately forecast energy demand and generation is more critical than ever. The increasing prevalence of variable renewable sources, coupled with unpredictable weather events, has made it difficult to find a single, universally optimal solution for prediction. Despite extensive prior research, existing models often yield divergent conclusions, underscoring the need for a comparative analysis of different forecasting methodologies. With this in mind, our study aims to address this challenge by implementing a diverse set of models: a Recurrent Neural Network (RNN), a Feed-Forward Neural Network (FFNN), a Linear Regression, and an ARIMA model. We then combine their value into an ensemble model.[1] By applying these distinct architectures and techniques to a comprehensive dataset from Spain, we seek to not only predict energy usage but also to evaluate our models' efficacy against current industry benchmarks and potentially surpass them.

2 Background and Related Work

The widespread use of artificial intelligence, along with factors such as climate change and rising electricity demand, places increasing strain on energy grids. Managing these grids optimally is challenging due to their dynamic components, including temperature, wind speed, and other environmental factors. The rise of renewable energy resources, combined

1. Robert T Clemen, "Combining forecasts: A review and annotated bibliography," *International journal of forecasting* 5, no. 4 (1989): 559–583.

with fluctuating demand and variable environmental conditions, further complicates grid management, as power output can change rapidly and unpredictably.[2]

Accurate load forecasting, which involves predicting future electricity demand based on historical data and influencing factors, has therefore become essential for maintaining grid stability and optimizing operations. Poor forecasting can have significant consequences: overproduction leads to wasted energy and financial losses, while underproduction risks power outages that disrupt both customers and businesses. By enabling grid operators to anticipate rapid changes in generation and periods of surplus or deficit, accurate forecasting improves the balance between supply and demand and improves overall grid stability and efficiency.[3] Consequently, the development and implementation of forecasting techniques have become a central focus in ensuring reliable and efficient energy grid operations.

Recent research has explored a variety of forecasting methods, reflecting the diversity of challenges and contexts in which they are applied. Tarmanini et al. compared the performance of an artificial neural network (ANN) and an ARIMA model across 700 Irish households, finding that the relative accuracy of each method varied depending on the forecasting horizon and the nature of the input data.[4] Similarly, Her et al. employed a trial-and-error approach to identify the most effective ANN configuration for short-term load forecasting.[5]

2. B. Brailey, *Transforming grid operations with accurate short-term energy predictions*, DNV, 2024, https://www.dnv.com/article/transforming-grid-operations-with-accurate-short-term-energy-predictions/.

3. Brailey.

4. C. Tarmanini et al., “Short term load forecasting based on ARIMA and ANN approaches,” *Energy Reports* 9 (2023): 550–557, https://doi.org/10.1016/j.egyr.2023.01.060.

5. O. Y. Her et al., “Artificial neural network based short term electrical load forecasting,” *International Journal of Power Electronics and Drive Systems / International Journal of Electrical and Computer Engineering* 13, no. 1 (2022): 586, https://doi.org/10.11591/ijpeds.v13.i1.pp586-593.

In contrast, Misiurek et al. provided a broad review of forecasting methods, emphasizing machine learning techniques and analyzing how different models incorporate external factors such as weather patterns and seasonality.[6]

Together, these studies highlight that while ANNs, ARIMA models, and other machine learning approaches have demonstrated strong potential, no single method consistently outperforms others across all forecasting scenarios. The effectiveness of a given technique often depends on the specific context such as time horizon, scale of analysis, and environmental underscoring the need for continued experimentation and hybrid approaches in load forecasting research.

3 Data

3.1 Data Overview

The dataset comprises two primary sources: an energy dataset and a weather dataset, covering four years of data from Spain's five largest cities: Barcelona, Bilbao, Madrid, Seville, and Valencia. The energy dataset includes 29 columns capturing hourly electrical consumption, generation, pricing, and related metrics. Key features include a timestamp column (recorded hourly) and continuous measurements for various power generation sources, such as biomass, fossil gas, fossil coal, hydro, nuclear, and others. All columns in the energy dataset are continuous.

6. K. Misiurek, T. Olkuski, and J. Zyśk, "Review of Methods and Models for Forecasting Electricity Consumption," *Energies* 18, no. 15 (2025): 4032, https://doi.org/10.3390/en18154032.

The weather dataset consists of 17 columns, including an hourly timestamp, a categorical column specifying the city, and continuous weather measurements (e.g., temperature, humidity). The city name, a nominal categorical variable, is the only non-continuous feature in this dataset.

Datasets can be found @ kaggle.com here:

https://www.kaggle.com/datasets/nicholasjhana/energy-consumption-generation-prices-and-weather

3.2 Data Merging

To integrate the energy and weather datasets, we merged them based on the timestamp column. This process presented challenges due to structural differences. The energy dataset provides hourly aggregate values for consumption and generation across all five cities, while the weather dataset contains separate rows for each city's hourly weather measurements, resulting in five rows per timestamp. To align the datasets, we transformed the weather data by grouping the five rows per hour into a single row, with columns renamed to reflect city-specific measurements (e.g., the `temp` column was renamed to `temp_Barcelona`, `temp_Bilbao`, etc., for each weather variable and city). This restructuring enabled a straightforward merge with the energy dataset on the timestamp column.

3.3 Timestamp Processing

The original timestamp column, which combined date and time, was not suitable for modeling in its default format. We deconstructed it into four separate columns (year, month, day, and hour) to capture the date and time patterns more effectively. The original timestamp column was then dropped.

3.4 Handling Missing Data

In the energy dataset, eight of the 29 columns were either empty or contained only zeros and were removed. After merging the datasets, we identified fewer than 20 rows in the weather dataset with timestamps absent from the energy dataset, resulting in null values for energy measurements. These rows were removed to ensure data consistency.

3.5 Removal of Irrelevant Features

Several columns were deemed irrelevant for training the machine learning models. In the energy dataset, columns containing energy companies' forecasted results were dropped, though retained for later comparison with our model's predictions. Pricing data was also removed, as it was not relevant to the model's objectives. In the weather dataset, four redundant columns (`weather_id`, `weather_main`, `weather_description`, and `weather_icon`) were dropped, as they duplicated information captured in other weather features.

3.6 Normalization

All continuous-valued columns, except the timestamp-derived columns (year, month, day, hour) and the target variable, were normalized using z-scoring. For each continuous feature, we computed the z-score as follows:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

where x is the feature value, μ is the mean, and σ is the standard deviation. This standardization ensures that features with different scales contribute equally to the model.

3.7 Final Dataset

After preprocessing, the merged dataset consisted of 72 columns: 71 features and 1 target variable. Of the 71 features, 67 were z-scored continuous variables, and 4 (year, month, day, and hour) remained unnormalized. This final dataset was used for training and evaluating the machine learning model.

4 Models & Results

4.1 Linear Regression Model

A Linear Regression is a model which describes the linear relationship between variables in order to predict a continuous outcome. It follows the classic slope-intercept formula, $y = mx + b$, where in this case:

- y is the predicted value of this point.

- m is the estimated slope based on the data, derived from the line of best fit.
- b is the bias factor, which acts as the y-intercept.

In a case with multiple features such as this, m is determined by the weighting of the various features. This means that, in this case, we extend the formula to the form $y = w_1x_1 + w_2x_2 + ... + w_Fx_F + b$, where w_i is the weight applied to feature x_i.

In order to determine the weight for each feature, we can use the direct least squares formula $w = (X^TX)^{-1}X^Ty$, where:

- w is a column vector containing the weights for each feature in order.
- X is the feature matrix for our training data.
- X^T is the transpose of said feature matrix.
- y is a column vector containing the true labels of our training data.

The resulting column vector w can then be multiplied by any set of features which follows this data structure, resulting in a y label prediction for said feature set. Since we're condensing multiple features into a single prediction stat, we can also think of this as a projection into the answer space. We can also fold the bias into our data by appending an additional feature to each feature set, containing 1 in all instances.

Once we've run these calculations on our training data, we can store the weight vector w as our model, which is able to make a series of predictions by simply following the formula $\hat{y} = Xw$, where:

- $\hat{y}$ is an array containing our predictions for each feature set.
- X is our feature matrix containing the feature sets we'd like to predict.
- w is our weight vector from the prior calculation.

4.1.1 Linear Regression Tuning

Due to the rather simplistic nature of Linear Regression, the primary method of tuning is through the addition and removal of features until finding a feature set which minimizes the error between the prediction and the true results. Our goal was to predict three different values: Grid Load, Solar Generation, and Wind Generation. Each of these used a different linear regression model, though all three have a similar set of excluded and included features. In the case of all three models, 20 of the 71 features available were dropped. These included the 14 features with names starting with "generation" (of which "generation solar" and "generation wind onshore" were the labels for the solar and wind forecasts respectively), "total load actual" (which was the label for the load forecast), and the "temp" data per city. Additionally, for every model, a "bias" feature as described previously was added. For the solar and wind models, the removal of these 20 features proved the most effective. However, the load forecast additionally removed the "wind_deg" data per city to achieve the best results.

4.1.2 Linear Regression Evaluation

The table below shows the RMSE and SMAPE values when comparing the predictions by the model vs. the true label at the given point.

Further, in cases of Grid Load prediction, Linear Regression is expected to be better with long-term predictions than short-term. We expect to see the opposite for Solar and Wind predictions, though, since weather notoriously becomes less predictable the further in the future you go. Since Solar and Wind generation totals are inherently tied to weather

	TRAINING	VALIDATION
RMSE(Load)	3852.322	3917.860
SMAPE(Load)	0.055	0.056
RMSE(Solar)	1210.093	1210.571
SMAPE(Solar)	0.506	0.503
RMSE(Wind)	2884.674	2905.937
SMAPE(Wind)	0.224	0.227

Table 1: RMSE and SMAPE of Training vs. Validation Sets

conditions, this means they too should become less predictable by proxy. If we retrain the model, keeping the data in order, we can see if these are true based on the first 30 days of the validation set vs. the last 30 days. The table below shows the resulting RMSE and SMAPE Values.

	First 30 Days	Last 30 Days	Full Training Set	Full Validation Set
RMSE(Load)	3110.341	2179.782	3792.061	4319.241
SMAPE(Load)	0.045	0.035	0.054	0.059
RMSE(Solar)	989.164	1095.299	1184.506	1397.652
SMAPE(Solar)	0.435	0.763	0.488	0.615
RMSE(Wind)	1212.193	1787.407	2869.708	2984.632
SMAPE(Wind)	0.116	0.200	0.223	0.235

Table 2: RMSE and SMAPE of First 30 vs. Last 30 days

As we can see, some general accuracy has been lost, since we're no longer randomizing our data, but we can see that the predictions later in time line up exactly as expected, with higher accuracy for grid load, and lower accuracy for solar and wind.

4.1.3 Linear Regression Consistency

To ensure the model maintained consistency, it was put through various runs of cross-validation using different amounts of folds. Below is a table showing the resulting Average and Standard Deviation of the RMSEs for all three predicted values. These were run after reintroducing the randomization, so we'd like to see values similar to those in the first table.

	10 Folds	100 Folds	1000 Folds
μ RMSE(Load)	3878.310	3878.066	3878.031
σ RMSE(Load)	0.297	0.106	0.038
μ RMSE(Solar)	1211.733	1211.634	1211.625
σ RMSE(Solar)	0.140	0.027	0.012
μ RMSE(Wind)	2895.508	2895.167	2895.119
σ RMSE(Wind)	0.344	0.113	0.039

Table 3: Mean and Standard Deviation of RMSEs in Cross Validation

As we can see, the predictions for grid load fall comfortably within the range we expect between 3852.322 and 3917.860. Wind is also comfortably within the expected range between 2884.674 and 2905.937. The solar predictions landed very slightly higher than the original range of 1210.093 to 1210.571, but still well within a reasonable distance. As such, it's reasonable to conclude that our model remains consistent through different inputs

4.1.4 Linear Regression Conclusions

The Linear Regression performs reasonably well in far off predictions for load forecasting, given its RMSE of 2179.782 and corresponding SMAPE of 3.5% in the last 30 days of the dataset. This suggests that the model is relatively effective at capturing long-term trends over a dataset, suggesting it's well prepared to account for factors such as seasonal differences and time of day.

However, it also demonstrates some limitations. The load forecast's poorer performance on days which are closer suggests that it is less equipped to handle in-the-moment variables which may impact power generation needs, such as sudden shifts in a storm's path.

This would explain why the load forecast accuracy improves the further you get from the end of the training dataset, because the further you get from the in-the-moment factors, the more the general trends become pronounced.

The Linear Regression also performs pretty well on Solar and Wind predictions. Unlike the Load predictions, they do get worse the further from the end of the training date you go, but the actual RMSE's are quite low. Solar is perhaps a bit concerning, since it has a SMAPE of 76.3% in the last 30 days, meaning it has very high variance. The RMSE isn't nearly as drastic of a leap, but such high variance does leave significant doubt about it's reliability at that distance from the training data. For reference, the time between the first 30 days vs. the last 30 spans more than a year, and it should be obvious that accurately predicting weather that far off is beyond impossible.

Wind provided significantly less variance, though its RMSE did increase more significantly. However, it still manages a rather impressive accuracy despite this, and it appears relatively reliable even so far out. This is likely in part due to wind having somewhat fewer inhibiting factors than solar. For instance, solar contends directly with clouds, while wind doesn't particularly care if the sky is clear or overcast.

As explained, these two being so heavily tied to weather would explain why they become harder to predict the further you go, and the data does bear that out. It's impressive that the wind predictions still manage a respectable RMSE despite this.

4.2 ARIMA Model

4.2.1 Setup

The Autoregressive Integrated Moving Average (ARIMA) model is a time series forecasting technique that uses univariate historical data to generate short-term predictions. An ARIMA model is typically denoted as ARIMA(p, d, q), where:

- p represents the number of autoregressive (AR) terms,
- d indicates the degree of differencing
- q specifies the number of moving average (MA) terms.

The integrated component $I(d)$ refers to differencing the data d times until it becomes stationary.

Differencing is applied to transform nonstationary data, which have patterns such as mean, variance, and autocorrelation that change over time, into stationary data, where these properties remain stable. Mathematically, the first difference of a time series $\{y_t\}$ is defined as:

$$\Delta y_t = y_t - y_{t-1},$$

and the d-th difference is obtained recursively as:

$$\Delta^d y_t = \Delta^{d-1} y_t - \Delta^{d-1} y_{t-1}, \quad \text{with } \Delta^0 y_t = y_t.$$

Here, y_t is the observed value at time t, $\Delta^d y_t$ represents the series after d differences, and $\Delta^0 y_t$ is the original series. Each differencing step removes trends to achieve stationarity, which is required before fitting the AR and MA components of the ARIMA model.

Stationarity can be assessed visually using a time series plot, where consistent mean and variance over time suggest stationarity, or statistically using the Augmented Dickey-Fuller (ADF) test, where a low p-value indicates that the data is likely stationary. Some datasets do not require differencing as they are already stationary, making $d = 0$. Once the data is stationary, the model uses autocorrelations and moving averages over residual errors to predict future values.[7] With the data prepared and stationarity ensured, ARIMA models can then be implemented to generate forecasts, requiring careful selection of the p, d, and q parameters for optimal performance.

Before fitting the model to these parameters, the data is split into training and validation sets as with other models. However, because the data is a time series, the observations cannot be randomized, and the chronological order must be preserved. In this project, the first two-thirds of the data were used for training, while the remaining one-third was reserved for validation. The calculations for AR (p) and MA (q) are performed on the differenced dataset if differencing is applied.

The autoregressive component, AR(p) predicts the current value of the time series based on its previous p observations known as lagged values. Lagged values are past values of the series that the model uses as predictors, and their weights (coefficients ϕ_1,..., ϕ_p are learned from the training data during model fitting. The AR(p) component can be expressed

7. N. Bora, *Understanding ARIMA models for machine learning*, Capital One, 2021, https://www.capitalone.com/tech/machine-learning/understanding-arima-models/.

as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon_t$$

where y_t is the observed value at time t, $y_{t-1}, ..., y_{t-p}$ are the lagged values and $\phi_1, ..., \phi_p$ are coefficients learned from the training data during model fitting, c is a constant (omitted if the series has zero mean), and ϵ_t represents white noise. ϕ is solved using least squares on the training data. Once the AR coefficients are calculated, the residuals (the differences between the observed and values predicted by the AR part can be calculated. These residuals represent the errors the AR model could not explain.

These residuals are then used to fit the moving average component, MA(q), which models the dependency of the current value on past forecast errors. The MA (q) component is expressed as:

$$y_t = c + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q} + \epsilon_t$$

where $\epsilon_{t-1}, ..., \epsilon_{t-q}$ are residuals from previous predictions, and $\theta_1, ..., \theta_q$ are coefficients that determine the influence of past errors on the current value. The constant c can be omitted if the data is mean-centered. Using the residuals from the AR model, the MA coefficients are estimated via least squares. Together, the AR and MA components enable the ARIMA model to capture patterns in both past observations and past forecast errors. To evaluate the model fit and help select the optimal (p, d, q) parameters, the Akaike Information Criterion (AIC) is calculated.

$$AIC = 2k - 2ln(L)$$

K is the number of estimated parameters (ϕ, θ and the constant) and L is the likelihood of the model given the data. L can be computed using the Gaussian log-likelihood equation:

$$ln(L) = -\frac{n}{2}ln(2\pi) + ln(\sigma^2) + 1$$

where σ^2 2 is the variance of the residuals. Lower AIC indicates a better balance between model fit and complexity.

Once the AR coefficients ϕ, MA coefficients θ, and residuals are estimated from the training data, these parameters are applied directly to the validation set to generate forecasts without recalculating them. The process begins by differencing the validation series if needed, then iteratively predicting each step ahead. At each forecast step, the predicted value is a combination of the AR contribution, calculated from previous lagged values using the training-set ϕ, and the MA contribution, calculated from past forecast errors using the training-set θ. This iterative procedure produces forecasts for the validation set, which are then compared to the actual values to evaluate model performance.

4.2.2 Results & Figures

Table 4: ARIMA Models Used for Each Series

Series	p	d	q
Load	2	0	3
Wind	1	1	1
Solar	2	0	2

The original dataset was collected at an hourly frequency. To reduce high-frequency noise and minimize sharp peaks and valleys, the series was aggregated to a daily frequency by computing the mean of the hourly values for each day. Each timestamp corresponds to one day of aggregated data. Aggregating to daily averages also helps to handle short-term volatility, which ARIMA models find challenging due to their linear and univariate nature, making them less capable of capturing sudden spikes or nonlinear patterns of energy-related data. The forecast plots show a 30-timestep horizon (1 timestep = 1 day) because this represents the optimal short-term performance of the ARIMA models. Longer horizons result in larger accumulated errors and less reliable predictions, particularly for highly variable series such as wind and solar generation.

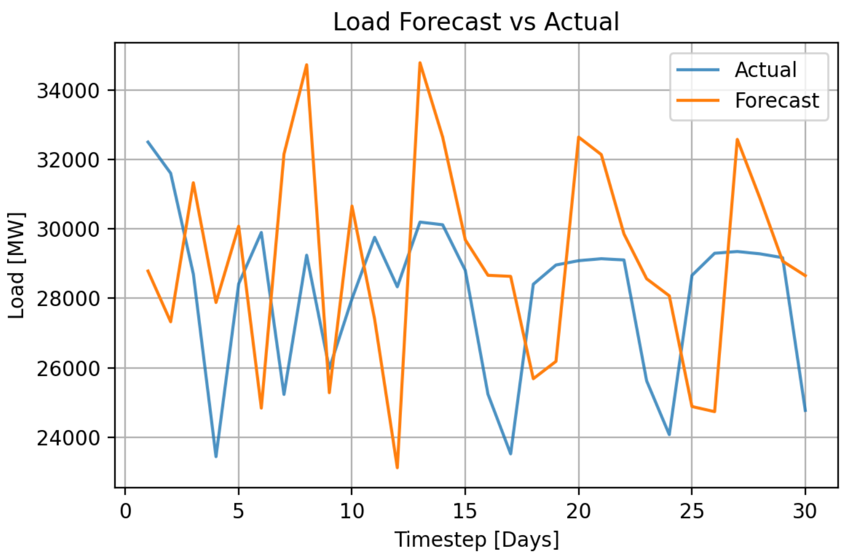

Figure 1: Forecast vs Actual Load (First 30 Timesteps; 1 timestep = 1 day).

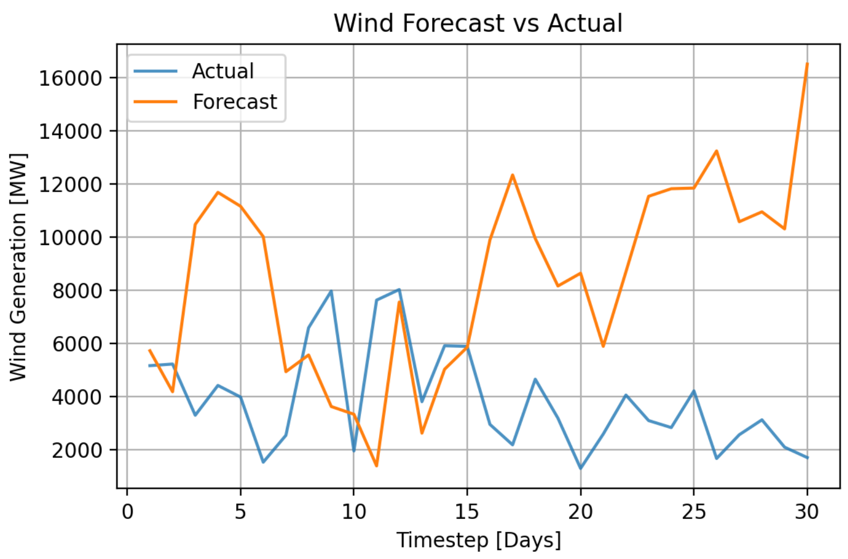

Figure 2: Forecast vs Actual Wind Generation (First 30 Timesteps; 1 timestep = 1 day).

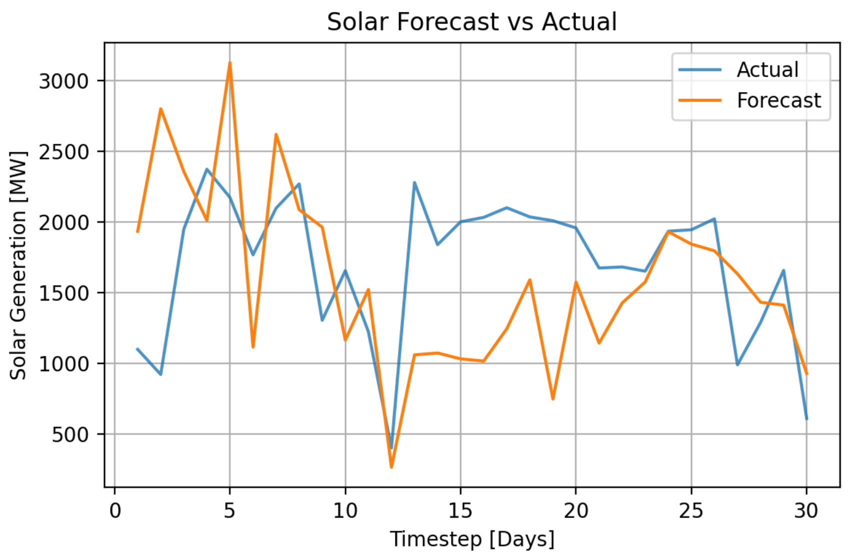

Figure 3: Forecast vs Actual Solar Generation (First 30 Timesteps; 1 timestep = 1 day).

The plots reveal that ARIMA performs best for load, which is smoother and more predictable. Wind shows high variability and large spikes, which are not captured well by the linear ARIMA model. Solar is intermediate, showing some predictability but struggling with sharp changes.

Table 5: Forecast Performance Metrics (1 timestep = 1 day)

Series	Dataset	RMSE	SMAPE (%)
Load	30-timestep Validation	3641.58	11.61
Load	Training	3111.06	9.15
Wind	30-timestep Validation	6697.15	86.69
Wind	Training	2121.97	24.39
Solar	30-timestep Validation	699.04	35.91
Solar	Training	694.33	47.25
Load	Entire Validation	4116.93	11.69
Load	Training	2262.91	6.39
Wind	Entire Validation	3962.98	59.22
Wind	Training	2278.96	40.28
Solar	Entire Validation	856.59	54.99
Solar	Training	525.32	37.85

The ARIMA(2,0,3) model for load forecasting performs reasonably well in the short term, achieving a 30-timestep validation RMSE of 3642 MW and SMAPE of 11.6%. However, over the entire validation set, errors increase (RMSE 4117 MW, SMAPE 11.7%), indicating that the model is better at short-term prediction than capturing long-term. However, since the SMAPE values are quite close, the model is relatively consistent in terms of predictive performance even for long-term forecasts."

Wind forecasting with ARIMA(1,1,1) is particularly challenging due to the high variability of wind generation. The model shows a large discrepancy between training and 30-day validation performance (SMAPE 24.4% vs. 86.7%), highlighting poor short-term generalization. Although performance improves somewhat over the full validation set (SMAPE 59.2%), it remains worse than load or solar forecasting. The RMSE over the entire validation set appears lower than for the 30-day forecast, but this masks substantial day-to-day prediction errors. SMAPE confirms that the model's forecasts are still highly inaccurate, underscoring the importance of evaluating ARIMA performance on shorter-term horizons. The difference between training and validation errors, particularly for wind, suggests some overfitting on the training data.

The ARIMA(2,0,2) model for solar performs reasonably well in the first 30 timesteps (RMSE $\approx$ 700 MW), but accuracy decreases over the entire set (SMAPE increasing from 35.9% to 55.0%). This suggests ARIMA captures short-term solar forecasts but struggles with seasonal or nonlinear effects.

Overall, ARIMA models show stronger performance on stable load data and weaker results on highly variable renewable series, particularly wind. These results highlight ARIMA's limitations: it is univariate, assumes linearity and stationarity, and does not account for external factors such as weather or wind speed, which contributes to its poor performance on long-term forecasts. Additionally, the model's performance depends on the selection of hyperparameters p, d, and q; tuning these parameters can improve short-term forecasts but also increases model complexity and the risk of overfitting.

4.3 Recurrent Neural Network

4.3.1 Setup

We implemented a basic recurrent neural network (RNN) for time series forecasting[8] with certain training decisions. The network consists of three parameter matrices: an input-to-hidden weight matrix $\mathbf{U} \in \mathbb{R}^{h\times 2}$, a hidden-to-hidden recurrent weight matrix $\mathbf{W} \in \mathbb{R}^{h\times h}$, and a hidden-to-output weight matrix $\mathbf{V} \in \mathbb{R}^{1\times(h+1)}$, where h is the hidden dimension and a bias term is incorporated.

For a sequence of length T, the forward pass calculates hidden states and final prediction as follows: The input at timestep t is augmented with a bias term:

$$\mathbf{x}_t = \begin{bmatrix} x_t \\ 1 \end{bmatrix} \in \mathbb{R}^{2\times 1} \tag{2}$$

Then the hidden state at timestep $t+1$ is computed using the hyperbolic tangent activation function:

$$\mathbf{s}_{t+1} = \tanh(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{s}_t) \tag{3}$$

8. Robin M Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," *arXiv preprint arXiv:1912.05911*, 2019,

where $\mathbf{s}_0 = \mathbf{0}$ is the initial hidden state. After that, the final hidden state $\mathbf{s}_T$ is augmented with a bias term for the output layer:

$$\tilde{\mathbf{s}}_T = \begin{bmatrix} \mathbf{s}_T \\ 1 \end{bmatrix} \in \mathbb{R}^{(h+1)\times 1} \tag{4}$$

and finally the predicted output is computed as:

$$\hat{y} = \mathbf{V}\tilde{\mathbf{s}}_T \tag{5}$$

The model is trained using the mean squared error loss function:

$$\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2 \tag{6}$$

where y is the target value and $\hat{y}$ is the predicted value. The gradients are computed using backpropagation through time (BPTT) with no truncation. The error signal is first computed as:

$$\delta_{\text{out}} = \hat{y} - y \tag{7}$$

Then, the gradient with respect to the output weights is calculated:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{V}} = \delta_{\text{out}}\tilde{\mathbf{s}}_T^T \tag{8}$$

After that, the error is then backpropagated to the final hidden state:

$$\boldsymbol{\delta}_T = \mathbf{V}^T \delta_{\text{out}} \odot (1 - \mathbf{s}_T^2) \tag{9}$$

where $\odot$ denotes element-wise multiplication and $(1 - \mathbf{s}_T^2)$ is the derivative of the tanh activation function. For timesteps $t = T, T-1, \ldots, 0$, the gradients are then iteratively computed as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{U}} \mathrel{+}= \boldsymbol{\delta}_t \mathbf{x}_{t-1}^T \tag{10}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} \mathrel{+}= \boldsymbol{\delta}_t \mathbf{s}_{t-1}^T \tag{11}$$

where $\frac{\partial \mathcal{L}}{\partial \mathbf{U}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{U}}$ are initially filled with elements equal to 0, and $\boldsymbol{\delta}_t$ is propagated backwards in time via:

$$\boldsymbol{\delta}_{t-1} = (\mathbf{W}^T \boldsymbol{\delta}_t) \odot (1 - \mathbf{s}_{t-1}^2) \tag{12}$$

Once the gradients are computed, the model parameters are finally updated using stochastic gradient descent (SGD) with momentum. Specifically, the parameter updates follow:

$$\mathbf{v}_{\mathbf{U}}^{(t+1)} = \mu \mathbf{v}_{\mathbf{U}}^{(t)} - \alpha^{(t)} \frac{\partial \mathcal{L}}{\partial \mathbf{U}} \tag{13}$$

$$\mathbf{v}_{\mathbf{W}}^{(t+1)} = \mu \mathbf{v}_{\mathbf{W}}^{(t)} - \alpha^{(t)} \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \tag{14}$$

$$\mathbf{v}_{\mathbf{V}}^{(t+1)} = \mu \mathbf{v}_{\mathbf{V}}^{(t)} - \alpha^{(t)} \frac{\partial \mathcal{L}}{\partial \mathbf{V}} \tag{15}$$

$$\mathbf{U}^{(t+1)} = \mathbf{U}^{(t)} + \mathbf{v}_{\mathbf{U}}^{(t+1)} \tag{16}$$

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} + \mathbf{v}_{\mathbf{W}}^{(t+1)} \tag{17}$$

$$\mathbf{V}^{(t+1)} = \mathbf{V}^{(t)} + \mathbf{v}_{\mathbf{V}}^{(t+1)} \tag{18}$$

where $\mu = 0.9$ is the momentum coefficient and $\mathbf{v}$ represents the velocity, initialized at zero.

We found that incorporating a learning rate schedule improved training, especially in later epochs. The learning rate follows an exponential decay schedule:

$$\alpha^{(t)} = \alpha_0 \gamma^{\lfloor t/s \rfloor} \tag{19}$$

where α_0 is the initial learning rate, γ is the decay factor, s is the step size for decay, and $\lfloor x \rfloor$ represents the floor function for some argument x. We trained our RNN model for 250 epochs and take the model parameters which best minimized the validation loss. From rough experimentation, we found the following hyperparameters worked best for our RNN implementation:

- Hidden dimension: $h = 64$
- Sequence length: $T = 32$
- Momentum coefficient: $\mu = 0.9$

- Initial learning rate: $\alpha_0 = 10^{-3}$
- Learning rate decay: $\gamma = 0.75$ every 25 epochs

4.3.2 Results & Figures

We first applied the recurrent neural network (RNN) to a small subset of the entire grid load data to gauge what the sequence length T should equal (Fig. 4). This subset consisted of 10 days of data taken approximately 8 days after New Year's. We can see that the grid load cycles approximately every 24 hours, and so therefore the sequence length should be at least 24. We can also see there are signs of a weekly effect which is reasonable given standard working hours. Therefore, setting T to be greater than 168 would allow the RNN to capture this effect, but in order to keep the model computationally tractable we compromised and set $T = 36$. We similarly tried different values for h and found $h = 64$ to be tractable and effective.

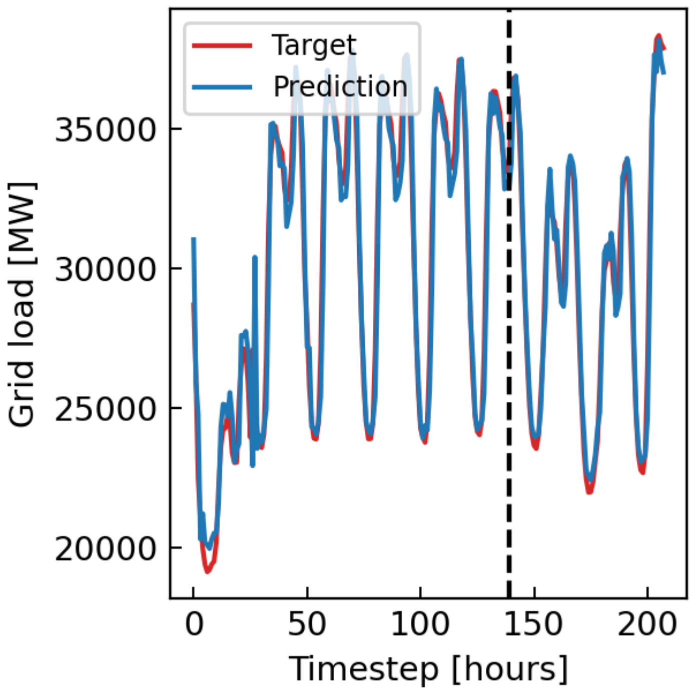

Figure 4: Grid load in megawatts versus timestep in hours. Shown are the target values (i.e. the actual grid load) and the values predicted by the RNN. The dashed vertical line corresponds to the training and validation split, where validation data is to the right of the line. The RMSE for this comparison is 784.83 compared to the company's RMSE of 417.80.

From Fig. 4, we can also see that our RNN's RMSE is about a factor of 2 worse than the company's prediction. This is likely due to the low number of recurrent events that the RNN is trained on, and when trained using the entire dataset we found the RMSE to improve (Fig. 5). We found that the RNN predicted the timeseries data strongly, but not as strongly as the company's internal model. We suspect that moving to a higher T and perhaps h would improve the model's efficacy, however we ran into computational limits for values much greater than the ones chosen here. Just-in-time compilation was applied to the forward and backward passes of the RNN model, and therefore the use of a GPU is advised for larger T and h.

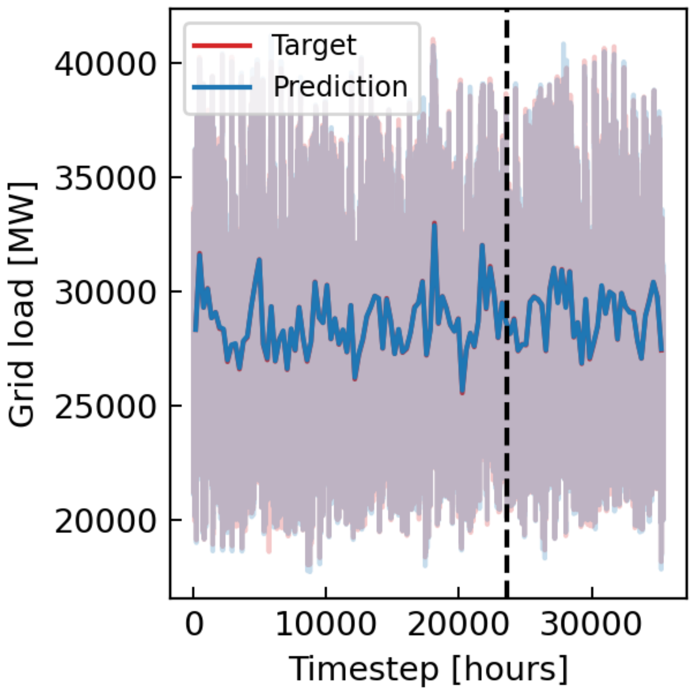

Figure 5: Grid load in megawatts versus timestep in hours. Shown are the target values and the values predicted by the RNN. The raw data is plotted transparently while the opaque data represents the block-averaged data using a block size of 300. The dashed vertical line corresponds to the training and validation split, where validation data is to the right of the line. The RMSE for this comparison is 527.47 compared to the company's RMSE of 403.46.

We then applied our RNN to forecasting solar generation from our dataset. We found that our RNN model did a better job at this forecasting task, and even better than the company's internal model (Fig. 6). From looking at the block-averaged data, we can see one reason why the RNN excels: the data has a much clearer cyclic pattern. Since solar generation is explicitly coupled to the diurnal cycle and the seasons, this stronger cyclic property is expected. Solar generation is also much more strongly related to the diurnal cycle than grid load, and so we suspect our choice of $T = 36$ is better suited to this forecasting task. We can also see that, occasionally, our RNN gives an nonphysical answer.

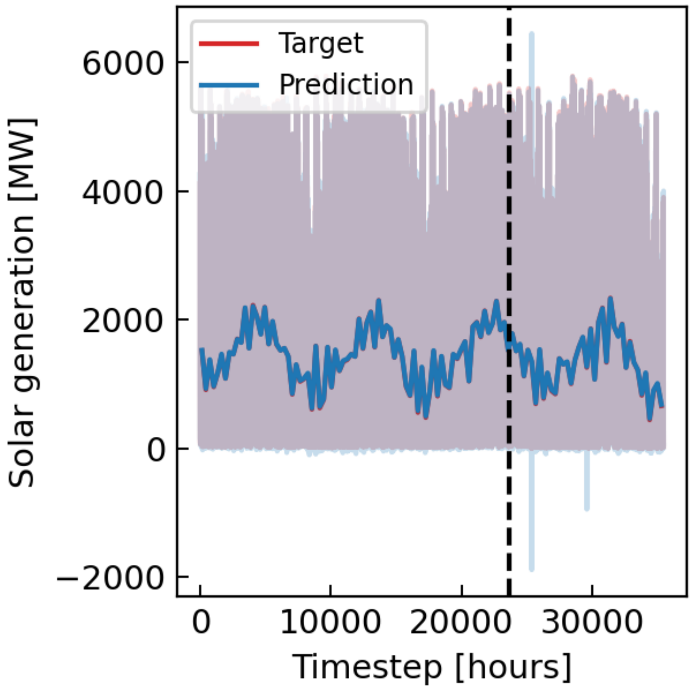

Figure 6: Solar generation in megawatts versus timestep in hours. Shown are the target values and the values predicted by the RNN. The raw data is plotted transparently while the opaque data represents the block-averaged data using a block size of 300. The dashed vertical line corresponds to the training and validation split, where validation data is to the right of the line. The RMSE for this comparison is 166.61 compared to the company's RMSE of 212.33.

Lastly, we tasked our RNN model with predicting wind generation. With regards to cyclic behavior, we see that wind generation (Fig. 7) is less visibly cyclic than the two previous targets. This observation is also seen in the performance of the company's internal model: of the three targets (grid load, solar generation, and wind generation), the company's RMSE for wind generation is the highest. We find this to also be the case with our RNN model (Fig. 7). While increase T would likely help, an ensemble approach combining physical, statistical, and AI models is recommended for predicting wind generation.[9] We explore the use of ensemble models in a coming section, but we note our lack of a physical model.

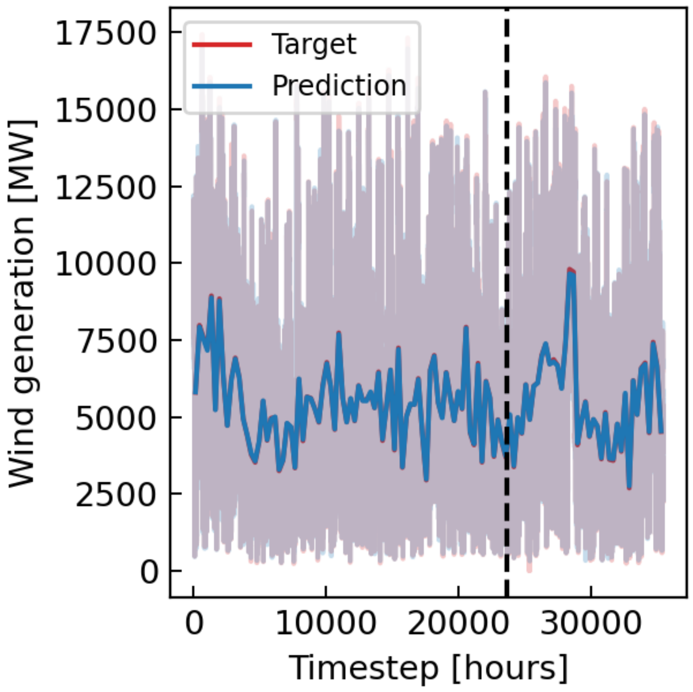

Figure 7: Wind generation in megawatts versus timestep in hours. Shown are the target values and the values predicted by the RNN. The raw data is plotted transparently while the opaque data represents the block-averaged data using a block size of 300. The dashed vertical line corresponds to the training and validation split, where validation data is to the right of the line. The RMSE for this comparison is 619.09 compared to the company's RMSE of 549.55.

Overall, our RNN model performs well and especially so for forecasting solar generation. Aside from increasing T, h, and consequently the model's computational expense, we note that incorporating more layers is a clear step towards a more robust model.[10] These

9. Bernhard Ernst et al., "Predicting the wind," *IEEE power and energy magazine* 5, no. 6 (2007): 78–89.

10. Junhua Mao et al., "Deep captioning with multimodal recurrent neural networks (m-rnn)," *arXiv preprint arXiv:1412.6632*, 2014,

so-called multimodal recurrent neural networks (m-RNN) excel at more complex tasks and could be used to forecast over different timescales simultaneously.

4.4 Feedforward Neural Network

4.4.1 Setup

We implemented a feedforward neural network (FFNN) for regression forecasting. The network consists of two weight matrices and two bias vectors: an input-to-hidden weight matrix $\mathbf{W}^{(1)} \in \mathbb{R}^{d\times h}$ with bias $\mathbf{b}^{(1)} \in \mathbb{R}^h$, and a hidden-to-output weight matrix $\mathbf{W}^{(2)} \in \mathbb{R}^{h\times 1}$ with bias $\mathbf{b}^{(2)} \in \mathbb{R}$. Here, d is the input dimension and h is the hidden dimension.

Given an input vector $\mathbf{x} \in \mathbb{R}^d$, the forward pass proceeds as follows:

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)\top}\mathbf{x} + \mathbf{b}^{(1)} \tag{20}$$

$$\mathbf{a}^{(1)} = \mathrm{ReLU}(\mathbf{z}^{(1)}) \tag{21}$$

$$\hat{y} = \mathbf{W}^{(2)\top}\mathbf{a}^{(1)} + \mathbf{b}^{(2)} \tag{22}$$

where $\mathrm{ReLU}(z) = \max(0, z)$ is applied element-wise.

The network is trained using the mean squared error (MSE) loss:

$$\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2 \tag{23}$$

with optional L_2 regularization:

$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \frac{\lambda}{2}\left(\|\mathbf{W}^{(1)}\|_F^2 + \|\mathbf{W}^{(2)}\|_F^2\right) \tag{24}$$

where λ is the regularization strength.

Gradients are computed via backpropagation. The error at the output layer is:

$$\delta^{(2)} = \hat{y} - y \tag{25}$$

The gradients with respect to the second layer parameters are:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(2)}} = \mathbf{a}^{(1)}\delta^{(2)} \tag{26}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(2)}} = \delta^{(2)} \tag{27}$$

The error is then backpropagated to the hidden layer:

$$\boldsymbol{\delta}^{(1)} = \left(\mathbf{W}^{(2)}\delta^{(2)}\right) \odot \mathbf{1}_{\mathbf{z}^{(1)}>0} \tag{28}$$

where $\odot$ denotes element-wise multiplication and $\mathbf{1}_{\mathbf{z}^{(1)}>0}$ is the ReLU derivative.

The gradients for the first layer parameters are:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(1)}} = \mathbf{x}\boldsymbol{\delta}^{(1)\top} \tag{29}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(1)}} = \boldsymbol{\delta}^{(1)} \tag{30}$$

Finally, parameters are updated using stochastic gradient descent (SGD):

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} \tag{31}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}} \tag{32}$$

where α is the learning rate and $l \in \{1, 2\}$ indexes the layers.

4.4.2 Results & Figures

Here we present the findings of our FFNN model, showcasing its performance across forecasting the three tasks in Load, Solar, and Wind. Within the FFNN it demonstrated a strong ability in regards to its prediction of wind generation, beating out the company's internal model within this category. These results will be showcased and discussed in further detail throughout this section. The FFNN also provides a comprehensive comparison with regard to the RNN, with the FFNN being able to levrage the concurrent features throughout our data.

Firstly looking at the Load performance, we can see that the RMSE is quite high within this model, at 1133.729, which compared to the company's prediction at 403.46, is not the best estimate. This suggests that the FFNN Model may fail to capture many of the fluctuations that happen hourly within the grid load. Within this model though we do see a very low SMAPE of 3.02%, which is great, showing while we may be missing the exact value, our predictions, on average, are within a small percentage of the true load. This leads to me thinking that the FFNN model may be better suited at capturing overall trends at a longer term basis. Now looking at this in with the RNN in mind, we can see that neither model was able to beat the company's prediction, but the RNN was able to offer a much closer value.

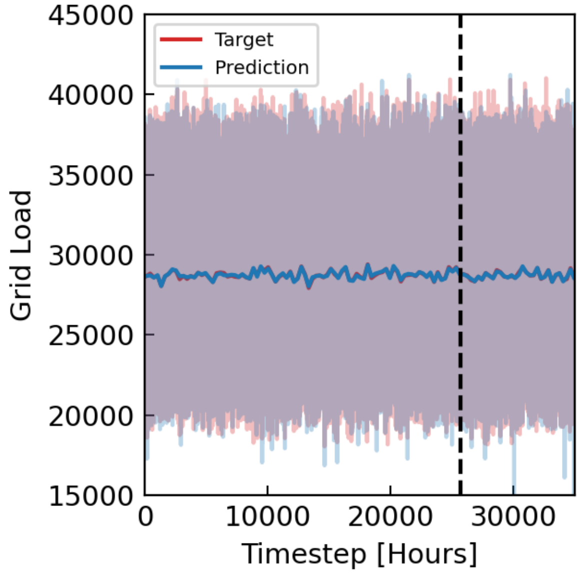

Figure 8: Load in megawatts versus timestep in hours. Shown are the target values and the values predicted by the FFNN. The raw data is plotted transparently while the opaque data represents the block-averaged data using a block size of 300. The dashed vertical line corresponds to the training and validation split, where validation data is to the right of the line. RMSE on the training set is 1063.600 and on the validation set is 1133.729. SMAPE on the training set is 2.93% and on the validation set is 3.02%

Now moving onto the solar model we also see that our RMSE is worse then what the company is providing, with ours at 603.267 and theirs being 212.33. With this model though the thing that is really intresting is the SMAPE value we see, 82.58%, which showcases that the FFNN struggles with the structure of solar data, which while I am not sure of why this is exactly happening, one thought I had was the potential for features, that the FFNN is not using properly. Within the solar model as well is where we see that the RNN is actually able to beat the company's prediction, showing that the FFNN might not be best designed for application with time-series data.

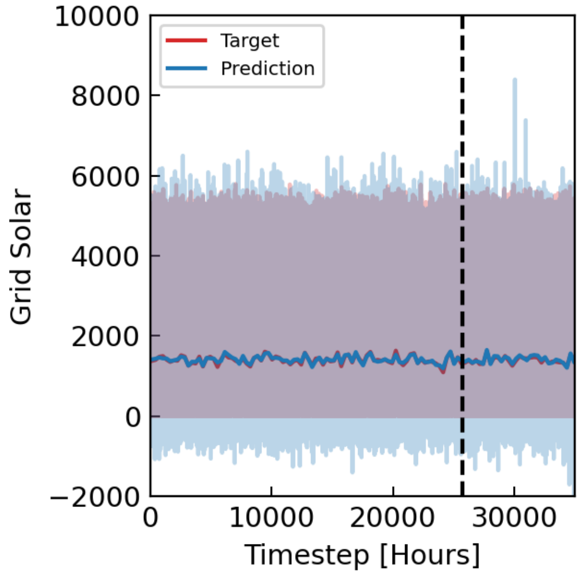

Figure 9: Solar generation in megawatts versus timestep in hours. Shown are the target values and the values predicted by the FFNN. The raw data is plotted transparently while the opaque data represents the block-averaged data using a block size of 300. The dashed vertical line corresponds to the training and validation split, where validation data is to the right of the line. RMSE on the training set is 584.659 and on the validation set is 603.267. SMAPE on the training set is 81.50% and on the validation set is 82.58%.

Looking at the wind model, by far this is the most promising and exciting of the three that were completed within the FFNN, with the fact that we we're able to beat the company's performance with this model. We predicted a value of 433.442, where as the company was at 549.55. Also within this model we see a relatively low SMAPE value of, 7.33%, which this really highlights that the FFNN might be well-suited for wind generation predictions. With wind being highly dependent on location, the fact that the FFNN has the ability to use the full feature set is likely partly attributed to this success. With regard to a comparison of the RNN, that model was not able to beat the company's prediction, which further highlights the impressiveness of the FFNN that it could beat both the RNN and the Company

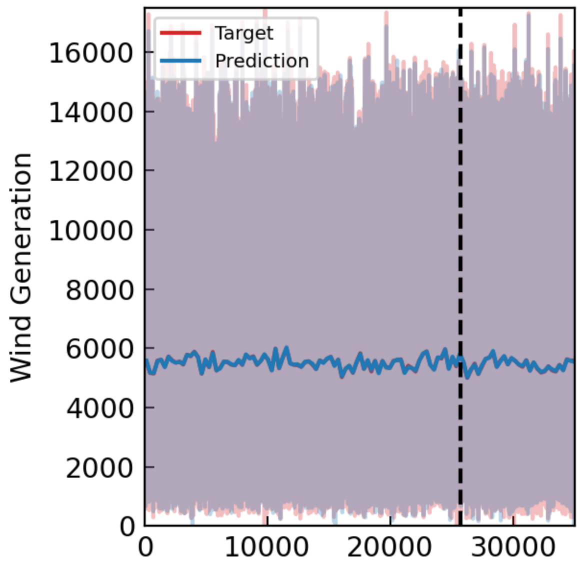

Figure 10: Solar generation in megawatts versus timestep in hours. Shown are the target values and the values predicted by the FFNN. The raw data is plotted transparently while the opaque data represents the block-averaged data using a block size of 300. The dashed vertical line corresponds to the training and validation split, where validation data is to the right of the line. RMSE on the training set is 383.261 and on the validation set is 433.442. SMAPE on the training set is 6.92% and on the validation set is 7.33%

Overall, the FFNN proved to be a powerful tool for forecasting, particularly in cases where the underlying data is influenced by a diverse set of factors. The model's impressive performance on wind generation, where it outperformed both the RNN and the company's internal model, is a testament to this approach. This suggests that a hybrid model, perhaps one that uses the FFNN for feature extraction and an RNN for sequential pattern recognition, could yield even more robust results. Future efforts could focus on integrating these two architectures to create a comprehensive forecasting solution.

4.5 Ensemble Model

4.5.1 Setup

- **Objective:** Combine two complementary forecasters—Linear Regression (LR) and a Recurrent Neural Network (RNN)—to obtain predictions that are more stable and accurate than either model alone.[11]
- **Targets and units:** Three independent series are modeled—LOAD, SOLAR, and WIND—each measured in MW and evaluated separately.
- **Temporal handling:** Data are kept in chronological order (no shuffling). Timestamps are strictly increasing and hourly; duplicates are dropped and short gaps forward-filled to avoid bias. All modeling respects time order to prevent leakage.
- **Data alignment:** For each timestamp t, ground-truth value y_t is aligned with per-model predictions $\{\hat{y}_t^{(m)}\}$ from LR and RNN. Rows with missing predictions are excluded.
- **Feature preparation (LR):** Lag-based features, calendar terms (hour-of-day, day-of-week, holidays), and optional interactions. Categorical features are one-hot encoded; numeric features standardized on training split.
- **Sequence preparation (RNN):** RNN consumes a fixed lookback window L and mirrored exogenous features. Inputs normalized on training split. Hidden size and dropout kept lightweight and reproducible.

11. John M Bates and Clive WJ Granger, "The combination of forecasts," *Journal of the operational research society* 20, no. 4 (1969): 451–468.

- **Train/validation/test splits:** Chronological split. LR and RNN trained on Train; hyperparameters and early stopping rely only on Validation. Test set held out for final evaluation.
- **Primary metric:** RMSE on validation and test splits; MAE monitored informally.
- **Post-processing:** Enforce non-negativity by clipping; no manual edits of spike artifacts.
- **Reproducibility:** Fixed random seeds; transforms fit on Train only; ensemble logic runs from a single script for deterministic outputs.

4.5.2 Method

Two combination rules are applied:

1. **Uniform average:**
$$\hat{y}_t^{\text{uni}} = \frac{1}{M}\sum_{m=1}^{M} \hat{y}_t^{(m)}$$
2. **Error-weighted average:** Nonnegative weights sum to one,
$$w_m \propto \frac{1}{\text{RMSE}_m + \varepsilon}, \quad \hat{y}_t^{\text{ens}} = \sum_m w_m \cdot \hat{y}_t^{(m)}$$
where $M = 2$ (LR and RNN) and ε prevents division by zero.

4.5.3 Selection Rule

Per series (LOAD, SOLAR, WIND), both combinations are computed on the validation window. The lower-RMSE scheme is selected, frozen, and evaluated once on the held-out Test window.

4.5.4 Results

- **LOAD:** Uniform average selected; both LR and RNN comparable, averaging reduces variance.
- **SOLAR:** Error-weighted ensemble selected; RNN tracks diurnal peaks better.
- **WIND:** Uniform average preferred; averaging tempers intermittency.

Target	Validation RMSE (Uniform)	Validation RMSE (Ensemble)	Selected Scheme	Test RMSE (Selected)
LOAD	2,364.04	4,516.57	uniform	2,313.46
SOLAR	1,437.93	954.63	ensemble	746.74
WIND	1,811.96	2,395.65	uniform	2,269.11

Table 6: Validation and Test RMSE for ensemble selection per target.

4.5.5 Takeaways

A two-scheme ensemble with a validation-time selector is easy to reproduce, leakage-free, and adapts per target. It stabilizes LOAD, tightens peak tracking for SOLAR, and improves WIND forecasts over individual LR or RNN predictions. Future extensions include stacked meta-learners, constrained blends (e.g., nonnegative least squares with temporal smoothness), and physics-informed features to capture extreme wind ramps.

4.5.6 Figures

- Ensemble comparison on held-out test period.

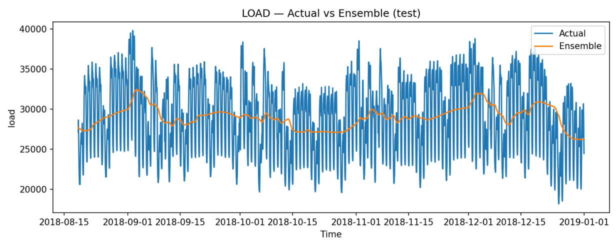

Figure 11: Error-weighted ensemble vs. actual (test)

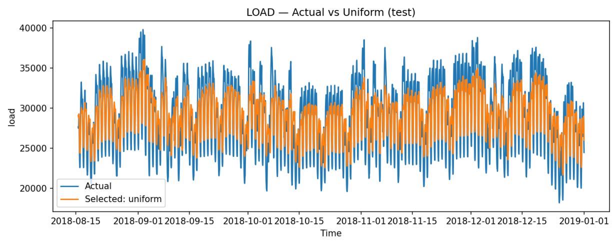

Figure 12: Uniform average vs. actual (test). Selected: uniform.

- Ensemble comparison on held-out test period.

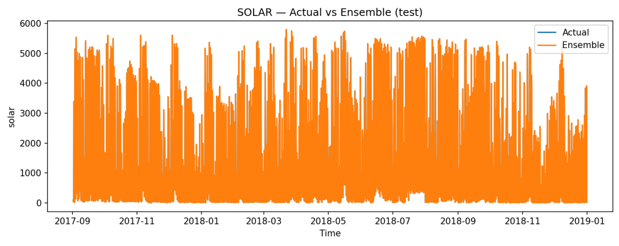

Figure 13: Error-weighted ensemble vs. actual (test).

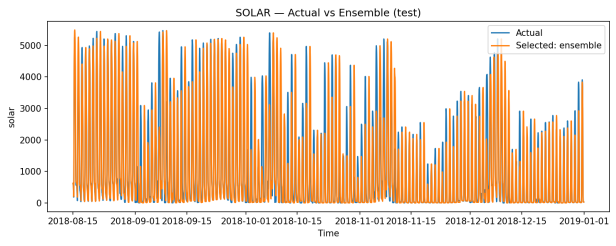

Figure 14: Selected method overlay (test). Selected: ensemble (error-weighted)

- Ensemble comparison on held-out test period.

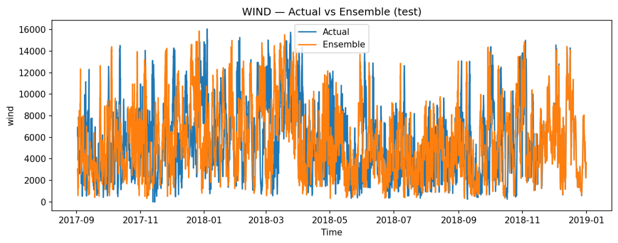

Figure 15: Error-weighted ensemble vs. actual (test).

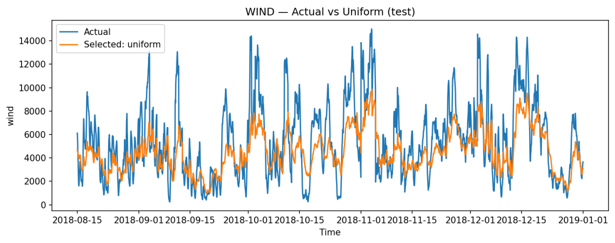

Figure 16: Uniform average vs. actual (test). Selected: uniform.

5 Conclusions and Future Work

5.1 Conclusions

This study evaluated six forecasting approaches for Spain's **LOAD**, **WIND**, and **SOLAR** series: a Baseline yardstick, Linear Regression (LR), Feedforward Neural Network (FFNN), ARIMA, Recurrent Neural Network (RNN), and an LR+RNN ensemble. No single model dominated across targets and horizons.

- **Baseline:** Offered a strong reference and was occasionally competitive on steadier horizons, especially for LOAD.

- **Linear Regression (LR):** Reliably captured trend and seasonality and tended to generalize better at longer horizons but struggled with rapid short-term fluctuations.
- **Feedforward Neural Network (FFNN):** Leveraged non-linear feature interactions and often exceeded LR when rich features were available; without temporal memory it was sensitive to regime shifts and overfitting.
- **ARIMA:** Most effective on smoother, short-horizon LOAD forecasts; performance declined on highly intermittent WIND and, as a univariate model, it could not exploit exogenous drivers.
- **Recurrent Neural Network (RNN):** Performed best on SOLAR, where strong diurnal/seasonal cycles align with recurrent modeling; gains were smaller on less-cyclic WIND, and at times it trailed the baseline on LOAD/WIND.
- **LR+RNN ensemble:** Provided the most robust practical gains: a leakage-free, per-series selector chose between uniform and error-weighted averaging, tempering variance on LOAD/WIND (uniform) and improving SOLAR peak tracking (error-weighted).

Overall, results indicate that complementary inductive biases matter more than any single.[12] Simple, validated blends of stable (LR) and expressive (RNN) forecasters may yield best accuracy–robustness trade-off and the most usable operational signals with proper tuning.

12. Tao Hong et al., *Probabilistic energy forecasting: Global energy forecasting competition 2014 and beyond*, 3, 2016, 896–913.

5.2 Future Work

- **Advanced ensembling:** Stacked/meta-learners and constrained blends (e.g., nonnegative least squares with temporal smoothness).
- **Richer inputs:** Region-level weather, calendar/holiday effects, irradiance/wind-profile features; consider SARIMAX/Prophet/VECM for explicit seasonality/multivariate structure.
- **Neural upgrades:** Longer sequences, larger hidden sizes, GRU/LSTM or TCN/N-BEATS variants, scheduled learning rates, and GPU-enabled training.
- **Regularized linear models:** Ridge/Lasso/Elastic Net with renewed feature selection to balance bias–variance.
- **Probabilistic forecasting:** Quantiles/intervals (pinball loss, conformal methods) to support grid risk decisions.
- **Evaluation realism:** Rolling-origin cross-validation, multi-resolution metrics (hourly and daily aggregates), and per-region diagnostics.
- **Physical and operational constraints:** Enforce non-negativity architecturally or via calibrated post-processing; add bias/variance calibration layers.
- **Productionization:** Reproducible pipelines (versioned data/code, fixed seeds), drift detection, backfills for missing hours, and monitoring dashboards.

5.3 Threats to Validity

Findings reflect the feature set and compute budget used (e.g., univariate ARIMA; limited sequence length/width for RNN/FFNN). Future studies should revisit conclusions under richer exogenous data and larger model capacities.

Bibliography

Bates, John M, and Clive WJ Granger. “The combination of forecasts.” *Journal of the operational research society* 20, no. 4 (1969): 451–468.

Bora, N. *Understanding ARIMA models for machine learning.* Capital One, 2021. https://www.capitalone.com/tech/machine-learning/understanding-arima-models/.

Brailey, B. *Transforming grid operations with accurate short-term energy predictions.* DNV, 2024. https://www.dnv.com/article/transforming-grid-operations-with-accurate-short-term-energy-predictions/.

Clemen, Robert T. “Combining forecasts: A review and annotated bibliography.” *International journal of forecasting* 5, no. 4 (1989): 559–583.

Ernst, Bernhard, Brett Oakleaf, Mark L Ahlstrom, Matthias Lange, Corinna Moehrlen, Bernhard Lange, Ulrich Focken, and Kurt Rohrig. “Predicting the wind.” *IEEE power and energy magazine* 5, no. 6 (2007): 78–89.

Her, O. Y., M. S. A. Mahmud, M. S. Z. Abidin, R. Ayop, and S. Buyamin. “Artificial neural network based short term electrical load forecasting.” *International Journal of Power Electronics and Drive Systems / International Journal of Electrical and Computer Engineering* 13, no. 1 (2022): 586. https://doi.org/10.11591/ijpeds.v13.i1.pp586-593.

Hong, Tao, Pierre Pinson, Shu Fan, Hamidreza Zareipour, Alberto Troccoli, and Rob J Hyndman. *Probabilistic energy forecasting: Global energy forecasting competition 2014 and beyond*, 3, 2016.

Mao, Junhua, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, and Alan Yuille. “Deep captioning with multimodal recurrent neural networks (m-rnn).” *arXiv preprint arXiv:1412.6632*, 2014.

Misiurek, K., T. Olkuski, and J. Zyśk. “Review of Methods and Models for Forecasting Electricity Consumption.” *Energies* 18, no. 15 (2025): 4032. https://doi.org/10.3390/en18154032.

Schmidt, Robin M. “Recurrent neural networks (rnns): A gentle introduction and overview.” *arXiv preprint arXiv:1912.05911*, 2019.

Tarmanini, C., N. Sarma, C. Gezegin, and O. Ozgonenel. “Short term load forecasting based on ARIMA and ANN approaches.” *Energy Reports* 9 (2023): 550–557. https://doi.org/10.1016/j.egyr.2023.01.060.